A MONTE CARLO STUDY OF MAXWELL'S DEMON COUPLED TO FINITE QUANTUM HEAT BATHS

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF

MASTER OF SCIENCE

IN

PHYSICS

By

Umutcan Güler

August 2020

A MONTE CARLO STUDY OF MAXWELL'S DEMON COUPLED TO FINITE QUANTUM HEAT BATHS

By Umutcan Güler

August 2020

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Cemal Yalabık(Advisor)

Bilal Tanatar

Şinasi Ellialtıoğlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

A MONTE CARLO STUDY OF MAXWELL'S DEMON COUPLED TO FINITE QUANTUM HEAT BATHS

Umutcan Güler
M.S. in Physics
Advisor: Cemal Yalabık
August 2020

When Maxwell's demon was introduced, it raised the question: Is there a way to decrease an isolated system's entropy, even though it was forbidden by the second law of thermodynamics. Then, a new idea which considered information as a physical entity was emerged, and an equivalence between information entropy and thermodynamic entropy was suggested. Under the light of new understandings, the original question modified into "Is there a way to decrease thermodynamic entropy of a system by using information entropy?" This work aims to demonstrate such a machinery is possible to exist in real world. Building on the model of Mandal et al. [1], it inquires whether if such a system is possible to build in nano scales. According to the theoretical relations, the correspondences between internal energy and effective temperature of finite fermionic and bosonic gases for varying number of particles and volumes were tabulated. Subsequently, a series of Monte Carlo simulations were executed under different circumstances. The outcomes of the simulations illustrate that production of information entropy can be used to compensate the decrease of thermodynamic entropy. The results indicate that using either one of the quantum gases as a finite quantum heat bath does affect the efficiency of the refrigerator. Based on this, using fermionic gas is superior to bosonic gas in terms of swiftness of the refrigeration, if all other variables are identical. Further research is needed to analyze the behaviour of the finite quantum heat baths at extremely low temperatures.

Keywords: Maxwell's Demon, information entropy, finite Fermi gas, finite Bose gas, Monte Carlo simulation.

ÖZET

SONLU KUANTUM ISI BANYOLARI İLE BİRLEŞİK MAXWELL'İN CİNİ ÜZERİNE BİR MONTE CARLO İNCELEMESİ

Umutcan Güler
Fizik, Yüksek Lisans
Tez Danışmanı: Cemal Yalabık
Ağustos 2020

Maxwell'in Cini sunulduğundan beri gündemde tuttuğu bir konu vardır: Termodinamiğin ikinci yasası yasaklasa dahi, yalıtımlı bir sistemin entropisini azaltmak mümkün müdür? Sonraları bilginin fiziksel olduğu fikri ileri sürüldü, ve bilgi entropisi ile termodinamik entropinin denkliği önerildi. Bu yeni kavrayışlar ışığında, ilk soru "Bir sistemin termodinamik entropisini bilgi entropisi kullanarak düşürmek mümkün müdür?" olarak değişti. Bu yapıtın amacı, böyle bir düzeneğin gerçek dünyada var olabileceğini göstermektir. Mandal vd. [1] tarafından oluşturulan model kullanılarak, böyle bir sistemin nano boyutlarda üretilebilme olasılığı sorgulanıyor. Kuramsal bağıntılar kullanılarak, farklı parçacık sayısı ve boyuta sahip sonlu fermiyonik ve bozonik gazların iç enerji ve etkin sıcaklık değerleri tasnif edildi. Ardından, farklı koşullar için bir dizi Monte Carlo benzetimi yürütüldü. Bu benzetimlerin neticesi, bilgi entropisi üretiminin termodinamik entropideki düşüşü telafi edebileceğini gösterdi. Sonuçlar, sonlu kuantum ısı banyosu olarak kullanılan kuantum gazı türünün, soğutucu verimine etki ettiğini işaret etmektedir. Buna istinaden, şayet bütün diğer şartlar aynıysa, soğutma hızı açısından fermiyonik gaz kullanımı bozonik gaz kullanımına üstündür. Sonlu kuantum ısı banyolarının aşırı düşük sıcaklıklardaki davranışlarının çözümlenmesi için ilave çalışma gerekmektedir.

Anahtar sözcükler: Maxwell'in Cini, bilgi entropisi, sonlu Fermi gazı, sonlu Bose gazı, Monte Carlo benzetimi.

Acknowledgement

There is a number of people without whose help, I would not be able to survive through the last three years.

First and foremost, I wish to express my deepest gratitude to my adviser, Professor Cemal Yalabık, who not only guided and encouraged me throughout my research, but also put up with my everlasting laziness. Without his endless help, this work would not have been completed. I would also like to thank my committee members, Professor Bilal Tanatar and Professor Şinasi Ellialtıoğlu for their helpful suggestions.

There have been many friends who helped me in this journey. I am very thankful to Kemal Emrecan Şahin, who has been supporting me since day one, in both real life and our online gaming meetings. I am deeply grateful to Burak Helvacıoğlu for saving my life, and helping me to keep it on track. I also wish to thank my friends Kaan Ertek, Alper Gencer, Zeynep Küçüksümer, and Aksel Magiya for being with me, even in my darkest times.

I would like to thank my friends here at Bilkent University. As an introverted person, connecting with so many people in so little time still amazes me.

Last but not least, I would like to thank my family. There is no words to describe how grateful I am for your never ending support and understanding. And thank you for bearing with me, even at the times I cannot.

To Sabahattin Genlik and Necmi Güler

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Even though the second law of thermodynamics is universally valid, since its proposition in 1867, Maxwell's demon challenged second law's validity on molecular level, and started a query to overcome the demon's implications. After years of debate and criticism, in 1929 Leo Szilard suggested a solution to the problem [2]. He claimed that, in order to demon select which molecules can pass through the hole, the demon must measure the velocity of the molecules. The measurement process requires an expenditure of energy, hence causes an increase in the demon's entropy, which compensates the decrease of the thermodynamic entropy. This idea was of utmost importance since it depicted information as a physical phenomenon, and associated it with thermodynamic entropy. Afterwards, in 1961 Rolf Landauer introduced his principle which assigns a lower limit to the expenditure of energy needed to process information in a logically irreversible operation [3]. He suggested that in order to decrease thermodynamic entropy of a system via measurements, the demon needs to obtain an information regarding the state of a passing molecule and either erase the information or store it. According to Landauer's principle, erasing the information would increase the thermodynamic entropy by at least an amount that increases the total thermodynamic entropy of the overall system; on the other hand storing the information in a memory register would increase the information entropy by at least an equivalent amount to ensure the total entropy of the overall system increases.

In this thesis, we focused on the second category of handing the acquired information which demonstrates the equivalence of the information and thermodynamic entropies. To do so, we used the model of Maxwell refrigerator established by Mandal et al. [1]. Their demon, while lowering the thermodynamic entropy, raises the information entropy by randomly producing information and storing it. However, while working out their model they used thermal reservoirs which has constant temperature alongside extraction and addition of energy. On the contrary, in our study we conducted a series of Monte Carlo simulations of their model coupled to large but finite fermionic and bosonic systems in place of the thermal reservoirs. We assumed these systems, entitled as finite heat baths, are large enough to use the theories of ideal Fermi and Bose gases, but also small enough to change their temperature under heat addition or extraction. Our motivation for this is to see the actual functioning of such a refrigerator, in case it could physically be constructed using developments in nanotechnology. This nano-refrigerator would be assigned to lower the effective temperature of a finite size system. Beyond that, we also address the question how does using fermionic and bosonic gases differ the behaviour of the refrigerator? We discovered that such a refrigerator can exist in a real world scenario, and under exact conditions, fermionic finite heat baths are more favorable for reaching low effective temperatures than bosonic finite heat baths.

Chapter 2 contains a summary of the second law of thermodynamics and different definitions of entropy. The statement of Maxwell's demon, and the description of the model used in this study are then presented in Chapter 3. Subsequently, the theoretical background of fermionic and bosonic gases are formulated in Chapter 4, which concludes the technical and theoretical introduction. From there on, next chapters reveal the original work. Chapter 5, addresses the problem regarding the quantization of temperature, proposes the modification needed for finite systems, and describes the simulation. Following that, Chapter 6 reports the results, and speculates about their significance. Finally, Chapter 7 summarizes the main conclusions, and identifies both the major issues faced throughout the study and suggestions to overcome them for further research.

We would like to end this chapter by listing some further readings regarding

the subjects discussed in this work. Firstly, the book *Maxwell's Demon 2: Entropy, Classical and Quantum Information, Computing* (Leff & Rex, 2003) [4] is a remarkable source that both gives an overview about the subject and compiles some of the most fundamental papers published until early 2000s. Beyond that, the attention that has been focusing on Maxwell's Demon in recent years induced a number of papers. In theoretical research, different systems and models have been proposed; for both classical Maxwell's Demon [5], [6], [7], and quantum Maxwell's Demon [8], [9], [10], [11], which functions with quantum measurements. Alongside theoretical researches, numerous experimental studies have been conducted concerning the demon. These experiments are of the utmost importance, since they demonstrate the real life operation of the demon, and identify new fields of usage for it. While some experimental researches have focused on using classical demon [12], [13], [14], there have been a number of works testing quantum version of the demon as well [15], [16].

Chapter 2

Second Law and Entropy

2.1 The Second Law of Thermodynamics

The first law of thermodynamics is a special application of universally valid principle of conservation of energy for thermodynamic systems [17]. Unlike in classical mechanics, in thermodynamic systems, which is composed of many particles, it is impossibly complicated to consider each particle's state. Instead, the motion of those many particles are described by macroscopic states, such as internal energy, temperature, and entropy.

The first law can be expressed as

$$\Delta U = Q - W$$

where, ΔU is the change of internal energy, Q is the heat supplied to the system, and W is the work done by the system on its surroundings. From this expression, what the first law tells is how internal energy balances itself in a thermodynamic process. However, it does not give any information regarding the direction of the process. At this stage, the second law of thermodynamics comes to the picture.

Consider a thermodynamic system out of equilibrium. Then, if there is no intervention, the system will always move toward equilibrium, although a change in the reverse direction also conserves energy: a broken glass will never be whole

again; using a wet towel will never make it dry while wetting its user; heat will never flow from a cold body to a hot body by itself. The phrase *by itself* is crucial here, because with an intervention heat can flow from a cold body to a hot one, which is called refrigeration. However, in a natural process, without any intervention, heat tends to distribute homogeneously. This is the definition of the second law of thermodynamics given by Clausius in 1854 [18].

2.2 Entropy

In everyday use entropy is used in many different ways: *the measure of randomness*, or *the tendency of things to go to disorder*, or *the inevitable deterioration of the universe*. A simple yet rigorous way to define entropy in thermal physics would be *a quantity representing the unavailable thermal energy per unit temperature for doing work* [19]. However, there are various ways to define entropy in different contexts. Following are *the thermodynamic definition*, *the statistical mechanics definition*, and *the information theory definition*.

2.2.1 Thermodynamic Entropy

Unlike classical mechanics, which considers the properties of the individual particles that constitutes a system of a number of particles, classical thermodynamics solely considers the averaged properties of a macroscopic system, i.e. temperature, heat capacity, and entropy. One of the earliest definition of entropy was given by Rudolf Clausius in his work *The Mechanical Theory of Heat* [20].

According to *Clausius Theorem*, for a reversible cyclic process in a closed homogeneous system,

$$\oint \frac{\delta Q}{T} = 0.$$

where T is the temperature of the system, and δQ is the incremental heat energy that was transferred along the process. The equality means the line integral

$$\int_L \frac{\delta Q}{T}$$

is path independent. Hence, $\delta Q/T$ defines a function of state S, called *entropy*, that satisfies

$$dS = \frac{\delta Q}{T},$$

and the entropy difference between two states can be found by evaluating the integral for a reversible path.

2.2.2 Statistical Entropy

In 1870s, Ludwig Boltzmann came up with his interpretation of entropy. Unlike Clausius, Boltzmann considered the microscopic components of the system. By analysing the statistical behaviour of the components, Boltzmann gave his definition of entropy as a measure of the microstates of the system.

Macrostates of a many body system are governed by the distribution of the microstates. For Ω being the number of the microstates that corresponds to the macrostate of the system at that moment, entropy is

$$S = k \ln \Omega, \tag{2.1}$$

where k is the *Boltzmann constant*. Formula (2.1), which is known as *the Boltzmann's entropy formula*, relates entropy to the number of different states which the particles of the thermodynamic system can be in. However, Boltzmann definition did that by using the fundamental postulate of statistical mechanics, that is all microstates are equally probable. This is but a special case when the system is at thermal equilibrium.

For systems that are far from equilibrium, a generalization of Boltzmann entropy, called *Gibbs entropy*, is given by

$$S = -k \sum_i p_i \ln p_i,$$

where p_i is the probability of the i^{th} microstate to occur. Note that for p_is being equal for all i, Gibbs entropy reduces to Boltzmann entropy, which also corresponds to the maximum entropy of that system.

2.2.3 Information Entropy

Other than the thermodynamic systems, entropy is used in information theory as well. It was originally derived by Claude Shannon in 1948 as a measure of the amount of information lost while a message is transmitted [21]; however, its area of use became much broader than the communication theory.

For any possible value of data, *Shannon entropy* is given by

$$H = -\sum_i p_i \log_b p_i,$$

where p_i is the probability of being at the i^{th} state of the phase space, and b is the base of logarithm in use. Most common values used for b are 2, 10, and *Euler's number*. According to the base of logarithm used, the unit of the information entropy changes: if $b = 2$ it is *bits* or *shannons*, if $b = 10$ it is *decimal digits* or *hartleys*, and if $b = e$ it is *natural units* or *nats*. Note that, as in statistical entropy, if probabilities of each information is equal, then

$$H = -\sum_i p \log_b p = \log_b M,$$

where $M = p^{-1}$.

Second Law Revisited

Entropy, an interesting notion as it is, has limited usability. Change in entropy, however, is a much more useful concept. In section 2.1, we gave a verbal definition of the Second Law of Thermodynamics. Now, equipped with the definition of entropy, we can give a mathematical understanding to it.

A technical definition of the Second Law of Thermodynamics is; for an isolated system, which tends to go towards thermodynamic equilibrium, the total entropy never decreases over time. If the system is in thermal equilibrium, or undergoing a reversible process the total entropy of the system and its surroundings remains constant. However, in processes that occurs in physical world, entropy always increases irreversibly. We can represent the idea above with

$$\Delta S \geq 0.$$

Chapter 3

The Demon

3.1 The Intelligent Demon

James Clerk Maxwell proposed a thought experiment to his friend Peter Guthrie Tait in one of his letters in 1867. This experiment pertained a hypothetical violation of the Second Law of Thermodynamics by the intervention of a *neat-fingered being*. Maxwell published this idea in his book *Theory of Heat* [22] in 1871, in which he explained the Second Law as:

> *...One of the best established facts in thermodynamics is that it is impossible in a system enclosed in an envelope which permits neither change of volume nor passage of heat, and in which both temperature and the pressure are everywhere same, to produce any inequality of temperature or of pressure without the expenditure of work...* [23]

He then explained his thought experiment as:

> *...For we have seen that the molecules in a vessel full of air at uniform temperature are moving with velocities by no means uniform, though the mean velocity of any great number of them, arbitrarily selected,*

> *is almost exactly uniform. Now, let us suppose that such a vessel is divided into two portions, A and B, by a division in which there is a small hole, and that a being, who can see the individual molecules, opens and closes this hole, so as to allow only the swifter molecules to pass from A to B, and only the slower ones to pass from B to A. He will thus, without expenditure of work, raise the temperature of B and lower that of A, in contradiction to the Second Law of Thermodynamics...* [24]

While proposing the experiment, Maxwell aimed to emphasize the statistical nature of the Second Law. That is, even though for a many particle system the law hold for almost all the time, there is still a non-zero probability that an anisotropic transfer to occur if a hole is left between the two portions [25].

There are important properties which was decided after discussions between Maxwell and William Thomson. The most important one is that the demon should be incapable of doing work, even though it opens and closes the frictionless and inertialess valves. This is essential since an additional work on the system would justify the decrease of the entropy. Next, the demon does not have to have preternatural abilities such as creating or annihilating energy, only its intelligence is required to be observant on the particles and distinguish their velocities [26].

There is no indication that Maxwell tried to challenge the validity of the Second Law by constructing his experiment, but wanted to show the limitations of it [27]. Nevertheless, the demon still stays as a trophy to be achieved, resulting from the arguments that it ignited. Some of the arguments that we considered in this work are the relationship between the thermodynamic and information entropies, and to see if it is possible to compensate the decrease of one with the increase of the other.

3.2 The Unintelligent Demon

Since Maxwell's Demon was proposed, numerous models has proposed to legitimize its existence. In this work, we conducted a study around the model

proposed by Mandal et al. [1], which proposes an exactly solvable model for an autonomous device that replaces the intelligent being with a *dumb device*. Like the Maxwell's Demon, this devices is also capable of driving the energy to flow against the thermal gradient without doing external work. What Mandal et al. proposed is a device with a classical two-state system which interacts with two heat baths and a memory register. While a stochastic transition occurs, the device induces energy transfer from one heat bath to the other, and simultaneously interacts and possibly modifies an entry in the memory register. When the concerning parameters are tuned properly, a steady state of continuous flow of energy from one bath to the other arises, accompanied with a continuous recording of the evolution of the system to the memory register.

Like its intelligence, Maxwell's Demon lost some of the original attributes, and become a more general term for any condition that causes a decrease of thermodynamic entropy due to rectification of microscopic fluctuation [1]. While the debates on the equivalence of thermodynamic entropy and information entropy, the consensus is without violating the Second Law, a physical device can decrease the thermodynamic entropy if it simultaneously writes information on a memory register [1]. The idea is an increase in the information entropy, caused by writing of stochastically generated information to a memory register, should be high enough to compensate the decrease in the thermodynamic entropy, caused by the refrigeration. Later, if the information is wished to be erased, by *Landauer's principle*, there must be an increase in thermodynamic entropy elsewhere. Then the two types of entropy, Shannon and Clausius entropies, can be related, and if the sum of these entropies is greater than zero the Second Law is satisfied.

3.3 The Model

The model of Mandal et al. consists of four components: two heat baths with temperatures T_c and T_h where $T_h > T_c$, a memory register, and a device that replaces the Maxwell's Demon which we will call *the demon*. The memory register is a frictionless band of equally spaced sequence of bits with two states with same energies that slides past the demon. The duration of interaction with

a bit is $\tau = l/v$, where l is the spacing between the bits and v is the constant speed of the band. The demon interacts with both of the heat baths and with the nearest bit of the memory register.

The demon is also a two state system which are labeled as u and d with the energy difference $\Delta E = E_u - E_d > 0$. There are two kinds of transitions between these states. The first one is called an *intrinsic transition*, in which the demon exchanges energy with the hot heat bath and does not modify the bit. The second one is called a *cooperative transition*, in which the demon exchanges energy with the cold heat bath while modifying the bit, if the required condition is satisfied. The condition is the combined state of the demon and the bit to be $0d$ or $1u$. The model also rules out the intrinsic transition between the states of the bits; that is, for a bit to change an interaction between the bit and the demon is mandatory.

The transition rates of the intrinsic transition satisfy *detailed balance*

$$\frac{R_{d\alpha\to u\alpha}}{R_{u\alpha\to d\alpha}} = e^{-\beta_h \Delta E} \tag{3.1}$$

where $\alpha = \{0, 1\}$, $\beta_h = (kT_h)^{-1}$. These rates can be parametrized as

$$R_{d\alpha\to u\alpha} = \gamma(1-\sigma), \quad R_{u\alpha\to d\alpha} = \gamma(1+\sigma), \quad \sigma = \tanh\frac{\beta_h \Delta E}{2}$$

where $0 < \sigma < 1$, and $\gamma > 0$ is a parameter which sets a characteristic rate for the transitions.

The transition rates of the cooperative transition also satisfy detailed balance

$$\frac{R_{d0\to u1}}{R_{u1\to d0}} = e^{-\beta_c \Delta E} \tag{3.2}$$

where $\beta_c = 1/kT_c$. The rates can be similarly parametrized as

$$R_{0d\to 1u} = (1-\omega), \quad R_{1u\to 0d} = (1+\omega), \quad \omega = \tanh\frac{\beta_c \Delta E}{2}$$

where $0 < \omega < 1$. Note that as $kT_c \to 0$, $\omega \to 1$, which makes $0d \to 1u$ less likely, and slows down the energy extraction from cold heat bath.

The parameter $0 \le \epsilon < 1$ is defined as

$$\epsilon = \frac{\omega - \sigma}{1 - \omega\sigma} = \tanh\frac{(\beta_c - \beta_h)\Delta E}{2},$$

which is used to quantify the temperature difference between heat baths.

Next, two parameters are defined to measure the change in the information

entropy. Let p_0 and p_1 be the probabilities of the next bit of the incoming band being 0's and 1's, respectively. Then

$$\delta \equiv p_0 - p_1$$

represents the proportional excess of 0's among the incoming bits. Likewise, let p_0' and p_1' be the probabilities of the next bit of the outgoing band being 0's and 1's, respectively. Then

$$\delta' \equiv p_0' - p_1'$$

represents the proportional excess of 0's among the outgoing bits.

The information entropy is given by

$$\begin{aligned} S(\delta) &= -\sum_{i=0}^{1} p_i \ln p_i \\ &= -\frac{1-\delta}{2} \ln \frac{1-\delta}{2} - \frac{1+\delta}{2} \ln \frac{1+\delta}{2}, \end{aligned}$$

and change in information entropy is

$$\Delta S_I = S(\delta') - S(\delta).$$

Note that $\Delta S_I > 0$ indicates that the demon writes information to the memory register produced by a stochastic process, whereas $\Delta S_I < 0$ indicates that the demon erases information from the memory register.

Lastly, $\Phi = p_1' - p_1$ is defined to quantify the average production of 1's per interaction interval in the outgoing bit stream relative to the incoming bit stream. Note that from

$$\begin{aligned} p_0 + p_1 = p_0' + p_1' &\Rightarrow p_1 - p_1' = p_0' - p_0, \\ \delta = p_0 - p_1 &\Rightarrow p_1 = p_0 - \delta, \\ \delta' = p_0' - p_1' &\Rightarrow p_1' = p_0' - \delta', \end{aligned}$$

Φ can be written as

$$\Phi = \frac{\delta - \delta'}{2}.$$

Since only cooperative transitions transfer energy ΔE from cold to hot heat bath, average transfer of energy from cold to hot heat bath per interaction interval is given by

$$Q_{c \to h} = \Phi \Delta E.$$

If $Q_{c\to h} < 0$, the demon transfers energy in the direction of thermal gradient, which increases thermal entropy; whereas if $Q_{c\to h} > 0$, the demon transfers energy against the thermal gradient, and decreases thermal entropy.

As it was shown in the supplementary material of [1] by Mandal et al., Φ can be written in terms of the model parameters $\Lambda \equiv (\delta, \sigma, \gamma, \omega, \tau)$ as

$$\Phi(\Lambda) = \frac{\delta - \epsilon}{2}\eta(\Lambda), \quad \eta > 0.$$

Moreover, they derived *modified Clausius inequality* as

$$Q_{c\to h}(\beta_h - \beta_c) + \Delta S_I \geq 0$$

From these equations, we can find a basis for the effects of the parameters δ and ϵ on the temperature. The sign of Φ is determined by the sign of $\delta - \epsilon$. Mandal et al. interpreted the difference $\delta - \epsilon$ as the competition between two effective forces [1]: δ represents the statistical bias caused by incoming bit stream, and ϵ represents the temperature gradient between two heat baths. Whichever dominates the other determines the sign of Φ, and the direction of the heat flow. From this we conclude the following crucial point: refrigeration occurs only in the $\delta > \epsilon$ region of the phase space.

Chapter 4

Heat Baths

In this chapter, we find the correspondences between internal energy of finite fermionic and bosonic system and their temperature. Note that in microscopic scale the definition of temperature is only valid in thermodynamic limit. For finite systems, what we mean by temperature is the effective temperature of the system that corresponds to its internal energy for a given number of particles and volume that encloses the particles. For convenience, we will mention effective temperature simply as temperature.

Another important point on this chapter is what we mean by finite heat bath. By definition, a heat bath, or a thermal reservoir is a thermodynamic system with an immensely large heat capacity. As a consequence, a thermal reservoir does not undergo any change in temperature when finite amounts of heat is added or extracted. However, any physical body whose heat capacity is larger than the energy added or extracted can be modeled as a thermal reservoir. We assume our finite systems are large enough to satisfy the later criterion, but small enough to undergo a change in temperature when heat is added or extracted. Thus, even though our *finite heat baths* are not exactly a thermal reservoir, our assumption allows us to model them as one. By this assumption, we are able to use the theories of ideal Fermi and Bose gases. Note that, in this work, we used the terms heat bath and finite heat bath interchangeably, and used only thermal reservoir with its original meaning.

4.1 Ideal Fermi Gas

At $T = 0$, internal energy of non-interacting Fermi gas is given by

$$U = \frac{3}{5} N \epsilon_F,$$

where ϵ_F is the Fermi energy

$$\epsilon_F = \frac{\hbar^2}{2m} \left(\frac{3\pi^2 N}{V} \right)^{2/3}, \tag{4.1}$$

and N is the total number of particles in the gas, V is the volume of the gas, m is the mass of each fermions, and $\hbar$ is the reduced Planck's constant. Note that even though we work with finite Fermi gas, we can use the definition above (see appendix A).

For temperature values $T \neq 0$, however, internal energy of the Fermi gas is a function of the temperature. In this section, we inspected fundamental relations of ideal Fermi gases to derive internal energy in finite temperature.

The equation of state of the ideal Fermi gas is

$$\frac{PV}{kT} = \ln \mathcal{Z}(z, V, T) = \sum_{\mathbf{p}} \ln(1 + z e^{-\beta \epsilon_{\mathbf{p}}}) \tag{4.2}$$

where P denotes pressure. $\mathcal{Z}(z, V, T)$ is the grand partition function for fermions

$$\mathcal{Z}(z, V, T) = \prod_{\mathbf{p}} (1 + z e^{-\beta \epsilon_{\mathbf{p}}}),$$

where $\beta = (kT)^{-1}$, $\epsilon_{\mathbf{p}} = \mathbf{p}^2/2m$, and $\mathbf{p}$ is the momentum vector. z is *fugacity*, defined as $e^{\beta\mu}$, where μ is *chemical potential* which satisfies

$$\lim_{T \to 0} \mu = \epsilon_F.$$

Moreover in high temperatures, chemical potential expands as

$$\mu \approx kT \ln \left[\frac{4}{3\sqrt{\pi}} \left(\frac{\epsilon_F}{kT} \right)^{3/2} \right]$$

From these we see that as $T \to 0$ we have $\mu \to \epsilon_F$ and $z \to \infty$, and as $T \to \infty$ we have $\mu \to -\infty$ but $kT/\mu \to -\infty$ so $z \to 0$. Hence $0 \leq z < \infty$ [28].

Total number of particles N is given by

$$N = \sum_{\mathbf{p}} \langle n_{\mathbf{p}} \rangle$$

where

$$\langle n_{\mathbf{p}} \rangle = \frac{1}{z^{-1}e^{-\beta\epsilon_{\mathbf{p}}} + 1}$$

is the average occupation number. Note that N can be written in terms of the grand partition function as

$$N = z\frac{\partial}{\partial z} \ln \mathcal{Z}(z, V, T) = \sum_{\mathbf{p}} \frac{ze^{-\beta\epsilon_{\mathbf{p}}}}{1 + ze^{-\beta\epsilon_{\mathbf{p}}}}, \tag{4.3}$$

which also confirms that $0 \leq z < \infty$, since N should be a non-negative number.

Since $\mathbf{p} = \pi\hbar\mathbf{n}/V^{1/3}$, the discreet possible values of momentum vectors form a continuum in the limit $V \to \infty$, and the summations in the formulae (4.2) and (4.3) can be replaced with integrals

$$\sum_{\mathbf{p}} \to \frac{V}{h^3} \int d^3p$$

Then, the new expressions for equation of states and number of particles are

$$\frac{PV}{kT} = \frac{V}{h^3} 4\pi \int_0^\infty dp\, p^2 \ln(1 + ze^{-\beta\epsilon_{\mathbf{p}}})$$
$$N = \frac{V}{h^3} 4\pi \int_0^\infty dp\, p^2 \frac{1}{z^{-1}e^{\beta\epsilon_{\mathbf{p}}} + 1}.$$

After rearrangements, we have

$$\frac{P}{kT} = \frac{4\pi}{h^3} \int_0^\infty dp\, p^2 \ln(1 + ze^{-\beta p^2/2m})$$
$$\frac{N}{V} = \frac{4\pi}{h^3} \int_0^\infty dp\, p^2 \frac{1}{z^{-1}e^{\beta p^2/2m} + 1}.$$

The integrals above can be written in terms of Fermi-Dirac functions shown in appendix B, which yield to

$$\frac{P}{kT} = \frac{g}{\lambda^3} f_{5/2}(z)$$
$$\frac{N}{V} = \frac{g}{\lambda^3} f_{3/2}(z)$$

where, $\lambda = \sqrt{2\pi\hbar^2/mkT}$ is the *thermal wavelength*, and the factor g is the degeneracy factor of fermions. By using $U = \frac{3}{2}PV$, and replacing $\mathbf{g}$ with 2, we have

$$\frac{U}{V} = \frac{3kT}{2\lambda^3} f_{5/2}(z) \tag{4.4}$$
$$\frac{N}{V} = \frac{2}{\lambda^3} f_{3/2}(z). \tag{4.5}$$

Finally, by dividing equation (4.4) to equation (4.5) we found the internal energy per particle at constant volume as

$$\frac{U}{N} = \frac{3}{2}kT\frac{f_{5/2}(z)}{f_{3/2}(z)}. \tag{4.6}$$

4.1.1 Numerical Limits

To find the numerical correspondence between internal energy and temperature for a constant number of particles at a constant volume, we numerically inverted equation (4.5) at different temperature steps. Then we entered each temperature-fugacity pair into equation (4.6) to find the internal energy per particle at a given N, V, and kT.

