phaseshifts Documentation

Release 0.1.4-dev

Liam Deacon

July 23, 2014

CONTENTS

Contents:

CHAPTER ONE

INTRODUCTION

This package is a Python-based implementation of the Barbieri/Van Hove phase shift (*phsh*) calculation package needed to produce phase shifts for various LEED packages (including CLEED), as well as for certain XPD packages.

To quote the original authors site:

“The phase shift calculation is performed in several steps:

1. Calculation of the radial charge density for a free atom.
2. Calculation of the radial muffin-tin potential for atoms embedded in a surface defined by the user (the surface is represented by a slab that is periodically repeated in 3 dimensions, within vacuum between the repeated slabs); various approximations to the exchange potential are available; relativistic effects are taken into account.
3. Calculation of phase shifts from the muffin-tin potential.
4. Elimination of pi-jumps in the energy dependence of the phase shifts.”

Note: You can get the original Fortran source (& learn more about the *phsh* programs) from:

http://www.icts.hkbu.edu.hk/surfstructinfo/SurfStrucInfo_files/leed/leedpack.html

The aim of this package is to both automate and simplify the generation of phase shift files in a manner that is easy for the computational hitch-hiker, but powerful for those that wish to extend the package for particular needs.

CHAPTER TWO

INSTALLING THE PHASESHIFTS PACKAGE

The phaseshifts package requires CPython 2.6 or later and also uses the numpy, scipy and periodictable packages. Currently, it has only been tested extensively with Python 2.7 on Windows, so there are no guarantees with other platforms. To perform a setup follow the steps below.

1. Install the numpy, scipy and periodictable packages.

   On systems compatible with PyPI this can be done using the command:

   ```
   pip install numpy scipy periodictable
   ```

   Or if you have the easy_install package:

   ```
   easy_install install numpy scipy periodictable
   ```

   Older versions of numpy & scipy did not allow simultaneous installation - if you experience problems then try first installing numpy before attempting to install scipy.

   The periodictable package allows lookup of the most common crystal structure for a given element and is instrumental in many of the convenience functions contained in the model module.

   Alternatively download and install these packages manually following the instructions provided for the respective packages.

2. To install the phaseshifts package:

   ```
   python setup.py install
   ```

   With any luck the package has been installed successfully. A set of test scripts are provided, however a simple check may suffice using an interactive session of the python interpreter:

   ```
   >>> import phaseshifts
   >>> from phaseshifts.lib import libphsh  # compiled FORTRAN .pyd or .so using f2py
   ```

   If these execute without errors then it is likely that all is well, but in case of problems or bugs please use the contact provided below and I will do my best to address the problem quickly.

Tip: On Windows systems it may be easier to install a scientific python distibution rather than install the dependencies from source - Python(x,y) with mingw (gcc & gfortran) installed is highly recommended.

CHAPTER

THREE

CONTACT

This package is developed/maintained in my spare time so any bug reports, patches, or other feedback are very welcome and should be sent to: liam.deacon@diamond.ac.uk

The project is in the early developmental stages and so anyone who wishes to get involved are most welcome (simply contact me using the email above).

CHAPTER

FOUR

ACKNOWLEDGEMENTS

As with all scientific progress, we stand on the shoulders of giants. If this package is of use to you in publishing papers then please acknowledge the following people who have made this package a reality:

- **A. Barbieri** and **M.A. Van Hove** - who developed most of the original fortran code. Use *A. Barbieri and M.A. Van Hove, private communication.* (see `doc/phsh2007.txt` for further details).
- **E.L. Shirley** - who developed part of the fortran code during work towards his PhD thesis (refer to the thesis: *E.L. Shirley, "Quasiparticle calculations in atoms and many-body core-valence partitioning", University of Illinois, Urbana, 1991*).
- **Christoph Gohlke** - who developed the elements.py module used extensively throughout for the modelling convenience functions (see 'elements.py' for license details).

I would also be grateful if you acknowledge this python package (*phaseshifts*) as: *L.M. Deacon, private communication.*

4.1 Thanks

I wish to personally add a heartfelt thanks to both Eric Shirley and Michel Van Hove who have kindly allowed the use of their code in the `libphsh.f` file needed for the underlying low-level functions in this package.

