On the Recently Discovered Periodic Solutions to the Three-Body Problem

Dan Girshovich - AEP4380 Final Report

May 14, 2013

1 Introduction

1.1 Background

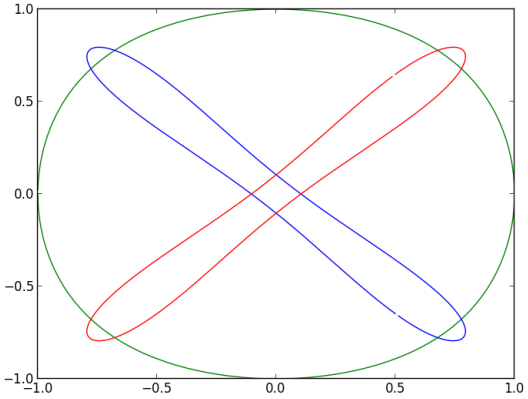
In March 2013, M. Šuvakov and V. Dmitrašinović (S & D) published a paper [1] describing their discovery of 13 previously unknown families of periodic three-body orbits. The discovery came from a numerical search within the small subspace of initial conditions where the bodies are collinear, equally spaced, and equally massive. Additionally, the net angular momentum in each of the discovered solutions is zero.

Before this discovery, there were only three families of known orbits: Lagrange-Euler, Broucke-Henon (Figure 1a), and Figure-8 (Figure 1b). It is useful to formally define a "family" of solutions before more are introduced. To do this, S & D followed the steps of Montgomery [2] by mapping their real-space periodic trajectories to closed curves in a topological object called a "shape sphere". These curves are grouped by considering their paths relative to the points on the shape sphere which correspond to collisions. This grouping scheme allows both families of orbits and the relationships between them to be defined.

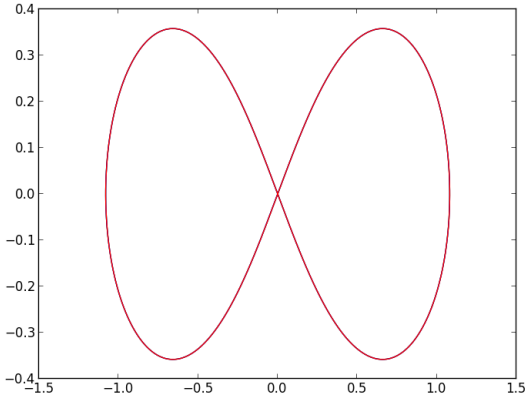
Very little has been published about the new solutions since their discovery - even the important question of the stability of the orbits is still open. Also, there may be several straightforward extensions of these new solutions which yield other, yet undiscovered, periodic solutions. For example, variations on the masses or total angular momentum have not been explored. Interestingly, the complex topological and group theoretic techniques used to classify orbits are used specifically in anticipation of nonzero angular momentum variants being discovered in the future.

1.2 Scope

The goal of this report is to replicate S & D's results and to describe both the numerical and analytical methods involved. Also, some first numerical attempts at determining the stability of the new orbits are explored.



(a) Broucke-Henon



(b) Figure-8

Figure 1: Two of the previously discovered periodic solutions.

The orbits are generated from the published initial conditions using a Runge-Kutta integration method. To verify the more complicated real-space trajectories, the simpler shape sphere contours are generated and compared to those published by S & D [3]. Also, phase-space plots and Fourier transforms are generated to show periodicity and to help reveal the scale of errors in the published initial conditions. Finally, a straightforward attempt at stability analysis is performed using numerical methods.

2 The Three-Body Problem

2.1 Overview

The classic "Three-Body Problem" is to find the trajectories of three bodies acting only under Newtonian gravitation. Here, the bodies are assumed to be point masses and the systems are all bound. Even with these assumptions, numerical techniques are the only known way of solving the general problem.

2.2 Equations of Motion

Let the position of each mass be defined as $\vec{x_i}$, where $i = 1, 2, 3$. Then, Newton's second law is

$$\frac{d^2\vec{x}_i}{dt^2} = \frac{\vec{F}_i}{m_i}.$$

Since gravity is the only force considered, we know

$$\vec{F}_i = Gm_i \sum_{j \neq i} m_j \frac{\vec{x_j} - \vec{x_i}}{|\vec{x_j} - \vec{x_i}|^3}.$$

So,

$$\frac{d^2\vec{x}_i}{dt^2} = G\sum_{j\neq i} m_j \frac{\vec{x_j} - \vec{x_i}}{|\vec{x_j} - \vec{x_i}|^3}.$$

2.3 Simplified System

Following the steps of S & D, define $G = 1$ and $m_i = 1$ for all i. This reduces the equation of motion to

$$\frac{d^2\vec{x}_i}{dt^2} = \sum_{j\neq i} \frac{\vec{x_j} - \vec{x_i}}{|\vec{x_j} - \vec{x_i}|^3}$$

and all results that follow are in this scaled unit system.

Furthermore, S & D only consider cases with zero net angular momentum. This provides a guarantee that the trajectories lie in a plane - three points are always coplanar, and gravity is a central force. Systems with net angular momentum in a direction non-orthogonal to this plane can have three dimensional trajectories. These will not be considered.

3 Numerical Solution

3.1 Overview

Efficient propagation of the orbits is prioritized in anticipation of running many propagations for analysis later on. This is primarily manifested in choosing an high-order Runge-Kutta integrator with embedded low-order solutions and, from those, an automatic step size algorithm. Since many of the trajectories exhibit long stretches of low-order motion when the bodies are far apart, with short bursts of jerky, high-order motion when they become closer together, automatic step size methods are well suited for this problem. In fact, choosing an automatic step size routine constitutes the most significant optimization performed.

3.2 Eighth-Order Dormand Prince

Section 17.2.4 of Numerical Recipes [4] recommends an eighth-order Dormand-Prince method for "production work" and the source is available online [5]. It is an improvement over the fifth-order Dormand Prince method because it produces a higher order approximation at each step (at the cost of more function evaluations). For a given region of the solution, this generally allows it to take larger steps than the fifth-order method would. Basic profiling of the propagation routine shows that, overall, the eighth order method is faster than the fifth-order method for the new orbits. This implies that the performance gain from larger step sizes outweighs the overhead of extra function evaluations.

Like most numerical integration methods, the eighth-order Dormand-Prince method requires the system of equations be written in first-order form. For the three-body system

described, these are

$$\frac{d\vec{v}_i}{dt} = \sum_{j \neq i} \frac{\vec{x_j} - \vec{x_i}}{|\vec{x_j} - \vec{x_i}|^3}$$
$$\vec{v_i} = \frac{d\vec{x}_i}{dt},$$

where initial conditions for each $\vec{x_i}$ and $\vec{v_i}$ must be provided.

3.3 Implementation

The high-level code for propagating an orbit is contained in the `propagate` function:

```
void propagate(double t) {
    rhs_grav derivs(masses, num_bodies);
    double abtol = tol, rtol = tol, hinit = 1e-4, hmin = 0.0;
    Odeint<StepperDopr853<rhs_grav> > ode(ics, 0, t, abtol, rtol, hinit, hmin,
                                          out, derivs);
    ode.integrate();
}.
```

This uses the framework discussed in Chapter 17 on Numerical Recipes [4]. Specifically, the integrator is templated on the stepper method (`StepperDopr853`) and an object describing the equations of the system (`rhs_grav`). The first of these can be found in the webnote [5], as mentioned before. The second is a straightforward implementation of the first-order equations and is available in the source code listing of Section 7.

3.4 Interpolation

For some analysis techniques, it is useful to have a fixed number of equally spaced output values (like the output of fixed step size methods). The `Odeint` framework provides a "dense" output mode for this very purpose. The `out` object referenced in the `propagate` function is initialized with an integer indicating the amount of points to output. This number is *not* related to step size in any way. Instead, it indicates the times *between* the step points which the interpolation method should find approximate values for the function. For the eighth-order Dormand-Prince method, this is done with a seventh-order interpolation method.

Examples of analysis techniques that benefit from having a known number of equally spaced output values are Fast Fourier Transforms (FFTs) and animations. As a warning, experience indicates that interpolation generally makes the eighth-order method slower than the simpler fourth-order method for these orbits. Because of this, the dense output should be switched off with the global `dense` flag when it is not needed and performance is a concern.

3.5 Results

A single period of each of the 13 new orbits are in Figures 2, 3, and 4. These trajectories were generated using the initial conditions and periods listed by S & D online [3]. Note that

each plot contains the points $(-1, 0)$, $(0, 0)$, and $(0, -1)$, which are the collinear starting positions for all of the new orbits.

To ensure smoothness of all plots, each was computed with a local error tolerance of 10^{-8} (the provided initial conditions are given to 1 part in 10^6) and a dense output of $10,000$ points. All subsequent results and analysis are done with this level of precision and resolution.

