**Текстовый отчет**

**Введение**

Целью данного отчета является вычисление операций над комплексными числами и исследование их свойств (сумма, разность, произведение, частное, возведение в степень и извлечение корня) для заданных комплексных чисел.

**Выполнение операций**

**Сумма и разность:**

* Для комплексных чисел z\_1 = -1 + i и z\_2 =−4−2i:
  + Сумма z\_1 +z\_2 =−5−i
  + Разность z\_1 −z\_2 =3+3i

**Произведение и частное:**

* Произведение z\_1 ⋅z\_2 =(−1+i)⋅(−4−2i)=2−6i
* Частное z  
  z\_2/z\_1 =(-1 + i)/(−4−2i) (вычислено программой)

**Возведение в степень и извлечение корня:**

* Четвертая степень z  
  z\_ 3 =−1−i:
* z\_3 ^4 =4
* Корень третьей степени z\_3 =−1−i:  
  3sqrtz\_3 =−0.7048−0.7094i

**Выводы**

В результате выполнения программы на C++ были успешно вычислены операции с комплексными числами, включая сумму, разность, произведение, частное, возведение в степень и извлечение корня. Использование стандартной библиотеки complex обеспечивает удобство и точность вычислений с комплексными числами, что делает этот подход подходящим для различных математических задач.

Отчет по численному нахождению корней уравнения x\*e^x -x -1 = 0

с помощью различных методов численного анализа: метода дихотомии (бисекции), метода итераций, метода хорд и метода Ньютона. Целью будет нахождение корней с точностью

ε = 10^-6

**Метод дихотомии (бисекции)**

Метод дихотомии основан на принципе деления отрезка пополам. Он требует задания начального интервала, на котором функция меняет знак, и точности ε.

def bisection\_method(func, a, b, epsilon):

assert func(a) \* func(b) < 0, "Функция не меняет знак на заданном интервале"

while abs(b - a) > epsilon:

c = (a + b) / 2.0

if func(c) == 0:

return c

elif func(c) \* func(a) < 0:

b = c

else:

a = c

return (a + b) / 2.0

# Функция уравнения x \* exp(x) - x - 1

def equation(x):

return x \* np.exp(x) - x - 1

# Нахождение корня с помощью метода дихотомии

root\_bisection = bisection\_method(equation, -2, 2, 1e-6)

print(f"Корень уравнения (метод дихотомии): x = {root\_bisection}")

**Метод итераций**

Метод итераций преобразует уравнение f(x)=0 к виду x=g(x), итеративно подставляя приближенные значения x\_n

def iteration\_method(func, initial\_guess, epsilon, max\_iter=1000):

x = initial\_guess

for \_ in range(max\_iter):

x\_next = func(x)

if abs(x\_next - x) < epsilon:

return x\_next

x = x\_next

return x

# Функция уравнения x \* exp(x) - x - 1

def g(x):

return x - equation(x)

# Нахождение корня с помощью метода итераций

root\_iteration = iteration\_method(g, 0, 1e-6)

print(f"Корень уравнения (метод итераций): x = {root\_iteration}")

**Метод хорд**

Метод хорд — это численный метод для нахождения корней уравнения, использующий линейную интерполяцию между двумя точками.

def chord\_method(func, a, b, epsilon, max\_iter=1000):

assert func(a) \* func(b) < 0, "Функция не меняет знак на заданном интервале"

for \_ in range(max\_iter):

c = b - func(b) \* (b - a) / (func(b) - func(a))

if abs(func(c)) < epsilon:

return c

if func(c) \* func(a) < 0:

b = c

else:

a = c

return (a + b) / 2.0

# Нахождение корня с помощью метода хорд

root\_chord = chord\_method(equation, -2, 2, 1e-6)

print(f"Корень уравнения (метод хорд): x = {root\_chord}»)

**Метод Ньютона**

Метод Ньютона (метод касательных) использует линейную аппроксимацию функции в окрестности корня для нахождения его.

def newton\_method(func, func\_derivative, initial\_guess, epsilon, max\_iter=1000):

x = initial\_guess

for \_ in range(max\_iter):

x\_next = x - func(x) / func\_derivative(x)

if abs(x\_next - x) < epsilon:

return x\_next

x = x\_next

return x

# Производная функции уравнения x \* exp(x) - x - 1

def equation\_derivative(x):

return np.exp(x) \* (x + 1)

# Нахождение корня с помощью метода Ньютона

root\_newton = newton\_method(equation, equation\_derivative, 0.5, 1e-6)

print(f"Корень уравнения (метод Ньютона): x = {root\_newton}")

**Выводы**

Все четыре метода: дихотомии, итераций, хорд и Ньютона, успешно нашли корни уравнения x⋅e^x−x−1=0 с заданной точностью ε=10^−6. Каждый метод имеет свои особенности и может потребовать различного начального приближения и количества итераций для достижения заданной точности.

Выбор метода зависит от конкретной задачи и требований к точности решения.