The Stefan-Boltzmann Radiation Law

Burak Kara
Boğaziçi University • Physics Department

***Abstract*—The Stefan-Boltzmann Law states that the thermal emission is proportional to the fourth power of the temperature. The emitted radiation is reduced by the inverse-square law depending on the distance from the radiator. In this experiment, both the inverse-square law of thermal emission and the Stefan-Boltzmann Law are tested using the Stefan-Boltzmann Lamp for high temperatures and an electrical oven for low temperatures. It is found that the radiation is reduced by the -1.94 $\pm$ 0.18 power of the distance, which is 0.33 σ away from the true value -2. The emission is found to be proportional to the 4.00 $\pm$ 0.064 power of temperature, which is 0σ away from the true value 4.**

I. Introduction

In 1877, Josef Stefan deduced the relationship between thermal emission and temperature in his article "Über die Beziehung zwischen der Wärmestrahlung und der Temperatur (On the relationship between thermal radiation and temperature)" based on the experiments by Irish Physicist John Tyndall [1]. He found that the radiation is proportional to the fourth power of the absolute temperature. In 1884, this empirical law was later calculated theoretically by Ludwig Boltzmann in his article "Ableitung des Stefan'schen Gesetzes, betreffend die Abhängigkeit der Wärmestrahlung von der Temperatur aus der electromagnetischen Lichttheorie (Derivation of Stefan's law, concerning the dependence of thermal radiation on temperature, from the electromagnetic theory of light)"[2]. The Law is therefore named after Josef Stefan and Ludwig Boltzmann for their contribution.

The Stefan-Boltzmann Law describes how thermal radiation changes with a varying temperature. In the ideal case, a radiator emits light directly proportional to its fourth temperature power. Also, since the radiation obeys the inverse-square law(See. Theory), the Stefan-Boltzmann Law can calculate bodies' temperature from their intensity. One of the examples is the temperature of the sun or other stars. Also, from the same relations radius of a star can be found[3]. In this experiment, the inverse-square and Stefan-Boltzmann laws are tested and validated.

II. Theory

This section gives the derivation of the Stefan-Boltzmann Law and the inverse-square law.

Energy of a photon:

$$\epsilon = \hbar\omega$$

Density of States for a Photon Gas:

$$g_{dos} \cdot dw = dN_{states} = \frac{Vw^2}{\pi^2 c^3}$$

Energy of the Photon Gas:

$$E = \int_0^\infty \frac{\hbar w g_{dos}(w)}{e^{\frac{\hbar w}{kT}} - 1}\, dw = \frac{VT^4k^4\pi^2}{15\hbar^3c^3}.$$

Energy Density of the Photon Gas:

$$\frac{E}{V} = \frac{T^4k^4\pi^2}{15\hbar^3c^3} = \frac{4\sigma}{c}T^4 \tag{1}$$

where $\hbar$ is reduced Planck Constant,
ω is angular frequency,
V is volume, c is the speed of the light,
T is the absolute temperature,
k is the Boltzmann Constant,
and σ is the Stefan Boltzmann Constant.

Equation 1 gives the Stefan Boltzmann Law. In the theory, it is assumed that the photons are radiated from every direction with every frequency. Therefore, there is no polarization of the photon gas. Since the radiation is measured on the surface of a sphere (See. Fig1) and the photon gas is not polarized, the intensity is:

$$I = \frac{P}{A} = \frac{P}{4\pi r^2} \tag{2}$$

where P is power, and A is the surface area.

Equation 2 gives the inverse square law of a source.

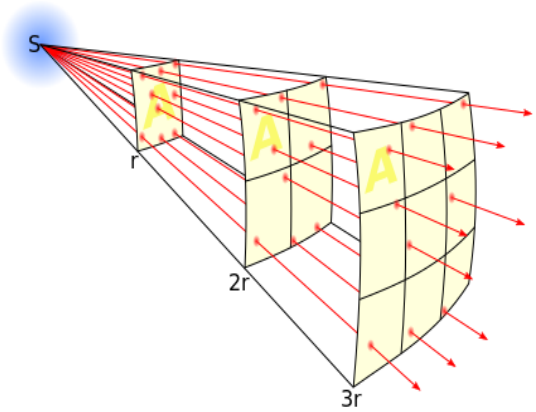

Fig. 1. Inverse Square Law.[4].

III. The Experimental Setup & Method

For the experiment, the following apparatus is used:

- Stefan Boltzmann Lamp.,
- Radiation Sensor,
- Digital Multimeter,
- Power Supply,
- Voltage Amplifier,
- Optical Bench,
- Electrical Oven,
- Thermometer,
- Connecting Leads,
- 1 $k\Omega$ resistor.

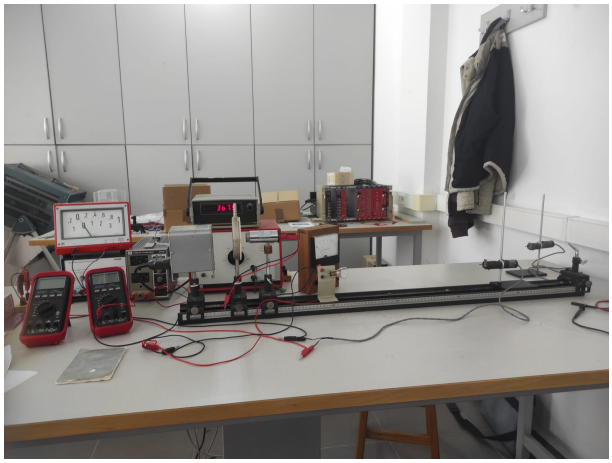

Fig. 2. Apparatus.

The experiment consists of 4 parts.

A. Determination of R_{300K}

From the Ohm's Law, one can calculate the resistance. However, since the resistance is also temperature-dependent, one should be careful about the applied volt. The resistance of a Stefan Boltzmann Lamp is calculated from Ohm's Law. The voltage and current readings are listed in Table.I.

1) Voltage is applied in the range 1 - 15 mV,
2) Voltage and current values are read.

