Use of the CCS/MCE Program

Chris Symonds

Abstract

The following is a brief description of the running asnd use of the MCE/CCS program. The following are discussed:

- The running of the program
- The input files
- The raw data output files
- The combined data outputs
- The data plotting outputs
- The procedure for finding the optimal running parameters, especially the number of repeat executions needed for convergence and the maximum timestep for propagation

1. Running the Program

The program allows a wide array of simulations using the MCE and CCS equations. The files are separated into three folders : `run`, `src` and `build`. The `src` folder contains all the source files, the `build` folder contains the Makefile and any temporary compilation files, and the `run` folder contains scripts to compile and run the program and collect and combine results in addition to the input files containing the simulation parameters, which are discussed in more detail in section 2. This section will describe the function of the scripts in the `run` folder.

The MCE and CCS methods usually require a large number of repeat executions. These repeat executions are split in two levels - firstly a number of different instances of the program are created in separate folders (these are named `1-run, 2-run,` etc), and secondly the program reruns itself for a number of times, which can be done on multiple processors simultaneously using OpenMP directives. To run the program the `run.sh` file in the run folder is executed, with the following arguments:

$1 - Total number of repeat calculations desired
$2 - Number of parallel threads per folder
$3 - Number of folders to split the job into (referred to as sub-folders).

For example, to run 128 jobs using 4 parallel cores, with the load split into 4 folders, you would use the command

```
./run.sh 128 4 4
```

which would have a total of 16 parallel threads running simultaneously. This allows OpenMP execution to be carried out over many more cores than would be present on a single node. Within the `run.sh` script, the arguments are checked first to ensure they comply with the following conditions:

- There are 3 arguments used;
- All arguments are integers;
- All arguments are greater than 0;
- $2 is not greater than the number of cores per node of the machine;
- $3 is less than 100;
- $1/$3 is less than 1000 (Less than 1000 repeats per program instance);
- If the conjugate_repeats flag is enabled, that the total number of repeats is an integer multiple of 2 × $2 × $3;
- If the conjugate_repeats flag is disabled, that the total number of repeats is an integer multiple of $2 × $3;
- The total number of cores requested ($2 × $3) is below 100;
- If there is no job management system present, that $3 is equal to 1.

Following this the folders in which the program will be executed are set up, and the program is compiled by way of the Makefile in the `build` folder. For portability the program is compiled using the GNU fortran compiler, gfortran. An issue encountered during development is that the intel ifort 12.1 compiler results in problems due to a known bug with allocation through subroutines when OpenMP is enabled, however the intel compiler can be used if not version 12.1 and this would require modifications to the makefile in the build folder. Once compilation is completed the input files and the program executable are copied to the execution folders, along with a job script used by the job management system which controls how long the program will run for, how much system memory it can be allocated, how many parallel cores are needed and how many simultaneous instances of the program will be running. By submitting this job script to the job management system all instances of the program will be run.

The `run.sh` script creates a second script, called `result.sh` which when run calls the `collate.sh` script which combines the results from all the completed runs. This script collects all the data from the different subfolders, calls averaging programs to combine this data, then puts all relevant files in a results folder, before deleting the raw data. By disabling a flag in the script, this raw data can be preserved. The `collate.sh` script requires the following arguments:

$1- The path of the folder in which the raw data exists
$2- The total number of repeats for the entire simulation
$3- The number of sub-folders in the raw data folder
$4- The random number generated by the run.sh script as a unique run identifier
$5- The name of the results file which calls this script (deleted at the end)

Once all the output files are collected and averaged a set of gnuplot scripts are created, and if gnuplot is present on the machine these scripts are run, allowing instant graphical results.

Often a cluster computer with many users will impose time limits on running jobs, and once this time limit is reached the job will be cancelled, aborting the program. If time limits cause a simulation to be aborted prematurely, this simulation can be restarted through the `restart.sh` script which uses the same arguments as the run.sh script. This file should be called from the running folder of the most recent partial run. When this script is run it collects the output files from the execution folders and uses them as input files for a new set of simulations before calling the `run.sh` script. This allows the simulation to restart with a wavefunction in the same state as when the program was aborted.

