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

Образовательная программа
«Прикладная математика»

ОТЧЕТ
по лабораторной работе № 1

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

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

1.3 Найдем частичные суммы ряда и их погрешности

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

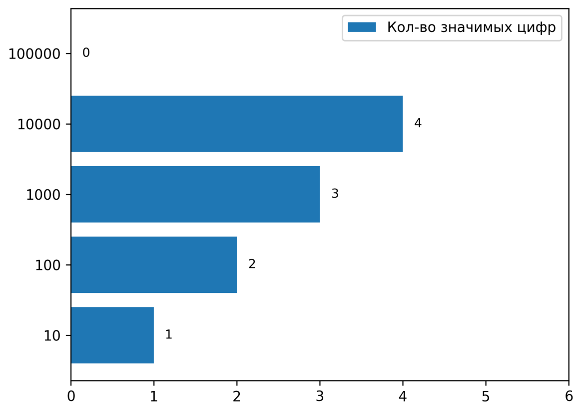
1.3.1 Код на Python

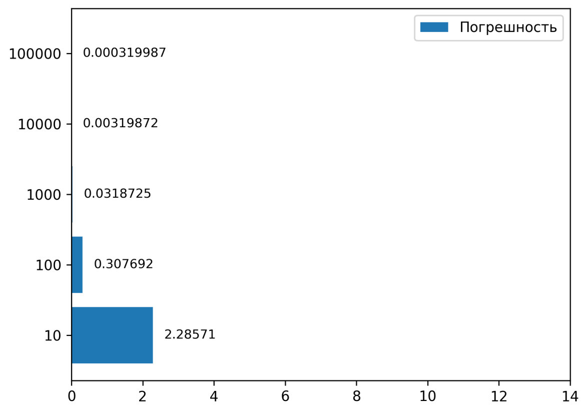
```python
from collections import defaultdict
from prettytable import *


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

```

```python

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

    sums = {}
    errors = {}
    digits = defaultdict(int)
    exact_sum = 8

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

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

    # for N in N_list:
    #     print(f"{sums[N]}\t±\t{errors[N]} \tc {digits[N]} значащих цифр, {N = }")

    table = PrettyTable()
    table.field_names = ["N", "Значение", "Погрешность", "Число значащих цифр"]
    table.align = 'l'
    for N in N_list:
        table.add_row([N, sums[N], errors[N], digits[N]])
    print(table)

    return {
        "sums": sums,
        "errors": errors,
        "digits": digits
    }
```

1.4 Посчитанные данные

```
+--------+-------------------+-----------------------+---------------------+
| N      | Значение          | Погрешность           | Число значащих цифр |
+--------+-------------------+-----------------------+---------------------+
| 10     | 5.714285714285714 | 2.2857142857142856    | 0                   |
| 100    | 7.69230769230769  | 0.30769230769231015   | 1                   |
| 1000   | 7.968127490039844 | 0.03187250996015578   | 2                   |
| 10000  | 7.9968012794882295 | 0.003198720511770503  | 3                   |
| 100000 | 7.999680012799457 | 0.00031998720054282614 | 4                   |
+--------+-------------------+-----------------------+---------------------+
```





2 Расчет погрешности матрицы (№ 1.9.2)

2.1 Формулировка

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 $\beta = 0.1$

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

2.2 Теория

ДОПИСАТЬ ТЕОРИЮ, СКАЗАТЬ О ТЕОРЕМЕ

2.3 Код на Python

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


```

```python
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)
```

2.4 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
```

2.5 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.

3 Нахождение машинnного нуля (№ 1.6, 1.7)

3.1 Формулировка

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

3.2 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()
```

3.3 Код на C++

```
#include <iostream>
#include <cmath>

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

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

template <typename T>
void print_epsilon() {
    int k = 0;
    T num = static_cast<T>(1);
    while (static_cast<T>(1.) + num > static_cast<T>(1.)) {
        num = static_cast<T>(num / 2);
        k += 1;
    }
    std::cout << typeid(T).name() << " 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;
}
```

3.4 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
```

3.5 Вывод кода на C++

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

d машинный ноль = 2^-1075
d машинная бесконечность = 2^1024
d машинное эпсилон = 2^-53

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

3.6 Сравнение результатов

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