LBNL-AMO-MCTDHF

is

.

a method for Atomic / Diatomic Rovibrational / Polyatomic Fixed-Nuclei many body quantum dynamics

.

Multiconfiguration Time-Dependent Hartree Fock

.

for ultrafast electronic and nonadiabatic nuclear dynamics

.

of atoms molecules

.

in strong laser fields

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

LBNL-AMO-MCTDHF is a collection of executable programs for the Multiconfiguration Time-Dependent Hartree-Fock method applied to ultrafast laser dynamics of atoms and molecules (in the gas phase). The code calculates nonadiabatic electronic and nuclear wave functions for the nonrelativistic Schrodinger equation. Currently it uses the dipole approximation in length and velocity gauge. It supports

- Electronic wave functions for atoms (`chmctdhf_atom`)
- Rovibrational wave functions diatoms using prolate coordinates (`chmctdhf_diatom`)
- Fixed nuclei wave functions for polyatomics (`chmctdhf_sinc`)

Future versions will support

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

The code and the equations it implements are described and used in the following list of papers.

- paper1

Contents

Chapter 1

Preliminaries

1.1 Installation

What you need to do is go to the `BIN.ecs.hermnorm` directory and

- Check that Makefile.header contains the correct compiler commands and flags. If not, link to a different Makefile.header in MCTDH.SRC, or make a new one yourself, but there are versions in MCTDH.SRC for mac, Carver, and Lawrencium/Axl.
- Run `Makeme`.

You will then have the executable files `chmctdh_diatom` and `chmctdh_atom`.

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!) you must include the line "`MPIFLAG = -D MPIFLAG`" in Makefile.header. You should have `mpif.h` for your distribution of MPI in your path.

If I were you I would put the program in your `$PATH` somehow, like by editing `.bashrc`. E.g.
`export PATH=./:/Users/haxton/programs/mctdhf/BIN.ecs.hermnorm/:$PATH`

1.2 Symbolic links

You should understand symbolic links (`ln -s` command). Like aliases, shortcuts. You should understand this because the directory structure of the code makes extensive use of them. If you try to copy the directories, you may screw the links up. On mac, use `cp -R` to copy links as is. Otherwise, they are copied as new files, and you have duplicates. Just look at the directories, see the symbolic links, then when if you copy it, check to make sure they're still there.

1.3 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.

1.4 File system hygiene

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

```
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. You may already be running in the scratch directory; good for you. Personally, I like to run in the home file system but 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.

1.5 Run Control

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

1.6 Changes from previous version (chmctdh versus chmctdhf_atom, etc. - May 2014)

The code underwent a substantial re-write. Here are a summary of the major changes.

- `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`.
- The variables `nuccharge1`, `nuccharge2`, `lbig`, and `mbig` are in namelist `&heparinp` or `&h2parinp`, not `&parinp`.
- All polarizations in both gauges for both atoms and diatoms are now possible using `pulsetheta` and `pulsephi`.
- `whichprojflux` and `corrflag` are deprecated. The options for the Hanning window implemented by Brant remain; whereas they were `corrflag`=3 or 4 before, now the variable is `brantflag`=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. It used to be 2. Values other than 0 or 1 are disabled and will result in an error. `improvednatflag`=1 and `improvedquadflag`=1 can be set at the same time reliably.

- 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, I 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.** I believe I have 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, I hope. for With `orbcompact=0`, it is not. This is exotic; don't try it without discussing it; I want to do it to see if I can get the Fluorine atom to photoionize correctly.

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.
- There is the option `littlesteps` to divide the A-vector propagation over the mean field timestep (`par_timestep`). This is needed if you want to try a big time step (`par_timestep`) yet the pulse is changing rapidly. If there is no pulse, there is no reason to use `littlesteps`.
- `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.
- Restricted configuration lists are good to go with `constraintflag`=1 or 2, density matrix or Dirac-Frenkel. BOTH require `dfrestrictflag`=1 or greater.
- **Try bigger timesteps.** The code performs more reliably because lots of little nuts and bolts have been tightened. Spf and A-vector propagation can restart without taking forever. So there is no need for `exposteps` which has been deprecated. If your orbital or a-vector iterations are restarting, you should be sure`maxexpodim` and `maxaorder` are high (but not too high – several hundred), but not worry if it is restarting after that. I have used steps of 1.0 for novelty purposes with decent and totally stable performance. There is now `littlesteps` which you should use if the pulse frequency is high compared to your mean field timestep `par_timestep`. Definitely don't start with `par_timestep` less than 0.05. Relaxation, 0.5.
- 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

- `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.

1.7 Optimizing your runs

- 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 after the start of the run. Consider increasing `littlesteps` if you are trying to take big steps (`par_timestep`) and have a pulse; you might as well.