In Figure 5, the step sizes produced by the fifth-order and eighth-order Runge-Kutta methods are compared for the Goggles orbit. Note that the fifth-order routine took smaller steps than the eighth-order one, as expected. Since the eighth-order routine performs nearly twice as many computations per step than the fifth-order one (11 function evaluations vs. 6), but generally takes steps that are more than twice as large (see the vertical scales in Figure 5), it is more efficient. For sharper orbits, this efficiency difference is even more pronounced.

Except for maybe the Yarn solution in Figure 4, all of the new solutions clearly exhibit a spatial offset in two of the bodies. This unexpected property is mediated by the third body, which swings the outer bodies away from the center periodically. S & D provide no comment on this peculiar feature.

Including the missing time dimension helps further see the nature of the solutions. One way to do this is by propagating trajectories in a three-dimensional space-time, generating "braids" instead of closed curves. This technique was used by C. Moore [6] in an effort to classify n-body solutions. These new solutions, however, would generate extremely complex braids. Instead, the time dimension is included in animations of the orbits, which are referenced as included files in Section 6. The animations also clearly show the way in which the inner body swings the others outward, maintaining the spatial separation.

3.6 Discovery

Of course, these results are only recreated in the sense that the orbits are confirmed to behave as S & D claimed (the images match nicely with those online [3]). Their true accomplishment, however, is in the search technique which produced the initial conditions. In their paper [1], this is described *very* briefly as a simple gradient descent search. Basically, the method starts by finding initial velocities that produce roughly periodic trajectories. This is done by propagating a solution (likely with techniques similar to those described here) and numerically checking for near repetitions in the phase-space values. These near repetitions correspond to roughly periodic solutions, which are then filtered down to truly periodic ones via a gradient descent search. Specifically, the initial conditions for the roughly periodic solutions are progressively changed only in ways that bring the positions and velocities after one period closer to their starting values. During the search, some of the roughly periodic solutions converge to more precisely periodic solutions, while others diverge.

All simple attempts to honestly rediscover the solutions failed, mostly due to the nuance of the search. Mainly, the fact that the exact period is unknown adds a degree of freedom to the gradient descent step which is not trivial to account for. Also, to randomly find

the roughly periodic solutions with a return proximity of 10^{-1} (as claimed in [1]) would be infeasible given the computation time required to propagate even the simplest of the new solutions with the methods described here.

4 Analysis

4.1 Shape Space

4.1.1 Motivation

A significant portion of S & D's recently published work [1] discusses the categorization of solutions using closed curves on a topological object called a "shape sphere". This is in anticipation of solutions actually indicating *families* of solutions with non-zero angular momentum (following from the known family of Figure-8 solutions). Also, S & D showed that the shape sphere approach allows families of solutions to be grouped into classes in a way that nicely reflects shared symmetry properties.

4.1.2 Mathematical Formulation

S & D follow the formulation given by Montgomery [2] of a shape space. Each point in the space uniquely represents a triangle formed by three masses. Rotations and translations of the triangle count as the same triangle, but reflections do not. The simple "side-side-side" theorem for triangles states that two triangles are congruent if and only if their three sides have equal lengths. This means the shape space can be represented by a Euclidean three-space (i.e the two are homeomorphic).

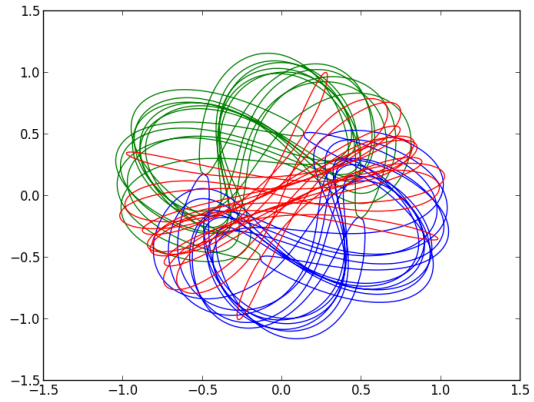
Points in shape space are formally defined as

$$< \frac{2\vec{\rho}\cdot\vec{\lambda}}{R^2}, \frac{\lambda^2-\rho^2}{R^2}, \frac{2\vec{\rho}\times\vec{\lambda}}{R^2} >,$$

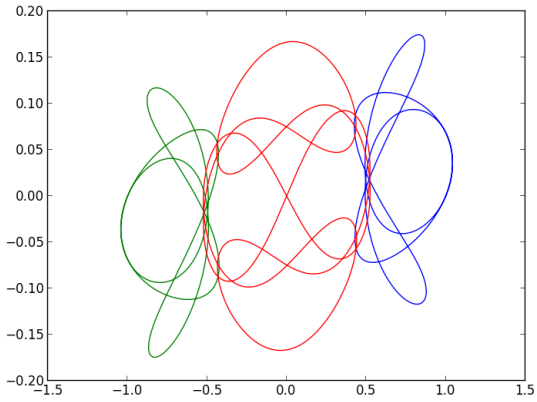
where

$$\begin{aligned}\vec{\rho} &= \frac{1}{\sqrt{2}}(\vec{x_1}-\vec{x_2})\\ \vec{\lambda} &= \frac{1}{\sqrt{6}}(\vec{x_1}+\vec{x_2}-2\vec{x_3})\\ R &= \sqrt{\rho^2+\lambda^2}.\end{aligned}$$

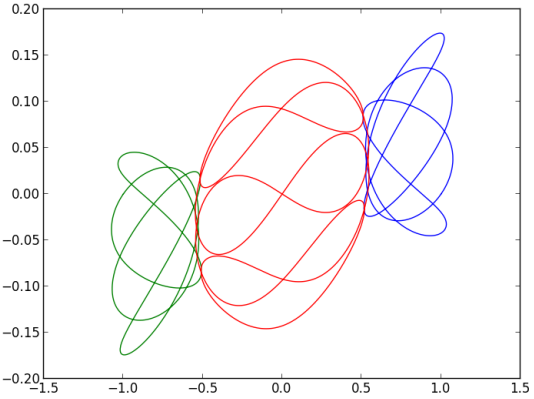
Here, $\vec{\rho}$ and $\vec{\lambda}$ are the Jacobi relative coordinate vectors and R is the hyper-radius (defines size of the system). Directions of vectors in this space are based off of the relative sizes of the distances between the three bodies. Magnitudes of these vectors can be related to the moment of inertia of the three bodies. This definition confines the contours in shape space to a solid sphere centered at the origin and with a radius normalized to one by the $\frac{1}{R^2}$ factors.



