LBNL-AMO-MCTDHF

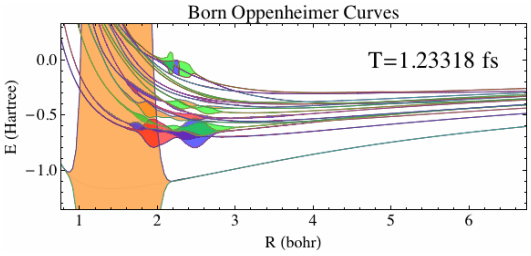
Multiconfiguration Time-Dependent Hartree Fock

for ultrafast electronic and nonadiabatic nuclear dynamics

of atoms and molecules in intense laser fields

Version 1.07

Atomic / Diatomic Vibronic / Polyatomic Fixed-Nuclei



D. J. Haxton C. W. McCurdy T. N. Rescigno K. V. Lawler J. Jones B. Abeln X. Li

LBNL-AMO-MCTDHF is a suite of codes for Multiconfiguration Time-Dependent Hartree-Fock applied to ultrafast laser dynamics of atoms and molecules. It calculates nonadiabatic electronic and nuclear wave functions for the nonrelativistic Schrödinger equation. Currently it uses the dipole approximation in length and velocity gauge and has options for a variety of laser pulses. It contains analysis and output routines and auxillary scripts for calculating absorption and stimulated emission, populations, total and partial photoionization, wave mixing, and other capabilities. It supports

- Electronic wave functions for atoms (`chmctdhf_atom`)
- Vibronic wave functions for diatoms using prolate coordinates (`chmctdhf_diatom`)
- Fixed nuclei wave functions for polyatomics without stretching or complex scaling (`chmctdhf_sinc`)

Future versions will support

- Electronic wave functions for atoms (`chmctdhf_atom`)
- Rovibronic wave functions for diatoms using prolate coordinates (`chmctdhf_diatom`)
- Rovibronic wave functions for diatoms using modified prolate coordinates (`chmctdhf_diatom2`)
- Vibronic wave functions for polyatomics and general Cartesian treatment (`chmctdhf_sinc`)

The method and equations are described in

- *MCTDHF treatment of electronic and nuclear dynamics in diatomic molecules.* D. J. Haxton, K. V. Lawler, C. W. McCurdy, Phys. Rev. A **83**, 063416 (2011)
- *Two methods for restricted configuration spaces within the MCTDHF method.* D. J. Haxton and C. W. McCurdy, Phys. Rev. A **91**, 012509 (2015)

Some of the largest nuts and bolts are due to

- EXPOKIT. *A Software Package for Computing Matrix Exponentials.* R. B. Sidje, ACM Transactions on Mathematical Software 24, 130 (1998)
- *Adiabatic formulation of heteronuclear hydrogen molecular ion.* B. D. Esry and H. R. Sadeghpour, Phys. Rev. A 60, 3604 (1999)
- *Solving the three-body Coulomb breakup problem using exterior complex scaling.* C. W. McCurdy, M. Baertschy, T. N. Rescigno, J. Phys. B. **37**, R137 (2004)
- *Grid-based methods for diatomic quantum scattering problems: A finite-element discrete-variable representation in prolate spheroidal coordinates.* L. Tao, C. W. McCurdy, T. N. Rescigno, Phys. Rev. A 79, 012719 (2009)
- *An efficient basis set representation for calculating electrons in molecules.* J. R. Jones, F.-H. Rouet, K. V. Lawler, E. Vecharynski, K. Ibrahim, S. Williams, B. Abeln, C. Yang, D. J. Haxton, C. W. McCurdy, X. Li, T. N. Rescigno, submitted.

The original open-source MCTDH code of H.-D. Meyer *et al.* at Heidelberg has driven our interest in MCTDHF and guided this numerical implementation directly, providing the foundation for the nomenclature and notation used here and in our articles in press, the input style, the variable names, and the layout of this manual. Reviews and guides available at the MCTDH website may help fill in the blanks.

- *An efficient & robust integration scheme for the equations of motion of the multiconfiguration time-dependent Hartree (MCTDH) method.* M. H. Beck and H.-D. Meyer, Z. Phys. D **42**, 113 (1997)
- *The multiconfiguration time-dependent Hartree method: A highly efficient algorithm for propagating wavepackets.* M. H. Beck, A. Jäckle, G. A. Worth, H.-D. Meyer, Physics Reports **324**, 1 (2000)

LBNL-AMO-MCTDHF has been funded primarily (2013-2018) by the United States Department of Energy Early Career Program, WAS#KC/CH57/5, and entirely under U.S. D.O.E. contracts #DE-FG02-94ER14413, #DE-AC02-05CH11231, and #DE-SC-0007182.

This project is dedicated to the memory of Donovan Emil Smith.

Contents

Preliminaries

0.1 Git install

The code is distributed through GitHub.com. Please visit

`http://commons.lbl.gov/display/csd/LBNL-AMO-MCTDHF`

to request access to the code. Send us a message via this website. We will receive the message and you will be added to Team Users on the LBNL-AMO-MCTDHF GitHub organization. You will get an email invitation when this happens. Then you can view the repository on your browser

`https://github.com/LBNL-AMO-MCTDHF/V1`

and perform a git clone to obtain the code.

```
prompt> git clone https://github.com/LBNL-AMO-MCTDHF/V1 master
```

Instead of git clone, you can download the zipfile

`https://github.com/LBNL-AMO-MCTDHF/V1/archive/master.zip`

The `README` that is visible on the website and included with the code is included below, and includes quick installation instructions for mac.

```
  ** LBNL-AMO-MCTDHF VERSION 1.07 **

Please watch the commits to stay notified about bugs and bug fixes:
     https://github.com/LBNL-AMO-MCTDHF/V1/commits/master

Known bugs appear as issues:
     https://github.com/LBNL-AMO-MCTDHF/V1/issues

The scripts in the example directories use bash and gnuplot.

Using the bash shell is recommended but may not be necessary.
To begin a bash shell, if it is not your default, simply execute the command
   prompt> bash
To set bash as your default shell on mac or other linux, execute the command
   prompt> chsh -s /bin/bash

Lots of scripts use gnuplot.  Gnuplot should be installed on your system,
if you want to have an easy time seeing the results of the example calculations.
If gnuplot is installed on your system then the command
   prompt> which gnuplot
should return the location of the executable file.  If it returns nothing then
it is not installed.  The best way to install gnuplot is probably with macports.

Possible workflow described below for mac.  The general idea is that you can
   work entirely within the directory that you clone and to which you pull, not
   copy the git distribution elsewhere for compilation.

--------------------------------------------------------------------------------

On Mac: minimal demonstration.  BIN*mac* directories use GFORTRAN.  You must have
gfortran installed to use the BIN*mac* directories.  To install gfortran -- which
you will need to do if the Makeme step fails -- visit

     https://gcc.gnu.org/wiki/GFortranBinariesMacOS

0)   git clone https://github.com/LBNL-AMO-MCTDHF/V1 master

          or download using a web browser

     https://github.com/LBNL-AMO-MCTDHF/V1/archive/master.zip
          and unzip this file.

     You then have a V1 or V1-master directory with the code in it.  change
     to that directory:

     cd ./V1
         or
     cd ./V1-master
```

```
     and PERFORM ALL OTHER STEPS IN THIS V1 DIRECTORY.

1)   cd COMPDIRS/BIN.ecs.hermnorm.mac
     ./Makeme

     if this fails, you need gfortran.  Also you may then

     cd ../debug.BIN.ecs.hermnorm.mac
     ./Makeme
     cd ../BIN.ecs.hermnorm.mac.mpi
     ./Makeme
     cd ../debug.BIN.ecs.hermnorm.mac.mpi
     ./Makeme

     You can delete the directories in COMPDIRS that you don't want.

2)   You need to have the code in your path.  Safe, temporary way:

2A)  cd COMPDIRS/BIN.ecs.hermorm.mac
     export PATH=$PATH:$PWD

2B)  Permanent way, system-wide, links in /opt/local/bin, requires root password:
     cd COMPDIRS/BIN.ecs.hermorm.mac
     ./LinkMe
     cd ../debug.BIN.ecs.hermorm.mac
     ./LinkMe debug
     cd ../BIN.ecs.hermorm.mac.mpi
     ./LinkMe mpi
     cd ../debug.BIN.ecs.hermorm.mac.mpi
     ./LinkMe mpi.debug

2C)  Permanent way, user-specific, script that edits your .bashrc file
     cd COMPDIRS/BIN.ecs.hermnorm.mac
     ./LinkMeLocal

     You must restart the terminal after step 2C.

3)   mkdir EXAMPLES          (you can also make other directories e.g.
     mkdir MYRUNS            )
     cp -R -p EXAMPLES-depot/H2PHOTO EXAMPLES

4)   cd EXAMPLES/H2PHOTO
     ./Relax.Bat
     ./Pulse.Bat
     ./Fourier.Bat
     ./Flux.Bat 500
     ./gnu.xsec
```

```
     as per the README in that H2PHOTO directory.  Also see the H2ABSORPTION
     example directory to start.

If you want to update the code and EXAMPLES-depot directories then perform
step 5, like steps 1-4, in the V1 working directory,

5)   git pull https://github.com/LBNL-AMO-MCTDHF/V1 master
     ... then ./Makeme should be sufficient, in the BIN directories,
         but ./Makeme clean; ./Makeme would certainly be fail safe
         after updating the code with git pull.

----------------------------    MODULES TO LOAD    -----------------------------
---------------------------- ON NERSC & LAWRENCIUM -----------------------------

  COMPDIRS/BIN.ecs.hermnorm.law,
  COMPDIRS/BIN.ecs.hermnorm.edison, etc.

LAWRENCIUM:

module load mkl

CARVER:

module unload pgi openmpi
module load intel openmpi-intel mkl
module unload torque
module load torque

HOPPER:

module swap PrgEnv-pgi PrgEnv-intel
#yes last
module unload cray-libsci

EDISON:

module unload cray-libsci
module load mkl
```

0.2 Running the code

The code looks for an `Input.Inp` input file. If another input file is desired, use e.g.

```
> chmctdh_atom Inp=Input.Inp.myinput
```

The code will perform a calculation with default parameters, if no input is given.

0.3 File system etiquette

See also section 5.3.1 and note script `findwork` at top level in the distribution.

If they do not exist in the working directory, the code makes the subdirectories

```
WALKS
Flux
Dat
Bin
timing
```

For more information on `WALKS` see section 3, Output. For more information on `Flux`, see 5.3.1. `Bin` is generally used to save the wave function, and generally `Dat` contains numerical text output from actions, section 2.5. Timing output, what is discussed in section **??**, is sent to directory `timing` if `&parinp` namelist variable `notiming` is set less than the value 2, its default, and is described in section **??**.

Some large files may end up in these directories. Beware of this. Again, see also section 5.3.1 and note script `findwork` at top level in the distribution. On a supercomputer e.g. NERSC machines, you **MUST** be reading and writing these files to the `$SCRATCH` directory. One could run on the axl condo cluster home file system `/clusterfs/axl` but make **symbolic links** to the scratch directory. E.g.

```
> mkdir $SCRATCH/blah5
> ln -s $SCRATCH/blah5 ./WALKS
```

0.4 Run Control

A running calculation can be stopped by creating a file named `stop` in its working directory.

0.5 Changes from version 0

- There is a polyatomic version under development and compiled under `chmctdhf_sinc`. Namelist `&sincparinp`.
  - This version is MPI parallelized over orbitals (totally). Set both `parorbsplit=3` in namelist `&parinp` and `orbparflag=.true.` in namelist `&sinc_params` to do this. For the other versions of the code (`chmctdhf_atom` and `chmctdhf_diatom`) option `parorbsplit=3` is not supported and will result in an error. If these options are set, `numpoints` needs to be divisible by the number of processors.
  - Velocity gauge was just implemented and barely tested; the factors might be right, but probably not.
  - Only cubic grids are supported. There is only one variable, `numpoints`, number of points on a side.

  - You can read in orbitals computed at double the grid spacing (e.g. `spacing=1.0d0`) for a calculation at the current grid spacing in the input file (e.g. `spacing=0.5d0`) by setting `reinterp_orbflag=1` in namelist `&parinp`. **This is very useful** if you are trying to get an initial state eigenfunction for a very large calculation. Starting with core orbitals takes too long, so you want to reinterpolate a prior solution for the orbitals onto the finer grid. You can now do this, but only reinterpolating on a grid with $\frac{1}{2}$ the grid spacing as the first. See example directories such as `EXAMPLES/HELIUM-INTERPTEST3`.
  - There are options for a complex absorbing potential, CAP.
  - There is also an attempt at smooth complex scaling, now, for `chmcthdf_sinc`, in which the x, y, and z coodinates are separately scaled; you can make the coordinates real at the nuclei. It does not perform well, however.

- `mcscfflag` is deprecated; there is no "mcscf mode." There is still `mcscfnum`, number of A-vectors relaxed or propagated. **You can propagate multiple A-vectors** and this may be important for some cases ... will discuss. No `mcscf.bin` is produced; everything is in `spfs.bin` and `avector.bin`.

- There is the option `littlesteps` to divide the A-vector propagation over the mean field timestep (`par_timestep`). This can be used to optimize for speed. It is needed if you want to to try a big time step (`par_timestep`) relative to the frequency of the pulse. Big `par_timestep`s are possible for weak pulses regardless of their frequency using `littlesteps` > 1. **Try bigger timesteps, with `littlesteps` > 1, especially if your pulse is weak.** E.g. `par_timestep=0.5d0` and `littlesteps=10`. Yes, for propagation!

- Try bigger time steps in general as lots of nuts and bolts have been tightened. `expotol` should be set low (stringent tolerance criterion), like 1d-8 or 1d-9, because in the end it goes faster for the entire run with a stringent criterion anyway. There is no need for `exposteps` which has been deprecated. But you may wish to optimize `expodim` (starting Krylov dimension) and especially `maxexpodim` (its maximum value). If your orbital or a-vector iterations are restarting, as per `expo.dat` or `avecexpo.dat`, you should be sure `maxexpodim` or `maxaorder` is high (but not too high – two or three hundred), but not worry if it is restarting after that. See "optimizing your runs" below.

- The variables `nuccharge1`, `nuccharge2`, `lbig`, and `mbig` are in namelist `&heparinp` or `&h2parinp`, not `&parinp`.

- All polarizations in both gauges are now possible using `pulsetheta` and `pulsephi`. `pulsestrength` is complex-valued, which permits the use of nonphysical rotating waves.

- `whichprojflux` and `corrflag` are deprecated. The options for the Hanning window implemented by B. A. remain; whereas they were `corrflag`=3 or 4 before, now the variable is `hanningflag`=3 or 4, with other values having no effect.

- `improvedquadflag=1` is the option for doing Newton solve for the A-vector to converge excited states. Options 2 and 3 are NOW REINSTATED (for orbitals). 2 = orbitals, 3 = both.

- `improvednatflag`=1 is fully debugged; it and `improvedquadflag`=1 can be set at the same time reliably. Note the message "Vectors converged on read" when performing relaxation and `sparseconfigflag`=1 that typically is printed each iteration as the relaxation is nearing convergence if one does not alter the default `lanthresh=1d-9`, even with `improvednatflag`=1.

Less important

- **There is orbital parallelization.** Running in parallel should speed the calculation regardless, unless you have a very small grid for the orbitals perhaps. Variable `parorbsplit`, default 1 (parallel). Note the `MPI` column in file `timings/actreduced.time.dat`. Not all aspects of the multiplication are parallel; a good choice is `nprocs`= $q\times$`nspf`, with q integer; then `q` copies of the orbitals are propagated and MPI communication is among groups of `nspf` processors.
- `expodim` is now the starting dimension of the orbital propagation, and new variable `maxexpodim` is its maximum value. Likewise for `aorder` and the new variable `maxaorder`.
- The nature of the stopping criterion `stopthresh` for relaxation has been changed. It now is independent of time step and a function of the rate of change of the orbitals. It is more robust. A typical value of 10^{-5} should be good, 10^{-4} for hard cases; as low as 10^{-7} should work if you want to be sure things are converged.
- Restricted configuration lists are good to go with `constraintflag`=1 or 2, density matrix or Dirac-Frenkel. BOTH require `dfrestrictflag`=1 or greater.
- There is a `numskiporbs` variable in `&parinp`, and `num_skip_orbs` in `&h2parinp` / `&heparinp`. They have different meanings: orbitals on file to skip, in the first case, and core orbitals to skip per m value for the generation of initial orbitals, in the latter cases.

