Power Spectra

Though SWIFT includes functionality for calculating power spectra, this tool runs on-the-fly, and as such after a run has completed you may wish to create a number of more non-standard power spectra.

These tools are available as part of the `swiftsimio.visualisation.power_spectrum` package. Making a power spectrum consists of two major steps: depositing the particles on grid(s), and then binning their fourier transform to get the one-dimensional power.

Depositing on a Grid

Depositing your particles on a grid is performed using `swiftsimio.visualisation.power_spectrum.render_to_deposit()`. This function performs a nearest-grid-point (NGP) of all particles in the provided particle dataset. For example:

```
from swiftsimio import load
from swiftsimio.visualisation.power_spectrum import render_to_deposit

data = load("cosmo_volume_example.hdf5")

gas_mass_deposit = render_to_deposit(
    data.gas,
    resolution=512,
    project="masses",
    parallel=True,
)
```

The specific field being depositied can be controlled with the `project` keyword argument. The `resolution` argument gives the one-dimensional resolution of the 3D grid, so in this case you would recieve a `512x512x512` grid. Note that the

`gas_mass_deposit` is a `swiftsimio.cosmo_array`, and as such includes cosmological and unit information that is used later in the process.

Generating a Power Spectrum

Once you have your grid deposited, you can easily generate a power spectrum using the `swiftsimio.visualisation.power_spectrum.deposition_to_power_spectrum()` function. For example, using the above deposit:

```
from swiftsimio.visualisation.power_spectrum import deposition_to_power_spectrum

wavenumbers, power_spectrum, _ = deposition_to_power_spectrum(
    deposition=gas_mass_deposit,
    box_size=data.metadata.box_size,
)
```

This power spectrum can then be plotted. Units are included on both the wavenumbers and the power spectrum. Cross-spectra are also supported through the `cross_deposition` keyword, but by default this generates the auto power.

More Complex Scenarios

In a realistic simualted power spectrum, you will need to perform 'folding' to achieve a viable dynamic range within an achievable memory footprint. Consider, for instance, a 1 Gpc simulation volume with a 1 kpc resolution limit. In that case, you would need a deposition grid with 10^{18} cells, amounting to a 4 EB (yes, exabyte!) memory footprint. That is, of course, not realistic!

As in a power spectrum we are only interested in the periodicity of the system, we can fold it back in on itself during the rendering process. The position of the particle in the box is set to be:

$$x_i' := \left(x_i \% \frac{L}{2^n}\right) \frac{2^n}{L}$$

where L is the box-size and n is some integer greater than or equal to zero. This allows you to probe modes in the reduced box-length $L/2^n$ with the same fixed resolution deposition buffer.

The `folding` parameter is available for both `render_to_deposit` and `deposition_to_power_spectrum`, but it may be easier to use the utility functions provided for automatically stitching together the folded spectra. The function `swiftsimio.visualsation.power_spectrum.folded_depositions_to_power_spectrum()` allows you to do this easily:

```
from swiftsimio.visualisation.power_spectrum import
folded_depositions_to_power_spectrum
import unyt

folded_depositions = {}

for fold in [x * 2 for x in range(5)]:
    folded_depositions[fold] = render_to_deposit(
        data.gas,
        resolution=512,
        project="masses",
        parallel=True,
        folding=2.0 ** fold,
    )

bins, centers, power_spectrum, foldings = folded_depositions_to_power_spectrum(
    depositions=folded_depositions,
    box_size=data.metadata.box_size,
    number_of_wavenumber_bins=128,
    wavenumber_range=[1e-2 / unyt.Mpc, 1e2 / unyt.Mpc],
    log_wavenumber_bins=True,
)
```

Depositions are automatically faded between using the cube-root of the number of grid points included in the bin. For two overlapping foldings,

$$P(k) = \frac{N(k)_i^{1/3} P(k)_i + N(k)_j^{1/3} P(k)_j}{N(k)_i^{1/3} + N(k)_j^{1/3}}$$

which can be visualised using the `folding_tracker` return value of the function.