MOwNiT2 Lab6

Zadanie 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.

Wszystkie metody zostały zawarte w klasach implementujących interfejs:

```
class IIntegration
{
  public:
    virtual ~IIntegration() = default;
  public:
    virtual void integrate(double from, double to,
                           std::function<double(double)> f) = 0;

};
```

Metoda prostokątów:

```
class RectangleRule : public IIntegration
{
  public:
    RectangleRule(int N) : N(N) {}

    void integrate(double from, double to, std::function<double(double)> f){
      double sum = 0;
      double x, value;
      double dx = (to - from) / N;
      for(int i=1; i<=N; i++){
        x = from + i * dx;
        sum += f(x);
      }
      value = sum * dx;
      std::cout << "Integral via Rectangle Rule =  "<< value << std::endl;
    }

  private:
    int N;
};
```

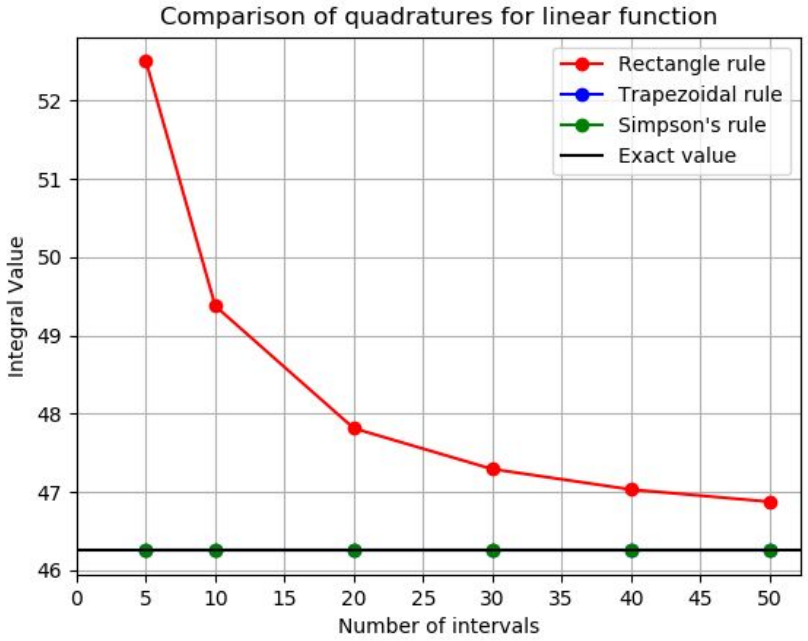
Metoda trapezów:

```
class TrapezoidalRule : public IIntegration
{
  public:
    TrapezoidalRule(int N) : N(N)  {}

    void integrate(double from, double to, std::function<double(double)> f){
     double sum = 0;
     double a, b, value;
     double dx = (to - from) / N;
     for(int i=1; i<=N; i++){
       a = from + (i-1) * dx;
       b = from + i * dx;
       sum += (f(a)+f(b))/2;
     }
     value = sum * dx;
      std::cout << "Integral via Trapezoidal Rule =  "<< value << std::endl;
    }

  private:
    int N;
};
```

Metoda Simpsona:

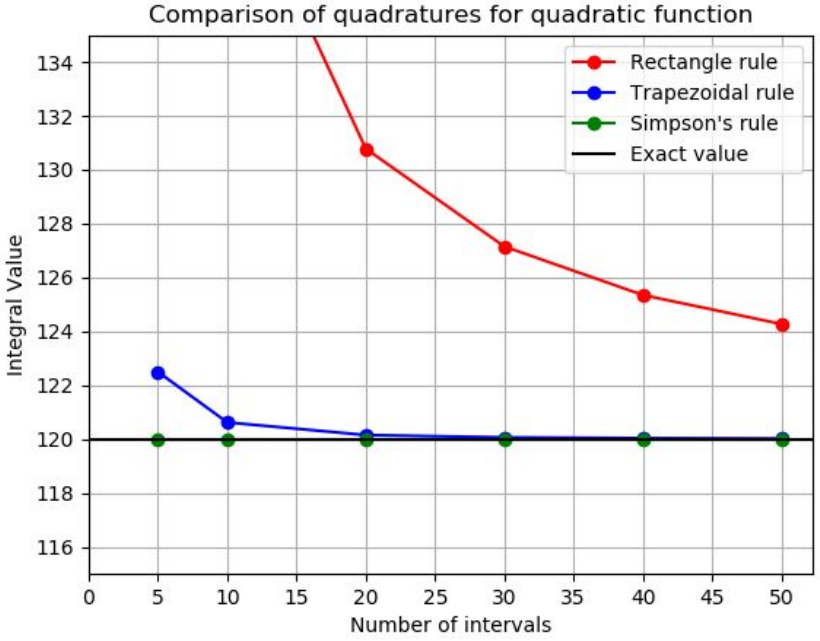
```
class SimpsonsRule : public IIntegration
{
  public:
    SimpsonsRule(int N) : N(N) {}

    void integrate(double from, double to, std::function<double(double)> f){
      double value = 0, middle = 0;
      double x;
      double dx = (to - from) / N;
      for(int i=1; i<=N; i++){
        x = from + i * dx;
        middle += f(x - dx / 2);
        if(i < N) value += f(x);
      }
      value = dx / 6 * (f(from) + f(to) + 2 * value + 4 * middle);
      std::cout << "Integral via Simpson's Rule =  "<< value << std::endl;
    }

  private:
    int N;
};
```

