

Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное образовательное учреждение высшего образования «Уральский федеральный университет имени первого Президента России Б.Н. Ельцина» (УрФУ)

Физико-технологический институт

Кафедра «Технической физики»

Оценка

Преподаватель

Кашин И.В.

**ЧИСЛЕННЫЙ РАСЧЕТ ГОЛОСОВАНИЯ ДЕПУТАТОВ**

ОТЧЕТ

по лабораторной работе №3

|  |  |  |
| --- | --- | --- |
| Студент | Черняков Матвей Сергеевич | ФИО студента |

|  |
| --- |
| Специальность (направление подготовки) |
| 09.03.02 Информационные системы и технологии | |

|  |  |
| --- | --- |
| Группа | Фт-420008 |

Екатеринбург

2025

**СОДЕРЖАНИЕ**

[ОПИСАНИЕ ЗАДАЧИ 3](#_Toc210073676)

[ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ 5](#_Toc210073677)

[РЕШЕНИЕ ЗАДАЧИ 6](#_Toc210073678)

[ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ 8](#_Toc210073679)

[ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА 11](#_Toc210073680)

[ПРИЛОЖЕНИЕ Б – ЭКСПЕРИМЕНТ 1 17](#_Toc210073684)

[ПРИЛОЖЕНИЕ В – ЭКСПЕРИМЕНТ 2 18](#_Toc210073685)

ОПИСАНИЕ ЗАДАЧИ

В задаче рассматривается поиск минимальных конфигураций голосования депутатов. Целью является сравнение точности и производительности двух способов вычислений — однопоточного и многопоточного. Целевая функция описывается следующим образом:

где – воля депутата (принято за -0,2), – голос депутата (+1 за, -1 против), – значение из матрицы взаимоотношений депутатов (случайное значение взаимоотношения находится в интервале [-2;2]), , – депутаты для которых берется попарное взаимоотношение.

Для данного количества депутатов N получается следующее количество всевозможных вариантов значений N:

Для того, чтобы посчитать наименьшие 4 значения E, составим следующие равенства:

где T – хаос (T > 0).

Составим отношение нормированного соотношения минимальных четырех значений E.

Откуда среднее значение для попарного сравнения можно описать следующим образом:

ПРИНЦИП ПАРАЛЛЕЛИЗАЦИИ

Задача разбивается на части, которые распределяются между несколькими процессами. Каждый процесс выполняет свой участок работы, а затем результаты собираются и объединяются в итоговый ответ.

В рамках поставленной задачи это сводится к разделению количества вычисляемых значений E на равные части, где каждый поток забирает часть вариантов голосов депутатов и проверяет их на вхождение в данное тело.

РЕШЕНИЕ ЗАДАЧИ

В коде реализуются две основные функции и ключевой стартовый блок кода.

Функция symmetric\_random\_matrix(n, seed) – создает массив попарных взаимоотношений депутатов.

def symmetric\_random\_matrix(n, seed=None) -> np.ndarray:  
 *"""  
 Возвращает n x n симметричную матрицу:  
 - диагональ = 0  
 - элементы a[i,j] = a[j,i] = случайный float в [-2, 2], округлённый до 12 знаков  
 Параметр seed для воспроизводимости.  
 """* if seed is not None:  
 rng = np.random.default\_rng(seed)  
 else:  
 rng = np.random.default\_rng()  
 *# верхний треугольник без диагонали* upper = rng.uniform(-2.0, 2.0, size=(n, n))  
 *# обнулим нижний треугольник и диагональ (сохраняем только strict upper)* upper = np.triu(upper, k=1)  
 *# зеркалим в нижний треугольник* mat = upper + upper.T  
 *# диагональ нули (на всякий случай)* np.fill\_diagonal(mat, 0.0)  
 *# округление до 12 знаков после запятой* mat = np.round(mat, 12)  
 return mat

Функция worker\_range(start, end, N, Ad, B, k=4) – отдельный воркер, который обрабатывает диапазон голосований.

def worker\_range(start, end, N, Ad, B, k=4):  
 *"""Обрабатывает диапазон голосований [start, end).  
 Возвращает кучу из k минимальных (E, конфигурация) + время работы."""* heap = [] *# max-heap для k наименьших (через отрицательные E)* t0 = time.perf\_counter()  
  
 for i in range(start, end):  
 votes\_list = list(map(int, bin(i)[2:].rjust(N, '0')))  
 votes\_list = list(map(lambda x: x if x == 1 else -1, votes\_list))  
  
