POLITECHNIKA LUBELSKA
WYDZIAŁ INŻYNIERII ŚRODOWISKA
INSTYTUT INŻYNIERII ODNAWIALNYCH ŹRÓDEŁ ENERGII

INSTRUKCJE WYKONANIA ZADAŃ NA ZAJĘCIACH LABORATORYJNYCH

INFORMATYCZNE PODSTAWY PROJEKTOWANIA

Sławomir Gułkowski

Politechnika Lubelska
Lublin 2014

Publikacja dystrybuowana bezpłatnie. Nakład 100 egzemplarzy.
Publikacja wydana w ramach projektu „Fotowolt – fizyka techniczna dla ekoinżyniera" współfinansowanego ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego,
nr umowy UDA – POKL.04.01.02 – 00 –181/12 – 01.

KAPITAŁ LUDZKI
NARODOWA STRATEGIA SPÓJNOŚCI

UNIA EUROPEJSKA
EUROPEJSKI
FUNDUSZ SPOŁECZNY

Publikacja współfinansowana ze środków Unii Europejskiej w ramach Europejskiego Funduszu Społecznego

Spis treści

Przedmowa

Umiejętność posługiwania się szeroko pojętymi technologiami informatycznymi w pracy badawczej oraz inżynierskiej stanowi dziś jedną z najważniejszych umiejętności jakie nabywają studenci w trakcie studiów. Bardzo istotne jest już w pierwszych latach studiów wykształcenie umiejętności tworzenia i analizy algorytmów, czyli sposobu postępowania prowadzącego do rozwiązania określonego problemu projektowego i obliczeniowego. Implementacja opracowanego algorytmu wymaga znajomości programów komputerowych, które umożliwiają przygotowanie własnych rozwiązań w postaci formuł, skryptów czy kodu utworzonego w wybranym języku programowania. Równie ważna jest graficzna prezentacja wyników obliczeń i ich ocena statystyczna. Poprawna interpretacja wyników badań jest istotną częścią większości, jeśli nie wszystkich, zajęć laboratoryjnych jakie student musi zaliczyć w trakcie swojego studiowania. Wykorzystanie do tego celu mniej lub bardziej zaawansowanego środowiska obliczeniowego umożliwia otrzymanie efektownych analiz, które będą stanowić wartość dodaną opracowania składanego do oceny.

Niniejszy skrypt stanowi uzupełnienie zajęć z przedmiotu Informatyczne Podstawy Projektowania prowadzonych w pierwszym semestrze studiów inżynierskich. Skrypt podzielony został na trzy części:

- Obliczenia w arkuszu kalkulacyjnym
- Podstawy programowania
- Wprowadzenie do środowiska Matlab

Pierwsza część to wprowadzenie do znanego i popularnego arkusza kalkulacyjnego firmy Microsoft, Excela. Na początku znajdują się proste zadania wymagające tworzenia formuł. W kolejnych ćwiczeniach znalazły się zagadnienia dotyczące zastosowania warunku oraz iteracji. Ostatnie ćwiczenie dotyczy prezentacji wyników w postaci graficznej oraz formatowania wykresu. W części drugiej skryptu przedstawiono podstawowe zasady tworzenia i kompilacji własnych programów komputerowych w języku C++. Część trzecia, to wprowadzenie do zaawansowanego środowiska naukowo-obliczeniowego, jakim jest Matlab. Każda z trzech części skryptu składa się z pięciu instrukcji, w których znajdują się teoretyczne podstawy omawianego zagadnienia, przykłady arkuszy, kodów a także zadania do samodzielnego wykonania.

Należy podkreślić, że zaprezentowany skrypt nie jest podręcznikiem opisującym wyczerpująco arkusz kalkulacyjny, język C/C++ czy pakiet Matlab. Są to bardzo zaawansowane narzędzia, których nawet krótki opis możliwości zdecydowanie wykracza poza ramy objętościowe niniejszego skryptu. Ideą, jaką kierowałem się wykonując tę pracę było przedstawienie sposobu rozwiązania wybranych problemów obliczeniowych w omawianych programach z jednoczesnym naciskiem na pozyskanie przez studenta umiejętności posługiwania się wybranym narzędziem informatycznym w stopniu umożliwiającym samodzielne wykonanie powierzonych zadań.

Część 1

Obliczenia w Arkuszu Kalkulacyjnym

1. Wprowadzenie do obliczeń w Arkuszu Kalkulacyjnym

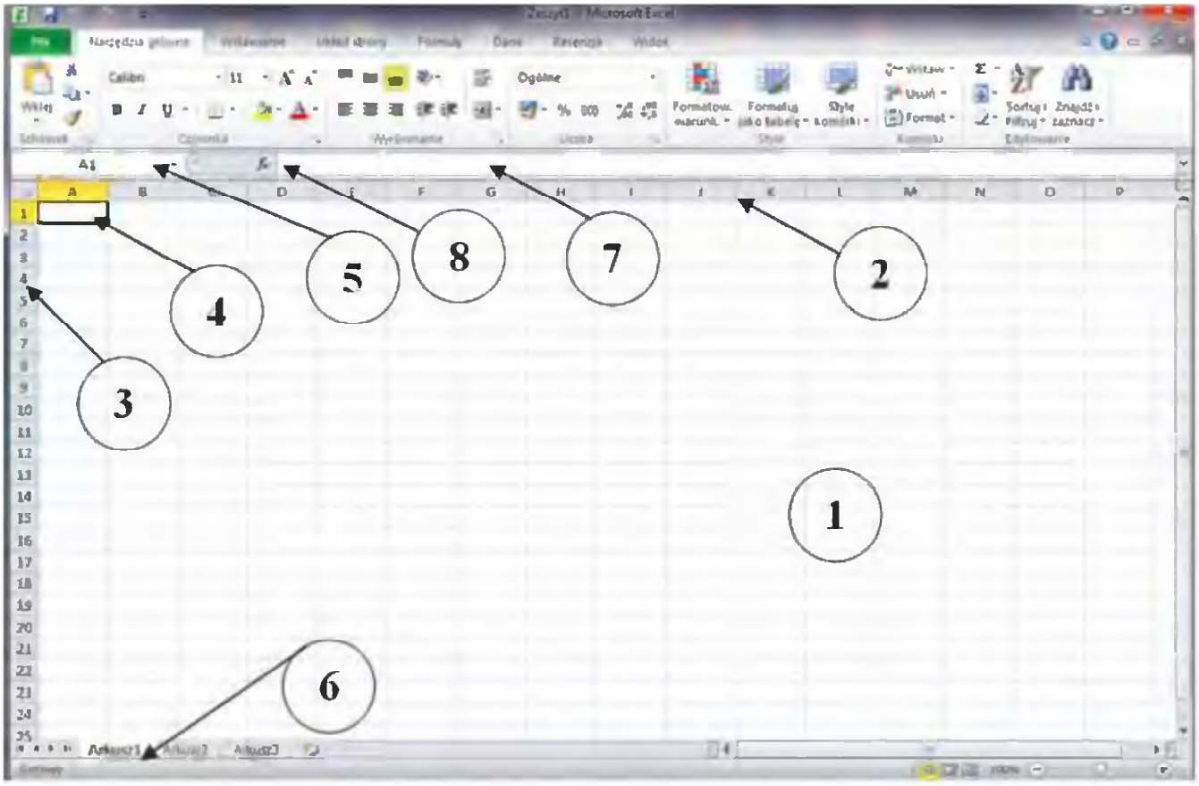
Wymagania sprzętowe i oprogramowanie:

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Arkusz kalkulacyjny MS Excel 2007 lub nowszy

Wstęp teoretyczny

Aplikacja Microsoft Excel stanowi jeden ze składników rozbudowanego pakietu Office firmy Microsoft. Jej głównym przeznaczeniem jest wykonywanie obliczeń oraz ich wizualizacja w postaci wykresów, co nie oznacza, że jest to jedyne zastosowanie programu. Umożliwia on także wykonanie prostych operacji bazodanowych. Implementacja języka VBA (*Visual Basic for Applications)* w pakiecie Microsoft Office, czyli języka programowania wysokiego poziomu dodatkowo rozszerza możliwości Excel'a o tworzenie własnych rozwiązań algorytmicznych w postaci kodu źródłowego. Właściwie zaprojektowany program, współpracujący z arkuszem kalkulacyjnym zwiększa jego efektywność i znacznie ułatwia przetwarzanie i analizę danych. Szeroki wachlarz zastosowań oraz stosunkowo tani dostęp do licencji (np. licencje do zastosowań domowych, licencje w ramach MSDNAA dla uczelni) sprawiają, że program Microsoft Excel jest powszechnie stosowanym pakietem obliczeniowym zarówno w zastosowaniach biurowych czy domowych jak i w obliczeniach naukowych i inżynierskich.

Rysunek 1.1 przedstawia okno główne programu Microsoft Excel 2010 wraz z otwartym, pustym skoroszytem, którego domyślna nazwa to Arkusz1.



Rys. 1.1. Główne okno aplikacji Microsoft Excel 2010 z otwartym arkuszem Arkusz1

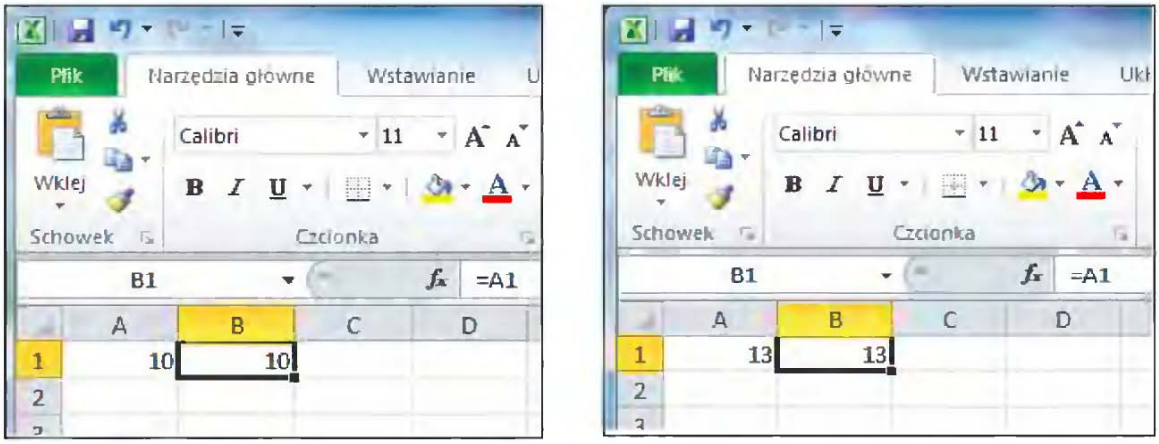
Poniżej opisane zostały najważniejsze elementy arkusza kalkulacyjnego oznaczone na rysunku 1.1 cyframi 1 8:

1 – komórki arkusza kalkulacyjnego, do których wprowadzane są dane lub wyświetlane są wyniki obliczeń,
2 –kolumny arkusza oznaczone kolejnymi literami alfabetu: A, B, C, D, …
3 – wiersze arkusza oznaczone kolejnymi liczbami całkowitymi 1, 2, 3, …
4 – aktywna komórka,
5 – pole nazwy dla aktywnej komórki,
6 – aktywny arkusz,
7 – pasek formuły do wprowadzania wyrażeń arytmetycznych i logicznych,
8 – ikona funkcji otwierająca dodatkowe okno z funkcjami dostępnymi w programie.

Posługiwanie się podstawowymi funkcjami arkusza kalkulacyjnego stanie się intuicyjne, jeżeli uświadomimy sobie w jaki sposób arkusz przechowuje dane. Do aktywnej komórki wprowadzamy w zależności od potrzeby liczbę, tekst lub formułę. Wprowadzona dana, załóżmy, że jest to liczba, będzie rozpoznawana w pozostałych komórkach arkusza poprzez adres określony literą kolumny, w której znajduje się ta liczba oraz numerem wiersza. Dla przykładu, jeżeli wprowadzimy do komórki o adresie A1 liczbę 10, a następnie przejdziemy do sąsiedniej komórki (o adresie B1) i wpiszemy prostą formułę:

=A1

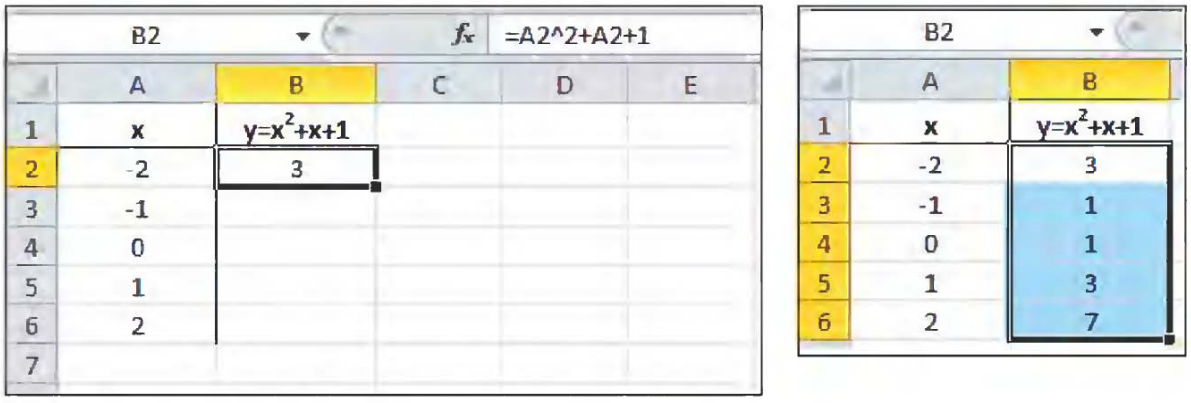
otrzymamy w wyniku w komórce B1 liczbę 10. Zmieniając liczbę w komórce A1, zauważymy, że zmieni się także liczba w komórce B1 (rysunek 1.2).



Rys. 1.2. Związek liczby z adresem komórki. Odwołując się do określonej danej znajdującej się w komórce należy posłużyć się adresem komórki

Przy projektowaniu arkusza kalkulacyjnego, wskazane jest stosowanie odwołania adresowego we wprowadzanych formułach. Dzięki temu projekt jest bardziej elastyczny w przypadku zmiany wartości danych w komórkach. Należy jednak pamiętać, że kopiowanie formuły prowadzi do zmiany adresów komórek w zależności od kierunku kopiowania:

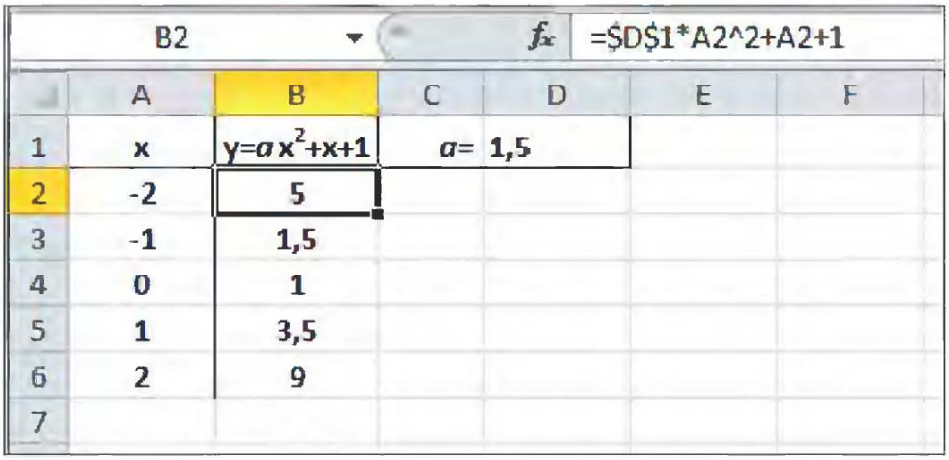
- poziomo – zmiana litery w adresie komórki
- pionowo – zmiana liczby w adresie komórki



Rys. 1.3. Kopiowanie formuły z komórki B2 do komórek B3–B6

Rysunek 1.3 przedstawia prosty przykład zastosowania funkcji kopiowania dla przypadku wyrażenia zawierającego adres komórki. W celu otrzymania wartości y dla przykładowej funkcji kwadratowej, należy wprowadzić w komórce B2 wzór funkcji (wprowadzone wyrażenie automatycznie pojawi się w pasku formuły). Następnie kliknąć na kwadrat w prawym dolnym rogu ramki zaznaczenia aktywnej komórki i przeciągnąć kursor do komórki B6 jednocześnie przytrzymując lewy klawisz myszy. Znajdujące się w komórce B2 wyrażenie zostanie przekopiowane do zaznaczonych komórek a adres komórki A2 w tym wyrażeniu zostanie odpowiednio zmieniony na A3, A4 itd. W przypadku kopiowania w kolumnach zmianie ulegnie litera w adresie komórki.

W sytuacji gdy chcemy przy kopiowaniu odwoływać się do pojedynczej, konkretnej komórki arkusza należy zastosować tzw. adresowanie bezwzględne tj. przed literą oraz liczbą w adresie komórki wstawić znak $, np. $A$1 (rysunek 1.4).



Rys. 1.4. Przykład adresowania bezwzględnego

Należy zauważyć, że wartość parametru oznaczonego literą ***a*** znajduje się w komórce D1. Z tego powodu umieszczenie znaku $ przed literą i liczbą adresu powoduje zablokowanie adresu przy kopiowaniu formuły do komórek. Inne formy blokowania adresu:

- znak $ tylko przed literą (np. $B1) oznacza zablokowanie tylko tej litery przy kopiowaniu,
- znak $ tylko przed liczbą (np. B$1) oznacza zablokowanie tylko liczby.

W tabeli 1.1 wymienione zostały operatory, które mogą być stosowane w formułach.

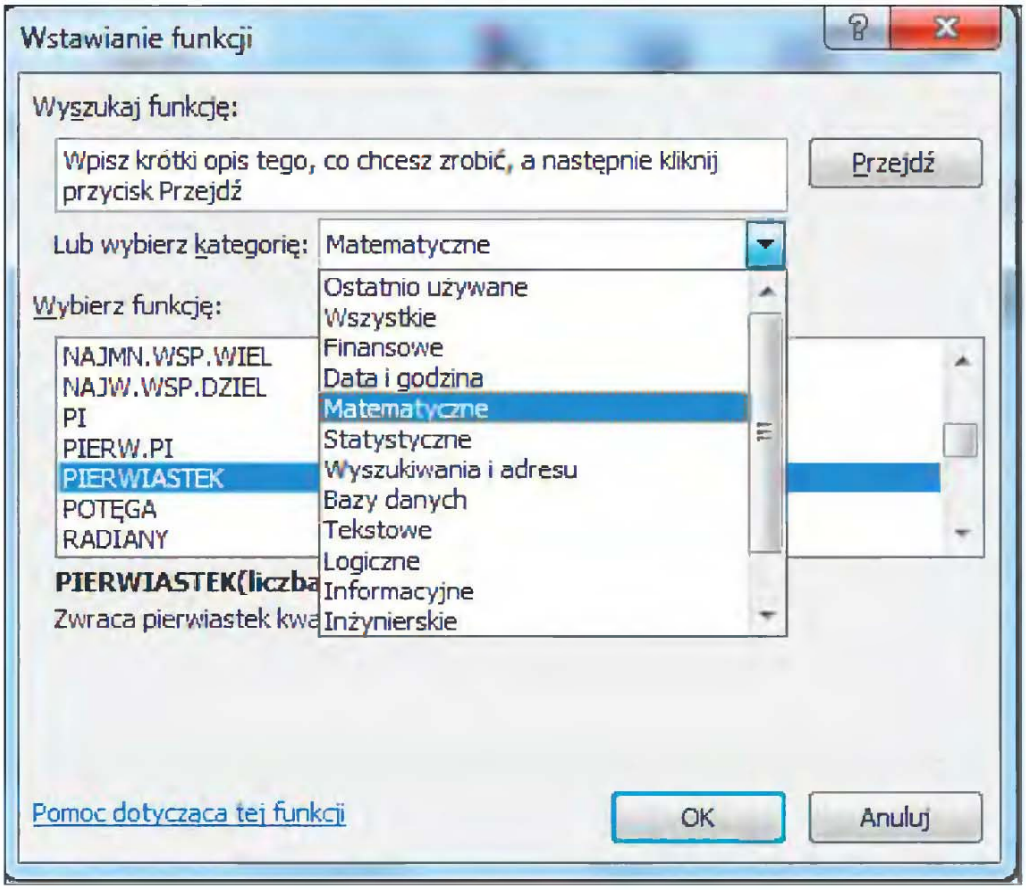
Tabela 1.1. Operatory arytmetyczne stosowane w arkuszu kalkulacyjnym

Operator	Działanie
+	dodawanie
-	odejmowanie
*	mnożenie
/	dzielenie
^	potęgowanie

Podobnie jak w znanej ze szkoły arytmetyce tak i tu mamy do czynienia z priorytetem operatorów, to znaczy kolejnością wykonywania działań. Pod tym względem najważniejszy jest operator potęgowania. Później na równi operatory mnożenia i dzielenia, na końcu zaś operatory dodawania i odejmowania. Należy o tym pamiętać przy konstruowaniu formuł. Jeżeli chcemy wpłynąć na kolejność wykonywania działań, używamy nawiasów. Wyrażenia w nawiasie obliczane są w pierwszej kolejności. Przykład wyrażenia z nawiasem:

=A1*(A2+A3)

Przedstawione w tabeli 1.1 operatory należą do najbardziej podstawowych. W arkuszu kalkulacyjnym Excel wbudowane są funkcje, które również wykonują określone działania, np. funkcja pierwiastek. Funkcje te odnaleźć można wybierając myszką ikonę funkcji (rysunek 1.1). Otwiera się wówczas dodatkowe okno (rysunek 1.5), w którym znajdują się nazwy funkcji wbudowanych oraz krótki opis ich działania. Wywołanie danej funkcji następuje poprzez podanie jej nazwy z odpowiednimi argumentami.



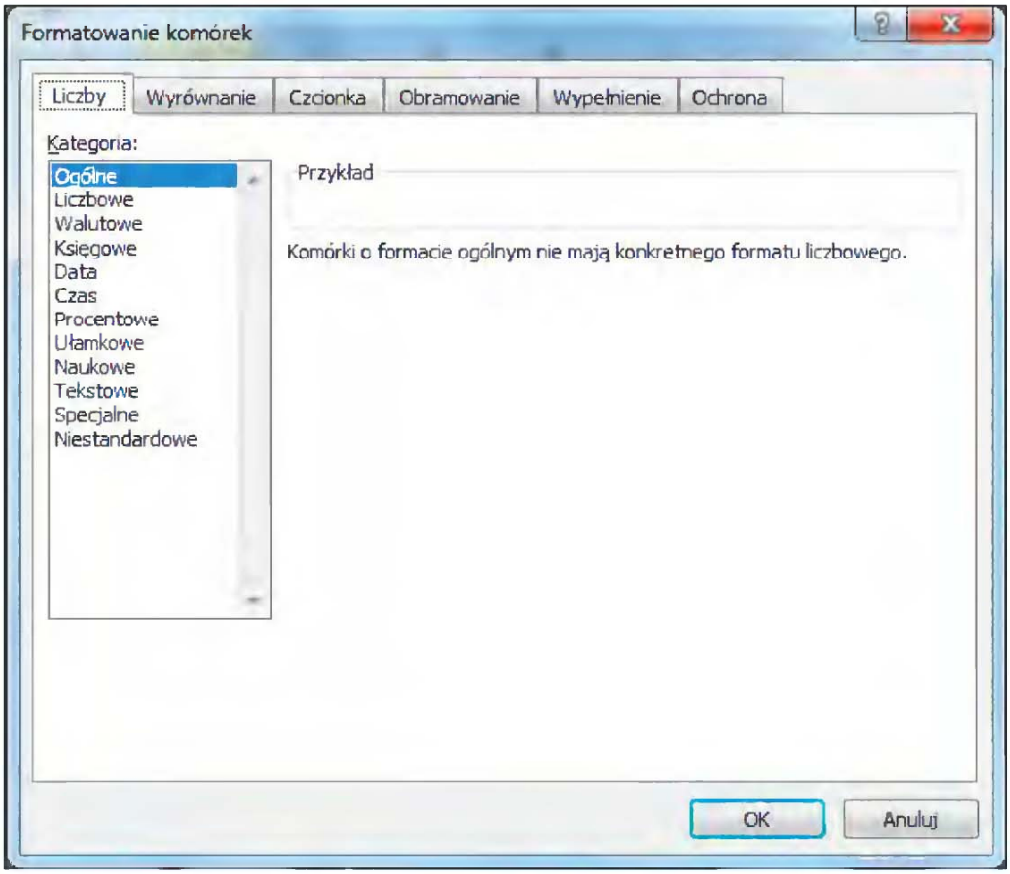
Rys. 1.5. Okno z nazwami funkcji wbudowanych

Wybrane funkcje oraz opis ich działania przedstawia tabela 1.2.

Tabela 1.2. Wybrane funkcje matematyczne Excel'a

Funkcja	Opis działania
PIERWIASTEK (LICZBA)	zwraca pierwiastek kwadratowy liczby
POTĘGA (LICZBA, WYKŁADNIK)	zwraca liczbę podniesioną do potęgi
SUMA(ZAKRES)	dodaje liczby w zakresie komórek; zakres komórek określa adres komórki początkowej i końcowej oddzielony dwukropkiem. zapis A1:A5 – oznacza zakres komórek od A1 do A5
ŚREDNIA(ZAKRES)	zwraca średnią arytmetyczną zakresu liczb
PI()	zwraca wartość liczby pi z dokładnością 15 cyfr po przecinku
EXP(X)	funkcja e^x
LN(LICZBA)	zwraca logarytm naturalny podanej liczby
SIN(LICZBA)	zwraca sinus kąta podanego w radianach
COS(LICZBA)	zwraca cosinus kąta podanego w radianach
TAN(LICZBA)	zwraca tangens kąta podanego w radianach
RADIANY(KĄT)	konwertuje stopnie na radiany
STOPNIE(KĄT)	konwertuje radiany na stopnie

Rysunek 1.6 przedstawia okno formatowania komórek dostępne z menu podręcznego arkusza (prawy klawisz myszy -> Formatuj komórki)



Rys. 1.6. Okno Formatowanie komórek

W zakładce liczby ustawiany jest format danych przechowywanych w komórce. Poza określeniem czy dane w komórce będą liczbowe czy tekstowe, możemy ustawić ilość wyświetlanych miejsc po przecinku dla liczb rzeczywistych lub określić czy będą wyświetlanie w tzw. notacji naukowej (inżynierskiej). W tabeli 1.3 przedstawiono przykłady zapisu liczb w różnych formatach. W kolejnych zakładkach dostępne są opcje formatowania takie jak wyrównanie danych w komórce, kierunek tekstu czy indeksy górny i dolny.

Tabela 1.3. Przykłady zapisu liczb w różnych formatach

Zapis liczby	Opis
3,14	format liczbowy z dwoma miejscami dziesiętnymi
3,00E-03	format naukowy z dwoma miejscami dziesiętnymi (tu: 0,003)
2013-05-10	data (jedna z możliwości)
15%	procent

Zadania do wykonania

Zadanie 1.

Obliczyć wartość wyrażenia dla zadanych parametrów:

a) $V = \pi l r^2$, *dlal=0.1cm, r=0.05cm*

b) $d = \frac{m}{\frac{1}{2}\pi r^2 s + \frac{1}{6}\pi s^3}$, *dlar*=1.3678 cm, *s*=0.6359cm, *m*=9.23638g

W celu wykonania powyższych przykładów należy wprowadzić do komórek dane będące znanymi parametrami, a następnie wprowadzić wzory funkcji za pomocą formuł. Należy pamiętać o kolejności działania operatorów oraz nawiasach. Liczbę π wpisujemy jako funkcję. Wzór arkusza dla przykładów przedstawiono na rysunku 1.7.

Zadanie 2.

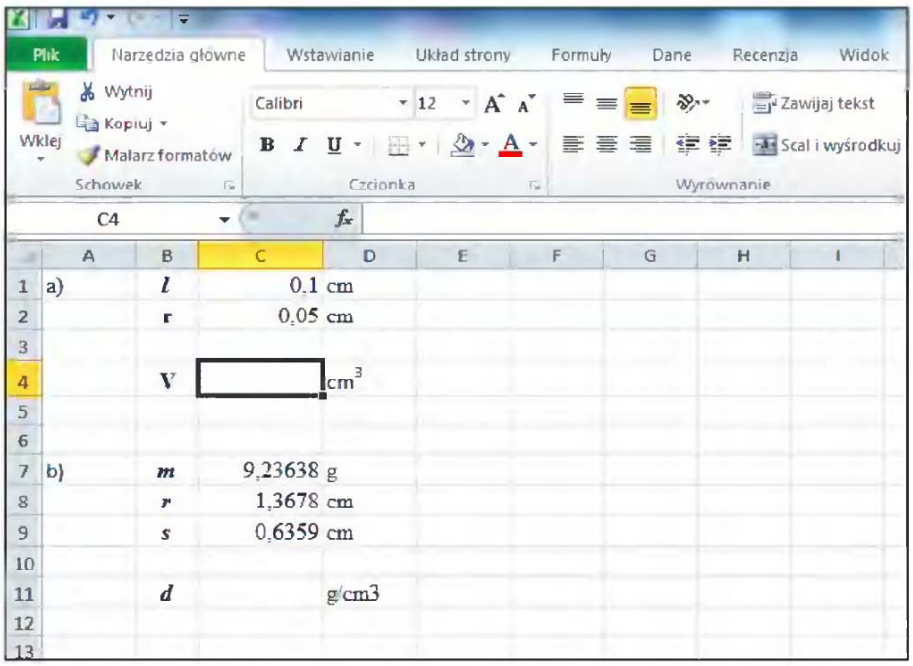
Oblicz wartości funkcji :

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

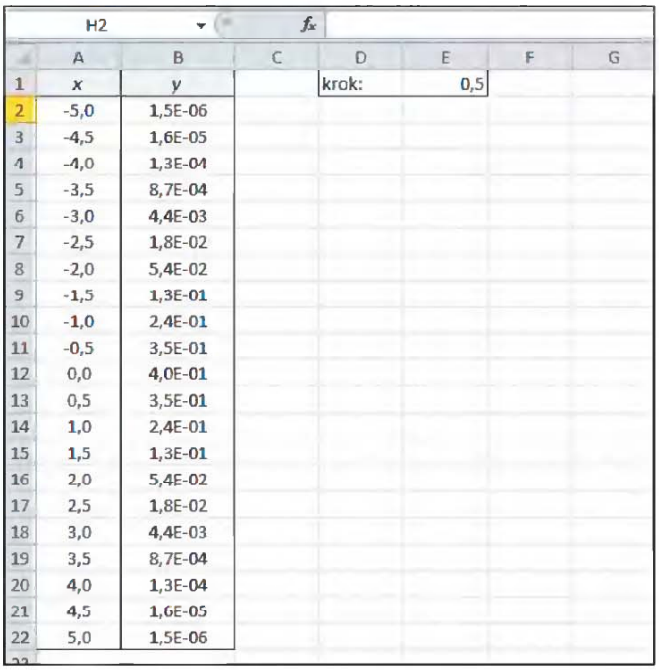
dla $x \in < -5; 5 >$ z krokiem 0.5.

Liczby w kolumnie *x* wyświetlić z dokładnością do jednego miejsca po przecinku. Format liczb w kolumnie *y* ustawić na naukowy z jednym miejscem dziesiętnym po przecinku.

W celu wykonania powyższego zadania należy utworzyć tabelę składającą się z dwóch kolumn. Pierwsza kolumna zawiera wartości *x* , zaś druga obliczone wg wzoru wartości *y*. Formułę w kolumnie *y* należy wprowadzić do pierwszej komórki, a następnie przy użyciu funkcji kopiuj – przekopiować na pozostałe komórki. Wzór arkusza dla tego przykładu przedstawia rysunek 1.8.



Rys. 1.7. Wzór arkusza do zadania 1



x	y
-5,0	1,5E-06
-4,5	1,6E-05
-4,0	1,3E-04
-3,5	8,7E-04
-3,0	4,4E-03
-2,5	1,8E-02
-2,0	5,4E-02
-1,5	1,3E-01
-1,0	2,4E-01
-0,5	3,5E-01
0,0	4,0E-01
0,5	3,5E-01
1,0	2,4E-01
1,5	1,3E-01
2,0	5,4E-02
2,5	1,8E-02
3,0	4,4E-03
3,5	8,7E-04
4,0	1,3E-04
4,5	1,6E-05
5,0	1,5E-06

Rys. 1.8. Wzór arkusza do zadania 2

Literatura

1. Bourg D. Excel w nauce i technice. Receptury. Gliwice, Helion, 2006.
2. Gonet M. Excel w obliczeniach naukowych i inżynierskich. Wydanie II. Gliwice, Helion, 2011.
3. Szydłowski H. Pracownia fizyczna. Warszawa, PWN, 1999.
4. Walkenbach J. Microsoft Excel 2010PL. Biblia. Gliwice, Helion, 2011.

2. Realizacja warunkowej koncepcji algorytmicznej na przykładzie rozwiązania równania kwadratowego

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Arkusz kalkulacyjny MS Excel 2007/2010

Wstęp teoretyczny

Zazwyczaj przed przystąpieniem do implementacji rozwiązania danego problemu obliczeniowego, bez względu na używane do tego celu środowisko programistyczne, tworzy się tzw. algorytm rozwiązania, czyli starannie dobrany ciąg instrukcji elementarnych będących zleceniami wykonania operacji podstawowych. Nazwa algorytm pochodzi od nazwiska arabskiego matematyka Muhammada ibn Musa al – Chorezmi (z Chorezmu), który opisał system dziesiętnego kodowania liczb i sztukę liczenia w tym systemie. Było to ok. 820 roku n.e [1]. Algorytm zawiera instrukcje sterujące, które odpowiadają za kolejność wykonywania akcji podstawowych. Instrukcje te możemy podzielić następująco:

- **Bezpośrednie następstwo**

Jest to instrukcja postaci:

- wykonaj polecenie 1, a następnie
- wykonaj polecenie 2, a następnie
- wykonaj polecenie 3 itd.

Prostym przykładem algorytmu zawierającego instrukcje tej postaci jest zagadnienie obliczania objętości prostopadłościanu. Otrzymamy wówczas następujące po sobie polecenia:

- podaj jednostkę wymiaru,
- podaj *długość* podstawy prostopadłościanu,
- podaj *szerokość* podstawy prostopadłościanu,
- podaj *wysokość* prostopadłościanu,
- oblicz objętość ze wzoru: *długość· szerokość·wysokość.*
- wyświetl *wynik.*

- **Wybór warunkowy**

Jest to instrukcja postaci:

***Jeżeli** Q, **to** wykonaj polecenie A, **w przeciwnym razie** wykonaj polecenie B*

Jak widać wykonanie określonego polecenia jest uzależnione od spełnienia albo niespełnienia warunku *Q*. Jeżeli warunek *Q* jest spełniony, tzn. jest *prawdą*, to wykonuje się polecenie A. Jeżeli warunek nie jest spełniony tzn. zwraca wartość *fałsz*, wykona się polecenie B. Zagadnieniem wyboru warunkowego zajmiemy się w dalszej części instrukcji.

