2011

Tunneling Conductance Characterization of a Quantum Dot in the Fractional Quantum Hall Regime

Douglas E. Willard
University of Central Florida

TUNNELING CONDUCTANCE CHARACTERIZATION OF A QUANTUM DOT IN THE FRACTIONAL QUANTUM HALL REGIME

by

DOUGLAS E. WILLARD
B.S. University of South Florida, 1991
M.S. University of Central Florida, 2000

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Physics
in the College of Sciences
at the University of Central Florida
Orlando, Florida

Summer Term
2011

Major Professor: Michael D. Johnson

ABSTRACT

This work represents a first-principles calculation of the electron tunneling current into quantum dots in the fractional quantum Hall effect regime. The system under consideration consists of an idealized Scanning Tunneling Microscope (STM) tip and a quantum dot with disk geometry and interacting electrons in a transverse magnetic field. Within the context of this model the tunneling current between the STM tip and the dot is examined for spin-polarized electrons at and around a filling factor of 1/3.

The current expression is based on a second-quantized Hamiltonian in which electrons in the dot are interacting, confined, and restricted to the lowest Landau level, necessary to capture the physics of the fractional quantum Hall effect. The Hamiltonian includes simple approximations for the STM tip and the tip-dot tunneling. An exact analytic expression for the first-order tunneling current is derived using a Green's function approach. To calculate the tunneling current numerically the infinite Hilbert space of the dot is truncated to have a finite dimension within the lowest Landau level. This simplification is appropriate for a low temperature system in the fractional quantum Hall regime because of the finite size of the quantum dot and the large energy gap between Landau levels. The tunneling current is then solved in two steps. First, many-electron energy eigenstates are calculated from the truncated Hamiltonian by numerical diagonalization. This is carried out for varying numbers of electrons N. The energy eigenstates form a set of complete basis states of the system and are used in the expression for the tunneling current. In the second step, the chemical potential in the dot is chosen to select a desired number of electrons and the tunneling current evaluated. We have carried out this program for filling factors near $1/3$ while modulating the system parameters of interest to determine functional dependencies.

To my loving family...Susie, Matthew and Julia.

ACKNOWLEDGMENTS

I am grateful to everyone who supported and encouraged me throughout this process. I would like to acknowledge and thank my advisor, Dr. Michael D. Johnson, for his patient and thoughtful direction through the years.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

The study of electron transport has enjoyed a productive and rich scientific history for well over a century. Interest in the electron's behavior and interaction with the world around it predates even the experimental identification of the electron itself in 1897. [1] One of the early areas of such investigation involved the effect of a magnet, and more specifically a magnetic field, on an electric current. In 1879, Edwin Herbert Hall conducted experiments on current-carrying conductors in the presence of an orthogonal magnetic field. Using a galvanometer he discovered a current across the width of the conductor that flowed in a direction perpendicular to both the conductor and the magnetic field. This transverse current created a transverse potential difference across the conductor's width. [2] Today Hall's name is associated with both the transverse voltage and this classical phenomenon known as the Hall Effect. [3]

Moving forward approximately one hundred years, after the development of quantum mechanics, condensed matter theory, electronic heterojunctions, and modern experimental techniques, it became possible to further investigate electron transport but this time from another perspective. Some of this more recent work also studied electron transport in the presence of a transverse magnetic field, B, but in contrast to the classical Hall Effect experiments, at much lower temperatures (1 Kelvin or lower) and much higher magnetic fields (typically 1 T or higher). At low temperatures and in high-mobility materials, two-dimensional electron gases can be created using semiconductor heterostructures. These systems display phenomena that is quantized in nature and described inherently by quantum mechanics. These two-dimensional electron gases (2DEG) environments can be further dimensionally constrained into flat dots with diameters on the order of ten nanometers that display phenomena which are quantized in nature and inherently described by quantum mechanics.

One such phenomenon is the Integer Quantum Hall Effect (IQHE), discovered by Klaus von Klitzing in 1980 [4], and for which he was awarded the Nobel Prize in Physics in 1985. Later experiments at even lower temperatures and with even cleaner samples led to the discovery of the

Fractional Quantum Hall Effect (FQHE) by Daniel C. Tsui, Horst L. Störmer and Arthur C. Gossard in 1982. [5] For their experimental discovery Tsui and Störmer, and for his theoretical explanation of the FQHE, Robert Laughlin, were awarded the Nobel Prize in Physics in 1998. Collectively these discoveries are referred to as the Quantum Hall Effects (QHE). [6]

Although the integer and fractional quantum Hall effects are distinct and occur for different reasons, they do share some similar characteristics which set the QHE apart from the classical effect. They are only seen experimentally under the very specific conditions of low temperatures, large B, high mobility and limited dimensionality. These characteristic combine to create a system with a discrete energy spectrum which is essential for the quantized behavior of the QHE. [7]

Experimentally the QHE is seen in two-dimensional electron gases systems (2DEG) which are created at the interface of dissimilar semiconductors. The conduction band electrons are spatially trapped at this interface, referred to as the inversion layer, by an electrostatic force. There are several methods and devices used to create this layer by way of an electrostatic force, but one of the more common experimental devices used to produce the 2DEG is the Silicon Metal Oxide Semiconductor Field Effect Transistor (MOSFET). By application of a voltage across a MOSFET, the electron conduction band energy can be adjusted such that it falls below the Fermi energy, E_F. This traps the conduction electrons in a potential well and they are confined to a quasi-two-dimensional layer. The electrons' state perpendicular to the plane stays (at low temperatures) in an unchanging quantized state. Confined in this one direction, the confined electrons are free to move in a plane that is parallel to the interface of the 2DEG. Their energy spectrum is given to high accuracy by

$$E = E_{BAND} + \frac{\hbar^2 k^2}{2m^*}. \tag{1.1}$$

The wave number, k, corresponds to the particle's in-plane wave vector. The effective mass, m^*, is a material dependant quantity related to the conduction band's curvature. The subband energy E_{BAND},

are the energies at the bottom of the subbands of the system. [3, 8]

1.1 Landau Levels

Now consider free electrons confined to a 2DEG and in the presence of a perpendicular, uniform, magnetic field $\vec{B}$. Classically an electron with velocity $\vec{v}$ will experience a Lorentz force

$$\vec{F} = -e\left(\vec{E} + \frac{\vec{v}}{c} \times \vec{B}\right) \tag{1.2}$$

and follow circular paths around a guiding center. Here $-e$ is the electron's charge and c is the speed of light. From Helmholtz's theorem we introduce $\vec{B} = \vec{\nabla} \times \vec{A}$ where $\vec{A}$ is the magnetic vector potential. [9] The electron's motion can be described using a Hamiltonian

$$\hat{\mathscr{H}} = \frac{\Pi^2}{2m^*} \tag{1.3}$$

where the generalized momentum $\hat{\Pi}$ is

$$\hat{\Pi} = -i\hbar\nabla + \frac{e}{c}\vec{A}. \tag{1.4}$$

Assuming the magnetic field is in the z-direction $\vec{B} = B\hat{k}$ and normal to the 2DEG we can separate the momentum into planar motion components

$$\hat{\mathscr{H}} = \frac{\left(\hat{\Pi}_X^2 + \hat{\Pi}_Y^2\right)}{2m^*}. \tag{1.5}$$

This can be analyzed using a central and relative coordinate method [10, 11] by defining the central coordinates (X,Y) of the electron's motion

$$X \equiv x - \xi \tag{1.6}$$

and

$$Y \equiv y - \eta \tag{1.7}$$

where the relative coordinates of the electron's motion ξ and η are

$$\xi = \left(\frac{c}{eB}\right)\hat{\Pi}_Y \tag{1.8}$$

and

$$\eta = -\left(\frac{c}{eB}\right)\hat{\Pi}_X. \tag{1.9}$$

From these relative coordinates emerges the cyclotron frequency ω_c [9]

$$\omega_c = \frac{eB}{m^* c}. \tag{1.10}$$

Selecting the Landau gauge vector potential $\vec{A} = [0, Bx, 0]$ the relative coordinates become

$$\xi = \frac{c}{eB}\left(-i\hbar\frac{\partial}{\partial x}\right) \tag{1.11}$$

and

$$\eta = \frac{c}{eB}\left(-i\hbar\frac{\partial}{\partial y} + \frac{e}{c}Bx\right). \tag{1.12}$$

To consolidate our expression we now consider the commutation relations of the relative coordinates

$$[\xi, \eta] = \left(\frac{c}{eB}\right)^2 \left(\hat{\Pi}_Y\hat{\Pi}_X - \hat{\Pi}_X\hat{\Pi}_Y\right). \tag{1.13}$$

Assuming the operators are acting on continuous, differentiable state functions we can say

$$\frac{\partial^2}{\partial x \partial y} = \frac{\partial^2}{\partial y \partial x} \tag{1.14}$$

and therefore

$$[\xi,\eta] = \frac{i\hbar c}{eB}\left(-x\frac{\partial}{\partial x} + \frac{\partial}{\partial x}x\right). \tag{1.15}$$

By defining the magnetic length ℓ_0,

$$\ell_0 \equiv \sqrt{\frac{\hbar c}{eB}}, \tag{1.16}$$

which is the radius of the circular path taken by the electron, the resulting commutation relation becomes

$$[\xi,\eta] = i\ell_0^2. \tag{1.17}$$

The Hamiltonian can then be written as

$$\hat{\mathscr{H}} = \frac{\hbar\omega_c}{2\ell_0^2}\left(\xi^2+\eta^2\right). \tag{1.18}$$

Because of the commutation relations this can be seen to have the form of a harmonic oscillator Hamiltonian; the energy eigenvalues are

$$E_n = \left(n+\frac{1}{2}\right)\hbar\omega_c, \tag{1.19}$$

indexed $n = 0,1,2,3,\ldots$. These quantized energy bands can be seen in Figure 1.1 and are known as Landau Levels. They are formed when our 2DEG system is placed in a uniform magnetic field. The bands are degenerate and the spacing between the bands grows proportionally to the magnetic field strength. The band with the lowest energy, $n = 0$, is the Lowest Landau Level (LLL). [12, 6] It turns out that the IQHE depends only on the single-particle physics outlined here, and in particular on the discreteness of the Landau level energy spectrum. This will be discussed further in the following section.

Another significant aspect of systems that exhibit the QHE is the scale of the current carrying conductors involved. In order to manifest the QHE a system must have dimensions that lie between the macroscopic and microscopic or atomic scale, sizes referred to as 'mesoscopic.' The size of a

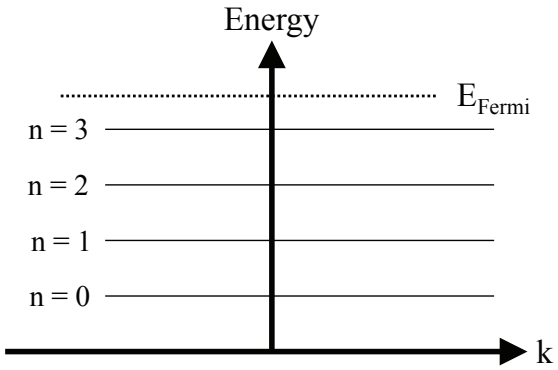

Figure 1.1 Landau levels of a system in a uniform magnetic field

conductor required to fall into this mesoscopic region vary as this is material and system dependent, however in general the conductor's dimensions must be no larger than any of the three characteristic lengths; the De Broglie wavelength, the mean free path and the phase relaxation length. The De Broglie wavelength is inversely proportionate to the electron's momentum. The mean free path is the average distance traveled by an electron before its initial momentum is changed. The relaxation length is the average distance traveled by an electron before its initial phase is lost. [13] A remarkable aspect of systems exhibiting the QHE is that, because of the properties of Landau levels, systems can remain mesoscopic even at millimeter lengths.

1.2 The Hall Bridge

Discoveries of the IQHE by Klaus von Klitzing in 1980 [4] and the FQHE by D. C. Tsui, H. L. Störmer, and A. C. Gossard in 1982 [5] were accomplished experimentally by systematically measuring voltages of semiconductor devices in a transverse magnetic field. These experiments used different semiconductor technologies, a MOSFET for the IQHE and a GaAs-AlGaAs heterojunction for the FQHE, but both used a similarly shaped current carrying conductor or bridge. Using semiconductor lithography techniques a multi-terminal circuit can be made that allows for the measurement of current and voltages. The device is commonly referred to as a Hall Bridge (see Figure 1.2). A potential bias is applied from source to sink of the bridge such that a longitudinal current, $J_{(tot)}$, results in the presence of a uniform, transverse magnetic field $\vec{B}$.

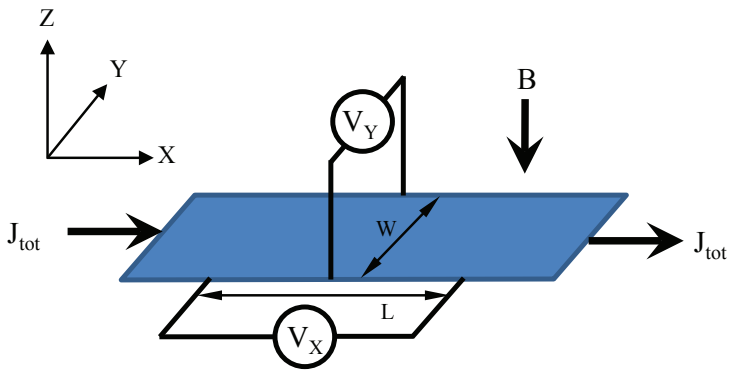

Figure 1.2 Hall Bridge

Two different resistivity quantities can be obtained from this circuit. The longitudinal resistivity Rxx is the dissipative resistivity measured in the same direction as the electron's drift velocity. This is associated with the longitudinal voltage V_x, measured along the length of the conductor L shown in Figure 1.2 and can be thought of as the 'normal' resistivity. The transverse or Hall resistivity Rxy, is measured across the width of the conductor and is transverse to the direction of the drift velocity.

One of the most significant aspects of the Hall Bridge is the confinement that the device itself establishes in the transverse direction across the conductor. This confinement creates transverse modes for the electrons and is one of the necessary elements for the QHE phenomenon. This physical confinement is represented through a confinement potential $W(\vec{r})$. The Schrödinger equation with this confinement potential is

$$\left[E_{BAND} + \frac{\left(i\hbar\nabla + \frac{e}{c}\vec{A}\right)^2}{2m^*} + W(\vec{r})\right]\Psi = E\Psi. \tag{1.20}$$

One can select potentials that capture the physics and also allow for an analytic solution. As an example, for a system with a parabolic confining potential $W(\vec{r}) = \Omega r^2$ and a transverse magnetic field, the resulting eigenenergies are

$$E = E_{BAND} + \frac{\hbar^2 k^2}{2m^*} + \left(n + \frac{1}{2}\right)\hbar\omega_o \tag{1.21}$$

where k is the longitudinal wave vector and $\omega_o \propto \Omega$. This energy spectrum is shown in Figure 1.3 where the discrete energy bands can be seen. Here also the bands are enumerated by the index n = 0, 1, 2, 3, ... [13]

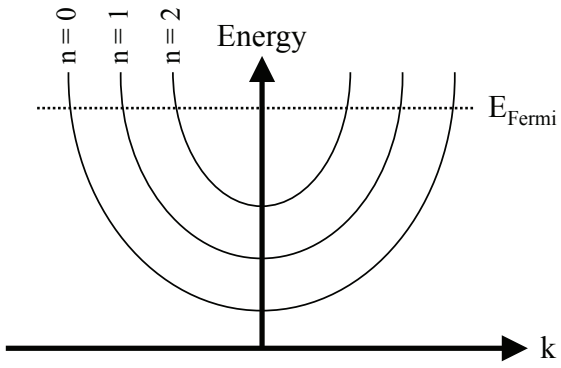

Figure 1.3 Landau levels of a system with a confinement potential

1.3 Integer Quantum Hall Effect

At low temperatures, clean 2 DEG systems start to display the IQHE - quantized steps in the resistivity as a function of magnetic field. [14, 15] Figure 1.4 shows the results of these experiments along with the Hall resistivity predicted by the classical theory. There is an obvious difference between the plateaus seen experimentally and the linear resistivity predicted by classical theory. The plateaus in the Hall resistivity correspond to zero longitudinal or diagonal resistivity (*bottomfigure*). These plateaus occur at specific quantized values when the Hall resistivity is

$$R_{xy} = \frac{2\pi\hbar}{ne^2} \tag{1.22}$$

where n in an integer.

This can be analyzed in terms of non-interacting electrons and the degeneracy of the LL. In a system of finite area, A, each LL has a fixed number of degenerate single-particle states available. The number of states per LL is the degeneracy and depends on the magnetic field as

$$M = \frac{BA}{\phi_0}, \tag{1.23}$$

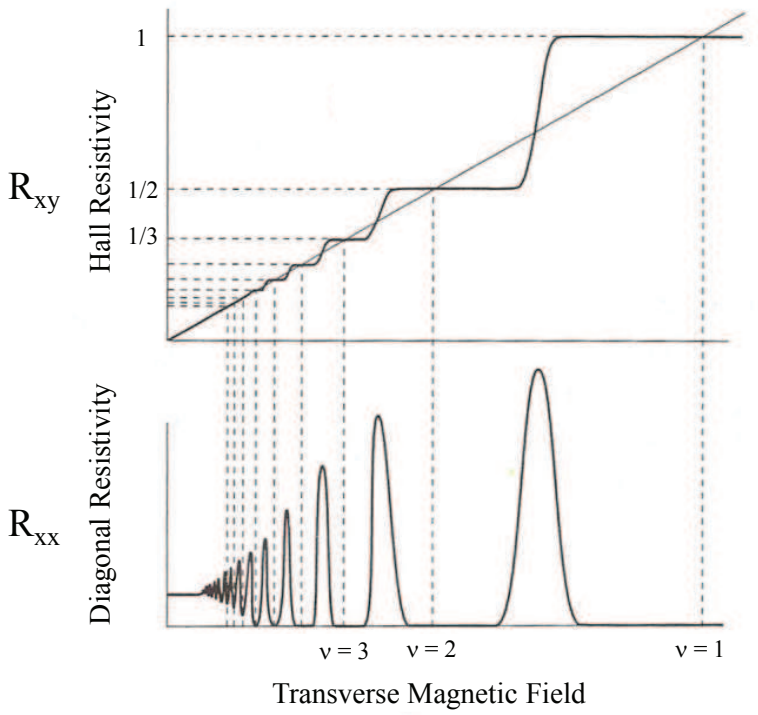

Figure 1.4 IQHE - Resistivity as a function of magnetic field

where A is the sample area and ϕ_0 is the flux quantum given by

$$\phi_0 = \frac{2\pi\hbar c}{e}. \tag{1.24}$$

Electrons are fermionic in nature and as such only one of a given spin can occupy a given state. As electrons are added to a system each LL is filled before moving to the next higher energy LL. The reason for this is the energy gap between LLs. The electrons must pay an energy gap penalty in order to get to the next higher LL and they avoid this penalty by filling each level entirely until beginning to fill the next.

When all of the available states in a LL are filled there will be M electrons in that LL and the same number of flux quanta, i.e. exactly one flux quanta per electron. It is useful to define a LL filling factor in terms of the degeneracy of each LL as

$$\nu = \frac{N}{M} \tag{1.25}$$

where N is the number of electrons. Therefore when then number of electrons is equal to an integer multiple of the degeneracy, $N = nM$, the next single electron to be added to the system will start to fill the next higher LL and pay the energy gap price.

During the process of filling a given LL, but before it is full, there is no energy gap penalty to pay for another electron to be added. This is often described in terms of a compressible fluid. As long as there is sufficient volume in a container, in this case any given LL, the next drop is added easily. However, as soon as the volume of the container is filled, the now incompressible fluid will not allow any further filling and the particle must go to next LL and pay the associated energy gap penalty. This energy gap avoidance by a single electron is the cause of the IQHE. IQHE plateaus occur for filling factors ν centered on integers.

1.4 Fractional Quantum Hall Effect

The FQHE appears in similar systems, but for even lower temperatures, higher magnetic fields and higher mobility samples. The IQHE and FQHE share many of the same characteristics including the plateaus in the Hall resistivity and corresponding dips in the longitudinal resistivity. The similarity is in fact deceptive as they occur for different reasons. Both phenomena are fundamentally due to energy gaps in the electronic spectrum, but the mechanisms that cause these gaps are completely different. In the IQHE case, the energy gaps are between the single-electron states in different LL. In contrast the energy gaps for the FQHE are due to electron-electron interactions and come from the energy differences between many-electron states. The FQHE is therefore fundamentally a many-body effect in strongly correlated electron systems while the IQHE is not. The higher magnetic fields required for the FQHE separate the LL even further apart than those of the IQHE systems. Higher mobilities are also required so that scattering effects are diminished to the point where this low-energy interaction effect can be seen.

A brief explanation of the FQHE can be provided by analyzing electron-electron interactions in Landau levels. For electrons in Landau levels, the electron-electron interaction can be decomposed into relative angular momentum (m) 'channels' with strength V_m. The interaction energy of each

electron pair depends on the pair's relative angular momentum m (limited to values $m = 1,3,5,\ldots$ by the anti-symmetry of electrons). Because the electron-electron interaction falls off with distance, it turns out that $V_1 > V_3 > V_5 > \ldots$. To avoid this energy gap cost, and in particular the V_1 energy, the multi-electron states combine in such a way as to avoid lower values of relative angular momentum. For example, at $\nu = \frac{1}{3}$ no electron pair has $m = 1$. Consequently excitations in the bulk at $\nu = \frac{1}{3}$ have an energy gap of order V_1. Thus the bulk is incompressible in a similar sense to that of the IQHE while any low-energy excitations occur at the edge.

1.5 Edge States

The energy gaps described above are for excitations in the incompressible bulk. Low energy (gapless) excitations still occur at edges. As a result an edge state picture has been developed that is able to explain both the IQHE and FQHE. The edge state picture provides a very appealing way of describing the QHE as, in both cases, the electron transport can be viewed as being carried by gapless edge excitation modes with an incompressible bulk. In quantum Hall systems, the 1D edge states are 'chiral'— that is, electrons travel only in one direction on a given edge (and the opposite direction on the other edge). In the case of the IQHE the edge states can be described theoretically as non-interacting one-dimensional (1D) chiral Fermi gases. However, because the FQHE arises from electron-electron interactions, an approximation of edge states in the system is based on 1D systems of interacting electrons. Wen predicted that this picture results in a 1D Chiral Luttinger Liquid (χLL) model. [16, 17] This model descends from Landau's recognition that interacting electron systems in 3D behave similarly to noninteracting Fermi gases — the 'Fermi liquid' model. The Landau Fermi liquid approach fails in 1D, even for weak interactions. Haldane predicted that most 1D systems of interacting particles should behave similarly to the Luttinger Model (a particular, ideal, solvable model of interacting 1D electrons) and, in analogy to Landau's term 'Fermi liquid' called these 'Luttinger Liquids'. Wen argued that edge states of FQHE systems behave like a 1D χLL system with parameters related to the filling factor, which followed from the incompressibility of the bulk for a system with sharp edges. [18, 16]

1.6 Experimental Results

One way to probe these postulated edge states is by using a STM tip to tunnel into and out of the edge states of a Hall system. Theoretical 1D χLL predictions about tunneling into edge states in a QH system gave a power law functional dependence with the exponent proportional to ν. Tunneling experiments confirmed this power law behavior at the specific predicted filling factor $\nu = \frac{1}{3}$. Thus the theoretically predicted χLL behavior appeared to be confirmed experimentally. These initial experiments, however, were only carried out at the one particular fractional filling factor $\nu = \frac{1}{3}$.

Later experiments repeated these measurements for filling factors at and around $\nu = \frac{1}{3}$ and obtained the same power law and exponents. It was also experimentally established that the tunneling results did not depend on whether the bulk was incompressible (exhibiting the QHE) or not. [19] This was surprising since the picture of QHE physics, including the chiral LL edge states, only makes sense for incompressible systems. Nonetheless the result was the same even with compressible systems where the edge state picture falls apart. Attempts to explain this have been put forth, e.g. a suggestion that just being in the Lowest Landau Level (LLL) gives rise to Luttinger Liquid-like behavior of excitations. [20]

The present work was motivated by this conundrum. The edge state picture, and the χLL model, are based on sweeping uncontrolled approximations. We instead proceed from first principles. This permits an unbiased microscopic examination of tunneling into quantum dots in the FQHE regime, in and around $\nu = \frac{1}{3}$. For the model we develop, results are numerically exact. This work represents the first first-principles calculation of tunneling into quantum dots in the FQHE regime.

The remainder of this dissertation are organized as follows. Chapter 2 develops the tunneling model that we will study. This incorporates electron-electron interactions exactly within a quantum dot—important to capture the physics of the FQHE—and uses a simple model of tunneling from a STM tip. Chapter 3 rewrites the resulting expression for tunneling current into a form suitable for numerical calculation. Chapter 4 describes the approach we take to perform the numerical analysis and details the specific methods used to numerically calculate the tunneling current as a function of the system's parameters. Chapter 5 includes the results of a systematic numerical analysis to

determine the tunneling current around filling factor $\nu = \frac{1}{3}$.

CHAPTER 2
TUNNELING MODEL

This chapter develops the model we will use to study tunneling into a quantum dot in the FQHE regime. A Hamiltonian formalism and the language of second quantization are used to mathematically model the tunneling system. Using this model as a starting point, an analytic expression for the tunneling current, I, will be derived using a Green's function approach. This current is expressed in terms of the system variables to be investigated: the temperature, magnetic field B, chemical potential μ and radial coordinate r from the center of the QD to the position of the tip where the electron tunneling occurs.

The model of this tunneling system consists of a Scanning Tunneling Microscope (STM) tip referred to as the 'lead', a QD with disk geometry or 'dot', and a gap across which electrons may tunnel. The STM and QD are assumed to be completely independent systems and only linked through the tunneling of electrons. The STM tip is assumed to be semi-infinite and consist of non-interacting electrons. The location of the STM tip is a system parameter that establishes the radial position where tunneling occurs. The QD consists of a 2DEG with a confining potential W(r) that restricts the motion of the conduction band electrons to a disk shaped area in a 2-D plane. The energy associated with the motion normal to the 2DEG plane is therefore frozen out. It is further assumed that the QD material has high mobility and the environment, strong magnetic field and low temperature, is such that all of conduction band electrons are in the Lowest Landau Level (LLL). Electron-electron interactions in the QD are included so that the confined electrons exhibit FQHE physics. The STM and QD, designated by the subscripts 'L' and 'D' respectively, are held at different chemical potentials μ_L, μ_D. These potentials are the energies associated with transferring a particle to or from the lead and dot respectively. The difference in these chemical potentials is the driving tunneling voltage that creates the tunneling current under consideration and we define this term as

$$eV \equiv \mu_L - \mu_D. \tag{2.1}$$

The quantum mechanical Hamiltonian and tunneling operator are constructed of electron creation $c^{\dagger}_{\vec{k}}$, and annihilation $c_{\vec{k}}$, operators in the case of the lead or $\psi^{\dagger}(\vec{r})$ and $\psi(\vec{r})$ in the dot. It is therefore worth being explicit as to what is meant by 'current' at this mesoscopic scale. Current is obtained by inspecting the change in the number of electrons in either the lead or the dot as electrons tunnel from one to the other. In second quantized language, this is done by evaluating the time derivative of a counting operator $\hat{N}_L$:

$$I(t) = -e\left\langle \frac{d\hat{N}_L}{dt} \right\rangle \tag{2.2}$$

where

$$\hat{N}_L = \sum_{\vec{k}} c^{\dagger}_{\vec{k}} c_{\vec{k}} \tag{2.3}$$

is the number of electrons in the lead. The bra and ket angle brackets represent the statistical average over all of the multi-particle states of the system. Since the system allows for particle exchange, the Grand Canonical Ensemble from statistical mechanics is used to perform the thermal average. This is provided by a sum over a complete basis of many-particle states α for all possible numbers of particles. The term in our current expression that undergoes this averaging becomes

$$\left\langle \frac{d\hat{N}_L}{dt} \right\rangle \equiv \sum_{\alpha} \left\langle \alpha \left| \hat{\rho} \frac{d\hat{N}_L}{dt} \right| \alpha \right\rangle \tag{2.4}$$

where the density matrix is

$$\hat{\rho} = \frac{1}{Z} e^{-\beta\left(\hat{\mathscr{H}} - \mu\hat{N}\right)}. \tag{2.5}$$

Here Z is the grand partition function and β is the inverse of the product of the Boltzmann constant k_b and temperature T. It is convenient to choose α to be many-particle energy eigenstates with energy E_α and particle number N_α so that

$$\left\langle \frac{d\hat{N}_L}{dt} \right\rangle = \frac{1}{Z} \sum_{\alpha} e^{-\beta(E_\alpha - \mu N_\alpha)} \left\langle \alpha \left| \frac{d\hat{N}_L}{dt} \right| \alpha \right\rangle. \tag{2.6}$$

The many-particle energy eigenstates are central to the statistical averaging and computation of the current, and a key part of this dissertation will lie in their direct numerical calculation. It is helpful at this point to introduce the appropriate terminology. The many-particle energy eigenstates $|\alpha\rangle$ or Ψ_α will be calculated by expanding them in terms of a complete basis of Multi-Particle States (MPS) $|J\rangle$ or Ψ_J. The MPS, in turn, are constructed as Slater determinants of appropriately chosen Single-Particle States (SPS) $\phi_m(\vec{r})$. The connection between the single-particle ket (in second quantization) and wave function (in first quantization) is

$$\phi_m(\vec{r}) = \langle \vec{r} | \, c_m^\dagger \, |0\rangle \tag{2.7}$$

where $|0\rangle$ represents the vacuum and $c_m^\dagger$ creates an electron in the SPS labeled by m.

