Astrophysical N-Body Simulations of Star Clusters

New Mexico
Supercomputing Challenge
Final Report
April 5th 2010

Team 68
Los Alamos High School

Team Members:

Sam Baty
Peter Armijo

Teachers:

Lee Goodwin
Diane Medford

Project Mentors:

Roy Baty
John Armijo

Report Contents

1.0 Executive summary 3

2.0 Problem definition 4

3.0 Solution method 6

3.1 Physical Modeling 6

3.2 Leapfrog Method 8

3.3 Validation Problems 10

3.3.1 One-Body Spring Mass System 10

3.3.2 Two-Body Spring Mass System 11

3.3.3 One-Dimensional Aerodynamic Drag 12

3.3.4 Two-Dimensional Aerodynamic Drag 13

4.0 N-Body Problems 15

4.1 Solar System Simulations 15

4.2 N-Body Simulations 17

4.2.1 500-Body Simulations 18

4.2.2 2,000-Body Simulations 19

4.2.3 10,000-Body Simulations 20

5.0 Summary and Conclusions 21

6.0 References 23

7.0 Figures 25

8.0 Appendices 43

8.1 C-Programs for Validation Problems 43

8.2 C-Programs for N-Body Simulations 47

9. Original Contributions and Acknowledgements 54

1.0 Executive Summary

This report presents simulations of the formation and evolution of small star clusters using a direct N-Body method. In this project, Newtonian mechanics and the Universal Law of Gravitation were combined with the Leapfrog numerical scheme to simulate the evolution of idealized star clusters. All simulations were performed with codes written in the computer language C. Sample mechanics problems were developed to understand and validate the Leapfrog method. The general N-Body code was then used to simulate a simple model of the Solar System, validating the algorithm. The N-Body code was applied to model the evolution of small star clusters with 500, 2,000 and 10,000 bodies.

2.0 Problem Definition

Questions about star clusters are central to understanding many problems in astronomy and cosmology. In our project we have worked to simulate the formation and evolution of small star clusters using a direct N-body method to model their gravitational interactions. Last year for the Supercomputing Challenge our team, Baty and Armijo [1], developed an N-Body code and simulations of the Solar System using a small number of bodies. This year we extended our analysis of the Solar System to model star clusters with up to 10,000 bodies.

The N-Body computational models use Newtonian mechanics and the Universal Law of Gravitation. Each star is modeled as a point-mass (or particle) with a prescribed initial position and velocity. A Leapfrog numerical method has been used to integrate the equations of motion based on Euclidean geometry and Newton's Universal Law of Gravitation. All simulations were performed with codes written in C. Simulations have been run on both Mac and PC platforms with both Apple and Microsoft operating systems. All final simulations for this report were run on a Mac with an OS 10.6 operating system.

In Section 3.0 the solution method is outlined and several basic problems in Newtonian mechanics are solved to develop and apply the Leapfrog integrator. The numerical solution of spring-mass problems and aerodynamic drag problems are presented. These problems were studied in order to understand how the Leapfrog integrator works and how to validate the numerical results. The mechanics problems helped us to develop coding techniques for the N-Body problems.

Section 4.0 applies the Leapfrog method to N-Body problems. A 15-body model of the Solar System is simulated with the Sun, the eight Planets, the Moon, Jupiter's four main moons and Pluto. These simulations are compared with the results from our earlier simulations with a fourth-order Runge-Kutta scheme, [1], and found to be in good agreement. Our new N-Body code, based on the Leapfrog method, is then applied to larger N-Body problems. Simulations with 500, 2,000 and 10,000 bodies have been successfully completed. A random number generator was applied to specify all initial data for these simulations.

Section 5.0 summarizes the results of the project. Conclusions from the moderately large N-Body simulations are highlighted in this section.

3.0 Solution Method

The physical model developed in our project is based on the classical mechanics of rigid bodies. Each body is idealized as a point-mass or particle in three-dimensional space, which is allowed to move in time as a complete system of N-Bodies evolves. This section of the report outlines the basic physical assumptions and develops the mathematical equations required to numerically simulate the motion of moderately large N-Body problems.

Section 3.1 reviews the simple kinematics equations for a point-mass with constant acceleration and introduces the Universal Law of Gravitation. In Section 3.2 the constant acceleration kinematics equations are restricted to short time intervals to predict the velocity and position of point-masses. These equations are then used to define the Leapfrog numerical integration scheme.

Section 3.3 applies the Leapfrog numerical scheme to four classical mechanics problems: one- and two-body spring mass systems, and one- and two-dimensional aerodynamic drag problems. In these problems, both linear and non-linear forces are considered, to validate the numerical method.

3.1 Physical Modeling

The constant acceleration mechanics equations are developed in almost all elementary physics books. Following Meriam [2], the basic kinematics equations in three-dimensions are given by

$$\mathrm{v} = \mathbf{v_0} + \mathbf{a}\, t, \tag{1}$$

$$\mathbf{x} = \mathbf{x_0} + \mathbf{v_0}\, t + \mathbf{a}\, t^2/2, \tag{2}$$

where $\mathbf{v}$ and $\mathbf{x}$ are the velocity and position vectors, and $\mathbf{v_0}$ and $\mathbf{x_0}$ are the initial velocity and position vectors. Moreover, t represents the time of interest and $\mathbf{a}$ represents a constant acceleration vector. In this report, all vectors are written in bold face and are three-dimensional. The present study was preformed in Cartesian coordinates using the notation: $\mathbf{x} = (x, y, z)$, $\mathbf{v} = (u, v, w)$ and $\mathbf{a} = (a_x, a_y, a_z)$. These equations are further discussed in the texts of Serway and Faughn [3], and Williams, et al. [4].

Equations (1) and (2) are used to find the velocity and position, respectively, of a point-mass. By providing initial data in the form of velocity and position we can predict the motion and location of a particle at any future time. These equations are valid only when the acceleration is constant.

Newton's Universal Law of Gravitation for an N-Body problem is given by the following equation:

$$\mathbf{F}_i = - G \sum m_i\, m_j\, (\mathbf{x}_i - \mathbf{x}_j) / (| \mathbf{x}_i - \mathbf{x}_j |^3), \tag{3}$$

where G is the gravitational constant, m_i and m_j are the masses of the N-Bodies, and $\mathbf{x}_i$ and $\mathbf{x}_j$ are the positions of the masses m_i and m_j, respectively. The summation in Equation (3) is taken over the index j when j is not equal to i. This gives the resulting gravitational force on the ith body. Also in Equation (3) the length of the kth position is given by

$$| \mathbf{x}_k | = (x_k^2 + y_k^2 + z_k^2)^{1/2}. \tag{4}$$

In Equations (3) and (4): i, j, k = 1, …, N. The Universal Law of Gravitation is discussed in detail in Aarseth [5].

The Universal Law of Gravitation, given in Equation (3), is related to the motion of the ith body using Newton's 2nd Law:

$$\mathbf{F}_i = m_i\, \mathbf{a}_i. \tag{5}$$

Combining Equations (3) and (5) yields the N-Body equation of motion:

$$\mathbf{a}_i = - G \sum m_j (\mathbf{x}_i - \mathbf{x}_j) / (| \mathbf{x}_i - \mathbf{x}_j |^3). \tag{6}$$

In our project, a variation of Equation (6) is used to simulate the motions and positions of large numbers of point-masses.

3.2 Leapfrog Method

To model an N-Body system, Equation (6) is used to approximate the acceleration of the ith body on a small time interval, Δt. On such a small time interval, the acceleration given by Equation (6) is assumed to be constant, which reduces the problem to the kinematics model of Equations (1) and (2). To see how this works, let

$$\mathbf{a}_i = \Delta\mathbf{v}_i / \Delta t. \tag{7}$$

Solving Equation (7) for $\Delta\mathbf{v}_i$ gives

$$\Delta\mathbf{v}_i = \mathbf{a}_i \Delta t. \tag{8}$$

Next, recalling that

$$\Delta\mathbf{v}_i = \mathbf{v}_i - \mathbf{v}_{0i}, \tag{9}$$

yields

$$\mathbf{v}_i = \mathbf{v}_{0i} + \mathbf{a}_i \Delta t, \tag{10}$$

which is Equation (1). Likewise, if we let

$$\mathbf{v}_i = \Delta\mathbf{x}_i / \Delta t, \tag{11}$$

where, $\Delta\mathbf{x}_i = \mathbf{x}_i - \mathbf{x}_{0i}$, so that

$$\mathbf{x}_i = \mathbf{x}_{0i} + \mathbf{v}_i \Delta t. \tag{12}$$

For an average (constant) acceleration Equation (12) may be shown to be equivalent to Equation (2).

From Reference [6], the Leapfrog method is defined by:

$$\mathbf{v}^{k+1/2} = \mathbf{v}^{k-1/2} + \mathbf{a}^{k}\,\Delta t, \tag{13}$$

$$\mathbf{x}^{k+1} = \mathbf{x}^{k} + \mathbf{v}^{k+1/2}\,\Delta t, \tag{14}$$

where k is the time index. Equations (13) and (14) are solved in time, calculating the positions and velocities at shifted half-time steps. This shift gives the integration method the name Leapfrog, because the position and velocity are calculated alternately at every half-time step. Equations (13) and (14) are the same as Equations (10) and (12) with the modification for the alternating half-time step. The Leapfrog scheme is also known as Verlet integration.

The Leapfrog method was used in our project because it provides an optimum balance of accuracy and computing efficiency. This method is a fixed time step method that minimizes the total computational error in energy, Heggie and Hut [7]. The Leapfrog scheme is a second order accurate numerical method. The scheme was chosen over other integration methods such as a Runge-Kutta method, Burden and Faires [8], because of computational efficiency. The Leapfrog scheme is useful for qualitative physics modeling on long time intervals.

The Leapfrog scheme may be extended to a variable time step scheme by factoring in the slope of the acceleration or the mechanical jerk, Hut, et al. [9]. A variable time step integration scheme allows the N-Body interactions to occur with relative accuracy, in a single simulation, while the interactions are on very different physical time

scales. The practical implementation of a variable time step Leapfrog method is a research activity and was not completed in the present project.

3.3 Validation Problems

3.3.1 One-Body Spring Mass System

Our first validation problem consisted of simulating the motion of a one-body spring mass system. Using Hooke's Law, Serway and Faughn [3],

$$F = - k x, \tag{15}$$

and Newton's 2nd Law, Equation (5), the equation of motion is given by

$$m a = - k x. \tag{16}$$

In Equations (15) and (16), k is the spring constant. Equation (16) may be written solving for acceleration in the form:

$$a = - (k/m) x. \tag{17}$$

Equation (17) is combined in the Leapfrog method given by Equations (13) and (14) to provide a computational model of the one-body spring mass system. At each time step Equation (17) is used to approximate the acceleration.

Following Kreyszig [10], the exact solution to Equation (17) is

$$x(t) = A \cos[\omega t] + B \sin[\omega t], \tag{18}$$

where $\omega = (k/m)^{1/2}$. Figures 1 and 2 compare the position and velocity of the Leapfrog solution (red) to the exact solution (green) assuming the parameters $k = m = 1$ and the initial data $x_0 = 1.0$ and $v_0 = 0.0$, respectively. These initial conditions imply $A = 1.0$ and $B = 0.0$. Figure 3 shows the absolute error between the numerical solution and the exact solution for the step size $\Delta t = \pi/100$. The maximum absolute error in position (red) is ~

0.047 and the maximum absolute error in velocity (green) is ~ 0.031. The C-program implementing the numerical solution of Equation (17) is listed in Appendix 8.1.

3.3.2 Two-Body Spring Mass System

Our second validation problem consisted of simulating the motion of a two-body spring mass system. The spring mass system consists of two masses, m_1 and m_2, restrained by three springs to move horizontally on a frictionless surface. Each spring is assumed to have the spring constant k. Following Greenberg [11], the equations of motion are given by

$$F_1 = - 2 k x_1 + k x_2, \quad (19)$$

$$F_2 = k x_1 - 2 k x_2, \quad (20)$$

where x_1 is the position of m_1, and x_2 is the position of m_2. Equations (19) and (20) come from a free body diagram. Applying Newton's 2nd Law to these equations produces,

$$a_1 = - (2 k/m_1) x_1 + (k/m_1) x_2, \quad (21)$$

$$a_2 = (k/m_2) x_1 - (2 k/m_2) x_2. \quad (22)$$

Equations (21) and (22) are combined in the Leapfrog method given by Equations (13) and (14) to provide a computational model of the two-body spring mass system. This problem was our first example of a system of equations.

From Greenberg [11], the exact solution to Equations (21) and (22) is

$$x_1(t) = 0.5 \cos[t] + 0.5 \cos[\omega t], \quad (23)$$

$$x_2(t) = 0.5 \cos[t] - 0.5 \cos[\omega t], \quad (24)$$

for $m_1 = m_2 = 1$ and $\omega = (3)^{1/2}$, assuming the initial conditions $x_{01} = 1.0$, and $x_{02} = 0.0$, and $v_{01} = v_{02} = 0.0$.

Figures 4 and 5 compare the positions of the Leapfrog solution (red) to the exact solution (green). Figures 6 and 7 compare the computed velocities (red) with the exact velocities (green). Figures 8 and 9 show the absolute errors in the positions as a function of time. For this example the step size was $\Delta t = \pi/200$. The C-program implementing the numerical solution of Equations (21) and (22) is listed in Appendix 8.1.

3.3.3 One-Dimensional Aerodynamic Drag

Our third validation problem consisted of simulating the nonlinear, free-fall motion of a gravity bomb in one-dimension with aerodynamic drag. Aerodynamic drag, D, is defined as the scalar function

$$D = (1/2)\ \rho\ V^2\ C_d\ A, \tag{25}$$

where ρ is the air density, V is the speed, C_d is the drag coefficient, and A is a reference area, Anderson [12]. Using a free body diagram, the acceleration for a falling body of mass m is given by

$$a_z = (1/2m)\ \rho\ V^2\ C_d\ A - g. \tag{26}$$

From Equation (26) the speed of a falling object may be shown to be

$$v(t) = -V_T \tanh (g\ t\ /\ V_T), \tag{27}$$

where V_T is the terminal velocity:

$$V_T = (2mg\ /\ \rho C_d A)^{1/2}. \tag{28}$$

Equations (26), (27) and (28) are discussed in Reference [13].

The Leapfrog scheme of Equations (13) and (14) is combined with Equation (26) to simulate the speed of a gravity bomb. For our calculations, approximate values were used for a GBU-28, Reference [14]. Here m = 2091 kg and A = 0.213 m^2. The drag

coefficient is approximated as $C_d = 0.75$ for transonic flow, Hoerner [15]. This simulation assumes a constant air density of 1.0 and an initial altitude of 12192 m (40,000 ft).

Figure 10 compares the speed computed with the Leapfrog scheme (red) and the solution given by Equation (27) for the step size $\Delta t = 0.01$. The simulation also computed the position of the bomb to estimate the terminal speed. Here the computed speed at ground level was 394.5 m/s in comparison to the terminal speed of 506.7 m/s. Figure 11 shows the absolute error for the speed as a function of time. The C-program implementing the numerical solution of Equation (26) is listed in Appendix 8.1.

3.3.4 Two-Dimensional Aerodynamic Drag

Our fourth validation problem consisted of simulating the nonlinear, free-fall motion of a gravity bomb in two-dimensions with aerodynamic drag. The acceleration of a free falling body in two-dimensions is given by the equations:

$$a_x = - (1/2m)\, \rho\, u\, |\, \mathbf{v}\, |\, C_d\, A, \tag{29}$$

$$a_z = - (1/2m)\, \rho\, w\, |\, \mathbf{v}\, |\, C_d\, A - g, \tag{30}$$

where $|\, \mathbf{v}\, |$ is the magnitude of the velocity vector. The vertical direction, z, is orientated so that down is negative, and $w < 0$. These equations and analytical approximations of their solutions are discussed in Gaude [16].

The Leapfrog scheme of Equations (13) and (14) is combined with Equations (29) and (30) to simulate the planar motion of a gravity bomb. The two-dimensional simulations repeated the analysis of the GBU-28 from Section 3.3.3 using the same physical parameters. This is a nonlinear problem that does not have an exact solution. Therefore, our first simulation duplicated results of the one-dimensional, free-fall

problem of Section 3.3.3 by setting the horizontal velocity to zero. This suggested that the code was working correctly.

Our second simulation was preformed for the initial velocity, **v** = (268.2, 0.0) m/s (600 mph) and initial position x = (0.0, 12192.0) m. Figure 12 shows the horizontal position (downrange) of the bomb as a function of time. Moreover, Figure 13 shows the altitude of the bomb as a function of time. This simulation was performed for the step size $\Delta t = 0.01$. While the results of this simulation could not be checked mathematically, the predicted values seem reasonable for the velocity and length scales of the problem. The C-program implementing the numerical solution of Equations (29) and (30) is listed in Appendix 8.1.

4.0 N-Body Problems

The next step in our project was to combine the acceleration of Equation (6) from the Universal Law of Gravitation with the Leapfrog method of Equations (13) and (14) to simulate example N-Body problems. Each body is idealized as a point mass that is allowed to move in space as a function of time. This section of the report outlines the N-Body numerical experiments that were performed.

Section 4.1 gives the results of a 15-Body model of the Solar System. These simulations are examples of an application of the general N-Body algorithm developed for this project. The results for the Solar System match the basic quantitative results for the motion of the planets.

Section 4.2 then applies the general N-Body Leapfrog algorithm to three moderately large N-Body experiments: 500-Bodies, 2,000-Bodies, and 10,000-Bodies. The computational expense of the N-Body algorithm is examined. Simulations are preformed with different time step sizes and softening constants.

4.1 Solar System Simulations

The first gravitational N-Body problem was to repeat a model of the Solar System, including the Sun, nine planets, the four major moons of Jupiter, and the Earth's Moon. This model used the Leapfrog scheme and accurately simulated the orbits of these bodies in comparison to published NASA data, Williams [17]. The results for the complete Solar System were calculated using large time steps, ranging up to two hours, because the orbits of some of the planets have very long periods around the sun. The code was used to simulate time periods of up to approximately 300 years. All simulations assumed

physical units of Moon masses (7.350 x 10^{23} kg), hours and kilometers.

In the simulations, we accounted for several physical differences in the geometry of the orbits of the planets. For instance, the orbits of both Mercury and Pluto are at an angle relative to the orbits of the other planets. Mercury's orbit is at a 7° angle and Pluto's is at a 17° angle. These angles were modeled to simulate the geometric differences in Mercury's and Pluto's orbits, so the calculations reflect these geometric effects correctly. Pluto's orbit intersects Neptune's orbit at two points. Figures 14 and 15 show the orbits of the inner and outer planet about the Sun, respectively. The Leapfrog integration scheme produced simulations that were in good qualitative agreement with our simulations from last year using a fourth-order Runge-Kutta scheme, Baty and Armijo [1].

For a fixed time step, the required computational time had to be balanced with the accuracy of the simulation. When a fairly large time step was chosen, the simulations could breakdown and give unreliable results. As an example, consider a simulation of Jupiter and its four major moons: Io, Europa, Ganymede, and Callisto. In the Solar System simulations, it was found that a time step greater than a few hours yielded simulations that would lose the moons of Jupiter after a few hundred iterations of the calculation. This behavior is shown in Figure 16. The error accumulated quickly, producing unstable and decaying orbits. This suggested that complex N-Body problems require detailed time step parameter studies. Figure 17 shows a stable simulation of the orbits of Jupiter's four major moons about the planet's trajectory. The C-program for the Solar System problem is listed in Appendix 8.2.