```
Go Oribtal 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 Oribtal 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 Oribtal 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
```

- Examine `cmf_prop.time.dat` to see the overall breakdown of timing.

	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, run in parallel. If matrix elements or constructing the reduced hamiltonian or density matrix are taking a long time, try increasing `par_timestep`. 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.

- 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).

	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

- If running in parallel, check to be sure you get the same results (like, to 10 digits for everything) for different numbers of processors.
- If using `improvednatflag` to enforce natural orbitals for relaxation, be sure it is working ok; do the run without the flag as well. When it works, now, it is perfect, but it sometimes shits the bed on large calculations. The parameters `biodim` and `biotol` affect this so play with them as well. The only reason you need to use this option in general is if you are going to perform excitations or annihilations subsequently.
- **Try larger time steps** 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.** Don't sweat getting transitions energies right, unless something is qualitatively wrong. Often transition/ionization energies can be off by an eV or more.
- 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.
- **For excited state convergence (`improvedquadflag`=1) I recommend performing a run with `spf_flag`=0 and using it as a guess for one with `spf_flag`=1.** Remember to use a sufficient `aorder`. It will give a clear warning if the Krylov iterations are restarting, which in this case is bad.

Chapter 2

Nomenclature

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

I sometime use the terms configuration and slater determinant interchangeably. Sometimes when I write configuration, I mean either a slater determinant or a spin adapted sum of slater determinants. But usually, I 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 I 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`), I will generally use the term "configuration."

We borrow from the Heidelberg MCTDH package for much of the terminology and also for the style of input in the code and presentation in the manual.

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; reduced potentials; the reduced operator $\mathbf{W}$; the one electron Hamiltonian. 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 $g_{\alpha\beta} = \langle\phi_i|i\frac{\partial}{\partial t}|\phi_j\rangle$ matrix. The matrix that occurs in the A-vector equation that uses g is called τ.

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

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

Chapter 3

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. 3.6. 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
```

3.1 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 the appendix, which is a printout of `parameters.f90` with comments. The namelist variables are also compactly listed in `getparams.f90`, subroutine getparams().

** 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 or `&heparinp` for atoms.

3.2 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.

3.2.1 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.

3.2.2 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 I advise using the real-valued versions `BIN/mctdhf_XXX` to prepare the initial state, or at least checking that you obtain the same result with the complex version.

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 **??**.

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 VARIABLES I AM ABOUT TO DESCRIBE** should be incremented by the chosen value of `dfrestrictflag`. That is because, in the code, they will be decremented by that amount.

What do I mean by incremented/decremented? By "incremented by 1," I mean, change the value by 1 in the direction that enlarges the restricted configuration space. By decremented, I mean change it by 1 in the other direction.

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. Hopefully they will both work and if they do, it is a powerful convergence check. FYI, if you want, remember that wave functions can be directly compared with each other using Actions 15 and 23.

3.2.3 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`.

3.2.4 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, I suppose, 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" I 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, I 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.

3.2.5 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.

3.2.6 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`.

3.2.7 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`.

3.3 Primitive basis and Hamiltonian

The primitive basis and hamiltonian are specified in `h2parinp` or `heparinp`.

```
&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.

3.4 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. I advise using length gauge,

if stable, and even grid spacing (no small first element) which I 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
```

3.4.1 Analyzing the propagation

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

3.4.2 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.

3.5 Action input

Things that one can optionally do to 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,...`.