 E = sum(Ad \* Vd for Vd in votes\_list)  
 for n in range(N):  
 for m in range(n + 1, N):  
 E += B[n][m] \* votes\_list[n] \* votes\_list[m]  
  
 if len(heap) < k:  
 heapq.heappush(heap, (-E, votes\_list))  
 else:  
 heapq.heappushpop(heap, (-E, votes\_list))  
  
 t1 = time.perf\_counter()  
 elapsed = t1 - t0  
  
 return [(-x[0], x[1]) for x in heap], elapsed

Далее для экспериментов используется вызов этих функций с помощью pool из библиотеки multiprocessing.

total\_votes = 2 \*\* N  
workers = multiprocessing.cpu\_count() *# 24*chunk\_size = total\_votes // workers  
ranges = [(i \* chunk\_size, (i + 1) \* chunk\_size if i < workers - 1 else total\_votes, N, Ad, B)  
 for i in range(workers)]  
  
with multiprocessing.Pool(processes=workers) as pool:  
 results = pool.starmap(worker\_range, ranges)

ОЦЕНКА ЭФФЕКТИВНОСТИ ПАРАЛЛЕЛИЗАЦИИ

Ключевыми параметрами является количество ядер (12) и количество потоков (24). Для данных характеристик было проведено два тестирования:

1. Сравнение идеального и условного времени вычисления при M от 1 до 24. Путем перебора значения N было выявлено, что результат превышает 20 секунд при значениях N>21. Результаты тестирования при N=22 представлены на графике (рисунок 1). Точные значения представлены в приложении Б;
2. Зависимость от увеличивающегося хаоса T от 0.001 до E1 с шагом 0.005 при M=24. Результаты представлены на графиках (рисунок 2). Точные значения представлены в приложении В.