- **Iteracja ograniczona**

Jest to instrukcja pętli o następującej postaci:

Wykonuj polecenie A dokładnie N razy

- **Iteracja warunkowa (nieograniczona)**

Podobnie jak w powyższym przypadku jest to pętla. Postać tej instrukcji przedstawia się następująco:

Dopóki Q wykonuj polecenie A

Różnica pomiędzy pierwszą i drugą instrukcją iteracji polega na tym, że w pierwszym przypadku z góry wiemy ile razy polecenie zawarte w pętli zostanie powtórzone. W drugim przypadku polecenie będzie powtarzane dopóty, dopóki wyrażenie warunkowe *Q* będzie spełnione tzn. będzie przyjmować wartość *prawda*. Zagadnieniem iteracji zajmiemy się dokładniej w następnym ćwiczeniu.

Jednym ze sposobów prezentacji algorytmu jest lista kroków. Przykład zastosowania listy kroków do rozwiązania równania kwadratowego postaci:

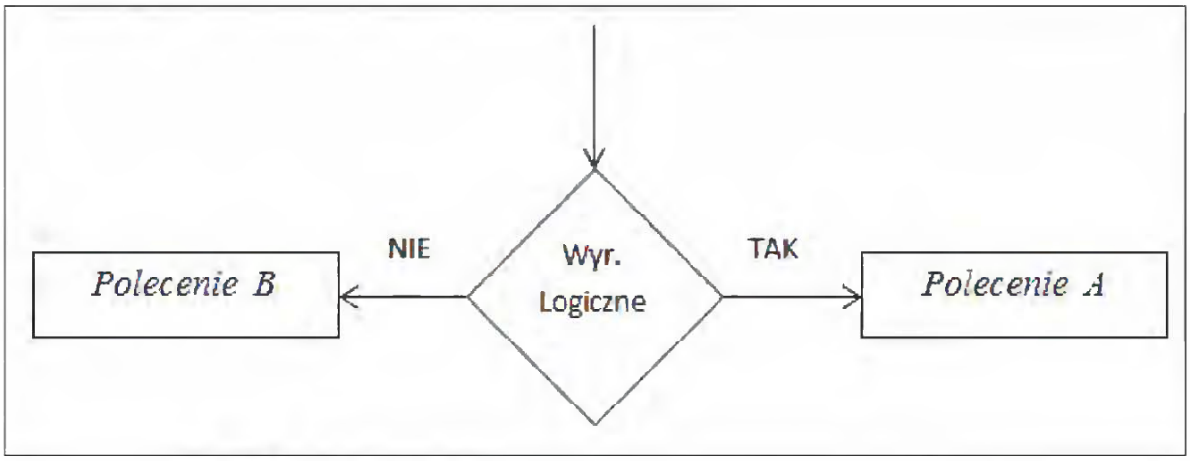
$$ax^2+bx+c=0 \qquad (1)$$

przedstawia się następująco:

1. Sprawdzamy, czy a=0
2. Jeżeli tak, to wyświetlamy komunikat: „równanie nie jest kwadratowe"
3. Jeżeli nie, to liczymy deltę
4. Jeżeli delta jest mniejsza od zera to wyświetlamy komunikat „brak rozwiązań rzeczywistych"
5. Jeżeli nie jest to liczymy x_1 i x_2 i wyświetlamy wynik

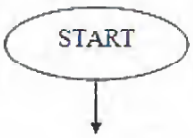
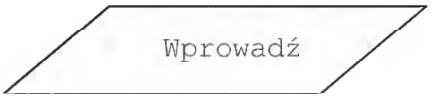
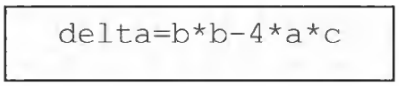
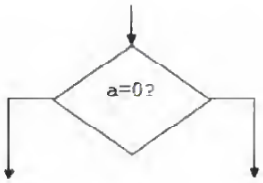
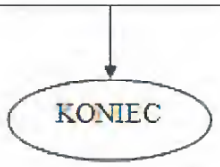
Powyższy sposób zapisu algorytmu nie jest wygodny, zwłaszcza w przypadku instrukcji zagnieżdżonych oraz iteracji. Lepszy wybór stanowi zapis w postaci schematu blokowego. Tabela 2.1 przedstawia podstawowe elementy schematu blokowego oraz opis ich znaczenia. Rysunki 2.1 i 2.2 przedstawiają algorytmiczną koncepcję wyboru warunkowego zapisanego w postaci schematu blokowego.

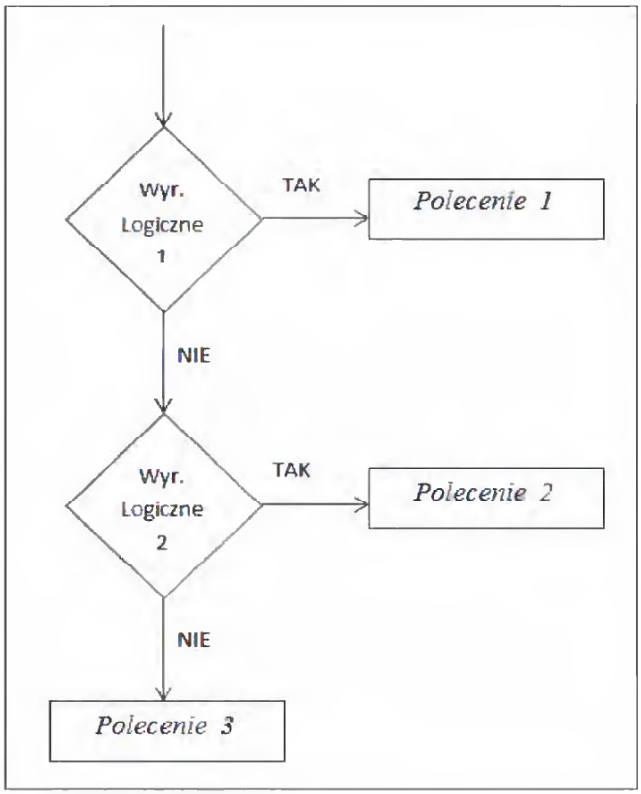
- Warunek pojedynczy (rysunek 2.1)
- Zagnieżdżona instrukcja warunkowa (rysunek 2.2)



Rys. 2.1. Schemat blokowy wyrażenia warunkowego (warunek pojedynczy)

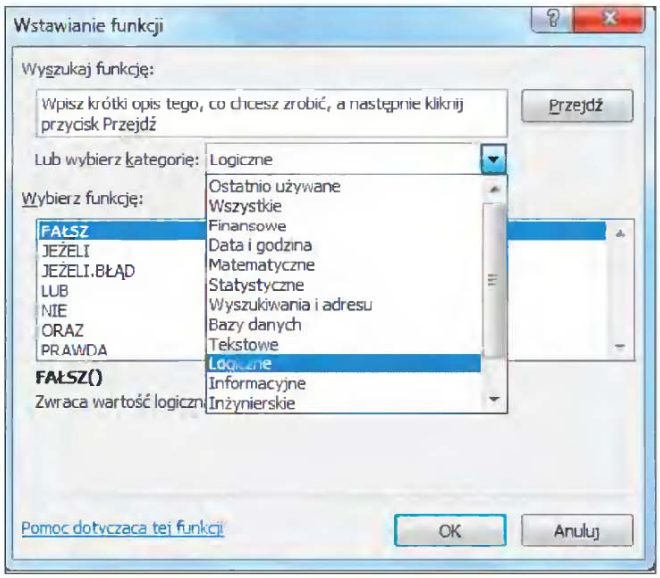
Tabela 2.1. Opis podstawowych elementów schematu blokowego

	Element rozpoczynający
	Blok wejścia/ wyjścia oznaczający czynność wprowadzania danych lub wyświetlenia wyniku
	Blok przetwarzania oznaczający wykonywanie określonych operacji (działanie, przypisanie)
	Blok decyzyjny (instrukcja warunkowa) oznacza element wyboru jednego z dwóch wariantów dalszego wykonywania programu
	Element kończący schemat blokowy,



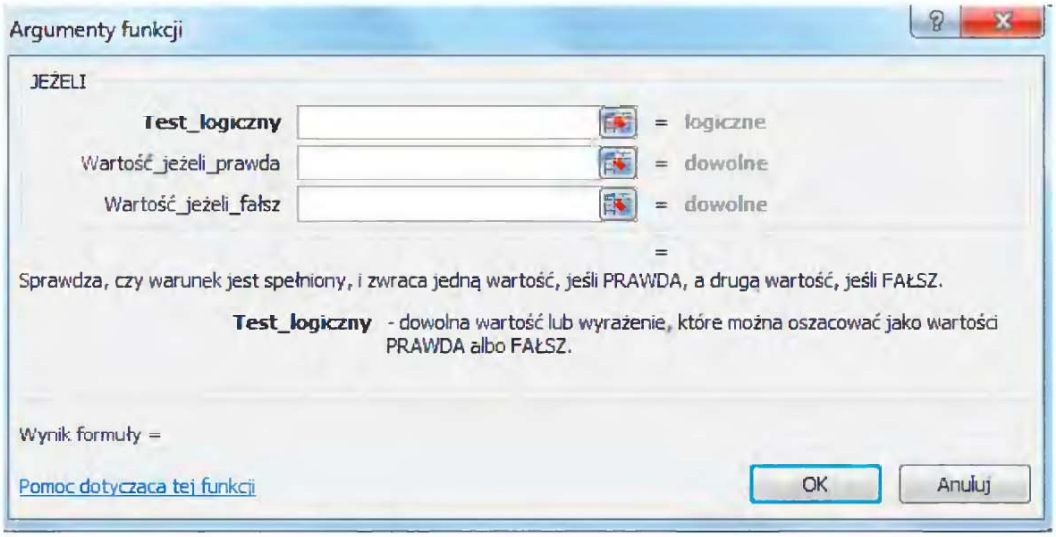
Rys. 2.2. Schemat blokowy wyrażenia warunkowego zagnieżdżonego

Do realizacji instrukcji warunkowej w arkuszu kalkulacyjnym Microsoft Excel służy funkcja *JEŻELI*, która znajduje się w grupie funkcji logicznych (rysunek 2.3).



Rys. 2.3. Okno wstawiania funkcji

Po wybraniu funkcji JEŻELI pojawi się okno przedstawione na rysunku 2.4.



Rys. 2.4. Okno funkcji warunkowej JEŻELI

Opis poszczególnych pól edycyjnych przedstawia się następująco:

- **Test logiczny**

Test logiczny, to wyrażenie warunkowe, które zwraca wartość PRAWDA lub FAŁSZ w zależności od spełnienia podanego warunku. Załóżmy na przykład, że nasz test logiczny to pytanie: „czy wartość wyrażenia w kolumnie A jest większa od wartości wyrażenia w kolumnie B?".

- **Wartość jeżeli PRAWDA**

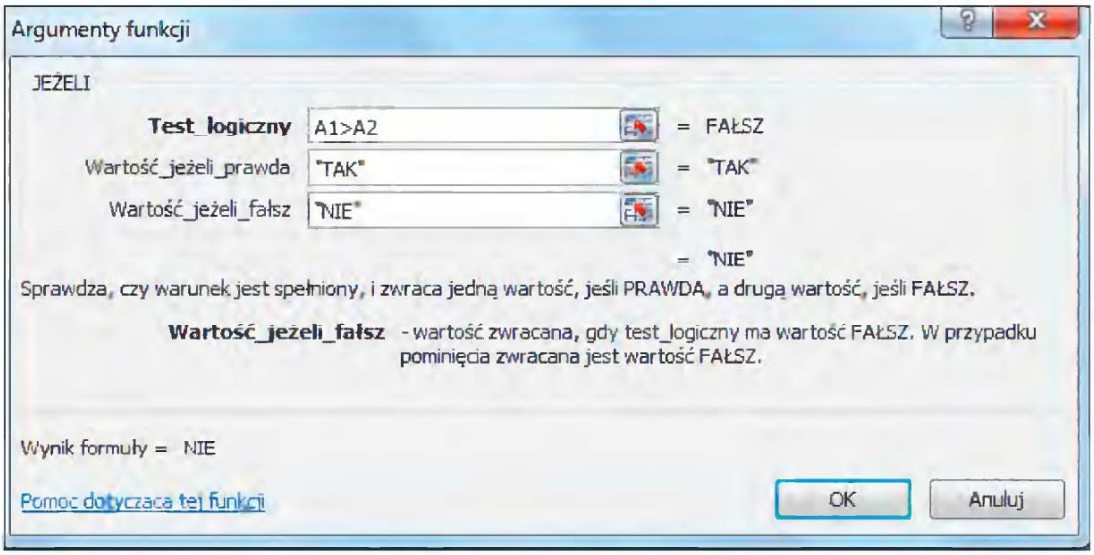
Jest to wartość obliczana (wyświetlana) jeżeli wartość zwrócona w teście logicznym jest PRAWDĄ. Załóżmy w tym przykładzie, że chcemy wyświetlić „TAK"

- **Wartość jeżeli FAŁSZ**

Jest to wartość obliczana (wyświetlana) jeżeli wartość zwrócona w teście logicznym jest FAŁSZEM. Załóżmy w tym przykładzie, że chcemy wyświetlić „NIE"

Uzupełnienie okna zgodnie z powyższym przykładem przedstawia rysunek 2.5 Formuła, która pojawi się w pasku poleceń będzie mieć następującą postać:

JEŻELI (A1 > A2 ; "TAK" ; "NIE")



Rys. 2.5. Okno funkcji warunkowej JEŻELI dla rozważanego przypadku porównania dwóch liczb wprowadzonych do komórek A1 i A2

Powyższe wyrażenie oznacza, że jeżeli wartość w komórce A1 jest większa od wartości liczby w komórce A2, to wówczas w aktywnej komórce, w której tę formułę wpisujemy, wyświetli się napis TAK. W przeciwnym razie wyświetli się napis NIE. Oczywiście w polach edycyjnych *wartość_jeżeli_prawda* oraz *wartość_jeżeli_fałsz* można wpisywać zarówno tekst jak i liczby oraz całe formuły. Podobnie pole *Test_logiczny* może zawierać bardziej skomplikowane wyrażenia logiczne.

Przy konstruowaniu wyrażenia logicznego korzysta się z operatorów relacji (tabela 2.2) oraz operatorów logicznych. W każdym przypadku zwracana jest jedna z dwóch wartości: prawda lub fałsz.

Tabela 2.2. Operatory relacji stosowane w arkuszu kalkulacyjnym

Operator	Działanie
>	większy niż
>=	większy lub równy
<	mniejszy niż
<=	mniejszy lub równy
=	równy
<>	różny

W przypadku operacji logicznych w Excelu korzysta się z funkcji wbudowanych, które przedstawione zostały w tabeli 2.3.

Tabela 2.3. Funkcje będące operatorami logicznymi

Operator	Działanie
NIE(Wyr1)	Zamienia wartość PRAWDA na FAŁSZ lub odwrotnie
LUB (Wyr1,Wyr2,Wyr3)	Zwraca wartość PRAWDA jeżeli którekolwiek z wyrażeń logicznych jest prawdziwe
ORAZ (Wyr1,Wyr2,..)	Zwraca wartość PRAWDA jeżeli wszystkie wyrażenia logiczne są prawdziwe

2.1. Zadania do wykonania

Zadanie 1.

Ułożyć formularz znajdujący pierwiastki równania kwadratowego :

$$ax^2 + bx + c = 0,$$

gdzie współczynniki a,b,c (dowolne) wprowadzane są z klawiatury przez użytkownika.

Należy rozważyć następujące przypadki:

- $a = 0,\ b \neq 0,\ c \neq 0$ - równanie ma rozwiązanie: $x = -\frac{c}{b}$
- $a \neq 0,\ b \neq 0,\ c \neq 0$ - rozwiązanie jest uzależnione od wartości delty: $\Delta = b^2 - 4ac$:
  - jeżeli $\Delta > 0$, to równanie ma dwa rozwiązania: $x_1 = \frac{-b - \sqrt{\Delta}}{2a}$; $x_2 = \frac{-b + \sqrt{\Delta}}{2a}$
  - jeżeli $\Delta = 0$, to obydwa pierwiastki są równe i wynoszą $x = \frac{-b}{2a}$
  - jeżeli $\Delta < 0$ - brak pierwiastków rzeczywistych

Literatura

1. Homo Informaticus. Red. M. Sysło. Warszawa, WWSI, 2012.
2. Harel D. Rzecz o istocie informatyki. Algorytmika. Warszawa, WNT, 2009.
3. Walkenbach J. Microsoft Excel 2010PL. Biblia. Gliwice, Helion, 2011.

3. Iteracja na przykładzie metody Newtona-Raphsona znajdowania miejsca zerowego funkcji

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Arkusz kalkulacyjny MS Excel 2007/2010

Wstęp teoretyczny

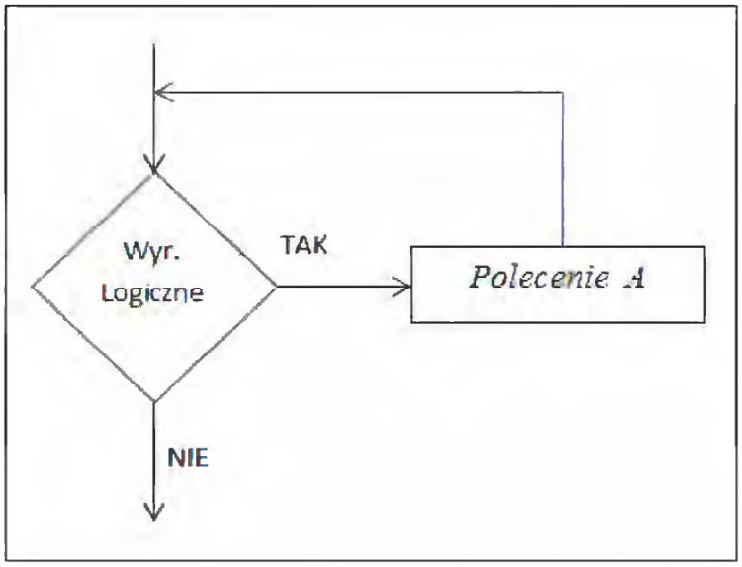
W zadaniach obliczeniowych realizowanych w arkuszu kalkulacyjnym czy aplikacji komputerowej często zachodzi potrzeba powtórzenia tych samych instrukcji, określoną ilość razy. Liczba powtórzeń może być znana z góry lub też iteracja jest kontynuowana dopóty, dopóki spełniony będzie określony warunek zadany w pętli. W pierwszym przypadku mówimy o *iteracji ograniczonej*. W drugim zaś o *iteracji warunkowej*.

- Iteracja warunkowa

Postać instrukcji warunkowej przedstawia się następująco:

Dopóki Wyr. Logiczne wykonuj Polecenie A

Jedna instrukcja powtórzenia nazywa się obrotem. Obrót nastąpi tylko wówczas, jeżeli wyrażenie logiczne będzie zwracało w wyniku wartość PRAWDA. Liczba cykli pętli będzie równa liczbie obliczeń wyrażenia logicznego, których obliczoną wartością jest wartość PRAWDA. Rysunek 3.1 przedstawia zapis iteracji warunkowej w postaci schematu blokowego.



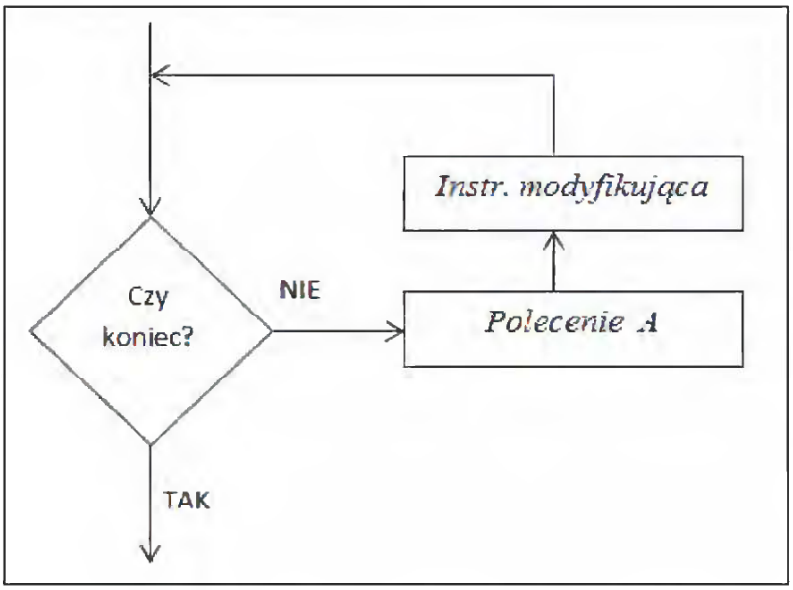
Rys. 3.1. Iteracja warunkowa (nieograniczona) zapisana w postaci schematu blokowego

- Iteracja ograniczona

Jest to instrukcja następującej postaci:

Wykonuj Polecenie A dokładnie N razy

Iterację ograniczoną stosuje się wtedy, gdy z góry znamy liczbę obrotów pętli. Schemat blokowy ograniczonej kompozycji algorytmicznej przedstawiono na rysunku 3.2. Znajduje się tam blok o nazwie Instrukcja modyfikująca. Zadaniem tego bloku instrukcji jest modyfikacja zmiennej odpowiedzialnej za zakończenie iteracji.



Rys. 3.2. Iteracja ograniczona. Zapis w postaci schematu blokowego

Najprostszy sposób realizacji iteracji w arkuszu kalkulacyjnym odbywa się poprzez kopiowanie formuł. Jest to możliwe dzięki automatycznej zmianie adresów komórek w wyrażeniach. Rozważmy prosty przykład dyskretyzacji funkcji danej wzorem:

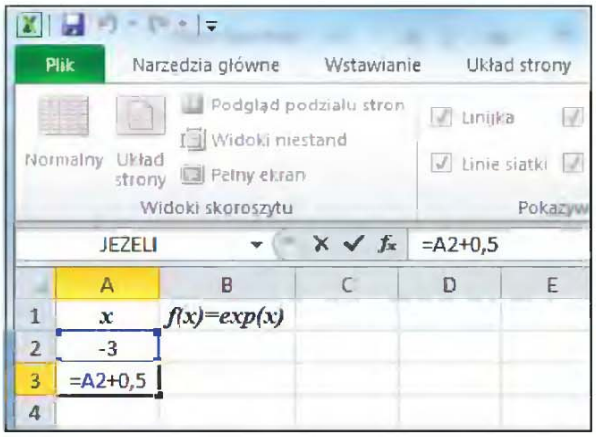
$$f(x)=exp(x) \tag{2}$$

Chcemy otrzymać współrzędne punktów w przedziale $x\in<-3;\ 3>$ z krokiem $x=0.5$. W kolumnie otrzymamy x, w kolumnie B zaś wartości $f(x)$. Po wprowadzeniu wartości -3 w komórce A2 przechodzimy do następnej komórki i wprowadzamy formułę (rysunek 3.3a):

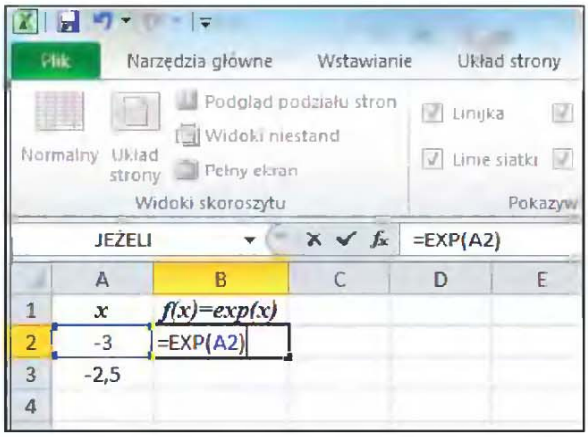
=A2+0.5

W komórce B2 wprowadzamy formułę ze wzorem funkcji, w której x to adres komórki A2 (rysysunek 3.3b):

=EXP(A2)

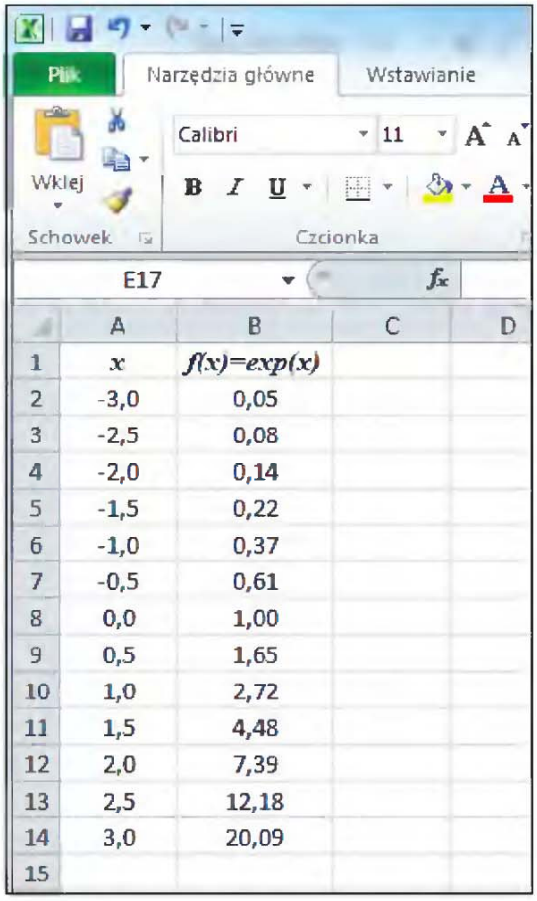


Rys. 3.3a. Formuła dla *x* w kolumnie A



Rys. 3.3b. Formuła dla f(*x*) w kolumnie B

Wprowadzone formuły kopiujemy na pozostałe komórki (iteracja). Rezultat wykonania zaprezentowany został na rysunku 3.4. Dokładność wyświetlania liczb w kolumnie B ograniczono do dwóch miejsc po przecinku.

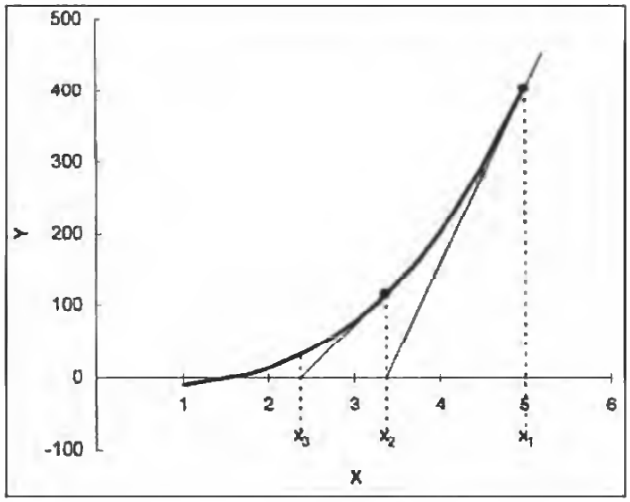


Rys. 3.4. Wynik zastosowania iteracji dla przykładu tabelaryzacji funkcji

W dalszej części zajmiemy się implementacją metody Newtona-Rapsona znajdowania miejsca zerowego funkcji. Metoda używa pierwszej pochodnej funkcji w punkcie w celu znajdowania kolejnych przybliżeń miejsca zerowego funkcji. Ogólny wzór znajdowania kolejnych przybliżeń miejsca zerowego:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} \tag{3}$$

Obliczenia miejsca zerowego kończone są po osiągnięciu określonej dokładności, tj. gdy zmiana wartości pierwiastka jest mniejsza niż założona wartości *epsilon*. Rysunek 3.5 przedstawia wykres funkcji *f(x)*.



Rys. 3.5. Wykres funkcji *f(x)*[1]

Niech x_1 będzie punktem startowym. Wówczas kolejne przybliżenie x_2 zgodnie ze wzorem (4) będzie równe:

$$x_2 = x_1 - \frac{f(x_1)}{f'(x_1)} \tag{4}$$

W celu obliczenia pochodnej funkcji w punkcie w arkuszu kalkulacyjnym wykorzystamy tzw. metody różnicowe. Dobre przybliżenie pochodnej uzyskamy dobierając odpowiednią wartość Δx, która zwiększy wartość *f(x)* o Δy. Pochodną oznaczymy jako *m* i zapiszemy:

$$f'(x) = m = \frac{\Delta y}{\Delta x} = \frac{f(x+\Delta x) - f(x)}{\Delta x} \tag{5}$$

Wstawiając (5) do (4) otrzymujemy:

$$x_2 = x_1 - \frac{\Delta x \cdot f(x_1)}{f(x_1+\Delta x) - f(x_1)} \tag{6}$$

Ostatecznie więc możemy zapisać:

$$x_2 = (m \cdot x_1 - f(x_1)) / m \tag{7}$$

Jeżeli przybliżenie x_2 nie jest satysfakcjonujące dokonuje się kolejnych przybliżeń x_3, x_4, itd. aż do osiągnięcia pożądanej dokładności.

Warunek zakończenia obliczeń:

$$|x_{k+1} - x_k| \leq \varepsilon \tag{8}$$

ZADANIA DO WYKONANIA

Dana jest funkcja [1]:

$$f(x) = 3x^3 + 2.5x^2 - 5x - 11 \qquad (9)$$

Zadanie 1.

Zaprojektuj arkusz (wzór poniżej) do obliczenia miejsca zerowego funkcji zadanej wzorem (9). Podaj wynik obliczeń z dokładnością do 5 miejsc po przecinku: $\varepsilon = 10^{-5}$. Ustawić parametr Δ=10. Ustalić liczbę iteracji konieczną do osiągnięcia dokładności 10^{-5}.

	A	B	C	D	E	F	G	H
1	Iteracyjna metoda Newtona-Raphsona poszukiwania miejsca zerowego							
2							Δ:	1.00E+01
3	Lp.	x_i	$y_i=f(x_i)$	$x_i+\Delta$	$f(x_i+\Delta)$	m	x_{i+1}	epsilon
4	0	5.000000	401.500000	15.000000	10601.500000	1020	4.606372549	3.3E-01
5	1	4.606373	312.238078	14.606373	9797.972125	948.5734	4.277206587	2.8E-01
6	2	4.277207	248.098226	14.277207	9157.949140	890.9851	3.99875274	2.4E-01
7	3	3.998753	200.801745	13.998753	8638.718961	843.7917	3.760777253	2.1E-01
8	4	3.760777	165.125772	13.760777	8210.774262	804.5648	3.555541127	1.8E-01
9	5	3.555541	137.673071	13.555541	7853.205686	771.5533	3.377104877	1.6E-01
10	6	3.377105	116.172564	13.377105	7550.857559	743.4685	3.220847329	1.4E-01
11	7	3.220847	99.068241	13.220847	7292.520380	719.3452	3.083127304	1.2E-01
12	8	3.083127	85.270157	13.083127	7069.751754	698.4482	2.961042141	1.1E-01
13	9	2.961042	73.999430	12.961042	6876.088815	680.2089	2.852252877	9.7E-02
14	10	2.852253	64.689298	12.852253	6706.510713	664.1821	2.754855952	8.8E-02
15	11	2.754856	56.920516	12.754856	6557.064489	650.0144	2.667287868	7.9E-02
16	12	2.667288	50.378277	12.667288	6424.599962	637.4222	2.58825347	7.2E-02
17	13	2.588253	44.822938	12.588253	6306.578776	626.1756	2.516671411	6.5E-02
18	14	2.516671	40.069765	12.516671	6200.934754	616.0865	2.451632224	5.9E-02
19	15	2.451632	35.974701	12.451632	6105.970365	606.9996	2.392365788	5.4E-02
20	16	2.392366	32.424206	12.392366	6020.278971	598.7855	2.338215834	5.0E-02
21	17	2.338216	29.327909	12.338216	5942.685747	591.3358	2.288619802	4.6E-02
22	18	2.288620	26.613218	12.288620	5872.202283	584.5589	2.243092794	4.2E-02
23	19	2.243093	24.221330	12.243093	5807.991360	578.377	2.201214695	3.9E-02
24	20	2.201215	22.104233	12.201215	5749.339345	572.7235	2.162619748	3.6E-02

Rys. 3.6. Wzór arkusza z implementacją metody Newtona- Raphsona

Należy zwrócić uwagę, że zaprezentowany na rysunku 3.6 arkusz jest tylko wycinkiem całości. Przy takiej wartości parametru delta iteracji, a więc także i wierszy, będzie znacznie więcej.

Zadanie 2.

Rysunek 3.7 przedstawia rezultaty obliczeń dla Δ=1.0. Jak widać zmiana delty wpłynęła na zwiększenie szybkości działania metody – zauważalna jest znacznie mniejsza liczba iteracji (wierszy) do osiągnięcia dokładności rzędu 10^{-5}.Zbadać, jak zmienia się liczba iteracji wraz ze zmianą parametru Δ. Wykonać wykres dla następujących wartości Δ: 0.01, 0.1, 0.5, 1.0, 2.0, 5.0, 10.0.

	A	B	C	D	E	F	G	H
1	Iteracyjna metoda Newtona-Raphsona poszukiwania miejsca zerowego							
2							Δ:	1.00E+00
3	Lp.	x_i	$y_i=f(x_i)$	$x_i+\Delta$	$f(x_i+\Delta)$	m	x_{i+1}	epsilon
4	0	5,000000	401,500000	6,000000	697,000000	295,5	3,641285956	8,7E-01
5	1	3,641286	148,780011	4,641286	319,588685	170,8087	2,770252846	5,4E-01
6	2	2,770253	58,113749	3,770253	166,465996	108,3522	2,23391188	3,1E-01
7	3	2,233912	23,750435	3,233912	100,438462	76,68803	1,924209883	1,6E-01
8	4	1,924210	10,009054	2,924210	70,771245	60,76219	1,759484849	8,1E-02
9	5	1,759485	4,283014	2,759485	57,277884	52,99487	1,678665447	3,7E-02
10	6	1,678665	1,842490	2,678665	51,205066	49,36258	1,641339799	1,7E-02
11	7	1,641340	0,793582	2,641340	48,518306	47,72472	1,624711478	7,3E-03
12	8	1,624711	0,341852	2,624711	47,344999	47,00315	1,617438512	3,2E-03
13	9	1,617439	0,147254	2,617439	46,836359	46,68911	1,614284584	1,4E-03
14	10	1,614285	0,063428	2,614285	46,616644	46,55322	1,612922106	5,9E-04
15	11	1,612922	0,027320	2,612922	46,521889	46,49457	1,612334508	2,5E-04
16	12	1,612335	0,011767	2,612335	46,481054	46,46929	1,612081279	1,1E-04
17	13	1,612081	0,005068	2,612081	46,463461	46,45839	1,611972182	4,7E-05
18	14	1,611972	0,002183	2,611972	46,455882	46,4537	1,611925187	2,0E-05
19	15	1,611925	0,000940	2,611925	46,452618	46,45168	1,611904944	8,7E-06
20	16	1,611905	0,000405	2,611905	46,451212	46,45081	1,611896225	3,8E-06
21	17	1,611896	0,000174	2,611896	46,450607	46,45043	1,61189247	1,6E-06
22	18	1,611892	0,000075	2,611892	46,450346	46,45027	1,611890852	7,0E-07
23	19	1,611891	0,000032	2,611891	46,450233	46,4502	1,611890155	3,0E-07
24	20	1,611890	0,000014	2,611890	46,450185	46,45017	1,611889855	1,3E-07

Rys. 3.6. Wzór arkusza z implementacją metody Newtona- Raphsona dla Δ=1.0

Zadanie3.