❖ Porównanie metod dla funkcji $f(x) = 2.5x + 3$ na przedziale <0, 5>:



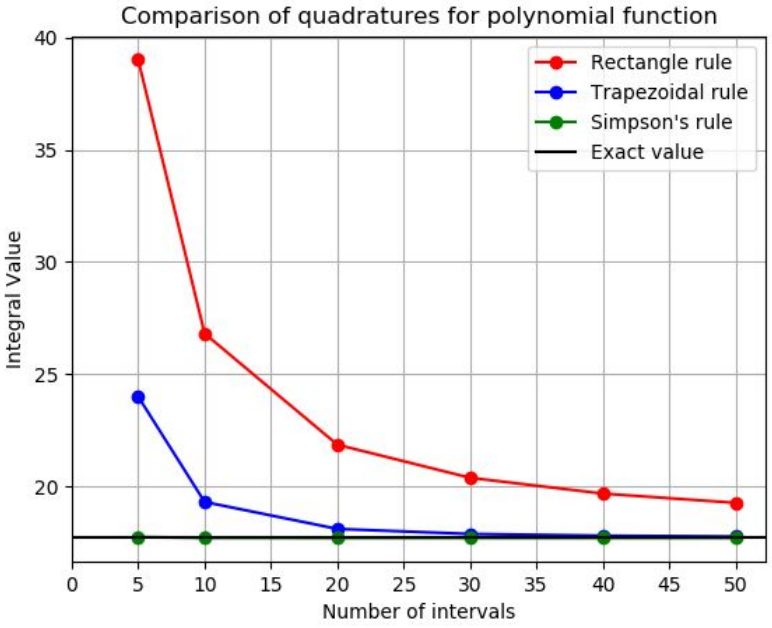
➢ Tylko pierwsza metoda nie zwraca dokładnego wyniku niezależnie od dokładności dla funkcji liniowych.

❖ Porównanie metod dla funkcji $f(x) = 3x^2 + 2x - 6$ na przedziale <0, 5>:

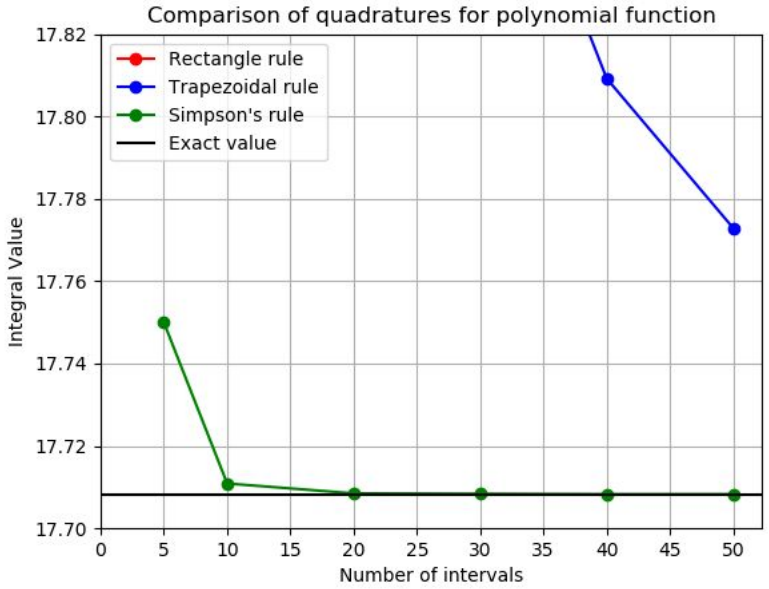


➢ Tylko metoda Simpsona zwraca dokładny wynik niezależnie od ilości przedziałów dla funkcji kwadratowych.

❖ Porównanie metod dla funkcji $f(x) = x^5 + 3.5x^4 - 2.5x^3 - 12.5x^2 + 1.5x + 9$ na przedziale <-3, 2>:



Inna skala:



➢ Dla wielomianów wyższego stopnia wszystkie metody zbiegają do wyniku
➢ W każdym przypadku metoda prostokątów zbliża się do rozwiązania najwolniej, a Simpsona - najszybciej.

