
spack Documentation

Release 0.1

Wendell Smith

January 03, 2017

1	A 2D or 3D sphere packing analysis package	3
2	Installation	5
3	Example Usage	7
4	Full Contents	11
4.1	Example Usage of spack	11
4.2	Reference	12
5	Indices and tables	15
	Python Module Index	17

Code available on [Github](#), and [documentation](#) at Read the Docs.

A 2D or 3D sphere packing analysis package

This package exists to enable fast, simple, and easy analysis of packings of spheres (3D) and disks (2D). It was developed for use in a Granular Materials lab, so some of the methods reflect that.

Installation

This library requires `numpy` at a minimum, which you will probably want to have installed before installing this. Optionally, you may want `vapory` and `povray` are required for making pretty pictures, and Voronoi tessellations are provided by `tess`. The optional dependencies can be installed at any time, at which point the associated methods (`scene()` and `tess()`) will work.

To install, use `pip` (or `easy_install`):

```
pip install --user spack
```

Or to install from [Github](#):

```
pip install --user git+git://github.com/wackywendell/spack@master
```

Example Usage

Make a *Packing*:

```
>>> from spack import Packing
>>> from numpy import array, pi
>>> L = 2.0066668050219723
>>> diameters = array([ 0.96,  0.97,  0.98,  0.99,  1. ,  1.01,  1.02,  1.03,  1.04])
>>> locs = array([[ 1.40776762,  1.26647724,  0.73389219],
...               [ 0.58704249,  2.11399   ,  1.52956579],
...               [ 1.75917911,  0.54290089,  1.27577478],
...               [ 2.13750384,  0.87508242,  0.21938647],
...               [ 1.07283961,  0.87692084,  1.9060841 ],
...               [ 0.09550267,  1.94404465,  0.56463369],
...               [ 1.07636871,  2.1942971 ,  0.63752152],
...               [ 0.49922725,  1.20002224,  1.13360082],
...               [-0.27724757,  1.62152603,  1.67262247]])
>>> pack = Packing(locs, diameters, L=L)
```

How many contacts are in my packing?

```
>>> pack.contacts()
(25, 25, 0)
>>> # 25 contacts found
>>> # 25 contacts required for stability (for a packing of 9 particles)
>>> # 0 floaters
```

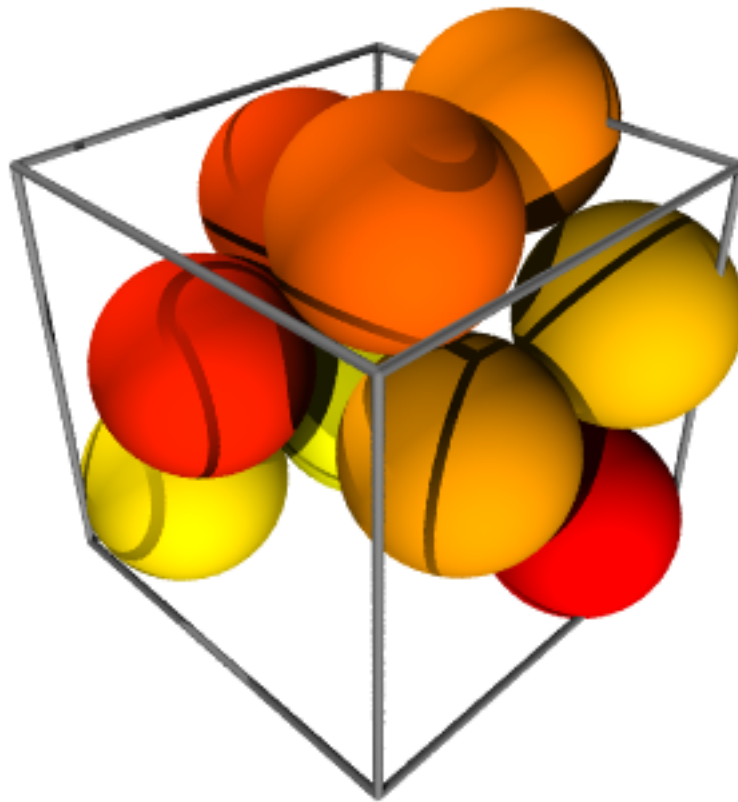
What are its highest resonant frequencies?

```
>>> freqs = pack.DM_freqs()
>>> for f in reversed(sorted(freqs)[-4:]):
...     print('{:.4f}'.format(f))
0.7018
0.6717
0.6416
0.6375
```