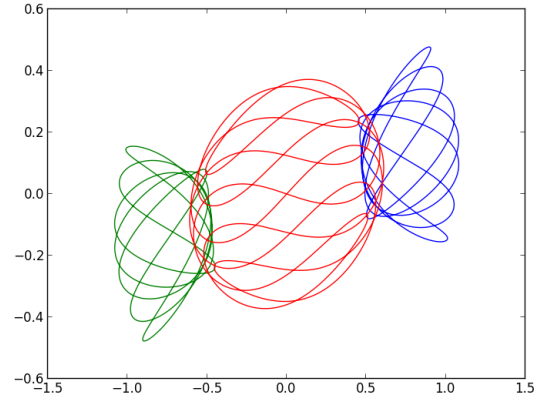
(a) Bumblebee



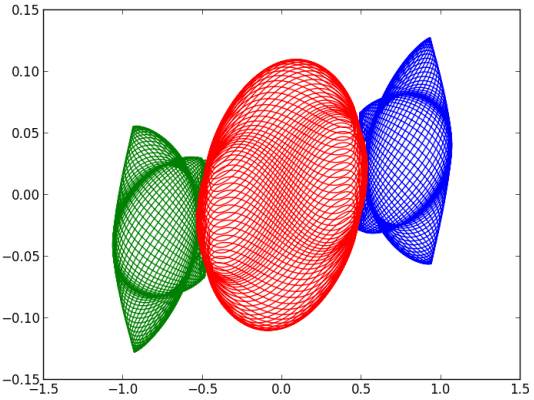
(b) Butterfly I



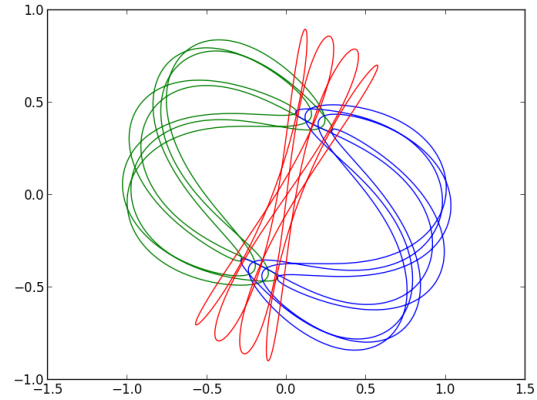
(c) Butterfly II



(d) Butterfly III

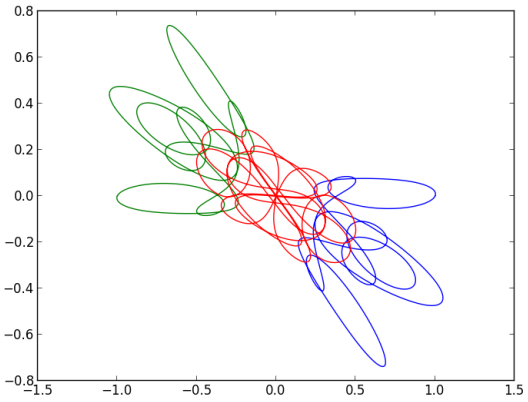


(e) Butterfly IV

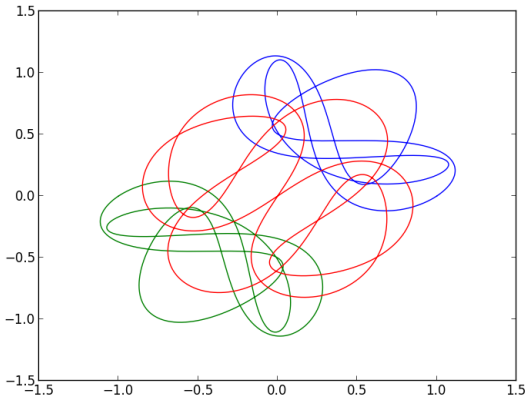


(f) Dragonfly

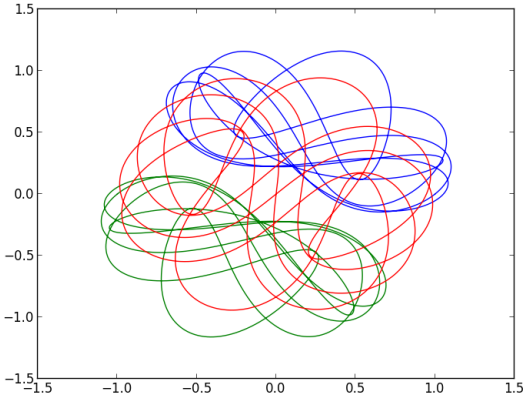
Figure 2: 1-6 of the 13 new periodic solutions.



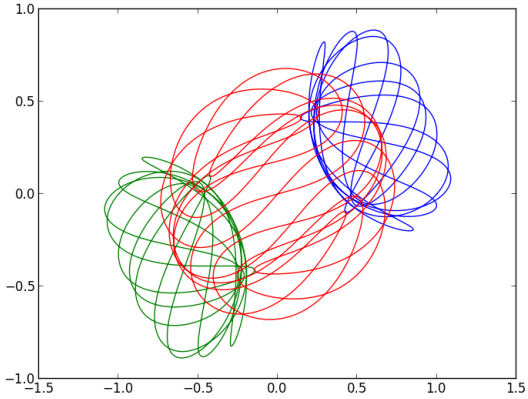
(a) Goggles



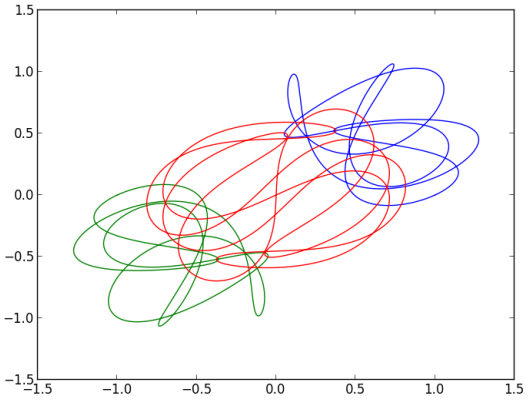
(b) Moth I



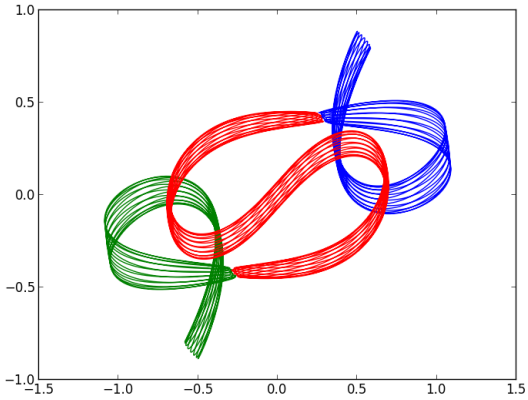
(c) Moth II



(d) Moth III



(e) YinYang I



(f) YinYang II

Figure 3: 7-12 of the 13 new periodic solutions.

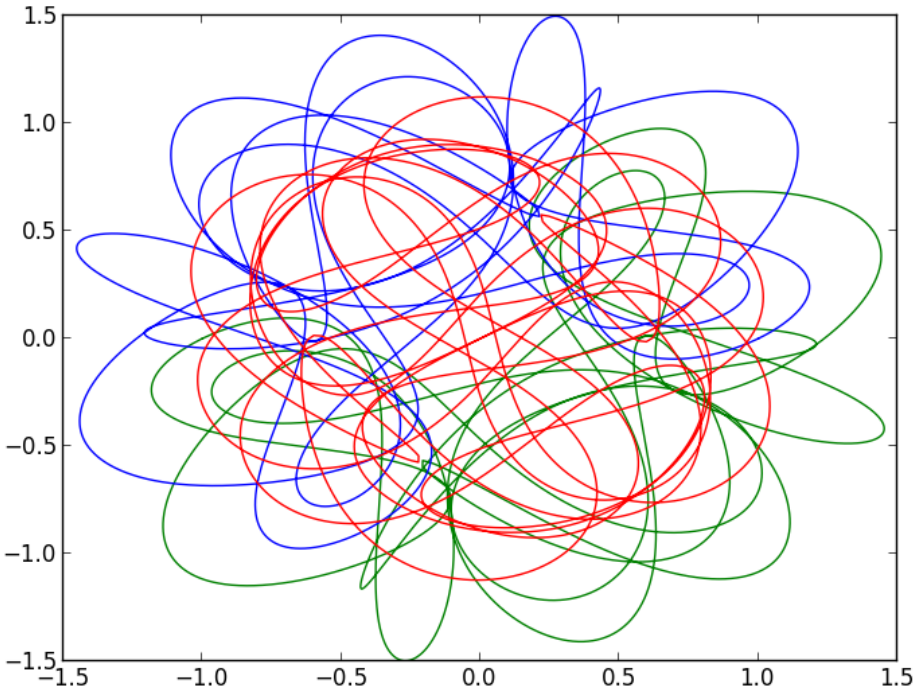
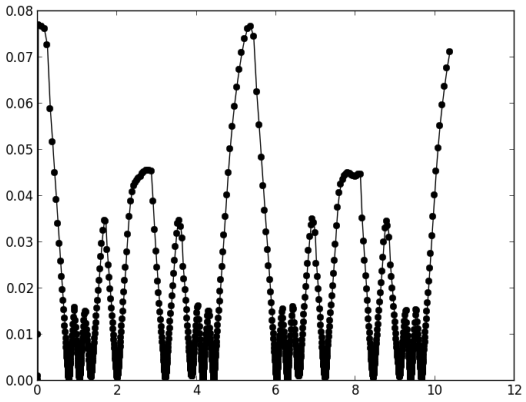
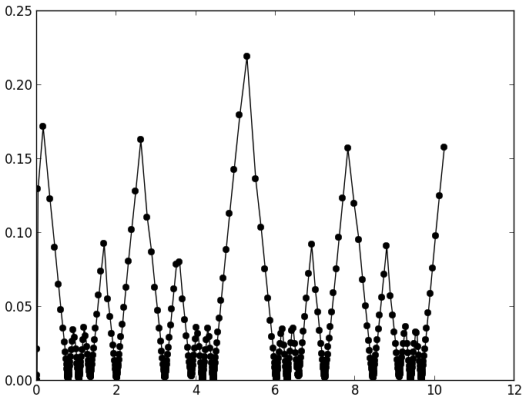


Figure 4: Yarn - the last of the 13 new periodic solutions.



(a) Fifth-order



(b) Eighth-order

Figure 5: Step sizes taken for the fifth-order and eighth-order Runge-Kutta methods when integrating the Goggles orbit (Figure 3a). All values are temporal.

Simple substitution shows that when $\vec{x_1} = \vec{x_2}$, $\vec{x_1} = \vec{x_3}$, or $\vec{x_2} = \vec{x_3}$, the point in shape space is on the equator of the sphere. These conditions correspond to the three binary collision points (the triple collision point is the origin), and are the basis for grouping the orbits. Namely, the contours in shape space corresponding to collision-less orbits cannot be smoothly deformed through a collision point after the system is stereographically projected onto a plane. This allows contours to be assigned group elements which describe their paths in the plane relative to the collision points. Orbits are defined to be in the same family if their group elements are in the same conjugacy class. Furthermore, simple exchange symmetries between conjugacy classes categorizes the *families* of orbits into three classes. This categorization neatly relates exchange symmetries in the free group elements to geometric symmetries in the shape space contours. See Table I of [1] and its description for more details about this categorization.

