Gliwice, 9.02.2018 r.

Laboratorium

Programowania Komputerów 3

Temat:

Projekt – Biblioteka numeryczna

do rozwiązywania układów liniowych

Mirosław Ściebura

Informatyka, semestr 3, grupa nr 3

Rok akademicki: 2017/2018

Prowadzący: Piotr Pecka

1. Temat programu

Temat podany przez prowadzącego brzmiał następująco:

Projekt nr 3. Napisać bibliotekę numeryczną do rozwiązywania układów równań liniowych. Biblioteka ma operować na obiektach klasy macierz, które tworzone są dynamicznie. Algorytmy (solvery) liczące układ równań powinny być podzielone na 2 grupy: iteracyjne (Jackobiego i Gaussa- Seidl'a) oraz ogólne – metoda Gaussa.

2. Analiza i projektowanie

2.1. Dobór algorytmu i struktury danych, ograniczenia specyfikacji

W związku z wymaganym paradygmatem obiektowym programu, strukturą danych w której będziemy przechowywać dane (elementy macierzy, jej wymiary) będzie klasa macierz. Klasa zawiera pola:

- lwierszy – liczba wierszy macierzy przechowywana w zmiennej typu int,

- lkolumn – liczba kolumn macierzy, podobnie jak w przypadku lwierszy przechowuje się ją w zmiennej typu int,

- pole – podwójny wskaźnik na zmienną typu double (gdyż potrzebujemy tablicy dynamicznej dwuwymiarowej liczb rzeczywistych).

Na bazie tej klasy powstaną bazy potomne: macierzkwadratowa (odpowiadająca macierzy $n \times n$) i wektor (odpowiadająca macierzy $n \times 1$). Będą posiadały one dokładnie te same pola, jednak konstruktory pozwolą na konstrukcję tylko z tą samą liczbą wierszy i kolumn (macierzkwadratowa) i z liczbą kolumn równą 1 (wektor). Ich zróżnicowanie potrzebne jest w celu przypisania różnych metod działania na nich.

Będziemy wykorzystywać algorytmy narzucone przez temat projektu, czyli algorytmy iteracyjne (Jackobiego i Gaussa- Seidl'a) oraz ogólny, czyli metodę Gaussa. Algorytmy iteracyjne polegają na odpowiednim tworzeniu i przemnażaniu macierzy A (macierzy współczynników) i b (macierzy wyrazów wolnych). Kolejne przybliżenia wyniku są realizowane poprzez operacje macierzowe na przybliżeniu aktualnym. Algorytmy te mogą dawać wyniki niepoprawne w przypadku braku spełnienia warunku zbieżności (macierz A musi być przekątniowo dominująca). Metoda Gaussa jest metodą ogólną polegającą na sprowadzeniu macierzy do postaci trapezowej i obliczania wartości kolejnych zmiennych. Jest to metoda która w przypadku podania wyniku (układ równań oznaczony) da zawsze poprawne wyniki.

W przypadku macierzy, w której nie da się stwierdzić czy podane dane spełniają warunek dominacji zostanie wyświetlony komunikat, w którym zasugerujemy wykonanie metody ogólnej, jednak sam algorytm zostanie wykonany. Program też będzie sprawdzał czy równania są liniowo niezależne (ustalenie określoności) i czy nie ma sprzeczności.

2.2. Analiza problemu, podstawy teoretyczne

Program będzie operował na macierzach i posługiwał się metodami odzwierciedlającymi operacje na nich. Ważne jest również pojęcie układu równań jako iloczynu dwóch macierzy, którego wynikiem jest macierz wyrazów wolnych.

Macierzą o n wierszach i m kolumnach nazywamy dowolną funkcję:

$$A: \{1,2,\dots,n\} \times \{1,2,\dots,m\} \to R.$$

Macierze utożsamiamy z prostokątnymi tablicami liczb:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nm} \end{bmatrix} = \left[a_{ij}\right]_{n\times m}.$$

Liczby a_{ij} nazywamy elementami macierzy A.

Macierz kwadratowa to macierz o tej samej liczbie wierszy i kolumn.

Wektor to macierz o liczbie kolumn równej 1.

Układ:

$$Ax = B,$$

gdzie:

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{n1} & \cdots & a_{nn} \end{bmatrix} B = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

nazywamy układem n równań liniowych z n niewiadomymi (macierz A jest macierzą kwadratową, B i x wektorami).