4.2 N-Body Simulations

The basic physics and astronomy of star clusters is very complex. Detailed references on galactic astronomy may be found in Binney et al. [18] and [19]. Moreover, recent work on the N-Body problem may be found in Aarseth et al. [20]. The strategy in our project was to define an idealized model and run numerical experiments on an increasing number of bodies. The N-Body simulations assumed a total mass of 1 for each simulation. Therefore, the mass of each body in a given simulation was 1/N. The gravitational constant G in Equation (6) was defined to be 1. Standardized units for N-Body problems are presented in Heggie and Mathieu [21].

A random number generator was used to define three initial positions in the eight octants of space, +/- x, +/- y, and +/- z for each particle. The initial positions were multiplied by $10*(3)^{1/2}$, which was an arbitrary length scale for the problem. This approach generated initial positions in a cube in space of width $20*(3)^{1/2}$. The particles outside of a sphere of constant radius $10*(3)^{1/2}$ were discarded so that the initial distribution of particles was spherical. The resulting particles contained in the sphere were shifted so that the average of position in each direction about the origin was zero. The initial velocity of all the particles was zero.

To prevent collision singularities in the numerical model, Equation (6) was modified with a softening term $\boldsymbol{\varepsilon}$

$$\mathbf{a}_i = - G \sum m_j (\mathbf{x}_i - \mathbf{x}_j) / (| \mathbf{x}_i - \mathbf{x}_j + \boldsymbol{\varepsilon} |^3). \quad (31)$$

This term prevents the denominator of Equation (31) from being zero, so that the acceleration is undefined. The softening parameter is an artificial fix in the N-Body model and its effect must be carefully considered on the results of numerical experiments.

To understand the computational expense of our direct N-Body algorithm, computational timing studies were performed for N-Body problems up to 10,000 bodies. The N-Body C-program was set to perform 100 iterations of the code. Figure 18 shows the computational time required to perform the iterations as a function of the number of bodies. Roughly the computational time (or expense) increases like N^2, where N is the number of bodies. The C-program for the arbitrary N-Body problem is listed in Appendix 8.2. A good reference for the C programming language is by Kernighan and Ritchie [22].

4.2.1 500-Body Simulations

For the 500-Body simulation we adjusted the total number of bodies to produce 500 bodies within the sphere of constant radius. The step size was $\Delta t = 0.1$, and the simulation was run to 4,000 iterations. This calculation took on the order of an hour to complete. A Gnuplot, [23] animation was generated using 60 frames in time showing the x-y coordinates of the evolving 500-Body system. A parametric family of simulations was run to study the effect of the softening constants of 0.001, 0.01, 0.1, and 1.0.

After the start of the simulation, the system collapses onto itself within a short amount of time. A small number of bodies then shoot through the system, not colliding with any of the other bodies, to escape the problem. Many of the other bodies collide and rebound from the collapsed system. After the system collapses, the majority of the bodies are grouped in a tight cluster near the origin of the calculation. This tight cluster slowly

evolves in three dimensions with particles leaving and returning to the principal globular concentration of mass. It appears that the system reaches a quasi-equilibrium state after about 2,000 iterations of the code.

Figure 19 shows the initial positions of the masses in the x and y coordinates. Figures 20 through 24 show various stages in the evolution of the cluster in time. All of these figures show the data in the x-y coordinate plane. Figure 25 shows the complete time history of all 500 bodies for the four different softening constants. These different softening constants produce different trajectories and velocities for the rebounding particles. The gross features of the cluster are less sensitive to the value of the softening constant than the escaping particles.

4.2.2 2,000-Body Simulations

For the next simulations, 2,000 bodies were modeled. Two simulations of this cluster were run, to study the effect of time step size on the calculations. The first simulation used a time step size of $\Delta t = 0.1$ and was run for 60,000 iterations. The second simulation had a step size of $\Delta t = 0.05$ and was run for 120,000 iterations. The first simulation required approximately 10 hours of computational time, while the second simulation required approximately 19.7 hours of computational time.

Both simulations showed the same basic behavior as the 500-Body problem, where the cluster first collapses on itself and bodies are lost from the system. A central globular cluster then forms close to the origin of the calculation. This mass concentration is stable until approximately 36,000 iterations of the code, when the simulation forms a jet in the center of the concentration of mass. This jet shoots out from the origin of the

simulation and does not return. It is not understood why the jet forms. One possibility is the buildup of the computational error and the resulting numerical instabilities. Both simulations predicted the formation of the jet, with the 120,000 iteration simulation showing it later in the calculation. Figures 26, 27 and 28 show the $\Delta t = 0.05$ simulation at various times. Figure 29 shows the jetting phenomenon at the end of the calculation.

4.2.3 10,000-Body Simulations

For our final simulation, a cluster was run consisting of 10,000 bodies. The beginning behavior of this cluster was similar to the behavior of the 500-Body and 2000-Body problems. After the initial collapse of the system, the cluster moved apart, creating two separate globular masses. These two masses then slowly moved away from each other and interacted with one another by transferring bodies back and forth between them. The globular masses formed around 1250 iterations of the simulation. It is possible that these structures are related to the formation of a spiral distribution of particles. This has not been proved. This simulation was run for 3,000 iterations with a step size of $\Delta t = 0.1$ and took approximately 12.4 hours to complete. Because of the size of the simulation no attempt to re-run the problem with a smaller step size was made.

Figures 30 through 33 show the beginning evolution of the 10,000-Body problem. Figures 34, 35, and 36 show the two globular masses at the end of the simulation (3000 iterations) giving their relative positions in three dimensions. Figure 34, Figure 35, and Figure 36 show the x-y plane, the x-z plane, and the y-z plane, respectively.

5.0 Summary and Conclusions

This year our team completed simulations of several sizes of N-Body systems, with various levels of physical detail. The team converted a 15-Body model of the Solar System, based on a fourth-order Runga-Kutta scheme, to a model based on a second-order Leapfrog scheme. To develop and validate the Leapfrog algorithm, the method was also applied to several mechanics problems with and without exact analytical solutions. These problems included spring mass systems and aerodynamic drag problems. After it was determined that the computational scheme was working, the Leapfrog algorithm was applied to simple models of star clusters with 500, 2,000, and 10,000 bodies.

Our key conclusions include observations about the results of the direct N-Body simulations. First, the N-Body problems become harder to simulate as the number of bodies increases, because the computational expense increases quadratically. Second, N-Body simulations required careful tracking of the time step size to avoid a degradation of the simulation. This sort of behavior was seen in one of our Solar System problems with the moons of Jupiter. Another observation is that large N-Body systems like the cluster simulations all initially behave like one another. The initial phase of all the large N-Body simulations consisted of the system collapsing in on itself and bodies shooting through the origin and out the other side. The differences between the simulations occur later in the calculations. For example, the 500-Body problem formed one cluster at the origin, while the 10,000-Body problem created two clusters that interacted with one another. Moreover, for the 2,000-Body problem, the importance of choosing a time step size to prevent accumulation errors in the simulation is implied by the formation of the jetting phenomenon. N-Body simulations demand a careful balance of all of the technical details

in the problem: the time needed to run the computational model, the time step size, the accuracy of the simulation, and softening method applied to remove collisional singularities. Each of these details requires in depth consideration and study to successfully apply and understand the results of an N-Body simulation.

6.0 References

1. Baty, S.R., and Armijo, P.J., Astrophysical N-Body Simulations of Star Clusters, Supercomputing Challenge, Final Report, 2009.
2. Meriam, J.L., **Engineering Mechanics Volume 2: Dynamics**, John Wiley & Sons, 1978.
3. Serway, R.A. and Faughn J.S., **Physics**, Holt, Rinehart and Winston, 2006.
4. Williams, J.E., Metcalfe, H.C., Trinklein, F.E. and Lefler, R.W., **Modern Physics**, Holt, Rinehart and Winston, Inc., 1968
5. Aarseth, S.J., **Gravitational N-Body Simulations**, Cambridge University Press, 2003.
6. Anonymous, Wikipedia Article: en.wikipedia.org/wiki/Leapfrog_integration, Called upon 3/28/2010.
7. Heggie, D.C. and Hut, P., **The Gravitational Million-Body Problem**, Cambridge University Press, 2003.
8. Burden, R.L., and Faires, J.D., **Numerical Analysis, 3rd Edition**, Prindle, Weber & Schmidt, 1985.
9. Hut, P., Makino, J., and McMillan, S., "Building A Better Leapfrog," The Astrophysical Journal, Volume 443, 1995.
10. Kreyszig, E., **Advanced Engineering Mathematics, 8th Edition**, John Wiley & Sons, 1999.
11. Greenberg, M.D., **Advanced Engineering Mathematics, 2nd Edition**, Prentice Hall, 1998.
12. Anderson, J.D. Jr., **Fundamentals of Aerodynamics**, McGraw-Hill, 1984.

13. Anonymous, Wikipedia Article: en.wikipedia.org/wiki/Free_fall, Called upon 3/31/2010.

14. Anonymous, Federation of American Scientists Article: www.fas.org/man/dod-101/sys/smart/gbu-28.htm, Called upon 3/31/10.

15. Hoerner, S.F., **Fluid Dynamic Drag**, published by Hoerner, 1965.

16. Gaude, B.W. "Solving Nonlinear Aeronautical Problems using the Carleman Linearization Method," Sandia National Laboratories SAND2001-3064, September 2001.

17. Williams, D.R., "NASA Planetary Fact Sheets," nssdc.gsfc.nasa.gov /planetary/factsheet, September 2004.

18. Binney, J., and Merrifield, M., **Galactic Astronomy**, Princeton University Press, 1998.

19. Binney, J., and Tremaine, S., **Galactic Dynamics 2nd Edition**, Princeton University Press, 2008.

20. Aarseth, S.J., Tout, C.A., and Mardling, R.A., Editors, The Cambridge N-Body Lectures, Springer, 2008.

21. Heggie, D.C., and Mathieu, R.D., "Standardised Units and Time Scales," in: **The Use of Supercomputers in Stellar Dynamics**, Edited by Hut, P. and McMillan S., Springer, 1986.

22. Kernighan, B. W. and Ritchie, D. M., **The C Programming Language, 2nd Edition**, Prentice Hall, 1988.

23. Crawford, D., Webpage Article: "gnuplot: An Interactive Plotting Program," www.gnuplot.info/docs_4.3/gnuplot.pdf, December 1998

7.0 Figures

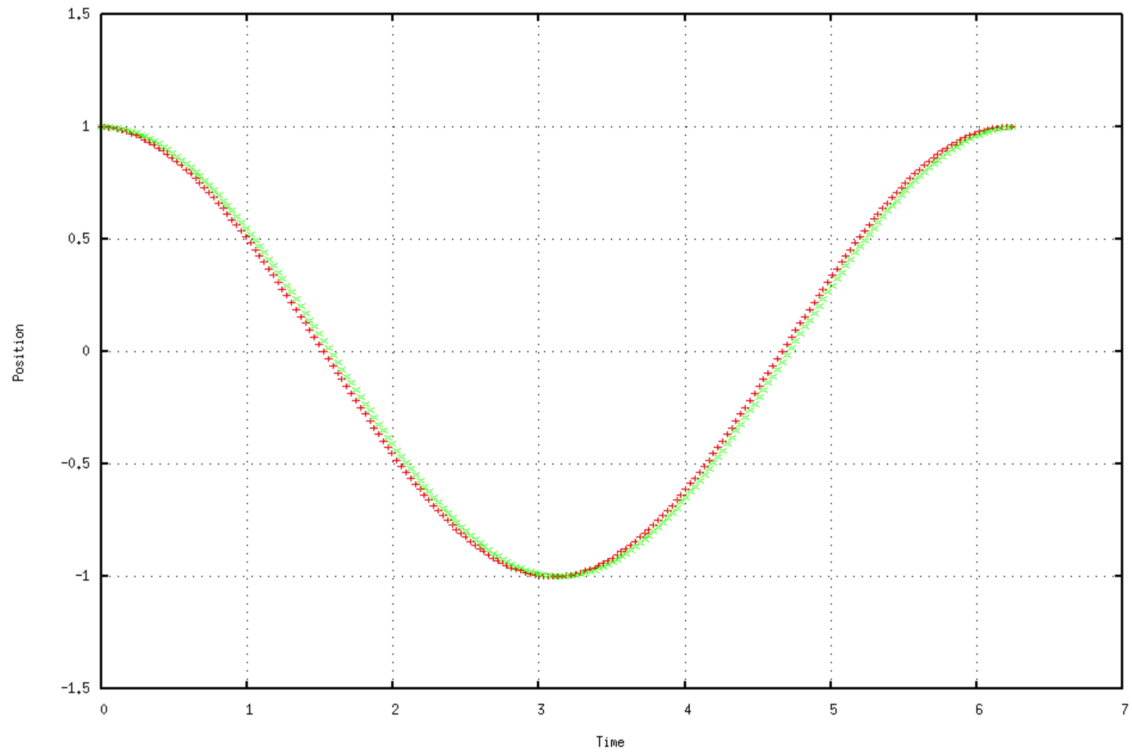

Figure 1. Spring Mass System, Position vs. Time, Leapfrog Solution is red.

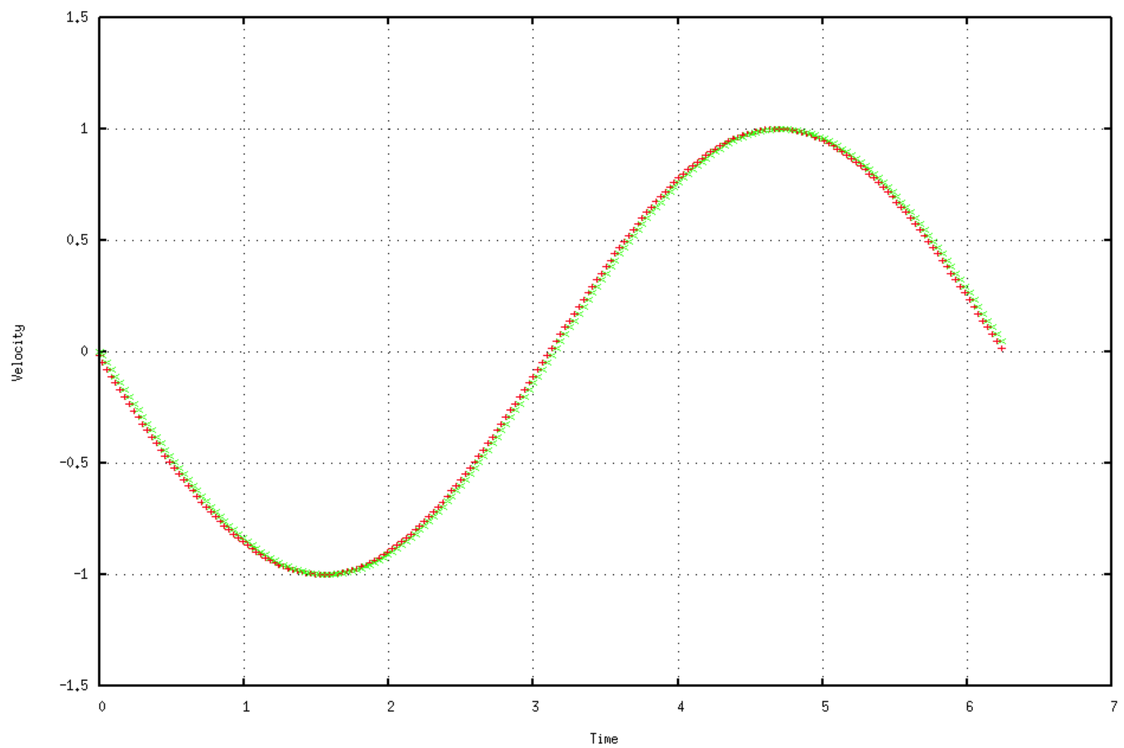

Figure 2. Spring Mass System, Velocity vs. Time, Leapfrog Solution is red.

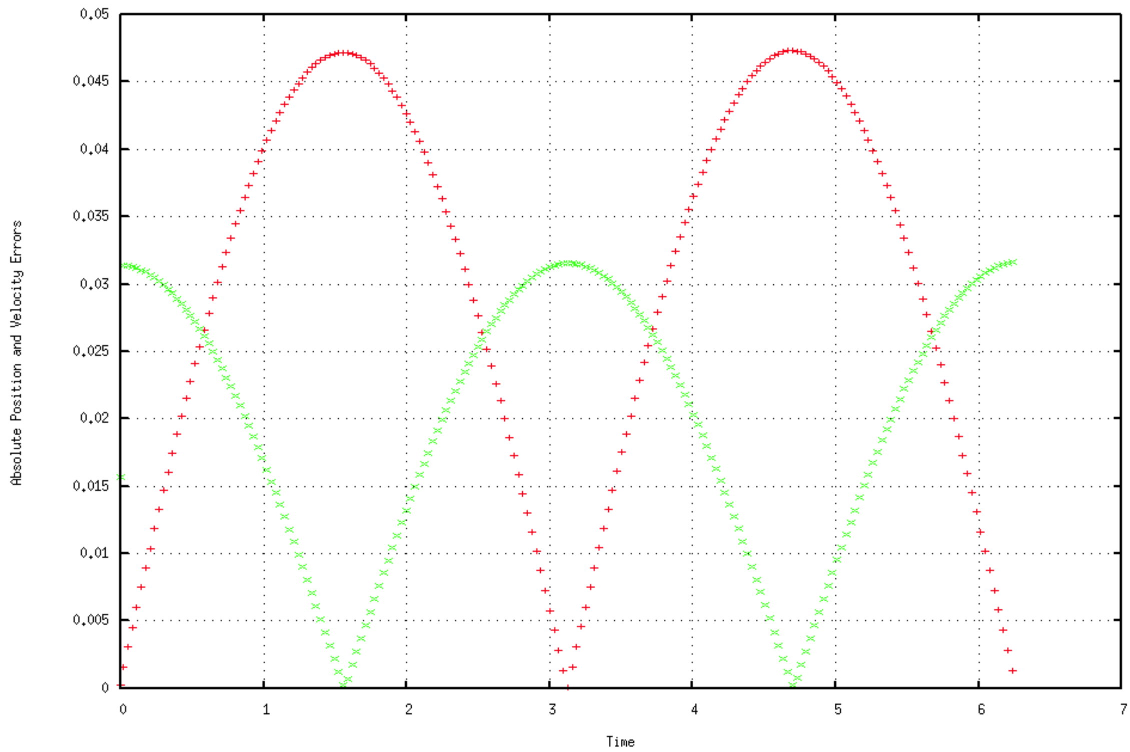

Figure 3. Spring Mass System, Absolute Position & Velocity Errors vs. Time.

