FCIQMC Algorithm for the UEG

1

Generated by Doxygen 1.8.6

Tue Mar 7 2017 18:16:11

Contents

Chapter 1

FCIQMC HEAD PAGE

1.1 Introduction

—$>$ INCLUDED IN HEADER $<$ GLOBAL_ORBITAL.H $>$ is " const int ORB_SIZE = 125 " $<$— —$>$ NB - - NUMBER OF ORBITALS MUST BE KNOWN A PRIORI FOR "binaryManip.C$*$ AND for KEsortedKpoints[size][3]

Chapter 2

FCIQMC_UEG

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

File Documentation

4.1 binaryManip.C File Reference

```
#include "binaryManip.H"
#include "GLOBAL_ORBITAL.H"
```

Functions

- void decimalToBinary (long int decimal, std::string &binaryNum)
- void binaryToDecimal (std::string binaryString, long int &decimal)
- size_t oneBitCount (size_t number)
- double binomialnCr (double n, double r)
- int getPositionInList (std::pair< long int, long int > &uniqueDet, std::vector< long int > &alphaDETLIST, std::vector< long int > betaDETLIST)

4.1.1 Function Documentation

4.1.1.1 void binaryToDecimal (std::string *binaryString*, long int & *decimal*)

4.1.1.2 double binomialnCr (double *n*, double *r*)

4.1.1.3 void decimalToBinary (long int *decimal*, std::string & *binaryNum*)

4.1.1.4 int getPositionInList (std::pair< long int, long int > & *uniqueDet*, std::vector< long int > & *alphaDETLIST*, std::vector< long int > *betaDETLIST*)

4.1.1.5 size_t oneBitCount (size_t *number*)

4.2 planeWaves.C File Reference

```
#include "planeWaves.H"
```

Functions

- void createPlaneWaveOrbitals (double kPoints[ ][3], const double KECutoff, int &numSpinOrbitals)
- void kPointsEnergySort (double kPoints[ ][3], double orderedkPoints[ ][3], const int kPointLength)

- void PRINTORBITALS (const int klistLength, double klist[ ][3])
- const double getCellLength (const double cellVolume)

4.2.1 Function Documentation

4.2.1.1 void createPlaneWaveOrbitals (double *kPoints[ ][3],* const double *KECutoff,* int & *numSpinOrbitals*)

4.2.1.2 const double getCellLength (const double *cellVolume*)

4.2.1.3 void kPointsEnergySort (double *kPoints[ ][3],* double *orderedkPoints[ ][3],* const int *kPointLength*)

4.2.1.4 void PRINTORBITALS (const int *klistLength,* double *klist[ ][3]*)

4.3 README.md File Reference

4.4 UEG_MAIN_binarytest.C File Reference

```
#include "binaryManip.H"
#include "planeWaves.H"
#include "UEGHamiltonian.H"
#include <fstream>
#include <time.h>
#include <set>
#include <map>
```

Functions

- constexpr std::uint64_t INLpow2 (std::uint64_t x)
- const int INLgetPositionInList (std::pair< long int, long int > &uniqueDet, std::vector< long int > &alphaDetList, std::vector< long int > &betaDetList)
- void SPAWN (const double cellLength, std::vector< int > &trueWalkerList, std::vector< int > &posWalkerList, std::vector< int > &negWalkerList, double(&KEsortedList)[ORB_SIZE][3], std::set< std::pair< long int, long int > > &uniqueDetSet, std::pair< std::set< std::pair< long int, long int > >::iterator, bool > &result, std::vector< long int > &alphaDets, std::vector< long int > &betaDets)
- void DEATH_CLONE (const double cellLength, std::vector< int > &trueWalkerList, double(&KEsortedList)[ORB_SIZE][3], double &SHIFT, std::vector< long int > &alphaDets, std::vector< long int > &betaDets, const double &HFEnergy)
- void ANNIHILATION (int &step, std::vector< int > &trueWalkerList, std::vector< int > &posWalkerList, std::vector< int > &negWalkerList, std::set< std::pair< long int, long int > > &uniqueDeterminantSet, std::vector< long int > &alphaDetsBinary, std::vector< long int > &betaDetsBinary)
- int totalWalkerNumber (std::vector< int > &trueWalkerList)
- double variableShift (const double &delt, const int &AShift, int &intTimestep, const double &zeta, std::vector< double > &totalWalkerTracker, std::vector< double > &shiftTracker)
- double projectorEnergy ()
- int main (void)