Możemy wykonywać operację matematyczne na macierzach, jednak dla niektórych muszą być spełnione odpowiednie warunki:

- dodawanie – macierze muszą mieć identyczne liczby wierszy i kolumn,

- mnożenie – operacja nieprzemienna, ilość kolumn macierzy lewej musi być równa ilości wierszy macierzy prawej,

- odwracanie (rozpatrujemy jedynie przypadek macierzy diagonalnej – elementy niezerowe tylko na głównej przekątnej) – macierz musi być kwadratowa, elementy na głównej przekątnej różne od 0.

Będziemy także mnożyć przez skalar, co nie wymaga dodatkowych warunków (wszystkie elementy mnożone przez określoną liczbę).

Ważnym jest też określenie, czy macierz jest dominująca co dla kwadratowej realizujemy przez odpowiednią zamianę wierszy macierzy A i b oraz sprawdzenie czy moduł elementu na głównej przekątnej jest większy od sumy modułów pozostałych elementów.

3. Specyfikacja zewnętrzna

Program jest uruchamiany poprzez uruchomienie odpowiedniego pliku wykonywalnego (Projekt – Układy równań.exe). Dane wejściowe są wczytywane z klawiatury. W zależności od etapu programu są to dowolne klawisze (przejście do następnego etapu) albo liczby całkowite lub zmiennoprzecinkowe (liczby te mają ograniczone wartości przez wartości maksymalne typów int i double). Wpisywanie wartości jest poprzedzone odpowiednim komunikatem i zabezpieczone przed na błędy wejścia/wyjścia (wpisywanie wartości niedodatnich w liczbie kolumn i wierszy, wpisywanie liter i innych znaków, zgodnie ze specyfiką strumienia cin białe znaki są pomijane i liczby po nich są pomijane). W przypadku błędu jest wyświetlany komunikat o nim. W przypadku wyboru algorytmu, wejście jest zabezpieczone na wybór innego klawisza niż „1”, „2” lub „3”.

W związku ze specyfiką algorytmu sprawdzana jest dominacja przekątniowa macierzy. W przypadku braku spełnienia tego warunku przez dane wejściowe wyświetlany jest komunikat o ewentualnej niezgodności wyników ze stanem rzeczywistym dla rozwiązań iteracyjnych i sugestii zastosowania metody ogólnej, jednak sam algorytm zostanie wykonany. Po wykonaniu zostaną zwrócone wyniki danego algorytmu.

4. Specyfikacja wewnętrzna

Projekt składa się z 3 plików: liczba.h, klasa.h, klasa.cpp, source.cpp.

Plik liczba.h jest plikiem nagłówkowym zawierającym definicję szablonu klasy liczba. Szablon ten jest zaimplementowany ze względu na potrzebę wczytywania liczb całkowitych i zmiennoprzecinkowych odpornych na błędy wejścia/wyjścia. Ze względu na działanie linkera w kompilatorze, metody operujące na tej klasie musiały być zdefiniowane właśnie tutaj. Klasa liczba jest zaprzyjaźniona z klasami macierz, macierzkwadratowa i wektor. Zaprzyjaźnienie to zostało wykorzystane ze względu na potrzebę prostego dostępu do jedynego pola klasy wartość (o określonym typie). Działanie klasy zostało dostosowane jedynie do typów danych int i double. Przy przekazywaniu elementów tego typu zostanie

wykorzystana referencja do liczby z const ze względu na brak lokalnej kopii elementu i brak możliwości zmiany przyjętego.

Szablon klasy liczba zawiera metody:

- konstruktory liczba(), liczba(char*), liczba(const liczba<T>&) oraz destruktor ~liczba(). Są to kolejno konstruktory bezargumentowy (ustalający wartość na 0), z komunikatem (realizuje wpis z odpowiednim komunikatem) i kopiujący (zwyczajnie kopiuje wartość tego samego typu). Destruktor realizowany jest przez system i usuwa zmienną klasową i cały obiekt.

