AGH

AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA
W KRAKOWIE

Metody Obliczeniowe w Nauce i Technice:

Kwadratury oraz metody Monte Carlo

Laboratorium 7

Przemysław Lechowicz

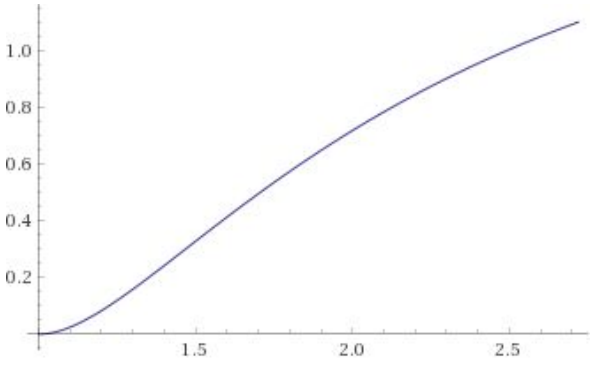
1. Funkcje testowe:

Metody całkowania zdecydowałem się testować na trzech poniższych funkcjach:

a.

$$f_1(x) = \frac{3 * ln^2(x)}{x},$$

w przedziale $< 1, e >$

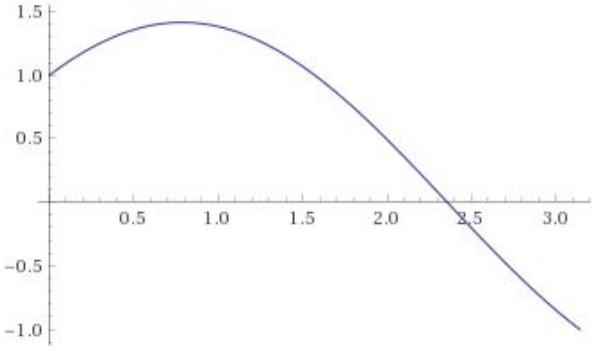


Całka z $f_1(x)$ w tym przedziale wynosi 1.

b.

$$f_2(x) = sin(x) + cos(x),$$

w przedziale $< 0, \pi >$

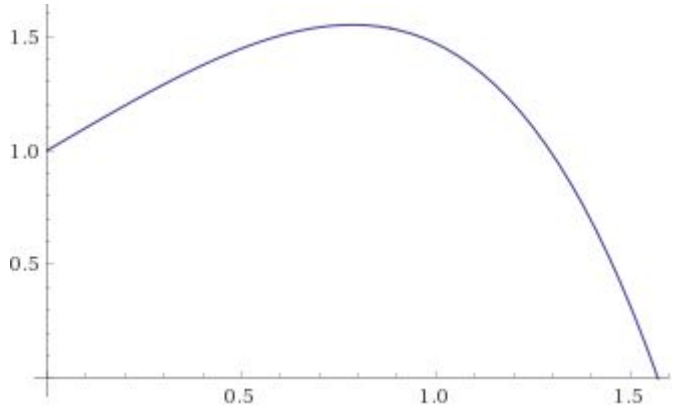


Całka z $f_2(x)$ w tym przedziale wynosi 2.

c.

$$f_3(x) = cos(x) * e^x,$$

w przedziale $< 0, \frac{\pi}{2} >$

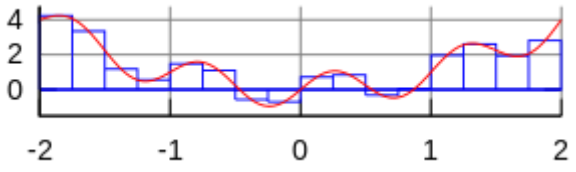


Całka z $f_3(x)$ w tym przedziale wynosi $\frac{1}{2}(e^{\frac{\pi}{2}} - 1) \approx 1.9052$.

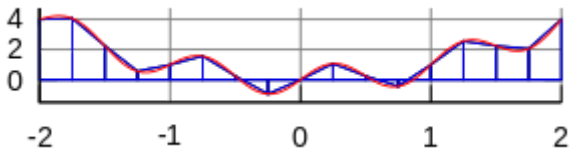
2. Zadanie 1

Proszę zaimplementować metodę prostokątów, trapezów oraz Simpsona obliczania całki numerycznej. Proszę zilustrować na wykresie jakość wyników. Dokonać stosownej analizy wyników.

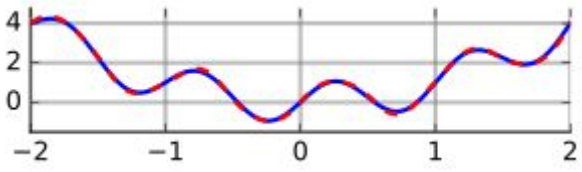
Metoda prostokątów polega na przybliżeniu wartości całki oznaczonej przy pomocy sumy pól odpowiednio dobranych prostokątów - można zauważyć, że stosujemy przybliżenie za pomocą funkcji stałych.



W metodzie trapezów zastępujemy prostokąty trapezami, które lepiej odwzorowują pole pod krzywą - w tym przypadku przybliżamy funkcję za pomocą funkcji liniowych.



