Using Google App Engine to Find Closed Orbits

Daniel Woelfel

Spring 2011

1 Introduction

We would like to find closed-form orbits in the N-body problem. We will use a simplified version of the general problem to demonstrate methods for finding these orbits. The problem of finding closed orbits is interesting because most orbits are not closed form.

This paper is especially interesting because we create a webapp using Google App Engine to do all of the computing. App Engine's scalability and ability to natively run Python code allow our methods to be feasible. The HTTP protocol also makes it easy to pass data from one task to another and to display our results.

2 Description of the problem

Our simplified problem is from Project 4, question 3.

Consider three-bodies, $\mathbf{r}, \mathbf{r_1}, \mathbf{r_2}$, in which two massive bodies revolve around each other in circular motion with

$$\mathbf{r_1}(t) = (0.5\cos(t), 0.5\sin(t)), \qquad \mathbf{r_2}(t) = -\mathbf{r_1}(t)$$

and the third body experiences time-dependent acceleration given by

$$\mathbf{a}(\mathbf{r}, t) = -\frac{1}{2}\left(\frac{\mathbf{d}_1(t)}{d_1^3(t)} + \frac{\mathbf{d}_2(t)}{d_2^3(t)}\right),$$

with

$$\mathbf{d}_1(t) = \mathbf{r} - \mathbf{r}_1(t), \qquad \mathbf{d}_2(t) = \mathbf{r} - \mathbf{r}_2(t)$$

Once we give the body denoted by $\mathbf{r}$ a starting position and velocity, the entire system is determined. There are uncountably many initial values, so we must find a way to reduce the problem space.

To reduce the problem space, we will restrict our search to the positive numbers. Due to symmetry, any closed orbit occurring with purely negative starting values will have a corresponding orbit with all positive values. Solutions with both negative and positive values will be lost.

Our next method to reduce the problem space is to restrict $\mathbf{r}$'s initial values to below 1. This is reasonable because far-away, fast-moving objects have a lower chance of being caught in the potential field of our two massive bodies.

Since we cannot test every possible combination, we will only test a handful of discrete values. In our case, we will partition $[0, 1]$ into seven parts, $[0.0, 0.1667, 0.3333, 0.5, 0.6667, 0.8333]$. This makes for $7^4 = 2401$ distinct initial positions.

We also notice that diverging orbits have a unique property. After a given amount of time, their orbit will be a straight line and their acceleration will approach zero. We can check for this in our computation of the orbits and stop computing once we discover that the change in velocity is approaching zero.

We will use a second-order symplectic algorithm to compute the position of $\mathbf{r}$.

For each time-step, dt, we will determine the next position, $\mathbf{r} = (x, y)$, and velocity, $d\mathbf{r}/dt = (vx, vy)$, with the following step algorithm,

$$x_{n+1/2} = x_n + vx_n * 0.5 * dt$$

$$y_{n+1/2} = y_n + vx_n * 0.5 * dt$$

$$t = t + 0.5 * dt$$

$$vx_{n+1} = vx_n + ax_{n+1/2} * dt$$

$$vy_{n+1} = vy_n + ay_{n+1/2} * dt$$

$$x_{n+1} = x_{n+1/2} + vx_{n+1} * 0.5 * dt$$

$$y_{n+1} = y_{n+1/2} + vx_{n+1} * 0.5 * dt$$

3 Computational Methodology

All of our computations are done with Google App Engine. Data is passed from one task to another using the HTTP protocol. Visualizations are created using Google's Visualizations API, which creates the visualizations using client-side Javascript on a web-browser.

Google App Engine is a platform for developing and hosting web applications[1]. It supports python and Java programming languages. We use Python in for our application because the author is familiar with Python web frameworks.

Our web app is available at `http://closedorbit.appspot.com`.

The computations occur in 7 stages. These could be compressed to fewer stages, but we explain them all to help the reader understand the source code. They are:

1. `MainPage` - Takes an integer value which it uses to partition the range, $[0, 1]$. In our case, we entered "7". It then creates a tuple containing the initial values, which it passes off to `TestStartingPosition`.

   The important thing to note here is that each call to `TestStartingPosition` starts a new task in a task queue. The tasks are no longer connected to `MainPage`, which exits as soon as it has created all of the tasks. All 2401 tasks are enqueued, and execute at a rate of about 40 per second.

2. `TestStartingPosition` - Takes a tuple containing the initial values and runs the step algorithm up to 100,000 times, with $dt = 0.001$. On each run, the step algorithm tests to see if the velocity in each of the x and y directions has changed by less than 0.0001. If

[1] `http://en.wikipedia.org/wiki/Google_App_Engine`

so, it adds 1 to a tracker. It also checks to see if the acceleration is in the same direction as the motion. If so, it adds 1 to a tracker. If the tracker goes above 10, the process exits. This indicates a divergent orbit.

If the tracker never goes above 10, then the starting position is added to the database, to be studied further. In our case, about 200 starting positions were added to the database.

3. `SendCalcToQueues` - This function creates a new task queue entry for every starting position in the datastore. It sends the item to `CalcPoints`. This process is very similar to `MainPage`.

4. `CalcPoints` - This function calculates every point in the trajectory of $\mathbf{r}$ and stores every 10th point in the database. The points will be used in future processes for graphing.

5. `ReturnGraphData` - This function takes an item in the datastore and uses the points stored by `CalcPoints` to return graph data for use in Google's Visualization API.

6. `GraphsPage` - This function returns a webpage with a graph of one of the starting positions in the database. The initial position is displayed at the top as a tuple in the form (x, y, vx, vy).

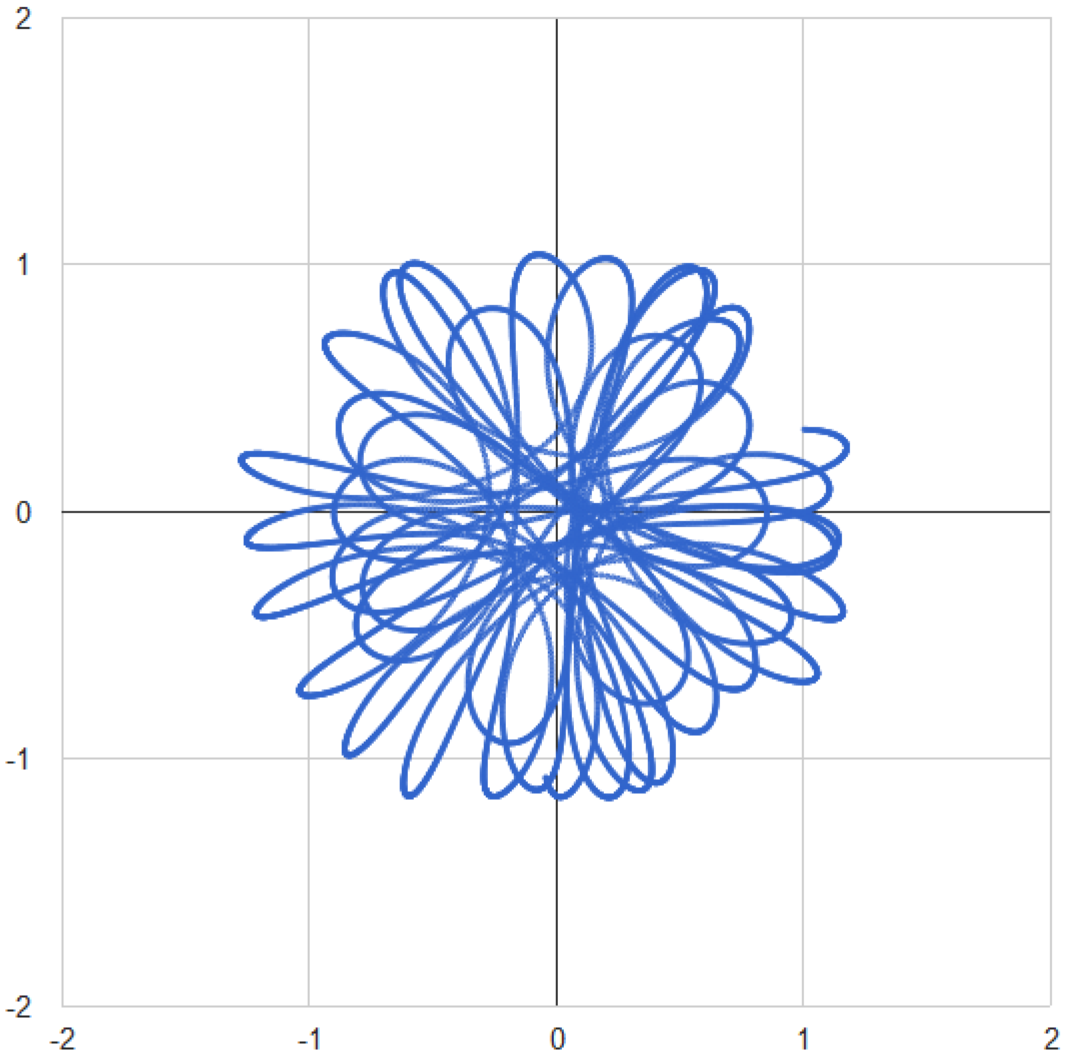
   The page gives the user the option to mark the graph as either good or bad. Good in our case means suitable for further study. Bad means obviously divergent.

