phaseshifts Documentation

Release 0.1.8

Liam Deacon

Feb 17, 2024

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

A local copy of the source files can be found under `phaseshifts/lib/.phsh.orig/phsh[0-2].f`.

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.

1.1 About the code

The example source codes provided in this package are intended to be instructional in calculating phase shifts. While it is not recommended to use the example code in production, the code should be sufficient to explain the general use of the library.

If you aren't familiar with the phase shift calculation process, you can read further information in `doc/` folder:

- *phshift2007.rst* - a brief user guide/documentation concerning the input files (& details of the original fortran *phshift* package).
- *phaseshifts API reference* - a more detailed overview of the library functions and how to calculate phase shifts using the convenience functions in this package. This is not yet finished and so the reader is referred to the above document for the time being.

For those wanting a crash course of the Van Hove / Tong programs, I advise reading the `phsh2007.txt` document. See the `examples/` directory to get an idea of the structure of the input files (for a random selection of models & elements). In particular see the `cluster_Ni.i` file for helpful comments regarding each line of input.

Tip: There is also a nice diagram overview (ignoring the LEED parts) contained within the AQuaLEED poster available online.

Those of you who are eager to generate phase shifts - first look at the example cluster files for a bulk and slab calculation, noting that the atoms in the model are in fractional units of the *a* basis vector for the unit cell (SPA units). Next, after creating a bulk and slab model in the `cluster.i` format, simply use the following python code:

```
>>> from phaseshifts.phsh import Wrapper as phsh
>>> phsh.autogen_from_inputs(bulk_file, slab_file)
```

This will hopefully produce the desired phase shift output files (at least for simple models) and works by assessing the two models to determine what output to produce. For more detailed documentation and function use refer to the pdf manual.

Tip: A standalone command line utility **phsh.py** is provided as a way of automating the generation of phase shifts as part of a script. For more information use:

```
phsh.py --help
```

Note: The *phaseshifts.leed* module provides a conversion class for CLEED `.inp` and `.bul` files. This is included as part of the *phsh.py* module, however the file extension is important (needs `.inp`, `.pmin`, `.bul`, or `.bmin`) and error checking is limited. There are also plans to include a validator to check the files for mal-formatted input at some point in the future.

1.1.1 Alternatives

A number of alternatives are available, notably the following:

1. AQuaLEED (with a useful poster overview of phaseshifts calculations). This is an officially mentioned piece of software on Michel Van Hove's LEED Calculation Homepage. Furthermore, although the poster mentions that the software is written in python, this software is not (currently) distributed on https://PyPI.org (or via alternative means such as a docker image on DockerHub) and therefore harder to intergrate with other python LEED-related projects such as CLEED and cleedpy.
2. Elastic Electron-Atom Scattering in Solids and Solid Surfaces (EEASiSSS) is authored by John Rundgren and first described in the paper: "J. Rundgren Phys. Rev. B 68 125405 (2003)". This program takes a different approach to calculating phase shifts by using optimised muffin-tin potentials for surface slabs with preassigned surface core-level shifts. Whilst the source code is not publicly available online (to this author's best knowledge), John Rundgren has been more than happy to assist when approached in the past.

   Note: It would be fantastic to include this software (and document it's use) as part of the phaseshifts python package allowing the user to choose the backend they wish to use for calculating phase shifts (e.g. `EEASiSSS` or `phshift2007`). As such John Rundgren should be contacted to see if he would be happy to collaborate on making this possible. This is being tracked by this item.

3. A fortran program is described in "McGreevy, E., & Stewart, A.L. (- Apr 1978). A program for calculating elastic scattering phase shifts for an electron colliding with a one-electron target using perturbation theory. Computer Physics Communications, 14(1-2), 99-107.", however this code is not publicly available online (pay-walled by journal).

Note: Should you know of alternatives, please either open an issue or (better yet) create a PR with changes to this documentation to keep this list up to date.

CHAPTER TWO

INSTALLING THE PHASESHIFTS PACKAGE

2.1 TDLR;

For python 3.11 or older:

```
pip install wheel numpy setuptools

# latest pypi release
pip install phaseshifts

# or local version with essential packages for development
git clone https://github.com/Liam-Deacon/phaseshifts
cd phaseshifts
pip install -e .[dev,test]  # !! best do this in a virtualenv

phsh --help
```

2.2 Details

The phaseshifts package requires CPython 2.7 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:

```
pip install phaseshifts
```

Note: Until a `pyproject.toml` with a working PEP-517 build backend is implemented then the user will first need to run `pip install numpy setuptools wheel` in order to have the necessary python pre-requisites available (along with a fortran compiler) in order to compile the FORTRAN source and wrap it to be available via python.