Even less important

- There is the flag `orbcompact` in `&parinp` set by default to zero. This is a new option; you can try it if you want. When spfrestrictflag.ne.0, i.e., when there are symmetry restrictions on the orbitals, and `mbig` is not zero, we believe you can now do things the fully best way by setting `orbcompact=1`. Things probably won't change, and it is not extensively debugged. But, you will probably notice that it takes fewer steps occasionally, if you examine `expo.dat`. Usually, it will be the same number of steps, and if you are doing things correctly with tight error tolerances, all the numbers, the results, will probably be exactly the same. **Calculations in which the Hamiltonian does not conserve the symmetries imposed on the orbitals are now possible.** We believe it is programmed it up correctly. In other words, with `orbcompact=1`, `spfrestrictflag=1`, `spfugrestrict=1`, a calculation done with a heteronuclear diatomic, or a pulse, is being done correctly, we hope. With `orbcompact=0`, it is not. This is exotic; don't try it without discussing it; we want to do it to see if we can get the Fluorine atom to photoionize correctly.
- `timefac` is set internally according to `improvedrelaxflag`. Has no effect in `&parinp` except if `timefacforce` isn't zero.
- The code can be compiled with script `Makeme` in e.g. `BIN.ecs.hermnorm`.
- The code is now linked separately making a version for each coordinate system, and the F has been added to the name. E.g. `chmctdhf_atom`, not `chmctdh`. `atomflag` is deprecated.
- The DVR for spherical polar coordinates has been changed. Results will not be identical.

0.6 Optimizing your runs

To successfully use the LBNL-AMO-MCTDHF package you must pay attention to timings. The mean field treatment has several components and it is necessary to intelligently select the basic parameters of the propagation, which are among the `&parinp` namelist variables, to achieve best performance.

A basic part of the mean field implementation is the separation of the orbital and A-vector propagation. The orbitals and the A-vector are propagated separately over a small time step, the mean field time step, the `&parinp` namelist variable `par_timestep`. Thus, a basic question is, which is taking longer, orbital propagation or A-vector?

We recommend performing rough draft calculations with grids as low-resolution as possible, such that the orbital calculation runs fast. We recommend not attempting to "converge" with respect to number of orbitals, namelist variable `nspf`. For first row diatomics, the ten-orbital calculation is the obvious starting point. However, if more orbitals are desired, the A-vector propagation may begin to take the majority of the computation time.

For the polyatomic version `chmctdhf_sinc`, you will find best performance using one process per node (as opposed to one process per core, for `chmctdhf_atom` and `chmctdhf_diatom`, which are generally run as sequential programs, not parallel). There are both vectorized (xSSE4.2, with intel FFT) and OpenMP (using FFTPACK) compilation directories provided for the lawrencium machine in `COMPDIRS`. The sequential (no shared memory, only MPI parallelization) version is directory `BIN.ecs.hermnorm.law.seq`. On this machine `$OMP_NUM_THREADS` controls the number of threads per process and for the vectorized and OpenMP versions, all threads are taken if it is not set.

To ask the question, "what is the breakdown in timings for my run?" please set `&parinp` namelist variable `notiming=0`. The code will answer, proceeding to output large amounts of information into separate text files in the `timing` directory. See sections 0.3 and 5.3.1 on file system etiquette.

- There is `Main.time.dat` but it should show that the time is all spent in propagation. Below you see there were about 6000 time units spent at the beginning, counted under Non MPI. MPI +Non MPI is the total time and if the parallel propagation is working ok then MPI (communication) should be smaller than Non MPI (computation). If you don't include the startup time which is only counted in the MPI and Non MPI columns, roughly speaking, MPI+ Non MPI should add to the sum of the others. TO-DO: ADD BIORTHOGONALIZATION COLUMN.

	Spfs	Prop	Act	Final	MPI	Non MPI
0.000	0	508	0	2	16	6381
0.050	0	857	0	5	23	6726
			.			
			.			
2.150	0	10362	0	121	232	16139
2.200	0	10539	0	124	242	16310

- To see the breakdown of the `Prop` column, examine `cmf_prop.time.dat`:

	Time	matel	denmat	reduced	spfprop	aprop	advance	constrain	#DERIVS
T=	0.100	40	2	0	146	640	3	0	81
T=	0.150	61	4	0	229	865	5	0	80
T=	0.200	81	5	0	282	1090	7	0	48
T=	0.250	103	6	0	360	1315	9	0	58
T=	0.300	124	8	0	424	1539	10	0	58
T=	0.350	145	9	0	521	1763	12	0	104
T=	0.400	166	10	0	592	1988	14	0	63
T=	0.450	186	12	0	664	2213	16	0	63

  If A-vector propagation is taking quite a bit longer than orbital (spf) propagation, as above, run in parallel and try both `parorbsplit`=0 and 1 if you want to be systematic and agnostic about it. **If matrix elements, density matrices, or any of the in-between steps are taking a long time, CERTAINLY consider using a large `par_timestep` and `littlesteps`> 1**. Most of the time is usually taken in orbital propagation, in which case `par_timestep` shouldn't affect overall speed that much, unless very small.

#DERIVS

is the number of evaluations of the mean field operator required to take the time step (both predictor and corrector **CHECK**) and should never get too much bigger than $400 \times 2 = 800$ or so. To get time per mean field evaluation, divide the other columns by the number of DERIVS.

- The timing file actreduced.time.dat is very important and tells about column "spfprop" in cmf.prop.time.dat above. However, note that this column spfprop is not only actreduced, which is the name of a subroutine; there is also the operation of the projector, which is counted separately, and so to see the breakdown of column spfprop in cmf.prop.time.dat, go to jacoperate.time.dat, note that the first column is quite a bit smaller than the second of the two columns, and then look at actreduced.time.dat which goes like this:

	rmult	ke	pot	pulse	nuc	twoe	invdenma	project	constrai	MPI
T= 0.4500	169	881	88	106	0	318	18	4	0	55
T= 0.1500	333	1688	177	212	0	632	35	8	0	110
T= 0.3500	496	2496	266	318	0	947	52	12	0	165
T= 0.6500	659	3303	355	424	0	1261	69	16	0	219

Note that there are two steps to the mean field propagation of the orbitals, the predictor step and the corrector step, so the times, the first numerical column, may not be in order. Orbital parallelization with parorbsplit set nonzero should affect the columns ke through twoe because those refer to steps that are distributed over processors by this option. Generally speaking, with nprocs=nspf, if the orbital parallelization is working well you should observe that the steps rmult, perhaps invdenmat, and MPI take a significant proportion of the time.

- Examine expo.dat which tells you about the orbital propagation. In the example below, notice the numbers 1 and 48 at the bottom. That's one Krylov iteration, no restarts, with 48 matrix-vector multiplications required. This is the output near the start of the run. It has increased the krylov dimension up to 46 and at that point it does not need to restart, whereas at the first step it required 2 steps, one restart, with 66 matrix vector multiplications total. There have been steps rejected. This will happen at the beginning of the run when it is adjusting the internal step size, but should NOT happen afterwards that much. If it is, something is wrong. It will increase the Krylov order (thisexpodim as reported below) from where it starts (namelist input expodim) up to the maximum krylov order (namelist input maxexpodim), and then if it is restarting then the number of steps will be greater than one, but it should not be rejecting steps and changing its stepsize. In a nutshell: increase maxexpodim up to 300, or something like that, and then if it is restarting, (steps > 1) so be it, but it should not be rejecting steps very often, after the start of the run. Consider increasing littlesteps if you are trying to take big steps (large par_timestep).

```
 Go Orbital Expoprop.  Tinit=  0.100000000000000       thisexpodim=          31
  step   5.000000000000000E-002
       ...REJECTED  0.5000E-01  0.3849E-06  0.1000E-05
       ...REJECTED  0.4000E-01  0.1023E-06  0.1000E-05
    End expo. Orthog error, steps, stepsize, iterations  6.050711511541938E-015
           2  3.200000000000001E-002          66

 Go Orbital Expoprop.  Tinit=  0.100000000000000       thisexpodim=          38
  step   5.000000000000000E-002
       ...REJECTED  0.5000E-01  0.2653E-06  0.1000E-05
    End expo. Orthog error, steps, stepsize, iterations  6.213364751595364E-015
           2  4.000000000000001E-002          80

 Go Orbital Expoprop.  Tinit=  0.150000000000000       thisexpodim=          46
  step   5.000000000000000E-002
    End expo. Orthog error, steps, stepsize, iterations  1.136962511459106E-014
           1  4.999999999999999E-002          48
```

- Check file avecexpo.dat, for the A-vector propagation, similar to expo.dat above.
- If running in parallel, check to be sure you get the same results (certainly to six digits for everything) for different numbers of processors.

- `improvednatflag` to enforce natural orbitals for relaxation has been extensively debugged but it involves a difficult transformation of the wave function. For very large A-vectors, especially, be sure it is working ok; do the run without the flag as well. It is conceivable that the biorthogonalization routine could require tweaking for difficult cases, using the variables `biodim` and `biotol`. The only two reasons, in general, for which one *needs* to use this option `improvednatflag` in general is if, first, one is going to perform excitations or annihilations subsequently – or alternatively, second, if using density matrix constraint (`constraintflag=1`) in the subsequent propagation step.
- **Try larger time steps** as described above. If you have a high frequency or strong laser and you are trying to take large time steps (e.g. for big A-vector), try the new variable `littlesteps` > 1 which may improve accuracy and stability in addition to speed, as described above.
- Use the smallest primitive basis possible until you are doing production runs. Go as small as you can without much static appearing in your result. First make sure things are qualitatively right. **Absolute energies are not relevant. Gauge dependence is.** Don't sweat getting transitions energies right, unless something is qualitatively wrong. Often transition/ionization energies can be off by an eV or more.

0.7 Nomenclature

Here are some quick statements about the terms used in this manual.

We sometime use the terms configuration and Slater determinant interchangeably. Sometimes when we write configuration, we mean either a Slater determinant or a spin adapted sum of Slater determinants. But usually, we just mean Slater determinant.

The wave function is made of a time dependent linear combination of Slater determinants. The coefficients in the linear combination are time dependent. The Slater determinants are also time dependent, because they are made of orbitals that are time dependent. Sometimes (with `allspinproject=1`, default) the spin symmetry of the wave function is enforced. Then, the wave function is first represented as a linear combination of spin adapted linear combinations (SALCs is an acronym we have seen) of Slater determinants. Those linear combinations, the SALCs, are computed when the code starts.

So, not worrying about whether or not we are discussing Slater determinants (`allspinproject=0`) or spin adapted linear combinations thereof (`allspinproject=1`), we will generally use the term "configuration."

Again, we borrow from the MCTDH implementation of the Heidelberg group for not only the notation used in the equations, but also the terms by which they are referred to and often even the variable names in the code.

Orbitals are the same thing as single particle functions, SPFs. The configuration coefficients are the "A-vector," $\vec{A}$. We have one- and two-electron reduced density matrix operators ρ and Γ, the reduced potential $\mathbf{w}$, the reduced mean field operator $\mathbf{W}_{\alpha\beta} = \sum_{\gamma\delta} \Gamma^{\gamma\delta}_{\alpha\beta} \mathbf{w}_{\gamma\delta}$, and the one-electron Hamiltonian h_0. The orbitals are $\vec{\phi}$; the coefficients of ϕ_α are $\vec{c}_\alpha$.

For restricted configuration spaces, in which the orbitals are constrained to rotate into one another, we compute the $g_{\alpha\beta} = \langle \phi_\alpha | i \frac{\partial}{\partial t} | \phi_\beta \rangle$ matrix. The matrix elements of g among configurations are called τ.

$$i\frac{\partial}{\partial t}\vec{A} = (H - \tau)\vec{A}$$

$$i\frac{\partial}{\partial t}\vec{\phi} = \left[(1 - P)\left(\rho^{-1}\mathbf{W} + h_0\right) + g\right]\vec{\phi}$$

Chapter 1

Installation

System administrators may use this section for very quick startup especially on supported machines. Users may proceed to the next section.

The directory structure of the code makes extensive use of symbolic links and if you copy the directories you should ensure that you maintain the symbolic link structure. On mac, use `cp -R` to copy links as is. Otherwise, they are copied as new files, and you have duplicates. This is not a big deal, but it can be a waste of space and if you update the file then you have inconsistencies. You want to keep one copy of the files only. Look at the directories, see the symbolic links, and then proceed to use these instructions:

The code is provided with the following directory structure. The result of the `ls` command is schematic and embellished to give you the idea.

```
prompt> tar -xvf MCTDHF.010115.VERSION1.0.tar.gz
prompt> cd MCTDHF.010115.VERSION1.0
prompt> ls
   LBNL-AMO-MCTDHF.pdf
   EXAMPLES
      H2PHOTO
   JOB_SETUP_SCRIPTS
      corotating_counterrotating_streaking.py
      TEMPLATES
         mpilaunchjob.slurm.template
         atomfile.txt
         Relax.Bat.template
   MCTDH.SRC
      DFFTPACK
         zfftf1.f
      SINCDVR
         sincDVR.f90
      H2PROJECT
         H2_params.f90
   COMPDIRS
      MCTDH.SRC -> ../MCTDH.SRC
      BIN.ecs.hermnorm.law.openmp.nofft
         Definitions.INC
         Name.Txt
         Makefile.header -> ../MCTDH.SRC/Makefile.header.lawrencium.openmp.nointelfft
         Definitions.ALL -> ../MCTDH.SRC/Definitions.ALL
         mctdhf.F90 -> ../MCTDH.SRC/mctdhf.f90
         DFFTPACK
            Makefile.header -> ../Makefile.header
            Makefile -> ../../MCTDH.SRC/DFFTPACK/Makefile
```

```
         zfftf1.f -> ../../MCTDH.SRC/DFFTPACK/zfftf1.f
      SINCDVR
         Makefile.header -> ../Makefile.header
         Makefile -> ../../MCTDH.SRC/SINCDVR/Makefile
         sincDVR.f90 -> ../../MCTDH.SRC/SINCDVR/sincDVR.f90
   debug.BIN.ecs.hermnorm.mac
   BIN.mac
   BIN.ecs.hermnorm.edison
```

etc.

There are three directories at top level, `MCTDH.SRC`, `EXAMPLES`, and `COMPDIRS`. The `MCTDH.SRC` directory at top level contains the actual hard copy of the fortran code. There are many directories ready-to-go for installation of the code on various machines, provided in `COMPDIRS`. There are optimized versions, and versions with debugging flags. Please compile the debugging version too and use it if the code seems to be producing an invalid or unstable result, or actually crashes. The supported machines, the machines with ready-to-go directories in `COMPDIRS`, are

- Lawrencium (extension `.law`, e.g. `BIN.ecs.hermnorm.law.openmp.nofft`). Intel sandybridge, intel fortran compiler with mkl.
- Edison on NERSC (extension `.edison`, e.g. `debug.BIN.ecs.hermnorm.edison`)
- Mac (extension `.mac`, e.g. `debug.BIN.mac`)

You may delete the directories in `COMPDIRS` that you don't want, but they don't take any space.

All you need to do is go to the `BIN` directory you want and run `./Makeme`. The code should then be compiled as, e.g.

```
prompt> mv COMPDIRS/*BIN*mac* .
prompt> rm -r COMPDIRS/*
prompt> mv *BIN*mac* COMPDIRS
prompt> cd COMPDIRS/BIN.ecs.hermnorm.mac
prompt> ./Makeme
prompt> ls
      .
      .
chmctdhf_sinc
chmctdhf_diatom
chmctdhh_atom
```

Then you need to put the program in your `$PATH` somehow so that you can use it in any working directory. We recommend

```
prompt> echo $PATH
  /usr/bin:/usr/sbin:/opt/local/bin:/home/me/bin
prompt> cd ~/bin
prompt> ln -s ~/myprogs/LBNL-AMO-MCTDHF/V1.0.INTELFFT/COMPDIRS/BIN.ecs.hermnorm/chmctdhf_diatom ./chmctdhf_diatom
prompt> ln -s ~/myprogs/LBNL-AMO-MCTDHF/V1.0.INTELFFT/COMPDIRS/debug.BIN/mctdhf_sinc ./mctdhf_sinc.debug
```

There are four versions of the code with different datatypes, but `chmctdhf` is all that is generally needed. Compilation directories are provided for `chmctdhf` and `mctdhf` only.

- `mctdhf` : REAL VALUED VERSION FOR RELAXATION. (e.g. `BIN.mac`)
- `pmctdhf` : COMPLEX VALUED, PROPAGATION WITHOUT ECS (e.g. `BIN.complex.mac`)
- `chmctdhf` : COMPLEX VALUED, WITH ECS: USUAL VERSION (e.g. `BIN.ecs.hermnorm.mac`)
- `cmctdhf` : COMPLEX VALUED, ECS, C-NORM: FOR RESONANCE RELAXATION (e.g. `BIN.ecs.cnorm.mac`)

If you are using an architecture different from all of these machines, then you should make a new `BIN` directory and a new `Makefile.header.mymachine` in `MCTDH.SRC`, linking the new header file as `Makefile.header` in the new `BIN` directory. There are parallel versions of `Makefile.header` for mac already done. Your fortran compiler must support the C preprocessor. You may have to figure out flags to invoke it. If you want to run the calculation in parallel (there is now some orbital parallelization!) then in Makefile.header you must include the line "`MPIFLAG = -D MPIFLAG`" and include paths or flags in variables `MYINCLUDE` and `LAPACK` if needed. Intel FFT libraries may be used by setting `FFTFLAG = -D INTELFFT`. Portland group compilers require `PGFFLAG = -D PGFFLAG`.

