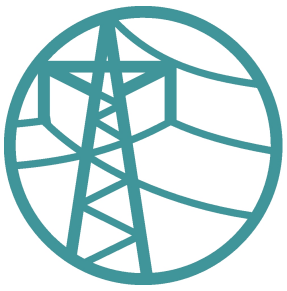


WYDZIAŁ
ELEKTROTECHNIKI
I INFORMATYKI
POLITECHNIKI RZESZOWSKIEJ

Kamil Madej

Analiza i interpretacja wybranych metod całkowania numerycznego

Praca dyplomowa inżynierska

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

Rzeszów, 2022

Spis treści

1. Wstęp/wprowadzenie

Pomimo tego, że dokładny wynik uzyskiwany przez analityczne obliczenie całki jest porządany, rzeczywiste problemy z jakimi mierzą się inżynierowie rzadko pozwalają na takie rozwiązanie. W związku z tym, całkowanie numeryczne jest niezastąpionym narzędziem w ich pracy. Całkowanie numeryczne można spotkać w praktycznie każdej dziedzinie naukowej, lecz podejście to jest szczególnie powszechne podczas przetwarzania losowych danych fizycznych, które nie są zgodne z żadną ciągłą, deterministyczną funkcją.

Podstawowym problemem całkowania numerycznego jest obliczenie przybliżonego rozwiązania całki oznaczonej z pewnym stopniem dokładności. Całka z pewnej funkcji $f(x)$ na przedziale $[a,b]$ może być przedstawiona jako pole powierzchni pomiędzy krzywą a osią odciętych. Wzór na całkę oznaczoną przedstawia wzór

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

Z twierdzenia Newtona- Leibniza wiemy, że funkcje ciągłe mają dokładne rozwiązanie analityczne. Oznaczają funkcję pierwotną jako $F(x)$, wiemy że całkę oznaczoną liczymy ze wzoru.

$$\int_a^b f(x)dx = F(b) - F(a) \tag{1.2}$$

gdzie $F(x) = \int f(x)dx$ to dowonla funkcja pierwotna funkcji f na tym przedziale.

W praktyce znalezienie funkcji pierwotnej jest niezwykle trudne lub niemożliwe, przez co konieczne jest zastosowanie całkowania numerycznego.
Jednym ze sposobów na uzyskanie numerycznego przybliżenia całki oznaczonej jest zastosowanie sumy Riemanna. Zasadniczo, można ją zdefiniować na wiele różnych sposobów. Najprostszym z nich jest metoda prostokątów, przybliżająca obszar pod krzywą za pomocą prostokątów. Bardziej zaawansowaną formą sumy Riemanna są metoda trapezów, metoda Simpsona oraz metody Gaussa. Dokładność przybliżenia wymienionych metod można poprawić, dzieląc przedział całkowania na większą liczbę figur o mniejszej szerokości.

Innym numerycznym sposobem na obliczenie całki jest metoda Monte Carlo, opracowana i pierwszy raz zastosowana przez Stanisława Ulama. Obliczanie całki tą metodą polega na losowym próbkowaniu funkcji na zadanym przedziale, a nastepnie obliczenie jaki procent wylosowanych punktów znajduje pod wykresem funkcji[1].

Cel całkowania numerycznego

Istnieje wiele przypadków gdy całkowania numeryczne jest lepszym rozwiązaniem niż całkowanie analityczne lub jedynym rozwiązaniem:

- gdy całka z funkcji $f(x)$ znana jest jedynie dla pojedyńczych punktów, na przykład w przypadku stabelaryzowanych danych

- gdy wzór całki jest znany, ale znalezienie funkcji pierwotnej jest bardzo skomplikowane lub nie możliwe do zrobienia. Przykładem takiej funkcji jest $f(x) = exp(-x^2)$

- gdy znalezienie funkcji pierwotnej jest możliwe analitycznie, ale obliczenie numeryczne jest prostsze. Taki przypadek może wysąpić, gdy funkcja pierwotna jest dana jako nieskończony szereg lub iloczyn[2].

2. Metody całkowania numerycznego

2.1. Metody Netwona-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[3].

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

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} w_i f(x_i) \tag{2.3}$$

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[4]:

$$\int_a^b f(x)dx \approx \int_a^b p(x)dx = \int_a^b \sum_{i=0}^{n} f_i L_{n,j}(x)dx = \sum_{i=0}^{n} (\int_a^b L_{n,j}(x)dx) f_i = \sum_{i=0}^{n} w_i f_i \tag{2.4}$$

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 0, czyli funkcji stałej. Dla pojedynczego podprzedziału metoda prostokątów wygląda następująco:

$$\int_a^b f(x) = I = f(a) * h \tag{2.5}$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = h * \sum_{i=0}^{n-1} f(x_i) \tag{2.6}$$

gdzie h jest szerokością pojedyńczego podprzedziału i równa się $h = \frac{b-a}{n}$ [5]. Listing 1 pokazuje przykładowy kod w języku Python implementujący metodą trapezów.

```
import numpy as np

def f_rectI(f,a,b,n):
    h = (b - a) / n

    X = np.linspace(a + 0.5 * h, b - 0.5 * h, num=n)

    Y = []
    for i in range(0, n):
        Y.append(f(X[i]))

    Y = np.array(Y)

    I = h * Y
    I = np.sum(I)
    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 1, czyli funkcji liniowej.

Dla pojedynczego podprzedziału metoda trapezów wygląda następująco:

$$\int_a^b f(x) = \frac{1}{2} * h[f(a) + f(b)] \tag{2.7}$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = \frac{1}{2} \sum_{i=0}^{n} [f(x_i + f(x_{i+1}))] \tag{2.8}$$

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

```
import numpy as np

def f_trapI(f, a, b, n):
    h = (b - a) / n

    X = np.linspace(a, b, num=n + 1)

    Y = []
    for i in range(0, n + 1):
        Y.append(f(X[i]))

    Y = np.array(Y)

    I = []
    for i in range(0, n + 1):
        if i == 0 or i == n:
            I.append(h * Y[i] / 2)
        else:
            I.append(h * Y[i])

    I = np.sum(I)
    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 2, czyli paraboli.

Dla pojedyńczego podprzedziału metoda trapezów wygląda następująco:

$$\int_a^b f(x) = \frac{1}{3} * h[f(a) + 4f(a+h) + f(a+2h)] \tag{2.9}$$

dla n podprzedziałów metoda prostokątów przybiera formę:

$$\int_a^b f(x) = \frac{1}{2} h[f(a) + f(b)] + \sum_{i=1,3,5}^{n-1} 4f(x_i) + \sum_{i=2,4,6}^{n-2} f(x_i) \tag{2.10}$$

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

```
def f_simp2I(f,a,b,n):
    h = (b-a)/n

    I = f(a) + f(b)

    for i in range(1,n):
        if i%2==1:
            I += 4 * f(a + i*h)
        else:
            I += 2 * f(a +i*h)

    I = h/3 * I
    return I
```

Listing 3: Kod w języku python implementujący metodę Simpsona

2.2. Kwadratura Gaussa Legendre'a

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

$$\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} w_i f(x_i) \tag{2.11}$$

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

Kwadratura Gaussa-Legendre'a jest formą kwadratury Gaussa, która pozwala na efektywne przybliżanie funkcji ze znanym zachowaniem asymptotycznym na brzegach przedziału całkowania. Kwadratura Gaussa jest szczególnie zalecana, jeśli całka jest holomorficzna w pewnym sąsiedztwie przedziału całkowania.

