## Национальный исследовательский университет «МЭИ» (Московский Энергетический Институт)

**Кафедра математического и компьютерного моделирования**

## Численные методы

## Лабораторная работа №1:

## «Теория погрешностей и машинная арифметика»

*Выполнил:* Солонин Е. В. А-14-19

*Преподаватель:* Амосова О. А.

Вариант 52

2021 Москва

**Постановка задачи:**

**Задача 1.1.** Найти значения машинного нуля, машинной бесконечности и машинного эпсилон.

**Теоретический материал:**

ВЭВМ для вещественных чисел используется двоичная система счисления и принята форма представления чисел с плавающей точкой ,

.

Здесь - мантисса ; - двоичные цифры, причем всегда =1, p-целое число, называемое двоичным порядком. Количество *t* цифр, которое отводится для записи мантиссы, называется разрядностью мантиссы. Диапазон представления чисел в ЭВМ ограничен конечной разрядностью мантиссы и значением числа *p*. Все представимые числа на ЭВМ удовлетворяют неравенствам: , где , . Все числа, по модулю большие , не представимы на ЭВМ и рассматриваются как машинная бесконечность. Все числа, по модулю меньшие , для ЭВМ не отличаются от нуля и рассматриваются как машинный нуль. Машинным эпсилон  называется относительная точность ЭВМ, то есть граница относительной погрешности представления чисел в ЭВМ. Покажем, что . Пусть , тогда граница абсолютной погрешности представления этого числа равна . Поскольку , то величина относительной погрешности представления оценивается так:

.

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 , где *n* - первое натуральное число, при котором происходит переполнение.

2. Положим , где *m* – первое натуральное число , при котором  совпадает с нулем.

3. Положим , где *k* – наибольшее натуральное число, при котором сумма вычисленного значения 1+  еще больше 1. Фактически  есть граница относительной погрешности представления числа .

**Код программы:**

import math  
  
*#machine infinity*def inf():  
 inf\_ = 1.  
 while inf\_ \* 2 != math.inf:   
 inf\_ \*= 2  
 return inf\_  
  
*#machine zero*def zero():  
 zer0 = 1.  
 while zer0 \* .5 != 0.:  
 zer0 \*= .5  
 return zer0  
  
*#machine epsilon*def eps():  
 epsilon = 1.  
 while 1. + epsilon \* .5 != 1.:  
 epsilon \*= .5  
 return epsilon \* .5  
  
print(**f'Machine infinity:** {inf():**.2e**}**'**)  
print(**f'Machine zero:** {zero():**.2e**}**'**)  
print(**f'Machine epsilon:** {eps():**.2e**}**'**)

**Результаты выполнения:**

Machine infinity: 8.99e+307

Machine zero: 4.94e-324

Machine epsilon: 1.11e-16

**Анализ результатов:**

Машинная бесконечность

Машинный нуль

Машинное эпсилон

**Постановка задачи:**

**Задача 1.2.** Исследовать поведение погрешности приближения функции F(x) частичными суммами на отрезке [a, b].

ПОРЯДОК РЕШЕНИЯ ЗАДАЧИ:

1. Используя разложения стандартных функций в ряд Тейлора в окрестности нуля, получить

разложение функции F(x) по степеням x.

1. Составить процедуру, вычисляющую частичную сумму N членов ряда S(x, N).
2. Построить графики исходной функции и первых пяти частичных сумм: S(x, 1), …S(x, 5).
3. Составить функции, вычисляющие абсолютную погрешность  и

относительную погрешность . Построить графики погрешностей первых пяти частичных сумм.

1. Определить количество членов ряда N, при котором величина относительной погрешности в средней точке отрезка станет меньше машинного эпсилон. Величину относительной погрешности вычислять как отношение прибавляемого члена к накопленной частичной сумме , взятое по модулю.
2. При найденном значении N построить графики абсолютной погрешности  и относительной погрешности .
3. Составить программу округления вычислений результата до t разрядов мантиссы и произвести расчеты п.4 с учетом округления.
4. Сравнить полученные результаты и составить отчет по задаче.

