Double Pendulum Simulation

By Charlie Seymour

Contents

Analysis

Background to problem

The Kingswood School Mathematics Department provides lots of support and additional teaching for able students. There is a set system throughout the senior school, which allows the stronger Maths students to be taught topics or ideas that may not appear in the standard curriculum. The GCSE set do their Maths GCSE earlier than the lower sets, and then spend the rest of the year preparing for an Additional Mathematics qualification that only they take. In the Sixth form, there is always a separate Further Maths set for both AS and A Level.

These students are encouraged to attend a weekly Further Maths Enrichment activity in school, where the students learn about different aspects of Maths that will broaden their knowledge and may be useful for applying for Maths based degrees at University. Further Maths Enrichment is run by Mr Chua.

Mr Chua wishes to teach Further Maths Enrichment about Dynamic Systems and Chaos Theory. He wants to show the students this in action with a demonstration of a double pendulum. He also wants to show the students what the graphs relating to the double pendulum look like and how they change depending on initial conditions, and to use them as a visualisation of Lissajous curves.

Description of current system

Mr Chua has not yet found a satisfactory means of teaching students about the double pendulum. (See Potential Solutions)

Identification of prospective user(s)

The solution will be to Mr Chua's specification so he is the main prospective user. However it is very likely that he will want his students to be able to use the simulation as well, so I need to take this into account when implementing the final solution.

Objectives

This is a transcript of the initial interview I had with Mr Chua, in which he outlines his desires and we discuss the limitations of the project.

Key: CS = Charlie Seymour (me), MC = Mr. Chua

CS: Hello Sir, just to outline the brief conversation before we begin the interview, as you know I will be doing a practical project for my A2 Computing, with the intention of solving a real problem. You

said that you are interested in having a maths/physics simulation made for teaching?
MC: Yes, as it currently stands, I have tried to teach the Further Maths enrichment activity about Chaos theory and dynamic systems. Do you know anything about chaos theory already?
CS: A little, I know that it is something to do with things that vary a lot from very small changes in initial conditions.
MC: Exactly, that's the basic essence of it, and it has applications in a variety of things, for example weather patterns; chaos theory explains why they are so hard to predict for more than a few days. One of the simplest systems that displays chaotic behaviour is the double pendulum. I would like to be able to teach students about chaos theory using a double pendulum as a system that demonstrates it. A real double pendulum is expensive and impractical, as it has friction and therefore energy loss, plus the students would be unable to all be using one at the same time.
CS: So you thought software would be more appropriate.
MC: Exactly. There are simulations available on the internet but none of them perform exactly as desired.
CS: Right. Could you take me through the things that you would want present in the simulation if I were to write one?
MC: What I need is a program that allows the user to enter the starting conditions of the double pendulum. These are the starting angle of each arm, the angular velocity of each arm, the length of each arm, and the mass of each head. It would also be quite good if the user could change how strong gravity was, and maybe turn it off altogether, although those features aren't essential.
CS: Right, what sort of range should the user's inputs be? For example if the mass of one of the heads was 1 unit and the other was a million units, that could potentially break the simulation.
MC: Preferably the users shouldn't be able to enter values that could cause the system to behave erratically, which happens if the numbers get too large.
CS: Perhaps the best input method would be to have sliders with a fixed range? That would prevent extreme values being entered and would make the program easier to use.
MC: Definitely a good idea, the program needs to be as easy to use as possible. So once the user has entered the starting conditions for the pendulum, they should be able to set the graph's axis. These should be the angle of either pendulum and the angular velocity of either pendulum. When they start the simulation, the pendulum should move as a double pendulum would move and it should do so at the same speed one in real life would move, as in "real-time". As the pendulum is moving, the graph should update at the same time.
CS: Okay. The strain on the computer will vary depending on how fast the computer is, lowering the accuracy of the simulation should make up for this if it is being run on a slower computer, so should I let users have the option of changing the accuracy?
MC: If that is the case then yes that would be a good option to have. I also want the pendulum to leave a trail of where it has been – the second head, that is – to show the students that if you change the starting conditions slightly the path will soon diverge completely.
CS: Okay that sounds fine. Should the users be able to control the flow of time, so to speak? As in pause, rewind fast forward etc.
MC: They should definitely be able to pause the simulation, and they should also be able to step through the simulation slowly so they can analyse certain bits of movement in detail. Fast-forwarding and rewinding shouldn't be needed; you could add them as an extra if they didn't affect the usability of the program. However I do want to be able to save the pendulum's current conditions, and then load back to that point.

CS: Right, fast-forwarding would probably be detrimental to the system, as you would have to seriously lower the accuracy in order for it to be able to process it quick enough, as the simulation should be running at the highest accuracy possible without it lagging when it is moving normally. Rewinding could be implemented but it would either require a whole new implementation of the simulation that runs in reverse, or it would require a "history" of the pendulum's movement to be constantly saved and updated, which may slow the system down.
MC: That's fine then, don't implement fast forwarding or rewinding as they aren't important for what I need the program for. Will the save/load be okay then?
CS: Yes, those two should be fine. What else do you need the program to be able to do?
MC: Ideally it should be able to save the graph and the pendulum's trail onto the computer's hard drive as image files.
CS: That should be fine.
MC: Also, it would be good if there was a sort of quick-reference help file that the users could bring up in the program to show them how to use it and what the controls are.
CS: Alright, and should there be sources that teach the students about Chaos theory in the program as well? As you said that this was going to be used as a teaching tool.
MC: No, there is no need for there to be any specific teaching material inside the program. The reason is because they [the students] will all be using it for the first time at the enrichment activity, where I will be teaching them about chaos theory whilst they use it. All the program needs to have in the help section is an explanation of the controls and a perhaps a brief explanation of the parts of the double pendulum. This will mainly serve as a reminder for the students if they choose to use the program after the activity or in the future, as I will teach them about the double pendulum before they use the simulation.
CS: Okay, that's fine. Is there anything else?
MC: I don't think so, that should be everything. Will you be able to write the program?
CS: I should be able to, yes!
MC: Thank you very much.
CS: I'll keep you updated!

Here is a refined list of the objectives obtained from the interview:

1. The program should allow the user to set the following initial attributes:
    a. Starting angle of both pendulums
    b. Starting angular velocity of both pendulums
    c. Starting mass of each head
    d. Starting length of each head
    e. Gravitational field strength
    f. Accuracy of simulation
2. The user should be limited in their entry of the initial attributes so that the pendulum does not endure extreme conditions
3. The simulation should move in real time
4. The simulation should have a graph that has customisable axis variables
5. The graph should update at the same time as the pendulum
6. The second pendulum head should draw a traced path of where it has been during the simulation
7. The users should be able to play and pause the simulation

8. The users should be able to step through the simulation
9. The pendulum's current state can be saved to and loaded from the computer's hard drive
10. The user should be able to save the images of the graph and the pendulum's trail to disk
11. The program should have a brief help section that explains to the user how to use the simulation
12. The simulation should be easy to use and user friendly.

Here is a screenshot of the email confirming the interview's content

Computing project interview transcript

Mr J Chua

Thursday, October 31, 2013 8:13 PM

Yes, that is what I said. Nothing has been omitted.

Mr Chua

Charlie Seymour

Actions

To: Mr J Chua

Sent Items

Thursday, October 31, 2013 7:16 PM

Dear Mr Chua,

Here is a transcript of the discussion we had with regards to my Computing project. Please could you confirm that this is what you said and that everything has been recorded.

