Правительство Российской федерации
Федеральное государственное автономное образовательное учреждение высшего профессионального образования

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ «ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

ЛАБОРАТОРНАЯ РАБОТА №1

"Теория погрешностей и машинная арифметика"

по численным методам
Вариант 15

№1.1.15, 1.4.4, 1.7, 1.6, 1.8

Выполнила:
Студентка 3 курса бакалавриата
группы БПМ211
Пугач Виктория Павловна

Преподаватель:
Кандидат физико-математических наук,
Брандышев Петр Евгеньевич

Москва, 2024

Содержание

1 Расчет частичных сумм ряда

1.1 Формулировка задачи

Дан числовой ряд.

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

Необходимо найти сумму этого ряда 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.

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

$$a_n = \frac{20}{n^2 + 4n + 3}$$

1.2 Аналитический расчет суммы ряда

Выпишем формулу суммы ряда:

$$S = \sum_{0}^{\infty} \frac{20}{n^2 + 4n + 3}$$

Разделим знаменатель выражения на простые многочлены и на разные слагаемые:

$$a_n = \frac{20}{(n+1)(n+3)} = \frac{A}{n+1} + \frac{B}{n+3}$$

$$a_n = \frac{A}{n+1} + \frac{B}{n+3} = \frac{An + 3A + Bn + B}{(n+2)(n+3)}$$

Вычислим значения в методе неопределенных коэффициентов:

$$3A + B = 20, An + Bn = 0 \Rightarrow A = -B$$

$$A = 10, B = -10$$

Распишем первые несколько членов ряда и заметим, что начиная со второго они сокращаются со слагаемым, идущим через 2 члена суммы:

$$S(N) = \frac{10}{0+1} - \cancel{\frac{10}{0+3}} + \frac{10}{1+1} - \cancel{\frac{10}{1+3}} + \cancel{\frac{10}{2+1}} - \frac{10}{2+3} + \cancel{\frac{10}{3+1}} - \frac{10}{3+3} + ... + \frac{10}{n+1} - \frac{10}{n+3}$$

Останутся слагаемые:

$$S(N) = \frac{10}{0+1} + \frac{10}{1+1} - \frac{10}{n+3} \Rightarrow$$

при переходе к пределу

$$S = \lim_{s \to \infty} S_N = \frac{10}{0+1} + \frac{10}{1+1} - 0 = 10 + 5 = 15$$

1.3 Пределы частичных сумм, найденные с Wolfram Mathematica

При N =10

```
S1 = Sum[20/(n^2 + 4 n + 3), {n, 0, 10}]
1045/78 // N
13.3974
```

При N $= 10^2$

```
S1 = Sum[20/(n^2 + 4 n + 3), {n, 0, 100}]
77770/5253 // N
14.8049
```

При N $= 10^3$

```
S1 = Sum[20/(n^2 + 4 n + 3), {n, 0, 1000}]
7527520/502503 // N
14.98
```

При N $= 10^4$

```
S1 = Sum[20/(n^2 + 4 n + 3), {n, 0, 10000}]
750275020/50025003 // N
14.998
```

При N $= 10^5$

```
S1 = Sum[20/(n^2 + 4 n + 3), {n, 0, 100000}]
75002750020/5000250003 // N
14.9998
```

1.4 Подсчет абсолютной погрешности и верно значащих цифр

Количество верно значащих цифр = количество цифр до первой цифры >1 слева

При N =10 Верно значащих цифр = 1

```
abserr1 = Abs[S - S1]
1.60256
```

При N $= 10^2$ Верно значащих цифр = 2

```
abserr2 = Abs[S - S2]
0.195127
```

При N $= 10^3$ Верно значащих цифр $= 3$

```
abserr3 = Abs[S - S3]
0.0199501
```

При N $= 10^4$ Верно значащих цифр $= 4$

```
abserr4 = Abs[S - S4]
0.0019995
```

При N $= 10^5$ Верно значащих цифр $= 5$

```
abserr5 = Abs[S - S5]
0.000199995
```