As mentioned before, $z \to \infty$ in the limit $T \to 0$. This makes numerical inversion difficult for low temperatures. Therefore, as the numerical inversion fails we used

$$\begin{aligned} U &= \sum_{\mathbf{p}} \epsilon_{\mathbf{p}} \langle n_{\mathbf{p}} \rangle \\ &\approx \frac{3}{5}N\epsilon_F \left[1 + \frac{5}{12}\pi^2 \left(\frac{kT}{\epsilon_F}\right)^2 - \frac{\pi^4}{16}\left(\frac{kT}{\epsilon_F}\right)^4\right] \end{aligned} \tag{4.7}$$

to find the internal energy. Note that in low temperatures equation (4.6) converges to (4.7) (See appendix D).

4.2 Ideal Bose Gas

Unlike fermions, bosons do not obey *Pauli Exclusion Principle*. As a result, their low temperature behaviour differ from the fermionic case: a large fraction of bosons occupy the lowest energy level for sufficiently low temperatures, known as *Bose-Einstein Condensation*. In this section we will investigate the behaviour of spin-0 massive bosons.

The equation of states of the ideal Bose gas is

$$\frac{PV}{kT} = \ln \mathcal{Z}(z, V, T) = -\sum_{\mathbf{p}} \ln(1 - ze^{-\beta\epsilon_{\mathbf{p}}}), \tag{4.8}$$

where $\mathcal{Z}(z,V,T)$ is the grand partition function for bosons

$$\mathcal{Z}(z,V,T) = \prod_{\mathbf{p}} \frac{1}{ze^{-\beta\epsilon_{\mathbf{p}}}}$$

Average occupation number of bosons is given by

$$\langle n_{\mathbf{p}} \rangle = \frac{1}{z^{-1}e^{-\beta\epsilon_{\mathbf{p}}} - 1}.$$

As a result, total number of particles of bosons is given by

$$N = z\frac{\partial}{\partial z} \ln \mathcal{Z}(z,V,T) = \sum_{\mathbf{p}} \frac{ze^{-\beta\epsilon_{\mathbf{p}}}}{1 - ze^{-\beta\epsilon_{\mathbf{p}}}}. \tag{4.9}$$

Note that, unlike fermions, equation (4.9) implies that for non-negative N fugacity should be $0 \leq z \leq 1$. We can verify this by checking the chemical potential of bosons. For low temperatures,

$$\mu = \begin{cases} 0 & T < T_{crit} \\ -\frac{9\zeta(3/2)^2}{16\pi}\frac{(kT - kT_{crit})^2}{kT_{crit}} & T > T_{crit} \end{cases}$$

and for high temperatures

$$\mu \approx kT \ln\left[\frac{4}{3\pi}\left(\frac{kT_{crit}}{kT}\right)^{3/2}\right],$$

where T_{crit} is the critical temperature

$$T_{crit} = \frac{2\pi\hbar^2}{mk}\left(\frac{N}{V\zeta(3/2)}\right)^{3/2},$$

and m is the mass of each bosons. These show us as $T \to 0$ we have $\mu \to 0$ but $kT/\mu \to 0$ so $z \to 1$, and as $T \to \infty$ we have $\mu \to -\infty$ but $kT/\mu \to 0$ so $z \to 0$. Hence $0 \leq z \leq 1$ [28].

Note that even though the discussion above is complete, there is an important assumption regarding its validity. A numerical study shows that $z = 1$ occurs only when the $V \to \infty$ [29]. However, for finite heat baths z gets closer but never reaches to 1, even when $T < T_{crit}$. So, we update the inequality as $0 \leq z < 1$ for our study.

Next, we will replace the summations on equations (4.8) and (4.9) with integrals. This time, however, the summations have a singularity: as $z \to 1$, the

term corresponding $\epsilon_{\mathbf{p}} = 0$ diverges. Due to that, we will separate it from the summation and take care of it separately. The resulting integrals are

$$\frac{PV}{kT} = -\frac{V}{h^3} 4\pi \int_0^\infty dp\, p^2 \ln(1 - ze^{-\beta\epsilon_{\mathbf{p}}}) - \ln(1-z)$$
$$N = \frac{V}{h^3} 4\pi \int_0^\infty dp\, p^2 \frac{1}{z^{-1}e^{\beta\epsilon_{\mathbf{p}}} - 1} + \frac{z}{1-z}.$$

Rearranging them, we have

$$\frac{P}{kT} = -\frac{4\pi}{h^3} \int_0^\infty dp\, p^2 \ln(1 - ze^{-\beta p^2/2m}) - \frac{1}{V}\ln(1-z)$$
$$\frac{N}{V} = \frac{4\pi}{h^3} \int_0^\infty dp\, p^2 \frac{1}{z^{-1}e^{\beta p^2/2m} - 1} + \frac{1}{V}\frac{z}{1-z},$$

in which, the integrals are in the form of Bose-Einstein functions shown in appendix C. By substituting them, and using $U = \frac{3}{2}PV$, we attained

$$\frac{U}{V} = \frac{3}{2}\frac{kT}{\lambda^3} g_{5/2}(z) - \frac{3}{2}\frac{kT}{V}\ln(1-z) \tag{4.10}$$
$$\frac{N}{V} = \frac{1}{\lambda^3} g_{3/2}(z) + \frac{1}{V}\frac{z}{1-z}. \tag{4.11}$$

Note that the second terms of equations (4.10) and (4.11) drops out as $V \to \infty$. This agrees with our finding about z never reaching to 1 for finite heat baths, since then those terms would diverge.

By multiplying equation (4.10) with V/N, we have the internal energy per particle at constant volume as

$$\frac{U}{N} = \frac{3}{2}\frac{V}{N}\frac{kT}{\lambda^3} g_{5/2}(z) - \frac{3}{2}\frac{kT}{N}\ln(1-z). \tag{4.12}$$

Chapter 5

Methods and New Ideas

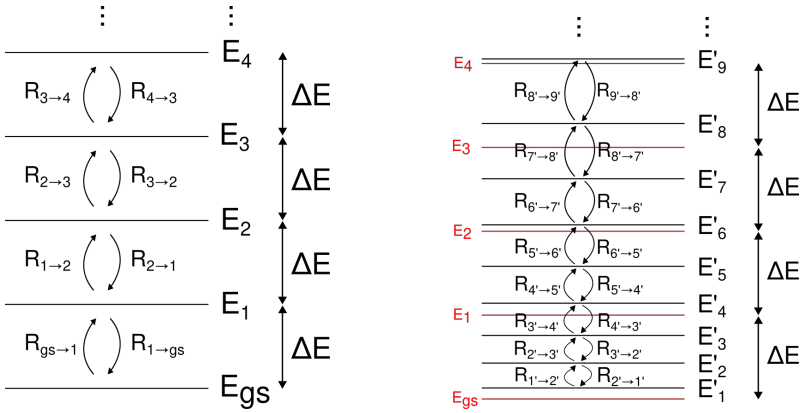
In the previous chapters, we described the model that is the base of our work, and the theoretical background that we used throughout our study. In this chapter, we presented the original work that we carried out in this thesis. Firstly we stated a problem regarding the energy temperature correspondence, and explained how we solved it. Then we described how we modified the model in order to work with finite heat baths. Lastly, we gave a short explanation on the numerical work with some important technical details.

5.1 A Note on Quantization of kT and U

As we will explain more in detail in chapter 6, equations (4.6), (4.7), and (4.12) are crucial in our study. We used them to correspond internal energy to temperature, which was used to run the simulation of the model. Even though in theory the correspondence between internal energy and temperature is continuous, for the sake of computation we quantized an interval with small step size. Although this quantization is necessary, it generates a problem with the transition rates.

Since the relation between internal energy an temperature is nonlinear, either one of them can be quantized with a constant step size, hence we need to choose.

If we had to choose internal energy to be quantized with equal intervals, then the cold heat bath would look like in figure 5.1a, where the transition rates on the left hand side and right hand side represent addition of energy to the heat bath by the demon and subtraction of energy from the heat bath by the demon, respectively. Note that from the transition rates defined on the model, we know that as $kT \to 0$, energy extraction from the cold heat bath ceases. When the cold heat bath reaches to ground state E_{gs}, it can either stays there and stops the energy extraction naturally, or absorbs ΔE amount of energy and moves up to the first excited state E_1.



(a) Quantization of U with equal intervals

(b) Quantization of kT with equal intervals

Figure 5.1: Visualization of quantization of U and kT

If, on the other hand, we quantize temperature the natural progression towards the ceasing is lost. Now, as we showed in figure 5.1b, ΔE is not equal to the difference between consecutive energy levels. Due to that, if one unit of ΔE is extracted, the state can cross more than one of the new energy levels E'_i, and land between two of them. This also causes to leave some energy in the heat bath, which is not accessible to be extracted.

Assume that after some time, the cold heat bath reaches to the energy level E'_2, which satisfies the inequality $E_{gs} < E'_2 < \Delta E$. At that energy level, the cold heat bath still has $E'_2 - E_{gs}$ amount of energy, which cannot be extracted since $\Delta E > E'_2 - E_{gs}$. This small amount of energy gives a finite temperature to the heat bath, hence a finite value for the transition rate from E'_2 to the next

one below. Given enough time, the demon would try to extract the remaining energy and fail, since it means the internal energy, and temperature to become a negative value. So at that point, the simulation should stop allowing an energy extraction. We implemented that to the simulation by hand, forbidding it to go negative values.

The remaining question is, why did we choose the problematic quantization out of two? Looking back at the equations (4.4), (4.5), (4.10), and (4.11), we observed that there is no practical way to determine kT by using U only; fugacity is also required. Of course, by definition $z = e^{\mu/kT}$, and by substituting this, we can achieve the desired correspondence between internal energy and temperature. However, μ is also a function of kT, and when we insert it to the functions, the functions become heavily complicated, and the inversion of kT by giving a value of U becomes impractical. Due to these reasons, we decided to proceed with the quantization of kT, instead of its competitor, and tried to overcome the problems that comes with it.

5.2 The Modification of the Model

After we obtained equations (4.6), (4.7), and (4.12), this is a good point to mention the modification that we did to the model. While giving the transition rates (3.1) and (3.2), Mandal et al. states that they satisfy detailed balance [1]. Although it is correct, their definition is a special case that comes with an assumption regarding the heat baths.

The most general statement that gives detailed balance is

$$\pi_d P_{d\to u} = \pi_u P_{u\to d},$$

or

$$\frac{\pi_d}{\pi_u} = \frac{P_{u\to d}}{P_{d\to u}},$$

where π_d and π_u are the equilibrium probabilities of being at the states down and up, respectively; and $P_{d\to u}$ and $P_{u\to d}$ are the transition probabilities from down to up and up to down, respectively.

The ratio of the equilibrium probabilities can also be represented as

$$\frac{\pi_d}{\pi_u} = \frac{\Gamma(E)}{\Gamma(E-\Delta)}, \tag{5.1}$$

where $\Gamma(E')$ is the volume of the phase space enclosed by the energy E', that is proportional to the number of available microstates at the energy E'. Equation (5.1) can be written as

$$\ln \frac{\pi_d}{\pi_u} = \ln \Gamma(E) - \ln \Gamma(E-\Delta).$$

Since entropy of an individual heat bath is $S(E,V) = k \ln \Gamma(E)$ [30], we have

$$\ln \frac{\pi_d}{\pi_u} = \frac{S(E)}{k} - \frac{S(E-\Delta)}{k} \Rightarrow \frac{\pi_d}{\pi_u} = e^{\frac{S(E)}{k} - \frac{S(E-\Delta)}{k}}.$$

When $E \gg \Delta$, entropy can be expanded as

$$\begin{aligned} \frac{S(E)}{k} - \frac{S(E-\Delta)}{k} &= \frac{S(E)}{k} - \left(\frac{S(E)}{k} - \frac{\Delta}{k} \frac{\partial S}{\partial E}\bigg|_E + \cdots \right) \\ &\approx \frac{\Delta}{k} \frac{\partial S}{\partial E}\bigg|_E. \end{aligned}$$

By definition, $\partial S/\partial E = 1/T$, hence

$$\frac{\pi_d}{\pi_u} = e^{\Delta/kT},$$

which gives

$$\frac{\pi_d}{\pi_u} = \frac{P_{u\to d}}{P_{d\to u}} = e^{\beta\Delta}. \tag{5.2}$$

To achieve equation (5.2), we needed to assume that $E \gg \Delta$ which is true for infinite heat baths. However, our study focuses on the behaviour of finite heat baths. As we will show in chapter 6, the condition $E \gg \Delta$ is not satisfied in some of our cases. In fermionic case, as the temperature approaches to zero, the internal energy of the heat bath approaches to Fermi energy. A comparison between the internal energy E and the energy difference between up and down states Δ shows that we can accept the assumption $E \gg \Delta$. However, in bosonic case, as the temperature approaches to zero, the internal energy also approaches to zero. In that instance, $E \sim \Delta$, and we cannot use equation (5.2). Thus instead, we used the general definition while writing our transition rates, namely

$$\frac{R_{u\to d}}{R_{d\to u}} = e^{\frac{S(E)}{k} - \frac{S(E-\Delta)}{k}},$$

and to be consistent in each case we used this form throughout the study.

Entropy of a heat bath with respect to internal energy is given by

$$S = \int_{U_{min}}^{U} \frac{dU}{kT},$$

or, we can write it as

$$S = \int_{0}^{T} \frac{dU}{dkT}\frac{dkT}{kT}.$$

Now, with the use of equations (4.6), (4.7), and (4.12), we can find the entropies of the heat baths. For fermions, it is given by

$$S = \begin{cases} \left[\frac{N\pi^2}{2}\frac{kT}{\epsilon_F} - \frac{3N\pi^4}{60}\left(\frac{kT}{\epsilon_F}\right)^3\right]\Big|_0^{kT} & \text{for low kT,} \qquad (5.3a) \\ \int_0^{kT} \frac{3N}{4kT}\left(5\frac{f_{5/2(z)}}{f_{3/2(z)}} - 3\frac{f'_{5/2(z)}}{f'_{3/2(z)}}\right)dkT & \text{otherwise.} \qquad (5.3b) \end{cases}$$

Note that the first case of fermions can be evaluated analytically, like its internal energy counterpart; however the second case that includes an integral required a numerical computation.

For bosons, on the other hand, we have one equation

$$S = \int_0^{kT} \frac{3V}{4\lambda^3 kT}\Bigg[5g_{5/2}(z) - \frac{2\lambda^3}{V}\ln(1-z) - 3(1-z)g_{3/2}(z)\left(\frac{V(1-z)g_{3/2}(z)+\lambda^3 z}{V(1-z)^2 g_{1/2}(z)+\lambda^3 z}\right)\Bigg]dkT. \qquad (5.4)$$

Again, we required a numerical computation to evaluate the integral above.

Note that at fist glance, it seems like the entropy of fermionic heat baths depend only on N, and the entropy of bosonic heat baths depend on V. However, both ϵ_F and z depend on N and V, hence in all cases bath size and number of particles alter the entropy.

There is a final point about our modification. In their article, Mandal et al. did not specify the nature of the heat baths that they used in their model. The only information we can deduce from the article comes from the phase space that they give at the end of the article. The phase space was constructed at a fixed temperature kT_c. Due to this reason, we concluded that they assumed canonical ensemble. In that case, the exchange of energy between the heat baths will not

alter the temperature of the cold heat bath, since it is infinitely large.

On the other hand, our work depends on finite heat baths. In our system, the only parameters that we assumed to be constant are internal energy U, number of particles N, and the volume of the system V, which lead us to assume microcanonical ensemble, rather than canonical ensemble. This brings an issue about the definition of temperature. In microscopic scales, temperature makes sense only when the statistical methods are applicable, i.e. when the system is in thermodynamic limit. In our study, what we did was to work with the internal energies throughout the simulations, and then find the corresponding effective temperatures for the internal energy values as if the heat baths were in thermodynamic limit.

5.3 The Refrigerator

The preceding sections were dedicated to exhibit the theoretical background of our study. In this section, we will offer a comprehensive description of the numerical work that we did to simulate the Maxwell's refrigerator, and refer to some technical details which are crucial when we elaborate the results.

The numerical work can be divided into two main components: First, by using the results that we found in sections 4.1, 4.2, and 5.2, we tabulated the values of internal energy and entropy at different temperatures which vary with small increments. Throughout the study we mainly considered dimensionless internal energy and entropy. To make internal energy dimensionless, we used ϵ_F for Fermi gases and kT_{crit} for the Bose gases. Finally, while we refer the temperature, we use the units kelvin (T) and electronvolt (kT) interchangeably; while using kelvin is better for intuitive understanding, using electronvolt in numerical work is more practical due to its small order of magnitude.

In both Fermi and Boson cases, we used finite number of non-interacting particles. Even though the theoretical work in preceding sections are valid for all Fermions and spin-0 Bosons, we needed to choose specific particles in order to fill the parameter *mass of the particle*. For the fermionic case, we chose to work with free electron gas; whereas for bosonic case, we used He^4 as our particles.

The method that we used to find the internal energy and entropy is fairly straightforward, however it differs slightly between fermionic and bosonic cases due to the numerical limits that we explained in section 4.1.1.

In fermionic case we started by calculating the Fermi energy, total internal energy and energy per particle at zero temperature. Then starting from our initial temperature and increasing it by small increments at each step, we calculated the thermal wavelength, which is given by $\lambda \approx 0.692/\sqrt{kT}$ nm for free electron gas. The next step is to determine if it is feasible to find fugacity or not. From the input parameters N and V, for each temperature step we checked the value of

$$f_{3/2}(z) = \frac{\lambda^3 N}{2V} \tag{5.5}$$

to see if we can perform the inversion. If it is infeasible, we use equations (4.7) and (5.3a) to determine internal energy and entropy. If fugacity is computable, then we find it by numerically inverting equation (5.5). By inserting the fugacity that we found into equations (4.6) and (5.3b) alongside with the temperature value, we calculate internal energy and temperature.

The procedure for bosonic case is similar, but has some discrepancies. This time we started by calculating critical temperature for the given N and V values. Then again we advanced our simulation with small temperature steps, and in each step we calculated the thermal wavelength, which is $\lambda \approx 8.101 \times 10^{-3}/\sqrt{kT}$ nm for He^4 gas. Since bosonic case does not have a numerical issue like fermionic case did, we used only equation (4.12). By inverting

$$g_{3/2}(z) + \frac{\lambda^3}{V}\frac{z}{1-z} = \frac{\lambda^3 N}{V}$$

we find the fugacity at the given temperature. Then, by inserting it into equations (4.12) and (5.4) with the corresponding temperature value, we determine internal energy and entropy.

Other than internal energy and entropy, we listed a number of functions: λ values at each temperature step were listed to ensure the heat baths satisfy the inequality $(V/N)^{1/3} \sim< r >\leq \lambda$, where $< r >$ is the average interparticle distance, and hence the heat baths are in the quantum realm. Moreover, for Bose gases we recorded both the critical volume v_c at each temperature T, and $< n_0 > /N$ to observe when the Bose-Einstein condensation occurs [29].

After we finished the listing the correspondence between temperature, internal energy and entropy, we moved to the second part of the numerical work, simulating the refrigerator. The simulation starts from given initial temperatures kT_c and kT_h. Using these with the predetermined value of ΔE, we then obtain the parameters σ and ω, which lead us to the initial transition rates.

When the first transition rates are determined, the simulation carries a loop for $b \times n$, where b is the number of initial bits, and n is the number of intervals τ is divided. Note that we use n to discretize the continuous time, which is essential for the iterative algorithm; instead of interacting with a bit for τ amount of time, the demon interacts with the bit n times.

Inside that loop, by using Monte Carlo method we mimic the behaviour of the demon. If the random process ends up with a transition, we update the internal energy, temperature, and the transition rates, and adjust the value of the bit according to the kind of the transition; else, everything stands still. After performing the first n iterations, we record the final value of the bit and move on to the next one.

Alongside the iterations, we record changes in internal energy, temperature, thermodynamic and information entropies, as well as the values of σ, ω, and ϵ, which we will use to analyze the behaviour of the system in the next chapter. Unlike the other variables, the calculation of changes in entropies requires more than a comparison between two adjacent values. To attain the statistical nature of the change in entropy, we used an interval with constant size to find the change in information entropy, and calculated the mean of the change in thermodynamic entropy in that interval. Finally we sum them up to check whether the total change in entropy is greater than zero, and verify the second law of thermodynamics.

Note that the number of iterations $b \times n$ is excessively large. Due to this reason we only record the data once in every i iteration. Moreover, to reduce the effects of momentary fluctuations we also calculated the mean of the data points around the recorded data points.

Chapter 6

Results and Discussion

6.1 Thermodynamic Properties of the Finite Heat Baths

Before presenting the results of the simulations, we exhibit some of the thermodynamic properties of the finite Fermi and Bose gases, which we attained by our theoretical and numerical work. Comparing the properties and identifying the differences between the gases gave valuable information when comparing the outcomes of the simulations.

The first property is the internal energy; figure 6.1 shows reduced internal energy per particle versus reduced temperature. In both fermionic and bosonic cases, the dashed lines represent the internal energy of the classical monatomic ideal gas, $1.5NkT$. Both cases converge to the classical ideal gas as $T \to \infty$, while as $T \to 0$ the quantum behaviour shows up: Internal energy of Fermi gas goes to 0.6 which can be seen from equation (4.1), whereas internal energy of Bose gas goes to 0 in a nonlinear fashion. Note that both of the graphs show the internal energies are strictly increasing functions, however while fermionic case is a convex function, bosonic case changes convexity at critical temperature, which is more observable in the next properties.

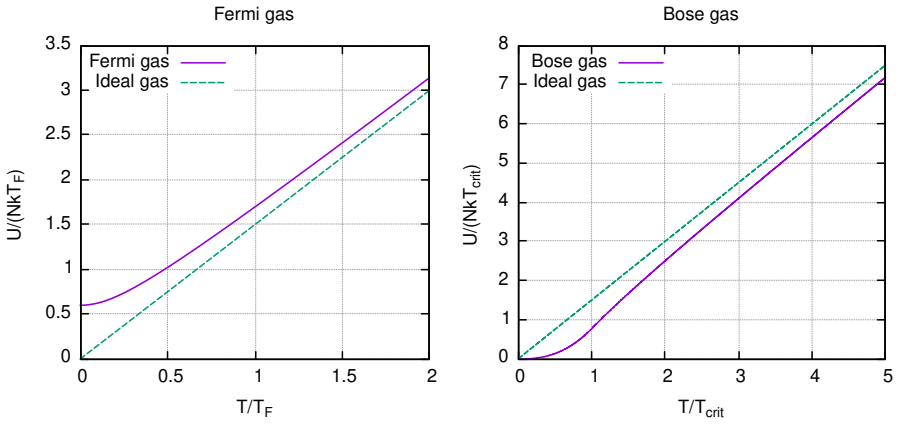


Figure 6.1: Reduced internal energy per particle versus reduced temperature graphs

The second property is the heat capacity of the gases at constant volume. Note that heat capacity is given by

$$C_v = \left(\frac{\partial U}{\partial T}\right)_v,$$

hence it gives information regarding the internal energy function. Figure 6.2 confirms the findings above; heat capacity of Fermi gas is a strictly increasing function, since internal energy is a strictly increasing convex function. Moreover, heat capacity of Bose gas shows where exactly convexity of internal energy function changes; when $T < T_{crit}$ heat capacity is strictly increasing, around $T = T_{crit}$ it has a global maximum, and when $T > T_{crit}$ it strictly decreases. This confirms with the change of convexity of the internal energy at critical temperature. We also confirm that as $T \to \infty$, $C_v \to 1.5$; which we showed analytically (see appendix E). Note that in thermodynamic limit, the global maximum is exactly at the point $T = T_{crit}$, however our consideration of finite gases slightly deviates where the peak occurs from $T = T_{crit}$ towards $T > T_{crit}$. The reason of this shift is the definition of critical temperature that we gave in section 4.2, which is

$$T_{crit} = \frac{2\pi\hbar^2}{mk}\left(\frac{N}{V\zeta(3/2)}\right)^{3/2}.$$

This definition is exact only for systems in thermodynamic limit, since it is given for $z \to 1$ [29]. For finite bosonic gases, condensation occur when z is close to,

but not equal to 1, due to that critical temperature of the system slightly varies. Since finding the exact value of z where condensation starts is not possible, we used the regular definition of critical temperature.

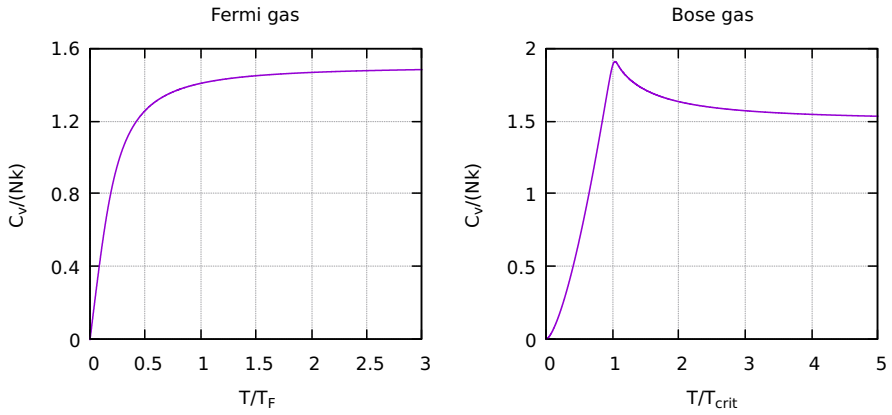


Figure 6.2: Reduced heat capacity per particle versus reduced temperature graphs

The final property is the internal entropy of the heat baths. We calculated the entropies from the equations (5.3a), (5.3b), and (5.4), and plotted them on figure 6.3. Note that while the entropy graph of Fermi gas acts as its heat capacity counterpart, the graph of Bose gas acts as its internal energy counterpart.

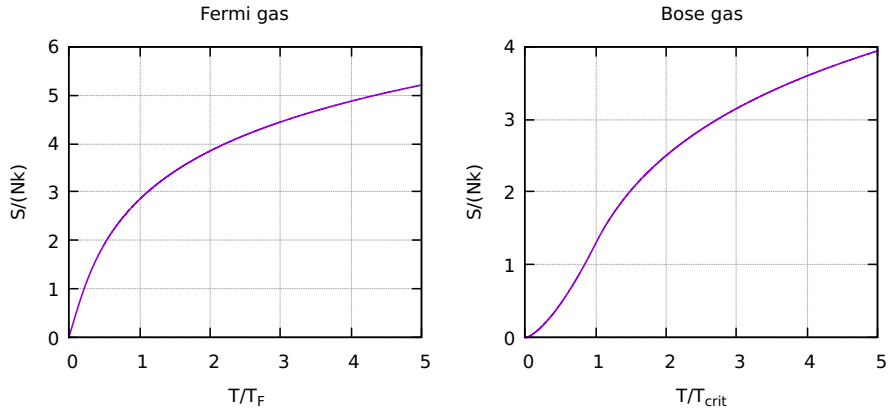


Figure 6.3: Reduced internal entropy per particle versus reduced temperature graphs

The characteristics of the Fermi and Bose gases differ in all cases for low temperature, and converges to the classical ideal gas in high temperatures. Due to this discrepancy, we expect the gases to act similar at high temperatures, but vary as the temperature of the cold heat bath starts to decrease and the quantum effects become more visible.

6.2 Simulations of the Refrigerator

The simulation contains a number of parameters that affect the outcome when they are altered. We classify these parameters under two groups according to which part of the simulation they belong to: Heat bath parameters and model parameters. Heat bath parameters are the ones that affect the aforementioned thermodynamic properties, which are N and V; whereas model parameters, which we introduced in sections 3.3 and 5.3, are the ones that affect the behaviour of the demon without changing the characteristics of the heat baths.

As we mentioned in section 3.3, refrigeration occurs only when $\delta > \epsilon$. Thus they are two model parameters that are of great interest to us. Note that ϵ is derived from ω and σ, which are related to kT_c and kT_h. Hence we can observe these parameters interchangeably. Moreover, τ is also implicitly responsible for the amount of energy transferred, and γ is responsible of setting a time scale to the model by regulating the number of intrinsic transitions, therefore adjusting their values results with change in time it takes to reach to the equilibrium. By using the parameters δ, ϵ, τ, and γ, we can work out the dynamics of the model.

