Numerical Ground State Energies of Ringium, Spherium and Glomium for Like-Spin Indistinguishable Fermions

Maximilian Williams u6338634 Supervised by Dr Cedric Simenel

February 2021

Abstract

We present numerical results for the ground state energy of like-spin fermions constrained on the surface of an (n-1) sphere[1] for $n \in \{2, 3, 4\}$ for differing numbers of particles, (n-1) sphere radii and fermion-fermion interaction potentials. The numerical results are produced by a quantum variational Monte Carlo (VMC) algorithm coded from scratch in the goLang programming language.

1 Introduction

Rarely in many-body quantum systems are exact solutions known or obtainable. As such, these problems typically require the use of simplified models to approximate the full behaviour of the system. Even with these simplified models, the dynamics is often fantastically complex and so is either unreachable by analytic methods or requires vast computing resources when evaluated numerically. However, the ground state is much easier to analyse and is the simplest characterstic of the system. The exact analytic ground state can be hard to find and so numerical methods, such as the variational method, are used. If we wish to study spatially uniform systems, then a spatially homogeneous model may be optimal, whereas if we care about behaviour that comes about due to spatial variance, then we should apply a spatially non-homogeneous model. In this work, we study ringiums, spheriums and glomuims which are non-homogeneous particle wave functions confined to the surface of a ring, sphere and 4D hypersphere respectively. These have been studied both analytically and numerically for limited interaction potentials as a basis for non-homogeneous models in fields such as quantum chemistry and nanophysics [16, 25]. Recently, by finding ground state energies for various interaction potentials, fermionic like-spin quantum rings have been shown numerically with the CASINO code to model basic features of nuclear systems [4]. However, this modelling using the CASINO code required non trivial modifications to it's code, making the process undesirable in generalizing to higher dimensions. Such higher dimensional systems, such as spherium and glomuim are more desireable, as they more closely resemble physical systems [17]. Our task then is to be able to find the ground state energies of fermionic ringium, spherium and glomuim systems for various potentials. To do this, a custom program was written by the author in the language goLang to perform Quantum Monte Carlo methods to find the ground state energy of ringium, as well as spherium and glomuim under restricted conditions. In this work, we present the ringium model as well as its generalization to higher dimensions, explain the key algorithms and methods used by the program, explore the effect of different settings on the programs performance and present the results of the program for ringium, spherium and glomuim, with comparisons to other work [4, 15] where available.

2 Mathematical Models Of Ringium and Higher Dimensional Generalizations

Here I introduce models for ringium, higher dimensional ringium and explain my method for finding the intrinsic energy of these systems. The mathematical models for ringium are from other work [15, 17]. Splitting the Hamiltonian into its extrinsic and intrinsic to correct for center of mass motion is my own work.

Ringium

The ringium model is a quasi one-dimensional model that consists of like-spin indistinguishable fermions constrained to move on a 1-sphere[1], commonly know as a ring [4, 17]. We define the usual azimuthal angle around the ring $\phi \in [0, 2\pi)$ and give our ring a radius R, so that the position of the ith particle is ϕ_i. The Hamiltonian, $\hat{H}$ in atomic units ($m = \hbar = c = 1$) is then:

$$\hat{H} = \hat{T} + \hat{V} = -\frac{1}{2R^2} \sum_{i=1}^{N} \frac{\partial^2}{\partial {\theta_i}^2} + \sum_{i<j=1}^{N} V(r_{i,j}), \tag{1}$$

where $r_{i,j}$ is the Euclidean distance between particles i and j and V the interaction potential between the particles [17, 4].

[1] A (n-1) sphere of radius a is all points $(x_1, ..., x_n) \in R^n$ such that ${x_1}^2 + ... + {x_n}^2 = a^2$

Higher Dimensional Ringium

We generalize our model of ringium to higher dimensions, taking the fermions to be constrained on the surface of an (n-1) sphere of radius R, which is the set of points $\{x_1, x_2, .., x_n\} \in R^n$ for which ${x_1}^2 + {x_2}^2 + ... + {x_n}^2 = R^2$ [2]. Note that in our generalization to higher dimensions we have used higher dimensional Cartesian coordinates rather than hyper-spherical coordinates out of ease of implementing derivatives in a general number of dimensions. Similarly to ringium, we define our Hamiltonian as:

$$\hat{H} = \hat{T} + \hat{V} = -\frac{1}{2}\sum_{i=1}^{N} \tilde{\nabla}_i^2 + \sum_{i<j=1}^{N} V(r_{i,j}), \tag{2}$$

where $\tilde{\nabla}_i^2$ is the Laplacian on the surface of the (n-1) sphere in the ith particles coordinates [7, 17].

Intrinsic and Extrinsic Hamiltonians

The Hamiltonians presented above use extrinsic co-ordinates[2] and so when evaluated on a state will give the sum of the intrinsic[3]and extrinsic energies. However, many past results present only intrinsic energies [15], and so it becomes useful to be able to find the intrinsic energy for verification purposes. We define an extrinsic Hamiltonian as:

$$\hat{H}_{ext} = \frac{1}{2M}(\sum_{i=1}^{N} \hat{p}_i)^2, \tag{3}$$

where M is the sum of particle masses and $\hat{p}_i$ is the momentum operator on the ith particle [19]. This extrinsic Hamiltonian is the Hamiltonain associated with the center of mass of the particles, and can be written as:

$$\hat{H}_{ext} = \frac{1}{2M}(\sum_{i=1}^{N} \hat{p}_i^2 + 2\sum_{i<j=1}^{N} \hat{p}_i \cdot \hat{p}_j). \tag{4}$$

As our full Hamiltonian is simply the sum of the intrinsic and extrinsic Hamiltonians, the intrinsic Hamiltonian is given by [19]:

$$\hat{H}_{int} = \hat{H} - \hat{H}_{ext}. \tag{5}$$

In the program, equation 5 is used as it allows for easy switching between finding the total energy and intrinsic energy of the system. Without this method, switching between finding the total and intrinsic energy would amount to switching between intrinsic and extrinsic coordinates. This would require writing near duplicate core functions within the program to deal with each case. In addition, compared to using center of mass co-ordinates, whose construction requires the use of a recursive algorithm [19], the above method is more computationally efficient and easier to implement.

3 Trial Wave Functions

Known techniques for constructing trial wave functions are explained. Solutions to the constant potential Schrödinger equation in higher dimensions are then explored through other works. Finally, an algorithm for filling lowest energy Slater determinants, which is my own work, is presented.

To apply variational methods to our ringium and higher dimensional ringium systems, we require a form of wave function with a finite set of parameters to optimize. For this we follow the method of Jastrow for fermions [10]. We pick an uncorrelated state which we encoded in a Slater determinant Ψ_0^N. Ideally, this Slater determinant is already optimized, being the Hartree-Fock solution or similar. However, in general this need not be the case, and Ψ_0^N can be picked rather arbitrarily, with solutions closer to the full correlated solution being optimal for accuracy [9]. To describe the interaction between particles, we introduce the Jastrow function J and write our multi-particle wave function Ψ^N as [10, 9]:

$$\Psi^N = e^J \Psi_0^N. \tag{6}$$

The Jastrow function is a function of inter-particle distances $r_{i,j}$ and finite parameters α_i which we wish to optimize over. In this work, I choose the form of the Jastrow function defined in [4, 6]:

$$J = \sum_{i<j=1}^{N} [(r_{i,j} - L_c)^3 \Theta(L_c - r_{i,j}) \sum_{k=0}^{\sigma} \alpha_k {r_{i,j}}^k]. \tag{7}$$

Here, Θ is the Heaviside step function, L_c the cut-off distance of interaction and σ the order of the Jastrow function. The Kato cusp condition at $r_{i,j} = 0$ [11] which guarantees the wave function to be continuous as particles approach each other enforces that:

$$\alpha_1 = \frac{3\alpha_0}{L_c} - \frac{\gamma}{{L_c}^3}, \tag{8}$$

[2]Here we take extrinsic coordinates to mean non relative coordinates. If I have two particles, an extrinsic definition of their position would be particle 1 at x_1 and particle 2 at x_2

[3]By intrinsic coordinates, we mean relative coordinates. If I have two particles 1 and 2, then in intrinsic coordinates I describe them as being $x_1 - x_2$ apart.

where $\gamma = \frac{1}{2}$ for our quasi one dimensional ringium [15] and $\frac{1}{4}$ for the higher dimensional systems [21]. We pick our cut-off distance $L_c = \pi R$, where R is the (n-1) sphere radius as this is larger than the maximum displacement of $2R$ between two particles allowing for interaction between all particles in the model.

It should be noted that the Slater determinant Ψ_0^N is an uncorrelated state because each particles state is independent of all others. This is not the case when considering the correlated state $\Psi^N = e^J \Psi_0^N$. This is because states in a configuration with a high Jastrow function value will have a larger magnitude coefficient and so will be more likely to occur. For example, consider a Jastrow that is $J = \sum_{i<j=1} \frac{1}{r_{i,j}}$. For configurations where all the particles are spread out this Jastrow function will be small, while for particles close together it will be large. So, states with their particles clumped together will be more heavily weighted than those who have their particles spread out. If I were to ask the position of one of the particles then, the second particle is more likely to be close by, and so the two are correlated. Our form of the Jastrow is much more general, but uses the same mechanism to induce correlations between particles.