B. Inverse Square Law

The radiation sensor is used to detect the thermal radiation. The obtained data is listed in the Table.III.

1) Voltage is applied to the Stefan Boltzmann Lamp to generate radiation,
2) Background radiation is read to obtain non-varying radiation (due to environment) and worked on that region,
3) The background radiation is eliminated by changing the offset,
4) The sensor voltage amplified by 10^5,
5) The Lamp is re-positioned to obtain different readings of sensor voltage.

C. High Temperature Measurement

The obtained data is listed in the Table.IV.

1) Voltage is applied to the Stefan Boltzmann Lamp to generate radiation,
2) The Stefan Boltzman Lamp is placed such that the background radiation does not vary due to the environment,
3) The voltage readings are amplified by 10^4,
4) The applied voltage and current are changed to obtain different readings in the sensor,
5) The Sensor Voltage, Lamp Voltage, and Lamp Current are recorded.

D. Low Temperature Measurements

An electrical oven is used to generate radiation. Temperature and Sensor Voltage values are recorded. The obtained data is listed in the Table.V.

1) The radiation sensor is placed in front of the oven,
2) The oven is heated up to 650 K while recording the temperature and sensor voltage simultaneously.

IV. The Data

The measurement is listed in this section.

TABLE I
Reference R_{300K} Measurement

Current (mA)	Voltage (mV)
5. ± 1.	1.30 ± 0.07
10.± 1.	2.8 ± 0.1
15.± 1.	4.3 ± 0.2
20.± 1.	5.8 ± 0.3
25.± 1.	7.3 ± 0.4
30.± 1.	9.0 ± 0.5
35.± 1.	10.3 ± 0.5
40.± 1.	12.1 ± 0.6
45.± 1.	13.7 ± 0.7

It is seen that the readings of background radiation become constant after $6.2 \pm 0.1cm$ distance. Therefore, the measurements are taken after that point.

TABLE II
TEMPERATURE AND CORRESPONDING R/R_{300K} VALUES[5]

Temperature (K)	R/R_{300K}
300 ± 100	1.00 ± 0.01
400 ± 100	1.43 ± 0.01
500 ± 100	1.87 ± 0.01
600 ± 100	2.34 ± 0.01
700 ± 100	2.85 ± 0.01
800 ± 100	3.36 ± 0.01
900 ± 100	3.88 ± 0.01
1000 ± 100	4.41 ± 0.01
1100 ± 100	4.95 ± 0.01
1200 ± 100	5.48 ± 0.01
1300 ± 100	6.03 ± 0.01
1400 ± 100	6.58 ± 0.01
1500 ± 100	7.14 ± 0.01
1600 ± 100	7.71 ± 0.01
1700 ± 100	8.28 ± 0.01
1800 ± 100	8.86 ± 0.01
1900 ± 100	9.44 ± 0.01
2000 ± 100	10.03 ± 0.01
2100 ± 100	10.63 ± 0.01
2200 ± 100	11.24 ± 0.01
2300 ± 100	11.84 ± 0.01
2400 ± 100	12.46 ± 0.01
2500 ± 100	13.08 ± 0.01
2600 ± 100	13.72 ± 0.01
2700 ± 100	14.34 ± 0.01
2800 ± 100	14.99 ± 0.01
2900 ± 100	15.63 ± 0.01
3000 ± 100	16.29 ± 0.01
3100 ± 100	16.95 ± 0.01
3200 ± 100	17.62 ± 0.01
3300 ± 100	18.28 ± 0.01
3400 ± 100	18.97 ± 0.01
3500 ± 100	19.66 ± 0.01
3600 ± 100	26.35 ± 0.01

TABLE III
INVERSE SQUARE LAW MEASUREMENTS
SOURCE VOLTAGE = 6. ± 1 V
SOURCE CURRENT = 2.0 ± .01 A
VOLTAGE AMPLIFICATION = 10^5.

Distance (cm)	Voltage(V)
42.7 ± 0.2	0.2 ±0.2
33.6 ± 0.2	0.4 ±0.2
26.9 ± 0.2	0.6 ±0.2
22.8 ± 0.2	0.8 ±0.2
20.7 ± 0.2	1.0±0.2
18.9 ± 0.2	1.2±0.2
17.6 ± 0.2	1.4±0.2
16.4 ± 0.2	1.6±0.2
15.5 ± 0.2	1.8±0.2
14.8 ± 0.2	2.0±0.2
14.1 ± 0.2	2.2±0.2
13.6 ± 0.2	2.4±0.2
12.5 ± 0.2	2.6±0.2
12.2 ± 0.2	2.8±0.2

TABLE IV
HIGH TEMPERATURE MEASUREMENTS
DISTANCE: 9.1 $\pm 0.1cm$.
VOLTAGE AMPLIFICATION: 10^4.

Signal Voltage (V)	Lamp Voltage (V)	Lamp Current (A)
3.0 ±0.2	8.41 ± 0.45	2.487 ± 0.055
2.6 ±0.2	7.39 ± 0.40	2.323 ± 0.051
2.2 ±0.2	6.95 ± 0.38	2.255 ± 0.050
2.0 ±0.2	6.06 ± 0.33	2.099 ± 0.047
1.8 ±0.2	5.88 ± 0.32	2.069 ± 0.046
1.6 ±0.2	5.21 ± 0.29	1.942 ± 0.044
1.2 ±0.2	4.11 ± 0.24	1.725 ± 0.040
0.8 ±0.2	2.84 ± 0.15	1.440 ± 0.034
0.6 ±0.2	2.47 ± 0.13	1.368 ± 0.032
0.4 ±0.2	0.645 ± 0.035	0.857 ± 0.022

