Tutorial for the Projector Augmented Wave Method

Version 2.0

Wun C. Chiou, Jr, University of California – Los Angeles
(September 3, 1999)

(This page intentionally left blank)

Contents

1 About the tutorial

Welcome to the PAW tutorial. It basically documents the steps which I followed while attempting to learn the usage of the features of the program. This includes problems that one might expect to encounter, possible solutions, and all of the relevant input and output. We start with studying a simple molecule, something not too cumbersome so that the calculations are relatively quick, and explore some of the things that one might want to do with this system. The methodology is to begin in a minimalist manner and then add the necessary extra features sequentially, so that we learn how they impact the calculations.

I am writing this while using the tcsh interaction shell. If you prefer to use a different unix shell, like ksh or bash, you may have to translate some of the commands into your native shell language (actually, the author of the PAW method prefers to use ksh, so you may notice that all of the program scripts are written for that shell). I also assume that you have a little bit of unix experience. If you are new to unix, you may still be alright, but it's best to become familiar with some basic unix commands before you continue.

In addition, you should know how to create and edit files with your favorite unix text editor (I use emacs for major changes and vi for quick changes).

Finally, there is always more than one way to do something. If, during the tutorial, you find that you prefer to do something differently than I (like run the executable with your own shell script or study a different system), please feel free to do so. This is just an introduction to some of the features of the PAW program and how to use them, from one individual's perspective. That said, let's begin.

2 Setup Your Environment

Questions:

- *What environmental settings are needed before we run the program?*
- *What do the directories in the main directory contain?*
- *How can I automate this environment setup?*

Things to know beforehand:

- *On which computer the PAW program is installed*
- *The directory in which to find the PAW program*
- *Basic unix commands*

1. First, log on to the computer where the program is installed (for me it's on alpha). Find out which shell you are running:

```
alpha:~> echo $shell
/usr/bin/tcsh
```

   If the return path doesn't end with csh or tcsh, like the above example, then start one of those shells.

2. Next, set the PAWDIR environment variable to point to the right place. The 'right place' should be the parent directory containing the following directories:

DX/	*IBM Data Explorer files*
F90PP/	*F90 Preprocessor used while compiling*
Manual/	*User's Manual in postscript format*
Objects/	*Objects used while compiling*
Parameters/	*Some parameter files*
Sample/	*Some sample input files*
Source/	*Source code used while compiling*
Tools/	*Analysis tools*
X/	*PAW program executables*

   For me, the 'right place' is /u/data39/iekxeaa/PAW, but substitute for the correct directory on your system in the following command:

```
alpha:~> setenv PAWDIR /u/data39/iekxeaa/PAW
```

3. After that, make sure these directories are in your path:

```
alpha:~> set path = ( $path $PAWDIR $PAWDIR/X )
```

4. If you want to use the IBM DataExplorer later on, you should also set:

```
alpha:~> setenv DXMACROS $PAWDIR/DX
alpha:~> setenv DXINCLUDE $PAWDIR/DX
```

5. Making life easier: You can have this done automatically when you start the shell by placing all the above commands in your $HOME/.cshrc (or $HOME/.tcshrc) file with any text editor. Otherwise you would have to redo all of this manually each time you log in.

3 Make the Input Files

Questions:

- *What input files are required, and what should they contain?*

- *What's a block? Can we have blank blocks in the input files?*
- *What's the difference between "1." and "1"?*
- *Are the input files case sensitive?*
- *Are comments allowed?*
- *What kind of whitespace can I include in the input files?*
- *Are file paths absolute or relative?*
- *Can we dereference environment variables?*

Things to know beforehand:

- *How to use a unix text editor to create and edit files.*

Now, we need to create our input files, which will tell the program what to do for us. There are two mandatory files required to do a basic calculation: a CNTL file and a STRC file. Their filenames should end with a .cntl and .strc suffix respectively. The prefix of the filenames should be the same, probably the name of the system to be studied. It's also a good idea to organize each system to be studied into it's own directory. For this tutorial, we will first look at a molecular system, formaldehyde (H_2CO).

1. Define a name for the system, like "h2co".
2. Create a directory where you will place the input files and change to that directory. This is also where you will run the system and get the output, so make sure you have enough free disk space in your quota or use a scratch area. I am putting it in my home directory:

```
alpha:~> mkdir tutorial/h2co
alpha:~> cd tutorial/h2co
alpha:~/tutorial/h2co>
```

3.1 Control File

The Control (CNTL) file will provide information about what kinds of settings we are going to use for the run, like how many timesteps to run and how much friction to put on the wavefunctions.

