1 N-body problems

A well known n-body problem involves calculation of gravity forces between N object, where the force can be between object i and j is determined by an equation

$$\boldsymbol{f}_{ij} = G\,\frac{m_i m_j}{\left\|\boldsymbol{r}_{ij}\right\|^2} \cdot \frac{\boldsymbol{r}_{ij}}{\left\|\boldsymbol{r}_{ij}\right\|}$$

,where the f_{ij} is the gravity on object i caused by object j. However, there is a problem when the divisor is close to zero. To fix the error, many studies suggested to and a small constant into the divisor to get smoothing effect [Aarseth 2003, Dyer and Ip 1993].

$$\boldsymbol{a}_i \approx G \sum_{1 \le j \le N} \frac{m_j \boldsymbol{r}_{ij}}{\left(\left\|r_{ij}\right\|^2 + \varepsilon^2\right)^{1.5}}$$

A python code for calculating all forces in the system can be simple as below.

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


N = 100
s = np.random.rand(N,2)*2-1
r = np.zeros((2,N,N))*2-1
v = np.zeros((N,2))

m=1.0
dt=0.01
G=0.1
epsilon=1e-3
while True :
    for i in xrange (N):
        for j in xrange (N):
            r[:,i,j] = s[j,:] - s[i,:]

    c = (r[0,:,:]**2 + r[1,:,:]**2 + epsilon**2)**(-1.5)
    a = r*c
    np.fill_diagonal(a[0,:,:], 0)
    np.fill_diagonal(a[1,:,:], 0)
    ai=np.transpose( G * m * np.sum(a,axis=2) )
    v = v + ai*dt

    s = s + v*dt

    plt.plot(s[:,0],s[:,1],'.r')
    plt.hold(False)
    plt.axis([-20,20,-20,20])
    plt.draw()
```

2 IPython.Parallel

In order to accelerate the computation we can use GPU as in

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html.

However, we are going to give the problem to a cluster to solve using IPython.Parallel. Here is the simplest code to distribute the task in the cluster.

```
# -*- coding: utf-8 -*-
"""
Created on Wed Apr 22 11:17:44 2015

@author: Wasit
"""

import time
import numpy as np
from six.moves import zip

from bokeh.plotting import *

def transition(sx,sy,vx,vy,m):
    dt=1
    G=0.1
    epsilon=1e0
    # http://http.developer.nvidia.com/GPUGems3/gpugems3_ch31.html

    for i in xrange(len(sx)):
        ax_i=0.0;
        ay_i=0.0;
        for j in xrange(len(sx)):
            if i!=j:
                rx_ij=sx[j]-sx[i]
                ry_ij=sy[j]-sy[i]
                f=(rx_ij**2 + ry_ij**2 + epsilon*2)**-1.5
                ax_i=ax_i + G*m[i]*rx_ij*f
                ay_i=ay_i + G*m[i]*ry_ij*f

        vx[i] = vx[i] + ax_i*dt
        vy[i]  =vy[i] + ay_i*dt

    sx = sx + vx*dt
    sy = sy + vy*dt
    return sx,sy,vx,vy

if __name__ == "__main__":
    N = 40

    x = np.random.random(size=N) * 100
    y = np.random.random(size=N) * 100
    vx = np.zeros(shape=N)
    vy = np.zeros(shape=N)
    m=np.random.random(size=N)

    colors = ["#%02x%02x%02x" % (r, g, 150) for r, g in
zip(np.floor(50+2*x), np.floor(30+2*y))]
```

```
TOOLS="resize,crosshair,pan,wheel_zoom,box_zoom,reset,tap,previewsave,
box_select,poly_select,lasso_select"

    #output_file("color_scatter.html", title="color_scatter.py
example")
    #output_server("scatter_animate", url='http://10.200.30.55:5006/')
    output_server("scatter_animate")

    p = figure(tools=TOOLS)
    p.scatter(x,y, radius=m, fill_color=colors, fill_alpha=0.6,
line_color=None,name="particles")

    show(p)  # open a browser
    #get renderer from object by tag name
    renderer = p.select(dict(name="particles"))
    #data from object
    ds = renderer[0].data_source
    while True:
        for i in xrange(len(x)):
            #update the value of the object
            #call transition function
            (x,y,vx,vy) = transition(x,y,vx,vy,m)
            ds.data["x"] = x
            ds.data["y"] = y
            cursession().store_objects(ds)
            time.sleep(0.01)
```

This may be a little bit confused because of the visualization. Here we use module Bokeh to plot the scatter plot on web page. So, after executing the code user can open web browser at url='http://10.200.30.55:5006/'.