Правительство Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего профессионального образования
"Национальный исследовательский университет
"Высшая школа экономики"

Московский институт электроники и математики Национального исследовательского университета "Высшая школа экономики"

Департамент прикладной математики

ОТЧЕТ
По лабораторной работе №1
на тему
«Теория Погрешностей и машинная арифметика»

| ФИО студента | Номер группы | Дата |
|---|---|---|
| Ткаченко Никита Андреевич | БПМ211 | 04.02.2024 |

Москва – 2024 г.

## 1. Погрешность суммы ряда

**Задача 1.1.** Дан ряд $\sum_{n=0}^{\infty} a_n$. Найти сумму ряда аналитически. Вычислить значения частичных сумм ряда

$S_N = \sum a_n$ и найти величину погрешности при значениях $N = 10, 10^2, 10^3, 10^4, 10^5$.

$$\frac{24}{7(n^2+8n+15)}$$

а) Посчитаем сумму ряда аналитически, при помощи математического инструмента wolfram mathematica:

$$\sum_{n=1}^{\infty} \frac{24}{7(n^2+8n+15)} = \frac{27}{35} \approx 0.77143$$

б) Найдем частичные суммы ряда и погрешности:

```
exact_value = 0.77143

def an(n: int):
    return 24 / (7 * (n ** 2 + 8 * n + 15))
```

```
def count_correct_significant_digits(error, value):
    if error > value:
        return 0

    error_string = str(format(float(error), 'f'))

    counter = 0
    for digit_error in error_string:
        if digit_error.isdigit():
            err = float(digit_error)
            if err <= 1:
                counter += 1
            else:
                break
    return counter
```

```
def calculate_error(N: int, exact_value: float):
    partial_sum = 0
    for i in range(1, N):
```

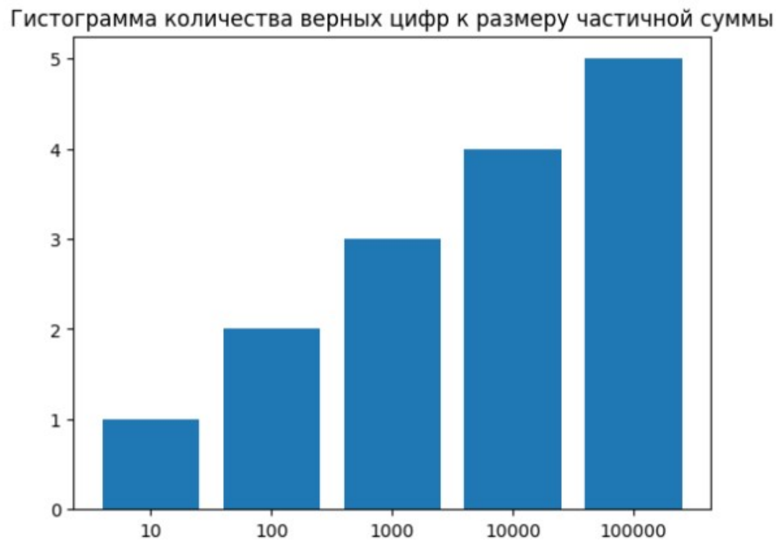
```
        partial_sum += an(i)
    error = np.abs(exact_value - partial_sum)

    correct_numbers = count_correct_significant_digits(error,
exact_value)
    return partial_sum, error, correct_numbers
```

```
M_array = []
N_array = []
for N in [10, 10 ** 2, 10 ** 3, 10 ** 4, 10 ** 5]:
    data = calculate_error(N, exact_value)
    M_array.append(data[2])
    N_array.append(N)
    print(f"S({N}) = {data[0]}, d({N}) = {data[1]}, M = {data[2]}")
```

Вывод программы:

```
S(10) = 0.5171114599686029, d(10) = 0.2543185400313971, M = 1
S(100) = 0.7383015043209216, d(100) = 0.033128495679078385, M = 2
S(1000) = 0.7680119572982819, d(1000) = 0.0034180427017180826, M = 3
S(10000) = 0.771085834242873, d(10000) = 0.0003441657571269241, M = 4
S(100000) = 0.771394286914239, d(100000) = 3.5713085761002183e-05, M = 5
```