3.6 Complete list of namelist variables

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

```
module parameters
  use fileptrmod
  implicit none

!! recently added, otherwise notable

integer :: parorbsplit=1                        !!  Parallelize orbital calculation.  Might speed up, might
                                                !!   slow down; check timing.
real*8 :: mshift=0d0                            !! shift configurations based on m-value.. to break
                                                !!  degeneracy for state averaged sym restricted
                                                !!  (mrestrictmin, mrestrictmax) mcscf; good idea.

!! ********************************************************************************************************* !!
!!   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.
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)
```

!! 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.
```

!! INITIALIZATON

```
integer :: loadspfflag=0         !! Spf=file     !! load spfs to start calculation?
                                                 !!    (Otherwise, core eigenfunctions.)
integer :: numspffiles=1
integer :: numskiporbs=0         !!              !! Reading orbs on file(s), skips members of combined set.
integer :: orbskip(1000)=0       !!              !! Which to skip
integer :: loadavectorflag=0     !! A=file       !! load avector to start calculation?
integer :: numavectorfiles=1
character (len=200) :: &         !!              !! A-vector binary file to read.  Can have different configs
     avectorfile(100)="Bin/avector.bin"          !!   but should have same number of electrons.
character (len=200) :: &         !!              !! Spf file to read.  Can have fewer m vals, smaller radial
      spffile(100)="Bin/spfs.bin"!! Spf=filename !!   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
```

!! PROPAGATION/RELAXATION

```
real*8 :: par_timestep=0.1d0     !! Step=        !! MEAN FIELD TIMESTEP
real*8 :: finaltime=4d4          !! T=           !! length of prop.  Overridden for pulse and relax.
integer :: improvedrelaxflag=0   !! Relax        !! For improved versus regular relaxtion.
integer :: threshflag=0          !!              !! Set to 1 for regular relaxation
integer :: improvedquadflag=0    !!              !! Use newton iteration not diagonalization for improvedrelax.
                                                 !!     (1 = A-vector newton [old option 2], others disabled)
integer :: improvednatflag=0     !!              !! If improved relax, replace with natorbs every iter
                                                 !!    enforced & required for non full CI calculation
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)
integer :: littlesteps=1         !!              !! Sub intervals of mean field time step for avector prop
integer :: maxexpodim=100        !!              !! Expokit maximum kry dimension
integer :: expodim=10            !!              !! Expokit starting krylov size
real*8 :: expotol=1d-4           !!              !! Expokit krylov convergence
real*8 :: denreg=1d-10           !! Denreg=      !! density matrix regularization parameter.
real*8 :: invtol=1d-12
```

!! 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
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

```
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):: 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
```

!! Biorthogonalization

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

!! 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.
!!               requires sparseconfigflag=1, splitflag=1.
!!   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=4d-1,&         !! Low energy boundary of F.T. (relative to eground)
     EFluxHi=14d-1               !! 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
```

!! SPARSE - if sparseconfigflag .ne. 0

```
integer :: sparseopt =1                          !! 0= direct CI  1= sparse matrix algebra (faster, more memory)
integer :: nosparseforce=0       !!              !! to override exit with large number of configs, no sparse
integer :: maxaorder=1000        !!              !!   lanczos order for sparse a-vector prop and improvedquad=2
integer :: aorder=30             !!              !!
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 :: aerror=1d-9            !!              !! lanczos error criterion for sparse a-vector CMF propagation
                                                 !!  within aerror to stop.
!!real*8 :: lanthresh=1.d-7
real*8 :: lanthresh=1.d-9       !!              !! convergence criterion.
```

!! MISC AND EXPERIMENTAL

```
integer :: drivingflag=0                         !!  Solve for the change in the wave function not wave function
real*8 :: drivingproportion=0.999999999999d0     !!   -- "psi-prime" treatment.
real*8 :: quadtol=1d-1           !!              !! Threshold for solution of Newton solve iterations.  For
                                                 !!    difficult cases increase this value.
integer :: quadprecon=1          !!              !! Precondition newton iterations
real*8 :: quadpreconshift=0d0
real*8 :: quadthresh=1.d+8       !!              !!   Threshold for turning on quadratic convergence.
integer :: spf_flag=1            !!              !! IF ZERO, FREEZE SPFS. (for debugging, or TDCI)
integer :: avector_flag=1        !!              !! IF ZERO, FREEZE AVECTOR. (for debugging)
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
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
                                 !!              !! i.e. 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
!! XI
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
```

Chapter 4

Output

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

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

During the main propagation loop it shows the expectation value of the energy and the norm. By default this is the expectation value of the instantaneous Hamiltonian $H(t) = H_0 + V(t)$. With the variable `timedepexpect=0` set in `&parinp`, it will give the expectation value of H_0.

```
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 `spfs.bin` and `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
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. You may already be running in the scratch directory; good for you. Personally, I like to run in the home file system but 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.

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

4.1 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 5

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

5.1 Relaxation calculation

The most important options for relaxation are