- operatory przypisania (liczba& operator=(const liczba<T> &) – zwraca referencje do wywołanego elementu, jedynie przepisuje wartość), różnowartościowy (bool operator!=(const liczba<T> &) – zwraca wartość logiczną czy jest różny) oraz 2 operatory jedynie dla typu int: dodawania (liczba<int> operator+(int i) – zwracający nową liczbę która ma wartość większą o podaną liczbę od wywołanej) i prefiksowy inkrementacji (liczba<int> operator+(int i) – zwiększa wartość o jeden),

- metodę void wpis(char*), która odczytuje w pętli dane podaną z klawiatury i wpisuje ją do wartości liczby w przypadku braku błędów I/O, w przeciwnym wypadku czyści bufory klawiatury (cin.sync() i cin.ignore(1000, '\n') – zgodność z Linuksem) oraz flagę błędu cin.fail(),

- metodę bool odwrotnosc(), która wpisuje liczbę odwrotną do wartości przechowywanej i zwraca komunikat o powodzeniu (przypadek liczby 0),

- metodę bool niedodatnia(), sprawdzającą niedodatniość liczby (wczyt liczb dodatnich) i zwraca informację o tym,

- metodę void zamien(liczba<T>&) zamieniającą dwie liczby (ze względu na zamianę z podaną liczba podana musi mieć referencję bez const),

- zaprzyjaźnione funkcje operatorowe friend std::istream& operator >> (std::istream& is, liczba<T>& t) i friend std::ostream& operator<<(std::ostream& os, const liczba<T>& t), które wpisują lub wypisują wartość liczby przy użyciu określonego strumienia.

Operatory matematyczne nie zostały zdefiniowane ze względu na brak możliwości określenia przez kompilator typu (moglibyśmy zrobić szablony dla konkretnych typów, jednak chcemy uniknąć nadmiaru kodu). Dzięki zaprzyjaźnieniu klasy z klasą macierz i pochodnymi otrzymujemy bezpośredni dostęp do wartości liczbowych. Do tego pliku nagłówkowego musimy załączyć biblioteki <iostream> i <Windows.h> ze względu na wpis liczby i odpowiednie oczekiwanie w jego trakcie.

Plik nagłówkowy klasa.h zawiera definicje klasy macierz oraz pochodnych. Zawiera także deklaracje wszystkich potrzebnych metod, których definicje zamieszczone są w pliku liczba.cpp. Wykorzystuje plik nagłówkowy liczba.h. Zawiera następujące klasy:

- macierz, która zawiera pola liczba<double> **pole (pełni funkcje dwuwymiarowej tablicy liczb typu double wczytanych z zabezpieczeniem na błędy I/O) oraz liczba<int> lwierszy i liczba<int> lkolumn (wymiary macierzy wczytane z zabezpieczeniem na błędy I/O),

- macierzkwadratowa – klasa potomna klasy macierz, zawiera te same pola, lecz ma kilka własnych metod i musi mieć tą samą liczbę wierszy i kolumn,

- wektor – klasa potomna klasy macierz, zawiera te same pola, lecz liczba jej kolumn musi być równa 1.

Zmienne klasowe klasy macierz muszą być protected ze względu na potrzebę dostępu do nich w klasach potomnych.

Klasa ta zawiera wiele metod używanych także w klasach potomnych. W przypadku przekazywania obiektów posługujemy się referencją z const, gdyż nie chcemy tworzyć lokalnej kopii ani zmieniać przekazanego argumentu. Metody używane przez te klasy:

- konstruktory macierz() (bezargumentowy, który ustawia pole jako nullptr, liczby i tak zostaną zainicjalizowane zerami), macierz(const liczba<int>&, const liczba<int>&) (otrzymuje 2 liczby typu int które określą wymiary i alokuje odpowiednio pole – inicjalizowane zerami), macierz(const macierz &) (konstruktor kopiujący, kopiuje wymiary i alokuje odpowiednią tablicę i kopiuje do niej wartości elementów macierzy) oraz destruktor ~macierz() (musi być określona dealokacja pola, zmienne liczba zostaną zostaną zdekonstruowane przez system),

- operatory indeksowania liczba<double>* operator[](int) oraz const liczba<double>* operator[](int) const – operatory indeksacji macierzy zwracające adres do konkretnego wiersz (potem możemy przeszukiwać jako tablicę elementów liczba). Drugi operator ze względu na przekazywanie do metod elementów z const.

