Getting to know the Beaver: QMcBeaver for Novices and Experts Alike

Chip Kent, Mike Feldmann, Rick Muller, Amos Anderson, and Dan Fisher

May 8, 2006

1 Introduction: What is QMcBeaver?

QMcBeaver is a finite, all electron variational and diffusion quantum Monte Carlo program. It was written by Chip Kent and Mike Feldmann at Bill Goddard's Materials and Process Simulation Center at the California Institute of Technology as part of their thesis work. The program is currently being developed by Amos Anderson and Dan Fisher at Caltech. The code is currently being distributed under the Gnu General Public License.

Several features distinguish QMcBeaver from other QMC programs:

- As already mentioned, the code is distributed free of charge under the GPL, which allows anyone to use or to modify the code, and also makes the algorithms that QMcBeaver uses open for review by anyone.
- The code runs on massively parallel architectures, and has a highly efficient decorrelation algorithm to properly decorrelate the data from different processors [1].
- The code uses a Manager-Worker based parallelization scheme which allows the efficient use of heterogeneous clusters [2].
- The code is written in modern object-oriented C++, and is clean, and easily understood and modified.

This document will serve as a guide to getting started using QMcBeaver. We assume that the user has some familiarity with basic quantum chemistry methods such as Hatree-Fock (HF) or Density Functional Theory (DFT). Additionally, some familiarity with the techniques of variational and diffusion QMC will be required to appreciate this guide.

2 Introduction to Quantum Monte Carlo

There are a couple good introductions to the subject material if you are unfamiliar. *Modern Quantum Chemistry* by Szabo and Ostlund[8] is a good place to start reading about Hartree-Fock and the relevant linear algebra. As a bonus, this book is published by Dover, so it's cheap. The book *Computational Physics* by Thijssen[9] has a good chapter on QMC. If you want to play with some toy code, covering both of these methods, look for the Sourceforge project PyQuante. Chip Kent's thesis is also a good reference[4], including a discussion of the algorithms he and Mike developed. In addition, pasted here is an introduction to the code from a computational aspect.

The most important information about a molecule is its ground state energy, calculated by means of the time independent Schrödinger equation

$$\langle E \rangle = \frac{\int \Psi(\bar{\boldsymbol{r}}) \hat{H} \Psi(\bar{\boldsymbol{r}}) d\bar{\boldsymbol{r}}}{\int \Psi^2(\bar{\boldsymbol{r}}) d\bar{\boldsymbol{r}}} \tag{1}$$

where $\Psi(\bar{\boldsymbol{r}}) : \mathbb{R}^{3N} \rightarrow \mathbb{R}$ is the wave function, mapping the $3N$ Cartesian coordinates of N electrons into a probability amplitude related to the probability density in Eq. (4). (Equation (1) includes the common restriction that $\Psi(\bar{\boldsymbol{r}})$ is a real valued function.) The Hamiltonian operator $\hat{H}$ is given by

$$\hat{H} = \nabla^2 + V(\bar{\boldsymbol{r}}) \tag{2}$$

where the Laplacian is over all $3N$ electronic coordinates and calculates the kinetic energy (in the unitless Hartree measure) of the electrons in the molecule. The $V(\bar{\boldsymbol{r}})$ term represents the potential energy due to Coulomb interactions between all pairs of electrons and nuclei. The energy E is the eigen value of $\hat{H}$ operating on the eigen function $\Psi(\bar{\boldsymbol{r}})$. The ground state energy is the lowest such eigen value, and is of primary interest here.

There are many methods to calculate Eq. (1) with varying degrees of accuracy and computational complexity. The highly accurate QMC family of algorithms [7] uses Metropolis [5] integration to fine tune the result provided by a cheaper method. It uses the *local energy*

$$E_L(\bar{\boldsymbol{r}}) = \frac{\hat{H}\Psi(\bar{\boldsymbol{r}})}{\Psi(\bar{\boldsymbol{r}})} = \frac{\nabla^2 \Psi(\bar{\boldsymbol{r}})}{\Psi(\bar{\boldsymbol{r}})} + V(\bar{\boldsymbol{r}}) \tag{3}$$

which represents an evaluation of the energy for a set of electronic coordinates. In terms of the stationary probability distribution of electrons

$$\rho(\bar{\boldsymbol{r}}) = \frac{\Psi^2(\bar{\boldsymbol{r}})}{\int \Psi^2(\bar{\boldsymbol{r}}) d\bar{\boldsymbol{r}}} \tag{4}$$

we can transform Eq. (1) into the Monte Carlo integration form

$$\langle E \rangle = \int \rho(\bar{\boldsymbol{r}}) E_L(\bar{\boldsymbol{r}}) d\bar{\boldsymbol{r}} = \lim_{N_t \rightarrow \infty} \frac{1}{N_t} \sum_{t=1}^{N_t} E_L(\bar{\boldsymbol{r}}_t). \tag{5}$$

Here $\bar{\boldsymbol{r}}_t$ are a series of electronic coordinates generated with respect to $\rho(\bar{\boldsymbol{r}})$ by some importance sampling scheme [10]. Since error scales as $1/\sqrt{N_t}$ in Monte Carlo methods a rather large number of samples is required to achieve useful accuracies. Additionally, it is common to run several independent series, called *walkers*, in order to minimize the error due to serial correlation between the N_t data points.

