**PyFIREBALL Beta.**

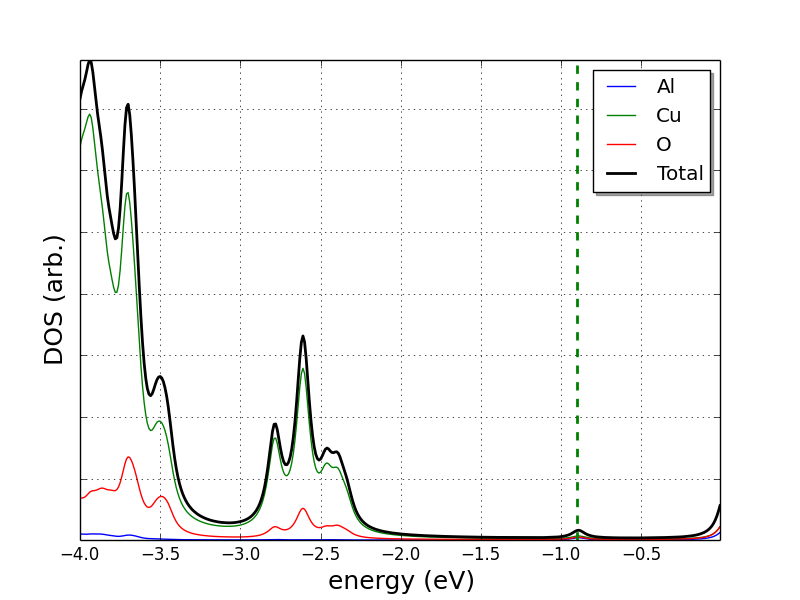
The PyFIREBALL toolbox is a collection of tools for analysis and generation of FIREBALL jobs. Principally, the toolbox exists to assist in the analysis of High Throughput studies, and came about as a natural re-engineering of the HT toolbox and Tornado.

PyFIREBALL is as “pythonic” as possible and therefore has the following features:

* an atom class, which contains all information about a specific atom in any cell.
  + therefore every atom object is unique and can interact with other atom objects:
    - e.g. atom1 – atom2 = distance between the atoms.
    - any string function (such as print) on the atom tells me about the atom’s specific features, including it’s position, species, etc.
    - atom.Neighbors(‘species’, N) returns a list of Nth neighbours of a specific species… therefore an atom object can tell you about how close it is to an impurity.
    - …. everything else FIREBALL keeps about an atom is in the atom object, for every single atom.
* a supercell class, which acts as a container for atoms. It also contains the supercell specific data, and a bunch of methods such as supercell.plotDOS(), which will carry out all the calculations and generate a rather pretty DOS plot immediately.
* Ability to directly edit the fireball.in (and related files), so I can observe 1000+ supercells, and then send 10 to carry out DOS or Optical calcuations.
* Draw supercells, atom groups, charge densities, etc. directly from the command line.
* Directly probe neighbours-of-neighbours, etc.

The philosophy behind PyFIREBALL Beta is that you can load the module into iPython Notebook, IDLE, etc, and immediately begin working on your dataset. Supercell objects can be created with the readDirectory() method, and Fdata can be directly read in via the readInfo() method. Then you can quickly process, analyse and report on your findings in an ETL fashion. The roadmap for PyFIREBALL includes further visualization abailities and xsf file handling, allowing for the multi step processes in generation and analysis of HT calcuations to be simplified with your own scripts or within an interactive python console.

PyFIREBALL is currently in a “Beta Release” phase and is currently available at:

***Generating a PDOS plot in PyFIREBALL.***

To generate a PDOS by species in PyFireball, you first need to have a directory where you have run a DOS calculation in FIREBALL. You will also need a copy of the PyFIREBALL module and to be able to use python interactively. This tutorial assumes you have all of that.

Step 1- load in the working directory and FData.

Open your python terminal, type “import PyFIREBALL as PF”

then read the Fdata:

elements = PF.readInfo('Path\_to\_FData')

read in the supercell:

thisCell = PF.directory2Supercell(elements, 'Path\_to\_Fireball\_Directory')

Step 2- generate the DOS object.