4.1.3 Results

The code to generate the points in shape space is in the `get_sphere_point` function, and is straightforward. Selected contours are shown in Figure 6. These figures contain enough information to reconstruct the real-space orbits shown above, as they are simply alternative representations of the same data. Because the contours are generally simpler to inspect visually than the real-space trajectories, they can be compared to the online versions [3] to provide another confirmation that the orbits were correctly propagated.

Perhaps the most obvious benefit of this representation is the exposure of symmetries. For example, the real-space representation of the Yarn orbit (Figure 4) appears to contain a reflection symmetry, but it is very hard to see because of the tangled orbit. On the other hand, the shape space representations in Figure 6 (e) and (f) clearly exhibit reflection symmetry. On the other hand, the Dragonfly orbit clearly has a reflection symmetry in real-space, but the shape space representation (Figure 6 (a) and (b)) still reveals an interesting property: each point of the contour lies on the surface of the same cylinder. S & D do not mention this peculiar feature of the Dragonfly orbit, and it is not obvious what physical feature of the orbit is responsible for this effect.

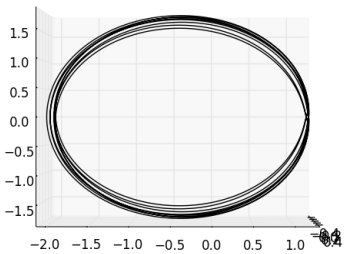
4.2 Periodicity

4.2.1 Motivation

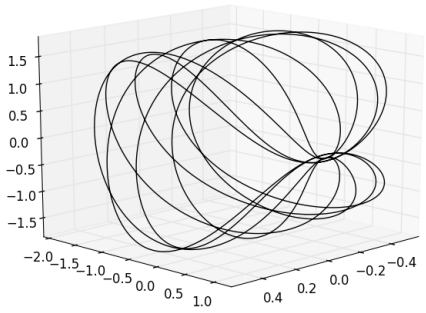
The stability of the new orbits is still unknown, and one of the simplest ways to conclude an orbit is stable is by verifying it has a fixed period over many cycles. So, before any stability analysis is done, it is worth verifying that the propagation methods provide accurate periodic solutions to start.

4.2.2 Phase Plots

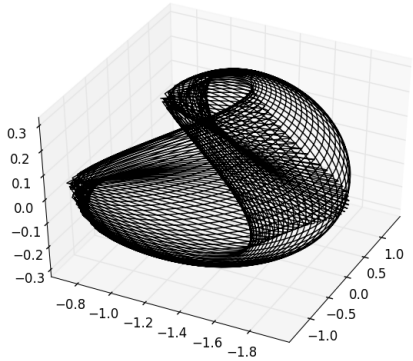
Perhaps the simplest tool for verifying periodicity is the phase plot. For each mass, the position and velocity in each dimensions are sampled at intervals of the expected period. The phase plot shows these values (e.g y_3 and v_{y3}) plotted against each other at those intervals.



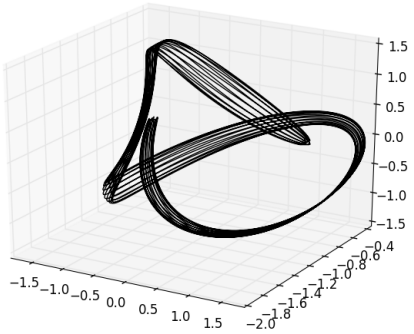
(a) Dragonfly (view 1)



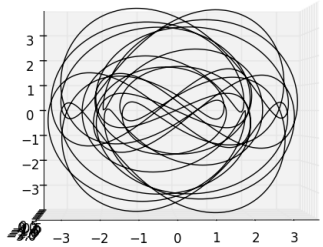
(b) Dragonfly (view 2)



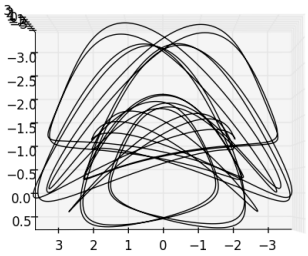
(c) Butterfly IV



(d) YinYang II



(e) Yarn (view 1)



(f) Yarn (view 2)

Figure 6: Unscaled contours in shape space corresponding to periodic real-space solutions to the three-body problem.

The resulting plots are shown in Figure 7 for select orbits propagated over ten periods.

The simpler orbits like Buttefly II appear as six tight clusters of points, since they neatly return to their initial position and velocity after their relatively short periods. The more complex orbits, however, wander from their initial phases space values over the course of the ten orbits. This is either an artifact of the accumulated integration error or insufficiently accurate initial conditions. One way to determine the culprit is rerunning the propagator with a smaller local error tolerance (like 10^{-10}). Doing so has an effect for some orbits like Butterfly IV (Figure 7e), but not others, like Broucke-Henon (Figure 7f).

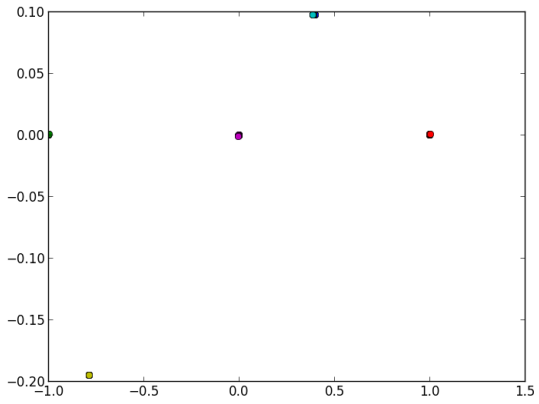
Because the Butterfly IV solution improved, the local tolerance must be carefully chosen in any analysis to ensure the global discretization error over several orbits can be ignored. There is no hope to improve orbits which are unaffected by the local tolerance change, since the initial conditions are limiting the accuracy. For the newly discovered orbits, there is currently only one source with their values ([1]), so this is as good as the phase plots can get.

4.2.3 Fourier Transforms

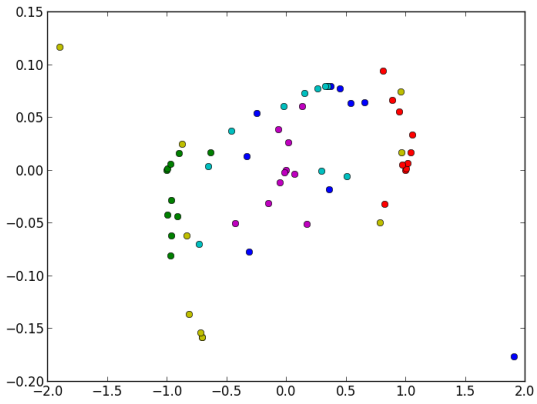
Phase plots discard nearly all of the information about the orbits by only considering the state of the system at intervals of one period. Between these intervals, the bodies interact with each other in complex ways, which have their own regular structure. In fact, most of the new orbits have motions that, in a sense, repeat at shorter times than the full period. These higher harmonics are exposed by performing a Fourier transform of a component of position (or velocity) for any of the masses. For some of the trajectories, these harmonics are also easily discerned from the real-space plots. However, this is infeasible for the more tangled orbits, even by carefully watching the animations provided.

The transform is taken using the Fast Fourier Transform (FFT) algorithm, which is described in Numerical Recipes Section 12.2 [4]. The implementation here uses the popular FFTW library [8] and is contained in the `write_fft` function. Care is taken to remove the spatial offsets (discussed in Section 3.5 above) in the component being transformed by using the `center` function. Otherwise, this offset would add zero frequency components to the transforms, which are unwanted. Transforms of select orbits are shown in Figure 7. The mass chosen corresponds to the green trajectories in Figures 2 and 3. Notice that each transform has spikes separated by 10 frequency units, indicating that any harmonic motion must occur an integral number of times over the course of one period. Anything else would contradict the fact that the system returns to the same conditions after one period.

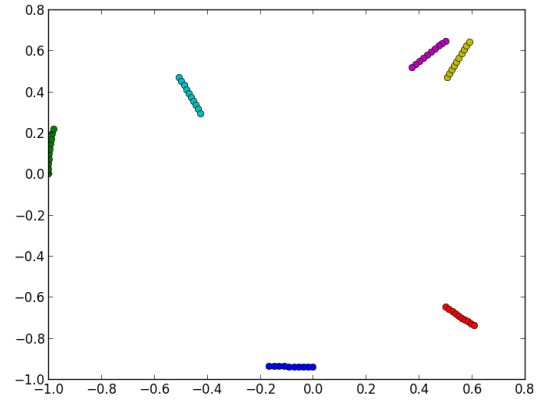
Several interesting properties of each of the orbits chosen in Figure 8 are revealed by the transforms. For example, it is clear that the ribbons of the YinYang II orbit are built with 8 tightly packed loops, since there is a dominating spike corresponding to motion 8 times as regular as the period. Similarly, the "wings" of the Bumblebee orbit evidently take 9 passes to construct.