CHAPTER

FIVE

SCRIPTS

5.1 phsh.py

5.1.1 Command line usage

The *phsh.py* script is placed into the system `PATH` during installation of the phaseshifts package. It can then be used from the command line, e.g. `phsh.py --help` will produce a list of command line options:

```
usage: phsh.py [-h] -b <bulk_file> -i <slab_file> [-t <temp_dir>] [-l <lmax>]
             [-r <start_energy> <final_energy> <step>] [-f <format>]
             [-S <subdir>] [-v] [-V]


phsh - quickly generate phase shifts

      Created by Liam Deacon on 2013-11-15.
      Copyright 2013-2014 Liam Deacon. All rights reserved.

      Licensed under the MIT license (see LICENSE file for details)

      Please send your feedback, including bugs notifications
      and fixes, to: liam.deacon@diamond.ac.uk

    usage:-

  optional arguments:
  -h, --help            show this help message and exit
  -b <bulk_file>, --bulk <bulk_file>
                        path to MTZ bulk or CLEED *.bul input file
  -i <slab_file>, --slab <slab_file>
                        path to MTZ slab or CLEED *.inp input file
  -t <temp_dir>, --tmpdir <temp_dir>
                        temporary directory for intermediate file generation
  -l <lmax>, --lmax <lmax>
                        Maximum angular momentum quantum number. [default: 10]
  -f <format>, --format <format>
                        Use specific phase shift format i.e. 'cleed', 'curve'
                        or 'none'. Choose 'curve' if you wish to produce
                        XYY... data for easy plotting. <format> is case
                        in-sensitive. [default: 'cleed']
  -r <energy> [<energy> ...], --range <energy> [<energy> ...]
                        Energy range in eV with the format:
                        '<start> <stop> [<step>]', where the <step> value is
                        optional.  Valid for relativistic calculations
                        only. [default: (20, 600, 5)]
  -S <subdir>, --store <subdir>
                        Keep intermediate files in subdir when done
  -v, --verbose         set verbosity level [default: None].
  -V, --version         show program's version number and exit
```

5.1.2 CLEED compatibility

It is possible to use this script to generate phase shift files iteratively during a geometry search for the CLEED package. In this manner phase shifts will be generated at the beginning of each cycle of the search.

For this to work, the environment variable `CSEARCH_LEED` must point to the `phsh.py` script, which will invoke the LEED program in `PHASESHIFT_LEED` after execution. When operating in this mode, the following assumptions are made:

1. *-b <bulk_file>* option is not needed and the filename is assumed by changing the file extension of *<slab_file>* to '.bul'
2. *-f CLEED* format is implied.
3. The generated phase shifts are stored in the directory set by the `CLEED_PHASE` environment variable, however a named copy with the iteration number (read from the matching '.log' file) will be placed in the same directory as the <slab_file>.
4. *<lmax>* is equal to 10, unless additional parameter syntax is given in the CLEED **.inp* file. To use phase shift specific lmax values, then add a new line with:

   ```
   lmax:  <phase_shift> <lmax>
   ```

   for each phase shift you wish to have a different lmax to that of the default.
5. The element and oxidation of each atom in a model is guessed by reading the phase shift tag from the CLEED input file. For example:

   ```
   po:  O_2-_COOH ...
   ```

   will be interpreted as a Oxygen with a -2 oxidation state and with a unique name tag of "O_2-_COOH" to show it is in a carboxylic group. Note the '-' may be at the beginning or the end of the oxidation sub-string. If no oxidation state is given then the atom is assumed to have zero charge.
6. The muffin-tin radius of the phase shift species is guessed from lines with:

   ```
   rm:  <phase_shift> <radius>
   ```

   However, if no value is found the radius is guessed from the ELEMENTS dictionary within `phaseshifts.elements` depending on the valency of the given phase shift element.

A full list of additional syntax to customise the generation of the phase shifts when using CLEED input files can be found in `phaseshifts.leed.Converter.import_CLEED()`.

Note: If the `PHASESHIFT_LEED` environment variable is not found, but `CLEED_PHASE` is, however, found then the program will place the generated files in this directory unless a specific `-S <subdir>` is provided.

CHAPTER

SIX

PHASESHIFTS API

6.1 Package Contents

This chapter covers the main modules of the phaseshifts and provides some API documentation for those wishing to incorporate this package into their own projects.

6.2 Subpackages

The main sub packages are listed below:

- `phaseshifts.gui` - includes all the necessary files for the graphical user interface.
- `phaseshifts.lib` - contains the Fortran libphsh library and the python wrappings.
- `phaseshifts.doc` - source documentation for the phaseshifts package.
- `phaseshifts.test` - modules for testing the phaseshift package.

6.3 Submodules

6.3.1 phaseshifts.atorb

atorb.py

Provides convenience functions for generating input and calculating atomic charge densities for use with the Barbieri/Van Hove phase shift calculation package.

See http://www.icts.hkbu.edu.hk/surfstructinfo/SurfStrucInfo_files/leed/

Requires f2py (for libphsh fortran wrapper generation)

Note: To generate libphsh fortran wrappers (libphsh.pyd) for your platform then use 'python setup.py' in the lib directory of this package to install into your python distribution. Alternatively, use:

```
f2py -c -m libphsh libphsh.f
```

Windows users may have to add appropriate compiler switches, e.g.

```
f2py -c -m libphsh --fcompiler=gfortran --compiler=mingw-32 libphsh.f
```

class `phaseshifts.atorb.`**`Atorb`**`(`***kwargs*`)`

Bases: `object`

Notes

Original author: Eric Shirley

There are nr grid points, and distances are in Bohr radii $a_0 \simeq 0.539$

$r(i) = r_{min} \cdot (r_{max}/r_{min})^{(i/n_r)}$, $i = 1, 2, 3, ...n_r - 1, n_r$

The orbitals are stored in phe(), first index goes $1...n_r$, the second index is the orbital index $(i...n_{el})$

Look at the atomic files after printing this out to see everything... Suffice it to say, that the charge density at radius $r(i)$ in units of electrons per cubic Bohr radius is given by:

$\sum_{j-1}^{n_el} occ(j) \cdot phe(i,j)^2/(4.0\,\pi\,r(i)^2)$

Think of the phe functions as plotting the radial wave-functions as a function of radius on a logarithmic mesh...

