



**WYDZIAŁ  
ELEKTROTECHNIKI  
I INFORMATYKI**  
POLITECHNIKI RZESZOWSKIEJ

**Kamil Madej**

Analiza i interpretacja wybranych metod całkowania  
numerycznego

**Praca dyplomowa inżynierska**

Opiekun pracy:

(dr. inż) Mariusz Borkowski (prof. PRz)

Rzeszów, 2022



# Spis treści

<b>1. Wstęp/wprowadzenie</b>	<b>5</b>
<b>2. Metody całkowania numerycznego</b>	<b>6</b>
2.1. Metody Newtona-Cotesa	6
2.1.1. Metoda prostokątów	7
2.1.2. Metoda trapezów	7
2.1.3. Metoda Simpsona	9
2.2. Kwadratury Gaussa	9
2.2.1. Gauss Legrande	10
2.3. Metody adaptacyjne	11
2.4. Metody Monte Carlo	12
2.4.1. Crude Monte Carlo	12
2.4.2. Monte Carlo	13
<b>3. Błędy bezwzględne w metodach całkowania</b>	<b>13</b>
<b>4. Bibliotek do całkowania numerycznego</b>	<b>14</b>
4.1. Przykład użycia biblioteki	16
4.1.1. Metoda prostokątów	16
4.1.2. Metoda trapezów	21
4.1.3. Metoda Simpsona	24
4.1.4. Metoda Gaussa Legendre’a	25
4.1.5. Metoda adaptacyjna trapezów	27
4.1.6. Metoda Crude Monte Carlo	28
4.1.7. Metoda Monte Carlo	31
4.2. Brak Metoda Adaptacyjna Trapezów	34
<b>5. Podsumowanie i wnioski końcowe</b>	<b>34</b>
<b>Załączniki</b>	<b>36</b>
<b>Literatura</b>	<b>37</b>



# 1. Wstęp/wprowadzenie

Całkowanie numeryczne, zwane też kwadraturą to szeroka rodzina algorytmów służących do obliczania liczbowej wartości całki oznaczonej. Przy pomocy kwadratur problemy matematyczne są formułowane w taki sposób, że można je rozwiązać przy pomocy operacji arytmetycznych.

Podstawowym problemem całkowania numerycznego jest obliczenie przybliżonego rozwiązania całki oznaczonej

$$\int_a^b f(x)dx \tag{1.1}$$

do pewnego stopnia przybliżenia

**Cel całkowania numerycznego** Istnieje wiele przypadków gdy całkowanie numeryczne jest lepszym rozwiązaniem niż całkowanie analityczne lub jednym rozwiązaniem. Całka z funkcji  $f(x)$  może być znana jedynie dla pojedynczych punktów, na przykład w przypadku próbkowania. Innym przypadkiem, gdzie numeryczne rozwiązanie jest lepsze jest przypadek, gdy wzór całki jest znany, ale znalezienie funkcji pierwotnej jest bardzo skomplikowane lub nie możliwe do zrobienia. Przykładem takiej funkcji jest  $f(x) = \exp(-x^2)$  wy

## 2. Metody całkowania numerycznego

Do 20% objętości pracy. W zależności od charakteru pracy ten rozdział powinien zawierać:

- a) opis tematyki zagadnienia – aktualny stan zagadnienia,
- b) metody i rozwiązania,
- c) dyskusja i krytyczna ocena stanu aktualnego,
- d) podsumowanie stanu wiedzy, techniki literaturowe itp.

### 2.1. Metody Newtona-Cotesa

W analizie matematycznej, metodami Newtona-Cotesa, nazywamy grupę metod do całkowania numerycznego, które bazują na oszacowaniu wartości całki na skończonym przedziale, przy użyciu interpolacji wielomianem odpowiedniego stopnia, wyznaczając  $n + 1$  równo rozmieszczonych punktów, które dzielą przedział całkowania na  $n$  podprzedziałów. [1]

Metodę Newtona-Cotesa dla  $n + 1$  punktów można zdefiniować jako:

$$\int_a^b f(x)dx \approx \sum_{i=0}^n w_i f(x_i) \quad (2.2)$$

gdzie  $x_i$  jest zbiorem równo rozmieszczonych punktów z przedziału  $[a, b]$

a  $w_i$  jest zbiorem wag

W metodzie Newtona-Cotesa wagi poszczególnych węzłów otrzymywane są poprzez interpolację wielomianem Lagrange’a:

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \int_a^b \sum_{i=0}^n f_i L_{n,j}(x)dx = \sum_{i=0}^n \left( \int_a^b L_{n,j}(x)dx \right) f_i = \sum_{i=0}^n w_i f_i \quad (2.3)$$

[2]

### 2.1.1. Metoda prostokątów

Metoda prostokątów to najprostszy wariant metody Newtona-Cotesa gdzie całkę  $\int_a^b f(x)$  przybliżamy przy użyciu interpolacji wielomianem Lagrange'a stopnia 1. Dla pojedynczego podprzedziału metoda prostokątów wygląda następująco:

$$\int_a^b f(x) = I = f(a) * h \quad (2.4)$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = h * \sum_{i=0}^{n-1} f(x_i) \quad (2.5)$$

gdzie h jest szerokością pojedynczego podprzedziału i równa się  $h = \frac{b-a}{n}$  [3]

Listing 1 pokazuje przykładowy kod w języku Python implementujący metodą trapezów.

```
1 import numpy as np
2
3 def f_rectI(f,a,b,n):
4     h = (b - a) / n
5
6     X = np.linspace(a + 0.5 * h, b - 0.5 * h, num=n)
7
8     Y = []
9     for i in range(0, n):
10         Y.append(f(X[i]))
11
12     Y = np.array(Y)
13
14     I = h * Y
15     I = np.sum(I)
16     return I
```

Listing 1: Kod w języku python implementujący metodę prostokątów

### 2.1.2. Metoda trapezów

Metoda trapezów to kolejny wariant metody Newtona-Cotesa, w którym całkę  $\int_a^b f(x)$  przybliżamy przy użyciu wielomianu Lagrange'a stopnia 2. Dla pojedynczego

podprzedziału metoda trapezów wygląda następująco:

$$\int_a^b f(x) = \frac{1}{2} * h[f(a) + f(b)] \quad (2.6)$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = \frac{1}{2} \sum_{i=0}^n [f(x_i) + f(x_{i+1})] \quad (2.7)$$