Thanks,

Charlie

CS: Hello Sir, just to outline the brief conversation before we begin the interview, as you know I will be doing a practical project for my A2 Computing, with the intention of solving a real problem. You said that you are interested in having a maths/physics simulation made for teaching?
MC: Yes, as it currently stands, I have tried to teach the Further Maths enrichment activity about Chaos theory and dynamic systems. Do you know anything about chaos theory already?
CS: A little, I know that it is something to do with things that vary a lot from very small changes in initial conditions.
MC: Exactly, that's the basic essence of it, and it has applications in a variety of things, for example weather patterns; chaos theory explains why they are so hard to predict for more than a few days. One of the simplest systems that displays chaotic behaviour is the double pendulum. I would like to be able to teach students about chaos theory using a double pendulum as a system that demonstrates it. A real double pendulum is expensive and impractical, as it has friction and therefore energy loss, plus the students would be unable to all be using one at the same time.
CS: So you thought software would be more appropriate.
MC: Exactly. There are simulations available on the internet but none of them perform exactly as desired.
CS: Right. Could you take me through the things that you would want present in the simulation if I were to write one?
MC: What I need is a program that allows the user to enter the starting conditions of the double pendulum. These are the starting angle of each arm, the angular velocity of each arm, the length of each arm, and the mass of each head. It would also be quite good if the user could change how strong gravity was, and maybe turn it off altogether, although those features aren't essential.
CS: Right, what sort of range should the user's inputs be? For example if the mass of one of the heads was 1 unit and the other was a million units, that could potentially break the simulation.
MC: Preferably the users shouldn't be able to enter values that could cause the system to behave erratically, which happens if the numbers get too large.
CS: Perhaps the best input method would be to have sliders with a fixed range? That would prevent extreme values being entered and would make the program easier to use.
MC: Definitely a good idea, the program needs to be as easy to use as possible. So once the user has entered the starting conditions for the pendulum, they should be able to set the graph's axis. These should be the angle of either pendulum and the angular velocity of either pendulum. When they start the simulation, the pendulum should move as a double pendulum would move and it should do so at the same speed one in real life would move, as in "real-time". As the pendulum is moving, the graph should update at the same time.
CS: Okay. The strain on the computer will vary depending on how fast the computer is, lowering the accuracy of the simulation should make up for this if it is being run on a slower computer, so should I let users have the option of changing the accuracy?
MC: If that is the case then yes that would be a good option to have. I also want the pendulum to leave a trail of where it has been – the second head, that is – to show the students that if you change the starting conditions slightly the path will soon diverge completely.
CS: Okay that sounds fine. Should the users be able to control the flow of time, go to sneak? As in pause, rewind fast forward etc

Analysis Data Dictionary

Here are the terms that I will be using in this document:

- **Double pendulum**
  - This is a pendulum on a fixed **pivot** with a second pendulum attached to its end.
  - This will be used to refer to both the physical system as well as the software implementation. The context will clarify which is being used.
- **Graph**
  - This is a mathematical graph that consists of a coordinate system and two defined **axes**. Each **axis** has a **variable** that it describes
- **Axis**
  - This is either the vertical or horizontal line on the graph that is used as a sort of ruler to gauge the current size of the **variable** that the axis is describing
- **Variable (graph)**
  - In this context, a variable is a property of the double pendulum that an axis on the graph will measure.
- **Radians**
  - This is a way of measuring angles, where the angle in a circle (i.e. one full rotation) is 2π. For an explanation on radians please visit http://goo.gl/k5AKh6.
- **Angular velocity**
  - This is how quickly something is turning, in this context it is the **arms** of the **double pendulum**. It is the first **derivative** with respect to time of the angle
- **Angular acceleration**
  - This is the rate at which the **angular velocity** is changing. It is the first **derivative** with respect to time of **angular velocity**, and is therefore the second **derivative** with respect to time of the angle.
- **Pendulum**
  - This refers to one of the two **arm**-**head** pairs making up the **double pendulum**, i.e. the first **arm** and the first **head** make up one of the two, and the latter is the second **arm** and second **head**. I will often write "for each pendulum"; meaning for each of the two **arm**-**head** pairs. Sometimes the whole **double pendulum** may be referred to as the **pendulum**; the intended meaning will be clear in context.
- **Pivot**
  - This is the "centre" of the **double pendulum**; it is fixed in space and allows the first **arm** to freely rotate around it.
- **Arm**
  - Massless rods that connect the **heads** of the **double pendulum**. They are rigid, and don't change length (after their length is defined that is). They can be considered to have length but no width.
- **Head**

    - These are point masses that are on the end of each of the pendulum's **arms**. They can be considered to take up a single point in space, and therefore have no dimensions.
- **Chaos theory**
    - A branch of mathematics that involves studying systems that are highly sensitive to initial conditions, i.e. a small change in initial conditions will result in completely different results as time goes on (divergence)
- **Differentiation**
    - This is a mathematical (calculus) technique that finds how something changes depending on something else changing. For example, in this context, we can say that the **angular velocity** is how the angle changes as time changes. This means we can say that **angular velocity** is the *derivative* of the angle *with respect* to time.

Data volumes

Overall, the program will be quite small. I can't foreseeably see there being any need for external files. Judging by the objectives laid out by Mr Chua, the only files that the program would save are images (~500KB) and the saved states of the pendulum, which should be very small (< <1KB). I would predict that the program itself will be quite small; somewhere between 20KB and 80KB. All files will be saved in the same directory as the program is being run in, so all data will be stored on the local hard drive.

Overall, the program should have a very small impact on the computer it is being run on in terms of storage space.

Identification of user needs and acceptable limitations

I have interviewed Mr Chua about what he wants the solution to be able to do, and so we could discuss acceptable limitations and how feasible his needs are for the system.

Mr Chua wants a simulation of a double pendulum that he can use to teach students about dynamic systems and chaos theory. He wants a simulation that you can easily enter starting condition into, and he wants the program to simultaneously draw graphs that the user can set the axis to draw the angle of either pendulum and the angular velocity of either pendulum. He also wants the simulations to be able to "trace" the head of the second pendulum; that is for the second pendulum to leave a trail behind it to see where it has travelled.

Mr Chua would also like the facility to be able to easily save the graphs at any time, to pause, play, and save and load the simulation, as well as being able to save to an image file the path traced by the second pendulum head.

The conditions that Mr Chua wants to be able to initialise are the starting angles of the two pendulums, the (relative) mass of each of the pendulum heads (the rods are massless), the length of each rod, and the initial angular velocity of each pendulum head. Mr Chua also said that it would be possible he may want to be able to adjust the strength of gravity, and maybe turn it off completely, so this will have to be taken into account.

We also agreed that another acceptable limitation would be that the number of pendulums would remain as two, venturing into triple pendulums and beyond presents a number of problems. Firstly, if the graphs were still to be drawn still using one path (and simultaneously displaying attributes of all three pendulums) they would have to be 3D, and for a quadruple pendulum and higher it wouldn't be possible to draw clear graphs with only a single path as mankind has yet to find a fourth spatial dimension!

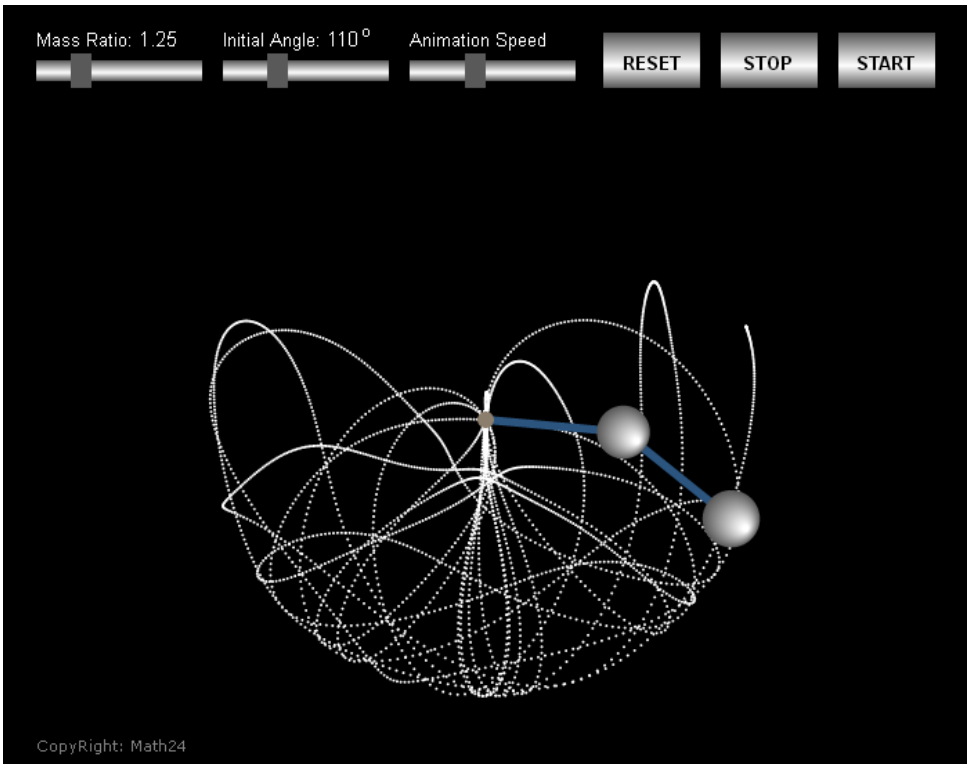
The other complication of more than two pendulums is that the maths involved becomes much longer and complicated, and would really start to bog down the design process.

