Installation

Required dependencies

The well-established dependencies are:

- Python 2.7 or 3.4
- scipy, numpy : the foundations for scientific Python
- mpi4py : MPI for Python
- h5py : support for HDF5 files in Python

with a suite of additional tools:

- astropy : a community Python library for astronomy
- pfft-python : a Python binding of pfft, a massively parallel Fast Fourier Transform implementation with pencil domains
- pmesh : a particle mesh framework in Python
- kdcount : pair-counting and Friends-of-Friends clustering with KD-Tree
- bigfile : a reproducible, massively parallel IO library for hierarchical data
- MP-sort : massively parallel sorting
- sharedmem : in-node parallelism with fork and copy-on-write

Optional dependencies

For reading data using pandas

- pandas and pytables are required to use the `Pandas` DataSource, which uses *pandas* for fast parsing of plain text files, as well as the *pandas* subset of HDF5

For creating data using a Halo Occupation Distribution

- halotools is required to use the `Zheng07HOD` DataSource, which provides a general framework for populating halos with galaxies using Halo Occupation Distribution modeling

For generating simulated data from linear power spectra

- classylss, a python binding of the Boltzmann code CLASS, is required to use the `ZeldovichSim` DataSource, which computes a simulated data catalog using the Zel'dovich approximation (from a linear power spectrum computed with CLASS)

Instructions

The software is designed to be installed with the `pip` utility like a regular Python package. The first step on all supported platforms is to checkout the source code via:

```
git clone http://github.com/bccp/nbodykit
cd nbodykit
```

Linux

The steps listed below are intended for a commodity Linux-based cluster (e.g., a Rocks cluster) or a Linux-based workstation / laptop.

To install the main nbodykit package, as well as the external dependencies listed above, into the default Python installation directory:

```
pip install -r requirements.txt
pip install -U --force --no-deps .
```

A different installation directory can be specified via the `--user` or `--root <dir>` options of the `pip install` command.

Mac OS X

The `autotools` software is needed on Mac:

```
sudo port install autoconf automake libtool
```

Using recent versions of MacPorts, we also need to tell `mpicc` to use `gcc` rather than the default `clang` compiler, which doesn't compile `fftw` correctly due to the lack of `openmp` support. Additionally, the `LDSHARED` environment variable must be explicitly set.

In bash, the installation command is:

```
export OMPI_CC=gcc
export LDSHARED="mpicc -bundle -undefined dynamic_lookup -DOMPI_IMPORTS"; pip install -r
requirements.txt
pip install -U --force --no-deps .
```

Development Mode

nbodykit can be installed with the development mode (`-e`) of pip

```
pip install -r requirements.txt -e .
```

In addition to the dependency packages, the 'development' installation of nbodykit may require a forced update from time to time:

```
pip install -U --force --no-deps -e .
```

It is sometimes required to manually remove the `nbodykit` directory in `site-packages`, if the above command does not appear to update the installation as expected.

Final Notes

The dependencies of nbodykit are not fully stable, thus we recommend updating the external dependencies occassionally via the `-U` option of `pip install`. Also, the `--force` option ensures that the current sourced version is installed:

```
pip install -U -r requirements.txt
pip install -U --force --no-deps .
```

To confirm that nbodykit is working, we can type, in a interactive Python session:

```
import nbodykit
print(nbodykit)

import kdcount
print(kdcount)

import pmesh
print(pmesh)
```

Or try the scripts in the bin directory:

```
cd bin/
mpirun -n 4 python nbkit.py -h
```

To run the test suite after installing nbodykit, install py.test and pytest-pipeline and run `py.test nbodykit` from the base directory of the source code:

```
pip install pytest pytest-pipeline
pytest nbodykit
```

Using nbodykit on NERSC

In this section, we give instructions for using the latest stable build of nbodykit on NERSC machines (Edison and Cori), which is provided ready-to-use and is recommended for first-time users. For more advanced users, we also provide instructions for performing active development of the source code on NERSC.

When using nbodykit on NERSC, we need to ensure that the Python environment is set up to work efficiently on the computing nodes. The default Python start-up time scales badly with the number of processes, so we employ the python-mpi-bcast tool to ensure fast and reliable start-up times when using nbodykit. This tool can be accessed on both the Cori and Edison machines.

General Usage

We maintain a daily build of the latest stable version of nbodykit on NERSC systems that works with the `2.7-anaconda` Python module and uses the *python-mpi-bcast* helper tool for fast startup of Python. Please see this tutorial for further details about using *python-mpi-bcast* to launch Python applications on NERSC.

In addition to up-to-date builds of nbodykit, we provide a tool (`/usr/common/contrib/bccp/nbodykit/activate.sh`) designed to be used in job scripts to automatically load nbodykit and ensure a fast startup time using *python-mpi-bcast*.

Below is an example job script that prints the help message of the `FFTPower` algorithm:

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -o nbkit-example
#SBATCH -n 16

# You can also allocate the nodes with salloc
#
# salloc -n 16
#
# and type the commands in the shell obtained from salloc

module unload python
module load python/2.7-anaconda

source /usr/common/contrib/bccp/nbodykit/activate.sh

# regular nbodykit command lines
# replace nbkit.py with srun-nbkit

srun-nbkit -n 16 FFTPower --help

# You can also do this in an interactive shell
# e.g.
```

Active development

If you would like to use your own development version of nbodykit directly on NERSC, more installation work is required, although we also provide tools to simplify this process.

We can divide the addititional work into 3 separate steps:

1. When building nbodykit on a NERSC machine, we need to ensure the Python environment is set up to work efficiently on the computing nodes.

If darshan or altd are loaded by default, be sure to unload them before installing, as they tend to interfere with Python:

```
module unload darshan
module unload altd
```

and preferentially, use GNU compilers from PrgEnv-gnu

```
module unload PrgEnv-intel
module unload PrgEnv-cray
module load PrgEnv-gnu
```

then load the Anaconda Python distribution,

```
module load python/2.7-anaconda
```

For convenience, these lines can be included in the shell profile configuration file on NERSC (i.e., `~/.bash_profile.ext`).

2. For easy loading of nbodykit on the compute nodes, we provide tools to create separate bundles (tarballs) of the nbodykit source code and dependencies. This can be performed using the `build.sh` script in the `nbodykit/nersc` directory in the source code tree.

```
cd nbodykit/nersc;

# build the dependencies into a bundle
# this creates the file `$NERSC_HOST/nbodykit-dep.tar.gz`
bash build.sh deps

# build the source code into a separate bundle
# this creates the file `$NERSC_HOST/nbodykit.tar.gz`
bash build.sh source
```

When the source code changes or the dependencies need to be updated, simply repeat the relevant `build.sh` command given above to regenerate the bundle.

3. Finally, in the job script, we must explicitly activate *python-mpi-bcast* and load the nbodykit bundles.

```
#!/bin/bash
#SBATCH -p debug
#SBATCH -o nbkit-dev-example
#SBATCH -n 16

# load anaconda
module unload python
module load python/2.7-anaconda

# activate python-mpi-bcast
source /usr/common/contrib/bccp/python-mpi-bcast/nersc/activate.sh

# go to the nbodykit source directory
cd /path/to/nbodykit

# bcast the nbodykit tarballs
bcast nersc/$NERSC_HOST/nbodykit-dep.tar.gz nersc/$NERSC_HOST/nbodykit.tar.gz

# run the main nbodykit executable
srun -n 16 python-mpi /dev/shm/local/bin/nbkit.py FFTPower --help
```