Институт информационных и вычислительных технологий

Лабораторная работа №2
“Решение нелинейных уравнений методом простой итерации”
по курсу
“**Вычислительные методы**”

Выполнил:
Студент Балашов С.А.
Проверила:
Старший преподаватель кафедры МКМ
Шевченко О.В.

Москва, 2021

Содержание

Цель работы.
Применить на практике простейшие численные методы вычисления интегралов и производных. Исследовать поведение погрешности методов при измельчении шага. Познакомиться с понятиями порядка точности и обусловленности (плохой/хорошей) задачи и их отражением в расчетах. Вычислить определенный интеграл с заданной точностью.

Задача 1.
Найти приближенные значения интеграла и производной, используя указанные в индивидуальном варианте методы. Организовать серию расчетов с шагами

$h_k = (b-a)/10^k$, k = (1,2, ...,15). Сделать выводы о порядке точности и обусловленности методов.

Метод численного интегрирования – метод **трапеций**.

Формула численного дифференцирования – **левая**.

1. Вычислить точное значение J интеграла по формуле Ньютона-Лейбница.

$J = \sin(x) * (1 + \cos(x)) / \sqrt{1 + \cos^2(x)} = 1.295587149392638$

2. Реализовать программно составную формулу численного интегрирования. Вычислить с ее помощью приближенные значения интеграла I_k для

k = 1,2, ... ,15. Заполнить второй столбец таблицы.

3. Для каждого приближенного значения интеграла найти погрешность $\Delta_k = J - I_k$. Заполнить третий столбец таблицы.

4. Вычислить точное значение D производной, подставив число a в формулу для f’(x).

$D = (-\sin^2(x) + \cos^4(x) + \cos^3(x) + \cos^2(x) + \cos(x) * (1 + \sin^2(x))) / (1 + \cos^2(x))^{3/2} = 1.414213562373095$

5. Реализовать программно формулу численного дифференцирования. Вычислить с ее помощью приближенные значения производной d_k для k = 1,2, … ,15. Заполнить 4-ый столбец таблицы.

6. Для каждого приближенного значения производной найти погрешность $\Delta_k = D - d_k$. Заполнить 5-ый столбец таблицы.

Табл. 1 Результаты интегрирования методом трапеций

Шаг h	Приближенное значение интеграла	Погрешность численного интегрирования	Затраченное на расчет время
(b-a)/10	1.2906159471058 3	0.00497120228680537	0.0001001358032 22656
$(b-a)/10^2$	1.2955375084044 4	$4.964098820203766*10^{-5}$	0.0001800060272 2168
$(b-a)/10^3$	1.2955866529898	$4.964028377241192*10^{-7}$	0.0010590553283 6914
$(b-a)/10^4$	1.2955871444286 1	$4.964027189302556*10^{-9}$	0.0121929645538 33
$(b-a)/10^5$	1.2955871493429 9	$4.9648951616632075*10^{-11}$	0.1380228996276 86
$(b-a)/10^6$	1.2955871493921 5	$4.840572387365683*10^{-13}$	0.8265459537506 1
$(b-a)/10^7$	1.2955871493927 8	$1.425526363618701*10^{-13}$	6.5794799327850 3
$(b-a)/10^8$	1.2955871493920 3	$6.046274592108603*10^{-13}$	67.015506267547 6
$(b-a)/10^9$	1.2955871493968 4	$4.204192549650543*10^{-12}$	619.66247797012 3