These limitations were fine with Mr Chua, as a double pendulum is sufficient for his needs, and little would be gained by having more than two, especially with comparison to the additional time required to implement it.

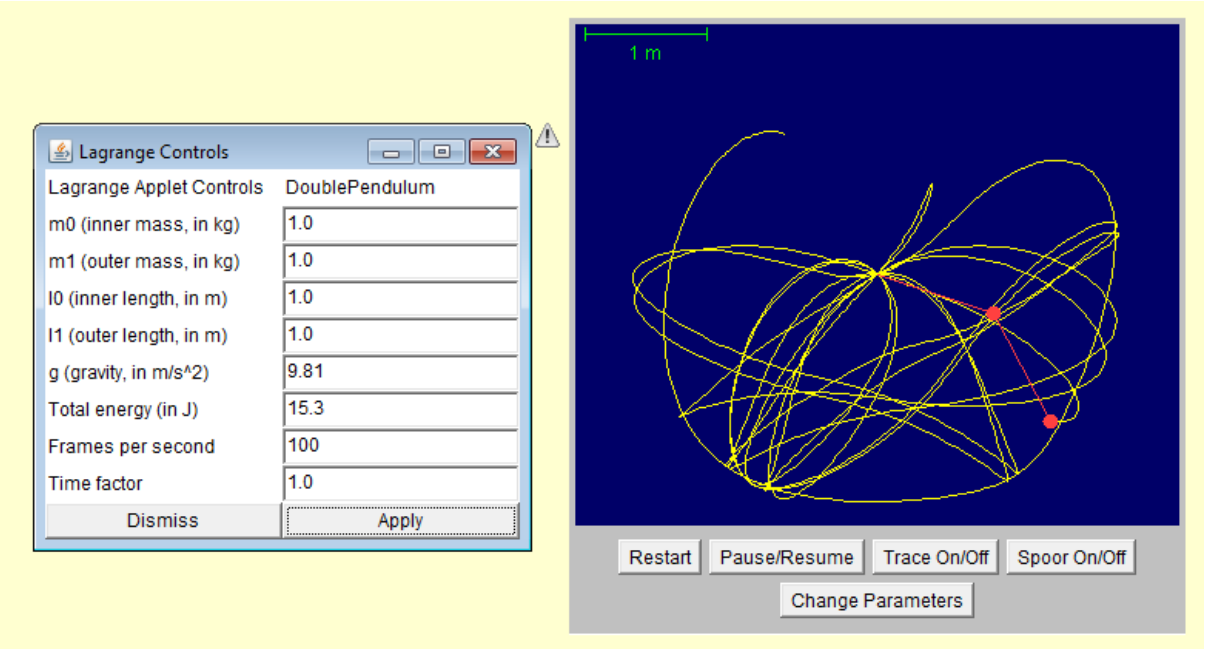
[Note that this was discussed (briefly) on a separate occasion to the interview in the Objectives section and so is not present in the transcript]

Potential solutions

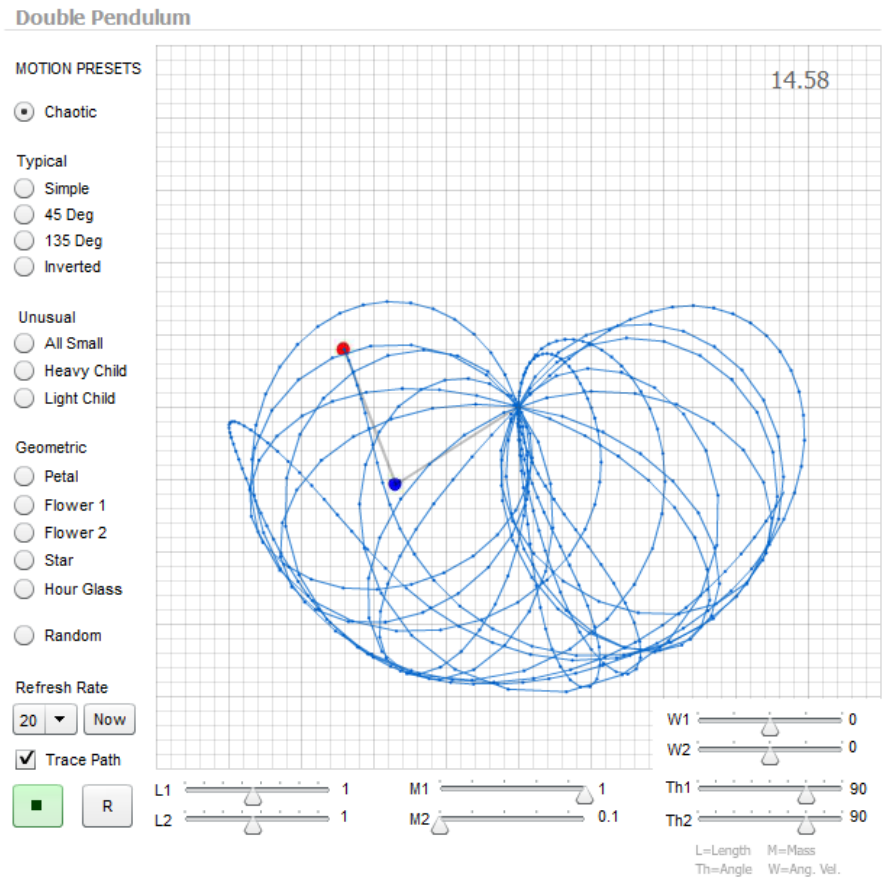
There are double pendulum simulations available on the internet. However, each one tends to lack something that Mr Chua needs. Here is a selected sample demonstrating their insufficiencies.



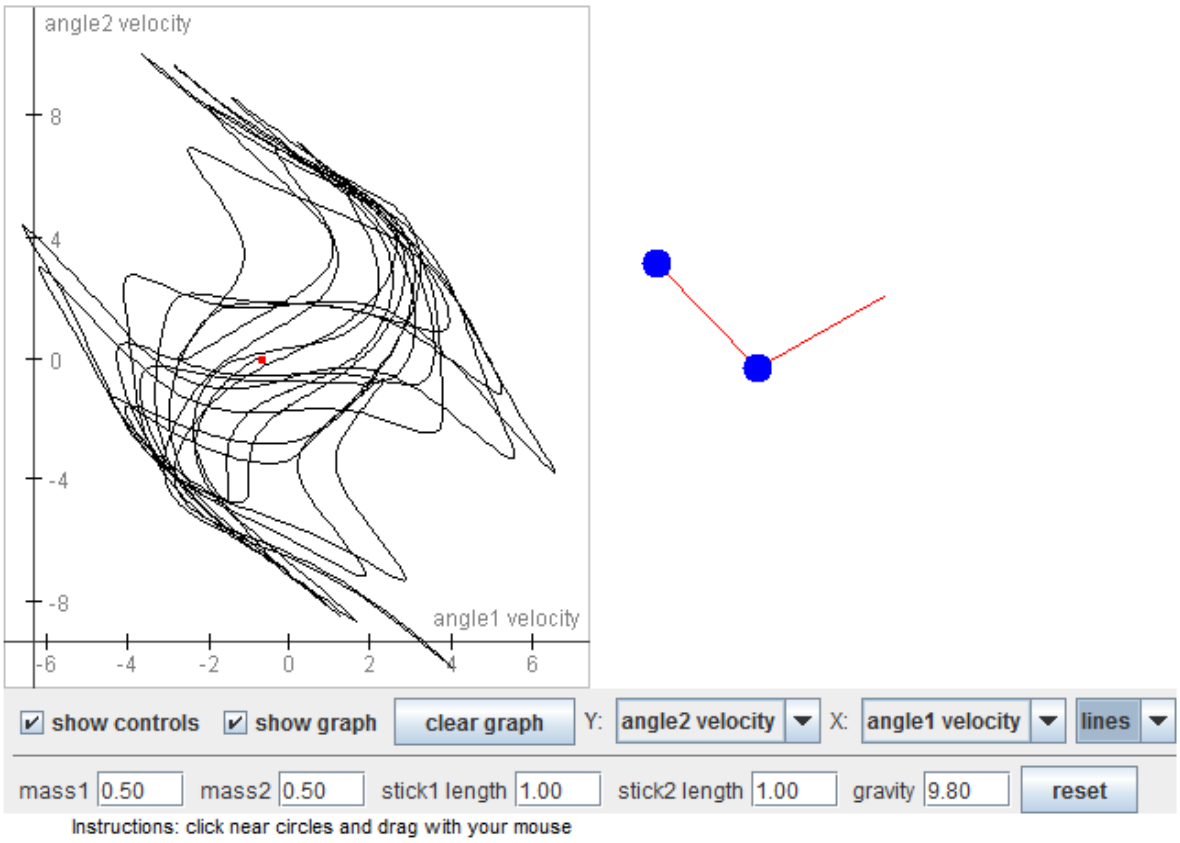
This simulation can be found at http://goo.gl/j7hIbO. Its main issue is the very limited controls; the only things that can be changed are the mass ratio, the animation "speed", and the starting angle of both arms (i.e. they cannot be set to start at different angles). Another problem is that the pendulum heads are excessively large and somewhat block the display.



