

**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) | 2 |
| 1.1 | Формулировка задания | 2 |
| 1.2 | Найдем сумму ряда аналитически | 2 |
| 1.3 | Найдем погрешности | 2 |
| 1.3.1 | Код на Python | 2 |
| 1.3.2 | Посчитанные данные | 4 |
| 1.3.3 | Визуализация посчитанных данных | 4 |
| 2 | Квадратное уравнение (№ 1.5.2) | 5 |
| 2.1 | Формулировка задания | 5 |
| 2.2 | Нахождение погрешности | 5 |
| 2.3 | Код на Python | 6 |
| 2.4 | Результат работы программы | 6 |
| 3 | Нахождение машинного нуля (№ 1.6, 1.7) | 7 |
| 3.1 | Формулировка задания | 7 |
| 3.2 | Теория | 7 |
| 3.3 | Код на Python | 7 |
| 3.4 | Код на C++ | 8 |
| 3.5 | Вывод кода на Python | 9 |
| 3.6 | Вывод кода на C++ | 9 |
| 3.7 | Сравнение результатов | 10 |
| 4 | Расчет погрешности матрицы (№ 1.9.2) | 11 |
| 4.1 | Формулировка задания | 11 |
| 4.2 | Теория | 11 |
| 4.3 | Код на Python | 11 |
| 4.4 | Вывод программы | 12 |
| 4.5 | Выводы | 12 |

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 $N = 10, 10^2, 10^3, 10^4, 10^5$.

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

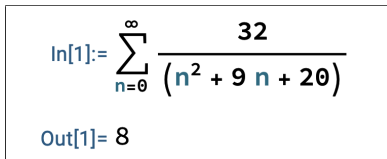
3. Вычислить погрешности для каждого N .

4. Вычислить количество значащих цифр для каждого N .

5. Построить гистограмму зависимости верных цифр результата от N .

1.2 Найдём сумму ряда аналитически

Посчитаем значение предела в Wolfram:



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

1.3 Найдём погрешности

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

1.3.1 Код на Python

```

1  import pandas as pd
2
3
4  def calculateNum(n: int):
5      return 32 / (n ** 2 + 9 * n + 20)
6
7
8  def calculateSum(N: int):
9      s = 0
10     for n in range(N):
11         s += calculateNum(n)

```

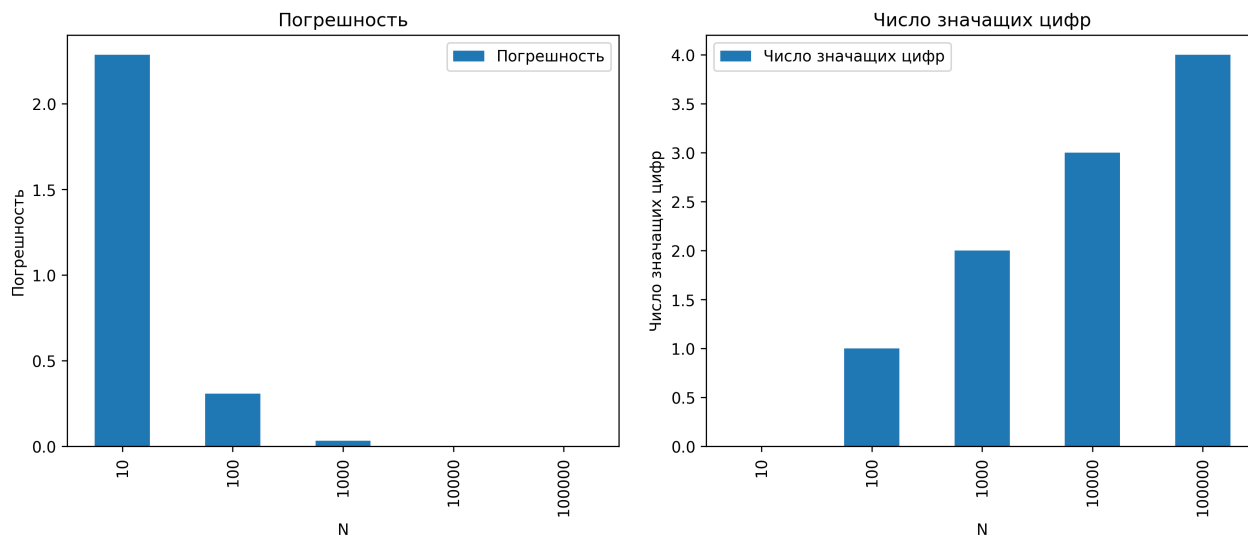
1.3 Найдем погрешности РАСЧЕТ ПОГРЕШНОСТИ ЧАСТИЧНЫХ СУММ РЯДА (№ 1.1.8)

```
12     return s
13
14
15 def calculateData():
16     N_list = [int(10 ** k) for k in range(1, 6)]
17
18     sums = {}
19     errors = {}
20     digits = {}
21     exact_sum = 8
22
23     for N in N_list:
24         sums[N] = calculateSum(N)
25         errors[N] = abs(sums[N] - exact_sum)
26
27     for N in N_list:
28         digits[N] = 0
29         for i in range(0, 50):
30             order_of_error = 1 / (10 ** i)
31             if errors[N] <= order_of_error:
32                 digits[N] += 1
33             else:
34                 break
35
36     sums_list = []
37     errors_list = []
38     digits_list = []
39     for N in N_list:
40         sums_list.append(sums[N])
41         errors_list.append(errors[N])
42         digits_list.append(digits[N])
43
44     pd.options.display.float_format = '{:,.8f}'.format
45     df = pd.DataFrame({
46         'N': N_list,
47         "Значение": sums_list,
48         "Погрешность": errors_list,
49         "Число значащих цифр": digits_list
50     })
51     return df
52
53
54 df = calculateData()
55 print(df)
56 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 обозначениях:

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

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:

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

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

Найдем погрешности корней при разных погрешностях c :

2.3 Код на Python