Generating Slater Determinants

The surface of our (n-1) sphere on which the fermions are contained is translationally invariant. As such, the single particle Hartree-Fock effective Hamiltonian is of the form [12]:

$$\hat{H}_{HF} = \hat{T} + \sum_{i=1}^{N} U_{\psi_i}, \tag{9}$$

where U is the effective potential and $\hat{T}$ is the single particle kinetic energy. As our system is translationally invariant, U must also be translationally invariant, and so much be constant. Therefore, by picking our reference energy to be that of U, our Hartree-Fock single particle states are just the free particle solutions [14]. To find these solutions, we solve equation 10

$$\hat{T}\psi = E\psi, \tag{10}$$

where E is the single particle energy, $\hat{T}$ the kinetic energy term of the Hamiltonian and ψ the single particle wave function. We guess the solution to equation 10 to be the hypersperical harmonics of n dimensions, $\gamma_{l_1,...,l_{n-1}}$, where $| l_1 | \leq l_2 \leq ... \leq l_{n-1}$ [7]. These hyperspherical harmonics satisfy

$$\triangle_{s^{n-1}} \gamma_{l_1,..,l_{n-1}} = l_{n-1}(2 - n - l_{n-1})\gamma_{l_1,..,l_{n-1}}, \tag{11}$$

where n is the dimension, and $\triangle_{s^{n-1}}$ is the spherical Laplace operator on the $n-1$ sphere [7]. For example, in three dimensions $n = 3$, equation 11 becomes:

$$[\frac{1}{\sin\theta}\frac{\partial}{\partial\theta}(\sin\theta\frac{\partial}{\partial\theta}) + \frac{1}{\sin\theta^2}\frac{\partial^2}{\partial\phi^2}]\gamma_{l_1,l_2} = -l_2(l_2+1)\gamma_{l_1,l_2}, \tag{12}$$

where θ, ϕ are the usual polar and azimuthal angles respectively [7]. By comparison between equations 10 and 11, as well as ignoring the wave functions normalization and radial dependence, we get the free particle wave functions $\psi_{l_1,l_2,...,l_{n-1}}$ and energies as $E_{l_{n-1}}$:

$$\psi_{l_1,l_2,...,l_{n-1}} = \gamma_{l_1,l_2,...,l_{n-1}}, E_{l_{n-1}} = -l_{n-1}(2 - n - l_{n-1}). \tag{13}$$

It should be noted that the energy of the wave functions in equation 13 are degenerate in indices $l_i, i \neq n-1$. Nevertheless, these are the Hartree-Fock orbitals and their associated energies [14]. We now introduce an algorithm for picking k of the lowest energy wave functions. We represent the indices as a vector $(l_1, .., l_{n-1})$. We generate the first vector by setting all indices to zero. (1) We then generate the next vector by looking at its rightmost undetermined entry, finding the smallest magnitude that it can be so that our new vector will not equal a previously chosen vector and repeat with the next entry to its left (2). (1) to (2) is then repeated until we have the desired number of vectors. For example, suppose that we had $n = 3$ and we want to select 4 vectors. We initialize by making the vector $(0, 0)$. We would then generate the next vector through the process between (1) and (2), taking its last index as 1 and its first index as 0 giving $(0, 1)$. The next would have its last index 1 and its first 1 giving $(1, 1)$. Finally, we would generate a vector with its last index 1 and its first index -1 as 1 is no longer available giving $(-1, 1)$.
For k particles then, we employ the algorithm above to generate k single particle states $\psi^{(1)}, .., \psi^{(k)}$. These k states are then arranged in a Slater determinant Ψ_0^N

$$\Psi_N^0 = \begin{vmatrix} \psi^{(1)}(x_1) & \psi^{(1)}(x_2) & ... & \psi^{(1)}(x_k) \\ \psi^{(2)}(x_1) & \psi^{(2)}(x_2) & ... & \psi^{(2)}(x_k) \\ . & . & ... & . \\ \psi^{(k)}(x_1) & \psi^{(k)}(x_2) & ... & \psi^{(k)}(x_k) \end{vmatrix}, \tag{14}$$

where x_i is the position of the ith particle. By generating the Slater determinant in this fashion, we have generated a low energy uncorrelated state which we can correlate using the Jastrow function as described above. It should be noted that in generating this Slater determinant, we have used extrinsic coordinates and so any properties derived using this state, say energy will have an extrinsic component, for this reason, where the method of splitting the Hamiltonian described in section *Intrinsic and Extrinsic Hamiltonians* is not available, as is currently the case in higher dimensions, we must pick a Slater determinant without any center of mass motion or submit our results as the sum of intrinsic and extrinsic energies.

4 Numerical Methods

Known numerical methods for differentiation, integration and sampling are explored, with a focus on how they scale with the dimension of the domain. The sampling of Gaussians on a (n-1) sphere is my own work.

The programs functions, dealing with integration, sampling and differentiation don't assume the form of their input. As such, the program uses numerical methods over their analytic analogs due to their ease of implementation and more consistent and predicable computational requirement.

Numerical Differentiation

Numerical differentiation is a method for approximating the derivative of functions. It has particular advantages over analytic differentiation when the function is non analytic, defined for only some points, the derivative is computationally expensive to compute and has the advantage of being easier to implement in the program. As we wish to deal with very general wave functions which must be differentiated many times over, we pick numerical differentiation for our program.

Consider a function $f : R \to R$ which we wish to take the derivative of about some $a \in R$. We consider $f(a \pm \Delta x)$ and Taylor expand about a producing:

$$f(a+\Delta x) = f(a) + \Delta x \frac{df}{dx}\bigg|_{x=a} + \frac{\Delta x^2}{2}\frac{d^2 f}{dx^2}\bigg|_{x=a} + \frac{\Delta x^3}{6}\frac{d^3 f}{dx^3}\bigg|_{x=a} + \mathcal{O}(\Delta x^4) \tag{15}$$

and,

$$f(a-\Delta x) = f(a) - \Delta x \frac{df}{dx}\bigg|_{x=a} + \frac{\Delta x^2}{2}\frac{d^2 f}{dx^2}\bigg|_{x=a} - \frac{\Delta x^3}{6}\frac{d^3 f}{dx^3}\bigg|_{x=a} + \mathcal{O}(\Delta x^4). \tag{16}$$

By rearranging equation 15 we can get the forward space numerical derivative which is accurate to $\mathcal{O}(\cdot\S)$.

$$\frac{df}{dx}\bigg|_{x=a} = \frac{f(a+\Delta x) - f(a)}{\Delta x} + \mathcal{O}(\Delta x) \tag{17}$$

Similary, using equation 16, the backward space numerical deriative can be found.

$$\frac{df}{dx}\bigg|_{x=a} = \frac{f(a) - f(a-\Delta x)}{\Delta x} + \mathcal{O}(\Delta x) \tag{18}$$

Finally, buy subtracting equation 15 and 16 we can get the central space numerical derivative:

$$\frac{df}{dx}\bigg|_{x=a} = \frac{f(a+\Delta x) - f(a-\Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^2). \tag{19}$$

As the central space derivative is 2nd order accurate, and we are guaranteed to have points $a+\Delta x$ and $a-\Delta x$ unobstructed by boundary conditions during our simulation, we use the central numerical derivative from this point forth.

For functions $f : R^N \to R$ the numerical derivative is easily generalized by keeping all but one variable constant.

$$\frac{\partial f}{\partial x_i}\bigg|_{x_1,..,x_i,..,x_N = a_1,...,a_i,..,a_N} = \frac{f(a_1,...,a_i+\Delta x_i,..,a_N) - f(a_1,...,a_i-\Delta x_i,..,a_N)}{2\Delta x_i} + \mathcal{O}(\Delta {x_i}^2). \tag{20}$$

and for higher order derivatives we simply apply to process used in equation 19 or 20 once more to $\frac{df}{dx}$ and $\frac{\partial f}{\partial x_i}$ to get their second order derivatives which are also second order accurate. Other derivative operators, such as grad and div can then be built out of sums of terms in equation 19 and 20.

In the program, the only boundary condition applicable to differentiation is a periodic boundary condition in one coordinate, call it $x_j \in [0, b)$, with a periodic boundary at $x_j = 0, b$. To incorporate this, the program sets x_j to x_j modulus b whenever the variable is used in a derivative, so in this case:

$$\frac{\partial f}{\partial x_j}\bigg|_{x_1,..,x_j,..,x_N = a_1,...,a_j,..,a_N} = \frac{f(a_1,...,mod(a_j+\Delta x_j, b),..,a_N) - f(a_1,...,mod(a_i-\Delta x_j, b),..,a_N)}{2\Delta x_j} + \mathcal{O}(\Delta {x_j}^2), \tag{21}$$

where $mod(x_j, b)$ is the modulus of x_j base b and all other notion carries over from equation 20.

Numerical Integration

Numerical integration is a class of methods used to find approximate solutions to integrals in a bounded domain. Numerical integration is useful over analytic integration when the integrands value is not known for some points or the antiderivative is expensive/impossible to find. In one dimension, a common method is to slice the domain, approximate the value in each slice by the value of the function in that slice and add up the value multiplied by width contribution from each slice. Common variations of this method are the trapezoidal rule, midpoint rule and Simpson's rule [23]. For higher dimensional integrals, we can break them up into 1 dimensional integrals and apply one of the above methods. However, these methods become computationally expensive as the number of computation points grows exponentially with the dimension [3]. To compute higher dimensional integrals, Monte Carlo methods are used in the program, as they do not suffer from exponentially growing numbers of computational points [3].

Monte Carlo Integration

If we have a higher dimensional integral

$$I = \int_\Omega f(\vec{x}) d\vec{x}, \tag{22}$$

where $\Omega \subset R^m$, then by taking N random samples $\vec{S_j} \in \Omega$,

$$I \approx I_N = \frac{V}{N} \sum_{j=1}^{N} f(\vec{S_j}), \tag{23}$$

where V is the volume of the space Ω and is defined by [1]:

$$V = \int_\Omega d\vec{x}. \tag{24}$$

The uncertainty in I_N, as measured by one standard deviation is:

$$\Delta I_N \approx \frac{V \sigma_f}{\sqrt{N}}, \tag{25}$$

where σ_f is the standard deviation of f acting on the random samples $\vec{S_j}$ [1]. However, this method of integration becomes inaccurate for a set number of computational points when f has a wide distribution of values. When possible, we employ importance sampling which samples non uniformly in the domain in order to minimise the variance in the samples σ_f [1].

