Julia Plewa
MOwNiT
Laboratorium 6

1. **Proszę zaimplementować metodę prostokątów, trapezów oraz Simpsona obliczania całki numerycznej. Proszę zilustrować na wykresie jakość wyników (dla rozwiązań kilku dowolnych całek - wynik najlepiej sprawdzić używając języka mathematica). Dokonać stosownej analizy wyników.**

Metoda prostokątów:

```
class RectangleMethodIntegration : public IIntegration
{
    public:
    RectangleMethodIntegration(std::function<double(double)> function, int N) :
        IIntegration(function, N) {}
    public:
    double Integrate(double start, double end) override
    {
        double dx = (end - start) / N;
        double rectangleSum = 0;
        for (int i = 1; i <= N; i++)
        {
            rectangleSum += function(getXk(i, start, end));
        }
        return (dx * rectangleSum);
    }
};
```

Metoda trapezów:

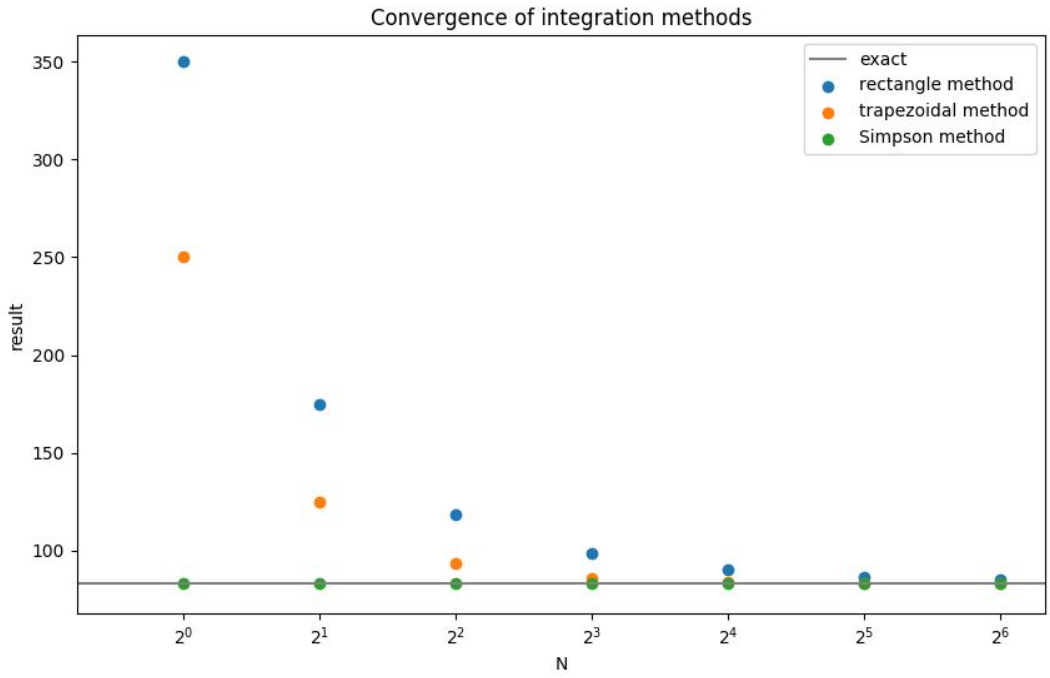
```
class TrapezoidalMethodIntegration : public IIntegration
{
    public:
    TrapezoidalMethodIntegration(std::function<double(double)> function, int N) :
        IIntegration(function, N) {}
    public:
    double Integrate(double start, double end) override
    {
        double dx = (end - start) / N;
        double trapezoidSum = 0;
        std::vector<double> fx;
        for (int i = 0; i <= N; i++)
        {
             fx.push_back(function(getXk(i, start, end)));
        }
        for (int i = 1; i <= N; i++)
        {
            trapezoidSum += (fx[i] + fx[i - 1]);
        }
        return trapezoidSum * dx / 2;
    }
};
```

Metoda Simpsona:

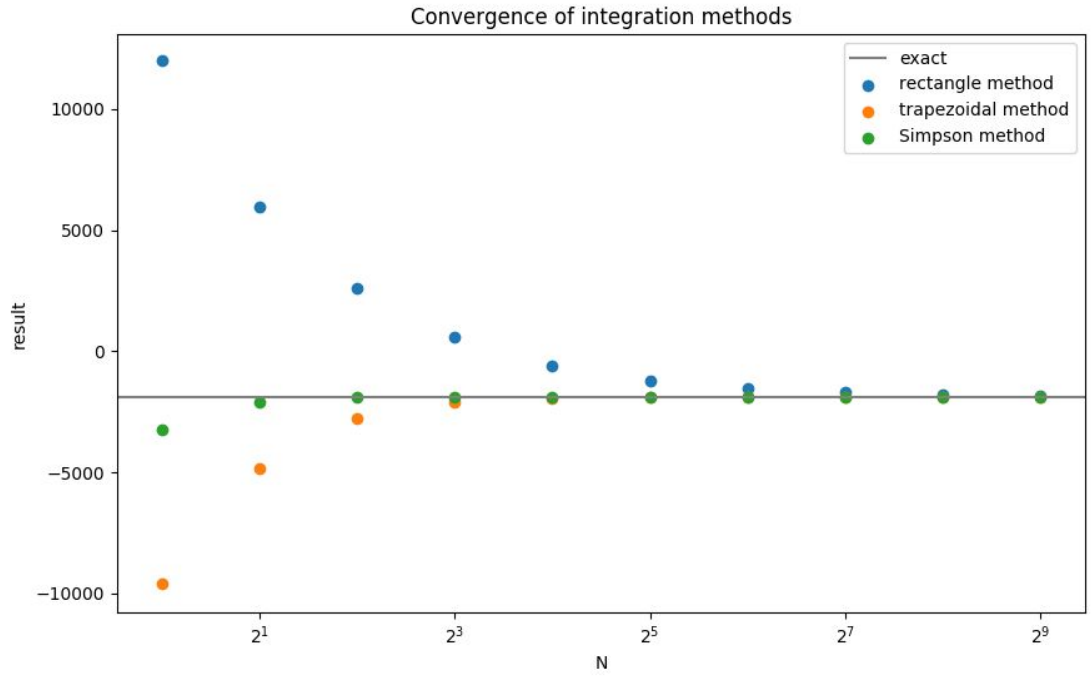
```
class SimpsonMethodIntegration : public IIntegration
{
    public:
    SimpsonMethodIntegration(std::function<double(double)> function, int N) :
        IIntegration(function, N) {}
    public:
    double Integrate(double start, double end) override
    {
        double dx = (end - start) / N;
        double simpsonSum = 0;
        double xLeft;
        double xRight = start;
        for (int i = 1; i <= N; i++)
        {
            xLeft = xRight;
            xRight = getXk(i, start, end);
            simpsonSum += function(xLeft) + function(xRight);
            simpsonSum += 4 * function((xRight + xLeft) / 2.0);
        }
        return simpsonSum * dx / 6;
    }
};
```

Porównanie działania:

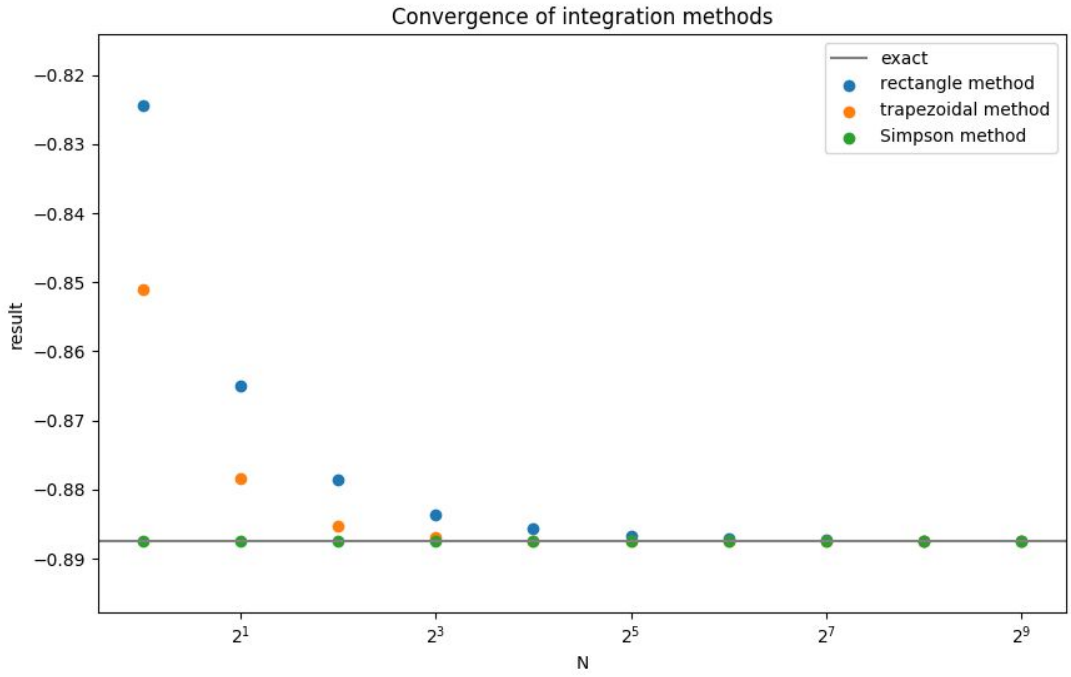
$f_1(x) = x^2 + 2x$



$f_2(x) = x^5 - 13e^x + 5$



$f_3(x) = x^3 - x^2 + xlnx - e^{\frac{x}{2}}$



Wnioski:

Metody prostokątów, trapezów oraz Simpsona opierają się o zastosowanie kwadratur Newtona-Cotesa na poszczególnych podprzedziałach zadanej funkcji. Dokładność wszystkich trzech przetestowanych metod rośnie wraz ze zwiększeniem liczby wykorzystanych podprzedziałów.

W przypadku zaimplementowanej wersji metody prostokątów pod uwagę bierzemy jedynie skrajny lewy punkt każdego podprzedziału. W metodzie trapezów pod uwagę bierzemy wartości na lewym i prawym krańcu podprzedziału. Metoda Simpsona wykorzystuje wartości na krańcach oraz wartość w centralnym punkcie. Łatwo zaobserwować, że będzie to miało spore znaczenie w przypadku funkcji o dużych wahaniach wartości oraz w przypadku podziału przedziału całkowania na niewystarczająco wiele fragmentów, zatem metoda Simpsona wypada najlepiej ze wszystkich trzech, a metoda prostokątów najgorzej.

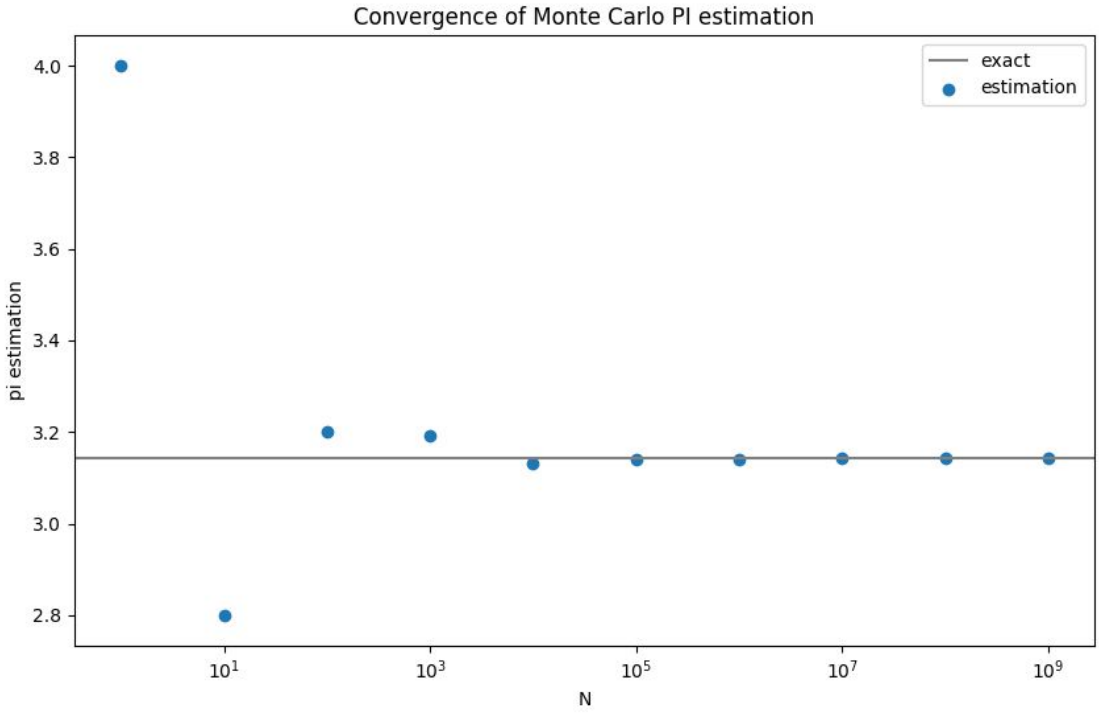
2. **Przeanalizować tutorial dotyczący metod Monte Carlo - zwłaszcza rozdział "Methods" oraz "Integration", polecam również "Practical example".**

