Software Driver Technical Manual

Joshua Horswill

August 14, 2020

Contents

1 Author and Permissions

This code was written by **Joshua Horswill** during an internship with the ILL theory group in June-November of 2020 under supervision of **Marie-Bernadette Lepetit** and **Elisa Rebolini**.

It is permissible to use and diffuse this code provided J. Horswill and MB Lepetit are kept informed. Modifications to this code can be made provided that the developer sends modifications to J. Horswill, M.B. Lepetit or E. Rebolini, so they may be included in the source code.

MB Lepetit
Institut Néel, CNRS UPR 2940
25 rue des Martyrs, BP 166, Bât. K
38042 Grenoble cedex 9
FRANCE

Email : Marie-Bernadette.Lepetit@Neel.CNRS.fr
E. Rebolini
Institut Laue-Langevin
71 Avenue des Martyrs,
38000 Grenoble,
France

Email : rebolini@ill.fr

J. Horswill
Institut Laue-Langevin
71 Avenue des Martyrs,
38000 Grenoble,
France

Email : horswill@ill.fr

2 Env2Seward

2.1 Introduction

- This code processes and formats the output from a program called ENV so it can be used in a calculation performed by a program called SEWARD.
- These simulation scripts are found in the local tools manual for the 'Fast calculation of the electrostatic potential in ionic crystals by direct summation method' written by MB Lepetit and A. Gellé.

  This local tools manual contains scripts that provide a "set of charges that allow the calculation of the Madelung potential of an ionic crystal within a predefined accuracy" [1].
- This is designed for the purpose of performing an ab-initio quantum calculation on a 'quantum fragment' that is surrounded by a first shell of ionic pseudo-potentials, and a second layer of renormalised point charges [2].

2.1.1 How to compile and run the script

- A **python 3** distribution such as 'Anaconda' or 'Active Python' can be downloaded, which provides the option of using integrated development environments (IDE) such as 'Spyder' for programming in the python language
- If you are using an IDE, open the file (typically an 'open file' option in the top left of the window, or you can drag it into the IDE text editor), hit run, and answer the prompts in the terminal
- If you want to install python directly, see `https://realpython.com/installing-python/`
- Typing '`python3 env2seward.py`' in the terminal running in the same directory as the python file should execute the script, generating the prompts in the command line. If you want to call the script from full path, just replace the `env2seward.py` with it's full path directory. The new file will appear in the same directory as the python script.

2.2 Prompt entry

2.2.1 Inputs and outputs

There are two input files:

- Prefix.env.sew0 is a help file that contains the input data for SEWARD of the MOLCAS chain. It contains the data for the quantum fragment, the TIPS and the renormalised charges.
- The second is called prefix.psd, and contains the information on the atoms represented by TIPs.

The script requires two groups of basis sets - one for the quantum fragment and one for the ionic pseudopotential. Once these have been prompted and entered through the terminal command line, an output file ready to be fed into SEWARD will be generated. This can be called prefix.sew.in.

In summary the files and data required for this program are:

- `prefix.env.sew0`
- `prefix.env.psd`

- Fragment basis sets and library locations.
- Pseudopotential basis sets and library locations.
- Name of the prefix.

Remember that for the prompt option, the input files must be in the same directory as the env2seward script. If you want the files to be called from a different relative or full path directory, it is recommended to employ the input method below.

2.2.2 Prefix, Title and Basis Set Formatting

The prefix entered can be anything you want, but it must match the prefix on the input files. For example, if the prefix was 'example', this automatically generates a seward input file called '`example.sew.in`'. However, this will not work unless the env input files have the filenames '`example.env.sew0`' and '`example.env.psd`'. The input on the prompt for the title will be written to the document introduction, and this does not depend on the name of the input files or the prefix.

Basis sets for the fragment potential are unique to the system being evaluated and so cannot be managed by env2seward. These must be entered individually when prompted by the command line, as well as their library location. If the corresponding library is default for the system, leave the input blank and press enter. If this is not the case, submit the name of the specified library.
It is almost the same for the pseudopotential, but there are usually more prompts. For example, for GdMn2O5 there are three elements but eight basis sets, since there are two types of gadolinium and manganese and four types of oxygen. This results in more prompts for the corresponding basis sets and libraries. The command line will distinguish between the inputs between the fragment and the TIPs. Continue until all basis sets have been entered and the file will be written.