Variables

- const double PI = 3.141592653589793
- const double rs = 1.0
- const double numElectrons = 14
- const int INTelectrons = numElectrons

- const double Kc_CUTTOFF = 2
- const double delt = 0.003
- const double zeta = 0.04
- const int AShift = 2
- const int numSteps = 500000
- const int walkerCritical = 500000
- int initRefWalkers = 50
- long int pow2Array [ORB_SIZE]

4.4.1 Function Documentation

4.4.1.1 void ANNIHILATION (int & *step,* std::vector< int > & *trueWalkerList,* std::vector< int > & *posWalkerList,* std::vector< int > & *negWalkerList,* std::set< std::pair< long int, long int > > & *uniqueDeterminantSet,* std::vector< long int > & *alphaDetsBinary,* std::vector< long int > & *betaDetsBinary*)

This represents the final (third) step in the population dynamics algorithm. The following description is verbatim from the original paper:

The annihilation step: In this (final) part of the algorithm, we run over all (newly-spawned, cloned and surviving parent) walkers, and annihilate pairs of walkers of opposite sign which are found to be on the same determinant. Each time an annihilation event occurs, the corresponding pair is removed from the list of walkers, and the total number of walkers N_w reduced by two.

4.4.1.2 void DEATH_CLONE (const double *cellLength,* std::vector< int > & *trueWalkerList,* double(&) *KEsortedList[ORB_SIZE][3],* double & *SHIFT,* std::vector< long int > & *alphaDets,* std::vector< long int > & *betaDets,* const double & *HFEnergy*)

This routine represents the second step in the population dynamics algorithm. The following description is verbatim from the original paper:

The diagonal death/cloning step: for each (parent) walker compute

$$p_d(i_\alpha) = \delta\tau(K_{i_\alpha i_\alpha} - S)(16)$$

If $p_d > 0$, the walker dies with probability p_d, and if $p_d < 0$ the walker is cloned with probability $|pd|$. The death event happens immediately, and such a parent does not participate in the following (annihilation) step to be described shortly. Cloning events are quite rare, and only occur for $S > 0$, and even then only on determinants for which $\langle D_i|K|D_i\rangle < S$. In simulations where we desire to grow the number of walkers rapidly, a positive value of S is adopted, and this can lead to cloning events. However, more often, the value of S is negative (as it tries to match the correlation energy), and in such cases there can be no cloning events at all. Generates a number between (0,1]

4.4.1.3 const int INLgetPositionInList (std::pair< long int, long int > & *uniqueDet,* std::vector< long int > & *alphaDetList,* std::vector< long int > & *betaDetList*) [inline]

This function purely resturns the position of a unique determinant within the list of Alpha and Beta determinants. It is necessary to know this, in order that we can add a new walker to the correct determinant (Our unique list is not sorted according to the walker number list, but the Alpha and Beta lists ARE sorted to associate with the walker number list)

4.4.1.4 constexpr std::uint64_t INLpow2 (std::uint64_t *x*) [inline]

Simply raises the input argument to the power 2, thus returning 2^x

4.4.1.5 int main (void)

M A I N - S T A R T S - H E R E

4.4.1.6 double projectorEnergy ( )

The Projector Energy gives an independent estimation of the energy from the shift, according to equation:

$$E_{proj} = \sum_{j \neq 0} \left\langle D_j | H | D_0 \right\rangle \frac{N_j(\tau)}{N_0(\tau)}$$

4.4.1.7 void SPAWN (const double *cellLength,* std::vector< int > & *trueWalkerList,* std::vector< int > & *posWalkerList,* std::vector< int > & *negWalkerList,* double(&) *KEsortedList[ORB_SIZE][3],* std::set< std::pair< long int, long int > > & *uniqueDetSet,* std::pair< std::set< std::pair< long int, long int > >::iterator, bool > & *result,* std::vector< long int > & *alphaDets,* std::vector< long int > & *betaDets*)