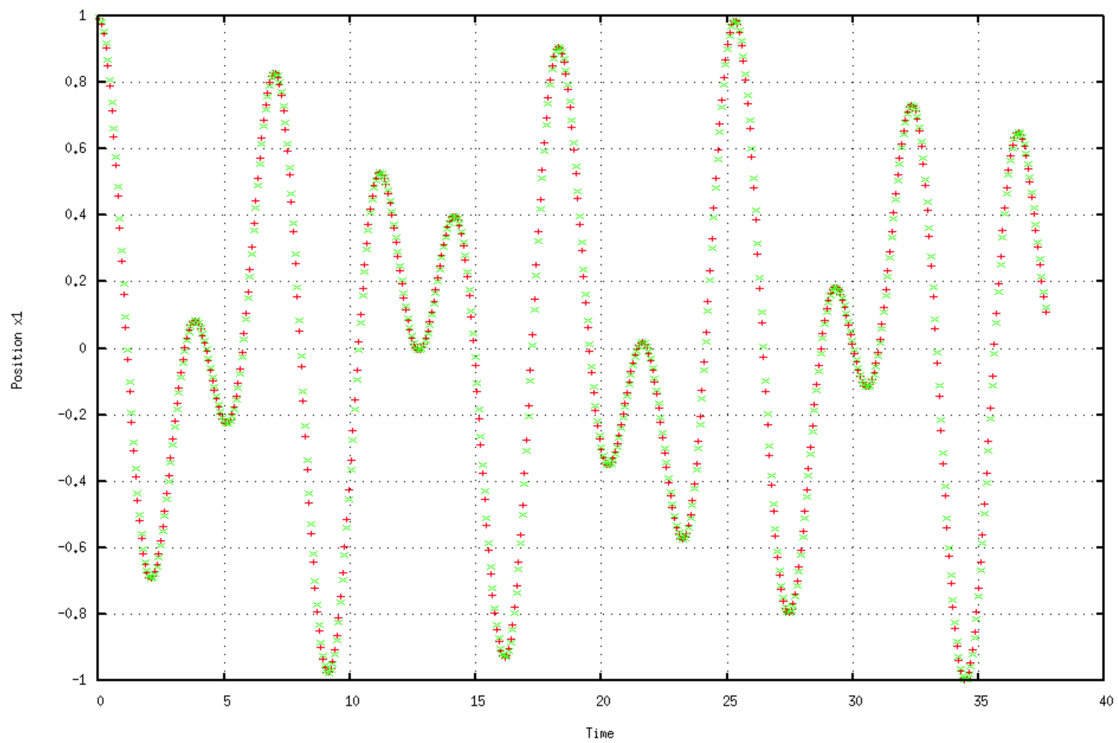

Figure 4. 2-Body Spring Mass System, x1 Position vs. Time, Leapfrog Solution is red.

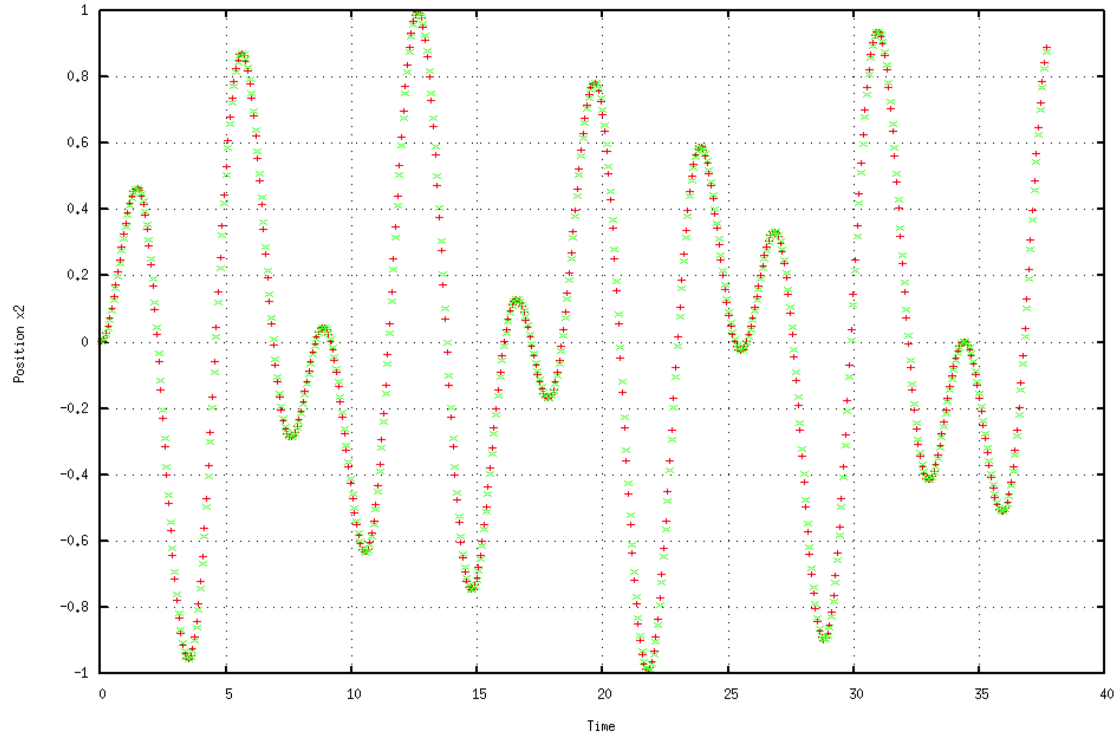

Figure 5. 2-Body Spring Mass System, x2 Position vs. Time, Leapfrog Solution is red.

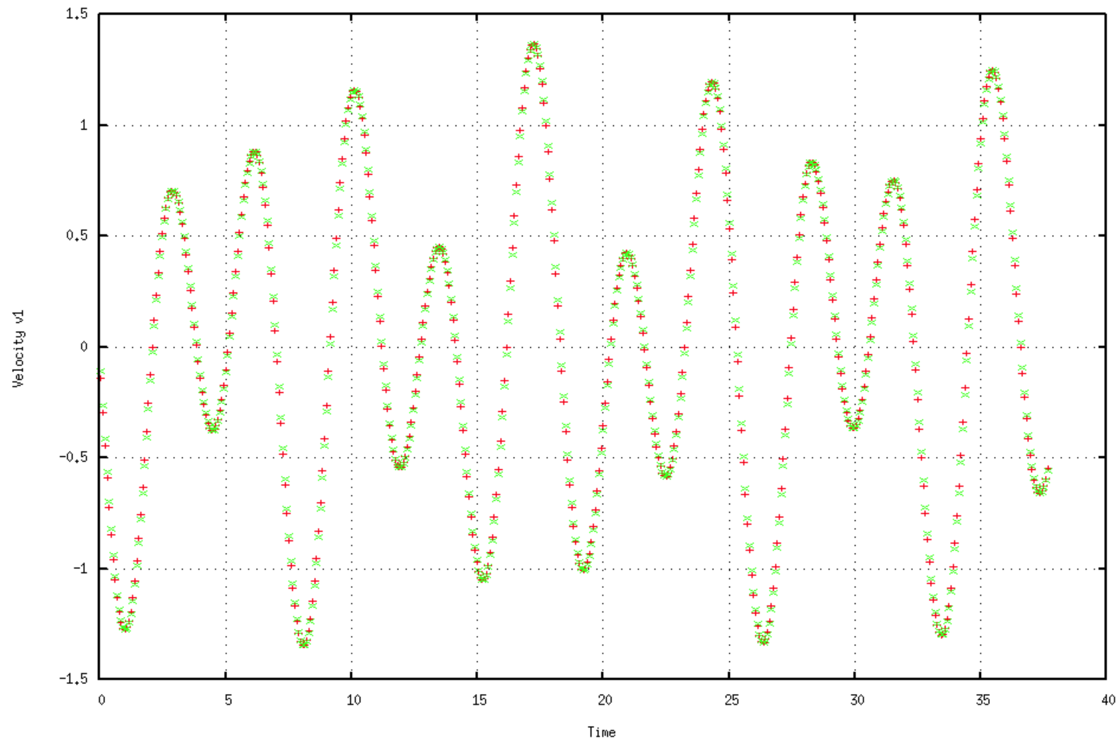

Figure 6. 2-Body Spring Mass System, v1 Velocity vs. Time, Leapfrog Solution is red.

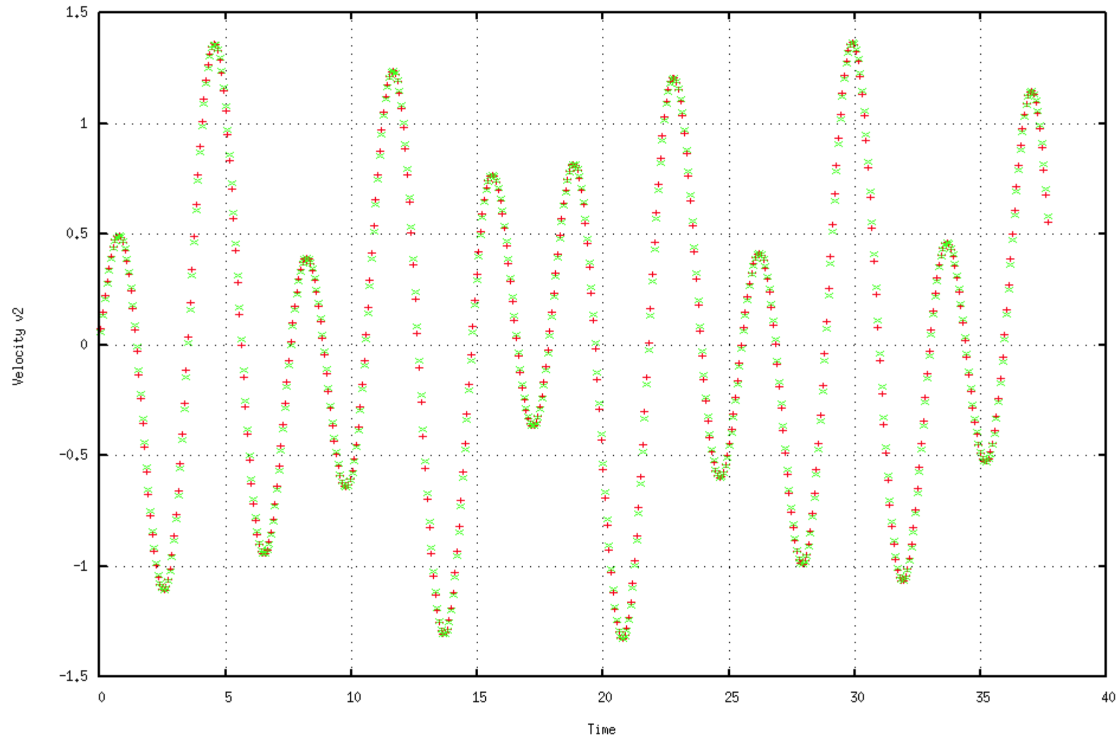

Figure 7. 2-Body Spring Mass System, v2 Velocity vs. Time, Leapfrog Solution is red.

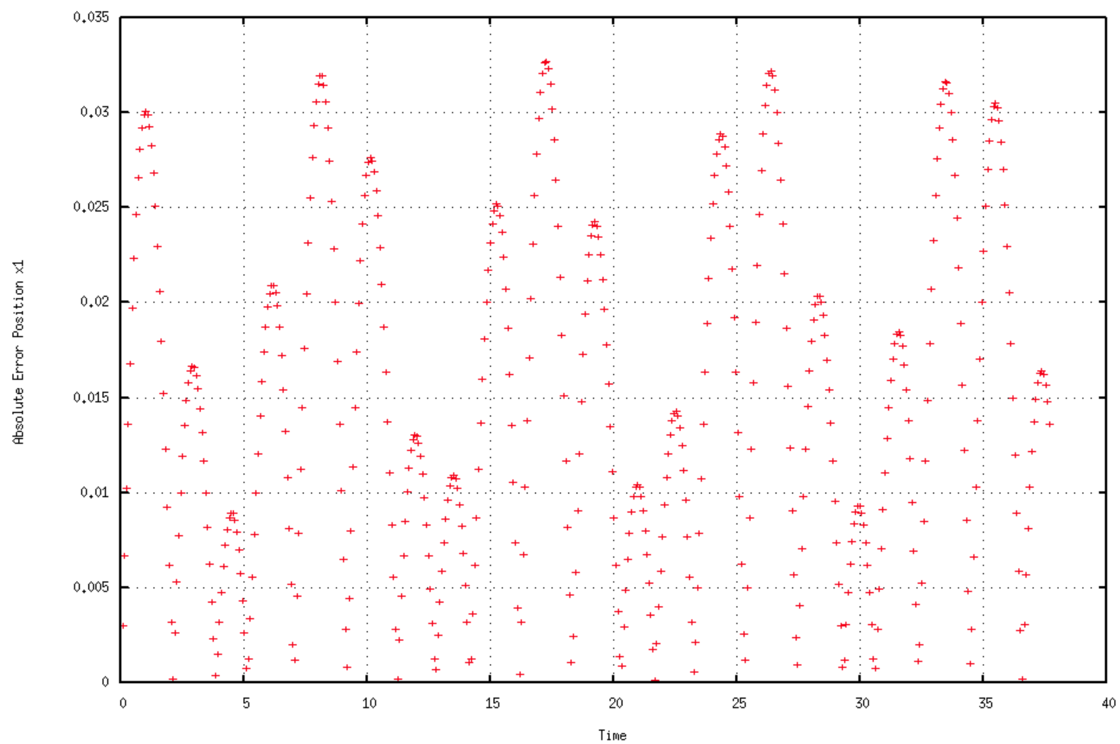

Figure 8. 2-Body Spring Mass System, x1 Position Absolute Error vs. Time.

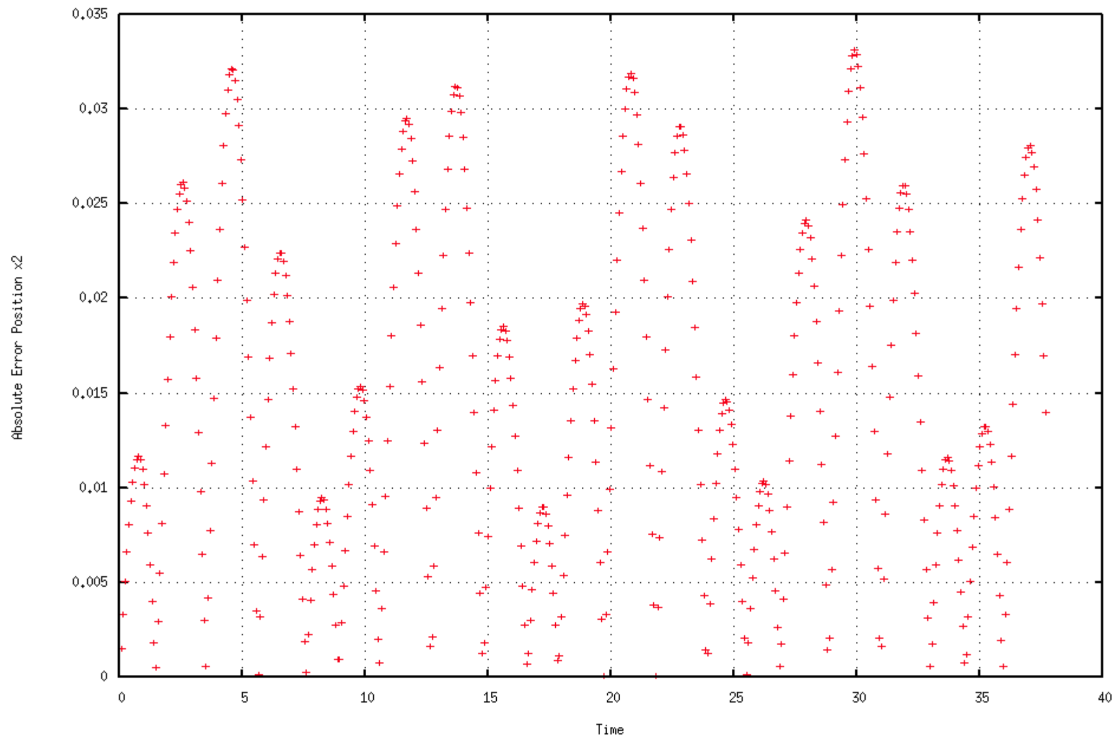

Figure 9. 2-Body Spring Mass System, x2 Position Absolute Error vs. Time.

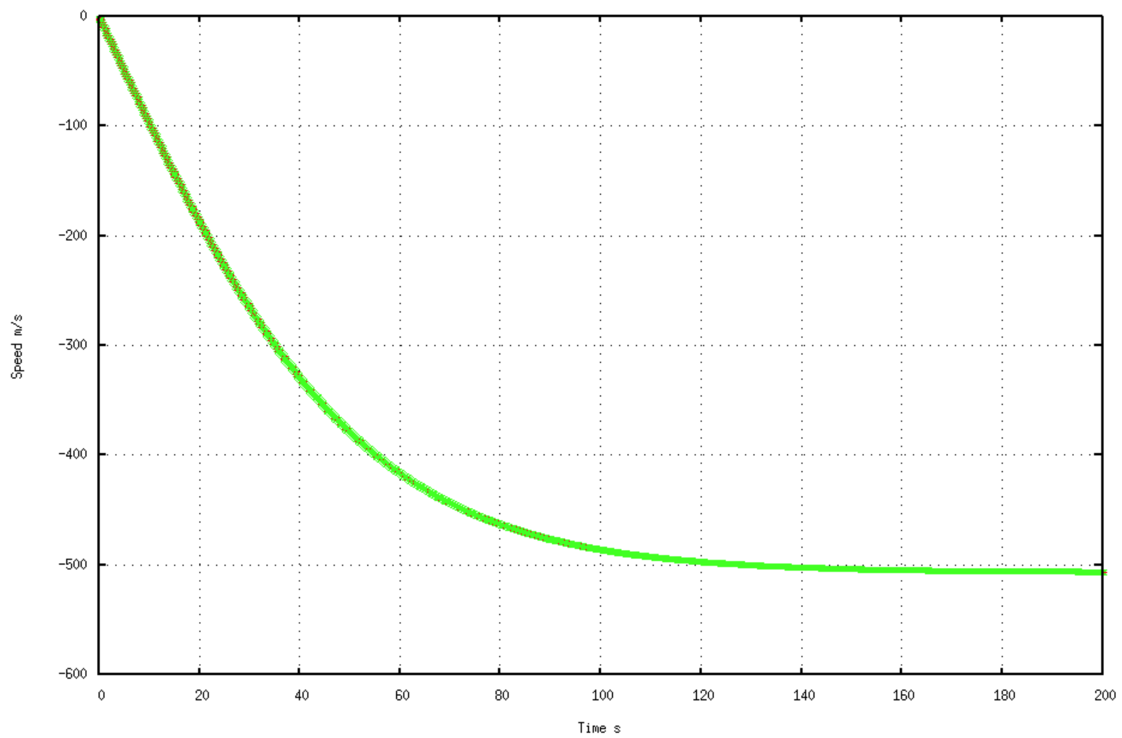

Figure 10. 1-D Free Fall Problem, Leapfrog Solution is red (under green line).

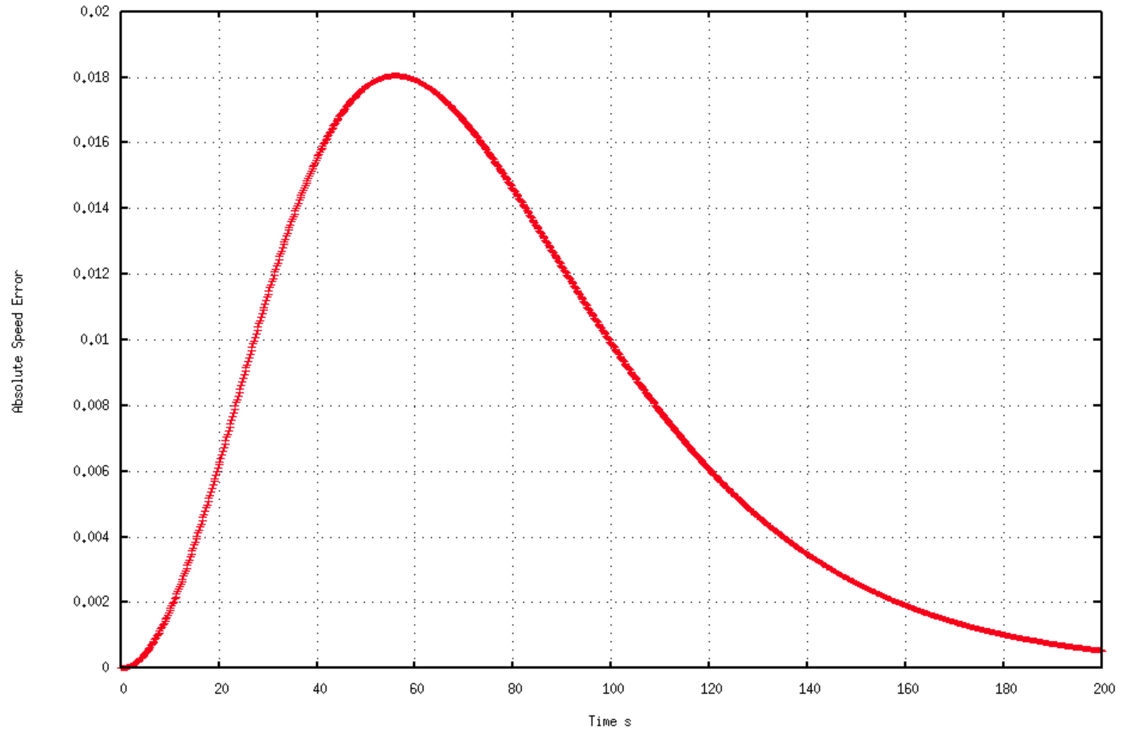

Figure 11. 1-D Free Fall Problem, Speed Absolute Error vs. Time.