In terms of computational complexity, the difficulty for QMC lies in the evaluation of $\nabla^2 \Psi(\bar{\boldsymbol{r}}_t)$ for each $E_L(\bar{\boldsymbol{r}}_t)$ as well as the evaluation of $\Psi(\bar{\boldsymbol{r}}_t)$ and $\nabla \Psi(\bar{\boldsymbol{r}}_t)$ which are used for importance sampling. The most common functional form for $\Psi(\bar{\boldsymbol{r}})$ has at least three nested stages of evaluation. At the first stage, we place a collection of N_{bf} basis functions centered at the nuclei in the $3D$ coordinate space. Typically a given nucleus is associated with multiple basis functions. The basis function takes as argument the local coordinates of a given electron (i) relative to the nucleus (j), $\vec{r}_{ij} = \vec{r}_i - \vec{R}_j$. The best results are achieved with the following functional form

$$\chi_j(x_{ij}, y_{ij}, z_{ij}) = x_{ij}^{k_j} y_{ij}^{l_j} z_{ij}^{m_j} \sum_{n_j} a_{n_j} e^{-b_{n_j} r_{ij}^2}. \tag{6}$$

For each basisfunction, R_j, k_j, l_j, m_j, n_j, a_{n_j} and b_{n_j} are parameters given as input to the QMC program. The $k_j, l_j, m_j \in \mathbb{N}$ parameters give the basisfunction the required symmetry, and $n_j \in \mathbb{N}^+$ helps select the quality of fit. The other parameters are all real numbers.

The second stage of evaluation takes linear combinations of basisfunctions to create *molecular orbitals*. The k^{th} orbital is given by $\phi_k(\vec{r}_i) = \sum_j \chi_j(r_{ij})c_{jk}$, where $c_{jk} \in \mathbb{R}$ are coefficients input to QMC. These orbitals represent the spread of the electron across the entire molecule.

Finally, the third stage of evaluation relevant to this study is the *Slater determinant*, chosen for its antisymmetric properties. For the N_s electrons of a given quantum spin α or β ($N = N_\alpha + N_\beta \sim 2N_\alpha$) the determinant is a function of the ϕ_k (which in turn are functions of the $\chi_j(r_{ij})$)

$$D_s(\bar{\boldsymbol{r}}_s) = |M_s(\bar{\boldsymbol{r}}_s)| = \begin{vmatrix} \phi_1(\vec{r}_1) & \phi_2(\vec{r}_1) & \cdots & \phi_N(\vec{r}_1) \\ \phi_1(\vec{r}_2) & \phi_2(\vec{r}_2) & & \\ \vdots & & \ddots & \\ \phi_1(\vec{r}_{N_s}) & & & \phi_{N_s}(\vec{r}_{N_s}) \end{vmatrix} \tag{7}$$

(here we partition $\bar{\boldsymbol{r}}$ into $\bar{\boldsymbol{r}}_\alpha$ and $\bar{\boldsymbol{r}}_\beta$) and the wavefunction is

$$\Psi(\bar{\boldsymbol{r}}) = D_\alpha(\bar{\boldsymbol{r}}_\alpha)D_\beta(\bar{\boldsymbol{r}}_\beta).$$

To calculate the kinetic energy, we first obtain $\nabla_i^2\phi_k(\vec{r}_i) = \sum_j \nabla_i^2\chi_j(\vec{r}_{ij})c_{jk}$, and then sum the contributions from all the electrons in all the orbitals

$$\frac{\nabla^2\Psi(\bar{\boldsymbol{r}})}{\Psi(\bar{\boldsymbol{r}})} = \sum_{s\in\{\alpha,\beta\}} \sum_{i,k\in N_s} \left[M_s^{-1}(\bar{\boldsymbol{r}}_s)\right]_{ki} \nabla_i^2\phi_k(\vec{r}_i). \tag{8}$$

A similar procedure is followed for calculating the gradient of the wavefunction for each electron with the exception that the final summation results in a vector of gradients.

To summarize the algorithm, we are given a set of nuclear coordinates, basis function parameters (available in the literature), and the c_{jk}, which describe the wavefunction as fit by some other (more approximate and cheaper) method. Additionally, we choose some parameters including the number of steps N_t, the number of walkers W, an initial guess scheme for positions $\bar{\boldsymbol{r}}$ of all the electrons, as well as several parameters relating to the importance sampling. With these in hand, the algorithm can be stated as shown in Algorithm 1 (the $\otimes$ represents matrix multiplication). The high degree of parallelism is evident since each processor can calculate all the linear algebra for its walkers and only needs to produce a single value; the energy. While some QMC algorithms only update one electron per Monte Carlo step, our method updates all at once [10].

