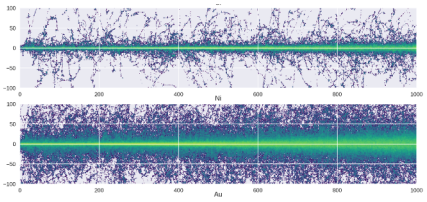
pysrim Documentation

Release 0.2.1

Chris Ostrouchov

Nov 08, 2018

Contents:



pysrim is a python package that aims to wrap and extend SRIM a popular tool for simulating ions traveling through a material. There are many pain points to SRIM and this package aims to address them. These inclue compatibility with all OS's, automation and crash recovery of SRIM calculations, parsing of all output files, and publication quality plots.

`pysrim` is heavily tested and documented. See `modindex` to search through api documentation or using the search to find a specific function.

Please see installation for getting pysrim intalled on your machine. Next you will want to follow the simple tutorial to get started. For a much more advanced example please see SiC ion damage production jupyter notebook.

CHAPTER 1

Installation

Installation of `pysrim` is easy via pip or conda. If you do not have python installed on your machine and are new to python I would suggest using anaconda.

Available on PyPi

- `pip install pysrim`

Available on Conda

- `conda install -c costrouc pysrim`

Available on Docker **recommended** no SRIM installation necessary

- `docker pull costrouc/pysrim`

You **do not** need to install SRIM if you are just doing analysis. Otherwise for windows this is straightforward and normal while for Linux and OSX you will need *wine* additionally installed.

1.1 Docker

There is a docker container with *pysrim* and SRIM already installed. Some interesting tricks had to be done with using wine and faking an X11 session. ‘xvfb-run -a ‘ creates a fake X11 session within the docker container therefore allowing SRIM to run on servers without displays. This is the method that I always use to run SRIM calculations.

Image: costrouc/pysrim

See examples/docker for an example of how to use the docker image.

1.2 Linux and OSX

For linux an OSX you will need to first have wine installed. See this post on installation of wine on OSX. For linux you will typically be able to install wine via `apt get install wine` or `yum install wine`. SRIM is compatible with wine.

Once you have wine installed run the installer script `install.sh`.

Click extract and then done. The installed version should be SRIM 2013. To check for this see that an executable `TRIM.exe` is in the directory.

1.3 Windows

A collegue of mine has gotten it to work easily on Windows but I myself have no experience. Just download the executable at srim.org. Next you will extract the SRIM files into a directory on your windows machine. Note the directory of installation as it will be needed from *trim.run()*. Make sure that your installed version is SRIM 2013. To check for this see that an executable `TRIM.exe` is in the directory.

CHAPTER 2

Tutorial

If you have already installed srim then lets get started! If not please make sure to look at installation first.

In this tutorial we will cover all of the basics of `pysrim`.

- running a TRIM simple calculation
- analyzing TRIM calculation output files

Most emphasis will be put on what is possible for analysis of SRIM calculations. Since all output files will be exposed as numpy arrays the sky is the limit for plotting.

We will assume python 3 for this notebook and you should be using it too. Python 2.7 is going be depreciated in 2020.

```
import os

import numpy as np
import matplotlib.pyplot as plt

from srim import TRIM, Ion, Layer, Target
from srim.output import Results
```

There are not too many important objects to import from srim. The first srim import is for all of the components needed for automating a SRIM calculation. While the second import is for the output results.

2.1 Run a SRIM Calculation

Running a srim calculation is much like the gui that SRIM provides.