If multiple partial runs are used, the data from the different partial runs can be combined with the `combine.sh` script which requires the file `folderlist.dat`, a file containing a list of the required folders which should be in order. This file is automatically created by the `restart.sh` script, however care should be taken to ensure that no confusion occurs when there are multiple simulations happening at the same time. This script overwrites the output files of the final partial run with the combined data (after making a backup), averages the data in each sub-folder (as would be done by the main program), and then calls the `collate.sh` script to combine and average the data.

2. Input Files

There are three input files containing all the parameters needed : `input.dat`, `inham.dat` and `prop.dat`. This section will contain a description of the various parameters in the different input files. The `input.dat` file controls most aspects of the program infrastructure and operation, and contains parameters given in tables 1 and 2. Unless otherwise stated, it should be assumed that the values given are case sensitive.

The `inham.dat` file contains parameters specific to the available systems. Different parameters are read depending on the value of the “System” parameter in the `input.dat` file. Table 3 gives these parameters, and unless otherwise stated all parameters are double precision real numbers.

Parameter	Values	Effect	Restrictions
System	SB	Spin Boson	Usable only with MCE
	HP	Harmonic Potential	Usable only with CCS
	FP	Free Particle	Usable only with CCS
	MP	Morse Potential	Usable only with CCS
	IV	Inverted Gaussian	Usable only with CCS
	CP	Coulomb Potential	Usable only with CCS
	HH	Hennon-Heiles Potential	Usable only with CCS
Runfolder	default	Runs program in a folder with the pattern <method>-<system>-<rand> ie CCS-HP-31254	Case insensitive
	<any string>	<method>-<system>-<string> ie CCS-HP-withgrids	
debug	0	No debug output generated	
	1	Debug outputs generated	Will not work with dynamic basis set sizes
gen	YES/NO	Basis set generated	
prop	YES/NO	Basis set propagated	
restart	0	Standard simulation	
	1	Restart a prior simulation	Set by restart.sh
cmprss	YES/NO	Automatically change the compression parameter or the grid spacing to ensure an acceptable norm	
method	CCS	Runs using CCS equations	Only valid for 1 PES
	MCEv1	Runs using MCEv1 eqns	Only valid for >1 PES
	MCEv2	Runs using MCEv2 eqns	Only valid for >1 PES
	MCE12	Two jobs are submitted, 1 using MCEv1 eqns, the other using MCEv2 eqns	Only valid for >1 PES
repeats	<number>	Number of repeats for this instance of the program	Integer. Set automatically by the run.sh script
Conjugate_Repeats	YES/NO	If yes, constructs the basis set for every other repeat around the conjugate of z_0 of the previous run.	If enabled must have an even no. repeats for each instance of the program. Also incompatible with restarting a prior simulation.
in_nbf	<number>	Initial number of basis functions.	Integer
ndim	<number>	Number of degrees of freedom in the system	Integer
in_PES	<number>	Initial electronic state of the wavefunction	Integer

Table 1: Parameters in the input.dat file, part 1

The `prop.dat` file contains parameters specific to the propagation of the wavefunction. Table 4 lists the parameters contained within this file. All values in the file are in atomic units, and all but the final parameter are double precision real numbers.

