Riod:

A Many-Body Gravitational Simulation

User Guide and Simulation Description
Updated: May 16, 2023

This document is the Riod simulation system description and user guide. The author uses this document as a living reference which is updated along with any new features, behaviors or methodology. Its primary use is to provide an accurate reference for the author after extended periods of development inactivity. If this document should its way to others and there are questions regarding its content, I can be reached at the email below.

I apologize to any reader in advance in that MS Word keeps messing up this document with false entries in the table of contents and arbitrarily making fonts bold and I have no idea why. Ugh!

By Rod Luhn

rcluhn@mailaps.org

First Issue – September 10, 2007

Table of Contents

1   Introduction

This document describes the usage and operations of the Riod[1] many-body gravitational simulation. The primary purpose for writing this user manual is for the author to track changes to the simulation code and to have a source to refer to after long periods of development inactivity. Of course, if this simulation makes its way to others for use, the manual provides a base for the use and understanding of what is going on inside the code.

Note too that this document is a living one, meaning that it is updated regularly (check out the Change History section as there are more than 24 pages of changes over the decades of development) as the author has time to work on the code and make changes. Unfortunately, because of this, this document is a bit sprawling in its scope and could use better organization, so I will apologize for those inadequacies in advance.

It is highly recommended that any user of this simulation thoroughly read this document to understand the simulation limitations and input/output expectations before actually running it.

1.1   Riod Simulation Features

The simulation has the following features and capabilities:

- Number of objects is "unlimited" in software and is no longer limited to 4000 and is only limited by the system memory. Note that picking the number of objects requires care and sticking to under a few tens of thousands is recommended.
- Options for classical or relativistic calculations see Section 4 for more details.
- Simulation can be restarted from the point it was stopped without losing any calculations.
- Temporal causality provisions (time retarded forces), see Section 4.5
- Many options for setting initial conditions, now specific mass distribution profiles can be created (e.g. exponential, power law, Plummer, NFW, Jaffe profiles are available). See Section 3.2.2 for details.
- Object size and the time delta can be controlled together, see Section 4.3
- Utilizes up to 32 computing cores or threads (although not as efficiently as I would want) using OpenMP extensions
- Collision options include, modified forces to allow inelastic collisions, elastic collisions, and object pass-through with exponential and Plummer force softening and other more exotic modes. See Section 3.2.3.2 for more details.
- Controlled run time of simulation runs, See Section 2
- Self-contained code, no Windows installation needed. Riod will run in directory the simulation files are put and all files needed/created will be relative to that top directory.
- Rich data logging of status and significant simulation events with status and console logs now saved periodically in the data directories as separate files. Control of Logging now included to reduce the amount of disk activity.
- Ability to plot and animate the output files (and to create videos of this output). Rplot.exe has is own documentation and should be referenced for more details on the standalone visualization program.
- An initial conditions log, "Creation.txt", has been created to monitor parts of the code during the creation of the simulation initial condition. The log is mostly useful to debug initial condition scenarios.
- Ability to run different simulations with different controls while retaining particle properties from a previous simulation. This seems to work with single EO scenarios, but not tested with multi-EO scenarios. I am looking into how this might be done but with the method currently used to make multi-EO distributions, there may be no perfect solution for this feature.

1.2   List of Acronyms

The following acronyms are used with the following text. I will try to define these as they are introduced but that may not always happen.

AMD	Advanced Micro devices, a CPU chip manufacturer.

---

[1] In this document, the author will highlight all references to the Riod simulation and its components in bold font. At least that is the intent but there are missed occasions that the author apologizes for in advance.

AU	Astronomical Unit is a distance unit which is defined as the mean distance from the Earth to the Sun.
BMP	Bit mapped file type that is saved from Rplot.exe when in screen capture mode.
COM	Center-of-momentum, the coordinates where the net vector momentum of all the objects is zero.
CPU	Central Processing Unit, the computer's processor
DLL	Dynamic Link Library, software programs used by the operating system and programs.
DOS	Disc Operating System
FSL	Force Softening Length. This is typically twice the particle size within the simulation. The transition distance from force softening to pure Newtonian forces depends on the force softening used.
EC	Elastic Collision; this event is one where the colliding pair elastically scatter.
EO	Extended Object, what I call an initial collection of smaller standard objects SO that is set up by the initial conditions of the simulation run. The simulation can create one or more than one EO.
IC	Inelastic Collision, for the purposes of this simulation, and EC simply combines the colliding objects into a single object while conserving the pair's momentum.
KE	Kinetic Energy
L-J	Lennard-Jones Potential/Force
MKS	Meters-Kilograms-Seconds, common units system used in physics calculations.
OS	Operating System
PC	Pass through Collision / Personal Computer (context left to reader)
PE	Potential Energy
PPV	Particle-Point-of-View, a new feature for the Rplot.exe visualization program.
RC	Replacement Collision, see Section 3.3.6.2 for more details.
RSU	RIOD simulation units, the flexible units used for the calculations in the simulation.
SO	Standard Object; the smallest mass element used in the simulation. An accumulation of SO can be configured to make an EO or many EO
TE	Total Energy
VC	Vanishing Collision, see Section 3.3.6.3 for more details.

1.3   Riod Origins

Work began on this simulation quite innocently in the early 1980's, while the author was a graduate student studying nuclear reaction physics at the University of Notre Dame. It was around 1981 when the Nuclear Structure Laboratory at the university purchased its first mini-computer, a Perkin-Elmer 3230 (I think). Because I was a graduate student, I spent many hours in the lab. During experimental runs, data would come in very slowly and I generally would have between 2- and 3-hour spans where I would have little to do. I decided that I would spend this idle time learning how to use the newly installed mini-computer. Now the experiment that I was running and that would eventually be used in my doctoral thesis was considered a three-body problem, and because I had always been interested in cosmology and space physics, I decided to write some code that would calculate the positions and velocities of gravitating objects. Understand too from my physics training, it was generally inconceivable to discuss systems of objects in precise terms or have exact mathematical solutions

of equations, because there are few if any n-body (n greater than 2) problems that one can solve directly. The difficulty of understanding these many-body scenarios factored into my thinking in starting this simulation.

The beginning code for this exercise was fraught with many missteps, as I really knew nothing about coding this type of simulation. Even the name of the program, which I have kept to this day, is a fat-fingered variant of "Roid" or short for asteroid. I discovered many pitfalls while creating this simulation. One of the most difficult lessons was what to do with objects that come too close to each other. Gravity is an inverse square law force (Newtonian physics is used almost exclusively in Riod) and the forces get very large as objects get closer. Simulations become unstable when numbers start getting large. Another lesson was what to do with units. I discovered that these and many other issues along the way needed to be dealt with and further sections will discuss these and other simulation related issues.

Keep in mind while reading this document that I am not a programmer. I have very limited skills in that regard and the intention of this project is not to learn programming but to try to understand systems of interacting objects.

This simulation is very CPU intensive and if left running, will not stop on its own[2] unless there is an error code that stops it. In addition, large problems might need to run for months[3], depending on the number of objects involved, before interesting results can be seen. It is up to the operator to size the problem properly. The newly added OpenMP extensions to the code speed up operations on multi-processor systems. However, one becomes more tempted to scale problems larger rather than use the increase in computational speed for shorter, smaller studies. That said, the multi-threaded code now allows the operator more options on what can be studied. The user should also note that this program should be Windows and hardware agnostic. However, since this simulation is CPU intensive, running it on a laptop is problematic because not all laptops employ proper cooling for the CPU and subsystems. Thus, be careful running this on a laptop for extended periods. Using the timed run feature described in the input discussion can help to mitigate this problem.

1.4   Riod as a Windows Application

This simulation is a grotesquely simplistic application of Newton's law of gravity (the code does have an option for doing a relativistic calculation but the implementation is flawed, see Section 4.2). This simulation does an Euler approximation solution, which I understand to be inherently unstable computationally. However, with a little work and much testing, this code is fairly stable provided one accounts for things that may cause the instabilities.

Riod is written entirely in FORTRAN is a single self-contained application. It will run in any directory you place it and all files created will be contained within that directory structure.

Riod is now a Microsoft Windows 64-bit program (as of 10/9/2018); I have completely stopped any development of 32-bit variations. Riod runs in Windows as a console application[4] (or command prompt as I will also call it later), which implies that there is no graphical user interface for this code. As far as I can tell, the code that once compiled, it will run in almost any Windows 64-bit operating system. As I am testing new OpenMP extensions with the code, I have tested this on my Windows 7 64 bit Pro, Windows 10 Pro and Home 64 bit. Hopefully I will be able test other Windows variants as I get more of the new code in shape. There are many input variables that are read from the "Riod3.ini" file, which must be placed in the same directory as the "Riod.exe" executable.

This implementation of Riod (as of 2013) is a multi-threaded application, meaning that it can use as many processors that are available on the hardware platform (up to 32 cores/threads). Thus, if you have multiple processors on your system, Riod will use as many processors as you configure it to use, depending on your input parameters. The current desktop and laptop systems appear to be mostly multi-core and multiprocessor architectures. The new multi-processor chips by Intel and AMD can increase the number of objects one can use in a Riod simulation.

Note that Riod when running, can (depending how you configure CPU usage) gobble 100% of the CPU time of one of your computer's CPU cores. For single core processors, if you want to do work on your computer while Riod is running, this is still possible but you will notice your system will be slow or sluggish[5]. One way to speed up your foreground application

---

[2] This is not totally true now as the operator can set up a timed run – see the input file instructions in Section 5.

[3] In fact, I have run some scenarios for more than a year before ending them.

[4] See Section 8.1 for starting and usage of command prompts.

[5] Unless you are running on a multi-core CPU system, then you will probably notice no change in system performance. Multi-core processor systems come highly recommended as workstations by the author (but are not necessary to run Riod) and now that Riod is a multi-threaded, one can use the full computational power of your workstation.

experience is to reduce the priority of the Riod process while it runs. This can be done by opening the Windows task manager, click on the process tab, find the "Riod.exe" process in the "Image Name" column, right click on the process, select the "Process Priority" option and finally set the priority to low. Section 8.2.2 discusses how to set up a batch file to automatically start "Riod.exe" in low priority mode.

Windows has an option that will put all running processes on more equal footing when requesting CPU cycles. Go to the control panel and select the system icon. Select the "Advanced" tab and then the performance button. Under the "Performance Options", select the "Advanced" tab and click the option for "Background Services" under "Processor Scheduling".

1.5   Notes on the OpenMP Implementation and Performance

After much experimentation with and reading about the OpenMP additions to FORTRAN, I have discovered that it is not the panacea I hoped for in terms of performance gains. The workstation that the OpenMP code was created on has 2 physical processors with 4 cores each and thus naively I expected an 8-fold performance gain when running the program. However, I noted immediately that there was a scale problem once I got the code into a state where it would use OpenMP. I tested and found that performance gains happened while running with 2, 3 or 4 processors but any more than that, the execution times would actually increase. I modified the core to "tighten" up my algorithms but that made little difference. I then experimented with execution chunk sizes as per OpenMP directives and there was some improvement but certainly not factors of two going from 4 to 8 processors.

I figured I have better read a bit about this and see if there are some other limitations. Wikipedia's discussion on OpenMP [6] implies there are memory architecture limitations for some platforms where OpenMP code must run. Moreover, I thought about that there are two physical processors in my system and perhaps memory sharing across these two engines might be limiting things. In that vein, I forced the code to use a single physical processor by using the "affinity" mask available in running code in Windows 7. When forced to run in that mode, I can get a factor of 2.3 performance gain, using 4 processors over 1. This is well below my expectations for what should be achievable but for now I will use this and continue testing. See Section 0 for more discussion on additional trials trying to improve the OpenMP efficiency.

Note too that the OpenMP additions only are in affect when a classical, Newtonian simulation is run. The relativistic portion of the code is still single threaded.

2   *Running Riod*

To run Riod, you need to understand what elements make up the process. There are three crucial pieces/files that need to be used. "New.bat" (see Section 8.2.1 for more information on the batch file) is used (but not required) to start new simulation runs, "Riod3.ini" is used to define simulation initial conditions and the "Riod.exe" is the actual computational executable. These pieces are discussed below.

2.1   Starting and Stopping Riod

To start Riod after a problem is initialized, a simple double click of the "Riod.exe" file in the Windows Explorer will start the simulation right from the point where the last simulation status file was saved. Typing "Riod" within a command prompt window, in the directory containing the executable "Riod.exe", can also start it.

To stop execution, Riod now looks for a file in its file system named "stop.riod". If this file exists, Riod continues to run. There is a batch file, "stop.bat" that if run will delete this file and thus cause the simulation to stop. One can also use Windows file manager to simply delete the file too. Thus, starting and stopping a simulation run at any point is very easy and it can be restarted from the point that it was stopped, never losing any computations. Also note that the simulation will save the current conditions periodically. Therefore, if for some reason the computer running it dies (power failure, blues screen of death, or other malady), not many computations will be lost.

2.2   Starting a New Simulation Run

To start a new simulation, the "New.bat" file is most conveniently used since it is designed to clean up all status log files and data files created during any previous simulation run. Depending on the value of the logging option, ICN(13), as many as six log files may be created during a typical simulation run. They are follows:

---

[6] Wikipedia's discussion on OpenMP can be found here: http://en.wikipedia.org/wiki/OpenMP

Log File Name	I/O Unit #	Description
Creation.txt	10	This is a log of information from the creation phase of starting a new simulation. This log is useful while the user is trying to size a simulation. It was created for the developer to test the myriad of initial conditions that are possible to start a new simulation run. See Section 3.3.9.
RiodOut.txt	12	This file echoes the console output while the simulation runs. Note that this file will only contain elements when their pairs of objects that require force modification as these are the only events really worth saving from the console output. As with other logged events, this file is saved to the position folders when the simulation creates a new folder for new position files. See Section 3.3.4 for more details.
Rdata and Sdata I/O	15	These routines read and write the system data files
Check Stop File	22	This is the stop file existence check.
Pdata Writes	24	This is used for writing the position data files.
Riod3.INI	26	Used to read the RIOD3.INI file
Nearmiss.log	30	This log details information from near collisions that may occur. This log is now created for all types of close encounters. See Section 3.3.7 for more details.
Hits.log	32	This log chronicles the details of any collisions when they occur. See Section 3.3.6 for more details.
Status.log	31	This log holds simulation information each time position files are created. This file is saved to the position folders when a new folder is created. See Section 3.3.5 for more details.
Lost.log	33	This log is created to track lost objects from the simulation. When an object obtains escape velocity and travels a significant distance from the rest of the simulation objects, it is removed from the simulation and tracked in this log. See Section 3.3.8 for more details about how this is done.
Riod3con,dat	15	This file is written once when ICN(1)=0. It contains all the initial members of the ICN and CON arrays when the simulation starts.

More information will be given on these logs as noted above.

"Riod3.dat" is a file that holds all the coordinates, velocities, velocity-change[7] array, and current conditions and is written periodically while the simulation is running and when the operator stops the simulation. It is generated internally when a new simulation run starts.

The four "log" files and the "dat" file are all erased when using the "New.bat" file and thus at the start of the new simulation run. "New.bat" also deletes the "Pos" directory that is created to hold all the temporal data files. If needed, a new "Pos" directory is created once a new simulation run begins. If the old "Pos" directory still exists, new temporal files will write over the old ones.

2.3   Modifying the Simulation Initial Conditions

The "fun" in running "Riod" is that there are a myriad of ways to change the initial conditions of each new run. The "Riod3.ini" file is the holder of all the variables that can be changed to control the initial conditions of each simulation.

---

[7] I will refer to the velocity-change arrays several times in this document. The name is somewhat inaccurate, since the values calculated for this array are not technically velocities. They become velocities when scaled with the universal gravitational constant, G. This relationship can be seen in Equation 3 where the values stored in the arrays is must be multiplied by G to become a velocity change in RSU.

Unfortunately, because there are so many variations of parameters, there is no perfect way of coming up with good sets of parameters without some planning/playing with the parameters to see what happens. Note, while trying to find good/interesting initial conditions, it is best to keep the number of objects small so that feedback on what is happening is more immediate.

The "Riod3.ini" file is a flat text file easily edited in a text editor, like the Windows accessory program "Notepad.exe". Important Note: The structure of the "Riod3.ini" file must be maintained exactly. A discussion of each entry in this file will be in a Section 5 below.

To modify this file, open a text editor and then open the "Riod3.ini" file. Modify the parameters of your choosing and save the file. Now you can start a new simulation as described above by running "New.bat". A helpful hint, if you open Riod3.ini in Notepad, set the "Word Wrap" under Format to "unchecked". This will keep any long text lines outside the window area making it easier to see and keep the file structure.

2.4   Command Line Options
The command line options when starting the simulation described below.

2.4.1   *Starting A New Simulation "Riod 1"*
If you examine the "New.bat" file, you can discover one of the options available to the user. When the command line is "Riod 1", this option forces the program to read the "Riod3.ini" and begin a new simulation. The batch file described above uses this option to start the new simulation run and to remove all the log files associated with the last run.

2.4.2   *Restarting Option to Change Simulation Operational Behaviors: "Riod 2"*
The other command line option allows the user to modify some parameters after the simulation parameters are already set up. The following parameters are changeable in this fashion, ICN(2), ICN(3), ICN(12), ICN(13), ICN(18), ICN(21), ICN(22), ICN(33) and CON(58). Note, none these parameters (except ICN(18)) change any of the simulation critical parameters but allow the user to change the operational behaviors.

Changing ICN(13) can allow the user to control the simulation output. This is useful in that one can increase or decrease the level of logging and file creation during the course of the simulation run. However, one should caution when increasing the level of log creation in that there may be unpredictable results. Decreasing, the logging should be fine. See Section 3.3.3 for more details.

Being able to change ICN(33) is useful of these options since this can control the length of time the simulation runs. It can be changed from the current value to another value depending on the desired run time.

ICN(22) and ICN(21) control the OpenMP chunk size and number of processors used for the calculation, respectively. The chunk size is something that can be tuned to the users liking, for better performance but for the current code implementation are not too sensitive a control. Changing the number of processors used during a simulation run will be useful for use on workstations that may only want to partially use the full performance or want to change the performance allocation mid-simulation run.

Changing ICN(18), allows one to make a new simulation from an already running one. For example, if a running simulation has its first close encounter and a new simulation is to be started at that point in time with a different collision control method, this option allows that to be done. To do this, one must copy the relevant time-stamp DAT file from the folder that one wants to start the new on from, then rename it to riod3.dat, since that is the current running status/.data file. Then run RIOD with the 2-command line option and pick 18 as the change modification. One can pick options of -3,0 or 1 for changing ICN(18). Once completed and backed out of the menu, a new entry will be appended to the Creation.txt file with the new RunString.

Changing CON(58) allows a change of the collision check distance multiplier. This was a necessary addition as the intent of making new simulations from old ones has the negative impact of keeping the original CON(58) which may want to be modified in the new variant of that old simulation. Use with caution; test it first! Values are limited between 2 and 20; use the smallest value that works within the bounds of the interaction distances for the simulation.

Follow the instructions on the screen to change what you want and hopefully the instructions are adequate.

2.4.3 Starting New Simulation While Retaining Particle Properties: "Riod -1"

This option will allow the user to start a new simulation using particle properties from a previous simulation. Using the above command, the program will open the Riod.ini file and use that configuration for all the simulation related control but will read in data from a simulation data file for all the velocities, positions, mass and sizes from a previously created simulation. Place the initially saved Riod3_0000.dat file in the new simulation folder, make changes wanted in the Riod.ini file and then run the above command to start it all off.

3   Riod Methodology and Usage

This section discusses the following operational usage topics:

- Simulation Units: will begin with a discussion of units and particularly the RIOD Simulation Unit (RSU).
- Simulation Scenario Class: Setting up initial conditions for different simulation scenario classes. This discussion includes the method for determining initial positions and velocities.
- Simulation Input and Output: RIOD input file and the meanings of the console and output logs

3.1   Units and the Riod Simulation Units

As with any physical calculation, the user knows the importance of choosing and using the proper units. The MKS system is extremely nice for solving problems in the real world but for this simulation, I have chosen to create what I call the Riod Simulation Units or RSU. RSU is simply a recasting of the standard MKS units but into a more convenient form. For example, in an Euler approximation solution of position and velocities are done in an iterative method, where in each computational iteration, each object requires a multiplication of the basic time delta or interval (see discussion in Section 4.1 below). I have chosen this Δt to be always unity, which simplifies the equations and reduces the computations required for simulation. The ramification of this is to immediately recast the universal gravitational constant to a new value depending on what value is chosen for the real Δt. More convenient mass and distance scales are also chosen based on the operator's problem scope.

For example, one might choose to use a distance scale of 1 astronomical unit (AU) as the RSU distance unit and one solar mass to be the RSU mass unit. Thus, if the time interval is picked to be 1 hour, velocities reported by the simulation would be in RSU of AU/hr. Note too that if using the above definitions and having an earth sized object orbiting the sun at 1AU, the orbital period is one year. So, a new choice of units can be convenient when describing simulation results.

3.2   Initial Conditions

Setting the initial conditions for the simulation require some thought and some trial and error to size the problem, settle on a mass distribution, determine the initial positions and velocities of the objects. Once this is all done, the user starts the simulation and can walk away for months (years in some cases) if need be. This section will try to guide the user on how do the above with some insights on what may work and how the Riod does what it does.

3.2.1   Use Cases With Large Central Mass (CON(16)>0)

All the cases below can be used with a large central mass. CON(13) is the value of the central mass in units of the SO mass. For example, if you want the central mass to be 10,000 times larger than the SO mass, CON(13)=10000. When adding a central mass from CON(13), the EO then has a total odd number of SO. For example, if CON(13) is greater than zero and the user sets up a problem with 400 standard objects in a single EO, then the user must enter 401 for the ICN(10). If the value of ICN(10) is not odd for this case, the program will halt.

When a central mass is used, obviously, its mass is manifested for all particles, even if there are Multi-EO in the final distribution. There currently three options for using a large central mass. It can be a point-like particle, CON(16)=0, which uses the SO object density to set its initial size and thus its force softening rules. If CON(16)>0, then the code will use a M(R) force softening profile as ICN(16)= 1 or 20 which are the exponential-cubed softening or Jaffe softening methods commensurate with a scaling from CON(16). These two were chosen for specific scenarios. The Exponential-Cubed (called EX3:1 as discussed below) form is basically a uniform sphere with smoothed cutoff and can be used with large values of CON(16) to simulate a large uniform mass density. The interior mass for this method is given by (where a=CON(16)):

$$M(R) = M_T \left(1 - e^{-\left(\frac{r}{a}\right)^3} \right).$$

The Jaffe profile has the property to provide a linear M(R) with increasing R, which intern can put particles in this potential in orbits where all the interior particles have the same orbital speed, like seen in the speeds of stars orbiting in galaxies. The Jaffe profile has an interior mass defined as (again, a=CON(16)):

$$M(R) = M_T[r/(r + a)]$$

These different profile computations are now done in a separate loop, where the large mass is always assumed to be the last element in all the particle arrays.

3.2.2   Class of Problem for the Simulation Run

The simulation class of problem is what I term the overall mass distribution type to be used to begin the run. Mass distributions are controlled by only a few parameters in the "Riod3.ini" file. The following subsections describe the possible scenarios for mass distributions. For the sake of discussion, I will use the term extended object (EO) to be a collection of smaller standard objects (SO), either bound to a larger object or separated be a relatively large distance from another extended objects.

The classes of initial conditions discussed below are not generally restrictive in what can be done. However, there are some constraints on choices made when considering EO and total numbers of objects. All EO are made up of standard objects and the code forces the number of SO to be an even number and that the number of created EO is also an even number. This is because the code forces the total center-of-mass for all EO to be zero by creating all the standard objects in identical but opposite r and v vector pairs. The same is true for EO, where they are created in pairs with opposite extended or boosted radii and velocities. There are no constraints on the total number of SO but practical limitations usually require the number to be less than a 100,000, as integer I/O as a 5-digit formatting. On my current fastest machines, a galaxy type scenario of 20,000 particles, evolving to 14 billion years, can take about 2 weeks minimum to complete. The largest simulations I have run have been 50,000 particles and a recent one completed 14 billion years in 55 days.

When in Multi-EO scenarios, EO can also be rotated with respect to their original orientations as an added bit of randomness to the final configuration. Input elements ICN(42)-ICN(45) control limits on the random rotations that can be possible. The angles described are theta and phi angular rotations about those spherical coordinates. Making them all zero means no rotation of the EO.

Each of the following three particle distribution methods has their own creation sequence and consequently a separate subroutine creates the final distribution

3.2.2.1   Riod Classic Single Extended Object (ICN(28)=0)

This use case is the classic Riod distribution method. I have retained this method as it allows the creation of disc shapes for the collection. All SO particles will be place in a spherical shape, this will be a uniform-like distribution or a disc shape depending on the value of ICN(27), 1 for more spherical and 0 for discs. Disc shape is controlled by ICN(26), ICN(28)=0 means expanding disc and 1 allows an inverted-like disc. The minimum and maximum radial distances are set by CON(5) and CON(6), respectively.

For this case, determining initial positions and velocities requires some thought as to how they should be set. This will be discussed below.

3.2.2.2   Single Extended Object, Specified SO Distribution Profile, (ICN(28)>0)

This case is similar to the above case, except now the code creates a single distribution of particles using the provisioned profile as given by ICN(28). When ICN(28)>0, the minimum radial distance is set by CON(5) but the scaling length for the particle distribution is on taken as CON(6). Profile types are described in Section 3.2.3.2. Again, for all these distribution types, particles are created in pairs, where the second of the pair is nearly opposite in space and velocity.

3.2.2.3   Multiple Extended Objects, Mass Distributions (ICN(25)>1 and ICN(28)>0)

If ICN(25) is set to something greater than 1, then the simulation creates ICN(25) total EO up to a maximum of half total number of SO. If the user enters a number too large for ICN(25), the program will force ICN(25)=ICN(10)/2. There may or may not be a large central mass as discussed above, depending on the value of CON(13). EO characteristics are controlled by CON(35) through CON(45). See the description for those controls in the RIOD3.INI section.

The creation method for this scenario starts by creating a distribution of the Multi-EO positions and velocities. The position distribution is based on the value of ICN(8) and depending on the value, will create the profile as specified, same as ICN(28). Velocities are determined the same way as for any distribution based on the mass of the particle, which will be the total mass in the EO. These positions and velocities will become the "boost" properties that individual EO will be given once they are created. Individual EO are created by first creating one EO with mass, size, positions and velocities as if it was an isolated system. Then this first EO is duplicated ICN(25) times, rotated by a random set of angles and the arrays are fully populated. Finally, the coordinates are boosted to their final positions and velocities.

If ICN(25)=2, then the EO pair will be centered along the y-axis, one EO on the +y-axis and the other EO on the –y-xais. If ICN(25)>2, then random positions are created for each addition pair of EO.

Note that running a simulation with more than one extended object creates simulation events behaving on two different time scales. An individual extended object has much faster internal time scale, where small objects will be bound to the larger object. The larger extended object interactions will be much slower to evolve. Picking the time scale for the simulation should be based on the internal interactions of each of the extended object. Some trial and error will be required to get the appropriate time scale. Also remember that the time scale also scales object size and therefore collision probabilities.

3.2.3 SO Initial Positions

3.2.3.1 Creating Uniform and Disc-Like Mass Distributions; ICN(28)=0

All coordinates used internally to the Riod Program are in typical x, y, and z coordinates. It is convenient to create initial conditions using spherical coordinates and then convert these to Cartesian values inside the program. Radial distances are calculated using random numbers and then scaling the result between values given in CON(5) and CON(6). Three-dimensional positions are established by using spherical coordinate angles θ and φ[8]. Random values are generated for θ ranges between CON(25) and CON(26). Random vales are generated for φ ranges between CON(23) and CON(24).Two methods are now available to the user to create disk like initial structures for the extended objects. The original method created an expanding wedge (If viewed in the x-z plane) where the z values for individual objects would increase with radial distance. This could be done by simply using values of θ in CON(25) and CON(26) that would mirror values on either side of 90 degrees For example, CON(25)=80 and CON(26)=100 for these values gives a 20 degree expanding wedge. This method was easy to implement in the code but created distributions that were not very physically interesting. This method is still available to the user when ICN(16)=0

A new method is now available to the user which creates a more "realistic" distribution of objects in the extended object. Setting ICN(26)=1 will create like an inverse of the original wedge. When ICN(26)=1, CON(25) and CON(26) take new meaning. They become a percentage of the maximum radius, CON(6) for the maximum z values. So if CON(6) is 100 RSU, and CON(26) is 30, then the maximum value for any objects z is 30. This maximum is scaled towards the smaller radii and will decrease to the CON(25) value at the maximum radii. This creates an inverted wedge when viewed in the x-z plane.

3.2.3.2 Creating Spherical Distributions Using Designated Profiles; ICN(28)>0

This new method allows the user to pick a in initial spherical mass distribution profile. The following table provides a listing of the currently available mass density profiles that can be created given the value of ICN(28) and the distance scaling parameter "a" and power "p" if required; provisioned as CON(6) and CON(3), respectively.

This Table uses the conventions established in my Spherical Mass Distribution Formalism document[9], so I will not reiterate all the details here. However, the density functions are defined as unitless profiles that all the radial distances are scaled by a single constant "a".

$$\rho(r) = \frac{M_T}{4\pi a^3 \beta} f(r/a)$$

The mass ratio g(R/a) is defined as:

$$g(R/a) = \frac{M(R)}{M_T} ; M(R) = \int_0^R \rho(r) \, d^3r$$

And the distribution function normalization, β is defined as:

$$\beta = \int_0^\infty x^2 f(x) dx$$

The above assumes that one would create an infinite number of particles over the entire range of distances. However, the reality of creating a finite distribution requires some limits on the interior and extent of the radial distribution. For a finite extent, one must find the radial minimum (r_0) and maximum (r_m) for the distribution. These extrema are tied to the distribution type, the scaling and the interior packing constraints.

The minimum value, r_0 is really constrained by the particle size and hence its softening length. Since the distribution is built up in radial shells, the initial shell size must be at a minimum constrained to the particle size in some way. As such, CON(5) can be used as the value of r_0 but if CON(5) is less than twice the particle size, it will use the larger of those two values as r_0.

Finding a maximum radial extent requires a bit more effort. I have chosen to use a method where the density profile decreases n orders of magnitude from the minimum radius to the maximum r. This relation is quantified in the equation below:

$\rho(r_0)10^{-n} = \rho(r_m)$.

With these radial limits imposing on the distribution creation, now the interior mass at distance integral becomes:

$$M(R) = \int_{r_0}^R \rho(r) \, d^3r ; R \ge r_0$$

The value of β with limits becomes:

$$\beta_l = \int_{x_0}^{x_m} x^2 f(x) dx ; x_0 = r_0/a \; and \; x_m = r_m/a$$

Note, that the table shows x_m as implemented in the code, with normally n=6 but for Jaffe Ext[10]., n=9. If no value of x_m appears in the table, that particular profile has not been corrected in the code.

I will define the following nomenclature for discussing profile distributions involving exponential functions. Let EXm:n, where m and n are integers, become a shorthand for the exponential profiles below and is defined as:

$EXm:n = e^{-x^{m/n}}$; x=r/a.

Note that EXm:n is also a variant of the Einasto profile. I have just broadened the definition to be compatible with known (and easily implementable) analytic solutions transforming the profile into M(R).

ICN(28)	Description	Profile:f(x); x=r/a	β	Mass Ratio g(z); z=R/a	$x_m = r_m/a$
1	EX3:1	e^{-x^3}	1/3	$1 - e^{-z^3}$	$\left[x_o^3 + ln(10^6)\right]^{1/3}$
2	EX3:2	$e^{-x^{3/2}}$	2/3	$1 - e^{-z^{\frac{3}{2}}}(z^{3/2} + 1)$	$\left[x_o^{3/2} + ln(10^6)\right]^{2/3}$
3	EX1:1	e^{-x}	2	$1 - e^{-z}(\frac{z^2}{2} + z + 1)$	$\left[x_o + ln(10^6)\right]$
4	EX3:4	$e^{-x^{3/4}}$	8	$1 - e^{-z^{\frac{3}{4}}}(z^{\frac{9}{4}}/6 + \frac{z^{\frac{6}{4}}}{2} + z^{3/4} + 1)$	$\left[x_o^{3/4} + ln(10^6)\right]^{4/3}$

[10] I have now implemented the extended Jaffe form where the usual Jaffe profile has p=1.

