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 Аналитический расчет суммы ряда

Выпишем формулу суммы ряда:

$$S = \sum_{0}^{\infty} \frac{32}{n^2 + 5n + 6}$$

Разделим знаменатель выражения на простые многочлены и на разные слагаемые:

$$a_n = \frac{32}{(n+2)(n+3)} = \frac{A}{n+2} + \frac{B}{n+3}$$

$$a_n = \frac{A}{n+2} + \frac{B}{n+3} = \frac{An + 3A + Bn + 2B}{(n+2)(n+3)}$$

Вычислим значения в методе неопределенных коэффициентов:

$$3A + 2B = 32, A = -B$$

$$A = 32, B = -32$$

Распишем первые несколько членов ряда и заметим, что соседние сокращаются:

$$S(N) = \frac{32}{0+2} - \frac{32}{0+3} + \frac{32}{1+2} - \frac{32}{1+3} ... + \frac{32}{n+2} - \frac{32}{n+3}$$

Перейдем к пределу и выведем аналитическое значение предела суммы:

$$S(N) = \frac{32}{0+2} - \frac{32}{n+3}$$

$$S = \lim_{s \to \infty} S_N = 16 - 0 = 16$$

1.3 Код на Python

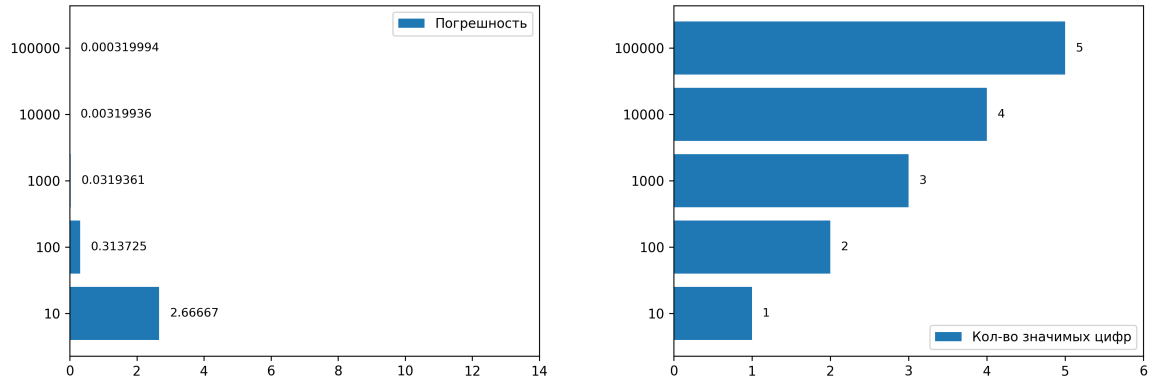
```python
#!/usr/bin/env python
# coding: utf-8

import matplotlib.pyplot as plt
from collections import defaultdict


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(1, 6)):
    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, 6)
plt.legend()
plt.savefig('plots/series_n_digits.png', dpi=300)
```

1.4 Результат работы программы

```
Sum, Error, Number of digits for n = 10
13.333333333333332 2.666666666666668 1
Sum, Error, Number of digits for n = 100
15.686274509803921 0.3137254901960791 2
Sum, Error, Number of digits for n = 1000
15.968063872255465 0.031936127744534915 3
Sum, Error, Number of digits for n = 10000
15.996800639872015 0.0031993601279847184 4
Sum, Error, Number of digits for n = 100000
15.999680006399592 0.00031999360040835256 5
```

1.5 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.

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

$$b = -30.9$$

$$c^* = 238.7$$

2.2 Теоретическая оценка погрешности

Выпишем формулу уравнения и погрешности переменных и общую формулу для функций:

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

$$c^* = c \pm \Delta c$$

$$\bar{\Delta} x = |x| \bar{\delta} x$$

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

Оценим погрешность корня уравнения в зависимости от коэффициента c:

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

$$\bar{\delta} x_{1,2} = \left| \frac{c}{x_{1,2}} \frac{\partial x_{1,2}}{\partial c} \right| \times \bar{\delta} c$$

Запишем общую формулу корня квадратного уравнения и вычислим производную:

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\frac{\partial x_{1,2}}{\partial c} = -\frac{1}{\sqrt{b^2 - 4ac}} = -10$$