Tip: Running `make check` with run a test suite designed to catch issues with the installation (however the `pytest` package is required).

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) or Anaconda with mingw (gcc & gfortran) installed is highly recommended. Mac OS X users can simply do `brew install gfortran` and Debian/Ubuntu users can do `sudo apt-get install -y gfortran`.

Note: On unix systems, setup the virtualenv on Python 3.10 or lower, activate it and run *make*.

Warning: Python 3.12 compatibility is a work in progress due to the removal of `numpy.distuils` build backend for `f2py` preventing simple installation via `pip install`, this github issue tracks progress on fixing this known issue.

CHAPTER

THREE

RUNNING

The *phsh.py* script (available after installing the package) aims to simplify these steps with a single command.

The simplest and most reliable cross-platform way to run *phsh.py* is through docker:

```
# obtain the image
docker pull ghcr.io/Liam-Deacon/phaseshifts:latest  # should only need to do this once

# run phsh.py via the docker image
docker run ghcr.io/Liam-Deacon/phaseshifts:latest  # will display usage

# or more generally (adjust as needed)
docker run ghcr.io/Liam-Deacon//phaseshifts:latest -v /path/to/host/input/data:/data [
↪<phsh-args> ...]
```

Tip: Development docker images can be built locally, e.g. `DOCKER_TAG=dev make docker`

Warning: There is a known possible bug where the compiled `libphsh.f` is not thread-safe (as ascertained by the fortran compiler), as such if you anticipate using this library in concurrent environments then it is advised to run `phsh.py` via `docker run ghcr.io/Liam-Deacon/phaseshifts:latest` as this works around this limitation due to the emphereal nature of container instances created using `docker run`.

CHAPTER

FOUR

CONTACT

This package is developed/maintained in my spare time so any bug reports, patches, or other feedback are very welcome!

The project is (still) in the early developmental stages and so anyone who wishes to get involved are most welcome, go to https://github.com/Liam-Deacon/phaseshifts/issues to get started.

CHAPTER

FIVE

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.*

5.1 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 SIX

SCRIPTS

6.1 phsh.py

6.1.1 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']
```

(continues on next page)

(continued from previous page)

```
-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
```

6.1.2 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

SEVEN

PHASESHIFTS API

7.1 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.

7.2 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.

7.3 Submodules

7.3.1 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.

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

# 64-bit
f2py -c -m libphsh --fcompiler=gfortran --compiler=mingw-64 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'
```

phaseshifts.atorb.**get_electron_config**(*element_obj*)

Obtain the electronic orbital configuration for the given *element_obj*.

phaseshifts.atorb.**get_element**(*element: str*, *backend: Literal['mendeleev', 'elementy', 'periodictable'] | None = None*) → object

Obtain an element object for querying information using *backend*.

7.3.2 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*)