Chapter 2

Input

The code is run from the command line and takes command line arguments and also parameters from an input file, by default `Input.Inp`. Command line arguments supersede input from file. All variables that can be set are listed in Sec. 2.6. To use an input filename different from `Input.Inp`, specify it as follows:

```
C:> chmctdhf_atom Inp=Input.Inp.myinput
```

If `Input.Inp` or the specified input filename does not exist it will do a default calculation. One can specify multiple command line arguments, e.g.

```
C: > chmctdhf_atom Inp=Input.Inp.awesome T=1000.0 Nspf=4 Act=1
```

2.1 Namelist input

Fortran includes a feature that makes it easy to input variable values from file, called namelist input. It is not case sensitive. Variable names in the namelists in the input file are the same as their names in the code. There are four namelists that can be used as input in the input file (default `Input.Inp`). The complete list of variables that can be set are listed in Sec. 2.6, which is a printout of `parameters.f90` with comments. You can also look at the code: for instance, the `&parinp` namelist variables can be seen in `getparams.f90`, subroutine getparams(); that's where the `&parinp` namelist is defined.

** In your input file: **

- You must have namelist `&parinp`.
- If variable `tdflag` in namelist `&parinp` is set to 1 (including a time-dependent pulse), or if doing a flux analysis calculation (action 16 or 17) (except if `noftflag` is set nonzero in `parinp`), you must also have namelist `&pulse`.
- You must have `&h2parinp` for diatoms (`chmctdhf_diatom`), `&heparinp` for atoms (`chmctdhf_atom`), or `&sincparinp` for polyatomics (`chmctdhf_sinc`).

2.2 Specifying the wave function

The orbitals and A-vector are specified independently. One may load either of them, or obtain them by diagonalization. For orbitals, the core Hamiltonian is used for diagonalization. One needs to specify the symmetry constraints for each. Restricting the symmetry of the A-vector requires restricting the symmetry of the orbitals, but not vice-versa.

2.2.1 Symmetry constraints

By default the S_z spin angular momentum projection quantum number is restricted. It is specified by `restrictms=X`. This is $2\times$ the S_z quantum number.

By default the wave function is restricted to be an eigenfunction of spin, $\hat{S}^2$, via `spinwalkflag=1` and `allspinproject=1`. The high spin case is calculated ($S = M_s$). Thus one specifies the multiplicity by setting `restrictms` = multiplicity-1, i.e. for a doublet, `restrictms`=1.

One generally wants to restrict spatial symmetry as well. To restrict the l_z quantum number (angular momentum projection) of the orbitals set `spfrestrictflag=1`. Then specify the m_z values of the orbitals by `spfmvals=0,1,-1,...` To restrict the parity of the orbitals set `spfugrestrict=1`. Then specify the parity by `spfugvals=1,-1,0,0,...` where 0 means no parity restriction.

Without `spfrestrictflag=1`, `spfmvals` is only used if you are starting from core orbitals, for your initial relaxation run for instance, in which case it is just used to select the initial orbitals. Similarly for `spfugrestrict`.

To restrict the overall angular momentum projection L_z of the wave function, `spfugrestrict` must be set nonzero; then set `mrestrictflag`=1 and `mrestrictval` to L_z. Similarly with `ugrestrictflag` and `ugrestrictval`.

To restrict L_z to a *range* of values, do not set `mrestrictflag`; instead set `mrestrictmin` and `mrestrictmax`. This is useful for relaxation or propagation of multiple wave functions. *See warnings in the section on relaxation.*

If one is reading in orbitals from file and `spfrestrictflag` or `spfugrestrict` is set, then the angular momentum projections or parities of the orbitals must match those in the present calculation.

2.2.2 Restricted configuration spaces

Restricted configuration spaces, non-full-configuration interaction, are a new capability and the numerical implementation is more difficult than the full CI method. There are more things to do, and the integrator ends up taking more steps to calculate the more rapidly changing orbitals. There are profound issues relating to symmetry broken solutions of a relaxation calculation, to prepare the initial state, and therefore we advise using the real-valued versions `BIN/mctdhf_XXX` to prepare the initial state, or at least checking that the same energy is obtained with both complex and real versions.

For these reasons using restricted configuration spaces be avoided until you have pushed the limit of a one node calculation. Doing the code in parallel over many nodes is not something we want to do if we want to make the best use of computer time. For instance, with neon, with greater than 14 orbitals, one is going to need a restricted configuration space, despite the use of `sparseconfigflag=0`, `sparseopt=0`, and small values for Krylov space parameters (`maxaorder`, `aorder`, even `maxexpodim` if the orbitals take up some space that can be freed).

You can also try frozen orbitals, `numfrozen`, which reduces the number of electrons in your full CI calculation, to avoid having to use a restricted configuration space. Frozen orbitals work, but they also require more things to do, slowing the calculation down, and make the problem more numerically difficult. *But, if you want, do try frozen orbitals if your physics does not perturb the core orbitals!* This will require consultation as the manual is not complete. See section 2.2.4.

There are two methods for restricted configuration spaces (see the paper). Density matrix (`constraintflag=1`) and dirac-frenkel (`constraintflag=2`). A restricted configuration space can be used with `constraintflag=0`, such that the g/τ matrix remains set to zero, but one is making uncontrolled error in the wave function.

With `constraintflag` not equal to zero, `dfrestrictflag` must be set nonzero. Unless you are running action 22, it should be set `dfrestrictflag=1`.

Then, **ALL THE FOLLOWING VARIABLES** should be incremented by the chosen value of `dfrestrictflag`. That is because, in the code, they will be decremented by that amount, to obtain the included list of configurations. "Incremented by 1" means, change the value by 1 in the direction that enlarges the restricted configuration space.

```
minocc
maxocc
numexcite
vexcite
```

Variables (namelist `&parinp`) specifying a restricted configuration list are as follows. The variables

```
numshells
shelltop
```

come first. They divide the `nspf` orbitals into shells. By default there is one shell, `numshells`= 1, and `shelltop(1)`=`nspf`. `shelltop` denotes the last spatial orbital in each shell. You must enter `shelltop` for shells 1 through numshells-1. The value of shelltop for the last shell is automatically set to `nspf`.

One way of doing a restricted configuration list, is to use one or the other or both of the two variables

```
numexcite
vexcite
```

and leave it at that. Variable `numexcite` may be set for shells 1 through numshells-1; `numexcite`(i) corresponds to the maximum number of holes in shells 1 through i (so the list, if `numshells`>2, should be never decreasing). `vexcite` is the opposite of `numexcite` and is only input for the last shell; it corresponds to the maximum number of electrons in the last shell and so can be used to easily restrict the configuration list to all singles and doubles (CISD), etc.

The other available options are `minocc` and `maxocc` which can be input for shells 1 through `numshells`. These are minimum and maximum occupation numbers.

To reiterate, `vexcite`, `numexcite`, `minocc`, and `maxocc` must all be incremented by the value of `dfrestrictflag` (1, unless using action 22, or unless debugging) from what you want them to be for the restricted configuration space you want. The configuration lists are built using the incremented values, but the wave function is constrained to occupy only the restricted configuration space you want. If you are just playing around and `constraintflag` and `dfrestrictflag` are both zero then just set these variables how you want them; you can do that, but you are making uncontrolled error.

TO PREPARE THE INITIAL STATE: To prepare the initial state for a propagation with the density matrix constraint (`constraintflag=1`), perform a relaxation calculation with `improvednatflag=1`, `constraintflag=0`, `dfrestrictflag=0`. You could also set `dfrestrictflag` to the same value (probably 1) that you use for the propagation, in which case your values for `vexcite`, `minocc`, etc. will be the same. But `constraintflag=0`, `improvednatflag=1` for density matrix initial state calculation. For Dirac-Frenkel (`constraintflag=2`), perform a relaxation calculation with the restricted configuration space machinery just as done for propagation. You may even find you have to do regular imaginary time relaxation calculation (`improvedrelaxflag=0`, `threshflag=1`). For relaxation or improved relaxation you will need small time steps. Set `constraintflag=2`, `dfrestrictflag=1` (or whatever nonzero), just like the propagation calculation.

Always, if you are indeed taking the bold leap into restricted configuration space calculations, you should definitely do it BOTH WAYS, density matrix and dirac-frenkel, `constraintflag=1` and 2. Hope-

fully they will both work and if they do, it is a powerful convergence check. Remember that you can check the overlap between two time-dependent wave functions using Actions 15 and 23.

2.2.3 Specifying the initial orbitals

By default the orbitals are obtained by diagonalizing the core hamiltonian. These would be used for a relaxation calculation. Thus, for a relaxation calculation, no extra input is required.

One loads orbitals from file with `loadsppflag=1`.

```
&parinp
  loadspfflag=1
  spffile="Bin/spfs.bin.mine"
```

By default, the orbitals are loaded from `./Bin/spfs.bin`, the same file that is written at the end of a run. In practice you would save this file elsewhere and specify its location as above.

If you are loading orbitals with `loadsppflag=1` and there are fewer orbitals in `spfs.bin` than in the calculation, then the additional orbitals are taken from core orbital eigenfunctions. This is useful if you want to converge an excited state with a small number of orbitals first, then add orbitals to get a better wave function.

For a relaxation run, obtaining the initial guesses for the orbitals from core diagonalization, even if specifying `spfmvals` and `spfugvals` as appropriate, one may find that the initial guess orbitals are incorrect for the desired calculation. For instance, the ordering of $3s$ and $3d_{z^2}$ will depend on small numerical errors. One may skip over the orbitals as follows: say that you want the 1st, 2nd, and 5th orbitals for m=0 and the 1st and 3rd for m=$\pm$1. One would specify

```
&h2parinp
  num_skip_orbs=3
  orb_skip=3,4,2
  orb_skip_mvalue=0,0,1
```

Thus you list the orbitals to skip and the corresponding m-values.

To read in two or more small sets of orbitals to combine into one set (useful for interacting fragment type description) use `numspffiles` and `spffile(1)`, etc. For instance you might have a run with three orbitals and one with four and you could combine these for a seven orbital calculation. You can skip over orbitals that are read in this step, by specifying `numskiporbs`, in namelist `&parinp`.

2.2.4 Frozen orbitals

If your calculation is getting out of hand (can't fit on one node, in memory), and it only involves valence electron dynamics, then you have two options – restricted configuration spaces, or frozen orbitals. If it involves core electron dynamics, you only have the restricted configuration space option. Otherwise, if you are really at this point, perhaps, try both.

Frozen orbitals work like this. Variable `numfrozen` in namelist `&parinp` is number of frozen orbitals (spatial orbitals, containing two electrons). So you might start with a small full CI calculation. Say Neon, 6 orbitals. Don't do hartree-fock! We do not have an actual fock Hamiltonian. We are missing the "scalar terms" in the reduced operator $\hat{\mathbf{W}}$. They do not matter when $(1 - P)$ is put in front of them. Therefore we do not have a good 1s energy eigenfunction. All we have is natural orbitals – or, now, the Dirac-Frenkel constraint – to define orbitals.

By "define orbitals" we mean, in better terms, "resolve orbitals". Given the space of orbitals – six, say, for neon, with an extra $3s$ orbital beyond the Hartree-Fock – resolve orbitals means specify a specific set of six orthonormal orbitals.

So, the easiest thing is to start with non-Hartree-Fock, `improvednatflag=1` calculation, with the restricted configuration space (no need for `dfrestrictflag` nonzero).

Take the orbitals on disk, and read them in for a calculation with frozen orbitals. Start with neon, six orbitals, then read those orbitals in, with `nspf=5`, `numfrozen=1`. `nspf` is the number of unfrozen orbitals. That will do a calculation with one frozen orbital, and five unfrozen orbitals, starting with the six orbitals from your first calculation; the frozen orbital is now the $1s$ natural orbital from the first calculation. You can go on and add more orbitals.

Specify `spfmvals` and `spfugvals` only for the unfrozen orbitals.

So, you could have done `nspf=8`, `numfrozen=1`, and it would have loaded the six orbitals, then added three more (the p orbitals) gotten from diagonalization of the core Hamiltonian.

Going from six orbitals to `nspf=13`, `numfrozen=1`, one would set the thirteen values of `spfmvals` and `spfugvals` appropriately, then, we think this is how it goes, one would need to specify `num_skip_orbs=1` and `orb_skip_mvalue=0` in `&h2parinp` or `&heparinp`, to get the usual set of fourteen orbitals.

2.2.5 Specifying the initial A-vector

Similar to the orbitals, one may specify

```
&parinp
  loadavectorflag=1
  avectorfile="Bin/avector.bin.mine"
```

With `mcscfnum` > 1 the code propagates multiple A-vectors, but still uses only one set of orbitals. To read in multiple A-vectors for such a run, use `numavectorfiles` and `avectorfile(1)=`, etc.

One may also excite or annihilate orbitals and the following subsections describe that.

When the excitation or annihilation is performed, it is done so in terms of the spin orbitals and the spin value is not conserved. What is obtained from the excitation or annihilation is then projected upon the spin value specified in `restrictval`, if `spinwalkflag`=1.

One should take care that the orbitals one is annihilating or exciting are meaningful. If one does a regular improved relaxation run, the orbitals will be arbitrarily mixed. If you want to have meaningful molecular orbitals, use `improvednatflag=1` in namelist `parinp` in the relaxation run; the orbitals will then be natural orbitals. Fock eigenfunctions are not implemented.

2.2.6 Annihilation

One may read in an $(N+1)$-electron wave function and then annihilate an electron.

Use `avectorhole=` in namelist `&parinp`.

```
&avectorhole=1
```

This will annihilate the first spin orbital. As it hapens the even numbered spin orbitals are spin up and the odd numbered are spin down. So if one is reading in a triplet wave function (`restrictms=2`) into a doublet calculation (`restrictms=2`) and annihilating an electron, because the high spin case is calculated one annihilates a spin up electron, odd numbered, as above.

One may use a linear combination of hole wave functions. This would be useful for annihilating the left or right core hole. To do this specify

```
&numholecombo=2
&avectorhole=1,3
```

This makes the (+) combination of orbitals (which each come out with an arbitrary phase). To make the (-) combination use `&avectorhole=1,-3`.

2.2.7 Excitation

One may read in an A-vector and then perform an excitation or annihilation.

To perform an excitation, use `avectorexcitefrom` and `avectorexciteto` in namelist `&parinp`. These variables should be assigned to the spin orbital indices. With the default value of `orderflag`, this means that to perform an excitation from the first spatial orbital, spin down, to the third spatial orbital, spin up, you'd do `avectorexcitefrom=2`, `avectorexciteto=5`. As for annihilation, the excitation does not preserve the $S(S+1)$ quantum number and the wave function obtained is subesequently projected on the given spin space if `spinwalkflag=1`.

2.3 Primitive basis and Hamiltonian

The primitive basis and hamiltonian are specified in the Fortran namelists `h2parinp`, `heparinp`, or `&sincparinp` in the input file. The namelist read depends upon the version of the code that is run (`chmctdhf_diatom`, `chmctdhf_atom`, or `chmctdhf_sinc`).

```
&parinp
nspf=5
spfmvals=0,0,1,1,1
...
/

&h2parinp
NucCharge1=1.d0
NucCharge2=5.d0
LBIG=13, MBIG=1
pro_hmass=1836.152701d0
pro_dmass=20213.07d0
bornopflag=0
xinumpoints=14
xinumelements= 8
xicelement=7
xiecstheta= 0.157d0
xielementsizes =  2.0d0, 10.0d0, 10.0d0, 10.0d0, 10.0d0, 10.0d0, 10.0d0, 10.d0, 10.d0,
rnumelements=1
rnumpoints=32
relementsize = 2.5d0
rstart=1.836d0
/
```

The most important options are :

- `bornopflag`: Performs a Born-Oppenheimer calculation, one is default. Zero: full nonadiabatic.
- `xicelement`: First element that is complex-scaled.
- `xinumelements`: Total number of elements in xi.
- `rnumelements`: Number of elements in R
- `LBIG`: Number of points in η minus one.
- `xinumpoints`, `rnumpoints`: number of points per element, including bridge functions.

The masses affect a fixed-nuclei (`bornopflag=1`, default) calculation because different masses will shift the origin in the prolate coordinate system. Different masses may give superior or inferior results for a given heteronuclear. If in doubt, don't include masses which will set masses equal.

2.4 Propagation parameters

See the advice at the top of the document about making runs go faster. The most important factor in speeding up the calculation is to use the maximum grid spacing possible. We advise using length gauge, if stable, and even grid spacing (no small first element) which we believe will give you the fastest possible converged calculation.