One big advantage of QMC relative to alternative methods is the freedom one has in choosing the functional form of $\Psi(\bar{\boldsymbol{r}})$. We exploit this by multiplying the Slater determinant wave function with a set of pairwise interaction terms which explicitly model some of the underlying physics. The only condition is that these terms, called *Jastrow* functions, preserve the antisymmetry of the wave function. To satisfy this condition, we use the functional form

$$J(\bar{\boldsymbol{r}}) = \prod_{q<p} e^{u_{pq}(r_{pq})} \tag{9}$$

which provides a term for each particle-particle interaction, where

$$u_{pq}(r_{pq}) = \frac{\sum_{\kappa=1}^{\Gamma} a_{pq\kappa} r_{pq}^\kappa}{1 + \sum_{\kappa=1}^{\Lambda} b_{pq\kappa} r_{pq}^\kappa} \tag{10}$$

Algorithm 1 The QMC algorithm

$E_{sum} \leftarrow 0$
for $w = 1$ to W **do**
  $\vec{r}_{ij} \leftarrow$ initialize()
  for $t = 1$ to N_t **do**
    for $s = \alpha$ and $s = \beta$ **do**
      $M_s \leftarrow \chi_j(\vec{r}_{ij}) \otimes c_{jk}$
      $X_s \leftarrow \frac{\partial}{\partial x_i}\chi_j(\vec{r}_{ij}) \otimes c_{jk}$
      $Y_s \leftarrow \frac{\partial}{\partial y_i}\chi_j(\vec{r}_{ij}) \otimes c_{jk}$
      $Z_s \leftarrow \frac{\partial}{\partial z_i}\chi_j(\vec{r}_{ij}) \otimes c_{jk}$
      $L_s \leftarrow \nabla_i^2\chi_j(\vec{r}_{ij}) \otimes c_{jk}$
    end for
    Jastrow $\leftarrow J(\bar{\boldsymbol{r}})$
    $\Psi \leftarrow \det M_\alpha * \det M_\beta *$ Jastrow
    $E_{sum} \leftarrow E_{sum} + E_L(M_s, \text{Jastrow}, \{derivatives\}...)$
    $\vec{r}_{ij} \leftarrow \text{sampling}(\Psi, \vec{r}_{ij}, X_s, Y_s, Z_s, L_s)$
  end for
end for
$E_{avg} \leftarrow E_{sum}/(N_t * W)$

and p and q index all electrons and nuclei, and r_{pq} is the distance separating the two particles. The number of terms (Γ and Λ) is arbitrary, and depends on the quality of fit. These parameters, along with $a_{pq\kappa}, b_{pq\kappa} \in \mathbb{R}$, are input to the QMC algorithm. With this modification, our wave function is now $\Psi_{QMC}(\bar{\boldsymbol{r}}) = D_\alpha(\bar{\boldsymbol{r}}_\alpha)D_\beta(\bar{\boldsymbol{r}}_\beta)J(\bar{\boldsymbol{r}})$, and there are chain rule effects for the gradient and Laplacian. The rationale for these additional terms is the improved convergence if the wave function is a better approximation of the eigen function of $\hat{H}$ to begin with.

3 Compiling QMcBeaver

If for some strange reason you choose to compile this using a program like Microsoft Visual Studio, then this section is irrelevant. The included Visual Studio project file in the `windows` directory should set up the program appropriately, although some effort may be required to change some of the options for debugging or ATLAS. Visual Studio by default uses the `cl.exe` compiler produced by Microsoft itself. We have not tried the `icc` compiler from Intel. A parallel version of QMcBeaver for Windows has not been explored as there are no free MPI (message passing interface) packages available to try this out on. Finally, with respect to the Windows platform, installing cygwin is a good option to try. Cygwin is free – it installs a "linux aware" environment running within Windows.

There are 3 important files in the process of compiling QMcbeaver; two are given, one is created. The python script `configure.py` will create the `Makefile.config` file, which is then used in conjunction with the `Makefile` script to compile the executable. There are a couple macros used in the code, and these will be discussed.

3.1 configure.py

There are a couple important options with this script. If you are in the QMcBeaver directory, typing `./configure.py` will list the options to your screen.

1. You must specify a compiler to use. Most machines have `gcc` installed, but some have vendor provided compilers. Our `configure.py` knows about many of them, if you are attempting to compile on a machine that we have not tried, then by comparing with our other compiler configurations, you should be able to guess a good set.

2. QMcBeaver requires parallel processing to fully take advantage of its algorithm. The parallel portions of QMcBeaver has been programmed using the MPI. This code is usually run and tested on LAM/MPI, but it should work on MPICH as well without any problem. You must select `MPICC` as your compiler. In addition, there might be some steps necessary to ensure that `mpicc` and relevant libraries are on your path.

3. There are a couple variations that can change QMcBeaver's behavior. None of them are required.

   - `--`atlas: The ATLAS[13, 12] library is a freely available Basic Linear Algebra Subroutines (BLAS) package available at Sourceforge. The advantage ATLAS provides over vender BLAS libraries is that it automatically tunes itself to each processor during compilation, and is thus cross-platform compatible. Essentially, this library can be used to handle the matrix-matrix multiplication used in QMcBeaver. This makes a substantial difference for larger molecules (larger basis sets or many orbitals). For smaller molecules and atoms, ATLAS will not provide much of an advantage and could possibly slow the execution time since the cost of calling the library would come into effect. We have not successfully compiled this library everywhere. On the other hand, there are precompiled libraries for many systems available online. While ATLAS should work everywhere, sometimes it will conflict with a debugger. Instructions for installing ATLAS are included in the `lib` directory. Using ATLAS with Visual Studio requires a few special instructions, like requiring changing the library names from `libatlas.a` to `atlas.lib` etc. Google has more information.
   - `--`debug: This option is probably only useful for developers. Including this option will place debug symbols in your code, useful when QMcBeaver is run with a program like `gdb` or whatever has been installed on your system. Note, this option sometimes conflicts with optimization flags. Specifying both may produce unpredictable results. For example, the compiler may choose between the optimize and debug options.
   - `--`optimize: This option should select the best set of compiler settings for the given architecture and compiler type. However, as this is largely a black art, hand-tweaking the options in `configure.py` may improve the executable. Since it is turned on by default for most compilers, you do not need to include this flag.
   - `--`profile: This option is probably only useful for developers. It will produce an extra output file when the executable is run. This output is useful with a program like `gprof` for analyzing QMcBeaver's calling tree for possible optimizations.
   - `--`float: The key parts of the code can be switched to running with single precision variables as opposed to the typical double precision variables. Single precision will hurt the accuracy of the result, but the code will run faster on many systems.

