Path Integral Monte Carlo Simulation of Positronium within a Dielectric Spherical Cavity

Zachary Wolfson

Department of Physics and Astronomy

Swarthmore College

A thesis submitted for the degree of

Bachelor of Arts

Acknowledgements

I would like to thank first and foremost my advisor, Amy Bug, for her immense amount of help with absolutely everything involved with this project. Thanks also to Tim Cronin, my co-researcher who studied Ps in cylindrical pores, for producing such an organized version of the code, and for many fruitful and interesting discussions throughout. I thank Catherine Crouch for taking the time to read my thesis and giving me very helpful feedback. This research was funded by the Office of the Provost of Swarthmore College.

Abstract

Positronium (Ps), the bound state of an electron and a positron, may be used as a probe of the porous space in bulk materials due to its long vacuum lifetime and the fact that this lifetime depends strongly on its environment. In particular, an accurate model relating the lifetime of the positronium to the size of the pore is necessary. The primary outcome of this research is a computer program that models Ps exactly as two quantum particles, using Path Integral Monte Carlo (PIMC) to simulate the electron and positron each as a classical polymer. As a further correction, we include the dielectric response of the surrounding material to the presence of the two charges.

We find that the material's polarization causes the Ps to be more attracted to the wall, decreasing the pickoff lifetime; this decrease is even more dramatic in the case of a bare positron within the cavity. Although this effect is more substantial for larger cavities, it does approach a limit as the radius of the cavity approaches infinity, since then the situation is that of Ps near a flat dielectric wall. Also a limit is reached in the behavior of the lifetime as the dielectric constant k_o increases, since the energy scales approximately like $(1-k_o)/(1+k_o)$. Using a two-particle simulation with no dielectric energy (or equivalently, setting $k_o = 1$) in general produces lifetimes significantly higher than the standard Tao-Eldrup model, but having $k_o > 1$ reduces the lifetime once more, agreeing with Tao-Eldrup in certain cases. We found that typical values of k_o have the same order of magnitude effect as do different pore geometries, and that our results explain certain discrepancies in data that other models could not.

Contents

1 Introduction **1**
1.1 Motivation 1
1.2 Positronium 3

2 Theory **9**
2.1 Quantum Statistical Mechanics 9
2.2 Polarization 14

3 Methods **21**
3.1 The Metropolis Monte Carlo Algorithm 21
3.2 A Simple Example of PIMC 25
3.3 Approximation Methods for Positronium 35

4 Results **40**
4.1 Summary 40
4.2 Data 46
4.3 Comparisons 49
4.4 Conclusion 51

A Solution to Poisson's Equation **56**

B Main Positronium Code, cavity_diel_Ps.f90 **59**

Chapter 1

Introduction

1.1 Motivation

Before beginning any scientific project, it is good to have an answer to the question "Why is this project important?" The main purpose of the project described in this document was to produce a computer program that calculates the lifetime of positronium (Ps), an exotic atom whose two constituent particles can annihilate with one another, when the Ps atom is contained in a spherically shaped cavity, which itself is surrounded by bulk material of a known dielectric constant. There are several ways to answer the above question as it applies to our project; we shall discuss them in this introductory section.

The first and most obvious question is "Why would anyone want to know the lifetime of Ps under these conditions?" and this is the easiest to answer. Many materials scientists wish to know how large the pores in their materials are, for these pores affect the properties of the material in significant ways. An important example for us is that of research groups who are attempting to lower the dielectric constants of materials by carving out pores (using appropriately named chemicals called "porogens") in these materials. The materials can suffer structural breakdown if the pores get too large, but more cavities (less material) creates a lower dielectric constant; this is highly desirable because, for example, it reduces RC time delays in circuits. Since there are experimental methods to measure the lifetime of Ps when it annihilates within a pore of such a material (notably positron annihilation lifetime spectroscopy, PALS), if these researchers

knew the relationship between the lifetime of Ps and the size of the cavity in which it annihilated, they could obtain the desired information about the structure of the voids that the porogens created.

A simple model that accomplishes this goal has existed for several decades now. It is known as the Tao-Eldrup (TE) model, it consists of a single simple equation,

$$\tau = (0.5ns)\left[\frac{0.16}{R_c} + \frac{1}{2\pi}\sin\left(2\pi\frac{R_c - 0.16}{R_c}\right)\right]^{-1} \tag{1.1}$$

(here R_c, the cavity radius, is measured in nm), and it has proven to be accurate where it is applicable.[1] So the next question to be answered is "Why bother to write a complicated computer program at all?" Of course the answer is that the TE model is too good to be true; one cannot get something for nothing when doing physics. While the TE equation is simple and easy to use, so is the "particle in a box " model it is based on, and this limits its applicability: the TE model is only valid in the low temperature, small cavity regime. A more robust, physically correct model is required for room temperature experiments, which of course is the temperature at which most PALS procedures occur.

The groundwork for our simulation was laid by past members of our research group (1), by creating a two particle simulation of Ps within spherical voids. The motivation for further developing this model, that is, the answer to the question "Why should we simulate the material's polarization as well?", can be understood by examining the following table(2):

	Radius from gas adsorption	Ps Lifetime
silica gel (grade 7)	1.25 nm	32 ns
silica gel	1.25 nm	42.2 ns
zeolite (MCM-48)	1.26 nm	26.9 ns

This data is problematic for a model hoping to relate cavity radii, alone, to positronium lifetimes in a consistent way, since the Ps lifetimes within these

[1] The standard references are (3) and (4).

samples are significantly different, yet their radii are almost exactly the same. One possible explanation for these discrepancies is that the materials had different dielectric constants.[1] A model that takes only cavity radius as input to determine the lifetime of positronium cannot hope to explain these data. By introducing the new parameter of dielectric constant into the computer simulation, we hope to fit the data even more precisely than one could with existing models.

1.2 Positronium

In 1928, Paul Dirac devised an equation based on the brand new quantum theory that was appropriate for a relativistic spin-1/2 particle, ie an electron. When he solved this equation, he found that one of the solutions' energy eigenvalues was given, as expected, by the positive square root of the relativistic formula $E^2 = p^2c^2 + m^2c^4$, which he (correctly) associated with an electron. However, the negative square root of the above, which should have been unphysical since any particle has positive kinetic/mass energy, also corresponded to a perfectly good solution of the Dirac equation. Though surprised, Dirac believed in the validity of his equation, and so he stood behind the negative square root solution, positing that all the negative energy states must be filled by a "sea" of electrons which did not interact with the rest of the world, so we do not detect them, but whose presence prevents "normal" electrons from falling into these unobserved negative energy states.(5)

Fortunately, this bizarre Dirac sea is a fiction;[2] the correct interpretation of the negative energy solution is that it corresponds to a real particle with the same mass and spin as the electron, but with opposite charge, and whose positive energy states look the same as negative electron states. This particle is now known

[1] Another explanation is that the cavities had different geometries, ie not just spherical but rectangular, cylindrical, etc. The way lifetime depends on pore geometry was explored by Consolati(6), and shall be discussed and compared to our model in Section 4.3 below.

[2] For example, it cannot be applied to bosons, for they have no exclusion principle, and thus there would be no way to prevent all observed bosons from falling into the bottomless negative energy sea.(7)

as the positron.[1] Dirac's marvelous prediction of the existence of positrons was confirmed experimentally by Carl Anderson in 1932, becoming the first example of antimatter and opening the door to the field of elementary particle physics.(5)

The same year that Anderson discovered the positron, he predicted that, due to the Coulomb attraction between a positron and electron, the two particles should be able to form a bound state, known as positronium. The state would be inherently short lived, since one of the fundamental processes of quantum electrodynamics is the annihilation of a positron and electron into photons, so that eventually positronium would self-annihilate. It was not until 1951 that Martin Deutsch provided the first experimental confirmation of the existence of positronium.(5) Although other "exotic atoms" have been predicted and discovered since then (for example muonium, which consists of an antimuon and an electron), positronium remains the most common and most useful, due to the fact that one of its spin states has the relatively long vacuum lifetime of 142ns. The reason for this lifetime will be explained below.

The first step in analyzing positronium is to solve the Schrödinger equation for the appropriate Hamiltonian. For most quantum systems, this is an impossible, or at least very difficult, task. But this is a two-body system, and the interaction is Coulombic, so if one is familiar with the undergraduate level solution to the hydrogen atom, it requires only a moment's thought to realize that the (free space) Hamiltonian for positronium differs from that for hydrogen most significantly in the value of the reduced mass.[2] Recall that in most of the formulas regarding the hydrogen atom, the electron mass m_e appears, but really this should be the reduced mass μ of the system, since the first step performed when solving the hydrogen atom was going to the center of mass frame and using the relative

[1] According to the Feynman calculus of quantum electrodynamics, a positron is "just" an electron moving backwards in time.

[2] The full story is a bit more complicated. For example, since neither particle of Ps is even close to being stationary, as hydrogen's proton is, the Coulomb attraction is truly electrodynamic, and not electrostatic, in nature. A term accounting for the finite propagation time of the electric field must therefore be included. As usual, see (5) for further information. The discussion here of the reduced mass is the most significant difference from hydrogen, but these other smaller effects must be accounted for in a full description of Ps.

coordinate r as the variable. (By definition, we have that

$$\mu \equiv \frac{m_1 m_2}{m_1 + m_2} \tag{1.2}$$

for a system of particles with masses m_1 and m_2.) In the case of hydrogen, where $m_1 = m_e \ll m_2 = m_p$ since the electron is so much less massive than the proton, $\mu \approx (m_e m_p)/m_p = m_e$, which is why the formulas get away with using the electron mass rather than the reduced mass.

However, for positronium, the situation is that $m_1 = m_2 = m_e$, so $\mu = m_e/2$. This change is quite minor, affecting only numerical values, not functional forms (for example, the ground state energy of positronium is 6.8eV, half that of hydrogen). Thus results concerning Ps in vacuum, namely its wavefunction, may be derived analytically.

What also carries over from hydrogen is an understanding of the spin states of positronium. The positron is a spin-1/2 particle, just like the proton, so the most basic angular momentum addition case is once again present, in which two spin-1/2 particles combine to form a degenerate triplet of states each with a total spin of 1, and a singlet of one state with a total spin of 0. Positronium in the triplet spin state is known as ortho-positronium (o-Ps), while singlet positronium is known as para-positronium (p-Ps).

We may use the above analysis to calculate the vacuum lifetime of positronium; that this lifetime is non-infinite is a decidedly non-hydrogenic feature of positronium. We know from quantum electrodynamics that a fundamental process is the annihilation of a positron and electron into some number of photons, and that the rate of this process depends on the number of photons produced. The reason is this: there is an elegant symmetry in particle physics, known as the principle of detailed balance, that requires that such fundamental processes have equal likelihood of proceeding in the forward direction as they do in the backward direction[1]. Since clearly it is less likely for many photons to come together and produce an electron and positron than for only a few to come together and do

[1]This follows from the fact that the Hamiltonian is invariant under time reversal. There is some question as to whether *all* processes are time reversible; it is thought that, although electromagnetic and strong interactions are, weak interactions break time reversibility. See Griffiths(5) p.134-135 for an interesting discussion.

the same, it follows that the annihilation of an electron and a positron into many photons is much less likely (that is, the rate is lower) than an annihilation which produces fewer photons. (5)

By going to the center of mass frame of the electron and positron, where there is initially no momentum, clearly an annihilation can never produce only one photon, for then there would be net momentum in the frame, violating conservation of momentum. There is a more obscure conservation law of particle physics, that of charge conjugation, which puts further restrictions on the number of photons. In order to conserve the "charge conjugation number" of our system, the number n of photons produced must satisfy

$$(-1)^{l+s} = (-1)^n \tag{1.3}$$

where l is the total orbital angular momentum of the electron and positron, and s is their total spin angular momentum. In positronium's ground state, $l = 0$, so we obtain that the parity of n must be the same as that of s. Thus ground state p-Ps, with $s = 0$, may only annihilate into an even number of photons, while o-Ps, with $s = 1$, may only annihilate into an odd number of photons. Since a one photon process has already be ruled out, it follows that an o-Ps annihilation must produce a minimum of 3 photons.

The probability for annihilation events of 4 photons or more is negligible compared to the 2 or 3 photon probabilities, so we may consider p-Ps and o-Ps to annihilate into 2 and 3 photons, respectively. But clearly it is less likely for 3 photons to intersect each other's paths than for 2 photons to do so, so the annihilation rate of o-Ps is much lower than that of p-Ps; in other words, o-Ps lives a good deal longer than p-Ps.

Quantum rate theory may be used to calculate the rates exactly: it turns out that the annihilation rate is, not too surprisingly, proportional to the probability $|\psi(0)|^2$ that the electron and the positron are found at the same location (where $\psi(r)$ is the relative wavefunction). Using the known hydrogenic wavefunction and the proper constants from rate theory (a calculation beyond the scope of this thesis - see (5)), we find that the lifetime of p-Ps is $(\Gamma_{vac})^{-1} = 0.13ns$, while that of o-Ps is $(\Gamma_{vac})^{-1} = 142ns$ (the symbol Γ is used to denote the annihilation rate, the inverse of the lifetime. The subscript "vac" is meant to indicate the

fact that this is the annihilation rate of Ps in a vacuum). We now see that the impressively long lifetime (by the standards of particle physics) of o-Ps follows from its inability to become any fewer than 3 photons.

This is all fine in a vacuum, but when positronium is put into a (in this thesis, what is assumed to be spherically shaped) cavity within a material, which is the situation simulated in our computer program, several things change. Due to both the confinement imposed by the wall of the cavity and the fact that the wall may polarize in the presence of the charged particles of positronium, the wavefunction is modified, so the crucial value $|\psi(0)|^2$ is changed as well. In fact, it is convenient to define the parameter

$$\kappa = \frac{|\psi(0)|^2}{|\psi_{vac}(0)|^2} \tag{1.4}$$

known as the internal contact density, where $\psi_{vac}(r)$ is the hydrogenic wavefunction in vacuum, and $\psi(r)$ is the relative wavefunction in the cavity. Since the annihilation rate is proportional to $|\psi(0)|^2$, it follows that the annihilation rate in the cavity is just

$$\Gamma = \kappa\Gamma_{vac} \tag{1.5}$$

We expect that confinement causes κ to increase since the wavefunction is compressed to exist in a smaller space, but the fact that the cavity polarizes in response to the charged particles of Ps will decrease κ, as the Coulomb attraction between the positron and electron is effectively shielded by the bound charge in the surrounding wall. Relatively little will be said about κ in this thesis; for more information, see (8).

Putting Ps in a cavity in a molecular solid introduces an even more drastic change to its lifetime, because near the walls of the cavity lie molecular electrons which are just as able to annihilate with the positron as is its own bound electron. Since the processes are characterized by very different lifetimes, we distinguish them by henceforth referring to annihilation with the electron of Ps as "self annihilation," and annihilation with molecular electrons as "pickoff annihilation." The total annihilation rate is then the sum of rates of these two independent

processes:[1]

$$\Gamma = \kappa\Gamma_{vac} + \Gamma_{p.o.}. \tag{1.6}$$

The pickoff annihilation rate clearly has to do with the probability that the positron is found near the molecular electrons. In fact rate theory gives that the relationship is(9)

$$\Gamma_{p.o.} = \pi r_e^2 c \int n_+(\mathbf{r}) n_-(\mathbf{r}) \gamma[n_+(\mathbf{r})] d^3\mathbf{r} \tag{1.7}$$

where r_e is the classical radius of the electron and n_+, n_- are the densities of the positron and electrons, respectively. The functional γ is known as the local enhancement factor, and it serves to keep track of the fact that the density of the electrons is affected by the presence of the positron. In our work, this functional is set to unity.

One successful method in dealing with Eq.1.7 is to treat the molecular electrons as if they intrude into the cavity a radial distance Δ with uniform density throughout that region, so that $n_-(\mathbf{r}) = const.$ for $R_c - \Delta < r < R_c$ and is zero otherwise. This simplifying assumption, along with solving for $n_+(\mathbf{r})$ under the premise that the positron is a single particle in a spherical box, comprises the Tao-Eldrup model, and results in Eq. 1.1.

[1]The fact that the two rates contribute independently to the total rate is an assumption of the model, but it is one justified by general quantum rate theory.

Chapter 2

Theory

2.1 Quantum Statistical Mechanics

The purpose of equilibrium quantum statistical mechanics is to calculate the thermal averages of observables for a system composed of many quantum particles when that system is in its equilibrium macrostate. The thermal average is performed over all the microstates that the system may occupy when it is at equilibrium with the environment (a heat bath of absolute temperature T). Since both standard quantum mechanics and classical statistical mechanics characterize a system by its "state," combining the two disciplines in order to accomplish this goal seems natural. However, since the equilibrium macrostate is by definition the totality of many microstates, standard "bra-ket" quantum mechanics cannot be used to describe equilibrium. Instead, the density matrix formalism must be employed (in this formalism's language, the equilibrium macrostate is known as a "mixed state").[1]

The density matrix, $\hat{\rho}$, of a quantum system is defined by

$$\hat{\rho} = \sum_n p_n |\psi_n\rangle\langle\psi_n| \tag{2.1}$$

where p_n is the probability that the system is in the state $|\psi_n\rangle$.(10) The density matrix is so named because it describes the "densities" (probabilities) that various

[1]Throughout this section, the reader is referred to (11) and (12) for additional information.

states occur, and it can be written as a matrix because, as can be clearly seen from its definition Eq. 2.1, it is an operator.

We shall restrict our attention to systems whose microstates are characterized by Maxwell-Boltzmann statistics, so that in the above equation we will always take $p_n = e^{-\beta E_n}$, where E_n is the energy of the system in its n^{th} microstate, and $\beta = 1/kT$ is the inverse temperature of the environment. Implicit in the above is that we have picked the n^{th} state to be an energy eigenstate, so that the ψ_n from above are eigenstates of the Hamiltonian $\hat{H}$. From this we see that

$$\hat{\rho} = \sum_n e^{-\beta E_n} |\psi_n\rangle\langle\psi_n| = e^{-\beta \hat{H}} \tag{2.2}$$

which is a very compact way of writing the density matrix. This representation is sometimes derived by taking as fundamental the Bloch equation

$$-\frac{\partial \hat{\rho}}{\partial \beta} = \hat{H}\hat{\rho} \tag{2.3}$$

rather than Eq. 2.1; the difference is that Eq. 2.1 is the general formula for the density matrix of any quantum system, whereas Eq. 2.2, the solution to Eq. 2.3, is the thermal density matrix.

