OpenMCTDHB User manual

v2.3

written by Kaspar Sakmann and Axel Lode

Nov 29, 2012

Contents

Chapter 1

Main Page

OpenMCTDHB is an open-source package for the many-body dynamics of ultracold bosons [1]. It is an implementation of the MCTDHB algorithm [2] to solve the many-body Schrödinger equation for bosons. The OpenMCTDHB project was initiated by Kaspar Sakmann.

OpenMCTDHB can solve the many-boson Schrödinger equation for 1D, 2D and 3D problems at variable degree of accuracy. The current version supports up to two orbitals and thereby allows to assess the validity of any Gross-Pitaevskii computation on the many-body level. The code is portable, serial and can be compiled using open-source software only. A number of scripts that come with the code allow to analyze the physics of a computation without the need to get deeply involved. The goal of OpenMCTDHB is to make many-boson dynamics accessible to everyone.

People:

- ©2007-2012: Kaspar Sakmann (head of project), Axel Lode, Alexej Streltsov, Ofir Alon, Lorenz Cederbaum

A user manual with worked through examples and installation instructions can be found at the OpenMCTDHB website. **OpenMCTDHB** uses some of the libraries of the Heidelberg MCTDH package and comes with a copy of the FFTW library (please see the links below).

If you use **OpenMCTDHB** (or some results obtained from it) in any publication, we politely request that you cite [1] below. You may also wish to cite the original MCTDHB paper [2] or the MCTDHB package [3]

References

[1] OpenMCTDHB v2.3, K. Sakmann, Axel U. J. Lode, A. I. Streltsov, O. E. Alon, L. S. Cederbaum (2012), http://OpenMCTDHB.uni-hd.de.

[2] O. E. Alon, A. I. Streltsov, L. S. Cederbaum Phys. Rev. A 77, 033613 (2008).

[3] The MCTDHB Package, A. I. Streltsov, K. Sakmann, A. U. J. Lode, O. E. Alon, and L. S. Cederbaum, Version 2.0, Heidelberg, (2010), http://MCTDHB.uni-hd.de.

Useful links

- The OpenMCTDHB website (downloads, manual, developer guidelines, roadmap, etc.)
- The MCTDHB website
- The MCTDH website
- The FFTW library

Chapter 2

Installation

This file describes the Installation of OpenMCTDHB. Start by downloading the package from here `OpenMCTDHBv2.1.tar.gz`,

unpack it (type `tar -xvzf OpenMCTDHBv2.1.tar.gz`) and cd into the OpenMCTDHBv2.1 directory. In the following all paths will be relative to this base directory.

This User Manual can be found in the doc directory.

The source code is documented using doxygen. If you have doxygen installed (version 1.7.3), you can generate the source code documentation yourself by typing `doxygen Doxyfile-1.7.3` when you are in the doc directory. Alternatively, you can browse through the source `Code Documentation` online at the OpenMCTDHB website

2.1 SYSTEM REQUIREMENTS

Currently, OpenMCTDHB has been tested on Linux only. These installation instructions are therefore for Linux. It is assumed that you have a recent version of gcc installed, we used gcc version 4.3.2.

There are at least three different options to install OpenMCTDHB:

- the fully open source option: using gfortran of the gcc compiler suite together with BLAS, LAPACK and FFTW
- The second option uses gfortran together with the ACML library
- The third option uses pgf95 together with the ACML library

2.2 INSTALLATION gfortran with FFTW

(i) The default is to use gfortran as a compiler together with the libraries BLAS, LAPACK and FFTW. We used gfortran version 4.3.2 and FFTW version 3.2.2. A copy of the

FFTW library can be found in the src/FFTW directory, just in case you haven't got it on your machine already.

1. (i) This step assumes you haven't got FFTW 3.2.2 installed and will install FFTW into the directory src/FFTW (this is where the default makefile of OpenMCTDHB expects to find FFTW; if you don't like that install FFTW anywhere else and adjust the path in the makefile).

   (a) cd into src/FFTW and unpack FFTW (type tar −xvzf fftw−3.2.2.tar.gz).

   (b) cd into src/FFTW/fftw−3.2.2

   (c) Type ./configure −−prefix=<full path to the src/FFTW directory>

   (d) Type make (this will take a while)

   (e) Type make install

   You should now have a few more directories in src/FFTW, e.g. src/FFTW/include. If this did not work for you, start reading the README of FFTW or contact one of your system administrators. Alternatively you can try one of the other installation options.

   (ii) If you already have the FFTW library installed somewhere else and want to use it you will have to set the path to it in the makefile. It should be the path to the directory that contains the include directory of the FFTW library.

2. cd into the src directory and type make. This should compile the program and put the executable OpenMCTDHB.x in the base directory.

3. If you would like to use the separate program V_ext.x to create files containing external potentials cd into the create_V_ext_file directory and type make. This should compile the program and put the executable V_ext.x in the directory above src.