Parameter	Values	Effect	Restrictions
npes	<number>	Number of electronic states in the system	Integer. Must be 1 for CCS and >1 for MCE
basis	SWARM	Basis set is a swarm	
	SWTRN	Basis set is a swarm of trains	
	TRAIN	Basis set is a single train	
	GRID	Basis set is a regular grid	Only valid for $ndim \in \{1, 3\}$
	GRSWM	Basis set is a regular grid in one dimension with swarms in the other two	Only valid for $ndim = 3$
ALCMP	<number>	Seed number for α_c	Double precision real number
nbfadapt	yes/no	Enables adaptive basis sets	Only valid with GRID or GRSWM basis
nbfepsilon	<number>	Adaptive basis set cutoff ζ parameter	Double precision real number in the form `1.0d-x`
gridsp	<number>	Spacing Δ between adjacent grid points	Real number in the range $0.8 \leq \Delta \leq 2.0$
qsizex	<number>	Size of a 3D grid in x_q	Integer
psizex	<number>	Size of a 3D grid in x_p	Integer
qsizey	<number>	Size of a 3D grid in y_q	Integer
psizey	<number>	Size of a 3D grid in y_p	Integer
qsizez	<number>	Size of a 1D or 3D grid in z_q	Integer
psizez	<number>	Size of a 1D or 3D grid in z_p	Integer
Cloning	yes/no	Enables basis set cloning	
max_cloning	<number>	Maximum number of cloning events	Integer
clon_freq	<number>	Minimum number of timesteps between cloning events	Integer ≥ 50
trainsp	<number>	Number of timesteps between adjacent basis functions in a train-type or swarm-train basis	Integer
def_stp	<number>	Number of basis functions per train in a swarm-train basis set	Integer. May be increased or decreased by 1 by program
matfun	zgesv/zheev	Allows change of function used for linear equations	
SEED	<number>	Seed of random number generator	Defaults to 0 which takes the seed from `/bin/urandom`
gamma	<number>	Width parameter for CSs	Default value 1.0d0
mu	<number>	Centre of z_0	Double precision real number
hbar	<number>	Value of $\hbar$.	Defaults to $\hbar = 1$.

Table 2: Parameters in the input.dat file, part 2

3. Output Files

There are three types of output files generated by the program. The first is the raw data, which is a set of files specific to each individual repeat simulation. The second is the combined data which is data that has been combined and averaged over a set or over all repeat simulations. The third is plotting outputs which contains gnuplot commands to graphically display data from other output files. This section will give a description

Parameter	Description
Spin Boson Model Parameters	
SBDelta	Tunnelling amplitude between states Δ, with default value of 1.0d0
SBEps	Bias detuning parameter ϵ
SBw	Cutoff frequency ω_c
SBwmax	Largest frequency of the bath modes, which should be $\omega_{max} = 5\omega_c$
SBkondo	Kondo parameter α_k
SBBeta	Thermal parameter $\beta = 1/kT$
SBupnorm	Largest allowed value for the initial norm
SBdownnorm	Smallest allowed value for the initial norm
Harmonic Potential Parameters	
HPw	Frequency of the harmonic oscillator
HPupnorm	Largest allowed value for the initial norm
HPdownnorm	Smallest allowed value for the initial norm
Free Particle Parameters	
FPmass	Mass of the free particle
FPupnorm	Largest allowed value for the initial norm
FPdownnorm	Smallest allowed value for the initial norm
Morse Potential Parameters	
MPw	Frequency of the Morse oscillator
MPmass	Mass of the particle in a Morse oscillator
MPDissEn	Dissociation Energy for Morse ocillator
MPWellParam	Width of potential well for Morse oscillator
MPupnorm	Largest allowed value for the initial norm
MPdownnorm	Smallest allowed value for the initial norm
Inverted Gaussian Parameters for High Harmonic Generation	
IVm	Mass of particle (electron) in strong field
IVw	Frequency of laser field
IVlambda	Width of Gaussian potential well
IVIntensity	Intensity of laser field
IVupnorm	Largest allowed value for the initial norm
IVdownnorm	Smallest allowed value for the initial norm
Coulomb Potential Parameters for High Harmonic Generation	
CPm	Mass of particle in a strong laser field
CPfrequency	Frequency of laser field
CPRc	Initial distance between particle and the potential well
CPIntensity	Intensity of the laser field
CPupnorm	Largest allowed value for the initial norm
CPdownnorm	Smallest allowed value for the initial norm
Henon-Heiles Parameters	
HHcoupling	Coupling between Henon-Heiles oscillators
HHupnorm	Largest allowed value for the initial norm
HHdownnorm	Smallest allowed value for the initial norm
Energy Checking Parameters	
ECheck (YES/NO)	Flag to enable checking of energy $p^2/2$ of single initial basis functions to ensure they will not run away from each other too fast (YES/NO)
Ntries (Integer)	Number of recalculations before program is aborted if energy check fails
Ebfmax	Maximum allowed energy of single initial basis function
Ebfmin	Minimum allowed energy of single initial basis function

Table 3: Parameters in the inham.dat file

Parameter	Description
dtmin	Minimum allowed step size, used for adaptive step size propagation
dtmax	Maximum allowed step size, used for adaptive step size propagation
dtinit	For adaptive step size propagation, this is the size of the first time step taken. For static step size propagation this is the size of every time step taken
time_end	End time of propagation
time_start	Start time of propagation. For restarted simulations this is set automatically
step	(adaptive/static) Determines the type of propagation.