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

$$x_i \approx cos(\pi \frac{\frac{1}{2}+i}{N}) \tag{2.12}$$

Wagi w_i można wyrazić wzorem:

$$w_i = \frac{2(1-x_i^2)}{[nP_{n-1}(x_i)]^2} = \frac{2}{[P_n^1(x_i)]^2} \tag{2.13}$$

[6]

W równaniu (2.11) można zauważyć, że całkujemy funkcję w granicy od -1 do 1. Zabieg ten został poczyniony by uprościć obliczenia matematyczne i w sposób jak najbardziej możliwy uogólnić metodę.

Do sprowadzenia dowolnej granicy całkowania do granicy -1 do 1, posłuży dodatkowa zmienna x_d, która jest w sposób liniowy związana ze zmienną x:

$$x = a_0 + a_1(-1) \tag{2.14}$$

Jeśli dolna granicą $x = a$, odpowiada $x_d = -1$, to wartości te można podstawić do równania (2.14)

$$a = a_0 + a1(-1) \tag{2.15}$$

Podobnie postępujemy z górną granicą, $x = b$, odpowiada $x_d = 1$, co daje

$$b = a_0 + a_1(1) \tag{2.16}$$

Równania (2.15) i (2.16), można rozwiązać jednocześnie co daje

$$a_0 = \frac{b+a}{2} \tag{2.17}$$

oraz

$$a_1 = \frac{b-a}{2} \tag{2.18}$$

po podstawieniu do wzoru (2.14) otrzymujemy

$$x = \frac{(b+a)+(b-a)x_d}{2} \tag{2.19}$$

Po zróżniczkowaniu równania (2.19) orzymujemy

$$dx = \frac{b-a}{2} dx_d \tag{2.20}$$

[7]

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 Gaussa Legendre'a

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[8].

Metody adaptacyjne polegają na wykorzystaniu tradycyjnych metod całkowania takich jak: Metoda Simpsona, trapezów do obliczenia całki na przedziale [a,b] z zadaną dokładnością ξ. 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[9].

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

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

$$I_j(f) = \int_{a_j}^{b_j} f(x)dx \tag{2.21}$$

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 zadaną dokładnością. Do ustalenia dokładności, obliczamy całkę na przedziale $[a_j, b_j]$, by uzyskać przybliżenie I_1, 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 z zadaną dokładnością[10].

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

```
def f_adapt(f,a,b,tol):
    m = (a+b)/2.0
    P1 = f_trapI(f,a,m)
    P2 = f_trapI(f,m,b)

    if abs(P1 - P2 ) < 3 * tol:
        return  P2
    else:
        return f_adapt(f,a,m,tol) + f_adapt(f,m,b,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. Wspomniane metody używają równo lub w sposób przemyślany rozmieszczonych węzłów, natomiast metoda Monte Carlo polega na obliczaniu pola powierzchni pod krzywą używając losowo rozmieszczonych punktów w obrębie granic całkowania.

2.4.1. Crude Monte Carlo

Podstawowa metoda Mone Carlo, zwana też Crude Monte Carlo polega na wylosowaniu n punktów w obrębie przedziału całkowania i na podstawie tych danych obliczenie średniej wartości funkcji[11].

$$f_{sr} = \frac{f(x_1) + f(x_2) + ... + f(x_n)}{n} \tag{2.22}$$

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

$$I = f_{sr} * |b - a| \tag{2.23}$$

Listing 6 pokazuje przykładowy kod w języku Python implementujący metodę Crude Monte Carlo.

```
  import numpy as np
def f_crudeMonteC(f,a,b,n):

    h = (b - a) / n

    X = (b-a)*np.random.random_sample(n)+a
    Y = []

    for i in range(0, n):
        Y.append(f(X[i]))

    Y = np.array(Y)

    I = h*Y
    I = np.sum(I)
    return I
```

Listing 6: Kod w języku python implementujący metodę Crude Monte Carlo

2.4.2. Monte Carlo

Inna wersją tego sposobu całkowania 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 $< f(a), f(b) >$. Metoda ta daje lepsze przybiżenie całki, lecz musimy znać minimalną i maksymalną wartość jaką przyjmuje funkcja na danym przedziale, w przeciwnym wypadku wartości te są ustalane na postawie wylosowanych punktów, co może powodować duży błąd w obliczniu całki. Po wylosowaniu n punktów wartość całki wyrażana jest jako:

$$I = P * \frac{c}{n} \tag{2.24}$$

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[**?**]

Listing 7 pokazuje przykładowy kod w języku Python implementujący metodę Monte Carlo.

```
import numpy as np
def f_MonteC(f,a,b,n,min,max):
    maxY = max
    minY = min

    X = np.random.uniform(a,b,n)

    Y = []
    for i in range(0, n):
        Y.append(f(X[i]))

    Yi = np.random.uniform(minY,maxY,n)