In[5]:= (* 2 *)

$$S1 = \sum_{n=0}^{10} \left(\frac{20}{n^2 + 4n + 3}\right)$$

In[6]:= $\frac{1045}{78}$ // N

численное приближение

Out[6]= 13.3974

In[7]:= $S2 = \sum_{n=0}^{10^2} \left(\frac{20}{n^2 + 4n + 3}\right)$

In[8]:= $\frac{77770}{5253}$ // N

численное приближение

Out[8]= 14.8049

In[9]:= $S3 = \sum_{n=0}^{10^3} \left(\frac{20}{n^2 + 4n + 3}\right)$

In[10]:= $\frac{7527520}{502503}$ // N

численное приближение

Out[10]= 14.98

In[11]:= $S4 = \sum_{n=0}^{10^4} \left(\frac{20}{n^2 + 4n + 3}\right)$

In[12]:= $\frac{750275020}{50025003}$ // N

численное приближение

Out[12]= 14.998

In[13]:= $S5 = \sum_{n=0}^{10^5} \left(\frac{20}{n^2 + 4n + 3}\right)$

In[14]:= $\frac{75002750020}{5000250003}$ // N

численное приближение

Out[14]= 14.9998

Код вольфрама

1.5 Представление результатов в виде гистограмм

```
N = list(map(str,[10,100,1000,10000,100000]))
errors = [1.60256, 0.195127, 0.0199501, 0.0019995, 0.000199995]
true_num = [1,2,3,4,5]

bars = plt.barh(N, true_num, label='Кол-во верно значащих цифр')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0, 5.2)
plt.legend()

bars = plt.barh(N, errors, label='Погрешность')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0, 1.7)
plt.legend()
```

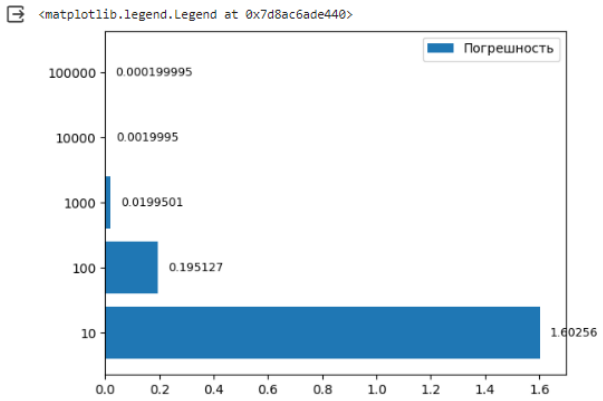

Рис. 1: Абсолютная погрешность

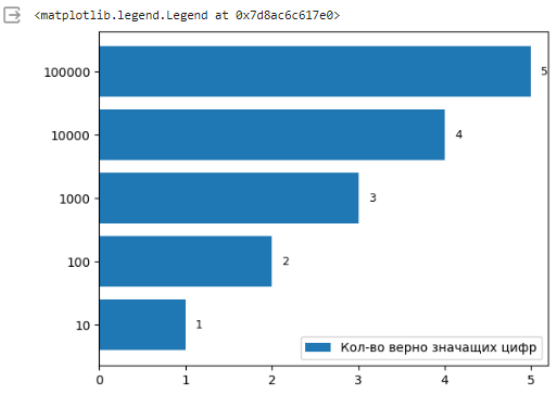

Рис. 2: Количество верно значащих цифр

Как и предполагалось ранее, увеличение числа членов ряда увеличивает число верных цифр в ответе, и уменьшает величину абсолютной погрешности.