1. Use any unix text editor, like emacs, to create a control input file for your system named .cntl (See Figure 1)

```
alpha:~/tutorial/h2co> emacs h2co.cntl
```

The program reads in this file at startup. Let's quickly look at what some of the parameters mean. Setting `START` to true states that this is a new, and not continuing calculation. `NSTEP` is the number of timesteps

```
!CONTROL
  !GENERIC START=T NSTEP=100 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !PSIDYN
    !AUTO FRIC(+)=0.3 FACT(+)=1.
          FRIC(-)=0.3 FACT(-)=0.97
  !END
!END
!EOB
```

Figure 1: *h2co.cntl (1)*

for which to run the calculation. `EPWSI` is the planewave cutoff energy for the wavefunctions, in Rydberg. `CDUAL` is the factor by which `EPWSI` is multiplied to set the planewave cutoff energy for the charge density. `PSIDYN` is the dynamics of the wavefunction. `FRIC` sets the friction for when the the energy increases (+) or decreases (-), which is then multiplied by `FACT` each timestep.

The format the input parser recognizes is a block. Blocks of data or parameters are usually entered in the form `!KEYWORD PARAMETER1 = value PARAMERTER2 = value ... !END`. I guess they could be blank, for if you wanted to make some generic template control file, but many keywords have mandatory parameters. Also, there is another facility which we will see later for making blocks invisible to the program.

Notice that some of the parameters are not capitalized. The input file parser is only case sensitive to: (a) the `!EOB` (b) unix paths and filenames. Also, if you want to specify floating point precision for integers, just put a period after the number. There is no built-in facility for comments, but you can probably use some non-conflicting keyword block, if you like.

2. Read manual section 4.1 for more info on input syntax.
3. Copy Figure 1 into your CNTL file. I just made this file from an example in the Manual, section 6.2.1.
4. Save your CNTL file to your working directory.

3.2 Structure File

The structure (STRC) file is where you specify the starting geometry and composition of the system to be studied.

1. Now, utilize your preferred unix text editor (again, I'll use emacs) to create a structure input file for your system named h2co.strc (see Figure 2)

```
alpha:~/tutorial/h2co> emacs h2co.strc
```

```
!STRUCTURE
  !GENERIC LUNIT=1.8897259926 !END
  !OCCUPATIONS NBAND=8 NSPIN=1 !END

  !LATTICE T= 0.00000 5.00000 5.00000
5.00000 0.00000 5.00000
    5.00000 5.00000 0.00000 !END

  !SPECIES NAME='H_' ZV=1.
     FILE='/u/data39/iekxeaa/PAW/Setups/BP86/001H/h_.75_6.0.out' !END
  !SPECIES NAME='C_' ZV=4.
     FILE='/u/data39/iekxeaa/PAW/Setups/BP86/006C/c_.75_6.0.out' !END
  !SPECIES NAME='O_' ZV=6.
     FILE='/u/data39/iekxeaa/PAW/Setups/BP86/008O/o_.75_6.0.out' !END

  !ATOM  NAME='H_1' R= -0.50000  0.00000 -0.86600 !END
  !ATOM  NAME='H_2' R= -0.50000  0.00000  0.86600 !END
  !ATOM  NAME='C_3' R=  0.00000  0.00000  0.00000 !END
  !ATOM  NAME='O_4' R=  1.50000  0.00000  0.00000 !END

  !ISOLATE NF=3 RC=0.5 RCFAC=1.5 GMAX2=3.0 DECOUPLE=T !END
!END
!EOB
```

Figure 2: *h2co.strc (1)*

You may notice that LUNIT is the conversion factor between atomic units and angstroms. By default, most of the PAW code uses atomic units, but we can use different units for the structure specification by setting this parameter. Let's take a look at some of the other settings, as well. NBAND is the number if bands and NSPIN is the number of spin directions. Here, we are doing a spin-restricted calculation. We have also specified the LATICE translation vectors defining an fcc supercell of 10 angstrom constant, but ISOLATE makes this an isolated molecule calculation.