There is one model parameter which we fixed throughout the simulations: Energy difference between the two states of the demon is fixed to $\Delta E = 10^{-4}$ eV, since it only affects the amount of energy transferred alongside a transition linearly, and has no effect on the model's behaviour.

In the next two section we will present our results on what behavioural changes do the parameters cause. Toward that end, we gave the parameters initial values, which will act as a control group. Then by altering each parameter separately, we will be able to observe what effects does that parameter have on the refrigerator. For both fermionic and bosonic heat baths, we assign the control parameters as

$N = 8000$, $V = 1000\ nm^3$, $\delta = 1$, $kT_c = kT_h = 0.025\ eV$, $\tau = 0.5$, and $\gamma = 1$. Note that for each parameter change, we feed the demon with enough bits so that the system can reach its stable state, if possible.

6.2.1 Model Parameters

The first parameter that we consider is the number of particles in each bath. Changing N alters both initial internal energy and final internal energy that the cold heat bath attains. By defining the extracted energy from the cold heat bath as $\Delta U_c = U_{c,initial} - U_{c,final}$, we observe how N affects ΔU_c, and the time it takes for the system to reach equilibrium t_{eq}. For various N, tables 6.1 and 6.2 lists ΔU_c, t_{eq}, and $kT_{c,final}$ values found from the simulations with fermionic and bosonic heat baths, respectively.

N	$\epsilon_F\ (eV)$	$\Delta U_c\ (eV)$	t_{eq}	$kT_{c,f}\ (eV)$
6000	1.204251	7.679829	274599.99	0.000086
7000	1.334591	8.086365	287199.99	0.000033
8000	1.458847	8.451434	298749.99	0.000016
9000	1.578016	8.790604	313449.99	0.000050
10000	1.692842	9.104347	323949.99	0.000055

Table 6.1: Table of N, ϵ_F, ΔU_c, t_{eq}, and $kT_{c,f}$ for simulations with fermionic heat baths

N	$kT_{crit}\ (eV)$	$\Delta U_c\ (eV)$	t_{eq}	$kT_{c,f}\ (eV)$
6000	0.00011424	224.966997	8966049.99	0.00000847
7000	0.00012661	262.455393	10310049.99	0.00000851
8000	0.00013839	299.942528	11948049.99	0.00000842
9000	0.00014970	337.426817	12872049.99	0.00000861
10000	0.00016059	374.909914	13964049.99	0.00000845

Table 6.2: Table of N, kT_c, ΔU_c, t_{eq}, and $kT_{c,f}$ for simulations with bosonic heat baths

Because ΔU_c depends on the internal energy, we expect it to be proportional to N. Indeed for fermionic heat baths, from equation (4.4) we derive the theoretical relationship as

$$\Delta U_c = 3V\left(\frac{kT_{c,i}}{\lambda_{c,i}^3} f_{5/2}(z_i) - \frac{kT_{c,f}}{\lambda_{c,f}^3} f_{5/2}(z_f)\right), \tag{6.1}$$

which is

$$\Delta U_c = \frac{\pi^2 N}{4\epsilon_F}[(kT_{c,i})^2 - (kT_{c,f})^2] + \mathcal{O}(T^4) \tag{6.2}$$

for the temperature values that we are working with, as we showed in appendix F. Since $\epsilon_F \propto N^{2/3}$, we have $\Delta U_c \propto N^{1/3}$. Moreover, by inserting V, $kT_{c,i}$ into equation (6.2), and assuming $kT_{c,f}$ is approximately zero, we find

$$\Delta U_c \approx 0.423 N^{1/3}\ eV.$$

To compare this with the numerical results, we interpolate the data points and find the curve passing through the points as

$$\Delta U_c \approx 0.423 N^{1/3}\ eV,$$

which agrees with the theoretical equation. Figure 6.4 shows the graph of ΔU_c versus N.

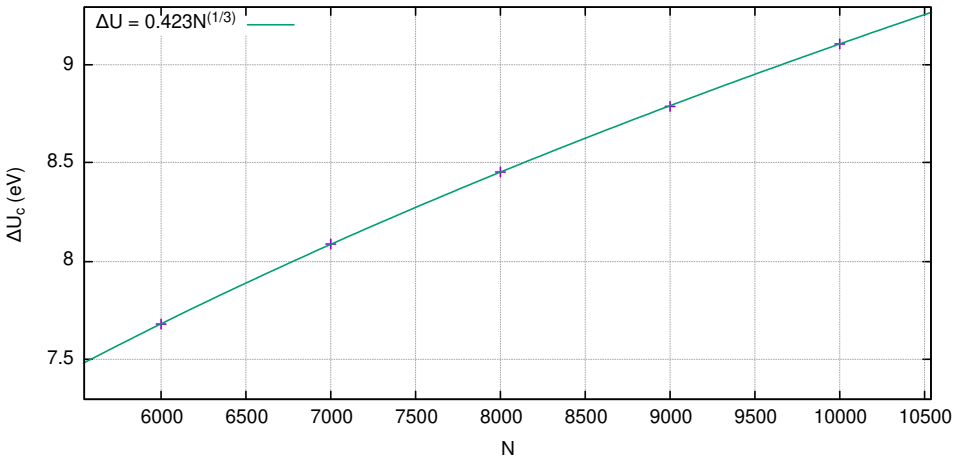


Figure 6.4: Graph of ΔU_c vs. N for fermionic systems

Before starting the analysis for the bosonic case, we need to mention an important distinction. While we study the fermionic case, the internal energy of cold

heat bath become stationary once there is no energy left to extract. However in bosonic case, even though the graphs seem to be stationary, there is still energy to be extracted. The extraction of the remaining energy is highly unlikely, and requires immensely long time, which makes the graphs unintelligible. Due to this reason, throughout the analysis, we will plot our graphs until the energy extraction slows down, and comment on the slow energy extraction part on section 6.3.

For bosonic case, predicting the theoretical relationships between ΔU_c and N is non-trivial, since the calculations we did for fermionic heat baths in appendix F cannot be replicated with the complicated formulae of bosonic heat baths. Thus instead, we will follow a qualitative approach. By plotting the data given in table 6.2, we deduce that the relationship between ΔU_c and N is linear, which we show in figure 6.5.

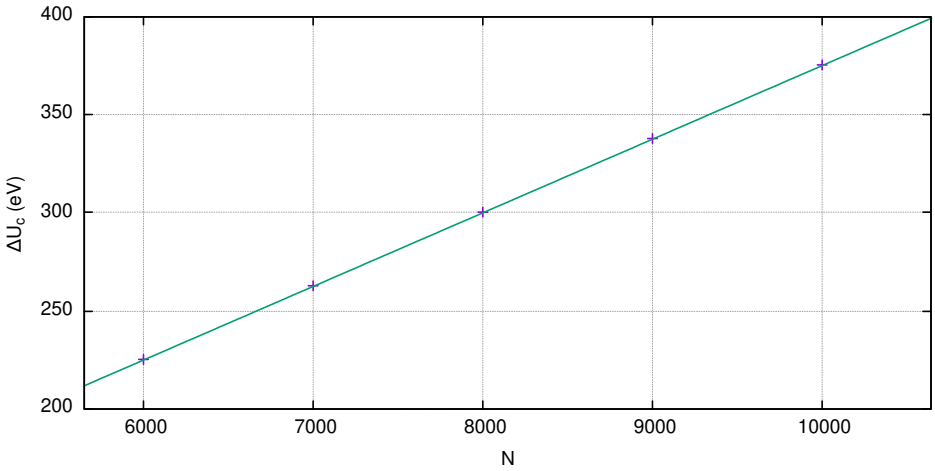


Figure 6.5: Relation between ΔU_c & N for bosonic heat baths

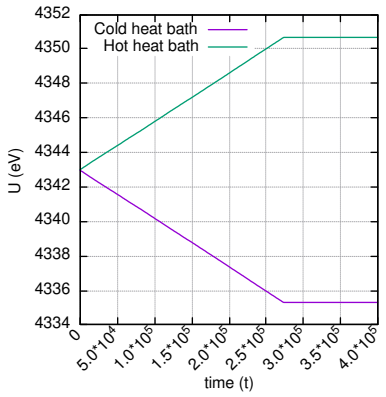
Finally, we give the graphs for varying N. Note that our analysis mostly focuses on the internal energy. For this reason, throughout this study, we give only the graphs of internal energy for all values of the parameters. On the other hand, since both temperature and change in entropy depend on internal energy, their behaviour also resembles. In particular, we observed that most of the parameters has no effects on temperature and entropy other than changing the t_{eq} by stretching the graph. Due to this reason, if a separate analysis is not required, we give graphs of temperature and entropy only for control parameters as an example of

what they look like.

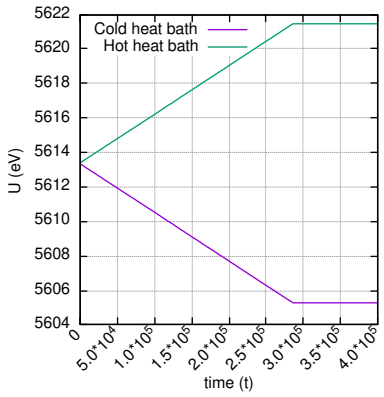
In figures 6.6 and 6.7, we give the internal energies of fermionic and bosonic heat baths, respectively. Moreover, in figure 6.8 we show the change of temperature of fermionic and bosonic heat baths at control parameters; and in figure 6.9 we give the change in entropies for simulations consisting fermionic and bosonic heat baths at control parameters.

Before going to the next parameter, there is an important note to mention regarding the internal energy graphs. Even though we expected that the extraction of energy to slow down as temperature of cold heat bath approaches to zero, our graphs shows us it proceeds like a linear graph. We will thoroughly examine why heat bahts behave like this in section 6.3.

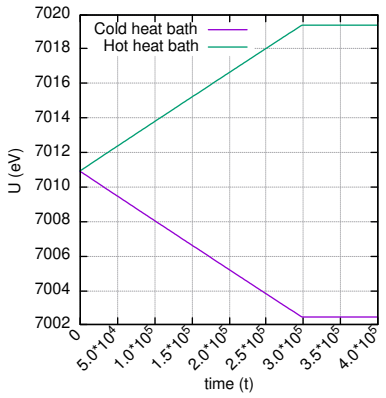
6.2.1.1 Graphs for various N values



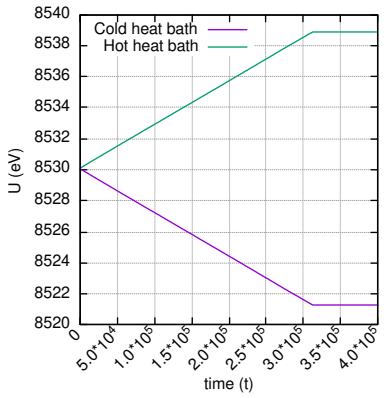
(a) $N = 6000$



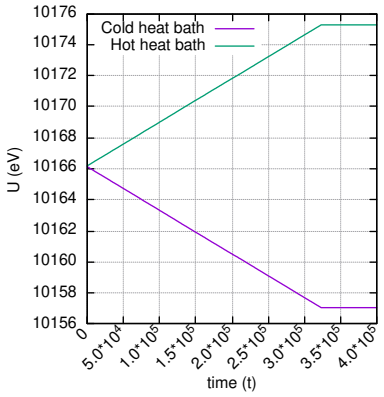
(b) $N = 7000$



(c) $N = 8000$

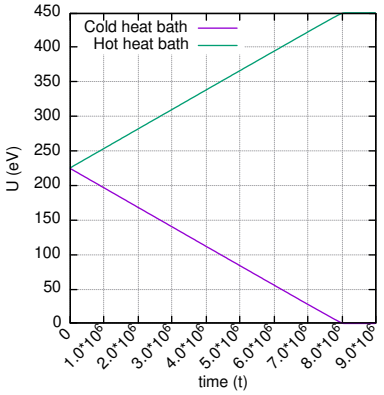


(d) $N = 9000$

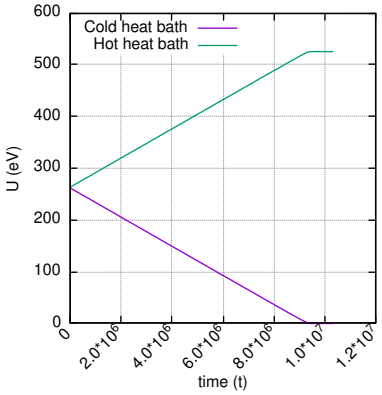


(e) $N = 10000$

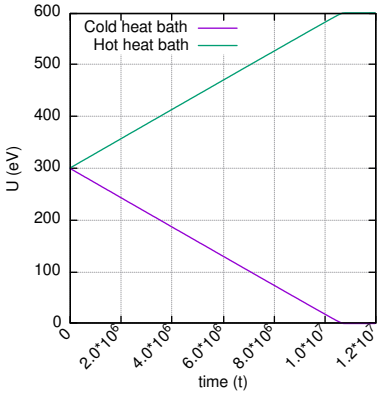
Figure 6.6: U vs. t graphs of fermionic heat baths for various N values



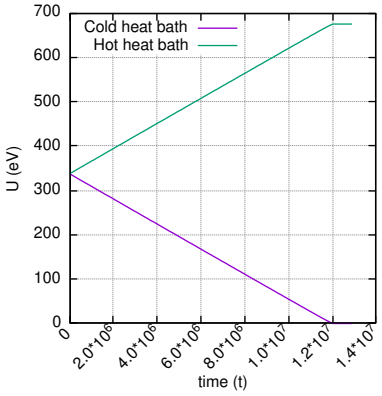
(a) $N = 6000$



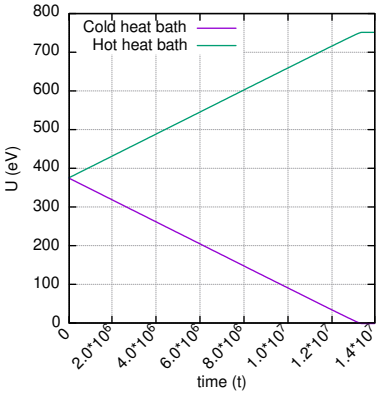
(b) $N = 7000$



(c) $N = 8000$

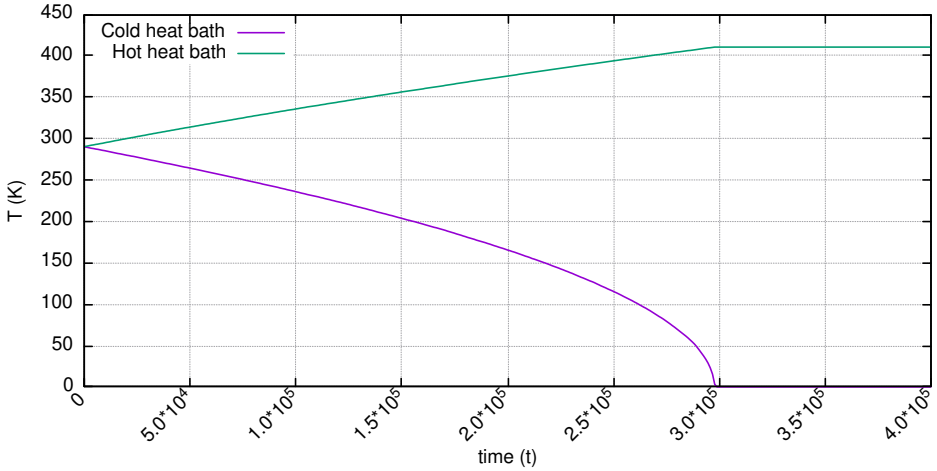


(d) $N = 9000$

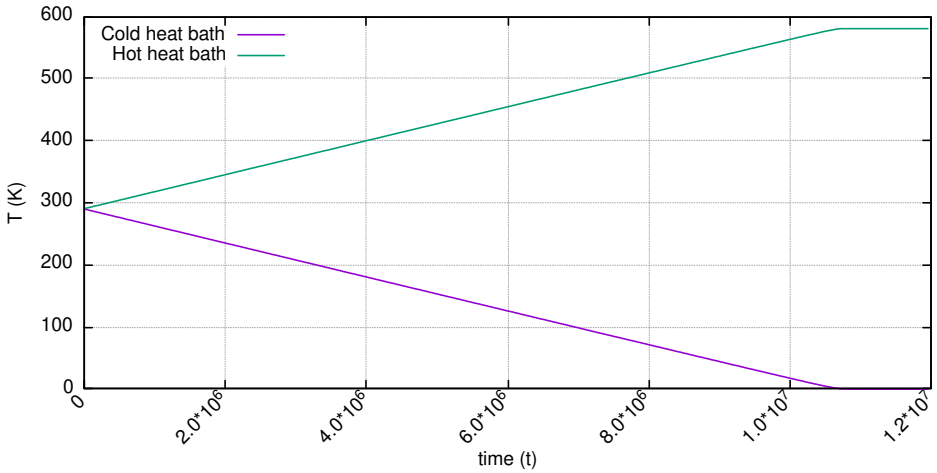


(e) $N = 10000$

Figure 6.7: U vs. t graphs of bosonic heat baths for various N values

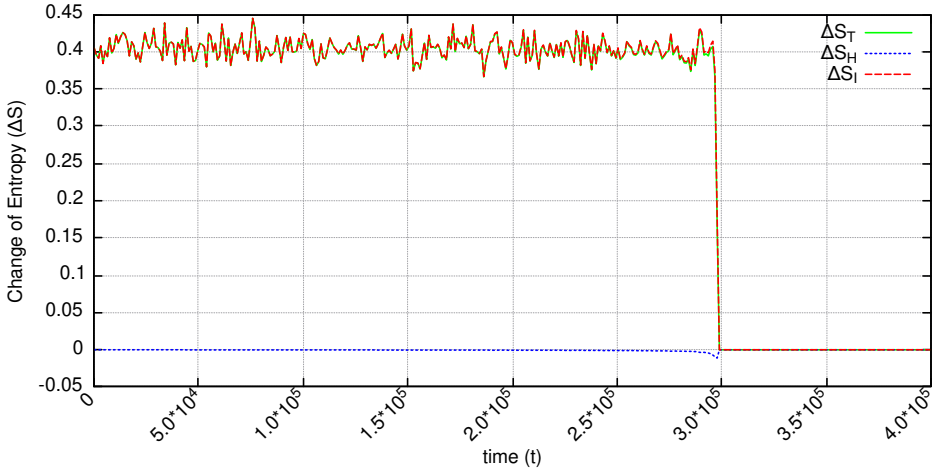


(a) Fermionic heat baths

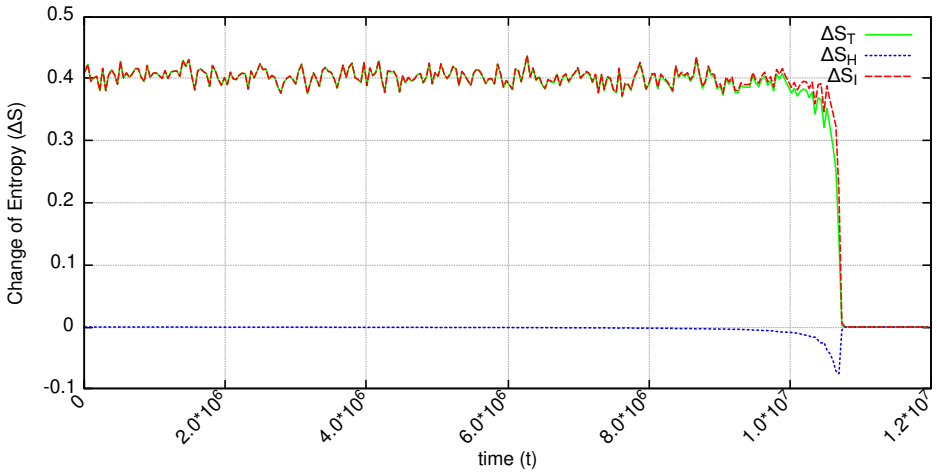


(b) Bosonic heat baths

Figure 6.8: T vs. t graphs at control parameters



(a) Fermionic heat baths



(b) Bosonic heat baths

Figure 6.9: ΔS vs. t graphs at control parameters

The second parameter that we study is the volume of a heat bath. Following tables display the data gathered from the simulations for fermionic and bosonic heat baths.

$V\ (nm^3)$	$\epsilon_F\ (eV)$	$\Delta U_c\ (eV)$	t_{eq}	$kT_{c,f}\ (eV)$
512	2.279448	5.410956	190599.99	0.000041
729	1.801045	6.849229	243099.99	0.000072
1000	1.458847	8.451434	298749.99	0.000016
1331	1.205658	10.227964	361749.99	0.000058
1728	1.013088	12.166447	432099.99	0.000023

Table 6.3: Table of V, ϵ_F, t_{eq}, and ΔU_c for simulations with fermionic heat baths

$V\ (nm^3)$	$kT_{crit}\ (eV)$	$\Delta U_c\ (eV)$	t_{eq}	$kT_{c,f}\ (eV)$
512	0.00021624	299.887922	10856049.99	0.00000913
729	0.00017086	299.920868	10940049.99	0.00000921
1000	0.00013839	299.942024	11948049.99	0.00000842
1331	0.00011437	299.956009	13292049.99	0.00000831
1728	0.00009611	299.965517	11906049.99	0.00000841

Table 6.4: Table of V, ϵ_F, t_{eq}, and ΔU_c for simulations with bosonic heat baths

For fermionic gas, varying V changes the internal energy, hence we expect to observe effects similar to the previous case. The results given in the table 6.3 agrees with our predictions.By substituting the Fermi energy formula (4.1) into equation (6.2), we get the theoretical relationship between V and ΔU_c as

$$\Delta U_c \approx \left(\frac{\pi}{3}\right)^{2/3} \frac{m_e N^{1/3}}{2\hbar^2} V^{2/3}[(kT_i)^2 - (kT_f)^2].$$

Inserting the appropriate values for the constants and parameters for each trial, and assuming $kT_{c,f}$ is approximately zero, we get

$$\Delta U_c \approx 0.0846 V^{2/3}\ eV.$$

On the other hand, by interpolating the data points and finding the curve passing through it, we find

$$\Delta U_c \approx 0.0845 V^{2/3}\ eV,$$

which again agrees with the theoretical prediction. We plot the curve of ΔU_c versus V, which is given in figure 6.6

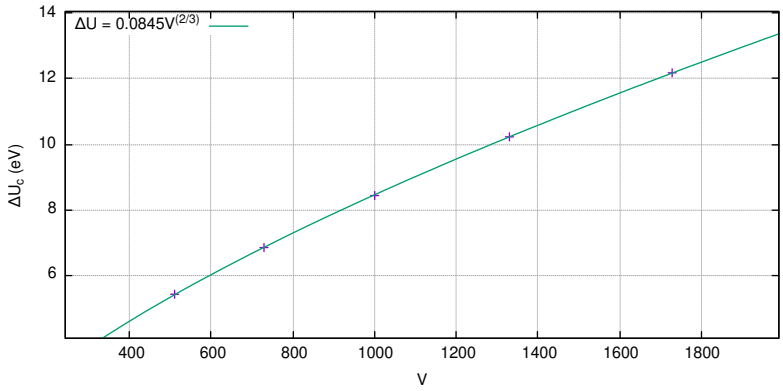


Figure 6.10: Graph of ΔU_c vs. V for fermionic systems

For bosonic case, finding a theoretical relationship between ΔU_c versus V is again non-trivial. Hence, we find it from the data given in table 6.4. From the plotting that we give in figure 6.11, we observe that changing volume has no significant effect on ΔU_c.

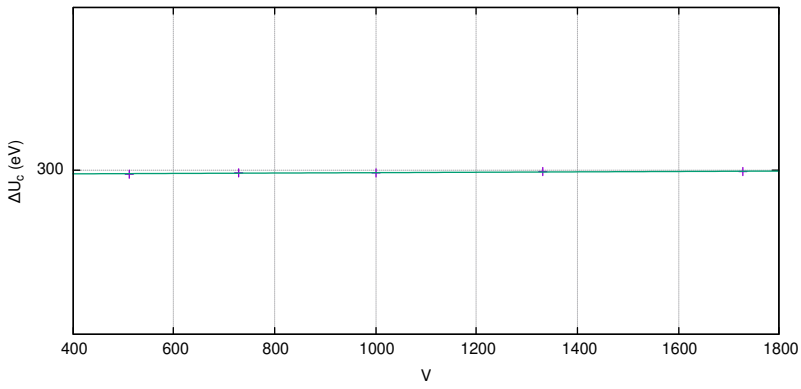
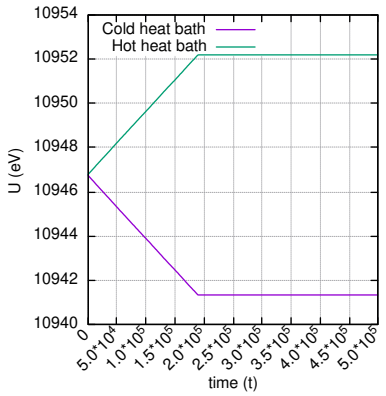


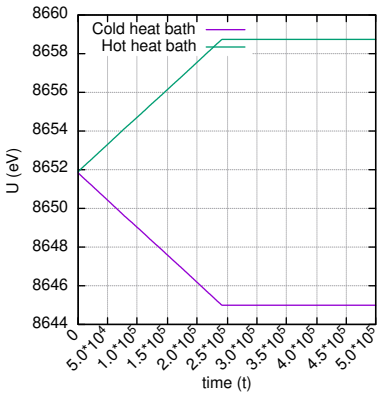
Figure 6.11: Relation between ΔU_c & V for bosonic heat bahts

We finally give the graphs for varying volume. Note that as we expressed earlier, giving the temperature and entropy graphs of systems in control parameters is sufficient for the analysis of the heat bath parameters. For this reason, we did not put the temperature and entropy graphs of fermionic and bosonic systems, as they are the same with the figures 6.8 and 6.9. Thus, we conclude this section by presenting the internal energy graphs, given in figures 6.12 and 6.13.

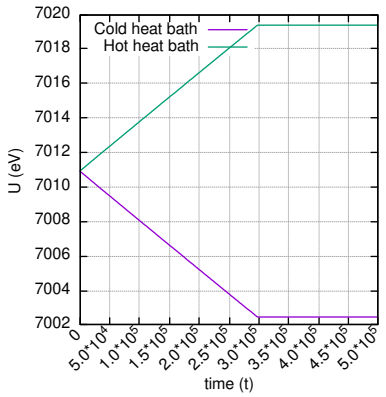
6.2.1.2 Graphs for various V values



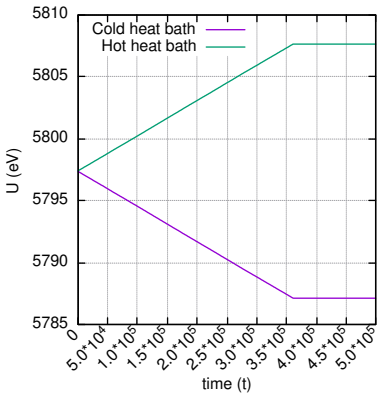
(a) $V = 512\ nm^3$



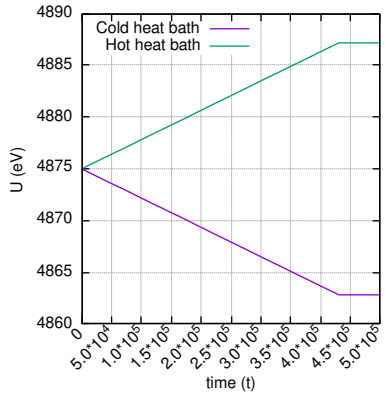
(b) $V = 729\ nm^3$



(c) $V = 1000\ nm^3$

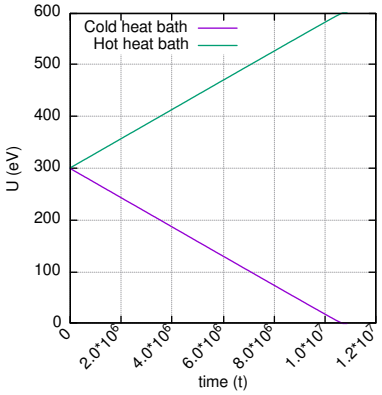


(d) $V = 1331\ nm^3$

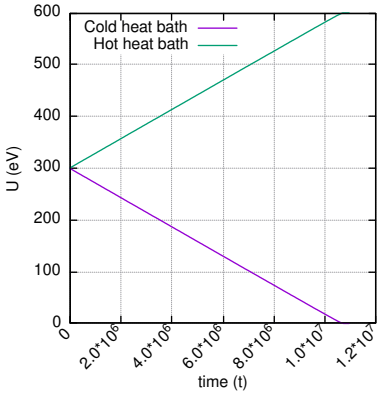


(e) $V = 1728\ nm^3$

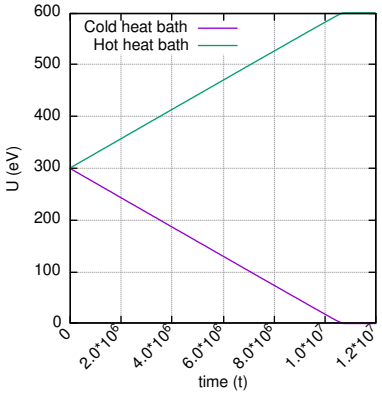
Figure 6.12: U vs. t graphs of fermionic heat baths for various V values



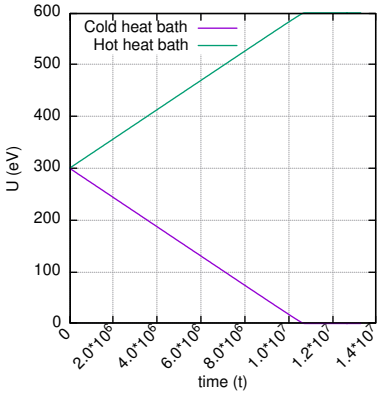
(a) $V = 512\ nm^3$



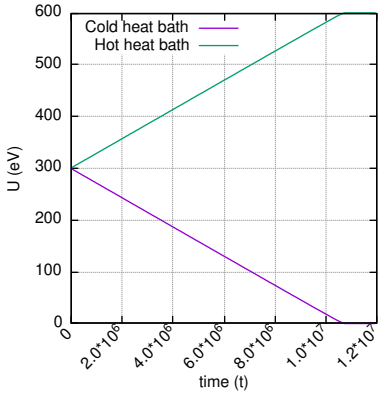
(b) $V = 729\ nm^3$



(c) $V = 1000\ nm^3$



(d) $V = 1331\ nm^3$



(e) $V = 1728\ nm^3$

Figure 6.13: U vs. t graphs of bosonic heat baths for various V values

6.2.2 Heat Bath Parameters

Now we start analyzing the model parameters. The first parameter that we consider is δ. Note that since refrigeration occurs in the region $0 \leq \delta \leq 1$, we only considered that range of the parameter. Figures 6.14 and 6.15 show the advancement of internal energies of fermionic and bosonic heat baths in time for different δ values while keeping every other parameter at the control value.

