Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет «Высшая школа экономики»

Образовательная программа
«Прикладная математика»

ОТЧЕТ
По лабораторной работе № 1

По предмету
«Численные методы»

По теме
«Теория погрешностей и машинная арифметика»

Выполнил
студент группы БПМ211
Кудряшов Максим Дмитриевич

Проверил
Брандышев Петр Евгеньевич

Москва - 2024

Содержание

1 Расчет погрешности частичных сумм ряда (№ 1.1.8)

1.1 Формулировка задания

1. Найти сумму ряда S аналитически.

$$S = \sum_{n=0}^{\infty} \frac{32}{n^2 + 9n + 20}$$

2. Найти частичные суммы ряда S_N при $N = 10, 10^2, 10^3, 10^4, 10^5$.

$$S_N = \sum_{n=0}^{N} \frac{32}{n^2 + 9n + 20}$$

3. Вычислить погрешности для каждого N.

4. Вычислить количество значащих цифр для каждого N.

5. Построить гистограмму зависимости верных цифр результата от N.

1.2 Найдем сумму ряда аналитически

Посчитаем значение предела в Wolfram:

In[1]:= $\sum_{n=0}^{\infty} \frac{32}{\left(n^2 + 9\, n + 20\right)}$

Out[1]= 8

Итого $S = \sum_{n=0}^{\infty} \frac{32}{n^2+9n+20} = 8$

1.3 Найдем погрешности

Для различных значений N вычислим частичные суммы, погрешности и количество значащих цифр.

1.3.1 Код на Python

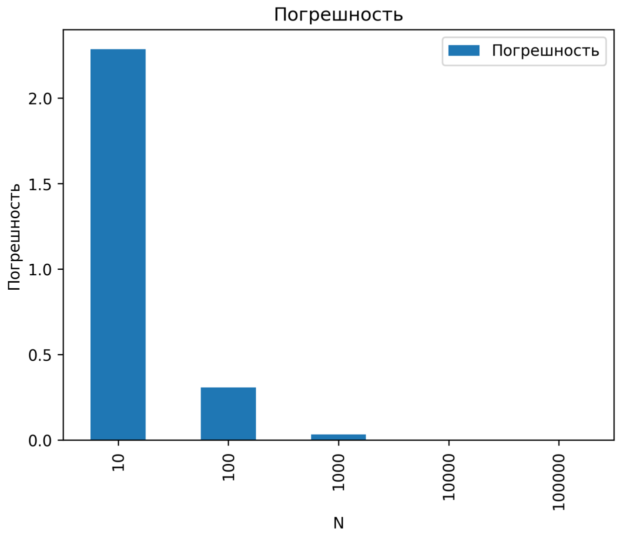
```
import pandas as pd


def calculateNum(n: int):
    return 32 / (n ** 2 + 9 * n + 20)


def calculateSum(N: int):
    s = 0
    for n in range(N):
        s += calculateNum(n)
```

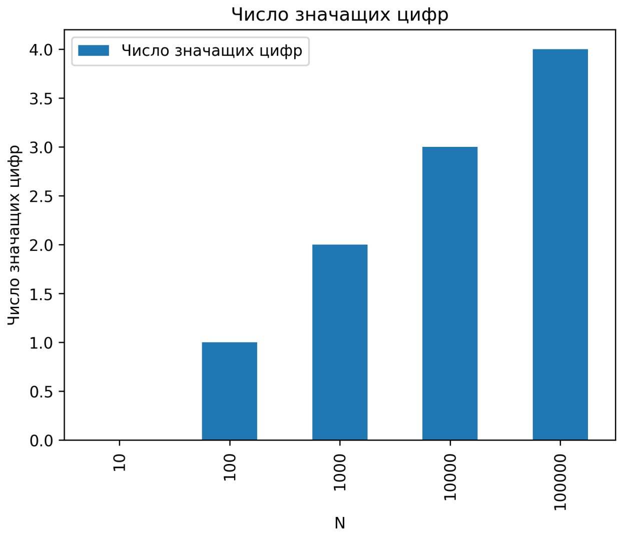
```
    return s


def calculateData():
    N_list = [int(10 ** k) for k in range(1, 6)]

    sums = {}
    errors = {}
    digits = {}
    exact_sum = 8

    for N in N_list:
        sums[N] = calculateSum(N)
        errors[N] = abs(sums[N] - exact_sum)

    for N in N_list:
        digits[N] = 0
        for i in range(0, 50):
            order_of_error = 1 / (10 ** i)
            if errors[N] <= order_of_error:
                digits[N] += 1
            else:
                break

    sums_list = []
    errors_list = []
    digits_list = []
    for N in N_list:
        sums_list.append(sums[N])
        errors_list.append(errors[N])
        digits_list.append(digits[N])

    pd.options.display.float_format = '{:,.8f}'.format
    df = pd.DataFrame({
        'N': N_list,
        "Значение": sums_list,
        "Погрешность": errors_list,
        "Число значащих цифр": digits_list
    })
    return df


df = calculateData()
print(df)
df.to_latex("table.tex")
```

