Print Form

University Honors College
HONORS CONTRACT

Course Eligibility: See Page Two

Please Print/Type

SEMESTER: ☑ **Fall** ☐ **Spring** ☐ **Summer** **YEAR:** 2018 **ID#** M01250797

DUE SEPT. 30th | DUE FEB. 15th | DUE MAY 30th

STUDENT NAME Cole (last) Jackson (first) Lane (middle) **E-MAIL** jlc2de@mtmail.mtsu.edu

MAJOR Physics **CELL PHONE** 615-971-0888 (if no cell #, then home #)

MINOR Mathematics **GRADUATION SEMESTER** Fall 2018 **GPA** 3.875

COURSE TITLE MATH 3180 - Introduction to Numerical Analysis **CREDIT HOURS** 3.000

COURSE RUBRIC and NUMBER N/A **CRN#** 80907

INSTRUCTOR Dr. Suk Jai Seo **E-MAIL** Suk.Seo@mtsu.edu

Describe the additional course requirements. ***Specifically*** address how the work proposed by this contract is different from and/or expands upon the requirements for the regular course.

☑ The regular class syllabus is attached

☑ A one-page detailed description of the Honors component is attached (see back page)
The Honors Contract will not be approved without BOTH of these documents!

We, the undersigned, hereby agree to pursue the project detailed above during this semester. As a result of the successful completion of this work, an Honors designation will be applied to the above-named course as it appears on the student's official transcript.

Approval Signatures

Student Date

Instructor Date

Honors College Academic Advisor Date

Honors College Dean or Associate Dean Date

NOTIFICATION OF COMPLETION

The attached e-mail notification certifies that _____ has completed the requirements above and has earned University Honors credit for the identified course.

Honors College Executive Secretary Date

Guidelines for the Honors Contract

UPPER-DIVISION CREDIT

Honors students may obtain Honors credit in a Non-Honors upper-division course in his or her major, concentration, or minor with an Honors Contract. Students may receive Honors credit in unlimited upper-division courses, but Honors College graduates can **only apply a maximum of four (4) hours of Honors Contract hours** toward the eleven-hour, upper-division Honors College requirement.

LOWER-DIVISION CREDIT

Honors students may obtain Honors credit in a Non-Honors lower-division course (2000 level) that is ***never*** offered as an Honors section. This does not apply if an Honors course isn't offered the semester you want to take it (i.e. a course offered every fall, but not spring, would not qualify in the spring). Students may receive Honors credit in unlimited lower-division courses (2000 level), but Honors students can **only apply a maximum of six (6) hours of Honors Contract hours** toward the 18-hour, lower-division Honors College requirement.
FRESHMAN LEVEL 1000 COURSES DO NOT QUALIFY.

The Honors Contract should create a partnership of mutual benefit to both the student and the faculty member. For an Honors student, the Contract should involve a project or activity that delves deeper in the course material and results in a better appreciation and understanding of the subject matter that can be used as a positive experience in preparation for future goals. Faculty members can use the Honors Contract to try innovative or interesting activities or projects that are not practical to do with an entire class. In all cases, the Honors component in the Honors Contract should target one or more of the following areas: scholarship, leadership, or service. **Questions regarding the nature of a specific Honors Contract can be directed to Dr. John Vile, (John.Vile@mtsu.edu), Dean, 615-898-2152, or Dr. Philip Phillips (Philip.Phillips@mtsu.edu), Interim Associate Dean, at 615-898-2699.**

The Honors Contract must ***explicitly state*** the work the student will undertake to earn the Honors designation. That work should be more rigorous than and/or go above and beyond what is expected of the other students in the course. For instance:

Students may undertake an additional, or more demanding research project;
give a special presentation to the class;
participate in a workshop;
expand a paper for presentation or submission at a conference;
work as a research assistant;
be involved in an internship;
conduct field work;
create/exhibit exceptional art work; or
give a performance or concert.

The contract should also state ***specifically*** *how* the work proposed by the Contract is different from the requirements of the other students in the course. For example:

Katie will write a research paper. **WILL NOT MERIT APPROVAL**
Katie will write a longer research paper. **WILL NOT MERIT APPROVAL**
Katie will complete additional projects to be decided upon as the course progresses. **WILL NOT MERIT APPROVAL**

The correct way to address the Honors Contract should be similar to, or as detailed as:

For Honors credit, Katie will write a 15-page research paper on a theory of ethics not covered in this class. This extra writing assignment will require Katie to use at least two other outside resources and to apply in-depth research methods and critical thinking.

At the end of the semester, the professor should send an e-mail confirmation (to Karen.Demonbreum@mtsu.edu) indicating that the student has completed the Honors component of this course. This should be done no later than seven (7) days after the final exam for the course. The Honors College will then notify the Records Office to assign the Honors designation on the student's transcript.

Introduction to Numerical Analysis, CSCI/MATH 3180
Fall 2018

Instructor: Dr. Suk Jai Seo

Office: KOM 303-A (904-8292)

Class time: TR 2:40-4:05pm

Office hours: TR 12:45 – 2:00 pm, TR 4:05-5:00pm
Other times by appointment and generally available seven days a week via email. See below for the details.

Email address: Suk.Seo@mtsu.edu

D2L: The D2L page for this class will contain links to course information, including this syllabus and some assignments. Access D2L through PipelineMT or directly using https://elearn.mtsu.edu/.

Catalog Description: Topics include series approximation, finite differences interpolation, summation, numerical differentiation and integration, iteration, curve fitting, systems of equations and matrices, and error analysis.

Prerequisites: MATH 1920 and CSCI 1170, each with a grade of C or higher.

Textbook: Numerical Mathematics and Computing (7th Edition) by Cheney and Kincaid

Learning Outcomes: Upon completion of this course, students should be able to

- understand a variety of methods by which a modern digital computer can be used to solve numerical problems
- understand how errors arise in numerical computing and methods for detecting, predicting and controlling errors
- have sharpened programming and problem solving skills

Course Communication:
Students are responsible for monitoring for MTMAIL **daily**. Following MTSU's FERPA-based e-mail policies, all course-related e-mail will be sent to your MTMAIL account; in turn, you are **required** to use your **MTMAIL** account when communicating with the instructor.
Emails sent to the instructor must contain the name of the sender, the subject, and the detailed message.
Do not try to contact the instructor through D2L.
Use Suk.seo@mtsu.edu

Use of personal electronic devices in the classroom:
Cell phones, laptops, tablets, and other electronic devices must be turned off and put away during class unless the instructor determines that these devices are allowed to be used in the class. Students are not permitted to take photos or record any part of a class/lab unless explicitly granted permission by the instructor or the MTSU Disability Access Center. Sanctions for violation of this policy will be determined by the instructor and may include dismissal from the class, attendance penalties or loss of class participation points, zero grades on quizzes or examinations, failure in the class, or other penalties that the instructor determines to be appropriate.
All course materials are protected by the law of copyright. Students are not authorized to use, reuse, reproduce, distribute, broadcast or publish the course materials, or any part of the course materials, in any medium, including via the internet and social media sites. This means that students are not allowed to photograph or reproduce course materials, and that students are not allowed to record lectures, except as provided for by an approved ADA request.