2.3 INSTALLATION gfortran with ACML 4.4

Assuming you have the ACML library installed (we used version 4.4) the installation proceeds as follows.

1. cd into the src directory

2. open the file Makefile-gfortran-acml and set the path to the ACML library

3. type echo $LD_LIBRARY_PATH, the path to your ACML library has to be in the list that will appear. If it is not, add it, e.g. by typing **export** LD_LIBRARY_PATH:<path to your gfortran acml library>

4. type make -f Makefile-gfortran-acml. This should compile the program and put the executable OpenMCTDHB.x in the base directory.

5. Proceed with step 3 of the previous installation instructions

2.4 INSTALLATION pgf95 with ACML

Assuming you have the PGI compiler pgf95 installed (we used the 64 bit version 8.0-1) the installation proceeds as follows. The compiler comes with its own copy of ACML and knows where to find it.

1. type make -f Makefile-gfortran-pgf95. This should compile the program and put the executable OpenMCTDHB.x in the base directory.
2. Proceed with step 3 of the previous installation instructions.

Chapter 3

Basic Usage

This Chapter discusses the basic usage of OpenMCTDHB It is assumed that you have successfully compiled OpenMCTDHB and that there is an executable OpenMCTDHB.x in the base directory.

3.1 Typical workflow

We will now discuss the typical workflow using OpenMCTDHB. OpenMCTDHB is designed such that you should only have to compile it once and never again, unless you intend to study particular problems with time-dependent Hamiltonians.

The typical workflow is as follows. We will illustrate it using the example BJJ−propagation−1d of the examples directory. In this particular example a purely coherent condensate of one-hundred particles resides initially in the left well of a double-well potential and subsequently tunnels through the barrier. The interaction strength is weak.

- create a directory TEST−BJJ−propagation−1d. Let's say for simplicity. in the OpenMCTDHBv2.1 directory where also the executables are.
- cd into the TEST−BJJ−propagation−1d directory and copy all the .inp files from examples/BJJ−propagation−1d/input there, i.e. the files restart.orbs.inp, restart.coef.inp, runpars.inp and V_ext.inp.
- create a symbolic link to (or copy) OpenMCTDHB.x in the directory
- type OpenMCTDHB.x
- the program should now start running.

While you are eagerly waiting for results (or after the program has finished) you can analyze the output. You will need gnuplot for the following scripts to work.

- open another shell and cd into the TEST−BJJ−propagation−1d directory. Assuming the program is still running there should be a growing number of files there.

- create a symbolic link to the scripts directory (type ln −s ../scripts)
- type scripts/1d−x−density.sh 60 to plot the one-body density at different time points. The value 60 determines the largest value on the y-axis of the plot. (the time difference between plots depends on the variable printstep in the file runpars.inp)
- type scripts/1d− plotprobabilityleft .sh to plot the probability density in the left potential well as a function of time (the temporal resolution is determined by the variable tinyStep in the runpars.inp file).
- type scripts/plotnatoccs.sh to see the natural orbital occupation numbers as a function of time
- type scripts/1d−k−density.sh 100 to plot the one-body momentum distribution at different time points. The value 100 determines the largest value on the y-axis of the plot.

These four scripts should give you an idea of what the physical system is doing. The computation will finish after 100 time units. The system has then tunneled through the barrier once. You can now compare your result files to those given in the directory examples/BJJ-propagation-1d, e.g. by typing vimdiff <file> ../examples/BJJ−propagation−1d/<file>

There should not be any significant difference between the results (somewhere in the last few digits maybe). You can try out any of the other 1d scripts in the scripts directory. on this problem. See the scripts chapter of this manual for details.

3.2 Input files

Usually, it is necessary to provide the following input files to a computation. We will illustrate this using the example, we have just discussed in the previous section.

1. runpars.inp which contains parameters of the current run
2. V_ext.inp which contains the external potential
3. restart.orbs.inp which contains the orbital(s) used to expand the wave function.
4. restart.coef.inp which contains the coefficients of the wave function.

The orbital and the coefficient files can also be left out, if a computation is to start from default orbitals (gaussians), see below. Also the external potential can be defined within the program, but this requires the recompilation of the program. It is recommended to use this option only for time-dependent external potentials.