1.3.2 Посчитанные данные

	N	Значение	Погрешность	Число значащих цифр
0	10	5.714286	2.285714	0
1	100	7.692308	0.307692	1
2	1000	7.968127	0.031873	2
3	10000	7.996801	0.003199	3
4	100000	7.999680	0.000320	4

1.3.3 Визуализация посчитанных данных





2 Квадратное уравнение (№ 1.5.2)

2.1 Формулировка задания

Дано квадратное уравнение. Предполагается, что один из коэффициентов уравнения (в индивидуальном варианте помечен *) получен в результате округления. Произвести теоретическую оценку погрешностей корней в зависимости от погрешности коэффициента. Вычислить корни уравнения при нескольких различных значениях коэффициента в пределах заданной точности. Сравнить полученные результаты.

$$ax^2 + bx + c^* = 0$$
$$a = 1$$
$$b = 27.4$$
$$c^* = 187.65$$

2.2 Нахождение погрешности

В общем случае:

$$x^* = x \pm \Delta x$$
$$\bar{\Delta}x = |x|\bar{\delta}x$$
$$\bar{\Delta}f(x) = |f'(x)|\bar{\Delta}x$$
$$\bar{\delta}f(x) = \frac{\bar{\Delta}f(x)}{|f(x)|} = \frac{|f'(x)|\bar{\Delta}x}{|f(x)|} = \frac{|f'(x)||x|\bar{\delta}x}{|f(x)|}$$

Итого:

$$\bar{\delta}f(x) = \frac{|f'(x)||x|}{|f(x)|}\bar{\delta}x$$

В других обобзначениях:

$$\bar{\delta}x(c) = \frac{|x'(c)||c|}{|x(c)|}\bar{\delta}c$$

Найдем производную и значение функции:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$x_1 = -13.9, x_2 = -13.5$$

$$x'(c) = -\frac{1}{\sqrt{b^2 - 4ac}} = -2.5$$

Подставим:

$$\bar{\delta}x_1 = \frac{2.5 \cdot 187.65}{13.9} \cdot \bar{\delta}c = 33.75 \cdot \bar{\delta}c$$

$$\bar{\delta}x_2 = \frac{2.5 \cdot 187.65}{13.5} \cdot \bar{\delta}c = 34.75 \cdot \bar{\delta}c$$

Оценка погрешности:

$c^* = 187.65$ было округлено => $\bar{\Delta}c = 0.005$

$\bar{\Delta}x(c) = |x'(c)|\bar{\Delta}c = -2.5 \cdot 0.005 = -0{,}0125$

Поэтому x_1 лежит в промежутке $[-13.5 - 0.0125, -13.5 + 0.0125] = [-13.5125, -13.4875]$

Поэтому x_2 лежит в промежутке $[-13.9 - 0.0125, -13.9 + 0.0125] = [-13.9125, -13.8875]$

Найдем погрешности корней при разных погрешностях c:

2.3 Код на Python

```python
import numpy as np

eq = np.array([1, 27.4, 187.65])

c = 187.65
error = 0.005
errors = np.arange(-error, error, 2 * error / 20)

min_x1 = [-13.5125, -13.4875]
for error in errors:
    x = np.roots(eq + np.array([0, 0, error]))
    x1 = x[1]
    is_in_range = (min_x1[0] <= x1 <= min_x1[1])
    print(f"{x1 = },\t{error = },\t{is_in_range = }")

print()

min_x2 = [-13.9125, -13.8875]
for error in errors:
    x = np.roots(eq + np.array([0, 0, error]))
    x2 = x[0]
    is_in_range = (min_x2[0] <= x2 <= min_x2[1])
    print(f"{x2 = },\t{error = },\t{is_in_range = }")