W metodzie Simpsona obliczamy sumy wycinków obszarów pod parabolami. Do przybliżenia kształtu funkcji zamiast stosowania funkcji stałych czy liniowych, używamy paraboli.



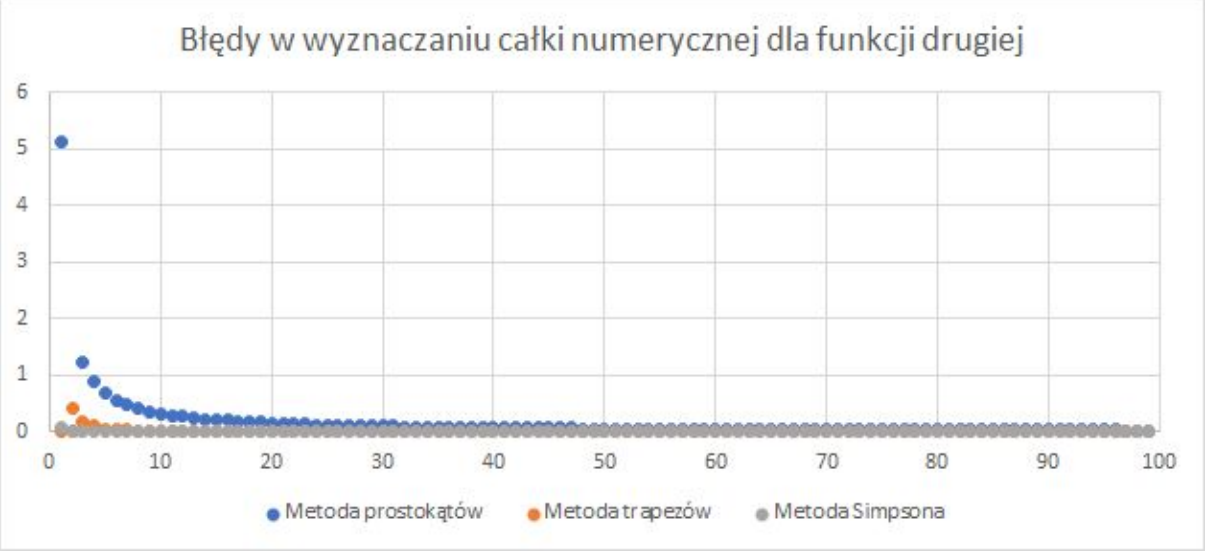
Zaimplementowane zostały wszystkie powyższe metody:

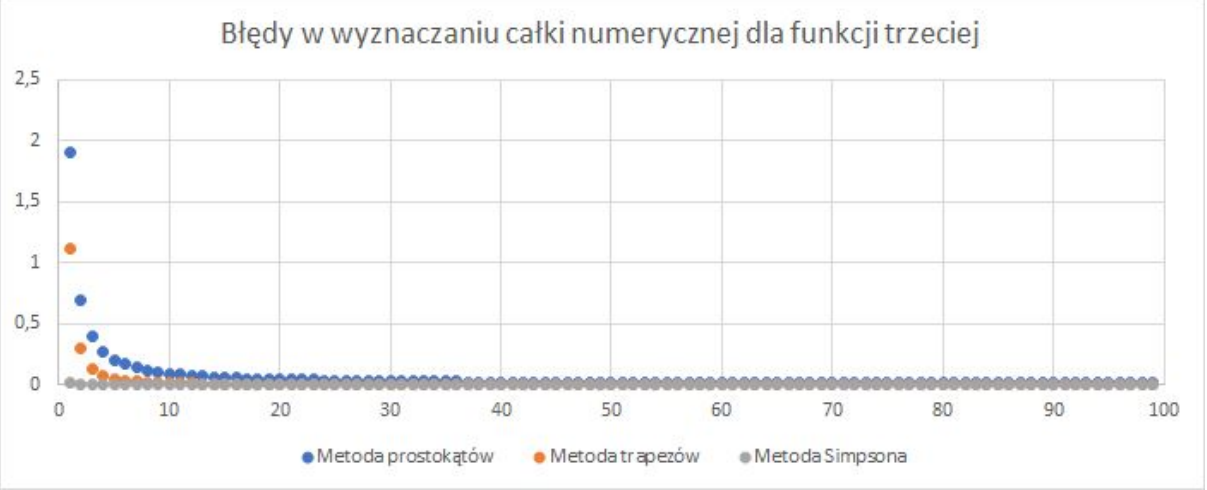
```
double rectangle(double f(double x), double x0, double x1) {
  double result = 0;
  double step = (x1 - x0) / count;
  for (int i = 1; i <= count; i++) {
    result += step * f(x0 + i * step);
  }
  return result
```

```
double trapezoidal(double f(double x), double x0, double x1) {
  double result = 0;
  double step = (x1 - x0) / count;
  for (int i = 1; i <= count; i++) {
    result += ((f(x0 + (i - 1) * step) + f(x0 + i * step)) * step) / 2;
  }
  return result;
}
```