3.2.1 The file runpars.inp

We begin with a discussion of the runpars.inp file of the BJJ−propagation−1d example.

```
&SYSTEMPARAMETERS
 LAMBDA0= 1.53535353535353531E-003,
 NPAR=          100,
 MORB=            2,
 DIMS=            1,
 USEIMEST=F,  /

&IMESTPARAMETERS
 IMESTPARAMETER1= 5.00000000000000028E-002,  /

&GRIDPARAMETERS
 XI= -12.000000000000000      ,
 XF=  12.000000000000000      ,
 YI=  0.0000000000000000      ,
 YF=  0.0000000000000000      ,
 ZI=  0.0000000000000000      ,
 ZF=  0.0000000000000000      ,
 NDX=         251,
 NDY=           1,
 NDZ=           1,  /

&PRINTPARAMETERS
 TINYSTEP= 0.10000000000000001      ,
 PRINTSTEP=  5.0000000000000000      ,
 PRINTALLSTEP=  70.331964898799995      ,
 PRINTCORRELATIONFUNCS=             0,
 REALSPACE2D = .FALSE.,
 MOMSPACE2D = .FALSE.,
 x1slice    = 0.d0,
 x2slice    = 0.d0,
 y1slice    = 0.d0,
 y2slice    = 0.d0,
 x1const    = .TRUE.,
 y1const    = .TRUE.,
 x2const    = .FALSE.,
 y2const    = .FALSE.,
 Pnot       = .FALSE.,
 xstart     = -10.d0,
 xstop     = -10.d0
/

&RUNPARAMETERS
 ABSTIME= 0.000000000000000      ,
 MAXTIME= 100.000000000000000      ,
 TOLERROR= 1.00000000000000022E-007,
 RESTART=T,
 RELAX=F,
 READPOTENTIAL=T,
 PROPDIRECTION=           1,
 TAG='BJJ-potential-1d',  /
```

The file contains several namelists, which we will now list.

- SYSTEMPARAMETERS contains information about the interaction strength LAMBDA0, the number of particles NPAR, the number of orbitals MORB, the number of dimensions of the problem DIMS, and a variable USEIMEST to decide whether the IMEST algorithm is to be used (T) to evaluate interaction matrix elements or not (F).

  If MORB=1, the MCTDHB equations reduce to the Gross-Pitaevskii equation which is then solved.

  If MORB=2, the MCTDHB equations constitute the systematic many-body general-

ization of Gross-Pitaevskii, please see the literature for details. Currently, up to two orbitals can be used in OpenMCTDHB.

If USEIMEST=F a delta function is used as an interaction potential (actually just a square box of the width of one grid point). A description of the IMEST algorithm can be found in Kaspar Sakmann's PhD thesis.

If USEIMEST=T (currently) a gaussian interaction potential is used. Other interaction potentials can be implemente dusing the IMEST algorithm, as long as the value of the interaction potential depends on the distance between two points only.

- IMESTPARAMETERS is a container for parameters in case the IMEST algorithm is used. Currently, IMESTPARAMETER1 determines the width of a gaussian interaction, that is normalized such that for vanishing width the delta-function potential is recovered.
- GRIDPARAMETERS contains the parameters that determine the lower and upper box boundaries and the number of grid points in each direction.
- PRINTPARAMETERS controls the frequency when certain files are printed.

  TINYSTEP is the time step at which quantities are printed that are to be monitored as continous functions of time, e.g., the natural orbital occupation numbers, the energy of the syste, in 1Dsystems: the Lieb-Liniger parameter and the density in the left half of space.

  PRINTSTEP determines how frequently the density of the system and its momentum distribution are plotted. Additionally, the files restart.coef.**out** and restart.orbs.**out** are printed out, which can be used as new restart.coef.inp and restart.orbs.inp files in case a computation was interrupted or finished early. These files are overwritten after every PRINTSTEP time units.

  PRINTALLSTEP determines how frequently the coefficients and the orbitals are written into files that are kept, as well as the computation of correlation functions, in case PRINTCORRELATIONS=1.

  PRINTCORRELATIONS determines whether correlation functions are to be computed. Currently, this works only for one-dimensional problems. Possible values: 0,1.

  REALSPACE2D determines whether the output of cuts through the one-body and two-body correlation functions in real space for 2 dimensional calculations is activated.

  MOMSPACE2D determines whether the output of cuts through the one-body and two-body correlation functions in momentum space for 2 dimensional calculations is activated.

  x1slice,y1slice,x2slice,y2slice determine at which values the cuts of the correlation functions $g^{(1)}(\vec{r}_1'|\vec{r}_1) = g^{(1)}([x1slice, y1slice]^T|[x2slice, y2slice]^T)$ or $g^{(2)}(\vec{r}_1' = \vec{r}_1, \vec{r}_2' = \vec{r}_2|\vec{r}_1, \vec{r}_2) = g^{(2)}(\vec{r}_1, \vec{r}_2) = g^{(2)}([x1slice, y1slice]^T, [x2slice, y2slice]^T)$ are written for 2D computations. Note that only two of these variables are adressed, see explanation for x1const, etc.. Depending on the value of MOMSPACE2D and REALSPACE2D the variables x1slice,y1slice,x2slice,y2slice are interpreted as momenta or positions.

  x1const,y1const,x2const,y2const If MOMSPACE2D and/or REALSPACE2D is true then exactly two of these variables have to be true, too. These two variables then determine which cuts are taken, e.g., x1const=.TRUE.,y1const=.TRUE.,x1slice=0.d0,y1slice=0.d0 will write the correlations $g^{(1)}([0.d0, 0.d0]^T|[x2, y2]^T)$ and $g^{(2)}([0.d0, 0.d0]^T, [x2, y2]^T)$.