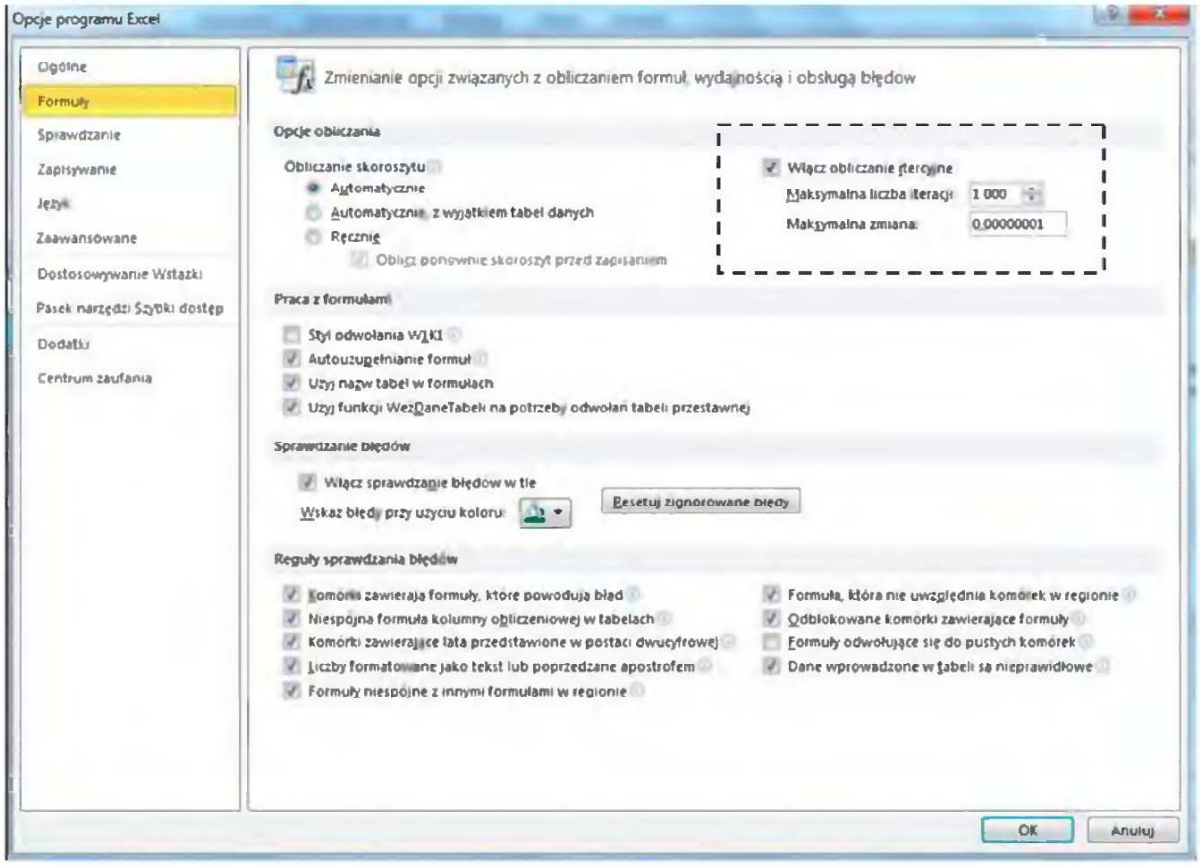
Wykorzystaj funkcję **Szukaj wyniku** (*Dane -> Analiza warunkowa -> Szukaj wyniku*), w celu szybkiego rozwiązania powyższego problemu. W tym celu należy przygotować dwie komórki arkusza: w pierwszej wprowadzić startowy *x*, w drugiej zaś wzór funkcji, dla której będą poszukiwane miejsca zerowe. Uzupełnić pola edycyjne wg wzoru:

Rys 3.7. Metoda szukaj wyniku

W celu ustawienia dokładności obliczeń metodą Szukaj wyniku należy wybrać:

Plik -> Opcje -> Formuły

Pojawi się wówczas okno przedstawione na rysunku 3.8, w którym możemy ustawić dokładność obliczeń. Należy pamiętać, że wraz ze wzrostem dokładności obliczeń rośnie również czas potrzebny na ich wykonanie.



Rys. 3.8. Okno, w którym można ustawić precyzję obliczeń iteracyjnych

Literatura

1. Billo J. Excel For Scientists and Engineers. New Jersey, Wiley. 2007.
2. Gonet M. Excel w obliczeniach naukowych i inżynierskich. Wydanie II. Gliwice, Helion, 2011.
3. Walkenbach J. Microsoft Excel 2010PL. Biblia. Gliwice, Helion, 2011.

4. Obliczanie pola pod wykresem krzywej

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Arkusz kalkulacyjny MS Excel 2007/2010

Wstęp teoretyczny

Obliczanie pola pod wykresem krzywej jest zagadnieniem dość powszechnym i nazywa się - w języku matematyki na poziomie studiów wyższych - całkowaniem. Wyznaczanie tzw. całki oznaczonej jest proste, o ile znana jest postać analizowanej funkcji oraz jej funkcja pierwotna. Wielokrotnie w pracy badawczej zdarza się, że chcemy wyznaczyć pole pod wykresem złożonym z danych doświadczalnych. Wówczas żadna postać funkcji nie jest znana i opieramy się na numerycznych metodach przybliżonych. W dalszej części instrukcji omówione zostaną następujące metody:

- Metoda prostokątów
- Metoda trapezów
- Metoda parabol

Idea wszystkich trzech metod jest podobna. Przedział całkowania dzieli się na równe podprzedziały, po czym dokonując interpolacji funkcji podcałkowej, obliczane są pola powierzchni poszczególnych podprzedziałów. Na koniec sumuje się obliczone pola.

Metoda prostokątów

W metodzie prostokątów funkcja podcałkowa interpolowana jest funkcją stałą, a obliczane pole jest polem prostokąta. Schemat działania metody jest następujący:

- dzielimy przedział całkowania na n równych części o długości h:

$$h = \frac{b-a}{n} \tag{9}$$

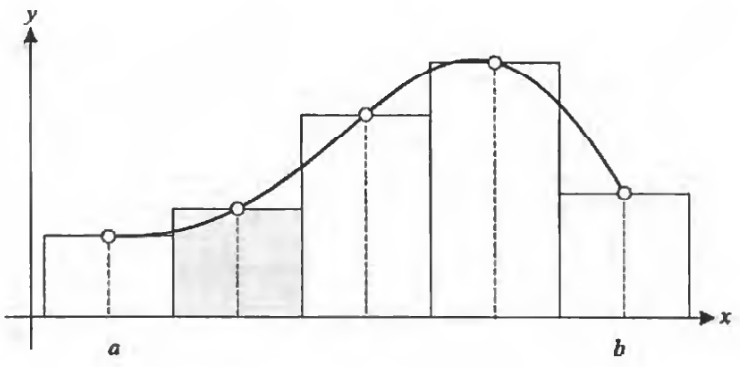
- obliczamy wartość funkcji w środku *i-tego* przedziału:

$$f_i = f\left(a + \frac{(2i-1)h}{2}\right) \tag{10}$$

- sumujemy pola powierzchni wszystkich prostokątów, których boki wynoszą h i f_i :
-

$$\int_a^b f(x)dx \cong h\sum_{i=1}^{n} f_i \tag{11}$$

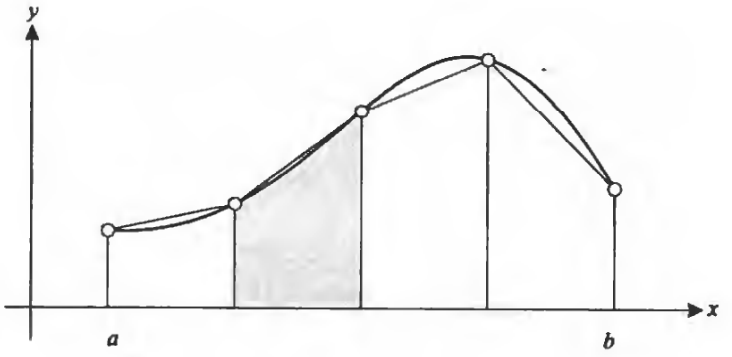
Rysunek 4.1 przedstawia schematycznie całkowanie metodą prostokątów.



Rys. 4.1. Przykład całkowania metodą prostokątów [1]

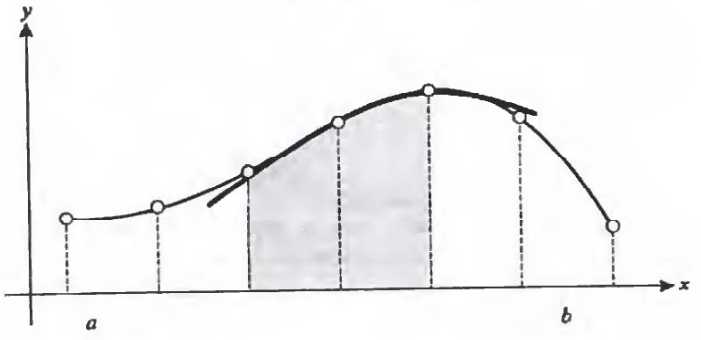
Metoda trapezów i parabol

Dokładniejszymi metodami całkowania numerycznego są metody trapezów i parabol, w której funkcja podcałkowa aproksymowana jest wielomianem pierwszego stopnia, a pole pod krzywą przybliżane jest sumą pól trapezów powstałych przez połączenie odcinkami węzłów z funkcji podcałkowej (rysunek 4.2).



Rys. 4.2. Ilustracja całkowania numerycznego metodą trapezów [1]

W metodzie parabol funkcją aproksymującą jest parabola, którą wpisuje się w trzy sąsiednie węzły – wartości funkcji podcałkowej (rysunek 4.3).



Rys. 4.3. Całkowanie metodą parabol [1]

Zadania do wykonania

Zadanie 1.

Dana jest funkcja: $y = \frac{x^3}{e^x - 1}$ [2]. Przyjmując Δx=0.1 obliczyć przybliżoną wartość pola pod krzywą w przedziale $x \in (0;15)$, korzystając ze wzoru:

$$Pole = \sum_{i=1}^{n-1} \frac{y_i + y_{i+1}}{2}(x_{i+1} - x_i) \tag{12}$$

Zwrócić uwagę, że 0 nie należy do przedziału więc należy wziąć wartość bliską zeru, np. 0.000001. Wynik porównać z wartością dokładną równą $\pi^4/15 = 6.494$. Obliczyć procentowy błąd stosowanej metody wg. wzoru:

$$\delta = \frac{|wartość\ obliczona - wartość\ dokładna|}{wartość\ dokładna} \cdot 100\% \tag{13}$$

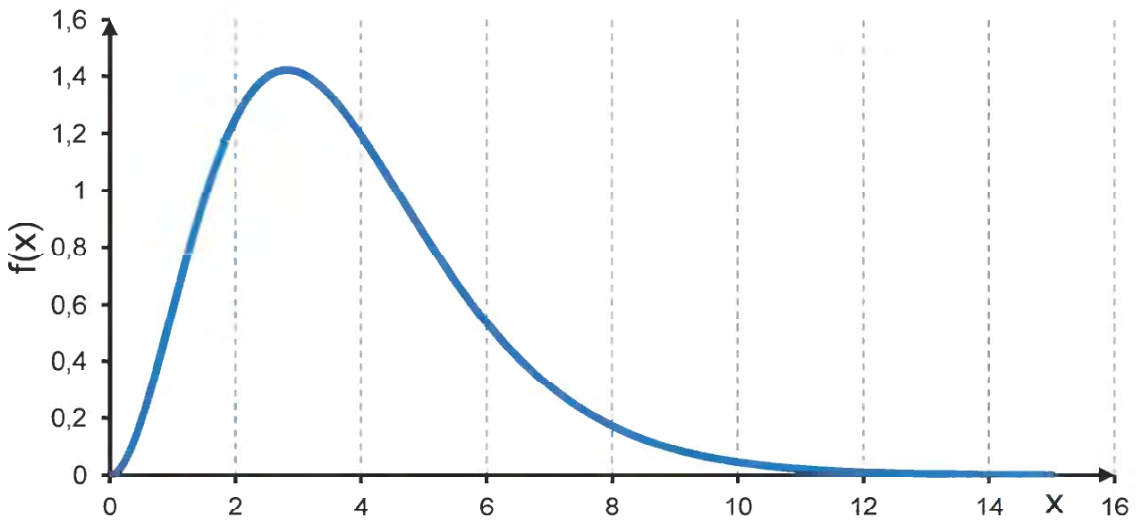
W celu wykonania zadania należy przygotować arkusz według wzoru zamieszczonego na rysunku 4.4. Kolejne kolumny zawierają:

- numerację węzła (kolumna A),
- argumenty funkcji z krokiem 0.1 (kolumna B),
- obliczone wartości funkcji (kolumna C),
- obliczone pola poszczególnych trapezów (kolumna D).

	A	B	C	D
1	**i**	**x**	**$f_1(x)=x^3/(e^x-1)$**	**$dP=(y_i+y_{i+1})/2*(x_{i+1}-x_i)$**
2	1	0,0000010	0,000000	
3	2	0,100	0,009508	0,000475
4	3	0,200	0,036133	0,002282
5	4	0,300	0,077174	0,005665
6	5	0,400	0,130128	0,010365
7	6	0,500	0,192687	0,016141
8	7	0,600	0,262736	0,022771
9	8	0,700	0,338347	0,030054
10	9	0,800	0,417775	0,037806
11	10	0,900	0,499451	0,045861
12	11	1,000	0,581977	0,054071
13	12	1,100	0,664117	0,062305
14	13	1,200	0,744790	0,070445
15	14	1,300	0,823063	0,078393
16	15	1,400	0,898141	0,086060

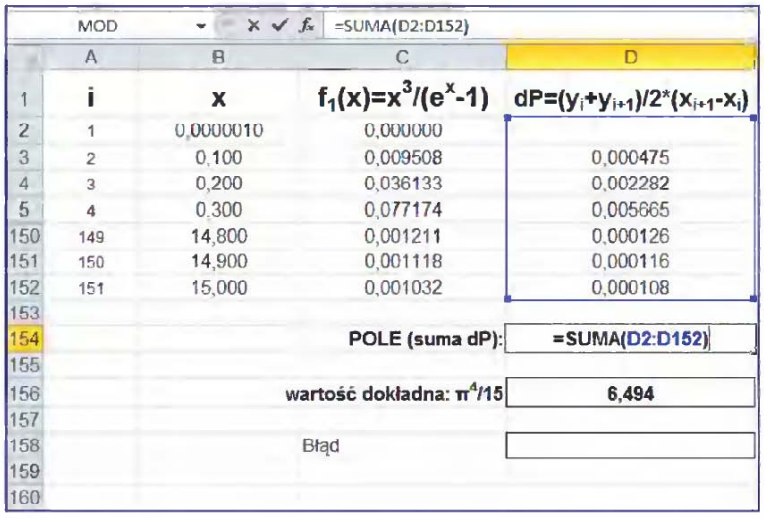
Rys. 4.4. Wzór arkusza do zadania 1

Przy wykonywaniu zadania należy pamiętać o stosowaniu funkcji kopiowania formuł. Najwygodniej jest wprowadzić wzory do komórek znajdujących się w początkowym wierszu, po czym dokonać kopiowania na pozostałe komórki arkusza. Po obliczeniu wartości *f(x)* oraz poszczególnych pól trapezu należy wykonać wykres punktowy funkcji, w celu sprawdzenia poprawności obliczeń. Wykres powinien wyglądać jak na rysunku 4.5.



Rys. 4.5. Wykres funkcji zadania 1

Następnie należy obliczyć pole całkowite pod wykresem krzywej poprzez wykonanie sumowania pól trapezów. Na koniec obliczany jest błąd.



	A	B	C	D
1	i	x	$f_1(x)=x^3/(e^x-1)$	$dP=(y_i+y_{i+1})/2*(x_{i+1}-x_i)$
2	1	0,0000010	0,000000	
3	2	0,100	0,009508	0,000475
4	3	0,200	0,036133	0,002282
5	4	0,300	0,077174	0,005665
150	149	14,800	0,001211	0,000126
151	150	14,900	0,001118	0,000116
152	151	15,000	0,001032	0,000108
153				
154			POLE (suma dP):	=SUMA(D2:D152)
155				
156			wartość dokładna: $\pi^4/15$	6,494
157				
158			Błąd	
159				
160				

Rys. 4.6. Wzór arkusza do obliczania sumy końcowej pól. Część wierszy została ukryta

Zadanie 2.

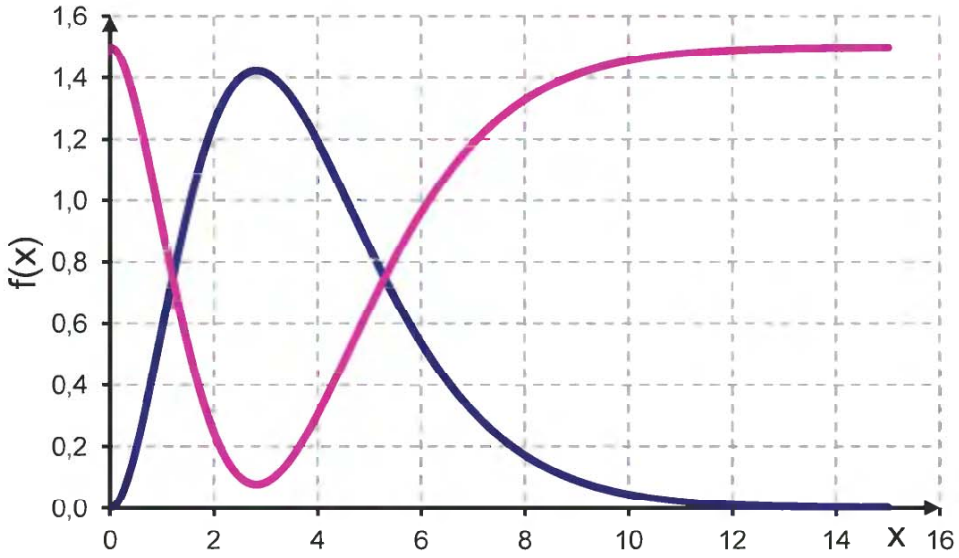
Dane są dwie funkcje:

$$f_1(x)=\frac{x^3}{e^x-1} \quad i \quad f_2(x)=1.5-\frac{x^3}{e^x-1} \tag{14}$$

Przyjmując $\Delta x=0.1$ obliczyć pole ograniczone wykresami funkcji zadanych wzorami (14). W tym celu należy:

- wykonać wykresy funkcji (w jednym układzie współrzędnych),
- znaleźć dokładne miejsca przecięcia się wykresów obydwu funkcji: x_a i x_b,
- dokonać podziału obszaru x: od x_a do x_b z krokiem $\Delta x=0.1$,
- obliczyć wartości funkcji $f(x)$ dla każdego x z przedziału od x_a do x_b,
- obliczyć pola trapezów dla poszczególnych wykresów funkcji,
- obliczyć pole obszaru ograniczonego wykresami funkcji.

Wykresy funkcji przedstawia rysunek 4.7.



Rys. 4.7. Wykresy funkcji z zadania 2

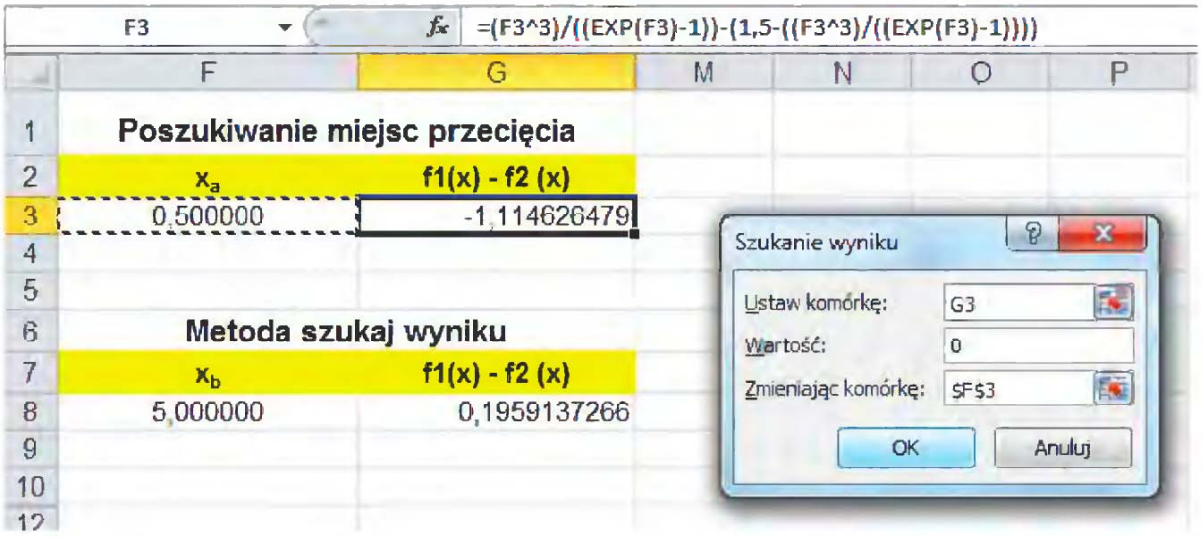
Z wykresów przedstawionych na rysunku 4.7 możemy odczytać przybliżone wartości x, przecięcia się wykresów funkcji danych wzorami (14). Pierwsze miejsce przecięcia, tj. x_a, znajduje się pomiędzy wartości 0 a 2. Drugie natomiast, x_b, pomiędzy wartościami 4 a 6. W celu precyzyjnego określenia miejsc przecięcia zastosujemy metodę *Szukaj wyniku*, omawianą w ćwiczeniu nr 3. Należy zauważyć, że podobnie jak poprzednio tak i tu będziemy poszukiwać wartości 0. Z tą różnicą, że teraz wartość 0, to nie miejsce zerowe ale różnica między wartościami funkcji f_1 i f_2.

Na rysunku 4.8 przedstawiony został fragment arkusza, w którym obliczone zostaną miejsca przecięcia się wykresów. W pierwszej komórce znajduje się wartość poszukiwana x, w drugiej zaś wzór funkcji, do którego program podstawia tę wartość. Po stwierdzeniu satysfakcjonującej dokładności – program kończy działanie.

F	G	M	N	O
Poszukiwanie miejsc przecięcia				
x_a	**f1(x) - f2 (x)**			
0,500000	=(F3^3)/((EXP(F3)-1))-(1,5-((F3^3)/((EXP(F3)-1))))			
Metoda szukaj wyniku				
x_b	**f1(x) - f2 (x)**			
5,000000	0,1959137266			

Rys. 4.8. Wzór arkusza do obliczenia punktów x_a i x_b przecięcia się wykresów funkcji

Po wprowadzeniu wzorów wybieramy x leżący blisko punktu przecięcia po czym ustawiamy parametry metody *Szukaj wyniku* tak jak na rysunku 4.9. Po znalezieniu pierwszego miejsca przecięcia analogicznie postępujemy w przypadku drugiego.



Rys. 4.9. Wzór arkusza do obliczenia punktów x_a i x_b przecięcia się wykresów

W dalszej części zadania należy dokonać podziału obszaru ograniczonego znalezionymi wartościami x_a i x_b . Następnie, postępując analogicznie jak w zadaniu 1, należy obliczyć wartości funkcji oraz pola trapezów pod wykresami. Wzór arkusza przedstawia rysunek 4.10.

	H	I	J	K	L
1	$x : \langle x_a ; x_b \rangle$	$f_1(x)=x^3/(e^x-1)$	$f_2(x)=1.5-(x^3/(e^x-1))$	Pole funkcji 1	Pole funkcji 2
2	1,206550	0,750000	0,750000411		
3	1,306550	0,828086	0,671913624	0,078904298	0,071095702
4	1,406550	0,902930	0,597069888	0,086550824	0,063449176
5	1,506550	0,973873	0,526126881	0,093840162	0,056159838
6	1,606550	1,040378	0,45962235	0,100712538	0,049287462
7	1,706550	1,102017	0,39798319	0,107119723	0,042880277
8	1,806550	1,158465	0,341534516	0,113024115	0,036975885
9	1,906550	1,209491	0,290508655	0,118397841	0,031602159
10	2,006550	1,254946	0,245053985	0,123221868	0,026778132
12	2,206550	1,328917	0,171083394	0,131183653	0,018816347
13	2,306550	1,357479	0,142520591	0,134319801	0,015680199
14	2,406550	1,380549	0,119450869	0,136901427	0,013098573
15	2,506550	1,398274	0,101725851	0,138941164	0,011058836
16	2,606550	1,410840	0,089159865	0,140455714	0,009544286
17	2,706550	1,418464	0,081536292	0,141465192	0,008534808
18	2,806550	1,421387	0,078613454	0,141992513	0,008007487
19	2,906550	1,419870	0,080130036	0,142062825	0,007937175
20	3,006550	1,414190	0,085810032	0,141702997	0,008297003

Rys. 4.10. Wzór arkusza do obliczania pól pod wykresami obydwu funkcji

Po obliczeniu pól powierzchni trapezów dla każdej z funkcji należy, analogicznie jak w zadaniu 1, wykonać sumowanie w wyniku którego otrzymamy pole pod wykresami funkcji. Pole obszaru ograniczonego wykresami funkcji obliczymy jako odpowiednia różnica tych pól.

Literatura

1. Stachurski M. Metody numeryczne w programie Matlab. Warszawa, Mikom, 2003.
2. Billo J. Excel For Scientists and Engineers. New Jersey, Wiley. 2007.
3. Walkenbach J. Microsoft Excel 2010PL. Biblia. Gliwice, Helion, 2011.

5. Tworzenie wykresów. Linie trendu

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Arkusz kalkulacyjny MS Excel 2007/2010

Wstęp teoretyczny

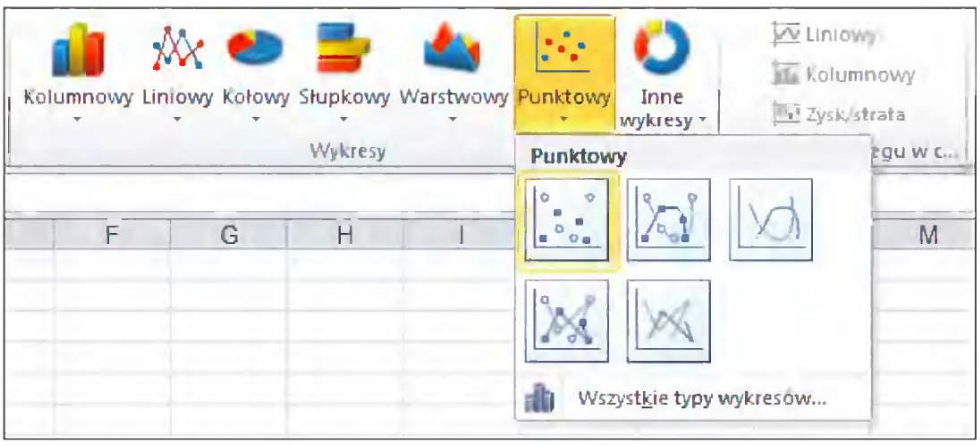
W niniejszej instrukcji opisane zostały podstawowe kwestie dotyczące tworzenia wykresów w arkuszu kalkulacyjnym pakietu Office. Omówienie wszystkich rodzajów wykresów i związanych z nimi opcji wykracza poza ramy tego opracowania. Szczególna uwaga skupiona została na poprawnym tworzeniu wykresów pod kątem opracowania wyników pomiarów lub obliczeń statystycznych wraz z opisem osi, ustawieniem linii siatki czy tworzenie linii trendu. Wiele opracowań studenckich a także poważnych publikacji zawiera wykresy, które zostały utworzone w Excelu i pozostawione bez żadnego formatowania. Celem tego ćwiczenia jest zaprezentowanie czytelnikowi, w jaki sposób dokonać podstawowego formatowania wykresu w celu poprawienia jego czytelności i estetyki.

Rysunek 5.1 przedstawia pewne dane eksperymentalne, które będziemy chcieli przedstawić w postaci wykresu punktowego.

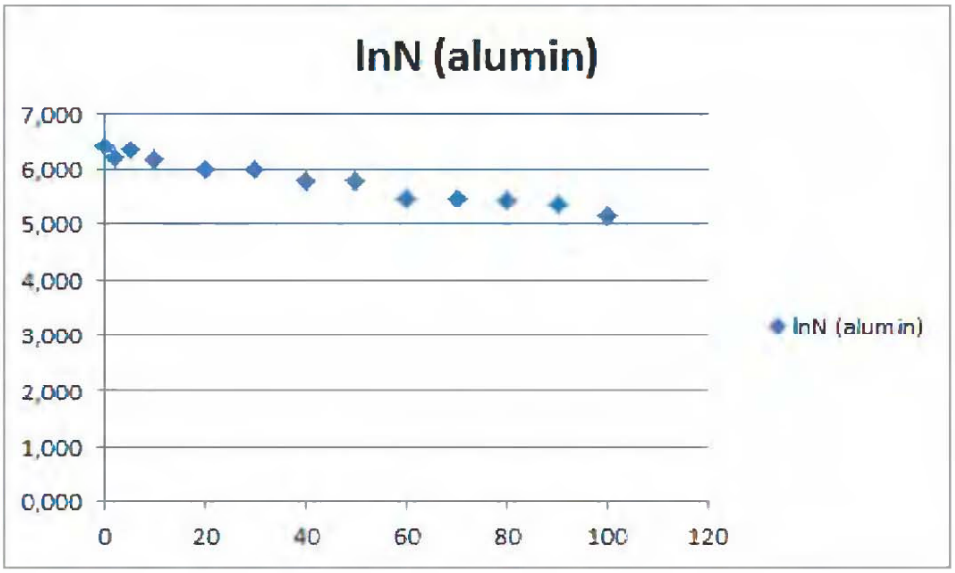
	A	B
1	X[mm]	lnN (alumin)
2	0	6,405
3	2	6,215
4	5	6,337
5	10	6,174
6	20	6,014
7	30	6,009
8	40	5,793
9	50	5,799
10	60	5,464
11	70	5,481
12	80	5,425
13	90	5,375
14	100	5,136
15		

Rys. 5.1. Przykładowe dane pomiarowe

W celu utworzenia wykresu punktowego z danych dwukolumnowych (a także, gdy kolumn jest więcej) w pierwszej kolejności należy zaznaczyć dane. W tym przykładzie będą to wiersze od 1 do 14 w kolumnach A i B. Następnie przechodzimy do zakładki *Wstawianie* i wybieramy interesujący nas wykres – *punktowy tylko ze znacznikami* (rysunek 5.2). Rezultat wykonania operacji przedstawiony został na rysunku 5.3.



Rys. 5.2. Karta *Wykresy* w zakładce *Wstawianie*



Rys. 5.3. Domyślny sposób prezentacji wykresu w arkuszu kalkulacyjnym

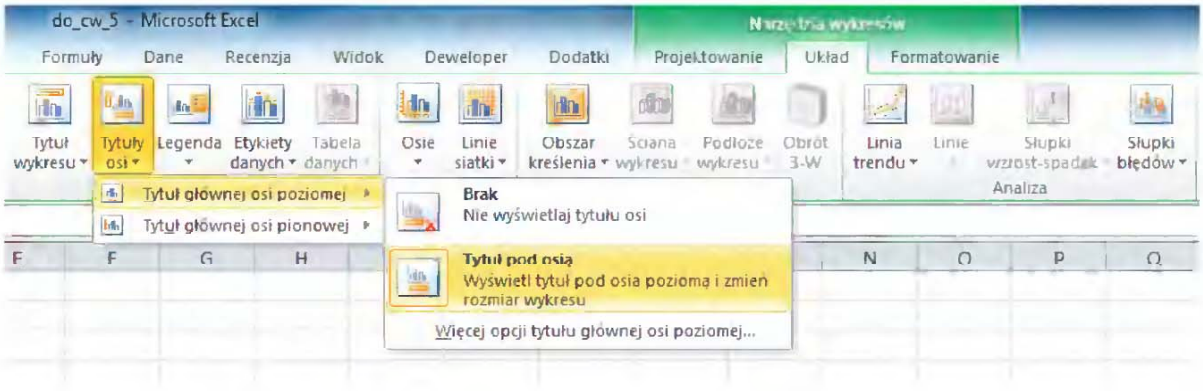
Jak widać na rysunku 5.3 domyślnie utworzony wykres wymaga uzupełnienia. Dokonamy modyfikacji następujących elementów wykresu:

- podpis osi poziomej i pionowej,
- pogrubienie osi poziomej i pionowej,
- zakończenie osi strzałką,
- pionowe i poziome linie siatki:
  - kolor,
  - grubość i rodzaj linii,
- skala osi pionowej.

Po utworzeniu wykresu w menu głównym programu pojawiają się trzy nowe zakładki *Narzędzia wykresów*:

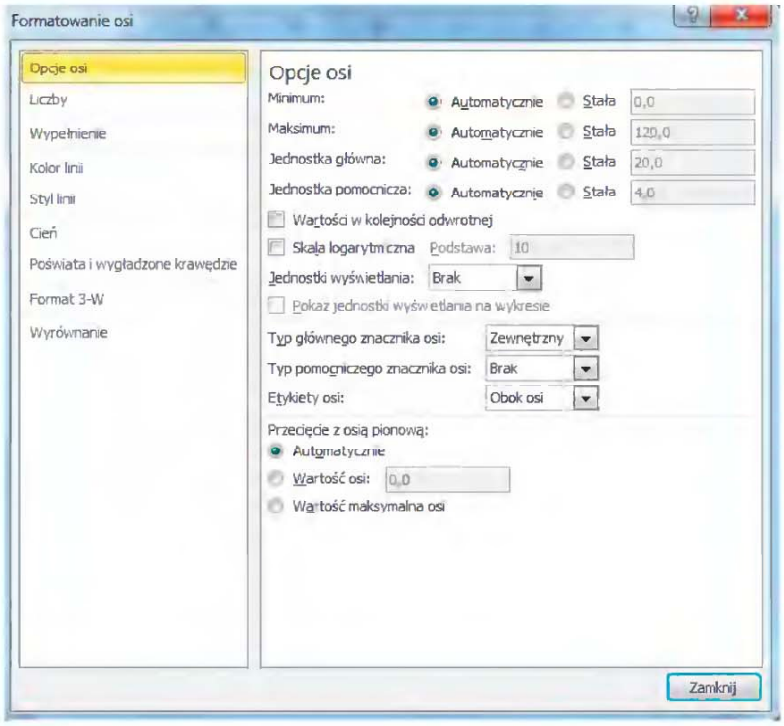
- Projektowanie
- Układ
- Formatowanie

Rysunek 5.4 przedstawia zakładkę *Układ*, zawierającą narzędzia do formatowania wyglądu wykresu. Idąc od lewej strony znajdziemy tu opcje związane z tytułem wykresu, tytułami osi. Dalej znajdujemy opcje formatowania osi i linii siatki.



Rys. 5.4. Menu *Narzędzia wykresów* z kartami zawierającymi funkcje formatujące

W pierwszej kolejności chcemy wykonać podpisy osi poziomej i pionowej. Wybieramy Tytuły osi →Tytuł osi poziomej → Tytuł pod osią i wpisujemy tytuł dla osi poziomej z klawiatury. Podobnie postępujemy w przypadku osi pionowej. Następnie pogrubimy osie oraz zakończymy je strzałkami. W tym celu wybieramy okno Formatowanie *osi* (rysunek 5.5): *Osie → Głowna oś pozioma → Więcej opcji głównej osi poziomej*.