2 Ранг Матрицы

2.1 Формулировка задачи

Найти ранг заданной матрицы A

$$A = \begin{pmatrix} 2 & 15 & 22 & 7 \\ 1 & 14.1 & 18.8 & 2.3 \\ 2 & 4 & 9 & 9 \\ -0.4 & 2.5 & 2.1 & -2.4 \end{pmatrix}$$

Затем внести погрешность в 0.1% в
а) элемент a_{11}
б) во все элементы матрицы

И вновь найти ранг матрицы. Объяснить полученные результаты

2.2 Теоретическая оценка влияния погрешности

Ранг заданной матрицы находится при помощи одной функции.

```
matrix = [[2,15,22,7],[1,14.1,18.8,2.3],[2,4,9,9],[-0.4, 2.5,2.1,-2.4]]
print("Ранг матрицы A = ",np.linalg.matrix_rank(matrix))
```

```
output: Ранг матрицы A =  2
```

Иначе дело обстоит в случае, когда элементы задаются с погрешностью. Пусть элементы матрицы обозначены через a_{ij}. Тогда каждый элемент матрицы теперь уже не равен конкретному значению, а может принимать любое значение из отрезка $[a_{ij}(1-\delta); a_{ij}(1+\delta)]$, при $a_{ij} > 0$, или из отрезка $[a_{ij}(1+\delta); a_{ij}(1-\delta)]$, при $a_{ij} < 0$.

Множество всех возможных значений элементов матрицы представляет собой замкнутое ограниченное множество в 16-мерном пространстве (матрица 4*4).

Так как изначально ранг матрицы равен 2, не получится анализировать изменение ранга, основываясь на том, принимает ли определитель значение 0 при различных значениях элементов, так как определитель заданной матрицы уже нулевой.

В таком случае, мы можем воспользоваться теоремой Вейерштрасса и рассмотреть значения ранга матрицы на гранях этого "16-мерного куба". Мы будем рассматривать значения ранга на максимально удаленных ("влево и вправо то есть с прибавленной погрешностью, и с отнятой погрешность) точках для каждого из элементов, и смотреть, какие значения будет принимать ранг матрицы в этих случаях.

2.3 Алгоритмическое решение на Python

Пункт а). С погрешностью задается только первый элемент матрицы

Для первого пункта достаточно рассмотреть два дополнительных случая.

```
matrix_low = [[2*(1-0.001),15,22,7],[1,14.1,18.8,2.3],[2,4,9,9],[-0.4, 2.5,2.1,-2.4]]
print("Ранг матрицы A_low = ",np.linalg.matrix_rank(matrix_low))
matrix_high = [[2*(1+0.001),15,22,7],[1,14.1,18.8,2.3],[2,4,9,9],[-0.4, 2.5,2.1,-2.4]]
print("Ранг матрицы A_high = ",np.linalg.matrix_rank(matrix_high))
```

```
output:
Ранг матрицы A_low =  3
Ранг матрицы A_high =  3
```

Пункт б). С погрешностью задаются все элементы матрицы

Для второго пункта понадобится рассмотреть все варианты матриц (необходимо будет рассмотреть 2^{16} вариантов матриц), и вычислить их ранг.

```
matrix = [[2,15,22,7],[1,14.1,18.8,2.3],[2,4,9,9],[-0.4, 2.5,2.1,-2.4]]
matrix_dets =set()

for i, signs in enumerate(itertools.product([1 + 0.001, 1 - 0.001], repeat=16)):
    delta_matrix = np.array(signs).reshape(4, 4)
    new_matrix = matrix * delta_matrix
    matrix_dets.add(np.linalg.matrix_rank(new_matrix))

print(matrix_dets)
```

```
output: {2, 3, 4}
```

2.4 Выводы по результатам выполнения программы

При задании относительной погрешности первому элементу матрицы, ранг матрицы увеличивается на 1. Однако пр задании относиетльной погрешности всем элементам матрицы, ранг матрицы принимает значения 2, 3 и 4. То есть число линейно независимых строк может как увеличиваться, так и уменьшаться.

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. Сравнить результаты

3.2 Теоретический материал

В ЭВМ для вещественных чисел используется двоичная система счисления и принята форма представления чисел с плавающей точкой $x = \mu * 2^p$.

