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"

Лабораторная работа №1

Теория погрешностей и машинная арифметика

Группа БПМ213
Вариант 8
Номера выполняемых задач: 1.1.8, 1.5.2, 1.7, 1.6, 1.9.2

Выполнила:
Варфоломеева Анастасия Андреевна
Преподаватель:
Токмачев Михаил Геннадьевич

Москва, 2024 г.

1 Расчет частичных сумм ряда

1.1 Формулировка задачи

Дан ряд. Найти сумму ряда S аналитически. Вычислить значения частичных сумм рядаи найти величину погрешности при значениях N = 10, 10^2, 10^2, 10^4, 10^5. Построить гистограмму зависимости верных цифр результата от N.

$$\frac{32}{n^2 + 9n + 20}$$

1.2 Аналитический расчет суммы ряда

Разложим методом неопределенных коэффециентов ряд и в итоге получим следующее:

$$S_N = \sum_0^N \frac{32}{n^2 + 9n + 20} = \sum_0^N \frac{32}{(n+4)(n+5)} = 32 \sum_0^N (\frac{1}{n+4} - \frac{1}{n+5}) =$$

$$= 32(\frac{1}{4} - \frac{1}{5} + ... + \frac{1}{n+4} - \frac{1}{n+5}) = 32(1/4 - \frac{1}{n+5})$$

При переходе к пределу получаем аналитическое решение:

$$S = \lim_{N \to \infty} S_N = 8$$

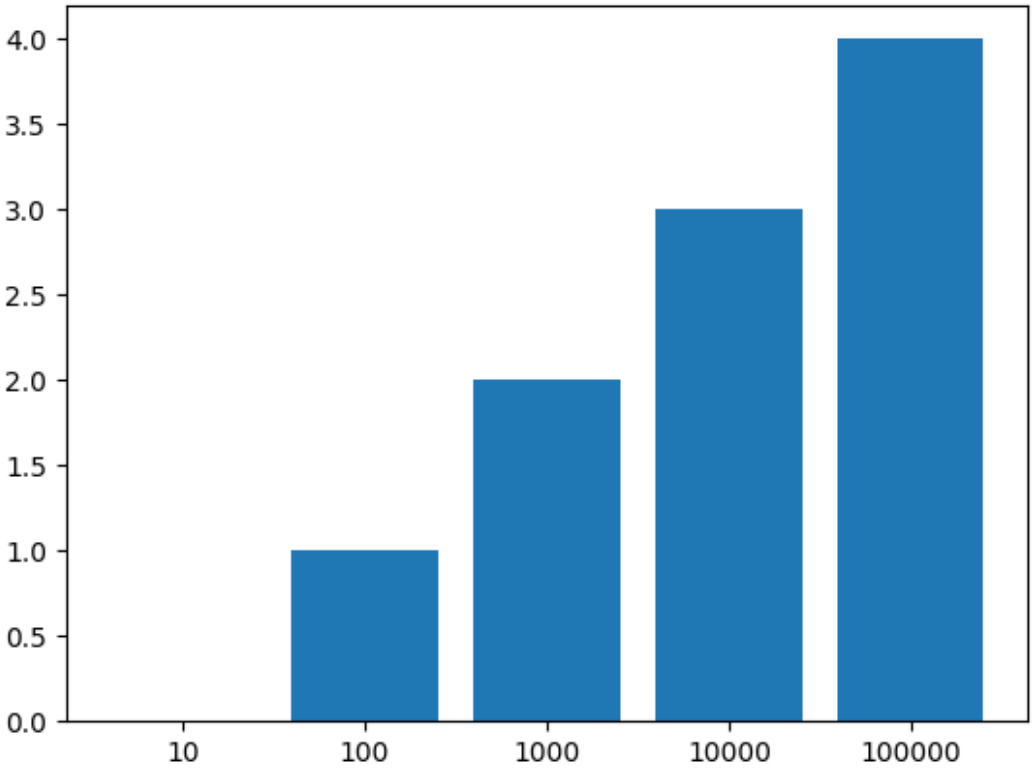
1.3 Код на Python

```
import matplotlib.pyplot as plt

analit_s = 8
S = {}
d = {}
M = dict()
for N in [10, 100, 1000, 10000, 100000]:
    s = 0
    M[N] = 0
    for n in range(N):
        s += 32/(n**2 + 9*n + 20)
    S[N] = s
    d[N] = abs(analit_s - s)
    num = 0
    while num > -20:
        if d[N] <= 10 ** num:
            M[N] += 1
            num -= 1
        else:
            break
    print(f'S({N}): {S[N]}, d({N}): {d[N]}, M({N}): {M[N]} \n--------------------')

bars = plt.bar(list(map(str, S.keys())), list(M.values()))
plt.show()
```

1.4 Результат работы программы

```
S(10): 5.714285714285714, d(10): 2.2857142857142856, M(10): 0
------------------
S(100): 7.69230769230769, d(100): 0.30769230769231015, M(100): 1
------------------
S(1000): 7.968127490039844, d(1000): 0.03187250996015578, M(1000): 2
------------------
S(10000): 7.9968012794882295, d(10000): 0.003198720511770503, M(10000): 3
------------------
S(100000): 7.999680012799457, d(100000): 0.00031998720054282614, M(100000): 4
------------------
```

1.5 Гистограмма точности результата



1.6 Вывод

Как видно из приведенного вычислительного эксперимента, увеличение числа членов ряда в 10 раз по сравнению с предыдущим случаем увеличивает число верных цифр в ответе на 1.