Табл. 2 Результаты дифференцирования

Шаг h	Приближенное значение производной	Погрешность численного дифференцирования	Затраченное на расчет время
(b-a)/10	1.4083779968112 6	0.0058355655618311 8	2.40802764892578E-05
(b-a)/10^2	1.4141554031982 3	5.81591748640697E-05	2.19345092773438E-05
(b-a)/10^3	1.4142129808008 8	5.81572213720349E-07	2.09808349609375E-05
(b-a)/10^4	1.4142135565573 7	5.81572012769982E-09	2.09808349609375E-05
(b-a)/10^5	1.4142135623149 4	5.81570347435445E-11	2.21729278564453E-05
(b-a)/10^6	1.4142135623725 1	5.81534820298657E-13	2.09808349609375E-05
(b-a)/10^7	1.4142135623730 9	5.55111512312578E-15	2.09808349609375E-05
(b-a)/10^8	1.4142135623731	2.22044604925031E-16	2.21729278564453E-05
(b-a)/10^9	1.4142135623731	0.0	2.09808349609375E-05

7. Сделать выводы (отдельно для каждой из двух формул).
1. Указать порядок точности формулы по h.
Формула трапеций – 2
Левая формула численного дифференцирования - 2
2. Пользуясь заполненной таблицей, показать, что расчет подтверждает указанный порядок точности.
Формула трапеций - при уменьшении шага в 10 раз погрешность уменьшается в 100 => порядок точности по h = log10(100) = 2
Левая формула численного дифференцирования - аналогично интегрированию
3. Отметить, все ли данные соответствующего столбца можно использовать для анализа порядка точности.

Формула трапеций - После 6 шага зависимость уменьшения погрешности от уменьшения шага меняется в сторону уменьшения порядка точности, поэтому эти строки для анализа использовать нежелательно.
Левая формула численного дифференцирования - На всех шагах зависимость погрешности от шага сохраняется, значит их все можно использовать для анализа.
4. Указать шаг h, при котором достигается наилучшая точность.
Для трапеций – $h = (b-a) / 10^{-7}$
Для дифференцирования – $h = (b-a) / 10^{-9}$
5. Определить, проявилась ли в расчетах (и в чем именно) хорошая или плохая обусловленность метода.
Для интегрирования – операции с каждым шагом требуют больше времени на выполнение примерно в 10 раз, причем после быстрого уменьшения погрешности до 6 шага, изменения становятся менее значительными.
Для дифференцирования – операции выполнялись с примерно одной скоростью, независимо от шага. С каждым шагом погрешность уменьшается в ~100 раз.

Задача 2.
Повторить расчет интеграла из Задачи 1 с помощью квадратурной формулы Симпсона. Сравнить результаты с результатами Задачи 1 (с учетом порядков точности использованных формул). Сделать выводы о порядке точности и обусловленности методов.
Вычислить значение интеграла из Задачи 1 с помощью составной квадратурной формул Симпсона с заданной в индивидуальном варианте точностью ε. (без разбиения отрезка интегрирования, см. алгоритм в Приложении). Предусмотреть возврат значения шага, на котором происходит выход из расчета. Заполнить таблицу

Табл. 3 Результат интегрирования методом Симпсона

Шаг h	Приближенное значение интеграла	Погрешность численного интегрирования	Затраченное на расчет время
(b-a)/10	1.2956169436736864	2.9794281048323157e-05	5.4836273193359375e-05
$(b-a)/10^2$	1.295587152239709	2.8470710233818863e-09	0.00010395050048828125
$(b-a)/10^3$	1.2955871493929225	2.844391389089651e-13	0.0007810592651367188
$(b-a)/10^4$	1.2955871493926392	1.1102230246251565e-15	0.009515047073364258
$(b-a)/10^5$	1.2955871493926387	6.661338147750939e-16	0.13813185691833496
$(b-a)/10^6$	1.295587149392632	5.995204332975845e-15	0.7282869815826416
$(b-a)/10^7$	1.295587149392688	4.9960036108132044e-14	6.614024877548218
$(b-a)/10^8$	1.295587149392505	1.3300471835009375e-13	77.24067306518555