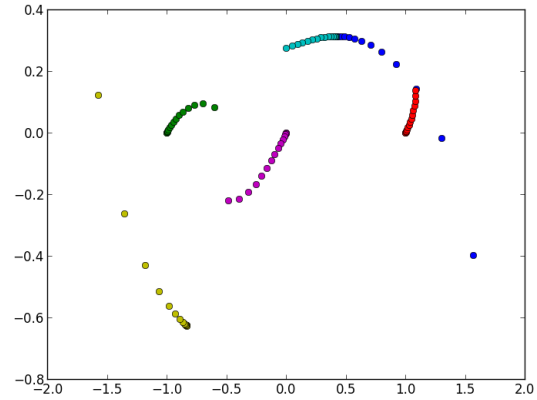
(a) Butterfly II



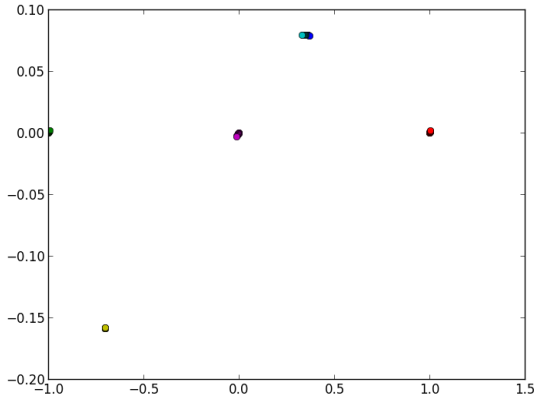
(b) Butterfly IV



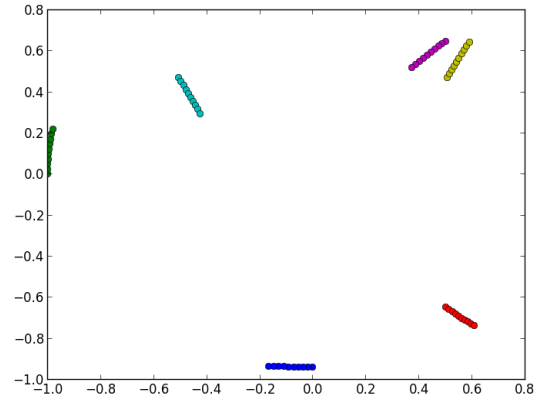
(c) Broucke-Henon



(d) YinYang II

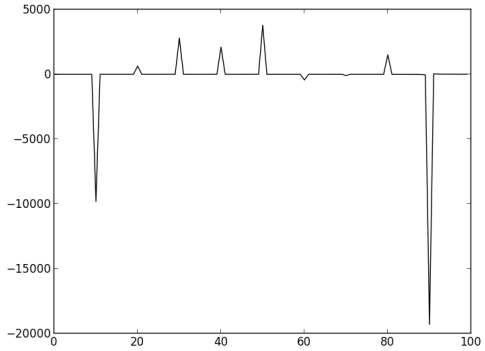


(e) Butterfly IV (smaller tolerance)

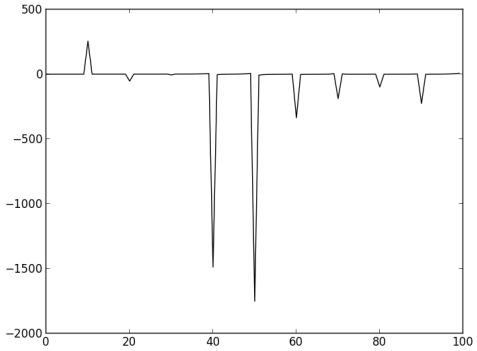


(f) Broucke-Henon (smaller tolerance)

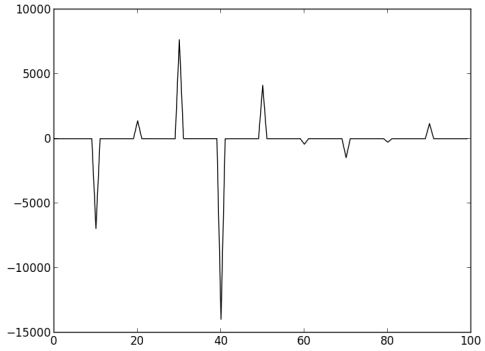
Figure 7: Phase plots for each dimension of the three masses. Pots (e) and (f) were done with local tolerance of 10^{-10}, where the rest have 10^{-8} (used in Section 3.5). YinYang II improves in a way similar to Butterfly IV when the tolerance is decreased, but is not shown here. All units are spatial.



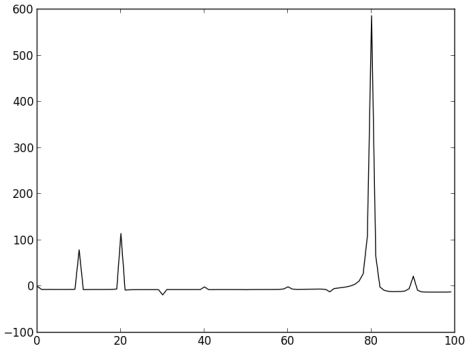
(a) Bumblebee



(b) Butterfly III



(c) YinYang I



(d) YinYang II

Figure 8: Unscaled Fourier transforms of the x component of one of the masses over 10 periods. The x axes represent frequency relative to the 10 periods (i.e. the motion of the full orbit is represented as a spike at 10, since it occurs 10 times).

4.3 Stability

4.3.1 Overview

S & D claim that it would be very unlikely for the numerical method they used to discover a completely unstable orbit. This is because they performed a gradient descent search that sought to bring each mass closer and closer to its initial conditions after one period. If an orbit is highly unstable, no nearby almost-periodic orbits are expected - small changes in initial conditions would generally send an unstable periodic orbit into a chaotic one. So, a gradient descent search is not expected to discover unstable orbits.

Still, the topic of the stability of the new orbits is worth exploring. This is mainly because three-body problems in general are still poorly understood (compared to two-body problems). Complicated and thorough stability analyses for the Figure-8 orbit have been recently done (see [7] and [2]), and the the stability properties of Euler-Lagrange orbits for equal masses has been known for a long time. However, nothing is definitively known about the stability of the 13 new orbits.

While the search technique may rule out *completely* unstable orbits, S & D are sure to mention that *marginally* unstable orbits are still possible. While the complex analytic techniques used to classify and prove things like marginal instability are out of scope for this report, it is still possible to get a first glimpse at the stable or unstable nature of the orbits with numerical techniques.

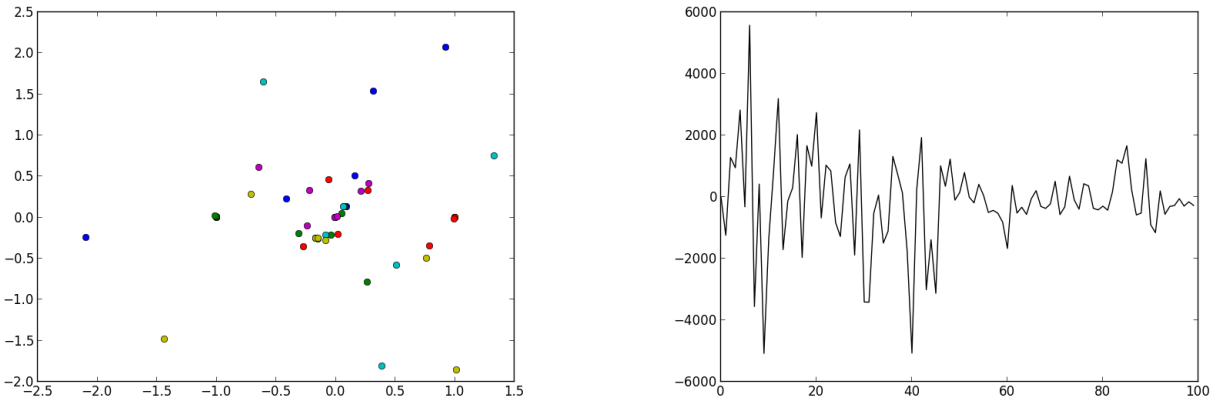
4.3.2 Finding Instability

In generating the phase plots and Fourier transforms for some orbits, the first signs of instability emerged. The phase plot and Fourier transform of the Goggles orbit are shown in Figure 9. These correspond to the trajectory shown in Figure 10c. Even by shrinking the local accuracy of the solution to 10^{-14}, this same chaotic motion still occurs.

Since the Fourier transform has frequency components that are not multiples of 10, this orbit no longer has a period of 10. However, the spikes still appear roughly equally spaced, which indicates a new fundamental period has emerged. This is characteristic of systems on the road to chaos. The trajectory of the Goggles orbit up to 20 periods is shown in Figure 10, and is clearly chaotic.

According to Montgomery [2], an orbit is KAM stable if "the solutions through most initial conditions sufficiently near the orbit stay near it for all time". Clearly, the Goggles orbit does not stay near its solution for all time with the published initial conditions. This could mean that the Goggles orbit is not KAM stable, assuming the initial conditions published are "sufficiently" close to true and the integration scheme used contributes negligible error.