The MPS basis states Ψ_J can then be constructed from the SPS wave functions using a Slater determinant. One such basis state is:

$$J(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = \begin{vmatrix} \phi_1(\vec{r}_1) & \phi_2(\vec{r}_1) & \phi_3(\vec{r}_1) & \cdots & \phi_N(\vec{r}_1) \\ \phi_1(\vec{r}_2) & \phi_2(\vec{r}_2) & \phi_3(\vec{r}_1) & \cdots & \phi_N(\vec{r}_2) \\ \phi_1(\vec{r}_3) & \phi_2(\vec{r}_3) & \phi_3(\vec{r}_1) & \cdots & \phi_N(\vec{r}_3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \phi_1(\vec{r}_N) & \phi_2(\vec{r}_N) & \phi_3(\vec{r}_1) & \cdots & \phi_N(\vec{r}_N) \end{vmatrix}. \tag{2.8}$$

In second quantization this same state is represented

$$|J\rangle = c_1^\dagger c_2^\dagger c_3^\dagger \ldots c_N^\dagger \, |0\rangle \, . \tag{2.9}$$

Constructing all possible Slater determinants made up of all possible sets of SPS produces a complete set of MPS basis states which we denote by Ψ_J or $|J\rangle$.

One then can expand a general N-particle state $|\alpha\rangle$ in the N-particle MPS basis states $|J\rangle$ as

$$|\alpha\rangle = \sum_J b_J \, |J\rangle \tag{2.10}$$

where the coefficients b_J form an eigenvector in the MPS basis — that is, specify the weight of the various MPS basis states. Seeking many-particle energy eigenstates $H|\alpha\rangle = E_\alpha|\alpha\rangle$ produces a matrix equation that can be solved by numerical diagonalization. This is discussed further in Chapter 4.

2.1 Model Hamiltonian

We will use a Hamiltonian formulation consisting of a quantum dot, a lead, and a tunneling term that connects them. [21] Thus our model system consists of three distinct regions, two independent multi-body systems and a potential barrier region across which electrons pass between the two. Formally the same approach works for other systems (e.g. tunneling between metals) by appropriate choice of the three terms. [21] In our case the most important requirement is to capture the FQHE nature of the electrons within the QD. Our Hamiltonian accounts for this by including terms in the Hamiltonian for the kinetic, potential and interaction energies of electrons in the dot.

The Hamiltonian is separated into three distinct parts. Individually these are referred to as the Tunneling Hamiltonian ($\hat{\mathscr{H}_T}$), STM Tip or Lead Hamiltonian ($\hat{\mathscr{H}_L}$), and QD or Dot Hamiltonian ($\hat{\mathscr{H}_D}$):

$$\hat{\mathscr{H}} = \hat{\mathscr{H}_L} + \hat{\mathscr{H}_T} + \hat{\mathscr{H}_D}. \tag{2.11}$$

The Hamiltonian $\hat{\mathscr{H}_L}$ describes the electrons in the lead that culminates in the STM tip. Details of the tip are unimportant for this work, and so it is sufficient to model the lead as a semi-infinite sea of non-interacting electrons. Each state in the lead is labeled by the wave vector $\vec{k}$ with energy $\varepsilon_{\vec{k}}$. The second-quantized Hamiltonian in the lead, written in terms of creation $\left(c^\dagger_{\vec{k}}\right)$ and annihilation $\left(c_{\vec{k}}\right)$ operators which operate on noninteracting electrons in the lead, is

$$\hat{\mathscr{H}_L} = \sum_{\vec{k}} \varepsilon_{\vec{k}} c^\dagger_{\vec{k}} c_{\vec{k}}. \tag{2.12}$$

The dot Hamiltonian $\hat{\mathscr{H}_D}$ describes the behavior of the electrons in the QD. This Hamiltonian contains terms for the kinetic energy T, confinement potential W, and electron-electron interactions

V. In first quantization

$$\hat{\mathscr{H}_D} = \underbrace{\sum_k T(x_k) + W(x_k)}_{KE\ and\ PE} + \underbrace{\frac{1}{2}\sum_{k\neq l} V(x_k, x_l)}_{Interaction}. \tag{2.13}$$

In second quantization this becomes

$$\hat{\mathscr{H}_D} = \sum_{m,m\prime} \langle m|T+W|m\prime\rangle\, c_m^\dagger c_{m\prime}^\dagger + \frac{1}{2}\sum_{m_1,m_2,m_3,m_4} \langle m_1 m_2|V|m_3 m_4\rangle\, c_{m_1}^\dagger c_{m_2}^\dagger c_{m_3} c_{m_4}, \tag{2.14}$$

where m labels the SPS $|m\rangle$ or ϕ_m. The potential energy term $W(r)$ represents the walls of our system.

A natural choice for the SPS are energy eigenstates of the kinetic energy term [22],

$$T(x) = \frac{1}{2m^*}\left(\vec{p} + \frac{e}{c}\vec{A}\right)^2. \tag{2.15}$$

The particles in our system are limited to movement in 2-dimensions x and y, while the uniform magnetic field $\vec{B} = -B\hat{e}_z$, is aligned along z. In our case, with a disk geometry, it is convenient to choose the symmetric gauge vector potential

$$\vec{A} = (-By, Bx, 0). \tag{2.16}$$

From the literature [3], in this gauge the single-particle energy eigenstates in the lowest Landau level are:

$$\phi_m(z) = \sqrt{\frac{1}{2\pi\ell_0^2 2^m m!}}\left(\frac{z}{\ell_0}\right)^m \exp\left(-\frac{|z|^2}{4\ell_0^2}\right) \tag{2.17}$$

where

$$\ell_0 \equiv \sqrt{\frac{\hbar c}{eB}} \tag{2.18}$$

$$z \equiv \frac{x - iy}{\ell_0} \tag{2.19}$$

and $m = 0, 1, 2, \ldots$. Figure 2.1 shows a plot the single-particle energy eigenstates as a function of radius for different angular momentum quantum numbers m.

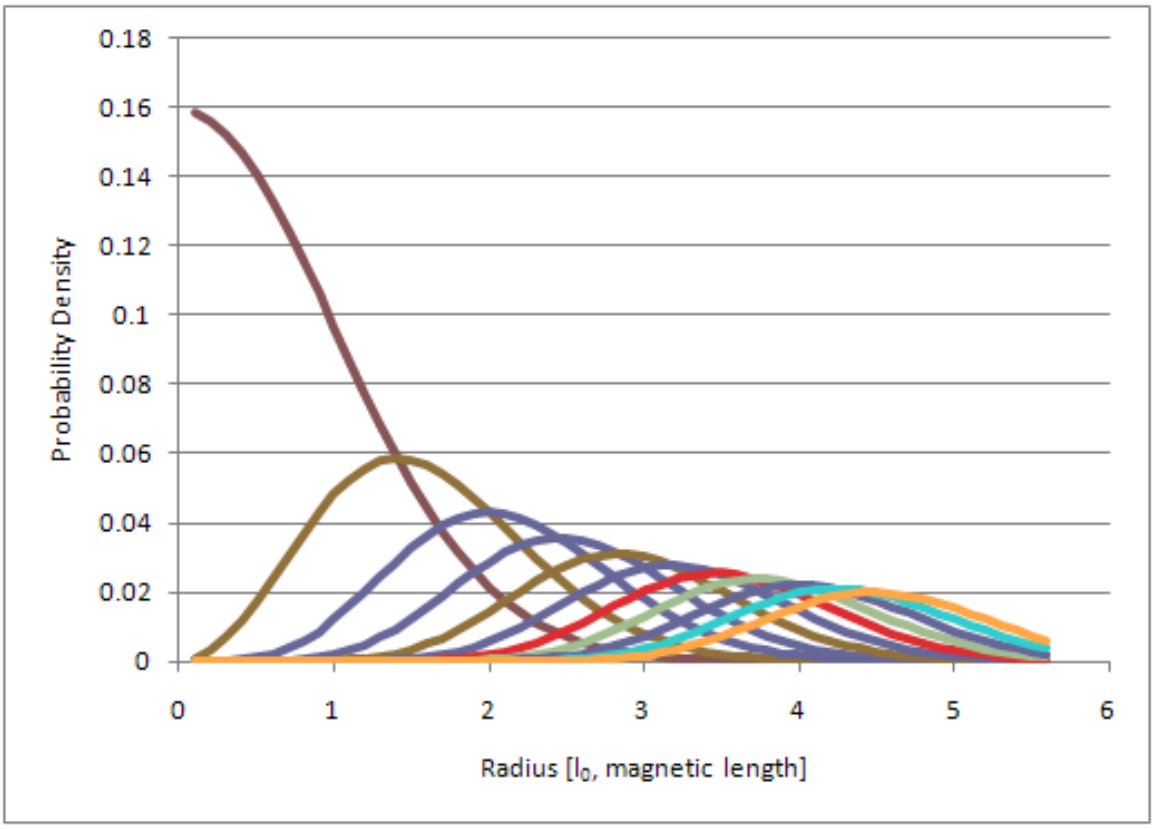

Figure 2.1 Single-Particle Energy Eigenstates as a Function Radius

The tunneling Hamiltonian $\hat{\mathscr{H}}_T$ describes the electrons that tunnel from a state $\vec{k}$ in the STM tip to a position $\vec{r}$ in the QD and vice versa. There are two operators used to create $\left(\psi^\dagger(\vec{r})\right)$ and destroy $(\psi(\vec{r}))$ electrons at a particular position $\vec{r}$ of the dot. The tunneling Hamiltonian is [23]:

$$\hat{\mathscr{H}}_T = \sum_{\vec{k}} \left(T_{\vec{k}}\, c^\dagger_{\vec{k}} \psi(\vec{r}) + T^*_{\vec{k}}\ \ \psi^\dagger(\vec{r}) c_{\vec{k}} \right). \tag{2.20}$$

Here $T_{\vec{k}}$ is the amplitude for tunneling from the dot into state $\vec{k}$ of the lead, and is assumed to be independent of the tip's position with respect to the dot.

The model assumes that the electrons in the lead and dot are strictly separate and therefore do not interact except via the $\hat{\mathscr{H}}_T$. This assumption is captured by asserting that the creation and annihilation operators for the lead and dot anti-commute (see Appendix A). As a result

$$\left[\hat{\mathscr{H}}_L, \hat{\mathscr{H}}_D\right] = 0. \tag{2.21}$$

In the statistical averaging we will assume that the dot and lead are held at different chemical potentials, μ_D and μ_L, and this difference is what controls the voltage drop across the gap ($eV = \mu_L - \mu_D$).

In the remainder of the chapter we work through steps that permit us to rewrite equation 2.2, using a Green's function formalism, into a form suitable for numerical evaluation.

2.2 Time Evolution Picture

The next step is to examine the time derivative operator $d\hat{N}_L/dt$ more precisely. There are several different 'pictures' of the time dependence of quantum mechanical operators and states, designated with the following subscript notation:

$$\hat{\mathscr{O}}_S \cdots \text{Schrödinger}$$
$$\hat{\mathscr{O}}_H \cdots \text{Heisenberg}$$
$$\hat{\mathscr{O}}_I \cdots \text{Interaction}$$

These pictures differ in whether the time dependance resides entirely within the operator, entirely within the states or some mixture in between. [24] Note that in this work operators are generally designated with a hat as in $\hat{\mathscr{O}}$. The exceptions to this are creation and annihilation operators which are left plain for simplicity. To this point we have been implicitly following the Schrödinger picture, where the time dependence is associated with the states of the system rather than with the operators.

The exception is $\hat{N}_L$ in Equation 2.2, which in fact is in the Heisenberg picture. In this picture the time dependence associated with the operators of a system rather than with the states.

Moving between the Schrödinger and Heisenberg pictures is accomplished by utilizing a unitary time development operator $\hat{U}(t,t_o)$, also known as an evolution operator. In the Schrödinger picture, this operator acts on states at some reference time t_o, such that the resulting state is transformed into what it will become at some other time t. This same evolution operator is also used to switch operators between the two pictures. Beginning with the evolution operator,

$$\hat{U}(t,t_o) = e^{-\frac{i}{\hbar}(t-t_0)\hat{\mathscr{H}}_S}, \tag{2.22}$$

and selecting our reference time as $t_o = 0$, using

$$\hat{U}(t) = e^{-\frac{i}{\hbar}(t)\hat{\mathscr{H}}_S}, \tag{2.23}$$

state development in the Schrödinger picture becomes

$$|\Psi_S(t)\rangle = \hat{U}(t)\,|\Psi_S(0)\rangle\,. \tag{2.24}$$

Alternatively one can switch to the Heisenberg picture and move the time dependance to the operators:

$$\hat{\mathscr{O}}_H(t) = \hat{U}^\dagger(t)\,\hat{\mathscr{O}}_S\hat{U}(t)\,. \tag{2.25}$$

Using the time-independent Heisenberg state $|\Psi_H\rangle \equiv |\Psi_S(0)\rangle$, one can show that all matrix elements are the same in both pictures. The expression for the tunneling current includes the time derivative of the counting operator $\hat{N}_L$ in the Heisenberg picture:

$$\frac{d\hat{N}_{L,H}}{dt} = \frac{d}{dt}\left(\hat{U}^\dagger\hat{N}_{L,S}\hat{U}\right) \tag{2.26}$$

$$\frac{d\hat{N}_{L,H}}{dt} = \left(\frac{d\hat{U}^\dagger}{dt}\right)\hat{N}_{L,S}\hat{U} + \hat{U}^\dagger\left(\frac{d\hat{N}_{L,S}}{dt}\right)\hat{U} + \hat{U}^\dagger\hat{N}_{L,S}\left(\frac{d\hat{U}}{dt}\right). \tag{2.27}$$

Because $\hat{N}_{L,S}$ has no explicit dependance on t, the middle term vanishes. From equation 2.22,

$$\frac{d\hat{U}}{dt} = -\frac{i}{\hbar}\hat{\mathscr{H}}_S\hat{U}, \tag{2.28}$$

and so

$$\frac{d\hat{N}_{L,H}}{dt} = \left(\frac{i}{\hbar}\hat{\mathscr{H}}_S^{\dagger}\hat{U}^{\dagger}\right)\hat{N}_{L,S}\hat{U} - \hat{U}^{\dagger}\hat{N}_{L,S}\left(\frac{i}{\hbar}\hat{\mathscr{H}}_S\hat{U}\right). \tag{2.29}$$

Since the Hamiltonian commutes with itself and is Hermitian $\left(\hat{\mathscr{H}}^{\dagger} = \hat{\mathscr{H}}\right)$, we can rearrange terms

$$\frac{\hbar}{i}\left(\frac{d\hat{N}_{L,H}}{dt}\right) = \hat{\mathscr{H}}_S\underbrace{\hat{U}^{\dagger}\hat{N}_{L,S}\hat{U}}_{\hat{N}_{L,H}} - \underbrace{\hat{U}^{\dagger}\hat{N}_{L,S}\hat{U}^{\dagger}}_{\hat{N}_{L,H}}\hat{\mathscr{H}}_S \tag{2.30}$$

and through the use of the commutator defined as $\left[\hat{A},\hat{B}\right] = \hat{A}\hat{B} - \hat{B}\hat{A}$, we find

$$i\hbar\left(\frac{d\hat{N}_{L,H}}{dt}\right) = \left[\hat{N}_{L,H},\hat{\mathscr{H}}_S\right]. \tag{2.31}$$

By substitution into equation 2.2 we express the tunneling current in terms of a commutator

$$I(t) = \frac{ie}{\hbar}\left\langle\left[\hat{N}_{L,H},\hat{\mathscr{H}}_S\right]\right\rangle. \tag{2.32}$$

Concentrating on the commutator of this expression we will switch back to the Schrödinger picture and expand the Hamiltonian to separately consider the three terms in detail. The is done to obtain an expression for the tunneling current in terms of creation and annihilation operators. To simplify the expressions the S-subscript will be dropped from the Hamiltonians in the Schrödinger picture giving

$$\left[\hat{N}_{L,H},\hat{\mathscr{H}}\right] = e^{\frac{i}{\hbar}t\hat{\mathscr{H}}^{\dagger}}\left[\hat{N}_{L,S},\hat{\mathscr{H}}\right]e^{-\frac{i}{\hbar}t\hat{\mathscr{H}}} \tag{2.33}$$

$$\left[\hat{N}_{L,H},\hat{\mathscr{H}}\right] = e^{\frac{i}{\hbar}t\hat{\mathscr{H}}^{\dagger}}\left(\underbrace{\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right]}_{1} + \underbrace{\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right]}_{2} + \underbrace{\left[\hat{N}_{L,S},\hat{\mathscr{H}}_D\right]}_{3}\right)e^{-\frac{i}{\hbar}t\hat{\mathscr{H}}} \tag{2.34}$$

Substituting for the Schrödinger operators and using anti-commutation relations (see Appendix A),

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right] = \hat{N}_{L,S}\hat{\mathscr{H}}_L - \hat{\mathscr{H}}_L\hat{N}_{L,S} \tag{2.35}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right] = \sum_{\vec{k}\vec{k'}} c^{\dagger}_{\vec{k}}c_{\vec{k}}\varepsilon_{\vec{k'}}c^{\dagger}_{\vec{k'}}c_{\vec{k'}} - \varepsilon_{\vec{k'}}c^{\dagger}_{\vec{k'}}c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}} \tag{2.36}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right] = \sum_{\vec{k}\vec{k'}} \varepsilon_{\vec{k'}}\left(c^{\dagger}_{\vec{k}}c_{\vec{k}}c^{\dagger}_{\vec{k'}}c_{\vec{k'}} - c^{\dagger}_{\vec{k'}}c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right) \tag{2.37}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right] = \sum_{\vec{k}\vec{k'}} \varepsilon_{\vec{k'}}\left[c^{\dagger}_{\vec{k}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k'}}c_{\vec{k}}\right)c_{\vec{k'}} - c^{\dagger}_{\vec{k'}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k}}c_{\vec{k'}}\right)c_{\vec{k}}\right]. \tag{2.38}$$

This double sum presents two possible cases that are considered. The first case is $\vec{k} = \vec{k'}$ and results in

$$\sum_{\vec{k}=\vec{k'}} \varepsilon_{\vec{k}}\left[c^{\dagger}_{\vec{k}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}} - c^{\dagger}_{\vec{k}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}}\right] = \sum_{\vec{k}=\vec{k'}} \varepsilon_{\vec{k}}\left[c^{\dagger}_{\vec{k}}\left(1 - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}} - c^{\dagger}_{\vec{k}}\left(1 - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}}\right] \tag{2.39}$$

$$\sum_{\vec{k}=\vec{k'}} \varepsilon_{\vec{k}}\left[c^{\dagger}_{\vec{k}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}} - c^{\dagger}_{\vec{k}}\left(\delta_{kk'} - c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)c_{\vec{k}}\right] = 0. \tag{2.40}$$

The second of the two cases is $\vec{k} \neq \vec{k'}$ and results in

$$\sum_{\vec{k}\neq\vec{k'}} \varepsilon_{\vec{k'}}\left(-c^{\dagger}_{\vec{k}}c^{\dagger}_{\vec{k'}}c_{\vec{k}}c_{\vec{k'}} + c^{\dagger}_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k'}}c_{\vec{k}}\right) = \sum_{\vec{k}\neq\vec{k'}} \varepsilon_{\vec{k'}}\left(-(-1)^2 c^{\dagger}_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k'}}c_{\vec{k}} + c^{\dagger}_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k'}}c_{\vec{k}}\right) \tag{2.41}$$

$$\sum_{\vec{k}\neq\vec{k'}} \varepsilon_{\vec{k'}}\left(-c^{\dagger}_{\vec{k}}c^{\dagger}_{\vec{k'}}c_{\vec{k}}c_{\vec{k'}} + c^{\dagger}_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k'}}c_{\vec{k}}\right) = 0. \tag{2.42}$$

Hence $\left[\hat{N}_{L,S},\hat{\mathscr{H}}_L\right] = 0$. One interpretation of this result is that since both the counting operator and the Lead Hamiltonian conserve particle number, their order should not matter and therefore they commute. Similarly we can show $\left[\hat{N}_{L,S},\hat{\mathscr{H}}_D\right] = 0$.

Substituting for the operators in the second commutator results in an expression that describes the tunneling behavior of the electrons of the system.

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = \hat{N}_{L,T}\hat{\mathscr{H}}_T - \hat{\mathscr{H}}_T\hat{N}_{L,T} \tag{2.43}$$

$$\begin{aligned}\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = &\sum_{\vec{k}\vec{k'}} c^{\dagger}_{\vec{k}}c_{\vec{k}}\left(T_{\vec{k'}}c^{\dagger}_{\vec{k'}}\psi(\vec{r}) + T^{*}_{\vec{k'}}\ \psi(\vec{r})c_{\vec{k'}}\right) \\ &-\left(T_{\vec{k'}}c^{\dagger}_{\vec{k'}}\psi(\vec{r}) + T^{*}_{\vec{k'}}\ \psi(\vec{r})c_{\vec{k'}}\right)c^{\dagger}_{\vec{k}}c_{\vec{k}}\end{aligned} \tag{2.44}$$

$$\begin{aligned}\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = &\sum_{\vec{k}\vec{k'}} T_{\vec{k'}}\left(c^{\dagger}_{\vec{k}}c_{\vec{k}}c^{\dagger}_{\vec{k'}}\psi(\vec{r}) - c^{\dagger}_{\vec{k'}}\psi(\vec{r})c^{\dagger}_{\vec{k}}c_{\vec{k}}\right) \\ &+T^{*}_{\vec{k'}}\ \left(c^{\dagger}_{\vec{k}}c_{\vec{k}}\psi^{\dagger}(\vec{r})c_{\vec{k'}} - \psi^{\dagger}(\vec{r})c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)\end{aligned} \tag{2.45}$$

$$\begin{aligned}\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = &\sum_{\vec{k}\vec{k'}} T_{\vec{k'}}\psi(\vec{r})\left((-1)^3 c^{\dagger}_{\vec{k}}c_{\vec{k}}c^{\dagger}_{\vec{k'}} - (-1)^1 c^{\dagger}_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right) \\ &+T^{*}_{\vec{k'}}\ \psi^{\dagger}(\vec{r})\left((-1)^2 c^{\dagger}_{\vec{k}}c_{\vec{k}}c_{\vec{k'}} + c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)\end{aligned} \tag{2.46}$$

$$\begin{aligned}\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = &\sum_{\vec{k}\vec{k'}} T_{\vec{k'}}\psi(\vec{r})\left(-c^{\dagger}_{\vec{k}}c_{\vec{k}}c^{\dagger}_{\vec{k'}} + (-1)^1 c^{\dagger}_{\vec{k}}c^{\dagger}_{\vec{k'}}c_{\vec{k}}\right) \\ &+T^{*}_{\vec{k'}}\ \psi^{\dagger}(\vec{r})\left((-1)^1 c^{\dagger}_{\vec{k}}c_{\vec{k'}}c_{\vec{k}} - c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right)\end{aligned} \tag{2.47}$$

$$\begin{aligned}\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right] = &\sum_{\vec{k}\vec{k'}} T_{\vec{k'}}\psi(\vec{r})\left[-c^{\dagger}_{\vec{k}}c_{\vec{k}}c^{\dagger}_{\vec{k'}} - c^{\dagger}_{\vec{k}}\left(\delta_{kk'}\ - c_{\vec{k}}c^{\dagger}_{\vec{k'}}\right)\right] \\ &+T^{*}_{\vec{k'}}\ \psi^{\dagger}(\vec{r})\left[-\left(\delta_{kk'}\ - c_{\vec{k'}}c^{\dagger}_{\vec{k}}\right)c_{\vec{k}} - c_{\vec{k'}}c^{\dagger}_{\vec{k}}c_{\vec{k}}\right]\end{aligned} \tag{2.48}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right]=\sum_{\vec{k}\vec{k}'}T_{\vec{k}'}\psi(\vec{r})\left(-\delta_{kk'}\ c^{\dagger}_{\vec{k}}\right)+T^{*}_{\vec{k}'}\ \psi^{\dagger}(\vec{r})\left(-\delta_{kk'}\ c_{\vec{k}}\right) \tag{2.49}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right]=\sum_{\vec{k}}T_{\vec{k}}\ \psi(\vec{r})\left(-c^{\dagger}_{\vec{k}}\right)+T^{*}_{\vec{k}}\ \psi^{\dagger}(\vec{r})\left(-c_{\vec{k}}\right) \tag{2.50}$$

$$\left[\hat{N}_{L,S},\hat{\mathscr{H}}_T\right]=\sum_{\vec{k}}T_{\vec{k}}\ c^{\dagger}_{\vec{k}}\psi(\vec{r})-T^{*}_{\vec{k}}\ \psi^{\dagger}(\vec{r})c_{\vec{k}}. \tag{2.51}$$

Substituting into equation 2.31-2.34 gives the time derivative of the counting operator with the operators in the Heisenberg picture as

$$\frac{d\hat{N}_{L,H}}{dt}=\frac{i}{\hbar}\sum_{\vec{k}}\left(T^{*}_{\vec{k}}\ \psi^{\dagger}_{H}(\vec{r})c_{\vec{k},H}-T_{\vec{k}}\ c^{\dagger}_{\vec{k},H}\psi_{H}(\vec{r})\right). \tag{2.52}$$

2.3 Tunneling Perturbation

The next step is based on the assumption that the tunneling interaction is weak — that there is a large energy barrier for tunneling between the STM tip and system so that the tunneling amplitude is small. This permits us to include the tunneling only to first order in perturbation theory. This calculation is a standard piece of many-body physics, and here will only be summarized briefly. One begins by separating the Hamiltonian into two terms

$$\hat{\mathscr{H}}=\hat{\mathscr{H}}_O+\hat{\mathscr{H}}_T \tag{2.53}$$

where

$$\hat{\mathscr{H}}_O=\hat{\mathscr{H}}_L+\hat{\mathscr{H}}_D. \tag{2.54}$$

The calculation proceeds by turning on the perturbation $\hat{\mathscr{H}}_T$ in a slow, adiabatic way beginning at time $t=-\infty$, expanding the time evolution using a time-ordering operator, and keeping perturbation

terms up to first order. The result is:

$$\left\langle \frac{d\hat{N}_{L,H}}{dt} \right\rangle = \left\langle \frac{d\hat{N}_{L,I}}{dt} \right\rangle + \frac{1}{i\hbar} \left\langle \int_{-\infty}^{t} dt' \left[\frac{d\hat{N}_{L,I}(t)}{dt}, \hat{\mathscr{H}}_{T,I}(t') \right] \right\rangle + \dots \tag{2.55}$$

where now the statistical average is over only the unperturbed Hamiltonian $\hat{\mathscr{H}}_O$. Here the time dependence is in the interaction picture. For a general operator $\hat{O}$,

$$\hat{O}_I(t) = \hat{U}_0^{\dagger}(t)\hat{O}_S\hat{U}_0(t), \tag{2.56}$$

where

$$U_o(t,t_o) \equiv e^{\frac{i}{\hbar}(t-t_0)\hat{\mathscr{H}}_O}. \tag{2.57}$$

The first term on the right hand side of Equation 2.55 is the unperturbed, equilibrium part corresponding to system $\hat{\mathscr{H}}_O$ (independent dot and lead without tunneling). In equilibrium no net current flows and electrons tunnel in both directions equally; hence this term vanishes. The second term on the right hand side is the tunneling to first order in perturbation theory. When substituted into equation 2.32 this results in an integral representation of the tunneling current:

$$I(t) = \frac{ei}{\hbar} \int_{-\infty}^{t} dt' \left\langle \left[\frac{d\hat{N}_{L,I}(t)}{dt}, \hat{\mathscr{H}}_{T,I}(t') \right] \right\rangle. \tag{2.58}$$

2.4 Green's Functions

Our final step in this chapter is to formulate the tunneling current in terms of Green's or correlation functions. Such expressions allow us to rearrange the time dependency to allow for numerical analysis.