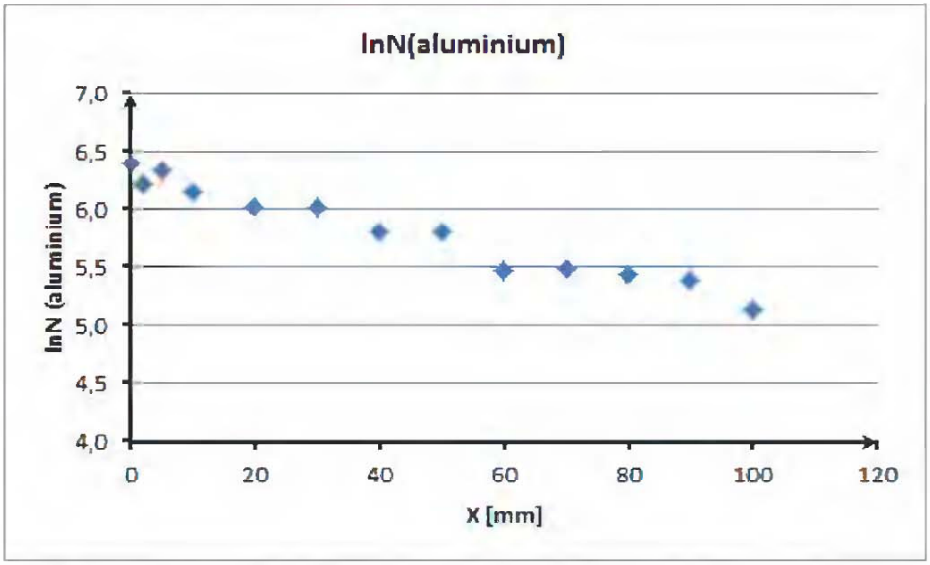


Rys. 5.5. Okno formatowania osi

Poniżej przedstawiono krótki opis najważniejszych zakładek:

- *Opcje osi* – opcje znajdujące się w tej zakładce umożliwiają dostosowanie skali osi, możemy również wybrać przecięcie z osią poziomą (dla ustawień osi pionowej) oraz pionową (dla ustawień osi poziomej).
- *Liczby* – zakładka, która umożliwia wybór formatu etykiet osi (liczba, tekst, waluta) i dostosowanie formatu dziesiętnego liczb
- *Kolor linii* – zakładka, która umożliwia dostosowanie koloru linii
- *Styl linii* – zakładka, która umożliwia ustawienia grubości linii osi, rodzaju linii a także typu zakończenia

Uwzględniając powyższy opis, dokonujemy formatowania osi zgodnie z rysunkiem 5.6 (porównaj z rysunkiem 5.3).



Rys. 5.6. Rezultat formatowania osi. Tytuł, etykiety osi, grubość linii, rodzaj zakończenia osi

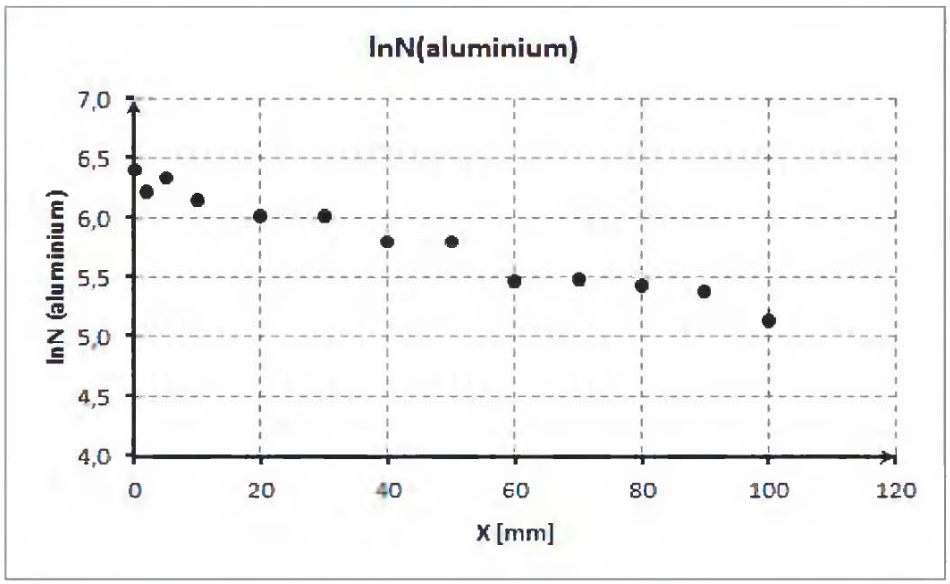
W następnej kolejności zajmiemy się liniami siatki. Dodamy podstawowe poziome i pionowe linie siatki. Ustawimy rodzaj linii na przerywany i kolor linii – czarny. W tym celu wybieramy

Układ→Podstawowe poziome linie siatki→Więcej opcji podstawowych poziomych linii siatki

Ukazuje się wówczas okno *Formatowanie głównych linii siatki*, które jest bardzo podobne do okna przedstawionego na rysunku 5.5. Odpowiednich ustawień dokonamy w zakładkach:

- kolor linii,
- styl linii,

które zawierają identyczne opcje jak opcje linii osi. Zmienimy również styl znacznika. Zaznaczamy znaczniki kursorem a następnie klikając prawym klawiszem wywołujemy menu podręczne, w którym wybieramy *Formatuj serię danych*. W zakładkach *Opcje znaczników i Wypełnienie znacznika* wybieramy odpowiednie ustawienia. Ostateczny wygląd wykresu przedstawiony został na rysunku 5.7.



Rys. 5.6. Ostateczny wygląd wykresu po formatowaniu (por. z 5.3)

W następnym przykładzie pokażemy w jaki sposób utworzyć wykres z wybranych kolumn arkusza. Podobnie jak poprzednio będzie to wykres punktowy. Dane przedstawione zostały na rysunku 5.7.

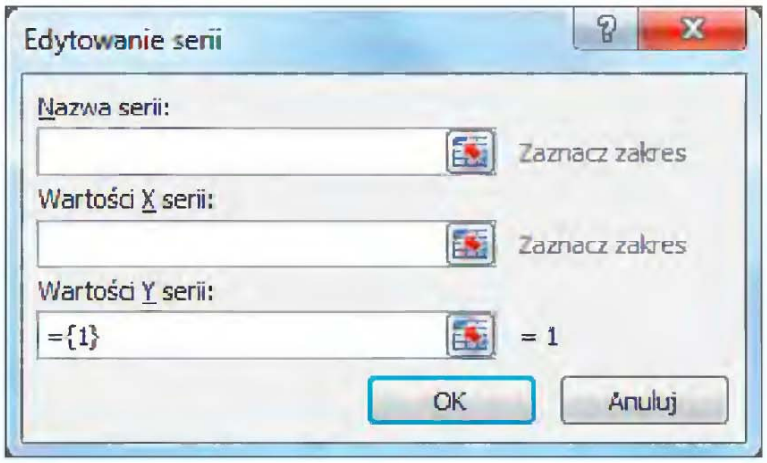
	A	B	C	D	E
1	x(mm)	1	2	3	średnia
2	0	960	978	963	967
3	0,03	802	807	803	804
4	0,06	689	737	682	703
5	0,09	625	698	650	658
6	0,12	614	650	588	617
7	0,15	534	547	546	542
8	0,2	474	521	483	493
9	0,25	398	423	371	397
10	0,33	317	346	284	316
11	0,44	202	209	193	201
12	0,57	114	93	118	108
13	0,69	82	67	85	78
14	0,84	78	76	65	73
15	1,04	79	61	66	69
16	1,26	83	60	63	69
17	1,52	77	68	75	73

Rys. 5.7. Przykładowe dane pomiarowe, na podstawie których wykonany zostanie wykres

Tym razem nie zaznaczamy żadnych danych, wybieramy natomiast wykres punktowy do wstawienia:

Wstawianie → Wykres: Punktowy

Następnie z zakładki *Projektowanie* wybieramy *Zaznacz dane*. Pojawi się wówczas okno, w którym wybieramy *Dodaj*. Pojawi się okno pokazane na rysunku 5.8.



Rys. 5.8. Okno edycji serii danych

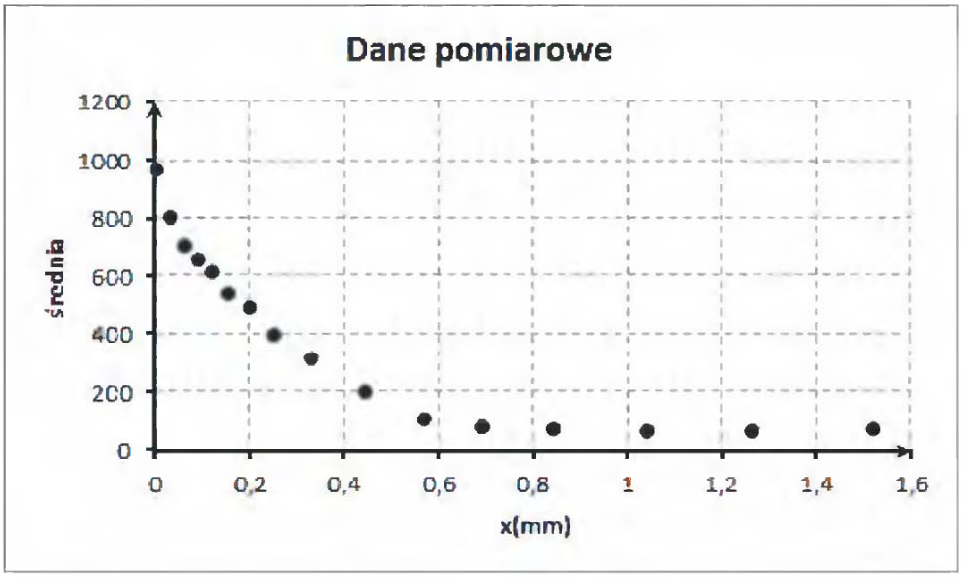
W oknie edycji zaznaczamy wartości danych *x* (oś pozioma) oraz *y* (oś pionowa). Dla danych przedstawionych w tabeli z rysunku 5.7 formuły w polach edycyjnych będą wyglądać następująco:

Wartości X serii:
=Arkusz1!$A$2:$A$17

Wartości Y serii:
=Arkusz1!$E$2:$E$17

Na koniec wprowadzamy nazwę serii.

Czytelnik może dokonać formatowania wykresu w podobny sposób do opisanego poprzednio. Ostateczny kształt wykresu przedstawia rysunek 5.9.

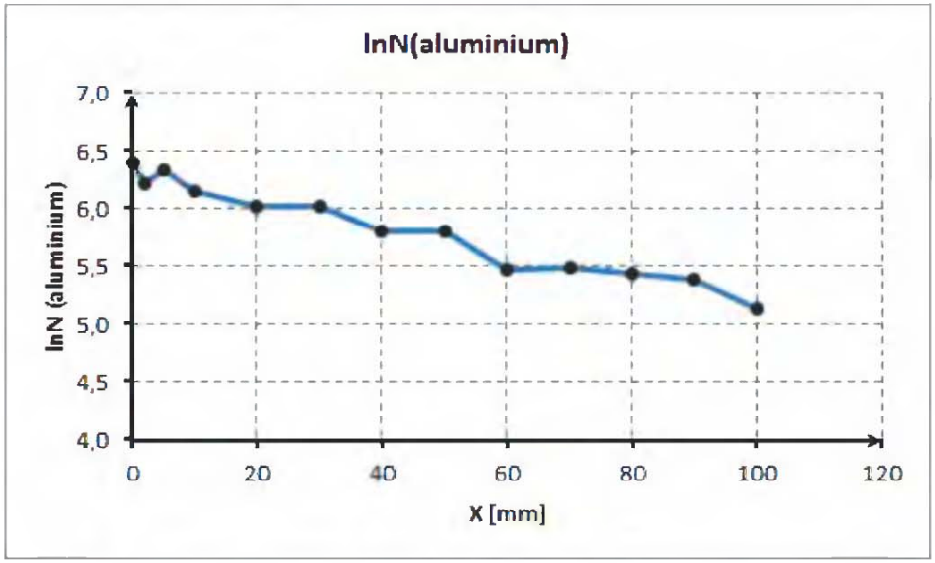


Rys. 5.9. Wykres utworzony w oparciu o dane z tabeli przedstawionej na rysunku5.7

Przedstawiony rodzaj wykresu, a także sposób formatowania to jedynie przykład możliwości pakietu Excel w tym zakresie. Zachęcam do rozszerzenia wiedzy o inne rodzaje wykresów poprzez samodzielne sprawdzanie oferowanych opcji.

Na zakończenie instrukcji wrócimy do wykresu przedstawionego na rysunku 5.6. Bardzo często tego rodzaju wykres jest wykresem danych otrzymanych w wyniku pomiarów bezpośrednich lub pośrednich. I równie często jest on rysowany poprzez połączenie punktów

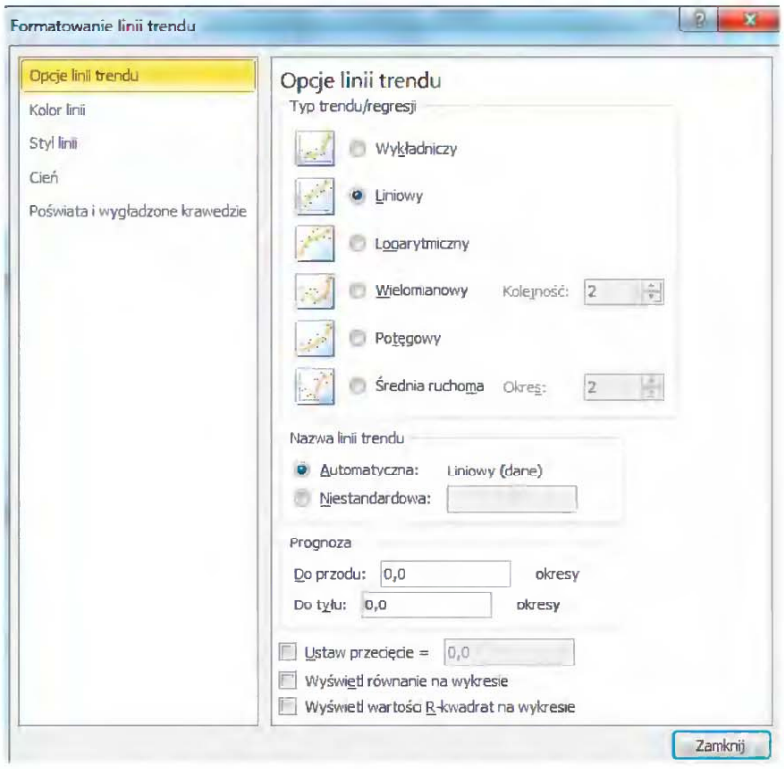
w sposób przedstawiony na rysunku 5.10. Od razu należy podkreślić, że w tym przypadku nie jest to poprawny sposób rysowania wykresu.



Rys. 5.10. Niepoprawny sposób prezentacji wyników pomiarowych

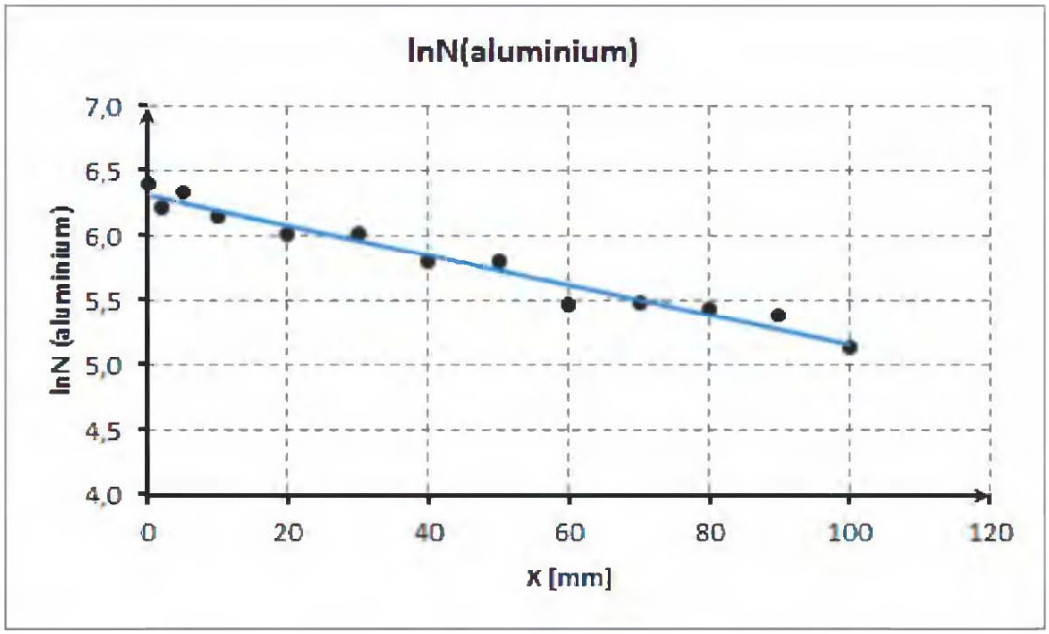
Z wykresu punktowego widać bowiem, że w przedstawionym zakresie danych punkty układają się wzdłuż prostej. W związku z tym zamiast łączyć punkty, należy wstawić linię trendu (rysunek 5.11):

Narzędzia wykresów → Układ → Linia trendu → Więcej opcji linii trendu



Rys. 5.11. Okno formatowania linii trendu

Wykres z linią trendu przedstawia poniższy rysunek:



Rys. 5.12. Wykres punktowy z linią trendu. Typ regresji: liniowa

Zadania do wykonania

Zadanie 1.

Poniższa tabela przedstawia pewne dane pomiarowe. Wykonać wykres punktowy danych. Sformatować wykres tak, jak to opisano w instrukcji. Dodać linię trendu.

X[mm]	lnN
0	6,405
2	6,211
5	6,312
10	5,883
15	5,743
25	5,778
30	5,587
35	5,030
40	5,153
45	4,990

Zadanie 2.

Narysować wykres funkcji dla $x \in < -5; 5 >$ z krokiem 0.5.

$$f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

Literatura

1. Walkenbach J. Microsoft Excel 2010PL. Biblia. Gliwice, Helion, 2011.

Część 2

Podstawy programowania

6. Podstawowe instrukcje języka C++. Struktura programu

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Kompilator języka C/C++

Wstęp teoretyczny

Co to jest programowanie? Programowanie to zdolność komunikacji z komputerami w zrozumiałym dla nich języku, z wykorzystaniem gramatyki i składni w celu wykonania użytecznych działań [1].

Do podstawowych pojęć związanych z programowaniem możemy zaliczyć:

- Problem algorytmiczny
- Algorytm
- Implementacja
- Kod źródłowy
- Program komputerowy

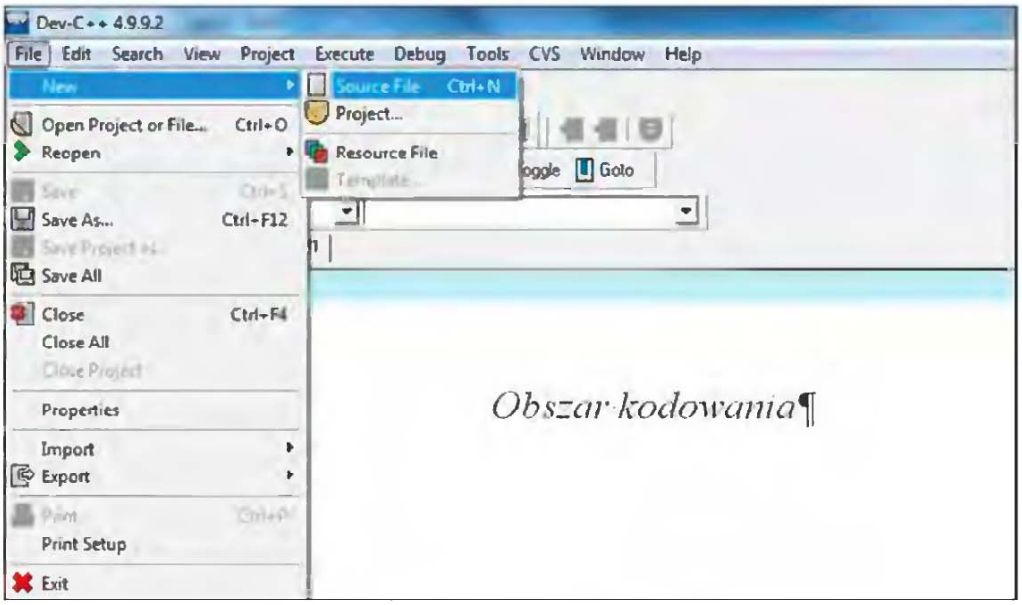
Algorytm to zapis np. w postaci listy kroków lub schematu blokowego, który przedstawia wszystkie operacje konieczne do realizacji problemu algorytmicznego tj. zadania o zadanej specyfikacji. Na przykład algorytm rozwiązania równania kwadratowego. Programy komputerowe tworzy się w języku programowania. Język programowania to specjalny język charakteryzujący się właściwą składnią, za pomocą którego według ściśle określonych zasad tworzy się kod źródłowy opisujący zadanie, które ma wykonać komputer.

W procedurze tworzenia programu komputerowego możemy wyróżnić następujące etapy: analiza problemu, określenie rozwiązania, kodowanie, testowanie programu. W pierwszej fazie programowania należy przede wszystkim zdefiniować specyfikację zadania programistycznego, tj. określić jakiego rodzaju danych program będzie oczekiwał od użytkownika a także jakie dane program będzie produkował jako wynik swojego działania. Następnie definiuje się rozwiązanie problemu krok po kroku najczęściej w postaci schematu blokowego. W fazie trzeciej następuje właściwe tworzenie kodu programu zgodnie z projektem algorytmu przedstawionym w poprzedniej fazie rozwiązania. Mamy tu do czynienia z implementacją różnych kompozycji algorytmicznych: sekwencyjną, warunkową czy iteracyjną. Bardzo często przy tworzeniu programu pomija się dwie pierwsze fazy projektowania i od razu przechodzi do tworzenia kodu. W przypadku prostych programów może to być zasadne jednak dla bardzo złożonych problemów konieczne jest stworzenie projektu programu w celu zwiększenia czytelności i przejrzystości rozwiązań. Ostatnią fazą jest weryfikacja działania programu poprzez test wszystkich kombinacji zmiennych na wejściu programu. Tabela 6.1 przedstawia procentowo czas programisty poświęcony poszczególnym fazom tworzenia programu.

Tabela 6.1. Czas poświęcony poszczególnym fazom tworzenia programu [2]

Faza	Czas
Analiza problemu	10%
Określenie rozwiązania	20%
Kodowanie	20%
Testowanie programu	50%

Rysunek 6.1 przedstawia okno kompilatora Dev C++ w wersji 4.9.9.2 rozpowszechnianego na licencji GENERAL PUBLIC LICENSE. Wybierając *File -> New -> Source File* ukazuje się okno, w którym znajdować się będzie właściwa część kodu.



Rys. 6.1. Okno główne kompilatora C/C++

Strukturę programu możemy prześledzić na pierwszym przykładzie, którego listing zaprezentowano na rysunku 6.2. Zadaniem programu będzie wyświetlenie na ekranie komputera napisu: „Pierwszy program".

```
1  # include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7   // główna część programu
8
9   cout<<"Pierwszy program"<<endl;
10
11  system ("pause");
12  return 0;
13
14 }
```

Rys. 6.2. Pierwszy program komputerowy. Struktura programu

Jak widać na rysunku 6.2 główna część programu znajduje się wewnątrz funkcji głównej o nazwie *main* pomiędzy klamrami – otwierającą '{' i zamykającą '}'. Zawarte w bloku instrukcje wykonywane są kolejno począwszy od pierwszej do ostatniej. W przedstawionym przykładzie instrukcja znajdująca się w linii nr 9 :

cout<<"Pierwszy program"<<endl;

odpowiada za wyświetlenie komunikatu na ekranie komputera. Polecenie endl oznacza koniec linii (enter). W linii nr 1 znajduje się przykład tzw. dyrektywy preprocesora, czyli specjalnej instrukcji, która powoduje dołączanie bibliotek w trakcie kompilacji programu.

W tym przypadku są to biblioteki odpowiedzialne za obsługę wejścia-wyjścia. Inne, często używane dyrektywy, to:

#include<cmath>- biblioteka matematyczna
#include<string> - biblioteka obsługująca łańcuchy

Bibliotekę *cmath* należy dołączyć, jeżeli chcemy korzystać z funkcji matematycznych np. funkcji pierwiastek, potęga, funkcji trygonometrycznych itp.

Wyrażenie *using namespace std* znajdujące się w 3. Linii kodu mówi kompilatorowi, że będziemy używać standardowej przestrzeni nazw. Linia 7. zawiera przykład komentarza. Wszystko, co znajduje się w linii po znakach // jest pomijane w procesie kompilacji. Jeżeli chcemy wstawić komentarz składający się z większej liczby linii, to wstawiamy znaki /* na początek komentarza i */ na koniec.

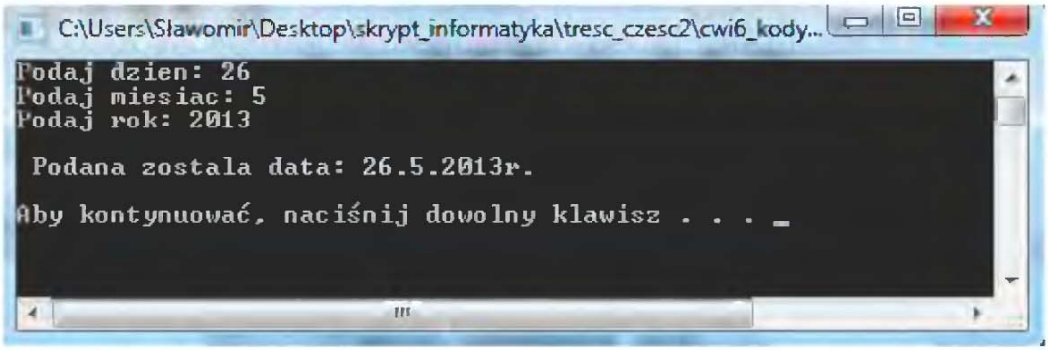
```
// pierwszy przykład komentarza
/*drugi przykład komentarza
  ta linia  też będzie niewidoczna dla kompilatora*/
```

Polecenie w linii 11 odpowiada za zatrzymanie na ekranie monitora okna z wynikiem działania programu. W linii 12 znajduje się polecenie *return 0* i oznacza, że funkcja główna *main* zwraca wartość *0*.

Pierwszy przykład pokazał w jaki sposób wyprowadzić tekst na ekran komputera. W dalszej części będziemy chcieli wprowadzić liczbę z klawiatury. Załóżmy, że będą to trzy liczby całkowite, oznaczające rok urodzenia, miesiąc oraz dzień. Do wprowadzenia liczby (i nie tylko) z klawiatury komputera służy polecenie: *cin*. Zanim jednak go użyjemy należy wcześniej zdefiniować typ zmiennej, tj. określić jakiego rodzaju będą to liczby: całkowite (*int*) czy rzeczywiste (tzw. zmiennoprzecinkowe – *float*). Należy również ustalić nazwy tych zmiennych. Przykład użycia zmiennych całkowitych oraz sposób wprowadzania zmiennych z klawiatury pokazuje rysunek 6.3. Linie 7–9 zawierają definicję zmiennych typu całkowitego o nazwach *rok, miesiąc, dzien.* Przykład wczytywania zmiennych z klawiatury przedstawiony został w wierszach 12, 14, 16. W linii 18 pojawia się wyrażenie '\n'. Oznacza znak nowej linii, podobnie jak polecenie endl. Należy również zwrócić uwagę na znak średnika ';' występujący po zakończeniu danej instrukcji. Wynik działania programu przedstawiony został na rysunku 6.4.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7   int rok;
8   int miesiac;
9   int dzien;
10
11  cout<<"Podaj dzien: ";
12  cin>>dzien;
13  cout<<"Podaj miesiac: ";
14  cin>>miesiac;
15  cout<<"Podaj rok: ";
16  cin>>rok;
17
18  cout<<"\n Podana zostala data: "<<dzien<<"."<<miesiac<<"."<<rok<<"r.\n"<<endl;
19
20  system("pause");
21  return 0;
22 }
23
```

Rys. 6.3. Przykład definiowania zmiennych i wczytywania danych z klawiatury



Rys. 6.4. Rezultat działania programu z listingu przedstawionego na rysunku 6.3

Tabela 6.2 zawiera nazwy wybranych typów reprezentujących obiekty całkowite, rzeczywiste (zmiennoprzecinkowe) oraz znakowe i łańcuchowe.

Tabela 6.2. Typy fundamentalne

Nazwa typu	Opis
int	Typ całkowity
float	Typ rzeczywisty (zmiennoprzecinkowy)
char	Typ znakowy
string	Typ łańcuchowy (konieczna jest Instrukcja *#include<string>*)

Wiadomo już w jaki sposób definiować zmienne oraz nadawać im określoną wartość z klawiatury. Działania na obiektach wykonywane są z użyciem operatorów. Tabela 6.3 przedstawia podstawowe operatory używane w tworzeniu programów. Są to operatory *dwuargumentowe*, co oznacza, że działają na dwóch obiektach znajdujących się po lewej i prawej stronie operatora.

Tabela 6.3. Podstawowe operatory wykorzystywane w języku C/C++

Nazwa typu	Opis
*	mnożenie: *a*b*
/	dzielenie: *a*/*b*
+	dodawanie: *a*+*b*
-	odejmowanie: *a*-*b*
%	reszta z dzielenia: *a*%*b* (liczby *a* i *b* muszą być typu całkowitego)
=	przypisanie: *pi*=3.14;

Przy tworzeniu wyrażeń czy formuł z użyciem operatorów należy pamiętać o priorytetach. Spośród operatorów z tabeli 6.3 najwyższy priorytet mają operatory *, /, %. W następnej kolejności są +, -. W przypadku złożonych wyrażeń należy używać nawiasów okrągłych ( ), np.

$(a+b+0.12)/c - 7.5*(13.1+s)+0.51$

Należy również zwrócić uwagę na fakt, że liczby zmiennoprzecinkowe wprowadzamy z **kropką** a nie przecinkiem jak w Excelu.

Operator przypisania = powoduje, że do obiektu stojącego po jego lewej stronie przypisana (podstawiona) zostaje wartość wyrażenia stojącego po prawej. I tu uwaga. Jeżeli po lewej stronie operatora mamy zmienną typu *int* , a po prawej typu *float* , to przypisanie nastąpi tylko w części całkowitej. Oznacza to, że następujące przypisanie:

```
int x=2.51;
```

w rezultacie spowoduje, że do zmiennej x zostanie przypisana liczba 2. Rozważmy jeszcze następujące przypisanie:

float x=1/2;

Należy zwrócić uwagę, że po prawej stronie mamy operator / który w tym przypadku działa na dwie liczby całkowite. Wynikiem tego działania będzie liczba 0, która zostanie przypisana do zmiennej *x*. W rezultacie pomimo, że zdefiniowaliśmy zmienną *x* jako rzeczywistą, nie zostanie do niej przypisana wartość 0.5. Prawidłowy zapis z użyciem operatora dzielenia, to:

float x=1.0/2.0;

albo po prostu

float x=0.5;

Zadania do wykonania

Zadanie 1.

Napisać program, który pobierze od użytkownika temperaturę w stopniach Celsjusza, a następnie zamieni tę temperaturę na stopnie Fahrenheita, zgodnie ze wzorem:

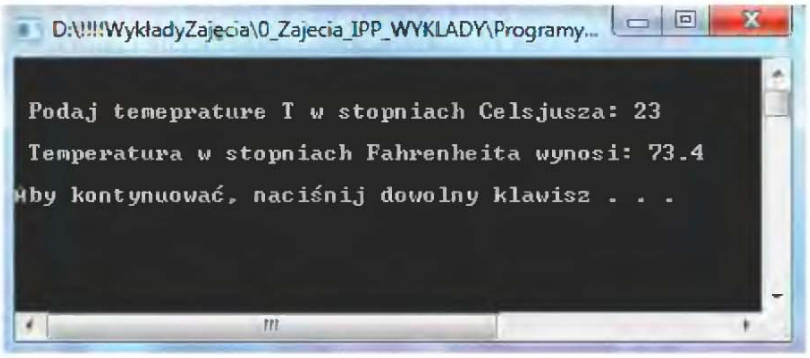
$$T[°\mathrm{F}] = T[°\mathrm{C}] * \left(\frac{9.0}{5.0}\right) + 32.0$$

Wzór kodu programu pokazany został na rysunku 6.5.