Concepts:

- a list of Layers forms a Target
- a Layer is a dict of elements, with density, and a width
- an Element can be specified by symbol, atomic number, or name, with a custom mass [amu]

- an Ion is like an Element except that it also requires an energy in [eV]

```
# Construct a 3MeV Nickel ion
ion = Ion('Ni', energy=3.0e6)

# Construct a layer of nick 20um thick with a displacement energy of 30 eV
layer = Layer({
        'Ni': {
            'stoich': 1.0,
            'E_d': 30.0,
            'lattice': 0.0,
            'surface': 3.0
        }}, density=8.9, width=20000.0)

# Construct a target of a single layer of Nickel
target = Target([layer])

# Initialize a TRIM calculation with given target and ion for 25 ions, quick␣
↪calculation
trim = TRIM(target, ion, number_ions=25, calculation=1)

# Specify the directory of SRIM.exe
# For windows users the path will include C://...
srim_executable_directory = '/tmp/srim'

# takes about 10 seconds on my laptop
results = trim.run(srim_executable_directory)
# If all went successfull you should have seen a TRIM window popup and run 25 ions!
```

The following code does a quick SRIM calculation of a 3 MeV Nickel ion in a nickel target. We have set the displacement energy for Nickel at 30 eV with a density of 8.9 [g/cm^3]. Also notice that after running the simulation with `trim.run` the results are automatically parsed for us. After the calculation has completed many times we will want to copy the results to a different directory.

```
output_directory = '/tmp/srim_outputs'
os.makedirs(output_directory, exist_ok=True)
TRIM.copy_output_files('/tmp/srim', output_directory)
```

If at a later point you would like to parse the srim calcualtions you can use the `srim.output.Results` class to gather all the output.

```
srim_executable_directory = '/tmp/srim'
results = Results(srim_executable_directory)
```

2.2 Plotting and Analysis of Results

Now we assume that we have completed several interesting SRIM calculations. For this tutorial we will use results within the `pysrim` repository. You will need to download these files. We will analyze results such as damage energy, ionization, and vacancy production.

```
def plot_damage_energy(folder, ax):
    results = Results(folder)
    phon = results.phonons
    dx = max(phon.depth) / 100.0 # to units of Angstroms
    energy_damage = (phon.ions + phon.recoils) * dx
```

(continues on next page)

(continued from previous page)

```
    ax.plot(phon.depth, energy_damage / phon.num_ions, label='{}'.format(folder))
    return sum(energy_damage)

def plot_ionization(folder, ax):
    results = Results(folder)
    ioniz = results.ioniz
    dx = max(ioniz.depth) / 100.0 # to units of Angstroms
    ax.plot(ioniz.depth, ioniz.ions, label='Ionization from Ions')
    ax.plot(ioniz.depth, ioniz.recoils, label='Ionization from Recoils')

def plot_vacancies(folder, ax):
    results = Results(folder)
    vac = results.vacancy
    vacancy_depth = vac.knock_ons + np.sum(vac.vacancies, axis=1)
    ax.plot(vac.depth, vacancy_depth, label="Total vacancies at depth")
    return sum(vacancy_depth)

folders = ['test_files/2', 'test_files/4']
image_directory = 'examples/images'
os.makedirs(image_directory, exist_ok=True)
```

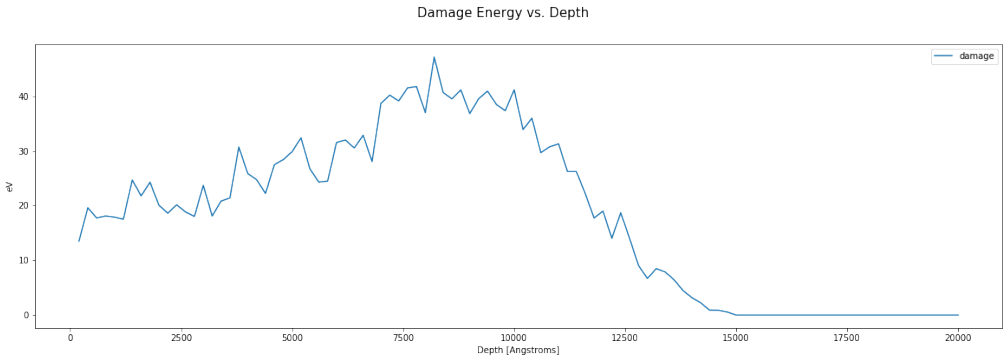
Here we initialize three plotting functions.