Bases: object

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

__fix_path(*file_path*)

Fix escaped characters in filepath

__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

7.3.3 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))
```

7.3.4 phaseshifts.leed

7.3.5 phaseshifts.model

7.3.6 phaseshifts.phsh

CHAPTER

EIGHT

PHSHIFT2007 PORTING NOTES

This document contains notes on porting the original Barbieri / Van Hove phshift2007 phase shift package code into the phaseshifts package.

In order to compile the FORTRAN code on a modern system, a number of changes were made, including:

- The original code was written in FORTRAN 77 with some FORTRAN 66 features that are not allowed in common open source f77 compilers such as `gfortran` and `flang`. In addition, some language-intrinsic functions were not portable such as `DFLOAT` and were replaced with the equivalent standard functions, e.g. `DFLOAT` (a GNU extension) was replaced with `DBLE`[1] and `dexp` replaced with `exp`.
- In order to use as a callable library, `PROGRAM` units were replaced with `SUBROUTINE` blocks. As part of this process these subroutines gained additional parameters for the input/output file names. Without these changes there would be multiple main blocks and the code would not be suitable for inclusion in a shared library.
- Data types were updated to be FORTRAN 77 compatible. Notably, `HOLLERITH`[2] constants representing string constants were replaced with the standard `CHARACTER` type.
- Leading tabs were replaced with 7 spaces to avoid `-Wtabs` warnings on `gfortran` due to the fact that tabs are not members of the Fortran Character Set[3] .

Additional changes were made to improve the readability of the code:

- Translated the comments in `PHSH3.FOR` (i.e. the CONPHAS program) from German to English. A python implementation can be found under `phaseshifts.conphas`.
- Trailing whitespace was removed from all lines.
- Continuation lines such as those starting with `'     1'` were replaced with `'     +'` to improve readability by more easily distinguishing continuation lines from labels.
- `real*8` was replaced with `double precision` (and related casts, i.e. `dfloat` to `dble`).
- `GOTO` semantics and DO with labels were refactored to more closely resemble other modern programming language constructs.
- `c$OMP PARALLEL DO` blocks were added where appropriate to allow for parallel execution of loops via OpenMP. This is only enabled when compiling with `-fopenmp`.

Danger: Even with the original code, the LEED Calculation Home Page offers no guarantees of correctness in the calculated phase shifts. Furthermore, compiling the code with a modern compiler against an unknown benchmark means that there are no assurances that the compiled programs will produce the same results as the original code authors' intended. There are probably plenty of bugs, as such please open an issue if you find any.

[1] https://github.com/Liam-Deacon/phaseshifts/commit/fbd701e20f83d4eca90e5d90ef696d8316717d41
[2] https://en.wikipedia.org/wiki/Hollerith_constant#Examples
[3] https://gcc.gnu.org/onlinedocs/gfortran/Error-and-Warning-Options.html#index-Wtabs

Note: A notebook guiding the user through the initial porting process can be found at porting-phshift2007-notes.ipynb

Warning: Significant changes were made to the original code in order to port it for `f2py`. Artifacts of this process are likely present in the code and may cause unexpected behaviour that deviates from the original intended purpose. As such, if you are looking for a reference implementation of the original code, please run `make phshift2007` and `sudo make install`, which will download the original phshift2007 package, compile the `phsh*` programs and install them to `$PREFIX/bin` (this is `/usr/local/bin` by default). They will then be available to run from the command line.

Tip: Should you not trust the bundled f2py library, then a future version of `phsh.py` will allow you to run the original phshift2007 programs via wrapped subprocess calls.

8.1 Compiler Notes

When originally porting the code back in 2014, the code was compiled with f2py, Python 2.7 (32-bit) and the mingw32 toolchain on Windows 7 (installed together via Python(x, y) version <2.7.6.1). This was a more permissive compiler toolchain than modern GCC-toolchain gfortran and LLVM-based (classic) flang compilers tested for the v0.1.7 release.

Note: According to wikipedia[4] `g77` is no longer included in the GCC project since v4 as the maintainer decided to no longer support it. Another prominent fortran compiler `g95` was also discontinued in 2012 and has diverged considerably from the original GNU compiler collection. As such it is possible that the fortran compiler included in the mingw32 toolchain used in the original porting was one of these compilers and this would explain why additional changes were required to compile the code with modern compilers.

Tip: Those wishing to perform a Windows build would be advised to use Anaconda and can be installed on Windows 10/11 using **`winget install --id Anaconda.Anaconda3`**. Once installed, the conda environment can be installed with **`conda env create -f environment.yml`** and activated with **`conda activate phaseshifts`**. The phshift2007 code *could* then be compiled with **`gfortran -static-libgcc -static-libgfortran ...`** (however no modern Windows build has been tried yet)

8.2 Compiler Test Matrix

The following table compilers provides some summary information on compilers and platforms tested:

[4] https://en.wikipedia.org/wiki/GNU_Compiler_Collection#Fortran

Compiler	Version	Platform	Architecture	Status	Notes	Date Tested	Commit / Tag
gfortran	11	Ubuntu 22.04	x86_64	✓	Built via `ubuntu-latest` GitHub Action runner[5]	2024-01-21	v0.1.8[6]
gfortran	11	Mac OS X 12	x86_64	✓	Built via `macos-latest` GitHub Action runner[Page 29, 5]	2024-01-21	v0.1.8[6]

8.3 Known Issues

The following issues are known to exist in the current version of the code:

1. The code is not thread-safe. This is due to the use of global variables in the original code as well as large arrays that do not fit into stack memory. This is not a major issue if the user is aware of this and the code is not used in a multi-threaded context. Should the user need to ensure thread-safety, a workaround is to run via ephemeral docker containers, see *Running* section.
2. Many minor compiler warnings have been ignored, such as those related to implicit typing of variables. These should be fixed in future releases.

[5] https://github.com/Liam-Deacon/phaseshifts/actions/workflows/publish-to-pypi.yaml
[6] https://github.com/Liam-Deacon/phaseshifts/releases/tag/v0.1.8

CHAPTER NINE

AUTHOR LIST

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

- Liam Deacon - *current maintainer*

9.1 Get Involved

If you would like to get involved in the phaseshifts project then please create an issue or a discussion on GitHub.

CHAPTER

TEN

LICENSE

CHAPTER

ELEVEN

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

11.1 Acknowledgement notice

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

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

11.2 Contact

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

11.3 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 $\mathrm{Rh}(111)-(2\times 2)-C_2H_3$ (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)

11.4 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:

11.4.1 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.

11.4.2 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.

11.4.3 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.

11.4.4 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 TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX