CT-AUX with spin-mixing

Generated by Doxygen 1.7.1

Mon May 7 2012 20:01:41

Contents

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 WC Class Reference

The wrapper class that implements the CT-AUX solver.

Public Member Functions

- **WC** (cd g_0_up_1[ ], cd g0_down_1[ ], cd f0_1[ ], double U, int Size, int Size1, int NoIt)
- void NIgreen ()
- void start_conf ()
- void sampling_proc ()
- void measure_times ()
- void save_and_det_Greentau (cd G_up[ ], cd G_down[ ], cd F[ ], int m, cd Gw_up[ ], cd Gw_down[ ], cd Fw[ ])
- void save_green (cd G[ ], int Size, string STR)
- void ReallocateArrays (int new_size)
- **WC** (cd g_0_up_1[ ], cd g0_down_1[ ], cd f0_1[ ], double U, int Size, int Size1, int NoIt)
- void **NIgreen** ()
- void **start_conf** ()
- void **sampling_proc** ()
- void **save_pertorder** ()
- void **green_measurement** (double tau_test, int l)
- void **save_MeasuredGreen** ()
- void **save_Green_Measure** (double Measurement_green[ ], int L)
- void **measure_times** ()
- void **save_Var_Measure** (double Measurement_var[ ], int L)
- void **save_and_det_Greentau** (cd G_up[ ], cd G_down[ ], cd F[ ], int m)
- void **save_green** (double G[ ], int Size, string STR)

Public Attributes

- int Size

  Number of Matsubara Frequencies in the sampling green function.

- int Size1

Number of Matsubara Frequencies for DMFT iteration.

- cd * g0_up

  Non-interacting green functions for DMFT iteration.

- cd * **g0_down**
- cd * **f0**
- double * tau

  Imaginary times for the sampling process.

- cd * G0_up

  Interpolated green functions for the sampling process.

- cd * **G0_down**
- cd * **F0**
- double **pi**
- int n

  dynamical perturbation order (Size of the Matrix N)

- double * taus

  dynamical imaginary times during the sampling

- int * spins

  dynamical Ising spins during the sampling

- cd * N

  dynamical matrix N. The first nxn elements are used of this 1D array to emulate a 2D matrix.

- long * seed

  seed for random number generator

- double **U**
- double K

  real parameter K for Hubbard-Stratonovich decoupling

- double * Gamma

  Vector Gamma for the sampling.

- double **phi**
- double * **omega**
- cd Sign

  Sign, just to check.

- int **Size2**
- int **NoIt**
- int **NoSteps**
- int L

  Number of Measurements for the green functions.

- cd * **G_meas**

- cd * M_meas [4]

  Accumulation M matrix for the QMC.

- double * **tau_meas**
- double pert_order

  Average perturbation order.

- double * **G0_up**
- double * **G0_down**
- double * **F0**
- double * **N**
- double * **Sign**
- double * **G_meas**

3.1.1 Detailed Description

The wrapper class that implements the CT-AUX solver.

3.1.2 Member Function Documentation

3.1.2.1 void WC::measure_times ( )

Performs the measurement for a particular configuration.

Both the imaginary time and Matsubara measurements are performed in this function.

The function first determines the Green's functions at all the required times, that result from the subtraction of all of the configuration times from one another.

The imaginary time measurements are performed with the appropriate matrix operations, taking into account of the 2x2 substructure of the spinful terms. The formula for this is:

$$G_{c,\sigma,\sigma'}(\tau) = G_{0,\sigma,\sigma'}(\tau) + \left[\sum_{lm} \mathbf{G}_0(\tau,\tau_l)\mathbf{M}_{lm}\mathbf{G}_0(\tau_m,0)\right]_{\sigma,\sigma'}$$

where the bold matrices indicate that these are indexed with spin numbers. For example:

$$[\mathbf{G}_0]_{\sigma,\sigma'} = G_{0,\sigma\sigma'}$$

The matrix M is defined through:

$$\mathbf{M}_{lm} = (e^V - 1)\mathbf{N}^c_{lm}$$

where e^V is a block diagonal matrix, with each block containing a diagonal of $e^{V_\uparrow}, e^{V_\downarrow}$. The matrix N is described in decide_insertion().