11	xEX1:1	xe^{-x}	6	$1 - e^{-z}(z^3/6 + \dfrac{z^2}{2} + z + 1)$	$\approx [x_o + ln(10^6) - ln(x_o)]$
12	Uniform	1	1/3	z^3	1
13	Power law function	$1/(1 + x^3)^{1+p}$	$\dfrac{1}{3p}; p > 0$	$1 - 1/(1 + z^3)^p$	
14	Power law function	$x/(1 + x^4)^{1+p}$	$\dfrac{1}{4p}; p > 0$	$1 - 1/(1 + z^4)^p$	
16	Plummer Profile	$1/(1 + x^2)^{2.5}$	1/3	$z^3/(1 + z^2)^{1.5}$	
17	Gaussian, EX2:1	e^{-x^2}	$\sqrt{\pi}/4$	$ERF(z) - \dfrac{2ze^{-z^2}}{\sqrt{\pi}}$	$[x_o^2 + ln(10^6)]^{1/2}$
19	NFW-200	$x^{-1}(1 + x)^{-2}$	Log(201)+ 200/201	$[log(z + 1) - z/(z + 1)]/\beta$	200
20	Jaffe Ext.	$x^{-2}(1 + x)^{-1-p}$	1/p	$1 - [1/(1 + x)^p]$	$\approx [x_o^2 10^9(1 + x_0)^{1+p}]^{1/(3+p)}$

The way these profiles are created is that they are built up in radial shells. I calculated how many objects are to be within each shell and then create random positions within that shell for that number of particles. Now each particle is created with a "mate" that is positioned opposite (I,e, $X_{i+1} = -X_i$) ensuring a more spherically symmetric distribution.

Here are some notes that I have on using these profiles.

- Interesting that for ICN(28)=1,12,13 (p=1),16 all have β=1/3, implying they all share the property that they can be considered like uniform sphere of radius "a".
- Using the NFW and Jaffe profiles can create SO very far from the origin and very concentrated numbers of SO near the origin. One must pick the scaling parameter and CON(33) carefully to ensure that the simulation can create the distribution given the constraints. This is also true to a lesser extent for ICN(28)=4 and 13 depending on the value of "p".
- ICN(28)=1,2,3,4,12,13,16,17 all share the property that for r << a, g(z) ~ z^3; implying a uniform distribution in the distribution interior.

A final note about using these distribution types. As noted above, they are created beginning as radial shells expanding outward. The code will create pairs of particles together in opposite coordinate quadrants. The code also then rotates the newly created particles between the positive z=-axis quadrants so to create more symmetric distribution. With small numbers of particles, a guide to their numbers would be to keep the number divisible by 8 so that all quadrants are have the same number of particles.

3.2.4 SO and Multi-EO Initial Velocities

All SO and Multi-EO velocity magnitudes are determined in basically the same way. First, there is a need to discuss how the velocity vector direction is determined and then what follows will describe how the magnitude of the velocity is determined.

3.2.4.1 Velocity Vectors for SO and EO

The initial magnitude and direction of SO velocities are determined at creation of the EO. The determination of the magnitude of the velocity is described in the following sections. The direction of the velocity is given by a unit vector and at creation time has its components multiplied by the magnitude (whose method is described in following sections).
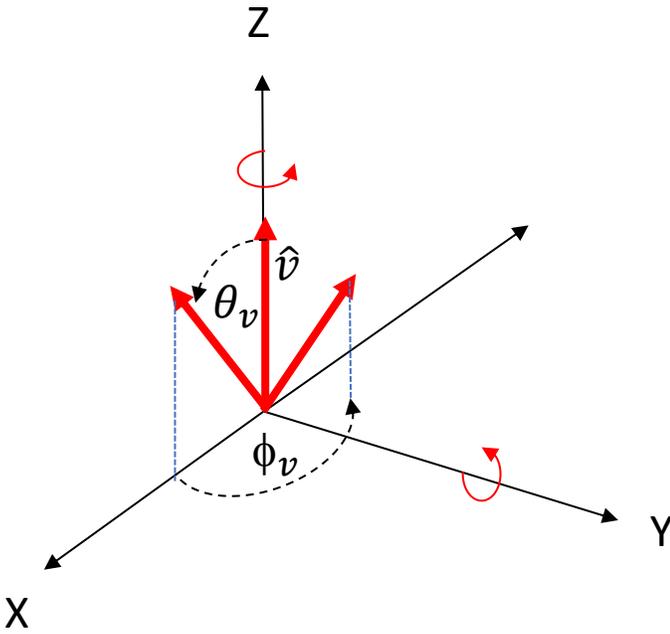
The figure above shows how the velocity vector created. It is best imagined as start as a unit vector, $\hat{v}$ which begins with only a Z-axis component. This unit vector is then rotated about the y-axis polar angle θ inputs. An azimuthal rotation about the z-axis angle ϕ which is a random number such that $CON(23) \leq \varphi \leq CON(24)$. Finally, this unit vector is rotated through the radial vector angles. In practice this done as random number in the range of the input numbers. For SO configurations, CON(20) to CON(25) control these angles. For EO configurations CON(40) and CON(41) control the polar angle theta and the azimuthal angle phi is hard coded to be between 0 and 360 degrees. In this way, the simulation can create, velocity directions ranging in a cone-like way from θ=0, for directions along the radius vector, to θ=90 degrees for directions orthogonal to the radius vector. Using restricted ϕ values, one can control the flow of particles in a plane.

3.2.4.2 Determining SO and Multi-EO Velocity Magnitudes

All SO and EO velocity magnitudes are now determined in basically the same way. Now the code uses the particle position within the mass distribution, either SO or EO to determine the interior total mass. Particles speed are then simply an eccentricity modified circular orbit based on the amount of interior mass. This is represented by the following equation:

$$v_i = \sqrt{GM_i(r)(1-e)/r_i} \ ,$$

Here M_i is the interior mass, G is our gravitational constant and r_i is the radial position. The eccentricity, e is a random number based on inputs. With an eccentricity of 0, this would produce circular orbits for the particles. Note that the eccentricity can be provisioned with the following limits: $-1 \leq e \leq 1$. For e>0, particle orbits begin in their apoapsis and fall inward. However, for e<0, then particles start their orbit at periapsis and move outward. For SO distributions, CON(7) and CON(8) are the eccentricity limits. For multi-EO, CON(38) and CON(39) are the eccentricity limits.

There are some differences in how the code determines M(R) for some cases. For classic SO distributions, one does not always have clean method determining the interior mass. In this case the code will create an M(R) profile by summing up the particles in each particle's interior. A fraction of the total mass is added to the mass of the particle and since, each particle mass is an integer, the fraction part can be extracted later to create the velocity; of course, the integer part of the mass is retained and the fractional part is discarded once the speed is determined.

For all other cases where ICN(28) and ICN(8) are greater than zero, M(R) is determined from the profile type directly. This will work fine as long as number of objects in the SO or EO are greater than a dozen or so. Object numbers less than that will yield some less that ideal initial conditions. One must be careful planning SO and multi-EO distributions.

3.2.4.3 EO Velocity Scaling With CON(20)

A new feature as of 7/24/2019 is the ability to scale the total kinetic (K) energy to force the kinetic energy to total energy ratio (K/E) to be the value of CON(20). Initially, the program calculates positions and velocities and forces the K/E ratio.

However, if CON(20) is >0, then a quantity γ can be defined as a multiplier of each velocity to for the desired ratio. If the desired ratio is $E_R=K/E$ and $K_\gamma=\gamma^2 K$, the it can be shown that:

$$\gamma^2 = \frac{|P|}{K}\left[\frac{E_R}{1+E_R}\right]$$

Where |P| is the magnitude of the total EO potential energy. Thus, once this scaling γ is determined, then all SO in the EO are scaled as $v_i= \gamma\, v_i$.

A different variant now is available to the user. One can set CON(20) to less than zero. In this case, the code will use the absolute value of CON(20) for the energy ratio but the negative value signals the resulting distributions of particle speed to be a Maxwell-Boltzmann distribution. This method, to work correctly assumes that the particle distribution is Gaussian and uses scaled position distances to create the individual SO speeds. Once this is done, the SO speeds are ordered lowest to smallest. I randomize the speeds amongst the SO by randomly swapping indices twice throughout the arrays.

The scaling is done by noting the average speed in a Maxwell-Boltzmann distribution is proportional to the thermal speed (kT). The average radius of all the particles scales all position distances and then they are multiplied by the above γ and finally by a factor $1.17^{0.5}$ (not sure why this works but for now I will accept that it does but maybe there is something in the math that makes this pop out?). I discovered this final factor empirically and it will give the corrected speeds a final energy ratio as provisioned (to within a couple of percent). This has been tested between $0.001 \le |E_R| \le 0.6$.

3.2.4.4 Hubble-Like Expansion with Velocity Modifications (added 9/9/2020).

 A new feature allows for Hubble constant expansion to be added to the SO velocities. The current Hubble constant is hard coded as 70.0 km/s/Mpc.

This feature is activated when the RSU distance parameter, CON(11) is entered in Riod3.ini as a negative number. The code interprets this as the user wanting to add the Hubble constant based velocity corrections to the newly created SO velocities. This is done very simply as the Hubble constant (H_0) is converted to RSU units. We know that H_0 describes the distance rate of change of relative velocities. As such the speed of each SO is modified thusly, $v=r\,H_0$. The direction of the modified Hubble speed is directed outward along the SO radial unit vector. This is done simply by adding the Hubble correction to the current velocity, in the x direction this is:

$$v_x = v_{x0} + xH_0\,,$$

here v_{x0} is the provisioned velocity for the simulation. For simulations that are typically run with this feature, the result of this velocity change is tiny compared to velocities in the final density configuration; about two orders of magnitude smaller. Depending on other initial conditions, adding the Hubble speeds will only delay the overall collapse of gravitating systems slightly.

3.3   The Simulation Process, Input and Output

First I will describe what is done internally when the program runs from the standpoint of the input and output information.

3.3.1   Starting a New Simulation

When starting a new simulation, the program begins by reading in the "Riod3.ini" file and setting up all the initial conditions based on that input. There is generally much to be done to set up the position, velocity and velocity-change arrays. These arrays are at first initialized to zero to start then the populated with the appropriate values based on the input.

Position arrays are calculated and velocity directions are determined. Then velocity magnitudes are calculated. Because the simulation is set to have COM position and velocity of zero and random numbers were used to set positions and velocities, they must be corrected to insure no net movement of the system of objects.

Once all initial conditions are set, the simulation data are all saved into the "Riod3.dat" file, the "Pos" directory is created and the first temporal file is written in the "Pos" directory structure. Now the simulation computations begin. The following information is then written to various locations for monitoring purposes.

Periodically the simulation status is written to the screen based on the value of ICN(12).

Temporal files are written periodically based on the value of ICN(9).

Simulation status saved periodically based on ICN(30).

Important note! If you are running a long term simulation and you are not planning on trying any new parameter changes, rename the New.bat to New.txt. This will prevent an accidental restart and deletion of the running simulation and its data. I learned this the hard way.

3.3.2 *Restarting the Simulation Run*

Restarting the simulation run begins by reading the "Riod3.dat" file. The positions and velocity arrays are reset and the simulation immediately begins normal calculations. It will run until stopped by the user or if there is some sort of system error. Remember too that copies of these files are created at position directory epochs and thus should the current Riod3.dat file become corrupted for some reason, one can go back and recover one of these archived files and restart the simulation from that file.

3.3.3 *Logging and Output Control*

The simulation can produce copious amounts of data files and log files. In previous versions of the code, there was a modest way of controlling the output stream. Now there is a bit more control of the simulation output. ICN(13) now controls the output files. As the value of ICN(13) increases, more output is generated. The table below reflects how the data is controlled and currently, there are 5 levels of logging/data file creation. I have chosen this method because as the value of ICN(13) increases, the "value" of the generated output is less interesting from an analysis perspective. Probably setting ICN(13)=2 is the best option for most common simulation types.

ICN(13) Control	Logs, Files Created and Written
0	When ICN(13)=0, no logging will happen and no data files will be created. This option can be used as a benchmark mode
1	For this option, Creation.txt file will also be created at the start of a new simulation. In addition, all position files will be written as directed by the Riod3.ini file input. Added out includes periodic saving of the Riod3.dat files and running the Stats.exe and directing the output to file.
2	Will add to options above with: Write the Hits.log and Lost.log. Periodically, will write the Nearmiss.log
3	Will add to options above with: Create and write the console.log files periodically.
4	Will add to options above with: Create and write the Status.log files periodically.

3.3.4 *Console output*

Below is a sample output as seen on the console as the simulation runs. I have added a heading for these entries since the program never indicates what these numbers mean. If ICN(18) = -1 meaning inelastic collisions are allowed, then the normal output looks like the following.

0106460616 2.024E+02 159 25 16   10 15367056 3.52E-01 3.52E-01 2 1

0106461605 2.024E+02 159 25 16   10 15368045 3.52E-01 3.50E-01 2 1

0106462594 2.024E+02 159 25 16   10 15369034 3.52E-01 3.50E-01 2 1

0106463583 2.024E+02 159 25 16   10 15370023 3.52E-01 3.50E-01 2 1

106460616	Current iteration number, now a ten-digit number
2.024E+02	Since this is an inelastic collision case, it will just display the time in years.

159	The number of objects remaining
25	The number of collisions
16	The number of lost objects
10	This is the number of iterations the two farthest objects are separated by the speed of light. In this case, light/gravity would take 10 iterations to travel between the two most distant objects. This is useful in setting the value of ICN(15). See Section 4.5 for more information on setting ICN(15).
15367056	Number of iterations since last collision, now a 10-digit number
3.52E-01	Size of the largest object
3.52E-01	Size of next largest object
2	Index of largest object
1	Index of next largest object

If ICN(18) not equal 0 then the output looks like the following:


```
Administrator: CommandPrompt-Development
00002121600 3.049E+09 4000    0    0    59      13105 1.0E-01 1.0E-01    1    2
00002121700 6.148E-01 4000    0    0    59      13109 1.0E-01 1.0E-01    1    2
00002121800 3.050E+09 4000    0    0    59      13116 1.0E-01 1.0E-01    1    2
```

000021216000	Current iteration number, now a ten-digit number
3.049E+09 Or 6.148E-01	If the code determines this is a gas ball simulation, then this is current simulation time which alternates between years and collapse times. If not, it will just display the time in years.
4000	The number of objects remaining
0	The number of collisions
0	The number of lost objects
59	This is the number of iterations the two farthest objects are separated by the speed of light. In this would take 10 iterations to travel between the two most distant objects. This case, light/gravity is useful in setting the value of ICN(15). See Section 4.5 for more information on setting ICN(15).
13105	Number of object pair which are experiencing force softening.
1.0E-01	Size of the largest object
1.0E-01	Size of next largest object
1	Index of largest object
2	Index of next largest object

Another output line is created when two objects are deemed close enough for a closer look. This new output line is output when conditions defined in Section 0. This is similar to the above but with some modifications at the end. Examples of this type of output are shown below.

00002115000	Current iteration number
3.040E+09 Or 6.129E-01	If the code determines this is a gas ball simulation, then this is current simulation time which alternates between years and collapse times. If not, it will just display the time in years.
4000	The number of objects remaining
0	The number of collisions
0	The number of lost objects
2	The number of close objects that potentially can collide
1.622	Distance between objects as a multiple of the sum of their two radii.
1.428	Predicted closest approach distance as a multiple of their two radii.
9.13E+02	Number of iterations before closest approach. Note that if the close approach is in the past, this output shows up as a negative number.
2023	Index of one close object
2534	Index of the other close object
3	Counter of number of close pair that is displayed. More than one pair may be close at any instant in time and this counter will indicate which pair the data presented. Note too that the counter number may change for the same pair from iteration to iteration.

Occasionally, additional lines of output will appear on the console. This output is written when a position file is written to disk. An example of this output is below.

The first line of output has two elements. The first element is the simulation run descriptive string. The second entry is the relative path of the position file what was written.

The second line is the average COM radius of all objects, the elapsed real time between writing the current position file and the last position file and finally the total simulation run time in hours.

Let's examine the descriptive string and its elements as it is a fairly compact way condensing the initial simulation run conditions. Note that I will deconstruct this string but ignore the "Underscore" characters as they delimit the different informational characters in the string.

String	Description
200331	This is the start date of the simulation in the format "YYMMDD"
IBUD[11]	This is a shortened identifier for the system host name of the computer that began the simulation run. It is now limited to 4 characters, the first and last three (non-blank) characters make up this portion of the string.

---

[11] *All my workstations are (in some way) named after my late thesis advisor Sperry "Bud" Darden. Infinite respect Bud!*

D17~104xD02~104	This string designates the overall particle distribution for the simulation. Note this example is for a multi-EO simulation. Note that the 'x' in the middle separates the SO and EO distributions. It begins with 'D17' which indicates that the SO are arranged with ICN(28)=17 or a Gaussian distribution. The next character is a velocity direction indication as described below and the '104 indicates 104 SO make up the EO. Again this is repeated after the 'x' for the EO components and in this case ICN(8) is 2 with 104 EO in the final distribution. For this simulation, there are 104x104=10816 total SO.
	The velocity direction string portion in this field is given by:
	"+": for velocities configured with average angular velocity direction theta of 70 degrees or less. "-": for velocities configured with average angular velocity direction theta of 110 degrees or greater. "~": for cases where the above conditions are not met. These cases are where the average velocity direction is more or less perpendicular to the radial vector.
	The distribution string "D17" can take on other values depending on initial conditions. Below describes the currently available possibilities depending on the values of ICN(28), CON(5) and CON(6):
	"Dxx": If ICN(28) not equal to zero where "xx" is the value of ICN(28) "SSu": If INC(28)=0 and CON(6)=CON(5). This scenario is for all particles arranged on the Surface of a Sphere. "USp": If INC(28)=0 and CON(6)/CON(5)>4. This scenario is for all particles arranged in a Uniform Sphere. "USh80": If INC(28)=0 and other than above this is described as a Uniform Shell Scenario. The numerals are 100*CON(5)/CON(6) to indicate the width of the shell.
PE002	This field tells the user what type of collisions are being used. P is pass through and in E indicates exponential cubed force softening. The three-digit number is an indication of the forces softening length. It is defined as INT(0.5+100*LOG(con(14)*2)).
	The collision string could have multiple other types depending on the value of ICN(18) and other initial input. Below are other current examples of this string:
	- ICN(18) = -3 ; "V" Vanishing Collisions - ICN(18) = -2 ; "R" Replacement Collisions - ICN(18)= -1; "I" Inelastic Collisions - ICN(18)= 0 ; "PE", Pass-through Collisions, here the 'E' part indicates EX3:1 force softening - ICN(18)= 1 ; "E06" , Elastic Collisions, here the 06 is the repulsive power as given in CON(17). - ICN(18) =2 ; "EP", Elastic Collisions, here "P" indicates a piecewise continuous elastic collision is used. - ICN(13)= 3; "PP", Pass-through Collisions and the second "P" indicates that Plummer force softening was used.
Vd-203	The field descriptor Vd is for Virial density and the number that follows 10 times the log of the virial density in g/m^3.
ER001	The field descriptor is ER and stands for Energy ratio of kinetic energy to total energy. The number is =100*LOG(K/|E|). Note that a value of 0 indicates a virial balance between Kinetic and Potential energies. Note that if the energy ratio is zero, the number displayed in the run string is -999.
M+379	The descriptor here is M for RSU mass. In this case, the number in the field is 10 times the log of SO mass in kg.
G-076	This is the value of the universal gravitational constant in RSU. As above the numerical value is 10*log(G) in RSU.

3.3.5 *Status.log Output File*

The status log has entries written to it every time a position file is written. The "status.log" file is not one complete log but is now separated into many logs. When a new data file directory is created, the current "status.log" file is copied to new directory, "pos\StatusLog". Each file has the digits of the directory current structure appended to the fine name when it is moved. For example if the directory "pos\001\35" is created, the "nearmiss.log" file is moved to the following name, "pos\StatusLog\status_00134.log".

Example of the "status.log" file shown below:

```
106680000 2.028E+02  159   25   16  12 15 130 145.88  0.012   15.85    0.68

106690000 2.028E+02  159   25   16  13 16 130 145.91  0.011   15.95    0.69

106700000 2.029E+02  159   25   16  16 20 130 145.94  0.011   15.89    0.70
```

106680000	Current iteration number
2.028E+02	This is the simulation time in years or the ratio of time to the relaxation time. The relaxation time is used only when con(13)=0 or icn(18)≠ -1.
159	The number of objects remaining
25	The number of collisions
16	The number of lost objects
12	The maximum number of close objects being tracked for possible collisions since the last entry into this log. This was changed on 6/2/2012.
15	Number of object pairs having softened forces
130	Potential future number of objects based on potential lost objects (??)
145.88	Average radial distance of all objects
0.012	Average velocity of all object times 1000 (not sure why…)
15.85	Elapsed real time since last Status.log entry
0.68	The total number of real-time hours the simulation has run.

3.3.6 **Hits.log Output File**

The Hits.log file is a record of all the collisions that occurred during the simulation run. There are three different output file structures depending on the value of ICN(18).

3.3.6.1 *Inelastic Collisions when ICN(18)= -1*

An example of the output generated for inelastic collisions is given below:

```
35   432    603    2 1465   108273 1.03E-01   1401   2. 1.835E-02 1.836E-02 0.000E+00

36   155    783    4 1464   112142 1.07E-01   3869   2. 1.835E-02 1.836E-02 0.000E+00

37   162   1225    7 1463   114388 1.09E-01   2246   3. 2.074E-02 2.074E-02 3.408E-03

38   187   1335    6 1462   114808 1.09E-01    420   2. 1.835E-02 1.836E-02 0.000E+00

39  1084   1135    5 1461   115103 1.09E-01    295   2. 1.835E-02 1.836E-02 0.000E+00

40  1116   1194    4 1460   115235 1.10E-01    132   2. 1.835E-02 1.836E-02 0.000E+00
```

35	Collision number
432	Index of first colliding object

603	Index of second colliding object.
2	Number of object pairs close and being tracked at time of collision
1464	Number of objects left in the simulation run
108273	Number of iterations into the simulation.
1.03E-01	Years into the simulation (converted from RSU)
1401	Number of iterations since last collision.
2.	Mass of new object in RSU
1.835E-02	Size of first object before collision in RSU
1.835E-02	Size of second object before collision in RSU
0.000E+00	Absolute value of the size difference of the two objects.
1.34E+00	Ratio of relative velocity to escape velocity (not shown above)

3.3.6.2   *Replacement Collisions when ICN(18)= -2*

An example of the output to the "Hits.log" file generated for the replacement collision option is given below:

104	Collision number
402	Index of first colliding object
879	Index of second colliding object.
1	Number of object pairs close and being tracked at time of collision
1000	Number of objects left in the simulation run
3608657	Number of iterations into the simulation.
205583	Number of iterations since last collision.
-3.688E-07	Total system energy
0.000E+00	The missing energy after the replacement; if this is zero, the total energy for this replacement could not be conserved.
1.338E+02	Virial radius before replacement of objects
2.246E-05	Magnitude of the velocity for the replacement objects. If this is zero, it reflects that the missing energy above it zero. In this case the replacement particles will just fall back to the center of mass.
1.095E+02	Average distance from the origin of the i and j objects at the time of the collision.
2.771E-13	Current system Center-Of-Mass distance to the simulation origin.
6.074E-22	Current system magnitude of the of Center-Of-Momentum velocity
1.34E+00	Ratio of relative velocity to escape velocity (not shown above)

3.3.6.3 Vanishing Collisions, When ICN(18)= -3

An example of the output to the "Hits.log" file generated for the vanishing collision option is given below:

```
Untitled - Notepad                                                    _ □ ×
File  Edit  Format  View  Help
     5    57   664    2   990     3231774    702339  1.761E-01  5.7621E+01 -1.1519E-06  5.9222E-07 -5.5971E-07
     6    58   663    2   988     3231774         0  1.761E-01  5.7504E+01 -1.1457E-06  5.9057E-07 -5.5510E-07
     7   329   367    2   986     5022261   1790487  2.737E-01  9.3719E+01 -1.1274E-06  5.7484E-07 -5.5252E-07
     8   330   368    2   984     5022261         0  2.737E-01  9.3529E+01 -1.1224E-06  5.7243E-07 -5.4995E-07
```

3	Collision number
57	Index of first colliding object
664	Index of second colliding object.
2	Number of object pairs close and being tracked at time of collision
990	Number of objects left in the simulation run
3231774	Number of iterations into the simulation.
702339	Number of iterations since last collision.
1.761E-01	Years into simulation
5.7621E+01	Collision distance from origin
-1.1519E-06	Total Potential Energy
5.9222E-07	Total Kinetic Energy
-5.5971E-07	Total system energy
1.34E+00	Ratio of relative velocity to escape velocity (not shown above)

3.3.7 Nearmiss.log Output File

The "Nearmiss.log" file records data from close encounters of object that don't meet the collision criteria. The decision to write to this file is now determined the following way. I estimate the time and distance of closest approach for the position and velocity of the previous iteration's values. I do it again with the current iteration's data after creating it by updating the position and velocity of the last iteration with new velocity changes calculated for this iteration. I compare the time to collision for those two estimates and when the time goes from positive to negative, indicating that the closest approach is in the past. When that transition happens, I set a flag in ICN(36) to indicate it is time to write to the Nearmiss.log file.

Examples from this file are given below:

```
#Miss     Iter.    Years     I    J     L  NPart   DistOut     TC       RD    CollDist  Rd/SmSiz   RelVel      EP         EK
008318  23500143 2.141E-02  431  5654   5  7502  3.494E+00  5.428E-02 2.723E-01 1.305E-01 3.782E+00 6.4424E-05 -2.0749E-06 2.1270E-06
008319  23502453 2.141E-02 5537  5564   7  7502  1.971E+00  2.296E-01 1.556E-01 1.358E-01 2.161E+00 5.9580E-05 -2.0749E-06 2.1270E-06
008320  23504663 2.141E-02  393  5598   9  7502  3.519E+00  1.399E-01 4.582E-01 1.414E-01 6.364E+00 5.4895E-05 -2.0749E-06 2.1270E-06
```

008318	Number of close encounters
712230390	Number of iterations into the simulation.
2.141E-02	Number of years into simulation time.
431	Index of first near object
5654	Index of second near object.
5	Number of object pairs close and being tracked at time of collision

7502	Current total number of objects
3.494E+00	Distance from origin at time of closest approach
5.428E-02	Time until closest approach in RSU. This number should be less than 1.
2.723E-01	Distance of closest approach in RSU
1.305E-01	Distance that objects must come to collide in RSU
3.782E+00	Ratio of the closest distance to the minimum required collision distance.
6.4424E-05	Relative velocity at time of close encounter in RSU
-2.0749E-06	Total system potential energy after the collision in RSU
2.1270E-06	Total system potential energy after the collision in RSU

The "nearmiss.log" file is not one complete log but is now separated into many logs. When a new data file directory is created, the current "nearmiss.log" file is copied to the directory, "pos\Nearmiss". Each file has the digits of the directory current structure appended to the fine name when it is moved. For example, if the directory "pos\001\35" is created, the "nearmiss.log" file is moved to the following name, "pos\Nearmiss\nearmiss_00134.log".

3.3.8 Lost.log Output File

The Lost.log file records the information of all lost objects to the simulation. An example of this file is given below:

1	Number of lost objects.
618000	Iterations into the simulation
1.191E-02	Number of years into the simulation
299	Number of objects into the simulation.
154	Index of the lost object.
2.0	Mass of the lost object
500.6	Distance of lost object when recorded as lost in RSU
7.987E-04	Current velocity in RSU
84.98	Ratio of Current velocity to escape velocity
203.	Distance to closest object in RSU
-3.39294E-08	Total potential energy calculated after the reduction in object number
6.93822E-05	Total kinetic energy calculated after the reduction in object number

3.3.9 Creation.txt Output File

When the operator starts a new simulation run, the code now logs selected events during the creation of initial conditions. The output of these log events is appended to a file called "Creation.txt". The file is created in the "Riod" folder at execution

time and then later copied into the "pos" folder for archive purposes. This capability was a convenient way to help debug the changes made to the "create" subroutine.

If the operator begins the simulation with the "New.bat" program, any existing "Creation.txt" will be deleted before the simulation starts and new log file will be created at run time.

The output from this file varies and will not mean much to anyone other than the simulation author, so no example output is included in this document.

3.3.10 RiodOut.txt File

A new file is now created for all the iteration output strings as described in Section 3.3.4 for Console Output. The "RiodOut.txt" file is created as an append file status containing the console output. The "RiodOut.log" file now is no longer one complete log but is now separated into many logs. When a new data file directory is created, the current "RiodOut.log" file is copied to new directory, "pos\ConsoleLog". Each file has the digits of the directory current structure appended to the fine name when it is moved. For example, if the directory "pos\001\35" is created, the "RiodOut.log" file is moved to the following name, "pos\ConsoleLog\RiodOut_00134.log".

This file is useful as the operator can go back and find pairs of objects to which are identified as being close. Originally, the operator would have to pick this information out from the console output, which is time consuming and random.

4 Riod Calculation Algorithms

A brief discussion of the Newtonian physics used in the simulation follows[12]. Also presented is a discussion on collisions, lost objects and object sizes.

4.1 Newtonian Gravitational Physics

Newton's second law of motion states that the acceleration of an object is directly proportional to the force exerted on it and inversely proportional to the mass of the object. This is famously written as:

$$\vec{F} = m\vec{a} = m\frac{d\vec{v}}{dt}$$

In this case, the force is gravity. From a Newtonian perspective, gravitational force is viewed as attractive force on object j from the object i as seen in the vector diagram below:
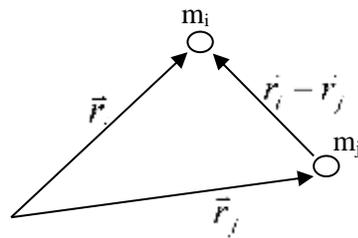
The Newtonian equations describing the magnitude and direction of this force are given by,

$$\vec{F}_j = -G\frac{m_i m_j}{\left|\vec{r}_i - \vec{r}_j\right|^2}\hat{r}_{ij} \quad \textbf{\textit{Equation 1}}$$

---

[12] Note that I have corrected the math and equations in this section several times. I cannot seem to get the indices or object forces straight. I continue to find small errors in this document much to my chagrin. Take this an apology for any continued errors in the document and rest assured that I have it all correct in the code.

Where m_i and m_j are the masses of those particular objects and $\hat{r}_{ij}$ is the unit vector pointing from object j to object i. Note too that G is again the universal gravitational constant. The value of this constant is hard coded into the simulation. I have found a new value for this constant[13] and will be using that going forward.

Equating Newton's second law and the gravitational force for a system of N objects, the force on the object "j" is the vector sum of all the other "i" object forces in the system:

$$\vec{F}_j = m_j \frac{d\vec{v}_j}{dt} = -\sum_{\substack{i=1 \\ j \neq i}}^{N} G \frac{m_i m_j}{\left|\vec{r}_i - \vec{r}_j\right|^3}(\vec{r}_i - \vec{r}_j) \qquad \text{Equation 2}$$

Rearranging, the time rate of change of the velocity of object "j" is:

$$\frac{d\vec{v}_j}{dt} = -G \sum_{\substack{i=1 \\ j \neq i}}^{N} \frac{m_i}{\left|\vec{r}_i - \vec{r}_j\right|^3}(\vec{r}_i - \vec{r}_j)$$

Writing this as the change in velocity:

$$\Delta\vec{v}_j = -\Delta t G \sum_{\substack{i=1 \\ j \neq i}}^{N} \frac{m_i}{\left|\vec{r}_i - \vec{r}_j\right|^3}(\vec{r}_i - \vec{r}_j)$$

This equation separates into each of the three x, y and z coordinates, for example for the x coordinate:

$$\Delta v_{xj} = -\Delta t G \sum_{\substack{i=1 \\ j \neq i}}^{N} \frac{m_i}{\left|\vec{r}_i - \vec{r}_j\right|^3}(x_i - x_j) \quad \textit{Equation 3}$$

The new velocity is the sum of the previous velocity and the change calculated above:

$$v_{xj}(t + \Delta t) = v_{xj}(t) + \Delta v_{xj}$$

The new positions are given by:

$$x_j(t + \Delta t) = x_j(t) + v_{xj}(t + \Delta t)\Delta t$$

Note the symmetry of Equation 1 about the i and j objects.

$$\vec{F}_{ij} = -\vec{F}_{ji}$$

This is Newton's equal but opposite action law in full force (so to speak). This principal is used to reduce the amount of looping the program has to do during each iterative cycle. For a system of N objects, there are N(N-1)/2 combinations of object pairs. The simulation computational time grows with the power of two as the number of objects increase. So, doubling the number of objects will increase the computational time by a factor of four. Keep this mind when setting up your initial conditions.