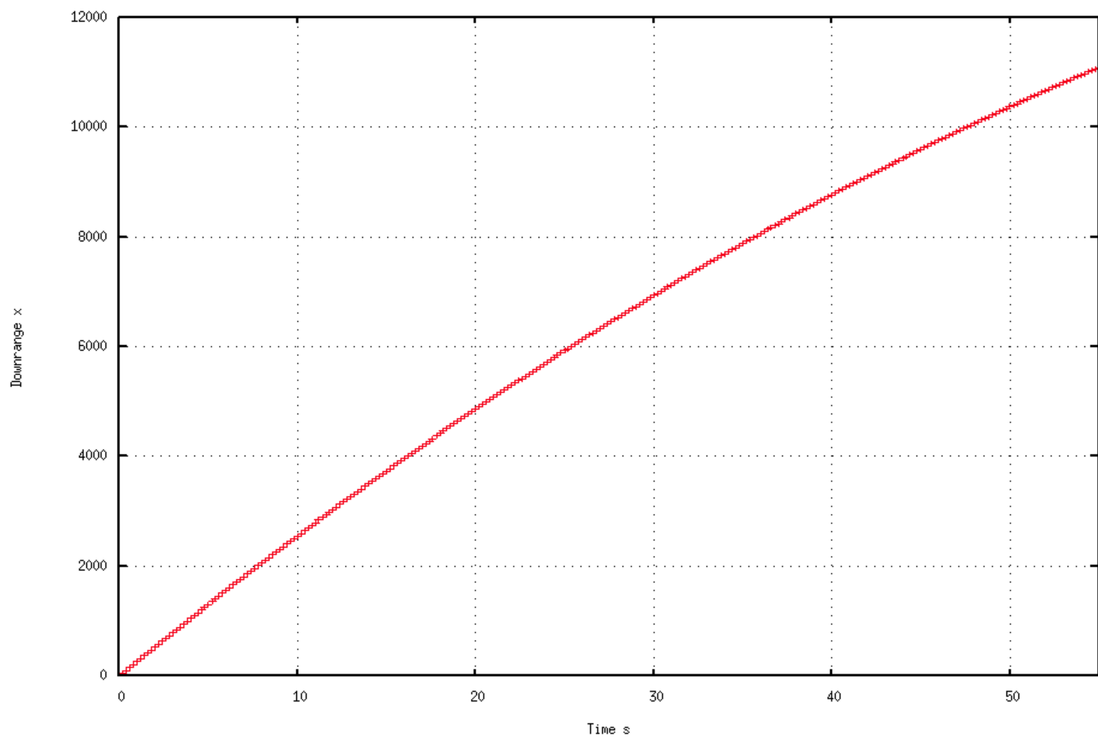

Figure 12. 2-D Free Fall Problem, Downrange (x Position) vs. Time.

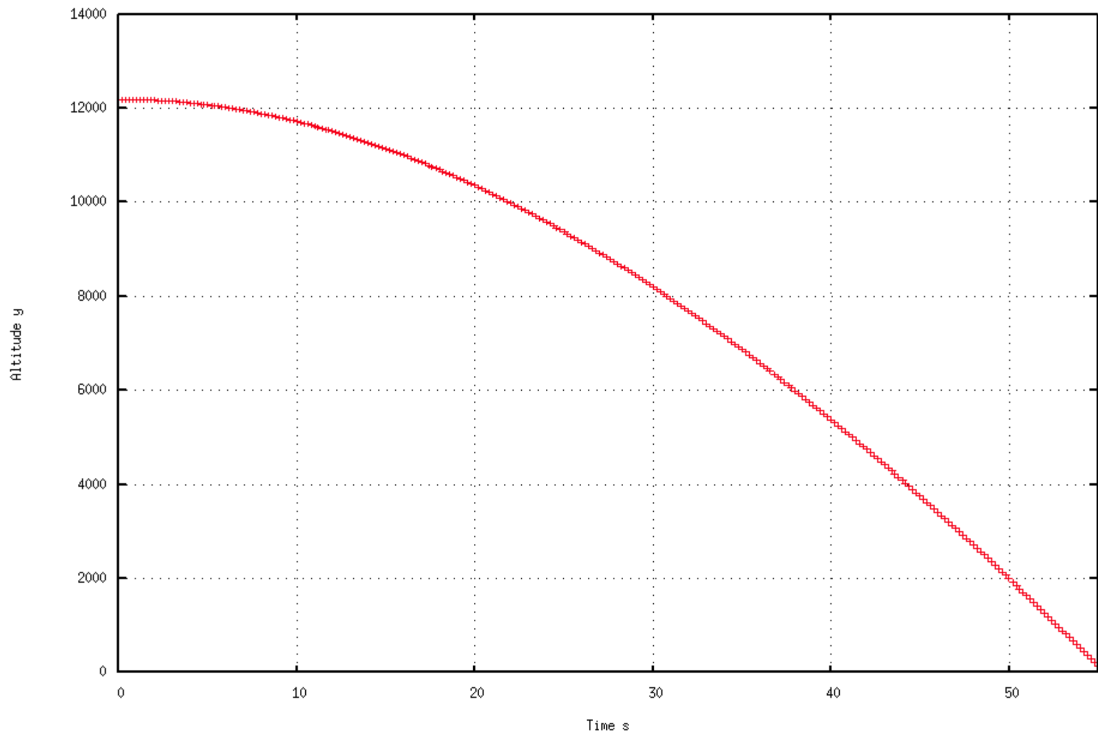

Figure 13. 2-D Free Fall Problem, Altitude (z Position) vs. Time.

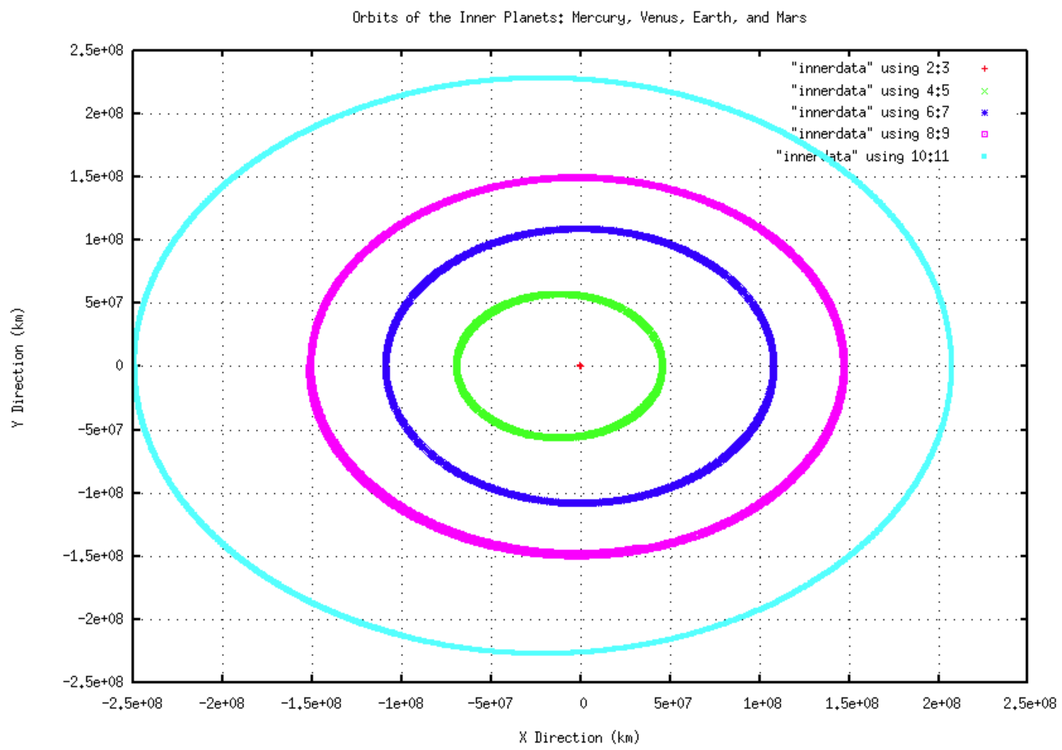

Figure 14. Solar System, Inner Planet Orbits: Mercury, Venus, Earth (& Moon) & Mars.

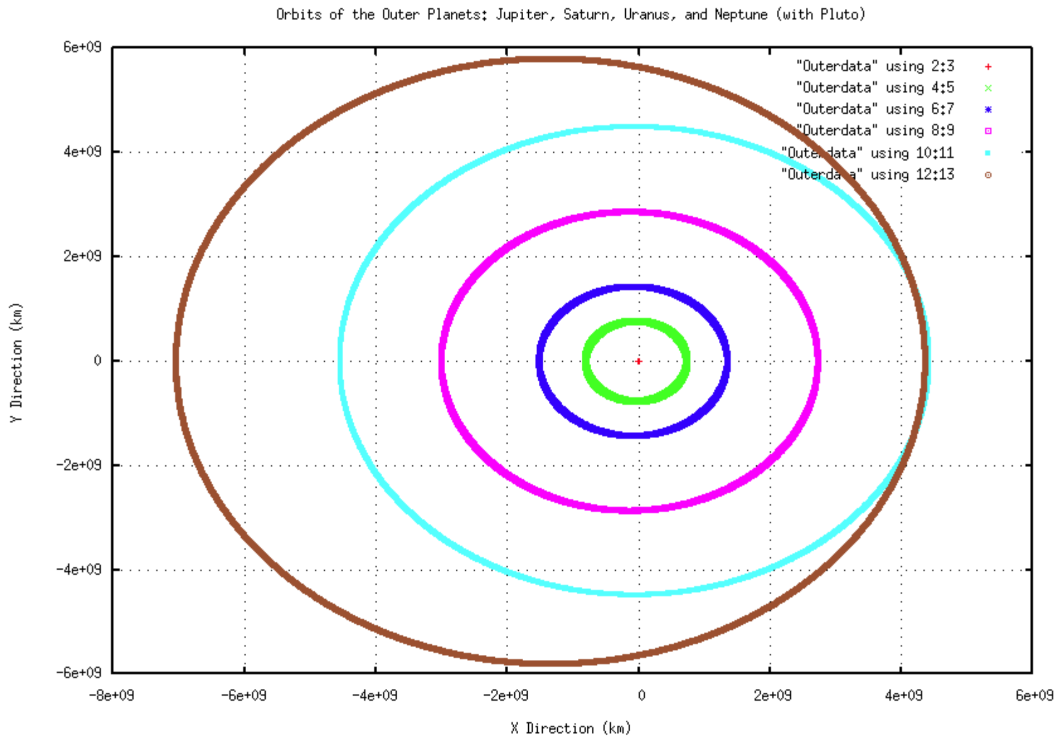

Figure 15. Solar System, Outer Planet Orbits: Jupiter, Saturn, Uranus, Neptune & Pluto.

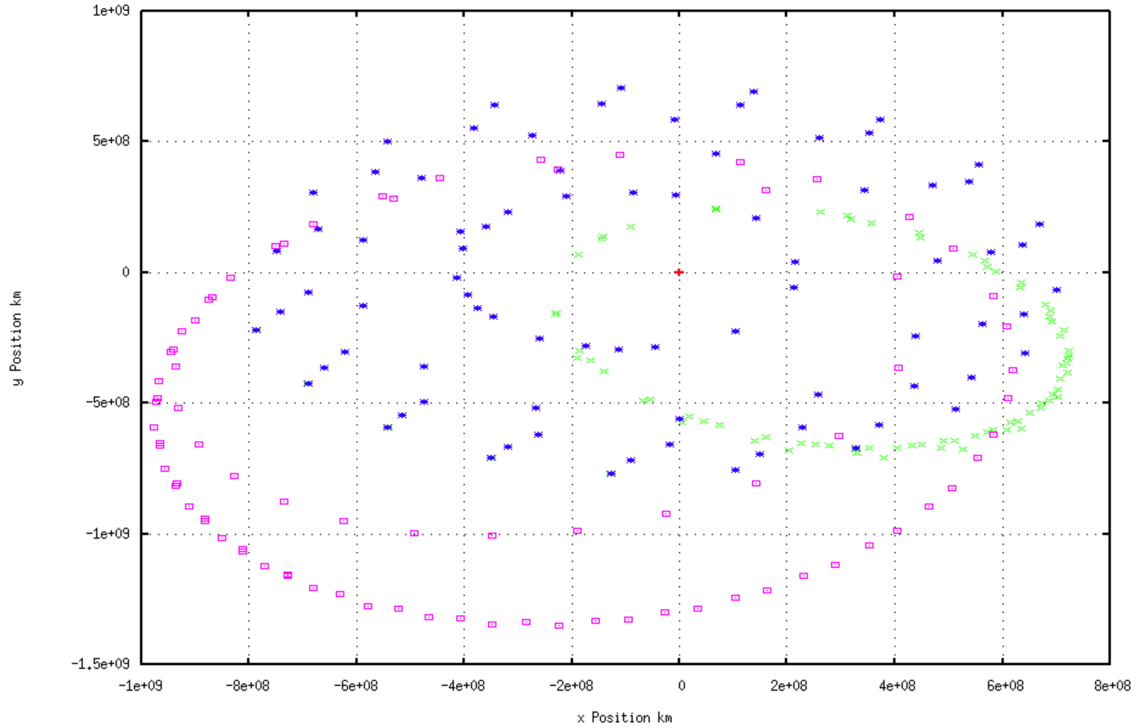

Figure 16. Unstable Orbits of Jupiter's 3 moons: Ganymede, Europa, & Callisto.

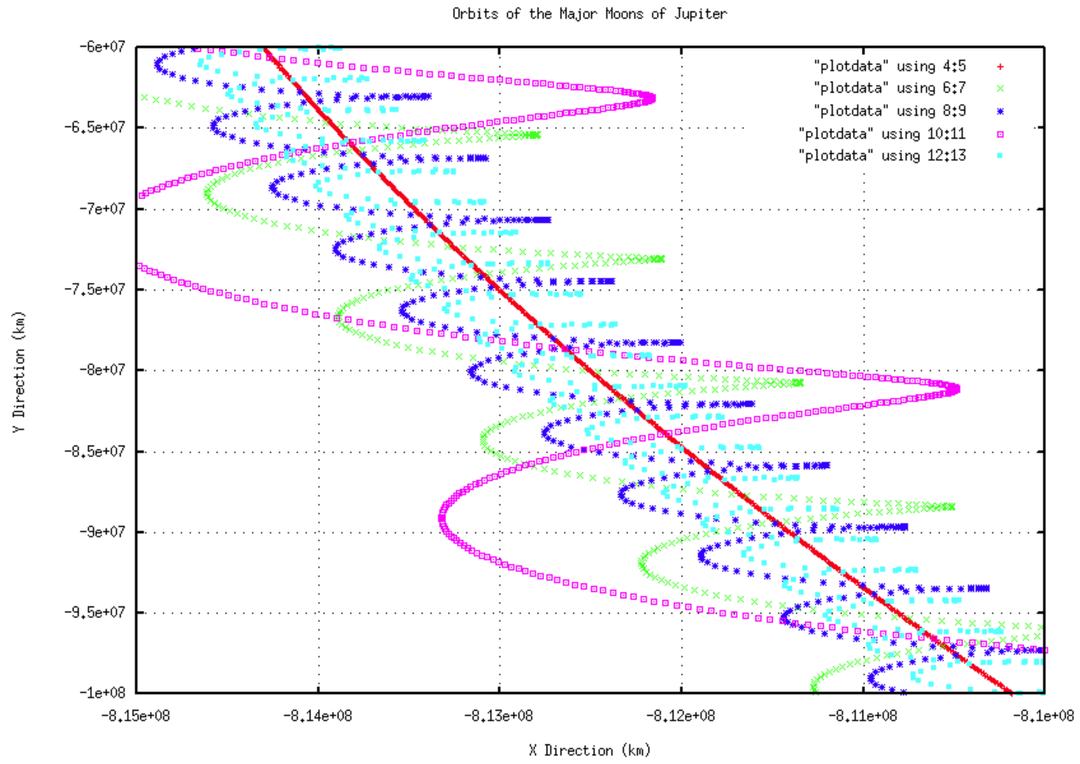

Figure 17. Stable Orbits of Jupiter's 4 moons: Io, Ganymede, Europa, & Callisto.

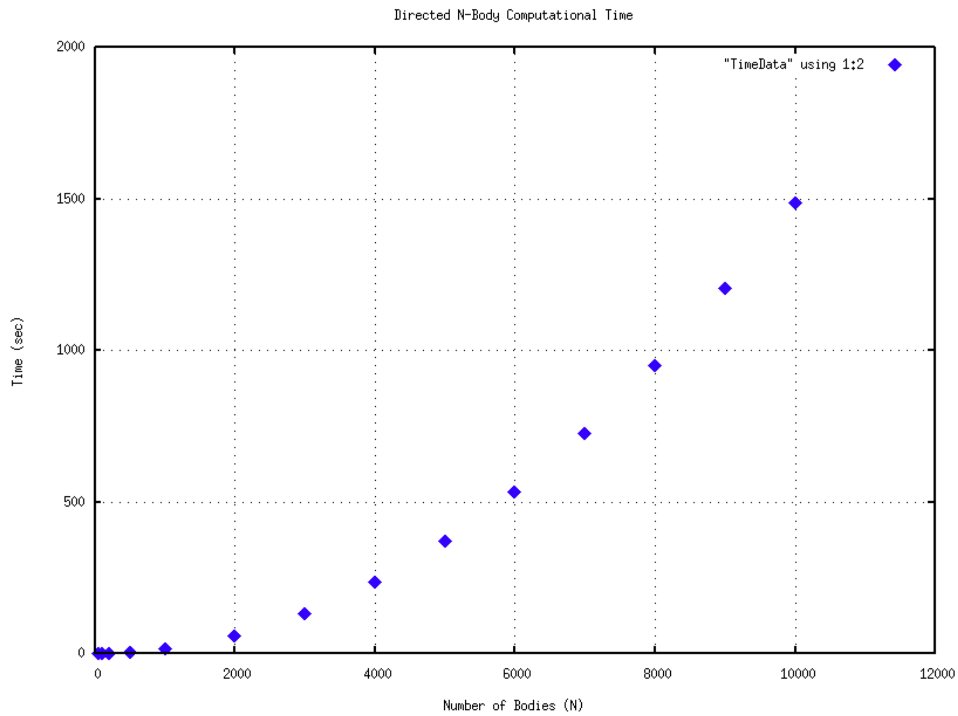

Figure 18. Computational Expense of the N-Body Leapfrog Algorithm.

