Zadanie 0 – Pochodna w punkcie (1 pkt)

Napisz funkcje

```
double forwardDiff(double(*f)(double), double x, double h);
double backwardDiff(double(*f)(double), double x, double h);
double centralDiff(double(*f)(double), double x, double h);
double richardsonDiff(double(*f)(double), double x, double h);
double centralSecondDiff(double(*f)(double), double x, double h);
```

Obliczające pochodne funkcji f w punkcie x z wykorzystaniem różnic odmiennych rodzajów. Wykorzystaj poniższe wzory:

$$f'_f(x) = \frac{f(x+h) - f(x)}{h} \tag{1}$$

$$f'_b(x) = \frac{f(x) - f(x-h)}{h} \tag{2}$$

$$f'_c(x) = \frac{f(x+h) - f(x-h)}{2h} \tag{3}$$

$$f'_r(x) = \frac{-f(x+2h) + 8f(x+h) - 8f(x-h) + f(x-2h)}{12h} \tag{4}$$

$$f''_c(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} \tag{5}$$

Porównaj dokładność metod liczenia pierwszej pochodnej.
Wykorzystaj poniższy szablon:

```
#include <cmath> //bez spacji przed >
#include <iostream>

double forwardDiff(double (*f)(double), double x, double h);
double backwardDiff(double (*f)(double), double x, double h);
double centralDiff(double (*f)(double), double x, double h);
double richardsonDiff(double (*f)(double), double x, double h);
double centralSecondDiff(double (*f)(double), double x, double h);

double poly(double x)
{
        double sum = 0;
        for (int i = 1; i != 9; ++i)
        {
                sum += (i + 1) * pow(x, i);
        }
        return sum;
}

int main()
{
        auto h = {0.01, 0.00001};
        auto v = {poly, cos};
        for (auto hi : h)
        {
                std::cout << "h=" << hi << std::endl;
```

```
                for (auto f : v)
                {
                        std::cout << forwardDiff(f, 0, hi) << std::endl;
                        std::cout << backwardDiff(f, 0, hi) << std::endl;
                        std::cout << centralDiff(f, 0, hi) << std::endl;
                        std::cout << richardsonDiff(f, 0, hi) << std::endl;
                        std::cout << centralSecondDiff(f, 0, hi) << std::endl;
                        std::cout << std::endl;
                }
        }
        return 0;
}
```

Output:
h=0.01
2.03041
1.9704
2.0004
2
6.001
-0.00499996

0.00499996
0
-1.85037e-15
-0.999992
h=1e-05

2.00003
1.99997
2
2
6
-5e-06

5e-06
0
0
-1

Zadanie 1 – różniczkowanie funkcji (1,67 pkt)

Uzupełnij funkcje

```
std::vector<double>* centralDiff(const std::vector<double>* x,
const std::vector<double>* y);

std::vector<double>* secondCentralDiff(const std::vector<double>* x,
const std::vector<double>* y);
```

Które przyjmują wskaźniki na wektory x oraz y niosące informacje o wartościach funkcji w kolejnych punktach. Funkcje tworzą nowy wektor liczb double na stosie (z użyciem operatora new), wypełniają

go odpowiednio pierwszą lub drugą pochodną (różnicą) centralną i zwracają go poprzez wskaźnik. Użyj poniższych wzorów:

$$f' = \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} \tag{6}$$

$$f'' = \frac{f(x_{i+1}) - 2f(x_i) + f(x_{i-1})}{(x_{i+1} - x_i)(x_i - x_{i-1})} \tag{7}$$

Zastanów się jaki rozmiar powinien mieć zwracany wektor. Możesz założyć, że przekazywane punkty są już posortowane po współrzędnej x.