```

1 import numpy as np
2
3 eq = np.array([1, 27.4, 187.65])
4
5 for error in [1 / (10**k) for k in range(1, 21)]:
6     x = np.roots(eq + np.array([0, 0, error]))
7     x1 = x[0]
8     x2 = x[1]
9     print(f"x1 = },\t{x2 = },\t{error = }")

```

2.4 Результат работы программы

```

x1 = (-13.7+0.24494897427828197j), x2 = (-13.7-0.24494897427828197j), error = 0.1
x1 = -13.87320508075684, x2 = -13.526794919243159, error = 0.01
x1 = -13.897484176581239, x2 = -13.50251582341876, error = 0.001
x1 = -13.899749843554309, x2 = -13.500250156445688, error = 0.0001
x1 = -13.899974998437234, x2 = -13.500025001562767, error = 1e-05
x1 = -13.899997499984318, x2 = -13.50000250001568, error = 1e-06
x1 = -13.899999749999795, x2 = -13.500000250000204, error = 1e-07
x1 = -13.899999974999915, x2 = -13.500000025000084, error = 1e-08
x1 = -13.899999997499963, x2 = -13.500000002500036, error = 1e-09
x1 = -13.899999999749967, x2 = -13.500000000250031, error = 1e-10
x1 = -13.89999999974925, x2 = -13.500000000025073, error = 1e-11
x1 = -13.8999999999745, x2 = -13.500000000002547, error = 1e-12
x1 = -13.89999999999652, x2 = -13.500000000000346, error = 1e-13
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-14
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-15
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-16
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-17
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-18
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-19
x1 = -13.89999999999936, x2 = -13.500000000000062, error = 1e-20

```

3 Нахождение машинного нуля (№ 1.6, 1.7)

3.1 Формулировка задания

Вычислить значения машинного нуля, машинной бесконечности, машинного эpsilon 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 эpsilon $\varepsilon_M = 2^{-n}$, где n - наибольшее натуральное число, при котором сумма вычисленного значения $1 + 2^{-n}$ еще больше 1.

3.3 Код на Python

```

1  import numpy as np
2
3
4  def print_zero(my_type):
5      k = 0
6      num = my_type(1)
7      while num != 0:
8          num = my_type(num / 2)
9          k += 1
10     print(my_type, f"машинный ноль = 2^{-k}")
11
12
13  def print_infinity(my_type):
14      k = 0
15      num = my_type(1)
16      while num != np.inf:
17          num = my_type(num * 2)
18          k += 1
19     print(my_type, f"машинная бесконечность = 2^{k}")
20
21
22  def print_epsilon(my_type):
23      k = 0
24      num = my_type(1)
25      while my_type(1.) + num > my_type(1.):
26          num = my_type(num / 2)
27          k += 1
28     print(my_type, f"машинное эpsilon = 2^{-k}")
29
30
31  for my_type in [np.single, np.double, np.longdouble]:

```