In the same way, other orbits devolve into chaos after only a handful of periods, and these are shown at various stages of becoming chaotic in Figure 11. Its worth noting that Dragonfly always keeps its form, unlike the others shown and Goggles. Instead of falling into



(a) Phase Plot

(b) Fourier Transform

Figure 9: The phase plot and Fourier transform of the Goggles orbit over 10 periods.

chaos, it diverges enough from periodicity to causes a "collision" (i.e a sufficiently close pass to break the integration algorithm). So, it is not unstable in the same sense as the others.

4.3.3 Forcing Instability

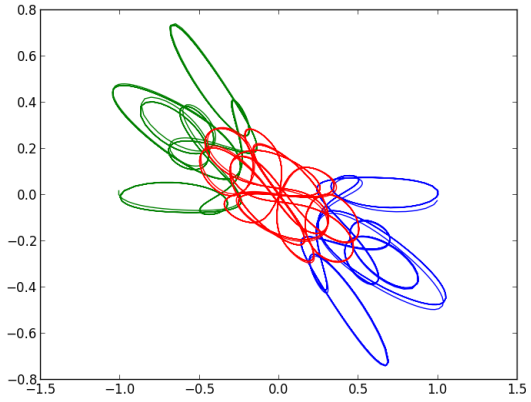
The other orbits that do not fall into chaotic motion after several periods may still be unstable. This is because the magnitude of perturbation required to cause unstable behavior may be significantly larger for these orbits than the others. In this case, the errors in the initial conditions and the integration method may be negligible - larger perturbations are needed to push past margin of stability.

One simple numerical method for probing this margin of stability is adding random perturbations to the motion. In physical systems of orbiting bodies, this is caused by external forces like gravity from faraway objects or collisions with debris.

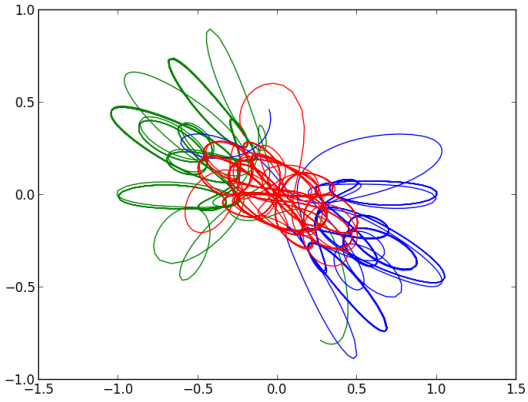
Effectively implementing such random perturbations turns out to be an slightly subtle task. This is due to the nature of the automatic step size algorithm used by the integration method. Namely, the algorithm expects a smooth and deterministic system of equations implemented by `rhs_grav`. It relies on these properties to accurately predict future values of the function, given known values. More importantly, it uses these properties to effectively estimate local error and, from that, the optimal change to its step size parameter.

Implementing a deterministic (as far as the integrator is concerned) method of randomly perturbing the orbits is possible. However, as long as the perturbations are restricted to extremely small, incremental changes, this is unnecessary.

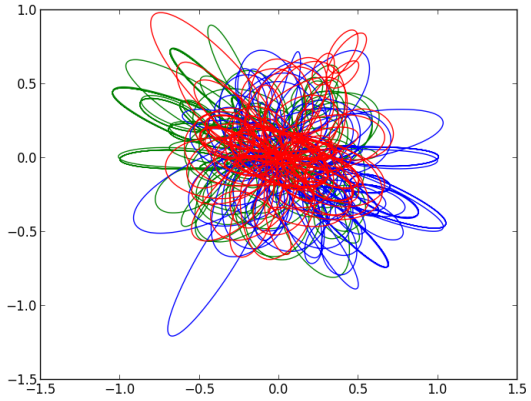
After much trial and error, the implementation that produced the most interesting results was one that simulated debris colliding with and being ejected from each mass. In the



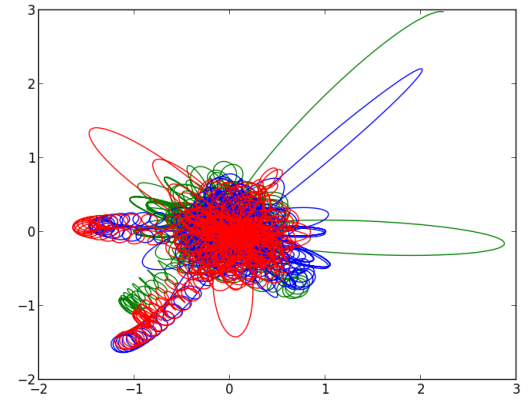
(a) 4 periods



(b) 5 periods

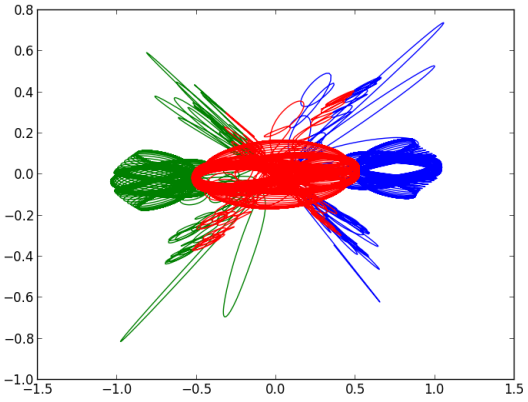


(c) 10 periods

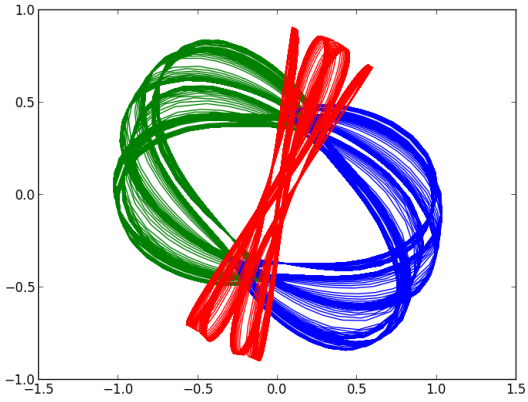


(d) 20 periods

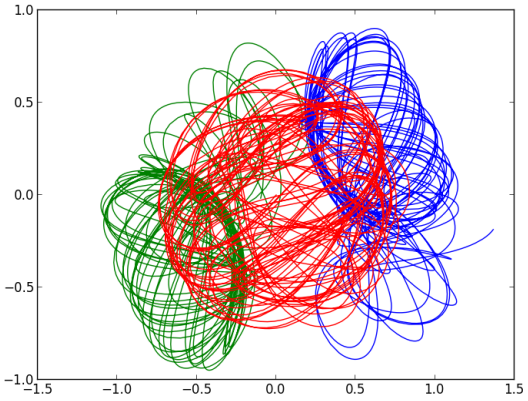
Figure 10: The chaotic trajectory of the Goggles orbit.



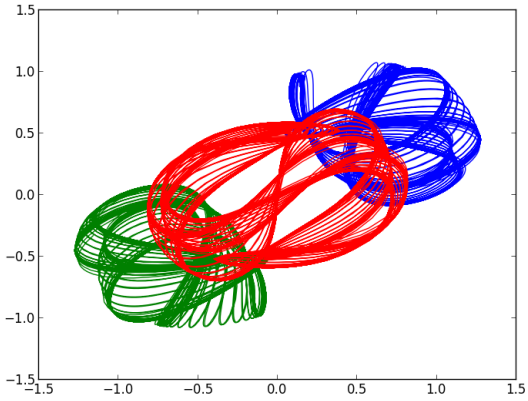
(a) Butterfly I (40 periods)



(b) Dragonfly (40 periods)



(c) Moth III (30 periods)



(d) YinYang I (30 periods)

Figure 11: Unperturbed orbits on the path to chaos.

simulation, ejecting debris is modeled by colliding with debris of negative mass. This is done with the `perturb_masses` function, shown here:

```
void perturb_masses(VecDoub &m) {
    for (int i = 0; i < num_bodies; i++) {
        double dm = 0.0;
        double r = rnd.doub();
        if(r < 0.3333){
            dm = -1 * perturb_amount;
        }
        else if(r < 0.6666){
            dm = perturb_amount;
        }
        if(rnd.doub() < perturb_prob){
            m[i] += dm;
        }
        if(m[i] < 1e-4){
            m[i] = 1e-4;
        }
        mass_of << m[i] << endl;
    }
}
```

This function exploits the fact that a one dimensional random walk with probability of walking less than $\frac{1}{2}$ has a probability of being away from the center that decreases exponentially with distance from the center. In this case, the probability is $\frac{1}{3}$ that it will walk left (subtract mass), $\frac{1}{3}$ that it will walk right (add mass), and $\frac{1}{3}$ that it stays put (leaves the mass alone). With this scheme, the each mass is virtually guaranteed to stay near its initial value of 1, only wandering away with exponentially decreasing probability.

