Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
Высшая школа компьютерных технологий и информационных систем

КУРСОВАЯ РАБОТА

Исследование зависимости светимости от температуры для абсолютно черного тела.

по дисциплине «Вычислительная математика»

Выполнил
студент гр. 5130901/20004
_____ Гайнутдинов М. Р.
(подпись)

Руководитель
_____ Куляшова З. В.
(подпись)

«___» _____ 2024 г.

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Абсолютно черное тело (далее АЧТ) — это теоретическая модель, представляющая собой идеализированный объект, который поглощает всё падающее на него излучение, не отражая и не пропуская его. Вследствие этого абсолютно черное тело является идеальным излучателем, испуская электромагнитное излучение всех частот в зависимости от своей температуры. Ключевой особенностью излучения абсолютно черного тела является то, что его спектр *определяется только температурой*, а не формой или составом объекта. Это означает, что при заданной температуре абсолютно чёрное тело излучает одинаково, независимо от его материала, формы или размера. Для наглядности демонстрации обратимся к рисунку диаграммы цветности абсолютно твердого тела (рис. 1.1), а также рисунку вертикальной шкалы цветности абсолютно твердого тела (рис. 1.2).

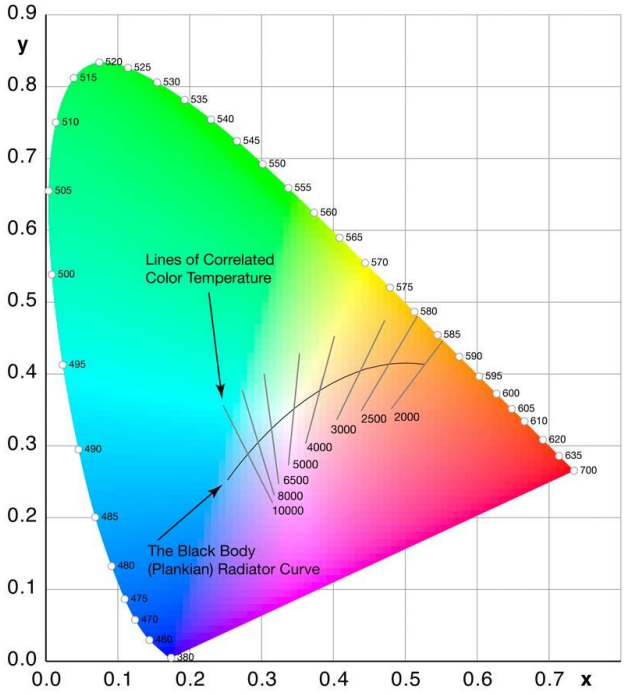

Рис. 1.1 - диаграмма цветности АЧТ

Рис. 1.2. - вертикальная шкала цветности АЧТ

Излучение АЧТ описывается законом Планка, который математически моделирует зависимость интенсивности излучения от частоты и температуры. Закон Планка позволяет нам точно предсказывать спектр излучения АЧТ для любой заданной температуры.

Существует несколько моделей АЧТ. Простейшая из них — это полость с маленьким отверстием (рис. 1.3). Излучение, проникающее в полость через отверстие, многократно отражается от ее стенок, пока не поглощается полностью. При этом отверстие само излучает как абсолютно черное тело, поскольку оно испускает все поглощенное излучение.

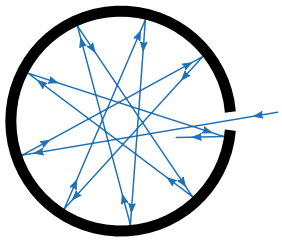

Рис 1.3 – Простейшая модель АЧТ

Модель абсолютно чёрного тела нашла свое применение в астрономии. Так, звезды, хотя и не являются идеальными АЧТ, приближаются к этой модели, излучая свет по закону Планка, но с некоторыми отклонениями. Измеряя спектр звездного излучения, астрономы могут определить их температуру, а также получать информацию о составе и структуре звезд.

Излучаемая энергия абсолютно черного тела — это суммарная энергия, излучаемая телом за единицу времени. Она зависит от температуры и площади поверхности объекта.

Светимость — это мощность излучения АЧТ, то есть скорость излучения энергии. Светимость согласно формуле Стефана-Больцмана пропорциональна четвертой степени температуры и площади поверхности тела.

1. Постановка задачи

Абсолютно черное тело излучает энергию пропорционально четвертой степени температуры.

$$E = 36.9 * 10^{-12} T^4$$

(E -мощность излучения в вт/см, T-температура в градусах Кельвина).

Часть общей энергии, заключенная в видимом спектре частот, с длиной волны от λ_1 до λ_2 находится интегрированием уравнения Планка:

$$E_{\text{видимая}} = E_{\text{в}} = \int_{\lambda_1}^{\lambda_2} \frac{2.39 * 10^{-11} dx}{x^5 \left(e^{\frac{1.432}{Tx}} - 1\right)}$$

Тогда светимость (в процентах) рассчитывается по формуле:

$$EFF = \frac{E_{\text{в}}}{E} * 100\% = \frac{64.77}{T^4} * \int_{\lambda_1}^{\lambda_2} \frac{dx}{x^5 \left(e^{\frac{1.432}{Tx}} - 1\right)}$$

Значения длин волн (λ_1, λ_2) задаются следующим образом:

$$\text{Значение } \lambda_1 = y * 31.66675 * 10^{-5},$$

$$\text{где } y - \text{ корень уравнения: } 2 * \sqrt{x} = \cos\frac{\pi * x}{2}$$

$$\text{Значение } \lambda_2 = -z * 3.039830 * 10^{-5},$$

$$\text{где } z - \text{ значение, минимизирующее } f(z) \text{ на } [-2, -1]:$$

$$f(z) = (e^z) * (2 * z^2 - 4) + (2 * z^2 - 1)^2 + e^{2*z} - 3 * z^4$$

Вычислить EFF в диапазоне температур от T=1000K до T=9000K с шагом 1000K. По полученным данным *построить график* светимости от температуры, взяв достаточное количество точек. *Оценить погрешность* результата и *влияние* на точность погрешности в задании λ_1 и λ_2.

2. Нахождение длин волн

Для вычислений, в том случае, когда $\lambda_2 < \lambda_1$ нам придется поменять лямбды местами, чтобы лямбда первая была меньше лямбды второй. Иначе рискуем найти отрицательную светимость.

Для решения поставленной задачи воспользуемся такими инструментами, как Python и Wolfram Alpha. А в частности, для нахождения длин волн, библиотекой scipy.

Импортируем необходимые модули и библиотеки, зададимся точностью $accuracy = 10^{-16}$. Также напишем сами функции f(x) и f(z).

```
import matplotlib.pyplot as plt
import scipy.optimize as optimize
from scipy.integrate import quad
import numpy as np
from math import sqrt, cos, pi

accuracy = 1e-16


def fun_x(x):
    return 2 * sqrt(x[0]) - cos((pi * x[0]) / 2)


def fun_z(z):
    return (np.e ** z) * (2 * z ** 2 - 4) + (2 * z ** 2 - 1) ** 2 + np.e ** (2 *
z) - 3 * z ** 4
```

Найдем x используя метод scipy.optimize.fsolve, который находит корни уравнений func(x) = 0 Под капотом fsolve использует метод поиска с переменной точкой, реализованный в алгоритмах hybrd и hybrj из MINPACK и по сути является их оберткой.

Для работы fsolve также требуется начальное приближение, возьмем число 0 в виду того, что квадратный корень из x определен только на положительном промежутке, а косинус в нуле равен единице и с увеличением x будет уменьшаться. Получаем:

```
res_x = optimize.fsolve(fun_x, np.array([0]))
print(res_x)
# [0.22105064]
```

Получаем решение: x = 0.22105064

Найдем значение z, минимизирующее f(z) на промежутке [-2,-1]. Для этого воспользуемся методом scipy.optimize.minimize, а в частности методом ‘TNC’ (Truncated Newton), который сочетает в себе преимущества метода Нелдера-Мида

(метод симплексного поиска) и метода сопряженных градиентов, однако все также требует начального приближения. В данном случае зададим границы [-2, -1], а начальное приближение выберем в качестве среднего арифметического этих границ.

```
lower_bound = -2
upper_bound = -1
bounds = optimize.Bounds(lower_bound, upper_bound)
res_z = optimize.minimize(fun_z,
                          np.array((lower_bound + upper_bound) / 2),
                          bounds=bounds,
                          method='TNC',
                          tol=accuracy).x
print(res_z)
# [-1.31597378]
```

Получаем минимизирующее значение: z = -1.31597378

Вычислим полученные лямбды по формулам. Заметим, что для того, чтобы наши расчеты имели какой-то смысл, λ_1 должна быть меньше λ_2.
Поэтому отсортируем их в порядке возрастания.

```
L_1 = res_x[0] * 31.66675 * 10 ** (-5)
L_2 = -res_z[0] * 3.039830 * 10 ** (-5)
L_1, L_2 = sorted([L_1, L_2])
print(f'TEST: lambda_1 = {L_1}; lambda_2 = {L_2}')

# lambda_1 = 4.000336578604378e-05
# lambda_2 = 6.999955338251561e-05
```

В результате:

$$\lambda_1 = 4.000336578604378 * 10^{-5}$$
$$\lambda_2 = 6.999955338251561 * 10^{-5}$$

Полученные значения действительно находятся в видимом диапазоне и представляют собой длины волн красного и фиолетового цвета соответственно (см. рис. 2.1).

Цвет	Диапазон длин волн, нм
Фиолетовый	380—450
Синий	450—480
Голубой	480—510
Зелёный	510—550
Салатовый	550—570
Жёлтый	570—590
Оранжевый	590—630
Красный	630—780

Рис. 2.1 Таблица характеристики длин волн видимого диапазона [2]

Для повторного поиска лямбд напишем функцию, благодаря которой мы сможем находить лямбды с разной точностью.

```
def get_lambdas(fun_x, fun_z, accuracy=1e-16):
    res_x = optimize.fsolve(fun_x, np.array([0]), xtol=accuracy)
    print(f'INFO: res_x = {res_x}')
    # [0.22105064]

    lower_bound = -2
    upper_bound = -1
    bounds = optimize.Bounds(lower_bound, upper_bound)
    res_z = optimize.minimize(fun_z,
                              np.array((lower_bound + upper_bound) / 2),
                              bounds=bounds,
                              method='TNC',
                              tol=accuracy).x
    print(f'INFO: res_z = {res_z}')
    # [-1.3159745]
    L_1 = res_x[0] * 31.66675 * 10 ** (-5)
    L_2 = -res_z[0] * 3.039830 * 10 ** (-5)
    L_1, L_2 = sorted([L_1, L_2])
    print(f'INFO: lambda_1 = {L_1}; lambda_2 = {L_2}')
    # lambda_1 = 4.000336578604378e-05
    # lambda_2 = 6.999955338251561e-05
    return L_1, L_2
```

3. Исследование зависимости светимости от температуры и погрешности заданных длин волн

Для поиска светимости EFF требуется взять интеграл. Для этого воспользуемся методом quad из библиотеки scipy.integrate. “quad” использует адаптивные алгоритмы для аппроксимации определенного интеграла функции. То есть он разбивает область интегрирования на мелкие интервалы и вычисляет приближенное значение интеграла на каждом интервале. Также он может принимать дополнительные аргументы для функции интегрирования, которые можно передать с помощью параметра args, в нашем случае переменную - T.

```
def integrand(x, T):
    return 1 / ((x ** 5) * (np.exp(1.432 / (T * x)) - 1))


def EFF(T, l_bound, u_bound):
    return (
            64.77 / (T ** 4) * quad(integrand, l_bound, u_bound, args=T,
epsabs=accuracy)[0]
    )
```

Для тестирования возьмём значение T = 1000

```
T = 1000.0
lambda_1, lambda_2 = get_lambdas(fun_x, fun_z, accuracy)
print(
    f'TEST:\n!wa integrate from {lambda_1} to {lambda_2} dx /
((x^5)(e^((1.432)/{T} * x) - 1) = {quad(integrand, lambda_1, lambda_2, args=T,
epsabs=accuracy)[0]}')
```

Для проверки попробуем взять тот же интеграл с помощью WolframAlpha:

Введя: integrate from 4.000336578604378*10^(-5) to 6.999955338251561*10^(-5): dx/((x^5) * (e^(1.432 / (1000 * x)) - 1))

Получаем:

WolframAlpha	3085873.2325267694
Quad	3085873.232526763

Было достигнуто совпадение до 7 цифры после запятой, поэтому продолжим использовать “quad” для всего диапазона T.

```
def calculate_EFF(lambda_1, lambda_2, label='EFF'):
    ans = []
    T0 = 1000
    step = 1000
```

```
    for i in range(9):
        T = T0 + i * step
        # t_res = EFF(T, lambda_1, lambda_2)
        ans.append(EFF(T, lambda_1, lambda_2))

    T_values = list(range(T0, T0 + step * len(ans), step))

    return np.array(ans), T_values
```

Вычислим светимость EFF с исходными значениями λ

```
lambda_1, lambda_2 = get_lambdas(fun_x, fun_z, accuracy)
# Вычислим EFF с исходными значениями lambda
original_EFF, T_values = calculate_EFF(lambda_1, lambda_2)
```

```
INFO: res_x = [0.22105064]
INFO: res_z = [-1.31597378]
INFO: lambda_1 = 4.000336578604378e-05; lambda_2 = 6.999955338251561e-05
```

```
print(f'INFO: original EFF: {original_EFF}')
```
Executed at 2024.05.26 19:51:24 in 5ms

```
INFO: original EFF: [1.99872009e-04 8.05765806e-01 8.25492052e+00 2.10610037e+01
 3.18215384e+01 3.76797112e+01 3.93061363e+01 3.82222087e+01
 3.56993964e+01]
```

Рис. 3.1 Результаты вычисления светимости

Теперь зададимся некоторым случайным отклонением от исходных значений λ

```
delta = 1e-6  # Вместо дельты может быть погрешностью измерений прибора или ошибка округления, поэтому мы также добавим случайность в уравнение.
# delta default: 1e-6
accuracy = 1e-16
 # Получаем
new_lambdas = get_lambdas(fun_x, fun_z, accuracy)
lambda_1_new = new_lambdas[0] + delta * np.random.uniform(-1, 1)  # lambda_1 * 0.01
lambda_2_new = new_lambdas[1] + delta * np.random.uniform(-1, 1)  # lambda_2 * 0.01

print(f"INFO: Original lambda_1: {lambda_1}, Original lambda_2: {lambda_2}")
print(f"INFO: New lambda_1: {lambda_1_new}, New lambda_2: {lambda_2_new}")
print(f"INFO: ABS Difference in lambda_1: {np.abs(lambda_1 - lambda_1_new)}")
print(f"INFO: ABS Difference in lambda_2: {np.abs(lambda_2 - lambda_2_new)}")
```

```
INFO: res_x = [0.22105064]
INFO: res_z = [-1.31597378]
INFO: lambda_1 = 4.000336578604378e-05; lambda_2 = 6.999955338251561e-05
INFO: Original lambda_1: 4.000336578604378e-05, Original lambda_2: 6.999955338251561e-05
INFO: New lambda_1: 4.0470844747097126e-05, New lambda_2: 7.047317667889374e-05
INFO: ABS Difference in lambda_1: 4.6747896105334824e-07
INFO: ABS Difference in lambda_2: 4.736232963781272e-07
```

Рис. 3.2 Значение новых λ и их абсолютная разница

Вычислим EFF для новых значений $\boldsymbol{\lambda}$, а также абсолютную разность между оригинальным значением и новым значением EFF:

```
new_EFF, T_values2 = calculate_EFF(lambda_1_new, lambda_2_new, label='New EFF')
```

```
diff_EFF = np.abs(original_EFF - new_EFF)
div_diff_EFF = diff_EFF / original_EFF * 100
```

```
print(f"INFO: Original EFF: {original_EFF}")
Executed at 2024.05.26 21:13:20 in 5ms
```

```
INFO: Original EFF: [1.99872009e-04 8.05765806e-01 8.25492052e+00 2.10610037e+01
 3.18215384e+01 3.76797112e+01 3.93061363e+01 3.82222087e+01
 3.56993964e+01]
```

```
print(f"INFO: New EFF: {new_EFF}")
Executed at 2024.05.26 21:13:20 in 3ms
```

```
INFO: New EFF: [2.24969758e-04 8.47371052e-01 8.47796194e+00 2.13389193e+01
 3.19439740e+01 3.75692472e+01 3.89904007e+01 3.77645742e+01
 3.51609588e+01]
```

```
print(f"INFO: Difference in EFF: {diff_EFF}")
Executed at 2024.05.26 21:13:20 in 3ms
```

```
INFO: Difference in EFF: [2.50977491e-05 4.16052459e-02 2.23041421e-01 2.77915610e-01
 1.22435603e-01 1.10463954e-01 3.15735689e-01 4.57634425e-01
 5.38437603e-01]
```

```
print(f"INFO: div_diff_EFF: {div_diff_EFF}")
Executed at 2024.05.26 21:13:20 in 3ms
```

```
INFO: div_diff_EFF: [12.55691038  5.16344148  2.70192088  1.31957439  0.38475702  0.29316561
  0.80327327  1.19729979  1.50825408]
```

Рис 3.3 Значения исходной светимости EFF, новой светимости для измененных $\boldsymbol{\lambda}$, их абсолютная разница и относительная абсолютная разница.

Теперь, когда нам известны все необходимые данные, мы можем нарисовать их графики. Для этого воспользуемся библиотекой matplotlib и модулем pyplot. Результаты работы программы при вышеописанных данных проиллюстрированы на рисунках 3.4, 3.5, 3.6. Однако они могут меняться от запуска к запуску, ввиду случайности значения новых $\boldsymbol{\lambda}$.

По рис. 3.4 видно, что максимум светимости достигается при температуре T=7000K и равен 39.3%. Для светимости с введенной погрешностью значение максимума тоже достигается при данной температуре. Это объясняет, почему дневной свет, имеющий близкую цветовую температуру (рис.1.2), кажется нам наиболее ярким.

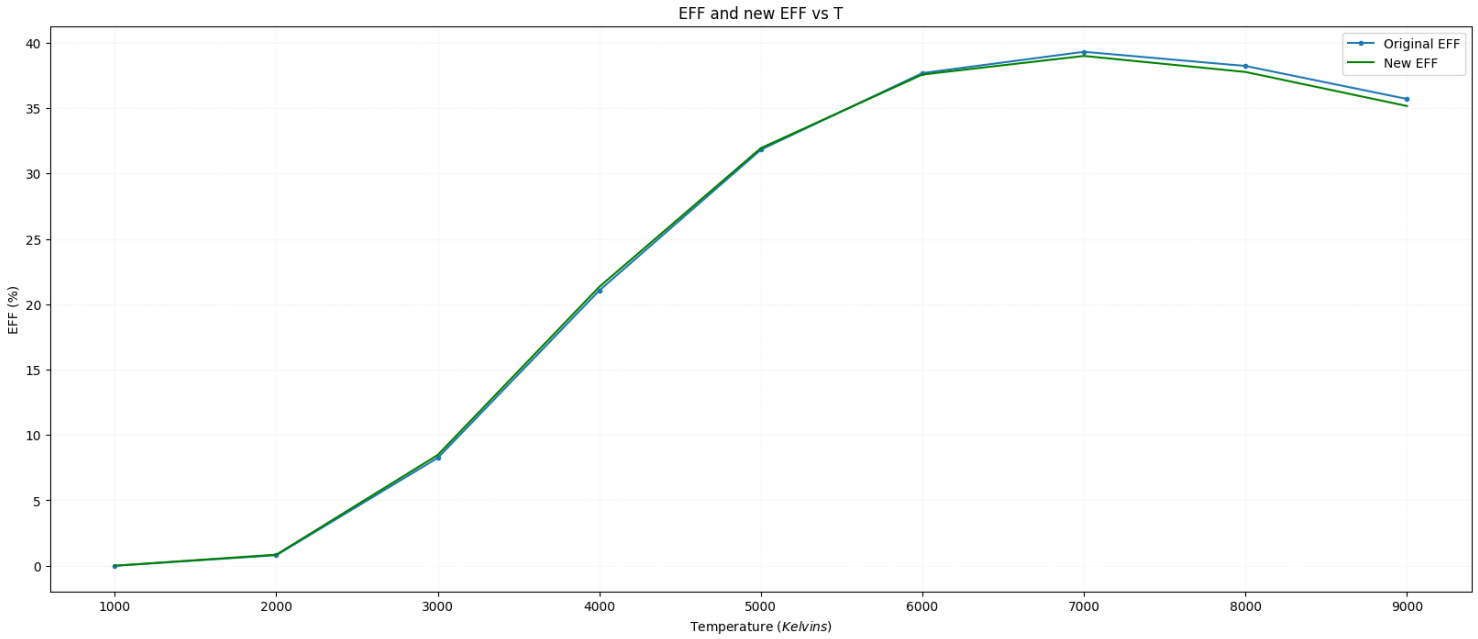

Рис. 3.4 Сравнительный график значений оригинальной светимости (*Original EFF*) и новой светимости (*New EFF*) в зависимости от температуры

При значении delta=10^{-6} и accuracy=10^{-16} тенденция абсолютная погрешности результата такова, что она невелика при малых температурах, однако с их увеличением она соответственно возрастает.

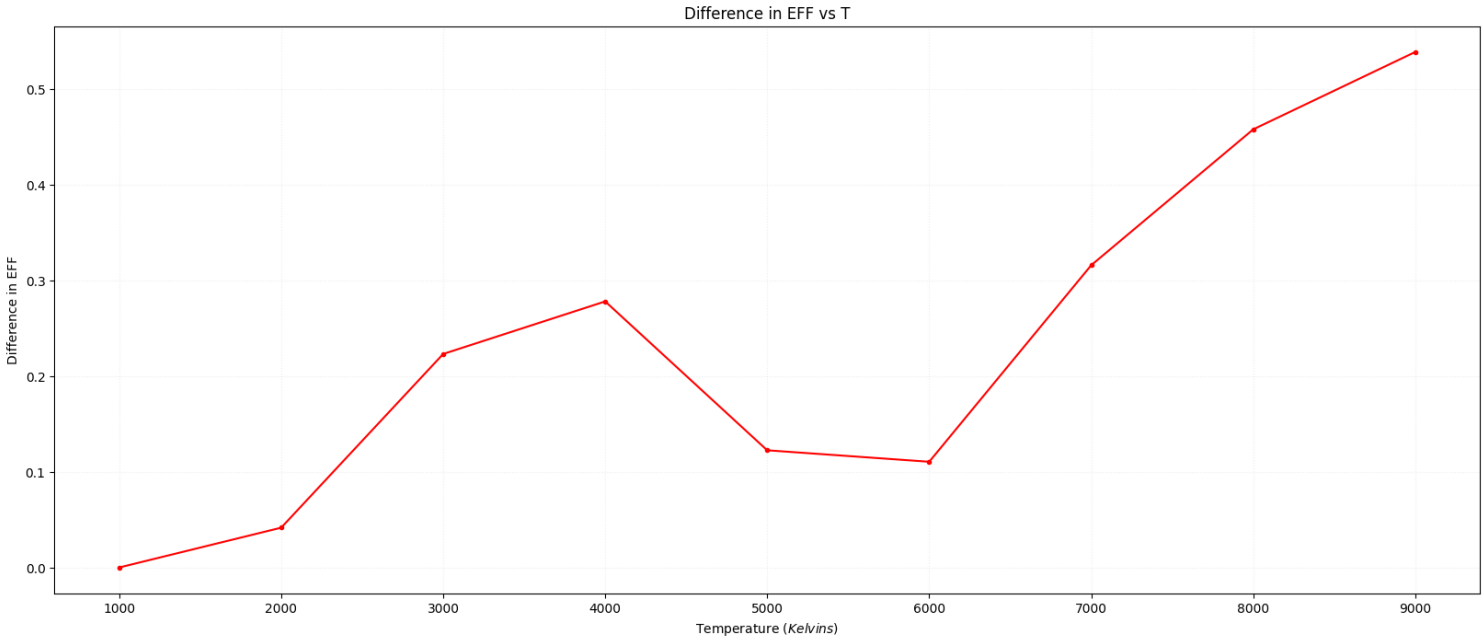

Рис. 3.4 График абсолютной погрешности в зависимости от температуры

Изменение в погрешности вычисления лямбд, при максимальной разности погрешности в 10^{-6} даёт высокую относительную абсолютную погрешность для малых температур – 12.55%, когда значение оригинальной светимости слишком мало. Однако, оно постепенно уменьшается до 1–2% при более высоких температурах.

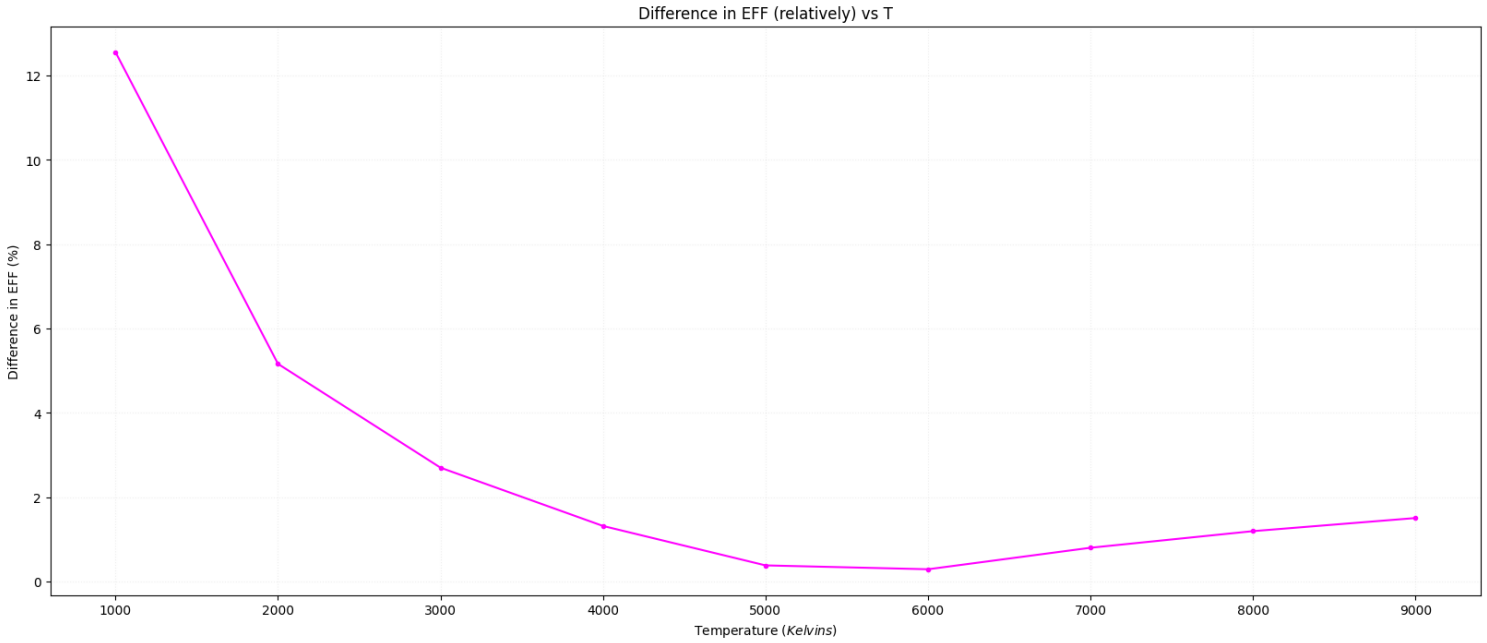

Рис. 3.4 Относительная абсолютная погрешность в зависимости от температуры

ЗАКЛЮЧЕНИЕ

В результате данной работы была исследована светимость абсолютно черного тела в видимом человеку диапазоне.

Полученное значение светимости в равное 39.3% при температуре 7000K может свидетельствовать о том, почему дневной свет с близкой цветовой температурой кажется нам наиболее ярким. Относительная абсолютная погрешность для данной температуры составляет не более 2%. Это говорит нам о том, что полученные результаты моделирования достаточно точны и достоверны. Что может означать практическую применимость при проектировании систем освещения: например, лампа с цветовой температурой 7000K может быть оптимальна для рабочих мест, фотосъемки и других областей, где требуется яркий и нейтральный свет.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ГОСТ 7.32-2001. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления

2. Hunt R. W. C. The Reproduction of Colour. — 6th edition. — John Wiley & Sons, 2004. — P. 4—5. — 724 p. — ISBN 978-0-470-02425-6

3. Абсолютно чёрное тело // Большой энциклопедический политехнический словарь. — 2004.

ПРИЛОЖЕНИЕ

Jupyter notebook:
https://colab.research.google.com/drive/1fobS6dlVt4u7gteZJ1jGVdz_dDoDCFIX?usp=sharing