The Dirac equation is solved for the orbitals, whereas their density is treated by setting $phe(i,j)$ to Dirac's $\sqrt{F(i,j)^2 + G(i,j)^2}$ times the sign of $G(i,j)$...

So we are doing Dirac-Fock, except that we are not treating exchange exactly, in terms of working with major and minor components of the orbitals, and the phe's give the CORRECT CHARGE DENSITY...

The above approximation ought to be very small for valence states, so you need not worry about it...

The Breit interaction has been neglected altogether...it should not have a huge effect on the charge density you are concerned with...

static **`calculate_Q_density`**(***kwargs*)

Parameters **kwargs may be any of the following.** :

element : int or str, optional

Generate element atorb input file on the fly. Additional kwargs may be used to govern the structure of the input file - please use `help(phaseshifts.Atorb.gen_input)` for more information.

input : str, optional

Specify atorb input file otherwise will use the class instance value.

output_dir : str, optional

Specify the output directory for the *at_*.i* file generated, otherwise the default current working directory is used.

Returns **str** : filename

Examples

```
>>> Atorb.calculate_Q_density(input='atorb_C.txt')
      18.008635    -33.678535
       4.451786    -36.654271
       1.569616    -37.283660
       0.424129    -37.355634
       0.116221    -37.359816
       0.047172    -37.360317
       0.021939    -37.360435
       0.010555    -37.360464
       0.005112    -37.360471
       0.002486    -37.360473
       0.001213    -37.360473
```

```
        0.000593    -37.360473
        0.000290    -37.360474
     N  L  M  J  S  OCC.
     1    0  0   -1/2    1     2.0000         -11.493862
     2    0  0   -1/2    1     2.0000          -0.788618
     2    1  1   -1/2    1     0.6667          -0.133536
     2    1  1   -3/2    1     1.3333          -0.133311
   TOTAL ENERGY =        -37.360474   -1016.638262

>>> Atorb.calculate_Q_density(element='H')
        0.500007     -0.343752
        0.152392     -0.354939
        0.065889     -0.357254
        0.028751     -0.357644
        0.012732     -0.357703
        0.005743     -0.357711
        0.002641     -0.357712
        0.001236     -0.357713
        0.000587     -0.357713
        0.000282     -0.357713
  N  L  M  J  S  OCC.
     1    0  0   -1/2    1     1.0000          -0.229756
  TOTAL ENERGY =         -0.357713      -9.733932
```

static **gen_input** (*element*, ***kwargs*)

Parameters **element** : int or str

Either the atomic number, symbol or name for a given element

output : str, optional

File string for atomic orbital output (default: 'at_<symbol>.i')

ngrid : int, optional

Number of points in radial grid (default: 1000)

rel : bool, optional

Specify whether to consider relativistic effects

filename : str, optional

Name for generated input file (default: 'atorb')

header : str, optional

Comment at beginning of input file

method : str, optional

Exchange correlation method using either 0.0=Hartree-Fock, 1.0=LDA, -alpha = xalpha (default: 0.0)

relic : float, optional

Relic value for calculation (default: 0)

mixing_SCF : float, optional

Self consisting field value (default: 0.5)

tolerance : float, optional

Eigenvalue tolerance (default: 0.0005)

ech : float, optional

(default: 100)

static **get_quantum_info** (*shell*)

Returns tuple : (int, int, list[float, float], list[float, float])

(n, l, j=[l-s, l+s], occ=[n_r^-, n_r^+])

Notes

- *n* is the principle quantum number ($n > 0$).
- *l* is the azimuthal quantum number ($0 \leq l \leq n - 1$).
- *s* is the spin quantum number ($s \pm \frac{1}{2}$).
- *j* is the total angular momentum quantum numbers for both $l - s$ or $l + s$, respectively.
- n_r is the occupancy of the spin-split $l - s$ and $l + s$ levels, respectively.

static **replace_core_config** (*electron_config*)

Parameters electron_config : str

String containing the electronic configuration of the given element.

Returns str : :

A substituted string where the nobel gas core has been replaced.

Examples

```
>>> Atorb.replace_core_config('[Ar] 4s2')
'1s2 2s2 2p6 3s2 3p6 4s2'

>>> Atorb.replace_core_config('[Xe] 6s2 5d1')
'1s2 2s2 2p6 3s2 3p6 3d10 4s2 4p6 5s2 4d10 5p6 6s2 5d1'
```

6.3.2 phaseshifts.conphas

conphas.py

Provides a native python version of the conphas (phsh3) FORTRAN program by W. Moritz, which is distributed as part of the SATLEED code (see "Barbieri/Van Hove phase shift calculation package" section) and can be found at: http://www.icts.hkbu.edu.hk/surfstructinfo/SurfStrucInfo_files/ leed/leedpack.html

The Conphas() class also provides a number of convenience functions (see docstrings below).