The density matrix contains all the information we can and would like to know about the equilibrium of the thermal system, as will be seen in the following two calculations. Recalling that the trace of an operator is defined to be the sum of its diagonal terms when the operator is written as a matrix (the basis chosen to do so does not matter, since the trace is basis-independent), we have

$$\mathrm{Tr}\hat{\rho} = \sum_m \langle\psi_m|\hat{\rho}|\psi_m\rangle = \sum_{m,n} e^{-\beta E_n} |\langle\psi_m|\psi_n\rangle|^2 = \sum_n e^{-\beta E_n} = Q. \tag{2.4}$$

That is, the trace of the quantum density matrix is the partition function of statistical mechanics. Also, the thermal average of any observable $\hat{A}$ can be

found according to:

$$\langle \hat{A} \rangle = \sum_n \frac{e^{-\beta E_n}}{Q} \langle \psi_n | \hat{A} | \psi_n \rangle = \sum_{k,m,n} \delta_{nk} \frac{e^{-\beta E_k}}{Q} \delta_{km} \langle \psi_m | \hat{A} | \psi_n \rangle$$

$$= \frac{1}{Q} \sum_{k,m,n} \langle \psi_n | \psi_k \rangle e^{-\beta E_k} \langle \psi_k | \psi_m \rangle \langle \psi_m | \hat{A} | \psi_n \rangle$$

$$= \frac{1}{Q} \sum_{m,n} \langle \psi_n | \hat{\rho} | \psi_m \rangle \langle \psi_m | \hat{A} | \psi_n \rangle = \frac{1}{Q} \sum_n \langle \psi_n | \hat{\rho} \hat{A} | \psi_n \rangle = \frac{\mathrm{Tr} \hat{\rho} \hat{A}}{\mathrm{Tr} \hat{\rho}} \tag{2.5}$$

We used the fact that $\sum_m |\psi_m\rangle\langle\psi_m|$ is the identity operator. Note in the first equality the distinction between the expectation value of $\hat{A}$ in the state n (which is $\langle \psi_n | \hat{A} | \psi_n \rangle$) and its thermal average. The expectation value is the quantum mechanical "average" of the observable for one microstate of the system, while the thermal average characterizes the ensemble of all microstates, the value of $\hat{A}$ associated with the system when it has found its equilibrium macrostate.

Therefore the density matrix does indeed contain all of the system's information. Unfortunately, referring to Eq. 2.1, it appears that calculation of the density matrix requires solving the eigenvalue problem of $\hat{H}$, which is impossible for two interacting quantum particles within a spherical "box," and so is all the more intractable when the space beyond the box is allowed to polarize. However, Feynman's path integral formalism may be used to cast time evolution problems in different terms, making them more amenable to being solved by computational methods. We shall apply the same idea to the thermal density matrix.

To solve an eigenvalue problem, the path integral formalism is used to find the usual propagator in the position basis:

$$U(\mathbf{r}, \mathbf{r}') = \langle \mathbf{r} | \hat{U} | \mathbf{r}' \rangle = \langle \mathbf{r} | e^{-\frac{it}{\hbar} \hat{H}} | \mathbf{r}' \rangle = e^{-\frac{it}{\hbar} H(\mathbf{r}, \mathbf{r}')} \tag{2.6}$$

Comparing to the density matrix's representation in the position basis (using Eq. 2.2), $\rho(\mathbf{r}, \mathbf{r}') = e^{-\beta H(\mathbf{r}, \mathbf{r}')}$, we see that the propagator and the density matrix will have the same solutions as long as we make the identification $it/\hbar \leftrightarrow \beta$. When solving for the propagator using path integrals, the quantum particle is imagined to traverse any path in $\mathbf{r} - t$ space; now the particle will be seen as following paths in $\mathbf{r} - \beta$ space. Since each "imaginary time slice" (each increment of β, which corresponds to imaginary time via the above substitution) actually exists

at the same instant in real time,[1] we shall imagine that there are many different copies of the actual quantum particle(s) in our system at once, each existing in different imaginary time slices and at various positions. These copies, which are artifacts of the path integral formalism and are not to be considered any more real than the infinitude of particles traversing each possible path in the standard path integral method, are known as beads. The bead locations can be thought of as possible locations of the true quantum particle that the beads represent, so that the beads capture the fact that quantum particles are not localized entities.

In fact we shall not apply the path integral idea directly to the density matrix, but instead to its trace, which we know from Eq. 2.4 is the partition function. We shall compute the trace using the position basis, so that

$$Q = \mathrm{Tr} e^{-\beta \hat{H}} = \int d\mathbf{r} \langle \mathbf{r} | e^{-\beta \hat{H}} | \mathbf{r} \rangle \tag{2.7}$$

(Whenever the endpoints of an integral are not labeled, it is implied that the integral is to be done over all space, from $-\infty$ to $+\infty$ in each dimension). It proves fruitful to separate the Hamiltonian into kinetic and potential energy terms, $\hat{H} = \hat{T} + \hat{V}$. [2] The problem with doing so is that whenever two operators $\hat{A}$ and $\hat{B}$ do not commute, the exponential of their sum is not the product of their individual exponentials, unlike with real numbers. However, the Trotter formula(11) allows us to factor the two terms at the price of taking a limit:

$$e^{\hat{A}+\hat{B}} = \lim_{P \to \infty} \left(e^{\hat{A}/P} e^{\hat{B}/P} \right)^P \tag{2.8}$$

where P takes on integer values as it approaches infinity. Thus the right hand side can be written as the factor $e^{\hat{A}/P} e^{\hat{B}/P}$ multiplied by itself P times, after which the limit $P \to \infty$ is taken. We apply this formula to Eq. 2.7 and insert $P - 1$

[1] In fact, as all calculations take place at equilibrium, there are no dynamics whatsoever.

[2] To avoid confusion between the kinetic energy operator $\hat{T}$ and the temperature T, we shall always use β whenever the temperature must appear in the same expression as $\hat{T}$.

identities ($\int d\mathbf{r}_i|\mathbf{r}_i\rangle\langle\mathbf{r}_i|$) between the P factors to obtain:

$$\begin{aligned} Q = \lim_{P\to\infty} \int d\mathbf{r}_1 \int d\mathbf{r}_2 \cdots \int d\mathbf{r}_P \langle \mathbf{r}_1|e^{-\beta\hat{T}/P}e^{-\beta\hat{V}/P}|\mathbf{r}_2\rangle \\ \times\langle \mathbf{r}_2|e^{-\beta\hat{T}/P}e^{-\beta\hat{V}/P}|\mathbf{r}_3\rangle \cdots \langle \mathbf{r}_P|e^{-\beta\hat{T}/P}e^{-\beta\hat{V}/P}|\mathbf{r}_1\rangle \\ = \lim_{P\to\infty} \prod_{i=1}^{P} \int d\mathbf{r}_i \langle \mathbf{r}_i|e^{-\beta\hat{T}/P}e^{-\beta\hat{V}/P}|\mathbf{r}_{i+1}\rangle \end{aligned} \tag{2.9}$$

where we have renamed $\mathbf{r}$ as $\mathbf{r}_1$ and let $\mathbf{r}_{P+1} \equiv \mathbf{r}_1$ for notational consistency and ease. Each matrix element in Eq. 2.9 can be found with much algebra but without much difficulty. The result is

$$\begin{aligned} Q = \lim_{P\to\infty} \left(\frac{mP}{2\pi\beta\hbar^2}\right)^{3P/2} \int d\mathbf{r}_1^{(1)} \cdots d\mathbf{r}_P^{(2)} \\ \times exp\left[-\beta\frac{\kappa}{2}\sum_{k=1}^{2}\sum_{j=1}^{P}\left(\mathbf{r}_j^{(k)} - \mathbf{r}_{j+1}^{(k)}\right)^2 - \frac{\beta}{P}\sum_{j=1}^{P} V(\mathbf{r}_j^{(1)}, \mathbf{r}_j^{(2)})\right] \end{aligned} \tag{2.10}$$

where m is the mass of each particle, $\mathbf{r}_j^{(k)}$ denotes the position of the j^{th} bead of the k^{th} particle ($k = 1$ for the positron, $k = 2$ for the electron), and we have defined

$$\kappa = \frac{mP}{(\beta\hbar)^2}. \tag{2.11}$$

Now suppose that we forego the exact mathematical limit and instead take P to be a large, but finite, number. This not only makes Eq. 2.10 simpler, it is also necessary for the eventual goal of computation via computer. Then the partition function of our system is exactly the same as the partition function for two chains of P classical particles (corresponding to the false beads), connected to each other with springs of spring constant κ (the chains are indeed closed because $\mathbf{r}_{P+1}^{(k)} \equiv \mathbf{r}_1^{(k)}$ for each particle). The chains interact with each other and with the environment via the same potential $V(\mathbf{r}_j^{(1)}, \mathbf{r}_j^{(2)})$ as in our original problem, but those interactions occur at the reduced inverse temperature β/P.[1] Because the same-chain adjacent beads interact as a harmonic oscillator, we sometimes call the chains "polymers".(13)

[1]This is essentially what allows the quantum problem to become classical: we are now studying a system at a higher temperature (lower β) than our actual system experiences; as a result, the beads possess high thermal energy and thus behave classically.

This surprising isomorphism between our quantum system and the classical system of two polymers is now complete. The kinetic energy has become spring energy between adjacent beads,[1] and the fact that quantum particles are not localized is handled by the large number P of beads used in each chain. Thus our derivations have culminated in the following amazing fact: a classical mechanics simulation may be used to obtain the solution to a quantum mechanical problem.[2]

In particular, if we find the equilibrium states of the classical system of the two chains of beads using our the potential energy function appropriate for our system, we will know the equilibrium states of positronium in the cavity. Secion 3.1 explains how the classical system is to be solved. For now, we turn to a calculation of the dielectric energy terms that appear in our potential $V(\mathbf{r}^{(e^+)}, \mathbf{r}^{(e^-)})$.

2.2 Polarization

We now proceed to calculate the potential energy terms that represent the the dielectric response of the wall. The first step is to find the electrostatic potential a single particle of charge q within a sphere of radius R_c whose dielectric constant k_i, which is embedded within a large (infinite) bulk material of dielectric constant k_o. This is done by solving Poisson's equation with the appropriate boundary conditions; the result is (see the appendix for the calculation)

$$\phi_{\mathbf{r}_s}(\mathbf{r}) = -\frac{q}{R_c}\frac{k_o - k_i}{k_i}\sum_{n=0}^{\infty}\frac{n+1}{nk_i + (n+1)k_o}\left(\frac{r_s r}{R_c^2}\right)^n P_n(\cos\theta) \tag{2.12}$$

where $\mathbf{r}_s$ is the position of the charge q, θ is the angle between $\mathbf{r}_s$ and $\mathbf{r}$, and P_n is the n^{th} Legendre polynomial. The notation $\phi_{\mathbf{r}_s}(\mathbf{r})$ is meant to indicate that this is the potential energy[3] experienced at $\mathbf{r}$ produced by a charge at $\mathbf{r}_s$. It turns out that the potential energy of a charge Q at point $\mathbf{r}$ is $V(\mathbf{r}) = 1/2Q\phi(\mathbf{r})$. The factor of $1/2$ is necessary because when the system of charges is assembled, the

[1] Recall that each bead actually represents the same physical particle, so that the quantum particle is interfering with itself in different imaginary time slices.

[2] Section 3.2 contains the code and an explanation for the quantum system consisting of only one particle in a harmonic oscillator potential, illustrating the theory derived above nicely.

[3] To be absolutely clear, this does *not* include the bare Coulomb interaction between the positron and electron.

bound charge induced in the walls of the cavity does not have work performed on it by the assembler (see (14) or (15) for further explanation).

The total potential in our cavity (whose interior we assume to be a vacuum, so that $k_i = 1.0$) is found from the superposition of the potentials due to each charge. Since each of the positron and electron both produce and experience the potential of Eq. 2.12, there are four terms to the total dielectric energy:

$$U_{pol} = \frac{1}{2}[q_1\phi_{\mathbf{r}_1}(\mathbf{r}_1) + q_2\phi_{\mathbf{r}_2}(\mathbf{r}_2) + q_2\phi_{\mathbf{r}_1}(\mathbf{r}_2) + q_1\phi_{\mathbf{r}_2}(\mathbf{r}_1)] \tag{2.13}$$

where $q_1 = -1$ and $q_2 = +1$ in atomic units are the charge of an electron and a positron, respectively, and $\mathbf{r}_1$ is the position of a positron bead and $\mathbf{r}_2$ is the position of the electron bead in its same imaginary time slice.[1] In Eq. 2.13, the first two terms are to be interpreted as the energy of each particle due to its own induced charge, and as such we call them the "self-energy" terms. See Fig. 2.1 below for a graph of the self energy term as a function of r when $k_o = 3$ and $R_c = 10au$. As expected, they are negative, as can be seen by combining Eqs. 2.12 and 2.13 (and noting that we are examining cases where $k_o > 1 = k_i$). Similarly, the last two terms are each positive, and we call them the "cross-energy" terms because they describe the interaction between the positron and the electron's induced charge, as well as between the electron and the positron's induced charge. We expect the cross-energy terms to be smaller in magnitude than the self-energy terms as each particle will end up on average closer to its own induced charge than the other particle's. (Note that this serves to polarize the positronium, reducing the internal contact density κ.) Also, noting the symmetry between r and r_s in Eq. 2.12, we see that the two cross-energy terms are actually equal.

In our code the infinite sum must be truncated after some finite number of summands. Since the important, size-determining factor in each summand is $(r_s r/R_c^2)^n$, and during most of the computation time this factor is small, since the beads spend most of their time close to the center of the cavity, we expect the

[1] Recall that this is indeed how the path integral works: the kinetic energy term becomes harmonic oscillators between adjacent beads (each in different imaginary time slices) within the same chain, while any potential energy terms act only between corresponding beads on the two different chains.

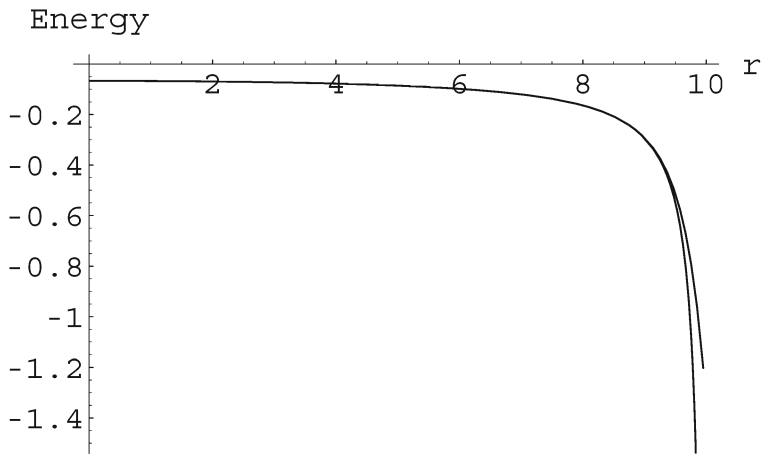

Figure 2.1: Mathematica plot of the self energy expression $\phi_{\mathbf{r}}(\mathbf{r})$ from Eq. 2.12, with $k_o = 3$ and $R_c = 10au$ (energy and distance are both in atomic units here). The lower curve uses 5000 terms in the sum, the upper uses only 25. We see that they differ significantly just for the $0.5au$ closest to the wall. Also note that the better series, the one with 5000 terms, shows Coulombic $1/(R_c - r)$ behavior near the wall: as one of the particles of Ps approaches the wall, its bound charge, which is usually distributed as a charge density, acts more and more like a point charge.

truncated series to approximate the true series quite well for most sets of bead positions realized during the run. This prediction is confirmed by Mathematica calculations: when the particle is farther than $0.5au$ from the cavity wall, the error between the sum with 25 terms and the infinite sum is found to be less than 1% in a cavity with $R_c = 10au$ and $k_o = 3$ (see Fig. 2.1). Thus 25 terms sufficed for most energy calculations performed during a run. Moreover, we have a method for estimating the neglected terms, described below.

We certainly cannot ignore the times that the beads are close to the cavity wall, and the way we handle the approximation is different for the self-energy and cross-energy terms. This is sensible because the self-energy terms have no Legendre polynomials (recall that $P_n(\cos\theta) = 1$ for $\cos\theta = 1$, which is certainly the case for the self energy terms since $\mathbf{r}_s = \mathbf{r}$ for them, and θ is the angle between those two vectors). Referring to Eq. 2.12, we see that the self-energy terms are each nearly a geometric series; since the beads are never allowed outside the cavity, the repeated term, $(r/R_c)^2$, has magnitude less than unity, so the proposed geometric series does indeed converge. In fact, as n gets large, the summands begin acting more and more geometric, as the ratio of n and $n+1$ goes to 1 for large n:

$$\frac{n+1}{nk_i + (n+1)k_o} = \frac{1}{\frac{n}{n+1}k_i + k_o} \approx \frac{1}{k_i + k_o}. \tag{2.14}$$

Thus after truncating the true series after finitely many summands, we applied the geometric correction term using the well-known formula:

$$\sum_{n=m}^{\infty} b^n = \frac{b^m}{1-b} \tag{2.15}$$

where $b = (r/R_c)^2$ for our problem. We took $m = 26$ (so the first 25 terms of the true series were used) and added this correction term even when the beads were close to the center since it is so calculationally inexpensive. This correction term is so good that we need no new figure to demonstrate its success: Mathematica plots the 25 term series with the geometric correction directly on top of the 5000 term series. Thus for our purposes the self-energy term as implemented in the code is *exact*.

However, there is a computational problem with the self-energy terms. It comes from the fact that they represent Coulombic attraction between a bead

and its induced charge, and as the bead approaches the wall, the image charge becomes simply a single point charge on the wall (as the bead approaches the spherical wall, the wall appears to be planar). This can be seen by approximating the *entire* series as geometric (take m=0 in Eq. 2.15) and using $d = R_c - r \ll R_c$ in Eq. 2.12:

$$\phi_{\mathbf{r}}(\mathbf{r}) = -\frac{q}{k_i}\frac{k_o - k_i}{k_o + k_i}\frac{1}{R_c - \frac{r^2}{R_c}} \propto \left(R_c - \frac{(R_c - d)^2}{R_c}\right)^{-1}$$
$$= \left(R_c - R_c\left(1 - \frac{d}{R_c}\right)^2\right)^{-1} \approx (R_c - R_c + 2d)^{-1} = \frac{1}{2d} \tag{2.16}$$

which is the potential of a point charge at position r interacting with another point charge a radial distance $2d$ away from it. That is, the charge and its point charge are each d away from the wall and interact Coulombically (see Fig. 2.1). The same problems that arose from the Coulombic interaction between the positron and electron therefore apply here.

We choose this time not to use the analogous Pollock propagator. Instead, we utilize the Yukawa potential, shown to bring about the desired functionality by Muser and Berne(16). In our case, this involves simply multiplying the entire potential by the factor

$$1 - exp\left(-\frac{R_c - r}{a}\right) \tag{2.17}$$

where a, the Yukawa radius, is to be determined and should be small. Muser and Berne show that a must scale as $(\beta/P)^{2/3}$, where P is the number of beads. We chose $a = 0.132au$ and simply ensured that the ratio β/P was kept fixed for each set of parameters. A power series expansion when the argument of the exponential is small easily reveals that in this case, Eq. 2.17 multiplied by Eq. 2.16 yields a finite number as $r \to R_c$, namely $1/a$. This fixes the divergent problem of the Coulomb potential, and does not significantly affect the value of the potential for $(R_c - r)/a \gg 1$; that is, when the bead is more than a from the cavity wall, which is of course where the beads spend the majority of their time. To confirm that the Yukawa modification does not significantly change the value of the potential energy except very near the wall, we made the following Mathematica plot, Fig. 2.2.