Note that the initial matrix $G_{0,\sigma,\sigma'}$ is not measured as part of every loop and instead added in the final step save_and_det_Greentau().

The Matsubara measurements are more direct, and are more simply expressed through:

$$G_{c,\sigma,\sigma'}(i\omega_n) = G_{0,\sigma\sigma'}(i\omega_n) + [\sum_{lm} \exp i\omega_n(\tau_l - \tau_m)\mathbf{G}_0\mathbf{M}\mathbf{G}_0]_{\sigma\sigma'}$$

where we can avoid much of the operation by simply accumulating values of $\mathbf{M}\exp i\omega_n(\tau_l - \tau_m$ for each iteration, and only performing the matrix product at the end.

3.1.2.2 void WC::NIgreen ( )

Fourier transform the imaginary time Weiss Green's functions to Matsubara frequency.

After Fourier transformation, this function will also spline interpolate the points to a finer grid in order to allow a better sampling procedure.

This function will also save the Green's function to "nigreen.txt".

3.1.2.3 void WC::ReallocateArrays (int *new_size*)

Reallocate the space for the configuration arrays and the N matrix.

This function increases the space for the configuration arrays tau_meas, taus and spins as well as the N matrix. It is necessary to be careful when copying the N matrix due to the storage mechanism.

3.1.2.4 void WC::sampling_proc ( )

The main part of the sampling procedure, which controls the QMC loops.

In this function, each loop of the sampling procedure is handled. For each iteration, a random choice is made to either add a new aux spin, or to remove an existing one. The success of this is then determined from the functions decide_insertion() and decide_removal().

Each 100 loops, the system will be measured. This number is chosen so that the configuration has a sufficient chance to change such that it is reasonably uncorrelated to the previous measurement. Note that for very large average_orders, this number should be made larger.

The measurements themselves are performed through measure_times().

As the size of the arrays are dynamically adjusted, this function will check to see if we have reached the current maximum size of the arrays and then call ReallocateArrays() in order to increase the size by 30.

3.1.2.5 void WC::save_and_det_Greentau (cd *G_up[ ]*, cd *G_down[ ]*, cd *F[ ]*, int *m*, cd *Gw_up[ ]*, cd *Gw_down[ ]*, cd *Fw[ ]*)

Finalise the Green's functions and save them to the appropriate files.

This function performs the operations that could be left out in the measure_times() function. It adds the non-interacting part to the imaginary time measurements, and performs the appropriate matrix multiplications for the Matsubara measurements.

The imaginary time measurements are then Fourier transformed back into Matsubara frequencies, and then these are all saved to the appropriate files. The output for each of these functions are also saved into the supplied arguments G_up, G_down, F, Gw_up, Gw_down and Fw.

The argument m exists to allow an id to set for each iteration of the solver in (for example) the DMFT loops. It can safely be set to m=0 without an issues.

3.1.2.6 void WC::save_green (cd *G[ ]*, int *Size*, string *STR*)

Convenience function to save a particular Green's function to a file.

This function saves the Green's function, which is in G and of size Size, to the file STR. The format is as rows for each element of the Green's function, and each row contains the real and imaginary parts as doubles in string format.

3.1.2.7 void WC::start_conf ( )

Generate the initial configuration for the solver.

This configuration will always be chosen with a single up auxillary spin. The two values for gamma are then determined and the N matrix is then configured.