SPECIES is a listing of the types of elements in the system. See Manual section 6.3.7 for specifics, but notice that he have set the masses to 2 for H and 5 for everything else. This is obviously not correct, but it makes our sample calculations simpler. If you do not specify a mass, the program automatically looks up the correct one for you. You also need to let the program know where to find the file containing partial waves and etc that you want to use. File paths should be absolute, and you cannot dereference environment variables like $PAWDIR here.

ATOM is where we actually set the structure of the molecule. The name should start with that of the corresponding previously stated species. All positions are given in absolute cartesion coordinates.

2. Copy Figure 2 to your STRC file. Again, I mostly copied this from the manual, section 6.3.1.
3. Save your STRC file to the directory you created.
4. Exit the editor.

4 Run a Basic Calculation

Questions:

- *How can one start a run?*
- *How can one monitor the progress of the run?*
- *How can one stop a run?*
- *What are some common problems that one might run into?*
- *Did you notice that all these questions have the word run in them?*

Things to know beforehand:

- *How to use the X-windows environment*
- *How to use the XMGR plotting program (make sure it's installed, first)*

Now that we have our input files set up, we are ready to start the run. If all of your input files are right, and your environment is set up correctly, we should have no problems, right? Heh, heh, heh . . .

4.1 First Attempt

Just try to run the calculation as is:

- Make sure you are in the directory with your input files
- Run it:

```
alpha:~/tutorial/h2co> paw_run -c h2co.cntl
```

- Observe the run output to the stdout (screen)

4.2 Ironing out some Errors

Well, if you blindly followed my instructions correctly so far, you will probably run into some errors. So did I. It may be best to correct them one at a time, just so you can see examples of error output from the program should you run into these in the future. So, here are some errors that you may come across while trying to get it running for the first time, and suggested corrections to make:

- Wrong block structure:

```
BLOCK STRUCTURE INCORRECT
STOP IN BUFFER__READ
BLOCK STRUCTURE INCORRECT
FILE IDENTIFIER NOT RECOGNIZED
VARIABLE ID HAS THE VALUE PROT
STOP IN FILEHANDLER__UNIT
 EXIT: 2
```

  This is because I used the wrong structure for a block in the h2co.cntl file (forgot a `!END`. The input file parser recognized the error (which is in `!PSIDYN !AUTO`). Fix it by editing the file.

- Special characters in the input files:

```
TABS AND OTHER SPECIAL LETTERS ARE NOT ALLOWED
IN BUFFER STRUCTURED INPUT FILES
VARIABLE LINE HAS THE VALUE !GENERIC    START=T NSTEP=100       !END
VARIABLE POSITION HAS THE VALUE !GENERIC        START=T NSTEP=100       !END
VARIABLE ASCII VALUE OF DISALLOWED CHARACTER HAS THE VALUE         9
STOP IN BUFFER_READ
TABS AND OTHER SPECIAL LETTERS ARE NOT ALLOWED
IN BUFFER STRUCTURED INPUT FILES
VARIABLE LINE HAS THE VALUE !GENERIC    START=T NSTEP=100       !END
VARIABLE POSITION HAS THE VALUE !GENERIC        START=T NSTEP=100       !END
VARIABLE ASCII VALUE OF DISALLOWED CHARACTER HAS THE VALUE         9
FILE IDENTIFIER NOT RECOGNIZED
VARIABLE ID HAS THE VALUE PROT
STOP IN FILEHANDLER__UNIT
 EXIT: 2
```

  In both of my input files, I made it easier to read by using TABs to format the input, but this trips up the parser, too. The error message tells us the ascii value of the character that shouldn't be there, 9, which is a TAB. So, just replace all TABs with regular spaces, and save the file again.

- Orthogonality:

```
LOOP FOR ORTHOGONALIZATION IS NOT CONVERGED
STOP IN WAVES_ORTHO_X
 EXIT: 2
```

  This one is not so obvious. In general, it means that orthogonality cannot be maintained, probably because the waves are moving too fast. In this case, I forgot to limit the number of projectors and angular momentum in the center expansion. Otherwise, it tries to run with higher defaults to get higher accuracy. So, I need to set these limits.

  We have to edit the h2co.strc file to specify these limits under `!SPECIES`. After `NAME=...` we place `npro=1 lrhox=1` for H, `npro= 1 1 0 lrhox=1` for C, and `npro= 1 1 lrhox=1` for O. See the manual p.52-53.