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

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

Tworzymy kwadrat o boku $2a$ i wpisujemy w niego koło o promieniu $r = a$. Możemy teraz obliczyć stosunek pól figur: $\frac{P_O}{P_k} = \frac{\pi r^2}{4a^2} = \frac{\pi}{4}$ z czego wynika $\pi = 4 * P_O/P_k$. Idea tej metody polega na losowaniu punktów należących do kwadratu i sprawdzanie czy należą też do koła. Zliczamy ilości tych punktów i podstawiamy je do wzoru za odpowiednie pola.

W szczególnie można uznać $a = 1$ i losować punkty tylko w ćwiartce kwadratu:

```
class MonteCarlo : public IIntegration
{
  public:
    MonteCarlo(int samples) : samples(samples) {}

    void findPi(){
      int interval, i;
      double x, y, d;
      int inside = 0;

      srand(time(NULL));

      for (int i=0; i<samples; i++) {

        x = ((double) rand() / (RAND_MAX));
        y = ((double) rand() / (RAND_MAX));

        // Distance from (0,0) squared
        d = x*x + y*y;

        // if true point is in the circle
        if (d <= 1) inside++;
      }
      double pi = 4 * inside / (double)samples;
      std::cout << "\n[" << samples << " samples] Pi  = " << pi;
    }

    // ... integration ...

  private:
    int samples;
  };
}
```

Wyniki dla różnych ilości punktów (kilka testów ze względu na losowość):

Samples	test1	test2	test3	test4	test5	Avg
10	3.6	3.6	3.2	2.8	4	3.44
100	3.28	3.32	3.04	3.2	3.32	3.232
1000	3.144	2.96	3.096	3.136	3.244	3.116
10000	3.1636	3.124	3.1304	3.1304	3.1636	3.1424
100000	3.13836	3.13924	3.14156	3.1374	3.14736	3.140784

Jak widać wraz ze zwiększeniem ilości punktów wyliczona wartość jest coraz bliższa liczbie π oraz maleje rozrzut pomiędzy wynikami dla różnych testów.

Zadanie 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 jest zbliżona do metody prostokątów, jednak tym razem zamiast sumowania pól prostokątów w każdym przedziale wybieramy losową wartość funkcji z każdego przedziału, uśredniamy ją i dla niej wyliczamy pole prostokąta, gdzie drugą liczbą jest długość przedziału.

Metoda znajduje się w tej samej klasie co w poprzednim zadaniu (MonteCarlo):

```
void integrate(double from, double to, std::function<double(double)> f){

  srand(time(NULL));
  double value  = 0;
  double dx = to - from;
  for(int i=0; i<samples; i++){
    double r = from + ((double) rand()/(double)(RAND_MAX) * dx);
    value += f(r);
  }
  value = dx * value / samples;
  std::cout << "Integral via Monte Carlo method = " << value << std::endl;
}
```

Nie rysuję wykresów dla metody która opiera się na losowaniu...

Funkcja: $f(x) = x^5 + 3.5x^4 - 2.5x^3 - 12.5x^2 + 1.5x + 9$ na przedziale <-3, 2>.

Kilkukrotne wykonanie algorytmu dla tej samej ilości przedziałów co poprzednio:

(Uśrednianie wyników trochę “pomaga” algorytmowi, bo on sam w sobie nie polega na uśrednieniu wyników kilku wywołań)

Intervals	test1	test2	test3	test4	test5	Avg
10	8.71774	9.09681	7.09097	3.80894	5.66139	6.87517
20	12.8316	23.1878	29.1833	25.5171	15.5404	21.25204
30	12.0351	10.9417	13.3299	13.5091	6.86274	11.335708
40	16.7733	12.5428	16.5453	19.0874	11.8383	14.362308
50	18.1272	14.6043	12.8247	13.4168	18.1807	15.43074

Łatwo można zauważyć duży rozrzut pomiędzy wynikami dla tych samych parametrów wejściowych.

Porównanie metod (w Monte Carlo użyto wyliczonych średnich):

Intervals	Oczekiwana wartość	Prostokąty	Trapezy	Simpson	Monte Carlo
10	17.7083	26.8125	19.3125	17.7109	6.87517
20	17.7083	21.8613	18.1113	17.7085	21.25204
30	17.7083	20.3876	17.8876	17.7084	11.335708
40	17.7083	19.6842	17.8092	17.7083	14.362308
50	17.7083	19.2729	17.7729	17.7083	15.43074

Przy ilości przedziałów. dla których metoda Simpsona jest już bardzo bliska wynikowi (tutaj dla 40 przedziałów zwraca już oczekiwaną wartość) metoda Monte Carlo jest jeszcze dość daleko od tej wartości.

Wnioski:
- Metoda Monte Carlo zbiega do rozwiązania dużo wolniej niż np. Simpsona
+ Metoda jest bardzo prosta
+ Niezależna od rodzaju funkcji i konsekwentna
+ Tak jak pozostałe można ją łatwo zrównoleglić