4.2   General Relativistic Calculations

I am loath to write this section since I no longer have the entire textbook[14] that I based my code calculations on but rather a copied Chapter 7 from that text. I have no formal schooling in general relativity and my special relativity understanding is pedestrian at best. What I am able to glean from Bowler's book, from page 93, equation 7.5.1, he gives the general relativistic

---

[13] As of 1/6/2016, Wikipedia: https://en.wikipedia.org/wiki/Gravitational_constant has a value for $G = 6.67408 \times 10^{-11}$ N m^2 kg^{-2}. Previously, I was using $G = 6.67384 \times 10^{-11}$ N m^2 kg^{-2}. See Section 10.3 for a more detailed history of the value of G used for the simulation.

[14] Bowler, M. G. Gravitation and Relativity. Oxford, England: Pergamon Press, 1976

equations necessary to explain the advance of the perihelion of Mercury. This equation is:

$$\frac{d\bar{v}}{dt} = -(1+\bar{v}^2)\bar{\nabla}\phi - 2\bar{\nabla}\phi^2 + 4\bar{v}(\bar{v}\bullet\bar{\nabla}\phi)$$

Bowler states that this equation is an approximation of the field equations, keeping only terms greater than the velocity squared and the gradient of the potential squared term. According to my notes, the velocity in the above equation is the velocity of the object in the rest frame of the potential creating object. Because I was unsure how to do a proper Lorentz transformation between these moving objects and because usually the velocities are small, I decided to use a linear transformation to determine the relative velocities.

Bowler defines $-\bar{\nabla}\phi$ as simply our old Newtonian gravitational acceleration. This equation has the standard Newton term plus three relativistic correction terms. Note too that the above equation is missing speed of light factors to make it computationally correct.

I will rewrite this equation to show explicitly what is done in the simulation. For object j, the time rate of change of velocity can be written as:

$$\frac{d\bar{v}_j}{dt} = -(1+\bar{v}_{ij}^{\ 2})\bar{\nabla}\phi - 2\bar{\nabla}\phi^2 + 4\bar{v}_{ij}(\bar{v}_{ij}\bullet\bar{\nabla}\phi)$$

And

$$\bar{v}_{ij} = \bar{v}_i - \bar{v}_j$$

is the non-Lorentz transformation as mentioned above.

Rewriting this into 4 terms:

$$\frac{d\bar{v}_j}{dt} = -\bar{\nabla}\phi - \bar{v}_{ij}^{\ 2}\bar{\nabla}\phi - 2\bar{\nabla}\phi^2 + 4\bar{v}_{ij}(\bar{v}_{ij}\bullet\bar{\nabla}\phi)$$

Each term is examined individually below.

The first term: $-\bar{\nabla}\phi = -Gm_i\dfrac{\hat{r}_{ij}}{\left|\bar{r}_{ij}\right|^2} = \dfrac{-Gm_i\bar{r}_{ij}}{\left|\bar{r}_{ij}\right|^3}$

The second term: $-\left|\bar{v}_{ij}\right|^2\bar{\nabla}\phi = \dfrac{-\left|\bar{v}_{ij}\right|^2 Gm_i\bar{r}_{ij}}{\left|\bar{r}_{ij}\right|^3}$

The third term: $-2\bar{\nabla}\phi^2 = -2\bar{\nabla}\left(\dfrac{-Gm_i}{r}\right)^2 = \dfrac{4G^2m_i^{\ 2}\bar{r}_{ij}}{\left|\bar{r}_{ij}\right|^4}$

The fourth term: $4\bar{v}_{ij}(\bar{v}_{ij}\bullet\bar{\nabla}\phi) = \dfrac{4Gm_i\bar{v}_{ij}}{\left|\bar{r}_{ij}\right|^3}(\bar{v}_{ij}\bullet\bar{r}_{ij})$

Summing the four terms and adding factors of c to make the units correct, we get:

$$\frac{d\bar{v}_j}{dt} = -\frac{Gm_i\bar{r}_{ij}}{\left|\bar{r}_{ij}\right|^3} - \frac{Gm_i\left|\bar{v}_{ij}\right|^2\bar{r}_{ij}}{c^2\left|\bar{r}_{ij}\right|^3} + \frac{4G^2m_i^{\ 2}\bar{r}_{ij}}{c^2\left|\bar{r}_{ji}\right|^4} + \frac{4Gm_i\bar{v}_{ij}}{c^2\left|\bar{r}_{ji}\right|^3}(\bar{v}_{ij}\bullet\bar{r}_{ji})\text{ Equation 4}$$

Rearranging the above:

$$\frac{d\bar{v}_j}{dt} = -\frac{Gm_i}{\left|\bar{r}_{ij}\right|^3}\left\{\left(1+\frac{\left|\bar{v}_{ij}\right|^2}{c^2} - \frac{4Gm_i}{c^2\left|\bar{r}_{ij}\right|}\right)\bar{r}_{ij} - \frac{4\bar{v}_{ij}}{c^2}(\bar{v}_{ij}\bullet\bar{r}_{ij})\right\}\textbf{\textit{Equation 5}}$$

Note that all the computational terms (ignoring the mass value) in Equation 6 are anti-symmetric (a sign change occurs) about the exchange of indices i and j. Note that the only quantity in the above equation that depends on a Lorentz boost is the velocity vector. All other quantities are invariant about a Lorentz transformation. Given the assumption above regarding the meaning of the velocity in the above equation, in order to do a proper general relativistic calculation within the simulation, the following process is required:

- Calculating the change of velocity on object j from potential creating object i.
- Find v_j in the rest frame of object j using a Lorentz transformation. This is the velocity that appears in Equation 5 above.
- Use v_j in Equation 5 to calculate the change in velocity, dvj for object j.
- Apply Lorentz transformation on the dv_j velocity change vector to transform the velocity change to the RSU coordinates.
- Sum the velocity change into the proper time retarded arrays.

Invoking the general relativity option for the simulation will not use any OpenMP multi-threading as this portion does not have OpenMP implemented.

4.3 Object Size

This section will discuss how to scale each of the objects in the simulation and what is done when they get too close.

4.3.1 Scaling the Object Size

As mentioned above, there is an inherent difficulty in computing positions and velocities of objects when they get too close to one another. Classically speaking, gravity is an inverse square law, so forces can get quite large as objects approach each other. There are many things that one can do to mitigate this issue but the approach implemented here is one where each object is given a size based on several factors but mostly based on the size of the time increment Δt. Two methods are now available to determine object size and are described below.

4.3.1.1 Specifying Δt in RIOD3.INI

If you specify a Δt in the initialization file, then the simulation uses the following discussion. The following question was asked to help size the objects: What is the orbital period of two of the smallest objects in a circular orbit at exactly their object size? It is assumed that the radial size $S=s_1+s_2$ is equal to the orbital distance. Kepler's third law is perfect for this situation, and the orbital period T is given by[15]:

$$T^2 = 2\pi^2 S^3 / GM$$

where for equal massed objects M and at orbital radius s. Now divide the orbital period into time n slices as in the simulation, needed to determine how many time slices are needed for a stable calculation of this orbit. With now $T = n\Delta t$, the value of n is decided from a computational stability argument and set Δt, the size of the objects is determined by:

$$S = \left[GMn^2 \Delta t^2 / 2\pi^2 \right]^{1/3} \quad \text{Equation 7}$$

Empirically, it is found that a good value for n=10,000. What this physically does is make all the objects very large when compared to real life objects with realistic densities. For example, if the simulation is with 100 earth-sized objects and you pick the time scale to be one hour, the radial size for each object is a huge 3×10^9 meter or about 0.02 AU.

While these unrealistic sizes may be disconcerting, remember that this simulation is designed to look at systems of objects that are supposed to be large distances apart. When masses are at larges distances, it doesn't matter what the object size is because they are considered to be point masses. What the large sizes do is allow a reasonable behavior at close quarters. This combined with inelastic collisions make for better overall behaved systems, computationally speaking.

4.3.1.2 Scaling the object size based on the EO size in RIOD3.INI

Setting CON(10) to zero in RIOD3.INI causes the simulation to calculate a Δt for the user. It will used the size of the SO object entered as ICON(14)=s to directly compute the time delta:

$$\Delta t = \sqrt{\frac{2\pi^2 S^3}{n^2 GM}}$$

4.3.2   When objects get too close

If two objects come too close, the simulation needs to do something to prevent instabilities and "unphysical" consequences. Perhaps the most obvious thing to do would be to allow the two objects to collide. Another possibility is to allow the objects to pass through each other but tempering the force as the objects pass (this is called force softening in the current Physics literature). Currently, there are five options, controlled by the value of ICN(18) implemented in the simulation and are discussed below.

4.3.2.1   Inelastic Collisions: ICN(18) = -1

The simulation provides the option to have collisions. When the collision flag is set, ICN(18)= -1, objects that come within a distance criterion will have an inelastic collision (IC). Currently, two different criteria determine if a IC is to happen. Here are the criteria

$$r \leq (s_i + s_j) * CON(29)$$

:Or

$$r < |s_i - s_j| \text{The}$$

Also, when collisions happen, the total number of objects decreases by one. A benefit of this is that the simulation time between iterations decreases by a factor $(N-1)^2/N^2$. Note also that the newly larger object takes the position of the lower object index in the calculation and the larger index now is swapped with the index of the last object. Note that if CON(29) is less than one, meaning the objects are "inside" each other, the equations below are used to determine the movements. This happens whether or not the collision condition is set.

Object pairs are monitored by the code to see which close pairs may collide. This is done with a simple test, comparing the distance between objects relative to the two objects combined size. Currently, objects I & J are flagged for possible collisions where their separation distance is 2*CON(29)*(SZ$_I$+SZ$_J$), where SZ$_I$ is the radial sized of object I. Thus if CON(29) is set to 0.8, then objects that are 1.6 times the combined sizes are flagged for possible collisions.

Now if ICN(18) is set for no collisions, then the check size is 0.5 times CON(29). This change was necessary because simulations without collisions can have many pairs of close objects and the arrays that track those pairs are limited to 750 elements.

4.3.2.2   Replacement Collisions: ICN(18) = -2

Another method of collisions is now available. Replacement collisions[17] (RC) happen when two objects satisfy the collision criteria, the two objects are removed from the simulation and replaced with new objects farther out from the origin. This new collision scenario is enabled when ICN(18)= -2. The distance out parameter is scaled with the CON(31) times the initial Virial radius CON(51). Currently the replacement objects position is determined with the usual initial object constraints and the orbital eccentricity is set to 0.5. Note that the radial distance is also modified a random dither of ±1/8 of the calculated radius.

The goal during a replacement collision scenario is to maintain energy conservation and zero total momentum. Unfortunately, if the user chooses a large replacement distance as governed by CON(31), energy cannot be conserved. This is because the total potential energy gets less negative and thus the total kinetic energy cannot compensate for the change.

The current method for determining positions and velocities is as follows. I will refer to the colliding particles pair as IJ. At the moment where impact happens, I calculate the center-of-mass for the IJ pair and then scale a new position as a multiplier of those IJ COM vectors. I create these new COM vectors and then convert them back to simulation coordinates.

---

[17] It might seem like an odd option to add to the simulation but there is a method to the madness. I may explain the addition at some point if the planned study goes well. It turned more interesting than initial reports (3/4/2018).

With new IJ positions set, I now need to have consistent velocity vectors. One important property is that the new velocity vectors are perpendicular to the object radius. A method to do this was to find vector cross-product of the radial vectors of the IJ pair. I create a unit vector of this cross-product (which by definition of the cross-product is perpendicular to the radius vector) and then scale it with a calculated speed. Each speed v_i as follows, where G is the usual gravitational constant, M_T is the total mass, m_i is one of the collision participants mass, ε is an eccentricity scaling and r_i is the distance of the larger of the two collision participants:

$$v_i = \sqrt{G(M_T - m_i)(1 - \varepsilon)/r_i}$$

Note that ε in the above is set at 0.4±10%.

In an effort to keep the entire system momentum zero, the IJ pair velocities are equal in magnitude and opposite in direction (at least that is the intent). Finally, to ensure that there is no net system momentum, after the collision positions and velocities are created, I correct the entire system velocity forcing it to zero (or very small computationally speaking.) That overall system speed can be monitored in the "Hits.log".

Note that this option excludes any force softening. Since the objects are replaced with new object positions when they collide (radii overlapping is the collision criterion), there was no reason to have the force modified for this option.

4.3.2.3 *Vanishing Collisions: ICN(18) = -3*

A new feature is the vanishing collision (VC) and is enabled with ICN(18)= -3. As its name suggests, when two objects radii overlap, this collision type of collision will cause both objects to be removed from the system. When this happens the actions by the code are quite simple.

1. The characteristics of the last two objects in the simulation (ICN(10) and ICN(10)-1) replace the two colliding objects. That means mass, size, position, velocity, velocity change arrays are all swapped between these two pair of objects.
2. The total number of objects is reduced by 2.
3. A new determination of the center-of-mass is calculated and corrected to be zero
4. A new determination of the center-of-momentum is calculated and corrected to be zero
5. Potential, Kinetic and Total Energy are calculated for the new system
6. Events are written to the "Hits.Log" file.

4.3.2.4 *Elastic Collisions, Repulsive Core, Modified Lennard-Jones Force: ICN(18) = 1*

It occurred to me that having an option to have elastic collisions (EC) might an interesting variant for the simulation. I thought about what would be needed to do this and how it might be done computationally. Experience with modeling nuclear forces showed me that a strong repulsive force at short distances would create elastic collisions. In the realm of molecular physics, a Lennard-Jones[18] (L-J) potential/force is used to model collisions. I examined several variations on the L-J force and settled on the following because it has all the required properties. Including a bit more rigor into this discussion as I don't always remember what I have done and why. The Lennard-Jones potential is discussed as a related energy potential, so if I am to use this concept to construct a repulsive force, I will first start with the gravitational potential energy is usually written as:

$$V(r) = m\phi(r) = -\frac{GmM}{r} = -\frac{k}{r}\mathrm{I}$$
$$V(r) = -\frac{k}{r} + \frac{\alpha}{(n+1)r^{n+1}}$$

Where α is a constant that must have the appropriate units to make this addition to the potential energy. Suppose, where S (here I use capitol S for the combined size of the two objects, s_1+s_2) is again the combined size of the two interacting objects:

$$\alpha = kS^n$$

Then we have:

$$V(r) = -\frac{k}{r} + \frac{kS^n}{(n+1)r^{n+1}} = -\frac{k}{r}\left[1 - \frac{S^n}{(n+1)r^n}\right]$$

---

[18] I leave to the reader to Google Lennard-Jones potentials (My Solid State physics class wasn't totally worthless after all!).

Since the contribution to the potential energy in brackets is unitless, this is candidate to use to create a repulsive force. We know that a force can be constructed from the potential in the following way:

$$\vec{F}(r) = -\vec{\nabla}V(r) = -\frac{\partial}{\partial r}V(r)\,\hat{r}$$

Sticking with the magnitude of the force and leaving the derivative exercise to the reader, the magnitude of the force is: $F(r) = -\frac{k}{r^2}\left[1 - \left(\frac{S^n}{r^n}\right)\right]$

$$F(r) = -\frac{k}{r^2}\left[1 - \left(\frac{S^n}{r^n}\right)\right]$$

This force has the correct $1/r^2$ behavior at large r and is strongly repulsive at short distances[19]. See Figure 1 for a closer look at the behavior of this function. Going forward, I will use the shorthand LJn for the above force.

This option is now available in the simulation. Setting ICN(18)=1 will cause the code to use the above formalism for inter-object distances less than RCN(6). See Section 5.3 for how this distance is determined.

I have tested[20] this formalism using CON(17)=3,6,9,10 and allowed two objects to fall into each other, a zero relative angular momentum case. In each case the tests were run for more than 200 million iterations with 25-35 collisions. The collision period in each test case remained constant to within 4-5 parts per million, with little net drifting of that period in any direction. I am confident that using this option will produce correct results.

4.3.2.5 Elastic Collisions Using Piecewise Continuous Repulsive Force: ICN(18)=2

I began investigating a piecewise continuous force hoping that if successful, the repulsive force would be more localized to the colliding objects than the L-J force used above. The method for creating this force is nearly identical to what is presented in Section 10.1.4 with the addition of added constants to the Potential Energy terms. I use four constants for the potential energy term as it is still a polynomial of rank3. Again, the potential energy, force and force's derivatives must be equal when r=S. That gives three equations and 4 unknowns but I will constrain the problem to have the PE at r=0 to be a multiple of the Newtonian PE at r=S. The interior potential can be written:

$$\phi_<(r) = b_3 r^3 + b_2 r^2 + b_1 r^1 + b_0$$

The constraint at r=0 for the potential, means that $b_0 = \gamma k/S$, where that value of γ will be set using value based on the maximum kinetic energy of the simulation. One can now solve this set of equations with the following results:

$$b_3 = -\frac{(3+\gamma)k}{S^4}\;;\; b_2 = \frac{(8+3\gamma)k}{S^3}\;;\; b_1 = -\frac{(3+2\gamma)k}{S^2}\;;\; b_0 = \frac{\gamma k}{S}$$

The determination of γ is all that is required to implement this new interior force. As stated above, I will use a multiple of the value of the PE at r=S as the benchmark and the maximum kinetic energy (KE_{MX}) possible in the created system simulation. For example, an estimate of the KE_{MX} would be if two SO fell from infinity and collided at the distance of the initial maximum radius. The individual speeds of the two colliding SO would be equal to twice the escape velocity from that distance.

$$KE_{MX} = 2\left(\frac{1}{2}mv_{es}^2\right) = mv_{es}^2 = m\left\{\sqrt{\frac{2GM_T}{R_M}}\right\}^2 = \frac{2GmM_T}{R_M} = \frac{2N_p Gmm}{R_M}$$

The final equivalent on the above inserts the number of particles, N_p times the SO mass as the total system mass. The maximum radius, R_M is take as the provisioned value of CON(6).

---

[19] Like the author of this document!

[20] My initial testing of this option, it appear there was some disconcerting behavior where the total energy of the system is not conserved as expected. I modified the force factor to S^3/r^3 from a 6th power law to reduce to a smaller power law at short distances as a way to mitigate this observation. Another way to mitigate this may be to arbitrarily make the object size larger, again reducing the effect of a rapidly changing slope of the force. One way to do this is to make ICN(35) larger. A new method for making the object size larger is now provided by the value of CON(15) is made more than 1.0, this value will multiply the calculated object size. New testing has shown with CON(15)=2 or 4, no energy conservation issues are seen. I will test again with CON(15)=1at some point to see if there are energy issues at the original condition.

$$V(r = S) = -Gmm/S$$

I will use the positive value at r=S and that the ratio of $KE_{MX}/|V(S)|$ will define the value of γ.

$$\gamma = \frac{KE_{MX}}{|V(S)|} = \frac{\dfrac{2N_p Gmm}{R_M}}{\dfrac{Gmm}{S}} = 2N_p S/R_M$$

For example, a typical simulation scenario, plug the following in the above equation N_p=1000, S=0.2 and R_M=100: then γ=4.

This is implemented in the code when ICN(18)=2, by determining γ and storing its value in CON(32). Then the other coefficients are calculated and populating RCN(7)=b_3; RCN(8)=b_2; RCN(9)=b_1.

Testing of this additional option for a repulsive force have not produced as consistent behaviors as the L-J option. I will leave this option in place for now but may remove it in the future after additional testing.

See Section 4.3.2.8 for graphs of this repulsive force and comparisons to the other forces used in this simulation.

4.3.2.6   Object Pass Through: EX3:1 or Smooth Gauss Method: ICN(18) = 0

An alternative to the above methods is to create a force that has similar behavior to the Gauss method equations (see Section 10.1.1 for details of the Gauss method) but has force between the objects have more piecewise and derivative continuous behavior. This simulation option I will call the EX3:1 method (or historically I have called this the smoothed Gauss Method) pass through collision (PC). Consider the following force function:

$$f(r) = -\frac{k}{r^2}\left[1 - e^{-\left(\frac{r}{S}\right)^3}\right]$$

In this case the force now is zero at r=0 as in the Gauss case and for large distances, this force defaults to the standard Gravitational force. This particular form is interesting for several reasons. First, from a computation perspective, the value of r^3 is already used in the code, so reusing it is convenient. Secondly, from a physical perspective, as two objects overlap as they get closer to a complete overlap, one understands that the net force on them should go to zero. See Figure 1 for a closer look at the behavior of this function.

This option is now available in the simulation. Setting ICN(18)=0 or -1 will cause the code to use the above corrected force at distance where r < 2.0*S.

4.3.2.7   Object Pass Through: Plummer Force Softening: ICN(18) = 3

The Plummer method of force softening has been added. When ICN(18)=3, the code will used the Plummer model for close interactions. I have added this option since with is what is used extensively in the literature for softening forces between pairs of objects. The interesting thing about using the Plummer potential is that it has an exact solution for the Poisson equation. However, this method of force softening has a much larger extended range than the method I have been using (discussed in the previous section). For example, the Smooth Gauss method I have been using defaults to 99.9% normal gravity at a distance of r/S=1.9 but Plummer softening doesn't reach achieve 99.9% until a distance of r/S $\approx$ 39. This for the same scaling distance S, force softening will have to be done 20 times further out in distance using the Plummer method. The magnitude of the Plummer force is given by (where x=r/S)

$$f(r) = -\frac{k}{S^2 x^2}\left[\frac{x^3}{(1 + x^2)^{3/2}}\right]$$

Note that the softening parameter discussed in the literature for the Plummer softening method is the value of S above (they often us the symbol "ε"). However, in my simulation, the time increment can be set by a SO size. In that case that particle size is half the value of S in the above equation; recall that "S" is the sum of the two particle sizes. For example, if the literature uses a force softening distance of 90 parsecs, the simulation should use a size of 45 parsecs to match the seen behaviors.

4.3.2.8   Force Models: Additional Discussion

Figure 1 shows how the force on an object depends on the separation distance for the models used in this simulation. Note that all 4 near-distance models have the correct large r behavior by 3 times the object size, the force felt has become our

standard $1/r^2$ Newtonian force (F_N). The modified L-J force shows that the force goes to zero at r=S (S=1 in this case) and then gets strongly repulsive very quickly. This modified force will give an elastic collision for virtually any collision kinetic energy. Because all these modified forces are still only dependent on the separation distance and that the direction of the force is always along the lines of the vector between them, these forces are conservative. Conservative forces are ones which energy is conserved for all interactions. In addition to energy, total angular momentum is also conserved. Thus, for these forces, we can expect that the total energy and angular momentum will be unchanged for the duration of the simulation run. Of course, note that for inelastic collisions, the total energy of the simulation is not conserved as is the case for these types of collisions.
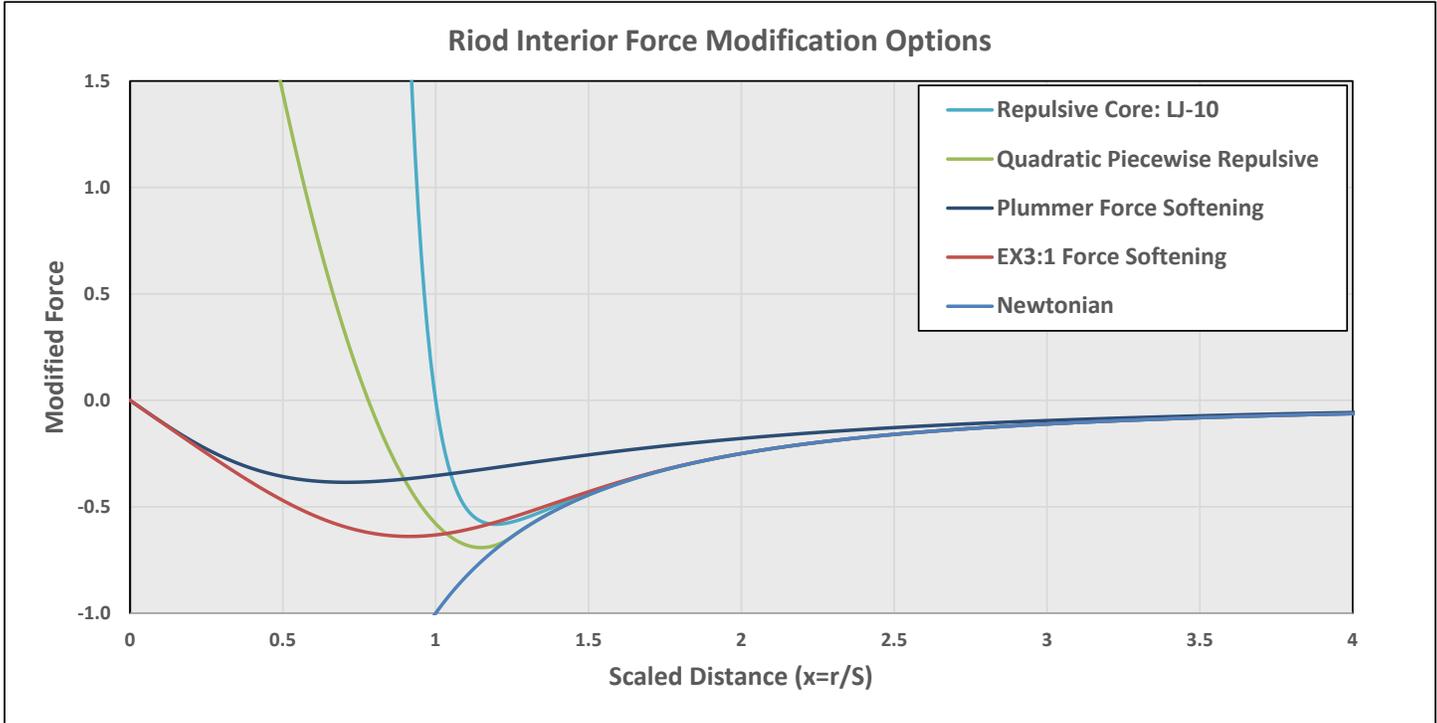
Figure 1: Current RIOD Force Modification Options. Shown is the LJ-10 option (CON(17)= -10) which has the property that the force is essentially Newtonian when S=2 just as the EX3:1 force modification.

The piecewise continuous repulsive force was created to make more localized repulsive force and that intent was successful as noted in the above comparison with the L-J repulsive force. However, the code itself behaves more consistently with the L-J option for some reason.

See Section 10.1 for additional discussion on other force models that were considered for use in this simulation.

4.4 Lost Objects

As mentioned above, if objects acquire enough kinetic energy, they can achieve the system escape velocity. These fast-moving objects can take a large amount momentum a large distance from the remaining objects, which have recoiled because of momentum conservation. It behooves the simulation to drop these fast-moving masses from the calculation because of this momentum transfer and because they no longer play a gravitational role in the evolution of the system of objects that remain. These fast movers are dropped when they reach large distances (controlled by several parameters) and then each remaining object has its velocity corrected so that the net COM velocity is now zero again. This also ensures that the remaining system doesn't drift too far from the original COM.

 Objects are determined to be lost if:The object's velocity is greater than the escape velocity

and the object's distance from system center-of-momentum is greater than CON(6)*CON(19)*CON(37)

and no other object is within 2*CON(6) (this is twice the original size scale of the system)

4.5 Causality and Gravitational Propagation Speed

Gravitational waves are presumed to travel at the speed of light. This finite propagation speed has the net effect to retard in time the force on one object by another by the speed of light propagation time. In the simulation, there is an attempt to

compensate for these time-retarded forces by determining the number of iterations, at the speed of light, the objects are from each other and then populating the velocity-change arrays for that pair of objects, to be used that number of iterations later. In practice, this array is currently set for 1000 temporal elements for each object. For really large scaled simulations, this array may need hundreds of thousands of elements. Should the simulation need more time iterations than allowed by the current array size, then the time, iterations is set to the maximum allowed by the array size.

The number of temporal array elements used in the velocity-change arrays is controlled by ICN(15). Setting this to the maximum will create a rather large "Riod3.dat" file, so size your problem accordingly. Unless the simulation uses galactic dimensions, only a dozen elements may be required.

Note that now the program will scale itself for this parameter if user wants to let it. Setting ICN(15) = 0 or less, will cause the program to use the largest scale distance for that particular simulation type. Two scenarios are covered for this internal calculation. One extended object: the maximum index for the array is calculated by the product of CON(6) and CON(19). More than one extended object: the maximum index for the array is calculated by the product of CON(6), CON(19) and CON(37).

Note that if the calculated value is greater than 1000, then ICN(15) is set to 1000. If ICN(15) is set to anything greater than zero, the program will use that value up to 1000.

5   "Riod3.ini" Input Parameters

This section will discuss in more detail the parameters required to run the Riod simulation. There are two arrays of number read in from the "Riod3.ini" file. The ICN array is now 50 members deep and is used to hold all the integer parameters. The CON array is also 60 members deep and holds all double precision floating point numbers. Note that the restricting of these array elements, Riod3.dat only requires 30 elements to be read into the ICN array and 45 elements to be read into the CON array.

I am trying to also arrange these input control into groupings that make better flow and thus easier to change between simulation types. I have also color coded the input categories to help with that flow. Here is a color guide for these categories:

Simulation Control
Input/Output Control
Simulation Scale
Extended Object Configuration
Collision Control
Standard Object Size
Multi-Extended Object Control
Non-Input Internal Constants

5.1   The ICN Array

Below each of the ICN array elements is discussed in the form of a numbered list. Each number corresponds to the array element number. There are now 80 elements to this array below will only refer to those elements in use.

Array Element	Config. Cat.	Description

ICN (1)	Sim. Control	This is the iteration index used internally. Set to 0 to start. It is also used in the creation of initial conditions as a temporary flag to tell the code that the creation phase has failed in some way.
		The total number of iterations is now=ICN(19)*MaxIt+ICN(1). MaxIt is set to 10^9 internally. This change allows the total iteration number to increase beyond the INTEGER*4 limit of 2^{31}-1. I will probably need to carry the MaxIt number as a member of this array at some point so that other programs can use it more easily without having to hard code in.
ICN(2)	I/O Control	This is used to now control when the program runs stats.exe. The lowest directory number is modulated by ICN(2). For example, if ICN(2)=5, then every 5th directory created will also have stats.exe run with its output text file put into the "pos\StatFiles" directory. ICN(2) can be 0 to 100.
ICN(3) 30	I/O Control	This is the elapsed time in minutes between writes of the simulation status is saved to Riod3.data. This ensures that status gets saved periodically to minimize the impact of OS crashes or power outages while the simulation runs. Allowed values are between 1 and 1440 minutes. If specified outside this range in the Riod3.ini, it will be set to the default of 60 minutes.
ICN(4)	I/O Control	I created a new option to include velocities in the position file output. If the input value is something other than below, the default is ICN(4)=0. I used ICN(4) to control this with the following:
		- ICN(4)=0, This option includes the mass and position values for each particle and has an extension of "D00" - ICN(4)=1, include the magnitude of the velocity with the position and mass data. Position files now have an extension to the filename of ".D01". - ICN(4)=2, include the XYZ velocity data and now have an extension to the filename of ".D02".
		Some auxiliary programs that use to POS data must be changed to account for this new behavior. The list includes Rplot.exe, Hist.exe, Extract.exe.
ICN(5)	Sim. Scale	Used to control how many times the simulation will try to get a new position before giving up and terminating the creation given the other input. I have used values of 100 and 400 as internally. This work but one and up the trials now using ICN(5).
ICN(6)	Sim. Scale	The number of tracked pairs of close particles for Nearmiss.log or collisions. This number is used to allocate array sizes and thus is part of memory usage. Keep this under 1000. Note, if ICN(13) is less than 2, this number will be ignored and set internally to 1.
ICN(7)		Used to count near miss events for the Nearmiss.log file.
ICN(8)		Used for distribution type for Multi-EO configurations. Same inputs as used for ICN(28).
ICN(9)	I/O Control	This is the number of iterations before writing the temporal position files. This is generally picked carefully since one doesn't want to write files more often than absolutely needed. Depending on the value of the time between iterations and number objects one should pick this to be 1000 to 10,000 or even as large as 100,000.
ICN(10)	Sim. Scale	This is total the number of objects when beginning the simulation. This number may change during the run because of collisions or lost objects. This is one of the more important choices to be made when sizing the simulation run. There is no maximum for this input but use care in the choice.
		Note that now a negative input will now add 8 large masses, external to the SO distribution. ICN(64) holds the current number of SO.