This simulation is an improvement over the last, because it gives you greater control over the attributes of the pendulum. However, the text boxes mean that if the user enters extreme numbers, it is possible to break the simulation. The frames per second and time factor control elements are not clear in their meaning, and the pendulum is often either to slow or jerky as a result. Also, it has no graph drawing capabilities. It can be found at http://goo.gl/fR84mA.



This simulation is better than the last one due to its use of sliders to enter user information. This means that it is more difficult to enter values that will break the pendulum, however it is still possible. If it happens, the simulation shows a message saying “UNSTABLE Strart Again” (The spelling error is a direct quote!), however Mr Chua would prefer a simulation that sufficiently limited the inputs so that it wouldn’t be possible to create such a situation. The traced path resets after a set period of time, and whilst you can alter how long it is, you cannot turn this off completely. Also, the simulation doesn’t provide any graph drawing capabilities, and the large gaps between the drawn points of the traced path suggest a low degree of accuracy. It can be found at http://goo.gl/4bXbHr.



This simulation is probably the best one out of those available, due to the fact it is the only one that draws live graphs as the pendulum moves. However, the graph automatically scales as the simulation goes on, which can often lead to distortion in the image. Also, the user can enter simulation breaking values; and rather than stopping the simulation with an “Unstable” sign, the pendulum flies off the screen and stays there. It can be found at http://goo.gl/CLj9H.

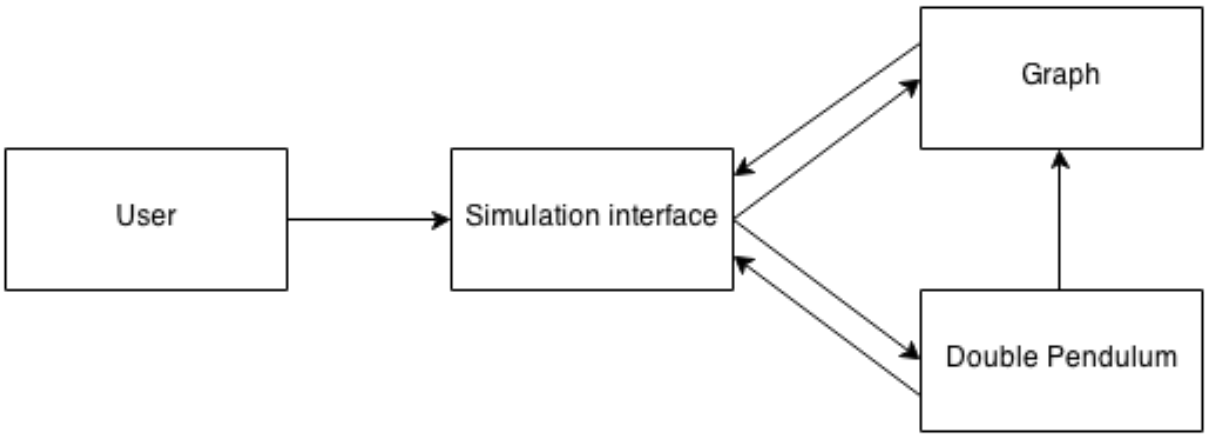
As well as all the issues mentioned above, none of the available simulations provide the option to export images of the traced path or of the drawn graph.

Another alternative would be for Mr Chua to buy a double pendulum, which would allow students to interact with a dynamic system with their own hands. However, professionally engineered double pendulums cost over £100, and obviously this would provide no graph drawing capabilities. Also, Mr Chua wants a system that has no energy loss, and the friction in a real world pendulum means that it is not an appropriate solution.

Justification of chosen solutions

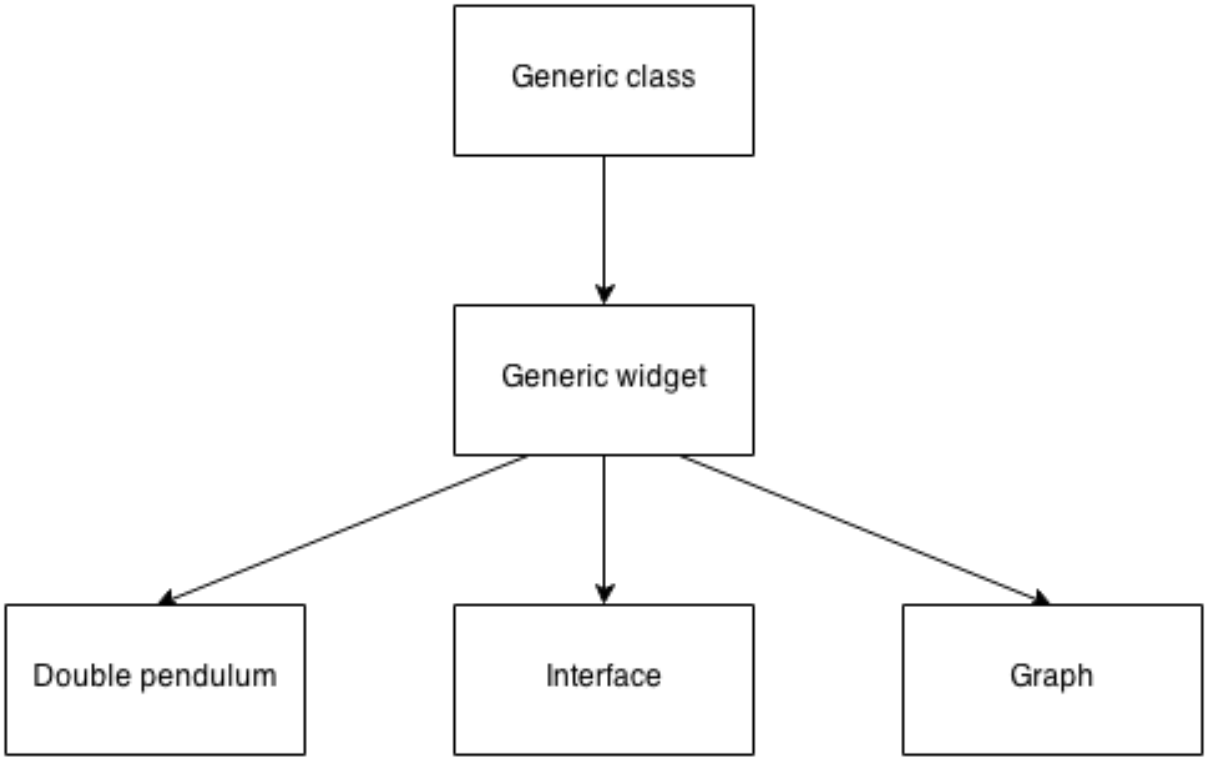
Despite there being several double pendulum simulations available on the internet, none of them provide everything Mr Chua wants in one simulation, as shown in the previous section. As purchasing a double pendulum is financially unfeasible and doesn’t have the ability to draw graphs, a bespoke piece of software is the most appropriate solution for Mr Chua’s problem.

Data source(s) and destination(s)



The user will set the initial attributes through the simulation's interface. This will send the appropriate information to the graph and to the double pendulum. As the simulation runs, the double pendulum will be sending information to the interface so that it knows what to display, as will the graph. The double pendulum will also be sending information to the graph, as the graph will be plotting the pendulum's current attributes, so it will need to receive them as they change.

Object analysis diagrams



The double pendulum, interface and graph classes will all inherit from a base widget class. This will either be user defined of library defined, depending on what I will use for the implementation. This

generic widget will inherit from a base class, which will either be library defined or language defined, again depending on what is used.

Design

Overall System Design

As the system will be used as a teaching aid, it should be easy to use overall and it should be self-explanatory as much as possible to assist with ease of use when the user has had no experience with the software before. However it should also have clear and easily understood user manual that includes examples and demonstrations of the different aspects and features of the system.