We begin with a closer look at how the time dependencies in our interaction picture. Considering a generalized operator in the interaction picture, the time dependance is:

$$\hat{O}_I(t) = e^{\frac{i}{\hbar}\left(\mathscr{H}_{D,S}+\mathscr{H}_{L,S}\right)t} \quad \hat{O}_S \quad e^{-\frac{i}{\hbar}\left(\mathscr{H}_{D,S}+\mathscr{H}_{L,S}\right)t}. \tag{2.59}$$

As mentioned in the previous section, a result of treating the tunneling term perturbatively is that the statistical average

$$\left\langle \hat{O} \right\rangle = \sum_{\alpha} \left\langle \alpha \left| \hat{\rho} \hat{O} \right| \alpha \right\rangle \tag{2.60}$$

is now based only on the unperturbed Hamiltonian $\hat{\mathscr{H}_O}$. Thus

$$\hat{\rho} = \frac{1}{Z} e^{-\beta\left(\mathscr{H}_{D,S} + \mathscr{H}_{L,S} - \mu_L N_{L,S} - \mu_D N_{D,S}\right)} \tag{2.61}$$

and it becomes useful to simplify the notation by defining the following

$$G \equiv \mathscr{H}_{D,S} + \mathscr{H}_{L,S} - \mu_L N_{D,S} - \mu_D N_{L,S}. \tag{2.62}$$

Therefore our generalized operator in the interaction picture becomes

$$\hat{O}_I(t) = e^{\frac{i}{\hbar}Gt} \left(e^{\frac{i}{\hbar}\left(\mu_L N_{D,S} + \mu_D N_{L,S}\right)} \quad \hat{O}_S \quad e^{-\frac{i}{\hbar}\left(\mu_L N_{D,S} + \mu_D N_{L,S}\right)} \right) e^{-\frac{i}{\hbar}Gt}. \tag{2.63}$$

Now we consider the combination of creation and annihilation operators in the commutator terms of the tunneling current expression Equation 2.58. These can be simplified by taking advantage of the anticommutation relations and using the generalized operator in our interaction picture as the time dependency model. Details are in Appendix B. The resulting simplified operator combinations are

$$\psi_I^{\dagger}(\vec{r},t)\, c_{\vec{k},I}(t) = e^{-ieVt} e^{\frac{i}{\hbar}Gt} \psi_I^{\dagger}(\vec{r})\, c_{\vec{k},I} e^{-\frac{i}{\hbar}Gt} \tag{2.64}$$

$$c_{\vec{k},I}^{\dagger}(t)\, \psi_I(\vec{r},t) = e^{ieVt} e^{\frac{i}{\hbar}Gt} c_{\vec{k},I}^{\dagger} \psi_I(\vec{r})\, e^{-\frac{i}{\hbar}Gt} \tag{2.65}$$

where

$$eV \equiv \mu_L - \mu_D. \tag{2.66}$$

It is useful to define auxiliary tunneling operators $\hat{A}(\vec{r},t)$ and $\hat{A}^{\dagger}(\vec{r},t)$:

$$\hat{A}(\vec{r},t) \equiv \sum_{\vec{k}} T_{\vec{k}}\, c^{\dagger}_{\vec{k},I}(t)\, \psi_I\left(\vec{r},t\right) \tag{2.67}$$

$$\hat{A}^{\dagger}(\vec{r},t) \equiv \sum_{\vec{k}} T^{*}_{\vec{k}}\ \psi^{\dagger}_I\left(\vec{r},t\right) c_{\vec{k},I}(t)\,. \tag{2.68}$$

What these operators do physically is move electrons from the dot to the lead $\left(\hat{A}(\vec{r},t)\right)$, and visa versa $\left(\hat{A}^{\dagger}(\vec{r},t)\right)$, with tunneling taking place at a time t. The time derivative of the counting operator and Hamiltonian that make up the tunneling current can then be written in terms of these correlation functions as

$$\frac{d\hat{N}_{L,I}(t)}{dt} = \frac{i}{\hbar}\sum_{\vec{k}}\left(e^{-ieVt}\hat{A}^{\dagger} - e^{ieVt}\hat{A}\right) \tag{2.69}$$

$$\hat{\mathscr{H}}_{T,I}\left(t'\right) = \sum_{\vec{k}}\left(e^{ieVt'}\hat{A} + e^{-ieVt'}\hat{A}^{\dagger}\right). \tag{2.70}$$

By substituting these operators and their conjugates (see Appendix C) into equation 2.58 we get the tunneling current in terms of these concise correlation functions

$$I(t) = \frac{e}{\hbar^2}\int_{-\infty}^{t} dt'\left(e^{\frac{i\left(t-t'\right)eV}{\hbar}}\left\langle\left[\hat{A}(\vec{r},t),\hat{A}^{\dagger}\left(\vec{r},t'\right)\right]\right\rangle - e^{\frac{-i\left(t-t'\right)eV}{\hbar}}\left\langle\left[\hat{A}^{\dagger}(\vec{r},t),\hat{A}\left(\vec{r},t'\right)\right]\right\rangle\right). \tag{2.71}$$

We can equivalently introduce a step function $\theta\left(t-t'\right)$ into the integrand and take the upper limit to infinity:

$$I(\vec{r},t) = \frac{e}{\hbar^2}\int_{-\infty}^{\infty} dt'\,\theta\left(t-t'\right)\left(e^{\frac{i\left(t-t'\right)eV}{\hbar}}\left\langle\left[\hat{A}(\vec{r},t),\hat{A}^{\dagger}\left(\vec{r},t'\right)\right]\right\rangle - e^{\frac{-i\left(t-t'\right)eV}{\hbar}}\left\langle\left[\hat{A}^{\dagger}(\vec{r},t),\hat{A}\left(\vec{r},t'\right)\right]\right\rangle\right). \tag{2.72}$$

This is the Green's function form of the current. Here we have made explicit the dependence on the location $\vec{r}$ of tunneling.

CHAPTER 3
CURRENT EXPRESSION

The final analytic step that is required is to transform the formal expression for tunneling current into a form that allows for numerical analysis. To accomplish this we first recognize that the integral in equation 2.72 has a form similar to the corresponding expression for superconductor tunneling which already has a well established method of solution in the literature. [21] This provides a starting point and analytic method for us to follow. In particular, the integrand is a retarded Green's function and can be transformed into frequency space to determine the spectral density function, simplify the time dependencies of the Green's function, and carry out the thermal averaging. The final form of the current expression is a Lehmann representation. [21] It is this representation will allow for numerical solution.

3.1 Retarded Green's Functions

Recognizing the integrand of our analytic expression contains a retarded Green's function we define

$$iX_{ret}\left(t-t'\right) \equiv \theta\left(t-t'\right)\left\langle\left[\hat{A}\left(t\right),\hat{A}^{\dagger}\left(t'\right)\right]\right\rangle \tag{3.1}$$

where the $\vec{r}$ notation is dropped from the correlation functions in equation 2.68 for clarity. In order to rewrite our current expression in terms of this retarded function we first determine the complex conjugate

$$X^{*}_{ret}\left(t-t'\right) = i\theta\left(t-t'\right)\left\langle\hat{A}\left(t\right)\hat{A}^{\dagger}\left(t'\right)-\hat{A}^{\dagger}\left(t'\right)\hat{A}\left(t\right)\right\rangle^{*} \tag{3.2}$$

$$X^{*}_{ret}\left(t-t'\right) = i\theta\left(t-t'\right)\left\langle\left(\hat{A}\left(t\right)\hat{A}^{\dagger}\left(t'\right)-\hat{A}^{\dagger}\left(t'\right)\hat{A}\left(t\right)\right)^{\dagger}\right\rangle \tag{3.3}$$

$$X^{*}_{ret}\left(t-t'\right) = i\theta\left(t-t'\right)\left\langle\hat{A}\left(t'\right)\hat{A}^{\dagger}\left(t\right)-\hat{A}^{\dagger}\left(t\right)\hat{A}\left(t'\right)\right\rangle \tag{3.4}$$

$$X^{*}_{ret}\left(t-t'\right) = -i\theta\left(t-t'\right)\left\langle\hat{A}^{\dagger}\left(t\right)\hat{A}\left(t'\right)-\hat{A}\left(t'\right)\hat{A}^{\dagger}\left(t\right)\right\rangle \tag{3.5}$$

$$iX^{*}_{ret}\left(t-t'\right) = \theta\left(t-t'\right)\left\langle\left[\hat{A}^{\dagger}\left(t\right),\hat{A}\left(t'\right)\right]\right\rangle \tag{3.6}$$

and substitute these into equation 2.72 to get

$$I(t) = \frac{e}{\hbar^2}\int_{-\infty}^{\infty} dt' \left(e^{\frac{i(t-t')eV}{\hbar}} iX_{ret}\left(t-t'\right) - e^{\frac{-i(t-t')eV}{\hbar}} iX^*_{ret}\left(t-t'\right)\right) \tag{3.7}$$

$$I(t) = \frac{ie}{\hbar^2}\int_{-\infty}^{\infty} dt' \left(e^{\frac{i(t-t')eV}{\hbar}} X_{ret}\left(t-t'\right) - e^{\frac{-i(t-t')eV}{\hbar}} X^*_{ret}\left(t-t'\right)\right). \tag{3.8}$$

Since the current must be real, this can be written as the sum of one term plus its complex conjugate:

$$I(t) = \int_{-\infty}^{\infty} dt' \left(\left(\frac{ie}{\hbar^2}\right) e^{\frac{i(t-t')eV}{\hbar}} X_{ret}\left(t-t'\right) + \left(-\frac{ie}{\hbar^2}\right) e^{\frac{-i(t-t')eV}{\hbar}} X^*_{ret}\left(t-t'\right)\right). \tag{3.9}$$

3.2 Fourier Transformation

Moving to frequency space using a Fourier transform we define:

$$\widetilde{F}(\omega) \equiv \int_{-\infty}^{\infty} dt'' e^{i\omega t''} \hat{F}\left(t''\right). \tag{3.10}$$

The transform of the retarded Green's function and its complex conjugate then turns out to have $\omega = \frac{eV}{\hbar}$:

$$\widetilde{X}_{ret}\left(\frac{eV}{\hbar}\right) = \int_{-\infty}^{\infty} dt' e^{\frac{i(t-t')eV}{\hbar}} X_{ret}\left(t-t'\right) \tag{3.11}$$

$$\widetilde{X}^*_{ret}\left(\frac{eV}{\hbar}\right) = \int_{-\infty}^{\infty} dt' e^{\frac{-i(t-t')eV}{\hbar}} X^*_{ret}\left(t-t'\right). \tag{3.12}$$

Substituting these into equation 3.9 we get the spectral density function

$$I(t) = \left(\frac{ie}{\hbar^2}\right)\widetilde{X}_{ret} + \left(-\frac{ie}{\hbar^2}\right)\widetilde{X}^*_{ret} \tag{3.13}$$

$$I(t) = \frac{ie}{\hbar^2}\left(\widetilde{X}_{ret} - \widetilde{X}^*_{ret}\right) \tag{3.14}$$

$$I(t) = \frac{ie}{\hbar^2}\left(2i\ Im\left(\widetilde{X}_{ret}\right)\right) \tag{3.15}$$

$$I(t) = -2\,Im\left(\frac{e}{\hbar^2}\,\widetilde{X}_{ret}\right). \tag{3.16}$$

Our final current expression will come directly from this function, however the thermal averaging must be performed before we can take the imaginary part of the retarded Green's function.

3.3 Statistical Averaging

To perform the thermal average we start with our Green's function definition in the time domain from equation 3.1 and write out the full expression in terms of the creation and annihilation operators:

$$X_{ret}\left(t-t'\right)=-i\theta\left(t-t'\right)\left\langle\left[\sum_{\vec{k}}T_{\vec{k}}\,c^{\dagger}_{\vec{k},I}\left(t\right)\psi_{I}\left(\vec{r},\vec{t}\right),\sum_{\vec{k'}}T^{*}_{\vec{k'}}\;\psi^{\dagger}_{I}\left(\vec{r},\vec{t'}\right)c_{\vec{k'},I}\left(t'\right)\right]\right\rangle. \tag{3.17}$$

Collecting the summations and grouping terms we have

$$X_{ret}\left(t-t'\right)=-i\theta\left(t-t'\right)\sum_{\vec{k},\vec{k'}}T_{\vec{k}}\,T^{*}_{\vec{k'}}\;\left\langle\left[c^{\dagger}_{\vec{k},I}\left(t\right)\psi_{I}\left(\vec{r},\vec{t}\right),\psi^{\dagger}_{I}\left(\vec{r},\vec{t'}\right)c_{\vec{k'},I}\left(t'\right)\right]\right\rangle. \tag{3.18}$$

Then by pushing the exponential time dependencies of the creation operator and its adjoint to the left of the operator as demonstrated in Appendix B and selecting a reference time of $t'=0$

$$X_{ret}\left(t\right)=-i\theta\left(t\right)\sum_{\vec{k},\vec{k'}}T_{\vec{k}}\,T^{*}_{\vec{k'}}\;e^{-\frac{i}{\hbar}\left(\varepsilon_{\vec{k}}-\mu_{L}\right)t}\left\langle\left[c^{\dagger}_{\vec{k},I}\psi_{I}\left(\vec{r},\vec{t}\right),\psi^{\dagger}_{I}\left(\vec{r},\vec{0}\right)c_{\vec{k'},I}\right]\right\rangle \tag{3.19}$$

we can remove the exponents from the commutator and simplify the averaging. Expanding terms and utilizing the anti-commutation relations found in Appendix A to pass operators by each other results in

$$X_{ret}\left(t\right)=-i\theta\left(t\right)\sum_{\vec{k},\vec{k'}}T_{\vec{k}}\,T^{*}_{\vec{k'}}\;e^{-\frac{i}{\hbar}\left(\varepsilon_{\vec{k}}-\mu_{L}\right)t}\left\langle\left[c^{\dagger}_{\vec{k},I}c_{\vec{k'},I}\psi_{I}\left(\vec{r},\vec{t}\right)\psi^{\dagger}_{I}\left(\vec{r},\vec{0}\right)-c_{\vec{k'},I}c^{\dagger}_{\vec{k},I}\psi^{\dagger}_{I}\left(\vec{r},\vec{0}\right)\psi_{I}\left(\vec{r},\vec{t}\right)\right]\right\rangle. \tag{3.20}$$

Starting with the previous definition of our partition function in equation 2.60 and noting that we are averaging over the lead and dot we rewrite the partition function to include separate terms

for the lead and dot

$$\left\langle \hat{O} \right\rangle = \sum_{L,D} \frac{1}{Z_L Z_D} \left\langle L,D \left| e^{\beta\left(\hat{K}_L + \hat{K}_D\right)} \hat{\rho}\hat{O} \right| L,D \right\rangle \tag{3.21}$$

where we define the terms in the exponent as

$$\hat{K}_L \equiv \hat{\mathscr{H}}_{L,S} - \mu_L \hat{N}_{L,S} \tag{3.22}$$

$$\hat{K}_D \equiv \hat{\mathscr{H}}_{D,S} - \mu_D \hat{N}_{D,S}. \tag{3.23}$$

The partition functions here are separated into one for the lead Z_L, and another for the dot Z_L, which can be expressed as a trace as follows

$$Z_L = Tr\left(e^{-\beta K_L}\right) \tag{3.24}$$

$$Z_D = Tr\left(e^{-\beta K_D}\right). \tag{3.25}$$

When separating these two parts it is also necessary to separate the multi-particle eigenstates $\mid \alpha \rangle$. Since the lead and dot are assumed independent we can say

$$\mid \alpha \rangle = \mid LD \rangle \tag{3.26}$$

$$\mid \alpha \rangle = \mid L \rangle \bigotimes \mid D \rangle \tag{3.27}$$

where the lead eigenstates $\mid L \rangle$, and dot eigenstates $\mid D \rangle$, commute with the operators from the opposite system. Recalling that $\left[\hat{K}_L, \hat{K}_D\right] = 0$ such that $e^{\hat{K}_L + \hat{K}_D} = e^{\hat{K}_L} e^{\hat{K}_D}$, we can say our thermal averaging takes the general form

$$\langle \hat{O} \rangle = \sum_{LD} \frac{1}{Z_L Z_D} \langle LD | e^{-\beta\left(\hat{K}_L + \hat{K}_D\right)} \hat{O} | LD \rangle \tag{3.28}$$

$$\langle \hat{O} \rangle = \sum_{L} \frac{1}{Z_L} \langle L | e^{-\beta\left(\hat{K}_L\right)} \hat{O} | L \rangle \bigotimes \sum_{D} \frac{1}{Z_D} \langle D | e^{-\beta\left(\hat{K}_D\right)} \hat{O} | D \rangle. \tag{3.29}$$

Substituting these expressions into the retarded potential we have

$$X_{ret}(t) = -i\theta(t)\sum_{\vec{k},\vec{k}'} T_{\vec{k}}\, T^{*}_{\vec{k}'}\; e^{-\frac{i}{\hbar}(\varepsilon_{\vec{k}}-\mu_L)t} \sum_{L,D} \frac{1}{Z_L Z_D} \tag{3.30}$$
$$\left\langle L,D \left| e^{-\beta\left(\hat{K}_L+\hat{K}_D\right)} \left(c^{\dagger}_{\vec{k},I} c_{\vec{k}',I} \psi_I\left(\vec{r},t\right) \psi^{\dagger}_I\left(\vec{r},\vec{0}\right) - c_{\vec{k}',I} c^{\dagger}_{\vec{k},I} \psi^{\dagger}_I\left(\vec{r},\vec{0}\right) \psi_I\left(\vec{r},t\right) \right) \right| L,D \right\rangle$$

and grouping the terms into operators related to the lead and dot

$$X_{ret}(t) = -i\theta(t)\sum_{\vec{k},\vec{k}'} T_{\vec{k}}\, T^{*}_{\vec{k}'}\; e^{-\frac{i}{\hbar}(\varepsilon_{\vec{k}}-\mu_L)t} \tag{3.31}$$
$$\left\{ \frac{1}{Z_L}\sum_{L} \left\langle L \left| e^{-\beta\hat{K}_L} c^{\dagger}_{\vec{k},I} c_{\vec{k}',I} \right| L \right\rangle \frac{1}{Z_D}\sum_{D} \left\langle D \left| e^{-\beta\hat{K}_D} \psi_I(\vec{r},t)\, \psi^{\dagger}_I(\vec{r},0) \right| D \right\rangle \right.$$
$$\left. \frac{1}{Z_L}\sum_{L} \left\langle L \left| e^{-\beta\hat{K}_L} c_{\vec{k}',I} c^{\dagger}_{\vec{k},I} \right| L \right\rangle \frac{1}{Z_D}\sum_{D} \left\langle D \left| e^{-\beta\hat{K}_D} \psi^{\dagger}_I(\vec{r},0)\, \psi_I(\vec{r},t) \right| D \right\rangle \right\}$$

Simplifying this notation for clarity the following notational declaration is made

$$\left\langle c^{\dagger}_{\vec{k}} c_{\vec{k}'} \right\rangle_L \equiv \frac{1}{Z_L}\sum_{L} \left\langle L \left| e^{-\beta\hat{K}_L} c^{\dagger}_{\vec{k},I} c_{\vec{k}',I} \right| L \right\rangle. \tag{3.32}$$

From Shankar equation 21.3.36 [25] and our anti-commutation relations in Appendix A, we can say

$$\left\langle c^{\dagger}_{\vec{k}} c_{\vec{k}'} \right\rangle_L = \delta_{\vec{k},\vec{k}'} \left(\frac{1}{e^{\beta(\varepsilon_{\vec{k}}-\mu_L)}+1} \right). \tag{3.33}$$

Defining the usual Fermi-Dirac function [26]

$$f\left(\varepsilon_{\vec{k}}\right) \equiv \frac{1}{e^{\beta(\varepsilon_{\vec{k}}-\mu_L)}+1} \tag{3.34}$$

enables us to express our lead operators as

$$\left\langle c^{\dagger}_{\vec{k},I} c_{\vec{k'},I} \right\rangle_L = \delta_{\vec{k},\vec{k'}} f\left(\varepsilon_{\vec{k}}\right) \tag{3.35}$$

$$\tag{3.36}$$

and

$$\left\langle c_{\vec{k},I} c^{\dagger}_{\vec{k'},I} \right\rangle_L = \delta_{\vec{k},\vec{k'}} \left(1 - f\left(\varepsilon_{\vec{k}}\right)\right). \tag{3.37}$$

Here $\vec{k} = \vec{k'}$ allows us to rewrite equation 3.31 as

$$X_{ret}(t) = -i\theta(t) \sum_{\vec{k}} \left| T_{\vec{k}} \right|^2 e^{-\frac{i}{\hbar}(\varepsilon_{\vec{k}} - \mu_L)t} \left\{ f\left(\varepsilon_{\vec{k}}\right) \left\langle \psi_I(\vec{r},t)\, \psi_I^{\dagger}(\vec{r},0) \right\rangle_D - \left(1 - f\left(\varepsilon_{\vec{k}}\right)\right) \left\langle \psi_I^{\dagger}(\vec{r},0)\, \psi_I(\vec{r},t) \right\rangle_D \right\}. \tag{3.38}$$

This final expression contains the 'Lehmann representation' of the dot operators as described in the literature. [21] This particular representation allows for the numerical analysis conducted and is used extensively throughout all subsequent calculations.

3.4 Analytic Current Expression

We need to select a method to enumerate the eigenstates of the dot. This provides the necessary amount of information to distinguish each unique state while at the same time provide information that will make our analysis task as easy as possible. To do this we will choose two quantum numbers to label the states. The first is the numbers of particle in the state 'N'. The other number, 'n', is a numerically increasing integer label starting at one that enumerates all the states with a given N. In this way we label all states.

Using this enumerating scheme and considering that we will be operating on these states with creation and annihilation operators it is also useful to define labels to represent state with one greater

and one fewer particle than that of a given state enumerated by N and n. States that have one more particle as a result of creation operator $N' \equiv N+1$ will be referred to as

$$|N', n'\rangle = |N+1, n'\rangle, \tag{3.39}$$

while states that have a particle annihilated $N'' \equiv N-1$ will be refereed to as

$$|N'', n''\rangle = |N-1, n''\rangle. \tag{3.40}$$

We now can right the Lehmann representation of the dot operators using our labeling convention. To provide better clarity the interaction 'picture' subscript and the dot operator's functional dependencies will be dropped.

$$\left\langle \psi\psi^{\dagger} \right\rangle_D = \frac{1}{Z_D} \sum_{N,n,n'} e^{-\beta\left(\varepsilon_{N,n}-\mu_D N\right)} e^{i\,t\left(E_{N,n}-E_{N+1,n'}+\mu_D\right)} \left|\left\langle N,n\left|\psi\right|N+1,n'\right\rangle\right|^2 \tag{3.41}$$

$$\left\langle \psi^{\dagger}\psi \right\rangle_D = \frac{1}{Z_D} \sum_{N,n,n''} e^{-\beta\left(\varepsilon_{N,n}-\mu_D N\right)} e^{i\,t\left(E_{N-1,n''}-E_{N,n}+\mu_D\right)} \left|\left\langle N-1,n''\left|\psi\right|N,n\right\rangle\right|^2. \tag{3.42}$$

Taking the Fourier transform as defined earlier of equation 3.41 and substituting into the imaginary part of our current expression we obtain

$$Im\left[\widetilde{X}_{ret}\left(\frac{eV}{\hbar}\right)\right] = -\pi \sum_{N=0}^{\infty} \sum_{n=1}^{\infty} \frac{1}{Z_D} e^{-\beta\left(E_{N,n}-\mu_D N\right)} \sum_{n'=1}^{\infty} \left|\left\langle N,n\left|\psi\right|N+1,n'\right\rangle\right|^2 \underbrace{\sum_{\vec{k}} \left|T_{\vec{k}}\right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}}+\varepsilon_{N,n}-\varepsilon_{N+1,n'}}{\hbar}\right)}_{integral} \tag{3.43}$$

where a similar expression is obtained from equation 3.42. Now turning our attention to how to model the lead we begin by considering the last summation in the above expression labeled 'integral'. We will model this as a 2-D lead of noninteracting electrons with a parabolic spectrum.

We start by converting the summation into an integral as follows

$$\sum_{\vec{k}} \left| T_{\vec{k}} \right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right) = \frac{a}{(2\pi)^2} \int_{-\infty}^{\infty} d^2k' \left| T_{\vec{k}} \right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right) \tag{3.44}$$

We then assume that the weighting constant T is a smooth relatively flat function of $\vec{k}$ and therefore can be removed from the integral. Further we assume the electron energy $\varepsilon_{\vec{k}}$ is comprised of a kinetic component and a conduction band energy offset ε_{cond}

$$\varepsilon_{\vec{k}} = \frac{\hbar^2 k^2}{2m'} + \varepsilon_{cond} \tag{3.45}$$

where m' is the effective mass of the lead electron which produces

$$\sum_{\vec{k}} \left| T_{\vec{k}} \right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right) = \tag{3.46}$$
$$\frac{a}{(2\pi)^2} |T|^2 \int_{-\infty}^{\infty} dk f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\frac{\hbar^2 k^2}{2m'} + \varepsilon_{cond} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right).$$

Assuming differential elements Δk and $\Delta\phi$ that when integrated will produce a symmetric bowl with area 'a' we get

$$\sum_{\vec{k}} \left| T_{\vec{k}} \right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right) = \tag{3.47}$$
$$\frac{a}{(2\pi)^2} |T|^2 \int_0^{2\pi} d\phi \int_0^{\infty} k\, dk\, f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\frac{\hbar^2 k^2}{2m'} + \varepsilon_{cond} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right).$$

We then determine dk

$$dk = \left(\sqrt{\frac{m'}{2\hbar^2\left(\varepsilon_{\vec{k}} - \varepsilon_{cond}\right)}}\right) d\varepsilon_{\vec{k}}. \tag{3.48}$$

By making an assumption that energy $\varepsilon > 0$, we introduce step function to maintain a physically

possible system which results in

$$\sum_{\vec{k}} \left| T_{\vec{k}} \right|^2 f\left(\varepsilon_{\vec{k}}\right) \delta\left(\frac{\varepsilon_{\vec{k}} + \varepsilon_{N,n} - \varepsilon_{N+1,n'}}{\hbar}\right) = \frac{am'}{2\pi\hbar} |T|^2 f\left(\varepsilon_{N+1,n'} - \varepsilon_{N,n}\right) \theta\left(\varepsilon_{N+1,n'} - \varepsilon_{N,n}\right). \tag{3.49}$$

Substituting this back into equation the tunneling current equations 3.43 and we collect terms find the expression for the tunneling current

$$I(r,\mu_D,\mu_L) = \frac{e|T|^2 am'}{z_D \hbar^3} \sum_{N=0}^{\infty} \sum_{n=1}^{\infty} e^{-\beta\left(E_{N,n} - \mu_D N\right)} \sum_{m=0}^{\infty} \phi_m^2(r) \left[\sum_{n'=1}^{\infty} \theta\left(E_{N+1,n'} - E_{N,n}\right) f\left(E_{N+1,n'} - E_{N,n}\right) \left\langle N+1, n' | c_m^\dagger | N, n \right\rangle^2 - \sum_{n''=1}^{\infty} \theta\left(E_{N,n} - E_{N-1,n''}\right) \left(1 - f\left(E_{N,n} - E_{N-1,n''}\right)\right) \left\langle N, n | c_m^\dagger | N-1, n'' \right\rangle^2 \right]. \tag{3.50}$$

CHAPTER 4
NUMERICAL ANALYSIS

This chapter describes how the analytic expression for the tunneling current was utilized to perform numerical analysis and produce results. Details of the numerical analysis are in the computer codes in the appendices, in particular in their very extensive comments. Here we outline the structure and main steps of the calculation. We begin by describing the necessary data structures, for example how the quantum mechanical many-body basis states are represented in the computer. Specifically addressed is the use of a truncated basis to replace the infinite summations in the analytic expression. The current expression is then rearranged to allow for better adaptation to a computational algorithm and avoid difficulties of the very large and small numbers that appear in statistical mechanics calculations.

After establishing the framework of data structures, the significant steps of the calculation are then described. Since this computation is a finite calculation of an inherently infinite system the limiting factors are time and computer resources. Strategies are presented that were used to decrease the computational time and increase the size of the system that could be investigated.

4.1 Representation of States in the Dot

We begin by describing how the many-body basis states in the dot (the MPS described in Chapter 2) are represented in the computer. In first quantization, the MPS are Slater determinants of single particle states labeled by m. We consider only spin-polarized electrons, so that each can occur no more than once. Each MPS can be completely specified by listing the single-particle states from which it is constructed. Equivalently, and more convenient for our purposes, one can list the entire set of single particle states and indicate whether each is included in a given MPS or not. The presence of a given single-particle state is represented in the computer code as a '1' (set bit) while its absence is represented by a '0' (clear bit). The binary nature of this representation was exploited throughout the computation. In particular the efficiency of binary computer algorithms

and operations were utilized whenever possible when dealing with these binary state representations.

In order to keep track of the single particle states and MPS a common labeling or enumerating strategy was used throughout computation. The angular momentum, m, of a given single particle state is used to label the state. This starts at $m = 0$ and increase to the maximum angular momentum permitted in the truncated basis. For a given MPS, a label is constructed by arranging the occupied states (set bits) and unoccupied states (clear bits) in ascending order of m. That is, each single-particle state is associated with a bit of a binary number starting from the least significant to most significant bit. In this way the occupancy of the various single-particle states is used to create a number in base two that both describes an MPS state and becomes its label. For example, consider a truncated single-particle basis with $m = 0, 1, 2, 3$. All possible MPS can be labeled by four-bit integers. For instance, the particular two-electron MPS basis state made up of single-particle states $m = 0$, $m = 1$, would be given the MPS binary label 0011. Throughout the calculation this binary label, or when convenient its equivalent decimal name, is used to uniquely and completely describe the MPS. The technical reason for this strategy is that the bit manipulation routines within the computer are computationally efficient.