Pnot is a logical variable which determines whether to write the nonescape probability for 1D calculations or not. It is defined as an integral on the diagonal of the reduced one-body density: $P_{not} = \int_{xstart}^{xstop} \rho(x)dx$.

xstart ,xstop are real values which determine the evaluation of the nonescape probability.

- RUNPARAMETERS determines the several parameters for this particular run.

  ABSTIME the initial time

  MAXTIME the maximal time

  TOLERROR the maximal error allowed in each step of the propagation. This is given as a parameter to the integrators for the coefficients and the oribitals.

  RESTART can be T or F; if true, the files restart.orbs.inp and restart.coef.inp are read in, otherwise the computation will start from some default orbitals (which can be useful for relaxation computations).

  RELAX can be T or F: if T the system is propagated in imaginary time in order to reach the ground state (if PROPDIRECTION=1). Otherwise propagated in real time.

  READPOTENTIAL can be T or F: if T, the program will read the potential from the input file V_ext.inp, if F, the external potential defined in the subroutine get_V_ext will be used. This latter option also allows to use time-dependent external potentials.

  PROPDIRECTION can be 1 or -1; if 1 and RELAX=F the Schrödinger equation is propagated forward in time; if -1 and RELAX=F the Schrödinger equation is propagated backward in time

  TAG is a user defined string that will be attached to all output files. Allows to give computations meaningful names.

3.2.2 The file V_ext.inp

The file V_ext.inp contains the external potential that is used in the computation. Its format is the same for 1D, 2D and 3D problems. The first three columns contain the values of x,y and z, while the fourth column contains the corresponding value V(x,y,z). x is the fast running variable, i.e., y is increased, only once x has run from the first to last grid point in the x-direction. Similarly, z is increased, once y has run once from its first to the last grid point. You can create the V_ext.inp file using the program V_ext.x, see below.

3.2.3 The file restart.orbs.inp

The file restart.orbs.inp contains the orbitals of the wave function. Its format is the same for 1D, 2D and 3D problems. The first three columns contain the values of x,y and z while the fourth column contains the corresponding value V(x,y,z) The next two columns contain the real and imaginary parts of the first orbital, and if MORB=2 the next two columns the respective values of the second orbital.

3.2.4 The file restart.coef.inp

The file restart.coef.inp contains three columns. The first column contains the index of a configuration, whereas the second and third column contain the real and imaginary part of the corresponding coefficient.

3.3 How to setup a new problem

It is now assumed that you have a working version of OpenMCTDHB installed, that reproduces the results of the examples directory. We will now discuss a convenient way to set up a new problem, assuming that the Hamiltonian of the problem is time-independent.

Let's say you would like to study the many-body ground state in a harmonic trap in 1-dimension. In particular, you would like to know whether the ground state of 100 bosons interacting via a delta-function potential at a given interaction strength is of the Gross-Pitaevskii type or not.

3.3.1 Get the external potential

The first thing you will have to do is to create a file containing the harmonic potential, called V_ext.inp. You can either generate this file yourself, e.g. using $MatLab$ or you can use the little program V_ext.x for this purpose.

Currently, this program allows you to generate V_ext.inp files only for two types of potentials, a 1D bosonic Josephson-Junction potential and parametrized harmonic potentials in 1D, 2D and 3D. However, it is not difficult to extend it to other potentials. You will have to edit the files src/mod/module_create_V_ext.F90 and src/create_V_ext/V_ext.F90 for this purpose.

However, to generate a 2D harmonic potential you only have to do the following.

1. cd into the directory input/HarmonicTrap−1D and open the file V_ext.nml
2. In this file you can specify the number of grid points, the boundaries of the grid, the zero of the potential and the trap frequencies in each direction. You can give the potential a meaningful name, here, e.g. 'Harmonic-2d-64x64'.
3. copy (or create a sym link to) V_ext.x in the current directory and type V_ext.x. This should give you two files containing the potential, called V_extHarmonic−2d−64x64.inp and V_extHarmonic−2d−64x64.nml which contains the parameters you just entered.

Here, is a listing of a file that generates a 2D harmonic trap located at the origin with unit trap frequencies in x and y.

```
&GENERALPARS
  XI= −6.0000000000000000      ,
  XF=  6.0000000000000000      ,
  YI= −6.0000000000000000      ,
  YF=  6.0000000000000000      ,
  ZI=  0.0000000000000000      ,
```

```
 ZF=   0.0000000000000000       ,
 NDX=             64,
 NDY=             64,
 NDZ=              1,
 DIMS=             2,   /

&TAGNAME
 TAG='Harmonic-2d-64x64',   /

&POTENTIALNAME
 POTNAME='HARMONIC',   /

&HARMONICPARAMETERS
 X0=   0.0000000000000000       ,
 Y0=   0.0000000000000000       ,
 Z0=   0.0000000000000000       ,
 W_X=  1.00000000000000000       ,
 W_Y=  1.00000000000000000       ,
 W_Z=  1.00000000000000000       ,   /
```

You can now copy the file with the external potential into any run-directory and use it. When you do so, make sure to rename it to V_ext.inp, because that's what the propram will look for.