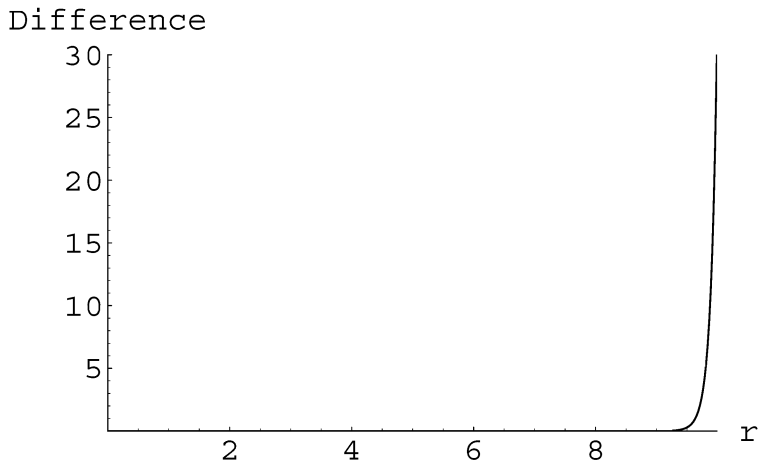

Figure 2.2: The percent difference between the potential with and without the Yukawa factor versus r (in au); again $R_c = 10au$, $k_o = 3$. We see that the percent difference between the true function and the one multiplied by the Yukawa term is less than 1% farther than $1au$ from the cavity wall. The fact that it grows as high as 30% close to the wall is in fact good, as it prevents our code from suffering an unphysical Coulomb-induced collapse of the beads onto the wall.

The cross-energy terms should not have the divergent Coulomb problem since, for example, a positron bead should never approach too closely to the positive charge induced by its corresponding electron bead. However, just in case, and also for consistency's sake, we apply the Yukawa potential to the cross-energy terms as well. Unfortunately the Legendre polynomials that are necessarily present in the infinite series prevent the geometric series correction from being valid; we offer no correction after cutting the series off after finitely many summands.[1] Note that the Legendre polynomials satisfy $-1 \leq P_n(x) \leq 1$ for any x between -1 and 1 (the possible values for $\cos\theta$), so that at least the Legendre polynomials can only to reduce (or at worst, not affect) the magnitude of each summand, thus helping with faster convergence.

Our code calculates the Legendre polynomials pointwise. That is, for each energy calculation it needs to do the code first calculates the value of $\cos\theta$ for the particular positron and electron bead being considered, and then finds the single value $P_n(\cos\theta)$ for all n in the finite sum. This process is accomplished quickly using a recurrence relation for the Legendre polynomials(17) that allows the code to find $P_n(\cos\theta)$ from the values $P_{n-1}(\cos\theta)$ and $P_{n-2}(\cos\theta)$ it just calculated.[2]

While we still may use only 25 terms when the beads are sufficiently far from the wall, within a distance δ from the wall, it becomes necessary to use substantially more terms. The number of terms and δ are both found by studying how quickly the series converges with Mathematica for different cavity radii; typical values are $\delta = 1au$ and 250 terms calculated before truncation. Fortunately this lengthy calculation does not occur too often, but finding all those Legendre polynomials does turn out to be a rate-limiting step.

[1] However, a similar idea could be made to work, since using the same approximation that turns the self-energy sum into a geometric series converts the cross-energy sum to the generating function(17) for the Legendre polynomials. This would be an excellent addition to this project that we did not have a chance to implement.

[2] Beginning the recurrence is not a problem since $P_0(z) = 1$ and $P_1(z) = z$.

Chapter 3

Methods

3.1 The Metropolis Monte Carlo Algorithm

Our quantum statistical mechanical problem would be completely solved if only we knew the partition function $Q = Tr\hat{\rho}$. Then the expectation value of any observable A of interest could be obtained from $\langle A \rangle = Tr(\hat{\rho}\hat{A})/Q$. However, since enumerating every microstate of the thousands of positron and electron beads existing in the pore is clearly impossible, so too is calculating the partition function. This is the reason a computer simulation is the only effective way to obtain equilibrium information about our system.[1]

The Metropolis Monte Carlo (MMC) algorithm, first described by N. Metropolis et al in 1953(18), is the heart of our computer simulation. The method ignores the intermediate goal of calculating Q,[2] instead directly solving for the equilibrium macrostate of the system; that is, it generates the microstates that characterize the system's equilibrium. It does so by allowing the system to explore its phase space until it equilibrates, upon which time the system visits its various available microstates almost exactly in proportion to how physically probable they are to occur. The reason MMC reproduces the microstate probabilities only

[1] Throughout this section, as Section 2.1, the reader is referred to (11) and (12) for additional information.

[2] Recall that the path integral calculation of Q in Section 2.1 was done to show the isomorphism between the quantum positronium and the classical beads. That being done, finding Q as a function of β exactly remains intractable, so we turn to this MMC algorithm.

approximately is both because we only allow it to run for a finite amount of time, and because it is in fact possible that some regions of phase space might be inaccessible from the starting configuration. We must ensure that this second problem does not occur; in practice, this means checking that beads are able to explore all parts of the cavity.[1]

There are two facts immediately to note about the above description. One is that nowhere is there any attempt to describe the dynamics of the system. Therefore the order in which the microstates are obtained during the course of the simulation might be completely different from the order an exact solution would dictate (or that would be followed in a real world experiment). This does not pose a problem so long as we are careful in comparing our results only to experiments that took place in thermal equilibrium.

The second fact to note is that this method makes trivial the procedure for finding the thermal average of an observable. The usual formula is

$$\bar{A} = \sum_n P_n a_n \tag{3.1}$$

where P_n is the probability that the observable A has the value a_n, which occurs when the system is in the microstate n. These probabilities are incalculable because the normalization factor is the intractable Q. But the output data of an MMC simulation is a list of microstates n, where the number of times a particular microstate appears in the list is proportional the probability P_n that the state is obtained by the system. Thus we may calculate the average of A according to

$$\bar{A} = \frac{\sum_n a_n}{N} \tag{3.2}$$

where a_n is the value of A in the n^{th} microstate, and there are N total microstates in our list. Since the a_n are picked from the biased list produced by the simulation, the probabilities P_n are implicit in the above equation.

We proceed now to show how MMC finds this correct list of microstates. The basic step during an MMC simulation is as follows:

[1] One previous version of our code appeared to move the beads preferentially in the negative x and y directions, so we were concerned that in fact the problem described above had occurred. The problem actually turned out to be due to our random number generator and was corrected.

1) At first the system is characterized by a set of parameters; in our case the positions of all the positron and electron beads.

2) A single parameter is proposed to be changed by a certain amount; for example, one of the beads is given a proposed new position (selected at random, but within reasonable limits).

3) MMC decides whether or not to accept the change (move) by conforming to the transition probabilities dictated by the fact that the system is in equilibrium and has particular energies associated with both the initial and proposed states.

4) The entire process is repeated until all parameters have had many chances to be changed several times.

Therefore our goal is to find the transition probability that the system will change from its initial state into the proposed one. Then the simulation will accept the proposed state according to this probability, and the correct list of microstates will be discovered.

We begin by describing how the probability that the system is in a state n, denoted by P_n, changes in time. We must account for the fact that there is a probability P_m that the system is in some other state m at time t, and that the system will move from state m to n with a transition probability per unit time, which is what we are eventually trying to solve for, $W_{m\to n}$. So the total probability that the system is in state m at time t and that it moves to state n in the next instant of time dt is $P_m W_{m\to n} dt$. Similarly, the probability that the system leaves its current state n in favor of state m is $P_n W_{n\to m} dt$. The sum over all possible other states m yields the total change in probability dP_n of being in state n. Thus we have

$$\frac{dP_n}{dt} = \sum_{m\neq n} (P_m(t) W_{m\to n} - P_n(t) W_{n\to m}). \tag{3.3}$$

This equation essentially expresses the conservation of probability. When the system attains equilibrium, by definition this means that the probabilities of being in each microstate have become constant in time, so that the time derivative[1] of P_n must be zero. The only way to guarantee that this is true for arbitrary n is

[1]For our simulation, time is discrete: each

for each term in the sum to be separately zero; thus

$$P_m(t)W_{m\to n} = P_n(t)W_{n\to m} \tag{3.4}$$

must hold for all states m and n. This equation is fundamental to equilibrium, hence must be true of the MMC algorithm, and is known as the condition of detailed balance. Thus in our MMC simulation we impose the condition of being in equilibrium by requiring that the transition probabilities $W_{m\to n}$ obey the above equation.

This still affords us some freedom in picking the $W_{m\to n}$, so the fact that systems tend toward states of lower energy may be enforced by putting $W_{m\to n} = 1$ whenever $E_n < E_m$, where E_n is the system's energy when it is in state n. Requiring detailed balance to hold, ie obeying Eq. 3.4, then gives a complete description of the desired transition probabilities:

$$W_{m\to n} = \begin{cases} 1 & \text{if } E_n < E_m \\ \frac{P_n}{P_m} & \text{if } E_n > E_m \end{cases} \tag{3.5}$$

Recall that statistical mechanics informs us that $P_n = \frac{1}{Q}e^{-\beta E_n}$, and Q is incalculable. However, in the above formula for the transition probabilities $W_{m\to n}$, we see that only the ratio between two probabilities is needed, in which case the unknown Q cancels out. This fact alone is what makes MMC such a useful procedure.

We may rewrite the above by using $P_n = \frac{1}{Q}e^{-\beta E_n}$ and defining $\Delta E = E_n - E_m$:

$$W_{m\to n} = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{-\beta \Delta E} & \text{if } \Delta E > 0 \end{cases} \tag{3.6}$$

It is now clear that the only calculation required is of the energy difference ΔE between successive configurations of the system. In our case, note that ΔE is the energy difference between configurations of the classical bead system, *not* between the energies of the actual quantum system!

One practical concern remains, namely that the above prescription relies on the computer choosing to accept the move (that is, make the transition from state m to n) with a certain probability. If $\Delta E < 0$ there is no problem since then the proposed move is always chosen. However, if $\Delta E > 0$, there seems to be a problem. The computer may not be presented with the opportunity to make

this same move ever again, so that it cannot simply choose to accept it a fraction $e^{-\beta\Delta E}$ of the times it is presented with the choice. Even if this were the case, the computer can only make "local" decisions about moves, as they occur, without "global" knowledge of the number of times that particular move will be proposed. Thus implementing a probabilistic decision in this manner is impossible.

The solution to this quandary is to make use of random numbers once again (recall that they are also used in generating a proposed move). There are ways to generate random real numbers between 0 and 1 so that each such number has an equal chance of being generated (we use a linear congruential method(12)). That is, the random number chosen is sampled from the uniform distribution on the interval [0,1]. This means that, given a certain fixed number x between 0 and 1, the probability that a random number ξ between 0 and 1 is picked so that $\xi < x$ is simply x. This follows because the fraction of numbers between 0 and 1 that are less than x is clearly x, and since ξ is equally likely to be any number between 0 and 1, the result $Prob(\xi < x) = x$ is obtained.

Thus, if we would like an event to occur with probability x, all we must generate the random number ξ; then if $\xi < x$, allow the event to be accepted, and if not, reject the event. In our case we let $x = e^{-\beta\Delta E}$. Then after the computer calculates ΔE, it also generates a random number ξ and in the above manner accepts the proposed move with the correct probability so that detailed balance is maintained. A large number of moves should be performed so as to approach the exact statistical limit (infinite moves) as much as possible. In the results to be quoted in this thesis, we typically performed around 400,000 moves, enough so that approximately twice as many are performed as are accepted.

3.2 A Simple Example of PIMC

"PIMC" is the use of Monte Carlo methods (in our case, Metropolis Monte Carlo) to sample positions of the classical chains of beads described in Section 2.1. As a concrete illustration of this procedure that was discussed theoretically at length above, we shall study a code that simulates a single quantum particle in a one-dimensional harmonic oscillator potential. The methodology described above

is much more evident here than in the full positronium code, for the following reasons:

1) It models only one particle in a single dimension

2) It has a smooth, continuous potential energy function; namely the potential has no hard walls and does not diverge anywhere (as is the case in the Ps code).

3) The energy states of this system are known and are analytically calculable.

Thus we need not employ many of the computational tricks (described in Section 3.3) that are used in the full code and that obscure the essential PIMC procedure at work. This code is included here for easy reference.

```
program pimc_harmosc

! Using the Metropolis algorithm, we'll calculate the average energy of a
! particle in a harmonic oscillator potential at various temperatures.
! Feynman's path integral method is employed, so that the system is
! characterized by iP "beads" on a chain, each in a harmonic potential.
! We have chosen the units so that Boltzmann's constant and hbar are
! both equal to 1. The particle's mass is also chosen to be 1 in those units.

implicit none

integer, parameter :: iP = 30, nruns = 10 ! number of beads, different temps
integer, parameter :: nmoves = 10000, iseed = 1
           ! # of MC steps, seed for random # generator
real, parameter :: omega = 1.0, xrange = 1.0

integer :: i, j, k, l, acc_count = 0, imove
           ! indices, then a counter for how many moves are
           ! accepted...I may or may not use it
           ! the "movee" is the guy thinking about moving

double precision :: kappa ! spring constant for bead-bead coupling
real :: step = 0.2 ! scale of the movee's contemplated jump

real, dimension(nruns) :: T ! temperature
double precision, dimension(nmoves, nruns) :: energy = 0.0
double precision, dimension(nruns) :: av_energy = 0.0
double precision, dimension(iP+1) :: x
           ! position of the ith bead...having it be one
           ! bigger than it needs to be is convenient for the
           ! do loops...but must have x(iP+1) = x(1)

call init()
open(unit=17, file='pimc_data', action='write')
```

```
write(17,*) '  Energy              ', 'Temp', '      # accepted moves'

do k=1,nruns
kappa = iP * (T(k)**2)
energy(1,k) = ecalc(T(k), kappa)

do l=1,20          ! just let it cook for 20 times, so it has time to explore
acc_count = 0
do j=2,nmoves
! time to pick a bead to move
imovee = 1 + (iP - 1)*(ran1(iseed))
call move(imovee, T(k), kappa)

energy(j,k) = ecalc(T(k), kappa)
end do
end do

do i=nmoves/2,nmoves      ! average over the latter half of the energies found
av_energy(k) = av_energy(k) + (energy(i,k) / (real(nmoves)/2.0))
end do

write(17,*) av_energy(k), T(k), acc_count
end do

close (17)

contains

! Amy's random number generator
double precision function ran1(idum)
implicit none
double precision ::  r(97)
integer, intent(IN) :: idum
save
```

```
integer, parameter :: M1=259200,IA1=7141,IC1=54773
real, parameter :: RM1=1.0d0/M1
integer, parameter :: M2=134456,IA2=8121,IC2=28411
real, parameter :: RM2=1.0d0/M2
integer, parameter :: M3=243000,IA3=4561,IC3=51349
integer :: IX1, IX2, IX3, jjj
integer ::  iff=0
if (idum < 0 .or. iff == 0) then
  iff = 1
  IX1 = mod(IC1-idum,M1)
  IX1 = mod(IA1*IX1+IC1,M1)
  IX2 = mod(IX1,M2)
  IX1 = mod(IA1*IX1+IC1,M1)
  IX3 = mod(IX1,M3)
  do jjj = 1,97
    IX1 = mod(IA1*IX1+IC1,M1)
    IX2 = mod(IA2*IX2+IC2,M2)
    r(jjj) = (dfloat(IX1)+dfloat(IX2)*RM2)*RM1
  end do
end if
IX1 = mod(IA1*IX1+IC1,M1)
IX2 = mod(IA2*IX2+IC2,M2)
IX3 = mod(IA3*IX3+IC3,M3)
jjj = 1+(97*IX3)/M3
if (jjj > 97 .or. jjj < 1) PAUSE
ran1 = r(jjj)
r(jjj) = (dfloat(IX1)+dfloat(IX2)*RM2)*RM1
end function ran1


subroutine init
implicit none

do i=1,iP
```

```
x(i) = 2.0 * xrange * (ran1(iseed) - 0.5)
! randomly places beads between -xrange and xrange
end do
x(iP+1) = x(1) ! really the same bead!

T(1) = 0.1
do j=2,nruns
T(j) = T(j-1) + 0.1 ! linearly scale up the different temperatures
end do

end subroutine init


double precision function ecalc(T, kappa) ! the energy calculator
implicit none

real, intent(in) :: T
double precision, intent(in) :: kappa

double precision :: kinetic, V
kinetic = 0.0
V = 0.0
do i=1,iP
kinetic = kinetic + 0.5 * kappa * (x(i+1) - x(i))**2
V = V + 0.5 * omega**2 * x(i)**2
end do

!ecalc = 0.5*T*iP + V/iP - kinetic
          ! the origin of the minus sign and 1/2kT is subtle...you have to
          ! track down all the beta dependences in Z
ecalc = V/iP
end function ecalc
```

```
subroutine move (imovee, T, kappa)
implicit none

integer, intent(in) :: imovee
real, intent(in) :: T
double precision, intent(in) :: kappa
double precision xtemp, delta_E, xless, xmore

xtemp = x(imovee) + step * 2.0 * (ran1(iseed) - 0.5)
                    ! randomly move between -step and step away

if (imovee==1) then
xless = x(iP)
else
xless = x(imovee-1)
end if

xmore = x(imovee+1)

delta_E = 0.5 * omega**2 * (xtemp**2 - x(imovee)**2) / iP + 0.5 * kappa * &
      ( ((xmore-xtemp)**2 + (xtemp - xless)**2) - &
      ((xmore - x(imovee))**2 + (x(imovee) - xless)**2) )
! the only change is in the interaction with the two nearest beads to the movee,
! and of course its own potential...no need to recalculate the whole energy
! Also note that delta_E is the change in the BEADS' energy!

if (delta_E .LE. 0.0) then ! do the move!
x(imovee) = xtemp
acc_count = acc_count + 1

else ! the heart of Metropolis
if ( ran1(iseed) .LE. exp((-delta_E) / T)) then
x(imovee) = xtemp
acc_count = acc_count + 1
```

```
else
delta_E = 0.0
end if
end if

x(iP+1) = x(1)

end subroutine move

end program pimc_harmosc
```

As in the main program, here we employ a unit system in which significant physical constants are equal to one; here, those constants are $\hbar$, the mass of the particle, and Boltzmann's constant.[1] This practice serves to make the physical parameters that characterize the system all have order unity, which is very convenient when dealing with output data files.

First, the code declares its variables (iP is the number of beads, *omega* is the angular frequency of the harmonic oscillator; the rest have self-explanatory names). Similarly to the positronium code, there is a subroutine, here called *init*, set up to initialize certain variables. Within *init* the positions of all the beads are chosen randomly within a given range. Also, the array of temperatures is initialized here: unlike in the main code, each time the harmonic oscillator program is run, the simulation is performed for a variety of temperatures. This is a reasonable feature because this simple program has such a short runtime.[2] After initialization, an output file in which to store the data is set up (in contrast, the positronium code has seven output files to record data on quantities such as the relative radial positron density, the absolute Cartesian positron density, the pickoff lifetime, and the orientation of Ps).

The effective spring constant of the kinetic energy term in the partition function, κ, depends on the temperature of the system and thus must be recalculated with each new run. The function *ecalc* calculates the actual energy of the system. This turns out *not* to be the same as the energy of the bead configuration,[3] and is instead found from the statistical mechanics formula

$$E = -\frac{\partial Q}{\partial \beta} \tag{3.7}$$

where Q is the partition function.

The main procedure of PIMC is found in the subroutine *move*. After the particular bead to be moved (*imovee*), as well as where it should be moved, are

[1] In the main code we use standard "atomic units" in which $\hbar = e = m_e = 1$, where e is the charge of an electron and m_e is the electron mass. As an interesting aside, due to the fact that the fine structure constant α is dimensionless, it turns out that in order for this unit system to be consistent, the speed of light must equal (approximately) 137.

[2] On a Macintosh G4, each of the three sets of data displayed at the end of this section took approximately 10 minutes to produce.

[3] See (13) or (19) for a discussion of how to estimate the energy for the quantum system.

chosen (both done at random), the program supposes that it has in fact made this move (hence *xtemp*) and calculates the difference between the energy[1] of the proposed configuration and the current one. According to the Metropolis procedure described above, the program chooses whether or not to actually perform the move, and updates the beads' positions accordingly. For computational efficiency only three terms of the whole energy are changed when a single bead moves: the spring terms connecting it with its two neighbors, and the potential energy of that bead. Thus we do not call the full *ecalc* function to find ΔE.