The following options may be useful for fine tuning the propagation.

```
&parinp
 aorder            !! Krylov dimension for A-vector propagation
 maxaorder         !! Krylov dimension for A-vector propagation
 aerror            !! Error tolerance for A-vector propagation  OR  improvedquadflag=1
 expodim           !! Corresponding Krylov dimension
 maxexpodim        !! Corresponding Krylov dimension
 expotol           !! Error tolerance
 littlesteps       !! Divide par_timestep into many steps for calculation with pulse
 denreg            !! density matrix regularization parameter (rarely matters)
 lioreg            !! if constraint, ESPECIALLY if DF constraint (constraintflag=2)
 invtol            !! regularization parameter for lots of matrix inversions/linear solves
```

2.4.1 Analyzing the propagation

See the section at the beginning of the manual about examining and improving the performance of your runs.

2.4.2 Using sparse configuration routines

Set `sparseconfigflag=0` to use nonsparse configuration routines; otherwise sparse is default. If `sparseconfigflag=0` is specified, there is a maximum number of configurations that is allowed and if it is exceeded the program quits. To override this you can specify `nosparseforce=1`. In practice there are few reasons to do this. If you have something like 100-300 configurations, you should check whether sparse or nonsparse is faster.

With `sparseconfigflag=0`, the default behavior is that the sparse CI Hamiltonian is constructed in sparse format. Sometimes – NO with 12 orbitals on Lawrencium, for instance – the calculation can't fit on one node with this default behavior. Setting `sparseopt=0`, as opposed to the default, 1, performs a direct CI instead. No sparse hamiltonian is constructed; memory use is less, but it is slower.

2.5 Action input

Things that one can optionally do to or with the wave function in-between time steps are called actions. Everything you want to do (fourier transform dipole moment, output orbitals, whatever) is an action, and specified by `numactions=XX`, `action=YY,ZZ,...`.

2.6 Complete list of namelist variables

Below we provide verbatim copies of the files `parameters.f90`, `H2PROJECT/H2_params.f90`, `HEPROJECT/He_params.f90`, and `SINCDVR/sinc_params.f90`. The first is for all calculations, namelist `&parinp`; the others are for namelists `&h2parinp`, `&heparinp`, and `&sincparinp`, which are required for `chmctdhf_diatom`, `chmctdhf_atom`, and `chmctdhf_sinc`, respectively. You will find description of lots of minor options in the code here.

```
module parameters
  use littleparmod;  use fileptrmod;  implicit none

!! ************************************************************************************************** !!
!!   Parameters for MCTDHF calculation; parinp NAMELIST input from Input.Inp (default)
!! ************************************************************************************************** !!
!!
!! Type, variable, default       !! Command      !! Description
!!   value                       !!  line        !!
                                 !!  option      !!
```

!! MAIN PARAMETERS

```
integer :: numelec=2             !!              !! NUMBER OF ELECTRONS
integer :: tdflag=0              !! Pulse        !! Use pulse?
integer :: mcscfnum=1            !! MCSCF=       !! Number of A-vectors (state avgd mcscf or prop)
integer :: sparseconfigflag=0    !! Sparse       !! Sparse configuration routines on or off (for large # configs)
integer :: orbcompact=0          !!              !! Compact orbitals for expo prop with spfrestrictflag?  Probably ok.
real*8 :: denreg=1d-10           !! Denreg=      !! density matrix regularization parameter.
real*8 :: invtol=1d-12
integer :: saveflag=1                            !! if zero does not save wave function at the end
integer :: save_every=0                          !! if nonzero saves wave function every save_every mean field steps
integer :: walkwriteflag=0                       !! Turning OFF writing of walks by default
integer :: spf_flag=1            !!              !! IF ZERO, FREEZE SPFS. (for debugging, or TDCI)
integer :: avector_flag=1        !!              !! IF ZERO, FREEZE AVECTOR. (for debugging)
integer :: parorbsplit=1                         !!  Parallelize orbital calculation.  Might speed up, might
                                                 !!   slow down; check timing.
!! FOR TOTAL ORBITAL PARALLELIZATION with SINC DVR, SET PARORBSPLIT=3
!!   and orbparflag=.true. in &sinc_params.  parorbsplit=3 not supported for atom or diatom.

character (len=200) :: &         !!              !! MAY BE SET BY COMMAND LINE OPTION ONLY: not namelist
  inpfile="Input.Inp        "    !! Inp=filename !!  input.  (=name of input file where namelist input is)
```

!! Biorthogonalization

```
integer ::      maxbiodim=100, &
     biodim=10                   !! Krylov dim for biorthogonalization
real*8 :: lntol=1d-12, &
     biotol=1.d-6
```

!! PROPAGATION/RELAXATION

```
real*8 :: par_timestep=0.1d0     !! Step=        !! MEAN FIELD TIMESTEP
integer :: improvedrelaxflag=0   !! Relax        !! For improved versus regular relaxtion.
integer :: threshflag=0          !!              !! Set to 1 for regular relaxation
real*8 :: expotol=1d-8           !!              !! Orbital krylov convergence parameter
integer :: maxexpodim=100        !!              !! Orbital maximum kry dimension OR DGMRES DIM improvedquad=2,3
real*8  :: expostepfac=1.2d0     !!              !! Miscellaneous algorithm parameter
```

!! **SPARSE - if sparseconfigflag .ne. 0**

```
integer :: maxaorder=100         !!              !!   lanczos order for sparse a-vector prop and improvedquad=1,3
integer :: sparseopt =1                          !! 0= direct CI  1= sparse matrix algebra (faster, more memory)
```

!! PROPAGATION

```
integer :: littlesteps=1         !!              !! Sub intervals of mean field time step for avector prop
real*8 :: finaltime=4d4          !! T=           !! length of prop.  Overridden for pulse and relax.
integer :: expodim=10            !!              !! Starting krylov size for orbital propagation (expokit)
```

!! **SPARSE - if sparseconfigflag .ne. 0**

```
integer :: aorder=30             !!              !!
real*8 :: aerror=1d-9            !!              !! lanczos error criterion for sparse a-vector CMF propagation
                                                 !!  within aerror to stop.
```

!! RELAXATION

```
integer :: improvednatflag=0     !!             !! If improved relax, replace with natorbs every iteration
real*8 :: stopthresh=1d-5        !!             !! Spf error tolerance for relaxation convergence (PRIMARY)
real*8 :: astoptol=1d-7                         !! Avector error tolerance for relax  (BACKUP - WAS STOPTHRESH)
real*8 :: timestepfac=1d0                       !! accelerate relax. multiply par_timestep by this each time
real*8 :: max_timestep=1d10                     !!    maximum time step (limit on exponential growth)
real*8 :: mshift=0d0                            !! shift configurations based on m-value.. to break
                                                !!  degeneracy for state averaged sym restricted
                                                !!  (mrestrictmin, mrestrictmax) mcscf; good idea.
integer :: improvedquadflag=0    !!             !! Use newton iteration not diagonalization for improvedrelax.
                                                !!     (1 = A-vector, 2 = orbitals, 3 = both)
real*8 :: quadstarttime=-1d0                    !! Waits to turn on orbital quad (2 or 3) until this time
real*8 :: maxquadnorm=1d10                      !! brakes to use if improvedquadflag=2 or 3 is diverging
real*8 :: quadtol=1d-1           !!             !! Threshold for solution of Newton solve iterations orbitals.
integer :: quadprecon=1          !!             !! Precondition newton iterations for A-vector?
real*8 :: quadpreconshift=0d0
```

!! **SPARSE - if sparseconfigflag .ne. 0**

```
integer :: lanprintflag=0
integer :: lanczosorder=200      !!             !!   lanczos order used in A-vector eigen.
integer :: lancheckstep=20       !!             !! lanczos eigen routine checks for convergence every this # steps
real*8 :: lanthresh=1.d-9       !!             !! convergence criterion.
```

!! ORBITALS (SINGLE PARTICLE FUNCTIONS, SPFS)

```
integer :: nspf=1                !! Nspf=       !! number of orbitals
integer :: numfrozen=0           !!             !! number of doubly occ orbs (removed from calculation)
integer :: spfrestrictflag=0     !!             !! Restrict m values of orbitals?
integer :: spfmvals(1000)=0      !!             !!   M-values of orbitals
integer :: spfugrestrict=0       !!             !! Restrict parity of orbitals?
integer :: spfugvals(1000)=0     !!             !!   Parity (+/-1; 0=either) of orbitals (ungerade/gerade)
```

!! CONFIGURATIONS

```
integer :: mrestrictflag=0       !!             !! If spfrestrictflag=1, restrict wfn to given total M.
integer :: mrestrictval=0        !!             !!    This is the value.
integer :: mrestrictmax= 99999   !!             !! If doing state averaged MCSCF, can include a range of m vals;
integer :: mrestrictmin=-99999   !!             !!    set these variables, with mrestrictflag=0, spfrestrictflag=1
integer :: ugrestrictflag=0      !!             !! like mrestrictflag but for parity
integer :: ugrestrictval=1       !!             !!    like mrestrictval but for parity (1=even,-1=odd)
integer :: restrictflag=1        !!             !! Restrict spin projection of determinants?
integer :: restrictms=0          !!             !! For restrictflag=1: 2*m_s: 2x total m_s (multiplicity of
                                                !!    lowest included spin states minus one)
integer :: spinwalkflag=1        !!             !! Calculate spin info (required for below 2 options)
integer :: allspinproject=1      !!             !! Constrain S(S+1) for propagation?
integer :: spinrestrictval=0     !!             !! For allspinproject=1: determines spin. Default high spin S=M_s.
                                                !!  To override use this variable. Equals 2S if S^2 eigval is S(S+1)
```

!! For restricted configuration lists (not full CI): SEE MANUAL about dfrestrictflag

```
integer :: numshells=1           !!             !! number of shells.  greater than one: possibly not full CI.
!!integer :: shelltop(100)=-1    !! Numfrozen=  !! shelltop is namelist input in parinp; the internal variable
                                                !!    is allshelltop.   shelltop(1) only may be assigned via
                                                !!    command line, with Numfrozen.
integer :: numexcite(100)=99     !! Numexcite=  !! excitations from core shells (i.e. defined for shells 1
                                                !!    through numshells-1).  Only numexcite(1) may be
                                                !!    assigned via command line input.
integer :: minocc(100)=-999      !!             !! minimum occupation, each shell
integer :: maxocc(100)=999       !!             !!    maximum
integer :: vexcite=99            !!             !! excitations INTO last shell. Use to restrict to doubles, etc.
```

!! INITIALIZATION

```
integer :: loadavectorflag=0     !! A=file       !! load avector to start calculation?
integer :: numavectorfiles=1     !!              !! number of avector files containing a-vectors to load into the
                                                 !! mcscfnum available slots, or if load_avector_product.ne.0,
                                                 !! number of one-wfn files for product, w/ total numelec=numelec
integer :: load_avector_product=0                !! make product wave function for multiple-molecule load
integer :: loadspfflag=0         !! Spf=file     !! load spfs to start calculation?
                                                 !!    (Otherwise, core eigenfunctions.)
integer :: reinterp_orbflag=0                    !! sinc dvr only, half spacing interpolation for orb load
integer :: spf_gridshift(3,100)=0                !! sinc dvr only, shift orbitals on read, slow index spffile
integer :: numspffiles=1         !!              !! for multiple-molecule (e.g. chemistry) calcs, load many
integer :: numskiporbs=0         !!              !! Reading orbs on file(s), skips members of combined set.
integer :: orbskip(1000)=0       !!              !! Which to skip
character (len=200) :: &         !! A=file       !! A-vector binary file to read.  Can have different configs
     avectorfile(MXF)="Bin/avector.bin"          !!   but should have same number of electrons.
character (len=200) :: &         !! Spf=file     !! Spf file to read.  Can have fewer m vals, smaller radial
     spffile(MXF)="Bin/spfs.bin"                 !!   grid, or fewer than nspf total orbitals.
integer :: avecloadskip(100)=0
integer :: numholes=0                            !! Load a-vector with this many more electrons and annihilate
integer :: numholecombo=1                        !! Number of (products of) annihilation operators to combine
                                                 !!   (for spin adapt)
integer :: numloadfrozen=0                       !! For loading a vector with orbitals to be frozen (dangerous)
integer, allocatable :: myavectorhole(:,:,:)     !! Namelist input is avectorhole.  Fast index numholes (#
                                                 !!   annihilation operators to multiply together); then
                                                 !!   numholecombo, number of such products to combine; then
                                                 !!   mcscfnum, wfn of current propagation.
integer :: excitations=0                         !! Similar to holes: number of products of excitation ops
integer :: excitecombos=1                        !!   number of products to linearly combine
integer,allocatable:: myavectorexcitefrom(:,:,:) !! Similar to avectorhole.  Namelist input avectorexcitefrom, etc.
integer,allocatable:: myavectorexciteto(:,:,:)   !!
                                                 !! For both excite and hole: value is spin orbital index
                                                 !! 1=1alpha, 2=1beta, 3=1alpha, etc.
                                                 !! negative input -> negative coefficent
```

!! CONSTRAINT: Constraintflag. NEED FOR RESTRICTED CONFIG LIST.

```
!!  1: Density matrix constraint: assume nothing, keep constant off block diag
!!         (lioville solve)
!!  2: Dirac-Frenkel (McLachlan/Lagrangian) variational principle.

integer :: constraintflag=0      !! Constraint=  !! As described immediately above
integer :: denmatfciflag=0                       !! If .ne. 0 then does denmat constrant as programmed
                                                 !!  before Miyagi's help
real*8 :: lioreg= 1d-9           !!              !! Regularization for linear solve for both
integer :: dfrestrictflag=0      !!              !! apply constraint to configuration list?  Must use this
                                                 !!  option if constraintflag /= 0.  1 is sufficient;
                                                 !!  dfrestrictflag=2 necessary for action 22.
                                                 !!  SEE MANUAL FOR PROPER USE OF dfrestrictflag/shell options.
integer :: conway=0                              !! for constraintflag=2, dirac frenkel constraint
                                                 !!   0=McLachlan 1=50/50 mix 2=Lagrangian
                                                 !!   3=Lagrangian with epsilon times McLachlan
real*8 :: conprop=1d-1                           !! epsilon for conway=3
```

!! INPUT / OUTPUT

```
integer :: notiming=2            !!NoTiming=0,1,2!! 0=write all 1=write some 2= write none
                                 !!  Timing=2,1,0!!     controls writing of all timing and some info files
integer :: timingout=499         !!              !! various routines output to file (timing info) every this
                                 !!              !!   # of calls
character (len=200) ::      avectoroutfile="Bin/avector.bin"  !! A-vector output file.
character (len=200) ::      spfoutfile="Bin/spfs.bin"         !! Spf output file.
character(len=200):: psistatsfile="Dat/psistats.dat"
character(len=200):: dendatfile="Dat/denmat.eigs.dat"
```

```
character(len=200):: denrotfile="Dat/denmat.rotate.dat"
character(len=200):: rdendatfile="Dat/rdenmat.eigs.dat"
character (len=200) :: ovlspffiles(50)="ovl.spfs.bin"
character (len=200) :: ovlavectorfiles(50)="ovl.avector.bin"
character(len=200):: zdipfile="Dat/ZDipoleexpect.Dat"
character(len=200):: zdftfile="Dat/ZDipoleft.Dat"
character(len=200):: ydipfile="Dat/YDipoleexpect.Dat"
character(len=200):: ydftfile="Dat/YDipoleft.Dat"
character(len=200):: xdipfile="Dat/XDipoleexpect.Dat"
character(len=200):: xdftfile="Dat/XDipoleft.Dat"
character(len=200):: corrdatfile="Dat/Correlation.Dat"
character(len=200):: corrftfile="Dat/Corrft.Dat"
character(len=200):: outovl="Dat/Overlaps.dat"
character(len=200):: fluxmofile="Flux/flux.mo.bin"
character(len=200):: fluxafile="Flux/flux.avec.bin"
character(len=200):: configlistfile="WALKS/configlist.BIN"
character(len=200):: fluxmofile2="Flux/flux.mo.bin"
character(len=200):: fluxafile2="Flux/flux.avec.bin"
character(len=200):: projfluxfile="Flux/proj.flux.wfn.bin"
character(len=200):: timingdir="timing"
character(len=200):: spifile="Dat/xsec.spi.dat"
character(len=200):: projspifile="Dat/xsec.proj.spi"
character(len=200):: natplotbin="Bin/Natlorb.bin"
character(len=200):: spfplotbin="Bin/Spfplot.bin"
character(len=200):: denplotbin="Bin/Density.bin"
character(len=200):: denprojplotbin="Bin/Denproj.bin"
character(len=200):: natprojplotbin="Bin/Natproj.bin"
character(len=200):: rnatplotbin="Bin/RNatorb.bin"
```

