CHAOTIC SCATTERING IN THE 2^{ND} POST-NEWTONIAN ORDER GRAVITATIONAL THREE-BODY PROBLEM

By

David E. Tanner

A senior thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Bachelor of Science

Department of Physics and Astronomy

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

DEPARTMENT APPROVAL

of a senior thesis submitted by

David E. Tanner

This thesis has been reviewed by the research advisor, research coordinator, and department chair and has been found to be satisfactory.

_____

Date David Neilsen, Advisor

_____

Date Eric Hintz, Research Coordinator

_____

Date Ross Spencer, Chair

ABSTRACT

CHAOTIC SCATTERING IN THE 2ND POST-NEWTONIAN ORDER GRAVITATIONAL THREE-BODY PROBLEM

David E. Tanner

Department of Physics and Astronomy

Bachelor of Science

The three-body problem is explored in general relativity using the second order post-Newtonian approximation. The results are compared to Newtonian gravity. The three bodies are set up as a binary system with an incoming body. The system is evolved through time for a large two dimensional space of initial values. Several properties for both the Newtonian and relativistic systems are mapped onto the initial-value space and analyzed. The two sets of maps show regions of similarity and contrast. The relativistic system's chaotic behavior diverges most significantly from the Newtonian system when the three bodies undergo relativistic interactions.

ACKNOWLEDGEMENTS

A huge thanks goes to my thesis advisor, Dr. David Neilsen. He helped me to numerically simulate planetary motion, collect data there from, and then make sense of it all. He also greatly helped me write my whole thesis.

I would also like to thank Dr. Eric Hirschmann who helped me along my long and bumpy journey through general relativity. Without his patience and after-school tutoring, Christoffel symbols, Γ, would just be a game of hangman.

Thank you especially to my wife for sharing in my excitement as I learned and understood principles in general relativity. She is a great woman indeed for being able to make sense of hypersurface-orthogonal vector fields.

And thanks, finally, to my two good friends MarylouX and Marylou4 of the Foulton Supercomputing Laboratory; their high computing power made finishing my simulations on time possible.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

N-body problems have been widely studied and are a versatile window into physical chaos. More specifically, the gravitational three-body problem has received much attention because of its simple setup and its application to astrophysical systems such as the solar system. It has been studied using different numbers of dimensions, different equations for gravity and different constraints on the system. Most work on the three-body problem has been done in the context of Newtonian gravity. The present work studies chaos in general relativity, Einstein's theory of gravitation, using the second order post-Newtonian approximation (hereafter abbreviated PPN2). This study sets up congruent experiments—the same except for the equations of gravity. Newton's equations of gravity are compared with Einstein's PPN2. This experiment brings to light differences and similarities between the equations especially with regard to their chaotic behavior.

1.2 Chaos

In physics, chaos denotes a dynamic system for which the outcome is highly sensitive to the initial conditions. This sensitivity makes long-term prediction impossible, as small uncertainties in the initial state lead to very large effects later on. The exponential sensitivity to initial conditions can be quantified using the Liapunov exponent. Let $\phi_1(t)$ and $\phi_2(t)$ represent two possible evolutions of a physical system (i.e. two solutions to the system's equations of motion). Let $\Delta\phi(t)$ represent the difference between the two solutions ($\phi_2(t) - \phi_1(t)$). As time goes on, if the solutions grow closer together (behave the same) $\Delta\phi(t)$ will approach zero. On the other hand, if the solutions consistently follow different paths, $\Delta\phi(t)$ approaches infinity (differing systems behave more and more differently). In the following expression

$$\Delta\phi(t) \sim Ke^{\lambda t}$$

$\Delta\phi(t)$ approaches zero or infinity depending on whether λ is positive or negative; K is a constant. The variable λ is the Liapunov exponent; if it is negative then long-term motion is non-chaotic (in fact it converges to a common solution) but if it is positive then the long-term motion is chaotic. Thus, systems are chaotic if different solutions diverge at an exponential rate.

To illustrate the impossibility of long-term predictions for chaotic systems, consider two states specified by 0.999999999999999 and 1. These states differ by only $1x10^{-15}$, but to a computer these are the same floating-point number. But to a chaotic system, these may represent initial conditions, or intermediate values, varied enough to produce exponentially differing outcomes. It does not take much time for a numerical inaccuracy to blow up into noticeable error. For example, transforming the above equation into

$\Delta\phi(t) = \Delta\phi(0)e^{\lambda t}$ and solving for how long it takes (t) for a numerical error ($\Delta\phi(t) = 1x10^{-15}$) to grow to order one ($\Delta\phi(t) = 1$) using $\lambda = ½$.

$$1 = 1x10^{-15}e^{2t}$$

It only takes t = 17.27 units of time for the "negligible" error to grow to order one. For this reason, running the same chaotic computer simulation twice can produce different results even if the initial conditions are meant to be the same.

Although chaos denotes unpredictability, it should not connote random. Buried inside the mysteries of chaos, there can be seen beautiful patterns. One example of a pattern born of chaos is a fractal. A fractal is a pattern which repeats itself on finer and finer magnifications. Figure 1.1 shows three images of the Mandelbrot [1] at different magnifications; note the pattern is ever-complex on finer and finer scales and has some repeating features. The Mandelbrot Set is defined as the set of all complex numbers such that $f_c^n(0) < \infty$, where $f_c^n(z)$ is the nth iteration of $f_c(z) = z^2 + c$; i.e. $f_c^3(0) = \left((0)^2 + c\right)^2 + c$. The black areas in Figure 1.1 belong to the Mandelbrot set while the colored regions are color coated according to how fast they diverge to infinity.

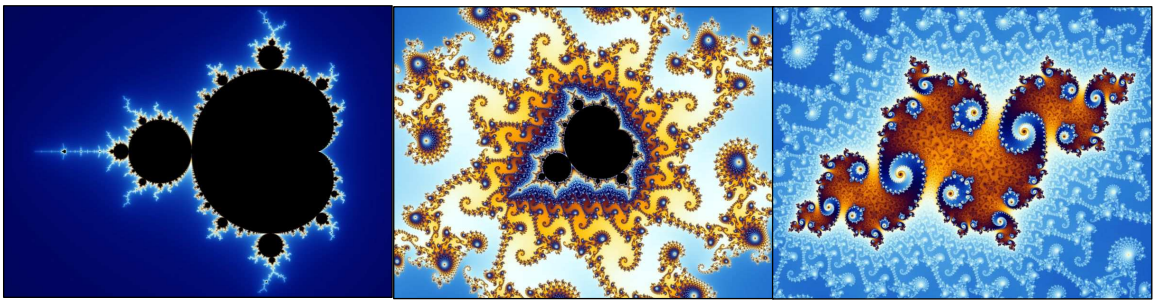

Figure1.1 Mandelbrot Set (zoom=1; zoom=2x10^5; zoom=6x10^{10})

Notice the chaotic border shown in the left-most figure. By zooming in almost a million times on the border, the center image is found. And zooming in almost a million times

more produces the far right image. These beautiful patterns seem to exist on ever finer and finer scales. Similar features can also be found in the chaotic gravitational three-body problem, discussed in Section 1.3. Though they are both chaotic and therefore extremely sensitive to initial conditions, they are not random—they feature beautiful patterns.

1.3 Three-body Problem

Gravitating bodies has been a very popular system for exploring chaos. The gravitational three-body problem is a very simple chaotic system; three masses orbit each other under their mutual gravitational attraction. They are in a vacuum (no drag) and the only forces they feel are gravitational forces. Figure 1.2, taken from [2], illustrates how three bodies may evolve in each other's gravitational field. The two body gravitational problem, on

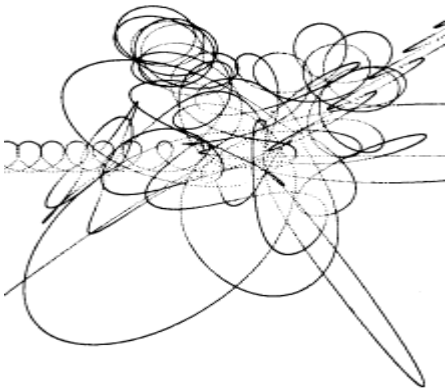

Figure 1.2 Three-body chaos in center-of-mass reference frame

the other hand, is quite simple to solve analytically. Newton was the first person to do so, which provided a new understanding of Kepler's laws. The gravitational two-body problem has a closed-form solution—a solution which can be written as a function. The solutions are conic sections: an elliptical orbit if their energies are negative, a parabola if their energies are zero or a hyperbola if their energies are positive. The solution is

deterministic and cannot behave chaotically.

The three-body problem, however, is much more complicated. It does not have a general analytic solution. Interestingly, over 100 years ago Poincaré discovered that the three-body problem is chaotic [3]. Because the n≥3-body problems are chaotic, orbits in our solar system can only be predicted accurately on the order of millions of years [4]. For general three-body problems, the equations of motion must be solved numerically.

Much research has already been done on the three-body problem. Chicone et al [5] setup a binary system which was driven by obliquely incident gravitational radiation and was dampened by radiating its own gravitational energy—it behaved similar to a driven dampened harmonic oscillator. Their driven dampened stellar system was made to behave both smoothly and chaotically. Hock [6] studied the Eüler problem (two fixed bodies and one orbiting body), sometimes called the restricted three-body problem, and showed that for Newtonian gravity, the solution was non-chaotic, whereas implementing equation of general relativity produced a chaotic system. Others have allowed three bodies to move freely in one dimension. Hietarinta et al [7] showed that a Newtonian three-body system, in one dimension, still behaves chaotically. Burnell et al [8] studied the chaotic behavior of the one-dimensional problem using general relativity. The Newtonian and general relativity gravity comparison is furthered in this study by looking at the three-dimensional three-body problem.

1.4 Post-Newtonian Gravity

When most think of gravity, they think of Sir Isaac Newton. Newton's Universal Law of Gravitation says that the force of gravitational attraction (*F*) between any two bodies is

$$\vec{F} = G\frac{m_1 m_2}{r^3}\vec{r}$$

where G is the universal gravitational constant, m_1 and m_2 are the masses of bodies 1 and 2, and r is the distance vector between the masses. According to Newton, the force between two masses is related only to the distance between them. For the most part, he was right…

About one hundred years ago Albert Einstein formulated a new—more complex—model of gravity called general relativity. To contrast their complexity, the average student will learn about Newton's Law of Universal Gravitation in middle school, whereas a typical astrophysicist does not learn Einstein's theory of general relativity until graduate school. This difference is due to the mathematical complexity of general relativity, which uses tensor equations on curved manifolds. Einstein showed that Newton's Law of Gravity it a simple approximation of general relativity. Since that time, mathematicians and physicists have extracted correctional terms to study, in a perturbative manner, Newtonian-like systems in general relativity.

Einstein models space and time as a curved manifold. Energy and mass curve space-time and this curvature is manifest as a "gravitational force." In the case of the three-body problem, the gravitational fields of the three bodies would determine the curvature. The bodies then move along the natural curves (known as geodesics) of space-time. Solving the full equations of general relativity for three-bodies in three dimensions would be far too complex. To avoid an exact analytical solution of such complex equations, this paper explores the three-body problem using PPN2. A second-order perturbative approximation is acceptable for weak fields such as a modeled solar system. This approximation generates only ordinary differential equations, whereas the full

equations of general relativity are partial differential equations (which are harder to numerically integrate).

1.5 Significance of Comparison

The general three-body problem has previously been explored using Newtonian gravity. I explore the same system using PPN2 and compare the results. An elegant study of he Newtonian three-body problem was conducted by Boyd and McMillan in 1993 [9]. I reproduce some of their results using Newtonian gravity then perform the same experiments using Einstein's PPN2 gravity. It is proposed that the two systems will differ the most when the bodies undergo relativistic interactions (high velocities and close approaches) as this is what PPN2 takes into account. This experiment will give insight into the chaotic behavior of general relativity.

Chapter 2

Problem Setup

2.1 Equations of Motion

The first step in simulating planetary motion is establishing the equations of motion which will govern the behavior of the planetary bodies. Two sets of equations are implemented in this study: one for Newtonian gravity and one for PPN2. The equations are presented as a system of ordinary differential equations derived from their respective Hamiltonians. The Hamiltonian for Newtonian gravity is

$$H = \sum_a m_a + \tfrac{1}{2}\sum_a \frac{p_a^2}{m_a} - \tfrac{1}{2}G\sum_a \sum_{b\neq a} \frac{m_a m_b}{r_{ab}}.$$

The first term sums the rest energy of the masses, the second term sums the kinetic energy of all the masses, and the third term sums the gravitational potential energy between each pair of masses. The Hamiltonian for Einstein's PPN2 was derived by Schäfer [10]:

$$H = \sum_a m_a + \tfrac{1}{2}\sum_a \frac{p_a^2}{m_a} - \tfrac{1}{2}G\sum_a\sum_{b\neq a}\frac{m_a m_b}{r_{ab}} - \tfrac{1}{8}\sum_a m_a\left(\frac{p_a^2}{m_a^2}\right)^2$$

$$-\tfrac{1}{4}G\sum_a\sum_{b\neq a}\frac{m_a m_b}{r_{ab}}\left(6\frac{p_a^2}{m_a^2} - 7\frac{\vec{p}_a\cdot\vec{p}_b}{m_a m_b} - \frac{(\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_b)}{m_a m_b}\right) + \tfrac{1}{2}G^2\sum_a\sum_{b\neq a}\sum_{c\neq a}\frac{m_a m_b m_c}{r_{ab}r_{ac}} + \tfrac{1}{16}\sum_a m_a\left(\frac{p_a^2}{m_a^2}\right)^3$$

$$+\tfrac{1}{16}G\sum_a\sum_{b\neq a}\frac{m_a m_b}{r_{ab}}\left[\begin{array}{l} 10\left(\frac{p_a^2}{m_a^2}\right)^2 - 11\frac{p_a^2 p_b^2}{m_a^2 m_b^2} - 2\frac{(\vec{p}_a\cdot\vec{p}_b)^2}{m_a^2 m_b^2} + 10\frac{p_a^2(\vec{n}_{ab}\cdot\vec{p}_b)^2}{m_a^2 m_b^2} - 12\frac{(\vec{p}_a\cdot\vec{p}_b)(\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_b)}{m_a^2 m_b^2} \\ -3\frac{(\vec{n}_{ab}\cdot\vec{p}_a)^2(\vec{n}_{ab}\cdot\vec{p}_b)^2}{m_a^2 m_b^2}\end{array}\right]$$

$$+\tfrac{1}{8}G^2\sum_a\sum_{b\neq a}\sum_{c\neq a}\frac{m_a m_b m_c}{r_{ab}r_{ac}}\left[\begin{array}{l} 18\frac{p_a^2}{m_a^2} + 14\frac{p_b^2}{m_b^2} - 2\frac{(\vec{n}_{ab}\cdot\vec{p}_b)^2}{m_b^2} - 50\frac{\vec{p}_a\cdot\vec{p}_b}{m_a m_b} + 17\frac{\vec{p}_b\cdot\vec{p}_c}{m_b m_c} - 14\frac{(\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_b)}{m_a m_b} \\ +14\frac{(\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_c)}{m_b m_c} + \vec{n}_{ab}\cdot\vec{n}_{ac}\frac{(\vec{n}_{ab}\cdot\vec{p}_b)(\vec{n}_{ac}\cdot\vec{p}_c)}{m_b m_c}\end{array}\right]$$

$$+\tfrac{1}{8}G^2\sum_a\sum_{b\neq a}\sum_{c\neq a}\frac{m_a m_b m_c}{r_{ab}^2}\left(\begin{array}{l} 2\frac{(\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ac}\cdot\vec{p}_c)}{m_a m_c} + 2\frac{(\vec{n}_{ab}\cdot\vec{p}_b)(\vec{n}_{ac}\cdot\vec{p}_c)}{m_b m_c} + 5\vec{n}_{ab}\cdot\vec{n}_{ac}\frac{p_c^2}{m_c^2} - \vec{n}_{ab}\cdot\vec{n}_{ac}\frac{\vec{n}_{ac}\cdot\vec{p}_c}{m_c^2} \\ -14\frac{(\vec{n}_{ab}\cdot\vec{p}_c)(\vec{n}_{ac}\cdot\vec{p}_c)}{m_c^2}\end{array}\right)$$

$$+\tfrac{1}{4}G^2\sum_a\sum_{b\neq a}\frac{m_a^2 m_b}{r_{ab}^2}\left(\frac{p_a^2}{m_a^2} + \frac{p_b^2}{m_b^2} - 2\frac{\vec{p}_a\cdot\vec{p}_b}{m_a m_b}\right) + \tfrac{1}{2}G^2\sum_a\sum_{b\neq a}\sum_{c\neq a,b}\left(\begin{array}{l}\frac{m_a m_b m_c}{(r_{ab}+r_{bc}+r_{ca})^2}(n_{ab}^i + n_{ac}^i)(n_{ab}^j + n_{cb}^j) \\ 8\frac{p_{ai}p_{cj}}{m_a m_b} - 16\frac{p_{aj}p_{ci}}{m_a m_c} + 3\frac{p_{ai}p_{bj}}{m_a m_b} + 4\frac{p_{ci}p_{cj}}{m_c^2} + \frac{p_{ai}p_{aj}}{m_a^2}\end{array}\right)$$

$$+\tfrac{1}{2}G^2\sum_a\sum_{b\neq a}\sum_{c\neq a,b}\frac{m_a m_b m_c}{(r_{ab}+r_{bc}+r_{ca})r_{ab}}\left(\begin{array}{l} 8\frac{\vec{p}_a\cdot\vec{p}_c - (\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_b)}{m_a m_c} - 3\frac{\vec{p}_a\cdot\vec{p}_b - (\vec{n}_{ab}\cdot\vec{p}_a)(\vec{n}_{ab}\cdot\vec{p}_b)}{m_a m_b} \\ -4\frac{p_c^2 - (\vec{n}_{ab}\cdot\vec{p}_c)^2}{m_c^2} - \frac{p_a^2 - (\vec{n}_{ab}\cdot\vec{p}_a)^2}{m_a^2}\end{array}\right)$$

$$-\tfrac{1}{2}G^3\sum_a\sum_{b\neq a}\sum_{c\neq a}\frac{m_a^2 m_b m_c}{r_{ab}^2 r_{bc}} - \tfrac{3}{8}G^3\sum_a\sum_{b\neq a}\sum_{c\neq a}\frac{m_a^2 m_b m_c}{r_{ab}^2 r_{ac}} - \tfrac{3}{8}G^3\sum_a\sum_{b\neq a}\sum_{c\neq a,b}\frac{m_a^2 m_b m_c}{r_{ab}r_{ac}r_{bc}}$$

$$-\tfrac{1}{64}G^3\sum_a\sum_{b\neq a}\sum_{c\neq a,b}\frac{m_a^2 m_b m_c}{r_{ab}^3 r_{ac}^3 r_{bc}}\left[18r_{ab}^2 r_{ac}^2 - 60r_{ab}^2 r_{bc}^2 - 24r_{ab}^2 r_{ac}(r_{ab}+r_{bc}) + 60r_{ab}r_{ac}r_{bc}^2 + 56r_{ab}^3 r_{bc} - 72r_{ab}r_{bc}^3 + 35r_{bc}^4 + 6r_{ab}^4\right]$$

$$-G^3\sum_a\sum_{b\neq a}\frac{m_a^2 m_b^2}{r_{ab}^3}$$

where G is a gravitational constant, m_a is the mass of body a, p_a is the momentum vector of body a, r_{ab} is the position vector between bodies a and $b\left(\left|\vec{r}_a - \vec{r}_b\right|\right)$, and $\vec{n}_{ab}$ is the normalized position vector between bodies a and $b\left(n_{ab} = \frac{\vec{r}_a - \vec{r}_b}{\left|\vec{r}_a - \vec{r}_b\right|}\right)$. The reader will notice that later terms are higher orders of $\frac{1}{r}$ and are therefore supposed to play a lesser

role than prior terms. Because the bodies are represented as point masses, this perturbative assumption is violated when the bodies approach infinitely close to one another and the later terms grow larger then the prior terms. Also of note is the presence of vector dot products in the PPN2. The behavior of the three bodies will change depending on their relative momentum vectors, not just their relative positions as with Newtonian gravity. It is clear that the Hamiltonian PPN2 is much longer and more complex than that of Newtonian gravity.

The equations of motion for the three bodies are derived from the above Hamiltonian using Hamilton's Equations:

$$\dot{q}_i = \frac{\partial H}{\partial p_i}, \quad \dot{p}_i = -\frac{\partial H}{\partial q_i},$$

where H is the Hamiltonian; q is a generalized coordinate vector, i ranges over all 3 masses as well as all 3 vector components (creating 18 equations in all), and p is the conjugate momentum, defined as $p_j = \frac{\partial L}{\partial \dot{q}_j}$, where L is the Legrangian. The 18 equations for both models of gravity are generated using Maple which also converts them to Fortran code for computational use. To exemplify the complexity of PPN2, it took six thousand lines of code to implement. This high level of complexity renders improbable the idea of writing a customized ODE solver; it is explained later that an ODE solver with an adaptive time step was needed to integrate such difficult equations.

2.2 Initial Conditions

As explained previously, the three-body problem has 18 equations and 18 unknown variables: each of the three bodies has three positional variables and three momentum

variables. The same problem setup as [9] is followed; imposing the laws of conservation of energy and conservation of momentum (both linear and angular) reduces the number of unknowns to 14. Restricting motion to a plane further reduces the number of unknowns to 8. Writing the equations in the center of mass frame imposes two more constraints. I use an initial configuration, shown in Figure 2.1, which is specified by only two free parameters: ρ and ϕ, which will be addressed below. Though the number of initial parameters has been pared down to two, the full 18 equations must be solved.

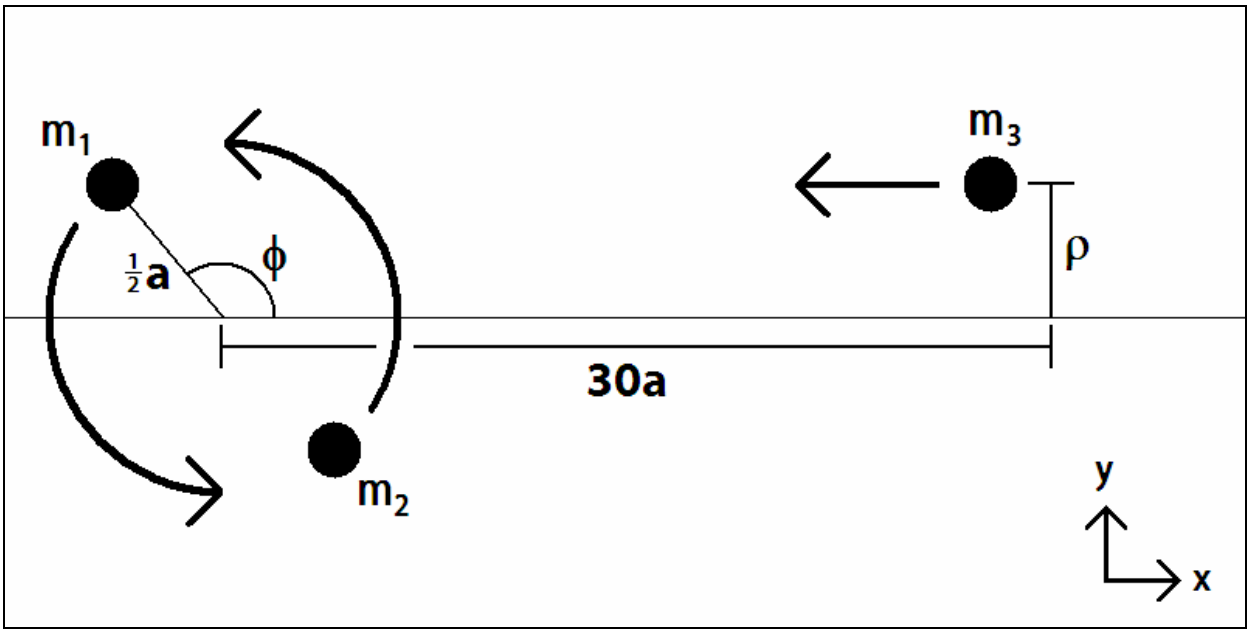

Figure 2.1 Three-body problem setup

The initial three-body configuration is shown in Figure 2.1. Two of the masses form a stable Newtonian binary star system; they initially orbit each other in a circle, separated a distance *a* apart. The third body is separated a distance 30*a* from the binary's center of mass. The two free parameters are the initial phase angle (ϕ) of the binary and the impact parameter (ρ) of the incoming field star. For ϕ=0, both masses one and two lie on the x-axis. ρ measures the perpendicular distance of mass three from the x-axis. The

arrows in Figure 2.1 show the initial trajectory of the three masses. The initial conditions are calculated as follows:

$$m = 1x10^{7}$$

$$m_1 = m_2 = m_3 = m$$

$$G = 6.673x10^{-11}$$

$$a = 1.7$$

$$\vec{x}_1 = \left\{\frac{a}{2}\cos(\phi), \frac{a}{2}\sin(\phi), 0\right\}$$

$$\vec{x}_2 = \left\{-\frac{a}{2}\cos(\phi), -\frac{a}{2}\sin(\phi), 0\right\}$$

$$\vec{x}_3 = \{30a, \rho, 0\}$$

$$v = \sqrt{G\frac{m}{2a}}$$

$$v_c = \sqrt{\frac{3Gm}{2a}}$$

$$\vec{v}_1 = \{-v\sin(\phi), v\cos(\phi), 0\}$$

$$\vec{v}_2 = \{v\sin(\phi), -v\cos(\phi), 0\}$$

$$\vec{v}_3 = \{-\tfrac{1}{2}v_c, 0, 0\}$$

Because this system is chaotic, a minute change in the initial ρ or ϕ can drastically change the system's behavior. To fully explore the system's behavior, many combinations of (ρ, ϕ) initial conditions are explored. The initial conditions have a ρ range of $-4.5a \leq \rho \leq 7.5a$ (no chaotic scattering is observed outside this range; verified by [9] and this study). The ϕ range used is $0 \leq \phi \leq \pi$; because the two binary bodies have equal masses, the range of $\pi \leq \phi \leq 2\pi$ is identical to $0 \leq \phi \leq \pi$.

The kinetic and potential energies of the binary system are determined by their masses and separation distance *a*. The potential energy of the third mass is set by the

masses of the three bodies and by its separation distance ($30a$) from the binary system. The third mass is given a velocity of half the critical velocity (velocity for the third body which makes the system's total energy exactly zero) which ensures that the system's total energy is negative. This negative energy guarantees the incoming field star can become bound to the binary such that all three bodies can be simultaneously bound.

In the computational simulations below, the three-body interaction ends when one of the three masses is ejected from the system—it is far from the remaining two bodies and its kinetic energy (escape velocity) is enough to break free of the gravitational pull of the remaining two masses. When the simulation ends, several system characteristics or escape values are recorded and their values are assigned to the initial condition coordinate (ρ, ϕ). The escape values are:

1. Escape Angle: angle at which the escaping body is traveling
2. Escape Time: how long the simulation ran before a body escaped
3. Change in Angular Momentum: $L_{final} - L_{initial}$
4. Minimum Approach Distance: minimum separation of all three bodies ($r_{12}+r_{13}+r_{23}$) attained during simulation
5. Maximum Momentum Dot Product: maximum $\mathbf{P}_1 \cdot \mathbf{P}_2 + \mathbf{P}_1 \cdot \mathbf{P}_3 + \mathbf{P}_2 \cdot \mathbf{P}_3$
6. Escaping Body: which of the three bodies escaped

To monitor the accuracy of the simulation, I also record the following conserved quantities (discussed in Section 4.3):

7. Linear Momentum Error: $P_{final} - P_{initial}$
8. Angular Momentum Error: $\dfrac{L_{final} - L_{initial}}{L_{initial}}$
9. Total Energy Error: $\dfrac{E_{final} - E_{initial}}{E_{initial}}$

The energy and momentum quantities are calculated according to Newtonian definitions. Calculating PPN2 energy in the Newtonian way may be a contributor to the energy error discussed in Section 4.3. It is also of note that some of these escape values and conserved quantities are physically meaningful (coordinate independent) and some are not (coordinate dependant). Regardless, it is still helpful to track quantities which have no strict physical value, since they do give insight into the behavior of the equations.

The most sensible way to compare all these extracted values is to graph them by initial conditions (ρ, ϕ) coordinates. The images in Chapter 4 contain these graphs. The x, y axes represent ρ, ϕ and the escape variable's value is represented by a color (ranging from blue to red). Generating colored maps of the escape values is an effective method of analyzing the overall behavior of the three-body system.

