Langmuir Python Documentation

Release 2.0

Adam Gagorik

February 28, 2014

CONTENTS

CHAPTER

ONE

GETTING STARTED

- numpy
- scipy
- pandas
- matplotlib

CHAPTER

TWO

MODULE LIST

2.1 common

@author: adam

common.**zhandle**(*handle*, *mode='rb'*)
Open a file using gzip if necessary.

Parameters **handle** (*str*) – filename

common.**splitext**(*handle*, **args*, ***kwargs*)
Extract stub and extension from a file object or string.

Parameters **handle** (*str*) – filename

common.**load_pkl**(*handle*, *max_objs=256*)
Load max objs from a pkl file

common.**load_pkls**(*pkls*)
Load a set of pkls into a list.

common.**save_pkl**(*obj*, *handle*)
Save obj to a pkl file.

common.**compare_dicts**(*dict1*, *dict2*)
Compare two dictionaries.

common.**compare_lists**(*list1*, *list2*)
Compare two lists.

common.**command_script**(*paths*, *name=None*, *stub='run'*, *command=None*)
Create a handy bash script that loops over the paths.

Parameters

- **paths** (*list*) – list of paths
- **name** (*str*) – name of script
- **stub** (*str*) – jobname stub
- **command** (*str*) – command to insert at each directory

2.2 regex

@author: adam

regex.**number**(*string*, *index=0*, *pytype=<type 'float'>*)
Get numbers in a string at index and convert them.

Parameters

- **string** (*str*) – string
- **index** (*int*) – which regex group to match
- **pytype** (*type*) – python type to convert to

```
>>> print lm.regex.number('asdfasdfa1.231e10')
12310000000.0
```

regex.**numbers**(*string*, *pytype=<type 'float'>*)
Find all numbers in a string and convert them.

Parameters

- **string** (*str*) – string
- **index** (*int*) – which regex group to match
- **pytype** (*type*) – python type to convert to

```
>>> print lm.regex.numbers('asdfasdfa1.231e10asdf1209asd asd0912 sdaf9 81')
[12310000000.0, 1209.0, 912.0, 9.0, 81.0]
```

regex.**strip_comments**(*string*)
Return string will all comments stripped.

Parameters **string** (*str*) – string

```
>>> print lm.regex.strip_comments('hello # goodbye')
hello
```

regex.**fix_boolean**(*string*)
Return string with true/false in correct python format.

Parameters **string** (*str*) – string

```
>>> print lm.regex.fix_boolean('true false True False true false')
True false True False True false
```

2.3 find

@author: adam

find.**find**(*work*, *r=False*, *single=True*, *absolute=True*, *stub='*'*, *ext=None*, *exclude_dirs=False*, *exclude_files=False*, *sort_by=<function numbers at 0x4527d70>*, *at_least_one=False*, *follow_links=False*)
A method for searching for files and directories using patterns.

Parameters

- **work** (*str*) – path to search
- **r** (*bool*) – perform the search recursivly
- **single** (*bool*) – do not return a list
- **absolute** (*bool*) – return absolute paths
- **stub** (*str*) – the wildcard-able search pattern (star is the wildcard)

- **ext** (*str*) – the filename extension
- **exclude_dirs** (*bool*) – do not include directories in the search
- **exclude_files** (*bool*) – do not include files in the search
- **sort_by** (*func*) – a function (applied to paths) used to sort the results
- **at_least_one** (*bool*) – make sure at least one thing was found
- **follow_links** (*bool*) – do not follow symbolic links

```
>>> import os
>>> work = os.getcwd()
>>> pkls = find(work, ext='txt*') # find all txt files in work
```

find.**systems**(*work*, ***kwargs*)

Search for directories that contain "runs".

Parameters

- **args** – see `runs()`
- **kwargs** – see `runs()`

```
>>> systems = lm.find.systems(r'/home/adam/simulations')
>>> print '\n'.join(systems)
'/home/adam/simulations/systemA'
'/home/adam/simulations/systemB'
'/home/adam/simulations/systemC'
'/home/adam/simulations/systemD'
```

find.**sims**(*work*, ***kwargs*)

Search for directories that contain "parts".

