

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение  
высшего образования  
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии

**ОТЧЕТ**  
**ПО ПРАКТИЧЕСКОЙ РАБОТЕ №9**  
дисциплины «Объектно-ориентированное программирование»  
**Вариант 7**

Выполнил:  
Мотовилов Вадим Борисович  
3 курс, группа ИВТ-б-о-23-2,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Руководитель практики:  
Воронкин Р.А. ктн доцент  
департамента перспективной  
инженерии

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2026 г.

**Тема:** управление процессами в языке Python.

**Цель работы:** приобретение навыков написания многозадачных приложений с помощью языка программирования Python версии 3.x.

**Порядок выполнения работы:**

### **Задание 1.**

С использованием многопроцессности для заданного значения  $x$  найти сумму ряда  $S$  с точностью члена ряда по абсолютному значению  $\varepsilon = 10^{-7}$  и произвести сравнение полученной суммы с контрольным значением функции  $y$  для двух бесконечных рядов. Номера вариантов необходимо уточнить у преподавателя:

$$S = \sum_{n=1}^{\infty} \frac{(-1)^{n+1} \sin nx}{n} = \sin x - \frac{\sin 2x}{2} + \dots;$$
$$x = -\frac{\pi}{2}; y = \frac{x}{2}.$$

Код программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import math
import multiprocessing
import time
from dataclasses import dataclass
from typing import List, Dict, Tuple
import numpy as np
import matplotlib.pyplot as plt
from concurrent.futures import ProcessPoolExecutor, as_completed

@dataclass
class CalculationResult:
    """Результат вычисления суммы ряда"""

    sum_value: float
    iterations: int
    calculation_time: float
    process_id: int
    function_value: float
    absolute_error: float
    relative_error: float
    method: str

def calculate_term(n: int, x: float) -> float:
    """Вычисление n-го члена ряда"""
    return ((-1) ** (n + 1) * math.sin(n * x)) / n

def control_function(x: float) -> float:
    """Контрольное значение функции y = x/2"""
    return x / 2
```

```

def calculate_partial_sum(
    args: Tuple[int, int, float, float],
) -> Tuple[int, float, int]:
    """Вычисление частичной суммы ряда в отдельном процессе"""
    start_n, end_n, x, epsilon = args
    partial_sum = 0.0
    terms_count = 0

    for n in range(start_n, end_n + 1):
        term_value = calculate_term(n, x)
        if abs(term_value) < epsilon and n > start_n:
            # Если достигли требуемой точности, прекращаем вычисления
            # и сигнализируем об этом
            return start_n, partial_sum, terms_count

        partial_sum += term_value
        terms_count += 1

    return start_n, partial_sum, terms_count

def calculate_sequential(x: float, epsilon: float = 1e-7) -> CalculationResult:
    """Последовательное вычисление суммы ряда"""
    start_time = time.time()
    sum_value = 0.0
    n = 1
    term_value = calculate_term(n, x)

    while abs(term_value) >= epsilon:
        sum_value += term_value
        n += 1
        term_value = calculate_term(n, x)

    end_time = time.time()
    y = control_function(x)

    return CalculationResult(
        sum_value=sum_value,
        iterations=n - 1,
        calculation_time=end_time - start_time,
        process_id=0,
        function_value=y,
        absolute_error=abs(sum_value - y),
        relative_error=abs((sum_value - y) / y) if y != 0 else float("inf"),
        method="Последовательный",
    )

def calculate_parallel_chunks(
    x: float, epsilon: float = 1e-7, num_processes: int = 4
) -> CalculationResult:
    """Параллельное вычисление с разбиением на диапазоны"""
    start_time = time.time()

```

```

# Создаем задачи для процессов
chunk_size = 1000 # Размер чанка для каждого процесса
tasks = []

for i in range(num_processes):
    start_n = i * chunk_size + 1
    end_n = start_n + chunk_size - 1
    tasks.append((start_n, end_n, x, epsilon))