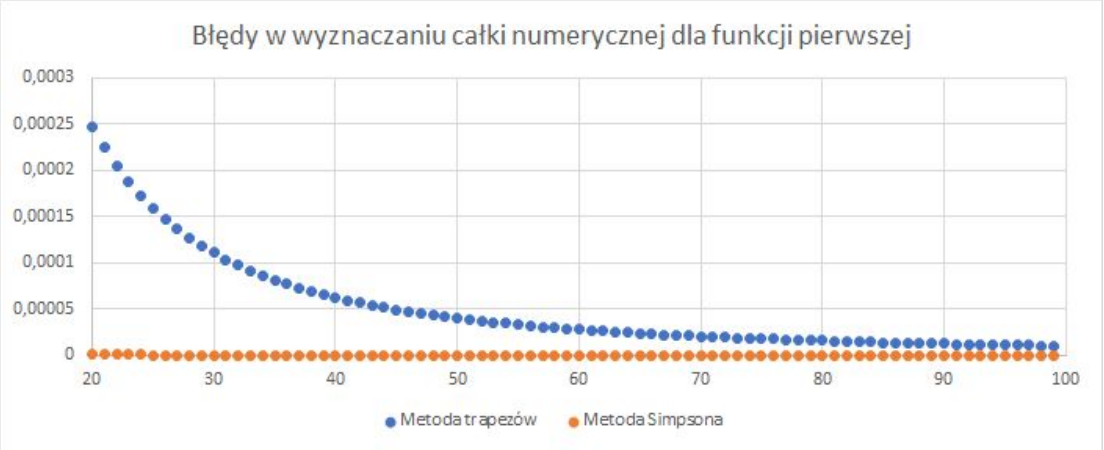
```
double simpson(double f(double x), double x0, double x1) {
  double s = 0;
  double st = 0;
  double h = (x1 - x0) / count;
  double x;
  for (int i = 1; i <= count; i++) {
    st += f(x0 + i * h - h / 2);
    if (i < count) {
      s += f(x0 + i * h);
    }
  }
  s = h / 6 * (f(x0) + f(x1) + 2 * s + 4 * st);
  return s;
}
```

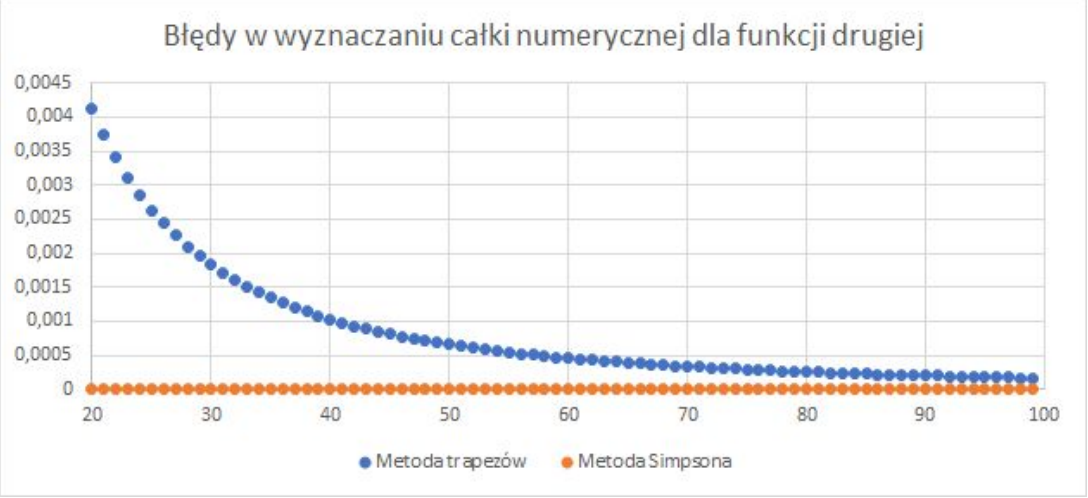
Następnie sprawdziłem jakość przybliżeń dla każdej metody dla różnej liczby równoodległych punktów. Wykres przedstawia wartość bezwzględną różnicy wartości otrzymanej przez daną metodę i wartości poprawnej od 1 do 100 równoodległych punktów.