2 Квадратное уравнение

2.1 Формулировка задачи

Дано квадратное уравнение. Предполагается, что один из коэффициентов уравнения (помечен $*$) получен в результате округления. Произвести теоретическую оценку по-

грешностей корней в зависимости от погрешности коэффициента. Вычислить корни уравнения при нескольких различных значениях коэффициента в пределах заданной точности. Сравнить полученные результаты.

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

$$b = 27.4$$

$$c^* = 187.65$$

2.2 Теоретическая оценка погрешности

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

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

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

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

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

Найдем корни для нашего уравнения:

$$x^2 + 27.4x + 187.65 = 0$$

$$x_1 = -13.9$$

$$x_2 = -13.5$$

Оценим погрешность корня уравнения в зависимости от коэффициента c:

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

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

Запишем общую формулу корня квадратного уравнения и вычисления производной:

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

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

Теперь вычислим производную для нашего уравнения:

$$\frac{\partial x_1}{\partial c} = -\frac{1}{\sqrt{b^2 - 4c}} = -2.5$$

$$\frac{\partial x_2}{\partial c} = \frac{1}{\sqrt{b^2 - 4c}} = 2.5$$

Рассчитаем теоретические погрешности корней в зависимости от относительной погрешности c:

$$\bar{\delta} x_1 = \frac{187.65}{13.9} \times 2.5 \times \bar{\delta} c = \ 33.75 \times \bar{\delta} c$$

$$\bar{\delta} x_2 = \frac{187.65}{13.5} \times 2.5 \times \bar{\delta} c = \ 34.75 \times \bar{\delta} c$$

2.3 Код на Python

```
import numpy as np

true_coefficients = np.array([1, 27.4, 187.65])
errors = [0.1, 10**-2, 10**-3, 10**-4, 10**-5, 10**-6, 10**-7, 10**-8, 10**-9, 10**-10
for err in errors:
    actual_coefficients = true_coefficients + np.array([0, err, 0])
    print('error:', err)
    print("x1={:.5f}, x2={:.5f}".format(*np.roots(actual_coefficients)))
```

2.4 Результат работы программы

```
error: 0.1
x1=-14.93849, x2=-12.56151
error: 0.01
x1=-14.12574, x2=-13.28426
error: 0.001
x1=-13.93223, x2=-13.46877
error: 0.0001
x1=-13.90345, x2=-13.49665
error: 1e-05
x1=-13.90035, x2=-13.49966
error: 1e-06
x1=-13.90003, x2=-13.49997
error: 1e-07
x1=-13.90000, x2=-13.50000
error: 1e-08
x1=-13.90000, x2=-13.50000
error: 1e-09
x1=-13.90000, x2=-13.50000
error: 1e-10
x1=-13.90000, x2=-13.50000
error: 1e-11
x1=-13.90000, x2=-13.50000
error: 1e-12
x1=-13.90000, x2=-13.50000
error: 1e-13
x1=-13.90000, x2=-13.50000
error: 1e-14
x1=-13.90000, x2=-13.50000
error: 1e-15
x1=-13.90000, x2=-13.50000
```

2.5 Вывод

При погрешности большей 10^{-6} значения становятся верными с сточностью до 5 знаков после запятой, хотя при погрешности в 0.1 значения корней далеки от истинных.

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 Код на Python

```
import numpy as np

types =  {
    'float': np.single,
    'double': np.double,
    'long double': np.longdouble
    }
for t in types:
    n_t = types[t]
    c = 0
    n = n_t(1)
    while n != np.inf:
        n = (n*2).astype(n_t)
        c += 1
    print(f'Infinity for {t} is 2^{c}')
    c = 0
    n = n_t(1)
    while n != 0:
        n = (n/2).astype(n_t)
        c += 1
    print(f'Zero for {t} is 2^-{c}')
    c = 0
    n = n_t(1)
    while n_t(1) + n > n_t(1):
        n = (n/2).astype(n_t)
        c += 1
    print(f'Epsilon for {t} is 2^-{c} \n')
```

3.3 Результат работы программы

```
Infinity for float is 2^128
Zero for float is 2^-150
Epsilon for float is 2^-24

Infinity for double is 2^1024
Zero for double is 2^-1075
Epsilon for double is 2^-53

Infinity for long double is 2^16384
Zero for long double is 2^-16446
Epsilon for long double is 2^-64
```