Academic honesty: *The Computer Science Department's Policy on Academic Integrity applies to this course. All work for this class (including exams, homework, and labs) is to be done on an individual basis. Unless otherwise directed, work alone on lab assignments. The penalty for unauthorized collaboration will range from a grade of zero for an assignment to a failing grade for the course.*

Attendance: Attendance is required and absences do not excuse one from class responsibilities. ***If for some unavoidable reason you must miss class, obtain class notes, handouts, and assignments from another class member.*** You are expected to be on time for class. Consistent lateness to class is disruptive and is considered to be

disrespectful. It is best to come late, however, rather than not at all!
Attendance is determined/judged/counted by the student being in attendance through the lecture and that student signing the name on the daily attendance sheet. It is each student's responsibility to locate and sign this sheet each class period during the class.
Students failing to attend the first two class meetings will be dropped from the class.

Grading policy:

- **Examination grades** – there will be four examinations counting 50 points each and a final examination counting 100 points. These exams will cover lectures, assigned readings, and labs. **No make-up exams will be given.** *If you miss a regularly scheduled exam, 50% of the final exam score will replace this exam.* If an exam is not missed then 50% of the final exam score can replace the lowest test score.
- **Laboratory assignments -** There will be ten lab assignments and they will be worth 100 points in total.
- **Quizzes -** There will be a quiz daily and they will be worth 100 points in total.

Point System: 500 total assigned points

- A: (450-500 pts)
- B: (400-449 pts)
- C: (350-399 pts)
- D: (300-349 pts)
- F: (0-299 pts)

WITH THE FOLLOWING EXCEPTIONS:

1. The total points of the five exams must be at least 180 points (60 %) to pass the course.
2. The total points of the Lab Assignments must be at least 60 points (60 %) to pass the course.
3. The total points of the quizzes must be at least 60 points (60 %) to pass the course.

NOTE: Any questions concerning a grade on a lab or exam must be handled within five days of the time the item was returned.

Attendance Bonus:

Absences	0	1	2
Bonus points	15	10	5

Important dates:

September 6	Exam 1	October 31	Last day to drop with a "W"
September 9	Last day to drop without a grade	November 15	Exam 4
September 27	Exam 2	December 4	Last Day of Class
October 25	Exam 3	December 11	Final Exam (3:30-5:30)

Reasonable accommodation for students with disabilities: If you have a disability that may require assistance or accommodation, or you have questions related to any accommodations for testing, note takers, readers, etc., please speak with the instructor as soon as possible. Any student interested in reasonable accommodations can consult the *Disability & Access Center (DAC)* website **www.mtsu.edu/dac**. Students may also contact the *DAC* for assistance at 615-898-2783 or ***dacemail@mtsu.edu***. Middle Tennessee State University is committed to campus access in accordance with Title II of the Americans with Disabilities Act and Section 504 of the Vocational Rehabilitation Act of 1973.

FINANCIAL AID NOTICE:
Do you have a lottery scholarship? To retain the Tennessee Education Lottery Scholarship eligibility, you must earn a cumulative TELS GPA of 2.75 after 24 and 48 attempted hours and a cumulative TELS GPA of 3.0 thereafter. A grade of C, D, F, FA, or I in this class may negatively impact TELS eligibility.

If you drop this class, withdraw, or if you stop attending this class you may lose eligibility for your lottery scholarship, and you will not be able to regain eligibility at a later time.

For additional Lottery rules, please refer to your Lottery Statement of Understanding form (http://www.mtsu.edu/financial-aid/forms/LOTFEV.pdf) or contact your MT One Stop Enrollment Coordinator (http://www.mtsu.edu/one-stop/counselor.php).

PROBLEMS, COMPLAINTS, OR SUGGESTIONS:
If you are having problems with the course, or have a complaint or suggestion you would like to voice, please bring this to the attention of the course instructor as soon as possible.

FREE TUTORING!
Learn how to study, get help with understanding difficult course material, receive better test grades, or simply improve your grade point average. Take advantage of our FREE tutoring that is available to you as an MTSU student. Tutoring is available in study skills and learning strategies, and over 180 courses including biology, history, computer information systems, physics, math, psychology, chemistry, economics, recording industry, and many more. The central location for tutoring is the Tutoring Spot, located in Walker Library. Tutoring is also conducted at various other campus sites. For available tutoring opportunities, visit http://mtsu.edu/studentsuccess/tutoring.php#on. For questions, call the Tutoring Spot at 615-904-8014.

TENTATIVE **Schedule (CSCI/MATH 3180)**

Dates	**Topic**	**Labs/Exams**
8/28, 30	Visual Studio 2015 Maple Chapter 1: Mathematical Preliminaries	LAB1
9/4, 6	Chapter 4: Numerical Differentiation	EXAM1
9/11,13	Chapter 7: Initial Value Problems	LAB2
9/18, 20	Chapter 7: Initial Value Problems	LAB3
9/25, 27	Chapter 2: Linear Systems	EXAM2
10/2, 10/4	Chapter 2: Linear Systems	LAB4
10/9, 11	Chapter 4: Interpolation	LAB5
10/18	Chapter 4: Interpolation	LAB6
10/23, 25	Chapter 6: Spline Functions	EXAM3
10/30, 11/1	Chapter 6: Spline Functions	LAB7
11/6, 8	Chapter 3: Nonlinear Equations	LAB8
11/13, 15	Chapter 3: Nonlinear Equations	EXAM4
11/20, 22	Chapter 5: Numerical Integration	LAB9
11/27	Chapter 5: Numerical Integration	LAB10
12/4	Review	
12/11	Final Exam (3:30-5:30pm)	FINAL EXAM

Honors Report
Evolutionary Runge-Kutta[1]

Jackson L. Cole

Department of Physics and Astronomy, Middle Tennessee State University

Fall 2018

[1] https://github.com/jacksonlanecole/rkev

Contents

List of Figures

1 Introduction

When analyzing physical systems, there are often cases when analytical solutions to ordinary differential equations describing the systems are either exceedingly difficult to find or simply do not exist. In these cases, it is advantageous to use numerical methods to iteratively solve the systems. While there are several methods for solving ordinary differential equations, the "workhorse" of initial value problem solvers is the fourth-order Runge-Kutta (RK4). [2]

Essentially, the RK4 method involves computing derivatives at four different points in the parameter space, and updating the solution at the iterated parameter to take into account the "weighted average" of the computed derivatives. In other words, if we define a function $f(t, x)$, we can say that the derivatives are

calculated as shown in equation (1).

$$
\begin{aligned}
k_1 &= f\left(t_i,\, x_i\right) \\
k_2 &= f\left(t_i + \frac{1}{2}h,\, x_i + \frac{1}{2}k_1\right) \\
k_3 &= f\left(t_i + \frac{1}{2}h,\, x_i + \frac{1}{2}k_2\right) \\
k_4 &= f\left(t_i + h,\, x_i + k_3\right)
\end{aligned}
\tag{1}
$$

The solution is updated

$$
x = x_i + \mathbf{\frac{1}{6}}\left(k_1 + 2k_2 + 2k_3 + k_4\right)h \tag{2}
$$

and the time is naturally updated by

$$
t = t_i + h. \tag{3}
$$

Eventually, from the given function which we assume to be the derivative of some function of interest, the Runge-Kutta will approximate the real value of the function of interest at some parameter. For this reason, we tend to call these methods "integrators", as they find a value for the real function of interest when all we know is the derivative.