If you choose POTNAME='BJJ' you will get a bosonic Josephson-Junction potential. Feel free to add other external potentials.

3.3.2 Relaxation starting from the default initial guess wave function

In the previous subsection you have generated a 2D external harmonic potential. You can now use this potential together with a runpars.inp file to relax the system to the ground state, given you have some input files for the orbitals and the coefficients. Here is a listing of such a file for a two orbital computation taken from the examples/HarmonicTrap-2d directory:

```
&SYSTEMPARAMETERS
 LAMBDA0=       0.01,
 NPAR=           100,
 MORB=             2,
 DIMS=             2,
 USEIMEST=F,   /

&IMESTPARAMETERS
 IMESTPARAMETER1= 5.00000000000000028E-002,   /

&GRIDPARAMETERS
 XI= -6.000000000000000       ,
 XF=  6.000000000000000       ,
 YI= -6.0000000000000000       ,
 YF=  6.0000000000000000       ,
 ZI=  0.0000000000000000       ,
 ZF=  0.0000000000000000       ,
 NDX=          64,
 NDY=          64,
 NDZ=           1,   /

&PRINTPARAMETERS
 TINYSTEP= 0.10000000000000001       ,
 PRINTSTEP=  5.0000000000000000       ,
```

```
 PRINTALLSTEP= 100.0000      ,
 PRINTCORRELATIONFUNCS=              0,   /

&RUNPARAMETERS
 ABSTIME= 0.000000000000000      ,
 MAXTIME= 100.000000000000000      ,
 TOLERROR= 1.00000000000000022E-008,
 RESTART=T,
 RELAX=T,
 READPOTENTIAL=T,
 PROPDIRECTION=              1,
 TAG='HarmonicTrap-2d-GS',   /
```

It is recommended to use the files restart.orbs_MORB2.**out**, restart.coef_MORB2.**out** as input files for the two orbital computation and the files restart.orbs_MORB1.**out**, restart.coef_MORB1.**out** for the one orbital computation. These files already contain the solution to the respective problems.

However, you do not necessarily have to provide any orbitals or coefficients if you set RESTART=F in runpars.inp. The program will then start from default orbitals.

If you decide to do so, it may be necessary to start with a low error tolerance, e.g. TOLERROR=1.0E-2, and propagate the system in imaginary time until the energy of the system does not decrease any longer in the first few digits (monitor the .dat file containing the energy). Then set RESTART=T and increase the error tolerance in the runpars.inp file. Also, don't forget to make the files restart.orbs.**out** and restart.coef.**out** the new input files for the orbitals and the coefficients, i.e. replace the suffix .out by .inp.

It turns out that the ground state at the given interaction strength is to a very good approximation of the Gross-Pitaevskii type. To minimize the size of the directory only some of the output files were saved.

As expected, the energy of the two orbital computation is below the Gross-Pitaevskii result. This can be seen by comparing the two files energyN100M1HarmonicTrap-2d-GS.dat and energyN100M2HarmonicTrap-2d-GS.dat.

You can also plot the densities of the two solutions, using the script 2d-density.sh. Please see the scripts section of this manual.

This completes our second worked through example. The current manual has hopefully helped the new user to get started with with OpenMCTDHB. It is also a good idea to read through the code documentation on the OpenMCTDHB website.

Chapter 4

Scripts

This Chapter documents the scripts in the scripts directory. As a general advice, it is a good idea to create a symbolic link (ln -s <path to the scripts directory>) to the scripts directory in any directory in which a computation was carried out. gnuplot is used as the plotting program. Make sure you have a reasonably new version of it. Square brackets indicate optional arguments.

4.1 Scripts for any number of dimensions

- plotnoccs.sh
  plots the natural orbital occupation numbers as a function of time.

4.2 Scripts for 1d computations

- 1d-x-density.sh [maximal y value]
  Collects all files ending on x-density.dat and plots them one after the other. The one-body densities of the system.
- 1d-k-density.sh [maximal y value]
  Collects all files ending on k-density.dat and plots them one after the other. The one-body momentum distributions of the system.
- 1d-x-densitydynamics.sh
  Collects all files ending on x-density.dat, concatenates them and plots them in a 2D plot. Allows to visualize the time-evolution the one-body density of the system.
- 1d-k-densitydynamics.sh
  Collects all files ending on k-density.dat, concatenates them and plots them in a 2D plot. Allows to visualize the time-evolution of one-body momentum distribution of the system.
- 1d-g1x.sh
  Takes the last file ending on x-correlations.dat and plots the absolute value squared

of Glauber's normalized first order correlation function $|g^{(1)}(x,x')|^2$. Allows to visualize the degree of first order coherence.