- --vtune: To profile the code using Intel vTune, some special parameters must be used to obtain "position independent code".
- --gpu: This is used to turn on the graphics card portions of the code. It is similar to using ATLAS.
- *tag*: You can tag your executables with the tag *tag*. The purpose is that if you are playing with different options, you can recompile all code for a given *tag* with the command `make updateall tag`.

3.2 Makefile.config

This file is generated by configure.py, and for most non-developers, this file can be safely ignored. This is the easiest place to start when toying with different compiler options.

3.3 Makefile

Once configure.py has been run and the Makefile.config file has been generated, simply enter the command make on the command line in the QMcBeaver directory. The make program will use the rules in the Makefile with the options in Makefile.config to create the executable, which will be placed in the bin directory. However, there are a few options given to make by the Makefile we provide that should be listed. When making QMcBeaver, make will create a separate directory for intermediate files for each type of Makefile.config created. For example, if the ATLAS option is specified through configure.py, a directory appropriately named will be created. If later configure.py is run without the ATLAS option, or with a different compiler, or with the debug option, or any difference in options, a new directory will be created for the temporary files. This is useful when the same home directory is used for multiple computers with different operating systems, like in the Goddard group. This means the compiler does not have to recreate all the temp files every time an option is changed, if that option has been compiled before.

- bruteforce: This option might help with some stubborn compilers. It ignores most of the options in Makefile.config
- --clean: This option will remove the executable and all intermediate compiler files associated with Makefile.config. That is, it will only clean one option set.
- --cleanall: This will remove the intermediate files for all the compiler options.
- --updateall *tag*: In the process of modifying code, the developer may wish to test a set of different compiler options all at once. Instead of forcing that developer to systematically recreate each Makefile.config and recompile, this option will recompile all the options that have been previously compiled with tag *tag*. This is possible because all of the compiler options were stored in the intermediate files when that option set was originally compiled.

4 Simple Atomic Calculations: Helium

We will start with a simple example: Helium atom using a 6-31G basis set. Helium is one of the few system whose exact answer is known (-2.9037 h), and so provides a good test of how well QMC is working.

4.1 Generating a Trial Wave Function

4.1.1 Using GAMESS to Generate a Trial Wave Function

Our first step is to run a GAMESS calculation for He. A sample GAMESS input deck looks like

```
 $CONTRL SCFTYP=RHF RUNTYP=ENERGY COORD=CART UNITS=ANGS $END
 $BASIS  GBASIS=N31 NGAUSS=6 $END
 $GUESS  GUESS=HUCKEL $END
 $DATA
He ground state
Dnh 2

He   2.0       0.0000     0.0000     0.0000
 $END
```

The GAMESS RHF calculation converges with an energy of -2.8552 h.

We can then run the script

```
% (QMcBeaver)/bin/gamess2qmcbeaver.py he.out
```

The variable `QMcBeaver` should be set to the root directory of the QMcBeaver program. This script will convert the GAMESS output to a `he.ckmf` input file for QMcBeaver. This file is described below.

4.1.2 Using Jaguar to Generate a Trial Wave Function

A Jaguar input file to do the same He calculation looks like

```
&gen
basis=6-31G
ip164=2
&
&zmat
He1     0.0000000000000    0.0000000000000    0.0000000000000
&
```

The flag `ip164=2` tells Jaguar to write a he.bas file with the basis set for the calculation, which is needed to create the QMcBeaver input. We can then run the script

```
% (QMcBeaver)/bin/jaguar2qmcbeaver.py he.01.in
```

For Jaguar calculations, the argument is the restart file, not the output file.

4.2 Structure of the simple he.ckmf input file

The `he.ckmf` file produced by the QMcBeaver conversion scripts has the form

```
&flags
atoms
 1
charge
 0
```

```
energy
 -2.8551604262
norbitals
 2
nbasisfunc
 2
run_type
 variational
chip_and_mike_are_cool
 Yea_Baby!
&
&geometry
HE      2       0.000000        0.000000        0.000000
&
&basis
HE      2       3
        3       s
                38.4216340      0.040139739346
                5.7780300       0.261246097041
                1.2417740       0.79318462463
        1       s
                0.2979640       1.0
&
&wavefunction
1 1
        0.592081                0.513586

0 0
        -1.149818               1.186959

Alpha Occupation
1       0

Beta Occupation
1       0

CI Coeffs
1.0

&

&Jastrow

ParticleTypes: Electron_Up Electron_Down
CorrelationFunctionType: FixedCuspPade
NumberOfParameterTypes: 2
NumberOfParametersOfEachType: 0 1
Parameters: 3.0
```

```
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
Constants: 0.5

ParticleTypes: Electron_Up HE
CorrelationFunctionType: FixedCuspPade
NumberOfParameterTypes: 2
NumberOfParametersOfEachType: 0 1
Parameters: 100.0
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
Constants: -2.0

ParticleTypes: Electron_Down HE
CorrelationFunctionType: FixedCuspPade
NumberOfParameterTypes: 2
NumberOfParametersOfEachType: 0 1
Parameters: 100.0
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
Constants: -2.0

&Jastrow
```

This file is mostly self-explanatory, but we'll cover a few points that are worth emphasizing nonetheless.