Listing 2 pokazuje przykładowy kod w języku Python implementujący metodą trapezów.

```
1 import numpy as np
2
3
4 def f_trapI(f, a, b, n):
5     h = (b - a) / n
6
7     X = np.linspace(a, b, num=n + 1)
8
9     Y = []
10    for i in range(0, n + 1):
11        Y.append(f(X[i]))
12
13    Y = np.array(Y)
14
15    I = []
16    for i in range(0, n + 1):
17        if i == 0 or i == n:
18            I.append(h * Y[i] / 2)
19        else:
20            I.append(h * Y[i])
21
22    I = np.sum(I)
23    return I
```

Listing 2: Kod w języku python implementujący metodę trapezów



### 2.1.3. Metoda Simpsona

Metoda Simpsona zwana też metodą parabol, to kolejny wariant metody Newtona-Cotesa. Tym razem całkę  $\int_a^b f(x)$  przybliżamy przy użyciu wielomianu Lagrange'a stopnia 3, czyli paraboli.

Dla pojedynczego podprzedziału metoda trapezów wygląda następująco:

$$\int_a^b f(x) = \frac{1}{3} * h[f(a) + 4f(a+h) + f(a+2h)] \quad (2.8)$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = \frac{1}{2}h[f(a) + f(b)] + \sum_{i=1,3,5}^{n-1} 4f(x_i) + \sum_{i=2,4,6}^{n-2} f(x_i) \quad (2.9)$$

Listing 3 pokazuje przykładowy kod w języku Python implementujący metodą simpsona.

```
1
2 def f_simp2I(f,a,b,n):
3
4     h = (b-a)/n
5
6     I = f(a) + f(b)
7
8     for i in range(1,n):
9         if i%2==1:
10             I += 4 * f(a + i*h)
11         else:
12             I += 2 * f(a + i*h)
13
14     I = h/3 * I
15     return I
```

Listing 3: Kod w języku python implementujący metodę simpsona

## 2.2. Kwadratury Gaussa

Kwadratury Gaussa to metody numeryczne służące do przybliżania skończonych całek, najczęściej określane jako ważona suma wartości funkcji w określonych punktach dziedziny całkowania. Całkowanie metodą Gauss'a opiera się na użyciu wielomianów to

przybliżenia funkcji podcałkowej  $f(x)$  na przedziale  $[-1,1]$ , poprzez użycie odpowiednich węzłów  $x_i$  oraz wag  $w_i$ .

$$\int_{-1}^1 f(x)dx \approx \sum_{i=1}^n w_i f(x_i) \quad (2.10)$$

Dokładność i optymalność wyniku całkowania zależy od odpowiedniego wyboru wielomianu interpolacyjnego. Użycie wielomianów Legendre’a całkowanie metodą Gaussa daje dokładny wynik dla wielomianów stopnia  $2n - 1$  lub niższego.

### 2.2.1. Gauss Legrande

Kwadratura Gaussa-Legendre’a jest specjalny przypadek kwadratury Gaussa, która pozwala na efektywne przybliżenie funkcji ze znanym zachowaniem asymptotycznym na brzegach przedziału całkowania. Kwadratura Gaussa jest szczególnie zalecana, jeśli całka jest holomorficzną w sąsiedztwie przedziału całkowania.

Węzły  $x_i$  są pierwiastkami wielomianu Legendre’a  $P(x)$  stopnia  $n$ . Nie istnieje prosty sposób wyznaczenia pierwiastków  $x_i$ , mogą one jednak zostać aproksymowane z dużą dokładnością przy użyciu wzoru:

$$x_i \approx \cos\left(\pi \frac{\frac{1}{2} + i}{N}\right) \quad (2.11)$$

Wagi  $w_i$  można wyrazić wzorem:

$$w_i = \frac{2(1-x_i^2)}{[nP_{n-1}(x_i)]^2} = \frac{2}{[P_n'(x_i)]^2} \quad (2.12)$$

[4]

Listing 4 pokazuje przykładowy kod w języku Python implementujący kwadraturę Gauss Legendre’a.

```

1
2 import numpy as np
3
4 def f_gauss_legrande(f,a,b,n):
5     half = float(b-a)/2
6     mid = (a+b)/2
7     [t,w] = np.polynomial.legendre.leggauss(n)
8
9     I = 0

```

```

10     for i in range(n):
11         I += w[i] * f(mid+half*t[i])
12
13     I *= half
14
15     return I

```

Listing 4: Kod w języku python implementujący metodę simpsona

### 2.3. Metody adaptacyjne

Tradycyjne metody całkowania Newtona-Cotesa, ignorują fakt, że całkowana funkcja posiada regiony o dużej, jak i małej zmienności. Metody adaptacyjne rozwiązują ten problem poprzez dostosowanie wielkości podprzedziałów, na mniejsze, w miejscach gdzie funkcja zmienia się gwałtownie i większe, w miejscach o mniejszej zmienności.[5]

Metody adaptacyjne polegają na wykorzystaniu tradycyjnych metod całkowania takich jak: Metoda Simpsona, trapezów do obliczenia całki na przedziale  $[a, b]$  aadaną dokładnością  $\xi$ . Jeśli po pierwszej iteracji wartość całki nie jest dostatecznie dokładna, to przedział dzielimy na połowy, i ponownie całkujemy każdą z otrzymanych połówek. Proces ten powtarzamy do uzyskania zadanej dokładności.[6]

Całkowita wartość całki jest obliczana jako suma przybliżeń całki na wszystkich podprzedziałach.