Aside from recording the average energy over all the configurations obtained during a particular run, there is nothing left to do. However, it should be noted that PIMC is in general not a good simulation technique if one is primarily interested in accurate predictions of a system's energy. The energy was noted in this program merely as a test, and in fact even after a large number of moves when the system was certainly in equilibrium, the energy values, calculated using the subroutine *ecalc*, did not settle to a particular value. We present some data taken from the program to illustrate this point.

	nruns	*step*	T	mean energy
Data 1	10	0.5	0.2	0.36(3)
Data 2	20	0.5	0.1	0.25(1)
Data 3	10	0.2	0.1	0.26(3)

[1]Since we are supposed to be finding the equilibrium state of the beads themselves, we use the beads' energy here, not the actual energy of the system as calculated in Eq. 3.7. The beads allow us to find the system's equilibrium state by evolving as if they were themselves a real statistical mechanical system.

Each of these data sets was taken with $nmoves = 10000$ and $iP = 30$; that is, with 10,000 chances to move a bead, and with 30 beads on the chain. Using the exact analytical solution to the one dimensional quantum harmonic oscillator, we included the first four excited states in our theoretical calculation of the expected thermal average of the particle's energy, obtaining a value of $0.295au$. As can be seen from the above data, PIMC produces energy values that are certainly not absurdly wrong, but also are not as close to the true value as one would like.

However, it turns out that unlike the energy, the binned averages of bead positions in a PIMC simulation do settle down, and since in the positronium project we care only about where the positron beads are in relation to the electron beads or to the cavity wall, PIMC is a good method for our purposes.

3.3 Approximation Methods for Positronium

In principle the above description of the Path Integral Monte Carlo method is complete, and one might think that following it exactly for arbitrarily large values of P would indeed generate microstates with the correct equilibrium probabilities. In practice, however, several modifications need to be made so that the computer program runs at all, and also so that the runtime of the program is not prohibitively long.

One reason that the program may not function as it should, given the methodology formulated above, is that we can only use finitely many beads. One would hope that using a large enough number would solve any problems that might arise, but it turns out that the Coulomb potential's $1/r$ dependence is so strong that with only finitely many beads, the two chains (representing the positron and electron) collapse into each other.(16) Informally, one can imagine that with infinitely many beads in each chain, every time two corresponding beads were attracted to each other Coulombically, they would have infinitely many harmonic oscillator potentials pulling them back, which would prevent collapse; the Coulomb potential can prevail over any finite number of harmonic oscillators, though.

There are several strategies to deal with this problem (such as the Yukawa potential we applied to the dielectric energy terms), but a standard successful

one is to follow an idea of Pollock's: factor out the Coulomb part of each bead's contribution to the total density matrix.(20) We write

$$e^{-\beta\hat{T}}e^{-\beta\hat{V}} = e^{-\beta\hat{T}}e^{-\beta\hat{V}_{ext}}e^{-\beta\hat{C}} \tag{3.8}$$

where $\hat{V}_{ext}$ is the potential energy due to anything but the Coulomb interaction (in our case, it is the term for both the hard wall and the dielectric response of the material). The operator $\hat{C}$, known as the Pollock propagator, is defined by Eq. 3.8; note that it is not necessarily equal to $-e^2/|\hat{\mathbf{r}}^{(1)} - \hat{\mathbf{r}}^{(2)}|$ because $\hat{V}$ and $\hat{C}$ need not commute, so the exponential does not necessarily factor in the usual way. Putting the factorization of Eq. 3.8 into Eq. 2.9, we see that the formalism will proceed as described there, but with $\hat{V}$ replaced by $\hat{V}_{ext}$, and multiplied by corresponding matrix elements of $\hat{C}$. These matrix elements, which amount to the exact solution of the pure Coulomb problem (so that we have cured the pathology by knowing the answer!) have been previously computed and tabulated.(20)

The next modification we make to the existing procedure is for the important sake of reducing runtime, so that the calculation is feasible. The problem is that while proposing completely random moves for the beads would indeed eventually cause the system to find its equilibrium microstates, as it does for the small values of P used in the harmonic oscillator code of Section 3.2, it would take an exorbitant amount of time to do so for large P, as is required in the positronium code. This follows because most of the random moves would be bad guesses as to what the aggregated beads "want" to do, and thus would be rejected by our algorithm. The other problem is that even if we managed to make better guesses so that more moves were accepted (indicating approach to equilibrium), it is still imperative that the system be allowed to explore all of its available phase space. Due to the fact that each chain acts like a polymer, it is very difficult to move the center of mass of a chain simply by moving individual beads; no particular bead will be able to venture away from the mass of beads connected to it via the attractive harmonic oscillator potentials.

The solution to the second problem is simple once the first is solved: every few hundred individual bead moves, we perform a center of mass move, in which the positions of every bead in one of the chains are proposed to be translated by a fixed amount. The move functions no differently from individual bead moves in

that it is accepted or rejected based on the change in energy it would cause (note that there is nothing in the Metropolis Monte Carlo formalism that prohibits this - it defines a move simply as something which changes the system's state and thus its energy). The direction and magnitude of the proposed move is random, but we implement an adaptive algorithm so that if "too many" CM moves are being accepted, we make the proposed move distance longer, and if too many are being rejected, the proposed move distance is made shorter.

We employ the same adaptive method for the individual bead moves, which does help in reaching thermal equilibrium in a timely fashion. However, there is yet another strategy to reduce runtime in two ways: by making more intelligent proposed bead moves, and by reducing the time it takes it calculate the energy for each configuration. This is accomplished by noting that when performing each integral of Eq. 2.9, there are two Gaussians involving the variable of integration. Following Pollock and Ceperley(21), we find that there is a way to manipulate these terms in order to produce a single Gaussian for each bead, in which (temporarily) the positions of the other beads are fixed. If the proposed move points are selected randomly not from a uniform, unbiased distribution, but instead from the appropriate Gaussian distribution, they will automatically obey the restrictions of the kinetic energy propagator. In other words, the kinetic energy can be thought of as putting a simple Gaussian bias on the moves that would be accepted based on the potential energy terms alone, so we can respond by picking moves from that biased distribution to begin with. Thus the kinetic energy term (that is, the interaction spring energy between adjacent beads) may be left out of the calculation of the configuration's total energy.

One other characteristic of the system that would normally prevent a short runtime is the problem of dealing with the cavity wall. In principle, the potential this adds is simple: it is zero within the cavity and infinite at the cavity wall and beyond.[1] However, if a bead doesn't "know" about the pore wall until it

[1]There is some debate (see, for example, (22) on whether an infinite wall is really appropriate, that perhaps a finite wall is more realistic. The infinite potential is meant to model the fact that the atomic nuclei at the wall are effectively impenetrable to the positronium. Certainly the error made in this assumption is no more than the error we make below in assuming that the bulk dielectric material outside may be treated by continuum electrostatics.

tries to move outside the pore, and that move is of course rejected, then it will take a substantial amount of moves for the chain to "realize" that it has much more available phase space toward the center of the pore, and thus in thermal equilibrium will have many more microstates with beads away from the wall. Instead, we use methods introduced by Barker(23) and refined by Kalos and Whitlock(24), among others, to cause the presence of the wall to make itself known to the beads even when they and their proposed moves are both within the cavity.

We now discuss some of the technical details of how this simulation was written and executed.

All computer codes used were written in the Fortran 90/95 programming language. We used the Absoft environment both to edit and to compile the code. The executable file generated by Absoft was run using Terminal, which is the Unix shell that is part of Mac OSX. On a Macintosh G5, the simulation of Ps with $R_c = 10au$, $\beta = 200$, $P = 800$ beads, and 500,000 staging passes took approximately 6 hours. The data was graphed and analyzed primarily with Kaleidagraph.

Two basic codes were utilized: one of them simulates only a positron in a spherical dielectric cavity, the other simulates positronium within such a cavity. The differences between the two are slight: the positron code has no Coulomb interaction between the positron and electron, and the cross energy terms are absent. Therefore the positron code actually simulates two quantum particles, of opposite charge, but which do not interact with each other in any way. As a bonus, two independent sets of data are obtained whenever this code is run.

The working backbone of the code, the PIMC two-particle simulation of positronium in a spherical cavity, has been in development by the Bug research group since summer 2000. Introducing the polarizable cavity was the new contribution to this code that is described in this thesis. Thus, we will focus on results that stem from a polarizable cavity in the following sections.

Output files contained data obtained from the beads' positions (ie absolute radial positron wavefunction, positronium orientation, etc.). These positions were collected by a binning method: we subdivided the cavity dimensions into small "bins," and recorded a "hit" in a particular bin each time a bead's position

coordinates fell within its range. Recall from the partition function isomorphism that these bead positions correspond to possible locations of the actual positron, so by binning in this way, we actually built up the positron's thermally averaged density, $n_+(\mathbf{r}) = |r\psi(\mathbf{r})|^2$, where here ψ is the positron's wavefunction.

We adjusted the temperature based on the cavity radius we were currently running to ensure that the Ps remained in its ground state in order to afford comparison to Tao-Eldrup, as well as to simplify our data (varying k_o and R_c already provides a wealth of possible data to take).

Chapter 4

Results

4.1 Summary

Our results are focused on how the pickoff lifetime of Ps is affected by additionally considering the polarization of the surrounding material. As we have mentioned in Section 1.1, experimental data might profitably be analyzed by recognizing this effect on lifetime. We will compare our findings with the results of the two-particle PIMC model of Ps with $k_o = 1$ (that is, when polarization is not taken into account), with the standard and ubiquitous Tao-Eldrup model (which we summarize below for completeness), with another model, and with experimental data.

As mentioned in Section 1.1, the Tao-Eldrup model is a simple way to relate the positronium's lifetime to the radius of the spherical cavity in which it annihilates. It treats Ps as a single quantum particle, with mass twice that of an electron, in a spherical infinite well, with the assumption that the temperature is low enough that the Ps is entirely in its ground state. An additional simplifying assumption, which we adopt in our work as well, is that the molecular electrons of the cavity wall may be treated as if they extended, with uniform density, a small distance $\Delta = 0.166nm$ into the cavity proper. This idea, as well as the particular value of Δ, is not derivable from theory, but is instead an empirical fact which allows the TE model to fit experimental data well (in the low temperature, small cavity regime to which it applies). Thus the general equation Eq. 1.7 reduces to

the following under the TE assumptions:

$$\Gamma_{p.o.} = (2ns^{-1}) \int_{r=R_c-\Delta}^{r=R_c} n_+(\mathbf{r}) d^3\mathbf{r} \tag{4.1}$$

where $n_+(\mathbf{r})$ is the positron density.[1] The integral is analytically solvable for a single particle in a box. Doing so yields the Tao-Eldrup equation, Eq. 1.1:

$$\Gamma_{p.o.} = (2ns^{-1}) \left[\frac{\Delta}{R_c} + \frac{1}{2\pi} \sin\left(2\pi \frac{R_c - \Delta}{R_c} \right) \right] \tag{4.2}$$

Here R_c is the full cavity radius, out to the nuclei of the surrounding atoms. This is a sensible quantity to deem the radius of the cavity, since the positron is repelled by the nuclei (like the TE model, we treat this repulsion as an infinite wall). However, in most papers it is customary to denote by 'R' the distance from the center to the electron layer, so that $R + \Delta$ is what we would call the full radius. The difference is crucial to remember when trying to compare results with experiment or other models.

In our two particle model of Ps with a polarizable cavity, we must perform the integral of Eq. 4.1 numerically. We first divide the total number of hits in the bins between $R_c - \Delta$ and R_c by the total number of hits in all bins; this gives the probability that the positron is to be found within the "annihilation zone"; in other words, that *is* the value of the integral, so we simply multiply by $2ns^{-1}$ and invert to obtain the pickoff lifetime.

We now turn to some of the general, qualitative features of the results produced by the simulation. These features were observed from many sets of data taken through various stages of the project, not all of which is reported here. These are actually the main results of this thesis; the data that is included afterwards may be regarded as a sample set of the multitude of data that could be taken with our code.

Any value of k_o greater than unity had the effect of attracting both particles of positronium to the cavity wall relative to their $k_o = 1$ propensity to be there.

[1] The prefactor of $2ns^{-1}$ is the spin-average annihilation rate of Ps, reflecting the fact that when the positron is picked off, it first enters either a triplet (with probability 3/4) or singlet (with probability 1/4) state with the picking off electron, and this affects the exact numerical value of its lifetime.

In the case of a single positron within the cavity, the effect was noticeably greater; this is due to the fact that only a single attractive self energy term was present. With a positronium atom inside the cavity, each particle is attracted by its own polarized charge, but is repelled by the charge induced by the other particle. The attraction is diminished, but not overcome, by the repulsion. Of course, the positron being pulled toward the wall results in a higher probability of pickoff annihilation, and thus a shorter lifetime. These results are illustrated in the following figures.

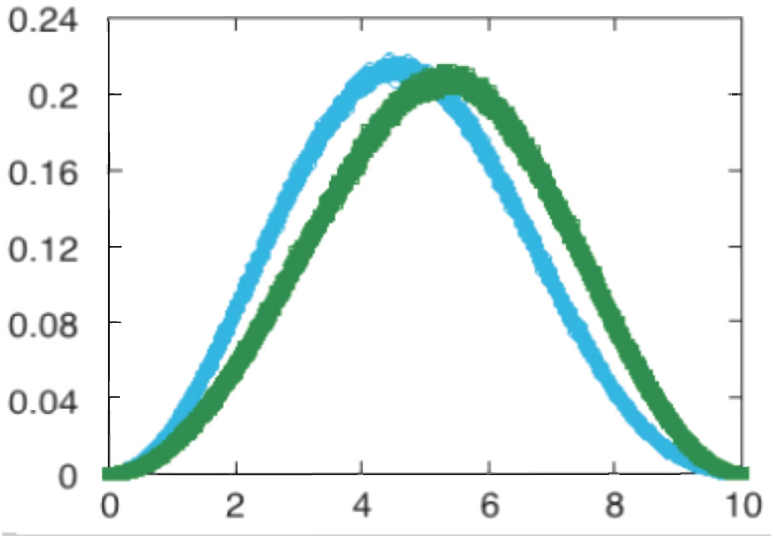

Figure 4.1: The positron density, as a function of absolute radial distance from the center of the cavity, for $R_c = 10au$. The curve on the left (blue) is the case $k_o = 1$, and the other (green) is $k_o = 15$ (we use the high dielectric constant of 15 to illustrate the effect more vividly).

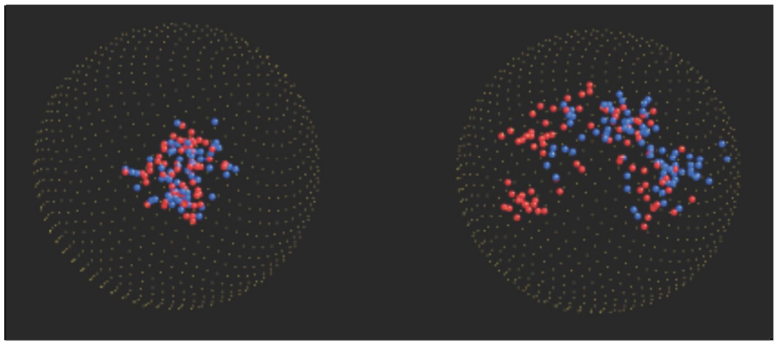

Figure 4.2: A 3-dimensional visualization of the Ps beads at equilibrium; again $R_c = 10au$ for both, $k_o = 1$ on the left, $k_o = 15$ on the right. Clearly each particle, both the e^+ and e^-, are attracted to (different parts of) the wall, and the atom as a whole has been pulled apart slightly in the right figure compared to the left (the latter effect is discussed below). Note that in fact thousands of beads were used in the calculation; visual clarity dictates that here we show only a few dozen.

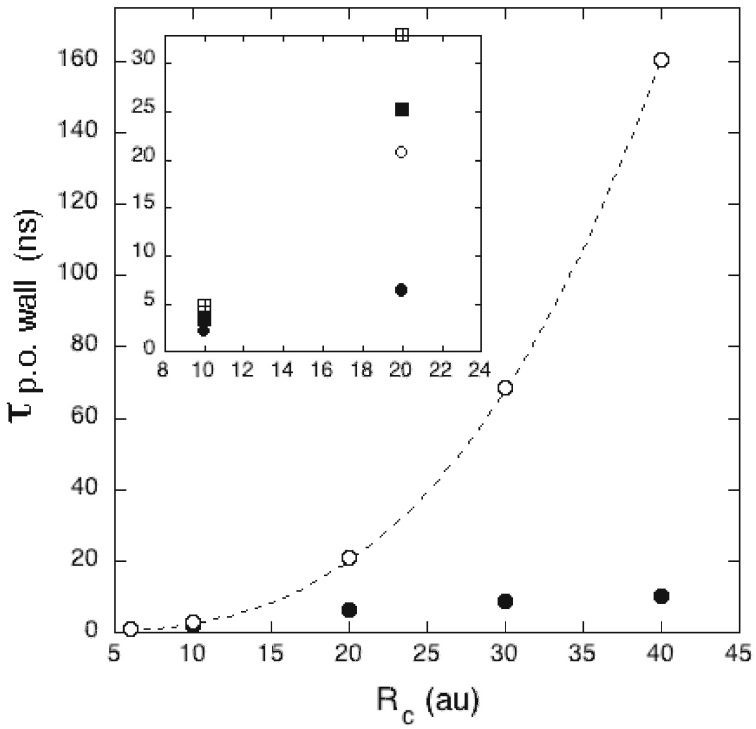

Figure 4.3: Pickoff lifetime as a function of the pore radius. The main part of the figure shows the lifetime of a bare positron, with open circles for $k_o = 1$, filled for $k_o = 3$. The inset repeats this data, and adds squares to show the lifetime of Ps (open is $k_o = 1$, filled $k_o = 3$ again).

The lifetime increases with larger cavity radii, whether we include polarizability or not, for the pickoff zone remains a fixed Δ away from the edge, so it occupies a smaller fraction of the total cavity volume as the radius increases. However, the limits as $R \to \infty$ differ dramatically between the cases $k_o = 1$ and $k_o > 1$. For in the former case, the pickoff lifetime may increase without bound; the Ps simply remains in the center of the cavity, and probably succumbs to self annihilation first (in the case of a single positron, the analytical result is that the lifetime increases as the volume, R_c^3). However, as Fig. 4.3 shows, whenever the cavity walls are even slightly polarizable, the positron or Ps enters into a bound state with the wall, causing the pickoff lifetime to approach a finite limit as the radius grows. Again in the case of a positron, an analytical result may be

found by solving the Schrödinger equation where the potential is that of a flat, dielectric wall (which the spherical wall appears to be as $R_c \to \infty$), assuming that the temperature is kept low enough that the ground state prevails. In the case $k_o = 3$, pickoff lifetime limit is $11.12ns$; compare with Fig. 4.3.

In a similar vein, the lifetime decreases with increasing dielectric constant, and this effect also approaches a limit. This fact is not immediately obvious, but is plausible given that the dielectric term scales as $(k_o - 1)/(k_o + 1)$. This effect is more noticeable for larger radii cavities, as then there is more room for the Ps to exhibit a preference for being near the wall in the presence of polarization. For example, for $R_c = 6au$[1], the lifetime decreases from 1.10(5) to only 0.99(1) ns as the dielectric constant increases from 1 to 3. However, for a cavity with radius $10au$, the decrease in lifetime is from 4.8(2) to 2.7(1)ns for the same range of dielectric constants (see the Table in Section 4.2 below).