Parameters

- **args** – see `parts()`
- **kwargs** – see `parts()`

```
>>> work = r'/home/adam/simulations/systemA/run.0'
>>> sims = lm.find.sims(work, stub='voltage.right')
>>> print '\n'.join(systems)
'/home/adam/simulations/systemA/run.0/voltage.right_+0.0'
'/home/adam/simulations/systemA/run.0/voltage.right_+0.2'
'/home/adam/simulations/systemA/run.0/voltage.right_+0.4'
'/home/adam/simulations/systemA/run.0/voltage.right_+0.8'
```

find.**runs**(**args*, ***kwargs*)

Search for directories of the form run*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> runs = lm.find.runs(r'/home/adam/simulations/systemA/')
>>> print '\n'.join(runs)
'/home/adam/simulations/systemA/run.0/'
'/home/adam/simulations/systemA/run.1/'
'/home/adam/simulations/systemA/run.2/'
'/home/adam/simulations/systemA/run.3/'
```

find.**run**(**args*, ***kwargs*)
Search for a single directory of the form run*.

Parameters

- **args** – see find()
- **kwargs** – see find()

```
>>> run = lm.find.run(r'/home/adam/simulations/systemA')
RuntimeError: found multiple run* in directory:
    /home/adam/simulations/systemA
        run.0
        run.1
        run.2
        run.3
```

find.**parts**(**args*, ***kwargs*)
Search for directories of the form part*.

Parameters

- **args** – see find()
- **kwargs** – see find()

```
>>> prts = lm.find.parts(r'~/simulations/systemA/run.0/voltage.right_+0.0')
>>> print '\n'.join(prts)
'/home/adam/simulations/systemA/run.0/voltage.right_+0.0/part.0/'
'/home/adam/simulations/systemA/run.0/voltage.right_+0.0/part.1/'
```

find.**part**(**args*, ***kwargs*)
Search for a single directory of the form part*.

Parameters

- **args** – see find()
- **kwargs** – see find()

```
>>> part = lm.find.parts(r'~/simulations/systemA/run.0/voltage.right_+0.0')
RuntimeError: found multiple part* in directory:
    /home/adam/simulations/systemA/run.0/voltage.right_+0.0
        part.0
        part.1
```

find.**inps**(**args*, ***kwargs*)
Search for files of the form *.inp*.

Parameters

- **args** – see find()
- **kwargs** – see find()

```
>>> inps = lm.find.inps('.')
```

find.**inp**(**args*, ***kwargs*)
Search for a single file of the form *.inp*.

Parameters

- **args** – see find()
- **kwargs** – see find()

```
>>> inp = lm.find.inp('.')
```

find.**chks**(**args*, ***kwargs*)
Search for files of the form *.chk*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> chks = lm.find.chks('.')
```

find.**chk**(**args*, ***kwargs*)
Search for a single file of the form *.chk*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> chk = lm.find.chk('.')
```

find.**parms**(**args*, ***kwargs*)
Search for files of the form *.parm*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> parms = lm.find.parms('.')
```

find.**parm**(**args*, ***kwargs*)
Search for a single file of the form *.parm*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> parm = lm.find.parm('.')
```

find.**dats**(**args*, ***kwargs*)
Search for files of the form *.dat*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> dats = lm.find.dats('.')
```

find.**dat**(**args*, ***kwargs*)
Search for a single file of the form *.dat*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> dat = lm.find.dat('.')
```

find.**txts**(**args*, ***kwargs*)
Search for files of the form *.txt*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> txts = lm.find.txts('.')
```

find.**txt**(**args*, ***kwargs*)
Search for a single file of the form *.txt*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> txt = lm.find.txt('.')
```

find.**pkls**(**args*, ***kwargs*)
Search for files of the form *.pkl*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> pkls = lm.find.pkls('.')
```

find.**pkl**(**args*, ***kwargs*)
Search for a single file of the form *.pkl*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> pkl = lm.find.pkl('.')
```

find.**pngs**(**args*, ***kwargs*)
Search for files of the form *.png*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> pngs = lm.find.pngs('.')
```

find.**png**(**args*, ***kwargs*)
Search for a single file of the form *.png*.

Parameters

- **args** – see `find()`
- **kwargs** – see `find()`

```
>>> png = lm.find.png('.')
```

find.**slice_path**(*path*, *regex*)
Return dirname of path where the regex matches.

Parameters

- **path** (*str*) – path to parse
- **regex** (*str*) – regex to match

```
>>> print slice_path('r/home/adam/Desktop', 'adam')
'/home/adam'
```

find.**slice_part**(*path*)
Return dirname of path where run directory is.

Parameters **path** (*str*) – path to parse

```
>>> print slice_path('r/system/run/sim/part')
'/system/run/sim/part'
```

find.**slice_sim**(*path*)
Return dirname of path where sim directory is.