Because the dot Hamiltonian is symmetric under rotations about the z-axis, the z component of total angular momentum is conserved. Thus the Hamiltonian is block diagonal with blocks labeled by particle number N and total angular momentum mtot. Likewise each many-body energy eigenstate is also an eigenstate of total angular momentum, and can be labeled by mtot as well as N. We order and organize the MPS by grouping them first by N and then by mtot. Within these groups the MPS are arranged by their base two labels in ascending order and stored in an array including the bases for all N and mtot. Constructed in this fashion, the array indices that enumerate the states also contain a piece of information for each state. In this way more information could be stored in less computer memory which was one of the primary computational limitations. The penalty for using this strategy is that the indexing and algorithms are very complex.

4.2 Truncation of Basis States

The analytic expression for the tunneling current contains infinite sums that can be traced back to the infinite dimension of the single-particle Hilbert space. A key step in this calculation, and the most significant approximation of the model, is therefore to truncate the single-particle basis set to a finite size. The approach is motivated by the low-temperature, high-magnetic-field physics of the quantum dot. To begin with, because of the large energy gap between Landau levels, we restrict the single-particle states to the lowest Landau level. With sufficiently few electrons, the confining potential then ensures that electrons in the low energy many-particle states are confined towards the center of the dot. As a result there is little or no contribution from single-particle states above some threshold m. It therefore is a good approximation for low energy states to exclude single-particle states above this threshold from the calculation—that is, truncate the basis. From another point of view, truncating the basis is equivalent to placing a very steep wall at the system's outermost edge. As long as the low energy many body states have little weight in single particle states nearing the wall, the presence of the wall has no effect. In the code we choose a parameter SPS to represent the number of single particle basis states, and include single particle states with $m = 0, 1, 2, \ldots, SPS - 1$.

A number of terms in the analytic expression take maximum values based on the choice of SPS. Wherever possible these were identified and exploited in the code. This computational strategy allowed us to investigate larger systems that would not have otherwise have been possible or would have taken extremely long run times even for relatively small systems. For example, given SPS, the total number of N-electron MPS basis states is $\binom{SPS}{N}$. But this is not important. Instead, because of the block diagonal nature of the Hamiltonian, the important related quantity is the number of MPS states for a given mtot (and given N and SPS). This quantity can be worked out numerically, and is represented in the code by a variable mtot_state. This is important because it specifies the number of states that need to be investigated for a given angular momentum. Sums over particle number go up to an Nmax.

The eigenstates were calculated numerically using the computer code found in the appendices.

A truncated basis comprised of a complete set of multi-particle basis states was used to construct a Hamiltonian in matrix form. The Hamiltonian contained the kinetic and confining potential terms as well as the interaction energies experienced by the particles. This matrix was then diagonalized to determine the eigenvalues and associated eigenvectors. These results were used to numerically determine the tunneling current in subsequent calculations.

These parameters are used used to limit the infinite summation limits of the current expression which is rewritten as:

$$I(r,\mu_D,\mu_L) = \frac{e|T|^2 am'}{Z_D\hbar^3} \sum_{N=1}^{N_{max}} \sum_{n=1}^{mtot\_state} \sum_{m=0}^{SPS-1} \sum_{n'=1}^{mtot\_state} \sum_{n''=1}^{mtot\_state} \phi_m^2(r) \tag{4.1}$$
$$\left[e^{-\beta\left(E_{N,n}-\mu_D N\right)} \theta\left(E_{N+1,n'}-E_{N,n}\right) f\left(E_{N+1,n'}-E_{N,n}\right) \left\langle N+1,n'|c_m^\dagger|N,n\right\rangle^2 \right.$$
$$\left. -e^{-\beta\left(E_{N,n}-\mu_D N\right)} \theta\left(E_{N,n}-E_{N-1,n''}\right) \left(1-f\left(E_{N,n}-E_{N-1,n''}\right)\right) \left\langle N,n|c_m^\dagger|N-1,n''\right\rangle^2 \right].$$

One issue with the truncated basis is that it introduced error into the otherwise infinite system. However, by choosing sufficiently large limits for the number of states and appropriate confining potential, the error was undetectable when compared to the computer's round-off error.

4.3 Computational Current Expression

One of the computational issues to deal with is the size limitation of numbers that can be efficiently represented by a computer. This limitation can lead to floating point overflow and underflow and in our case make the calculation impossible for non-trivial systems. The problem stems from the size of the partition function, Z_D, on the outside of the summations. It is summed over all the allowable state combinations and over all particle numbers N which results in very large numbers. In addition, at low temperatures the thermal factors can become unmanageably large. There is a simple but important workaround. As will be explained in more detail in the next chapter, the chemical potential in the dot is chosen so that the calculation is dominated by tunneling to and from a dot with a chosen targeted number of electrons, N_{target}. This system has a lowest-energy many

particle eigenstate with energy $E^{GS}_{N_{target}}$. We modify the term $\exp(-\beta(E_{Nn}-\mu_D N))$ in the above sum by multiplying it by $\exp(+\beta(E^{GS}_{N_{target}}-\mu_D N_{target}))$ with a corresponding correction to the partition function. The leading term in the calculation of the partition function becomes 1 and all numbers become computationally manageable.

4.4 Calculation Steps

The first step in the calculation is to establish the array of MPS basis states. All of the basis states for all possible SPS, N and mtot are determined at once, as the summations require that all of these would eventually be needed. As a result it is much more efficient to calculate them once and store them for all future calculations. The MPS labels are stored in an array such that the indices of the array along with the base two label of the basis state would provide all the required information necessary for the current calculation.

Once the MPS basis is established, the energy eigenstates are calculated. This is done for each pair (N, mtot) by calculating the relevant block of the Hamiltonian. This produces a square array with mtot_state columns and rows. The Hamiltonian is a real, symmetric matrix and therefore only half of its elements need to be calculated which helps reduce the computation time and memory requirements. The Hamiltonian's diagonal terms depend on the confinement potential, W. We choose a simple model, a $W(r)$ which vanishes for $r < r_e$ and increases linearly from zero for $r > r_e$. This depends on two parameters, the radial position r_e and the slope chosen. The electron-electron interactions are computed using pseudopotential parameters V_m, as described in Chapter 2. These can easily be incorporated for any model interaction, including the screened or unscreened Coulomb interaction. Here we choose a simple approximation that has been shown to capture and indeed highlight FQHE physics at $\nu = 1/3$. This is the 'V1' model which accounts for the interaction with the electron's nearest neighbors. For this model, only V_1 is nonzero. After all the Hamiltonian terms are calculated it is diagonalized using a standard package to determine the eigenvalues and eigenvectors. The eigenvectors are represented by a 1D list—the coefficients of each MPS basis state for the given energy eigenstate. The eigenvectors and eigenenergies are stored in an external

file for later use.

The eigenstate energies are used to select the appropriate chemical potential μ_D to single out a desired target number of electrons N_{target} in the dot. The energies are also used in the Fermi-Dirac term. A final program takes all of the stored data and for a particular targeted number of particles in the QD will perform all of the summations to give the associated tunneling current.

CHAPTER 5
RESULTS

We have carried out the program described in the previous chapters to calculate the tunneling current for many combinations of system parameters: particle numbers, confining potentials, magnetic fields, temperatures, and positions of the STM tip with respect to the dot. In this chapter we highlight examples of these calculations to explain the resulting phenomena. We first present examples of energy eigenstate calculations, and then explain how they are used to choose a targeted number of electrons in the dot by tuning the dot's chemical potential. We then present representative plots of tunneling current as a function of tip position, and explain some of their features. We end with conclusions and possible directions for future work.

5.1 Many-Particle Energy Eigenstates

The many-particle energy eigenstates depend on a number of system parameters including the number of electrons in the dot, position and shape of the dot's edge, temperature and magnetic field strength. Calculating the tunneling current when any parameter changes requires first solving for all of the energy eigenstates for the given parameters. A typical set of eigenvalues is shown in figure 5.1. This figure displays some of the lower energy eigenvalues for a system with five electrons, in a truncated system of fifteen SPS, temperature of 1 Kelvin and magnetic field strength of 10 Tesla.

Figure 5.1 captures an important point mentioned earlier: the role of total angular momentum. The energies are plotted against mtot. Each column shows some of the lowest many-body energy eigenvalues calculated from the Hamiltonian block at the given mtot, starting with the lowest energy and ascending. The ground state can typically be seen easily when viewed in this way as there is typically a relatively large jump in energy between the ground state and the next eignenvalue. This is the energy gap that is at the heart of FQHE physics. In fact, the ground state in the plot is the $\nu = 1/3$ closely approximated by Laughlin's famous wave function.

These energy eigenvalues and in particular the ground state energy are next used to determine the chemical potential of the dot that will result in a particular number of electrons in the dot.

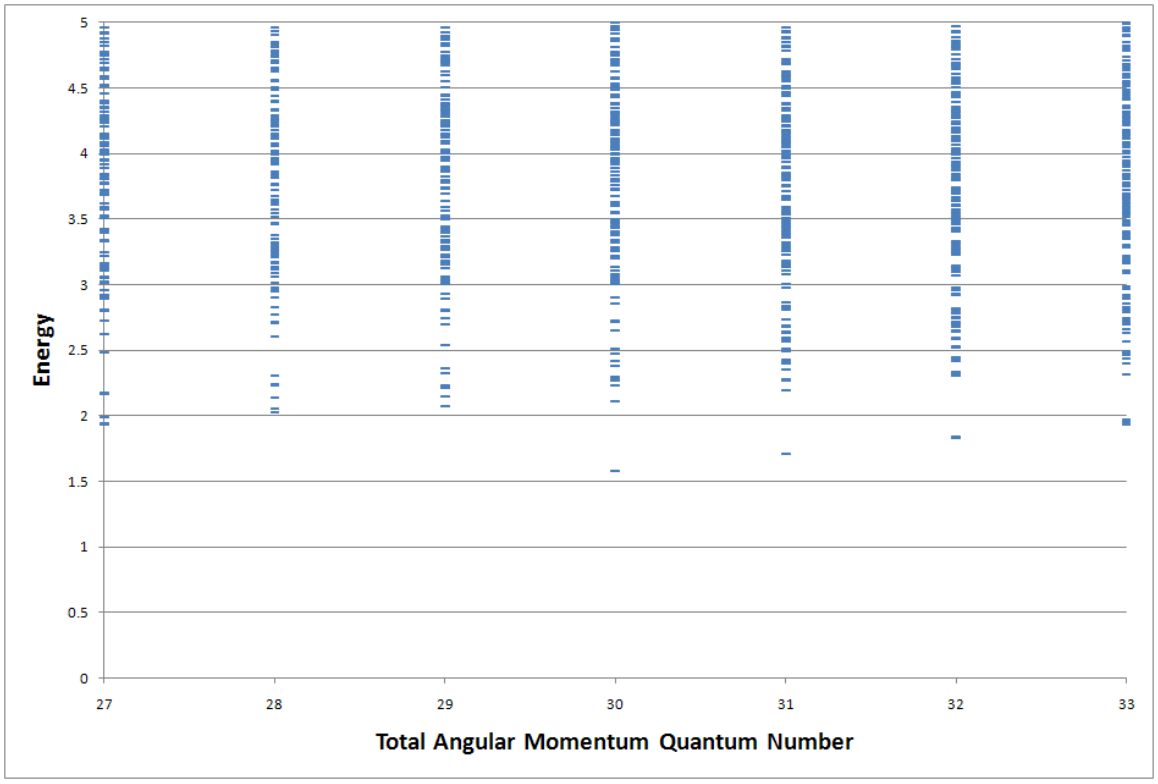

Figure 5.1 MPS Eigenenergy as function of Angular Momentum

5.2 Chemical Potential

The chemical potential of the dot can be tuned to force electrons on or off of the dot. The magnitude of the chemical potential required to perform this is directly linked to the energy eigenvalues and therefore depends on the system parameters mention in the previous section. The chemical potential controls the expected number of electrons in the dot via a statistical averaging term:

$$\langle N \rangle = \frac{\sum_n N_n e^{\beta(E_n - \mu_D N_n)}}{\sum_n e^{\beta(E_n - \mu_D N_n)}}. \tag{5.1}$$

At very low temperatures this expression is dominated by the lowest-energy state (the ground state) E_N^{GS} for each particle number N. This gives a simple way to get a feel for the chemical potentials required. This can be done by plotting $\left(E_N^{GS} - \mu_D N\right)$ as a function of μ_D for two states that differ in particle number by one particle. The intersection of these functions is the chemical potential at

which the system jumps from N to $N+1$ electrons at zero temperature. That is,

$$E_N^G S - \mu_D N = E_{N+1}^G S - \mu_D (N+1) \tag{5.2}$$

$$\mu_D = E_{N+1}^{GS} - E_N^{GS} \tag{5.3}$$

An example of expected particle number as a function of chemical potential in the dot is shown in Figure 5.2,. Here the temperature is nonzero, so the jumps are not discontinuous. As the chemical potential increases, more and more electrons are drawn onto the dot.

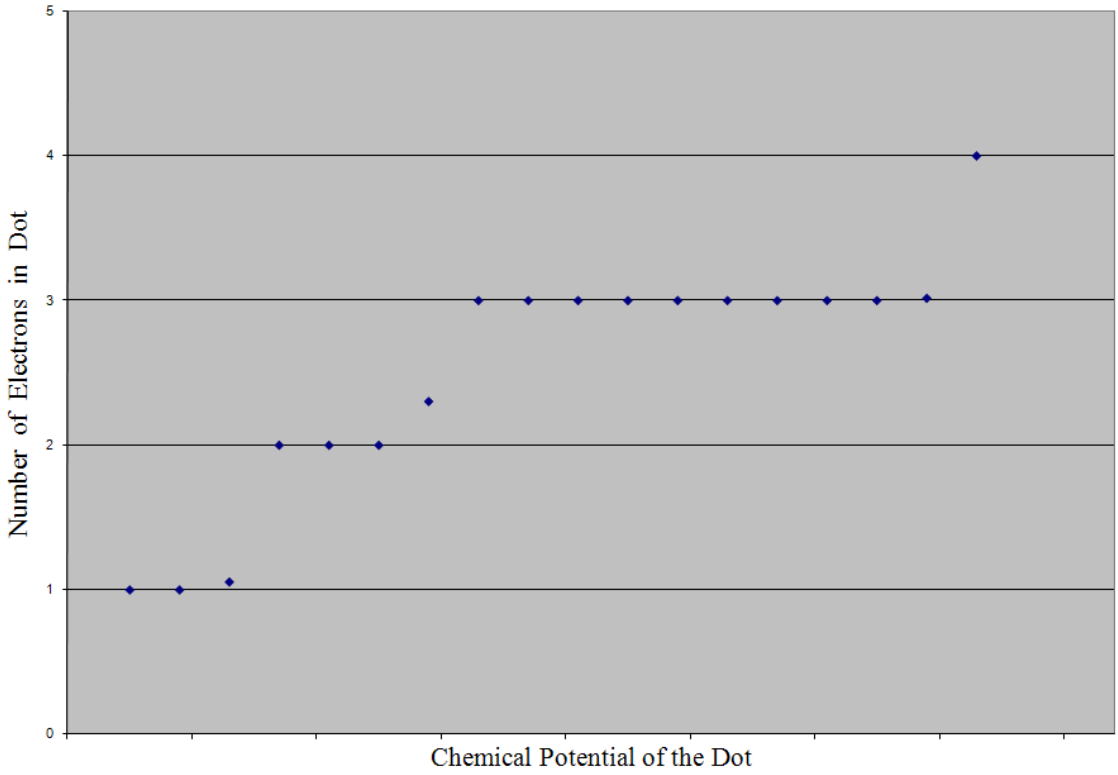

Figure 5.2 Particle Number in Dot as a Function of Dot Chemical Potential

However, the positions of the transitions are accurately given by the process just outlined. In our calculations of tunneling current we focused on tunneling into and out of dots with well-defined numbers of electrons. To do so, we selected a chemical potential midway between the value that gave the $N-1$ to N and N to $N+1$ transitions. Using this value of μ_D, even at nonzero temperatures, produces a system with a well-defined expected number of electrons in the dot, very close to N.

5.3 Tunneling Current

We now present representative results for the tunneling current as a function of radial tunneling position for several system. Figures 5.3 through 5.6 display tunneling currents as a function of position from systems with different particle numbers.

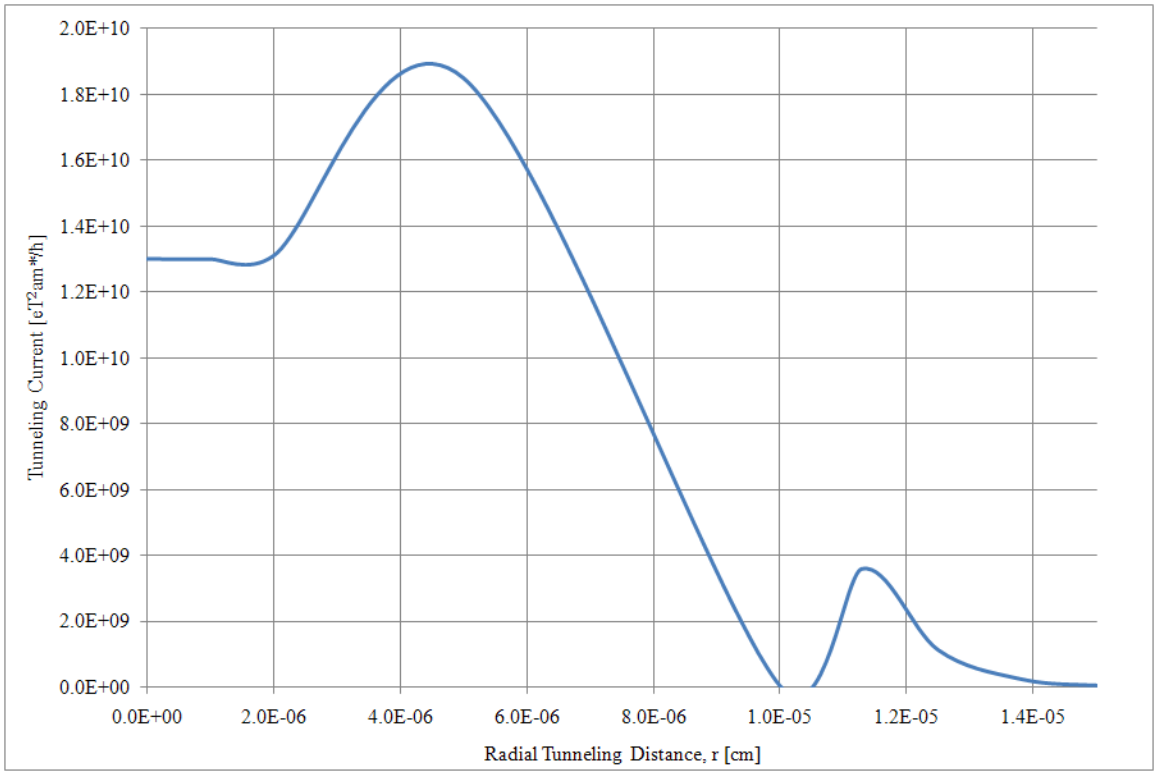

Figure 5.3 Tunneling Current vs Radial Tunneling Position for N = 3

All of the other parameters were kept the same for all the plots so that the effect of dot's occupancy and radial tunneling position were isolated. What can be seen from these plots is that although the radial position effects the systems differently, they do all share similar structured peaks. Some insight into the tunneling was obtained by examining which many-body eigenstates gave the largest contribution to the tunneling current. The choice of μ_D described above makes it clear that the dominant contributions are connected to the ground state of the system with N_{target} electrons. Indeed, the contribution of the ground state with N_{target} electrons is many orders of magnitude greater than the contribution to tunneling of any other N_{target} eigenstates.

It turns out in addition that the connections between this state and relatively few states with

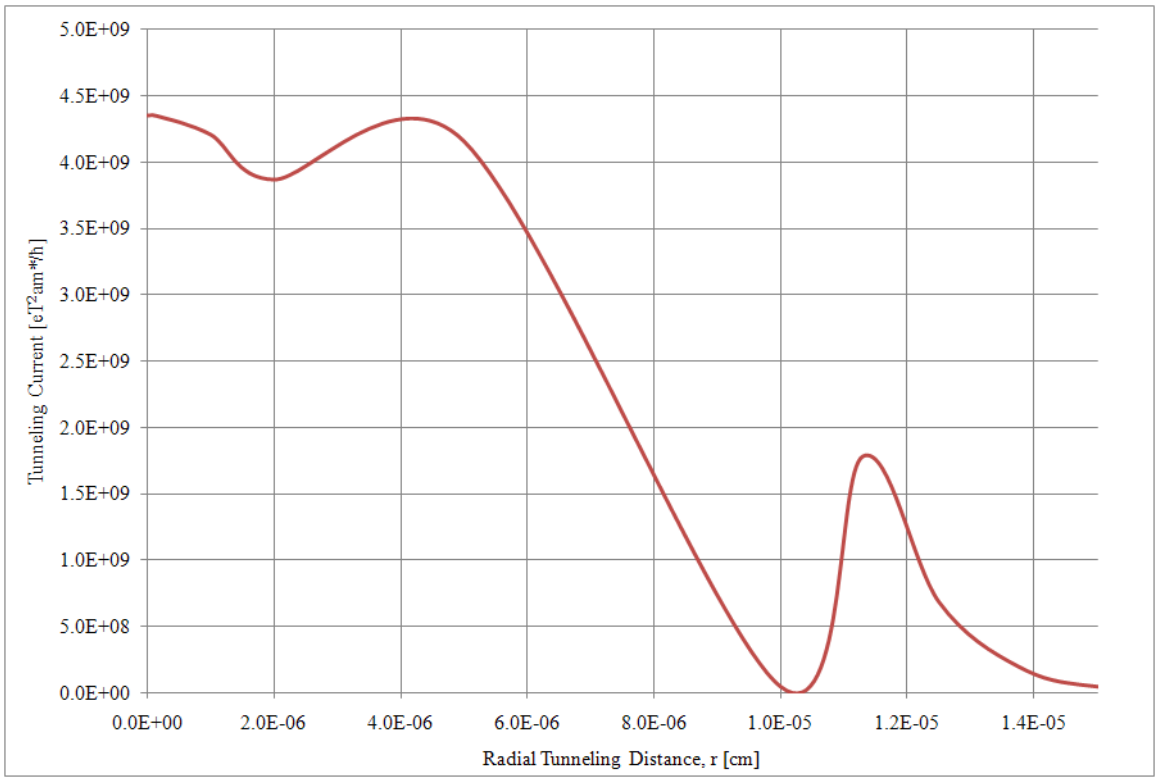

Figure 5.4 Tunneling Current vs Radial Tunneling Position for N = 4

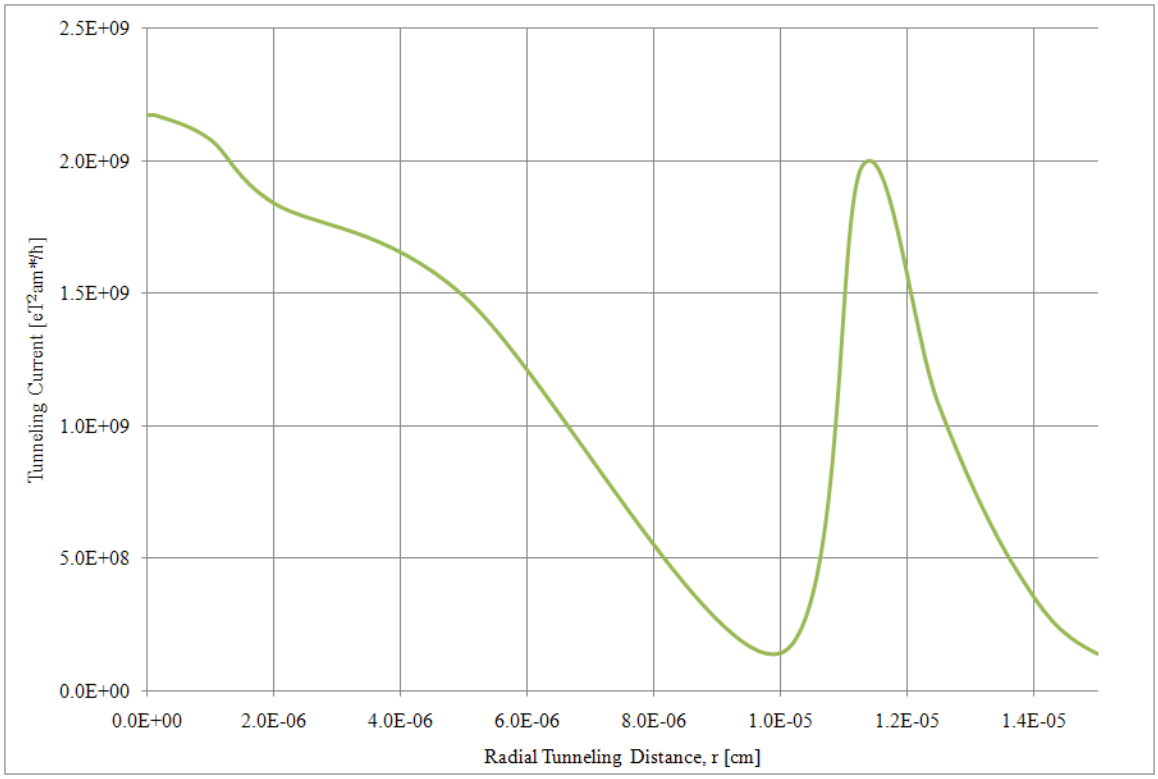

Figure 5.5 Tunneling Current vs Radial Tunneling Position for N = 5

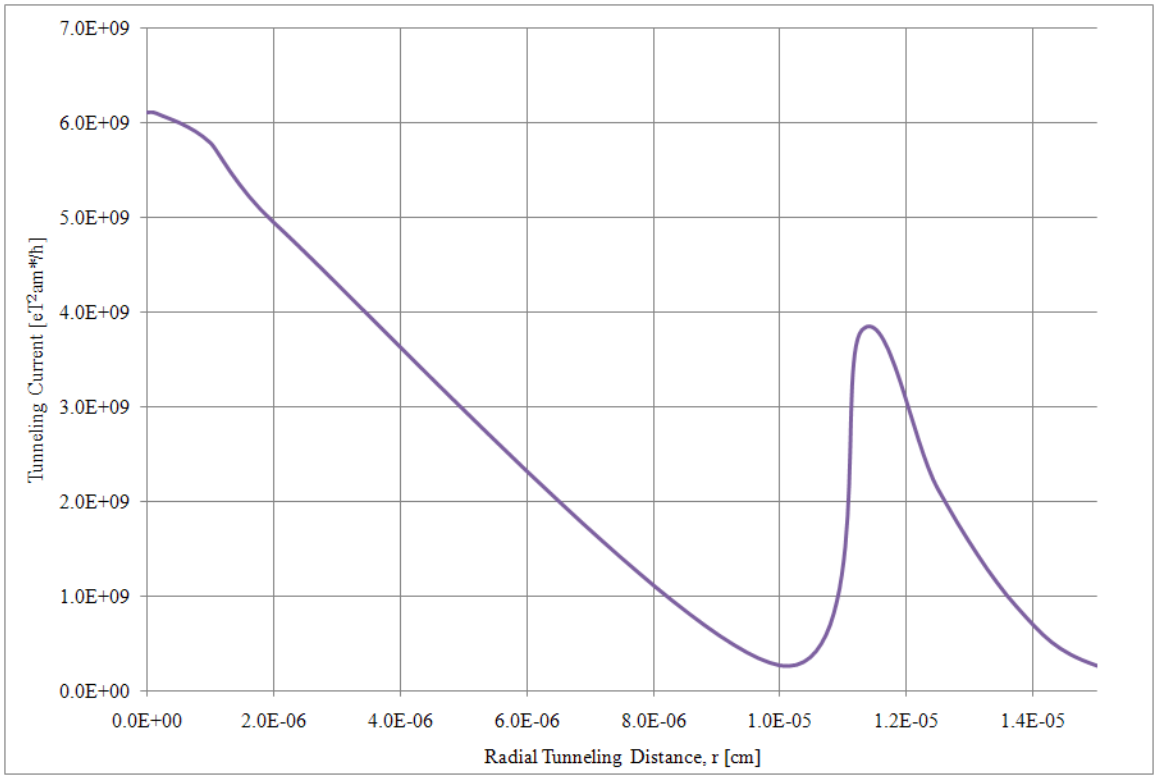

Figure 5.6 Tunneling Current vs Radial Tunneling Position for N = 6

$N \pm 1$ electrons give the dominant contribution to tunneling at each r. Although exactly which states cannot be predicted before hand, by comparing the current contributions it is clear that the dominant states contributes many orders of magnitude more than any other states. However, the dominant states had a relatively small range of mtot at each value of r. A simple explanation is given by the relatively localized nature of the single-particle states in the lowest Landau level. If the single-particle state with momentum m is the state largest near a particular r, then tunneling at r will be dominated by $N \pm 1$ many-body eigenstates whose total angular momentum differs from the N_{target} ground state by approximately m.

