Maciej Kozub

Laboratorium 7 – kwadratury oraz metody Monte Carlo

1. Kwadratury

Metody kwadratur polegają na obliczaniu numerycznym (przybliżaniu) całek - a więc liczeniu pól pod wykresem funkcji na zadanym przedziale.

a) Metoda prostokątów - polega na przybliżeniu pola pod wykresem za pomocą sumy pól prostokątów dla pewnej liczby przedziałów.

Kod metody prostokątów:

```
double integrals::rectangleMethod(double start, double end, int density, const std::function <double(double)>& function){
    double result = 0;
    double interval = (end - start) / density;
    for(int i = 1; i <= density; i++){
        result += (interval * function(start + interval * i));
    }
    return result;
}
```

b) Metoda trapezów - polega na przybliżeniu pola pod wykresem za pomocą sumy pól trapezów dla ustalonej liczby przedziałów. Dzięki wykorzystaniu figury trapezu zamiast prostokąta uzyskujemy lepsze przybliżenie.

Kod metody trapezów:

```
double integrals::trapeziumMethod(double start, double end, int density, const std::function <double(double)>& function){
    double result = 0;
    double interval = (end - start) / density;
    for(int i = 0; i < density; i++){
        double fx0 = function(start + interval * i);
        double fx1 = function(start + interval * (i + 1));
        result += ((fx0 + fx1) * interval / 2);
    }
    return result;
}
```

c) **Metoda Simpsona** - polega na wykorzystaniu parabol do przybliżania krzywej funkcji i obliczania całki z funkcji kwadratowej (która jest trywialna). Metoda ta jest najdokładniejszą z kwadratur.

Kod metody Simpsona:

```
double integrals::simpsonMethod(double start, double end, int density, const std::function <double(double)>& function){
    double result = 0;
    double interval = (end - start) / density;
    for(int i = 0; i < density; i++){
        double fx0 = function(start + interval * i);
        double fx1 = function(start + interval * (i + 0.5));
        double fx2 = function(start + interval * (i + 1));
        result += ((interval / 6) * (fx0 + 4 * fx1 + fx2));
    }
    return result;
}
```

2. Obliczanie wartości PI metodą Monte Carlo

Metoda Monte Carlo polega na „próbkowaniu” poprawnego rozwiązania - testowania losowo wygenerowanych wartości w odniesieniu do warunków tego, co chcemy obliczyć. W poniższym przykładzie zostanie przybliżona w ten sposób liczba Pi.

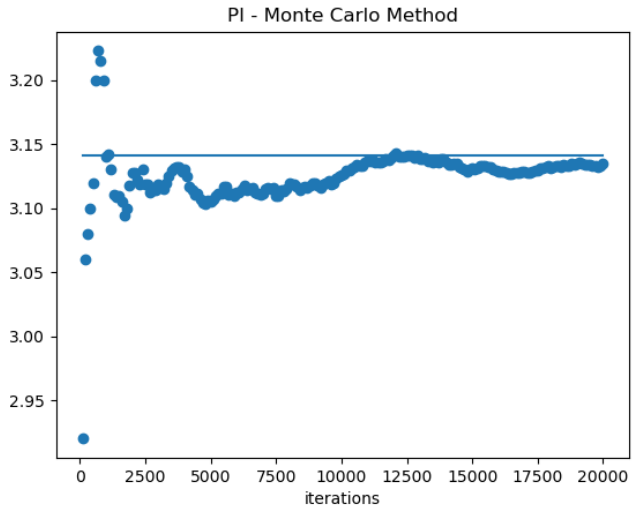
Losowo wygenerowane punkty w kwadracie jednostkowym sprawdzamy pod kątem bycia wewnątrz koła - ponieważ będziemy obliczali jego pole (warunek na odległość od środka). Następnie liczbę trafionych próbek dzielimy przez liczbę wszystkich „strzelonych” punktów - i w ten sposób otrzymujemy stosunek pola tej „ćwiartki” koła do pola kwadratu jednostkowego. Stąd już tylko krok do obliczenia wartości liczby Pi.

W metodzie Monte Carlo oczekujemy, że wraz ze wzrostem liczby próbek otrzymamy coraz dokładniejszy wynik. Nie zawsze się tak dzieje, ponieważ metoda ta opiera się na losowaniu, jednak statystycznie zbieżność do idealnego wyniku jest coraz lepsza przy większej liczbie „strzałów”.

a) Kod metody Monte Carlo obliczającej wartość liczby Pi:

```
double integrals::monteCarloPI(int samplesNumber){
    int innerSamples = 0;
    double radius = 1.0;
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis( a: 0.0, radius);
    for(int i = 0; i < samplesNumber; i++){
        double x = dis( &: gen);
        double y = dis( &: gen);
        if(sqrt( x: x*x + y*y) <= radius){
            innerSamples++;
        }
    }
    return 4.0 * innerSamples / samplesNumber;
}
```

b) Zbieżność wyników w zależności od liczby próbek:



c) Wnioski

Metoda Monte Carlo daje dosyć dokładne wyniki, szczególnie dla dużej liczby próbek. Znajduje to zastosowanie w symulowaniu trudnych obliczeniowo problemów i wszelkich sytuacji, gdzie poznania idealnego wyniku jest niemożliwe, ale jesteśmy w stanie określić warunki tak, by je skutecznie zasymulować.

3. Całkowanie numeryczne – Monte Carlo

Przybliżanie całki za pomocą tej metody sprowadza się do losowego wybrania n punktów z przedziału, w którym obliczamy całkę, i obliczenia średniej wartości funkcji w tych punktach. Następnie przemnażamy tę średnią przez długość przedziału (traktując pole jak prostokąt) - i to jest szukane przez nas przybliżenie.