Table 4: Parameters in the prop.dat file

of all the possible output files, their data fields and under what conditions the file will be created.

3.1. Raw Data Outputs

The raw data outputs are created in the running sub-folder (named `1-run, 2-run,` etc). Most of them refer to a single repeat simulation of the program, and have the naming convention "<name>-#.out", where # denotes the repeat number.

(a) `normpop-#.out` contains data for each time step for a single repeat and is always created. This is the main data file for the simulation, and for most simulations this will be the only output file used in data processing. The `normpop-#.out` file contains the time, norm, autocorrelation function (real, imaginary and absolute), population in each electronic state and an extra field with real, imaginary and absolute parts which can be determined by the system (for example dipole acceleration for HHG simulations, or cross-correlation function). If there are two electronic states then this file will also contain the population sum and population difference.

(b) `Outbs-#.out` contains the wavefunction data for the most recent time step, and can be used to restart a simulation. If the program is run only to generate an initial basis set, this is also the main output file. It contains all the wavefunction data needed to restart a simulation and is laid out in three sections. Firstly there is a preamble containing the number of degrees of freedom, number of electronic states, number of basis functions, initial electronic state of the wavefunction, the linear algebra equation used (zgesv/zheev) and the current simulation time. Secondly there is the real and imaginary parts of the $|\mathbf{z}_0\rangle$ state in all dimensions. Finally there are the wavefunction parameters D_k, $d_k^{(r)}$, $S_k^{(r)}$, and $z_k^{(m)}$ for all values of k, r, and$m$. This file does not however contain the information contained in `prop.dat` or `inham.dat` or any information about cloning so a restarted simulation will still need the main input files.

(c) `timesteps.out` contains a list of all time step sizes and is only generated when adaptive step size propagation is enabled

(d) `clonearr-#.out` contains a list of the basis function indices ($k = 1 \ldots N_{bf}$) with both the number of times that particular basis function has undergone cloning and the most recent time step in which that basis function was cloned. This file is only generated when basis function cloning is enabled, and can be used to restart a simulation which has used cloning to ensure that N_{cln} is not exceeded and that two cloning events do not occur too close together in time.

(e) `Clonetrack-#.out` is written to every time a cloning event occurs, and contains the index of the basis function which has been cloned, the index to which the new cloned basis function has been written, the time step number at which cloning occurred and the absolute values of the single configuration amplitudes $a_k^{(r)}$. The data in this output file can be used to reconstruct a cloning history of the wavefunction.

(f) `normpop-#_interp.out` is an interpolated form of the `normpop-#.out` file, created when adaptive step size propagation is used. This is needed for averaging all the data from each repeat execution in such cases as it ensures that each file being averaged will have the same size time step.

(g) `wavefn-#.out` gives the D_k amplitude and the real and imaginary parts of the $z_k^{(m)}$ values for all basis functions. This is only generated when a 1D adaptive grid is used, and the data within it can be used to visualise the wavefunction. As a regular grid removes the need for repeat executions, the number in the filename is the time step number, with one file being created at each time step.

(h) `PropVars-#.out` contains all the wavefunction parameters D_k, $d_k^{(r)}$, and $S_k^{(r)}$ for all values of $k, \text{and} r$ at each time step. This file is only created when the debug flag is enabled in the `input.dat` file.

(i) `Traj-#.out` contains all the real and imaginary parts of the coherent state variables $z_k^{(m)}$ for each value of k and m at each time step. This file is also only created if the debug flag in the `input.dat` file is enabled.