```
1
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      float T_C;      // temepratura w stopniach Celsjusza
9      float T_F;      // temperatura w stopniach Fahrenheita
10
11     cout<<"\n\n Podaj temeprature T w stopniach Celsjusza: ";
12     cin>>T_C;
13
14     //obliczamy temperature w stopniach Fahrenheita
15
16     T_F=T_C*(9.0/5.0)+32.0;
17
18     //wyswietalmy wynik
19
20     cout<<"\n Temperatura w stopniach Fahrenheita wynosi: "<<T_F<<endl;
21
22     system("PAUSE");
23     return 0;
24 }
```

Rys. 6.5. Przykładowy kod programu do przeliczania temperatury

Rezultat działania programu przedstawia rysunek 6.6.



Rys. 6.6. Rezultat działania programu z listingu przedstawionego na rysunku 6.5

Zadanie 2.

Wzorując się na przykładzie z zadania 1 napisać program, który pobierze od użytkownika temperaturę w stopniach Celsjusza, a następnie zamieni tę temperaturę na Kelviny, zgodnie ze wzorem:

$$T[K] = T[°C] + 273.15$$

Zadanie 3.

Napisz program, który oblicza i wyświetla napięcie na wyjściach dwu obwodów oraz sumę tych napięć. Napięcie na pierwszym obwodzie należy policzyć ze wzoru [3]:

$$U1 = \frac{150\,V}{0.38\,f},$$ gdzie f=*144* Hz oraz V=1.2V

Napięcie na drugim obwodzie dane jest wzorem:

$$U2 = \frac{230V}{\sqrt{56^2+(0.98f)^2}},$$ gdzie f=*100* Hz oraz V=2.3V

Zadanie4.

Energia fotonu może być obliczona z następującego wzoru [3]:

$E = \frac{hc}{\lambda}$, gdzie h=6.6256·10^{-34} J/s oraz c=299 792 458 *m/s*.

Napisz program liczący energię światła fioletowego o długości 5.9·10^{-6} m.

Literatura

1. Kingsley-Hughes A., Kingsley-Hughes A. Programowanie. Od podstaw. Gliwice, Helion, 2005.
2. Gręboszl J. Symfonia C++. Tom I . Kraków, Oficyna Kallimach, 1999.
3. Bronson G. C++ For Engineers & Scientists. Boston, Cengage Learning, 2010.

7. Implementacja warunkowej kompozycji algorytmicznej

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Kompilator języka C/C++

Wstęp teoretyczny

Instrukcja warunkowa, to instrukcja postaci:

Jeżeli *Q*, ***to*** *wykonaj polecenie A*, ***w przeciwnym razie*** *wykonaj polecenie B*

Jak widać wykonanie określonego polecenia jest uzależnione od spełnienia warunku *Q*. Jeżeli warunek *Q* jest spełniony, tzn. jest *prawdą*, to wykonuje się polecenie A. Jeżeli warunek nie jest spełniony tzn. zwraca wartość *fałsz*, wykona się polecenie B.

Implementację warunku w języku C/C++ wykonuje się za pomocą polecenia ***if .. else ..***

Najprostszy warunek będzie miał postać:

```
if (wyrażenie) instrukcja1;
```

W pierwszej kolejności obliczana jest wartość wyrażenia w nawiasie. Jeżeli jest ona niezerowa (*prawda*), to wykonywana jest *instrukcja1*. Jeżeli natomiast wartość w nawiasie jest zerowa (*fałsz*), instrukcja ta nie jest wykonywana.

Kolejna postać warunku:

```
if (wyrażenie) instrukcja1;
else           instrukcja2;
```

Podobnie jak w poprzednim przykładzie wykonanie odpowiedniej instrukcji zależy od wartości wyrażenia obliczonego w nawiasie. Jeżeli jest ono niezerowe (*prawda*) , wówczas wykona się *instrukcja1*. W przeciwnym razie (*fałsz*) wykona się *instrukcja2*. W przypadku tej konstrukcji mamy do czynienia z wyborem dwuwariantowym. Za pomocą *if .. else ...* można również dokonać konstrukcji wyboru wielowariantowego:

```
if (wyrażenie1) instrukcja1;
elseif (wyrażenie2) instrukcja2;
elseif (wyrażenie3) instrukcja3;
else                instrukcja4;
```

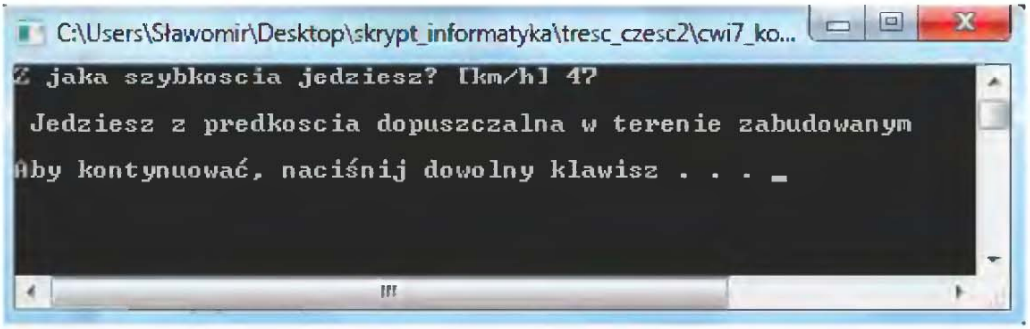
W przypadku, gdy chcemy, żeby nie tylko pojedyncza instrukcja ale cały blok instrukcji był wykonywany warunkowo, należy zastosować klamry:

```
{
    Instrukcja1;
    Instrukcja2;
    Instrukcja3;
}
```

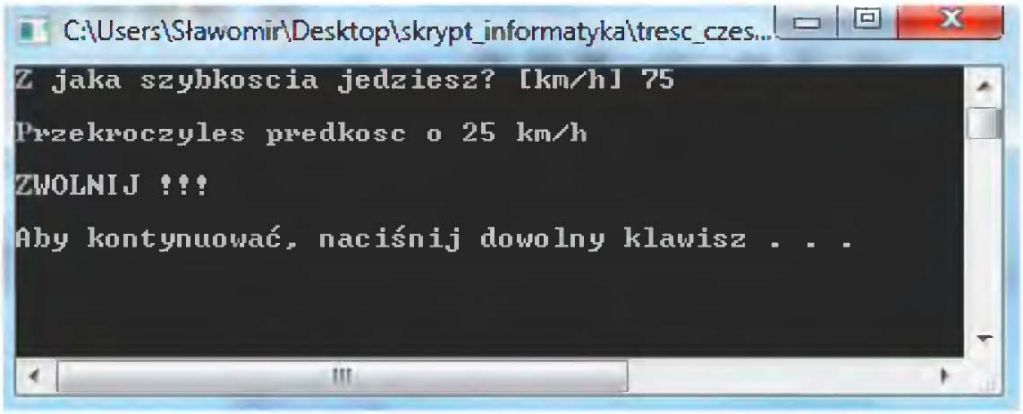
```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     float V;
8     float Vgr=50.0; //km/h
9
10    cout<<"Z jaka szybkoscia jedziesz? [km/h] ";
11    cin>> V;
12
13    if (V<Vgr)
14         cout<<"\n Jedziesz z predkoscia dopuszczalna w terenie zabudowanym\n\n";
15    else
16      {
17          cout<<"\nPrzekroczyles predkosc o "<<V-Vgr<<" km/h\n"<<endl;
18          cout<<"ZWOLNIJ !!!\n\n";
19      }
20
21    system("PAUSE");
22    return 0;
23 }
```

Rys. 7.1. Przykład implementacji warunku w języku C/C++

Rysunek 7.1 przedstawia listing programu, w którym znajduje się kompozycja warunkowa. Zadaniem programu jest ocena, czy użytkownik jedzie z dozwoloną szybkością obowiązującą w terenie zabudowanym czy nie. Szybkość graniczna zdefiniowana została w linii nr 8. Użytkownik wprowadza z klawiatury szybkość, z jaką aktualnie porusza się i w zależności od wartości wyrażenia obliczonego w nawiasie, wyświetli się odpowiedni komunikat (rysunki 7.2 i 7.3).



Rys. 7.2. Wynik działania programu w sytuacji, gdy warunek jest prawdziwy



Rys. 7.3. Wynik działania programu w sytuacji, gdy warunek nie jest prawdziwy

Należy zwrócić uwagę, że w wyrażeniu po *if* występuje tylko jedna instrukcja, podczas gdy wyrażenie *else* zawiera już dwie instrukcje. W tym drugim przypadku konieczne, więc było zastosowanie klamry.

W wyrażeniu warunkowym bardzo często stosuje się operatory relacji oraz operatory logiczne. Postać tych operatorów oraz ich opis znajduje się w tabeli 7.1.

Tabela 7.1. Operatory logiczne i operatory relacji w C/C++

Operator	Opis
<	mniejszy niż
<=	mniejszy lub równy
>	większy niż
>=	większy lub równy
==	równy
!=	różny
!	negacja
&&	koniunkcja (AND – i)
||	alternatywa (OR – lub)

Do konstrukcji złożonych wyrażeń logicznych zaleca się stosowanie nawiasów okrągłych.

Kolejny przykład (rysunek 7.4) przedstawia sposób realizacji wyboru wielowariantowego z wykorzystaniem instrukcji *if .. elseif ... else ...*

```
8   int ktory;
9
10  cout<<"Wybierz liczbe od 1 do 3: "<<endl;
11
12  cout<<"1. Dodawanie"<<endl;
13  cout<<"2. Odejmowanie"<<endl;
14  cout<<"3. Mnozenie"<<endl;
15
16  cin>> ktory;
17
18  cout<<"\nWybrales zadanie nr: "<<ktory <<"\n";
19
20  if (ktory==1)
21  {
22     cout<<"Wybrales 1 wiec bedziemy wykonywac dodawanie"<<endl;
23  }
24  else if (ktory==2)
25  {
26     cout<<"Wybrales 2 wiec bedziemy wykonywac odejmowanie"<<endl;
27  }
28  else if (ktory==3)
29  {
30     cout<<"Wybrales 3 wiec bedziemy wykonywac mnozenie"<<endl;
31  }
32  else
33  {
34     cout<<"Nie wybrales zadnej liczby z zakresu 1 - 3"<<endl;
35  }
36
```

Rys. 7.4. Przykład realizacji wyboru wielowariantowego z użyciem instrukcji *if ... elseif .. else*

Wybór wielowariantowy można również zrealizować używając instrukcji *switch*. Postać instrukcji przedstawia się następująco:

```
switch(wyrażenie)
{
        case 1:
                instrukcja1;
                break;
        case 2:
                instrukcja2;
                break;
        case 3:
                instrukcja3;
                break;
        default:
                instrukcjaN;
                break;
}
```

Jeżeli wartość wyrażenia przyjmuje wartość danej etykiety *case* wówczas wykonuje się instrukcja od miejsca etykiety aż do instrukcji *break*. W przypadku, gdy wyrażenie nie przyjmuje żadnej z wartości etykiet, wykonuje się instrukcja znajdująca się w etykiecie *default*. Rysunek 7.5 przedstawia dokładnie to samo zadanie, co wcześniej wykonane za pomocą instrukcji *switch*.

```
6  int main()
7  {
8      int ktory;
9      cout<<"Wybierz liczbe od 1 do 3: "<<endl;
10
11     cout<<"1. Dodawanie"<<endl;
12     cout<<"2. Odejmowanie"<<endl;
13     cout<<"3. Mnozenie"<<endl;
14
15     cin>> ktory;
16     cout<<"\nWybrales zadanie nr: "<<ktory <<"\n";
17
18     switch (ktory)
19     {
20       case 1:
21            cout<<"Wybrales 1 wiec bedziemy wykonywac dodawanie"<<endl;
22            break;
23       case 2:
24            cout<<"Wybrales 2 wiec bedziemy wykonywac odejmowanie"<<endl;
25       case 3:
26            cout<<"Wybrales 3 wiec bedziemy wykonywac mnozenie"<<endl;
27       default:
28            cout<<"Nie wybrales zadnej liczby z zakresu 1 - 3"<<endl;
29     }
30
31     system("PAUSE");
32     return EXIT_SUCCESS;
33 }
```

Rys. 7.5. Przykład realizacji wyboru wielowariantowego z użyciem instrukcji switch

Zadania do wykonania

Zadanie 1.

Wzór Herona: $A = \sqrt{\left[s(s-a)(s-b)(s-c)\right]}$, gdzie $s = \frac{(a+b+c)}{2}$

pozwala obliczyć pole (A) trójkąta, jeśli znane są długości *a, b, c* jego boków.

Napisz program, który:

- Pobierze od użytkownika długości odcinków *a, b, c*,
- Obliczy wartość ***s***,
- Obliczy wartość ***u=s(s – a)(s - b)(s - c)***,
  - Jeżeli wartość ***u>0*** – obliczy pole A ze wzoru,
  - w przeciwnym razie wyświetli komunikat, że z podanych długości *a, b, c* nie można utworzyć trójkąta.

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  float a,b,c;
  float s,u,A;

   cout<<"Podaj dlugosci a,b,c: "<<endl;
   cin>>a;
   cin>>b;
   cin>>c;

   s= (a+b+c)/2;
   u=s*(s-a)*(s-b)*(s-c);

    if (u>0)
      {
            A=sqrt(u);
            cout<<"Pole wynosi: "<<A<<endl;
      }
    else
        cout<<"Z podanych odcinkow nie mozna zbudowac trojkata"<<endl;

  system ("pause");
  return 0;
}
```

Rys. 7.6. Wzór rozwiązania do zadania 1

Zadanie2.

Napisać program, który wczytuje długości trzech odcinków *a, b, c* i odpowiada na pytanie czy z tych odcinków można zbudować trójkąt oraz jeżeli tak, to jakiego rodzaju trójkąt powstanie:

- prostokątny,
- ostrokątny,
- rozwartokątny.

I etap zadania – sprawdzenie czy z odcinków a,b,c można zbudować trójkąt.

W tym celu należy skorzystać z definicji, która mówi, że z danych trzech odcinków można zbudować trójkąt, jeżeli suma dwóch krótszych odcinków będzie dłuższa od najdłuższego odcinka. Problem sprowadza się więc do zamiany wartości zmiennych *a, b, c* tak, aby jedna ze zmiennych, np. *c* zawierała wartość najdłuższego odcinka. Później pozostaje już tylko sprawdzenie warunku *(a+b)>c*.

Poniższy fragment kodu pokazuje w jaki sposób zamienić miejscami zmienne *a* i *b:*

```
if (a>b)
        {
                temp=a;
                a=b;
                b=temp;
        }
```

Analogicznie należy sprawdzić i ewentualnie przestawić ze sobą zmienne *b* i *c*. Należy również zwrócić uwagę, że do zamiany miejscami potrzebna jest dodatkowa zmienna (temp), która pełni rolę zmiennej pomocniczej, przechowującej na chwilę wartość zmiennej *a*. Gdy mamy pewność, że wartość najdłuższego odcinka zapisana jest pod zmienną *c*, sprawdzamy, czy z podanych długości możemy zbudować trójkąt. Poniższy fragment kodu realizuje to zadanie.

```
if (c>a+b)
        cout<<"Z podanych odcinkow możemy zbudowac trojkat";
else
        cout<<"Z podanych odcinkow nie można zbudowac trojkata";
```

Podsumowując, kod z I etapu zadania będzie sprowadzał się do następujących kroków:

1. deklaracja/definicja zmiennych
2. wczytanie długości odcinków z klawiatury komputera
3. sprawdzenie czy *a>b* i ewentualna zamiana miejscami, podobnie dla przypadku *b>c*;
4. sprawdzenie czy *c>a+b* i wyświetlenie odpowiedniego komunikatu

II etap zadania – sprawdzenie jakiego rodzaju trójkąt powstanie.

II etap zadania sprowadza się do uzupełnienia kodu utworzonego w etapie I o moduł sprawdzający jakiego rodzaju trójkąt powstanie z podanych odcinków. W tym celu należy sprawdzić następujące nierówności:

- $c^2 = a^2 + b^2$- trójkąt prostokątny
- $c^2 < a^2 + b^2$- trójkąt ostrokątny
- $c^2 > a^2 + b^2$- trójkąt rozwartokątny

Możemy zatem obliczyć wartość wyrażenia $c^2 - a^2 - b^2$ i wynik obliczeń porównać do 0 stosując instrukcję wyboru wielowariantowego, tak jak poniżej:

```
wyr=c*c-a*a-b*b;

if (wyr==0)
        cout<<"Trojkat jest prostokątny."<<endl;
else if (wyr>0)
        cout<<"Trojkat jest rozwartokątny."<<endl;
else
        cout<<"Trojkat jest ostrokątny."<<endl;
```

Czytelnikowi pozostawiam wykonanie całości kodu do zadania2.

Literatura

1. Gręboszj. Symfonia C++. Tom 1 . Kraków, Oficyna Kallimach, 1999.

8. Iteracja. Znajdowanie miejsc zerowych funkcji

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Kompilator języka C++

Wstęp teoretyczny

Iteracja, czyli powtarzanie procedury wykonania danej instrukcji w zależności od wartości wyrażenia warunkowego, może być realizowana w języku C++ za pomocą trzech konstrukcji:

- for
- while
- do ... while ...

To, której instrukcji użyjemy będzie zależało od rozpatrywanego problemu. W ćwiczeniu nr 3 części I skryptu omówione zostały dwa rodzaje iteracji: warunkowa i ograniczona. W pierwszym przypadku pętla wykonywana jest dopóty, dopóki wyrażenie warunkowe tej pętli zwraca wartość *prawda*. W iteracji ograniczonej instrukcje wykonują znaną z góry liczbę obrotów. W myśl tego podziału, instrukcję *for* wygodniej jest stosować do rozwiązywania problemów iteracyjnych ograniczonych. Natomiast *while* i do ... *while* ... do problemów warunkowych. Należy podkreślić, że jest to wyłącznie kwestia wygody programisty. Można bowiem z instrukcji *for* zrobić pętlę warunkową a z *while* ograniczoną. W dalszej części omówione zostaną wymienione wyżej konstrukcje iteracyjne.

Pętla for

```
for (instr_init; wyraz_warunk; instr_kroku)
{
   Instrukcja 1;
      Instrukcja 2;
}
```

Wyjaśnienie poszczególnych części pętli:

- *instr_init* – instrukcja wykonywana zanim pętla zostanie po raz pierwszy uruchomiona, często jest to inicjalizacja zamiennej kroku lub przypisanie wartości startowej, np. int i=0;
- *wyraz_warunk* – wyrażenie warunkowe, które obliczane jest przed każdym obiegiem pętli, w przypadku, gdy wartość wyrażenia jest niezerowa, to wykonywane są instrukcje zawarte w pętli; np.: i<10;
- *instr_kroku* – instrukcja wykonywana na zakończenie każdego obiegu pętli, np. i=i+1;

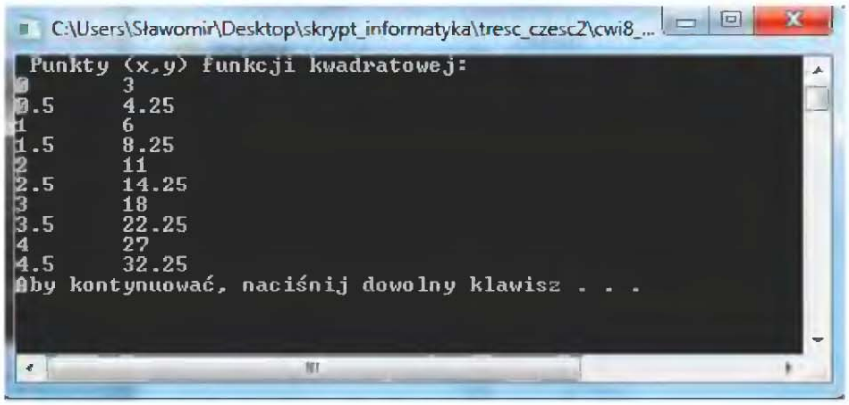
Instrukcja *instr_init* nie musi być tylko jedną instrukcją. Jeżeli chcemy wstawić kilka instrukcji na początek pętli, należy je wówczas oddzielić przecinkiem.

Listing przedstawiony na rysunku 8.1 jest przykładem implementacji pętli do zadania tabelaryzacji funkcji kwadratowej $f(x)=x^2+2x+3$.

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      float a,b,c;
8      float x=0.0, y;
9
10     a=1.0;
11     b=2.0;
12     c=3.0;
13
14     cout<<" Punkty (x,y) funkcji kwadratowej:"<<endl;
15
16     for (int i=0;i<10;i++)
17     {
18      y=a*x*x+b*x+c;
19      cout<<x<<"\t"<<y<<endl;
20      x=x+0.5;
21     }
22
23     system("PAUSE");
24     return 0;
25 }
26
```

Rys. 8.1. Przykład programu z użyciem pętli *for*

Wiersze 16 – 21 zawierają implementację pętli *for*. Z warunku pętli widać, że wykonuje się ona 10 razy, od i=0 do 9 (włącznie). Instrukcja i++ to tzw. operator *inkrementacji*, zwiększający zmienną *i* o 1. Jest tożsamy z wyrażeniem i=i+1. Innym, bardzo często wykorzystywanym w programowaniu operatorem, jest operator *dekrementacji* i--, zmniejszający wartość zmiennej *i* o 1 (i=i-1). Rezultat działania programu przedstawionego na rysunku 8.1 przedstawiony został na rysunku 8.2.



Rys. 8.2. Wynik działania programu z użyciem pętli for

Pętla while(...)

Pętla *while* ma postać:

```
while (wyrażenie)
        Instrukcja1;
```

Działanie pętli przedstawia się następująco. W pierwszej kolejności obliczane jest wyrażenie w nawiasie okrągłym. Jeżeli wartość wyrażenia jest *prawdą*, to wykonuje się instrukcja1, po czym obliczana jest ponownie wartość wyrażenia. Jeżeli w pętli znajduje się więcej niż jedna instrukcja należy użyć nawiasów { }. Pętla kończy swoje działanie jeżeli wyrażenie w nawiasie przyjmuje wartość 0 (*fałsz*). Przykładem zastosowania pętli *while* jest zadanie, w którym należy obliczyć wartości NWD dwóch liczb dodatnich za pomocą algorytmu Euklidesa (rysunek 8.3).

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7    int a,b;
8    int kopia_a, kopia_b;
9    int r;      //reszta z dzielenia
10
11   cout<<"Podaj liczby: "<<endl;
12   cin>>a;
13   cin>>b;
14
15   kopia_a=a;
16   kopia_b=b;
17
18   while(b>0)
19   {
20        r=a%b;
21        a=b;
22        b=r;
23   }
24
25   cout<<"NWD liczb "<<kopia_a<<" i "<<kopia_b<<" wynosi: "<<a<<endl<<endl;
26
27   system("pause");
28   return 0;
29 }
```

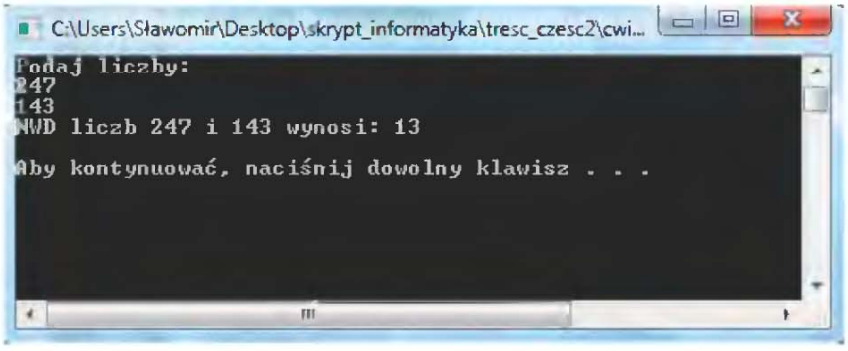
Rys. 8.3. Implementacja algorytmu Euklidesa obliczania wartości NWD dwóch liczb całkowitych

Pętla *while* znajduje się w liniach 18–23. Jak widać obroty wykonywane są dopóty, dopóki wartość zmiennej *b* jest większa od zera. W linii nr 20 obliczana jest reszta z dzielenia (operator %) *a* przez *b* i wynik działania przypisywany jest do zmiennej *r*. Ponieważ chcemy wyświetlić z jakich liczb liczony jest NWD, a przypisania w liniach 21 i 22 powodują zmianę wartości zmiennych *a* i *b* , wykonana została kopia wprowadzonych w linii 12 i 13 wartości.

Tabela 8.1 przedstawia wartości zmiennych *a* i *b* oraz *r*, w każdym obrocie pętli. Wynik działania programu przedstawia rysunek 8.4.

Tabela 8.1. Wartości przyjmowane przez zmienne w kolejnych obrotach pętli

a	*b*	*r*
247	143	104
143	104	39
104	39	26
39	26	13
26	13	0
13	0	



Rys.8.4. Rezultat działania programu do obliczania NWD

Pętla do ... while ...

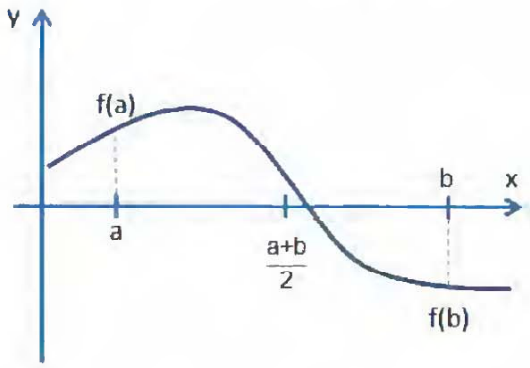
Kompozycja iteracyjna bardzo podobna do omówionej poprzednio pętli *while(...)*. Zasadnicza różnica w działaniu obydwu pętli polega na tym, że w pętli *while* warunek sprawdzany jest przed wykonaniem instrukcji. W pętli *do .. while* w pierwszej kolejności wykonywana jest instrukcja, a dopiero potem sprawdzany jest warunek.

Forma pętli jest następująca:

```
do
 {
        Instrukcja1;
        Instrukcja2;
}while(wyrażenie);
```

Blok instrukcji wykonywany jest wówczas, gdy wyrażenie w nawiasie przyjmuje wartość *prawda*. Należy więc zwrócić uwagę, że pętla wykona się zawsze co najmniej jeden raz.

W ćwiczeniu nr 3 omówiona została jedna z iteracyjnych metod poszukiwania miejsc zerowych funkcji. Implementacji dokonano w arkuszu kalkulacyjnym Excel. W dalszej części tej instrukcji przedstawimy metodę połowienia przedziału – inny przykład znajdowania miejsc zerowych funkcji z użyciem własnego programu komputerowego. Rysunek 8.5 przedstawia schematycznie wykres funkcji w przedziale <a,b>.



Rys. 8.5. Schematyczna ilustracja metody połowienia przedziału

Omówimy ideę metody. W pierwszej kolejności należy wybrać dwie wartości *a* i *b*, które leżą po przeciwnych stronach szukanego miejsca zerowego, a więc takie, które spełniają następującą nierówność:

$$f(a)\cdot f(b)<0 \tag{14}$$

Następnie podany przedział <*a,b*> dzielony jest na połowę zgodnie z następującym wyrażeniem:

$$c=\frac{a+b}{2} \tag{15}$$

Należy teraz sprawdzić, po której stronie leży wartość *c* i dokonać zawężenia przedziału. Sprawdzamy zatem następujące nierówności:

$$f(a)\cdot f(c)<0 \tag{16}$$

$$f(c)\cdot f(b)<0 \tag{17}$$

Jeżeli pierwsza nierówność (16) jest prawdziwa, wówczas oznacza to, że wartości *a* i *c* leżą po przeciwnych stronach miejsca zerowego i zawężamy przedział zmieniając jego koniec od strony *b* , tj. wykonujemy następujące przypisanie:

$$b=c \tag{18}$$

Analogicznie postąpimy jeżeli nierówność (17) jest prawdziwa. Przypisanie wówczas będzie miało postać:

$$a=c \tag{19}$$

Kryteria zakończenia iteracji:

- zakładając, że *c* jest poszukiwanym miejscem zerowym sprawdzamy, czy *f(c) <eps*, gdzie przez *eps* rozumiemy dokładność z jaką chcemy znaleźć miejsce zerowe (np. *eps*=0.0001).
- maksymalna liczba iteracji

Możliwe jest również połączenie obydwu powyższych kryteriów. Pozwoli to na zmniejszenie czasu oczekiwania na wynik w przypadku, gdy kryterium dokładności nie pozwala na zakończenie pętli.

Zadania do wykonania

Zadanie 1.

Znaleźć miejsce zerowe funkcji zadanej wzorem z dokładnością ε=0.001 [1]:

$$f(x)=x^2 + 2x - 15$$

a=2.8, b=3.1

Kod programu przedstawia rysunek 8.6. W linii nr 2 znajduje się wyrażenie dołączające do programu bibliotekę matematyczną. Wykorzystana zostanie zdefiniowana w tej bibliotece funkcja *fabs* (linia 37), obliczająca wartość bezwzględną z liczby. Wynik działania programu pokazuje rysunek 8.7.

Zadanie 2.

Znaleźć miejsce zerowe funkcji zadanej wzorem z dokładnością ε=0.00001 [1]:

$$f(x) = \sin(x)\, e^x$$

a=8.0, b=10.0

Wskazówka: wykorzystać funkcje: *sin(x)* i *exp(x)*z biblioteki *<cmath>*.

Zadanie3.

Znaleźć miejsce zerowe funkcji zadanej wzorem z dokładnością ε=0.001 [1]:

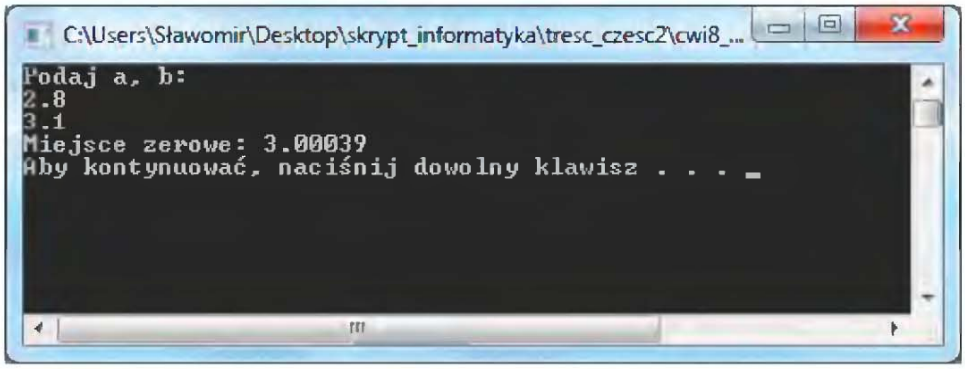
$$f(x) = tg(x) \cdot \sqrt{x}$$

a=8.0, b=10.0

Wskazówka: wykorzystać funkcje: *sqrt(x)* i *tan(x)*z biblioteki *<cmath>*.

```cpp
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
  double a,b,c;
  double fa, fb, fc; //wartości funkcji f(a), f(b) i f(c)
  double eps=0.01;

  cout<<"Podaj a, b:"<<endl;
  cin>>a;
  cin>>b;

  fa=a*a+2*a-15;
  fb=b*b+2*b-15;

  if (fa*fb<0)
  {
       do
         {
            c=0.5*(a+b);
            fc=c*c+2*c-15;

            if (fa*fc<0)
              {  b=c;
                 fb=fc;
              }
            else
              {  a=c;
                 fa=fc;
              }
            //  cout<<"c: "<<c<<endl;
            //  cout<<"fc: "<<fc<<endl;

         }while(fabs(fc)>eps);
    c=0.5*(a+b);
    fc=c*c+2*c-15;
    cout<<"Miejsce zerowe: "<<c<<endl;
    //cout<<"fc: "<<fc<<endl;
  }
  else
  {
    cout<<"Blad danych wejsciowych!"<<endl;
  }


  system("pause");
  return 0;

}
```

Rys.8.6. Kod programu do zadania 1



Rys.8.7. Rezultat działania programu z zadania 1

Literatura

1. Bronson G. C++ For Engineers & Scientists. Boston, Cengage Learning, 2010.
2. Gręboszj. Symfonia C++. Tom I . Kraków, Oficyna Kallimach, 1999.

9. Operacje na tablicach. Sortowanie

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Kompilator języka C++

Wstęp teoretyczny

Tablice umożliwiają przechowywanie grupy zmiennych w pamięci komputera. Grupa ta charakteryzuje się określonym typem oraz nazwą. Na przykład chcemy przechować w tablicy liczby. Definiujemy wówczas tablicę:

```
int liczby[10];
```

Nawias kwadratowy oznacza, że mamy do czynienia z tablicą. W przypadku powyższej definicji tablica ma 10 elementów typu całkowitego. Oczywiście analogicznie wyglądać będzie definicja dla innych typów:

```
float liczby[10];  char znaki[10];
```

Poszczególne elementy tablicy, to:

liczby[0] liczby[1] liczby[2] liczby[3] ... liczby[9]

Należy zwrócić uwagę, że **numeracja elementów tablicy zaczyna się od zera.** Dostęp do elementów tablicy odbywa się w taki sam sposób jak w przypadku zwykłych zmiennych. Jeżeli chcemy zapisać liczbę do elementu tablicy, to wpiszemy polecenie:

```
liczby[2]=123;
```

Poniższe polecenie spowoduje wczytanie liczby z klawiatury do elementu tablicy o indeksie 5

```
cin>> liczby[5];
```

Polecenie, które wyświetli zawartość elementu o indeksie 5 będzie miało następującą postać:

```
cout<< liczby[5];
```

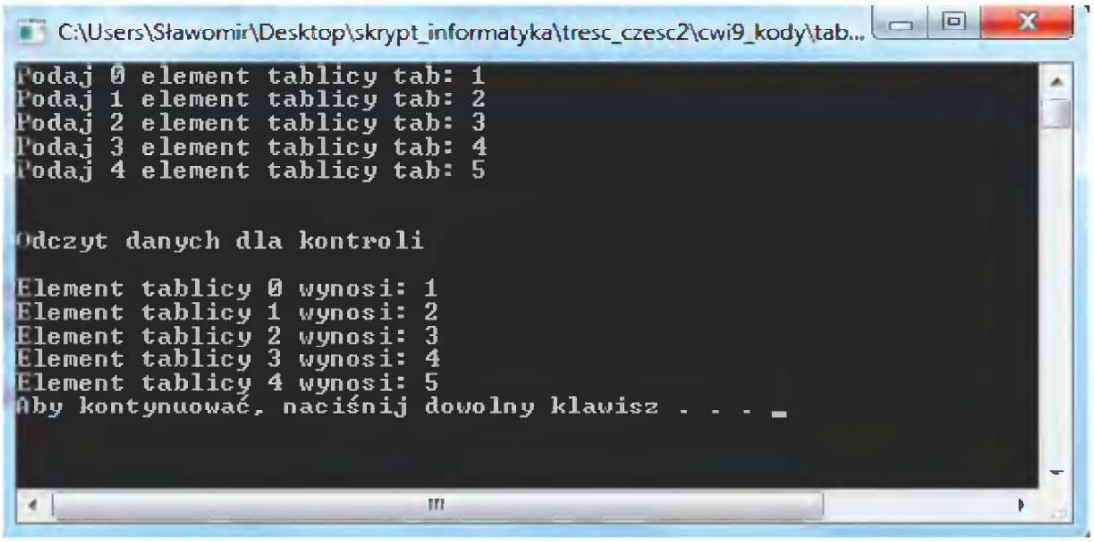
Bardzo często do obsługi tablic wykorzystuje się pętlę *for*. Na przykład, gdy chcemy wprowadzić liczby do tablicy, możemy zapisać:

```
for (int i=0; i<10; i++)
  {
    cout<<"Podaj liczbę:";
    cin>>liczby[i];
  }
```

Przykład kodu wykorzystującego tablice przedstawia rysunek 9.1. Rezultat działania programu przedstawia rysunek 9.2.

```
#include <iostream>

using namespace std;

int main()
{
   float tab[5];    //rozmiar tablicy

   //wczytujemy dane do tablicy
   for (int i=0;i<5;i++)
   {
      cout<<"Podaj "<< i <<" element tablicy tab: ";
      cin>>tab[i];
   }

   //odczytujemy z tablicy
   cout<<"\n\nOdczyt danych dla kontroli"<<endl;
   for (int i=0;i<5;i++)
   {
      cout<<"\nElement tablicy "<< i <<" wynosi: "<<tab[i];

   }

    cout<<"\n";
    system("PAUSE");
    return EXIT_SUCCESS;
}
```

Rys. 9.1. Przykład programu wczytującego dane do tablicy *tab*[]i wyświetlającego wynik



Rys. 9.2. Wynik działania programu z listingu przedstawionego na rysunku 9.1

W języku C++ istnieje również możliwość tworzenia tablic wielowymiarowych. Definicja tablicy zawierającej 3 wiersze i 2 kolumny wygląda następująco:

int macierz[3][2];

Elementy tablicy będą następujące:

macierz[0][0] macierz[0][1]

macierz[1][0] macierz[1][1]
macierz[2][0] macierz[2][1]

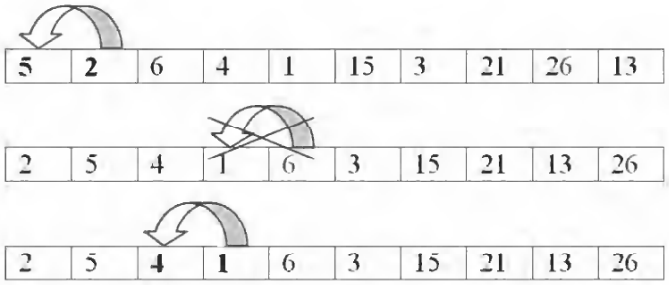
W tym przypadku należy również pamiętać o numeracji elementów zaczynającej się od 0. Do obsługi tablicy dwuwymiarowej należy użyć pętli zagnieżdżonej *for*.

```
for (int i=0; i<3; i++)
    {
for (int j=0; j<2; j++)
            cout<<macierz[i][j];
cout<<endl;
 }
```

Zadania do wykonania

Zadanie 1.