Na przykładzie metody trapezów adaptacyjna metoda wygląda następująco Przedziale  $[a, b]$  jest dzielona na  $n$  podprzedziałów  $[a_j, b_j]$ , dla  $j = 0, 1, \dots, n-1$ , a następnie dla każdego podprzedziału obliczana jest całka według wzoru:

$$I_j(f) = \int_{a_j}^{b_j} f(x) dx \quad (2.13)$$

Powyższe podejście nie różni się niczym od klasycznej metody trapezów, jednak w metodach adaptacyjnych podprzedział  $[a_j, b_j]$  jest dzielony na pół, gdy wartość  $I_j(f)$  nie została obliczona z adadaną dokładnością. Do ustalenia dokładności, używamy kwadratur na przedziale  $[a_j, b_j]$ , by uzyskać przybliżenie  $I_{1j}$ , a następnie całkujemy funkcję dzieląc przedział  $[a_j, b_j]$  na dwa podprzedziały, by obliczyć drugie przybliżenie  $I_2$ . Jeśli  $I_1$  oraz  $I_2$  są dostatecznie zbliżone, wtedy możemy stwierdzić, że przybliżenie jest wystarczające i nie ma potrzeby dalszego dzielenia  $[a_j, b_j]$ . W przeciwnym przypadku

dzielimy  $[a_j, b_j]$  na dwa podprzedziały i powtarzamy proces ponownie. Używamy tej techniki na wszystkich podprzedziałach tak długo, aż funkcja podcałkowa  $f$  zostanie przybliżona zadaną dokładnością.[7]

Listing 5 pokazuje przykładowy kod w języku Python implementujący adaptacyjną metodę trapezów.

```
1
2 def f_trapI(f, xa, xb):
3     h = xb-xa
4     I = h*(f(xa) + f(xb))/2
5     return I
6
7 def f_adapt(f, ax, bx, tol):
8     m = (ax+bx)/2.0
9     P1 = f_trapI(f, ax, m)
10    P2 = f_trapI(f, m, bx)
11
12    if abs(P1 - P2) < 3 * tol:
13        return P2
14    else:
15        return f_adapt(f, ax, m, tol/2) + f_adapt(f, m, bx, tol)
```

Listing 5: Kod w języku python implementujący metodę simpsona

## 2.4. Metody Monte Carlo

Metoda Monte Carlo to zupełnie odmienne od metod Newtona Cotesa oraz metod Gaussa podejście do obliczania wartości całki. Polega ona na obliczaniu pola powierzchni pod krzywą używając losowo rozmieszczonych punktów w obrębie granic całkowania.

Istnieją dwa rodzaje metod Monte Carlo.

### 2.4.1. Crude Monte Carlo

Podstawowa metoda Monte Carlo, zwana też Crude Monte Carlo polega ona na wylosowaniu  $n$  punktów w obrębie przedziału całkowania i na podstawie tych danych obliczenie średniej wartości funkcji. [9]

$$f_{sr} = \frac{f(x_1) + f(x_2) + \dots + f(x_n)}{n} \quad (2.14)$$

Przybliżoną wartość całki otrzymujemy dzieląc uzyskaną średnią wartość funkcji przez długość przedziału całkowania.

$$I = f_{sr} * |b - a| \quad (2.15)$$

### 2.4.2. Monte Carlo

Bardziej dokładną wersją jest metoda Monte Carlo, która polega na wylosowaniu  $n$  punktów znajdujących się w polu kwadratu, który wyznaczany jest przez przedział całkowania  $< a, b >$  oraz zakres wartości funkcji w tym przedziale  $< f(a), f(b) >$ . Po wylosowaniu  $n$  punktów wartość całki wyrażana jest jako:

$$I = P * \frac{c}{n} \quad (2.16)$$

gdzie  $c$  to ilość punktów leżących się pod krzywą.

Wraz ze zwiększaniem ilości losowanych punktów, rozkładają się one bardziej równomiernie w obrębie wyznaczonego prostokąta, dając coraz dokładniejszy wynik.[10]

## 3. Błędy bezwzględne w metodach całkowania

Błąd bezwzględny w metodzie prostokątów wyraża się wzorem:

$$E_R = f'(c) \int_a^b (t - a) dt = \frac{f'(c)}{2} (b - a)^2 \quad (3.17)$$

Błąd bezwzględny w metodzie trapezów u wyraża się wzorem:

$$E_T = -\frac{1}{12} f''(\xi) (b - a)^3 \quad (3.18)$$

gdzie  $\xi$  leży gdzieś w przedziale  $[a, b]$ . Równanie wskazuje, że jeśli całkowana funkcja jest liniowa, metoda trapezów będzie przybliżała funkcję dokładnie.

Błąd dla wielu złożonej metody trapezów może być uzyskany przez zsumowanie pojedynczych błędów w każdym segmencie

$$E_T = -\frac{(b - a)^3}{12n^3} \sum_{i=1}^n f''(\xi_i) \quad (3.19)$$

Według powyższego wzoru, jeśli ilość segmentów zostanie podwojona, błąd bezwzględny zostanie zmniejszony czterokrotnie.

Błąd bezwzględny metody Simpsona 1/3 dla pojedynczego segmentu ma postać

$$E_t = -\frac{(b-a)^5}{2880} f^4(\xi) \quad (3.20)$$

Metoda simpsona daje dokładny wynik dla wielomianów stopnia 3.

Podobnie jak metoda Trapezów, złożona metoda Simpsona, jest sumą indywidualnych błędów w poszczególnych segmentach

$$E_a = -\frac{(b-a)^5}{180n^4} \bar{f}^4(\xi) \quad (3.21)$$

gdzie  $\bar{f}^4$  jest średnią czwartą pochodną przedziału

Ogólny błąd dla kwadratury Gaussa- Legendre'a wyraża się wzorem:

$$E_t = \frac{2^{2n+3}[(n+1)!]^4}{(2n+3)[(2n+2)!]^3} f^{2n+2}(\xi) \quad (3.22)$$

gdzie  $n$  - liczba punktów minus jeden  $f^{2n+2}(\xi)$  -  $(2n+2)$  pochodna funkcji po zmianie zmiennej na  $\xi$  ulokowanej gdzieś w przedziale  $[-1,1]$ .

## 4. Bibliotek do całkowania numerycznego

Następne podrozdziały przedstawiają wyniki i omówienie działania zaimplementowanych przeze mnie metod całkowania dla równan (4.23) (4.24) (4.25). Dla metod całkowania, których logiką działania jest liczenie całki przez podział przedziału całkowania na podprzedziały, skuteczność metody wyrażana będzie w ilości podprzedziałów, na które musiał zostać podzielony przedział całkowania, by otrzymać dokładność przybliżenia rzędu 0.01 lub jeśli to możliwe, dokładną wartość całki. Dla metod Monte Carlo skuteczność liczona będzie ilością punktów potrzebnych do uzyskania zadowalającej dokładności. W adaptacyjnej metodzie trapezów miarą skuteczności będzie ilość wywołań funkcji, potrzebna do uzyskania zadanej dokładności przybliżenia funkcji.