!! PULSE. (If tdflag=1)

```
integer :: numpulses=1
integer :: velflag=0             !!             !!  Length (V(t)) or velocity (A(t))
integer ::  pulsetype(10)=1      !!             !!  Pulsetype=1:  A(t) = pulsestrength * sin(w t)^2,
real*8  :: omega(10)=1.d0        !!             !!  2:  A(t) = strength * sin(w t)^2
real*8 :: omega2(10)=1.d0        !!             !!             * sin(w2 t + phaseshift),
real*8 :: pulsestart(10)=0.1d0   !!             !!
real*8 :: phaseshift(10)=0.d0    !!             !!    pulsestart < t < pulsestart + pi/w; 0 otherwise
real*8 :: chirp(10)=0d0          !!             !!
real*8 :: ramp(10)=0d0
real*8 :: longstep(10)=1d0       !!             !!  Pulsetype 3 available: monochromatic, sinesq start+end
!! NOW COMPLEX
DATATYPE :: pulsestrength(10)=.5d0 !!           !!  A_0 = E_0/omega (strength of field)
real*8 :: intensity(10)= -1.d0   !!             !! overrides pulse strength.  Intensity, 10^16 W cm^-2
real*8 :: pulsetheta(10)=0.d0    !!             !!  angle between polarization and bond axis (radians)
real*8 :: pulsephi(10)=0.d0      !!             !!  polarization in xy plane
real*8 :: maxpulsetime=1.d20     !!             !!
real*8 :: minpulsetime=0.d0      !!             !!  By default calc stops after pulse (overrides finaltime,
                                                !!   numpropsteps); this will enforce minimum duration
```

!! ACTIONS may also be specified by Act=X where X is an integer on the command line

```
integer :: numactions=0          !!
integer :: actions(100)=0        !!             !! ACTIONS

!!   Act=1    Autocorrelation; set corrflag=1 for fourier transform
!!   Act=2    Save natorbs
!!   Act=3    Save spfs
!!   Act=4    Save density
!!   Act=5    Save R-natorbs
!!   Act=6    Save projections of natural configurations  (with Mathematica data in NatCurves/)
!!   Act=7    Save curve data files in LanCurves/ for Mathematica plotting.
!!   Act=8    Enter plotting mode (do not run calculation) and plot natorbs
```

```
!!   Act=9    Enter plotting mode and plot spfs
!!   Act=10   Enter plotting mode and plot density
!!   Act=11   Enter plotting mode and plot natorbs in R
!!   Act=12   Enter plotting mode and plot projections from act=6
!!   Act=13   Nuclear FLUX
!!   Act=14   Enter plotting mode and analyze nuclear flux
!!   Act=15   Save ELECTRONFLUX
!!   Act=16   Enter plotting mode and analyze ELECTRONFLUX
!!   Act=17   Enter plotting mode and analyze ELECTRONFLUX (projected)
!!   Act=18   Plot denproj from act=6
!!   Act=19   Enforce natorbs between steps (experimental)
!!   Act=20   Overlaps with supplied eigenfunctions
!!   Act=21   Fourier transform dipole moment with pulse for emission/absorption
!!   Act=22   With Dirac Frenkel restrition, constraintflag=2, check norm of error - NEEDS
!!                dfrestrictflag=2 not 1
!!   Act=23   Enter plotting/analysis mode and read flux.bin files from Act=15 for overlaps
!!                between two time dependent wave functions
!!   Act=24   keprojector
integer :: nkeproj=200           !!  For keprojector
real*8 :: keprojminenergy=0.04d0 !!   "
real*8 :: keprojenergystep=0.04d0!!   "
real*8 :: keprojminrad=30        !!   "
real*8 :: keprojmaxrad=40        !!   "

integer :: hanningflag=0         !! for hanning window -- was corrflag
integer :: diptime=100           !! For act=20, outputs copies every diptime atomic units
integer :: dipmodtime=200        !! do ft every autotimestep*dipmodtime
integer :: numovlfiles=1
real*8 :: autopermthresh=0.001d0 !! Autoperm=
real*8 :: autonormthresh=0.d0    !!
real*8 :: eground=0.d0           !! Eground=     !! energy to shift fourier transform
complex*16 :: ceground=(0.d0,0d0)!!              !! input as complex-valued instead if you like
real*8 :: autotimestep=1.d0      !!
real*8 :: fluxtimestep=0.1d0     !!
integer :: nucfluxflag=0         !! 0 = both 1 =electronic 2= nuclear  NOT nuclear flux action 13,14
```

!! PHOTOIONIZATION (actions 15,16,17)

```
integer :: computeFlux=500, &      !! 0=All in memory other: MBs to allocate
     FluxInterval=50,&           !! Multiple of par_timestep at which to save flux
     nEFlux=1001,&               !! Number of energies in flux FT
     FluxSkipMult=1              !! Read every this number of time points.  Step=FluxInterval*FluxSkipMult
integer :: nucfluxopt=0          !! Include imaginary part of hamiltonian from nuc ke
integer :: FluxSineOpt=1,&       !! Use windowng function
     FluxOpType=1                !! 0=Full ham 1=halfnium
real*8 :: EFluxLo=0.01,&         !! Low energy boundary of F.T. (relative to eground)
     EFluxHi=2d0                 !! High energy boundary
integer :: FluxNBins=4           !! number of previous times to plot in xsec.spi.dat
```

!! PLOTTING OPTIONS

```
integer :: plotmodulus=10        !! PlotModulus= !! For saving nat/spf (Act=2, 6), par_timestep interval
real*8 :: plotpause=0.25d0       !! PlotPause=   !!    for saving natorbs.  plotskip is for stepping over
real*8 :: plotrange=0.2d0        !! PlotZ=       !!    the saved natorbs on read.  others are dimensions
real*8 :: plotcbrange=0.001d0    !!              !!
real*8 :: plotxyrange=2.d0       !! PlotXY=      !!
real*8 :: plotview1=70.d0        !!              !! viewing angle, degrees
real*8 :: plotview2=70.d0        !!              !! viewing angle, degrees
integer :: plotnum=10            !! PlotNum=     !! Max number of plots
integer :: plotterm=0            !!              !! 0=x11, 1=aqua
integer :: pm3d=1                !! PM3D         !! Turn pm3d on when plotting
integer :: plotres=50            !!              !! Resolution of plot
integer :: plotskip=1            !! PlotSkip=    !! For plotting (Act=3,5,7), number to skip over
```

```
real*8 :: povmult=1d0            !!  Mult df3 data by factor.  For small part of orbs.
integer :: povres=10             !!              !! Povray resolution
integer :: numpovranges=1        !!              !! number of magnifications to plot
real*8 :: povrange(10)=(/ 5,15,& !!              !! Povray plotting ranges (unitless - each magnification)
  80,80,80,80,80,80,80,80 /)
real*8  :: povsparse=1.d-3       !!              !! Sparsity threshold for transformation matrix in povray
```

!! MISC AND EXPERIMENTAL

```
logical :: readfullvector=.true.
logical :: walksinturn=.false.   !!              !! if you have problems with MPI i/o, maybe try this
integer :: turnbatchsize=5
integer :: nosparseforce=0       !!              !! to override exit with large number of configs, no sparse
integer :: iprintconfiglist=0
integer :: drivingflag=0                         !!  Solve for the change in the wave function not wave function
real*8 :: drivingproportion=0.999999999999d0     !!   -- "psi-prime" treatment.
integer :: noftflag=0            !!              !! turns off f.t. for flux. use for e.g. core hole propag'n.
integer :: timefacforce=0        !!              !!  override defaults
DATATYPE :: timefac=&            !! Prop/        !! d/dt psi = timefac * H * psi
        DATANEGONE               !!  Relax       !!
integer :: timedepexpect=0  !! expectation value of H_0(t) or H(t) reported
integer ::  checktdflag=0   !!  makes pulse constant and evaluates expectation value of h(t)
                            !!  to check energy conservation for dipole operators (consistency
                            !!  between reduced and matel)
integer :: dipolewindowpower=1   !!  multiply by cosine^dipolewindowpower for dipole ft
integer :: diffdipoleflag=1 !! fourier transform derivative of dipole moment not dipole moment

integer :: cmf_flag=1            !! CMF/VMF      !! CMF/LMF/QMF or VMF?
integer :: intopt=3              !! RK, GBS      !! SPF/VMF Integrator: 0, RK; 1, GBS, 2, DLSODPK
                                                 !!  for CMF: 3=expo 4=verlet
integer :: verletnum=80          !!              !! Number of verlet steps per CMF step
integer :: jacprojorth=0         !! 1: projector = sum_i |phi_i> <phi_i|phi_i>^-1 <phi_i|
                                 !! 0:             sum_i |phi_i> <phi_i|
integer :: jacunitflag=0         !! 1:  (1 x v)_j = sum_i |v_i><v_i|v_j>  for homogeneous third order
                                 !! 0: usual
integer :: jacsymflag=0          !! 3:  use 1-PHP  not (1-P)H
integer :: biocomplex=0          !! 1=old way complex zg/hpiv  0=always real
integer :: debugflag=0
real*8 :: debugfac=1d0
integer :: nonsparsepropmode=1   !! 0 = ZGCHBV expokit; 1 = mine expmat
```

```
!! PARAMETERS FILE FOR PROLATE SPHEROIDAL DVR BASIS and HAMILTONIAN :
!!            h2parinp NAMELIST input.
!!
module myparams
implicit none

integer :: debugflag=0

!! HAMILTONIAN
real*8 :: nuccharge1=1, &        !!              !! Nuclear charges
     nuccharge2=1                !!              !!
integer :: twoeattractflag=0     !!              !! make 1/r12 attractive
integer :: reducedflag=0         !!              !! Use reduced e^- masses?  Default 1 (yes) with bornopflag=0
integer :: bornopflag=1          !!              !! Born-Op calculation, or with nuclear KE?
                                 !!              !!   (Sets &parinp nonuc_checkflag=1.)
real*8 :: pro_Hmass=1836.152701d0!!              !! Masses of nuclei: pro_Hmass is the mass of nucleus one and
!! real*8 :: pro_Dmass=3670.483014d0
real*8 :: pro_Dmass=1836.152701d0!!              !!   pro_Dmass is the mass of nuc 2.
integer :: JVALUE=0              !!              !! J value for improved adiabatic

!! Additional atoms - hardwired Z=1 (hydrogen) for now
!! old way: hlocs puts hatoms on gridpoints, hlocrealflag=0
!! new way: turn on hlocrealflag; hlocreal is position r,theta (h2 or he)  phi not yet

integer :: numhatoms=0           !!              !! number of h atoms
integer :: hlocs(3,100)=1        !!              !! dimension (3, numhatoms): first 2 indices xi,eta gridpoint
                                 !!              !! then -1 or 1 for left or right -- all h's in plane for now
integer :: hlocrealflag=0
real*8 :: hlocreal(2,100)=0d0

!! BASIS

integer :: lbig=3,&              !!              !! Number of points in eta minus 1
     mbig=0                      !!              !! Number of m-values exp(imphi)

!! DVR / FEM-DVR for R
!! R
integer :: rnumelements=1        !!              !!  Number of elements in R
integer :: rcelement=100         !!              !!  First element at which ecs begins
real*8 :: rthetaecs=0.d0         !!              !!  ECS scaling angle
integer :: rnumpoints=3          !!              !!  Number of points per element including endpoints in R
real*8 :: relementsize=2d-16     !!              !!  Size of R elements
real*8 :: rstart=1.4d0           !!              !!  First (excluded) gridpoint in R
integer :: capflag=0             !!              !!
real*8 :: capstrength=0.d0       !!              !!
integer :: cappower=2            !!              !!

!! XI / r  radial electronic DOF

integer :: xinumpoints=14        !! Numpts=      !!  Number of points per element including endpoints in xi
integer :: xinumelements=2       !! Numel=       !!  Number of elements in xi
real*8 :: xielementsizes(100)=5  !!              !!  Sizes of elements
integer :: xicelement=100        !! Celement=    !!  First element which is scaled
real*8 :: xiecstheta=0.157d0     !!              !!  Scaling angle, radians

!! ORBITAL INITIALIZATION

integer :: num_skip_orbs=0                       !! For skipping core orbitals for initial diagonalization.
integer :: orb_skip_mvalue(100)= 99              !! Input how many you want to skip, their index (orb_skip)
integer :: orb_skip(20)= -1                      !!   (order in energy, each m value) and m-value
```

```
!! PARAMETERS FILE FOR ATOM DVR INPUT:  heparinp NAMELIST input.

module myparams
implicit none

integer :: debugflag=0

!! HAMILTONIAN AND BASIS

integer :: hecelement=100,&      !! Celement=    !! Frist element that is scaled
     henumelements=2,&           !! Numel=       !! Number of elements in r
     henumpoints=14              !! Numpts=      !! Number of points per elements
real*8 :: heelementsizes(100)=5,&!!              !! Sizes of elements in r
     heecstheta=0.157d0          !!              !! ECS scaling angle
integer :: lbig=3,mbig=0
real*8 :: nuccharge1=1d0

!! ORBITAL INITIALIZATION

integer :: num_skip_orbs=0                      !! For skipping core orbitals for initial diagonalization.
integer :: orb_skip_mvalue(100)= 99             !! Input how many you want to skip, their index (orb_skip)
integer :: orb_skip(20)= -1                     !!   (order in energy, each m value) and m-value
```

```
!! PARAMETERS FILE FOR SINC DVR POLYATOMIC BASIS AND HAMILTONIAN:
!!            sincparinp namelist input.

module myparams
implicit none

!! FOR TOTAL ORBITAL PARALLELIZATION, SET orbparflag=.true., AND parorbsplit=3 in &parinp
logical :: orbparflag=.false.
!! integer :: orbparlevel=3  !!  in namelist, but not in myparams

!! THE FOLLOWING FLAG IS THEN RELEVANT.  Option for parallel KE matvec, rate limiting step.
integer :: zke_paropt=1   !! 0=sendrecv 1=SUMMA (bcast before) 2=reduce after

!! fft_batchdim: determines batch size for matrix elements and
!! fft_circbatchdim: determines sub batch size for FFT
!!    defaults set small (less memory, more MPI messages) to avoid MPI problems when doing large
!!    calculations.  Otherwise bigger values will be faster.  There is a message size sweet spot
!!    on many machines.
integer :: fft_batchdim=1     !! 1 = do nspf matrix elements in nspf batches (less memory)
                              !! 2 = do nspf^2 in one batch (faster unless MPI problems)
integer :: fft_circbatchdim=1 !! 0,1,2, circbatchdim < batchdim; larger faster unless MPI problems
integer :: fft_mpi_inplaceflag=1     !! fft_mpi_inplaceflag:
                                     !!  0 = out-of-place fft, out-of-place fft inverse
                                     !!      3d FFT + (summa/circ C.T. depending on fft_ctflag)
                                     !!  1 = 3 x (1d FFT , all-to-all index transposition)
integer :: fft_ct_paropt=1           !! fft_ct_paropt, relevant if fft_mpi_inplaceflag=0
                                     !!    like zke_paropt: 0 = sendrecv 1 = summa
integer :: num_skip_orbs=0
integer :: orb_skip(200)=-1

integer :: toothnsmall=40
integer :: toothnbig=240

integer :: numcenters=1
integer :: centershift(3,100)=0    !! grid point index for each center
real*8 :: nuccharges(100)=2d0

integer :: numpoints(100)=15
real*8 :: spacing=0.25d0

integer :: orblanorder=500         !! krylov order for block lanczos orbital calculation
integer :: orblancheckmod=10       !! check every
real*8 :: orblanthresh=1d-4

integer :: capflag=0               !! Number of complex absorbing potentials
integer :: capmode=0               !! Capmode=1 is
integer :: cappower(100)=2         !!   v_i(r)= capstrength_i*(r/capstart_i)^cappower_i
real*8 :: capstart(100)=0.001d0    !! Capmode=0 is
real*8 :: capstrength(100)=0.01d0  !!   v_i(r)= capstrength_i*max(0,r-capstart_i)^cappower_i
real*8 :: mincap=0d0 , maxcap=1d30 !! V_CAP = -i* max(mincap,min(maxcap,sum_i v_i))

integer :: maskflag=0
integer :: masknumpoints=0

integer :: scalingflag=0           !! 1 = SMOOTH EXTERIOR COMPLEX SCALING
real*8 :: scalingdistance=10000d0  !! atomic units (bohr)
real*8 :: smoothness=5             !! atomic units (bohr)
real*8 :: scalingtheta=0d0         !! scaling angle
integer :: scalingorder=2          !! should be 2 or greater!
 real*8 :: tinv_tol=1d-3
```

Chapter 3

Output

The code outputs a substantial amount of information to screen, which can be redirected e.g.

```
chmctdhf_diatom Inp=Input.Inp.Relax |tee Outs/Out.relax
```

During the main propagation loop it shows the expectation value of the energy and the norm. By default the energy is the expectation value of the field-free Hamiltonian H_0. With the variable `timedepexpect=1` set in `&parinp`, it will give the expectation value of $H(t) = H_0 + V(t)$.

```
T=          1.16000  Energy:  -0.1170575477E+01  0.3036152171E-10    Norm:   0.1000000000E+01
T=          1.18000  Energy:  -0.1170575477E+01  0.3036153835E-10    Norm:   0.1000000000E+01
  Saving natorb !          18           3
  Saving spf !          18
T=          1.20000  Energy:  -0.1170575477E+01  0.3036156777E-10    Norm:   0.1000000000E+01
```

The code saves the wave function in `Bin/spfs.bin` and `Bin/avector.bin` at the end of the calculation; it will also save at intervals during the propagation if you set `saveflag=1`. The code makes the directories

```
WALKS
Flux
Dat
Bin
timing
```

Some large files may end up in these directories. Beware of this. On a supercomputer e.g. NERSC machines, you **MUST** be reading and writing these files to the `$SCRATCH` directory. If one prefers to run in the home file system one may make symbolic links to the scratch directory. E.g.

```
> mkdir $SCRATCH/blah5
> ln -s $SCRATCH/blah5 ./WALKS
```

In `WALKS` it saves `walks.BIN`, which contains all the information about configurations and spin eigenfunctions, after that information is calculated at the start. Often this initial setup takes a long time; if you rename `walks.BIN` to `savewalks.BIN`, it will read the latter when you run a calculation, instead of recalculating the information. That is the reason for saving these large files. Unfortunately there is no option to disable this functionality (writing `WALKS`) a this time (TO-DO ITEM). It is possible that `ln -s /dev/null ./WALKS` might work on some systems.

