PySpace Documentation

Release 0.0.2

Aditya Bhosale, Rahul Govind, Raj Krishnan

April 30, 2016

Contents

1 Overview **3**
1.1 PySpace: a python-based framework for galactic simulations 3

2 Installation and getting started **5**
2.1 Installation 5

3 The framework and library **7**
3.1 The PySpace Framework 7

4 Tutorials **11**
4.1 Tutorial 11

5 Reference Documentation **15**
5.1 PySpace Reference Documentation 15

6 Indices and tables **17**

Python Module Index **19**

PySpace is an open-source framework for galactic simulations. It is implemented in Cython and the computational part is implemented in pure C++.

PySpace provides parallel support through OpenMP and GPU support through CUDA.

CHAPTER 1

Overview

1.1 PySpace: a python-based framework for galactic simulations

PySpace is an open-source framework for galactic simulations. It is implemented in Cython and the computational part is implemented in pure C++.

PySpace provides parallel support through OpenMP and GPU support through CUDA.

Here's a video of galaxy collision simulation using PySpace.

1.1.1 Features

- A python interface for high performance C++ implementation of N-body simulation algorithms.
- PySpace has a numpy friendly API which makes it easier to use.
- Parallel support using OpenMP.
- GPU support using CUDA
- Dumps vtk output which allows users to take advantage of tools like ParaView, MayaVi, etc. for visualization.

1.1.2 Credits

PySpace has been developed as a part of ME766 (High Performance Scientific Computing) project at IIT Bombay.

Lead developers:

- Aditya Bhosale
- Rahul Govind
- Raj Krishnan

CHAPTER 2

Installation and getting started

2.1 Installation

2.1.1 Dependencies

- Numpy
- PyEVTK (`pip install pyevtk`)
- gcc compiler
- OpenMP (optional)
- ParaView / MayaVi or any other vtk rendering tool (optional)

2.1.2 Linux and OSX

To install the latest stable version, run:

```
$ pip install pyspace
```

To install development version, clone this repository by:

```
$ git clone https://github.com/adityapb/pyspace.git
```

To install, run:

```
$ python setup.py install
```

To install without OpenMP, set `USE_OPENMP` environment variable to 0 and then install:

```
$ export USE_OPENMP=0
$ python setup.py install
```

To install without GPU support, set `USE_CUDA` environment variable to 0 and then install:

```
$ export USE_CUDA=0
$ python setup.py install
```

Troubleshooting

If you run into any issues regarding installation or otherwise, please report here.

Some common issues are addressed below

CUDA not found

Make sure if the CUDA toolkit is installed. If you still get this message after installation, follow the instructions given below.

Add CUDA it to `PATH` environmental variable and try again

Or, set `CUDAHOME` environmental variable to path of the CUDA installation by:

```
$ export CUDAHOME=/usr/local/cuda
```

Image not found

If your code compiles and you get this error at runtime, make sure you have a CUDA compatible device installed.

If you don't, install without GPU support (see Installation)

PySpace doesn't support Windows currently

2.1.3 Running the tests

For running the tests you will need to install `nose`, install using:

```
$ pip install nose
```

To run the tests, from project's root directory run:

```
$ make test
```

2.1.4 Running the benchmarks

For running benchmarks you will need to install `pandas`, install using:

```
$ pip install pandas
```

To run the benchmarks, cd to benchmarks directory and run:

```
$ python run_benchmarks.py
```

CHAPTER 3

The framework and library

3.1 The PySpace Framework

This document is an introduction to the design of PySpace. This provides high level details on the functionality of PySpace. This should allow the user to use the module and extend it effectively.

To understand the framework, we will work through a general N-body problem.

Here we will be using the `BruteForceSimulator`, however the framework is essentially the same for any `Simulator`.

3.1.1 N-body problem

Consider a problem of n bodies with mass m_i for the i^{th} planet.