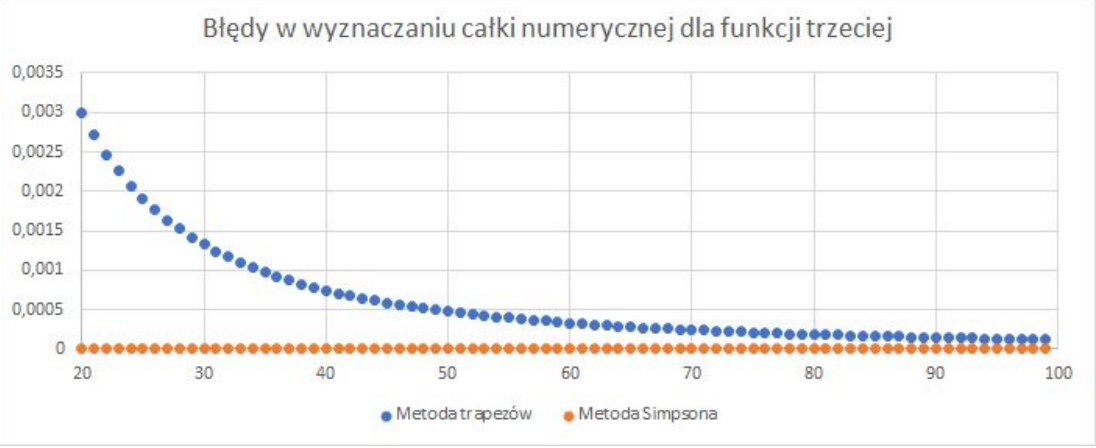




Z powyższych wykresów wynika, że im większa liczba przedziałów całkowania, tym nasze wyniki są bliższe wartości dokładnej, a błędy są mniejsze. Widać także, że najgorszą dokładność w każdym przypadku ma metoda prostokątów. Dla dwóch pozostałych metod rozpatrzmy wykresy, na którym znajdują się tylko błędy tych metod od 20 punktów:







Teraz dokładnie widać, że najlepszą dokładność posiada metoda Simpsona, następnie metoda metoda trapezów, najgorszą dokładność, tak jak zostało napisane wcześniej, ma metoda prostokątów.

3. Zadanie 3

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

Obliczanie liczby π za pomocą metody Monte Carlo opiera się na skorzystaniu ze wzoru:

$$\pi = 4 * \frac{P_{kola}}{P_{kwadratu}}$$

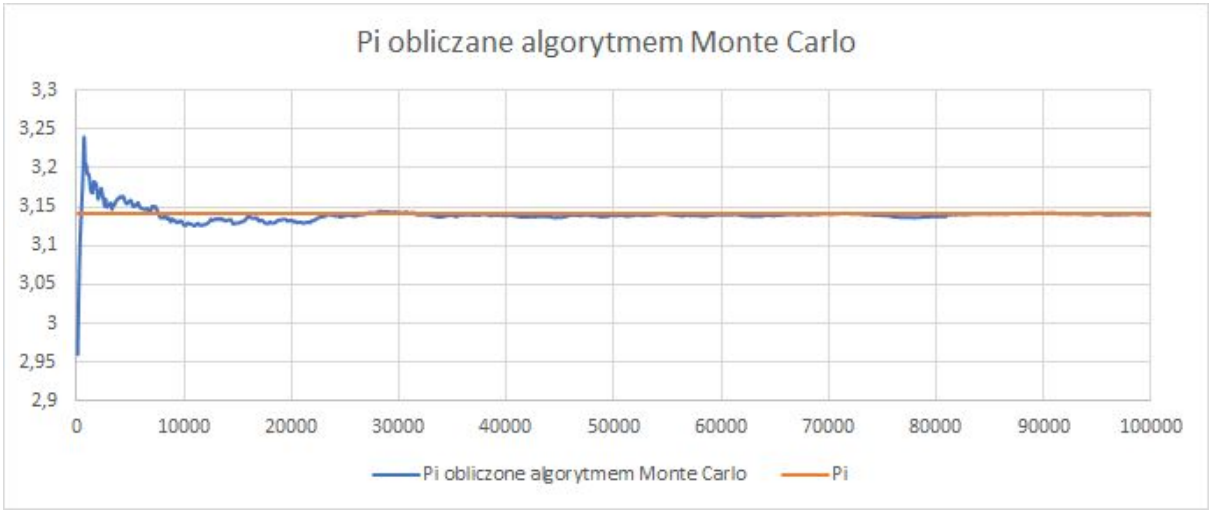
Gdzie koło jest kołem wpisanym w kwadrat.

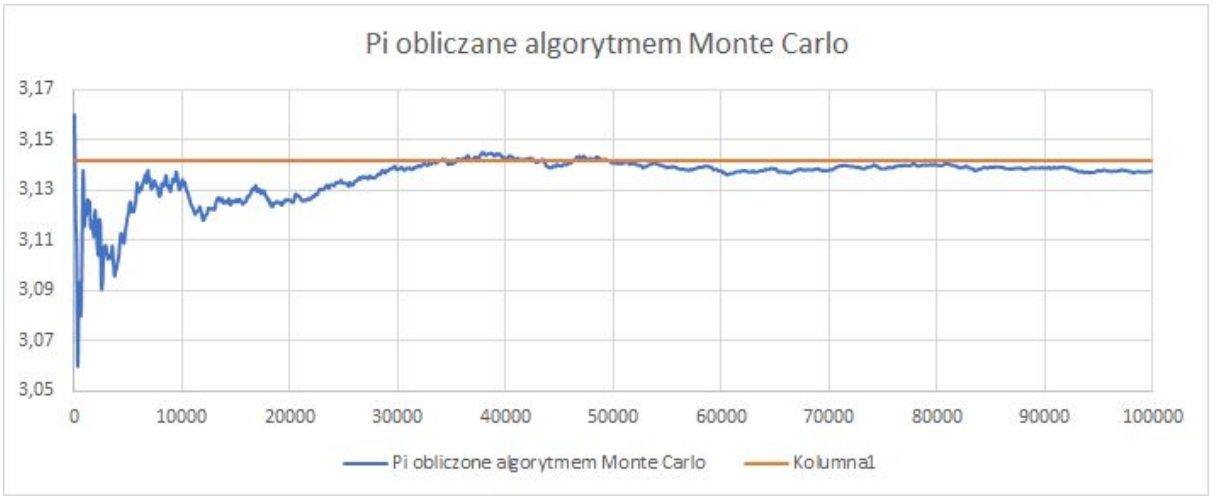
Metoda polega na losowaniu punktów o współrzędnych od -r do r, gdzie r jest promieniem koła, a także połową boku kwadratu. Następnie sprawdzając, czy dany wylosowany punkt należy do koła, zostaje on wliczony do pola koła, lub jeśli nie należy - nie jest wliczany. Tym sposobem korzystając z podanego wyżej wzoru otrzymamy przybliżenie liczby π.