Parameters **path** (*str*) – path to parse

```
>>> print slice_path('r/system/run/sim/part')
'/system/run/sim'
```

find.**slice_run**(*path*)
Return dirname of path where run directory is.

Parameters **path** (*str*) – path to parse

```
>>> print slice_path('r/system/run/sim/part')
'/system/run'
```

2.4 checkpoint

@author: adam

class checkpoint.**CheckPoint**(*handle=None*)
A class to open lm checkpoint files.

Attribute	Description
electrons	list of int
holes	list of int
traps	list of int
defects	list of int
trapPotentials	list of float
fluxState	list of int
randomState	list of int
parameters	Parameters

Parameters **handle** (*str*) – filename or file object; Can be None.

load(*handle*)
Load checkpoint from a file.

Parameters **handle** (*str*) – filename or file object

```
>>> chk = lm.checkpoint.CheckPoint()
>>> chk.load('out.chk')
```

save(*handle*)
Save checkpoint to a file.

Parameters **handle** (*str*) – filename or file object

```
>>> chk = lm.checkpoint.CheckPoint()
>>> chk.save('sim.inp')
```

clear()
Forget all stored information.

```
>>> chk = lm.checkpoint.CheckPoint()
>>> chk.clear()
```

reset(*keep_elecs=False*, *keep_holes=False*, *traps=True*, *defects=True*)
Reset a simulation.

Parameters

- **keep_elecs** – do not delete electrons
- **keep_holes** – do not delete holes

update(**args*, ***kwargs*)
Update parameters.

has_key(*key*, **args*, ***kwargs*)
Check if key exists in parameters.

fix_traps()
Check trap parameter validity.

```
>>> chk = lm.checkpoint.CheckPoint()
>>> chk.fix_traps()
```

checkpoint.**load**(*handle*)
Load checkpoint file.

Parameters **handle** (*str*) – filename or file object

```
>>> chk = lm.checkpoint.load('out.chk')
```

checkpoint.**load_last**(*work*, ***kwargs*)
Load the last checkpoint file found in the working directory.

Parameters **work** (*str*) – directory to look in

```
>>> chk = lm.checkpoint.load_last('/home/adam/simulations')
```

checkpoint.**compare**(*chk1*, *chk2*)
Rigorously compare to checkpoint files.

Parameters

- **chk_i** (*lm.checkpoint.CheckPoint*) – checkpoint object 1

- **chk_j** (*lm.checkpoint.CheckPoint*) – checkpoint object 1

Returns dict, bool

2.5 parameters

@author: adam

class parameters.**Parameter**
Parameter(key, pytype, default, units, fmt)

default
Alias for field number 2

fmt
Alias for field number 4

key
Alias for field number 0

pytype
Alias for field number 1

units
Alias for field number 3

class parameters.**Parameters**(*handle=None*)
A class to store Langmuir simulation parameters.

Create Parameters instance.

Parameters **handle** (*str*) – filename or file object

```
>>> parm = lm.parameters.Parameters('out.parm')
```

set_defaults()
Set parameters to default values.

```
>>> parm = lm.parameters.Parameters()
>>> parm.set_defaults()
```

load(*handle*)
Load parameters from a file.

Parameters **handle** (*str*) – filename or file object

```
>>> parm = lm.parameters.Parameters()
>>> parm.load('out.parm')
```

save(*handle*)
Save parameters to a file.

Parameters **handle** (*str*) – filename or file object

```
>>> parm = lm.parameters.Parameters()
>>> parm.save('sim.inp')
```

to_ndarray()
Convert parameters to a numpy array.

to_series()
Convert parameters to a pandas series.

to_dict()
Convert to simple python dict.

parameters.**load**(*handle*)
Create Parameters object from file.

Parameters **handle** (*str*) – filename or file object

parameters.**compare**(*parm1*, *parm2*)
Compare two Parameter objects.

2.6 datfile

@author: adam

class datfile.**Column**
Column(key, pytype, default, units, fmt)

default
Alias for field number 2

fmt
Alias for field number 4

key
Alias for field number 0

pytype
Alias for field number 1

units
Alias for field number 3

datfile.**fix**(*dat*)
Fix columns in dataframe.

datfile.**load**(*handle*, ***kwargs*)
Load datfile into a Pandas dataframe.

2.7 surface

@author: adam

surface.**load_ascii**(*handle*, *square=False*, *cube=False*, *shape=None*, ***kwargs*)
Wrapper around np.loadtxt. Forces data to be at least 3 dimensions.