   Due to time constraints, we have stopped here. More work can be done to further refine the starting positions to find closed-orbits that are closer to perfection.

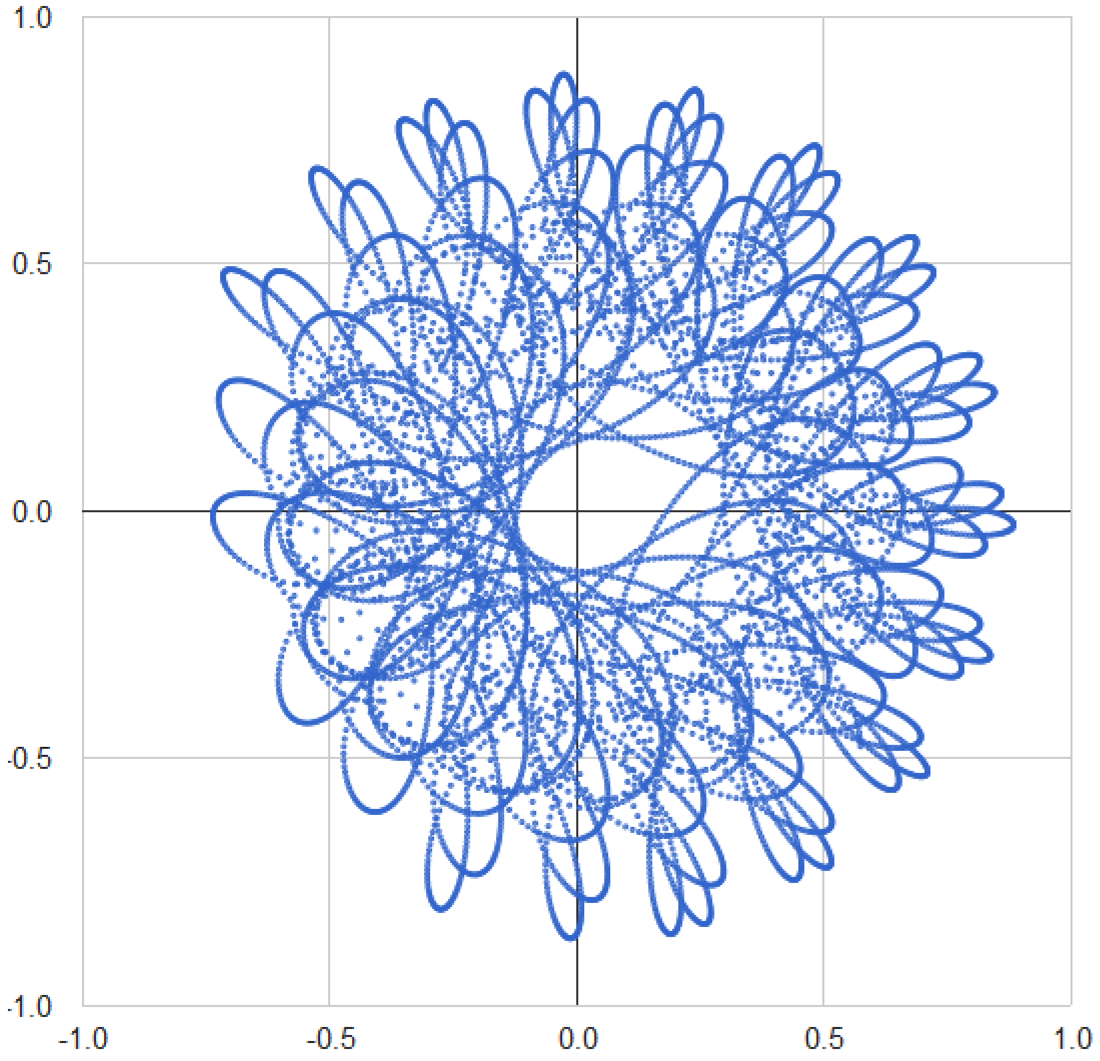