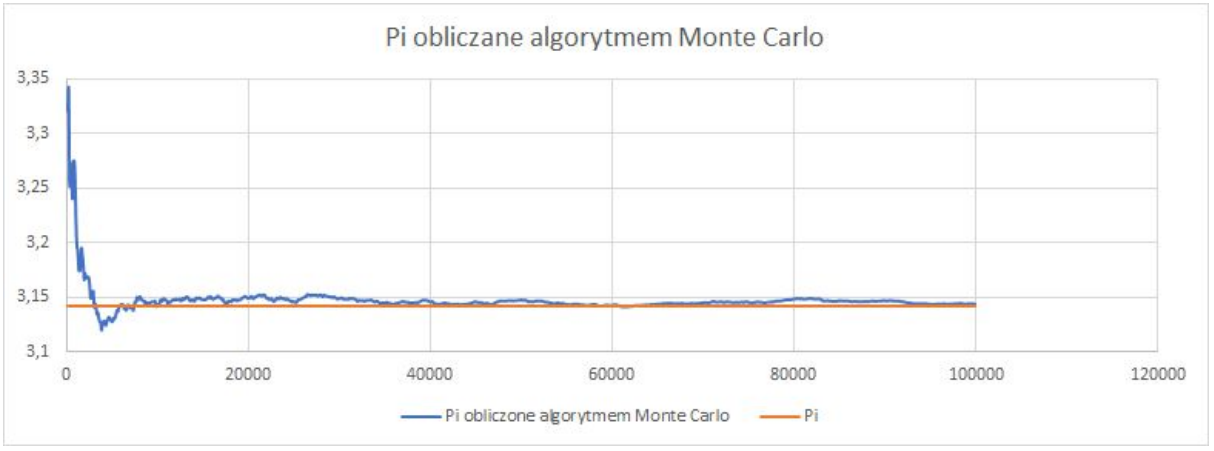
Zaimplementowana została wyżej opisana funkcja:

```
double monteCarlo(long long num) {
  srand(time(NULL));
  long long inside = 0;
  for (int i = 0; i < num; i++) {
    double x = (double)rand() * 2 / (double)RAND_MAX - 1;
    double y = (double)rand() * 2 / (double)RAND_MAX - 1;
    if (x * x + y * y <= 1) {
      inside++;
    }
  }
  return (double)(4.0 * (double)inside) / num;
}
```

Następnie staram się obliczyć liczbę π dla różnej liczby punktów. Zaczynając od 100, za każdym razem zwiększając liczbę punktów o 100, a kończąc na 100000. Próbę powtórzyłem trzykrotnie:







Metoda Monte Carlo jest metodą, która opiera się na losowaniu punktów, dlatego za każdym razem daje inne wyniki. Również dlatego nie zawsze większa ilość punktów jest równa większej dokładności. Jednak w większości przypadków zwiększenie liczby punktów dla paru prób i obliczenie średniej arytmetycznej powinno bardzo dobrze przybliżyć liczbę pi.

Uruchomiłem metodę również dla 10 000 000 000 punktów - również dla trzech prób. Zostały zwrócone liczby:
3,141559696,
3,141591572,
3,141597526.

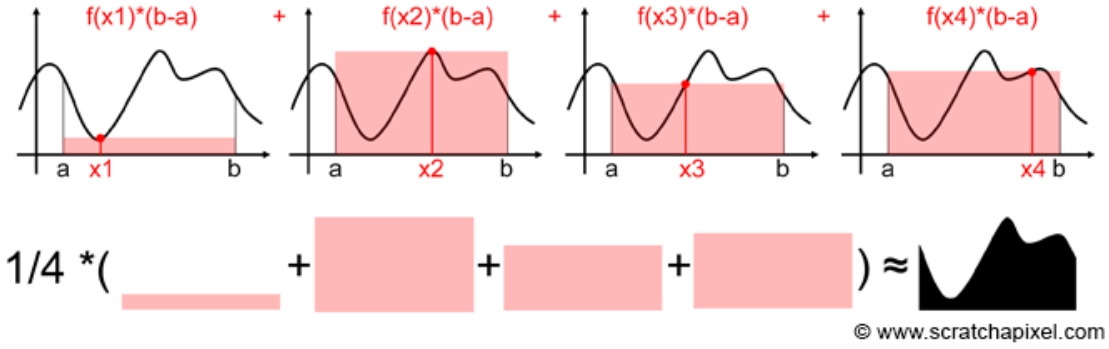
Za każdym razem osiągnięta została dokładność czterech liczb po przecinku. Dla drugiej i trzeciej wartości dokładność wyniosła pięć liczb po przecinku. Po obliczeniu średniej arytmetycznej otrzymaliśmy liczbę 3,141591572 - co daje również dokładność do 5 liczby po przecinku.