```
improvedrelaxflag=2
improvednatflag=1
improvedquadflag=1
stopthresh=1d-8
```

We perform relaxation calculations to obtain initial eigenfunctions. 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`. All other values of improvedrelaxflag are deprecated at the moment.

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.

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` to iteratively refine that state.

There are several options available for reading in initial orbitals and wave functions. Consider this contrived example: I start with a 12 orbital calculation, done on three sigma states. I then use those orbitals, loading them, and freezing them with `spf_flag=0`, in calculations on pi and delta states. Then, I want to perform an 11 orbital, three state averaged calculation on the third sigma state, one pi state, and one delta state, in which I discard 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` 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.

5.1.1 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.

5.2 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` is overridden), except if `minpulsetime` or `maxpulsetime` is set. One may include multiple pulses. For certain appilcations I've only programmed the case in which all the pulses are aligned (all `pulsetheta`s are the same), but one can apply any pulse possible with the input.

```
&pulse
  numpulses=2
  pulsetype=3,1
  omega=0.152,0.1
  omega2=0.6,0.3
  intensity=1d-3,1d-2
  pulsetheta=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`$)$

5.3 Propagation calculation, no pulse

For flux or other analysis one needs to propagate the wave function with no pulse; I've been calling this the correlation run. One typically uses Action 15 to save the wave function for a pulse calculation, and/or Action 1 along with `corrflag=`± 1 for a correlation run. See the chapter on flux for pertinent input. The variable `finaltime` should be set in the input (default 2000au).

One might want to reuse the input file for this calculation as the input for a subsequent flux analysis calculation for photoionization. In this case one might want to go ahead and set `eground=XXX`, the ground state energy that is used to define the photon energy in the flux calculation, in this file sooner rather than later.

Chapter 6

Actions

I call various things that can be done to the wavefunction "actions," which are not necessarily mutually exclusive. Some actions skip the MCTDHF calculation and are used only for analysis. One can specify multiple actions. However, if one of the actions is an analysis routine, that will be the only action run.

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

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

```
!!   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.
!!               requires sparseconfigflag=1, splitflag=1.
!!   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
```

6.1 Flux and projected flux for photoionization

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.

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

In order to save the wavefunction you must specify action 15 during the propagation run. So that's `Act=15` on the command line or namelist input as follows. You also want to specify or check `FluxSkipMult`.

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

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

The wave function is saved in `flux.mo.bin` and `flux.avec.bin`.

6.2 Total flux (Action 16) for total photoionization

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

```
> chmctdh Inp=Input.Inp.Corr |tee Outs/Out.Corr
```

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

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

The options that you may set are below. `eground` is important!

```
&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
`FluxInterval`	Interval for flux read (F.T. timestep=`par_timestep` x `FluxSkipMult` x `FluxInterval`)
`nEFlux`	Number of energies at which to calculate the flux
`EFluxLo`	Lowest photon energy (relative to eground)
`EFluxHi`	Highest photon energy ()
`fluxoptype`	Approximation to Im(H): Default 0=no approx 1=halfnium potential approx

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 can be varied if the flux calculation is taking an extremely long time. It has taken two weeks in some cases.

```
&parinp
biodim
biotol
biocomplex
```

The flux calculation produces `xsec.spi.dat` (spi for single photoionization). This contains the columns: photon energy; one over the fourier transform; cross section (complex valued, numerically); cross sections at previous times.

6.3 Projected flux (Action 17)

In order to calculate projected flux one must calculate the final $(N-1)$-electron states for the projection. One does this with a relaxation run. The relaxation run can be used in the usual mode or in the mcscf

mode, which is described below. 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`, or `cation.mcscf.bin`. In order to project on multiple final states one must use the mcscf option.

