Лаборатная работа, вариант №24

Теория погрешностей и машинная арифметика

Задания: 1.1.24, 1.5.6, 1.6, 1.7, 1.9.6

Студент:
Паршина Софья Романовна
3 курс, группа БПМ213

Содержание

1 Частичные суммы ряда(1.1.24)

1.1 Формулировка задачи

Дан ряд

$$\sum_{n=0}^{\infty} a_n,$$

надо найти сумму аналитически как предел частичных сумм S(N), затем вычислить частичные суммы в зависимости от N=10, ..., 10^5 и сравнить абсолютную погрешность и кол-во верных цифр в частичной сумме.

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

$$a_n = \frac{96}{n^2 + 9n + 20}$$

1.2 Аналитический расчет суммы ряда

Выпишем формулу суммы ряда:

$$S = \sum_{n=0}^{\infty} \frac{96}{n^2 + 9n + 20}$$

$$S_N = \sum_{n=0}^{N} \frac{96}{n^2+9n+20} = \sum_{n=0}^{N} \frac{96}{(n+4)(n+5)} = 96*\sum_{n=0}^{N}(\frac{1}{n+4} - \frac{1}{n+5}) = 96*(\frac{1}{4} - \frac{1}{N+5}),$$

$$S = \lim_{n\to\infty} S_N = 24 - 0 = 24$$

Ответ:

$$S = \sum_{n=0}^{\infty} \frac{96}{n^2+9n+20} = 24$$

1.3 Код на Python

```
# -*- coding: utf-8 -*-
"""partial sums of a series.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SJOn0Mh6wBP7DNpXf_ruBxG0BjaFP0nf
"""

import matplotlib.pyplot as plt
from collections import defaultdict

def Summa(n):
    return 96/(n ** 2 + 9 * n + 20)

N_values = [10, 100, 10**3, 10**4, 10**5, 10**6]
sums = {}
errors = {}
right_figure = defaultdict(int) #??
for N in N_values:
    s = 0.
    true_s = 24.
    for i in range(N):
        s += Summa(i)
    sums[N] = s
    errors[N] = abs(s - true_s)

    i = 1
    while i < 30:
        if errors[N] <= 10 ** (i):
            right_figure[N] += 1
            i -= 1
        else:
            break

    print(f'For N = {N}: \nsum = {sums[N]} \nabs error = {errors[N]} \nnumders of rig

keys = list(map(str, sums.keys()))
bars = plt.barh(keys, list(right_figure.values()))
```

```
plt.bar_label(bars, padding=1, fontsize=10)
plt.xlim(0, 6)
plt.xlabel("Кол-во значимых цифр")
plt.ylabel("Кол-во чисел ряда")
plt.legend()
plt.savefig('right_figure.png', dpi=300)
```

1.4 Результат работы программы

```
For N = 10:
sum = 17.142857142857142
abs error = 6.857142857142858
numders of right figure = 1

For N = 100:
sum = 23.076923076923062
abs error = 0.9230769230769376
numders of right figure = 2

For N = 1000:
sum = 23.904382470119508
abs error = 0.09561752988049221
numders of right figure = 3

For N = 10000:
sum = 23.990403838464626
abs error = 0.009596161535373682
numders of right figure = 4

For N = 100000:
sum = 23.9990400383994
abs error = 0.0009599616005999678
numders of right figure = 5

For N = 1000000:
sum = 23.99990400038543
abs error = 9.599961456885353e-05
numders of right figure = 6
```

1.5 Графики точности результата

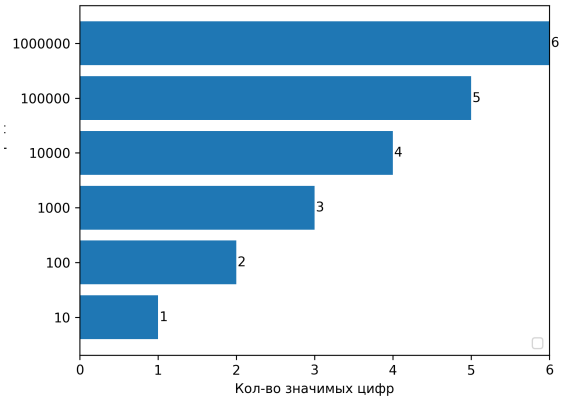

Рис. 1: Точность в зависимости от N

2 Квадратное уравнение(1.5.6)

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.

