PyGravity

http://github.com/russloewe/PyGravity

Russell Loewe

7 May 2015

This is the supporting documentation for *PyGravity*

Contents

Chapter 1

Foreward

1.1 About

PyGravity is a gravity simulating tool written in Python. PyGravity's purpose is to be able to take an arbitrary of objects from small particles to large bodies such as planets or stars and calculate the progression of their movements in an arbitrary number of dimensions. The main focus on this project is precision of calculations and not on speed. This program will quickly hit very long run times with even small data sets.

PyGravity is not for real time calculations. The aim is to take an input file that contains the position, mass, and velocities of objects in a system and output the evolution of the system to a high degree of precision. The outputted data can be either directly graphed with graphing software such as Matplotlib, or saved to a CSV file where it can later either be graphed, animated or plugged back into PyGravity for further calculations.

1.2 Features

Arbitrary Dimension PyGravity defaults to 3 spatial dimensions, but it can easily be configured to work in any number of spatial dimensions. The underlying code treats all vectors as having arbitrary dimension. As long as the dimension of the vectors match any calculation can be performed, from norming vectors, finding distances, or calculating and applying forces. There is currently a dimensions attribute on the base package. Usage is outlined in the next chapter, but it's existence is strictly for the CSV parser when loading object from a file. This is even just a stand in until a smarter parser is created.

Precision The precision is carried out by Python's Decimal module. The precision is defaulted to 200 but can be adjusted for either greater accuracy or greater speed.

Vector Physics There are a number of vector physics employed in PyGravity that are accessible independently. Some examples are getting directional and unit vectors between two locations, finding the force of gravity in a system of many objects, finding the escape velocity for each individual object in a system of many.

1.3 Maintainer

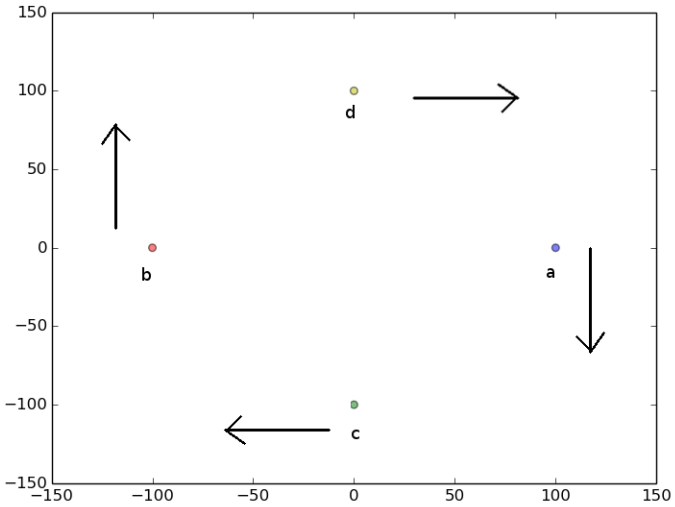
Latest code is on github at *http://github.com/russloewe/PyGravity* . Issues can be directly emailed to the maintainer at russloewe@gmail.com .

Chapter 2

Examples

2.1 4 Body Circular Orbit

Starting the examples with one of my favourites, let's simulated a ring of planets. In this example each planet, or asteroid, will have equal mass, start off in a ring, with initial velocities pointing pointing in a circle. This is illustrated in the below graphic. The black arrows indicate the initial velocity of each object. The positions and velocities are defined in example_data.csv .

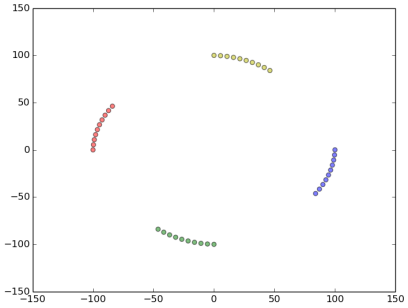


exmple_data

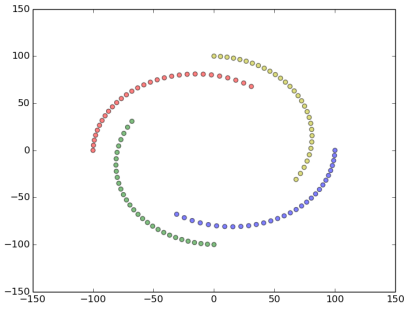
```
a:100:0:0:0:-0.09:0:1.55e10
b:-100:0:0:0:0.09:0:1.55e10
c:0:-100:0:-0.09:0:0:1.55e10
d:0:100:0:0.09:0:0:1.55e10
```

code

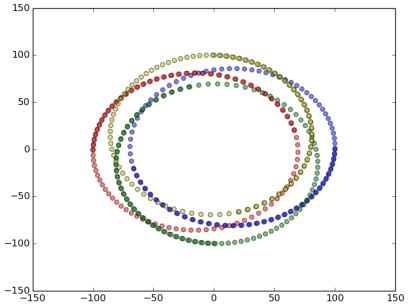
```
import numpy as np
import matplotlib.pyplot as plt
import PyGravity

# Created base instance for PyGravity
base = PyGravity.PyGravity()

# Set spatial dimension, not required, default is 3
base.dimension = 3

# Load data from data file
base.read_file('example_data.csv')

# Set time step to 60 seconds (1 minute)
base.Physics.timestep = 60

# lower precision to 100, from default of 200
base.Physics.prec = 100

# create empty lists for each objects x and y position
# ignore z since all z values are zero
ax = []
ay = []
bx = []
by = []
cx = []
cy = []
dx = []
dy = []

# save each objects location into the above lists for each timestep, or
    in this case for
# each minute
for i in range(10):
    ax.append(base.Physics.objects[0].P[0])
    ay.append(base.Physics.objects[0].P[1])
    bx.append(base.Physics.objects[1].P[0])
    by.append(base.Physics.objects[1].P[1])
    cx.append(base.Physics.objects[2].P[0])
    cy.append(base.Physics.objects[2].P[1])
    dx.append(base.Physics.objects[3].P[0])
    dy.append(base.Physics.objects[3].P[1])

   # calculate the positions of each object for the next time interval
       (minute)
    base.Physics.step_all()

# add each data list to the graph
plt.scatter(ax, ay, s=10, c='b', alpha=0.5)
plt.scatter(bx, by, s=10, c='r', alpha=0.5)
plt.scatter(cx, cy, s=10, c='g', alpha=0.5)
plt.scatter(dx, dy, s=10, c='y', alpha=0.5)
plt.show()
```



10 min



20 min



90 min