- operator przypisania macierz& operator=(const macierz&), który zwraca referencję do wywołanego obiektu. Sprawdza warunek przypisania do samego siebie. Kopiuje wymiar macierzy. W zależności od niego realokuje pole (wcześniej dealokując aktualne) i przepisując do niego wartości.

- metody bool dodajdomnie(const macierz &) void mnozskalar(double) bool mnozprawo(const macierz&) bool domnozzlewej(const macierz &) będące odpowiednikami operacji na macierzach (dodawanie, mnożenie przez skalar, mnożenie macierzy z obu stron). W przypadku operacji stricte macierzowych zwracamy informację czy można było przeprowadzić (w przeciwnym wypadku wywołany obiekt się nie zmienia). Wyniki działania zapisane są w wywołanym obiekcie (optymalizacja pamięci). Przy

mnożeniach macierzowych następuje realokacja pamięci (przechowanie aktualnego pola w zmiennej tmp). Po zakończeniu działania elementy są przepisywane a tmp dealokowane.

- metody bool zamianawierszy(wektor&) i bool robtrapezowa(int) zamieniające kolejność wierszy macierzy, a w przypadku rob trapezowa odpowiednie odejmowanie wierszy od siebie. Ma to na celu przygotowanie macierzy do dalszych operacji i określenie czy na danych w ogóle można przeprowadzić poprawne operacje (zwrot bool). W zamiana wierszy przekazany argument to wektor wyrazów wolnych który również musi być odpowiednio zamieniony (referencja bez const).

- metody I/O void wpis() i void wypisz() służące do wczytu i wypisu na ekran danych z macierzy. Wpis opiera się na wpisie z klasy liczba wraz z odpowiednim komunikatem (otrzymywanym na podstawie wartości iteratorów pętli – wiersz i kolumna, wykorzystujemy operacje na ciągach znaków w stylu c – funkcje strcat i _itoa – wymagana dyrektywa preprocesora _CRT_SECURE_NO_WARNINGS).

Kolejną klasą jest macierz kwadratowa dziedzicząca publicznie z macierz. Nie zawiera dodatkowych pól, jedynie dodatkowe metody. Użyte konstruktory i metody to:

- macierzkwadratowa(), macierzkwadratowa(const liczba<int>&), macierzkwadratowa(const macierzkwadratowa&) oraz destruktor ~macierzkwadratowa() – wykorzystują odpowiadające konstruktory macierz (macierzkwadratowa(const liczba<int>&) wykorzystuje macierz(const liczba<int>&, const liczba<int>&) podstawiając 2 razy ta samą liczbę)

- metody macierzkwadratowa zrobL(), macierzkwadratowa zrobD(), macierzkwadratowa zrobU(), macierzkwadratowa zrobLU() tworzące macierze diagonalne górno- lub dolnotrójkątne na podstawie kwadratowej (macierz współczynników A) i zwracające je.

- metody matematyczne bool odwrocD() i bool dominacja(wektor&). Odwaracanie jest ograniczone do macierzy diagonalnej (odwrócenie elementów na przekątnej – może się nie udać jeżeli jest 0). W dominacja sprawdzany jest warunek dominacji przekątniowej macierzy kwadratowej (moduł elementu na głównej przekątnej większy od sumy modułów pozostałych elementów macierzy – potrzebujemy sprawdzić w którym wierszu który element dominuje, w przypadku gdy w każdym jest to jeden element i dla wszystkich wierszy – zamiana). Zwracają informację czy się udało.

Następnie mamy trzy metody które odpowiadają poszczególnym algorytmom: Jackobiego, Gaussa-Seidla czy metodzie Gaussa):

- bool Jackobi(const wektor&, wektor&) – metoda do obliczenia metodą Jackobiego. Otrzymuje jako pierwsze wektor wyrazów wolnych b (nie zmieniamy go tak samo jak macierzy na rzecz której wywołaliśmy) i wektor referencje do wektora który będziemy zmieniali (początkowo są to same 0). Algorytm działa na zasadzie wykonywania określoną ilość razy (zmienna iteracja odczytana z klawiatury) operacji $x^{n+1} = Mx^n + Nb$. Na początku

kopiujemy wartości macierzy A i b i sprawdzamy czy są dominujące. W przypadku powodzenia obliczamy macierze $D^{-1} = N$ oraz $M = -D^{-1}(L + U) = -N(L + U)$. W przypadku niepowodzenia którejś z operacji zwracany jest komunikat o niepowodzeniu algorytmu. Następnie rządaną operację wykonujemy w pętli zapisując wyniki w macierzy wynikowej. Po wykonaniu pętli zwracamy info o powodzeniu.