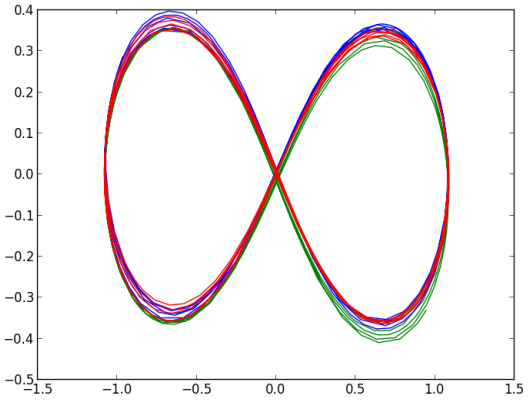
Figure 12 shows the result of putting previously stable orbits into similar debris fields. Some maintain their periodicity, while others clearly fall into chaotic trajectories. Rerunning the simulation with different weights and densities for the debris quickly gives a good sense of which orbits are more stable than others.

Figure 13 shows the masses over time as the bodies pass through the debris. Notice that the increments are small enough to avoid issues with the automatic step size algorithm.

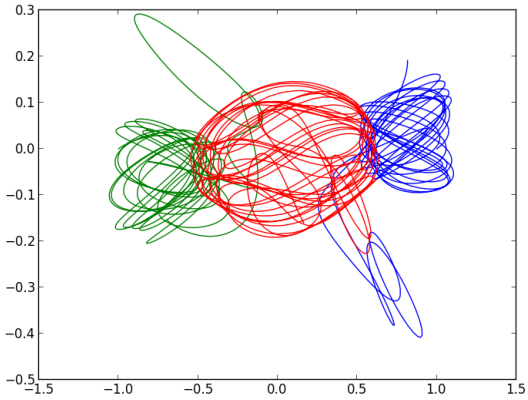
5 Conclusion

There is still much to be learned about three-body orbits, and numerical techniques are often the only choice in exploring their features. S & D used numerical methods to discover the 13 new solutions, and future work with these solutions will likely also be heavily numerical. This includes finding more solutions in existing families, finding new families, and performing further stability analyses.

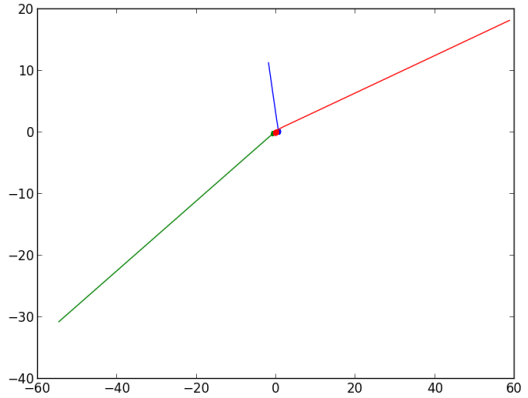
S & D also show that combining the three orbit trajectories in two-dimensional space into a single trajectory in three-dimensional space provides an elegant and generalizable method for analyzing properties of the orbits. Importantly, symmetry properties not evident in the



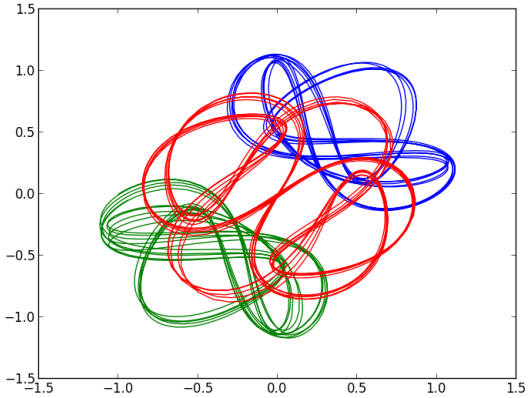
(a) Figure-8



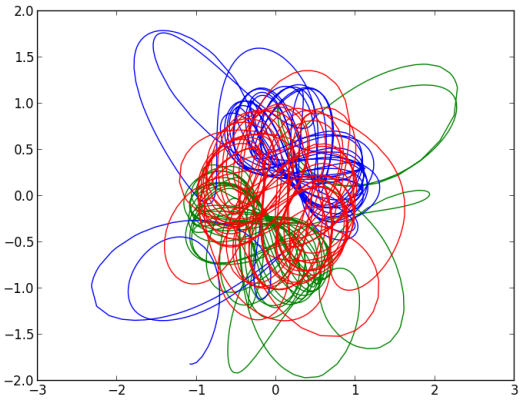
(b) Butterfly II



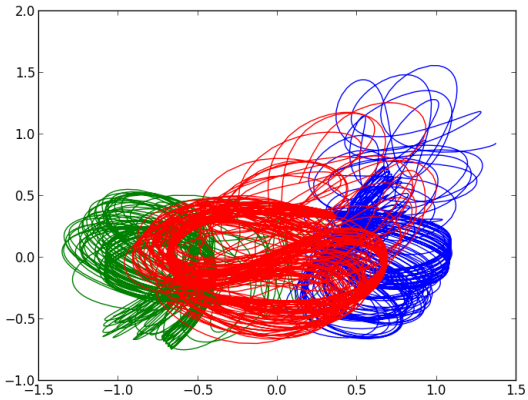
(c) Butterfly III



(d) Moth I



(e) Moth II



(f) YinYang II

Figure 12: Perturbed bodies (debris field) - some stable and some on the road to chaos.

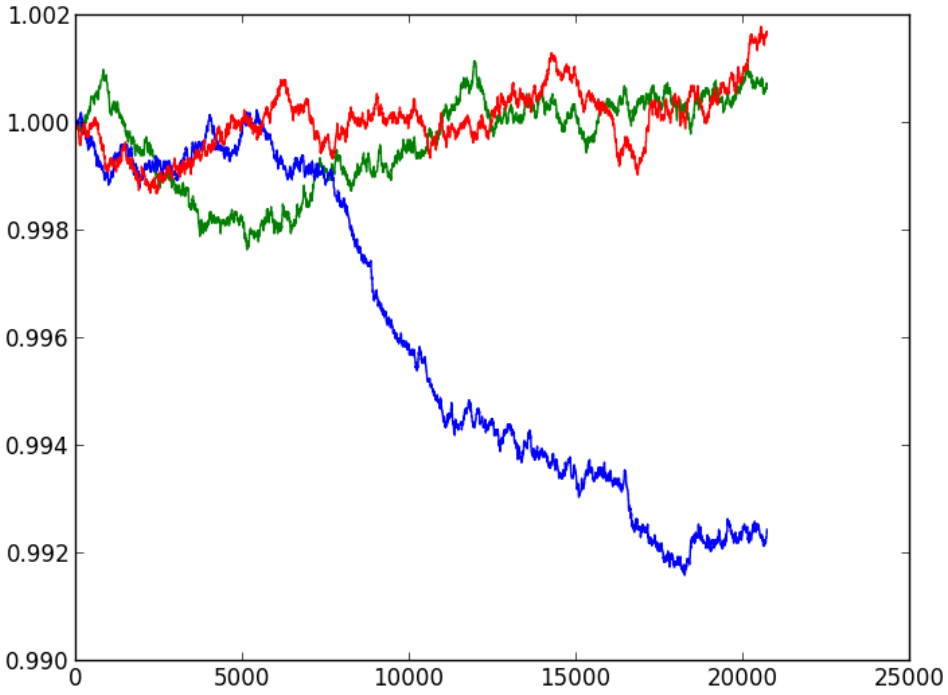


Figure 13: The masses of the bodies going through the debris field in the Moth II orbit.

real-space trajectories are revealed.

Furthermore, phase plots and Fourier transforms are useful tools for analyzing periodicity, and future work could use these for detecting instabilities programmatically (perhaps as part of a Monte Carlo method). This path may be the only way to define the stability of the new solutions, considering the complex analytical methods necessary to determine the stability of the simple Figure-8 [7]. Finally, as shown, even simple numerical techniques can provide good intuition about the relative stability of the new orbits.

6 Included Files

Included in the zip are all the C++, Python, and Bash source files. Also, the "orbits" folder contains files with the initial conditions for each orbit. These files are parsed by `parse_orbits.py`, and the data is then sent to C++ via command line arguments (see `gen_orbit_data.sh`). Also in `gen_orbit_data.sh` are options to generate animations, shape spheres, etc... some of which open interactive matplotlib figures. `run.sh` is the entry point for generating data about any of the listed orbits, and it calls `gen_orbit_data.sh`.