Importance Sampling

If our numerical integral can be written in the form:

$$I = \int_\Omega P(\vec{x}) g(\vec{x}) d\vec{x}, \tag{26}$$

where $P(\vec{x})$ is a non uniform probability distribution, then we can employ importance sampling [5]. Similarly to normal Monte Carlo integration, we sample N elements $\vec{S_j} \in \Omega$, but rather than sample them uniformly, we sample them from the probability distribution P using the Metropolis-Hastings algorithm as it is computationally efficient for sampling distributions [1, 5]. In a similar fashion then,

$$I \approx I_N = \frac{1}{N} \sum_{j=1}^{N} f(\vec{S_j}). \tag{27}$$

Sampling Probability Distributions and the Metropolis Algorithm

The Hastrings-Metropolis algorithm is used to sample a probability distributions whose value at any point is difficult or time consuming to obtain [5]. We suppose that we have a domain Ω and a probability distribution P on Ω that we wish to sample from. We initialize by picking a point $x \in \Omega$. Then we pick a probability distribution g which is symmetric $(g(x|y) = g(y|x))$ [20] and use it to generate a candidate point x_c. A random number r between 0 and 1 is then picked. If $r < P(x_c)/P(x)$ then we record x_c and move to the candidate position by setting $x = x_c$. If $r > P(x_c)/P(x)$ then we record x and don't move to the candidate position, not changing x. We then repeat the process over and over (excluding the initialization step) producing a list of samples [5, 20]. Conceptually the Hastings-Metropolis algorithm can be thought of as probabilistically seeking peaks in the probability distribution function P, sampling more times from regions of high probability than those with low probability. However, this method of sampling is not perfect, and care should be taken when picking the distribution g. A g with too high or too low a standard deviation will make the samples converge more slowly to P [5]. Additionally, the initial samples generated by the Hastings-Metropolis algorithm are typically biased by the initialization position and so the first 10 percent are discarded in our program [8].

Sampling of Points on a (n-1) Sphere

Monte Carlo Integration with and without importance sampling require us to sample uniformly and non uniformly in our domain. With our domain being the surface of a (n-1) sphere, such sampling is non-trivial. In the case without importance sampling, we need to be able to uniformly sample directly from the surface of the (n-1) sphere, whereas in the case with importance sampling, we need to be able to sample the surface using a symmetric distribution g.

Uniformly Sampling on a (n-1) Sphere

To randomly pick a point on the surface of the (n-1) sphere ${x_1}^2 + ... + {x_N}^2 = R^2$, we pick each x_i randomly from $[0, 1]$. Then we divide each x_i by $\sqrt{{x_1}^2 + {x_2}^2 + ... + {x_N}^2}$ putting the vector $(x_1, .., x_N)$ on the surface of the n-sphere. By multiplying each x_i by R we have randomly generated a point on the (n-1) sphere of radius R [18].

Sampling a Gaussian on a (n-1) Sphere

To sample using a Gaussian on our (n-1) sphere we use the following algorithm. We first pick a normalized one dimensional Gaussian $g(x, y, \sigma)$ that has variable x, center y and standard deviation σ. We pick a point $(y_1, ..., y_i, .., y_N)$ to sample about on the surface of our (n-1) sphere of radius R. We then project this point onto the unit (n-1) sphere in primed co-ordinates $y_i' = \frac{y_i}{R}$. Then we sample α_i from $g(x_i', y_i', \sigma)$ for $i \in \{1, ..., N\}$ and let $y_i'' = y_i' + \alpha_i$. To project this back onto the radius R (n-1) sphere we define $y_i''' = \frac{y_i''}{\sqrt{{y_1''}^2 + ... + {y_N''}^2}} R$. The vector $(y_1''', ..., y_N''')$ is then our new sampled point on the surface of the (n-1) sphere. Importantly, this method is symmetric, it is just as likely to start at $(y_1, .., y_N)$ and sample $(y_1''', ..., y_N''')$ as it is to start at $(y_1''', ..., y_N''')$ and sample $(y_1, .., y_N)$. This can easily be seen by considering that every point on the surface of the (n-1) sphere is identical. This makes this method of sampling points on a sphere applicable to sampling in our Metropolis-Hastings algorithm and so is used in our program.

5 Zero-Temperature QMC Methods

A broad look at zero-temperature quantum Monte Carlo (QMC) methods is given. One algorithm, variational Monte Carlo (VMC) is explained in detail with a genetic algorithm used over other optimization techniques. The application of a genetic algorithm to VMC was developed independently but has been explored in other work [24].

The Variational Monte Carlo (VMC) algorithm together with the Diffusive Monte Carlo (DMC) algorithm are the main algorithms used in zero-temperature Quantum Monte Carlo (QMC), a class of algorithms designed to find the ground state energy of a quantum system. The two algorithms are complimentary, with VMC being better at finding approximate solutions fast, and DMC better for accuracy, but slower to converge [20]. For this reason, in other programs such as CASINO that use QMC, both VMC and DMC are used, with VMC being used initially to find the approximate solution until DMC takes over to find a more exact solution [20]. Here, our program only uses VMC as it is the more general and the easier to implement of the two.

The VMC Algorithm

The VMC algorihtm follows the variational principle, estimating a wave function, evaluating its energy and then changing the wave function to minimize this energy [20]. We pick a trial wave function $\phi(\alpha, x)$ which takes parameters α and variables x. The energy associated with this wave function parameterized by α, $E(\alpha)$ is then:

$$E(\alpha) = \frac{1}{\int_\Omega \mid \phi(\alpha, x) \mid^2 dx} \int_\Omega \phi^*(\alpha, x) \hat{H} \phi(\alpha, x) dx, \tag{28}$$

where Ω is the space in which x exists. Equation 28 can then be rewritten as [20]:

$$E(\alpha) = \int_\Omega P(\alpha, x) E_l(\alpha, x) dx, \tag{29}$$

where,

$$P(\alpha, x) = \frac{\mid \phi(\alpha, x) \mid^2}{\int_\Omega \mid \phi(\alpha, y) \mid^2 dy} \tag{30}$$

and,

$$E_l(\alpha, x) = \frac{\hat{H}\phi(\alpha, x)}{\phi(\alpha, x)}. \tag{31}$$

In this form, the integral in equation 30 and 29 can be evaluated by Monte Carlo integration with and without importance sampling respectively. We can therefore evaluate the energy E associated with the wave function for given parameters α and we are left to search over all αs to minimise E.

Search Algorithms

Traditionally there are two main methods for minimising E in parameter space. The first minimises variance in energy across parameter space and the second directly minimises energy by diagonalization of the Hamiltonian [20]. Both of these techniques were found to be difficult to implement. Instead, a genetic algorithm was used to minimise the energy directly.

Genetic Algorithms (GAs)

Genetic algorithms are a class of evolutionary algorithms that rely on simulated natural selection and genetic crossover to optimize or search [13]. In our case we wish to optimize $E(\alpha)$ over α. We first generate a random population P of size *GAPopSize* that is a set of randomly generated parameters. (1) We then give each element of the population $p \in P$ a score $E(p)$. The lowest *breedingFraction*GAPopSize* scoring ps in P are then "bred" with each other producing a new population P the same size as the original (2). In our case, this breeding step is done by randomly selecting two of the lower scoring ps in P and taking the average of their *alpha* values while adding some random noise of magnitude *geneticNoise*. We then repeat (1) to (2) epoch number of times until we reach a final population P [22]. The parameters here, *GAPopSize*, *breedingFraction*, *geneticNoise* and *epochs* can all be tuned so that the genetic algorithm produces a population with as low a score as possible. In general, increasing *GAPopSize* and *epochs* improve the accuracy of the genetic algorithm, while *breedingFaction* being small leads to quick, but inaccurate convergence, *GAPopSize* being large leads to slow but accurate convergence. *geneticNoise* being too large can lead to population divergence, while being too small can lead to inaccurate convergence [22]. Once terminated the genetic algorithm with adequate *GAPopSize*, *breedingFaction*, *geneticNoise* and *epochs* produces a set of parameters which are the approximate solution to minimise E in parameter space. We then can act E on each of this set of optimal parameters to produce a set of energies $\{E_{optimal}\}$. The average of this set is then used as the value of the minimum energy of the system and its standard deviation used as our uncertainty in this measurement.

6 Program

Important features of the program, its inputs, broad view of how it runs and constraints on what it can compute are explained.

Program Inputs and outputs

To run the program, a *setupState* must be specified by the user. This *setupState* encodes everything the program needs to know about the problem it is being presented with to run. The program then outputs its data for the ground state energy of the defined problem in a csv file.

Variable	Type	Description
R	float64	Radius of Ringium
particleNumber	int	Number of particles simulated
dimension	int	Dimension of the problem, for ringium this is set to 1
Lc	float64	Cutoff length for interactions
setupSize	float64	Size of step used in numerical differentiation
numberOfPoints	float64	Number of points used in Monte Carlo Integration
pot	potential	Interaction potential between particles
order	int	Order of the Jastrow function
geneticNoise	float64	Random noise when mutating in the genetic algorithm
breedingFraction	float64	Percentage of population that gets to breed in the genetic algorithm
epochs	int	Iterations of the genetic algorithm
elitePercent	float64	Percent of population that does not die between epochs
includeCM	bool	Whether to include Kinetic energy from center of mass motion
GAPopSize	int	Number of individuals used in each population of the genetic algorithm
MaxJastroMagnitude	float64	Maximum value of the Jastrow factor

Figure 1: Important variables encoded in the *setupState* of the program

Program Execution

The program takes a *setupState* and returns an estimate of a ground state energy per particle for the specified system. It does this in three main steps.

1. Generating the trial state given a set of parameters
2. Evaluating the trial state energy
3. Repeating 1. and 2. to search for optimal parameters

In 1 we take our *setupState* and a set of parameters to generate a Slater determinant and Jastrow factor which we combine to generate a wave function Ψ^N. In 2 we use numerical differentiation functions to compute the derivatives of the wave function. Together with our potential specified in the *setupState* allows us to compute $\Psi^{N*}\hat{H}\Psi^N$ at a single point in space, where $\hat{H}$ is either to total or intrinsic Hamiltonian. Monte Carlo Integration is then used to integrate $\Psi^{N*}\hat{H}\Psi^N$ to estimate the ground state energy given a set of parameters. In 3 we use the ability to evaluate a states energy given a set of parameters to search the space of parameters bound by MaxJastroBound for a minimum using a genetic algorithm.

Constraints

The program can be split in two. One that deals with only ringium, called ringium.go and uses spherical polar co-ordinates and one that deals with the higher dimensional generelizations like spherium and glomuim called general.go. Both can compute the ground state energy of a given system, but only for ringium.go does the *includeCM* setting work. That is, for ringium, the program can produce intrinsic and intrinsic plus extrinsic ground state energies, whereas for spherium and glomium the program can only produce intrinsic plus extrinsic ground state energies.