4 Results

Here we include pictures of graphs produced by our webapp and we give the starting positions:

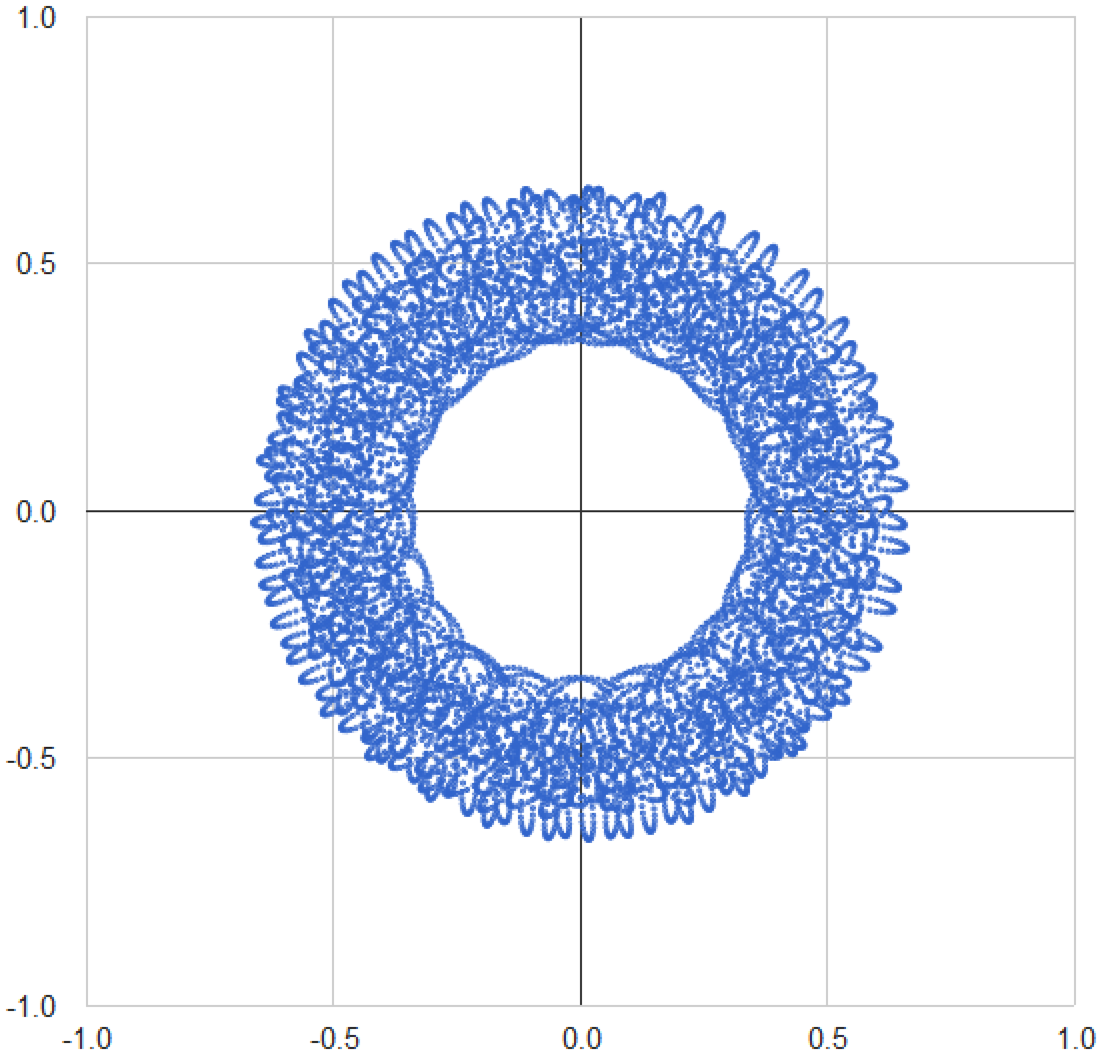
$$x = 1.0, y = 0.3333, vx = 0.6667, vy = 0.0$$