What does it look like? (requires *vapory* package and *povray* program)

```
>>> size = 400
>>> sc = pack.scene(rot=pi/4, camera_dist=2, cmap='autumn')
>>> sc.render('example-packing.png', width=size, height=size, antialiasing=0.0001)
```

Colors indicate diameter, with floaters drawn in gray.



docs/example-packing.png

Let's look at all sides, using *moviepy*:

```
>>> from moviepy.editor import VideoClip
>>> import moviepy.editor as mpy
>>>
>>> duration = 10
>>> def make_frame(t):
...     return (
...         pack.scene(rot=(t/duration + .125)*2.*pi,
...                    camera_dist=2, cmap='autumn', bgcolor=[1,1,1])
...         .render(width=size, height=size, antialiasing=0.001)
...     )
>>> vc = VideoClip(make_frame, duration=duration)
>>>
```

```
>>> vc.write_gif("example-packing.gif", fps=24)
```

And this is the output:

There are a few other methods for things like finding the backbone of the packing (*backbone()*), the adjacency matrix (*neighbors()*), or getting the Voronoi tessellation (*tess()*, requires *tess*). See the [Reference](#) for more details.

Full Contents

4.1 Example Usage of spack

This is an ipython notebook in `spack/docs/example-usage.ipynb`, demonstrating how to use spack.

First, we import our modules:

```
import spack
from math import pi
```

Now we put in some data.

The data below is from a simple packing I made using `pyfarm.packmin`. Normally you'd load your own data from a file.

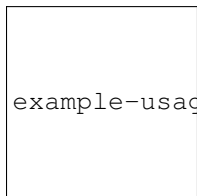
```
L = 2.0066668050219723
diameters = [ 0.96, 0.97, 0.98, 0.99, 1. , 1.01, 1.02, 1.03, 1.04]
locs = [[ 1.40776762, 1.26647724, 0.73389219],
        [ 0.58704249, 2.11399 , 1.52956579],
        [ 1.75917911, 0.54290089, 1.27577478],
        [ 2.13750384, 0.87508242, 0.21938647],
        [ 1.07283961, 0.87692084, 1.9060841 ],
        [ 0.09550267, 1.94404465, 0.56463369],
        [ 1.07636871, 2.1942971 , 0.63752152],
        [ 0.49922725, 1.20002224, 1.13360082],
        [-0.27724757, 1.62152603, 1.67262247]]
```

Now we make the packing:

```
pack = spack.Packing(locs, diameters, L=L)
```

What does it look like?

```
size = 400
sc = pack.scene(rot=pi/4, camera_dist=2, cmap='autumn', bgcolor=[1,1,1])
sc.render('ipython', width=size, height=size, antialiasing=0.001)
```



example-usage_files/example-usage_9_0.png

Let's make a movie!

```
from moviepy.editor import VideoClip
import moviepy.editor as mpy

duration = 10
def make_frame(t):
    return (
        pack.scene(rot=(t/duration + .125)*2.*pi,
                    camera_dist=2, cmap='autumn', bgcolor=[1,1,1])
        .render(width=size, height=size, antialiasing=0.001)
    )
vc = VideoClip(make_frame, duration=duration)
```

Write the movie to file. This takes about 10 minutes on my machine to render all 240 frames, but it does give you some pretty spiffy output.

```
vc.write_gif("example-packing.gif", fps=24)
```

```
[MoviePy] >>>> Building file example-packing.gif
[MoviePy] Generating GIF frames...
[MoviePy] Optimizing the GIF with ImageMagick...
[MoviePy] >>>> File example-packing.gif is ready !
```

And display the movie, in an ipython notebook.

```
from IPython.display import Image
Image(url="example-packing.gif")
```

4.2 Reference

spack: a package for spherical packing analysis

class `spack.Packing` (*rs, diameters, shear=0.0, L=1.0*)

A class representing a packing of spheres in a periodic box.