Z racji na czas przeprowadzanych obliczeń nie podjąłem próby obliczania dla większej ilości punktów.

4. Zadanie 4

Proszę zaimplementować metodę Monte Carlo obliczania całki numerycznej. Porównać empirycznie tą metodę z pozostałymi. Dokonać stosownej analizy wyników.

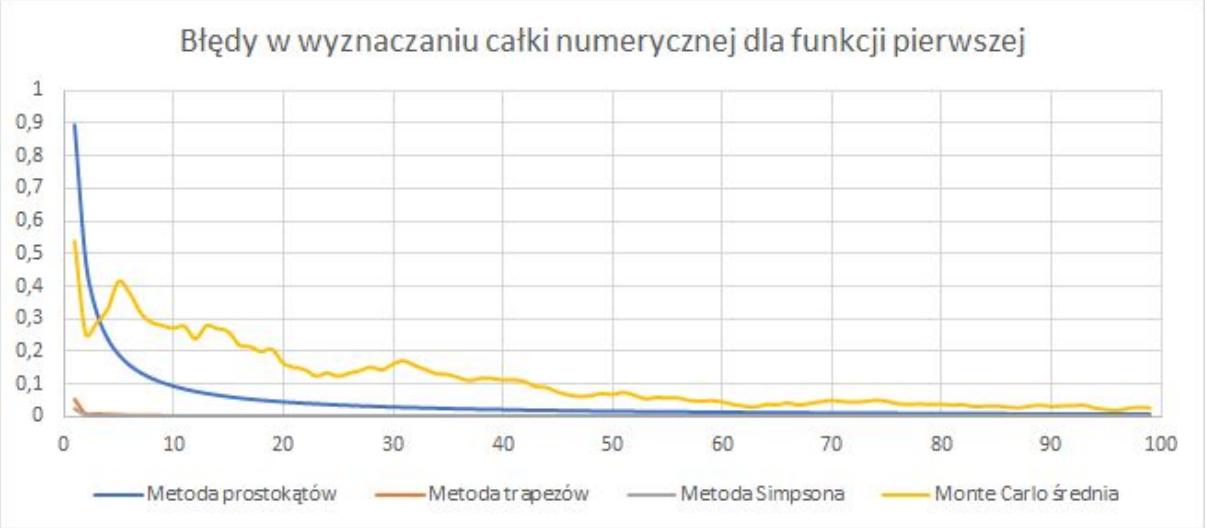
Ponownie opieramy metodę o losowość. Tym razem losujemy jedną liczbę x, należącą do przedziału całkowania. Następnie wyznaczamy wartość funkcji $f(x)$, którą całkujemy, oraz wyznaczamy pole prostokąta o bokach $f(x)$ i $x_0 - x_1$, gdzie x_0 i x_1 są krańcami przedziału, zwiększając wynik o to pole. Powtarzamy dla dowolnej ilości punktów, na sam koniec dzieląc wynik przez liczbę wykorzystanych punktów.