Wprowadzić do tablicy 10 liczb. Napisać program wykonujący sortowanie liczb w kolejności rosnącej, poprzez porównanie sąsiednich liczb i ewentualną zamianę miejscami w przypadku spełnienia warunku, że następna liczba jest mniejsza od rozważanej tak jak na schemacie poniżej:



Schemat działania:

1. Zdefiniować dziesięcioelementową tablicę. Rozmiar tablicy określić za pomocą polecenia :
   #define ROZMIAR 10
2. Wczytać 10 liczb do tablicy o nazwie ***tab*** i dla kontroli wyświetlić zawartość tablicy.

```
1  #include <iostream>
2  #define ROZMIAR 10
3
4  using namespace std;
5
6  int main()
7  {
8     float tab[ROZMIAR];    //rozmiar tablicy
9
10    for (int i=0;i<ROZMIAR;i++)
11    {
12       cout<<"Podaj "<< i <<" element tablicy tab: ";
13       cin>>tab[i];
14    }
15
16     cout<<"\n";
17     system("PAUSE");
18     return 0;
19 }
```

Rys. 9.3. Definicja tablicy oraz kod wczytywania liczb do tablicy

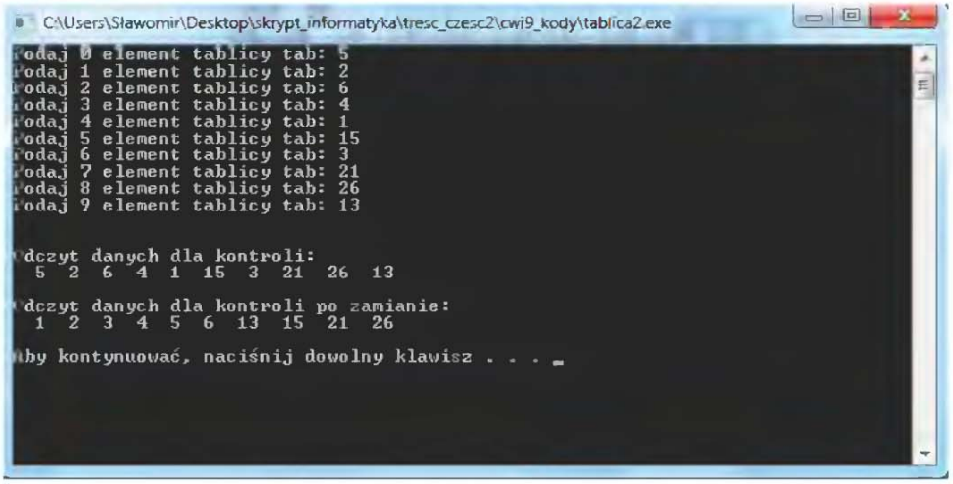
3. Użyć instrukcję warunkową ***if ... else ...***do porównania sąsiadujących w tablicy liczb. Należy zauważyć, że liczba porównań będzie mniejsza niż rozmiar tablicy.

```
24    float temp;
25    for (int i=0;i<ROZMIAR-1;i++)
26    {
27       if (tab[i]>tab[i+1])
28        {
29           temp=tab[i];
30           tab[i]=tab[i+1];
31           tab[i+1]=temp;
32        }
33
34    }
```

Rys. 9.4. Fragment kodu odpowiedzialny za zamianę liczb

4. Zmodyfikować program tak, aby powtarzał procedurę zamiany liczb miejscami aż do całkowitego posortowania tablicy.

Wynik działania programu przedstawia rysunek 9.5.



Rys. 9.5. Wynik działania programu z zadania 1

Zadanie 2.

Poniżej znajduje się kod programu losującego 6 liczb z zakresu 1 – 49 . Program korzysta z tzw. generatora liczb pseudolosowych. Funkcja *int losuj(void)* zawierająca linię *los=1+rand()%49*; odpowiada za wylosowanie liczby z pożądanego zakresu. Losowanie 6 liczb zostało zaprogramowane w głównej funkcji *main()* w taki sposób, aby wylosowane liczby, zapisywane do tablicy Lotto, nie powtórzyły się.

Wykorzystując poniższy kod napisać program, który:

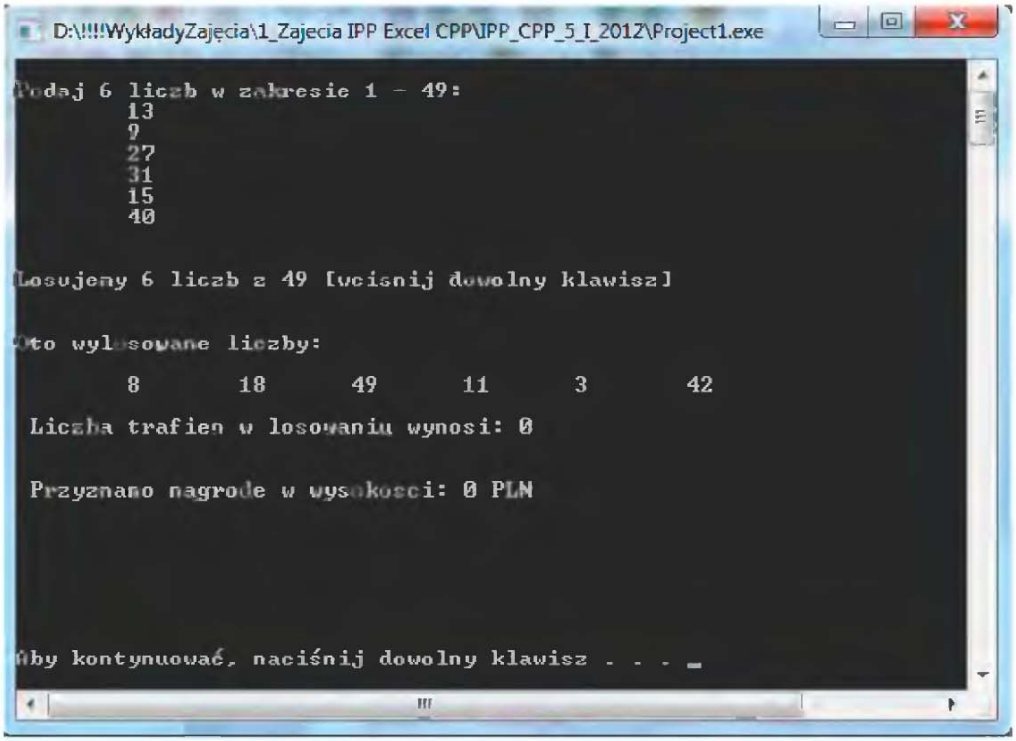
1. Pobierze od użytkownika programu 6 liczb z zakresu 1 – 49 i zapisze w tablicy *int Moje[6]*;
2. Wykona losowanie 6 liczb (kod)
3. Sprawdzi i wyświetli liczbę trafień
4. Przyzna nagrodę w zależności od liczby trafień:

a. 1 – 10 zł
b. 2 100zł
c. 3 – 1000zł
d. 4 – 10000zł
e. 5 – 100000zł
f. 6 – 1000000zł

```
#include <cstdlib>
#include <iostream>
#include <conio.h>

using namespace std;
intlosuj(void);
int main(intargc, char *argv[])
{
int los;
intMoje[6];
int Lotto[6]={0};
boolLosujPonownie=false;
intLiczbaTrafien=0;
int nagroda=0;

cout<<"\nPodaj 6 liczb w zakresie 1 - 49: "<<endl;
for (inti=0; i<6;i++)
  {
cout<<"\t";
cin>>Moje[i];
  }

cout<<"\n\nLosujemy 6 liczb z 49 [wcisnij dowolny lawisz]"<<endl;
for (inti=0; i<6;i++)
  {
    do
    {
LosujPonownic=false;
      los=losuj();
      for (int j=0; j<6;j++)
      {
        if (los==Lotto[j]) LosujPonownie=true;
      }
    }while(LosujPonownic==true);

Lotto[i]=los;
  }
cout<<"\n\nOto wylosowane liczby: \n"<<endl;
for (inti=0; i<6;i++)
  {
cout<<"\t"<<Lotto[i];
  }
cout<<"\n\n\n\n\n\n\n";
  system("PAUSE");
  return EXIT_SUCCESS;
}
//---------------------------------------------------------------
```

```
intlosuj(void)
{
int los;
 char c;

 while(1)
  {
     los=1+rand()%49;
     if(kbhit())
{
        break;
     }
  }

 return los;
}
```

Do powyższego kodu należy dopisać część sprawdzającą liczbę trafień i wyświetlającą wartość przyznanej nagrody. Rezultat działania programu przedstawiono na rysunku 9.6.



Rys. 9.6. Wynik działania programu Lotto

Literatura

1. Grębosz J. Symfonia C++. Tom I . Kraków, Oficyna Kallimach, 1999.

10. Funkcja i jej zastosowanie

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- Kompilator języka C++

Wstęp teoretyczny

Funkcje stanowią bardzo ważny element języka programowania. Pozwalają na utworzenie własnych rozwiązań, mogą być wielokrotnie wykorzystywane w różnych programach. Działanie własnych funkcji jest podobne do działania funkcji znajdujących się w bibliotekach języka, np. bibliotece matematycznej. Gdy chcemy obliczyć wartość pierwiastka ze zmiennej *x* wpisujemy w kodzie wyrażenie *sqrt(x)*. Jest to funkcja o nazwie *sqrt*, której parametrem jest zmienna *x* i która zwraca określoną wartość (wynik). Podobnie można utworzyć funkcję obliczającą obwód koła o nazwie *obwod*. Parametrem takiej funkcji będzie zmienna *r* – promień koła i funkcja ta zwracać będzie wynik. W pierwszym przypadku, do prawidłowego działania funkcji wystarczy dołączenie biblioteki *cmath*. W tym drugim programista musi utworzyć jej definicję, tzn. określić w jaki sposób funkcja ta będzie działać. Niniejsza część skryptu w głównej części poświęcona jest zasadom tworzenia programu komputerowego z wykorzystaniem funkcji. Rysunek 10.1 przedstawia przykład programu wykorzystującego funkcję.

```
1  #include <iostream>
2
3  using namespace std;
4
5  float kolo(float promien)
6  {
7       const float pi=3.14;
8       float wynik;
9
10      wynik=pi*promien*promien;
11
12      return wynik;
13
14 }
15
16 int main()
17 {
18     float r;
19
20     cout<<"Podaj promien: ";
21     cin>>r;
22
23     cout<<"\nPole kola: "<<kolo(r)<<endl;
24
25     system("PAUSE");
26     return 0;
27 }
28
```

Rys. 10.1. Listing programu obliczającego pole koła o zadanym promieniu *r*

W pierwszej kolejności należy zwrócić uwagę na strukturę programu. Do tej pory programy składały się tylko z jednej funkcji *main()*. Jak widać na rysunku powyżej funkcja obliczająca pole znajduje się na zewnątrz funkcji głównej. W języku C/C++ funkcje nie mogą być zagnieżdżane. Własne funkcje można definiować poniżej funkcji głównej, z uprzednią deklaracją funkcji, którą umieszcza się przed *main()*.

W ogólnym przypadku funkcja będzie miała następującą postać:

```
TypFunkcji  NazwaFunkcji  (Deklaracja Parametrow Funkcji)
   {
         Instrukcje Funkcji;

         return Wartosc;
   }
```

- Typ Funkcji – typ zmiennej zwracanej przez funkcję (*int, float, bool, itd.*), może być także typ *void* - wówczas funkcja nie zwraca żadnej wartości i instrukcja *return* jest pomijana. W omawianym przykładzie funkcja jest typu *float*.
- Nazwa Funkcji – nazwa funkcji: *kolo.*
- DeklaracjaParametrow Funkcji – argumenty funkcji; kolejne parametry oddzielane są od siebie przecinkiem (lista może być także pusta).
- InstrukcjeFunkcji – blok instrukcji stanowiący główną część funkcji – sposób działania funkcji (tu: wzór na pole koła o zadanym promieniu).
- return Wartosc – jeżeli funkcja nie jest typu *void* wówczas w tym miejscu wpisywana jest wartość, którą funkcja zwraca jako wynik swojego działania (wynik – obliczone pole koła).

Funkcja *float* kolo (float promien)wywoływana jest z parametrem typu *float promien* i zwraca jako rezultat liczbę typu *float.* Argument *promien* w tym przypadku przesyłany jest przez wartość. Oznacza to, że nie ulega on zmianie nawet jeżeli wewnątrz funkcji przypisana zostanie mu jakakolwiek wartość. Program w tym przypadku tworzy kopię parametru, na której pracuje. Po wyjściu z funkcji kopia jest niszczona. Przesyłanie argumentów przez referencję natomiast, pozwala funkcji na modyfikowanie zmiennych znajdujących się poza funkcją. Parametry takie deklaruje się poprzedzając je znakiem &. Przykład nagłówka funkcji z argumentami przesyłanymi przez wartość i referencję:

```
int funkcja(int argument1, int &argument2);
```

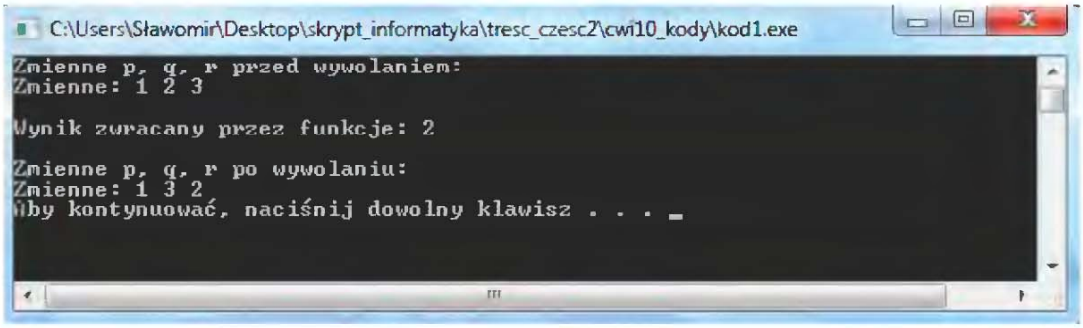
Parametr argument1 przesyłany jest przez wartość, natomiast argument2 przez referencję. Przykład programu demonstrującego przekazywanie parametrów do funkcji przedstawiono na rysunku 10.2. Wywołanie funkcji znajduje się w linii 23 kodu z rysunku 10.1. W ogólnym przypadku będzie to nazwa funkcji z parametrami, z którymi ta funkcja jest wywoływana:

```
NazwaFunkcji (ParametryPrzesyłaneDoFunkcji);
```

```
1  #include <iostream>
2  using namespace std;
3
4  float funkcja(int a, int &b, int &c);
5
6  int main()
7  {
8    int p=1;
9    int q=2;
10   int r=3;
11
12   cout<<"Zmienne p, q, r przed wywolaniem: "<<endl;
13   cout<<"Zmienne: "<<p<<" "<<q<<" "<<r<<endl;
14     cout<<"\nWynik zwracany przez funkcje: "<<funkcja(p,q,r)<<endl;
15   cout<<"\nZmienne p, q, r po wywolaniu: "<<endl;
16   cout<<"Zmienne: "<<p<<" "<<q<<" "<<r<<endl;
17
18   system("pause");
19   return 0;
20
21 }
22 float funkcja(int a, int &b, int &c)
23 {
24    a=b;
25    b=c;
26    c=a;
27
28   return (a+b+c)/3;
29 }
```

Rys. 10.2. Przykład programu pokazującego dwa sposoby przekazywania parametrów do funkcji: przez wartość i przez referencję

Wynik działania programu przedstawia rysunek 10.3.



Rys. 10.3. Rezultat działania programu

W funkcji głównej *main* zdefiniowane zostały 3 zmienne (linie 8–10). Jak widać na rysunku 10.3 wartości zmiennych *q* i *r* przekazanych do funkcji przez referencję uległy zmianie, podczas gdy *p* przekazana przez wartość nie zmieniła się. Należy ponadto zwrócić uwagę na układ programu. Funkcja zdefiniowana przez użytkownika znajduje się pod funkcją główną *main*. Konieczne jest w takim przypadku zadeklarowanie funkcji przed jej wywołaniem tj. przed funkcją *main*. Deklaracja funkcji znajduje się w linii 4.

Wiemy już jak tworzyć funkcje i przekazywać do nich argumenty. W dalszej części pokażemy w jaki sposób przekazać tablicę do funkcji. Umiejętność utworzenia własnych funkcji na potrzeby analizy danych jest wskazane w przypadku, gdy mamy do czynienia z danymi pomiarowymi, które muszą zostać przetworzone w celu uzyskania określonej informacji. Dane takie najczęściej grupowane są w tablicach. Załóżmy, że mamy tablicę liczb – wyników pomiarów (tabela 10.1 [2]).

Tabela 10.1. Przykładowe wyniki pomiarów [2]

Lp.	x_i
1	35.6
2	35.8
3	35.7
4	35.5
5	35.6
6	35.9
7	35.7
8	35.8
9	35.9
10	35.4

Pokażemy jak utworzyć funkcję, która jako rezultat zwróci nam średnią arytmetyczną danych x_i . Wykorzystamy wzór:

gdzie $n=10$.

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}, \tag{20}$$

W przypadku tablicy nie istnieje możliwość przesyłania przez wartość. Tablicę przekazuje się do funkcji podając adres zerowego elementu tej tablicy, tj. nazwę. Na przykład dla następującej funkcji:

```
float funkcja(float tablica[], int rozmiar)
{
...
}
```

wywołanie funkcji będzie miało następującą postać:

```
funkcja (tablica, 10);
```

W przedstawionym na rysunku 10.4 przykładzie użytkownik wprowadza dane w funkcji głównej, na podstawie których zostaną wykonane dalsze obliczenia. Dane te są wysyłane do funkcji, która oblicza średnią arytmetyczną i zwraca wynik. Wywołanie funkcji, znajdujące się w 22. linii kodu, składa się z dwóch argumentów: pierwszy to adres zerowego elementu tablicy, która przekazywana jest do funkcji, drugi natomiast to rozmiar tej tablicy – konieczny, jeżeli chcemy np. przeglądać całą tablicę. Rezultat działania programu dla danych przedstawionych w tabeli 10.1 pokazuje rysunek 10.5.

Program do obliczania średniej arytmetycznej pomiarów z wykorzystaniem funkcji przedstawiono na rysunku 10.4.

```
1 #include <iostream>
2 #define SIZE 10
3
4 using namespace std;
5
6 float srednia(float dane[], int n);
7
8 int main()
9 {
10     float dane_p[SIZE];
11     int n;
12
13     cout<<"Podaj liczbe danych do wprowadzenia (max 10): ";
14     cin>>n;
15
16     for (int i=0; i<SIZE; i++)
17     {
18         cout<<"Podaj wartosc: ";
19         cin>>dane_p[i];
20     }
21
22     cout<<"Srednia arytmetyczna punktow wynosi: "<<srednia(dane_p,n);
23
24     system("pause");
25     return 0;
26
27 }
28
29 float srednia(float dane[], int n)
30 {
31   float suma=0;
32
33     for (int i=0; i<n; i++)
34           suma=suma+dane[i];
35
36   return suma/n;
37 }
```

Rys. 10.4. Przykład programu pokazujący przekazywanie tablicy do funkcji

```
C:\Users\Sławomir\Desktop\skrypt_informatyka\tresc_czesc2\cwi10_kody\kod2.exe
Podaj liczbe danych do wprowadzenia (max 10): 10
Podaj wartosc: 35.6
Podaj wartosc: 35.8
Podaj wartosc: 35.7
Podaj wartosc: 35.5
Podaj wartosc: 35.6
Podaj wartosc: 35.9
Podaj wartosc: 35.7
Podaj wartosc: 35.8
Podaj wartosc: 35.9
Podaj wartosc: 35.4

Srednia arytmetyczna punktow wynosi: 35.69

Aby kontynuować, naciśnij dowolny klawisz . . .
```

Rys. 10.5. Wynik działania programu z listingu przedstawionym na rysunku 10.4

Zadania do wykonania

Zadanie 1.

W oparciu o listing z rysunku 10.4 zaprojektować program komputerowy obliczający odchylenie standardowe dla danych przedstawionych w tabeli 10.1. Program powinien składać się z trzech funkcji:

- main– zawierającej interfejs do komunikacji z użytkownikiem oraz możliwość wprowadzenia danych pomiarowych
- srednia – do obliczania średniej z danych pomiarowych wprowadzonych przez użytkownika

- odchylenie – obliczającej odchylenie standardowe

Wykorzystać funkcję do liczenia średniej arytmetycznej omówioną powyżej.

$S_x = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$

n – liczba pomiarów

$\bar{x}$ – średnia arytmetyczna

x_i – wynik pomiaru o numerze i

Zadanie 2.

Wzór Herona: $A = \sqrt{[s(s-a)(s-b)(s-c)]}$ gdzie $s = \frac{(a+b+c)}{2}$

pozwala obliczyć pole (3) trójkąta, jeśli znane są długości *a, b, c* jego boków.

Napisz program, który:

a) Pobierze od użytkownika parametry *a, b, c* ;
b) Obliczy wartość *s* oraz *u=s(s – a)(s - b)(s - c)*;
c) Jeżeli wartość *s(s – a)(s - b)(s - c)* będzie dodatnia – obliczy pole A, w przeciwnym razie wyświetli komunikat, że z podanych długości *a, b, c* nie można utworzyć trójkąta.

Zadanie należy rozwiązać, definiując funkcję o nazwie Heron. Funkcja jako wynik, będzie zwracać wartość pola A dla przypadku $u > 0$ oraz wartość -1, gdy $u < 0$.

Przykład funkcji:

```
float Heron(float a, float b, float c)
{
        float s;
        float u;
float A;
        s= (a+b+c)/2;
        u=s*(s-a)*(s-b)*(s-c);

        if (u>0)
        {
                a=sqrt(u);
                return A;
        }
        else
        {
                return -1;
        }
}
```

LITERATURA

1. Grębosz J. Symfonia C++. Tom I . Kraków, Oficyna Kallimach, 1999.
2. Szydłowski H. Pracownia fizyczna. Warszawa, PWN, 1999.

Część 3

Wprowadzenie do środowiska Matlab

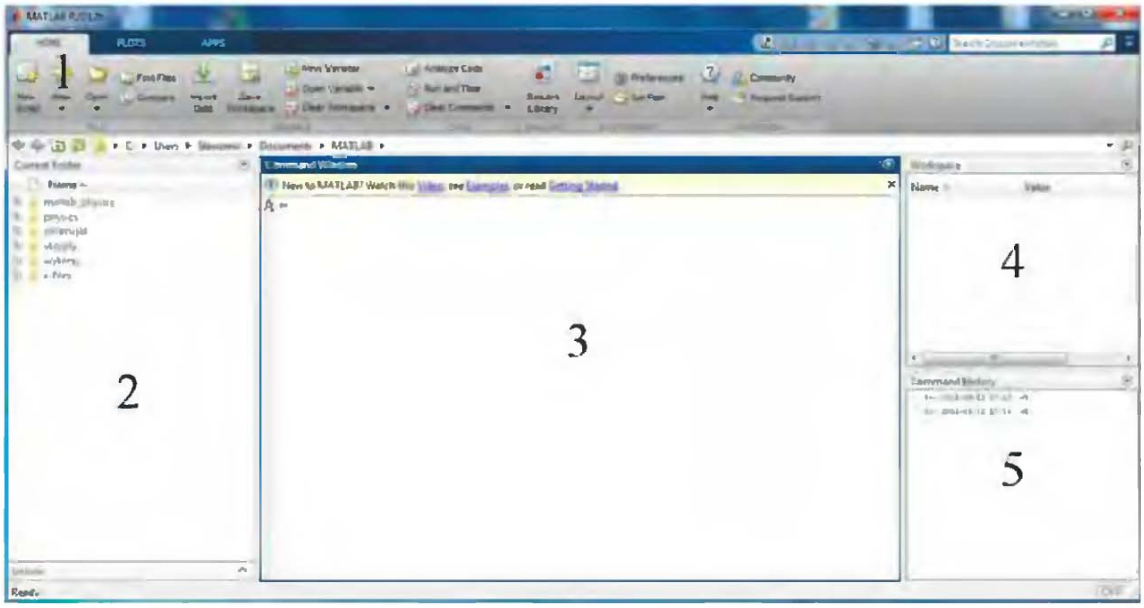
11. Macierze. Operacje na macierzach

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- System obliczeniowy MATLAB

Wstęp teoretyczny

Środowisko do obliczeń naukowo-inżynierskich MATLAB jest na chwilę obecną światowym standardem, jeżeli chodzi o sprawne wykonanie określonych obliczeń numerycznych i wizualizację ich rezultatów. Śmiało można pokusić się o stwierdzenie, że jest to system obliczeniowy przeznaczony do szerokiego spektrum zastosowań poprzez wbudowane narzędzia zwane *Toolboxami*, zawierające biblioteki i funkcje właściwe dla danej dyscypliny naukowej: od równań różniczkowych cząstkowych (*PDE Toolbox*) przez statystykę, ekonomię, biologię aż do baz danych i obsługę urządzeń zewnętrznych. Są również narzędzia do tworzenia interfejsów programu. Szczegółowe informacje można znaleźć na stronie producenta – firmy *MathWorks* [1].

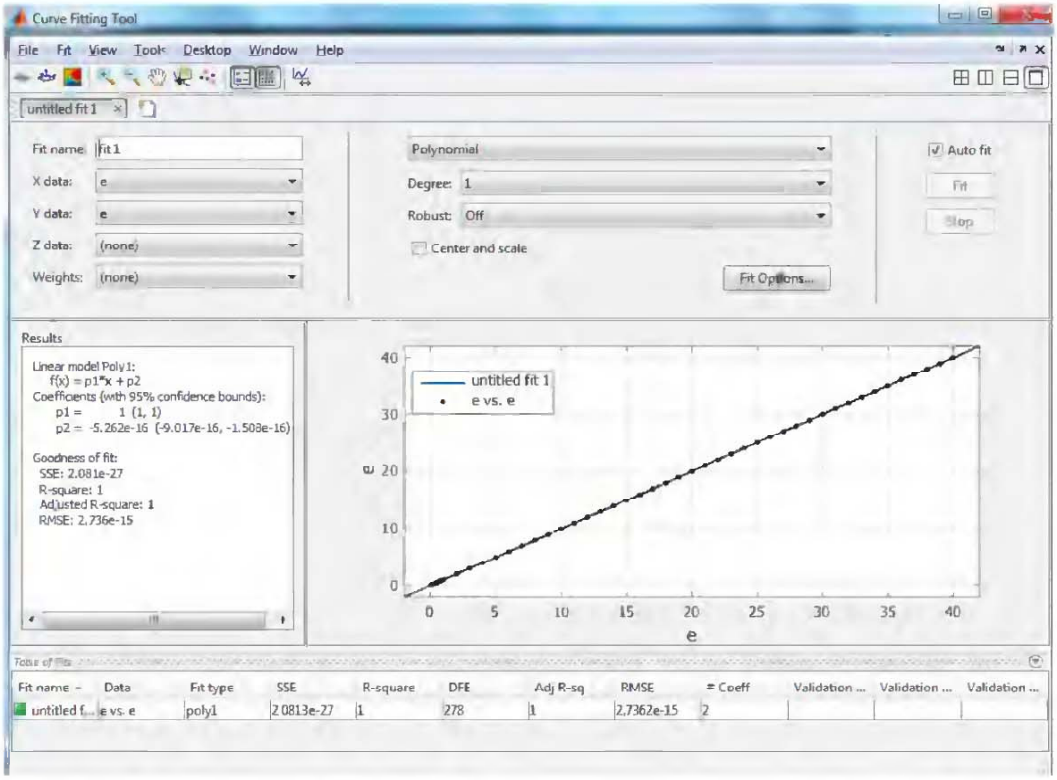


Rys. 11.1. Główne okno programu Matlab

Główne okno środowiska przedstawiono na rysunku 11.1. Numerem 1 oznaczono trzy zakładki *Home, Plots, Apps*. Pierwsza zawiera ikony uruchamiające podstawowe funkcje programu jak np. ustawienia (*Preferences*) czy ikonę tworzącą nowy skrypt (*NewScript*). W kolejnej zakładce znajduje się paleta możliwych do wyboru wykresów. Ostatnia zakładka *Apps* zawiera ikony uruchamiające dodatkowe biblioteki (tzw. *Toolboxy*) przeznaczone do specjalistycznych obliczeń. Rysunek 11.2 przedstawia zakładkę *Apps*, z trzema narzędziami: *PDE, Statistics, CurveFitting*.

Rys. 11.2. Zakładka *Apps* z *Toolboxami* do specjalistycznych obliczeń

Numerem 2 z rysunku 11.1 oznaczono okno aktywnego folderu, najczęściej jest to folder z aktualnymi projektami utworzonymi w programie, które są uruchamiane z poziomu wiersza poleceń lub skryptu. *Toolboxy* mają swoje własne interfejsy graficzne z dostępnymi funkcjami. Możliwe jest oczywiście tworzenie skryptów z wykorzystaniem bibliotek bez interfejsu graficznego. Na rysunku 11.3 pokazano moduł *CurveFitting*.



Rys. 11.3. Interfejs graficzny modułu do modelowania krzywych *CurveFittingToolbox*

Okno nr 3 to okno poleceń. Jest to najważniejsze okno systemu, ponieważ umożliwia komunikację programisty z systemem. W oknie tym wprowadzając odpowiednie polecenia możemy definiować zmienne, wykonywać obliczenia i wizualizować rezultaty tych obliczeń. W oknie 4 zamieszczane są zmienne zdefiniowane przez użytkownika. W 5 zaś znajduje się historia wprowadzanych poleceń. Wciskając klawisze strzałek dół – góra możemy przesuwać się po poleceniach znajdujących się w historii poleceń bez konieczności ich ponownego wpisywania z klawiatury komputera.

Przejdziemy teraz do pracy w środowisku wiersza poleceń. Podstawowym obiektem pakietu Matlab jest macierz (stąd nazwa MatLab – MatrixLaboratory). Należy o tym pamiętać szczególnie w przypadku operacji arytmetycznych. Będzie o tym mowa w dalszej części skryptu. W odróżnieniu od tworzenia programów w języku programowania, tu nie ma konieczności definiowania typów. Wpisując zatem w wierszu poleceń :

```
>>a=5
```

spowodujemy, że zmienna a zostanie zdefiniowana w systemie. Wprowadzimy trzy dowolne zmienne a, b, c i wypiszemy polecenie:

```
>>whos
```

Otrzymamy wówczas informację o obiektach znajdujących się w pamięci podręcznej programu (rysunek 11.4).

```
>> whos
  Name        Size              Bytes  Class     Attributes

  a           1x1                   8  double
  b           1x1                   8  double
  c           1x1                   8  double

>>
```

Rys. 11.4. Informacja o zmiennych znajdujących się w pamięci podręcznej programu

W jaki sposób definiuje się macierz? Macierz definiujemy tak, jak zwykłą zmienną pamiętając o tym, że elementy macierzy umieszczamy w nawiasach kwadratowych a poszczególne wiersze są oddzielone są średnikami. Jeżeli macierz ma tylko jeden wiersz lub kolumnę – jest wtedy wektorem wierszowym lub kolumnowym. Przykład implementacji macierzy przedstawia rysunek 11.5.

```
>> A=[1 2 3; 4 5 6; 7 8 9]

A =

     1     2     3
     4     5     6
     7     8     9

>> A=[1 2 3 4 5]

A =

     1     2     3     4     5
```

Rys. 11.5. Definiowanie macierzy w systemie

Dostęp do poszczególnych elementów macierzy możliwy jest przez wpisanie nazwy macierzy i indeksu elementu, który chcemy przetwarzać, np. polecenie.:

>> A(2,1)

spowoduje wyświetlenie elementu znajdującego się w wierszu 2 i kolumnie 1. W celu przetwarzania całych wierszy lub kolumn należy użyć operatora dwukropek ':' . Polecenie:

>> A(2,:)

spowoduje wyświetlenie całego drugiego wiersza. Analogicznie:

>> A(:,1)

spowoduje wyświetlenie elementów pierwszej kolumny. Tabela 11.1 przedstawia podstawowe operacje wykonywane na macierzach oraz ich operatory z podziałem na operacje tablicowe i macierzowe.

Tabela 11.1. Macierzowe i tablicowe operatory arytmetyczne

Działanie	Operator macierzowy	Operator tablicowy
suma	+	+
różnica	-	-
iloczyn	*	.*
dzielenie	/ lub \	./
potęgowanie	^	.^
transpozycja	'	

Operatory mnożenia i dzielenia tablicowego oznaczają takie działania, w których mnożone lub dzielone są odpowiadające sobie elementy. Rozmiar macierzy, na których wykonywane są operacje musi być taki sam. W przypadku mnożenia macierzowego (w sensie Cauchy'ego) liczba kolumn macierzy A musi być równa liczbie wierszy macierzy B.

Rysunek 11.6 przedstawia przykład działania operatorów sumy, różnicy, mnożenia macierzowego i tablicowego dla poniższych macierzy:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 9 & 8 & 7 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix}$$

```
>> A+B

ans =

    10    10    10
    10    10    10
    10    10    10

>> A-B

ans =

    -8    -6    -4
    -2     0     2
     4     6     8

>> A*B

ans =

    30    24    18
    84    69    54
   138   114    90

>> A.*B

ans =

     9    16    21
    24    25    24
    21    16     9

>>
```

Rys. 11.6. Przykłady działań z wykorzystaniem operatorów macierzowych i tablicowych

Podobnie jak omawiany w poprzednich częściach niniejszego skryptu pakiet MS Excel czy kompilator języka C++, Matlab ma zdefiniowane biblioteki z wieloma funkcjami matematycznymi, które wywoływane są poprzez podanie nazwy funkcji oraz jej argumentów w nawiasie okrągłym. Na przykład funkcję pierwiastek wywołuje się następującym poleceniem:

```
>>sqrt(X)
```

gdzie X może być zarówno liczbą, wektorem jak i macierzą.

Przykłady wybranych funkcji matematycznych przedstawiono w tabeli 11.2.

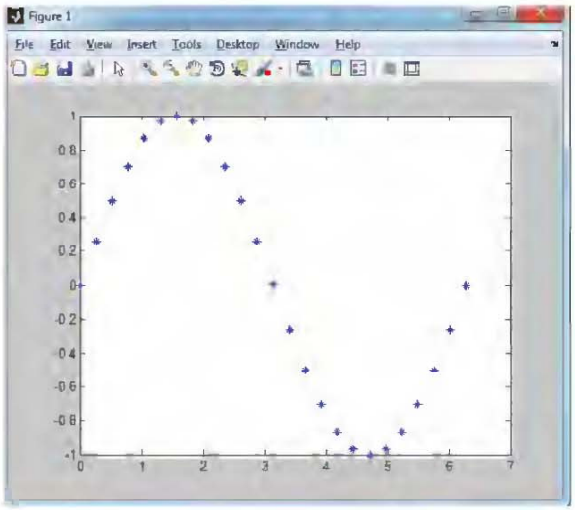
Tabela 11.2. Wybrane funkcje matematyczne programu Matlab

funkcja	Opis
Y=sqrt(X)	Macierz pierwiastków kwadratowych z macierzy X
Y=exp(X)	Macierz o elementach e^{λ}
Y=log(X)	Logarytm naturalny elementów macierzy X
Y=log10(X)	Logarytm dziesiętny elementów macierzy X
Y=sin(X)	Sinus elementów macierzy X
Y=cos(X)	Cosinus elementów macierzy X
Y=tan(X)	Tangens elementów macierzy X
Y=abs(X)	Wartość bezwzględna el. X
Y=ceil(X)	Zaokrągla elementy macierzy w górę
Y=floor(X)	Zaokrągla elementy macierzy w dół
Y=round(X)	Zaokrągla elementy macierzy do najbliższej liczby całkowitej
Y=rem(X,Z)	Oblicza resztę z dzielenia X przez Z

Poniższy kod ilustruje sposób wykorzystania zapisu macierzowego i funkcji w zastosowaniu do tabelaryzacji prostej funkcji *sin(x)* w celu wyrysowania wykresu funkcji. Wykres wykonany będzie dla $x \in (0 ; 2\pi)$.

```
>> x=0:pi/12:2*pi;
>> y=sin(x);
>> plot(x,y,'*')
```

Pierwsze polecenie generuje wektor liczb od 0 do 2π z krokiem $\pi/12$. W kolejnej linii kodu obliczane są wartości y funkcji sinus. Linia trzecia to polecenia narysowania prostego wykresu punktowego. Rezultat działania powyższego kodu przedstawia rysunek 11.7.