However, while the choice of the bolded leading coefficient seems to come from a desire to appropriately average the derivatives, Holmes [2] points out that this constant is largely arbitrary, but is as standard chosen to be $\frac{1}{6}$ to match the results of other methods. This derivation is not of importance in the case of this project.

To be more explicit in the definition of the scheme of the RK4, we could express it a bit differently, as shown in equation (4).

$$
\begin{array}{lllllll}
k_1 = f(t_i & +(0)h, & x_i+ & (0)k_1 & +(0)k_2 & +(0)k_3 & +(0)k_4 \quad) \\
k_2 = f(t_i & +(\frac{1}{2})h, & x_i+ & (\frac{1}{2})k_1 & +(0)k_2 & +(0)k_3 & +(0)k_4 \quad) \\
k_3 = f(t_i & +(\frac{1}{2})h, & x_i+ & (0)k_1 & +(\frac{1}{2})k_2 & +(0)k_3 & +(0)k_4 \quad) \\
k_4 = f(t_i & +(1)h, & x_i+ & (0)k_1 & +(0)k_2 & +(1)k_3 & +(0)k_4 \quad) \\
x = & & x_i+ & [(\frac{1}{6})k_1 & +(\frac{2}{6})k_2 & +(\frac{2}{6})k_3 & +(\frac{1}{6})k_4 \quad]h
\end{array}
\tag{4}
$$

If we extract the coefficients of the step sizes, the internal coefficients, and the weights of the derivatives in

the bottom line of equation (4), with some abstraction, we obtain what is known as the Butcher Tableau describing the scheme of the RK4. This is shown in Figure 1. Because this is clearly just a lower triangular matrix, we can more simply express it as in Figure 2. It should be noted that explicit Runge-Kutta routines have the requirement that their associated Butcher Tableau is strictly lower diagonal, as this removes the possibility of needing to solve implicit equations while calculating successive derivatives. Further, a sample C++ snippet has been given for running an RK4 routine as shown below in listing 1.

0	0	0	0	0
$\frac{1}{2}$	$\frac{1}{2}$	0	0	0
$\frac{1}{2}$	0	$\frac{1}{2}$	0	0
1	0	0	1	0
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Figure 1: Full Butcher Tableau for explicit fourth-order Runge-Kutta (RK4). This form comes from an abstraction of the full RK4 scheme.

0				
$\frac{1}{2}$	$\frac{1}{2}$			
$\frac{1}{2}$	0	$\frac{1}{2}$		
1	0	0	1	
	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{6}$

Figure 2: Butcher Tableau for explicit fourth-order Runge-Kutta (RK4). This form of the tableau is found in many texts, so I will omit a reference here.

Listing 1: Fourth-order Runge-Kutta (RK4)

```
// C++
for (int i = 0; i < nSteps; i++) {
    k1 = f(t, x);
    k2 = f(t + (1/2.)*h, x + (1/2.)*k1);
    k3 = f(t + (1/2.)*h, x + (1/2.)*k2);
    k4 = f( t + h, x + k3);
    x = x + h*(k1 + 2*k2 + 2*k3 + k4)/6;
    t = t + h;
}
```

Although the Runge-Kutta method is much more accurate than lower order methods or methods like Euler, an issue that arises in the study of physical systems is that we are often more aware of the conditions of our system than is our integrator. When we look at systems in celestial mechanics, like the three-body problem that has no analytical solution, we should acknowledge that under the assumption that the system

is closed, the total mechanical energy should remain constant. Operating with this knowledge, the validity of the solution provided by the RK4 must be called into question as the total energy is *not* conserved over long time integrations. [2]

When modeling physical systems, a common method of of preserving conserved quantities is to use *symplectic* integrators. While the scope of this project does not necessarily extend to symplectic integrators outside of the possible implementation of one for comparison of the validity of this method, they should receive some discussion. In general, many physical systems can be described in the Hamiltonian formulation of mechanics; this is a far more flexible approach to mechanics than is Newtonian or even Lagrangian formulation, especially in terms of the generalization of coordinates and the conservation of naturally conserved quantities. **The following derivation of the Hamitonian is based on that found in Taylor [3].**

In general, we can arrive at what is known as the Lagrangian for a particular system, which can be defined by

$$\mathcal{L} = T - U \tag{5}$$

where T is the kinetic energy expression for the system, and U is the potential energy expression. We also assume that the Lagrangian is a function of n generalized coordinates

$$\mathcal{L} = (q_1, q_2, q_3, \ldots, q_n, \dot{q}_1, \dot{q}_2, \dot{q}_3, \ldots, \dot{q}_n) = T - U \tag{6}$$

which means that in terms of the generalized coordinates, Lagrange's equations of motion can be written as second order differential equations as

$$\frac{\partial \mathcal{L}}{\partial q_i} = \frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} \quad \text{where} \quad i = [1, \ldots, n] \tag{7}$$

However, using the Hamiltonian approach allows us to more easily express the equations of motion as two first order differential equations. We can define the conjugate momentum (momenta in multiple dimensions) is expressed as

$$p_i = \frac{\partial \mathcal{L}}{\partial \dot{q}_i} \tag{8}$$

Now with this defined, the Hamiltonian is defined as

$$\mathcal{H} = \sum_i^n p_i \dot{q}_i - \mathcal{L}. \tag{9}$$

Without derivation, the equations of motion can be expressed as

$$\dot{q} = \frac{\partial \mathcal{H}}{\partial p} \qquad \dot{p} = -\frac{\partial \mathcal{H}}{\partial q}. \tag{10}$$

Using these equations of motion, integration schemes can be defined that iteratively solve the system on a symplectic manifold on which the system is defined in the first place, which means that quantities that are conserved in the physical system will also be conserved in the iterative solution.