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

$$b = -3.29$$

$$c^* = 2.706$$

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:

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

$$\frac{\partial x_{1,2}}{\partial c} = -\frac{1}{\sqrt{b^2 - 4ac}} = -100$$

Рассчитаем теоретические погрешности корней в зависимости от относительной погрешности c:

$$\bar{\delta} x_1 = \frac{2.706}{1.65} \times 100 \times \bar{\delta} c = \ 164 \times \bar{\delta} c$$

$$\bar{\delta} x_2 = \frac{2.706}{1.64} \times 100 \times \bar{\delta} c = \ 165 \times \bar{\delta} c$$

2.3 Код на Python

```
# -*- coding: utf-8 -*-
"""quadratic equation.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SJ0n0Mh6wBP7DNpXf_ruBxG0BjaFP0nf
"""

import numpy as np
a = 1
b = -3.29
c =  2.706
errors = (10**n for n in range(-12, 1))

for error in errors:
    now_c = c + error
    print(f'Error = {error}, roots = {np.roots([a, b, now_c])}')
```

2.4 Результат работы программы

```
Error = 1e-12, roots = [1.65 1.64]
Error = 1e-11, roots = [1.65 1.64]
Error = 1e-10, roots = [1.64999999 1.64000001]
Error = 1e-09, roots = [1.6499999 1.6400001]
Error = 1e-08, roots = [1.649999 1.640001]
Error = 1e-07, roots = [1.64998999 1.64001001]
Error = 1e-06, roots = [1.64989898 1.64010102]
Error = 1e-05, roots = [1.64887298 1.64112702]
Error = 0.0001, roots = [1.645+0.00866025j 1.645-0.00866025j]
Error = 0.001, roots = [1.645+0.03122499j 1.645-0.03122499j]
Error = 0.01, roots = [1.645+0.09987492j 1.645-0.09987492j]
Error = 0.1, roots = [1.645+0.31618824j 1.645-0.31618824j]
Error = 1, roots = [1.645+0.9999875j 1.645-0.9999875j]
```

3 Машинная точность(1.7)

3.1 Формулировка задачи

Вычислить значения машинного нуля, машинной бесконечности и машинного эпсилон в режимах одинарной, двойной и расширенной точности на двух алгоритмических языках - Python и C++

3.2 Код на Python

```
# -*- coding: utf-8 -*-
"""parametrs.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SJ0n0Mh6wBP7DNpXf_ruBxG0BjaFP0nf
"""

import numpy as np
import warnings

def float_exp():
    k = 0
    n = np.single(1)
    while n != 0:
      n = np.single(n/2)
      k += 1
    print(f'Float zero: 2^(-{k})')

    k = 0
    n = np.single(1)
    while n!=np.inf:
        n = np.single(2*n)
        k += 1
    print(f'Float infinity: 2^{k}')

    k = 0
    n = np.single(1)
    while np.single(1) + n > np.single(1):
        n = np.single(n/2)
        k += 1
    print(f'Float epsilon: 2^(-{k})\n')

def double_exp():
    k = 0
    n = np.double(1)
    while n != 0:
      n = np.double(n/2)
      k += 1
    print(f'Double zero: 2^(-{k})')
```

```
    k = 0
    n = np.double(1)
    while n!=np.inf:
        n = np.double(2*n)
        k += 1
    print(f'Double infinity: 2^{k}')

    k = 0
    n = np.double(1)
    while np.double(1) + n > np.double(1):
        n = np.double(n/2)
        k += 1
    print(f'Double epsilon: 2^(-{k})\n')


def longdouble_exp():
    k = 0
    n = np.longdouble(1)
    while n != 0:
      n = np.longdouble(n/2)
      k += 1
    print(f'Longdouble zero: 2^(-{k})')

    k = 0
    n = np.longdouble(1)
    while n!=np.inf:
        n = np.longdouble(2*n)
        k += 1
    print(f'Longdouble infinity: 2^{k}')

    k = 0
    n = np.longdouble(1)
    while np.longdouble(1) + n > np.longdouble(1):
        n = np.longdouble(n/2)
        k += 1
    print(f'Longdouble epsilon: 2^(-{k})\n')



warnings.filterwarnings('ignore')
float_exp()
double_exp()
longdouble_exp()
```

3.3 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)