Parameters

- **square** – reshape as if data is NxN
- **cube** – reshape as if data is NxNxN
- **square** – bool
- **cube** – bool
- **shape** – list

Parm shape reshape data

surface.**load**(*handle*, **args*, ***kwargs*)
Load surface from file.

surface.**save**(*handle*, *obj*, **args*, ***kwargs*)
Save object to a file. Takes into account the file extension.

Parameters

- **handle** – filename
- **obj** – object to save

surface.**threshold**(*a*, *v=0*, *v0=0*, *v1=1*, *copy=False*)
Set values in array above {v} to {v1}, and below {v} to {v0}.

Parameters

- **v** (*float*) – threshold value
- **v0** (*float*) – lower value
- **v1** (*float*) – upper value
- **copy** (*bool*) – copy array

surface.**linear_mapping**(*array*, *n=0.0*, *m=1.0*)
Map values in array to fall in the range [n,m].

Parameters

- **array** (*list*) – array like object
- **n** (*float*) – lower bound
- **m** (*float*) – upper bound

surface.**rfunc**(*size=None*)
Produces numbers in the range [-0.5, 0.5].

Parameters **size** – shape of output

Type int

class surface.**WaveDimensions**(*L=6.283185307179586*, *n=1*)
Compute wavelength, wavenumber, etc from an interval length (L) and number of waves (n).

Parameters

- **L** (*float*) – interval length
- **n** (*int*) – number of waves in interval

```
>>> wx = WaveDimensions(10, 2)
>>> print wx
[Wave Dimensions]
    L      = 10
    n      = 2
    lambda = 5.00000e+00
    nubar  = 2.00000e-01
    k      = 1.25664e+00
```

calc(*L=None*, *n=None*)
Perform calculations to compute wavelength, wavenumber, etc. Called automatically in constructor.

Parameters

- **L** (*float*) – interval length

- **n** (*int*) – number of waves in interval

surface.**f_gyroid**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for gyroid.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> gyroid(x, y, z, w.k, w.k, w.k)
```

surface.**gyroid**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_scherk_first_surface**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for scherk.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> scherk_first_surface(x, y, z, w.k, w.k, w.k)
```

surface.**scherk_first_surface**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_schwarz_p_surface**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for psurface.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> schwarz_p_surface(x, y, z, w.k, w.k, w.k)
```

surface.**schwarz_p_surface**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_schwarz_d_surface**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for dsurface.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> schwarz_d_surface(x, y, z, w.k, w.k, w.k)
```

surface.**schwarz_d_surface**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_bandXY**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for bands that run along z-direction.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> bandXY(x, y, z, w.k, w.k, w.k)
```

surface.**bandXY**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_bandXZ**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for bands that run along y-direction.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)

- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> bandXZ(x, y, z, w.k, w.k, w.k)
```

surface.**bandXZ**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

surface.**f_bandYZ**(*x*, *y*, *z*, *kx*, *ky*, *kz*)
Surface function f(x,y,z) for bands that run along x-direction.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **kx** (*float*) – 2 pi nx / Lx
- **ky** (*float*) – 2 pi ny / Ly
- **kz** (*float*) – 2 pi nz / Lz

```
>>> w = WaveDimensions(10, 2)
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> bandYZ(x, y, z, w.k, w.k, w.k)
```

surface.**bandYZ**(*x*, *y*, *z*, *wx*, *wy*, *wz*)
Wrapper around f_* that uses WaveDimensions.

class surface.**Kernel**(*xmin*, *xmax*, *ymin*, *ymax*, *zmin*, *zmax*, *spacing=1.0*)
An x, y, z, v mgrid.

Parameters

- **xmin** (*float*) – lower x
- **xmax** (*float*) – upper x
- **ymin** (*float*) – lower y
- **ymax** (*float*) – upper y
- **zmin** (*float*) – lower z
- **zmax** (*float*) – upper z
- **spacing** (*float*) – grid spacing

class surface.**FuncKernel**(*func*, **args*, ***kwargs*)
An x, y, z, v mgrid. Computes v using the function = f(x, y, z) passed. See Kernel for more parameters.

class surface.**SimpleKernel**(*func*, *nx=3*, *ny=3*, *nz=3*, *spacing=1.0*)
An x, y, z, v mgrid. Computes v using the function = f(x, y, z) passed. Creates x, y, z domain using spacing and number of points.