```
32 print_zero(my_type)
33 print_infinity(my_type)
34 print_epsilon(my_type)
35 print()
```

3.4 Код на C++

```
1  #include <iostream>
2  #include <cmath>
3  #include <unordered_map>
4  #include <string>
5
6  template <typename T>
7  std::string print_type() {
8      std::unordered_map<std::string, std::string> m_types;
9      m_types[typeid(float).name()] = "float";
10     m_types[typeid(double).name()] = "double";
11     m_types[typeid(long double).name()] = "long double";
12     return m_types[typeid(T).name()];
13 }
14
15 template <typename T>
16 void print_zero() {
17     int k = 0;
18     T num = 1;
19     while (num != static_cast<T>(0)) {
20         num /= 2;
21         k += 1;
22     }
23     std::cout << print_type<T>() << " машинный ноль = 2^-" << k << std::endl;
24 }
25
26 template <typename T>
27 void print_infinity() {
28     int k = 0;
29     T num = 1;
30     while (num < std::numeric_limits<T>::max()) {
31         num *= 2;
32         k += 1;
33     }
34     std::cout << print_type<T>() << " машинная бесконечность = 2^" << k << std::endl;
35 }
36
37 template <typename T>
38 void print_epsilon() {
39     int k = 0;
40     T num = 1;
41     while (static_cast<T>(1) + num > static_cast<T>(1)) {
```

```

42     num /= 2;
43     k += 1;
44 }
45 std::cout << print_type<T>() << " машинное эpsilon = 2^- " << k << std::endl;
46 }
47
48 int main() {
49
50     print_zero<float>();
51     print_infinity<float>();
52     print_epsilon<float>();
53     std::cout << std::endl;
54
55     print_zero<double>();
56     print_infinity<double>();
57     print_epsilon<double>();
58     std::cout << std::endl;
59
60     print_zero<long double>();
61     print_infinity<long double>();
62     print_epsilon<long double>();
63     std::cout << std::endl;
64
65     return 0;
66 }

```

3.5 Вывод кода на Python

```

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

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

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

```

3.6 Вывод кода на C++

```

float машинный ноль = 2^-150
float машинная бесконечность = 2^128
float машинное эpsilon = 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 эpsilon, 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 $2^9 = 512$. 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

```

1  import numpy as np
2  from itertools import product
3
4
5  def is_inverse_matrix_exist(matrix: np.ndarray):
6      matrix_det = np.linalg.det(matrix)
7      print(f'Определитель без погрешности {matrix_det}')
8      print()
9
10
11  def is_inverse_matrix_exist_with_error(matrix: np.ndarray, delta: float):
12      matrix_dets = []

```

```

13     for sign in list(product([-1, 1], repeat=9)):
14         new_matrix = matrix * (1 + np.array(sign).reshape(3, 3) * delta)
15         metrix_dets.append(np.linalg.det(new_matrix))
16
17     min_det = np.min(metrix_dets)
18     max_det = np.max(metrix_dets)
19     print(f'Минимальное значение определителя = {min_det}')
20     print(f'Максимальное значение определителя = {max_det}')
21
22     if min_det < 0 < max_det:
23         print(f"При относительной погрешности {delta} определитель может обратиться в 0")
24     else:
25         print(f"При относительной погрешности {delta} определитель не может обратиться в 0")
26     print()
27
28
29 matrix = np.array([[30, 34, 19],
30                   [31.4, 35.4, 20],
31                   [24, 28, 13]])
32
33 is_inverse_matrix_exist(matrix)
34 is_inverse_matrix_exist_with_error(matrix, 0.05)
35 is_inverse_matrix_exist_with_error(matrix, 0.1)

```

4.4 Вывод программы

Определитель без погрешности 9.6000000000000069

Минимальное значение определителя = -984.87280000000016

Максимальное значение определителя = 1027.89900000000008

При относительной погрешности 0.05 определитель может обратиться в 0

Минимальное значение определителя = -2965.2384

Максимальное значение определителя = 3032.29599999999985

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