DECOHERENCE EFFECT OF FLUCTUATING PATCH POTENTIALS ON OPTICAL TRAPPING

by

Ashmita Panda

A Master's Bypass Report

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Master of Science

Department of Physics

Indianapolis, Indiana

December 2024

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL

Dr. Ricardo Decca, Chair

Department of Physics

Indiana University, Indianapolis

Dr. Gautam Vemuri

Department of Physics

Indiana University, Indianapolis

Dr. Vivek Amin

Department of Physics

Indiana University, Indianapolis

To my family and friends

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Panda, Ashmita, Masters, Purdue University, November 2024. Decoherence Effect of Fluctuating Patch Potentials on Optical Trapping. Major Professor: Dr. Ricardo S. Decca.

The electric trapping potential produced by an optical tweezer can be approximated to be a simple harmonic oscillator near the focal point of the trap. A trapped quantum dot cooled down to the ground state of the oscillator potential can be a novel tool for probing weak interactions. The weak gravitational forces at sub-micron ranges can be probed using such a setup designing a mass-modulated attractor mass, which is free to move in the plane perpendicular to its surface. The quantum dot can be placed at close proximity to the attractor mass, at distances of about 200 nm. The center of mass of the quantum dot can be cooled down ground state of the harmonic oscillator potential by cavity cooling techniques. The attractor mass can then be designed to be one of the mirrors of the cavity, with a metallic coating to obtain a mirror-like surface, which leads to the formation of patch potentials on its surface. The moving patch potentials produce moving electric fields in space, which lead to unwanted decoherence effects on the energy levels of the harmonic trapping potential. The effect of these moving patch potentials on the lifetime of the energy levels of the oscillator are obtained in this report, by utilizing several analytic and computational methods. Another major source of noise present in all optical systems with a laser is the shot noise. But it is found to have no influence on the decoherence effect of the energy levels. In conclusion, we observe that the electrostatic interaction effect due to moving patch potentials produces the greatest dispersion in the energy levels and reduces the lifetime of the states to $\sim 43.60\,\mu$s.

1. INTRODUCTION

The gravitational force is one of the four fundamental forces in physics which has been studied for the longest time by physicists. From Newton's laws of gravity, to Einstein's theory of general relativity [1], our understanding of gravity has undergone several changes. The general theory of relativity has stood the test of experiments and is relevant for cosmic distance scales. While Newtonian gravity is valid as an approximation to general theory of relativity, there has developed a strong notion that it is does not give the complete picture of interactions at close distances [2]. The efforts for understanding the quantum nature of gravity and unification theories for gravity with standard model of physics is still ongoing, but these extension models predict the possibility of a low mass boson [3]. The existence of such a boson would also indicate the existence of another fundamental force mediated by them.

Bosons with a non-zero mass can be parametrized by a Yukawa-like potential, which can be added to the standard Newtonian gravity, with some interaction strength α and range λ given by the compton wavelength,

$$V = -\frac{Gm_1m_2}{r}\left(1 + \alpha e^{-r/\lambda}\right) . \tag{1.1}$$

Till date, a large region of the α-λ parameter space has been excluded by a range of experiments. There have been several long range experiments designed for studying planetary orbits, such as Earth-LAGEOS [4] , LAGEOS-Lunar [5] and Lunar precession [6]. Their results are summarized in figure 1.1 None of them have observed any deviations from General Relativity. There have also been several geophysics experiments which explore the ten to ten-thousand meter range in the parameter space, such as mine/borehole [7], ocean [8], lake [9], [10] and tower [11] experiments. Their results are also summarized in figure 1.1. There is still a large area which is unexplored in the α-λ parameter space, specially for $\lambda < 10^{-6}\,\mathrm{m}$. One major challenge encountered by any experiment trying to probe the sub-micron range is the interaction of the system with its thermal environment, as the force to be measured is too weak. One possible method of isolating a system of interest could be optically trapping it

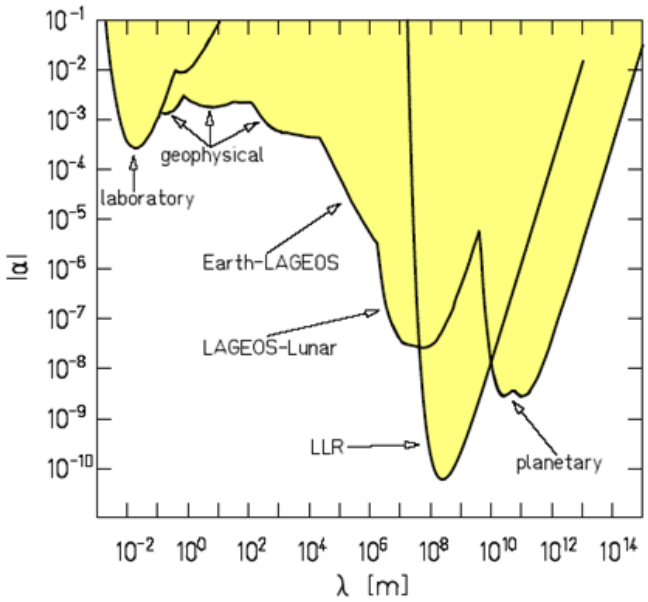

Figure 1.1. Parameter space of α-λ coupling of the Yukawa-like gravitational potential adapted from [12]. The shaded regions have been excluded by experiment or observational data.

using optical tweezers [13]. In an optical tweezer, the gradient force points towards the focal point of the laser beam, i.e., in the direction of the beam's maximum intensity, which causes any dielectric material to be effectively trapped and levitated at the focal point of the laser beam. The trapped dielectric can also be moved, allowing us to control its dynamics [14]. This gives us a novel tool to probe the sub-micron range for weak gravitational interactions.

The trapping potential of the optical tweezers can be approximated to a harmonic oscillator about the focal point. If we can cool the trapped dielectric down to the ground state of the harmonic oscillator, it gives us a probe to detect weak interactions, by looking for the excitations of the dielectric to higher energy states. The temperature we need to cool the dielectric down to depends on the mechanical frequency, Ω of the harmonic oscillator. The mechanical frequency Ω is related to the spring constant of the optical trap and mass of the trapped dielectric as

$$\Omega = \sqrt{\frac{k_{sp}}{M}}\,. \tag{1.2}$$

The spring constant of the trapping laser is obtained to be [15]

$$\vec{k}_{sp} = \begin{cases} k_{sp}(x) = \dfrac{\alpha_{dielectric}}{x}\nabla(\vec{E}.\vec{E*})\Big|_{y,z=0} = \dfrac{4\mathrm{Re}(\alpha_{dielectric})(NA)^4\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,, \\ k_{sp}(y) = \dfrac{\alpha_{dielectric}}{y}\nabla(\vec{E}.\vec{E*})\Big|_{y,z=0} = \dfrac{4\mathrm{Re}(\alpha_{dielectric})(NA)^4\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,, \\ k_{sp}(z) = \dfrac{\alpha_{dielectric}}{z}\nabla(\vec{E}.\vec{E*})\Big|_{y,z=0} = \dfrac{2\mathrm{Re}(\alpha_{dielectric})(NA)^6\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,. \end{cases} \tag{1.3}$$

In the above expression, $\alpha_{dielectric}$ is the polarizability of the trapped dielectric, NA is the numerical aperture, λ_{laser} is the wavelength of the laser, P_{laser} is the laser power and $\vec{E}$, $\vec{E}*$ are the electric field of the laser and its complex conjugate. Thus, we can observe that k_{sp} is proportional to the power of the laser and polarizability of the trapped dielectric, i.e.,

$$k_{sp} \propto \alpha_{dielectric}P_{laser} \implies k_{sp} = \kappa\,\alpha_{dielectric}P_{laser}\,. \tag{1.4}$$

κ is an appropriate proportionality constant. The temperature we need to cool the system down to get the ground state populated and have the first excited state almost empty, is approximately given as

$$T \lesssim \frac{\hbar\Omega}{k_B}\,, \tag{1.5}$$

where k_B is the Boltzmann constant and $\hbar$ is the Planck's constant. Thus, we need to increase the mechanical frequency of the harmonic oscillator trap to facilitate cooling into the ground state.

The polarizability of any spherical dielectric with core-shell structure is derived in chapter 2.2 and is found to be proportional to the volume of the sphere. The complete expression for polarizability is given as

$$\alpha_{dielectric} = 4\pi\epsilon_0 r_{shell}^3\left(\frac{\epsilon_e - \epsilon_0}{\epsilon_e + 2\epsilon_0}\right)\,, \tag{1.6}$$

$$\epsilon_e = \epsilon_S\frac{r_{shell}^3(\epsilon_C + 2\epsilon_S) + 2r_{core}^3(\epsilon_C - \epsilon_S)}{r_{shell}^3(\epsilon_C + 2\epsilon_S) - r_{core}^3(\epsilon_C - \epsilon_S)}\,. \tag{1.7}$$

In the above expressions, r_{shell} is the radius of the particle, r_{core} is the radius of the core of the particle, ϵ_e is the effective dielectric constant, ϵ_C is the dielectric constant of the core

material and ϵ_S is the dielectric constant of the shell material. The mass of any particle is proportional to its volume, i.e.,

$$M = \rho_e \frac{4\pi}{3} r_{shell}^3 \,, \tag{1.8}$$

such that ρ_e is the effective density of the material and r_{shell} is the radius of the sphere. The expression for ρ_e can be obtained in terms of ρ_{core}, the density of the core material and ρ_{shell}, the density of the shell material as

$$\rho_e = \left(\frac{r_{core}}{r_{shell}}\right)^3 (\rho_{core} - \rho_{shell}) + \rho_{shell} \,. \tag{1.9}$$

This allows us to write the expression for the spring constant as

$$\Omega = \sqrt{\frac{\kappa\, \alpha_{dielectric} P_{laser}}{M}} = \sqrt{\frac{\kappa P_{laser}}{\rho_e} 4\pi\epsilon_0 \left(\frac{\epsilon_e - \epsilon_0}{\epsilon_e + 2\epsilon_0}\right)} \,, \tag{1.10}$$

and to first order, effect r_{shell}^3 gets canceled. So, we can increase the mechanical frequency by increasing the power of the laser and the effective dielectric constant of the material being used. Increasing the laser power also increases the risk of melting the trapped particle. Thus we need to use materials which can sustain their structure at high laser powers.

We can significantly increase the dielectric constant of the material by using quantum dots [16] as our trapped particles. Quantum dots are nanocrystalline semiconductors with radii in the nanometer ranges. We specifically choose CdSe-CdS quantum dots [17] due to their high dielectric constants for a high wavelength light, in the order of $\lambda_{laser} \sim 1.0\,\mu$m. Also quantum dots have an inherent cooling mechanism of its internal temperature, which would prevent them from melting in the presence of a high power laser [18].

The significant challenge of the project is to cool the center of mass of the trapped quantum dot to the ground state of the harmonic oscillator. In recent studies, Delic *et al.* [19] have reported the 3-dimensional cooling of a trapped silica nanosphere in an optical cavity. They used lasers of wavelength $\lambda_{laser} = 1064\,$nm, power $P \approx 0.17\,$W, and a high-finesse ($\mathcal{F} = 73.0$) cavity inside a vacuum chamber to to cool down silica nanospheres of radius 71.5 nm. They also used an elliptical trap with non-degenerate mechanical frequencies, $(\Omega_x, \Omega_y, \Omega_z) = 2\pi(190, 170, 38)\,$kHz and obtained a minimal temperature of $\sim 1\,$K. The

ground state cooling temperature for the given Delic *et al.*[19] trap parameters is in the order of $\sim 10^{-5}$ K.

We can use a setup similar to Delic *et al.* [19] to probe the weak interactions at sub-micron ranges. One major modification would be to replace one of the cavity mirrors with a mass-modulated attractor mass, free to move in the plane perpendicular to the axis of the cavity, with the trapped and cooled quantum dot located around 200 nm from its surface. We would also need to the trapping laser directly behind the attractor mass, outside of the cavity, to ensure that the focal point of the laser lies close enough to the attractor mass.

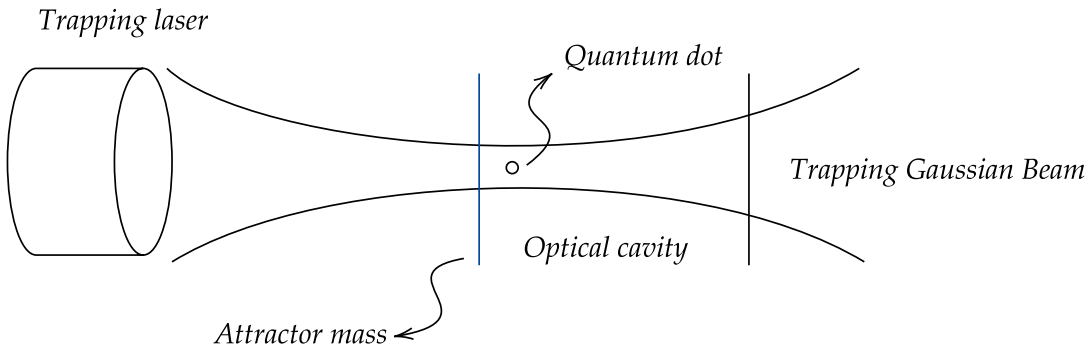

Figure 1.2. Possible experimental setup

In this setup, to ensure the optical cavity is formed, we need to design the attractor mass such that it is transparent to the trapping laser beam, but opaque to the cavity-filling beam. Also, we need to obtain mirror finish on the side of the attractor mass facing the quantum dot, which would require a metallic coating on its surface. Thus, we can approximate the attractor mass surface to a conductor plate. As it is a conductor plate, it would have spatial variations in its electrostatic potential, known as patch potentials. The patch potentials would produce electric fields in space, which would interact with the trapped quantum dot placed in close proximity. The patch potentials, and consequently the electric fields produced, are also moving in space as the attractor mass is free to rotate. The moving electric fields would cause a dispersion of the energy levels of the harmonic trapping potential. The amount

of dispersion would depend on the polarizability of the quantum dot and the strength of the fluctuating patch potentials.

In this report, we have attempted to obtain an estimation of the dispersion of the energy levels, and thus the lifetime of the energy levels of the harmonic trapping potential. We first obtain the polarizability of the quantum dot in chapter 2. In chapter 3, we evaluate the electric field for a single conducting disk. In chapter 4, we simulate the surface of the conductor plate to be composed of $N = 3000$ conducting disks and then evaluate the spatial variations in the electric fields. In the final chapter 5, we use the results from the previous chapters and obtain the lifetime of the energy levels of the trapped quantum dot. We also consider the shot noise of the laser and evaluate its effect on lifetime of the oscillator states.

2. POLARIZATION OF QUANTUM DOT

Quantum dots are semiconductor nanocrystalline particles with optical and electronic properties influenced by quantum mechanical effects. In our system, we will be using a CdSe-CdS quantum dot [18] with a core-shell structure as the particle in our optical trap. The physical properties of the dot have an effect on the properties of the trap. Polarizability of the dot is the most important one, which directly affects the spacing of the energy levels in the trap potential. We can estimate the polarizability by classically solving the Laplace equation. We can also solve for the wavefunction of the quantum dot and obtain the polarizability of the dot quantum mechanically via the Stark effect.

We first consider the classical model for a CdSe-CdS dot with a core-shell structure in Section 2.1. The Laplace equation equation is solved to find the potential for the dot. From the expansion of the potential outside the dot, we identify the dipole term and obtain the effective permittivity and classical polarizability of the dot.

In Section 2.2 we consider the quantum model for the CdSE-CdS dot. We write the unperturbed Hamiltonian for the same, with potential bounds of the core-shell structure for the electrons and holes in the conduction and valence bands. We solve the Schrodinger's equation for the system and numerically obtain the unperturbed wavefunctions.

The Coulomb interaction between the electrons and holes in the quantum dot produces a non-negligible effect on the energy levels and wavefunctions, which we explore in Section 2.3. We add the Coulomb interaction as a perturbation to the Hamiltonian and obtain the first order correction to the ground state.

In Section 2.4 we consider the quantum dot to be placed in an electric field and explore the weak Stark effect. We add the Stark energy to the previously perturbed Hamiltonian as a further perturbation, and obtain the second order correction to the ground state energy. The quantum approximation for the polarizability of the quantum dot can then be obtained as the change in the energy of the state with respect to the electric field.

2.1 Classical Polarizability of a Quantum Dot

We plan to use a CdSe-CdS quantum dot with a core-shell structure in our optical trap setup, illustrated in figure 2.1. We can estimate the polarizability of the quantum dot classically, by solving for the Laplace equation to find the electric potential for the neutral dot. We can start with the general solution.

$$\Phi(r,\theta) = \sum_{l=0}^{\infty} \left(A_l\, r^l + \frac{B_l}{r^{l+1}} \right) P_l(\cos\theta)\,. \tag{2.1}$$

It has a cadmium selenide core of radius $r = r_{core}$, permittivity ϵ_C and a cadmium sulfide shell of radius $r = r_{shell}$, permittivity ϵ_S. We assume vacuum outside the dot. $P_l(\cos\theta)$

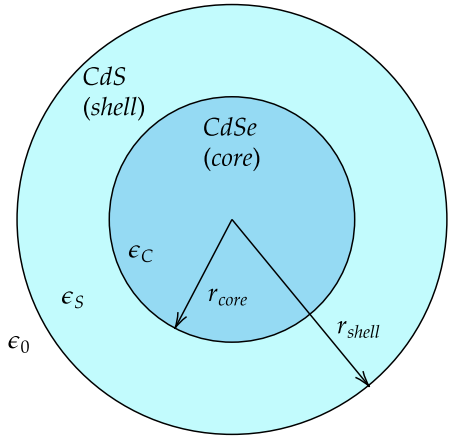

Figure 2.1. CdSe-CdS core-shell quantum dot

are the Legendre polynomials. The potential has different expansions inside the core, in the shell and outside the dot, as shown in figure (2.1).

$$\Phi_{core}(r,\theta) = \sum_{l=0}^{\infty} A_l\, r^l\, P_l(\cos\theta)\,, \tag{2.2}$$

$$\Phi_{shell}(r,\theta) = \sum_{l=0}^{\infty} \left(B_l\, r^l + \frac{C_l}{r^{l+1}} \right) P_l(\cos\theta)\,, \tag{2.3}$$

$$\Phi_{out}(r,\theta) = -E_0 r\cos\theta + \sum_{l=0}^{\infty} \frac{F_l}{r^{l+1}}\, P_l(\cos\theta)\,. \tag{2.4}$$

We now need to enforce boundary conditions to ensure the continuity of potential and rate of change of potential across at each boundary, i.e., the core-shell boundary and the shell-vacuum boundary, illustrated in figure (2.1). Also, far outside the dot, the potential goes to infinity, i.e., $\Phi_{out} \longrightarrow -E_0 r\cos\theta$ as $r \gg r_{shell}$.

We can derive the polarizability for the dot using the above expressions. The complete derivation is included in Appendix A.1. The final expression for polarizability is given as

$$\boxed{\begin{aligned} \alpha_{dot} &= 4\pi\epsilon_0 r_{shell}^3 \left(\frac{\epsilon_e - \epsilon_0}{\epsilon_e + 2\epsilon_0}\right), \qquad (2.5)\\ \epsilon_e &= \epsilon_S \frac{r_{shell}^3(\epsilon_C + 2\epsilon_S) + 2\, r_{core}^3(\epsilon_C - \epsilon_S)}{r_{shell}^3(\epsilon_C + 2\epsilon_S) - r_{core}^3(\epsilon_C - \epsilon_S)}. \qquad (2.6) \end{aligned}}$$

The real part of the permittivity of any material is given as :

$$\epsilon_R = (n^2 - k^2)\epsilon_0 \,. \tag{2.7}$$

n and k are the real and imaginary part of the refractive index of the material at given wavelengths of light. For CdSe [20] and CdS [21], at $\lambda = 1\,\mu$m, the values are :

$$n_{\text{CdSe}} = 2.3847\,, \quad k_{\text{CdSe}} = 0.065147\,, \tag{2.8}$$

$$n_{\text{CdS}} = 2.2708\,, \quad k_{\text{CdS}} = -5.158 \times 10^{-18}\,. \tag{2.9}$$

We consider light of wavelength $\lambda = 1\,\mu$m as a $\lambda = 1064$ nm$= 1.064\,\mu$m wavelength laser used by Delic *et al.* [19] in their setup for optical cooling. The value of absorption coefficient of CdSe, k_{CdSe} is too large and for our actual experimentation higher values of λ will be used. This gives us the permittivity for CdSe and CdS as

$$\epsilon_{\text{CdSe}} = \epsilon_C = 5.683\,\epsilon_0\,, \tag{2.10}$$

$$\epsilon_{\text{CdS}} = \epsilon_S = 5.322\,\epsilon_0\,. \tag{2.11}$$

Now using equation (2.6) to find the effective permittivity of the dot,

$$\boxed{\epsilon_e = 5.367\,\epsilon_0\,.} \tag{2.12}$$

Thus, using equation (2.5) we can find the classical polarizability of the quantum dot

$$\boxed{\alpha_{dot} = 1.608 \times 10^{-36}\,\mathrm{C\,m^2\,V^{-1}}\,.} \tag{2.13}$$

2.2 Unperturbed Wavefunctions of a Quantum Dot

A quantum dot is effectively a particle in a box system. We have electrons and holes in the conduction and valence bands respectively, interacting with each other via the Coulomb interaction. The confinement potential energy for the quantum dot, illustrated in figure 2.2,

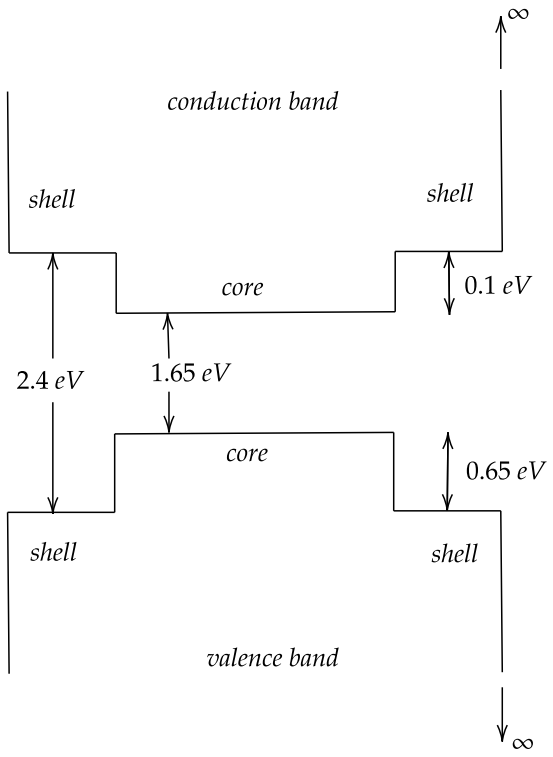

Figure 2.2. Potential well of a quantum dot showing confinement potentials of the core and shell for both valence and conduction bands

can be written as

$$V^{c/v} = \begin{cases} V^{c/v}_{core} & : \, r \leq r_{core} \\ V^{c/v}_{shell} & : \, r_{core} < r < r_{shell} \\ \infty & : \, r \geq r_{shell} \end{cases} \tag{2.14}$$

The c/v stand for "conduction band" and "valence band" respectively. We always treat the $V^{c/v}_{core} = 0$ eV and define the rest of the potential energies accordingly. Electrons reside in the conduction band and holes in the valence band, in both core and shell.

The Hamiltonian of the quantum dot system, disregarding Coulomb interaction, is

$$H_0 = \left(\frac{\hat{p_e}^2}{2m_e} + V_e \right) + \left(\frac{\hat{p_h}^2}{2m_h} + V_h \right) = H_0^e + H_0^h \, . \tag{2.15}$$

Then the Schrödinger equation for the system and the wavefunction for the exciton states of the quantum dot are obtained,

$$\begin{aligned} H_0 \Psi^{exc} &= E^{exc} \Psi^{exc} \, , \\ \implies (H_0^e + H_0^h) \Psi^{exc} &= E^{exc} \Psi^{exc} \, . \end{aligned}$$

The bound state of an electron and hole is known as an exciton state. We can write the exciton wavefunction as a product of the individual wavefunctions of the conduction electrons and valence holes as

$$\Psi^{exc}(r) = \psi^e(r) \, \psi^h(r) \, . \tag{2.16}$$

The product of the wavefunction can be put into the RHS of the Schrodinger equation to obtain

$$\begin{aligned} \implies E^{exc} \Psi^{exc} &= (H_0^e + H_0^h) \psi^e \, \psi^h \, , \\ \implies E^{exc} \Psi^{exc} &= (H_0^e \, \psi^e) \, \psi^h + (H_0^h \, \psi^h) \, \psi^e \, , \\ \implies E^{exc} \Psi^{exc} &= E^e \, \psi^e \, \psi^h + E^h \, \psi^e \psi^h \, , \\ \implies E^{exc} \Psi^{exc} &= (E^e + E^h) \, \psi^e \, \psi^h = (E^e + E^h) \, \Psi^{exc} \, . \end{aligned}$$

Thus, we obtain the energy of the exciton as the sum of the individual conduction band electron and valence band hole states,

$$E^{exc} = E^e + E^h \,. \tag{2.17}$$

The wavefunctions for the electrons and holes can be separated in two regions, core and shell, as

$$\psi_e^c = \psi_e^{c,core} + \psi_e^{c,shell} \,, \quad \psi_h^v = \psi_h^{v,core} + \psi_h^{v,shell} \,. \tag{2.18}$$

The radial part of the conduction electrons/valence holes wavefunctions can be written using spherical Bessel functions of first kind (j_l) and second kind (n_l) [18], given as

$$\psi^{c/v,core}(r,\theta,\varphi) = A_{e/h,l}^{c/v,core}\, j_l(k_{e/h,l}^{c/v,core} r)\, Y_l^m(\theta,\varphi) \,, \tag{2.19}$$

$$\psi^{c/v,shell}(r,\theta,\varphi) = \left[A_{e/h,l}^{c/v,shell}\, j_l(k_{e/h,l}^{c/v,shell} r) + B_{e/h,l}^{c/v,shell}\, n_l(k_{e/h,l}^{c/v,shell} r)\right] Y_l^m(\theta,\varphi) \,. \tag{2.20}$$

$A_{e/h,l}^{c/v,core}$, $A_{e/h,l}^{c/v,shell}$ and $B_{e/h,l}^{c/v,shell}$ are the normalization constants, and $Y_m^l(\theta,\varphi)$ are the spherical harmonics. The k values obtained from Muchuan Hua's thesis [18] are

$$k_{e/h}^{c/v,core/shell} = \sqrt{\frac{2\,(\varepsilon_{e/h}^{c/v,core/shell} - V^{c/v,core/shell})\, m_{e/h}^{c/v,core/shell}}{\hbar^2}} \,. \tag{2.21}$$

$\varepsilon_{e/h}^{c/v,core/shell}$ is the energy for conduction electrons or valence holes, which can be found by enforcing the boundary conditions for the wavefunctions. We must also note that the conduction band electrons and valence band holes have different effective masses in the core and the shell. They can be written in terms of the electron's rest mass m_0 [18], as shown in table 2.1.

Table 2.1. Mass values for electron and holes

Material	Electron Effective Mass	Heavy Hole Effective Mass
CdSe	0.11 m_0	0.8 m_0
CdS	0.14 m_0	0.51 m_0

The wavefunctions and their derivatives need to be continuous across the core-shell boundary, and the shell-wavefunction should vanish at the shell-boundary [18]. Hence,

$$\psi_{e/h}^{c/v,core}\Big|_{r=r_{core}} = \psi_{e/h}^{c/v,shell}\Big|_{r=r_{core}}\,, \tag{2.22}$$

$$\frac{1}{m_{e/h}^{c/v,core}}\frac{\partial}{\partial r}\psi_{e/h}^{c/v,core}\Big|_{r=r_{core}} = \frac{1}{m_{e/h}^{c/v,shell}}\frac{\partial}{\partial r}\psi_{e/h}^{c/v,shell}\Big|_{r=r_{core}}\,, \tag{2.23}$$

$$\psi_{e/h}^{c/v,shell}\Big|_{r=r_{shell}} = 0\,. \tag{2.24}$$

We can enforce the first two boundary conditions, (2.22) and (2.23), and obtain the expressions of A and B coefficients [18]. The complete derivation is included in Appendix A.2.

$$A_{e/h,l}^{c/v,shell} = \frac{j_l(k_{e/h,l}^{c/v,core}r_{core})n_l'(k_{e/h,l}^{c/v,shell}r_{core}) - \frac{m_{e/h}^{c/v,shell}k_{e/h,l}^{c/v,core}}{m_{e/h}^{c/v,core}k_{e/h,l}^{c/v,shell}}j_l'(k_{e/h,l}^{c/v,core}r_{core})n_l(k_{e/h,l}^{c/v,shell}r_{core})}{j_l(k_{e/h,l}^{c/v,shell}r_{core})n_l'(k_{e/h}^{c/v,shell}r_{core}) - j_l'(k_{e/h,l}^{c/v,shell}r_{core})n_l(k_{e/h,l}^{c/v,shell}r_{core})}A_{e/h,l}^{c/v,core}\,, \tag{2.25}$$

$$B_{e/h,l}^{c/v,shell} = \frac{j_l(k_{e/h,l}^{c/v,core}r_{core})j_l'(k_{e/h,l}^{c/v,shell}r_{core}) - \frac{m_{e/h}^{c/v,shell}k_{e/h,l}^{c/v,core}}{m_{e/h}^{c/v,core}k_{e/h,l}^{c/v,shell}}j_l'(k_{e/h,l}^{c/v,core}r_{core})j_l(k_{e/h,l}^{c/v,core}r_{core})}{j_l'(k_{e/h,l}^{c/v,shell}r_{core})n_l(k_{e/h}^{c/v,shell}r_{core}) - j_l(k_{e/h,l}^{c/v,shell}r_{core})n_l'(k_{e/h,l}^{c/v,shell}r_{core})}A_{e/h,l}^{c/v,core}\,. \tag{2.26}$$

The A^{core} can be obtained by normalizing the complete wavefunction of the quantum dot for both electrons and holes.

This problem needs to solved computationally. The detailed code and algorithm are included in Appendix A.3, and the results are summarized here.

The quantum dot considered for this problem has a radius of 2.9 nm, with $r_{core} = 1.5\,\text{nm}$ and $r_{shell} = 2.9\,\text{nm}$. Thus, the shell thickness is about 1.4 nm. The core potential for both conduction electron and valence holes are taken to be 0, i.e., $V^{c/v,core} = 0.0\,\text{eV}$. So, the potential for the shell can be taken as the difference between the core and the shell for each band, shown in figure (2.2). The allowed energy values, in increasing order of energy of the

exciton, obtained computationally, is given in table 2.2. The energy level for the exciton is represented by "n".

Table 2.2. n, l_{CE} and l_{VH} represent energy levels for the bounded exciton, conduction band electrons and valence band holes respectively. $n = 0$, $l_{CE} = 0$ and $l_{VH} = 0$ represent respective ground states. E_l^{CE}, E_l^{VH} and E_n^{exc} represent energy values for the corresponding energy levels of the conduction band electron, valence band hole and bounded exciton respectively.

n	l_{CE}	l_{VH}	E_l^{CE}	E_l^{VH}	E_n^{exc}
0	0	0	0.37468117 eV	0.15986651 eV	0.53454768 eV
1	0	1	0.37468117 eV	0.32234768 eV	0.69702885 eV
2	0	2	0.37468117 eV	0.52251117 eV	0.89719234 eV
3	1	0	0.76754997 eV	0.15986651 eV	0.92741648 eV
4	1	1	0.76754997 eV	0.32234768 eV	1.08989765 eV
5	0	3	0.37468117 eV	0.75589044 eV	1.13057161 eV
6	1	2	0.76754997 eV	0.52251117 eV	1.29006114 eV
7	2	0	1.22241281 eV	0.15986651 eV	1.38227932 eV
8	1	3	0.76754997 eV	0.75589044 eV	1.52344041 eV
9	2	1	1.22241281 eV	0.32234768 eV	1.54476049 eV
10	2	2	1.22241281 eV	0.52251117 eV	1.74492398 eV
11	3	0	1.73634639 eV	0.15986651 eV	1.89621290 eV
12	2	3	1.22241281 eV	0.75589044 eV	1.97830325 eV
13	3	1	1.73634639 eV	0.32234768 eV	2.05869407 eV
14	3	2	1.73634639 eV	0.52251117 eV	2.25885756 eV
15	3	3	1.73634639 eV	0.75589044 eV	2.49223683 eV

The energy values reported are with respect to $V^{c/v,core} = 0$ and $V^{c,shell} = 0.1$ eV and $V^{h,shell} = 0.65$ eV. The first four wavefunctions in order of exciton energy are illustrated in figures 2.3-2.6.