2 Purpose

While symplectic integrators are the go-to choice for numerically solving systems in which a Hamiltonian can be defined, Runge-Kutta methods are still widely used for solving physical systems as well. Further, the Runge-Kutta method should be able to be optimized such that conserved quantities in *any* system would be respected. To be more general, in any system in which a cost function can be defined, the hope is that a method can be developed that improves the results on a system specific basis. The primary directive of this project is to explore the use of evolutionary algorithms to evolve the Butcher Tableau describing an s-stage Runge-Kutta scheme to improve explicit Runge-Kutta methods in systems in which a cost function can be defined. In celestial mechanics, for example, the obvious examples are gravitationally bound systems in which the energy of the system must be conserved to ensure the stability of orbits.

A secondary goal relates to a personal interest in multi-language projects, and this will be explored by implementing the computationally intensive portion of this project in C++ and wrapping it as a Python library.

3 Methods

3.1 General purpose explicit Runge-Kutta algorithm

In order to have the ability to optimize the weights, nodes, and internal coefficients of the tableau describing a Runge-Kutta scheme, a general purpose Runge-Kutta algorithm was developed. Without specific details about processes in the Runge-Kutta algorithm, the general idea can be visualized in Figure 3.

For this reason, a general Runge-Kutta integrator shared library has been implemented in C++ using the Boost Python Library. This library contains methods for iteratively solving an IVP for either a single value or for an array of values, and does so using a Runge-Kutta scheme defined by a general Butcher Tableau and an abstracted algorithm to process the Butcher Tableau.

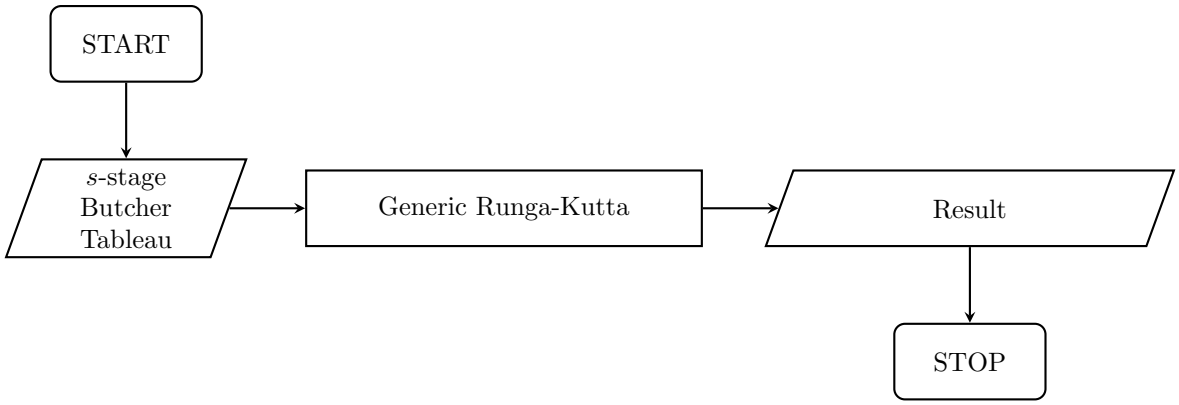


Figure 3: Generic Runge-Kutta algorithm

This algorithm should be flexible enough to handle any explicit Runge-Kutta scheme that can be defined in a Butcher Tableau that is strictly lower-triangular. For instance, the fifth order Runge-Kutta-Fehlberg method essentially follows the general RK form, but has an extra step and is defined by its lower-triangular Butcher Tableau. The methods included in this library should handle the entire tableau appropriately. While this has not been tested, there have been even higher order Runge-Kutta schemes established ($s > 10$) in astronomical simulations that should be able to be processed by this algorithm.

If the integrator is given the Butcher Tableau for the classic fourth-order Runge-Kutta (RK4) as in Figure 2, the integrator will integrate using a classic RK4 routine. Although there are also implicit Runge-Kutta methods that work with non-lower-triangular Butcher Tableaus, my algorithm assumes that the matrix given is strictly lower-triangular, and therefore, only explicit Runge-Kutta schemes can be used.

3.2 `nbody` package

In astronomy and astrophysics, a problem that arises in nearly every discussion of celestial mechanics is simply known as the n-body problem. When two bodies are in orbit around a common center of mass, their orbits can be described by purely analytical functions. In other words, the motions of two orbiting bodies have solutions that can simply be written down on paper. In the case of systems of n-bodies where $n \gg 2$, the solutions describing the motions of the bodies can be approximated by cloud approaches, where each particle or star is not analyzed individually. However, in cases where the number of bodies is greater than two, but less than that which could be approximated by cloud physics, an analytical solution to the n-body problem does not exist, and therefore, the problem must be solved by numerical methods alone.

The initial motivation for this project was the inherent loss of energy that occurs when iteratively solving the n-body problem without using a symplectic integrator. For this reason, the method by which the general Runge-Kutta library was going to be tested was by iteratively solving the n-body problem for systems where

$n < 4$. Rather than do an ad-hoc implementation of this, a better solution supporting reproducibility of results and simplification of the codebase was to implement `Body` and `System` classes, where the `System` class is a system of n `Body` objects. These are implemented in Python, and are now in a package called `nbody`.

The `System` class contains methods for calculating the radii between all bodies in the system and accelerations of all bodies due to other bodies. The Runge-Kutta library expects to receive a name of a Python function receiving arguments relating to the time step, the current value, and in the case of the vector Runge-Kutta methods, the index of the current element in the array. This allows users to write their differential equations as Python functions, which are then called in the Runge-Kutta library.

This class also saves positions and velocities for each of the bodies in the system, which can then be visualized after it finishes running.

Further, by creating this class, the interaction with the Runge-Kutta library can be veiled in a simple `run` method. This then handles computation of the gravitational forces between each pair of bodies in the system and performs the integration at each time step.

4 Preliminary results

At this point, the Runge-Kutta shared library has been successfully built and tested with several trivial first order ordinary differential equations. It successfully accepts any lower-triangular Butcher Tableau and performs the RK routine it defines.

Further, a n-body model has successfully been written in Python using the Runge-Kutta library previously mentioned. The interface to this `nbody` package is very simple, and requires that the user define a few bodies, instantiate the system using a list of these bodies, and then define the time bounds and step size to set up the RK integrators for x, y, and z positions and velocities. Then the user just has to use the run method provided to solve the system.

While this is now working, the results still reflect the lack of correction for energy loss that occurs as a result of our method of integration. Basically, the Runge-Kutta library works as expected, but returns bad solutions due to the nature of the problem. One such solution can be seen in Figure 4, where we simulate the Earth-Sun system. Without correction or optimization to minimize changes in energy, the orbit of the Earth rapidly decays in less than one revolution. This is the point at which evolutionary programming can be used, as at this point, we operate on the assumption that the n-body problem can be considered as an optimization problem for some evolutionary algorithm, where the cost function to minimize is the energy loss. The next steps are described in 5.