The *flags* section contains general input parameters, including the number of atoms, the molecular charge, the number of orbitals and basis functions. Note that it also contains the *run_type* flag, currently set to *variational*, indicating that we will first be doing VQMC calculations. The *flags* section contains all of the parameters that the user specifies to control the calculation. The flags what they do are described in a later section.

The *geometry* section contains the geometry of the molecule. Cartesian coordinates in atomic units (Bohr) are given for each nucleus. The *basis* section defines the two basis functions we will be using, and the *wavefunction* section describes the coefficients of the trial molecular orbitals in terms of these basis functions.

The *Jastrow* section is worth describing in slightly more detail. This section details the correlation functions we will use in the QMC calculations to follow. A correlation function is defined for each pair of particles in the molecule. In the case of the He atom, we have electron-up-electron-down, electron-up-He, and electron-down-He correlation functions. If there were more than one up electron, there would be an electron-up-electron-up correlation function, and so on.

In this example, all functions are set to the *FixedCuspPade* form, and guesses as to the appropriate parameters are included. The format for the constants and paramters in this and other available correlation function forms will be demonstrated in later sections of this manual.

4.3 Running a VQMC Calculation

A variational QMC calculation allows us to statistically sample the input wave function, including the Jastrow terms. By running several of these calculations, we can optimize the parameters in the Jastrow function and get a more accurate approximation to the true energy.

We can run a simple VQMC calculation via the command

```
% $(QMcBeaver)/bin/QMcBeaver.linux he.ckmf
```

This command will run 1,000,000 steps of VQMC using the Jastrow parameters specified in the input ckmf file, and will then write several files, including a log of various quantities every 1000 steps of the simulation (he.qmc), a summary of the statistics for the run (he.rslts), and a new input file (he.01.ckmf) and a checkpoint file that can be used to restart the calculation.

This example calculation is only an evaluation of the trial wavefunction defined in the .ckmf file. The optimization parts of the code are turned off, so all of the parameters are fixed at their default values.

Several points are worth investigating in the he.rslts. We first consider the energy, which is reported to be $-2.6337228773 \pm 0.1738595133$ h. This is significantly worse than either the HF (-2.855) or exact (-2.903) energies for several reasons. First, the statistics aren't converged, as indicated by the variance estimate of ± 0.1738595133. The size of this number indicates that not even the first decimal place of the total energy is converged, and that we will have to run a significantly longer calculation to obtain higher precision.

If we simply want higher precision, we can restart the VQMC calculation using the command

```
% $(QMcBeaver)/bin/QMcBeaver.linux he.01.ckmf
```

Another statistic that is reported in the he.rslts file is the Virial Ratio. From the Virial Theorem we know that in a system of particles interracting through a Coulomb potential, the ratio of the average potential energy to the average kinetic energy, $-\langle V \rangle / \langle T \rangle$, is 2. The fact that the program reports this ratio as $2.6311988719 \pm 0.3675653079$ is another sign that the statistics are not sufficiently converged.

But the most important cause of error in this result is the fact that we have not optimized the Jastrow parameters that go into the calculation, we simply used the guess values that the conversion program gave us. Optimizing the parameters in the wavefunction is covered in later sections of the manual.

4.4 Running a DQMC Calculation

Once we have sufficiently converged the Jastrow parameters via VQMC, we are ready to begin a DQMC calculation that can further optimize the energy by allowing the walkers to breed/die or reweight themselves based upon the true potential. A good rule of thumb is that the Jastrow parameters have been sufficiently converged when the variance is 50% of the variance from the HF input guess.

5 A Slightly More Complicated Example: Water

To be completed.

6 The Flags

This section lists the flags that can be specified in the .ckmf file.

6.1 The Molecule

Natoms The number of atoms in the molecule.

charge Charge of the system in atomic units (+1 if a neutral molecule loses an electron).

energy The initial trial energy in hartree au for the molecule.

6.2 The Trial Wavefunction

Norbitals Total number of orbitals (occupied and unoccupied) in the SCF part of the trial wavefunction.

Nbasisfunc Number of basis functions used to create the orbitals of the trial wavefunction.

Ndeterminants Number of Slater determinants in the SCF part of the wavefunction.

6.3 Sampling

walker_initialization_method Specifies the method by which the initial electronic configurations for the walkers are generated. The default option is `mikes_jacked_walker_initialization`, which assigns the electrons for each atom randomly to a spherical gaussian centered on the atom. The other option is `dans_walker_initialization`, which uses STRAW [3] to generate initial configurations that need very few equilibration steps.

iseed The initial seed for the random number generator. The random number generator used in the program is ran1 from Numerical Recipes [6]. This number must be a negative integer.

sampling_method This is the method by which new configurations are proposed and either accepted or rejected. The first option is `no_importance_sampling`, which means that the displacement of each electron in each direction is distributed with respect to a Gaussian whose standard deviation is the square root of the time step. The second option is `importance_sampling`, in which the electrons are first displaced according to the quantum force and then diffuse randomly. The third option is `umrigar93_importance_sampling`, which uses Umrigar's accelerated Metropolis method [10].