Parameters

- **func** (*func*) – function of x, y, z

- **nx** (*int*) – x-direction has 2*nx + 1 points
- **ny** (*int*) – y-direction has 2*ny + 1 points
- **nz** (*int*) – z-direction has 2*nz + 1 points
- **spacing** (*double*) – grid spacing

class surface.**RandomKernel**(**args*, ***kwargs*)
An x, y, z, v mgrid. Computes v using random noise. Creates x, y, z domain using spacing and number of points. See SimpleKernel for more parameters.

class surface.**GaussianKernel**(*sx*, *sy*, *sz*, *mx=0.0*, *my=0.0*, *mz=0.0*, *spacing=1.0*)
An x, y, z, v mgrid. The size of the grid is determined using the stdev of the Gaussian PDF.

Parameters

- **sx** (*float*) – sigma x
- **sy** (*float*) – sigma y
- **sz** (*float*) – sigma z
- **mx** (*float*) – mean x
- **my** (*float*) – mean y
- **mz** (*float*) – mean z
- **spacing** (*float*) – grid spacing

class surface.**Isotropic**(*grid*, *kernel*, *rfunc=<function rfunc at 0x57cde60>*, *v=0.0*, *mode='same'*, *verbose=False*)
Performs convolution of random noise with a kernel to make morphology.

Parameters

- **xsize** (*int*) – x grid points
- **ysize** (*int*) – y grid points
- **zsize** (*int*) – z grid points
- **kernel** (*Kernel*) – Kernel instance
- **rfunc** (*func*) – function that produces random numbers in range [-0.5,0.5]

surface.**f_isotropic**(*x*, *y*, *z*, *sx*, *sy*, *sz*, *full=False*)
Surface function f(x,y,z) for isotropic morphology according to Jake.

Parameters

- **x** (*float*) – x-value(s)
- **y** (*float*) – y-value(s)
- **z** (*float*) – z-value(s)
- **sx** (*float*) – sigma x
- **sy** (*float*) – sigma y
- **sz** (*float*) – sigma z

```
>>> x, y, z = np.mgrid[0:10:100j,0:10:100j,0:10:100j]
>>> isotropic(x, y, z, 1, 1, 1)
```

2.8 grid

@author: adam

class `grid.`**`Grid`**(*xsize*, *ysize*, *zsize*)

A class to represent the Langmuir simulation lattice. Alternate contructors exist.

Parameters

- **xsize** (*int*) – x points
- **ysize** (*int*) – y points
- **zsize** (*int*) – z points
- **mode** (*str*) – left, center, or right

classmethod **`from_checkpoint`**(*chk*)

Create grid instance from checkpoint file.

Parameters **chk** – checkpoint filename or object

```
>>> grid = lm.grid.Grid.from_checkpoint('out.chk')
```

`create_zeros`()

Compute a grid of zeros.

`parameters`()

Return a `langmuir.parameters.Parameters`.

`refine`(*factor=0*)

Refines the mesh. If factor=0 or dx/factor > 1.0, the mesh is reset to the default spacing of 1.0.

Parameters **factor** (*float*) – zoom factor; > 0 refines the mesh, < 0 zooms out.

class `grid.`**`IndexMapper`**(*grid*)

A class that maps between site indicies the Langmuir way.

Parameters

- **xsize** (*int*) – x-dimension of grid, like grid.x
- **ysize** (*int*) – y-dimension of grid, like grid.y
- **zsize** (*int*) – z-dimension of grid, like grid.z

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> imap = lm.grid.IndexMapper(grid)
```

`indexS`(*x_index*, *y_index*, *z_index*)

Compute (Langmuir's) site index from x, y, and z index

Parameters

- **x_index** (*int*) – x-site
- **y_index** (*int*) – y-site
- **z_index** (*int*) – z-site

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> imap = lm.grid.IndexMapper(grid)
>>> s = imap.indexS(0, 0, 0)
```

indexX(*s_index*)
Compute (Langmuir's) x index from site index

Parameters **s_index** (*int*) – site index

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> imap = lm.grid.IndexMapper(grid)
>>> x = imap.indexX(10)
```

indexY(*s_index*)
Compute (Langmuir's) y index from site index

Parameters **s_index** (*int*) – site index

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> imap = lm.grid.IndexMapper(grid)
>>> y = imap.indexX(10)
```

indexZ(*s_index*)
Compute (Langmuir's) z index from site index

Parameters **s_index** (*int*) – site index

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> imap = lm.grid.IndexMapper(grid)
>>> z = imap.indexX(10)
```

class grid.**XYZV**(*grid*, *s*, *v=None*)
Put site values on a mesh using site ids.

Parameters

- **grid** (Grid) – grid object
- **site_ids** (*list*) – site indcies
- **site_values** (*list or scalar*) – site values