Otherwise, the variable `notiming` sets the amount of output. `notiming=2`, least amount of output, is default. `notiming=0` gives the maximum including all timing information.

notiming=2	No optional output
Dat/Pulse.Dat	If tdflag=1, the pulse waveform A(t) or V(t)
Dat/PulseFT.Dat	Its F.T.
notiming=1	In addition:
denmat.eigs.dat	Orbital occupation numbers.
rdenmat.eigs.dat	R-natorb occupation numbers.
rdenmat.expect.dat	Expectation value of bond length for each R-natorb.
PsiStats.Dat	Expectation values of various operators for atom
DiatomStats.dat1	For diatom
notiming=0	In addition: timing and other information
expo.dat	Information about exponential propagation of orbitals
abstiming.dat	Time stamp (wall clock) each step
Main.time.dat	Overall timings of all parts of calculation.
cmf_prop.time.dat	Timings of steps of orbital propagation
actreduced.time.dat	Timings of action of reduced operator during expo prop
actreduced.eops.time.dat	Further breakdown of operator timings
cmf_aprop.time.dat	Timings for a-vector propagation
reducedham.time.dat	Timings for reduced hamiltonian construction
matel.time.dat	Timings for calculation of matrix elements
twoe.time.dat	Breakdown for two-electron part
Actions.time.dat	Timings for actions

3.1 Action output (incomplete list)

Various actions that can be performed, described below, like overlaps, produce output. The main output files are listed below. They can be set in namelist &parinp using the variable name in the left column.

fluxmofile	Bin/flux.mo.bin	Molecular orbitals from Act=15 for photoionization calculation Act=16 or 17
fluxafile	Bin/flux.avec.bin	A-vector for photoionization calculation
spifile	Dat/xsec.spi.dat	Cross section from analysis of total photoionization (Act=16)
projfluxfile	Dat/xsec.proj.X.spi.dat	Partial cross section for state number X from projected photoionization (Act=17)
corrdatfile	Dat/Correlation.dat	Autocorrelation function for Act=1
corrftfile	Dat/Corrft.dat	Its fourier transform
zdftfile	Dat/ZDipoleft.Dat	Fourier transforms of field E and induced dipole moment D for absorption. Colum D(omega), E(omega), D(omega)* x E(omega) * is complex conjugate. Thus absorption is column 7.
zdipfile	Dat/ZDipoleexpect.Dat	Induced dipole moment D(t)
ovlfile	Overlaps.Dat	For Act=20, overlaps, these are the overlaps.

Less frequently used:

`Natlorb.bin`	Natural orbitals from Action 2; read by Action 8.
`Spfplot.bin`	Similarly, with Act=3 and 9.
`Density.bin`	Similarly, with Act=4 and 10.
`RNatorb.bin`	Similarly, with Act=5 and 11.
`Natproj.bin`	Similarly, with Act=6 and 12.
`NatCurves/`	If Act=6, `NatCurves/` contains data files and `MC-Nat.txt1`,
`MC-Nat.txtx`	`MC-Nat.txt2`, etc. contain mathematica input.
`LanCurves/`	If Act=7, `LanCurves/` contains data files and `MC-Lan.txt1`,
`MC-Lan.txtx`	`MC-Lan.txt2`, etc. contain mathematica input.
`Natorb/`	If Act=8, plotting natural orbitals, and Povray output is chosen, the data files go in this directory.
`Spfs/`	Similarly, with Act=9.
`Density/`	Similarly, with Act=10.

Chapter 4

Types of calculation

There are five general types of calculation that are generally performed:

- Relaxation for initial state
- Relaxation for many states for projection or overlap (called "mcscf" here)
- Propagation with a pulse
- Propagation without a pulse - usually done only for photoionization calculation, after first running pulse
- Analysis, for flux, plotting, and other tasks that use an already calculated wave function: will be described in actions section

4.1 Relaxation calculation

In a relaxation calculation the orbitals are propagated in imaginary time and the configuration coefficients are obtained as eigenvectors. The eigenvectors may be obtained through sparse diagonalization or, if an excited state is desired, by inverse iterations using `improvedquadflag`.

The most important options for relaxation are

```
loadavectorflag=0
mcscfnum=3
improvedrelaxflag=2
improvednatflag=1
improvedquadflag=1
stopthresh=1d-8
```

Given those input parameters, the code will diagonalize the Hamiltonian to start, take eigenvectors 2-4, then use inverse iterations on those vectors as the orbitals are optimized to convergence, always constraining the orbitals to be the averaged natural orbitals of the three states.

The three main types of relaxation calculation possible are:

- Regular time relaxation: solution of time dependent Schrodinger equation in imaginary time for ground state. Set `improvedrelaxflag=0`, `threshflag=1` in `&parinp`.

- Improved relaxation: `improvedrelaxflag>1`. Orbitals are propagated in imaginary time and A-vector hamiltonian is diagonalized to obtain the wave function for a ground or excited state. Set by command line option `>mctdh Relax=N` or variable `improvedrelaxflag=N` to calculate the Nth eigenvector. Root flipping problems may emerge, in which case use:
- Iterative solve: `improvedquadflag` can be set to 1 to perform an iterative solve to lock onto an eigenvector, from a good initial guess. For instance, one could converge the first 5 roots using MCSCF, then optimize the 5th with a calculation on that state only using `improvedquadflag=1`.

One would want to adjust the variables `par_timestep` (to a lower value than the default 0.5) and `stopthresh` (convergence parameter) for difficult cases.

- `improvednatflag=1`:

  One can rotate the orbitals to be natural orbitals during a relaxation calculation using `improvednatflag=1`. Thus the final orbitals will be natural orbitals. This is useful if one is to subsequently excite or annihilate the orbitals using `avectorexcite` or `avectorhole`. Be sure to check the output in denmat.rotate.dat (second of three columns) to verify that the natorbs are correct. This column should be near zero. If not you should check your orbital integration tolerance (`expotol` usually). Sparse A-vector eigenfunctions and solve may need increased tolerances too. For difficult cases use nonsparse if possible. `improvednatflag` is compatible with `improvedquadflag`; the iterations, energies, most things should be same. Improvednatflag should not hurt the performance of improvedquadflag.

Using `mcscfnum` > 1, one can perform a state-averaged MCSCF calculation, and calculate more than one A-vector for the one set of orbitals, for any of the three ways of relaxing mentioned above. This is most commonly done to obtain wave functions for analysis, projected flux (Action #17) or overlaps (Action #20). Just set variable `mcscfnum` to the number of states desired.

The default is to use core orbitals and eigenfunctions. However, one may wish to read in an arbitrary set of initial wave functions. For instance, one may wish to converge the first five states including an excited Π state using relaxation, then select one component of that Π state and use `improvedquadflag`=1 or 3 to iteratively refine that state.

There are several options available for reading in initial orbitals and wave functions. Consider this contrived example: we start with a 12 orbital calculation, done on three sigma states. Then use those orbitals, loading them, and freezing them with `spf_flag=0`, in calculations on pi and delta states. Then, perform an 11 orbital, three state averaged calculation on the third sigma state, one pi state, and one delta state, discarding the 10th orbital, keeping 1-9, 11, 12. The input for that run would include:

```
&parinp
  mcscfnum=3
  loadspfflag=1
  spffile="Bin/spfs.bin.sigma"
  num_skip_orbs=1
  orb_skip=10     CHECK THIS
  loadavectorflag=1
  numavectorfiles=3
  avectorfile="Bin/avector.bin.delta","Bin/avector.bin.pi","Bin/avector.bin.sigma"
  avecloadskip=0,0,2
  improvedquadflag=1
  spfrestrictflag=1
  mrestrictflag=0
  mrestrictmin=0
  mrestrictmax=2
```

Because the lowest states aren't being calculated, `improvedquadflag`=1 or 3 must be invoked, otherwise the 3rd sigma state will collapse to the 1st.

When doing a MCSCF calculation, one often wants wave functions with different M-values. Thus, `mrestrictflag=0` must be set to include all these configurations. However, `spfrestrictflag` should in most cases be set to one so that the orbitals have good m quantum numbers. And, one can use `mrestrictmin` and `mrestrictmax` as above.

4.1.1 Utility

There is a program `MCSCF-matel` that may be compiled and used to compute matrix elements between mcscf wavefunctions (in `mcscf.bin` files). These can have different orbitals. The program takes the same input as the main program, and in addition the following input:

```
&mcminp
  nummcmfiles=5
  mcmfiles="mcscf.bin.holecationdoubletset.2","mcscf.bin.holecationdoubletset.3", \
           "mcscf.bin.holecationdoubletset.4","mcscf.bin.holecationdoubletset.5", \
           "mcscf.bin.holecationdoubletset.6"
/
```

One should check that the usual namelist input includes the information defining configurations and primitive basis; other parameters are not used. The program produces various files:

`MCMatel.Dat`	Energies
`MCMatel.Ovl.Dat`	Overlap matrix elements
`MCMatel.ZDipole.Dat`	Dipole matrix elements, parallel
`MCMatel.XYDipole.Dat`	Dipole matrix elements, perpendicular

The namelist variable `mcmskip` can be set to one to just produce the `MCMatel.Dat` file.

4.2 Pulse calculation

One specifies a pulse calculation using `tdflag=1` in namelist `&parinp`, or `Pulse` on the command line. Namelist `&pulse` is then required, in which one specifies the pulse parameters. The calculation is stopped after the pulse is finished (`finaltime` in `&parinp` is overridden), except if `minpulsetime` or `maxpulsetime` is set in namelist `&pulse`. One may include multiple pulses.

```
&pulse
  numpulses=2
  pulsetype=3,1
  omega=0.152,0.1
  omega2=0.6,0.3
  intensity=1d-3,1d-2
  pulsetheta=0d0,0d0
  pulsephi=0d0,0d0
  phaseshift=0d0,0.3d0
  chirp=1d0,-1d0
  ramp=0d0,1d0
  longstep=3d0
/
```

The intensity is in units of 10^{16} W cm^{-2}. It is converted to `pulsestrength`, which is the coefficient of the pulse waveform in the velocity gauge, and which may alternately be specified in the namelist. The pulse waveform is output in `Pulse.Dat` and its Fourier transform in `PulseFT.Dat`. The window for the output of the fourier transform is set with variables `efluxlo` and `efluxhi` in `&parinp`.

`chirp` and `ramp` are only available for velocity. `chirp` has units of energy and `ramp` is unitless.

The pulse types are as follows: NOT ALL CORRECT FIXME FIXME FIXME

$\omega(t) =$ `omega2` + `chirp` $\times(t - \pi/$`omega`$/2)/(\pi/$`omega`$)$

$A_0(t) =$ `pulsestrength` $\times(1+$ `ramp` $\times\ (t - \pi/$`omega`$/2)/(\pi/$`omega`$/2)$

`pulsetype=1` $A(t) = A_0(t)\sin($`omega` $t)^2$ $t < \pi/$`omega`

`pulsetype=2` $A(t) = A_0(t)\sin($`omega` $t)^2 \times \sin(\omega(t)\ t +$ `phaseshift`$)$ $t < \pi/$`omega`

`pulsetype=3` $A(t) = A_0(t)\sin(\omega(t)\ t +$ `phaseshift`$)\ \times$

$$\begin{cases} \sin((2\ \texttt{longstep}+1)\texttt{omega}\ \ \mathrm{t}\ )^2 & (t < \pi/(2\ \texttt{omega}/(2\ \ \texttt{longstep}{+}1)) \text{ or} \\ & (t > \pi/\texttt{omega} - \pi/(2\ \texttt{omega}/(2\ \ \texttt{longstep}{+}1))) \\ 1 & \text{Otherwise if } t < \pi/\texttt{omega} \end{cases}$$

`pulsetype=4` $A(t) =$ `pulsestrength` $\times \sin($`omega2` $t+$`phaseshift`$)$

4.3 Propagation calculation, no pulse (fourier run)

For flux or other analysis one needs to propagate the wave function with no pulse, called here a fourier run. Specify `tdflag=0`, `improvedrelaxflag=0`, and `finaltime` in namelist `&parinp`.

4.4 Analysis / plotting

Certain actions (see below) correspond to analysis or plotting routines (actions 8-12, 14, 16-18, 23). If these actions are specified in the input file (`numactions` and `actions` in `&parinp`) or on the command line (e.g. `chmctdhf_diatom Inp=Inp.Inp Act=16 Eground=-49.545`) then the code does not perform propagation and instead terminates after the analysis or plotting routine is complete.

Chapter 5

Actions

We call various things that can be done to the wavefunction "actions." Some actions are performed during a MCTDHF calculation; some actions skip the MCTDHF calculation and are used only for analysis or plotting of a run that has already been performed or that is in progress. One can specify multiple actions. However, if an analysis/plotting action is specified, then the first such specified action will be the only action that is performed.

```
> chmctdh Act=1 Act=2 Act=4...,
```

or in namelist `&parinp`, e.g. `numactions=3`, `actions=1,2,4`.

```
!!   Act=1    Autocorrelation;
!!   Act=2    Save natorbs
!!   Act=3    Save spfs
!!   Act=4    Save density
!!   Act=5    Save R-natorbs
!!   Act=6    Save projections of natural configurations  (with Mathematica data in NatCurves/)
!!   Act=7    Save curve data files in LanCurves/ for Mathematica plotting.
!!   Act=8    Enter plotting mode (do not run calculation) and plot natorbs
!!   Act=9    Enter plotting mode and plot spfs
!!   Act=10   Enter plotting mode and plot density
!!   Act=11   Enter plotting mode and plot natorbs in R
!!   Act=12   Enter plotting mode and plot projections from act=6
!!   Act=13   Nuclear FLUX                       DEPRECATED
!!   Act=14   Enter plotting mode and analyze nuclear flux
!!   Act=15   Save ELECTRONFLUX
!!   Act=16   Enter plotting mode and analyze ELECTRONFLUX
!!   Act=17   Enter plotting mode and analyze ELECTRONFLUX (projected)
!!   Act=18   Plot denproj from act=6
!!   Act=19   Enforce natorbs between steps (experimental)
!!   Act=20   Overlaps with supplied eigenfunctions
!!   Act=21   Fourier transform dipole moment with pulse for emission/absorption
!!   Act=22   With Dirac Frenkel restrition, constraintflag=2, check norm of error - NEEDS
!!                 dfrestrictflag=2 not 1
!!   Act=23   Enter plotting/analysis mode and read flux.bin files from Act=15 for overlaps
!!                 between two time dependent wave functions
!!   Act=24   keprojector
```

5.1 Autocorrelation, action 1

Fourier transforms of autocorrelation functions are calculated by using `&parinp` namelist variables `corrflag` to 1 and setting an action to 1. On the command line these can be done with `Act=1`.

The autocorrelation is performed at the same time as the propagation, so these settings should go in that input file. The autocorrelation function is output in `Correlation.Dat` and its Fourier transform is output in `Corrft.Dat`. The zero of energy is set by `eground`.

5.2 Saving orbitals (actions 2-4) and plotting orbitals (actions 8-10)

See subsection 6.2.

5.3 Flux and projected flux for photoionization (actions 15, 16, 17)

To calculate photoionization use the flux and projected flux option. These are specified by actions 15, which saves the data during a propagation run, and actions 16, and 17, which compute total and projected flux, respectively. The analysis routines can be run before the calculation is finished; just specify `T=` on the command line to ensure you don't read past what's written. In other words you can run a flux calculation in the same working directory in which the correlation calculation is running.