The interface should be clean and efficient, its appearance and design should be familiar to the user even if they haven't used it before, so it is essential to use a library that support 'native' GUI widgets, e.g. buttons, sliders, labels; that the user is comfortable using and so using the interface will not obstruct the teaching that the system aids.

I will be using Python (version 3.3) as the programming language for the simulation. I will also be using PyQt4 for the GUI. This is a Python binding of Qt; a rich framework that can create professional looking interfaces and supports native widgets.

Design data dictionary

Term	Definition
Python	This is the language being used. The version is Python 3.3
PyQt	This is the main library being used
[Double] pendulum [object/widget[	This refers to the object in the program that serves as the double pendulum widget. It will be user defined, but will inherit some functionality from PyQt
Graph [object/widget]	This refers to the object in the program that serves as the graph widget. It will be user defined, but will inherit some functionality from PyQt
Control panel	This refers to the section of the main program that the user can interact with. It will consist of a series of sliders and buttons in order to allow user input.
Runge-Kutta	This is the mathematical technique that performs the numerical integration necessary to run the simulation

Storage media

The simulation will be entirely run from the computer's local hard drive. Any files it creates, such as exported images or saved states, will also be saved in the same directory, on the local hard drive.

Description of modular structure of system

The system can be broken down into three main parts.

Double Pendulum Display

This will be the region of the window that is actually showing the double pendulum in action. It will consist of a central fixed pivot, two arms and two heads. The heads and the pivot will be circles. The animation of the swinging pendulum should be smooth and comfortable to watch. The colours should not clash with the rest of the environment. The second pendulum head should leave a 'trail' behind it, showing the path it has taken. The user will be able to export a bitmap file of the current pendulum trail.

Graph Display

This will be the region where graphs are dynamically displayed corresponding to the real time data that is also being displayed by the double pendulum display. The user will have the option of choosing different variables for each axis, e.g. angle 1, angle 2, angular velocity 1 or 2; the graph will update and change as the simulation continues. The graph should be scaled appropriately for the conditions of the pendulum it is corresponding to, i.e. the graph should be perfectly 'zoomed in' for the current set of results, which will show more clearly the detail of graphs drawn for low energy initial settings than if the graphs all had fixed axes. Also the user will be able to export a bitmap of the current graph display.

Control panel

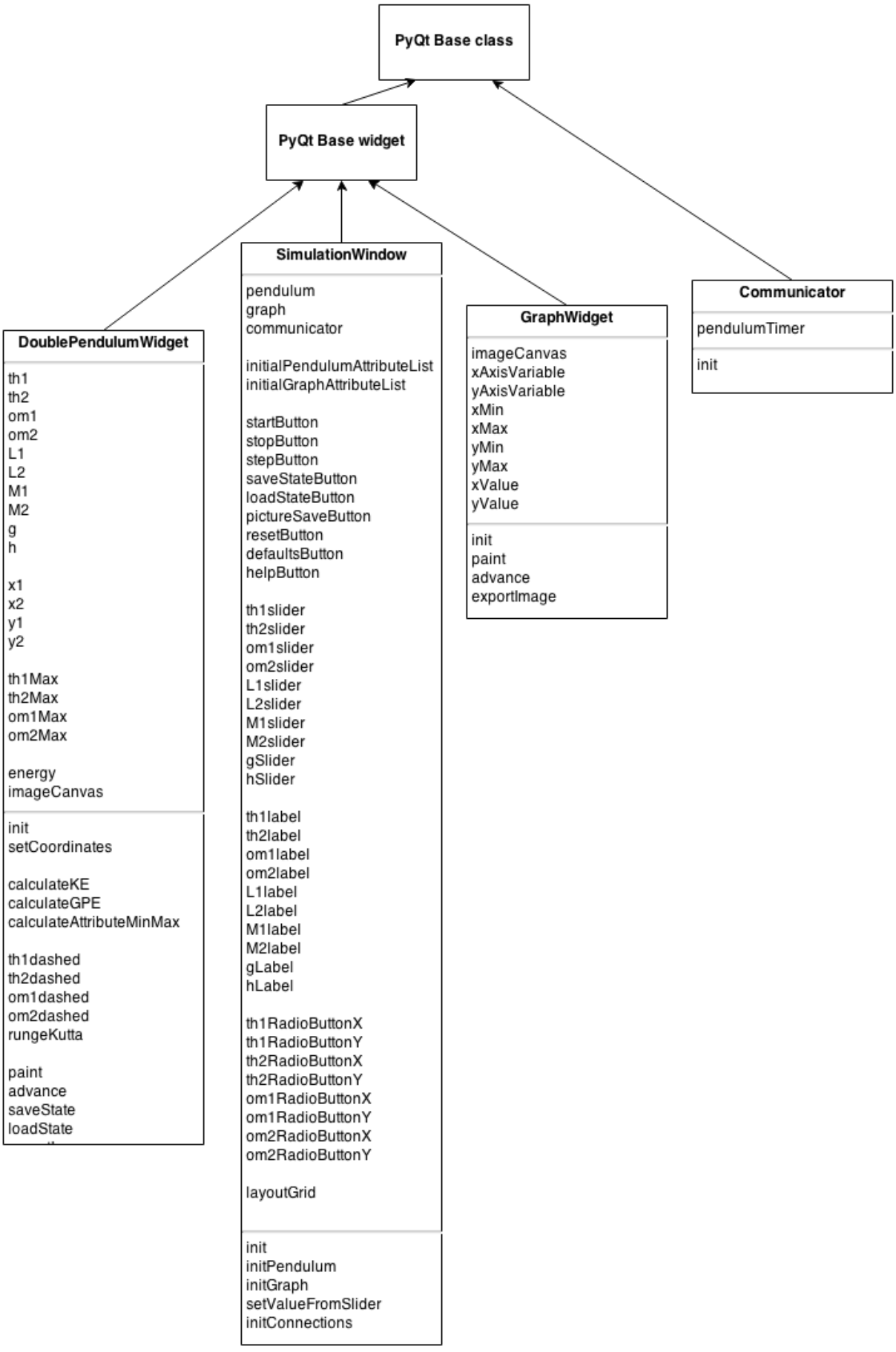
This is the part of the interface that the user will directly interact with. It is therefore essential that it should be clear and easy to use, without unnecessary complications, and with all the features clearly displayed.

For this reason, I am shying away from drop down menus, as they tend to require exploration and memorising before they become fast, fluid and easy to use. Instead, the control panel should consist of self-explanatory control buttons (e.g. start, stop, save, load etc.), easy to use sliders to set initial conditions, and an easy mechanism to choose the variable on each axis for the graph. For this I will use radio buttons, as they provide a clear way to choose only one option out of several.

Detailed object analysis

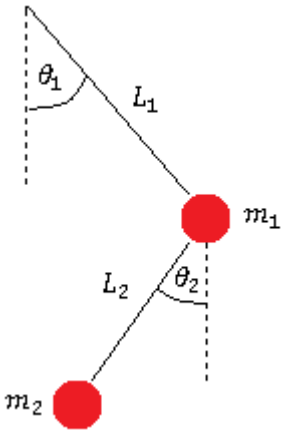
This is an object diagram that plans out the classes of the system, their attributes and their methods. Note that these are all user defined, so those inherited from PyQt4 (those shown as the PyQt base class and base widget below) have no attributes or methods listed here. For the user defined classes, the title is at the top of each box, the section beneath it lists its attributes, and the last section lists its methods.

The final program may differ slightly in its exact methods and attributes for a variety of reasons, such as name abbreviations, simplification of certain processes, certain processes becoming less simple (i.e. splitting sections of a large function into other smaller functions), or just generally as slightly better ways of doing certain things present themselves as the program is being implemented.