ICN(11)	I/O Control	This is the nu mber of temporal position files written in the "Pos" directory. Note that there are two nested sub directories created under "Pos" with the potential of 100 directories in each of these nested directories. This is done to keep the number of files in each of these nested directories manageable. Over the course of a long simulation run, up to a 10,000 directories can be created with ICN(11) files per directory. Keep ICN(11) to 100-500 and your OS and plotting program will thank you.
ICN(12)	I/O Control	This is now just the number of iterations between writes to the screen. The value will not change with changing numbers of objects.
ICN(13)	I/O Control	This is now used to control all the simulation output. Values of ICN(13) may be between 0 and 4. When ICN(13)=0, then no output is generated. For larger values of ICN(13), more output is generated. See Section 3.3.3 for more details.
ICN(14)	Sim. Control	This is the index for the velocity-change arrays. This is now (2/20/2017) set to 0 internally and was surprised that it wasn't initialized properly in the case where it is read from the INI file.
ICN(15)	Sim. Scale	This is the maximum number of iterations used in the velocity-change arrays to account for the speed of light propagation of the speed of light. This can bet set to the maximum of 1000. For huge problems, the speed of light can limit how many iterations forces can be delayed. See sections above regarding causality. Now this parameter can be determined internally by setting this flag to –1. Then the program uses the maximum size scale of the simulation CON(6)*CON(19)*CON(37) divided by the speed of light to set this value. If the value that is calculated is greater than the 1000 maximum, then it is set to 1000. See Section 4.5 for more details.
ICN(16)	EO Config.	This is the mass distribution type to be used for a big mass scenario. Allowed types are 0, for a point mass, 1 for exponential cubed distribution and 20 for a Jaffe distribution. If not these types, it will be set to 0 for "point mass". Set CON(16) if needed.
ICN(17)	Sim. Control	This is set to 0 for classical Newtonian calculation. If set to 1, a general relativistic calculation is done between objects. See Section 4.2 for discussion of limitations of using this option.
ICN(18)	Collision Control	This flag determines what to do when objects get too close. There are now many modes available to use: • ICN(18)= -3, this option creates VC events as discussed in Section 4.3.2.3. • ICN(18)= -2, this option creates RC events as described in Section 4.3.2.2. • ICN(18)= -1, IC events for this option per Section 4.3.2.1. • ICN(18)= 0, then only PC events in Section 4.3.2.6. • ICN(18)= 1, then objects will have an EC as per Section 4.3.2.4. • ICN(18)= 2, Piecewise continuous EC; See Section 4.3.2.5 • ICN(18)= 3, This PC option uses Plummer force softening. See Section 4.3.2.7 for details.
ICN(19)	Collision Control	Set this flag as 1 to use the cross-section in CON(30) to set the collision distance.
ICN(20)	Sim. Control	This is an internal control. Now used to hold the number of times the iteration has passed the maximum allowed. Now the total number of iterations is calculated using ICN(19) and ICN(1). The total number of iterations is now=ICN(20)*MaxIt+ICN(1)

ICN(21)	Sim. Control	Number of processors to use, if set to zero, the simulation will use the maximum number available up to 32. This parameter is also changeable at the command line. See Section 2.4 for more details.
ICN(22)	Sim. Control	Is used for the chunk size in the OpenMP directives. This value must be in the range of 1-100. This parameter is also changeable at the command line. See Section 2.4 for more details. Some experimentation with this number may improve the simulation's performance.
ICN(23)	Sim. Control	Used internally to determine if the Nearmiss.log entry about to be written is the first. The control is set to zero to start and once the first element is written, it is set to 1. This is reset once a new log file is needed.
ICN(24)	EO Config	This is the mass multiplication factor. All small objects will be scaled by this multiplicative factor which will be randomly picked from the smallest mass to this factor times that small mass. For example, if ICN(24) is 10. Then all masses will be random between 1 and 10 times the scaled mass units defined in CON(12).
ICN(25)	Multi EO Config	The value of ICN(25) will be the number of EO. Make sure that ICN(25) and ICN(10) are even numbers! If they are not even numbers, the code will force it to be even.
ICN(26)	EO Config.	This is a flag that now determines the object distribution. When equal 0, the standard distributions are created using spherical coordinates for the initial positions. If set to 1, then CON(25) and CON(26) take on values of percentage of the maximum radius CON(6) for the minimum and maximum z values. See Section 3.2.3 for more details.
ICN(27)	EO Config.	Use ICN(27)=1 for more spherically symmetric distribution of objects in the extended object. Use 0 for more disk shapes which ICN(26) can help control.
ICN(28)	EO Config.	Used to set EO distribution type. If ICN(28)=0, the code will use the legacy options for creating the EO distribution; see Section 3.2.4.2. If ICN(28)>0, then the code will setup the EO as one of many different mass density profiles (E.g. exponential variants, uniform, NFW, Jaffe and other forms; see Section 3.2.3.2 more details.
ICN(29)33	Sim. Control	Number of minutes to run the simulation before a timed stop. Zero means run until operator stops it. Note this option can be used to set up a scheduled task in Windows and run the simulation during nighttime hours.
ICN(30)	SO Size	This is the value of n in the equation 4. This is empirically found to be 10,000 and is set there. If once wants to experiment with code stability, this can be lowered or raised but lowered is a more compelling direction because this reduces the object size for the same time delta.
ICN(31) (1)		Used as large central mass flag, equal 1 when true.
ICN(32)		Used internally to tell CreateProfile that we are making distribution of SO (=0) or EO (=1)
ICN(33)		Used to hold icn(10) minus the number of big masses. Big mass number is restricted to 1.
ICN(34)		Temp used as write flag.
ICN(35)	EO Config.	Used temporarily to hold the total number of failed SO position failures during creation of the mass distribution profile when ICN(28)>1. Reset to zero after EO setup.
ICN(36)		Used as a flag for the writing an event to the Nearmiss.log
ICN(37)		Set as 1 if this is a potential Gas Ball type simulation.
ICN(38)		This is the number of collisions, determined internally, determined internally. Set to 0 to start.

ICN(39)		This is the number of iterations since the last collision, determined internally. Set to 0 to start.
ICN(40)		This is the stop index, determined internally. Set to 0 to start.
ICN(41)		This is the index of the largest object. Set to 1 to start.
ICN(42)		This is the index of the next largest object. Set to 2 to start.
ICN(43)		This is the number of objects escaped from the system, determined internally. Initialized to 0 along with ICN array in Create subroutine.
ICN(44)		Set internally as the random number seed.
ICN(45)		This is the current directory sublevel that positions are written. Set internally to –1 at the start of the simulation.
ICN(46)		This is the current directory sub-sublevel that positions are written. Set internally to –1 at the start of the simulation.
ICN(47)		Used to track first time entering Classical or Relativistic calculations, set internally
ICN(48)		This is now used to count the number of object pairs that are now experiencing force softening. This means currently the object pair needs to be within RCN(6) times the sum of the objects radii. It is also used temporarily in the "Create" subroutine as a flag to tell "Newpos" that it is to calculate positions differently for an EO.
ICN(49)		This is the maximum iterations in time that two objects are separated at speed of light. Set internally.
ICN(50)		Used as print control for the simulation creation phase in order to print to the Creation.txt file.
ICN(61)	EO Config	Holds the flag to tell initial conditions to add Hubble expansion to SO velocities.
ICN(62)		Counter to track the number of times the offset positions are calculated. Note for a newly initialized simulation, this counter will increment every time a new directory for position files is created.
ICN(63)		Holds the initial number of large masses configured. There currently are only two options: 1 and 8.
ICN(64)		Holds the number standard objects. ICN(10) now is the total number of objects, ICN(63)+ICN(64)

The CON Array

Below each of the ICN array elements is discussed in the form of a numbered list. Each number corresponds to the array element number. There are now 80 elements to this array below will only refer to those elements in use.

Array Element	Config. Cat.	Description
CON(1)	Physics Constant	Is the Universal Gravitational constant in MKS units, will be converted to RSU. If set to zero, 6.67408d-11 is hard coded as an option and will be used.
CON(2)	Physics Constant	Is the speed of light in MKS units, will be converted to RSU. If set to zero, 2.9979d+08 is hard coded as an option and will be used.
CON(3)	EO Config	For ICN(28) >0, this is the power of the mass distribution if needed. Right now this is for the cases of ICN(28)=13,14
CON(4)	EO Config	Is now used to hold the initial collapse time of the system.

CON(5)	EO Config	This is the minimum radial distance that an object can have from the COM position of either the extended object or the only system of objects. This is again in RSU based on CON(10). Note if ICN(28)>0, this will be the minimum radial distance that SO will be created at.
CON(6)	EO Config	This is the maximum radial distance that an object can have from the COM position of either the extended object or the only system of objects.
CON(7)	EO Config	This is the minimum orbital eccentricity for orbiting objects. A value of zero will give circular orbits. Acceptable values are –1 to 1. Note for values less than one, this means that the orbit starts at the minimum orbital distance and values, i.e. orbital perigee. CON(7) must always be less than or equal to CON(8). This provisioning is now different if ICN(28)>0. See Section 3.2.3.2 for guidance on how to set this when ICN(28)>0.
CON(8)	EO Config	This is the maximum orbital eccentricity for orbiting objects. A value of zero will give circular orbits. Acceptable values are –1 to 1. Note for values less than one, this means that the orbit starts at the minimum orbital distance and values, i.e. orbital perigee. CON(7) must always be less than or equal to CON(8). This provisioning is now different if ICN(28)>0. See Section 3.2.3.2**Error! Reference source not found.** for guidance on how to set this when ICN(28)>0.
CON(9)	EO Config	This is the dither amount to be used for Classic SO distributions. It is now based on the size of the smallest particle and the input amount multiplies the size. So, CON(9)=5, will dither the SO coordinates a random amount between ±5 times the smallest particle diameter.
CON(10)	Sim. Scale	This is the time scale conversion factor to convert seconds to the RSU time interval. Note that this is not final time interval value. The final Δt value is the product of CON(10) and CON(14). Examples are:1 hr=3600 sec and 1yr=3.15576d+07 sec.
CON(11)	Sim. Scale	This is the distance conversion factor to convert MKS units to RSU. This number is entered in meters. Examples are: 1parsec=3.086d+16 m, 1AU=1.4957e+11 m and 1 lt-yr=9.4605d+15 m
CON(12)	Sim. Scale	This is the MKS mass number for all the RIOD mass units. Note that internally, this is the building block mass and thus unless scaled using another method, elemental masses numerically would be 1. Examples masses are: Earth Mass=5.979d+24 kg; Solar mass=1.991d+30 kg
CON(13)	EO Config	This is the mass of the central object in the extended object case. In RSU mass units as defined by CON(12).
CON(14)	SO Size	Now use this to directly set the SO size in RSU for the smallest unit when ICON(10)=0
CON(15)	SO Size	This constant will multiply the calculated object size if set to a number greater than one. Otherwise set to zero if intending to use the internally calculated object size.
CON(16)	SO Size	Now used to size scale the central mass, CON(13) and will be used in that way as an M(R) calculation using the formula $M(R)=CON(13)*(1-\exp(-r^3/con(16)^3)$. There are some issue with this current implementation that need addressing; use with caution.
CON(17)	SO Size	Used to hold the repulsive core power used for the elastic collision option; ICN(18)=1. This number should be set as negative, between -2 and -20. The default if CON(17) is outside that range is -3.
CON(18)	EO Config.	Multiplier of the velocity magnitude. Set to 1.0 unless doing expanding systems which require higher velocities than the eccentricity method allows.
CON(19)	Sim Scale	This is the number of times the initial size of the simulation an object must travel to escape the system. This is best set to 25 to 100. It is multiplied by CON(6) and CON(37) if there is an extended object.
CON(20)	EO Config.	This number now represents the Kinetic to total Energy ratio desired for the final EO configurations. If CON(20)> 0, then this feature is activated. If CON(20)<0 the simulation will set the velocity constant for all particles no matter their distance. It will continue to use the prescribed velocity direction as per the input below. The magnitude of the velocity is as per

		the entered ration for CON(20) and the total potential energy derived from the particle positions. See section 6.1.4 for details.
CON(21)	EO Config.	This is the minimum spherical coordinate theta for the initial velocity vector. It is restricted between 0 and 180 degrees and must be ≤ CON(22)
CON(22)	EO Config.	This is the maximum spherical coordinate theta for the initial velocity vector. It is restricted between 0 and 180 degrees and must be ≥ CON(21)
CON(23)	EO Config.	This is the minimum spherical coordinate phi for the initial velocity vector. It is restricted between 0 and 360 degrees and must be ≤ CON(24)
CON(24)	EO Config.	This is the maximum spherical coordinate phi for the initial velocity vector. It is restricted between 0 and 180 degrees and must be ≥ CON(23)
CON(25)	EO Config.	If ICN(26)=0, this is the minimum spherical coordinate angle θ, which can be between 0 and 180 degrees. CON(25) must always be less than CON(26). If ICN(26)=1, this becomes the minimum z coordinate value in percentage of the maximum radius CON(6). See Section 3.2.3 for more details.
CON(26)	EO Config.	If ICN(26)=0, this is the maximum spherical coordinate angle θ, which can be between 0 and 180 degrees. CON(25) must always be less than CON(26). If ICN(26)=1, this becomes the maximum z coordinate value in percentage of the maximum radius CON(6). See Section 3.2.3 for more details.
CON(27)	EO Config.	This is the minimum spherical coordinate angle φ, which can be between 0 and 360 degrees. CON(23) must always be less than CON(24).
CON(28)	EO Config.	This is the maximum spherical coordinate angle φ, which can be between 0 and 360 degrees. CON(24) must always be greater than CON(23).
CON(29)	Collision Control	CON(29) controls the collision check distance, CON(58). Depending on ICN(18), it is used as follows: • ICN(18) = 2: This is the case for elastic collisions using the piecewise continuous repulsive force; CON(58)=1.5*CON(29) and CON(29) is set to 1. • ICN(18) = 1: This is the elastic collision (EC) mode. Check distance CON(58)= 2*CON(29). • ICN(18) = 0: This is pass-through collision (PC) mode with exponential cubed force softening. Check distance CON(19)=1.0 and then CON(58)=2.*CON(29), since mass fraction is 0.999 for r=1.9 for exponential cubed force softening. • ICN(18) = -1: This is the inelastic collision (IC) mode. This is the fraction of the objects combined size objects must come within in order for an inelastic collision to occur. This is best set to 1.0d or less. I usually set it for between 1.0 and 0.5. Check distance CON(58)= 2*CON(29) • ICN(18) ≤ -2: Input from RIOD3.INI is ignored and CON(29) is set to 1.0d0. CON(58)= 1.5*CON(29). If ICN(19)=1, then CON(58)=2.5*CON(29)
CON(30)	Collision Control	Set this as the cross-section for interaction for colliding SO in MKS units; it will be converted to RSU internally to the code. The only known value for this is 3×10^{-32} m^3 sec^{-1}.
CON(31)	Collision Control	Now multiplies the virial radial distance for replacement objects when the replacement collisions option is used; ICN(18)= -2.
CON(32)	Collision Control	Calculated internally: Holds the multiplicative constant for the piecewise continuous interior force used for elastic collision type when ICN(18)=2. See Section 4.3.2.5.
CON(33)	EO Config	Now sets the minimum distance between objects at the time the initial positions are set. This number times the size of the two objects is the minimum distance allowed.
CON(34)	Multi EO Config	This number now represents the Kinetic to total Energy ratio desired for the final EO distribution. If CON(34)> 0, then this feature is activated.

CON(35)	Multi EO Config	Minimum EO allowed separation as multiple of max EO radius CON(6)
CON(36)	Multi EO Config	This is the minimum EO boost radius and a multiple of the EO size radius. For example, if the minimum EO size parameter, CON(5) is 10, then the minimum boost radius the be CON(36)*CON(5)
CON(37)	Multi EO Config	This is the maximum EO boost radius and a multiple of the EO size radius. For example, if the maximum EO size parameter, CON(6) is 10, then the minimum boost radius the be CON(37)*CON(6)
CON(38)	Multi EO Config	Minimum EO boost eccentricity. Not a true orbital eccentricity but provides a mechanism to vary the EO relative speeds. This can be a value between –1 and 1. For simulations using several EO, one may have to be willing to experiment with this parameter to obtain bound systems. Right now a value of 0.5 seems to work with EO numbers of 4 to 6.
CON(39)	Multi EO Config	Maximum EO boost eccentricity. Not a true orbital eccentricity but provides a mechanism to vary the EO relative speeds. This can be a value between –1 and 1. For simulations using several EO, one may have to be willing to experiment with this parameter to obtain bound systems. Right now a value of 0.5 seems to work with EO numbers of 4 to 6.
CON(40)	Multi EO Config	Minimum EO boost velocity direction angle, between 0-360 degrees but must be less than CON(41)
CON(41)	Multi EO Config	Maximum EO boost velocity direction angle, between 0-360 degrees but must be greater than CON(40)
CON(42)	Multi EO Config	Minimum EO orientation rotation Phi, between 0 and 180 degrees but must be less than CON(43)
CON(43)	Multi EO Config	Maximum EO orientation rotation Phi, between 0 and 180 degrees but must be greater than CON(42)
CON(44)	Multi EO Config	Minimum EO orientation rotation Theta, between 0 and 360 degrees but must be less than CON(45)
CON(45)	Multi EO Config	Maximum EO orientation rotation Theta, between 0 and 360 degrees but must be greater than CON(44)
CON(46)	EO Config	Used temporarily when ICN(28) >0 while creating the initial EO mass distribution. It holds the EO total mass excluding any large central mass. Reset to zero after EO setup.
CON(47)		Holds the Limited Beta correction factor generated by the BetaCorr subroutine
CON(48)		Holds the Limited Beta generated by the BetaCorr subroutine
CON(49)		Initial crossing time in RSU. If CON(13) is zero and ICN(18) not = -1.
CON(50)		Initial Relaxation time in RSU, calculated from CON(51), If CON(13) is zero and ICN(18) not = -1.
CON(51)		Current Virial Radius calculated with the Total Energy after any energy changing collisions. Note this implies that the newly calculated virial radius is the expected new value after the event. Note that this value will be used for RC to scale the replacement particles positions. If CON(13) is zero and ICN(18) not = -1. Also this is now updated when a new position folder is created.
CON(52)		Current Total System Potential Energy, updated after any energy changing collisions and now updated when a new position folder is created.
CON(53)		Current Total System Kinetic Energy, updated after any energy changing collisions and now updated when a new position folder is created.
CON(54)		This is the mass density of all standard objects (SO). Calculated internally

CON(55)		This is the total system mass, determined internally.
CON(56)		This is the multiplicative constant of the object mass to get size. Calculated internally.
CON(57)		This is the collision check distance set, determined internally
CON(58)	EO Config	This is now used as an internal place holder for the collision check distance and it is set based on the values of CON(29) and ICN(18). See CON(29) input for details.
CON(59)		This is used to hold the total number of real-time seconds that the simulation has been running.
CON(60)		This holds the initial EO virial radius for Multi-EO scenarios.
CON(61)	EO Config	Holds the Hubble constant in RSU. H_0 is hard coded as 70 km/s/Mpc then converted to RSU.
CON(62)	Sim. Data	Holds the calculated X offset position. This is updated each time a new position file is created. The other two position offsets are calculated the same way. The way this is done is that the current virial radius is calculated first. Then the last know offset vector is difference to position vectors and if the difference is less that Rv/2, then it is used in a center-of-mass calculation. Once cycled through all the particles, then a new center-of-mass is found an becomes the core offset vector.
CON(63)	Sim. Data	Holds the calculated **Y** offset position. This is updated each time a new position file is created.
CON(64)	Sim. Data	Holds the calculated **Z** offset position. This is updated each time a new position file is created.
CON(6**5**)	Sim. Data	Holds current virial radius. This is updated each time a new position file is created.

5.2   Picking Values for Interesting Simulation Scenarios

The definitions of all the input parameters in the previous two sections can appear to be somewhat daunting if trying to get a set of input for starting a new simulation. It was always my intent provide some guidance for picking values for the members of the ICN and CON arrays. Normally, it is advised for the user to begin with some basic ideas on scaling of the problem for the simulation to use and then modify those parameters to suite the overall needs for a particular simulation type. Much trial and error will be needed to settle on some interesting input parameters and this can be a bit time consuming and frustrating. In this section, I will try to lend some "wisdom" from collected testing of various simulation scenarios over the years.

5.2.1   Picking the "Big Three"

The three most important parameters in the simulation input are the Mass, Distance and Time scaling, CON(12), CON(11) and CON(10), respectively. Picking values for these three numbers dictate the RSU value of the Universal Gravitational Constant used for the duration of the simulation.

As a general rule these three parameters should be picked to complement each other in the sense that they need to be commensurate with the scale of envisioned simulation scale. For example, if one is planning a simulation scaled to the size of our solar system, one might use the moon's mass as the smallest elemental mass for RSU-M, the astronomical unit as the distance RSU-D scale and time scale of 60 seconds as the RSU-T.

Be aware that setting these parameters also sets the object size therefore its density. As per the discussion in Section 4.3, the object size is generally calculated internally and it will almost always ends generating object sizes and thus densities which are unphysical for standard objects. When picking these three parameters, again one must balance them so that one gets close to intended scales for the simulation of interest. In that spirit, I have created spreadsheet calculation tool to aid in picking numbers to start the simulation.

5.2.2   RiodPad Spreadsheet Simulation Scaling Tool

To avoid some of the trial and error in picking starting parameters for the simulation, I have created a spreadsheet tool that one can use to do some "what-if" testing before entering numbers into the "RIOD3.INI" file. Below is screen shot from that tool from the above example.

As noted above, the object size computed for this moon massed item is quite large, 45,000 km as opposed to the Moon's known radius of 1700 km. Using the spreadsheet, you can modify the "big three" to balance the size of the objects and other potential constraints.

Another observation I have noted over the years is that simulations that have their big three parameters set to produce interesting simulation results usually have values of the gravitational constants in RSU near the same order of magnitude as the MKS value. In the example below, the RSU value of G is several orders of magnitude smaller than the MKS G. This is an indication that perhaps this choice of "big three" parameters is perhaps not the best. Note in the screen shot below, the numbers in column C to the right of the RSU values show what numbers could be entered into the number adjacent in column B to make the RSU value of G numerically the same as the MKS value.

The spreadsheet example below is for a single simulation. I have modified the spreadsheet to now have two duplicate sheet calculations so that one can compare different scenarios in the same sheet. The example below shows only one of the two possibilities.

▲	A	B	C
3			
4	RSU Mass Unit (kilograms) CON(14)	6.00000E+18	3.37500E+20
5	RSU Distance Unit (meters) (CON(9)	1.50000E+08	39148676.41
6	RSU Time Unit (seconds) CON(10)	1.00000E+02	750
7			
8	Universal Gravitational Constant (MKS)	6.67408E-11	
9	Universal Gravitational Constant (RSU)	1.18650E-12	
10	Speed of Light (RSU)	1.99860E+02	
11			
12	Time scale constant (ICN(35))	1.00000E+04	
13	Object Radius (meters)	2.72733E+06	
14	Object Radius (RSU-D)	1.81822E-02	
15	Combined size for softening checks (meters)	5.45466E+06	
16	Combined size for softening checks (RSU-D)	3.63644E-02	
17			
18	Object Density (kg/m^3)	1.76518E-12	
19	Object Density (RSU-M/RSU-D^3)	3.97166E+04	
20			
21	Number of objects ICN(10)	7.50000E+02	
22	Mass of Central Object (RSU-M) CON(4)	0.00000E+00	
23	Total Mass of simulation (kg)	4.50000E+21	
24	Total Mass of simulation (RSU-M)	7.50000E+02	
25	Maximum Radius for Object separation (RSU-D -- Con(6))	1.00000E+02	
26	Maximum Radius for Object separation (RSU-D -- Con(5))	3.00000E+01	
27	Volume available for Objects (RSU)	4.07569E+06	
28	Volume of N objects (RSU)	1.88838E-02	
29	Minimum distance between objects (CON(33))	1.00000E+02	
30	Volume needed for objects with CON(33) constraint	2.36047E+03	
31	Number of objects with CON(33) contraint in Max Volume	1.29498E+06	
32			
33	Escape Velocity from Max Radius CON(6) (m/s)	6.32807E+00	
34	Escape Velocity from Max Radius CON(6) (RSU)	4.21871E-06	
35	Circular Orbital Speed at CON(6) (m/s)	4.47462E+00	
36	Circular Orbital Speed at CON(6) (RSU)	2.98308E-06	
37			

5.2.3   Simulation Scenarios and Computational Equivalence

After decades of playing with this simulation and the spreadsheet tools now available, I have finally come to understand the concept of computational equivalence between seemingly vastly different simulation scenarios. The idea of computation equivalence means that simulations of vastly different scales will produce identical simulation results. Picking the simulation units yields simulation calculations which use the same scale of numbers across these different simulation scales.

Let's examine exactly what is meant by computational equivalence. I will use three different simulation scales as examples. First, I will use as particles from the astrophysical literature where they use a length scale of a hundred parsecs and force softening length of 90 pc. Next, from this scenario I will create two other equivalent simulations, one with length scale of meters and finally one with particles that are length scaled by the nanometer. I will refer to these simulation scenarios in the text that follows as Astro, Human and Nano scales, respectively. The table below will delineate the values for these three

simulations. We will assume Astro scenario 20000 particles in a uniform spherical mass distribution to start, with a collective density of 1×10^{-24} kg m^{-3}.

Real World Scale	Astro Scale (pc)		Human Scale (m)		Nano Scale (10^{-9} m)	
	Descriptive Units	MKS/RSU	Descriptive Units	MKS/RSU	Descriptive Units	MKS/RSU
Distance	100 pc	3.08×10^{18} m	1m	1m	Nanometer	1×10^{-9} m
Mass	2.1×10^4 M$_\odot$	4.2×10^{34} kg	1 kg	1kg	$100\times M_p$	1.67×10^{-25} kg
Time	~14 KY	4.33×10^{11} s	1 s	16.4 s	Seconds	1.27 s
G (MKS/RSU)	6.67×10^{-11}	1.799×10^{-8}	6.67×10^{-11}	1.799×10^{-8}	6.67×10^{-11}	1.799×10^{-8}
FS Scale	90 pc	0.90	0.90 m	0.90 m	0.90 nm	0.9×10^{-9}
Sphere Rad.	19 kpc	190	190 m	190 m	190 nm	190
Escape Vel.	1.38×10^4 m/s	1.95×10^{-3}	1.19×10^{-4} m/s	1.95×10^{-3}	1.53×10^{-12} m/s	1.95×10^{-3}
Crossing Time	2.68×10^9 Yr	1.95×10^5 It.	1.01×10^{-1} Yr	1.95×10^5 It.	7.86×10^{-3} Yr	1.95×10^5 It.
Average Density	9.95×10^{-25} kg/m^3	6.96×10^{-4}	6.96×10^{-4} kg/m^3	6.96×10^{-4} kg/m^3	1.16×10^{-1} kg/m^3	6.96×10^{-4}

As one can see from the above table all the RSU values are equal across these disparate scenarios. Positions, velocities and all iterative calculations use RSU and as such the behavior of these scenarios are considered identical. This will breakdown in the extremes, for example, when velocities get near the speed of light, clearly the simulation will diverge from the others if relativity is included. In addition, the "Nano" scenario is clearly in the quantum realm and cannot fully be understood within the confines of this classical simulation. However, just running these three scenarios, one will get the exact computational results from all three for the same number of iteration cycles.

5.3   The RCN Array; Run-Time Constants

This array is used to hold constants that were once included in the CON array from the inception of the simulation run but now are calculated at the start of the simulation and each time the simulation is restarted. In this way, I can reuse those elements of the CON array for other initialization constants/parameters. Currently, the array size is set to 10 elements and of course can be expanded in the future should the need arrive.

Note that RCN(6) is the force softening range and it is now calculated for the specific force modifiers used depending on collision type specified with ICN(18). For the forces used herein, we have modified forces for $r<rcn(6)*S$; where S is the sum of the two particle's radii. The ratio of the modified force (F_M) to the Newtonian force (F_N)can be written:

$$\frac{F_M}{F_N} = \frac{[-Gm_1m_2/r^2]\,f(r)}{[-Gm_1m_2/r^2]} = f(r).$$

Here I used f(r) as the force modification which is:

$$ICN(18) \leq 0;\ f(r) = 1 - e^{-\left(\frac{r}{S}\right)^3}$$

$$ICN(18) = 1;\ f(r) = 1 - \left(\frac{S^n}{r^n}\right).$$

Both the above functions decrease in value from unity when r is large. I chose to invoke these modifications when there is a 0.1% change in the forces from the Newtonian case. For each of these cases, it can be shown that:

$$ICN(18) \leq 0 \qquad \frac{r}{S} = [-ln\,(0.001]^{\frac{1}{3}} \approx 1.9$$

$$ICN(18) = 1 \qquad\qquad r/S = (1000)^{1/n}.$$

$$ICN(18) = 2 \qquad\qquad r/S = 1$$

Note that for the case of INC(18)≤0, the values of r/S is less that what I was using, so keeping RCN(6) as 2.5 will not change anything. For ICN(18)=1 and n=3 (con(17) in this case is -3), the RCN(6) will now be 10 which is twice what I was using previously. When using of larger values of n, RCN(6) will decrease in size. Note that the new option of ICN(18)=2 sets the value of RCN(6)=1

Below is the current list of constants used in the RCN array. The user will never have a need to update these constants as they are all calculated internally. This list is the current implementation to document for future reference.

RCN(1)	Unused
RCN(2)	Unused
RCN(3)	This holds the value of π. It is now set internally using the definition RCN(3)=ACOS(-1)
RCN(4)	This is the weighted average radial distance of all the objects, calculated internally.
RCN(5)	This is iterations to years conversion factor calculated internally.
RCN(6)	This is a new control for force softening control. This control multiplies the combined sizes of approaching objects to determine if softening is required at that distance. It is based on the value of ICN(18). • If ICN(18) $\le$ 0, then RCN(6) = 2. • If ICN(18) = 1, then RCN(6) = $(1000)^{1/n}$ • If ICN(18) = 2, then RCN(6) = 1 • If ICN(18) = 3, then RCN(6) = 39.
RCN(7)	Coefficient for the R^2 term for the piecewise continuous repulsive force.
RCN(8)	Coefficient for the R^1 term for the piecewise continuous repulsive force.
RCN(9)	Coefficient for the R^0 term for the piecewise continuous repulsive force.
RCN(10)	Now becomes the distance modifier to allow close encounter checks. It is will be modified real time to a smaller number should the arrays to hold close encounter pairs become nearly full.

5.4 Example of Riod3.ini File

Below is an example of the first few lines of the "Riod3.ini" file. The structure of the file is evident in this screen capture. It is important that this file structure is unchanged as each line is read into the code and numbers are extracted in a specific order. Only modify the numbers on the left are required, keeping integers as integers and real constants as real constants. If the order or lines are missing, the program will crash or worse, run with bad data!

```
Riod3.ini - Notepad                                                    —    □    ✕

File  Edit  Format  View  Help
*********************************************************************************
  Simulation I/O Control    See Notes at the End of file
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
1         icn(13) Simulation Logging:  =0; no logging or data files created;=1, Creation file =2 Bes
10        icn(12) Number of iterations between screen writes. riod 2
100       icn(9) Number of iterations between writes of data files
1000      icn(11) Number of files per directory
1         icn(4) Used to control position file contents; 0=no velocities; 1= vel. mag; 2 vel XYZ val
3         icn(3) Number of minutes before simulation status is saved
1         icn(2) if > 0 Controls how often stats.exe is executed if 1 will write file every new dire
0         icn(29) Number of minutes to run the simulation before a timed stop. Zero means run until
*********************************************************************************
  Simulation Threading and Velocity Change Array Control
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
4         icn(21) Number of threads/processors to use; if zero will use the maximum number available
25        icn(22) OpenMP chunks; 25 seems like a good number but others may be better; experiment
*********************************************************************************
  Simulation Constants and Units Control -- The BIG three
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
1.0d0     con(11) distance scale factor in meters; 1parsec=3.086d+16m; 1 milli AU=1.4957d+08; 1 lt-y
1.d0      con(12) is the mass scale constant Earth Mass=5.979d+24 kg; Solar mass=1.991d+30 kg
0.0d0     con(10) time scale factor in seconds. Set to zero to use CON(14) size to force calculatior
*********************************************************************************
  Simulation Distribution Control
>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
5000      icn(10) is the number of objects ****** DO YOU HAVE A BIG CENTRAL MASS, add 1 to total
100       icn(25) Number of Extended objects. Must be an even number; code will force it to be if nc
```

6 Auxiliary Programs

There are currently two auxiliary programs that can be used to understand what is happening as the simulation progresses. "Stats.exe" is a console program that prints to the screen information about each object. This useful program is described below.

6.1 Stats.exe

This program is useful to examine what is happening during the course of a simulation run. Below is a discussion on what the program output means and a digression to help understand some additional output. This program now has the capability to handle any number of particles and is no longer limited to 4000.

6.1.1 Program Output

This program is a console program and is best used while using a "Command Prompt" window. There is no input required to use this program. Typing "Stats" at the command line will dump the contents of the output to the console.

Before outputting the individual object status, the program calculates and outputs some system measures.