    k = 0
    for i in range(0,n):
        if (Yi[i] > 0) and (Yi[i] <= Y[i]):
            k += 1
        elif (Yi[i] < 0) and (Yi[i] >= Y[i]):
            k -= 1
    P = np.abs(b-a) * np.abs(maxY - minY)
    I = k/n * P
    return I
```

Listing 7: Kod w języku python implementujący metodę Crude Monte Carlo

3. Użyte narzędzia

3.1. Język Python

Python jest wysokopoziomowym,interpertowanym,zorientowanym obiektowo językiem programowania, który w ostatnich latach zdobył wielką popularność i stał się głównym językiem wykorzystywanym w data science[13]. Pozwala on na przeprowadzanie złożonych obliczeń statystycznych, tworzenia wizualizacji danych, budowania

algorytmów uczenia maszynowego, manipulowanie i analizowanie danych.
Przy pomocy bibliotek takich jak matplotlib python daje olbrzymie możliwości wizualizacji danych przy pomocy całego szeregu dostępnych wykresów. Biblioteki takie jak TensorFlow i Keras, pozwalają na szybkie i efektowne tworzenie programów do analizy danych i uczenia maszynowego[14].

3.2. Użyte biblioteki

NumPy to jedna z najpopularniejszych bibliotek Python'a, skupiająca się na obliczeniach naukowych. Dostarcza wielowymiarowe obiekty tablicowe, które są lepsze pod względem wydajności i szybkości od standardowych list występujących w Pythonie[15]. Tablice te są homogeniczne, co oznacza, że mogą przechowywać tylko jeden typ danych na raz.Ta restrykcja umożliwia oddelegowanie wykonywania matematycznych operacji na tablicy do zoptymalizowanego kodu w języku C, który jest znacznie szybszy od interpretera Pythona[16].

Matplotlib jest biblioteką dostarczającą szeroki zakres funkcji do wizualizacji danych. Biblioteka ta jest rozszerzeniem biblioteki NumPy. Jako, że jest darmowa, oraz zawiera proceduralny interfejs "Pylab", który został zaprojektowany tak, by jak najbardziej przypominał MATLAB stanowi ona jego realną alternatywę.

3.3. Środowisko Jupyter Notebook

Jupyter notebook to interaktywna aplikacja internetowa pozwalająca na tworzenie i udostępnianie dokumentów obliczeniowych.[17]. Dokumenty te są bazowanymi na przeglądarce interaktywnymi programami zwanymi REPL(read-eval-print loop). Programy tego typu pobierają od użytkownika dane, wykonują ję, a następnie zwracają użytkownikowi wynik[18]. W strukturę tworzonego dokumentu wchodzą komórki zawierające kod programu,tekst,wykresy lub media interaktwyne(animacje,pilki wideo,pliki audio). Pod warstwą wizualną Jupyter Notebook znajduje się dokument JSON, zwykle kończący się rozszerzeniem ".ipynb". Dokumenty tworzone w środowisku Jupyter mogą zostać przekonwertowane na wiele popularnych formatów(HTML, LaTeX,PDF, Markdwon) oraz być udostępniane innym użytkownikom przy pomocy emaila, Dropbox'a. GitHub'a lub za pomocą Jupyter Notebook Viewer[19].
Paltforma Jupyter może połączyć się z wieloma kernelami, które pozwalają na wy-

konywanie kodu w ponad 100 jezykw programowania, wliczając w to obecnie najpopularniejsze takie jak Python, Java, R, Matlab, Octave i wiele innych[20]. Mnogość dostępnych języków pozwala osobą tworzącym dokumenty na wybranie odpowiedniego języka dla konkretnych zastosowań. Niezależnie od wybranego języka inteffejs użytkownika pozostaje taki sam.

4. Instrukcja laporatoryjna

Jednym z elementów niniejszej pracy inżynierskiej było stworzenie laboratorium dla przedmiotu "Metody Numeryczne". Laboratorium zostało stworzone przy pomocy opisanej wyżej platformy Jupyter Notebook oraz zaimplementowane zostały w nim opisane w poprzednim rozdziale metod całkowania.

Całość dokumentu wchodzącego w skład labratorium można podzielić na 4 części.

Na samym początku notatnika umieszczony został spis treści widoczny na rysunku 4.1 , w którym zobaczyć można całą strukturę dokumentu.

Rysunek 4.1: Spis treść z instrukcji laboratoryjnej

Kolejnym segmentem dokumentu jest krótki wstęp teoretyczny, w którym opisane zostały zagadnienia takie jak: definicja całki nieoznaczonej, definicja całki ozna-

czonej, twierdzenie Newtona Leibniza. Rysunek 4.2 pokazuje komórkę ze wstępem teoretycznym na temat całkowania

Część teoretyczna

Całka jest jednym z najważniejszych pojęć współczesnej analizy matematycznej. Stosuje się ją w matematyce, fizyce, technice i wielu innych dziedzinach. W matematyce badaniem własności i obliczeniem wartości całek zajmuje się dział zwany rachunkiem całkowym.

Całka nieoznaczona

Całką nieoznaczoną nazywamy odwrotność pochodnej danej funkcji.

$$F'(x) = F(x) \tag{1}$$

Funkcję $F(x)$ nazywamy **funkcją pierwotną** danej funkcji $f(x)$. Całkę nieoznaczoną opisujemy:

$$\int f(x)\,dx = F(x) + C \tag{2}$$

Całka oznaczona

Całka oznaczona to pole powierzchni między wykresem funkcji $f(x)$ w pewnym przedziale $< a, b >$. Całka oznaczona funkcji $f(x)$ w przedziale $< a, b >$ oznaczana jest symbolem:

$$\int_a^b f(x) \tag{3}$$

Jeśli znamy całkę nieoznaczoną $F(x)$ funkcji podcałkowej $f(x)$, to całkę oznaczoną funkcji $f(x)$ w przedziale $< a, b >$ możemy obliczyć z twierdzenia Newtona Leibniza:

$$\int_a^b f(x)\,dx = F(b) - F(a) \tag{4}$$

Rysunek 4.2: Rozdział teoretyczny o całkowaniu analitycznym

Następnie umieszczona została seria ćwiczeń, polegających na uzupełnieniu kodu funkcji całkującej poprzedzona krótkim przybliżeniem teoretycznym danej metody całkowania oraz algorytmem, na którym bazowane są wspomniane metody. W opisie teoretycznym postawiono na zwięzłą formę, by student wykonujący tę instrukcję mógł sobie szybko przypomieć najważniejsze informacje o danych metodach całkowania. Algorytm jak i zaimplementowane funkcje zostały napisane w przystępny sposób, krok po kroku, więc nie powiniem sprawić trudności osobą wykonującym tę laboratorium. Rysunek 4.3 pokazuję przykładowe ćwiczenie z dokumenty Jupyter Notebook.

Ćwiczenie 2

Korzystając z Algorytmu 1 dokonaj implementacji kwadratury Newtona-Cotesa - metoda prostokątów. Znajduje się ona w poniższej komórce. Jej wywołanie wygląda następująco:

```
I = f_rectI(f,a,b,n)
```

gdzie jej parametry to:

- f - funkcja, dla której szukamy całki
- a, b - przedział całkowania
- n - ilość podprzedziałów, na które dzielimy odcinek <a,b>

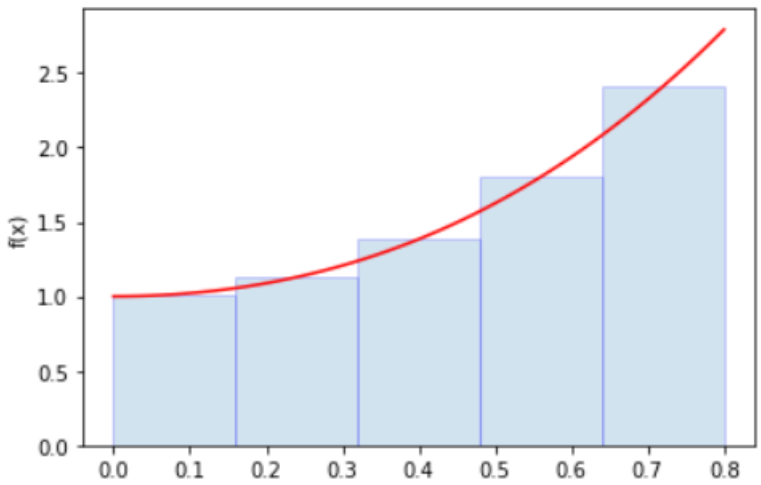
a zwracane wartości to:

- I - przybliżenie całki

```
import numpy as np

def f_rectI(f,a,b,n):
    """
        :param fun: funkcja
        :param xa: dolny przedział całkowania
        :param xb: górny przedział całkowania
        :param n: ilość podprzedziałów, na które dzielimy odcinek <a,b>
        :return: przybliżona wartość całki
        """
    # oblicz szerokość pojedynczego przedziału
    h =