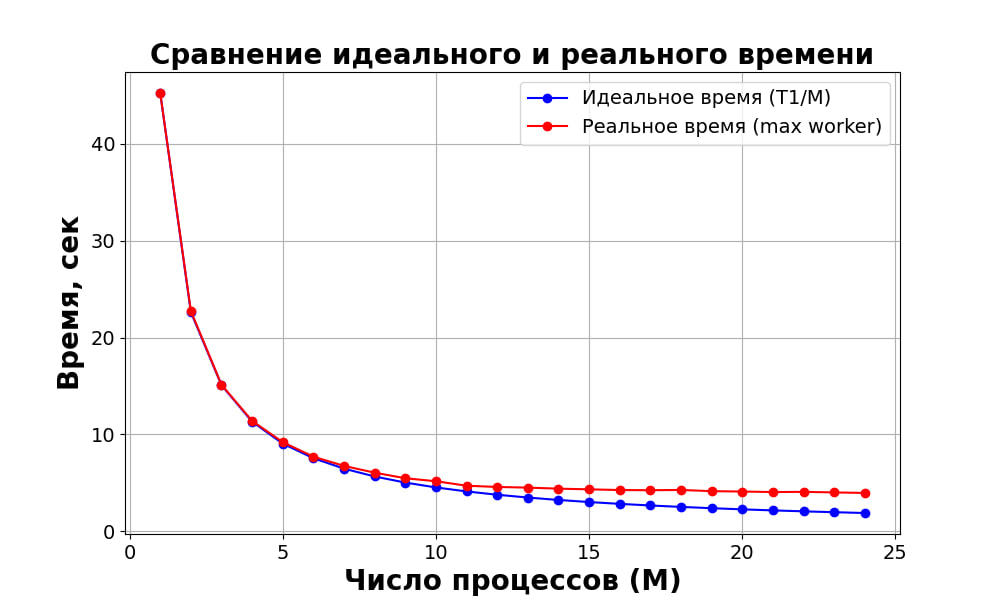
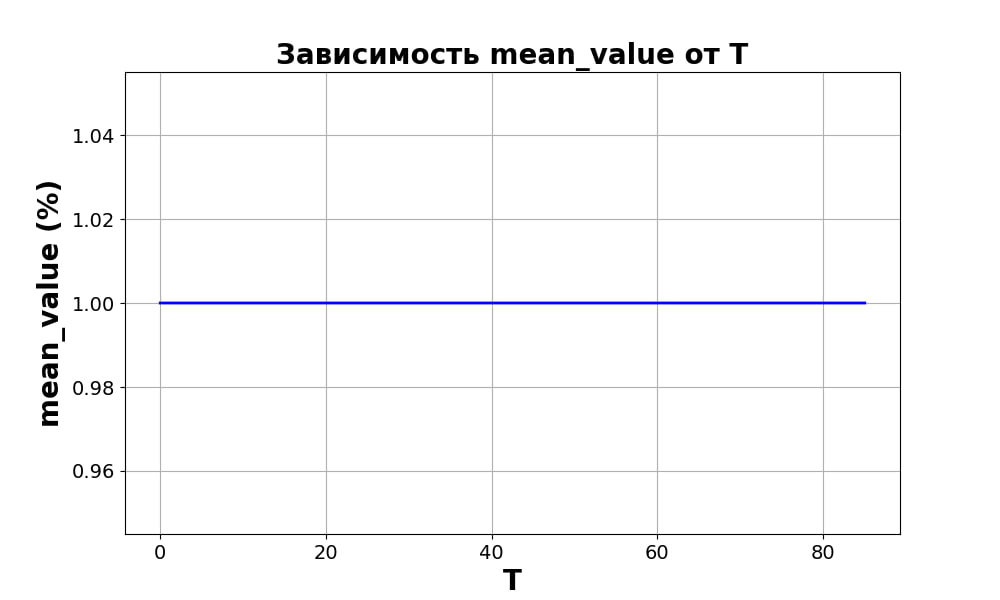
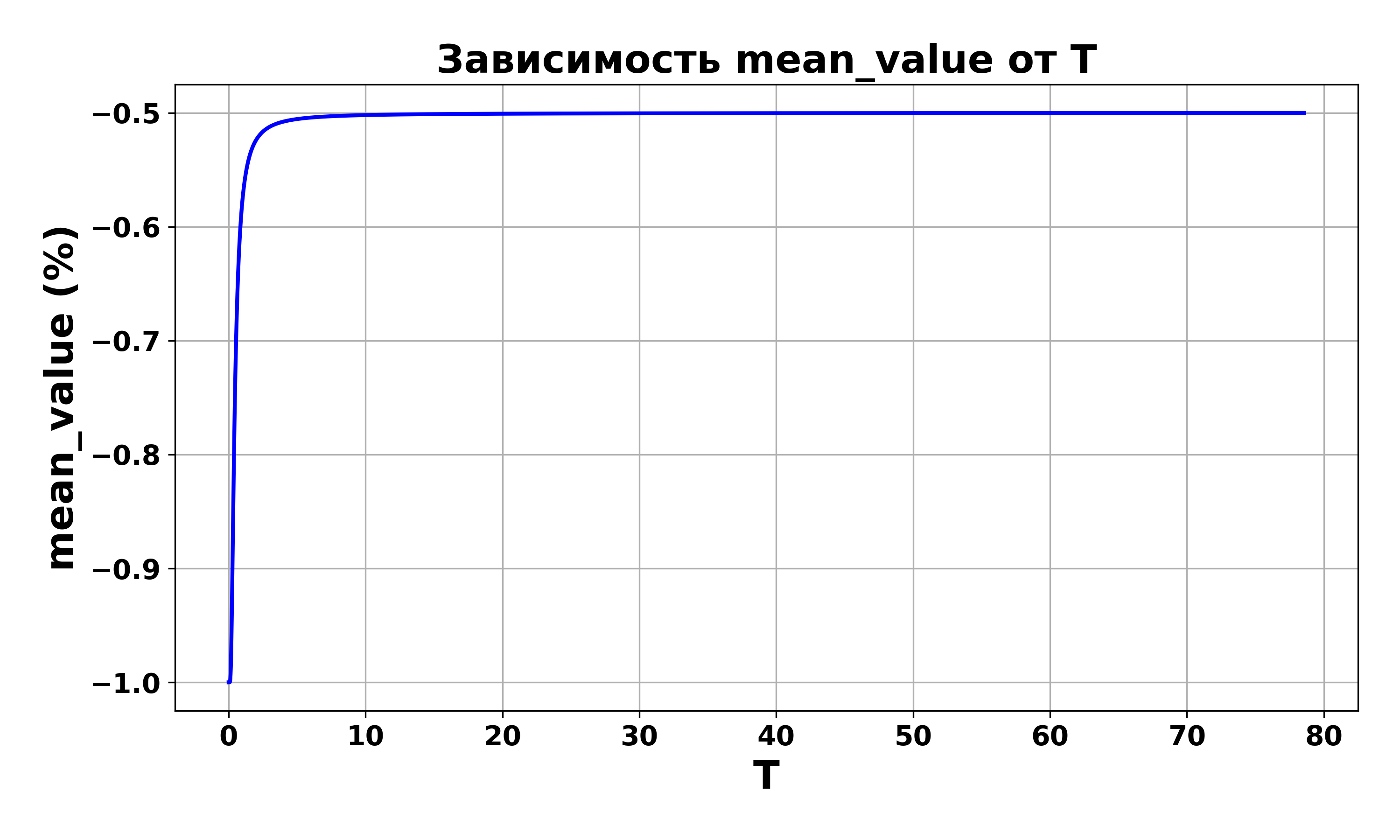


