Programowanie i metody numeryczne

Aleksandra Fijałkowska

6 kwietnia 2018

Zadanie 4. (1 p)
Znaleźć epsilon maszynowy dla zmiennych typu float i double. Wynik porównać z wartością otrzymaną ze statycznej metody epsilon w szablonie std::numeric_limits.

Zadanie 5. (4 p)
a). Napisz funkcję liczącą pochodną dwupunktową metodą centralną:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h} \tag{1}$$

Sprawdzić działanie programu dla dla $f = sin(x)$ dla $x = \frac{2\pi}{3}$ oraz $f = x^3$ dla $x = 2, 5$, wykorzystaj zmienne typu float.
Wskazówka: (przyda się w kolejnej części zadania)
Zaprojektuj program tak, aby każda część algorytmu znalazła się w osobnej funkcji np.:

```
float getFunValue(float x);\\
float getFunDerivative(float x);\\
float getFunDerivativeCentralMeth(float x, float h);\\
float getDerivativeError(float x, float h);\\
```

b). Zmodyfikuj program dla funkcji $f = x^3$ tak, aby w wygodny sposób umożliwiał sprawdzenie algorytmu dla zmiennych typu float i double. Funkcje przyjmujące zmienne typu float przepisz na szablony funkcji.
Wskazówka:
Przykładowy szablon funkcji, która zwraca kwadrat przyjętego argumentu:

```
template<typename T> T square(T x)
{
    return x*x;
}
```

c). Wyrysuj błąd metody w funkcji parametru h w skali log-log dla zmiennych typu float i double.

Zadanie 6. (2 p)
Napisz program sumujący liczby od 1 do 10000 zmiennymi int, float i double. Sprawdzić wynik sumując od przodu, od tyłu i metodą Kahana *https* : *//en.wikipedia.org/wiki/Kahan_summation_algorithm*3 .

Zadanie 7. (2 p)

Napisz program alokujący dynamicznie tablice liczb całkowitych o wczytanym rozmiarze. Następnie program powinien wyzerować podaną liczbę elementów z początku tablicy lub nie, w zależności od wyboru użytkownika. Po tym użytkownik zdecyduje, czy zwolnić zaalokowaną pamięć i następnie kiedy zakończyć program.

Zadanie 8. (3 p)

Napisz procedurę wczytującą liczby typu float z wejścia standardowego. Po wpisaniu pustej linii procedura powininna wyświetlić wczytane liczby na ekran w odwrotnej kolejności.

Wskazówka:

Najłatwiejszą metodą rozwiązania tego zadania jest skorzystania z wektora (za 2 punkty). Rozmiar wektora może być łatwo powiększany metodą push_back. Zachęcam do zaimplementowania trudniejszego algorytmu, bazującego na tablicy (za 3 punkty). Po przekroczeniu początkowo zadanego rozmiaru tablicy powinna ona dwukrotnie zwiększać swój rozmiar.

Zadanie 9. (1 p)

Zapisz procedurę z zadania 2 przy pomocy szablonu tak, by mogła przyjmować tablicę dowolnego typu.

Zadanie 10. (3 p)

Zaimplementuj strukturę reprezentującą macierz dowolnego wymiaru (niekoniecznie kwadratową). W polach struktura powinna mieć: liczbę kolumn, liczbę wierszy oraz dynamicznie alokowaną tablicę elementów macierzowych (można użyc vector<float>).

Zaimplementuj metody: konstruktor przyjmujący liczbę wierszy i kolumn oraz dane macierzy, konstruktor tworzący macierz stałych wartości o zadanym rozmiarze, konstruktor kopiujący, metodę wyświetlającą macierz, operację transpozycji oraz operację mnożenia macierzy (metoda musi sprawdzać, czy rozmiar mnożonych macierzy się zgadza) oraz mnożenia macierzy przez liczbę. Przykładowy nagłówek struktury może wyglądać tak:

```
class Matrix
{
public:
    Matrix(int rowNr, int colNr, std::vector<float>& value);
    Matrix(int rowNr, int colNr, float constVal = 0.);
    Matrix(const Matrix &right);
    ~Matrix();
    float getElement(int rowNr, int colNr);
    void setElement(int rowNr, int colNr, float value);
    void print();
    scale(float x);
    Matrix multiply(Matrix& B);
    Matrix getTranspose();
    int getNrOfColumns();
```

```
    int getNrOfRows();
private:
    int rowNrs;
    int colNrs;
    std::vector<float> values;
};
```

Sprawdź działanie Twojej struktury na prostych przykładach. Sprawdzenie powinno zawierać wykorzystanie każdego z konstruktorów, wyświetlanie elementów, mnożenie dwóch macierzy, transpozycja i mnożenie macierzy przez liczbę.

Wskazówka:

```
float Matrix::getElement(int rowNr, int colNr)
{
    if(rowNr > this->rowNrs || colNr > this->colNrs)
        throw std::invalid_argument("Matrix::getElement(int rowNr, int colNr): input paramet
    return values.at(rowNr*colNrs + colNr);
}

void Matrix::setElement(int rowNr, int colNr, float value)
{
    if(rowNr > this->rowNrs || colNr > this->colNrs)
        throw std::invalid_argument("Matrix::getElement(int rowNr, int colNr): input paramet
    values.at(rowNr*colNrs + colNr) = value;
}
```

Zadanie 11. (2 p)
Korzystając ze struktury macierzy z zadania 1 zaimplementuj metodę potęgową (*https* : *//en.wikipedia.org/wiki/Power_iteration*) liczenia największej co do wartości bezwzględnej wartości własnej. Zauważ, że iloczyn skalarny wektorów możemy uzyskać mnożąc wektorowo wketor transponowany przez wektor właściwy. W ten sam sposób można wyznaczyć normę tablicy. Jako punkt startowy metody wygenerować wektor o zrandomizowanych elementach (nagłówek <random>). Iterować aż do uzyskania zadanego stopnia zbieżności wartości własnej. Procedura powinna mieć sygnaturę float poweriter(const matrix & A, float epsilon).

Procedurę sprawdz dla macierzy $A = \begin{bmatrix} 5 & -2 & -2 \\ -3 & 5 & 0 \\ 23 & -19 & -6 \end{bmatrix}$