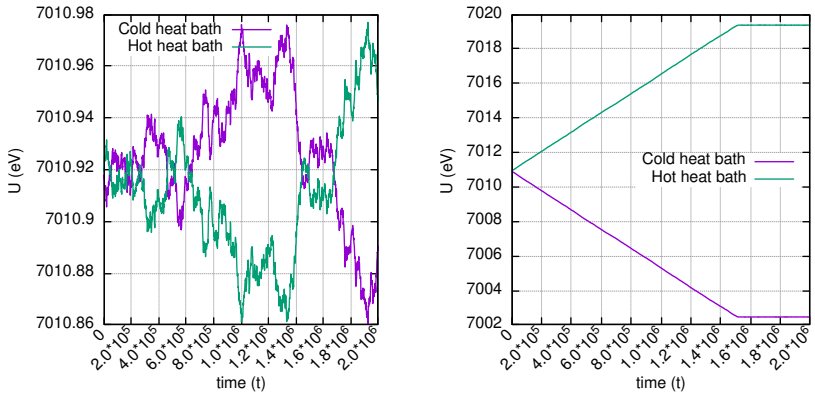
As expected, in both fermionic and bosonic cases, lowering the value of δ decreases the efficiency of the refrigerator as well by extending the time it requires to reach equilibrium point. However, we recognized a difference between two particle types: In fermionic systems, for $\delta > 0$, all refrigerators reaches to the same lowest point. On the other hand, bosonic systems reaches to their lowest point only if $\delta = 1$. For all other cases, bosonic systems stop extracting energy, even though there is energy to remove from the cold heat bath. For comparison, we list the energy left on the cold heat bath when the simulation is over in table 6.5. The reason of this is the quantization of kT, which we described in 5.1. In our simulations, only fermionic systems requires a manual intervention before trying to lower their energies to negative values; on the contrary, bosonic systems reach their equilibrium point naturally. Due to this difference, the balance between δ and ϵ is less preserved in fermionic cases. Consequently, ϵ jumps to a value larger than δ as the energy of the cold heat bath reaches to its lowest point, and the demon switches into an eraser. In the bosonic cases, the natural decrease of the internal energy causes a smooth transition between the regions $\delta > \epsilon$ and $\delta < \epsilon$, so that before cold heat bath reaches to lowest energy, the demon transforms into an eraser.

δ	$U_{c,final}$ (eV)
0.2	2.343136
0.4	0.565792
0.6	0.171098
0.8	0.056992
1.0	0.000095

Table 6.5: Table of $U_{c,final}$ of bosonic heat baths for simulations with various δ

The different behaviour that we described above also affects the entropy production after reaching to equilibrium. In fermionic systems, ϵ jumps suddenly to the values above δ. For that reason, the demon starts to erase bits abruptly, and produces thermodynamic entropy. In contrast, the natural process that occurs on bosonic systems keeps the produced entropy on a lower level, since the increase in thermal entropy pushes the demon back to the refrigeration region. In figures 6.16 and 6.17, we present the differences between entropy production and the balance between δ and ϵ. Note that for better visual representation, we smoothed the change in entropy curves.

6.2.2.1 Graphs for various δ values

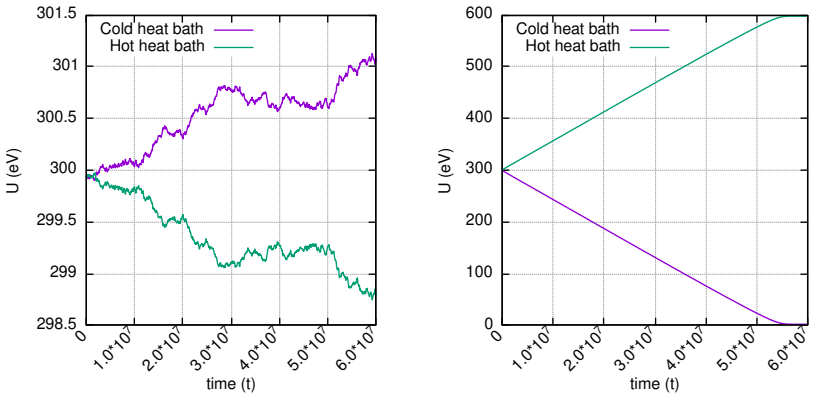


(a) $\delta = 0$ (b) $\delta = 0.2$

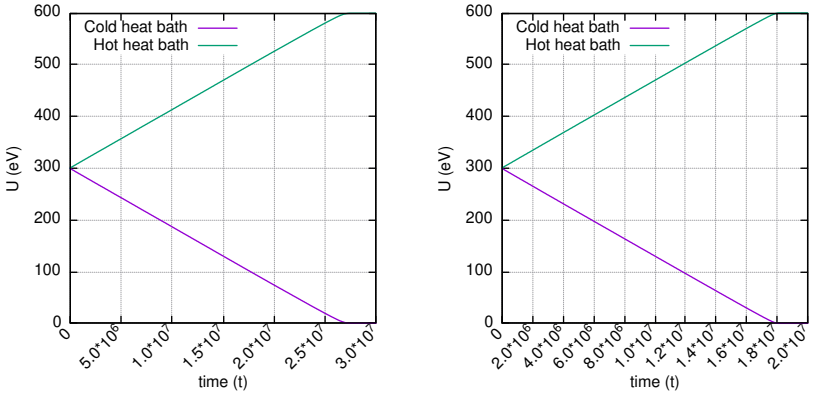
(c) $\delta = 0.4$ (d) $\delta = 0.6$

(e) $\delta = 0.8$ (f) $\delta = 1$

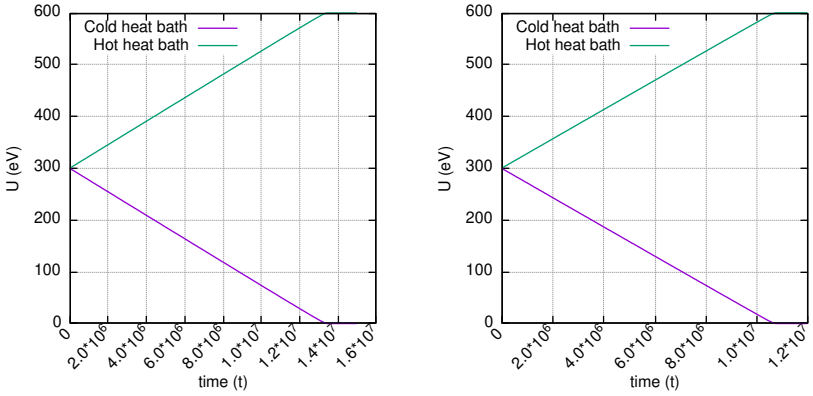
Figure 6.14: U vs. t graphs of fermionic heat baths with various δ values



(a) $\delta = 0$ (b) $\delta = 0.2$

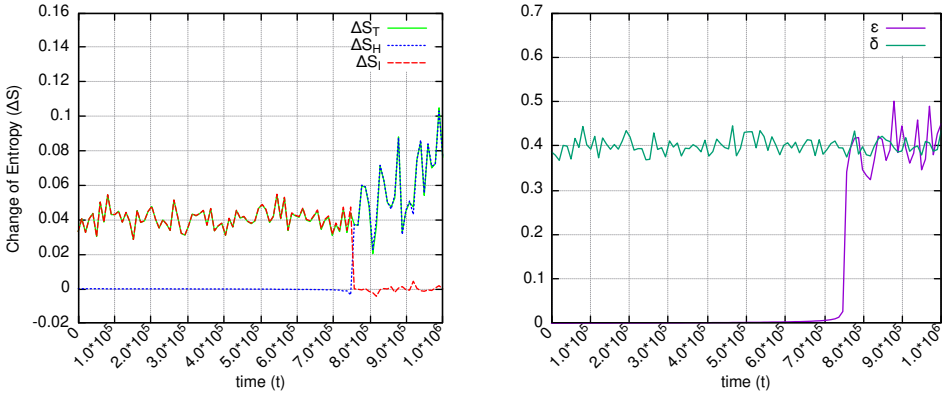


(c) $\delta = 0.4$ (d) $\delta = 0.6$

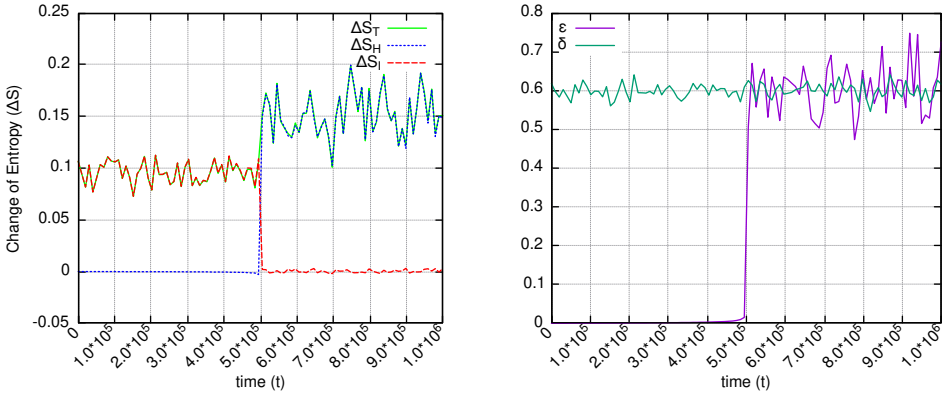


(e) $\delta = 0.8$ (f) $\delta = 1$

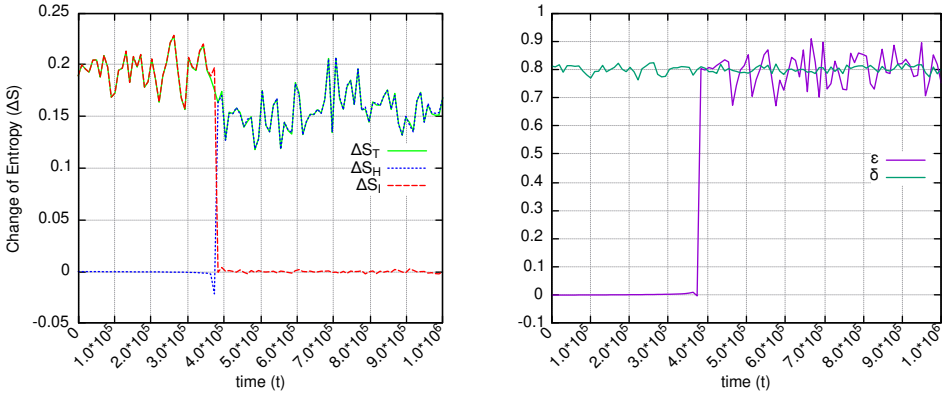
Figure 6.15: U vs. t graphs of bosonic heat baths with various δ values



(a) $\delta = 0.4$

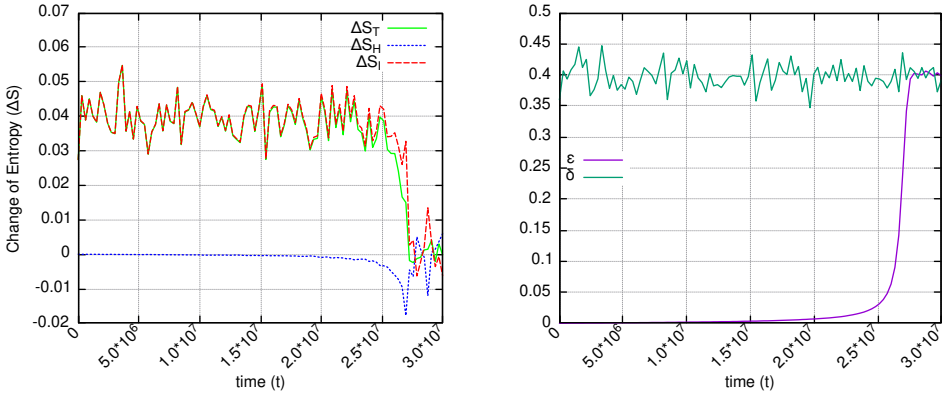


(b) $\delta = 0.6$

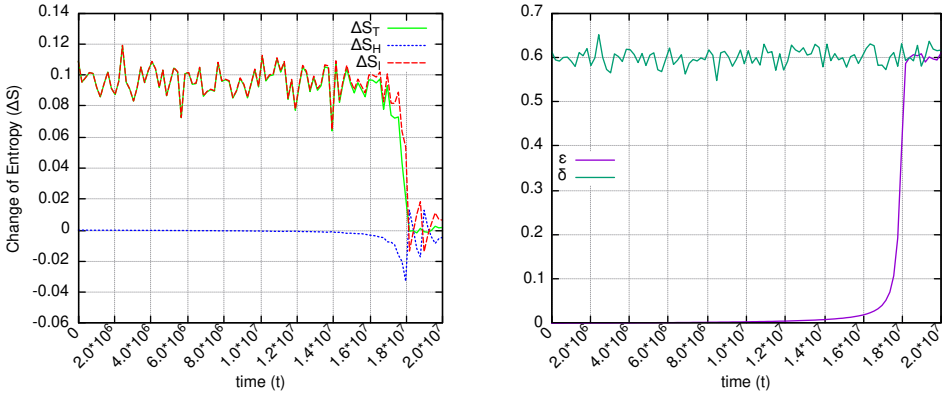


(c) $\delta = 0.8$

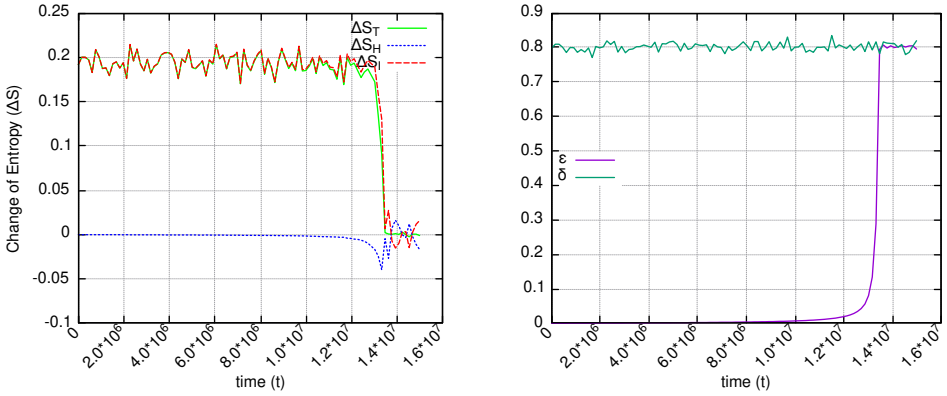
Figure 6.16: S vs. t & ϵ vs. t graphs of simulations with fermionic heat baths for various δ values



(a) $\delta = 0.4$



(b) $\delta = 0.6$



(c) $\delta = 0.8$

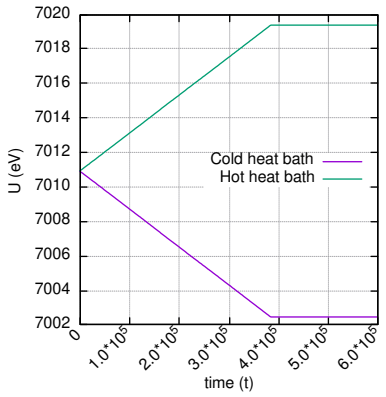
Figure 6.17: S vs. t & ϵ vs. t graphs of simulations with bosonic heat baths for various δ values

Next, we examine γ, which sets a time scale to the model by adjusting the number of intrinsic transitions as mentioned before. Because of its duty, increasing γ increases the probability of the demon being at the d state. Since being at d increases the chance of carrying energy from cold heat bath to hot heat bath according to the dynamics of our system, we expect that as γ increases, the time it takes to reach equilibrium shortens. Indeed, the graphs given in figures 6.18 and 6.19 agree with our expectations.

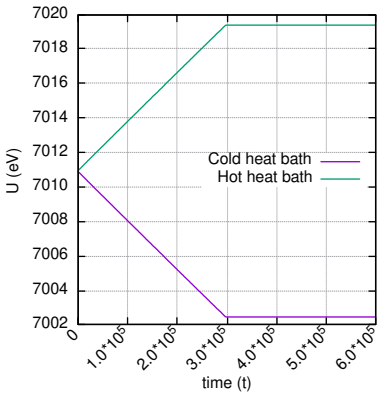
Note that as γ starts to raise from a small value its effect is more observable. However, as Mandal et al. showed in Supplementary Material of [1], when $\gamma \to \infty$, the equation governing the energy transfer converges to a finite point, and the effect of increasing γ gets less observable.

Moreover, as energy extraction from cold heat bath increases, entropy production should increase as well. Hence, increasing γ also increases the entropy production per time, as we demonstrate in figures 6.20 and 6.21. Like internal energy, amount of entropy produced and the time it takes to reach equilibrium converge to a finite value as γ increases.

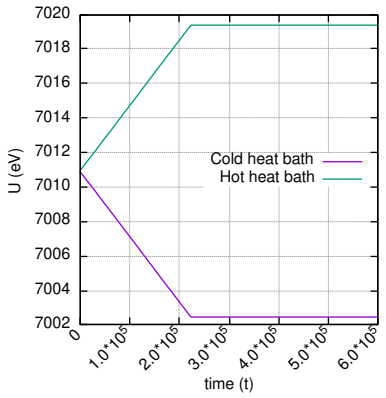
6.2.2.2 Graphs for various γ values



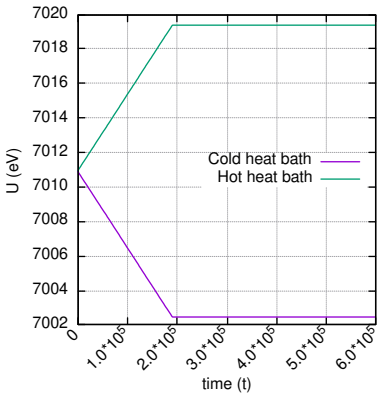
(a) $\gamma = 0.5$



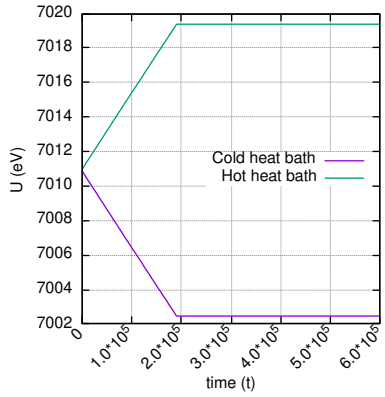
(b) $\gamma = 1$



(c) $\gamma = 10$

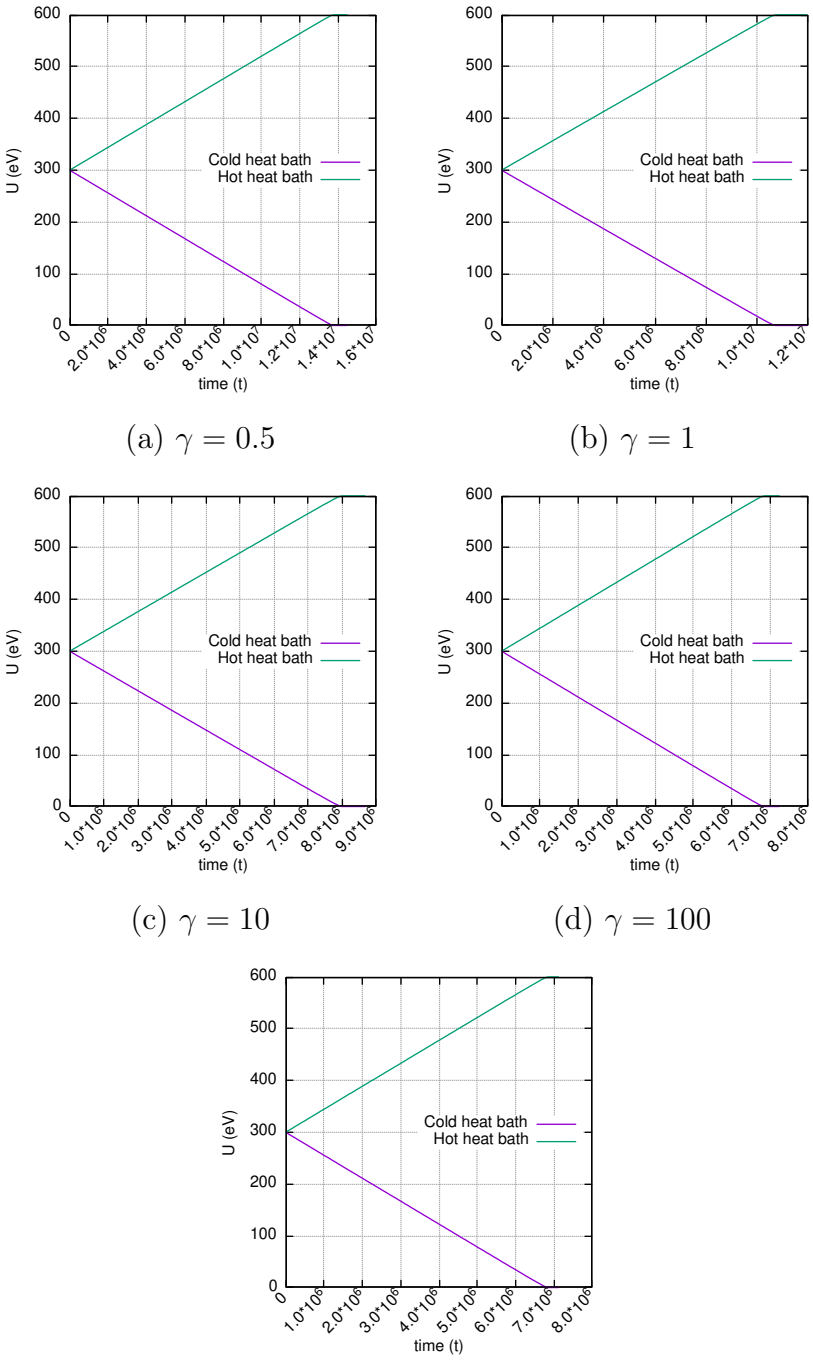


(d) $\gamma = 100$



(e) $\gamma = 1000$

Figure 6.18: U vs. t graphs of fermionic heat baths for various γ values



(a) $\gamma = 0.5$

(b) $\gamma = 1$

(c) $\gamma = 10$

(d) $\gamma = 100$

(e) $\gamma = 1000$

Figure 6.19: U vs. t graphs of bosonic heat baths for various γ values

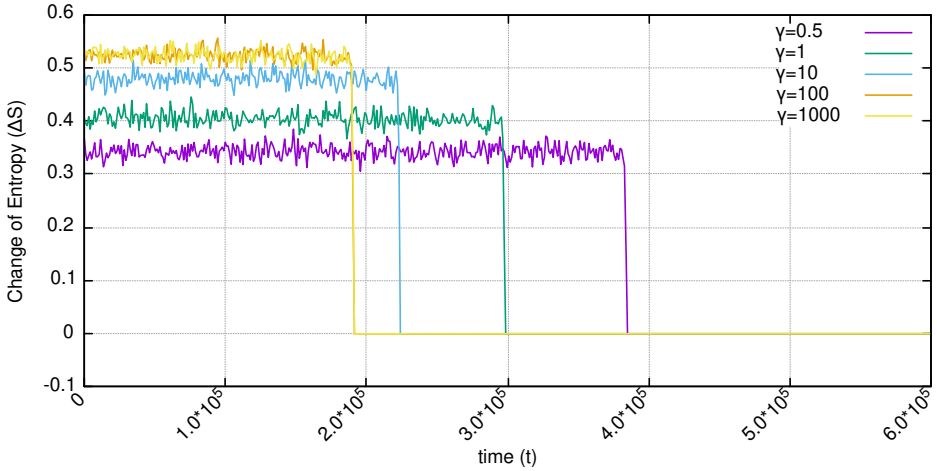


Figure 6.20: ΔS_T graphs of simulations with fermionic heat baths for various values of γ

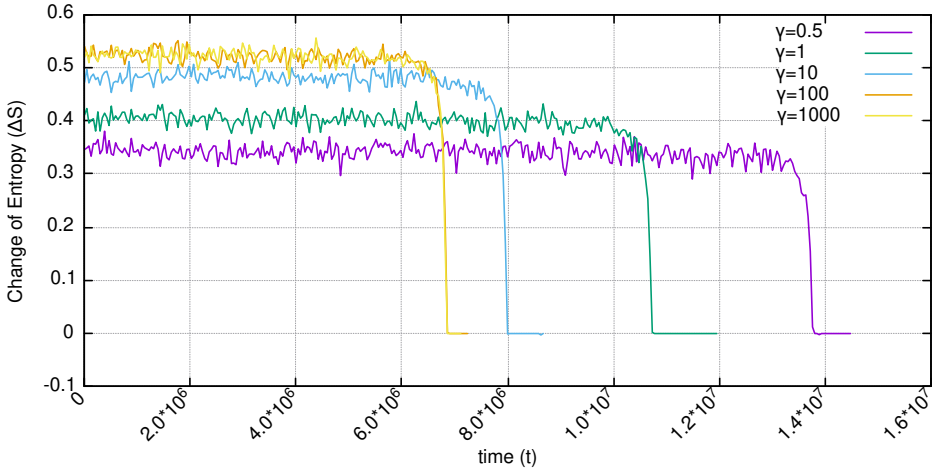


Figure 6.21: ΔS_T graphs of simulations with bosonic heat baths for various values of γ

The next parameter that we investigate is τ; however before giving the results, we need to point out a significant note. In section 5.3, we mentioned another parameter n, which we used to discretize the continuous time: It represents the maximum number of transitions that the system can perform in τ amount of time. Throughout all simulations we kept it as a constant at $n = 50$. However, in a real world scenario with continuous time, by changing τ we affect the number

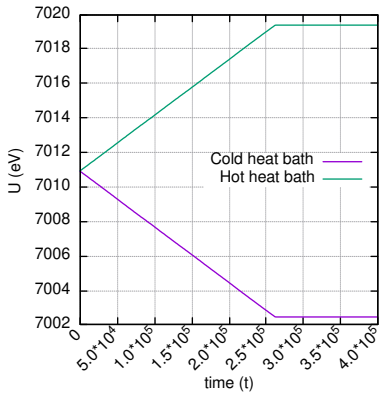
of possible interactions between the demon and the bits. Due to this reason, only for this case, as we change τ, we also change n with the same proportion. By doing so, we simulate the real life scenario better; and keep $\Delta t = \tau/n$ constant in all simulations, which standardizes the magnitude of the transition rates.

For various values of τ, we exhibit the graphs of internal energies in figures 6.22 and 6.23. From these graphs, we observe that as τ increases, time it takes to reach equilibrium point extends as well. This is solely because the demon spends much more time interacting with each bit, which slows down the dynamics of the system.

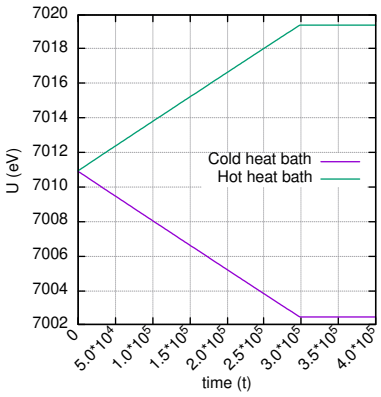
However, the effect of increasing τ has an opposite effect on total entropy production. As we observed in γ, if the time it takes to reach equilibrium decreases, the entropy production per time increases, so that the total entropy production stays in the same order. For τ, on the other hand, the opposite is true. Both the time it takes to reach equilibrium and the total entropy production are increased, as we showed in figures 6.24 and 6.25.