Chapter 3

Problem Solution

3.1 Solving the Equations

In order to explore the behavior of the three-body problem, computational tools are needed to numerically solve the differential equations analyze the solution. Hamilton's equations of motion are first order equations of the form

$$\frac{dy}{dt} = f(t, y)$$

where y is a state vector of the unknowns (x, y, z, v_x, v_y, v_z) for each body and f is a vector function. These equations are solved numerically using the Adams-Bashforth method

$$y_{n+1} = y_n + h\sum_{j=0}^{s} b_j f\left(t_{n-j}, y_{n-j}\right),\ 0 \le \text{r} \le \text{n},$$

$$b_j = \frac{(-1)^j}{j!(s-j)!}\int_0^1 \prod_{i=0, i\neq j}^{s}(u+i)du\ ,\ \text{j=0,…,s.}$$

This scheme can support different orders of approximation, and therefore different orders

of dependency in *y*. I use LSODA: the Livermore Solver for Ordinary Differential equations with Automatic method switching for stiff and non-stiff problems; it is publicly available at http://netlib.org. This integrator adaptively adjusts the time steps and automatically switches to implicit methods for stiff problems. A stiff differential equation is one in which two drastically different time scales are present in the solution. The result is that the integration must use a very small time step to achieve a reasonably accurate solution. It is proposed that the system of bodies used in this study does become stiff since two differing time scales do exist—when a tightly bound binary revolves very fast while the third body executes a large slow orbit around the binary. In this instance LSODA must use a very small time step and an implicit solver; if the needed time step is smaller than the minimum threshold, the integration will terminate with an error. LSODA was in fact encountering stiff situations while solving the equations in this paper.

Because Hamilton's equations are symplectic, a symplectic integrator would be preferred as it guarantees strict conservation of the Hamiltonian. However, such integrators require fixed time steps. A more robust integrator with an adaptive time step is required to handle the overwhelming complexity of the three-body equations. Because a non-ideal integrator is used, energy conservation is monitored to ensure an accurate integration.

3.2 run

Because the three-body problem in both Newtonian and PPN2 gravity allows only numerical solutions, many different programs were needed to step from the initial setup to the color maps. The first program needed is a Perl script called *run*. *Run* converts the initial conditions (ρ, ϕ) into the numerical setup of the three body system and begins its

evolution. The user may specify the value of G, the universal gravitational constant, and the masses of the three bodies. The user also specifies the range of the initial conditions as well as how many simulations to run in that interval. For example, one may select a ϕ range of $\frac{1}{4}\pi \leq \phi \leq \frac{1}{2}\pi$ and to perform one hundred simulations in that interval. In this case *run* will execute a three-body simulation with an initial ϕ of $\frac{100}{400}\pi$, $\frac{101}{400}\pi$, $\frac{102}{400}\pi$,…, $\frac{200}{400}\pi$, and a similar group of values for ρ. After calculating the system setup from the ρ, ϕ initial value pair as well as gravity and mass values, the information is given to *nbodyPN* which evolves the system.

3.3 nbodyPN

NbodyPN is a FORTRAN program which evolves n gravitating bodies in either Newtonian or PPN2 gravity. *NbodyPN* stores in two of its subroutines the ordinary differential equations of motion for both models of gravity. It reads the initial conditions from *run*.

At its core, nbodyPN uses LSODA to perform the differential equation integration. NbodyPN uses this integrator to step the three bodies along their paths of motion until one of the three bodies meets the escape requirements. A body has escaped when its distance from the center of mass is almost one hundred times the original separation of the field body from the binary's center and its energy is positive. At this point, all escape values are printed to a data file.

3.4 DaFilt

The data file needs to be processed before its information can be extracted. DaFilt ("Data

Filter"), a Java program, filters the data, to ensure that all the correct information has been recorded and all erroneous data has been corrected. The raw data file has numbers scattered across many lines which belong in a group. DaFilt reads all the information, analyzes it, and rewrites it cleanly to a new file. DaFilt records the ranges of ρ and ϕ, $d\rho$, $d\phi$, the number of runs in each dimension, and how many errors were present in the file. Errors are substituted with dummy numbers (such as -99) so that the data file can still be processed but that errors are not misunderstood as actual data.

3.5 Grapher

Grapher, a MatLab program, is used to analyze all the data and display it visually as the colored maps shown in Chapter 4. This program uses an array of matrices to store all the data which it reads from the file generated by DaFilt. Each matrix has the same dimensions as ρ and ϕ. Each escape variable gets its own matrix. The value for that escape variable for every run within the ρ, ϕ range is loaded into the matrix.

Because the escape values can have a wide range of values, they are normalized for visualization purposes. The time duration of a simulation, for example, can range from a few seconds to several million seconds. First, the most extreme values are brought within the average range. For example, if most of the simulation times were between one and two thousand seconds, any values below one thousand seconds were set equal to one thousand seconds and all values above two thousand seconds were set equal to two thousand seconds. Second, that average range is scaled between $0 \rightarrow 60$ which MatLab interprets as a color ranging from blue (0) to red (60) (Figure 4.1).

Chapter 4

Results

4.1 Introduction

This chapter presents the results of the three-body simulations. The figures in this chapter compare the chaotic features between the Newtonian and PPN2 models of gravity by plotting the data against the initial data parameters ρ and ϕ. Each physical characteristic listed in Section 2.2 is plotted in five different figures below. The first shows the entire parameter space (titled whole space), and the remaining four show detailed regions that highlight the distinctions between Newtonian and PPN2 gravity. Figure 4.1 is a reference showing the color spectrum for all plots.

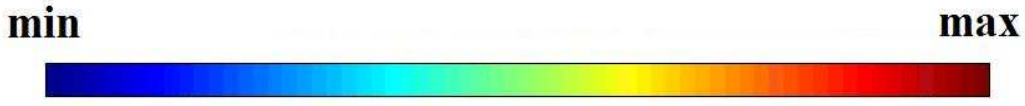

Figure 4.1 Color scale for figures

The two maps in Figure 4.2 of the whole space's Escape Angle identify sub-regions 1 through 4. The map on the top represents Newtonian gravity, while the map on

the bottom represents Einstein's gravity. Region 1 is outlined by the lower left box; region 2 is the upper left box; region 3 is the lower right box; region 4 is the upper right box. The colors in each graph represent the value of the variable being mapped. In the case of these two maps, the colors represent the Escape Angle.

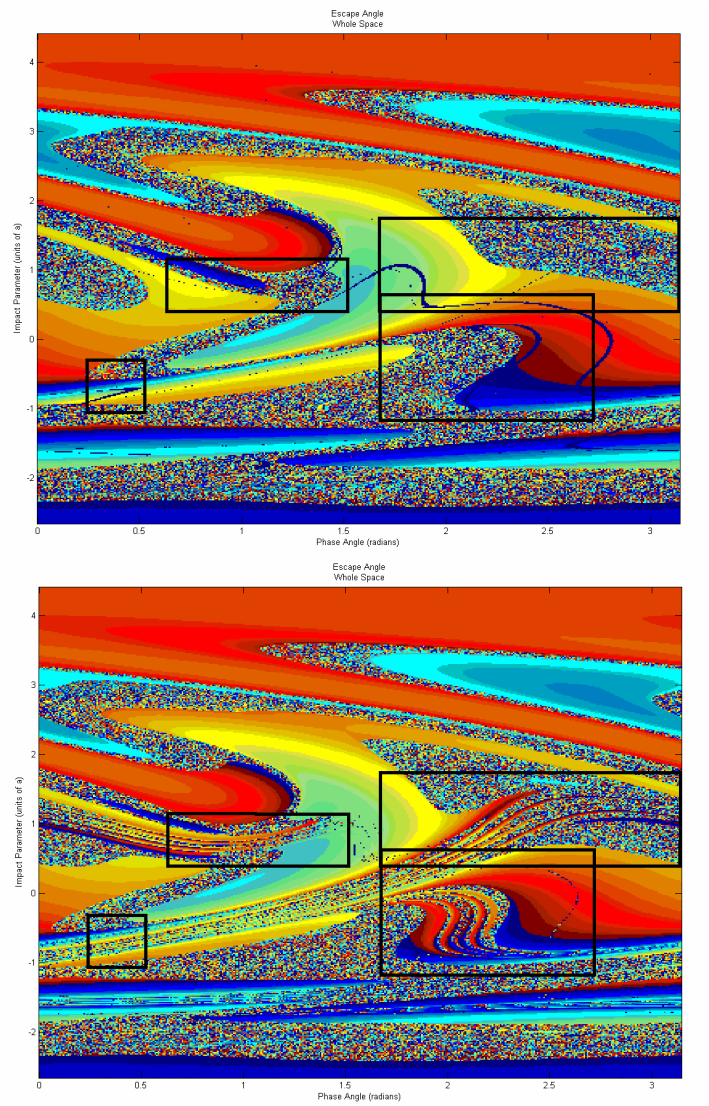

Figure 4.2 Boxes outlining Regions 1-4 of Whole Space; Newtonian (top) and PPN2 (bottom). Region 1 is outlined by the lower left box; region 2 is the upper left box; region 3 is the lower right box; region 4 is the upper right box.

Each region is discussed individually in the next sections, contrasting the features of the Newtonian and PPN2 plots.

4.2 Whole Space

4.2.1 Escape Angle

Figure 4.3 features both regions of smooth flowing color and chaotic speckled colors. The colors in these two maps (and all others labeled Escape Angle) represent the angle at which the escaping body was ejected from the system. Smooth colors, where the red transitions into orange, yellow, green and finally to blue, represent regions of non-chaotic interactions. This means that changing the initial conditions slightly, either in the ρ or ϕ dimensions (moving vertically or horizontally on the map by a pixel), results in a slightly different outcome (the exit angle color has only changed a shade). On the other hand, the speckled regions show chaos; a small change in initial conditions (to a neighboring pixel) results in a substantial change in the escape angle (color).

For the most part, these figures appear very similar but with one difference: the lower map (PPN2) has a few extra stripes in clusters of four or five—fourfold rivers. These fourfold rivers are seen in a few places in the lower map but nowhere in the upper map; they are unique to the PPN2 model of gravity, and appear in most PPN2 data maps. The four sub-region maps (below) zoom in on these fourfold rivers. One slight difference between each set of maps is that the PPN2 map seems to be shifted to the left: corresponding points will have a lesser ϕ coordinate in the PPN2 map than the Newtonian map.

Because the x-axis represents an angle, the map actually wraps on itself; the 0 and π radians match. Figure 4.4 shows the PPN2 map of the Whole Space copied three times.

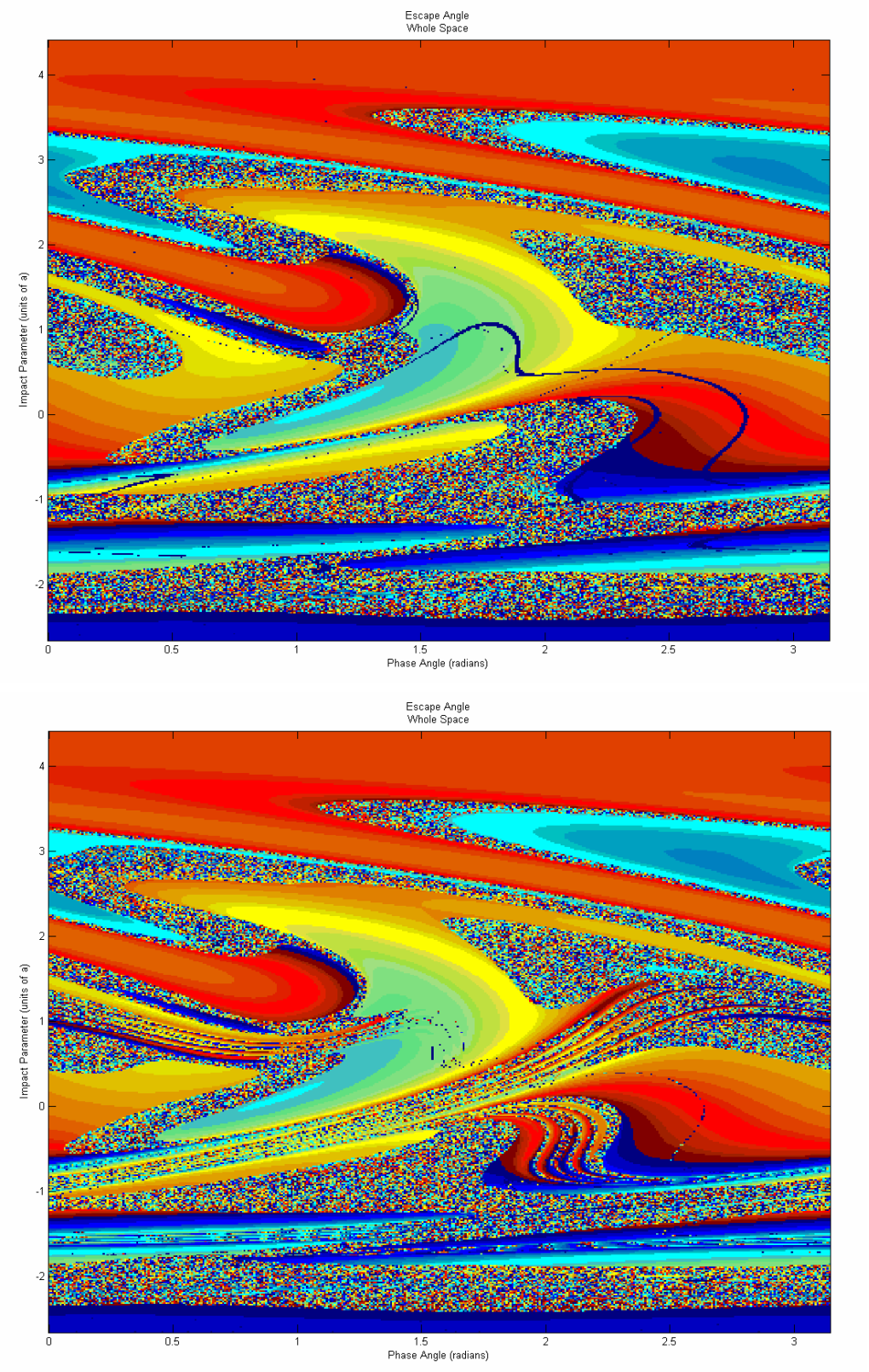

Figure 4.3 Escape Angle of Whole Space; Newtonian (top) [0, 2π] and PPN2 (bottom) [0, 2 π]

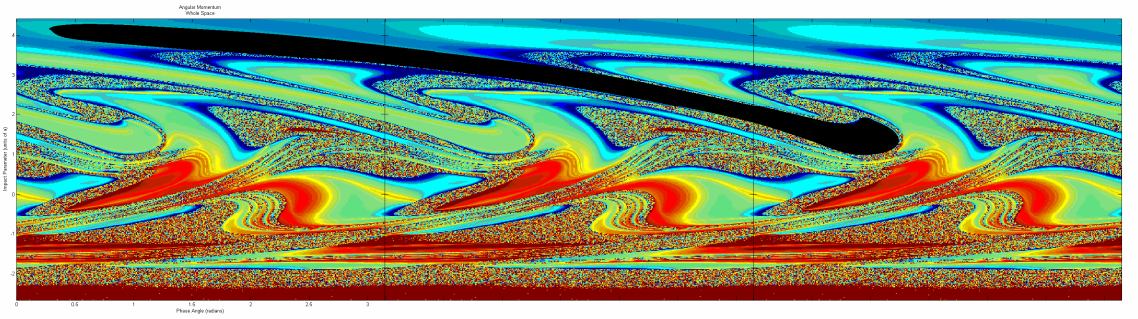

Figure 4.4 Thrice-repeated figure of Angular Momentum Error of Whole Space; black structure identifies a feature which wraps across parameter space three times.

The black stripe in Figure 4.4 exemplifies a structure which spans three all three copies. It is much easier to pick out this structure seeing the three copies aligned. It is, however, recognizable by following along a single map and wrapping around it a few times. There are other macroscopic features which can be seen more clearly. These continuous patterns suggest a large structural pattern in this three-body setup. While some of the data analysis in this Whole Space region will address macro-structures, the analysis of the four sub-regions (below) is meant to explore smaller-scale structures. Other escape values will give a more thorough look at these interactions.

4.2.2 Escape Time

The same chaotic and non-chaotic regions are reflected in Figure 4.5 of the interaction time. The colors in this map reflect how much time had elapsed before the escaping body was ejected from the system. The homogenous blue color indicates that most of the non chaotic interactions took a similar amount of time. While the chaotic colors suggest that the interaction times varied highly through the chaotic regions. The main reason for drastic changes in interaction time is the complexity of the interaction. A simple interaction in which the incoming star simply scatters around the binary pair will take the least amount of time. Whereas an interaction where the incoming star becomes quasi-

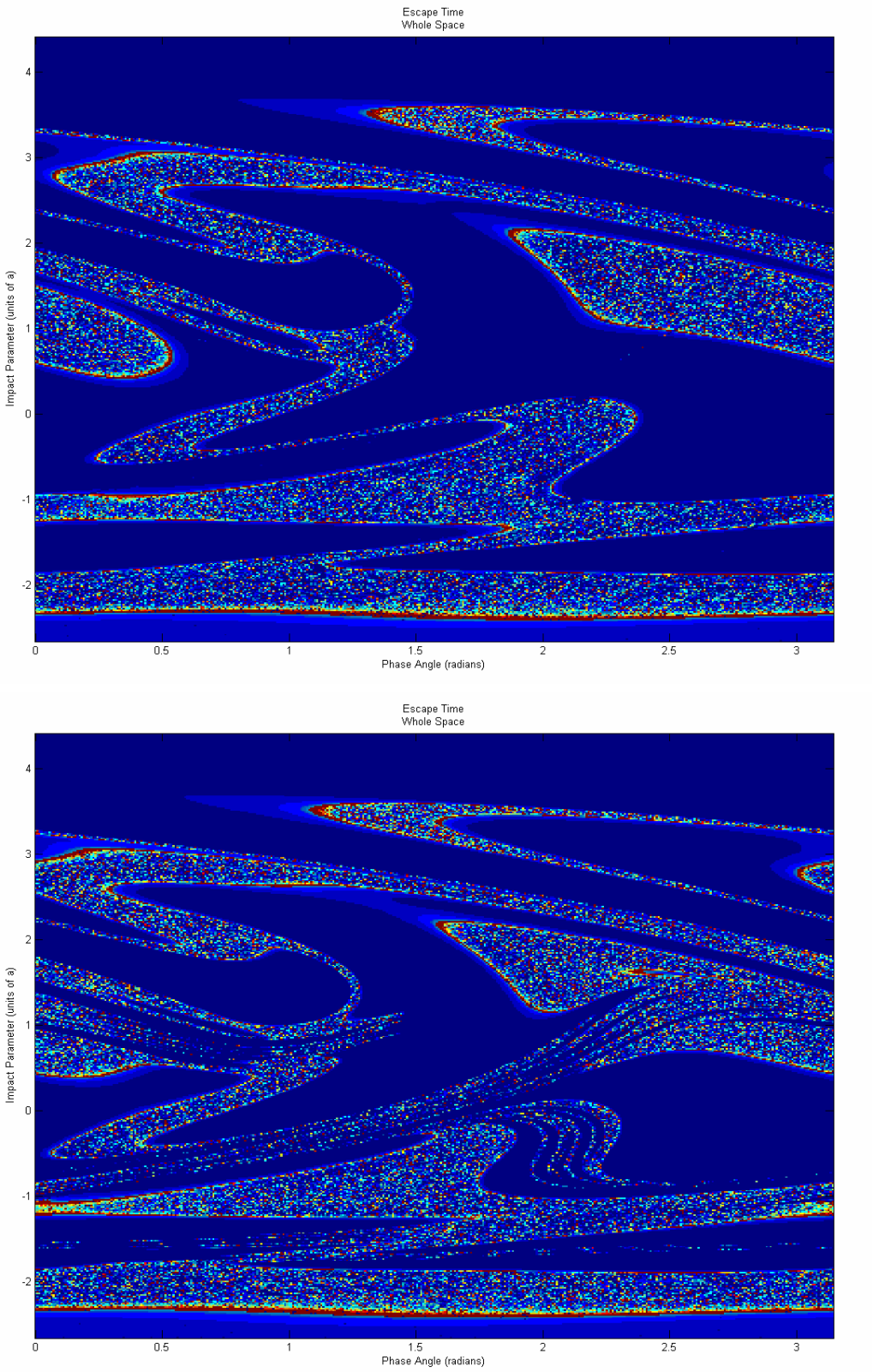

Figure 4.5 Time of Whole Space; Newtonian (top) [3.6x10^3, 9.7x10^4] and PPN2 (bottom) [3.2x10^3, 2.8x10^7]

bound to the other two and all three bodies orbit around each other may take a very long time. Thus, in the chaotic regions, the amount of time which the planets spend orbiting around one another varies highly.

Notice that much of the boundary between the smooth and chaotic regions in Figure 4.5 is outlined in red. The red represents a very long interaction time. While the red speckles in the chaotic regions may be caused by planets orbiting each other many times, the red outline is caused by one long orbit. Boyd [3] explains that on the boundary between smooth and chaos, the third (incoming) body comes very close to one of the binary bodies and there is a strong change in energy. The third body becomes very weakly bound to the other two such that it executes a very long orbit. It is this long orbit which separates the chaotic and non-chaotic regions.

4.2.3 Change in Angular Momentum

Figure 4.6 represents the change in angular momentum. Ideally, each of these values should be zero. They are not. Because the integrator, LSODA, uses some specified algorithm, it can be assumed that the degree of error is directly proportional to characteristics of the interaction. Thus, maps of the change in angular momentum are significant though it is not specifically known what about interaction causes the error.

The topological characteristics of these two maps are similar for the above two maps featuring the Escape Angle. There are some regions where smooth transitions in color testify of smooth behavior: a small change in initial conditions results in a small change in angular momentum error. The regions which were chaotic for the Escape Angle are also chaotic in these maps.

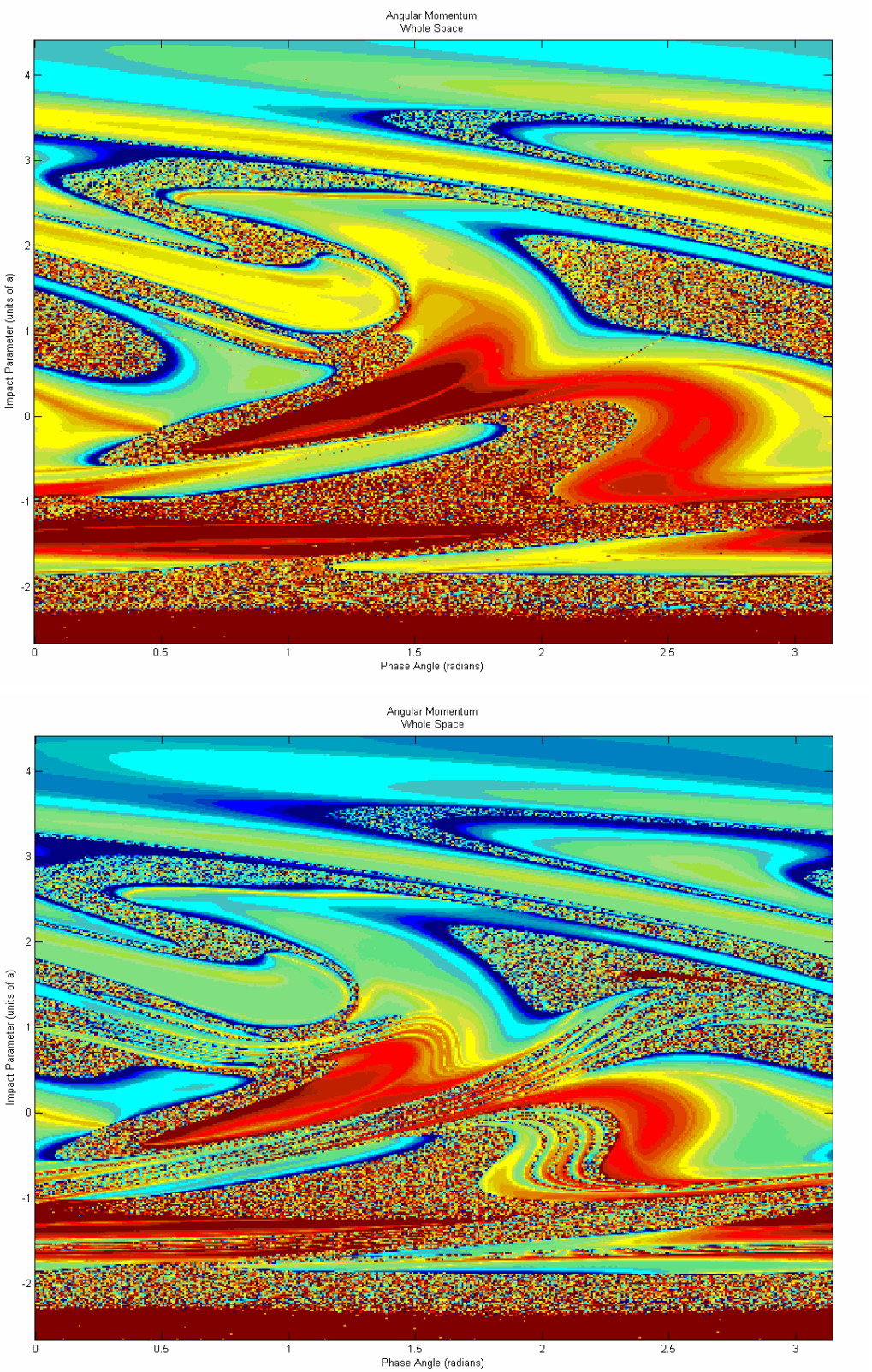

Figure 4.6 Change in Angular Momentum of Whole Space; Newtonian (top) [-2.2x10^5, 2.3x10^5] and PPN2 (bottom) [-2.0x10^5, 2.4x10^5]

4.2.4 Minimum Approach Distance

Maps of the minimum approach distance are important because the higher order terms of PPN2 have multiples of $\frac{1}{r}$. The closer the bodies approach one another, the more of a role those terms will play. Figure 4.7 reveals some underlying features. Compare Figure 4.7 with their Escape Angle counterparts (Figure 4.3). Notice that the regions which behave smooth (according to the Escape Angle) appear here in red and yellow colors while chaotic regions show up blue. Recall the color spectrum (Figure 4.1); blue represents low values, while red represents high values. Why would chaotic interactions always experience shorter approach distance? It makes sense that the longer the interaction time, the more opportunities the bodies have to pass by one another and statistically the higher the chance of a very close approach. Therefore the chaotic, more complex, interactions have more opportunities for close approaches. On the other hand, the smooth regions are often caused by the third incoming body simply passing around the binary system and being ejected; this would allow only one opportunity for a short approach distance.

The other major characteristic seen in these two maps is a pattern of concentric ovals. In the very center of both these maps, there are red concentric ovals. The smaller ovals in the center are a lighter shade of red representing a closer approach distance. Along the bottom of the maps is a string of blue concentric ovals. The smaller ovals in the center get darker and darker, which also represents a closer approach distance. Both the red and blue ovals appear in regions which were classified as smooth by the Escape Angle maps (Figure 4.3). The rivers of chaos seem to streak through the smooth regions, but in the Minimum Approach Distance maps, they do not interrupt the concentric ovals.

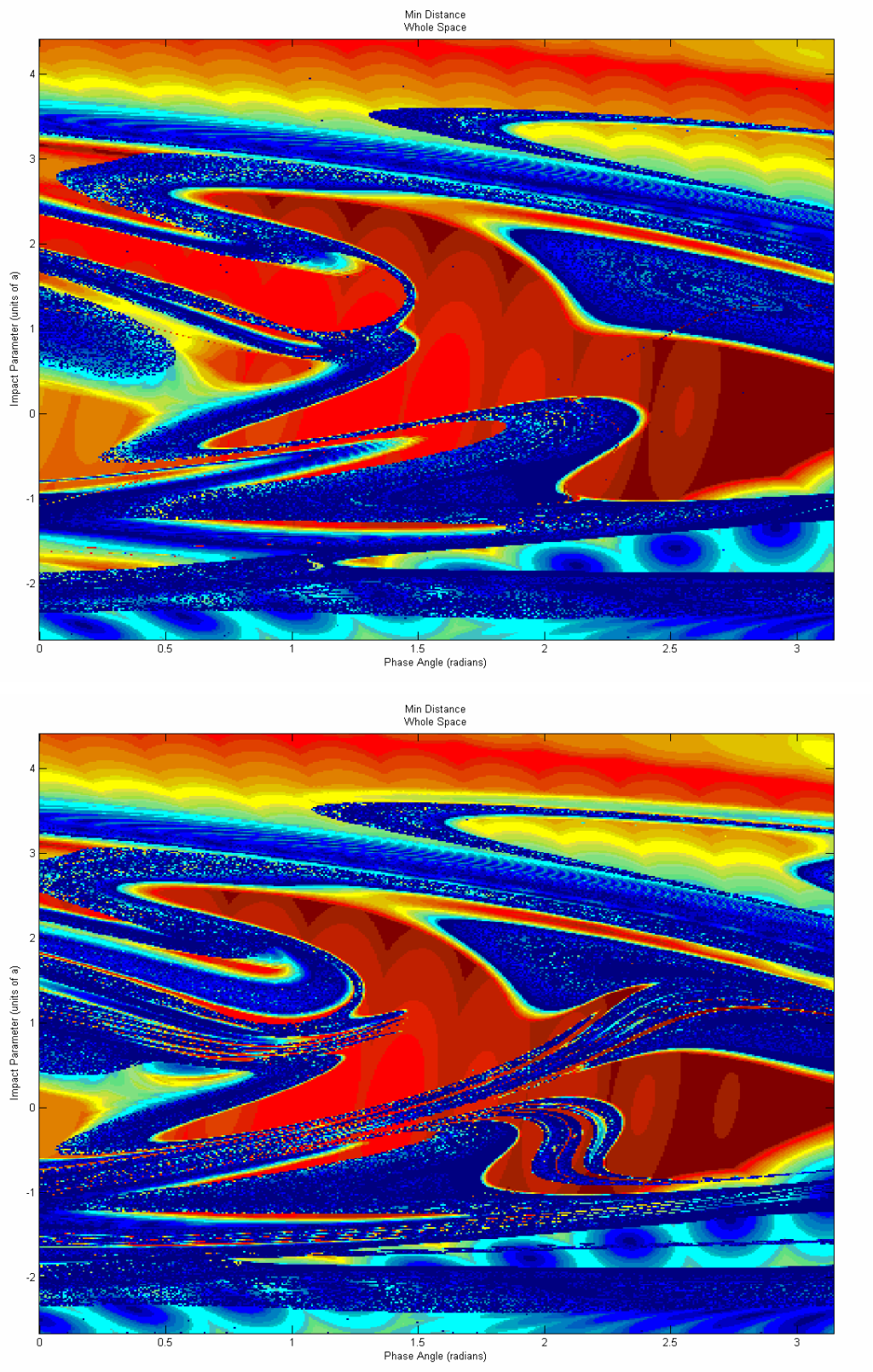

Figure 4.7 Minimum Approach Distance of Whole Space; Newtonian (top) [$9.9x10^{-8}$, $7.3x10^{-3}$] and PPN2 (bottom) [$6.7x10^{-8}$, $7.3x10^{-3}$]

The ovals seem to continue their patterns in the background of the chaos. These features will be seen more clearly in the sub-region maps below.

4.2.5 Maximum Momentum Dot Product

The relativistic Hamiltonian depends of the momentae of the bodies, while the Newtonian potential depends only on position. Only the relativistic Hamiltonian depends on the dot products $\mathbf{P}_a \cdot \mathbf{P}_b$. For this reason, the products $\mathbf{P}_1 \cdot \mathbf{P}_2 + \mathbf{P}_1 \cdot \mathbf{P}_3 + \mathbf{P}_2 \cdot \mathbf{P}_3$ are computed and mapped to shed light on the difference between the two models of gravity. These maps show the maximum total vector momentum achieved during a simulation.

Figure 4.8 sheds some light on what appears to be a singularly appearing phenomenon. Refer back to Figure 4.5 bottom. Near the coordinate $(\rho, \phi) = (1, 1.5)$ there is a river of chaos, flowing from left to right, which abruptly ends. The many other fourfold rivers, which appear in the PPN2 maps, flow from one place to another; they spring out of one location and bury into another. The river at (1, 1.5) has no destination. Why? The above map showing the PPN2 dot product offers some explanation. The same river can be seen flowing left to right at (1, 1.5). This river features a speckled dot product meaning its values are also chaotic. The region where the river dies seems to have a "shore" of light blue color. This light blue shore fades into a smooth deep blue color. The smooth deep blue area represents a local minimum. It is this minimum and whatever mechanism of the three body interaction that causes it, which ends the (1, 1.5) river and prevents it from reaching the light blue shore across the way.

Another key feature is the location of the highest dot products. They tend to pool together in two main regions. The first is located at ($-1 \rightarrow 0$, $2.5 \rightarrow 3$). The second is the

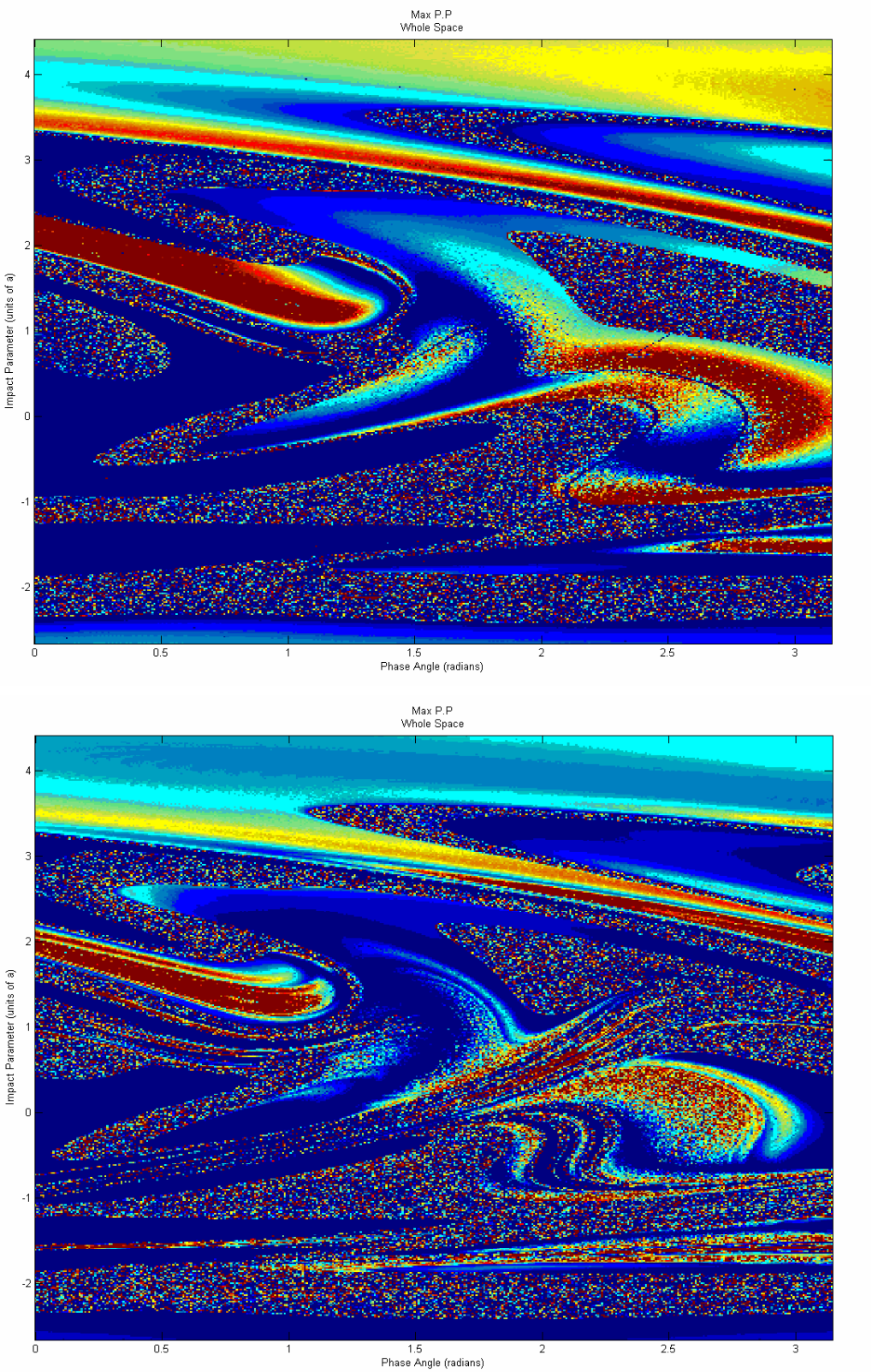

Figure 4.8 Max Momentum Dot Product of Whole Space; Newtonian (top) [$9.0x10^{6}$, $3.1x10^{26}$] and PPN2 (bottom) [$3.3x10^{18}$, $1.8x10^{26}$]

same region colored in black in the third map of Section 4.2.1. That stripe which spanned three spaces appears here to be comprised of the highest dot product values.

4.2.6 Escaping Body

Figure 4.9 illustrates which of the three bodies escaped: light green—1, orange—2, red—3, and blue represents a simulation which terminated because of an error. Recall that masses one and two form the binary, while mass three comes in from infinity. It makes sense that the areas near $\rho = -3$ and $\rho = 4$ are marked by body 3 escaping. For a large impact parameter body three passes far around the binary and simply scatters around them on a hyperbolic path.

Notice also that both plots of Figure 4.9 wrap around on themselves (like all other plots) but shift colors. An initial setup of $\phi = 0$ and $\phi = \pi$ is exactly the same except that bodies one and two have switched places. That is why, in Figure 4.9, the structure at $\phi = 0$ and $\phi = \pi$ are the same except that orange (body 2) is swapped for green (body 1).

4.3 Error and Data

As with all experimental and computational physics, only an accurate experiment can give accurate results. The error in these planetary simulations tells how accurate the simulations were carried out. I tracked the linear momentum error, angular momentum error, and energy error. The average errors were on the order of 10^{-7} for linear momentum, $10^{-3} \rightarrow 10^{-1}$ for angular momentum and 1 for energy. In the case of angular momentum and energy, only 5% of the simulations caused the error while 95% of the simulations had less than 1% error. The simulations of highest error were consistently

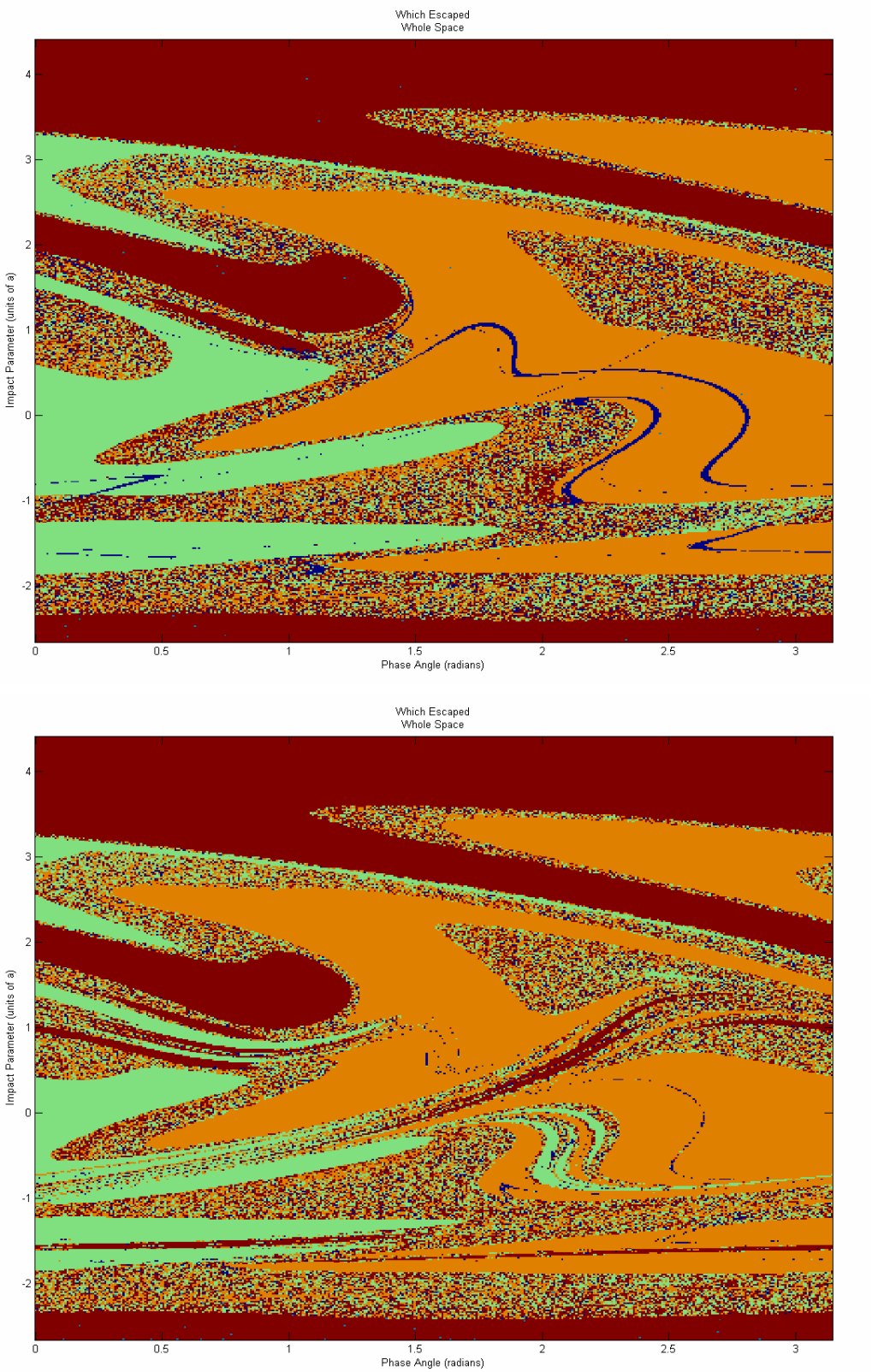

Figure 4.9 Escaping Body of Whole Space; Newtonian (top) [1, 3] and PPN2 (bottom) [1, 3]

within the chaotic regions where an accurate integration is improbable to begin with. These error values reflect positively on the reliability of the simulations.

For the reader's reference, Tables 4.1→4.5 give the ranges and averages of data used to make the figures in this paper; thus the reader may know the approximate values represented by the various colors.

	Newtonian			**PPN2**		
Variable	Minimum	Average	Maximum	Minimum	Average	Maximum
Esc Time	3.660E+03	9.764E+04	1.356E+07	3.200E+03	9.821E+04	2.861E+07
ΔL	-2.232E+05	-4.249E+02	2.318E+05	-2.058E+05	-3.583E+02	2.489E+05
Min Appr	9.912E-08	3.243E-03	7.302E-03	6.786E-08	2.876E-03	7.384E-03
Max P·P	9.000E+06	1.053E+24	3.116E+26	3.328E+18	1.346E+24	1.875E+26

Table 4.1 Statistical Data for Whole Space figures

	Newtonian			**PPN2**		
Variable	Minimum	Average	Maximum	Minimum	Average	Maximum
Esc Time	4.460E+03	1.236E+05	6.475E+06	3.200E+03	1.105E+05	8.606E+06
ΔL	-1.009E+05	-5.870E+02	9.845E+04	-9.676E+04	-6.990E+01	1.260E+05
Min Appr	2.745E-08	2.540E-03	6.117E-03	1.195E-07	1.172E-03	6.322E-03
Max P·P	9.000E+06	6.548E+23	3.819E+26	9.000E+06	7.510E+23	2.922E+26

Table 4.2 Statistical Data for Region 1 figures

	Newtonian			**PPN2**		
Variable	Minimum	Average	Maximum	Minimum	Average	Maximum
Esc Time	4.220E+03	9.912E+04	1.070E+07	3.200E+03	8.452E+04	1.424E+07
ΔL	-1.135E+05	-5.489E+02	2.525E+05	-1.103E+05	-3.192E+02	2.458E+05
Min Appr	3.705E-07	3.432E-03	6.715E-03	3.870E-07	3.253E-03	6.809E-03
Max P·P	9.000E+06	8.020E+23	5.305E+25	3.780E+20	1.226E+24	1.237E+26

Table 4.3 Statistical Data for Region 2 figures

	Newtonian			**PPN2**		
Variable	Minimum	Average	Maximum	Minimum	Average	Maximum
Esc Time	4.380E+03	7.199E+04	1.419E+07	3.160E+03	6.424E+04	7.835E+06
ΔL	-1.030E+05	7.152E+01	1.403E+05	-8.831E+04	6.373E+00	1.462E+05
Min Appr	1.614E-08	4.206E-03	7.172E-03	4.633E-09	4.364E-03	7.233E-03
Max P·P	9.000E+06	1.306E+24	1.944E+26	2.019E+19	2.107E+24	6.345E+28

Table 4.4 Statistical Data for Region 3 figures

	Newtonian			PPN2		
Variable	Minimum	Average	Maximum	Minimum	Average	Maximum
Esc Time	7.840E+03	9.912E+04	1.507E+07	3.180E+03	1.250E+05	1.000E+08
ΔL	-1.361E+05	-6.232E+02	2.524E+05	-1.698E+05	-5.524E+02	2.509E+05
Min Appr	1.395E-07	3.866E-03	7.340E-03	2.379E-07	2.603E-03	7.390E-03
Max P·P	9.000E+06	1.438E+24	2.692E+26	3.851E+20	1.574E+24	2.047E+26

Table 4.5 Statistical Data for Region 4 figures

4.4 Four Sub-Regions

There are similarities and differences between the Newtonian figures and the PPN2 figures. Since regions featuring the differences are of greatest interest, they are zoomed in on and explored in greater detail. Below are four regions which feature a fourfold river—a region in the PPN2 figure which has four (or five) alternating bands of smooth then chaos; a corresponding structure does not appear in the Newtonian figure.

4.4.1 Escape Angle

A close up view of the Escape Angle in these four sub-regions gives clear images of the banded rivers of smooth and chaos. Each pair of maps (Figures 4.10 through 4.13) is of a region where the system's escape angle behaves smoothly in the Newtonian picture but consists of four to five chaotic rivers in the PPN2 picture. The chaotic rivers all flow parallel to one another. Both the Newtonian and PPN2 maps have a fine blue line. Figure 4.14 is one example of this line so the reader may find it in each pair of maps. The line is made thick and black in Figure 4.14 and runs horizontally across the middle of both images.

Both of these Escape Angle maps of Region 2 show the line horizontal through the middle bowing downward. It seems strange that a random line appears in both maps but foretells where the chaotic rivers will appear in the PPN2 maps. What causes this line?

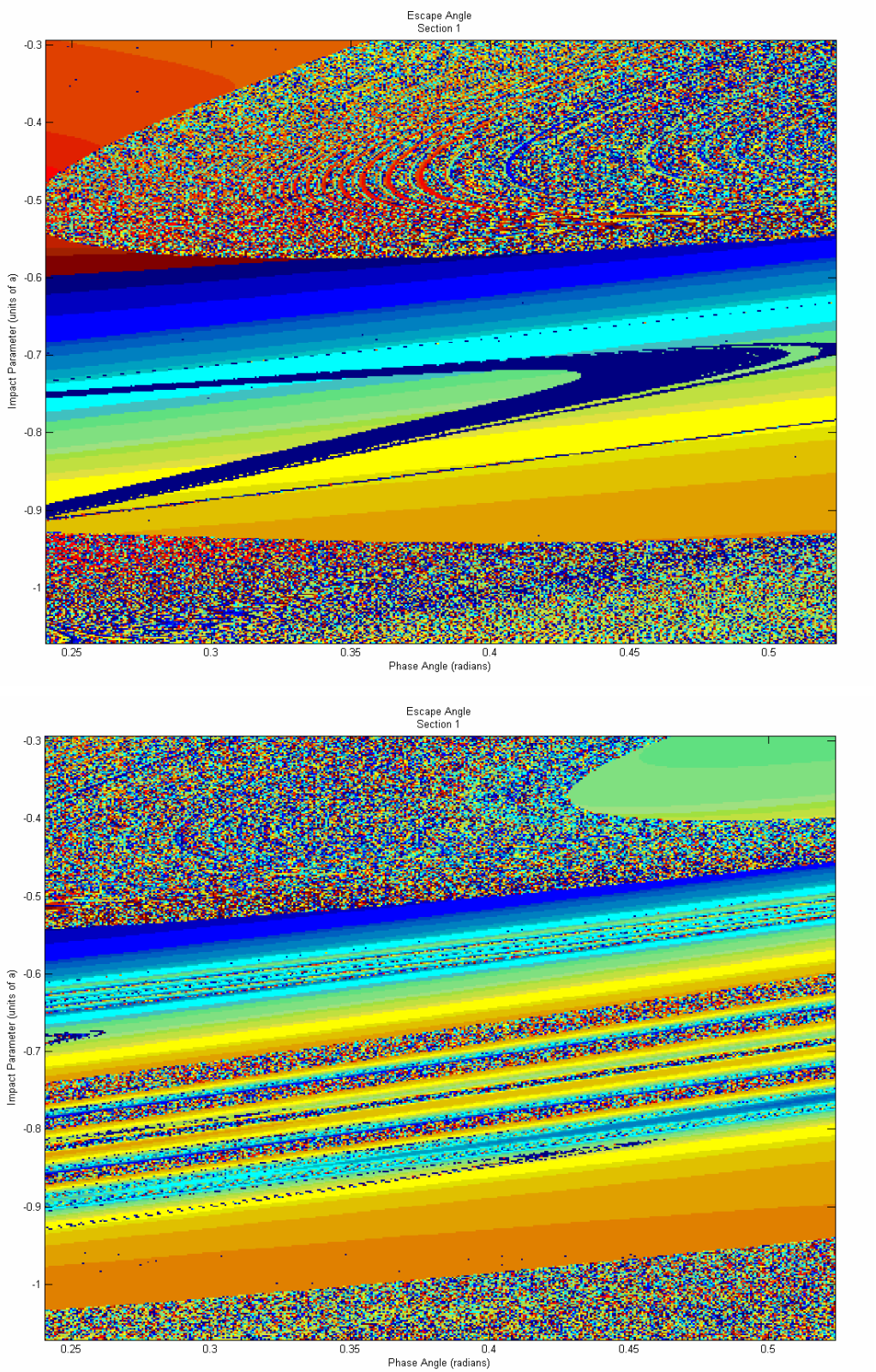

Figure 4.10 Escape Angle of Region 1; Newtonian (top) [0, 2π] and PPN2 (bottom) [0, 2π]

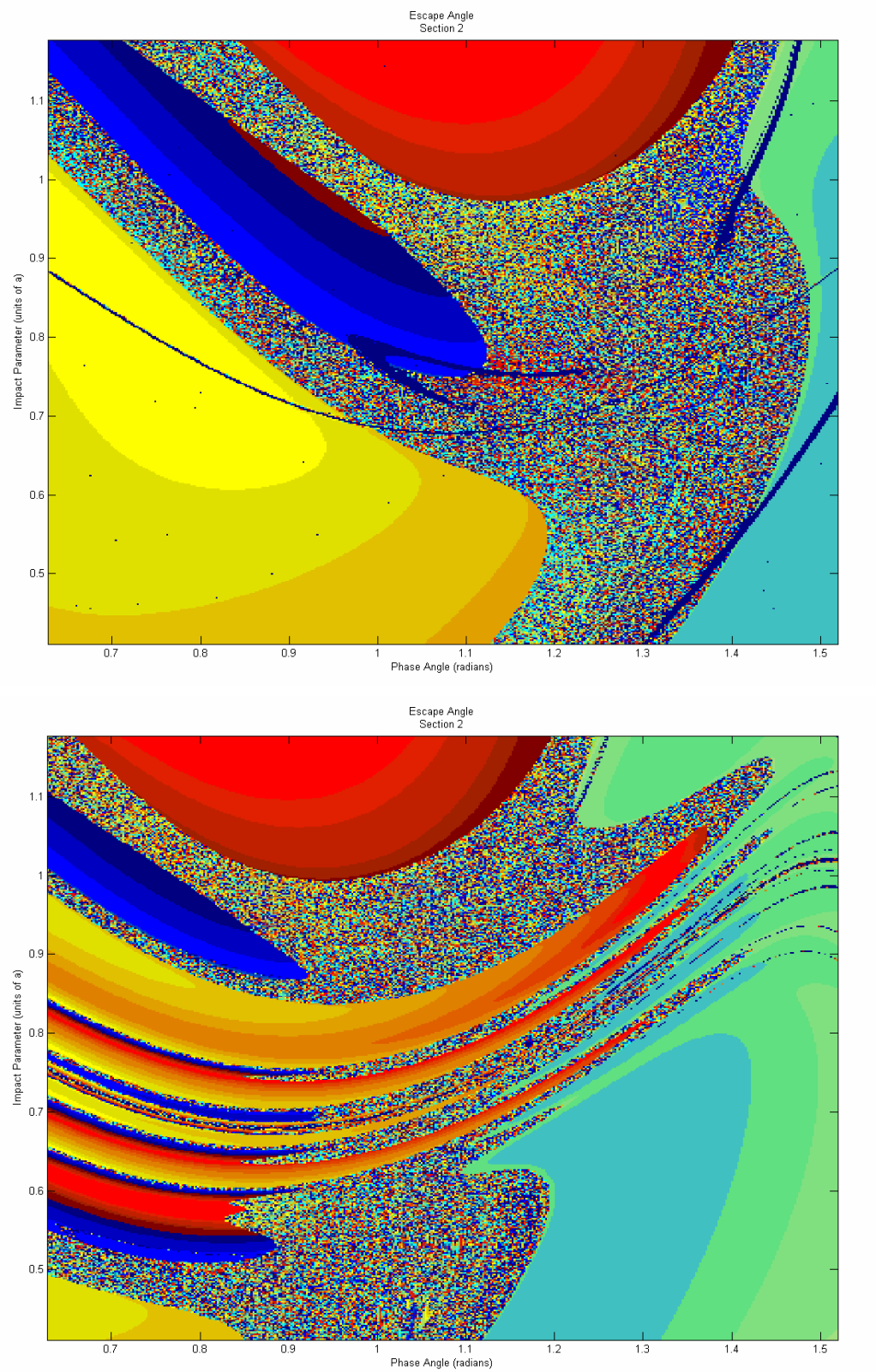

Figure 4.11 Escape Angle of Region 2; Newtonian (top) [0, 2π] and PPN2 (bottom) [0, 2π]

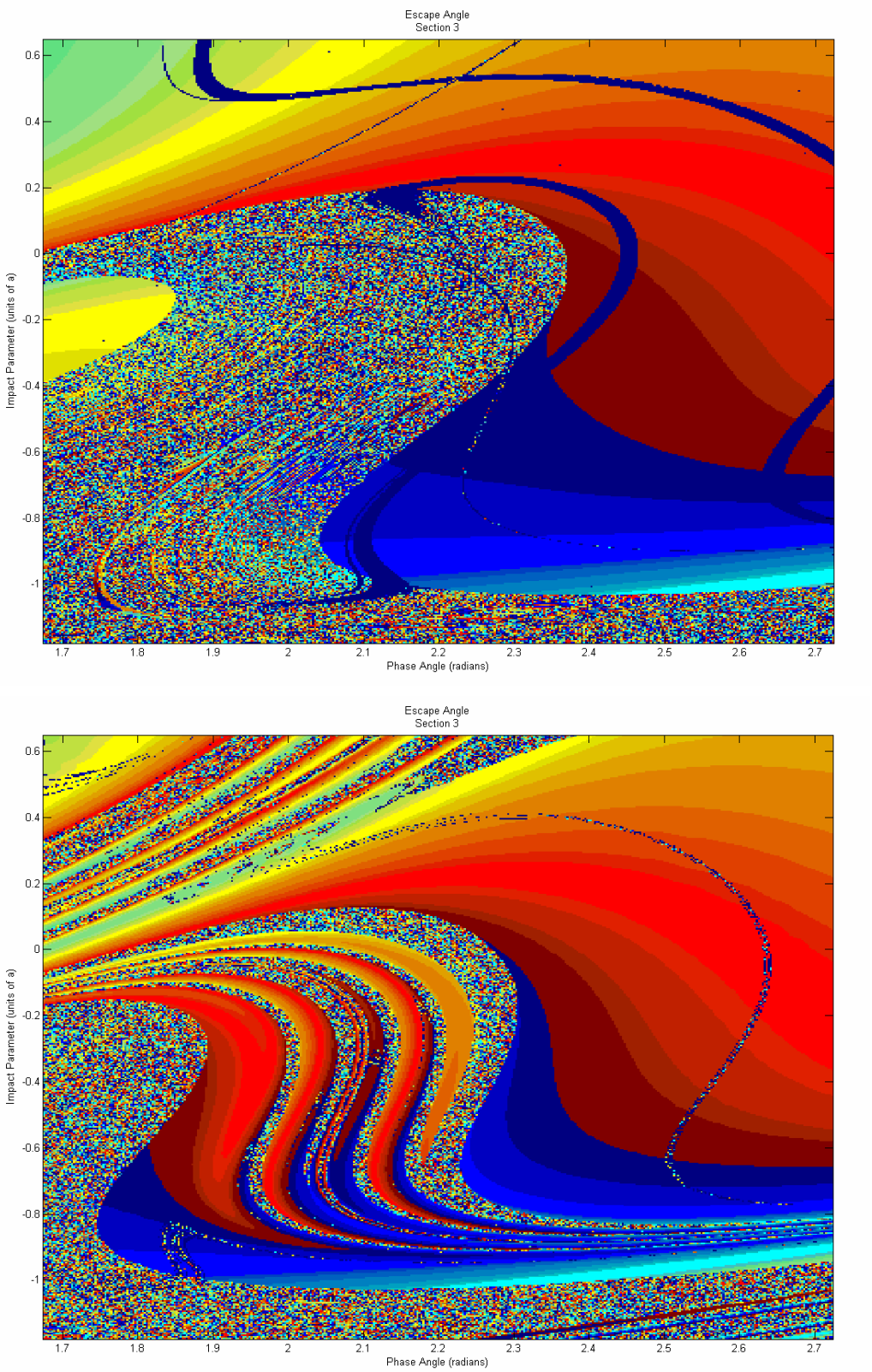

Figure 4.12 Escape Angle of Region 3; Newtonian (top) [0, 2π] and PPN2 (bottom) [0, 2π]

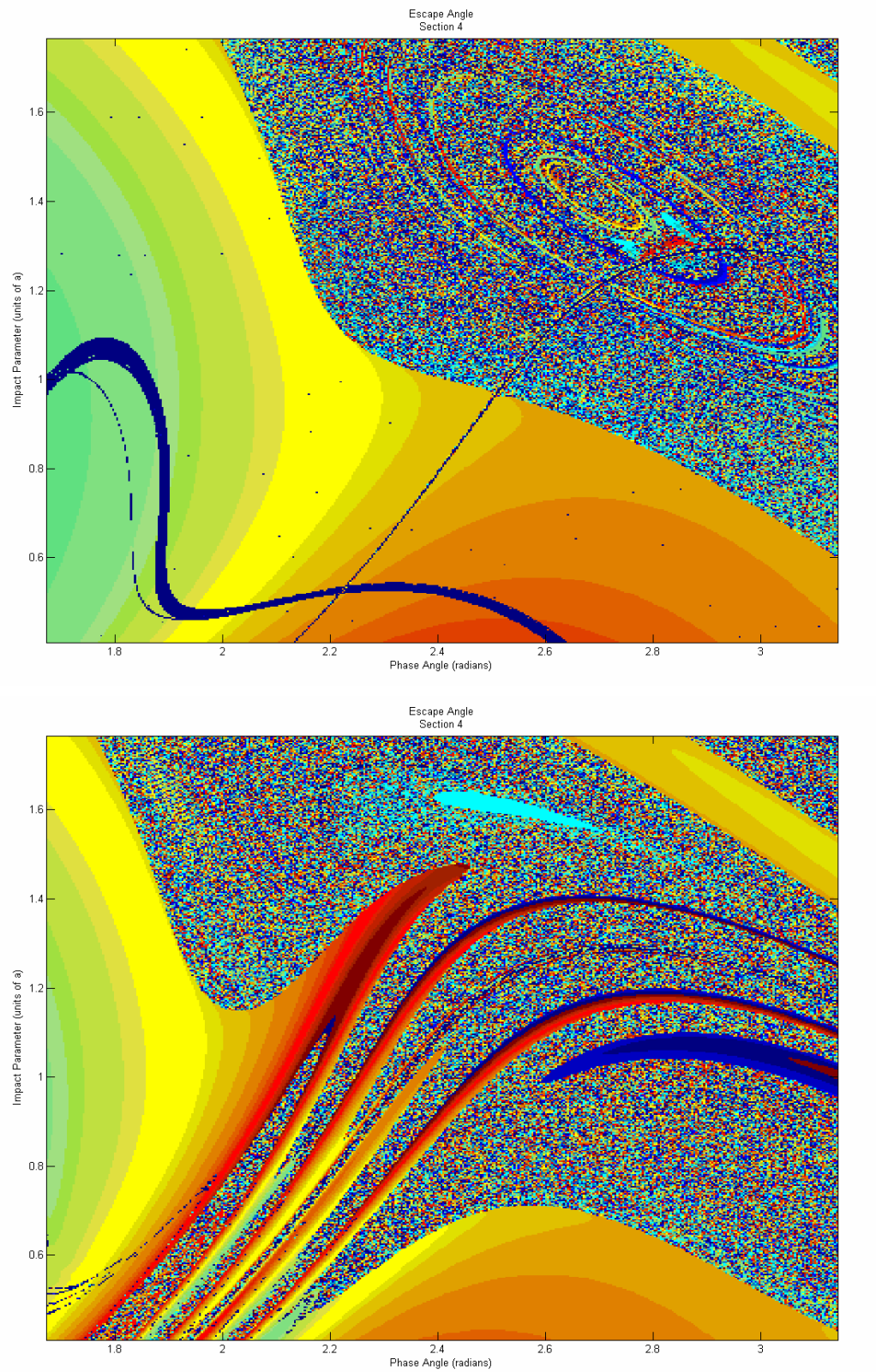

Figure 4.13 Escape Angle of Region 4; Newtonian (top) [0, 2π] and PPN2 (bottom) [0, 2π]

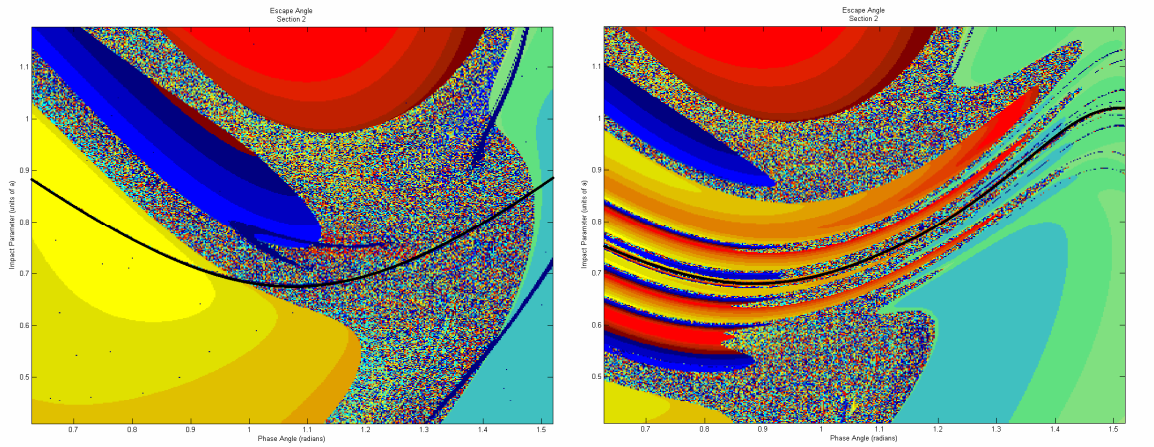

Figure 4.14 Escape Angle showing fine line of error; Newtonian (left) and PPN2 (right)

The black lines in Figure 4.14 represent unsuccessful LSODA integrations. Recall that the LSODA integrator is used to step the simulation along in time. If LSODA, while trying to calculate a time step, decides it can't perform enough time sub-steps to complete the time step, it returns an error. So the fine blue line (and all other random blue lines present) is a string of simulations that LSODA was not able to complete.

It is clear from these images that some of the fourfold chaotic rivers are broken—meaning they do not completely bridge one region of chaos to another. It is not known yet if any unbroken bridges exist. If in fact no unbroken bridges exist, then the overall chaotic topological genus of the system could be preserved.

4.4.2 Escape Time

Each Newtonian map (Figures 4.15 through 4.18) of the simulation time shows the same red outline, representing an extended orbit, between smooth and chaotic regions. The features on the PPN2 map which are consistent with their Newtonian counterpart also exhibit that red outline. In clear contrast, though, the fourfold chaotic rivers are not outlined in red. This would suggest, contrary to a possibility discussed in Section 4.3,

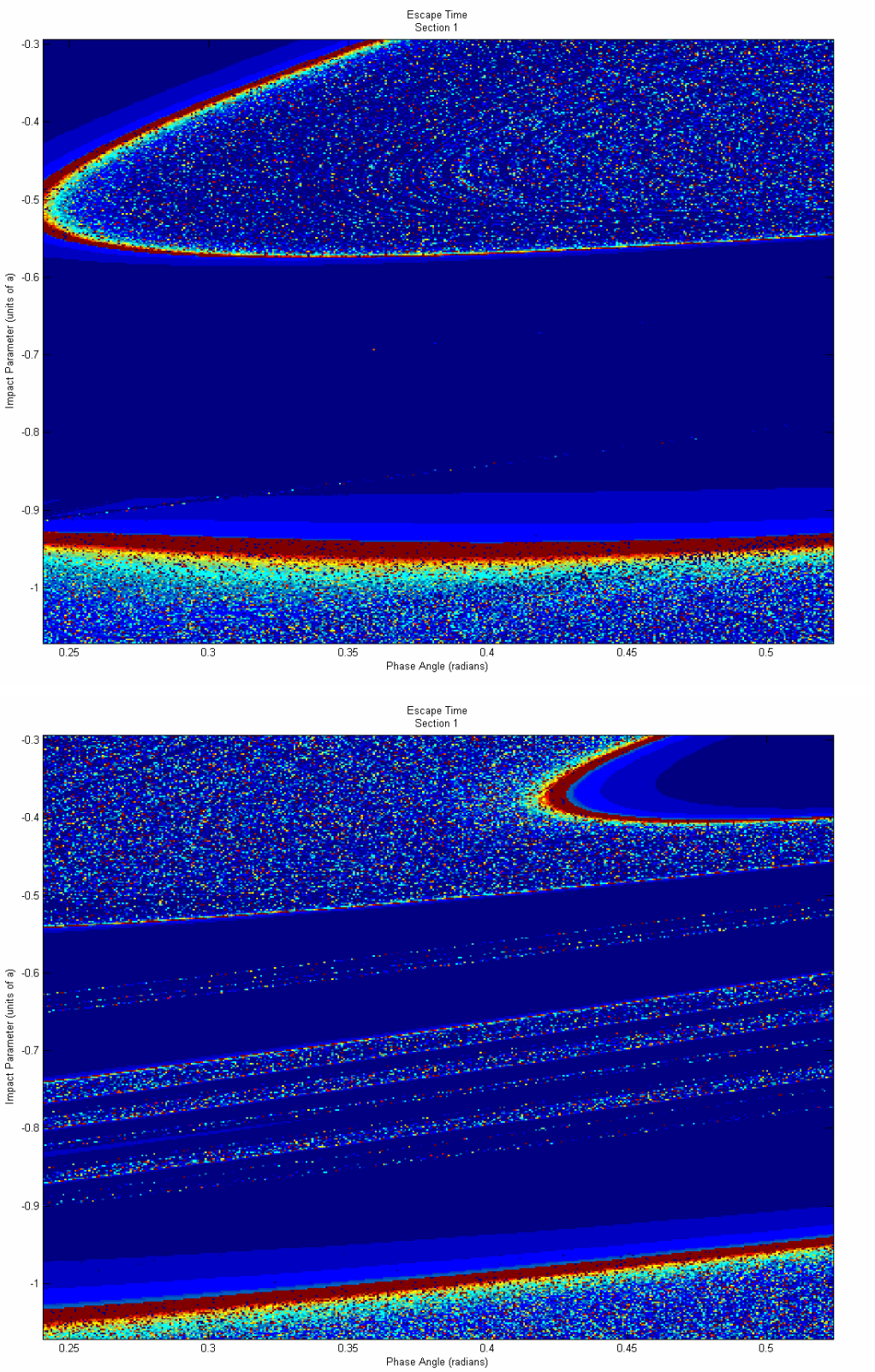

Figure 4.15 Escape Time of Region 1; Newtonian (top) [4.4x10^3, 6.4x10^6] and PPN2 (bottom) [3.2x10^3, 8.6x10^6]

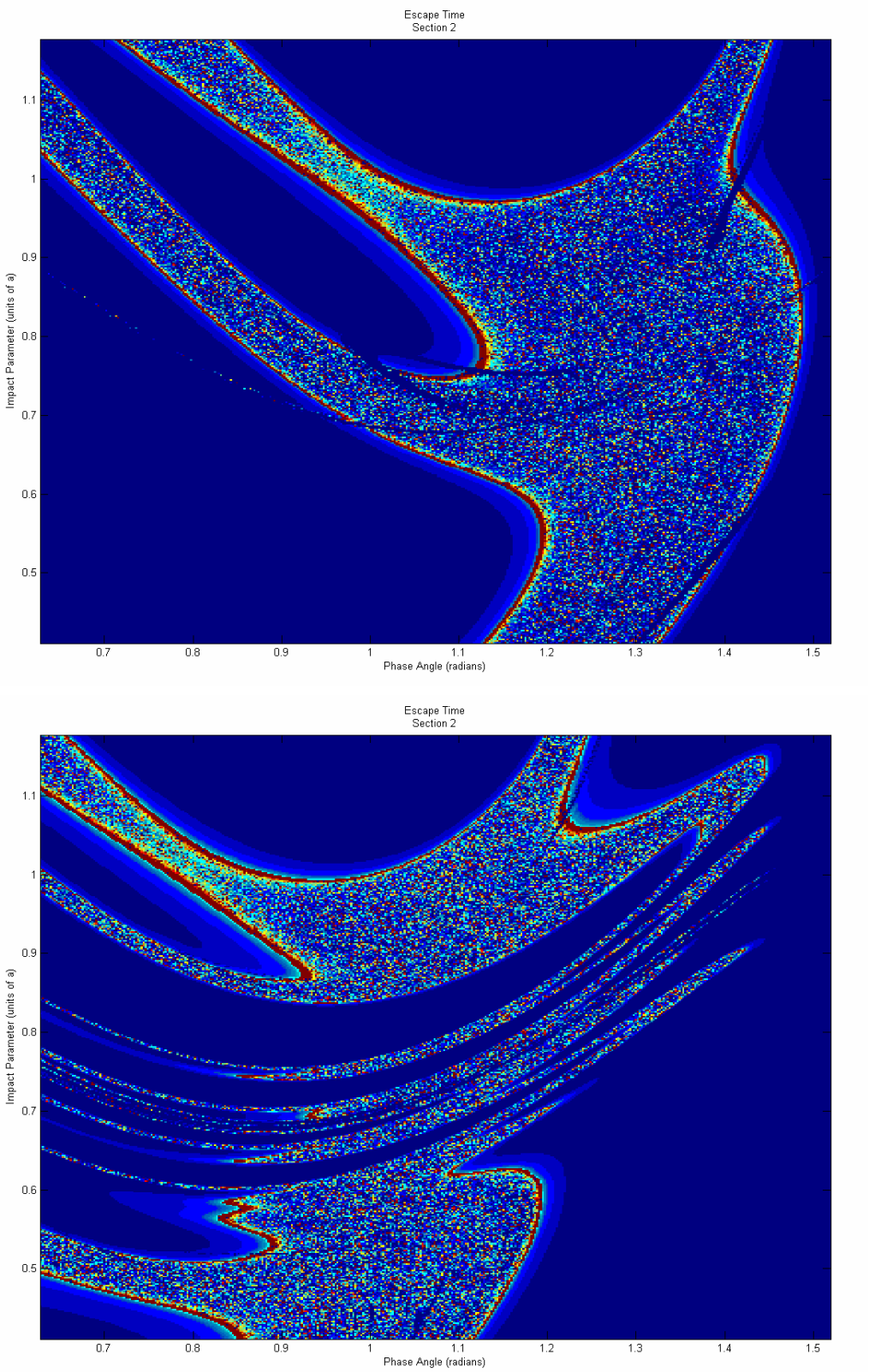

Figure 4.16 Escape Time of Region 2; Newtonian (top) [4.2x10^{3}, 1.0x10^{7}] and PPN2 (bottom) [3.2x10^{3}, 1.4x10^{7}]

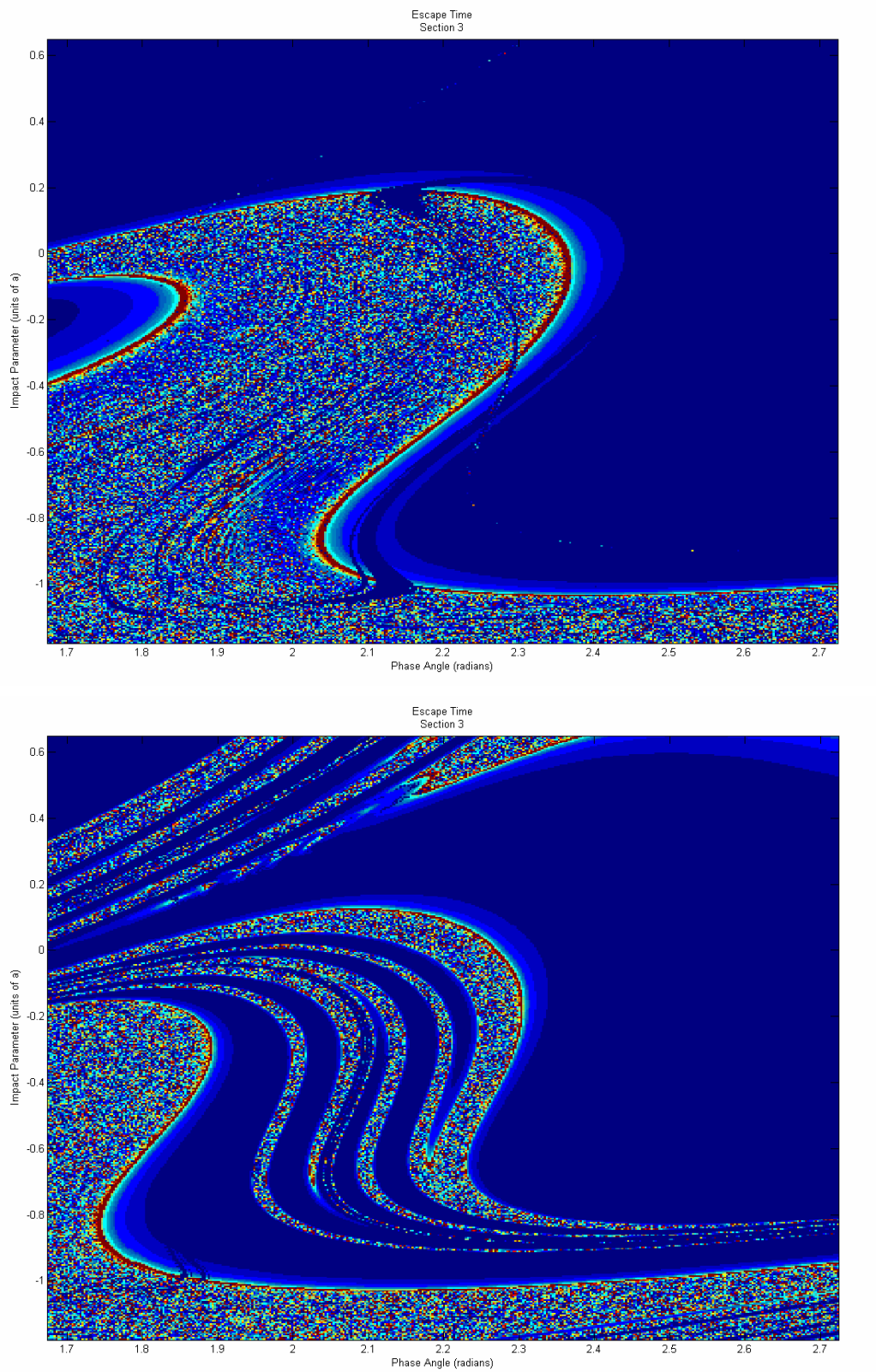

Figure 4.17 Escape Time of Region 3; Newtonian (top) [4.3x10^{3}, 1.4x10^{7}] and PPN2 (bottom) [3.1x10^{3}, 7.8x10^{6}]

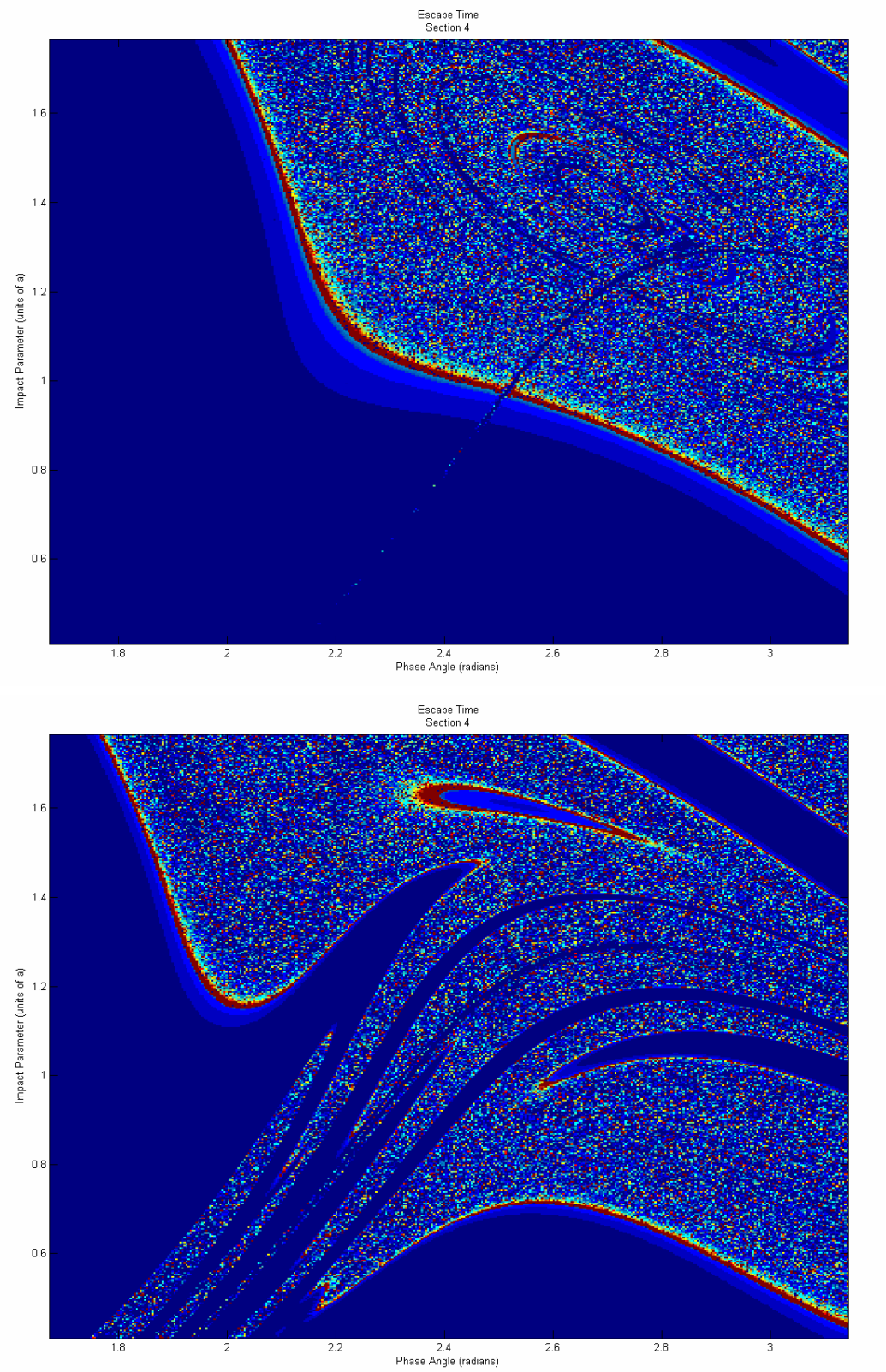

Figure 4.18 Escape Time of Region 4; Newtonian (top) [7.8x10^{3}, 1.5x10^{7}] and PPN2 (bottom) [3.1x10^{3}, 1.0x10^{8}]

that the chaotic topology is entirely changed, meaning the PPN2 fourfold rivers are not simply caused by a stretching of the smooth/chaotic boundaries already existing in the Newtonian map.

4.4.3 Change in Angular Momentum

The maps of the Change in Angular Momentum (Figures 4.19 through 4.22) offer more information regarding the fine structure of the three-body interaction. The smooth color gradients follow different contours than those from the Escape Angle maps. The smooth color gradients in these Angular Momentum maps run parallel to the smooth/chaotic boundary line. The smooth color gradients intersect the boundaries from all directions as though they do not influence one another. Thus, the Change in Angular Momentum gives additional information that some part of the smooth interactions is in fact influenced by the shape of the chaotic boundaries.

4.4.4 Minimum Approach Distance

The Minimum Approach Distance (Figures 4.23 through 4.26) exemplifies two more features of the three body interactions. The first feature was already mentioned in Section 4.3.4 and that is the ovals in the background. The ovals which permeate the background of the Minimum Approach Distance maps show up very clear in these maps.

The second feature is the fine structure within the chaotic regions in each of these maps. A clear example of this is the upper right quadrant of Figure 4.26 (top). It just so happens that the fine structure is comprised of concentric ovals (completely unrelated to the ovals previously mentioned). The chaotic wavy lines run parallel to the smooth/chaotic boundary.

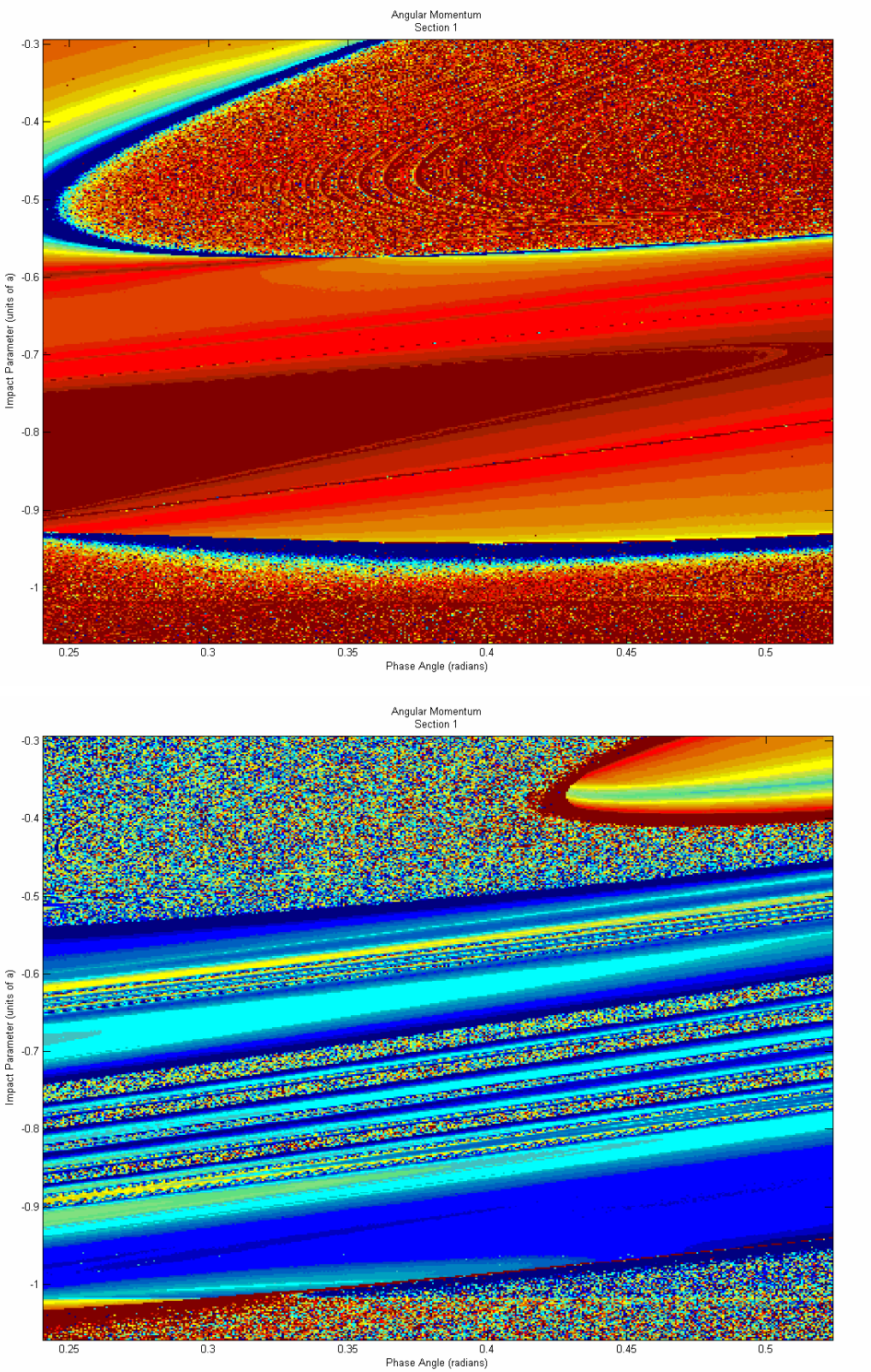

Figure 4.19 Change in Angular Momentum of Region 1; Newtonian (top) [-1.0x10^5, 9.8x10^4] and PPN2 (bottom) [-9.6x10^4, 1.2x10^5]

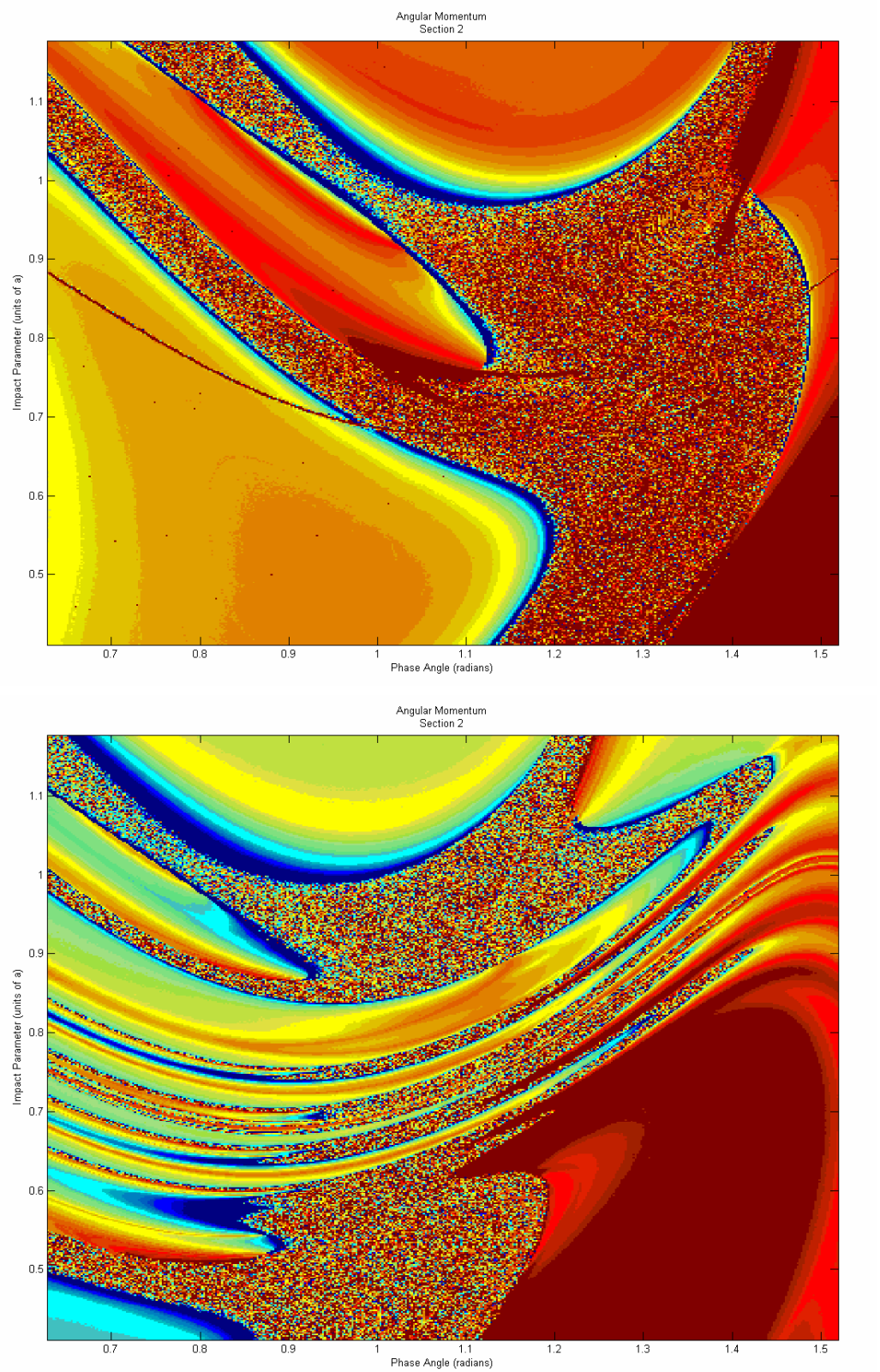

Figure 4.20 Change in Angular Momentum of Region 2; Newtonian (top) [-1.1x10^5, 2.5x10^5] and PPN2 (bottom) [-1.1x10^5, 2.4x10^5]

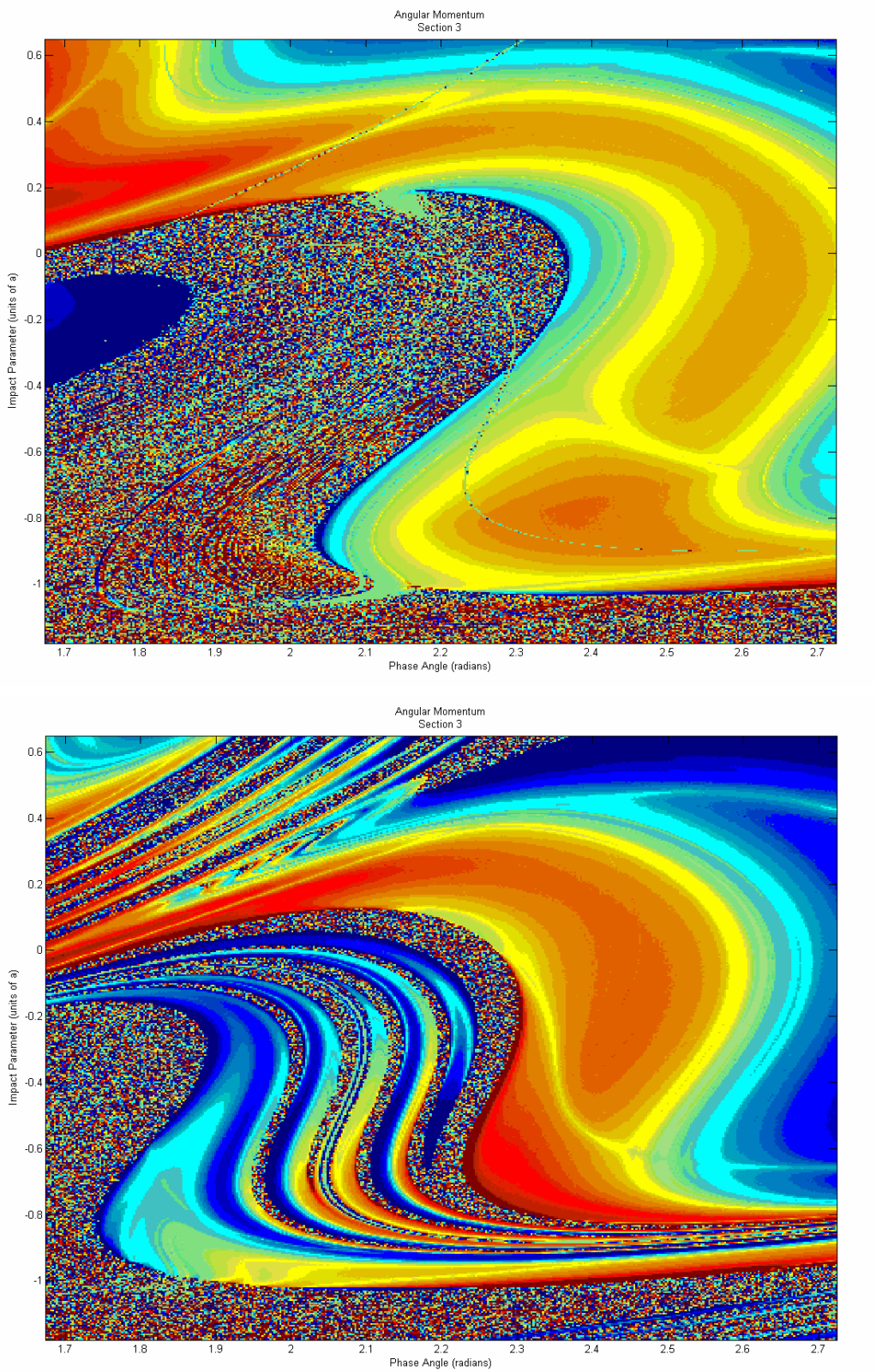

Figure 4.21 Change in Angular Momentum of Region 3; Newtonian (top) [-1.0x10^5, 1.4x10^5] and PPN2 (bottom) [-8.8x10^4, 1.4x10^5]

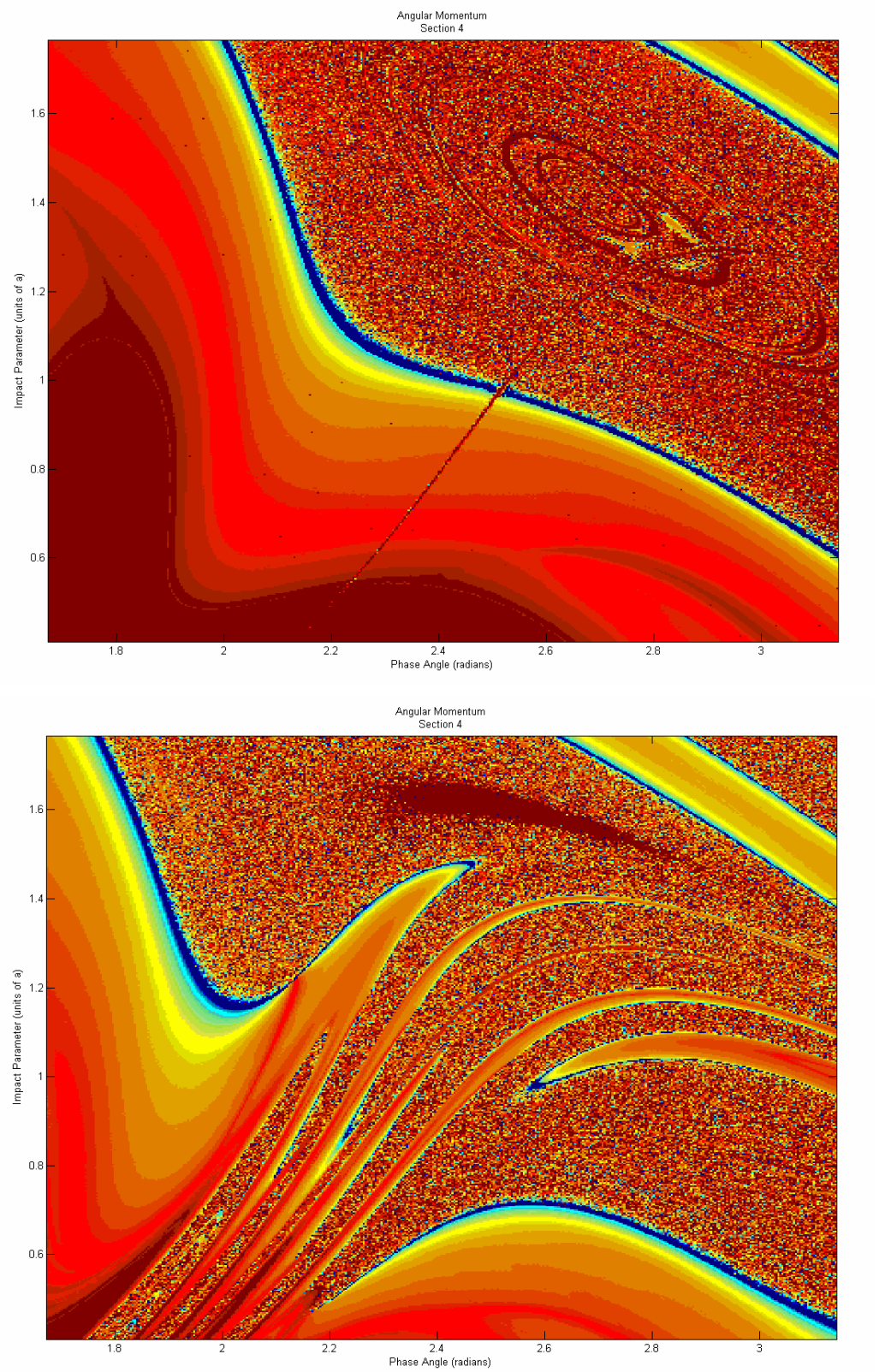

Figure 4.22 Change in Angular Momentum of Region 4; Newtonian (top) [-1.3x10^5, 2.5x10^5] and PPN2 (bottom) [-1.6x10^5, 2.5x10^5]

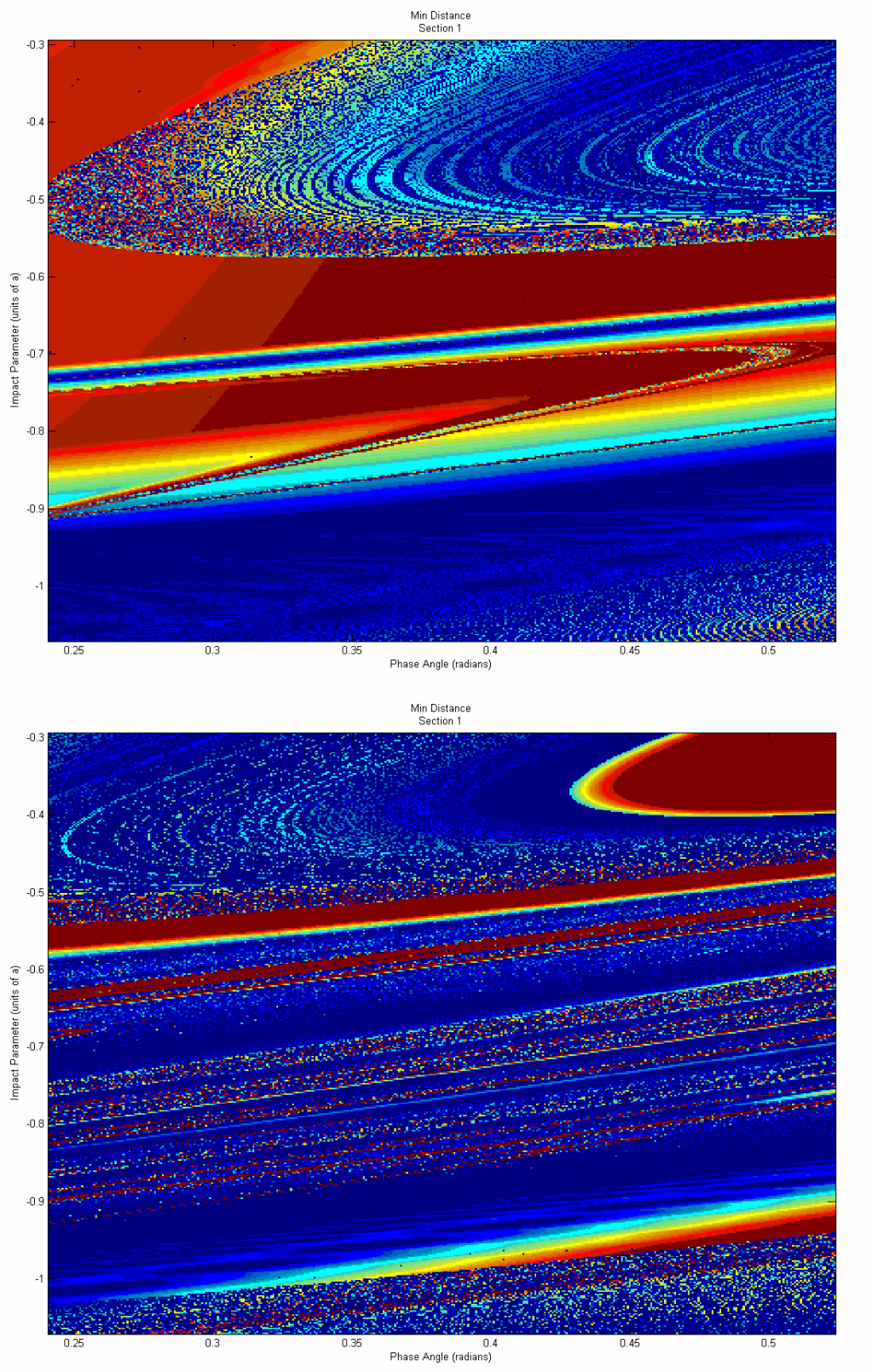

Figure 4.23 Minimum Approach Distance of Region 1; Newtonian (top) [$2.7x10^{-8}$, $6.1x10^{-3}$] and PPN2 (bottom) [$1.1x10^{-7}$, $6.3x10^{-3}$]

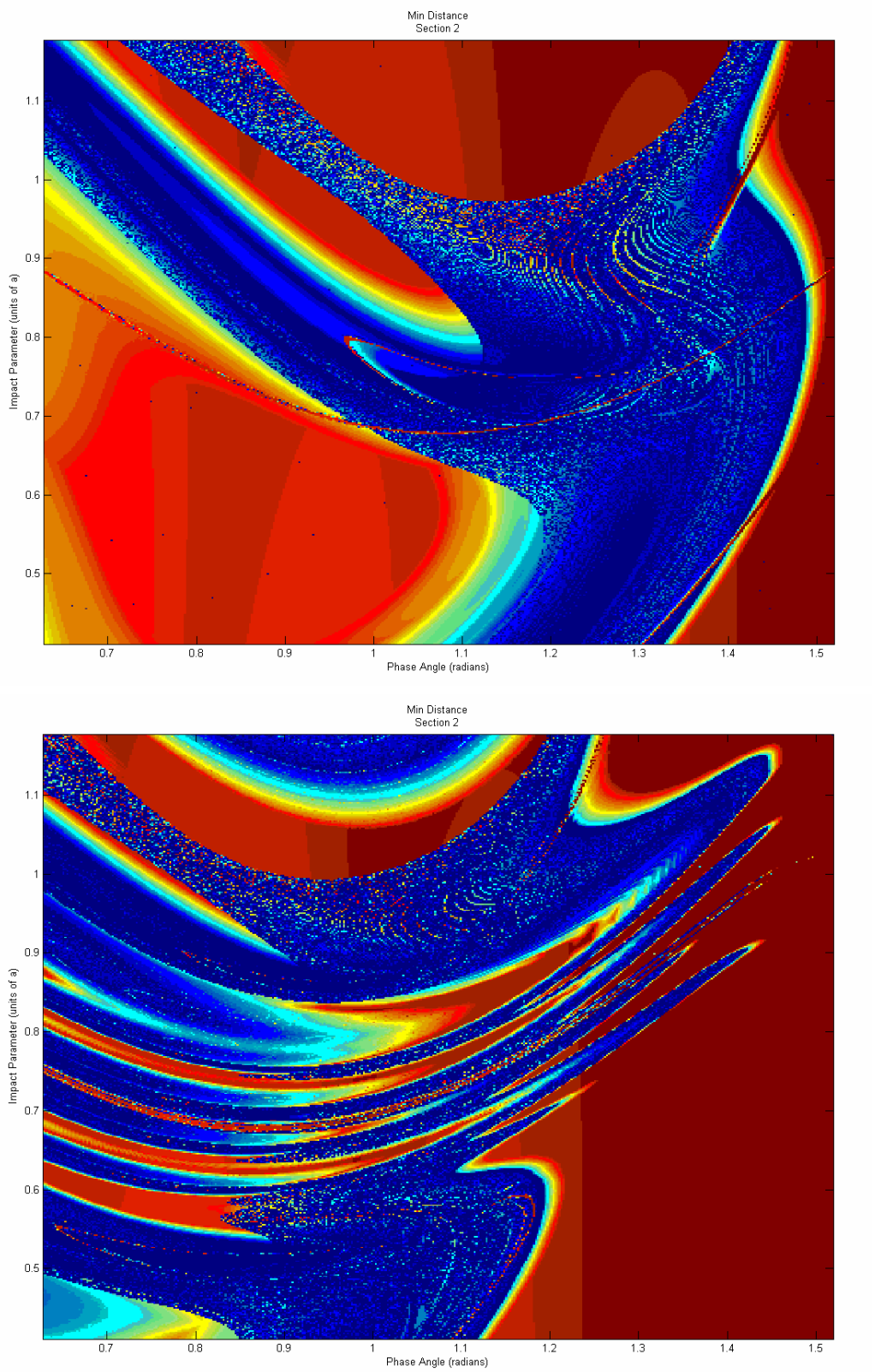

Figure 4.24 Minimum Approach Distance of Region 2; Newtonian (top) [3.7x10^{-7}, 6.7x10^{-3}] and PPN2 (bottom) [3.8x10^{-7}, 6.8x10^{-3}]

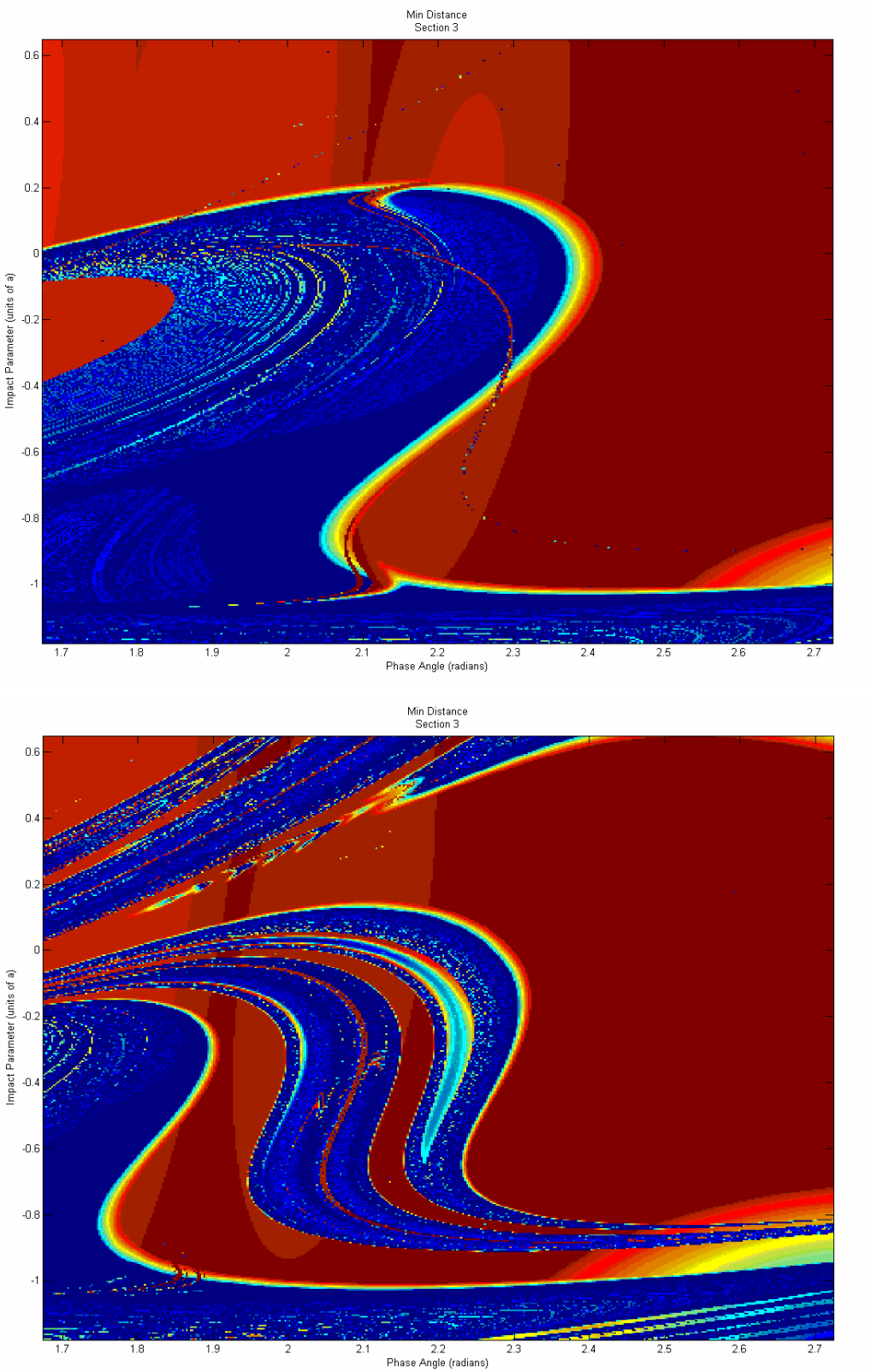

Figure 4.25 Minimum Approach Distance of Region 3; Newtonian (top) [1.6x10^{-8}, 7.1x10^{-3}] and PPN2 (bottom) [4.6x10^{-9}, 7.2x10^{-3}]

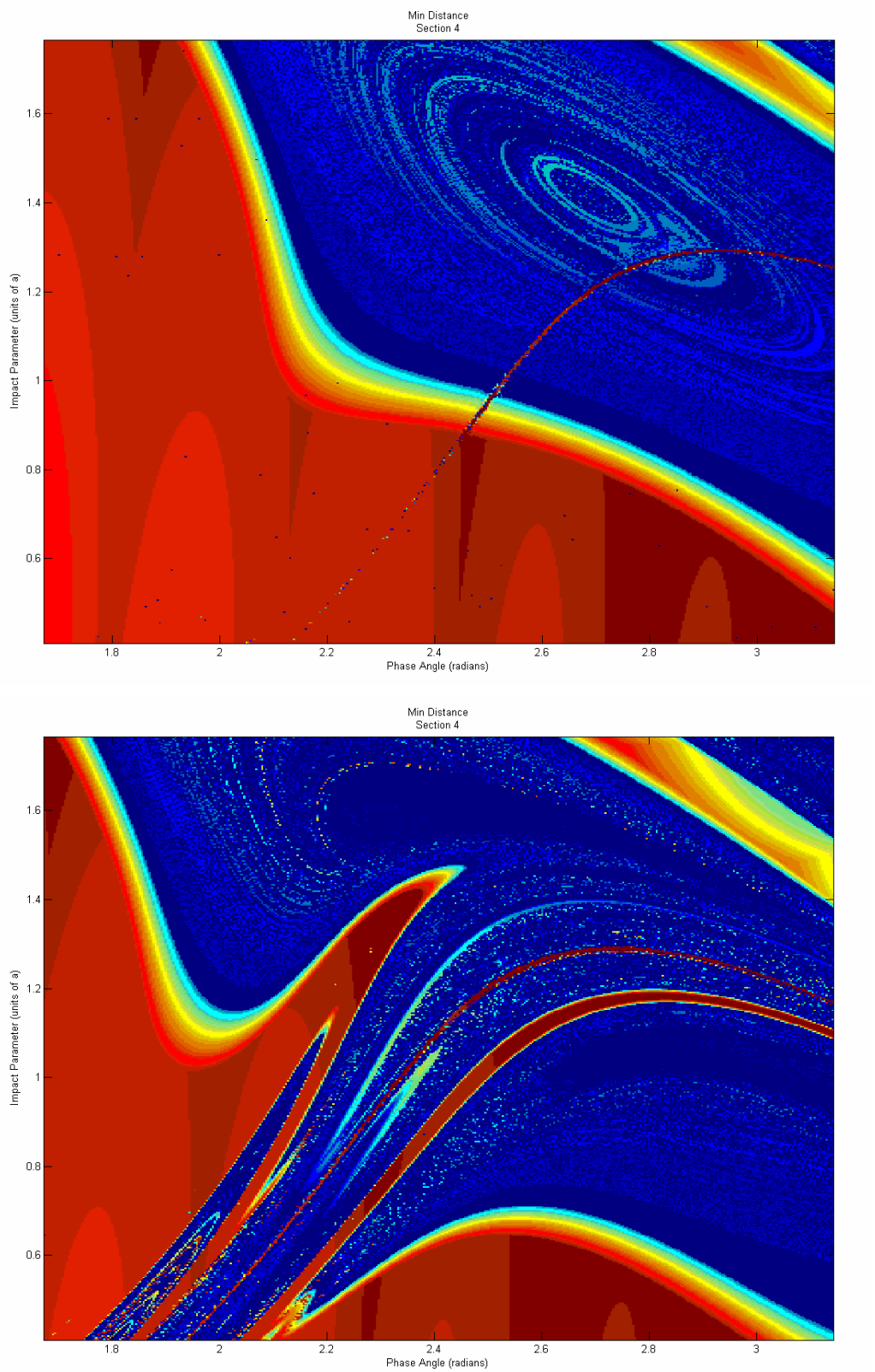

Figure 4.26 Minimum Approach Distance of Region 4; Newtonian (top) [1.3x10-7, 7.3x10-3] and PPN2 (bottom) [2.3x10-7, 7.3x10-3]

4.4.5 Maximum Momentum Dot Product

When viewed from up close, there appears in the Momentum Dot Product maps (Figures 4.27 through 4.30) a hybrid characteristic between smooth and chaotic. For example, refer to the Newtonian (upper) map of Figure 4.30. There is a red colored region which gradually blends into a yellow then blue as it disperses outward. But this gradual blending of colors is accompanied with many speckled dots, each representing a simulation with a drastically different maximum dot product. It seems dual-natured that the Maximum Dot Product could be generally determined according to its map coordinate and simultaneously be chaotic and unrelated to neighboring values. Perhaps its coordinate can smoothly determine value but only within a percentage.

