Федеральное государственное автономное образовательное учреждение высшего образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
"ВЫСШАЯ ШКОЛА ЭКОНОМИКИ"

Московский институт электроники и математики имени А. Н. Тихонова
Программа "Прикладная математика"

Нигматуллин Роман Максимович

Лабораторная работа

Теория погрешностей и машинная арифметика

3 курс, группа БПМ203

Преподаватель:
Брандышев Петр Евгеньевич

Москва, 2021 г.

Содержание

1 Расчет частичных сумм ряда

1.1 Формулировка задачи

Дан ряд, надо найти сумму аналитически как предел частичных сумм, затем вычислить частичные суммы в зависимости от N и сравнить абсолютную погрешность и кол-во верных цифр в частичной сумме.

$$S(N) = \sum_{0}^{N} a_n$$

$$a_n = \frac{32}{n^2 + 5n + 6}$$

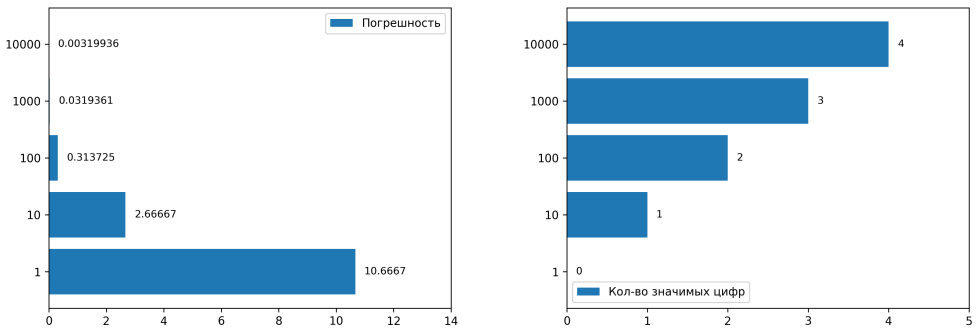
1.2 Код на Python

```
#!/usr/bin/env python
# coding: utf-8

import matplotlib.pyplot as plt
from collections import defaultdict
```

```python
def calculate_nth(n):
    return 32 / (n ** 2 + 5 * n + 6)


sums = {}
errors = {}
n_digits = defaultdict(int)
for N in (10 ** p for p in range(5)):
    s = 0.
    true_s = 16.
    for i in range(N):
        s += calculate_nth(i)
    sums[N] = s
    errors[N] = true_s - s
    i = -1
    while i < 30:
        if errors[N] <= 10 ** (-i):
            n_digits[N] += 1
            i += 1
        else:
            break
    print(f'Sum, Error, Number of digits for n = {N}')
    print(sums[N], errors[N], n_digits[N])

keys = list(map(str, sums.keys()))
errors = list(errors.values())
n_digits = list(n_digits.values())

bars = plt.barh(keys, errors, label='Погрешность')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0, 14)
plt.legend()
plt.savefig('plots/series_error.png', dpi=300)
plt.close()

bars = plt.barh(keys, n_digits, label='Кол-во значимых цифр')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0, 5)
plt.legend()
```

```
plt.savefig('plots/series_n_digits.png', dpi=300)
```

1.3 Графики точности результата



(a) Абсолютная погрешность (b) Кол-во значащих цифр

Рис. 1: Точность в зависимости от N

2 Квадратное уравнение

2.1 Формулировка задачи

Дано квадратное уравнение. Предполагается, что один из коэффициентов уравнения (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.

$$x^2 + bx + c = 0$$

2.2 Код на Python

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np

coefficients = np.array([1, -30.9, 238.7])
```

```
errors = (10**p for p in range(-15, -1))

for error in errors:
    actual_coefficients = coefficients + np.array([0, 0, error])
    print('Error:', error)
    print('Roots:', np.roots(actual_coefficients))
```

3 Машинная точность

3.1 Формулировка задачи

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

3.2 Код на Python

```
#!/usr/bin/env python
# coding: utf-8

import numpy as np
import warnings

displayer = {
    'float16': np.float16,
    'float32': np.float32,
    'float64': np.float64,
}


def experiment(type_name):
    k: int = 0
    num_type = displayer[type_name]
    num = num_type(1)
    while num != 0:
        num = num_type(num / 2)
        k += 1
    print(type_name, "zero is 2^-" + str(k))
    k = 0
    num = num_type(1)
```

```
    while num != np.inf:
        num = num_type(num * 2)
        k += 1
    print(type_name, "infinity is 2^" + str(k))
    k = 0
    num = num_type(1)
    while 1. + num > 1.:
        num = num_type(num / 2)
        k += 1
    print(type_name, "epsilon is 2^-" + str(k))


with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    for typename in displayer.keys():
        print()
        experiment(typename)
```

3.3 Код на C++