```

2.4 Результат работы программы

```
x1 = -13.487867965644103,       error = -0.005,       is_in_range = True
x1 = -13.489049768902753,       error = -0.0045000000000000005,       is_in_range = True
```

```
x1 = -13.490238230366051,       error = -0.004000000000000001,       is_in_range = True
x1 = -13.49143346385391,       error = -0.0035000000000000014,       is_in_range = True
x1 = -13.492635586466816,       error = -0.003000000000000002,       is_in_range = True
x1 = -13.493844718719183,       error = -0.0025000000000000022,       is_in_range = True
x1 = -13.495060984680846,       error = -0.0020000000000000026,       is_in_range = True
x1 = -13.496284512125445,       error = -0.001500000000000003,       is_in_range = True
x1 = -13.497515432686884,       error = -0.0010000000000000035,       is_in_range = True
x1 = -13.498753882025108,       error = -0.0005000000000000039,       is_in_range = True
x1 = -13.500000000000062,       error = -4.336808689942018e-18,       is_in_range = True
x1 = -13.501253930856514,       error = 0.0004999999999999952,       is_in_range = True
x1 = -13.50251582341876,       error = 0.0009999999999999948,       is_in_range = True
x1 = -13.50378583129656,       error = 0.0014999999999999944,       is_in_range = True
x1 = -13.50506411310391,       error = 0.001999999999999994,       is_in_range = True
x1 = -13.506350832689687,       error = 0.0024999999999999935,       is_in_range = True
x1 = -13.507646159383315,       error = 0.002999999999999993,       is_in_range = True
x1 = -13.508950268254644,       error = 0.0034999999999999936,       is_in_range = True
x1 = -13.510263340389939,       error = 0.003999999999999992,       is_in_range = True
x1 = -13.511585563185918,       error = 0.004499999999999991,       is_in_range = True

x2 = -13.912132034355894,       error = -0.005,       is_in_range = True
x2 = -13.910950231097246,       error = -0.0045000000000000005,       is_in_range = True
x2 = -13.909761769633947,       error = -0.004000000000000001,       is_in_range = True
x2 = -13.908566536146088,       error = -0.0035000000000000014,       is_in_range = True
x2 = -13.907364413533182,       error = -0.003000000000000002,       is_in_range = True
x2 = -13.906155281280816,       error = -0.0025000000000000022,       is_in_range = True
x2 = -13.904939015319153,       error = -0.0020000000000000026,       is_in_range = True
x2 = -13.903715487874553,       error = -0.001500000000000003,       is_in_range = True
x2 = -13.902484567313115,       error = -0.0010000000000000035,       is_in_range = True
x2 = -13.901246117974889,       error = -0.0005000000000000039,       is_in_range = True
x2 = -13.899999999999936,       error = -4.336808689942018e-18,       is_in_range = True
x2 = -13.898746069143483,       error = 0.0004999999999999952,       is_in_range = True
x2 = -13.897484176581239,       error = 0.0009999999999999948,       is_in_range = True
x2 = -13.89621416870344,       error = 0.0014999999999999944,       is_in_range = True
x2 = -13.89493588689609,       error = 0.001999999999999994,       is_in_range = True
x2 = -13.893649167310311,       error = 0.0024999999999999935,       is_in_range = True
x2 = -13.892353840616684,       error = 0.002999999999999993,       is_in_range = True
x2 = -13.891049731745355,       error = 0.0034999999999999936,       is_in_range = True
x2 = -13.88973665961006,       error = 0.003999999999999992,       is_in_range = True
x2 = -13.888414436814081,       error = 0.004499999999999991,       is_in_range = True
```

3 Нахождение машинного нуля (№ 1.6, 1.7)

3.1 Формулировка задания

Вычислить значения машинного нуля, машинной бесконечности, машинного эпсилон в режимах одинарной, двойной и расширенной точности на двух алгоритмических языках. Сравнить результаты.