TABLE V
LOW TEMPERATURE MEASUREMENTS

Voltage (V)	Temperature (^oC)
0.30 ± 0.41	55.7 ±0.1
1.300 ± 0.068	131.4 ±0.1
1.700 ± 0.088	149.1 ±0.1
2.20 ± 0.11	174.8 ±0.1
2.70 ± 0.14	196.4 ±0.1
3.00 ± 0.15	210.4±0.1
3.30 ± 0.17	219.8±0.1
3.40 ± 0.17	226.1±0.1
3.50 ± 0.18	227.6±0.1
3.60 ± 0.18	230.7±0.1
3.70 ± 0.19	234.2±0.1
3.80 ± 0.19	237.7±0.1
3.90 ± 0.20	241.6±0.1
4.00 ± 0.20	244.8±0.1
4.10 ± 0.23	247.0±0.1
4.20 ± 0.24	251.3±0.1
4.30 ± 0.24	253.4±0.1
4.40 ± 0.25	257.5±0.1
4.50 ± 0.26	261.6±0.1
4.60 ± 0.26	263.7±0.1
4.70 ± 0.27	265.9±0.1
4.80 ± 0.27	269.2±0.1
4.90 ± 0.28	271.5±0.1
5.00 ± 0.28	274.2±0.1
5.10 ± 0.29	276.8±0.1
5.20 ± 0.29	279.6±0.1
5.30 ± 0.30	282.6±0.1
5.40 ± 0.30	285.0±0.1
5.50 ± 0.31	288.2±0.1
5.60 ± 0.31	291.3±0.1
5.70 ± 0.31	294.2±0.1
5.80 ± 0.32	295.9±0.1
5.90 ± 0.33	298.8±0.1
6.00 ± 0.33	300.6±0.1
6.10 ± 0.33	303.6±0.1
6.20 ± 0.34	305.7±0.1
6.30 ± 0.34	308.6±0.1
6.40 ± 0.35	310.8±0.1
6.50 ± 0.35	313.4±0.1
6.60 ± 0.36	315.6±0.1
6.70 ± 0.36	318.2±0.1
6.80 ± 0.37	319.7±0.1
6.90 ± 0.38	322.4±0.1
7.00 ± 0.38	325.0±0.1
7.10 ± 0.39	327.2±0.1
7.20 ± 0.39	329.2±0.1
7.30 ± 0.40	331.4±0.1
7.40 ± 0.40	333.6±0.1
7.50 ± 0.41	335.8±0.1
7.60 ± 0.41	338.0±0.1
7.70 ± 0.42	339.8±0.1
7.80 ± 0.42	342.2±0.1
7.90 ± 0.43	344.4±0.1
8.00 ± 0.43	346.2±0.1
8.10 ± 0.44	348.6±0.1
8.20 ± 0.44	350.2±0.1
8.30 ± 0.44	352.8±0.1
8.40 ± 0.45	354.2±0.1
8.50 ± 0.45	356.7±0.1
8.60 ± 0.46	358.6±0.1
8.70 ± 0.46	361.0±0.1
8.80 ± 0.47	362.5±0.1
8.90 ± 0.47	364.9±0.1
9.00 ± 0.48	366.4±0.1
9.10 ± 0.48	368.4±0.1

V. THE ANALYSIS AND RESULTS

The following equation is considered for the error propagation, assuming there is no correlation between variables.

$$\sigma_f = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 \sigma_x^2 + \left(\frac{\partial f}{\partial y}\right)^2 \sigma_y^2 + \left(\frac{\partial f}{\partial z}\right)^2 \sigma_z^2 + \cdots} \quad (11)$$

The uncertainties of the voltage and current readings (Table.I's Voltage readings, Table.III's Voltage readings, Table.IV's Lamp Voltage and Lamp Current readings, Table.V's Voltage readings.) are determined by the multimeter's manual. The other readings' uncertainties are taken as the least significant figures. For the implementation of the error propogation see Appendix.

A. *Determination of* R_{300K}

To find the resistance of the lamp, the tableI is used. A line is fitted to the data points using ROOT(See. Appendix). From Ohm's Law $V = I \cdot R$, the resistance(R) can be found by the slope of the line.

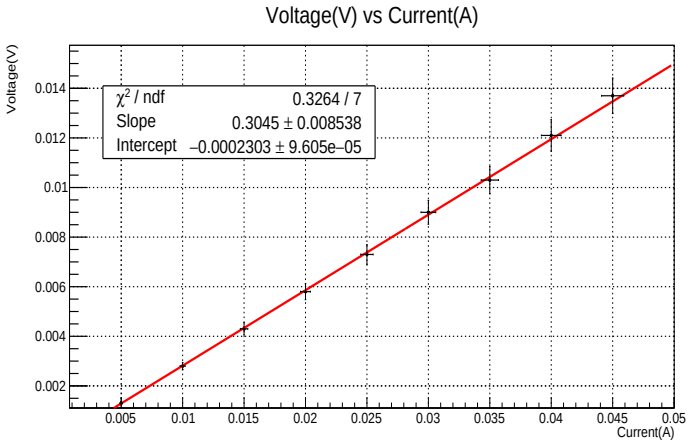

Fig. 3. Determination of R_{300K} - V vs I Graph at room temperature(T ≈ 300K)

As can be seen from the Fig.3, χ^2 value is reasonable. The resistance of the lamp is found to be: R_{300K} = 0.3045 ± 0.0085.

B. *Inverse-Square Law*

As Eq.2 suggests, the intensity is proportional to the inverse square of the radius. From the Table.III, an exponential fit is made (See. Fig.4). Also, at the log scale, the same data should give a line fit. This behavior also can be seen in Fig.5.

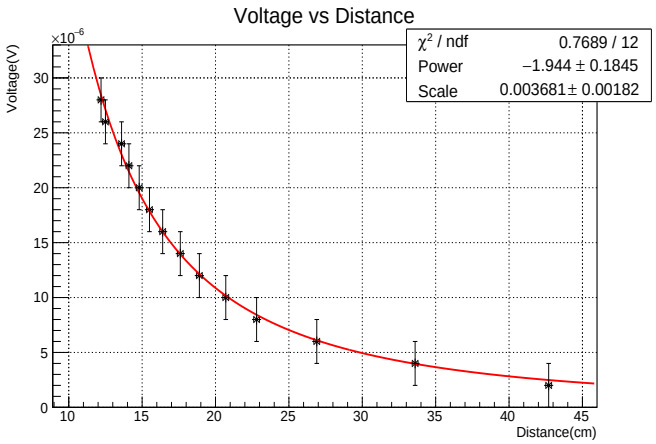

Fig. 4. Inverse Square Law - Exponential Fit.