Dokładność wszystkich metod całkowania zostanie zbadana na podstawie poniższych funkcji:

Funkcja wielomianowa

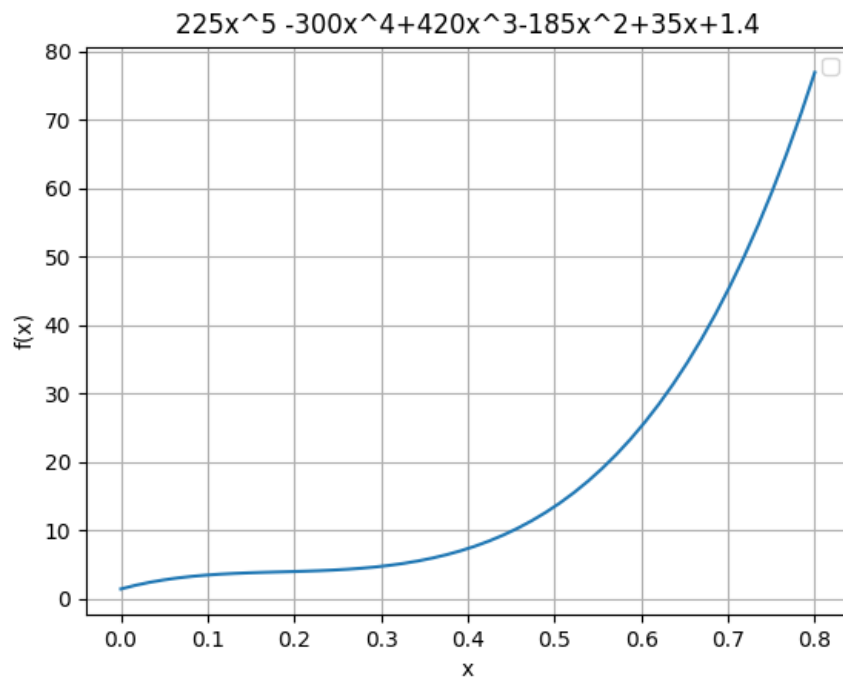
$$f(x) = 225x^5 - 300x^4 + 420x^3 - 185x^2 + 35x + 1.4 \quad (4.23)$$

Funkcja trygonometryczna

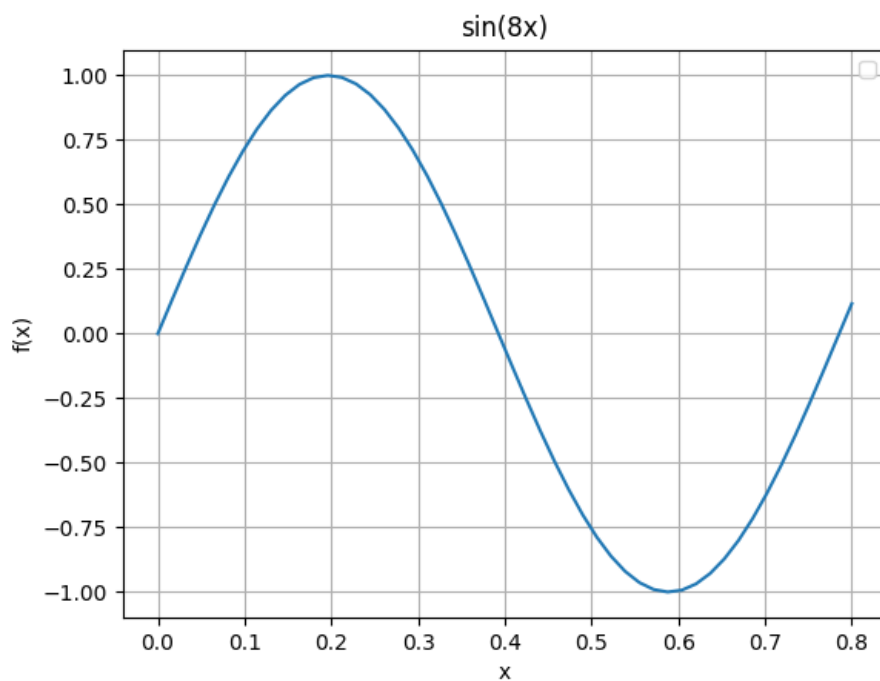
$$f(x) = \sin(8x) \quad (4.24)$$

Funkcja mieszana

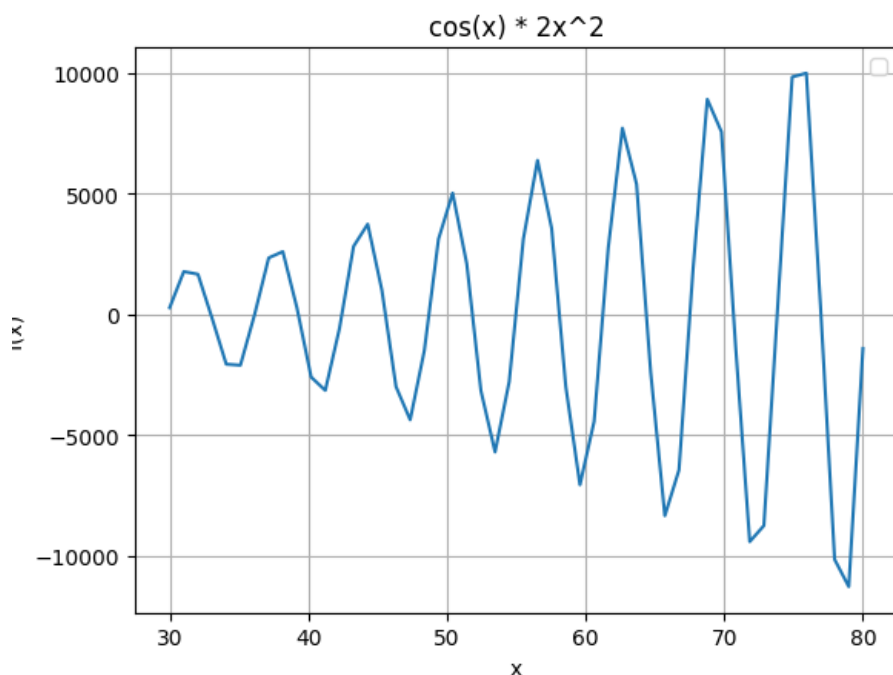
$$f(x) = \cos(2x + 2) + 13x^2 \quad (4.25)$$



Rysunek 4.1: Wykres funkcji zdefiniowanej wzorem(4.23)



Rysunek 4.2: Wykres funkcji zdefiniowanej wzorem(4.24)



Rysunek 4.3: Wykres funkcji zdefiniowanej wzorem(4.25)

## 4.1. Przykład użycia biblioteki

### 4.1.1. Metoda prostokątów

Funkcja implementująca metodę prostokątów, przyjmuje 4 parametry. Pierwszym parametrem jest badana funkcja, drugim początek dolny przedział całkowania, trzecim górny przedział całkowania, a ostatnim, ilość podprzedziałów na które dzielimy odcinek  $[a, b]$ . Metoda zwraca przybliżoną wartość całki.