Szablon programu:

```
#include <iostream> //bez spacji przed >
#include <vector>

std::vector<double>* centralDiff(const std::vector<double>* x,
const std::vector<double>* y);

std::vector<double>* secondCentralDiff(const std::vector<double>* x,
const std::vector<double>* y);

int main()
{
        std::vector<double> x = {1, 1.101, 1.202, 1.303, 1.404,
        1.505, 1.606, 1.707, 1.808, 1.909};

        std::vector<double> y = {1.975, 1.85955, 1.72085, 1.55678,
         1.36518, 1.14394, 0.890923, 0.603991, 0.281013, -0.0801417};

        std::cout << "first:" << std::endl;
        std::vector<double>* v = centralDiff(&x, &y);
        for (auto i : (*v) )
        {
                std::cout << i << " ";
        }
        std::cout << std::endl;
        delete v;

        v = secondCentralDiff(&x, &y);
        std::cout << "second:" << std::endl;
        for (auto i : *v)
        {
                std::cout << i << " ";
        }
        std::cout << std::endl;
        delete v;

        return 0;
}
```

Output
first:

-1.25817 -1.49886 -1.76074 -2.04376 -2.34781 -2.67301 -3.01936 -3.3868
second:
-2.27919 -2.48701 -2.69876 -2.9056 -3.11509 -3.32467 -3.53358 -3.74245

Uwaga: Nie kopiować wektorów x i y. Nie używać gotowych implementacji algorytmów różniczkujących.
Podpowiedź: Alokowanie pamięci na stosie odbywa się poprzez operator *new*. Aby dodać element o nazwie *el* do obiektu $std :: vector$ wskazywanego przez wskaźnik o nazwie v, należy użyć $v- >$push_back(el);.

Zadanie 2 – metoda bisekcji (1pkt)

Napisz szablon funkcji

```
template <class T>
bool bisection(double &x, T f, double from, double to, double eps);
```

znajdujący metodą bisekcji pierwiastek x równania z jedną niewiadomą f w przedziale od from do to. Procedura znajdowania pierwiastka powinna trwać do osiągniecia dokładności eps ($to - from < eps$) Sprawdź, czy wartości przekazane do argumentów są sensowne ($from < to, 0 < eps < to - from$) oraz spełnione są założenia. Jeżeli tak, funkcja powinna zwrócić wartość true i przypisać znaleziony pierwiastek do x. W innym przypadku zwrócona powinna zostać wartość false.
Przykład:

```
#include <cmath>
#include <iomanip>
#include <iostream>

template <class T>
bool bisection(double &x, T f, double from, double to, double eps);

double f1(double x) { return x * x; }
double f2(double x) { return x * x - 2.; }
double f3(double x) { return exp(x) + x - 1; }

int main()
{

        double x;
        for (auto fx : {f1, f2, f3})
        {
                for (auto eps : {0.1, 0.01, 0.001, 0.0001, 0.0000001})
                {
                        if (bisection(x, fx, -1, 6, eps))
                        {
                                std::cout << std::setprecision(8)
                                << "eps = " << eps
                                << "\t root = " << x << std::endl;
                        }
                        else
                        {
                                std::cout << "Unable to find root" << std::endl;
                                break;
                        }
```

```
                }
                std::cout << std::endl;
        }
        return 0;
}
```

Przykładowy output: Unable to find root

eps = 0.1 root = 1.4335938
eps = 0.01 root = 1.4165039
eps = 0.001 root = 1.4143677
eps = 0.0001 root = 1.4142342
eps = 1e-07 root = 1.4142136

eps = 0.1 root = 0.01171875
eps = 0.01 root = 0.0014648438
eps = 0.001 root = 0.00018310547
eps = 0.0001 root = -3.8146973e-06
eps = 1e-07 root = 1.8626451e-08

Uwaga: Nie tworzyć nowych tablic czy kolekcji wewnątrz funkcji bisection. Nie korzystać z gotowych implementacji algorytmu. Wartości w outpucie niekoniecznie muszą być identyczne z przykładem, ale powinny zbiegać do prawdziwych rozwiązań.