3.2 Теория

Машинная бесконечность $X_\infty = 2^n$, где n - первое натуральное число, при котором происходит переполнение.

Машинный ноль $X_0 = 2^{-n}$, где n - первое натуральное число, при котором 2^{-n} совпадает с нулем.

Машинная эпсилон $\varepsilon_M = 2^{-n}$, где n - наибольшее натуральное число, при котором сумма вычисленного значения $1 + 2^{-n}$ еще больше 1.

3.3 Код на Python

```
import numpy as np


def print_zero(my_type):
    k = 0
    num = my_type(1)
    while num != 0:
        num = my_type(num / 2)
        k += 1
    print(my_type, f"машинный ноль = 2^-{k}")


def print_infinity(my_type):
    k = 0
    num = my_type(1)
    while num != np.inf:
        num = my_type(num * 2)
        k += 1
    print(my_type, f"машинная бесконечность = 2^{k}")


def print_epsilon(my_type):
    k = 0
    num = my_type(1)
    while my_type(1.) + num > my_type(1.):
        num = my_type(num / 2)
        k += 1
    print(my_type, f"машинное эпсилон = 2^-{k}")


for my_type in [np.single, np.double, np.longdouble]:
```

```
    print_zero(my_type)
    print_infinity(my_type)
    print_epsilon(my_type)
    print()
```

3.4 Код на C++

```cpp
#include <iostream>
#include <cmath>
#include <unordered_map>
#include <string>

template <typename T>
std::string print_type() {
    std::unordered_map<std::string, std::string> m_types;
    m_types[typeid(float).name()] = "float";
    m_types[typeid(double).name()] = "double";
    m_types[typeid(long double).name()] = "long double";
    return m_types[typeid(T).name()];
}

template <typename T>
void print_zero() {
    int k = 0;
    T num = 1;
    while (num != static_cast<T>(0)) {
        num /= 2;
        k += 1;
    }
    std::cout << print_type<T>() << " машинный ноль = 2^-" << k << std::endl;
}

template <typename T>
void print_infinity() {
    int k = 0;
    T num = 1;
    while (num < std::numeric_limits<T>::max()) {
        num *= 2;
        k += 1;
    }
    std::cout << print_type<T>() << " машинная бесконечность = 2^" << k << std::endl;
}

template <typename T>
void print_epsilon() {
    int k = 0;
    T num = 1;
    while (static_cast<T>(1) + num > static_cast<T>(1)) {
```

```cpp
        num /= 2;
        k += 1;
    }
    std::cout << print_type<T>() << " машинное эпсилон = 2^-" << k << std::endl;
}

int main() {

    print_zero<float>();
    print_infinity<float>();
    print_epsilon<float>();
    std::cout << std::endl;

    print_zero<double>();
    print_infinity<double>();
    print_epsilon<double>();
    std::cout << std::endl;

    print_zero<long double>();
    print_infinity<long double>();
    print_epsilon<long double>();
    std::cout << std::endl;

    return 0;
}
```

3.5 Вывод кода на Python

```
<class 'numpy.float32'> машинный ноль = 2^-150
<class 'numpy.float32'> машинная бесконечность = 2^128
<class 'numpy.float32'> машинное эпсилон = 2^-24

<class 'numpy.float64'> машинный ноль = 2^-1075
<class 'numpy.float64'> машинная бесконечность = 2^1024
<class 'numpy.float64'> машинное эпсилон = 2^-53

<class 'numpy.longdouble'> машинный ноль = 2^-1075
<class 'numpy.longdouble'> машинная бесконечность = 2^1024
<class 'numpy.longdouble'> машинное эпсилон = 2^-53
```

3.6 Вывод кода на C++

```
float машинный ноль = 2^-150
float машинная бесконечность = 2^128
float машинное эпсилон = 2^-24

double машинный ноль = 2^-1075
double машинная бесконечность = 2^1024
```

```
double машинное эпсилон = 2^-53

long double машинный ноль = 2^-1075
long double машинная бесконечность = 2^1024
long double машинное эпсилон = 2^-53
```

3.7 Сравнение результатов

Видно, что и машинный ноль, и машинное эпсилон, и машинная бесконечность совпадают для Python и для C++.