Tabela (4.1) przedstawia dane otrzymane w wyniku działania implementacji metody Prostokątów dla funkcji (4.23) na przedziale  $[0, 0.8]$ . Pierwszą kolumną tabeli jest obliczona wartość całki, drugą ilość podprzedziałów dla danej iteracji, a trzecią błąd metody wyrażony jako różnica wartości całki obliczonej analitycznie i wartości obliczonej numerycznie.

Jak widać w tabeli początkowo dla pojedynczego podprzedziału błąd metody był bardzo duży i wynosił  $-8.08110$ . Wynika to z kształtu funkcji (4.23).

Do osiągnięcia założonej dokładności metoda prostokątów potrzebowała 31 podprzedziałów.

```
1 def f_rectI(f, a, b, n):
```



```
2     """
3     """
```

Listing 6: Kod w języku python implementujący metodę prostokątów



Tabela 4.1: Dane z iteracji w metodzie prostokątów dla funkcji wielomianowej

	Wartość całki	l.podp	Błąd
0	5.843200	1	-8.081100
1	11.635200	2	-2.289100
2	12.884780	3	-1.039520
3	13.335200	4	-0.589100
4	13.545974	5	-0.378326
5	13.661077	6	-0.263223
6	13.730687	7	-0.193613
7	13.775950	8	-0.148350
8	13.807020	9	-0.117280
9	13.829263	10	-0.095037
10	13.845731	11	-0.078569
11	13.858262	12	-0.066038
12	13.868017	13	-0.056283
13	13.875760	14	-0.048540
14	13.882008	15	-0.042292
15	13.887122	16	-0.037178
16	13.891361	17	-0.032939
17	13.894914	18	-0.029386
18	13.897921	19	-0.026379
19	13.900489	20	-0.023811
20	13.902699	21	-0.021601
21	13.904614	22	-0.019686
22	13.906286	23	-0.018014
23	13.907752	24	-0.016548
24	13.909047	25	-0.015253
25	13.910195	26	-0.014105
26	13.911218	27	-0.013082
27	13.912133	28	-0.012167
28	13.912955	29	-0.011345
29	13.913697	30	-0.010603
30	13.914367	31	-0.009933

**Funkcja wielomianowa** Tabela (4.2) przedstawia dane zwrócone z działania metody prostokątów dla funkcji (4.24) na przedziale  $[0, 0.8]$ . Początkowo wartość błędu obliczonej całki jest duża, z racji przyjętego kształtu funkcji oraz przyjętego przedziału całkowania. Obie wartości funkcji na krańcach przedziałów są dodatnie oraz leżą bardzo blisko osi odciętych, przez co prostokąt przybliżający funkcję nijak ma się do rzeczywistego kształtu funkcji. Wraz ze zwiększającą się ilością podprzedziałów dokładność przybliżenia znacząco rośnie, aż dla 16 podprzedziałów przyjmuję zakładaną dokładność.

Tabela 4.2: Dane z iteracji w metodzie prostokątów dla funkcji trygonometrycznej

	Wartość całki	l.podp	Błąd
0	1.978716	1	1.734009
1	-1.293375	2	-1.538083
2	1.427055	3	1.182348
3	0.538234	4	0.293527
4	0.391699	5	0.146991
5	0.335697	6	0.090990
6	0.307385	7	0.062677
7	0.290809	8	0.046102
8	0.280172	9	0.035465
9	0.272899	10	0.028192
10	0.267689	11	0.022982
11	0.263820	12	0.019113
12	0.260863	13	0.016156
13	0.258550	14	0.013843
14	0.256705	15	0.011998
15	0.255209	16	0.010502
16	0.253978	17	0.009271

### Funkcja trygonometryczna

**Funkcja mieszana** Tabela (4.2) prezentuje wyniki działania metody prostokątów dla funkcji (4.25) na przedziale  $[0, 2]$ . Na krańcach przedziału całkowania wartości cał-

kowanej funkcji przyjmują wartości dodatnie, przez co prostokąt przybliżający pomija zupełnie ujemnie wartości funkcji, co skutkuje znacznym dodatnim przeszacowaniem funkcji. Z racji bardzo dynamicznie zmieniających się wartości funkcji sytuacja ta powtarza się dla kilku kolejnych iteracji. Wraz z rosnącą ilością podprzedziałów funkcja jest coraz lepiej przybliżana przez funkcję, by przy 133 podprzedziałach przyjąć wartość zadaną dokładnością.

Tabela 4.3: Dane z iteracji w metodzie prostokątów dla złożenia funkcji wielomianowej i trygonometrycznej

	Wartość całki	l.podp	Błąd
0	19.299321	1	19.690535
1	23.217411	2	23.608625
2	4.506768	3	4.897982
3	24.144424	4	24.535638
4	3.105923	5	3.497138
*	*	*	*
129	-0.401735	130	-0.010521
130	-0.401572	131	-0.010358
131	-0.401413	132	-0.010199
132	-0.401257	133	-0.010043
133	-0.401106	134	-0.009891

#### 4.1.2. Metoda trapezów

Funkcja implementująca metodę trapezów, przyjmuje dokładnie te same parametry co metoda prostokątów. Pierwszym parametrem jest funkcja podcałkowa, kolejnymi dwoma kolejno górny i dolny przedział całkowania, a ostatnim ilość podprzedziałów, na które dzielimy przedział całkowania. Funkcja zwraca przybliżoną wartość całki.

```

1 def f_trapI(f, a, b, n):
2     """
3     """

```

Listing 7: Kod w języku python implementujący metodę trapezów

**Funkcja wielomianowa** Tabela (4.4) przedstawia wyniki uzyskane z działania metody trapezów na funkcji (4.23).

Dla pojedynczego podprzedziału błąd funkcji jest bardzo duży i wynika, z tego, że wartość funkcji dla końcowych argumentów z przedziału gwałtownie rośnie, przez co prosta łącząca krańce przedziału całkowania robi to z dużym nadmiarem. Po podzieleniu podprzedziału na 2, błąd maleje o 1 rząd wielkości, a po kolejnych 3 o kolejny rząd. Do otrzymania zadanej dokładności metoda trapezów potrzebuje 44 podprzedziałów.