7 Source Code

```
// AEP 4380 Final Project
// Dan Girshovich
// 5/14/13
// Generates orbits and data about the new solutions to the 3 body problem
// Compile: clang++ -std=c++0x -stdlib=libc++ -g -o ../output/$1/main ../cpp/main.cpp
// Tested on Mac OSX 10.8.2 with a Intel Core 2 Duo


#include <cstdio>
#include <cstdlib>
#include <cmath>
#include <fstream>
#include <iostream>
#include <sstream>
#include "nr3.h"
#include "stepper.h"
#include "odeint.h"
#include "stepperdopr853.h"
// #include "stepperdopr5.h"
#include <fftw3.h>
#include "ran.h"
#include <ctime>


string orbit_name;
int num_bodies;
double period;
VecDoub ics, masses;
```

```
Output out;

const double pi = 4.0 * atan(1.0);
const double G = 1;
const double tol = 1e-8; // atol and rtol
const int num_cycles = 5; // should be >= 1 for fft
// const bool dense = true;
const bool dense = false;
// ignored unless dense
const int points_per_cycle = 500;
const int num_points = num_cycles * points_per_cycle;

bool perturb = true;
double perturb_amount = 5e-5;
double perturb_prob = 0.2;
Ran rnd(time(NULL));

double mag(double a, double b) {
    return sqrt(a * a + b * b);
}

ofstream mass_of;

void perturb_masses(VecDoub &m) {
    for (int i = 0; i < num_bodies; i++) {
        double dm = 0.0;
        double r = rnd.doub();
        if(r < 0.3333){
            dm = -1 * perturb_amount;
        }
        else if(r < 0.6666){
            dm = perturb_amount;
        }
        if(rnd.doub() < perturb_prob){
            m[i] += dm;
        }
        if(m[i] < 1e-4){
            m[i] = 1e-4;
        }
        mass_of << m[i] << ' ';
    }
    mass_of << endl;
}

// functor for the 'derivs' calculation in the n body problem
struct rhs_grav {
    VecDoub m;
    double my_t = -1.0;
    rhs_grav(VecDoub mm, int nn) : m(mm) {}
    int x_index(int k) {
        return k;
    }
    int y_index(int k) {
        return k + num_bodies;
```

```
    }
    int vx_index(int k) {
        return k + 2 * num_bodies;
    }
    int vy_index(int k) {
        return k + 3 * num_bodies;
    }
    void set_vals(double &x, double &y, double &vx, double &vy, int k, VecDoub &w) {
        x = w[x_index(k)];
        y = w[y_index(k)];
        vx = w[vx_index(k)];
        vy = w[vy_index(k)];
    }
    // w should contain {x1,..., xn, y1,..., yn, vx1,..., vxn, vy1,..., vyn}
    void operator() (const double t, VecDoub &w, VecDoub &dwdt) {
        if(t > my_t){
            my_t = t;
            if (perturb) perturb_masses(m);
        }

        double xi, xj, yi, yj, vxi, vxj, vyi, vyj;

        for (int i = 0; i < num_bodies; i++) {
            set_vals(xi, yi, vxi, vyi, i, w);
            double sumx = 0, sumy = 0;
            for (int j = 0; j < num_bodies; j++) {
                if (i != j) {
                    set_vals(xj, yj, vxj, vyj, j, w);
                    double dx = xj - xi, dy = yj - yi;
                    double dist = mag(dx, dy);
                    double den = dist * dist * dist;
                    sumx += m[j] * dx / den;
                    sumy += m[j] * dy / den;
                }
            }
            dwdt[x_index(i)] = vxi;
            dwdt[y_index(i)] = vyi;
            dwdt[vx_index(i)] = G * sumx;
            dwdt[vy_index(i)] = G * sumy;
        }
    }
};

// uses odeint with rhs_grav to generate the positions of the three bodies
// results are saved in the global out object
void propagate(double t) {
    rhs_grav derivs(masses, num_bodies);
    double abtol = tol, rtol = tol, hinit = 1e-4, hmin = 0.0;
    Odeint<StepperDopr853<rhs_grav> > ode(ics, 0, t, abtol, rtol, hinit, hmin,
                                          out, derivs);
    ode.integrate();
}

// writes the orbit trajectories
```

```
void write_orbit_data() {
    ofstream of;
    of.open("../output/" + orbit_name + "/orbits.dat");
    for (int i = 0; i < out.count; i++) {
        of << out.xsave[i] << " ";
        for (int j = 0; j < 6; j++) {
            of << out.ysave[j][i] << " ";
        }
        of << endl;
    }
    of.close();
}

// returns the point on the shape space sphere for to the 3 body configuration
// return value is not scaled with the hyper-radius
// see http://suki.ipb.ac.rs/3body/info.php
string get_sphere_point(double q[]) {
    double x1 = q[0], x2 = q[1], x3 = q[2], y1 = q[3], y2 = q[4], y3 = q[5];
    double rho_x = (1 / sqrt(2.0)) * (x1 - x2);
    double rho_y = (1 / sqrt(2.0)) * (y1 - y2);
    double rho = mag(rho_x, rho_y);
    double lambda_x = (1 / sqrt(6.0)) * (x1 + x2 - 2 * x3);
    double lambda_y = (1 / sqrt(6.0)) * (y1 + y2 - 2 * y3);
    double lambda = mag(lambda_x, lambda_y);
    double n_x = 2 * (rho_x * lambda_x + rho_y * lambda_y);
    double n_y = lambda * lambda - rho * rho;
    double n_z = 2 * (rho_x * lambda_y - rho_y * lambda_x);
    stringstream ss;
    ss << n_x << " " << n_y << " " << n_z;
    return ss.str();
}

// writes the shape sphere data
void write_sphere_data() {
    ofstream of;
    of.open("../output/" + orbit_name + "/sphere.dat");
    double q[6];
    for (int i = 0; i < out.count; i++) {
        for (int j = 0; j < 6; j++) q[j] = out.ysave[j][i];
        of << get_sphere_point(q) << endl;
    }
    of.close();
}

// writes the positions and velocities after every period
void write_phase_data() {
    ofstream of;
    of.open("../output/" + orbit_name + "/phases.dat");
    for (int i = 0, j = 0; i < num_cycles; i++, j += points_per_cycle) {
        for (int k = 0; k < 6; k++) {
            of << out.ysave[k][j] << " " << out.ysave[k + 6][j] << " ";
        }
        of << endl;
    }
```

```
    of.close();
}

// updates the array so that the values are centered around 0.0
void center(fftw_complex *q, int len) {
    double sum = 0.0;
    for (int i = 0; i < num_points; i++) sum += q[i][0];
    double average = sum / num_points;
    for (int i = 0; i < num_points; i++) q[i][0] = q[i][0] - average;
}


void write_fft() {
    if (!dense) {
        cout << "Error: fft requires dense output" << endl;
        return;
    }
    fftw_complex *fft_in, *fft_out;
    fftw_plan plan;
    fft_in = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * num_points);
    fft_out = (fftw_complex *) fftw_malloc(sizeof(fftw_complex) * num_points);
    plan = fftw_plan_dft_1d(num_points, fft_in, fft_out, FFTW_FORWARD,
                            FFTW_ESTIMATE);
    for (int i = 0; i < num_points; i++) {
        fft_in[i][0] = out.ysave[0][i];
        fft_in[i][1] = 0; // all real
    }

    center(fft_in, num_points);

    fftw_execute(plan);

    ofstream of;
    string fname = "../output/" + orbit_name + "/fft_x1" + ".dat";
    of.open(fname);
    for (int i = 0; i < num_points; i++) {
        of << fft_out[i][0] << endl;
    }
    of.close();
}

// parses command line args into global vars
void init_globals(char **argv) {
    orbit_name = string(argv[1]);
    num_bodies = atoi(argv[2]);
    period = atof(argv[3]);
    masses = VecDoub(num_bodies);
    for (int i = 0; i < num_bodies; i++) masses[i] = atof(argv[i + 4]);
    ics = VecDoub(2 * 2 * num_bodies); // 2 dimensions * 2 eq/dim * # masses
    for (int i = 0; i < 4 * num_bodies; i++) ics[i] = atof(argv[num_bodies + 4 + i]);
    out = Output(dense ? num_points : 0);
    mass_of.open("../output/" + orbit_name + "/mass.dat");
}
```

```
int main(int argc, char **argv) {
    // TODO: check valid args
    init_globals(argv);
    propagate(num_cycles * period);
    write_orbit_data();
    // write_sphere_data();
    // write_phase_data();
    // write_fft();
    mass_of.close();
}
```

References

[1] M. Šuvakov and V. Dmitrašinović, Three classes of Newtonian three-body periodic orbits, Phys. Rev. Lett. 110 (2013) 114301

[2] R. Montgomery, Nonlinearity 11, 363 - 376 (1998)

[3] http://suki.ipb.ac.rs/3body/

[4] W.H. Press, S.A Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipies, The Art of Scientific Computing*. Camb. Univ. Press, 3rd Edition, 2007.

[5] Numerical Recipes Software 2007, "Routine Implementing an Eighth-order Runge-Kutta Method," Numerical Recipes Webnote No. 20, at http://www.nr.com/webnotes?20

[6] C. MOORE 1993, Braids in classical gravity, Phys. Rev. Lett. 70, 3675–3679.

[7] G. Roberts, Linear Stability Analysis of the Figure-eight Orbit in the Three-body Problem. 2007.

[8] http://fftw.org