Рассчитаем теоретические погрешности корней в зависимости от относительной погрешности c:

$$\bar{\delta} x_1 = \frac{30.9}{15.5} \times 10 \times \bar{\delta} c = \ 19.935 \times \bar{\delta} c$$

$$\bar{\delta} x_2 = \frac{30.9}{15.4} \times 10 \times \bar{\delta} c = \ 20.065 \times \bar{\delta} c$$

2.3 Код на Python

```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))
```

2.4 Результат работы программы

```
Error: 1e-15
Roots: [15.5 15.4]
Error: 1e-14
Roots: [15.5 15.4]
Error: 1e-13
Roots: [15.5 15.4]
Error: 1e-12
Roots: [15.5 15.4]
Error: 1e-11
Roots: [15.5 15.4]
Error: 1e-10
Roots: [15.5 15.4]
Error: 1e-09
Roots: [15.49999999 15.40000001]
Error: 1e-08
Roots: [15.4999999 15.4000001]
Error: 1e-07
Roots: [15.499999 15.400001]
Error: 1e-06
Roots: [15.49999 15.40001]
Error: 1e-05
Roots: [15.4998999 15.4001001]
Error: 0.0001
Roots: [15.49898979 15.40101021]
Error: 0.001
Roots: [15.48872983 15.41127017]
Error: 0.01
Roots: [15.45+0.08660254j 15.45-0.08660254j]
```

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 = {
    'float': np.single,
    'double': np.double,
    'long double': np.longdouble,
}


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 num_type(1.) + num > num_type(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 Результат работы программы

```
float zero is 2^-150
float infinity is 2^128
float epsilon is 2^-24

double zero is 2^-1075
double infinity is 2^1024
double epsilon is 2^-53

long double zero is 2^-1075
long double infinity is 2^1024
long double epsilon is 2^-53
```

3.4 Код на C++

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

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

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

    k = 0;
    while(b != 0){
        b /= 2;
        k++;
    }
    std::cout << "double zero = 2^-" << k << std::endl;
```

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

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

3.5 Результат работы программы

```
float zero = 2^-150
float infinity = 2^128
float epsilon = 2^-24
double zero = 2^-1075
double infinity = 2^1024
double epsilon = 2^-53
long double zero = 2^-1075
long double infinity = 2^1024
long double epsilon = 2^-53
```

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

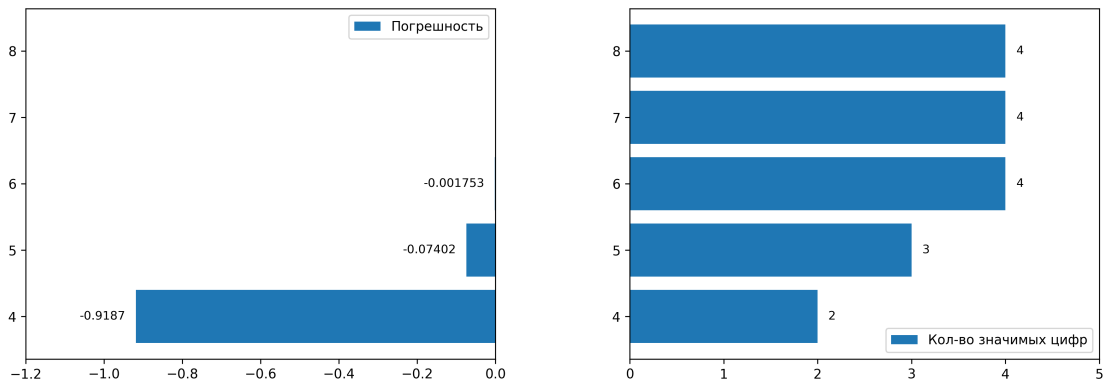
```
#!/usr/bin/env python
# coding: utf-8

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 Результат работы программы

```
Sum, Error, Number of digits for bit_depth = 4
16.91869999999773 -0.918699999997731 2
Sum, Error, Number of digits for bit_depth = 5
16.07401999999661 -0.07401999999661157 3
Sum, Error, Number of digits for bit_depth = 6
16.00175300000028 -0.001753000000281446 4
Sum, Error, Number of digits for bit_depth = 7
15.99730419999936 0.0026958000006391813 4
Sum, Error, Number of digits for bit_depth = 8
15.996850950000152 0.00314904999984833 4
```

4.4 Графики точности результата



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

Рис. 2: Точность в зависимости от разрядности