5.4 Conclusions and future directions

This work represents the first first-principles calculation of tunneling currents into quantum dots exhibiting the FQHE. The choice of a first-principles approach gives unbiased results, but limits the calculation to relatively small systems. Because the FQHE is a result of short-range interactions,

such small systems are known to give accurate and useful insight into the larger systems available experimentally. Thus the approach used here, and the tunneling currents obtained, are of some value.

There are several possible future directions that this work could take. By taking advantage of the approximate selection rule found in the plots explained above, it may be possible to identify which many-body energy eigenstates will contribute significantly beforehand. If so, it will be possible to study larger systems by using a numerical method that obtains only the desired eigenstates (e.g., Lanczos). This work represents the first time the tunneling current was calculated from first principles and the coefficients were not calculated, however other longer-term work could include repeating the calculation at various temperatures and generalizing this approach to study AC tunneling to determine the coefficients.

APPENDIX A
ANTI-COMMUTATION RELATIONS

The expressions for tunneling current include terms for the creation and annihilation for particles. These are required in order to determine the origin and destination of the particles and therefore the states that contribute to the tunneling. The physics of this system of Fermions requires that these operators follow anti-commutation relations. The following anti-commutation relations are those required:

$$\left\{c_{\vec{k}}^{\dagger}, \psi^{\dagger}(\vec{r})\right\} = \left\{c_{\vec{k}}^{\dagger}, \psi(\vec{r})\right\} = \left\{c_{\vec{k}}, \psi^{\dagger}(\vec{r})\right\} = \left\{c_{\vec{k}}, \psi(\vec{r})\right\} = 0$$

$$\left\{c_A, c_B\right\} = \left\{c_A^{\dagger}, c_B^{\dagger}\right\} = \left\{\psi_A, \psi_B\right\} = \left\{\psi_A^{\dagger}, \psi_B^{\dagger}\right\} = 0$$

$$\left\{c_A, c_B^{\dagger}\right\} = \left\{\psi_A, \psi_B^{\dagger}\right\} = \delta_{AB}.$$

APPENDIX B
TIME DEPENDENT CREATION AND ANNIHILATION OPERATORS

When performing calculations using the language of second quantization it is necessary to understand and take every advantage of the commuting and non-commuting properties of operators. In particular, manipulation of the time dependencies of creation and destruction operators is of interest. This appendix demonstrated in detail a treatment of the time propagation of a time dependent annihilation operator. This operator's Hamiltonian is comprised of non-commuting terms and therefore algebraic care must be taken.

Given the following time dependent annihilation operator,

$$
\begin{aligned}
c_{\vec{k}}(t) &= e^{\frac{i}{\hbar}K_L t} c_{\vec{k}}\, e^{-\frac{i}{\hbar}K_L t} \\
K_L &\equiv \hat{\mathscr{H}}_L - \mu_L \hat{N}_L \\
\hat{\mathscr{H}}_L &\equiv \sum_{\vec{k}'} \varepsilon_{\vec{k}'} c^{\dagger}_{\vec{k}'} c_{\vec{k}'} \\
\hat{N}_L &\equiv \sum_{\vec{k}'} c^{\dagger}_{\vec{k}'} c_{\vec{k}'}
\end{aligned}
$$

we have our starting point

$$
c_{\vec{k}}(t) = e^{\frac{i}{\hbar}\sum_{\vec{k}'}\left(\varepsilon_{\vec{k}'}-\mu_L\right)c^{\dagger}_{\vec{k}'}c_{\vec{k}'}t}\, c_{\vec{k}}\, e^{-\frac{i}{\hbar}\sum_{\vec{k}'}\left(\varepsilon_{\vec{k}'}-\mu_L\right)c^{\dagger}_{\vec{k}'}c_{\vec{k}'}t}.
$$

The time dependent terms can be regrouped such that they reside only to the left of the annihilation operator as

$$
c_{\vec{k}}(t) = e^{-\frac{i}{\hbar}\left(\varepsilon_{\vec{k}}-\mu_L\right)t} c_{\vec{k}}.
$$

It is worth slowing down here and understanding the meaning of the terms in these expressions so that we can gain some insight into the physics. First note $\vec{k}$ indicates a specific eigenstate and carats signify operators with the exception of creation and annihilation operators where they have been omitted for clarity. These operators are associated with particles in the lead, signified by the subscript L, and not in the dot.

The following definitions are made to allow for a more compact representation of our starting point and are used throughout this appendix.

$$A \equiv \frac{i}{\hbar} K_L t$$

$$B \equiv c_{\vec{k}}$$

So that our starting point becomes

$$c_{\vec{k}}(t) = e^{A} B e^{-A}.$$

The following is not a rigorous proof, but rather is intended to give a sense for how the math works. We begin by expanding the series and keep terms up to 3rd order.

$$\begin{aligned} B(t) &= (1 + A + \frac{A^2}{2} + \cdots) B (1 - A + \frac{A^2}{2} - \cdots) \\ B(t) &= B - BA + \frac{BA^2}{2} + AB - ABA + \frac{ABA^2}{2} + \frac{A^2 B}{2} \\ &\quad - \frac{A^2 BA}{2} + \frac{A^2 BA^2}{4} + \cdots \\ B(t) &= B + AB - BA + \frac{BA^2}{2} - ABA + \frac{A^2 B}{2} + \cdots \end{aligned}$$

Now grouping terms of similar order.

$$B(t) = B + [A, B] + \left(\frac{BA^2}{2} - ABA + \frac{A^2 B}{2} \right) + \cdots$$

In order to further reduce the expression of equation , we will investigate the details of AB. From the definitions of A, B, $\hat{k}_L$, $\hat{\mathscr{H}}_L$ and $\hat{N}_L$, and noting that t commutes with all of the operators,

$$AB = \frac{i}{\hbar} t \sum_{\vec{k}'} \left(\varepsilon_{\vec{k}'} - \mu_L \right) c^{\dagger}_{\vec{k}'} c_{\vec{k}'} c_{\vec{k}}$$

At this point it would be useful to recall the anti-commutation relations of the creation and annihilation operators,

$$\begin{aligned}\left\{c_x, {c_y}^\dagger\right\} &= \delta_{xy} \\ \left\{c_x, c_y\right\} &= 0 \\ \left\{{c_x}^\dagger, {c_y}^\dagger\right\} &= 0.\end{aligned}$$

Using these to rearrange the operators we have

$$\begin{aligned}AB &= \frac{i}{\hbar}t\sum_{\vec{k'}}\left(\varepsilon_{\vec{k'}}-\mu_L\right)c^\dagger_{\vec{k'}}c_{\vec{k}}c_{\vec{k'}}(-1) \\ AB &= \frac{i}{\hbar}t\sum_{\vec{k'}}\left(\varepsilon_{\vec{k'}}-\mu_L\right)\left(\delta_{\vec{k}\vec{k'}}-c_{\vec{k}}c^\dagger_{\vec{k'}}\right)c_{\vec{k'}}(-1) \\ AB &= \frac{i}{\hbar}t\sum_{\vec{k'}}\left(\varepsilon_{\vec{k'}}-\mu_L\right)\left(c_{\vec{k}}c^\dagger_{\vec{k'}}c_{\vec{k'}}\right)-\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}}-\mu_L\right)c_{\vec{k}} \\ AB &= BA-\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}}-\mu_L\right)c_{\vec{k}}\end{aligned}$$

Writing this in terms of a commutator so that a direct substitution can be made gives

$$[A,B] = -\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}}-\mu_L\right)B$$

so that

$$B(t) = (1)B-\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}}-\mu_L\right)B+\left(\frac{BA^2}{2}-ABA+\frac{A^2B}{2}\right)+\cdots$$

Now consider the rightmost term. We will look at each of these terms separately. The goal here is to factor a B out to the right so that hopefully the exponential series that seems to be forming does

materialize. The first term gives us,

$$\begin{aligned}
\frac{BA^2}{2} &= \frac{1}{2}(BA)A \\
\frac{BA^2}{2} &= \frac{1}{2}\left(AB + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)B\right)A \\
\frac{BA^2}{2} &= \frac{1}{2}\left(A + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)\right)BA \\
\frac{BA^2}{2} &= \frac{1}{2}\left(A + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)\right)\left(AB + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)B\right) \\
\frac{BA^2}{2} &= \frac{1}{2}\left(A + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)\right)^2 B
\end{aligned}$$

The middle term is factored in a similar manner,

$$\begin{aligned}
-ABA &= -A\left(BA\right) \\
-ABA &= -A\left(AB + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)B\right) \\
-ABA &= \left(-A^2 - \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)A\right)B
\end{aligned}$$

The last term is already in the desired form. We can now make substitutions to get,

$$\begin{aligned}
B(t) = (1)B - \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)B + \frac{1}{2}\left(A + \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)\right)^2 B \\
+ \left(-A^2 - \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)A\right)B + \frac{A^2 B}{2} + \cdots
\end{aligned}$$

This is more easily written if we introduce another definition,

$$D \equiv \frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)$$

Substituting D and expanding we can see that many terms will cancel,

$$B(t) = (1)B - (D)B + \frac{A^2 B}{2} + \frac{D^2 B}{2} + DAB - A^2 B - DAB + \frac{A^2 B}{2} + \cdots$$

From here we find our result as follows

$$B(t) = (1)B - (D)B + \left(\frac{D^2}{2}\right)B + \cdots$$
$$B(t) = \left(1 - D + \frac{D^2}{2} + \cdots\right)B$$
$$B(t) = e^{-D}B$$
$$c_{\vec{k}}(t) = e^{-\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)}c_{\vec{k}}.$$

Another way to demonstrate this is by considering the partial time derivative of the time dependant annihilation operator,

$$c_{\vec{k}}(t) = e^{\frac{i}{\hbar}K_L t}c_{\vec{k}}\, e^{-\frac{i}{\hbar}K_L t},$$

resulting in

$$i\hbar\frac{\delta c_{\vec{k}}(t)}{\delta t} = e^{\frac{i}{\hbar}K_L t}\left[c_{\vec{k}}, K_L\right]\, e^{-\frac{i}{\hbar}K_L t}.$$

The commutator can be written in terms of an energy difference and operator as

$$\left[c_{\vec{k}}, K_L\right] = \left(\varepsilon_K - \mu_L\right)c_{\vec{k}}$$

and by substituting this into the time derivative equation we have

$$i\hbar\frac{\delta c_{\vec{k}}(t)}{\delta t} = \left(\varepsilon_K - \mu_L\right)c_{\vec{k}}(t)$$

and therefore

$$c_{\vec{k}}(t) \quad = e^{-\frac{i}{\hbar}t\left(\varepsilon_{\vec{k}} - \mu_L\right)}c_{\vec{k}}.$$

APPENDIX C
GENERAL OPERATOR CONJUGATION

Given a general operator $\hat{\Omega}$, the complex conjugation is accomplished in the following way.

$$\begin{aligned}
\left\langle \hat{\Omega} \right\rangle^* &= \left(\frac{1}{z} \sum_n e^{-\beta(E_n - \mu N_n)} \left\langle n \left| \hat{\Omega} \right| n \right\rangle \right)^* \\
&= \frac{1}{z} \sum_n \left(\left\langle n \left| e^{-\beta(E_n - \mu N_n)} \hat{\Omega} \right| n \right\rangle \right)^* \\
&= \frac{1}{z} \sum_n \left\langle n \left| \hat{\Omega}^\dagger \left(e^{-\beta(E_n - \mu N_n)} \right)^\dagger \right| n \right\rangle \\
&= \frac{1}{z} \sum_n \left\langle n \left| \hat{\Omega}^\dagger e^{-\beta(E_n - \mu N_n)} \right| n \right\rangle \\
&= \frac{1}{z} \sum_n e^{-\beta(E_n - \mu N_n)} \left\langle n \left| \hat{\Omega}^\dagger \right| n \right\rangle
\end{aligned}$$

This can also be stated more generally and compactly as

$$\left\langle i \left| \hat{\Omega} \right| j \right\rangle^* = \left\langle j \left| \hat{\Omega}^\dagger \right| i \right\rangle .$$

APPENDIX D
NUMERICAL ANALYSIS CODE: DISKMAIN.F

```
      program diskmain
!************************************************************************
!  Establish a complete set of multi-particle basis states and determine the
!  eigenvectors and energy eigenvalues of a fqhe disk with a V1 model or
!  nearest neighbor pseudopotential
!************************************************************************
!  N - number of electrons.
!  SPS - Single Particle States that compose the Multi-Particle States
!  Mouter = SPS - 1 (a useful value for programming since counting sometimes
!       begins with zero rather than one).
!  The program solves for N electrons feeling the confining potential W(m).
!  This potential follows the V1 pseudopotential model which uses two
!       parameters with variable names medge and diagscale.
!  Single-particle eigenstates are labelled by m = 0 to Mouter (<=sps), so an
!       N-body state is labelled by m(1),m(2),...,m(N).
!  The number of many-body states in the Hilbert space is (Mouter+1 choose N)
!       which must be <= hilbertspace(sps,n).
!  H(state) is the Hamiltonian matrix in packed form---i.e., the
!       lower triangle, stored row-wise.
!  The basis of states is stored in mbasis:  mbasis(ielectron,istate) for
!       ielectron=1 to N gives the occupied m's for the istate'th basis state.
!  C(i,j) = (i choose j).
!  CC(m1,m2,mp) is the coefficient arising from the decomposition of
!       phi(m1,z1)phi(m2,z2) into
!       (sum m=0 to m1+m2)phi(m,(z1-z2)/rt(2))phi(m1+m2-m,(z1+z2)/rt(2)).
!  Here m = 2*mp-1 which is needed for the pseudopotential calculation.
!  The routine computes separately for each total angular momentum Mtot
!       where Mtot = m(1) + m(2) + ... + m(N).
!  mtot_state is the number of basis states with the same mtot.
!************************************************************************
      use DECLARATIONS
      use SYS_PARAMETERS
```

```
      implicit none
      integer N,Mouter,Mtotmin,Mtotmax,total_states, mtot_state
      integer dewdummy, mp, m, I, J, K, b_state_error, mtot_index
      integer e_data_error, matz
      character*25 e_name, b_name
      logical more_sps, more_N

      write(*,'(14x,''meffect ='',E11.4,/,''  magnetic_length(BT) =
     .'',E11.4,/
     .,'' cyclotron_energy(BT) ='',E11.4,/,17x,''cerg ='',E11.4,/,10x,
     .''beta(BT, T) ='',E11.4,/)') meffect , magnetic_length(BT),
     .cyclotron_energy(BT), coulomb_energy(BT), beta(BT, T)
!*************************************************************************
!  matz is a switch that is used by the RSP routine.  It is passed to
!  RSP through the DIAGONALIZE subroutine found in this main program.
!  The matz integer variable is set to zero if only eigenvalues are
!  desired or set to any non-zero integer for both eigenvalues and
!  eigenvectors.
!*************************************************************************
      matz = 1

!*************************************************************************
!  List of the output files.
!*************************************************************************
!     UNIT        Name                    Contents
!     ----        ----                    --------
!      1          Hamiltonians.dat        undiagonalized lower-triangle format
!      2          Interactions.dat        undiag hamiltonian w/ interactions
!      3          eigenvalues.dat         all the E-values for the run
!      4          eigenvectors.dat        all the E-vectors for the run
!      5          e_data_##_##.dat        eigensystem data for ##(N)_##(SPS)
!      6          b_state_##_##.dat       basis states for ##(N)_##(SPS)
!*************************************************************************
```

```
!  The following files are used to verify and test the program and should be
!  commented out after testing and debugging.
!    open(unit=1,file='C:\temp\Hamiltonians.dat',ACCESS='SEQUENTIAL',
!     .    ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
!     .    PAD='YES',STATUS='REPLACE')
!      open(unit=2,file='C:\temp\Interactions.dat',ACCESS='SEQUENTIAL',
!     .    ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
!     .    PAD='YES',STATUS='REPLACE')
      open(unit=3,file='C:\temp\Eigenvalues.dat',ACCESS='SEQUENTIAL',
     .    ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='REPLACE')
      open(unit=4,file='C:\temp\Eigenvectors.dat',ACCESS='SEQUENTIAL',
     .    ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='REPLACE')

      write(*,'(''Input the SPS that composes the MPS.'')')
      read(*,*) SPS
      more_sps = .true.
!*******************************************************************************
!*******************************************************************************
!*******************************************************************************
!  Loop through input SPS while user wants to continue

      SPS_loop:  do while (more_sps.eq..true.)
         write(*,'(''     SPS = '',I2)') SPS
         write(*,*)

!  Remember that SPS starts counting from 1 while Mouter starts from 0.
      Mouter = SPS - 1
         write(*,'(''State m where potential begins and its slope.'')')
         write(*,'(''Medge ='',f4.1,5X,''diagscale ='',f4.1)') Medge,
     .        diagscale
         write(*,*)
```

```
!  V1 model.
         allocate (Vpseudo(1 : sps))
         Vpseudo = 0.0
         Vpseudo(1) = 1.0
!         Vpseudo(2) = 0.01

         allocate (W(0 : sps))
         allocate (Welec(0 : sps))
         allocate (CC(0 : sps , 0 : sps , 1 : sps))
         call projcoeff(Mouter)

         write(*,'(''Input number of electrons.'')')
         write(*,'(''This must be greater than zero , less than'')')
         write(*,'(''SPS and less than 30 or program terminates.'')')
         read(*,*) N
         more_N=.true.
!*************************************************************************
!*************************************************************************
!  Loop through different numbers of particles N.
         N_loop:  do while (more_N.eq..true.)
            write(*,'(''     N = '',I2)') N
            write(*,*)
!  Terminate the program for bad N
            if (N.le.0. or .N.gt.sps. or .N.gt.30) stop 'bad N'

            e_name =
     .           'C:\temp\e_data_'//suffix(N)//'_'//suffix(SPS)//'.dat'
            open(unit=5,file=e_name,ACCESS='SEQUENTIAL',
     .          ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
     .          PAD='YES',STATUS='REPLACE')
            b_name =
     .           'C:\temp\b_state_'//suffix(N)//'_'//suffix(SPS)//'.dat'
```

```
            open(unit=6,file=b_name,ACCESS='SEQUENTIAL',
     .           ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
     .           PAD='YES',STATUS='REPLACE')

!*************************************************************************
!  mbasis is used throughout the program and must be sized based on the
!  largest array required for the given system (given N and SPS).  This
!  maximum size value is found beforehand by recording data from previous runs
!  (initially guessing high and recording the peak value of mtot_state).
!  These are stored as a CASE lookup function called stack_dim(sps) in the
!  DECLARATIONS module.
            allocate (mbasis(0 : N , 1 : stack_dim(sps)))
!*************************************************************************
!  Allocate/initiallize data arrays entering the angular momentum loop.  These
!  are used to store the data for a particular run and then dumped to one of
!  the data files.  After the data is recorded the data arrays are then
!  deallocated.
            allocate (e_data_array(1 : STACK_DIM(SPS),
     .                      1 : (MAXMTOT(SPS,N) - MINMTOT(SPS,N)) + 1,
     .                      0 : STACK_DIM(SPS)))
            e_data_array = 0
            allocate (b_state_array(0 : STACK_DIM(SPS),
     .                  1 : (MAXMTOT(SPS,N) - MINMTOT(SPS,N)) + 1))
            b_state_array = 0

!  Loop through different mtot from min to max
            total_states = 0
            angular_momentum_sweep:  do Mtot = minmtot(sps,n),
     .                                                 maxmtot(sps,n)
               call setbasis(Mouter,N,mtot_state)
               mtot_index = (mtot - minmtot(sps,n)) + 1
               b_state_array(0,mtot_index) = mtot_state
               basis_state_info: do i = 1, mtot_state
```

```
                  b_state_array(i ,mtot_index) = mbasis(0,i)
               end do basis_state_info
               total_states = total_states + mtot_state
               H_size = (mtot_state*(mtot_state+1))/2
               allocate (H(1 : H_size))
               call diagonal(Mouter,N,mtot_state)
!*************************************************************************
!  For testing, comment write statements if not testing.
!               write(1,'(''Mouter / N / Mtot: '',3i6)') mouter, n, mtot
!               write(1,'(''H('',i3,'')='',g20.9)') (dewdummy,
!     .              H(dewdummy), dewdummy = 1,
!     .              (mtot_state*(mtot_state+1))/2)
               allocate (evalues(1 : mtot_state))
               allocate (e_total(1 : mtot_state))
               allocate (evectors(1 : mtot_state, 1 : mtot_state))
               call interaction(Mouter,N,mtot_state)
!*************************************************************************
!  For testing, comment write statements if not testing.
!               write(2,'(''Mouter / N / Mtot: '',3i6)') mouter, n, mtot
!               write(2,'(''H('',i2,'')='',g20.14)') (dewdummy,
!     .              H(dewdummy),dewdummy=1,H_size)
!  Initialize the eigensystem arrays, calculate the vectors/values and
!  output them to a file for checking/debugging purposes.
               evalues = 0
               evectors = 0
               call diagonalize(Mouter,N,mtot_state,matz)
!  The energies that have been calculated to this point and stored the evalues
!  array are associated with the interaction (potential energy) and confining
!  potential parts of the overall Hamiltonian.  These are in energy units of
!  [cerg], but don't contain the kinetic energy.  Here we will add the kinetic
!  energy as discribed in the notes which will result in the total energies
!  E_total in units of [cerg] and store them in the e_total array.  The
!  evalues and e_total arrays are of the same size and corresponding data will
```

```
!  be stored with the same index within their respective arrays.
              forall (i = 1:mtot_state) e_total(i) = evalues(i) +
     .                (cyclotron_energy(BT)*N)/(2*coulomb_energy(BT))
              write(3,'(i5,2x,g20.14,2x,g20.14)') (Mtot,evalues(i),
     .             e_total(i), i=1, mtot_state)
              write(4,'(i5,5x,<mtot_state>f20.14)')(Mtot,
     .              (evectors(i,j),i=1,mtot_state),j=1,mtot_state)
!  Store the eigensystem data in an array that will later be written to a
!  file all at once.
              e_data_row:  do i = 1, mtot_state
                 e_data_array(i,mtot_index,0) = e_total(i)
                 e_data_z:  do k = 1, mtot_state
                    e_data_array(i,mtot_index,k) = evectors(k,i)
                 end do e_data_z
              end do e_data_row
              deallocate (H,evalues,evectors,e_total)
           end do angular_momentum_sweep
!*************************************************************************
!  For testing, comment write statements if not testing.  Write the
!  eigensystem information to a file and then get rid of the arrays.
!
!  We will record the parameters that discribe the run as follows:
!       Magnetic Induction Field [T]
!       temperature [K]
!       dielectric constant []
!       the single-particle potential begins at m = medge
!       slope of confining potential (diagscale)
!       row (first) dimension of e_data_array
!       column (second) dimension of e_data_array
!       z (third) dimension of the e_data_array
            write(5,'(f20.14)') BT, T, dielec, medge, diagscale
            write(5,'(3i8)') STACK_DIM(SPS),
     .           (MAXMTOT(SPS,N) - MINMTOT(SPS,N)) + 1, STACK_DIM(SPS)
```

```
            write(5,'(f20.14)') e_data_array
!  The following loops write the data in a way that allows for easier
!  checking by a human.  If you need to verify the data comment out the above
!  line and uncomment the following loops.
!            write(5,'(f20.14)') BT, T, dielec, medge, diagscale
!            eigen_output_row: do i = 1, STACK_DIM(SPS)
!               eigen_output_column: do j = 1, ((MAXMTOT(SPS,N) -
!     .                                             MINMTOT(SPS,N)) + 1)
!                  eigen_output_z:  do k = 0, STACK_DIM(SPS)
!                     write(5,'(i4,i4,i4,f20.14)')
!     .                               i,j,k,e_data_array(i,j,k)
!                  end do eigen_output_z
!               end do eigen_output_column
!            end do eigen_output_row

!  Write the basis state information to a file and then get rid of the arrays
            write(6,'(2i8)') (STACK_DIM(SPS)),
     .                          (MAXMTOT(SPS,N) - MINMTOT(SPS,N)) + 1
            write(6,'(i10)') b_state_array
!  The following loops write the data in a way that allows for easier checking
!  by a human.  If you need to verify the data comment out the above line and
!  uncomment the following loops.
!            write(6,'(2i8)') (STACK_DIM(SPS)),
!     .                          (MAXMTOT(SPS,N) - MINMTOT(SPS,N)) + 1
!            basis_output_column: do j = 1, ((MAXMTOT(SPS,N) -
!     .                                             MINMTOT(SPS,N)) + 1)
!               basis_output_row: do i = 0, STACK_DIM(SPS)
!                  write(6,'(i4,i4,i10)') i,j,(b_state_array(i,j))
!               end do basis_output_row
!            end do basis_output_column

            deallocate (b_state_array, stat = b_state_error)
            deallocate (e_data_array, stat = e_data_error)
```

```
            deallocate (mbasis)


            write(*,*)
            write(*,'(''Total number of states in Hilbert space '',
     .           i5)')hilbertspace(sps,n)
            write(*,'(''Total number of states '',i5)') total_states
            write(*,'(''If you would like to try another N for the same
     .SPS enter a number between 1 and SPS.'')')
            write(*,'(''If you would like to try a different SPS enter 0
     . (default).'')')
            read(*,*) N
            if (N.ge.1 .and. N.le.SPS) then
               more_N = .true.
               write(*,'(''*****************************'')')
            else
               more_N = .false.
               write(*,'(''***************  NEW SPS  ***************'')')
            end if
         end do N_loop
!*************************************************************************
!*************************************************************************
!  For testing, comment write statements if not testing.
         deallocate (Vpseudo, CC, W, Welec)
         write(*,*)
         write(*,'(''If you would like to try another SPS enter a num
     .ber between 1 and 30.'')')
         write(*,'(''If you would quit enter 0 (default).'')')
         read(*,*) SPS
         if (SPS.ge.1 .and. SPS.le.30) then
            more_SPS = .true.
         else
            more_SPS = .false.
         end if
```

```
         end do SPS_loop
!*******************************************************************************
!*******************************************************************************
!*******************************************************************************
      end
```