QF_modification_type `none` if the quantum force, $\frac{\nabla\Psi}{\Psi}$, is not to be modified. Other possibilities are `umrigar93_equalelectrons` and `umrigar93_unequalelectrons`, both of which are described in [10].

energy_modification_type `none` if the local energy is not to be modified when weighting the walkers. Other possiblities are `modified_umrigar93` and `umrigar93`, both of which are described in [10].

umrigar93_equalelectrons_parameter If Umrigar's 1993 QMC algorithm [10] is used with the equal-electron modifications to the local energy and quantum force, this is the parameter describing how the modifications will occur. This parameter can be in the range $(0, 1]$, and the default value is 0.5.

walker_reweighting_method Algorithm used to reweight the walkers after a time step. The possibilities are `simple_symmetric`, `simple_asymmetric`, and `umrigar93_probability_weighted`, all of which are described in [10].

branching_method Method for branching walkers in DMC calculations. There is no branching in VMC. Possible methods are `non_branching`, in which the weights are varied instead of branching, `unit_weight_branching`, in which the walkers have integer weights [**?**], and `nonunit_weight_branching`, which is described in [10].

branching_threshold If a walker's weight exceeds this threshold, it will branch when the branching method allows the walkers to have fractional weights. The default value is 2.0.

fusion_threshold If a walker's weight is less than this threshold, it will fuse with another low weight walker when the branching method allows the walkers to have fractional weights. The default value is 0.5.

old_walker_acceptance_parameter Parameter which can be used to prevent persistent configurations from interfering with a calculation [10].

dt This is the time step used for the propagation phase of the calculation. The default value is .001. Notes: There is sort of a balance here. If dt is large, then your error will be large. If dt is small, it will take longer to explore all the space. Officially, you're supposed to estimate an extrapolation to dt = 0. That is, run it for (for example) 1e-3, then 1e-4, then 5e-5, etc until you can guess what the value at zero would be.

population_control_parameter Parameter used in controlling the number of walkers or total weight of all walkers during a branching QMC calculation. The default value is 1.

correct_population_size_bias 1 if the correction for the population size bias for branching calculations is used, and 0 otherwise. See [10].

6.4 Equilibration

equilibration_steps The number of equilibration steps that are to be taken before statistics are gathered. This is necessary to allow the walkers to reach regions of high probability density, so that their configurations reflect the desired distribution.

dt_equilibration The time step used during the equilibration phase of the calculation. The default value is 0.02. Notes: You might want to set this higher than you would set your regular dt since the point is to get the electrons in approximately the right place, not to make measurements.

equilibration_function The function used during the equilibration phase of the calculation. The first option is `step`, in which the time step changes abruptly from *dt_equilibration* during the equilibration phase to *dt* when equilibration is finished. The second option is `ramp`, in which the time step changes linearly from *dt_equilibration* to *dt* during the equilibration phase. The third option is `CKAnnealingEquilibration1`, which is the same as `step` except that for a certain number of steps, essentially every step proposed is accepted. If importance sampling is used, this allows the walkers to reach high density regions very quickly.

CKAnnealingEquilibration1_parameter If `CKAnnealingEquilibration1` is used, this is the number of steps taken in which every proposed displacement is accepted. This number must be less than the number of equilibration steps.

equilibrate_first_opt_step 1 if the first VMC calculation used to generate correlated sampling configurations for optimization equilibrates for the specified equilibration time, and 0 otherwise. If a DMC calculation is performed, 1 if the calculation equilibrates, and 0 otherwise.

equilibrate_every_opt_step 1 if every VMC calculation used to generate correlated sampling configurations for optimization equilibrates, and 0 otherwise.

use_equilibration_array 1 if the equilibration array is to be used and 0 otherwise. This defines an array of statistics gathering objects, in which the ith element becomes active on the 2^ith iteration. The object with the smallest standard deviation for the energy is chosen, which automatically determines the correct number of equilibration steps.

6.5 Controlling the Calculation

run_type `variational` or `diffusion` QMC.

temp_dir The temporary directory where correlated sampling configurations are to be written.

max_time_steps The maximum number of propagation steps to be carried out in the calculation. In a parallel calculation, this is the total number of propagation steps on *all* processors.

desired_convergence If the standard deviation of the energy in the global results falls below this level, sampling stops. Notes: It's difficult to determine a priori how many time steps it will take to reach a given convergence, so it's often easier to control the length of the run by setting this to 0.0 and choosing an appropriate max_time_steps.

number_of_walkers The number of walkers on each processor. At each time step, the statistics for the walkers on the processor are preblocked. That is, for each observable, the average over the walkers on the processor becomes one sample.

output_interval At this interval on the root node, a summary of the global statistics are printed out. Quantities reported include the global energy, its standard deviation, and the total number of samples collected.