```
>>> chk  = lm.checkpoint.load('out.chk')
>>> grid = lm.grid.Grid.from_checkpoint(chk)
>>> xyzv = lm.grid.XYZV(grid, chk.electrons, -1)
```

class grid.**PrecalculatedMesh**(*grid*)
Perform fast computation of Coulomb interactions and distances with a precomputed mesh.

Parameters **grid** (Grid) – grid object

```
>>> grid = lm.grid.Grid(5, 5, 5)
>>> mesh = lm.grid.Mesh(grid)
```

coulomb(*xi_ids*, *yi_ids*, *zi_ids*, *xj_ids=None*, *yj_ids=None*, *zj_ids=None*, *q=1*)
Compute coulomb interaction at j's due to charges at i's. If no j's are passed, then the answer will be computed at every mesh point (expensive!)'

Parameters

- **xi_ids** (*list, int*) – charge x-position(s)
- **yi_ids** (*list, int*) – charge y-position(s)
- **zi_ids** (*list, int*) – charge z-position(s)
- **xj_jds** – energy x-position(s)
- **yj_jds** – energy y-position(s)

- **zj_jds** – energy z-position(s)
- **q** (*int, float*) – charge

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> mesh = lm.grid.PrecalculatedMesh(grid)
>>> coul = mesh.coulomb(0, 0, 0, 1, 1, 1, -1)
```

distances (*xi_ids*, *yi_ids*, *zi_ids*, *xj_ids*=[ ], *yj_ids*=[ ], *zj_ids*=[ ])
Compute all distances between i's, or compute all distances between i's and j's

Parameters

- **xi_ids** (*list, int*) – initial x-position(s)
- **yi_ids** (*list, int*) – initial y-position(s)
- **zi_ids** (*list, int*) – initial z-position(s)
- **xj_jds** – final x-position(s)
- **yj_jds** – final y-position(s)
- **zj_jds** – final z-position(s)

```
>>> grid = lm.grid.Grid(10, 10, 10)
>>> mesh = lm.grid.PrecalculatedMesh(grid)
>>> dist = mesh.distances(0, 0, 0, 1, 1, 1)
```

2.9 analyze

@author: adam

analyze.**combine** (*objs*)
Combine a set of panda's DataFrames into a single DataFrame

analyze.**calculate** (*obj*)
Compute all flux.

analyze.**equilibrate** (*obj*, *last*, *equil=None*)
Get the difference between two steps.

2.10 plot

@author: adam

plot.**convert** (*ifile*, *ofile*, *density=300*)
Convert a file.

plot.**crop** (*name*, *border=10*)
Crop a file.

plot.**save** (*name*, *border=10*, ***kwargs*)
Save a file and crop it.

plot.**subplots** (*nrows=1*, *ncols=1*, *w=6*, *h=6*, *l=1.5*, *r=0.5*, *t=0.5*, *b=1.5*, *ws=0.0*, *hs=0.0*, ***kwargs*)
Wrapper around pyplot.subplots that uses absolute numbers for widths, etc.

plot.**multiple_locator** (*axis=None*, *x=None*, *y=None*, *s=None*, *which='major'*)
Easier way to set mpl.ticker.MultipleLocator.

plot.**maxn_locator**(*axis=None*, *x=None*, *y=None*, *s=None*, *which='major'*)
Easier way to set mpl.ticker.MaxNLocator.

plot.**scilimits**(**args*, ***kwargs*)
Easier way to set scilimits.

plot.**zoom**(*axis=None*, *factor=0.05*, *l=None*, *r=None*, *t=None*, *b=None*)
Zoom out.

plot.**shift_subplots**(*fig=None*, *horizontal=0.0*, *vertical=0.0*)
Shift center of subplots.

plot.**create_colorbar_axes**(*axis=None*, *shift=0.01*, *width=0.05*)
Create colorbar axis next to subplot.

plot.**fix_colorbar_labels**(*cax*)
Align colorbar labels.

plot.**fake_alpha**(*rgb*, *alpha*)
Compute color.

plot.**rectangle**(*x0*, *y0*, *x1*, *y1*, *axis=None*, ***kwargs*)
Draw a rectangle.