Листинг кода

```
"""
Код решающий задачу о нахождении светимости (в процентах) при заданной температуре и заданных
lambda 1 и lambda 2
Код хорошо задокументирован в самой курсовой работе, а также в соответствующим юпитер блокноте

https://colab.research.google.com/drive/1fobS6dlVt4u7gteZJ1jGVdz_dDoDCFIX?usp=sharing

"""
import matplotlib.pyplot as plt
import scipy.optimize as optimize
from scipy.integrate import quad
import numpy as np
from math import sqrt, cos, pi

accuracy = 1e-16


def fun_x(x):
    return 2 * sqrt(x[0]) - cos((pi * x[0]) / 2)


def fun_z(z):
    return (np.e ** z) * (2 * z ** 2 - 4) + (2 * z ** 2 - 1) ** 2 + np.e ** (2 * z) - 3 * z ** 4


def get_lambdas(fun_x, fun_z, accuracy=1e-16):
    res_x = optimize.fsolve(fun_x, np.array([0]), xtol=accuracy)
    print(f'INFO: res_x = {res_x}')
    # [0.22105064]

    lower_bound = -2
    upper_bound = -1
    bounds = optimize.Bounds(lower_bound, upper_bound)
    res_z = optimize.minimize(fun_z,
                              np.array((lower_bound + upper_bound) / 2),
                              bounds=bounds,
                              method='TNC',
                              tol=accuracy).x
    print(f'INFO: res_z = {res_z}')
    # [-1.3159745]
    L_1 = res_x[0] * 31.66675 * 10 ** (-5)
```

```
    L_2 = -res_z[0] * 3.039830 * 10 ** (-5)
    L_1, L_2 = sorted([L_1, L_2])
    print(f'INFO: lambda_1 = {L_1}; lambda_2 = {L_2}')

    # lambda_1 = 4.00033875088364e-05
    # lambda_2 = 6.999955338251561e-05
    return L_1, L_2


def integrand(x, T):
    return 1 / ((x ** 5) * (np.exp(1.432 / (T * x)) - 1))


def EFF(T, l_bound, u_bound):
    return (
            64.77 / (T ** 4) * quad(integrand, l_bound, u_bound, args=T,
epsabs=accuracy)[0]
    )


def calculate_EFF(lambda_1, lambda_2, label='EFF'):
    ans = []
    T0 = 1000
    step = 1000
    for i in range(9):
        T = T0 + i * step
        # t_res = EFF(T, lambda_1, lambda_2)
        ans.append(EFF(T, lambda_1, lambda_2))

    T_values = list(range(T0, T0 + step * len(ans), step))

    return np.array(ans), T_values


lambda_1, lambda_2 = get_lambdas(fun_x, fun_z, accuracy)
# Вычислим EFF с исходными значениями lambda
original_EFF, T_values = calculate_EFF(lambda_1, lambda_2)

# Создадим некоторое отклонение от исходных значений lambda
delta = 1e-6  # Вместо дельты может быть погрешностью измерений прибора или ошибка
округления, поэтому мы также
# добавим случайность в уравнение.
# delta default: 1e-6
accuracy = 1e-16
# Получаем
new_lambdas = get_lambdas(fun_x, fun_z, accuracy)
lambda_1_new = new_lambdas[0] + delta * np.random.uniform(-1, 1)  # lambda_1 * 0.01
lambda_2_new = new_lambdas[1] + delta * np.random.uniform(-1, 1)  # lambda_2 * 0.01

print(f"INFO: Original lambda_1: {lambda_1}, Original lambda_2: {lambda_2}")
print(f"INFO: New lambda_1: {lambda_1_new}, New lambda_2: {lambda_2_new}")
print(f"INFO: Difference in lambda_1: {np.abs(lambda_1 - lambda_1_new)}")
print(f"INFO: Difference in lambda_2: {np.abs(lambda_2 - lambda_2_new)}")
# Вычислим EFF для новых значений лямбда
new_EFF, T_values2 = calculate_EFF(lambda_1_new, lambda_2_new, label='New EFF')
```

```python
# Вычисляем разность между оригинальным значением and новым значением EFF
diff_EFF = np.abs(original_EFF - new_EFF)
div_diff_EFF = diff_EFF / original_EFF * 100

# График EFF и new EFF
plt.figure(figsize=(10, 6))
plt.plot(T_values, original_EFF, marker='1')
plt.title('EFF and new EFF vs T')
plt.xlabel('Temperature' + r' ($Kelvins$)')
plt.ylabel('EFF (%)')
plt.plot(T_values2, new_EFF, marker='x', color='green')
plt.legend(['Original EFF', 'New EFF'])
plt.grid(True, alpha=0.2, linestyle=':')
plt.show()

print(f"INFO: Original EFF: {original_EFF}")
print(f"INFO: New EFF: {new_EFF}")
print(f"INFO: Difference in EFF: {diff_EFF}")
print(f"INFO: div_diff_EFF: {div_diff_EFF}")

# График модуля отклонения
plt.figure(figsize=(10, 6))
plt.plot(T_values, diff_EFF, marker="1", color='red')
plt.title('Difference in EFF vs T')
plt.xlabel('Temperature' + r' ($Kelvins$)')
plt.ylabel('Difference in EFF')
plt.grid(True, alpha=0.2, linestyle=':')
plt.show()

# Относительная погрешность
plt.figure(figsize=(10, 6))
plt.plot(T_values, div_diff_EFF, marker='.', color='magenta')
plt.title('Difference in EFF (relatively) vs T')
plt.xlabel('Temperature' + r' ($Kelvins$)')
plt.ylabel('Difference in EFF (%)')
plt.grid(True, alpha=0.2, linestyle=':')
plt.show()
```