- bool GaussSeidel(const wektor&, wektor &) - metoda do obliczenia metodą Gaussa-Seidla'a. Otrzymuje jako pierwsze wektor wyrazów wolnych b (nie zmieniamy go tak samo jak macierzy na rzecz której wywołaliśmy) i wektor referencje do wektora który będziemy zmieniali (początkowo są to same 0). Algorytm działa na zasadzie wykonywania określoną ilość razy (zmienna iteracja odczytana z klawiatury) operacji $x^{n+1} = D^{-1}b - D^{-1}Lx^{n+1} - D^{-1}Ux^{n}$. Na początku kopiujemy wartości macierzy A i b i sprawdzamy czy są dominujące. Potem obliczamy wartości macierzy pomocniczych a następnie w pętli wartości odpowiednio przemnożonych kolejnych współczynników. Po wykonaniu pętli zwracamy info o powodzeniu.

- bool metGauss(const wektor&, wektor &) - metoda do obliczenia metodą Gaussa. Otrzymuje jako pierwsze wektor wyrazów wolnych b (nie zmieniamy go tak samo jak macierzy na rzecz której wywołaliśmy) i wektor referencje do wektora który będziemy zmieniali (początkowo są to same 0). Nie jest to metoda iteracyjna więc zwraca od razu dobry wynik. Początkowo tworzy macierz [A|b] i na niej tworzy postać trapezową. W przypadku niepowodzenia zwraca informację o błędzie i kończy działanie. Następnie sprawdza czy da się podzielić odpowiednie współczynniki macierzy przez siebie (z uwzględnieniem z sumowania współczynników z poprzednio obliczonymi zmiennymi). W przypadku niemożności zwracana informacja o układzie sprzecznym lub nieoznaczonym. Jeżeli udało się to dla wszystkich wierszy zwracamy informację o powodzeniu.

Ostatnią klasą jest wektor dziedzicząca publicznie z macierz. Nie zawiera dodatkowych pól i metod. Użyte konstruktory to:

- wektor(), wektor(const liczba<int>&) wektor(const macierz &) oraz destruktor ~wektor() - wykorzystują odpowiadające konstruktory macierz (wektor(const liczba<int>&) przypisuje przekazaną liczbę jako liczbę wierszy a liczbę kolumn inkrementuje (inicjalizacja zerem) a następnie alokuje odpowiednią tablicę pole (elementy inicjalizowane zerami)).

Dziedziczenia zostały użyte ze względu na precyzyjne określenie typu macierzy i brak możliwości przypisania złego typu.

$$\begin{array}{c} macierz \\ \swarrow \quad \searrow \\ wektor \quad macierzkwadratowa \end{array}$$

Diagram obrazujący dziedziczenie w projekcie.

W pliku source.cpp znajduje się odpowiednie wywołanie metod z biblioteki. Są tu zmienna liczba o typie int n do wczytu liczby kolumn i wierszy macierzy, zmienne obiektowe A, B oraz wynik(odpowiadające odpowiednim macierzom) zmienna znakowa znak do wyboru działania programu a także zmienna logiczna powodzenie określająca powodzenie działa algorytmu (czy wynik w zmiennej wynikowa jest dobry). Umieszczone tu są również komunikaty dotyczące działania programu.

5. Testowanie

Program był testowany na błędy wejścia/wyjścia (wprowadzanie liter zamiast cyfr, liczb niedodatnich przy odczycie ilości wierszy lub kolumn, wprowadzanie ciągów cyfr i liter, wprowadzanie znaków innych niż żądane przy wyborze algorytmu).