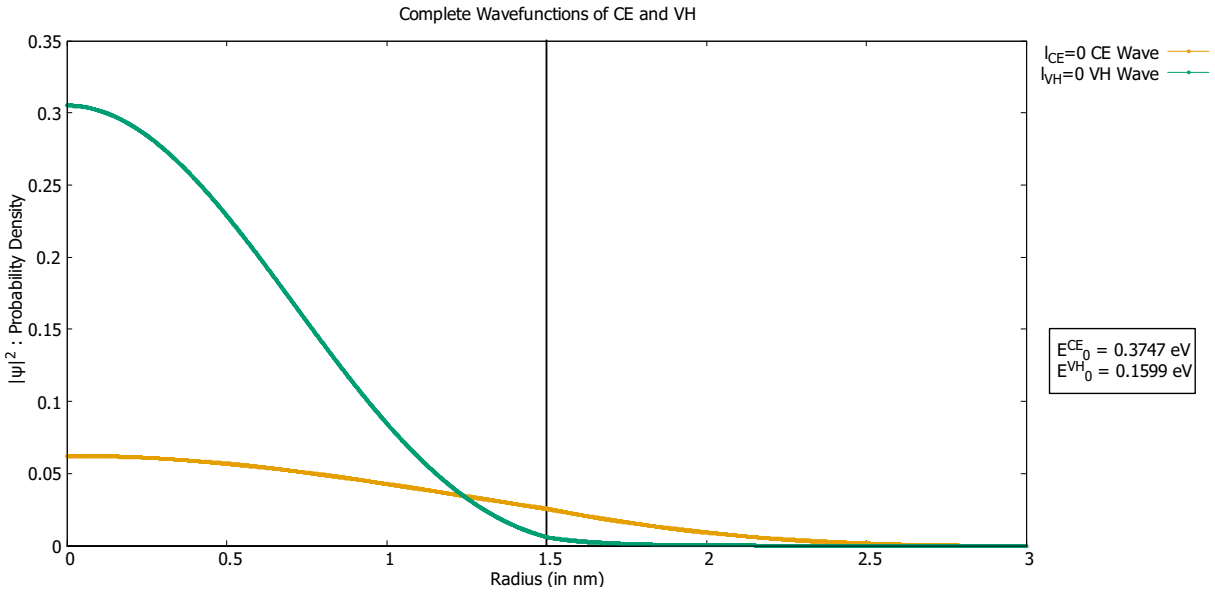

Figure 2.3. Ground state : $n = 0$, $l_{CE} = 0$, $l_{VH} = 0$

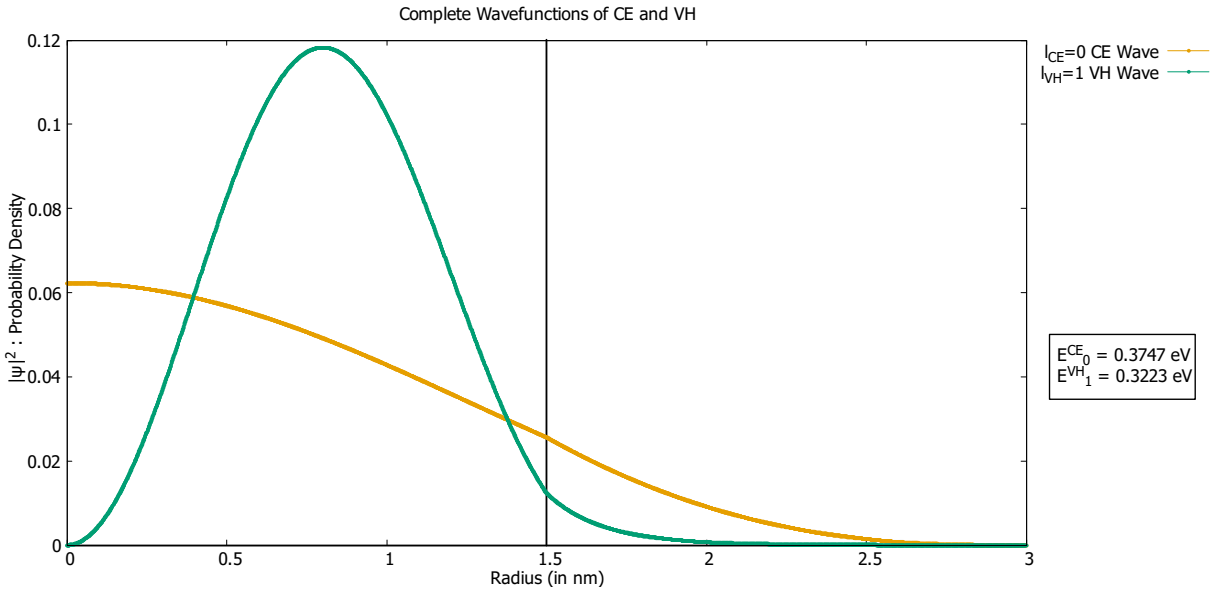

Figure 2.4. First excited state : $n = 1$, $l_{CE} = 0$, $l_{VH} = 1$

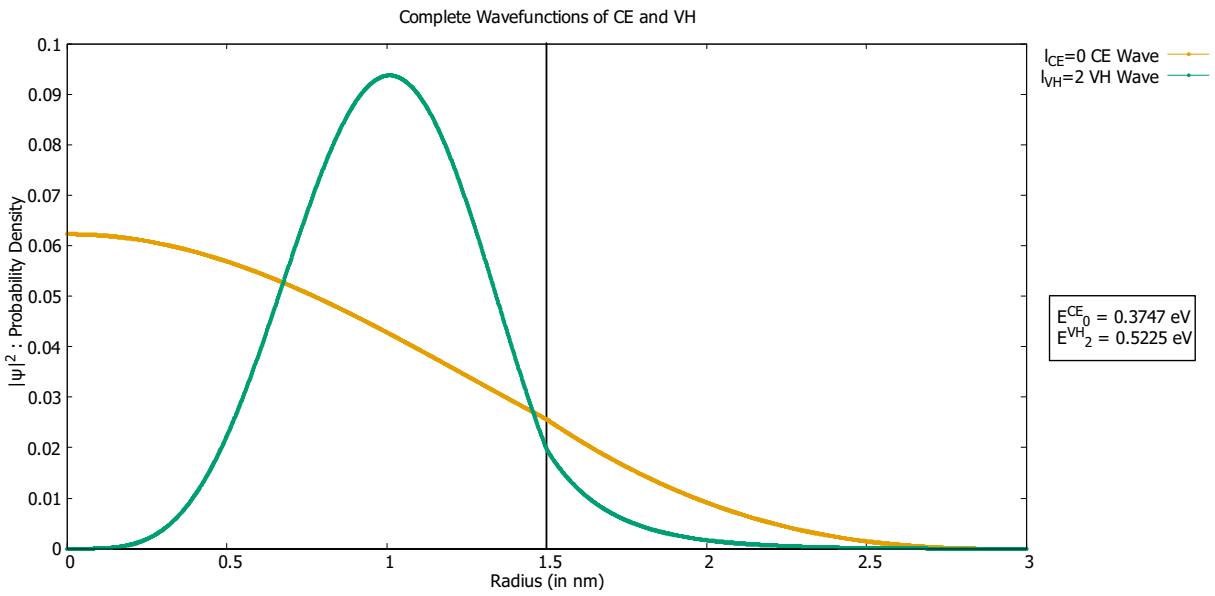

Figure 2.5. Second excited state : $n = 2$, $l_{CE} = 0$, $l_{VH} = 2$

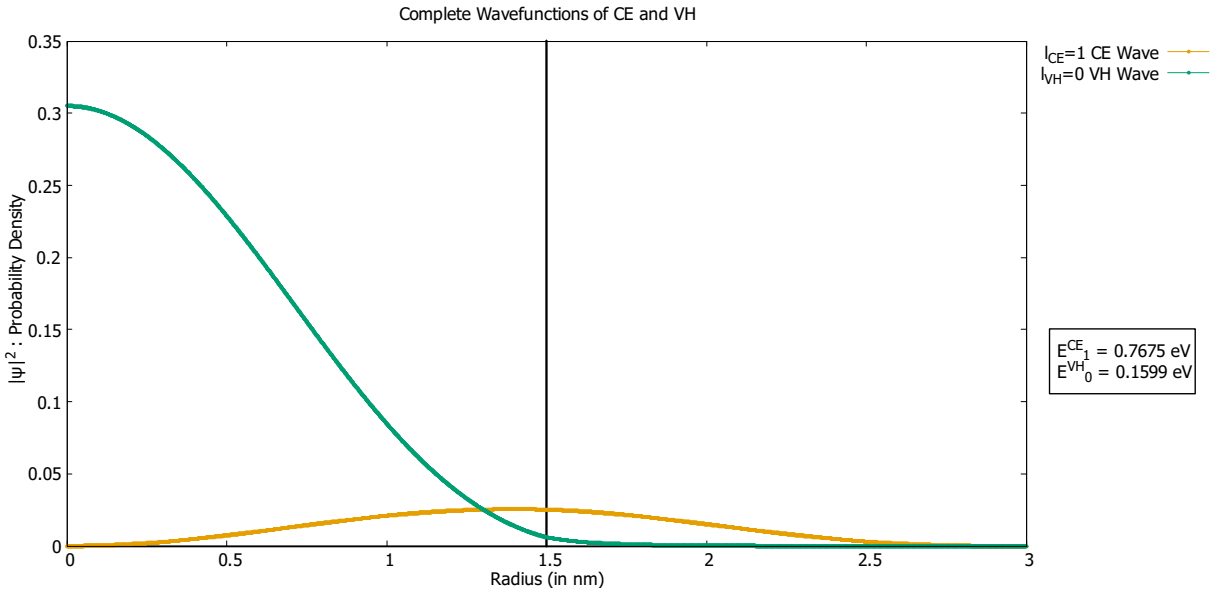

Figure 2.6. Third excited state : $n = 3$, $l_{CE} = 1$, $l_{VH} = 0$

2.3 Corrected Wavefunctions with Coulomb Interaction Perturbation

Once we have the unperturbed wavefunctions due to the free Hamiltonian, we can add the Coulomb interaction in the Hamiltonian as a perturbation, and find the corrections to the energy states and wavefunctions.

$$H_C = -\frac{1}{4\pi\epsilon_e(r)}\frac{e^2}{|\vec{r}_e - \vec{r}_h|}\,. \tag{2.27}$$

The first order correction to the energy and wavefunctions are given as :

$$E_n = E_n^{(0)} + \langle n^{(0)}|H_C|n^{(0)}\rangle\,, \tag{2.28}$$

$$|n\rangle = |n^{(0)}\rangle + \sum_{m\neq n}\frac{\langle m^{(0)}|H_C|n^{(0)}\rangle}{(E_n^{(0)} - E^{(0)})}\,|m^{(0)}\rangle\,. \tag{2.29}$$

The wavefunction for the exciton can be written as a product of the electron and hole wavefunctions.

$$|n\rangle = |\psi_e(r_e)\rangle|\psi_h(r_h)\rangle\,. \tag{2.30}$$

We have found the corrections numerically, and have included the detailed code in the Appendix A.4. The solutions are generated with the assumption that the valence hole is located at $r = 0$ and the conduction electron is free to move. The assumption is strong, but is made to simplify the calculation and get an estimate for the correction due to the Coulomb interaction. Numerically, the first order correction to the ground state of the exciton is given as

$$E_0^{(1)} = -0.034381123\ \text{eV}\,. \tag{2.31}$$

We can then find the corrections to the ground state wavefunction as well. We have terminated the summation at 15 terms, as the contribution from each excited wavefunction kept getting smaller as the energy increased.

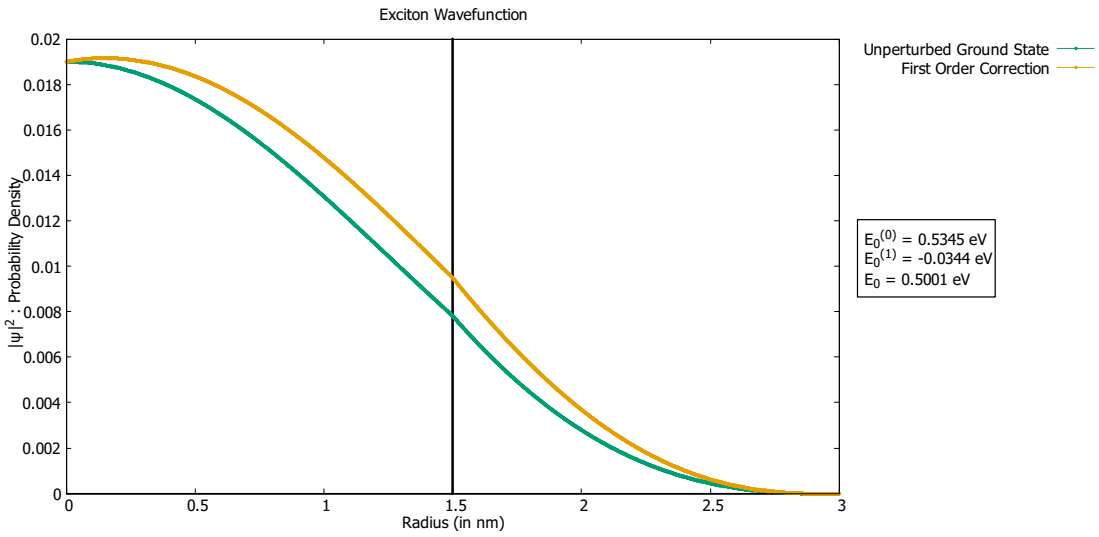

Figure 2.7. Unperturbed and corrected ground state wavefunction

2.4 Interaction of the Quantum Dot with a Weak Electric Field

The trapped quantum dot will be placed in close proximity to the mass-modulated attractor mass. We consider the origin of the coordinate system to lie at the center of the attractor mass and the quantum dot to lie along the z-axis, as shown in figure 2.8. Then the attractor mass is free to move along the xy-plane. The attractor mass will have some spatial variations of electrostatic potentials on its surface, which are known as patch potentials. The patch potentials on the surface of the attractor mass produces an electric field in space which interacts which the quantum dot. We can estimate the polarization of the dot arising due to this weak quantum mechanical interaction. The interaction Hamiltonian can be written as

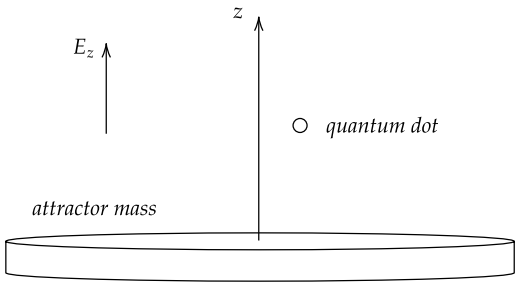

Figure 2.8. Attractor mass with quantum dot

$$\hat{H}_S = -\vec{p}\cdot\vec{F} = e\vec{r}\cdot\vec{F} = eFz\,. \tag{2.32}$$

To qualify as a weak interaction, the Stark energy must be smaller than the separation between adjacent energy levels. The average electric field at 200 nm from the disk due to the patch potentials moving in space will be estimated in the next chapter, but we can use the results here,

$$F_z^{avg} = 1.007 \times 10^3\,\mathrm{Vm}^{-1}\,. \tag{2.33}$$

We use only the electric field along z-axis, as we see in chapter 4.3 that the fluctuations of electric field due to the patch potentials is the largest along the z-axis. We place the quantum dot at a height of $z = 200\,\mathrm{nm}$ above the attractor mass. With this, we get the Stark energy is

$$eFz = 3.227 \times 10^{-23}\mathrm{J}\,.$$

We obtain the difference between the ground state and the first excited state from from the table 2.2 and convert it to Joules as

$$\Delta E_n^{(0)} = 3.154 \times 10^{-20}\mathrm{J}\,.$$

Thus, we see that the Stark energy is much lower than the energy difference between adjacent states, and we can use the weak field effect approximation.

The calculation is done numerically, and the code is included in Appendix A.5. The first order correction to the ground state energy will be zero, due to the parity of the ground state. So, we will look at second order correction due to the perturbation, which is given as

$$E_n^{(2)} = \sum_{k=0,\,k\neq n}^{\infty} \frac{|\langle n|\hat{H}_S|k\rangle|^2}{E_n^{(0)} - E_k^{(0)}}\,. \tag{2.34}$$

We are only concerned with the ground state, so for us the expression will look like

$$E_0^{(2),Stark} = \sum_{k\neq 0}^{\infty} \frac{|\langle n=0|\hat{H}_S|k\rangle|^2}{E_0^{(0),C} - E_k^{(0),C}}\,. \tag{2.35}$$

We will use the corrected wavefunctions from the previous section for this calculation.

As the energy of the state $E_k^{(0)}$ increases, the denominator keeps increasing as well. So the contributing terms as a whole, keep decreasing. We can get an upper limit on the correction, if we replace all denominators with $E_0^{(0)} - E_1^{(0)}$, as

$$E_0^{(2),Stark} < \frac{e^2\,F^2}{E_0^{(0),C} - E_1^{(0),C}} \sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 \,. \tag{2.36}$$

Now focusing on the summation term, we obtain

$$\sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 = \sum_{k} |\langle 0|z|k\rangle|^2 - |\langle 0|z|0\rangle|^2 \,.$$

We know that the expectation value of z for the ground state is zero, which allows us to drop the second term and write

$$\begin{aligned} \implies \sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 &= \sum_{k} \langle 0|z|k\rangle\langle k|z|0\rangle \,, \\ \implies \sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 &= \langle 0|z^2|0\rangle \,. \end{aligned}$$

Using the corrected ground state exciton wavefunction, $|n=0\rangle = \psi_g^{exc}(r)$ from section 2.3 we calculate the correction term as

$$\implies \sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 = \int_0^{2\pi}\int_0^{\pi}\int_{r=0}^{r=b} |\psi_g^{exc}(r)|^2 z^2 r^2 \sin(\theta) dr\, d\theta\, d\phi \,.$$

We also use $z = r\cos(\theta)$ in the above expression,

$$\implies \sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 = 2\pi \int_0^{\pi} \cos^2(\theta)\sin(\theta) d\theta \int_{r=0}^{r=b} |\psi_g^{exc}(r)|^2 r^4 dr \,.$$

Thus we finally obtain

$$\sum_{k\neq 0}^{\infty} |\langle 0|z|k\rangle|^2 = \frac{4\pi}{3} \int_{r=0}^{r=b} |\psi_g^{exc}(r)|^2 r^4 dr \,. \tag{2.37}$$

We then get the upper limit to the second order correction as

$$E_0^{(2),Stark} < \frac{e^2\,F^2}{E_0^{(0),C} - E_1^{(0),C}} \frac{4\pi}{3} \int_{r=0}^{r=b} |\psi_g^{exc}(r)|^2 r^4 dr\,. \tag{2.38}$$

The energy values of the states are the corrected values due to Coulomb interaction.

Once we have an upper limit to the second order correction to the energy due to Stark effect, we can find the upper limit to the quantum polarization of the dot. The dipole polarizability is the rate of change of the dipole moment with respect to the applied electric field. The dipole moment, in turn, is the change in energy with respect to the change in the electric field.

$$\alpha_{dot} = \frac{dp}{dF} = -\frac{d^2 E^{(2),Stark}}{dF^2}\,. \tag{2.39}$$

Thus, we obtain the upper limit on α_{dot} as

$$\boxed{\alpha_{dot} < -\frac{8\pi}{3}\left(\frac{e^2}{E_0^{(0),C} - E_1^{(0),C}}\right) \int_{r=0}^{r=b} |\psi_g^{exc}(r)|^2 r^4 dr\,.} \tag{2.40}$$

The integration is performed numerically and the detailed code is included in Appendix A.5. We obtain the final result as

$$\boxed{\alpha_{dot} < 2.48171631237367 \times 10^{-37}\ \text{C m}^2\ \text{V}^{-1}\,.} \tag{2.41}$$

We thus find that the quantum mechanically obtained upper limit to the polarizability is an order of magnitude smaller than the classical polarizability of the CdSe-CdS quantum dot obtained in equation (2.13). This can be due to the strong assumption made in chapter 2.4 that valence hole is located at $r = 0$.

For further calculations, we have used the classical polarizability of the quantum dot. The classical approximation is valid as the polarizability is a phenomenon exhibited by the complete quantum dot containing several electron-hole pairs, not just a single one. The radius of the quantum dot is also in the order of a few nanometers and classical approximations for its behavior can be made.

3. ELECTRIC FIELD DUE TO CONDUCTING DISK

The electrostatic potential on the surface of a conductor is not a constant quantity, and has spatial variations, which we can call as patch potentials [22]. These patch potentials give rise to a non-uniform electric field which in turn can give rise to a non-zero electrostatic energy between the conductor plate and any particle held in close proximity to it. In this chapter, we will derive and visualize the electric field generated in space due to patch potentials from a conductor plate.

We will first start with a single conducting disk and derive the electric field in space due to it. We will then derive the electrostatic interaction of a quantum dot placed 200 nm away from the conducting disk, as it moves over it along a straight line. This mimics the effect of moving the potential patch underneath the quantum dot.

In Section 3.1, we will first obtain the surface charge density of a conducting disk by approximating it to be a limiting case of an ellipsoid with one axis flattened. We will observe that the charge density tends to blow up the boundary of the disk, but has finite values everywhere else.

Once we have the surface charge density, in Section 3.2 we obtain the analytic expression for potential in space due to the conducting disk. We first start with potential along $z-$axis of the disk, perform a Taylor expansion about $z = a$, where a is the radius of the disk and compare it with a general power series expansion to obtain the coefficients. Then, the potential expression is generalized to all space.

In Section 3.3, we start with the expression for potential in all space and obtain the electric field due to the conducting disk in space. The calculations are all performed in the cylindrical coordinate system, and separate expressions are obtained regions inside and outside the sphere of radius a.

3.1 Surface Charge Density of a Conducting Disk

The first step to obtaining the electric field for a disk is to obtain its surface charge density. To calculate the surface charge density of the conducting disk, we need to consider it as a limiting case of a squashed ellipsoid, as shown in figure 3.1.

Let us consider an ellipsoid of focal length L, semi-major axis length b and semi-minor axis length a. Let distance of any point P from the two focii be r_+ and r_- respectively. Then, the ellipsoid surface containing the point P is the locus of points such that

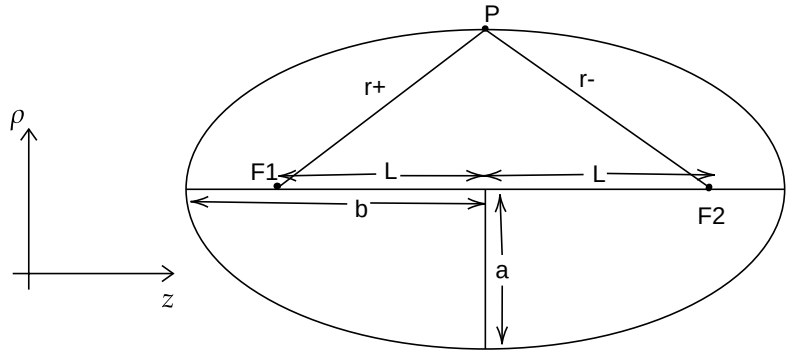

Figure 3.1. Ellipsoid

$$r_+ + r_- = 2b\,. \tag{3.1}$$

The charge density for the ellipsoid can be found the derivative of its surface potential.

$$\sigma = -\epsilon_0 \nabla\varphi \cdot \hat{n}\Big|_S\,. \tag{3.2}$$

To obtain the potential distribution in the ellipsoidal surface, we can solve for the equipotential surface of a line charge distribution.

Let us consider a line charge of length $2L$ and total charge Q. Let us assume it has uniform line charge density λ_{charge}. We can write the potential at any point $P(z,\rho)$ as

$$\varphi(z,\rho) = \frac{\lambda_{charge}}{4\pi\epsilon_0}\int_{-L}^{L}\frac{dz'}{\sqrt{(z'-z)^2+\rho^2}}\,. \tag{3.3}$$

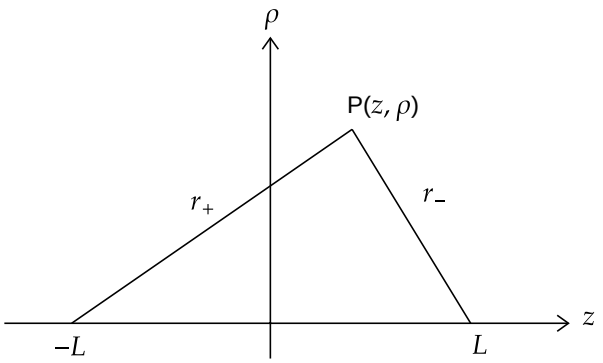

Figure 3.2. Line charge

Solving the above integral we obtain

$$\therefore\ \varphi(z,\rho) = \frac{\lambda_{charge}}{4\pi\epsilon_0} \ln\left[\frac{z+L+\sqrt{(L+z)^2+\rho^2}}{z-L+\sqrt{(L-z)^2+\rho^2}}\right] . \tag{3.4}$$

We can now define a new coordinate system.

$$r_\pm = \sqrt{\rho^2+(L\pm z)^2}\,, \tag{3.5}$$

$$u = \frac{1}{2}(r_+ + r_-)\,, \tag{3.6}$$

$$t = \frac{1}{2}(r_+ - r_-)\,. \tag{3.7}$$

Rewriting the potential (3.4) in the new coordinate system, we obtain

$$\varphi(u,L) = \frac{\lambda_{charge}}{4\pi\epsilon_0} \ln\left[\frac{u+L}{u-L}\right] . \tag{3.8}$$

So, we observe that φ is independent of t. As L is a constant, φ is constant when u is constant. So, equipotential surfaces defined by the above potential are surfaces of constant u, i.e., they are ellipsoids.

Setting $u = b$, we can superimpose a conducting ellipsoid on any of the equipotential surfaces. Now we can use (3.2) and (3.8) to find $\sigma(r_S)$ for the ellipsoid. The detailed derivation is performed in Appendix B.1 and the final expression is obtained as

$$\sigma(z,\rho) = \frac{Q}{4\pi a^2 b}\left(\frac{\rho^2}{a^4} + \frac{z^2}{b^4}\right)^{-1/2} . \tag{3.9}$$

The above equation is valid for any ellipsoid. In the $b \to 0$ limit it collapses to a flat circular disk of radius a. To avoid singularity issues in this limit, we can rewrite (3.9) as

$$\boxed{\sigma(\rho) = \frac{Q}{4\pi a^2}\left[\frac{\rho^2}{a^2}\left(\frac{b^2}{a^2} - 1\right) + 1\right]^{-1/2} .} \tag{3.10}$$

In the $b \to 0$ limit, we obtain

$$\boxed{\sigma(\rho) = \frac{Q}{4\pi a}\frac{1}{\sqrt{a^2 - \rho^2}} .} \tag{3.11}$$

This relationship tells us that at the center of the disk, the charge density is the least, and it steadily increases as we approach the boundary. As $\rho \to a$, the charge density blows up as $\sigma \longrightarrow \infty$.

3.2 Potential due to the Conducting Disk in Space

Now that we have obtained the surface charge density of the disk (3.11), we can calculate the potential in space due to it. We will first focus on potential along the z-axis of the disk, then generalize it to all space.

The circular disk has some radius a and the point P lies along the z-axis. The potential at point P can be written as

$$V(\rho,z) = \frac{1}{4\pi\epsilon_0}\int \frac{\sigma(\rho')\,\rho'\,d\rho'\,d\phi'}{\sqrt{z^2 + \rho'^2}} . \tag{3.12}$$

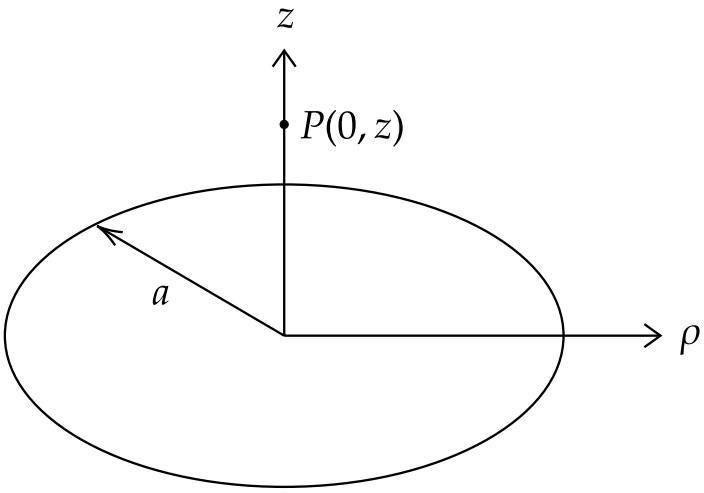

Figure 3.3. Potential along z-axis

Due to azimuthal symmetry, $d\phi'$ integrates out to 2π. The detailed derivation is performed in Appendix B.2. We find the final expression for the potential as

$$\boxed{V(z) = \frac{Q}{8\pi\epsilon_0 a}\left[\frac{\pi}{2} - \tan^{-1}\left(\frac{z}{a}\right)\right] = \frac{Q}{8\pi\epsilon_0 a}\tan^{-1}\left(\frac{a}{z}\right)\,.} \tag{3.13}$$

We can write the most general expression for $V(z)$ as a power series,

$$V(z) = \sum_{l=0}^{\infty}\left[\frac{A_l}{z^{l+1}} + B_l z^l\right]\,. \tag{3.14}$$

For a sphere of radius a centered at the origin, the potential has different expansions inside and outside the sphere,

$$\text{For } z < a\,:\quad V(z) = \sum_{l=0}^{\infty} B_l z^l\;;\quad A_l = 0 \text{ as } z^{-(l+1)} \to \infty\,, \tag{3.15}$$

$$\text{For } z > a\,:\quad V(z) = \sum_{l=0}^{\infty} \frac{A_l}{z^{l+1}}\;;\quad B_l = 0 \text{ as } z^l \to \infty\,. \tag{3.16}$$

We need to use the two different expressions for the potential in (3.13) for the two regions in space.

Case 1 : z $\leq$ a

We use the first expression for the potential in (3.13) as in this case we have $\frac{z}{a} < 1$ as $z < a$. We can then do a Taylor expansion of $\tan^{-1}\left(\frac{z}{a}\right)$ about 0,

$$V(z) = \frac{Q}{8\pi\epsilon_0 a}\left[\frac{\pi}{2} - \frac{z}{a} + \frac{1}{3}\left(\frac{z}{a}\right)^3 - \frac{1}{5}\left(\frac{z}{a}\right)^5 + \cdots\right]. \tag{3.17}$$

Comparing the equations (3.17) and (3.15), we can obtain a general equation for the coefficients B_l for the expansion,

$$\boxed{B_n = \begin{cases} \dfrac{Q}{16\epsilon_0 a} & : \quad n = 0\,, \\ \dfrac{Q}{8\pi\epsilon_0}\dfrac{(-1)^{\frac{n+1}{2}}}{n a^{n+1}} & : \quad n = \text{odd}\,, \\ 0 & : \quad n = \text{even},\ n \neq 0\,. \end{cases}} \tag{3.18}$$

Case 2 : z $\geq$ a

We now use the second expression for the potential in (3.13) as in this case we have $\frac{a}{z} < 1$ as $a < z$. We can then do a Taylor expansion of $\tan^{-1}\left(\frac{a}{z}\right)$ about 0,

$$V(z) = \frac{Q}{8\pi\epsilon_0 a}\left[\frac{a}{z} - \frac{1}{3}\left(\frac{a}{z}\right)^3 + \frac{1}{5}\left(\frac{a}{z}\right)^5 + \cdots\right]. \tag{3.19}$$

Comparing the equations (3.19) and (3.16), we can obtain a general equation for the coefficients A_l for the expansion,

$$\boxed{A_n = \begin{cases} \dfrac{Q}{8\pi\epsilon_0}\dfrac{(-1)^{\frac{n}{2}} a^n}{(n+1)} & : \quad n = \text{even}\,, \\ 0 & : \quad n = \text{odd}\,. \end{cases}} \tag{3.20}$$

Now that we have the expression for potential along z-axis, we can generalize it to all space. The coefficients determined remain the same,

$$\boxed{V(r,\theta) = \sum_{l=0}^{\infty} \left[\frac{A_l}{r^{l+1}} + B_l\, r^l \right] P_l(\cos(\theta))\,.} \tag{3.21}$$

We can also perform a coordinate transformation and write the potential in the cylindrical coordinate system as

$$\boxed{V(\rho,z) = \sum_{l=0}^{\infty} \left\{ \frac{A_l}{(\rho^2+z^2)^{\frac{l+1}{2}}} + B_l(\rho^2+z^2)^{\frac{l}{2}} \right\} P_l\left(\frac{z}{\sqrt{\rho^2+z^2}} \right)\,.} \tag{3.22}$$

3.3 Electric Field due to Conducting Disk in Space

We start with the expression for the potential in the cylindrical coordinate system given in (3.22). The electric field can be calculated separately for regions inside and outside the sphere of radius a.