The documentation for this class was generated from the following files:

- Weak_Coup_SpinFlip/WC_SpinFlip.cpp
- Weak_Coup_SpinFlip/WC_SpinFlip_new.cpp

Chapter 4

File Documentation

4.1 danny_interface.cpp File Reference

The interface to the main CT-AUX procedure.

```
#include <iostream>
#include <complex>
#include <assert.h>
#include <string.h>
#include "Weak_Coup_SpinFlip/nrutil.h"
#include "Weak_Coup_SpinFlip/fft1d.cpp"
#include "Weak_Coup_SpinFlip/random_fcts.cpp"
#include "Weak_Coup_SpinFlip/Splines.cpp"
#include "Weak_Coup_SpinFlip/matrix_manip.cpp"
#include "Weak_Coup_SpinFlip/WC_SpinFlip.cpp"
```

Defines

- #define **cd** complex<double>
- #define **LOG**(...)
- #define **LOGDONE**()

Functions

- const cd **I** (0, 1)
- int **main** (int argc, char **argv)

4.1.1 Detailed Description

The interface to the main CT-AUX procedure. This code was written by Danny in order to interface the CT-AUX code to python. It takes several arguments via the command line, as well as possibly reading some data via stdin.

The output is then given back in either designated files, or via stdout.

The arguments to the function must be:

1. `input_filename`: The filename of the input data. Can be "-", see comments.
2. `output_filename`: The filename where to outptu to. Can be "-", see comments.
3. `U`: The interaction strength.
4. `num_omega`: The number of Matsubara frequencies.
5. `num_sweeps`: The number of QMC iterations to perform.

It is also possible to supply an argument `--K` which will set the optional parameter K. If `--binary` is given, then the data will be saved in direct binary format (danger for portability!) or otherwise as plain text format.

When `input_filename` or `output_filename` are set to "-", then the data will be read or written to or from stdin and stdout respectively.

The input data should be given as either a series of rows of 7 numbers in text mode, which specify omega, real and imaginary parts of G0_up, real and imaginary parts of G0_down and real and imaginary parts of F0. Or as a direct binary dump of G0_up, G0_down and F0 in binary mode.

On exit, for text mode, the perturbation order and average sign of the solver will be saved, followed by rows identical to the input data, but for the Green's functions. For binary mode, the following will be saved:

1. `pert_order` (double)
2. `Sign` (complex double)
3. `G_up` (complex double array)
4. `G_down` (complex double array)
5. `F` (complex double array)
6. `Gw_up` (complex double array)
7. `Gw_down` (complex double array)
8. `Fw` (complex double array)

The quantities which include w in their names result from Matsubara measurements, whereas the other quantities results from imaginary time measurements (yet are still given in Matsubara frequencies).

4.2 Weak_Coup_SpinFlip/fft1d.cpp File Reference

Routines to perform the Fourier transforms.

```
#include <gsl/gsl_fft_complex.h>
```

Defines

- #define **REAL**(z, i) ((z)[2∗(i)])
- #define **IMAG**(z, i) ((z)[2∗(i)+1])

Functions

- void fft1d (cd in_fct[ ], cd out_fct[ ], int dim1)
- void fftMatsubara (cd G_Tau[ ], cd G_omega[ ], int Size)
- void fftMatsubaraF (cd G_Tau[ ], cd G_omega[ ], int Size)
- void ifftMatsubara (cd G_Tau[ ], cd G_omega[ ], int Size)
- void ifftMatsubaraF (cd G_Tau[ ], cd G_omega[ ], int Size)

4.2.1 Detailed Description

Routines to perform the Fourier transforms. The routines here use the GSL (GNU Scientific Library) to perform a FFT on the Green's functions, from and to Matsubara frequency and imaginary time.

4.2.2 Function Documentation

4.2.2.1 void fft1d (cd *in_fct[ ],* cd *out_fct[ ],* int *dim1*)

A wrapped around the GSL Fourier transform library.

4.2.2.2 void fftMatsubara (cd *G_Tau[ ],* cd *G_omega[ ],* int *Size*)

Fourier transform the supplied Matsubara frequency function into imaginary time. This specific function is for the functions diagonal in spin.