- Species masses:
  Since we are not worried about dynamics of the cores yet, we can make the masses of the specified atoms something that will give us better results for the optimization algorithm, rather than the true values. Set `M=2` (a.u.) for H and `M=5` for all other atoms in the .strc file. As a note, if you do not specify a mass, PAW will insert the correct one for you from its internal periodic table.

- Density Functional files:
  I also did not set the right files containing the partial waves, projector functions, etc, for each atom. If x is the proper species, what I really want to use is in

  `$PAWDIR/Setups/PBE/00xx/x_.75_6.0.out`

- Density Functional parameterization:
  Another thing to check is that the density functional parameterization that you tell PAW to use should match the method in the atomic files that you tell PAW to load. Otherwise, we get meaningless results. I neglected to include this parameter in my .strc file, so it went to the default (Perdew-Zunger parameterization of the Ceperley-Alder). Edit the file to make this Perdew-Burke-Ernzerhoff exchange and correlation:
  In the `!CONTROL` block, set `!DFT type=10 !END`. See manual p.18.

- Refer to the updated CNTL and STRC files in Figure 3 and Figure 4 respectively when you update your own input files.

4.3 Monitor the Output

1. Now try again:

   - Make sure you are in the directory with your input files

```
!CONTROL
  !GENERIC START=T NSTEP=100 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !DFT TYPE=10 !END
  !PSIDYN
    !AUTO FRIC(+)=0.3 FACT(+)=1.
          FRIC(-)=0.3 FACT(-)=0.97 !END
  !END
!END
!EOB
```

Figure 3: *h2co.cntl (2)*

```
!STRUCTURE
  !GENERIC LUNIT=1.8897259926 !END
  !OCCUPATIONS NBAND=8 NSPIN=1 !END

  !LATTICE T= 0.00000 5.00000 5.00000
      5.00000 0.00000 5.00000
      5.00000 5.00000 0.00000 !END

  !SPECIES NAME='H_' ZV=1. M=2.  npro=1  lrhox=1
     FILE='/u/data39/iekxeaa/PAW/Setups/PBE/001H/h_.75_6.0.out' !END
  !SPECIES NAME='C_' ZV=4. M=5. npro= 1 1 0 lrhox=1
     FILE='/u/data39/iekxeaa/PAW/Setups/PBE/006C/c_.75_6.0.out' !END
  !SPECIES NAME='O_' ZV=6. M=5. npro= 1 1 lrhox=1
     FILE='/u/data39/iekxeaa/PAW/Setups/PBE/008O/o_.75_6.0.out' !END

  !ATOM  NAME='H_1' R= -0.50000  0.00000 -0.86600 !END
  !ATOM  NAME='H_2' R= -0.50000  0.00000  0.86600 !END
  !ATOM  NAME='C_3' R=  0.00000  0.00000  0.00000 !END
  !ATOM  NAME='O_4' R=  1.50000  0.00000  0.00000 !END

  !ISOLATE NF=3 RC=0.5 RCFAC=1.5 GMAX2=3.0 DECOUPLE=T !END
!END
!EOB
```

Figure 4: *h2co.strc (2)*

```
ONE-PARTICLE ENERGIES
=====================
EIGENVALUES[EV] FOR K-POINT  1 AND SPIN 1
 -44.762   3.132  11.241  17.958  21.581  24.656  43.961  48.345


    NFI  T[PSEC] T[K] EKIN(PSI)     E(RHO)      ECONS   ANNEE   ANNER
!>    2  0.00048     0  4.00051    0.12856    4.12907 0.29100 0.00000
!>    3  0.00073     0  3.75332   -3.49319    0.26013 0.28227 0.00000
!>    4  0.00097     0  3.38443   -7.00627   -3.62184 0.27380 0.00000
!>    5  0.00121     0  3.47601  -10.87233   -7.39632 0.26559 0.00000
!>    6  0.00145     0  3.25136  -14.18138  -10.93002 0.25762 0.00000
!>    7  0.00169     0  2.69259  -16.68125  -13.98866 0.24989 0.00000
```

Figure 5: *Excerpt from h2co.prot (1)*

- Run it:

```
alpha:~/tutorial/h2co> paw_run -c h2co.cntl
```

- Observe the run output to the stdout (screen)
- It should be okay. You can run it in the background by hitting control-Z to suspend the program and then typing "bg". If you want to put it in the foreground again, just type "fg" (if you have no other jobs running). Always make sure that a previous run is completely done before you start a new one for the same system in the same working directory. Otherwise, you may have two programs writing to the same output files.