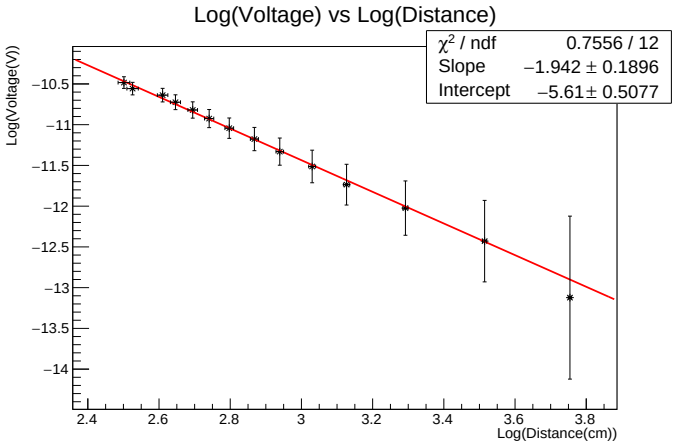

Fig. 5. Inverse Square Law - Log Scale Line Fit.

From Fig.4 and Fig.5, it can be seen that the χ^2 values are reasonable. Therefore, the power of the distance is -1.94 $\pm$ 0.19, and -1.94 $\pm$ 0.18 respectively. The results are 0.31σ and 0.33σ away from the true value -2.

C. High Temperature

In this section, the Stefan-Boltzmann Law is tested for the high temperatures. The Table.IV is used to see the T^4 behavior. However, the temperature is not directly measured, but extracted from the resistance of the lamp indirectly. To extract the temperature value, the Table.II is used. Since the maximum R/R_{300K} value achieved at the experiment is 11.11 $\pm$ 0.64 Ω, the last line of the dataset is omitted for a simpler fit. The temperature function is empirically found by fitting a line to the data set (See. Fig.6).

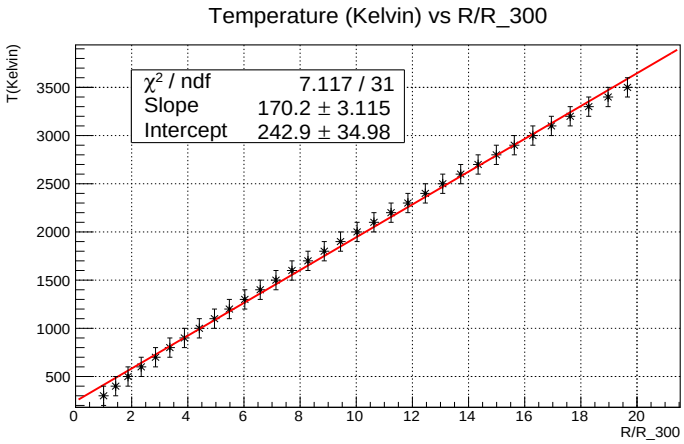

Fig. 6. Resistance Dependence of the Temperature T(R) - Line Fit.

As can be seen from the Fig.6, $T(R/R_{300K})$ = $170.2R/R_{300K}$ + 242.9.

The obtained temperature and resistance values are listed below:

TABLE VI
CALCULATED RESISTANCE AND TEMPERATURE

Resistance (Ω)	Temperature (K)
3.38 $\pm$ 0.20	2132.98 $\pm$ 111.52
3.18 $\pm$ 0.19	2021.00 $\pm$ 105.66
3.08 $\pm$ 0.18	1965.55 $\pm$ 102.74
2.89 $\pm$ 0.17	1856.59 $\pm$ 97.11
2.84 $\pm$ 0.17	1831.36 $\pm$ 95.80
2.68 $\pm$ 0.16	1742.40 $\pm$ 91.26
2.38 $\pm$ 0.15	1574.61 $\pm$ 82.78
1.97 $\pm$ 0.11	1346.39 $\pm$ 63.54
1.81 $\pm$ 0.10	1253.70 $\pm$ 58.39
0.753 $\pm$ 0.045	663.53 $\pm$ 25.71

Using the Table.VI, and the Table.IV, the following plots are obtained.

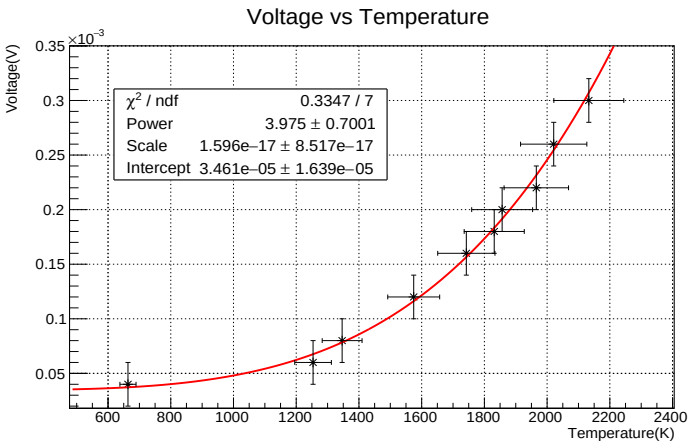

Fig. 7. High Temperature - Exponential Fit with an Intercept Parameter.