PyQt Base class
PyQt Base widget
SimulationWindow
pendulum
graph
communicator
initialPendulumAttributeList
initialGraphAttributeList
startButton
stopButton
stepButton
saveStateButton
loadStateButton
pictureSaveButton
resetButton
defaultsButton
helpButton
th1slider
th2slider
om1slider
om2slider
L1slider
L2slider
M1slider
M2slider
gSlider
hSlider
th1label
th2label
om1label
om2label
L1label
L2label
M1label
M2label
gLabel
hLabel
th1RadioButtonX
th1RadioButtonY
th2RadioButtonX
th2RadioButtonY
om1RadioButtonX
om1RadioButtonY
om2RadioButtonX
om2RadioButtonY
layoutGrid
init
initPendulum
initGraph
setValueFromSlider
initConnections
DoublePendulumWidget
th1
th2
om1
om2
L1
L2
M1
M2
g
h
x1
x2
y1
y2
th1Max
th2Max
om1Max
om2Max
energy
imageCanvas
init
setCoordinates
calculateKE
calculateGPE
calculateAttributeMinMax
th1dashed
th2dashed
om1dashed
om2dashed
rungeKutta
paint
advance
saveState
loadState
GraphWidget
imageCanvas
xAxisVariable
yAxisVariable
xMin
xMax
yMin
yMax
xValue
yValue
init
paint
advance
exportImage
Communicator
pendulumTimer
init

Identification of processes and suitable algorithms for data transformation

A large portion of my project will be based around the mathematics of the double pendulum system. I will be using the Newtonian technique, where I will find expressions to describe the position, velocity and acceleration of the two pendulums, and then find expressions of the forces acting on them. Combining the two will give me a set of differential equations that I will be able to numerically integrate. I will use the Runge-Kutta method to numerically integrate these expressions.



In our model of the double pendulum, we define the following symbols:

m_n = the mass of the head n (i.e. $n = 1$ or 2)

L_n = the length of rod n

θ_n = the angle made by rod n, where positive is anticlockwise and 0 is straight down (so in the diagram θ_1 would be positive and θ_2 would be negative)

x_n = horizontal position of pendulum mass n

y_n = vertical position of pendulum mass n

The origin (the point with (x, y) coordinate $(0, 0)$) is situated at the pivot of the upper pendulum. x increases to the right of the origin and y increases upwards (the coordinates for both pendulums in this diagram would have a negative y value).

Using basic trigonometry, we can establish a set of formulae for the x and y coordinates of each pivot.

$$x_1 = L_1 \sin\theta_1$$

$$y_1 = -L_1 \cos\theta_1$$

$$x_2 = x_1 + L_2 \sin\theta_2$$

$$y_2 = y_1 - L_2 \cos\theta_2$$

To obtain the velocity, we differentiate both sides of all four equations with respect to time. The notation used here is: $\frac{dx}{dt} = x'$

$$x_1' = \theta_1' L_1 \cos\theta_1$$

$$y_1' = \theta_1' L_1 \sin\theta_1$$

$$x_2' = x_1' + {\theta_2}' L_2 \cos\theta_2$$

$$y_2' = y_1' + {\theta_2}' L_2 \sin\theta_2$$

As velocity is the first derivative of position, and acceleration is the first derivative of velocity, to obtain acceleration we can differentiate position with respect to time again, resulting in the second derivative of position. Our notation is a logical extension of the notation from earlier, where $\frac{d^2x}{dt^2} = x''$

$$x_1'' = -{\theta_2}'^2 L_1 \sin\theta_1 + \theta_1'' L_1 \cos\theta_1$$

$$y_1'' = \theta_1'^2 L_1 \cos\theta_1 + \theta_1'' L_1 \sin\theta_1$$

$$x_2'' = x_1'' - {\theta_2}'^2 L_2 \sin\theta_2 + \theta_2'' L_2 \cos\theta_2$$

$$y_2'' = y_1'' + \theta_2'^2 L_2 \cos\theta_2 + \theta_2'' L_2 \sin\theta_2$$

Now that we have equations describing the motion of each pendulum mass, we need to find equations to describe the forces acting on them. We can treat each mass as being 'point particles'; assuming their mass acts and one point in space. We will also define the following:

$T_n =$ tension in rod n
$m_n =$ mass of pendulum head n (defined earlier)
$g =$ gravitational constant

The forces that act on the upper pendulum are the tension in the upper rod, the tension in the lower rod, and its weight. Splitting the forces into their horizontal and vertical components, and using Newton's $F = ma$ we get the following equations:

$$m_1 x_1'' = -T_1 \sin\theta_1 + T_2 \sin\theta_2$$

$$m_1 y_1'' = T_1 \cos\theta_1 - T_2 \cos\theta_2 - m_1 g$$

The forces acting on the lower pendulum are tension from the top rod and its weight. Similarly to above we get the following equations:

$$m_2 x_2'' = -T_2 \sin\theta_2$$

$$m_2 y_2'' = T_2 \cos\theta_2 - m_2 g$$

With the aim of finding expressions for ${\theta_1}''$ and ${\theta_2}''$ in terms of θ_1, ${\theta_1}'$, θ_2 and ${\theta_2}'$ we merge the four equations we have describing the forces to gain the following:

$$m_1 x_1'' = -T_1 \sin\theta_1 - m_2 x_2''$$

$$m_1 y_1'' = T_1 \cos\theta_1 - m_2 y_2'' - m_2 g - m_1 g$$

If we multiply both equations by $\sin\theta_1$, we can rearrange and equate them to each other, leading to the equation:

$$\sin\theta_1 \,(m_1 y_1'' + m_2 y_2'' + m_2 g + m_1 g) = -\cos\theta_1 \,(m_1 x_1'' + m_2 x_2'')$$

Similarly, we can multiply the two equations worked out for the forces acting on the lower pendulum by $\cos\theta_2$ and $\sin\theta_2$ respectively, rearrange and equate again to get:

$$\sin\theta_2 \,(m_2 y_2'' + m_2 g) = -\cos\theta_2 \,(m_2 x_2'')$$

The problem we now have is one of rearranging many complicated equations to gain expressions for θ_1'' and θ_2''. As the first derivative with respect to time of θ is angular velocity, ω, the second derivative with respect to time (which is the angular velocity differentiated with respect to time) is the angular acceleration, ω'.

The fastest way to do this is using would be using algebraic manipulation software, such as Mathematica. It can be done by hand, but it would be very time and space consuming to write it out the intermediate steps here, so instead I will write our expressions for angular acceleration in terms of $\theta_1, \omega_1, \theta_2, \omega_2$

$$\omega_1' = \frac{-g(2m_1 + m_2)\sin\theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2\sin(\theta_1 - \theta_2)\, m_2 ({\omega_2}^2 L_2 + {\omega_1}^2 L_1 \cos(\theta_1 - \theta_2))}{L_1(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))}$$

$$\omega_2' = \frac{2\sin(\theta_1 - \theta_2)\,({\omega_1}^2 L_1(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + {\omega_2}^2 L_2 m_2 \cos(\theta_1 - \theta_2))}{L_2(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))}$$

Clearly these can't be integrated so instead we have to use numerical methods. I will be using the Runge-Kutta algorithm, a very accurate method for approximately solving differential equations. It takes the following form:

Let's say we have m variables, $x_1, x_2, x_3 \ldots x_m$ which all vary according to some other variable. In our simulation, that other variable is time, and our variables are $\theta_1, \omega_1, \theta_2, \omega_2$. Let's say we also have m differential equations, each one corresponding with a variable:

$$x_1' = f_1(x_1, x_2, \ldots x_m)$$

$$x_2' = f_2(x_1, x_2, \ldots x_m)$$

…

$$x_m' = f_m(x_1, x_2, \ldots x_m)$$

As you can see these all consist of a derivative on the left hand side, all with respect to the variable they all vary with (time in our case as aforementioned) , and no derivatives on the right hand side. In our simulation, the differential equations will be:

$$\theta_1' = \omega_1$$

$$\theta_2' = \omega_2$$

$$\omega_1' = \frac{-g(2m_1 + m_2)\sin\theta_1 - m_2 g \sin(\theta_1 - 2\theta_2) - 2\sin(\theta_1 - \theta_2)\, m_2(\omega_2{}^2 L_2 + \omega_1{}^2 L_1 \cos(\theta_1 - \theta_2))}{L_1(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))}$$

$$\omega_2' = \frac{2\sin(\theta_1 - \theta_2)\,(\omega_1{}^2 L_1(m_1 + m_2) + g(m_1 + m_2)\cos\theta_1 + \omega_2{}^2 L_2 m_2 \cos(\theta_1 - \theta_2))}{L_2(2m_1 + m_2 - m_2\cos(2\theta_1 - 2\theta_2))}$$

which we derived earlier. Note that while they will all vary with time, time itself is not one of the variables inside the differentials.

For compactness, we can write this in vector notation as:

$$\bar{\boldsymbol{x}}' = \bar{\boldsymbol{f}}(\bar{\boldsymbol{x}})$$

where $\bar{\boldsymbol{x}} = (x_1, x_2, x_3 \ldots x_m)$ and $\bar{\boldsymbol{f}}$ is a sort of vector of functions, where $\bar{\boldsymbol{f}} = (f_1, f_2, f_3 \ldots f_m)$.