7 Results

Interaction potentials used throughout results are explained. The results of ringium.go are then compared with known analytic results [15, 17]. Program settings are then varied to show their impact on the programs convergence and accuracy. Numerical results produced by CASINO from other work [4] is then compared with results from ringium.go and general.go for ringium to confirm both programs accuracy. Finally, spherium and glomuim ground state energy results are pressented.

Interaction Potentials

Various potentials are used to compare the programs results with other work [4]. Here all potentials will be functions of interparticle distance between particle i and j which we call $r_{i,j}$. The first two potentials are the repulsive and attractive Coulomb potentials

$$V_{cpos}(r_{i,j}) = \frac{1}{r_{i,j}} \tag{32}$$

and,

$$V_{cneg}(r_{i,j}) = -\frac{1}{r_{i,j}} \tag{33}$$

respectively. We then have the zero potential:

$$V_{zero}(r_{i,j}) = 0. \tag{34}$$

Finally, we borrow the nuclear 1 and nuclear 2 potentials from [4] defined as:

$$v_{nuc1}(r_{i,j}) = \frac{1}{r_{i,j}}[100e^{-2r_{i,j}} - 64e^{-1.5r_{i,j}}] \tag{35}$$

and,

$$v_{nuc2}(r_{i,j}) = \frac{1}{r_{i,j}}[12e^{-2r_{i,j}} - 8e^{-1.0r_{i,j}}]. \tag{36}$$

In the program, each of these potentials are their own function and can be easily interchanged, appended to or customized for future work.

Comparison To Known Analytic Results

For two particle Ringium with a repulsive Coulomb potential, analytical results for the intrinsic ground state energy per particle of $\frac{9}{92}$ and $\frac{1}{3}$ have been found for radii of $R = \sqrt{\frac{23}{2}}$ and $R = \sqrt{\frac{3}{2}}$ [15]. Figure 2 and 3 show that as we iterate our genetic algorithm, the program asymptotically approaches the exact value within uncertainty.

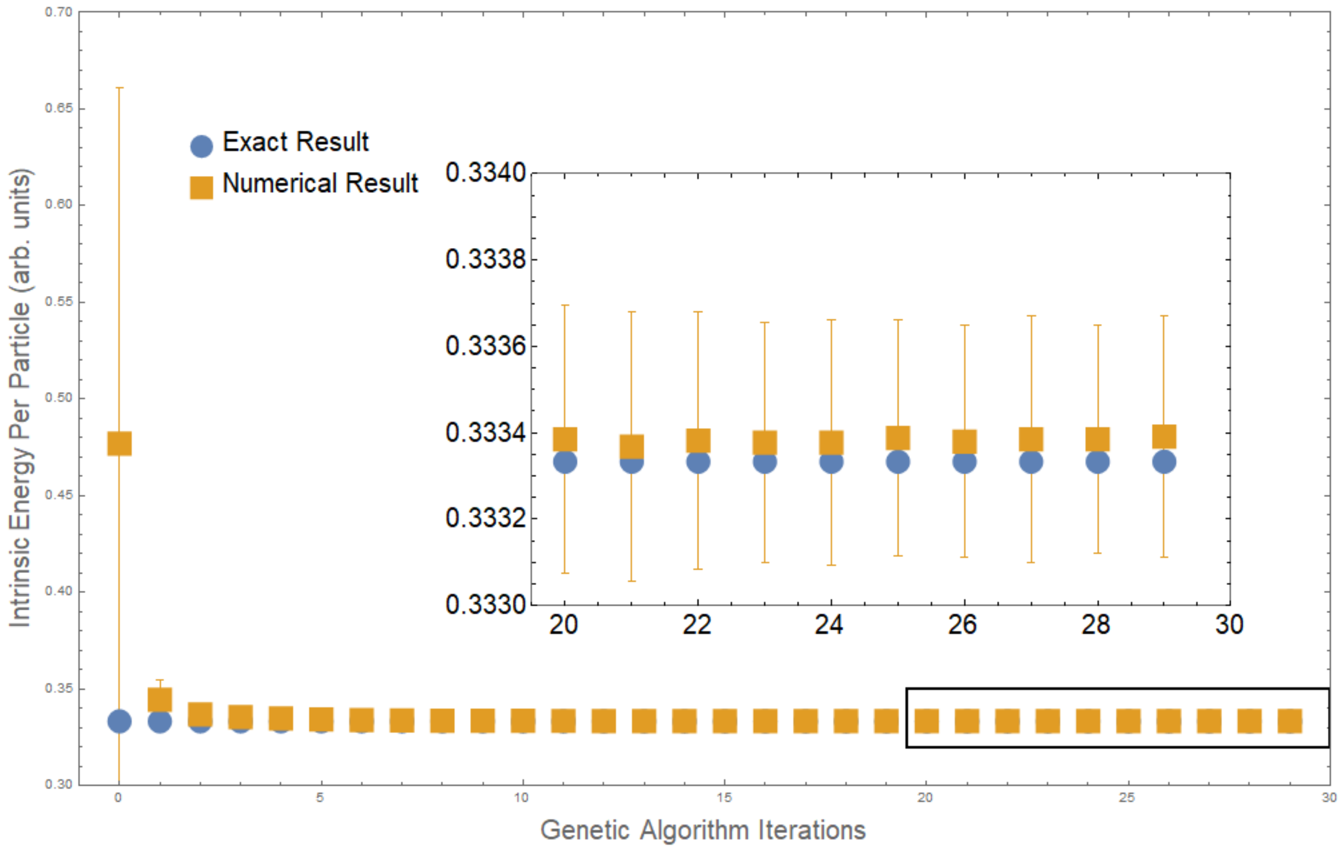

Figure 2: Convergence of ringium.go to analytic result for two particle ringium with radius of $\sqrt{\frac{3}{2}}$. Here circular blue data points are exact analytic results, yellow square data points are generated from ringium.go.

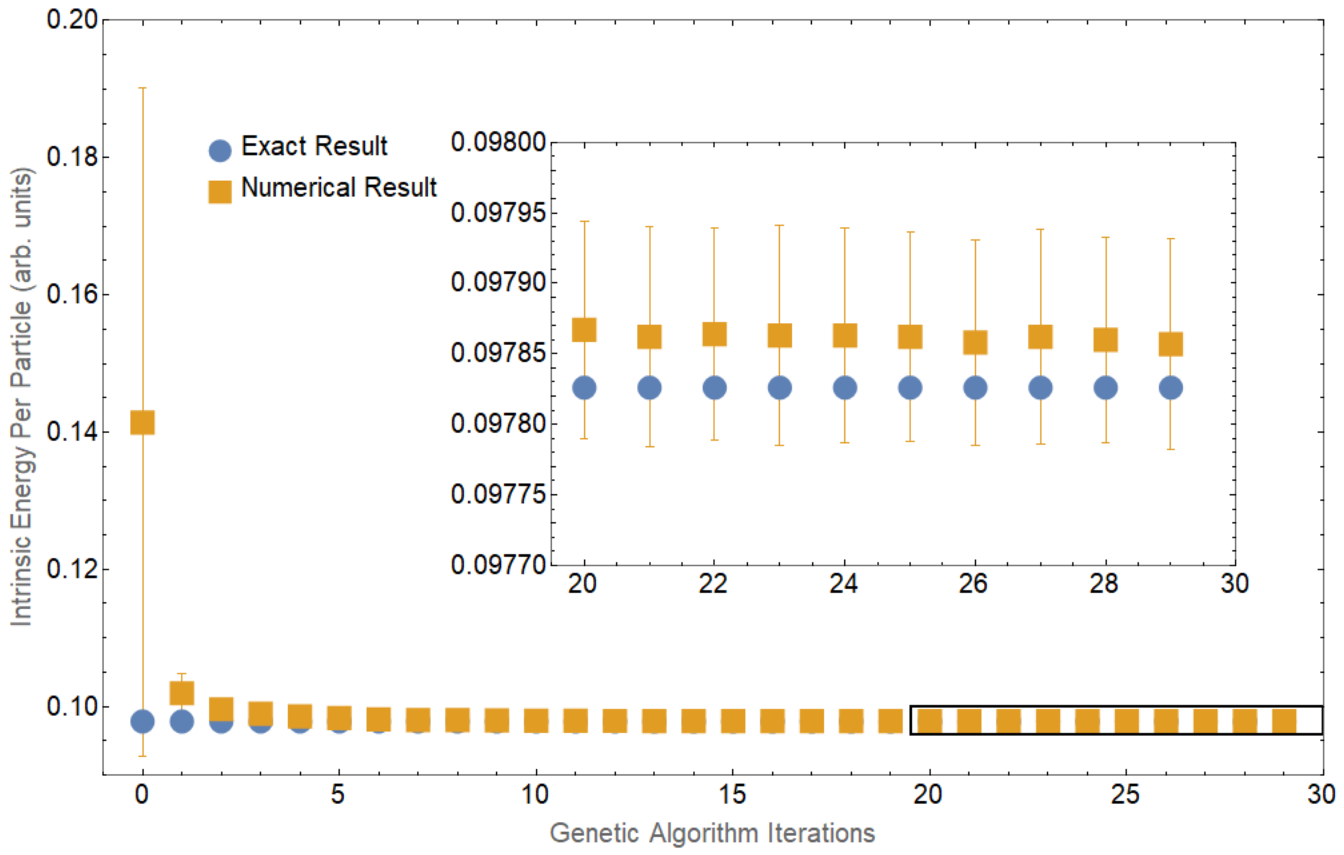

Figure 3: Convergence of ringium.go to analytic result for two particle ringium with radius of $\sqrt{\frac{23}{2}}$. Here circular blue data points are exact analytic results, yellow square data points are generated from ringium.go.

Affect of Varying Settings

As seen in figure 4, using a higher number of points in the Monte Carlo Integrals (yellow square) reduces the uncertainty and provides a closer result to the exact (green diamond) when compared to using lower number of integration points (blue circle).