Табл. 4 Сравнение результатов методов трапеций и Симпсона

Шаг h	Приближенное значение интеграла	Погрешность численного интегрирования	Приближенное значение интеграла методом Симпсона	Погрешность численного интегрирования
$(b-a)/10$	1.2906159471058 3	0.0049712022868 0537	1.2956169436736 864	2.9794281048323157e-05
$(b-a)/10^2$	1.2955375084044 4	$4.964098820203766*10^{-5}$	1.2955871522397709	2.8470710233818863e-09
$(b-a)/10^3$	1.2955866529898	$4.964028377241192*10^{-7}$	1.2955871493929225	2.8443913890896 51e-13
$(b-a)/10^4$	1.2955871444 2861	$4.964027189302556*10^{-9}$	1.2955871493926392	1.1102230246251565e-15
$(b-a)/10^5$	1.2955871493 4299	$4.9648951616632075*10^{-11}$	1.2955871493926387	6.6613381477509 39e-16
$(b-a)/10^6$	1.2955871493 9215	$4.840572387365683*10^{-13}$	1.2955871493922632	5.9952043329758 45e-15
$(b-a)/10^7$	1.2955871493 9278	$1.425526363618701*10^{-13}$	1.2955871493922688	4.9960036108132044e-14
$(b-a)/10^8$	1.2955871493 9203	$6.046274592108603*10^{-13}$	1.2955871493922505	1.3300471835009375e-13
$(b-a)/10^9$	1.2955871493 9684	$4.204192549650543*10^{-12}$	-	-

Метод Симпсона до 5 шага обладает погрешностью в 100 раз меньше, чем у метода трапеций согласно таблице. Это соответствует порядкам точности 4 у метода Симпсона и 2 - у метода трапеций. Значит, использовать метод Симпсона на начальных шагах более правильно из-за меньшей погрешности, но после 5 шага их значения примерно выравниваются, однако метод Симпсона затрачивает больше времени на вычисления. Поэтому, после 5 шага предпочтительнее использовать метод трапеций.

Табл. 5 Результаты интегрирования методом Симпсона с заданной точностью

Значение точности	Точное значение J	Приближенное значение I	Абсолютная погрешность	Значение шага интегрирования	Затраченное на расчет время	Количество итераций
0.0001	1.295587149392638	1.2955095847231204	7.756466951769347e-05	(b-a)/10	0	3
		1.2955747392790031	1.2410113634953035e-05	$(b-a)/10^2$	0	1
		1.2955870252919441	1.2410069394341 86e-07	$(b-a)/10^3$	0.003002882003784 1797	1
		1.2955871481516326	1.2410055205691606e-09	$(b-a)/10^4$	0.025021314620971 68	1
		1.2955871493802222	1.2415846128988 05e-11	$(b-a)/10^5$	0.241211891174316 4	1

		1.2955871493925 52	8.615330671091215e-14	(b-a)/10^6	2.246970653533955	1
		1.2955871493927 3	9.192646643896296e-14	(b-a)/10^7	22.28554940223694	1
		1.2955871493928 481	2.1005419625907962e-13	(b-a)/10^8	221.6754541397094 7	1

Выводы:

Сделать выводы: сравнить значение шага, на котором достигнута заданная точность, с данными из предыдущей таблицы и объяснить, проявилось ли преимущество одной из формул над другой.

Согласно таблицам 3 и 5, составной формуле для достижения точности потребовалось дойти до шага (b - a) / (10 * 2^n), где 2^n - количество итераций = 2^3 = 8, а простой формуле требуется шаг (b - a) / 10. Однако, время на вычисление по составной формуле потребуется меньше, значит она более эффективна.