2.3 Reading Filenames, Basis Sets and Libraries from an Input File

There is another option for entering the data required to format the ENV output files, and this is to write the filenames, title and basis set information in a text file. To do this, write the text file in the following format (each bullet point is a new line, although the order does not matter):

- `filename = chosen_filename`
- `title = chosen_title`
- `sew0_file = prefix.env.sew0`
- `psd_file = prefix.env.psd`
- `lib_frag = {"atom_type":{"loc":"specified basis set library","key":"basis set for atom_type"},"atom_type2":...}`
- `lib_pseudo = {"atom_type":{"loc":"specified basis set library","key":"basis set for atom_type"},"atom_type2":...}`

The format of the basis set libraries is in the python dictionary syntax. This means each dictionary key (e.g `"atom_type"`,`"loc"` and `"key"`) corresponds to a piece of data. In this case the data is either a string or another dictionary. Each atom is assigned to it's own dictionary containing a key for the specified basis set library (set to `"loc":""` if default) and a key for the basis set (`"key":"basis set"`). Here is an example for a GdMn2O5 input file, where the fragment dictionary (`lib_frag`) has no specified library but the TIP dictionary (`lib_pseudo`) basis sets are found in PSEUDO:

```
filename = example.sew.in
title = example
sew0_file = GdMn2O5_J1.env.sew0
psd_file = GdMn2O5_J1.env.psd
lib_frag = {'Mn':{'loc':'','key':'Mn.ano-rcc.Roos.21s15p10d6f4g2h.6s4p3d1f0g.'},
'O':{'loc':'','key':'O.ano-rcc.Roos.14s9p4d3f2g.4s3p1d0f'}}
lib_pseudo = {'Gd1':{'loc':'PSEUDO','key':'Gd.ECP.Marie.0s.0s.0e-Gd1-GdMn2O5.'},
'Gd2':{'loc':'PSEUDO','key':'Gd.ECP.Marie.0s.0s.0e-Gd2-GdMn2O5.'},
'Mn1':{'loc':'PSEUDO','key':'Mn.ECP.Marie.0s.0s.0e-Mn1-GdMn2O5.'},
'Mn2':{'loc':'PSEUDO','key':'Mn.ECP.Marie.0s.0s.0e-Mn2-GdMn2O5.'},
'O1':{'loc':'PSEUDO','key':'O.ECP.Marie.0s.0s.0e-O1-GdMn2O5.'},
'O2':{'loc':'PSEUDO','key':'O.ECP.Marie.0s.0s.0e-O2-GdMn2O5.'},
'O3':{'loc':'PSEUDO','key':'O.ECP.Marie.0s.0s.0e-O3-GdMn2O5.'},
'O4':{'loc':'PSEUDO','key':'O.ECP.Marie.0s.0s.0e-O4-GdMn2O5.'}}
```

Make sure that all variables are on the **same line** as the string/dictionary they are being assigned to. There also must be spacing between the variable, the = sign and the data, so that the parsing algorithm can identify the data and write it to the output file.

If you are giving the relative path (filename), make sure this user-generated input file is in the same directory as the python script, and if not, quote the full path in the command line when calling it instead of the filename. This is also the case for the ENV files. If the `sew0` and `psd` files are non-local, replace the file name with the full path directory.

- `python3 env2seward.py chosen_input_filename`

If the setup has been performed correctly, the script will output 'File has been created' and the sew.in file will appear in the input file directory.

2.4 Jupyter Notebook Version

To install Jupyter Notebook, follow this guide: `https://jupyter.org/install`. For the creation of the `env2seward_notebook.ipynb` file, the Anaconda distribution was used to install Jupyter Notebook with the `conda` command. This is the recommended method (see `https://www.anaconda.com/products/individual` for distribution download).
Once you have installed Jupyter notebook you need to download and localise the following files into the same directory/folder:

- `env2seward.py`
- `env2seward_notebook.ipynb`
- `prefix.env.sew0`
- `prefix.env.psd`

Next we need to open the Jupyter Notebook web app. We can do this by:

- Opening up a terminal in the directory containing the four files mentioned above.
- Entering '`jupyter notebook`' to initialise the web app.

Figure 1: Web app dashboard menu

- Your browser should open it in a new tab and you should see these four files in your notebook directory.
- Now open the `env2seward_notebook.ipynb` file from the web app dashboard menu.