Гистограмма:



## 2. Погрешность для диагональных элементов матрицы

**Задача 1.2.** Дана матрица $A = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}$. В каждый из диагональных элементов матрицы $A$ по очереди внести погрешность в 1%. Как изменился определитель матрицы $A$? Указать количество верных цифр и вычислить величину относительной погрешности определителя в каждом случае.

| | | | |
|---|---|---|---|
| 1.2.6 | 3 | 1 | 13 |
| | 5 | 3 | 15 |
| | 11 | 5 | 40 |

Посчитаем относительную и абсолютную погрешности для 6 случаев внесения погрешности в диагональные элементы:

```
delta = 0.01
a1, a2, a3, a4, a5 = sp.symbols("a1 a2 a3 a4 a5")
matrix_a = sp.Matrix([
    [a1, 1, a5],
    [5, a2, 15],
    [a4, 5, a3]
])
print(f"Exact determinant: {matrix_a.det().subs([(a1, 3), (a2, 3), (a3, 40), (a4, 11), (a5, 13)])}")

exact_determinant = matrix_a.det().subs([(a1, 3), (a2, 3), (a3, 40), (a4, 11), (a5, 13)])

for case in [1 + delta, 1 - delta]:
    for aii, i in zip([a1, a2, a3, a4, a5], [(0, 0), (1, 1), (2, 2), (0, 2), (2, 0)]):
        new_matrix = matrix_a.copy()
        new_matrix[i[0], i[1]] = new_matrix[i[0], i[1]] * case
        determinant = new_matrix.det()

        derivative = sp.diff(determinant, aii)

        relative_error = ((sp.Abs(derivative) * sp.Abs(aii)) / sp.Abs(determinant))  * delta
        absolute_error = relative_error * sp.Abs(aii)
```

```
    significant_digits =
count_correct_significant_digits(absolute_error.subs([(a1, 3), (a2, 3),
(a3, 40), (a4, 11), (a5, 13)]), abs(exact_determinant))

    print(f"{aii}: {case}; Relative error =
{relative_error.subs([(a1, 3), (a2, 3), (a3, 40), (a4, 11), (a5, 13)])},
Absolute error = {absolute_error.subs([(a1, 3), (a2, 3), (a3, 40), (a4,
11), (a5, 13)])}, True numbers: {significant_digits}")
```

Вывод программы:

*Exact determinant: -4*
*a1: 1.01; Relative error = 0.514528301886792, Absolute error = 1.54358490566037, True numbers: 1*
*a2: 1.01; Relative error = 0.148592750533046, Absolute error = 0.445778251599139, True numbers: 1*
*a3: 1.01; Relative error = 0.673333333333332, Absolute error = 26.9333333333333, True numbers: 0*
*a4: 1.01; Relative error = 0.532321428571420, Absolute error = 5.85553571428562, True numbers: 0*
*a5: 1.01; Relative error = 0.163087349397589, Absolute error = 2.12013554216865, True numbers: 0*
*a1: 0.99; Relative error = 0.249813084112150, Absolute error = 0.749439252336449, True numbers: 1*
*a2: 0.99; Relative error = 0.206374622356501, Absolute error = 0.619123867069502, True numbers: 1*
*a3: 0.99; Relative error = 0.282857142857143, Absolute error = 11.3142857142857, True numbers: 0*
*a4: 0.99; Relative error = 0.877398648648672, Absolute error = 9.65138513513539, True numbers: 0*
*a5: 0.99; Relative error = 0.733161764705920, Absolute error = 9.53110294117697, True numbers: 0*

## 3. Нахождение машинных чисел

**Задача 1.7.** Вычислить значения машинного нуля, машинной бесконечности, машинного эпсилон в режимах одинарной , двойной и расширенной точности на двух алгоритмических языках. Сравнить результаты.