Case 1 : $\mathbf{r \le a} \implies \mathbf{(\rho^2+z^2)^{1/2} \le a}$

For inside the sphere, the potential is written as

$$V^{in}(\rho,z) = \sum_{l=0}^{\infty} B_l\, (\rho^2+z^2)^{l/2}\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}} \right)\,. \tag{3.23}$$

The electric field inside the sphere can be obtained by taking the gradient of the potential,

$$\vec{E}^{in}(\rho,z) = -\vec{\nabla} V^{in}(\rho,z)\,,$$

$$\therefore\ \vec{E}^{in}(\rho,z) = -\Bigg[\underbrace{\frac{\partial V^{in}}{\partial \rho}(\rho,z)\,\hat{\rho}}_{E^{in}_{\rho}} + \frac{1}{\rho}\cancelto{0}{\frac{\partial V^{in}}{\partial \phi}(\rho,z)\hat{\phi}} + \underbrace{\frac{\partial V^{in}}{\partial z}(\rho,z)\hat{z}}_{E^{in}_{z}} \Bigg]\,.$$

The ρ and z components can be solved separately for ease of calculation. The complete derivation for the same can be found in Appendix B.3. The final expressions are obtained as

$$E_\rho^{in} = \sum_{l=0}^{\infty} B_l\, l\, (\rho^2+z^2)^{\frac{l}{2}-1} \left\{ \frac{z}{\rho}\sqrt{z^2+\rho^2}\, P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - \frac{(\rho^2+z^2)}{\rho}\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}, \tag{3.24}$$

$$E_z^{in} = \sum_{l=0}^{\infty} \left\{ -B_l\, l\, (\rho^2+z^2)^{\frac{l}{2}-1} \sqrt{z^2+\rho^2}\, P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}. \tag{3.25}$$

Case 2 : $\mathbf{r \geq a} \implies \mathbf{(\rho^2+z^2)^{1/2} \geq a}$

The potential outside the sphere can be written as

$$V^{out}(\rho, z) = \sum_{l=0}^{\infty} \frac{A_l}{(\rho^2+z^2)^{\frac{l+1}{2}}}\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right). \tag{3.26}$$

We obtain the electric field by taking the gradient of the potential, similar to the previous case. The detailed derivation can be found in Appendix B.3. The final results are given as

$$E_\rho^{out} = \sum_{l=0}^{\infty} A_l\, (\rho^2+z^2)^{-\frac{l+3}{2}} \left[\frac{l(\rho^2-z^2)+\rho^2}{\rho}\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + l\,\frac{z}{\rho}\sqrt{z^2+\rho^2}\, P_{l-1}\left(\frac{z}{z^2+\rho^2}\right)\right], \tag{3.27}$$

$$E_z^{out} = \sum_{l=0}^{\infty} A_l\, (\rho^2+z^2)^{-\frac{l+3}{2}} \left[z(2l+1)\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - l\,\sqrt{z^2+\rho^2}\, P_{l-1}\left(\frac{z}{z^2+\rho^2}\right)\right]. \tag{3.28}$$

4. ELECTROSTATIC INTERACTION OF A QUANTUM DOT WITH MOVING PATCH POTENTIALS

Now that we have the electric field in space due to a single conducting disk, we can numerically solve for electric field in space for multiple overlapping disks. For our setup, we have a mass modulated attractor mass placed approximately 200 nm away from the quantum dot. The attractor mass would form one of the mirrors for the cavity which will be movable, and is expected to have patch potentials on the surface due to its metallic nature.

As a consequence of the patch potentials, we expect to see a non-negligible electrostatic interaction between the quantum dot and the attractor mass, which would be time varying, or moving with respect to the dot. We will start by estimating the electrostatic interaction energy of the quantum dot with one moving patch, and then obtain the results for higher number of moving patches and multiple possible trajectories.

In Section 4.1 we simulate the patch potentials and attempt to fill up the attractor mass surface of a given size with randomly generated conducting disks with varying sizes and potentials distributed in space. We find that using $N = 3000$ disks gives us sufficient spatial coverage.

In Section 4.2 we use the attractor mass surfaces generated for various number of conducting disks N, and obtain the electrostatic interaction energy of the quantum dot with the surface. For these simulations, we assume the quantum dot to be placed 200 nm above the surface, and moving on a straight line. We look into the variation of electrostatic interaction for a limited number of trajectories for varying number of conductor disks.

In Section 4.3, we only consider the attractor mass surface with $N = 3000$ conducting disks. We then consider closely spaced trajectories, constant X and constant Y lines, spaced at every 0.2 μm from 0 μm to 100 μm along both x- and y-axes. The electric field experienced at any spatial point by the dot is a superposition of the electric fields due to all 3000 disks making up the surface. This gives us the complete mapping of the electrostatic interaction energy between the quantum dot and attractor mass in space.

4.1 Simulating Patch Potentials

We will first start with simulating a conductor plate of size $100\,\mu$m $\times 100\,\mu$m composed of conducting disks of varying radii and base potential values. Assuming we have a gold plating on our conductor plate, we take the following steps to generate the surface [22] :

1. The radius (a) of each disk is chosen from a Gaussian distribution with $a_{avg} = 3\,\mu$m and standard deviation$= 1\,\mu$m, motivated by [22].
2. The x- and y-coordinate of the center (x, y) of each circle is chosen randomly from a uniform distribution between $0\,\mu$m to $100\,\mu$m, respectively.
3. The z-coordinate of all disks is 0.
4. The potential (V_0) for each disk is chosen from a Gaussian distribution with $V_0^{avg} = 0$ V and standard deviation$= 10^{-3}$ V, motivated by [22].

The charge for each patch can be related to the voltage by using equation (3.23). We can evaluate the potential at $z = 0$, which should give us V_0. Thus, we obtain

$$\boxed{Q = 16\epsilon_0 a V_0\,.} \tag{4.1}$$

The goal of generating these patch potentials is to fill the $100\,\mu$m $\times 100\,\mu$m conductor plate. The complete code is provided in Appendix C.1. The results for $N = 1$, 5, 100, 1000, 3000, 5000 are included in figure 4.1.

We find that using $N = 3000$ disks gives us sufficient coverage, and we use the same for the complete analysis of electrostatic energies. We do not use the $N = 5000$ disks surface, even though it provides complete coverage, in favor of better computational speed.

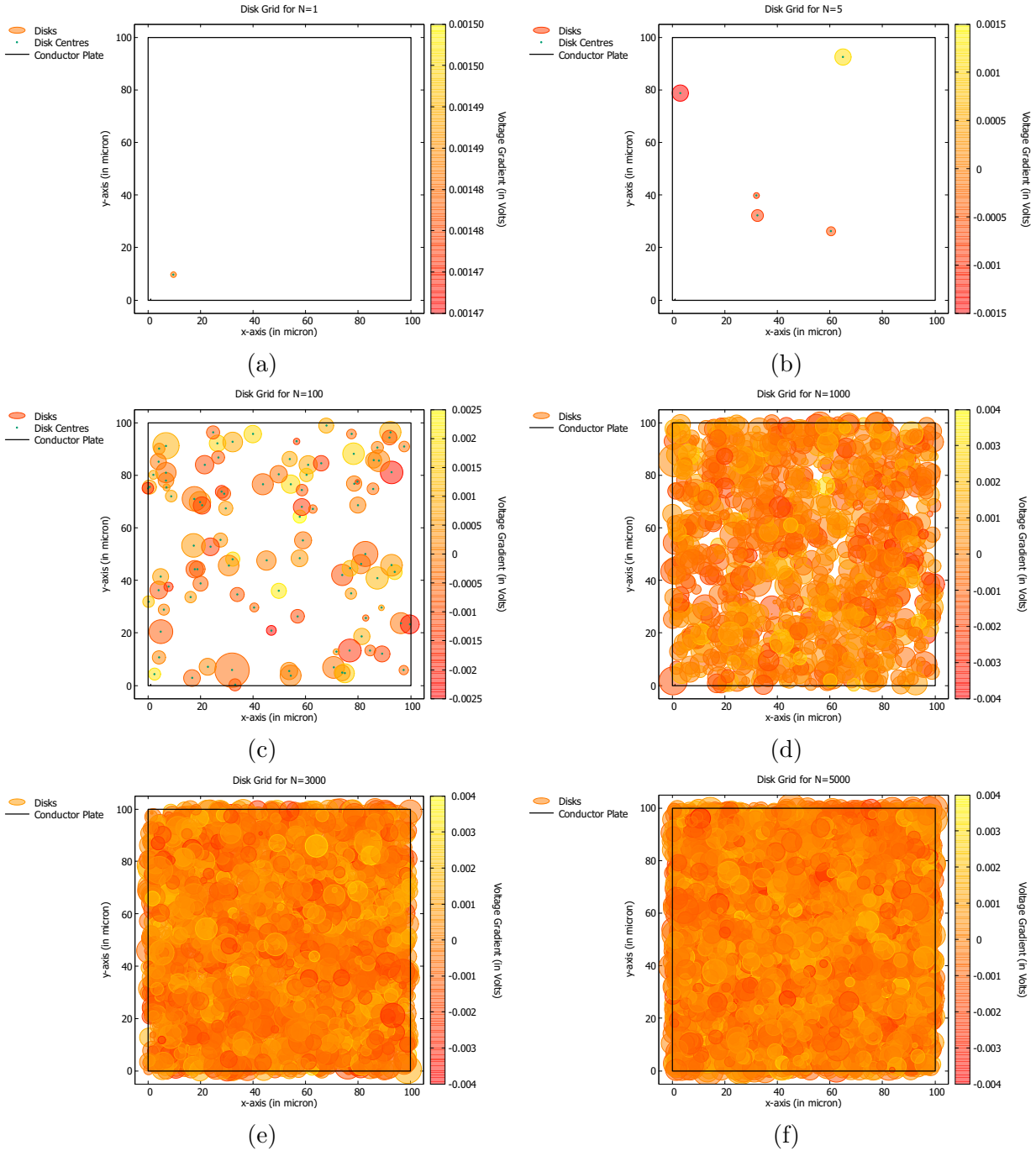

Figure 4.1. Coverage for (a) N=1, (b) N=5, (c) N=100, (d) N=1000, (e) N=3000 and (f) N=5000 randomly generated disks

4.2 Electrostatic Interaction with Moving Patches

The electrostatic interaction energy of a polarizable particle with an electric field in space is given as

$$\boxed{U = \alpha |\vec{E}|^2 \,.} \tag{4.2}$$

α is the polarizability of the particle. The net electric field at any spatial point due to the patch potentials can be obtained by taking the vector sum of the electric fields due to each patch.

We can mimic the effect of the attractor mass moving by considering the quantum dot to be moving in defined trajectories over it. The electrostatic energy fluctuates as the dot moves and the solution needs to be found numerically due to its complexity. The general algorithm followed is outlined below. The complete code can be found in Appendix C.2.

1. Generate the attractor mass surface as a superposition of conducting disks. This has been performed in the previous section.
2. Define the trajectory of the dot over the attractor mass and parametrize it. This forms the outermost loop of the code.
3. For every point (x, y, z_{dot}) in the chosen/defined trajectory, the electric field due to every disk needs to be evaluated. This needs to be done in a few steps.

   i.) For each disk, identify if the point (x, y, z_{dot}) lies inside or outside the sphere defined by it, as that decides which expression for the electric field to use, i.e.,

   $$\sqrt{(x - x_{disk})^2 + (y - y_{disk})^2 + z_{dot}^2} \geq a_{disk} \implies \text{dot outside sphere} \implies E_\rho^{out} \text{ and } E_z^{out} \,,$$
   $$\sqrt{(x - x_{disk})^2 + (y - y_{disk})^2 + z_{dot}^2} \leq a_{disk} \implies \text{dot inside sphere} \implies E_\rho^{in} \text{ and } E_z^{in} \,.$$

   ii.) Once we know which expressions to use, an estimation needs to be obtained for the infinite series. The series $E_{\rho/z}^{in/out}$ are assumed to converge when the higher terms change the value of $E_{\rho/z}^{in/out}$ by less than 0.01%. The converged series gives us the ρ- and z-electric fields as $E_{\rho,disk}$ and $E_{z,disk}$ respectively.

iii.) From the calculated $E_{\rho,disk}$, obtain the $E_{x,disk}$ and $E_{y,disk}$ at that point (x, y). $E_{z,disk}$ obtained is used as is.

$$E_{x,disk} = \frac{x - x_{disk}}{\sqrt{(x - x_{disk})^2 + (y - y_{disk})^2}} E_{\rho,disk}\,, \tag{4.3}$$

$$E_{y,disk} = \frac{y - y_{disk}}{\sqrt{(x - x_{disk})^2 + (y - y_{disk})^2}} E_{\rho,disk}\,. \tag{4.4}$$

4. Repeat step 3 for all disks in the attractor mass surface. The total electric field along each axis at point (x, y, z_{dot}) is obtained by summing them all up.

$$E^{tot}_{x/y/z}(x, y, z_{dot}) = \sum_{all\ disks} E_{x/y/z,disk}\,. \tag{4.5}$$

5. Calculate the electrostatic energy at point (x, y, z_{dot}).

$$U(x, y, z_{dot}) = \alpha \left[(E^{tot}_x)^2 + (E^{tot}_y)^2 + (E^{tot}_z)^2\right]\,. \tag{4.6}$$

6. Repeat steps 3, 4 and 5 for every point on the trajectory.
7. Repeat steps 2-6 for multiple chosen trajectories at desired spacing to generate a map of electrostatic energy at all points above the attractor mass.

In this section, we will be including the results for a limited number of trajectories for a varying number of conductor disks to understand how the electrostatic energy varies in space.

Case 1 : N=1 Disk

We will start with generating one disk, and obtaining the electrostatic energy plot as the quantum dot moves in a straight line across it.

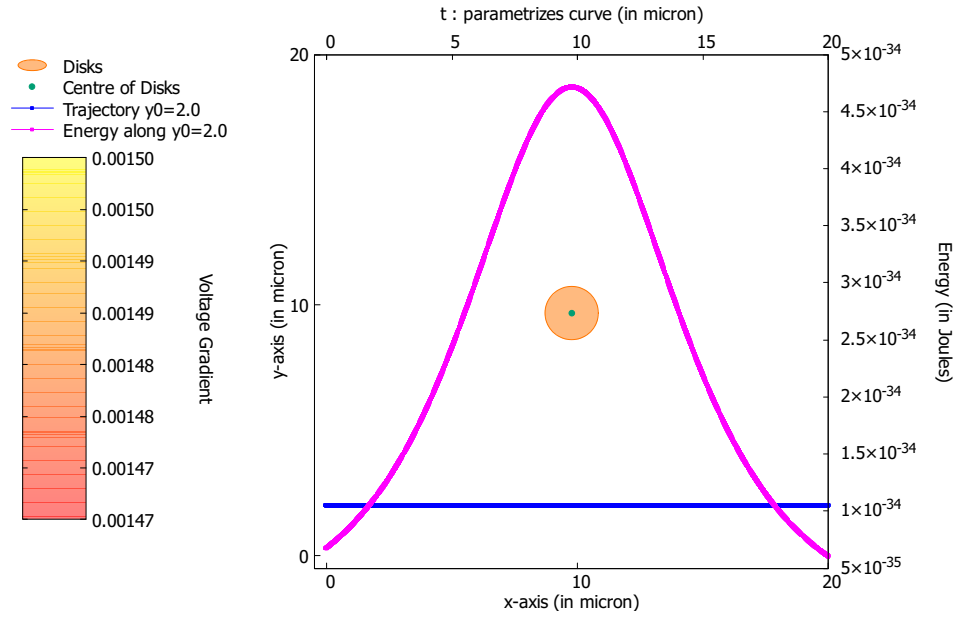

Figure 4.2. Energy for $y_0 = 2.0$ trajectory with $N = 1$ disk

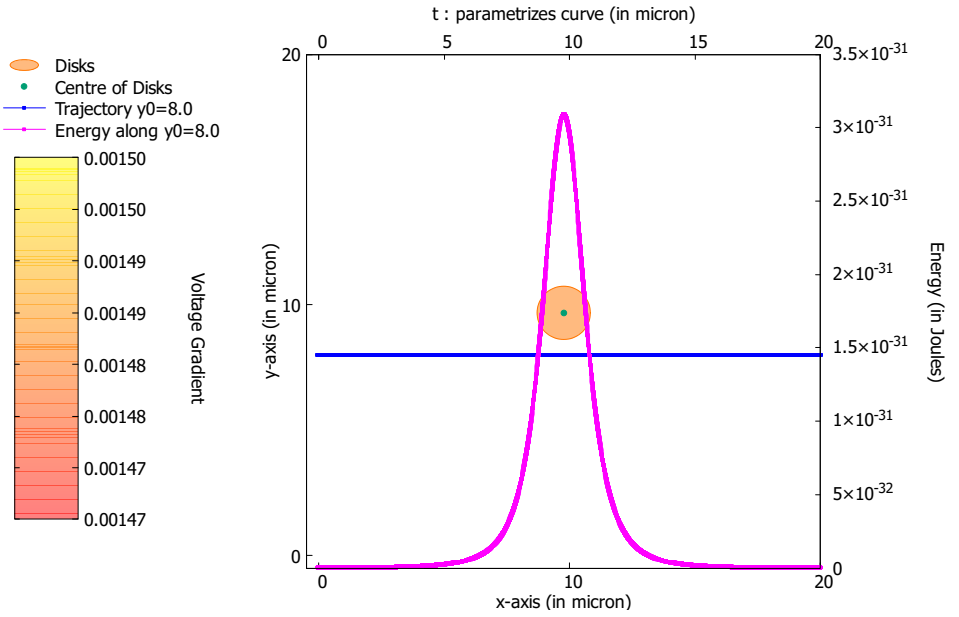

Figure 4.3. Energy for $y_0 = 8.0$ trajectory with $N = 1$ disk

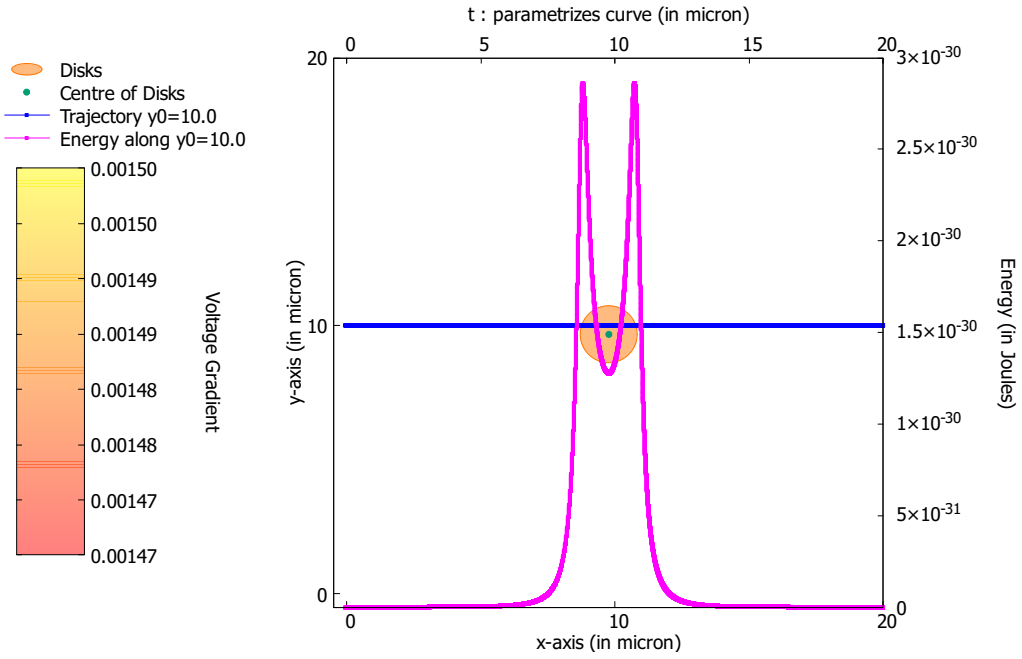

Figure 4.4. Energy for $y_0 = 10.0$ trajectory with $N = 1$ disk

Here we have considered constant y-lines for our trajectories. We observe in figures 4.2, 4.3 and 4.4 that the electrostatic energy peaks around the edges of the patch potential. We expect this behavior as the surface charge density of a patch tends to infinity at the boundary, as illustrated in equation (2.24). For all the plots, the dot remains at a height of $z_{dot} = 200\,\text{nm}$ above the patch potential surfaces.

Case 2 : N=2 Disks

As the number of disks increase, the dependence of energy also becomes more and more complicated. For $N = 2$ disks, we see illustrated in figures 4.5 and 4.6, a dependence similar to $N = 1$ when the trajectory is closer to one of the disks, but it changes for trajectories going between the two disks.

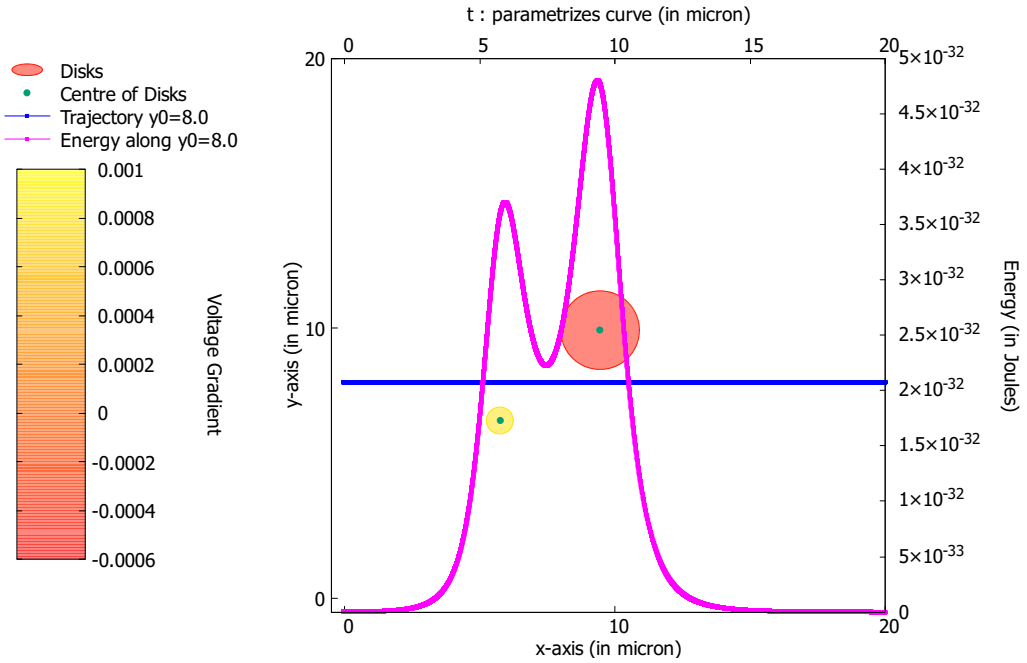

Figure 4.5. Energy for $y_0 = 8.0$ trajectory with $N = 2$ disks

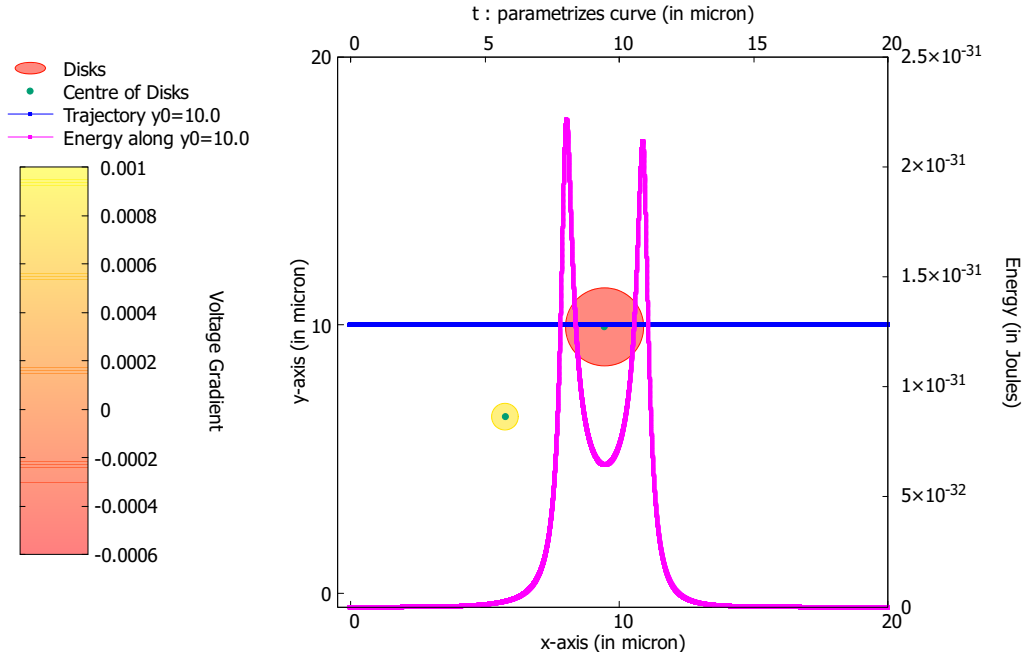

Figure 4.6. Energy for $y_0 = 10.0$ trajectory with $N = 2$ disks

Case 3 : N=5 Disks

We now have 5 disks sparsely distributed in space, illustrated in figures 4.7 and 4.8.

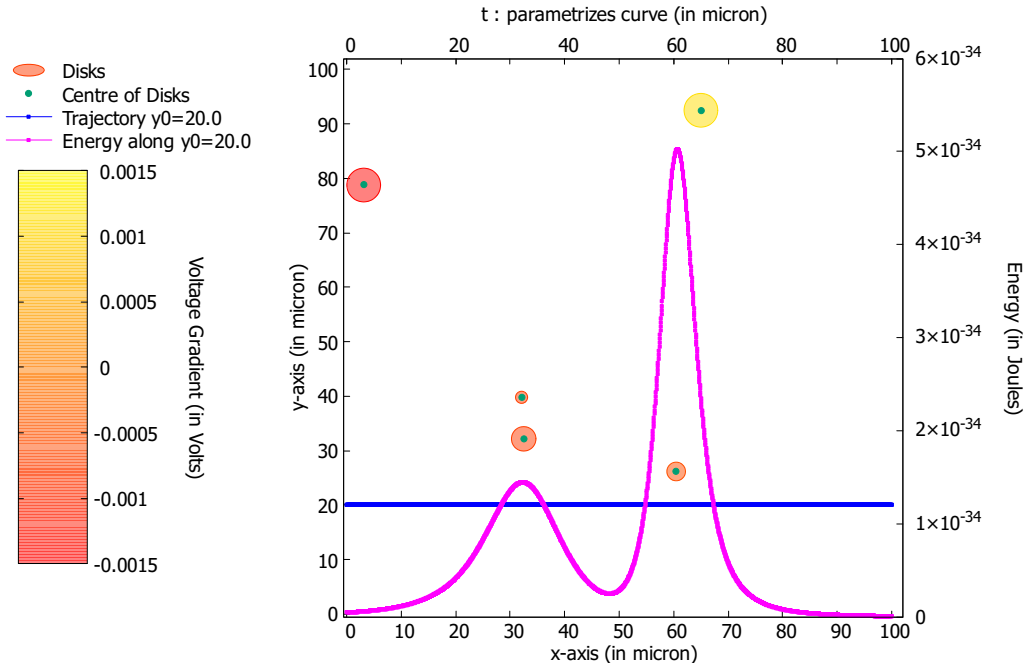

Figure 4.7. Energy for $y_0 = 20.0$ trajectory with $N = 5$ disks

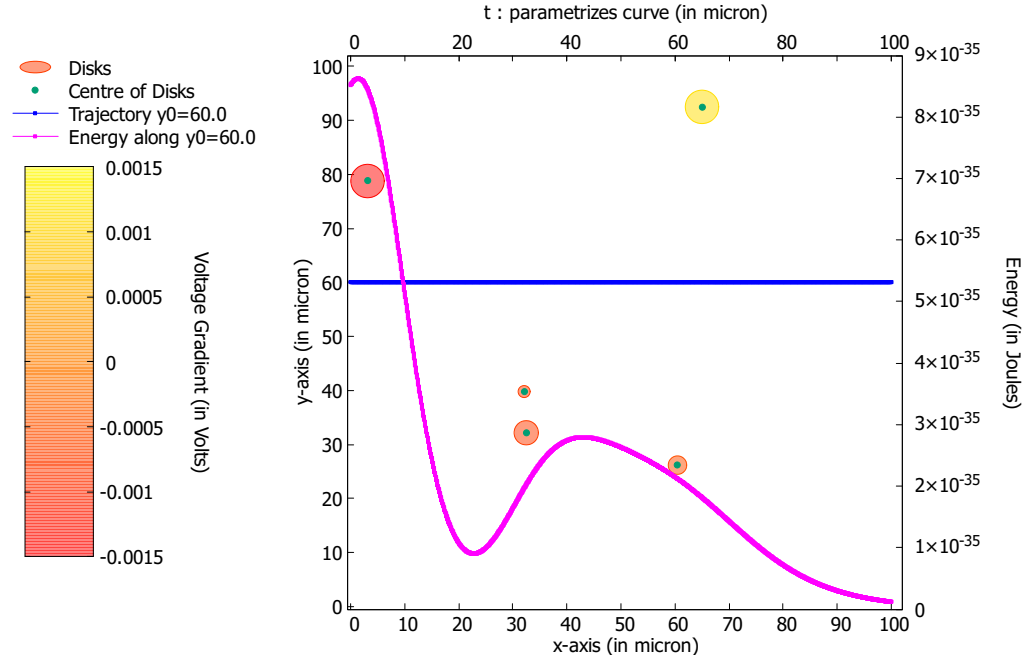

Figure 4.8. Energy for $y_0 = 60.0$ trajectory with $N = 5$ disks

Case 4 : N=100 Disks

On increasing the number of disks, we start seeing the fluctuations more prominently, illustrated in figures 4.9, 4.10, 4.11 and 4.12.

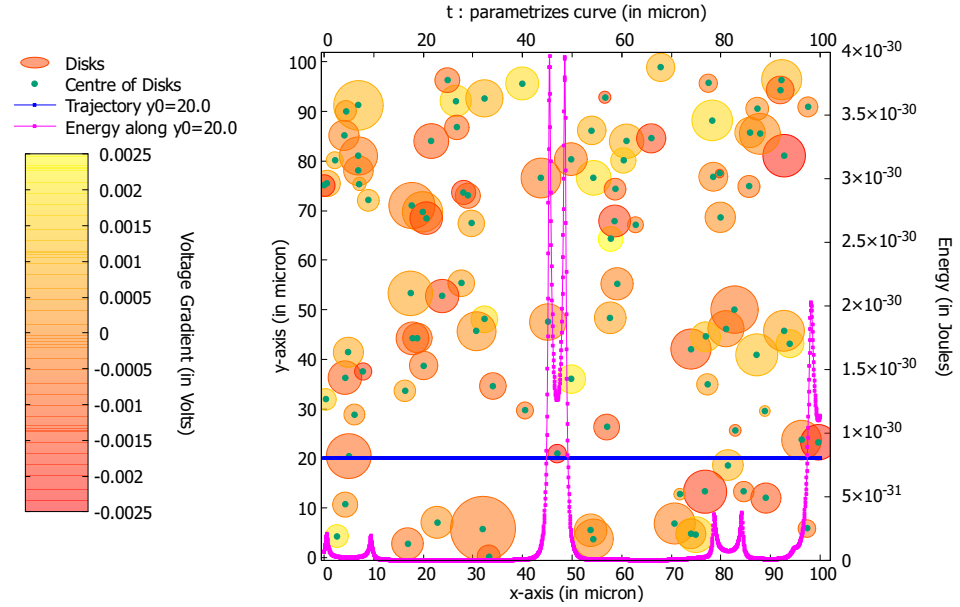

Figure 4.9. Energy for $y_0 = 20.0$ trajectory with $N = 100$ disks

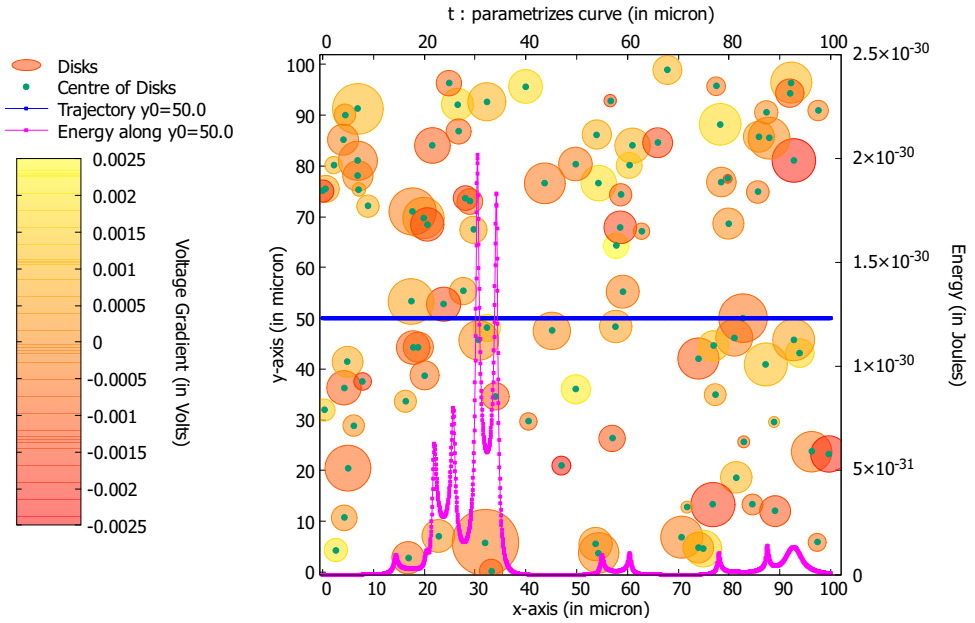

Figure 4.10. Energy for $y_0 = 50.0$ trajectory with $N = 100$ disks

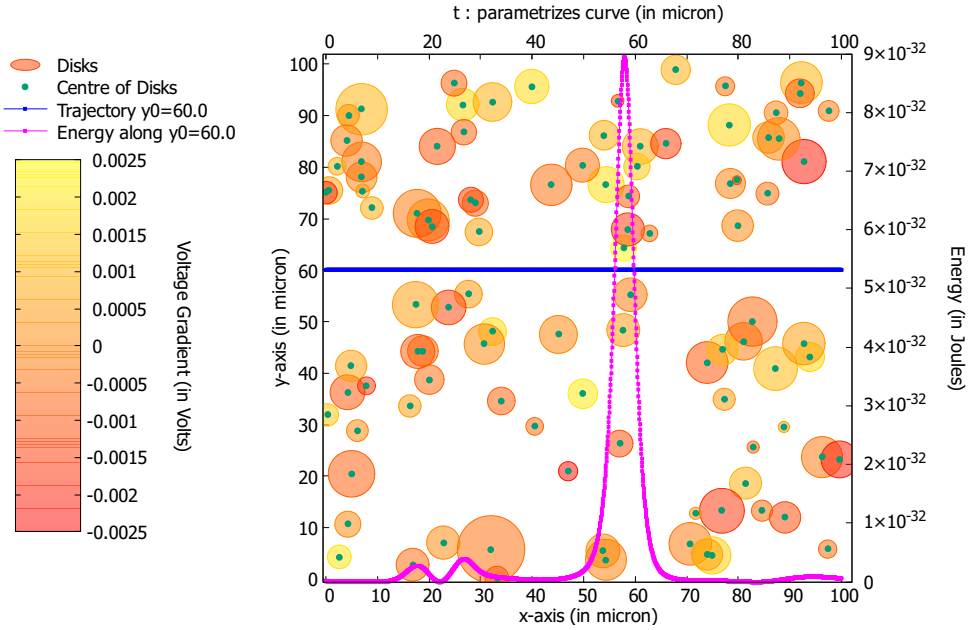

Figure 4.11. Energy for $y_0 = 60.0$ trajectory with $N = 100$ disks

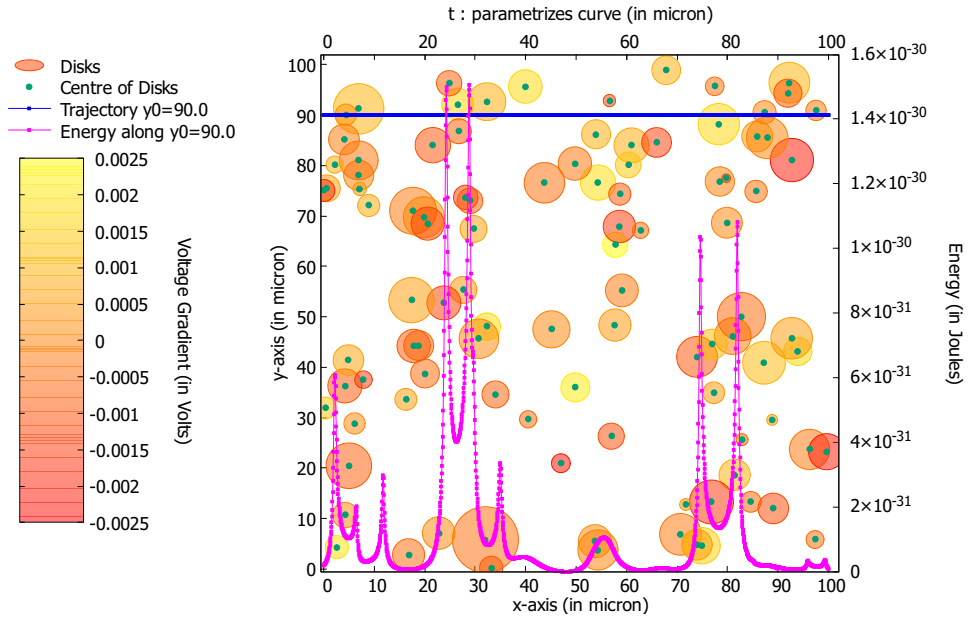

Figure 4.12. Energy for $y_0 = 90.0$ trajectory with $N = 100$ disks

4.3 Mapping of Electric Fields and Electrostatic Energy for N=3000 Disks

In this section, we use the simulation of $N = 3000$ randomly generated conducting disks in superposition to produce the surface of the attractor mass, as illustrated in figure 4.13.

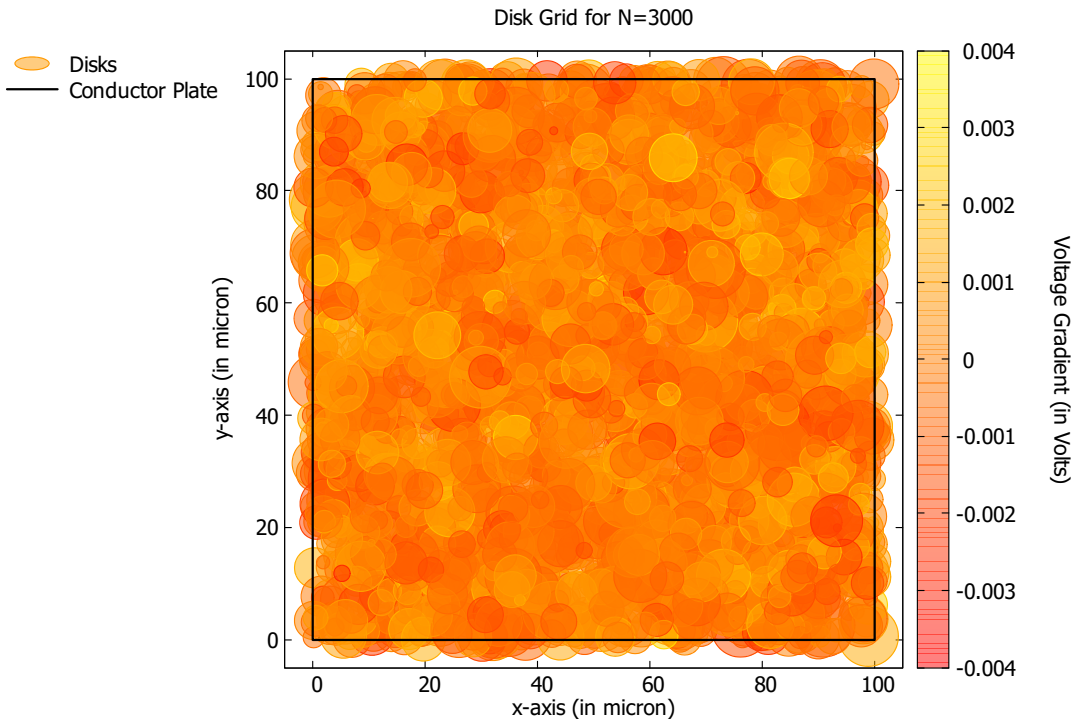

Figure 4.13. Attractor mass surface with N=3000 patch potentials

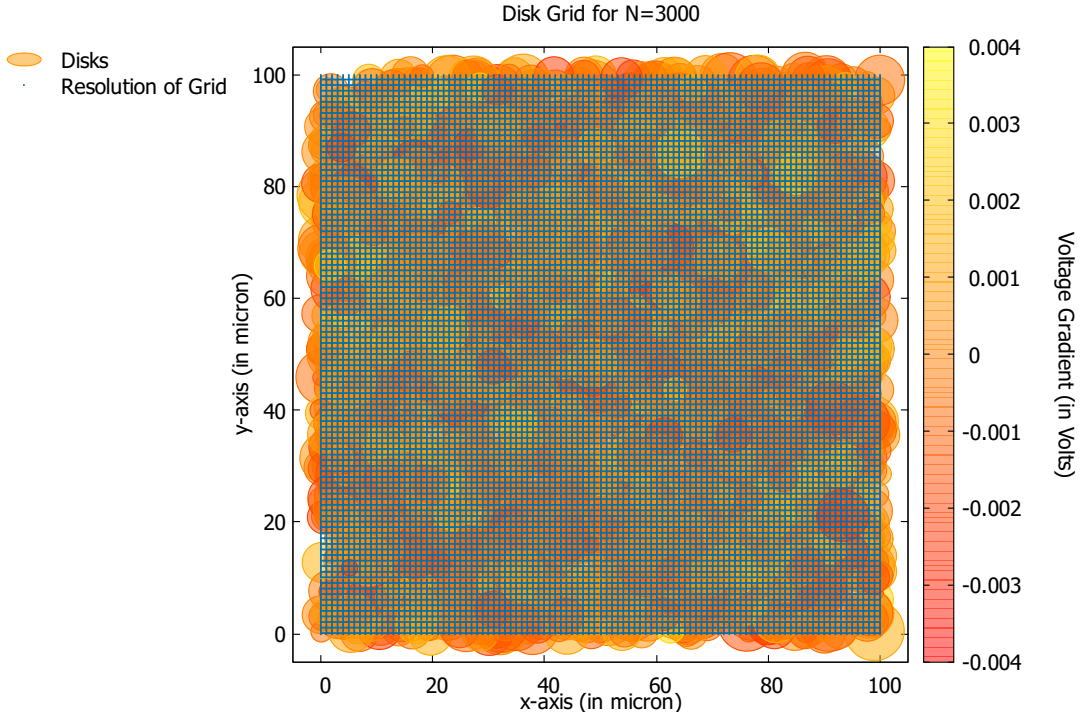

Figure 4.14. Grid produced by trajectories above attractor mass

Once we have the attractor mass surface, we consider closely spaced trajectories of constant X and Y lines, generated at spacings of 0.2 μm from 0 μm to 100 μm, shown in figure 4.14. The grid generated by trajectory maps the variation of electric fields and electrostatic energies over the entire surface of the conductor plate, at a height of $z_{dot} = 200$ nm above it.

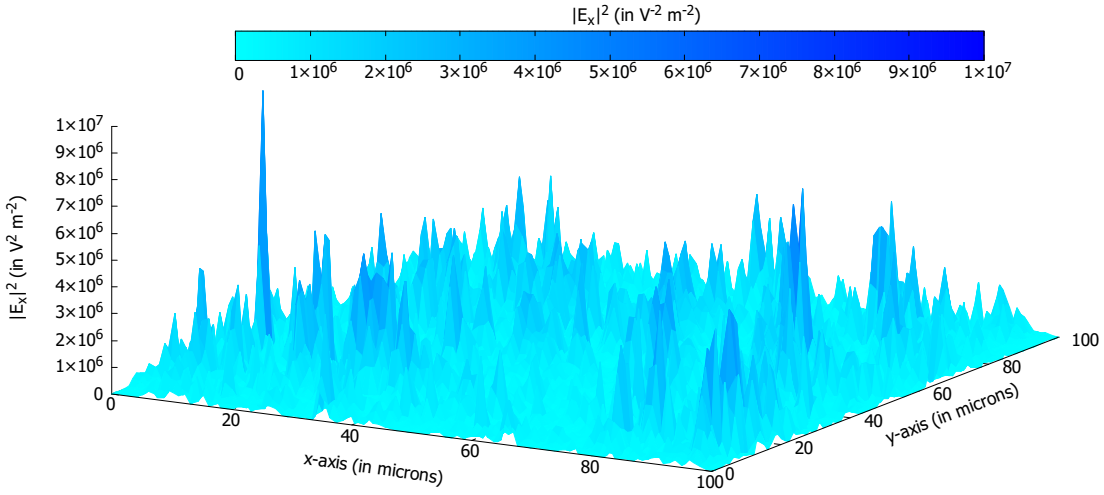

Figure 4.15. Variation of $|E_x|^2$ over attractor mass

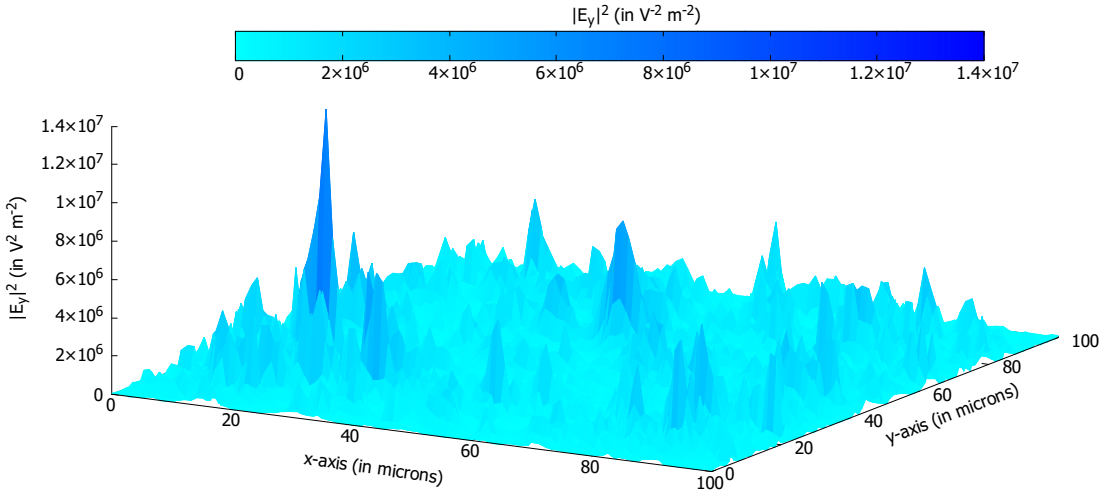

Figure 4.16. Variation of $|E_y|^2$ over attractor mass

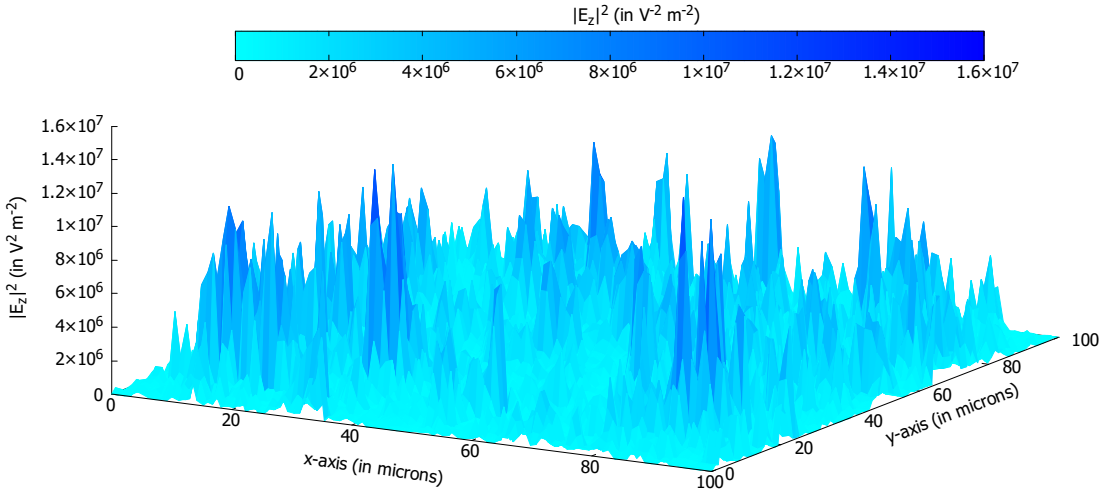

Figure 4.17. Variation of $|E_z|^2$ over attractor mass

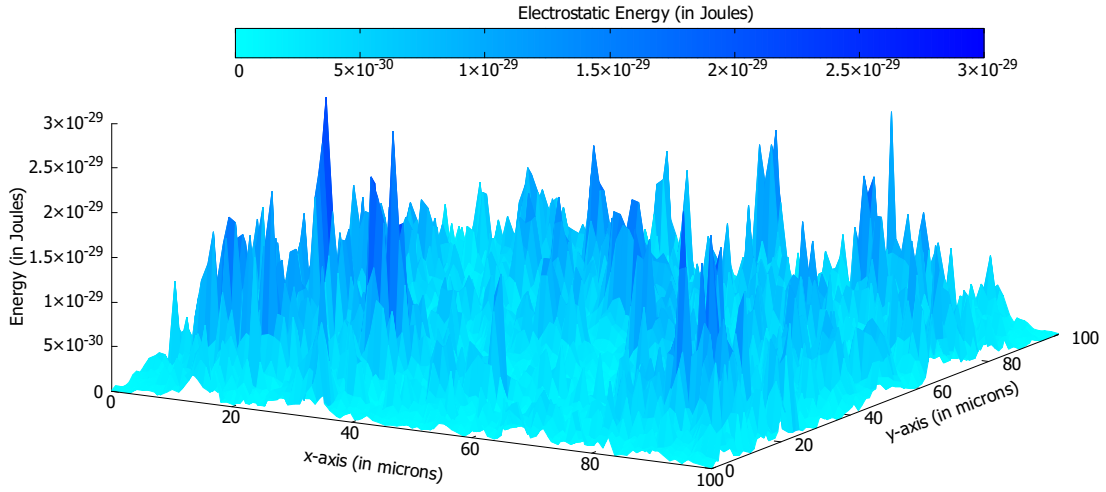

Figure 4.18. Variation of electrostatic energy over attractor mass

The fluctuations of electric fields along x-, y- and z-directions, along with the variation in electrostatic potential energy of the system at every point above the surface of the conductor, obtained numerically, is illustrated in figures 4.15, 4.16, 4.17 and 4.18, respectively. We observe that the fluctuations in $|E_z|^2$ are the largest, compared to $|E_x|^2$ and $|E_y|^2$. We can

perform statistical analysis on the mod-square of the electric fields to obtain the average and standard deviation values.

$$\text{Average value of } |E_x|^2 \text{ over all trajectories} = 474210.62365\text{V}^2\,\text{m}^{-2}\,, \tag{4.7}$$

$$\text{Standard Deviation of } |E_x|^2 \text{ over all trajectories} = 730011.950865\text{V}^2\,\text{m}^{-2}\,, \tag{4.8}$$

$$\text{Average value of } |E_y|^2 \text{ over all trajectories} = 467177.47843\text{V}^2\,\text{m}^{-2}\,, \tag{4.9}$$

$$\text{Standard Deviation of } |E_y|^2 \text{ over all trajectories} = 731587.38545\text{V}^2\,\text{m}^{-2}\,, \tag{4.10}$$

$$\text{Average value of } |E_z|^2 \text{ over all trajectories} = 966950.97306\text{V}^2\,\text{m}^{-2}\,, \tag{4.11}$$

$$\text{Standard Deviation of } |E_z|^2 \text{ over all trajectories} = 1503264.05289\text{V}^2\,\text{m}^{-2}\,, \tag{4.12}$$

The electric field squared along the z-direction is the strongest compared to the x- and y-directions. It makes sense as the electric fields along the z-axis get all added up, while the electric fields along x- and y-directions can have some contributions canceled out.

5. VARIATION IN TRAP POTENTIAL PARAMETERS

Optical trapping is an experimental technique in which highly focused laser beams are used to trap or levitate particles by counteracting the gravitational force using optical forces. The electric trapping potential, which has a Gaussian intensity profile, can be approximated to a simple harmonic potential about the focal point. To observe the quantum mechanical nature of the harmonic oscillator, the trapped particle must be cooled down to an appropriate temperature. But even under ideal temperature conditions, the discrete energy levels of the oscillator obtain a spread due to interactions with other components of the experimental setup or the inherent noise of a laser. In this chapter, we will be summarizing the effect of two major noise contributions which deform the harmonic oscillator levels.

In Section 5.1, we will be estimating the spread in the energy levels due to presence of a weak electric field in space. We are interested in this scenario as our experimental setup includes an optically trapped quantum dot in close proximity to a mass-modulated attractor mass which would have patch potentials on its surface due to its metallic nature. The electric field produced due to these patch potentials produces a non-negligible correction to the energy levels.

In Section 5.2, we will consider the shot noise of the trapping laser beam, which is a property of the electromagnetic field itself and cannot be ignored. The shot noise of the laser affects the power of the laser, which in turn affects the spring constant of the trap potential. The spring constant of the trap potential directly affects the separation of energy levels of the harmonic oscillator and causes a spread in its levels.

5.1 Dispersion of Energy Levels due to Electric Field

The trapping potential of the laser can be approximated with a 3-dimensional harmonic oscillator, about the focal point of the trap, where Ω represents the frequency of the oscillator.

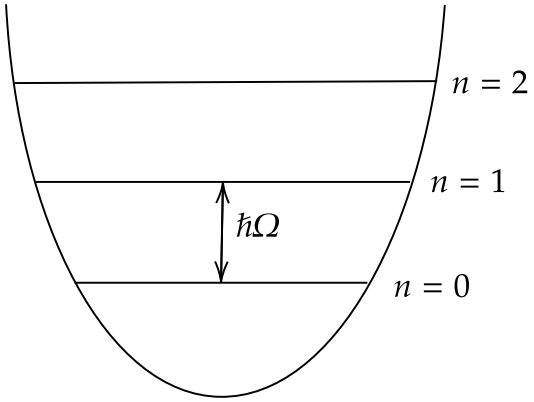

Figure 5.1. Harmonic oscillator potential

The energy levels of a harmonic oscillator can be written as

$$E_n = \hbar\Omega\left(n + \frac{1}{2}\right) . \tag{5.1}$$

We observed in the previous chapter 4.3 that the fluctuations of electric field was the strongest along the z-direction. Also, we know from the expression of the spring constant of the harmonic oscillator trap in equation (1.3) that the value of k_{sp} is different along z-direction than along x- or y-direction. Thus, to obtain an estimate on the maximum dispersion of the energy levels, we can consider the electric field only along the z-direction. This reduces the problem to a one-dimensional harmonic oscillator along the z-axis. Then, the perturbation to the oscillator Hamiltonian can be added as

$$H_1 = \alpha_{dot}|E_z|^2 . \tag{5.2}$$

Here α_{dot} is the polarizability of the quantum dot and E_z is the electric field along the z-axis due to the patch potentials derived in the previous chapter.

Performing statistical analysis on the electric field data obtained previously, we obtain the average and standard deviation of the electric field square in equations (4.11, 4.12). The

spread in the energy levels can be characterized using the spread in the electric field, i.e., standard deviation of $|E_z|^2$, as

$$\Delta E = \alpha_{dot}\sigma = \alpha_{dot}|E_z|^2 \,. \tag{5.3}$$

The spread in the energy values also gives us the spread in the frequency of the oscillator. Plugging in the value of α_{dot} from equation (2.13) and σ from equation (4.12), we obtain

$$\Delta\Omega = \frac{\Delta E}{\hbar} = \frac{2.417 \times 10^{-30}}{1.054 \times 10^{-34}}\,\mathrm{s}^{-1} = 2.293 \times 10^{4}\,\mathrm{s}^{-1} \,. \tag{5.4}$$

The lifetime of the energy levels can is then obtained as the reciprocal of the spread of the states, given as

$$\boxed{t_{lifetime} = 4.36 \times 10^{-5}\,\mathrm{s}\,.} \tag{5.5}$$

Thus, the states have a lifetime of approximately $43.60\,\mu$s due to the dispersion caused by the electric field along z-direction arising from the patch potentials.

5.2 Dispersion of Energy Levels due to Shot Noise

Let us consider we have a laser of wavelength λ_{laser} and power P_{laser}. Then, the energy of the laser is given as

$$E = P_{laser} \times t \,. \tag{5.6}$$

Here, t is the pulse time of the laser. The total energy is carried by say n photons

$$E = nh\nu = n\frac{hc}{\lambda_{laser}} \,. \tag{5.7}$$

Comparing the two energy expressions, we obtain the total number of photons as

$$n = P_{laser} \times t \times \frac{\lambda_{laser}}{hc} \,. \tag{5.8}$$

We can also estimate the number of photons per unit time

$$N = P_{laser}\frac{\lambda_{laser}}{hc}\,. \tag{5.9}$$

The shot noise of the laser or photon shot noise is a noise which originates due to the discrete nature of photons and the random nature of the photon emission process. It is a random noise and can be modeled with a Poisson distribution as [23]

$$\boxed{\text{Shot Noise} \propto \sqrt{\frac{\lambda_{laser}P_{laser}}{hc}} \propto \sqrt{N}\,.} \tag{5.10}$$

It is independent of the electronics of the devices and its effect can be reduced by reducing the power of the laser.

The spacing between the harmonic oscillator levels can be approximated using the spring constant and mass of the quantum dot, which has been defined in equation (1.2) as

$$\Omega = \sqrt{\frac{k_{sp}}{M}}\,. \tag{5.11}$$

The spring constant can then be written as [24]

$$\vec{k}_{sp} = \begin{cases} k_{sp}(x) = \dfrac{\alpha_{dot}}{x}\nabla(\vec{E}.\vec{E}*)\Big|_{y,z=0} = \dfrac{4\mathrm{Re}(\alpha_{dot})(NA)^4\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,, \\ k_{sp}(y) = \dfrac{\alpha_{dot}}{y}\nabla(\vec{E}.\vec{E}*)\Big|_{y,z=0} = \dfrac{4\mathrm{Re}(\alpha_{dot})(NA)^4\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,, \\ k_{sp}(z) = \dfrac{\alpha_{dot}}{z}\nabla(\vec{E}.\vec{E}*)\Big|_{y,z=0} = \dfrac{2\mathrm{Re}(\alpha_{dot})(NA)^6\pi^3}{c\epsilon_0\lambda_{laser}^4}\,P_{laser}\,. \end{cases} \tag{5.12}$$

Here (NA) is the numerical aperture of the laser beam, P_{laser} is the power of the laser, λ_{laser} is the wavelength and α_{dot} is the polarizability of the trapped particle. From the above expression, we can also write the spring constant as a linear function of $\alpha_{dot}\,P_{laser}$, with κ being a constant.

$$k_{sp} = \sqrt{k_{sp}^2(x) + k_{sp}^2(y) + k_{sp}^2(z)} = \kappa\,\alpha_{dot}\,P_{laser}\,. \tag{5.13}$$

Thus, using equations (5.13) and (5.11), we can write the spacing between levels as a function of P_{laser},

$$\Omega = \sqrt{\frac{\kappa\,\alpha_{dot}\,P_{laser}}{M}} = \sqrt{\frac{\kappa\,\alpha_{dot}}{M}}\,(P_{laser})^{1/2}\,. \tag{5.14}$$

The error or spread in Ω can be written as

$$\sigma_\Omega = \frac{1}{2}\Omega\left|\frac{\sigma_{P_{laser}}}{P_{laser}}\right|\,. \tag{5.15}$$

The power of the laser can be written in terms of the number of photons per unit time

$$P_{laser} = \frac{hc}{\lambda_{laser}}\,N\,. \tag{5.16}$$

Then the error in P_{laser} can be written as

$$\sigma_{P_{laser}} = \left|P_{laser}\frac{\sigma_N}{N}\right|\,. \tag{5.17}$$

The error in N is obtained as the shot noise of the laser

$$\sigma_N \propto \sqrt{N}\,. \tag{5.18}$$

Putting everything together, we obtain the uncertainty in the laser power as

$$\sigma_{P_{laser}} \propto \sqrt{\frac{hc}{\lambda_{laser}}P_{laser}}\,. \tag{5.19}$$

Compiling together the equations (5.19), (5.15) and (5.14), we obtain

$$\sigma_\Omega \propto \frac{1}{2}\sqrt{\frac{\kappa\,\alpha_{dot}}{M}\left(\frac{hc}{\lambda_{laser}}\right)}\,, \tag{5.20}$$

$$\kappa = 2(NA)^4\sqrt{8+(NA)^4}\,\frac{\pi^3}{c\epsilon_0\lambda_{laser}^4}\,. \tag{5.21}$$

We can find the mass of the quantum dot by considering the density of CdSe [25] and CdS [26] given as

$$\rho_{CdSe} = 5.82\,\mathrm{g/cm^3}\,, \tag{5.22}$$

$$\rho_{CdS} = 4.82\mathrm{g/cm^3}\,. \tag{5.23}$$

The volume of CdSe and CdS is obatined for $r_{core} = 1.5\,\mathrm{nm}$ and $r_{shell} = 2.9\,\mathrm{nm}$ as

$$V_{CdSe} = \frac{4\pi}{3}(1.5\times10^{-9})^3 = 1.414\times10^{-26}\,\mathrm{m^{-3}}\,,$$
$$V_{CdS} = \frac{4\pi}{3}\times(2.9-1.5)^3\times(10^{-9})^3 = 1.149\times10^{-26}\,\mathrm{m^{-3}}\,.$$

Then we obtain the masses of CdSe, CdS and the total quantum dot as

$$m_{CdSe} = 8.229\times10^{-23}\,\mathrm{kg}\,, \tag{5.24}$$

$$m_{CdS} = 5.538\times10^{-23}\,\mathrm{kg}\,, \tag{5.25}$$

$$M = 1.377\times10^{-22}\,\mathrm{kg}\,. \tag{5.26}$$

For the rest of the parameters in equation (5.20), we will first obtain the error in Ω for the experimental parameters used by Delic *et al.* [19] in their experiment

$$(NA)_{Delic} = 0.8\,, \tag{5.27}$$

$$\lambda_{laser,\,Delic} = 1064\,\mathrm{nm}\,. \tag{5.28}$$

Using all these parameters, the uncertainty in Ω for the Delic *et al.* setup [19] using a CdSe-CdS quantum dot is given as

$$\boxed{\sigma_{\Omega,Delic} \approx 3.436\times10^{-3}\,\mathrm{s^{-1}}\,.} \tag{5.29}$$

The lifetime of energy levels can then be obtained as the reciprocal of the spread of the states as

$$\boxed{t_{lifetime} \approx 2.910 \times 10^2 \,\mathrm{s}\,.} \tag{5.30}$$

Thus, the electrostatic interactions have a greater effect on reducing the lifetime of energy states of he oscillator levels while the shot noise of the laser has no influence on the same.

REFERENCES

[1] A. Einstein, "The foundation of the general theory of relativity.," *Annalen Phys.*, vol. 49, no. 7, J.-P. Hsu and D. Fine, Eds., pp. 769–822, 1916. DOI: 10.1002/andp.19163540702.

[2] E. Fischbach, D. E. Krause, C. Talmadge, and D. Tadic, "Higher order weak interactions and the equivalence principle," *Phys. Rev. D*, vol. 52, pp. 5417–5427, 1995. DOI: 10.1103/PhysRevD.52.5417.

[3] N. Arkani-Hamed, S. Dimopoulos, and G. Dvali, "Phenomenology, astrophysics, and cosmology of theories with submillimeter dimensions and tev scale quantum gravity," *Physical Review D*, vol. 59, no. 8, Mar. 1999, ISSN: 1089-4918. DOI: 10.1103/physrevd.59.086004. [Online]. Available: http://dx.doi.org/10.1103/PhysRevD.59.086004.

[4] R. H. Rapp, "An estimate of equatorial gravity from terrestrial and satellite data," *Geophysical Research Letters*, vol. 14, no. 7, pp. 730–732, 1987. DOI: https://doi.org/10.1029/GL014i007p00730. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/GL014i007p00730. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/GL014i007p00730.

[5] D. E. Smith, D. C. Christodoulidis, R. Kolenkiewicz, P. J. Dunn, S. M. Klosko, M. H. Torrence, S. Fricke, and S. Blackwell, "A global geodetic reference frame from lageos ranging (sl5.1ap)," *Journal of Geophysical Research: Solid Earth*, vol. 90, no. B11, pp. 9221–9233, 1985. DOI: https://doi.org/10.1029/JB090iB11p09221. eprint: https://agupubs.onlinelibrary.wiley.com/doi/pdf/10.1029/JB090iB11p09221. [Online]. Available: https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/JB090iB11p09221.

[6] J. O. Dickey, P. L. Bender, J. E. Faller, *et al.*, "Lunar laser ranging: A continuing legacy of the apollo program," *Science*, vol. 265, no. 5171, pp. 482–490, 1994. DOI: 10.1126/science.265.5171.482. eprint: https://www.science.org/doi/pdf/10.1126/science.265.5171.482. [Online]. Available: https://www.science.org/doi/abs/10.1126/science.265.5171.482.

[7] S. C. Holding, F. D. Stacey, and G. J. Tuck, "Gravity in minesmdashan investigation of newton's law," *Phys. Rev. D*, vol. 33, pp. 3487–3494, 12 Jun. 1986. DOI: 10.1103/PhysRevD.33.3487. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.33.3487.

[8] M. A. Zumberge, J. A. Hildebrand, J. M. Stevenson, R. L. Parker, A. D. Chave, M. E. Ander, and F. N. Spiess, "Submarine measurement of the newtonian gravitational constant," *Phys. Rev. Lett.*, vol. 67, pp. 3051–3054, 22 Nov. 1991. DOI: 10.1103/PhysRevLett.67.3051. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.67.3051.

[9] A. Cornaz, B. Hubler, and W. Kündig, "Determination of the gravitational constant at an effective interaction distance of 112 m," *Phys. Rev. Lett.*, vol. 72, pp. 1152–1155, 8 Feb. 1994. DOI: 10.1103/PhysRevLett.72.1152. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.72.1152.

[10] B. Hubler, A. Cornaz, and W. Kündig, "Determination of the gravitational constant with a lake experiment: New constraints for non-newtonian gravity," *Phys. Rev. D*, vol. 51, pp. 4005–4016, 8 Apr. 1995. DOI: 10.1103/PhysRevD.51.4005. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.51.4005.

[11] A. J. Romaides, R. W. Sands, E. Fischbach, and C. L. Talmadge, "Final results from the wabg tower gravity experiment," *Phys. Rev. D*, vol. 55, pp. 4532–4536, 8 Apr. 1997. DOI: 10.1103/PhysRevD.55.4532. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevD.55.4532.

[12] C. Talmadge, J. P. Berthias, R. W. Hellings, and E. M. Standish, "Model Independent Constraints on Possible Modifications of Newtonian Gravity," *Phys. Rev. Lett.*, vol. 61, pp. 1159–1162, 1988. DOI: 10.1103/PhysRevLett.61.1159.

[13] A. Ashkin, J. M. Dziedzic, J. E. Bjorkholm, and S. Chu, "Observation of a single-beam gradient force optical trap for dielectric particles," *Opt. Lett.*, vol. 11, no. 5, pp. 288–290, May 1986. DOI: 10.1364/OL.11.000288. [Online]. Available: https://opg.optica.org/ol/abstract.cfm?URI=ol-11-5-288.

[14] A. Ashkin, "Optical trapping and manipulation of neutral particles usinglasers," *Proceedings of the National Academy of Sciences*, vol. 94, no. 10, pp. 4853–4860, 1997. DOI: 10.1073/pnas.94.10.4853. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.94.10.4853. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.94.10.4853.

[15] J. Gieseler, B. Deutsch, R. Quidant, and L. Novotny, "Subkelvin parametric feedback cooling of a laser-trapped nanoparticle," *Physical Review Letters*, vol. 109, no. 10, Sep. 2012, ISSN: 1079-7114. DOI: 10.1103/physrevlett.109.103603. [Online]. Available: http://dx.doi.org/10.1103/PhysRevLett.109.103603.

[16] D. Bera, L. Qian, T.-K. Tseng, and P. H. Holloway, "Quantum dots and their multimodal applications: A review," en, *Materials (Basel)*, vol. 3, no. 4, pp. 2260–2345, Mar. 2010.

[17] Y. Nandan and M. S. Mehata, "Wavefunction engineering of Type-I/Type-II excitons of CdSe/CdS Core-Shell quantum dots," *Scientific Reports*, vol. 9, no. 1, p. 2, Jan. 2019.

[18] M. Hua, "Optical refrigeration on CdSe/CdS quantum dots," Oct. 2019. DOI: 10.25394/PGS.9808400.v1. [Online]. Available: https://hammer.purdue.edu/articles/thesis/Optical_refrigeration_on_CdSe_CdS_quantum_dots/9808400.

[19] U. . Deli , M. Reisenbauer, D. Grass, N. Kiesel, V. Vuleti , and M. Aspelmeyer, "Cavity cooling of a levitated nanosphere by coherent scattering," *Phys. Rev. Lett.*, vol. 122, p. 123 602, 12 Mar. 2019. DOI: 10.1103/PhysRevLett.122.123602. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.122.123602.

[20] S. Ninomiya and S. Adachi, "Optical properties of cubic and hexagonal cdse," *Journal of Applied Physics*, vol. 78, no. 7, pp. 4681–4689, Oct. 1995, ISSN: 0021-8979. DOI: 10.1063/1.359815. [Online]. Available: https://doi.org/10.1063/1.359815.

[21] R. E. Treharne, A. Seymour-Pierce, K. Durose, K. Hutchings, S. Roncallo, and D. Lane, "Optical design and fabrication of fully sputtered cdte/cds solar cells," *Journal of Physics: Conference Series*, vol. 286, no. 1, p. 012 038, Mar. 2011. DOI: 10.1088/1742-6596/286/1/012038. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/286/1/012038.

[22] R. O. Behunin, D. A. R. Dalvit, R. S. Decca, *et al.*, "Kelvin probe force microscopy of metallic surfaces used in casimir force measurements," *Phys. Rev. A*, vol. 90, p. 062 115, 6 Dec. 2014. DOI: 10.1103/PhysRevA.90.062115. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevA.90.062115.

[23] Y. Blanter and M. Büttiker, "Shot noise in mesoscopic conductors," *Physics Reports*, vol. 336, no. 12, pp. 1–166, Sep. 2000, ISSN: 0370-1573. DOI: 10.1016/s0370-1573(99)00123-4. [Online]. Available: http://dx.doi.org/10.1016/S0370-1573(99)00123-4.

[24] J. Gieseler, B. Deutsch, R. Quidant, and L. Novotny, "Subkelvin parametric feedback cooling of a laser-trapped nanoparticle," *Phys. Rev. Lett.*, vol. 109, p. 103 603, 10 Sep. 2012. DOI: 10.1103/PhysRevLett.109.103603. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevLett.109.103603.

[25] J. Rumble, *Crc handbook of chemistry and physics, 98th edition*, Jun. 2017. [Online]. Available: https://www.taylorfrancis.com/books/9781498784559.

[26] Pubchem. [Online]. Available: https://pubchem.ncbi.nlm.nih.gov/compound/Cadmium-Sulfide#section=Color-Form.

A. SUPPLEMENTARY MATERIAL FOR CHAPTER 2

A.1 Classical Polarizability of a Quantum Dot

We can find the classical polarizability for a quantum dot by solving the Laplace equation for the dot in an electric field.

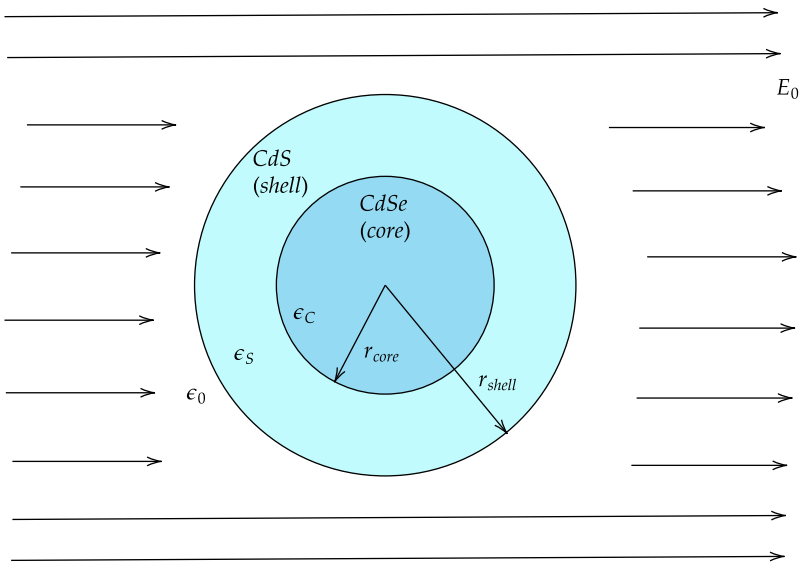

Figure A.1. Quantum Dot in Electric Field

We can write the general solution for the potential in terms of Legendre polynomials

$$\Phi_{core}(r,\theta) = \sum_{l=0}^{\infty} A_l\, r^l\, P_l(\cos\theta)\,, \tag{A.1}$$

$$\Phi_{shell}(r,\theta) = \sum_{l=0}^{\infty} \left(B_l\, r^l + \frac{C_l}{r^{l+1}}\right) P_l(\cos\theta)\,, \tag{A.2}$$

$$\Phi_{out}(r.\theta) = \sum_{l=0}^{\infty} \left(D_l\, r^l + \frac{F_l}{r^{l+1}}\right) P_l(\cos\theta)\,. \tag{A.3}$$

We know that at $r \gg r_{shell}$, the potential takes the form $\Phi_{out} \longrightarrow -E_0 r\cos\theta$. This acts as our first boundary condition. On enforcing this, we conclude that only D_1 survives, and all other values for $D_{l,l\neq 0} = 0$

$$\boxed{D_1 = -E_0\,.} \tag{A.4}$$

Now, the potential outside the box can be written as

$$\Phi_{out}(r,\theta) = D_1 r\cos\theta + \frac{F_0}{r} + \frac{F_1}{r^2}\cos\theta + \cdots . \tag{A.5}$$

The term corresponding to F_1 in the above expression encapsulates the dipole moment of the quantum dot

$$\vec{p}_{dipole} = \alpha_{dot}\vec{E}_0 . \tag{A.6}$$

So, we need to solve for F_1 in terms of the background electric field E_0. We can do so by enforcing the continuity of the electric field across all boundaries boundaries

$$-\epsilon_S \frac{\partial \Phi_{shell}}{\partial r}\bigg|_{r=r_{shell}} = -\epsilon_0 \frac{\partial \Phi_{out}}{\partial r}\bigg|_{r=r_{shell}} , \tag{A.7}$$

$$-\frac{1}{r_{shell}}\frac{\partial \Phi_{shell}}{\partial \theta}\bigg|_{r=r_{shell}} = -\frac{1}{r_{shell}}\frac{\partial \Phi_{out}}{\partial \theta}\bigg|_{r=r_{shell}} , \tag{A.8}$$

$$-\epsilon_C \frac{\partial \Phi_{core}}{\partial r}\bigg|_{r=r_{core}} = -\epsilon_S \frac{\partial \Phi_{shell}}{\partial r}\bigg|_{r=r_{shell}} , \tag{A.9}$$

$$-\frac{1}{r_{core}}\frac{\partial \Phi_{core}}{\partial \theta}\bigg|_{r=r_{core}} = -\frac{1}{r_{core}}\frac{\partial \Phi_{shell}}{\partial \theta}\bigg|_{r=r_{core}} . \tag{A.10}$$

Writing the $l = 1$ terms for the above boundary conditions

$$\text{Boundary 1}: \; B_1 - \frac{2C_1}{r_{shell}^3} = \frac{\epsilon_0}{\epsilon_S}\left(D_1 - \frac{2F_1}{r_{shell}^3}\right) , \tag{A.11}$$

$$\text{Boundary 2}: \; B_1 + \frac{C_1}{r_{shell}^3} = D_1 + \frac{F_1}{r_{shell}^3} , \tag{A.12}$$

$$\text{Boundary 3}: \; A_1 = \frac{\epsilon_S}{\epsilon_C}\left(B_1 - \frac{2C_1}{r_{core}^3}\right) , \tag{A.13}$$

$$\text{Boundary 4}: \; A_1 = B_1 + \frac{C_1}{r_{core}^3} . \tag{A.14}$$

Now, using Boundary 4 in equation (A.14) we can write B_1 in terms of A_1 and C_1 as

$$B_1 = A_1 - \frac{C_1}{r_{core}^3} .$$

Putting this in Boundary 3, we can now get C_1 completely in terms of A_1 as

$$C_1 = \left(\frac{\epsilon_S - \epsilon_C}{\epsilon_C}\right) \frac{r_{core}^3}{3} A_1 \,. \tag{A.15}$$

Thus, we can also obtain B_1 purely as a function of A_1 as

$$B_1 = \frac{2\epsilon_S + \epsilon_C}{3\epsilon_S} A_1 \,. \tag{A.16}$$

Now using equations (A.16, A.15) in Boundary 1 (equation A.11) as

$$F_1 = \frac{r_{shell}^3}{2\epsilon_0} \left[\epsilon_0 D_1 - \left\{\left(\frac{2\epsilon_S + \epsilon_C}{3}\right) - \frac{2r_{core}^3}{3r_{shell}^3}(\epsilon_S - \epsilon_C)\right\} A_1\right] . \tag{A.17}$$

Plugging equation (A.17) into Boundary 2 (equation A.12), we obtain

$$\begin{aligned}
&\left[\left(\frac{2\epsilon_S + \epsilon_C}{3\epsilon_S}\right) + \frac{r_{core}^3}{3r_{shell}^3}\left(\frac{\epsilon_S - \epsilon_C}{\epsilon_S}\right)\right] A_1 = D_1 + \frac{F_1}{r_{shell}^3} \,, \\
\implies &\left[\left(\frac{2\epsilon_S + \epsilon_C}{3\epsilon_S}\right) + \frac{r_{core}^3}{3r_{shell}^3}\left(\frac{\epsilon_S - \epsilon_C}{\epsilon_S}\right)\right] A_1 = D_1 + \frac{D_1}{2} - \frac{1}{2\epsilon_0}\left[\frac{2\epsilon_S + \epsilon_C}{3} + \frac{2r_{core}^3}{3r_{shell}^3}(\epsilon_C - \epsilon_S)\right] A_1 \,, \\
\implies &\left[\left(\frac{2\epsilon_S + \epsilon_C}{3}\right)\left(\frac{1}{\epsilon_S} + \frac{1}{2\epsilon_0}\right) + \frac{r_{core}^3}{2r_{shell}^3}(\epsilon_C - \epsilon_S)\left(\frac{1}{\epsilon_0} - \frac{1}{\epsilon_S}\right)\right] A_1 = \frac{3D_1}{2} \,, \\
\implies &\left[\frac{(2\epsilon_S + \epsilon_C)(2\epsilon_0 + \epsilon_S)}{2\epsilon_0\epsilon_S} + \frac{r_{core}^3}{r_{shell}^3}\frac{(\epsilon_C - \epsilon_S)(\epsilon_S - \epsilon_0)}{\epsilon_0\epsilon_S}\right] \frac{A_1}{3} = \frac{3D_1}{2} \,, \\
\implies &\left[\frac{r_{shell}^3(2\epsilon_S + \epsilon_C)(2\epsilon_0 + \epsilon_S) + 2a^3(\epsilon_C - \epsilon_S)(\epsilon_S - \epsilon_0)}{2r_{shell}^3\epsilon_0\epsilon_S}\right] \frac{A_1}{3} = \frac{3D_1}{2} \,.
\end{aligned}$$

We finally get A_1 in terms of D_1, thus in terms of E_0 as

$$\boxed{\therefore\ A_1 = -9E_0 \left[\frac{r_{shell}^3\epsilon_0\epsilon_S}{r_{shell}^3(2\epsilon_S + \epsilon_C)(2\epsilon_0 + \epsilon_S) + 2r_{core}^3(\epsilon_C - \epsilon_S)(\epsilon_S - \epsilon_0)}\right] .} \tag{A.18}$$

We can then find F_1 in terms of E_0 as

$$\begin{aligned}
&F_1 = \frac{r_{shell}^3}{2\epsilon_0}\left[\epsilon_0 D_1 - \frac{A_1}{3}\left\{\frac{(2\epsilon_S + \epsilon_C)r_{shell}^3 + 2r_{core}^3(\epsilon_C - \epsilon_S)}{r_{shell}^3}\right\}\right] \,, \\
\implies &F_1 = \frac{r_{shell}^3 D_1}{2} - \frac{A_1}{3\epsilon_0}\left\{(2\epsilon_S + \epsilon_C)r_{shell}^3 + 2r_{core}^3(\epsilon_C - \epsilon_S)\right\} \,,
\end{aligned}$$

$$\implies F_1 = \frac{r_{shell}^3 D_1}{2} - \frac{3D_1}{2\epsilon_0} \frac{r_{shell}^3 \epsilon_0 \epsilon_S \left\{(2\epsilon_S + \epsilon_C) r_{shell}^3 + 2r_{core}^3(\epsilon_C - \epsilon_S)\right\}}{r_{shell}^3(2\epsilon_S + \epsilon_C)(2\epsilon_0 + \epsilon_S) + 2a^3(\epsilon_C - \epsilon_S)(\epsilon_S - \epsilon_0)},$$

$$\implies F_1 = -\frac{r_{shell}^3 E_0}{2} \times 2 \frac{[r_{shell}^3(2\epsilon_S + \epsilon_C) - r_{core}^3(\epsilon_C - \epsilon_S)]\epsilon_0 - [r_{shell}^3(2\epsilon_S + \epsilon_C) + 2a^3(\epsilon_C - \epsilon_S)]\epsilon_S}{[r_{shell}^3(2\epsilon_S + \epsilon_C) - r_{core}^3(\epsilon_C - \epsilon_S)]2\epsilon_0 + [r_{shell}^3(2\epsilon_S + \epsilon_C) + 2a^3(\epsilon_C - \epsilon_S)]\epsilon_S},$$

$$\implies F_1 = -r_{shell}^3 E_0 \left[\frac{\epsilon_0 - \left\{ \frac{r_{shell}^3(2\epsilon_S + \epsilon_C) + 2r_{core}^3(\epsilon_C - \epsilon_S)}{r_{shell}^3(2\epsilon_S + \epsilon_C) - r_{core}^3(\epsilon_C - \epsilon_S)} \right\} \epsilon_S}{2\epsilon_0 + \left\{ \frac{r_{shell}^3(2\epsilon_S + \epsilon_C) + 2r_{core}^3(\epsilon_C - \epsilon_S)}{r_{shell}^3(2\epsilon_S + \epsilon_C) - r_{core}^3(\epsilon_C - \epsilon_S)} \right\} \epsilon_S} \right].$$

We can define effective permittivity ϵ_e and get the final expression for F_1 as

$$\boxed{F_1 = r_{shell}^3 E_0 \left(\frac{\epsilon_e - \epsilon_0}{\epsilon_e + 2\epsilon_0} \right),} \tag{A.19}$$

$$\boxed{\epsilon_e = \epsilon_S \frac{r_{shell}^3(\epsilon_C + 2\epsilon_S) + 2a^3(\epsilon_C - \epsilon_S)}{r_{shell}^3(\epsilon_C + 2\epsilon_S) - r_{core}^3(\epsilon_C - \epsilon_S)}.} \tag{A.20}$$

Thus, we get α as

$$\boxed{\alpha = 4\pi\epsilon_0 r_{shell}^3 \left(\frac{\epsilon_e - \epsilon_0}{\epsilon_e + 2\epsilon_0} \right).} \tag{A.21}$$

A.2 Unperturbed Wavefunctions of a Quantum Dot : Analytic Derivation

The quantum dot is a particle in a box system for conduction electrons and valence holes. The wavefunctions for the conduction electrons/valence holes can be written using spherical Bessel functions of first (j_l) and second (n_l) kind as

$$\psi^{c/v,core}(r,\theta,\varphi) = A_{e/h,l}^{c/v,core}\, j_l(k_{e/h,l}^{c/v,core} r)\, Y_m^l(\theta,\varphi), \tag{A.22}$$

$$\psi^{c/v,shell}(r,\theta,\varphi) = \left[A_{e/h,l}^{c/v,shell}\, j_l(k_{e/h,l}^{c/v,shell} r) + B_{e/h,l}^{c/v,shell}\, n_l(k_{e/h,l}^{c/v,shell} r) \right] Y_m^l(\theta,\varphi). \tag{A.23}$$

The k value can be written as

$$k_{e/h}^{c/v,core/shell} = \sqrt{\frac{2\left(\varepsilon_{e/h}^{c/v,core/shell} - V^{c/v,core/shell}\right) m_{e/h}^{c/v,core/shell}}{\hbar^2}}. \tag{A.24}$$

The wavefunctions and its derivatives must be continuous across the core-shell boundary, and the shell-wavefunction should vanish at the shell-boundary,

$$\psi_{e/h}^{c/v,core}\Big|_{r=r_{core}} = \psi_{e/h}^{c/v,shell}\Big|_{r=r_{core}}\,, \tag{A.25}$$

$$\frac{1}{m_{e/h}^{c/v,core}}\frac{\partial}{\partial r}\psi_{e/h}^{c/v,core}\Big|_{r=r_{core}} = \frac{1}{m_{e/h}^{c/v,shell}}\frac{\partial}{\partial r}\psi_{e/h}^{c/v,shell}\Big|_{r=r_{core}}\,, \tag{A.26}$$

$$\psi_{e/h}^{c/v,shell}\Big|_{r=r_{shell}} = 0\,. \tag{A.27}$$

The first boundary condition can be explicitly written as

$$A_{e/h,l}^{c/v,core} j_l(k_{e/h,l}^{c/v,core} r_{core}) = A_{e/h,l}^{c/v,shell} j_l(k_{e/h,l}^{c/v,shell} r_{core}) + B_{e/h,l}^{c/v,shell} n_l(k_{e/h}^{c/v,shell} r_{core})\,. \tag{A.28}$$

The second boundary condition can be written as

$$\begin{aligned}\frac{m_{e/h}^{c/v,shell}}{m_{e/h}^{c/v,shell}}\left[A_{e/h,l}^{c/v,core} k_{e/h,l}^{c/v,core} j_l(k_{e/h,l}^{c/v,core} r_{core})\right] &= A_{e/h,l}^{c/v,shell} k_{e/h,l}^{c/v,shell} j_l'(k_{e/h,l}^{c/v,shell} r_{core}) \\ &\quad + B_{e/h,l}^{c/v,shell} k_{e/h,l}^{c/v,shell} n_l'(k_{e/h,l}^{c/v,shell} r_{core})\,.\end{aligned} \tag{A.29}$$

Now, we can multiply equation (A.29) with $\dfrac{n_l(k_{e/h,l}^{c/v,shell} r_{core})}{n_l'(k_{e/h,l}^{c/v,shell} r_{core})}$ and subtract it from equation (A.28) to finally obtain

$$A_{e/h,l}^{c/v,shell} = \frac{j_l(k_{e/h,l}^{c/v,core} r_{core}) n_l'(k_{e/h,l}^{c/v,shell} r_{core}) - \dfrac{m_{e/h}^{c/v,shell} k_{e/h,l}^{c/v,core}}{m_{e/h}^{c/v,core} k_{e/h,l}^{c/v,shell}} j_l'(k_{e/h,l}^{c/v,core} r_{core}) n_l(k_{e/h,l}^{c/v,shell} r_{core})}{j_l(k_{e/h,l}^{c/v,shell} r_{core}) n_l'(k_{e/h}^{c/v,shell} r_{core}) - j_l'(k_{e/h,l}^{c/v,shell} r_{core}) n_l(k_{e/h,l}^{c/v,shell} r_{core})} A_{e/h,l}^{c/v,core}\,. \tag{A.30}$$

We can then multiply equation (A.29) with $\dfrac{j_l(k_{e/h,l}^{c/v,shell} r_{core})}{j_l'(k_{e/h,l}^{c/v,shell} r_{core})}$ and subtract it from equation (A.28) to obtain

$$B_{e/h,l}^{c/v,shell} = \frac{j_l(k_{e/h,l}^{c/v,core} r_{core}) j_l'(k_{e/h,l}^{c/v,shell} r_{core}) - \dfrac{m_{e/h}^{c/v,shell} k_{e/h,l}^{c/v,core}}{m_{e/h}^{c/v,core} k_{e/h,l}^{c/v,shell}} j_l'(k_{e/h,l}^{c/v,core} r_{core}) j_l(k_{e/h,l}^{c/v,core} r_{core})}{j_l'(k_{e/h,l}^{c/v,shell} r_{core}) n_l(k_{e/h}^{c/v,shell} r_{core}) - j_l(k_{e/h,l}^{c/v,shell} r_{core}) n_l'(k_{e/h,l}^{c/v,shell} r_{core})} A_{e/h,l}^{c/v,core}\,. \tag{A.31}$$

A.3 Unperturbed Wavefunctions of a Quantum Dot : Computational Solution

Since the wavefunctions of the quantum dot are complicated functions of Bessel functions, they need to be solved for numerically. I have used Python to produce a working code for the same. In this section, I will first attempt to explain the different sections of the code written, in the order of execution, and then finally include the complete codes.

Step 1 : Import all libraries needed and define all general functions

I first import all the libraries I would need for the code and then define all the functions necessary, like numerical integration, writing into files, reading files, matrix multiplication, root finding and so on.

```
# Generating difference of wavefunctions of Quantum Dot and finding what
    is the root energy for CE or VH for various l values (Boundary 3 -
    infinite well)

import math
import cmath
import sys
import time
import os
import numpy as np
import scipy
from scipy.special import spherical_jn
from scipy.special import spherical_yn
from scipy.special import spherical_in
from scipy.special import spherical_kn
import matplotlib
from matplotlib import pyplot


# all external functions needed for the code

# READ FILE
```

```
21 def read_matrix(x): #more than one column #parameter: x ==> name of file
22     f=open(x,'r') #'r' ==> read only
23     X=[[float(num) for num in line.split(' ')] for line in f]
24     f.close()
25     return(X)
26 #
27
28 # APPEND FILE
29 def append_file(name, str): #arguments: name ==> name of file, str ==>
       string to append
30     f=open(name, 'a') #'a' ==> append file
31     f.write(str)
32     f.close()
33     #
34 #
35
36 #MOD SQUARE OF A COMPLEX NUMBER
37 def modulus(x): #x ==> complex number
38    real=x.real**2
39    imag=x.imag**2
40    modsq=real+imag
41    return(modsq)
42 #
43
44 # DERIVATIVE
45 def derivative(f, x, h): # arguments = equation, point at which derivative
       is to be found, h
46     y=(f(x+h)-f(x-h))/(2*h)
47     return(y)
48 #
49
50 # NEWTON RAPHSON
51 def newton_raphson(f, a, max_itr, nm): #arguments ==> formula, trial root,
       maximum iterations, name of file to store errors
52     err=pow(10, -10)
53     n=0
```

```
    X=[0 for i in range(max_itr)]
    while n<max_itr: #so that it does not cross max number of allowed iterations
        if n==0:
            X[n]=a
        else:
            h=X[n-1]*0.0001
            X[n]=X[n-1]-(f(X[n-1])/derivative(f, X[n-1], h))
            #error = x_{n+1}-x_{n} o equivalently, x_{n}-x_{n-1}
            append_file(nm, f'{n} {abs(X[n] - X[n - 1])}\n') #appends absolute error value with iteration number to file
            if abs(X[n]-X[n-1])<err: return(X[n], True)
            #
        n+=1
        #
    return(X[n-1], False) #if root not obtained even after max_iterations
#

def simpson(f, lower, upper, N, h, name):
    I=0
    X=[0 for i in range(N+1)] #for storing N+1 values, including upper and lower
    X[0]=lower
    for i in range(1,N+1):
        X[i]=lower+h*i #finding the next point
        mid=(X[i-1]+X[i])/2 #finding mid value
        h1=(X[i]-X[i-1])/2  #diving every slice in two equal slices, and finding h relevant to that slice
        #calculating the integration
        I=I+(h1/3)*(f(X[i-1])+4*f(mid)+f(X[i]))
        #
    #storing N vs I value in file
    append_file(name, f'{N} {I}\n')
    return(I)
#
```

Step 2: Choose a unit system for the code and define all constants

I am using natural units for the codes developed here, and I will include the full conversions in detail in Appendix A.6.

```
#---------------------- MAIN CODE STARTS -------------------------#

#clock beginning time to measure time taken by code to run
begin=time.time()

#setting path of file
path=os.path.dirname(os.path.realpath(__file__))
sys.path.append(path)
# ------------------------------------------------------------------------ #
#defining all constants => masses (natural units, electron volts)
hbar=1.0
m0=0.511*(10**6) #rest mass of electon in natural units
mecore=0.11*m0
meshell=0.14*m0
mhcore=0.8*m0
mhshell=0.51*m0
# ------------------------------------------------------------------------
   #


#defining all constants => radii (in nanometers)
rcorenm=1.5
shellthicknessnm=1.4
rshellnm=rcorenm+shellthicknessnm
# converting to natural units
confac=5.066*(10**(-3))
rcore=confac*rcorenm
rshell=confac*rshellnm
# ------------------------------------------------------------------------ #
```

Step 3 : Define the variables of the problem

Now, the variables of the problem, like $k_{e/h}^{c/v,core/shell}$, $V_{e/h}^{c/v,core/shell}$, the wavefunctions and its coefficients can be defined, symbolically.

```
#defining "k" as a function of energy, potential and mass (for e/h in c/v
    bands respectively)
def k(E,V,M):
    hbar=1.0
    if (E-V)>=0 : kval=math.sqrt((2*(E-V)*M)/(hbar**2))
    else : kval=cmath.sqrt((2*(E-V)*M)/(hbar**2))
    #print(f'E-V={E-V}\n k={kval}')
    return(kval)
#
# ------------------------------------------------------------------------ #

#defining potential for conduction (for e) band as a function of r (in eV)
def Vc(r,core,shell):
    #defining all constants => potentials (natural units, electron volts)
    Vcshell=0.0
    Vccore=-0.1 #wrt Vcshell=0
    #note, Vccore and Vvcore are not zero wrt each other
    #
    if abs(r)<=core: V=Vccore
    elif abs(r)>core and abs(r)<=shell: V=Vcshell
    else: V=10**15 #tends to infinity
    return(V)
#

#defining potential for valence (for h) band as a function of r (in eV)
def Vv(r,core,shell):
    #defining all constants => potentials (natural units, electron volts)
    Vvcore=-0.65 #wrt Vcshell=0
    Vvshell=0.0
    #note, Vccore and Vvcore are not zero wrt each other
    #
```

```
    if abs(r)<=core: V=Vvcore
    elif abs(r)>core and abs(r)<=shell: V=Vvshell
    else: V=-10**15 #tends to infinity
    return(V)
#

# ------------------------------- functions for wavefunction (CE and VH)
   ------------------------------------------ #

# defining wavefunction of core for electrons (in conduction band)
def psi_CE_core(Acecore,l,kcecore,r):
    psicecore=Acecore*spherical_jn(l, kcecore*r)
    return(psicecore)
#

# defining wavefunction of shell for electrons (in conduction band)
def psi_CE_shell(Aceshell,Bceshell,l,kceshell,r):
    psiceshell=Aceshell*spherical_jn(l,kceshell*r)+Bceshell*spherical_yn(l
   ,kceshell*r)
    return(psiceshell)
#

#defining wavefunction of core for holes (in valence band)
def psi_VH_core(Avhcore,l,kvhcore,r):
    psivhcore=Avhcore*spherical_jn(l,kvhcore*r)
    return(psivhcore)
#

# defining wavefunction for shell for holes (in valence band)
def psi_VH_shell(Avhshell,Bvhshell,l,kvhshell,r):
    psivhshell=Avhshell*spherical_jn(l,kvhshell*r)+Bvhshell*spherical_yn(l
   ,kvhshell*r)
    return(psivhshell)
#
# ---------------------------------------- A,B functions (CE and VH)
   ------------------------------------------------ #
```

```
63
64 # defining A_CE_shell
65 def A_CE_shell(Acecore,l,kcecore,kceshell,rcore,Mcecore,Mceshell):
66     numceshell=(spherical_jn(l,kcecore*rcore).item()*spherical_yn(l,
    kceshell*rcore,derivative=True).item()-((Mceshell*kcecore)/(Mcecore*
    kceshell))*spherical_jn(l,kcecore*rcore,derivative=True).item()*
    spherical_yn(l,kceshell*rcore).item())
67     denomceshell=spherical_jn(l,kceshell*rcore).item()*spherical_yn(l,
    kceshell*rcore,derivative=True).item()-spherical_jn(l,kceshell*rcore,
    derivative=True).item()*spherical_yn(l,kceshell*rcore).item()
68     #
69     #print((kceshell*rcore).imag,spherical_in(l,(kceshell*rcore).imag).
    item(),spherical_kn(l,(kceshell*rcore).imag),spherical_in(l,kceshell*
    rcore,derivative=True),spherical_kn(l,kceshell*rcore,derivative=True).
    tolist(),denomceshell)
70     Aceshell=Acecore*(numceshell/denomceshell)
71     return(Aceshell)
72 #
73
74 # defining B_CE_shell
75 def B_CE_shell(Acecore,l,kcecore,kceshell,rcore,Mcecore,Mceshell):
76     numceshell=(spherical_jn(l,kcecore*rcore).item()*spherical_jn(l,
    kceshell*rcore,derivative=True).item()-((Mceshell*kcecore)/(Mcecore*
    kceshell))*spherical_jn(l,kcecore*rcore,derivative=True).item()*
    spherical_jn(l,kceshell*rcore).item())
77     denomceshell=spherical_jn(l,kceshell*rcore,derivative=True).item()*
    spherical_yn(l,kceshell*rcore).item()-spherical_jn(l,kceshell*rcore).
    item()*spherical_yn(l,kceshell*rcore,derivative=True).item()
78     Bceshell=Acecore*(numceshell/denomceshell)
79     return(Bceshell)
80 #
81
82 # defining A_VH_shell
83 def A_VH_shell(Avhcore,l,kvhcore,kvhshell,rcore,Mvhcore,Mvhshell):
84     numvhshell=(spherical_jn(l,kvhcore*rcore).item()*spherical_yn(l,
    kvhshell*rcore,derivative=True).item()-((Mvhshell*kvhcore)/(Mvhcore*
```

```
    kvhshell))*spherical_jn(l,kvhcore*rcore,derivative=True).item()*
    spherical_yn(l,kvhshell*rcore).item())
85      denomvhshell=spherical_jn(l,kvhshell*rcore).item()*spherical_yn(l,
    kvhshell*rcore,derivative=True).item()-spherical_jn(l,kvhshell*rcore,
    derivative=True).item()*spherical_yn(l,kvhshell*rcore).item()
86      Avhshell=Avhcore*(numvhshell/denomvhshell)
87      return(Avhshell)
88  #
89
90  # defining B_CE_shell
91  def B_VH_shell(Avhcore,l,kvhcore,kvhshell,rcore,Mvhcore,Mvhshell):
92      numvhshell=(spherical_jn(l,kvhcore*rcore).item()*spherical_jn(l,
    kvhshell*rcore,derivative=True).item()-((Mvhshell*kvhcore)/(Mvhcore*
    kvhshell))*spherical_jn(l,kvhcore*rcore,derivative=True)*spherical_jn(l
    ,kvhshell*rcore).item())
93      denomvhshell=spherical_jn(l,kvhshell*rcore,derivative=True).item()*
    spherical_yn(l,kvhshell*rcore).item()-spherical_jn(l,kvhshell*rcore).
    item()*spherical_yn(l,kvhshell*rcore,derivative=True).item()
94      Bvhshell=Avhcore*(numvhshell/denomvhshell)
95      return(Bvhshell)
96  #
```

Step 4 : Find Allowed Energy Values

To generate the complete wavefunction, we first need to know the allowed values of energy, $\varepsilon_{e/h}^{c/v}$. We have already enforced the first two boundary conditions, given in equations (A.25, A.26). We now need to enforce the last boundary condition in equation (A.27) numerically. For that, we will first choose a value of l and generate $\psi_{e/h}^{c/v,shell}$ for continuous values of energy. We then plot the generated data, and identify the points where the wavefunction goes to 0. We then use the identified points as guess root values and find the exact roots of the wavefunction using root-finding algorithms, like Newton-Rhapson. The lowest root value for each l is our required energy value.

I define two separate functions for conduction electrons and valence holes. I also store the wavefunction generated as function of energy as text files, and plot them using Gnuplot. Finally, I store the identified energy values for each l for both conduction electrons and valence holes in a text file as well.

```
# ---------------------- CONDUCTION ELECTRONS ----------------------- #
# defining big file to store roots of CE wavefunctions
nmrootsCE=path+"\\Wavefunctions\\"+f'CE_Roots(B3).txt'
frootsCE=open(nmrootsCE, "w")
frootsCE.close()
# defining function to call for generating roots
def CE_rootsfunc(lwave):
    # defining constants and logistics about terms to generate (step sizes, initial points, final points, number of terms)
    hE=10**(-3) #step size in energy (in eV)
    E0=-0.1
    NE=200
    # first need to define a simple function of psidiff as E to easily call Newton Rhapson later
    def psiCEShellasE(E):
        Vceshell=Vc(rshell,rcore,rshell) #potential for the iteration
        Vcecore=Vc(rcore,rcore,rshell)
        kcecore0=k(E,Vcecore,mecore)
        kceshell0=k(E,Vceshell,meshell)
        Aceshell0=A_CE_shell(1,lwave,kcecore0,kceshell0,rcore,mecore,meshell)
        Bceshell0=B_CE_shell(1,lwave,kcecore0,kceshell0,rcore,mecore,meshell)
        psiceshell0=psi_CE_shell(Aceshell0,Bceshell0,lwave,kceshell0,rshell)
        return(psiceshell0)
    #
    # naming files ==> may not use always in the code
    nmdiffCE=path+"\\Wavefunctions\\"+"\\EnergyVals\\"+f'CE_l(B3)={lwave}.txt'
    fdiffCE=open(nmdiffCE,"w")
```

```
26      fdiffCE.close()
27      #now calling the above function => we will only do this to generate
    plots (rest of the time, this will be turned off)
28      for iCE in range(NE):
29          E=E0+(iCE**(3/2))*hE #defines E for that iteration
30          psival=psiCEShellasE(E)
31          append_file(nmdiffCE,f'{E} {psival.real} {psival.imag}\n')
32      #
33      # from plots, we define a guess root, then call Newton Rhapson
34      maxiterCE=1000
35      trialrootCEReal=float(input(f'Please enter value of trial root for l={
    lwave} (CE-Real).\n'))
36      nmCEerr=path+"\\Wavefunctions\\"+"\\EnergyVals\\"+f'CE_err_l={lwave}.
    txt'
37      fCEerr=open(nmCEerr,"w")
38      fCEerr.close()
39      #as the function outputs a value and a bool function, the first term
    is the root (if found), the second is the bool
40      ErootCE,ErootCEbool=newton_raphson(psiCEShellasE,trialrootCEReal,
    maxiterCE,nmCEerr)
41      #ErootCEImag,ErootCEboolImag=newton_raphson(psiCEShellasEImag,
    trialrootCEReal,maxiterCE,nmCEerr)
42      if ErootCEbool==True:
43          #print(f'\nRoot found for CE, and the value is given as = {ErootCE
    }\n')
44          append_file(nmrootsCE, f'{lwave} {ErootCE}\n')
45      #
46      else: print(f'\nRoot not found.\n')
47      return(0)
48  #
49
50  # ---------------------- VALENCE HOLES ---------------------------- #
51  # defining big file to store roots of CE wavefunctions
52  nmrootsVH=path+"\\Wavefunctions\\"+f'VH_Roots(B3).txt'
53  frootsVH=open(nmrootsVH, "w")
54  frootsVH.close()
```

```
def VH_rootsfunc(lwave):
    # defining constants and logistics about terms to generate (step sizes, initial points, final points, number of terms)
    hE=10**(-4) #step size in energy (in eV)
    E0=-0.65
    NE=500
    # first need to define a simple function of psidiff as E to easily call Newton Rhapson later
    def psiVHShellasE(E):
        Vvhshell=Vv(rshell,rcore,rshell) #potential for the iteration
        Vvhcore=Vv(rcore,rcore,rshell)
        kvhcore0=k(E,Vvhcore,mhcore)
        kvhshell0=k(E,Vvhshell,mhshell)
        Avhshell0=A_VH_shell(1,lwave,kvhcore0,kvhshell0,rcore,mhcore,mhshell)
        Bvhshell0=B_VH_shell(1,lwave,kvhcore0,kvhshell0,rcore,mhcore,mhshell)
        psivhshell0=psi_VH_shell(Avhshell0,Bvhshell0,lwave,kvhshell0,rshell)
        return(psivhshell0.real)
    #
    # naming files ==> may not use always in the code
    nmdiffVH=path+"\\Wavefunctions\\"+"\\EnergyVals\\"+f'VH_1(B3)={lwave}.txt'
    fdiffVH=open(nmdiffVH,"w")
    fdiffVH.close()
    #now calling the above function => we will only do this to generate plots (rest of the time, this will be turned off)
    for iVH in range(NE):
        E=E0+(iVH**(3/2))*hE #defines E for that iteration
        psival=psiVHShellasE(E)
        append_file(nmdiffVH,f'{E} {psival.real} {psival.imag}\n')
    #
    # from plots, we define a guess root, then call Newton Rhapson
    maxiterVH=1000
```

```
    trialrootVH=float(input(f'Please enter value of trial root for l={lwave} (VH).\n'))
    nmVHerr=path+"\\Wavefunctions\\"+"\\EnergyVals\\"+f'VH_err_l={lwave}.txt'
    fVHerr=open(nmVHerr,"w")
    fVHerr.close()
    #as the function outputs a value and a bool function, the first term is the root (if found), the second is the bool
    ErootVH,ErootVHbool=newton_raphson(psiVHShellasE,trialrootVH,maxiterVH,nmVHerr)
    if ErootVHbool==True:
        #print(f'\nRoot found for CE, and the value is given as = {ErootCE}\n')
        append_file(nmrootsVH, f'{lwave} {ErootVH}\n')
    #
    else: print(f'\nRoot not found.\n')
    return(0)
#

# now, we can generate roots for any of the two ==> CE of VH
lmax=4
for lw in range(lmax):
    ceroots=CE_rootsfunc(lw)
    vhroots=VH_rootsfunc(lw)
#
# ------------------------------------------------ #
end=time.time()
print(f'Time taken for code to run = {end-begin}\n')
```

The wavefunction vs energy graphs are obtained as follows :

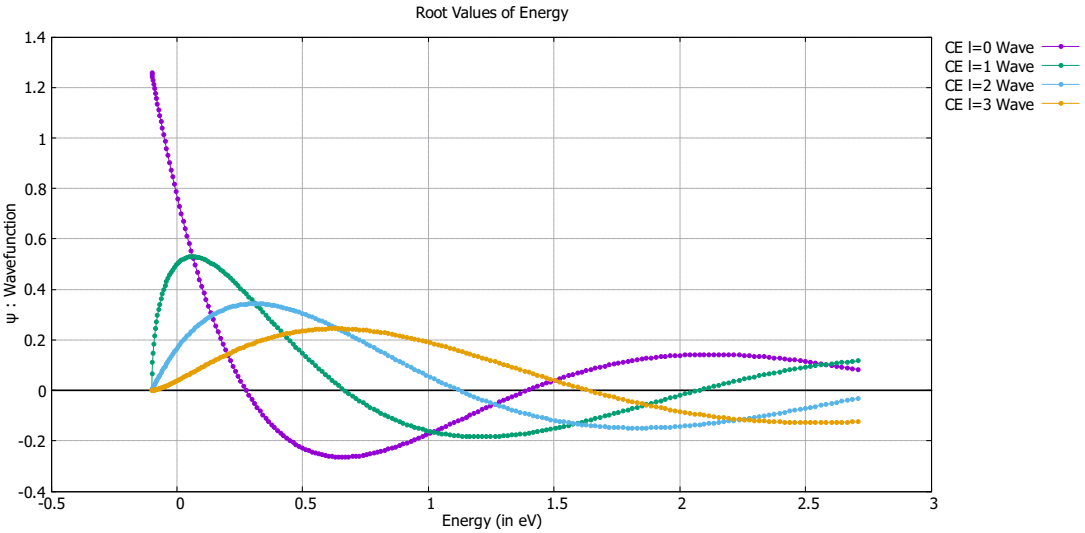

Figure A.2. Conduction electrons wavefunction vs energy

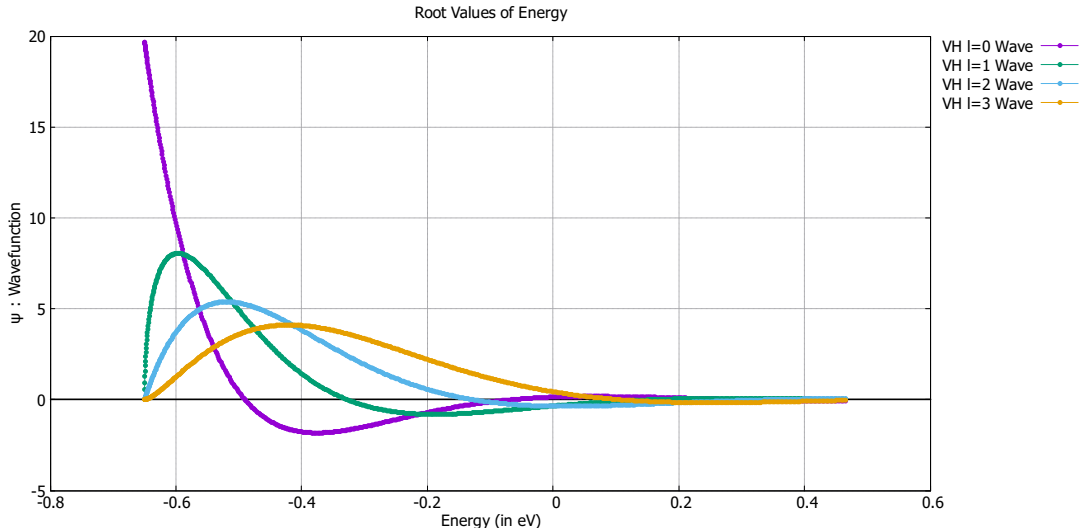

Figure A.3. Valence holes wavefunction vs energy

Step 5 : Generating wavefunctions

Now that we have found the allowed energy for each l, we can build the complete wavefunction as a function of r. I import the energy values, define a small step size for r in natural units, normalize the wavefunction first, then store the normalized wavefunction data in a text file.

```
# Generating wavefunctions of Quantum Dot

import math
import cmath
import sys
import time
import os
import numpy as np
import scipy
from scipy.special import spherical_jn
from scipy.special import spherical_yn
from scipy.special import spherical_in
from scipy.special import spherical_kn


# all external functions needed for the code

# READ FILE
def read_matrix(x): #more than one column #parameter: x ==> name of file
    f=open(x,'r') #'r' ==> read only
    X=[[float(num) for num in line.split(' ')] for line in f]
    f.close()
    return(X)
#

# APPEND FILE
def append_file(name, str): #arguments: name ==> name of file, str ==>
   string to append
    f=open(name, 'a') #'a' ==> append file
    f.write(str)
    f.close()
    #
#

#MOD SQUARE OF A COMPLEX NUMBER
def modulus(x): #x ==> complex number
```

```
36      real=x.real**2
37      imag=x.imag**2
38      modsq=real+imag
39      return(modsq)
40  #
41
42  # DERIVATIVE
43  def derivative(f, x, h): # arguments = equation, point at which derivative
        is to be found, h
44      y=(f(x+h)-f(x-h))/(2*h)
45      return(y)
46  #
47
48  # NEWTON RAPHSON
49  def newton_raphson(f, a, max_itr, nm): #arguments ==> formula, trial root,
        maximum iterations, name of file to store errors
50      err=pow(10, -9)
51      n=0
52      X=[0 for i in range(max_itr)]
53      while n<max_itr: #so that it does not cross max number of allowed
        iterations
54          if n==0:
55              X[n]=a
56          else:
57              h=X[n-1]*0.001
58              X[n]=X[n-1]-(f(X[n-1])/derivative(f, X[n-1], h))
59              #error = x_{n+1}-x_{n} o equivalently, x_{n}-x_{n-1}
60              append_file(nm, f'{n} {abs(X[n] - X[n - 1])}\n') #appends
        absolute error value with iteration number to file
61              if abs(X[n]-X[n-1])<err: return(X[n], True)
62              #
63          n+=1
64          #
65      return(X[n-1], False) #if root not obtained even after max_iterations
66  #
67
```

```
68 # Integration
69 def simpson(f, lower, upper, h, name):
70     N=int(abs(upper-lower)/h)
71     I=0
72     X=[0 for i in range(N+1)] #for storing N+1 values, including upper and
       lower
73     X[0]=lower
74     for i in range(1,N+1):
75         X[i]=lower+h*i #finding the next point
76         mid=(X[i-1]+X[i])/2 #finding mid value
77         h1=(X[i]-X[i-1])/2  #diving every slice in two equal slices, and
      finding h relevant to that slice
78         #calculating the integration
79         I=I+(h1/3)*(f(X[i-1])+4*f(mid)+f(X[i]))
80         #
81     #storing N vs I value in file
82     append_file(name, f'{N} {I}\n')
83     return(I)
84 #
85
86
87 #------------------- MAIN CODE STARTS ----------------------#
88
89 #clock beginning time to measure time taken by code to run
90 begin=time.time()
91
92 #setting path of file
93 path=os.path.dirname(os.path.realpath(__file__))
94 sys.path.append(path)
95 # ------------------------------------------------------------------- #
96
97 #defining all constants => masses (natural units, electron volts)
98 hbar=1.0
99 m0=0.511*(10**6) #rest mass of electon in natural units
100 mecore=0.11*m0
101 meshell=0.14*m0
```

```
mhcore=0.8*m0
mhshell=0.51*m0
# ------------------------------------------------------------------------
   #

#defining all constants => radii (in nanometers)
rcorenm=1.5
shellthicknessnm=1.4
rshellnm=rcorenm+shellthicknessnm
# converting to natural units
confac=5.066*(10**(-3))
rcore=confac*rcorenm
rshell=confac*rshellnm
wavefuncsqnmcon=1/((1.974*100)**(3))
# ------------------------------------------------------------------------ #

#defining "k" as a function of energy, potential and mass (for e/h in c/v
   bands respectively)
def k(E,V,M):
    hbar=1.0
    if (E-V)>=0 : kval=math.sqrt((2*(E-V)*M)/(hbar**2))
    else : kval=cmath.sqrt((2*(E-V)*M)/(hbar**2))
    #print(f'E-V={E-V}\n k={kval}')
    return(kval)
#
# ------------------------------------------------------------------------ #

#defining potential for conduction (for e) band as a function of r (in eV)
def Vc(r,core,shell):
    #defining all constants => potentials (natural units, electron volts)
    Vcshell=0.0
    Vccore=-0.1 #wrt Vcshell=0
    #note, Vccore and Vvcore are not zero wrt each other
    #
    if abs(r)<=core: V=Vccore
    elif abs(r)>core and abs(r)<=shell: V=Vcshell
```

```
136     else: V=10**15 #tends to infinity
137     return(V)
138 #
139 
140 #defining potential for valence (for h) band as a function of r (in eV)
141 def Vv(r,core,shell):
142     #defining all constants => potentials (natural units, electron volts)
143     Vvcore=-0.65 #wrt Vcshell=0
144     Vvshell=0.0
145     #note, Vccore and Vvcore are not zero wrt each other
146     #
147     if abs(r)<=core: V=Vvcore
148     elif abs(r)>core and abs(r)<=shell: V=Vvshell
149     else: V=-10**15 #tends to infinity
150     return(V)
151 #
152 
153 # ---------------------------------- functions for wavefunction (CE and VH)
    ------------------------------------------ #
154 
155 # defining wavefunction of core for electrons (in conduction band)
156 def psi_CE_core(Acecore,l,kcecore,r):
157     psicecore=Acecore*spherical_jn(l, kcecore*r)
158     return(psicecore)
159 #
160 
161 # defining wavefunction of shell for electrons (in conduction band)
162 def psi_CE_shell(Aceshell,Bceshell,l,kceshell,r):
163     psiceshell=Aceshell*spherical_jn(l,kceshell*r)+Bceshell*spherical_yn(l
    ,kceshell*r)
164     return(psiceshell)
165 #
166 
167 #defining wavefunction of core for holes (in valence band)
168 def psi_VH_core(Avhcore,l,kvhcore,r):
169     psivhcore=Avhcore*spherical_jn(l,kvhcore*r)
```

```
    return(psivhcore)
#

# defining wavefunction for shell for holes (in valence band)
def psi_VH_shell(Avhshell,Bvhshell,l,kvhshell,r):
    psivhshell=Avhshell*spherical_jn(l,kvhshell*r)+Bvhshell*spherical_yn(l
    ,kvhshell*r)
    return(psivhshell)
#
# ---------------------------------------- A,B functions (CE and VH)
    ------------------------------------------------- #

# defining A_CE_shell
def A_CE_shell(Acecore,l,kcecore,kceshell,rcore,Mcecore,Mceshell):
    numceshell=(spherical_jn(l,kcecore*rcore).item()*spherical_yn(l,
    kceshell*rcore,derivative=True).item()-((Mceshell*kcecore)/(Mcecore*
    kceshell))*spherical_jn(l,kcecore*rcore,derivative=True).item()*
    spherical_yn(l,kceshell*rcore).item())
    denomceshell=spherical_jn(l,kceshell*rcore).item()*spherical_yn(l,
    kceshell*rcore,derivative=True).item()-spherical_jn(l,kceshell*rcore,
    derivative=True).item()*spherical_yn(l,kceshell*rcore).item()
    #
    #print((kceshell*rcore).imag,spherical_in(l,(kceshell*rcore).imag).
    item(),spherical_kn(l,(kceshell*rcore).imag),spherical_in(l,kceshell*
    rcore,derivative=True),spherical_kn(l,kceshell*rcore,derivative=True).
    tolist(),denomceshell)
    Aceshell=Acecore*(numceshell/denomceshell)
    return(Aceshell)
#

# defining B_CE_shell
def B_CE_shell(Acecore,l,kcecore,kceshell,rcore,Mcecore,Mceshell):
    numceshell=(spherical_jn(l,kcecore*rcore).item()*spherical_jn(l,
    kceshell*rcore,derivative=True).item()-((Mceshell*kcecore)/(Mcecore*
    kceshell))*spherical_jn(l,kcecore*rcore,derivative=True).item()*
    spherical_jn(l,kceshell*rcore).item())
```

```
193     denomceshell=spherical_jn(l,kceshell*rcore,derivative=True).item()*
    spherical_yn(l,kceshell*rcore).item()-spherical_jn(l,kceshell*rcore).
    item()*spherical_yn(l,kceshell*rcore,derivative=True).item()
194     Bceshell=Acecore*(numceshell/denomceshell)
195     return(Bceshell)
196 #
197
198 # defining A_VH_shell
199 def A_VH_shell(Avhcore,l,kvhcore,kvhshell,rcore,Mvhcore,Mvhshell):
200     numvhshell=(spherical_jn(l,kvhcore*rcore).item()*spherical_yn(l,
    kvhshell*rcore,derivative=True).item()-((Mvhshell*kvhcore)/(Mvhcore*
    kvhshell))*spherical_jn(l,kvhcore*rcore,derivative=True).item()*
    spherical_yn(l,kvhshell*rcore).item())
201     denomvhshell=spherical_jn(l,kvhshell*rcore).item()*spherical_yn(l,
    kvhshell*rcore,derivative=True).item()-spherical_jn(l,kvhshell*rcore,
    derivative=True).item()*spherical_yn(l,kvhshell*rcore).item()
202     Avhshell=Avhcore*(numvhshell/denomvhshell)
203     return(Avhshell)
204 #
205
206 # defining B_CE_shell
207 def B_VH_shell(Avhcore,l,kvhcore,kvhshell,rcore,Mvhcore,Mvhshell):
208     numvhshell=(spherical_jn(l,kvhcore*rcore).item()*spherical_jn(l,
    kvhshell*rcore,derivative=True).item()-((Mvhshell*kvhcore)/(Mvhcore*
    kvhshell))*spherical_jn(l,kvhcore*rcore,derivative=True)*spherical_jn(l
    ,kvhshell*rcore).item())
209     denomvhshell=spherical_jn(l,kvhshell*rcore,derivative=True).item()*
    spherical_yn(l,kvhshell*rcore).item()-spherical_jn(l,kvhshell*rcore).
    item()*spherical_yn(l,kvhshell*rcore,derivative=True).item()
210     Bvhshell=Avhcore*(numvhshell/denomvhshell)
211     return(Bvhshell)
212 #
213 # ---------------------------------------- defining constants and building
    up wavefunctions ---------------------------------------- #
214
```

```
215 #defining logistics about terms to generate (step sizes, initial points,
    final points, number of terms)
216 hr=10**(-3)*confac #step size in r (in converted units)
217 hE=10**(-7) #step size in energy (in eV)
218 r0=0.0
219 E0=10**(-10)
220 rf=confac*3.0
221 Ef=0.0002
222 Nr=int(math.ceil((rf-r0)/hr))
223 NE=int(math.ceil((Ef-E0)/hE))
224 #
    ------------------------------------------------------------------------------#


225
226 # We need to import the value of l and correponding roots from the file.
227
228 # ---------------------------------------- CONDUCTION ELECTRONS
    ---------------------------------------------- #
229 nmrootsCE=path+"\\Wavefunctions\\"+f'CE_Roots(B3)_final.txt'
230 frootsCE=open(nmrootsCE,'r')
231 RootsCE=[[float(num) for num in line.split(' ')] for line in frootsCE] #
    read entire file
232
233 #After this, all functions will be in a for loops. Outermost loops will be
     "l" and "m" which define the orders.
234 #Then, the loops will be for "r" and "E". This is the final generation of
    data section.
235
236 for itr in range(len(RootsCE)):
237     lwave=RootsCE[itr][0]
238     ErootCE=round(RootsCE[itr][1],4)
239     print(lwave, ErootCE)
240     #
241     #Next, we need to normalise the complete wavefunction to find Acore,
    which will allow us to obtain all Ace and Bce, thus psice (first for
    conduction electrons)
```

```
242     # for this, we need a function which squares psicoreCE*r^2 for given
    root energy value
243     def psiCEmodsquare(rce):
244         if abs(rce)<=rcore:
245             # defining potential and K for iteration
246             Vce=Vc(rce,rcore,rshell) #potential for the iteration
247             kcecoreMod=k(ErootCE,Vce,mecore) # k for iteration
248             psiCErcoresquare=(rce**2)*modulus(psi_CE_core(1,lwave,
    kcecoreMod,rce))*4*math.pi
249             #print(Vce,kcecoreMod,psiCErsquare)
250             return(psiCErcoresquare)
251         #
252         elif abs(rce)>rcore and abs(rce)<=rshell:
253             VceshellOO=Vc(rce,rcore,rshell) #potential for the iteration
254             VcecoreOO=Vc(rcore,rcore,rshell) #constant potential of core
255             kcecoreOO=k(ErootCE,VcecoreOO,mecore)
256             kceshellOO=k(ErootCE,VceshellOO,meshell)
257             AceshellOO=A_CE_shell(1,lwave,kcecoreOO,kceshellOO,rcore,
    mecore,meshell)
258             BceshellOO=B_CE_shell(1,lwave,kcecoreOO,kceshellOO,rcore,
    mecore,meshell)
259             psiCErshellsquare=(rce**2)*modulus(psi_CE_shell(AceshellOO,
    BceshellOO,lwave,kceshellOO,rce))*4*math.pi
260             return(psiCErshellsquare)
261         #
262         else: return(0)
263     #
264     #now calling the numerical integration function
265     nmpsiCEnorm=path+"\\Wavefunctions\\"+f'Normalisation of psiCEcore.txt'
266     fpsiCEnorm=open(nmpsiCEnorm,"w")
267     fpsiCEnorm.close()
268     IntCEcore=simpson(psiCEmodsquare,r0,rshell,hr,nmpsiCEnorm)
269     #print(IntCEcore)
270     AcoreCE=math.sqrt(1/IntCEcore)
271     print(f'Normalization constant AcoreCE = {AcoreCE}\n')
272     #
```

```
273     #now generating the complete wavefunction for given lwave, CE for a
    given energy value =>ErootCE
274     # we will also generate |psi|^2
275     nmCEwave=path+"\\Wavefunctions\\"+f'CE_wavefunc_l={int(lwave)}.txt'
276     fCEwave=open(nmCEwave,"w")
277     fCEwave.close()
278     for rvals in range(Nr):
279         r=r0+rvals*hr
280         if abs(r)<=rcore:
281             VcecoreOO=Vc(r,rcore,rshell) #potential for the iteration
282             kcecoreOO=k(ErootCE,VcecoreOO,mecore)
283             psiCErcore=psi_CE_core(AcoreCE,lwave,kcecoreOO,r)
284             append_file(nmCEwave,f'{r/confac} {psiCErcore.real*(
    wavefuncsqnmcon**(1/2))} {psiCErcore.imag*(wavefuncsqnmcon**(1/2))} {
    modulus(psiCErcore)*(wavefuncsqnmcon)}\n')
285         #
286         elif abs(r)>rcore and abs(r)<=rshell:
287             VceshellOO=Vc(r,rcore,rshell) #potential for the iteration
288             VcecoreOO=Vc(rcore,rcore,rshell) #constant potential of core
289             kcecoreOO=k(ErootCE,VcecoreOO,mecore)
290             kceshellOO=k(ErootCE,VceshellOO,meshell)
291             AceshellOO=A_CE_shell(AcoreCE,lwave,kcecoreOO,kceshellOO,rcore
    ,mecore,meshell)
292             BceshellOO=B_CE_shell(AcoreCE,lwave,kcecoreOO,kceshellOO,rcore
    ,mecore,meshell)
293             psiCErshell=psi_CE_shell(AceshellOO,BceshellOO,lwave,
    kceshellOO,r)
294             append_file(nmCEwave,f'{r/confac} {psiCErshell.real*(
    wavefuncsqnmcon**(1/2))} {psiCErshell.imag*(wavefuncsqnmcon**(1/2))} {
    modulus(psiCErshell)*(wavefuncsqnmcon)}\n')
295         #
296         else: append_file(nmCEwave,f'{r/confac} {0} {0} {0}\n')
297     #
298 #
299
```

```
#
    ------------------------------------------------------------------------
     VALENCE HOLES
    ------------------------------------------------------------------ #

nmrootsVH=path+"\\Wavefunctions\\"+f'VH_Roots(B3)_final.txt'
frootsVH=open(nmrootsVH,'r')
RootsVH=[[float(num) for num in line.split(' ')] for line in frootsVH] #
    read entire file

#After this, all functions will be in a for loops. Outermost loops will be
     "l" and "m" which define the orders.
#Then, the loops will be for "r" and "E". This is the final generation of
    data section.

for itr in range(len(RootsVH)):
    lwave=RootsVH[itr][0]
    ErootVH=round(RootsVH[itr][1],4)
    print(lwave, ErootVH)
    #
    #Next, we need to normalise the core wavefunction to find Acore, which
     will allow us to obtain all Ace and Bce, thus psice (first for
    conduction electrons)
    # for this, we need a function which squares psicoreCE*r^2 for given
    root energy value
    def psicoreVHmodsquare(rvh):
        if abs(rvh)<=rcore:
            # defining potential and K for iteration
            Vvh=Vv(rvh,rcore,rshell) #potential for the iteration
            kvhcoreMod=k(ErootVH,Vvh,mhcore) # k for iteration
            psiVHrcoresquare=(rvh**2)*modulus(psi_VH_core(1,lwave,
    kvhcoreMod,rvh))*4*math.pi
            return(psiVHrcoresquare)
        #
        elif abs(rvh)>rcore and abs(rvh)<=rshell:
            VvhshellOO=Vv(rvh,rcore,rshell) #potential for the iteration
```

```
            VvhcoreOO=Vv(rcore,rcore,rshell) #constant potential of core
            kvhcoreOO=k(ErootVH,VvhcoreOO,mhcore)
            kvhshellOO=k(ErootVH,VvhshellOO,mhshell)
            AvhshellOO=A_VH_shell(1,lwave,kvhcoreOO,kvhshellOO,rcore,
mhcore,mhshell)
            BvhshellOO=B_VH_shell(1,lwave,kvhcoreOO,kvhshellOO,rcore,
mhcore,mhshell)
            psiVHrshellsquare=(rvh**2)*modulus(psi_VH_shell(AvhshellOO,
BvhshellOO,lwave,kvhshellOO,rvh))*4*math.pi
            return(psiVHrshellsquare)
        #
        else: return(0)
    #
    #now calling the numerical integration function
    nmpsiVHnorm=path+"\\Wavefunctions\\"+f'Normalisation of psiVHcore.txt'
    fpsiVHnorm=open(nmpsiVHnorm,"w")
    fpsiVHnorm.close()
    IntVHcore=simpson(psicoreVHmodsquare,r0,rshell,hr,nmpsiVHnorm)
    AcoreVH=math.sqrt(1/IntVHcore)
    print(f'Normalization constant AcoreVH = {AcoreVH}\n')
    #
    #now generating the complete wavefunction for given lwave, CE for a
given energy value =>ErootCE
    # we will also generate |psi|^2
    nmVHwave=path+"\\Wavefunctions\\"+f'VH_wavefunc_l={int(lwave)}.txt'
    fVHwave=open(nmVHwave,"w")
    fVHwave.close()
    for rvals in range(Nr):
        r=r0+rvals*hr
        if abs(r)<=rcore:
            VvhcoreOO=Vv(r,rcore,rshell) #potential for the iteration
            kvhcoreOO=k(ErootVH,VvhcoreOO,mhcore)
            psiVHrcore=psi_VH_core(AcoreVH,lwave,kvhcoreOO,r)
            append_file(nmVHwave,f'{r/confac} {psiVHrcore.real*(
wavefuncsqnmcon**(1/2))} {psiVHrcore.imag*(wavefuncsqnmcon**(1/2))} {
modulus(psiVHrcore)*(wavefuncsqnmcon)}\n')
```

```
        #
        elif abs(r)>rcore and abs(r)<=rshell:
            VvhshellOO=Vv(r,rcore,rshell) #potential for the iteration
            VvhcoreOO=Vv(rcore,rcore,rshell) #constant potential of core
            kvhcoreOO=k(ErootVH,VvhcoreOO,mhcore)
            kvhshellOO=k(ErootVH,VvhshellOO,mhshell)
            AvhshellOO=A_VH_shell(AcoreVH,lwave,kvhcoreOO,kvhshellOO,rcore
    ,mhcore,mhshell)
            BvhshellOO=B_VH_shell(AcoreVH,lwave,kvhcoreOO,kvhshellOO,rcore
    ,mhcore,mhshell)
            psiVHrshell=psi_VH_shell(AvhshellOO,BvhshellOO,lwave,
    kvhshellOO,r)
            append_file(nmVHwave,f'{r/confac} {psiVHrshell.real*(
    wavefuncsqnmcon**(1/2))} {psiVHrshell.imag*(wavefuncsqnmcon**(1/2))} {
    modulus(psiVHrshell)*(wavefuncsqnmcon)}\n')
        #
        else: append_file(nmVHwave,f'{r/confac} {0} {0} {0}\n')
     #
#

# ------------------------------------------------ #
end=time.time()
print(f'Time taken for code to run = {end-begin}\n')
```

A.4 Corrected Wavefunction with Coulomb Interaction Perturbation : Computational Solution

Once we have generated the unperturbed wavefunctions, we can generate the corrected wavefunctions for Coulomb interaction perturbation by numerically solving for equations (2.28, 2.29).

```
# python file to generate corrections due to energy of quantum dot due to
    perturbations
import math
import cmath
import sys
```

```
import time
import os
import numpy as np
import scipy
from scipy.special import spherical_jn
from scipy.special import spherical_yn
from scipy.special import spherical_in
from scipy.special import spherical_kn

# all external functions needed for the code

# READ FILE
def read_matrix(x): #more than one column #parameter: x ==> name of file
    f=open(x,'r') #'r' ==> read only
    X=[[float(num) for num in line.split(' ')] for line in f]
    f.close()
    return(X)
#

# APPEND FILE
def append_file(name, str): #arguments: name ==> name of file, str ==>
   string to append
    f=open(name, 'a') #'a' ==> append file
    f.write(str)
    f.close()
    #
#

#MOD SQUARE OF A COMPLEX NUMBER
def modulus(x): #x ==> complex number
   real=x.real**2
   imag=x.imag**2
   modsq=real+imag
   return(modsq)
#

```

```
#MOD SQUARE OF A COMPLEX NUMBER
def modulus(x): #x ==> complex number
   real=x.real**2
   imag=x.imag**2
   modsq=real+imag
   return(modsq)
#

# Integration
def simpson(f, lower, upper, h, name):
    N=int(abs(upper-lower)/h)
    I=0
    X=[0 for i in range(N+1)] #for storing N+1 values, including upper and
    lower
    X[0]=lower
    for i in range(1,N+1):
        X[i]=lower+h*i #finding the next point
        mid=(X[i-1]+X[i])/2 #finding mid value
        h1=(X[i]-X[i-1])/2  #diving every slice in two equal slices, and
   finding h relevant to that slice
        #calculating the integration
        I=I+(h1/3)*(f(X[i-1])+4*f(mid)+f(X[i]))
        #
    #storing N vs I value in file
    append_file(name, f'{N} {I}\n')
    return(I)
#

# sorting array with respect to certain column ==> bubble algortihm
def arraysort(X,col):
    n=len(X)
    ncol=len(X[0])
    for i in range(n):
        for j in range(i+1,n):
            if (X[i][col]>X[j][col]):
                for k in range(ncol) :
```

```
                    temp=X[i][k]
                    X[i][k]=X[j][k]
                    X[j][k]=temp
                #
            #
        #
    #
    return(X)
#

# SHIFT VALUES OF MATRIX BY MULTIPLCATION ==> multiply some constant/ variable to terms of a matrix
def shift_matrix_multiply(M,val,col): #arguments: M ==> original matrix which needs shifting, val ==> value to add to the matrix (all terms), col ==> which column of the matrix to multiply (if 1D matrix, then col=0)
    n=len(M) #number of rows
    if col==0:
        CopyM=[0 for i in range(n)]
        for i in range(n):
            CopyM[i]=M[i]*val
    #
    else:
        m=len(M[0])
        CopyM=[[0 for j in range(m)] for i in range(n)]
        for i in range(n):
            for j in range(m):
                if j ==col : CopyM[i][col]=M[i][col]*val
                else : CopyM[i][j]=M[i][j]
            #
        #
    #
    return(CopyM)
#

```

```
105 #------------------------------------------------- Main Code Starts
    -------------------------------------------------#
106
107 #clock beginning time to measure time taken by code to run
108 begin=time.time()
109
110 #setting path of file
111 path=os.path.dirname(os.path.realpath(__file__))
112 sys.path.append(path)
113 # ------------------------------------------------------------------- #
114
115 # defining constants in natural units
116 confac=5.066*(10**(-3))
117 epsilon0=1
118 epsilon=5.367
119 k=(1/(4*math.pi*epsilon0*epsilon))
120 k0=9*(10**9)
121 el0=1.602*(10**(-19))
122 elconfac=1.0/(5.289*(10**(-19)))
123 el=el0*elconfac
124 wavefuncsqnmcon=((1.974*100)**(3))
125 Jouletoev=1/(1.602*(10**(-19)))
126
127 # reading energy from files
128 nmexcitonEN=path+"\\Wavefunctions\\"+'Energy of Exciton.txt'
129 fecxitonEN=open(nmexcitonEN,'r')
130 header_line=next(fecxitonEN) # ignoring the first line of the file
131 EnergyVals=[[float(num) for num in line.split(' ')] for line in fecxitonEN
    ] #read entire file
132 fecxitonEN.close()
133
134 # as we only careabout the first two energies for the first order energy
    correction, we will store them
135 E0=EnergyVals[0][4]
136 E1=EnergyVals[1][4]
137
```

```
# we only care about the ground state, so we will import only that
lCE=0
lVH=0
# ground state
nmCE=path+"\\Wavefunctions\\"+f'CE_wavefunc_l={lCE}.txt'
fileCE=open(nmCE,'r')
# storing data in array
CE=[[float(num) for num in line.split(' ')] for line in fileCE] #0 :
    position ; 3 : |\psi|^2 # l=0 for CE
fileCE.close()
nmVH=path+"\\Wavefunctions\\"+f'VH_wavefunc_l={lVH}.txt'
fileVH=open(nmVH,'r')
# storing data in array
VH=[[float(num) for num in line.split(' ')] for line in fileVH] #0 :
    position ; 3 : |\psi|^2 # l=0 for VH
fileVH.close()
#
# first excited state
nmCE1=path+"\\Wavefunctions\\"+f'CE_wavefunc_l={lCE}.txt'
fileCE1=open(nmCE1,'r')
# storing data in array
CE1=[[float(num) for num in line.split(' ')] for line in fileCE1] #0 :
    position ; 3 : |\psi|^2 # l=0 for CE
fileCE1.close()
nmVH1=path+"\\Wavefunctions\\"+f'VH_wavefunc_l={lVH+1}.txt'
fileVH1=open(nmVH1,'r')
# storing data in array
VH1=[[float(num) for num in line.split(' ')] for line in fileVH1] #0 :
    position ; 3 : |\psi|^2 # l=0 for VH
fileVH1.close()


# ------------------------------------------------------------------
    FIRST ORDER ENERGY CORRECTION
    ----------------------------------------------------#
```

```
# performing integration over CE by keeping VH constant at r=0 => ground
    state
psiVH=VH[0][3]
#print(psiVH)
Int=0.0 # total integration value for CE
delr=CE[1][0]-CE[0][0]
itr=int(1) # counter
while itr<(len(CE)-1):
    r1=CE[itr][0]
    psisq1=CE[itr][3]
    HC1=-(k0/epsilon)*(el0**2)*(1/r1)
    Int1=4*math.pi*psisq1*HC1*(r1**2)*delr # first point
    #
    r2=CE[int(itr+1)][0]
    psisq2=CE[int(itr+1)][3]
    HC2=-(k0/epsilon)*(el0**2)*(1/r2)
    Int2=4*math.pi*psisq2*HC2*(r2**2)*delr # second point
    #
    Intavg=(Int1+Int2)/2.0
    Int=Int+Intavg
    #
    itr=int(itr+2)
#
print(Int, psiVH)
print(f'\nGround State Energy = {E0} eV')
print(f'First Order Energy Correction Due to Coulomb Interaction = {Int*
    psiVH*Jouletoev*(10**9)} eV\n')

# performing integration over CE by keeping VH constant at r=0 => first
    excited state
psiVH1=VH1[0][3]
#print(psiVH)
Int=0.0 # total integration value for CE
itr=int(1) # counter
while itr<(len(CE1)-1):
    r1=CE1[itr][0]*(10**(-9))
```

```
200     psisq1=CE1[itr][3]
201     HC1=-(k0/epsilon)*(el0**2)*(1/r1)
202     Int1=4*math.pi*psisq1*HC1*(r1**2)*delr # first point
203     #
204     r2=CE1[int(itr+1)][0]*(10**(-9))
205     psisq2=CE1[int(itr+1)][3]
206     HC2=-(k0/epsilon)*(el0**2)*(1/r2)
207     Int2=4*math.pi*psisq2*HC2*(r2**2)*delr # second point
208     #
209     Intavg=(Int1+Int2)/2.0
210     Int=Int+Intavg
211     #
212     itr=int(itr+2)
213 #
214 print(f'\nFirst Excited State Energy = {E1} eV')
215 print(f'First Order Energy Correction Due to Coulomb Interaction = {Int*
    psiVH1*Jouletoev*(10**9)} eV\n')
216
217 # ------------------------------------------------------------------
    FIRST ORDER CORRECTION TO WAVEFUNCTIONS
    ------------------------------------------------------------------#


218
219 # to find the first order correction to the wavefunction, we need to (
    technically) know all the states, but we can make an estimate to that.
220 # we first need to sort the energy vals array with respect to the exciton
    energy
221 EnergyValsSort=arraysort(EnergyVals,4)
222
223
224 # ------------------------------------------ we need to find
    contributiong factors for each energy level ==> for wavefunction
    correction
225 EcorrWave=[0 for i in range(len(EnergyVals))] # storing correction factor
226 for i in range(1,len(EnergyValsSort)):
227     IntState=0.0 # total integration value
```

```
    lCE=int(EnergyValsSort[i][0])
    lVH=int(EnergyValsSort[i][1])
    # importing CE wavefunction for that state
    nmce=path+"\\Wavefunctions\\"+f'CE_wavefunc_l={lCE}.txt'
    filece=open(nmce,'r')
    # storing data in array
    CEext=[[float(num) for num in line.split(' ')] for line in filece] #0
   : position ; 3 : |\psi|^2
    filece.close()
    # importing VH wavefunction for that state
    nmvh=path+"\\Wavefunctions\\"+f'VH_wavefunc_l={lVH}.txt'
    filevh=open(nmvh,'r')
    # storing data in array
    VHext=[[float(num) for num in line.split(' ')] for line in filevh] #0
   : position ; 3 : |\psi|^2
    filevh.close()
    # storing the r=0 val for psi_VH
    psivh1=VHext[0][1]
    psivhg1=VH[0][1]
    # performing the integration
    itr=int(1) # counter
    while itr<(len(CEext)-1):
        r1=CEext[itr][0]
        psi1=CEext[itr][1]
        psig1=CE[itr][1]
        HC1=-(k0*(10**27)/epsilon)*(el0**2)*(1/r1) # in nm units
        IntS1=4*math.pi*psi1*psig1*HC1*(r1**2)*delr # first point
        #
        r2=CEext[int(itr+1)][0]
        psi2=CEext[int(itr+1)][1]
        psig2=CE[int(itr+1)][1]
        HC2=-(k0*(10**27)/epsilon)*(el0**2)*(1/r2)
        IntS2=4*math.pi*psi2*psig2*HC2*(r2**2)*delr # second point
        #
        IntStateavg=(IntS1+IntS2)/2.0
        IntState=IntState+IntStateavg
```

```
        #
        itr=int(itr+2)
    #
    Ei=EnergyValsSort[i][4]
    IntState=IntState/((E0-Ei)*1.602*(10**(-1))) # converting to nmJoules
    EcorrWave[i]=IntState*psivh1*psivhg1
    #print(IntState)
    #print(f'Correction factor for m={i} (lCE={lCE}, lVH={lVH}) : {IntState*psivh1*psivhg1*(10**9)} eV')
    #print(EcorrWave)
#

# ------------------------------------ generating corrected wavefunctions
nmcorr=path+"\\Wavefunctions\\"+f'FirstCorrectedExciton.txt'
fcorr=open(nmcorr,"w")
fcorr.close()
nmoriginal=path+"\\Wavefunctions\\"+f'UnperturbedExciton.txt'
forig=open(nmoriginal,"w")
forig.close()
# restoring ground state wavefunction
CorrWave=shift_matrix_multiply(CE,VH[0][1],1)
Correction=[0 for i in range(len(CE))]
# running loop to generate corrections
for i in range(1,len(EcorrWave)):
    lCE=int(EnergyValsSort[i][0])
    lVH=int(EnergyValsSort[i][1])
    # importing CE wavefunction for that state
    nmce=path+"\\Wavefunctions\\"+f'CE_wavefunc_l={lCE}.txt'
    filece=open(nmce,'r')
    # storing data in array
    CEext=[[float(num) for num in line.split(' ')] for line in filece] #0 : position ; 3 : |\psi|^2
    filece.close()
    # importing VH wavefunction for that state
    nmvh=path+"\\Wavefunctions\\"+f'VH_wavefunc_l={lVH}.txt'
    filevh=open(nmvh,'r')
```

```
    # storing data in array
    VHext=[[float(num) for num in line.split(' ')] for line in filevh] #0
    : position ; 3 : |\psi|^2
    filevh.close()
#

# corrected wavefunctions
for i in range(len(CorrWave)):
    append_file(nmoriginal,f'{CorrWave[i][0]} {CorrWave[i][1]} {0.0} {
    modulus(CorrWave[i][1])}\n')
    CorrWave[i][1]=CorrWave[i][1]+Correction[i]
    append_file(nmcorr,f'{CorrWave[i][0]} {CorrWave[i][1]} {0.0} {modulus(
    CorrWave[i][1])}\n')
# ------------------------------------------------ #
end=time.time()
print(f'Time taken for code to run = {end-begin}\n')
```

A.5 Interaction of Quantum Dot with a Weak Electric Field : Computational Solution

We use the corrected ground state wavefunction generated by Coulomb interaction to obtain the upper limit to the second order energy correction given in equation (2.38).

```
# energy correction due to stark effect

import math
import cmath
import sys
import time
import os
import numpy as np
import scipy
from scipy.special import spherical_jn
from scipy.special import spherical_yn
from scipy.special import spherical_in
from scipy.special import spherical_kn
```

```
14
15 # all external functions needed for the code
16
17 # READ FILE
18 def read_matrix(x): #more than one column #parameter: x ==> name of file
19     f=open(x,'r') #'r' ==> read only
20     X=[[float(num) for num in line.split(' ')] for line in f]
21     f.close()
22     return(X)
23 #
24
25 # APPEND FILE
26 def append_file(name, str): #arguments: name ==> name of file, str ==>
       string to append
27     f=open(name, 'a') #'a' ==> append file
28     f.write(str)
29     f.close()
30     #
31 #
32
33 #MOD SQUARE OF A COMPLEX NUMBER
34 def modulus(x): #x ==> complex number
35    real=x.real**2
36    imag=x.imag**2
37    modsq=real+imag
38    return(modsq)
39 #
40
41 #MOD SQUARE OF A COMPLEX NUMBER
42 def modulus(x): #x ==> complex number
43    real=x.real**2
44    imag=x.imag**2
45    modsq=real+imag
46    return(modsq)
47 #
48
```

```
# Integration
def simpson(f, lower, upper, h, name):
    N=int(abs(upper-lower)/h)
    I=0
    X=[0 for i in range(N+1)] #for storing N+1 values, including upper and
    lower
    X[0]=lower
    for i in range(1,N+1):
        X[i]=lower+h*i #finding the next point
        mid=(X[i-1]+X[i])/2 #finding mid value
        h1=(X[i]-X[i-1])/2  #diving every slice in two equal slices, and
   finding h relevant to that slice
        #calculating the integration
        I=I+(h1/3)*(f(X[i-1])+4*f(mid)+f(X[i]))
        #
    #storing N vs I value in file
    append_file(name, f'{N} {I}\n')
    return(I)
#

# sorting array with respect to certain column ==> bubble algortihm
def arraysort(X,col):
    n=len(X)
    ncol=len(X[0])
    for i in range(n):
        for j in range(i+1,n):
            if (X[i][col]>X[j][col]):
                for k in range(ncol) :
                    temp=X[i][k]
                    X[i][k]=X[j][k]
                    X[j][k]=temp
                #
            #
        #
    #
    return(X)
```

```
#

# SHIFT VALUES OF MATRIX BY MULTIPLCATION ==> multiply some constant/ variable to terms of a matrix
def shift_matrix_multiply(M,val,col): #arguments: M ==> original matrix which needs shifting, val ==> value to add to the matrix (all terms), col ==> which column of the matrix to multiply (if 1D matrix, then col =0)
    n=len(M) #number of rows
    if col==0:
        CopyM=[0 for i in range(n)]
        for i in range(n):
            CopyM[i]=M[i]*val
    #
    else:
        m=len(M[0])
        CopyM=[[0 for j in range(m)] for i in range(n)]
        for i in range(n):
            for j in range(m):
                if j ==col : CopyM[i][col]=M[i][col]*val
                else : CopyM[i][j]=M[i][j]
            #
        #
    #
    return(CopyM)
#

#------------------------------------------------- Actual Code Starts -------------------------------------------------#

#clock beginning time to measure time taken by code to run
begin=time.time()

#setting path of file
path=os.path.dirname(os.path.realpath(__file__))
sys.path.append(path)
```

```
# ------------------------------------------------------------------- #

# defining constants in natural units
confac=5.066*(10**(-3))
epsilon0=1
epsilon=5.367
k=(1/(4*math.pi*epsilon0*epsilon))
el0=1.602*(10**(-19))
elconfac=1.0/(5.289*(10**(-19)))
el=el0*elconfac
evJoule=1.602*(10**(-19))

EF2=1014223.237


# reading energy from files
nmexcitonEN=path+"\\Wavefunctions\\"+'Energy of Exciton.txt'
fecxitonEN=open(nmexcitonEN,'r')
header_line=next(fecxitonEN) # ignoring the first line of the file
EnergyVals=[[float(num) for num in line.split(' ')] for line in fecxitonEN
    ] #read entire file
fecxitonEN.close()

# as we only careabout the first two energies for the first order energy
    correction, we will store them
E0=EnergyVals[0][4]
E1=EnergyVals[1][4]
Ecorr0=-0.0465 #first order correction to ground state

# first importing the ground state
# ground state
nmExc=path+"\\Wavefunctions\\"+f'FirstCorrectedExciton.txt'
fileExc=open(nmExc,'r')
# storing data in array
Exc=[[float(num) for num in line.split(' ')] for line in fileExc] #0 :
    position ; 3 : |\psi|^2 # l=0 for CE
```

```
fileExc.close()

# performing the two integrations
IntH1=0.0
IntH2=0.0
itr=int(1) # counter
delr=Exc[1][0]-Exc[0][0]
while itr<(len(Exc)-1):
    r1=Exc[itr][0]
    psisq1=Exc[itr][3]
    HC2r1=r1**2
    # first point
    Int2r1=4*math.pi*psisq1*HC2r1*(r1**2)*(1/3)*delr
    #
    r2=Exc[int(itr+1)][0]
    psisq2=Exc[int(itr+1)][3]
    HC2r2=r2**2
    # second point
    Int2r2=4*math.pi*psisq2*HC2r2*(r2**2)*(1/3)*delr
    #
    IntavgH2=(Int2r1+Int2r2)/2.0
    IntH2=IntH2+IntavgH2
    #
    itr=int(itr+2)
#

EcorrStark=(-(el0**2))*2*(IntH2*(10**(-18)))/((E0+Ecorr0-E1)*evJoule)
print(f'Correction={EcorrStark}')

# ------------------------------------------------ #
end=time.time()
print(f'Time taken for code to run = {end-begin}\n')
```

A.6 Conversion to Natural Units

Natural units is a system of units in which we set the speed of light and Planck's constant to be 1, and re-frame all other dimension-full quantities appropriately.

$$c = 3 \times 10^{-8}\ \mathrm{m\ s^{-1}}\,, \quad \hbar = 1.0546 \times 10^{-34} \mathrm{kg\,m^2\,s^{-1}}\,. \tag{A.32}$$

Any physical quantity in Si units has dimensions given as

$$[x] = [\mathrm{kg}^{\alpha}\mathrm{m}^{\beta}\mathrm{s}^{\gamma}]\,. \tag{A.33}$$

We can express it in terms of E, $\hbar$ and c as well, as

$$[x] = [E^{p}\hbar^{q}c^{r}]\,. \tag{A.34}$$

Comparing the two unit systems in equations (A.34) and (A.33), we get

$$\begin{aligned}
& [\mathrm{kg}^{\alpha}\mathrm{m}^{\beta}\mathrm{s}^{\gamma}] = [E^{p}\hbar^{q}c^{r}] \\
\implies & [\mathrm{kg}^{\alpha}\mathrm{m}^{\beta}\mathrm{s}^{\gamma}] = [\mathrm{kg}^{p}\mathrm{m}^{2p}\mathrm{s}^{-2p} \cdot \mathrm{kg}^{q}\mathrm{m}^{2q}\mathrm{s}^{-q} \cdot \mathrm{m}^{r}\mathrm{s}^{-r}]\,, \\
\implies & [\mathrm{kg}^{\alpha}\mathrm{m}^{\beta}\mathrm{s}^{\gamma}] = [\mathrm{kg}^{p+q}\mathrm{m}^{2p+2q+r}\mathrm{s}^{-2p-2q-r}]\,.
\end{aligned}$$

We then get a system of three equations as

$$\begin{aligned}
\alpha &= p + q\,, \\
\beta &= 2p + 2q + r\,, \\
\gamma &= -2p - q - r\,.
\end{aligned}$$

Solving the above equations, we can obtain p, q and r in terms of α, β and γ as

$$p = \alpha - \beta - \gamma\,, \tag{A.35}$$

$$q = \beta + \gamma\,, \tag{A.36}$$

$$r = \beta - 2\alpha\,. \tag{A.37}$$

Thus, we can now express the physical quantity in terms of E, $\hbar$ and c as

$$[x] = [\mathrm{kg}^{\alpha}\mathrm{m}^{\beta}\mathrm{s}^{\gamma} = [E^{\alpha-\beta-\gamma}\hbar^{\beta+\gamma}c^{\beta-2\alpha}]\,. \tag{A.38}$$

By setting $\hbar = c = 1$, we can express all physical units in some powers of energy. The above equation gives us the required conversion factors for mass and length as

$$(1\ \mathrm{eV})_{\mathrm{mass}} = 1.780 \times 10^{-36}\mathrm{kg}\,, \tag{A.39}$$

$$(1\ \mathrm{eV})_{\mathrm{length}} = 1.974 \times 10^{-7}\mathrm{m}\,. \tag{A.40}$$

If we now also set $\epsilon_0 = 1$, we get charge to be a dimensionless quantity, with

$$(1\ \mathrm{charge})_{\mathrm{Natural}} = 5.289 \times 10^{-19}\ \mathrm{Coulomb}\,. \tag{A.41}$$

We must also remember that wavelengths have dimensions of $(\mathrm{length})^{-3/2}$, and we have defined lengths in terms of nanometers. We must perform appropriate conversions where needed. Some fundamental conversions can be obtained as

$$(1\ \mathrm{nm})_{Natural} = 5.066 \times 10^{-3}\ \mathrm{eV}^{-1}\,, \tag{A.42}$$

$$([\psi] = [\mathrm{nm}^{-3/2}])_{Natural} = (1.974)^{3/2} \times 10^{3}\ \mathrm{eV}^{3/2}\,, \tag{A.43}$$

$$(m_0)_{Natural} = 0.511\ \mathrm{MeV}\,, \tag{A.44}$$

$$(e)_{Natural} = 0.303\,. \tag{A.45}$$

B. SUPPLEMENTARY MATERIAL FOR CHAPTER 3

B.1 Surface Charge Density of a Conducting Disk

In this section, I provide the detailed derivations for the surface charge density of a conducting disk.

The surface charge density of any surface can be obtained by taking a derivative of the potential of the surface. To obtain the potential of an ellipsoidal surface, we can solve the problem of equipotential surfaces of a line charge.

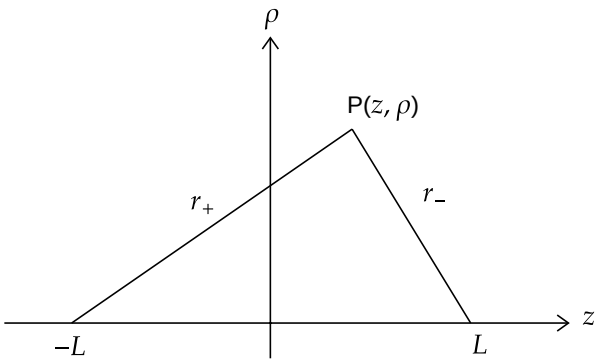

Figure B.1. Line charge

For a given line charge system of length $2L$ and uniform line charge density λ_{charge}, the potential at any point $P(z,\rho)$ can be written as

$$\varphi(z,\rho) = \frac{\lambda_{charge}}{4\pi\epsilon_0} \ln\left[\frac{z+L+\sqrt{(L+z)^2+\rho^2}}{z-L+\sqrt{(L-z)^2+\rho^2}}\right] . \tag{B.1}$$

We can define a new coordinate system to simplify the above equation as

$$r_\pm = \sqrt{\rho^2+(L\pm z)^2}\,, \tag{B.2}$$

$$u = \frac{1}{2}(r_+ + r_-)\,, \tag{B.3}$$

$$t = \frac{1}{2}(r_+ - r_-)\,. \tag{B.4}$$

From this, we can obtain ut as well as

$$\begin{aligned} ut &= \frac{1}{4}(r_+^2 - r_-^2)\,, \\ \implies ut &= \frac{1}{4}\left[\cancel{\rho^2} + (L+z)^2 - \cancel{\rho^2} - (L-z)^2\right]\,, \\ \therefore\ ut &= zL\,. \end{aligned} \tag{B.5}$$

We can also write r_+ and r_- in terms of u and t as

$$r_+ = u + t\,, \quad r_- = u - t\,. \tag{B.6}$$

Rewriting the potential (B.1) in terms of the new coordinate system,

$$\begin{aligned} \varphi(u,L) &= \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{z+L+(u+t)}{z-L+(u-t)}\right]\,, \\ \implies \varphi(u,L) &= \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{ut/L+L+(u+t)}{ut/L-L+(u-t)}\right]\,, \\ \implies \varphi(u,L) &= \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{u+L+t+ut/L}{u-L-t+ut/L}\right]\,, \\ \implies \varphi(u,L) &= \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{(u+L)+(t/L)(u+L)}{(u-L)+(t/L)(u-L)}\right]\,, \\ \implies \varphi(u,L) &= \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{(u+L)(1+t/L)}{(u-L)(1+t/L)}\right]\,. \end{aligned}$$

Thus we finally obtain

$$\varphi(u,L) = \frac{\lambda_{charge}}{4\pi\epsilon_0}\ln\left[\frac{u+L}{u-L}\right]\,. \tag{B.7}$$

The equipotential surfaces are ellipsoids with semi-major axis as $u = b$. As we expect the electric field to be normal to the surface of the ellipsoid, i.e.,

$$\vec{\nabla}\varphi \parallel \hat{n} \implies \vec{\nabla} \parallel \varphi(c_1\hat{\rho} + c_2\hat{z})\,.$$

c_1 and c_2 can depend on where we are on the ellipsoid. The surface charge density can then be written as

$$\begin{aligned}
\sigma = \epsilon_0 |\nabla\varphi|_S &= \epsilon_0 \left| \frac{\partial\varphi}{\partial\rho}\hat{\rho} + \frac{\partial\varphi}{\partial z} \right|_S , \\
\implies \sigma &= \epsilon_0 \sqrt{\left(\frac{\partial\varphi}{\partial\rho}\right)^2 + \left(\frac{\partial\varphi}{\partial z}\right)^2} \Bigg|_S , \\
\therefore \sigma &= \epsilon_0 \left| \frac{\partial\varphi}{\partial u} \right|_{u=b} \sqrt{\left(\frac{\partial u}{\partial\rho}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2} \Bigg|_S .
\end{aligned} \tag{B.8}$$

We now need to calculate the terms individually. Starting with $\left| \frac{\partial\varphi}{\partial u} \right|_{u=b}$,

$$\begin{aligned}
\frac{\partial\varphi}{\partial u} &= \frac{\lambda_{charge}}{4\pi\epsilon_0} \frac{\partial}{\partial u} \left[\ln(u+L) - \ln(u-L) \right] , \\
\implies \frac{\partial\varphi}{\partial u} &= \frac{\lambda_{charge}}{4\pi\epsilon_0} \left[\frac{1}{u+L} - \frac{1}{u-L} \right] , \\
\implies \frac{\partial\varphi}{\partial u} &= \frac{\lambda_{charge}}{4\pi\epsilon_0} \left[\frac{\cancel{u} - L - \cancel{u} - L}{(u+L)(u-L)} \right] , \\
\implies \frac{\partial\varphi}{\partial u} &= \frac{\lambda_{charge}}{4\pi\epsilon_0} \frac{(-2L)}{(u^2 - L^2)} .
\end{aligned}$$

Now, we have $\lambda_{charge} = \dfrac{Q}{2L}$ and at the surface of the ellipsoid $u = b$. As $L^2 = b^2 - a^2$ for an ellipsoid, we also have $u^2 - L^2 = a^2$. Thus, we get the final expression as

$$\therefore \left| \frac{\partial\varphi}{\partial u} \right|_{u=b} = \frac{Q}{4\pi\epsilon_0 a^2} . \tag{B.9}$$

Calculating the next term

$$\begin{aligned}
\left| \frac{\partial u}{\partial\rho} \right|_S &= \frac{1}{2} \left[\frac{\partial r_+}{\partial\rho} + \frac{\partial r_-}{\partial\rho} \right]_S , \\
\implies \left| \frac{\partial u}{\partial\rho} \right|_S &= \frac{1}{2} \left[\frac{2\rho}{2r_+} + \frac{2\rho}{2r_-} \right]_S , \\
\implies \left| \frac{\partial u}{\partial\rho} \right|_S &= \frac{\rho}{2} \left[\frac{r_- + r_+}{r_+ r_-} \right]_S .
\end{aligned}$$

Recalling that at the surface, we have $u = \frac{1}{2}(r_+ + r_-) = b$, we get

$$\implies \left|\frac{\partial u}{\partial \rho}\right|_S = \frac{b\rho}{r_+ r_-}\bigg|_S .$$

Now evaluating $r_+ r_-$, we can write

$$\begin{aligned} r_+ r_- &= u^2 - t^2 \,, \\ \implies r_+ r_-\big|_S &= \left(u^2 - \frac{z^2 L^2}{u^2}\right)\bigg|_S \,, \\ \implies r_+ r_-\big|_S &= b^2 - \frac{z^2(b^2 - a^2)}{b^2} \,, \\ \implies r_+ r_-\big|_S &= \frac{1}{b^2}\left[b^4 - z^2(b^2 - a^2)\right] . \end{aligned}$$

The equation of an ellipsoid is given as

$$\begin{aligned} \frac{x^2}{a^2} + \frac{y^2}{a^2} + \frac{z^2}{b^2} &= 1 \,, \\ \therefore \frac{\rho^2}{a^2} + \frac{z^2}{b^2} &= 1 \,. \end{aligned}$$

Going back to the partial derivative of u with respect to ρ that we were evaluating, we obtain

$$\begin{aligned} \implies \left|\frac{\partial u}{\partial \rho}\right|_S &= b\rho\,\frac{b^2}{b^4 - z^2(b^2 - a^2)} \,, \\ \implies \left|\frac{\partial u}{\partial \rho}\right|_S &= \frac{\rho b^3}{b^4 - z^2 b^2 + z^2 a^2} \,, \\ \implies \left|\frac{\partial u}{\partial \rho}\right|_S &= \frac{\rho/b}{1 - \dfrac{z^2}{a^2} + \dfrac{z^2 a^2}{b^4}} \,, \\ \implies \left|\frac{\partial u}{\partial \rho}\right|_S &= \frac{\rho/b}{\dfrac{\rho^2}{a^2} + \dfrac{z^2 a^2}{b^4}} \,. \end{aligned}$$

Thus we finally obtain

$$\therefore \left|\frac{\partial u}{\partial \rho}\right|_S = \frac{\rho}{a^2 b}\left[\frac{\rho^2}{a^4} + \frac{z^2}{b^4}\right]^{-1} . \tag{B.10}$$

Now calculating the derivative of u with respect to z, we get

$$\begin{aligned}
&\left|\frac{\partial u}{\partial z}\right|_S = \frac{1}{2}\left(\frac{\partial r_+}{\partial z} + \frac{\partial r_-}{\partial z}\right)\Bigg|_S, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{1}{2}\left[\frac{1}{2r_+}.2(L+z) - \frac{1}{2r_-}.2(L-z)\right]\Bigg|_S, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{1}{2}\left(\frac{L+z}{r_+} - \frac{L-z}{r_-}\right)\Bigg|_S, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{1}{2}\left[\frac{r_-(L+z) - r_+(L-z)}{r_+ r_-}\right]\Bigg|_S, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{-Lt + zu}{r_+ r_-}\Bigg|_S = \frac{-L^2 z/u + zu}{r_+ r_-}\Bigg|_S, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{-(b^2-a^2)z/b + zb}{\frac{1}{b^2}(b^2 - z^2b^2 + z^2a^2)}, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{[(a^2-b^2)z + zb^2]b^2}{b(b^4 - z^2b^2 + z^2a^2)}, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{za^2b}{b^4 - z^2b^2 + z^2a^2} = \frac{za^2/b^3}{1 - \frac{z^2}{b^2} + \frac{z^2a^2}{b^4}}, \\
\implies &\left|\frac{\partial u}{\partial z}\right|_S = \frac{za^2/b^3}{\frac{\rho^2}{a^2} + \frac{z^2a^2}{b^4}} = \frac{z\cancel{a^2}/b^3}{\cancel{a^2}\left(\frac{\rho^2}{a^4} + \frac{z^2}{b^4}\right)}.
\end{aligned}$$

Thus we obtain the final expression as

$$\therefore \left|\frac{\partial u}{\partial z}\right|_S = \frac{z}{b^3}\left(\frac{\rho^2}{a^4} + \frac{z^2}{b^4}\right)^{-1}. \tag{B.11}$$

Putting all the equations together, we can find σ

$$\begin{aligned}
&\sigma = \epsilon_0 \left|\frac{\partial \varphi}{\partial u}\right|_{u=b} \sqrt{\left(\frac{\partial u}{\partial \rho}\right)^2 + \left(\frac{\partial u}{\partial z}\right)^2}\Bigg|_S, \\
\implies &\sigma = \epsilon_0 \frac{Q}{4\pi\epsilon_0 a^2}\sqrt{\frac{\rho^2}{a^4b^2} + \frac{z^2}{b^4b^2}}\left(\frac{\rho^2}{a^4} + \frac{z^2}{b^4}\right)^{-1},
\end{aligned}$$

$$\implies \sigma = \cancel{6}\frac{Q}{4\pi\cancel{6}a^2 b}\sqrt{\frac{\rho^2}{a^4}+\frac{z^2}{b^4}}\left(\frac{\rho^2}{a^4}+\frac{z^2}{b^4}\right)^{-1}.$$

Thus, we obtain the final expression for σ as

$$\boxed{\sigma = \frac{Q}{4\pi a^2 b}\left(\frac{\rho^2}{a^4}+\frac{z^2}{b^4}\right)^{-1/2}.} \tag{B.12}$$

B.2 Potential due to the Conducting Disk in Space

The general expression for potential at some point P along the z-axis of a conducting disk can be written as (3.12)

$$V(\rho,z) = \frac{1}{4\pi\epsilon_0}\int \frac{\sigma(\rho')\,\rho'\,d\rho'\,d\phi'}{\sqrt{z^2+\rho'^2}}. \tag{B.13}$$

Due to azimuthal symmetry $d\phi'$ integrates out to 2π. The integral then simplifies as

$$\begin{aligned}
\implies V(\rho,z) &= \frac{1}{4\pi\epsilon_0}2\pi\int \frac{\sigma(\rho')\,\rho'\,d\rho'}{\sqrt{z^2+\rho'^2}},\\
\implies V(\rho,z) &= \frac{1}{2\epsilon_0}\frac{Q}{4\pi a}\int_0^a \frac{\rho'\,d\rho'}{\sqrt{a^2-\rho'^2}\sqrt{z^2+\rho'^2}},\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\int_0^a \frac{\rho'\,d\rho'}{\sqrt{a^2-\rho'^2}\sqrt{z^2+\rho'^2}}.
\end{aligned}$$

We can define a new variable "s" such that

$$s = \sqrt{z^2+\rho'^2} \implies ds = \frac{\rho'\,d\rho'}{\sqrt{z^2+\rho'^2}}.$$

We also get new limits of integration due to this change of variables, as

$$\begin{aligned}
\rho' = 0 &\implies s = z,\\
\rho' = a &\implies s = \sqrt{z^2+a^2}.
\end{aligned}$$

Rewriting the integral in the new coordinate system

$$\implies V(\rho,z) = \frac{Q}{8\pi\epsilon_0 a}\int_z^{\sqrt{z^2+a^2}} \frac{ds}{\sqrt{a^2+z^2-s^2}}\,.$$

We will now define another new variable "m" such that

$$m = \frac{s}{\sqrt{a^2+z^2}} \implies dm = \frac{ds}{\sqrt{a^2+z^2}}\,.$$

Rewriting the integral again in terms of m,

$$\begin{aligned}
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\int_{limits} \frac{\sqrt{a^2+z^2}\,dm}{\sqrt{a^2+z^2-m^2(a^2+z^2)}}\,,\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\int_{limits} \frac{\sqrt{a^2+z^2}\,dm}{\sqrt{(1-m^2)\,(a^2+z^2)}}\,,\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\int_{limits} \frac{dm}{\sqrt{1-m^2}}\,.\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\sin^{-1}(m)\Big|_{limits}
\end{aligned}$$

Changing back to the original variables

$$\begin{aligned}
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\sin^{-1}\left(\sqrt{\frac{z^2+\rho'^2}{a^2+z^2}}\right)\Bigg|_{\rho'=0}^{\rho'=a},\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\left[\sin^{-1}\left(\sqrt{\frac{\cancel{a^2+z^2}}{\cancel{a^2+z^2}}}\right) - \sin^{-1}\left(\sqrt{\frac{z^2}{a^2+z^2}}\right)\right],\\
\implies V(\rho,z) &= \frac{Q}{8\pi\epsilon_0 a}\left[\frac{\pi}{2} - \sin^{-1}\left(\frac{z}{\sqrt{a^2+z^2}}\right)\right].
\end{aligned}$$

Using trigonometric identities, we can finally obtain

$$\boxed{V(z) = \frac{Q}{8\pi\epsilon_0 a}\left[\frac{\pi}{2} - \tan^{-1}\left(\frac{z}{a}\right)\right] = \frac{Q}{8\pi\epsilon_0 a}\tan^{-1}\left(\frac{a}{z}\right).} \tag{B.14}$$

B.3 Electric Field due to Conducting Disk in Space

To simplify the problem, the electric field can be derived separately for regions inside and outside the sphere of radius a.

Case 1 : $\mathbf{r \leq a} \implies \mathbf{(\rho^2 + z^2)^{1/2} \leq a}$: $\hat{\rho}$ Part

The potential inside the sphere is given in equation (3.23). We can take its derivative with respect to ρ and z to find E_ρ^{in} and E_z^{in}. We will first solve it for E_ρ^{in}, as

$$\begin{aligned} E_\rho^{in} &= -\frac{\partial V^{in}}{\partial \rho}(\rho, z)\,, \\ \implies E_\rho^{in} &= -\sum_{l=0}^{\infty}\left[B_l\,\frac{l}{2}\,(\rho^2+z^2)^{\frac{l}{2}-1}2\rho\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + B_l(\rho^2+z^2)^{\frac{l}{2}}\frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right]. \end{aligned}$$

We now need to find the derivative of the Legendre polynomial function. Let us define a new variable "x", as

$$x = \frac{z}{\sqrt{z^2+\rho^2}} \implies x^2 - 1 = \frac{-\rho^2}{\rho^2+z^2}\,. \tag{B.15}$$

The derivative of Legendre polynomial can then be computed as follows

$$\begin{aligned} \frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) &= \frac{\partial}{\partial\rho}P_l(x) = \frac{\partial P_l}{\partial x}\,\frac{\partial x}{\partial\rho}\,, \\ \implies \frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) &= z\left(\frac{-1}{2}\right)(\rho^2+z^2)^{-3/2}\,2\rho\,\frac{dP_l(x)}{dx}\,, \\ \therefore \frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) &= \frac{-z\rho}{(\rho^2+z^2)^{3/2}}\,\frac{dP_l(x)}{dx}\,. \end{aligned}$$

The Legendre Polynomials follow a recurrence relation given as

$$\frac{dP_l(x)}{dx} = \frac{l}{x^2-1}\left[x\,P_l(x) - P_{l-1}(x)\right]. \tag{B.16}$$

Putting all of this together, we obtain the complete derivative of $P_l(x)$ w.r.t ρ, as

$$\frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \frac{l\,z}{\rho(\rho^2+z^2)^{1/2}}\left[\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right]. \tag{B.17}$$

We now need to put all of this back in the expression for E_ρ^{in}, which gives us,

$$\begin{aligned}
\implies E_\rho^{in} &= -\sum_{l=0}^{\infty} B_l \left[(\rho^2+z^2)^{\frac{l}{2}-1} \, l\rho \, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + \frac{lz}{\rho}\frac{1}{(\rho^2+z^2)^{1/2}}(\rho^2+z^2)^{l/2} \right. \\
&\qquad \left. \times \left\{ \frac{z}{\sqrt{z^2+\rho^2}} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right\} \right] , \\
\implies E_\rho^{in} &= -\sum_{l=0}^{\infty} B_l \, l \, (\rho^2+z^2)^{\frac{l}{2}-1} \left[\rho P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + \frac{z}{\rho}\sqrt{z^2+\rho^2} \right. \\
&\qquad \left. \times \left\{ \frac{z}{\sqrt{z^2+\rho^2}} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right\} \right] , \\
\implies E_\rho^{in} &= -\sum_{l=0}^{\infty} B_l \, l \, (\rho^2+z^2)^{\frac{l}{2}-1} \left[\left\{ \rho + \frac{z^2}{\rho} \right\} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - \frac{z}{\rho}\sqrt{z^2+\rho^2} \, P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right] .
\end{aligned}$$

Thus we finally obtain the final expression as

$$\boxed{\begin{aligned} E_\rho^{in} = \sum_{l=0}^{\infty} B_l \, l \, (\rho^2+z^2)^{\frac{l}{2}-1} \left\{ \frac{z}{\rho}\sqrt{z^2+\rho^2} \, P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right. \\ \left. - \frac{(\rho^2+z^2)}{\rho} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right\} . \end{aligned}} \tag{B.18}$$

Case 2 : $\mathbf{r \le a} \implies \mathbf{(\rho^2+z^2)^{1/2} \le a}$: $\hat{\mathbf{z}}$ Part

We can now focus on the z-component of the electric field, given as

$$\begin{aligned}
E_z^{in} &= -\frac{\partial V^{in}}{\partial z} , \\
E_z^{in} &= -\sum_{l=0}^{\infty} \left[B_l \frac{l}{2} (\rho^2+z^2)^{\frac{l}{2}-1} 2z \, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + B_l (\rho^2+z^2)^{\frac{l}{2}} \frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \right] .
\end{aligned}$$

We now need to find the derivative of the Legendre polynomial with respect to z. We can use the previously defined variable x in (B.15), as

$$\begin{aligned}
&\frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \frac{\partial}{\partial z} P_l(x) = \frac{\partial P_l}{\partial x}\frac{\partial x}{\partial z} , \\
\implies &\frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \left\{ \frac{1}{\sqrt{z^2+\rho^2}} - \frac{1}{2}\frac{2z^2}{\sqrt{(\rho^2+z^2)^{3/2}}} \right\} \frac{dP_l}{dx} .
\end{aligned}$$

Using the recurrence relation for Legendre polynomials in (B.16), we get

$$\implies \frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \frac{1}{\sqrt{z^2+\rho^2}}\left\{1-\frac{z^2}{\sqrt{z^2+\rho^2}}\right\}\frac{l\,(\rho^2+z^2)}{(-\rho^2)} \times\left[\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)-P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right],$$

$$\implies \frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \frac{1}{\sqrt{z^2+\rho^2}}\frac{\cancel{\rho^2}}{\cancel{(\rho^2+z^2)}}(-l)\frac{\cancel{(\rho^2+z^2)}}{\rho^2} \times\left[\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)-P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right].$$

So we finally get the derivative of P_l w.r.t z as

$$\therefore \frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) = \frac{-l}{\sqrt{z^2+\rho^2}}\left[\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)-P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right]. \quad \text{(B.19)}$$

Putting this back in the expression for E_z^{in}, we obtain

$$\implies E_z^{in} = -\sum_{l=0}^{\infty}\left[B_l\,l\,(\rho^2+z^2)^{\frac{l}{2}-1}z\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)-l\,B_l\,(\rho^2+z^2)^{\frac{l}{2}}\frac{1}{\sqrt{z^2+\rho^2}} \times\left\{\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)-P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}\right],$$

$$\implies E_z^{in} = -\sum_{l=0}^{\infty}B_l\,l\,(\rho^2+z^2)^{\frac{l}{2}-1}\left[\cancel{z\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)}-\sqrt{z^2+\rho^2} \times\left\{\cancel{\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)}-P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}\right].$$

Thus, we obtain the final expression for E_z^{in} as

$$\boxed{E_z^{in}=\sum_{l=0}^{\infty}\left\{-B_l\,l\,(\rho^2+z^2)^{\frac{l}{2}-1}\sqrt{z^2+\rho^2}\,P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}.} \quad \text{(B.20)}$$

Case 3 : $\mathbf{r \geq a \implies (\rho^2 + z^2)^{1/2} \geq a}$: $\hat{\rho}$ Part

The potential outside the sphere is given in the equation (3.26). We can take its derivative with respect to ρ to find the ρ-component of the electric field, as

$$E_\rho^{out} = -\frac{\partial V^{out}}{\partial \rho}(\rho, z)\,,$$
$$\implies E_\rho^{out} = \sum_{l=0}^{\infty}\left[(-1)\frac{l+1}{2}A_l\,(\rho^2+z^2)^{-\frac{l+1}{2}}2\rho\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + A_l\,(\rho^2+z^2)^{-\frac{l+1}{2}}\frac{\partial}{\partial\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right].$$

We have already found the derivative of P_l with respect to ρ in (B.17). Putting that in the expression for E_ρ^{out}, we get

$$\implies E_\rho^{out} = -\sum_{l=0}^{\infty}\left[(-1)(l+1)A_l\,(\rho^2+z^2)^{-\frac{l+3}{2}}\rho\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + A_l\,(\rho^2+z^2)^{-\frac{l+1}{2}}l\,\frac{z}{\rho(\rho^2+z^2)^{1/2}}\right.$$
$$\left.\times\left\{\frac{z}{\sqrt{z^2+\rho^2}}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}\right],$$
$$\implies E_\rho^{out} = -\sum_{l=0}^{\infty}A_l(\rho^2+z^2)^{-\frac{l+3}{2}}\left[(-1)(l+1)\rho P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + l\,\frac{z^2}{\rho}P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right.$$
$$\left.-l\,\frac{z\sqrt{z^2+\rho^2}}{\rho}P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right].$$

Thus, we can finally write

$$\boxed{E_\rho^{out} = \sum_{l=0}^{\infty}A_l\,(\rho^2+z^2)^{-\frac{l+3}{2}}\left[\frac{l(\rho^2-z^2)+\rho^2}{\rho}\,P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right.}$$
$$\boxed{\left.+\,l\,\frac{z}{\rho}\sqrt{z^2+\rho^2}\,P_{l-1}\left(\frac{z}{z^2+\rho^2}\right)\right].} \tag{B.21}$$

Case 4 : $\mathbf{r \geq a \implies (\rho^2 + z^2)^{1/2} \geq a}$: $\hat{z}$ Part

We can take the derivative of V^{out} with respect to z to find the z-component of the electric field, as

$$E_z^{out} = -\frac{\partial V^{out}}{\partial z}(\rho, z)\,,$$

$$\implies E_\rho^{out} = \sum_{l=0}^{\infty} \left[(-1)\frac{l+1}{2} A_l \left(\rho^2+z^2\right)^{-\frac{l+1}{2}} 2z\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + A_l \left(\rho^2+z^2\right)^{-\frac{l+1}{2}} \frac{\partial}{\partial z} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right].$$

We have already obtained the partial derivative of P_l with respect to z in (B.19). Using that in the expression for E_z^{out}, we get

$$\begin{aligned}
\implies E_\rho^{out} = \sum_{l=0}^{\infty} \Bigg[& (-1)\frac{l+1}{2} A_l \left(\rho^2+z^2\right)^{-\frac{l+1}{2}} 2z\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \\
& + A_l \left(\rho^2+z^2\right)^{-\frac{l+1}{2}} \frac{-l}{\sqrt{z^2+\rho^2}} \left\{ \frac{z}{\sqrt{z^2+\rho^2}} P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) - P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\right\}\Bigg], \\
\implies E_\rho^{out} = \sum_{l=0}^{\infty} A_l (\rho^2+z^2)^{-\frac{l+3}{2}} \Bigg[& (l+1) z\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) + z\, l\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \\
& - l\sqrt{z^2+\rho^2} P_{l-1}\left(\frac{z}{\sqrt{z^2+\rho^2}}\right)\Bigg].
\end{aligned}$$

Thus we finally obtain the expression as

$$\boxed{\begin{aligned} E_z^{out} = \sum_{l=0}^{\infty} A_l \left(\rho^2+z^2\right)^{-\frac{l+3}{2}} \Bigg[& z(2l+1)\, P_l\left(\frac{z}{\sqrt{z^2+\rho^2}}\right) \\ & - l\sqrt{z^2+\rho^2}\, P_{l-1}\left(\frac{z}{z^2+\rho^2}\right)\Bigg]. \end{aligned}} \tag{B.22}$$

C. SUPPLEMENTARY MATERIAL FOR CHAPTER 4

C.1 Simulating Patch Potentials : Computational Code

The code to simulate patch potentials on the attractor mass surface is given here.

```
#python code to generate disks

#first we will import some basic libraries
import math
import os
import time
import sys
import numpy as np

path=os.path.dirname(os.path.realpath(__file__))
sys.path.append(path)

epsl0=8.85*(10**(-18)) #in Farad micrometer-inverse units
#
N=5 #defined length ==> number of disks
#
X=np.random.uniform(0,100,N) #chosen from a uniform distribution between 0
    and 100 (X-coordinate)
Y=np.random.uniform(0,100,N) #chosen from a uniform distribution between 0
    and 100 (Y-coordinate)
R=abs(np.random.normal(3,1,N)) #chosen from a Gaussian distribution with
   mean=3,std=1 (radius)
V0=np.random.normal(0,10**(-3),N) #choose from gaussian distribution
   centred at 0
#
Q=[16*epsl0*R[i]*V0[i] for i in range(N)] #defining Q's for the disks
#

#Creating File to store information about disks
nmdisk=path+'\\Results\\'+f'DiskGrid(N={N}).txt' #information of the disks
fd1=open(nmdisk,"w")
```

```
fd1.close()
fd2=open(nmdisk,"a")
#

#storing information about the disk (x,y,radi,Q,V0)
for i in range(N): fd2.write(f'{X[i]} {Y[i]} {0.0} {R[i]} {V0[i]} {Q[i]}\n')
```

C.2 Electrostatic Interaction with Moving Patches : Computational Code

The code to generate electrostatic interaction energy due to interaction with trajectories is given below.

```
#python code to obtain electrostatic energies, generating polynomials through recurrence relation and changes Al's and Bl's to prevent blowups.
#Also, this code attempts to parallelize the functioning

#first we will import some basic libraries
import math
import os
import time
import sys
import numpy as np
import multiprocessing

path=os.path.dirname(os.path.realpath(__file__))
sys.path.append(path)

#measuring beginning time to find time taken by code to run
begin=time.time()

#defining additional functions which might be needed
#
#Legendre Polynomial of given order
def P(n,x,Plnm1,Plnm2):
```

```
    if n>=2 : Pln=(1/n)*((2*n-1)*x*Plnm1-(n-1)*Plnm2)
    if n==1 : Pln=Plnm1
    if n==0 : Pln=Plnm2
    return Pln
#

#declare fixed constants, epsilon0 and position of the dot on z-axis
epsl0=8.85*(10**(-18))
zdot=0.2

#defining polarizability of complete CdSe-CdS dot ==> all in SI units
epsl0SI=epsl0*(10**6) #epsilon naught in SI units
b=2.9*(10**(-9)) #radius of quantum dot in meters
a=1.5*(10**(-9)) # radius of core of quantum dot in meters
nCDSEref=2.3847 #real part of refractive index of CdSe for 1 micron
    wavelength
kCDSEref=0.065147 #imaginary part of refractive index of CdSe for 1 micron
     wavelength
nCDSref=2.307 #real part of refractive index of CdS for 1 micron
    wavelength
kCDSref=0.0 #imaginary part of refractive index of CdS for 1 micron
    wavelength
epslCdSeR=((nCDSEref**2)-(kCDSEref**2))*epsl0SI #real part of dielectric
    constant of CdSe dot
epslCdSR=((nCDSref**2)-(kCDSref**2))*epsl0SI
numeff=(b**3)*(epslCdSeR+2*epslCdSR)+ 2*(a**3)*(epslCdSeR-epslCdSR)
denomeff=(b**3)*(epslCdSeR+2*epslCdSR)-(a**3)*(epslCdSeR-epslCdSR)
epslEff=epslCdSR*(numeff/denomeff)
alpha=(4*math.pi*epsl0SI*(b**3))*((epslEff-epsl0SI)/(epslEff+2*epsl0SI)) #
    polarizability for CdSe-CdS dot

#b=10*(10**(-9)) #radius of quantum dot in meters
#nref=2.3847 #real part of refractive index of CdSe for 1 micron
    wavelength
#kref=0.065147 #imaginary part of refractive index of CdSe for 1 micron
    wavelength
```

```
#epslCdSeR=((nref**2)-(kref**2))*epsl0SI #real part of dielectric constant of CdSe dot
#alpha=(4*math.pi*epsl0SI*(b**3))*((epslCdSeR-epsl0SI)/(epslCdSeR+2*epsl0SI)) #polarizability for CdSe dot (not shell structure yet)

#defining A's, B's, Erho's and Ez's as functions
def A(Q,l): return (Q/(8*math.pi*epsl0))*(((-1)**(l/2))/(l+1)) #only for even numbers
def B(Q,l): return (Q/(8*math.pi*epsl0))*(((-1)**((l+1)/2))/(l)) #only for odd numbers
#
def Erhoin(Q,a,l,rho,z,Pln,Plnm1):
    #cscth=abs(z/(math.sqrt(z**2 + rho**2))) #cos theta term ==> coefficient of the Legendre polynomials
    return B(Q,l)*l*(((rho**2 + z**2)/(a**2))**(l/2))*(1/((rho**2 +z**2)*a))*((z/rho)*math.sqrt(z**2 + rho**2)*Plnm1-((rho**2 + z**2)/rho)*Pln)
#
def Ezin(Q,a,l,rho,z,Pln,Plnm1):
    #cscth=abs(z/(math.sqrt(z**2 + rho**2))) #cos theta term ==> coefficient of the Legendre polynomials
    return -B(Q,l)*l*(((rho**2 + z**2)/(a**2))**(l/2))*(1/(a*math.sqrt(rho**2 + z**2)))*Plnm1
#
def Erhoout(Q,a,l,rho,z,Pln,Plnm1):
    cscth=abs(z/(math.sqrt(z**2 + rho**2))) #cos theta term ==> coefficient of the Legendre polynomials
    return A(Q,l)*(((a**2)/(rho**2 + z**2))**(l/2))*((rho**2 +z**2)**(-3/2))*(((l*(rho**2 - z**2)+rho**2)/rho)*Pln+l*(z/rho)*math.sqrt(z**2 + rho**2)*Plnm1)
#
def Ezout(Q,a,l,rho,z,Pln,Plnm1):
    cscth=abs(z/(math.sqrt(z**2 + rho**2))) #cos theta term ==> coefficient of the Legendre polynomials
    return A(Q,l)*(((a**2)/(rho**2 + z**2))**(l/2))*((rho**2 +z**2)**(-3/2))*(z*(2*l+1)*Pln-l*math.sqrt(z**2 + rho**2)*Plnm1)
#
```

```
73
74 #next, we need to import grid data from grid text file
75 N=3000 #number of disks
76 nmdisk=path+'\\Results\\'+f'DiskGrid(N={N}).txt' #information of the disks
77 fd=open(nmdisk,'r')
78 gridarray=[[float(num) for num in line.split(' ')] for line in fd] #read
       entire file
79 numrows=len(gridarray) #number of rows
80 numcol=len(gridarray[0]) #number of columns
81 #defining empty arrays
82 X=[0 for i in range(numrows)]
83 Y=[0 for i in range(numrows)]
84 R=[0 for i in range(numrows)]
85 V0=[0 for i in range(numrows)]
86 Q=[0 for i in range(numrows)]
87 #storing values appropriately
88 for i in range(numrows):
89     X[i]=gridarray[i][0]
90     Y[i]=gridarray[i][1]
91     R[i]=gridarray[i][3]
92     V0[i]=gridarray[i][4]
93     Q[i]=gridarray[i][5]
94 #
95 #next, we need to define the trajectory of the dot moving with zdot
       constant
96 #
97 #If we have a straight and parallel line, this section of the code will be
       used. We need to choose the type of line. specify choice by converting
       desired line variable to "True"
98 parallelx=True
99 parallely=False
100 slopeline=False
101 #
102 # Now, we will define a function to generate results for one line at a
       time. We will then call this function using a multiprocessing tool.
103 def MultiLine(ln):
```

```
104     # If lines parallel to x-axis (constant y lines) ==> parallelx==True
105     if parallelx==True:
106         c0=0.0; c1=1.0 #coefficients for x (c0 gives value of x for
    constant x line)
107         d0=lowlim+prec*ln; d1=0.0 #coefficients of y (d0 gives value of y
    for constant y line)
108         #defining file name
109         nmener=path+'\\Results\\'+'\\N=3000(old)\\'+f'Energy(N={N},y0={
    round(d0,1)},Constant Y line).txt' #energy data file
110     #
111     # If lines parallel to y-axis ==> parallely==True
112     if parallely==True:
113         c0=lowlim+prec*ln; c1=0.0 #coefficients for x (c0 gives value of x
     for constant x line)
114         d0=0.0; d1=1.0 #coefficients of y (d0 gives value of y for
    constant y line)
115         #defining file name
116         nmener=path+'\\Results\\'+'\\N=3000(old)\\'+f'Energy(N={N},x0={
    round(c0,1)},Constant X line).txt' #energy data file
117     #
118     # If line has a non-zero slope (and non-infinity) ==> slopeline==True
119     if slopeline==True:
120         m=1.0 #slope of line
121         c=0.0 #y-intercept
122         c0=0.0; c1=1.0 #coefficients for x (c0 also determines x-intercept
    )
123         d0=m*c0+c; d1=m*c1 #coefficients for y
124         #defining file name
125         nmener=path+'\\Results\\'+f'Energy(N={N},m={m},x0={c0},y0={d0},
    Sloped line).txt' #energy data file
126     #
127     t0=0.0; tmax=100.0 #limits on t
128     inc=0.05
129     t=t0
130     #
131     #defining file to store values of energy in
```

```
132     fn1=open(nmener,"w")
133     fn1.close()
134     fn2=open(nmener,"a")
135     #fn2.write('Xdot Ydot Zdot tDot Udot\n')
136     #
137     #I am choosing to define it as a straight line. ==> This is the first
    (and outermost loop).
138     while t<=tmax:
139         #calculate x and y from t
140         x=round(c0+c1*t,3)
141         y=round(d0+d1*t,3)
142         #calculate rho for dot
143         rhodot=math.sqrt((x**2)+(y**2))
144         #defining total electric field
145         Etotx=0.0
146         Etoty=0.0
147         Etotz=0.0
148         Utot=0.0
149         #now we need to run a loop through all disks ==> This is the
    second loop
150         for i in range(N):
151             #rholim=(R[i]**2) - (zdot**2)
152             rhodotdisk=((X[i]-x)**2)+((Y[i]-y)**2) #rho squared of point
    with respect to disk
153             disdotdisk=rhodotdisk+(zdot**2) #actual distance of dot from
    center of disk
154             rhodotdisksqrt=math.sqrt(rhodotdisk)
155             Etotrhodisk=0.0
156             Etotzdisk=0.0
157             if disdotdisk<=(R[i]**2): #in this condition we have B's
158                 Pl=[]
159                 Pl.append(1)
160                 Pl.append(zdot/(math.sqrt(disdotdisk)))
161                 #
162                 l=1 #beginning count on number of terms
```

```
163             if disdotdisk<=(0.000001*(R[i]**2)): #doing an additional cutoff check ==> if the point is very close to center, only one term is enough
164                 Ezinterm=Ezin(Q[i],R[i],l,rhodotdisksqrt,zdot,Pl[1],Pl[0])
165                 Erhointerm=Erhoin(Q[i],R[i],l,rhodotdisksqrt,zdot,Pl[1],Pl[0])
166                 Etotrhodisk+=Erhointerm
167                 Etotzdisk+=Ezinterm
168             #
169             else:
170                 Ein=False #variable to decide whether to end while loop to count number of terms
171                 while Ein==False: #loop runs until sum is under a tolerable level
172                     if l==1: Pln=Pl[1]; Plnm1=Pl[0]
173                     else :
174                         Plnm1=P(l-1,zdot/math.sqrt(disdotdisk),Pl[l-2],Pl[l-3])
175                         Pl.append(Plnm1)
176                         Pln=P(l,zdot/math.sqrt(disdotdisk),Pl[l-1],Pl[l-2])
177                         Pl.append(Pln)
178                     #
179                     Ezinterm=Ezin(Q[i],R[i],l,rhodotdisksqrt,zdot,Pln,Plnm1)
180                     Erhointerm=Erhoin(Q[i],R[i],l,rhodotdisksqrt,zdot,Pln,Plnm1)
181                     #print(x,y,l,Plnm1, f'Erhointerm={Erhointerm}', f'Erhoinsum={Etotrhodisk}', f'Ezterm={Ezinterm}', f'Ezsum={Etotzdisk}')
182                     if abs(Erhointerm)<=(0.0001*abs(Etotrhodisk)) and abs(Ezinterm)<=(0.0001*abs(Etotzdisk)): Ein=True #if wihtin tolerance, end loop
183                     Etotrhodisk+=Erhointerm
184                     Etotzdisk+=Ezinterm
185                     l+=2
```

```
                    #
                #
            #
            else: #in this condition we have A's
                Pl=[]
                Pl.append(1)
                Pl.append(zdot/(math.sqrt(disdotdisk)))
                #
                l=0 #beginning count on number of terms
                if disdotdisk>=(10000*(R[i]**2)): #doing additiobnal
cutoff check ==> if point is very far away from disk, use only one term
                    Erhooutterm=Erhoout(Q[i],R[i],l,rhodotdisksqrt,zdot,Pl
[0],0)
                    Ezoutterm=Ezout(Q[i],R[i],l,rhodotdisksqrt,zdot,Pl
[0],0)
                    Etotrhodisk+=Erhooutterm
                    Etotzdisk+=Ezoutterm
                #
                else:
                    Ein=False #variable to decide whether to end while
loop to count number of terms
                    while Ein==False: #loop runs until sum is under a
tolerable level
                        #print(l,P(l,zdot/math.sqrt(disdotdisk)))
                        if l==0: Pln=Pl[0]; Plnm1=0
                        elif l==2:
                            Plnm1=Pl[1]
                            Pln=P(l,zdot/math.sqrt(disdotdisk),Pl[l-1],Pl[
l-2])
                            Pl.append(Pln)
                        #
                        else :
                            Plnm1=P(l-1,zdot/math.sqrt(disdotdisk),Pl[l
-2],Pl[l-3])
                            Pl.append(Plnm1)
```

```
                            Pln=P(l,zdot/math.sqrt(disdotdisk),Pl[l-1],Pl[l-2])
                            Pl.append(Pln)
                        #
                        Erhooutterm=Erhoout(Q[i],R[i],l,rhodotdisksqrt,zdot,Pln,Plnm1)
                        Ezoutterm=Ezout(Q[i],R[i],l,rhodotdisksqrt,zdot,Pln,Plnm1)
                        if abs(Erhooutterm)<=(0.0001*abs(Etotrhodisk)) and abs(Ezoutterm)<=(0.0001*abs(Etotzdisk)): Ein=True #if wihtin tolerance, end loop
                        Etotrhodisk+=Erhooutterm
                        Etotzdisk+=Ezoutterm
                        l+=2
                    #
                #
            #
            #Erho needs to be added vectorially, so we need to convert it to Ex and Ey
            Etotxdisk=((x-X[i])/(rhodotdisksqrt))*Etotrhodisk
            Etotydisk=((y-Y[i])/(rhodotdisksqrt))*Etotrhodisk
            Etotx+=Etotxdisk
            Etoty+=Etotydisk
            Etotz+=Etotzdisk
            #this gets repeated for all disks
        #
        Utot=alpha*((Etotx**2)+(Etoty**2)+(Etotz**2))*(10**(12)) #total energy at the point(x,y,zdot) due to all disks
        #storing the values in a file
        fn2.write(f'{round(x,3)} {round(y,3)} {zdot} {round(t,3)} {Utot} {Etotx*(10**6)} {Etoty*(10**6)} {Etotz*(10**6)}\n')
        #increments go in last
        t=t+inc
        #if blowup==True: fcon2.write(f'\n\nPoint Change\n')
    #
    fn2.close()
```

```
#
#
# Now that we have our function completeley defined, we will call the
    function from a multiprocessing unit.
# range of lines to generate, with precision
prec=0.2
uplim=0.0
lowlim=0.0
noflines=int((uplim-lowlim)/prec) #number of lines to generate
if __name__ == '__main__':
    pool_obj=multiprocessing.Pool(processes=5)
    pool_obj.map(MultiLine,range(0,noflines+1))
    pool_obj.close()
#
#measuring end time
end=time.time()
#printing time taken for code to run
print(f'Time taken by code to run = {end-begin}')
```