As we will be numerically integrating these functions, we need to decide how small a step in time we are going to have in between each 'state'. This will correspond to how accurate our simulation is, as in real life, time is incremented in infinitely small steps, so the smaller the step, the more accurate our simulation is. We will call this small step in time h. We can now label our time states, $\bar{\boldsymbol{x}}_n$ and $\bar{\boldsymbol{x}}_{n+1}$ which will be one h length step later. This means that $\bar{\boldsymbol{x}}_n$ holds the values of each of the m variables at time t_n, and $x_{1,n}$ is the value of the first variable (x_1) at the time t_n. To show this in full:

$$\bar{\boldsymbol{x}}_n = (x_{1,n}, x_{2,n} \ldots x_{m,n})$$

$$\bar{\boldsymbol{x}}_{n+1} = (x_{1,n+1}, x_{2,n+1} \ldots x_{m,n+1})$$

The Runge-Kutta algorithm will take the values for $\bar{\boldsymbol{x}}_n$ and calculate the values for $\bar{\boldsymbol{x}}_{n+1}$, which will of course be a step of length h after. It does this via a weighted averaging process of approximated values. This averaging process is what gives it superior accuracy over simpler numerical integration techniques such as the Euler method, on which the Runge-Kutta method is based.

This is the algorithm in full:

$$\bar{\boldsymbol{x}}_{n+1} = \bar{\boldsymbol{x}}_n + \frac{h}{6}(\bar{\boldsymbol{a}}_n + 2\bar{\boldsymbol{b}}_n + 2\bar{\boldsymbol{c}}_n + \bar{\boldsymbol{d}}_n)$$

where:

$$\bar{\boldsymbol{a}}_n = \bar{\boldsymbol{f}}(\bar{\boldsymbol{x}}_n)$$

$$\bar{\boldsymbol{b}}_n = \bar{\boldsymbol{f}}\left(\bar{\boldsymbol{x}}_n + \frac{h}{2}\bar{\boldsymbol{a}}_n\right)$$

$$\bar{\boldsymbol{c}}_n = \bar{\boldsymbol{f}}\left(\bar{\boldsymbol{x}}_n + \frac{h}{2}\boldsymbol{b}_n\right)$$

$$\bar{\boldsymbol{d}}_n = \bar{\boldsymbol{f}}(\bar{\boldsymbol{x}}_n + h\,\bar{\boldsymbol{c}}_n)$$

As aforementioned, the new vector $\bar{\boldsymbol{x}}_{n+1}$ will be the new state of all the variables. To make this clearer, we can return to the original, non-vector notation and write it like this:

$$x_{j,n+1} = x_{j,n} + \frac{h}{6}(a_{j,n} + 2b_{j,n} + 2c_{j,n} + d_{j,n})$$

where:

$$a_{j,n} = f_j(x_{1,n}, x_{2,n} \dots, x_{m,n})$$

$$b_{j,n} = f_j\left[\left(x_{1,n} + \frac{h}{2}a_{1,n}\right), \left(x_{2,n} + \frac{h}{2}a_{2,n}\right), \dots, \left(x_{m,n} + \frac{h}{2}a_{m,n}\right)\right]$$

$$c_{j,n} = f_j\left[\left(x_{1,n} + \frac{h}{2}b_{1,n}\right), \left(x_{2,n} + \frac{h}{2}b_{2,n}\right), \dots, \left(x_{m,n} + \frac{h}{2}b_{m,n}\right)\right]$$

$$d_{j,n} = f_j[(x_{1,n} + h\,c_{1,n}), (x_{2,n} + h\,c_{2,n}), \dots, (x_{m,n} + h\,c_{m,n})]$$

Each of these equations is applied to each variable $j = (1, 2 \dots m)$ to get the full set of new variables to describe the simulation one step later. Most computers have a 'resolution' of 1 millisecond. This means that the smallest step in time that the operating system can realistically and reliably pass to high level programs is 0.001 seconds. This provides a high degree of accuracy for my simulation, and this will be the lower limit for one step; if the user's computer isn't fast enough to smoothly run the simulation at that accuracy, then they will have the option to increase it.

Another aspect of the design that involves mathematical derivation is the scaling of the graph axes. I had to derive this part without assistance from external sources, as I couldn't find any such derivation on the internet or in written word. In order to ensure the correctness of the derivation, I had a member of staff from both the school's Physics department and the school's Maths department to check that there weren't any mistakes.

The gravitational potential energy of an object is found using the following equation:

$$E_P = mgh$$

where m is the object's mass, g is the gravitational field strength and h is the object's height above the ground. In my system, I defined the central pivot as the origin. However, this would mean that if the pendulum was below the origin, it would have a negative gravitational potential energy. As it is impossible to have a negative gravitational potential energy in real life, for the calculations involving potential energy I instead define the "ground" to be level with the pendulum's lowest point; i.e. the position of the second head when the pendulum is vertically downwards. This means we need to shift the y values of each head so that they are measured starting from the new ground. As the

ground is $l_1 + l_2$ away from the origin, if we simply add this value to each of our y values, then we can calculate the potential energies and be certain they will be positive. To save space, we will define this constant for shifting as

$$S = l_1 + l_2$$

which will mean the potential energy of the pendulum is

$$E_P = m_1 g(y_1 + S) + m_2 g(y_2 + S)$$

Using the definitions for y we derived earlier, we can rewrite this as

$$E_P = -m_1 g \cos\theta_1 + m_1 gS - m_2 g l_1 \cos\theta_1 - m_2 g l_2 \cos\theta_2 + m_2 gS$$

I have purposefully not simplified the equation, as we are about to rearrange it. As we are trying to calculate maximum values for θ_1 and θ_2, we can make some assumptions. Firstly, the maximum of each one will occur when the pendulum is stationary. This is because the total (unchanged) energy of the pendulum is always the sum of the potential energy and the kinetic energy. When the pendulum is still, the kinetic energy is 0. This means that all the energy is potential energy, meaning the pendulum can be at its highest, which means that the angles will be at their greatest. Another assumption we can make is that when one angle is at its maximum, the other one will be 0. This is for a similar reason as the last one. When the other angle is at 0, its potential energy is at a minimum, again meaning more energy is "available" to have the first angle greater.

So setting $\theta_2 = 0$ and rearranging to get θ_1 as the subject, we get

$$\theta_1 = \operatorname{acos}\left(\frac{g(S(m_1 + m_2) - m_2 l_2) - E_T}{l_1 g(m_1 + m_2)}\right)$$

where $E_T = E_P + E_K$, the system's total energy (a constant, as mentioned before), the constituent parts of which can be calculated using the energy formulae above (for potential energy) and below (for kinetic). This is the maximum value of θ_1.

If we set $\theta_1 = 0$ and rearrange to make θ_1 the subject, we get

$$\theta_2 = \operatorname{acos}\left(\frac{g(m_1 + m_2)(S - l_1) - E_T}{l_2 g m_2}\right)$$

which is the maximum value of θ_2.

However, precautions must be taken. This is because if the energy is high enough, even if the pendulum is vertically upwards (i.e. maximum potential energy) it may still have enough energy to be moving. This means we have to check to see if the pendulum's maximum potential energy for each head is less than the total energy. If that is the case, then the maximum possible angle will simply be π radians, i.e. vertically upwards. Thus we must make sure that

$$E_T \leq g(2l_1(m_1 + m_2) + l_2 m_1)$$

otherwise θ_1 will be a maximum of π, and

$$E_T \leq l_2 g(m_1 + 2m_2)$$

otherwise θ_2 will be a maximum of π.

The kinetic energy of a moving object is found by the following equation:

$$E_K = \frac{1}{2}mv^2$$

where m is the object's mass and v is the object's speed. As the pendulum has two moving parts, the total kinetic energy is the sum of the kinetic energy of each part. In order to get the velocity of each pendulum, we will use the components of velocity we derived earlier and combine them using Pythagoras's theorem. This give the result

$$E_K = \frac{1}{2}m_1({x_1}'^2 + {y_1}'^2) + \frac{1}{2}m_2({x_2}'^2 + {y_2}'^2)$$

This multiplies out to get

$$E_K = \frac{1}{2}m_1{l_1}'^2\theta_1'^2 + \frac{1}{2}m_2(l_1^2\theta_1'^2 + l_2^2\theta_2'^2 + 2l_1l_2\theta_1'\theta_2'\cos(\theta_1 - \theta_2))$$

In order to work out the maximum angular velocity for each head, we need to make some assumptions. Firstly, the pendulum will be in the vertically downwards position. This means that $\theta_1 = \theta_2 = 0$, causing the $\cos(\theta_1 - \theta_2)$ to be equal to 1, and to disappear from the above equation. As the pendulum is vertically downwards, this will mean that potential energy is at a minimum, meaning kinetic energy will be at a maximum. This allows the angular velocity of each head to be at a maximum. However, there are two other things that must be taken into account.