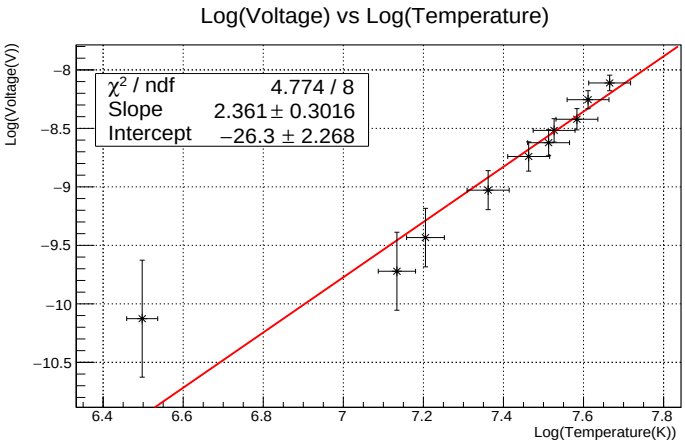

Fig. 8. High Temperature - Log Scale Line Fit.

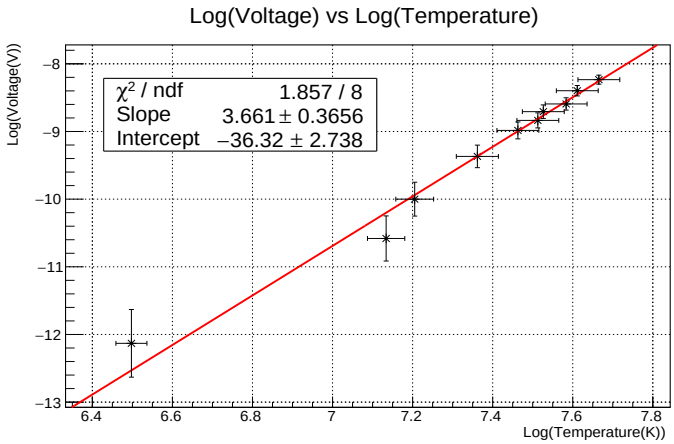

Fig. 10. High Temperature - Log Scale Line Fit. (Substracted $3.461 \cdot 10^{-5}V$ from the original voltage readings)

At Fig.7 an intercept parameter is added to see the T^4 behavior. However, the theory does not suggest an intercept. This behavior is because the amplifier was not working properly. This is also can be seen from the Fig.8. The log plot's line fit should have slope = 4; however, due to the intercept value, this is not the case.

From these plots, it is assumed that the amplifier is biased by the intercept. That is the amplifier was wrongly adding $3.461 \cdot 10^{-5} \pm 1.639 \cdot 10^{-5}V$ to the readings. Subtracting the intercept value from the voltage readings, the following plots are obtained:

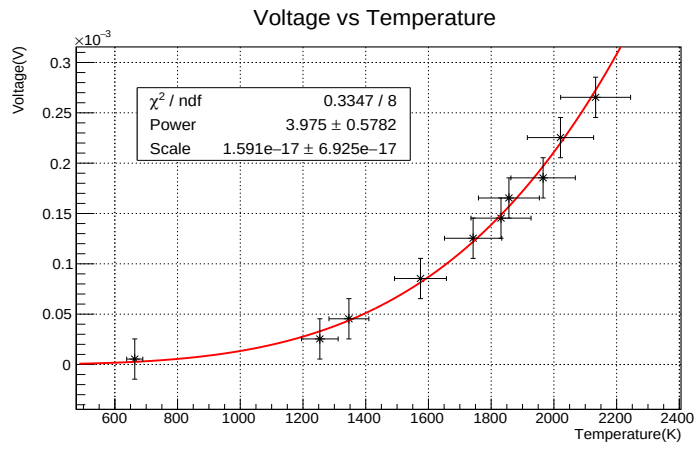

Fig. 9. High Temperature - Exponential Fit without an Intercept Parameter. (Substracted $3.461 \cdot 10^{-5}V$)

As can be seen from Fig.9 and Fig.10, the χ^2 values are reasonable; moreover, the slope and the power are consistent. The power of temperature is found to be 3.98 $\pm$ 0.58 and 3.66 $\pm$ 0.37. The σ values are 0.034 and 0.92 respectively.

D. Low Temperature

In this subsection, the Stefan-Boltzmann Law is tested at low temperatures. The Table.V is used for an exponential fit. The obtained plots are below:

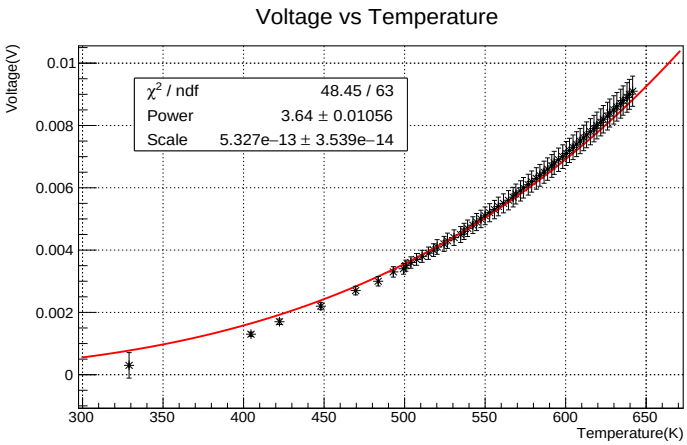

Fig. 11. Low Temperature - Exponential Fit.

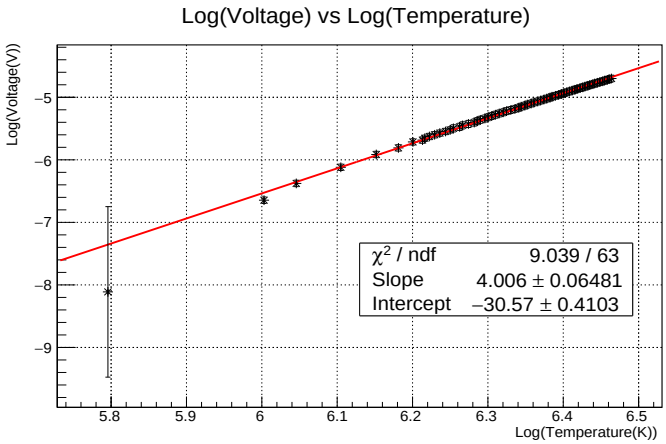

Fig. 12. Low Temperature - Log Scale Line Fit.