Longdouble zero: 2^(-16446)
Longdouble infinity: 2^16384
Longdouble epsilon: 2^(-64)
```

3.4 Код на C++

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

using namespace std;
void float_exp(){
    int k = 0;
    float n = 1;
    while(n != 0){
        n = n / 2;
        k++;
    }
    cout << "Float zero: 2^(-" << k << ")" << endl;

    k = 0;
    n = 1;
    while(n < numeric_limits<float>::max()){
        n = n * 2;
        k++;
    }
    cout << "Float infinity: 2^" << k << endl;

    k = 0;
    n = 1;
    while(1 + n > 1){
        n = n / 2;
        k++;
    }
    std::cout << "Float epsilon = 2^(-" << k << ")\n" <<std::endl;
}

void double_exp(){
    int k = 0;
    double n = 1;
    while(n != 0){
        n = n / 2;
        k++;
    }
    cout << "Double zero: 2^(-" << k << ")" << endl;
```

```
    k = 0;
    n = 1;
    while(n < numeric_limits<double>::max()){
        n = n * 2;
        k++;
    }
    cout << "Double infinity: 2^" << k << endl;

    k = 0;
    n = 1;
    while(1 + n > 1){
        n = n / 2;
        k++;
    }
    cout << "Double epsilon: 2^(-" << k << ")\n" << endl;
}

void longdouble_exp(){
    int k = 0;
    long double n = 1;
    while(n != 0){
        n = n / 2;
        k++;
    }
    cout << "Longdouble zero: 2^(-" << k << ")" << endl;

    k = 0;
    n = 1;
    while(n < numeric_limits<double>::max()){
        n = n * 2;
        k++;
    }
    cout << "Longdouble infinity: 2^" << k << endl;

    k = 0;
    n = 1;
    while(1 + n > 1){
        n = n / 2;
        k++;
    }
    cout << "Longdouble epsilon: 2^(-" << k << ")" << endl;
}



int main() {
    float_exp();
    double_exp();
    longdouble_exp();
    return 0;
```

```
}
```

3.5 Результат работы программы

```
C:\Users\Choni\CLionProjects\solution\cmake-build-debug\solution.exe
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^(-16446)
Long double infinity: 2^16384
Long double epsilon: 2^(-64)

Process finished with exit code 0
```

Вывод: полученные результаты совпали и соотвествуют тем, которые могли получиться теоретически.

4 Существование обратной матрицы(1.9.6)

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 a) a = 0.001 и b) b = 0.001 Найти относительную погрешность результата.

$$\begin{bmatrix} -3 & -1 & -13 \\ 26,8 & 22,4 & 46 \\ 5 & 3 & 15 \end{bmatrix}$$

4.2 Код на Python

```python
# -*- coding: utf-8 -*-
"""parametrs.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SJOn0Mh6wBP7DNpXf_ruBxG0BjaFPOnf
"""

import numpy as np
import statistics
```

```
M = np.array([[-3., -1., -13.], [26.8, 22.4, 46.], [5., 3., 15.]])
print("Задана матрица:")
for i in range(3):
  print(M[i])
print("\n")
det = np.linalg.det(M)
if (det != 0):
    print(f"Обратная матрица существует, так как определитель не равен нулю, а равен =

Ver = []
N = 9
a = b = 0.001
for i in range(2**N):
    Tmp = M.copy()
    matr = np.full(N, 0)
    for t in range(N):
      if i % 2:
         matr[t] = 1
      i, rest = divmod(i, 2)
    matr = matr.reshape(3, 3)
    #print(matr)

    for l in range(3):
      for k in range(3):
        if matr[l][k] == 1:
          Tmp[l][k] = M[l][k]*(1+a)
        else:
          Tmp[l][k] = M[l][k]*(1-a)
    det_tmp = np.linalg.det(Tmp)
    Ver.append(det_tmp)


otr = [min(Ver),max(Ver)]
print(f"Получили отрезок значений определителя = {otr}. 0 не принадлежит отрезку -> об

average = statistics.mean(Ver)
error = (abs(det - average)) / abs(det)
print(f"Относительная погрешность результата = {error}")
```

4.3 Результат работы программы

```
Задана матрица:
[ -3.  -1. -13.]
[26.8 22.4 46. ]
[ 5.  3. 15.]


Обратная матрица существует, так как определитель не равен нулю,
а равен = -11.200000000000008
```

```
Получили отрезок значений определителя = [-16.11028478879992, -6.2798848112003265].
0 не принадлежит отрезку -> обратная матрица существует

Относительная погрешность результата = 8.564577618536915e-15
```