$\mu = (\gamma_1 * 2^{-1} + \gamma_2 * 2^{-2} + \ldots + \gamma_t * 2^{-t})$. Здесь μ - мантисса,γ_1, γ_2,. . . ,γ_t - двоичные цифры, причем всегда $\gamma_1 = 1$, p - целое число, называемое двоичным порядком. Количество цифр t, отводимое для записи мантиссы, называется разрядностью мантиссы. Диапазон представления чисел в ЭВМ ограничен конечной разрядностью мантиссы, значением числа p. Все представимые на ЭВМ числа удовлетворяют неравенствам:

$$0 < X_0 \leq |x| < X_\infty$$

Где $X_0 = 2^{-(p_max+1)}$, $X_\infty = 2^{(p_max)}$. Все числа по модулю большие X_∞, не представимы на ЭВМ и рассматриваются как машинная бесконечность. Все числа, по модулю меньшие X_0, не отличаются от нуля для ЭВМ, и рассматриваются как машинный нуль.
Машинным эпсилон называется относительная точность ЭВМ, то есть граница относительной погрешности представления чисел в ЭВМ.

Машинное эпсилон определяется разрядностью мантиссы и способом округления чисел, реализованным на конкретной ЭВМ.
Примем следующие способы определения приближенных значений параметров, требуемых в задаче:

1. Положим $X_\infty = 2^n$, где n - первое натуральное число, при котором происходит переполнение.
2. Положим $X_0 = 2^{-m}$, где m - первое натуральное число, при котором 2^{-m} совпадает с нулем.
3. Положим $\epsilon_M = 2^{-k}$, где k - наибольшее натуральное число, при котром $1 + \epsilon_M > 1$. То есть ϵ_M - граница относительной погрешности представления $x^* \approx 1$

3.3 Решение на языке Python

```
for typename in (np.single, np.double, np.longdouble):
    print()

    k: int = 0
    num = typename(1)

    while num != 0:
        num = typename(num / 2)
        k += 1
    print(typename, "zero is 2^-" + str(k))
    k = 0
    num = typename(1)

    while num != np.inf:
        num = typename(num * 2)
        k += 1
    print(typename, "infinity is 2^" + str(k))
    k = 0
    num = typename(1)

    while typename(1.) + num > typename(1.):
        num = typename(num / 2)
        k += 1
    print(typename, "epsilon is 2^-" + str(k))
```

```
output:
<class numpy.float32> zero is 2^-150
<class numpy.float32> infinity is 2^128
<class numpy.float32> epsilon is 2^-24

<class numpy.float64> zero is 2^-1075
<class numpy.float64> infinity is 2^1024
<class numpy.float64> epsilon is 2^-53

<class numpy.float128> zero is 2^-16446
<class numpy.float128> infinity is 2^16384
<class numpy.float128> epsilon is 2^-64
```

3.4 Решение на языке C++

```cpp
#include <iostream>
#include <cmath>
#include <typeinfo>

template <typename T>
void printCharacteristic(const std::string& characteristic) {
    int k = 0;
    T num = static_cast<T>(1);
    if (characteristic == "zero") {
        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;
    } else if (characteristic == "infinity") {
        while (!std::isinf(num)) {
            num = static_cast<T>(num * 2);
            k += 1;
        }
        std::cout << typeid(T).name() << " машинная бесконечность = 2^" << k << std::en
    } else if (characteristic == "epsilon") {
        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;
    } else {
        std::cout << "Неизвестная характеристика" << std::endl;
    }
}

int main() {
    printCharacteristic<float>("zero");
    printCharacteristic<float>("infinity");
    printCharacteristic<float>("epsilon");
    std::cout << std::endl;

    printCharacteristic<double>("zero");
    printCharacteristic<double>("infinity");
    printCharacteristic<double>("epsilon");
    std::cout << std::endl;

    printCharacteristic<long double>("zero");
    printCharacteristic<long double>("infinity");
    printCharacteristic<long double>("epsilon");
    std::cout << std::endl;

    return 0;}
```

```
output:
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^-16446
e машинная бесконечность = 2^16384
e машинное эпсилон = 2^-64
```

Таким образом мы получили все необходимые значения. Интересно, что при выслении при помощи Python и C++ значения совпали.