    # zwracóc n równo rozmieszczonych punktów z przedziału <a+(1/2*h),b-(1/2*h)>
    X = np.linspace()

    # inicjalizacja listy Y
    Y = []
```

Rysunek 4.3: Przykład ćwiczenia z instrukcji laboratoryjnej

Ostatnim elementem laboratorium pomagającym w lepszym wyobrażeniu metod całkowania są funkcję, które przedstawiają wizualizacje danej metody całkowania. Funkcje te zostały zaimplementowane w oparciu o metody dostarczone wraz z biblioteką matplotlib. Przykład funkcji przedstawiającej wizualizację metody prostokątów jest widoczny na rysunku 4.4.



Rysunek 4.4: Funkcja wizualizująca metodę prostokątów

5. Bibliotek do całkowania numerycznego

Następne podrozdziały przedstawiają wyniki i omówienie działania zaimplementowanych przeze mnie metod całkowania dla równan (5.25) (5.26) (5.27).

Dla metod dzielących przedział całkowania na mniejsze części, skuteczność wyrażana będzie jako ilośc wspomnianych podprzedziałów potrzebnych do obliczenia całki z dokładnością 0.01. Skuteczność metod Monte Carlo liczona będzie ilością losowych punktów potrzebnych do uzyskania zadowalającej dokładności. W adaptacyjnej metodzie trapezów miarą skuteczności będzie ilość podprzedziałów, potrzebna do uzyskania zadanej dokładności przybliżenia funkcji.

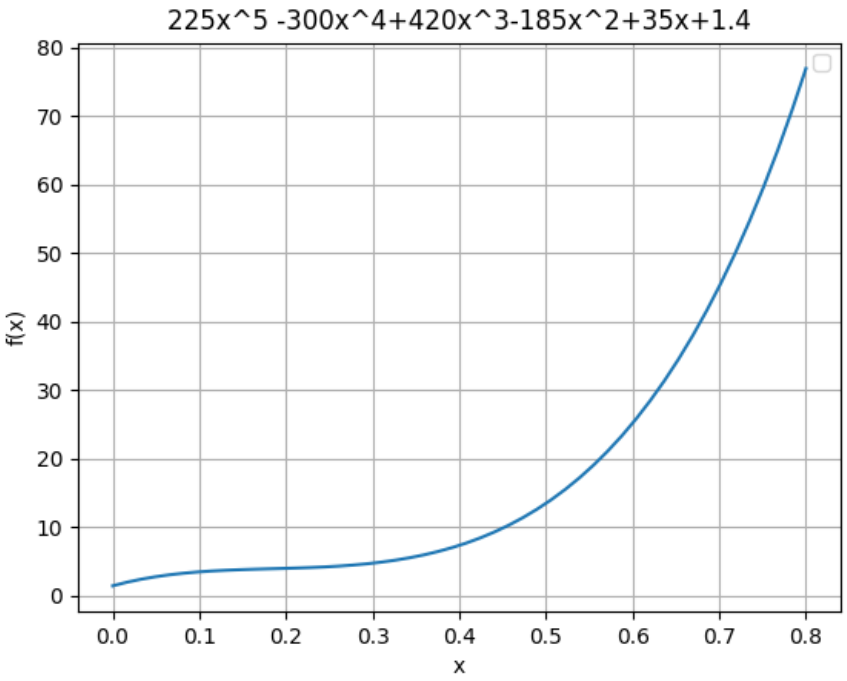
Rezultaty z działania wszystkich metod całkowania zostaną zbadane na podstawie funkcji wielomianowej (5.25) widocznej na rysunku 5.5 oraz trygonometrycznej (5.26), którą przedstawia rysunek 5.6. Dodatkowo funkcja (5.27) widoczna na rysunku 5.7, która cechuje się dużą zmiennością przy prawym krańcu przedziału całkowania zostanie użyta do zbadania wyższości metody adaptacyjnej trapezów nad zwykłą metodą trapezów.

Wszystkie rezlutaty z działania funkcji zostaną przedstawione w tabelach. Dla zmniejszenia wielkości tabel, do opisu kolumn zostały użyte nazwy skrócone.
Wykaz nazw używanych do opisu kolumn.

I - wartość całki
E - błąd metody
n - liczba podprzedziałów, na które została podzielona funkcja lub w przypadku metod Monte Carlo liczba użytych losowych punktów
Tol - tolerancja z jaką aproksymowana była funkcja w metodzie adaptacyjnej.
AVG - średnia wartość całki dla 100 iteracji w metodach Monte Carlo
maxValue - maksymalna wartość uzyskana przy całkowaniu Monte Carlo dla zadanej ilości punktów
minValue - minimalna wartość uzyskana przy całkowaniu Monte Carlo dla zadanej ilości punktów
Metoda - nazwa metody całkującej

Funkcja wielomianoa

$$f(x) = 225x^5 - 300x^4 + 420x^3 - 185x^2 + 35x + 1.4 \tag{5.25}$$



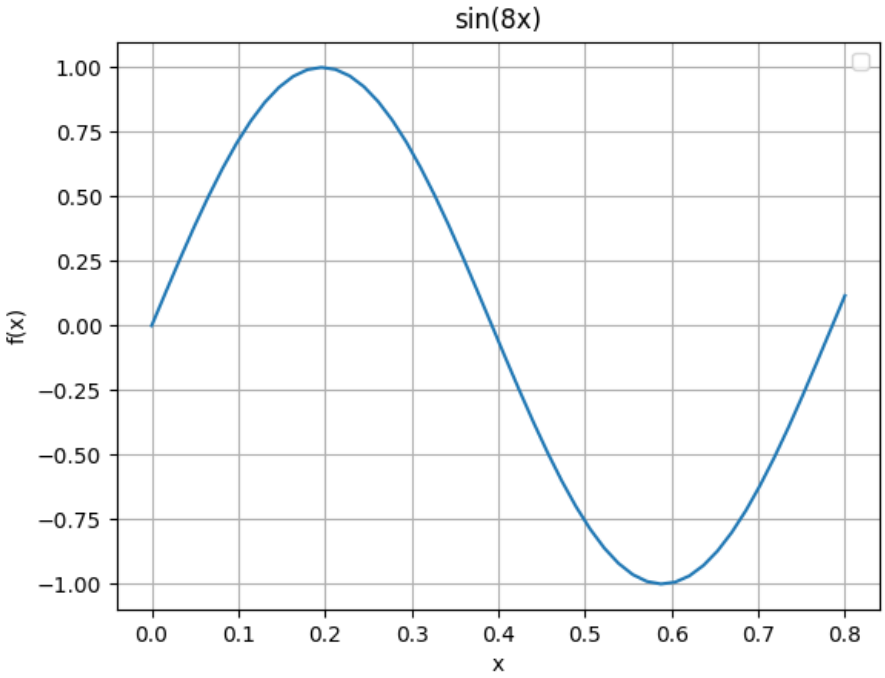
Rysunek 5.5: Wykres funkcji zdefiniowanej wzorem(5.25)

Funkcja trygonometryczna

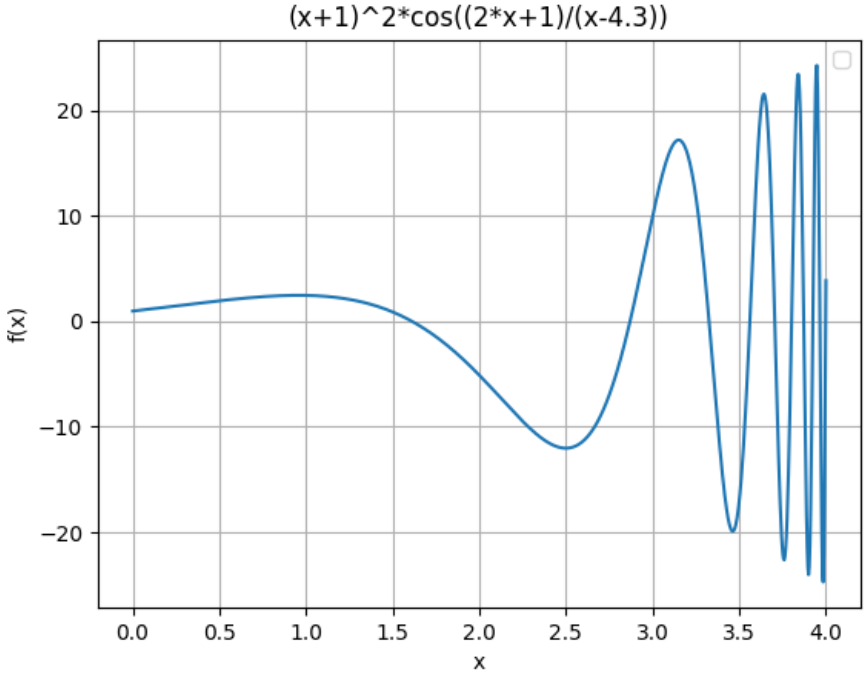
$$f(x) = sin(8x) \tag{5.26}$$

Funkcja oscylacyjna

$$f(x) = (x+1)^2 * cos((2*x+1)/(x-4.3)) \tag{5.27}$$



Rysunek 5.6: Wykres funkcji zdefiniowanej wzorem(5.26)



Rysunek 5.7: Wykres funkcji zdefiniowanej wzorem(5.27)

5.1. Przykład użycia biblioteki

5.1.1. Metoda prostokątów

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