**Funkcja trygonometryczna** Tabela (4.5) przedstawia dane uzyskane z metody trapezów na funkcji podcałkowej (4.24). Największy błąd dla tej funkcji pojawia się, gdy ilość podprzedziałów wynosi 3, dzieje się tak dlatego, że wartości funkcji dla granic środkowego podprzedziału przyjmują bardzo niskie wartości, przez co trapez przybliżający funkcję na tym przedziale pomija całkowicie całą dodatnią część sinusoidy.

Metoda trapezów przyjmuje wartość zadaną dokładnością przy ilości podprzedziałów równej 23.

że przy takim podziale każdy prawy kraniec danego podprzedziału znajduje się pod osią odciętych oraz dla każdego punktu granicznego wartość funkcji przyjmują niską wartość, co skutkuje tym, że trapezy przybliżające funkcję podcałkową

Tabela 4.5: Dane z iteracji w metodzie trapezów dla funkcji trygonometrycznej

	Wartość całki	l.podp	Błąd
0	-0.287903	1	-0.532611
1	0.845407	2	0.600699
2	-1.269118	3	-1.513825
3	-0.223984	4	-0.468692
4	-0.011437	5	-0.256145
5	0.078969	6	-0.165739
6	0.127564	7	-0.117144
7	0.157125	8	-0.087583
8	0.176585	9	-0.068122
9	0.190131	10	-0.054577
10	0.199961	11	-0.044746
11	0.207333	12	-0.037375
12	0.213008	13	-0.031699
13	0.217474	14	-0.027233
14	0.221053	15	-0.023654
15	0.223967	16	-0.020740
16	0.226371	17	-0.018336
17	0.228379	18	-0.016329
18	0.230073	19	-0.014635
19	0.231515	20	-0.013192
20	0.232754	21	-0.011954
21	0.233825	22	-0.010882
22	0.234758	23	-0.009949

**Funkcja mieszana** Tabela (4.6) przedstawia wyniki uzyskanie z działania metody trapezów na funkcji (4.25). W pierwszych 4 iteracjach widzimy bardzo rozbieżne wyniki. Funkcja (4.25) zmienia się bardzo dynamicznie na zadanym przedziale, przez co mała liczba podprzedziałów może dawać wyniku zupełnie odbiegające od realnej wartości całki. Do uzyskania zadanej dokładności metoda trapezów potrzebowała 188 podprzedziałów.

Tabela 4.6: Dane z iteracji w metodzie trapezów dla funkcji mieszanej

	Wartość całki	l.podp	Błąd
0	34.492755	1	34.883969
1	26.896038	2	27.287252
2	1.374685	3	1.765899
3	25.056724	4	25.447939
4	-1.163902	5	-0.772688
*	*	*	*
183	-0.380849	184	0.010365
184	-0.380962	185	0.010253
185	-0.381072	186	0.010142
186	-0.381181	187	0.010033
187	-0.381288	188	0.009926

### 4.1.3. Metoda Simpsona

```

1 def f_simp2I(f,a,b,n):
2     """
3     """

```

Listing 8: Kod w języku python implementujący metodę prostokątów

**Funkcja wielomianowa** Tabela (4.7) przedstawia wyniki uzyskanie z działania metody Simpsona dla funkcji podcałkowej (4.23). Metoda Simpsona potrzebowała 6 podprzedziałów by osiągnąć zadaną dokładność.

Tabela 4.7: Dane z iteracji w metodzie Simpsona dla funkcji wielomianowej

	Wartość całki	l.podp	Błąd
0	14.333867	2	0.409567
1	13.949867	4	0.025567
2	13.929323	6	0.005023



**Funkcja trygonometryczna** Tabela (4.8) przedstawia wyniki uzyskanie z działania metody Simpsona dla funkcji podcałkowej (4.24). Metoda Simpsona potrzebowała 12 podprzedziałów by osiągnąć zadaną dokładność.

Tabela 4.8: Dane z iteracji w metodzie Simpsona dla funkcji trygonometrycznej

	Wartość całki	l.podp	Błąd
0	1.223177	2	0.978469
1	-0.580448	4	-0.825156
2	0.528331	6	0.283623
3	0.284161	8	0.039454
4	0.257320	10	0.012613
5	0.250121	12	0.005413

**Funkcja mieszana** Tabela (4.9) przedstawia wyniki uzyskanie z działania metody Simpsona dla funkcji podcałkowej (4.25). Metoda Simpsona potrzebowała 76 podprzedziałów by osiągnąć zadaną dokładność.

Tabela 4.9: Dane z iteracji w metodzie Simpsona dla funkcji mieszanej

	Wartość całki	l.podp	Błąd
0	24.363799	2	24.755013
1	24.443620	4	24.834834
2	3.462740	6	3.853954
3	24.448524	8	24.839738
4	1.682648	10	2.073863
*	*	*	*
33	-0.406964	68	-0.015750
34	-0.404972	70	-0.013757
35	-0.403293	72	-0.012078
36	-0.401868	74	-0.010654
37	-0.400653	76	-0.009438

#### 4.1.4. Metoda Gaussa Legendre'a

```

1 def f_gauss_legrande(f,a,b,n):
2     """
3     """

```

Listing 9: Kod w języku python implementujący metodę trapezów

**Funkcja wielomianowa** Tabela (4.10) przedstawia wyniki uzyskanie z działania metody Gaussa Legendre’a dla funkcji podcałkowej (4.23). Funkcja , dla której liczona jest całka jest wielomianem stopnia 5, a metoda Gaussa Legendre’a jest dokładna dla wielomianów stopnia  $2n + 1$ , dlatego do uzyskania zadanej dokładności potrzebowała jedynie 3 podprzedziałów.

Tabela 4.10: Dane z iteracji w metodzie Gauss’a Legendre’a dla funkcji wielomianowej

	Wartość całki	l.podp	Błąd
0	5.843200	1	-8.081100
1	13.651200	2	-0.273100
2	13.924267	3	-0.000033

**funkcja trygonometryczna** Tabela (4.11) przedstawia wyniki uzyskanie z działania metody Gaussa Legendre’a dla funkcji podcałkowej (4.24).