As Mandal et al. showed in the Supplementary Material of [1], extending τ increases the energy transfer from cold to hot heat bath per interaction interval. Then, the production of 1s should also increase, which causes a higher information entropy production. However, due to the longer τ values, total information entropy production spreads over a longer time period.

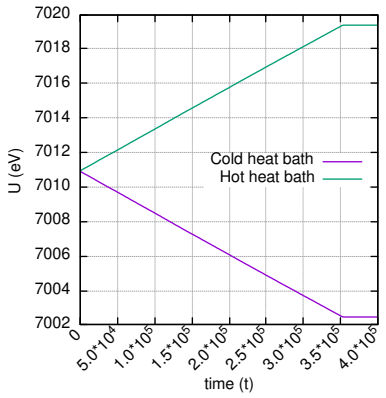
6.2.2.3 Graphs for various τ values



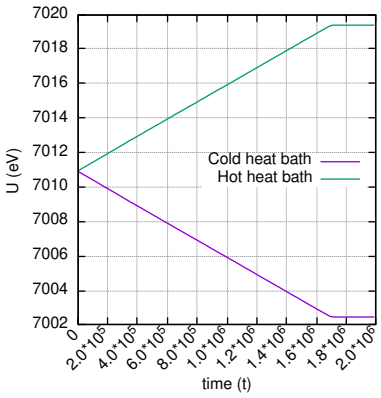
(a) $\tau = 0.1$



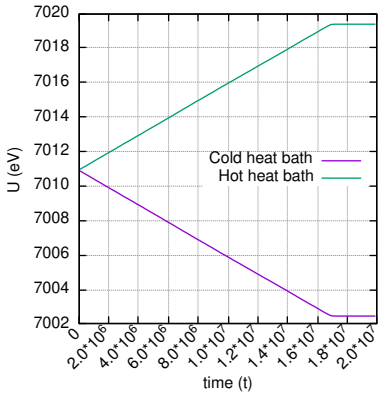
(b) $\tau = 0.5$



(c) $\tau = 1$

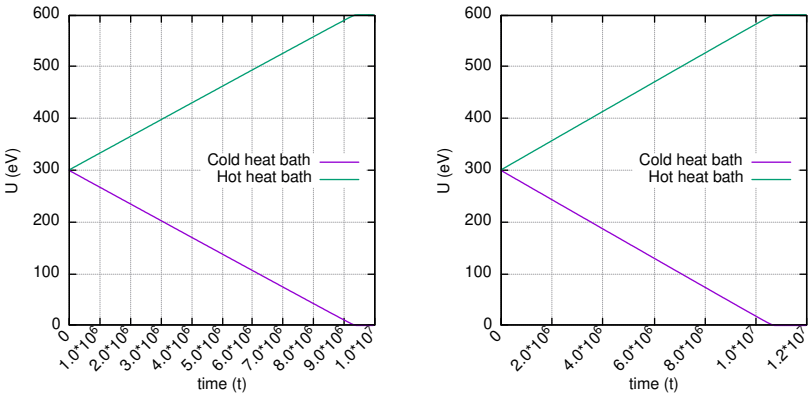


(d) $\tau = 10$



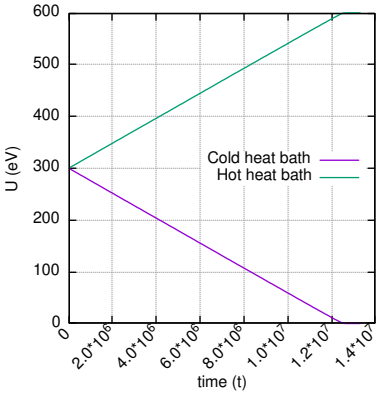
(e) $\tau = 100$

Figure 6.22: U vs. t graphs of fermionic heat baths for various τ values

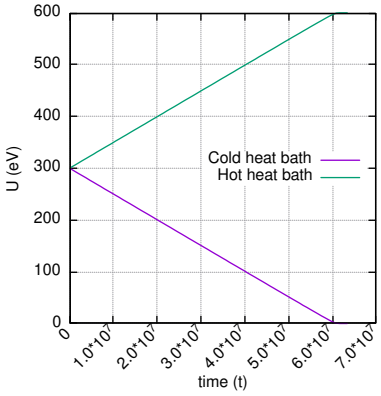


(a) $\tau = 0.1$

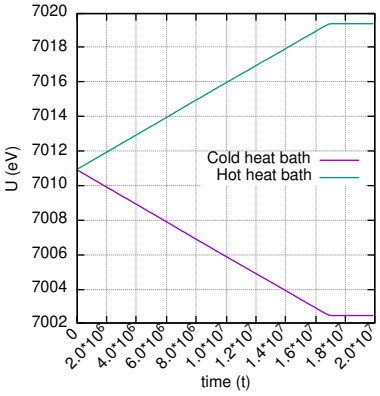
(b) $\tau = 0.5$



(c) $\tau = 1$



(d) $\tau = 10$



(e) $\tau = 100$

Figure 6.23: U vs. t graphs of bosonic heat baths for various τ values

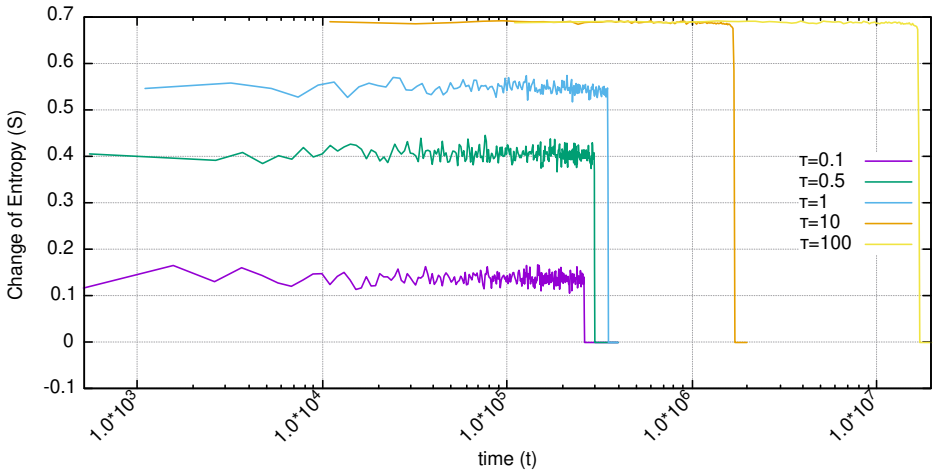


Figure 6.24: ΔS_T graphs of simulations with fermionic heat baths for varying values of τ

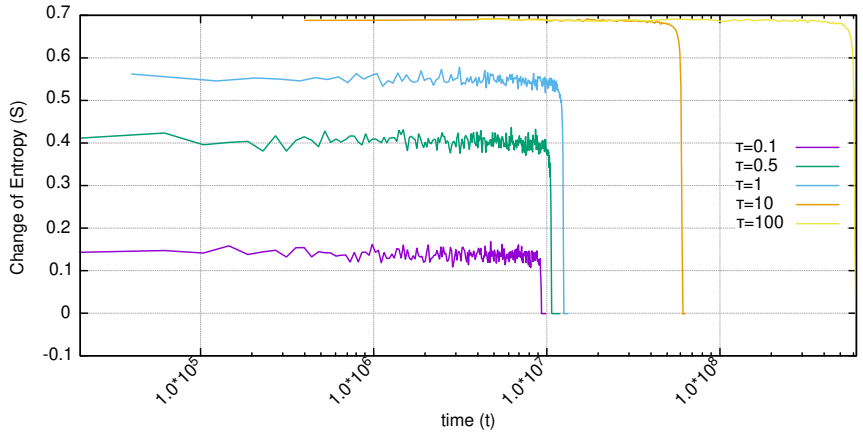


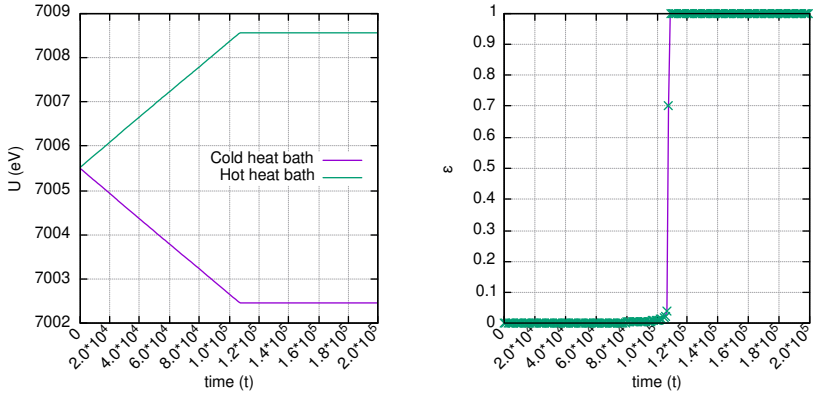
Figure 6.25: ΔS_T graphs of simulations with bosonic heat baths for varying values of τ

The last parameter we inspect is ϵ, or the difference between the pair kT_h and kT_c. Mandal et al. defined ϵ to quantify the temperature difference between heat baths [1]; however, there is a contrast between the role of ϵ in our work and in theirs. Since Mandal et al. used two thermal reservoirs, the temperatures do not change under energy transfer, and ϵ stays constant during the course of transitions. Oppositely in our case, as transitions take place, ϵ varies. Thus, even by running one simulation, we scan through a range of different ϵ values.

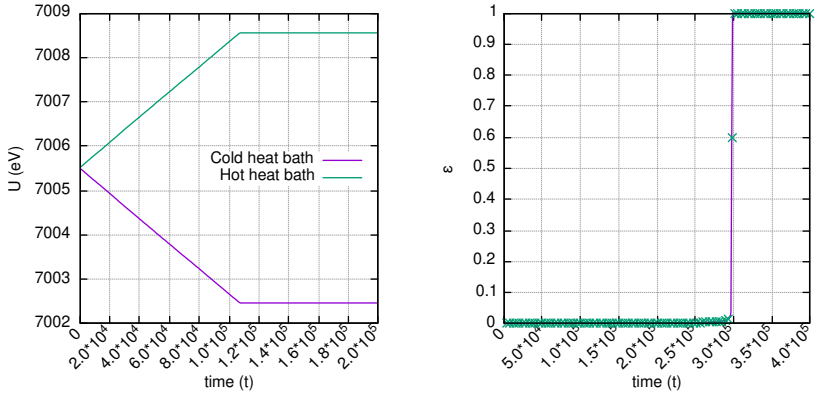
The only problem with checking a wide range of ϵ is caused by the quantization of kT, which we explained in section 5.1. Compared to the rise of ϵ, the decrease of $S(E_c)$, and consequently the decrease of kT_c, is so sudden that before we can observe some part of the range of ϵ, the heat transfer stops. We ran the simulation for different initial temperatures, yet the range of ϵ was always similar, as shown in figure 6.26. As the graphs of ϵ vs. t shows, our simulation mostly covers the region of ϵ near zero; and after a while it jumps to 1. Since the problem is caused by the quantization of kT, like in the investigation of δ, we mostly observe it in fermionic cases. Consequently, as we demonstrate in fig 6.27, ϵ scatters around a wider range in the simulations with bosonic heat baths.

We would like to point out that, at first, starting from the same temperatures seems to be the issue; however, even if we start from different temperatures, the issue remains. The reason of this is the deterministic nature of the system, which we demonstrated in figure 6.28. Suppose that we run two simulations in which all parameters are equal except initial temperatures; the first one starts from $kT_{c,init} = kT_{h,init}$, and the other one starts from $kT_{c,init} < kT_{h,init}$. Then, if the initial temperature values of the second simulation are on the temperature trajectories of the first simulation, those two simulations' outcome would be the same, other than the small fluctuations caused by Monte Carlo method. Moreover, since our system is deterministic, we can look from the other way around, any instance with two distinct initial temperatures can be extended into a case with equal initial temperatures. Hence assuming $kT_{c,init} = kT_{h,init}$ is not causing any issue.

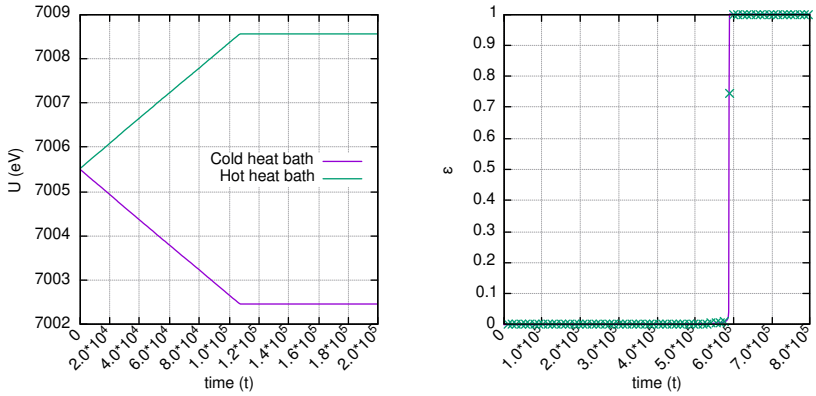
6.2.2.4 Graphs for various ϵ values



(a) $kT_{c,init} = kT_{h,init} = 0.015$

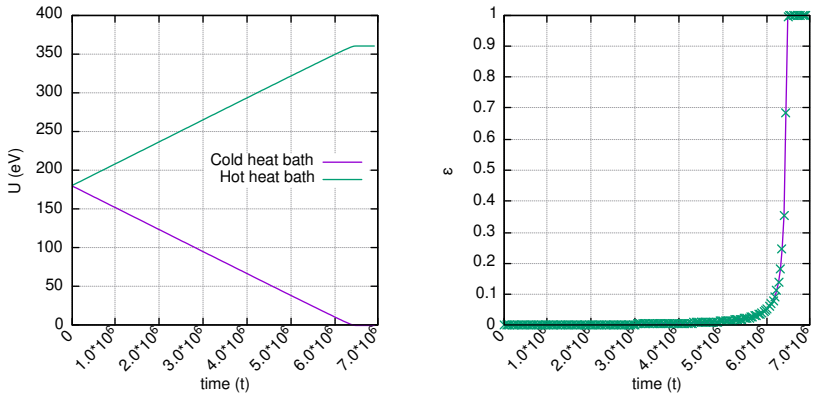


(b) $kT_{c,init} = kT_{h,init} = 0.025$

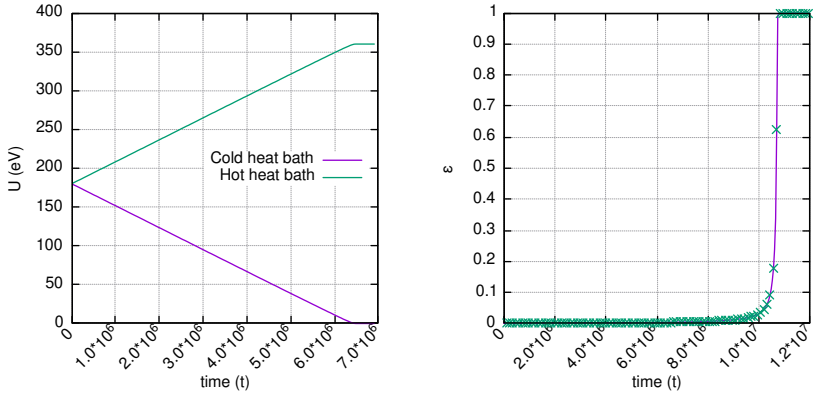


(c) $kT_{c,init} = kT_{h,init} = 0.035$

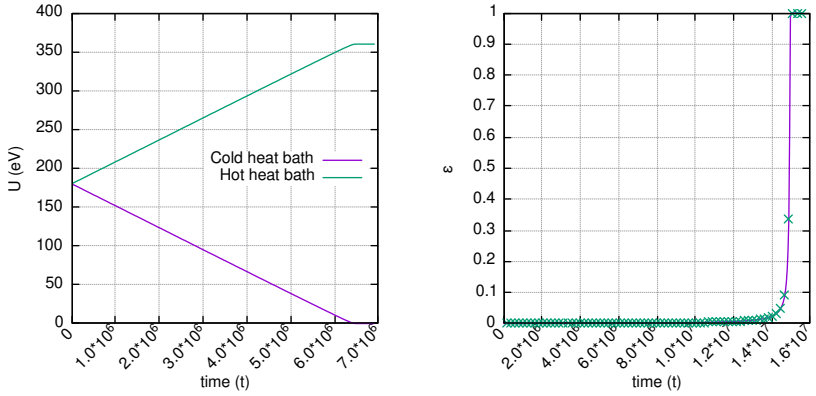
Figure 6.26: U vs. t & ϵ vs. t graphs of fermionic heat baths for various initial temperatures



(a) $kT_{c,init} = kT_{h,init} = 0.015$



(b) $kT_{c,init} = kT_{h,init} = 0.025$



(c) $kT_{c,init} = kT_{h,init} = 0.035$

Figure 6.27: U vs. t & ϵ vs. t graphs of bosonic heat baths for various initial temperatures

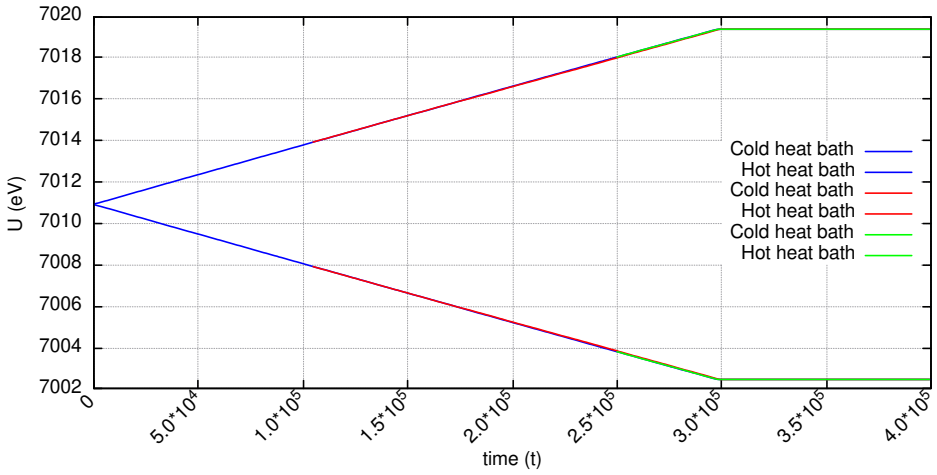


Figure 6.28: Progression of simulations with fermionic heat baths for various initial temperatures

6.3 Discussion

In this section, we have two main objectives: First, we will comment on the effects of particle type, discuss the effects of the parameters thoroughly, and try to find the optimal set of parameters that results with the most efficient refrigeration. Next, we will point out the reasons of the issues that we faced while presenting our results.

Before commenting on the effects of parameters, we would like to point out a fundamental observation regarding the particle type. Comparing our results, we recognize that, when all other parameters are identical, systems with fermionic heat baths reaches to the low temperatures much faster than the bosonic heat baths. The main reason for this is the immense difference between the amount of energy needed to be extracted from the cold heat baths to reach low temperatures, which we can see from the tables 6.1, 6.2, 6.3, and 6.4. The discrepancy is more visible if we consider t_{eq} as well. For instance, when both of the systems are at control parameters, the ratio between ΔU_c values is

$$\frac{\Delta U_{c,Bose}}{\Delta U_{c,Fermi}} \approx 35.490134,$$

and the ratio between t_{eq} values is

$$\frac{t_{eq,Bose}}{t_{eq,Fermi}} \approx 39.993474.$$

Hence, the ratio of energy extracted per time is approximately 0.887398.

All of the data that we gathered from our simulations have similar results. Hence we conclude that selecting Fermi gas is superior, in terms of the speed of refrigeration. However, we would like to mention another important point, the feasibility of the heat baths: In this study, we compare the gases that have same amount of particles in the same volume. We assume that both of these systems are producible, which is not necessarily true with the current technology. Due to this reason, this comparison is only true theoretically, and the real world application might require further analysis.

Next, we compare the effects of the heat bath parameters. As we mentioned earlier, increasing N extends the time it takes to reach equilibrium in both particle types; whereas increasing V affects only fermionic systems' equilibrium time, and has no observable effects on the bosonic case. Note that, in all cases that experience a deceleration, the reason is the increase of the total internal energy of the gases. Hence, like in type of particle selection, the choice of N and V depends on the practicability. However, an important point to remember while selecting these parameters is to satisfy $\sqrt[3]{V/N} \leq \lambda$ to observe quantum effects.

Thereafter, we comment on the influence of the model parameters in the same order as section 6.2.2. The first parameter, δ is related to the states of the incoming bits. As we mentioned earlier, for refrigeration to happen δ must staisfy $\delta > \epsilon$. Moreover, a higher value of δ increases the chance of having the transition $0d \rightarrow 1u$, which speeds up the refrigeration process, as we presented in figures 6.14 and 6.15. Of course speeding up has its own cost, which is the higher rate of production of information entropy. In both fermionic and bosonic case, for $\delta = 1$, the average difference between incoming and outgoing bits is $\delta - \delta' \approx 0.3$. Note that as δ decreases, the difference decreases as well; for $\delta \approx 0.8$ the difference is $\delta - \delta' < 0.2$, and for $\delta \approx 0.2$ it is $\delta - \delta' < 0.2$. However, other than quickness, a higher value of δ brings sustainability as well. By circling the band, the outgoing bits can be used as incoming bits in a second turn, and keeps the refrigerator working for a longer time. Due to these reasons, $\delta = 1$ is the most favorable

value.

The second model parameter γ accelerates the energy extraction, until it reaches to a limit. It also increases the production of information entropy per time. However, we observed that the total entropy production decreases when we increase γ. From the graphs that we gave in figures 6.20 and 6.21, we saw that the total area under the curves tend to decrease as we increased γ. For instance, in fermionic systems, the total entropy production per time and t_{eq} are approximately 0.402624 and 298749.99 for $\gamma = 1$; and 0.521752 and 191649.99 for $\gamma = 100$. From these values, we found that total entropy production is 120283.915974 for the first case and 99993.765582 for the second one. Hence, as long as it is feasible, increasing γ until energy extraction reaches its limit favours refrigeration.

By contrast, increasing τ decelerates energy extraction while accelerating total entropy production. However, as we mentioned earlier, the rise of entropy production is due to slowdown of the overall system. Since each bit spends much more time interacting with the demon, the overall information entropy spreads across the time. Nevertheless, we did not notice any benefit of having a larger τ value; hence keeping it as low as possible is advantageous.

Lastly, we examine ϵ. Since thermal reservoirs have constant temperatures, Mandal et al. were able to choose the value of ϵ as an independent parameter. However, this is not true in our case. Since ϵ varies as the demon carries energy from one heat bath to the other, in our case it depends on the dynamics of the system. As a consequence, in our work, it is not useful to compare its effect on the system. The only remark that we can present is the lower ϵ is the better for refrigeration. By reminding the definition of ϵ as

$$\epsilon = \frac{\omega - \sigma}{1 - \omega\sigma},$$

where

$$\sigma = \tanh\Big(\frac{S(E_h) - S(E_h - \Delta E)}{2k}\Big), \quad \omega = \tanh\Big(\frac{S(E_c) - S(E_c - \Delta E)}{2k}\Big),$$

we see that there is only one to keep it low: By minimizing the values of σ and ω we can keep ϵ small. The way of doing that is selecting a low ΔE. Even though this will slow down refrigeration process, it will ensure that the demon acts as a

refrigerator for a longer period by keeping the inequality $\delta > \epsilon$ satisfied.

Now, we will return to the issues that we faced in section 6.2. Firstly, we comment on the extreme slowdown of energy extraction while using bosonic heat baths. If the model parameters allow the cold heat baths to reach internal energy near zero, energy extraction becomes extremely slow. We noticed that both of the heat bath parameters changes the point where this stagnation starts. A closer inspection revealed that this phenomenon occurs when z becomes sufficiently large, which is when Bose Einstein Condensation becomes observable. Again, the starting point of the stagnation changes with N and V selections. Moreover, determining the exact value is not possible with the resolution that our data have. However, the general trend indicates that as particle density increases, starting point of slowdown gets closer to $z \approx 1$. This result is accurate, since increasing particle density helps us to observe quantum effects.

The second issue that we would like to address is the linearity of the internal energy graphs. For high temperatures, linear graphs are agreeable; but as the internal energy of cold heat baths decrease, we expected them to curve, since extracting energy from a low energy system is harder. However, we could not be able to observe this phenomenon. The reason of this is the sudden jump on the rates of cooperative transitions.

In our modified model, the rates of cooperative transitions were defined as

$$\frac{R_{d0\to u1}}{R_{u1\to d0}} = e^{-\left(\frac{S(E_c)-S(E_c-\Delta E)}{k}\right)},$$

where the transition rates were parameterized in terms of

$$\omega = \tanh\left(\frac{S(E_c)-S(E_c-\Delta E)}{2k}\right).$$

We can substitute $x = [S(E_c) - S(E_c - \Delta E)]/2k$, and rewrite ω as $\tanh(x)$, which we give in the following figure.

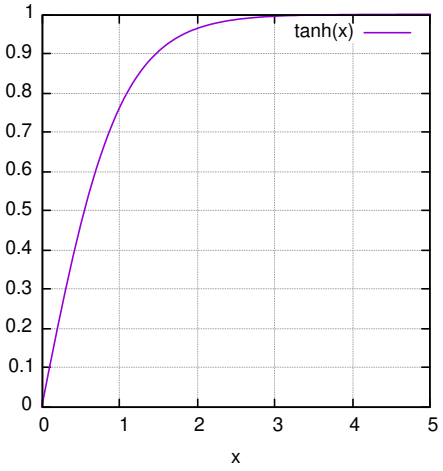


Figure 6.29: Graph of $\tanh(x)$

Note that $\tanh(x)$ converges to 1 rapidly as x gets larger. From graphs 6.3, we see that as temperature decreases, the graphs of internal entropies become steeper. Hence, as cold heat bath gets colder, x grows rapidly, which makes ω to jump to 1. The problem was worse when we first worked with the original transition rates; and as a solution, we proposed the idea in section 5.2. Even though it fixed that problem upto some degree, at low temperatures transition rates proceed to jump. Therefore, a further study is necessary regarding the low temperature region.

Chapter 7

Conclusion

The objective of this thesis is to investigate the functioning of a nano-refrigerator which operates solely by producing information entropy with random processes; and if it works, to determine which type of particles are better candidates for being the finite heat baths. To this end, we used the work of Mandal et al. [1] as our basis; but replaced their thermal reservoirs with finite thermal systems, which we modeled as finite Fermi and Bose gases. Our assumption was, these finite heat baths are small enough to change temperature under energy alteration, but also large enough to have a high heat capacity compared to the order of energy added or extracted. Then, we proposed a modification to the transition rates in the original work, to be able to use the model with finite systems.

In order to replicate the refrigerator, we performed a series of Monte Carlo simulations, and analyzed our results both quantitatively and qualitatively. The outcomes showed us that, such a nano-refrigerator can perform in real world. Moreover, our results demonstrated that, using Fermi particles are more favourable than using Bose particles. In addition, we also determined the advantageous conditions for the refrigerator to operate, in terms of parameters.

While we ran our analysis, we faced a number of problems, most of which occurred in the low temperature region. Even though Fermi gases are better candidates to be the finite quantum heat baths, alongside the simulations, they induced most of the issues. First and foremost, only fermionic systems needed a

manual intervention in the low temperature region to stop them moving below zero temperature, which we explained in section 5.1. This behaviour was caused by the nonzero transition rate around zero temperature. On the contrary, bosonic systems demonstrated a more natural decrease in terms of transition rates, and did not require an intervention.

Another problem that we faced during this study was the linearity of internal energy graphs. Our expectation was to observe curving graphs for internal energy; since, as the internal energy of cold heat baths decrease, extracting energy becomes less likely due to the decrease of the responsible transition rates. However, the transition rates did not decrease fast enough to fulfill our expectations. Instead, they stayed on a finite value, and jumped to zero when energy extraction becomes impossible. Note that this behaviour is a reflection of the previous issue that we explained earlier. However, this problem emerged in bosonic systems as well. Because these issues appeared in low temperature regions, a further study should focus on the behaviour of the refrigerator at low temperatures, and try to overcome the complications concerning the relevant transition rates.

Bibliography

[1] D. Mandal, H. T. Quan, and C. Jarzynski, "Maxwell's refrigerator: An exactly solvable model," *Phys. Rev. Lett.*, vol. 111, p. 030602, Jul 2013.

[2] L. Szilard, "Über die Entropieverminderung in einem thermodynamischen System bei Eingriffen intelligenter Wesen," *Zeitschrift für Physik*, vol. 53, pp. 840–856, 1929.

[3] R. Landauer, "Irreversibility and heat generation in the computing process," *IBM Journal of Research and Development*, vol. 5, no. 3, pp. 183–191, 1961.

