Zadanie 1

Uzupełnij funkcję

```
double GauLeg357(std::function<double(double)> f,
                 double a, double b, int points) {
    // knots for 3-point quadrature
    static const double k3[] = {
                               -0.774596669241483,
                                0,
                                0.774596669241483
                               };
    // weights for 3-point quadrature
    static const double w3[] = {
                                0.555555555555556,
                                0.888888888888889,
                                0.555555555555556
                               };

    // knots for 5-point quadrature
    static const double k5[] = {
                               -0.906179845938664,
                               -0.538469310105683,
                                0,
                                0.538469310105683,
                                0.906179845938664
                               };
    // weights for 5-point quadrature
    static const double w5[] = {
                                0.236926885056189,
                                0.478628670499366,
                                0.568888888888889,
                                0.478628670499366,
                                0.236926885056189
                               };


    // knots for 7-point quadrature
    static const double k7[] = {
                               -0.949107912342759,
                               -0.741531185599394,
                               -0.405845151377397,
                                0,
```

```
                                0.405845151377397,
                                0.741531185599394,
                                0.949107912342759
                               };
    // weights for 7-point quadrature
    static const double w7[] = {
                                0.129484966168870,
                                0.279705391489277,
                                0.381830050505119,
                                0.417959186373469,
                                0.381830050505119,
                                0.279705391489277,
                                0.129484966168870
                               };
    // ...
}
```

która pobiera funkcję f (czyli wskaźnik funkcyjny, lambdę lub obiekt wywoływalny), granice całkowania a i b i wartość points, która może być 3, 5 lub 7 i określa ilość punktów w kwadraturze Gaussa-Legendre'a (powinna być to zatem kwadratura dokładna dla wielomianów stopnia, odpowiednio, 5, 9 i 13). Funkcja zwraca obliczoną wartość całki. Pamiętaj, że podane wartości węzłów i wag odpowiadają przedziałowi całkowania $[-1, 1]$; dla innych przedziałów konieczna jest zamiana zmiennych.

Następnie zdefiniuj *rekurencyjną* funkcję

```
/*
 *   Recursive function integrating f in interval [a,b].
 *   Adaptive algorithm using points-point Gauss-Legendre
 *   quadrature (GaulLeg357) with points = 3, 5 or 7.
 *   Argument 'eps' is the relative precision required.
 *   Argument 'whole', if not zero, passes the integral
 *   on the whole [a,b] interval calulated before, so only
 *   integrals over two halves are recursively calculated.
 */
double GauLegAdaptive(std::function<double(double)> f,
                double a, double b, double eps,
                int points, double whole=0) {
    // ...
}
```

która realizuje adaptacyjną metodę liczenia całki za pomocą dzielenia przedziału całkowania na połowy. Ostatni argument (z wartością domyślną 0) przeznaczony jest do tego, aby w wywołaniu rekurencyjnym można było przekazać już policzoną w poprzedniej iteracji całkę na całym przedziale.

Na przykład następująca funkcja **main**

```
int main() {
    using std::cout; using std::endl;
    double res, eps = 1e-12;

      // 6x^5+5x^4+4x^3+3x^2+2x+1 on [0,1] should be 6
    res = GauLegAdaptive(
              [] (double x) -> double {
                  return ((((6*x+5)*x+4)*x+3)*x+2)*x+1;
              }, 0,1,eps,3);
    cout << "Int[0,1] 6x^5+5x^4+4x^3+3x^2+2x+1"
         << " -> " << res << endl;

      // x*e^x on [0,1] should be 1
    res = GauLegAdaptive(
              [] (double x) -> double {return x*exp(x);},
              0,1,eps,5);
    cout << "Int[0,1] x*e^x"
         << " -> " << res << endl;

      // 5*exp(2*x)*cos(x)/(exp(PI)-2)
      // integrated on [0,PI/2] should be 1
    res = GauLegAdaptive(
              [] (double x) -> double {
                  return 5*exp(2*x)*cos(x)/(exp(M_PI)-2);
              }, 0,M_PI/2,eps,7);
    cout << "Int[0,pi/2] 5*exp(2*x)*cos(x)/(exp(PI)-2)"
         << " -> " << res << endl;
}
```

powinna wydrukować

```
Int[0,1] 6x^5+5x^4+4x^3+3x^2+2x+1 -> 6
Int[0,1] x*e^x -> 1
Int[0,pi/2] 5*exp(2*x)*cos(x)/(exp(PI)-2) -> 1
```

Zadanie 2

Napisz funkcje

```
double integTrap(std::function<double(double)> f,
                 double a, double b, size_t n) {
    // ...
}

double integSimp(std::function<double(double)> f,
                 double a, double b, size_t n) {
    // ...
}
```

które obliczają całkę funkcji f w granicach $[a, b]$ dzieląc obszar całkowania na n równych części i wykonującą całkowanie złożoną metodą trapezów i złożoną metodą Simpsona (zwaną też metodą Keplera). W metodzie Simpsona n musi być parzyste! Wykonując numerycznie całkę o znanym wyniku sprawdź, posiłkując się wykresem otrzymanym za pomocą programu ***gnuplot***, że popełniany błąd metody jest proporcjonalny do $1/n^2$ (metoda trapezów) i $1/n^4$ (metoda Simpsona).