Tabela 4.11: Dane z iteracji w metodzie Gauss’a Legendre’a dla funkcji trygonometrycznej

	Wartość całki	l.podp	Błąd
0	1.978716	1	1.734009
1	-0.184912	2	-0.429619
2	1.974615	3	1.729907
3	-0.611561	4	-0.856269
4	0.456455	5	0.211748
5	0.212634	6	-0.032073
6	0.248022	7	0.003315

**Funkcja mieszana** Tabela (4.12) przedstawia wyniki uzyskanie z działania metody Gaussa Legendre’a dla funkcji podcałkowej (4.25).

Tabela 4.12: Dane z iteracji w metodzie Gauss’a Legendre’a dla funkcji mieszanej

	Wartość całki	l.podp	Błąd
0	19.299321	1	19.690535
1	-24.723636	2	-24.332421
2	21.272072	3	21.663286
3	-1.592403	4	-1.201189
4	9.728832	5	10.120047
5	4.126645	6	4.517860
6	7.358866	7	7.750081
7	-7.385625	8	-6.994411
8	-5.565234	9	-5.174020

#### 4.1.5. Metoda adaptacyjna trapezów

```

1 def f_adapt(f, ax, bx, tol):
2     """
3     """

```

Listing 10: Kod w języku python implementujący metodę trapezów

**Funkcja wielomianowa** Tabela (4.13) przedstawia wyniki uzyskanie z działania metody Adaptacyjnej Trapezów dla funkcji podcałkowej (4.23).

Tabela 4.13: Dane z iteracji w metodzie Adaptacyjnej Trapezów dla funkcji mieszanej

	Wartość całki	l.wywołań funkcji	Tol	Błąd
0	14.017668750000004	8	0.1	-0.09336875000000333
1	13.944175145339969	28	0.01	-0.01987514533996837
2	13.927649165493257	82	0.001	-0.0033491654932564785
3	13.924743300654185	206	0.001	-0.00044330065418485276

**Funkcja trygonometryczna** Tabela (4.14) przedstawia wyniki uzyskanie z działania metody Adaptacyjnej Trapezów dla funkcji podcałkowej (4.24).

Tabela 4.14: Dane z iteracji w metodzie Adaptacyjnej Trapezów dla funkcji trygonometrycznej

	Wartość całki	l.wywołań funkcji	Tol	Błąd
0	0.25261703598873475	14	0.1	-0.007909600948311668
1	0.2431580077253209	40	0.01	0.0015494273151021865
2	0.24374843688117723	94	0.001	0.0009589981592458552
3	0.2445954848450854	248	0.001	0.00011195019533769002

## Funkcja Mieszana

### 4.1.6. Metoda Crude Monte Carlo

```

1 def f_crudeMonteC(f,a,b,n):
2     """
3     """

```

Listing 11: Kod w języku python implementujący metodę trapezów

**Funkcja wielomianowa** Tabela (4.15) przedstawia wyniki uzyskanie z działania metody Crude Monte Carlo dla funkcji podcałkowej (4.23).

Tabela 4.15: Dane z iteracji w metodzie Crude Monte Carlo dla funkcji wielomianowej

	Średnia wartość całki	l.podp	Błąd
0	11.987235	1	1.277747
1	12.144665	2	-0.803306
2	13.335565	5	0.800181
3	13.662372	10	-0.173489
4	13.675885	25	0.35888
5	14.043784	50	-0.043648
6	13.98955	100	0.021207
7	14.017083	250	0.019494
8	13.941606	500	-0.032747
9	13.941598	1000	0.040999
10	14.027992	2500	0.017816
11	13.926729	5000	-0.005491
12	13.906282	10000	-0.015816

**Funkcja trygonometryczna** Tabela (4.16) przedstawia wyniki uzyskanie z działania metody Crude Monte Carlo dla funkcji podcałkowej (4.24).

Tabela 4.16: Dane z iteracji w metodzie Crude Monte Carlo dla funkcji trygonometrycznej

	Średnia wartość całki	l.podp	Błąd
0	0.438109	1	0.053971
1	0.344761	2	0.005632
2	0.290026	5	0.112585
3	0.190505	10	-0.016218
4	0.234674	25	-0.009399
5	0.28502	50	-0.006675
6	0.261066	100	0.000407
7	0.236683	250	-0.001331
8	0.260367	500	0.001856
9	0.246922	1000	0.002864
10	0.242618	2500	-0.00156
11	0.244954	5000	-0.000572
12	0.243623	10000	-0.000763

**Funkcja mieszana** Tabela (4.17) przedstawia wyniki uzyskanie z działania metody Crude Monte Carlo dla funkcji podcałkowej (4.25).

Tabela 4.17: Dane z iteracji w metodzie Crude Monte Carlo dla funkcji mieszanej

	Średnia wartość całki	l.podp	Błąd
0	3.492531	1	-0.72964
1	0.822848	2	0.606175
2	0.326543	5	-0.60249
3	0.666921	10	0.835416
4	0.248838	25	0.063623
5	-0.131566	50	0.135121
6	-0.239317	100	-0.120759
7	-0.867171	250	-0.065519
8	-0.420814	500	-0.00342
9	-0.383231	1000	0.067716
10	-0.475464	2500	0.002454
11	-0.428383	5000	0.014121
12	-0.394148	10000	0.013985

#### 4.1.7. Metoda Monte Carlo

```

1 def f_MonteC(f,a,b,n):
2     """
3     """

```

Listing 12: Kod w języku python implementujący metodę trapezów

**Funkcja wielomianowa** Tabela (4.18) przedstawia wyniki uzyskane z działania metody Monte Carlo dla funkcji podcałkowej (4.23).

Tabela 4.18: Dane z iteracji w metodzie Monte Carlo dla funkcji wielomianowej

	Średnia wartość całki	l.podp	Błąd
0	12.449432	1	1.242667
1	14.334916	2	-0.3618
2	15.302392	5	-0.176548
3	13.937148	10	0.406375
4	14.03971	25	0.276175
5	14.330742	50	-0.456621
6	14.412072	100	0.097807
7	13.981018	250	-0.006538
8	14.016232	500	0.17736
9	13.896245	1000	0.035619
10	13.891548	2500	0.017377
11	13.897097	5000	0.005051
12	13.93564	10000	0.003088

**Funkcja trygonometryczna** Tabela (4.19) przedstawia wyniki uzyskanie z działania metody Monte Carlo dla funkcji podcałkowej (4.24).