Działanie samych algorytmów było działanie przy użyciu poniższych danych:

Dla podanych danych:

$$A = \begin{bmatrix} 4 & -1 & -0.2 & 2 \\ -1 & 5 & 0 & -2 \\ 0{,}2 & 1 & 10 & -1 \\ 0 & -2 & -1 & 4 \end{bmatrix} \; b = \begin{bmatrix} 3 \\ 0 \\ -10 \\ 5 \end{bmatrix}$$

program zwrócił wyniki:

- dla metody Jackobiego (liczba iteracji: 2): $\begin{bmatrix} 6{,}825 \\ 2 \\ -1{,}025 \\ 1 \end{bmatrix}$,

- dla metody Gaussa-Seidla (liczba iteracji: 2): $\begin{bmatrix} 6{,}9725 \\ 2{,}0645 \\ -1{,}1784 \\ 1{,}98765 \end{bmatrix}$,

-dla metody Gaussa (nieiteracyjnej): $\begin{bmatrix} 0{,}194751 \\ 0{,}557436 \\ -0{,}930017 \\ 1{,}29621 \end{bmatrix}$.

Wynik z ostatniego algorytmu jest wynikiem dokładnym. Rozbieżności z metodami iteracyjnymi wynikają z małej liczby iteracji.

Dla danych:

$$A = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \; b = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

program zwrócił wyniki:

- dla metody Jackobiego (liczba iteracji: 10): $\begin{bmatrix}2\\3\end{bmatrix}$,

- dla metody Gaussa-Seidla (liczba iteracji: 10): $\begin{bmatrix}2\\3\end{bmatrix}$,

-dla metody Gaussa (nieiteracyjnej): $\begin{bmatrix}2\\3\end{bmatrix}$.

Za każdym razem wynik równa się wynikowi faktycznemu. W tym dość prostym przykładzie udowodniliśmy, że program radzi sobie z odpowiednim przekładaniem wierszy macierzy gdy została podana jako niedominująca przekątniowo, natomiast może spełnić ten warunek. Wyświetla się komunikat mówiący o braku dominacji (co pociąga możliwość złych wyników).

Dla takich danych:

$$A = \begin{bmatrix}2 & 1\\4 & 2\end{bmatrix}\ b = \begin{bmatrix}7\\10\end{bmatrix}$$

program zwrócił wyniki:

- dla metody Jackobiego (liczba iteracji: 10): $\begin{bmatrix}50\\-100\end{bmatrix}$,

- dla metody Gaussa-Seidla (liczba iteracji: 10): $\begin{bmatrix}102{,}5\\-200\end{bmatrix}$,

-dla metody Gaussa (nieiteracyjnej): Układ sprzeczny.

Układ faktycznie był sprzeczny. Przy zmianie liczby iteracji można było zauważyć, że przy coraz większych wartościach iteracji wyniki zmierzają do $\pm\infty$. Program poinformował o braku dominacji przekątniowej, czyli możliwości złego wyniku.

Dla ostatniego zestawu danych

$$A = \begin{bmatrix}-6 & 3\\-8 & 4\end{bmatrix}\ b = \begin{bmatrix}9\\12\end{bmatrix}$$

program zwrócił wyniki:

- dla metody Jackobiego (liczba iteracji: 10): $\begin{bmatrix}0\\0\end{bmatrix}$,

- dla metody Gaussa-Seidla (liczba iteracji: 10): $\begin{bmatrix}-1{,}5\\0\end{bmatrix}$,

-dla metody Gaussa (nieiteracyjnej): Układ nieoznaczony.

Dla podanego układu nieoznaczonego program również zwrócił informację o braku dominacji przekątniowej, co może wskazywać na niepoprawność wyników.

6. Wnioski

Wykonanie tego projektu pomogło w zrozumieniu paradygmatu projektowania obiektowego. We wcześniejszych wersjach programu przy zamieszczeniu komunikatów o wejściu do danej metody lub operatora można było zauważyć, że przy deklaracji zmiennej obiektowej z przypisaniem wywoływany jest konstruktor kopiujący. Dzięki temu widać też było jak ważne jest używanie referencji przy przekazywaniu obiektów do metod lub funkcji (mniej kopii lokalnych). Ważne było też zrozumienie samego algorytmu działania metod i sprawdzanie wszelkich warunków działania programu (błędy wejścia/wyjścia, przekątniowość podanej macierzy. Pokazało to też, że szablon należy definiować w pliku nagłówkowym. Ważne było też zastosowanie dziedziczenia (rozpoznawanie macierzy jednokolumnowych i kwadratowych) oraz zaprzyjaźnienia (odczyt zmiennych klasy liczba). Ważnym było też zrozumienie kiedy należy zwracać referencję do obiektu przy operatorze, a kiedy obiekt.