You should see the menu shown in figure (1) and the notebook shown in figure (2). In the notebook you will see a page of Jupyter cells, some in markdown (a text formatting language) and some to run python3 code. If you want to learn more about how these work, see `https://www.dataquest.io/blog/jupyter-notebook-tutorial` In order to run these cells individually, click on them and hit `Ctrl+Enter` or press the run button (below the cell tab in the toolbar). Please follow the instructions in between the code cells to enter the correct inputs. If you would like to see the atom-types that require basis set entry, follow the instructions under the subtitle 'Basis atoms (see instructions below on how to run this cell):'. The next steps are:

- Make sure the inputs have all been entered according to the instructions (filename, title, sew0name, psdname, lib_frag, lib_pseudo)
- Select the button that says 'restart the kernel and re-run the whole notebook' that looks like a fast-forward icon.
- If you cannot find this, click the kernel icon (in English, it may be different in other languages), next to cell and widgets, and select 'Restart and Run All'.
- You will see a prompt in the first code cell (`import env2seward as e2s`). Please enter 'Y'.
- After this the last code cell (`e2s.finalwrite(filename,...)` will output 'File has been created' which means you now have a seward input file ready in the webapp directory. You can download this to a specific directory by going back to the dashboard menu, ticking the box next to '`chosen filename`' and clicking 'Download'.
- The output file will also naturally appear in the same directory as your `env2seward_notebook.ipynb` file.

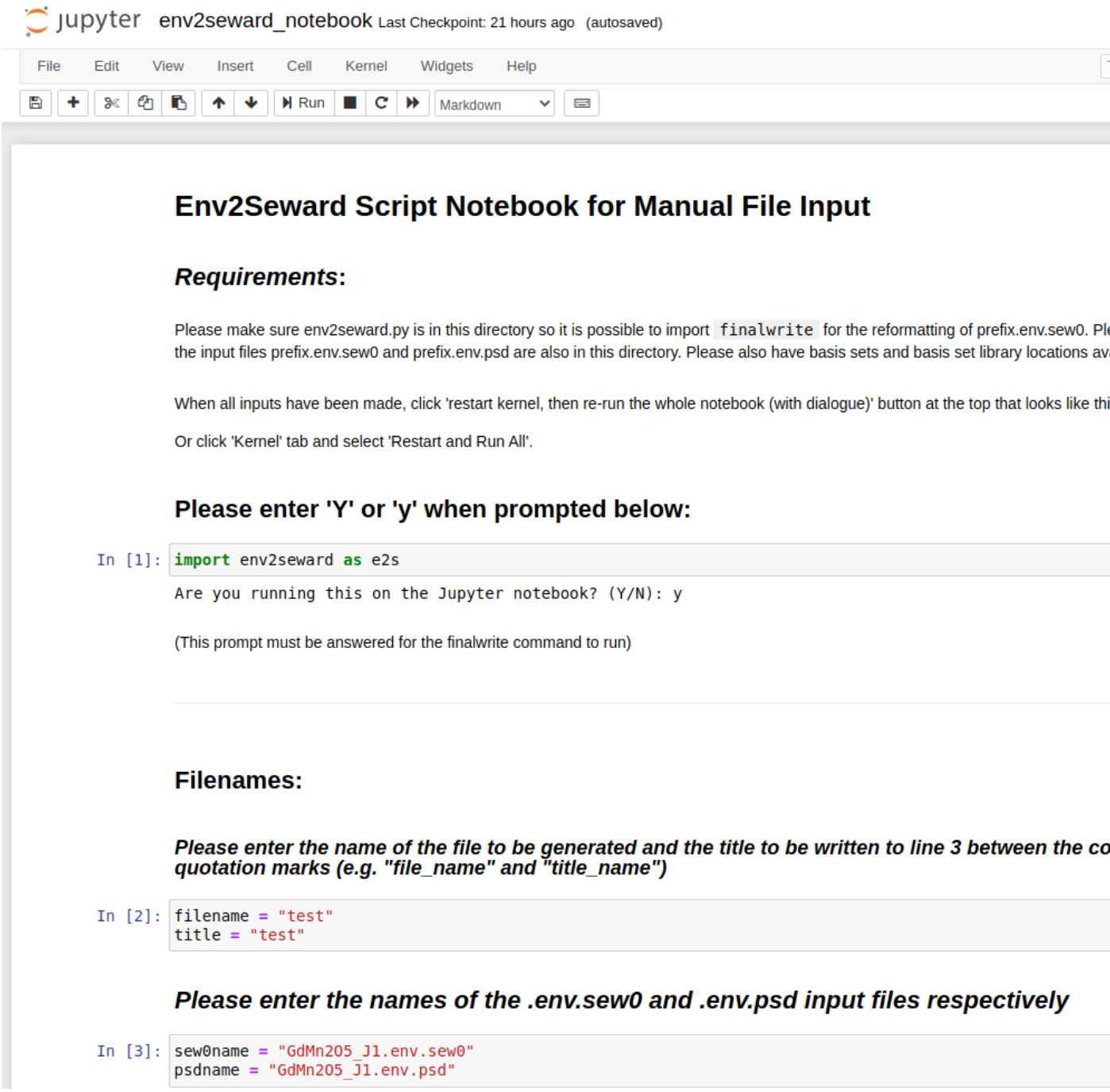

Figure 2: env2seward_notebook when opened

3 Disp_solve

3.1 What is Disp_solve?

Disp_solve is a script that generates a number of atomic displacement cell files for a crystal compound. These displaced unit cell atoms result from enacting a range of uniform electric fields on a system. It takes input files from a CRYSTAL [1] simulation to generate the necessary matrices for this displacement calculation.