Because the Fourier transform is not well suited (numerically) to transforming the leading $1/(i\omega_N)$ factor, this is treated as a separate term in the transform, and manually transformed. This is possible, as one knows that this term must have a coefficient of 1.

4.2.2.3 void fftMatsubaraF (cd *G_Tau[ ],* cd *G_omega[ ],* int *Size*)

Perform the Fourier transform from Matsubara to imaginary time, for the cross term Green's functions.

In this case we do not need to worry about the $1/(i\omega_N)$ factor, as it is not present here.

4.2.2.4 void ifftMatsubara (cd *G_Tau[ ],* cd *G_omega[ ],* int *Size*)

The inverse transform of fftMatsubara(). That is, from imaginary time to Matsubara frequency.

The leading order term is also accounted for here.

4.2.2.5 void ifftMatsubaraF (cd *G_Tau[ ],* cd *G_omega[ ],* int *Size*)

The inverse transform of fftMatsubaraF(). That is, from imaginary time to Matsubara frequency.

4.3 Weak_Coup_SpinFlip/matrix_manip.cpp File Reference

Routines to handle the core work of updating the N matrix of the QMC simulation.

Functions

- cd **sign** (cd l)
- int index (int l, int m, int Size)
- void set_S_tilde (cd S_tilde[ ], cd N[ ], int j, int Size2)
- void invert (cd M[ ], cd Det)
- cd det (cd M[ ])
- void mat_red (cd N[ ], double taus[ ], int spins[ ], cd S[ ], int j, int n, int Size2)
- void mat_inc (cd N[ ], double taus[ ], int spins[ ], cd Q[ ], cd R[ ], cd S[ ], cd S_tilde[ ], double Tau, int Spin, int n, int Size2)
- int decide_removal (cd N[ ], double taus[ ], int spins[ ], double K, double Gamma[ ], int n, int Size2, long seed[ ], cd &Sign)
- int decide_insertion (cd N[ ], double taus[ ], int spins[ ], double Gamma[ ], cd G0_up[ ], cd G0_-down[ ], cd F0[ ], double tau[ ], int n, int Size, int Size2, long seed[ ], double K, cd &Sign)

4.3.1 Detailed Description

Routines to handle the core work of updating the N matrix of the QMC simulation. These functions make use of fast matrix updates, which are possible due to the rank-1 updates (i.e. simple insert of a row and column to the matrix).

The N matrix itself is defined through a block like structure, where each block:

$$\mathbf{N^{-1}}_{lm} = e^{V_l}\delta_{lm} - \mathbf{G}_{0,lm}(e^{V_l} - \mathbf{I})$$

contains a 2x2 block that describes the spin structure, where e^{V_l} is a diagonal matrix with the elements $(e^{\gamma s_l}, e^{-\gamma s_l})$.

4.3.2 Function Documentation

4.3.2.1 int decide_insertion (cd *N[ ]*, double *taus[ ]*, int *spins[ ]*, double *Gamma[ ]*, cd *G0_up[ ]*, cd *G0_down[ ]*, cd *F0[ ]*, double *tau[ ]*, int *n*, int *Size*, int *Size2*, long *seed[ ]*, double *K*, cd & *Sign*)

Determine the probability to insert a new spni and act out the insertion.

This function randomly chooses a new time to insert a random spin and then calculates the ratio between the configuration probabilities for the system before and after insertion.

The ratio is simply the ratio of determinants of the N matrices. Due to the structure of the matrices, this comes down to the Schur complement that arises in the block matrix inversion.

As many interpolation routines must be called in this function, these values are saved and, in the case that the insertion is successful, are passed to the mat_inc() function that performs the actual insertion.

4.3.2.2 int decide_removal (cd *N[ ]*, double *taus[ ]*, int *spins[ ]*, double *K*, double *Gamma[ ]*, int *n*, int *Size2*, long *seed[ ]*, cd & *Sign*)

Determine the probability to remove a spin and act out the removal.