Приложение 1. Программа для расчета интегрирования и дифференцирования.

main.py

```
import math
import time
import first as F
import second
import second as S
import math

print(F.J)

# Вывод интеграла и дельты
for i in range(1, 10):
    start_time = time.time()
    tmp = F.trap(F.f, 0, math.pi / 2, i)
    print('I(', i, ') = ', tmp)
    print('J - I(', i, ') = ', abs(F.J - tmp))
    end_time = time.time()
    print('Time was used: ', end_time - start_time)
print()
print('D = ', F.d(0))

# Вывод производной и дельты
for i in range(1, 15):
    start_time = time.time()
    tmp = F.dif(F.f, 0, math.pi / 2, i)
    print('d(', i, ') = ', tmp)
    print('D - d(', i, ') = ', abs(F.D - tmp))
    end_time = time.time()
    print('Time was used: ', end_time - start_time)
print()

# Вывод интеграла по формуле симпсона и дельты
for i in range(1, 10):
    start_time = time.time()
    tmp = S.simpson_rule(F.f, 0, math.pi / 2, i)
    print('S(', i, ') = ', tmp)
    print('J - S(', i, ') = ', abs(F.J - tmp))
    end_time = time.time()
    print('Time was used: ', end_time - start_time)
print()

# Вывод интеграла по формуле с симпсона с заданной точностью и дельы
for i in range(1, 9):
    start_time = time.time()
    tmp = S.simpson(F.f, 0, math.pi / 2, 0.0001,  i)
    print('SimpS(', i, ') = ', tmp[0])
    print('J - SimpS(', i, ') = ', abs(F.J - tmp[0]))
    print('Номер итерации: ', tmp[1])
    end_time = time.time()
    print('Time was used: ', end_time - start_time)
print()
```

first.py

```
from math import cos, pi, sin, sqrt, pow, asinh

# ПЕРВАЯ ЗАДАЧА
# Функция
def f(x):
```

```
    return sin(x) * (1 + cos(x)) / sqrt(1 + pow(cos(x), 2))
# Интеграл квадратурной формулой трапеций
def trap(func, a, b, k):
    h = 1.0 * (b - a) / 10**k
    Ik = 0.5 * (func(a) + func(b))
    for i in range(1, int(pow(10, k))):
        Ik += func(a + i * h)

    return Ik * h
# Точный интеграл функции по формуле Ньютона-Лейбница
J = -1 + sqrt(2) + asinh(1)
# Производная функции по x
def d(x):
    return (-pow(sin(x), 2) + pow(cos(x), 4) +
            pow(cos(x), 3) + pow(cos(x), 2) +
            cos(x) * (1 + pow(sin(x), 2))) / (
               pow(1 + pow(cos(x), 2), 3 / 2))


# Производная функции в точке a
D = d(0)
# Формула левого численного дифференцирования
def dif(func, a, b, k):
    h = 1.0 * (b - a) / pow(10, k)
    return (func(a) - func(a - h)) / h
```

second.py

```
import math
from math import cos, pi, sin, sqrt, pow, ceil
import first as F

# ВТОРАЯ ЗАДАЧА
# Интеграл по формуле Симпсона
def simpson_rule(func, a, b, k):
    h = 1.0 * (b - a) / pow(10, k)
    simp_sum = (func(a) + 4 * func(a + h) + func(b))
    for i in range(1, int(10**k / 2)):
        simp_sum += 2 * func(a + (2 * i) * h) + 4 * func(a + (2 * i + 1) * h)
    return simp_sum * h / 3

def double_trap(func, a, b, k, n):
    h = 1.0 * (b - a) / 10**k
    Ik = 0.5 * (func(a) + func(b))
    for i in range(1, int(n * 10**k)):
        Ik += func(a + i * h / n)
    return Ik * h / n

def simpson(func, a, b, eps, k):
    n = 2
    I = F.trap(func, a, b, k)
    I_next = double_trap(func, a, b, k, n)
    ans =  I_next
    est = abs(I_next - I) / (2**3 - 1)
    while est > eps:
        n *= 2
        I = I_next
        I_next = double_trap(func, a, b, k, n)
        est = abs(I_next - I) / (2 ** 3 - 1)
        ans = I_next
    return ans, math.log2(n)
```