4 Вычисления с ограниченной разрядностью

4.1 Формулировка задачи

Составить программу, моделирующую вычисления на ЭВМ с ограниченной разрядностью m. Решить задачу 1.1 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

```
def compute_partial_sum(m, n):
    partial_sum = 0
    for i in range(0, n + 1):
        temp = round(20 / round((round(i ** 2,m) + round(4 * i, m) + 3),m),m)
        partial_sum += temp
    return partial_sum

N = 10000  # Количество элементов ряда
S_real = 15
errors = []
for m in range (4,9):
  result = compute_partial_sum(m, N)
  error = np.abs(result - S_real)
  errors.append(error)
  print(f"Частная сумма ряда для n={N} с ограниченной разрядностью {m} равна {result} \

print(errors)
n = list(map(str,[4,5,6,7,8]))
bars = plt.barh(n, errors, label='Погрешность')
plt.bar_label(bars, padding=8, fontsize=9)
plt.xlim(0.0018, 0.03)
plt.legend()
```

```
output:
Частная сумма ряда для n=10000 с ограниченной разрядностью 4 равна 14.9731999999999

Частная сумма ряда для n=10000 с ограниченной разрядностью 5 равна 14.991429999999552

Частная сумма ряда для n=10000 с ограниченной разрядностью 6 равна 14.99728699999822

Частная сумма ряда для n=10000 с ограниченной разрядностью 7 равна 14.997989899999963

Частная сумма ряда для n=10000 с ограниченной разрядностью 8 равна 14.998000439999915

[0.02680000000009919, 0.008570000000448275, 0.002713000001779875, 0.002010100000036985,
```

4.3 График изменения погрешности

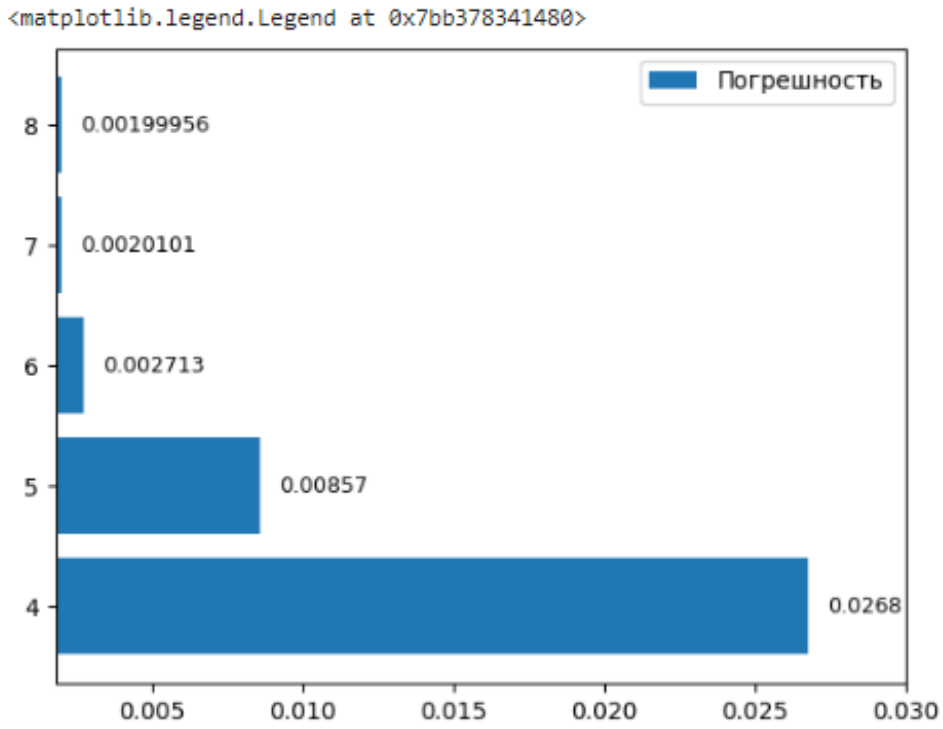

Рис. 3: График зависимости абсолютной погрешности от

4.4 Результат работы программы

Как и следовало предположить, с увеличением разрядности ЭВМ погрешность вычислений уменьшается. Результат становится наиболее приближен к истинному.