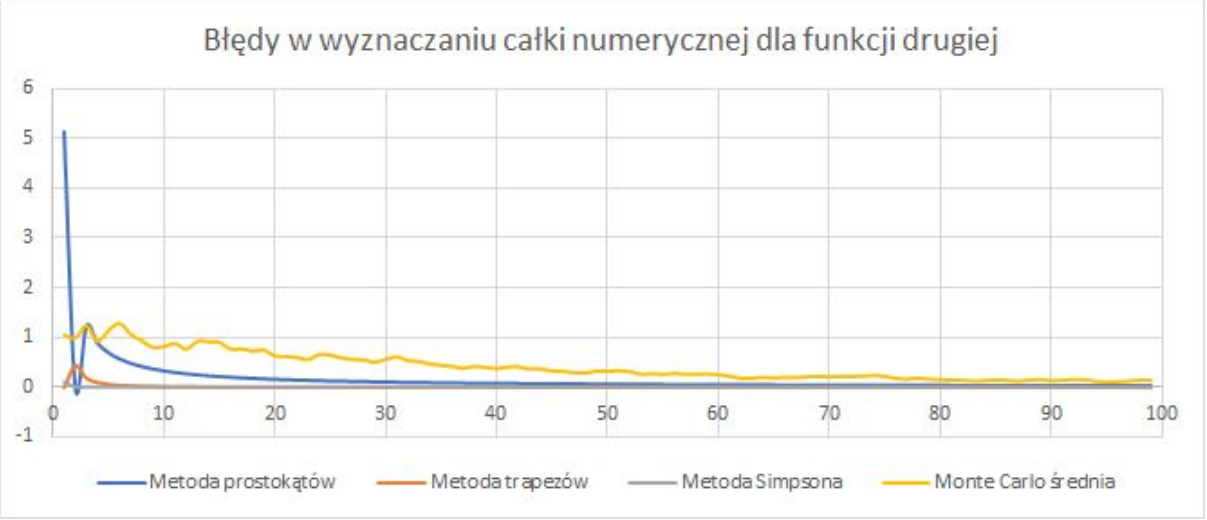


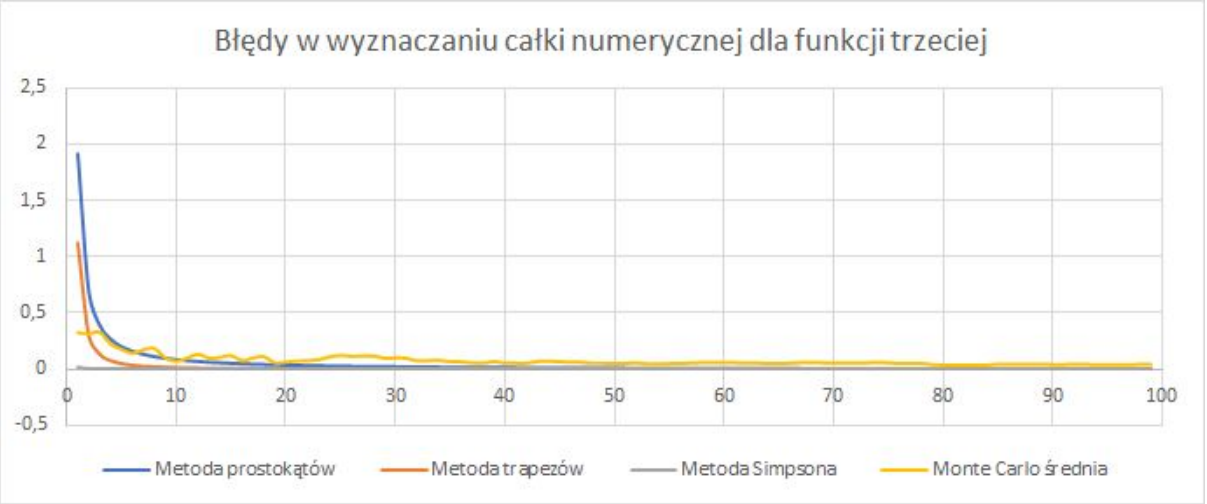
Zaimplementowana została funkcja realizująca całkowanie metodą Monte Carlo:

```
double integrationMC(double f(double x),
                     double x0,
                     double x1,
                     long long count) {
  srand(time(NULL));
  double result = 0;
  for (int i = 0; i < count; i++) {
    double x = (double)rand() * (x1 - x0) / (double)RAND_MAX + x0;
    result += (f(x) * (x1 - x0)) / count;
  }
  return result;
}
```

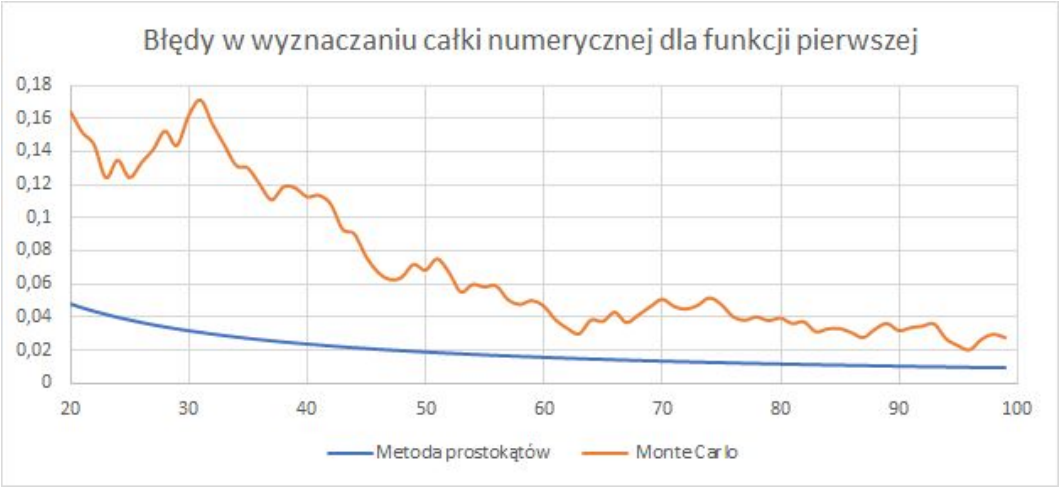
Podobnie jak dla pozostałych metod całkowania numerycznego przeprowadziłem pomiar dokładności między wartością wyznaczoną przez funkcję. Na wykresie użyłem średniej arytmetycznej z trzech pomiarów całkowania metodą Monte Carlo dla tej samej funkcji.