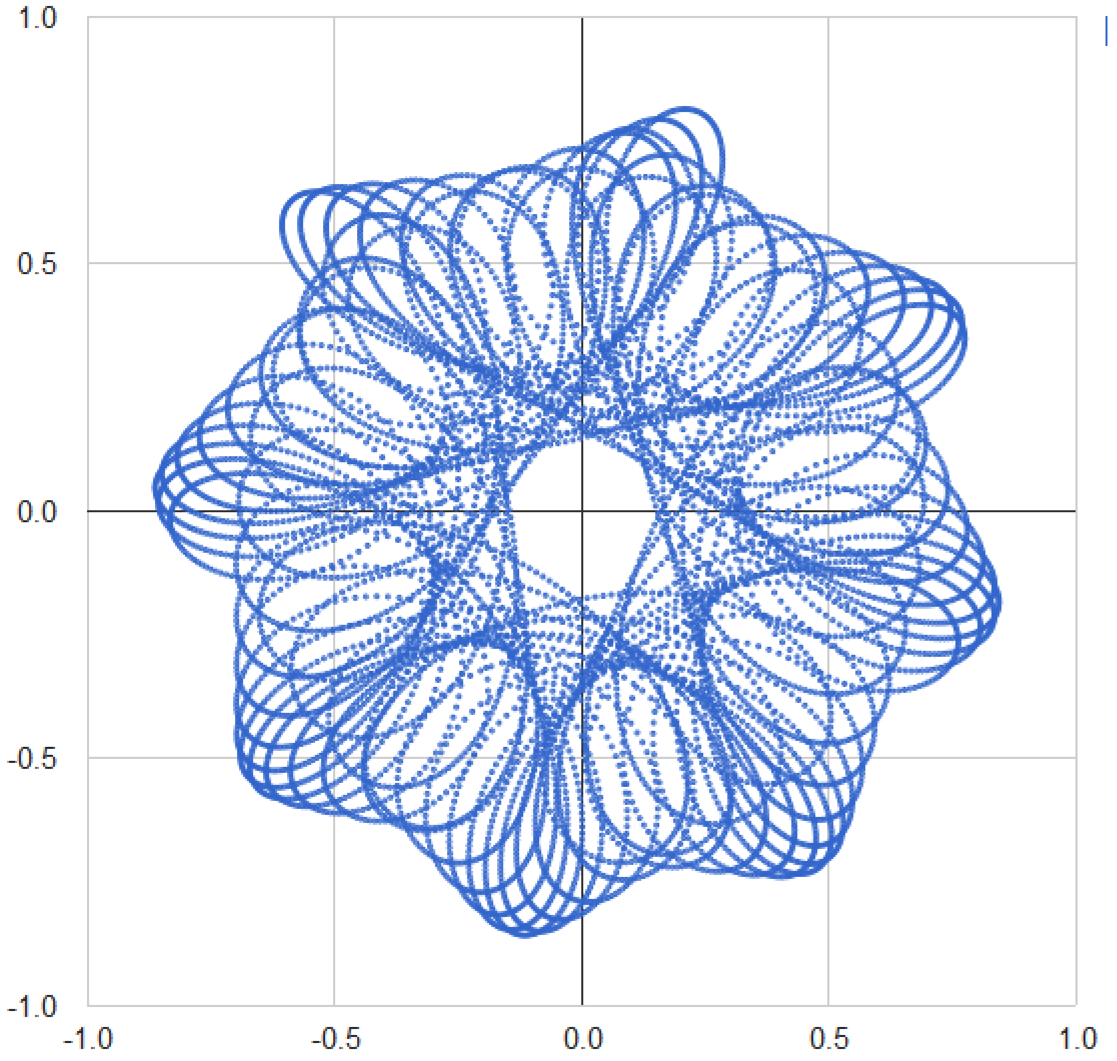
$x = 0.3333, y = 0.3333, vx = 0.6667, vy = 0.5$