mpireduce_interval The interval at which the statistics are collected by the root node. This interval must be less than or equal to the output_interval. The way in which the statistics from different processors are combined is equivalent to *appending* the time steps from each processor to form one long run. Notes: the longer this is, the longer it will take for the calculation to converge. The short this is, the more time the calculation will spend shuttling data back and forth. This balance might be molecule specific since the communication time depends on how much data to send.

mpipoll_interval The interval at which the worker nodes check to see if there are any commands from the root node. The default value is 1. Notes: if this is too large, then the root processor might end of spending a long time waiting. There isn't a significant penalty to making this small. Perhaps this is the sort of parameter we want to remain opaque to the user.

checkpoint 1 if checkpoint files for restarting the calculation are to be written, 0 otherwise. A checkpoint file contains the entire state of the calculation on one processor, and contains the rank of the processor in the file name. Walker configurations, weights, and accumulated statistics are all written to the file.

checkpoint_interval The interval at which checkpoint files are to be written. New checkpoint files overwrite old ones.

use_available_checkpoints 1 if available checkpoint files are to be used in restarting the calculation.

zero_out_checkpoint_statistics 1 if the the accumulated statistics in the checkpoint file are to be discarded, 0 if they are to be used.

print_transient_properties 1 if the accumulated statistics should be printed out during the calculation and 0 otherwise.

print_transient_properties_interval The interval at which the accumulated statistics are printed out.

write_all_energies_out 1 if the local energy for each walker at each time step should be written out and 0 otherwise.

calculate_bf_density 1 if the basis function density should be collected and written to a .density file. This amounts to calculating the expectation value for each basis function, which can be used for fitting analytical densities to QMC simulations [**?**].

print_configs 1 if the electronic configurations are to be written to a .cfgs file in the temp directory, 0 if not. Configurations are always written if the wavefunction is being optimized. Notes: turning this on is required for wavefunction optimization. However, it can *easily* take up all of your compute time since it obviously requires extensive harddrive access. If you select optimization, this will automatically be set to 1. Also, depending on print_config_frequency and molecule size, these files can easily get up to be several gigabytes in size. The files can be output in ASCII or binary – that option is set in code in QMCConfigIO. Eventually, we'll probably add HDF5 functionality, but last I (AGA) checked they were still working on their C++ API.

print_config_frequency The interval at which configurations are written to the .cfgs file in the temp directory. This number should be chosen carefully- if it is too small, the .cfgs file will become very large. If it is too large, not enough configurations will be written to meaningfully compare sets of parameters.

6.6 Optimization

optimize_Psi 1 if the parameters in the Jastrow function are to be optimized, 0 if not. Optimization of the SCF part of the wavefunction (CI coefficients, orbitals, basis functions) is not implemented at this time. Notes: This will rerun the entire VMC calculation each time it optimizes. That is, your calculation time might go up by a factor of max_optimize_Psi_steps, depending on convergence.

max_optimize_Psi_steps One step is defined as follows- A series of configurations is generated with respect to the most accurate trial function available and correlated sampling configurations are written. Then, using the configurations, several sets of parameters are evaluated and a new trial wavefunction is generated. Default value is 10. Notes: Wavefunction convergence can take a long time, depending on your Jastrow parameters.

optimize_Psi_criteria Objective function to use in comparing sets of parameters when optimizing a VMC wavefunction. Possibilities are `energy_variance`, `energy_average`, `umrigar88`, which is described in [11], and `monkey_spank`, which is a combination of the variance of the

local energy and a penalty function when the parameters become majorly different from the ones used in generating the correlated sampling configurations. Notes: for QMC, if your wavefunction is an eigenfunction, your variance will be exactly zero. That's why you might want to minimize your variance. However, this isn't necessarily the best proceedure...

numerical_derivative_surface Objective function to use when calculating derivatives with respect to the parameters for optimizing a VMC wavefunction. Possibilities are the same as those for *optimize_Psi_criteria*.

optimize_Psi_barrier_parameter Objective function parameter used in forming a barrier around a valid region of parameter space for the `monkey_spank` objective function. A valid region of parameter space is one where the wavefunction is not majorly different from the one used to generate the walkers when correlated sampling is used.

optimize_Psi_method Numerical optimization algorithm to use when optimizing a VMC wavefunction. Possibilities are `steepest_descent`, `BFGSQuasiNewton`, and `CKGeneticAlgorithm1`.

optimization_max_iterations Maximum number of iterations to use when optimizing a VMC wavefunction with one set of correlated sampling configurations. Default value is 100.

optimization_error_tolerance Tolerance used to determine when a numerical optimization with one set of correlated sampling configurations has converged. Default value is 0.001.

ch_genetic_algorithm_1_population_size Population size used when the CKGeneticAlgorithm1 is used to optimize a VMC wavefunction. Default value is 30.

ck_genetic_algorithm_1_initial_distribution_deviation Standard deviation used when initializing the CKGeneticAlgorithm1. Default value is 1.

ck_genetic_algorithm_1_mutation_rate Mutation rate used when the CKGeneticAlgorithm1 is used to optimize a VMC wavefunction. Default value is 0.2.

line_search_step_length Algorithm used to determine the step length used when a line search algorithm is used to optimize a VMC wavefunction. Possibilities are `MikesBracketing` and `Wolfe`.

singularity_penalty_function_parameter Parameter used for the logarithmic barrier when D.R. Kent IV's algorithm for optimizing possibly singular Jastrow functions is used. This parameter should be a small positive number. The default value is 1.0e-6.