[4] H. S. Leff and A. F. Rex, *Maxwell's Demon 2 - Entropy, Classical and Quantum Information, Computing*. Institute of Physics Publishing, second ed., 2003.

[5] P. Strasberg, G. Schaller, T. Brandes, and M. Esposito, "Thermodynamics of a physical model implementing a Maxwell demon," *Phys. Rev. Lett.*, vol. 110, p. 040601, Jan 2013.

[6] A. B. Boyd and J. P. Crutchfield, "Maxwell demon dynamics: Deterministic chaos, the Szilard map, and the intelligence of thermodynamic systems," *Phys. Rev. Lett.*, vol. 116, p. 190601, May 2016.

[7] R. E. Spinney, M. Prokopenko, and D. Chu, "Information ratchets exploiting spatially structured information reservoirs," *Phys. Rev. E*, vol. 98, p. 022124, Aug 2018.

[8] S. W. Kim, T. Sagawa, S. De Liberato, and M. Ueda, "Quantum Szilard engine," *Phys. Rev. Lett.*, vol. 106, p. 070401, Feb 2011.

[9] C. Elouard, D. Herrera-Martí, B. Huard, and A. Auffèves, "Extracting work from quantum measurement in Maxwell's demon engines," *Phys. Rev. Lett.*, vol. 118, p. 260603, Jun 2017.

[10] L. Buffoni, A. Solfanelli, P. Verrucchi, A. Cuccoli, and M. Campisi, "Quantum measurement cooling," *Phys. Rev. Lett.*, vol. 122, p. 070603, Feb 2019.

[11] Y.-H. Shi, H.-L. Shi, X.-H. Wang, M.-L. Hu, S.-Y. Liu, W.-L. Yang, and H. Fan, "Quantum coherence in a quantum heat engine," *Journal of Physics A: Mathematical and Theoretical*, vol. 53, p. 085301, Feb 2020.

[12] V. Serreli, C.-F. Lee, E. R. Kay, and D. A. Leigh, "A molecular information ratchet," *Nature*, vol. 445, p. 523–527, Feb 2007.

[13] M. Gavrilov and J. Bechhoefer, "Erasure without work in an asymmetric double-well potential," *Phys. Rev. Lett.*, vol. 117, p. 200601, Nov 2016.

[14] M. Ribezzi-Crivellari and F. Ritort, "Large work extraction and the Landauer limit in a continuous Maxwell demon," *Nature Physics*, vol. 15, p. 660–664, Jul 2019.

[15] N. Cottet, S. Jezouin, L. Bretheau, P. Campagne-Ibarcq, Q. Ficheux, J. Anders, A. Auffèves, R. Azouit, P. Rouchon, and B. Huard, "Observing a quantum Maxwell demon at work," *Proceedings of the National Academy of Sciences*, vol. 114, no. 29, pp. 7561–7564, 2017.

[16] L. L. Yan, T. P. Xiong, K. Rehan, F. Zhou, D. F. Liang, L. Chen, J. Q. Zhang, W. L. Yang, Z. H. Ma, and M. Feng, "Single-atom demonstration of the quantum Landauer principle," *Phys. Rev. Lett.*, vol. 120, p. 210601, May 2018.

[17] F. Mandl, *Statistical Physics*, ch. 1. John Wiley & Sons, 1988.

[18] *The London, Edinburgh and Dublin philosophical magazine and journal of science.*, vol. ser.4:v.12 (1856:July-Dec.). London :Taylor & Francis, 1856.

[19] G. W. Drake, "Entropy." `https://www.britannica.com/science/entropy-physics`, June 2018. Access Date: May 12, 2020.

[20] R. Clausius, *The Mechanical Theory of Heat.* Macmillan and Co., 1879. Translated by Walter R. Browne.

[21] C. E. Shannon, "A mathematical theory of communication," *The Bell System Technical Journal*, vol. 27, no. 3, pp. 379–423, 1948.

[22] J. C. Maxwell, *Theory of Heat.* Text-books of science, Longmans, Green, and Co., 1871.

[23] J. C. Maxwell, *Theory of Heat*, ch. XXII, p. 338. Text-books of science, Longmans, Green, and Co., 1902. with corrections and additions (1891) by Lord Rayleigh.

[24] J. C. Maxwell, *Theory of Heat*, ch. XXII, pp. 338–339. Text-books of science, Longmans, Green, and Co., 1902. with corrections and additions (1891) by Lord Rayleigh.

[25] H. S. Leff and A. F. Rex, *Maxwell's Demon 2 - Entropy, Classical and Quantum Information, Computing*, ch. 1, p. 4. Institute of Physics Publishing, second ed., 2003.

[26] H. S. Leff and A. F. Rex, *Maxwell's Demon 2 - Entropy, Classical and Quantum Information, Computing*, ch. 1, p. 5. Institute of Physics Publishing, second ed., 2003.

[27] H. S. Leff and A. F. Rex, *Maxwell's Demon 2 - Entropy, Classical and Quantum Information, Computing*, ch. 1, p. 6. Institute of Physics Publishing, second ed., 2003.

[28] B. Cowan, "On the chemical potential of ideal Fermi and Bose gases," *Journal of Low Temperature Physics*, 09 2019.

[29] K. Huang, *Statistical Mechanics*, ch. 12.3, p. 288. John Wiley & Sons, 1987.

[30] K. Huang, *Statistical Mechanics*, ch. 6.2, p. 131. John Wiley & Sons, 1987.

[31] K. Huang, *Statistical Mechanics*, ch. 8.6. John Wiley & Sons, 1987.

[32] R. Kubo, *Statistical Mechanics - An Advanced Course with Problems and Solutions*, ch. 4, p. 241. North-Holland Physics Publishing, 1988.

Appendix A

Fermi energy on finite gases

In a three-dimensional infinite square well, energy of a single particle is given by

$$E_{k_x,k_y,k_z} = \frac{\hbar^2}{2m}(k_x^2 + k_y^2 + k_z^2), \tag{A.1}$$

where, for $i \in x, y, z$, $k_i = 0, \pm\frac{2\pi}{L}, \ldots, \pm\frac{n_i 2\pi}{L}$ are the wavenumbers for the quantum numbers $n_i \in \mathbb{Z}$, and L is the length of one side of the well. Note that the energy levels are degenerate, e.g. $E_{k,0,0} = E_{0,k,0} = E_{0,0,k}$.

Now, assume that we have N non-interacting fermions in the well, where N and V are large enough for us to treat k_x, k_y, k_z as continuous variables, i.e. $\vec{k} = (k_x, k_y, k_z)$. Then, the number of particles can be represented as

$$N = \frac{4}{3}\pi k_F^3 \times \frac{1}{(2\pi/L)^3} \times g,$$

where $\frac{4}{3}\pi k_F^3$ is the volume of the *Fermi Sphere* for $|\vec{k_F}|$ being its radius, $(2\pi/L)^3$ is the volume of each allowed state in reciprocal space, and the factor g comes from the degeneracy. For spin-$\frac{1}{2}$ particles $g = 2$, then we have

$$N = \frac{V k_F^3}{3\pi^2} \rightarrow k_F^3 = 3\pi^2 \frac{N}{V}.$$

Then Fermi energy, energy corresponding to the points $|\vec{k_F}|$ at $T = 0$, is given by

$$\epsilon_F = \frac{\hbar^2}{2m} k_F^2 = \frac{\hbar^2}{2m}\left(3\pi^2 \frac{N}{V}\right)^{2/3}. \tag{A.2}$$

By the same reasoning, for many particles we have k_{x_F}, k_{y_F}, k_{z_F} such that the equation (A.1) gives the energy of the highest occupied state at $T = 0$, i.e. the Fermi energy as

$$E_{k_{x_F},k_{y_F},k_{z_F}} = \frac{\hbar^2}{2m}(k_{x_F}^2 + k_{y_F}^2 + k_{z_F}^2). \tag{A.3}$$

Since we work on finite systems, we need to use (A.3). Yet to do that on computer, we need to count each filled state one by one, which is time-consuming. Moreover for large N, the difference between equations (A.3) and (A.2) is negligible, which we confirmed by finding the number of particles in the Fermi sphere for both cases. Hence we used (A.2) while finding Fermi energy.

Appendix B

Fermi-Dirac Functions

The most general form of Fermi-Dirac functions is

$$f_n(z) \equiv \frac{1}{\Gamma(n)} \int_0^\infty \frac{dx\ x^{n-1}}{z^{-1}e^x + 1}, \quad 0 \leq z < \infty$$

where $\Gamma(n)$ is *gamma function*, and n is an index which is most generally a complex number. However, for the physical applications only integer and half integer values of n will be used.

In the interval $0 \leq z \leq 1$, the series expansion of the Fermi-Dirac functions is

$$f_n(z) = \sum_{l=1}^{\infty} (-1)^{l-1} \frac{z^l}{l^n}$$

whereas for $z \gg 1$, the asymptotic expansion of them is;

$$f_n(z) = \frac{(\ln z)^n}{\Gamma(n+1)} \left[1 + \sum_{k=2,4,\ldots} 2n(n-1)\ldots(n-k+1) \left(1 - \frac{1}{2^{k-1}} \right) \frac{\zeta(k)}{(\ln z)^k} \right]$$

where $\zeta(k)$ is *Riemann zeta function.*

For $n \geq 1$, the Fermi-Dirac functions have the recurrence relation

$$z \frac{\partial f_n(z)}{\partial z} = f_{n-1}(z).$$

For the integer and half integer n values that we are interested in, the functions $f_n(z)$ are positive, monotonically increasing, unbounded functions.

In the study of finite heat baths, for both Fermi and Boson, only the functions that correspond to $n = 3/2$ and $n = 5/2$ are used explicitly, which are

$$f_{3/2}(z) = \frac{1}{\Gamma(3/2)} \int_0^\infty \frac{dx\ x^{1/2}}{z^{-1}e^x + 1},$$
$$f_{5/2}(z) = \frac{1}{\Gamma(5/2)} \int_0^\infty \frac{dx\ x^{3/2}}{z^{-1}e^x + 1}.$$

The functions above come from the most general definition, include a square root in the numerator. While performing numerical calculation, use of square root slows down the process abundantly. To get rid of the square roots, we will write the functions in an alternative form, which is also used in the literature [31].

For $f_{3/2}(z)$, a change of variables $x \to y^2$, $dx \to 2ydy$ is sufficient to get the desired form

$$f_{3/2}(z) = \frac{1}{\Gamma(3/2)} \int_0^\infty \frac{dy\ 2y^2}{z^{-1}e^{y^2} + 1}.$$

Note that for half integers, gamma function can be written as

$$\Gamma\left(n' + \frac{1}{2}\right) = \frac{(2n')!}{4^{n'}n'!}\sqrt{\pi}$$

where $n' \in \mathbb{Z}^+$. After $y \to x$, we have

$$f_{3/2}(z) = \frac{4}{\sqrt{\pi}} \int_0^\infty \frac{dx\ x^2}{z^{-1}e^{x^2} + 1}.$$

For $f_{5/2}(z)$, the change of form is more indirect. First, by performing an integration by parts we have,

$$\begin{aligned} f_{5/2}(z) &= \frac{1}{\Gamma(5/2)} \int_0^\infty \frac{dx\ x^{3/2}}{z^{-1}e^x + 1} \\ &= -\frac{1}{\Gamma(5/2)} x^{3/2} \ln(1 + ze^{-x})\Big|_0^\infty + \frac{1}{\Gamma(5/2)} \int_0^\infty dx\ \frac{3\ln(1 + ze^{-x})x^{1/2}}{2}, \end{aligned}$$

in which the first term is zero. Then, like in previous case, the change of variables $x \to y^2$ gives

$$f_{5/2}(z) = \frac{3}{2\Gamma(5/2)} \int_0^\infty dy\ 2\ln(1 + ze^{-y^2})y^2.$$

Finally, by plugging in the value of $\Gamma(5/2)$ and changing the variable of the function back, we have

$$f_{5/2}(z) = \frac{4}{\sqrt{\pi}} \int_0^\infty dx\ x^2 \ln(1 + ze^{-x^2}).$$

In figure B.1, we show the graph of the functions $f_{3/2}(z)$, $f_{5/2}(z)$, and $f_{5/2}(z)/f_{3/2}(z)$, which we will use throughout our study.

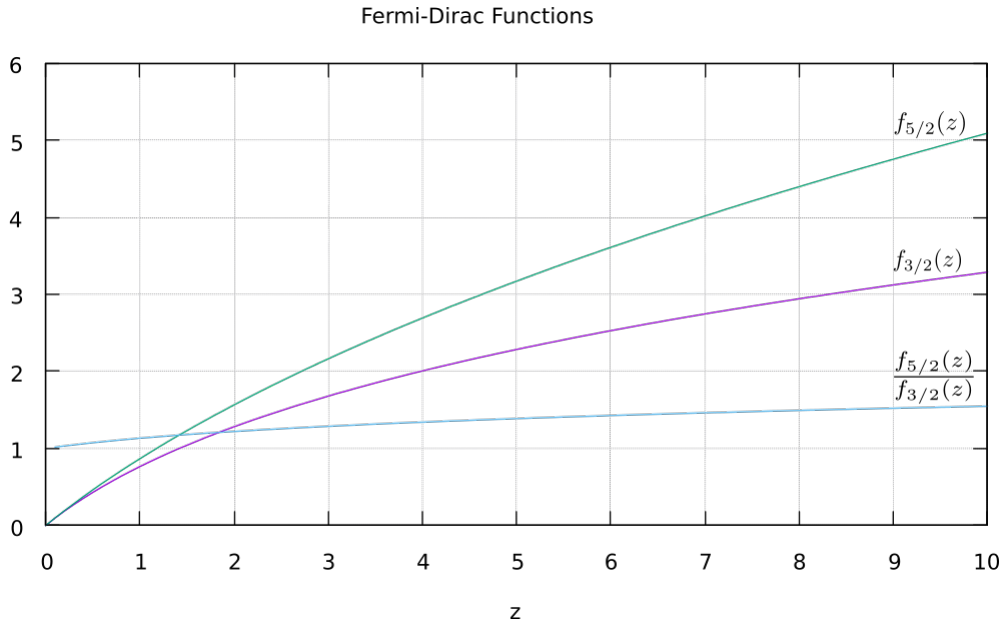


Figure B.1: Graph of $f_{3/2}(z)$, $f_{5/2}(z)$, and $f_{5/2}(z)/f_{3/2}(z)$

Appendix C

Bose-Einstein Functions

The most general form of the Bose-Einstein functions is

$$g_n(z) \equiv \frac{1}{\Gamma(n)} \int_0^\infty \frac{dx\ x^{n-1}}{z^{-1}e^x - 1}, \quad 0 \leq z \leq 1.$$

Since the interval of the functions is bounded, unlike Fermi-Dirac functions, we only have the series expansion

$$g_n(z) = \sum_{l=1}^{\infty} \frac{z^l}{l^n}$$

which is convergent in the interval $0 \leq z \leq 1$. Note that for $n \geq 1$, the functions $g_n(z)$ are positive, monotonically increasing functions, and for $n > 1$, they are bounded. When $z = 1$, the summation above becomes a *p-series*, and if $n > 1$ the Bose-Einstein functions can be written in terms of *Riemann zeta function*.

$$g_n(1) = \zeta(n) = \sum_{l=1}^{\infty} \frac{1}{l^n}.$$

For $n \geq 1$, we have the recurrence relation

$$z\frac{\partial g_n(z)}{\partial z} = g_{n-1}(z).$$

Like mentioned before, we are mostly interested in the cases $n = 3/2$ and $n = 5/2$, which are

$$g_{3/2}(z) = \frac{1}{\Gamma(3/2)} \int_0^\infty \frac{dx\ x^{1/2}}{z^{-1}e^x - 1}$$
$$g_{5/2}(z) = \frac{1}{\Gamma(5/2)} \int_0^\infty \frac{dx\ x^{3/2}}{z^{-1}e^x - 1}.$$

By carrying out the same procedures that we applied to Fermi-Dirac functions, we achieve the desired forms,

$$g_{3/2}(z) = \frac{4}{\sqrt{\pi}} \int_0^\infty dx\ \frac{x^2}{z^{-1}e^{x^2} - 1}$$
$$g_{5/2}(z) = -\frac{4}{\sqrt{\pi}} \int_0^\infty dx\ x^2 \ln(1 - ze^{-x^2}).$$

Figure C.1 shows the graphs of $g_{3/2}(z)$, $g_{5/2}(z)$, and $g_{5/2}(z)/g_{3/2}(z)$. There is an important point that we need to mention regarding the behaviour of the function $g_{3/2}(z)$ before concluding this chapter. As it can be seen from the figure C.1, $g_{5/2}(z)$ starts from the origin and ends in a finite value without any issues. $g_{3/2}(z)$, on the other hand, does not move as smooth as $g_{5/2}(z)$: At $z = 1$, it is equal to $\zeta(3/2)$ which is approximately 2.612375. However, at that point its derivative diverges. This can be seen from the recursion relation given above,

$$g'_{3/2}(z) = z g_{1/2}(z) \xRightarrow{z=1} g'_{3/2}(1) = g_{1/2}(1) = \zeta(1/2),$$

which gives a divergent p-series.

This divergence is not an issue in theoretical work, since for finite heat baths we did not use the full range of the function, as it will be explained in section 4.2. However, in computational work divergent derivative is problematic, both while calculating the function at $z \approx 1$, and while numerically inverting the function. Thus, while doing the numerical work around $z = 1$, we reduced our precision and performed some calculations by hand, when it was necessary.

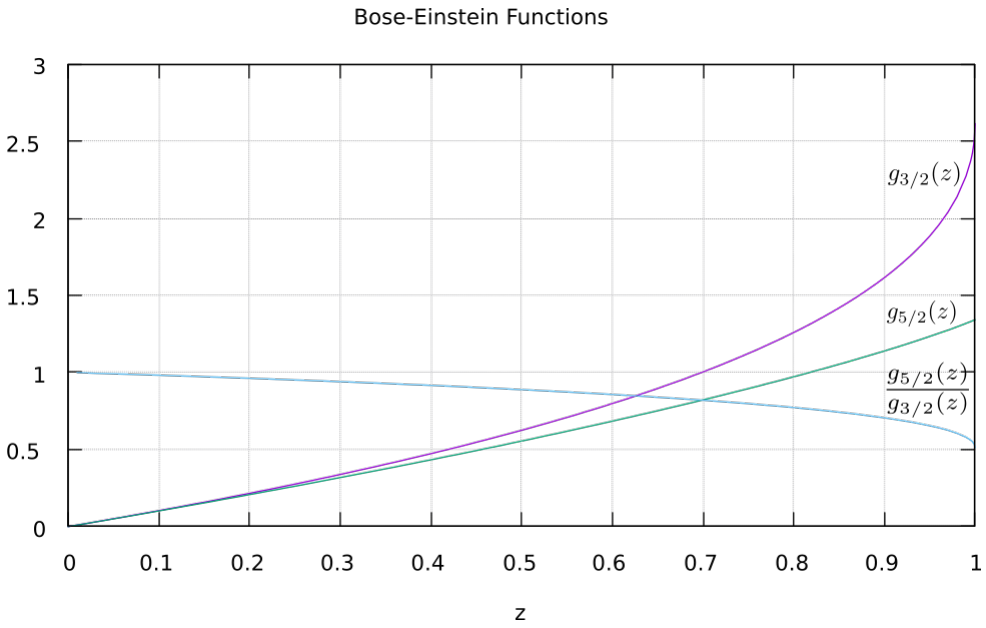


Figure C.1: Graph of $g_{3/2}(z)$, $g_{5/2}(z)$, and $g_{5/2}(z)/g_{3/2}(z)$

Appendix D

Numerical Limitations

In the low temperature limit, the asymptotic expansion of $f_{3/2}(z)$ is

$$f_{3/2}(z) \propto (\ln z)^{3/2},$$

which can be rearranged to find fugacity as

$$z \propto e^{\epsilon_F/kT}.$$

This shows that as $kT \to 0$, $z \to \infty$ exponentially. Due to this growth rate, after a certain point the numerical inversion algorithm used on the equation (4.5) to find z does not converge. We observed this point to be around $z \approx 3.6 \times 10^{33}$, or $\epsilon_F/kT \approx 77.2662$. Moreover, from equation (4.5), we have

$$f_{3/2}(z) = \frac{\lambda^3 N}{2V} = \frac{N}{2V}\left(\frac{2\pi\hbar^2}{mkT}\right)^{3/2}.$$

Note that

$$\epsilon_F \frac{\hbar^2}{2m}\left(\frac{3\pi^2 N}{V}\right)^{2/3} \Rightarrow \frac{\hbar^2}{m} = 2\epsilon_F\left(\frac{V}{3\pi^2 N}\right)^{2/3}.$$

Hence, we have

$$f_{3/2}(z) = \frac{N}{2V}\left[2\epsilon_F \frac{2\pi}{kT}\left(\frac{V}{3\pi^2 N}\right)^{2/3}\right]^{3/2} = \frac{1}{6\pi^2}\left(4\pi\frac{\epsilon_F}{kT}\right)^{3/2},$$

which only depends on ϵ/kT. For the values that we gave above, we found that the numerical inversion starts to fail around $f_{3/2}(z) \approx 511$. Due to this problem,

in low temperature limit we used equation (4.7) to correspond kT to U. In the remaining part of this chapter, we will show the equivalence of the equations (4.6) and (4.7) for that limit.

In low temperatures, the chemical potential of the fermi gas is given by [32]

$$\mu = \epsilon_F \left[1 - \frac{\pi^2}{12} \left(\frac{kT}{\epsilon_F} \right)^2 - \frac{\pi^4}{80} \left(\frac{kT}{\epsilon_F} \right)^4 + \mathcal{O}((kT)^6) \right].$$

Moreover, in the large fugacity limit the Fermi-Dirac functions are expanded as

$$f_n(z) = \frac{(\ln z)^n}{\Gamma(n+1)} \left[1 + n(n-1)\frac{\pi^2}{6}(\ln z)^{-2} + n(n-1)(n-2)(n-3)\frac{7\pi^4}{360}(\ln z)^{-4} + \mathcal{O}((\ln z)^{-6}) \right].$$

Then,

$$\begin{aligned} \frac{f_{5/2}(z)}{f_{3/2}(z)} &= \frac{\frac{(\ln z)^{5/2}}{\Gamma(7/2)}}{\frac{(\ln z)^{3/2}}{\Gamma(5/2)}} \frac{1 + \frac{5\pi^2}{8}(\ln z)^{-2} - \frac{7\pi^4}{384}(\ln z)^{-4}}{1 + \frac{\pi^2}{8}(\ln z)^{-2} + \frac{7\pi^4}{640}(\ln z)^{-4}} \\ &= \frac{2}{5} \ln z \frac{1 + \frac{5\pi^2}{8}(\ln z)^{-2} - \frac{7\pi^4}{384}(\ln z)^{-4}}{1 + \frac{\pi^2}{8}(\ln z)^{-2} + \frac{7\pi^4}{640}(\ln z)^{-4}}. \end{aligned}$$

Since $z = e^{\mu/kT}$,

$$\frac{f_{5/2}(z)}{f_{3/2}(z)} = \frac{2}{5} \frac{\mu}{kT} \frac{1 + \frac{5\pi^2}{8}(\frac{kT}{\mu})^2 - \frac{7\pi^4}{384}(\frac{kT}{\mu})^4}{1 + \frac{\pi^2}{8}(\frac{kT}{\mu})^2 + \frac{7\pi^4}{640}(\frac{kT}{\mu})^4},$$

and if we expand the right hand side for low kT, we have

$$\frac{f_{5/2}(z)}{f_{3/2}(z)} = \frac{2}{5} \frac{\mu}{kT} \left[1 + \frac{\pi^2}{2} \left(\frac{kT}{\mu} \right)^2 - \frac{11\pi^4}{120} \left(\frac{kT}{\mu} \right)^4 + \mathcal{O}((kT)^6) \right].$$

Plugging this to equation (4.6), we find

$$\begin{aligned} U &= \frac{3}{2} NkT \frac{2}{5} \frac{\mu}{kT} \left[1 + \frac{\pi^2}{2} \left(\frac{kT}{\mu} \right)^2 - \frac{11\pi^4}{120} \left(\frac{kT}{\mu} \right)^4 \right] \\ &= \frac{3}{5} N \left[\mu + \frac{\pi^2}{2} \frac{(kT)^2}{\mu} - \frac{11\pi^4}{120} \frac{(kT)^4}{(\mu)^3} \right]. \end{aligned}$$

Finally, if we use the expression that we found for chemical potential, we have

$$
\begin{aligned}
U = \frac{3}{5}N\Bigg\{\epsilon_F\Bigg[1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4\Bigg] \\
+\frac{\pi^2}{2}\frac{(kT)^2}{\epsilon_F\Big[1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4\Big]} \\
-\frac{11\pi^4}{120}\frac{(kT)^4}{\left(\epsilon_F\Big[1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4\Big]\right)^3}\Bigg\} \\
= \frac{3}{5}N\epsilon_F\Bigg\{1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4 \\
+\frac{\pi^2}{2}\frac{(kT/\epsilon_F)^2}{1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4} \\
-\frac{11\pi^4}{120}\frac{(kT/\epsilon_F)^4}{\Big[1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4\Big]^3}\Bigg\} \\
= \frac{3}{5}N\epsilon_F\Bigg\{1-\frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{80}\left(\frac{kT}{\epsilon_F}\right)^4+\frac{\pi^2}{2}\left(\frac{kT}{\epsilon_F}\right)^2+ \\
\frac{\pi^4}{24}\left(\frac{kT}{\epsilon_F}\right)^4-\frac{11\pi^4}{120}\left(\frac{kT}{\epsilon_F}\right)+\mathcal{O}((kT)^6)\Bigg\} \\
U = \frac{3}{5}N\epsilon_F\Bigg[1+\frac{5\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2-\frac{\pi^4}{16}\left(\frac{kT}{\epsilon_F}\right)^4\Bigg],
\end{aligned}
$$

which shows that equations (4.6) and (4.7) converges at low kT.