`&parinp`	
`fluxmofile`	Output binary file for orbitals. Default "flux.mo.bin"
`fluxafile`	Output binary file for the configuration coefficients. Default "flux.avec.bin"
`spifile`	Output for the analysis run (act=16): total photoionization. Default "xsec.spi.dat"
`eground`	Ground state energy. Important! Sets zero of photon energy for cross sections.
`computeFlux`	Batch memory size (MB) 0=all in core

The range and resolution of output for actions 16 and 17 is set as follows,

`nEFlux`	Number of energies at which to calculate the flux
`EFluxLo`	Lowest photon energy
`EFluxHi`	Highest photon energy ()

The photon energy is defined as the difference between the absolute energy and `eground` so be sure to set it!

There are parameters that can be used to control the biorthogonalization for the linear solve via $X = \exp(-\ln A)B$ using the `expokit` routine. These may not have a big effect but may need to be varied if the flux calculation is taking an extremely long time.

```
&parinp
biodim
biotol
biocomplex
```

5.3.1 Options for saving wave function during propagation (Action 15)

In order to save the wavefunction you must specify action 15 during the propagation run; this propagation in which you save the wave function is often what we call a Fourier run. The wave function is saved in files `Flux/flux.mo.bin` and `Flux.avec.bin`, which filenames can be changed via `&parinp` namelist input variables `fluxmofile` and `fluxafile`, respectively. So that's `Act=15` on the command line or namelist input as follows. When reading, action 16 or 17, you also want to specify or check in the input that `FluxSkipMult` > 1, if you have saved wave functions, given `par_timestep` and `FluxInterval` in

the `Act=15` Fourier run that saves the wave function, more frequently than is required, to compute the flux-flux correlation function at a high enough photon frequency for your purposes.

```
&parinp
numactions=1
actions=15
```

`FluxInterval` Multiple of `par_timestep` at which the wave function is saved

The files `Flux/flux.mo.bin` and `Flux.avec.bin` **MAY BE VERY VERY LARGE**, especially if you are not careful. They contain the entire wave function propagated for the entire run. Do not use an excessively small save interval or you will waste large quantities of disk space. The save interval is `FluxInterval` times `par_timestep`. The `Flux` directory is separate for a reason! Please note the section at the top of the document about "filesystem etiquette." On a supercomputer with multiple users, ensure that you do not write, e.g., `WALKS`, `timing`, and `Flux`, to the `home` filesystem; these should always be put on scratch. Many users prefer to operate on `$SCRATCH` entirely; that is the most responsible option. Others run in a working directory in the `home` filesystem, such that the main results are preserved, and symbolically link the directories to scratch. The script `findwork` is included; this finds a new directory and symbolically links it as `WORK` in the working directory. One may run

```
prompt> rm -r Flux timing WALKS
prompt> findwork
prompt> mv WORK Flux
prompt> findwork
prompt> mv WORK timing
prompt> findwork
prompt> mv WORK WALKS
```

et cetera (perhaps also `Dat`). The script `findwork` follows:

```
# lawrencium
scratchdir=$SCRATCH
#
# nersc
# scratchdir=$GSCRATCH
i=0; flag=0
while [[ $flag == 0 ]]
do
        (( i++ ))
        file="$scratchdir/WORK$i"
        if [[ ! -e $file ]]
        then
                flag=1
        fi
done
mkdir $file
# or not
rm -r WORK
ln --symbolic $file ./WORK
```

5.3.2 Total photoionization (Action 16)

```
&parinp
```

`FluxSkipMult` Interval for flux read (F.T. timestep=`par_timestep` x `FluxSkipMult` x `FluxInterval`)

`fluxoptype` Approximation to Im(H): 0=no approx 1=halfnium potential approx (default)

(The option `fluxoptype=0` is the exact expression, but it has not been used recently and may be buggy. TO-DO DEBUG.)

To calculate flux you run a calculation with action 16. For instance, say you have `Input.Inp.Fourier` that you are using for the propagation run (which you set with action 15). You could be running

```
> chmctdhf_sinc Inp=Input.Inp.Fourier |tee Outs/Out.Fourier
```

Let's say it has gotten to 300 time units. In the same directory you could run

```
> chmctdhf_sinc Inp=Input.Inp.Fourier Act=16 T=300 |tee Outs/Out.Flux.T300
```

The flux calculation produces `xsec.spi.dat` (spi for single photoionization). This filename may be changed by `&parinp` namelist variable `spifile`.

The analysis of the wave function for photoionization is done separately from the propagation. During propagation, with `Act=15`, the only thing that is done is the wave function is saved. It is saved every `FluxInterval` times `par_timestep` atomic units.

NO - DEPRECATE FLUXINTERVAL AS INPUT - DO FLUXTIMESTEP

Photoionization cross sections are calculated with e.g.

```
chmctdhf_diatom Eground=-99.80004 Inp=Input.Inp.Fourier Act=16
```

for total photoionization.

The output in `Dat/xsec.spi.dat` is the quantum mechanical cross section in Megabarns (10^{-18} cm^2):

(5.1) *equation*

The cross section is in column 3 and the photon energy in Hartree is in column 1 in the file `Dat/xsec.spi.dat`.

Note that the equation does not include a factor of $\frac{1}{3}$. The perpendicular and parallel cross sections for fixed nuclei H_2, just like the physical cross section for the molecule, is approximately 12 Megabarns at onset. Many definitions include a factor of $\frac{1}{3}$ and sometimes $\frac{2}{3}$ for the perpendicular fixed-nuclei cross section, such that the total cross section is variously the sum or the 1:2 weighted sum of the parallel and perpendicular cross sections, but we simply report the quantum mechanical cross section, that which becomes the classical cross section for the problem as defined, regardless of the dimensionality of nuclear motion that is included.

Remember, the photon energy is defined relative to `eground`. See the examples section for the script `Flux.Bat`, which reads `eground` by `grep`ping from a computed output file from a relaxation run, automating the execution of the `chmctdhf Eground=` command above.

5.3.3 Partial photoionization (Action 17)

`&parinp`
`FluxSkipMult` Interval for flux read (F.T. timestep=`par_timestep` x `FluxSkipMult` x `FluxInterval`)

In order to calculate projected flux one must calculate the final $(N-1)$-electron states for the projection. These must be saved in the files `Bin/cation.spfs.bin` and `Bin/cation.avector.bin`. The setting of these filenames is not supported at this time.

One does this with a relaxation run. After you perform the $(N-1)$-electron relaxation, one must rename the binary files so that they may be read by the flux calculation. You add the prefix "`cation.`" at the start of the filename. Thus you would have `cation.spfs.bin` and `cation.avector.bin`.

Projected flux produces files `xsec.proj.spi.datN` with `N` being the final state.

5.4 Calculating overlaps with given wave functions (action 20)

Using action 20 one may read in wave functions at the start of the calculation and calculate their overlaps with $\Psi(t)$. Perform one or more relaxation calculations to obtain the states upon which you want to project. Then, to read them in your propagation run, include the following.

```
&parinp
  numactions=1
  actions=20
  numovlfiles=2
  ovlspffiles(1:2)="spfs.bin.sigma","spfs.bin.pi"
  ovlavectorfiles(1:2)="avector.bin.sigma","avector.bin.pi"
```

5.5 Fourier transform of dipole moment for absorption (action 21)

The absorption and emission spectrum is calculated via the "response function"

$$S(\omega) = 2\omega Im(D(\omega)E^*(\omega)) \tag{5.2}$$

In which $D(\omega)$ is the fourier transform of the induced dipole moment, and $E^*(\omega)$ is the complex conjugate of the Fourier transform of the field.

5.6 Overlaps between wave functions on file (actions 15, 23)

xxx

5.7 Keprojector (action 24)

xxx

5.8 Unsupported/deprecated actions

Other saving (5-7) and plotting actions (11,12,18): Action 5 (R natural orbitals) may work; otherwise these options are not actively supported. 12 and 18 are redundant. Nuclear flux and plotting it (actions 13,14) are deprecated, not supported.

Chapter 6

Viewing the output

6.1 Two auxillary files

There are two files which need to be included in the working directory for some of the plotting routines to work, for actions `Act=`6, 7, 8, 9, and 10. This functionality and the need for these files can be eliminated by answering no to the questions in the `Install` script that ask about Povray and Mathematica. If the files are not present in the working directory, the code will do what it can; you will still have the data files in `NatCurves/` and `LanCurves` for `Act=` 6 and 7, and you will have `.df3` files in `Spfs/`, `Natorb/`, and/or `Density/`, but Povray will not be called to produce `.tga` files, for `Act=` 8, 9, and 10.

Action		Required file
`Act=6`	Save natural projections	} MC-LinesQQQ.txt
`Act=7`	Save B.O. curves	
`Act=8`	Read spfs	} Density.Bat if ploting with Povray, not gnuplot
`Act=9`	Read natorbs	
`Act=10`	Read density	

6.2 Viewing natural orbitals, orbitals, density, R-natural orbitals, and projections of natural configurations

The code is built with a plotting mode (controlled by internal variable `skipstuff`) in which it reads the input as normal, but skips various parts of the setup and then goes directly to an interactive plotting subroutine. So if you want to plot any of these results, you re-run the code with the same input file and command line arguments, but additionally with one of the plotting mode actions specified. Viewing natural orbitals, orbitals, density, R-natural orbitals, and projections of natural configurations is accomplished by actions 8 through 12.

Action	Reads file	Comments
`Act=8`	`Natlorb.bin`	View natural orbitals using gnuplot or povray.
`Act=9`	`Spfplot.bin`	View orbitals using gnuplot or povray.
`Act=10`	`Density.bin`	View single particle density using gnuplot or povray.
`Act=11`	`RNatorb.bin`	View R-natural orbitals using gnuplot.
`Act=12`	`Natproj.bin`	View projections of natural configurations using gnuplot.

So you can re-run your calculation with `Act=`8, 9, 10, 11, or 12, respectively, as a(n additional) command line argument. You can do this as the MCTDHF calculation is running, if you'd like. Only one of

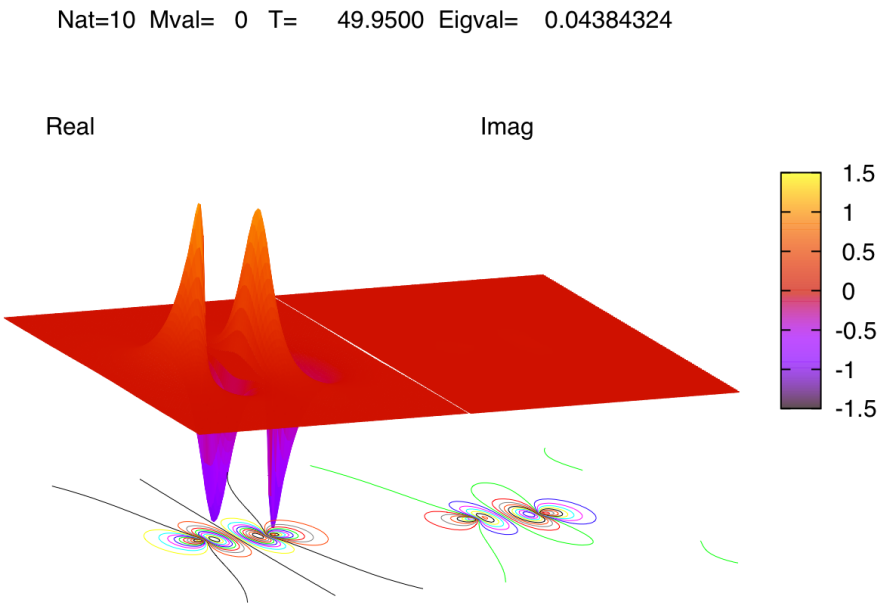


Figure 6.1: Example of gnuplot orbital plot.

these options may be used at a time. This works with the example directory scripts, so you can run `C:> Third.Bat Act=9` for instance. Otherwise for example

```
C:> pmctdh Inp=Input.Inp.myinput Act=10
```

Some of the usual output will appear on screen, and then, for instance,

```
*****************************
   VIEWING NATURAL ORBITALS
*****************************

USE POVRAY? y for yes.
```

At this point the plot subroutine branches depending on whether you want to use gnuplot or Povray to produce the output. Gnuplot may output to screen as either an X11 or AquaTerm plot, or as a .gif file, via the "change terminal" option. Povray produces .tga files which must then be assembled into an animation. If you answer no to "USE POVRAY?" then the following menu will appear for gnuplot:

```
 What should I do?
        s = change plotskip (now            1 )
        n = change plotnum (now           10 )
        z = change zrange (now    0.8000000000000000      )
        x = change xyrange (now    2.0000000000000000      )
         t = change terminal ( now x11    )
        d = change pm3d (now           0 )
       v1= change view rotation 1 (now    70.000000000000000      ) degrees
        v2= change view rotation 2 (now    70.000000000000000      ) degrees
  default = continue (plot with these options)
c
     Enter natorb number.  Negative to stop.
3
```

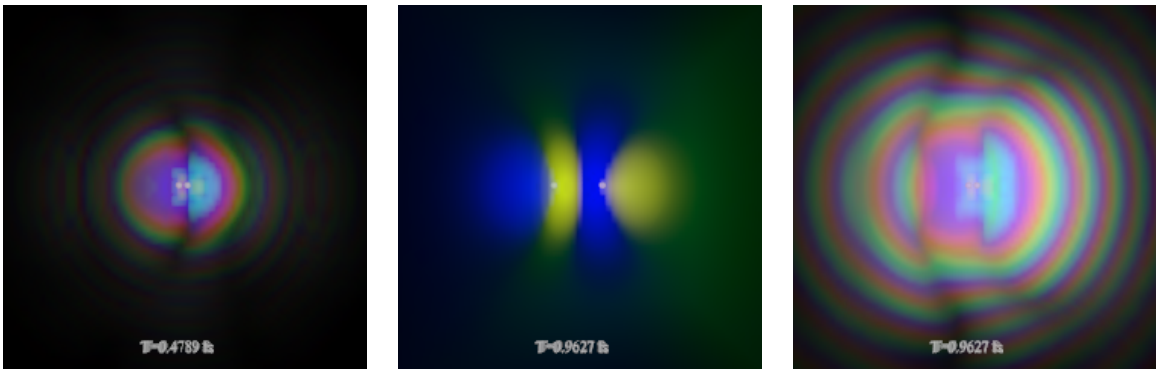


Figure 6.2: Example of Povray natural orbital plot: the tenth ($2\sigma_u$) orbital of a valence-space CAS calculation, directory O2.Small. Left: at 0.47 fs, from a range of 40 bond lengths (which are 2.2819... bohr); middle and right, 0.96fs, from a range of 7 (middle) and 40 (right) bond lengths. The middle and left plots are the same orbital at the same time.

If the options are ok you can enter any input except for the labeled options (for instance, type "c" then enter) to continue to plot. Choose the natural orbital or spf that you want to plot. The program will output some information to screen as it builds the text file for gnuplot. This may take a while. Then, a plot should appear, as in Figure 6.1. You then have the option to plot again.

If you choose yes to "USE POVRAY?" then you will instead only have the following options:

```
What should I do?
       s = change plotskip (now            1 )
       n = change plotnum (now           10 )
 default = continue (plot with these options)
```

If you need to change the other parameters for Povray plots, you will need to do this in the input file, in namelist `&parinp`:

```
&parinp
    povres=14
    numpovranges=2
    povrange=8.0d0, 50.0d0
    povsparse=1.d-3
    ...  /
```

`povres` determines the resolution of the `.df3` file; the data file is $(2\times$`povres`$+1)^3$ big. `numpovranges` is the number of distances, in units of $R\ a_0$, at which to view the orbitals or density, and the array `povrange` contains those distances. The parameter `povsparse` determines the zero cutoff for the prolate-to-cartesian transformation matrix – it is necessary to store this memory in sparse format for large `povres`. Select an orbital and the plotting subroutine proceeds:

```
 Enter Natorb     number.  Zero to change options.  Negative to stop.
1
 Good read at record            1
  Maxsparse ...
   Got maxsparse:           662316                     11523120.
   Got maxsparse:           183544                     11523120.
   Get spherical sparse.
      ...done.
  POV-plotting Natorb !!            0           1   8.0000000000000000
```

```
Persistence of Vision(tm) Ray Tracer Version 3.6.1 (/usr/bin/g++-4.2 4.2.1 @
 i386-apple-darwin10)
```

...and the povray output will continue. To plot all the natural orbitals or spfs, choose a number of orbitals greater than the number in the calculation. The program outputs .tga files, which may be assembled by you into an animation. The output should appear as in Fig. 6.2. The plot is three dimensional and colors the wavefunction according to its phase on the color wheel. Concentrated parts of the wavefunction may be saturated in what you see. In Fig. 6.2, the middle and left panels are the same orbital at the same time, but at different magnifications; the intensity range is adjusted to different levels according to the magnification of the plot (variable `povranges`).