As can be seen from the Fig.11 and Fig.12, the power is n = 3.640 ± 0.011, and the log plot's slope is n = 4.006 ± 0.065. They are 33 σ and 0.092 σ away from the true value 4. The 33σ value is due to the temperature values' errors being too small. Increasing the uncertainties by hand leads to a better result; however, since the uncertainty is not known, it is taken as the least significant figure.

VI. The Conclusion

From the Inverse-Square Law subsection, the inverse-square law is validated. From the log plot, the power of the distance is found to be -1.94 ± 0.18, which is 0.33 σ away from the true value -2. Therefore the experiment was a success.

The Stefan-Boltzmann Law is tested for high temperature and low temperature. From the log plots, the power of the temperature is found to be: 3.66 ± 0.37, and 4.006 ± 0.065 respectively. The weighted average of these values gives 4.00 ± 0.064, which is 0σ away from the true value 4. However, at high-temperature measurements, data is fixed by hand. This is because the amplifier(most probably) that is used was malfunctioning. As can be seen from Fig.7, adding an intercept parameter fixes the issue. Therefore, the experiment was partly successful.

The experiments were done in an environment, where many possible sources different than the lamp or oven itself were present. Even the experimenters' body heat affects the experiment, which is observed several times. A more isolated environment could lead to better results.

Moreover, the law is not validated on extreme temperatures like near 0 Kelvin or much higher temperatures. These cases need further experiments. Also, in the theory and analysis, it is assumed that the source is a point-like object. However, this is not the case. Both radiators and sensor geometry should taken into account for more sophisticated results. Lastly, the emissivity of the radiators should be taken into account.

References

[1] *Über die Beziehung zwischen der Wärmestrahlung und der Temperatur*. URL: https://babel.hathitrust.org/cgi/pt?id=hvd.32044093294874&view=1up&seq=419 (visited on 12/12/2023).

[2] *Ableitung des Stefan'schen Gesetzes, betreffend die Abhängigkeit der Wärmestrahlung von der Temperatur aus der electromagnetischen Lichttheorie*. URL: https://babel.hathitrust.org/cgi/pt?id=uc1.a0002763670&view=1up&seq=327 (visited on 12/12/2023).

[3] *Luminosity of Stars*. URL: https://web.archive.org/web/20070629232956/http://outreach.atnf.csiro.au/education/senior/astrophysics/photometry_luminosity.html (visited on 12/12/2023).

[4] *Inverse Square Law Visual*. URL: https://upload.wikimedia.org/wikipedia/commons/2/28/Inverse_square_law.svg (visited on 12/12/2023).

[5] E. Gülmez. *Advanced Physics Experiments*. 1st. Boğaziçi University Publications, 1999.

VII. Appendix

We have used ROOT Version: 6.28/04 on Ubuntu 22.04.2 LTS machine for the following macros.

The functions header file:

```
#ifndef functions
#define functions

#include <math.h>
#include <vector>
#include <string>
#include <iostream>
#include <iomanip>
using namespace std;

void printResult(double val, double error, int
    sf)
{
   int factor = sf - ceil(log10(fabs(error)));
   if (factor <= 0)
   {
      cout << setprecision(sf) << val;
      cout << " +- " << error << endl;
   }
   else
   {
      cout << fixed << setprecision(factor) <<
          val;
      cout << " +- " << error << endl;
   }
}

double calculateVoltUncertainty(double V)
{
   double error;
   if (V <= 0.4)
   {
      error = (3 * V / 100) + 0.4;
   }
   else if (V <= 4)
   {
      error = (5 * V / 100) + 0.003;
   }
   else if (V <= 40)
   {
      error = (5 * V / 100) + 0.03;
   }
   else if (V <= 400)
   {
```

```
        error = (5 * V / 100) + 0.3;
    }
    else
    {
        error = (V / 100) + 4;
    }
    return error;
};


double calculateAmpUncertainty(double A)
{
    return (2 * A / 100) + 0.005;
};


double calculateLogError(double val, double
    sigma)
{
    return sigma / val;
}
#endif
```

Resistance and T(R) calculations:

```
    #include "functions.h"
using namespace std;

void ohm()
{
    // Initialize:
    gStyle->SetOptFit(1);
    auto *graph1 = new TGraphErrors();
    auto *graph2 = new TGraphErrors();
    auto *linefit = new TF1("linefit", "[0]*x +
        [1]");

    graph1->SetTitle("Voltage(V) vs
        Current(A);Current(A);Voltage(V)");
    graph2->SetTitle("Temperature (Kelvin) vs
        R/R_300;R/R_300;T(Kelvin)");
    linefit->SetParNames("Slope", "Intercept");

    vector<double> current = {5, 10, 15, 20,
        25, 30, 35, 40, 45};
    vector<double> volt = {1.3, 2.8, 4.3, 5.8,
        7.3, 9.0, 10.3, 12.1, 13.7};
    double current_error = 1;

    // Fill Graph1:
    for (size_t i = 0; i < current.size(); i++)
    {
        double volt_true = volt[i] * 1e-3;
        double current_true = current[i] * 1e-3;
        double volt_error =
            calculateVoltUncertainty(volt[i]) *
            1e-3;
        double current_error =
            calculateAmpUncertainty(current[i])
            * 1e-3;
        graph1->SetPoint(i, current_true,
            volt_true);
        graph1->SetPointError(i, current_error,
            volt_error);
    }

    // Fit Graph1:
    auto *c1 = new TCanvas("c1", "c1", 200, 10,
        600, 400);
    c1->SetGrid();
    c1->Draw();
    graph1->Fit("linefit");
    graph1->Draw("A*");

    // Print Results:
    double R = linefit->GetParameter(0);
    double R_error = linefit->GetParError(0);
    cout << "R = ";
    printResult(R, R_error, 2);

    // Find T(R):
    vector<double> table_R = {1.0, 1.43, 1.87,
        2.34, 2.85, 3.36, 3.88, 4.41, 4.95,
        5.48,
                        6.03, 6.58, 7.14, 7.71,
                            8.28, 8.86, 9.44,
                            10.03, 10.63, 11.24,
                        11.84, 12.46, 13.08,
                            13.72, 14.34,
                            14.99, 15.63,
                            16.29, 16.95,
                        17.62, 18.28, 18.97,
                            19.66};

    vector<double> table_T = {300, 400, 500,
        600, 700, 800, 900, 1000, 1100, 1200,
                        1300, 1400, 1500, 1600,
                            1700, 1800, 1900,
                            2000, 2100, 2200,
                        2300, 2400, 2500, 2600,
                            2700, 2800, 2900,
                            3000, 3100, 3200,
                        3300, 3400, 3500};

    // Fill Graph:
    for (size_t i = 0; i < table_R.size(); i++)
    {
        graph2->SetPoint(i, table_R[i],
            table_T[i]);
        graph2->SetPointError(i, 0.01, 100);
    };

    // Fit Graph2:
    auto *c2 = new TCanvas("c2", "c2", 200, 10,
        600, 400);
    c2->SetGrid();
    c2->Draw();
    graph2->Fit("linefit");
    graph2->Draw("A*");

    // Print Results:
    double slope = linefit->GetParameter(0);
    double slope_error =
        linefit->GetParError(0);
    double intercept = linefit->GetParameter(1);
    double intercept_error =
        linefit->GetParError(1);
    cout << "Slope = ";
    printResult(slope, slope_error, 2);
    cout << "Intercept = ";
    printResult(intercept, intercept_error, 2);

    //170.2x - 242.9
};
```