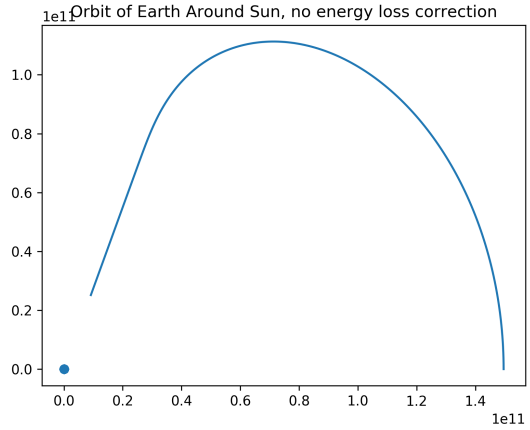


Figure 4: Orbital decay within one revolution of the Earth around the Sun

5 Future work

5.1 Evolutionary algorithm to optimize variable stage Butcher Tableau using DEAP

If the n-body simulation is carried out without any correction, the orbits quickly decay due to energy loss. This can be seen in a simple model of the Earth and the Sun, shown in Figure 4, where the orbit decays in less than one revolution. This is to be expected, as we are using a non-symplectic integrator to solve a system for which solutions exist on a symplectic manifold.

To try to programmatically reduce this error, we want to optimize the Butcher Tableau defining the RK scheme via evolutionary algorithms. Because we know that energy should be conserved, the change in total energy of the system should give us a cost function that we naturally want to minimize when evolving the Butcher Tableau.

The evolution of the Butcher Tableau will be done in Python using the DEAP (Distributed Evolutionary Algorithms in Python) framework. This is a framework developed "for rapid prototyping and testing of ideas," and to "make algorithms explicit and data structures transparent."[1] This framework provides a method to create individuals using user-defined data types, which is helpful for quickly generating a solution for evolving the tableaus.

Further, a common problem with evolutionary techniques and techniques in machine learning is that while they work exceptionally well in many cases, they can quickly become an intellectually unsatisfying black-box. The hope is that using a dedicated evolutionary algorithm framework will allow some degree of transparency into the actual optimization process.

The repository for the DEAP project can be found at https://github.com/DEAP/deap.

References

[1] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[2] Mark H. Holmes. *Introduction to Scientific Computing and Data Analysis*. Springer International Publishing, 2018.

[3] John R. Taylor. *Classical mechanics*. Univ. Science Books, 2005.

A Resources and specifications

The repository for this project can be found at https://github.com/jacksonlanecole/rkev. There is a detailed README located here that describes how the software is built and run.

B rkev Jupyter Notebook (2018-12-14 15:50)

rkev

- Exploring evolutionary approaches to Runge-Kutta (and discovering why symplectic approaches are better)
- Author : Jackson Cole (http://jacksoncole.io)
- Institute: Middle Tennessee State University
- Semester : Fall 2018

Description

- This notebook contains two primary items:
- Implementation of a nbody Python package that simplifies the setup of a full *n*-body simulation.
- **(12/14/18 01:25) IN PROGRESS** An implementation of a genetic algorithm to optimize a Butcher Tableau to reduce changes in energy over the course of the simulation.
- In both of these cases, a general purpose s-stage Runge-Kutta library written in C++ is also being tested. As of (12/14/18 01:25), this appears to be working. It is wrapped up in the integration methods of the **nbody** package.

Imports

In [1]:
```
import json
import math
import multiprocessing
import random
import time

from deap import algorithms, base, creator, tools
import matplotlib
import matplotlib.pyplot as plt
import numpy as np

import nbody as nb
```

In [2]:
```
# Matplotlib configuration
matplotlib.use('nbagg')
%matplotlib notebook
```

Setting up Butcher Tableaus

Here, I'm defining a dictionary of well known Runge-Kutta schemes. These can be used to show that the underlying methods work.

In [3]:
```
BT = {}
BT['RK4'] = [
    [   0.,   0.,   0.,   0.,   0.],
    [ 1/2., 1/2.,   0.,   0.,   0.],
    [ 1/2.,   0., 1/2.,   0.,   0.],
    [   1.,   0.,   0.,   1.,   0.],
    [   0., 1/6., 1/3., 1/3., 1/6.]
]

BT['RK38'] = [
    [   0.,   0.,   0.,   0.,   0.],
    [ 1/3., 1/3.,   0.,   0.,   0.],
    [ 2/3.,-1/3.,   1.,   0.,   0.],
    [   1.,   1.,  -1.,   1.,   0.],
    [   0., 1/8., 3/8., 3/8., 1/8.]
]

BT['zeros'] = [
    [ 0., 0., 0., 0., 0.],
    [ 0., 0., 0., 0., 0.],
    [ 0., 0., 0., 0., 0.],
    [ 0., 0., 0., 0., 0.],
    [ 0., 0., 0., 0., 0.]
]
```

Testing the rk_ext library with some simple first order ODEs

The last set of tests uses a Butcher Tableau of zeros to show that there are no hidden tableaus elsewhere in the library.

In [5]:
```
def test_ode(t, x):
    return x/(1 + t)

def test_analytical(t):
    return t + 1
```

In [6]:
```
RK4 = nb.rk_ext.RKIntegrator(test_ode, BT['RK4'], 10, 0, 2, 1, True)
solution = RK4.run()
error = ((solution - test_analytical(2))/test_analytical(2))*100
print("solution = {}".format(solution))
print("error    = {}".format(error))
```

```
solution = 3.0000000000000004
error    = 1.4802973661668752e-14
```

In [7]:
```
RK38 = nb.rk_ext.RKIntegrator(test_ode, BT['RK38'], 10, 0, 2, 1, True)
solution = RK38.run()
error = ((solution - test_analytical(2))/test_analytical(2))*100
print("solution = {}".format(solution))
print("error    = {}".format(error))
```

```
solution = 3.0000000000000004
error    = 1.4802973661668752e-14
```

In [8]:
```
RKzero = nb.rk_ext.RKIntegrator(test_ode, BT['zeros'], 10, 0, 2, 1, Tr
ue)
solution = RKzero.run()
error = ((solution - test_analytical(2))/test_analytical(2))*100
print("solution = {}".format(solution))
print("error    = {}".format(error))
```

```
solution = 1.0
error    = -66.66666666666666
```

Implementation (no GA)

NOTES

- (12/14/18 00:52) The initial implementation of the nbody code was carried out step by step for readability in the Jupyter Notebook, but at this point, I am moving all of that code into a wrapper function that will be used as an evaluation function for DEAP.

Setting up bodies

In [9]:
```
sun = nb.Body(mass=1.988e30,
              position=[0, 0, 0],
              velocity=[0, 0, 0],
              name='sun',
             )

earth = nb.Body(mass=5.972e24,
              position=[149597870700, 0, 0],
              velocity=[0, 29.8e3*3600, 0],
              name='earth',
             )

#mars = nb.Body(mass=0.64171e24,
#              position=[227.92e9, 0, 0],
#              velocity=[0, 24.07e3*3600*24, 0],
#              name='earth',
#             )
#
#jupiter = nb.Body(mass=1898.19e24,
#              position=[778.57e9, 0, 0],
#              velocity=[0, 13.06e3*3600*24, 0],
#              name='earth',
#             )
bodies = [sun, earth]
```

The below is a time dictionary where the first key is the unit, and the second key is a common value.