Rys. 11.7. Wykres punktowy funkcji *sin (x)* dla $x \in (0 ; 2\pi)$

Zadania do wykonania

Zadanie 1.

Dane są dwa wektory w przestrzeni trójwymiarowej:

$$\vec{a} = 2\vec{i} + 3\vec{j} + 4\vec{k}$$

$$\vec{b} = 3\vec{i} + 5\vec{j} - 2\vec{k}$$

Wykorzystując wiersz poleceń i dostępne operatory arytmetyczne zdefiniować obiekty w przestrzeni Matlab'a a następnie obliczyć:

a) Sumę wektorów
b) Różnicę wektorów $\vec{a} - \vec{b}$
c) Iloczyn skalarny wektorów
d) Iloczyn wektorowy wektorów

Zadanie 2.

Znaleźć współrzędne biegunowe punktów (1,1); (1, -1); (-1,1); (-1, -1) wiedząc, że:

$x=r\cdot cos(\varphi)$
$y=r\cdot sin(\varphi)$

Sprawdzić otrzymane wyniki obliczeń używając funkcji **cart2pol**

Literatura

1. http://www.mathworks.com/
2. Zalewski A, Cegieła R. Matlab. Obliczenia numeryczne i ich zastosowanie. Poznań, Wydawnictwo Nakom, 1997.
3. Krzyżanowski P. Obliczenia inżynierkie i naukowe. Warszawa, PWN, 2011.

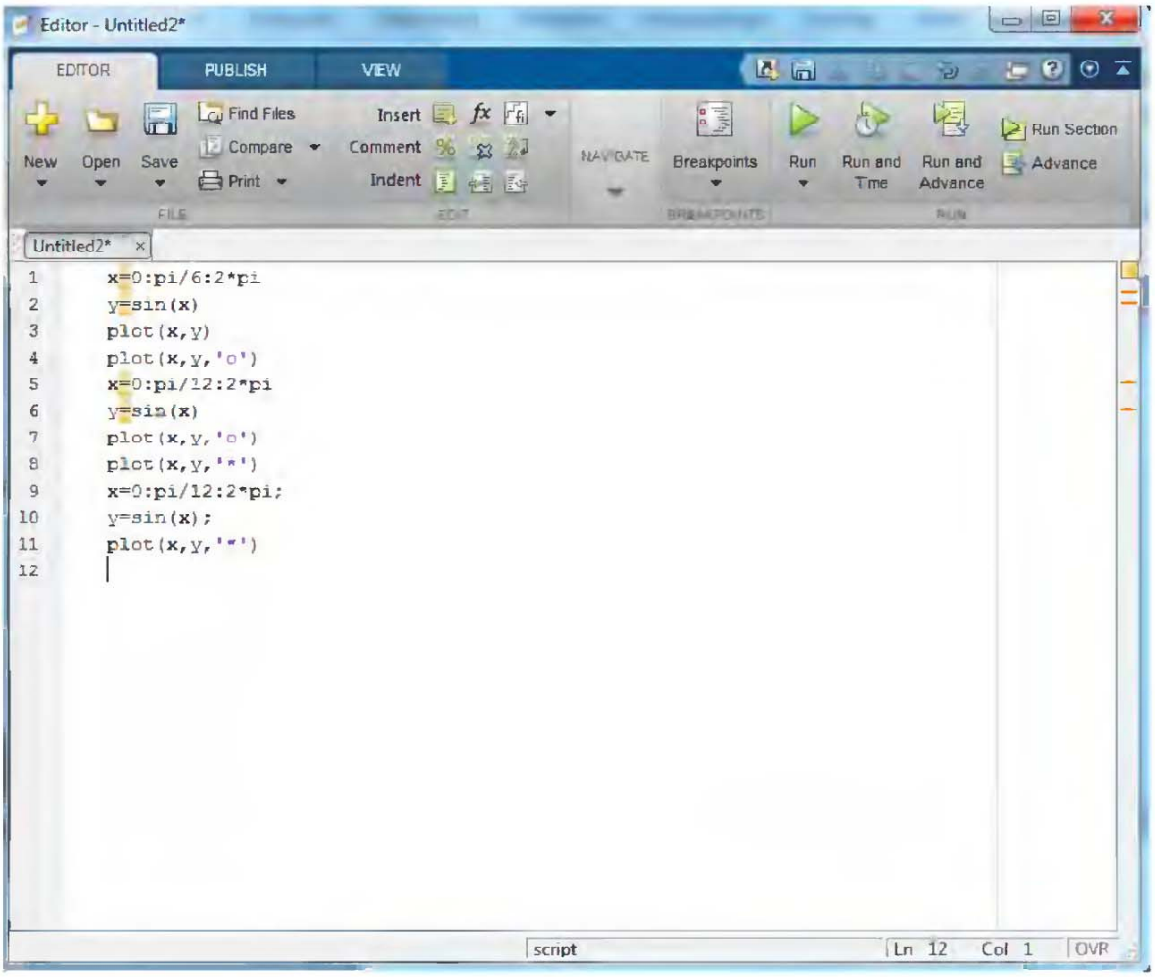
12. Podstawowe koncepcje programistyczne. Skrypty

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- System obliczeniowy MATLAB

Wstęp teoretyczny

W poprzednim ćwiczeniu skupiliśmy uwagę na wykorzystaniu okna poleceń do wykonania podstawowych obliczeń. Nie jest to jednak wygodny sposób w przypadku, gdy liczba linii kodu jest duża. Jakakolwiek zmiana wymaga wówczas powtórzenia wszystkich poleceń, których ta zmiana dotyczy. Dlatego zaleca się tworzenie skryptu, który zapisuje się na dysku w postaci pliku o rozszerzeniu .m (tzw. m-file). Możliwe jest tworzenie złożonych skryptów w postaci kilku plików odwołujących się do siebie. Skrypty tworzy się w edytorze, który przedstawiony został na rysunku 12.1



Rys. 12.1. Edytor do tworzenia skryptów

Skrypt uruchamiany jest przyciskiem *Run*. Po jego wciśnięciu wczytywane są kolejne polecenia zapisane w pliku, a zdefiniowane zmienne zapamiętywane są w pamięci i dostępne w środowisku również z poziomu wiersza poleceń. Na początku każdego skryptu zaleca się użyć polecenia clear all w celu wyczyszczenia pamięci.

W środowisku Matlab możliwe jest również implementowanie znanych z poprzednich rozdziałów konstrukcji programistycznych (warunkowych, iteracyjnych itp). Idea działania jest analogiczna do tej, którą poznaliśmy w poprzednim rozdziale, omawiając język C/C++ więc ograniczymy się jedynie do podania składni.

Iteracja

W środowisku Matlab występują dwie instrukcje iteracyjne :

- Instrukcja *for*
- Instrukcja *while*

Składnia instrukcji for jest następująca:

```
for zmienna=od:krok:do
        ciąg instrukcji
end
```

Przykład:

```
for i=1:2:10
        suma=suma+i;
end
```

Składnia instrukcji *while*

```
while wyrażenie_logiczne
        ciąg instrukcji
end
```

Przykład:

```
while (i<10)
        suma=suma+i;
        i=i+1;
end
```

Warunek

Podobnie jak w języku C++ tak i w środowisku Matlab istnieje instrukcja warunkowa **if**. W ogólnym przypadku (wybór wielowariantowy) składnia instrukcji jest następująca:

```
ifwyrażenie_logiczne1
          ciąg_instrukcji1
else  if  wyrażenie_logiczne2
          ciąg_instrukcji2
else
        ciąg_instrukcji3
end
```

Przy konstrukcji wyrażenia logicznego zarówno w przypadku instrukcji warunkowej jak i iteracyjnej korzysta się z operatorów logicznych oraz relacji (patrz tabela 12.1)

Tabela 12.1. Operatory w Matlab

Operator	Opis
a < b	a mniejsze niż b
a <= b	a mniejsze lub równe b
a > b	a większe niż b
a >= b	a większe lub równe b
a == b	a równe b
a ~= b	a różne od b
~a	Negacja; nie **a**
a&b	koniunkcja (AND – i)
a|b	alternatywa (OR – lub)

Do wyboru wielowariantowego można posłużyć się instrukcją *switch*. Składnia instrukcji przedstawia się następująco:

```
switch zmienna
        case wartość1
                instrukcje
        case wartość2
                instrukcje
        ...
        otherwise
                instrukcje
end
```

W środowisku Matlab można realizować zadania typowe dla kompilatora języka C/C++. Należy jednak pamiętać, że nadużywanie instrukcji warunkowej lub iteracyjnej jest nieeleganckie. Wykorzystując macierzowy charakter obiektów oraz dostępne operatory należy tak przygotowywać skrypty aby unikać stosowania pętli. Mówi się w tym przypadku o *wektoryzacji kodu*.

Tabela 12.2. Funkcje wspomagające wektoryzację kodu

Funkcja	Opis
sum(X)	Suma elementów macierzy
prod(X)	Iloczyn elementów macierzy
min(X)	Najmniejszy element w kolumnach
max(X)	Największy element w kolumnach
sort(X)	Sortuje kolumny macierzy X
mean(X)	Średnia arytmetyczna X
median(X)	Mediana X
std(X)	Odchylenie standardowe X

Na rysunku 12.2 przedstawiono przykład kodu, który oblicza wartość wyrażenia (20):

$$Suma=\sum_{i=1}^{N}\frac{1}{n^2} \tag{20}$$

```
1 -    clear all;
2
3 -    s=0;
4 -    N=10000;
5 -    for n=1:N % początek pętli
6 -    s = s + 1/(n^2);
7 -    end
8 -    fprintf(' Sum = %g \n',s) % wyświetlenie odpowiedzi w oknie poleceń
9
10     |
```

Rys. 12.2. Przykład skryptu obliczającego wartość wyrażenia (1)

Używając zapisu macierzowego dla obiektów oraz operatorów zasięgu ':' i dzielenia tablicowego ' ./ ' oraz potęgowania tablicowego '.^' możemy to samo zadanie zrealizować za pomocą następujących poleceń:

```
n=1:N;
sum(1./n.^2)
```

Powyższy sposób zapisu prowadzi do znacznych oszczędności czasu obliczeń w porównaniu do metody przedstawionej na rysunku 12.2.

Zadania do wykonania

Zadanie 1.

Dla następujących danych (wektor, macierz) oblicz średnią arytmetyczną. Użyj poleceń *mean* oraz *nanmean*.[1]:

a) [1 5 9 4 14 22 17 16 21 25]
b) [9 11 3; 4 7 6; 3.5 8.9 2.26]
c) [1 NaN 1.45; 2.5 7.3 NaN; 1.2 1.1 1.5]

NaN – Not a Number

Zadanie 2.

Napisz skrypt, wykorzystujący instrukcję iteracji, który będzie obliczał wartość średniej arytmetycznej danych z zadania 1.

Wzór programu do przykładu a)

```
1 -    N=10;
2 -    S=0;
3 -    X=[1 5 9 4 14 22 17 16 21 25];
4
5 -    for i=1:N
6 -        S=S+X(i);
7 -    end
8
9 -    fprintf('Srednia=%f \n',S/N)
10
```

Rys. 12.3. Wzór programu obliczającego średnią arytmetyczną wektora X

Wzór programu do przykładu b)

```
1 -    N=3;
2 -    S=0;
3 -    X=[9 11 3; 4 7 6; 3.5 8.9 2.26];
4
5
6 -    for i=1:N
7 -      for j=1:N
8 -          S=S+X(j,i);
9 -      end
10
11 -     fprintf('Srednia=%f \n',S/N)
12 -     S=0;
13 -   end
```

Rys. 12.4. Wzór programu obliczającego średnią arytmetyczną macierzy X

Zadanie 3.

Utwórz skrypt obliczający odchylenie standardowe dla wektora i macierzy wg następującego wzoru [1]:

$$s = \left[\frac{1}{n-1}\sum_{i=1}^{n} (X_i - M_1)^2\right]^{\frac{1}{2}}$$

Dane:

a) [1 5 9 4 14 22 17 16 21 25]

b) [9 11 3; 4 7 6; 3.5 8.9 2.26]

Zadanie 4.

Oblicz odchylenie standardowe do danych z zadania 3 używając polecenia *std*.Porównaj otrzymane wyniki z wynikami z zadania 3.

Zadanie 5.

Utwórz skrypt znajdujący najmniejszy i największy element listy z pominięciem nieznanych lub błędnych wartości a następnie różnicę między tymi wartościami. Użyj poleceń *nanmin, nanmax, range*.

Zadanie 6.

Napisać skrypt obliczający pierwiastki równania kwadratowego. Skrypt powinien zapytać użytkownika o współczynniki równania oraz wyświetlić miejsce zerowe z informacją, czy są to pierwiastki rzeczywiste czy zespolone.

Zadanie 7.

Napisać funkcje przeliczające: radiany na stopnie, km/h na m/s, stopnie Celsjusza na stopnie Fahrenheita ($1°F=9/5 \cdot °C+32$)

Literatura

1. Regel W. Statystyka matematyczna w programie Matlab. Warszawa, Mikom, 2003.
2. *http://www.mathworks.com/*
3. Zalewski A, Cegieła R. Matlab. Obliczenia numeryczne i ich zastosowanie. Poznań, Wydawnictwo Nakom, 1997.
4. Krzyżanowski P. Obliczenia inżynierkie i naukowe. Warszawa, PWN, 2011.
5. Stachurski M. Metody numeryczne w programie Matlab. Warszawa, Mikom, 2003.

13. Wykresy dwuwymiarowe i trójwymiarowe

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- System obliczeniowy MATLAB

Wstęp teoretyczny

Bardzo ważnym etapem opracowania wyników obliczeń i pomiarów jest ich wizualizacja w postaci wykresów. Matlab zawiera odpowiednie biblioteki umożliwiające tworzenie dwu i trójwymiarowych wykresów, zawierających dodatkowe funkcje do tworzenia etykiet, zmiany skali, tworzenia tekstu. Pakiet oferuje również specjalne narzędzie, zwane *CurveFittingToolbox* umożliwiające zaawansowaną analizę danych. W niniejszym ćwiczeniu skupimy uwagę głównie na dwuwymiarowych wykresach punktowych i liniowych oraz trójwymiarowych wykresach powierzchniowych. Najprostszym rodzajem wykresu jest wykres funkcji jednej zmiennej. Będziemy chcieli utworzyć wykres liniowy następujących funkcji:

$y_1=sin(x)$ (21)

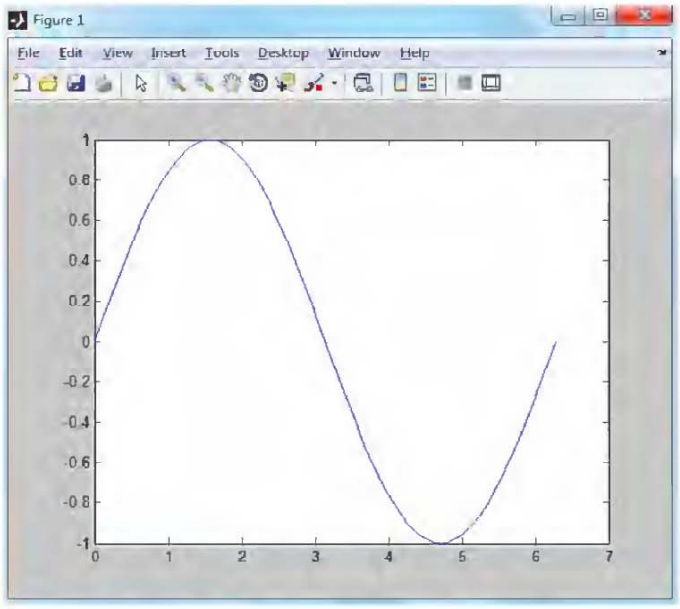
$y_2=cos(x)$, dla $x \in (0; 2\pi)$ z krokiem $\pi/100$.

W tym celu należy utworzyć trzy wektory (mogą być kolumnowe) – pierwszy z wartościami x, obliczonymi w podanym wyżej przedziale – drugi i trzeci odpowiednio z obliczonymi wartościami y_1 i y_2. Poniższe polecenia realizują te zadania:

```
x=0:pi/100:2*pi;
y1=sin(x);
y2=sin(y);
```

Do utworzenia prostego wykresu pojedynczej funkcji należy wprowadzić polecenie:

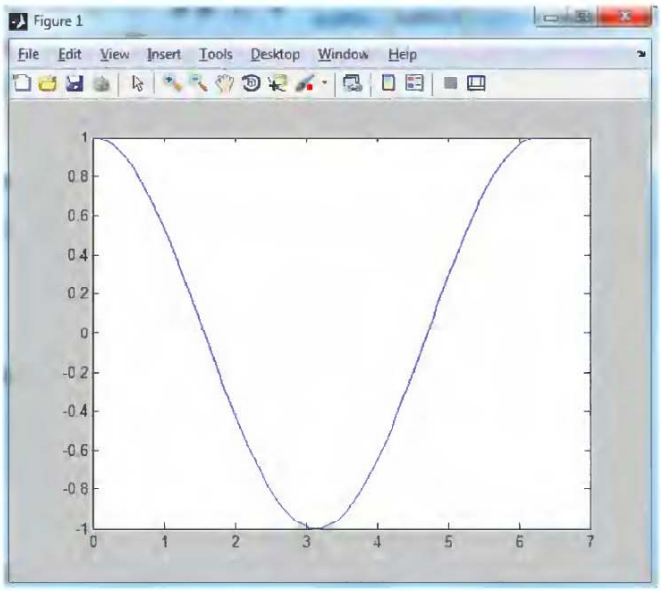
```
plot(x,y1);
```



Rys. 13.1. Wykres funkcji *sin(x)* dla $x \in (0; 2\pi)$

Wykresy tworzone są w oddzielnym oknie o tytule *Figure* i kolejnym numerem rysunku. Należy pamiętać jednak, że wpisując ponownie polecenie plot dla drugiej funkcji, jej wykres zostanie utworzony w tym samym oknie Figure 1:

plot(x,y2);



Rys. 13.2. Wykres funkcji *cos(x)* dla $x \in (0; 2\pi)$

W celu utworzenia wykresów w różnych oknach należy kolejne polecenia plot poprzedzić poleceniem figure w celu przygotowania pustego okna dla następnego wykresu. Rysunek 13.3 przedstawia skrypt, realizujący to zadanie.

s1.m

```
1     %przygotowanie punktow
2 -   x=0:pi/100:2*pi;
3 -   y1=sin(x);
4 -   y2=cos(x);
5
6     %wykres funkcji sin(x)
7 -   plot(x,y1);
8 -   title ('Wykres funkcji sin(x)');
9 -   xlabel('x = 0:2\pi');
10 -  ylabel('y = sin(x)');
11
12 -  figure;
13
14    %wykres funkcji cos(x)
15 -  plot(x,y2);
16 -  title ('Wykres funkcji cos(x)');
17 -  xlabel('x = 0:2\pi');
18 -  ylabel('y = cos(x)');
19
20
```

Rys. 13.3. Kod skryptu rysującego funkcje *sin(x)* i *cos (x)* w dwóch oknach

Należy zwrócić uwagę na polecenia nadające tytuł wykresu (linie 8 i 16) oraz polecenia etykiet osi x i y (linie 9,10, 17,18). Polecenie \pi, znane z edytora LaTeX, utworzy symbol π w etykiecie osi *x*. Inne przykłady znaczników podane zostały w tabeli 13.1.

Tabela 13.1. Znaczniki liter greckich

Znacznik	Litera grecka
\alpha	α
\beta	β
\gamma	γ
\delta	δ
\epsilon	ε
\phi	φ
\theta	θ
\lambda	λ
\sigma	σ

W pakiecie Matlab istnieje również możliwość utworzenia dwu (lub więcej) wykresów na jednym rysunku. Służy do tego celu następujące polecenie:

subplot(m, n, p)

m – liczba wykresów w poziomie
n – liczba wykresów w pionie
p – uaktywnia określoną część rysunku

Jeżeli zatem chcemy utworzyć dwa wykresy jedne pod drugim, to przed poleceniem plot wpiszemy :

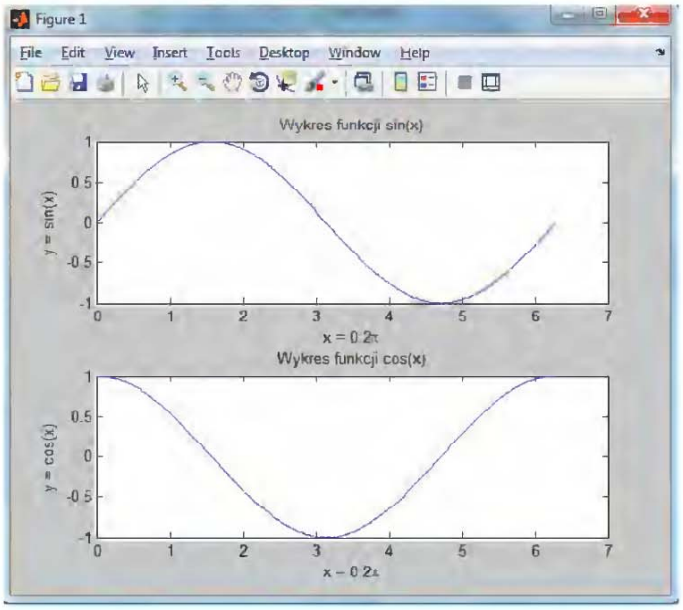
subplot(2,1, p);

s1.m × s2.m ×

```
1 -  clear all
2    %przygotowanie punktow
3 -  x=0:pi/100:2*pi;
4 -  y1=sin(x);
5 -  y2=cos(x);
6
7    %wykres funkcji sin(x)
8 -  subplot(2,1,1);
9 -  plot(x,y1);
10
11 - title ('Wykres funkcji sin(x)');
12 - xlabel('x = 0:2\pi');
13 - ylabel('y = sin(x)');
14
15   %wykres funkcji cos(x)
16 - subplot(2,1,2);
17 - plot(x,y2);
18
19 - title ('Wykres funkcji cos(x)');
20 - xlabel('x = 0:2\pi');
21 - ylabel('y = cos(x)');
```

Rys. 13.4. Skrypt rysujący dwa wykresy na jednym rysunku

Rezultat wykonania kodu przedstawiony został na rysunku 13.5.



Rys. 13.5. Wykresy utworzone z wykorzystaniem funkcji subplot

Jak wspomniano wcześniej, wywołanie funkcji *plot* powoduje narysowanie nowego wykresu w tym samym oknie *figure*, w którym znajdował się poprzedni wykres. W celu zatrzymania poprzedniego wykresu i narysowania nowego w tym samym oknie, należy po instrukcji plot użyć polecenia:

```
hold on
```

Powrót do domyślnego trybu pracy realizowany jest poleceniem:

```
hold off
```

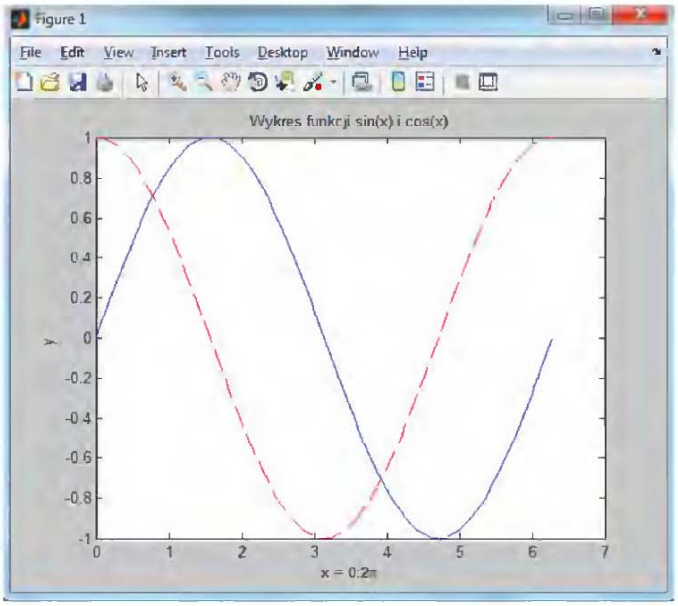
Wynik działania skryptu z rysunku 13.3 zmodyfikowanego pod kątem rysowania dwóch wykresów w jednym układzie współrzędnych przedstawia rysunek 13.6. Dodatkowo dla wykresu funkcji $y = cos(x)$ ustawiony został kolor linii na czerwony i kształt – linia przerywana. Wykres funkcji $y = sin(x)$ narysowano kolorem niebieskim i linią ciągłą. Ustawienia koloru, rodzaju linii oraz markera dokonuje się poprzez umieszczenie za parą argumentów łańcucha znaków zawierającego odpowiedni kod. W tym konkretnym przykładzie dla funkcji sinus zapiszemy:

```
plot(x,y1, 'b-');
```

natomiast dla funkcji cosinus zapiszemy:

```
plot(x,y1, 'r--');
```

Łańcuchy 'b-'i 'r--'oznaczają odpowiednio kolor niebieski i linia ciągła oraz kolor czerwony i linia przerywana.



Rys. 13.6. Wykres obydwu funkcji w jednym układzie współrzędnych

Tabele 13.2 i 13.3 zawierają kody kolorów i oznaczenia rodzaju linii.

Tabela 13.2. Kody kolorów stosowane w poleceniu plot

Kod	Kolor
b	niebieski (blue)
g	zielony (green)
r	czerwony (red)
w	biały (white)
k	czarny (blacK)
y	żółty (yellow)
m	karmazynowy (magenta)
c	siny (cyan)

Tabela 13.3. Kody kolorów stosowane w poleceniu plot

Kod	Typ linii
.	punkty
o	kółka
x	znaki x
+	znaki +
*	znaki *
-	linia ciągła
--	linia przerywana
:	dwukropek
-.	kreska - kropka

Przykłady łańcuchów i ich oznaczenia:
'y+' – żółte markery w punktach
'r--' – czerwona linia przerywana
'ko' – czarne kółka w punktach

W celu wykonania wykresu trójwymiarowego należy w pierwszej kolejności utworzyć macierze opisujące położenie węzłów siatki. Oczywiście nie oznacza to, że macierze będziemy tworzyć sami. Zrobi to za nas Matlab, a właściwie jego funkcja :

meshgrid

Dla przykładu narysujemy wykres następującej funkcji:

$Z=sin(R)/R$, gdzie $R = \sqrt{x^2 + y^2} + eps$ (22)

oraz

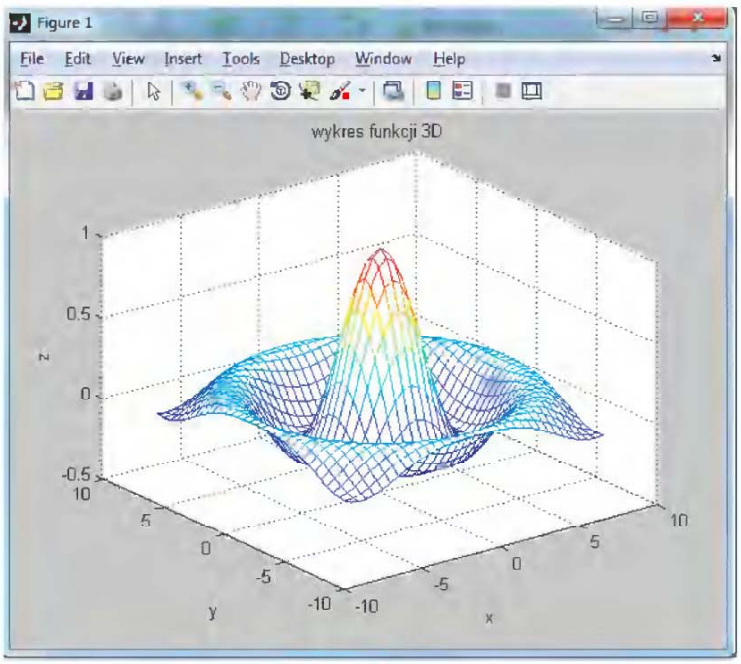
$x \in (-8; 0.5; 8)$ i $y \in (-8; 0.5; 8)$

Kod skryptu rysujący wykres 3W funkcji (22) przedstawiono na rysunku 13.7. Rezultat działania przedstawiono na rysunku 13.8.

s1.m | s2.m | s3.m

```
1 -  [X,Y]=meshgrid(-8:0.5:8,-8:0.5:8);
2
3 -  R=sqrt(X.^2+Y.^2)+eps;
4 -  Z=sin(R)./R;
5
6 -  mesh(X,Y,Z);
7
8 -  title ('wykres funkcji 3D');
9 -  xlabel('x');
10 - ylabel('y');
11 - zlabel('z');
12
```

Rys. 13.7. Kod programu rysującego wykres funkcji trójwymiarowej

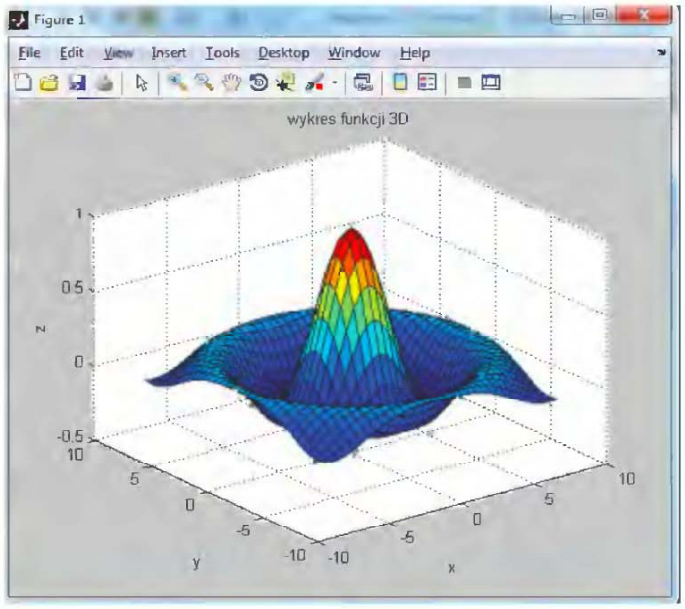


Rys. 13.8. Wykres powierzchniowy (siatka) funkcji *f(x,y)*

W przypadku, gdy chcemy uzyskać powierzchnię zamalowaną używamy polecenia

surf (X,Y,Z)

Rezultat działania polecenia przedstawiono na rysunku 13.9.



Rys. 13.9. Wykres powierzchniowy wykonany za pomocą funkcji *surf*

Zadania do wykonania

Zadanie 1.

Narysuj wykres punktowy funkcji:

$$s(t)=sin(2\pi 5t)cos(2\pi 3t)+e^{-0.1t}$$

dla wektora t od 0 do 10 z krokiem 0.1

Zadanie 2.

Oblicz i narysuj funkcję

$$f(x,y)=x^2+y^2$$

W obszarze [-2,2] x [-1,1] z krokiem 0.2 w kierunku x i 0.1 w kierunku y.

Literatura

1. Zalewski A, Cegieła R. Matlab. Obliczenia numeryczne i ich zastosowanie. Poznań, Wydawnictwo Nakom, 1997.

14. Rozwiązywanie układów równań w Matlabie

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- System obliczeniowy MATLAB

Wstęp teoretyczny

Dwa ostatnie ćwiczenia skryptu poświęcone zostały bardziej zaawansowanym zastosowaniom pakietu obliczeniowego Matlab. W niniejszym ćwiczeniu skupimy uwagę na układach równań. Wiele zagadnień z dziedziny nauk przyrodniczych oraz inżynierskich sprowadza się do ułożenia, w oparciu o działające prawa, układu równań a następnie jego rozwiązania. W ćwiczeniu pokażemy przykłady w jaki sposób można te układy tworzyć a następnie jak je implementować w macierzowym zapisie Matlab'a oraz jak uzyskać ich rozwiązanie.

Rozważmy poniższy układ równań:

$$\begin{cases} x_1 - x_2 + 3x_3 = 5 \\ 5x_1 - x_3 = -2 \\ x_1 + x_2 + x_3 = 3 \end{cases} \tag{23}$$

W pierwszej kolejności rozwiążemy układ metodą wyznaczników. Oznaczymy poszczególne wyznaczniki:

$$W = \begin{vmatrix} 1 & -1 & 3 \\ 5 & 0 & -1 \\ 1 & 1 & 1 \end{vmatrix} \tag{24}$$

$$W_1 = \begin{vmatrix} 5 & -1 & 3 \\ -2 & 0 & -1 \\ 3 & 1 & 1 \end{vmatrix}$$

$$W_2 = \begin{vmatrix} 1 & 5 & 3 \\ 5 & -2 & -1 \\ 1 & 3 & 1 \end{vmatrix}$$

$$W_3 = \begin{vmatrix} 1 & -1 & 5 \\ 5 & 0 & -2 \\ 1 & 1 & 3 \end{vmatrix}$$

Jeżeli $W \neq 0$, to rozwiązanie układu będzie miało postać:

$$x_1 = \frac{W_1}{W}, \tag{25}$$

$$x_2 = \frac{W_2}{W},$$

$$x_3 = \frac{W_3}{W},$$

Nie będziemy liczyć wyznacznika macierzy „na piechotę". Wykorzystamy funkcję:

det(A)

Skrypt wykonujący zadanie przedstawiono na rysunku 14.1.

s4.m × s5.m × s6.m ×

```
1 -  clear all;
2
3 -  M=[1 -1 3; 5 0 -1; 1 1 1];
4
5 -  M1=[5 -2 3; -1 0 1; 3 -1 1];
6 -  M2=[1 5 1; 5 -2 3; 3 -1 1];
7 -  M3=[1 5 1; -1 0 1; 5 -2 3];
8
9 -  W=det(M)
10 - W1=det(M1)
11 - W2=det(M2)
12 - W3=det(M3)
13
14 - x1=W1/W;
15 - x2=W2/W;
16 - x3=W3/W;
17
18 - fprintf('x1=%f \n',x1)
19 - fprintf('x2=%f \n',x2)
20 - fprintf('x3=%f \n',x3)
```

Rys. 14.1. Skrypt rozwiązujący układ równań (23) metodą wyznaczników

Układ (23) możemy zapisać również w postaci macierzowej w następujący sposób:

$$\mathbf{A}\cdot\mathbf{X}=\mathbf{B} \tag{26}$$

$$\begin{bmatrix}1 & -1 & 3\\ 5 & 0 & -1\\ 1 & 1 & 1\end{bmatrix}\cdot\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}=\begin{bmatrix}5\\ -2\\ 3\end{bmatrix} \tag{27}$$

Rozwiązanie równania otrzymujemy wpisując polecenie:

$$\mathbf{X}=\mathbf{A}^{-1}\cdot\mathbf{B} \tag{28}$$

W środowisku Matlab utworzenie macierzy odwrotnej do **A,** a następnie wykonanie mnożenia tej macierzy przez macierz **B** nie jest zalecane. Z uwagi na efektywność obliczeń stosuje się w celu rozwiązania równania (26) operator \ :

$$\mathbf{X}=\mathbf{A}\setminus\mathbf{B} \tag{29}$$

Przykładowy skrypt przedstawiający rozwiązanie układu równań (23) zaprezentowano na rysunku 14.2. Wynik jego działania przedstawia rysunek 14.3.

s4.m

```
1       %wprowadzenie macierzy A
2 -     A=[1 -1 3; 5 0 -1; 1 1 1];
3
4       %wprowadzenie macierzy B
5 -     B=[5; -2; 3];
6
7       %obliczenie wyniku
8 -     x=A \ B;
9
```

Rys. 14.2. Skrypt rozwiązujący układ równań (23)

Command Window

```
>> s4

A =

     1    -1     3
     5     0    -1
     1     1     1


B =

     5
    -2
     3


x =

     0
     1
     2

fx >>
```

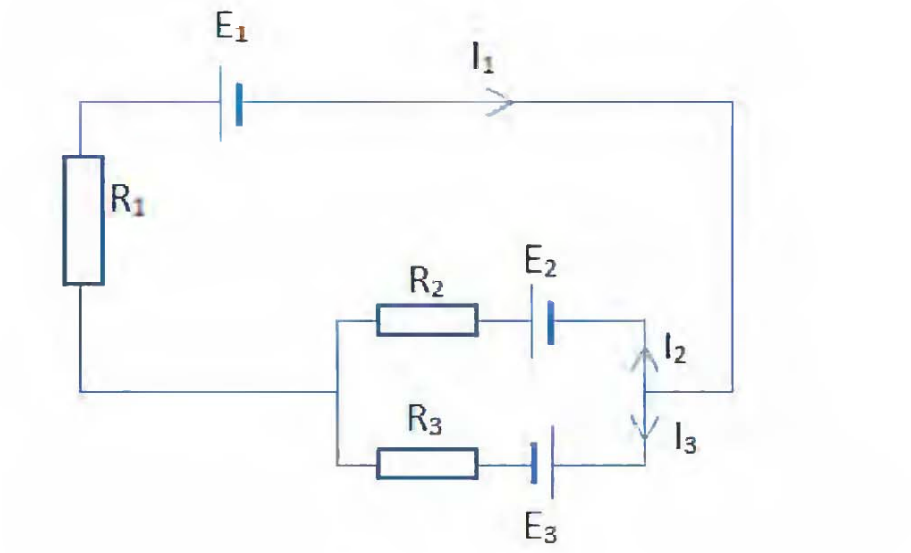
Rys. 14.3. Wynik działania skryptu z rysunku 14.1

Rozwiązaniem układu równań (23) są liczby:

$x_1=0, \quad x_2=1, \quad x_3=2.$

W kolejnym przykładzie pokażemy w jaki sposób można wykorzystać oprogramowanie Matlab do zadań, które sprowadzają się do ułożenia a następnie rozwiązania układu równań w celu znalezienia interesujących wielkości.