```
def f_rectI(f,a,b,n):
        :param fun: funkcja
          :param a: dolny przedzial calkowania
          :param b: gorny przedzial calkowania
          :param n: ilosc podprzedzialow, na ktore dzielimy
   odcinek <a,b>
          :return: przyblizona wartosc calki
    """
```

Listing 8: Kod w języku python implementujący metodę prostokątów

Funkcja wielomianowa

Tabela 5.1 przedstawia dane otrzymane w wyniku działania implementacji metody prostokątów dla funkcji (5.25) 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 (5.25).
Do osiągnięcia założonej dokładności metoda prostokątów potrzebowała 31 podprzedziałów.

Tabela 5.1: Dane z iteracji w metodzie prostokątów dla funkcji wielomianowej

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

Tabela 5.2 przedstawia dane zwrócone z działania metody prostokątów dla funkcji (5.26) 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 funkcje nijak ma się do rzeczywistego kształtu funkcji. Wraz ze zwiększającą się ilością podprzedziałów dokładność przybliżenia znacząco rośnie, aż dla 16 podprzedziałów przyjmuję zakładaną dokładność.

Funkcja trygonometryczna

Tabela 5.2: Dane z iteracji w metodzie prostokątów dla funkcji trygonometrycznej

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

5.1.2. Metoda trapezów

Funkcja implementująca metodę trapezów, przyjmuje dokładnie takie 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.

```
def f_trapI(f, a, b, n):
    """
        :param fun: funkcja
          :param a: dolny przedzial calkowania
          :param b: gorny przedzial calkowania
          :param n: ilosc podprzedzialow, na ktore dzielimy
   odcinek <a,b>
          :return: przyblizona wartosc calki
    """
```

Listing 9: Kod w języku python implementujący metodę trapezów

Funkcja wielomianowa

Tabela 5.3 przedstawia wyniki uzyskane z działania metody trapezów na funkcji (5.25).
Dla pojedyńczego podprzedziału błąd funkci jest bardzo duży i wynika, z tego, że wartość funkcji dla końcowych argumentów z przedziału gwałtownie rośnie, przez co prosta łącząca krańce przedziału całkowanai 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 5.4 przedstawia dane uzyskane z działania metody trapezów na funkcji podcałkowej (5.26). Największy błąd dla tej funkcji pojawia się, gdy ilość podprzedziałów wynosi 3, dzieje się tak dlatego, że wartości funkcji dla granic środkowego podprzedziału przymuja bardzo niskie wartości, przez co trapez przybliżający funkcję na tym przedziale pomija całkowice całą dodatnią cześć sinusiody.

Tabela 5.3: Dane z iteracji w metodzie trapezów dla funkcji wielomianowej

	I	n	E
31.315200	1	17.390900	
18.579200	2	4.654900	
16.018410	3	2.094110	
15.107200	4	1.182900	
14.682819	5	0.758519	
*	*	*	
13.936159	40	0.011859	
13.935586	41	0.011286	
13.935054	42	0.010754	
13.934558	43	0.010258	
13.934095	44	0.009795	

Metoda trapezów przyjmuję wartość z zadaną dokładnością przy ilości podprzedziałów równej 23.

Tabela 5.4: Dane z iteracji w metodzie trapezów dla funkcji trygonometrycznej

I	n	E
-0.287903	1	0.532611
0.845407	2	0.600699
-1.269118	3	1.513825
-0.223984	4	0.468692
-0.011437	5	0.256145
*	*	*
0.230073	19	0.014635
0.231515	20	0.013192
0.232754	21	0.011954
0.233825	22	0.010882
0.234758	23	0.009949

5.1.3. Metoda Simpsona

Funkcja implementująca metodę Simpsona, tak jak poprzednie metody Newtona-Cotesa przyjmuje 4 parametry. Pierwszym jest całkowana funkcja, drugim dolny przedział całkowania, trzecim górny przedział całkowania, a ostatnim jest ilośc podprzedziałów, na które dzielimy przedział całkowania.

```
def f_simp2I(f,a,b,n):
    """
        :param fun: funkcja
          :param a: dolny przedzial calkowania
          :param b: gorny przedzial calkowania
          :param n: ilosc podprzedzialow, na ktore dzielimy
   odcinek <a,b>
          :return: przyblizona wartosc calki
    """
```

Listing 10: Kod w języku python implementujący metodę Simpsona

Funkcja wielomianowa

Tabela 5.5 przedstawia wyniki uzyskane z działania metody Simpsona dla funkcii podcałkowej (5.25). Metoda Simpsona potrzebowała 6 podprzedziałów by osiągnąć zadaną dokładność.

Tabela 5.5: Dane z iteracji w metodzie Simpsona dla funkci wielomianowej

Ii	n	E
14.333867	2	0.409567
13.949867	4	0.025567
13.929323	6	0.005023

Funkcja trygonometryczna

Tabela 5.6 przedstawia wyniki uzyskane z działania metody Simpsona dla funkcii podcałkowej (5.26). Metoda Simpsona potrzebowała 12 podprzedziałów by osiągnąć zadaną dokładność.

Tabela 5.6: Dane z iteracji w metodzie Simpsona dla funkci trygonometrycznej

I	n	E
1.223177	2	0.978469
-0.580448	4	0.825156
0.528331	6	0.283623
0.284161	8	0.039454
0.257320	10	0.012613
0.250121	12	0.005413

5.1.4. Metoda Gaussa Legendre'a

Funkcja implementująca metodę Gaussa Legendre'a tak jak wszystkie funkcję bazujące na przybliżaniu krzywej wielomianami, przyjmuje 4 parametry. Pierwszym z nich jest całkowana funkcja, drugim dolny, a trzecim górny przedział całkowania, jako 4 parametr funkcja przyjmuję ilosć podprzedziałów na którę podzieli przedział całkowania. Wartością zwracaną przez funkcję jest przybliżenie całki.

```
1 def f_gauss_legrande(f,a,b,n):
2     """
3           :param fun: funkcja
4           :param a: dolny przedzial calkowania
5           :param b: gorny przedzial calkowania
6           :param n: ilosc podprzedzialow, na ktore dzielimy
    odcinek <a,b>
7           :return: przyblizona wartosc calki
8     """
```

Listing 11: Kod w języku python implementujący metodę Gaussa Legendre'a

Funkcja wielomianowa

Tabela 5.7 przedstawia wyniki uzyskane z działania metody Gaussa Legendre'a dla funkcii podcałkowej (5.25). Funkcja , dla której liczona jest całka jest wielomianem stopnia 5, a metoda Gaussa Legendre'a daje wynik dokładny dla wielomianów stopnia $2n-1$, dlatego do uzyskania zadanej dokładności metoda potrzebowała 3 węzłów

interpolacyjnych. Uzyskany błąd wynika z błedu zaokrągleń.

Tabela 5.7: Dane z iteracji w metodzie Gauss'a Legendre'a dla funkcji wielomianowej

I	n	E
5.843200	1	8.081100
13.651200	2	0.273100
13.924267	3	0.000033

Funkcja trygonometryczna

Tabela 5.8 przedstawia wyniki uzyskane z działania metody Gaussa Legendre'a dla funkcii podcałkowej (5.26). Całkowana funkcja nie jest wielomianem, więc uzyskanie dokładnej wartości przy użyciu metody Gaussa jest niemożliwe, lecz jej kształ sprawia, że daje się ona łatwo przybliżyć, przy użyciu wielomianów, co skutkuje, że do uzyskania zakładanej dokładności kwadratura Gaussa potrzebowała jedynie 7 węzłów interpolacyjnych.

Tabela 5.8: Dane z iteracji w metodzie Gauss'a Legendre'a dla funkcji trygonometrycznej

I	n	E
1.978716	1	1.734009
-0.184912	2	0.429619
1.974615	3	1.729907
-0.611561	4	0.856269
0.456455	5	0.211748
0.212634	6	0.032073
0.248022	7	0.003315

5.1.5. Metoda adaptacyjna trapezów

Metoda adaptacyjna trapezów przyjmuje 4 parametry. Tak jak w poprzednich funkcjach trzema pierwszymi są: całkowana funkcja, dolny przedział całkowania, górny przedział całkowania. Jako 4 parametry funkcja przyjmuję tolerancje, na podstawie

której będzie decydowała, czy dokonywać dalszych podziałów przedziału całkowania, czy nie. Funkcja zwaraca przybliżoną wartość całki.