DM (*masses=None*)

Dynamical matrix for array *rs*, size *ds*. Assumes *epsilon* is the same for all.

Parameters *masses* : an array of length *N* of the masses of the particles.

DM_freqs (*masses=None*)

Find the frequencies corresponding to the eigenvalues of the dynamical matrix.

This is just a short wrapper around `DM()`.

backbone (*tol=1e-08*)

Returns (backbone indices, neighbor matrix)

cages (*M=10000, R=None, Rfactor=1.2, padding=0.1, Mfactor=0.1*)

Find all cages in the current “packing”.

The algorithm uses Monte Carlo: it finds *M* random points within a sphere of radius *R* from each particle, and sees if that particle could sit there without conflicting with other particles. Then (number of accepted points) / (number of test points) * (volume of sphere) is the volume of the cage.

The algorithm is adaptive: if not enough test points are accepted ($n < M * Mfactor$), it tries more test points. If any test points are within *padding* of the edge, *R* is (temporarily) expanded.

Parameters **M** : Number of points in the sphere to test

R : Size of sphere to test (will be expanded if necessary)

Rfactor : How much to increase R by when the cage doesn't fit

padding : How much larger the sphere should be than the cage (if it isn't, the sphere is expanded)

Mfactor : Mfactor * M is the minimum number of points to find per cage. If they aren't found, more points are tested.

Returns **points** : a list of (A x 3) lists, A indeterminate (but larger than M * Mfactor), with each list corresponding to the points within one cage.

Vs : The approximate volumes of each cage.

contacts (*tol=1e-08*)

Returns (number of backbone contacts, stable number, number of floaters)

dist (*other, tol=1e-08, maxt=1000000*)

dist_tree (*other, tol=1e-08*)

Find the distance between two packings.

Requires pyparm.

forces ()

Find Fij on each particle, assuming a harmonic potential, $U = 1/2 (1 - r/\sigma)^2$

Returns a dxNxN matrix.

neighbors (*tol=1e-08*)

For a set of particles at xs,ys with diameters diameters, finds the distance vector matrix (d x N x N) and the adjacency matrix.

Assumes box size 1, returns (adjacency matrix, diffs)

plot_contacts (*ax=None, tol=0, reshape=True, **kw*)

Designed for use with plot_disks, this will plot a line between neighboring particles.

plot_disks (*ax=None, color=None, alpha=0.4, reshape=True*)

Plot the packing as a set of disks.

Color can be None (uses the standard sets), 'diameter' (colors by diameter), or a list of colors.

'reshape' means set axis scaled, etc.

scene (*pack, cmap=None, rot=0, camera_height=0.7, camera_dist=1.5, angle=None, light_strength=1.1, orthographic=False, pad=None, floatercolor=(0.6, 0.6, 0.6), bgcolor=[1, 1, 1]*)

Render a 3D scene.

Requires vapory package, which requires the povray binary.

Parameters **cmap** : a colormap

Returns **scene** : vapory.Scene, which can be rendered using its .render() method.

size_indices (*tol=1e-08*)

Returns [idx of sigma1, idx of sigma2, ...]

tess ()

Get a tess.Container instance of this.

Requires tess.

Indices and tables

- `genindex`
- `modindex`
- `search`

S

spack, [12](#)

B

`backbone()` (`spack.Packing` method), [12](#)

C

`cages()` (`spack.Packing` method), [12](#)

`contacts()` (`spack.Packing` method), [13](#)

D

`dist()` (`spack.Packing` method), [13](#)

`dist_tree()` (`spack.Packing` method), [13](#)

`DM()` (`spack.Packing` method), [12](#)

`DM_freqs()` (`spack.Packing` method), [12](#)

F

`forces()` (`spack.Packing` method), [13](#)

N

`neighbors()` (`spack.Packing` method), [13](#)

P

`Packing` (class in `spack`), [12](#)

`plot_contacts()` (`spack.Packing` method), [13](#)

`plot_disks()` (`spack.Packing` method), [13](#)

S

`scene()` (`spack.Packing` method), [13](#)

`size_indices()` (`spack.Packing` method), [13](#)

`spack` (module), [12](#)

T

`tess()` (`spack.Packing` method), [13](#)