While we have so far focused exclusively on, and will continue to concentrate on, the pickoff lifetime, the effect our dielectric correction has on the self annihilation rate deserves some mention. Recall that the standard way to measure this effect is to report values for the internal contact density, κ (Eq. 2.11). It turns out that in small cavities, the fact that our walls are infinite barriers causes κ to exceed its vacuum value of 1, which it never does in reality. However, given that offset, we can still compare the cases $k_o = 1$ and $k_o > 1$. As was hinted at above, a larger dielectric constant tends to reduce κ, as the Ps atom is slightly pulled apart as the positron and electron are attracted to their own, different, sections of the wall. The surrounding dielectric material in effect shields the Coulomb attraction between the two particles. Again the effect is more prominent in larger cavities: in a pore with $R_c = 20au$, we find that $\kappa = 0.96(1)$ for $k_o = 3$, versus $\kappa = 1.00(1)$ for $k_o = 1$ (Fig. 4.2 shows this result qualitatively).

[1] An atomic unit (au) of length is one Bohr radius, which is approximately half an angstrom. As mentioned in another footnote, the atomic unit system we employ is created by setting $\hbar = m_e = e = 1$.

4.2 Data

Here we present the calculated pickoff lifetimes of positronium within three small cavities over a range of 5 different dielectric constants. Note that most materials that would interest experimentalists have dielectric constants in the range of 2-3. The table and two figures contain the same data, displayed differently to emphasize different effects. Note the complicated relationship between TE and our results displayed in the table: no particular dielectric constant causes our results to agree exactly with TE.

	$R_c = 6au$	$R_c = 8au$	$R_c = 10au$
TE	0.920	1.720	3.010
$k_o = 1$	1.123(6)	2.512(2)	5.0(4)
$k_o = 2$	1.06(1)	2.18(4)	4.1(2)
$k_o = 3$	1.034(8)	2.03(5)	3.532(8)
$k_o = 5$	1.005(4)	1.896(8)	3.0(1)
$k_o = 15$	0.985(2)	1.72(4)	2.66(4)

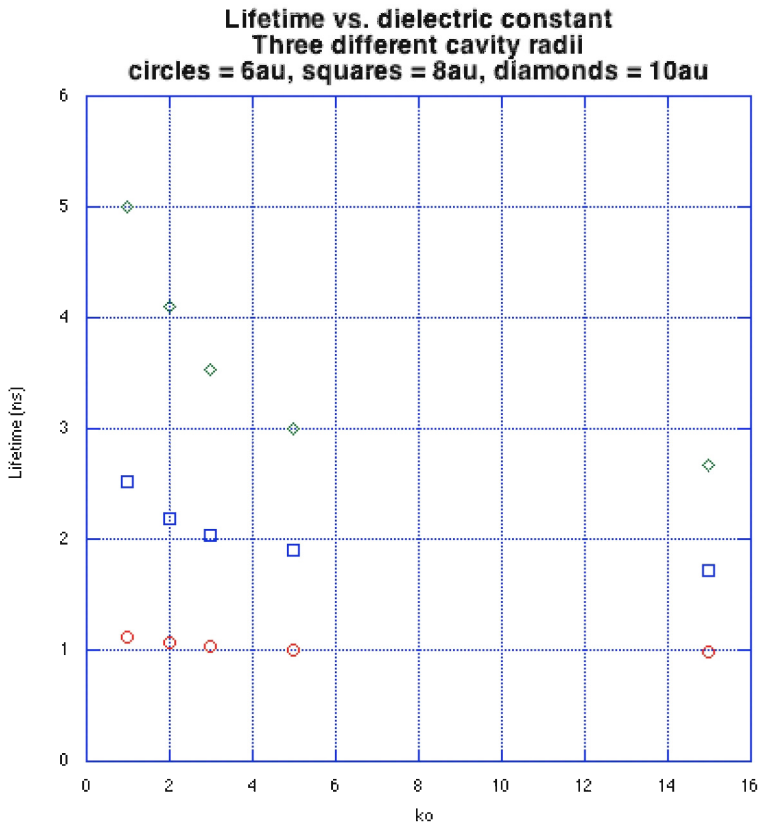

Figure 4.4: Lifetime vs. dielectric constant for radii of 6, 8, and 10au. Notice the leveling off as $k_o \to \infty$, and that the curve for $R_c = 10au$ (the top one, diamonds) shows much steeper change in the region $1 < k_o < 5$ than the other two; in fact, the $R_c = 6au$ curve barely varies with changing k_o.

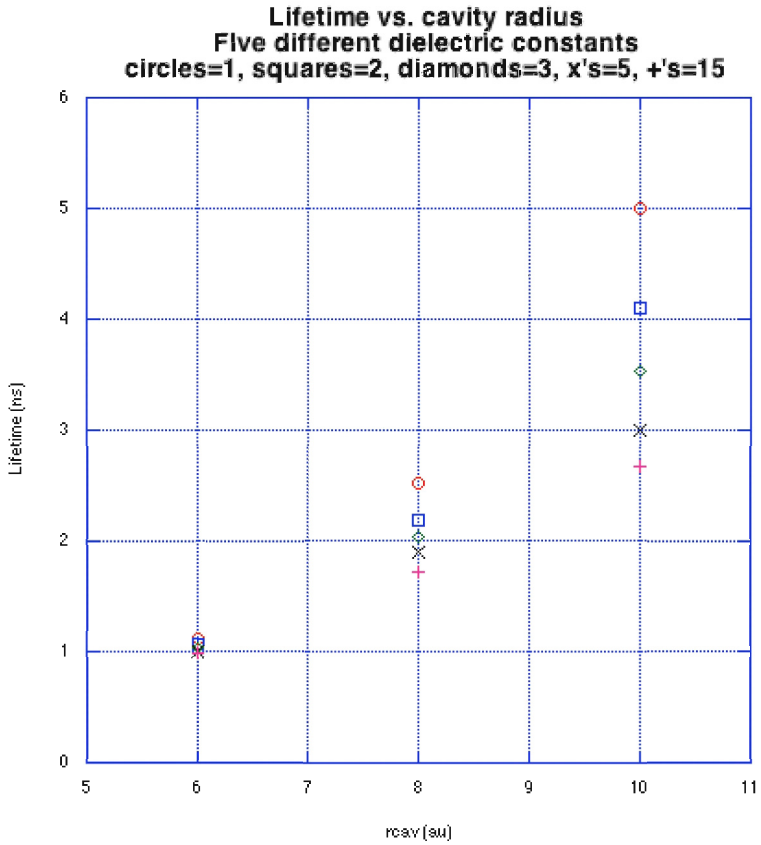

Figure 4.5: Lifetime vs. cavity radius R_c for dielectric constants of 1, 2, 3, 5, and 15; the same data as in Fig. 4.4, but organized in the converse way. The fact that the points are more spread out as R_c increases is indicative of this fact that larger cavities show the effect of dielectric constant more dramatically.

4.3 Comparisons

It was mentioned in the Section 1.1 that another factor that may be responsible for discrepancies among the lifetimes in different materials with the same pore sizes is that the pores in these various materials are shaped differently. We have dealt exclusively with spherical pores in our project, but certainly this is a simplification; in reality, micropores are almost certainly irregular in shape, and may be closer to cuboidal than spherical. A recent paper(6) of Consolati's examines the effect of pore geometry on the relationship between lifetime and cavity size.

Consolati attacks the problem from the experimentalist's viewpoint: given a Ps lifetime, he shows what size the pore was for different assumptions of pore geometry. The geometries he considers are spheres, cylinders, layers (the space between two infinite planes), cubes, and cuboids (cubes with one dimension a scaling factor of s different from the other two equal dimensions). In all of these, the dielectric constant of the bulk material is tacitly assumed to be unity. The cylinders and layers are not relevant to our project, as they are essentially two dimensional and represent the geometries of open pores: pores that are connected to the environment exterior to the material in question. Consolati has data for four lifetimes: 1.99(6), 12.0(4), 12.80, and 48.87 ns. Our results may be compared most directly to the $1.99ns$ data. We list his results in Table 2 below:[1]

Shape	Relevant dimension in au.
Sphere	17(1)
Cube	16(1)
Cuboid, $s = 0.35$	23(2)
Cuboid, $s = 0.5$	20(1)
Cuboid, $s = 2$	15(1)

Note that the first two cuboids are thinner in the third dimension than their other two, whereas the last cuboid is twice as long in its third dimension. To

[1]The "relevant dimension" is the sphere's diameter, the cube's side, and the side of the cuboid's square cross section (a cuboid has dimensions (a, a, sa); the number listed here is a)

compare to our cavity radii, all of the above numbers should be halved. Doing this, we see that, given a $2ns$ lifetime, Consolati would predict a cube with side $16au$, while we would predict a sphere of diameter $16au$ in a material with dielectric constant $k_o = 3$. This shows that the two effects are comparable in magnitude. It is difficult to make other direct comparisons, as our results are obtained by picking a cavity radius as input, whereas Consolati picks a lifetime as input. However, we note that our results (the Table in Section 4.2) report that in a radius $10au$ spherical cavity, the lifetime drops 47 from its value when $k_o = 1$ to its value when $k_o = 15$, and similarly Consolati's results show a 35 decrease from the largest possible cavity size (the $s = 0.35$ cuboid) to the smallest size (the $s = 2$ cuboid). Clearly different values of s and k_o would produce different percentages, but again we see that the two effects are comparable in magnitude.

Thus, as we stated in Section 1.1, both different dielectric constants and different cavity shapes are possible satisfactory explanations for different Ps lifetimes in cavities which gas adsorption determines to be the same size.

As a final analysis of our results, we compare to data taken from the same paper(2) as the data presented in Section 1.1.[1] We list it below.

	Radius from gas adsorption (au)	Ps Lifetime (ns)
zeolite(4A)	10.1	3.8(1)
zeolite(4A)	10.1	5.0
zeolite(13X)	11.3	4.4(3)
zeolite(13X)	11.3	8.0

Here the same material with the same size pores obtains different lifetimes because the data comes from two different experimental groups (refer to (2) for this information). It's unlikely that different pore geometries could explain this discrepancy, as two different samples of the same material should have basically

[1]While it would be more satisfying to compare to that data exactly, unfortunately those pores were larger than those on which we collected data. Those data in Section 1.1 were chosen because they had such dramatically different lifetimes, to illustrate the point being made there.

the same porous structure. However, it is possible that the two samples of each zeolite had a different amount of total void space within them, which would change their dielectric constants, and might account for the substantial differences in Ps lifetime.

The zeolite(4A) data compares very favorably to our $R_c = 10au$ results: the first one matches exactly with our $k_o = 1$ result, and it looks like putting $k_o = 2.5$ into our code would yield the second data point's lifetime of $3.8ns$. Extrapolating to $R_c = 11au$, it appears that the two zeolite(13X) data points could be explained by those same two dielectric constants. Certainly $k_o = 2.5$ is a reasonable dielectric constant, typical of these materials. As for the points that match our $k_o = 1$ data, it is possible that the material's dielectric constant was indeed close to unity (our code's results for, say, $k_o = 1.5$ would probably be equivalent, within error, to $k_o = 1.0$, and 1.5 is a possible dielectric constant for a material).

Thus our results have precisely the right order of magnitude to explain such experimental data ($k_o = 2.5$ is a very reasonable dielectric constant, typical of such materials). Given that pore geometry seems an insufficient explanation in this case, we conclude that the dielectric constant of the material is an excellent candidate for explaining two data points for which the pore radii are equal but Ps has different lifetimes. If experimentalists were to use only Ps to measure the size of the pores in a material, they would have to include this dielectric effect in order to obtain the correct answer.

4.4 Conclusion

We successfully carried out a Path Integral Monte Carlo simulation of positronium within the spherical voids of dielectric solids. The resulting pickoff lifetimes have a complicated relationship with the Tao-Eldrup model's predictions, in that no one particular dielectric constant causes our two-particle model of Ps to agree with Tao-Eldrup. The main hallmark of a dielectric constant greater than one is that the lifetime approaches a limit as $R_c \to \infty$ instead of increasing without bound as it would were $k_o = 1$. Letting $k_o \to \infty$ also causes the lifetime to approach a limit, owing to the approximate $(1 - k_o)/(1 + k_o)$ dependence of the dielectric potential. All effects on the lifetime were more dramatic for a bare

positron than for Ps, as well as being more dramatic in larger cavities than in smaller ones.

Much further work could be done with this project. A small modification to the code, but one that could dramatically reduce runtime, would be to apply a correction to the cross-energy terms using the generating function for the Legendre polynomials, as we very successfully applied the geometric series correction to the self-energy terms (see the footnote on p. 20). In general, much data remains to be taken, particularly for cavities with radii of more than a nanometer. We purposely neglected to vary the temperature as we collected data in order to make only ground state comparisons, but this would almost certainly make an interesting study. Also, our inclusion of the dielectric response is based on a bulk electrostatics calculation, a macroscopic property of materials. In reality, when near the cavity wall, the positronium can "distinguish" individual electrons, and their effects on each other should not be construed as polarization, but more in terms of detailed (quantum) electrodynamics. Modeling this complex interaction would also do away with the empirical parameter Δ, but unfortunately this sort of computation is probably unfeasible and would certainly require major modifications of the existing code.

References

[1] L. Larrimore, R.N. McFarland, P.A. Sterne, and A.L.R. Bug. "A Two- Chain Path Integral Model of Positronium." *J. Chem. Phys.* **113**, 10642 (2000). 2

[2] K. Ito, H. Nakanishi, and Y. Ujihira. "Extension of the Equation for the Annihilation Lifetime of *ortho*-Positronium at a Cavity Larger than 1 nm in Radius". *J. Phys. Chem. B* **103**, 4555 (1999). 2, 50

[3] S.J. Tao *J. Chem. Phys.* **56**, 5499 (1972). 2

[4] M. Eldrup, D. Lightbody, and J.N. Sherwood. *Chem. Phys.* **63**, 51 (1981). 2

[5] D.J. Griffiths. *Introduction to Elementary Particles.* New York: John Wiley & Sons, 1987. 3, 4, 5, 6

[6] G. Consolati. "Positronium trapping in small voids: Influence of their shape on positron annihilation results." *J. Chem. Phys.* **117**, 7279 (2002). 3, 49

[7] F. Halzen and A.D. Martin. *Quarks and Leptons: An Introductory Course in Modern Particle Physics.* New York: John Wiley & Sons, 1984. 3

[8] A. Dupasquier. "Quasipositronium in Liquids and Solids." In *Positron Solid State Physics*, edited by W. Brandt and A. Dupasquier. New York: North Holland, 1983. 7

[9] B. Barbiellini, M.J. Puska, et al. "Correlation effects for positron annihilation with core and semicore electrons." *Appl. Surface Science* **116**, 283 (1997). 8

[10] R.P. Feynman. *Statistical Mechanics*. Benjamin Cummings, 1972. 9

[11] D.P. Landau and K. Binder. *A Guide to Monte Carlo Simulations in Statistical Physics*. Cambridge: Cambridge Univ. Press, 2000. 9, 12, 21

[12] P.K. MacKeown. *Stochastic Simulation in Physics*. Springer, 1997. 9, 21, 25

[13] M.P. Allen and D.J. Tildesley. *Computer Simulation of Liquids.* New York: Oxford University Press, 1987. 13, 33

[14] D.J. Griffiths. *Introduction to Electrodynamics*, 3rd ed. Upper Saddle River, NJ: Prentice-Hall, 1999. 15, 57

[15] J.D. Jackson. *Classical Electrodynamics.*, 3rd ed. New York: John Wiley & Sons, 1999. 15

[16] M.H. Muser and B.J. Berne. "Circumventing the pathological behavior of path-integral Monte Carlo for systems with Coulomb potentials." *J. Chem. Phys.* **107**, 571 (1997). 18, 35

[17] K.F. Riley, M.P. Hobson, and S.J. Bence. *Mathematical Methods for Physics and Engineering*. New York: Cambridge University Press, 1997. p. 458 20

[18] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller and E. Teller. "Equation of State Calculations by Fast Computing Machines." *J. Chem. Phys.* **21**, 1087 (1953). 21

[19] D.M. Ceperley. "Path integrals in the theory of condensed helium." *Rev. Mod. Phys.* **67**, 279 (1995). 33

[20] E.L. Pollock. "Properties and Computation of the Coulomb Pair Density Matrix." *Comput. Phys. Commun.* **52**, 49 (1988) 36

[21] E.L. Pollock and D.M. Ceperley. "Simulation of quantum many- body systems by path-integral methods." *Phys. Rev. B* **30**, 2555 (1984). 37

[22] E. Ley-Koo and S. Rubinstein. "The hydrogen atom within spherical boxes with penetrable walls." *J. Chem. Phys.* **71**, 351 (1979). 37

[23] J.A. Barker. "A quantum-statistical Monte Carlo method; path integrals with boundary conditions." *J. Chem. Phys.* **70**, 2914 (1979). 38

[24] P.A. Whitlock and M.H. Kalos. *J. Comp. Phys.* **30**, 361 (1979). 38

Appendix A

Solution to Poisson's Equation

We present here a derivation[1] of Eq. 2.12: the electrostatic potential within a radius R_c sphere of dielectric constant k_i, surrounded by a material of dielectric constant k_o, with a point charge q at the position $\mathbf{r}_s$ within the sphere. We choose the z-axis to lie through $\mathbf{r}_s$ so that the polar angle θ is the angle between $\mathbf{r}_s$ and the variable position $\mathbf{r}$ at which the potential is being taken.

Though for our purposes the potential outside the sphere is unimportant, we must solve for the potential everywhere in space in order to match boundary conditions. It is well known that the solution to Laplace's equation, ie the homogeneous solution to Poisson's equation, is

$$\phi_h(\mathbf{r}) = \begin{cases} \sum_{n=0}^{\infty} A_n r^n P_n(\cos\theta) & r < R_c \\ \sum_{n=0}^{\infty} B_n r^{-(n+1)} P_n(\cos\theta) & r > R_c \end{cases} \tag{A.1}$$

where the coefficients A_n and B_n are to be determined by the boundary conditions.

We must add on the particular solution due to the charge within the sphere. This is simply the usual Coulomb term $q/|\mathbf{r} - \mathbf{r}_s|$, but we expand this in terms of Legendre polynomials in order to be able match coefficients with the homogeneous

[1] We use cgs units, the proper units of electromagnetism. These also translate properly into the atomic units we employed in the code.

solution.

$$\phi_p(\mathbf{r}) = \sum_{n=0}^{\infty} \frac{q}{k_i r_s} \left(\frac{r}{r_s}\right)^n P_n(\cos\theta) \quad r < r_s$$
$$\sum_{n=0}^{\infty} \frac{q}{k_i r} \left(\frac{r_s}{r}\right)^n P_n(\cos\theta) \quad r > r_s \tag{A.2}$$

The full potential is then the sum of ϕ_h and ϕ_p, where ϕ_p is zero outside the sphere.