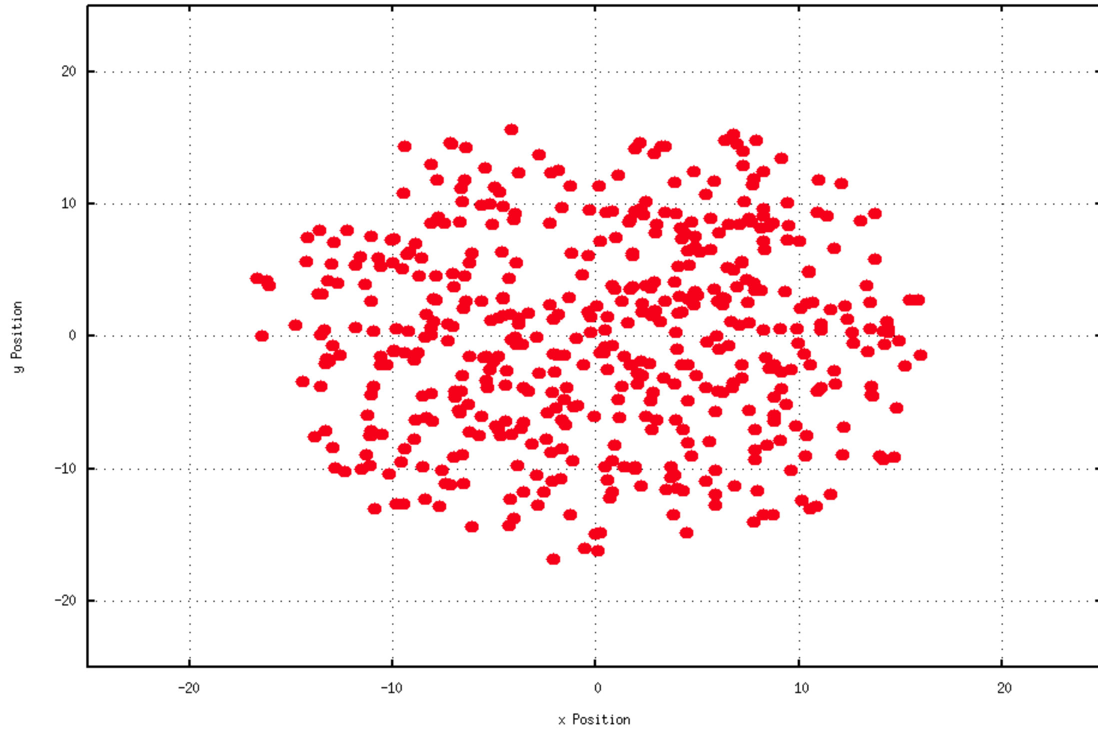

Figure 19. Initial Positions for the 500-Body Problem (x-y Plane).

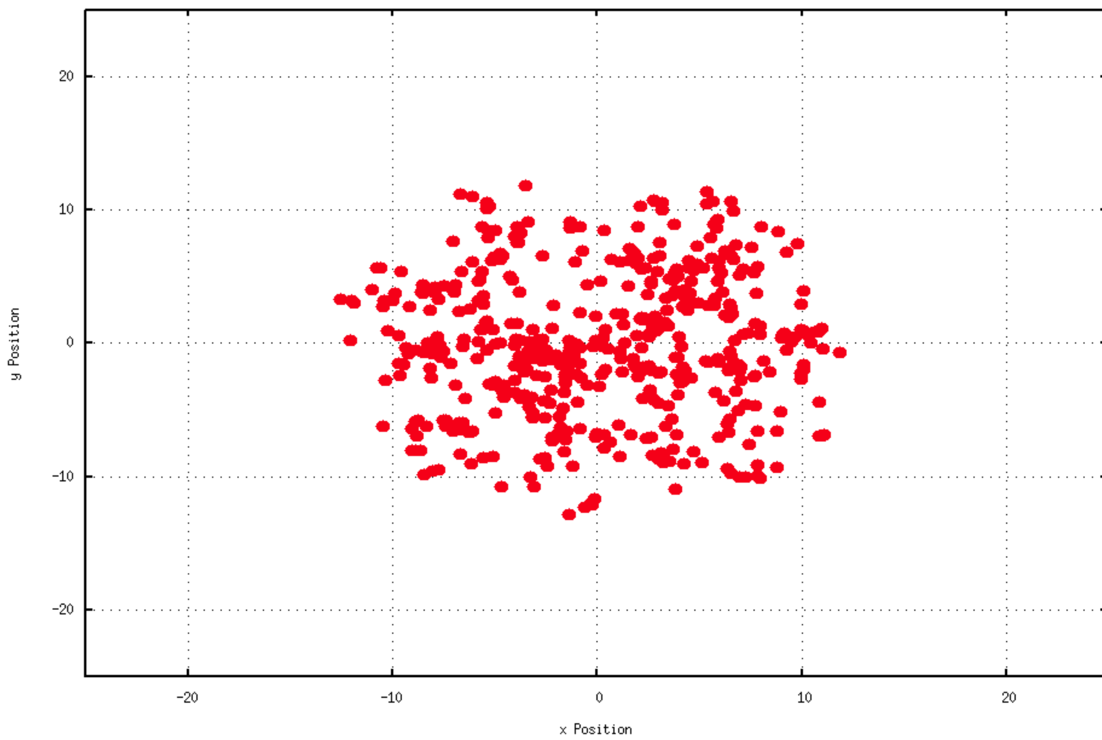

Figure 20. 500 Iterations of the 500-Body Problem (x-y Plane).

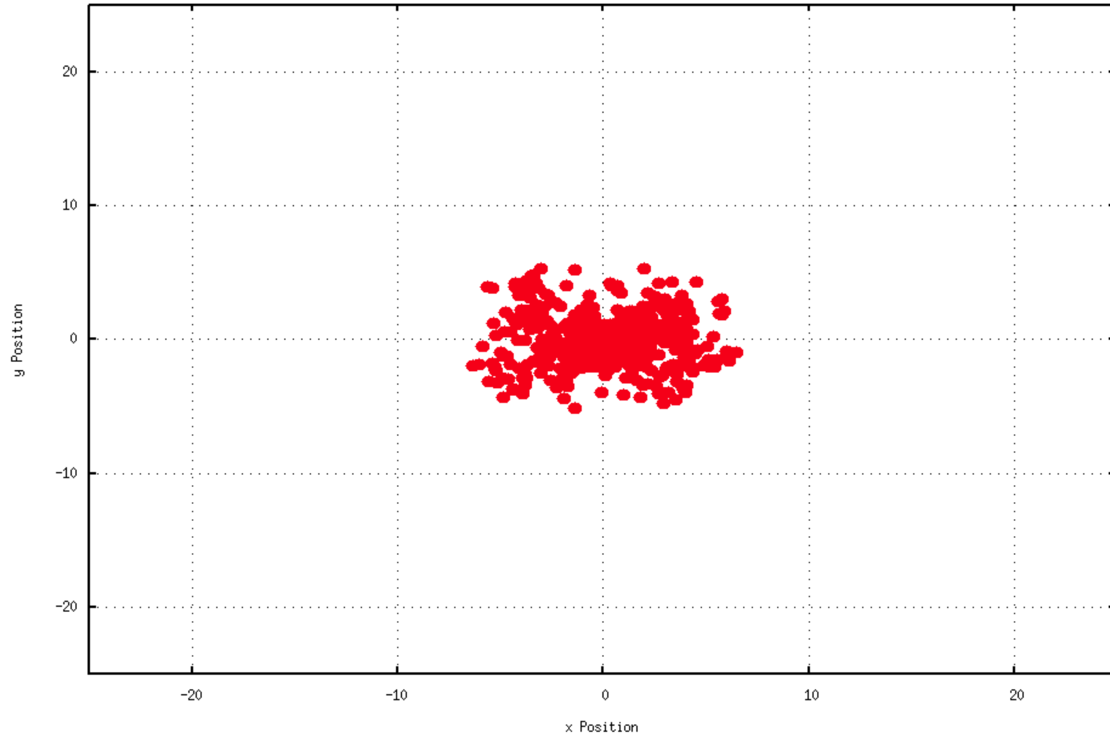

Figure 21. 900 Iterations of the 500-Body Problem (x-y Plane).

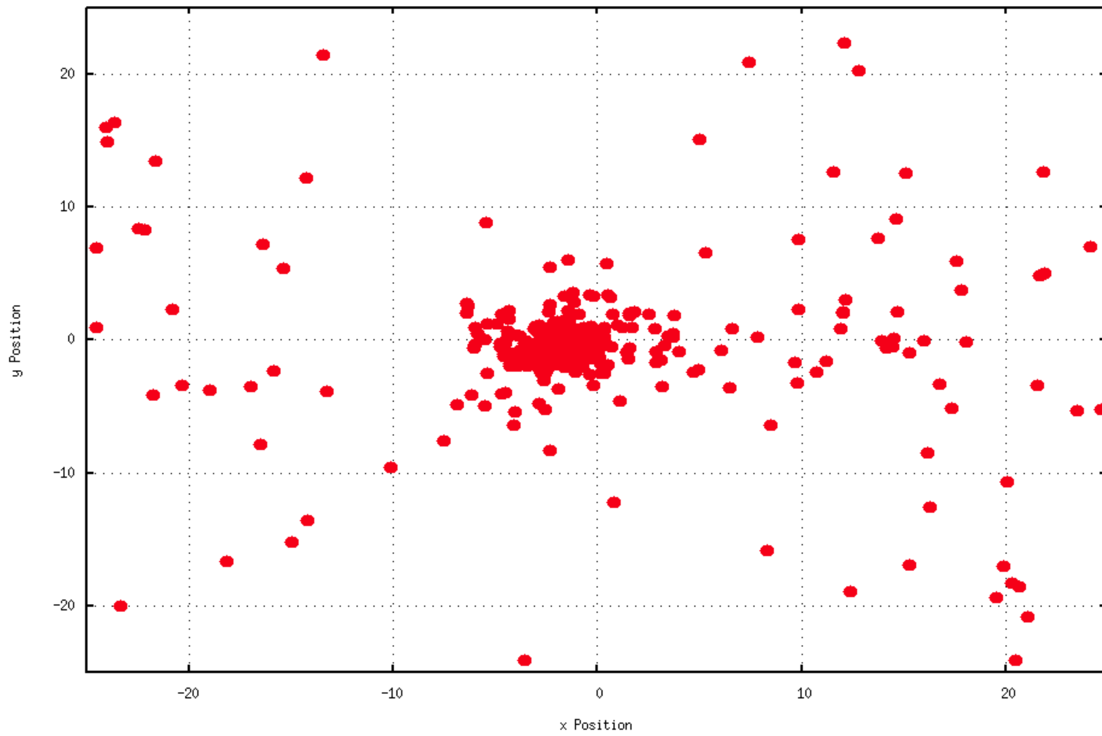

Figure 22. 1,500 Iterations of the 500-Body Problem (x-y Plane).

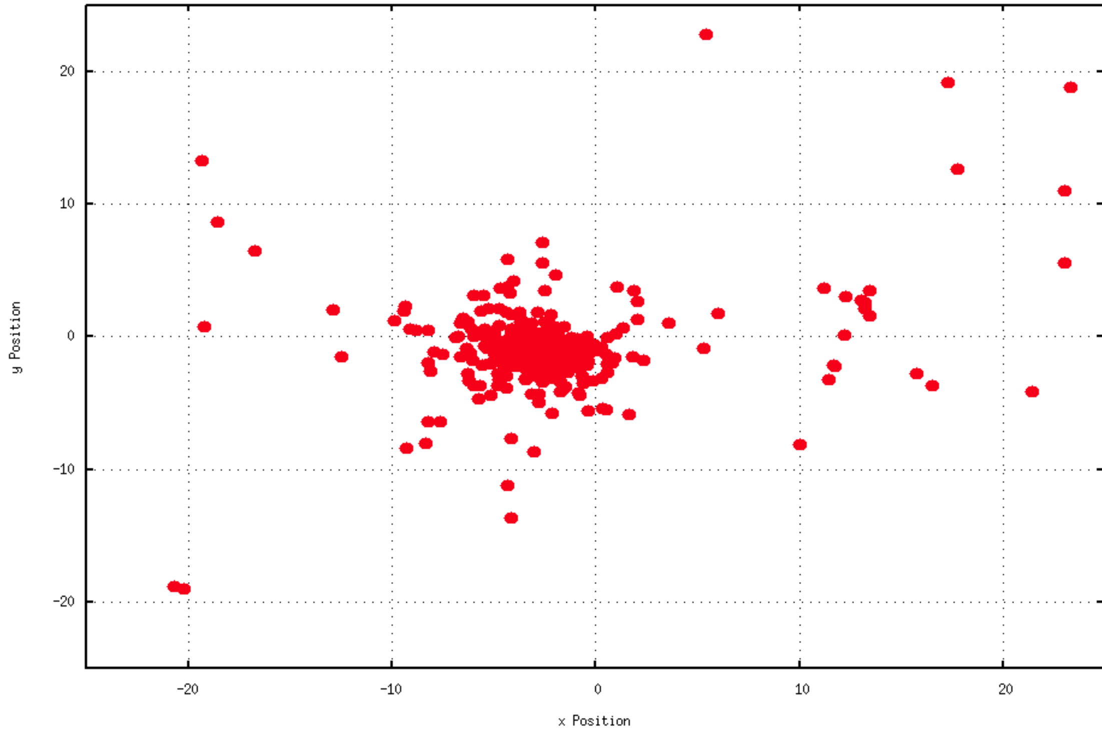

Figure 23. 2,250 Iterations of the 500-Body Problem (x-y Plane).

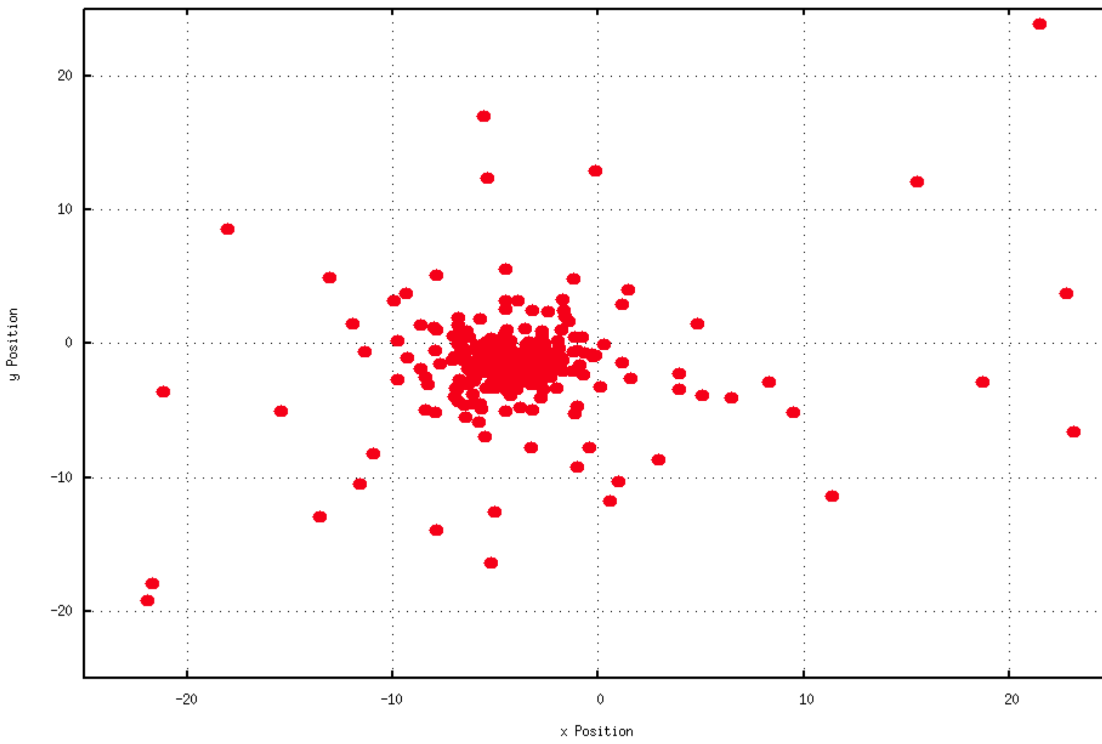

Figure 24. 3,000 Iterations of the 500-Body Problem (x-y Plane).

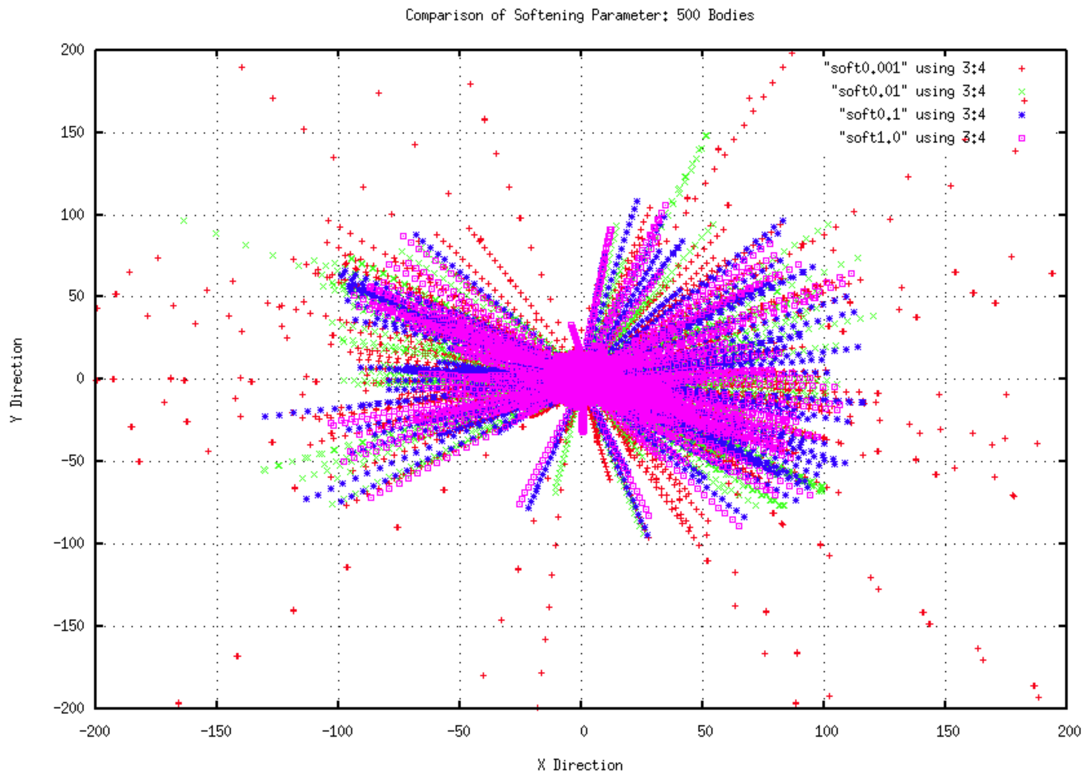

Figure 25. Softening of the 500-Body Problem, 3,000 Iterations (x-y Plane).