2. Check the output:

- The protocol (PROT) file has the info you want to check. This file should be created when the program runs. In true lab notebook style, its contents are not overwritten when you restart a run, but just appended to the previously created file.
- Monitor it as the calculation runs with the "tail" program (which can be killed with control-C):

```
alpha:~/tutorial/h2co> tail -f h2co.prot
```

- The PROT file starts with some information about your setup, then starts to print out the one-particle energies in the manner shown in Figure 5:
- The headings we are interested in mean:

`NFI`	*Number of iterations*
`T[K]`	*Temperature in Kelvins*
`EKIN(PSI)`	*Wavefunction Kinetic Energy*
`E(RHO)`	*Energy of the System*
`ANNEE`	*The friction applied*

3. Watch the energy of the system:
   - This energy should be going down.
   - The friction should go down as the energy does.
   - If the energy goes up, it should switch to the higher friction
   - If the friction continually switches back and forth between low and high friction, we are in trouble. We may think we converge when we are actually not. I seem to have had this problem, so let's try to fix it. . .
4. Fixing an oscillation: Edit the file to make sure it has a constant and low friction: Friction values (f) are from 0 <= f <= 1.
   - set `!PSIDYN FRIC=0.01` in the .cntl file
   - change the previous friction keyword from `!AUTO` to `!AUTO_OFF`. This makes that particular block invisible, but we can still keep it for future reference or if we want to turn the variable friction back on later. Note that the `_OFF` option can be used for any block!
   - You can prematurely stop the program in a nice way by creating a .exit file. PAW will then exit after completion of the next iteration.

4.4 Try Again

1. Finally . . . be sure to change to `START=F` in your h2co.cntl file to tell the program to read the previous settings, which are stored in the h2co.rstrt, rather than starting from scratch again. Refer to Figure 7. Run it again and we have a converged system for formaldehyde with a total energy of -22.537 H.
2. Look at energy convergence with `paw_show`.

   ```
   alpha:~/tutorial/h2co> paw_show -e h2co
   ```

   This program uses the X-windows plotting program xmgr. It will plot the energy as a function of the time (there are also other stats it can plot; evoke it with the -h option to find out more). See Figure 6. You may be able to see the jumps in energy as we switch friction. Ideally, we want an energy without friction. So, if your energy is not yet converged to your satisfaction, you can always run it again (maybe with more timesteps and/or less friction) until you can see it with `paw_show`.

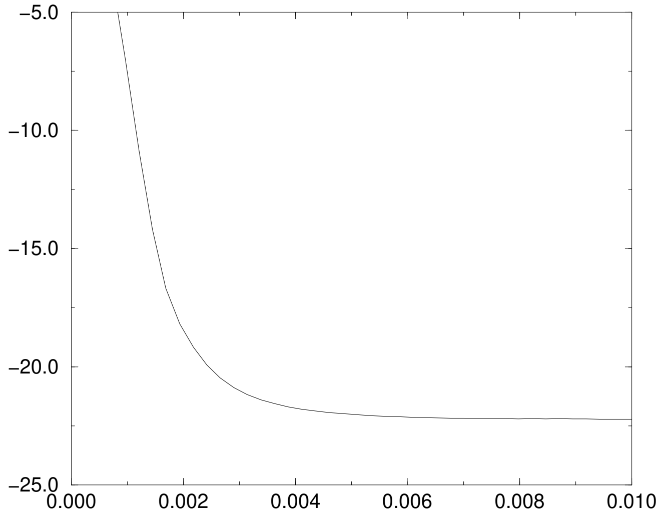

Figure 6: *Sample output from paw_show (after scaling) for formaldehyde, with static energy (H) vs time (psec).*

```
!CONTROL
  !GENERIC START=F NSTEP=50 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !DFT type=10 !end
  !PSIDYN fric=0.01
    !AUTO_OFF FRIC(+)=0.2 FACT(+)=1.
          FRIC(-)=0.1 FACT(-)=0.98 !END
  !END
!END
!EOB
```

Figure 7: *h2co.cntl (3)*

5 Optimize the Geometry

Questions:

- *How do I do a geometry optimization?*
- *Is there a tool to help me visualize the resulting structure?*

Things you should know beforehand:

- *What visualization programs you have on your system that can view a .cssr file*
- *How to use that program (I'll use Cerius^2)*

Now that we have an energy, we want to be sure that we also have the right geometry. The following giudes you through adding a geometry optimization to the calculation. There are not really any switches that you set to tell the PAW program this. Instead, you tell the it that we can now have atomic center dynamics by adding an `!RDYN` block to the CNTL file. This has the same kinds of options as the `!PSIDYN` block that we have already used.