APPENDIX E
NUMERICAL ANALYSIS CODE: MU_DOT.F

```
      program mu_dot
!************************************************************************
!  This program calculates the MPS Eigen-energies (E) for a given number of
!  SPS.  It does this for all of the possible number of particles (N) for the
!  given SPS (in other words for N = 1 upto N = SPS).  It records these
!  energies along with their associated particle numbers so that they can be
!  used to determine the chemical potential of the dot (muD).  This is needed
!  to find the thermal average <N> in the tunnelling current expression.
!************************************************************************
!  See comments from diskmain.f as this program uses the same variables except
!  as described:
!************************************************************************
      use DECLARATIONS
      use SYS_PARAMETERS

      implicit none
      integer N,Mouter,Mtotmin,Mtotmax,total_states, mtot_state, next_N
      integer dewdummy, mp, m, I, J, K, mtot_index, mda_dim, mda_index
      integer matz
      real*8 gs_energy
      character*23 mu_name
      logical more_sps



      write(*,'(14x,''meffect ='',E11.4,/,''  magnetic_length(BT) =
     .'',E11.4,/
     .,'' cyclotron_energy(BT) ='',E11.4,/,17x,''cerg ='',E11.4,/,10x,
     .''beta(BT, T) ='',E11.4,/)') meffect , magnetic_length(BT),
     .cyclotron_energy(BT), coulomb_energy(BT), beta(BT, T)

!************************************************************************
!  This is a list of the output files.
!************************************************************************
```

```
!     UNIT        Name                    Contents
!     ----        ----                    --------
!      1          mu_##.dat               Chemical Potential of Dot
!                                         for SPS of ##
!*****************************************************************************
!  matz is a switch that is used by the RSP routine.  It is passed to
!  RSP through the DIAGONALIZE subroutine found in this main program.
!  The matz integer variable is set to zero if only eigenvalues are
!  desired or set to any non-zero integer for both eigenvalues and
!  eigenvectors.
      matz = 0

      write(*,'(''Input the SPS that composes the MPS.'')')
      read(*,*) SPS
        write(*,'(''     SPS = '',I2)') SPS
         write(*,*)

         mu_name = 'C:\temp\mu_'//suffix(SPS)//'.dat'
         open(unit=1,file=mu_name,ACCESS='SEQUENTIAL',
     .               ACTION='READWRITE', BLANK='NULL', FORM='FORMATTED',
     .               PAD='YES',STATUS='REPLACE')

!  Remember that SPS starts counting from 1 while Mouter starts from 0.
         Mouter = SPS - 1

!  V1 model.
         allocate (Vpseudo(1 : sps))
         Vpseudo = 0.0
         Vpseudo(1) = 1.0

         allocate (W(0 : sps))
         allocate (Welec(0 : sps))
         allocate (CC(0 : sps , 0 : sps , 1 : sps))
```

```
        W = 0
        Welec = 0
        CC = 0
        call projcoeff(Mouter)

!*************************************************************************
!  Initialize mda_index which is used to keep track of the number of e-values
!  written to the ne_data_array and therefore the last column in the array
!  which to which data was stored.  Using this information allows the next
!  group of e-values to be written to the array in the next available slot
!  (not over-writing existing data or creating unnecessary holes in array).
!  mda_index is bumped up after each pass through the angular_momentum_sweep
!  by the amount mtot_state and therefore keeps the running total of e-values
!  stored in ne_data_array.
        mda_index = 0

!*************************************************************************
!  Allocate/initiallize data array entering the angular momentum loop.  This
!  is used to store the data for a loop with a particular N.  Once each loop
!  is completed, the energy eigenvalues are dumped to a file and the array is
!  then deallocated.  The size of the ne_data_array will run from 1 upto
!  mda_dim.  mda_dim is the sum of number of e-values of the system with
!  N = 1, 2, 3,...  upto N = SPS.
        mda_dim = 0
        mda_dim_loop:  do i = 1 , SPS
           mda_dim = mda_dim + combin(SPS,i)
        end do mda_dim_loop
!*************************************************************************
!  The reason that the column (second) dimension starts at -4 rather than 1 is
!  that from column 0 to -4 is where the parameters that define the run are
!  stored.  Details of these parameters and their location in the array are
!  fully explaned at the comments proceding the WRITE statements at end of
!  this program.  Knowing these parameters will be important for subsequent
```

```
!  calculations.
         allocate (ne_data_array(1:2, 1:mda_dim))
         ne_data_array = 0.0
         allocate (mu_data_array(-4:sps))
         mu_data_array = 0.0
         next_N = 0
!************************************************************************
!************************************************************************
!  Loop through different numbers of particles N.
         N_loop:  do N = 1, sps

!  Terminate the program for bad N
            if (N.le.0. or .N.gt.sps. or .N.gt.30) stop 'bad N'

!  mbasis is used throughout the program and must be sized based on the
!  largest array required for the given system (given N and SPS).  This
!  maximum size value is found beforehand by recording data from previous runs
!  (initially guessing high and recording the peak value of mtot_state).
!  These are stored as a CASE lookup function called stack_dim(sps) in the
!  DECLARATIONS module.
            allocate (mbasis(0 : N , 1 : stack_dim(sps)))

!************************************************************************
!  Loop through different mtot from min to max
            total_states = 0
            angular_momentum_sweep:  do Mtot = minmtot(sps,n),
     .                                                  maxmtot(sps,n)
               call setbasis(Mouter,N,mtot_state)
               mtot_index = (mtot - minmtot(sps,n)) + 1
               total_states = total_states + mtot_state
               H_size = (mtot_state*(mtot_state+1))/2
               allocate (H(1 : H_size))
               H = 0
```

```
                call diagonal(Mouter,N,mtot_state)
                allocate (evalues(1 : mtot_state))
                evalues = 0
                call interaction(Mouter,N,mtot_state)
                call diagonalize(Mouter,N,mtot_state,matz)

!  Store the energy data in an array.
                ne_data_dump:  do i = 1, mtot_state
                   ne_data_array(1,mda_index + i) = N
                   ne_data_array(2,mda_index + i) = evalues(i)
                end do ne_data_dump
!  mda_index keeps track of the index of the previous loop's last entry
!  in the ne_data_array.  This is used on the following loop such that
!  the first entry of the next loop can pick-up where the previous loop
!  left off.
                mda_index = mda_index + mtot_state
                deallocate (H,evalues)
            end do angular_momentum_sweep
!*******************************************************************************
            deallocate (mbasis)

         end do N_loop
!*******************************************************************************
!*******************************************************************************
!  The energies that have been calculated to this point and stored in the
!  second column of ne_data_array (called E_code in the handwritten notes) are
!  associated with the interaction (potential energy) and confining potential
!  parts of the overall Hamiltonian.  These are in energy units of [cerg], but
!  don't contain the kinetic energy.  Here we will add the kinetic energy as
!  discribed in the notes which will result in the total energies E_total in
!  units of [cerg].

         energy_loop:  do i = 1, mda_index
```

```
            ne_data_array(2,i) = (cyclotron_energy(BT)
     .                       * ne_data_array(1,i))/(2*coulomb_energy(BT))
     .                       + ne_data_array(2,i)
!*************************************************************************
!  Just the same loop, sort through the energies for a given number of
!  particles, N, to find the minimum energy (ground) state.  Store the ground
!  state for each N, found in the ne_data_array, in the mu_data_array and
!  later write these to a file all at one time.  These ground state energies
!  will give us the T=0[K] chemical potentials at which the average N, <N>,
!  changes from N to N+1 particles.  Note that each time N changes for a given
!  SPS, the energy of the state first considered will be the initial guess
!  (stored in mu_data_array).  Following that, the energies of all subsequent
!  states with that same N will be sorted to find the minimum.  The process
!  will then move onto the next N.
            if (next_N.ne.ne_data_array(1,i)) then
               next_N = ne_data_array(1,i)
               mu_data_array(ne_data_array(1,i)) = ne_data_array(2,i)
            else
               if (mu_data_array(ne_data_array(1,i))
     .                                     .gt.ne_data_array(2,i)) then
                  mu_data_array(ne_data_array(1,i)) = ne_data_array(2,i)
               end if
            end if
         end do energy_loop
!  Before array is written out to a file we will record the parameters that
!  discribe run in row one (1) of the mu_data_array as follows:
!  mu_data_array(-4) = Magnetic Induction Field [T]
!  mu_data_array(-3) = temperature [K]
!  mu_data_array(-2) = dielectric constant []
!  mu_data_array(-1) = the single-particle potential begins at m = medge
!  mu_data_array(0) = slope of confining potential (diagscale)
         mu_data_array(-4) = BT
         mu_data_array(-3) = T
```

```
   mu_data_array(-2) = dielec
   mu_data_array(-1) = medge
   mu_data_array(0) = diagscale
   write(1,'(f20.14)') mu_data_array
   deallocate (Vpseudo, CC, W, Welec, ne_data_array,mu_data_array)
end
```

APPENDIX F
NUMERICAL ANALYSIS CODE: CORRELATIONS.F

```
      program correlation
!************************************************************************
      use DECLARATIONS
      use SYS_PARAMETERS

      implicit none
      real*8, allocatable, dimension(:,:,:) :: eigen_bra
      integer, allocatable, dimension(:,:) :: basis_bra
      real*8, allocatable, dimension(:,:,:) :: eigen_ket
      integer, allocatable, dimension(:,:) :: basis_ket
      real*8, allocatable, dimension(:,:) :: corr_array
      integer :: N, mtot_index, bra_basis_state, ket_basis_state
      integer :: bra_mtot, ket_mtot, k, m, i, j, x, y
      integer :: ket_row, ket_column, bit_count, sign
      integer :: bra_e_r, bra_e_c, bra_e_z, bra_b_r, bra_b_c
      integer :: ket_e_r, ket_e_c, ket_e_z, ket_b_r, ket_b_c
      integer :: N_target, b_state_index
      real*8 :: valid_bra, valid_ket, valid_pair, valid_sum, test, test2
      real*8 :: bra_BT, bra_T, bra_dielec, bra_medge, bra_diagscale
      real*8 :: ket_BT, ket_T, ket_dielec, ket_medge, ket_diagscale
      real*8 :: muD_BT, muD_T, muD_dielec, muD_medge, muD_diagscale
      real*8 :: I1, bra_energy, ket_energy, e_diff, corr_A, corr_B
      real*8 :: muD, muL, eGS_target, z, e_check, w_sum, I_tunnel
      character(80) :: e_bra_name, b_bra_name, e_ket_name, b_ket_name
      character(80) :: data_file, occupancy_file

!************************************************************************
!  N_target - Number of particles in the system that is desired to be
!           investigated.  It should also be noted that this is the number of
!           particles used to determining the chemical potential of the dot.
!           This is necessary as a result of how muD is selected (using the
!           mu_##.dat file from the program N_and_E.f) and the way N is cycled
!           through.  The value of muD needs to be found using N_target and
```

```
!           then fixed throughout the rest of the calculation.
!***************************************************************************
!  Find out what system the user is interested in processing
      write(*,'(''Input the number of Single Particle States (SPS) '')')
      write(*,'(''for the system you would like to investigate.'')')
      read(*,*) SPS
      write(*,'(''Input the number of electrons (N_target) for the '')')
      write(*,'(''system you want to investigate.  Note this '')')
      write(*,'(''program sweeps through systems with particles of '')')
      write(*,'(''(N-1) through (N_target + 1), so make sure that '')')
      write(*,'(''all the required data files are available.'')')
      read(*,*) N_target
!***************************************************************************
! This uses the same naming convention as was used to produce the data files
! and opens them as the following units:
!***************************************************************************
!     UNIT        Contents
!     ----        --------
!      1          Eigensystem data for the bra
!      2          Basis State data for the bra
!      3          Eigensystem data for the ket
!      4          Basis State data for the ket
!      5          File for testing and data output
!      6          Chemical Potential data for the dot (muD)
!      7          Scratch file used for data testing and current output
!      8          Scratch file investigating occupancy and interactions
!***************************************************************************
!  These are the file names to open for the <(N+1)|c_dagger|N> correlations.
!  Here the bra is associated with the (N+1)-particle system, while
!  the ket is associated with the (N)-particle system.  Later in the program
!  these same 'name' variables (used to assign the names of the data files to
!  be opened and used for subsequent calculations) will be reassigned for the
!  <N|c_dagger|(N-1)> correlations.  Also note the name of the data_file,
```

```
!  which is for the (N+1)-to-(N) data.  This will also be changed later for
!  other data set.  As a result there will be two output data files per single
!  run of this program.
!
!  The muD and scratch files never change names during a run once they
!  are opened.
      open(unit=6,file='C:\temp\mu_'//suffix(SPS)//'.dat',
     .    ACCESS='SEQUENTIAL', ACTION='READ', BLANK='NULL',
     .    FORM='FORMATTED', PAD='YES',STATUS='OLD')
      open(unit=7,file='C:\temp\test_file.out',
     .    ACCESS='SEQUENTIAL', ACTION='WRITE', BLANK='NULL',
     .    FORM='FORMATTED', PAD='YES',STATUS='REPLACE')

!  Here we can set the muD for the rest of the run and should not be with
!  the loop that sweeps through particle count.
      read(6,'(f20.14)') muD_BT, muD_T, muD_dielec, muD_medge,
     .                   muD_diagscale
      allocate (muD_array(1:SPS))
      read(6,'(f20.14)') muD_array
      muD = (muD_array(N_target+1)-muD_array(N_target-1))/2
      muL = muD + muL_ratio*((muD_array(N_target+1)-
     .      muD_array(N_target))-muD)
      eGS_target = muD_array(N_target)
!  Initialize the partition function and tunneling current
      I_tunnel = 0
      z = 0
!*************************************************************************
!  This is the outer loop for sweeping particle number from 3 upto a target
!  particle number (N_target) choosen by the user.  The reason this starts at
!  3 is that we don't look at data for systems with 1 particle and the code
!  looks at (N-1) systems and therefore actually looks at (3-1)=> 2 particle
!  systems.
      particle_sweep: do N = 3, N_target
```

```
!  These files change with every N and thus inside the particle_sweep loop
      e_bra_name = 'C:\temp\e_data_'//suffix(N+1)//'_'
     .//suffix(SPS)//'.dat'
      b_bra_name = 'C:\temp\b_state_'//suffix(N+1)//'_'
     .//suffix(SPS)//'.dat'
      e_ket_name = 'C:\temp\e_data_'//suffix(N)//'_'
     .//suffix(SPS)//'.dat'
      b_ket_name = 'C:\temp\b_state_'//suffix(N)//'_'
     .//suffix(SPS)//'.dat'
      data_file =
     .    'C:\temp\'//suffix(SPS)//'_('//suffix(N)//
     .     '+1)_ct_('//suffix(N)//').out'
      open(unit=1,file=e_bra_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=2,file=b_bra_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=3,file=e_ket_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=4,file=b_ket_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
!      open(unit=5,file=data_file,ACCESS='SEQUENTIAL',
!     .    ACTION='WRITE', BLANK='NULL', FORM='FORMATTED',
!     .    PAD='YES',STATUS='REPLACE')
!  Read data from files.  Use the header data in each file to allocate and
!  populate the arrays with data.
      read(1,'(f20.14)') bra_BT, bra_T, bra_dielec, bra_medge,
     .                   bra_diagscale
      read(1,'(3i8)') bra_e_r, bra_e_c, bra_e_z
      allocate (eigen_bra(1 : bra_e_r, 1 : bra_e_c, 0 : bra_e_z))
```

```
      eigen_bra = 0
      read(1,'(f20.14)') eigen_bra

      read(2,'(2i8)') bra_b_r, bra_b_c
      allocate (basis_bra(0:bra_b_r, 1:bra_b_c))
      basis_bra = 0
      read(2,'(i10)') basis_bra

      read(3,'(f20.14)') ket_BT, ket_T, ket_dielec, ket_medge,
     .                   ket_diagscale
      read(3,'(3i8)') ket_e_r, ket_e_c, ket_e_z
      allocate (eigen_ket(1 : ket_e_r, 1 : ket_e_c, 0 : ket_e_z))
      eigen_ket = 0
      read(3,'(f20.14)') eigen_ket

      read(4,'(2i8)') ket_b_r, ket_b_c
      allocate (basis_ket(0:ket_b_r, 1:ket_b_c))
      basis_ket = 0
      read(4,'(i10)') basis_ket

!*************************************************************************
!  At this point we have access to all of the energy and eigensystem data
!  in RAM and is a good point to investigate occupancy and interaction
!  energy behavior as a function of filling factor and confinement
!  potential.  This entire section can be commented out without effecting
!  the rest of the code.  The output of this will be a filed OPENed as
!  UNIT = 8 with the name "occupancy_##_##.dat".
!
!      occupancy_file =
!     .    'C:\temp\occupancy_'//suffix(N)//'_'//suffix(SPS)//'.out'
!      open(unit=8, file=occupancy_file, ACCESS='SEQUENTIAL',
!     .    ACTION='WRITE', BLANK='NULL', FORM='FORMATTED',
!     .    PAD='YES', STATUS='REPLACE')
```

```
!
!  To find the interaction energy (h_int) any given eigenstate we will need]=[
!  the following data for the particular eigenstate in question:
!   -- Confinement potential parameters for MP e-state (medge & daigscale)
!   -- Number of particles []
!   -- Cyclotron energy [erg]
!   -- Coulomb energy [erg] (defined as a [cerg]...energy unit of e-state
!      and confinement energies)
!   -- Energy of the particular MP eigenstate [cerg]
!
!  Note that the KE term = cyclotron_energy(BT)*N)/(2*coulomb_energy(BT))
!  which is explained in the notes and can be seen in use near the end of
!  'diskmain.for'.  This KE term in [cerg] is added to all of the e-state
!  energies (also in {cerg] in the e_data_##_##.dat files.
!
!  See notes for equation to find h_int in units of [cerg].  The only thing
!  note directly available from RAM is confinement energy <W> (see notes)
!  which calculated here.
!
!  Populate the confining potential array W(m).  m runs from 0 to (SPS-1).
!  The potential is zero upto m = medge where it is "turned on", and is
!  thereafter linear in m with a slope of diagscale.
!      allocate (w(0:(SPS-1)))
!      w = 0
!      confinement_loop:  do m = 0,(SPS-1)
!         if (m .le. medge) then
!            W(m) = 0
!         else
!            W(m) = (m - medge) * diagscale
!         end if
!      end do confinement_loop
!  Allocate/initialize the confining potential array 'hint_array' to hold
!  the potentials for all the eigenstates for a system of given SPS and
```

```
!  N.  As it turns out this array will have the same size as that of the
!  eigen_ket array (which is for N-particles here).  Be sure to keep
!  track of the adjustible values in SYS_PARAMETERS so that the results
!  are meaningful and can be compared between runs of different systems.
!
!  The array setup will be similar to that of the eigen_bra and eigen_ket
!  arrays in that the rows and column of hint_array will correspond to
!  the eigenstates of the same row and column indexes.  The z-dimension
!  will start at -1 upto (SPS-1).  The results will be stored in
!  the z = -1 level, while z = 0 and beyond will store the running sums
!  needed to calculate the interactions <W>.
!      allocate (hint_array(1 : ket_e_r, 1 : ket_e_c, -1 : (SPS-1)))
!      hint_array = 0
!  Allocation an array to keep the confinement energy for each eigenstate
!  once calculated so that these can be used in later Hint calculations.
!      allocate (confinement_energy(1 : ket_e_r, 1 : ket_e_c))
!      confinement_energy = 0
!  Loop through all of the eigenstates starting with row 1, column 1 and
!  go down each column (eigenstates with the same mtot).  Rows cycle
!  from 1 upto mtot_state for that column (mtot_state data for any given
!  mtot (column) is stored in row=0 of the basis_ket array.
!      write(8,'('' N = '', i2)') N
!      write(8,'('' KE = '', f8.3)')
!     .                  (cyclotron_energy(BT)*N)/(2*coulomb_energy(BT))
!      confinement_column: do j = 1,ket_e_c
!         confinement_row: do i = 1, basis_ket(0, j)
!  Figure out the running sums of the squared coefficients needed to
!  calculate the confinement energy.  These sums are stored in the
!  hint_array with the x and y corresponding to the given eigenstate and
!  with the z = m running from m=0 to m=mtot_state.  Cycle over all of
!  eigenstates, looking at all the basis states for the given eigenstate.
!  Each basis state is considered to determine the position of the
!  particles (set bits).  The associated m (the particles position)
```

```
!  determines which sum (array position) will be increased.
!           basis_state_loop: do b_state_index = 1, basis_ket(0, j)
!              basis_bit_loop: do m = 0, (SPS-1)
!                 if(BTEST(basis_ket(b_state_index, j),m).eq..true.)then
!                    hint_array(i,j,m) = hint_array(i,j,m) +
!     .                                  eigen_ket(i,j,b_state_index)**2
!                 end if
!              end do basis_bit_loop
!           end do basis_state_loop
!        end do confinement_row
!     end do confinement_column
!  <W> is calculated for all the eigenstates and stored in the
!  confinement_energy(i,j) array where i and j are the same as in the
!  hint_array.  Hint is then calculated and stored in the Z=-1 position of
!  the same i and j (row and column).
!     hint_column: do j = 1,ket_e_c
!        hint_row: do i = 1, basis_ket(0, j)
!           confinement_summing: do m = 0, (SPS-1)
!  Use the following for testing...uncomment the next statement and comment
!  the following one.  The test should return the number of particles (N)
!  for all of the eigenstates.
!              confinement_energy(i,j) = confinement_energy(i,j) +
!     .                                          hint_array(i,j,m)
!              confinement_energy(i,j) = confinement_energy(i,j) +
!     .                                     w(m)*hint_array(i,j,m)
!              write(8,'(''h_int('',i2,'','',i2,'', ''i2,'')='',
!     .          f8.3)') i,j,m,hint_array(i,j,m)
!           end do confinement_summing
!           hint_array(i,j,-1) = (eigen_ket(i,j,0) -
!     .            (cyclotron_energy(BT)*N)/(2*coulomb_energy(BT))) -
!     .            confinement_energy(i,j)
!           write(8,'(''confinement('',i2,'','',i2,'') ='',
!     .       f8.3,''     h_int('',i2,'','',i2,'', -1) ='',
```

```
!     .       f8.3)') i,j,confinement_energy(i,j),i,j,hint_array(i,j,-1)
!         end do hint_row
!      end do hint_column
!      deallocate(w,hint_array,confinement_energy)
!      close(UNIT=8, STATUS='KEEP')
!*************************************************************************
!  Here we're going to determine the partition function which includes
!  an offset discribed in my notes.  This offset is realized in this code
!  as the function 'stat_shift'.  We are going to look at the energies of
!  the N-particle system which are the |ket> energies at this point in the
!  code.  All of the energies for N=2 thru (N_target + 1) are included in
!  the sum.  Since we will already have all of this data in memory we will
!  look at each N-particle system as they go by and sum as we go.  The way
!  that the loop is indexed (starting at N = 3 to N_target) will require
!  we get the N = 2 energies from the second half of the code (the |ket>
!  for N = 3).  While the (N_target + 1) energies will come from this half
!  of the code (the <bra| for N = N_target).  This will be accomplished
!  using if-statements.  For all the remaining energy sets, the |ket> from
!  this half will be used (for N = 3 thru N_target) with no need for
!  special filtering statements.
!
!  Here's the filter for the energies associated with N = (N_target + 1).
      if (N.eq.N_target)then
         N_plus_part_col: do j = 1, bra_e_c
            N_plus_part_row: do i = 1, basis_bra(0,j)
               z = z + part_function(eigen_bra(i,j,0), muD, (N+1))*
     .                          stat_shift(eGS_target, muD, N_target)
!              write(7,'(''energy = '',ES11.5,''     running z sum = '',
!     .             ES11.5)')   eigen_bra(i,j,0),z
            end do N_plus_part_row
         end do N_plus_part_col
      endif
!  This is the regular energy loop for N particle systems
```

```
      partition_col: do j = 1, ket_e_c
         partition_row: do i = 1, basis_ket(0,j)
            z = z + part_function(eigen_ket(i,j,0), muD, N)*
     .                        stat_shift(eGS_target, muD, N_target)
!           write(7,’(’’energy = ’’,ES11.5,’’     running z sum = ’’,
!     .           ES11.5)’)   eigen_ket(i,j,0),z
         end do partition_row
      end do partition_col

!  Before we go any further we’ll check to make sure that the e_data files
!  for the bra and the ket as well as muD are based on the same parameters
!  as are in SYS_PARAMETERS.  If there are differences with the parameters
!  then stop and say so.
      if (ket_BT.ne.BT .or. bra_BT.ne.BT .or. muD_BT.ne.BT .or.
     .ket_T.ne.T .or. bra_T.ne.T .or. muD_T.ne.T
     ..or.ket_dielec.ne.dielec .or. bra_dielec.ne.dielec .or.
     .muD_dielec.ne.dielec .or. ket_medge.ne.medge .or.
     .ket_medge.ne.medge .or.
     .bra_medge.ne.medge .or. muD_medge.ne.medge .or.
     .ket_diagscale.ne.diagscale .or. bra_diagscale.ne.diagscale .or.
     .muD_diagscale.ne.diagscale) then
         stop ’file mismatch - system parameter error’
      endif

!  This array is where the results of both calculations are stored.
      allocate (corr_array(0:(SPS-1),1:3))
      corr_array = 0.0

!  Calculated phi^2 for all m’s for this run.  These will be stored in
!  the first column of corr_array to be used in subsequent calculations.
      phi_loop: do m = 0,(SPS-1)
         corr_array(m,1) = wave(r,m)**2
      end do phi_loop
```

```
!*****************************************************************************
!  The following general data is written to test_file for verification and
!  testing.  This can be commented out if testing is not being done.
      write(7,'(''        pi = '', ES24.15,/
     .''     medge = '', ES24.15,/
     .'' diagscale = '', ES24.15,/
     .''        BT = '', ES24.15,/
     .''         T = '', ES24.15,/
     .''    dielec = '', ES24.15,/
     .''        kb = '', ES24.15,/
     .''      hbar = '', ES24.15,/
     .''        me = '', ES24.15,/
     .''         e = '', ES24.15,/
     .''         c = '', ES24.15,/
     .''   meffect = '', ES24.15,/
     .''         r = '', ES24.15,/
     .''       muL = '', ES24.15,/
     .''       muD = '', ES24.15)')
     . pi,medge,diagscale,BT,T,dielec,kb,hbar,me,e,c,meffect,r,muL,muD
      write(7,*)
      write(7,'(''magnetic_length (BT) = '', ES12.5,/
     .''cyclotron_energy(BT) = '', ES12.5,/
     .''  coulomb_energy(BT) = '', ES12.5,/
     .''         beta(BT, T) = '', ES12.5)')
     . magnetic_length (BT),cyclotron_energy(BT),coulomb_energy(BT),
     . beta(BT, T)
!      write(7,*)
!      wave_test: do i = 0, (SPS-1)
!      write(7,'(''wave(r = '',ES11.5,'',m = '',i2,'') = '',ES11.5)')
!     .   r,i,wave(r,i)
!      end do wave_test
!      write(7,*)
```

```
!*****************************************************************************
!  Visit every basis state of every MPS of the bra.  The eigensystem data
!  for each MPS of the bra is contained in the eigensystem array,
!  (eigen_bra), and corresponding basis states for each of the MPS will
!  be looked at in an ascending order.  This will be done by looping through
!  the basis state array for the bra, basis_bra(row,column).  The columns
!  will be looped over on the outside and the rows on the inside such that
!  all of the basis states for a given column (mtot) will be considered from
!  1 to the 'mtot_state' for that column (mtot_state for each column is stored
!  row zero of the corresponding column).
!
!  There will be 'mtot_state' basis states (with the same SPS, N+1 and mtot).
!  Each of these will be considered in an ascending fassion from the basis
!  state stored in row=1 upto row=mtot_state.  The row index is how these
!  basis states will be enumerated (i.e. for given SPS/N+1 and mtot the basis
!  states within that column will have the same mtot and the state stored in
!  row 6 is 'basis state 6' for SPS/N+1/mtot.  For each of these (N+1) basis
!  states in the bra there will be N+1 corresponding ket basis states (with
!  SPS/N/mtot lesser than or equal to the bra's mtot).  These corresponding
!  ket basis states are the states that have non-zero overlap integrals and
!  are found by removing one of the existing bra's particles while keep all
!  of the rest.  Since there are N+1 particles to be removed there will be
!  N+1 corresponding N particle basis states in the ket.
!
!  Each SPS of bra's basis states are checked to see if there is a particle
!  (set bit) starting from the least significant upto the most significant
!  bit (or until N+1 particles are found).  Once a particle is found, the
!  particle is removed (the SPS-bit is cleared) and the result represents the
!  ket basis state (with SPS and N particles) with an mtot that is less by the
!  angular momentum of the particle removed.  This ket basis state is used
!  by all of the ket MPS for the resulting SPS/N/(reduced mtot) and therefore
!  we get non-zero contributions from all of the MPS in the corresponding
!  column.  This 'reduced mtot' column of the ket is looped through and
```

```
!  the constant from each ket MPS for the given ket basis state and the
!  constant from the bra basis state are multiplied and further calculations
!  are performed.

      bra_column: do j = 1,((maxmtot(sps,(n+1))-minmtot(sps,(n+1))) + 1)
         bra_row: do i = 1, basis_bra(0,j)
            bra_basis_state = basis_bra(i,j)
            bra_mtot = minmtot(sps,(n+1)) + j - 1
!  The bit count used to determine the sign of the result is initialize before
!  looking at the bits of each bra basis state
            bit_count = 0
            bra_bit_loop: do k = 1, SPS
!  k represents the SPS being tested to see if it contains a particle (set
!  bit) or not (clear bit).  Each bit is checked in succession from 1 up to
!  the SPS bit.  If a particle (set bit) is found it is clear (that particle
!  is annihilated) and the calculation to find the contribution to the
!  tunneling current is carried out.

!  Now it is natural (at least to me) to start counting states at 1 and end
!  at SPS, but a more useful way of counting would be to start at 0 upto
!  SPS-1.  This is because that index would correspond to the way angular
!  momentum (m) of the individual SPS is counted.
!
!  m is the angular momentum of the particle being removed from the bra
!  basis state to get the ket basis state.  These angular momentum values
!  start with m = 0, or one less than the value of k and is why we have the
!  following expression.  Also note that BTEST and IBCLR (both are intrinsic
!  bit level functions) consider the right most or least significant bit
!  to be bit zero (0) which is the same counting scheme as m.
               m = k - 1
               if (BTEST(bra_basis_state,m).eq..true.) then
                  ket_basis_state = IBCLR(bra_basis_state, m)
!  ket_mtot is equal to the mtot of the bra minus the angular momentum of
```

```
!  the particle that was removed, m.
                ket_mtot = bra_mtot - m
!  Now we find the position of the ket basis state in the basis_ket array
!  by searching in column ket_mtot and record the row index containing
!  ket_basis_state.  The ket row index is used to enumerate the basis states
!  and will be used to find the associated data stored in the eigensystem
!  array.
                ket_column = ket_mtot - minmtot(sps,n) + 1
                ket_row = 1
!  A running total of the number of set bits found is incremented each time
!  another bit is found.  This is done starting at the right-most bit (least
!  significant bit) such that the first bit found give bit_total = 1, the
!  next bit will give 2 and so on.  This is used to alternate the sign of
!  the resulting product of the eigenstate coefficients.
                bit_count = bit_count + 1

                do while (ket_basis_state.ne.
     .                          basis_ket(ket_row,ket_column))
                   ket_row = ket_row + 1
                end do

!  'sign' that gives the sign of the resulting valid eigenstate overlaps.
                sign = (-1)**(bit_count - 1)

!*************************************************************************
!  write the intermediate results to a file for checking purposes.
!  The following write statements should be commented out when checking of
!  this data is not required.
!                 write(5,'(''-------------------------'')')
!                 write(5,'(''bit='',i2,'' m='',i2,'' bram='',
!     .                i2,'' ketm='',i2)')k, m, bra_mtot, ket_mtot
!                 write(5,'(''bra_basis_state('',i2,'','',i2,'') ='',i2)
!     .                ') i,j,bra_basis_state
```

```
!                  write(5,'(''ket_basis_state('',i2,'','',i2,'') ='',i2)
!     .                  ') ket_row, ket_column, ket_basis_state
!                  write(5,'(''sign ='',i2)') sign
!                  write(5,'(''wave ='',e8.3)') wave(r,m)
!*****************************************************************************
!  Each ket basis state corresponds to a number of eigenvalues stored in
!  the eigensystem array ('mtot_state' in fact).  The row index of the ket
!  basis states (ket_row) gives the z dimension of the eigensystem array
!  of these eigenvalues for the same column index.
!
!  For example, given a system with SPS=5/N=2 the basis state |DECNDX = 6>
!  has an mtot of 3 and it is location indecies in the basis state array
!  are ket_row=2 and ket_column=3.  In other words it was in the mtot=3
!  column and it was the '2nd' basis state using our method of enumeration.
!  Looking at ket_row=0 of ket_column=3 in the basis state array will give
!  the mtot_state for this column...in this case mtot_state=2.  This tells
!  us that there are not only 2 basis states with SPS=5/N=2/mtot=3, but
!  due the real symetric nature of the Hamiltonian there are also 2
!  eigenvalues that correspond to this basis state AND (because of the way
!  we've stored this data in the eigensystem array) they are located at
!  a z-index of 2 in the same column (in this case 3).  Note that the data
!  stored in z-index of 1 and the same column corresponds to the '1st' basis
!  state which for SPS=5/N=2/mtot=3 is |DECNDX=9>...the reason this is
!  mentioned is simply to help with orientation with the arrays so that
!  it's clear which eigenvalues go with which basis states.
!
!  To provide some way of checking that the code is functioning as intended
!  and the correct eigenvalues are being selected the following will get the
!  data for each 'valid' pair of bra and ket, multiply them together (as this
!  will eventually be done) and store them.
                  valid_bra_loop: do x = 1, basis_bra(0, j)
                     valid_ket_loop: do y = 1, basis_ket(0, ket_column)
                        bra_energy = eigen_bra(x, j, 0)
```

```
                        ket_energy = eigen_ket(y, ket_column, 0)
                        e_diff = bra_energy - ket_energy
                        if (e_diff.gt.0.0) then
                            corr_A = f(e_diff, muL)*
     .                         stat_shift(eGS_target, muD, N_target)*
     .                         thermal(ket_energy, muD, N)*
     .                         (eigen_bra(x, j, i)*
     .                         eigen_ket(y, ket_column, ket_row))**2
                           corr_array(m,2) = corr_array(m,2) + corr_A
!*************************************************************************
!  The following code (between the stars) is for testing and verification
!  and should be commented out if this is no longer required.
!
!  The coefficient associated with the valid bra (these coefficients come from
!  the energy eigenvectors) is located at eigen_bra(x, j, i).
!  The coefficient associated with the valid ket (these coefficients come from
!  the energy eigenvectors) is located at eigen_ket(y, ket_column, ket_row).
!  There is another bit of code earlier in this file that also is used for
!  testing (look for unit 5 write statements) and should be commented out when
!  not in use as well.
!                           write(5,''('' bra='',f8.3,'' ket='',
!     .                        f8.3)'') eigen_bra(x, j, i),
!     .                        eigen_ket(y, ket_column, ket_row)
!                           write(7,''('' f(e_diff = '',f5.3,'') = '',
!     .                         ES12.5,'' or rounded '',i2)'')
!     .                         e_diff, f(e_diff, muL),
!     .                         NINT(f(e_diff, muL))
!          write(7,''('' thermal(ket_energy = '',f5.3,'', muD, N = ''
!     .         ,i2,'') =       '',ES12.5,'' or rounded '',i8)'')
!     .         ket_energy, N, thermal(ket_energy, muD, N),
!     .         NINT(thermal(ket_energy, muD, N))
!*************************************************************************
                        endif
```

```
                    end do valid_ket_loop
                  end do valid_bra_loop
               endif
            end do bra_bit_loop
         end do bra_row
      end do bra_column

!*************************************************************************
!*************************************************************************
!*************************************************************************
!  This ends the first part which works with <(N+1)|c_dagger|N> and begins the
!  second which works with <N|c_dagger|(N-1)>.
!  This is certainly NOT elegant and a better programmer would have done this
!  with half as many lines, however it is clear and easily read/debugged.
!*************************************************************************
!*************************************************************************
!*************************************************************************
!  Get rid of the data currently in the bra and ket arrays.  There is a chance
!  to speed things along since the N-particle data is used again.  However,
!  it is used in the ket for the first part and then for the bra in the second.
!  For now it's cleaner and clearer to simply get rid of it in the ket and
!  reassign it to the bra so that the rest of the code stays the same.
      deallocate(eigen_bra, basis_bra, eigen_ket, basis_ket)
!  Close out the current files.
      close(UNIT=1, STATUS='KEEP')
      close(UNIT=2, STATUS='KEEP')
      close(UNIT=3, STATUS='KEEP')
      close(UNIT=4, STATUS='KEEP')
!      close(UNIT=5, STATUS='KEEP')

!  Modify the file name variables for the <N|c_dagger|(N-1)> correlations and
!  open the associated files.  This time the N-particle system is the bra and
!  the (N-1)-particle system is the ket.
```

```
      e_bra_name = 'C:\temp\e_data_'//suffix(N)//'_'
     .//suffix(SPS)//'.dat'
      b_bra_name = 'C:\temp\b_state_'//suffix(N)//'_'
     .//suffix(SPS)//'.dat'
      e_ket_name = 'C:\temp\e_data_'//suffix(N-1)//'_'
     .//suffix(SPS)//'.dat'
      b_ket_name = 'C:\temp\b_state_'//suffix(N-1)//'_'
     .//suffix(SPS)//'.dat'
      data_file =
     .    'C:\temp\'//suffix(SPS)//'_('//suffix(N)//
     .     ')_ct_('//suffix(N)//'-1).out'
      open(unit=1,file=e_bra_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=2,file=b_bra_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=3,file=e_ket_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
      open(unit=4,file=b_ket_name,ACCESS='SEQUENTIAL',
     .    ACTION='READ', BLANK='NULL', FORM='FORMATTED',
     .    PAD='YES',STATUS='OLD')
!      open(unit=5,file=data_file,ACCESS='SEQUENTIAL',
!     .    ACTION='WRITE', BLANK='NULL', FORM='FORMATTED',
!     .    PAD='YES',STATUS='REPLACE')

!  Read data from files.  Use the header data in each file to allocate and
!  populate the arrays with data.
      read(1,'(f20.14)') bra_BT, bra_T, bra_dielec, bra_medge,
     .                   bra_diagscale
      read(1,'(3i8)') bra_e_r, bra_e_c, bra_e_z
```

```
      allocate (eigen_bra(1 : bra_e_r, 1 : bra_e_c, 0 : bra_e_z))
      read(1,'(f20.14)') eigen_bra

      read(2,'(2i8)') bra_b_r, bra_b_c
      allocate (basis_bra(0:bra_b_r, 1:bra_b_c))
      read(2,'(i10)') basis_bra

      read(3,'(f20.14)') ket_BT, ket_T, ket_dielec, ket_medge,
     .                   ket_diagscale
      read(3,'(3i8)') ket_e_r, ket_e_c, ket_e_z
      allocate (eigen_ket(1 : ket_e_r, 1 : ket_e_c, 0 : ket_e_z))
      read(3,'(f20.14)') eigen_ket

      read(4,'(2i8)') ket_b_r, ket_b_c
      allocate (basis_ket(0:ket_b_r, 1:ket_b_c))
      read(4,'(i10)') basis_ket

!  Here's the filter for the energies associated with N = 2.  Remember that
!  the |ket> energies are for a system with (N-1) particles (therefore the
!  following if filter which will result in (3-1)=2 particle energies
      if (N.eq.3)then
         N_minus_part_col: do j = 1, ket_e_c
            N_minus_part_row: do i = 1, basis_ket(0,j)
               z = z + part_function(eigen_ket(i,j,0), muD, (N-1))*
     .                          stat_shift(eGS_target, muD, N_target)
!              write(7,'(''energy = '',ES11.5,''     running z sum = '',
!     .            ES11.5)')   eigen_ket(i,j,0),z
            end do N_minus_part_row
         end do N_minus_part_col
      endif

!  Another check to make sure that the e_data files and muD parameters
!  with those of SYS_PARAMETERS.  If there are differences then stop and say so.
```

```
       if (ket_BT.ne.BT .or. bra_BT.ne.BT .or. muD_BT.ne.BT .or.
      .ket_T.ne.T .or. bra_T.ne.T .or. muD_T.ne.T .or.
      .ket_dielec.ne.dielec .or. bra_dielec.ne.dielec .or.
      .muD_dielec.ne.dielec .or. ket_medge.ne.medge .or.
      .ket_medge.ne.medge .or.
      .bra_medge.ne.medge .or. muD_medge.ne.medge .or.
      .ket_diagscale.ne.diagscale .or. bra_diagscale.ne.diagscale .or.
      .muD_diagscale.ne.diagscale) then
          stop 'file mismatch - system parameter error'
       endif

!  This is a repeat of the previous code with the following exceptions:
!  -- loop names which have a _2 suffix to distinguish them from those above
!  -- 'n+1' changed to 'n' three different places...twice in the bra_column_2
!     line and once in the bra_mtot line.
!  -- 'n' changed to 'n-1'...once in the ket_column line

      bra_column_2: do j = 1,((maxmtot(sps,(n))-minmtot(sps,(n)))+1)
         bra_row_2: do i = 1, basis_bra(0,j)
            bra_basis_state = basis_bra(i,j)
            bra_mtot = minmtot(sps,(n)) + j - 1
            bit_count = 0
            bra_bit_loop_2: do k = 1, SPS
               m = k - 1
               if (BTEST(bra_basis_state,m).eq..true.) then
                  ket_basis_state = IBCLR(bra_basis_state, m)
                  ket_mtot = bra_mtot - m
                  ket_column = ket_mtot - minmtot(sps,(n-1)) + 1
                  ket_row = 1
                  bit_count = bit_count + 1
                  do while (ket_basis_state.ne.
     .                           basis_ket(ket_row,ket_column))
                     ket_row = ket_row + 1
```

```
                end do
                sign = (-1)**(bit_count - 1)
!                write(5,'(''-------------------------'')')
!                write(5,'(''bit='',i2,'' m='',i2,'' bram='',
!     .               i2,'' ketm='',i2)')k, m, bra_mtot, ket_mtot
!                write(5,'(''bra_basis_state('',i2,'','',i2,'') ='',i2)
!     .               ') i,j,bra_basis_state
!                write(5,'(''ket_basis_state('',i2,'','',i2,'') ='',i2)
!     .               ') ket_row, ket_column, ket_basis_state
!                write(5,'(''sign ='',i2)') sign
!                write(5,'(''wave ='',E8.3)') wave(r,m)
                valid_bra_loop_2: do x = 1, basis_bra(0, j)
                   valid_ket_loop_2: do y = 1,basis_ket(0, ket_column)
                      bra_energy = eigen_bra(x, j, 0)
                      ket_energy = eigen_ket(y, ket_column, 0)
                      e_diff = bra_energy - ket_energy
                      if (e_diff.gt.0.0) then
                          corr_B = f(e_diff, muL)*
     .                        stat_shift(eGS_target, muD, N_target)*
     .                        thermal(bra_energy, muD, N)*
     .                        (eigen_bra(x, j, i)*
     .                        eigen_ket(y, ket_column, ket_row))**2
                          corr_array(m,3) = corr_array(m,3) + corr_B
!                          write(5,'('' bra='',f8.3,'' ket='',6
!     .                        f8.3)') eigen_bra(x, j, i),
!     .                        eigen_ket(y, ket_column, ket_row)
!                          write(7,'('' f(e_diff = '',f5.3,'') = '',
!     .                         ES12.5,'' or rounded '',i2)')
!     .                         e_diff, f(e_diff, muL),
!     .                         NINT(f(e_diff, muL))
                      endif
                   end do valid_ket_loop_2
                end do valid_bra_loop_2
```

```
                endif
              end do bra_bit_loop_2
            end do bra_row_2
         end do bra_column_2

!  Dump data from corr_array to output file and then get rid of the data
!       write(7,'(ES12.5)') corr_array
!       write(7,'(''-------------------------'')')

         current_sum_row: do i = 0 , (SPS-1)
            I_tunnel = I_tunnel + corr_array(i,1)*
     .                           (corr_array(i,2)-corr_array(i,3))
         end do current_sum_row
         deallocate(eigen_bra, basis_bra, eigen_ket, basis_ket, corr_array)
!  Close out the current files.
         close(UNIT=1, STATUS='KEEP')
         close(UNIT=2, STATUS='KEEP')
         close(UNIT=3, STATUS='KEEP')
         close(UNIT=4, STATUS='KEEP')
!       close(UNIT=5, STATUS='KEEP')

         end do particle_sweep

         close(UNIT=6, STATUS='KEEP')

!  Calculate the tunneling current and write to output file
         write(7,'(ES12.5)') I_tunnel
         close(UNIT=7, STATUS='KEEP')
         end
```