```
def f_adapt(f,a,b,tol):
    """
        :param fun: funkcja
            :param a: dolny przedzial calkowania
            :param b: gorny przedzial calkowania
            :param tol: tolerancja
            :return: przyblizona wartosc calki
    """
```

Listing 12: Kod w języku python implementujący adaptacyjną metodę trapezów

Funkcja wielomianowa

Tabela 5.9 przedstawia wyniki uzyskanie z działania metody adaptacyjnej trapezów na funkcii podcałkowej (5.25). Pomimo, że bład nie jest mniejszy niż zakładana tolerancja, to obie te wartości ulegają zmniejszeniu wraz ze zmniejszeniem parametru tol. Najdokładniejsze przybliżenie metoda osiąga dla 1000 podprzedziałów, kolejne iteracje dają wynik nieznacznie gorszy. Fakt ten wynikać ze zbyt dużej dokładności, ponieważ analityczny wynik całkowania badanego wielomianu, wynosi 13.9243, a próbowano uzyskać dokładność rzędu 0.00000001.

Tabela 5.9: Dane z iteracji w metodzie Adaptacyjnej Trapezów dla funkcji wielomianowej

I	n	Tol	E
14.08788	8	0.1	0.16358
13.950712	22	0.01	0.026412
13.929132	48	0.001	0.004832
13.925456	98	0.0001	0.001156
513.925456	208	0.00001	0.000265
13.924337	442	0.000001	3.7E-5
13.924281	1000	0.0000001	1.9E-5
13.924269	2208	0.00000001	3.1E-5
13.924267	4812	0.000000001	3.3E-5
13.924267	10346	0.0000000001	3.3E-5

Funkcja trygonometryczna

Tabela 5.10 przedstawia wyniki uzyskanie z działania metody adaptacyjnej trapezów na funkcii podcałkowej (5.26). W tabeli zauważyć można, że wraz ze wzrostem deklarowanej dokładności, błąd maleje. Nie zawsze udaje się uzyskać błąd taki jak przekazana jako argument tolerancja, ale trzeba pamiętać, że to tylko szacunek, a nie gwarancja. Po podziale funkcji na 5571 podprzedziałów udaje się uzyskać wynik dokłady do 6 miejsca po przecinku.

Tabela 5.10: Dane z iteracji w metodzie Adaptacyjnej Trapezów dla funkcji trygonometrycznej

I	n	Tol	E
0.275097	12	0.1	0.03039
0.226516	26	0.01	0.018191
0.243565	54	0.001	0.001143
0.244287	116	0.0001	0.00042
0.244644	234	0.00001	6.4E-5
0.244686	476	0.000001	2.1E-5
0.244702	964	0.0000001	5.0E-6
0.244706	1952	0.00000001	2.0E-7
0.244707	5571	0.000000001	2.0E-7
0.244707	12406	0.0000000001	3.3E-8

5.1.6. Metoda Crude Monte Carlo

Metoda Crude Monte Carlo przyjmuje 4 parametry: całkowaną funkcję, dolny przedział całkowania, górny przedział całkowania oraz liczbę punktów, którymi funkcja będzie próbkowana. Wartością zwracaną przez funkcję jest przybliżenie całki.

```
def f_crudeMonteC(f,a,b,n):
    """
        :param f: funkcja
            :param a: dolny przedzial calkowania
            :param b: gorny przedzial calkowania
            :param n: ilosc losowych punktow
            :return: przyblizona wartosc calki
    """
```

Listing 13: Kod w języku python implementujący metodę Crude Monte Carlo

Funkcja wielomianowa

W tabeli 5.11 przedstawiono wyniki uzyskane z działania metody Crude Monte Carlo dla funkcii podcałkowej (5.25). Otrzymany błąd dla jednego podprzedziału może

wydawać, się zaskakująco mały, lecz badanie dla każdej ilości podprzedziałów było wyknoywane 100 razy. Dla jednego podprzedziału wartości całki może się wahać od 1.2 do 61.5103. Wraz ze zwiększaniem ilości punktów, zakres otrzymywanych wyników ulega zmniejszeniu. W tabeli można również zauważyć, że wartości otrzymane dla 100 jak i 1000 punktów, są bardzo podobne i wynika to z losowości ich rozmieszczenia. Przy 100000 losowych punktów średni otrzymany błąd przybliżenia wynosi 0.01

Tabela 5.11: Dane z iteracji w metodzie Crude Monte Carlo dla funkcji wielomianowej

n	minValue	maxValue	AVG	E
1	1.416807038	60.99560428	11.987235	13.91%
10	2.70581996	26.80458791	13.662372	1.88%
100	10.89037611	19.49690441	13.98955	0.47%
1000	12.37476196	14.94271099	13.941598	0.12%
510000	13.61746918	14.30824118	13.906282	0.13%
100000	13.76707602	14.03293431	13.922587	0.01%

Funkcja trygonometryczna

Tabela 5.12 przedstawia wyniki uzyskane z działania metody Crude Monte Carlo dla funkcii podcałkowej (5.26). Na podstawie pierwszych 3 wierszy tabeli widać, że otrzymywany błąd wraz z dziecięsiokrotnym wzrostem ilości punktów maleje około 4-krotnie. Dalsze 10-krotne zwiększenie ilości użytych punktów daje około 7-krotne zmniejszenie błędu. Przy 10000 punktów błąd maleje o połowę, a użycie 100000 punktów daje błąd przybliżenia całki na poziomie 0.12

Tabela 5.12: Dane z iteracji w metodzie Crude Monte Carlo dla funkcji trygonometrycznej

n	minValue	maxValue	AVG	E
1	-1.999712843	1.999075162	0.438109	79.03%
10	-0.6383128719	1.133784472	0.190505	22.15%
100	-0.1079735697	0.5645576157	0.261066	6.68%
1000	0.1529912591	0.3583314798	0.246922	0.9%
10000	0.2148820935	0.2831131619	0.243623	0.44%
100000	0.2349722501	0.2594303218	0.244408	0.12%

5.1.7. Metoda Monte Carlo

Listing 11 prezentuje nagłówek metody Monte Carlo. Metoda ta przyjmuje 5 parametrów. Pierwszym jest całkowana funkcja, drugim dolny przedział całkowania, trzecim górny przedział całkowania, czwartym ilośc punktów, którymi będziemy próbkować funkcję. Ostatni jest argument opcjonalny i może nim być minimalna oraz maksymalna wartość przyjmowana przez funkcję na przedziale $[a, b]$. Funkcja zwraca przybliżoną wartość całki.