3.4 Код на C++

```
#include<iostream>
#include <limits>
using namespace std;

int main() {

  int c = 0;
  float fl = 1;
  while(fl != numeric_limits<float>::infinity()) {
    fl *= 2;
    c += 1;
  }
  cout << "Infinity for float is 2^" << c << endl;

  c = 0;
  fl = 1;
  while(fl != 0) {
    fl /= 2;
    c += 1;
  }
  cout << "Zero for float is 2^-" << c << endl;

  c = 0;
  fl = 1;
  while(fl + 1 > 1) {
    fl /= 2;
    c += 1;
  }
  cout << "Epsilon for float is 2^-" << c << endl << endl;

  double db = 1;
  c = 0;
  while(db != numeric_limits<double>::infinity()) {
    db *= 2;
    c += 1;
  }
  cout << "Infinity for double is 2^" << c << endl;

  c = 0;
  db = 1;
  while(db != 0) {
    db /= 2;
    c += 1;
  }
  cout << "Zero for double is 2^-" << c << endl;

  c = 0;
  db = 1;
  while(db + 1 > 1) {
```

```
    db /= 2;
    c += 1;
  }
  cout << "Epsilon for double is 2^-" << c << endl << endl;

  long double ldb = 1;
  c = 0;
  while(ldb != numeric_limits<long double>::infinity()) {
    ldb *= 2;
    c += 1;
  }
  cout << "Infinity for long double is 2^" << c << endl;

  c = 0;
  ldb = 1;
  while(ldb != 0) {
    ldb /= 2;
    c += 1;
  }
  cout << "Zero for long double is 2^-" << c << endl;

  c = 0;
  ldb = 1;
  while(ldb + 1 > 1) {
    ldb /= 2;
    c += 1;
  }
  cout << "Epsilon for long double is 2^-" << c << endl << endl;

  return 0;
}
```

3.5 Результат работы программы

```
Infinity for float is 2^128
Zero for float is 2^-150
Epsilon for float is 2^-24

Infinity for double is 2^1024
Zero for double is 2^-1075
Epsilon for double is 2^-53

Infinity for long double is 2^16384
Zero for long double is 2^-16446
Epsilon for long double is 2^-64
```

3.6 Сравнение результатов

Полученные значения для каждого из типов на языке C++ и Python оказались равными и соответсвуют предполагаемым значениям.

4 Существование обратной матрицы

4.1 Формулировка задачи

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 $\delta = \alpha\%$ и $\delta = \beta\%$.
Найти относительную погрешность результата.

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

$$\alpha = 0.05$$

$$\beta = 0.1$$

В нашем случае детерминант матрицы A равен 9.6, следовательно она является невырожденной, а значит выполнено необходимое и достаточное условие существования обратной матрицы → у нее существует обратная.

4.2 Код на Python

```
import numpy as np

A = np.array([[30, 34, 19],
              [31.4, 35.4, 20],
              [24, 28, 13]], dtype=float)
alpha = 0.05
beta = 0.1

print(f'det = {np.linalg.det(A)} => A is invertible\n------------')

mn1 = np.inf
mx1 = -np.inf
for i in range(512):
    new_A = A * np.array([1+(1-2*int(j))*alpha/100 for j in bin(i)[2:].zfill(9)]).resh
    new_det = np.linalg.det(new_A)
    mn1 = min(new_det, mn1)
    mx1 = max(new_det, mx1)

print(f'alpha = {alpha} \n{mn1} <= new det <= {mx1}')
if mn1 <= 0 and 0 <= mx1:
    print('matr is not invertible')
else:
```

```
    print('matr is invertible')
print(f'relative error in det = {(abs(mn1-np.linalg.det(A)) + abs(mx1-np.linalg.det(A)
```

```
mn2 = np.inf
mx2 = -np.inf
for i in range(512):
    new_A = A * np.array([1+(1-2*int(j))*beta/100 for j in bin(i)[2:].zfill(9)]).resha
    new_det = np.linalg.det(new_A)
    mn2 = min(new_det, mn2)
    mx2 = max(new_det, mx2)
print(f'beta = {beta} \n{mn2} <= new det <= {mx2}')
if mn2 <= 0 and 0 <= mx2:
    print('matr is not invertible')
else:
    print('matr is invertible')
print(f'relative error in det = {(abs(mn2-np.linalg.det(A)) + abs(mx2-np.linalg.det(A)
```

4.3 Результат работы программы

```
det = 9.600000000000176 => A is invertible
------------
alpha = 0.05
3.6042036012009144 <= new det <= 15.589803598798975
matr is invertible
relative error in det = 0.6242499998748875
------------
beta = 0.1
-2.397585590399123 <= new det <= 21.573614390399435
matr is not invertible
relative error in det = 1.2484999989999022
------------
```

4.4 Вывод

При погрешности $\alpha = 0.05\%$ отрезок не содержит 0, следовательно у мат- рицы существует обратная. При погрешности $\beta = 0.1\%$ отрезок содержит 0, следовательно возникает неопределенность и вычислить обратную матрицу не представлется возможным. В определителях с относительными погрешностями α и β получаем 62% и 125% относительной погрешности.