The boundary conditions we impose are that both the potential and the perpendicular component (in this case, this is the radial component) of the electric displacement $\mathbf{D} = k\mathbf{E} = -k\nabla\phi$ must be continuous[1] at $r = R_c$.(14) The fact that this must hold for all θ means that we must match the coefficients in the infinite sums. Requiring the potential to be continuous yields

$$A_n R_c^n + \frac{q}{k_i} \frac{r_s^n}{R_c^{n+1}} = \frac{B_n}{R_c^{n+1}}. \tag{A.3}$$

The second condition amounts to requiring

$$\hat{r} \cdot (k_o \nabla\phi_{out} - k_i \nabla\phi_{in}) = 0 \tag{A.4}$$

where ϕ_{out} is the potential as $r \to R_c$ from outside, and ϕ_{in} is the potential as $r \to R_c$ from inside. This in turn becomes

$$k_i \left(n A_n R_c^{n-1} - \frac{q}{k_i} \frac{(n+1) r_s^n}{R_c^{n+2}}\right) = k_o \left(-(n+1) \frac{B_n}{R_c^{n+2}}\right) \tag{A.5}$$

Putting the expression for B_n given by Eq. A.3 into Eq. A.5 allows us to solve for A_n without much trouble. The result is

$$A_n = \frac{q}{k_i} \frac{r_s^n}{R_c^{2n+1}} \frac{(n+1)(k_i - k_o)}{n k_i + (n+1) k_o}. \tag{A.6}$$

For completeness, we state the (unneeded) result for the coefficients B_n here as well:

$$B_n = \frac{q r_s^n (2n+1)}{n k_i + (n+1) k_o}. \tag{A.7}$$

[1] The difference in the perpendicular component of the electric displacement across a boundary is proportional to the free surface charge density at the boundary; there is no such surface charge in our problem.

Note that although we needed to include the term ϕ_p in order to properly solve for the potential, this term represents the direct Coulombic interaction between the charge producing this potential and whatever charge experiences it. Since we handle that interaction energy a different way in our code, we do not include this term in the dielectric energy calculation. Thus the potential inside the sphere given by Eq. A.1, when the expression for A_n of Eq. A.6 is put into it, is our desired result; indeed, this agrees with the stated Eq. 2.12.