Na rysunku 14.4 przedstawiony jest prosty obwód elektryczny.



Rys. 14.4. Obwód elektryczny. Strzałkami wskazano założony kierunek przepływu prądu

Korzystając z praw Kirchoffa dla obwodów elektrycznych obliczymy wartości natężeń prądów płynących w obwodzie. W tym celu należy:

- ułożyć odpowiedni układ równań,
- dokonać implementacji układu w postaci macierzy,
- rozwiązać układ równań wykorzystując operator \

Układ równań:

$$\begin{cases} I_1 - I_2 - I_3 = 0 \\ I_1R_1 + I_2R_2 = E_2 - E_1 \\ I_2R_2 - I_3R_3 = E_2 + E_3 \end{cases} \tag{30}$$

Układ równań (8) w zapisie macierzowym będzie miał następującą postać:

$$\begin{bmatrix} 1 & -1 & -1 \\ R_1 & R_2 & 0 \\ 0 & R_2 & -R_3 \end{bmatrix} \cdot \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 0 \\ E_2 - E_1 \\ E_2 + E_3 \end{bmatrix} \tag{31}$$

Zwróćmy uwagę, że wykonując mnożenie powyższych macierzy otrzymamy dokładnie postać układu równań daną wzorem (8).

Przyjmując następujące oznaczenia macierzy:

$$\mathbf{R}=\begin{bmatrix} 1 & -1 & -1 \\ 0 & -R_2 & R_3 \\ R_1 & R_2 & 0 \end{bmatrix},$$

$$\mathbf{I}=\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix},$$

$$\mathbf{E}=\begin{bmatrix} 0 \\ E_2 - E_1 \\ E_2 + E_3 \end{bmatrix},$$

rozwiązanie układu wyraża się wzorem:

$$\mathbf{I} = \mathbf{R} \setminus \mathbf{E} \tag{32}$$

Skrypt realizujący powyższe zadanie przedstawiony został na rysunku 14.4. Przyjęto następujące dane wejściowe:

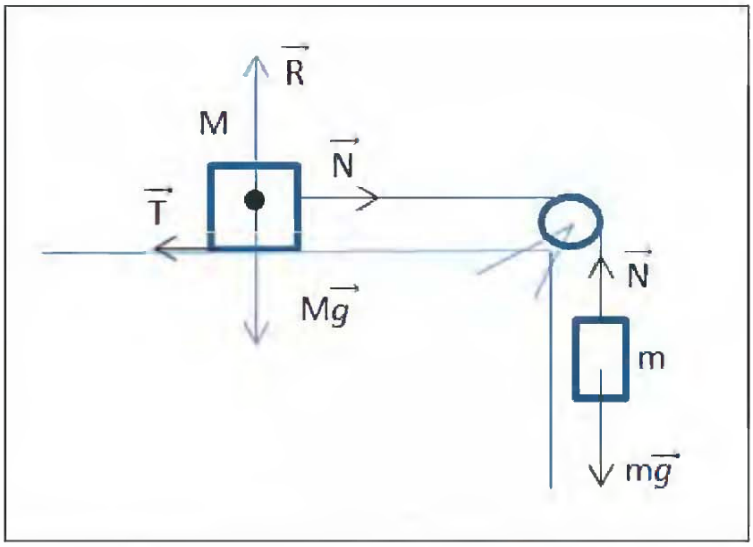
$R_1=5\Omega$,
$R_2= R_3=2\Omega$,
$E_1=12V$,
$E_2=3V$,
$E_3=5V$,

Kirchoff.m

```
1
2       %Dane zapisane w postaci wektorowej
3 -     R=[5 5 2];
4 -     E=[12 3 5];
5
6       %Przygotowanie macierzy oporów
7 -     Rx=[1 -1 -1; R(1) R(2) 0; 0 R(2) -R(3)];
8
9       %Przygotowanie macierzy Ex
10 -    Ex=[0 E(2)-E(1) E(2)+E(3)]';
11
12      %Rozwiązanie układu równań
13 -    I=Rx\Ex;
14
```

Rys. 14.4. Skrypt wyznaczający wartości prądów w obwodzie

Kolejny przykład jest klasycznym problemem dynamiki Newtona. W tym przypadku również rozwiązanie zadania możemy sprowadzić do rozwiązania układu równań.



Rys. 14.5. Układ ciał połączonych nicią

Rozwiązanie zadania sprowadza się do rozwiązania następującego układu równań:

$$\begin{cases} ma = mg - N \\ Ma = N - fMg \end{cases} \tag{33}$$

w którym niewiadomymi są przyspieszenie a oraz siła naciągu nici N. Przekształcimy układ (33) do następującej postaci:

$$\begin{cases} ma + N = mg \\ Ma - N = -fMg \end{cases} \tag{34}$$

Przyjmując następujące oznaczenia:

$\mathbf{Mx}=\begin{bmatrix} m & 1 \\ M & -1 \end{bmatrix}$,

$\mathbf{B}=\begin{bmatrix} mg \\ -fMg \end{bmatrix}$,

$\boldsymbol{X} = \begin{bmatrix} a \\ N \end{bmatrix}$,

możemy zapisać równanie macierzowe:

$$\begin{bmatrix} m & 1 \\ M & -1 \end{bmatrix} \cdot \begin{bmatrix} a \\ N \end{bmatrix} = \begin{bmatrix} mg \\ -fMg \end{bmatrix} \tag{35}$$

Rozwiązanie otrzymamy wpisując polecenie :

X=Mx \ B (36)

Zadania do wykonania

Zadanie 1.

Sprawdzić czy układ jest układem Cramera, jeżeli tak, to rozwiązać następujące układy równań:

a)

$$\begin{cases} 3x - 2y = 17 \\ -4x + 5y = -31 \end{cases}$$

b)

$$\begin{cases} x + 2y - z = 1 \\ -2x - y + 3z = 0 \\ -x + z = 0 \end{cases}$$

c)

$$\begin{cases} x + y - z = 2 \\ -x + 3y + 2z = 0 \\ 4x + y - 3z = 0 \end{cases}$$

d)

$$\begin{cases} 3x + y - 2z = -6 \\ x + y - 2z = 3 \\ -x - 2y + z = -5 \end{cases}$$

Zadania należy rozwiązać zapisując układy w postaci macierzowej (26). Można również utworzyć skrypt wyznaczający niewiadome w układach równań metodą wyznaczników.

Zadanie 2.

Napisać skrypt zgodnie z poniższym wzorem realizujący zadanie przedstawione równaniami (34) – (36). Przyjąć następujące dane:

M=5kg,
m=2kg,
f=0.1,
g=9.81;

Wyświetlić macierz rozwiązań **X**.

Literatura

1. Zalewski A, Cegieła R. Matlab. Obliczenia numeryczne i ich zastosowanie. Poznań, Wydawnictwo Nakom, 1997.
2. Krzyżanowski P. Obliczenia inżynierkie i naukowe. Warszawa, PWN, 2011.

15. Aproksymacja wielomianami różnego stopnia

Wymagania sprzętowe i oprogramowanie

- Komputer klasy PC
- System operacyjny Windows XP/Vista/7
- System obliczeniowy MATLAB

Wstęp teoretyczny

Ostatnie ćwiczenie niniejszego skryptu poświęcone zostało niezwykle istotnym zagadnieniom z punktu widzenia analizy danych – aproksymacji. Bardzo często w praktyce badawczej istnieje konieczność znalezienia zależności funkcyjnej zmierzonych wartości. Pokażemy w jaki sposób tworzyć skrypty, które umożliwiają analizę danych doświadczalnych i pozyskiwanie z nich określonych informacji.

W tabeli 15.1 przedstawione są przykładowe wyniki pomiarów natężenia promieniowania gamma przechodzącego przez warstwę absorbentu od grubości tej warstwy. Przedstawioną zależność można wyrazić wzorem (37):

$$lnN=lnN_0-\mu x, \tag{37}$$

gdzie
N_0 *to liczba zliczeń N dla* $x=0$

Celem ćwiczenia będzie wyznaczenie tzw. liniowego współczynnika absorpcji μ.

Tabela 15.1. Przykładowe dane pomiarowe

x [cm]	N
0	2880
0,03	2391
0,06	2088
0,09	1953
0,12	1830
0,15	1605
0,2	1458
0,25	1170
0,33	924
0,44	582
0,57	303
0,69	213
0,84	198
1,04	186
1,26	183
1,52	198

W tym celu przygotujemy skrypt, którego zadaniem będzie w pierwszej kolejności obliczenie *lnN*, a następnie narysowanie wykresu zależności *lnN(x)*. Na podstawie wykresu będzie możliwe określenie współczynnika absorpcji poprzez wpisanie w punkty doświadczalne prostej o równaniu (38):

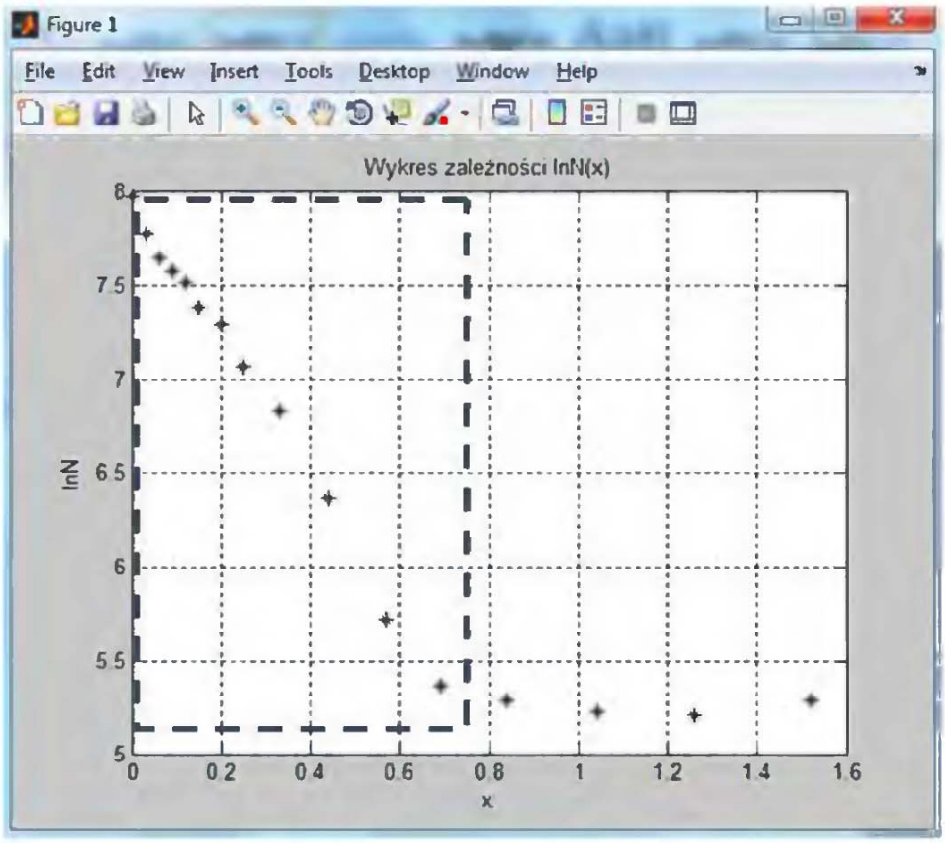
$$y=ax+b \tag{38}$$

Początkowy fragment skryptu oraz wynik jego działania przedstawione zostały odpowiednio na rysunkach 15.1 i 15.2.

```
c18.m
1
2 -    clear all;
3
4 -    x=[0 0.03 0.06 0.09 0.12 0.15 0.2 0.25 0.33 0.44 0.57 0.69 0.84 1.04 1.26 1.52];
5 -    N=[2880 2391 2088 1953 1830 1605 1458 1170 924 582 303 213 198 186 153 198]
6
7 -    lnN=log(N);
8
9 -    plot(x,lnN,'*k');
10
11 -   xlabel('x');
12 -   ylabel('lnN');
13
14 -   title('Wykres zależności lnN(x)');
15 -   grid on;
```

Rys.15.1. Fragment skryptu tworzący wykres *lnN(x)*

Polecenie clear all czyści zadeklarowane zmienne znajdujące się w przestrzeni roboczej (workspace). Linie 4 i 5 zawierają wektory z danymi zebranymi w tabeli 15.1. W linii 7 utworzony zostaje wektor wartości *lnN*. Kolejne polecenia to, znane z wcześniejszych ćwiczeń, polecenia utworzenia wykresu punktowego z markerem typu '*' oraz tytułem i etykietami osi.



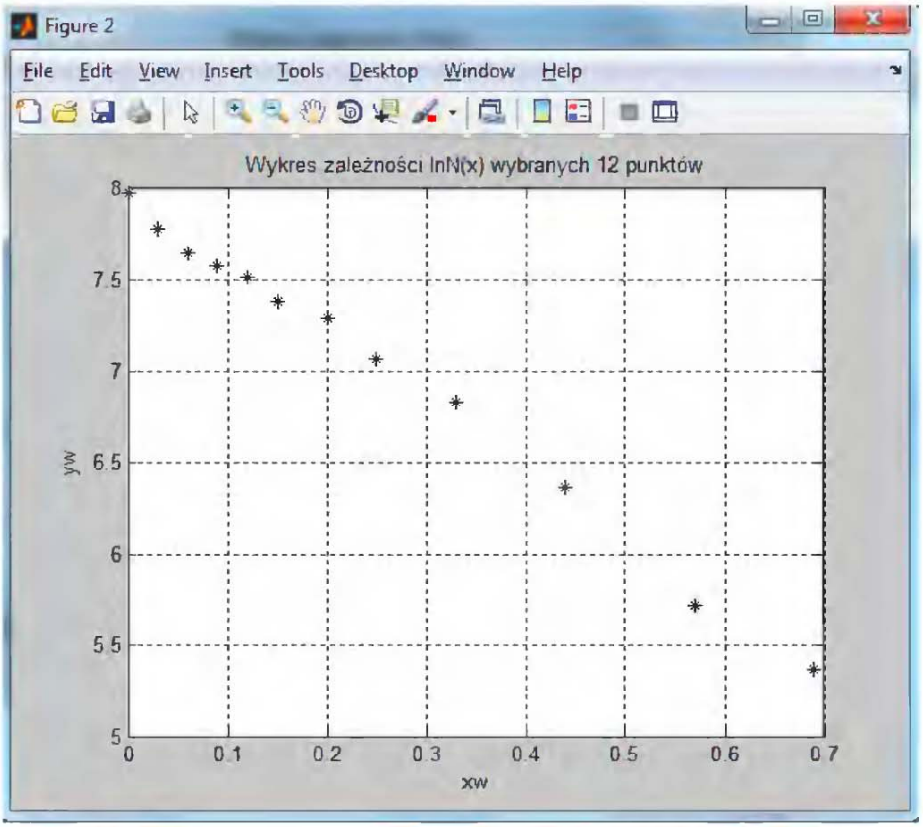
Rys.15.2. Wykres zależności *lnN(x)* danych z tabeli 15.1

Z wykresy przedstawionego na rysunku 15.2 wynika, że nie wszystkie punkty doświadczalne zachowują trend liniowy. Nie możemy zatem dopasować prostej do wszystkich punktów pomiarowych zaprezentowanych na wykresie. Po dokonaniu wstępnej oceny wybrano punkty pomiarowe, które zaznaczono na rysunku 15.2 linią przerywaną.

Spośród wszystkich punktów pomiarowych wybieramy 12 początkowych. Utworzymy nowe wektory *xw* i *yw*:

```
xw=x(1,1:12);
yw=lnN(1,1:12);
```

Nawiasy okrągłe oznaczają dostęp do określonych elementów macierzy (w tym przypadku wektora). Pierwsza liczba oznacza wiersz, druga kolumnę. Użycie operatora dwukropka oznacza wybór kolumn od nr 1 do nr 12. Dla kontroli warto wyświetlić wybrane punkty, żeby jednoznacznie stwierdzić, że zachowują trend liniowy. Wykorzystujemy analogiczne polecenie *plot*, z tą różnicą, że jego argumentami będą nowe wektory *xw* i *yw*. Rezultat przedstawiony jest na rysunku 15.3.



Rys. 15.3. Wykres punktowy danych wybranych do wpisania prostej $y = ax+b$

Po wybraniu właściwych danych można przystąpić do wpisania prostej, wyznaczenia współczynników *a, b* oraz wyrysowania jej wykresu.

W środowisku Matlab standardową metodą aproksymacji jest aproksymacja średniokwadratowa wielomianami wybranego stopnia. Współczynniki wielomianu stopnia *n* dopasowanego do danych wektorów *x* i *y* możemy otrzymać stosując polecenie *polyfit*, np.:

```
coeff=polyfit(x,y,n);
```

x, y – punkty, w które wpisywany jest wielomian,
n – stopień wielomianu
coeff - nazwa wektora z otrzymanymi współczynnikami wielomianu

W omawianym przypadku wybieramy wielomian stopnia pierwszego (*n=1*) oraz wektory *xw* i *yw*. Rysunek 15.4 przedstawia fragment skryptu realizujący procedurę aproksymacji.

```
29 -    coeff=polyfit(xw,yw,1);
30 -    fprintf('a= %4.2f \n',coeff(1,1));
31 -    fprintf('b= %4.2f \n',coeff(1,2));
32
33 -    yf=polyval(coeff,xw);
34
35 -    plot(xw,yf,'--b');
```

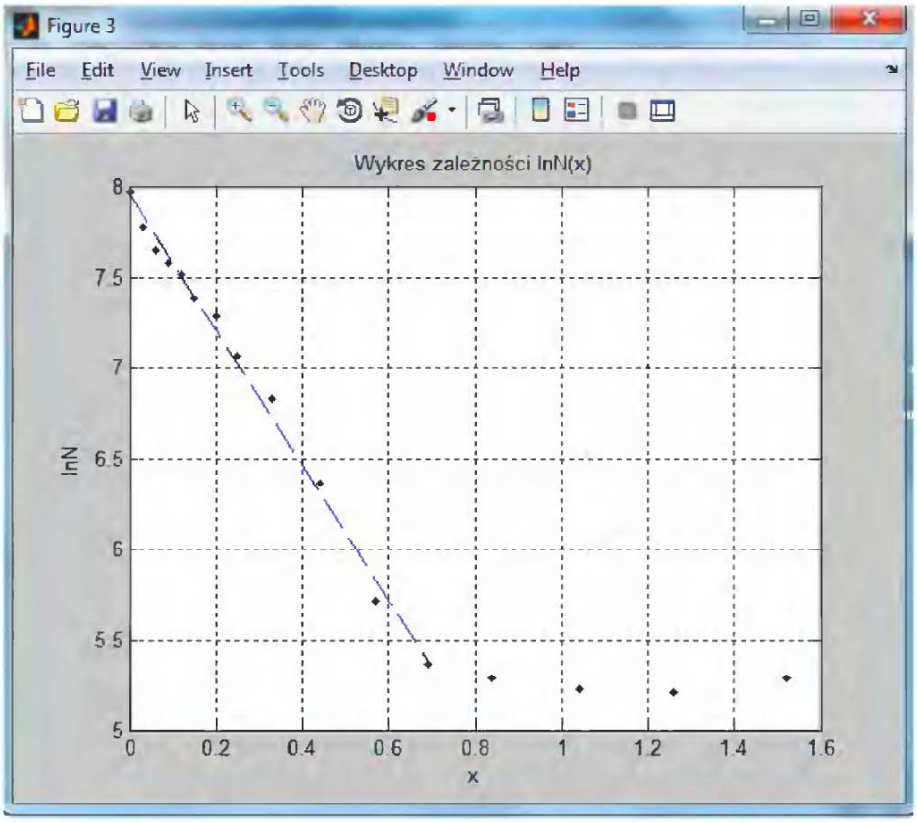
Rys. 15.4. Fragment skryptu realizujący aproksymację wielomianem pierwszego stopnia

Należy podkreślić, że zwrócone przez funkcję **polyfit** współczynniki wielomianu, znajdujące się w macierzy *coeff* ustawione są w kolejności począwszy od współczynnika przy najwyższej potędze wielomianu a kończąc na wyrazie wolnym. Zatem w rozważanym przypadku element wektora *coeff(1,1)* to współczynnik *a*, natomiast element *coeff(1,2)*, to współczynnik *b*. Polecenia znajdujące się w liniach 30 i 31 powodują wyświetlenie wartości współczynników w oknie poleceń.

Listing fragmentu skryptu przedstawiony na rysunku 15.4 zawiera jeszcze jedno istotne polecenie, mianowicie:

yf=polyval(coeff,xw)

Powoduje ono obliczenie wartości *yf* dla wektora *xf* w oparciu o wzór funkcji, której wyznaczone współczynniki znajdują się w macierzy *coeff*. Polecenie to jest potrzebne m. in. do narysowania wykresu wpisanej prostej. Ostateczny wynik aproksymacji przedstawia rysunek 15.5. Listing skryptu wykonującego aproksymację przedstawiono na rysunku 15.6.



Rys. 15.5. Końcowy wynik aproksymacji dla wybranych punktów doświadczalnych

c18.m*

```
clear all;

x=[0 0.03 0.06 0.09 0.12 0.15 0.2 0.25 0.33 0.44 0.57 0.69 0.84 1.04 1.26 1.52];
N=[2380 2391 2088 1953 1830 1605 1458 1170 924 582 303 213 198 186 183 198];

lnN=log(N);

plot(x,lnN,'*k');

xlabel('x');
ylabel('lnN');

title('Wykres zależności lnN(x)');
grid on;

xw=x(1,1:12);
yw=lnN(1,1:12);

figure(2)
plot(xw, yw, '*k');
xlabel('xw');
ylabel('yw');
grid on;

title('Wykres zależności lnN(x) wybranych 12 punktów');
hold on;

coeff=polyfit(xw,yw,1);
fprintf('a= %4.2f \n',coeff(1,1));
fprintf('b= %4.2f \n',coeff(1,2));

yf=polyval(coeff,xw);

plot(xw,yf,'--b');

hold off;

figure(3)
plot(x,lnN,'.k');
hold on;
plot(xw,yf,'--b');
xlabel('x');
ylabel('lnN');
grid on;

title('Wykres zależności lnN(x)');
```

Rys. 15.6. Listing skryptu wykonującego aproksymację

W dalszej części ćwiczenia przedstawione zostaną rezultaty aproksymacji wielomianami różnego stopnia pod kątem najlepszego dopasowania krzywej do wyników pomiarów obarczonych określoną niepewnością pomiarową.

Tabela 15.2 przedstawia pomiary uzyskane w trakcie doświadczenia badającego cząstki elementarne. Przedmiotem badań są zderzenia sprężyste ujemnie naładowanych mezonów K z protonami, przy ustalonej energii mezonu K. Wyniki pomiarów równe są liczbie zderzeń (wielkość bezwymiarowa), przy których cosθ (cosinus kąta rozpraszania) przyjmował wartość należącą do małego przedziału wokół $t_j = \cos\theta_j$. Jako błędy pomiarowe założono wartości błędów statystycznych, tzn. wartości pierwiastków kwadratowych z liczby obserwacji [2].

Tabela 15.2. Dane pomiarowe. Zakładamy $\sigma_j = \sqrt{y_j}$ (źródło: [1])

j	t_j=cosθ$_j$	y_j
1	-0.9	81
2	-0.7	50
3	-0.5	35
4	-0.3	27
5	-0.1	26
6	0.1	60
7	0.3	106
8	0.5	189
9	0.7	318
10	0.9	520

Zadanie polega w pierwszej kolejności na wykonaniu wykresów punktowych w oparciu o dane z tabeli 15.2 wraz z zaznaczonymi słupkami błędów. W tym celu należy użyć polecenia:

```
errorbar(x,y,sig,'.k')
```

zamiast polecenia *plot*. Składnia jest podobna z tą różnicą, że w powyższym poleceniu dodaje się wartość odpowiadającą słupkom błędu dla poszczególnych punktów (x,y). Rozmiary wektorów *x, y* jak i *sig* muszą być jednakowe. Następnie należy przedstawić na oddzielnych wykresach wyniki dopasowania wielomianów następującego stopnia:

- funkcja liniowa,
- parabola,
- wielomian stopnia 3,
- wielomian stopnia 4.

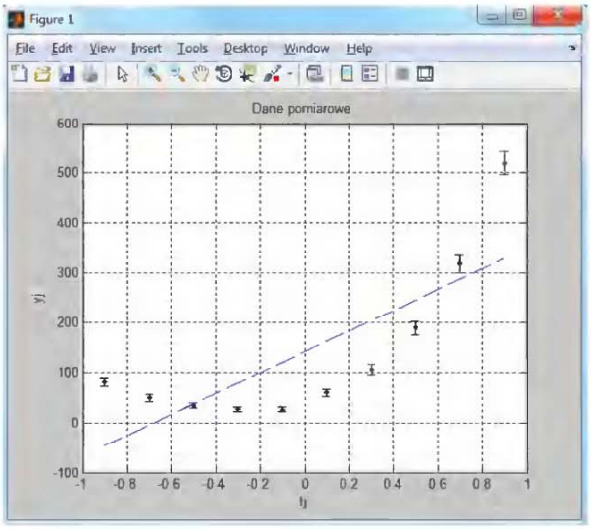
Po wykonaniu aproksymacji należy sprawdzić, które z wpisanych wielomianów dają wynik zgodny z danymi doświadczalnymi (dane wraz z zaznaczonymi błędami powinny przecinać dopasowane krzywe) a następnie odpowiedzieć na pytanie, jaki jest najniższy rząd wielomianu poprawnie opisującego przedstawione dane doświadczalne.

Rysunek 15.7 przedstawia listing skryptu realizującego rozważany przykład. Rezultaty aproksymacji wielomianami kolejnych stopni przedstawione zostały na rysunku 15.8 a – d.

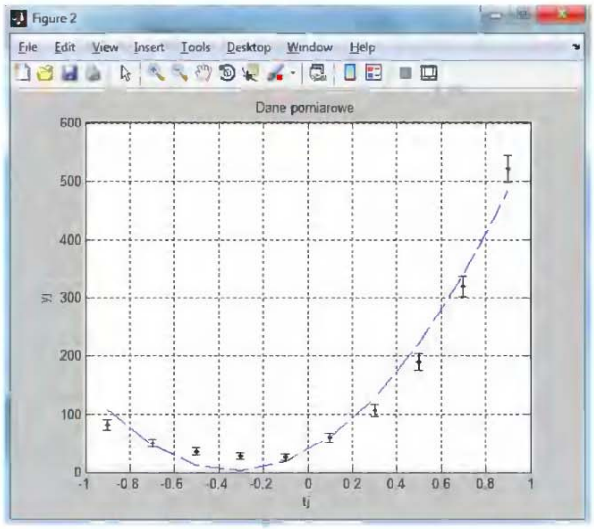
```
clear all;
close all;

tj=[-0.9 -0.7 -0.5 -0.3 -0.1 0.1 0.3 0.5 0.7 0.9];
yj=[81 50 35 27 26 60 106 189 318 520];

sig=sqrt(yj);

%plot(tj,yj,'.')

figure(1)
errorbar(tj,yj,sig,'.k');
hold on;

p=polyfit(tj,yj,1);

yn=polyval(p,tj);
```

```
plot(tj,yn,'--b');

xlabel('tj');
ylabel('yj');
title('Dane pomiarowe ');
grid on;

figure(2)
errorbar(tj,yj,sig,'.k');
hold on;

p=polyfit(tj,yj,2);

yn=polyval(p,tj);
plot(tj,yn,'--b');

xlabel('tj');
ylabel('yj');
title('Dane pomiarowe ');
grid on;

figure(3)
errorbar(tj,yj,sig,'.k');
hold on;

p=polyfit(tj,yj,3);

yn=polyval(p,tj);
plot(tj,yn,'--b');

xlabel('tj');
ylabel('yj');
title('Dane pomiarowe ');
grid on;

figure(4)
errorbar(tj,yj,sig,'.k');
hold on;

p=polyfit(tj,yj,4);

yn=polyval(p,tj);
plot(tj,yn,'--b');

xlabel('tj');
ylabel('yj');
title('Dane pomiarowe ');
gridon;
```

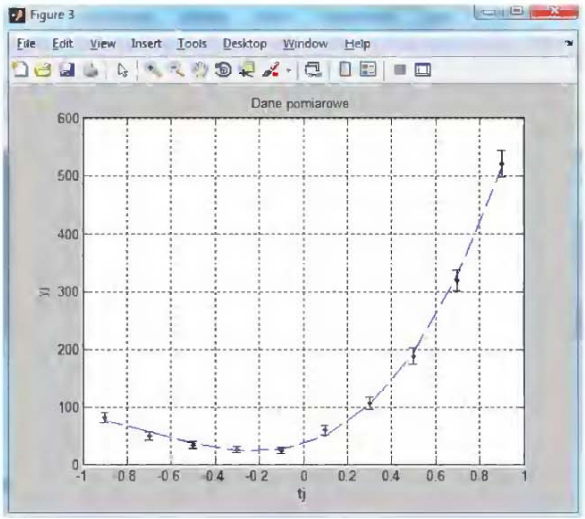
Rys. 15.7. Listing przedstawiający skrypt aproksymujący wielomianami różnego stopnia



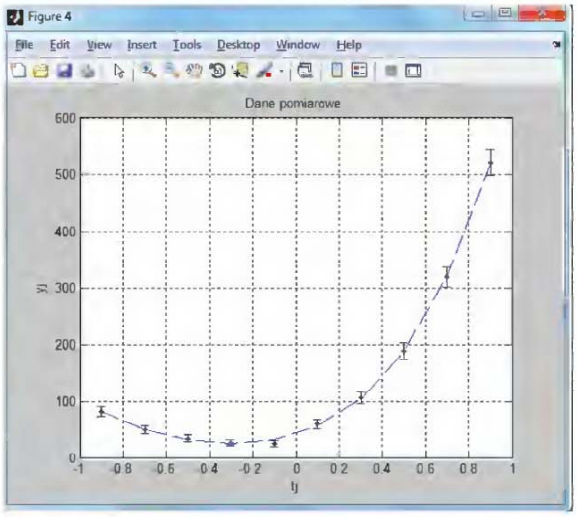
Rys. 15.8a. Wynik aproksymacji *n=1*



Rys. 15.8b. Wynik aproksymacji *n=2*



Rys. 15.8c. Wynik aproksymacji *n=3*



Rys. 15.8d. Wynik aproksymacji *n=4*

Z wykresów przedstawionych na rysunkach 15.8.a-d wynika, że rezultaty dwóch pierwszych aproksymacji (n=1 i n=2) są niezadowalające. Wpisana krzywa przebiega w znacznej odległości od większości punktów pomiarowych wraz z zaznaczonymi słupkami niepewności pomiarowych. Dwie kolejne natomiast dają już wynik zadowalający. Można więc stwierdzić, że w celu poprawnego opisu danych doświadczalnych należy użyć wielomianu o stopniu co najmniej równym 3.

ZADANIA DO WYKONANIA

Zadanie 1.

Na podstawie danych w tabeli 15.3 oraz poniższego wzoru wyznaczyć wynik stałej sieci (a_o) dla krzemu.

$$f(\theta) = 0{,}5 \cdot \left(\frac{\cos^2 \theta}{\sin \theta} + \frac{\cos^2 \theta}{\theta} \right)$$

Wartości parametru a obliczone na podstawie d_{hkl} dla różnych płaszczyzn (hkl), a więc różnych kątów θ_i przedstawiamy na wykresie jako rzędne, a odpowiadające im wartości $f(\theta_i)$ – jako odcięte i przez te punkty prowadzimy prostą. Najdokładniejsza wartość parametru jest w punkcie $f(\theta)=0$. Wartość tę należy wyznaczyć stosując aproksymację liniową. Należy pamiętać, że argumenty funkcji sinus czy cosinus podaje się w radianach.

Tabela 15.3. Dane do zadania na wyznaczenie stałej sieci dla krzemu

Substancja	Kąt θ_i	(hkl)	a[A]	a_o[A]
Si	20,6	(111)	5,63	
	35,1	(220)	5,64	
	42,2	(311)	5,65	
	53,9	(400)	5,67	
	61,8	(331)	5,67	

Literatura

1. Brandt S. Analiza danych. Warszawa, PWN, 2002.
2. Kamińska A, Pańczyk B. Ćwiczenia z Matlab. Przykłady i zadania. Warszawa, MIKOM, 2002.
3. Zalewski A, Cegieła R. Matlab. Obliczenia numeryczne i ich zastosowanie. Poznań, Wydawnictwo Nakom, 1997.
4. Krzyżanowski P. Obliczenia inżynierkie i naukowe. Warszawa, PWN, 2011.