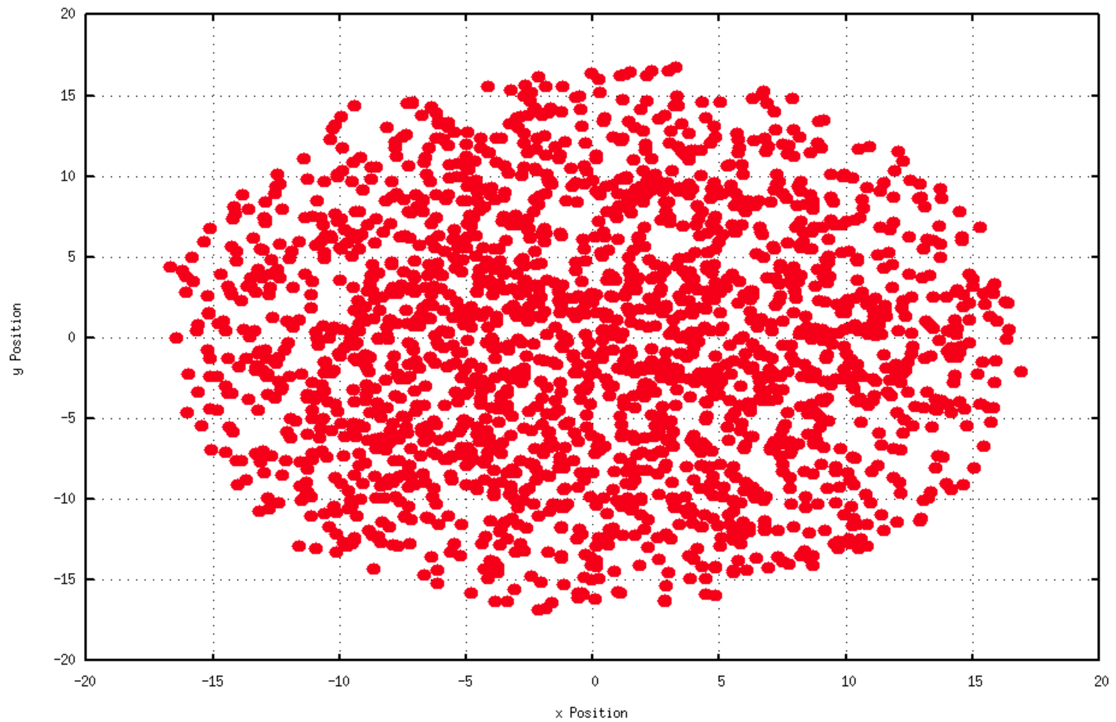

Figure 26. Initial Positions for the 2,000-Body Problem (x-y Plane).

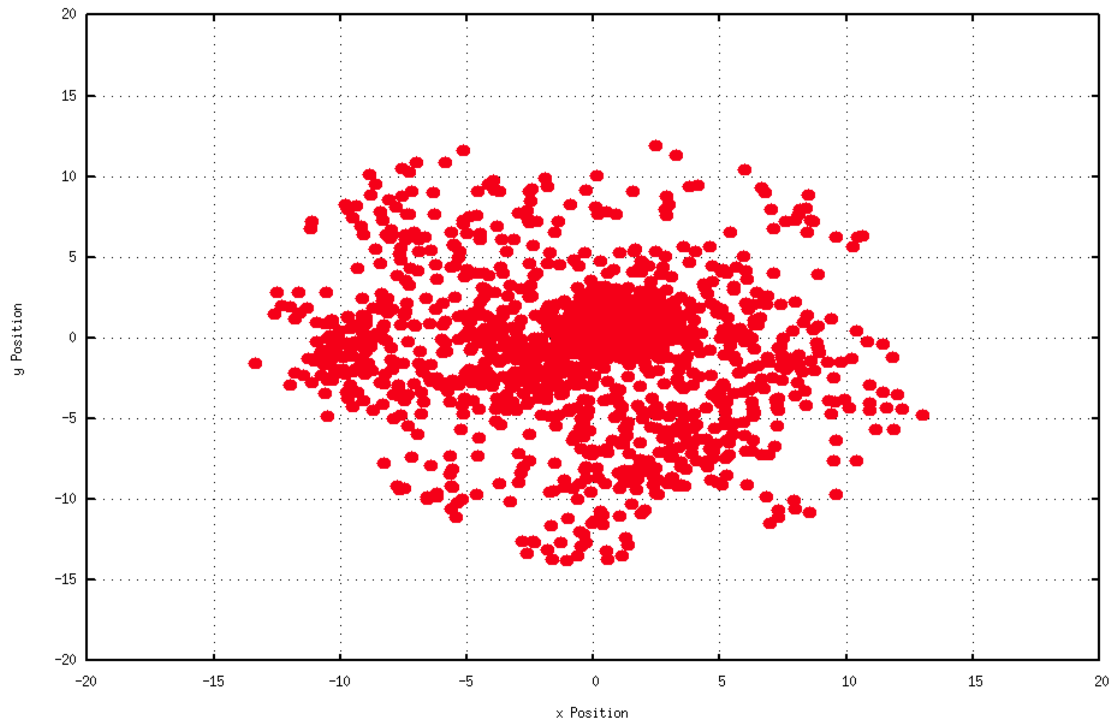

Figure 27. 2,000 Iterations of the 2,000-Body Problem (x-y Plane).

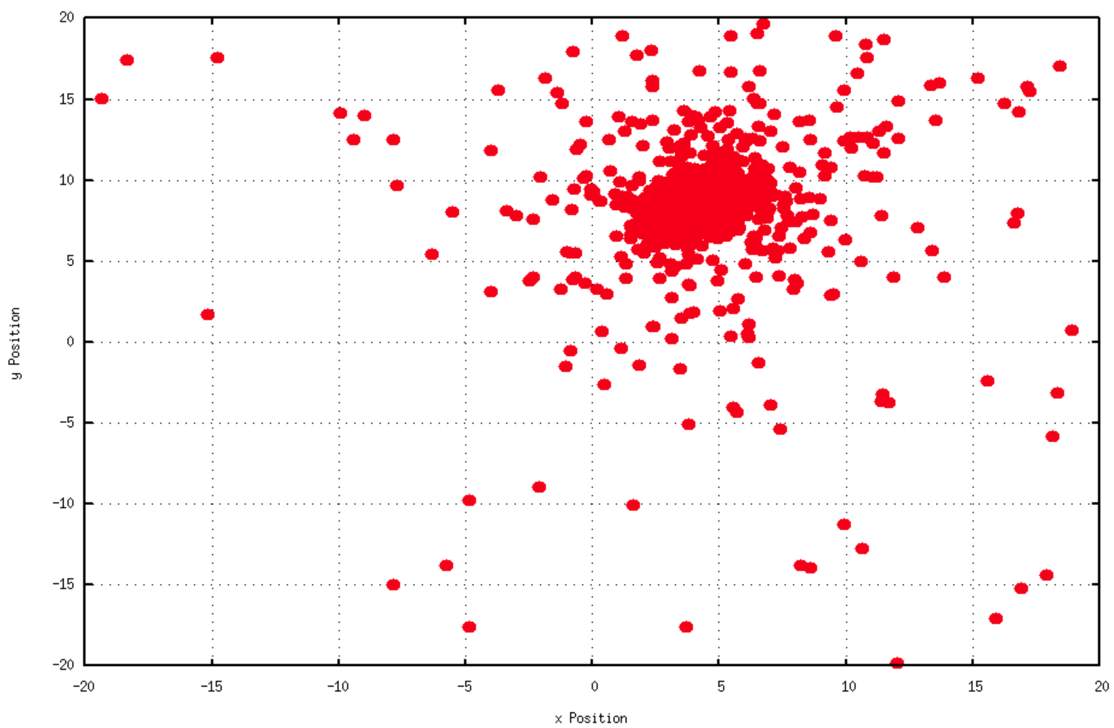

Figure 28. 6,000 Iterations of the 2,000-Body Problem (x-y Plane).

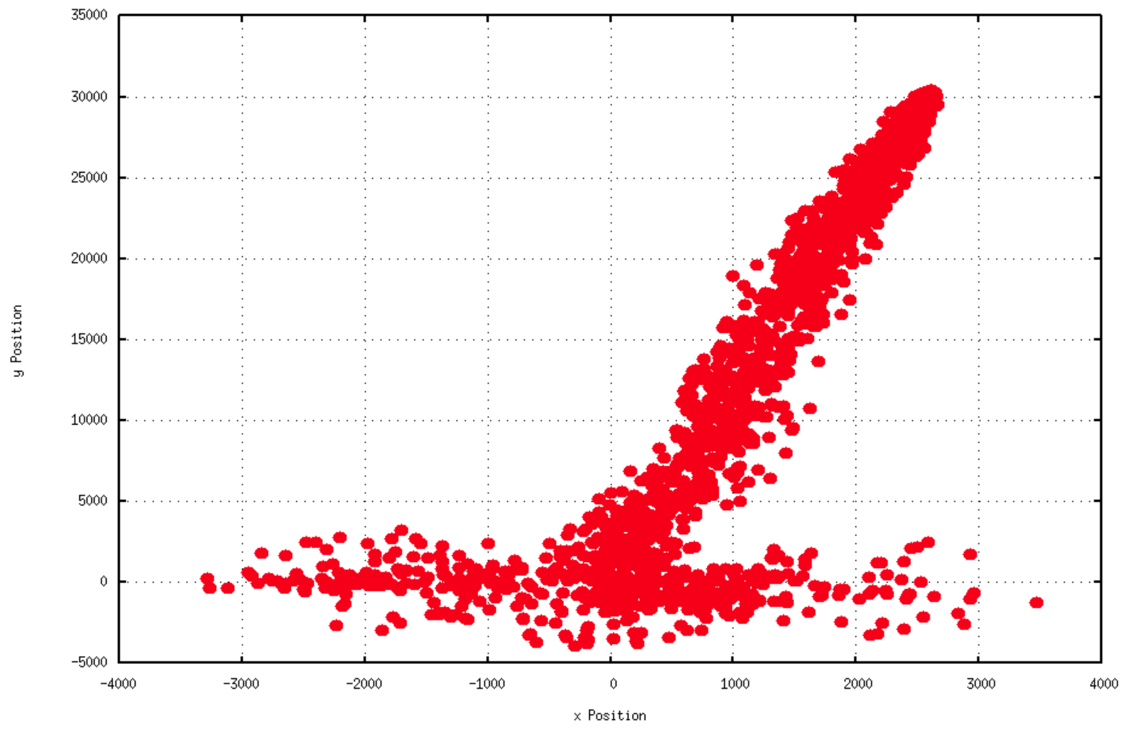

Figure 29. 120,000 Iterations of the 2,000-Body Problem (x-y Plane).

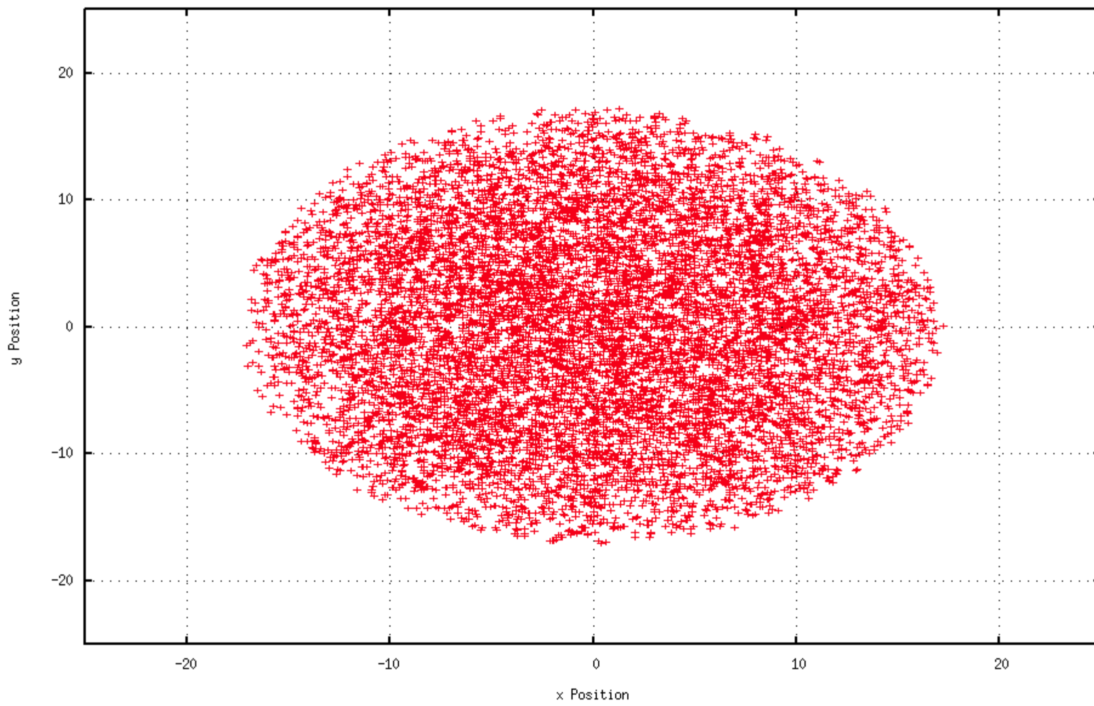

Figure 30. Initial Positions for the 10,000-Body Problem (x-y Plane).

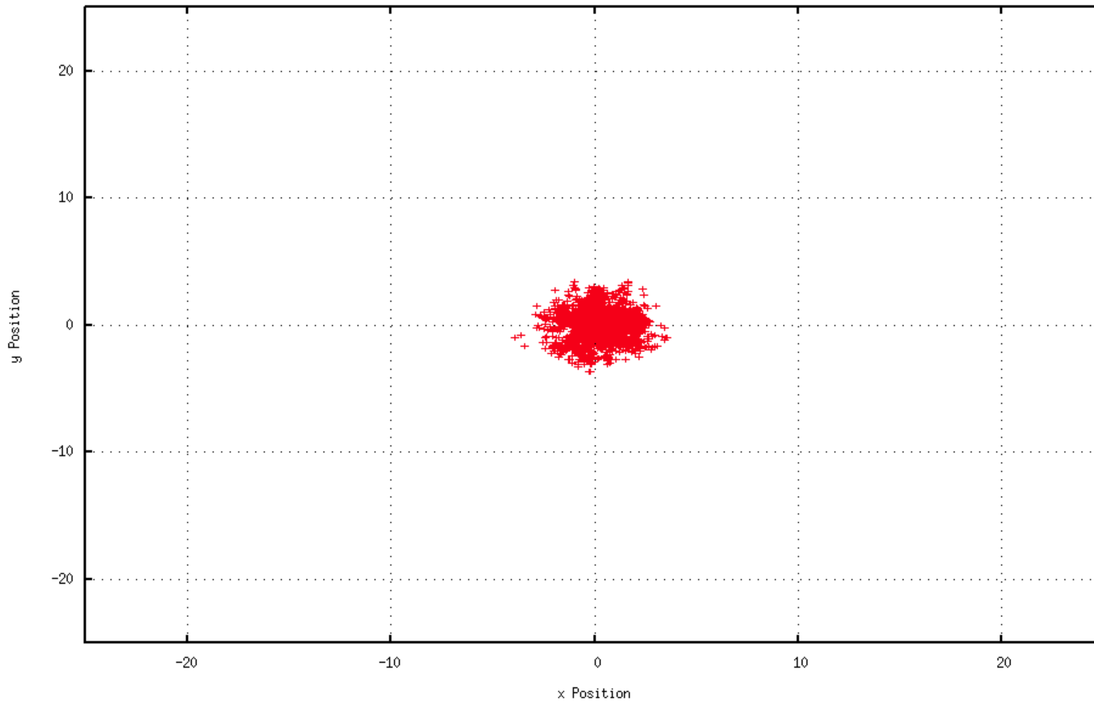

Figure 31. 800 Iterations of the 10,000-Body Problem (x-y Plane).

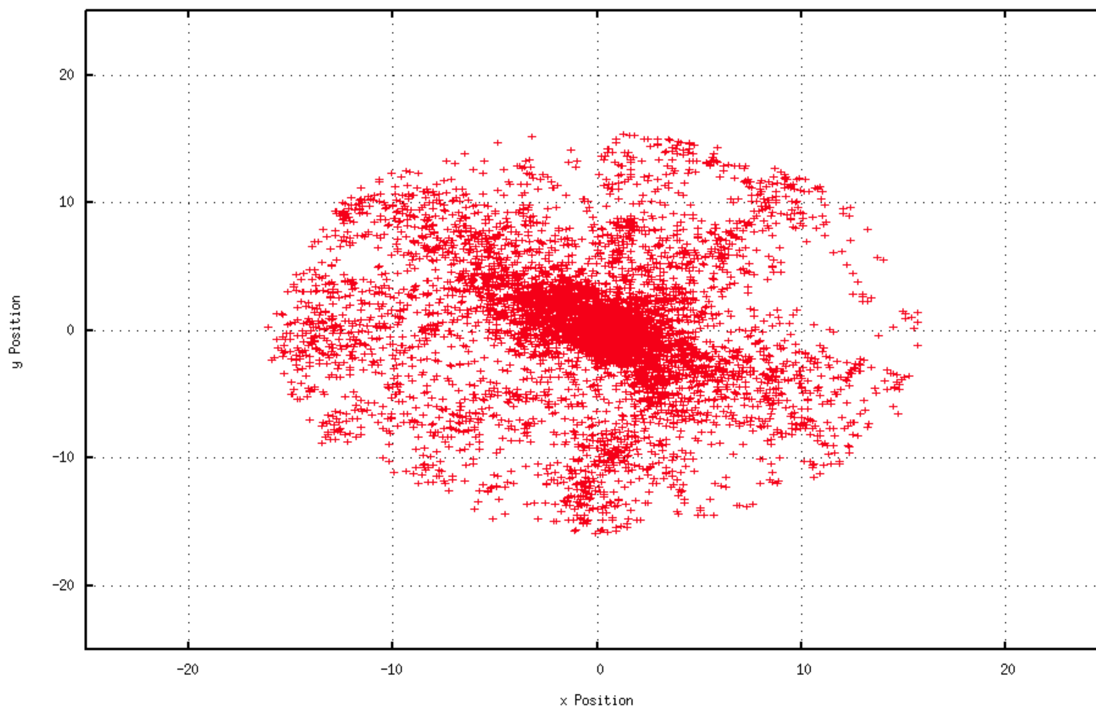

Figure 32. 1,000 Iterations of the 10,000-Body Problem (x-y Plane).

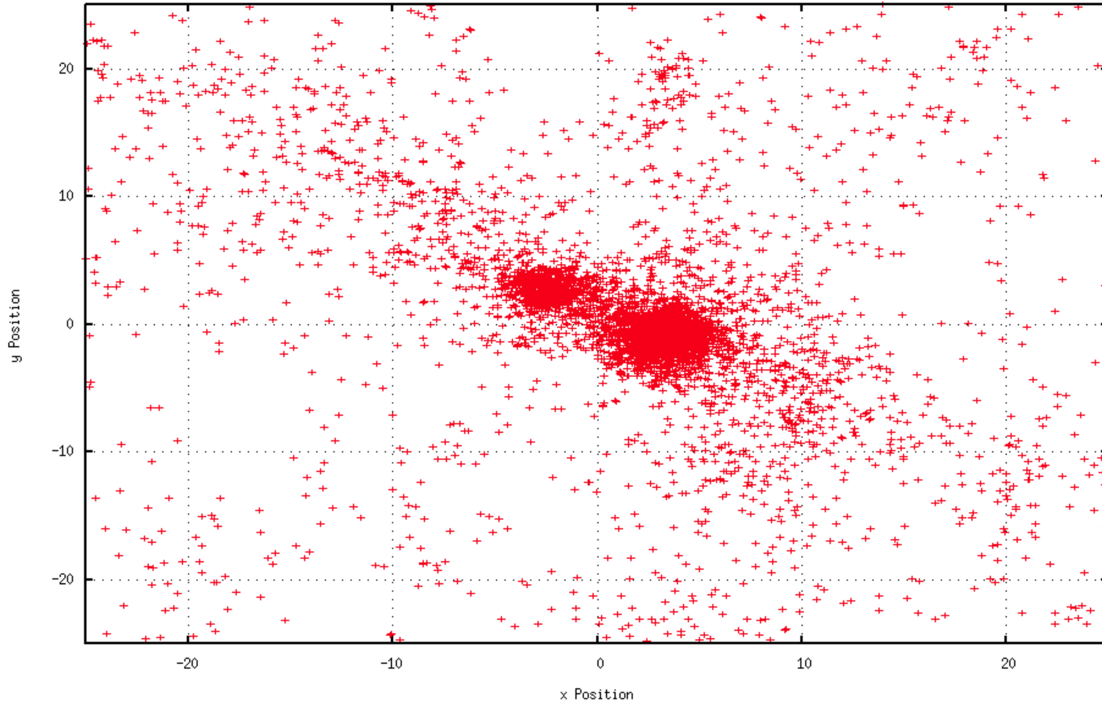

Figure 33. 1,250 Iterations of the 10,000-Body Problem (x-y Plane).