Damage energy vs depth.

```
fig, axes = plt.subplots(1, len(folders), sharex=True, sharey=True)

for ax, folder in zip(np.ravel(axes), folders):
    energy_damage = plot_damage_energy(folder, ax)
    print("Damage energy: {} eV".format(energy_damage))
    ax.set_xlabel('Depth [Angstroms]')
    ax.set_ylabel('eV')
    ax.legend()

fig.suptitle('Damage Energy vs. Depth', fontsize=15)
fig.set_size_inches((20, 6))
fig.savefig(os.path.join(image_directory, 'damagevsdepth.png'), transparent=True)
```



Ionization energy vs depth

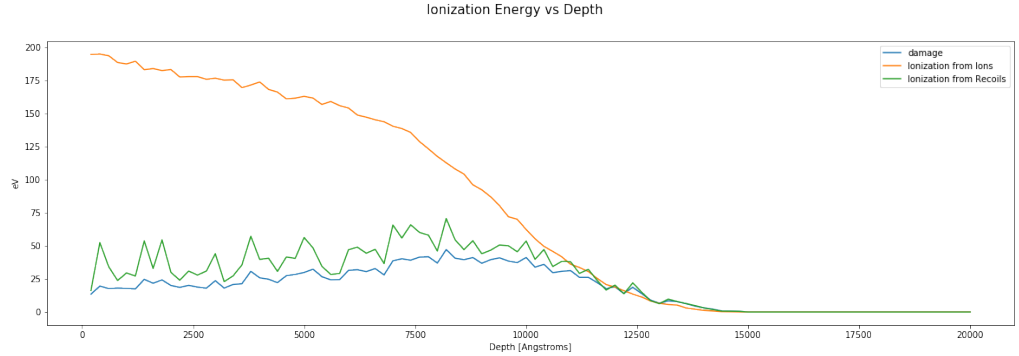
```
fig, axes = plt.subplots(1, len(folders), sharey=True, sharex=True)

for ax, folder in zip(np.ravel(axes), folders):
```

(continues on next page)

(continued from previous page)

```
    plot_damage_energy(folder, ax)
    plot_ionization(folder, ax)
    ax.legend()
    ax.set_ylabel('eV')
    ax.set_xlabel('Depth [Angstroms]')
fig.suptitle('Ionization Energy vs Depth', fontsize=15)
fig.set_size_inches((20, 6))
fig.savefig(os.path.join(image_directory, 'ionizationvsdepth.png'), transparent=True)
```