Zadanie 3

Napisz i przetestuj funkcje znajdujące pierwiastki równań nieliniowych metodami

- bisekcji,
- Newtona,
- z funkcją iterującą (na przykład dla równania $\ln x = \frac{1}{x}$).

Przedstaw na wykresie zależność błędu od liczby iteracji.

Zadanie 4

Napisz i przetestuj szablony funkcji sortujących tablice metodami

- przez wybór (*selection sort*);
- przez wstawianie (*insertion sort*);
- kopcową (*heap sort*).

Na przykład dla sortowania przez wstawianie kod testujący mógłby wyglądać tak (bez mierzenia czasu):

```
#include <iostream>
#include <algorithm> // sort
#include <utility>   // swap

template <typename T, typename Comp = std::less<T>>
void sel_sort(T* lower, T* upper, Comp cmp=std::less<T>()) {
      // sorting elements from *lower inclusive
      // to *upper  e x c l u s i v e
    // ...
}

template <typename T>
void writeTable(const T* lower, const T* upper,
                std::ostream& str = std::cout) {
    for ( ; lower != upper; ++lower) str << *lower << " ";
    str << std::endl;
}

int main () {
    int arr[] = {4,2,1,9,5};
    size_t size = sizeof(arr)/sizeof(*arr);
```

```
    std::cout << "Before\n";
    writeTable(arr, arr+size);

    sel_sort(arr, arr+size);
    std::cout << "After\n";
    writeTable(arr, arr+size);

    sel_sort(arr, arr+size,
            [](int a, int b) {return b < a;});
    std::cout << "and reversed...\n";
    writeTable(arr, arr+size);

    // sort from the standard library
    std::sort(arr,arr+size);
    std::cout << "and the library sort\n";
    writeTable(arr, arr+size);
    std::sort(arr,arr+size,
            [](int a, int b) {return b < a;});
    std::cout << "and the library sort reversed\n";
    writeTable(arr, arr+size);
}
```

Zauważ, że funkcja sortująca (podobnie jak drukująca) pobierają wskaźniki do pierwszego i „pierwszego za ostatnim" elementu tablicy, a nie tablicę i jej wymiar! Trzecim, opcjonalnym parametrem funkcji sortującej jest komparator, czyli „coś" (wskaźnik do funkcji, obiekt wywoływalny, lambda) co umie porównać dwa elementy odpowiadając **true** wtedy i tylko wtedy, gdy pierwszy jest mniejszy (ostro) od drugiego.

Zbadaj czas wykonania różnych metod sortowania w funkcji rozmiaru tablicy dla danych przypadkowych oraz dla tablic już posortowanych. Sporządź odpowiednie wykresy. Dla porównania użyj też bibliotecznej funkcji **std::sort** (z nagłówka *algorithm*).

Pamiętaj, że w czasie sortowania elementy tablicy mogą być porównywane tylko za pomocą podanego (lub domyślnego) komparatora cmp. Do zamiany elementów możesz użyć bibliotecznej funkcji **std::swap**.

Zadanie 5

Zmienne losowe A, B, C mają jednorodny rozkład prawdopodobieństwa na odcinku $[0, 1)$ i są niezależne. Oszacuj, za pomocą dużej liczby losowych prób, prawdopodobieństwo tego, że równanie $Ax^2 + Bx + C = 0$ ma rozwiązanie rzeczywiste.
Porównaj wynik z wartością $(5 + 3\ln 4)/36 \approx 0.25441342$.

Zadanie 6

Piotr i Paweł rzucają monetą N razy. Po każdym rzucie Paweł daje złotówkę Piotrowi, jeśli wypadła reszka, a dostaje złotówkę od Piotra, jeśli wypadł orzeł. Oszacuj,

symulując wiele takich gier, średnią wygraną Piotra po N rzutach i jej wariancję. Powtórz dla innych wartości N. Wykreśl zależność wariancji jako funkcji N.

Zadanie 7

Przy pomocy generatora liczb losowych oceń prawdopodobieństwa

- otrzymania przynajmniej jednej szóstki w czterech rzutach kostki;
- otrzymania przynajmniej jednej podwójnej szóstki w 24 rzutach dwoma kostkami.

Które z nich jest większe? Jest to problem kawalera de Méré, którego rozwiązanie podał Blaise Pascal.

Zadanie 8

Rozwiąż metodami Eulera, Heuna i punktu środkowego równanie oscylatora harmonicznego $\ddot{x}(t) + \omega^2 x(t) = 0$ (można położyć $\omega = 2\pi$; okres drgań będzie wtedy 1). Wykreśl rozwiązanie dla t od 0 do 10 w postaci wykresu na płaszczyźnie fazowej (położenie-prędkość).