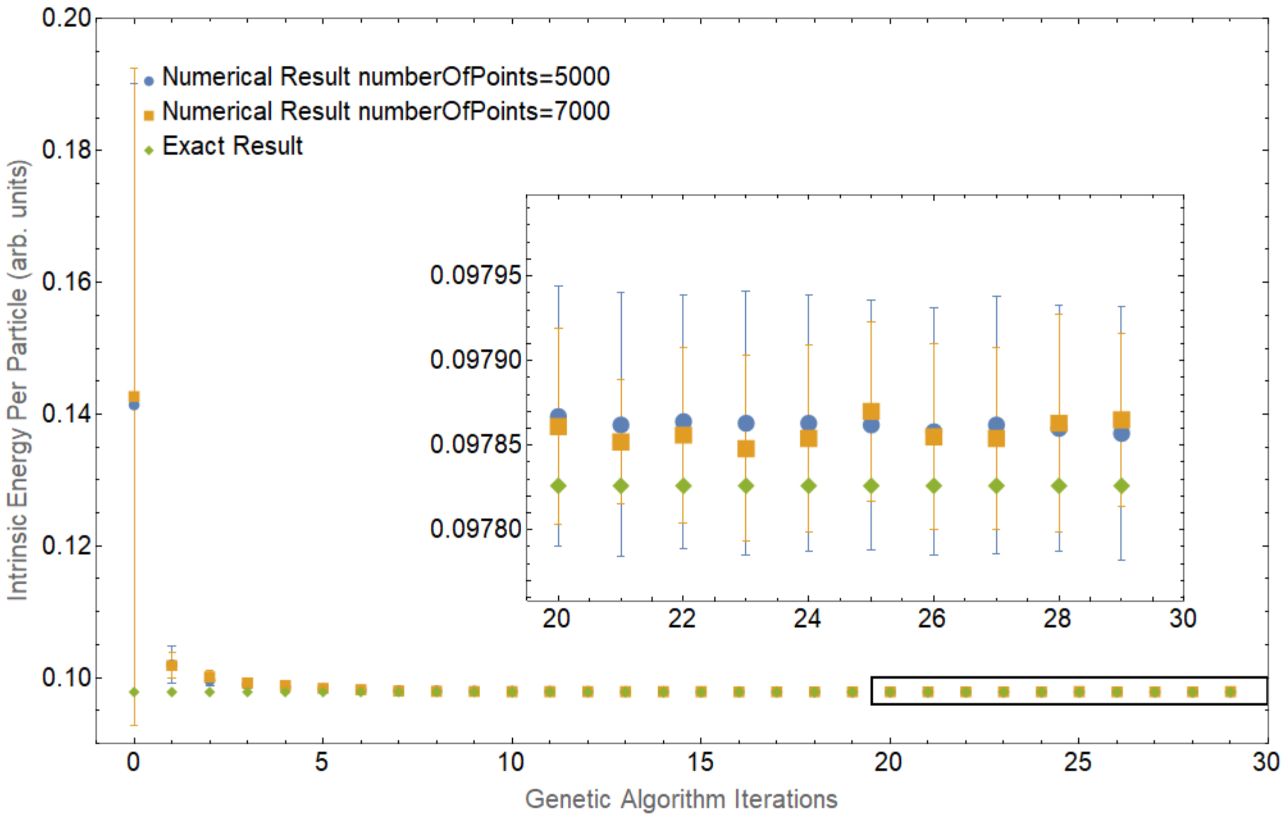

Figure 4: Comparison of convergence to exact results (green diamonds) for a higher and lower *numberOfPoints* represented by yellow squares and blue circles respectively. *numberOfPoints* is the number of points used in Monte Carlo integration. Here the system is two particle ringium with a repulsive Coulomb potential and a radius of $\sqrt{\frac{23}{2}}$

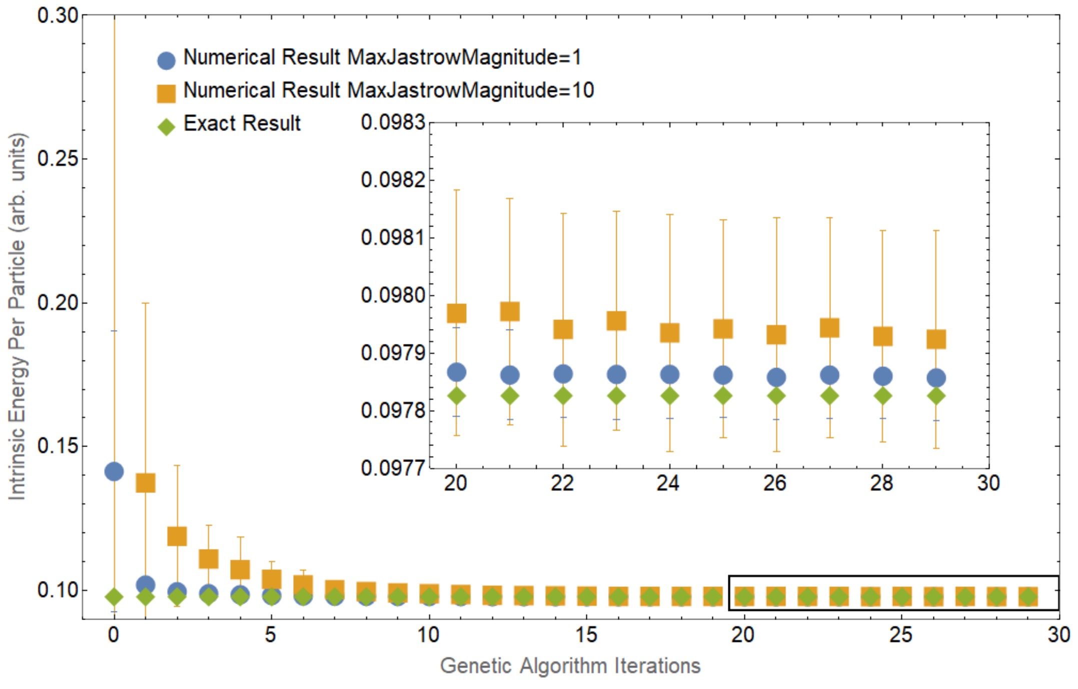

Figure 5: Comparison of convergence for higher Jastrow bound data (yellow square) and lower Jastrow bound data (blue circle) to exact results (green diamond) for two particle ringium with repulsive Coulomb potential and radius $\sqrt{\frac{23}{2}}$.

In the case of the positive Coulomb potential, figure 5 shows that when the Jastrow bound is made smaller (blue circle data), the convergence is faster and result more uncertain when compared to higher Jastrow bound data (yellow square). However, smaller Jastrow bounds are not generally beneficial, as they constrain the magnitude of the Jastrow factor, and so weaken the maximum correlation the wave function can describe. This can be seen in figure 6, where the higher Jastrow bound (blue circle) finds a lower and thus better ground state approximation than the lower Jastrow bound (yellow square).

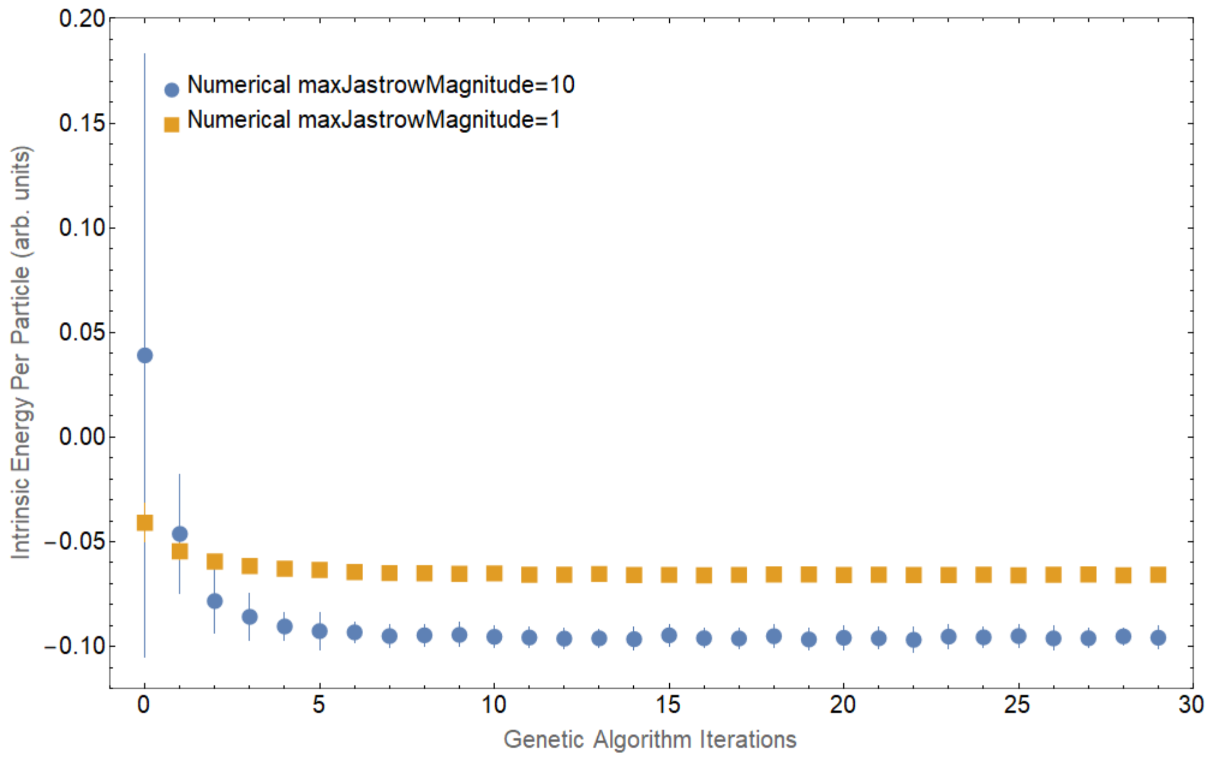

Figure 6: Comparison of convergence between Jastrow bounds of 1 (Yellow square) and 10 (blue circle) for two particle Ringium with radius of 10 and attractive Coulomb potential

The population used in the genetic algorithm must also be considered, for genetic population too small compared to the search space, the genetic algorithm may converge to a local minima rather than a global one, similar to equation 6. This affect is shown in figure 7, with the high genetic population (blue circle) converging closer to the exact solution (green diamond) compared to the low genetic population (yellow square) data. The eractic nature of the low population (yellow square) data should also be noted, with error bars shrinking and growing throughout convergence compared to the gradual error bar shrinking of the high population data (blue circle). In general then, a higher genetic population is desirable for the program, as it provides convergence that is more likely to be correct and is less erratic in its convergence. However, increasing the genetic population comes at the cost of compute time, linearly, and so should be adjusted based on hardware.