In [10]:
```
times = {
    'second':{
        'second':1,
        'minute':60,
        'hour':3600,
        'day':24*3600,
        'year':365.25*24*3600,
    },
    'minute':{
        'second':1/60.,
        'minute':1.,
        'hour':60,
        'day':24*60,
        'year':365.25*24*60,
    },
    'hour':{
        'second':1/3600.,
        'minute':1/60.,
        'hour':1,
        'day':24,
        'year':365.25*24,
    },
    'day':{
        'second':1/(24*3600),
        'minute':1/(24*60.),
        'hour':1/24.,
        'day':1,
        'year':365.25,
    }
}
```

Plotting Function

In [11]:
```
def plot(system, corrected, window_size, time_dict):
    t_init = time_dict['t_init']
    t_fin = time_dict['t_fin']
    steps = time_dict['steps']
    if steps == int(1e4):
        bound = list(range(steps+2))
    elif steps == int(1e5):
        bound = list(range(steps+1))

    AU = 149597870700
    fig = plt.figure()
    ax1 = fig.add_subplot(111)
    ax1.scatter(system.data['x'][0], system.data['y'][0], color='orang
e', s=100)
    for i in range(1, len(system.body_list)):
        ax1.scatter(system.data['x'][i],
                 system.data['y'][i],
                 c=bound, cmap="inferno", s=0.2)

    ax1.set_xlim([-window_size*AU,window_size*AU])
    ax1.set_ylim([-window_size*AU,window_size*AU])
    #cax = ax1.imshow()
    #cbar = fig.colorbar(cax)

    plt.title('Orbit of Bodies Around Sun, {} energy loss correction'.
format(corrected))
    plt.savefig("decaying_orbits.png", dpi=300)
    plt.show()

    E = system.energy
    dE = [((e - E[0])/abs(E[0]))*100 for e in E]

    fig = plt.figure()
    #ax1 = fig.add_subplot(111)
    plt.scatter(range(len(dE)), dE, c=bound, cmap="inferno", s=0.2)
    plt.colorbar()
    plt.title('Percent Change in Energy from Initial')
    plt.ylabel('Percentage of Initial Energy')
    plt.xlabel('Iteration')
    plt.tight_layout()
    plt.show()
```

run definition

In [12]:
```
def run(bt, bodies, time_dict):
    t_init = time_dict['t_init']
    t_fin = time_dict['t_fin']
    steps = time_dict['steps']

    system = nb.System(bodies)
    system.setup_integrators(bt, t_init, t_fin, steps)

    start = time.process_time()

    system.run()

    end = time.process_time()
    elapsed = end - start

    print("time elapsed = " + str(elapsed))
    print("-----------------------------------------------------")
    print("TOTAL ENERGY CHANGE")
    print("-----------------------------------------------------")
    print("absolute   = {:15.5e}".format(system.energy_change['absolut
e']))
    print("percentage = {:15.5e}".format(system.energy_change['percent
age']))

    return system
```