Код, который считает значения машинного нула, бесконечности и эпсюлон, на Python:

```
def machine_zero(type):
  k = 0
  value = type(1)
  while value != 0:
    value = type(value / 2)
    k += 1
  print(f"{type.__name__}: Машинный ноль = 2^-{k}")


def machine_infinity(type):
  k = 0
  value = type(1)
  while value != np.inf:
    value = type(value * 2)
```

```
        k += 1
    print(f"{type.__name__}: Машинная бесконечность = 2^{k}")


def machine_epsilon(type):
    k = 0
    value = type(1)
    while type(1.) + value > type(1.):
        value = type(value / 2)
        k += 1
    print(f"{type.__name__}: Машинное эпсилон = 2^-{k}")


for my_type in [np.single, np.double, np.longdouble]:
    machine_zero(my_type)
    machine_infinity(my_type)
    machine_epsilon(my_type)
    print()
```

Вывод программы:

*float32: Машинный ноль = 2^-150*
*float32: Машинная бесконечность = 2^128*
*float32: Машинное эпсилон = 2^-24*

*float64: Машинный ноль = 2^-1075*
*float64: Машинная бесконечность = 2^1024*
*float64: Машинное эпсилон = 2^-53*

*longdouble: Машинный ноль = 2^-16446*
*longdouble: Машинная бесконечность = 2^16384*
*longdouble: Машинное эпсилон = 2^-64*

Код на C++:

```
#include <iostream>

template <typename type>
void machine_zero() {
    int k = 0;
    type value = static_cast<type>(1);
    while (value != static_cast<type>(0)) {
        value = static_cast<type>(value / 2);
        k += 1;
    }
```

```
    std::cout << typeid(type).name() << " Машинный ноль = 2^-"
<< k << std::endl;
}

template <typename type>
void machine_infinity() {
    int k = 0;
    type value = static_cast<type>(1);
    while (value + static_cast<type>(1) != value) {
        value = static_cast<type>(value * 2);
        k += 1;
    }
    std::cout << typeid(type).name() << " Машинная
бесконечность = 2^" << k << std::endl;
}

template <typename type>
void machine_epsilon() {
    int k = 0;
    type value = static_cast<type>(1);
    while (value + static_cast<type>(1.) > static_cast<type>(1.)) {
        value = static_cast<type>(value / 2);
        k += 1;
    }
    std::cout << typeid(type).name() << " Машинное эпсилон =
2^-" << k << std::endl;
}

int main() {
    machine_zero<float>();
    machine_infinity<float>();
    machine_epsilon<float>();
    std::cout << std::endl;

    machine_zero<double>();
    machine_infinity<double>();
    machine_epsilon<double>();
    std::cout << std::endl;

    machine_zero<long double>();
    machine_infinity<long double>();
```

```
    machine_epsilon<long double>();
    std::cout << std::endl;

    return 0;
}
```

Вывод программы:

*f Машинный ноль = 2^-150*
*f Машинная бесконечность = 2^24*
*f Машинное эпсилон = 2^-24*

*d Машинный ноль = 2^-1075*
*d Машинная бесконечность = 2^53*
*d Машинное эпсилон = 2^-53*

*e Машинный ноль = 2^-16446*
*e Машинная бесконечность = 2^64*
*e Машинное эпсилон = 2^-64*