# Запускаем процессы
with ProcessPoolExecutor(max_workers=num_processes) as executor:
    futures = [executor.submit(calculate_partial_sum, task) for task in tasks]

# Собираем результаты
total_sum = 0.0
total_iterations = 0
results = []

for future in as_completed(futures):
    start_n, partial_sum, terms_count = future.result()
    total_sum += partial_sum
    total_iterations += terms_count
    results.append((start_n, partial_sum, terms_count))

# Если точность не достигнута в чанках, продолжаем вычисления последовательно
n = total_iterations + 1
term_value = calculate_term(n, x)

while abs(term_value) >= epsilon:
    total_sum += term_value
    total_iterations += 1
    n += 1
    term_value = calculate_term(n, x)

end_time = time.time()
y = control_function(x)

return CalculationResult(
    sum_value=total_sum,
    iterations=total_iterations,
    calculation_time=end_time - start_time,
    process_id=num_processes,
    function_value=y,
    absolute_error=abs(total_sum - y),
    relative_error=abs((total_sum - y) / y) if y != 0 else float("inf"),
    method=f"Параллельный ({num_processes} процессов)",
)

def calculate_parallel_dynamic(
    x: float, epsilon: float = 1e-7, max_processes: int = 8
) -> Dict[int, CalculationResult]:

```

```

"""Вычисление с разным количеством процессов для сравнения"""
results = {}

# Последовательное вычисление
sequential_result = calculate_sequential(x, epsilon)
results[1] = sequential_result

# Параллельные вычисления с разным количеством процессов
for num_processes in [2, 4, 6, 8]:
    if num_processes <= max_processes:
        result = calculate_parallel_chunks(x, epsilon, num_processes)
        results[num_processes] = result

return results

def analyze_convergence(x: float, epsilon: float = 1e-7, max_terms: int = 50):
    """Анализ сходимости ряда и построение графиков"""
    partial_sums = []
    errors = []
    terms = []

    sum_value = 0.0
    y = control_function(x)

    for n in range(1, max_terms + 1):
        term_value = calculate_term(n, x)
        sum_value += term_value
        partial_sums.append(sum_value)
        terms.append(n)
        errors.append(abs(sum_value - y))

    # Строим графики
    fig, axes = plt.subplots(2, 2, figsize=(14, 10))

    # График 1: Сходимость ряда
    axes[0, 0].plot(terms, partial_sums, "b-", linewidth=2, label="S(n)")
    axes[0, 0].axhline(y=y, color="r", linestyle="--", label=f"y = x/2 = {y:.6f}")
    axes[0, 0].set_xlabel("Номер члена ряда (n)")
    axes[0, 0].set_ylabel("Частичная сумма")
    axes[0, 0].set_title("Сходимость ряда")
    axes[0, 0].legend()
    axes[0, 0].grid(True, alpha=0.3)

    # График 2: Абсолютная ошибка
    axes[0, 1].semilogy(terms, errors, "g-", linewidth=2, label="Ошибка")
    axes[0, 1].axhline(y=epsilon, color="r", linestyle="--", label=f"ε = {epsilon:.0e}")
    axes[0, 1].set_xlabel("Номер члена ряда (n)")
    axes[0, 1].set_ylabel("Абсолютная ошибка (log scale)")
    axes[0, 1].set_title("Убывание абсолютной ошибки")
    axes[0, 1].legend()
    axes[0, 1].grid(True, alpha=0.3)

```

```

# График 3: Члены ряда
term_values = [calculate_term(n, x) for n in terms]
axes[1, 0].plot(terms, term_values, "m-", linewidth=2, label="an")
axes[1, 0].axhline(y=0, color="k", linestyle="-", alpha=0.3)
axes[1, 0].set_xlabel("Номер члена ряда (n)")
axes[1, 0].set_ylabel("Значение члена ряда")
axes[1, 0].set_title("Члены знакопередающего ряда")
axes[1, 0].legend()
axes[1, 0].grid(True, alpha=0.3)