```
1     def f_MonteC(f,a,b,n,*args):
2     """
3             :param fun: funkcja
4             :param a: dolny przedzial calkowania
5             :param b: gorny przedzial calkowania
6             :param n: ilosc losowych punktow
7             :param *args: opcjonalny parametr, minimalna oraz
    maksymalna wartosc funkcji
8             :return: przyblizona wartosc calki
9             """
```

Listing 14: Kod w języku python implementujący metodę Monte Carlo

Funkcja wielomianowa

Tabela 5.13 przedstawia wyniki uzyskane z działania metody Monte Carlo dla

funkcii podcałkowej (5.25). Z uwagi na specyfikę tej metody wynik dla 10 jak i 100 punktów jest praktycznie identyczny. Przy 1000 losowych punktów błąd metody spadł 3 krotnie. Nastepne zwiększenie punktów z 1000 do 10000 poskutkowało zmniejszeniem błędu o 2 rzędy wielkości. Ostatni test przyniósł wynik bardzo podobny pomimo, że liczba punktów została zwiększona 10- krotnie.

Tabela 5.13: Dane z iteracji w metodzie Monte Carlo dla funkcji wielomianowej

n	minValue	maxValue	AVG	E
10	0.0	43.05728	14.270413	0.346113
100	7.996352	22.143744	14.288866	0.364566
1000	12.056038	16.607808	13.814621	0.109679
10000	13.24934	14.7871	13.926631	0.002331
100000	13.671917	14.164615	13.922184	0.002116

Funkcja trygonometryczna

Tabela (5.14) przedstawia wyniki uzyskanie z działania metody Monte Carlo dla funkcii podcałkowej (5.26). Jak widać w tabeli losowość tej metody sprawiła, że dla 10 punktów błąd aproksymacji wyniósł 0.003293. Następny tak dobry wynik udało się osiągnąć dopiero dla 10000 punktów.

Tabela 5.14: Dane z iteracji w metodzie Monte Carlo dla funkcji trygonometrycznej

n	minValue	maxValue	AVG	E
10	-2.0	1.6	0.248	0.003293
100	-0.24	0.76	0.2556	0.010893
1000	0.08	0.384	0.232	0.012707
10000	0.1708	0.3076	0.248704	0.003997
100000	0.22332	0.2624	0.244092	0.000615

5.2. Porównanie metod

Porównanie metod bazujących na interpolacji wielomianowej

Tabela 5.15 przedstawia porównanie metod całkowania opartych o interpolację wielomianową dla 2 podprzedziałów. Jak widzimy metodą dającą najgorszy wynik jest metoda trapezów, zwraca ona wynik 2-krotnie gorszy niż metoda prostokątów. Metoda Simpsona daje o 1 rząd wielkości błąd mniejszy niż metody trapezów i prostokątów. Metoda Gaussa Legendrea uzyskała błąd prawie o połowę niższy niż metoda Simpsona.

Tabela 5.15: Dane z iteracji metod całkowania opartych na interpolacji wielomianem dla 2 podprzedziałów

n	Metoda	I	E
2	prostokątów	11.6352	2.2891
2	trapezów	18.579200	4.654900
2	Simpsona	14.333867	0.409567
2	Gaussa Legendre	13.651200	0.273100

W tabeli 5.16 umieszczone zostały wyniki z działania metod dla 4 podprzedziałów. Jak widać w tabeli stosunek różnicy pomiędzy metodą prostokątów i trapezów pozostał podobny. Błąd w metodzie Prostokątów i Trapezów wynosi $\frac{1}{2^2}$ w stosunku do błędu dla 2 podprzedziałów. W metodzie Simpsona wraz z podwojeniem podprzedziałów błąd zmniejsza się o $\frac{1}{2^4}$. Metoda Gaussa Legendrea daje wynik dokładny dla wielomianów stopnia $2n-1$, w związku z czym już dla 3 węzłów zwróciła ona wynik dokładny.

Porównanie metody Simpsona oraz Gaussa Legendra

Jak widać w tabeli 5.17 początkowoa metoda Simpsona wykazuje dokładniejsze przybliżenie całki niż metoda Gaussa Legendre'a, przy 6 podprzedziałach zwraza błąd 10-krotnie mniejszy. Dopiero gdy liczba podprzedziałów wynosi 8 metoda Gaussa okazuje się być lepsza i zwracać błąd mniejszy o 2 rzędy wielkości. Przy 10 podprzedziałach kwadratura Gaussa zwraca wynik dokładny jeśli zaokrąglimy analitycznie obliczoną całkę do 6 miejsca po przecinkuo.

Porównanie metody trapezów i adaptacyjnej metody trapezów

Tabela 5.16: Dane z metod całkowania opartych na interpolacji wielomianem dla 4 podprzedziałów

n	Metoda	I	E
4	prostokątów	13.33520	0.58910
4	trapezów	15.107200	1.182900
4	Simpsona	13.949867	0.025567
4	Gaussa Legendre	13.924267	0.000033

Tabela 5.17: Dane z iteracji metod Gaussa i Simpsona

n	Metoda	I	E
2	Simpsona	1.223177	9.784691e-01
4	Simpsona	-0.580448	8.251555e-01
6	Simpsona	0.528331	2.836232e-01
8	Simpsona	0.284161	3.945382e-02
10	Simpsona	0.257320	1.261270e-02
2	Gaussa Legendre	-0.184912	4.296190e-01
4	Gaussa Legendre	-0.611561	8.562686e-01
6	Gaussa Legendre	0.212634	3.207296e-02
8	Gaussa Legendre	0.244457	2.499462e-04
10	Gaussa Legendre	0.244707	6.560406e-07

Tabela 5.18 przedstawia porównanie metody trapezów i adaptacyjnej metody trapezów. Porównanie zostało przeprowadzone na funkcji (5.25). Funkcja ta wraz z posuwaniem się na prawo po osi odciętych przyjmuje coraz większą wartość, przez co żeby mogła osiągnąć zadeklarowaną dokładność, dzieli prawą stronę równania na mniejsze podprzedziały, gdzie zwykła metoda trapezów dzieli funkcję na podprzedziały równo rozmieszczone wzdłuż całego przedziału całkowania. Stąd wynik uzyskany przez metodę adaptacyjną jest 2-krotnie dokładniejszy dla ostatnich dwóch iteracji pokazanych w tabeli.

Tabela 5.18: Dane z iteracji metody trapezów i adaptacyjnej metody Trapezów dla funkcji (5.25)

n	Metoda	I	E
4	trapezów	15.1072	1.1829
6	trapezów	14.451595	0.527295
8	trapezów	14.2212	0.2969
10	trapezów	14.158947	0.234647
12	trapezów	14.056336	0.132036
16	trapezów	13.998575	0.074275
4	adaptacyjna	15.1072	1.1829
6	adaptacyjna	14.236700	0,3124
8	adaptacyjna	14,08788	0,16358
10	adaptacyjna	14.017669	0,093369
12	adaptacyjna	13.995822	0,071522
16	adaptacyjna	13.95722	0,03292

Metoda adaptacyjna radzi sobię lepiej od metody trapezów gdy całkowana funkcja przyjmuje różne tempo zmian wartości na zadanym przedziale. Przykładem takiej funkcji jest (5.27). Funkcja ta coraz bardziej oscyluje w pobliżu prawego punktu końcowego.

Wyniki działania obu metod całkowania tej funkcji prezentuje tabela 5.19.

Liczba podprzedziałów nie jest parametrem metody adaptacyjnej dlatego została dobrana na podstawie tego na jaką ilość podprzedziałów została podzielona funkcja przy danej tolerancji.

Dla 4 podprzedziałów widzimy, że zadana tolerancja była na tyle duża, że metoda adaptacyjna podzieliła przedział całkowania w ten sam sposób co metoda trapezów. Wraz ze wzrostem ilości podprzedziałów adaptacyjna metoda wykazuje znacznie lepszą dokładność w przybliżaniu całki. Dla 98 podprzedziałów błąd metody adaptacyjnej jest o 2 rzędy wielkości mniejszy niż przy zwykłej metodzie trapezów.