4 Расчет погрешности матрицы (№ 1.9.2)

4.1 Формулировка задания

Для матрицы A решить вопрос о существовании обратной матрицы в следующих случаях:

1. элементы матрицы заданы точно;

2. элементы матрицы заданы приближенно с относительной погрешностью $\alpha = 0.05$ и $\beta = 0.1$

$$A = \begin{pmatrix} 30 & 34 & 19 \\ 31.4 & 35.4 & 20 \\ 24 & 28 & 13 \end{pmatrix}$$

4.2 Теория

Пусть элементы матрицы определителя заданы приближенно с относительной погрешностью δ. Пусть элементы матрицы обозначены через a_{ij}. Тогда каждый элемент матрицы теперь уже не равен конкретному значению, а может принимать любое значение из отрезка $[a_{ij}(1-\delta), a_{ij}(1+\delta)]$ если $a_{ij} > 0$, и из отрезка $[a_{ij}(1+\delta), a_{ij}(1-\delta)]$ если $a_{ij} < 0$.

Множество всех возможных значений элементов матрицы представляет собой замкнутое ограниченное множество в 9-мерном пространстве. Сам определитель является непрерывной и дифференцируемой функцией 9 переменных – элементов матрицы a_{ij}. По теореме Вейерштрасса эта функция достигает на указанном множестве своего наибольшего и наименьшего значений M и m.

Если отрезок $[m, M]$ не содержит точку 0, то это означает, что при всевозможных допустимых значениях элементов матрицы a_{ij} определитель не обращается в 0. Если же точка 0 принадлежит отрезку $[m, M]$, такое утверждение будет неправомерным. Будет иметь место неопределённость.

Нахождению значений m и M помогают следующие рассуждения. Как функция своих аргументов (элементов матрицы a_{ij}) определитель обладает таким свойством (принцип максимума): эта функция достигает своего наибольшего и наименьшего значений всегда на границе области. Более того, можно доказать, что эти значения достигаются в точках, координаты которых имеют вид $a_{ij}(1 \pm \delta)$. Таких точек $2\hat{}9 = 512$. В каждой из них следует вычислить определитель, а затем выбрать из полученных значений самое большое и самое маленькое. Это и будут числа M и m.

4.3 Код на Python

```python
import numpy as np
from itertools import product


def is_inverse_matrix_exist(matrix: np.ndarray):
    matrix_det = np.linalg.det(matrix)
    print(f'Определитель без погрешности {matrix_det}')
    print()


def is_inverse_matrix_exist_with_error(matrix: np.ndarray, delta: float):
    metrix_dets = []
```

```
    for sign in list(product([-1, 1], repeat=9)):
        new_matrix = matrix * (1 + np.array(sign).reshape(3, 3) * delta)
        metrix_dets.append(np.linalg.det(new_matrix))

    min_det = np.min(metrix_dets)
    max_det = np.max(metrix_dets)
    print(f'Минимальное значение определителя = {min_det}')
    print(f'Максимальное значение определителя = {max_det}')

    if min_det < 0 < max_det:
        print(f"При относительной погрешности {delta} определитель может обратиться в 0")
    else:
        print(f"При относительной погрешности {delta} определитель не может обратиться в 0")
    print()


matrix = np.array([[30, 34, 19],
                   [31.4, 35.4, 20],
                   [24, 28, 13]])

is_inverse_matrix_exist(matrix)
is_inverse_matrix_exist_with_error(matrix, 0.05)
is_inverse_matrix_exist_with_error(matrix, 0.1)
```

4.4 Вывод программы

Определитель без погрешности 9.600000000000069

Минимальное значение определителя = -984.8728000000016
Максимальное значение определителя = 1027.8990000000008
При относительной погрешности 0.05 определитель может обратиться в 0

Минимальное значение определителя = -2965.2384
Максимальное значение определителя = 3032.2959999999985
При относительной погрешности 0.1 определитель может обратиться в 0

4.5 Выводы

Если значения заданы точно, то определитель не равен 0, а следовательно существует обратная матрица.

Если лементы матрицы заданы приближенно с относительной погрешностью $\alpha = 0.05$ и $\beta = 0.1$, то определитель можно равняться нулю, значит обратная матрица можно не существовать.