4.4.6 Escaping Body

Another chaotic feature manifests itself in the Escaping Body maps (Figures 4.31 through 4.34)—a different body is escaping within the fourfold rivers than outside the rivers. First, to understand the coloring, green represents body one, orange represents body two, and red represents body three. In Region 1, body two escapes within the rivers whereas body one was escape in the Newtonian regime. In Region 2, body one escaped in the rivers and body three otherwise. In Region 3, bodies one and three escape within their respective rivers and body two escapes around them. In Region 4, body three escapes in the rivers whereas body two escapes outside the rivers.

The consistent change in escaping body within the new PPN2 rivers suggests that the change taking place in these four regions, for the PPN2 regime, is also consistent.

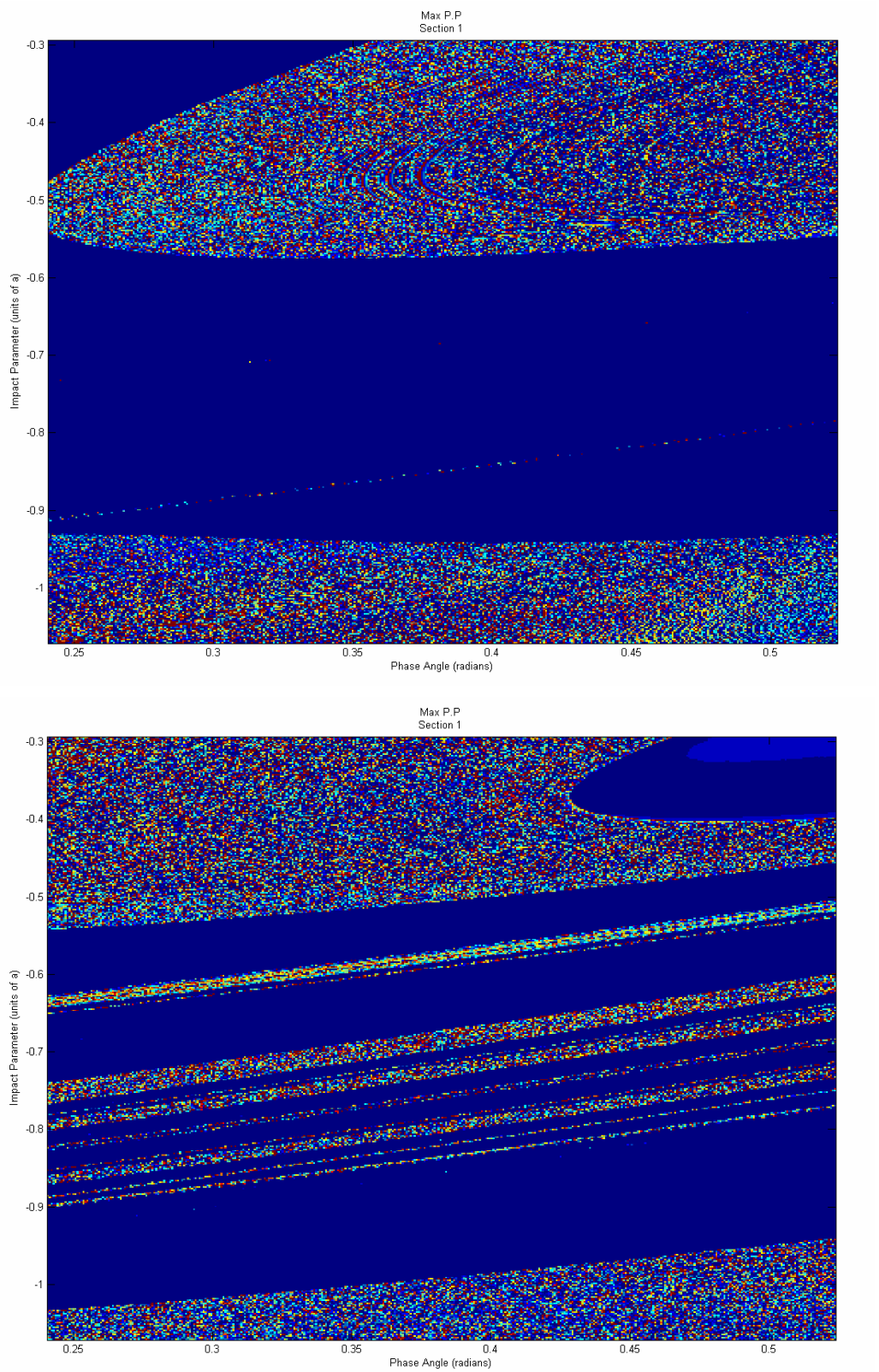

Figure 4.27 Max Momentum Dot Product of Region 1; Newtonian (top) [$9.0x10^6$, $3.8x10^{26}$] and PPN2 (bottom) [$9.0x10^6$, $2.9x10^{26}$]

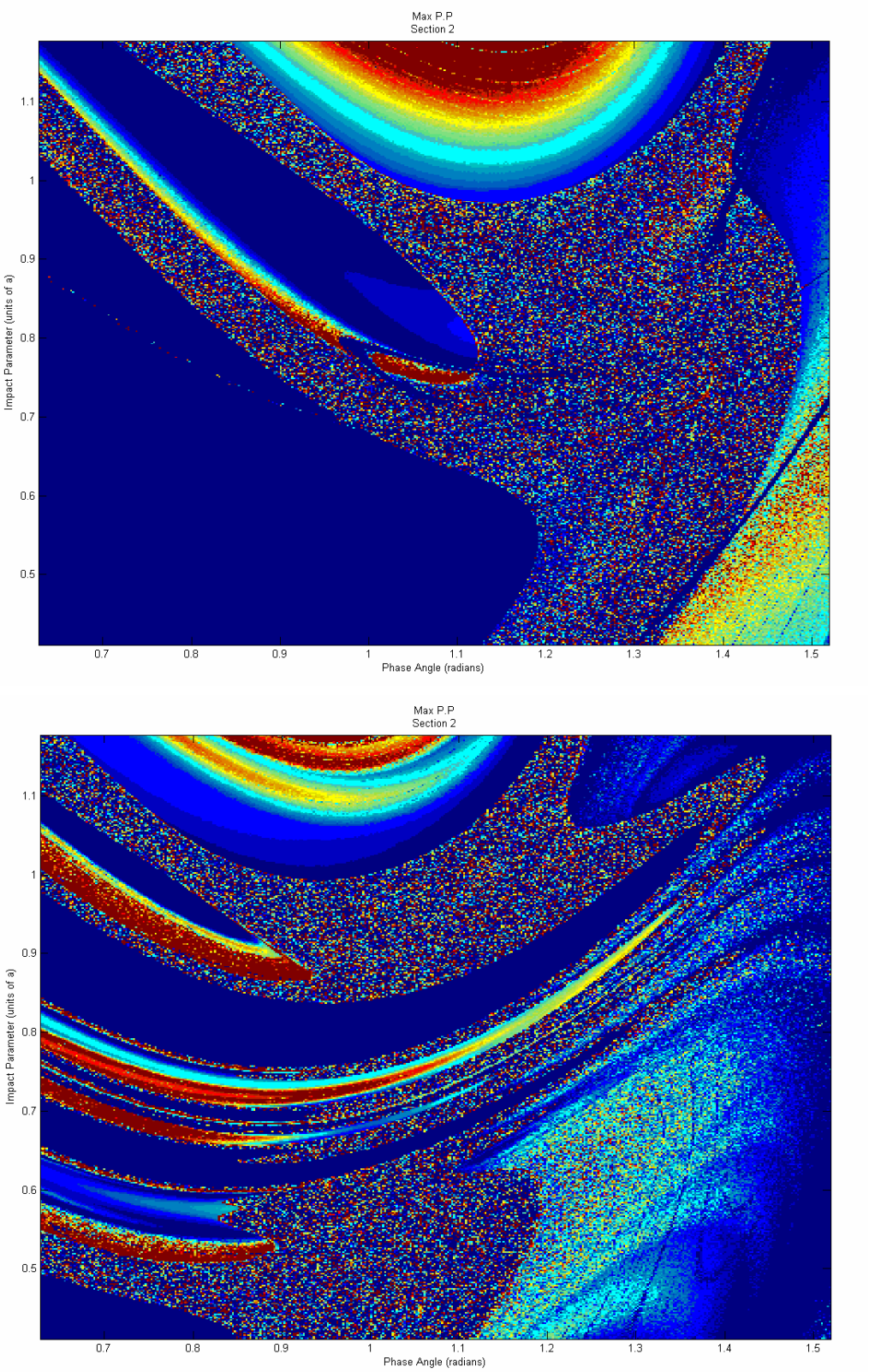

Figure 4.28 Max Momentum Dot Product of Region 2; Newtonian (top) [$9.0x10^{6}$, $5.3x10^{25}$] and PPN2 (bottom) [$3.7x10^{20}$, $1.2x10^{26}$]

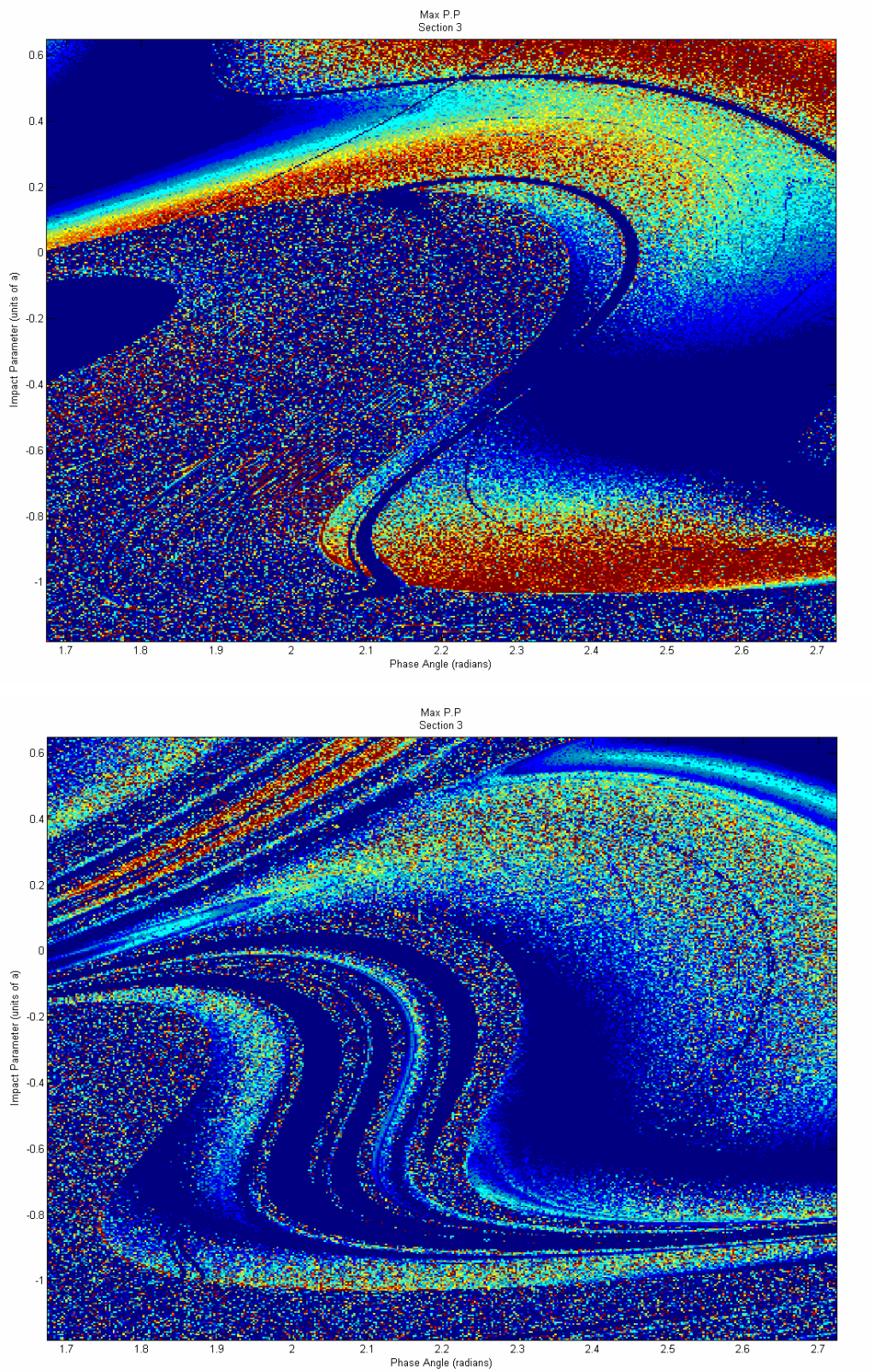

Figure 4.29 Max Momentum Dot Product of Region 3; Newtonian (top) [$9.0x10^{6}$, $1.9x10^{26}$] and PPN2 (bottom) [$2.0x10^{19}$, $6.3x10^{28}$]

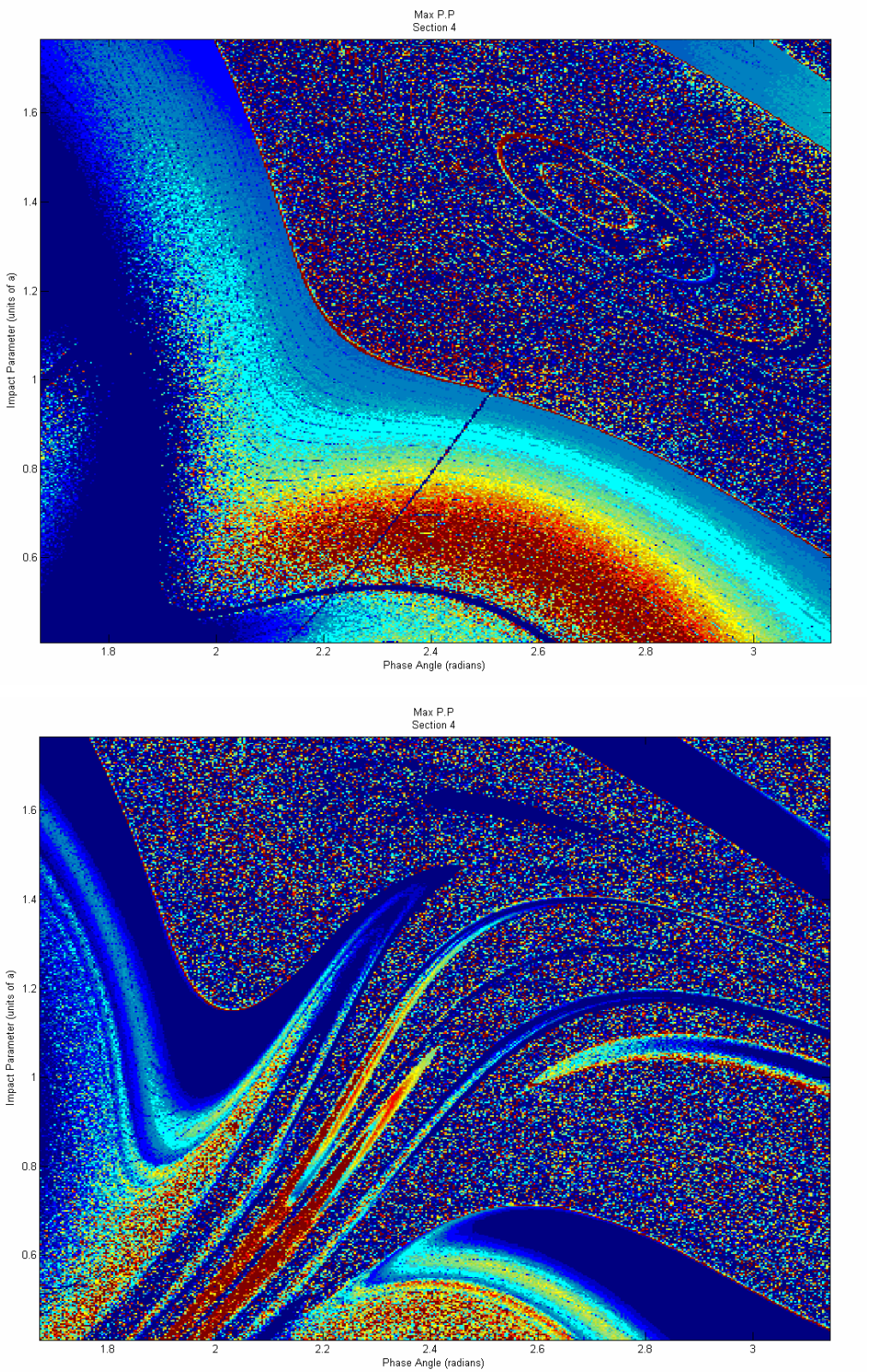

Figure 4.30 Max Momentum Dot Product of Region 4; Newtonian (top) [$9.0x10^{6}$, $2.6x10^{26}$] and PPN2 (bottom) [$3.8x10^{20}$, $2.0x10^{26}$]

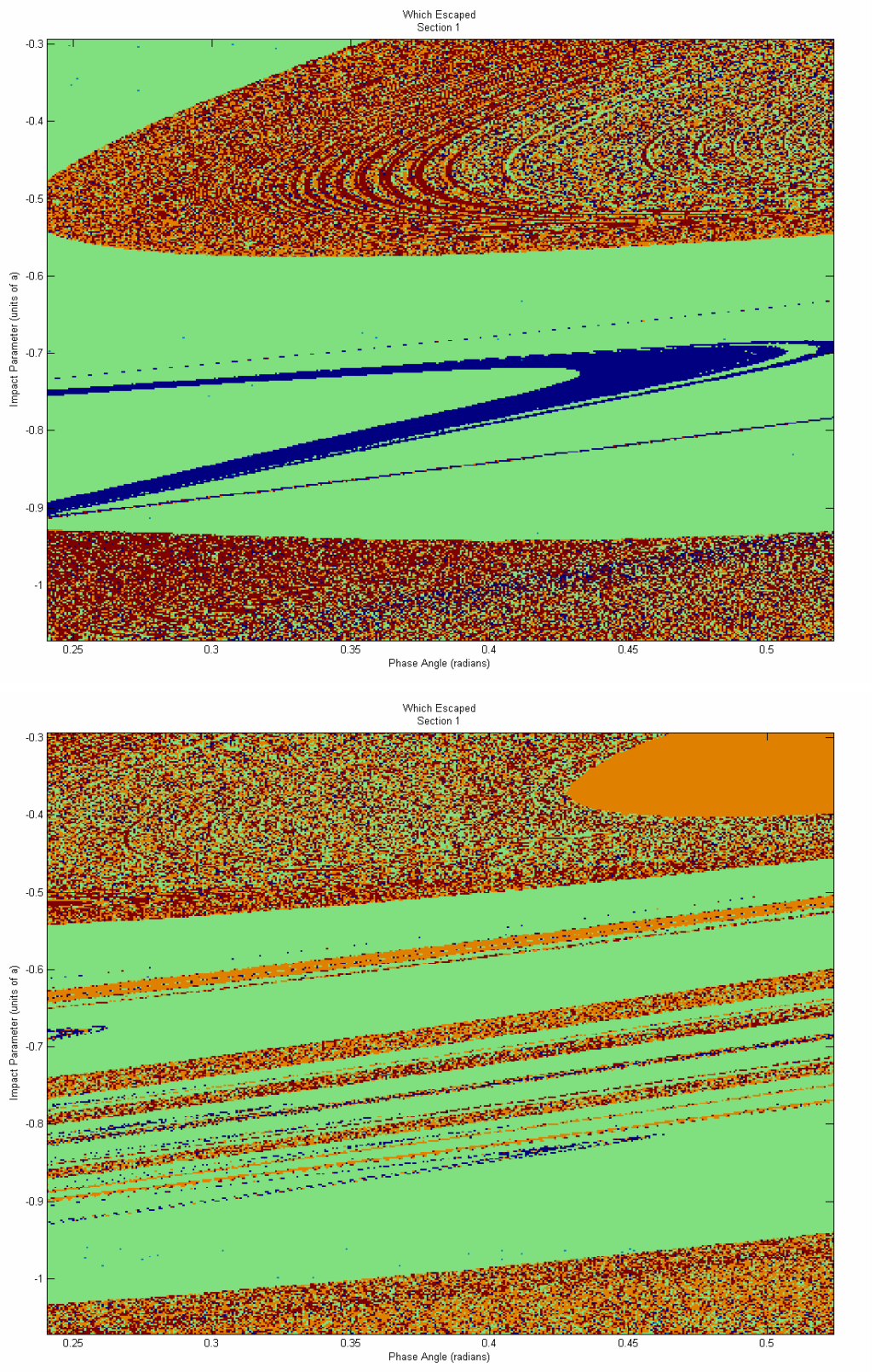

Figure 4.31 Escaping Body of Region 1; Newtonian (top) [1, 3] and PPN2 (bottom) [1, 3]

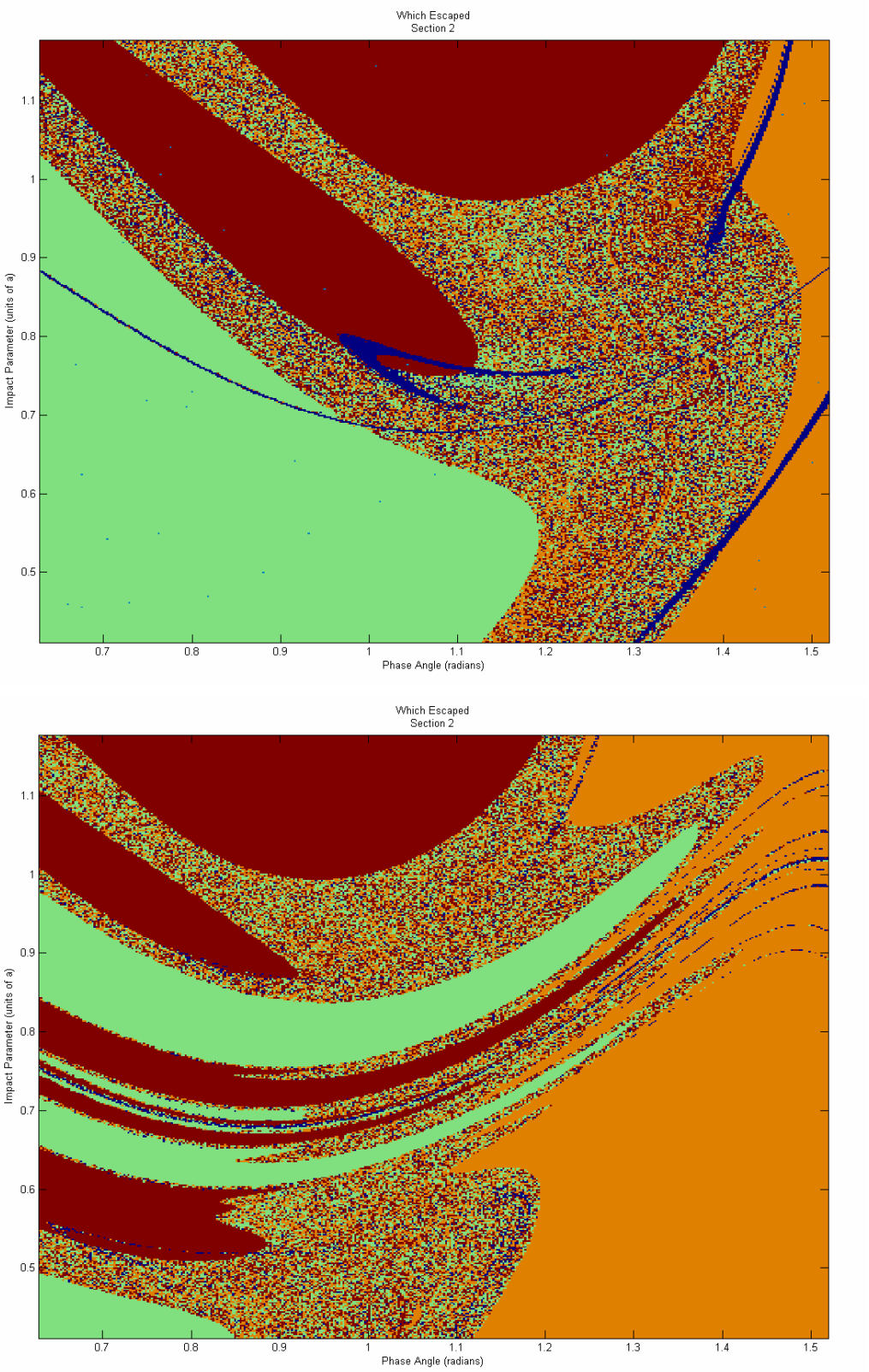

Figure 4.32 Escaping Body of Region 2; Newtonian (top) [1, 3] and PPN2 (bottom) [1, 3]

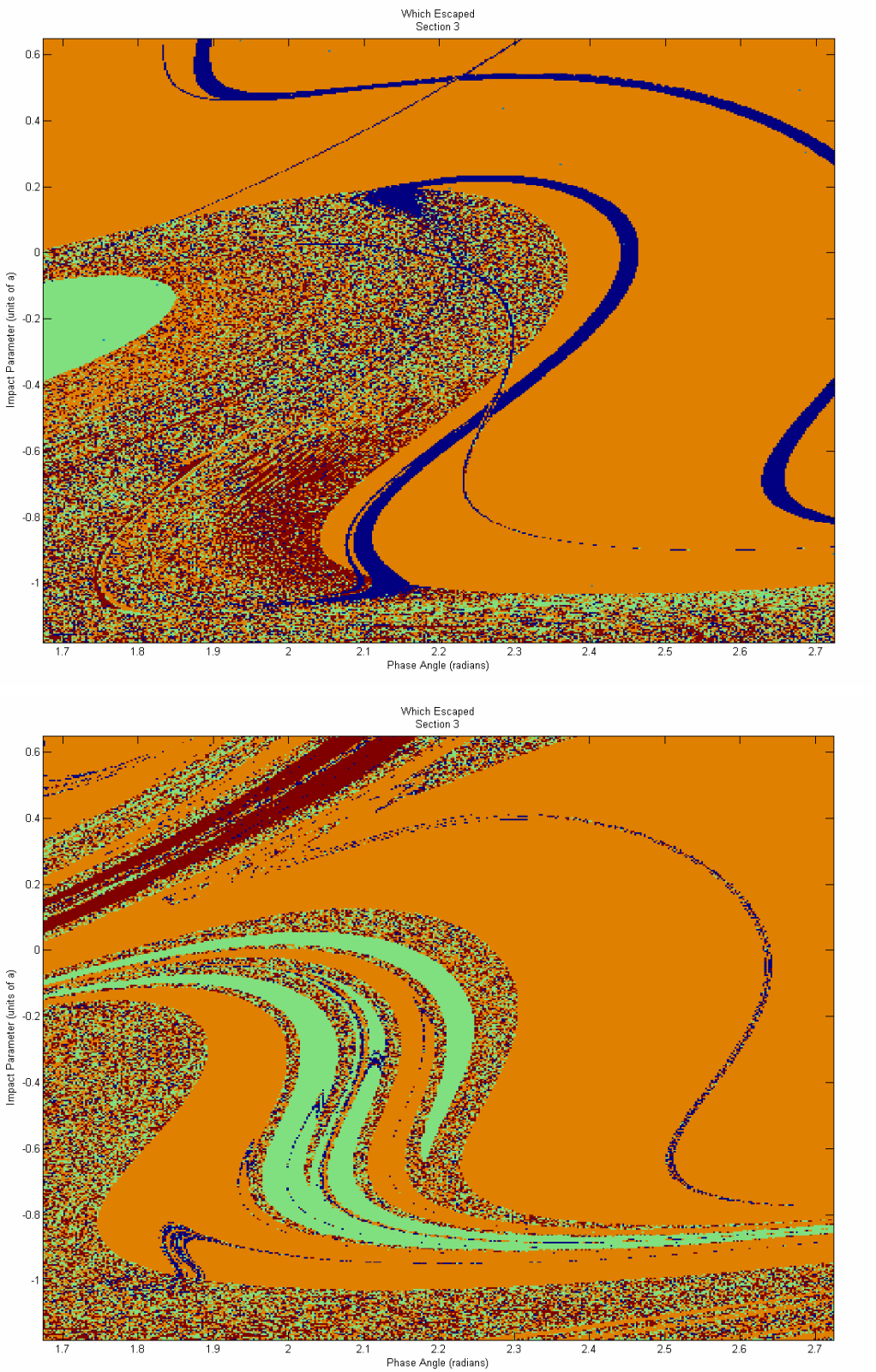

Figure 4.33 Escaping Body of Region 3; Newtonian (top) [1, 3] and PPN2 (bottom) [1, 3]

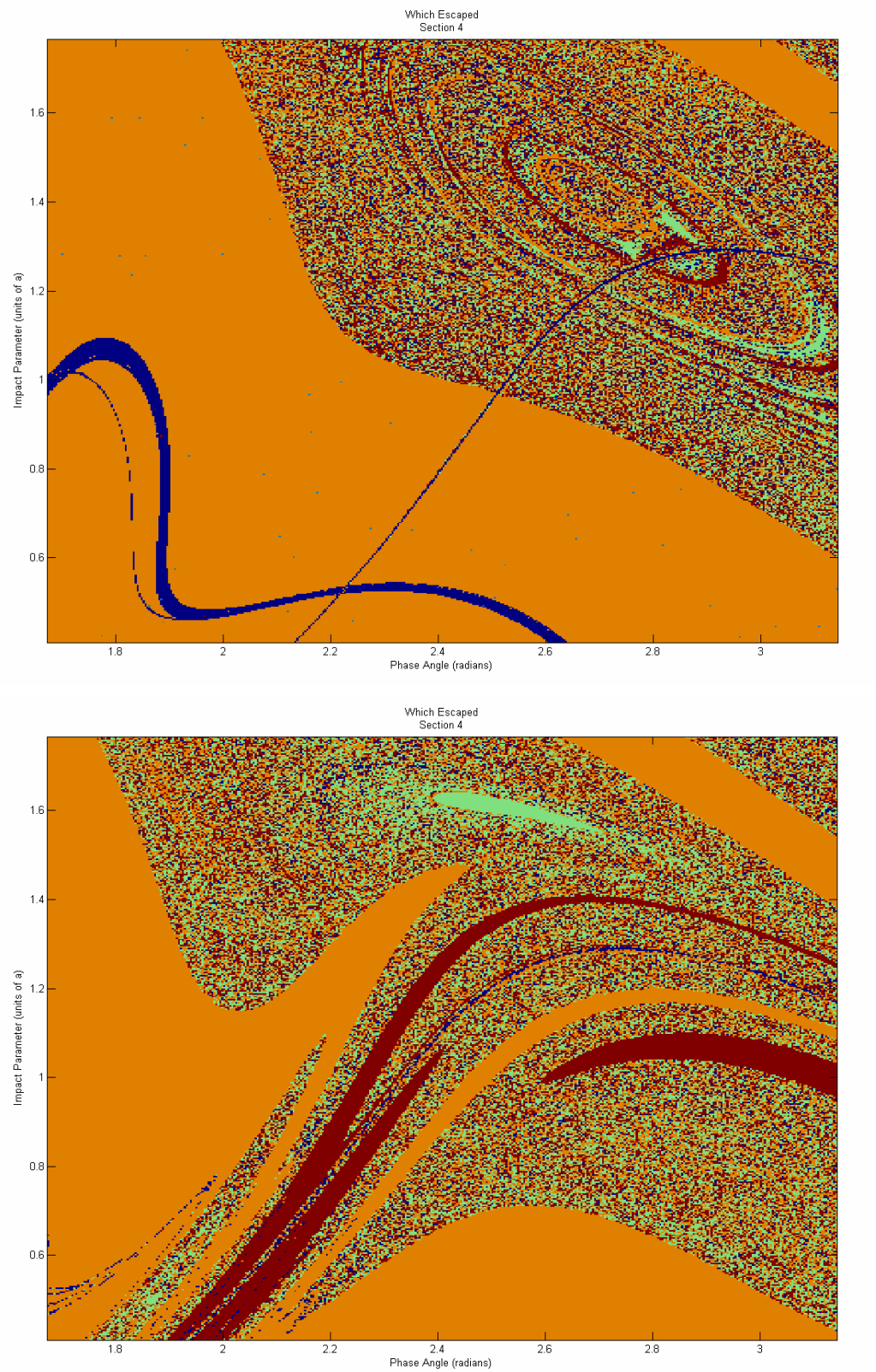

Figure 4.34 Escaping Body of Region 4; Newtonian (top) [1, 3] and PPN2 (bottom) [1, 3]

The way in which the interactions are changing between the Newtonian and PPN2 maps is constant enough that each manifests not only four to five chaotic rivers but a change in the escaping body between those chaotic rivers.

Chapter 5

Conclusion

5.1 Significance

The gravitational many-body problem has been very influential in the study of chaos. Chaos in the Newtonian problem has been studied extensively since the time of Poincaré; today, studies explore solar systems, extra solar planetary systems, and stellar clusters. Much less is known, however, about the many-body problem in general relativity. This paper explores similarities and differences between Newtonian and general relativity gravity as applied to the three body problem.

The complexity of solving the Einstein equations, for which even the Kepler problem must be solved numerically, is certainly the most significant obstacle in these studies. To make the problem tractable for current computational capabilities, I study a perturbative expansion of the Einstein equations, the post-Newtonian equations to second order. This is the first fully dynamic investigation of chaos in the three-body problem of general relativity that I am aware of.

5.2 Findings

The numerical experiments were conducted on a single family of initial conditions. I explored the existence of chaos for this system by observing qualitatively large differences in the outcome of different experiments of slightly varying initial data. The system consists of a binary pair in a circular orbit which interacts with an incoming third body. All bodies have equal mass and their motion is confined to a plane. The total energy of the system is negative to ensure that all three bodies can be simultaneously bound.

Several quantities were monitored during the evolution to characterize the system, including some conserved quantities for diagnostic purposes. The escape angle and identity of the escaping object are two coordinate independent measures of the system's final state. When making small changes in initial data, qualitatively large changes in the final outcome were sometimes observed. This sensitivity to initial conditions, present in both the Newtonian model and general relativity, is highly suggestive of chaos in the relativistic three-body problem as well as for Newtonian gravity (already known to be chaotic). However, I do not attempt to directly evaluate Lyaponuv exponents for the second order post-Newtonian approximation in this work.

The numerical experiments compare results using both Newtonian gravity and the second order post-Newtonian equations. The overall results are very similar suggesting that most of the interactions are non-relativistic. This is expected when velocities are small, compared to the speed of light, and separations are relatively large, compared to the event horizon. There are some marked differences in certain regions of the initial data space; these were discussed extensively in Chapter Four. Thus, the three-body

problem in general relativity apparently shares the same chaotic behavior known in Newtonian gravity, at least in the post-Newtonian limit.

5.3 Future Research

While this study has found evidence of chaos in general relativity for three bodies, several interesting questions remain which will be the focus of future research. As one direction for future research, different regions of the parameter space can be explored. For example, the bodies can be given different masses, the motion can be expanded into three dimensions, or very different families of initial data can be investigated. A second direction of research would be to analyze the effect of the expansion order for the post-Newtonian equations. The work in this study, for example, did not include the first order post-Newtonian equations. Initial work suggested some interesting divergences from the second order equations. Furthermore, additional physics could be added to the system by including higher order radiation terms, such as the emission of gravitational waves in the second and a half post-Newtonian order. A third area for additional research might focus on numerical algorithms and their effect on the calculated solutions. For efficiency I used an adaptive Adams-Bashforth-Moulton integrator to solve the differential equations in this work. Improvements in the numerical algorithms, including some suggested benefits of symplectic integrators, have not been examined in this thesis. Each avenue holds great promise of adding to the cumulative understanding of Einstein's equations of general relativity and the role chaos plays therein.

Bibliography

[1] Wikipedia contributors, “Mandelbrot set,” http://en.wikipedia.org/wiki/Mandelbrot_set (Accessed January 11, 2007).

[2] P. Hut, “Three-Body Problem: Encounters, ” http://www.ids.ias.edu/~piet/act/astro/three/ (Accessed July 05, 2007).

[3] J. Barrow-Green, *Poincare and the Three Body Problem*, (American Mathematical Society, London, 1997).

[4] J. J. Lissauer, “Chaotic motion in the Solar System,” Reviews of Modern Physics 71 (3), 835—845 (1999).

[5] C. Chicone, B. Mashhoon, D. G. Retzloff, “Chaos in the Kepler System,” Classical and Quantum Gravity 16, 507—527 (1999).

[6] K. M. Hock, “Chaos in the relativistic Euler problem,” Chaos 6 (4), 564—567 (1996).

[7] J. Hietarinta, S. Mikkola, “Chaos in the one-dimensional gravitational three-body problem,” Chaos 3 (2), 183—203 (1993).

[8] F. Burnell, J. J. Malecki, R. B. Mann, “Chaos in an exact relativistic three-body self-gravitating system,” Physical Review E 69, 016214-1—30 (2004).

[9] P. T. Boyd, S. L. W. McMillan, “Chaotic scattering in the gravitational three-body problem,” Chaos 3 (4), 507—523 (1993).

[10] G. Schäfer, “Three-body Hamiltonian in General Relativity,” Phys. Lett. A 123 (7), 336—339 (1987).

Appendix

run.pl *-drives nbodyPN for many different initial conditions*

```
#!/usr/bin/perl
$pi = 3.1415926535897;
$a  = 1.7;
#==================================
# nbody driver for 3 chaotic bodies
#==================================

#----------------------------------
# Initialize variables
#----------------------------------

$whichorder = 1;
$runnumber = 206;

$phiMin = 0.5333*$pi;
$phiMax = 1*$pi;
$phiNum = 150;

$rhoMin = 0.7*$a;
$rhoMax = 3*$a;
$rhoNum = 150;

#initialize time variables
$tFinal = 100000000;
$tStep = 20;
$which = 3;

#initialize file variables
$stdFileBase = "stdout";
$ICFile = "ICFile.$runnumber";
$errFileBase = "errout";

#initialize constants
$m1 = 1e7;
$m2 = 1e7;
$m3 = 1e7;
$mT = $m1+$m2+$m3;
$G = 6.673e-11;
$vc = sqrt($G*$m1*($m1+$m2+$m3)/(2*$a*$m3));
$v = sqrt($G*$m1/2/$a);
```

```
#initialize phi range
if($phiNum < 2)
{
      $dphi = 0;
}
else
{
      $dphi = ($phiMax-$phiMin)/($phiNum-1);
}

#initialize rho range
if($rhoNum < 2)
{
      $drho = 0;
}
else
{
      $drho = ($rhoMax-$rhoMin)/($rhoNum-1);
}
$totalNum = $rhoNum*$phiNum;

#-----------------------------------
# Drive nbody with ICs
#-----------------------------------

for($r=0; $r<$rhoNum ; $r=$r+1 )#iterate over rho
{
      for($p=0; $p<$phiNum ; $p=$p+1)#iterate over phi
      {
            #initialize parameters
            $rho=$rhoMin+$r*$drho;
            $phi=$phiMin+$p*$dphi;

            #print "Evolving: ($r, $p)\n";

            #initialize positions
            $x1=($a/2)*cos($phi);
            $y1=($a/2)*sin($phi);
            $z1=0;
            $x2=-($a/2)*cos($phi);
            $y2=-($a/2)*sin($phi);;
            $z2=0;
            $x3=30*$a;
            $y3=$rho;
            $z3=0;

            #initialize velocities
            $vx1=-$v*sin($phi);
            $vy1= $v*cos($phi);
            $vz1=0;
            $vx2= $v*sin($phi);
            $vy2=-$v*cos($phi);
            $vz2=0;
            $vx3=-0.5*$vc;
            $vy3=0;
            $vz3=0;
```

```
            #make transformations to center-o-mass,momentum
            $CX=($m1*$x1+$m2*$x2+$m3*$x3)/$mT;
            $CY=($m1*$y1+$m2*$y2+$m3*$y3)/$mT;
            $CZ=($m1*$z1+$m2*$z2+$m3*$z3)/$mT;

            $CVX=($m1*$vx1+$m2*$vx2+$m3*$vx3)/$mT;
            $CVY=($m1*$vy1+$m2*$vy2+$m3*$vy3)/$mT;
            $CVZ=($m1*$vz1+$m2*$vz2+$m3*$vz3)/$mT;

            #impliment position transformations
            $x1=$x1-$CX;
            $y1=$y1-$CY;
            $z1=$z1-$CZ;
            $x2=$x2-$CX;
            $y2=$y2-$CY;
            $z2=$z2-$CZ;
            $x3=$x3-$CX;
            $y3=$y3-$CY;
            $z3=$z3-$CZ;

            #impliment velocity transormations
            $vx1=$vx1-$CVX;
            $vy1=$vy1-$CVY;
            $vz1=$vz1-$CVZ;
            $vx2=$vx2-$CVX;
            $vy2=$vy2-$CVY;
            $vz2=$vz2-$CVZ;
            $vx3=$vx3-$CVX;
            $vy3=$vy3-$CVY;
            $vz3=$vz3-$CVZ;

            #make IC file
            open(IC, ">$ICFile");
            printf IC "$m1 $x1 $y1 $z1 $vx1 $vy1 $vz1\n";
            printf IC "$m2 $x2 $y2 $z2 $vx2 $vy2 $vz2\n";
            printf IC "$m3 $x3 $y3 $z3 $vx3 $vy3 $vz3\n";
            close (IC);

            #drive nbody
            system("./nbodyPN $tFinal $tStep $G 0 $runnumber < $ICFile >
$stdFileBase ");
            system("./nbodyPN $tFinal $tStep $G 2 $runnumber < $ICFile >
$stdFileBase ");
      }

}
```

Grapher.m *-generates the colored figures using data from nbodyPN*

```
clc; clear; close all;
%-------------------------------------
%   load file
%-------------------------------------
fileName = 'results.2.9992.txt';
resolution = 0;%0-ignore; 1-close coupled values; 2-looser coupled
values
fineTuner = 1;%0 < fT >= 1
findErrors = 1;%1=true, 0=false
rhoNum = 0;
phiNum = 0;
fprintf('Loading File...\n');
data = load(fileName);
lines = max(size(data));
numVars = min(size(data))-2;
averages = 1:numVars;
averages(:) = averages(:)*0;
upperBound = averages;
oldUpperBound = averages;
lowerBound = averages;
oldLowerBound = averages;
a = 1.7;
m = 1e7;
G = 6.73e-11;
initialTotalEnergy = -G*(3/8)*m^2/a;
fprintf('Initial Total Energy: %g\n', initialTotalEnergy);

%-------------------------------------
%   determine dimensions
%-------------------------------------
fprintf('Analyzing Dimensions...\n');
tempPhiNum=1;
rhoNum=1;
for l=2:lines
    if abs(data(l,1)-data(l-1,1))>1e-8
        rhoNum = rhoNum+1;
        if tempPhiNum>phiNum
            phiNum=tempPhiNum;
        end
        tempPhiNum=0;
    else
        tempPhiNum=tempPhiNum+1;
    end
end

%-------------------------------------
%   Establish variable ranges (for normalization)
%-------------------------------------
fprintf('Finding Statistical Values...\n')
maxVals=1:numVars;
maxVals(:) = -10000*maxVals(:);
minVals=1:numVars;
minVals(:) = 10000*minVals(:);
line = 0;
```

```
for r = 1:rhoNum
    for p = 1:phiNum
        line = line + 1;
        for v = 1:numVars
            if (data(line,v+2) == 0 && findErrors)
                data(line,v+2) = 900000;
            end
            if (data(line,v+2) > maxVals(v) && data(line,v+2) ~= -99 &&
data(line,v+2) ~= 0)
                maxVals(v) = data(line,v+2);
            end
            if (data(line,v+2) < minVals(v) && data(line,v+2) ~= -99 &&
data(line,v+2) ~= 0)
                minVals(v) = data(line,v+2);
            end
            averages(v) = averages(v) + data(line, v+2);
        end
    end
end
%set the extreem values---------------
averages(:) = averages(:)/(rhoNum*phiNum);
oldUpperBound(:) = averages(:);
oldLowerBound(:) = averages(:);
numAbove = 1:numVars;
numBelow = 1:numVars;
for resolutionIteration = 1:resolution
    upperBound(:) = 0;
    lowerBound(:) = 0;
    numAbove(:) = 0;
    numBelow(:) = 0;
    line = 0;
    for r = 1:rhoNum
        for p = 1:phiNum
            line = line + 1;
            for v = 1:numVars
                if (data(line,v+2) > oldUpperBound(v))
                    upperBound(v) = upperBound(v) + data(line,v+2);
                    numAbove(v) = numAbove(v)+1;
                    %fprintf('%g: %g >
%g\n',numAbove,data(line,v+2),oldUpperBound(v));
                end
                if (data(line,v+2) < oldLowerBound(v))
                    lowerBound(v) = lowerBound(v) + data(line,v+2);
                    numBelow(v) = numBelow(v)+1;
                end
            end
        end
    end
    oldUpperBound(:) = upperBound(:)./(numAbove(:)+1);
    oldLowerBound(:) = lowerBound(:)./(numBelow(:)+1);
end
upperBound(:) = oldUpperBound(:);
lowerBound(:) = oldLowerBound(:);
%variable names----------------------
if numVars == 7 vars = ['Escape Angle    '; 'Total Energy    ';
'Kinetic Energy  '; 'Potential Energy'; 'Linear Momentum '; 'Angular
Momentum'; 'Escape Time     '];
```

```
elseif numVars == 6 vars = ['Which Escaped'; 'Escape Angle '; 'Escape
Time  '; 'Escape KE    '; 'Escape PE    '; 'Num Bounds   '];
elseif numVars == 11 vars = ['Escape Angle    ';'Total Energy
';'Escape KE       ';'Escape PE       ';'Momentum        ';'Angular
Momentum';'Escape Time     ';'Max KE          ';'Max P.P         ';'Min
Distance    ';'Min Total Dist  '];
elseif numVars == 13 vars = ['Escape Angle    ';'Total Energy
';'Escape KE       ';'Escape PE       ';'Momentum        ';'Angular
Momentum';'Escape Time     ';'Max KE          ';'Max P.P         ';'Min
Distance    ';'Min Total Dist  ';'Fourth Term     ';'Which Escaped
'];
else vars = null; fprintf('UNRECOGNIZED VARIABLE CONFIGURATION');
end
variables = cellstr(vars);
%print variable ranges----------------
for n = 1:numVars
    fprintf([variables{n},': %g -(%g)-<%g>-(%g)->
%g\n'],minVals(n),lowerBound(n),averages(n),upperBound(n),maxVals(n));
end

%-------------------------------------
%initialize rho
%-------------------------------------
fprintf('Initializing Variables...\n')
rhoMax = max(data(:,1))/a;
rhoMin = min(data(:,1))/a;
drho = (rhoMax - rhoMin)/(rhoNum-1);
rho = rhoMin:drho:rhoMax;

%-------------------------------------
%initialize phi
%-------------------------------------
phiMax = max(data(:,2));
phiMin = min(data(:,2));
dphi = (phiMax - phiMin)/(phiNum-1);
phi = phiMin:dphi:phiMax;

%-------------------------------------
%initialize vars
%-------------------------------------
var = 1:numVars;
errorRange = 1:100;
errorArray = zeros(100);

%-------------------------------------
%generate grid
%-------------------------------------
fprintf('Generating Grid...\n')
[Var,Rho,Phi] = ndgrid(var,rho,phi);
[RHO,PHI] = ndgrid(rho,phi);
matrixArray = 0.*Var.*Rho.*Phi;
dispMatrix = 0.*RHO.*PHI;
errorMatrix = 0.*RHO.*PHI;

%-------------------------------------
%load matrices
%-------------------------------------
```

```
fprintf('Loading Matrices...\n')
line = 0;
for r = 1:rhoNum
   for p = 1:phiNum
      line = line + 1;
      for v = 1:numVars
          if (abs(maxVals(v)-minVals(v)) < 1e-10)
              matrixArray(v,r,p) = 0;
          else
              if (resolution > 0)
                  if (data(line,v+2) > (fineTuner*upperBound(v)+(1-
fineTuner)*(averages(v))))
                      data(line,v+2) = fineTuner*upperBound(v)+(1-
fineTuner)*(averages(v));
                  end
                  if (data(line,v+2) < (fineTuner*lowerBound(v)+(1-
fineTuner)*(averages(v))))
                      data(line,v+2) = fineTuner*lowerBound(v)+(1-
fineTuner)*(averages(v));
                  end
                  if(0 == abs((fineTuner*upperBound(v)+(1-
fineTuner)*(averages(v)))-(fineTuner*lowerBound(v)+(1-
fineTuner)*(averages(v)))))
                      matrixArray(v,r,p) = 0;
                  else
                      matrixArray(v,r,p) = 65*(data(line,v+2)-
(fineTuner*lowerBound(v)+(1-
fineTuner)*(averages(v))))/((fineTuner*upperBound(v)+(1-
fineTuner)*(averages(v)))-(fineTuner*lowerBound(v)+(1-
fineTuner)*(averages(v))));
                  end
              else
                  matrixArray(v,r,p) = 65*(data(line,v+2)-
minVals(v))/(maxVals(v)-minVals(v));
              end

          end
          if (data(line,v+2) == -99)%a quick fix
              matrixArray(v,r,p)=0;
          end
      end
      errorMatrix(r,p)= abs((data(line,4)-
initialTotalEnergy)/initialTotalEnergy);
      %errorMatrix(r,p) = min(100,errorMatrix(r,p));
      currentError = double(errorMatrix(r,p));
      currentError = int16(cast(currentError,'int16'));
      if (currentError > 99)
          currentError = 99;
      end
      errorArray(currentError+1) = 1 + errorArray(currentError+1);
   end
end

%-------------------------------------
%display matrices
%-------------------------------------
fprintf('Displaying Matrices...\n')
```

```
for v = numVars:-1:1
    for r = 1:rhoNum
        for p = 1:phiNum
            dispMatrix(r,p) = matrixArray(v,r,phiNum+1-p);
        end
    end

    figure
    image([phiMax phiMin], [rhoMax/a rhoMin/a], dispMatrix);
    set(gca, 'YDir', 'normal');
    ylabel({'Impact Parameter (units of a)',''});
    xlabel({'','Phase Angle (radians)'});
    title({variables{v}, fileName,''});
end
figure
surf(errorMatrix);
title({'Momentum-ness', fileName,''});

figure
errorArray(:) = errorArray(:)/(r*p);
bar(errorArray(1:100));
ylabel({'This percent of simulations...', ''});
xlabel({'...had this percent error.', ''});
title({'Error Spread', fileName, ''});
```