Appendix E

Heat Capacity of Bose Gas at High Temperatures

When $T > T_{crit}$, we can approximate the equations (4.11) and (4.12) as

$$\frac{N}{V} \cong \frac{1}{\lambda^3} g_{3/2}(z)$$
$$\frac{U}{N} \cong \frac{3}{2} \frac{V}{N\lambda^3} kT g_{5/2}(z),$$

which can be combined as

$$\frac{U}{Nk} = \frac{3}{2} T \frac{g_{5/2}(z)}{g_{3/2}(z)}.$$

From

$$C_v = \left(\frac{\partial U}{\partial T}\right)_v$$

we have

$$\begin{aligned}
\frac{C_v}{Nk} &= \frac{\partial}{\partial T}\left(\frac{3}{2} T \frac{g_{5/2}(z)}{g_{3/2}(z)}\right) \\
&= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} + \frac{3}{2} T \frac{\partial}{\partial T}\left(\frac{g_{5/2}(z)}{g_{3/2}(z)}\right) \\
&= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} + \frac{3}{2} T \frac{\partial z}{\partial T}\frac{\partial}{\partial z}\left(\frac{g_{5/2}(z)}{g_{3/2}(z)}\right).
\end{aligned}$$

Note that

$$\begin{aligned} g_{3/2}(z) = \frac{N\lambda^3}{V} \Rightarrow & \frac{\partial}{\partial T} g_{3/2}(z) = \frac{\partial}{\partial T}\frac{N\lambda^3}{V} \\ & \frac{\partial z}{\partial T}\frac{\partial}{\partial z} g_{3/2}(z) = -\frac{3}{2}\frac{N\lambda^3}{VT} \\ & \frac{\partial z}{\partial T}\frac{1}{z} g_{1/2}(z) = -\frac{3}{2}\frac{N\lambda^3}{VT} \\ & \frac{\partial z}{\partial T} = -\frac{3}{2}\frac{g_{3/2}(z)}{T}\frac{z}{g_{1/2}(z)}. \end{aligned}$$

Then

$$\begin{aligned} \frac{C_v}{Nk} &= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} + \frac{3}{2}T\bigg(-\frac{3}{2}\frac{g_{3/2}(z)}{T}\frac{z}{g_{1/2}(z)}\bigg)\frac{\partial}{\partial z}\bigg(\frac{g_{5/2}(z)}{g_{3/2}(z)}\bigg) \\ &= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} - \frac{9}{4}z\frac{g_{3/2}(z)}{g_{1/2}(z)}\frac{g'_{5/2}(z)g_{3/2}(z) - g_{5/2}(z)g'_{3/2}(z)}{(g_{3/2}(z))^2}. \end{aligned}$$

By using the recurrence relation $zg'_n(z) = g_{n-1}(z)$, we obtain

$$\begin{aligned} \frac{C_v}{Nk} &= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} - \frac{9}{4}\frac{1}{g_{1/2}(z)}\frac{g_{3/2}(z)g_{3/2}(z) - g_{5/2}(z)g_{1/2}(z)}{g_{3/2}(z)} \\ &= \frac{3}{2}\frac{g_{5/2}(z)}{g_{3/2}(z)} - \frac{9}{4}\frac{g_{3/2}(z)}{g_{1/2}(z)} + \frac{9}{4}\frac{g_{5/2}(z)}{g_{3/2}(z)} \\ \frac{C_v}{Nk} &= \frac{15}{4}\frac{g_{5/2}(z)}{g_{3/2}(z)} - \frac{9}{4}\frac{g_{3/2}(z)}{g_{1/2}(z)}. \end{aligned}$$

Note that as $T \gg T_{crit}$, the functions $g_{1/2}(z)$, $g_{3/2}(z)$, $g_{5/2}(z) \approx z$. Hence

$$\frac{C_v}{Nk} = \frac{15}{4} - \frac{9}{4} = \frac{3}{2}.$$

Appendix F

ΔU_C for Fermionic Heat Baths

We can rewrite the equation (6.1) as

$$\begin{aligned}\Delta U_c &= 3V\left(\frac{kT_i}{\lambda_i^3}f_{5/2}(z_i) - \frac{kT_f}{\lambda_f^3}f_{5/2}(z_f)\right)\\ &= 3V\left(\frac{m}{2\pi\hbar^2}\right)^{3/2}((kT_i)^{5/2}f_{5/2}(z_i) - (kT_f)^{5/2}f_{5/2}(z_f))\\ &= \frac{9\sqrt{\pi}}{8\epsilon_F^{3/2}}((kT_i)^{5/2}f_{5/2}(z_i) - (kT_f)^{5/2}f_{5/2}(z_f))\end{aligned}$$

Note that for both initial temperature that we give and the final temperature that the system reach, fugacity values are large enough to use the $z \gg 1$ expansion of $f_{5/2}(z)$

$$f_{5/2}(z) \approx \frac{8}{15\sqrt{\pi}}(\ln z)^{5/2} + \frac{\pi^{3/2}}{3}(\ln z)^{1/2}.$$

Then ΔU_c becomes

$$\begin{aligned}\Delta U_c = \frac{9N\sqrt{\pi}}{8\epsilon_F^{3/2}}\Bigg\{&(kT_i)^{5/2}\left[\frac{8}{15\sqrt{\pi}}(\ln z_i)^{5/2} + \frac{\pi^{3/2}}{3}(\ln z_i)^{1/2}\right]\\ &- (kT_f)^{5/2}\left[\frac{8}{15\sqrt{\pi}}(\ln z_f)^{5/2} + \frac{\pi^{3/2}}{3}(\ln z_f)^{1/2}\right]\Bigg\},\end{aligned}$$

and by substituting $z = e^{\mu/kT}$, we can rewrite equation above as

$$\begin{aligned}\Delta U_c &= \frac{9N\sqrt{\pi}}{8\epsilon_F^{3/2}}\left(\frac{8\mu_i^{5/2}}{15\sqrt{\pi}} - \frac{8\mu_f^{5/2}}{15\sqrt{\pi}} + \frac{\pi^{3/2}}{3}(kT_i)^2\sqrt{\mu_i} - \frac{\pi^{3/2}}{3}(kT_f)^2\sqrt{\mu_f}\right) \\ &= \frac{3}{5}\frac{N}{\epsilon_F^{3/2}}(\mu_i^{5/2} - \mu_f^{5/2}) + \frac{3\pi^2 N}{8\epsilon_F^{3/2}}[(kT_i)^2\sqrt{\mu_i} - (kT_f)^2\sqrt{\mu_f}].\end{aligned}$$

Note that for $\epsilon_F \gg kT$, chemical potential is given by

$$\mu \approx \epsilon_F\left[1 - \frac{\pi^2}{12}\left(\frac{kT}{\epsilon_F}\right)^2\right].$$

Inserting this to the equation above, we get

$$\begin{aligned}\Delta U_c = \frac{3}{5}\frac{N}{\epsilon_F^{3/2}}\Bigg\{&\epsilon_F^{5/2}\left[1 - \frac{\pi^2}{12}\left(\frac{kT_i}{\epsilon_F}\right)^2\right]^{5/2} - \epsilon_F^{5/2}\left[1 - \frac{\pi^2}{12}\left(\frac{kT_f}{\epsilon_F}\right)^2\right]^{5/2}\Bigg\} \\ &+\frac{3\pi^2 N}{8\epsilon_F^{3/2}}\Bigg\{(kT_i)^2\sqrt{\epsilon_F}\left[1 - \frac{\pi^2}{12}\left(\frac{kT_i}{\epsilon_F}\right)^2\right]^{1/2} \\ &\qquad - (kT_f)^2\sqrt{\epsilon_F}\left[1 - \frac{\pi^2}{12}\left(\frac{kT_f}{\epsilon_F}\right)^2\right]^{1/2}\Bigg\}.\end{aligned}$$

By expanding the powers 5/2 and 1/2, the equation yields to

$$\begin{aligned}\Delta U_c = \frac{3}{5}\frac{N}{\epsilon_F^{3/2}}\Bigg\{&\epsilon_F^{5/2}\left[1 - \frac{\pi^2}{12}\frac{5}{2}\left(\frac{kT_i}{\epsilon_F}\right)^2\right] - \epsilon_F^{5/2}\left[1 - \frac{\pi^2}{12}\frac{5}{2}\left(\frac{kT_f}{\epsilon_F}\right)^2\right]\Bigg\} \\ &+\frac{3\pi^2 N}{8\epsilon_F^{3/2}}\Bigg\{(kT_i)^2\sqrt{\epsilon_F}\left[1 - \frac{\pi^2}{12}\frac{1}{2}\left(\frac{kT_i}{\epsilon_F}\right)^2\right] \\ &\qquad - (kT_f)^2\sqrt{\epsilon_F}\left[1 - \frac{\pi^2}{12}\frac{1}{2}\left(\frac{kT_f}{\epsilon_F}\right)^2\right]\Bigg\}.\end{aligned}$$

Finally, rearranging the equation gives

$$\Delta U_c = \frac{\pi^2 N}{4\epsilon_F}[(kT_i)^2 - (kT_f)^2] - \frac{\pi^4 N}{64\epsilon_F^3}[(kT_i)^4 - (kT_f)^4],$$

which is equivalent to equation (6.2).

Appendix G

Codes

fermiheatbath_v10.c

```
#define _USE_MATH_DEFINES

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

static double conversion = 11604.51812;
static double alphamax = 505.;
static double debroglie = 0.692; // nm

double f1_2_int(double z) { // finds the function by using
    composite simpson's rule
  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
  int_old = -5.;
  int_new = -2.;
  n = 5;
```

```
  while (fabs(int_new - int_old) > 1e-10) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = z * (x_max * x_max) * exp(x_max * x_max) / ((z +
        exp(x_max * x_max)) * (z + exp(x_max * x_max)));
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * z * (x * x) * exp(x * x) / ((z + exp(x *
          x)) * (z + exp(x * x)));
      mult = 6 - mult;
    }
    int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
  }
  return int_new;
}

double f3_2_int(double z) { // finds the function by using
    composite simpson's rule
  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
  int_old = -5.;
  int_new = -2.;
  n = 5;

  while (fabs(int_new - int_old) > 1e-10) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = z * (x_max * x_max) / (z + exp(x_max * x_max));
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * z * (x * x) / (z + exp(x * x));
      mult = 6 - mult;
    }
    int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
```

```
  }
  return int_new;
}

double f5_2_int(double z) { // finds the function by using
    composite simpson's rule
  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
  int_old = -5.;
  int_new = -2.;
  n = 5;

  while (fabs(int_new - int_old) > 1e-10) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = (x_max * x_max) * log(1 + z * exp(-x_max * x_max));
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * (x * x) * log(1 + z * exp(-x * x));
      mult = 6 - mult;
    }
    int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
  }
  return int_new;
}

double F_int(double z) {
  double f52overf32;
  f52overf32 = f5_2_int(z) / f3_2_int(z);
  return f52overf32;
}

double F_int_2(double z) {
  double f32overf12;
  f32overf12 = f3_2_int(z) / f1_2_int(z);
  return f32overf12;
```

```
}

double inverter(double f, double (*func)(double)) { // inverts a
    function f(x) to find x
  double min, max, f_min, f_max, new, f_new=0., dx, dfdx;
  min = 0.01;
  max = 10.;
  f_min = (*func)(min);
  f_max = (*func)(max);

  while (fabs(f - f_new) > 1e-12) {
    dfdx = (f_max - f_min)/(max - min);
    dx = (f - f_min)/dfdx;
    new = min + dx;
    f_new = (*func)(new);

    if (f_new == f) {
      return new;
    } else if (f_new < f) {
      min = new;
      f_min = f_new;
    } else {
      max = new;
      f_max = f_new;
    }
  }
  return new;
}

int main() {
  clock_t start = clock();

  double Lin, N, Ef, Utot, Uone, alpha, z_int, first, Uint, kbT,
     Uev, dUdkT, dUdkToverkT, S = 0., lambdaT;
  int flag;

  printf("Enter L (nm): ");
  scanf("%lf", &Lin);
  int Linint = Lin;
  printf("Enter N: ");
```

```
scanf("%lf", &N);
int Nint = N;

char Lchar[10];
char Nchar[10];
snprintf(Lchar, 10, "%d", Linint);
snprintf(Nchar, 10, "%d", Nint);

char *filename = "fermi_";
char *fileext = ".txt";
char filefull[strlen(filename)+strlen(Lchar)+strlen(Nchar)+
    strlen(fileext)+2];
snprintf(filefull, sizeof(filefull), "%s%s_%s%s", filename,
    Nchar, Lchar, fileext);
FILE *f = fopen(filefull, "wb");

Ef = cbrt(9.*M_PI*N*N)*(debroglie*debroglie)/(4.*Lin*Lin);
Utot = (3./5.)*N*Ef;
Uone = (3./5.)*Ef;

printf("debroglie = %f nm\nEf      = %f eV\nUtot    = %f eV\nUtot
       = %f (unitless)\nUtot/N = %f eV\nUtot/N = %f (unitless)\n
    \n", debroglie, Ef, Utot, Utot/Ef, Uone, Uone/Ef);

fprintf(f, "N = %d\nL = %d nm\ncbrt(V/N) = %f\n", (int) N, (int
    ) Lin, Lin/cbrt(N));
fprintf(f, "debroglie = %f nm\nEf      = %f eV\nUtot    = %f eV\
    nUtot    = %f (unitless)\nUtot/N = %f eV\nUtot/N = %f (
    unitless)\n\n", debroglie, Ef, Utot, Utot/Ef, Uone, Uone/Ef)
    ;
fprintf(f, "kbT - T - Uint/Ef - Uev - dU/dkT(Cv/k) - S -
    lambdaT - flag\n");

kbT = 1e-6;
double h = 1e-6;
for (int i = 0; i < 50000; i++) {
  lambdaT = debroglie/sqrt(kbT); // thermal wavelength
  alpha = (N*lambdaT*lambdaT*lambdaT)/(2*(Lin*Lin*Lin)); // the
      factor 2 comes due to spin
  if (alpha > alphamax) {
```

```
      first = (0.6)*Ef*(1+(5./12.)*M_PI*M_PI*(kbT*kbT)/(Ef*Ef)-((
          M_PI*M_PI)*(M_PI*M_PI)/16)*(kbT*kbT)*(kbT*kbT)/((Ef*Ef)
          *(Ef*Ef))); // unitless energy per particle
      flag = 0;
    } else {
      z_int = inverter(alpha, f3_2_int);
      first = 1.5*kbT*F_int(z_int); // unitless energy per
          particle
      flag = 1;
    }

    Uev = first*N; // total internal energy in eV
    Uint = Uev/Ef; // unitless total internal energy

    if (alpha > alphamax) {
      dUdkT = N*M_PI*M_PI*kbT/(2*Ef) - 3*N*(M_PI*M_PI)*(M_PI*M_PI
          )*kbT*kbT*kbT/(20*Ef*Ef*Ef);
      // dUdkToverkT = dUdkT/(kbT);
      S = N*M_PI*M_PI*kbT/(2*Ef) - 3*N*(M_PI*M_PI)*(M_PI*M_PI)*
          kbT*kbT*kbT/(60*Ef*Ef*Ef);
    } else {
      dUdkT = 15*N*F_int(z_int)/4 - 9*N*F_int_2(z_int)/4;
      dUdkToverkT = dUdkT/(kbT);
      S += dUdkToverkT*h;
    }
    fprintf(f, "%.8f %10.6f %.12lf %.12f %12.6f %14.8f %10.6f %d\
        n", kbT, kbT*conversion, Uint, Uev, dUdkT, S, lambdaT,
        flag);
    kbT += h;
  }

  fclose(f);

  printf("Took %f seconds\n", ((double)clock()-start)/
      CLOCKS_PER_SEC);
  printf("Done!");

  return 0;
}
```

boseheatbath_v3.c

```
#define _USE_MATH_DEFINES

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <time.h>

static double conversion = 11604.51812;
static double g32_1 = 2.612375;

static double lambdacube = 0;
static double V = 0;

double g1_2_int(double z) { // finds the function by using
    composite simpson's rule
  if (z > 1) {
    printf("z is out of range!\n");
    return 1;
  }

  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
  int_old = -5.;
  int_new = -2.;
  n = 5;

  if (z <= 0.99999999999) {
    while (fabs(int_new - int_old) > 1e-10) {
      int_old = int_new;
      dx = x_max / (n - 1);
      int_new = z * (x_max * x_max) * exp(x_max * x_max) / ((exp(
          x_max * x_max) - z) * (exp(x_max * x_max) - z));
      mult = 4;
      for (i = 1; i < n; i++) {
```

```
        x = i * dx;
        int_new += mult * z * (x * x) * exp(x * x) / ((exp(x * x)
            - z) * (exp(x * x) - z));
        mult = 6 - mult;
      }
      int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
      n = (n - 1) * 2 + 1;
    }
  } else {
    while (fabs(int_new - int_old) > 1e-7) {
      int_old = int_new;
      dx = x_max / (n - 1);
      int_new = z * (x_max * x_max) * exp(x_max * x_max) / ((exp(
         x_max * x_max) - z) * (exp(x_max * x_max) - z));
      mult = 4;
      for (i = 1; i < n; i++) {
        x = i * dx;
        int_new += mult * z * (x * x) * exp(x * x) / ((exp(x * x)
            - z) * (exp(x * x) - z));
        mult = 6 - mult;
      }
      int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
      n = (n - 1) * 2 + 1;
    }
  }

  return int_new;
}

double g3_2_int(double z) { // finds the function by using
    composite simpson's rule
  if (z > 1) {
    printf("z is out of range!\n");
    return 1;
  }

  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
```

```
  int_old = -5.;
  int_new = -2.;
  n = 5;

  if (z <= 0.99999999999) {
    while (fabs(int_new - int_old) > 1e-10) {
      int_old = int_new;
      dx = x_max / (n - 1);
      int_new = z * (x_max * x_max) / (exp(x_max * x_max) - z);
      mult = 4;
      for (i = 1; i < n; i++) {
        x = i * dx;
        int_new += mult * z * (x * x) / (exp(x * x) - z);
        mult = 6 - mult;
      }
      int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
      n = (n - 1) * 2 + 1;
    }
  } else {
    while (fabs(int_new - int_old) > 1e-7) {
      int_old = int_new;
      dx = x_max / (n - 1);
      int_new = z * (x_max * x_max) / (exp(x_max * x_max) - z);
      mult = 4;
      for (i = 1; i < n; i++) {
        x = i * dx;
        int_new += mult * z * (x * x) / (exp(x * x) - z);
        mult = 6 - mult;
      }
      int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
      n = (n - 1) * 2 + 1;
    }
  }

  return int_new;
}

double g3_2_corr(double z) { // finds the function by using
    composite simpson's rule
  if (z > 1) {
```

```
  printf("z is out of range! %f\n", z);
  return 1;
}

int i, n, mult;
double x, x_max, int_old, int_new, dx;

x_max = 10.;
int_old = -5.;
int_new = -2.;
n = 5;

if (z <= 0.99999999999) {
  while (fabs(int_new - int_old) > 1e-10) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = z * (x_max * x_max) / (exp(x_max * x_max) - z);
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * z * (x * x) / (exp(x * x) - z);
      mult = 6 - mult;
    }
    int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
  }
} else {
  while (fabs(int_new - int_old) > 1e-7) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = z * (x_max * x_max) / (exp(x_max * x_max) - z);
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * z * (x * x) / (exp(x * x) - z);
      mult = 6 - mult;
    }
    int_new = 4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
  }
```

```
  }
  int_new = int_new + lambdacube*z/(V*(1-z));

  return int_new;
} // only to find z_int value by inverter

double g5_2_int(double z) { // finds the function by using
    composite simpson's rule
  if (z > 1) {
    printf("z is out of range!\n");
    return 1;
  }

  int i, n, mult;
  double x, x_max, int_old, int_new, dx;

  x_max = 10.;
  int_old = -5.;
  int_new = -2.;
  n = 5;

  while (fabs(int_new - int_old) > 1e-10) {
    int_old = int_new;
    dx = x_max / (n - 1);
    int_new = (x_max * x_max) * log(1 - z * exp(-x_max * x_max));
    mult = 4;
    for (i = 1; i < n; i++) {
      x = i * dx;
      int_new += mult * (x * x) * log(1 - z * exp(-x * x));
      mult = 6 - mult;
    }
    int_new = -4. * int_new * dx / (sqrt(4. * atan(1.)) * 3.);
    n = (n - 1) * 2 + 1;
  }
  return int_new;
}

double G_int(double z) {
  double g52overg32;
  g52overg32 = g5_2_int(z) / g3_2_int(z);
```

```
  return g52overg32;
}

double inverter(double f, double (*func)(double)) { //inverts a
    function f(x) to find x
  double min, max, f_min, f_max, new, f_new=0., f_new_old, dx,
      dfdx;
  min = 0.001;
  max = 0.99999;
  f_min = (*func)(min);
  f_max = (*func)(max);

  int i = 0;
    while (fabs(f - f_new) > 1e-10) {
      dfdx = (f_max - f_min)/(max - min);
      dx = (f - f_min)/dfdx;
      new = min + dx;
      f_new = (*func)(new);
      f_new_old = f_new;

      if (f_new == f) {
        return new;
      } else if (f_new < f) {
        min = new;
        f_min = f_new;
      } else {
        max = new;
        f_max = f_new;
      }

      if (fabs(f_new-f) < 1e-7 && f_new_old == f_new) {
        i++;
      }
      if (i == 100) {
        return new;
      }
      // printf("%.8f\n", fabs(f_new-f));
    }
    return new;
} // change max if z not in range error occurs
```

```
int main() {
  clock_t start = clock();

  double Lin, N, kbT, lambda, UoverN, alpha, z_int, Uev, Uint, C1
     , C2, kTc, vc, n0overN, S = 0., dUdkToverkT, K1, K2,
     Voverlambdacube, Vonez, lambdacubez;
  double Lprime = 8.101*1e-3; //nm

  printf("Enter L (nm): ");
  scanf("%lf", &Lin);
  int Linint = Lin;
  printf("Enter N: ");
  scanf("%lf", &N);
  int Nint = N;

  char Lchar[10];
  char Nchar[10];
  snprintf(Lchar, 10, "%d", Linint);
  snprintf(Nchar, 10, "%d", Nint);

  char *filename = "bose_";
  char *fileext = ".txt";
  char filefull[strlen(filename)+strlen(Lchar)+strlen(Nchar)+
     strlen(fileext)+2];
  snprintf(filefull, sizeof(filefull), "%s%s_%s%s", filename,
     Nchar, Lchar, fileext);
  FILE *g = fopen(filefull, "wb");

  V = Lin*Lin*Lin;
  C1 = 1.5*V/N;
  C2 = 1.5/N;
  kTc = (Lprime/Lin)*(Lprime/Lin)*cbrt((N/g32_1)*(N/g32_1));

  fprintf(g, "N = %d\nL = %d nm\nV = %f nm3\nV/N = %f\n<r> = %f
     nm\nLprime = %f nm\nkTc = %.8f eV\n\n", (int) N, (int) Lin,
     V, V/N, cbrt(V/N), Lprime, kTc);
  fprintf(g, "kbT - T - Uint - Uev - dU/dkT(Cv/k) - S - Vc - BEC
     - n0/N - lambda - z\n");
```

```
int xx = 1000;
int yy = 1;
int zz = 0;
double h = 4.1e-7;
for (int i = 1; i < 2; i++) {
  kbT = h*i;
  lambda = Lprime/(sqrt(kbT));
  lambdacube = lambda*lambda*lambda;
  vc = lambdacube/g32_1; // critical volume
  alpha = lambdacube*N/V;
  z_int = inverter(alpha, g3_2_corr);
  n0overN = z_int/(N*(1-z_int)); // ratio of number of
     condensed particles to number of total particles
  UoverN = C1*kbT*g5_2_int(z_int)/lambdacube - C2*kbT*log(1.-
     z_int); // energy per particle in eV

  Uev = UoverN*N; // total internal energy in eV
  Uint = Uev/kTc; // total unitless energy

  if (alpha > g32_1) {
    zz = 1;
  } else {
    zz = 0;
  }

  Voverlambdacube = V/lambdacube;
  Vonez = V*(1-z_int);
  K1 = 3.75*Voverlambdacube;
  K2 = 2.25*Vonez/lambdacube;
  lambdacubez = lambdacube*z_int;

  dUdkToverkT = (K1*g5_2_int(z_int) - 1.5*log(1-z_int) - K2*
     g3_2_int(z_int)*(Vonez*g3_2_int(z_int) + lambdacubez)/(
     Vonez*(1-z_int)*g1_2_int(z_int)+lambdacubez))/kbT;
  S += dUdkToverkT*h;

  fprintf(g, "%.8f %10.6f %22.10lf %18.10lf %12.6lf %18.6lf
     %13.9f %3d %10.8f %.8f %.12f\n", kbT, kbT*conversion, Uint
     , Uev, dUdkToverkT*kbT, S, vc, zz, n0overN, lambda, z_int)
     ;
```

```
    if (i > 0 && i % xx == 0) {
      printf("%d\n", xx*yy);
      yy++;
    }
  }


  // for (int i = 0; i < 10000; i++) {
  //   double z = i*h*100;
  //   double g12 = g1_2_int(z);
  //   double g32 = g3_2_int(z);
  //   double g52 = g5_2_int(z);
  //   fprintf(g, "%f %f %f %f\n", z, g12, g32, g52);
  // }

  // dUdkToverkT = (3.75*V*g5_2_int(z_int)/lambdacube - 1.5*log
     (1-z_int) - 2.25*g3_2_int(z_int)*((V*V*(1-z_int)*(1-z_int)
     *(1-z_int)*g3_2_int(z_int)/z_int + lambdacube*V*(1-z_int)
     *(1-z_int))/(V*(1-z_int)*(1-z_int)*lambdacube*g1_2_int(z_int
     )/z_int + lambdacube*lambdacube)))/kbT;
  // S += dUdkToverkT*h;

  fclose(g);

  printf("Took %f seconds\n", ((double)clock()-start)/
     CLOCKS_PER_SEC);
  printf("Done!\n");

  return 0;
}
```

fermi_montecarlo_fridge_v9.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>

#define TABLEN 50000
#define ARRAYSIZE 100
#define N 50 // # of intervals that tau is divided

static double kbT[TABLEN] = {0};
static double Uint[TABLEN] = {0};
static double heatbathentro[TABLEN] = {0};
static double conversion = 11604.51812;

struct Sinfodelta {
  double Sinfo, delta;
};

struct Sinfodelta Sinfo(double info[]) {
  struct Sinfodelta ret;
  double S = 0;
  int num0 = 0;
  int num1 = 0;
  double p0, p1, d, isdouble;
  for (int i = 0; i < ARRAYSIZE; i++) {
    if (info[i] == 0) {
      num0++;
    } else {
      num1++;
    }
  }
  isdouble = (double) ARRAYSIZE;
  p0 = num0/isdouble;
  p1 = num1/isdouble;
  d = p0 - p1;
```

```
  if (d == 1) {
    S = -((1.+d)/2.)*log((1.+d)/2.);
  } else {
    S = -((1.-d)/2.)*log((1.-d)/2.)-((1.+d)/2.)*log((1.+d)/2.);
  }
  ret.Sinfo = S;
  ret.delta = d;
  return ret;
}

struct energyentropy {
  double energy, hbentropy;
};

struct energyentropy temptoenergy(double singlevalue) {
  struct energyentropy ret;
  int i, j, k;
  if (singlevalue < kbT[0] || singlevalue > kbT[TABLEN - 1]) {
    printf("Out of limit! temptoenergy %f\n", singlevalue);
    exit(1);
  }
  i = 0;
  k = TABLEN - 1;
  while (k-i != 1) {
    j = (i+k)/2;
    if (kbT[j] > singlevalue) {
      k = j;
    } else {
      i = j;
    }
  }
  ret.energy = Uint[i] + (Uint[k] - Uint[i])*(singlevalue - kbT[i
     ])/(kbT[k] - kbT[i]);
  ret.hbentropy = heatbathentro[i] + (heatbathentro[k] -
     heatbathentro[i])*(singlevalue - kbT[i])/(kbT[k] - kbT[i]);
  return ret;
}

double energytotemp(double singlevalue) {
  int i,j,k;
```

```
  if (singlevalue < Uint[0] || singlevalue > Uint[TABLEN - 1]) {
    printf("Out of limit! energytotemp %f\n", singlevalue);
    exit(1);
  }
  i = 0;
  k = TABLEN - 1;
  while (k-i != 1) {
    j = (i+k)/2;
    if (Uint[j] > singlevalue) {
      k = j;
    } else {
      i = j;
    }
  }
  return kbT[i] + (kbT[k] - kbT[i])*(singlevalue - Uint[i])/(Uint
      [k] - Uint[i]);
}

double Sheat(double entropy[]) {
  double S = 0;
  for (int i = 0; i < N * ARRAYSIZE - 1; i++) {
    S = S + entropy[i];
  }
  S = S/ARRAYSIZE;
  return S;
}

int main() {
  clock_t start = clock();

  char *filename = "fermi_8000_10.txt";
  char *filepath = "/home/umutcan/Dropbox/School Stuff/Projects/
     Project_Maxwell/Heat Baths/Fermi/Data/";
  char filefull[strlen(filepath)+strlen(filename)+1];
  snprintf(filefull, sizeof(filefull), "%s%s", filepath, filename
     );
  FILE *fr = fopen(filefull, "r");
  if (fr == NULL) {
    printf("Could not open file %s\n", filefull);
    return 1;
```

```
}

char *data_ = "data_";
char datafull[strlen(data_)+strlen(filename)+1];
snprintf(datafull, sizeof(datafull), "%s%s", data_, filename);
FILE *fw = fopen(datafull, "wb");

char *graph_ = "graph_";
char graphfull[strlen(graph_)+strlen(filename)+1];
snprintf(graphfull, sizeof(graphfull), "%s%s", graph_, filename
   );
FILE *fg = fopen(graphfull, "wb");

char *mean_ = "mean_";
char meanfull[strlen(mean_)+strlen(filename)+1];
snprintf(meanfull, sizeof(meanfull), "%s%s", mean_, filename);
FILE *fm = fopen(meanfull,"wb");

double aa, bb, cc, dd, ee, ff, gg;
int hh;
int counter = 0;
while (!feof(fr)) {
  fscanf(fr, "%lf %lf %lf %lf %lf %lf %lf %d\n", &aa, &bb, &cc,
      &dd, &ee, &ff, &gg, &hh);
  kbT[counter] = aa;
  Uint[counter] = cc;
  heatbathentro[counter] = ff;
  counter++;
}
fclose(fr);

double wcud, wcdu, whud, whdu, P0d, P0u, P1d, P1u, tau, deltat,
    r, rr, Sh, DSh,
       sigma, omega, epsilon, KTc, KTh, deltademon,
          deltademon_u, Ucold, Uhot,
       Ef, Siin, Siout, DSi, deltain, deltaout, gamma, KTc_init
          , KTh_init,
       SEh, SEc, Udummy, Sdummy, ktdummy, Npart;
int maxint, num0i, num1i, num0f, num1f, Nhdu, Nhud, Ncdu, Ncud,
```

```
    tape, Nh, Nc, infocounter_in, infocounter_out,
       entropycounter;
char * level;
double info_in[ARRAYSIZE] = {0};
double info_out[ARRAYSIZE] = {0};
double entropy[N*ARRAYSIZE] = {0};
double Ucoldmean = 0., Uhotmean = 0., KTcmean = 0., KThmean =
   0., sigmamean = 0.,
       omegamean = 0., epsilonmean = 0., DSimean = 0., DShmean
          = 0.,
       deltainmean = 0., deltaoutmean = 0.; // variables for
          the mean part
int m = 0, mean = 21; // variables for the mean part
long int imean = 0;

srand(time(NULL));
maxint =~ (1 << (8*sizeof(int) - 1));

long int Nlong = N;
long int range_i = 5*1e3;
long int cell = 2*1e6; // # of cells on the tape
tau = 5e-1; // time it takes to move from one bit to another
deltat = tau/N; // time it takes to go from one interval to
   another

num0i = 0, num1i = 0, num0f = 0, num1f = 0; // # of initial and
    final states of bits
Nhdu = 0, Nhud = 0, Ncdu = 0, Ncud = 0; // # of total
   transitions while band is moving