(j) `normpop.out` is an averaged combination of all the `normpop-#.out` files for a single instance of the program (or if adaptive step sizes are used, an averaged combination of all the `normpop-#_interp.out` files). This file is created by the main program itself at the end of propagation, but only when the program has completed execution uninterrupted. If the program has restarted at any point this file will not be created and is instead made by a small external program called `subavrg.exe` (with the help of a separate interpolation program if needed)

3.2. Combined Outputs

The combined outputs are created in the output folder upon running of the `collate.sh` script. Within this folder also is the output files from the job management system if any exist, which gives the status of the program as it runs and contains any error messages generated as the program runs. In these files, for the most part, # will denote the sub-folder number from which the data file originated.

(a) `normpop_#.out` is a copy of the file `normpop.out` from each sub-folder, containing the time, norm, autocorrelation function (real, imaginary and absolute), population in each electronic state and an extra field with real, imaginary and absolute parts which can be determined by the system (for example dipole acceleration for HHG simulations, or cross-correlation function). If there are two electronic states then this file will also contain the population sum and population difference.

(b) `normpop_cumul_#.out` contains a cumulative average of the data from the `normpop_#.out` files. There are always as many files of the `normpop_cumul_#.out` type as there is of the `normpop_#.out`, and are numbered by the number of repeat executions considered, so if each subfolder has 32 repeats then the first will be named `normpop_cumul_32.out` and be identical to `normpop_1.out`, the second will be named `normpop_cumul_64.out` and be an average of the data from the `normpop_1.out` and `normpop_2.out` files, the third will be named `normpop_cumul_72.out` and be an average of the data from the `normpop_1.out`, `normpop_2.out` and `normpop_3.out` files, and so on. The highest numbered file will be an average over all repeat executions. The way in which these files can be used to test convergence is detailed in section 4.

(c) `Outbs-#_#.out` is a copy of the `Outbs-#.out` file from each subfolder, with the first numerical identifier being the repeat number and the second being the sub-folder number. These files are used for restarting a prematurely aborted simulation.

(d) `timesteps.out` is a concatenation of all the `timesteps-#.out` files from each subfolder, containing the sizes of time steps for all repeat executions of the program. This is only generated if adaptive step size propagation has been used.

(e) `timehist_#.out` contains histogram data generated from the individual `timesteps-#.out` files. This is also only created if adaptive step size propagation is used.

(f) `timehist_all.out` contains histogram data generated from the `timesteps.out` file, and so is a histogram of time steps for all instances of the simulation.

(g) `popdiffresiduals.out` contains the differences between the population difference from the highest numbered `normpop_cumul_#.out` file and that from each of the lower numbered `normpop_cumul_#.out` files. This can be used to assess convergence as is shown in section 4.

3.3. Plotting Outputs

The plotting outputs can be divided into two categories. The first, used mainly as a preliminary check on the validity of a simulation and to ensure that no major errors have occurred, plots data from the files enumerated in section3.1. The second plots data from the files enumerated in section 3.2 and can be used to assess the combined data and in the process of finding the optimal running parameters. The second type will be discussed in detail in section 4, while the first type will be discussed here. As in section 3.1, unless otherwise stated it should be assumed that the numerical identifiers # refer to the repeat number.

(a) `plotacf-#.gpl` plots the real, imaginary and absolute values of the autocorrelation function from the data in the `normpop-#.out` file.

(b) `plotdif-#.gpl` plots the population difference from the `normpop-#.out` file

(c) `plotext-#.gpl` plots the real, imaginary and absolute values from the "extra" output data field in the `normpop-#.out` file, which can be the dipole acceleration in the case of HHG simulations or the cross-correlation function for example.

(d) `plotnrm-#.gpl` plots the norm from the `normpop-#.out` file.

(e) `plotamps-#.gpl` plots the real values of the D_k prefactors against their imaginary counterparts for each basis function k. The data for these plots are taken from the `PropVars-#.out` file and as with the data file, this plotting file will only be created if the debug flag is enabled in the `input.dat` file

(f) `plotd-#.gpl` plots the real values of the $d_k^{(r)}$ single configuration amplitudes against their imaginary counterparts for each basis function k and each electronic state r. The data for this plot also comes from the `PropVars-#.out` file and so again this file is only created when the debug flag is enabled

(g) `plotact-#.gpl` plots the action $S_k^{(r)}$ against time for each of the basis functions k and each electronic state r. Again the data for this plot originates in the `PropVars-#.out` file and so the plotting file is only created when the debug flag is enabled

(h) `plot-Traj-#.out` plots the real and imaginary parts of the coherent state parameters $z_k^{(m)}$ for each basis function k and each degree of freedom m. The data used is taken from the `Traj-#.out` file and so this plotting file is only created when the debug flag is enabled.

4. Finding Optimal Running Parameters

When running a simulation it is important that the correct size of time step is used. A useful way of ensuring that the correct time step is always used is by using the adaptive time step system. Unfortunately if the basis set is constructed using trains at all this is not an option, as all basis functions must be equally temporally spaced at all times during simulation, else instabilities could result. A good way of ensuring that an adequate time step is used is by running the simulation first with a swarm-type basis set and adaptive step size propagation. Once the simulation data is collected a histogram of all timestep sizes can be generated by the files `plothist.gpl` and `plothistall.gpl`. The `plothistall.gpl` file uses data from the `timehist_all.out` file and so returns a histogram of all time steps taken by all instances of the program and as a check the `plothist.gpl` file plots the individual histograms from the time step data from each of the sub-folders to ensure that no outliers are effecting the combined data. The importance of carrying out this preliminary run can be seen in figures 1 and 2, which show the histogram data for adaptive step size simulations of the symmetric and asymmetric cases of the spin boson model respectively. For the symmetric case of the spin boson

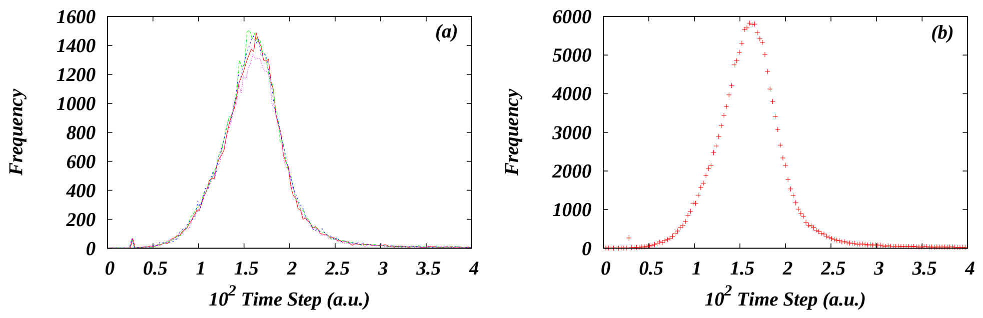

Figure 1: Histograms of the time steps taken during adaptive time step propagation for the symmetric case of the spin boson model, with individual histograms from each of the sub-folders in (a) and a histogram of all the data combined in (b)

model the peak of the distribution of time steps is in the region of $dt = 0.016$ and considering the width of the distribution an acceptable value of the static step size would be $dt = 0.01$. This value however would be completely unsuitable for the asymmetric case of the spin boson model, which peaks around $dt = 0.0036$ giving an acceptable value of the static step size at $dt = 0.002$.

A second important consideration is that the number of repeat executions of the program be sufficient to ensure an acceptable level of convergence. To aid in this, two plotting files are created, which compare the population difference from each of the cumulatively averaged data files `normpop_cumul_#.out` and the residuals between these population differences and that from the highest numbered `normpop_cumul_#.out` file.

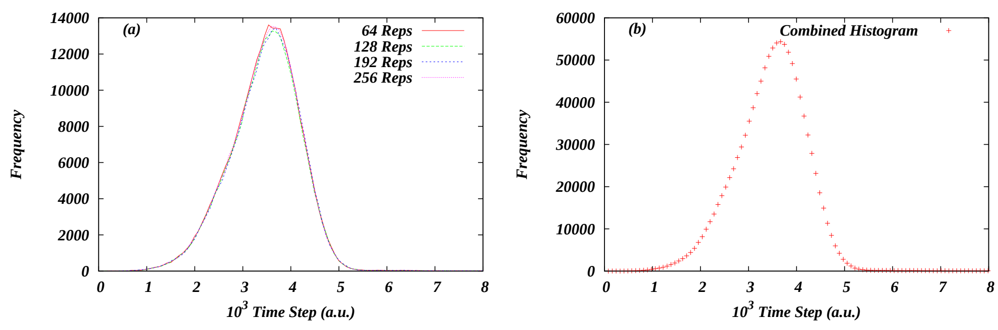

Figure 2: Histograms of the time steps taken during adaptive time step propagation for the asymmetric case of the spin boson model, with individual histograms from each of the sub-folders in (a) and a histogram of all the data combined in (b)

Examples of these plots are given in figures 3 and 4 for the symmetric and asymmetric cases of the spin boson model, using uncloned swarm-type basis sets and the MCEv2 equations. In some cases a lack of convergence will be readily apparent in the population difference plot (a), however sometimes the residual plot (b) would need to be considered. In the cases above it can be easily seen that convergence is approached, with the instability in the residuals greatly reduced by $N_{rpt} = 192$ in the asymmetric case and the overestimation of the oscillations reduced in the symmetric case, although it should be noted that the residuals for the symmetric case are never as large as is seen for the asymmetric case, indicating that a larger number of repetitions may be needed for the asymmetric case than for the symmetric case.

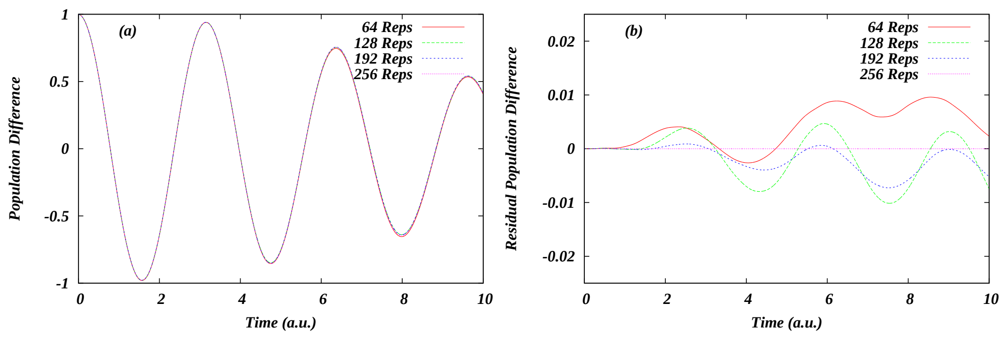

Figure 3: Plots used to test convergence in the population difference for the symmetric case of the spin boson model, where (a) is the population difference averaged over different numbers of repeats, and (b) is the difference between the those population differences and that averaged over 256 repeats

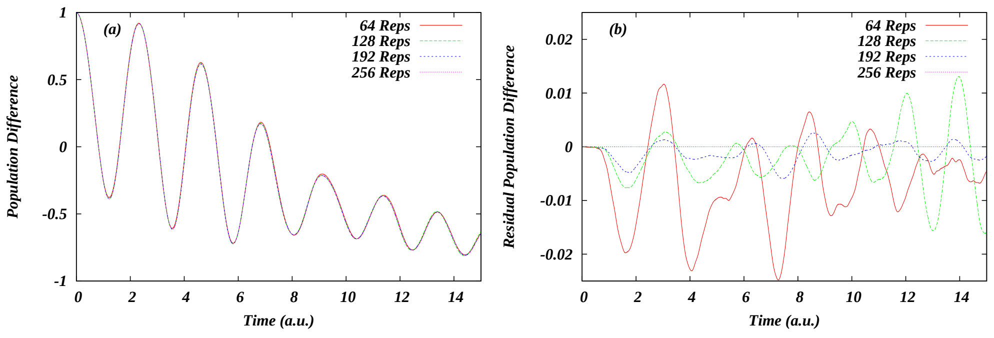

Figure 4: Plots used to test convergence in the population difference for the asymmetric case of the spin boson model, where (a) is the population difference averaged over different numbers of repeats, and (b) is the difference between the those population differences and that averaged over 256 repeats