Inverse-Square Law Calculations:

```
   #include "functions.h"
using namespace std;

void inverse()
{
   vector<double> distance = {90.3, 81.2,
       74.5, 70.4, 68.3, 66.5, 65.2, 64.0,
       63.1,
                        62.4, 61.7, 61.2,
                            60.1, 59.8};

   vector<double> signal_V = {0.20, 0.40,
       0.60, 0.80, 1.00, 1.20, 1.40, 1.60,
       1.80, 2.00,
                        2.20, 2.40, 2.60,
                            2.80};

   double signal_V_error = 0.2;
   double distance_error = 0.2;
   double offset = 47.6;
   double amp = pow(10, 5);

   gStyle->SetOptFit(1);
   auto *linefit = new TF1("linefit", "[0]*x +
       [1]");
   linefit->SetParName(0, "Slope");
   linefit->SetParName(1, "Intercept");
   auto *graph1 = new TGraphErrors();
   auto *graph2 = new TGraphErrors();
   graph1->SetTitle("Log(Voltage) vs
       Log(Distance); Log(Distance(cm));
       Log(Voltage(V))");
   graph2->SetTitle("Voltage vs Distance;
       Distance(cm); Voltage(V)");

   for (size_t i = 0; i < distance.size(); i++)
   {
      double distance_true = distance[i] -
          offset;
      double V_true = signal_V[i] / amp;
      double V_true_error = signal_V_error /
          amp;
      graph1->SetPoint(i, log(distance_true),
          log(V_true));
      graph1->SetPointError(i,
          calculateLogError(distance_true,
          distance_error),
          calculateLogError(V_true,
          V_true_error));
      graph2->SetPoint(i, distance_true,
          V_true);
      graph2->SetPointError(i, distance_error,
          V_true_error);
   }

   graph1->Fit("linefit");
   graph1->Draw("A*");

   double n = linefit->GetParameter(0);
   double n_error = linefit->GetParError(0);

   cout << "Power = ";
   printResult(n, n_error, 2);

   auto *expfit = new TF1("expfit",
       "[1]*TMath::Power(x,[0])");
   expfit->SetParNames("Power", "Scale");
   auto *c2 = new TCanvas("c2", "c2", 200, 10,
       600, 400);
   c2->SetGrid();
   c2->Draw();
   graph2->Fit("expfit");
   graph2->Draw("A*");

   n = expfit->GetParameter(0);
   n_error = expfit->GetParError(0);
   cout << "Power = ";
   printResult(n, n_error, 2);
};
```