3. **Proszę wykorzystać metodę Monte Carlo do obliczenia wartości liczby PI. Dokonać stosownej analizy wyników.**

Kod:

```
double estimatePi(int N)
{
    std::default_random_engine gen;
    std::uniform_real_distribution distr;
    int hits = 0;
    for (int i = 0; i < N; i++) {
        float x = distr(gen);
        float y = distr(gen);
        float l = sqrt(x * x + y * y);
        if (l <= 1)
        {
            hits++;
        }
    }
    return ((double) 4 * hits) / (double) N;
}
```

Wykres wyników:



Opis:

Metoda Monte Carlo umożliwia modelowanie matematyczne złożonych procesów - opiera się na losowym wyborze wielkości charakteryzujących proces. W przypadku estymowania wartości liczby π w kolejnych iteracjach losowane są zgodnie z rozkładem jednostajnym liczby leżące na kwadracie o wymiarach $2x2$ (w powyższej implementacji wykorzystywana jest tak naprawdę ćwiartka takiego kwadratu) - stosunek liczby punktów leżących wewnątrz koła o promieniu 1 znajdującego się wewnątrz kwadratu do całkowitej liczby punktów stanowi oszacowanie pola tego koła. Ponieważ $r = 1$, a pole koła opisywane jest wzorem $A = \pi r^2$, uzyskany wynik jest równocześnie oszacowaniem wartości π.

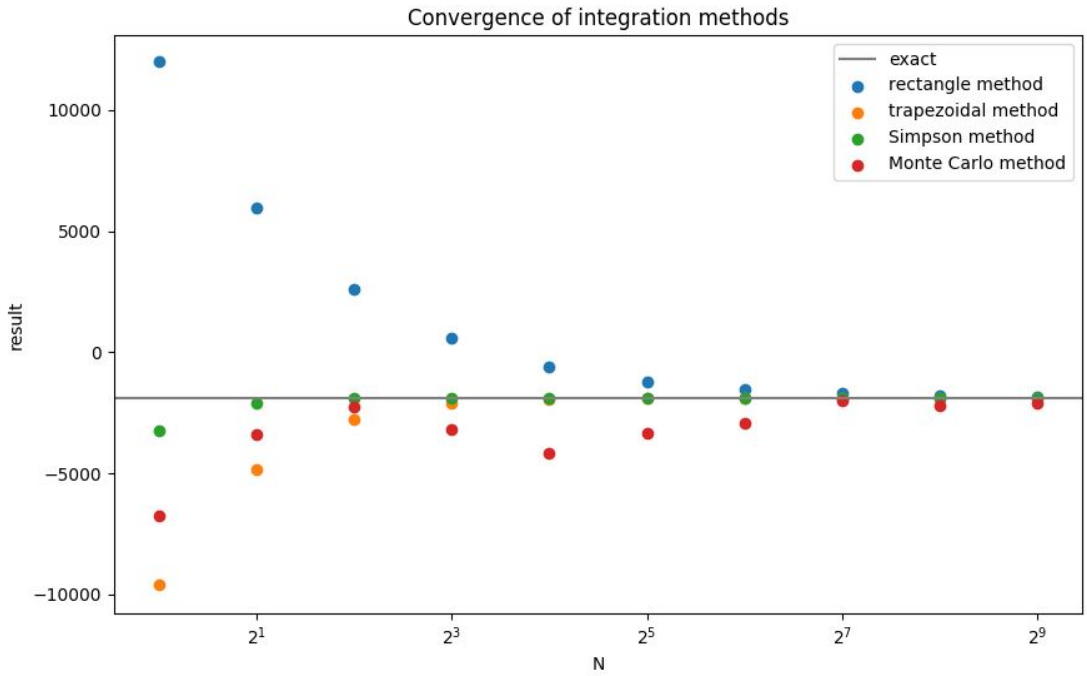
4. **Proszę zaimplementować metodę Monte Carlo (na bazie tutoriala z Zadania 4) obliczania całki numerycznej. Porównać empirycznie tą metodę z pozostałymi. Dokonać stosownej analizy wyników.**