Examples

```
>>> from os.path import join
>>> from phaseshifts.conphas import Conphas
>>> con = Conphas(output_file=join('testing', 'leedph_py.d'),
                  lmax=10)
>>> con.set_input_files([join('testing', 'ph1')])
>>> con.set_format('cleed')
>>> con.calculate()
```

class phaseshifts.conphas.**Conphas**(*input_files=[ ]*, *output_file=[ ]*, *formatting=None*, *lmax=10*, ***kwargs*)

Class Conphas

Notes

This work is based on the original conphas (phsh3) FORTRAN program by W. Moritz, which is distributed as part of the SATLEED code (see "Barbieri/Van Hove phase shift calculation package" section) and can be found at: http://www.icts.hkbu.edu.hk/surfstructinfo/SurfStrucInfo_files/ leed/leedpack.html

_Conphas__fix_path(*file_path*)

Fix escaped characters in filepath

_Conphas__set_data(*data=None*)

calculate()

Calculates continuous phase shifts from input file(s).

Examples

```
>>> con = Conphas(output_file=r'testing\leedph_py.d', lmax=10)
>>> con.set_input_files([r'testing\ph1'])
>>> con.set_format('cleed')
>>> con.calculate()
 L = 0
 jump between 25.0 eV and 30.0 eV; IFAK = -1
 L = 1
 jump between 65.0 eV and 70.0 eV; IFAK = -1
 L = 2
 jump between 20.0 eV and 25.0 eV; IFAK = 1
 jump between 80.0 eV and 85.0 eV; IFAK = 0
 L = 3
 L = 4
 jump between 275.0 eV and 280.0 eV; IFAK = 1
 L = 5
 L = 6
 L = 7
 L = 8
 L = 9
 L = 10
```

load_data(*filename*)

Load (discontinuous) phase shift data from file

Parameters **file** : str

Path to phase shift file.

Returns **tuple: (double, double, int, int, ndarray)** :

(initial_energy, energy_step, n_phases, lmf, data)

Notes

- *initial_energy* is the starting energy of the phase shifts.
- *energy_step* is the change in energy between consecutive values.
- *n_phases* is the number of phase shifts contained in the file.
- *lmf* is the maximum azimuthal quantum number considered.
- *data* is a (2 x n_phases) array containing the phase shift data.

`read_datafile` (*filename*)
Read in discontinuous phase shift file

Parameters **filename** : str

The path to the discontinuous phase shift file

`set_format` (*formatting=None*)
Set appropriate format from available options

Parameters **format** : str, optional

The format identifier for different packages; can be 'cleed' or None.

`set_input_files` (*input_files=*[ ])
set list of input filenames

`set_lmax` (*lmax*)
Set max orbital angular momentum (azimuthal quantum number)

Parameters **lmax** : int

Maximum azimuthal quantum number to be considered in calculations.

`set_output_file` (*output_file*)
set output filename

static `split_phasout` (*filename*, *output_filenames=*[ ])
split phasout input file into separate files

6.3.3 phaseshifts.elements

Properties of the chemical elements.

Each chemical element is represented as an object instance. Physicochemical and descriptive properties of the elements are stored as instance attributes.

Author Christoph Gohlke

Version 2013.03.18

Requirements

- CPython 2.7, 3.2 or 3.3

References

1. http://physics.nist.gov/PhysRefData/Compositions/
2. http://physics.nist.gov/PhysRefData/IonEnergy/tblNew.html
3. http://en.wikipedia.org/wiki/%(element.name)s
4. http://www.miranda.org/~jkominek/elements/elements.db

Examples

```
>>> from elements import ELEMENTS
>>> len(ELEMENTS)
109
>>> str(ELEMENTS[109])
'Meitnerium'
>>> ele = ELEMENTS['C']
>>> ele.number, ele.symbol, ele.name, ele.eleconfig
(6, 'C', 'Carbon', '[He] 2s2 2p2')
>>> ele.eleconfig_dict
{(1, 's'): 2, (2, 'p'): 2, (2, 's'): 2}
>>> sum(ele.mass for ele in ELEMENTS)
14659.1115599
>>> for ele in ELEMENTS:
...     ele.validate()
...     ele = eval(repr(ele))
```

6.3.4 phaseshifts.leed

Provides CLEED validator and Converter classes.

The CLEED_validator() class provides a method for checking the input files for errors, whereas the Converter.import_CLEED() method allows importing CLEED input files as a MTZ_model class

class `phaseshifts.leed.`**`CLEED_validator`**

Bases: `object`

Class for validation of CLEED input files

static **`is_CLEED_file`** (*filename*)

Determine if file is a CLEED input file

Returns **True** :

if a valid filename ending in any of .bul, .inp, .bsr, .bmin

False :

otherwise

`validate` (*filename*, *aoi=False*)

Checks CLEED input file for errors

Parameters **filename** : str

Path to input file. Should be *.bul , *.ctr, *.inp or *.bmin

aoi : bool

Check for angle of incidence parameters

class `phaseshifts.leed.`**`CSearch`**(*model_name*, *leed_command=None*)

Bases: `object`

`_getResults`()

`getIteration`(*iteration*)

`getRFactor`(*iteration*)

`getTimeStamp`(*iteration*)

`setModel`(*name*)

class `phaseshifts.leed.`**`Converter`**

Bases: `object`

Convert different input into phaseshift compatible input

static **`import_CLEED`**(*filename*)

Imports CLEED input file and converts model to muffin-tin input.