```
&parinp
whichprojflux    Cation vector location: 0=cation.spfs.bin, cation.avector.bin 1,2=cation.mcscf.bin 1=real 2=complex
```

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

6.4 Calculating overlaps with given wave functions

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 MCSCF calculations to obtain mcscf.bin files. Specify

```
&parinp
  numovlfiles=2
  ovlfiles="ovl.mcscf.bin.sigma","ovl.mcscf.bin.pi"
```

6.5 Fourier transform of dipole moment for absorption

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

$$S(\omega) = 2\omega Im(D(\omega)E(\omega)) \tag{6.1}$$

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

6.6 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` and `Corr`. `Act=1` calculates the autocorrelation functions, and `corrflag`=1 tells the code to propagate the wavepacket both forward and backward in time, and to calculate the fourier transform once in awhile.

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`.

Chapter 7

Viewing the output

7.1 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	

7.2 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

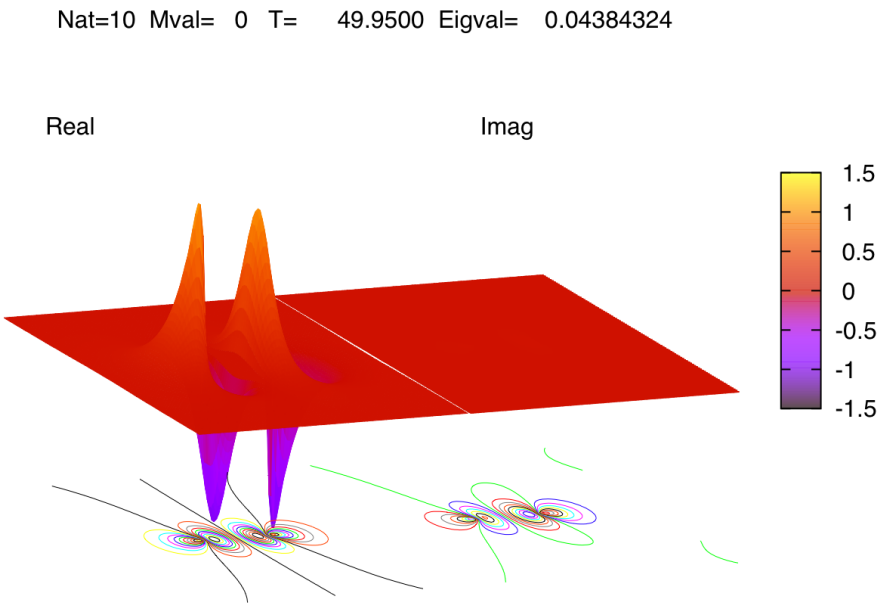

Figure 7.1: Example of gnuplot orbital plot.

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
```

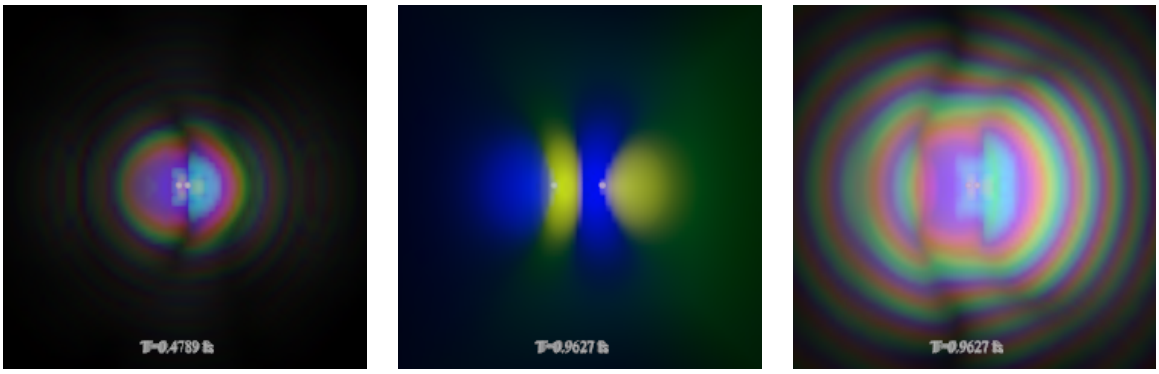

Figure 7.2: Example of Povray natural orbital plot: the tenth ($2\sigma_u$) orbital of a valence-space CAS calculation, directory O2.Small. Left: at 0.47 fs, from a range of 40 bond lengths (which are 2.2819... bohr); middle and right, 0.96fs, from a range of 7 (middle) and 40 (right) bond lengths. The 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 7.1. 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. 7.2. 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. 7.2, 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 8

Example inputs

8.1 Photoionization

Here are four inputs for a photoionization calculation on HF: For the initial state; for the pulse; for the propagation afterward; and for the cation states used for projected flux. The propagation calculation input is used for the flux calculations, e.g. `> chmctdh Inp=Input.Inp.Corr Act=16`.