APPENDIX G
NUMERICAL ANALYSIS CODE: SYS_ PARAMETERS.F

```
!************************************************************************
!  MODULE: SYS_PARAMETERS
!
!  PURPOSE: Establishes the arrays used during calculation and storage of the
!     Multi-Particle State (MPS) data.
!
!           Establishes variables that can be used by other program units and
!     while keeping them centrally located for easy modification.
!
!           Provides a centralized 'library' of functions that are useful
!     when counting bits, decimal numbers, binary numbers, etc.
!************************************************************************
module SYS_PARAMETERS
implicit none
!************************************************************************
!  Assigned Parameters:
!     medge/diagscale - the single-particle potential begins at m = medge and
!                       is thereafter linear in m with a coefficient (in units
!                       of V1) equal to diagscale.
!                  BT - Magnetic Induction Field [T]
!                   T - Temperature [K]
!              dielec - Dielectric Constant []
!
!  Constants:
!                  kb - Boltzmann's Constant [erg / K]
!                hbar - Planck's Constant [erg * s]
!                  me - Electron Mass [g]
!                   e - Electron Charge [esu]
!                   c - Speed of Light [cm / s]
!             meffect - Effective Mass [g]
!                   r - location of tip from center of dot, "radius" [cm]
!                muL - Chemical Potential of lead [erg]
!                muD - Chemical Potential of dot [erg]
```

```
!*************************************************************************
      real*8, parameter ::        pi = 3.14159265358979D0
      real*8, parameter ::     medge = 6.0D0
      real*8, parameter :: diagscale = 0.1D0

      real*8, parameter ::        BT = 1.0D0
      real*8, parameter ::         T = 1.0D0

      real*8, parameter ::        kb = 1.3807D-16
      real*8, parameter ::      hbar = 1.0546D-27
      real*8, parameter ::        me = 9.1095D-28
      real*8, parameter ::         e = 4.8032D-10
      real*8, parameter ::         c = 2.9979D10
      real*8, parameter ::    dielec = 12.6D0

      real*8 :: meffect = .067D0 * me
      real*8 :: r = 1.3750D-5
      real*8 :: muL_ratio = 0.1D0

end module SYS_PARAMETERS
```

APPENDIX H
NUMERICAL ANALYSIS CODE: DECLARATIONS.F

```
module DECLARATIONS
!***************************************************************************
!  MODULE: DECLARATIONS
!
!  PURPOSE: Establishes the arrays used during calculation and storage of the
!     Multi-Particle State (MPS) data.
!
!           Establishes variables that can be used by other program units and
!     while keeping them centrally located for easy modification.
!
!           Provides a centralized 'library' of functions that are useful
!     when counting bits, decimal numbers, binary numbers, etc.
!***************************************************************************
    use SYS_PARAMETERS

implicit none
!***************************************************************************
!  These are variables used throughout the entire program.
!
!  SPS - 'Single Particle States' which is the number of single particle
!       states (input by the user) used as the basis for the Multi-Particle
!       States (MPS).  Counting of this variable starts with 1 (opposed to 0).
!       These SPS are arranged as binary digits, in order from less
!       significant to most significant, to compose the MPS.  The MPS, which
!       is a composit of ones and zeros, can be thought of as a binary number
!       and referenced using the decimal equivalent of the binary.
!
!  MIDSPS - an integer value equal to half the SPS value.
!       For even SPS - division by two gives another integer.
!       For odd SPS - the value is rounded to the next lowest integer.
! It should be noted that MIDSPS is used to determine the the largest
! number of MPS for a particular run of cases.  As a result of the
! arrangement used to store the MPS in the data array (all MPS in a
```

```
!       column correspond to cases with the same number of particles), the
!       number MPS for any column is the combinatoric value (SPS choose
!       BITTOT).  As a result of the combinatoric nature of these systems
!       and how the data is stored in the data array, the centeral column(s)
!       of the array will always contain the largest number of MPS.
!       Therefore the value of ('SPS' choose 'MIDSPS') is used to determine
!       the minimum size of the arrays required to store the MPS data.  The
!       rounding down of MIDSPS (opposed to up) is arbitrary since the
!       binomial coefficient is the same for both rounding cases.  Just to
!       be consistant throughout the code I've choosen to always round down.
!
!       Example for even SPS - SPS=8 the maximum number of MPS is
!           (8 choose 4) = 70
! Example for odd SPS - SPS=7 the maximum number of MPS is either
! (7 choose 3) = (7 choose 4) = 35
!*****************************************************************************
!  MAXDEC - integer value equal to the maximum decimal for an MPS with given
!       number of SPS (decimal value if all MPS bits are set to ones).  For
!       example, if SPS=4, then the largest decimal that corresponds is
!       (2^0)+(2^1)+(2^2)+(2^3) = 1 + 2+ 4 + 8 = 15.  Note that the first SPS
!       is associated with (2^0), the second SPS is associated with
!       (2^1), etc.
!
!  BITTOT - total number of set bits in MPS (set bits represent filled SPS).
!
!  MTOT - total angular momentum of an MPS (word of caution...MTOT is
!       calculated using loops so keep this in mind when polling it's value)
!
!  H_size -  sizes the hamiltonian array (H) and associated arrays used for
!       solving the eigensystem (calls to RSP routine).  It uses the equation
!       found in the comments of RSP to give the number of elements for a
!       symmetric lower-triangle packed array.  The symmetric array is sized
!       to just accommodate the number of basis states with the same mtot
```

```
!       (variable mtot_state) for the current loop.  This value changes as
!       the loop increments and keeps the memory allocated for the Hamiltonian
!       to a minimum.
!
!***************************************************************************
integer :: SPS, MIDSPS, MAXDEC, BITTOT, MTOT, H_SIZE

      character*2 suffix(30)/'01','02','03','04','05','06','07','08',
     . '09','10','11','12','13','14','15','16','17','18','19','20',
     . '21','22','23','24','25','26','27','28','29','30'/

      real*8, allocatable, dimension(:,:,:) :: e_data_array
      real*8, allocatable, dimension(:,:) :: ne_data_array
      real*8, allocatable, dimension(:) :: mu_data_array
      integer, allocatable, dimension(:,:) :: b_state_array
      real*8, save, allocatable, dimension(:) :: Vpseudo
      real*8, save, allocatable, dimension(:,:,:) :: CC
      real*8, save, allocatable, dimension(:,:) :: mbasis
      real*8, save, allocatable, dimension(:) :: W
      real*8, save, allocatable, dimension(:) :: Welec
      real*8, save, allocatable, dimension(:) :: evalues
      real*8, save, allocatable, dimension(:) :: e_total
      real*8, save, allocatable, dimension(:,:) :: evectors
      real*8, save, allocatable, dimension(:) :: H
      real*8, save, allocatable, dimension(:) :: muD_array
      real*8, save, allocatable, dimension(:,:,:) :: hint_array
      real*8, save, allocatable, dimension(:,:) :: confinement_energy

      contains
!***************************************************************************
!     These are functions used throughout the program
!  COMBIN(N,M) - 'N choose M'
!  FACTORIAL(N) - N!
```

```
!  minmtot(sps,n) - minimum Total Angular Momentum (TAM)
!  maxmtot(sps,n) - maximum TAM
!        hilbertspace(sps,n) -  size of the entire Hilbert space (includes
!                               all Mtot) and is (SPS choose N).
!        stack_dim(sps) - maximum number of MPS for given SPS/N/mtot (which
!                         occurs for N = SPS/2
!*************************************************************************
         pure real*8 function combin(n,m)
!*************************************************************************
!  Computes "n choose m"
!  The input to the function is 'n' and 'm' and the combinatorial number
!  (also known as 'combination' or 'binomial coefficient') is returned.
!  The intermediate real value 'com' is required in order to perform
!  the division operation.  In the end, the returned result will always
!  be an integer and therefore the last step converts the real to an
!  integer before combin is returned.
!*************************************************************************
            implicit none
            integer, intent(in) :: n,m
            integer :: i,mm
!            real*8 factorial
            combin=1.0
            if (m.eq.0) return
            mm = min(m,n-m)
            combin_loop:  do i = n-mm+1,n
               combin = combin*i
            end do combin_loop
            combin = combin/factorial(mm)
            return
         end function combin

         pure real*8 function factorial(n)
!*************************************************************************
```

```
!  Computes n!
!  The input to the function is 'n' and factorial is returned
!**************************************************************************
implicit none
integer, intent(in) :: n
integer :: i
factorial = 1.0
if (n .le. 1) return
factorial_loop:  do i = 2,n
               factorial = factorial*i
end do factorial_loop
return
 end function factorial

 pure function hilbertspace(sps,n)
!**************************************************************************
!        Computes the total size of the Hilbert space for the system in
!        question given SPS and N.  This is the combinatoric (SPS choose N).
!**************************************************************************
    implicit none
    integer, intent(in) :: sps, n
            integer :: hilbertspace
            hilbertspace = factorial(sps)/
     .                                  (factorial(N)*factorial(sps-N))
            return
         end function hilbertspace

 pure function minmtot(sps,n)
!**************************************************************************
!  Computes the minimum TAM for our system.  The inputs to this function
!  are the number of Single Particle States (SPS) and the number of
!  particles 'n'.  Note that SPS starts counting as SPS => 1,2,3,... but
!  correspond to the angular momentum => 0,1,2,...
```

```
!
!        Looking at this as a series what this returns is,
!
!              minmtot = 0 + 1 + 2 + ... + (N-1)
!*****************************************************************************
  implicit none
  integer, intent(in) :: sps, n
  integer :: minmtot, i
  minmtot = 0
  do 30 i = 0, (n-1)
30   minmtot = minmtot + i
  return
 end function minmtot

 pure function maxmtot(sps,n)
!*****************************************************************************
!  Computes the maximum TAM for our system.  The inputs to this function
!  are the number of Single Particle States (SPS) and the number of
!  particles 'n'.  Note that SPS starts counting as SPS => 1,2,3,... but
!  correspond to the angular momentum => 0,1,2,...
!
!        Looking at this as a series what this returns is (noting that
!        SPS = Mouter + 1),
!
!              maxmtot = Mouter + (Mouter-1) + ... + (Mouter-(N-1))
!                                      -or-
!              maxmtot = (SPS-1) + ((SPS-1)-1) + ... + ((SPS-1)-(N-1))
!*****************************************************************************
    implicit none
    integer, intent(in) :: sps, n
    integer :: maxmtot, i
    maxmtot = 0
    maxmtot_loop:  do i = (sps - 1) , (sps - n) , -1
```

```
         maxmtot = maxmtot + i
            end do maxmtot_loop
    return
         end function maxmtot

   pure function magnetic_length (BT)
!*****************************************************************************
!  Computes the magnetic length of our system.  The argument of this
!        function is the magnetic induction field, BT,  in units of Tesla [T]
!        of our system.
!*****************************************************************************
    implicit none
        real*8, intent(in) :: BT
    real*8 :: magnetic_length
    magnetic_length = ((hbar * c)/(e * BT * 10000))**.5
    return
         end function magnetic_length

   pure function cyclotron_energy(BT)
!*****************************************************************************
!  Computes the cyclotron energy, hbar*omega_c, of our system.  The
!        argument of this function is the magnetic induction field, BT,  in
!        units of Tesla [T] of our system.
!*****************************************************************************
    implicit none
    real*8, intent(in) :: BT
    real*8 :: cyclotron_energy
    cyclotron_energy = (hbar * e * BT * 10000)/(meffect * c)
    return
         end function cyclotron_energy

   pure function coulomb_energy(BT)
!*****************************************************************************
```

```
!  Computes the coulomb energy of our system.  The argument
!        of this function is the magnetic induction field, BT,  in units of
!        Tesla [T] of our system.
!***************************************************************************
    implicit none
    real*8, intent(in) :: BT
    real*8 :: coulomb_energy
    coulomb_energy = (e**2)/(dielec * magnetic_length(BT))
         end function coulomb_energy

   pure function wave(r, m)
!***************************************************************************
!  Computes the Lowest Landau Level wave function for a dot (with
!        disk geometry).  The arguments of this function are:
!           r - distance of particle from center of dot
!           m - angular momentum of particle in question
!***************************************************************************
    implicit none
    real*8, intent(in) :: r
    integer, intent(in) :: m
    real*8 :: wave
    wave = (2 * pi * magnetic_length (BT)**2 * 2**m * factorial(m))
     .             **(-.5) * (r/magnetic_length(BT))**m *
     .             exp(-(abs(r)**2)/(4 * magnetic_length(BT)**2))
         end function wave

   pure function f(e_diff, muL)
!***************************************************************************
!  Computes a weighting function
!           e_diff - energy argument of the function which here is the energy
!                    difference between the energy of the bra and ket states
!              muL - Chemical Potential of the lead
!***************************************************************************
```

```
    implicit none
    real*8, intent(in) :: e_diff, muL
    real*8 :: f
            f = (exp(beta(BT, T)*(e_diff - muL)) + 1)**-1
         end function f

         pure function thermal(eNn, muD, N)
!************************************************************************
!  Computes the thermal weighting function associated with the chemical
!        potential of the dot
!              eNn - energy of the |N,n> state...be careful to make sure
!                    that the correct energy is passed and not that of the
!                    |N+1,n'> or |N-1,n''> states.
!              muD - Chemical Potential of the dot
!                N - number of particles in the MPS
!************************************************************************
    implicit none
    real*8, intent(in) :: eNn, muD
    integer, intent(in) :: N
    real*8 :: thermal
            thermal = (exp(beta(BT, T)*(eNn - muD*N)))**-1
         end function thermal

         pure function stat_shift(eGS_target, muD, N_target)
!************************************************************************
!  Computes a term used to shift the summed statistical terms in both
!        numerator and denominator (partition function).  This done for
!        numerical reasons to allow the computer to handle the size of the
!        numbers involved.
!              eGS_target - ground state energy (lowest value) for the system
!                           with the given SPS N_target particles
!                   muD - Chemical Potential of the dot
!               N_target - target number of particles in the MPS
```

```
!*************************************************************************
    implicit none
    real*8, intent(in) :: eGS_target, muD
    integer, intent(in) :: N_target
    real*8 :: stat_shift
            stat_shift = exp(beta(BT, T)*(eGS_target - muD*N_target))
         end function stat_shift

         pure function part_function(E_energy, muD, N)
!*************************************************************************
!  Computes a term used to shift the summed statistical terms in both
!        numerator and denominator (partition function).  This done for
!        numerical reasons to allow the computer to handle the size of the
!        numbers involved.
!         E_energy - eigen energy of the MPS
!              muD - Chemical Potential of the dot
!                N - number of particles in the MPS
!*************************************************************************
    implicit none
    real*8, intent(in) :: E_energy, muD
    integer, intent(in) :: N
    real*8 :: part_function
            part_function = (exp(beta(BT, T)*(E_energy - muD*N)))**-1
         end function part_function

   pure function beta(BT, T)
!*************************************************************************
!  Computes Beta, 1/kT, of our system.  The arguments of this function
!        are the magnetic induction field, BT,  in units of Tesla [T] and
!        temperature, T, in units of Kelvin [K] of our system.
!*************************************************************************
    implicit none
    real*8, intent(in) :: BT, T
```

```
    real*8 :: beta
    beta = (1 / (kb * T)) * coulomb_energy(BT)
         end function beta

         pure function stack_dim(sps)
!*************************************************************************
!  These cases give the 'bin depth' for the given SPS which were determined
!  after the runs were made the first time.  Initially the 'bin depth' is
!  guessed at on the high side such that the arrays to be written are sized
!  big enough to handle all the data.  Once the exact bin depth is determined
!  (by looking at the non-zero terms in the arrays) this value is used to size
!  the arrays to optimize the array size, free up memory and reduce run time.
!*************************************************************************
            implicit none
            integer, intent(in) :: sps
            integer stack_dim

      select case (SPS)
      case(:1)
         STACK_DIM = 1
      case(2)
         STACK_DIM = 1
      case(3)
         STACK_DIM = 1
      case(4)
         STACK_DIM = 2
      case(5)
         STACK_DIM = 2
      case(6)
         STACK_DIM = 3
      case(7)
         STACK_DIM = 5
      case(8)
```

```
   STACK_DIM = 8
case(9)
   STACK_DIM = 12
case(10)
   STACK_DIM = 20
case(11)
   STACK_DIM = 32
case(12)
   STACK_DIM = 58
case(13)
   STACK_DIM = 94
case(14)
   STACK_DIM = 169
case(15)
   STACK_DIM = 289
case(16)
   STACK_DIM = 526
case(17)
   STACK_DIM = 910
case(18)
   STACK_DIM = 1667
case(19)
   STACK_DIM = 2934
case(20)
   STACK_DIM = 5448
case(21)
   STACK_DIM = 9686
case(22)
      STACK_DIM = 18084
case(23)
      STACK_DIM = 32540
case(24)
      STACK_DIM = 61108
```

```
      case(25)
            STACK_DIM = 140000
      case(26)
            STACK_DIM = 240000
      case default
         STACK_DIM = 1
      end select

      return
         end function stack_dim

end module DECLARATIONS
```