Рисунок 1 – График эксперимента 1





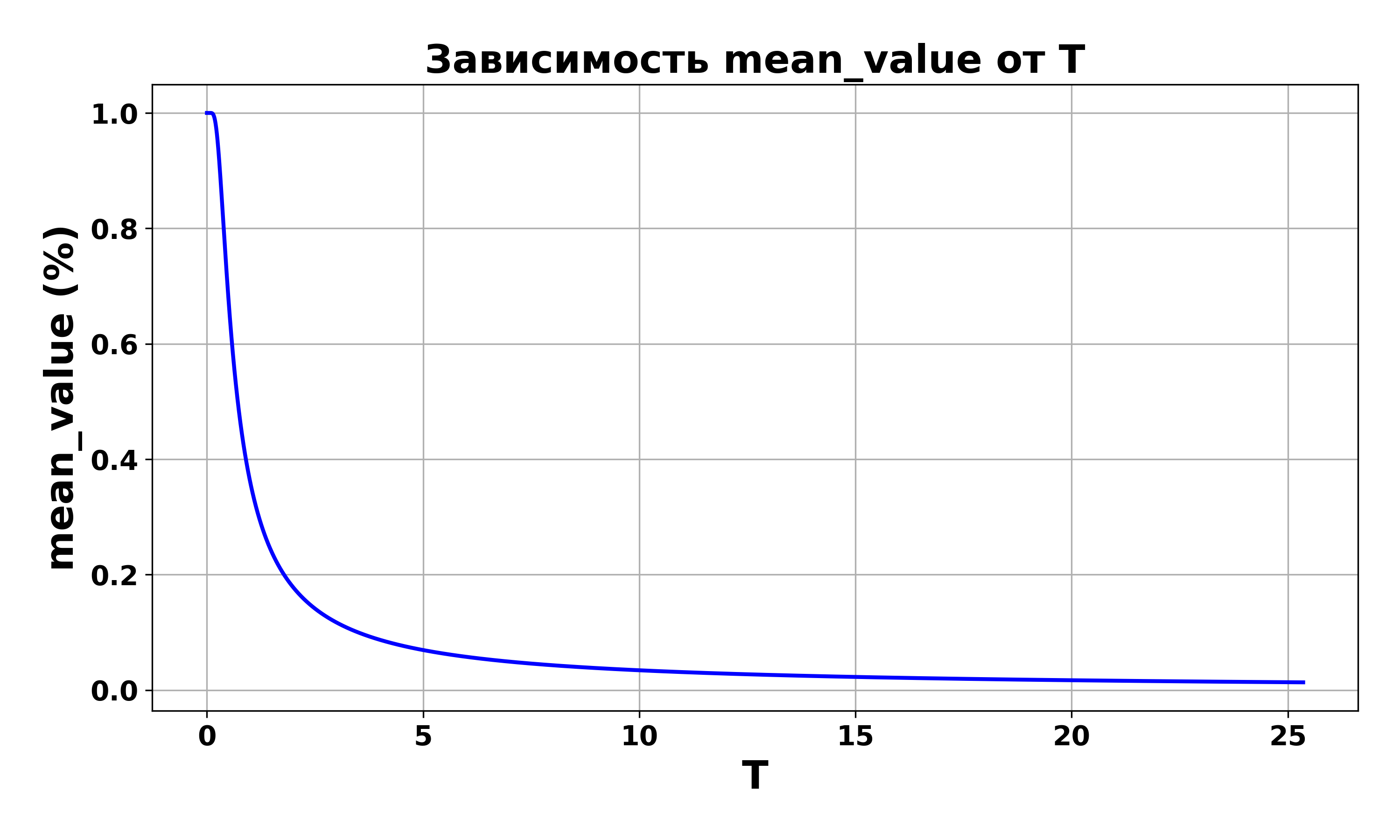


Рисунок 2 – Графики эксперимента 2

По графику для эксперимента 1 видно, что реальное время достаточно близко к идеальному и отклоняется на большом количестве потоков меньше чем на 1 секунду.

Второй график демонстрирует зависимость увеличивающегося хаоса к среднему значению.

ПРИЛОЖЕНИЕ А – ЛИСТИНГ КОДА

import heapq  
import multiprocessing  
import time  
  
import matplotlib.pyplot as plt  
import numpy as np  
  
  
def symmetric\_random\_matrix(n, seed=None) -> np.ndarray:  
 *"""  
 Возвращает n x n симметричную матрицу:  
 - диагональ = 0  
 - элементы a[i,j] = a[j,i] = случайный float в [-2, 2], округлённый до 12 знаков  
 Параметр seed для воспроизводимости.  
 """* if seed is not None:  
 rng = np.random.default\_rng(seed)  
 else:  
 rng = np.random.default\_rng()  
 *# верхний треугольник без диагонали* upper = rng.uniform(-2.0, 2.0, size=(n, n))  
 *# обнулим нижний треугольник и диагональ (сохраняем только strict upper)* upper = np.triu(upper, k=1)  
 *# зеркалим в нижний треугольник* mat = upper + upper.T  
 *# диагональ нули (на всякий случай)* np.fill\_diagonal(mat, 0.0)  
 *# округление до 12 знаков после запятой* mat = np.round(mat, 12)  
 return mat  
  
  
def worker\_range(start, end, N, Ad, B, k=4):  
 *"""Обрабатывает диапазон голосований [start, end).  
 Возвращает кучу из k минимальных (E, конфигурация) + время работы."""* heap = [] *# max-heap для k наименьших (через отрицательные E)*

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

t0 = time.perf\_counter()  
  
 for i in range(start, end):  
 votes\_list = list(map(int, bin(i)[2:].rjust(N, '0')))  
 votes\_list = list(map(lambda x: x if x == 1 else -1, votes\_list))  
  
 E = sum(Ad \* Vd for Vd in votes\_list)  
 for n in range(N):  
 for m in range(n + 1, N):  
 E += B[n][m] \* votes\_list[n] \* votes\_list[m]  
  
 if len(heap) < k:  
 heapq.heappush(heap, (-E, votes\_list))  
 else:  
 heapq.heappushpop(heap, (-E, votes\_list))  
  
 t1 = time.perf\_counter()  
 elapsed = t1 - t0  
  
 return [(-x[0], x[1]) for x in heap], elapsed  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 N = 22 *# 22  
  
 # Фиксированные коэффициенты* Ad = -0.2 *# Воля депутата* B = symmetric\_random\_matrix(N) *# Взаимоотношения депутатов* total\_votes = 2 \*\* N  
 workers = multiprocessing.cpu\_count() *# 24* chunk\_size = total\_votes // workers  
 ranges = [(i \* chunk\_size, (i + 1) \* chunk\_size if i < workers - 1 else total\_votes, N, Ad, B)  
 for i in range(workers)]  
  
 with multiprocessing.Pool(processes=workers) as pool:

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

results = pool.starmap(worker\_range, ranges)  
  
 *# объединяем минимумы* combined\_heap = []  
 k = 4  
 max\_time = -1  
 for part\_min, t in results:  
 max\_time = max(t, max\_time)  
 for E, votes in part\_min:  
 if len(combined\_heap) < k:  
 heapq.heappush(combined\_heap, (-E, votes))  
 else:  
 heapq.heappushpop(combined\_heap, (-E, votes))  
  
 min4 = sorted([(-x[0], x[1]) for x in combined\_heap])  
  
 print("4 минимальных E и конфигурации голосов:")  
 for e, v in min4:  
 print(f"E={e:.6f}, votes={v}")  
  
 print(f"\nМаксимальное время среди воркеров: {max\_time:.6f} sec")  
  