plot.**contourf**(*x*, *y*, *v*, *n*, *index=(slice(None*, *None*, *None)*, *slice(None*, *None*, *None)*, *0)*, ***kwargs*)
wrapper around contourf

plot.**contour**(*x*, *y*, *v*, *n*, *index=(slice(None*, *None*, *None)*, *slice(None*, *None*, *None)*, *0)*, ***kwargs*)
wrapper around contour

plot.**errorbar**(*x*, *y*, *color='r'*, ***kwargs*)
wrapper around errorbar

plot.**transformA**(*x*, *y*, *a=None*)
transform angle from data to screen coordinates

plot.**transformX**(*x*, *transform_from=None*, *transform_to=None*)
transform x from axes to data coords

plot.**transformY**(*y*, *transform_from=None*, *transform_to=None*)
transform y from axes to data coords

plot.**scale_bars**(*patches*)
scales bars of bar plot

class plot.**BarPlot**(*groups*, *members*, *rwidth=0.5*, *gshift=1.0*)
Helper for calculating x-values of bar plots.

```
>>> bar = BarPlot(groups=3, members=2)
>>> bar.ydata[0,:] = [1,1] #group 0
>>> bar.ydata[1,:] = [2,2] #group 1
>>> bar.ydata[2,:] = [3,3] #group 2
>>> bar.plot()
```

class plot.**PeakFinder**(*figure=None*, *mode='fit'*, *handle=None*, *callback=None*, *savename='plot.pdf'*)
Interactive way to find peaks on a xy plot.

```
>>> fig, ax1 = plt.subplots(1, 1)
>>> x = np.linspace(-2*np.pi, 2*np.pi)
>>> plt.plot(x, np.sin(x), 'r-')
>>> finder = PeakFinder(figure=fig)
>>> plt.show()
```

2.11 fit

A set of classes used to fit 2D data.

class `fit.Fit` (*x*, *y*, *func*, *popt=None*, *yerr=None*)
Base class for fitting.

Parameters

- **x** (*list*) – array of x-values
- **y** (*list*) – array of y-values
- **func** (*function*) – function object
- **popt** (*list*) – initial guess for parameters

Warning: Do not explicitly create Fit instance, its a base class.

```
>>> fit = Fit(xdata, ydata, func)
>>> x = np.linspace(0, 10, 100)
>>> y = fit(x)
>>> plt.plot(x, y, 'r-')
```

plot (*xmin=None*, *xmax=None*, *xpoints=100*, ***kwargs*)
Wrapper around `matplotlib.pyplot.plot()`. Plots the fit line.

Parameters

- **xmin** (*float*) – min x-value
- **xmax** (*float*) – max x-value
- **xpoints** (*int*) – number of xpoints

```
>>> fit.plot(xmin=-1.0, xmax=1.0, xpoints=10, lw=2, color='blue')
```

text (*s*, *x*, *y=None*, *transform=None*, *rotate=False*, *draw_behind=True*, ***kwargs*)
Wrapper around `matplotlib.pyplot.text()`. Useful for putting text along the fit line drawn by `Fit.plot()`, and rotating the text using the derivative.

Parameters

- **s** (*str*) – text
- **x** (*float*) – x-value of text
- **y** (*float*) – y-value of text; if None, use the fit line
- **transform** (`matplotlib.transforms.Transform`) – matplotlib transform; if None, plt.gca().transAxes
- **rotate** (*bool*) – rotate text using angle computed from derivative
- **draw_behind** (*bool*) – hide the plot objects behind the text

```
>>> fit.text('Hello!', 0.1, rotate=True, fontsize='large')
```

solve (*y=0*, *x0=0*, *return_y=False*, ***kwargs*)
Wrapper around `scipy.optimize.fsolve()`. Solves y=fit(x) for x.

Parameters

- **y** (*float*) – y-value

- **x0** (*float*) – initial guess
- **return_y** (*bool*) – return x and fit(x)

Returns xval, yval

```
>>> xval, yval = fit.solve(y=0, x0=1.0)
```

brute (*a=0*, *b=1*, *find_max=False*, *return_y=False*, ***kwargs*)
Wrapper around `scipy.optimize.brute()`. Finds minimum in range.

Parameters

- **a** (*float*) – lower x-bound
- **b** (*float*) – upper x-bound
- **find_max** (*bool*) – find max instead of min
- **return_y** (*bool*) – return x and fit(x)

Returns xval, yval

```
>>> xval, yval = fit.brute(a=-1, b=1)
```

maxbrute (*a=0*, *b=1*, *return_y=False*, ***kwargs*)
Calls `Fit.brute()` with find_max=True

minimize (*x0=0*, *find_max=False*, *return_y=True*, ***kwargs*)
Wrapper around `scipy.optimize.minimize()`. Finds minimum using a initial guess.