5.1 First Attempt

First optimization: Make some changes to the CNTL file:

- Add an `!RDYN` section, which is similar to the psidyn section, but which will describe the motion of the atomic centers.
- Turn on the friction in both sections again, and set them to 0.01, except the high psi friction, which should be set to 0.1. When you remove the OFF option for AUTO, these values now override the constant friction we set previously. We set these lower frictions to allow the centers to relax better.
- Set the friction scaling factors to 1., except for the lower R value, which should be 0.95.
- Set the number of steps: `NSTEP=500`
- Make sure you have the correct masses in your STRC file, too, now that we are moving atomic centers.
- Refer to Figure 8 to see the changes made.
- Run the executable again.
- Observe the protocol output.
- Plot the energies with `paw_show` as before and observe how the energy changes when, for example, the friction changes.

```
!CONTROL
  !GENERIC START=F NSTEP=500 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !DFT type=10 !end
  !PSIDYN fric=0.01
    !AUTO FRIC(+)=0.1 FACT(+)=1.
          FRIC(-)=0.01 FACT(-)=1. !END
  !END
  !RDYN fric=0.01
    !AUTO FRIC(+)=0.01 FACT(+)=1.
          FRIC(-)=0.01 FACT(-)=0.95 !END
  !END
!END
!EOB
```

Figure 8: *h2co.cntl (4)*

5.2 Getting Closer

Closer optimization:

- Once it's close, we want the structure to optimize without any friction so that we get a more accurate structure.
- Edit the .cntl to lower the friction. The lower R friction should be 0.0 so it can relax as much as possible.
- Let's also increase the number of steps: `NSTEP=1000`
- Your h2co.cntl file should now look like Figure 9
- Re-run the calculation and let it converge. Again, you can monitor the energies by looking at the h2co.prot file and by using `paw_show`.

5.3 Visualize the Structure

It's always nice to be able to visualize a structure and save a picture so that we can later reminisce about how good we were with using PAW. The program comes with a tool which will help us do just that.

- Use `paw_strc h2co`, which will create an h2co.cssr file for the molecule in cartesian coordinates (angstrom) that you can use to visualize the structure in *Cerius*^{2}. Also, back up this file because we are going to write out a different version of it:

```
!CONTROL
  !GENERIC START=F NSTEP=1000 !END
  !FOURIER EPWPSI=30 CDUAL=2 !END
  !DFT type=10 !end
  !PSIDYN fric=0.01
    !AUTO FRIC(+)=0.01 FACT(+)=1.
          FRIC(-)=0.005 FACT(-)=1. !END
  !END
  !RDYN fric=0.01
    !AUTO FRIC(+)=0.01 FACT(+)=1.
          FRIC(-)=0.0 FACT(-)=0.95 !END
  !END
!END
!EOB
```

Figure 9: *h2co.cntl (5)*

```
alpha:~/tutorial/h2co> paw_strc h2co
MOLECULAR OUTPUT PRODUCED
======= TASK FINISHED ========
alpha:~/tutorial/h2co> ls *.cssr
h2co.cssr
alpha:~/tutorial/h2co> mv h2co.cssr h2co-m.cssr
```

- Use `paw_strc -c h2co`, which will create an h2co.cssr file for the latice that you can use to visualize the structure in *Cerius*^{2}:

```
alpha:~/tutorial/h2co> paw_strc h2co
CRYSTAL OUTPUT PRODUCED
======= TASK FINISHED ========
alpha:~/tutorial/h2co> ls *.cssr
h2co.cssr     h2co-m.cssr
```

- So the h2co.cssr file has the crystal structure, and the h2co-m.cssr file has the molecular structure. One can then use the former to visualize the spacing and determine if your lattice constant is sufficient for the system you want to examine. If it's not, you will see the resulting problems with the ctructure visually. Give it a try, and print an image showing bond lengths.

- Check out the density of states . . .