Вариант 52:

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy  
import matplotlib.pyplot as plt  
import math  
  
#

def f(x):  
 return numpy.exp(-x) + numpy.cos(x)  
# k-ый член в разложении в ряд Тейлора

def taylor\_term(x, k):  
 return (-x)\*\*k / math.factorial(k) + ((-1)\*\*k \* x\*\*(2.\*k)) / math.factorial(2.\*k)

# частичная сумма N членов ряда

def S(x, N):  
 result = 0.  
 for i in range(N):  
 result = result + taylor\_term(x, i)  
 return result

# функции округления вычислений

def S\_rounded\_(x, N):  
 result = 0.  
 for i in range(N):  
 result = ROUND(result + ROUND(taylor\_term(x, i)))  
 return result  
  
def S\_rounded(x, N):  
 result = x.copy()  
 for i in range(len(x)):  
 result[i] = S\_rounded\_(x[i], N)  
 return result

def ROUND(x):  
 return float(numpy.format\_float\_scientific(x, precision=4))

# нахождение абсолютной погрешности для частичной суммы  
  
def absError(s, f):  
 return numpy.abs(s-f)

# нахождение относительной погрешности для частичной суммы  
  
def relError(s, f):  
 return absError(s, f) / numpy.abs(s)  
  
*# machine epsilon*def eps():  
 epsilon = 1.  
 while 1. + epsilon \* .5 != 1.:  
 epsilon \*= .5  
 return epsilon \* .5  
  
  
a = 1.  
b = 7.  
c = (a + b) / 2  
x\_data = numpy.linspace(a, b, 1000)

# построение графиков функции и частичных сумм

fig, axs = plt.subplots(1, 2, figsize=(15, 5))  
axs[0].plot(x\_data, f(x\_data), label=**'$f(x)$'**, color=**"black"**)  
  
line\_styles = ((0, (3, 1, 1, 1, 1, 1)), **'dotted'**, **'dashed'**, **'dashdot'**, (0, (1, 10)))  
for i in range(1,6):  
 axs[0].plot(x\_data, S(x\_data, i), label=**f'$S(x,**{i}**)$'**, ls=line\_styles[i-1])

# построение графиков погрешностей частичных сумм  
  
for i in range(1,6):  
 axs[1].plot(x\_data, absError(S(x\_data, i), f(x\_data)), label=**f'$\Delta S(x,**{i}**)$'**, ls=line\_styles[i-1])  
  
axs[0].legend()  
axs[1].legend()  
axs[0].set(ylim=(-5., 5.))  
axs[1].set(ylim=(0, 5.))  
plt.show()

# определение количества членов ряда N, при котором величина относительной погрешности в средней точке отрезка станет меньше машинного эпсилон

p\_sum = taylor\_term(c, 1)  
n\_t = taylor\_term(c, 2)  
N\_machine\_error = 2  
  
while numpy.abs(n\_t/p\_sum) > eps():  
 p\_sum += n\_t  
 N\_machine\_error += 1  
 n\_t = taylor\_term(c, N\_machine\_error)  
print(N\_machine\_error)

# графики погрешностей для найденного N  
  
fig, axs = plt.subplots(1, 2, figsize=(15, 10))  
  
axs[0].plot(x\_data, absError(S(x\_data, N\_machine\_error), f(x\_data)), label=**f'$\Delta S(x,**{N\_machine\_error}**)$'**, color=**"black"**)  
axs[1].plot(x\_data, relError(S(x\_data, N\_machine\_error), f(x\_data)), label=**f'$\delta S(x,**{N\_machine\_error}**)$'**, color=**"black"**)  
axs[0].legend()  
axs[1].legend()  
axs[0].set(ylim=(0.0, 10E-15))  
axs[1].set(ylim=(0.0, 10E-15))  
plt.show()

# графики погрешностей с учетом функций округления вычислений

fig, axs = plt.subplots(1, 2, figsize=(15, 5))  
  
axs[0].plot(x\_data, absError(S\_rounded(x\_data, N\_machine\_error), f(x\_data)), label=**f'$\Delta S(x,**{N\_machine\_error}**)$'**)  
axs[1].plot(x\_data, relError(S\_rounded(x\_data, N\_machine\_error), f(x\_data)), label=**f'$\delta S(x,**{N\_machine\_error}**)$'**)  
axs[0].legend()  
axs[1].legend()  
plt.show()

**Результаты выполнения:**

Графики функции, частичных сумм(слева); абсолютные погрешности этих сумм(справа):