Below is a screen shot of the first few lines of output from the program:

```
stats_0000.txt - Notepad                                               —    □    ✕

File  Edit  Format  View  Help
 Iter.= 0000000000    Yrs.=  0.00000E+00    G=  9.99297E-08    c=  9.93398E+01
  PE= -1.49453E-01 KE=  1.33263E-02 TE= -1.36126E-01  Rv= 7.341E+01  Trel= 3.580E+11
       1   1.0E+00 9.00E-01 1.6095E+02 1.06E-02 1.42E-05 0.283  5639  6.89E+00  0.0E+00
       2   1.0E+00 9.00E-01 1.6137E+02 1.06E-02 1.42E-05 0.284  5640  6.38E+00  0.0E+00
       3   1.0E+00 9.00E-01 1.7490E+02 1.15E-02 1.59E-05 0.330  4683  1.26E+01  0.0E+00
       4   1.0E+00 9.00E-01 1.7530E+02 1.15E-02 1.59E-05 0.330 13744  1.23E+01  0.0E+00
       5   1.0E+00 9.00E-01 1.5657E+02 1.03E-02 1.34E-05 0.263 11351  6.87E+00  0.0E+00
       6   1.0E+00 9.00E-01 1.5540E+02 1.02E-02 1.34E-05 0.262 11352  6.66E+00  0.0E+00
```

The first line of output has the following displayed:

- Total number of iterations at the time of running the stats.exe.
- The number of years into the simulation as determined by the total number of iterations executed.
- The Universal Gravitational Constant in RSU.
- The speed of light in RSU

The next line has the following displayed:

- The total potential energy of the system
- The total kinetic energy of the system
- The total energy of the system as the sum of the above.
- This is now the Virial radius in RSU. See discussion below for this and the next output parameter.
- This is now the relaxation time in years.

What follows is an object-by-object list of all the objects left in the simulation. The following table lists the meaning of the values displayed above.

1	The object number
1.0E+00	The object mass in RSU
9.00E-01	The object radial size in RSU
1.6095E+02	The object distance from the COM origin in RSU.
1.06E-02	The ratio of the above distance to the distance it must travel to escape the simulation.
1.42E-05	The ratio of object velocity to the speed of light
0.283	The ratio of the object velocity to the simulation escape velocity.
5639	The object closest to the current object.
6.89E+00	The distance from this object to its closest neighbor above. Now if the nearest neighbor and the current object are bound gravitationally, this number will be negative.
0.0E+00	This is an estimate of the time it will take for this object to leave the simulation system in RSU. If non-zero, then this object is being tracked to leave the system. It must have velocity to escape velocity ratio greater than one, it must be greater than 40% of the escape distance as defined by the simulation input and it must not be close to any one neighbor by some amount, which is twice the original size of the system CON(6).

Note now the Stats.exe has three new lines of output as the last things output before it finishes running. The three lines contain the system X-Y-Z center-of-mass coordinates, then the total system velocity and finally the system total angular momentum.

6.1.2 Command Line Option

A command line option has been created for the purpose of outputting to a file the energy values for the saved simulation data files. There are three expected numbers on the command line, the start file, stop file and increment. These number correspond to the saved ".dat" files saved in the "pos\DatFiles" folder. For example, there should be files named "riod3_nnnn.dat" where "nnnn" is the number identifier. Operationally, the Stats.exe program will copy the file from the Datfiles folder to the Riod top directory and rename it "Riod3-Stat.dat" before opening it to begin the calculations.

For example, if one runs the command "stats 50 60 2", the program will dump the usual output to the screen but also create the following file:

```
StatOut-0050-0060-0002.txt - Notepad
File  Edit  Format  View  Help
0050 000050000000 -4.19419E-07  2.23422E-07 -1.95997E-07  1.25890E+02  1.00282E-09
0052 000052000000 -4.20064E-07  2.24309E-07 -1.95755E-07  1.26046E+02  1.00468E-09
0054 000054000000 -4.18417E-07  2.22662E-07 -1.95755E-07  1.26046E+02  1.00468E-09
0056 000056000000 -4.15228E-07  2.19878E-07 -1.95349E-07  1.26307E+02  1.00781E-09
0058 000058000000 -4.10905E-07  2.15623E-07 -1.95282E-07  1.26350E+02  1.00833E-09
0060 000060000000 -4.05803E-07  2.10603E-07 -1.95199E-07  1.26404E+02  1.00897E-09
```

Below is a list of the contents of the file:

0050	DatFile number
000050000000	Iterations into the simulation
-4.19419E-07	Total potential energy
2.23422E-07	Total Kinetic energy
-1.95997E-07	Total Energy
1.25890E+02	Current virial radius
1.00282E-09	Current relaxation time in years

Note that the file name incorporates the input parameters. The output values are the iteration number, potential energy, kinetic energy, total energy, virial radius and the relaxation time in years. This file will be overwritten if the same command line parameters are reused.

6.1.3   Time Measures of Evolving Many-Body Systems

A short digression to discuss the some of the useful time measures mentioned in the literature. In particular, the relaxation time, crossing time and collapse time I have found useful on occasion. I have used as a reference to understand certain simulation types the concepts of the crossing and relaxation times as used in Converse and Stahler[21]. In this paper, these useful concepts are defined as:

$$t_{relax} = \frac{N}{8\,ln\,(0.4N)}\;t_{cross}$$

$$t_{cross} = \left[\frac{8r_v^3}{G\,M_t}\right]^{1/2}$$

Where N is the number of objects, G the Universal Gravitational Constant and M_t is the total mass of the system of objects. The virial radius, r_v is somewhat ambiguously discussed but I found a definition[22] which is discussed in the next section below.

For collapsing systems of particles, the collapse time is a very useful measure. Since most of the scenarios I have been running for the last decade have been this type of simulation, I have integrated this timing method into the console output.

 One can find references to the collapse time in various places and there is a derivation of the free-fall collapse time here[23] and is defined as:

---

[21] http://astro.berkeley.edu/~stahler/papers/converse11.pdf

[22] http://en.wikipedia.org/wiki/N-body_units

[23] https://www.astro.uu.se/~hoefner/astro/teach/apd_files/apd_collapse.pdf

$$t_{collapse} = \left[\frac{3\pi}{32G\rho}\right]^{1/2}$$

6.1.4 Virial Theorem and Radius

A more detailed discussion of the Virial Theorem can be found elsewhere[24] but we know from energy conservation and the Virial Theorem that:

$$\langle E \rangle = \langle K + U_t \rangle = \langle K \rangle + \langle U_t \rangle = -\frac{\langle U_t \rangle}{2} + \langle U_t \rangle = \frac{\langle U_t \rangle}{2}$$

Where E, U and K are the total Energy, total Potential Energy and total Kinetic energy, respectively. Note that this theorem holds for any time during the evolution of the system of particles. Thus, for a system that is expected to reach virial condition as stated above, since energy is conserved, the initial total energy will equal half the total potential energy in the virial state. Thus, we can predict the final viral radius by using the initial total energy because $V_t=2E$.

$$\frac{1}{r_v} = \frac{2}{M_t^2} \sum_{\substack{i=1 \\ j=1 \\ i \neq j}}^{N} \frac{m_i m_j}{|\vec{r}_j - \vec{r}_i|} = \frac{2}{M_t^2} \frac{|U_t|}{G}$$

Where V_t is the total potential energy of the system of objects[25].

$$\frac{1}{r_v} = \frac{2}{M_t^2} \sum_{\substack{i=1 \\ j=i+1 \\ i \neq j}}^{N} \frac{m_i m_j}{|\vec{r}_j - \vec{r}_i|} = \frac{2}{M_t^2} \frac{U_t}{G} = \frac{4}{M_t^2} \frac{|E|}{G}$$

Thus the States.exe program uses the following to calculate the virial radius:

$$r_v = \frac{GM_t^2}{4\,|E|}$$

As noted above this is true at any point in time during the evolution of the particle system. However, most simulations begin in a specific state with designated total kinetic energy and potential energies. It is useful to be able to predict a final virial radius based on initial conditions. We know that for systems that conserve energy that the total initial energy (E_i) equals the final total energy (E_f), $E_i=E_f = E$ (Duh!). In the simulation, I use the quantity $\varepsilon=K/|E|$ to force the system's initial energy state. Remembering that for bound systems of particles, E and U are negative quantities.

$$K = \varepsilon|E| = -\varepsilon E$$

But:

$$E = K + U = -\varepsilon E + U$$

And,

$$E(1 + \varepsilon) = U \ and \ \frac{1}{|E|} = \frac{1 + \varepsilon}{|U|}$$

Since at t=0, we know from above discussion, the initial virial radius, r_v^i and the final virial radius, r_v^f can be written:

$$r_v^f = \frac{G\,M_t^2}{4|E|} \ and \ r_v^i = \frac{G\,M_t^2}{2|U|}$$

---

[24] https://en.wikipedia.org/wiki/Virial_theorem

[25] Note that V_t is a negative quantity and for bound systems E is also negative, and thus the use of brackets around these constructs above denote the absolute value of the quantity.

Substituting into the above, we get:

$$\frac{4r_v^f}{G\,M_t^2} = \frac{2r_v^i\,(1+\varepsilon)}{G\,M_t^2} \text{ and thus } r_v^f = r_v^i\,(1+\varepsilon)/2$$

This yields the expected results for when ε=0 and ε=1. We see that for ε<1, the system collapses to a smaller virial radius and thus for ε>1, the final virial radius will expand to a value larger than the initial virial radius.

Now CON(20) can be used to set a constant velocity for all particles and this is activated when CON(20)<0. So if K=ε|E|, then it can be shown that the |E|=|U|/(1+ε) and K=|U|ε/(1+ε). The program computes U from the particle positions and particle velocities are gotten from K=Nmv^2/2, where N is the number of SO in the distribution, m is the SO mass.

7 Riod Change History

Below is a table listing the Riod change history and possible future fixes and features. This history table was begun in February of 2007 and is fairly accurate since that date. The entries before that are reconstructed as best that can be determined from my logbook. There are many other changes that have happened over the years but there was almost a 10-year gap between entries in the logbook.

To do	Investigate adding relativistic corrections into the Isaac Subroutine and then totally removing the Albert subroutine. The relativistic corrections would be just another multiplicative factor in the gravity engine and should be quite doable. 2/14/2016
5/16/2023	After noting that the change on 5/9/2023 was restricting CON(58) between 2 and 5 was not enough, I changed the input values allowed to between 2 and 20. Twenty is probably too much but having the option of larger is better; the user needs to understand that larger means more particle pairs are being tracked and that if the number goes over 750, then collisions might be missed.
5/9/2023	I added a new option in the command line option to change parameters in mid-simulation. I added the first real number change in CON(58), which is the parameter that controls the collision check distance inside the gravity engine subroutine. I made this change to modify the current running VC collision check distance as it was defaulting to 2.0 after creating the sim from a different sim.
1/29/2023	I modified the CON(20) < 0 scenario from 1/13/2023. Now is this condition is met the code assumes that the SO positions are a Gaussian distribution and sues those positions to create Maxwell-Boltzmann speed distribution for the speeds based on the value of CON(20). For more details, see Section 3.2.4.3.
1/22/2023	I modified the value for energy ratio in the run string. Now it is logarithm based with the number being 100*LOG (K/|E|). If the ratio is zero, the run string will show a value of -999. I also modified the virial density part of the run string, putting a 0.5 addition to the resulting real number before the integer conversion. This gives a better representation of the number in the run string.
1/13/2023	I added a new feature to using K/|E|=CON(20) input. If CON(20) is enter as a negative number, the simulation will force the particle velocities to be a constant value based on the ratio of K/|E|.
11/2/2022	I fixed a problem with NaN for some variables (CON(62) to CON(65)). I learned that a NaN event can be tested for easily by checking if it is not equal to itself.
10/11/2022	Entering this a few days after the fact so I may not remember all that I did. I created a method for adding 8 exterior large masses to a simulation. These eight masses are positioned on the corners of a cube of dimensions 10 times the SO distribution size (CON(6) for now) and have the mass of the total SO count. They are also given the size for force softening equal to the SO distribution size (again CON(6)). This option is enabled when the number of SO is entering into RIOD.INI as a negative number. ICN(64) now holds the SO count and ICN(10) holds the SO count plus the number of large masses. ICN(63) holds the number of large masses.
10/8/2022	*Note to the below entry: I was able to create the scenario I wanted but as for the operation of doing the intended change to a running simulation, it is not clean and will need investigating if I make more simulations that need this feature.* Beginning on 10/5, I started making modifications to add a command line option to modify ICN(18) which the collision control switch. I am still testing but now it is possible

	to change the collision control of a running simulation. What this allows one to do is to make a new simulation from an already running one, one must copy the relevant time-stamp DAT file from the folder that one wants to start the new on from, then rename it to riod3.dat, since that is the current running status/.data file. Then run RIOD with the 2 command line option and pick 18 as the change mods. One can pick options of -3,0 or 1 for changing ICN(18). Once completed and backed out of the menu, a new entry will be appended to the Creation.txt file with the new RunString.
9/12/2022	I got a strange threading error that shut down one of my running simulations earlier this month. I did some testing and found that for new simulations, the code was hanging up for several seconds, depending on how many particles it was provisioned for. I could not find the problem but then checked older code and the code from earlier this year did not exhibit the issue. So I backed out of the changes which were mostly the CM core coordinates calculation. The odd thing was that none of that new code was multi-threaded. What I did discovered was that the call to the EnergyTotals subroutine that was added along with the other code so that a fresh virial radius was available, was called improperly and thus the arrays that are allocated in that subroutine were improper sizes. Once I made the changes for that subroutine call, the issue went away. I am not convinced that this fix will fix the problem that originally halted the code but time will tell.
9/3/2022	I think I have finally settled the code for finding the core mass center-of-mass coordinates. I have left the text file log in place for now to continue testing the method but seem solid now. I decide on a two-iteration method for the first calculation where the test begins radius at $1.2*R_v$ then finishes with $2/3*R_v$. Subsequent time entering the routine will just use the 2/3 value, since there is a saved existing coordinate to begin the calculation.
8/29/2022	Still testing the feature below and not completely satisfied with the results. I will update this log when and if I ever am.
	Note that I must have forgotten to update the behaviors for logging data in the 5/16/2019 entry. I updated that change today in the data logging section.
8/25/2022	The system core position feature I added on 6/13/2022 still does not seem to be getting core off set values that are anything close to what the Hist.exe. or Rplot.exe get as offsets. As such, I added code to create a log file to log each time the FindCm subroutine is called. The file is for now considered a temporary feature but will use it for now and perhaps take it out later. The file is called "FindCore.txt" as has many outputs, probably more than needed. There is a explanation string output every ten times the subroutine is called. Output to this file is fairly easy to understand, so no more information will be given at this time. If I keep the code in, I will add more details later.
8/24/2022	I was looking at the changes made on 6/13/2022 and found a typo in the code. Hopefully, the update code will now calculate the offsets correctly.
6/14/2022	I modified the algorithm that calculates the EX3:1 potential. I wasn't sure if what was implemented was correct, so I created a spreadsheet to examine and test the new algorithm. I am now certain that this calculation is correct as the numbers in the spreadsheet and those in the code agree. This same function had to be modified in the hist.exe program too!
6/13/2022	I added some code to determine the core offset positions and am currently testing the implementation. I have also added registers in the Riod3.dat file to hold the current system virial radius and the calculated XYZ positions of the offset. There is also a counter to tell how many times the actual offset positions have been calculated. The virial radius stored in CON(51) is now updated each there is a new position folder created.
3/28/2021	I needed to fix Riod.exe behavior when starting with the "Riod -1" option. I was picking up the SO size from the old data file. The intent of this feature is to retain positions and velocities from the older simulation and I guess the masses too. However, if you want to change sizes that was also being passed from the old data. Now I have not chosen to pass the SZ array from reading the old data. I did need to recreate the SZ array using input from the RIOD.INI file. So now, positions, velocities and mass are retained from the old data but not the particle sizes.
3/6/2021	I changed the way tracking distance is determined for elastic collisions. Previously, CON(29 using the input number from Riod3.INI but now it is set to 1.0. CON(58) which sets the tracking distance as

	CON(29)*CON(58) and now CON(58) is set to 1.5. This will reduce the number of tracked pair and reduce the number of nearmiss events.
2/1/2021	One more Plot enhancement today. I now output the particle density to the plot it is outside the bounds of the plot scale.
12/31/2020	I cleaned up the way total energy is calculated in three programs, Riod.exe, State.exe and Hist.exe. I finally figured out (D'oh!) that the reason my total energy calculations were not perfect was that I was calculating it using pure Newtonian physics for all the particles and ignored those particle pairs that were close enough to have force modifications. As such, the potential energy for those particle pairs would be underestimated. For force modifications that use the EX3:1 profile, there is no closed form for the potential energy and so I needed to figure out how to calculate the potential energy for pairs with those force modifications. As of this writing, these changes were tested in Stats.exe and Hist.exe but not in Riod.exe; testing to follow probably when I start new simulations as that is the time when the subroutine is called most often.
12/28/2020	With the arrival of a new computational system with 24 possible threads, I have increased the maximum number of the threads to 32. I also had to change what I do to initialize the arrays. The Riod3con.dat file was being used but I included that for a reason but I am not sure why. The problem introduced by this was that when trying to change the number of threads for an already running simulation, reading data from the Riod3con.dat file always reverted back to the original number of threads. I changed this initial read back to the current Riod3.dat file. I am sure there will be consequences for this change but I can't think of what right now.
12/24/2020	Insignificant change adding a space in the output format for Hits.log for vanishing collisions.
12/16/2020	I fixed a section for the Runstring where the elastic collision scenario was only a single character and needed two. I also fixed the string for when reading in an old initial input file, there are now five zeros in the file name that needed to be accounted for. I have tested this scenario and seems to work now.
10/30/2020	I made some changes to fix an issue with the starting of Riod when the number of thread available is less than what is provisioned. The ICN(21) which is the number of threads was changed after reading in the data file and I now force it to be what is calculated based in number threads available from the system the provisioned value. This was necessary because the arrays are allocated based on the original read of data and then ICN(21) changes later.

I also changed the maximum number of threads to be 32 to account for future CPU that may have that many threads available. |
| 10/18/2020 | I cleaned up some the EXn distribution creation method to match what I do in the Mdis.exe program. I added a the EXN function to the code and use it for the 4 EXn, n=1,2,3,4 options. Also, I found a weirdness in the Mdis.exe program when determining the check distance and I modified that to make more sense. This tends to increase the value needed for CON(33) when creating distributions, so that isn't necessarily bad. I did a quick test of these changes for the four cases above.

I am also trialing a change in the execution of the Stats.exe program. I have now used the "Start" option to send the execution to a new window so that Riod.exe doesn't halt while Stats.exe runs. This hopefully will be only a minor annoyance to the workstation but we shall see. Note, I am now testing this with the start command in minimized mode so that the desktop isn't affected by the Stats.exe execution (10/20/2020) |
| 9/9/2020 | I added a new option to add a Hubble constant related velocity to simulations. At this point, the current Hubble constant is hard coded as 70.0 km/s/Mpc. When the input of con(11), the distance RSU unit is input as a negative number, the program will do everything the same except just before completing the creation phase will update all the velocities by adding the Hubble value as component in an outward direction from the SO position. For more details, see section, |
| 8/26/2020 | I made a change to the output of the Nearmiss.log file which changes the format of the text line that begins a new instance of the file. Hopefully, this change will omit the space before the output so that text will align with the numbers when pasted into a spreadsheet. |

7/29/2020	I made some changes to the Hits.log output file today for the case of replacement collisions. This is strictly cosmetic in nature but added some more output to help test the new velocity cases.
7/23/2020	I made a change today in how multi-EO configurations are created. I was using the maximum SO distribution size to determine the DR step size in the final EO distribution. I decided that was too large because DR becomes too large. I settled on using the distribution scaling factor, which is just CON(6). This gives a finer DR step size but there is something still bothers me about what is going on.
7/18/2020	Added new subroutine to calculate unit vector velocity directions based on the inputs from the polar angle limits input by the user. This routine can be used by all the other creation subroutines and perhaps I will include in them over time. Right this new routine is used in the replacement collision subroutine for determining the direction of the replaced particle's velocity.
7/1/2020	Recently added the collapse time to compare timing with on the command line output and noticed that there was a mistake in the calculation. I fixed that this morning.
6/17/2020	A couple of updates. I happened to find and fixed a problem in RC mode where the initial new velocities were not configured in the "z" direction. I have not tested this but soon will run some RC scenarios. I changed the text output to now show the time as compared to the initial collapse time. Before it was set as the crossing time and that didn't make much sense for these collapse scenarios that I am running. The program should use the initial collapse time for this comparison. The initial collapse time is now saved in CON(4).
6/13/2020	More changes to the VC and RC behaviors. I noted that the interaction radius is determined by the relative impact velocity but the interaction distance becomes twice that radius; like two balls touching at their combined radius. I also increased the check distance to 6.5 times the smallest provisioned interaction radius. This was because the initial interaction speeds are lower and thus will have the collision farther out.
6/2/2020	I made some changes in the VC behavior and in the hit.txt file and the nearmiss.log for the VC case. For VC, three are now three checks for a collision. I put in the case where the current separation distance is less than the sum of the sizes. Presumably if the scenario is sized conservatively enough, the actual closest approach will always be less than sum of the SO sizes. However, that may not always be the case, so I now force the collision if less than the sum of the SO sizes. I also put some text in the HITS.TXT file to indicate what the columns are. Finally, changed the output to NEARMISS.LOG to have better understanding in the VC case.
5/14/2020	I had to fix an issue I created in the last entry. For the case where the time increment, CON(10) is given as input, the CON(14) size in RSU was not calculated correctly. It needed to be divided by the length unit to get RSU.
5/5/2020	I made some changes to the creation subroutine today to improve the initial determination of the SO size (CON(14)) and the time delta (CON(10)). There was too much twisted code that created behaviors which were not entirely what I wanted for cased where ICN(18) was either -2 or -3, VC and RC collisions. There should be no other impacts on usage from this change, just slight behavior differences for those ICN(18) cases. In the case where there is no CON(14) or CON(10), it will compute them from the annihilation cross-section.
4/1/2020	I found and hopefully fixed a potential problem with my OpenMP code for the EnergyTotals Subroutine. I was passing the number of threads to the subroutine when I should have been passing one less than that as the arrays count from zero to NT=CON(21)-1. The OpenMP loop uses the thread number to be NT+1. I wonder if this will fix some of the unexplained crash issues I see but I doubt it. Also, for multi-EO scenarios, I added the calculation of the initial EO virial radius and store it in CON(60).
3/31/2020	The RunString continues to be a work in progress. I added a force softening length indication to the string today and reduced some of the other superfluous string components. The forces softening length is now represented by three numeric characters which are now 10 times the softening length. See section 3.3.4 for more details.

3/26/2020	I changed the particle distribution profile creation code. Now for ICN(8) and ICN(28) >0, this code places alternate particle positions in different coordinate quadrants. Note that alternate particles are always in the opposite quadrant. Now, every other pair is put in a different quadrant in an attempt make the resulting distribution more or less balanced. Thus using numbers of particles in each of the distribution types multiples of 8 will make each quadrant be hit an equivalent number of times.
3/1/2020	I added a new control point for the energy ratio. I was using CON(20) for the multi-EO scenario as well as the single EO distributions. I now still use CON(20) for the single EO configurations but now I use CON(34) for the Multi-EO configurations. When doing this sort of change, the RIOD3.INI file, the read in subroutine and the code creation all need to be changed.
2/27/2020	I learned from the SF people that the system call "EXECUTE_COMMAND_LINE" subroutine call is now the standard way in FORTRAN to execute a system call. I am trying this now to see if the seemingly random memory segmentation faults will no longer happen. Hopeful but doubtful.
2/26/2020	I cleaned up some of the spaghetti code in Pdata subroutine and also bit the bullet on making the position data file structure to be expanded. Now the position file structure is the following: "pos/NNN/NN/NNNNNNNN.D0N" The changes is that now there is three digits for the top directory. This change will rarely be needed but as I have started to make runs with fewer files per folder, this may come into play more. At this point, I have changed nearly all the I/O structures for the simulation.
2/25/2020	I changed all the integer and real constant arrays to be now 80 elements. I did this to make sure there were more provision-able parameters available in the future and since I now have new data file format for other things, this will make the transition to this new "release" semi-complete. So far no new parameters have been added but will update the lists above when things happen. Also, now that the Riod3.ini file has a new format, that format must change with each added parameter that must be provisioned.
2/23/2020	I removed all references to RIOD2 from the code as now all files for this release going forward will be RIOD3. Auxiliary programs, Stats.exe, Rplot.exe and Hist.exe also were expunged of the old file name format.
2/22/2020	I completely updated the Riod3.ini file to a new format. The program now reads in the new file, Riod3.ini. All the input parameters are now arranged into common groupings so that various inputs are modified in a similar area of the flat file when editing. For example, all the SO distribution input parameters are grouped together so that it is easier to modify those groupings all at once. Note that the structure of the new Riod3.ini is therefore completely different. The parameters should be edited but no new lines entered into the file. The file structure must be maintained exactly as it is. If not, the program will either halt on a bad read or perhaps continue with bad input.
	Because of the above changes and that there were other programs using the INI file to set up initial arrays, I have now created a new file that holds all the ICN and CON array elements that can be read in by other programs to get these constants. The new file is riod3con.dat and resides in the home directory. It is also deleted by using the New.bat starting batch file. Rplot.exe and Hist.exe needed to read from this file to be able to allocate arrays.
2/20/2020	I modified the classic distribution dither component in CON(9). It is now based on the minimum SO size and CON(9) multiplies that size as the dither amount, which is also randomized over that size range. This change was needed because the previous method allowed the average particle spacing to be too small and this caused potential issues with initial conditions. The new method makes on modest changes to the positions and thus keeps the spacing as defined in CON(33) to more reasonable values.
2/16/2020	I fixed some things and modified a feature for one scenario. I updated the Run String to be correct for Multi-EO configurations. Now the Run String shows the distribution type for the EO distribution using ICN(28), just like as the SO distribution uses ICN(28).
	I have added the feature of setting the energy ratio with CON(20) to the Multi-EO scenario. This is done after the EO distribution and velocities are created and then the gamma correction is determined and the velocity vector magnitude is modified. This is only applied to the EO distribution as the SO

	distribution does not affect the total energy as much as the EO distribution. I am not sure that the gamma correction works with the single SO distribution yet as I have not tested it.
Done about 2/4/2020 To do	Create new options for large masses to use a different mass density profile than what is used for small objects. For example, using the $M(R)=M_T r^3/(a^3+r^3)$ model for very large masses would allow creation dark matter distributions in galaxy sized objects. A mass scaling parameter would be needed and probably tied to a standard density for such objects. New simulation run types would suggest themselves with this new option. Also, I need to create a single large mass to dominate the many possible EO that can be created.
2/4/2020	I cleaned up a behavior when creating a profile distribution (ICN(8) or ICN(28) >0). The calculated interior mass is encoded as the fractional part of the particle mass. However, the farthest interior particle has no mass inside its position, so it will have no mass and thus no initial speed. I corrected this by splitting one mass between the two most interior particles, so the code gives them half a mass unit to start with.
2/2/2020	Since the last update (and completely negating it), I have spent the last two weeks completely rewriting the Create subroutine. My original code was so spaghettified that I could no longer follow and keep track of all the options and creation flow sequencing. I have now created a separate subroutine for the (currently) three main final distribution type and their individual variants. The three variations are first the classic Riod distribution of SO into a single EO in either the spherical mode or disc shape mode. Second, is a distribution of SO into a single EO using the distribution types defined for ICN(28)>0. Third is a multi-EO assembly using only the distribution profiles as per ICN(28) for the individual SO distributions and the EO distribution using ICN(8) to determine the distribution profile.
	Code for doing all the above is more modular in that subroutines that support the creation calls are mostly reusable for different scenarios. For example, the call the create a profile is used for individual SO in creating an EO but is also called for creating the distribution of EO. I also create new temporary arrays to hold boosted EO mass positions and velocities.
1/22/2020	Many changes have been made to the Create subroutine. I have streamlined some of the code as it was very confusing and I add many comments. Essentially, the initial particle distribution is created in a new way. First, I determine if it is to be a multi-EO configuration and if there is a central big mass. For a multi-EO configuration, I set that up first using the number of EO as the first elements in all the particle arrays. I used the CreateProfile subroutine for creating the distribution using ICN(16) as the profile type. If there is a big mass, that is added to the boost velocities of the EO. Then those particle properties are stored at the end of the particle arrays for use later. Next, the distribution of SO is created, This can be done with either old method or the CreateProfile method, ICN(28)>0. For multi-EO configurations, the SO distribution is copied into the other array's slots after the specified rotation and boosting the positions and velocities. Finally, big mass is added into the last member of the particle arrays. All these changes are untested at this point but that testing will begin shortly.
1/15/2020	I made changes today to fix small issues with the initial particle distributions. For ICN(28)>0, I was creating distributions based on an infinite number of particles over the range $0 < r < \infty$. The reality of the distribution creation is that there is a minimum and maximum radius to create the distribution and those limits affect the distribution integrals. I have changed the distribution creation method for the following distribution types with ICN(28) values in parenthesis, Uniform (12), Gaussian (17), NFW(19), Jaffe(20), and Exponential variations (1,2,3,4,11). See section 3.2.3.2 for more details.
12/2/2019	Fixed a behavior for disc shaped distributions. Specifically, when there is an inverted wedge disc, I was not accounting for the radius properly. I start with a cylindrical coordinates distance from the center, then determine the z-wedge component. But when figuring the spherical coordinate theta angle, I was using the sum of the pho and z parts to get the final spherical radial distance. This was a simple fix but I was unsure how this worked previously.
11/8/2019	I made a few changes to the Nearmiss.txt file output. I expanded the output format for several of the parameters as I needed more precision for testing purposes. The output parameters have not changed, just the formatting.

10/11/2019	While debugging this halting problem, I did some code inspection and found bug in the multi-threading code for the "EnergyTotals" subroutine. I was dimensioning the threading arrays one element too small. I am not sure this is the problem I am looking for because this subroutine is not called often but could have caused memory issues in the long run. The bug most certainly would have produced incorrect energy calculations and that was dumb!
10/10/2019	I have been trouble shooting the code for the last couple days and made some changes to perhaps help the issue. Most of these changes were to initialize variables that were not properly initialized at startup. None of these changes has helped or hurt it seems.
	I am also looking at changing the RunString format a bit. I have changed the host name portion of the string to only be four characters, the first character of the hostname and last 3 in blank characters now will be the hostname identifier. I also changed how the number and distribution of the particles are displayed. This was done in anticipation of possible changes in how the initial conditions will be created. I still haven't decided what that will be yet but big changes coming for how that is to be done.
10/7/2019	I added a new option for ICN(28). Now picking ICN(28)=17 gives a Gaussian distribution for the intial configuration. See Section 3.2.3.2 for the details.
10/3/2019	I changed the output to the creation.txt file so that now, I can paste the end output into a CSV format analysis spreadsheet. The top of that paste is the two line with the total energies and the virial radius, and crossing, relaxation iteration times. Including that directly into the spreadsheet means I don't have to rely on other input methods like the initial Stats file.
	I also changed the command line option to alter internal controls, where ICN(12) was limited to changes between 20 and 100000. I changed that to between 1 and 100000, since for very large simulations, one might want to screen print after every iteration.
	I also made some cosmetic changes to the "Nearmiss.log" file, changing the spacing of the text descriptors.
9/30/2019	I needed to change the profile creation when ICN(28)>0. Reason for this was that the distribution of particles once the profile was created deviated from expectations where the distribution was spherical in shape. The ICN(28)=0 behavior was one where the particles were created in equal but opposite pairs. I needed something similar to that for these other profile cases. In the cases for ICN(18)>0, I now create the profile with pairs that are opposite from each other but have similar radii but not the same. In this way the profile is more spherically symmetric. I will use this for a while to see if it works out better.
9/29/2019	While testing some different profile scenarios, it became obvious that the CM velocity was not zero at the start of the simulation and thus the center would drift over time. I corrected this with a change to correct the CM velocities once the final initial EO positions and velocities were finalized. There is also an entry in the Creation.txt file for when this happens.
9/28/2019	I made some small changes to the output in the Create.txt file. Now the end of the file will print out all the initial parameters in the same format (e.g. "NNN , Description") so that this can be pasted directly in the analysis spreadsheet in the CSV format. This is done for the ICN, CON, RCN arrays as well as the run descripted string.
9/16/2019	I made some changes to when the code checks for the number of processors/threads. I was doing that too late, which was after where the threaded arrays are allocated. I corrected that during the creation phase. It also needed correction in the start phase for the continued simulation since a simulation may get restarted on a different machine with a fewer number of threads are available.
8/18/2019	I changed the characters in the fix below to be less confusing with other characters. See Section 3.3.4 for more details.
8/18/2019	I made a small change in the character string today. I included a character to indicate if initial velocities are inward, outward or neither. The character in the string depends on the values of CON(21) and CON(22) a single EO and CON(40) and CON(41) for multi-EO configurations. See Section 3.3.4 for more details.

7/31/2019	I added a new action to help alleviate the issue of accidentally running the "New.bat" file on an old or existing simulation run and then deleting everything. This has happened more than once and I am putting this code in to help stop it. Now when the first new position file folder is created (ie. pos/00/01), it will check to see if the "New.bat" files exists and if so, renames it to New.txt. The text file is benign and cannot be executed but can be rename if needed.
7/24/2019	I have added a feature to now scale the kinetic energy to total energy ratio using CON(20). When CON(20 is greater than zero, it will use that as a ratio to scale all the kinetic energies by a factor gamma.
6/27/2019	I improved the output of the simulation descriptive string. Now for simulations with more than one EO, there is a new string with the number of particles in the EO and the number of EO represented. For example, for 5000 total particles and 20 EO the string appears as "N250x20".
6/25/2019	I fixed an issue with the behavior for 2 EO. I wanted for two Eo that the EO would be positioned in Y-axis and the velocities would be in the X direction. This wasn't working correctly and after much stumbling around in the code I figured out what had to be done. This seems to be working correctly.
6/14/2019	I made a modest change to the code to ensure that ICN(48) has a value when saved to the POS files. ICN(48) is the number of pairs of objects being softened during the current iteration. This number has become important for analysis purposes.
5/21/2019	I have updated RIOD.EXE to have output to handle more than 10000 SO. The code was already capable of creating and running simulations with no maximum number (up to the limits of the memory) but now the IO strings can properly output the number of SO with the limit of 5 characters meaning 99999 is the IO limit. This should not be a problem for the rest of my lifetime as I do not anticipate doing simulations larger than that any time soon. I also updated Stats.exe, Hist.exe and Extract.exe as these programs also had a 4-character limit for the number of SO. I also changed from of the output structure for Stats.exe to make the output more spreadsheet friendly but not eyeball friendly.
5/16/2019	This code is really messed up, there is some sort of issue that causes a crash and I can't find why. My only real recourse is to fall back to code that was working before these changes. I am restarting with code from 4/24/2019. Here is how I am redoing the changes, making sure I test better each time: 1. First the most important change I have made since then is the change the position files to double precision output. Since this was a fair benign change in RIOD.EXE, this change seems to work just fine. Other programs I changed to use this data seem to work as well. 2. I want to reduce the file writing for collapse scenarios, so I will use ICN(13)=1 in those cases. However, the Stats.EXE program is linked to ICN(13)=2 and greater. I have now linked it to run when ICN(13)=1 which is a very simple change. 3. Next, I want to limit the maximum number of tracked close particles. Currently the arrays are sized to 750 elements. The simplest change here is to tell the array index be a maximum of 750. I think where I got into trouble was when I assigned ICN(6) to be the array size and then allocated that size. I may not put that back in. Added the initial shell size ratio to the simulation descriptive string. For CON(6)<5*CON(5) this is a shell condition and the string id UShXX where XX=INT(100*CON(5)/CON(6)).
5/16/2019	4. <mark>I made a change in when close SO are tracked. Before, they were always tracked but in certain simulation scenarios, there are times when one does not want to track these as there can become many to track. So I made the change that if ICN(13) does not include the Hits, Lost, or Nearmiss logs, i.e. ICN(13)≤1, then there is no point in tracking close encounters. I also only allocate memory for 1 element of the tracking arrays no matter what ICN(6) is set for. This is useful when doing collapse scenarios.</mark>
5/15/2019	I made some changes trying to fix a problem that only one of my machines seems to have. I get a "Bad File Descriptor" error and crash on Surfbud. This only happens with specific memory usage. If ICN(10)=8000 and ICN(6)=6000, this crash happens. I can't seem to get it to happen with other combinations. I suspect that this is a memory issue but can't seem to find it.

5/14/2019	I changed Riod.exe position files to now have full double precision output. This change was precipitated by changes to the analysis programs which suffered when using single precision numbers. Previously all position files were single precision. I have also updated the auxiliary programs that read this data and hopefully they will work as expected. Programs updated are Hist.exe, Extract.exe and Rplot.exe. Note that Rplot.exe still uses single precision numbers to plot, they are converted at read time. A new suffix for the file names "D01 or "D02" when saving velocity information.
5/11/2019	I fixed a problem with the implementation from 5/4/2019. The array index for the tracked pairs in the threaded part of the code was still allowed to get larger than the allocated size. The fix was to be sure the max number was done at the time increasing the number beyond the allocated size. I also added on more thing to the Simulation description string. For uniform shell distributions, I added two numeric characters to give the ratio of the max to the min size. So the number is Int(100*CON(5)/CON(6)).
5/8/2019	I updated State.exe to now have allocated particle arrays. As a result, the number of particles can be above 4000. I also changed over the code to FORTRAN 90 compliant. In addition to the above, I updated Hist.exe and Rplot.exe to now have arrays allocated at run time based on the total number of objects at simulation creation. These changes are not completely tested but do seem to accept more than 4000 objects without problem.
5/7/019	I removed the code I put in the previous entry to change the check distance. CON(58) is back controlling the check distance to fill the close encounter arrays.
5/4/2019	I am running larger simulations and as such was running into cases where the tracking of close particle arrays were not large enough. I changed code to accommodate an input from ICN(6) to allow this to be sized at run time. Now these are arrays are allocated at run time after reading in the initial RIOD2.INI file read. In addition to these changes, the arrays can still be oversubscribed in some simulation scenarios and this I have forced the maximum tracking to be the be the ICN(6) value. Now the code will plow through the case where the tracked pairs hits ICN(6) but note that some close encounters will not have Nearmiss.Log events created because of this limitation. Also some collision events may be missed too… Not sure what to do about that but since collision events affect particle numbers, hopefully this will not be an issue. In addition, I added some code to change the close encounter tracking distance if the arrays start to get too full. I am not sure how this will work but when the array gets to 75% full, the multiplier changes from the value of ICN(58) times the combined size to 1.1 time the combined size. It won't change back until the array is under 25%. Now RCN(10) is the scaling number modifying the combined size.
4/24/2019	I changed the output structure to the console to now alternatively display the simulation time in years and crossing times.
4/20/2019	I changed the number of tracked particle pairs from 500 to 750 today as a running simulation had over the 500 limit. I need to rethink how this tracking is done perhaps. I fixed one issue that affected the behavior noted above. There was a link to ICN(19) in the definition of CON(29) making it larger which shouldn't be the case since ICN(19) is a small object collision case for ICN(18)≤-2. I removed that dependency.
3/20/2017	The changes I made on 3/11/2019 for velocity creation method did not make it into the specific profile creation process. I added that feature to that subroutine today so that when ICN(28)>0 indicating a profile selection, that velocities would be treated the same way. I also added this capability to EO initial velocities but since there are not enough free input parameters in Riod2.ini to accommodate full degrees of freedom on these angles, I am use CON(40) and CON(41) as the azimuthal minimum and maximum angles for theta and phi will be random between 0 and 360 degrees.
3/13/2019	I fixed the way the initial positions of particles are dithered today. I was doing it wrong and now seems to be correct.
3/12/2019	I made some changes in the descriptive string. I removed the time and length string elements and added the value of G in RSU. Using only G with the Mass gives a better indication of the simulation scale and

	that is why I am decided to make the change. I also added a new descriptive element to describe the initial mass distribution. See Section3.3.4 for more details.
3/11/2019	I added a new feature today to allow simulations to reuse positions, velocities, masses and sizes from past simulations. In doing so, there is now a new command line option to cause this to happen. What happens is that the initial data file saved from a previous simulation is now read in and the particle properties are reused in the new simulation. Changes in simulation properties such as collision type and other properties not specifically tied to the particle properties and can be changed in the Riod.ini file, since it is read in as well and the configurations are intertwined. See Section 2.4.3 for more details.
3/10/2019	I added code today to "fix" a perplexing behavior in the code where when the code determines that two objects are close and calls the "Collck" subroutine. What was happening was the performance would slow incredibly and CPU usage would drop. It took some time to realize that it was the close proximity case that was causing this behavior change as it appeared to be random. What I discovered was the program was spending in inordinate amount of time in the "EnergyTotals" subroutine and thus causing the code to fall out what I can efficient use of the processors/threads. I fixed this in two ways, first I realized that the "EnergyTotals" subroutine only needs to be called if there is change in the number of particles. Secondly, I turned this single threaded calculation into multi-threaded logic. This seems to have fixed the problem but I still have some other issues with how threading is used by the OS where the full CPU usage for some (not all) running simulations will not use their full requested CPU resources unless there are other simulations running to use the remaining processing power.
3/8/2019	In order to get a more spherically symmetric initial distribution of particles, I added some code to rotate the initial unit position vectors. I rotate the position unit vectors by a random angles, first about the y-axis, then by the z-axis. The random angles are computed by multiplying the random number by 10π.
3/7/2019	I think I found some "bug" in the computation engine today where I omitted some variables from the OpenMP directives to be private. They were not declared at all.
2/20/2019	I am now using ICN(5) to control how many attempts are to be made at getting a new position point in the EO creation phase. It was set internally to 400, but now using ICN(5) one can make this be larger or smaller.
2/19/2019	Changed the initial velocity vector method to be more adaptable. First moved CON(23) to CON(27) and CON(24) to CON(28). CON(23) and CON(24) become the Min and Max spherical coordinate phi for the velocity vector. CON(21) and CON(22) become the velocity vector theta angles.
2/7/2019	I have added the ability to have SO now force softened using the Plummer profile. This option is used with ICN(18)=3. See Section 4.3.2.7 for more details.
2/6/2019	I have added a subroutine to create a descriptive string for any new simulation run. I am still working the details of this and also need to figure out if I want to or even can use this in some other way. This is a work in progress.
2/5/2019	Changed the behavior of the Nearmiss.log headed. I now use ICN(23) to tell the code to write the header only for the first entry of each log. ICN(23) is set internally to zero to begin and once a first entry is written to the log, it is set to 1.
2/4/2019	I found problems with the implementation of the RC events and have moved to fix those issues. The RC events worked fine the last time I used it but somehow I messed it up. In addition, the Collck subroutine was a real spaghetti mess, especially the RC section. I moved the RC section into its own subroutine to help separate it from the other types in Collck. I will probably do the same for the other collision types as need warrants.
	I also added a descriptor line to the top of the Hits.log file to help describe the output quantities.
2/3/2019	I changed the what values are stored in CON(51), CON(52), and CON(53). Previously, they were the initial virial radius, initial total potential energy and initial total kinetic energy, respectively. Now these values are the current values after any energy changing collisions that happen. Thus these will change for some simulation collision types. This change was done so that for replacement collisions, the new replacement particles use the current virial radius to scale the new positions.

1/29/2019	I created a new option to include velocities in the position file output. I used ICN(4) to control this with the following:
	• ICN(4)=0, same file name and structure as always. • ICN(4)=1, include the magnitude of the velocity with the position and mass data. Position files now have an extension to the filename of ".001". • ICN(4)=2, include the XYZ velocity data and now have an extension to the filename of ".002". All my auxiliary programs that use to POS data must be changed to account for this new behavior.
1/22/2019	Around this date I added a description string to the beginning of the "Nearmiss.log" file so that there is some clue as to the output. It is still somewhat mysterious but better than nothing. This string will be added when after the old file is copied out of the main directory but apparently doesn't do it for the first one.
1/8/2019	The immediate changes below were necessary with the new "feature" I am testing. I have added the option for using a cross-section to determine the interaction distance for collisions when ICN(18) ≤ -2. ICN(19) turns this method on and CON(30) holds the cross-section in MKS units.
1/8/2019	New parameter configurations are as follows • *ICN(19) becomes new collision control flag to use cross-section number in CON(30) for controlling collision distance.* CON(30) holds the cross-section in MKS units
1/8/2019	Making some changes on the ICN and CON array indices as follows: • ICN(19)→ICN(20) which is currently unused. Moving this because it aligns with other Simulation control parameters. Will used ICN(19) for another collision control flag. • CON(30)→CON(20) which is unused. Moving this because it aligns with other simulation scale configuration parameters. • CON(34)→CON(35) which is unused. Moving this because it aligns with other Multi EO configuration parameters.
12/18/2018	• I improved the output for situations where the number of iterations required for time retarded forces. Previously if the number was greater than 9999, the output would show overflow as "****". Now it will show "9999" if greater than that number.
11/16/2018	I changed the code to now allow use up to 20 threads as now I have one computing system that can make use of that many cores/threads.
11/16/2018	Fixed a logic bug in when the rank-3 arrays are allocated. When entering the program through the command line change parameters interface, the number the threads can be changed. Thus the threaded arrays have to be allocated after this subroutine and process is complete. I just moved the allocation code and the array zeroing down after the call to the parameter change subroutine. Actually, I had to fix it again as I didn't have the allocation positioned within the if-else portion of the set up.
11/12/2018	I began yesterday to make all the data arrays in RIOD.EXE to be allocated at run time. This is a continuation from work from the 10/9/2018 entry. Now the position, velocity, velocity change, mass and size arrays are all allocated at run time. In doing so, my runtime memory footprint is significantly reduced. For example, before the 10/0/2018 changes, RIOD could and would gobble up over 1 Gigabyte of system RAM. The changes on 10/9/2018 reduced that to under 100 Megabytes. Today's changes take a typical simulation scenario of say 2000 objects, and now has a memory runtime footprint of 6 Megabytes. Internally, the changes are subtle. For new simulations, the program now reads in the Riod2.ini file to fill in the initial 30 entries of the ICN array. From this initial read, the position velocity, velocity change, size and mass arrays have their sizes allocated. Then the Create subroutine is called set the two other parameters that determine how to allocate array sizes for the three rank-3 arrays used to do the threaded velocity changes in the compute engine. I now also allocate those there arrays in the main program once

	and pass them to the compute subroutine via a call. For running new problems, a similar first read of the Riod2.dat file gets the ICN array extent parameters and then all the arrays are allocated. Operationally, there are no changes to how the code works or how the computing engine works. These changes are all transparent to the user.
10/29/2018	I made a small change to the output in the Creation.txt file. The density output to that file is the SO density, not the EO density, so I made that more explicit in the output text.
10/23/3018	I made two behavioral changes today. I decided that using the initial relaxation time for the screen output time for gas ball scenarios doesn't make sense, and I have convinced myself that the crossing time is a better time representation since it only depends on the initial density and not the numbers of objects like the relaxation time. I also finally decided that ICN(12) should just be the number of iterations run between screen writes to the console. I have also changed the text for the command line option to change constants.
10/22/2018	I added a new elastic collision option; ICN(18)=2. This option uses a piecewise continuous quadratic force inside the combined radii of the colliding objects. Changes include adding four new constants, RCN(7), RCN(8), RCN(9) and CON(32) to represent the constants needed for the interior force. I am testing this now and it does not look as promising as I had hoped. I will leave this option in the code for now. See Section 4.3.2.5 for more details.
10/10/2018	While debugging the changes from yesterday, I noted some stupid spaghetti code on how ICN(15) was set initially. Recall that ICN(15) controls how many temporal iterations the code uses for time deferred forces. Now, if ICN(15) is set to zero or less, the code will calculate the maximum number of iterations needed to accommodate the simulation size based on the speed of light and the distance objects must travel to escape the system. If ICN(15) is set greater than zero, it will use that value provided it is less than 1000.
10/9/2018	This is the demarcation for converting Riod to a 64-bit program. Going forward it is my intention to only use 64 executables for Riod. In addition to the 64-bit architecture, I have also converted my "old" FORTRAN 77 (and older) code to a more modern FORTRAN 90 like structure. For example, all the DO loops that ended on number lines are replaced with the DO-ENDDO structures. There are a lot less continued lines now as well, making the code easier to read and work with. These changes were precipitated by some memory issues and a change to new SimplyFortran version. The biggest change I made to the code was taking the three largest arrays in the compute engine and allocate the array to the exact size needed within the subroutine. Since these arrays were had three dimensions, they were needing a lot of memory. Now I simply allocate these arrays at the time of entering the subroutine, zero them out and once finished with them, I deallocate the array. This has a significant impact on the memory used during run time but an unexpected benefit seems to be that the code runs significantly faster; initial testing suggests run times about 0.64 faster and perhaps even better than that in the final version. Testing is ongoing.
8/27/2018	In anticipation of future simulation runs that will require more than the old position file directory function, I have now changed the directories to have to top level go up to 999. I intentionally programed this so that older auxiliary programs like Rplot.exe and Hist.exe will still work with the older structure but new structure will require changes in those codes eventually when and only if the highest directory goes over 99. I have tested this out beyond 100 and seems to work ok.
7/25/2018	I decided to change way SO sizes are determined. I now use CON(14) as the direct minimum size for a SO. The time delta is calculated from CON(14) directly if CON(10) is zero.
7/24/2018	Add method to create different types of EO mass density profiles is being finalized. I created a standalone program to test the methodology and is now incorporated into RIOD.EXE. With the new method, I can create 10 different spherical profiles including uniform, exponential forms, Plummer, NFW and other variations of these profiles. As long as there is an M(R) function, I can create the density profile. See Sections 3.2.4 and 3.2.3 for the details on these distributions and the impact on the SO initial velocities. ==To do 8/18/2018==

7/18/2018	There was some sort of logic bug that was not handling collisions correctly for ICN(18)= -3. I noticed that the Nearmiss.log file had and entry that was less than 1,0 for the closest approach distance but that event should have been a collision. I noted that for some reason, it was picking up the CON(29) value which was 0.85 instead of being set to 1. I have redone the logic handling of CON(29) now to set each ICN(18) case separately. See the details of CON(29) for the logic.
7/7/2018	I added some additional output to the Hits.log and Nearmiss.log files. Now when there is an event to these logs, I have calculated the interacting particle's relative velocity and divided it by the escape velocity for that position, assuming total system mass. I have out that ratio as the last entry on the line written to each of the above files. See Sections 3.3.6 and 3.3.7 for these files description.
6/27/2018	I noticed that the performance was degraded from the changes made on the 20th so I revisited what I did and noticed there was some things that could be cleaned up. Those new changes are included now and the performance was restored based on limited testing.
6/23/2018	I added some information to the Lost.log, specifically the total potential energy and kinetic energy is added to end of each written line in the log. I want to more easily track energy changes as the simulation progresses. I also rearranged some of the code order to facilitate the above change. I have reviewed the overall Lost.log output and found some of the information as unneeded, so I removed some things and changed the formatting of others. See Section 3.3.8 how the output now appears.
6/20/2018	I modified the repulsive character for elastic collision scenarios. Now the repulsive power can be anything between -2 and -20. This value can be set using CON(17). See Section4.3.2.4 for more details. In addition to this change, I changed the way the check distance is determined. Before, I was just setting it to a specific value but now I calculate that value based on the modified force used. See Section 5.3 for more details.
6/14/2018	I found an error in the Stats.exe program that lead to finding a fumble in the Riod.exe program. I noticed after some time running in the vanishing collision mode that the virial radius predicted by the total energy in the Stats.exe program was wrong. Stats.exe was using the total mass stored in the Con array, which was not being changed for VC. I fixed Stats.exe by summing the masses while determining Total Kinetic and Potential energies. Then I changed Riod.exe to subtract the colliding masses from the total mass array member, CON(55). This won't change to the correct value for running simulations but State.exe will make the correct calculation.
3/31/2018	I made a small change today to fix the formatting of the "Hits.log" file. I changed the integer format for the number of iterations and iterations since last collision to I12.
3/12/2018	I changed the output options and some operational methods in Stats.exe. Now if there are arguments on the command line, the usual output to the screen is suppressed. Also there is the file number for looped operations as part of the output file.
3/10/2018	I changed the usage of ICN(3) to now be the number of minutes the simulation runs between saving the Riod2.Dat File. The old method was cumbersome and this is much cleaner.
3/10/2018	The improved method for speeds in the EO in the previous entry exposed something that I deem undesirable. That is that the symmetry of the object pairs remains so incredibly tight that one test case shows that the symmetry is retained for much longer than expected or desired. Because of this, I have introduced a dither on the mirrored object distance out. The percent of the dither is controlled by CON(9) and will dither the mirrored object radius by $\pm$ dither percentage. This should break the symmetry. Note that now with these changes, a addition CM correction is done of the positions before the speeds for the SO are determined.
3/1/2018	I improved the way initial speeds for objects are determined inside an EO. See Section 3.2.4 for the details of the current method.
3/1/2018	I added a new collision type today which is essentially a vanishing collision, VC. This event happens when ICN(18) is set to -3. Read about this in Section 4.3.2.3.
2/15/2018	Added a feature to Stats.exe to now had command line options to create a single line output file with the energy parameters. If there are 3 command line arguments, when this file is created.

12/21/2017	I found a bug in the output to the Hits.log file for replacement collisions. The distance out reported in the third from the end of the output line was somehow getting mixed up and would report the same number over and over. I am not sure why this was but I changed the name of the variable to "distout" and that seems to have cleared the mix up. I also cleaned up the number as it was just the distance out for the "i" object. Now I take the average of i and j object's distance and report it.
7/7/2017	I found conditions where events were not being written to the Nearmiss.log file. After thinking about the method for how this is determined, I realized that I could improve the estimate of the time and distance of closest approach. I already had embedded algorithm but I realized that since the code does these estimates between the calculation of the velocity changes and the actual position and velocity updates, I had the information available for a better the estimate all along. How this is now done, I estimate the time and distance of closest approach for the position and velocity of the previous iteration's values. I do it again with the current iteration's data after creating it by updating the position and velocity of the last iteration with new velocity changes calculated for this iteration. I compare the time to collision for those two estimates and when the time goes from positive to negative, indicating that the closest approach is in the past. When that transition happens, I set a flag in ICN(36) to indicate it is time to write to the Nearmiss.log file.
6/9/2017	I changed the formatting for the Hits.log file and all occurrences of the Iter8 variable so that the simulation iteration number can now allow 11 digits. I then changed the format of the normal output to an E7.1 for the largest object sizes to accommodate the normal window size during execution. Dumb oversite but corrected now. Actually, I don't think I have ever run a simulation for so many iterations that I needed more than an I9 format.
2/15/2017	I made some quick changes today to test an idea that just didn't work as expected. The change I made was to now have a new input CON(16) which is now the size scaling for the central mass given in CON(13). Using CON(16) as a large number, creates an M(R) like calculation for objects inside the value of CON(16). This option must be used with the pass-through condition, ICN(18)=0.
	This change was made quickly and some of the initial condition ramifications were not completely flushed out and rectified with the other scenarios possible. Also there seems to be a memory allocation issue if the number of objects is more than 500. When I have time, I will investigate these issues since I have been planning to add modes where M(R) calculations are done on object pairs.
	Note that this is the first initial step in creating halo like mass objects for the simulation. Currently, the code using only the halo function where:
	$M(R)=M_T(1-exp(-(r/S)^3)$ which is already build into the code.
2/8/2017	I added a new option to the change parameters subroutine. Now one can change the rate at which the simulation data is saved. This is ICN(3) and it now is changeable using the run time options of "Riod 2". See Section 2.4 for more details.
2/3/2017	I still note that the replacement collisions have a net system velocity change that is not desirable for a long-term sustained simulation run. I have included code to correct the total system momentum to be zero after the replacement objects position and velocity vectors are created.
2/2/2017	I added a dither function that takes a percent as input and then will create a random number that is in the range of 1±input/2. So a 6 percent dither would create a random number between 0.93 and 1.03.
2/1/2017	While testing the changes below, I revisited the velocity component of the replaced objects. I realized the method below, each of the I and J objects involved would have a different radial distance from the system coordinates, so I calculate the average speed for both based on their different distances. The dither is changed to 10% now. See Sections 4.3.2.2 and 3.3.6.2 for more discussion on replacement collisions and the logs.
1/30/2017	I have changed the way replacement collisions create new positions for the collided objects. I was unhappy that my previous method would over time cause the entire collection of objects to drift from the original center of mass.

	The new method that I am testing is to take the objects at the collision point and go into their COM and extent their COM positions out to the a multiple of the initial virial radius contained in CON(51) and CON(31) the scaling distance. There still is a dither of 1/8 for these distances.
	I have also have changed how the velocity component of the replaced objects is determined. I have found the cross product of the vectors that point between the I and J objects and their new radius vector. This creates a vector perpendicular to the radius vector which I then make a unit vector and then calculate the vector with a speed calculation.
11/25/2016	I finished the movement of the real constant array CON values around. I Put the big three together, so now time distance and mass are elements 10, 11 and 12, respectively. Many other elements were moved too to try to put input parameters together that control similar aspects. I am still looking to configure these inputs better. In addition to moving things around, I moved internally calculated items to the end of the CON array where they do not need to be an input selection in Riod2.ini.
	I also put an addition to the creation.txt output where all the CON and ICN array elements are written to the log after all the creation steps are complete, so there is a record of how each simulation began. There are other changes that have happened in the last week that I can't completely remember.
11/19/2016	I started remaking the Riod.ini file today and is now called Riod2.ini. The system Dat file also is remade and called Riod2.dat. These changes were precipitated by a desire to simplify/reduce the required input fields in Riod.ini. In addition, for upcoming new options, I needed to free up some of the input for control and scaling parameters. See the to-do list above. I began daunting exercise by expanding the total number of constants as follows:
	The ICN array goes to a total of 50 elements of which only 30 are actual inputs from Riod.ini. The CON array goes to 60 total elements but for now keeping the current 45 elements read from Riod2.ini. I completely shuffled the ICN array members to move internally calculated members to the highest array elements and then moved the remaining element that were outside of the 30 count into the vacated elements that were moved out to the higher elements. I will do similar things to the CON array in time but must scope that our more since there are dependencies that I will undoubtedly mess up. I made changes in Rplot.exe and Stats.exe to cover these changes. There probably changes needed for Hist.exe but not looked that yet.
11/18/2016	Made some changes in the Riod.ini file and how output is sent to the Creation.txt. I wanted make the creation.txt file more user friendly for use in spreadsheets (see changes on 11/1/2016). The changes previously didn't really satisfy me, so I created a string array to read in the Riod.ini file descriptive text so that that text could be output to the Creation.txt file. In do this, I also changed the format of the Riod.ini file to exclude the second comment line. Now the Riod.ini file has the format of a comment text string of which 132 characters are read in for output. Then the next line is the number be read in. Section 5.4 shows this new format.
11/5/2016	I made some significant changes to control the simulation output. ICN(13) now controls the output files and logs. Values can range from 0 (no files or logging) to 4 which is full logging. See Section 3.3.3 for all the details. Note, that I was not able to test all the control options yet.As the result of these changes, ICN(23) is no longer used.
	I also added a parameter control option for ICN(13) when invoking the command line option to change various control parameters after the start of the simulation. Section 2.4 discusses this.
11/4/2016	Planning to improve the logging options and discovered that some of the logs used the same unit numbers to read and right from. In order eliminate possible confusion, I went through and created unique unit numbers for all the logging. The unit numbers are now tracked in a table in Section 2.2.
11/1/2016	I made a small change in Riod and output to the Creation.txt file. I inserted a comma immediately after the ICN and CON array numbers are output. In this way, I intend to use the RIOD.INI values from this output in spreadsheets and I can use the comma separation to have clean delineation between the number and the following text. I have not tested this change yet and I will have to change the current spreadsheet template to accommodate this new format for initialization data.

10/5/2016	I made a small change in Riod today as I added some output to the Creation.txt log to monitor the change in object size as dictated by CON(15).
9/29/2016	Riod changes began with attempts at creating a better replacement collisions mode but that failed completely, well the implemented solution was not particularly satisfying or physically realistic. While messing with that, I rethought about the elastic collision option and tried a solution with an increased object size to see if that would help the energy conservation issue I was seeing. Initial trial shows that perhaps there is some benefit to doing this. I also noted that CON(15) was really unused, so now there is a new option to use CON(15) to upscale the object size. Now if CON(15) is greater than 1, the initial sizes of the objects are multiplied by CON(15). Note I do not allow down scaling!
9/19/2016	I made several changes in some of my auxiliary programs. First a change in Stats.exe to force a space between the output of the number of iterations. I also forced that output to have a I10.10 format so that the user will know how big that number could be. The new output can be observed in Section **Error! R eference source not found.**.
	I had to update Hist.exe (which isn't discussed within this document) as it was not using the extended iteration number.
9/5/2016	I changed the way the eccentricity of the new replacement collisions objects is calculated. I picked the minimum orbital distance to be scaled between 1 and 0.75 times the currently calculated virial radius. See section 4.3.2.2 for more details.
7/22/2016	I made a change to Stats.exe today to test if an object is bound gravitationally to its nearest neighbor. I calculate the total energy of the pair in its center-of-mass and if that quantity is negative, odds are that the pair is currently bound. If the negative total energy for the pair is negative I write the distance of separation as negative to indicated the bound pair.
5/19/2016	I made a small change today in the code to correct an output to the "Create.txt" file. The format statement that prints out the text for the velocity values as they are created was shifted because the text was right justified for some reason. Anyway, I just made the string match the size of the text.
5/5/2016 To do	I have taken one of the "To Do" items off the list as I have added a few new sections to help users pick values for the simulation. I have also created a spreadsheet to help with picking suitable scaling values for mass, distance and time. This is all described beginning in Section 5.2. Note that the discussion of picking initial conditions and the spreadsheet are considered as an evolving topic for this document. <mark>Write section in this document for describing tips for setting up initial conditions for certain simulations.</mark>
2/12/2016	I have filled out the RCN array and it now has 6 elements. See Section 5.3 for the current details.The sixth element is a new control for determining the softening radius. I used to have a hard coded value that scaled the distance at which two approaching objects would have their forces softened as per the figure in Section 4.3.2.8. Now this scaling factor is different depending on the type of near distance force modification. See the RCN array section noted above for more details.
2/11/2016	I began this transition today by creating a run-time constants array called RCN. The first entry is for holding the value of Pi. I will be adding more soon but there is much baggage to this operation. The RCN array currently has 10 elements and that should be enough for now. To do: Investigate creating local constants array that is initialized on start up. I have identified candidates from the CON array for that in Section 0.
1/26/2016	Made some small change today. I changed the way CON(28) is determined. Now for pass through collisions, CON(28) is set equal to CON(29). For other cases, CON(28)=2*con(29). The other change was that I changed what was output to RiodOut.log. Now only output to RiodOut.log only when there are objects being tracked as close for collision types.
1/14/2016	I have known there is a memory leak in the current code for months. I have not used any tools to figure out what is causing it but I have always guessed that it was when I introduced a common block into the code back on 10/3/2015. Since the common block was only a modest change, I have removed it and

	added one or two new arguments to the call list where needed. It will need to be extensively tested to be sure this cured the memory leak.
	I also changed some output when the code writes a new position file, the file is preceded by the iteration number and the value of the iteration rollover constant ICN(19).
	I modified STATS.EXE program to now use an E8.1 format to print out the mass in the main output stream. This is to accommodate larger mass variants in the future.
1/10/2016	While updating the value of G yesterday, I discovered a flaw in the code that used CON(3) in the RSU calculation of G. The code should have been and now only uses CON(11), CON(10) and CON(12) to determine the RSU value for G. Actually, CON(3) serves no purpose in this code and I should probably remove that from options in RIOD.INI.
1/9/2016	I discovered that there is an updated value to the Universal Gravitational Constant in Wikipedia. The value of G= 6.67408×10^{-11} N m^2 kg^{-2} is now being used. Previously I was using G=6.67384×10^{-11} N m^2 kg^{-2}. The MKS unit's value of G is hard coded into the simulation but of course the RSU value of G is computed internally after the unit's conversion.
11/29/2015	I made two small changes today to make the integer output to the Creation.txt file allow for two more digits for the number of failed position attempts for the SO and EO positions. This is now an I6 format.
11/18/2015	I noted a problem for initial velocities within an SO group when con(6)=con(5). I checked the code and there was a blatant error in the formulation of the M(r) estimate. I also did some testing and saw that when con(6)=con(5), the equations in Section 3.2.4 didn't have a correct result for M(r). See the new text in that section that described how this case in now handled.
11/17/2015	I changed the name of the creation log file to "Creation.txt". I did this simply because it will open cleaner in notepad than having the name with the log extension.
10/18/2015	I made some cosmetic changes to the output of the "Creation.txt" file. I cleaned up and formatted some of the unformatted output. This is an ongoing thing as I finally discovered how to do inline formatting and am putting that to use in the log.
10/4/2015	I made a small change today to allow printing of the number of iterations since the last collision of the case of the replacement collisions (ICN(18)= -2). Now for this case, the number of iterations since the last collision printed if there are no close objects during that iteration.
10/3/2015	I have made a couple of changes in the last few days. First, I modified the code so that there is no force softening for the case of replacement collisions (ICN(18)= -2). This change is a trial period as I test it out for the latest study I am planning.
	The biggest change is that I have included code now to get by the iteration number going beyond the 2^{31}-1 limit. I am now using ICN(19) to hold the incrementing multiplier for numbers over a set maximum. I will set this for 10^9 after I am testing it. This change also required significant changes be made to Rplot.exe and a few changes to Stats.exe. I have made those modifications and tested those programs but not as thoroughly as I would have liked. I also made a small change in the force softening distance. I now start the softening at 2.5*object's size. It was set as 3 times before but looking at the graph in Section 4.3.2.8, that distance was probably too large.
8/13/2015	After the previous change, I figured that the ICN(180= -2 case needed better output in the Hits.log file, so now there are two outputs for collision cases. The current output may change later but for now, I output the total energy, the virial radius before the replacement, the magnitude of the replacement objects velocity and the distance out from the origin of the collision. For the latest, see Section 4.3.2.2.
8/11/2015	I created a new collision scenario today for what I am calling replacement collisions. Replacement collisions are when two objects satisfy the collision criteria, the two objects are removed from the simulation and replaced with new objects farther out from the origin. Energy conservation and zero total momentum are maintained in the course of the replacement. This new collision scenario is enabled when ICN(18)= -2. The distance out parameter is scaled with the CON(31) times the Virial radius. See Section 4.3.2.2.While developing and testing this new feature, I modified the Hits.log actions, now opening the file sooner in the Collck routine to accommodate debugging I/O. The output itself is

	unchanged but now I also changed the file open status as append and no longer scan through the file to the end but just append to the file. Thus, duplicate entries are possible.
7/26/2015	I found a small bug with how the code determines the maximum velocity change array index. The code was using the old definition for extended objects and so now it conforms to what is written in Section 4.5. Also I modified the output to the console log to simply output ICN(49), which is the current maximum separation distance in iterations between the farthest apart SO.
7/20/2015	I made a change in "Stats.exe" that now calculates and prints out the virial radius and relaxation times as defined by the Stahler paper. These parameters will not be relevant for all simulation types but are a nice addition for the simulations I have been running recently. See Section 6.1.3 for more information.
7/19/2015	I made cosmetic changes to the "Lost.log" and the "Nearmiss.log" output. • Lost.log: I added a year field to the output file right after the number of iterations. Nearmiss.log: I appended the distance to the first object to the end of the output string. I also truncated two other fields to smaller formats.
7/15/2015	I made subtle change in the way the code operates between iterations. Before, the collision check subroutine (collck) was only called of the collision flag ICN(18)= -1 was set. I became interested in being able to monitor close encounters for all types of close interactions. Thus I allowed entry into Collck whenever there was a pair of objects close, as defined by CON(29). Now what happens is that when there are close pairs of objects, Collck is called and first determines conditions needed for object pairs to collide and at the same time determines if there was a nearmiss. If things are close and the nearmiss.log file conditions are met, then a nearmiss.log file entry is written. Then if there is a collision during the current iteration and depending on ICN(18) will do the collision typical rearrangements. This change for this log file precipitated changes to the other logs files. Read about those changes in Section 3.3.
7/11/2015	I changed the logic flags for how force models are used and how collisions are used. This is all controlled by ICN(18). This was changed to facilitate future collision modes. The new way things are mapped are as follows: • ICN(18)= -1, is the inelastic collision mode, this mode has always used force softening up to the collision distance. • ICN(18)= 0, is the pass through mode, this modes has always used force softening. • ICN(18)=1, is the elastic collision mode which uses the fore model in Section 4.3.2.4
7/3/2015	I made some minor changes to the output to the "RiodOut.txt" file which made more specific that the boost position tries were failing and how many failed during the creation phase. I also upped the number of allowed failed attempts to 100 from the original 20.
7/1/2015	A test case crashed today and looks like there was a logic error in the code that creates the "RiodOut.txt" file. I am not sure I fixed what caused the crash but what was missing was the check on when to close the output device, so I added that. I also cleaned up the code around the output statements at the end of each iteration to make it more viewable.
6/29/2015	Made a small change today in the way the code checks if SO are too close during the creation phase. I wasn't checking the first object for closeness.
6/22/2015	I cleaned up some of the console output that is duplicated to the "Creation.txt" so that now that output only goes to the file and not to the console. It is much more convenient to view the log than to revisit what happened on the console screen. I also cleaned up the comment below about the ICN(13) benchmark flag. Now only the "Creation.txt" file is created when the benchmark flag is set.
6/19/2015	Added new behaviors for the "status.log" file, see Section 3.3.5 for details. I also created a console output file called "RiodOut.txt" to mirror the console output. See Section 3.3.10 for details on this. I need to fix this and add the option to control this will ICN(13), the benchmark flag.

6/16/0215	About this time, I found a small bug in "Stats.exe" and the way it calculated the total energies. I cleaned this up and made the calculations cleaner to read. The result fixed the total potential energy calculation. Comparing the old and new yielded only a small difference in the output.
6/14/2015	A new feature created today is a log to monitor what happens during initial condition creation. I now output to a "creation.txt" file events that I have happened to include output for in the log. I deemed this useful while debugging the previous changes so that I could look at what happened while the code is running. It is also useful in experimenting with the spacing of SO within the EO. One can change CON(33) to its largest value to force to most uniform distribution of SO in the confines of the EO shape. The Creation log will tell the user how many failed attempts at meeting the CON(33) requirement there has been.
6/12/2015	Today I began a major overhaul of the initial condition's creation subroutine, appropriately called "Create". I made so many changes that it no longer resembles the old capabilities. In essence, any number of extended objects can be created up to half the total number of particles (ICN(10)). Each EO can have a large mass at the center. All EO are created identically, so the particles are created in pairs with opposite radii and velocity vectors. I am still debugging and testing three days later as there are so many options and variations that I suspect some options will go untested and thus perhaps may not work as expected. Resulting from these changes is an increase (from 35 to 45) in the number of elements in the floating-point constant array con. Changes had to be made in Rplot.exe and Stats.exe to accommodate these changes. Reading about all these changes will be important and the document should eventually include a discussion of everything that is new or changed.
3/23/2015	Modified the Stats.exe program. The biggest change is that after determining the center of mass coordinates, I now have it find the COM coordinates of only the objects within the virial radius and use that as the offset/origin in determining all the position distances. I am hoping this will give me a more realistic mass distribution, especially long after objects are ejected from the system.

I made one cosmetic change and that was to put a space after the equal sign when printing the iteration number. This helps when pasting data into spreadsheets when the iteration number gets really large. |
3/7/2015	Searching for issue with the main loop in the Isaac subroutine, I found a small error with which time iteration was being used. ICN(15) is the maximum index for the velocity change array and I was truncating it one iteration too soon. This would not change the calculations much since all it would do in effect is to decrease the time depth ICN(15) by 1. I also found one variable that was not in the MP shared list and added it in.
3/5/2015	I modified the code to accommodate up to 12 processors/threads. I changed the part of the code that one can use to modify the ICN(21) thread count to allow up to 12 but not more if there are more than 12 processor/threads available to be used.
2/16/2015	I made some small changes in some type conversion functions, like dabs() for abs().
2/11/2015	I modified Riod to removal all the temporal delay code. I am testing this code to see if it is more efficient and if there are any anomalies in the original code that will be solved by removing the time delayed forces. This new code requires testing and I haven't decided on the best way to do that. More on this later.
4/20/2014	Several changes made to the code today. ICN(48) is used to track the number of object pairs that are have their force softened because they are too close. There is a new option for printing out to the Status,log file where the maximum number of force soften pairs are output. I added ICN(2) to the list of parameters that can be changed using the command line option "2". ICN(2) can be 1 to 100 and controls how many times Stats.exe is called. ICN(2)=1 means every time a new folder is created, Stats.exe is executed and file is created in the "pos" folder. ICN(2)=2, means every other time a folder is created Stats.exe is run.
7/6/2013	Added option now to calculate the simulation time interval based on EO size. CON(14) is used to scale the EO size to the object size. See Section 4.3 for more details.
7/6/2013	Made cosmetic change to the code and changed ICON array name to ICN. This allows unambiguous searches in the code for ICN or CON variables. I have changed all references in this document to match.

5/26/2013	Modified the stats.exe program output to include the number of iterations executed and the totally number of years into the simulation, based on the number of iterations. I also changed the output to be more Excel friendly by using the letter "E" in the scientific notation output rather than "D".
5/19/2013	Added new feature for when collisions are allowed. If a smaller object is completely inside a larger one, use that as an additional collision criterion. Now if the distance between objects radii is < the sum of their sizes times CON(29) or if the separation is less than the absolute value of the size differences, then a collision happens. See Section 4.3.2.1 for more details.
5/19/2013	Fixed the calculation on the velocity change arrays when ICN(15) set to -1. Now I fold in the extended object separation distance when needed into the maximum number of array elements.
5/18/2013	Added a feature to automatically execute the stats.exe program and output to a text file when a new directory is created. This text file has the following naming format: stats_xxyy.txt. Where xx is the top directory and yy is the intermediate directory for all the position files. See the description of ICN(2), which controls the behavior of this feature.
2/24/2013	I have changed the object pass through method. This is used with and without the inelastic collision option for ICN(18)=0 or 1
2/24/2013	I have now implemented an option to do elastic collisions. The method for doing this is discussed in Section 4.3.2.4. ICN(18)=2 turns on this option.
2/2/2013	Changed the way the code checks to see if it is time to save a position file. I now use the MOD function, which is much more astatically pleasing. Now ICN(2) is unused.
1/29/2013	Added the use of CON(33) to define the minimum distance allowed between objects during the set up of the initial conditions. See Section 0 for more details.
1/27/2013	Added dialog to change number of processors, ICN(21) and the OpenMP chunk size ICN(22) from command line. See Section 2.4 for more details.
1/27/2013	Rewrote code for changing parameter with the command line option "2". Pretty much works the same as before just cleaned up some of the spaghetti. Will also make it cleaner to add other change options.
1/26/2013	Simulation was having weird problems with the random number generator that I have been using for years. The function was RAN2 out of the "Numerical Recipes" book and for some reason it was doing strange things. I ended up just using the GNU RAND function as my random number generator.
1/19/2013	Added a new provisioning versatility. ICN(27) now is used to create better spherically symmetric EO distributions. See section 5.1 for an explanation.
1/16/2013	Began adding OpenMP extensions and paralleling the code for multi-threaded running. This required creating temporary arrays for holding the velocity change arrays. This added a significant presence to the memory usage. I also had to add a compiler option to use more heap space before the code would run. I continue to test the compiler options and various scenarios with input parameters. New parameters are added to Riod.ini to help control the use of processors. ICN(21) is used to set the number of processors used. See Section 5.1 for these changes.
1/3/2013	I changed the output line to the screen. The changes are mostly cosmetic, see section 3.3.4.
1/3/2013	I begin using GNU FORTRAN about now. I also found the "Simply FORTRAN" development environment about a week later.
6/24/2012	I made a change to the Animate.exe program to be sure the program uses the coordinates in the position files based on the starting file number that the user enters. This will begin plotting properly now when using a negative entry for the first object coordinates.
6/2/2012	I added a feature that now determines the maximum number of close objects between events written to the "Status.log" file. The maximum number of close objects is now output to the "Status.log" file. See Section 3.3.5.

4/22/2012	I rewrote the logic for determining initial velocities. There were some flaws that were exposed in the other recent changes. Now the flow is better and accounts for the three different classes of problems better.
4/22/2012	Made a small change in the way the collision check distance is used. Now, CON(28) stores the check distance which is determined internally depending on the type of simulation scenario. For collisions the check distance is 2*CON(29) and for no collisions, it is CON(29)/2. I was doing this calculation each time I would enter the main calculation subroutine, which was really dumb.
4/21/2012	Fixed issue with creating a large central object. This was a problem for a single EO or 2 EO. I was not excluding the first object, which is the large mass object, when creating the velocities of the other objects. The large central object is given zero velocity in its EO. Later it is given a velocity kick for its EO and of course all the small objects in the EO get the same velocity kick. Each EO has a COM of zero.
4/16/2012	I added another input to the "Animate.exe" program to allow the user to start viewing files later in the simulation evolution.
6/28/2009	Minor change in the calculation of lost objects. Probably will not affect the actual determination of when an object is lost but is cleaner coding. The initialization of the comparison distance for other close objects was starting with an undetermined value.
9/10/2007	Changed the output of the "Hits.log", "Lost.log" and "Nearmiss.log" files to have a nine-digit integer for the number of iterations.
6/7/2007	Changed the output to the screen to show that when two objects are being tracked for a collision, the program prints out the predicted closest approach of the two objects as a multiple of their two radii. The program used to print out the actual distance apart but I deemed that this number was not too useful since the collision distance is based on the fraction of the sum of the two radii. Now the program prints out the actual distance scaled to the collision radii and the predicted distance scaled to the collision radii.
5/24/2007	I changed the criteria for determining how an object is lost. Before an object was determined to be lost if the velocity was greater than the escape velocity and its distance out from the COM was greater than the maximum distance set from the parameters CON(6), CON(19) and CON(31) and if the closest other object was greater than 0.4 of the maximum distance. The 0.4 was a hard coded fraction of the maximum distance from the COM. Now the closest object distance is now just twice the original size CON(6).
	Bug fix for the "Pos" directory structure. The code was trying to use the number of files in each directory value ICN(11) as the directory count too. Unfortunately, the directory creation is limited to 2 characters and thus the maximum for these is 100. Internally, the code forces the maximum directories to be in the range 0-99. This also fixed in "Animate.exe"
5/14/2007	I fixed one issue that was occurred when a new simulation starts, the program tries to copy the "Riod2.dat" file to the "pos" directory structure. When initializing the simulation, the "Riod2.dat" file hadn't been created yet, so the copy would fail. I changed the order of operations to ensure that the file copy happens after the file is created.I also added a series of commands to copy the "Riod.ini" file at the start of the simulation to the "pos" directory. This ensures that an original copy of this file is preserved after the start of the simulation.
	The simulation now uses CON(59) to store the total number of real-time run seconds during the entire simulation run. This number is output in units of hours to the screen and the "Status.log" file
5/12/2007	Changed the way object pairs are tracked for being too close. Before, all objects were tracked if they came within 2 times CON(29). Now if ICN(18) is set for no collisions, then the check size is 0.5 times CON(29). This change was necessary because simulations without collisions can have many pairs of close objects and the arrays that track those pairs are limited to 500 elements.
4/29/2007	Changed the method for determining which objects are close enough to be tracked for possible collisions. Before it was if objects came within twice their combined size, then they would be tracked.

	Now it is if they are within twice their collision distance, then they are tracked. Note that the new check distance is just a multiple of the combined size, CON(29) and two.
4/24/2007	Changed the internal calculation of the scaling constants. This resulted from the potential problems of manipulating these constants when scaling the simulation to very large distances and masses. Once the simulation is in RSU, then there is no problem but only scaling to the RSU is where these numbers can get unwieldy.
4/22/2007	Started adding general relativistic discussion in the main text. This is discussed in Section 4.2.Added another output number to the "Nearmiss.log" file. A ratio of the combined object distance to the collision distance is now output.
4/16/2007	Added another output number to the "Status.log" file. The number of close objects has been added after the number of lost objects.
4/15/2007	Fixed the general relativistic calculation main loop to include the correction when the two close objects are within their combined radii. The correction is exactly as done in the non-relativistic case. I am not sure this is exactly correct but it should default to the Newtonian case for low velocities. So I guess one can cross the fingers and hope this is a good guess.Fixed a sign error in the general relativistic calculation. I hope to include a discussion on the relativistic equations I am using soon.Changed the output of the "Stats.exe" program to output the ratio v/c instead of an absolute velocity.
4/8/2007	Fixed output to "Hits.log" file to show 4 digits for the object number.
4/7/2007	Fixed bug for simulations that use expanding simulation options, when ICN(16)=1. Added code to correctly determine the largest and next to largest objects in the case of no central mass. This would correct itself after a few collisions would happen in previous versions. Fixed "Stats.exe" output to show 4 digits for the object number.
2/24/2007	Added new method for creating extended object mass distributions. Now the user can create an inverted wedge as opposed the expanding wedge of objects. See Section 3.2.3 for details.Added internal calculation of the maximum indices needed for the simulation as set by the maximum size of the system divided by the speed of light CON(6)*CON(19)*CON(31)/CON(12). If this number is greater than the maximum allowed it is set to the maximum of 1000. Set ICN(15) to –1 to force an internal calculation. See Section 4.5 for more details.
2/7/2007	Added new command line options. See Section 2.4 for details.
2/4/2007	Changed the method for determining the initial object velocities using an interior mass calculation depending on the distance from the COM. See Section 3.2.3 for details.
11/29/2003	Changed/fixed the method for determining the size of objects based on the period of orbiting objects at a distance equal to their radii sum. See Section 4.3.1 for details.
4/10/1994	First changes for the object size and tests for stability when objects get too close.
2/13/1994	Added the feature that allows objects that get too close to pass through each other or at least when they "enter" within their combined radial distances, the force felt is modified to be a constant. See Section 0 for more details.
12/18/1993	It looks like the ability to have initial conditions such that all objects have initial velocities radially outward happens about this time. See Section 3.2.3 for initial conditions discussion.
9/25/1993	General relativistic calculation option is added at this time. From what I can tell, the issue with using this option is that other object velocities used in the calculation must be in the rest frame of the object of interest. I took the short cut of using a non-relativistic transformation to get the rest frame velocities. As a result, this calculation is not correct in a general relativity sense but will probably be okay for most cases since velocities are rarely relativistic. There is no discussion of the method used for these calculations used yet. Stay tuned.
9/29/1992	Lost objects are removed from the simulation at this time.

	Changes in the output lines to show close objects being tracked is added.
	Added that the "Riod2.dat" file should be written/saved every ICN(30) screen writes.
11/2/1991	First attempt at creating an object size based on orbital speeds of close objects. This has been modified several times since then.
9/22/1991	Code now uses the orbital eccentricity as "real" parameter in determining orbital velocities for orbiting objects. Normally eccentricities, e are given as positive numbers but as it turns out if we use negative numbers, the orbit begins at the orbital perigee (maximum orbital distance) and positive numbers will have orbits beginning at the orbital apogee (minimum orbital distance). See Section 3.2.2.3 for more details.
9/22/1990	First logbook entries show the beginnings of trying to lay some framework around determining how to size objects and relate that to the time deltas and calculation stability.

8 Windows Related Operations

This section will discuss things the operator needs to understand about the interactions of MS Windows and the programs discussed above. The discussion below is assuming a Windows XP variant but will behave similarly in all previous Windows versions.

8.1 Command Prompts

First, a command prompt (mention of a console session above means the same thing) is just a command window where you can run commands like in the old DOS and Unix days.

To open a command prompt, go to your Windows start tab and scroll down to the "Run" option and select it.

In the window that opens, enter on the dialog box "cmd.exe" (no quotes of course) and then click the "Ok" button.

A command window should open and be ready for your commands, looking something like the following

To use Riod, you need to change directories to the directory you have Riod in. If Riod is on the same drive as what is shown in the command prompt then just type "cd \riod" or whatever the directory is. If it is on a different drive, say drive "E:" then issue the commands as shown below.
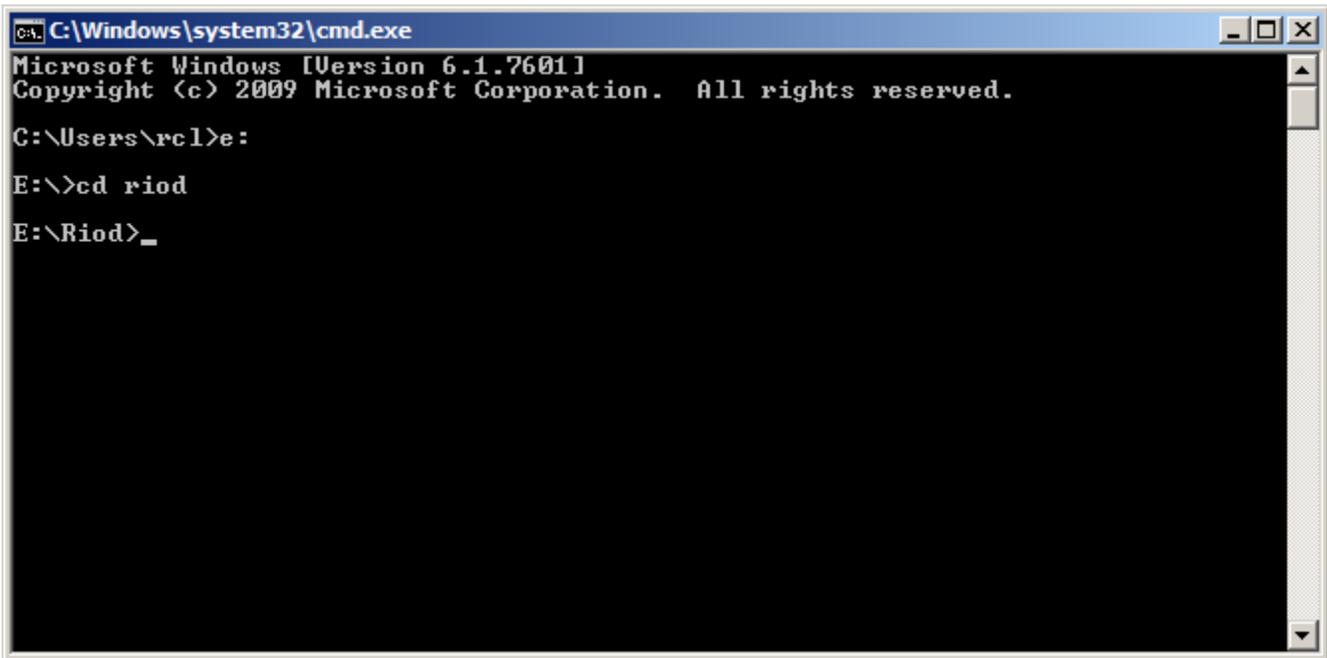
Riod can be run within the command prompt as above and when stopped, the window will remain open. If Riod is run by double clicking the executable, that window will close when Riod is stopped.

Also, it is best to run "Stats.exe" in a command window and if you use the old DOS and Unix "Pipe to more" option like "Stats.exe |more", then you can see the output, one page of information at a time.

8.2 Batch Files

I use batch files to augment the usage of the "Riod.exe" application.

8.2.1 "New.bat" – Begin a new Simulation

As mentioned in section **Error! Reference source not found.**, "New.bat" is a batch file that can be used to start a new s imulation. Simply typing "New" in a command prompt window will instantly start a new simulation run. This batch process will delete current status files and delete the "Pos" directory that holds the position files from the previous simulation run. Note too that this batch file is assumed to be in the same directory structure as all the other programs used in the simulation.

The contents of the "New.bat" file are shown below:
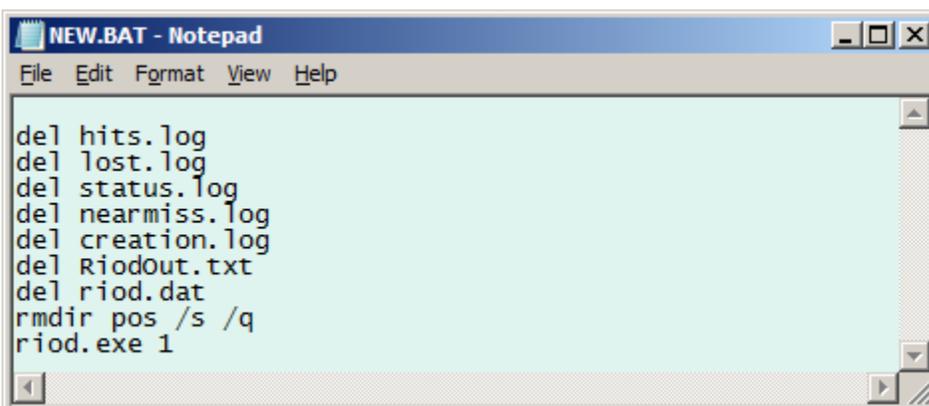
8.2.2 "RiodStart.bat" – Restart/Continue Running a Simulation

I use another simple batch program to start "Riod.exe" as a low CPU priority task in Windows so that the computer will be less sluggish during times when it is being used[26]. This batch program uses the Windows "Start" command line option to start a task. There are many options of the "Start" command and the user can investigate those by typing at the command line "Start /?".

For the purposes of the simulation, I use this batch file in conjunction with a desktop shortcut to start the simulation. Create a shortcut on the desktop that points to the "Riod.bat" file and then each time the simulation is started, it will run in its own window and with a low system CPU priority.

The contents of the "Riod.bat" file are shown below (note again this must be run in the directory where all the simulation files are located):
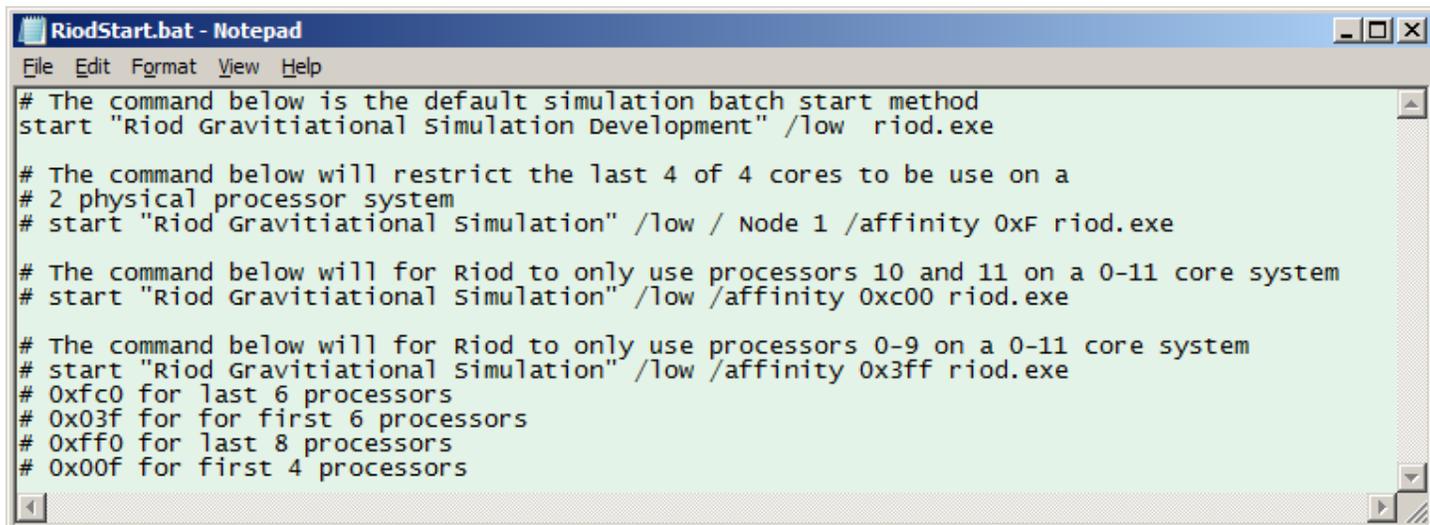
```
# The command below is the default simulation batch start method
start "Riod Gravitiational Simulation Development" /low  riod.exe

# The command below will restrict the last 4 of 4 cores to be use on a
# 2 physical processor system
# start "Riod Gravitiational Simulation" /low / Node 1 /affinity 0xF riod.exe

# The command below will for Riod to only use processors 10 and 11 on a 0-11 core system
# start "Riod Gravitiational Simulation" /low /affinity 0xc00 riod.exe

# The command below will for Riod to only use processors 0-9 on a 0-11 core system
# start "Riod Gravitiational Simulation" /low /affinity 0x3ff riod.exe
# 0xfc0 for last 6 processors
# 0x03f for for first 6 processors
# 0xff0 for last 8 processors
# 0x00f for first 4 processors
```

n a Windows 7 system you can use the "Start" command options to limit execution to a single physical processor and to include exclude specific cores on that physical processor. .

There are three examples in the above file for using the Affinity mask and the Node selection. The first uses 4 of 4 cores on the processor number 2 (Node 1 of Nodes 0-1). The other two examples show how to set the Affinity Mask for physical processors that have 12 core/threads. Read up on the start command for more details on how to best use it.

8.2.3 "Stop.bat" – Stop a Currently Running Simulation.

A new batch file is now included in the Riod stable of auxiliary files. The file Stop.bat has the only function to delete the "stop.riod" file. The simulation checks the existence of this file and as long as it exists, the simulation runs. Stop.bat deletes the "stop.riod" file, stopping the simulation. This file looks just like this:
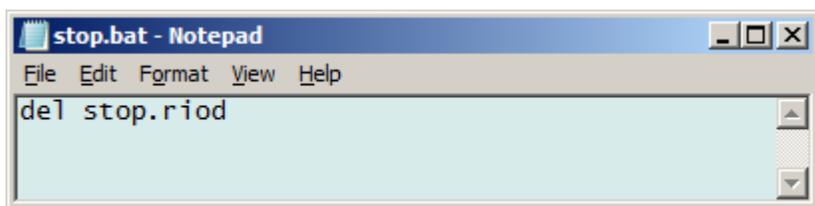
```
del stop.riod
```

One other was can be used to stop the simulation execution. One can simply do a CNTL-C in the command prompt and the program will terminate. Doing this will cause execution iterations to be lost back the last system save but usually one can set that up to be only a few minutes of lost execution. Restarting the simulation will begin again at the last save point and overwrite any redundant position files or log entries.

---

[26] Note that running with a low priority is not as much an issue these days where Windows systems have multiple CPU cores at their disposal. One can use this option anyway and if for some reason, your systems use their processor cores 100% of the time (like my systems do), then you are assured that "Riod" will not interfere too much with the operations of other running tasks.

9 Tested Initial Condition Scenarios

There are so many options/variations for initial conditions that I will use this section to chronical cases tested for this version of the code. Test case files are included in the simulation files. The files take on the name such that the filename ends with the Case ID. One can rename any of the test case files to "Riod3.ini" and run that scenario to start a simulation. For example, to run Case ID P06, find the filename "Riod_TS_P06.ini" can rename to "Riod3.ini" and then start a new simulation.

9.1 Planetary System Sized Cases

The following test cases were created to test various scenarios for the code. All of these test cases use initial conditions which are planetary system sized.

Case ID	EO	N	EO Shape	Big Mass	EO Rot.	Velocity	Notes
P01	1	500	Sphere	No	No	$0<\alpha<360$ Perp.	6/24/2015
P02	1	500	Sphere	10^4	No	$0<\alpha<360$ Perp.	6/24/2015
P03	1	500	Disc Expand	10^4	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=0 – Get floating point exception for this case…
P04	1	500	Disc Expand	No	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=0
P05	1	500	Disc Diminish	No	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=1
P06	1	500	Disc Diminish	10^4	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=1 - Get floating point exception for this case…
P07	1	500	Sphere	No	No	Radial	6/24/2015, ICN(16)=1, this is the only 1EO radial case that makes sense to test.
P08	2	500	Sphere	No	$0<\varphi<360$ $10<\theta<170$	$0<\alpha<360$ Perp.	6/24/2015
P09	2	500	Sphere	10^4	$0<\varphi<360$ $10<\theta<170$	$0<\alpha<360$ Perp.	6/24/2015
P10	2	500	Disc Expand	10^4	$0<\varphi<360$ $10<\theta<170$	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=0
P11	2	500	Disc Expand	No	$0<\varphi<360$ $10<\theta<170$	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=0
P12	2	500	Disc Diminish	No	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=1
P13	2	500	Disc Diminish	10^6	No	$80<\alpha<100$ Perp.	6/24/2015, ICN(26)=1
P14	2	500	Sphere	No	No	Radial	6/24/2015, ICN(16)=1, this is the only 2EO radial case that makes sense to test.
P15	4	500	Sphere	No	$0<\varphi<360$ $10<\theta<170$	$0<\alpha<360$ Perp.	6/24/2015
P16	4	500	Sphere	10^6	$0<\varphi<360$ $10<\theta<170$	$0<\alpha<360$ Perp.	6/24/2015

Case	EO	N	EO Shape	Big Mass	EO Rot.	Velocity	Notes
P17	4	500	Disc Expand	10^4		0<φ<360 10<θ<170	80<α<100 Perp.	6/24/2015, ICN(26)=0
P18	4	500	Disc Expand	No		0<φ<360 10<θ<170	80<α<100 Perp.	6/24/2015, ICN(26)=0
P19	4	500	Disc Diminish	No		0<φ<360 10<θ<170	80<α<100 Perp.	6/24/2015, ICN(26)=1
P20	4	500	Disc Diminish	10^6		0<φ<360 10<θ<170	80<α<100 Perp.	6/24/2015, ICN(26)=1
P21	4	500	Sphere	No		0<φ<360 10<θ<170	Radial	6/24/2015, ICN(16)=1, this is the only 4EO radial case that makes sense to test.
P22	250	500	Sphere	No		No	0<α<360 Perp.	6/24/2015 – take care in creating this scenario in that the separation distances need to be carefully chosen.

9.2 Solar System Sized Cases

These test cases are solar system sized with SO masses of 0.1 M_{Earth}, Distance of 1A.U. and Big mass is 10^7 M_{Earth}. The case numbers are picked to match the other number type which just have different system scale. Foe example, S02 is a Solar system sized equivalent to P02 which is a sized to planetary dimensions. This list is smaller since these cases are ones that might be the most common or expected scenarios.

Case	EO	N	EO Shape	Big Mass	EO Rot.	Velocity	Notes
S02	1	500	Sphere	10^7	No	0<α<360 Perp.	6/25/2015 – Get floating point exception for this case…
S03	1	500	Disc Expand	10^7	No	80<α<100 Perp.	6/24/2015, ICN(26)=0 – Get floating point exception for this case…
S06	1	500	Disc Diminish	10^7	No	80<α<100 Perp.	6/24/2015, ICN(26)=1 - Get floating point exception for this case…
S08	2	500	Sphere	10^7	No	0<α<360 Perp.	6/29/2015
S11	2	500	Disc Expand	10^7	No	80<α<100 Perp.	6/29/2015, ICN(26)=0 – Get floating point exception for this case…
S13	2	500	Disc Diminish	10^7	No	80<α<100 Perp.	6/24/2015, ICN(26)=1 - Get floating point exception for this case…

10 Notes on RIOD Activities

I will use this section as an Appendix to document other simulation work for my future reference.

Failed attempt at Improved OpenMP efficiency (2/15/2016).

I will document this attempt at improving my simulation engine's performance using OpenMP so that I will remember what I did and not repeat my mistakes.

Monday (2/15/2016) night I woke up early at around 3:00 am and got some good think time in. During that session, I revisited in my mind how I might improve the efficiency of the OpenMP section of my code, which is the simulation's compute engine. During this thought session, I ruminated on how to change from a parallel structure where all the simulation object pairs are calculated matrix style with OpenMP constructs that would hack up the pieces using do-loop chunks. The chuck-size could be specified at run time to find the most efficient number of chunk-like elements for the simulation. However, based on what I understand about OpenMP operations, using this style of looping, each loop has a chunk but the number of calculations in each chuck decrease as the loop continues from start to end.

Here is an example of the current implementation using 11 objects for a simulation, or 55 object pairs. If the simulation uses a chunk size of 3, my current method would calculate 27 of the 55 pairs in one thread, 18 in the second thread, 9 in the third thread and 1 in the final. Clearly this is an extreme example but it illustrates the potential inefficiencies that I suspected.

It is no secret that the above implementation does not produce efficient use of the CPU resources in a multi-core CPU. For example, running the code with 2 CPU might give a benchmark of 28 seconds to do a certain number of iterations. Using 4 CPU, one might expect to see it only take 14 seconds, but in practice, it may only decrease the timing to 18 seconds. The above example is also true adding more CPU; the time for the same number of iterations decreases. However, to me, there still seemed to be an inefficiency in my implementation that what was driving me hard to think about improving the method and impetus for my think session that morning.

My thoughts on changing the algorithm has at its heart hacking up the matrix of object pairs into linearized loop-pieces of equal (or near equal) size. To linearize things, I needed to find where the break points are in the pair matrix, or more specifically, what I, J pairs of objects in the matrix correspond to a uniform number of object pairs that could be coded in similarly sized loops for each thread. For the above example, thinking was if there was three CPU working on the above example problem, then each CPU would need to do 18.333 object pairs but since a loop can't do a third of a loop, one loop will have to do 19 pairs. Still finding the break points for the linearize loops means brute force marching through the matrix and marking the points when you come to them. Since one only needs to do this once (or until the number of objects changes), finding the break points only needs to be done at run time.

Once the break points and the number of object pairs associated with that break point are determined, an OpenMP parallel loop can be started with the number of CPU determining the total number of loop elements. Note now the chuck size is one and thus each loop element has a similar number of computational elements.

Now as I lay awake, I imagined a method that was germinating in my brain over the last year or so, but had not tried and as I lay there, I figured out how the actual coding might go. However, I was still married to the idea that I could use the matrix style code that I currently use but with some modifications. It turns out that ideas I had while lying awake were not truly what I ended up with, for example, I thought I would need to save the start and end points of the matrix breaks but all I needed really was the I,J pairs for the breaks and the number of elements between breaks.

Once I got up, I created a new FORTRAN project to keep current code and anything new I produced separate since there was the potential to really mess things up. To make a long story short, since I had already mostly figured out the algorithm to determine the break points, coding wasn't too hard and was not fraught with too many mistakes. However, I did have to change to the pair and number for break points alluded to above. After a couple missteps, I had code that worked began testing the old code vs. the new code.

When I started testing, what I discovered was there was no improvement, in fact the new code performed worse in some cases. I was flabbergasted by this behavior and am still mystified. I was convinced that this new method would be at its

worst marginally better but results show otherwise. I even turned off all optimization for the two methods, thinking that one method would be optimized better in the compiler for some reason but the performance differences didn't change.

I am sure I won't stop thinking about this but for now I have been thwarted! Ugh…

10.1 Further Notes on Force and Potential Models.

In this section I will explore other forms of the classical gravitational potentials that may be of use in the confines of this simulation. My primary motivation for adding this section is to document some of the ideas that are ruminating around in head. Also because I struggled with the arithmetic around some of this work, documentation is probably a good idea since I will quickly forget the important aspects of these examinations.

10.1.1 Simple Gauss Pass Through

Note that this option is no longer implemented in the simulation but is included for discussion[27].

This section describes a pass-through method that could be used in the simulation code. From Gauss' law it known that for a small mass is inside a larger, uniform mass distribution, the force felt on that smaller mass is related only to the mass inside the current position. Left to the reader as an exercise, the equations can be recast from the result of Equation 1 to two parts:

$$\text{For: } r_{ji} > s, \;\rightarrow\; \vec{F}_{ij} = G \frac{m_i m_j}{|\vec{r}_j - \vec{r}_i|^2} \, \hat{r}_{ji}$$

$$\text{For: } r_{ji} < s, \;\rightarrow\; \vec{F}_{ij} = G \frac{m_i m_j}{s^3} \, r \, \hat{r}_{ji}$$

Where "s" is the sum of the two close-objects radii. Note for when two objects are "inside" each other, the force decreases with distance but outside behaves as two normally gravitating point masses.

One can extend this pass-through concept a bit for the simulation by moderating or "softening" the force between two objects in the method described above as the objects moved closer and inside each other's physical radius.

10.1.2 Smooth Gauss or EX3:1 Based Force Softening and Potential

I have been using the EX3:1 localized method as the primary force softening method for most all recently run simulations. The EX3:1 profile is convenient and short ranged. Recently, I discovered why when using force modifications, calculations of the total system energy are slightly off from what energy conservation dictates. The reason for the deviations was that while calculating the potential energy, I was using a pure Newtonian potential energy for particle pairs that were close enough for force modifications. Clearly, force modifies particle pairs need to have the potential energy calculated associated with force modification.

The potential energy contribution for the EX3:1 profile is not so straight forward. While the force for this profile is easy to work with, the potential energy is not. This is because the analytic solution for the potential energy for a particle pair is not a simple calculation in a spreadsheet or in FORTRAN. In fact, the incomplete Gamma function must be computed, Thus, the Potential Energy from a particle pair for the EC3:1 profile becomes:

$$U(r) = -Gm_1 m_2 \left\{ \left(3 - x\,\Gamma\left(\tfrac{1}{3}, x^3\right)\right) / 3x \right\}.; \; x = r/S$$

I needed a way to calculate the potential energy within the numerical confines of the tools I have. Then I looked closely the at the EX3:1 force, Since the calculation would be done for small values of r/S, I decided to look at an exponential series expansion about x=r/S=0. Below, I will use k=Gm$_1$m$_2$ and S is the sum of the two particle radii.

$$F(r) = -\frac{k}{r^2}\left[1 - e^{-x^3}\right] = -\frac{k}{r^2}\left\{1 - \sum_{n=0}^{\infty} \frac{(-x^3)^n}{n!}\right\} = -\frac{k}{r^2}\left\{1 - 1 - \sum_{n=1}^{\infty} \frac{(-x^3)^n}{n!}\right\} = -\frac{k}{r^2}\left\{-\sum_{n=1}^{\infty} \frac{(-x^3)^n}{n!}\right\}$$

---

[27] Actually, stumbling on some other material I discovered that my recollection of application of Gauss' Law was a bit confused. I was mixing up solutions to various Electrostatics problems and moving too liberally between gravitational potentials and forces. I have now (hopefully) corrected those inaccuracies and also limited the discussion to a more suitable and direct discussion for use in this simulation.

$$F(r) = \frac{k}{S^2} \sum_{n=1}^{\infty} \frac{(-1)^n (x^{3n-2})}{n!}$$

Let's take a step back and recall that for central potentials $\varphi(r)$ we know that for Newtonian gravity, the force on a particle of mass m is:

$$\vec{F} = -m \vec{\nabla} \varphi(r)$$

The potential energy for a pair of gravitating particles is:

$$U(r) = -m\varphi(r)$$

And from above:

$$\vec{F} = -\vec{\nabla} U(r) = -\frac{\partial}{\partial r} U(r)$$

Now suppose we have potential energy for our pair of particles with the following form, where C is an arbitrary constant to be determined later:

$$U(r) = C + \frac{-k}{S} \sum_{n=1}^{\infty} \frac{(-1)^n (x^{3n-1})}{n!\,(3n-1)}$$

A partial derivative of the above with respect to r gives the force series above indicating that it is indeed a potential energy series solution for the EX3:1 force modification. The complication of the unknown constant is remedied when we match the solution with the Newtonian value at x=2, where the modified potential is nearly the Newtonian value. I find the value of C is -1.35412 and must be added to the series sum to get the right behavior.

10.1.3 The Plummer Model

I came across something referred to as Plummer's model[28] for mass density and gravitational potential. Plummer's model gives a full density model as the following:

$$\rho(r) = \frac{3M_T}{4\pi a^3 \ (1 + r^2/a^2)^{5/2}}$$

Note that the Plummer model has a complete solution to the Poisson equation and I have verified that the density function and gravitational potential satisfy the Poisson condition.

In the same reference, the gravitational potential is given as[29]:

$$\phi(r) = -\frac{GM}{(a^2 + r^2)^{1/2}}$$

---

[28] Plummer's model is mentioned in this reference on slide 18: http://www.maths.ed.ac.uk/~heggie/taiwan.pdf

[29] Note the potential in the reference material has an incorrect form in that if you check the units, they are dimensionally incorrect. I have corrected that in my discussion above. What appears in that reference is the following:

$$\phi(r) = -\frac{GM}{(1 + r^2/a^2)^{1/2}}$$

Since the quantity in the denominator above is unitless and we need to divide the above by a distance; this is corrected by dividing by the scaling factor. (Note, Wikipedia confirms this on its page for the Plummer Model https://en.wikipedia.org/wiki/Plummer_model.

The magnitude of the force for this central potential becomes:

$$F(r) = -\frac{GmM\,r}{(a^2 + r^2)^{3/2}}$$

Figure 3 shows the form of the Plummer force and compares it to the other forces considered within this document. Note that the Plummer force defaults to the standard Newtonian form at large distances but much slower than Modified Gauss method currently used in the simulation. Using the Plummer force model would require force softening at a larger separation distance and thus probably is less desirable as a force softening option.
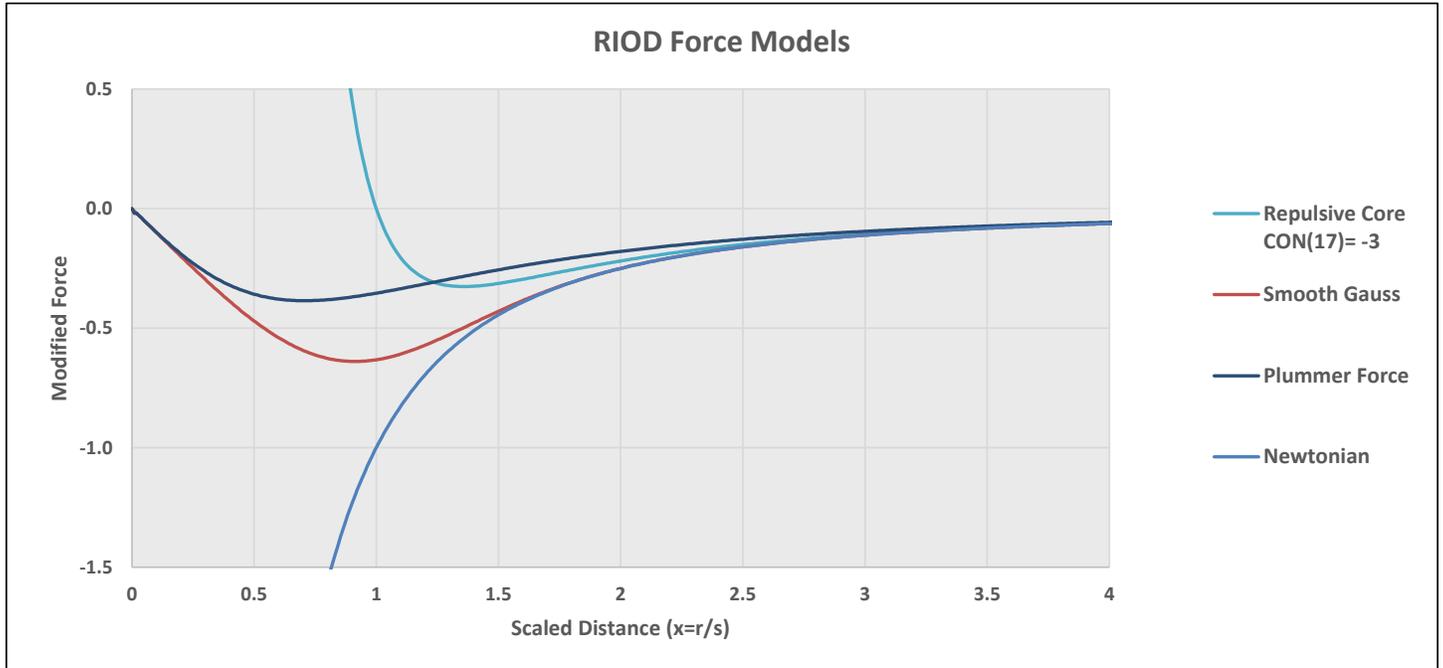

Figure 2: Riod Force models with Plummer Force for comparison.

10.1.4 Piece-wise Continuous Quadratic Force Model (8/6/2016)

Section 4.3.2.8 discussed the currently used force models in the simulation. I have looked at piece-wise continuous potentials and forces off and on over the years. What do I mean by piece-wise continuous? For the application of this simulation, what is meant is that the gravitational potential and/or force is split into two pieces, a piece for distances greater than or equal to the object size and a piece for distances less than the object size. Operationally speaking, using this sort of force for the simulation would be a simple change if I decided that I wanted to implement it.

As I have pondered this sort of modification to the simulation, I have determined that there are constraints for what sort of functions that can be used and also constraints on the way those functions are used. I will use the following method to discuss the inner ($\phi_<$) and ($\phi_>$) and outer functional gravitational potential segments.

Now recall that the gravitational potential for a point particle is given by:

$$\phi = -\frac{k}{r}; where\ k = GM$$

Also remember that the classical central Gravitational force is dependent only on the separation distance, r:

$$\vec{F} = -m\,\overline{\nabla}\,\phi = -m\,\frac{\delta}{\delta r}\,\phi\,(r)\,\hat{r}$$

From here on, we will refer to scaler components of the Force. There are 4 constraints that can be put on the gravitational potential and force. We will constrain the potential functions inside and outside the object size to be equal. We will also require that the derivative of $\phi(r)$ (and hence force) and the derivative of F(r) be equal at r=s. Finally, we want the force to be zero at r=0. These constraints can be summarized below:

$$\phi_<(s) = \phi_>(s)\,;where\ \phi_>(r) = -k/r$$

$$F_<(s) = F_>(s) \; ; where \; F_>(r) = -\frac{k_F}{r^2} \; ; where \; k_F = GmM$$

$$F'_<(s) = F'_>(s) \; ; where \; F'_>(r) = 2\, k_F / r^3$$

$$F_<(0) = 0$$

With the above constraints and the known values for these functions for r>s, we can construct candidates for the potentials and forces. Perhaps the simplest case to examine would be power series in r for the inner potential function. Consider the following:

$$\phi_<(r) = b_3 r^3 + b_2 r^2 + b_1 r^1 + b_0$$

Note that the values, b_i are constants and need to be determined. I have chosen this form for the potential for a couple reasons. First it is the smallest power series in r that will have a non-vanishing second derivative. Also, there are exactly 4 unknowns and four constraints which should yield a unique solution. I will leave it as an exercise to the reader to grind[30] through the "arithmetic" to extract the potential and force for his case. As a hint, the fourth constraint above forces (so to speak) b_1 to be zero. The final forms for the force and potential are as follows:

$$\phi_<(r) = -\frac{k}{s}\left(\frac{r^3}{s^3} - 2\,\frac{r^2}{s^2} + 2\right)$$

$$F_<(r) = -\frac{k}{s^2}\left(-3\,\frac{r^2}{s^2} + 4\,\frac{r}{s}\right)$$

Note that in the above results that when r=s, these functions do indeed default to the expected values. With the above results we can evaluate the suitability of this force model for the simulation. I have added the curve for the force to the same plot in Section 4.3.2.8 to give visual inspection to how it compares with the current methods.

In Figure 3 one can see from the red curve, labeled "Quadratic Force" above the desired properties described in the constraints are met with this set of functions. Overall, the suitability of this force function seems at first blush an interesting variation of the currently used soften force. This new model goes to zero more sharply than the other modified forces and in fact given my empirical experience with the repulse force used with elastic collision option, this might not be a desirable modified force. Recall discussion above where the repulse force results in simulations were the total energy is not conserved, which is a highly undesirable result. All the other implemented force softening methods preserve energy conservation and I suspect that the steep slope of the repulsive portion causes the unwanted behavior. In that light, the Quad Force above has a similar steep slope and may also cause issues.

That said, this new force model is attractive in that force softening would start at the r=s point and not a factor or s farther out. In addition, this new force may have some computational efficiencies over the current method.

10.1.5 Additional Discussion of Force Models.

Figure 3 below shows the force models considered in this document for use within the simulation. The various force models are discussed within this document in the following places:

- Repulsive Core: Section 4.3.2.4
- Smooth Gauss: Section 4.3.2.6
- Plummer Force: Section 10.1.3
- Quadratic Force: Section 10.1.4

Another possible outcome that should be considered is the ability to create/sustain bound pairs. Bound pairs can be created (one can manipulate the initial conditions to have bound pairs too) during a simulation run. Creation of a bound pair is unusual and generally requiring 3 or more particles interacting to create the pair. However, pair creation is an important aspect for the simulation because these types of interactions will drive simulation behaviors which one would expect from real systems of particles.

Bound pairs require a dip in the potential/force the once can see in Figure 3. The Quadratic force for example has the deepest dip and would allow bound pairs well inside the combined radii of the two objects. For a more complete discussion of

---

[30] *The real reason for documenting this result is that I struggled getting the math right on the synthesis for the potential. I just couldn't keep adequate track of minus signs and other constants. So dumb…*

binding of pairs one would have to include a discussion of the "Effective Potential" which includes angular momentum component but that is not the purpose of this discussion. One can generalize important aspects of binding with a blanket statement that the deeper the potential well, the more possibilities for bound conditions exist. Note that for the repulsive core force used for the elastic collisions option, there are the fewest options for bound states as it has the shallowest well of any of these options.
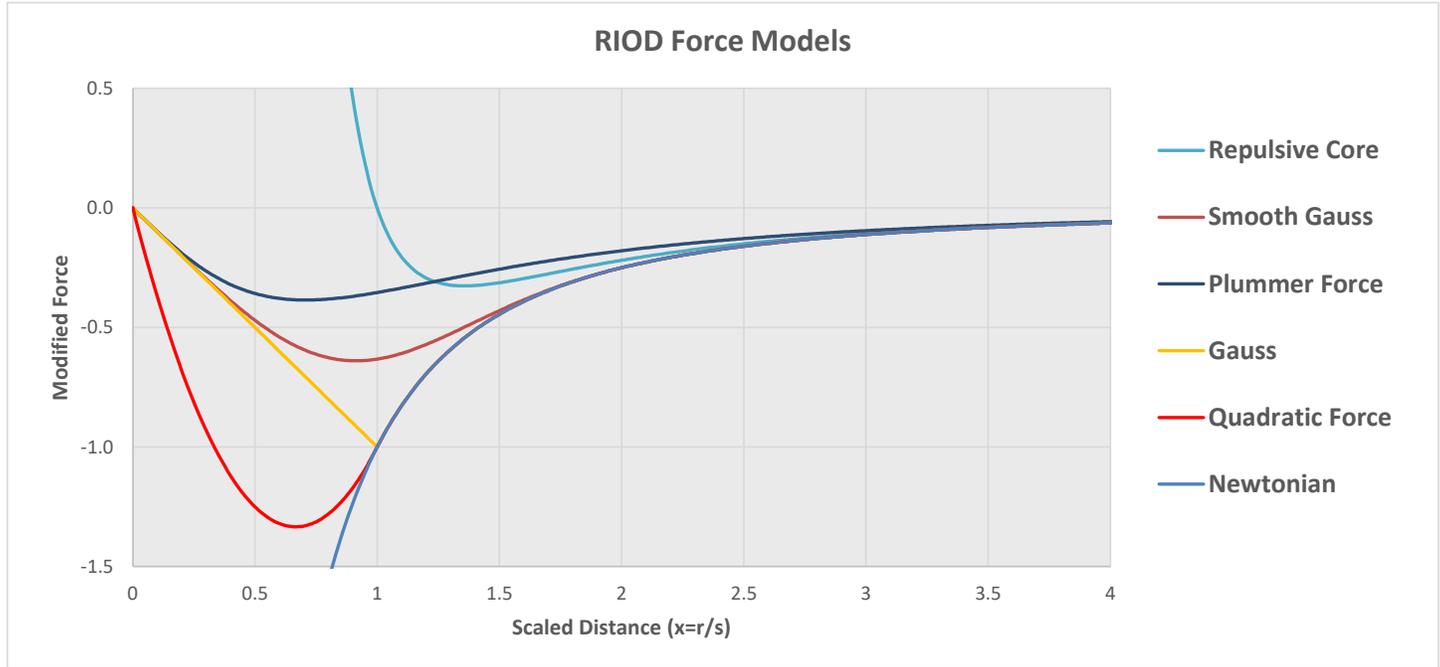

Figure 3: Force Models

10.2 Discussion on Replacement Collision Methodology Failures

Replacement collisions are a variation of collision type implement with Riod but so far have vexed me as to how best to create this type of scenario. Specifically, as a system of objects evolves, collisions will happen and I want to move the colliding objects out farther from the rest and let them fall back into the group. Unfortunately, in order to keep this as a physically viable configuration, there are some propertied of doing this that somewhat undesirable. I will discuss those in the text body in Section 4.3.2.2. This appendix section will address the failures that I have tried.

10.2.1 Attempt for setting velocity content for replacement collisions.

I tried the following method but the results were unsatisfying. So far whatever I have tried has shown that the mass distribution continues to expand and a core never really forms.

A new method determine how to scale the new particle's velocity. The velocity eccentricity is now determined to allow the CON(31) to be the maximum orbital radius and the minimum is set to be $0.75*R_{vir}$. Recall that or elliptical orbits (we make the assumption that the orbital character would be elliptical), the semi-major axis distance "a" is related to the maximum and minimum orbital distances as:

$$R_{max} = a(1 + \varepsilon) \; and \; R_{min} = a(1 - \varepsilon)$$

Currently CON(31)*R_{vir} is R_{max} and R_{min} is scaled as a random number between R_{vir} and $0.75*R_{vir}$. As an exercise to the reader, the eccentricity ε takes on the following range:

$$\varepsilon_{Max} = \frac{CON(31) - 0.75}{CON(31) + 0.75} \; and \; \varepsilon_{Min} = \frac{CON(31) - 1}{CON(31) + 1}$$

10.3 Old Method of Determining Eccentricities

I no longer use this section for eccentricities for this use case; all particles speeds are determined as above.

The only thing that changes from the above discussion is how the eccentricity is determined. It became apparent with some of these new mass distribution profiles that some change in how the eccentricity of the initial orbit must be modified. For example, the NFW and Jaffe profiles create distributions with particles extended far from the profile origin. It would not be wise to create these objects with eccentricities with $\varepsilon<0$; implying they are at a minimum orbital distance. What one needs is for these objects to be in highly eccentric orbits that will bring them back to the core area of the distribution so the probability of interactions is higher.

Conversely, these profiles that have high densities at the origin need to have their eccentricities such that their initial velocities taken them away from the origin. This means that for the SO with the smallest distances from the origin, their eccentricities should all be less than zero.

Given the constraints and concerns above, a new method was devised to make the eccentricity range used by the simulation to be dependent on the position of the SO and the distribution virial radius (R_V) and the provisioned eccentricity range given between $\varepsilon_{pn} = $CON(7) and $\varepsilon_{px} = $CON(8). The code uses ranges of eccentricity, maximum (ε_{Rx}) and minimum (ε_{Rm}) and they are determined by:

Eccentricity Range	Conditions based on R_V	Internally calculated Range based on position and R_V
ε_{Rx}	$0 < \dfrac{r}{R_V} \leq 9$	$\varepsilon_{px} log(1 + \dfrac{r}{R_V})$
ε_{Rx}	$\dfrac{r}{R_V} > 9; \dfrac{r}{R_V} = 9$	ε_{px}
ε_{Rm}	$\dfrac{r}{R_V} < 1$	$-\varepsilon_{pm}$
ε_{Rm}	$1 \leq \dfrac{r}{R_V} \leq 9$	$\dfrac{\varepsilon_{px} + \varepsilon_{pm}}{log(9)} \varepsilon_{px} log\left(\dfrac{r}{R_v}\right) - \varepsilon_{pm}$
ε_{Rm}	$\dfrac{r}{R_V} > 9; \dfrac{r}{R_V} = 9$	$(\varepsilon_{px} + \varepsilon_{pm}) \varepsilon_{px} - \varepsilon_{pm}$

Error! Reference source not found. below visually shows how the eccentricity range is determined. Depending on the value of r and R_V, the ranges of eccentricity fall between the black and red curves below. As show in the above table, for distances greater than 9 times R_v, that ratio is set to 9 internally. This causes the range to flatten as seen in the figure.
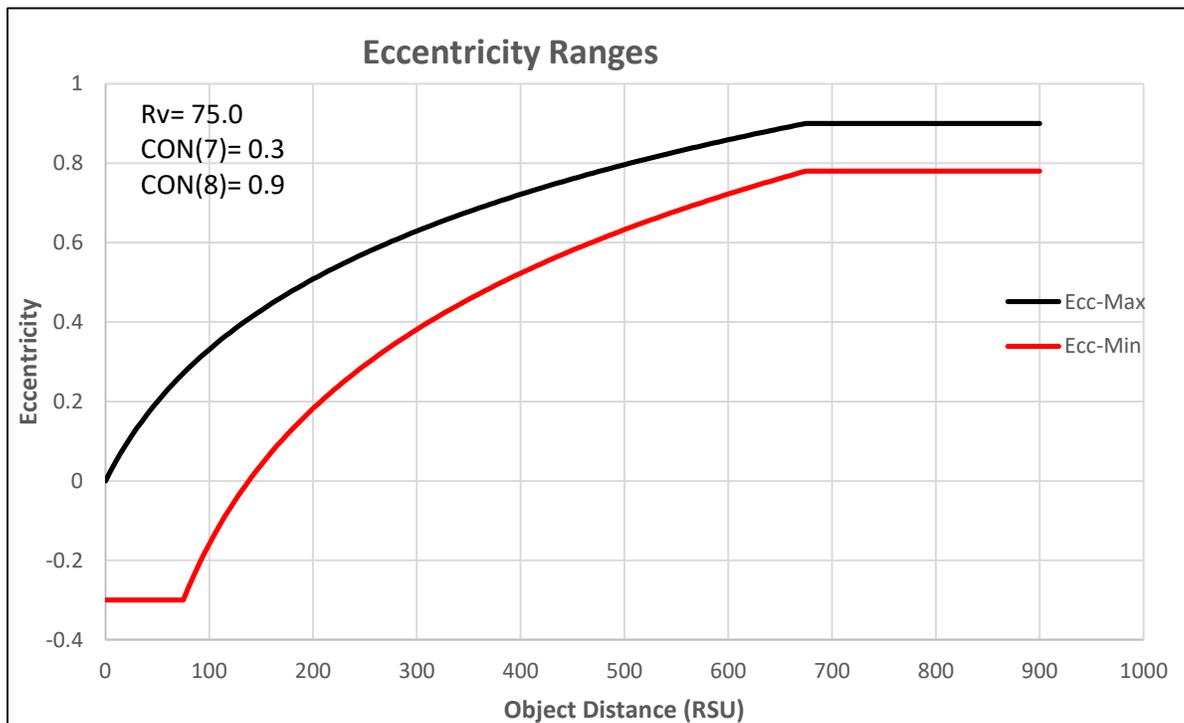
Figure 4: Eccentricity ranges as determined by the new method for distribution profiles when ICN(28)>0. Here the virial radius is 100 and the provisioned ranges are used as per the table.

The net effect of this method is to create orbital speeds need the distribution center that are essentially an orbital speed minimum as the SO is at its orbital periapsis. For SO far from the distribution center, they are given an eccentricity range to ensure orbital speed is consistent with an apoapsis position and will fall back to the distribution center.

10.4 Universal Gravitation Constant History

The code needs to use the universal gravitational constant, G. I try to keep up with the latest value for G and for reference I will create a table with the values I have used and the time periods when specific values are used.

Begin Date	End Date	Value (x10^{-11} N$\cdot$m^2 $\cdot$kg^{-2})
1/19/2016	To Date	6.67408
Before 1/19/2016	1/19/2016	6.67384

11 Acknowledgments

The work documented here was spawned by my working in a vacuum for years/decades, using stuff I learned from my Physics training and extensions of that knowledge. Literature references that helped with certain concepts are captured in footnotes throughout the document. This section acknowledges others who have contributed in some way to the software development that I would like to thank.

11.1 Julie Zhu (12/1/2012)

During one of our many discussions about things over the years, the subject of this simulation would come up. After expressing unhappiness with the FORTRAN package I was using and the lack of affordable options, Julie suggested I look into the free GNU FORTRAN. This suggestion (something I should have thought of myself, D'oh!) started a chain of events that led me to the Simply FORTRAN product. Changing to this new development environment spurred a burst of activity that led me to develop multithreaded code, the creation of the new RPLOT.EXE data viewing program, and the creation of many other tools and features. Thanks Julie!

11.2 Joe Henson (8/8/2016)

Joe helped me with a solution to a mystifying problem with MS Word. For some reason, Word will inexplicably change the header/footer spacing but the worst was the spacing increases for footnotes. The fix for this (so I will remember the next time this happens as it has happened more than once) is the following:

- Go into the "View" tab and click the "Draft" mode button.
- Then go to the "References" tab and click the "Show Footnotes".
- At the bottom of the page, there is a drop-down box labeled "All Footnotes". Enter each option of that drop-down list and remove all the unwanted spacing.

Thanks Joe!

11.3 Sean Emer (3/19/2020)

As a videographer, Sean provided video expertise as I could not get my videos to play properly on Youtube. His solution was to create the video in 4K resolution, which forces Youtube to treat the video with higher quality. This did work for me as now the videos have a much better viewing experience on Youtube. Thanks again Sean!

11.4 Ed Rojek (8/12/2021)

I have known and been friends with Ed since graduated school days. He is a go-to source of knowledge on many subjects and is always willing to help. I consulted with Ed regarding the particle-point-of-view feature now included in Rplot.exe. I was struggling with how to display the data for this feature but at the time, I was just trying to figure out why the output looked wrong. The discussion with Ed helped crystalize some concepts but also helped me rethink what I was doing and

find the bugs in the code, which were dumb (interchanged the names of the Z and Y rotation subroutines) and an ill-conceived coordinate system model.

However, the most important aspect of our conversation was that he reminded me that I needed a reference imaging method. I had considered creating a camera perspective but was originally thinking that the transformation math would be prohibitive. After fixing the above bugs, I dug into the imaging aspect and found a simple camera obscura (pin hole camera) transformation which suited my needs perfectly[31]. See section **Error! Reference source not found.** for details on how this Rplot.exe feature works.

Thanks for your help, Ed and your continued friendship.

---

[31] *Pin hole camera transformation link: http://www.cs.toronto.edu/~jepson/csc420/notes/imageProjection.pdf*