In [13]:
```
times['second']['year']
```

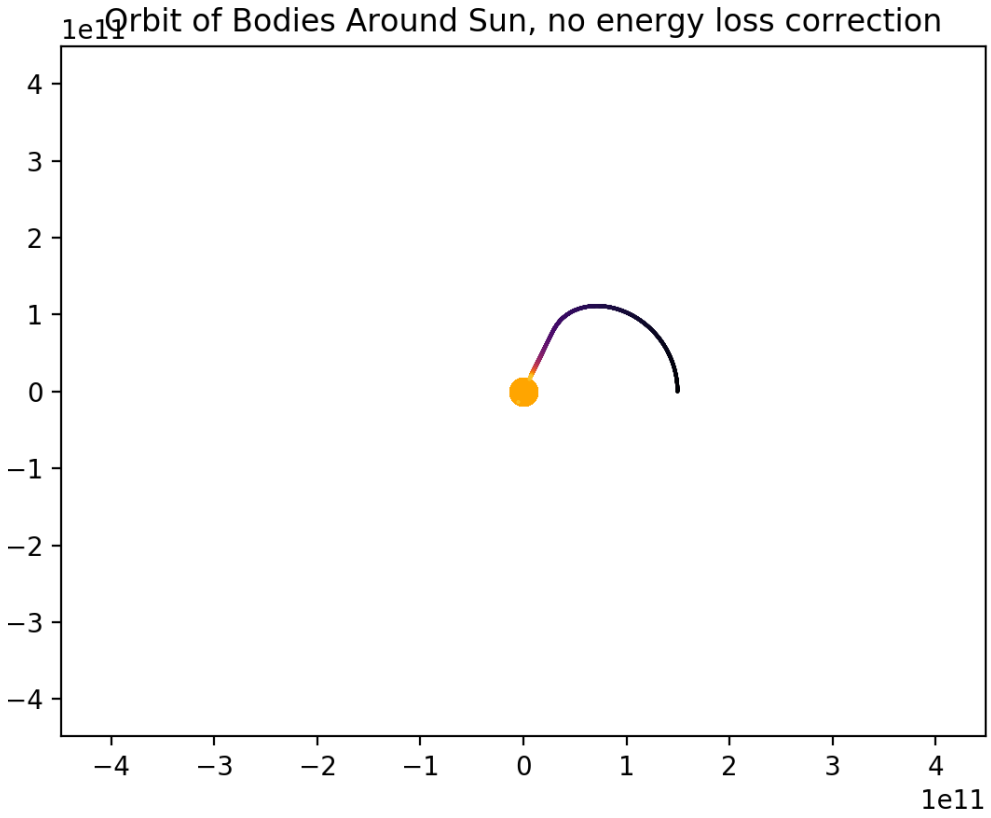
Out[13]:
```
31557600.0
```

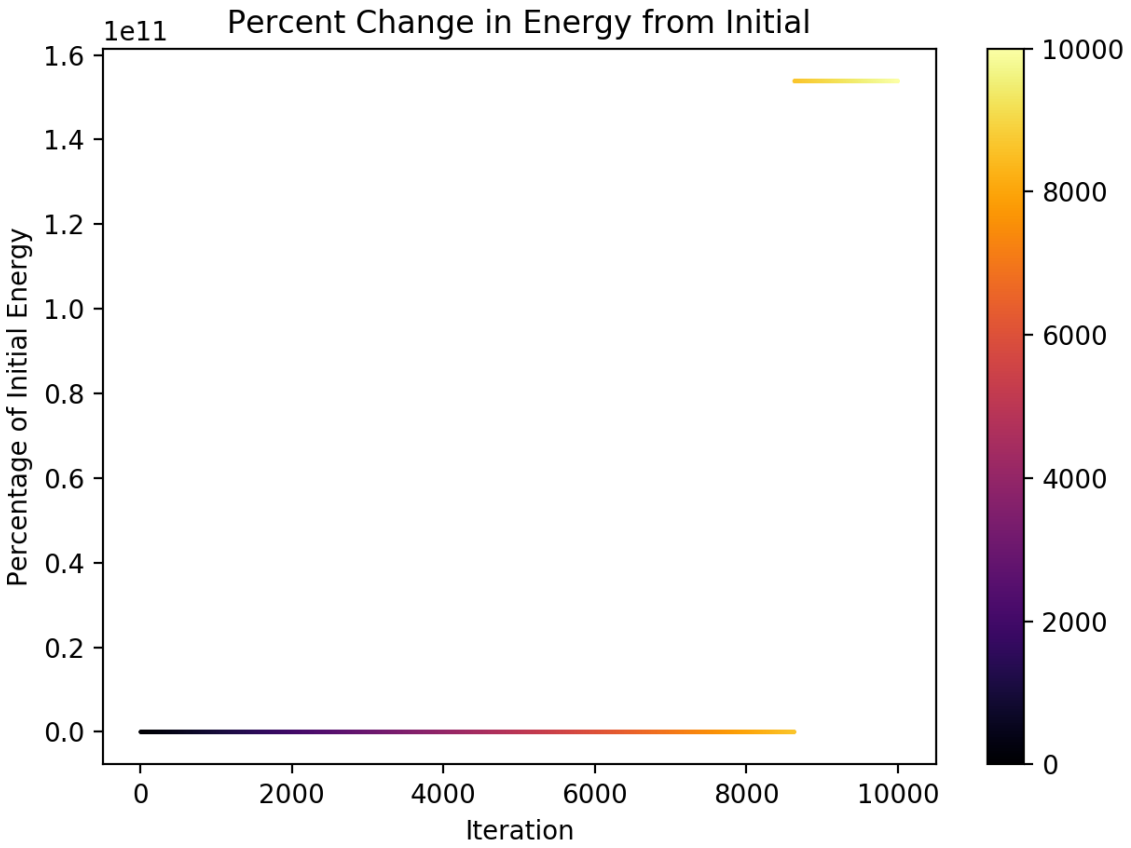
In [15]:

```
time_dict = {
    't_init':0.,
    't_fin':1.5*times['hour']['year'],
    'steps':int(1e4),
}

window_size = 3
system = run(BT['RK4'], bodies, time_dict)
plot(system, 'no', window_size, time_dict)
```

```
time elapsed = 1.404484
-----------------------------------------------------
TOTAL ENERGY CHANGE
-----------------------------------------------------
absolute   =     5.27282e+49
percentage =     1.53821e+11
```





In [ ]:
```
#print(system.data['vx'][0][:5])
#print(system.data['vy'][0][:5])
#print(system.data['vz'][0][:5])
#print(system.data ['x'][0][:5])
#print(system.data ['y'][0][:5])
#print(system.data ['z'][0][:5])
#print(system.data['ac'][0][-5:])
```

Moving on to GA using DEAP

The standard Runge-Kutta method is undoubtedly *not* the best method for these types of problems, and we could easily arrive at a solution that conserves energy by using a symplectic integrator... but that's not what we're going to do.

In this case, we're going to attempt to use DEAP to evolve the Butcher Tableau to minimize the overall change in energy. Basically, we've got a giant, multipurpose hammer (non-symplectic Runge-Kutta), so we just need to make our problem look like a nail.

In [32]:
```
def evaluate(individual, time_dict):
    """This is the function that will be used to evaluate
    the individuals. To be used for other systems, all of
    the definitions need to either be made here or the
    argument list must be augmented to contend with extra
    parameters.
    """
    t_init = time_dict['t_init']
    t_fin = time_dict['t_fin']
    steps = time_dict['steps']

    # Initialize the system (notice the usage of "individual")
    system = nb.System(bodies)
    #print("HERE IT IS: {}".format(type(individual)))
    system.setup_integrators(individual, t_init, t_fin, steps)

    # Run
    start = time.process_time()

    system.run()

    end = time.process_time()
    elapsed = end - start

    #print("time: {}".format(elapsed))

    return (system.energy_change['percentage'],)

# TEST: Make sure the above function actually works
energy_change = evaluate(BT["RK4"], time_dict)
print("energy change = {}".format(energy_change))
```

```
energy change = (-1018.5356785671061,)
```

In [21]:
```
creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)
```

In [33]:
```
bts = list(BT.values())
def init_tableau(icls):
    bt = bts[random.randint(0, 1)]
    bt = np.asarray(bt)*random.random()
    bt = bt.tolist()
    return icls(bt)
```

In [ ]:
```
toolbox = base.Toolbox()
toolbox.register("init_individual", init_tableau)
toolbox.register("individual", toolbox.init_individual, creator.Indivi
dual)
toolbox.register("population", tools.initRepeat, list, toolbox.individ
ual)

toolbox.register("evaluate", evaluate, time_dict=time_dict)
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutGaussian, mu=0, sigma=0.3, indpb=
0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

pool = multiprocessing.Pool()
toolbox.register("map", pool.map)

pop = toolbox.population(n=50)
hof = tools.HallOfFame(1)
stats = tools.Statistics(lambda ind: ind.fitness.values)
stats.register("avg", np.mean)
stats.register("std", np.std)
stats.register("min", np.min)
stats.register("max", np.max)
```

In [ ]:
```
population = toolbox.population(n=20)
NGEN=15
for gen in range(NGEN):
    offspring = algorithms.varAnd(population, toolbox, cxpb=0.7, mutpb
=0)
    fits = toolbox.map(toolbox.evaluate, offspring)
    for fit, ind in zip(fits, offspring):
        ind.fitness.values = fit
    population = toolbox.select(offspring, k=len(population))
top10 = tools.selBest(population, k=10)
```