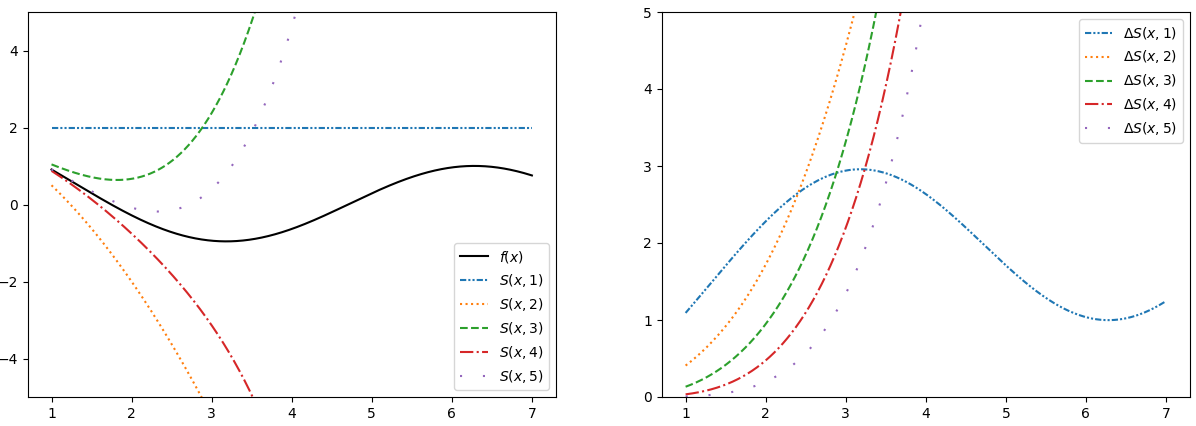
****

График абсолютной погрешности частичной суммы N = 32 членов ряда(слева); относительной(справа):

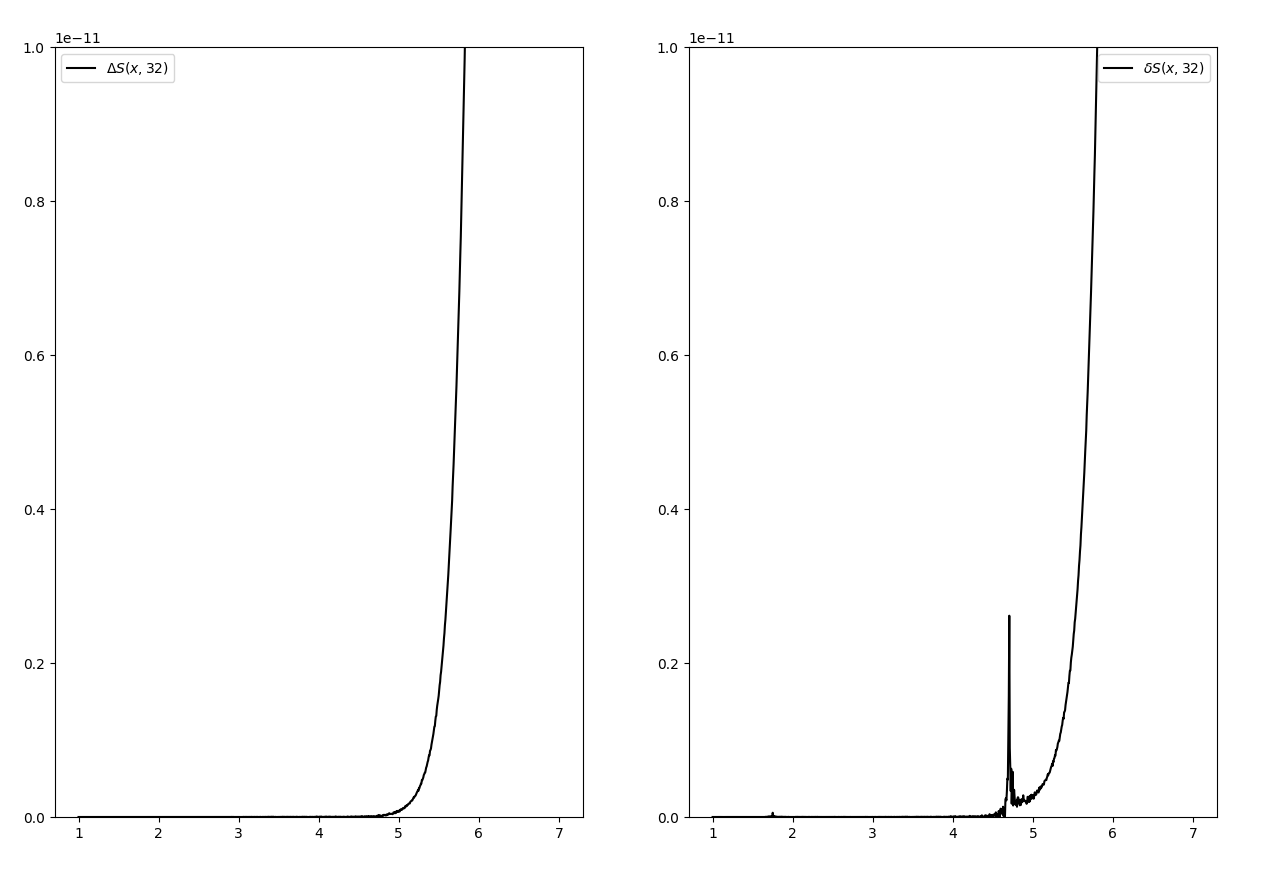
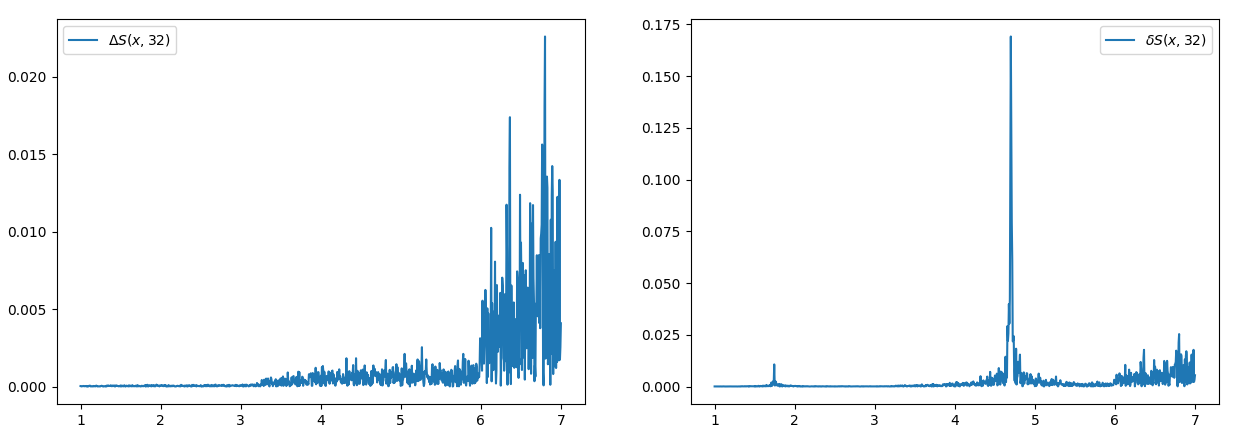
****