6 Analyze the Output

6.1 Density of States

There is a lot of useful information about local bonding that can be extracted by using `paw_dos` that is worth taking a look at now. We can plot the total density of states, the contribution to the COOP of a combination of particular atoms or orbitals, or both.

1. Read section 12 in the Manual to get an idea of the theoretical basis for the PAW density of states tool.
2. Create a .dcntl file, which will tell the `paw_dos` program what to do: (h2co.dcntl_1)
   - The structure of this file is similar to that of the .cntl file.
   - We set the location of input and output files. One can also output to the stdout for piping to xmgr and the like. Let's analyze the total dos, and output to a file h2co.total_dos_out
   - We tell it what we want to analyze, with `!WEIGHT` for diagonal elements of the PDOS operator or `!COOP` for the off-diagonal ones. Each of these can be referenced with a `NAME` so we can direct the output of each individually.
3. Create the plotable data: `paw_dos h2co.dcntl`
4. Plot the data with xmgr (using $X_1, Y_1, Y_2, \cdots$ format) to see the total density of states plot. The default has no gaussian witdth expansion, so we probably see just a bunch of lines at the state occupations.
5. Add in grid information into your h2co.dcntl file to get a nicer plot.

6.2 Wavefunction

The PAW suite includes a tool for converting our wavefunction information into data for the IBM Dataexplorer. We can then plot wavefunctions or the density of a set of wavefunctions.

1. Get the right wave output from PAW (`h2co.cntl_5`)
   - We need to set the `!CONTROL !ANALYSE !WAVE` section in our .cntl file.
   - Let's show the HOMO, the sixth band for H_2CO:

```
!WAVE NAME='HOMOofH2CO' FILE='h2co_homo.wv'
      TYPE='WAVE'
      !STATE B=6 K=1 S=1 !END
!END
```

- and also the density of H_2CO:

```
!WAVE NAME='DensityofH2CO' FILE='h2co_rho.wv'
      TYPE='DENSITY' WEIGHTING='TOTAL' FORMAT='ALL'
!END
```

- Set the number of iterations to 1, and give it a higher planewave cutoff so we get a higher quality picture (`CDUAL=4`)
- Run the calculation again and we get the two new output files specified above.

2. Read the Manual section 10.
3. Create a wave control file (.wcntl is a good extension)
   - Set the input `h2co_homo.wv` and `h2co.strc_out` files
   - Set the output .dx file
   - Set the viewbox: `(-5 -5 -5) <(10 0 0) (0 10 0) (0 0 10)>`
   - Save the file to `h2co.wcntl`
4. Convert our data to .dx format:
   - `paw_wave h2co.wcntl`
   - We get our corresponding output .dx file.
   - Now go back and do the same for the Density of H2CO .wv file.
   - We get out .dx file for the density.
5. Use the IBM Dataexplorer:
   - `dx -image &`
   - Load all macros in PAW/DX
   - Open the `dx_wave` file
   - Open your data file in the Main Control Panel.

   Now you can look at several different representations of the data graphically. Remember to "execute" each change you make. You probably want to set the controls to "roam" so you can rotate it with a crystal-ball interface. Cool! Save an image for posterity.

7 Find a Transition State

Let's now attempt to find a transition state for formaldehyde, say moving one of the hydrogens onto the oxygen. We will define the two states, then calculate the force at various intervals along the way. The transition state will be where the energy is a maximum and the force is zero along the transition axis. Sounds good, but how is this done in PAW? We do this by setting constraints on the system in our .strc file. For example, we can restrict the hydrogen mentioned to a plane, and then move that plane slowly toward the desired position. Let's give it a try.

1. Edit the .strc file to describe our new dynamic system (`h2co.strc_2`)

    - We will use the `!CONSTRAINTS!MIDPLANE` block to describe our desired constraint:

    ```
    !CONSTRAINTS
      !TRANSLATION !END
      !MIDPLANE ATOM1='C_3' ATOM2='H_1' ATOM3='O_4'
                MOVE=T NSTEP=350 VALUE=0.5 SHOW=T !END
    !END
    ```

    - The TRANSLATION block is applied to the 'ALL' group by default, so above, we are preventing the molecular system from running away from us in translational space.
    - ATOM1 and ATOM3 define the vector normal to our constraint plane and ATOM2 is the atom that is to be restricted.
    - MOVE says that we want to change the value of the constraint to a new VALUE, which in this case is x in the location where the plane is attached to the vector, e.g. at point $R_1 + x(R_3 - R_1)$. By moving this attachment point, we move the plane from ATOM1 when x=0 to ATOM3 when x=1. Let's begin with x=0.5.
    - NSTEP is the number of steps we want to take to get to the new constraint value. Please note that if we don't give it enough steps, we will be moving too fast to get an accurate description of what is happening, as in most AIMD calculations.
    - Setting SHOW to .true (boolean values can be entered as T,F or .true,.false) will print out some extra information into the constraint report, `h2co_constr.report`, file. In particular, we are looking for the value and force of the constraint so that we can watch the dynamics by monitoring this output.