Код на Java:

```java
public class Main {
    public static void machine_zero_singleprecision() {
        int k = 0;
        float value = 1.0f;
        while (value != 0f) {
            value /= 2.0f;
            k += 1;
        }
        System.out.printf("float: Машинный ноль = 2^-%d%n", k);
    }

    public static void machine_zero_doubleprecision() {
        int k = 0;
        double value = 1.0;
        while (value != 0.0) {
            value /= 2.0;
            k += 1;
        }
        System.out.printf("double: Машинный ноль = 2^-%d%n", k);
    }

    public static void machine_infinity_singleprecision() {
        int k = 0;
```

```java
        float value = 1.0f;
        while (value + 1.0f != value) {
            value *= 2.0f;
            k += 1;
        }
        System.out.printf("float: Машинная бесконечность = 2^%d
%n", k);
    }

    public static void machine_infinity_doubleprecision() {
        int k = 0;
        double value = 1.0;
        while (value + 1.0 != value) {
            value *= 2.0;
            k += 1;
        }
        System.out.printf("double: Машинная бесконечность = 2^
%d%n", k);
    }

    public static void machine_epsilon_singleprecision() {
        int k = 0;
        float value = 1.0f;
        while (1.0f + value > 1.0f) {
            value /= 2.0f;
            k += 1;
        }
        System.out.printf("float: Машинное эпсилон = 2^-%d%n", k);
    }

    public static void machine_epsilon_doubleprecision() {
        int k = 0;
        float value = 1.0f;
        while (1.0 + value > 1.0) {
            value /= 2.0;
            k += 1;
        }
        System.out.printf("float: Машинное эпсилон = 2^-%d%n", k);
    }
```

```
    public static void main(String[] args) {
        machine_zero_singleprecision();
        machine_zero_doubleprecision();
        machine_infinity_singleprecision();
        machine_infinity_doubleprecision();
        machine_epsilon_singleprecision();
        machine_epsilon_doubleprecision();
    }
}
```

Вывод программы:

*float: Машинный ноль = 2^-150*
*double: Машинный ноль = 2^-1075*
*float: Машинная бесконечность = 2^24*
*double: Машинная бесконечность = 2^53*
*float: Машинное эпсилон = 2^-24*
*float: Машинное эпсилон = 2^-53*

Вывод: Машинные нули, бесконечности и эпсюлон совпадают между тремя языками для одинаковых типов данных.

## 4. Погрешность матрицы.

**Задача 1.9.** 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 a) $\delta = \alpha\%$ и b) $\delta = \beta\%$. Найти относительную погрешность результата.

| | | | | | |
|---|---|---|---|---|---|
| 1.9.3 | 3 | 1 | 13 | | |
| | 13.4 | 11.4 | 23 | 0.05 | 0.1 |
| | 5 | 3 | 15 | | |

Определитель является непрерывной и дифференцируемой функцией 9 переменных - элементов матрицы Aij. По теореме Вейерштрасса эта функция достигает на множестве [aij * ( 1 + дельта ) ; aij* (1 - дельта ) ] своего наибольшего и наименьшего значений M и m. Если отрезок [m, M] не содержит нуля, значит при любом значении из множества определитель не принимает значения ноль (т.е. матрица обратима), и существует значение погрешности такое, что определитель равен нулю иначе.

Код на Python:

```
from itertools import product

def inverse_matrix_check(matrix):
```

```
    print(f'Без погрешности: detA = {np.linalg.det(matrix)}\n')


def inverse_matrix_check_error(matrix, delta):
    determinants = []
    for j in list(product([-1, 1], repeat=9)):
        determinants.append(
            np.linalg.det(
                matrix * (1 + delta * np.array(j).reshape(3, 3))
            )
        )

    min_determinant, max_determinant = np.min(determinants),
np.max(determinants)
    print(f'Минимальный определитель = {min_determinant}')
    print(f'Максимальный определитель = {max_determinant}')

    if min_determinant < 0 < max_determinant:
        print(f"С погрешностью {delta}: определитель может быть
0\n")
    else:
        print(f"С погрешностью {delta}: определитель не может
быть 0\n")


matrix = np.array([
    [3, 1, 13],
    [13.4, 11.4, 23],
    [5, 3, 15]
])
inverse_matrix_check(matrix)
inverse_matrix_check_error(matrix, 0.05)
inverse_matrix_check_error(matrix, 0.1)
```

Вывод программы:

*Без погрешности: detA = 1.5999999999999945*

*Минимальный определитель = -129.94380000000007*
*Максимальный определитель = 120.61580000000012*
*С погрешностью 0.05: определитель может быть 0*

*Минимальный определитель = -274.01439999999997*
*Максимальный определитель = 233.52960000000022*
*С погрешностью 0.1: определитель может быть 0*

Вывод: Изначальная матрица обратима (определитель не равен 0), однако при значениях относительной погрешности 0.05 или 0.1 сущесвует точка, в которой определитель равен 0, значит матрица не является обратимой.