Equations

$$\vec{a_i} = \sum_{\substack{j=1 \\ j \neq i}}^{n} G \frac{m_j}{r_{ij}^3} (\vec{r_j} - \vec{r_i}) \tag{3.1}$$

Handling collisions

From the above equation, it is clear that force will become infinite when $r_{ij} = 0$

To solve this problem we use a gravity softening method proposed by Aarseth in 1963. Thus we change our force equation to

$$\vec{a_i} = \sum_{\substack{j=1 \\ j \neq i}}^{n} G \frac{m_j}{(\epsilon^2 + r_{ij}^2)^{3/2}} (\vec{r_j} - \vec{r_i}) \tag{3.2}$$

where ϵ is the softening factor. By default, $\epsilon = 0$

Numerical Integration

`BruteForceSimulator` uses leap frog integrator for updating velocity and positions of planets.

$$x_{i+1} = x_i + v_i \Delta t + \frac{1}{2} a_i \Delta t^2 \tag{3.3}$$

$$v_{i+1} = v_i + \frac{1}{2}(a_i + a_{i+1}) \Delta t \tag{3.4}$$

Understanding the framework

PySpace uses `pyspace.planet.PlanetArray` for storing planets.

`pyspace.planet.PlanetArray` stores numpy arrays for $x, y, z, v_x, v_y, v_z, a_x, a_y, a_z, m, r$.

```
cdef public ndarray x
cdef public ndarray y
cdef public ndarray z
cdef public ndarray v_x
cdef public ndarray v_y
cdef public ndarray v_z
cdef public ndarray a_x
cdef public ndarray a_y
cdef public ndarray a_z
cdef public ndarray m
cdef public ndarray r
```

Note: Currently `r` doesn't have any use per se. However, we plan to use it for better collision handling in the future.

`pyspace.simulator.Simulator` stores pointers to these numpy arrays and then passes these raw pointers to the C++ function, `brute_force_update` which then updates the pointers using the above numerical integration scheme.

3.1.2 Algorithms

A number of techniques for solving the N-body problem are available. Following are currently implemented in PySpace.

Brute Force

This is implemented in `pyspace.simulator.BruteForceSimulator` which uses the $O(n^2)$ brute force algorithm for calculating forces in a planet.

Barnes Hut

This is implemented in `pyspace.simulator.BarnesSimulator` which uses the $O(nlogn)$ barnes hut algorithm for calculating forces in a planet.

For details see this wikipedia article.

3.1.3 Visualization

PySpace dumps a vtk output of the simulations. These can then be visualized using tools such as Paraview, MayaVi, etc.

The vtk dump is controlled by the `dump_output` flag in `Simulator::simulate`. The vtk dump by default only dumps v_x, v_y, v_z ie. velocities of the planets. For dumping custom data, use `set_data` in `pyspace.simulator.Simulator`.

CHAPTER 4

Tutorials

4.1 Tutorial

4.1.1 Imports

```
import numpy

# PySpace PlanetArray imports
from pyspace.planet import PlanetArray

# PySpace Simulator imports
from pyspace.simulator import BruteForceSimulator
```

Note: These are common to all simulations with brute force. For dumping vtk output with custom data, use the `set_data` function in `pyspace.simulator.Simulator`.

For using `BarnesSimulator`, use

```
# Import BarnesSimulator instead of BruteForceSimulator
from pyspace.simulator import BarnesSimulator
```

4.1.2 Setting up PlanetArray

`PlanetArray` is the container for all objects in a simulation. The following example sets up a PlanetArray with planets arranged in a cube

```
x, y, z = numpy.mgrid[0:500:5j, 0:500:5j, 0:500:5j]
x = x.ravel(); y = y.ravel(); z = z.ravel()

pa = PlanetArray(x=x, y=y, z=z)
```

4.1.3 Setting up the Simulation

`Simulator` is the base of all computations in PySpace. The following snippet shows how to set up the simulation.

```
G = 1
dt = 0.1

sim = BruteForceSimulator(pa, G, dt, sim_name = "square_grid")
```

For using `BarnesSimulator`, you need to define θ (`thetha`) (see framework).

```
theta = 0.1
sim = BarnesSimulator(pa, G, dt, theta, sim_name = "square_grid")
```

Note: As θ is increased, speed of simulation will increase, but accuracy will decrease.

Running without GPU support

By default PySpace uses CUDA version of `BruteForceSimulator`. To use serial or OpenMP version you need to reinstall PySpace with the `USE_CUDA` environmental variable set to 0.

Gravity softening

For using gravity softening (see framework), you can set the value of ϵ by doing the following.

```
epsilon = 1
G = 1
dt = 0.1

sim = BruteForceSimulator(pa, G = G, dt = dt, epsilon = epsilon, sim_name = "square_grid")
```

Note: Use ϵ only when planets are colliding.