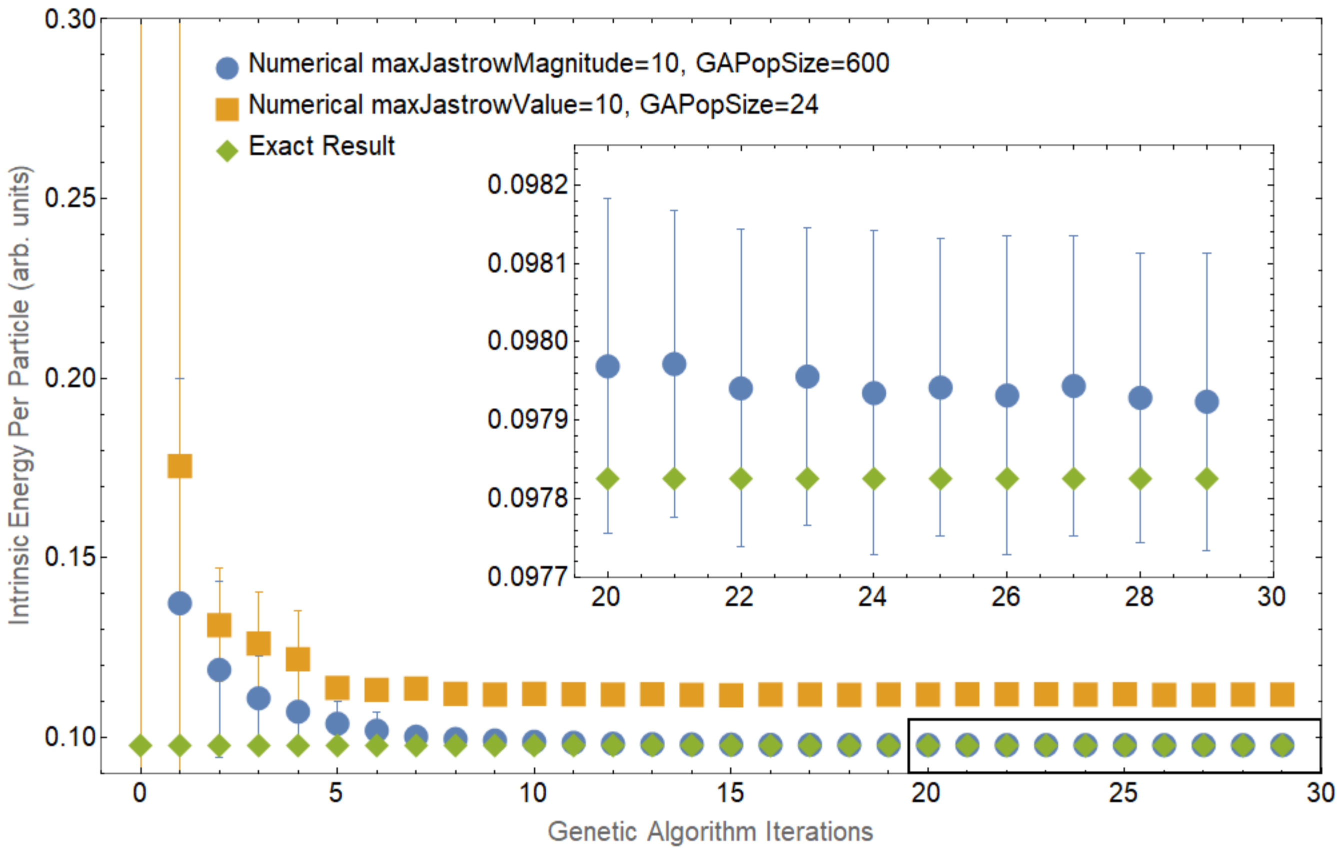

Figure 7: Comparison of convergence for a higher genetic algorithm population (blue circle) and lower genetic algorithm population (yellow square) to exact results (green diamond) for two particle ringium with repulsive Coulomb potential and radius $\sqrt{\frac{23}{2}}$.

In summary then, the programs convergence and certainty are dependent on the genetic population ($GAPopSize$), Jastrow factor bound (MaxJastrowMagnitude), number of points used in Monte Carlo Integrals ($numberOfPoints$). with larger genetic population and Monte Carlo integral points producing faster and more accurate convergence. The Jastrow factor bound however should be made as large as possible while considering that larger genetic populations are required for larger Jastrow factor bounds. With larger values of each of these programs then, we require more compute time.

Numerical Comparison

In this section, the results of ringium.go are compared to numerical results of CASINO for 5 particle ringium. General.go is then compared to ringium.go for 3 particle ringium.

As seen in figure 8, the ringium.go program reproduces the results of the CASINO program within uncertainty for various potentials and radii. However, the uncertainty displayed in figure 8 varies widely between potentials and over radii. At large radii, this is especially the case for the attractive Coulomb potential and nuclear 2. Both of these potentials were numerically shown to deviate substantially from their Hartree-Fock solution at these radii [4]. As such, their optimal Jastrow is non trivial and consequently harder for the program to find, leading to higher uncertainty in these regions. A similar argument explains the wide error bars for all radii for the nuclear 2 potential.

Due to general.go's more general less effecient algorithms, comparison between ringium.go and general.go for three particles is used over a 5 particle comparison between general.go and the CASINO results from [4].
For three particles, ringium.go and general.go agree on all data points. The major difference between two datasets is the uncertainty bars on the nuclear 1 potential for radii of 0.75 and 1, with ringium.go's error being an order of magnitude above general.go's. When runnning the program a second time on the two outlier points, the result could not be replicated. Because of this, and the small genetic population and epochs used in generating the data for figure 9, this can be explained by an unfavourable random walk in either the genetic algorithm or importance sampling Monte Carlo integration. In future, a higher genetic population and more epochs together with more compute time should be used to avoid this effect.

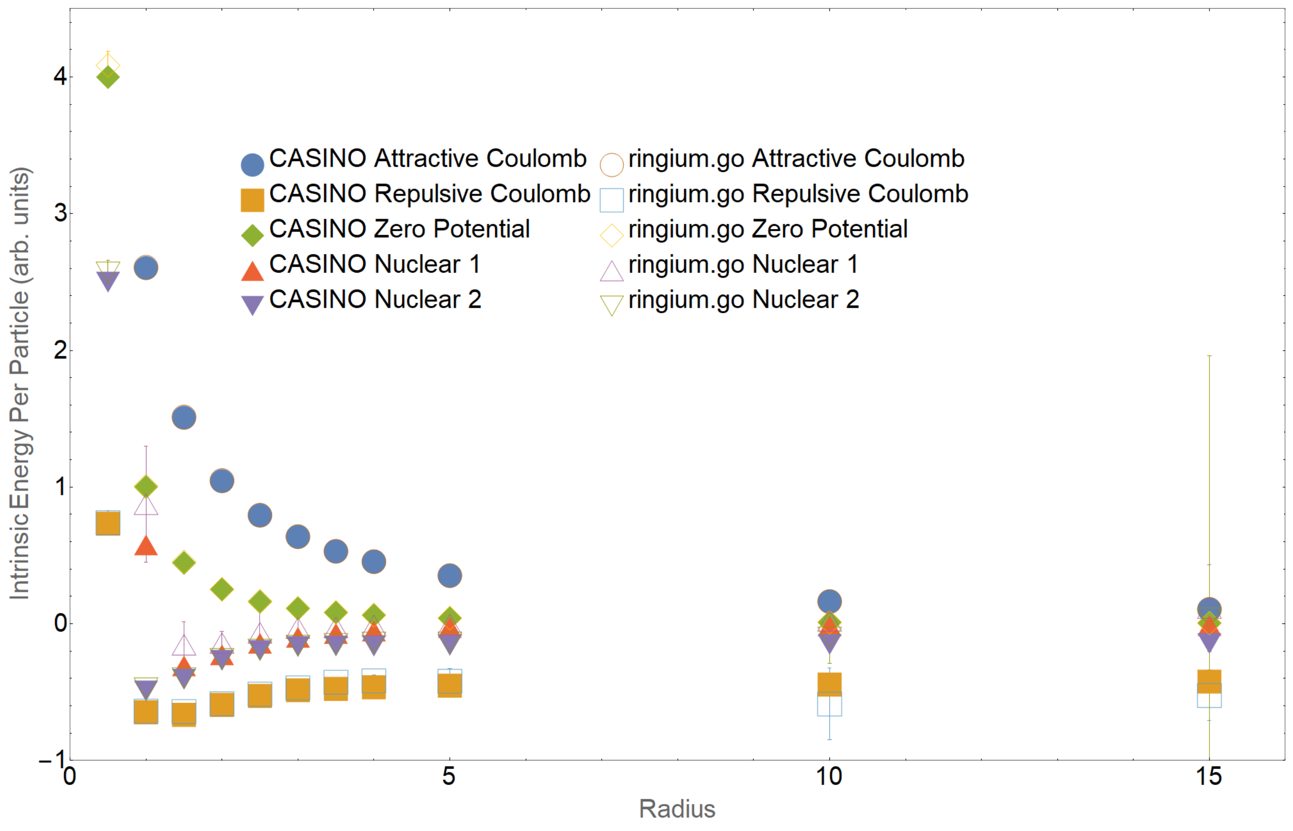

Figure 8: Comparison of intrinisic ground state energy per particle for 5 particle ringium under different potentials for varying ring radii. Here solid shapes are known numerical results, whereas the unfilled shapes are data generated using the Ringium only program.

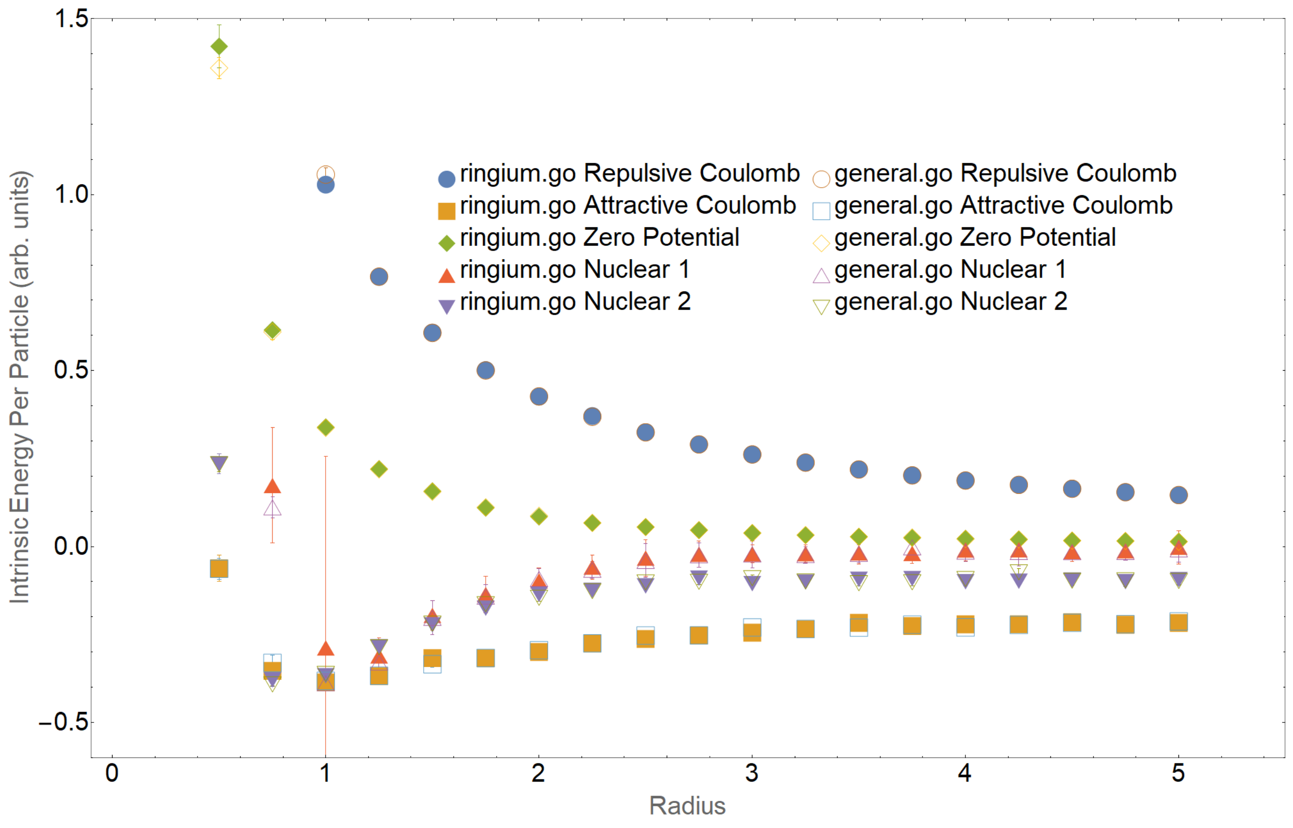

Figure 9: Results for 3 particle Ringium from the ringium only program (filled shapes) and higher dimensional program (unfilled shapes) for equivalent program settings.

Higher Dimensional Results

To generate three particle spherium, general.go must be used instead of ringium.go due to ringium.go only working for rings. To get the intrinsic energy of our three particle system, we pick states in our Slater determinant such that there is no motion in the center of mass and thus no extrinsic component to the energy. To do this, we make our Slater determinant from 3D spherical harmonics $\gamma_{m,l}$, with (m,l) values of $(m=0,l=0),(m=-1,l=1)$ and $(m=1,l=1)$. That way we have three particles represented by three wave functions. The $(m=0,l=0)$ is uniformly distributed and so has no center of mass motion, while the $(m=-1,l=1)$ and $(m=1,l=1)$ wave functions represent counter rotating particles. Thus, we have no center of mass motion for the system, and any energy we attain is the intrinsic energy. Pictured in figure 10 and 9, we see that the ground state of three particle spherium (figure 10) for repulsive Coulomb and zero potentials follow a similar trend to 3 particle ringium (figure 9), decreasing as radius increases. However, the attractive Coulomb potential is not similar. For ringium, figure 9 shows the energy first decreases in radius until reaching a minimum at a radius of 1 before gradually increasing and flattering off at higher radii. For spherium, shown in figure 10 the energy similarly first decreases in radius but does not have a noticeable minima nor gradual increase in energy, instead becoming constant in energy for larger radii.

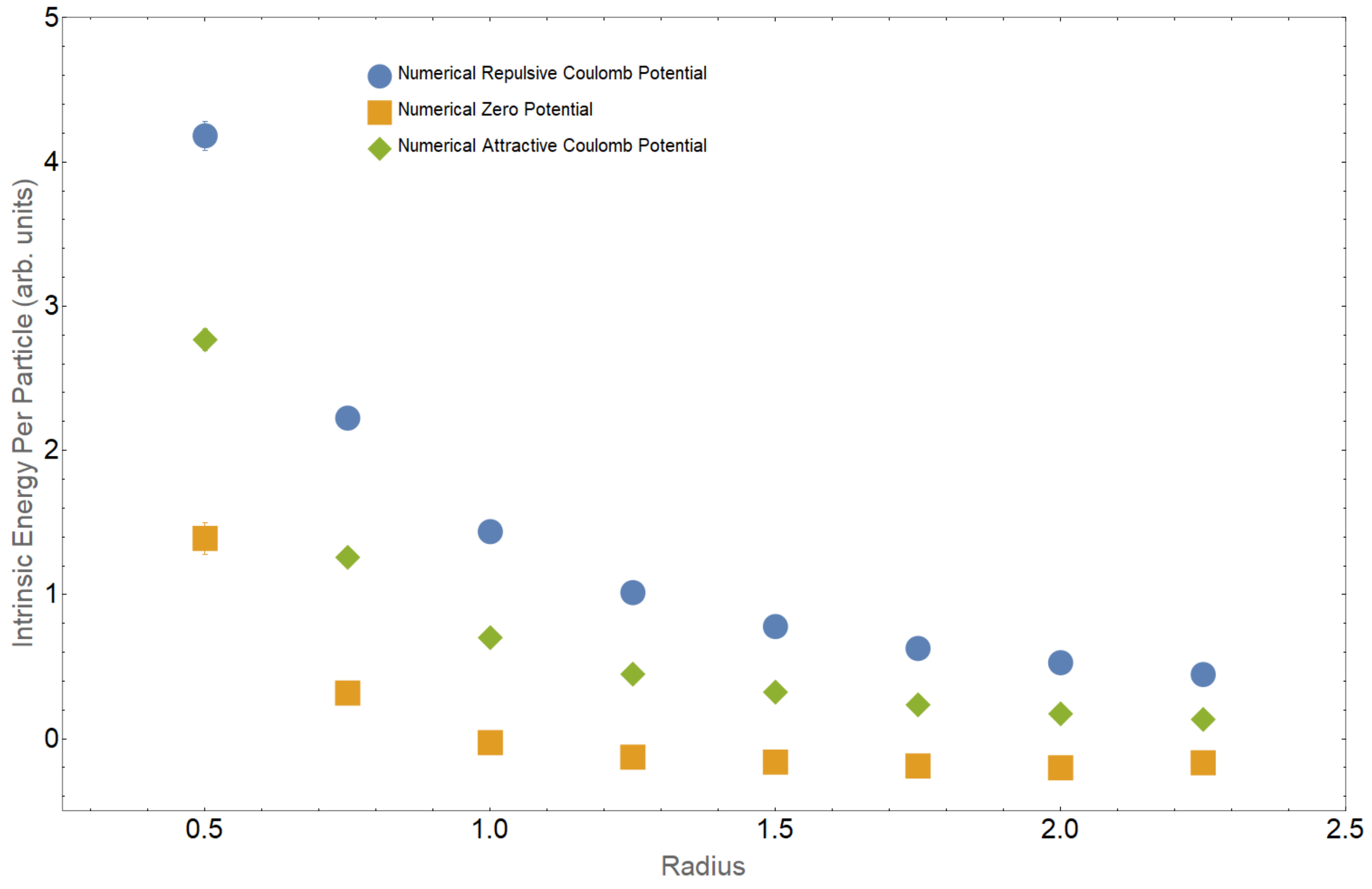

Figure 10: Numerical intrinsic ground state energies for three particle spherium for varying radius and three different interaction potentials.

The three particle glomuim results were generated by general.go. Similar to spherium, we pick the Slater determinant such that there is no motion in the center of mass. We make our Slater determinant out of single particles states $\gamma_{m,n,l}$, with (m,n,l) of values $(m=0,n=0,l=0)$, $(m=1,n=1,l=1)$ and $(m=-1,n=1,l=1)$. In doing so, the Slater determinant is composed of a wave function representing a uniformly distributed particle and two counter propagating wave functions. The center of mass of the system therefore is stationary making the energy purely intrinsic. In figure 10 we again see a general decrease in ground state energy similar to spherium. However, the uncertainty associated with glomium in figure 11 is more than an order of magnitude larger than the associated spherium uncertainty in figure 10. This could indicate two things:

1. That glomuim ground states are considerably more difficult to optimize for or,
2. Some of the glomuim based code is inaccurate

In future, with sufficient compute resources, 1 could be tested by increasing the genetic population size (*GAPopSize*) or the accuracy of the integrals (*numberOfPoints*) and observing the uncertainty. This will require considerable computing resources or efficiency improvements to the program, as a modern CPU with 12 logical cores at 3 GHz took over 24 hours to compute figure 11.

