AKADEMIA GÓRNICZO-HUTNICZA

Dokumentacja do projektu

Biblioteka do obsługi macierzy

z przedmiotu

Języki Programowania Obiektowego

EiT, 3 rok

Mateusz Szych

Piątek, 11:30

prowadzący: Jakub Zimnol

07.01.2025

1. Wstęp

Biblioteka pozwala na tworzenie kilku wersji macierzy (Matrix, SquareMatrix oraz IdentityMatrix) oraz wykonywanie na nich podstawowych operacji. Obiekty tych klas mogą przyjmować różne typy danych. Macierze wspierają operacje na liczbach zespolonych. Biblioteka może zostać użyta w projektach wykorzystujących operacje na macierzach.

2. Opis funkcjonalności

Dostępne są 3 klasy: **Matrix**, dziedziczący po Matrix **SquareMatrix** i **IdentityMatrix** (macierz jednostkowa). Dwie ostatnie są do użytku w bardziej specyficznych przypadkach, ze względu na pewne ograniczenia. Macierz może zostać utworzona na 3 sposoby: poprzez podanie wymiarów, np. ***Matrix<double> m1(2,4)***, co spowoduje utworzenie macierzy 2x4 z domyślnymi wartościami 0. Można również podać początkową wartość jako trzeci argument. Istnieje też opcja utworzenia macierzy poprzez przekazanie wektora wektorów. Po utworzeniu obiektu można na nim wykonywać następujące operacje:

- Dodawanie (+, +=)
- Odejmowania (-,-=)
- Mnożenie (*,*=)
- Dzielenie (wyłącznie przez liczbę; /, /=)
- Porównanie (==, !=)
- Przypisanie wartości (=)
- Operator [] (***m1[n]*** – zwraca n-ty wiersz (zaczynając od 0), ***m1[n][m]*** – wartość na pozycji n,m)
- Operator << (do wyświetlania macierzy, np. ***cout<<m1***)
- Wyznacznik (***m1.det()***)
- Dodawanie oraz usuwanie wierszy/kolumn (***m1.addColumn(vector)***, ***m1.remColumn()***). ***remColumn()*** oraz ***remRow()*** zawsze usuwają ostatni wiersz lub kolumnę.
- Transpozycja (***m1.transpose()***). Transpozycja modyfikuje obiekt, nie zwraca kopii.

Macierze akceptują liczby zespolone jako typ, np. Matrix<ComplexNumber<double>> c1(3,3). Wszystkie opisane wyżej operacja działają na tym typie.

SquareMatrix działa prawie tak samo, jedynym wyjątkiem jest brak możliwości zmiany wymiarów macierzy po jej utworzeniu; funkcje modyfikujące wymiary (takie jak addRow()) są niedostępne. Przy mnożeniu (*=) liczba kolumn macierzy po prawej musi zgadzać się z liczbą kolumn macierzy po lewej, tak aby wynikiem była macierz kwadratowa.

Obiekt klasy **IdentityMatrix** po utworzeniu nie może zostać w żaden sposób zmodyfikowany; dostępne są jedynie funkcje pozwalające odczytać wartości. Przy tworzeniu podajemy wymiar (jeden argument); ***IdentityMatrix<double> i1(4)***.

3. Sposób wykorzystania oraz kompilacja

Przy potrzebie wykorzystania biblioteki należy dołączyć do projektu plik matrix.hpp oraz complex.hpp przy potrzebie korzystanie liczb zespolonych. Pliki te znajdują się w folderze „include”. W folderze „example” znajduje się plik main.cpp przestawiający przykładowe wykorzystanie biblioteki. „Nadrzędny” CMakeLists.txt znajduje się w głównym folderze projektu.

W celu uruchomienia main.cpp należy:

1. Windows (przy założeniu że zarówno cmake i make są zainstalowane w systemie oraz odpowiednio skonfigurowane – ja korzystam z Windows11 oraz narzędzi zainstalowanych za pomocą msys2)

mkdir build (w folderze projektu)

cd build

cmake .. -G „MinGW Makefiles” (bez -G używany jest ninja)

mingw32-make

2. Linux (u mnie Ubuntu 24.04)

mkdir build

cd build

cmake ..

make

Po wykonaniu tych operacji, plik wykonywalny powinien znajdywać się w build/example jako main_example.exe. Wykorzystanie CMake w tym przypadku jest jedynie „dla pokazu”, w razie problemów wystarczy skompilować main.cpp oraz uruchomić plik wynikowy: g++ main.cpp, ewentualnie skorzystać z wtyczki CodeRunner w VS Code.