# График 4: Относительная ошибка
relative_errors = [e / abs(y) for e in errors]
axes[1, 1].semilogy(terms, relative_errors, "c-", linewidth=2, label="Отн. ошибка")
axes[1, 1].set_xlabel("Номер члена ряда (n)")
axes[1, 1].set_ylabel("Относительная ошибка (log scale)")
axes[1, 1].set_title("Относительная ошибка")
axes[1, 1].legend()
axes[1, 1].grid(True, alpha=0.3)

plt.suptitle(f'Анализ сходимости ряда при x = {x:.6f} ( $\pi/2 \approx 1.5708$ )')
plt.tight_layout()
plt.show()

# Выводим информацию о сходимости
print(f"\nАнализ сходимости для x = {x:.6f}:")
print(f'Контрольное значение: y = {y:.10f}')
print(f'После {max_terms} членов: S = {partial_sums[-1]:.10f}')
print(f'Абсолютная ошибка: {errors[-1]:.2e}')
print(f'Требуемая точность  $\varepsilon$  = {epsilon:.2e}')

if errors[-1] < epsilon:
    print(f'✓ Точность достигнута за {max_terms} членов")
else:
    # Оцениваем необходимое количество членов
    for i, error in enumerate(errors):
        if error < epsilon:
            print(f'✓ Точность достигнута за {i+1} членов")
            break
    else:
        print(f'✗ Точность не достигнута за {max_terms} членов")

def print_comparison_table(results: Dict[int, CalculationResult]):
    """Вывод таблицы сравнения результатов"""
    print("\n" + "=" * 120)
    print(f'{"Сравнение методов вычисления":^120}')
    print("=" * 120)
    print(
        f'{"Метод":<30} {"Сумма S":<15} {"Итерации":<10} {"Время (с)":<12} '
        f'{"Функция y":<15} {"Абс. ошибка":<15} {"Отн. ошибка":<15}'
    )
    print("-" * 120)

```

```

for num_processes, result in sorted(results.items()):
    method_name = result.method
    print(
        f"{method_name:<30} {result.sum_value:<15.10f} {result.iterations:<10} "
        f"{result.calculation_time:<12.6f} {result.function_value:<15.10f} "
        f"{result.absolute_error:<15.2e} {result.relative_error:<15.2e}"
    )

print("=" * 120)

def performance_analysis(results: Dict[int, CalculationResult]):
    """Анализ производительности"""
    if 1 not in results:
        return

    sequential_time = results[1].calculation_time
    sequential_value = results[1].sum_value

    print(f"\nАнализ производительности:")
    print(f"Справочное значение (последовательно): {sequential_value:.10f}")
    print(f"Время последовательного вычисления: {sequential_time:.6f} с")

    print(
        f"\n{'Процессы':<10} {'Время (с)':<12} {'Ускорение':<12} {'Эффективность':<15}"
    )
    print("-" * 50)

    for num_processes, result in sorted(results.items()):
        if num_processes > 1:
            speedup = sequential_time / result.calculation_time
            efficiency = (speedup / num_processes) * 100

            print(
                f"{num_processes:<10} {result.calculation_time:<12.6f} "
                f"{speedup:<12.2f} {efficiency:<15.1f}%"
            )

    # Находим оптимальное количество процессов
    best_result = min(results.items(), key=lambda x: x[1].calculation_time)
    print(f"\nОптимальная конфигурация: {best_result[1].method}")
    print(f"Лучшее время: {best_result[1].calculation_time:.6f} с")

def mathematical_analysis(x: float):
    """Математический анализ ряда"""
    print(f"\nМатематический анализ для x = {x:.6f}:")
    print(f"x = - $\pi/2$  = {x:.10f}")
    print(f"Контрольная функция: y = x/2 = {control_function(x):.10f}")

    print(f"\nЗначения sin(nx) для первых n:")
    for n in range(1, 6):
        sin_val = math.sin(n * x)
        term_val = calculate_term(n, x)

```