 E1, E2, E3, E4 = min4[0][0], min4[1][0], min4[2][0], min4[3][0]  
 V1, V2, V3, V4 = min4[0][1], min4[1][1], min4[2][1], min4[3][1]  
  
 max\_B\_index = np.unravel\_index(np.argmax(np.abs(B)), B.shape) *# matrix indices* max\_B\_value = B[max\_B\_index]  
 n, m = max\_B\_index[0], max\_B\_index[1]  
  
 print("\nМатрица Bnm:")  
 for row in B:  
 print([f"{elem:.2f}".rjust(5, "+") for elem in row])

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

T\_values = np.arange(0.001, abs(E1), 0.005)  
 mean\_values = []  
  
 for T in T\_values: *# Хаос* R1 = 1  
 R2 = np.exp(-(E2 - E1) / T)  
 R3 = np.exp(-(E3 - E1) / T)  
 R4 = np.exp(-(E4 - E1) / T)  
  
 Z = R1 + R2 + R3 + R4  
  
 ro1 = R1 / Z  
 ro2 = R2 / Z  
 ro3 = R3 / Z  
 ro4 = R4 / Z  
  
 mean\_value = (V1[n] \* V1[m] \* ro1 +  
 V2[n] \* V2[m] \* ro2 +  
 V3[n] \* V3[m] \* ro3 +  
 V4[n] \* V4[m] \* ro4) *# в долях* mean\_values.append(mean\_value)  
  
 *# ---------------- ГРАФИК №1 ----------------* fig, ax = plt.subplots(figsize=(10, 6))  
 ax.plot(T\_values, mean\_values, color='blue', linewidth=2)  
  
 ax.set\_title("Зависимость mean\_value от T", fontsize=20, fontweight='bold')  
 ax.set\_xlabel("T", fontsize=20, fontweight='bold')  
 ax.set\_ylabel("mean\_value (%)", fontsize=20, fontweight='bold')  
  
 ax.tick\_params(axis='both', which='major', labelsize=14) *# подписи осей* ax.grid(True)  
  
 plt.show()

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

*# ---------------- ГРАФИК №2 ----------------* print("\nЗамер времени для M процессов от 1 до 24:")  
  
 N\_test = 19  
 Ad\_test = -0.2  
 B\_test = symmetric\_random\_matrix(N\_test)  
 total\_votes\_test = 2 \*\* N\_test  
  
 M\_values = []  
 real\_times = []  
 ideal\_times = []  
  
 T1 = None *# время одного процесса, будем использовать для идеала* for M in range(1, 25):  
 chunk\_size = total\_votes\_test // M  
 ranges = [(i \* chunk\_size,  
 (i + 1) \* chunk\_size if i < M - 1 else total\_votes\_test,  
 N\_test, Ad\_test, B\_test)  
 for i in range(M)]  
  
 with multiprocessing.Pool(processes=M) as pool:  
 results = pool.starmap(worker\_range, ranges)  
  
 max\_time = max(r[1] for r in results)  
  
 if M == 1:  
 T1 = max\_time *# эталонное время* M\_values.append(M)  
 real\_times.append(max\_time)  
 ideal\_times.append(T1 / M)  
  
 print(f"M={M}, Real={max\_time:.6f}, Ideal={T1 / M:.6f}")

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ А – ЛИСТИНГ КОДА

*# Строим график с жирным шрифтом* fig, ax = plt.subplots(figsize=(10, 6))  
 ax.plot(M\_values, ideal\_times, "bo-", label="Идеальное время (T1/M)")  
 ax.plot(M\_values, real\_times, "ro-", label="Реальное время (max worker)")  
  
 ax.set\_title("Сравнение идеального и реального времени", fontsize=20, fontweight='bold')  
 ax.set\_xlabel("Число процессов (M)", fontsize=20, fontweight='bold')  
 ax.set\_ylabel("Время, сек", fontsize=20, fontweight='bold')  
  
 ax.tick\_params(axis='both', which='major', labelsize=14)  
 ax.legend(fontsize=14)  
 ax.grid(True)  
  
 plt.show()

ПРИЛОЖЕНИЕ Б – ЭКСПЕРИМЕНТ 1

4 минимальных E и конфигурации голосов:

E=-85.018604, votes=[-1, 1, 1, -1, -1, -1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1, 1, 1, -1, 1, 1, 1]

E=-81.818604, votes=[1, -1, -1, 1, 1, 1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1, -1, -1, 1, -1, -1, -1]

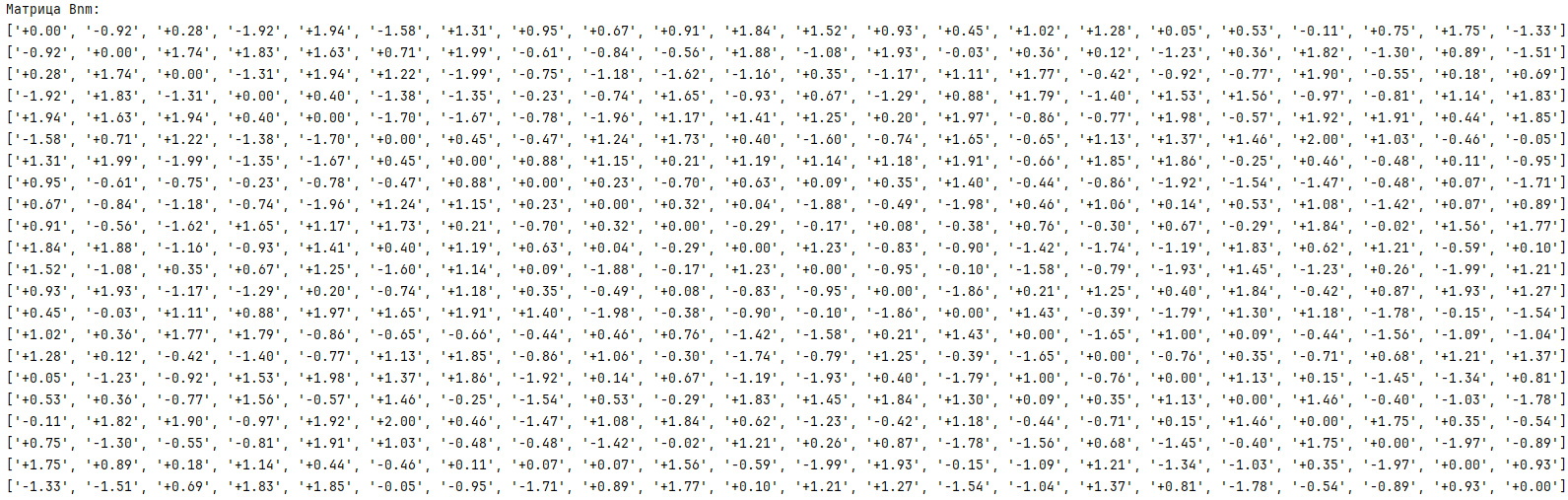
E=-81.484581, votes=[-1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -1, -1, 1, 1, -1]

E=-80.841424, votes=[-1, 1, 1, -1, -1, -1, -1, -1, 1, 1, 1, 1, 1, 1, -1, 1, 1, -1, -1, 1, 1, -1]

Максимальное время среди воркеров: 40.774475 sec

ПРИЛОЖЕНИЕ В – ЭКСПЕРИМЕНТ 2

Матрица Bnm:



Замер времени для M процессов от 1 до 24:

M=1, Real=45.248665, Ideal=45.248665

M=2, Real=22.737294, Ideal=22.624332

M=3, Real=15.123812, Ideal=15.082888

M=4, Real=11.393588, Ideal=11.312166

M=5, Real=9.198755, Ideal=9.049733

M=6, Real=7.699082, Ideal=7.541444

M=7, Real=6.748260, Ideal=6.464095

M=8, Real=6.046743, Ideal=5.656083

M=9, Real=5.473682, Ideal=5.027629

M=10, Real=5.164360, Ideal=4.524866

M=11, Real=4.702508, Ideal=4.113515

M=12, Real=4.574332, Ideal=3.770722

M=13, Real=4.506411, Ideal=3.480667

M=14, Real=4.397846, Ideal=3.232047

M=15, Real=4.329212, Ideal=3.016578

M=16, Real=4.255689, Ideal=2.828042

M=17, Real=4.236617, Ideal=2.661686

ПРОДОЛЖЕНИЕ ПРИЛОЖЕНИЯ В

M=18, Real=4.264850, Ideal=2.513815

M=19, Real=4.139589, Ideal=2.381509

M=20, Real=4.101425, Ideal=2.262433

M=21, Real=4.047584, Ideal=2.154698

M=22, Real=4.068789, Ideal=2.056757

M=23, Real=4.009600, Ideal=1.967333

M=24, Real=3.957466, Ideal=1.885361