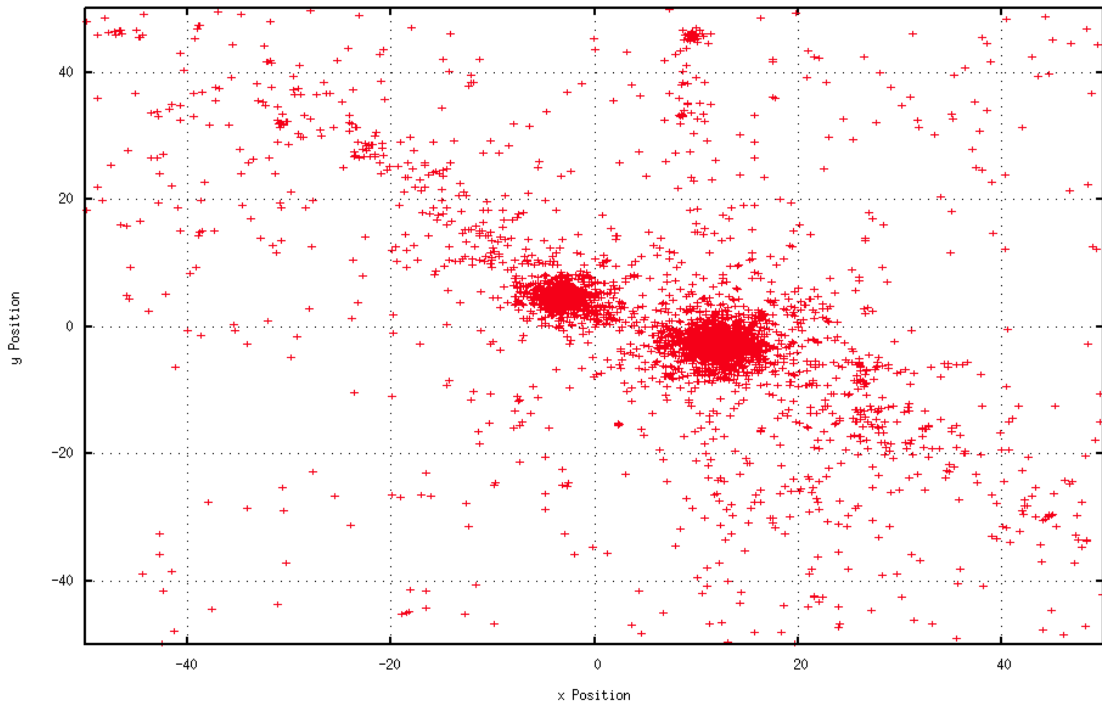

Figure 34. 3,000 Iterations of the 10,000-Body Problem (x-y Plane).

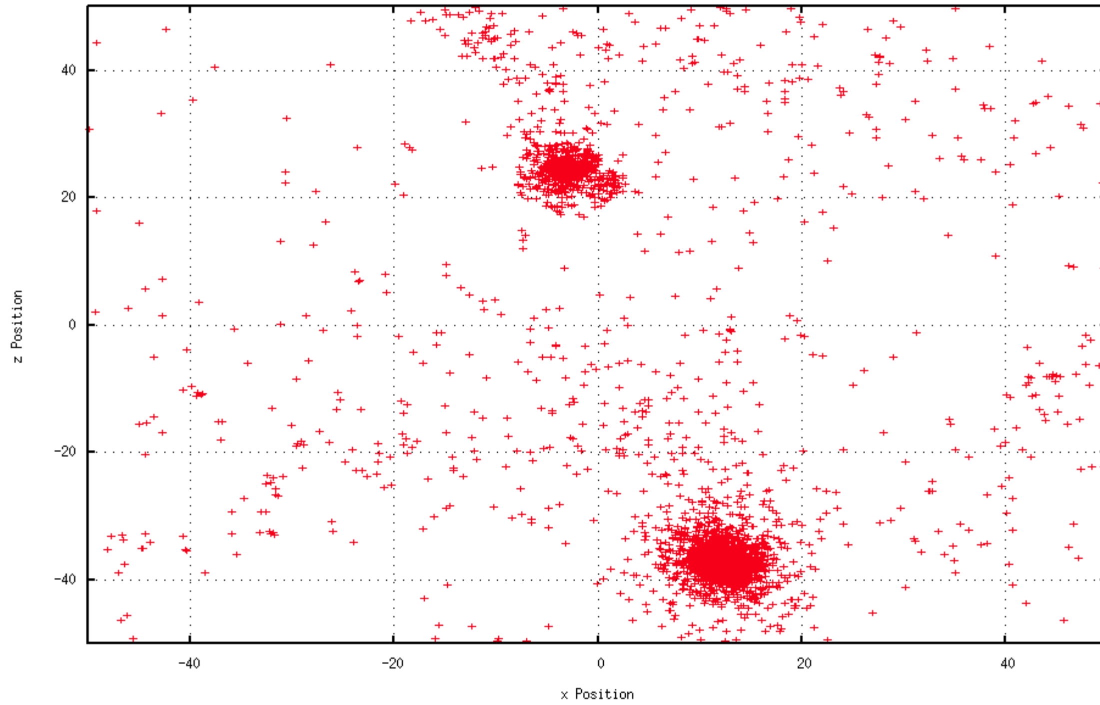

Figure 35. 3,000 Iterations of the 10,000-Body Problem (x-z Plane).

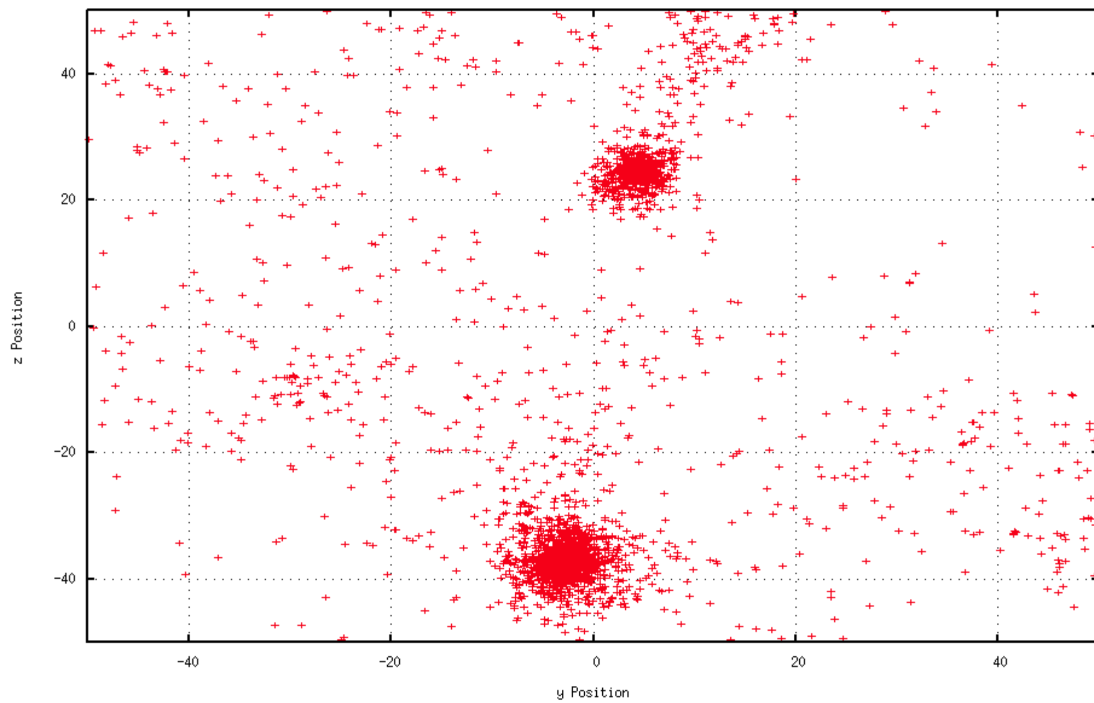

Figure 36. 3,000 Iterations of the 10,000-Body Problem (y-z Plane).

8.0 Appendices

8.1 C-Programs for Validation Problems

```
/* Basic Linear Oscillation Code */
/* Leapfrog Scheme */

#include <stdio.h>
#include <math.h>

int main()
{
     int i, icyl, iprt, iter;
     double dt0, dt, t;
     double c1, pi, k, m, v0, x0, v1, x1;
     double exv, exx, erv, erx;

     /* Computational Interval and Parameters */

     pi=3.14159265;

     iter=200;                 /* Total Number of Integration Loop Iterations */
     icyl=1;                   /* Appoximate Number of Harmonic Oscillations */

     dt0=2.0*pi*icyl/iter;     /* Time Step Size */

     iprt=1;                   /* Print Output Frequency */

     /* Mechanical Parameters: Mass m, Spring Constant k */

     m=1.0;                    /* System Mass */
     k=1.0;                    /* Spring Constant */
     c1=k/m;                   /* Equation Coefficient with Spring Constant */

     /* Initial Data: Velocity v0 and Position x0 */

     v0=0.0;                   /* Initial Velocity */
     x0=1.0;                   /* Initial Position */
     t=0.0;                    /* Initial Time */

     /* Integration of Newton's Second Law */

     for(i=1;i<=iter;i++)      /* Begin Time Integration Loop */
     {

     if(i==1) dt=0.5*dt0;
     if(i>1) dt=dt0;

     v1=v0-(c1*x0)*dt;         /* Compute Velocity at t+dt */
     x1=x0+v1*dt;              /* Compute Position at t+dt */

     exx=cos(t);               /* Compute Exact Position at t+dt */
     exv=-sin(t);              /* Compute Exact Velocity at t+dt */

     erx=fabs(exx-x1);          /* Compute Absolute Error of Position */
     erv=fabs(exv-v1);          /* Compute Absolute Error of Velocity */

     /* Print Solution */

     if(i%iprt==0) {printf("%12i %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e\n",i, t, v1, x1, exv, exx, erv, erx);}

     /* Advance Solution in Time */

     t=t+dt;                  /* Update Time */
     v0=v1;                   /* Update Velocity */
     x0=x1;                   /* Update Position */

     }                        /* End Time Integation Loop */

     return;
}
```

C-Program 1. 1-D Spring Mass System.

```c
/* Basic Linear Oscillation Code */
/* Leap Frog Scheme */

#include <stdio.h>
#include <math.h>

int main()
{
     int i, icyl, iprt, iter;
     double dt0, dt, t;
     double c1, pi, k, m, omg;
     double v10, x10, v1, x1;
     double v20, x20, v2, x2;
     double exx1, exx2, exv1, exv2;
     double erx1, erx2, erv1, erv2;

     /* Computational Interval and Parameters */

     pi=3.14159265;

     iter=2400;              /* Total Number of Integration Loop Iterations */
     icyl=6;                  /* Appoximate Number of Harmonic Oscillations */

     dt0=2.0*pi*icyl/iter;    /* Time Step Size */

     iprt=5;                 /* Print Output Frequency */

     omg=sqrt(3.0);

     /* Mechanical Parameters: Mass m, Spring Constant k, Friction Constant f */

     m=1.0;                   /* System Mass */
     k=1.0;                   /* Spring Constant */
     c1=k/m;                  /* Equation Coefficient with Spring Constant */

     /* Initial Data: Velocity v0 and Position x0 */

     v10=0.0;                  /* Initial Velocity Mass 1 */
     x10=1.0;                  /* Initial Position Mass 1 */
     v20=0.0;                  /* Initial Velocity Mass 2 */
     x20=0.0;                  /* Initial Position Mass 2 */
     t=0.0;                    /* Initial Time */

     /* Integration of Newton's Second Law */

     for(i=1;i<=iter;i++)     /* Begin Time Integration Loop */
     {

     if(i==1) dt=0.5*dt0;
     if(i>1) dt=dt0;

     v1=v10-(c1*x10+c1*(x10-x20))*dt;  /* Compute Velocity of Mass 1 at t+dt */
     x1=x10+v1*dt;                     /* Compute Position of Mass 1 at t+dt */
     v2=v20-(c1*(x20-x10)+c1*x20)*dt;  /* Compute Velocity of Mass 2 at t+dt */
     x2=x20+v2*dt;                     /* Compute Position of Mass 2 at t+dt */

     exx1=0.5*cos(t)+0.5*cos(omg*t);   /* Compute Exact Position x1 at t+dt */
     exx2=0.5*cos(t)-0.5*cos(omg*t);   /* Compute Exact Position x2 at t+dt */

     exv1=-0.5*sin(t)-0.5*omg*sin(omg*t);  /* Compute Exact Velocity v1 at t+dt */
     exv2=-0.5*sin(t)+0.5*omg*sin(omg*t);  /* Compute Exact Velocity v2 at t+dt */

     erx1=fabs(exx1-x1);  /* Compute Absolute Error of Position x1 */
     erx2=fabs(exx2-x2);  /* Compute Absolute Error of Position x2 */

     erv1=fabs(exv1-v1);  /* Compute Absolute Error of Velocity v1 */
     erv2=fabs(exv2-v2);  /* Compute Absolute Error of Velocity v2 */


     /* Print Solution */

     if(i%iprt==0) {printf("%14i %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e %14.6e\n"
                           ,i, t, v1, x1, exv1, exx1, v2, x2, exv2, exx2, erv1, erv2, erx1, erx2);}

     /* Advance Solution in Time */

     t=t+dt;                 /* Update Time */
     v10=v1;                 /* Update Velocity of Mass 1 */
     x10=x1;                 /* Update Position of Mass 1 */
     v20=v2;                 /* Update Velocity of Mass 2 */
     x20=x2;                 /* Update Position of Mass 2 */

     }                      /* End Time Integation Loop */

     return;
}
```

C-Program 2. 2-D Spring Mass System.

```c
/* Basic Nonlinear Terminal Velocity Code */
/* Leap Frog Scheme */

#include <stdio.h>
#include <math.h>

int main()
{
      int i, iprt, iter;
      double dt0, dt, t;
      double a, Cd, c1, g, pi, k, m, r, rho, v0, x0, v1, x1, vt, erv, err;

      /* Computational Interval and Parameters */

      pi=3.14159265;

      iter=20000;                /* Total Number of Integration Loop Iterations */
      dt0=0.01;                  /* Time Step Size: Total Time Interval 200 Seconds */

      iprt=20;                    /* Print Output Frequency */

      /* GBU-28 Aerodynamic Parameters */

      m=4600/2.2;                /* Mass of GBU-28 in Kg */
      Cd=0.75;                   /* Drag Coefficient for Mach 1: See Hoerner Page 16 - 27*/
      r=14.5*2.54/(2*100);       /* GBU-28 Radius in Meters */
      a=2.0*pi*r*r;              /* GBU-28 Reference Area in Meters^2 */
      g=9.81;                    /* Acceleration of Gravity in SI units */
      rho=1.0;                   /* Air Density in SI units Kg / Meters^3 */

      c1=Cd*a*rho/(2*m);         /* Drag Force Constant */

      /* Initial Data: Position x0, y0 and Velocity v0, u0 */

      v0=0.0;                    /* Initial Velocity */
      x0=12192.0;                /* Initial Position in Meters (40,000 ft) */

      vt=2*m*g/(rho*Cd*a);       /* Terminal speed */
      vt=sqrt(vt);               /* Terminal speed */

      t=dt0;                     /* Initial Time */

      /* Integration of Newton's Second Law */

      for(i=1;i<=iter;i++)       /* Begin Time Integration Loop */
      {

      if(i==1) dt=0.5*dt0;
      if(i>1) dt=dt0;

      v1=v0+(c1*v0*v0-g)*dt;     /* Compute Velocity at t+dt */
      x1=x0+v1*dt;               /* Compute Position at t+dt */

      erv=-vt*tanh(g*t/vt);      /* Analytical Solution */
      err=fabs(v1-erv);          /* Absolute Error */

      /* Print Solution (@ Velocity Time) */

      if(i%iprt==0) {printf("%12i %12.6e %12.6e %12.6e %12.6e %12.6e\n",i, t, v1, x1, erv, err);}

      /* Advance Solution in Time */

      t=t+dt;                    /* Update Time (@ Position Time) */

      v0=v1;                     /* Update Velocity */
      x0=x1;                     /* Update Position */

      }                          /* End Time Integration Loop */

      return;
}
```

C-Program 3. 1-D Aerodynamic Drag Problem.

```c
/* Basic 2D Terminal Velocity Code */
/* Leap Frog Scheme */

#include <stdio.h>
#include <math.h>

int main()
{
      int i, iprt, iter;
      double dt0, dt, t;
      double a, Cd, c1, g, pi, k, m, r, rho, v0, x0, v1, x1;
      double u0, u1, y0, y1, spd;

      /* Computational Interval and Parameters */

      pi=3.14159265;

      iter=20000;                /* Total Number of Integration Loop Iterations */
      dt0=0.01;                  /* Time Step Size: Total Time Interval 200 Seconds */

      iprt=20;                    /* Print Output Frequency */

      /* GBU-28 Aerodynamic Parameters */

      m=4600/2.2;                /* Mass of GBU-28 in Kg */
      Cd=0.75;                   /* Drag Coefficient for Mach 1: See Hoerner Page 16 - 27 */
      r=14.5*2.54/(2*100);       /* GBU-28 Radius in Meters */
      a=2.0*pi*r*r;              /* GBU-28 Reference Area in Meters^2 */
      g=9.81;                    /* Acceleration of Gravity in SI units */
      rho=1.0;                   /* Air Density in SI units */

      c1=Cd*a*rho/(2*m);         /* Drag Force Constant */

      /* Initial Data: Position x0, y0 and Velocity v0, u0 */

      v0=268.224;                /* Initial Velocity in the x Direction (600 mi/hr) */
      u0=0.0;                    /* Initial Velocity in the y Direction */
      x0=0.0;                    /* Initial Position in the x Direction in Meters */
      y0=12192.0;                /* Initial Position in the y Direction in Meters (40,000 ft) */

      t=dt0;                     /* Initial Time */

      /* Integration of Newton's Second Law */

      for(i=1;i<=iter;i++)       /* Begin Time Integration Loop */
      {

      if(i==1) dt=0.5*dt0;
      if(i>1) dt=dt0;

      spd=v0*v0+u0*u0;
      spd=pow(spd,0.5);

      v1=v0+(-c1*v0*spd)*dt;      /* Compute Velocity in the x Direction at t+dt */
      x1=x0+v1*dt;                /* Compute Position in the x Direction at t+dt */

      u1=u0+(-c1*u0*spd-g)*dt;    /* Compute Velocity in the y Direction at t+dt */
      y1=y0+u1*dt;                /* Compute Position in the y Direction at t+dt */

      /* Print Solution (@ Velocity Time) */

      if(i%iprt==0) {printf("%12i %12.6e %12.6e %12.6e %12.6e %12.6e %12.6e\n",i, t, x1, y1, v1, u1, -spd);}

      /* Advance Solution in Time */

      t=t+dt;                    /* Update Time (@ Position Time) */

      v0=v1;                     /* Update Velocity in the x Direction */
      x0=x1;                     /* Update Position in the x Direction */

      u0=u1;                     /* Update Velocity in the y Direction */
      y0=y1;                     /* Update Position in the y Direction */

      }                          /* End Time Integration Loop */

      return;
}
```

C-Program 4. 2-D Aerodynamic Drag Problem.