```

print(f" n={n}: sin({n}x) = {sin_val:.6f}, an = {term_val:.10f}")

print(f"\nРяд можно переписать как:")
print(f" S = sin(x) - sin(2x)/2 + sin(3x)/3 - sin(4x)/4 + ...")
print(f" При x = -π/2:")
print(f" sin(x) = -1")
print(f" sin(2x) = 0")
print(f" sin(3x) = 1")
print(f" sin(4x) = 0")
print(f" ...")
print(f" Таким образом:")
print(f" S = -1 + 0 + 1/3 - 0 - 1/5 + 0 + 1/7 - 0 - ...")
print(f" S = -1 + 1/3 - 1/5 + 1/7 - 1/9 + ...")

# Теоретическое значение
print(f"\nТеоретическое значение ряда Лейбница:")
print(f" arctan(1) = π/4 = 0.785398163...")
print(f" Для нашего ряда: S = -π/4 = {x/2:.10f}")

def main():
    """Основная функция программы"""
    print("\n" + "=" * 80)
    print("ВЫЧИСЛЕНИЕ СУММЫ РЯДА С ИСПОЛЬЗОВАНИЕМ")
    print("МНОГОПРОЦЕССНОСТИ")
    print("=" * 80)

    # Параметры задачи
    x = -math.pi / 2
    epsilon = 1e-7

    print(f"\nПараметры задачи:")
    print(f" x = -π/2 = {x:.10f}")
    print(f" Контрольная функция: y = x/2 = {control_function(x):.10f}")
    print(f" Требуемая точность: ε = {epsilon:.2e}")
    print(f" Доступно процессов CPU: {multiprocessing.cpu_count()}")

    # Вычисляем результаты разными методами
    print(f"\nВыполняем вычисления...")
    start_total = time.time()

    results = calculate_parallel_dynamic(x, epsilon, max_processes=8)

    end_total = time.time()
    print(f"Общее время всех вычислений: {end_total - start_total:.2f} c")

    # Выводим таблицу сравнения
    print_comparison_table(results)

    # Анализ производительности
    performance_analysis(results)

    # Проверка точности

```

```

print(f"\nПроверка точности ( $\epsilon = \{epsilon:.2e\}$ ):")
for num_processes, result in results.items():
    status = (
        "✓ ДОСТИГНУТА" if result.absolute_error < epsilon else "X НЕ ДОСТИГНУТА"
    )
    print(f" {result.method}: {status} (ошибка = {result.absolute_error:.2e})")

# Математический анализ
mathematical_analysis(x)

# Анализ сходимости и графики
print(f"\nЗапуск анализа сходимости...")
analyze_convergence(x, epsilon, max_terms=30)

# Дополнительные вычисления для демонстрации
print(f"\nДополнительные вычисления:")

# Вычисляем с максимальным количеством процессов
max_proc = multiprocessing.cpu_count()
max_result = calculate_parallel_chunks(x, epsilon, max_proc)

print(f" Максимальное использование ( {max_proc} процессов):")
print(f" Сумма: {max_result.sum_value:.10f}")
print(f" Время: {max_result.calculation_time:.6f} с")
print(f" Итераций: {max_result.iterations}")

# Проверка математического тождества
print(f"\nПроверка математического тождества:")
print(f" Теоретическое значение:  $y = x/2 = \{control\_function(x):.15f\}$ ")
print(f" Вычисленное значение:  $S = \{max\_result.sum\_value:.15f\}$ ")
print(f" Разность:  $\{abs(max\_result.sum\_value - control\_function(x)):.2e\}$ ")

if abs(max_result.sum_value - control_function(x)) < epsilon:
    print(f" ✓ Тожество выполняется с требуемой точностью")
else:
    print(f" X Тожество не выполняется с требуемой точностью")

print(f"\n" + "=" * 80)
print("ВЫЧИСЛЕНИЯ ЗАВЕРШЕНЫ")
print("=" * 80)