$x = 0.5, y = 0.1667, vx = 0.6667, vy = 0.3333$



$x = 0.5, y = 0.3333, vx = 1.0, vy = 0.8333$



As you can see, not all of these graphs are perfect closed-orbits. However, they are close and there are probably perfect closed orbits at nearby starting positions.

To rate orbits as good or bad, you can go to `http://closedorbit.appspot.com/graphs/page`. To see all of the orbits, you can go to `http://closedorbit.appspot.com/all/orbits`.

We have not yet rated all of the 233 starting positions in the database.

5 Conclusion

The above images are representative of most of the starting positions in the database. From this, we conclude that our methods of narrowing down the possibilities are valid. In addition, the starting position given in Project 4, problem 3 was not rejected by our process. This also indicates that our method is valid.

Further work should focus on discovering the starting positions that produce perfect closed orbits, using the starting positions that we found as a jumping off points.

We hope this project has demonstrated the usefulness of Google App Engine in exploratory problem solving. We were able to do lots of computations in parallel, which enabled us to push through problems quicker. It also has built-in methods for storing data and passing data from one task to another, even to separate applications like the visualization API.

Unlike most desktop programs, applications on the internet have standardized protocols for talking to one another. In our case, it was easy to convert our points into a JSON object that could be read by Google's Visualization API.

All of the code used here will be available on Github at `https://github.com/dwwoelfel/Closed-Orbit`.

6 Bibliography

As this was exploratory programming using methods which we discussed in class and demonstrated on the homework, the bibliography is bare. However, I will list the documentation for the resources used to create the app:

- App Engine Documentation - `http://code.google.com/appengine/docs/python/overview.html`
- Google Visualization API Documentation - `http://code.google.com/apis/chart/interactive/docs/index.html`
- Python Documentation - `http://docs.python.org/tutorial/`

7 Program Listing

I list here only the main parts of the program. The full listing can be found on GitHub at `https://github.com/dwwoelfel/Closed-Orbit`.