This function randomly determines a spin to remove using random_removal() and then calculates the ratio between the configuration probabilities for the system before and after removal.

This probably is simply the ratio of the determinant of the N matrices associated to each configure. Due to the structure of these matrices, the ratio may be calculated simply as the inverted determinant of the 2x2 block that will be removed.

In the case that the removal is successful, the inverted block is sent in the call to mat_red(), which does the actual work to remove the spin.

4.3.2.3 cd det (cd *M[ ]*)

A convenience function for the determinant of a 2x2 matrix.

4.3.2.4 int index (int *l*, int *m*, int *Size*)

A convenience function for indexing the large matrices, which are stored as 1D arrays.

4.3.2.5 void invert (cd *M[ ]*, cd *Det*)

A convenience function to invert a 2x2 matrix.

4.3.2.6 void mat_inc (cd *N[ ]*, double *taus[ ]*, int *spins[ ]*, cd *Q[ ]*, cd *R[ ]*, cd *S[ ]*, cd *S_tilde[ ]*, double *Tau*, int *Spin*, int *n*, int *Size2*)

Perform the actual matrix insertion on the matrix N.

This function takes as arguments the new elements to be inserted in the matrix. These come from interpolation calls of decide_insertion().

This function makes use of block matrix inversion in order to speed up the insertion process.

4.3.2.7 void mat_red (cd *N[ ]*, double *taus[ ]*, int *spins[ ]*, cd *S[ ]*, int *j*, int *n*, int *Size2*)

Perform the removal update on the matrix N.

This function uses the block matrix formula:

$$A^{-1} = P - QS^{-1}R$$

where

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} P & Q \\ R & S \end{bmatrix}^{-1}$$

For optimization, the matrix S^{-1} is supplied by the previous code, which has already used it to determine the removal weight.

4.3.2.8 void set_S_tilde (cd *S_tilde[ ]*, cd *N[ ]*, int *j*, int *Size2*)

A convenience function to extract one 2x2 part of the total N matrix.

4.4 Weak_Coup_SpinFlip/PythonSplines.cpp File Reference

Spline interpolation routines, using python.

Functions

- char * CreateTempFile (const char *template_name)
- void PythonSplines (double *x_in, double y_in[ ], double x_out[ ], double y_out[ ], int Size, int n)
- void PythonSplinesComplex (double *x_in, cd y_in[ ], double x_out[ ], cd y_out[ ], int Size, int n)

4.4.1 Detailed Description

Spline interpolation routines, using python.

4.4.2 Function Documentation

4.4.2.1 char* CreateTempFile (const char * *template_name*)

A convenience function for creating a safe temporary file.

By default, the function will use the location "/tmp". However, if the TMPDIR environment variable is set, then this will be used. A file in this location will then be created, using "template_name" as the filename. "template_name" should contain "XXXXXX", such that the function mkstemp() can properly function.

4.4.2.2 void PythonSplines (double * *x_in*, double *y_in[ ]*, double *x_out[ ]*, double *y_out[ ]*, int *Size*, int *n*)

This function performs a spline interpolation, using the python class UnivariateSpline.

This function was written to better handle the derivatives at the endpoints of the imaginary time Green's functions.

4.4.2.3 void PythonSplinesComplex (double * *x_in*, cd *y_in[ ]*, double *x_out[ ]*, cd *y_out[ ]*, int *Size*, int *n*)

An interpolation as for PythonSplines(), but allowing complex functions.

4.5 Weak_Coup_SpinFlip/random_fcts.cpp File Reference

Random number routines.

```
#include <stdlib.h>
```

Functions

- double num_recipes_ran1_old (long idum[ ])
- double num_recipes_ran1 (long idum[ ])
- double random_tau (long tau_seed[ ])
- int random_spin (long spin_seed[ ])
- int random_removal (long rem_seed[ ], int n)
- bool acceptance (long acc_seed[ ], double prop)

4.5.1 Detailed Description

Random number routines. This file contains some random number routines, as implemented in Numerical Recipes.