DOSofCell = thisCell.genDOS(elements)

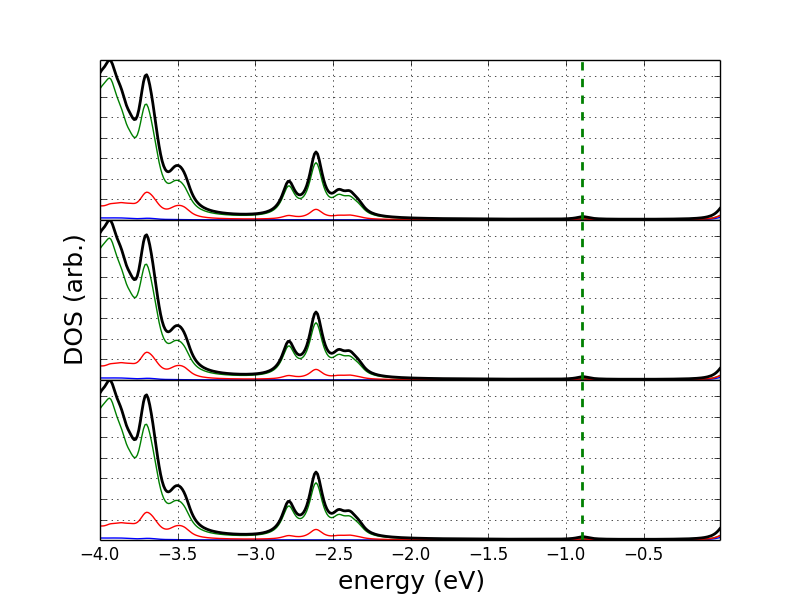
Step 3- visualize.

DOSofCell.plotme()

If you want to put in the HOMO line, change the limits of the plot, etc, there are a number of arguments you can apply to the plotme() method, for example try:

DOSofCell.plotme(HOMO = thisCell.HOMO, legend = True, plotContributions = True, offset = True, contriblw = 2, xlim = [-2,2])

refer to the documentation for a full list of plotme() options.

***Generating a stacked 3-plot DOS in PyFIREBALL:***

Although the image on the right is the DOS for three supercells that are actually the same, this is because I didn’t have data when writing the tutorial. There are many occasions when you want to show the PDOS of a couple of calculations together for comparison. This tutorial explains how to generate a stacked plot of 3 DOS calculations, using the plot3DOS() method in PyFIREBALL Beta.

Step 1- Import the Module, and read in the Fdata.

import PyFireball as PF

elements = PF.readInfo('Path\_to\_Fdata')

If you need to add another Fdata to the elements list, used the “update” method:

elements.update(PF.readInfo('Path\_to\_Other\_Fdata'))

Step 2- read in the supercells you plan to plot:

cell1 = PF.directory2Supercell(elements,'Path\_to\_FIREBALL\_Directory')

cell2 = PF.directory2Supercell(elements, ' Path\_to\_FIREBALL\_Directory ')

cell3 = PF.directory2Supercell(elements, ' Path\_to\_FIREBALL\_Directory ')

In this tutorial, we are going to create a plot of all three supercell’s PDOS’s in a stacked plot, so we want a list of each supercell’s HOMO for the purposes of either offsetting each PDOS plot or to add to the plot as it is.

multiHOMO = {'Cell1' : cell1.HOMO, 'Cell2' : cell2.HOMO, 'Cell3' : cell3.HOMO, }

names can include LaTex symbols using the $ as an escape sequence. (try naming something ‘CuGa$\_{0.98}$Fe$\_{0.02}$O$\_2$’)

We generate a list now, because you can generate DOS objects of multiple supercells in one plot and have a different list of HOMOs for each plot in the stack.

If the plots are each with many HOMOs, like a combined plot in each DOS, then used a multiHOMO list: multiHOMOList = [multiHOMO] for this tutorial, we don’t need that.

Now we package our HOMOs and cells together into a list of DOS objects which will contain the PDOS by species.

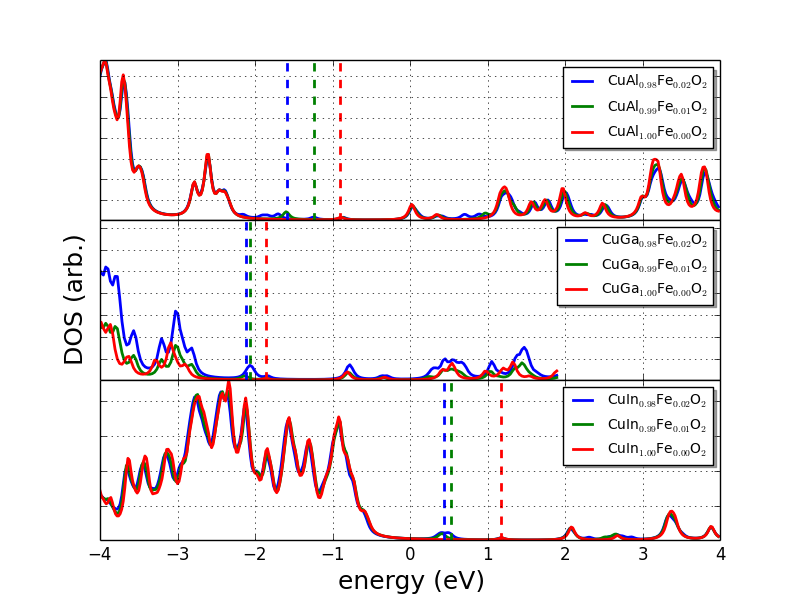
Step 3- Make a list

listOfDens = [cell1.genDOS(elements), cell2.genDOS(elements), cell3.genDOS(elements)]

Step 4- plot.

PF.plot3DOS(listOfDens, plotTOT = True, subLegend = True, contriblw =2, xlim = [-4, 4], name = 'StackedDOS')

Image appears on screen and file is saved.

***Create a stacked plot of total DOS for a number of supercells.***

For a recent publication, I needed to plot the DOS of nine different supercells for comparison to one another. There are three sets of three different materials. Here I describe how to generate the plot in the image, which shows the total DOS for three different supercells in each of three stacked plots. We will combine the data from each cell into a DOS object for each plot, then package those and pass that to 3DOSplot() in PyFIREBALL beta. Resulting in the final image as shown.

In terms of steps, the procedure to generate this plot is a bit of a pain, so I actually created a python script file just to take care of it, which is at the end of this entry.

Step 1- Load the PyFIREBALL beta module and the Fdata:

import PyFireball as PF

print "Reading in Fdata"

elements = PF.readInfo('./MultiCupratePaper/Al\_Fdata/')

elements.update(PF.readInfo('./MultiCupratePaper/Ga\_Fdata/'))

elements.update(PF.readInfo('./MultiCupratePaper/In\_Fdata/'))

We use the .update method for the elements dictionary to add multiple Fdata’s together.

Step 2- read in the supercells, there are nine in this case:

CuAl098Fe002 = PF.directory2Supercell(elements, './MultiCupratePaper/CuAl0.98Fe0.02/90/')

CuAl099Fe001 = PF.directory2Supercell(elements, './MultiCupratePaper/CuAl0.99Fe0.01/DOSrun/')

CuAl100Fe000 = PF.directory2Supercell(elements, './MultiCupratePaper/CuAl1.00Fe0.00/CuAl1.00Fe0.00\_NewDOS\_for\_Even\_energies/001/')

CuGa098Fe002 = PF.directory2Supercell(elements, './MultiCupratePaper/CuGa0.98Fe0.02/57/')

CuGa099Fe001 = PF.directory2Supercell(elements, './MultiCupratePaper/CuGa0.99Fe0.01/')

CuGa100Fe000 = PF.directory2Supercell(elements, './MultiCupratePaper/CuGa1.00Fe0.00/')

CuIn098Fe002 = PF.directory2Supercell(elements, './MultiCupratePaper/CuIn0.98Fe0.02/3/')

CuIn099Fe001 = PF.directory2Supercell(elements, './MultiCupratePaper/CuIn0.99Fe0.01/001/')

CuIn100Fe000 = PF.directory2Supercell(elements, './MultiCupratePaper/CuIn1.00Fe0.00/001/')

Step 3- pack together the HOMO information.

In this step, we have nine different HOMOs, one for each cell. The plot method accepts the HOMO for each cell as a dictionary called multiHOMO addressed by the same name as the cell in the dictionary of supercells which we will create in a minute. We create three dictionaries of HOMOs, one for each of the plots in the stacked plot:

InMultiHOMO = {'CuIn$\_{0.99}$Fe$\_{0.01}$O$\_2$' : CuIn099Fe001.HOMO, 'CuIn$\_{1.00}$Fe$\_{0.00}$O$\_2$' : CuIn100Fe000.HOMO, 'CuIn$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuIn098Fe002.HOMO}

GaMultiHOMO = {'CuGa$\_{0.99}$Fe$\_{0.01}$O$\_2$' : CuGa099Fe001.HOMO, 'CuGa$\_{1.00}$Fe$\_{0.00}$O$\_2$' : CuGa100Fe000.HOMO, 'CuGa$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuGa098Fe002.HOMO}

AlMultiHOMO = {'CuAl$\_{0.99}$Fe$\_{0.01}$O$\_2$' : CuAl099Fe001.HOMO, 'CuAl$\_{1.00}$Fe$\_{0.00}$O$\_2$' : CuAl100Fe000.HOMO, 'CuAl$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuAl098Fe002.HOMO}

multiHOMOList = [AlMultiHOMO, GaMultiHOMO, InMultiHOMO]

Notice that we can use the string symbol ($) as an escape sequence to LaTex, therefore we have subscripts in the legend in the final plots.

Step 4 – Generate DOS objects and package them up for the plot method:

In this step, we generate a dictionary of DOS objects for each of the three plots. Each of these dictionaries contains 3 DOS objects- one for each supercell.

InDOSes = {'CuIn$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuIn098Fe002.genDOS(elements), 'CuIn$\_{0.99}$Fe$\_{0.01}$O$\_2$' :CuIn099Fe001.genDOS(elements), 'CuIn$\_{1.00}$Fe$\_{0.00}$O$\_2$' :CuIn100Fe000.genDOS(elements)}

AlDOSes = {'CuAl$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuAl098Fe002.genDOS(elements), 'CuAl$\_{0.99}$Fe$\_{0.01}$O$\_2$' :CuAl099Fe001.genDOS(elements), 'CuAl$\_{1.00}$Fe$\_{0.00}$O$\_2$' :CuAl100Fe000.genDOS(elements)}

GaDOSes = {'CuGa$\_{0.98}$Fe$\_{0.02}$O$\_2$' : CuGa098Fe002.genDOS(elements), 'CuGa$\_{0.99}$Fe$\_{0.01}$O$\_2$' :CuGa099Fe001.genDOS(elements), 'CuGa$\_{1.00}$Fe$\_{0.00}$O$\_2$' :CuGa100Fe000.genDOS(elements)}

Next, we concatenate the TOTAL column in each of the supercells and create a new DOS object, which contains the TOTAL DOS for each of it’s constituents as it’s contributions:

InDOS\_combined= PF.catDOSTots(InDOSes)

GaDOS\_combined = PF.catDOSTots(GaDOSes)

AlDOS\_combined = PF.catDOSTots(AlDOSes)

And finally, we just put these combined DOS objects together for easy passing to the plot3DOS method:

listOfDens = [AlDOS\_combined, GaDOS\_combined, InDOS\_combined]

Step 5- Pass this to plot3DOS:

PF.plot3DOS(listOfDens, multiHOMO = multiHOMOList, subLegend = True, contriblw =2, xlim = [-4, 4], name = 'TotalDOS\_Fig5')

And that’s it, you will get an onscreen plot and it will save a .png to your local directory, called “TotalDOS\_Fig5” in this case.

One final note: if you wanted to create a PDOS of just the DOS from one species in the cell, you can change the arguments to genDOS() in Step 4 (try: .genDOS(‘Cu’) to only return the Cu-based DOS).

**Viewing specific atoms in PyFireball.**

One huge advantage of PyFireball is that we can select specific atoms within a list and probe their properties directly. If you have MayAvi installed, you can also render them with custom labels or other information. For example, I may want to visualize a specific trimer within a supercell and then see it’s charge density. I may want to actually check out a trimer that is close to a doping site, and one that is a distance from any doping site in a supercell, etc. etc.

So, from any Python IDE, import PyFireball:

>>> import PyFireball as PF

And I have a “quickstart” routine to open a specific directory and load in the data from it:

First import the Fdata:

>>> elements = readInfo('./MultiCupratePaper/Al\_Fdata/')

Then we load the FIREBALL directory into memory:

>>> CuAl = directory2Supercell(elements, './MultiCupratePaper/CuAl0.98Fe0.02/90/')

Using the neighbours routines and list comprehension, you can easily select a number of sites within your supercell that have or have not got specific properties.

For example, I created a list of 4 random atoms out of a supercell, and told drawAtoms to render a sphere with respect to charge on the atom, draw bonds between those atoms, and to represent the atoms in colours based on the JMol colouring scheme. The command is:

>>> PF.drawAtoms(CuAl, someAtoms, numNN = 0, implicitBonds = True, chargeCloud = True, atomsAlso = True, chargeLabel = True, secretZero = True)

and the result is:

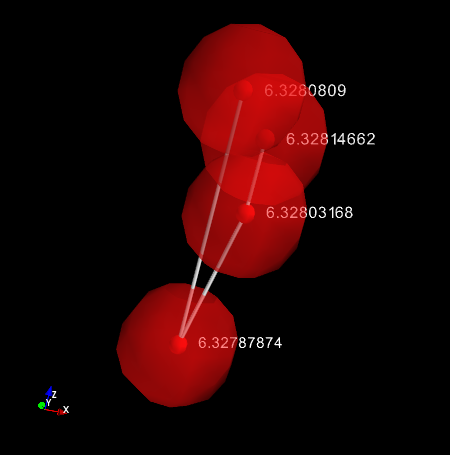


Figure PF.drawAtoms(CuAl, someAtoms, numNN = 0, implicitBonds = True, chargeCloud = True, atomsAlso = True, chargeLabel = True, secretZero = True)

Where you can see each oxygen, each atom, it’s charge represented by a value and the size of the semitransparent sphere around it, and it’s position relative to the others.

More practically, I can use the “numNN” kwarg, and I can tell PyFireball to let me see an Fe atom in this system, and it’s 12 nearest neighbours:

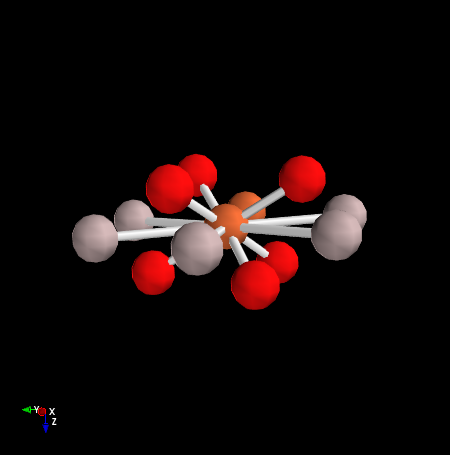


Figure PF.drawAtoms(CuAl, [CuAl.atomList[0]], numNN = 12, implicitBonds = True)

For this, we use the drawAtoms() method in PyFireball, which has a lot of options. The implicitBonds argument above, tells the renderer to show me the bonds. Simply adding the “speciesLabel = True” labels the atoms by species:

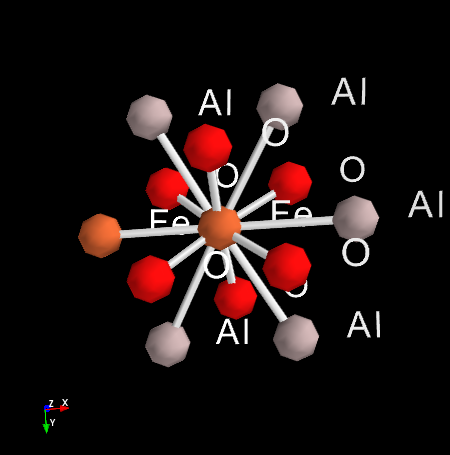


Figure PF.drawAtoms(CuAl, [CuAl.atomList[0]], numNN = 12, implicitBonds = True, speciesLabel = True)

All elements in the render can be edited with the built in toolbar, along the top of the plot. When you have the image as you’d like, just save to file.