8.2 C-Programs for N-Body Simulations

```
/* Basic Solar System Code for Leap Frog Check */
/* Including Moon Data (Earth or Jupiter) */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include<time.h>
#include<string.h>

int randint();

int main()
{
     int i, f, j, k, iter;
     double c1, c2, c3;
     double ax, ay, az, g;
     double dt, dt0, err, rn, t, nrm, pi;
     double m[20], x[20], y[20], z[20];
     double x0[20], y0[20], z0[20];
     double u[20], v[20], w[20];
     double u0[20], v0[20], w0[20];

     /* Zero Arrays */

     for(j=0;j<=19;j++){
     x[j]=0.0;
     y[j]=0.0;
     z[j]=0.0;
     x0[j]=0.0;
     y0[j]=0.0;
     z0[j]=0.0;
     u[j]=0.0;
     v[j]=0.0;
     w[j]=0.0;
     u0[j]=0.0;
     v0[j]=0.0;
     w0[j]=0.0;
     m[j]=0.0;
     }

     /*  Initialize Physical Values for the Masses */

     m[1] = 27066267.52;  /* Sun */
     m[2] = 4.49;         /* Mercury */
     m[3] = 66.27;        /* Venus */
     m[4] = 81.28;        /* Earth */
     m[5] = 1.00;         /* Moon */
     m[6] = 8.73;         /* Mars */
     m[7] = 25834.81;     /* Jupiter */
     m[8] = 2.01;            /* Jupiter Moon 1 2.01 */
     m[9] = 0.65;            /* Jupiter Moon 2 0.65 */
     m[10] = 1.47;           /* Jupiter Moon 3 1.47 */
     m[11] = 1.21;           /* Jupiter Moon 4 1.21 */
     m[12] = 7735.20;     /* Saturn */
     m[13] = 1181.55;     /* Uranus */
     m[14] = 1393.80;     /* Neptune */
     m[15] = 0.17;        /* Pluto */

     /*  Specify Initial Position and Velocity for the N-Bodies */

     /*  Position of the Sun m1 x[1], y[1], z[1]  */
     /*  Velocity of the Sun m1 u[1], v[1], w[1]  */

     x0[1]=1.57835543;    /* position in the x direction of mass m1 */
     y0[1]=0.0;           /* position in the y direction of mass m1 */
     z0[1]=0.0;           /* position in the z direction of mass m1 */

     u0[1]=0.0;               /* velocity in the x direction of mass m1 */
     v0[1]=0.01521836*3600;   /* velocity in the y direction of mass m1 */
     w0[1]=0.0;               /* velocity in the z direction of mass m1 */
```

C-Program 5 Part 1. Solar System Problem.

```
/*  Position of Mercury m2 x[2], y[2], z[2]  */
/*  Velocity of Mercury m2 u[2], v[2], w[2]  */

x0[2]=-69299572.31;     /* position in the x direction of mass m2 */
y0[2]=0.0;              /* position in the y direction of mass m2 */
z0[2]=8508917.56;       /* position in the z direction of mass m2 */

u0[2]=0.0;              /* velocity in the x direction of mass m2 */
v0[2]=-38.86*3600;      /* velocity in the y direction of mass m2 */
w0[2]=0.0;              /* velocity in the z direction of mass m2 */

/*  Position of Venus m3 x[3], y[3], z[3] */
/*  Velocity of Venus m3 u[3], v[3], w[3] */

x0[3]=-108940000.0;     /* position in the x direction of mass m3 */
y0[3]=0.0;              /* position in the y direction of mass m3 */
z0[3]=0.0;              /* position in the z direction of mass m3 */

u0[3]=0.0;              /* velocity in the x direction of mass m3 */
v0[3]=-34.79*3600;      /* velocity in the y direction of mass m3 */
w0[3]=0.0;              /* velocity in the z direction of mass m3 */

/*  Position of the Earth m4 x[4], y[4], z[4]  */
/*  Velocity of the Earth m4 u[4], v[4], w[4]  */

x0[4]=-152100000.0;    /* position in the x direction of mass m4 */
y0[4]=0.0;             /* position in the y direction of mass m4 */
z0[4]=0.0;             /* position in the z direction of mass m4 */

u0[4]=0.0;             /* velocity in the x direction of mass m4 */
v0[4]=-29.29*3600;     /* velocity in the y direction of mass m4 */
w0[4]=0.0;             /* velocity in the z direction of mass m4 */

/*  Position of the Moon m5 x[5], y[5], z[5] */
/*  Velocity of the Moon m5 u[5], v[5], w[5] */

x0[5]=x0[4]-405500.0;    /* position in the x direction of mass m5 */
y0[5]=0.0;               /* position in the y direction of mass m5 */
z0[5]=0.0;               /* position in the z direction of mass m5 */

u0[5]=0.0;               /* velocity in the x direction of mass m5 */
v0[5]=v0[4]-0.964*3600; /* velocity in the y direction of mass m5 */
w0[5]=0.0;               /* velocity in the z direction of mass m5 */

/*  Position of Mars m6 x[6], y[6], z[6]  */
/*  Velocity of Mars m6 u[6], v[6], w[6]  */

x0[6]=-249230000.0;  /* position in the x direction of mass m6 */
y0[6]=0.0;           /* position in the y direction of mass m6 */
z0[6]=0.0;           /* position in the z direction of mass m6 */

u0[6]=0.0;           /* velocity in the x direction of mass m6 */
v0[6]=-21.97*3600;   /* velocity in the y direction of mass m6 */
w0[6]=0.0;           /* velocity in the z direction of mass m6 */

/*  Position of Jupiter m7 x[7], x[7], z[7]  */
/*  Velocity of Jupiter m7 u[7], v[7], w[7]  */

x0[7]=-816620000.0;    /* position in the x direction of mass m7 */
y0[7]=0.0;             /* position in the y direction of mass m7 */
z0[7]=0.0;             /* position in the z direction of mass m7 */

u0[7]=0.0;             /* velocity in the x direction of mass m7 */
v0[7]=-12.44*3600;     /* velocity in the y direction of mass m7 */
w0[7]=0.0;             /* velocity in the z direction of mass m7 */

/*  Position of Jupiter Moon #1 m8 x[8], y[8], z[8] */
/*  Velocity of Jupiter Moon #1 m8 u[8], v[8], w[8] */

x0[8]=x0[7]-1070000.0; /* position in the x direction of mass m8 */
y0[8]=0.0;              /* position in the y direction of mass m8 */
z0[8]=0.0;              /* position in the z direction of mass m8 */

u0[8]=0.0;              /* velocity in the x direction of mass m8 */
v0[8]=v0[7]-39169.0;    /* velocity in the y direction of mass m8 */
w0[8]=0.0;              /* velocity in the z direction of mass m8 */
```

C-Program 5 Part 2. Solar System Problem.

```
/*  Position of Jupiter Moon #2 m9 x[9], y[9], z[9]  */
/*  Velocity of Jupiter Moon #2 m9 u[9], v[9], w[9]  */

x0[9]=x0[7]-670900.0;  /* position in the x direction of mass m9 */
y0[9]=0.0;             /* position in the y direction of mass m9 */
z0[9]=0.0;             /* position in the z direction of mass m9 */

u0[9]=0.0;             /* velocity in the x direction of mass m9 */
v0[9]=v0[7]-49464.0;   /* velocity in the y direction of mass m9 */
w0[9]=0.0;             /* velocity in the z direction of mass m9 */

/*  Position of Jupiter Moon 3 m10 x[10], y[10], z[10] */
/*  Velocity of Jupiter Moon 3 m10 u[10], v[10], w[10] */

x0[10]=x0[7]-1883000.0; /* position in the x direction of mass m10 */
y0[10]=0.0;             /* position in the y direction of mass m10 */
z0[10]=0.0;             /* position in the z direction of mass m10 */

u0[10]=0.0;             /* velocity in the x direction of mass m10 */
v0[10]=v0[7]-29556.0;   /* velocity in the y direction of mass m10 */
w0[10]=0.0;             /* velocity in the z direction of mass m10 */

/*  Position of Jupiter Moon 4 m11 x[11], y[11], z[11]  */
/*  Velocity of Jupiter Moon 4 m11 u[11], v[11], w[11]  */

x0[11]=x0[7]-421600.0;  /* position in the x direction of mass m11 */
y0[11]=0.0;             /* position in the y direction of mass m11 */
z0[11]=0.0;             /* position in the z direction of mass m11 */

u0[11]=0.0;             /* velocity in the x direction of mass m11 */
v0[11]=v0[7]-62424.0;   /* velocity in the y direction of mass m11 */
w0[11]=0.0;             /* velocity in the z direction of mass m11 */

/*  Position of Saturn m12 x[12], x[12], x[12]  */
/*  Velocity of Saturn m12 u[12], v[12], w[12]  */

x0[12]=-1514500000.0;  /* position in the x direction of mass m12 */
y0[12]=0.0;            /* position in the y direction of mass m12 */
z0[12]=0.0;            /* position in the z direction of mass m12 */

u0[12]=0.0;            /* velocity in the x direction of mass m12 */
v0[12]=-9.09*3600;     /* velocity in the y direction of mass m12 */
w0[12]=0.0;            /* velocity in the z direction of mass m12 */

/*  Position of Uranus m13 x[13], x[13], x[13] */
/*  Velocity of Uranus m13 v[13], v[13], v[13] */

x0[13]=-3003620000.0;   /* position in the x direction of mass m13 */
y0[13]=0.0;             /* position in the y direction of mass m13 */
z0[13]=0.0;             /* position in the z direction of mass m13 */

u0[13]=0.0;             /* velocity in the x direction of mass m13 */
v0[13]=-6.49*3600;      /* velocity in the y direction of mass m13 */
w0[13]=0.0;             /* velocity in the z direction of mass m13 */

/*  Position of Neptune m14 x[14], x[14], x[14]  */
/*  Velocity of Nepture m14 v[14], v[14], v[14]  */

x0[14]=-4545670000.0;   /* position in the x direction of mass m14 */
y0[14]=0.0;             /* position in the y direction of mass m14 */
z0[14]=0.0;             /* position in the z direction of mass m14 */

u0[14]=0.0;             /* velocity in the x direction of mass m14 */
v0[14]=-5.37*3600;      /* velocity in the y direction of mass m14 */
w0[14]=0.0;             /* velocity in the z direction of mass m14 */

/*  Position of Pluto m15 x[43], x[44], x[45] */
/*  Velocity of Pluto m15 v[43], v[44], v[45] */

x0[15]=-7047587322.65;   /* position in the x direction of mass m15 */
y0[15]=0.0;              /* position in the y direction of mass m15 */
z0[15]=2176202264.16;    /* position in the z direction of mass m15 */

u0[15]=0.0;              /* velocity in the x direction of mass m15 */
v0[15]=-3.71*3600;       /* velocity in the y direction of mass m15 */
w0[15]=0.0;              /* velocity in the z direction of mass m15 */
```

C-Program 5 Part 3. Solar System Problem.

```
    pi=3.14159265;

    iter=400000;
    dt0=0.01;  /* dt0=5.0 for ~ 75 minute step size */
    t=0.0;

    f=3;    /* f=1 for Inner Planets; f=2 for Outer Planets f=3 for Jupiter's Moons */

    g=63460000000.0;

    err=0.001;

    for(k=1;k<=iter;k++){

    if(k==1){dt=0.5*dt0;}
    if(k>1){dt=dt0;}

    for(i=1;i<=15;i++){

    ax=0.0;
    ay=0.0;
    az=0.0;

    for(j=1;j<=15;j++){
    if(j!=i){
    nrm=(x0[i]-x0[j])*(x0[i]-x0[j])+(y0[i]-y0[j])*(y0[i]-y0[j])+(z0[i]-z0[j])*(z0[i]-z0[j]);
    nrm=pow(nrm,1.5);
    ax=ax+m[j]*(x0[i]-x0[j])/(nrm+err);
    ay=ay+m[j]*(y0[i]-y0[j])/(nrm+err);
    az=az+m[j]*(z0[i]-z0[j])/(nrm+err);
    }
    }

    u[i]=u0[i]-ax*g*dt;
    v[i]=v0[i]-ay*g*dt;
    w[i]=w0[i]-az*g*dt;

    x[i]=x0[i]+u[i]*dt;
    y[i]=y0[i]+v[i]*dt;
    z[i]=z0[i]+w[i]*dt;

    u0[i]=u[i];
    v0[i]=v[i];
    w0[i]=w[i];

    x0[i]=x[i];
    y0[i]=y[i];
    z0[i]=z[i];

    /*    if(k==1) {printf("%12i %14.6e %14.6e %14.6e %14.6e\n", i, t, x[i], y[i], z[i]);}     */
   /*     if(k%25==0) {printf("%12i %14.6e %14.6e %14.6e %14.6e\n", i, t, x[i], y[i], z[i]);}  */

    }

    c1=k;
    c2=1000;
    c3=fmod(c1,c2);

    if (c3 == 0.0){

        if(f==1){printf("%12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g\n"
                        ,t, x[1], y[1], x[2], y[2], x[3], y[3], x[4], y[4], x[6], y[6]);}
        if(f==2){printf("%12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g\n"
                        ,t, x[1], y[1], x[7], y[7], x[12], y[12], x[13], y[13], x[14], y[14], x[15], y[15]);}
        if(f==3){printf("%12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g %12.8g\n"
                        ,t, x[1], y[1], x[7], y[7], x[8], y[8], x[9], y[9], x[10], y[10], x[11], y[11]);}
    }

    t=t+dt;

    }

    return 0;
}
```

C-Program 5 Part 4. Solar System Problem.

```c
/* Basic Initial Data Code of Cluster Formation */
/* Pseudo-Random Number Generator */

#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include<time.h>
#include<string.h>

int randint();

int main()
{
      int i, j, k, ic, iter, kc;
      double ax, ay, az, g;
      double dm, dt, dt0, err, rn, t, nrm, pi;
      double cmx, cmy, cmz;
      double r[4], s[4];
      double x[10000], y[10000], z[10000];
      double x0[10000], y0[10000], z0[10000];
      double u[10000], v[10000], w[10000];
      double u0[10000], v0[10000], w0[10000];

      for(i=1;i<=999;i++){
      u0[i]=0.0;
      v0[i]=0.0;
      w0[i]=0.0;
      }

      pi=3.14159265;
      kc=1;

      iter=4000;
      dt0=0.1;
      t=0.0;

      g=1.0;

      err=0.01;

      for(k=1;k<=970;k++){

      for(j=1;j<=6;j++){

      i=randint();
      rn=(i*1.0)/(2.15*1.0e9);

      if(j==1){r[1]=rn*10*pow(3.0,0.5);}
      if(j==2){r[2]=rn*10*pow(3.0,0.5);}
      if(j==3){r[3]=rn*10*pow(3.0,0.5);}

      if(j==4)
      {
      if(rn>=0.5){s[1]=1.0;}
      if(rn<=0.5){s[1]=-1.0;}
      }

      if(j==5)
      {
      if(rn>=0.5){s[2]=1.0;}
      if(rn<=0.5){s[2]=-1.0;}
      }
      if(j==6)
      {
      if(rn>=0.5){s[3]=1.0;}
      if(rn<=0.5){s[3]=-1.0;}
      }

      }
```

C-Program 6 Part 1. Arbitrary N-Body Solver.

```
x0[kc]=s[1]*r[1];
y0[kc]=s[2]*r[2];
z0[kc]=s[3]*r[3];

u0[kc]=0.0;
v0[kc]=0.0;
w0[kc]=0.0;

nrm=x0[kc]*x0[kc]+y0[kc]*y0[kc]+z0[kc]*z0[kc];
nrm=pow(nrm, 0.5);
ic=0;
rn=pow(300.0,0.5);
if(nrm>rn) {
x0[kc]=0.0;
y0[kc]=0.0;
z0[kc]=0.0;
ic=1;}

if(ic==0){kc=kc+1;}
}

kc=kc-1;
dm=1.0/(1.0*kc);

cmx=0.0;
cmy=0.0;
cmz=0.0;

for(k=1;k<=kc;k++){
    cmx=+x0[k];
    cmy=+y0[k];
    cmz=+z0[k];
 }

cmx=cmx/(1.0*kc);
cmy=cmy/(1.0*kc);
cmz=cmz/(1.0*kc);

for(k=1;k<=kc;k++){
    x0[k]=x0[k]-cmx;
    y0[k]=y0[k]-cmy;
    z0[k]=z0[k]-cmz;
}

for(k=1;k<=iter;k++){

if(k==1){dt=0.5*dt0;}
if(k>1){dt=dt0;}

for(i=1;i<=kc;i++){

ax=0.0;
ay=0.0;
az=0.0;

for(j=1;j<=kc;j++){
if(j!=i){
nrm=(x0[i]-x0[j])*(x0[i]-x0[j])+(y0[i]-y0[j])*(y0[i]-y0[j])+(z0[i]-z0[j])*(z0[i]-z0[j]);
nrm=pow(nrm,1.5);
ax=ax+dm*(x0[i]-x0[j])/(nrm+err);
ay=ay+dm*(y0[i]-y0[j])/(nrm+err);
az=az+dm*(z0[i]-z0[j])/(nrm+err);
}
}

u[i]=u0[i]-ax*g*dt;
v[i]=v0[i]-ay*g*dt;
w[i]=w0[i]-az*g*dt;

x[i]=x0[i]+u[i]*dt;
y[i]=y0[i]+v[i]*dt;
z[i]=z0[i]+w[i]*dt;

u0[i]=u[i];
v0[i]=v[i];
w0[i]=w[i];
```

C-Program 6 Part 2. Arbitrary N-Body Solver.

```
        x0[i]=x[i];
        y0[i]=y[i];
        z0[i]=z[i];

            if(k==1) {printf("%12i %14.6e %14.6e %14.6e %14.6e\n", i, t, x[i], y[i], z[i]);}
            if(k%50==0) {printf("%12i %14.6e %14.6e %14.6e %14.6e\n", i, t, x[i], y[i], z[i]);}

        }

        t=t+dt;

        }

        return 0;
}

int randint()
{
    return rand();
}
```

C-Program 6 Part 3. Arbitrary N-Body Solver.

9.0 Original Contributions and Acknowledgements

For this project a number of original contributions have been made. Three moderately large clusters were studied, 500, 2,000 and 10,000 body problems. The main contributions were the experimental computational results of these simulations. The initial development of a cluster was shown in the 500-Body problem. The 2,000-Body problem produced the same initial behavior as the 500-Body, and then developed a jet, which may be related to a numerical instability. Finally, for the 10,000-Body problem, it was shown that the cluster evolves into two separate globular mass distributions within the system, which was an unexpected result.

We would like to acknowledge Roy Baty for his assistance through out this project, as well as John Armijo. Also we would like to thank Mr. Lee Goodwin and Mrs. Diane Medford for supporting the Supercomputing Challenge at Los Alamos High School. Finally, the team would like to thank David Kratzer, of Los Alamos National Laboratory, for his effort in sponsoring the Supercomputing Challenge.