2. Edit the .cntl file (`h2co.cntl_6`)

    - Turn off the wave analysis

- Change the upper RDYN friction to 0.001 and make sure all scale factors are unity.

3. Run the calculation: `paw_run -c h2co.cntl`

4. Observe the output

   - It will probably print out some info to the stdout mentioning deviation from the constraint. Don't worry about this unless the values look really extreme. It is really just debugging information.
   - Look at the .prot file to get an idea of how the total energy is changing. We should see it going up as the movement of the hydrogen occurs (because we are making it do something it doesn't necessarily want to do). The temperature should go up, too.
   - Now, look at the `h2co_constr.report` file. It will print out the following info at each timestep: $(x force)$, where x is our constraint value and force is the force on that constraint. We should see the force increase in the opposite direction as x goes toward 0.5.
   - We can visualize this in a better way if we plot both the constraint and the force vs the number of timesteps (to do this I had to use awk and emacs to format the output so xmgr was happy). This shows us that we are not yet at the right transition state because the force is not yet zero. `paw_show -e` also elucidates that we are not yet at an energy maximum.

5. Getting warmer . . . (`h2co.strc_3`)

   - Looks like we need to go a little bit further. Edit the .strc file, change the value of the constraint from 0.5 to 0.7, and change the number of steps to 150 (because we are not going as far). Save the file.
   - Aside . . . *At this point I decided to make life a little easier on myself by writing a little script to run the calculation, rather than having to remember what to type in each time. Basically create a h2co.doit file and include the command you want in that file. Make the file executable (*`chmod u+x h2co.doit`*) and now all we have to do is type "h2co.doit" to run the calculation. Neat, huh?* . . . end of Aside.
   - Re-run the calculation.
   - Plot the energy, and see that it has passed a maximum during our transition.
   - Plot the constraint and the force vs. number of timesteps, shown in Figure 10, and see that the force crosses zero when the constraint is about 0.6. Looks like we have a winner!
   - Now we want to go back to 0.6 slowly and then let it relax so we can see what it looks like. Change the constraint to 0.6 and re-run it.

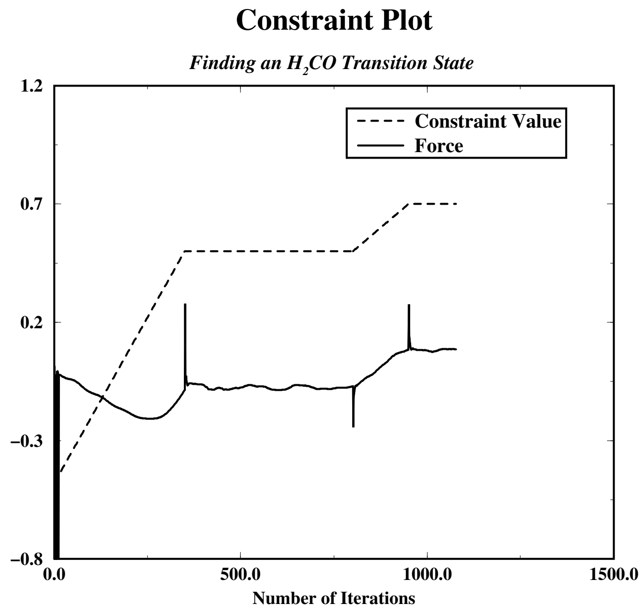

Figure 10: *Notice the spikes that result from our sudden starting and stopping, such that we briefly lose the electrons. Nifty!*

6. Letting it Relax (ahhhh)

   - Notice that we gave the hydrogen 350 timesteps to reach its destination, and we ran a total of 500 timesteps. This is to give it a little time to relax, but `paw_show -e` elucidates that we should give it a bit more time, perhaps with an appropriate change to the friction.