This function represents first step in the population dynamics of the algorithm. The spawning routine goes as follows, described verbatim from the original paper:

The spawning step: for each walker α (located on $D_{i\alpha}$), we select a (coupled) determinant D_j with normalised probability pgen(j|i) and attempt to spawn a child there with probability

$$p_s(j|i_\alpha) = \frac{\delta\tau|K_{i_\alpha j}|}{p_{gen} j|i_\alpha}(15)$$

If a spawning event is successful, (i.e. if ps exceeds a uniformly chosen random number between 0 and 1), then the sign of the child is determined by the sign of $K_{i_\alpha j}$ and the sign of the parent: it is the same sign as the parent if $K_{i_\alpha j} < 0$, and opposite to the parent otherwise. Our method to compute the generation probabilities p_{gen} is given in Appendix B. If $p_s > 1$, then multiple copies of walkers are spawned on j (namely with probability 1, $|ps|$ walkers are spawned, and with probability $ps|ps|$ an additional walker is spawned).

4.4.1.8 int totalWalkerNumber (std::vector< int > & *trueWalkerList*)

Simply counts the current number of walkers associated with all determinants, irrespective of sign. Returns: $N_w = \sum_i |N_i|$

4.4.1.9 double variableShift (const double & *delt,* const int & *AShift,* int & *intTimestep,* const double & *zeta,* std::vector< double > & *totalWalkerTracker,* std::vector< double > & *shiftTracker*)

variableShift function controls the shift in response to the population, and relates exactly to the equation:

$$S(\tau) = S(\tau - A\delta\tau) - \frac{\zeta}{A\delta\tau} ln \frac{N_w(\tau)}{N_w(\tau - A\delta\tau)}$$

4.4.2 Variable Documentation

4.4.2.1 const int AShift = 2

AShift controls how frequently the shift is changed in response to the population in the variable shift mode (AShift = 1 means every step)

4.4.2.2 const double delt = 0.003

delt is the Imaginary timestep for the propogation of the "walker" population

4.4.2.3 int initRefWalkers = 50

initRefWalkers is the number of wlakers which are initially placed on the reference (i.e Hartree Fock) determinant to begin the spawning

4.4.2.4 const int INTelectrons = numElectrons

4.4.2.5 const double Kc_CUTTOFF = 2

Kc_CUTTOFF is the kinetic energy cutoff for the plane wave basis orbitals. E.g, a cutoff of "2" will allow the orbital [4 0 0] but not [5 0 0]. Set cutoff = 2.4 for 57 Orbitals (114 Spin Orbitals)

4.4.2.6 const double numElectrons = 14

Total number of electrons -> half allocated Alpha spin, the other half allocated Beta Spin.

4.4.2.7 const int numSteps = 500000

Number of steps after which to terminate the algorithm

4.4.2.8 const double PI = 3.141592653589793

A number, nobody knows what it means

4.4.2.9 long int pow2Array[ORB_SIZE]

4.4.2.10 const double rs = 1.0

rs controls density of the Electron Gas. "rs" is the radius of the sphere whose volume is that of the cell divided by the number of electrons

4.4.2.11 const int walkerCritical = 500000

After "walker critical" walkers have been spawned after a complete cycle (post annihilation) the variable shift mode is turned on

4.4.2.12 const double zeta = 0.04

Zeta is a damping parameter which controls the agressiveness of the "shift" in the variable shift mode of the algorithm

4.5 UEGHamiltonian.C File Reference

```
#include "UEGHamiltonian.H"
#include "binaryManip.H"
#include <iostream>
#include <stdlib.h>
#include <algorithm>
```