Metoda Monte Carlo:

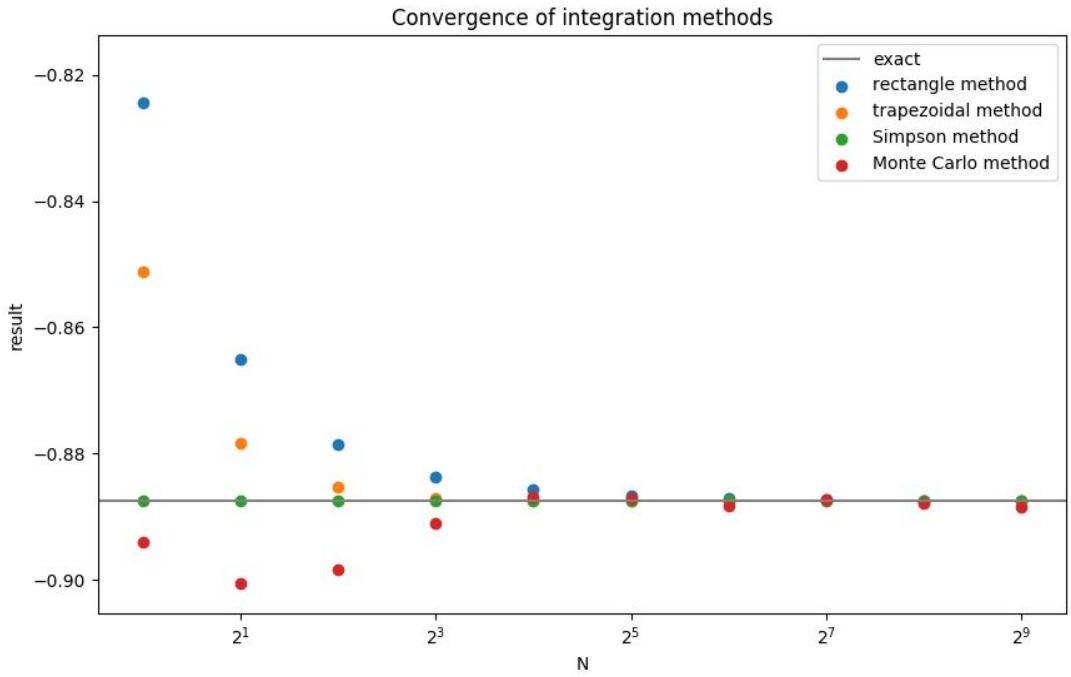
```
class MonteCarloIntegration : public IIntegration
{
    public:
    MonteCarloIntegration(std::function<double(double)> function, int N) :
        IIntegration(function, N) {}
    public:
    double Integrate(double start, double end) override
    {
        std::default_random_engine gen;
        std::uniform_real_distribution distr;
        double dx = end - start;
        double monteCarloSum = 0;
        for (int i = 0; i < N; i++)
        {
            monteCarloSum += function(start + distr(gen) * dx);
        }
        return monteCarloSum * dx / N;
    }
};
```

Działanie:

$f_2(x) = x^5 - 13e^x + 5$



$f_3(x) = x^3 - x^2 + xlnx - e^{\frac{x}{2}}$



Opis:

Całkowanie oparte o metodę Monte Carlo jest proste w implementacji i łatwo skalowalne na modele wielowymiarowe, a ponadto możliwe jest dokonywanie obliczeń w sposób równoległy. Złożoność obliczeniowa tej metody jest dużo lepsza niż wykładnicza złożoność metod deterministycznych w przypadku rozszerzenia modelu na wiele wymiarów. Błąd metody Monte Carlo jest jednak trudny do oszacowania, a dokładność wyniku zależy w dużym stopniu nie tylko od liczby sprawdzeń, ale też od jakości użytego generatora liczb pseudolosowych.

W obliczonych przykładach prędkość zbieżności metody Monte Carlo wypada gorzej niż w przypadku metody Simpsona, ale lepiej lub porównywalnie do metody prostokątów i trapezów.