3.2 How to Use Disp_solve

3.2.1 Installing python 3 and necessary modules

A **python 3** compiler must be installed (see section 2.1.1 for information on how to install python 3 or an associated IDE [2]). If an error message occurs that mentions a missing module, execute the command

'`pip install <module_name>`'

in the terminal. The following python modules are required to compile the script:

- numpy
- sys
- itertools
- copy
- os.path
- tabulate
- ast
- pathlib
- json

Most if not all of these modules should be native to your python 3 package. You can check if you have these by executing the command:

'`pip list | grep <module_name_you_want_to_check>`'

or by running the script and interpreting the error for missing modules.

3.2.2 Required Files

The following files are required by the python script:

- (Compulsory) the CRYSTAL ouput file of a phonon calculation of your system, including IR intensities (Born charges) and Hessian matrix printed in fractional coordinates (lattice vector basis).
- (Compulsory) the system.cell input file of the env code corresponding to the undistorted system.
- (Optional) the disp_solve command line input file. If absent prompts will ask you for the input data.

[1] Density functional quantum ab-initio program.
[2] Integrated development environment

3.2.3 Executing the Script and Input File Format

Once you have made sure all of these files are present, you have two options:

- 'python3 disp_solve.py' (with no input file: the code will prompt you for the inputs).
- 'python3 disp_solve.py <input_file_path>' Passing input file as a command line argument.

The prompt option allows you to enter the information directly into the terminal. You will receive the following prompts:

- Unit cell generator parameter (direct,charge,auto).

  Enter direct if you want the unit cell irreducible atom information, coordinates and charge data to all come from initial cell file.

  Enter charge or auto if you want the coordinates and irreducible atom groups to come from crystal output file.

  auto automatically generates charge data from initial cell file, whereas charge allows for manual input.
- Name/full path of crystal output file:

  Input the name of the crystal output if it is local to the script directory, or the full path of the file if it is not.
- Name/full path of initial cell file containing the unit cell and charge data:

  Same again, name if local, full path if non-local.
- start,stop,step separated by commas for E_a (E_b, E_c are done sequentially after) in units of kV/m.

  Simply input starting electric field value, final value and size of step as a tuple.
- If charge is chosen, float charge values will then be prompted

The alternative is writing an input text file that contains the above information in a certain format. Below is an example of an input file format:

```
crystal_file = ht.frequence.B1PW_PtBs.loto.out
cell_init = ymno3.cell
ea = [0,0.5,0.1]
eb = [0,0.2,0.05]
ec = [0,1,0.2]
unit_source = charge
charge_dict = {"Y":3.0, "MN":3.0, "O1":-2.0, "O2":-2.0}
```

We have the following rules:

- Each line element is separated by a space i.e. there is a space either side of the '=' sign.
- crystal_file is the variable assigned to the relative/full path directory of the crystal output file.
- cell_init is assigned to the relative/full path directory of the initial cell file.
- The ea, eb, ec lists must be in the format [start,stop,step] where start is the initial value, stop is the final value and step is the increment between values given in kV/m also.
- unit_source is the variable assigned to the unit cell generator parameter (charge,auto,direct).

- If `charge` is passed, a dictionary in python format must assign each atom to a charge value. The general format is {`"key":value`}. See section 2.3 for more examples.

Once this input file has been written, one can execute the script with this input file passed as an argument on the command line as above.

3.2.4 Output files

- Output cell file is named after the initial cell file with an added (Ea,Eb,Ec) component label. If the files are called locally, output cell file grid will be generated and stored in a new folder inside the script directory that is named after the array parameters for a,b,c.
- If the inputs files are called from full path, the output cell files will be generated and stored in a new folder inside either the directory of the cell file or the output file (specified by the command line).

References

[1] Alain Gellé and Marie-Bernadette Lepetit. Fast calculation of the electrostatic potential in ionic crystals by direct summation method. *The Journal of chemical physics*, 128(24):244716, 2008.

[2] J Varignon, S Petit, A Gellé, and MB Lepetit. An ab initio study of magneto-electric coupling of ymno3. *Journal of Physics: Condensed Matter*, 25(49):496004, 2013.