Functions

- void INLdecimalToBinary (long int &decimal, std::string &binaryNum)
- void Di_H_Dj (const double &cellLength, double(&KEsortedList)[ORB_SIZE][3], int &i, int &a, int &b, bool &ibSpinDifferent, double &RESULT)
- void Di_H_Di (const double &cellLength, const int &numElectrons, long int &alphaBin, long int &betaBin, double(&KElist)[ORB_SIZE][3], double &RESULT)
- void excitationAlpha_iaBeta_jb (int &alpha_i, int &alpha_a, int &beta_j, int &beta_b, const int &numElectrons, const int &numOrbitals, long int &alphaDetBin, long int &betaDetBin, double(&KEsortedList)[ORB_SIZE][3], int &sign)
- void excitationSameSpinij_ab (int &spin1_i, int &spin1_a, int &spin1_j, int &spin1_b, const int &numElectrons, const int &numOrbitals, long int &spin1DetBin, double(&KEsortedList)[ORB_SIZE][3], int &sign)

4.5.1 Function Documentation

4.5.1.1 void Di_H_Di (const double & *cellLength,* const int & *numElectrons,* long int & *alphaBin,* long int & *betaBin,* double(&) *KElist[ORB_SIZE][3],* double & *RESULT*)

This function calculates the diagonal matrix elements of the Hamiltonian, according to the following equation:

$$2\frac{\pi^2}{L^2}\sum_i k_i^2 - \frac{1}{\pi L}\sum_i\sum_{i\neq j}\frac{1}{|K_i - k_j|^2}$$

This is again equivalent to the Slater-Condon rules which give the double integral $\langle ij||ij\rangle$.

4.5.1.2 void Di_H_Dj (const double & *cellLength,* double(&) *KEsortedList[ORB_SIZE][3],* int & *i,* int & *a,* int & *b,* bool & *ibSpinDifferent,* double & *RESULT*)

This function calculates the OFF-diagonal matrix elements of the Hamiltonian, for double electronic excitations, using the simplifications of the Slater-Condon rules. It calculates the double electron integral $\langle D_I|H|D_J\rangle = \langle ij||ab\rangle = \langle ij|ab\rangle - \langle ij|ba\rangle$, where the spin-orthogonality imposed upon the system mean that $\langle ij|ba\rangle = 0$ if orbital i and orbital b are of opposite spin. From the Determinant given by kpoints in detKlist: "i" is the index from which one (alpha spin) electron is excited into orbital index "a". A second (beta spin) electron is chosen from a filled orbital, j, and is excited into the unfilled orbital index "b". It is assumed that D_J has already been chosen as a coupled DOUBLE excitation of D_I such that CRYSTAL MOMENTUM (k) and SPIN are conserved.

4.5.1.3 void excitationAlpha_iaBeta_jb (int & *alpha_i,* int & *alpha_a,* int & *beta_j,* int & *beta_b,* const int & *numElectrons,* const int & *numOrbitals,* long int & *alphaDetBin,* long int & *betaDetBin,* double(&) *KEsortedList[ORB_SIZE][3],* int & *sign*)

EXCITATION - OPERATOR

i and a are ALPHA spin ; j and b are BETA spin

1) Pick a filled alpha obrital *RANDOMLY*, i

2) choose an UNfilled alpha determinant *RANDOMLY*, a —> record change in K values from i–>a, delk_ia (for each n,m, and l)

3) choose a filled beta orbital *RANDOMLY*, j

4) Run through ALL unfilled beta orbitals, and record those which conserve K, given delk_ia (i.e, given -delk_ia) —> Of those which conserve K, choose an unfilled one at *RANDOM*

5) Hopefully, the excitation ij —> ab conserves K with 100% gauruntee

4.5.1.4 void excitationSameSpinij_ab (int & *spin1_i,* int & *spin1_a,* int & *spin1_j,* int & *spin1_b,* const int & *numElectrons,* const int & *numOrbitals,* long int & *spin1DetBin,* double(&) *KEsortedList[ORB_SIZE][3],* int & *sign*)

EXCITATION - OPERATOR

i,j,a,&b are ALL of the same spin (Either Alpha or Beta)

4.5.1.5 void INLdecimalToBinary (long int & *decimal,* std::string & *binaryNum*) `[inline]`

This inline function simply converts a base 10 integer (which represents the unique determinant) into its binary representation, but with type string.