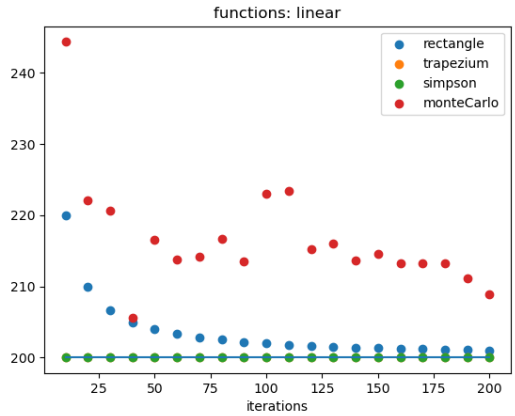
Kod metody przybliżającej numerycznie całkę metodą Monte Carlo:

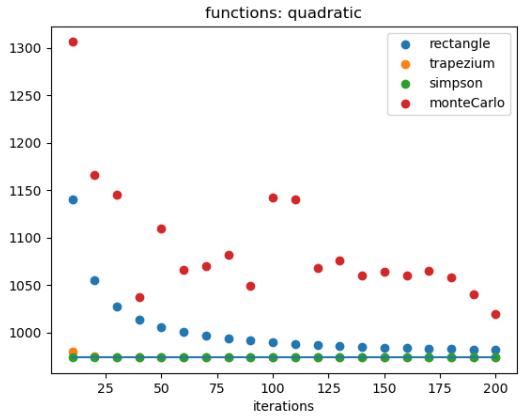
```
double integrals::monteCarloMethod(double start, double end, int density, const std::function <double(double)>& function){
    double totalArea = 0;
    std::random_device rd;
    std::mt19937 gen(rd());
    std::uniform_real_distribution<> dis(start, end);
    for(int i = 0; i < density; i++){
        double x0 = dis( & gen);
        double tmpArea = function(x0) * (end - start);
        totalArea += tmpArea;
    }
    return totalArea / density;
}
```

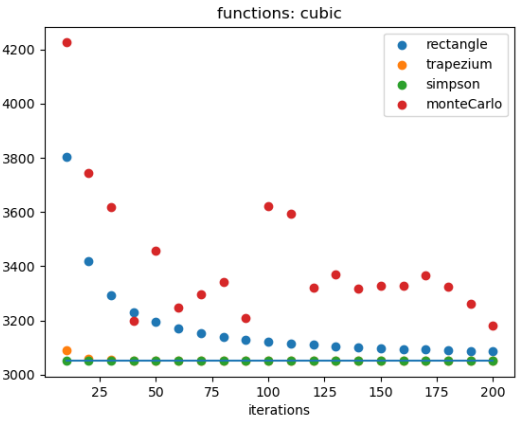
4. Porównanie zbieżności metod kwadratur metodą Monte Carlo

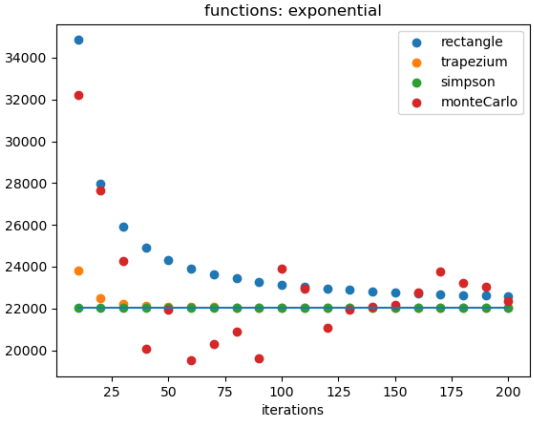
Po zaimplementowaniu wszystkich metod obliczyłem pięć całek dla funkcji: liniowej, kwadratowej, sześciennej, wykładniczej i sinusoidalnej w celu porównania dokładności metod.

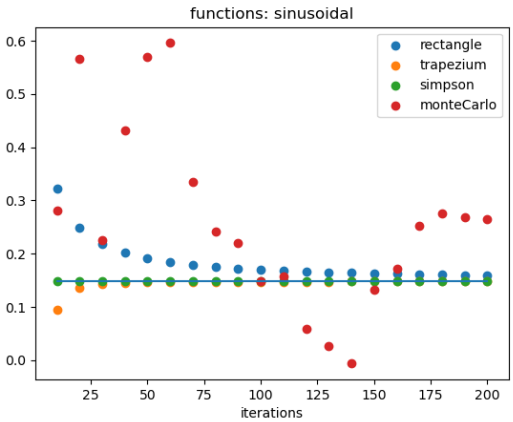
Na poniższych wykresach przedstawiono zbieżność metod do dokładnego wyniku (wynik dla różnej, rosnącej liczby przedziałów - figur, na które podzielono pole pod wykresem - w przypadku kwadratur, dla metody Monte Carlo jest to liczba próbek).











5. Wnioski

Metoda Simpsona zbiega zdecydowanie najszybciej, a metoda prostokątów najwolniej. Monte Carlo dla tak niewielkiej liczby próbek daje wyniki dalece niedokładne - co pokazuje, że dla prostych obliczeń efektywniejsze są kwadratury. Metoda Monte Carlo znajduje swoje zastosowanie tam, gdzie następuję duże skomplikowanie obliczeń lub stają się one niemożliwe do wykonania w skończonym czasie.