Low Temperature Calculations:

```
   #include "functions.h"
using namespace std;

void lowTemperature()
{
   // Initialize:
   gStyle->SetOptFit(1);
   auto *linefit = new TF1("linefit", "[0]*x +
       [1]");
   auto *graph1 = new TGraphErrors();
   auto *graph2 = new TGraphErrors();

   linefit->SetParNames("Slope", "Intercept");
   graph1->SetTitle("Log(Voltage) vs
       Log(Temperature) ;Log(Temperature(K))
       ;Log(Voltage(V))");
   graph2->SetTitle("Voltage vs Temperature
       ;Temperature(K);Voltage(V)");

   vector<double> V = {
      0.30, 1.30, 1.70, 2.20, 2.70, 3.00,
          3.30, 3.40, 3.50, 3.60,
      3.70, 3.80, 3.90, 4.00, 4.10, 4.20,
          4.30, 4.40, 4.50, 4.60,
      4.70, 4.80, 4.90, 5.00, 5.10, 5.20,
          5.30, 5.40, 5.50, 5.60,
      5.70, 5.80, 5.90, 6.00, 6.10, 6.20,
          6.30, 6.40, 6.50, 6.60,
      6.70, 6.80, 6.90, 7.00, 7.10, 7.20,
          7.30, 7.40, 7.50, 7.60,
      7.70, 7.80, 7.90, 8.00, 8.10, 8.20,
          8.30, 8.40, 8.50, 8.60,
      8.70, 8.80, 8.90, 9.00, 9.10};

   vector<double> T = {
      55.70, 131.40, 149.10, 174.80, 196.40,
          210.40, 219.80, 226.10,
      227.60, 230.70, 234.20, 237.70, 241.60,
          244.80, 247.00, 251.30,
      253.40, 257.50, 261.60, 263.70, 265.90,
          269.20, 271.50, 274.20,
      276.80, 279.60, 282.60, 285.00, 288.20,
          291.30, 294.20, 295.90,
      298.80, 300.60, 303.60, 305.70, 308.60,
          310.80, 313.40, 315.60,
      318.20, 319.70, 322.40, 325.00, 327.20,
          329.20, 331.40, 333.60,
      335.80, 338.00, 339.80, 342.20, 344.40,
          346.20, 348.60, 350.20,
      352.80, 354.20, 356.70, 358.60, 361.00,
          362.50, 364.90, 366.40,
      368.40};
```

```
    double T_error = 0.1;

    // Fill Graph1 and Graph2:
    for (size_t i = 0; i < V.size(); i++)
    {
        double logV_error =
            calculateLogError(V[i] * pow(10,
            -3), calculateVoltUncertainty(V[i])
            * pow(10, -3));
        double logT_error =
            calculateLogError(T[i] + 273.2,
            T_error);
        graph1->SetPoint(i, log(T[i] + 273.2),
            log(V[i] * pow(10, -3)));
        graph1->SetPointError(i, logT_error,
            logV_error);

        double V_error =
            calculateVoltUncertainty(V[i]) *
            pow(10, -3);
        graph2->SetPoint(i, T[i] + 273.2, V[i] *
            pow(10, -3));
        graph2->SetPointError(i, T_error,
            V_error);
    }

    // Fit Graph1:
    auto *c1 = new TCanvas("c1", "c1", 200, 10,
        600, 400);
    c1->SetGrid();
    c1->Draw();
    graph1->Fit("linefit");
    graph1->Draw("A*");

    // Print Result:
    double n = linefit->GetParameter(0);
    double n_error = linefit->GetParError(0);
    cout << "n = ";
    printResult(n, n_error, 2);

    // Fit Graph2:
    string fitarg = "[1]*TMath::Power(x,[0])";
    cout << fitarg << endl;
    auto *expfit = new TF1("expfit",
        fitarg.c_str());
    expfit->SetParameter(0,1.70743068e-17);
    expfit->SetParNames("Power", "Scale");
    auto *c2 = new TCanvas("c2", "c2", 200, 10,
        600, 400);
    c2->SetGrid();
    c2->Draw();
    graph2->Fit("expfit");
    graph2->Draw("A*");
    n = expfit->GetParameter(0);
    n_error = expfit->GetParError(0);
    cout << "n = ";
    printResult(n, n_error, 2);

};
```

High Temperature Calculations:

```
    #include "functions.h"
using namespace std;

void highTemperature()
{
    // Initialize:
    gStyle->SetOptFit(1);
    auto *linefit = new TF1("linefit", "[0]*x +
        [1]");
    auto *expfit = new TF1("expfit",
        "[1]*TMath::Power(x,[0])");
    auto *graph1 = new TGraphErrors();
    auto *graph2 = new TGraphErrors();

    linefit->SetParNames("Slope", "Intercept");
    expfit->SetParNames("Power", "Scale",
        "Intercept");
    graph1->SetTitle("Log(Voltage) vs
        Log(Temperature) ;Log(Temperature(K))
        ;Log(Voltage(V))");
    graph2->SetTitle("Voltage vs Temperature
        ;Temperature(K);Voltage(V)");

    vector<double> signal_V = {3.0, 2.6, 2.2,
        2.0, 1.8, 1.6, 1.2, 0.8, 0.6, 0.4};
    vector<double> bulb_V = {8.41, 7.39, 6.95,
        6.06, 5.88, 5.21, 4.11, 2.843, 2.474,
        0.645};
    vector<double> bulb_I = {2.487, 2.323,
        2.255, 2.099, 2.069, 1.942, 1.725,
        1.440, 1.368, 0.857};

    double signal_V_error = 0.2;
    double offset = 47.6;
    double amp = pow(10, 4);
    double distance = 56.7 - offset;
    double R_ref = 0.3045;
    double R_ref_error = 0.0085;
    double T_ref = 300;

    // Fill Graph1 and Graph2:
    for (size_t i = 0; i < bulb_V.size(); i++)
    {
        double V_true = signal_V[i] / amp;
        double V = bulb_V[i] / amp;
        double I = bulb_I[i] / amp;
        double R = V / I;

        double V_true_error = signal_V_error /
            amp;
        double V_error =
            calculateVoltUncertainty(bulb_V[i])
            / amp;
        double I_error =
            calculateAmpUncertainty(bulb_I[i]) /
            amp;
        double R_error = sqrt(pow(V_error / I,
            2) + pow(V * I_error / (I * I), 2));

        double Temperature = 242.85 + 170.2 * (R
            / R_ref);
        double Temperature_error = 156.2 *
            sqrt(pow(R_error / R_ref, 2) + pow(R
            * R_ref_error / (R_ref * R_ref), 2));

        graph1->SetPoint(i, log(Temperature),
            log(V_true-3.461e-5));
        graph1->SetPointError(i,
            calculateLogError(Temperature,
            Temperature_error),
            calculateLogError(V_true,
            V_true_error));

        graph2->SetPoint(i, Temperature,
            V_true-3.461e-5);
```

```
      graph2->SetPointError(i,
          Temperature_error, V_true_error);
   }

   // Fit Graph1:
   //linefit->SetParLimits(0,3.5,4.5);
   linefit->SetRange(7,8);
   auto *c1 = new TCanvas("c1", "c1", 200, 10,
       600, 400);
   c1->SetGrid();
   c1->Draw();
   graph1->Fit("linefit");
   graph1->Draw ("A*");

   // Print Result:
   double n = linefit->GetParameter(0);
   double n_error = linefit->GetParError(0);
   cout << "n = ";
   printResult(n, n_error, 2);

   // Fit Grah2:
   expfit->SetParameter(1, 1.70743068e-17);
   expfit->SetParameter(0, 4);
   auto *c2 = new TCanvas("c2", "c2", 200, 10,
       600, 400);
   c2->SetGrid();
   c2->Draw();
   graph2->Fit("expfit");
   graph2->Draw("A*");

   // Print Result:
   n = expfit->GetParameter(0);
   n_error = expfit->GetParError(0);
   cout << "n = ";
   printResult(n, n_error, 2);
};
```