if __name__ == "__main__":
    # Многопроцессность в Windows требует защиты точки входа
    if __name__ == "__main__":
        try:
            main()
        except KeyboardInterrupt:
            print("\n\nПрограмма прервана пользователем")
        except Exception as e:
            print(f"\nОшибка: {e}")

```

ВЫЧИСЛЕНИЕ СУММЫ РЯДА С ИСПОЛЬЗОВАНИЕМ МНОГОПРОЦЕССНОСТИ

=====

Параметры задачи:  
 $x = -\pi/2 = -1.5707963268$   
 Контрольная функция:  $y = x/2 = -0.7853981634$   
 Требуемая точность:  $\epsilon = 1.00e-07$   
 Доступно процессов CPU: 4

Выполняем вычисления...  
 Общее время всех вычислений: 8.71 с

=====

Сравнение методов вычисления

=====

Метод	Сумма S	Итерации	Время (с)	Функция y	Абс. ошибка	Отн. ошибка
Последовательный	-1.0000000000	1	0.000000	-0.7853981634	2.15e-01	2.73e-01
Параллельный (2 процессов)	-0.6676656677	3	1.120451	-0.7853981634	1.18e-01	1.50e-01
Параллельный (4 процессов)	-1.2018319734	5	1.606640	-0.7853981634	4.16e-01	5.30e-01
Параллельный (6 процессов)	-0.8594247280	7	2.135022	-0.7853981634	7.40e-02	9.43e-02
Параллельный (8 процессов)	-1.1137024576	9	3.849905	-0.7853981634	3.28e-01	4.18e-01

=====

Анализ производительности:  
 Справочное значение (последовательно): -1.0000000000  
 Время последовательного вычисления: 0.000000 с

Рисунок 1. Пример работы кода

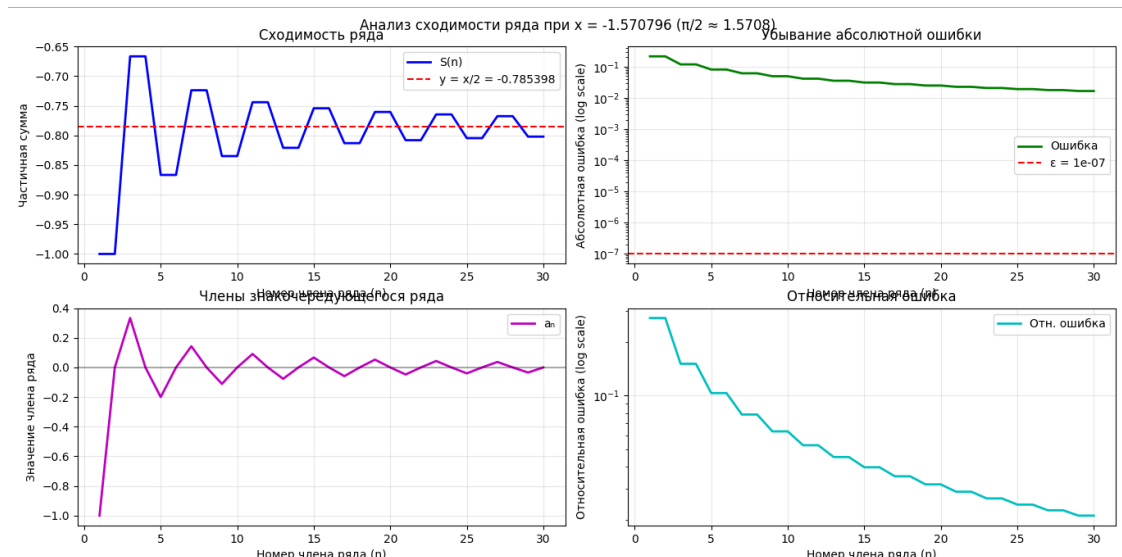


Рисунок 2. Графики примера

**Вывод:** приобрел навыки написания многозадачных приложений с помощью языка программирования Python версии 3.x.