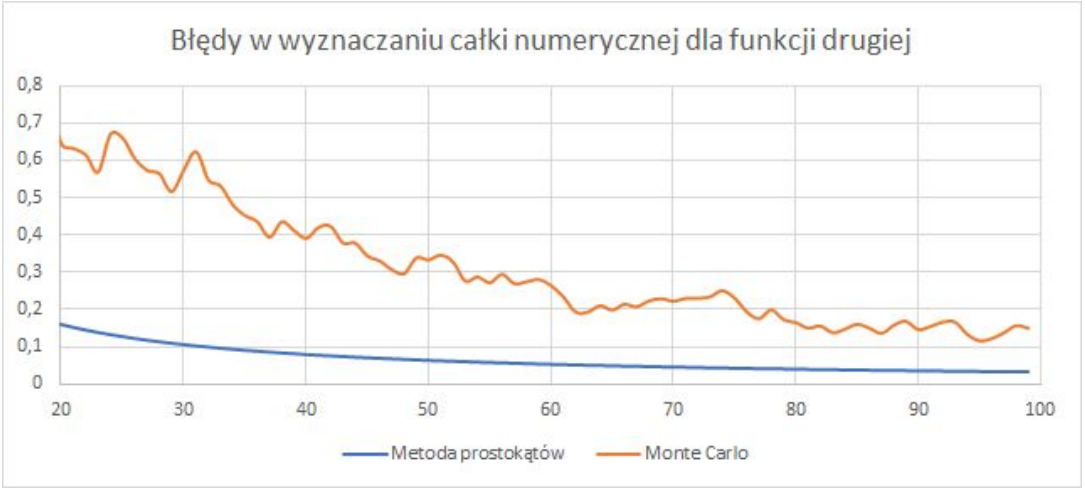


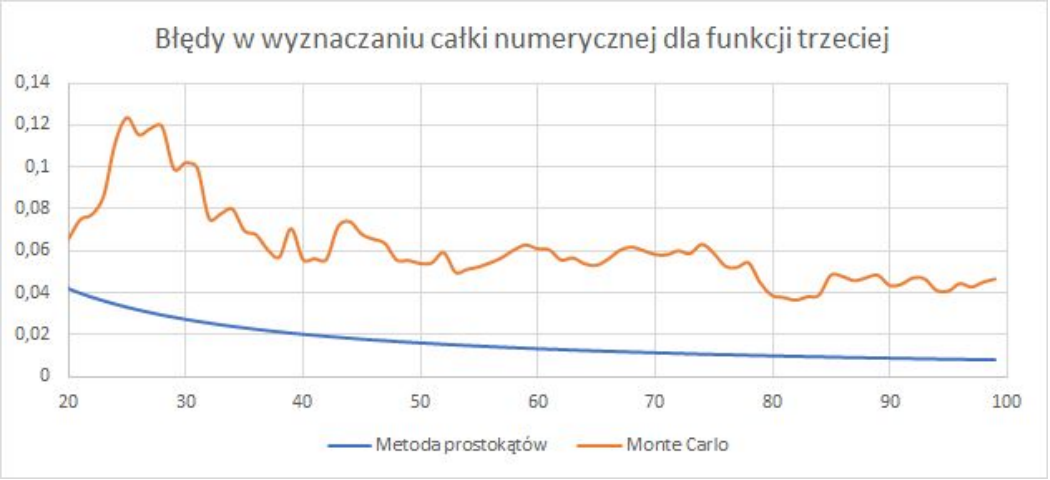




Bardzo dobrze widać losowość wyników dla metody Monte Carlo - mimo, że są uśrednione z trzech pomiarów, nadal dokładność jest czasami lepsza, czasami gorsza. Widać również, że dla tak małej liczby punktów, metoda Monte Carlo jest daje zdecydowanie najgorsze wyniki. Poniżej porównam jeszcze dokładniej metodę Monte Carlo z metodą prostokątów, która z poprzednich trzech metod miała najgorszą dokładność:







Teraz widać dokładniej, że w każdym przypadku metoda prostokątów daje lepsze rezultaty niż metoda Monte Carlo.

Policzyłem także różnicę między dokładnym wynikiem, wynikiem zwróconym przez metodę Monte Carlo dla 100 000 000 punktów - dla każdej z funkcji po trzy próby. Zostały zwrócone liczby:

	Funkcja pierwsza	Funkcja druga	Funkcja trzecia
Pierwsza próba	4,04E-05	0,000306	2,94E-06
Druga próba	2,16E-05	0,00015	4,06E-06
Trzecia próba	0,00011	0,000384	7,95E-05
Średnia	5,75E-05	2,80E-04	2,88E-05

Widzimy, że dla takiej ilości punktów obliczenia stały się zdecydowanie dokładniejsze. Można zatem zaobserwować wzrost dokładności wyniku wraz ze wzrostem liczby punktów.