Tabela 4.19: Dane z iteracji w metodzie Monte Carlo dla funkcji trygonometrycznej

	Średnia wartość całki	l.podp	Błąd
0	0.634379	1	0.577622
1	0.652452	2	0.482485
2	0.737774	5	0.547541
3	0.78407	10	0.505059
4	0.742246	25	0.521307
5	0.763304	50	0.492513
6	0.750657	100	0.50375
7	0.754001	250	0.507398
8	0.7491	500	0.516027
9	0.754272	1000	0.503948
10	0.748031	2500	0.50554
11	0.751496	5000	0.5045
12	0.750644	10000	0.504339

**Funkcja mieszana** Tabela (4.20) przedstawia wyniki uzyskanie z działania metody Monte Carlo dla funkcji podcałkowej (4.25).

Tabela 4.20: Dane z iteracji w metodzie Monte Carlo

	Średnia wartość całki	l.punktów	Błąd
0	-0.212457	2	0.561218
1	-1.126922	5	-0.475718
2	0.476199	10	1.032861
3	-0.507484	25	-0.753507
4	-0.488903	50	-0.189389
5	-0.184741	100	-0.396897
6	-0.641896	250	-0.147126
7	0.001082	500	-0.014857
8	-0.622579	1000	-0.025403
9	-0.411349	2500	0.023891
10	-0.353893	5000	-0.00468
11	-0.393726	10000	-0.078784

## 4.2. Brak Metoda Adaptacyjna Trapezów

## 5. Podsumowanie i wnioski końcowe

1 ÷ 3 stron merytorycznie podsumowanie najważniejszych elementów pracy oraz wnioski wynikające z osiągniętego celu pracy. Proponowane zalecenia i modyfikacje oraz rozwiązania będące wynikiem realizowanej pracy.

Ostatni akapit podsumowania musi zawierać wykaz własnej pracy dyplomanta i zaczynać się od sformułowania: „Autor za własny wkład pracy uważa: ...”.

Tabela 4.4: Dane z iteracji w metodzie trapezów dla funkcji wielomianowej

	Wartość całki	l.podp	Błąd
0	31.315200	1	17.390900
1	18.579200	2	4.654900
2	16.018410	3	2.094110
3	15.107200	4	1.182900
4	14.682819	5	0.758519
5	14.451595	6	0.527295
6	14.311938	7	0.387638
7	14.221200	8	0.296900
8	14.158947	9	0.234647
9	14.114396	10	0.190096
10	14.081422	11	0.157122
11	14.056336	12	0.132036
12	14.036809	13	0.112509
13	14.021312	14	0.097012
14	14.008809	15	0.084509
15	13.998575	16	0.074275
16	13.990092	17	0.065792
17	13.982984	18	0.058684
18	13.976967	19	0.052667
19	13.971830	20	0.047530
20	13.967409	21	0.043109
21	13.963576	22	0.039276
22	13.960233	23	0.035933
23	13.957299	24	0.032999
24	13.954709	25	0.030409
25	13.952413	26	0.028113
26	13.950367	27	0.026067
27	13.948536	28	0.024236
28	13.946891	29	0.022591
29	13.945408	30	0.021108
30	13.944066	31	0.019766
31	13.942848	32	0.018548
32	13.941739	33	0.017439
33	13.940727	34	0.016427

## Załączniki

Według potrzeb zawarte i uporządkowane uzupełnienie pracy o dowolny materiał źródłowy (wydruk programu komputerowego, dokumentacja konstrukcyjno-technologiczna, konstrukcja modelu – makiety – urządzenia, instrukcja obsługi urządzenia lub stanowiska laboratoryjnego, zestawienie wyników pomiarów i obliczeń, informacyjne materiały katalogowe itp.).

## Literatura

- [1] <https://en.wikipedia.org/wiki/Newton>
- [2] Willie Aboumrad, CME 108/MATH 114 Introduction to Scientific Computing
- [3] <https://www.cs.mcgill.ca/>
- [4] Abramovitz, Milton; I. Stegun (1964). Handbook of mathematical functions. National Bureau of Standards. ISBN 0-486-61272-4.
- [5] Numerical Methods for Engineers, 6th Ed
- [6] <https://www.mimuw.edu.pl/~leszekp/dydaktyka/MO19L-g/adaptn.pdf>
- [7] <https://www.math.usm.edu/lambers/mat460/fall09/lecture30.pdf>
- [8] <https://home.agh.edu.pl/~zak/downloads/wyk5.pdf>
- [9] <http://www.algorytm.org/procedury-numeryczne/calowanie-numeryczne-metoda-monte-carlo-ii.html>
- [10] <http://www.algorytm.org/procedury-numeryczne/calowanie-numeryczne-metoda-monte-carlo-i.html>
- [11] <http://weii.portal.prz.edu.pl/pl/materialy-do-pobrania>. Dostęp 5.01.2015.
- [12] Jakubczyk T., Klette A.: Pomiary w akustyce. WNT, Warszawa 1997.
- [13] Barski S.: Modele transmitancji. Elektronika praktyczna, nr 7/2011, str. 15-18.
- [14] Czujnik S200. Dokumentacja techniczno-ruchowa. Lumel, Zielona Góra, 2001.
- [15] Pawluk K.: Jak pisać teksty techniczne poprawnie, Wiadomości Elektrotechniczne, Nr 12, 2001, str. 513-515.

**STRESZCZENIE PRACY DYPLOMOWEJ INŻYNIERSKIEJ**

**ANALIZA I INTERPRETACJA WYBRANYCH METOD  
CAŁKOWANIA NUMERYCZNEGO**

Autor: Kamil Madej, nr albumu: 161876

Opiekun: (dr. inż) Mariusz Borkowski (prof. PRz)

Słowa kluczowe: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po polsku

**BSC THESIS ABSTRACT**

**TEMAT PRACY PO ANGIELSKU**

Author: Kamil Madej, nr albumu: 161876

Supervisor: (academic degree) Imię i nazwisko opiekuna

Key words: (max. 5 słów kluczowych w 2 wierszach, oddzielanych przecinkami)

Treść streszczenia po angielsku