closedorbit.py, contains the logic that runs the app

```
from google.appengine.api import mail
from google.appengine.api import users
from google.appengine.api import taskqueue
from google.appengine.ext import webapp
from google.appengine.ext import db
from google.appengine.ext.webapp import template
from google.appengine.ext.webapp.util import run_wsgi_app
from google.appengine.ext.db import Key
from google.appengine.api import urlfetch
from google.appengine.api import images
import png
import struct
import cgi
import logging
import os
import sys
import wsgiref.handlers
import uuid
import urlparse
import urllib2
import random
from BeautifulSoup import ICantBelieveItsBeautifulSoup as ICBS
from datetime import datetime, date, time, timedelta
from dateutil.relativedelta import *
from django.utils import simplejson as json
os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
from google.appengine.dist import use_library
use_library('django', '1.2')
import itertools
import operator
import hashlib
import math
import gviz_api

class Promising(db.Model):
    values = db.ListProperty(float)
    str_values = db.StringProperty()
    point_blob = db.BlobProperty()
       # stores every 10th value -- 'x,y,x,y,...'
       # watch out for the trailing comma
    good = db.BooleanProperty()
    checked = db.BooleanProperty()

class MainPage(webapp.RequestHandler):
    def get(self):

        def par(my_range, i):
            return 1.0*i/(my_range-1)

        my_range = int(self.request.get('my_range'))

        for i in range(my_range):
            for j in range(my_range):
                for k in range(my_range):
                    for l in range(my_range):
                        x = par(my_range, i)
                        y = par(my_range, j)
                        vx = par(my_range, k)
                        vy = par(my_range, l)
                        taskqueue.add(queue_name='optimize',
                                      url='/test/starting/position',
                                      params={'x':x, 'y':y, 'vx':vx, 'vy':vy},
                                      method='GET')
        directory = os.path.dirname(__file__)
        path = os.path.join(directory, '../templates/front_page.html')
        #self.response.out.write(template.render(path, template_values))
        self.response.out.write('added them all!')

class TestStartingPosition(webapp.RequestHandler):
    def get(self):
        def step((x,y,vx,vy,t,dt,tracker)):
            vxtemp = vx
            vytemp = vy

            x = x + vx*0.5*dt
            y = y + vy*0.5*dt

            t = t + 0.5*dt

            r1x = 0.5*math.cos(t)
            r1y = 0.5*math.sin(t)
            r2x = -0.5*math.cos(t)
            r2y = -0.5*math.sin(t)
            d13 = pow(math.sqrt(pow(x-r1x,2)+pow(y-r1y,2)),3)
            d23 = pow(math.sqrt(pow(x-r2x,2)+pow(y-r2y,2)),3)
            ax = -0.5*((x-r1x)/d13 + (x-r2x)/d23)
            ay = -0.5*((y-r1y)/d13 + (y-r2y)/d23)

            vx = vx + ax*dt
            vy = vy + ay*dt

            x = x + vx*0.5*dt
            y = y + vy*0.5*dt

            t = t + 0.5*dt

            if math.fabs(vxtemp - vx) < 0.0001 and math.fabs(vytemp - vy) < 0.001:
                tracker += 1

            if (ay != 0 and y != 0) and math.fabs(ax/ay - x/y) < 0.001:
                tracker += 1
            else:
                tracker = 0

            return (x,y,vx,vy,t,dt,tracker)
        def test(ran, next):
            for i in range(ran):
                next = step(next)
                if next[-1] > 10:
                    return (100,100)
            return next

        x = float(self.request.get('x'))
        y = float(self.request.get('y'))
        vx = float(self.request.get('vx'))
        vy = float(self.request.get('vy'))

        tryit = test(100000, step((x,y,vx,vy,0,0.001,0)))

        if math.fabs(tryit[0]) < 50 and math.fabs(tryit[1]) < 50:
            new_promise = Promising()
            new_promise.values = [x,y,vx,vy]
            new_promise.put()
            logging.info('put  [%s,%s,%s,%s]', str(x), str(y), str(vx), str(vy))

        else:
            logging.info('didn\'t put [%s,%s,%s,%s]', str(x), str(y), str(vx), str(vy))

class ConvertToString(webapp.RequestHandler):
    query = db.Query(Promising)
    results = query.fetch(1000)
    for result in results:
        L = result.values
        result.str_values = str(L[0]) + ',' + str(L[1]) + ',' + str(L[2]) + ',' + str(L[3])
        result.put()


class SendCalcToQueues(webapp.RequestHandler):
    def get(self):
        query = db.Query(Promising)
        results = query.fetch(1000)
        for result in results:
            taskqueue.add(queue_name='optimize',
                          url='/calc/points',
                          params={'str_values':result.str_values,},
                          method='GET')

class CalcPoints(webapp.RequestHandler):
    def get(self):
        # we'll only store every 5th value
        str_values = self.request.get('str_values')
        query = db.Query(Promising)
        query_filter = query.filter('str_values =', str_values)
        result = query_filter.get()
        L = result.values

        result.point_blob = ''

        next = self.step((L[0], L[1], L[2], L[3], 0, 0.001, 0))

        for i in range(100000):
            next = self.step(next)
            if i%10 == 0:
                result.point_blob += "%.4f" % next[0] + ',' "%.4f" % next[1] + ','

        result.put()

        logging.info('put all the point values')

    def step(self,(x,y,vx,vy,t,dt,tracker)):
        vxtemp = vx
        vytemp = vy

        x = x + vx*0.5*dt
        y = y + vy*0.5*dt

        t = t + 0.5*dt

        r1x = 0.5*math.cos(t)
        r1y = 0.5*math.sin(t)
        r2x = -0.5*math.cos(t)
        r2y = -0.5*math.sin(t)
        d13 = pow(math.sqrt(pow(x-r1x,2)+pow(y-r1y,2)),3)
        d23 = pow(math.sqrt(pow(x-r2x,2)+pow(y-r2y,2)),3)
        ax = -0.5*((x-r1x)/d13 + (x-r2x)/d23)
        ay = -0.5*((y-r1y)/d13 + (y-r2y)/d23)

        vx = vx + ax*dt
        vy = vy + ay*dt

        x = x + vx*0.5*dt
        y = y + vy*0.5*dt

        t = t + 0.5*dt

        return (x,y,vx,vy,t,dt,tracker)

class ReturnGraphData(webapp.RequestHandler):
    def get(self):
        str_values = self.request.get('str_value')

        description = {"x": ("number", "x"),
                       "y": ("number", "y")}

        data = []

        query = db.Query(Promising)
        filter_query = query.filter('str_values =', str_values)
        result = filter_query.get()
        point_blob = result.point_blob

        point_list = point_blob.split(',')
        del point_list[-1] # due to trailing comma
        for i in range(len(point_list)):
            if i%2 == 0:
                data.append({'x': float(point_list[i]), 'y': float(point_list[i+1])})

        data_table = gviz_api.DataTable(description)
        data_table.LoadData(data)

        logging.info(data_table.ToJSonResponse())

        self.response.out.write(data_table.ToJSonResponse())

class CreateGoodProperty(webapp.RequestHandler):
    def get(self):
        query = db.Query(Promising)
        results = query.fetch(1000)

        for result in results:
            result.good = True
            result.checked = False
            result.put()

class GraphsPage(webapp.RequestHandler):
    def get(self):
        query = db.Query(Promising)
        filter_query = query.filter('checked =', False)
        result = query.get()

        template_values = {
            'str_values': result.str_values,
            }

        directory = os.path.dirname(__file__)
        path = os.path.join(directory, '../templates/graphs_page.html')
        self.response.out.write(template.render(path, template_values))

class GoodOrBad(webapp.RequestHandler):
    def get(self):
        str_values = self.request.get('str_values')
        good = self.request.get('good_option')

        query = db.Query(Promising)
        filter_query = query.filter('str_values =', str_values)
        result = filter_query.get()

        result.good = bool(good)
        result.checked = True

        result.put()

        self.redirect('/graphs/page')
```

main.py, maps urls to functions in closedorbit.py

```
from logic import closedorbit

from google.appengine.ext import webapp
from google.appengine.ext.webapp import util
import os
os.environ['DJANGO_SETTINGS_MODULE'] = 'settings'
from google.appengine.dist import use_library
use_library('django', '1.2')
webapp.template.register_template_library('templatetags.templatetags')
def main():
application = webapp.WSGIApplication([
('/', closedorbit.MainPage),
                ('/test/starting/position', closedorbit.TestStartingPosition),
                ('/convert/to/string', closedorbit.ConvertToString),
                ('/send/calc/to/queues', closedorbit.SendCalcToQueues),
                ('/calc/points', closedorbit.CalcPoints),
                ('/return/graph/data', closedorbit.ReturnGraphData),
                ('/graphs/page', closedorbit.GraphsPage),
                ('/create/good/property', closedorbit.CreateGoodProperty),
                ('/good/or/bad', closedorbit.GoodOrBad),
  ],debug=True)
util.run_wsgi_app(application)


if __name__ == '__main__':
main()
```