APPENDIX I
NUMERICAL ANALYSIS CODE: SETBASIS.F

```
      subroutine setbasis(Mouter,N,mtot_state)
!*************************************************************************
!  Set up the basis of states with total angular momentum
!  m(1) + m(2) + ... + m(N) = Mtot.  Each m(j) can range from 0 to Mouter.
!
!  mtot_state is returned as the number of basis states with the same mtot.
!
!  The states are listed in mbasis(j,mtot_state)=m(j), j=1 to N.
!  In addition, mbasis(0,mtot_state) is a number labelling the MPS
!  (constructed from its m(j)'s).  This labeling number is referred to as the
!  'decimal index' and is used throughout the rest of the program to
!  enumerate MPS. It uses a stategy that has each SPS correspond to the bits
!  of a whole number.  These bits are easily read in binary where a set bit
!  (1) corresponds to a filled SPS and a (0) is empty.  The decimal index is
!  then sorted in ascending decimal order from 1 up to the number of MPS for
!  the system in question (given N and Mouter).
!
!  The decimal index, decndx, is stored in the mbasis(0,mtot_state).
!
!  Here the states are cycled through in this order, and those with the right
!  Mtot are added into mbasis.  There are some attempts to stop the search
!  when it's headed into a direction with no Mtot states (using complicated
!  'if' statements).
      use DECLARATIONS

      integer Mouter, N, mtot_state, msofar, lastj, decndx
      integer :: m(0:n)
      integer :: occupied(0:sps)

!  You need to initialize the mbasis array each time this is called.  If you
!  do not then once you index past the point with the maximum number of states
!  with the same MTOT, there will be states that don't get overwritten.  Upto
!  that point the states from the last call get overwritten by the states from
```

```
!  the current call.

      mbasis = 0
!*************************************************************************
!  If there is only one electron, this is easy.
!  The following cuts down on execution time for case of only one electron.
!  However the resulting mbasis(0,1) entry will be inconsistent with the cases
!  for N > 1.  This state one is the first state with the particular MTOT
!  currently being considered.  In the case below with N=1 the mbasis(0,1) = 0
!  regardless of where the SPS occupancy occurs.  With that said, it should
!  also be noted that this will not affect what we are concerned with, except
!  that this inconsistency might cause differences if the results of the older
!  and newer code for the case of N=1.  It should also be noted that this
!  doesn't affect the eigenvalues or vectors calculated by the code.

      if (N .eq. 1) then
         mbasis(0,1) = 0
         mbasis(1,1) = Mtot
         mtot_state = 1
         return
      end if

!  lastj is the last particle whose m has been changed.
!  msofar is 'm so far' i.e., m(1) + m(2) + ... + m(lastj).
      msofar = 0
      lastj = 1
      m(0) = -1
      m(1) = -1
      mtot_state = 0
!      write(*,*)
!      write(*,'(''SETBASIS--> Mtot='',i4)') Mtot
!  Increase m(lastj)
10    m(lastj) = m(lastj) + 1
```

```
      msofar = msofar + m(lastj)

!  The two tests make sure that m(lastj) isn't too big.
!*************************************************************************
!  The first test is true if the remaining m(j) can't all be made less than or
!  equal to Mouter.  The second test says that the sum must exceed Mtot:
!  i.e., msofar + (m(lastj)+1) + (m(lastj)+2)+...+(m(lastj)+N-lastj) > Mtot.
      if ((m(lastj) .gt. Mouter-N+lastj) .or.
     .(msofar+m(lastj)*(N-lastj)+((N-lastj)*(N-lastj+1))/2 .gt. Mtot))
     .then
         if (lastj .eq. 1) then
!            write(*,'(''SETBASIS--> Size of Hilbert space='',i5)')
!     .           mtot_state
      if (mtot_state .gt. hilbertspace(sps,n)) then
            write(*,'(''hilbertspace too small'')')
         write(*,'(''Mtot='',i5)') Mtot
               write(*,'(''hilbertspace ='',i5,''  mtot_state='',
     .              i10)') hilbertspace(sps,n),mtot_state
         stop 'hilbertspace bad'
            end if
            return
         end if
         msofar = msofar - m(lastj)
         lastj = lastj - 1
         msofar = msofar - m(lastj)
         go to 10

!  This is true if the sum can't manage to reach Mtot:  i.e.,
!  msofar + (Mouter) + (Mouter-1) +...+ (Mouter-(N-lastj-1)) < Mtot.

      else if
     .(msofar + Mouter*(N-lastj) - ((N-lastj)*(N-lastj-1))/2 .lt. Mtot)
     .then
```

```
         msofar = msofar - m(lastj)
         go to 10

!  This is true if we're on target for an allowed state.
      else if (lastj .lt. N-1) then
         lastj = lastj + 1
         m(lastj) = m(lastj-1)
         go to 10

!  We've found a good state.
      else
         m(N) = Mtot - msofar

         mtot_state = mtot_state + 1

!  Calculate decndx numbers and record the basis states in mbasis array
         if (mtot_state .le. hilbertspace(sps,n)) then
            decndx = 0
            do 20 j = 1,N
               mbasis(j,mtot_state) = m(j)
               decndx = ibset(decndx,m(j))
20          continue
            mbasis(0,mtot_state) = decndx
         end if

!  These loops and write statements let you inspect the bits that are
!  set for the current good state being considered.
         occupied = 0
         do 40 j = 1,N
40       occupied(m(j)) = 1

!         write(*,'(''SETBASIS--> mtot_state '',i3,'', decimal index'',
!     .        i5, '' = '',19i1)') mtot_state,decndx,
```

```
!     .        (occupied(j),j=Mouter,0,-1)
!  Adjust msofar to look for another basis state
         msofar = msofar - m(N-1)
         go to 10
      end if
      end
```

APPENDIX J
NUMERICAL ANALYSIS CODE: INTERACTION.F

```
      subroutine interaction(Mouter,N,mtot_state)
!*************************************************************************
!  Compute off-diagonal elements of the Hamiltonian matrix that result from
!  the electrons' interaction.  Do this as a sum over pseudopotentials.
!
!  The Hamiltonian is H(bra,ket).  However, as it stands, this routine puts
!  the result in a packed format in array H.  H contains the lower triangle
!  of the real symmetric matrix, packed row-wise.
!
!  The basis of states is in mbasis:  mbasis(j,ket) for j=1 to N is the list
!  m(1),m(2),...,m(N) specifying the occupied single-particle states of the
!  ket'th basis state.  (The m's range over 0 to Mouter.)
!
!  CC(m1,m2,mp) gives combinatorial factors resulting from the projection of
!  the product phi(m1,z1)phi(m2,z2) onto the relative-angular-momentum states
!  where m=2*mp-1 is the relative angular momentum.
!
!  mtot_state is returned as the number of basis states with the same mtot.

      use DECLARATIONS

      implicit none
      integer Mouter,N,mtot_state,occupied(0:sps),ket,bra,m
      integer j,j1,j2,m1,m2,m1p,m2p,s,mbra(n),state,mp,bra_decndx
      real*8 Hij,Hij2,Hij2H,Hij2F

!  Annihilate the states m1 and m2 (m2>m1) and then fill the states m2'
!  and m1' (where m2'>m1').  S gives the correct sign for fermions.  The
!  factor of 2 in H is from an overall 1/2 times 4 because we've restricted
!  m1 < m2 and m1' < m2'.
!
!  ket is the index labelling the ket state.
!  bra is the index labelling the bra state.
```

```
!
!  This loop will cycle through all of the MPS for the current system of
!  basis states with the same Mtot enumerated by "ket" from 1 to mtot_state.
!  These various 'systems' refer to the MPS for a given set of values of the
!  follows varibles:
!
!       N - number of particles
!  Mouter - number of SPS (actually this is one less than SPS since it's
!           counting starts at zero rather than one)
!    Mtot - total angular momentum of the MPS

      ket_state:  do ket = 1,mtot_state
!*****************************************************************************
!  First initialize and populate the occupied array with zeros.  Empty states
!  are assigned 0's (empty SPS) and filled states assigned 1's (filled SPS).
!  The 1-D "occupied" array is a temporarily array used to store the MPS being
!  considered for the current loop's pass.  It is used for calculation
!  purposes out of convience rather the 2-D MBASIS array.  As ket loops, each
!  MPS is considered in succession from first basis state to the last basis
!  state with the same MTOT.

         occupied = 0
         do j = 1,N
            occupied(mbasis(j,ket)) = 1
         end do

!  m1 and m2 are assigned integer values corresponding to the filled SPS.
!  Taking into consideration that m1<m2, each possible valid combination is
!  visited.  Each of these combinations is directly related to one of the
!  off-diagonal elements of the H array.  With each pass through these loops
!  only valid m1-m2-combinations are considered and the first thing to do is
!  anihilate (remove) these particles from the MPS.  The creation of two other
!  particles M1' and M2' can then be considered.  The particles are 'removed'
```

```
!  from the MPS only temporarily for the duration of the loop's cycle.  These
!  particles, m1 and m2, are put back at the end of the loop's cycle so that
!  the MPS will remain unchanged when it leaves the loop at line 80.
!
!  m1 is looped from 1 to N-1 because m1 > m2 and therefore can never be
!  assigned an angular momentum of the N'th particle but rather the maximum
!  angular momentum value possible for m1 (the 'left particle') is N-1.  In a
!  similar way, m2 (the 'right particle') can never be assigned a value less
!  than m1.  Therefore m1 is first assigned it's value inside the j1 loop and
!  this is used to adjust the lower limit of the j2 loop which is where m2 is
!  assigned.
        left_particle:  do j1 = 1,N-1
           m1 = mbasis(j1,ket)
           occupied(m1) = 0
           right_particle:  do j2 = j1+1,N
              m2 = mbasis(j2,ket)
              occupied(m2) = 0

!  I have notes describing how the following expresions limit the loop to only
!  considering m1' and m2' values that meet the condition we want which are:
!                        m1' < m2', but m1' + m2' = m1 + m2.
!  Here m1p and m2p are the variable names of m1' and m2' respectively.
!  m1p is looped through based on the particular m1 and m2 particles in
!  question. m2p simply falls out from about condition.
              prime_particles:  do m1p=max(0,m1+m2-Mouter),(m1+m2-1)/2
                 m2p = (m1+m2) - m1p
!  These two conditions eliminates that impossible situation of creating a
!  fermion in a SPS that already contains one.  It does this for both the m1'
!  and m2' particles.
                 if ((occupied(m1p).eq.0).and.(occupied(m2p).eq.0))then
!  As we pass (commute) creation and anihilation operators past each other we
!  gain sign changes based on the commutation laws of our system.  The code
!  keeps track of this sign change using S.  S is the integer valued exponent
```

```
!  of an overall negation factor (-1)^S.
!
!  To do this we first need to commute the anihilation operators to the j1
!  and j2 particles.  Then we need to count the commutation sign changes as a
!  result of the creation operators needed for the m1' and m2' particles.
!  Also note that the MPS stored in our temporary 'occupied' array is modified
!  such that both m1' and m2' particiles are created and their states are
!  filled.  These states will be emptied at the end of this 'if statement' so
!  that the MPS will be the same exiting as when it entered the 'if statement'.
                    S = j2 - j1 - 1
                    occupied(m2p) = 1
                    do m = m1p+1,m2p-1
                       S = S + occupied(m)
                    end do
                    occupied(m1p) = 1

!  mbra is filled with the SPS angular momentum values for each of the
!  particles.  The mbra array is a 1-D integer array that is allocated n
!  elements.  The array starts at 1 and only the first N elements are
!  significant since these represent the m's for each of the particles in the
!  current MPS.  The m's can range between 0 and Mouter and increase with
!  increasing array element in mbra such that mbra(1)<mbra(2)<...<mbra(N).
                    j = 0
                    do m = 0,Mouter
                       if (occupied(m).eq.1) then
                          j = j + 1
                          mbra(j) = m
                       end if
                    end do

!  The bra index is now assigned.  This is based on the current MPS's index as
!  determined in the STATEM function.  Based on the m values of the MPS's
!  various particles, back-mapping is done to determine the index assigned to
```

```
!  the given MPS.  This index is used as the index that labels the bra state.
!  The first part of the if statement just speeds up the process for the case
!  of m1=m1' AND m2=m2'.
                    if ((m1p.eq.m1) .and. (m2p.eq.m2)) then
                       bra = ket
                    else
                        bra_decndx = 0
                        do j = 1,N
                           bra_decndx = ibset(bra_decndx,mbra(j))
                        end do
                        j = 1
                        bra = 1
                        do while (mbasis(0,j).ne.bra_decndx)
                           bra = bra + 1
                           j = j + 1
                        end do
                    end if
!*************************************************************************
!  Now we calculate the elements of the Hamiltonian (actually half of the
!  elements in this case of a real symmetric matrix).  First we through out
!  the cases where bra < ket since we get these for free from the other side
!  of the triangle.  Then we sum over pseudopotentials for the various angular
!  momentums.  Keep in mind that actually we are using a modified angular
!  momentum index, mp = 1,2,3... , to sum over since it's easier to keep track
!  of than m = 1,3,5... (i.e. m=2*mp-1 where m =1,3,5,...,m1+m2, so mp=1 to
!  (m1+m2+1)/2.  Here we also include in the sum the coefficients needed to go
!  into relative and center-of-mass coordinates (see PROJCOEFF.F)
                    if (bra .ge. ket) then
                       Hij = 0
                       Hij_loop:  do mp = 1,(m1+m2+1)/2
                          if (Vpseudo(mp).ne.0.) Hij = Hij +
     .                         Vpseudo(mp)*CC(m1,m2,mp)*CC(m1p,m2p,mp)
                       end do Hij_loop
```

```
!  This adjusts the sign of the Hamiltonian elements based on exponent S
!  calculated earlier.
                        Hij = 2*Hij*((-1)**S)
!  In this routine 'state' is the index used for the elements of the 1-D
!  Hamiltonian array.  The 2-D array is stored sequentially in a 1-D packed
!  row-wise using this index.
                        state = bra*(bra-1)/2 + ket
                        H(state) = H(state) + Hij
                     end if
!  This returns the MPS back to it's configuration before the m1' and m2'
!  particles were created.
                     occupied(m1p) = 0
                     occupied(m2p) = 0
                  end if
               end do prime_particles
!  This returns the MPS back to it's configuration before the m1 and m2
!  particles were annihilated.
               occupied(m2) = 1
            end do right_particle
            occupied(m1) = 1
         end do left_particle
      end do ket_state
      end
```

APPENDIX K
NUMERICAL ANALYSIS CODE: DIAGONAL.F

```
      subroutine diagonal(Mouter,N,mtot_state)
!*************************************************************************
!  Put the diagonal terms into H.  Note that H is the lower triangle of a real
!  symmetric matrix, stored row-wise in packed form. Thus H(i,i) here becomes
!  H(1),H(3),H(6), etc., for i=1,2,3.
!
!  The diagonal terms arise from the confining potential W(m).  This is felt
!  by the electrons.   The confining potential felt here takes the shape of a
!  straight edge beginning at 'Medge' (zero before this value is reached) and
!  with a slope given by the variable 'diagscale'.
!
!  mtot_state is returned as the number of basis states with the same mtot.
!*************************************************************************
!  This routine is setup to look at the case of electrons with a V1 Model for
!  the pseudopotential
!*************************************************************************
      use DECLARATIONS
      use SYS_PARAMETERS

      implicit none
      integer Mouter,N,mtot_state,i,m,state,j,l,mp
      real*8 diag,sum,offset
      logical first
      data first/.true./
      save first
!*************************************************************************
!  The first time that this routine is called it assigns the confinement
!  potential that effects the particles and stores these values in the W(m)
!  array.  These values never change for the given run of the program so this
!  is only calculated on the first call and skipped for all subsequent calls.
!  However, the Hamiltonian, H(i) can change each time.
!  This routine calculates the diagonal elements of H(i) which result from
!  the confinement potential.  This can change for any given confinement
```

```
!  potential depending on the basis states involved (these states are stored
!  in the mbasis(m,state) array).  Therefore the Hamiltonian's diagonal is
!  recalculated each time this routine is called opposed to only once.

      if (.not. first) go to 130
!*************************************************************************
!  Here it is assumed that the single-particle potential begins
!  at m=Medge, and is thereafter linear in m with coefficient,
!  in units of V1, equal to diagscale.  The specific effect of this on the
!  potential array is that for W(m=Medge)=0 and has a slope defined by
!  diagscale from there all the way to m=Mouter.  The potential values
!  are initially stored in the Welec(m) array and subsequently transferred
!  to the W(m) array.  For our case (of electrons) W(m) = Welec(m).

         do 60 m = 0,Mouter
            if (m .le. Medge) then
               Welec(m) = 0
            else
               Welec(m) = (m-Medge)*diagscale
            end if
            W(m) = Welec(m)
60       continue

!      write(*,'(''DIAGONAL--> W(m)='',30f9.4)') (W(m),m=0,Mouter)
130   continue

      H = 0

      i = 0
      do 170 state = 1,mtot_state
         i = i + state
         diag = 0
         do 160 j = 1,N
```

```
160       diag = diag + W(mbasis(j,state))
          H(i) = diag
170    continue
       first = .false.
       end
```

APPENDIX L
NUMERICAL ANALYSIS CODE: PROJCOEFF.F

```
      subroutine projcoeff(Mouter)
!***************************************************************************
!  Compute the coefficients that arise when a product of two
!  lowest-level states is expanded into relative and center-of-
!  mass coordinates:
!    phi(m1,z1)phi(m2,z2) = (sum(m=0 to m1+m2))
!                           Cm(m1,m2)phi(m,z)phi(m1+m2-m,zbar)
!  where z = (z1-z2)/rt(2) and zbar = (z1+z2)/rt(2).
!  Here compute CC(m1,m2,mp)=Cm(m1,m2) (i.e., the m=2*mp-1 term) for m1 and m2
!  ranging from 0 to Mouter.  Need m =1,3,5,...,m1+m2, so mp=1 to (m1+m2+1)/2.
!***************************************************************************
      use DECLARATIONS
      implicit none
      integer Mouter,m1,m2,mp,m,k
      real*8 x, sum
! Initialize the CC matrix
      CC = 0.0
      m1_loop:  do m1 = 1,Mouter
         m2_loop:  do m2 = 0,m1-1
            x = sqrt(factorial(m1))*sqrt(factorial(m2)) *
     .          (2.d0**((m1+m2)/2.d0))
            cc_loop:  do mp = 1,(m1+m2+1)/2
               if (Vpseudo(mp).ne.0.) then
                  m = 2*mp - 1
                  sum = 0
                  sum_loop:  do k = max(0,m-m1),min(m,m2)
                     sum = sum + ((-1)**k)*combin(m1,m-k)*combin(m2,k)
                  end do sum_loop
                  CC(m1,m2,mp) = sqrt(factorial(m)) *
     .                           sqrt(factorial(m1+m2-m)) * sum/x
               end if
            end do cc_loop
         end do m2_loop
```

```
      end do m1_loop
      do 50 mp = 1,Mouter
      do 50 m1 = 0,Mouter
      do 50 m2 = m1+1,Mouter
         CC(m1,m2,mp) = -CC(m2,m1,mp)
50    continue
      end
```

LIST OF REFERENCES

[1] J. J. Thompson. *The Discharge of Electricity Through Gases*. Archibald Constable and Company, Westminster, United Kingdom, 1898.

[2] E. H. Hall. On a new action of the magnet on electric currents. *American Journal of Mathematics*, 2:287–292, November 1879.

[3] T. Chakraborty and P. Pietiläinen. *The Quantum Hall Effects*. Springer-Verlag, Heidelberg, Germany, 1995.

[4] K. v. Klitzing, G. Dorda, and M. Pepper. New method for high-accuracy determination of the fine-structure constant based on quantized Hall resistance. *Physical Review Letters*, 45(6):494–497, August 1980.

[5] D. C. Tsui, H. L. Stormer, and A. C. Gossard. Two-dimensional magnetotransport in the extreme quantum limit. *Physical Review Letters*, 48(22):1559–1562, May 1982.

[6] A. H. MacDonald. *Quantum Hall Effect: A Perspective*. Kluwer Academic Publishers, Boston, Massachusetts, 1989.

[7] K. von Klitzing, editor. *The Quantized Hall Effect*, Stuttgart, Germany, 1985. The Royal Swedish Academy.

[8] T. Ando, A. B. Fowler, and F. Stern. Electronic properties of two-dimensional systems. *Reviews of Modern Physics*, 54(2):437–476, April 1982.

[9] D. J. Griffiths. *Introduction to electrodynamics*. Prentice-Hall, Upper Saddle River, NJ, 1999.

[10] Peter Y. Yu and Manuel Cardona. *Fundamentals of Semiconductors: Physics and Materials Properties*. Springer-Verlag, Heidelberg, Germany, 1999.

[11] R. Kubo, S. J. Miyake, and N. Hashitsume. Quantum theory of galvanonlagnetic effect at extremely strong magnetic fields. *Solid State Physics*, 17:269–364, 1965.

[12] L. Landau. Diamagnetismus der metalle. *Zeitschrift fur Physics*, 64:629–637, July 1930.

[13] S. Datta. *Electronic Transport in Mesoscopic Systems*. Cambridge University Press, Cambridge, United Kingdom, 1999.

[14] D. C. Tsui and H. L. Stormer. *IEEE Journal of Quantum Electronics*, 22(9):1711–1719, September 1986.

[15] H. L. Stormer. Two-dimensional electron correlation in high magnetic fields. *Physica B: Condensed Matter*, 177(1-4):401–408, March 1992.

[16] X. G. Wen. Chiral Luttinger liquid and the edge excitations in the fractional quantum Hall states. *Physical Review B*, 41(18):12838–12844, June 1990.

[17] F. P. Milliken, C. P. Umbach, and R. A. Webb. Indications of a Luttinger liquid in the fractional quantum Hall regime. *Solid State Communications*, 97(4):309–313, 1996.

[18] F. D. M. Haldane and E. H. Rezayi. Fractional quantization of the hall effect: A hierarchy of incompressible quantum fluid states. *Physical Review Letters*, 51(7):605–608, August 1983.

[19] M. Grayson, D. C. Tsui, L. N. Pfeiffer, K. W. West, and A. M. Chang. Continuum of chiral Luttinger liquids at the fractional quantum Hall edge. *Physical Review Letters*, 80(5):1062–1065, February 1998.

[20] B. I. Halperin. Quantized Hall conductance, current-carrying edge states, and the existence of extended states in a two-dimensional disordered potential. *Physical Review B*, 25(4):2185–2190, February 1982.

[21] G. D. Mahan. *Many Particle Physics*. Plenum Press, New Yort, New York, 1990.

[22] D. Yoshioka. *The Quantum Hall Effect*. Springer-Verlag, Heidelberg, Germany, 2002.

[23] M. H. Cohen, L. M. Falicov, and J. C. Phillips. Superconductive tunneling. *Physical Review Letters*, 8:316–318, April 1962.

[24] Eugen Merzbacher. *Quantum Mechanics*. John Wiley and Sons, Inc., New York, NY, 1998.

[25] R. Shankar. *Principles of Quantum Mechanics*. Plenum Press, New Yort, New York, 1994.

[26] Neil W. Ashcroft and N. David Mermin. *Solid State Physics*. Saunders College Publishing, New Yort, New York, 1976.

[27] R. B. Laughlin. Quantized Hall conductivity in two dimensions. *Physical Review B*, 23(10):5632–5633, May 1981.

[28] U. Zülicke, J. J. Palacios, and A. H. MacDonald. Fractional-quantum-Hall edge electrons and Fermi statistics. *Physical Review B*, 67(045303):1–8, January 2003.

[29] G. Fano, F. Ortolani, and E. Colombo. Configuration-interaction calculations on the fractional quantum Hall effect. *Physical Review B*, 34(4):2670–2680, August 1986.

[30] F. D. M. Haldane and E. H. Rezayi. Finite-size studies of the incompressible state of the fractionally quantized Hall effect and its excitations. *Physical Review Letters*, 54(3):237–240, January 1985.

[31] G. Murthy and R. Shankar. Hamiltonian theory of the fractional quantum Hall effect: Effect of Landau level mixing. *Physical Review B*, 65(245309):1–11, June 2002.

[32] W. Pan, H. L. Stormer, D. C. Tsui, L. N. Pfeiffer, K. W. Baldwin, and K. W. West. Fractional quantum Hall effect of composite fermions. *Physical Review Letters*, 90(016801):1–4, January 2003.

[33] C. L. Kane and M. P. A. Fisher. Impurity scattering and transport of fractional quantum Hall edge states. *Physical Review B*, 51(19):13449–13466, May 1995.

[34] A. H. MacDonald. No end of tricks: Electrons in the fractional quantum Hall regime. *Solid State Communications*, 102(2-3):143–154, 1997.

[35] X. G. Wen. Electrodynamical properties of gapless edge excitations in the fractional quantum Hall states. *Physical Review Letters*, 64(18):2206–2209, April 1990.

[36] X. G. Wen. Edge transport properties of the fractional quantum Hall states and weak-impurity scattering of a one-dimensional charge density wave. *Physical Review B*, 44(11):5708–5719, September 1991.

[37] A. V. Shytov, L. S. Levitov, and B. I. Halperin. Tunneling into the edge of a compressible quantum Hall state. *Physical Review Letters*, 80(1):141–144, January 1998.

[38] J. J. Palacios and A. H. MacDonald. Numerical tests of the chiral Luttinger liquid theory for fractional Hall edges. *Physical Review Letters*, 76(1):118–121, January 1996.

[39] M. D. Johnson and A. H. MacDonald. Composite edges in the $\nu = 2/3$ fractional quantum Hall effect. *Physical Review Letters*, 67(15):2060–2063, October 1991.

[40] I. L. Aleiner and L. I. Glazman. Novel edge excitations of two-dimensional electron liquid in a magnetic field. *Physical Review Letters*, 72(18):2935–2938, May 1994.

[41] I. L. Aleiner, D. Yue, and L. I. Glazman. Acoustic excitations of a confined two-dimensional electron liquid in a magnetic field. *Physical Review B*, 51(19):13467–13474, May 1995.

[42] A. M. Chang, L. N. Pfeiffer, and K. W. West. Observation of chiral Luttinger behavior in electron tunneling into fractional quantum Hall edges. *Physical Review Letters*, 77(12):2538–2541, September 1996.

[43] U. Zülicke and A. H. MacDonald. Periphery deformations and tunneling at correlated quantum Hall edges. *Physical Review B*, 60(3):1837–1841, July 1999.

[44] X. G. Wen. Theory of the edge states in fractional quantum Hall effects. *International Journal of Modern Physics B*, 6(10):1711–1762, March 1992.

[45] D. H. Lee and X. G. Wen. Edge tunneling in fractional quantum hall regime. *arXiv:cond-mat*, 9809160:1–4, September 1998.

[46] J. K. Jain. The role of analogy in unraveling the fractional quantum hall effect mystery. *Physica E*, 20:79–88, 2003.

[47] A. L. Fetter and J. D. Walecka. *Quantum Theory of Many-Particle Systems*. McGraw-Hill, Boston, Massachusetts, 1971.

[48] Z. F. Ezawa. *Quantum Hall Effects: Field Theoretical Approach and Related Topics*. World Scientific Publishing Co., Singapore, 2000.

[49] S. D. Sarma and A Pinczuk. *Perspectives in Quantum Hall Effects: Novel Quantum Liquids in Low-Dimensional Semiconductor Structures*. John Wiley and Sons, Inc., New York, NY, 1997.

[50] K. N. Shrivastava. *Introduction to Quantum Hall Effect*. Nova Science Publishers, Inc., Hauppauge, NY, 2002.