6.7 The Jastrow Function

link_Jastrow_parameters 1 if the parameters for up and down electrons are to be set equal to each other, 0 if not. For example, if this flag is on, parameters for the the electron-up-electron-up and electron-down-electron-down correlation functions will be set equal to each other. This reduces the number of parameters that have to be optimized and reduces spin contamination for singlet states.

scale_Jastrow_distances 1 if distances in the Jastrow function are to be scaled by by $R = \frac{1-exp[-kr]}{k}$, where k is a positive adjustable parameter. As $r \rightarrow \infty$, $R \rightarrow \frac{1}{k}$, which makes the function converge more rapidly with powers of r.

calc_three_body_Jastrow 1 if three body correlations are to be calculated. The three body Jastrow is of the form used by Umrigar [**?**]. Not finished yet!

6.8 Misc

chip_and_mike_are_cool C'mon, throw them a bone.

7 The Jastrow Function

Correlation functions are defined for each distinct pair of particles in the molecule in the &Jastrow section of the .ckmf file. An example of a correlation function is

```
ParticleTypes: Electron_Up Electron_Down
CorrelationFunctionType: FixedCuspPade
NumberOfParameterTypes: 2
NumberOfParametersOfEachType: 0 1
Parameters: 3.0
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
Constants: 0.5
```

This is a correlation function for an up electron and a down electron. A fixed cusp Pade function is used, which means that as these two particles approach each other, the function will approach the exact quantum mechanical solution for that collision. The cusp condition for opposite spin electrons is 0.5, while the cusp condition for same spin electrons is 0.25 and the cusp condition for an electron and a nucleus is the opposite of the charge of the nucleus.

The other possibilities function types are `None`, in which the particles are not correlated and `Pade`, in which the cusp condition is not fixed.

The fixed cusp Pade correlation function is a fraction of polynomials, and the parameters and constants are entered as follows:

The number of parameter types equals 2 because there are parameters in the numerator and the denominator. There are 0 numerator parameters and 1 denominator parameter. The denominator parameter is the first degree coefficient of the polynomial, because the zeroth order coefficient is always one. The number of constant types is one because there is only one constant, the cusp condition, which is the first order term in the numerator. This function, therefore, is $U_{\uparrow\downarrow}(r) = \frac{0.5r}{1+3.0r}$. The only adjustable parameter is the 3.0 in the denominator.

More powers of r can be added to the function by changing the parameters part of the definition. For example, the following definition

```
ParticleTypes: Electron_Up Electron_Down
CorrelationFunctionType: FixedCuspPade
NumberOfParameterTypes: 2
NumberOfParametersOfEachType: 2 3
Parameters: 3.0 1.5 0.75 0.4 0.2
NumberOfConstantTypes: 1
NumberOfConstantsOfEachType: 1
Constants: 0.5
```

would define the function $U_{\uparrow\downarrow}(r) = \frac{0.5r+3.0r^2+1.5r^3}{1+0.75r+0.4r^2+0.2r^3}$.

For the Pade correlation function, the part of the definition dealing with the constants is ignored, and the function is defined entirely with the parameters.

When the wavefunction is optimized, parameters are varied and the contants are not.

8 Optimization Methods

References

[1] M.T. Feldmann, D.R. Kent IV, and W.A. Goddard III. Efficient algorithm for 'on-the-fly' error analysis of local or distributed serially-correlated data. *Journal of Computational Chemistry*, 2004. Submitted.

[2] M.T. Feldmann, D.R. Kent IV, R.P. Muller, J.C. Cummings, and W.A. Goddard III. Manager-worker-based model for the parallelization of Quantum Monte Carlo on heterogeneous and homogeneous networks. *Journal of Computational Chemistry*, 2004. Submitted.

[3] Dan Fisher. Improving the efficiency of quantum Monte Carlo calculations. I. The stratified walker initialization. 2006. In Progress.

[4] David Randall C̈hipK̈ent IV. *New Quantum Monte Carlo Algorithms to Efficiently Utilize Massively Parallel Computers*. PhD thesis, Caltech, 2003.

[5] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087, 1953.

[6] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, 1993.

[7] P. J. Reynolds, D. M. Ceperley, B. J. Alder, and W. A. Lester. Fixed-node Quantum Monte Carlo for molecules. *Journal of Chemical Physics*, 77(11):5593–5603, 1982.

[8] Atilla Szabo and Neil S. Ostlund. *Modern Quantum Chemistry*. McGraw-Hill, New York, 1982.

[9] J. M. Thijssen. *Computational Physics*. Cambridge, New York, 1999.

[10] C. J. Umrigar, M. P. Nightingale, and K. J. Runge. A diffusion Monte Carlo algorithm with very small time-step errors. *Journal of Chemical Physics*, 99(4):2865–2890, 1993.

[11] C. J. Umrigar, K. G. Wilson, and J. W. Wilkins. Optimized trial wave-functions for Quantum Monte Carlo calculations. *Physical Review Letters*, 60(17):1719–1722, 1988.

[12] R. Clint Whaley and Jack J. Dongarra. *Automatically Tuned Linear Algebra Software*. 1999. CD-ROM Proceedings.

[13] R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra. Automated empirical optimization of software and the ATLAS project. *Parallel Computing*, 27(1–2):3–35, 2001.