MHPC 2015 - Python assignment Documentation

Release 1.0

David Grellscheid

September 18, 2015

CHAPTER

ONE

PHYSICS BACKGROUND

1.1 Solving the time evolution of a system of point masses

The state of any mechanical system at a given time t_0 can be fully described by a state vector of all positions and velocities involved.

$$\vec{s}(t_0) = \Big(x_1(t_0), \ldots, x_i(t_0), \ldots, v_1(t_0), \ldots, v_i(t_0), \ldots \Big)$$

To estimate the state of the system a short time later, at $t_1 = t_0 + \Delta t$, we can make use of this definition of a derivative:

$$\frac{\mathrm{d}\vec{s}}{\mathrm{d}t}(t_0) \approx \frac{\vec{s}(t_1) - \vec{s}(t_0)}{\Delta t}$$

In a mechanical system the derivative on the left is a **known** quantity, made of velocities and accelerations, determined by the physics involved:

$$\frac{\mathrm{d}\vec{s}}{\mathrm{d}t}(t_0) = \Big(v_1(t_0), \ldots, v_i(t_0), \ldots, a_1(t_0), \ldots, a_i(t_0), \ldots \Big)$$

Therefore, we can rearrange for the unknown state $\vec{s}(t_1)$:

$$\vec{s}(t_1) \approx \vec{s}(t_0) + \frac{\mathrm{d}\vec{s}}{\mathrm{d}t}(t_0)\, \Delta t$$

This can now be repeated for every following time step. The pre-prepared code

```
steppers.euler( state, d_dt, dt )
```

implements this behaviour. Given a starting state vector `state` at t_0, a python **function** `d_dt` which implements the derivative and a timestep `dt`, it will return the new state vector at t_1. The function `d_dt` here takes a `state` vector and returns the vector of the time derivative of the state.

1.2 Gravitational physics reminder

The acceleration of a point mass m under gravitational attraction from another mass M is

$$\vec{a} = \frac{1}{m}\,\vec{F} = \frac{1}{m}\,\frac{G\,mM}{|\vec{r}|^3}\vec{r} = \frac{G\,M}{|\vec{r}|^3}\vec{r}$$

where $\vec{r}$ is the vector from m to M.

CHAPTER

TWO

TASKS

2.1 Task 1 — Single-body orbit

Simulate and plot the trajectory of a point mass which is gravitationally attracted by a fixed central mass. Choose interesting initial conditions to show a circular, elliptical and hyperbolic orbit.

The plot does not have to be animated.

2.2 Task 2 — Flexible initial conditions

Initial conditions should be read interactively from the user, with sensible default values pre-set.

2.3 Task 3 — Two-body system

Allow the position of the central body to vary, too. Plot several interesting binary star systems.

2.4 Task 4 — Three-body system

Introduce a planet into your binary star system.

2.5 Task 5 — User-defined many-body system

Make your program flexible enough to deal with an arbitrary number of bodies, selectable by the end user through a configuration file.