4.1.4 Running the simulator

`BruteForceSimulator::simulate` simulates the system for a given time. Following is the syntax for `simulate`.

```
# Simulate for 1000 secs, ie. 1000/0.1 = 10e4 time steps
sim.simulate(total_time = 1000, dump_output = True)
```

Note: `dump_output = True` essentially dumps a vtk output for every timestep.

4.1.5 Dumping custom vtk output

`pyspace.simulator.BruteForceSimulator` by default only dumps v_x, v_y, v_z ie. the velocity in the generated vtk output. To dump additional data, you need to use `pyspace.simulator.Simulator.set_data` function.

Using this method for the above problem, you can write,

```
# Do all imports and set up the PlanetArray as done above

# Set up the simulator
sim = BruteForceSimulator(pa, G, dt, sim_name = "square_grid")

# Use set_data() to tell the simulator what to dump
# For this problem, lets say you only need a_x, a_y and a_z
sim.set_data(a_x = 'a_x', a_y = 'a_y', a_z = 'a_z')

sim.simulate(total_time = total_time, dump_output = True)
```

Note: Arguments of `set_data` is a property name, attribute name pair. For the above example, we could have called `set_data` as `set_data(acc_x = 'a_x', ...)` and it would still work.

CHAPTER 5

Reference Documentation

5.1 PySpace Reference Documentation

Auto-generated documentation by Sphinx

5.1.1 Module planet

class pyspace.planet.**PlanetArray**
PlanetArray for storing planets

com()
Return centre of mass of the system of planets

concatenate()
Concatenates 'other' PlanetArray to self

dist()
Distance between planet 'i' and planet 'j'

Parameters:

i, j: int Indices of planets whose distance is sought.

get_number_of_planets()
Returns number of planets in the PlanetArray

Parameters:

None

Returns:

int: Number of planets in PlanetArray

kinetic_energy()
Returns total kinetic energy of PlanetArray

kinetic_energy_planet()
Returns kinetic energy of planet 'j'

potential_energy()
Returns total potential energy of PlanetArray

potential_energy_planet()
Returns potential energy of planet 'i'

Parameters:

G: double Universal Gravitational constant

i: int Index of the particle whose potential energy is sought

total_energy()
Returns total energy of PlanetArray

total_energy_planet()
Returns total energy of planet 'i'

5.1.2 Module simulator

class pyspace.simulator.**BarnesSimulator**
Simulator using Barnes Hut algorithm

class pyspace.simulator.**BruteForceSimulator**
Simulator using Brute Force algorithm

class pyspace.simulator.**Simulator**
Base class for all simulators

reset()
Deletes all existing simulations of the same name

set_data()
Sets what data has to be dumped to the vtk output

Parameters:

**kwargs: {property name = attribute name}

simulate()
Calculates position and velocity of all particles after time 'total_time'

Parameters:

total_time: double Total time for simulation

dump_output: bool Set True if vtk dump is required

5.1.3 Module utils

pyspace.utils.**dump_vtk**(*pa*, *filename*, *base='.'*, ***data*)
Dumps vtk output to file 'base/filename'

pyspace.utils.**get_planet_array**(**args*, ***kwargs*)
Returns a PlanetArray

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

p

pyspace.planet, 15
pyspace.simulator, 16
pyspace.utils, 16

B

BarnesSimulator (class in pyspace.simulator), 16
BruteForceSimulator (class in pyspace.simulator), 16

C

com() (pyspace.planet.PlanetArray method), 15
concatenate() (pyspace.planet.PlanetArray method), 15

D

dist() (pyspace.planet.PlanetArray method), 15
dump_vtk() (in module pyspace.utils), 16

G

get_number_of_planets() (pyspace.planet.PlanetArray method), 15
get_planet_array() (in module pyspace.utils), 16

K

kinetic_energy() (pyspace.planet.PlanetArray method), 15
kinetic_energy_planet() (pyspace.planet.PlanetArray method), 15

P

PlanetArray (class in pyspace.planet), 15
potential_energy() (pyspace.planet.PlanetArray method), 15
potential_energy_planet() (pyspace.planet.PlanetArray method), 15
pyspace.planet (module), 15
pyspace.simulator (module), 16
pyspace.utils (module), 16

R

reset() (pyspace.simulator.Simulator method), 16

S

set_data() (pyspace.simulator.Simulator method), 16
simulate() (pyspace.simulator.Simulator method), 16
Simulator (class in pyspace.simulator), 16

T

total_energy() (pyspace.planet.PlanetArray method), 16
total_energy_planet() (pyspace.planet.PlanetArray method), 16