It assumes the following:

- the basis vectors a1, a2, & a3 are x,y,z cartezian coordinates
- if no a3 is found, the maximum z distance between atoms multiplied by four is given
- the unitcell is converted from cartezian to fractional coordinates
- atom sites are converted from Angstrom to Bohr units
- additional info from the phase shift filename is provided by splitting the '_' chars:
  1. First string segment is element or symbol, e.g. Ni
  2. Second string segment is the oxidation, e.g. +2
- lines with '**rm:**' provide the radii dictionary of the atomic species
- if no '**rm:**' found for that species, the atomic radius is used for zero valence, otherwise the covalent radius is used.

Additional information can, however, be provided using 'phs:' at the start of a line within the input file and may have the following formats:

1. "**phs:** c <*float*> nh <*int*> nform <*int*> exchange <*float*>"
2. "**phs:** <*phase_shift*> valence <*float*> radius <*float*>"

The identifiers `exchange`, `nform`, `valence` and `radius` may be abbreviated to `exc`, `nf`, `val` and `rad`, respectively.

Parameters **filename** : str

Path to input file.

Returns **phaseshifts.model.MTZ_model** :

Raises **IOError** : filename invalid

ValueError : bad formatting of input

6.3.5 phaseshifts.model

model.py

Provides convenience functions for generating input and calculating atomic charge densities for use with the Barbieri/Van Hove phase shift calculation package.

class `phaseshifts.model.`**`Atom`**(*element, coordinates=[0.0, 0.0, 0.0], **kwargs*)

Bases: `object`

Atom class for input into cluster model for muffin-tin potential calculations.

`set_coordinates`(*coordinates*)

`set_mufftin_radius`(*radius*)

Sets the muffin-tin radius of the atom in Angstroms.

`set_valence`(*valency*)

Sets the valency of the atom

exception `phaseshifts.model.`**`CoordinatesError`**(*msg*)

Bases: `exceptions.Exception`

Coordinate exception to raise and log duplicate coordinates.

class `phaseshifts.model.`**`MTZ_model`**(*unitcell*, *atoms*, ***kwargs*)

Bases: `phaseshifts.model.Model`

Muffin-tin potential Model subclass for producing input file for muffin-tin calculations in the Barbieri/Van Hove phase shift calculation package.

`calculate_MTZ`(*mtz_string='', **kwargs*)

Parameters **atomic_file** : str

The path to the atomic input file. If this is omitted the default is generate one using the MTZ_model.gen_atomic() method.

cluster_file : str

The path to the cluster input file which may be a bulk or slab model.

slab : int or bool

Determines whether the MTZ calculation is for a slab model (True). The default is a bulk calculation.

output : dict

Dictionary output of 'mtz' - muffin-tin potential & 'output_file' - the path to the MTZ output file.

Returns **output_files** : list(str)

Paths to the MTZ output file.

`create_atorbs`(***kwargs*)

Returns **output_files** : dict

Dictionary list of atorb*.i input files for the Atorb class to calculate the charge density from.

`gen_atomic`(***kwargs*)

Parameters **input_dir** : str

Input directory where at*.i files are kept.

input_files : tuple

List of input files to generate atomic input file from.

output_file : str

The filename of the resulting atomic*.i output file, which is simply a superimposed set of the radial charge densities from the individual input files.

Returns output_file : str

Returns the output file path string.

Raises IOError : exception

If either input directory or files do not exist.

Notes

If 'input_files' is not given then the default list of input files are inferred from the list of atoms in the model.

`gen_input`(***kwargs*)

Returns filename on success :

Raises CoordinatesError : exception

if the model atoms have duplicate coordinates and the 'pos_check' kwarg is set to True.

`get_MTZ`(*filename*)
Retrieves muffin-tin potential from file

`get_elements`()
Return the unique elements in model

`load_from_file`(*filename*)

Parameters filename : str

The path of the input file (e.g. cluster*.i or *slab*.i)

Raises IOError : exception

If the file cannot be read.

TypeError : exception

If a input line cannot be parsed correctly.

`set_exchange`(*alpha*)
Sets the alpha exchange term for muffin-tin calculation

`set_nform`(*nform*)
Sets form of muffin-tin calculation

Parameters nform : int or str

This governs the type of calculation, where nform can be:

1. "cav" or 0 - use Cavendish method
2. "wil" or 1 - use William's method
3. "rel" or 2 - Relativistic calculations

`set_nh`(*nh*)
Sets the nh muffin-tin zero estimation parameter

`set_slab_c`(*c*)

Examples

For Re the bulk c distance is 2.76Å, whereas a possible slab c distance could be ~10Å.

class `phaseshifts.model.`**`Model`**(*unitcell*, *atoms*, ***kwargs*)
Bases: `object`

Generic model class.

`_nineq_atoms`()

Returns **nineq_atoms, element_dict** : tuple

nineq_atoms [The estimated number of inequivalent atoms based on] the valence and radius of each atom.

element_dict [a dictionary of each element in the atom list where] each element contains an atom dictionary of 'nineq_atoms', 'n_atoms' and a complete 'atom_list'

`add_atom`(*element*, *position*, ***kwargs*)
Append an Atom instance to the model

Parameters **element** : str or int

Either an element name, symbol or atomic number.

position : list(float) or ndarray

(1x3) array of the fractional coordinates of the atom within the unit cell in terms of the lattice vector a.

`check_coordinates`()
Check for duplicate coordinates of different atoms in model.

Raises **CoordinateError** : exception

If duplicate positions found.

`set_atoms`(*atoms*)
Set the atoms for the model.

Parameters **atoms** : list(Atoms)

Array of Atom instances. Entries in the list which are not of type Atom will be ignored.

Raises **TypeError** : exception

If atoms parameter is not a list.

`set_unitcell`(*unitcell*)
Set the unitcell for the model

Parameters **unitcell** : Unitcell

Instance of Unitcell class to set to model.

Raises **TypeError** : exception

If unitcell parameter is not a Unitcell.

class `phaseshifts.model.`**`Unitcell`**(*a*, *c*, *matrix_3x3*, ***kwargs*)
Bases: `object`

Unitcell class

`set_a`(*a*)

Parameters a: float :

The magnitude of the in-plane lattice vector in Angstroms

Notes

To retrieve a in terms of Angstroms use 'unitcell.a', whereas the internal parameter 'unitcell._a' converts a into Bohr radii (1 Bohr = 0.529Å), which is used for the muffin-tin potential calculations in libphsh (CAVPOT subroutine).

`set_alpha`(*alpha*)

`set_beta`(*beta*)

`set_c`(*c*)

Parameters c : float

The magnitude of the in-plane lattice vector in Angstroms

Notes

To retrieve c in terms of Angstroms use 'unitcell.c', whereas the internal parameter 'unitcell._c' converts c into Bohr radii (1 Bohr = 0.529Å), which is used for the muffin-tin potential calculations in libphsh (CAVPOT subroutine).

`set_gamma`(*gamma*)

`set_vectors`(*m3x3*)

6.3.6 phaseshifts.phsh

phsh.py - quickly generate phase shifts

phsh provides convenience functions to create phase shifts files suitable for input into LEED-IV programs such as SATLEED and CLEED.

Examples

```
phsh.py -i *.inp -b *.bul -f CLEED -S phase_dir
```

CHAPTER

SEVEN

AUTHOR LIST

Below is a list of contributors who have helped to develop this package:

- Liam Deacon - *current maintainer*

7.1 Get Involved

If you would like to get involved in the phaseshifts project then please email liam.deacon@diamond.ac.uk.

CHAPTER EIGHT

LICENSE

CHAPTER

NINE

APPENDIX I: BARBIERI/VAN HOVE PHASE SHIFT PACKAGE - A BRIEF USER GUIDE

A. Barbieri, M.A. Van Hove

Lawrence Berkeley Laboratory, University of California, Berkeley, California 94720, USA

9.1 Acknowledgement notice

Please acknowledge use of the Barbieri/Van Hove phase shift package, as:

A. Barbieri and M.A. Van Hove, private communication.

9.2 Contact

M.A. Van Hove: vanhove@cityu.edu.hk

9.3 Contents

The following files should be included with the Barbieri/Van Hove distribution. If any are missing please contact Michel Van Hove (vanhove@cityu.edu.hk) for replacements.

- `phshift2007.rst` - This file contains this user guide to use the phase-shifts programs. It should be supplemented with the information contained in the input files provided. Includes definitions of I/O files, contents and basic hints on running the programs.

The files listed below contain FORTRAN programs that correspond to the basic steps necessary to obtain the phase shifts needed in a LEED structural determination.

- `PhSh0.for` - calculation of the atomic orbital charge densities.
- `PhSh1.for` - calculation of the muffin tin potential (bulk & slab).
- `PhSh2*.for` - calculation of phase shifts.
- `PhSh3.for` - removal of pi jumps from phase shifts.

INPUT-OUTPUT

Contains samples of the input and output files for the case of bulk Rh and a $\mathrm{Rh}(111)-(2\times 2)-\mathrm{C_2H_3}$ (H neglected) structure.

- *atorbC* : input of PhSh0 for C

- *atorbRh* : input of PhSh0 for Rh
- *atC.i* : output of PhSh0 for C
- *atomic.i* : input of PhSh1 for C_2H_3 on Rh(111)
- *clusRh* : input of PhSh1 for bulk Rh
- *clusC2Rh* : input of PhSh1 for C_2H_3 on Rh(111) (slab)
- *ph1* : input of PhSh3 (Rh)
- *ph2* : input of PhSh3 (C)
- *leedph.d* : output of PhSh3 (for C_2H_3 on Rh(111))

Not included are two output files:

- *mufftin.d* : output of PhSh1 for C_2H_3 (MTZ=-1.74)
- *phasout* : output of PhSh2rel for C_2H_3 on Rh(111)

9.4 Overview of the Programs

We explain here how to use the PHASE SHIFTS codes to obtain the phase shifts which are needed in a LEED calculation.

This documentation does not try to explain any of the details and subtleties of the calculation, but rather it simply tries to put anybody with a minimum knowledge of basic quantum mechanics in the position of obtaining good phase shifts. Additional documentation is contained as comments within some of the codes (but not all!).

The various codes have been obtained from different authors, whose names can be found in the source codes. The original codes were modified to make them more general and, at input-output level, so as to make their use more straightforward.

The codes have been tested on an IBM RISC 6000 workstation. There is no guarantee that the programs will work correctly when transported to different computers with different FORTRAN compilers.

Basically, the computation of phase-shifts appropriate for a LEED calculation can be divided into several distinct steps:

9.4.1 Step 0: `PhSh0.for`

Description

First we need to perform a free atom self-consistent calculation for each of the N elements for which phase shifts are required. This is accomplished by using a self-consistent Dirac-Fock (*i.e.* relativistic approach which computes, separately for each element, the self-consistent atomic orbitals. Notice that no local exchange approximation is made in these codes but some other minor approximations are used; see program for details.

The input needed at this stage is some basic information about the shell structure of the atom under consideration, an example of which is provided in the file ATORB for the case of Rhodium. The information required is usually contained in any advanced Chemistry or Solid State book (*e.g.* Ashcroft and Mermin, Solid State Physics, Saunders College, 1976).

The orbitals can then be used to compute the total radial charge densities associated to each element, which are collected in the file *atomic.i*.

Files

INPUT: *atorb*

OUTPUT: *atelem.i*

To summarize, the user will run `PhSh0.for` for the different inputs *atorb1*, *atorb2*, ... *atorbN*, corresponding to the N elements of interest and produce the corresponding output *atelem1.i*, *atelem2.i*, ... *atelemN.i* for the charge density of each of the N elements.

Note: The occupation number for each level corresponds to the total number of electrons filling that level. For instance, in the case of Rh, the orbital 3,2,2,-2.5 has $l = 2$ and $j = 2 + 1/2$. The occupancy of the filled level is then $N^{+}_{occ} = 2j + 1 = 6$. In the case of partially filled orbitals when the atomic configuration available does not distinguish between $l + 1/2$ and $l - 1/2$ levels, it is customary to assign the occupancy so that the ratio for the partially filled orbitals equals the ratio of the occupancies if those orbitals were completely filled. Consider for instance the case of Rh where the atomic configuaration (Ashcroft and Mermin) is [Kr]4 *d* 8 5 *s* 1. There is no ambiguity associated to the 5,0,0,1/2 level and $N_{occ} = 1$ in that case. As for the 4,2,2,3/2 and 4,2,2,5/2 levels the ratio of full occupancies is 4/6 ; the eight 4 *d* electrons will then be split among the two levels so as to preserve the 4/6 ratio: hence 3.2/4.8. The sum of all occupancies for a neutral atom should of course equal Z.

9.4.2 Step 1: `PhSh1.for`

Description

Run interactively

Now one computes the muffin tin potential by following Mattheiss' prescription (Ref. T. L. Loucks, Augmented Plane Waves Method, Benjamin, 1967). In essence, the atomic charge densities of the different elements making up the structure that we are interested in are superimposed to reflect the actual position of these elements in the structure. Note that for the purpose of obtaining the phase shifts needed in a LEED calculation it is not necessary to know the exact position of the atoms in the structure we are interested in, because the phase shifts and hence the calculated intensities are not strongly dependent on the manner in which the phase shifts are produced. (In principle, one could iterate the phase shift calculation after the LEED structure analysis to further refine the structure.) For the substrate atoms, a bulk terminated structure will be sufficient in almost all cases. In general, we prefer using a slab-supercell approach in defining the surface structure rather than embedding the adatoms in a sometimes artificial bulk structure. The slab is a free-standing film with a thickness of a few atomic layers, repeated periodically as a stack of identical slabs separated by slices of vacuum. The main subtlety about the slab approach is related to the definition of the muffin tin zero (see comment 3).

The total potential energy in each muffin-tin sphere is obtained by adding the electrostatic component computed by using the charge density distribution, and a local Slater-like exchange term. The final potential is then shifted to set its zero at the level of the average energy in the interstitial region (Muffin Tin Zero). This part of the program is relatively well documented.

Files

INPUT:

- *cluster.i* - Mainly contains the structural information about the slab which will be used to produce the muffin-tin potential. See example provided for a Rh crystal in *clusterRh.i* and for a $\mathrm{Rh}(111) - (2 \times 2) - \mathrm{C_2H_3}$ surface with H neglected in *clusterC2Rh.i*.
- *atomic.i* - It contains the atomic charge densities for the NINEQ inequivalent atoms specified in *cluster.i*. Furthermore, *atomic.i* has to be generated from the output *atelemJ.i* $J = 1, N$ by appending the *atelem** files

corresponding to the different elements in the order in which they appear as inequivalent atoms in the file *cluster.i*

- interactively: question: slab or bulk calculation? answer: 1 (slab) or 0 (bulk) enter value for bmtz (bulk muffin tin zero; see comment 3)

OUTPUT:

- *mufftin.d*
- *check.o*
- *bmtz* (if bulk calculation)

Note:

1. Cluster.i contains an option for producing output suitable for the three versions of the next step. The value of the alpha constant can be obtained from K. Schwarz, Phys. Rev. B 5, 2466 (1972)
2. Notice that an *atelem.i* corresponding to one element might need to be appended more than once to generate *atomic.i*. For instance in the case of *clusterRh.i* : *atomic.i = atelemRh.i + atelemRh.i + atelemRh.i*

   In the case of clusterC2Rh.i:

   atomic.i = atelemRh.i + atelemRh.i + atelemRh.i + atelemRh.i + atelemC.i + atelemC.i

   Where '+' indicates the appending of one file after the other
3. The specification of the Muffin tin zero requires some care when doing a calculation for a slab. Here by slab we mean a specified geometry in *cluster.i* with a large vacuum gap between slabs. The computed muffin tin zero (mtz) is the average of the energy in the interstitial region, including the vacuum: the average is highly distorted by the presence of the vacuum. A reasonable value for mtz is the bulk value even in the case of a slab calculation (small errors are anyway adjusted by the fitting of the inner potential in the LEED calculation). Therefore the suggested procedure is the following:

   - Perform first a bulk calculation for the substrate with the appropriate input files. When asked whether a bulk or slab calculation input 0 (bulk) and record the output value of bulk mtz
   - Perform a second slab calculation (of course now with different input files); input 1 for slab calculation and, when asked, use the previously recorded value as input for *bmtz*. The output of this second calculation will be used in STEP 2.

   Running this step interactively will clarify our points.

9.4.3 Step 2: `PhSh2cav.for`, `PhSh2wil.for` & `PhSh2rel.for`

Description

Here one computes the phase shifts from the muffin-tin potential(s).

An important detail is that, as a function of energy, the calculated phase shifts may, and often do, show discontinuities by ::math::*pi*, i.e. jumps by ::math::*pi* at some energies. Since the LEED programs interpolate phase shifts between energies at which they are provided, such discontinuities would give totally erroneous results at such discontinuities. Therefore these discontinuities must be removed: this is done internally in *PhSh2wil.for*, but separately in `PhSh3.for` after `PhSh2cav.for` or `PhSh2rel.for` is run.

Different packages

- `PhSh2cav.for` is a Cavendish program which produces non- relativistic phase shifts (Schroedinger equation), with possible discontinuities in energy.
- `PhSh2wil.for` is a program, written originally by Williams, which again produces non-relativistic phase shifts (Schroedinger equation), but without continuities in energy. This is the preferred program for non-relativistic phase-shifts calculations.
- `PhSh2rel.for` computes relativistic phase shifts (Dirac equation), but is possibly discontinuous in energy.

Files

INPUT:

- *mufftin.d* - (as output from STEP 1)

OUTPUT:

- *phasout*
- *dataph.d*
- *inpdat*
- *leedph.d* (in wil only)

Note:

1. Whether one can run the cav, wil or rel version depends on the input NFORM specified in STEP 1 in the input *cluster.i*.
2. The energy range (20-300 eV) for which phase shifts are computed, the energy spacing (5eV) and the number of phase-shifts (13) are set. An easy way to modify these is to use NFORM=2, because the values will appear in an obvious way in the input *mufftin.d*. Such input (the output of STEP 1) can be edited and the parameters can be modified for each of the inequivalent atoms in the calculation.
3. The output *phasout* contains the phase shifts of all the inequivalent atoms NIEQ (the number of such atoms was specified in *cluster.i* of STEP 1) in the calculation. *phasout* will be used to create the input files needed in STEP 3.
4. *dataph.d* is an output of the phase shifts in a form suited to plotting such data.

9.4.4 Step 3: `PhSh3.for`

Description

Run interactively

The phase shifts produced from *phsh2cav.for* and *phsh2rel.for* are not necessarily continuous in energy (since phase shifts are defined modulo pi). *phsh3.for* makes them continuous and produces output suitable as input for LEED programs. For the output of `phsh2wil.for`, `phsh3.for` is used to reformat the phase shifts.

Files

INPUT:

- *phJ* $J = 1, N$ generated from phasout. For this purpose *phasout* must be split into files each containing phase shifts of a single element. *phJ* will contain the phase shifts of the J ‘th element in the input file for the LEED programs (*i.e. tleed5.i*)

OUTPUT:

- *leedph.d*
- *dataph.d*

Note: The actual number of sets of phase-shifts that one might want to use in a LEED calculation might be different from NINEQ. It is quite typical for instance to use a single set of phase shifts to describe substrate atoms in different layers.

CHAPTER

TEN

INDICES AND TABLES

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

PYTHON MODULE INDEX