Sh = 0.; // initial value of the change of heat entropy
epsilon = 0.; // initial value of epsilon

tape = 0; // holds the initial value of the bit on tape
level = "d"; // holds the initial state of the electron
infocounter_in = 0; // counter for array that holds incoming
   bits
infocounter_out = 0; // counter for array that holds outgoing
   bits
entropycounter = 0;
```

```
printf("Enter N: ");
scanf("%lf", &Npart);
double Utot = (3./5.)*Npart;

printf("Enter Ef (eV): ");
scanf("%lf", &Ef);

deltademon = 1e-4; // eV
deltademon_u = deltademon/Ef; // unitless
KTc = 0.025; // eV
KTh = 0.025; // eV
KTc_init = KTc;
KTh_init = KTh;
gamma = 1.;

struct energyentropy rc = temptoenergy(KTc);
Ucold = rc.energy;
SEc = rc.hbentropy;
if (strcmp(level, "d") == 0) {
  Udummy = Ucold - deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  omega = tanh((SEc - Sdummy)/2);
} else {
  Udummy = Ucold + deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  omega = tanh((Sdummy - SEc)/2);
}
wcud = 1.+omega; // transition rate from u to d due to cold
    bath
wcdu = 1.-omega; // transition rate from d to u due to cold
    bath

struct energyentropy rh = temptoenergy(KTh);
Uhot = rh.energy;
SEh = rh.hbentropy;
```

```
if (strcmp(level, "d") == 0) {
  Udummy = Uhot - deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  sigma = tanh((SEh - Sdummy)/2);
} else {
  Udummy = Uhot + deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  sigma = tanh((Sdummy - SEh)/2);
}
whud = (1.+sigma)*gamma; // transition rate from u to d due to
    hot bath
whdu = (1.-sigma)*gamma; // transition rate from d to u due to
    hot bath

fprintf(fg, "I)iteration II)time       III)U_cold      IV)U_hot
         V)KT_c   VI)KT_h  VII)T_c    VIII)T_h   IX)sigma     X
    )omega      XI)epsilon          XII)Delta SI   XIII)Delta
    ST  XIV)Delta S    XV)delta      XVI)delta'\n");
fflush(fg);
fprintf(fm, "I)iteration II)time       III)U_cold      IV)U_hot
         V)KT_c   VI)KT_h  VII)T_c    VIII)T_h   IX)sigma     X
    )omega      XI)epsilon          XII)Delta SI   XIII)Delta
    ST  XIV)Delta S    XV)delta      XVI)delta'\n");
fflush(fm);

for (long int i = 0; i < Nlong*cell; i++) {
  if (i % Nlong == 0) {
    rr = (double) rand() / maxint;
    if (rr < 0.) {
      tape = 1;
      num1i++;
    } else {
      tape = 0;
      num0i++;
    }
    info_in[infocounter_in] = tape;
```

```
  infocounter_in++;
  if (infocounter_in == ARRAYSIZE) {
    struct Sinfodelta Sin = Sinfo(info_in);
    Siin = Sin.Sinfo;
    deltain = Sin.delta;
    infocounter_in = 0;
  }
}
r = (double) rand() / maxint;
if (tape == 0 && level == "d") {
  P1u = wcdu*deltat;
  P0u = whdu*deltat;
  if (r <= P1u) {
    Ncdu++;
    tape = 1;
    level = "u";
    Sh = -deltademon/KTc;
    Ucold = Ucold - deltademon_u;
    KTc = energytotemp(Ucold);
    if (Ucold > Utot + deltademon_u) {
      struct energyentropy rc = temptoenergy(KTc);
      SEc = rc.hbentropy;
      Udummy = Ucold - deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      omega = tanh((SEc - Sdummy)/2);
    } else {
      omega = 1.;
    }
    wcud = 1.+omega;
    wcdu = 1.-omega;
  } else if (P1u < r && r <= P1u+P0u) {
    Nhdu++;
    level = "u";
    Sh = -deltademon/KTh;
    Uhot = Uhot - deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
    SEh = rh.hbentropy;
```

```
    Udummy = Uhot - deltademon_u;
    ktdummy = energytotemp(Udummy);
    struct energyentropy r = temptoenergy(ktdummy);
    Sdummy = r.hbentropy;
    sigma = tanh((SEh - Sdummy)/2);
    whud = (1.+sigma)*gamma;
    whdu = (1.-sigma)*gamma;
  } else {
    Sh = 0.;
  }
} else if (tape == 0 && level == "u") {
  P0d = whud*deltat;
  if (r <= P0d) {
    Nhud++;
    level = "d";
    Sh = deltademon/KTh;
    Uhot = Uhot + deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
    SEh = rh.hbentropy;
    Udummy = Uhot + deltademon_u;
    ktdummy = energytotemp(Udummy);
    struct energyentropy r = temptoenergy(ktdummy);
    Sdummy = r.hbentropy;
    sigma = tanh((Sdummy - SEh)/2);
    whud = (1.+sigma)*gamma;
    whdu = (1.-sigma)*gamma;
  } else {
    Sh = 0.;
  }
} else if (tape == 1 && level == "d") {
  P1u = whdu*deltat;
  if (r <= P1u) {
    Nhdu++;
    level = "u";
    Sh = -deltademon/KTh;
    Uhot = Uhot - deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
    SEh = rh.hbentropy;
```

```
      Udummy = Uhot - deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      sigma = tanh((SEh - Sdummy)/2);
      whud = (1.+sigma)*gamma;
      whdu = (1.-sigma)*gamma;
    } else {
      Sh = 0.;
    }
  } else if (tape == 1 && level == "u") {
    P1d = whud*deltat;
    P0d = wcud*deltat;
    if (r <= P1d) {
      Nhud++;
      level = "d";
      Sh = deltademon/KTh;
      Uhot = Uhot + deltademon_u;
      KTh = energytotemp(Uhot);
      struct energyentropy rh = temptoenergy(KTh);
      SEh = rh.hbentropy;
      Udummy = Uhot + deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      sigma = tanh((Sdummy - SEh)/2);
      whud = (1.+sigma)*gamma;
      whdu = (1.-sigma)*gamma;
    } else if (P1d < r && r <= P1d+P0d) {
      Ncud++;
      tape = 0;
      level = "d";
      Sh = deltademon/KTc;
      Ucold = Ucold + deltademon_u;
      KTc = energytotemp(Ucold);
      struct energyentropy rc = temptoenergy(KTc);
      SEc = rc.hbentropy;
      Udummy = Ucold + deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
```

```
    Sdummy = r.hbentropy;
    omega = tanh((Sdummy - SEc)/2);
    wcud = 1.+omega;
    wcdu = 1.-omega;
  } else {
    Sh = 0.;
  }
}
entropy[entropycounter] = Sh;
entropycounter++;
if (entropycounter == N*ARRAYSIZE) {
  DSh = Sheat(entropy);
  entropycounter = 0;
}
epsilon = (omega - sigma)/(1. - omega*sigma);
if (i % Nlong == Nlong-1) {
  if (tape == 0) {
    num0f++;
  } else {
    num1f++;
  }
  info_out[infocounter_out] = tape;
  infocounter_out++;
  if (infocounter_out == ARRAYSIZE) {
    struct Sinfodelta Sout = Sinfo(info_out);
    Siout = Sout.Sinfo;
    deltaout = Sout.delta;
    infocounter_out = 0;
  }
}
DSi = Siout - Siin;
if(i % range_i == Nlong*ARRAYSIZE-1){
  if (m == (mean-1)/2) {
    fprintf(fg, "%11ld %11.4f %15.8f %15.8f %f %f %10.6f
       %10.6f %f %f %.11f %9f %9f %9f %9f %9f\n", i, i*deltat
       , Ucold, Uhot, KTc, KTh, KTc*conversion, KTh*
       conversion, sigma, omega, epsilon, DSi, DSh, DSi+DSh,
       deltain, deltaout);
    fflush(fg);
  }
```

```
    m++, imean += i;
    Ucoldmean += Ucold, Uhotmean += Uhot, KTcmean += KTc,
       KThmean += KTh, sigmamean += sigma,
           omegamean += omega, epsilonmean += epsilon, DSimean
              += DSi, DShmean += DSh,
           deltainmean += deltain, deltaoutmean += deltaout;
    if (m == mean) {
      fprintf(fm, "%11ld %11.4f %15.8f %15.8f %f %f %10.6f
         %10.6f %f %f %.11f %9f %9f %9f %9f %9f\n", imean/m,
         imean*deltat/m, Ucoldmean/m, Uhotmean/m, KTcmean/m,
         KThmean/m, KTcmean*conversion/m, KThmean*conversion/m,
          sigmamean/m,
             omegamean/m, epsilonmean/m, DSimean/m, DShmean/m,
                (DSimean+DShmean)/m, deltainmean/m,
                deltaoutmean/m);
      fflush(fm);
      m = 0, imean = 0;
      Ucoldmean = 0., Uhotmean = 0., KTcmean = 0., KThmean =
         0., sigmamean = 0.,
             omegamean = 0., epsilonmean = 0., DSimean = 0.,
                DShmean = 0.,
             deltainmean = 0., deltaoutmean = 0.;
    }
  }
}

Nh = Nhud - Nhdu; // absolute # of change due to hot bath
Nc = Ncud - Ncdu; // absolute # of change due to cold bath

fprintf(fw, "kTc = %f eV\nkTh = %f eV\n\n# of intervals: %d\n#
   of cells: %d\ntau: %f sec.\ngamma = %f\nDelta Demon: %f eV\
   nDelta Demon: %.8f (unitless)\nEf: %f eV"
"\n\n# of 0s on the band (initially): %d\n# of 1s on the band (
   initially): %d"
"\n# of 0s on the band (finally): %d\n# of 1s on the band (
   finally): %d"
"\n\nprobability of incoming 0: %f\nprobability of incoming 1:
   %f\ndelta_tot = %f"
"\n\nprobability of outgoing 0: %f\nprobability of outgoing 1:
   %f\ndelta'_tot = %f"
```

```
  "\n\nNh = %d\nNc = %d"
  , KTc_init, KTh_init, N, cell, tau, gamma, deltademon,
     deltademon_u, Ef, num0i, num1i, num0f, num1f
  , (double) num0i/cell, (double) num1i/cell, (double) num0i/cell
     -(double) num1i/cell
  , (double) num0f/cell, (double) num1f/cell, (double) num0f/cell
     -(double) num1f/cell, Nh, Nc);

  fclose(fw);
  fclose(fg);
  fclose(fm);

  printf("Took %f seconds\n", ((double)clock()-start)/
     CLOCKS_PER_SEC);
  printf("Done!\n");

  return 0;
}
```

bose_montecarlo_fridge_v5.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <math.h>
#include <string.h>

#define TABLEN 75001 // 150k for N 8000, 50k for the rest
#define ARRAYSIZE 100
#define N 50 // # of intervals that tau is divided

static double kbT[TABLEN] = {0};
static double Uint[TABLEN] = {0};
static double heatbathentro[TABLEN] = {0};
static double conversion = 11604.51812;

struct Sinfodelta {
  double Sinfo, delta;
};

struct Sinfodelta Sinfo(double info[]) {
  struct Sinfodelta ret;
  double S = 0;
  int num0 = 0;
  int num1 = 0;
  double p0, p1, d, isdouble;
  for (int i = 0; i < ARRAYSIZE; i++) {
    if (info[i] == 0) {
      num0++;
    } else {
      num1++;
    }
  }
  isdouble = (double) ARRAYSIZE;
  p0 = num0/isdouble;
  p1 = num1/isdouble;
  d = p0 - p1;
```

```
  if (d == 1) {
    S = -((1.+d)/2.)*log((1.+d)/2.);
  } else {
    S = -((1.-d)/2.)*log((1.-d)/2.)-((1.+d)/2.)*log((1.+d)/2.);
  }
  ret.Sinfo = S;
  ret.delta = d;
  return ret;
}

struct energyentropy {
  double energy, hbentropy;
};

struct energyentropy temptoenergy(double singlevalue) {
  struct energyentropy ret;
  int i, j, k;
  if (singlevalue < kbT[0] || singlevalue > kbT[TABLEN - 1]) {
    printf("Out of limit! temptoenergy %f\n", singlevalue);
    exit(1);
  }
  i = 0;
  k = TABLEN - 1;
  while (k-i != 1) {
    j = (i+k)/2;
    if (kbT[j] > singlevalue) {
      k = j;
    } else {
      i = j;
    }
  }
  ret.energy = Uint[i] + (Uint[k] - Uint[i])*(singlevalue - kbT[i
     ])/(kbT[k] - kbT[i]);
  ret.hbentropy = heatbathentro[i] + (heatbathentro[k] -
     heatbathentro[i])*(singlevalue - kbT[i])/(kbT[k] - kbT[i]);
  return ret;
}

double energytotemp(double singlevalue) {
  int i,j,k;
```

```
  if (singlevalue < Uint[0] || singlevalue > Uint[TABLEN - 1]) {
    printf("Out of limit! energytotemp %f\n", singlevalue);
    exit(1);
  }
  i = 0;
  k = TABLEN - 1;
  while (k-i != 1) {
    j = (i+k)/2;
    if (Uint[j] > singlevalue) {
      k = j;
    } else {
      i = j;
    }
  }
  return kbT[i] + (kbT[k] - kbT[i])*(singlevalue - Uint[i])/(Uint
      [k] - Uint[i]);
}

double Sheat(double entropy[]) {
  double S = 0;
  for (int i = 0; i < N * ARRAYSIZE - 1; i++) {
    S = S + entropy[i];
  }
  S = S/ARRAYSIZE;
  return S;
}

int main() {
  clock_t start = clock();

  char *filename = "bose_8000_10.txt";
  char *filepath = "/home/umutcan/Dropbox/School Stuff/Projects/
     Project_Maxwell/Heat Baths/Bose/Data/";
  char filefull[strlen(filepath)+strlen(filename)+1];
  snprintf(filefull, sizeof(filefull), "%s%s", filepath, filename
     );
  FILE *fr = fopen(filefull, "r");
  if (fr == NULL) {
    printf("Could not open file %s\n", filefull);
    return 1;
```

```
}

char *data_ = "data_";
char datafull[strlen(data_)+strlen(filename)+1];
snprintf(datafull, sizeof(datafull), "%s%s", data_, filename);
FILE *fw = fopen(datafull, "wb");

char *graph_ = "graph_";
char graphfull[strlen(graph_)+strlen(filename)+1];
snprintf(graphfull, sizeof(graphfull), "%s%s", graph_, filename
   );
FILE *fg = fopen(graphfull, "wb");

char *mean_ = "mean_";
char meanfull[strlen(mean_)+strlen(filename)+1];
snprintf(meanfull, sizeof(meanfull), "%s%s", mean_, filename);
FILE *fm = fopen(meanfull,"wb");

double aa, bb, cc, dd, ee, ff, gg, ii, jj, kk;
int hh;
int counter = 0;
while (!feof(fr)) {
  fscanf(fr, "%lf %lf %lf %lf %lf %lf %lf %d %lf %lf %lf\n", &
     aa, &bb, &cc, &dd, &ee, &ff, &gg, &hh, &ii, &jj, &kk);
  kbT[counter] = aa;
  Uint[counter] = cc;
  heatbathentro[counter] = ff;
  counter++;
}
fclose(fr);

double wcud, wcdu, whud, whdu, P0d, P0u, P1d, P1u, tau, deltat,
    r, rr, Sh, DSh,
       sigma, omega, epsilon, KTc, KTh, deltademon,
          deltademon_u, Ucold, Uhot,
       kTcrit, Siin, Siout, DSi, deltain, deltaout, gamma,
          KTc_init, KTh_init,
       SEh, SEc, Udummy, Sdummy, ktdummy;
int maxint, num0i, num1i, num0f, num1f, Nhdu, Nhud, Ncdu, Ncud,
```

```
    tape, Nh, Nc, infocounter_in, infocounter_out,
        entropycounter;
char * level;
double info_in[ARRAYSIZE] = {0};
double info_out[ARRAYSIZE] = {0};
double entropy[N*ARRAYSIZE] = {0};
double Ucoldmean = 0., Uhotmean = 0., KTcmean = 0., KThmean =
    0., sigmamean = 0.,
        omegamean = 0., epsilonmean = 0., DSimean = 0., DShmean
            = 0.,
        deltainmean = 0., deltaoutmean = 0.,
        whudmean = 0., whdumean = 0., wcudmean = 0., wcdumean =
            0.; // variables for the mean part
int m = 0, mean = 21; // variables for the mean part
long int imean = 0;
double KTcprint = 0.;

srand(time(NULL));
maxint =~ (1 << (8*sizeof(int) - 1));

long int Nlong = N;
long int range_i = 2*1e5;
long int cell = 60*1e7; // # of cells on the tape
tau = 5e-1; // (double) tau;
deltat = tau/N;

num0i = 0, num1i = 0, num0f = 0, num1f = 0; // # of initial and
    final states of bits
Nhdu = 0, Nhud = 0, Ncdu = 0, Ncud = 0; // # of total
    transitions while band is moving

Sh = 0.; // initial value of the change of heat entropy
epsilon = 0.; // initial value of epsilon

tape = 0; // holds the initial value of the bit on tape
level = "d"; // holds the initial state of the electron
infocounter_in = 0; // counter for array that holds incoming
    bits
infocounter_out = 0; // counter for array that holds outgoing
    bits
```

```
entropycounter = 0;

printf("Enter kTcrit (eV): ");
scanf("%lf", &kTcrit);

deltademon = 1e-4; // eV
deltademon_u = deltademon/kTcrit; // unitless
KTc = 0.025; // eV
KTh = 0.025; // eV
KTc_init = KTc;
KTh_init = KTh;
gamma = 1.;

struct energyentropy rc = temptoenergy(KTc);
Ucold = rc.energy;
SEc = rc.hbentropy;
if (strcmp(level, "d") == 0) {
  Udummy = Ucold - deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  omega = tanh((SEc - Sdummy)/2);
} else {
  Udummy = Ucold + deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  omega = tanh((Sdummy - SEc)/2);
}
wcud = 1.+omega; // transition rate from u to d due to cold
    bath
wcdu = 1.-omega; // transition rate from d to u due to cold
    bath

struct energyentropy rh = temptoenergy(KTh);
Uhot = rh.energy;
SEh = rh.hbentropy;
if (strcmp(level, "d") == 0) {
  Udummy = Uhot - deltademon_u;
  ktdummy = energytotemp(Udummy);
```

```
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  sigma = tanh((SEh - Sdummy)/2);
} else {
  Udummy = Uhot + deltademon_u;
  ktdummy = energytotemp(Udummy);
  struct energyentropy r = temptoenergy(ktdummy);
  Sdummy = r.hbentropy;
  sigma = tanh((Sdummy - SEh)/2);
}
whud = (1.+sigma)*gamma; // transition rate from u to d due to
    hot bath
whdu = (1.-sigma)*gamma; // transition rate from d to u due to
    hot bath

fprintf(fg, "I)iteration   II)time          III)U_cold
    IV)U_hot          V)KT_c      VI)KT_h     VII)T_c     VIII)T_h
      IX)sigma      X)omega             XI)epsilon
                   XII)Delta SI   XIII)Delta ST   XIV)Delta S
    XV)delta       XVI)delta'      XVII)whud   XVIII)whdu XIX)wcud
          XX)wcdu\n");
fflush(fg);
fprintf(fm, "I)iteration   II)time          III)U_cold
    IV)U_hot          V)KT_c      VI)KT_h     VII)T_c     VIII)T_h
      IX)sigma      X)omega             XI)epsilon
                   XII)Delta SI   XIII)Delta ST   XIV)Delta S
    XV)delta       XVI)delta'      XVII)whud   XVIII)whdu XIX)wcud
          XX)wcdu\n");
fflush(fm);

for (long int i = 0; i < Nlong*cell; i++) {
  if (i % Nlong == 0) {
    rr = (double) rand() / maxint;
    if (rr < 0.) {
      tape = 1;
      num1i++;
    } else {
      tape = 0;
      num0i++;
    }
```

```
  info_in[infocounter_in] = tape;
  infocounter_in++;
  if (infocounter_in == ARRAYSIZE) {
    struct Sinfodelta Sin = Sinfo(info_in);
    Siin = Sin.Sinfo;
    deltain = Sin.delta;
    infocounter_in = 0;
  }
}
r = (double) rand() / maxint;
if (tape == 0 && level == "d") {
  P1u = wcdu*deltat;
  P0u = whdu*deltat;
  if (r < P1u) {
    Ncdu++;
    tape = 1;
    level = "u";
    Sh = -deltademon/KTc;
    Ucold = Ucold - deltademon_u;
    KTc = energytotemp(Ucold);
    if (Ucold > deltademon_u) {
      struct energyentropy rc = temptoenergy(KTc);
      SEc = rc.hbentropy;
      Udummy = Ucold - deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      omega = tanh((SEc - Sdummy)/2);
    } else {
      omega = 1.;
    }
    wcud = 1.+omega;
    wcdu = 1.-omega;
  } else if (P1u <= r && r < P1u+P0u) {
    Nhdu++;
    level = "u";
    Sh = -deltademon/KTh;
    Uhot = Uhot - deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
```

```
    SEh = rh.hbentropy;
    Udummy = Uhot - deltademon_u;
    ktdummy = energytotemp(Udummy);
    struct energyentropy r = temptoenergy(ktdummy);
    Sdummy = r.hbentropy;
    sigma = tanh((SEh - Sdummy)/2);
    whud = (1.+sigma)*gamma;
    whdu = (1.-sigma)*gamma;
  } else {
    Sh = 0.;
  }
} else if (tape == 0 && level == "u") {
  P0d = whud*deltat;
  if (r <= P0d) {
    Nhud++;
    level = "d";
    Sh = deltademon/KTh;
    Uhot = Uhot + deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
    SEh = rh.hbentropy;
    Udummy = Uhot + deltademon_u;
    ktdummy = energytotemp(Udummy);
    struct energyentropy r = temptoenergy(ktdummy);
    Sdummy = r.hbentropy;
    sigma = tanh((Sdummy - SEh)/2);
    whud = (1.+sigma)*gamma;
    whdu = (1.-sigma)*gamma;
  } else {
    Sh = 0.;
  }
} else if (tape == 1 && level == "d") {
  P1u = whdu*deltat;
  if (r <= P1u) {
    Nhdu++;
    level = "u";
    Sh = -deltademon/KTh;
    Uhot = Uhot - deltademon_u;
    KTh = energytotemp(Uhot);
    struct energyentropy rh = temptoenergy(KTh);
```

```
      SEh = rh.hbentropy;
      Udummy = Uhot - deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      sigma = tanh((SEh - Sdummy)/2);
      whud = (1.+sigma)*gamma;
      whdu = (1.-sigma)*gamma;
    } else {
      Sh = 0.;
    }
  } else if (tape == 1 && level == "u") {
    P1d = whud*deltat;
    P0d = wcud*deltat;
    if (r <= P1d) {
      Nhud++;
      level = "d";
      Sh = deltademon/KTh;
      Uhot = Uhot + deltademon_u;
      KTh = energytotemp(Uhot);
      struct energyentropy rh = temptoenergy(KTh);
      SEh = rh.hbentropy;
      Udummy = Uhot + deltademon_u;
      ktdummy = energytotemp(Udummy);
      struct energyentropy r = temptoenergy(ktdummy);
      Sdummy = r.hbentropy;
      sigma = tanh((Sdummy - SEh)/2);
      whud = (1.+sigma)*gamma;
      whdu = (1.-sigma)*gamma;
    } else if (P1d < r && r <= P1d+P0d) {
      Ncud++;
      tape = 0;
      level = "d";
      Sh = deltademon/KTc;
      Ucold = Ucold + deltademon_u;
      KTc = energytotemp(Ucold);
      struct energyentropy rc = temptoenergy(KTc);
      SEc = rc.hbentropy;
      Udummy = Ucold + deltademon_u;
      ktdummy = energytotemp(Udummy);
```

```
    struct energyentropy r = temptoenergy(ktdummy);
    Sdummy = r.hbentropy;
    omega = tanh((Sdummy - SEc)/2);
    wcud = 1.+omega;
    wcdu = 1.-omega;
  } else {
    Sh = 0.;
  }
}
entropy[entropycounter] = Sh;
entropycounter++;
if (entropycounter == N*ARRAYSIZE) {
  DSh = Sheat(entropy);
  entropycounter = 0;
}
epsilon = (omega - sigma)/(1. - omega*sigma);
if (i % Nlong == Nlong-1) {
  if (tape == 0) {
    num0f++;
  } else {
    num1f++;
  }
  info_out[infocounter_out] = tape;
  infocounter_out++;
  if (infocounter_out == ARRAYSIZE) {
    struct Sinfodelta Sout = Sinfo(info_out);
    Siout = Sout.Sinfo;
    deltaout = Sout.delta;
    infocounter_out = 0;
  }
}
DSi = Siout-Siin;
if (omega < 0.99999) {
  if(i % range_i == Nlong*ARRAYSIZE-1){
    if (m == (mean-1)/2) {
      fprintf(fg, "%12ld %14.4f %18.8f %18.8f %.8f %.8f %10.6
         f %10.6f %f %.12f %.17f %9f %9f %9f %9f %9f %10f %10
         f %14.10f %14.10f\n", i, i*deltat, Ucold, Uhot, KTc,
          KTh, KTc*conversion, KTh*conversion, sigma, omega,
         epsilon, DSi, DSh, DSi+DSh, deltain, deltaout,
```

```
                whud, whdu, wcud, wcdu);
        fflush(fg);
      }
      m++, imean += i;
      Ucoldmean += Ucold, Uhotmean += Uhot, KTcmean += KTc,
          KThmean += KTh, sigmamean += sigma,
              omegamean += omega, epsilonmean += epsilon,
                  DSimean += DSi, DShmean += DSh,
              deltainmean += deltain, deltaoutmean += deltaout,
                  whudmean += whud, whdumean += whdu,
              wcudmean += wcud, wcdumean += wcdu;
      if (m == mean) {
        fprintf(fm, "%12ld %14.4f %18.8f %18.8f %.8f %.8f %10.6
            f %10.6f %f %.12f %.17f %9f %9f %9f %9f %9f %10f %10
            f %14.10f %14.10f\n", imean/m, imean*deltat/m,
            Ucoldmean/m, Uhotmean/m, KTcmean/m, KThmean/m,
            KTcmean*conversion/m, KThmean*conversion/m,
            sigmamean/m,
                omegamean/m, epsilonmean/m, DSimean/m, DShmean/m
                    , (DSimean+DShmean)/m, deltainmean/m,
                    deltaoutmean/m, whudmean/m, whdumean/m,
                    wcudmean/m, wcdumean/m);
        fflush(fm);
        m = 0, imean = 0;
        Ucoldmean = 0., Uhotmean = 0., KTcmean = 0., KThmean =
            0., sigmamean = 0.,
                omegamean = 0., epsilonmean = 0., DSimean = 0.,
                    DShmean = 0.,
                deltainmean = 0., deltaoutmean = 0.,
                whudmean = 0., whdumean = 0., wcudmean = 0.,
                    wcdumean = 0.;
      }
    }
  } else {
    if (KTcprint != KTc) {
      fprintf(fm, "%12ld %14.4f %18.8f %18.8f %.8f %.8f %10.6f
          %10.6f %f %.12f %.17f %9f %9f %9f %9f %9f %10f %10f
          %14.10f %14.10f\n", i, i*deltat, Ucold, Uhot, KTc, KTh
          , KTc*conversion, KTh*conversion, sigma, omega,
          epsilon, DSi, DSh, DSi+DSh, deltain, deltaout,
```

```
                whud, whdu, wcud, wcdu);
        fflush(fm);
      }
      KTcprint = KTc;
    }
  }

  Nh = Nhud - Nhdu; // absolute # of change due to hot bath
  Nc = Ncud - Ncdu; // absolute # of change due to cold bath

  fprintf(fw, "kTc = %f eV\nkTh = %f eV\n\n# of intervals: %d\n#
     of cells: %d\ntau: %f sec.\ngamma = %f\nDelta Demon: %f eV\
     nDelta Demon: %.8f (unitless)\nkTcrit: %f eV"
  "\n\n# of 0s on the band (initially): %d\n# of 1s on the band (
     initially): %d"
  "\n# of 0s on the band (finally): %d\n# of 1s on the band (
     finally): %d"
  "\n\nprobability of incoming 0: %f\nprobability of incoming 1:
     %f\ndelta_tot = %f"
  "\n\nprobability of outgoing 0: %f\nprobability of outgoing 1:
     %f\ndelta'_tot = %f"
  "\n\nNh = %d\nNc = %d"
  , KTc_init, KTh_init, N, cell, tau, gamma, deltademon,
     deltademon_u, kTcrit, num0i, num1i, num0f, num1f
  , (double) num0i/cell, (double) num1i/cell, (double) num0i/cell
     -(double) num1i/cell
  , (double) num0f/cell, (double) num1f/cell, (double) num0f/cell
     -(double) num1f/cell, Nh, Nc);

  fclose(fw);
  fclose(fg);
  fclose(fm);

  printf("Took %f seconds\n", ((double)clock()-start)/
     CLOCKS_PER_SEC);
  printf("Done!\n");

  return 0;
}
```