HamiltEqs.map *-generates the Hamiltonian and subsequent equations*

```
M := array(1..3);
M[1] := Ma;
M[2] := Mb;
M[3] := Mc;

P := array(1..3,1..3);
P[1,1] := pax;
P[1,2] := pay;
P[1,3] := paz;
P[2,1] := pbx;
P[2,2] := pby;
P[2,3] := pbz;
P[3,1] := pcx;
P[3,2] := pcy;
P[3,3] := pcz;

Q := array(1..3,1..3);
Q[1,1] := qax;
Q[1,2] := qay;
Q[1,3] := qaz;
Q[2,1] := qbx;
Q[2,2] := qby;
Q[2,3] := qbz;
Q[3,1] := qcx;
Q[3,2] := qcy;
Q[3,3] := qcz;

#-------------------------------------------------------------------------
#     r[i,j] is the distance between mass i and mass j.
#-------------------------------------------------------------------------
r := array(1..3,1..3);
for i from 1 to 3 do
for j from 1 to 3 do
  r[i,j] := sqrt(sum('(Q[i,ss]-Q[j,ss])^2','ss'=1..3));
od;od;

#-------------------------------------------------------------------------
#     n is the normal vector:  n_{ab} = x_a - x_b.
#     first two indicies label masses; third index is the component.
#-------------------------------------------------------------------------
n := array(1..3,1..3,1..3);
for i from 1 to 3 do
for j from 1 to 3 do
for k from 1 to 3 do
  if (j <> i) then
    n[i,j,k] := (Q[i,k] - Q[j,k])/r[i,j];
  else
    n[i,j,k] := 0;
  end if;
od;
od;
od;
```

```
#-------------------------------------------------------------------------
#     Define P^2
#-------------------------------------------------------------------------
P2 := array(1..3);
for i from 1 to 3 do
  P2[i] := sum('P[i,ss]^2','ss'=1..3);
od;

#########################################################################
#
#     Define the terms of the sum
#
#########################################################################
T := array(1..18);

#-------------------------------------------------------------------------
#     Term 1
#-------------------------------------------------------------------------
T[1] := sum(M['ss'],ss='1..3');

#-------------------------------------------------------------------------
#  Term 2
#-------------------------------------------------------------------------
T[2] := 1/2*sum('P2[ss]/M[ss]','ss'=1..3);

#-------------------------------------------------------------------------
#     Term 3
#-------------------------------------------------------------------------
T[3] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
  if (j <> i) then
    T[3] := T[3] + M[i]*M[j]/r[i,j];
  end if;
od;
od;
T[3] := -T[3]/2;

#-------------------------------------------------------------------------
#     Term 4
#-------------------------------------------------------------------------
T[4] := -1/8*sum('M[ss]*(P2[ss]/M[ss]^2)^2','ss'=1..3);

#-------------------------------------------------------------------------
#     Term 5
#-------------------------------------------------------------------------
T[5] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
  if (j <> i) then
    T[5] := T[5] + M[i]*M[j]/r[i,j]*(6*P2[i]/M[i]^2 \
            - 7/(M[i]*M[j])*sum('P[i,ss]*P[j,ss]','ss'=1..3) \
            - sum('n[i,j,aa]*P[i,aa]','aa'=1..3) \
                 *sum('n[i,j,bb]*P[j,bb]','bb'=1..3)/(M[i]*M[j]));
  end if;
od;
od;
```

```
T[5] := -G/4*T[5];

#-------------------------------------------------------------------------
#     Term 6
#-------------------------------------------------------------------------
T[6] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
for k from 1 to 3 do
  if (j <> i) then
    if (k <> i) then
      T[6] := T[6] + M[i]*M[j]*M[k]/(r[i,j]*r[i,k]);
    end if;
  end if;
od;
od;
od;
T[6] := 1/2*G^2*T[6];

#-------------------------------------------------------------------------
#     Term 7
#-------------------------------------------------------------------------
T[7] := 1/16*sum('M[ss]*(P2[ss]/M[ss]^2)^3','ss'=1..3);

#-------------------------------------------------------------------------
#     Term 8
#-------------------------------------------------------------------------
T[8] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
  if (j <> i) then
    T[8] := T[8] + M[i]*M[j]/r[i,j] * (\
              10*(P2[i]/M[i]^2)^2 - 11*P2[i]*P2[j]/(M[i]*M[j])^2 \
           - 2/(M[i]*M[j])^2*(sum('P[i,ss]*P[j,ss]','ss'=1..3))^2 \
           +
10/(M[i]*M[j])^2*P2[i]*(sum('n[i,j,ss]*P[j,ss]','ss'=1..3))^2 \
          -12/(M[i]*M[j])^2*sum('P[i,ss]*P[j,ss]','ss'=1..3)\
               *sum('n[i,j,ss]*P[i,ss]','ss'=1..3)\
               *sum('n[i,j,ss]*P[j,ss]','ss'=1..3)\
          - 3/(M[i]*M[j])^2*(sum('n[i,j,ss]*P[i,ss]','ss'=1..3))^2\
               *(sum('n[i,j,ss]*P[j,ss]','ss'=1..3))^2
         );
  end if;
od;
od;
T[8] := G/16*T[8];

#-------------------------------------------------------------------------
#     Term 9
#-------------------------------------------------------------------------
T[9] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
for k from 1 to 3 do
  if (j <> i) then
    if (k <> i) then
      nij_pi := sum('n[i,j,ss]*P[i,ss]','ss'=1..3);
```

```
      nij_pj := sum('n[i,j,ss]*P[j,ss]','ss'=1..3);
      nij_pk := sum('n[i,j,ss]*P[k,ss]','ss'=1..3);
      nik_pk := sum('n[i,j,ss]*P[k,ss]','ss'=1..3);
      nij_nik := sum('n[i,j,ss]*n[i,k,ss]','ss'=1..3);
      pi_pj  := sum('P[i,ss]*P[j,ss]','ss'=1..3);
      pj_pk  := sum('P[j,ss]*P[k,ss]','ss'=1..3);

      T[9] := T[9] + M[i]*M[j]*M[k]/(r[i,j]*r[i,k])*(\
              18*P2[i]/M[i]^2 + 14*P2[j]/M[j]^2 -2*nij_pj^2/M[j]^2 \
             - 50*pi_pj/(M[i]*M[j]) + 17*pj_pk/(M[j]*M[k]) \
             - 14*nij_pi*nij_pj/(M[i]*M[j]) \
             + 14*nij_pj*nij_pk/(M[j]*M[k]) \
             + nij_nik*nij_pj*nik_pk/(M[j]*M[k])
            );
    end if;
  end if;
od;
od;
od;
T[9] := G^2/8*T[9];

#-------------------------------------------------------------------------
#     Term 10
#-------------------------------------------------------------------------
T[10] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
for k from 1 to 3 do
  if (j <> i) then
    if (k <> i) then
      nij_pi := sum('n[i,j,ss]*P[i,ss]','ss'=1..3);
      nij_pj := sum('n[i,j,ss]*P[j,ss]','ss'=1..3);
      nij_pk := sum('n[i,j,ss]*P[k,ss]','ss'=1..3);
      nik_pk := sum('n[i,j,ss]*P[k,ss]','ss'=1..3);
      nij_nik := sum('n[i,j,ss]*n[i,k,ss]','ss'=1..3);
      T[10] := T[10] + M[i]*M[j]*M[k]/(r[i,j]^2)*(\
                2*nij_pi*nik_pk/(M[i]*M[k]) \
              + 2*nij_pj*nik_pk/(M[j]*M[k]) \
              + 5*nij_nik*P2[k]/M[k]^2 \
              - nij_nik*nik_pk^2/M[k]^2 \
              - 14*nij_pk*nik_pk/M[k]^2
            );
    end if;
  end if;
od;
od;
od;
T[10] := G^2/8*T[10];

#-------------------------------------------------------------------------
#     Term 11
#-------------------------------------------------------------------------
T[11] := 0;
for i from 1 to 3 do
for j from 1 to 3 do
  if (j <> i) then
    T[11] := T[11] + M[i]^2*M[j]/r[i,j]^2*  (\
```

```
            P2[i]/M[i]^2 + P2[j]/M[j]^2 \
          - 2/(M[i]*M[j])*sum('P[i,ss]*P[j,ss]','ss'=1..3)\
                                              );
  end if;
od;
od;
T[11] := G^2/4*T[11];

#-------------------------------------------------------------------------
#     Term 12
#-------------------------------------------------------------------------
T[12] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> a) then
      if (c <> b) then
        for i from 1 to 3 do
        for j from 1 to 3 do

          T[12] := T[12] + M[a]*M[b]*M[c]/((r[a,b] + r[b,c] +
r[c,a])^2) \
                   * (n[a,b,i] + n[a,c,i])*(n[a,b,j] + n[c,b,j]) \
                   * (  8*P[a,i]*P[c,j]/M[a]/M[c] \
                      - 16*P[a,j]*P[c,i]/M[a]/M[c] \
                      + 3*P[a,i]*P[b,j]/M[a]/M[b] \
                      + 4*P[c,i]*P[c,j]/M[c]^2 \
                      + P[a,i]*P[a,j]/M[a]^2 \
                     );
        od;
        od;
      end if;
    end if;
  end if;
od;
od;
od;
T[12] := G^2/2*T[12];

#-------------------------------------------------------------------------
#     Term 13
#-------------------------------------------------------------------------
T[13] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> a) then
      if (c <> b) then
        pa_pb := sum('P[a,ss]*P[b,ss]','ss'=1..3);
        pa_pc := sum('P[a,ss]*P[c,ss]','ss'=1..3);
        nab_pa := sum('n[a,b,ss]*P[a,ss]','ss'=1..3);
        nab_pb := sum('n[a,b,ss]*P[b,ss]','ss'=1..3);
        nab_pc := sum('n[a,b,ss]*P[c,ss]','ss'=1..3);
        T[13] := T[13] + M[a]*M[b]*M[c]/(r[a,b]*(r[a,b] + r[b,c] +
r[c,a]))\
```

```
              * (  8*(pa_pc - nab_pa*nab_pc)/(M[a]*M[b]) \
                 - 3*(pa_pb - nab_pa*nab_pb)/M[a]/M[b] \
                 - 4*(P2[c] - nab_pc^2)/M[c]^2 \
                 - (P2[a] - nab_pa^2)/M[a]^2);

      end if;
    end if;
  end if;
od;
od;
od;
T[13] := 1/2*G^2*T[13];

#-------------------------------------------------------------------------
#     Term 14
#-------------------------------------------------------------------------
T[14] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> b) then
      T[14] := T[14] + M[a]^2*M[b]*M[c]/(r[a,b]^2*r[b,c])
    end if;
  end if;
od;
od;
od;
T[14] := -1/2*G^3*T[14];

#-------------------------------------------------------------------------
#     Term 15
#-------------------------------------------------------------------------
T[15] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> a) then
      T[15] := T[15] + M[a]^2*M[b]*M[c]/(r[a,b]^2*r[a,c])
    end if;
  end if;
od;
od;
od;
T[15] := -3/8*G^3*T[15];

#-------------------------------------------------------------------------
#     Term 16
#-------------------------------------------------------------------------
T[16] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> a) then
      if (c <> b) then
```

```
        T[16] := T[16] + M[a]^2*M[b]*M[c]/(r[a,b]*r[a,c]*r[b,c])
      end if;
    end if;
  end if;
od;
od;
od;
T[16] := -3/8*G^3*T[16];

#-------------------------------------------------------------------------
#     Term 17
#-------------------------------------------------------------------------
T[17] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
for c from 1 to 3 do
  if (b <> a) then
    if (c <> a) then
      if (c <> b) then
        T[17] := T[17] + M[a]^2*M[b]*M[c]/(r[a,b]^3*r[a,c]^3*r[b,c]) \
              *(  18*r[a,b]^2*r[a,c]^2 - 60*r[a,b]^2*r[b,c]^2 \
               - 24*r[a,b]^2*r[a,c]*(r[a,b]+r[b,c]) \
               + 60*r[a,b]*r[a,c]*r[b,c]^2 \
               + 56*r[a,b]^3*r[b,c] \
               - 72*r[a,b]*r[b,c]^3 \
               + 35*r[b,c]^4 \
               + 6*r[a,b]^4 );
      end if;
    end if;
  end if;
od;
od;
od;
T[17] := -1/64*G^3*T[17];

#-------------------------------------------------------------------------
#     Term 18
#-------------------------------------------------------------------------
T[18] := 0;
for a from 1 to 3 do
for b from 1 to 3 do
  if (b <> a) then
    T[18] := T[18] + M[a]^2*M[b]^2/r[a,b]^3
  end if;
od;
od;
T[18] := -G^3/4*T[18];

#########################################################################
#
#     Construct the Hamiltonian
#
#########################################################################
H := sum('T[ss]','ss'=1..18);

#########################################################################
#
```

```
#     Generate Hamilton's Equations
#
#############################################################################
RHS := array(1..18);
RHS[1]  := diff(H,P[1,1]):
RHS[2]  := diff(H,P[1,2]):
RHS[3]  := diff(H,P[1,3]):
RHS[4]  := diff(H,P[2,1]):
RHS[5]  := diff(H,P[2,2]):
RHS[6]  := diff(H,P[2,3]):
RHS[7]  := diff(H,P[3,1]):
RHS[8]  := diff(H,P[3,2]):
RHS[9]  := diff(H,P[3,3]):
RHS[10] := - diff(H,Q[1,1]):
RHS[11] := - diff(H,Q[1,2]):
RHS[12] := - diff(H,Q[1,3]):
RHS[13] := - diff(H,Q[2,1]):
RHS[14] := - diff(H,Q[2,2]):
RHS[15] := - diff(H,Q[2,3]):
RHS[16] := - diff(H,Q[3,1]):
RHS[17] := - diff(H,Q[3,2]):
RHS[18] := - diff(H,Q[3,3]):

#---------------------------------------------------------------------------
#     Write equations to Fortran
#---------------------------------------------------------------------------
with(codegen):
fortran(RHS,optimized,precision=double,filename=`hameqs.h`);
```

nbodyPN.f *-runs (integrates) the three-body interaction until ejection*

```
      program        nbodyPN
      implicit       none

C-------------------------------------------------------------
C     whichescaped: which body left unbound
C     escapeangle: trajectory of escaped body
C     numcloseapp: number of close approaches
C     binaryenergy: change in binary system energy
C     escapetime: when the body escaped
C     escaperadius: distance that a body has escaped
C     t0: approx time when 3 bodies first meet
C-------------------------------------------------------------
      real*8         escapeangle, binaryenergy, escapetime
      real*8         escaperadius, binaryecc, rho, phi, t0
      real*8         escapeke,escapepe
      integer        whichescaped, numcloseapp
      integer        numBoundChanges, bounded
      logical        escaped
      real*8         maxke, maxpdotp, minr, minrtot
      real*8         fourthterm

C-------------------------------------------------------------
C       Original variables
C-------------------------------------------------------------
      real*8         dvvdot

      character*5    cdnm
      parameter    ( cdnm = 'nbodyPN' )

      integer        iargc,         indlnb,         i4arg
      real*8         r8arg

      real*8         r8_never
      parameter    ( r8_never = -1.0d-60 )

c-------------------------------------------------------------
c     tfinal:        Final integration time
c     dtout:         Output interval
c     ntout:         # of output times (computed)
c     trace:         Enables tracing of "conserved quantities"
c-------------------------------------------------------------
      real*8         tfinal,         dtout
      integer        ntout
      logical        trace

c-------------------------------------------------------------
c     Common communication with routine 'fcn' in 'fcn.f' ...
c
c     Includes defn of maximum # of particles, storage
c     for particle masses, Newton's gravitational
c     constant ...
c-------------------------------------------------------------
      include       'fcnPN.inc'
      integer        maxneq,         neq
```

```
      parameter    ( maxneq = 6 * maxnp )

c-------------------------------------------------------------
c     Storage for energy, momentum, center of mass ...
c-------------------------------------------------------------
      real*8         ke(maxnp),     pe(maxnp)
      real*8         etot,          ketot,         petot
      real*8         pmom(d,maxnp), jmom(d,maxnp),
     &               ptot(d),       jtot(d)
      real*8         com(d)
      character*100  fileName
      character*11  defaultFileName
      parameter    ( defaultFileName = 'results.dat')

c-------------------------------------------------------------
c     LSODA Variables.
c-------------------------------------------------------------
      external       fcn0, fcn1, fcn2, jac

      real*8         y(d,2,maxnp)
      real*8         tbgn,       tend
      integer        itol
      real*8         rtol,       atol
      integer        itask,      istate,     iopt
      integer        lrw
      parameter    ( lrw = 22 + 9 * maxneq + maxneq**2 )
      real*8         rwork(lrw)
      integer        liw
      parameter    ( liw = 20 + maxneq )
      integer        iwork(liw)
      integer        jt
      real*8         tol
      real*8         default_tol, default_G
      parameter    ( default_tol = 1.0d-8 )
      parameter    ( default_G   = 1.0d-2 )

c-------------------------------------------------------------
c     Other locals
c-------------------------------------------------------------
      integer        a, i, j, runnumber, whichorder,
     &               itout
      logical        ltrace
      parameter(     ltrace = .true.     )

c-------------------------------------------------------------
c     vars for glbpp
c-------------------------------------------------------------
      real*8         min_radius,    mass_to_radius
      real*8         max_radius

c-------------------------------------------------------------
c     Fix gravitational constant
c-------------------------------------------------------------
      G = 5.0d-2
      tol=default_tol

c-------------------------------------------------------------
```

```
c     Parse command line arguments (initial values) ...
c-----------------------------------------------------------
      if( iargc() .lt. 3 ) then
      write(0,*) 'Error 1'
      go to 900
      end if


      tfinal = r8arg(1,r8_never)
      dtout  = r8arg(2,r8_never)
      G      = r8arg(3,default_G)
      whichorder = r8arg(4,r8_never) + 0.1
      runnumber = r8arg(5, r8_never) + 0.1
      trace  = .false.
c      if( tfinal .eq. r8_never  .or.  dtout .eq. r8_never
c     &    .or. dtout .le. 0.0d0 .or. whichorder .lt. 0
c     &    .or. whichorder .gt. 3 .or. runnumber .lt. 0) then
c      write(0,*) 'Error 2'
c      go to 900
c      end if

c-----------------------------------------------------------
c     Compute number of uniform time steps requested.
c-----------------------------------------------------------
      ntout = tfinal / dtout + 1.5d0

c-----------------------------------------------------------
c     Get particle masses, initial positions and
c     velocities.
c-----------------------------------------------------------
      call getivs('-',m,y,d,maxnp,np)
      neq = 6 * np
      escaperadius=1.01*(y(1,1,3)-0.5*(y(1,1,1)+y(1,1,2)))**2
      rho=y(2,1,3)-0.5*(y(2,1,1)+y(2,1,2))
      phi=atan2(y(2,1,1)-y(2,1,2),y(1,1,1)-y(1,1,2))
      t0=-y(1,1,3)/y(1,2,3)
      numBoundChanges=0
      bounded=4
      maxke = 0
      maxpdotp = 0
      minr = 10000
      minrtot = 100000

c-----------------------------------------------------------
c     Dump # of particles, particle masses, initial
c     time and initial particle positions to standard
c     output.
c-----------------------------------------------------------
      min_radius = 0.2d0
      max_radius = 0.4d0
      mass_to_radius = 0.1d0
      tbgn = 0.0d0

c-----------------------------------------------------------
c     Compute initial energy, center of mass, linear mom
c     and ang mom about center of mass and output if
c     standard error tracing is enabled.
```

```
c-------------------------------------------------------------
         call clce(y,m,ke,pe,ketot,petot,etot,G,d,np)
         call clccom(y,m,com,d,np)
         call clcmom(y,m,pmom,jmom,ptot,jtot,com,d,np)

c-------------------------------------------------------------
c     Set LSODA parameters ...
c-------------------------------------------------------------
      itol   = 1
      rtol   = tol
      atol   = tol
      itask  = 1
      iopt   = 1
C user defined jacobian=1, internally generated=2
      jt     = 2

c=============================================================
c     ***********************************************
c     *************** Begin Integration **************
c     ***********************************************
c=============================================================
      do itout = 2 , ntout
         tend = tbgn + dtout

c-------------------------------------------------------------
c     Check for escaping body
c-------------------------------------------------------------
        call extreema(y,m,maxke,maxpdotp,minr,minrtot,fourthterm)
        call hasescaped(y,escaperadius,escaped,whichescaped,
     &                  pe,ke,m,d,np,t0,tend)

        if (escaped) then
           go to 500
        end if

c-------------------------------------------------------------
c        Call LSODA Integrator
c-------------------------------------------------------------
        istate = 1
c iword(6) = max number of steps per run
        iwork(6) = 100000

        if (whichorder .eq. 0) then
         call lsoda(fcn0,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
        end if

        if (whichorder .eq. 1) then
         call lsoda(fcn1,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
        end if
        if (whichorder .eq. 2) then
         call lsoda(fcn2,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
```

```
        end if

c-------------------------------------------------------
c       Check accuracy error istate = -2, -3; Recall LSODA
c-------------------------------------------------------
234     if ( istate .eq. -2 .or. istate .eq. -3) then
            rtol = rtol * iwork(14)*2
            atol = atol * iwork(14)*2
            if (rtol .ge. 1e10 .or. atol .ge. 1e10 ) then
               istate = 1
               go to 235
            end if
            istate = 3
            iwork(11) = 0
            iwork(6) = 50000
c-----------0 Order-----------------------------------------
            if (whichorder .eq. 0) then
            call lsoda(fcn0,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
            end if
c-----------2 Order-----------------------------------------
            if (whichorder .eq. 2) then
            call lsoda(fcn2,neq,y,tbgn,tend,
     &              itol,rtol,atol,itask,
     &              istate,iopt,rwork,lrw,iwork,liw,jac,jt)
            end if
c-----------Loop accuracy error to top----------------------
            go to 234
        end if

c-------------------------------------------------------------
c        Check for unhandled LSODA error
c-------------------------------------------------------------
235     if( istate .lt. 0 ) then
            go to 950
        end if

c-------------------------------------------------------------
c        Compute energy, COM, linear momentum and angular
c        momentum about the COM and output if tracing
c        is enabled.
c-------------------------------------------------------------
            call clce(y,m,ke,pe,ketot,petot,etot,G,d,np)
            call clccom(y,m,com,d,np)
            call clcmom(y,m,pmom,jmom,ptot,jtot,com,d,np)
            if ( trace ) then
            write(0,5000) tend,
     &                   com(1),  com(2),  com(3),
     &                   etot,  ketot,  petot,
     &                   sqrt(dvvdot(ptot,ptot,d)),
     &                   sqrt(dvvdot(jtot,jtot,d))
5000        format(1p,10E25.16)
         end if
c-------------------------------------------------------------
c    **************  End of integration loop.*************
c-------------------------------------------------------------
```

```
      end do

c----------
c  Exits  -
c  Exits  -
c  Exits  -
c  Exits  -
c----------

c-------------------------------------------------------------
c     EXIT: Successful Integration
c-------------------------------------------------------------
500   continue
      call clce(y,m,ke,pe,ketot,petot,etot,G,d,np)
      call clccom(y,m,com,d,np)
      call clcmom(y,m,pmom,jmom,ptot,jtot,com,d,np)

      call clcexit(tend,whichescaped,y,m,ke,pe,d,np,
     &             numcloseapp,binaryenergy,escapeangle,
     &              escapetime,binaryecc,escapeke,escapepe)

      call fileout(rho,phi,escapeangle,etot,ketot,petot, ptot, jtot,
     &         whichescaped,escapetime,binaryecc,escapeke,escapepe,
     &         whichorder,runnumber,maxke,maxpdotp,minr,minrtot,
     &         fourthterm)
      stop

c-------------------------------------------------------------
c     EXIT: Wrong User Parameters
c-------------------------------------------------------------
900   continue
         write(0,*) 'usage: '//cdnm//
     &              ' <t final> <dt out> [<tol> <trace>]'
         write(0,*) ' '
         write(0,*) '       Masses, initial positions and'
         write(0,*) '       velocities of particles read from'
         write(0,*) '       standard input'
      stop

c-------------------------------------------------------------
c     EXIT: LSODA Error
c-------------------------------------------------------------
950   continue
         binaryecc=-1
         whichescaped=-1
         binaryenergy=-1
         numcloseApp=-1
         write(0,*) 'nbody: Error return from LSODA'
            write(0,*)'  rho = ',rho
            write(0,*)'  phi = ',phi
            write(0,*)'  cdnm = ',cdnm
            write(0,*)'  istate = ',istate
            write(0,*)'  itout = ',itout
            write(0,*)'  ntout = ',ntout
            write(0,*)'  tbgn = ',tbgn
            write(0,*)'  tend = ',tend
         call fileout(rho,phi,escapeangle,etot,ketot,petot, ptot, jtot,
```

```
     &         whichescaped,escapetime,binaryecc,escapeke,escapepe,
     &         whichorder,runnumber,maxke,maxpdotp,minr,minrtot,
     &         fourthterm)
          write(0,*) ' '
          stop
       stop
       end


c-------------------------------------------------------------
c  Subroutines  -----------------------------------------------
c-------------------------------------------------------------

c=============================================================
c     Computes energy quantities: individual kinetic
c     and gravitational potential energies, and total KE
c     and PE and total mechanical energy.
c=============================================================
      subroutine clce(y,m,ke,pe,ketot,petot,etot,G,d,np)
         implicit     none
         real*8       dvsum
         integer      d,          np
         real*8       y(d,2,np),  m(np),    ke(np),   pe(np),
     &                ketot,      petot,    etot,     G
         real*8       vsq,        rsq,      c1
         integer      a,          i,        j

         do i =1 , np
            vsq = 0.0d0
            do a = 1 , d
               vsq = vsq + y(a,2,i)**2
            end do
            ke(i) = 0.5d0 * m(i) * vsq

            pe(i) = 0.0d0
            c1 = -G * m(i)
            do j = 1 , np
               if( j .ne. i ) then
                  rsq = 0.0d0
                  do a = 1 , d
                     rsq = rsq + (y(a,1,j) - y(a,1,i))**2
                  end do
c-------------------------------------------------------------
c                 Associate 1/2 the potential energy of an
c                 interaction with each particle.
c-------------------------------------------------------------
                  pe(i) = pe(i) + 0.5d0 * c1 * m(j) / sqrt(rsq)
               end if
            end do
         end do
         ketot = dvsum(ke,np)
         petot = dvsum(pe,np)
         etot = ketot + petot
         return
      end
```

```
c=============================================================
c     Computes individual linear and angular momentum about
c     specified origin and total linear and angular momentum
c     about said origin.  Currently implemented only for
c     d = 3.
c=============================================================
      subroutine clcmom(y,m,pmom,jmom,ptot,jtot,o,d,np)
         implicit     none
         integer      d,          np
         real*8       y(d,2,np),  pmom(d,np),  jmom(d,np),
     &                m(np),      ptot(d),     jtot(d),
     &                o(d)
         integer      a,          i

         call dvls(ptot,0.0d0,d)
         call dvls(jtot,0.0d0,d)
         if( d .ne. 3 ) then
            write(0,*) 'clcmom: Not implemented for d .ne. 3 '
            write(0,*) 'clcmom: Invoked with d = ', d
            return
         end if

c-------------------------------------------------------------
c        Linear momentum
c-------------------------------------------------------------
         do i = 1 , np
            do a = 1 , d
               pmom(a,i) = m(i) * y(a,2,i)
               ptot(a)   = ptot(a) + pmom(a,i)
            end do
         end do

c-------------------------------------------------------------
c        Angular momentum
c-------------------------------------------------------------
         do i = 1 , np
            jmom(1,i) = (y(2,1,i) - o(2)) * pmom(3,i) -
     &                  (y(3,1,i) - o(3)) * pmom(2,i)
            jmom(2,i) = (y(3,1,i) - o(3)) * pmom(1,i) -
     &                  (y(1,1,i) - o(1)) * pmom(3,i)
            jmom(3,i) = (y(1,1,i) - o(1)) * pmom(2,i) -
     &                  (y(2,1,i) - o(2)) * pmom(1,i)
            do a = 1 , d
               jtot(a) = jtot(a) + jmom(a,i)
            end do
         end do
         return
      end

c=============================================================
c     Computes center-of-mass of distribution of particles.
c=============================================================
      subroutine clccom(y,m,com,d,np)
         implicit     none
         real*8       dvsum
         integer      d,          np
         real*8       y(d,2,np),  m(np),   com(d)
```

```
         real*8       mtotm1
         integer      a,          i

         call dvls(com,0.0d0,d)
         do i = 1 , np
            do a = 1 , d
               com(a) = com(a) + m(i) * y(a,1,i)
            end do
         end do
         call dvsm(com,1.0d0/dvsum(m,np),com,np)

         return
      end

c=============================================================
c     Vector dot product.
c=============================================================
      double precision function dvvdot(v1,v2,n)
         implicit    none
         real*8      v1(*),    v2(*)
         integer     i,        n

         dvvdot = 0.0d0
         do i = 1 , n
            dvvdot = dvvdot + v1(i) * v2(i)
         end do

         return

      end

c=============================================================
c     Vector sum.
c=============================================================
      double precision function dvsum(v,n)
         implicit    none
         real*8      v(*)
         integer     i,         n

         if( n .gt. 0 ) then
            dvsum = v(1)
            do i = 2 , n
               dvsum = dvsum + v(i)
            end do
         end if
         return
      end

c=============================================================
c     Load vector with scalar.
c=============================================================
      subroutine dvls(v1,s1,n)
         implicit    none
         real*8      v1(*)
         real*8      s1
         integer     i, n
```

```
         do i = 1 , n
            v1(i) = s1
         end do
         return
      end

c=============================================================
c     Vector-scalar multiply
c=============================================================
      subroutine dvsm(v1,s1,v2,n)
         implicit    none
         real*8      v1(*),     v2(*)
         real*8      s1
         integer     i,         n

         do i = 1 , n
            v2(i) = s1 * v1(i)
         end do
         return
      end

C=============================================================
C     Has a body escaped and is the simulation over?
C=============================================================
      subroutine hasescaped(y,escaperadius,escaped,whichescaped,
     &                  pe,ke,m,d,np,t0,tend)
          implicit     none
           integer     d, np, whichescaped
           real*8      y(d,2,np), m(np), escaperadius
           real*8      pe(np),ke(np),t0,tend
           logical     escaped
           integer i
           whichescaped=1
           escaped=.false.
           do i=1,3
               if ((y(1,1,i)**2+y(2,1,i)**2+y(3,1,i)**2)
     &               .ge.escaperadius*40.0.and.
     &                (pe(i)+ke(i)).ge.0) then
                   whichescaped=i
                   escaped=.true.
                   return
               end if
           end do

      end

C=============================================================
C     Calculate exit conditions
C=============================================================
      subroutine clcexit(tend,whichescaped,y,m,ke,pe,d,np,
     &             numcloseapp,binaryenergy,escapeangle,
     &              escapetime,binaryecc,escapeke,escapepe)
           implicit     none

           integer numcloseapp
           real*8  escapeke, escapepe
           real*8  binaryenergy, escapeangle, escapetime, binaryecc
```

```
          integer d, np, whichescaped
          real*8  tend, y(d,2,np), m(np), ke(np), pe(np)
          escapeke=ke(whichescaped)
          escapepe=pe(whichescaped)
          escapetime=tend
          numcloseapp=0
          binaryecc=0
          binaryenergy=ke(1)+ke(2)+ke(3)
          escapeangle=atan2(y(2,2,whichescaped),y(1,2,whichescaped))
          if(escapeangle.lt.0) then
                escapeangle=escapeangle+2*3.14159265
          end if
      end

c=====================================================================
c     Calculate max and min values to explain 2nd order divergence
c=====================================================================
        subroutine extreema(y,m,maxke,maxpdotp,minr,minrtot,
     &                      fourthterm)
        implicit none
        real*8       y(3,3,2), m(3), maxke, maxpdotp
        real*8       minr, minrtot
        real*8       tempr, tempke, tempdot, fourthterm
        integer      a, b, d
c-------minimum distance between planets---------------------------
        do a = 1,3
        do b = 1,3
             if (a.ne.b) then
                  tempr = sqrt((y(1, 1, a)-y(1, 1, b))**2+
     &                       (y(2, 1, a)-y(2, 1, b))**2+
     &                       (y(3, 1, a)-y(3, 1, b))**2)
                  if (tempr.lt.minr) then
                       minr=tempr
                  end if
             end if
        end do
        end do
c-------maximum kinetic energy-----------------------------------------
        tempke = 0
        do a = 1,3
             do b = 1,3
                  tempke = tempke + y(b, 2, a)**2/2/m(a)
             end do
        end do
        if (tempke.gt.maxke) then
             maxke = tempke
        end if
c-------minimum total distance of all bodies from eachother----------
        tempr = 0
        do a = 1,3
        do b = 1,3
             if (a.ne.b) then
                  tempr = tempr+m(a)*m(b)/
     &                    sqrt((y(1, 1, a)-y(1, 1, b))**2+
     &                         (y(2, 1, a)-y(2, 1, b))**2+
     &                         (y(3, 1, a)-y(3, 1, b))**2)
             end if
```

```
        end do
        end do
        if (tempr.lt.minrtot) then
             minrtot = tempr
        end if
c-------maximum dot product of 2 body momentums----------------------
        tempdot = 0
        do a = 1,3
        do b = 1,3
             if (a.ne.b) then
                  do d = 1,3
                       tempdot=tempdot+m(a)*m(b)*y(d,2,a)*y(d,2,b)
                  end do
             end if
        end do
        end do
        if (tempdot.gt.maxpdotp) then
             maxpdotp = tempdot
        end if

c-------fourth term-----------------------------------------------------
        tempdot = 0
        do a = 1,3
             tempdot = tempdot + m(a)*(y(1,2,a)**2+y(2,2,a)**2+
     &                 y(3,2,a)**2)**2
        end do
        if (fourthterm.gt.tempdot) then
             fourthterm = tempdot
        end if
        end
```