График абсолютной погрешности частичной суммы N = 32 членов ряда(слева); относительной(справа) с учетом округления вычислений:



**Анализ результатов:**

Из графиков частичных сумм видно, что последовательность частичных сумм стремится «ближе» к графику функции с увеличением N, однако частичные суммы хорошо аппроксимируют функцию только в начале заданного отрезка, а дальше существенно отличаются.

Абсолютная погрешность частичных сумм уменьшается по мере увеличения N.

Для 32-х членов ряда величина относительной погрешности в средней точке отрезка станет меньше машинного эпсилон.

Если использовать функции округления результатов вычисления, результат будет менее точен, по сравнению с отказом от их использования.

На графике относительной погрешности частичных сумм внутри отрезка [4,5] график стремится к бесконечности – это значение соответствует нулю функции f(x).

**Постановка задачи:**

**Задача 1.3.** Дана функция . Значения переменных указаны в варианте со всеми верными цифрами. Оценить погрешность результата двумя способами: а) используя оценки погрешности для арифметических операций, б) используя общую формулу погрешностей. Результат представить в двух формах записи: с явным указанием погрешностей и с учетом количества верных цифр.

(см. ***Приложение 2.B***)

Вариант 52:

Для нахождения погрешности функции следует использовать следующие утверждения.

**Утверждение 1.** Абсолютная погрешность алгебраической суммы не превосходит суммы абсолютных погрешностей слагаемых.

**Утверждение 2. Если**  и, то для оценки границ относительных погрешностей произведения и частного можно использовать приближенные равенства:, 

**Утверждение 3. Пусть** - дифференцируемая функция m переменных, вычисление которой производится при приближенно заданных значениях аргументов , , ….Тогда если , то можно

использовать равенства: , .