Tabela 5.19: Dane z iteracji metody trapezów i adaptacyjnej metody Trapezów dla funkcji (5.27)

n	Metoda	I	e
4	trapezów	9.731300	12.556630
8	trapezów	-8.227000	5.401670
10	trapezów	7.068914	9.894244
18	trapezów	5.822512	2.997182
46	trapezów	-2.827854	0.002524
56	trapezów	-1.467744	1.357586
98	trapezów	2.295961	0.529369
4	adaptacyjna	9.731300	12.556630
8	adaptacyjna	-9.439613	6.614283
10	adaptacyjna	-11.1515886	8.326258
18	adaptacyjna	-3.881971	1.056641
46	adaptacyjna	-2.868944	0.043614
56	adaptacyjna	-2.846306	0.020976
98	adaptacyjna	-2.825574	0.000244

Porównanie metod Monte Carlo

Tabela 5.20 przedstawia wyniki z całkowania funkcji (5.26) metodami Crude Monte Carlo oraz Monte Carlo. Początkowo metoda Monte Carlo wykazuje wększą dokładność niż metoda Crude Monte Carlo, osiągając błąd o 1 rząd wielkości mniejszy. Sytuacja odwraca się przy 10000 punktów, metoda Monte Carlo z racji swojej specyfiki osiągnęła błąd większy niż w poprzedniej iteracji, podczas gdy metoda Crude zwiększa dokładność waz z kolejnymi iteracjami. Po próbkowaniu 1000000 punktami, metoda Crude okazała sie 3-krotnie lepsza niż metoda Monte Carlo.

Tabela 5.20: Dane z działania metod Crude Monte Carlo oraz Monte Carlo dla funkcji trgonometrycznej

n	Metoda	I	E
10	Crude	0.217109	0.027599
100	Crude	0.284756	0.040048
1000	Crude	0.25273	0.008022
10000	Crude	0.245071	0.000364
100000	Crude	0.244593	0.000115
1000000	Crude	0.244602	0.000106
10	Monte Carlo	0.144	0.100707
100	Monte Carlo	0.2384	0.006307
1000	Monte Carlo	0.24528	0.000573
10000	Monte Carlo	0.247472	0.002765
100000	Monte Carlo	0.245164	0.000457
1000000	Monte Carlo	0.245026	0.000318

Tabela 5.20 przedstawia wyniki uzyskane z całkowania funkcji (5.25) metodami Crude Monte Carlo oraz Monte Carlo. Podobnie jak w przypadku całkowania poprzedniej funkcji tak i tu dla 1000000 punktów, metoda Crude okazała się około 3-krotnie lepsza niż standardowa metoda Monte Carlo.

Tabela 5.21: Dane z działania metod Crude Monte Carlo oraz Monte Carlo dla funkcji wielomianowej

n	Metoda	I	E
10	Crude	13.43583	0.48847
100	Crude	14.133026	0.208726
1000	Crude	13.825492	0.098808
10000	Crude	13.945694	0.021394
100000	Crude	13.925673	0.001373
1000000	Crude	13.92384	0.00046
10	Monte Carlo	14.331923	0.407623
100	Monte Carlo	13.895199	0.029101
1000	Monte Carlo	13.878592	0.045708
10000	Monte Carlo	13.910085	0.014215
100000	Monte Carlo	13.93213	0.00783
1000000	Monte Carlo	13.92549	0.00119

6. Podsumowanie i wnioski końcowe

Celem niniejszej pracy inżynierskiej była implementacja oraz analiza wybranych metod całkowania numerycznego oraz stworzenie laboratorium dla przedmiotu "Metody Numeryczne."

W pierwszym rozdziale pracy omówiony został ogólny problem obliczania całki oznaczonej przy pomocy metod numerycznych oraz krótko nakreślone zostały metody całkowania poruszane w niniejszej pracy. W drugim rozdziale zostały opisane teoretyczne podstawy oraz przykładowy kod metod całkowania, które zostały zaimplementowane w stworzonej przeze mnie bibliotece. Trzeci rozdział został poświęcony językowi programowania, biblioteką oraz narzędzią które został wykorzystane do zaimplementowania wspomnianych kwadratur. Rozdział czwarty zawiera opis stworzonej instrukcji laboratoryjnej w środowisku Jupyter Notebook W piątym rozdziale zawarte zostały wyniki oraz komentarze odnośniez działania zaimplementowanych metod całkowania. Celem rozdziału piątego było porównanie poszczególnych metod całkowania ze sobą oraz wyjaśnienie relacji zachodzących między nimi. Rozdział ten zawiera porównanie metod

bazujących na interpolacji wielomianowej, porównanie metody Simpsona z metodą Gaussa Legendre, porównanie metody metody trapezów z adaptacyjną metodą trapezów, oraz porównanie metod Monte Carlo. Wszystkie porównania zostały dokonane w oparciu o stabelaryzowane dane uzyskane przez działania metod z zaimplementowanej przeze mnie biblioteki.

Sposobem na ulepszenie pracy było by implementacja algorytmów w sposób bardziej zwięzły i odporny na potencjalne błędy. Powód dla, którego zdecydowano się na taką a nie inną implementację jest to, że utworzone metody miały posłużyć do stworzenia laboratorium dla studentów, a ich celem było jak najprostsze pokazanie działana podstawowych algorytmów całkowania.

Autor za własny wkład pracy uważa stworzenie biblioteki funkcji zawierającej wybrane metody całkowania numerycznego oraz utworzenie instrukcji laboratoryjnej dla przedmiotu "Metody Numeryczne". Opracowana instrukcja laboratoryjna opiera się na wspomnianej stworzonej bibliotece metod całkowania, zawiera także skrócony opis teoretyczny wykorzystywanych kwadratur oraz szereg zadań do realizacji przez osoby z niej korzystające.

.

Literatura

[1] http://www.drjamesnagel.com/notes/Nagel

[2] Numerical Methods for Engineers, 6th Ed(str 588)

[3] https://en.wikipedia.org/wiki/Newton

[4] Willie Aboumrad, CME 108/MATH 114 Introduction to Scientific Computing

[5] https://www.cs.mcgill.ca/

[6] https://en.citizendium.org/wiki/Legendre-Gauss_Quadrature_formula

[7] Numerical Methods for Engineers, 6th Ed(644-645)

[8] Numerical Methods for Engineers, 6th Ed

[9] https://www.mimuw.edu.pl/ leszekp/dydaktyka/MO19L-g/adaptn.pdf

[10] https://www.math.usm.edu/lambers/mat460/fall09/lecture30.pdf

[11] http://www.algorytm.org/procedury-numeryczne/calkowanie-numeryczne-metoda-monte-carlo-ii.html

[12] http://www.algorytm.org/procedury-numeryczne/calkowanie-numeryczne-metoda-monte-carlo-i.html

[13] https://www.python.org/doc/essays/blurb/

[14] https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python

[15] https://www.w3schools.com/python/numpy/numpy_intro.asp

[16] https://www.pythonlikeyoumeanit.com/Module3_IntroducingNumpy/VectorizedOperations

[17] https://jupyter.org/

[18] https://en.wikipedia.org/wiki/Read

[19] https://en.wikipedia.org/wiki/Project_Jupyter

[20] https://jupyter4edu.github.io/jupyter-edu-book/jupyter.html

POLITECHNIKA RZESZOWSKA im. I. Łukasiewicza Rzeszów, 2022
Wydział Elektrotechniki i Informatyki

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

RZESZOW UNIVERSITY OF TECHNOLOGY Rzeszow, 2022
Faculty of Electrical and Computer Engineering

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