Parameters

- **x0** (*float*) – initial guess for x
- **find_max** (*bool*) – find max instead of min
- **return_y** (*bool*) – return x and fit(x)

Returns xval, yval

```
>>> xval, yval = fit.minimize(x0=0.1)
```

maximize (*x0*, *return_y=True*, ***kwargs*)
Calls `Fit.minimize()` with find_max=True

derivative (*x*, ***kwargs*)
Wrapper around `scipy.misc.derivative()`, unless an inheriting class implements the derivative analytically.

Parameters **x** (*float*) – x-value to evaluate derivative at

Returns float

tangent (*x*, ***kwargs*)
Compute a tangent line at x.

Parameters

- **x** (*float*) – x-value to evaluate derivative at
- **xmin** (*float*) – min x-value
- **xmax** (*float*) – max x-value
- **xpoints** (*int*) – number of xpoints

```
>>> func = fit.tangent(x=1.5)
>>> print func(1.5)
... 0.0
```

plot_tangent(*x*, *xmin=None*, *xmax=None*, *xpoints=100*, ***kwargs*)
Plot a tangent line at x.

Parameters

- **x** (*float*) – x-value to evaluate derivative at
- **xmin** (*float*) – min x-value
- **xmax** (*float*) – max x-value
- **xpoints** (*int*) – number of xpoints

```
>>> fit.plot_tangent(x=1.5, lw=2, color='blue')
```

summary()
Create a summary `dict` of fit results.

Returns `dict`

sort(***kwargs*)
Sort x and y values.

class `fit.`**FitPower**(*x*, *y*, *order=1*, *popt=None*, *yerr=None*)
$\sum_{i=0}^{N} c_i x^i$

Parameters **order** (*int*) – order of polynomial

summary()
create a summary `dict`

class `fit.`**FitLinear**(*x*, *y*, *popt=None*, *yerr=None*)
$mx + b$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary `dict`

class `fit.`**FitQuadratic**(*x*, *y*, *popt=None*, *yerr=None*)
$ax^2 + bx + c$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary `dict`

class `fit.`**FitInterp1D**(*x*, *y*, *popt=None*, *yerr=None*, ***kwargs*)
Fit to interpolating function.

Parameters **kind** (*str*) – linear, nearest, zero, slinear, quadratic, cubic.

kind : linear, nearest, zero, slinear, quadratic, cubic

class `fit.`**FitUnivariateSpline**(*x*, *y*, *popt=None*, *yerr=None*, ***kwargs*)
Fit to spline.

Parameters **k** – degree of spline (<=5)

class fit.**FitLagrange**(*x*, *y*, *popt=None*, *yerr=None*, ***kwargs*)
Fit to lagrange interpolating spline. It sucks.

class fit.**FitBarycentric**(*x*, *y*, *popt=None*, *yerr=None*, ***kwargs*)
Fit to lagrange interpolating spline. It sucks.

class fit.**FitTanh**(*x*, *y*, *popt=None*, *yerr=None*)
$a\tanh(bx+c)+d$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary dict

class fit.**FitLog**(*x*, *y*, *popt=None*, *yerr=None*)
$a\tanh(bx+c)+d$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary dict

class fit.**FitXTanh**(*x*, *y*, *popt=None*, *yerr=None*)
$ax\tanh(bx+c)+d$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary dict

class fit.**FitErf**(*x*, *y*, *popt=None*, *yerr=None*)
$a\,\mathrm{erf}(bx+c)+d$

summary()
create a summary dict

class fit.**FitXErf**(*x*, *y*, *popt=None*, *yerr=None*)
$ax\,\mathrm{erf}(bx+c)+d$

summary()
create a summary dict

class fit.**FitGaussian**(*x*, *y*, *popt=None*, *yerr=None*)
$ae^{\frac{-(x-m)^2}{2\sigma^2}}$

summary()
create a summary dict

class fit.**FitSin**(*x*, *y*, *popt=None*, *yerr=None*)
$a\sin(bx+c)+d$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary dict

class fit.**FitCos**(*x*, *y*, *popt=None*, *yerr=None*)
$a\cos(bx+c)+d$

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary `dict`

class `fit.`**FitLinearZero**(*x*, *y*, *popt=None*, *yerr=None*)
mx

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary `dict`

class `fit.`**FitQuadraticZero**(*x*, *y*, *popt=None*, *yerr=None*)
mx

derivative(*x*, ***kwargs*)
analytic derivative of function

summary()
create a summary `dict`

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

INDEX

G

H

I

K

L

M

N

P

R

S

T

U

W

X

Z