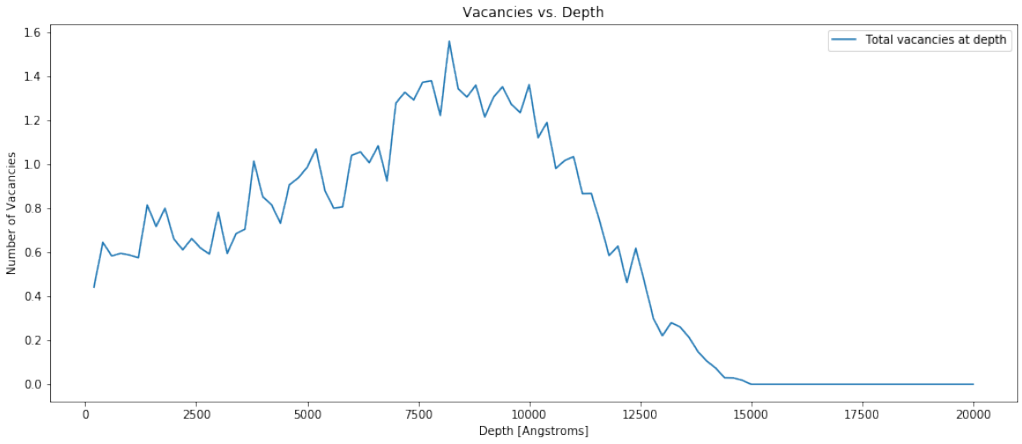


Total number of vacancies vs depth.

```
fig, ax = plt.subplots()

for i, folder in enumerate(folders):
    total_vacancies = plot_vacancies(folder, ax)
    print("Total number of vacancies {}: {}".format(folder, total_vacancies))

ax.set_xlabel('Depth [Angstroms]')
ax.set_ylabel('Number of Vacancies')
ax.set_title('Vacancies vs. Depth')
ax.legend()
fig.set_size_inches((15, 6))
fig.savefig(os.path.join(image_directory, 'vacanciesvsdepth.png'), transparent=True)
```



For a much more advanced example please see SiC ion damage production jupyter notebook. This tutorial is also available in notebook form.

CHAPTER 3

Benchmarks

3.1 Docker vs Standard Linux

I wanted to benchmark running srim calculations on normal linux vs within a docker container. TRIM on linux showed more variability between job runs. When I refer to TRIM on linux it is equivalent to `wine TRIM.exe`.

I used the parallel command to test differing number of cores. Yes this is a simple approach but will be in the right ballpark and trends are certainly compelling for the docker container. All test we done on my two core Lenovo t440s.

Docker startup cost is 8.5 seconds vs 5.8 seconds for standard linux process. But the docker container is significantly faster using =xvfb-run= best performance is linux 8.3 ions/second vs docker 13.2 ions/second. These tests indicate that the docker container may have better performance due to rending in a virtual X frame buffer.

The python simple script to be run is a Nickel in Nickel irradiation.

```
import os
from srim import Ion, Layer, Target, TRIM

ion = Ion('Ni', energy=3.0e6)
layer = Layer({
        'Ni': {
            'stoich': 1.0,
            'E_d': 30.0,
            'lattice': 0.0,
            'surface': 3.0
        }}, density=8.9, width=20000.0)
target = Target([layer])
trim = TRIM(target, ion, number_ions=100, calculation=1)
srim_executable_directory = '/tmp/srim'
results = trim.run(srim_executable_directory)
os.makedirs('/tmp/output', exist_ok=True)
TRIM.copy_output_files('/tmp/srim', '/tmp/output')
```

And the benchmarks results.

```
time parallel -j 6 python ni.py -- 1 2
```

ions	cores	linux [s]	linux [ion/s]	linux [ion/s core]
1	1	5.8	0.17241379	0.17241379
100	1	29	3.4482759	3.4482759
200	1	53	3.7735849	3.7735849
200	2	35	5.7142857	2.8571429
400	2	63	6.3492063	3.1746032
300	3	43	6.9767442	2.3255814
600	3	79	7.5949367	2.5316456
400	4	55	7.2727273	1.8181818
800	4	101	7.9207921	1.9801980
1600	4	192	8.3333333	2.0833333
500	5	65	7.6923077	1.5384615
1000	5	121	8.2644628	1.6528926

```
time parallel -j 6 \
         docker run \
              -v $PWD/examples/docker/:/opt/pysrim/ \
              -v /tmp/output:/tmp/output
              -it costrouc/pysrim sh -c "xvfb-run -a python3.6 /opt/pysrim/ni.py" --␣
↪1 2
```

ions	cores	docker [s]	docker [ion/s]	docker [ion/s core]
1	1	8.5	0.11764706	0.11764706
100	1	27	3.7037037	3.7037037
200	1	46	4.3478261	4.3478261
200	2	31	6.4516129	3.2258065
400	2	52	7.6923077	3.8461538
300	3	39	7.6923077	2.5641026
600	3	69	8.6956522	2.8985507
400	4	39	10.256410	2.5641026
800	4	67	11.940299	2.9850746
1600	4	121	13.223140	3.3057851
500	5	51	9.8039216	1.9607843
1000	5	69	14.492754	2.8985507

CHAPTER 4

Indices and tables

- genindex
- modindex
- search