- 1d-g2x.sh
  Takes the last file ending on x-correlations.dat and plots the diagonal of Glauber's normalized second order correlation function $g^{(2)}(x,x',x',x)$. Gives a measure for the degree of second order coherence in x-space.

- 1d-g2k.sh
  Takes the last file ending on k-correlations.dat and plots the diagonal of Glauber's normalized second order correlation function $g^{(2)}(k,k',k',k)$. Gives a measure for the degree of second order coherence in k-space.

- 1d-G2x.sh
  Takes the last file ending on x-correlations.dat and plots the diagonal of Glauber's second order correlation function $G^{(2)}(x,x',x',x)$. This is the density-density correlation function.

- 1d-G2k.sh
  Takes the last file ending on k-correlations.dat and plots the diagonal of Glauber's second order correlation function $G^{(2)}(k,k',k',k)$. This is the momentum-momentum correlation function.

- 1d-plotprobabilityleft.sh
  Plots the probability density in the left half of space, i.e., the integral over the one-body density from $-\infty$ up to zero.

4.3 Scripts for 2d computations

- 2d-density.sh
  Collects all files ending on x-density.dat and plots them one after the other. Allows to visualize the density in 2 dimensions.

Chapter 5

Developer Guidelines for OpenMCTDHB

These OpenMCTDHB developer guidelines are largely based on the “Standards, Guidelines and Recommendations for writing Fortran 95 codes" of S.-A. Boukabara and P. van Delst, and are tailored specifically to OpenMCTDHB.

Prospective developers: if you would like to contribute to OpenMCTDHB please contact the developers.

Current developers: it is likely that very soon there will be an official roadmap of OpenMCTDHB. However, right now the roadmap is only very loosely defined. Therefore, please coordinate with the other developers before writing any new code.

5.1 Introduction:

The purpose of this document is to ensure that new Fortran95 additions to OpenMCTDHB will be as portable and robust as possible, as well as consistent throughout the system. It builds upon experience to avoid error-prone practices and gathers guidelines that are known to make codes more robust. This document covers items in order of decreasing importance (see below).

- Standards: Aimed at ensuring portability, readability and robustness. Compliance with this category is mandatory. Identified by the (**) sign.
- Guidelines: Good practices. Compliance with this category is strongly encouraged. The case for deviations will need to be argued by the programmer. Identified by the (*) sign.
- Recommendations: Compliance with this category is optional, but is encouraged for consistency purposes. No sign for these items.

Please adhere to the standards religiously. This document also lists a set of Fortran features that are not to be used in any new code. These features are known to be error-prone, difficult to maintain/read or listed as obsolescent in the Fortran95 standard. In many cases, newer Fortran95 constructs provide the same functionality in a more readable and robust fashion.

5.2 Encouraged Compilation Practices:

- To increase robustness, it is recommended to compile the code with as many compilers as available, to minimize the errors that may be missed by using a single compiler.
- If available, the use of the -ISO compiler option (International Standardization Organization) is encouraged, to ensure that no extension is used within the code.
- All parts of the code should be compiled using the 'all warnings' option. The only exception are the files in the lib directory which are taken from MCTDH.

5.3 Encouraged Features

These usually help in the robustness of the code (by checking interface compatibility for example) and in the readability, maintainability and portability. They are reminded here:

- Encapsulation: Use of modules for procedures, functions, data.
- Dynamic Memory allocation for optimal memory usage.
- Derived types or structures which generally lead to stable interfaces, optimal memory usage, compactness, etc.
- Optional and keyword arguments in using routines.
- Functions/subroutines overloading capability (however, do not overload operators in OpenMCTDHB)
- Intrinsinc functions: bits, arrays manipulations, kinds definitions, etc.

5.4 Portability / Inter-systems Compatibility

Standards:

- (**) Source code shall conform to the ISO Fortran95 standard.
- (**) No compiler- or platform-dependent extensions shall be used.
- (**) While use of IOSTAT and STAT is encouraged to handle I/O and other status errors, no interpretation of actual returned values shall be made, as these values are compiler-dependent.

Guidelines:

- (*) Note that STOP is a F90/95 standard. EXIT(N) is an extension and should be avoided. It is recognized that STOP does not necessarily return an error code. If an error code must be passed to a script for instance, then the extension EXIT could be used but within a central place, so that to limit its occurrences within the code to a single place.
- (*) Precision: Parameterizations should not rely on vendor-supplied flags to supply a default floating point precision or integer size. The F90/95 KIND feature should be used instead.
- (*) Do not use tab characters in the code to ensure it will look as intended when ported. They are not part of the Fortran characters set. Recommendations:
- For applications requiring interaction with independently-developed frameworks, the use of KIND type for all variables declaration is encouraged to facilitate the integration.

5.5 Readability / Maintainability

Standards:

- (**) Use free format syntax. All new code must be written in free format Fortran 95 and all new files must have the .F90 extension, which means that they will also be preprocessed.
- (**) Use consistent indentation across the code. Indent everything by two spaces. See the file OpenMCTDHB.F90 as a reference. Also the comments should be indented this way (this still needs some work).
- (**) Use the Fortran 90 '!' to comment something. Never use the old 'C'
- (**) Comments that are to enter the Doxygen documentation must start with either !> for subroutine documentation or !< if dummy variables are documented inline. See , e.g. the module modulegrid.F90. IMPORTANT: Doxygen is case sensitive. This means that you will have to make sure that all variables and subroutines that you write must always be written the same way. Otherwise, you will mess up the Doxygen documentation. This is particularly important for subroutines. Check this before you commit.
- (**) Use modules to organize source code. Only one module per file. Modules must have the same (lower case only) name as the file that they are in and start with with the word 'module'. All subroutines must be in modules.
- (**) Use meaningful, understandable words for variables and parameters. Do not abbreviate these words. and concatenate words to camelCase notation (first word lower case, all following start with an upper case letter, e.g. printLiebLiniger).
- (**) Subroutines should start with a verb (preferably one of get, write or read) followed by an underscore and a variable, e.g. get_liebLinigerParameter.

- (**) Each function, subroutine has to contain a Doxygen friendly header, explaining the purpose, the author and the date when it was first introduced into the code. When modifying existing code the name of the modifying developer should be added to the list of authors together with the modification made.
- (**) Preprocessor macros should be used on the lowest level only to allow for different libraries to be used, e.g. for calls to different FFT libraries in modulefft.F90.
- (**) Do not use GOTO statements. These are hard to maintain and complicate understanding the code.
- (**) Remove code that was used for debugging once this is complete.
- (**) Always use the :: notation, even if there are no attributes.
- (**) Line up vertically: attributes, variables, comments within the variables declaration section.
- (**) Remove unused variables and unused code

Guidelines:

- (*) Use construct names to name loops, to increase readability, especially in nested loops.
- (*) Similarly, use construct names in subroutines, functions, main programs, modules, operator, interface, etc.
- (*) Include comments to describe the input, output and local variables of all procedures.
- (*) Use comments as required to delineate significant functional sections of code.
- (*) Do not use FORTRAN statements and intrinsinc function names as symbolic names.
- (*) Use named parameters instead of mmagic numbers ; REAL, PARAMETER :: PI=3.14159

Recommendations:

- When writing new code, adhere to the style standards of the existing code to keep consistency.
- Use the same indentation for comments as for the rest of the code.
- Functions, procedures, data that are naturally linked should be grouped in modules.
- Limit to 80 the number of characters per line (maximum allowed under ISO is 132)
- Use of operators $<, >, <=, >=, ==, / =$ is encouraged (for readability) instead of .lt., .gt., .le., .ge., .eq., .ne.
- Use blanks to improve the appearance of the code, to separate syntactic elements (on either side of equal signs, etc) in type declaration statements

5.6 Robustness

Standards:

- (**) Use IMPLICIT NONE in all codes: main programs, modules, etc. To ensure correct size and type declarations of variables/arrays.
- (**) All subroutines and functions MUST be in modules.
- (**) Write subroutines instead of copying code (even if it appears just twice). Also, put even few lines of code that do something specific into a subroutine (with a suitable name).
- (**) Use PRIVATE in modules before explicitly listing data, functions, procedures to be PUBLIC. This ensures encapsulation of modules and avoids potential naming conflicts. Exception to previous statement is when a module is entirely dedicated to PUBLIC data/functions (e.g. a module dedicated to constants).
- (**) Initialize all variables. Do not assume machine default value assignments.
- (**) Currently the code is purely procedural and no derived data types are used. Before using object-oriented features contact all other developers.
- (**) Do not initialize variables of one type with values of another.
- (**) Do not use the operators $==$ and $/=$ with floating-point expressions as operands. Check instead the departure of the difference from a pre-defined numerical accuracy threshold (e.g. epsilon comparison).
- (**) Avoid to share data via use association. Large arrays, such as V_{ext} should be passed explicitly wherever possible. The structure of the integrators makes this complicated at the moment, but the general rule is that it has to be clear what happens and what variables are modified when calling a routine by reading through the main program.
- (**) Use explicit shape dummy arrays, i.e. pass the sizes of all arrays to the subroutines. Try to minimize use association of variables to maximize modularity of the code.

Guidelines:

- (*) In mixed mode expressions and assignments (where variables of different types are mixed within an expression or in an assignment), type conversions should be written explicitly (not assumed). Do not compare expressions of different types for instance. Explicitly perform the type conversion first.
- (*) No include files should be used. Use modules instead, with USE statements in calling programs.
- (*) Structures (derived types) should be defined within their own module. Procedures, Functions to manipulate these structures should also be defined within this module, to form an object-like entity.