Chapter 7

Examples

There are several example directories now included in the `EXAMPLES` directory at top level.

```
prompt> tar -xvf MCTDHF.010115.VERSION1.0.tar.gz
prompt> cd MCTDHF.010115.VERSION1.0
prompt> ls
   LBNL-AMO-MCTDHF.pdf
   EXAMPLES
      H2PHOTO
      HEPHOTO
      O2-photo
      Methane.HF.0.06
      Neon.HF.0.06
      He.transient.absorption
      O2.transient.absorption-ONE
```

etc. This chapter goes into these examples. In the example directories, you will find bash scripts for doing a variety of things. We imagine that users will adapt these scripts to their needs. They provide functionality for various things but most notably, using the method of Domcke et al. [CITE] for calculating macroscopically phase matched signals nonlinear wave mixing experiments.

7.1 Total and partial photoionization cross sections: directories H2PHOTO, HEPHOTO, O2-photo

A total or partial photoionization cross section calculation is performed in four or five steps. The same primitive basis should be used for all steps.

- 1) Relaxation for initial state, `Input.Inp.Relax` in the examples
- 2) For partial photoionization cross sections, one or more `Input.Inp.Cation`s must be run to produce the $(N-1)$-electron wave functions that are used for projection.
- 3) A short pulse propagation, `Input.Inp.Pulse` in the examples. At the end of the pulse, very little of the wave function should have been absorbed by the complex scaled part of the grid. Otherwise, it won't be counted as flux.
- 4) A long propagation that is used to construct the flux-flux correlation function as per the formalism of Meyer [CITE] and paper XX. The sharper the features in the photoionization cross section that are to be resolved, and the longer the real part of the grid is, the longer the propagation must be. Useful durations range from 100 to 8000 atomic units (2.5 to 20fs).

- 5) The final step is the analysis step. It is run separate from the propagation.

Again, all the inputs for the `EXAMPLES/` calculations are collected at the end of the chapter.

The `H2PHOTO` and `HEPHOTO` directories are very similar and include scripts for total photoionization. `O2-photo` also includes partial photoionization.

The MCTDHF calculation for the photoionization calculation is performed by the following three steps, in order:

```
prompt> chmctdhf_diatom Inp=Input.Inp.Relax |tee Outs/Out.Relax
prompt> chmctdhf_diatom Inp=Input.Inp.Relax |tee Outs/Out.Pulse
prompt> chmctdhf_diatom Inp=Input.Inp.Fourier |tee Outs/Out.Fourier
```

While the last is running, one may calculate the cross section so far computed. A script `Flux.Bat` is included in the example directories; here is is for `H2PHOTO`:

```
if [[ "$1 " == " " ]]
then
        echo "need time"
        exit
fi
ee=‘grep T= Outs/Out*ax |tail -n 1 |colrm 1 28 |colrm 21 10000 |cut -f 1 -d "E"‘
energy=‘echo "$ee * 10" |bc -l‘
chmctdhf_diatom Inp=Input.Inp.Fourier Act=16   Eground=$energy T=$1 |tee Outs/Out.flux.t$1
cp Dat/xsec.spi.dat Dat/xsec.spi.dat.t$1
```

For `O2-photo`, to calculate the cation wave functions for partial photoionization, one can use the script `RunCat.Bat` if working on a parallel machine (`RunCat.Bat` uses `mpirun` so that killing the `wait` command kills all of them). This does all of the cation wave functions at once. Otherwise, run each cation input, e.g. `Input.Inp.Cation.pi_g`, manually, or edit `RunCat.Bat` to your purposes. Here is `RunCat.Bat`:

```
for ext in doub quart; do for sym in sig pi; do for ger in u g
do
mpirun -n 1 chmctdhf_diatom.seq Inp=Input.Inp.Cation.$ext.${sym}_${ger} |tee Outs/Out.cation.$ext.${sym}_${ger} &
done; done; done
wait
```

Again, in these examples the `&parinp` namelist input variable `eground` is not set in the input, so one must specify it on the command line. Script `ProjFlux.Bat` in `O2-photo` takes care of this. Here is the guts of `ProjFlux.Bat`:

```
if [[ "$2 " == " " ]]
then
        echo "need time and extension"
        exit
fi
rm WALKS/cation.configlist.BIN
rm Bin/cation.spfs.bin Bin/cation.avector.bin
ln -s cation.configlist.BIN.$2 WALKS/cation.configlist.BIN
ln -s avector.bin.cation.$2 Bin/cation.avector.bin
ln -s spfs.bin.cation.$2 Bin/cation.spfs.bin
ee=‘grep T= Outs/Out*ax |tail -n 1 |colrm 1 28 |colrm 21 10000 |cut -f 1 -d "E"‘
energy=‘echo "$ee * 1000" |bc -l‘
chmctdhf_diatom Inp=Input.Inp.Fourier Act=17   Eground=$energy T=$1 |tee Outs/Out.projflux.t$1
for file in Dat/FTGTau*_*Dat Dat/KVLsum*_*dat Dat/xsec.proj*_*dat
do
        mv $file $file.$2
done
```

For O_2 partial photoionization there are many final cation states with different symmetries. However currently the projected flux can only be calculated for one set of cation states at a time. Always, it needs `WALKS/cation.configlist.BIN`, the cation configuration list, and the cation wave function(s), `Bin/cation.spfs.bin` and `cation.avector.bin`. These file names are hard wired in the code presently. In the input files that are run by `RunCat.Bat`, the output file names are specified, for instance,

```
configlistfile="WALKS/cation.configlist.BIN.doub.pi_g"
avectoroutfile="Bin/avector.bin.cation.doub.pi_g"
spfoutfile="Bin/spfs.bin.cation.doub.pi_g"
```

and so, as you can see above, what `ProjFlux.Bat` does is make symbolic links to these files.

7.2 Transient absorption/emission and wave mixing

The code calculates the expectation value of the dipole moment and fourier transforms it using action 21. The output can be post processed. The example directories `O2.transient.absorption-ONE`, `O2.transient.absorption`, and `Helium.transient absorption` are included in the `EXAMPLES` subdirectory. One XUV-IR time delay is used in the first; there are scripts in the last two directories compute to 2D spectra.

7.2.1 O2.transient.absorption.ONE

7.2.2 2D spectra for O_2 and He

Chapter 8

Programmers' guide

Again, the directory structure, right out of the package, is as follows.

```
MCTDH.SRC
   DFFTPACK
   DGMRES
   SINCDVR
   H2PROJECT
   HEPROJECT
COMPDIRS
   MCTDH.SRC -> ../MCTDH.SRC
   BIN.ecs.hermnorm.law.openmp.nofft
      Definitions.INC
      Name.Txt
      Makefile.header -> ../MCTDH.SRC/Makefile.header.lawrencium.openmp.nointelfft
      Definitions.ALL -> ../MCTDH.SRC/Definitions.ALL
      mctdhf.F90 -> ../MCTDH.SRC/mctdhf.f90
      DFFTPACK
         Makefile.header -> ../Makefile.header
         Makefile -> ../../MCTDH.SRC/DFFTPACK/Makefile
         zfftf1.f -> ../../MCTDH.SRC/DFFTPACK/zfftf1.f
      SINCDVR
         Makefile.header -> ../Makefile.header
         Makefile -> ../../MCTDH.SRC/SINCDVR/Makefile
         sincDVR.f90 -> ../../MCTDH.SRC/SINCDVR/sincDVR.f90
   debug.BIN.ecs.hermnorm.mac
   BIN.mac
   BIN.ecs.hermnorm.edison
```

Notice the following things.

- The `*BIN*` compilation directories are identical except for `Name.txt`, `Makefile.header`, and `Definitions.INC`.
- All of the subdirectories of these `BIN` directories are real subdirectories; they contain nothing but links.
- Notice that most of the symbolic links to fortran files end in `.F90` whereas all the fortran files themselves in `MCTDH.SRC` end in `.f90`. This is done to invoke the c preprocessor with some compilers (with the `.F90`) but the fortran interpreter with emacs (with the `.f90`).
- The compilation directories have Symbolic links and directory-specific `Name.txt`, `Makefile.header`, and `Definitions.INC`.

These BIN directories contain the compiled version of the code and all the object files and miscellaneous files created during compilation. They have subdirectories, symbolic links, and two real files, Definitions.INC and Name.Txt:

Name.Txt contains the name of the program version, e.g. chmcdthf for chmctdhf_diatom, etc.; alternatively mctdhf, pmctdhf, or cmctdhf; and is used by the Makefile. Definitions.INC is a header for most of the .f90 files that contains preprocesor directives that define data types and the corresponding LAPACK subroutines differently for the different BIN directories, implementing real or complex data types, ECS or no ECS, and hermitian norm or, for ECS, c-norm, mctdhf, chmctdhf, pmctdhf, cmctdhf. To effect the differences, the macros REALGO, ECSFLAG, and CNORMFLAG are either defined or undefined, and then there are conditional statements that do the rest of the work in Definitions.ALL.

```
Definitions.INC :

  #define REALGO
  #define ExxCSFLAG
  #define CxxNORMFLAG

  #include "Definitions.ALL"

Definitions.ALL :

  #ifdef REALGO

  #define DATATYPE real*8
  #define MYGEMM DGEMM
  ...
```

The fortran files then use these c preprocessor macros,

```
  #include "Definitions.INC"

  module automod
    implicit none

    DATATYPE, allocatable :: overlaps(:,:)
    integer, allocatable :: calledflags(:)
  ...
```

8.1 Main program files

The MCTDH.SRC directory contains the following files. They are grouped by nature and importance. parameters.f90 and main_modules.f90 are the main dependencies; most of the other files depend on these files, and none other. Again we reiterate that we sometime use the terms configuration and slater determinant interchangeably. Sometimes when we write configuration, we mean either a slater determinant or a spin adapted sum of slater determinants. But usually, we just mean slater determinant.

```
-rw-------  Definitions.ALL                x C preprocessor macros
-rw-------  Makefile.header.edison         x Makefile.headers
-rw-------  Makefile.header.mac.mpi.debug  x   . . . etc
-rw-------  Makefile                       x Makefile
-rwxr-xr-x  Makeme                         x Compilation script

drwx------  DFFTPACK                       x Fourier transform
drwx------  DGMRES                         x GMRES
drwxr-xr-x  COREPROJECT                    x coreproject.f90
```

```
drwx------  HEPROJECT                    x Atom project
drwx------  H2PROJECT                    x Diatom project

-rw-------  parameters.f90               x VARIABLES, Namelist &parinp
-rw-------  main_modules.f90             x Modules
-rw-------  getparams.f90                x Routine to load input file, otherwise set variables

-rw-------  mctdhf.f90                   x Main program
-rw-------  prop.f90                     x Core of main program
-rw-------  derivs.f90                   x Orbital propagation subroutines - working equation, call of expo_driver
-rw-r--r--  configstuff.f90              x Subroutine for configuration propagation and diagonalization

-rw-------  expo_driver.f90              x Orbital propagation subroutines - call of expokit

-rw-------  matel.f90                    x Orbital and configuration matrix element subroutines

-rw-------  mean.f90                     x Constructs 2-e reduced denmat
-rw-------  denmat.f90                   x 1-e reduced denmat and miscellaneous (denmat constraint for restricted
                                                                                configuration list)
-rw-------  walks.f90                    x Matrix elements among slater determinants
-rw-r--r--  walkmult.f90                 x Use them to multiply vector of configuration coefficients
-rw-r--r--  newconfig.f90                x Configuration subroutines (get configuration list, get configuration index)
-rw-------  spin.f90                     x Spin (S(S+1)) adaptation
-rw-------  spinwalks.f90                x Matrix elements of spin operator among configurations

-rw-r--r--  biortho.f90                  x Biorthogonalization workhorse

-rw-r--r--  driving.f90                  x Psi-prime treatment

-rw-------  quad.f90                     x Improvedquadflag - inverse iterations for diagonalization

-rw-------  second_derivs.f90            x Verlet intopt=4

-rw-r--r--  blocklanczos.f90             x Diagonalization

-rw-------  dfconstrain.f90              x Restricted configuration list - computation of g and tau, orbital derivatives

-rw-------  actions.f90                  x Action driver routines.
-rw-------  autocall.f90
-rw-------  autosub.f90
-rw-------  readactions.f90
-rw-r--r--  saveactions.f90
-rw-r--r--  natprojaction.f90
-rw-------  povactions.f90
-rw-r--r--  orbvectoractions.f90
-rw-r--r--  dipolecall.f90
-rw-------  dipolesub.f90
-rw-------  electronflux.f90
-rw-------  projeflux.f90
-rw-------  ovlsub.f90
-rw-r--r--  keprojector.f90

-rw-------  MPI.f90                      x Parallel subroutines
-rw-r--r--  proputils.f90                x Utilities, including pulse subroutines
-rw-r--r--  spfs.f90                     x Utilities relating to orbitals
-rw-------  eigen.f90                    x LAPACK wrappers
-rw-r--r--  utils.f90                    x Miscellaneous
-rw-r--r--  psistats.f90                 x Expectation values of wave function
-rw-r--r--  configload.f90               x I/O
-rw-r--r--  loadstuff.f90                x I/O

-rw-------  expokit.f                    x Exponential propagator for orbitals, also psi-prime & miscellaneous
```

```
                                                    Roger B. Sidje U Queensland        modified in key places by me
-rw-------  gaussq.f                      x Quadrature
-rw-------  jacobi.f                      x Quadrature
-rw-------  odex.f                        x Implicit integrator intopt=1
-rw-------  opkda1.f                      x General purpose solver intopt=2
-rw-------  rkf45.f                       x Runge-Kutta intopt=0
-rw-------  arg.c                         x Some c routines
```

8.2 Project directories

The project directories contain files required to build the different compiled versions of the code, running different coordinate systems. In the past, the code had a flag, and all coordinate systems were contained in one program; now it is different.

There are three projects currently, atom, diatom, and polyatomic, in `H2PROJECT`, `HEPROJECT`, and `SINCPROJECT`, respectively. The first two share one file; duplication is avoided by writing one file, `MCTDH.SRC/COREPROJECT/coreproject.f90`, that is shared. This is not the general operation; project directories, like `SINCPROJECT`, in general should be autonomous.

To add a new project directory, put the source code in a subdirectory of `MCTDH.SRC`, perhaps `MCTDH.SRC/NEWPROJECT`; then create a directory (not a link, `mkdir`) in one of the `BIN` directories, also called `NEWPROJECT`. Then, go into that directory and link each the files in `MCTDH.SRC/NEWPROJECT` to `BIN/NEWPROJECT`, with the same name except for capital-F `.F90` extension.

That should set up `BIN/NEWPROJECT`; then you can recursively copy `BIN.NEWPROJECT` to `BIN.ecs.hernorm.debug` or any other compilation directories that you have set up (keeping symbolic links as symbolic links!!) and they should be ready to go, if you did your `Makefile` in `NEWPROJECT` like it is in the existing project directories.

8.3 About the Makefile and Makeme

The code is meant to be compiled with the script `Makeme` which should be linked in the `BIN` directory in question. All it does is run `make` in the project directories, the other subdirectories, and then in the main directory. So if one adds a project, one adds a line in `Makeme`, to compile the code in the project directory that you have designed (with your makefile, which should be invoked by the "`make`" command that `Makeme` runs in your project directory). In the `Makefile`, one would add a new project called "newproject" as follows.

```
MYDEFAULT=$(NAME)_atom $(NAME)_diatom $(NAME)_sinc $(NAME)_newproject

NEWPROJECT=NEWPROJECT/myprojectar.a
HEPROJECT=HEPROJECT/heprojectar.a
H2PROJECT=...
```

and an instruction lower down,

```
$(NAME)_newproject:   $(SRCS) mctdhf.o  $(NEWPROJECT)
        $(F90) $(LOADFLAGS) -o $(NAME)_newproject  $(openmp) $(DFFTPACK) $(SRCS) mctdhf\
.o $(bessrcso) $(DFFTFILES) $(dgmfiles) $(dgmparfiles) $(NEWPROJECT) ...
```

The makefile starts out by including the file `Makefile.header` which has your compiler options, and also the file `Name.txt` which just has the name of the code for the particular `BIN` directory. For instance, for `BIN.ecs.hermnorm`, `Name.txt` is as follows:

```
NAME=chmctdhf
```

8.4 Important variables

It's all about the `xarr` data type and the variable `yyy`. But also you need to understand how the sparse matrix routines are done – data types `Type(CONFIGPTR)` and `Type(SPARSEPTR)`.

The key variables are

`yyy`

yyy is type `xarr` that `actions.f90` and `prop.f90` some other things can use via `xxxmod`. It contains the wave function and other things, for the current time step (time step 0) and the previous one (time step 1).

`yyy%cmfpsivec(:,0)`

The wave function (orbital coefficients and A-vector) at the present time step. yyy%cmfpsivec(:,1) is the one previous.

`spfstart, astart(mcscfnum)`

For `psitype=1`, a CI wavefunction, these are the indices at which the SPF-vector and A-vector start. Thus, to pass a subroutine the current spfs, use `call mysub(yyy%cmfpsivec(spfstart,0))`.

`spftotdim`

This is the size of the SPF-vector.

`totadim`

This is the size of the A-vector.

...not finished

8.5 Project directories

8.5.1 Functions/subroutines that are necessary to define