```
#include <iostream>
#include <limits>

int main() {
    int k = 0;
    float a = 1;
    double b = 1;
    long double c = 1;

    std::cout << "Float" << std::endl;
    while(a != 0){
        a /= 2;
        k++;
    }
    std::cout << "zero = 2^-" << k << std::endl;
    k = 0;
    a = 1;
    while(a < std::numeric_limits<float>::max()){
        a *= 2;
        k++;
    }
```

```
std::cout << "infinity = 2^" << k << std::endl;
k = 0;
a = 1;
while(1 + a > 1){
    a /= 2;
    k++;
}
std::cout << "epsilon = 2^-" << k << std::endl;

std::cout << "Double" << std::endl;
k = 0;
while(b != 0){
    b /= 2;
    k++;
}
std::cout << "zero = 2^-" << k << std::endl;
k = 0;
b = 1;
while(b < std::numeric_limits<double>::max()){
    b *= 2;
    k++;
}
std::cout << "infinity = 2^" << k << std::endl;
k = 0;
b = 1;
while(1 + b > 1){
    b /= 2;
    k++;
}
std::cout << "epsilon = 2^-" << k << std::endl;

std::cout << "Long Double" << std::endl;
k = 0;
while(c != 0){
    c /= 2;
    k++;
}
std::cout << "zero = 2^-" << k << std::endl;
k = 0;
c = 1;
while(c < std::numeric_limits<long double>::max()){
```

```
        c *= 2;
        k++;
    }
    std::cout << "infinity = 2^" << k << std::endl;
    k = 0;
    c = 1;
    while(1 + c > 1){
        c /= 2;
        k++;
    }
    std::cout << "epsilon = 2^-" << k << std::endl;

    return 0;
}
```

4 Вычисления с ограниченной разрядностью

4.1 Формулировка задачи

Составить программу, моделирующую вычисления на ЭВМ с ограниченной разрядностью m. Решить задачу о вычислении суммы ряда для случая $n = 10000$, используя эту программу. Составить график зависимости погрешности от количества разрядов $m = \{4, 5, 6, 7, 8\}$.

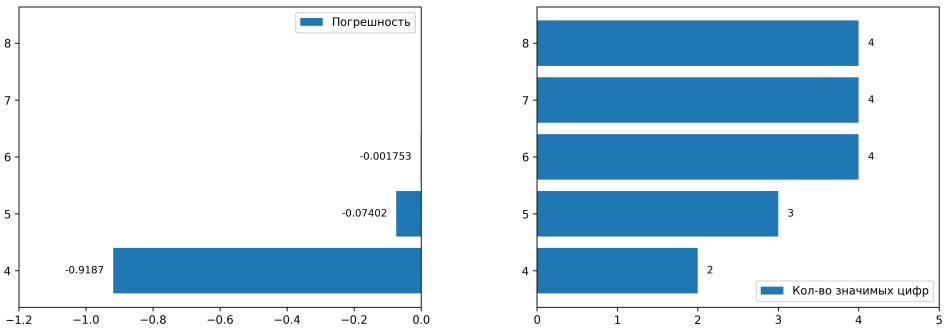
4.2 Код на Python

```
#!/usr/bin/env python
# coding: utf-8

import matplotlib.pyplot as plt

from math import ceil
from collections import defaultdict


def fixed_bit_depth(x, m):
    p = 0
    while x >= 1:
        x /= 10
        p += 1
```

```
    x *= 10**p
    return ceil(x * 10**(m - p)) / 10**(m - p)


def calculate_nth(n, m):
    true_val = 32 / (n ** 2 + 5 * n + 6)
    return fixed_bit_depth(true_val, m)


true_s = 16.
N = 10000
sums, errors, n_digits = {}, {}, defaultdict(int)

for bit_depth in range(4, 9):
    s = 0.
    for i in range(N):
        s += calculate_nth(i, bit_depth)
    sums[bit_depth] = s
    errors[bit_depth] = true_s - s
    i = -1
    while i < 30:
        if abs(errors[bit_depth]) <= 10 ** (-i):
            n_digits[bit_depth] += 1
            i += 1
        else:
            break
    print(f'Sum, Error, Number of digits for bit_depth = {bit_depth}')
    print(sums[bit_depth], errors[bit_depth], n_digits[bit_depth])

keys = list(map(str, sums.keys()))
errors = list(errors.values())
n_digits = list(n_digits.values())

bars = plt.barh(keys, errors, label='Погрешность')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(-1.2, 0)
plt.legend()
plt.savefig('plots/series_fixed_error.png', dpi=300)
plt.close()

bars = plt.barh(keys, n_digits, label='Кол-во значимых цифр')
```

```
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0, 5)
plt.legend()
plt.savefig('plots/series_fixed_n_digits.png', dpi=300)
```

4.3 Графики точности результата



(a) Абсолютная погрешность (b) Кол-во значащих цифр

Рис. 2: Точность в зависимости от разрядности