- (*) Procedures should be logically flat (should focus on a particular functionality, not several ones)
- (*) Module PUBLIC variables (global variables) should be used with care and mostly for static or infrequently varying data.

Recommendations:

- Use parentheses at all times to control evaluation order in expressions.
- Use of structures is encouraged for a more stable interface and a more compact design. Refer to structure contents with the

5.7 Vectors and Arrays

Standards:

- (**) Subscript expressions should be of type integer only.
- (**) When arrays are passed as arguments, code should not assume any particular passing mechanism.

Guidelines:

- (*) Use of arrays is encouraged as well as intrinsinc functions to manipulate them.
- (*) Use of assumed shapes is fine in passing vectors/arrays to functions/arrays.

Recommendations:

- Declare DIMENSION for all non-scalars

5.8 Dynamic Memory Allocation / Pointers

Standards:

- (**) Generally there is no need for pointers in the current code.
- (**) Use of allocatable arrays is preferred to using pointers, when possible. To minimize risks of memory leaks and heap fragmentation.
- (**) Use of pointers is allowed when declaring an array in a subroutine and making it available to a calling program.
- (**) Always initialize pointer variables in their declaration statement using the NULL() intrinsinc. INTEGER, POINTER :: x=> NULL()

Guidelines:

- (*) Always deallocate allocated pointers and arrays. This is especially important inside subroutines and inside loops.
- (*) Always test the success of a dynamic memory allocation and deallocation.

Recommendations:

- Use of dynamic memory allocation is encouraged, when sensible, e.g. for input dependent variables. It makes code generic and avoids declaring with maximum dimensions.
- For simplicity, use Automatic arrays in subroutines whenever possible, instead of allocatable arrays.

5.9 Loops

Standards:

- (**) Do not use GO TO to exit/cycle loops, use instead EXIT or CYCLE statements.
- (**) Do not number DO loops (DO 10 ...10 CONTINUE), instead name (long) loops, etc (eg: readif: IF (...) THEN .....)

5.10 Functions/Procedures

Standards:

- (**) Use the save declaration where appropriate. Do not assume the value of the variable will be kept by the processor.
- (**) Do not use an entry in a function subprogram.
- (**) Functions must not have pointer results.
- (**) All dummy arguments, except pointers, must include the INTENT clause in their declaration.
- (**) Error conditions. When an error condition occurs inside a function/procedure, a message describing what went wrong should be printed. The name of the routine in which the error occurred must be included. It is acceptable to terminate execution within a package, but the developer may instead wish to return an error flag through the argument list.
- (**) Functions/procedures that perform the same function but for different types/-sizes of arguments, should be overloaded, to minimize duplication and ease the maintainability.
- (**) When explicit interfaces are needed, use modules, or contain the subroutines in the calling programs (through CONTAINS statement), for simplicity.

Guidelines:

- (*) Do not use external routines (subroutine not contained within a module and not within the CONTAINS statement of the main program) as in some cases, these functions need interface blocks that would need to be updated each time the interface of the external routine is changed. An exception is made for the external routines that are passed to the integrators (because they were written like that a long time ago)

5.11 Inputs/Outputs

Standards:

- (**) I/O statements on external files should contain the status specifier parameters err=, end=, iostat=, as appropriate.

Recommendations:

- Use write rather than print statements for non-terminal I/O.
- Use Character parameters or explicit format specifiers inside the Read or Write statement. DO not use labeled format statements (outdated).

5.12 Fortran Features that are obsolescent and/or discouraged:

Standards:

- (**) No Common blocks. Modules are a better way to declare/store static data, with the added ability to mix data of various types, and to limit access to contained variables through use of the ONLY and PRIVATE clauses.
- (**) No GO TOs - use the CASE construct instead
- (**) No arithmetic IF statements - use the block IF construct instead
- (**) No implicit changing of the shape of an array when passing it into a subroutine. Use the RESHAPE intrinsic. Although actually forbidden in the standard it was very common practice in FORTRAN 77 to pass 'n' dimensional arrays into a subroutine where they would, say, be treated as a 1 dimensional array. This practice, though banned in Fortran 90, is still possible with external routines for which no Interface block has been supplied. This only works because of assumptions made about how the data is stored.

Guidelines:

- (*) Do not make use of the equivalence statement, especially for variables of different types. Use pointers or derived types instead.

5.13 Repository etiquette

Standards:

- (**) Before submitting: make a test directory and run OpenMCTDHB using the inputs of the examples directory. Then check whether you get the same results. Please do not alter or add any files in exisiting examples directories. They are intended for reference only.
- (**) Please take the time to document your commits sufficiently. Remember that other people really have no idea about what you have been doing.