8.1.1 Initial state: Input.Inp.Relax

I run this with `mctdh Inp=Input.Inp.Relax |tee Outs/Out.Relax` and then save `spfs.bin` to `Bin/spfs.bin.relax` and likewise for a-vector.

```
BLAH
```

8.1.2 Pulse: Input.Inp.Pulse

I run this with `chmctdh Inp=Input.Inp.Pulse |tee Outs/Out.Pulse` and then save `spfs.bin` to `Bin/spfs.bin.afterpulse` and likewise for a-vector.

```
BLAH
```

8.1.3 Propagation and Flux: Input.Inp.Corr

I run this with `chmctdh Inp=Input.Inp.Corr |tee Outs/Out.Corr` .

Photoionization cross sections are calculated with e.g. `chmctdh Eground=-99.80004 Inp=Input.Inp.Corr Act=16` or action 17. `eground` is **NOT** set in the input so I must specify it on the command line for flux. (I do it this way because it is easier with scripts)

```
BLAH
```

8.1.4 Cation for projected flux: Input.Inp.Cation

Remember to rename `walk.bin` and `mcscf.bin` to `cation.walk.bin` and `cation.mcscf.bin` after running this.

```
BLAH
```

8.2 Overlaps

8.3 Absorption/emission

Chapter 9

Programmers' guide

In the main directory you should see the directories

```
MCTDH.SRC
BIN.ecs.hermnorm
```

the first contains the source files; the second is a directory in which a version of the code is compiled.

9.1 Directory structure

Here are some of the subdirectories. These are all actual directories, not links.

`MCTDH.SRC`	All source files
`H2PROJECT`	One coordinate system (project) directory
`COREPROJECT`	A directory with a file shared between atom and atom projects
`DFFTPACK`	A directory with library routines (fourier transform)
`...`	
`BIN.ecs.hermnorm`	Compliation directory. Symbolic links and directory-specific `Name.txt`, `Makefile.header`, and `Definitions.INC`.
`H2PROJECT`	Actual directory with symbolic links to `../Makefile.header`, `../Definitions.INC`, and files in `MCTDH.SRC`.
`DFFTPACK`	Ditto
`...`	

The source files are all contained in `MCTDH.SRC`. Each different version of the code that is compiled, is compiled in a different directory with `BIN` in the name.

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`:

```
-rw-r--r--  1 dha...17:14 Definitions.INC
-rw-r--r--  1 dha...17:14 Name.Txt
lrwxr-xr-x  1 dha...17:14 Definitions.ALL -> ../MCTDH.SRC/Definitions.ALL
drwxr-xr-x  1 dha...17:14 DGMRES
drwxr-xr-x  1 dha...17:14 HEPROJECT
lrw-r--r--  1 dha...17:14 Makefile -> ../MCTDH.SRC/Makefile
lrw-r--r--  1 dha...17:14 Makefile.header -> ../MCTDH.SRC/Makefile.header.lawrencium.debug
lrw-r--r--  1 dha...17:14 MPI.f90 -> ../MCTDH.SRC/MPI.f90
...
```

`Name.Txt` contains the name of the program version, e.g. `chmcdthf_diatom`, etc., 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. 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(:)
  ...
```

9.2 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 I reiterate that I sometime use the terms configuration and slater determinant interchangeably. Sometimes when I write configuration, I mean either a slater determinant or a spin adapted sum of slater determinants. But usually, I 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
```

9.3 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.

Presently there are two projects, atom and diatom, in `H2PROJECT` and `HEPROJECT`. `SINCPROJECT` is pending. The two current projects are similar, and I have avoided duplication by writing one file, `MCTDH.SRC/COREPROJECT/coreproject.f90`, that is shared. This is not the general operation; project directories, like the pending `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 (one by one, or faster if you are clever) the files in `MCTDH.SRC/NEWPROJECT` to `BIN/NEWPROJECT` (with the same name).

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 I have in the existing project directories.

9.4 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)_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 is output by `Install` and 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
```

9.5 Source codes

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

9.6 Project directories

9.6.1 Functions/subroutines that are necessary to define