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

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

May 10, 2005

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, with consultation by Mike and Chip. 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.

There are a couple good introductions to the subject material if you are unfamiliar. *Modern Quantum Chemistry* by Szabo and Ostlund[?] 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[?] 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.

2 Compiling QMcBeaver

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.

If 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 a compiler produced by Microsoft itself, and is probably not a good option – it produces really slow code. There is an alternative compiler for Windows available from Intel, but it costs money and has not been tried with QMcBeaver yet (a free version of this compiler for Linux is also available). 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.

2.1 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 platform on which to compile QMcBeaver. For example, if you are compiling in OSX, Linux, cygwin, or any Unix type system with the GNU compiler collection installed, you could try this option. For your information, many IBM supercomputers run AIX and many SGI machines run IRIX. However, the manufacturer might have a better compiler available for their architecture. To determine which operating system your computer is running, look at the `HOSTTYPE, OSTYPE,` or `VENDOR` environment variables.

2. QMcBeaver requires parallel processing to fully take advantage of its algorithm. The parallel portions of QMcBeaver has been programmed using the MPI (as opposed to OpenMP) library. This library comes in two common flavors, `LAMPI` and `MPICH`, but some manufacturers release their own versions. HOW DO YOU KNOW WHICH TO CHOOSE? You do not need to specify 'None' to run a serial version.

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

    - `--atlas`: The ATLAS library is a freely available Basic Linear Algebra Subroutines (BLAS) package 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. 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 the libraries to be named in a fashion Visual Studio understands. Google knows how to do this.
    - `--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 usually 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.
- `--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.
- `--single`: 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.

2.2 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. This file is specific to the options used with `configure.py` so it must be regenerated every time the operating system is changed.

2.3 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. That is, 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`: 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. This is possible because all of the compiler options were stored in the intermediate files when that option set was originally compiled.

3 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.

3.1 Generating a Trial Wave Function

3.1.1 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.

3.1.2 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.

3.2 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.

3.3 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.

3.4 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.

4 A Slightly More Complicated Example: Water

To be completed.

5 The Flags

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

5.1 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.

5.2 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.

5.3 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[?] 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.[3] 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.[4]

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.[4]

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.[4]

umrigar93_equalelectrons_parameter If Umrigar's 1993 QMC algorithm[4] 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.[4]

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.[4]

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.[4]

dt This is the time step used for the propagation phase of the calculation. The default value is .001.

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.[4]

5.4 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.

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 $2^i th$ iteration. The object with the smallest standard deviation for the energy is chosen, which automatically determines the correct number of equilibration steps.

5.5 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.

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.

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.

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.

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.

5.6 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.

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.

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,[5] 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.

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.

5.7 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.[?]

5.8 Misc

chip_and_mike_are_cool C'mon, throw them a bone.

6 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.

7 Optimization Methods

References

1. Feldmann, M.; Kent IV, D.; Goddard III, W. *Journal of Computational Chemistry* **2004,** Submitted.

2. Feldmann, M.; Kent IV, D.; Muller, R.; Cummings, J.; Goddard III, W. *Journal of Computational Chemistry* **2004,** Submitted.

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

4. Umrigar, C.; Nightingale, M.; Runge, K. *Journal of Chemical Physics* **1993,** *99,* 2865–2890.

5. Umrigar, C.; Wilson, K.; Wilkins, J. *Physical Review Letters* **1988,** *60,* 1719–1722.