Appendix B

Main Positronium Code, cavity_diel_Ps.f90

Appendix B - cavity_diel_Ps.f90

```
PROGRAM PIMC_ES  ! Path Integral Monte Carlo
!!~~~~~~~~~~~~~~~~~~~~!!
!! TABLE OF CONTENTS !!
!!~~~~~~~~~~~~~~~~~~~~!!
!! Written 6.30.2005

!! 1) Program History           (~ 120 lines)
!! 2) Variable Declaration      (~  80 lines)
!! 3) Main Program Body         (~ 160 lines)
!! 4) Subroutines               (~ 680 lines)
!!      binit           (~  10 lines)
!!      cartbin         (~  10 lines)
!!      init_beads      (~  70 lines)
!!      Initialize      (~ 100 lines)
!!      move            (~ 140 lines)
!!      move_cm         (~  20 lines)
!!      plot_beads      (~  70 lines)
!!      ps_orientation  (~  20 lines)
!!      RadialBin       (~  50 lines)
!!      ReadInput       (~  50 lines)
!!      tryboth         (~  40 lines)
!! 5) Functions                 (~ 170 lines)
!!      gauss           (~ 10 lines)
!!      gcav            (~ 20 lines)
!!      legp            (~ 10 lines)
!!      ran1            (~ 30 lines)
!!      vcoulomb        (~ 10 lines)
!!      vdielec         (~ 60 lines)
!!      vfun1           (~ 10 lines)
!!      vquant          (~ 20 lines)



! The HISTORY of this program
! This code originated as sphere4.2.f90 in Summer, 1999
! via the routine sphere_esV5.f90 in March, 2000

! FORMER VERSION (sphere4.2.f90) WAS ...
! a staging program of a single particle
! The potential was the image potential due to a hard sphere.
! 7-27: created from sphere3.1.f90 to print out virial energy
! separately for each bead

! energyCAV not included in energy averages
! 7-28: virial function bug fix (r2 -> rplus2)
! 7-29: 4's changed back to 2's in virial and gcav functions
! 3-3-00: virial energies disabled

! NEWER VERSION (sphere_esV5.f90) WAS ...
! 3-1-00:  MODIFIED to use pseudopotential from electronic structure calculat
! 3-3-00:  to read discrete potentials (from file v.dat)
! 3-3-00: reads  unformatted (from file v_unform.dat)
! 3-6-00: and we have periodic boundary conditions; are testing a sinusoidal
! potential
! 3-7-00:  We use a cubic spline interpolation routine due to J.E. Pask
!  to interpolate between values in file
! 3-8-00: We integrate the bead density, weighted by an electronic charge
! density which is read as an input file, and also interpolated

! NEWER VERSION (periodic1.f90) WAS ...
! From periodic.f90 which was a more f9x compliant version of sphere_esV5.f90
! and tailored to do periodic potentials
! Intermediate version a code called kronig.f90 which removed the hard cavity
! The potential and charge density no longer come from direct reads to files
! but rather, via the module vcg_setup
! This makes module called interpolation , present in
! periodic.f90 and kronig.f90, obsolete and it has been removed
!
! Uses routines collected by Phil Sterne into a wrapper vcg.f90 which
! we have adapted into this program
! Below are comments from Sternes vcg.f90 program:
!  purpose:
!    program to calculate positron potentials by interpolation
!    from the fepot output vcg file
!   calls:
!         subs: readin potset recip clust gradpt
!          fns: vatpt, fnatpt, gammap
!
!  written:  philip sterne, sterne1@llnl.gov
!     date:  March 10, 2000

! NEWER VERSION (periodic2.f90) WAS ...
! a routine with TWO chains, electron and positron, interacting via
! a Yukawa potential

! NEWER VERSION (Mac_PEZ) WAS ...
! A two chain model interacting through the coulombic thermal density propaga
! a routine that employs an atom-electron pseudopotential. It
!  also has improvements to efficiency as well as a new observable: R(s),
```

```
!  the bead correlation function...
!  Runs on the Mac as well as Oxford, thanks to formatted input since data fi
! were  created (for no good reason) only
!  on Oxford

! NEWER VERSION (Mac_PEZnew) WAS ...

! 09-jul-01: calculates pair correlation function g(r), writes to output 23
! 10-jul-01: updates centers of mass (xc, xc1, xc2), bins xc2(1) in output 28
!            energy variables declared & initialized with others (instead of
! 11-jul-01: cavbin initalized (error in previous code)

! NEWER VERSION (Mac_PPEZnew) IS ....
! A parallel version of Mac_PEZ designed to run on the Mac cluster
! 20-jul-01: cavbin no longer calculated (dist from e+ to origin, irrelevant)
! 23-jul-01: -g(r) calculated from cm (binned in gr_cm_bin, output 23) and
!                         from every bead (binned in gr_bead_bin, output 24)
!                        -new subroutine move_cm: called every nevalu passes
!                         cm moved in random direction, ac is now acsum/(neva
! 24-jul-01: -binning done every jump steps
!                        -corrections from comparing to PPEZ:
!                                -setting UseExternal=.false. now has effect
!                                -R_Correlation and x_changed arrays made all
!                                -files given names (instead of FOR0??.DAT)
!                        -c_overlap written every jump to output 30, final ou
! 08-aug-01: c_overlap_count bug fixed (no longer writes NaN to first line)
! 10-aug-01: ibin replaced by ibin+1 in RadialBin subroutine (prevents error
! 13-aug-01: no longer calculates total energy before nequil
! NEWER VERSION
! Commenting has taken a back seat.  But this code became CartbinZ_E.f90,
! which may have run in parallel on Macs, but was made to work on gps
! in late July, 2002.  Debugged so would not overwrite arrays
! and so hits in bins would be integers
! NEWEST VERSION
! cavity.f90   will have particle-in-cavity potential re-enabled.  Can
! either one or two interacting chains in a cavity


!! Program adopted by Tim Cronin 6/20/2005 !!
!! Initial changes involve switching to a cylindrical geometry
!! 6/21/2005 -- Added in adjustable Monte Carlo stuff for center of mass move
       !! Also a diagnostic for averaging the number of beads moved per stag
       !! made number of bins sensitive to the cavity radius
!! 6/22/2005 -- Added in restriction on g_cav that sets propagator to zero if
       !! tries to move outside the wall
!! 6/23/22005 -- Now also write decomposition of relative e+e- radius
       !! into component in r-theta plane and z-direction
!! 6/27/2005 -- Put in a subroutine that analyzes the orientation of the posi
       !! as a function of CM coordinate
       !! writes to file 51, called _orientation
!! 6/28/2005 -- Put in an energy estimator that outputs to the terminal windo
       !! also incorporated a short routine to "burn" a few random numbers a
       !! much as the dealer "burns" cards in texas hold 'em to ensure rando
       !! specifying a different number of initial burns allows for totally
       !! using the same random numbers
!! 6/29/2005 -- Began some major organizational changes to the program, inclu
       !! deleted many superfluous subroutines and modules
       !! alphabetized and separated subroutines and functions
       !! created a table of contents for each category
!! 6/30/2005 -- and for the whole program, as well


!! Program inherited by Zach Wolfson from Tim 6/30/2005 !!
!! 6/30/2005 -- Switched back to spherical geometry
       !! Added working vdielec subroutine (worked in pre-Tim cavity_nosolid
       !! Added error checker in main...stops if a bead is outside the cavit
       !! ko and ki are now parameters...maybe put them in cavity.in later?
!! 7/6/2005 -- Added cmbin subroutine to make sure we're exploring the whole
!! 7/7/2005 -- Amy and I modified the cartbin() subroutine to also output abs
       !! ...the data is put in the for49 file
       !! also, needed to place some strategic 4's in dealing with the for50
       !! the separation btw e+ and e- beads...one 2 for Cartesian coords, a
       !! it's the difference rather than absolute position
!! 7/11/2005 -- Added legp function to calculate the first nleg Legendre poly
       !! put the appropriate energy cross terms into vdielec...exciting, no
!! 7/13/2005 -- Put ko and iburnrand in the cavity_diel.in file, where they r
!! 7/15/2005 -- Allow nleglim < = nleg ... ie fewer legendre polynomials outs
       !! fewer means 25 and zone means 1 au
!! 7/15/2005 --  Orientation is put back to be appropriate for sphere, not fo
!! 7/19/2005 -- It's double precision day
!! 8/3/2005 -- Realized that the two cross terms are actually *identical*...b
       !! we can't use this to our advantage to lower the runtime (it only t


use types
use Table

implicit none

!! variables pertaining to random number generation or frame-shifting to obta
integer, parameter :: initrand = 0
integer :: iburnrand                    !! number of thousands of random numb
```

```
real :: burnrand

!! variables for binning position data
integer, parameter :: nbinmax = 10000                          ! limiting nu
real, parameter :: pi = 3.14159
integer :: nbins, norientbins, ncartbins, ncmbins              ! number of b
real :: rbinmax                                                ! limiting ra
integer :: n_total_binned=0, n_in_annihilation_zone=0
real, parameter :: annihilation_zone_size=1.656                ! size in Ang
integer :: radbin(nbinmax)                                     ! radial bins
integer :: radrhobin(nbinmax)                                  ! projections
integer :: radzbin(nbinmax)
integer :: cavbin(nbinmax)                                     ! radial bins
integer :: psrcount(nbinmax)                                   ! bins for th
integer :: psdenom                                             ! bin count,
integer :: cartesianbin(nbinmax,3)                             ! difference
integer :: cartesianbin1(nbinmax,3)                            ! absolute po
integer :: cmassbin(nbinmax,2,3)                               ! CM position


!! most of the important integers
integer :: i,j,ib,id,ic,irun,ibin              ! counters
integer :: nequil                              ! after nequil MC passes, beg
integer :: nevalu = 10                         ! after each nevalu MC passes
integer :: ninit                               ! type of initialization of b
integer :: nb, mb, npass, jump                 ! see read(11,*)'s below for
real :: mbavg                                  !! average number of beads mo
integer :: CorrelationCounts                   ! Number of Correlation calcs
integer :: deg                                 ! degree of spline
integer :: nleg                                ! number of Legendre polynomi

! Parameter Variables
LOGICAL :: UsePolluck                          ! Use Polluck or Yukawa poten
LOGICAL :: UseExternal                         ! Use external potentials
LOGICAL, ALLOCATABLE :: x_changed(:,:)         ! Flag for beads that are mov

! position and some energy variables !
real (dp), DIMENSION(:,:,:), ALLOCATABLE :: x,xn
real (dp), ALLOCATABLE :: xEnergyOld(:,:)            ! potential energy of ea
real (dp), ALLOCATABLE :: xEnergyNew(:,:)            ! potential energy of ea

real (dp) :: DP_New, DP_Old = 0.0                    ! Density Potentials
real (dp) :: xc1(3), xc2(3), xc(3)                   ! centroid of each path,
double precision :: xper, yper, zper

!! orientation variables
```

```
real :: psrdot(nbinmax)

! electric field
real (dp) :: Efield                            ! Magnitude of Electric field
real (dp) :: AddToE

! energies
integer :: energy_count=0                                  ! counter for averagi
real :: energy_sum=0.0

!! input variables, adaptive monte carlo stuff
real :: ko, ki = 1.0                                       ! outside and inside
real :: amass, beta, hbar, rcav, aep, zdelta               ! see read(11,*)'s be
real :: rcav2                                              ! rcav squared
real :: wave                                               ! deB wavelength of f
real (dp) :: ac, acsum                                         ! acceptance rat
real (dp) :: accm, accmsum, cmmove=0.2                         ! acceptance rat
real :: sigelec, sigposi ,sigelec_self, sigposi_self       ! width of path
real (dp) :: rrel, rdot,  r_cm                                                 ! r
real :: rforbin(nbinmax), rorientforbin(nbinmax), rcartforbin(nbinmax)     ! r
real :: rcmforbin(nbinmax), rcartforbin1(nbinmax)

!! miscellany/residue
integer, parameter :: fileNameLength=70 ! length of character string for file
integer   :: iui                        ! logical unit number for input
integer   :: iuo                        ! logical unit number for output
integer   :: error                                         ! error flag
integer   :: ierror                     !ditto



!!mmmmmmmmmmmm!!
!!~~~~~~~~~~~~!!
!! BEGIN MAIN !!
!!~~~~~~~~~~~~!!
!!mmmmmmmmmmmm!!



 call ReadInput()
 call Initialize()

print *, "we will burn",iburnrand*1000, "random numbers"
do i=1,iburnrand*1000
        burnrand=ran1(initrand)
        burnrand=0.0
```

```
end do

! Loop through the PIMC routine for a total of 'npass' times
MC_passes:  do irun = 0, npass-1

  ! Do a Monte Carlo move
  call move(ac)
! diagnostic
!write(6,*) irun, 'moving'

  ! The beads are *not* allowed outside...this is an especially important dia
  ! the dielectric potential can be very attractive (doing Yukawa saves it, t
  do ic = 1, 2
     do ib = 1, nb
        if((x(ib,ic,1)**2 + x(ib,ic,2)**2 + x(ib,ic,3)**2) .gt. rcav*rcav) th
               write(6,*) ' error, bead ', ib, ic, 'is outside cavity'
               stop
        endif
     end do
  end do

  ! Update the centers of mass:
  do i=1,3
    xc1(i) = sum(x(:,1,i))/float(nb)
    xc2(i) = sum(x(:,2,i))/float(nb)
  end do
  xc = (xc1+xc2)/2.0

  ! The magic of integer arithmetic allows us to write code which
  ! prints out what percent of the calculation has been completed in a single
  ! if( irun*100/npass > (irun-1)*100/npass ) write(*,*) (irun*100/npass), "p

  ! Now we update the acceptance rate by adding 'ac' to 'acsum'.
  ! ac was set to 1 by move(ac) is a successful move was made.
  acsum = acsum + ac

  ! Every nevalu-th move, we look at acsum, and see how many moves are being
  ! If the rate is too low or two high, we adjust the number of beads moved i
  if (mod(irun,nevalu) == nevalu-1) then
    ac = acsum/(nevalu-1)               ! Determine fraction of accepted move
                                                                  ! sin
  if (mod(irun,jump)==0) then    ! When irun is a multiple of jump...
        !write(6,100) irun,ac,mb        ! A simple diagnostic
        endif
    100 format ("",I8,"       ",F6.2,"          ",I4)
    ! adjust mb so the acceptance rate is roughly 50%
    if(ac >  0.5) mb=mb+1
    if(ac < 0.5) mb=mb-1
    if(mb < 1) mb=1
    if(mb > nb) mb=nb
    acsum = 0
  end if

  !! compute the average number of beads moved in a staging pass
  !! average would be total # beads moved divided by total number of staging

  mbavg=mbavg+real(mb)/real(npass)

  !! Now we update the acceptance rate by adding 'accm' to 'accmsum'.
  !! accm was set to 1 by move(ac) if a successful move was made.
  accmsum = accmsum + accm

  !! Every (nevalu^2)-th move, we look at accmsum, and see how many moves are
  !! If the rate is too low or two high, we adjust the size of the cm step.
  if (mod(irun,nevalu**2) == nevalu**2-1) then
    accm = accmsum/nevalu               !! Determine fraction of accepted mov
    !! adjust cmmove so the acceptance rate is roughly 50%
    if(accm.GT.0.6.AND.cmmove.LT.rcav/4.0) cmmove=cmmove*1.2
    if(accm.GT.0.6.AND.cmmove.GE.rcav/4.0) cmmove=rcav/4.0
    if(accm.LT.0.4) cmmove=cmmove*0.8
    accmsum = 0
  end if

  ! Every jump-th move, we make calculations regarding the centroids of the b
  ! their dispersion, and their correlation function.

  if (mod(irun,jump)==0) then    ! When irun is a multiple of jump...

    ! if we have completed the specified number of equilibration steps, then.
    if (irun.gt.nequil) then

      ! ...we bin the cartesian coordinates of the CM of each particle
      call cmbin()

      ! ... and the relative, radial, seperation of the electron and positron
      call RadialBin()

      !! also determine the orientation of the Ps atom
      call ps_orientation()

      !! bin the data according to cartesian separation
      call cartbin()
```

```
      !! and determine the average energy
      energy_count=energy_count+1
      energy_sum=energy_sum+3.0*nb/beta-vquant()*nb*amass/(2*beta**2)+vdielec

    end if

    ! Calculate electron and positron bead dispersions
    sigelec = 0.0
    sigposi = 0.0
    sigelec_self=0.0
    sigposi_self=0.0

    do ib = 1,nb
      sigelec = sigelec + (x(ib,1,1)-xc(1))**2 + (x(ib,1,2)-xc(2))**2 &
      + (x(ib,1,3) - xc(3))**2
      sigposi = sigposi + (x(ib,2,1)-xc(1))**2 + (x(ib,2,2)-xc(2))**2 &
      + (x(ib,2,3) - xc(3))**2
      sigelec_self = sigelec_self + (x(ib,1,1)-xc1(1))**2 + (x(ib,1,2)-xc1(2)
      + (x(ib,1,3) - xc1(3))**2
      sigposi_self = sigposi_self + (x(ib,2,1)-xc2(1))**2 + (x(ib,2,2)-xc2(2)
      + (x(ib,2,3) - xc2(3))**2
    end do
    sigelec = (sigelec/float(nb)) ** .5
    sigposi = (sigposi/float(nb)) ** .5
    sigelec_self = (sigelec_self/float(nb)) ** .5
    sigposi_self = (sigposi_self/float(nb)) ** .5

    ! Write these values out to the 'fort.12' file.
    write(12,"(i6, 1x, 4f12.5)") irun, sigposi,sigelec, sigelec_self, sigposi

 end if

end do MC_passes


! call plot_beads()             ! Zach: I don't want to plot right now

! write final bead positions to file
  open(unit = 17, file = 'for17_final_bead_new',  action = 'write', iostat =
  charge4: do ic = 1, 2
    bead4: do ib = 1, nb
      write(17,*,iostat = ierror) x(ib,ic,1), x(ib,ic,2), x(ib,ic,3)
    end do bead4
  end do charge4
```

```
!! Write the radial distribution data

write(20,*) "Dist'n fcn of rel coord, e+ dist from (0,0,0)"
write(20,*) "Beta = ", beta
write(20,*) "# of beads =", nb
write(20,*) "# of runs  =", npass
write(20,*) "ko =", ko, "ki =", ki
do ibin = 1, nbins
 write(20,*) rforbin(ibin), radbin(ibin), cavbin(ibin)
end do


!! write important statistics about the run
!! The answer to the question of Life, the Universe, and Everything

write(42,*) "Important information about the latest run ::"
write(42,*) "Beta = ", beta
write(42,*) "Temperature in K = ", 315795.9/beta
write(42,*) "Number of beads = ", nb
write(42,*) "Number of staging passes = ", npass
write(42,*) "nequil = ", nequil
write(42,*) "Cavity Radius = ", rcav, "au"
write(42,*) "ko =", ko, "ki =", ki
write(42,*) "Number of random numbers discarded = ", 1000*iburnrand
write(42,*) "Seed = ", initrand
write(42,*) "Width of pickoff zone = ", 1.8893*annihilation_zone_size, "au"
write(42,*) "P(e+ in pickoff zone) = ", real(n_in_annihilation_zone)/real(n_t
write(42,*) "Pickoff lifetime = ", 1.0/(2.0*real(n_in_annihilation_zone)/real
write(42,*) "average number of beads moved in a staging pass = ", mbavg
write(42,*) "average energy of the Ps = ", energy_sum/real(energy_count)


!! write cartesian binned data

write(49,*) "Cartesian absolute Distribution"
write(49,*) "Beta = ", beta
write(49,*) "# of beads =", nb
write(49,*) "# of runs  =", npass
do ibin = 1, ncartbins
        write(49,*) rcartforbin1(ibin), cartesianbin1(ibin,1), cartesianbin1(
end do


!! write cartesian binned data

write(50,*) "Cartesian separation Distribution"
```

```
write(50,*) "Beta = ", beta
write(50,*) "# of beads =", nb
write(50,*) "# of runs  =", npass
do ibin = 1, ncartbins
        write(50,*) rcartforbin(ibin), cartesianbin(ibin,1), cartesianbin(ibi
end do


!! write the orientation data
write(51,*) "Positronium orientation"
write(51,*) "Beta = ", beta
write(51,*) "# of beads =", nb
write(51,*) "# of runs  =", npass
do ibin = 1, norientbins
 psdenom  = max(psrcount(ibin),1)
 write(51,*) rorientforbin(ibin), psrdot(ibin)/psdenom,&
  1.0-psrdot(ibin)/psdenom,&
 psrcount(ibin)  !! radius, <rdot^2>, <thetadot^2>, # of counts
end do


!! write CM data
write(60,*) "CM position of positron (cartesian coordinates x,y,z)"
write(60,*) "Beta = ", beta
write(60,*) "# of beads =", nb
write(60,*) "# of runs  =", npass
write(60,*) "ko =", ko, "ki =", ki
do ibin = 1, ncmbins
     write(60,*) rcmforbin(ibin), cmassbin(ibin,2,1), cmassbin(ibin,2,2), cma
end do


print *, "program finished, see data files for results"



!!mmmmmmmmmm!!
!!~~~~~~~~~~!!
!! END MAIN !!
!!~~~~~~~~~~!!
!!mmmmmmmmmm!!


 contains
```

```
!!sssssssssssssssssss!!
!!~~~~~~~~~~~~~~~~~~~~!!
!!~~~~~SUBROUTINES~~~~~!!
!!~~~~~~~~~~~~~~~~~~~~!!
!!sssssssssssssssssss!!



!! SUBROUTINE LIST (alphabetized, and ordered in the program accordingly) ::
!! list created 6.29.2005, updated 7.6.2005
!!      binit           (~  10 lines)
!!      cartbin         (~  10 lines)
!!      cmbin           (~  20 lines)
!!      init_beads      (~  70 lines)
!!      Initialize      (~ 100 lines)
!!      move            (~ 140 lines)
!!      move_cm         (~  20 lines)
!!      plot_beads      (~  70 lines)
!!      ps_orientation  (~  20 lines)
!!      RadialBin       (~  50 lines)
!!      ReadInput       (~  50 lines)
!!      tryboth         (~  40 lines)



!!~~~~~~~~~~~~~~~~~~~!!
!! BINIT SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~!!

!! initializes a set of bins for storing data

subroutine binit(nbns,rbmax, rforbin)
implicit none
integer :: nbns
real :: rbmax
real, intent(out) :: rforbin(nbns) ! radius corresponding to bin
do ibin = 1, nbns
 rforbin(ibin) = (ibin - .5)*rbmax / float(nbns)
 end do
end subroutine binit



!!~~~~~~~~~~~~~~~~~~~~!!
```

```
!! CARTBIN SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~!!

!! Cartesian-coordinate binning for the separation between the e+ and e-

subroutine cartbin()
implicit none
real (dp) :: separation

do ib=1,nb
        do id=1,3
                separation=x(ib,1,id)-x(ib,2,id)
                ibin = int(separation/(4.0*rbinmax)*float(ncartbins)+float(nc
                ibin = min(ibin, ncartbins-1)  !the outer bin contains all ou
                cartesianbin(ibin+1,id)=cartesianbin(ibin+1,id)+1
                                        ! now bin absolute positions
                ibin = int((x(ib,2,id)*ncartbins/2/rbinmax + float(ncartbins).
                ibin = min(ibin, ncartbins-1)  !the outer bin contains all ou
                cartesianbin1(ibin+1,id) = cartesianbin1(ibin+1,id)+1
        end do
end do

end subroutine cartbin


!!~~~~~~~~~~~~~~~~~~~!!
!! CMBIN SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~!!

! Bins the CM position of both the electron and the positron

subroutine cmbin()
implicit none
real (dp) :: cmpos

do ic=1,2
   do id=1,3
      cmpos = ( sum(x(:,ic,id)) - x(nb+1,ic,id) ) / float(nb)   ! don't count
      ibin = int(ncmbins/2*(cmpos/rbinmax + 1.0))
      ! as in cartbin above, Cartesian coordinates are a pain...turn negative
      ibin = min(ibin, ncmbins-1)
      cmassbin(ibin+1,ic,id)=cmassbin(ibin+1,ic,id) + 1
   end do
end do

end subroutine cmbin
```

```
!!~~~~~~~~~~~~~~~~~~~~~~~~!!
!! INIT_BEADS SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~~!!

!! Initializes the chains of beads

subroutine init_beads(ni)
implicit none
integer :: is = 1       ! (rightnow, a dummy) variable for gaussian RNG
integer :: id           ! counter for dimension
integer :: ierror
integer :: ibeadcount   ! number of beads in reading file
integer :: ni           ! type of initialization: 0 de novo or 1 from file
integer :: ic           ! charge (electron = 1 or positron = 2)
real (dp) :: xsum,xshift
double precision :: gg

if(ni == 0) then
gg = min(wave*wave,rcav*rcav/12.0)
! start gg smaller if you wish
!   gg = 0.1
  dim: do id = 1,3
    xsum = 0.0
    charge1: do ic = 1,2
      bead1: do ib = 1, nb
        x(ib,ic, id) = gauss(gg,is)
        xsum=xsum+x(ib,ic,id)
      end do bead1
      x(nb+1,ic,id) = x(1,ic,id)
    end do charge1
    xsum = xsum/2.0/float(nb)
    xshift = xc(id)-xsum
    charge2: do ic = 1,2
      bead2: do ib = 1,nb+1
        x(ib,ic,id)=x(ib,ic,id)+xshift
      end do bead2
    end do charge2
  end do dim
end if

if (ni == 1) then
  ibeadcount = 0
  open(unit = 16, file = 'for17_final_bead', status = 'old', action = 'read',
```

```
  if(ierror /= 0) then
    write(*,*) 'An error occured opening for17_final_bead'
    write(*,*) 'Consider changing ninit in the cavity.in file to 0.'
    STOP
  end if
  charge3: do ic = 1, 2
    bead3: do ib = 1, nb
      read(16,*,iostat = ierror) x(ib,ic,1), x(ib,ic,2), x(ib,ic,3)
      ibeadcount = ibeadcount+1
      if (ierror /= 0) STOP
    end do bead3
  end do charge3

  !Close off the chain
  x(nb+1,:,:) = x(1,:,:)

  if(ibeadcount /= 2*nb) then
    write(*,*) 'too few or too many beads in for17_final_bead file'
    STOP
  end if
end if

if (ni > 1) then