The Numerical Recipes routine here was replaced by Danny for the glibc routine. This was done in the process of testing for bugs, and I found it to be more reliable in its even distribution of generated numbers.

4.5.2 Function Documentation

4.5.2.1 bool acceptance (long *acc_seed[ ],* double *prop*)

Choose whether to accept or reject a configuration update, based on the propability prop.

This is equivalent to asking the question x $<$ prop, where x is a random number linearly distributed between 0 and 1.

4.5.2.2 double num_recipes_ran1 (long *idum[ ]*)

Replacement random generator using the glibc function.

Note that the argument is required as an artifact of the Numerical Recipes function. This argument should be a negative number in the first call of this function, at which time the random number generator will be seeded with the absolute value of that number.

It is recommended to assign a value to this number using a truely random source, for example /dev/urandom (see WC::WC()). Note that using the current time is a bad choice, as multiprocessing can cause many runs of this code to use the same seed for the random number generator.

4.5.2.3 double num_recipes_ran1_old (long *idum[ ]*)

The old Numerical Recipes random number generator, superceeded by num_recipes_ran1().

4.5.2.4 int random_removal (long *rem_seed[ ],* int *n*)

Choose a random spin from the current set of spins in the configuration.

This is equivalent to a random integer modulo n.

4.5.2.5 int random_spin (long *spin_seed[ ]*)

Generate a random up or down spin.

4.5.2.6 double random_tau (long *tau_seed[ ]*)

Generate a random time for the configuration.

4.6 Weak_Coup_SpinFlip/Splines.cpp File Reference

This file contains some spline interpolation routines, that (I believe) are from Numerical Recipes.

Functions

- void **spline_derivative** (double y[ ], int n, double y2[ ])
- double **splint** (double xa[ ], double ya[ ], double y2a[ ], int n, double x)
- void **splines** (double x_in[ ], double y_in[ ], double y_prime[ ], double x_out[ ], double y_out[ ], int Size, int n)
- void **linear** (double x_in[ ], double y_in[ ], double x_out[ ], double y_out[ ], int n, int Size)
- cd **single_linear** (double x_in[ ], cd y_in[ ], double x_out, int Size)

4.6.1 Detailed Description

This file contains some spline interpolation routines, that (I believe) are from Numerical Recipes. The routines here have been replaced in the main program by Danny, because they suffered from some issues of derivative choice at the endpoints of the function.

4.7 Weak_Coup_SpinFlip/WC_SpinFlip.cpp File Reference

The main file of the solver including the class.

```
#include "PythonSplines.cpp"
```

Classes

- class WC

  The wrapper class that implements the CT-AUX solver.

Functions

- double sign (int l)

4.7.1 Detailed Description

The main file of the solver including the class. This file contains the majority of the control of the solver. All of the details are contained in a class WC, which keeps track of all important details. In order to use this, one should call the functions with something similar to:

```
int Size = 20000;
WC * ImpSol=new WC(g0_up, g0_down, f0, U, Size, num_omega*2,num_sweeps);
ImpSol->K = K;
ImpSol->NIgreen();
ImpSol->sampling_proc();
ImpSol->save_and_det_Greentau(G_up, G_down, F, 0,Gw_up,Gw_down,Fw);
delete ImpSol;
```

where the appropriate Matsubara input Weiss green's functions are specified in g0_up, g0_down and f0 and the output is written to files and into the arrays supplied by G_up, G_down, F, Gw_up, Gw_down and Fw.

Note that this code assumes that the inverse temperature is beta=1. This parameter should therefore be scaled appropriately for the input Weiss Green's functions and interaction parameters.

4.7.2 Function Documentation

4.7.2.1 double sign (int *l*)

A convenience function. Effectively returns a/|a|.

Index