Также используется формула:

**Код программы** *(пояснения к коду красным цветом)***:**

import numpy  
import math

# класс числа с полями(характеристиками):

# value - значение

# abs\_error – абсолютная погрешность

# rel\_error – относительная погрешность

class Number:  
 def \_\_init\_\_(self, value, abs\_error):  
 self.value = value  
 self.abs\_error = abs\_error  
 self.rel\_error = abs\_error / numpy.abs(value)  
  
#

def f(a, b, c):  
 return (a.value\*\*2 + b.value\*\*2) / (a.value-c.value)  
  
# вычисление абсолютной погрешности функции через относительную погрешность с помощью оценок погрешностей для арифметических операций

def f\_abs\_error\_manually(a, b, c):  
 return f\_rel\_error\_manually(a, b, c) \* f(a, b, c)  
  
# вычисление относительной погрешности функции с помощью оценок погрешностей для арифметических операций

def f\_rel\_error\_manually(a, b, c):  
 a\_pow\_2\_error = a.rel\_error \* 2  
 b\_pow\_2\_error = b.rel\_error \* 2  
 numerator\_error = ((a.value\*\*2 \* a\_pow\_2\_error) + (b.value\*\*2 \* b\_pow\_2\_error)) / (a.value\*\*2 + b.value\*\*2)  
 denominator\_error = ((a.value \* a.rel\_error) + (c.value \* c.rel\_error)) / (a.value - c.value)  
 return numerator\_error + denominator\_error

#

def df\_da(a, b, c):  
 return (a.value\*\*2 - 2 \* a.value \* c.value - b.value\*\*2) / (a.value - c.value)\*\*2

#

def df\_db(a, b, c):  
 return 2 \* b.value / (a.value - c.value)  
  
#

def df\_dc(a, b, c):  
 return (a.value\*\*2 + b.value\*\*2) / (a.value-c.value)\*\*2  
  
# вычисление абсолютной погрешности функции с помощью общей формулы погрешностей

def f\_abs\_error\_auto(a, b, c):  
 a\_ = numpy.abs(df\_da(a, b, c)) \* a.abs\_error  
 b\_ = numpy.abs(df\_db(a, b, c)) \* b.abs\_error  
 c\_ = numpy.abs(df\_dc(a, b, c)) \* c.abs\_error  
 return a\_ + b\_ + c\_

# вычисление относительной погрешности функции через абсолютную с помощью общей формулы погрешностей  
  
def f\_rel\_error\_auto(a, b, c):  
 return f\_abs\_error\_auto(a, b, c) / numpy.abs(f(a, b, c))  
# нахождение первой значащей цифры числа  
  
def first\_digit(x):  
 for i in str(x):  
 if i != **'0'** and i != **'.'**:  
 return int(i)

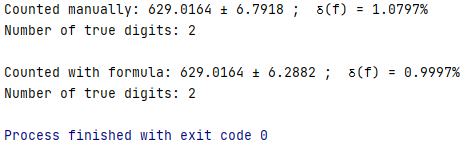
# определение количества верных цифр через относительную погрешность с помощью формулы: , где – первая значащая цифра числа, n – количество верных цифр. Из формулы   
  
def true\_digits(x, rel\_er):  
 return int(round(-math.log10(first\_digit(x) \* rel\_er) + 1))

# исходные данные  
  
A = Number(25.18, .01)  
B = Number(24.98, .01)  
C = Number(23.18, .01)

# вывод результатов

print(**'Counted manually: '** +  
 str(f(A, B, C)) +  
 **' ± '** +  
 str(round(f\_abs\_error\_manually(A, B, C), 4)) +  
 **' ;** \u03B4**(f) = '** +  
 str(round(f\_rel\_error\_manually(A, B, C) \* 100, 4)) +  
 **'%'**)  
print(**'Number of true digits: '** + true\_digits(f(A, B, C), f\_rel\_error\_manually(A, B, C)).\_\_str\_\_())  
print()  
  
print(**'Counted with formula: '** +  
 str(f(A, B, C)) +  
 **' ± '** +  
 str(round(f\_abs\_error\_auto(A, B, C), 4)) +  
 **' ;** \u03B4**(f) = '** +  
 str(round(f\_rel\_error\_auto(A, B, C) \* 100, 4)) +  
 **'%'**)  
print(**'Number of true digits: '** + true\_digits(f(A, B, C), f\_rel\_error\_auto(A, B, C)).\_\_str\_\_())

**Результаты выполнения:**

****

**Анализ результатов:**

Вычисление погрешностей с помощью формулы является более точным, чем вычисление через оценку погрешностей для арифметических операций