  write(*,*) 'erroneous flag for reading beads'
  STOP
end if

xn = x

end subroutine init_beads



!!~~~~~~~~~~~~~~~~~~~~~~~~!!
!! INITIALIZE SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~~!!

!! sets up data files to write to, and allocates memory for many arrays

subroutine Initialize()
  implicit none

  ! We need to initialize everything:
  ! Starting with allocating memory for beads arrays and energies

  allocate(x(nb+1,2,3),  stat = error)
if (error .ne. 0) then
  write(*,*) "Unable to allocate memory for the array: x(:,:,:)"
  stop
endif
allocate(xn(nb+1,2,3), stat = error)
if (error .ne. 0) then
  write(*,*) "Unable to allocate memory for the array: xn(:,:,:)"
  stop
endif
allocate(xEnergyOld(nb+1,2), stat = error)
if (error .ne. 0) then
  write(*,*) "Unable to allocate memory for the array: xEnergyOld(:,:)"
  stop
endif
allocate(xEnergyNew(nb+1,2), stat = error)
if (error .ne. 0) then
  write(*,*) "Unable to allocate memory for the array: xEnergyNew(:,:)"
  stop
endif
allocate(x_changed(nb,2), stat = error)
if (error .ne. 0) then
  write(*,*) "Unable to allocate memory for the array: x_changed(:)"
  stop
endif

!! Set up the files we'll be writing our data to

open(unit=12,file='for12_bead_disp',status='replace',action='write')

open(unit=20,file='for20_rad_dist',status='replace',action='write')

open(unit=42,file='for42_stats',status='replace',action='write')

open(unit=49,file='for49_bin_abs_pos',status='replace',action='write')

open(unit=50,file='for50_bin_sep',status='replace',action='write')

open(unit=51,file='for51_orientation',status='replace',action='write')

open(unit=60,file='for60_CM_pos',status='replace',action='write')


! And initializing the thermal density propagator table:
call CreateTable(beta/float(nb))
```

```
! Setting the dimensions and zeroing the radial bin for positron and electr
nbins = npass/150 !! use more bins if we've got more staging passes
ncartbins = npass/150
ncmbins = npass/150
norientbins = 100 !! we use fewer bins for the orientation data, as it's no
rbinmax = 10.0 !! set the maximum bin to 10
if (rcav.GT.10.0) rbinmax=rcav !! or to the cavity radius, whichever is lar
radbin(:) = 0
radrhobin(:) = 0
radzbin(:) = 0
cavbin(:) = 0
psrcount(:) = 0
psrdot(:) = 0.0
cartesianbin(:,:) = 0.0
cmassbin(:,:,:) = 0.0
call binit(nbins, rbinmax, rforbin)
call binit(norientbins, rbinmax, rorientforbin)
call binit(ncartbins, 4*rbinmax, rcartforbin)  ! make them twice as big to
call binit(ncartbins, 2*rbinmax, rcartforbin1) ! coordinate bin twice *that
call binit(ncmbins, 2*rbinmax, rcmforbin)
rcartforbin=rcartforbin-2*rbinmax     ! center them at 0.0
rcartforbin1=rcartforbin1-rbinmax
rcmforbin = rcmforbin - rbinmax


! Zeroing the centroid positions
xc1(:) = 0.0
xc2(:) = 0.0
xc(:) = 0.0

x_changed(:,:) = .TRUE.

! Initializing Energy Arrays:
xEnergyOld(:,:) = 0.0
xEnergyNew(:,:) = 0.0

! Create an initial bead configuration: A gaussian distribution (i.e.: free
call init_beads(ninit)

! ready to initialize information for potential, charge density and gamma
!  set the input and output unit numbers and read the main f file
iui = 5
iuo = 6

write(*,*) "Program cavity: PIMC with spherical cavity and dielectric mater
```

```
  ! ***The heart and soul of the PIMC routine starts here:***
  !  We set acsum, a measure of the acceptance rate, to zero:
  acsum = 0.0d0

end subroutine Initialize



!!~~~~~~~~~~~~~~~~~~!!
!! MOVE SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~!!

!! this is really the body of the code -- accepting and rejecting MC moves
!! calls upon module Mac_table to get Pollock propagator info

subroutine move(ac)
use types
implicit none
real (dp), intent(out) :: ac
real (dp) :: vsum, vsumnew, vchange, de, det, gsum, gsumnew
real (dp) :: vee, yuk_e_poo
real :: effBeta        ! beta / number of beads
real (dp) :: relec(3)  ! electron coordinates
real (dp) :: dis1(3), dis2(3), dis12(3)
real (dp) :: rdis1, rdis2, rdis12 !sep's for electron-positron interaction
real :: qep(2) ! charges on electron and positron
! temporary testing variables
integer :: bnum, eorpo

qep(1) = -1.0; qep(2) = 1.0
effBeta = beta / float(nb)
ac = 0.00 !set acceptance rate for this step equal to zero

! do a staging move on the beads (re-pick from a gaussian distribution)
! do electron and positron moves serially

if (mod(irun,nevalu).eq.0) then
  call move_cm(xn)
else
  call tryboth(xn)
end if

! DP_* are variable for the thermal density propagator 'potential'.
DP_Old = 0.0
DP_New = 0.0
```

```
if (irun == 0) then
        do bnum=1,nb
                do eorpo=1,2
                        xEnergyOld(bnum,eorpo) = ((-1)**(eorpo+1))*xn(bnum,eo
                end do
        end do
        end if

do i = 1,nb
  if( x_changed(i,1) .OR. x_changed(i,2) ) then
    ! If a bead has been moved, recalculate its energies...
    dis1 = x(i,1,:) - x(i,2,:)
    dis2 = x(i+1,1,:) - x(i+1,2,:)
    dis12 = dis1 - dis2
    rdis1  = sqrt(sum(dis1**2))
    rdis2  = sqrt(sum(dis2**2))
    rdis12 = sqrt(sum(dis12**2))

    if(UsePolluck) then
      DP_Old = DP_Old - LookUpTable(real(rdis1), real(rdis2), real(rdis12), e
    else
      DP_Old = DP_Old + vfun1(rdis1,qep,aep) * effBeta
    endif

    xper = xn(i,2,1)
    yper = xn(i,2,2)
    zper = xn(i,2,3)
    relec = (/xn(i,1,1), xn(i,1,2), xn(i,1,3)/)
    dis1 = xn(i,1,:) - xn(i,2,:)
    dis2 = xn(i+1,1,:) - xn(i+1,2,:)
    dis12 = dis1 - dis2
    rdis1  = sqrt(sum(dis1**2))
    rdis2  = sqrt(sum(dis2**2))
    rdis12 = sqrt(sum(dis12**2))

    if(UsePolluck) then
      DP_New = DP_New - LookUpTable(real(rdis1), real(rdis2), real(rdis12), e
    else
      DP_New = DP_New + vfun1(rdis1,qep,aep)  * effBeta
    endif

        do eorpo=1,2
        if (x_changed(i,eorpo)) then
         !  electric field also modifies the energy as follows
           xEnergyNew(i,eorpo) = ((-1)**(eorpo+1))*xn(i, eorpo, 3) * Efield
        endif
        end do

    endif
end do

! Now we can sum up old and new potentials
vsum = 0.0
vsumnew = 0.0

if (UseExternal) then
        write(6,*) 'We do not use external potential here'
        STOP
        endif

  do i = 1,nb
    vsum = vsum + sum(xEnergyOld(i,:))
    vsumnew = vsumnew + sum(xEnergyNew(i,:))
  enddo

gsum = gcav(x)  ! contribution to propagator from cavity
gsumnew = gcav(xn)

vchange = (vsumnew - vsum)*effBeta + DP_New - DP_Old
vchange = vchange - log(gsumnew/gsum) + (vdielec(xn, ko, ki) - vdielec(x, ko,

!vchange = -log(gsumnew/gsum) + (vdielec(xn, ko, ki) - vdielec(x, ko, ki))*ef

det = -vchange
de = dlog(ran1(initrand) + 1.0d-10)
ac = 0.0d0
accm = 0.0d0
if (det > de ) then
  if (mod(irun,nevalu).eq.0) then
    accm = 1.0d0
    ! write(*,*) "CM MOVE ACCEPTED!"
  else
    ac=1.0d0    ! Flag the fact that a move has been accepted
  end if
  vee = vsumnew
  do i = 1,nb
    do j = 1,2
      if( x_changed(i,j) ) then
        x(i,j,:) = xn(i,j,:)
        if(i==1) x(nb+1,j,:) = xn(nb+1,j,:)
        xEnergyOld(i,j) = xEnergyNew(i,j)
      endif
```

```
    enddo
  enddo
else
  vee = vsum
  do i = 1,nb
    do j = 1,2
      if( x_changed(i,j) ) then
        xn(i,j,:) = x(i,j,:)
        if(i==1) xn(nb+1,j,:) = x(nb+1,j,:)
        xEnergyNew(i,j) = xEnergyOld(i,j)
      endif
    enddo
  enddo
end if

yuk_e_poo = 0.0
! Added up the coulombic energy in the beads, using the Yukawa potential.
do i = 1,nb
  rdis1 = sqrt(sum( (xn(i,1,:)-xn(i,2,:))**2) )
  yuk_e_poo = yuk_e_poo + vfun1(rdis1,qep,aep)
enddo
vee = (vee + yuk_e_poo) / float(nb)

end subroutine move



!!~~~~~~~~~~~~~~~~~~~~~!!
!! MOVE_CM SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~!!

!! moves the center of mass of the two chains
!! is called every nevalu moves, I believe

subroutine move_cm(xnew)
implicit none
real (dp) :: d(3)
real (dp), intent(INOUT), DIMENSION(:,:,:) :: xnew

d(1) = cmmove*(ran1(initrand) - 0.5)
d(2) = cmmove*(ran1(initrand) - 0.5)
d(3) = cmmove*(ran1(initrand) - 0.5)
do ic = 1,2
  do ib = 1,nb+1
    xnew(ib,ic,:) = xnew(ib,ic,:) + d
  end do
end do
x_changed(:,:) = .true.

end subroutine move_cm



!!~~~~~~~~~~~~~~~~~~~~~~~~!!
!! PLOT_BEADS SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~~!!

!! Plots the configurations of the beads, can be called anytime
!! will prompt user for output device
!! requires that the main program be linked to the PLPLOT library

subroutine plot_beads()
implicit none
real :: zmax(2), zmin(2), zrange(2)
integer :: red_content(16),green_content(16),blue_content(16)
integer :: ib, red, green, blue

!! color list is:
!! 0=white, 1=black, 2=red, 3=blue, 4=green
!! 5=magenta, 6=mustard, 7=aquamarine
!! 8=earth, 9=forest, 10=midnight
!! 11=brass, 12=indigo, 13=brick, 14=ocean, 15=adjustable
!! you may or may not agree with these descriptions
!!                0  1    2    3    4    5    6    7    8    9   10   11   12
red_content=  (/255, 0, 255,   0,   0, 200, 200,   0, 140,  70,  70, 220,  90
green_content=(/255, 0,   0,   0, 255,   0, 200, 200,  70, 140,  70, 150,   0
blue_content= (/255, 0,   0, 255,   0, 200,   0, 200,  70,  70, 140, 100, 250
 call plscmap0_(red_content,green_content,blue_content,16)
!! Color table created by Tim Cronin 6.15.2005

 call plinit_()  !! initializes the plot
 call plenv_(-rcav,rcav,-rcav,rcav,0,1)  !! set plot bounds
 call pllab_('x-position','y-position','positions of beads in Ps')  !! axis l
 call plcol0_(2) !! set the color of one set of bead links to red
 call plline_(nb+1,x(:,1,1),x(:,1,2))
 call plcol0_(4) !! and the other to green
 call plline_(nb+1,x(:,2,1),x(:,2,2))
 zmax(1)=x(1,1,3)
 zmin(1)=x(1,1,3)
 zmax(2)=x(1,2,3)
 zmin(2)=x(1,2,3)
 do ib=1,nb !! loop to determine the max and min z values of the e+ and e- ch
```

```
!! this is done for shading purposes
       if (x(ib,1,3).GT.zmax(1)) zmax(1)=x(ib,1,3)
       if (x(ib,1,3).LT.zmin(1)) zmin(1)=x(ib,1,3)
       if (x(ib,2,3).GT.zmax(2)) zmax(2)=x(ib,2,3)
       if (x(ib,2,3).LT.zmin(2)) zmin(2)=x(ib,2,3)
end do
zrange=zmax-zmin
do ib=1,nb !! now color the beads based on how far "down" the z-axis they ar
!! white is far away, black is nearby
       red=255*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       green=255*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       blue=255*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       call plscol0_(15, red, green, blue)
       call plcol0_(15)
       call plpoin_(1,x(ib,1,1),x(ib,1,2),4)
end do
do ib=1,nb !! this loop colors the other type of beads
       red=255*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       green=255*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       blue=255*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       call plscol0_(15, red, green, blue)
       call plcol0_(15)
       call plpoin_(1,x(ib,2,1),x(ib,2,2),4)
end do
do ib=1,nb !! here we plot the links between e+ and e- beads of the same tim
!! and we also plot the cylindrical boundary (just a circle in an x-y plane
       red=127*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       green=127*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       blue=127*(1-(x(ib,2,3)-zmin(2))/zrange(2))
       red=red+127*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       green=green+127*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       blue=blue+127*(1-(x(ib,1,3)-zmin(1))/zrange(1))
       call plscol0_(15,red, green, blue)
       call plcol0_(15)
       !call plline_(2,(/x(ib,1,1),x(ib,2,1)/),(/x(ib,1,2),x(ib,2,2)/))
       call plcol0_(1)
       call plline_(2,(/rcav*cos(2.0*pi*ib/nb),rcav*cos(2.0*pi*(ib+1)/nb)/),
       (/rcav*sin(2.0*pi*ib/nb),rcav*sin(2.0*pi*(ib+1)/nb)/))
end do

call plend_()

end subroutine plot_beads
```

```
!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!!
!! PS_ORIENTATION SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~~~~~~~!!

!! Determines the projection of the separation vector between e+ and e-
!! onto r  direction
!! (thus deducing the theta projection, as well)

subroutine ps_orientation()
do ib=1,nb
        !! determine projection of the Ps atom in the r- direction
        rrel=(x(ib,1,1) - x(ib,2,1))**2 + &
       (x(ib,1,2) - x(ib,2,2))**2 +  (x(ib,1,3) - x(ib,2,3))**2
        r_cm=0.5*sqrt((x(ib,1,1)+x(ib,2,1))**2+(x(ib,1,2)+x(ib,2,2))**2 &
        + (x(ib,1,3)+x(ib,2,3))**2)

        rdot=sqrt((((x(ib,1,1)-x(ib,2,1))*0.5*(x(ib,1,1)+x(ib,2,1)))+&
        ((x(ib,1,2)-x(ib,2,2))*0.5*(x(ib,1,2)+x(ib,2,2))) &
        + ((x(ib,1,3)-x(ib,2,3))*0.5*(x(ib,1,3)+x(ib,2,3))))**2/rrel)
        rdot=(rdot/r_cm)**2


        ibin = int(r_cm/rbinmax*float(norientbins))
        ibin = min(ibin, norientbins-1)  !the outer bin contains all outside
        psrcount(ibin+1) = psrcount(ibin+1)+1
        psrdot(ibin+1) = psrdot(ibin+1)+rdot
end do
end subroutine ps_orientation



!!~~~~~~~~~~~~~~~~~~~~~~~~~!!
!! RADIALBIN SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~~~!!

!! heavily modified from original !!
!! now includes several more figures than it did originally
!! writes to for20_rad_dist, above

subroutine RadialBin()

     do ib = 1, nb
       rrel = (x(ib,1,1) - x(ib,2,1))**2 + &
       (x(ib,1,2) - x(ib,2,2))**2 +  (x(ib,1,3) - x(ib,2,3))**2
       !! full spherical radius -- separation of e+ and e-
       rrel = sqrt(rrel)
```

```
        ibin = int(rrel/rbinmax*float(nbins))
        ibin = min(ibin, nbins-1)  !the outer bin contains all outside
        radbin(ibin+1) = radbin(ibin+1)+1

        rrel = x(ib,2,1)**2 + x(ib,2,2)**2 + x(ib,2,3)**2
        !! distance of e+ from cavity center...rrel is really not a good name
        rrel = sqrt(rrel)
        ibin = int(rrel/rbinmax*float(nbins))
        ibin = min(ibin, nbins-1)  !the outer bin contains all outside
        cavbin(ibin+1) = cavbin(ibin+1) + 1

        !! increment the pickoff zone counter if the positron is in the annihi
        if (rrel.GT.(rcav-1.8893*annihilation_zone_size)) n_in_annihilation_zo
        n_total_binned=n_total_binned+1
        !! note that 1.8893 converts angstroms to au's
      end do

end subroutine RadialBin



!!~~~~~~~~~~~~~~~~~~~~~~~!!
!! READINPUT SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~~!!

!! Reads the input file cavity.in
!! thus gleaning many important parameters

subroutine ReadInput()
  implicit none
  integer :: i,j

  open(unit=11,file='cavity_diel.in',status='old',action='read')
  read(11,*);read(11,*) nb      ! # beads for the particle
  read(11,*);read(11,*) mb      ! # beads moved per staging pass
  read(11,*);read(11,*) npass   ! # staging passes
  read(11,*);read(11,*) amass   ! mass of a single quantum particle
  read(11,*);read(11,*) beta    ! beta = 1/kT (in au where hbar=1)
  read(11,*);read(11,*) hbar    ! making hbar smaller reduces quantum effects
  read(11,*);read(11,*) jump    ! number of passes between printing data
  read(11,*);read(11,*) rcav    ! radius of cylindrical cavity
  read(11,*);read(11,*) aep     ! Yukawa radius
  read(11,*);read(11,*) ninit   ! Flag for initializing beads from a file
  read(11,*);read(11,*) nequil  ! Equilibration steps
  read(11,*);read(11,*) AddToE  !Efield is incremented by AddToE in each run
  read(11,*);read(11,*) zdelta  !size of cartesian bin
  read(11,*);read(11,*) CorrelationCounts ! # of corr calculations
  read(11,*);read(11,*) deg     ! degree of spline
  read(11,*);read(11,*) i,j     ! flags for coulomb prop and external potenti
  if( i == 1 ) then
        UsePolluck = .TRUE.
  else
        UsePolluck = .FALSE.
  end if
  if( j == 1 ) then
        UseExternal = .TRUE.
  else
        UseExternal = .FALSE.
  end if

  read(11,*);read(11,*) ko              ! dielectric constant
  read(11,*);read(11,*) iburnrand       ! thousands of random #'s to be disca
  read(11,*);read(11,*) nleg            ! number of Legendre polynomials to c
                                        ! ~100 for rcav=6, ~150 for rcav=10,

  rcav2 = rcav*rcav

  close(11)

  if(npass <= nequil) then
    write(*,*) 'npass is not greater than nequil :('
    STOP
  end if

  ! The deB wavelength is a useful bit of trivia, so we print it out here.
  wave = sqrt(beta*hbar*hbar/amass) !wave is the free particle deB wavelength
  print*, 'wave ', wave

end subroutine ReadInput



!!~~~~~~~~~~~~~~~~~~~~~~!!
!! TRYBOTH SUBROUTINE !!
!!~~~~~~~~~~~~~~~~~~~~~~!!

subroutine tryboth(xnew)
implicit none
integer :: is = 1 !(rightnow, a dummy) variable for gaussian RNG
real (dp), intent(INOUT), DIMENSION(:,:,:) :: xnew
double precision :: const, g
```

```
 const=2.0d0*wave*wave/dfloat(nb)

x_changed(:,:) = .FALSE.

! pick new bead positions according to  gaussian distn
! id is the axis direction
charge: do ic=1,2
  dim: do id=1,3
    ! we go from the j bead to the j+mb bead
    ! (j is selected at random)
    j=int(nb*ran1(initrand))+1
    beads:  do i=1,mb
      ib=j+mb-i+1
        ! account for periodicity in the chain
        if (ib .GT. nb) ib = ib-nb
        ! the gaussian width depends on which bead we are at
        g=const*dfloat(mb-i+1)/dfloat(mb-i+2)
        xnew(ib,ic,id)= (xnew(ib+1,ic,id)*(mb-i+1)+xnew(j,ic,id))/float(mb-i+
          + gauss(g,is)

        ! flag the fact that this bead has been moved
        x_changed(ib,ic) = .TRUE.

        ! close the chain if we have moved the 1st bead.
        if(ib == 1)then
          xnew(nb+1,ic,id)=xnew(1,ic,id)
        endif
     end do beads
  end do dim
end do charge

end subroutine tryboth



!!fffffffffffffffffff!!
!!~~~~~~~~~~~~~~~~~~~!!
!!~~~~~FUNCTIONS~~~~~!!
!!~~~~~~~~~~~~~~~~~~~!!
!!fffffffffffffffffff!!



!! FUNCTION LIST (alphabetized, and ordered in the program accordingly) ::
!! list created 6.29.2005
!!      gauss           (~ 10 lines)
!!      gcav            (~ 20 lines)
!!      legp            (~ 10 lines)
!!      ran1            (~ 30 lines)
!!      vcoulomb        (~ 10 lines)
!!      vdielec         (~ 60 lines)
!!      vfun1           (~ 10 lines)
!!      vquant          (~ 20 lines)



!!~~~~~~~~~~~~~~~~~~!!
!! GAUSS FUNCTION !!
!!~~~~~~~~~~~~~~~~~~!!

!! Generates numbers taken from a Gaussian distribution
double precision function gauss(g,ix)
implicit double precision (a-h, o-z)
double precision :: rr, ss
integer :: ix

rr = (-dlog(ran1(initrand)+1.0d-10)*g ) ** 0.5
ss = 6.283185307d0*ran1(initrand)
gauss = rr*dcos(ss)

end function gauss



!!~~~~~~~~~~~~~~~~!!
!! GCAV FUNCTION !!
!!~~~~~~~~~~~~~~~~!!

!! Cavity propagator (must be altered for different geometries)
double precision function gcav(z)
implicit none
integer :: ib,ic
real (dp)  :: ra2, rb2, temp, temp_e
real (dp), intent(INOUT), DIMENSION(:,:,:) :: z


gcav = 1.0d0
do ib = 1, nb
ethenp3: do ic = 1,2
        ra2 = z(ib,ic,1)**2 + z(ib,ic,2)**2 + z(ib,ic,3)**2
        rb2 = z(ib+1,ic,1)**2 + z(ib+1,ic,2)**2 + z(ib,ic,3)**2
        temp = ( - (rcav2 - ra2) * (rcav2 - rb2) * nb/2.0/rcav2)
```

```
        temp_e = dexp(temp/beta)
        gcav = gcav * ( 1  - temp_e)
        gcav = max(gcav,1.0D-26)
        if((ra2 .gt. rcav2).or.(rb2 .gt. rcav2)) gcav = 1.0D-26
        end do ethenp3
    end do

end function gcav



!!~~~~~~~~~~~~~~~~!!
!! LEGP FUNCTION !!
!!~~~~~~~~~~~~~~~~!!

!! Calculates all the Legendre polynomials P_n from n=1 to n=nlim at a partic
function legp(y,nlim)

implicit none
integer :: n
integer :: nlim
real (dp) :: y
real (dp), dimension(nleg) :: legp

legp(1) = y
legp(2) = 0.5*(3*(y**2) - 1)

do n=2,nlim-1
        legp(n+1) = 2*y*legp(n) - legp(n-1) - (y*legp(n) - legp(n-1)) / float
end do

end function legp



!!~~~~~~~~~~~~~~~~!!
!! RAN1 FUNCTION !!
!!~~~~~~~~~~~~~~~~!!

!! Generates random numbers via a multiple linear congruential method
!! and a table
double precision function ran1(idum)
implicit none
double precision ::  r(97)
integer, intent(IN) :: idum
save
integer, parameter :: M1=259200,IA1=7141,IC1=54773
real, parameter :: RM1=1.0d0/M1
integer, parameter :: M2=134456,IA2=8121,IC2=28411
real, parameter :: RM2=1.0d0/M2
integer, parameter :: M3=243000,IA3=4561,IC3=51349
integer :: IX1, IX2, IX3, jjj
integer ::  iff=0
if (idum < 0 .or. iff == 0) then
  iff = 1
  IX1 = mod(IC1-idum,M1)
  IX1 = mod(IA1*IX1+IC1,M1)
  IX2 = mod(IX1,M2)
  IX1 = mod(IA1*IX1+IC1,M1)
  IX3 = mod(IX1,M3)
  do jjj = 1,97
    IX1 = mod(IA1*IX1+IC1,M1)
    IX2 = mod(IA2*IX2+IC2,M2)
    r(jjj) = (dfloat(IX1)+dfloat(IX2)*RM2)*RM1
  end do
end if
IX1 = mod(IA1*IX1+IC1,M1)
IX2 = mod(IA2*IX2+IC2,M2)
IX3 = mod(IA3*IX3+IC3,M3)
jjj = 1+(97*IX3)/M3
if (jjj > 97 .or. jjj < 1) PAUSE
ran1 = r(jjj)
r(jjj) = (dfloat(IX1)+dfloat(IX2)*RM2)*RM1
end function ran1



!!~~~~~~~~~~~~~~~~~~~~!!
!! VCOULOMB FUNCTION !!
!!~~~~~~~~~~~~~~~~~~~~!!

!! Computes the total coulomb interaction energy between the e+ and e- beads
double precision function vcoulomb()
implicit none
vcoulomb=0.0
do ib = 1, nb
        vcoulomb=vcoulomb-1.0/(nb*sqrt((x(ib,1,1)-x(ib,2,1))**2+&
        (x(ib,1,2)-x(ib,2,2))**2+(x(ib,1,3)-x(ib,2,3))**2))
end do
end function vcoulomb
```

```
!!~~~~~~~~~~~~~~~~~~~!!
!! VDIELEC FUNCTION !!
!!~~~~~~~~~~~~~~~~~~~!!

!! The energy due to the polarization of the material around the cavity
double precision function vdielec(z,ko,ki)
implicit none
real (dp) :: r, rs, C, v, costheta   ! r is the distance from the origin of t
                                     ! rs is that of the charge producing
real :: ko, ki
integer :: n, ic, ib
integer :: nleglim                   ! limit of number of leg polys we nee
real (dp), dimension(:,:,:) :: z
real (dp), dimension(nleg) :: P
vdielec = 0.0

do ic = 1,2
   do ib = 1,nb
        if (x_changed(ib,ic)) then      ! only bother computing if the bead h

        v = 0.0
        r = ( z(ib,ic,1)**2 + z(ib,ic,2)**2 + z(ib,ic,3)**2 )**0.5
        rs = ( z(ib,mod(ic,2)+1,1)**2 + z(ib,mod(ic,2)+1,2)**2 + z(ib,mod(ic,
            ! the mod business makes sure the index is that of the other char

        if ( (r .ge. rcav) .or. (rs .ge. rcav) ) then
            vdielec = 0.0               ! the series diverges horribly if out
        else
            ! first compute the energy of the e-,e+ due to their own induced

            C = (ki-ko)/(ki*rcav)       ! should be q_producing * q_feeling,
            do n = 0,24                 ! the true series for the first 25 te
                v = v + ((r/rcav)**(2*n)) / (ko + ( float(n)/(float(n+1)) )*k
            end do
            v = v + 1.0/(ki+ko)*((r/rcav)**(2*25))/(1-(r/rcav)**2)      ! geo

            v = v*(1-exp(-(rcav-r)/aep))                ! Yukawa saves beads
            vdielec = vdielec + 0.5*C*v                 ! the infamous 1/2 ou


            ! now compute the "cross" energies of the positron due to the ele
            v = 0.0
            costheta = ( z(ib,1,1)*z(ib,2,1) + z(ib,1,2)*z(ib,2,2) + z(ib,1,3
            ! costheta = rvec *dot* rsvec / r*rs
            if((r.lt.rcav-1.0).and.(rs.lt.rcav-1.0)) then    ! reduced number
                nleglim = min(25,nleg)
            else
                nleglim = nleg
            endif

            P = legp(costheta,nleglim)          ! since I don't know how to n
                                                ! same time as passing its ar

            C = -C              ! q_producing * q_feeling = -1 always for the
            v = 1.0/ko          ! the n=0 term...since I don't have a P(0) in
                                ! Fortran for having the first array index be
            do n = 1,nleglim
                v = v + ( (r*rs/(rcav**2))**n ) / (ko + (float(n)/(float(n+1)

                ! sidenote: yay for cos(theta) being an even function! We don
                ! "current" z-axis, that is, whether the angle is positive or
            end do

            ! no correction since we don't know what the appropriate correcti
            v = v*(1-exp(-(rcav-r)/aep))
            vdielec = vdielec + 0.5*C*v
        endif

        endif
   end do
end do

end function vdielec



!!~~~~~~~~~~~~~~~~~!!
!! VFUN1 FUNCTION !!
!!~~~~~~~~~~~~~~~~~!!

! Yukawa potential energy
double precision function vfun1(r,q,a)
implicit none
real (dp) :: r
real :: q(2),a
vfun1 = q(1) * q(2) * (1-exp(-r/a))/(r)
end function vfun1
```

```
!!~~~~~~~~~~~~~~~~~!!
!! VQUANT FUNCTION !!
!!~~~~~~~~~~~~~~~~~!!

! Quantum potential energy stored in springs between beads
!! (actually, computes total separations squared, and needs to be multiplied
!! by a spring constant to give energy)
double precision function vquant()
implicit none
real (dp) :: dx,dy,dz,dr2

vquant = 0.0
do ic = 1, 2
  do ib = 1, nb-1
    dx = x(ib,ic,1) - x(ib+1,ic,1)
    dy = x(ib,ic,2) - x(ib+1,ic,2)
    dz = x(ib,ic,3) - x(ib+1,ic,3)
    dr2 = dx**2 + dy**2 + dz**2
    vquant = vquant + dr2
  end do
  vquant = vquant + ((x(nb,ic,1)-x(1,ic,1))**2+(x(nb,ic,2)-x(1,ic,2))**2+&
  (x(nb,ic,3)-x(1,ic,3))**2)
end do
return
end function vquant


end PROGRAM PIMC_ES
```