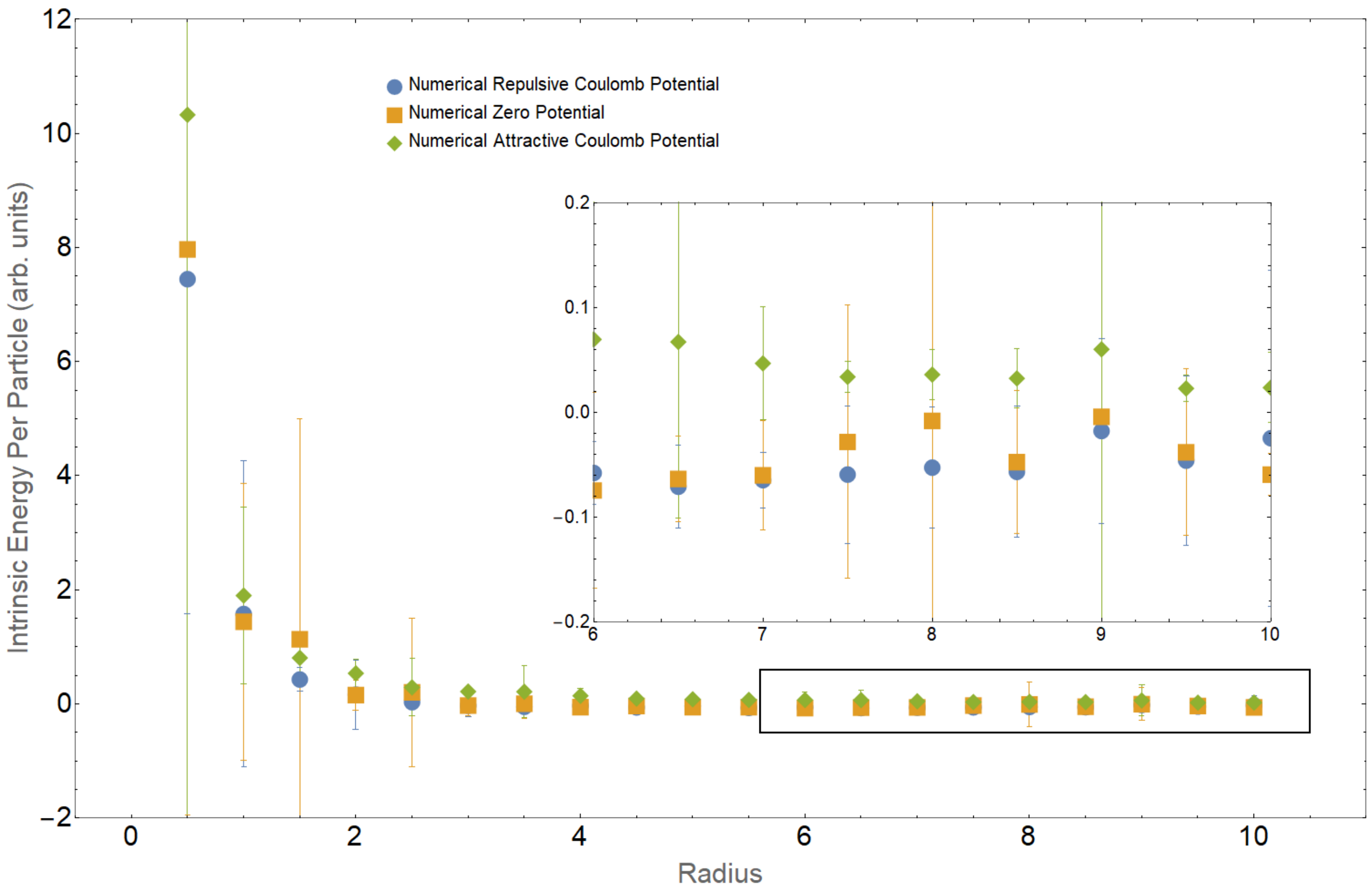

Figure 11: Numerical ground state energies for three particle glomuim for varying radius and three different interaction potentials.

Future Improvements and Extensions

There are three core improvements that could be made to the program. The first is a user interface. Currently to edit the *setupState* in the program, the user must open the source code and change its parameters directly. Functionally, the ability to correct for center of mass in general.go is most important, as this would allow for computation of intrinsic energy for arbitrary numbers of particles without the user manually inputing the states for the Slater determinant. There are two approaches for this, the first is to figure out how to compute the center of mass Hamiltonian for particles on a (n-1) sphere in cartesian coordinates. This involves terms like $\hat{p}_i \cdot \hat{p}_j$ used in the computation of $\hat{H}_{ext}$ in equation 4, where $\hat{p}_k$ is the momentum operator on the kth particle constrained to a (n-1) sphere. Currently, general.go's implementation of this does not produce results consistent with analytic results. Alternatively, general.go could be converted to hyper-spherical coordinates to compute $\hat{p}_i \cdot \hat{p}_j$. In doing so, the computation of $\hat{p}_i \cdot \hat{p}_j$ should become much easier, as the constraint that each particle must lie on the (n-1) sphere is naturally encoded by suppressing any radial derivatives. The form of the optimized wave function could also be output by ringium.go and general.go with minimal changes. This would allow for features of the wave function to be studied and uncover behaviour such as the particles clumping together as is suggested in [4]. Lastly, the efficiency of the program could be greatly improved through management of Slater determinants. Currently, the program constructs and evaluates the Slater determinant every time that the wave function is sampled. In theory, this construction only needs to happen once and so would dramatically speed up the program as it involves expensive matrix methods. The genetic algorithm could also be replaced with the more traditional direct energy minimisation or variance minimisation methods used by the CASINO code.

Conclusion

A program was written to numerically find the ground state energy for ringium, spherium and glomuim for varying radii and interaction potential. The program used a Varitional Monte Carlo (VMC) algorithm together with a genetic optimization algorithm to find these results. With adequate program settings, the intrinsic ground state energy of like-spin fermionic ringium with various potentials, particle numbers and ringium radii was found to match analytic [17] and numerical results [4]. The settings of the program were then varied, showing the effects on program convergence and accuracy for ringium. Finally, the intrinsic ground state energy for spherium and glomuim for varying radii and interaction potential were presented and compared with each other. Results from this program can be applied to fields in which these quantum ring, sphere and hypersphere models are studied, predominantly quantum chemistry as models for uniform electron gasses [16] and nuclear physics to test nuclear many-body approximations [4].

Acknowledgements

I would like to thank Professor Cedric Simenel for suggesting and supervising this project.

Appendix

The programs general.go and ringium.go are collectively ≈ 4000 lines long and so are provided by link at: https://1drv.ms/u/s!AgFTkFmsRVTpk9tP65hghrJlHKCodQ?e=WZRiaZ

References

[1] Monte Carlo Integration Wikipedia, Dec 2020.

[2] N-sphere wikipedia, Jan 2021.

[3] Kurt Binder, David M Ceperley, J-P Hansen, MH Kalos, DP Landau, D Levesque, H Mueller-Krumbhaar, D Stauffer, and J-J Weis. *Monte Carlo methods in statistical physics*, volume 7. Springer Science & Business Media, 2012.

[4] Alexander W Bray and Cédric Simenel. Fermions with long and finite-range interactions on a quantum ring. *Physical Review C*, 103(1):014302, 2021.

[5] Siddhartha Chib and Edward Greenberg. Understanding the metropolis-hastings algorithm. *The american statistician*, 49(4):327–335, 1995.

[6] ND Drummond, MD Towler, and RJ Needs. Jastrow correlation factor for atoms, molecules, and solids. *Physical Review B*, 70(23):235119, 2004.

[7] Christopher Frye and Costas J Efthimiou. Spherical harmonics in p dimensions. *arXiv preprint arXiv:1205.3548*, 2012.

[8] Andrew Gelman, John B Carlin, Hal S Stern, David B Dunson, Aki Vehtari, and Donald B Rubin. *Bayesian data analysis*. CRC press, 2013.

[9] SS Gylfadottir, A Harju, T Jouttenus, and C Webb. Interacting electrons on a quantum ring: exact and variational approach. *New Journal of Physics*, 8(9):211, 2006.

[10] Robert Jastrow. Many-body problem with strong forces. *Physical Review*, 98(5):1479, 1955.

[11] Tosio Kato. On the eigenfunctions of many-particle systems in quantum mechanics. *Communications on Pure and Applied Mathematics*, 10(2):151–177, 1957.

[12] Tony Kim. An iterative technique for solving the n-electron hamiltonian: The Hartree-Fock method, 2021.

[13] Annu Lambora, Kunal Gupta, and Kriti Chopra. Genetic algorithm-a literature review. In *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*, pages 380–384. IEEE, 2019.

[14] Pierre-François Loos. Generalized local-density approximation and one-dimensional finite uniform electron gases. *Physical Review A*, 89(5):052523, 2014.

[15] Pierre-François Loos and Peter MW Gill. Exact wave functions of two-electron quantum rings. *Physical review letters*, 108(8):083002, 2012.

[16] Pierre-François Loos and Peter MW Gill. Uniform electron gases. i. electrons on a ring. *The Journal of chemical physics*, 138(16):164124, 2013.

[17] Pierre-Fran çois Loos and Peter M. W. Gill. Two electrons on a hypersphere: A quasiexactly solvable model. *Phys. Rev. Lett.*, 103:123008, Sep 2009.

[18] George Marsaglia et al. Choosing a point from the surface of a sphere. *The Annals of Mathematical Statistics*, 43(2):645–646, 1972.

[19] Albert Messiah. Quantum mechanics. vol. I. *North-Holland Publishing Co., Amsterdam*, 5(26):20, 1961.

[20] Richarad J Needs, Michael D Towler, Neil D Drummond, and P López Ríos. Continuum variational and diffusion quantum Monte Carlo calculations. *Journal of Physics: Condensed Matter*, 22(2):023201, 2009.

[21] Richard Needs, Mike Towler, Neil Drummond, and Paul Kent. User's guide version 1.7. 7. 2005.

[22] Andrew N Sloss and Steven Gustafson. 2019 evolutionary algorithms review. *arXiv preprint arXiv:1906.08870*, 2019.

[23] Gilbert Strang, Edwin Herman, and Paul Seeburger. 2.5: Numerical integration - midpoint, trapezoid, simpson's rule, 2021.

[24] Tianchen Zhao, Giuseppe Carleo, James Stokes, and Shravan Veerapaneni. Natural evolution strategies and variational Monte Carlo. *Machine Learning: Science and Technology*, 2(2):02LT01, 2020.

[25] Elzbieta Zipper, Marcin Kurpas, Janusz Sadowski, and Maciej M Maska. Semiconductor quantum ring as a solid-state spin qubit. *arXiv preprint arXiv:1011.2540*, 2010.