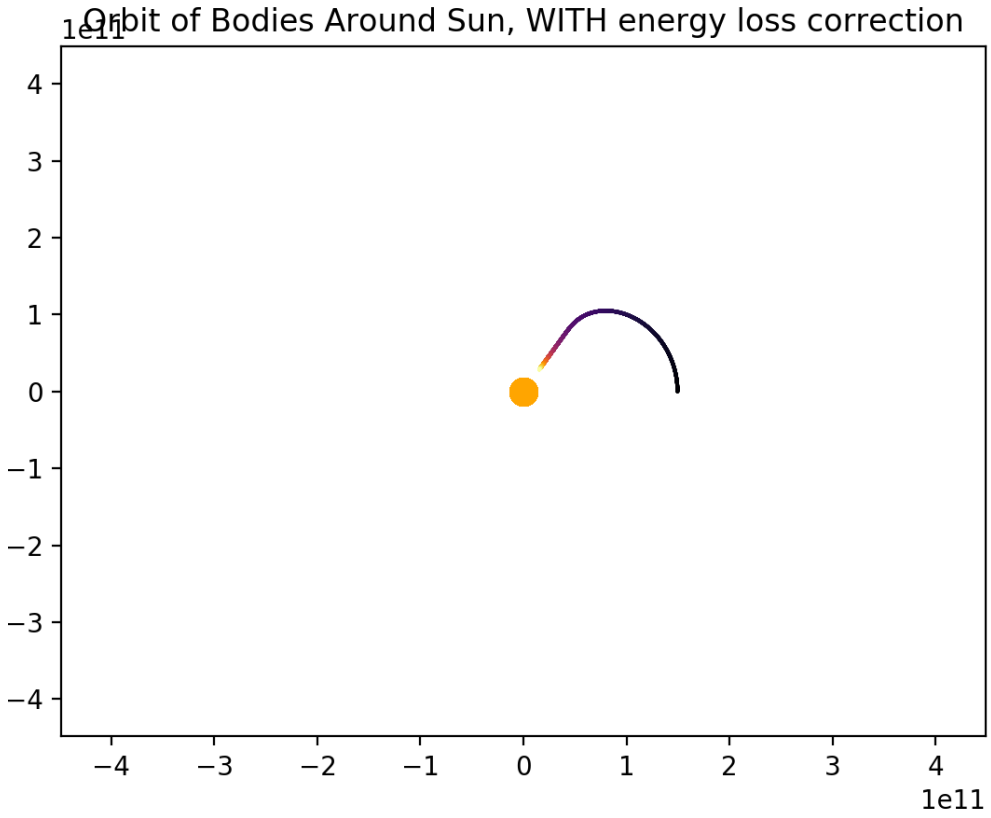
In [19]:
```
print('Best Remaining Individual after {} generations:'.format(NGEN))
a = [[0.0, 0.0, 0.0, 0.0, 0.0],
 [0.3636733144010224, 0.3636733144010224, 0.0, 0.0, 0.0],
 [0.3636733144010224, 0.0, 0.3636733144010224, 0.0, 0.0],
 [0.7677566283239672,
  0.7677566283239672,
  -0.7677566283239672,
  0.7677566283239672,
  0.0],
 [0.0,
  0.0959695785404959,
  0.28790873562148767,
  0.28790873562148767,
  0.0959695785404959]]
print('[')
for row in a:
    print('    ' + str(row) + ',')
print(']')
#print(ind.fitness.values)
```

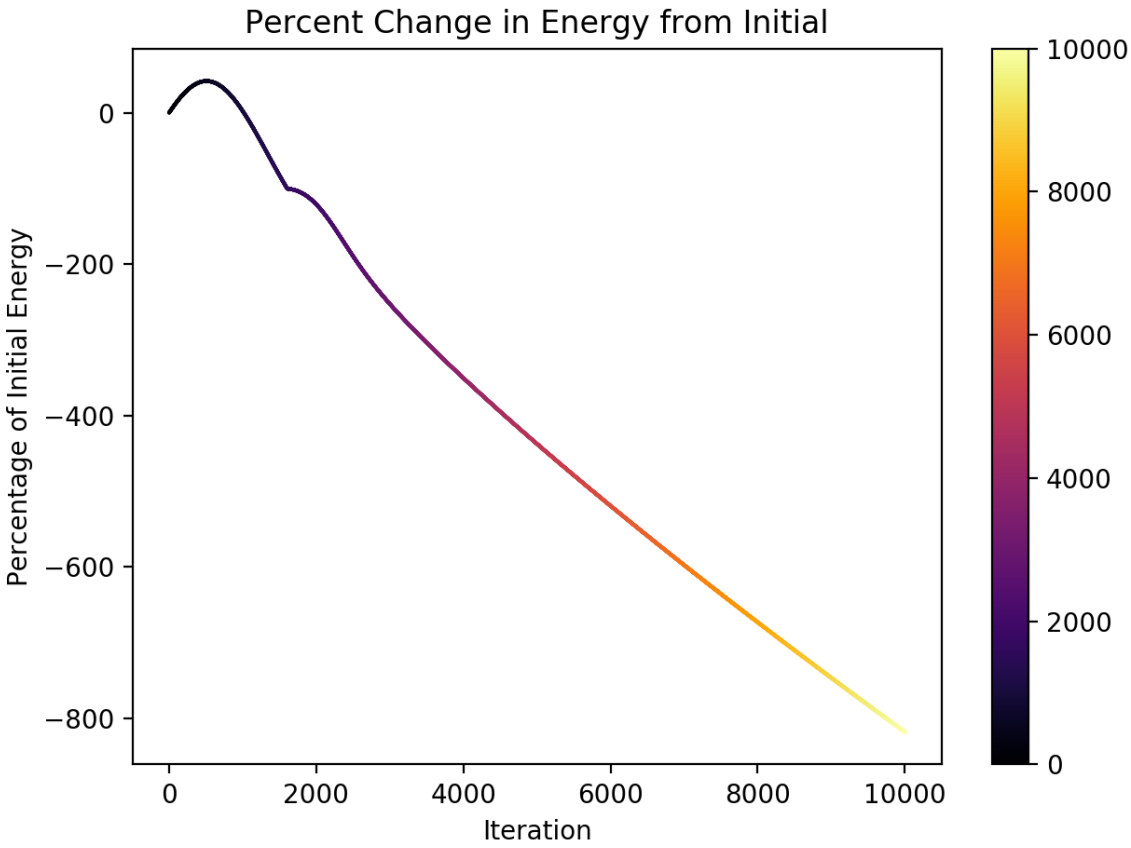
```
Best Remaining Individual after 15 generations:
[
    [0.0, 0.0, 0.0, 0.0, 0.0],
    [0.3636733144010224, 0.3636733144010224, 0.0, 0.0, 0.0],
    [0.3636733144010224, 0.0, 0.3636733144010224, 0.0, 0.0],
    [0.7677566283239672, 0.7677566283239672, -0.7677566283239672, 0.76
77566283239672, 0.0],
    [0.0, 0.0959695785404959, 0.28790873562148767, 0.2879087356214876
7, 0.0959695785404959],
]
```

In [16]:
```
time_dict = {
    't_init':0.,
    't_fin':1.5*times['hour']['year'],
    'steps':int(1e4),
}

window_size = 3
system = run(a, bodies, time_dict)
plot(system, 'WITH', window_size, time_dict)
```

```
time elapsed = 1.3868929999999988
-----------------------------------------------------
TOTAL ENERGY CHANGE
-----------------------------------------------------
absolute   =    -2.80393e+41
percentage =    -8.17977e+02
```





Resources

- https://thesesergio.wordpress.com/2013/05/31/deap-a-self-made-tutorial-12/comment-page-1/#comment-98 (https://thesesergio.wordpress.com/2013/05/31/deap-a-self-made-tutorial-12/comment-page-1/#comment-98)