The first thing that needs to be taken into account is the fact that the potential energy, whilst it is at a minimum, is not 0. This will mean that the energy "available" will be $= E_T - E_{P_{min}}$. To work out the minimum potential energy, we consider the pendulum in the vertically downward position, and we remember that for energy calculations we have stated that the "ground" is level with the bottom pendulum head. This means that the second head has no potential energy (as its height $h = 0$), and that the potential energy of the system is simply the potential energy of the first head. This head will be at a height $h = l_2$, meaning that $E_{P_{min}} = m_1gl_2$.

The second thing that has to be taken into account is that we must calculate the value of the angular velocity for the *other* head that will allow a maximum angular velocity for this one. In other words, if we want to find out the value of $\omega_{1_{max}}$, we need to find out what the value for ω_2 will be at the same time. To do so, we can group the coefficients in the above equation, and use our new knowledge of the energy to get the following:

$$E - E_{P_{min}} = {\theta_1'}^2\left(\frac{1}{2}l_1^2(m_1 + m_2)\right) + {\theta_2'}^2\left(\frac{1}{2}m_2l_2^2\right) + \theta_1'\theta_2'(l_1l_2)$$

In order to solve this, we can define some constants so that the expression is easier to work with. We will define the following constants:

$$A = \frac{1}{2} l_1^2 (m_1 + m_2)$$

$$B = \frac{1}{2} m_2 l_2^2$$

$$C = l_1 l_2 m_2$$

$$D = E - E_{P_{min}}$$

Another step we can take (again, just to make it easier to work with, it is perfectly possible to do this all without renaming anything) is to rename our variables:

$$\theta_1' = x$$

$$\theta_2' = y$$

This means that we can write the expression thus:

$$Ax^2 + Cxy + (By^2 - D) = 0$$

And we can observe that it is now in the form of a quadratic equation, $ax^2 + bx + c = 0$, meaning we can use the quadratic formula to solve it:

$$x = \frac{-b \pm \sqrt{b^2 + 4ac}}{2a}$$

As we are looking for the *maximum* value for x, we can rewrite the $\pm$ as just a $+$, which gives us:

$$x = \frac{-Cy + \sqrt{C^2 y^2 - 4A(By^2 - D)}}{2A}$$

We now need to work out what value of y will give this maximum value for x. The x in the above equation is no longer a variable; it is a constant value, which means its differential will be 0. So, if we differentiate the above expression with respect to y, we obtain the following:

$$0 = -\frac{C}{2A} + \frac{y(C^2 - 4AB)}{2A\sqrt{y^2(C^2 - 4AB) + 4AD}}$$

Again, we are going to define some more constants to make this expression easier to work with, the end result will be the same but the steps will be easier to make.

$$F = C^2 - 4AB$$

$$G = 4AD$$

If we put in our new constants and rearrange to get y as the subject, we find that

$$y = -\sqrt{\frac{GC^2}{F(F - C^2)}}$$

With the negative sign indicating that we want the value to be a negative number, to allow x to be as large as possible.

If we substitute this new value for y into the expression we made for x, we will get the maximum angular velocity. In order to do the same to work out the maximum angular velocity for the other head, we swap our definitions of y and x, which means that if you swap the values of the constants of A and B, you will obtain the maximum for the second head.

Translation of algorithms to pseudocode

In the implementation of the Runge-Kutta algorithm, we have four differential functions. As they all require the same variables to be calculated, they can all be passed the same list of variables. Here the dot notation simply means that the variable is an attribute of that object.

```
variable_list <- [pendulum.theta_1, pendulum.theta_2, pendulum.omega_1,
pendulum.omega_2]

function theta_1_differential(variable_list)

      return variable_list[2]

end


function theta_2_differential(variable_list)

      return variable_list[3]

end


function omega_1_differential(variable_list)

      theta_1 <- variable_list[0]

      theta_2 <- variable_list[1]

      omega_1 <- variable_list[2]

      omega_2 <- variable_list[3]

      mass_1 <- pendulum.mass_1

      mass_2 <- pendulum.mass_2

      length_1 <- pendulum.length_1

      length_2 <- pendulum.length_2

      gravity <- pendulum.gravity
```

```
      A <- -gravity * (2 * mass_1 + mass_2) * sin(theta_1)
      B <- -mass_2 * gravity * sin(theta_1 - 2 * theta_2)
      C <- -2 * sin(theta_1 - theta_2) * mass_2
      D <- omega_2 ^ 2 * length_2
      E <- omega_1 ^ 2 * length_1 * cos(theta_1 - theta_2)
      F <- (2 * mass_1 + mass_2 * (1 - cos(2 * (theta_1 - theta_2))))
      G <- length_1 * F
     return (A + B + C * (D + E)) / G
end

function omega_2_differential(variable_list)
      theta_1 <- variable_list[0]
      theta_2 <- variable_list[1]
      omega_1 <- variable_list[2]
      omega_2 <- variable_list[3]
      mass_1 <- pendulum.mass_1
      mass_2 <- pendulum.mass_2
      length_1 <- pendulum.length_1
      length_2 <- pendulum.length_2
      gravity <- pendulum.gravity
      A <- omega_1 ^ 2 * length_1 * (mass_1 + mass_2)
      B <- gravity * (mass_1 + mass_2) * cos(theta_1)
      C <- omega_2 ^ 2 * length_2 * mass_2 * cos(theta_1 - theta_2)
      D <- 2 * sin(theta_1 - theta_2) * (A + B + C)
      E <- (2 * mass_1 + mass_2 * (1 - cos(2 * (theta_1 - theta_2))))
      F <- length_2 * E
      Return D / F
end

function_list <- [theta_1_differential, theta_2_differential,
omega_1_differential, omega_2_differential]
```

```
function multiple_function_vector(variable_list, function_list,
letter_list, h)
      new_letter_list <- <empty list>
      temporary_variable_list <- <empty list>

      for each variable and letter in variable_list and letter_list do
            add (variable + h * letter) to temporary_var_list
      end

      for each function in function_list do
            add function(temporary_var_list) to new_letter_list
      end

      return new_letter_list
end

function runge_kutta(variable_list, function_list)
      h <- pendulum.h
      new_variable_list <- <empty list>
      a_list <- multiple_function_vector(variable_list, function_list,
variable_list, 0)
      b_list <- multiple_function_vector(variable_list, function_list,
a_list, h / 2)
      c_list <- multiple_function_vector(variable_list, function_list,
b_list, h / 2)
      d_list <- multiple_function_vector(variable_list, function_list,
c_list, h)

      for each variable, a, b, c, d in variable_list, a_list, b_list,
c_list, d_list do
            add (variable + (h / 6) * (a + 2 * b + 2 * c + d)) to
new_variable_list
```

```
        end
        return new_variable_list
end
```

That was the pseudocode for the functions associated with the Runge-Kutta algorithm. Below are those needed to perform the energy calculations to work out the maximum attribute values, as described in the section above. Rather than repeatedly type: pendulum.length_1, pendulum.length_2 etc., I have omitted them from the pseudocode below. In the real implementation, each attribute would need to be accessed from the object, as done so above.

```
function calculate_omega_max(omega_1_or_omega_2)
        A <- 0.5 * length_1 ^ 2 * (mass_1 + mass_2)
        B <- 0.5 * mass_2 * length_2 ^ 2
        C <- length_1 * length_2 * mass_2
        D <- pendulum_total_energy – pendulum_minimum_kinetic_energy

        if omega_1_or_omega_2 = 2 then
                temporary <- A
                A <- B
                B <- temporary
        end

        F <- C ^ 2 – 4 * A * B
        G <- 4 * A * D

        other_omega_value <- -square_root((G * C ^ 2) / (F * (F – C^2)))
        omega_maximum <- (-C * other_omega_value + square_root(C ^ 2 *
other_omega_value ^ 2 – 4 * A * (B * other_omega_value ^ 2 - D))) / (2 * A)
        return omega_maximum
end

function theta_1_maximum()
        S <- length_1 + length_2
```

```
      if pendulum_total_energy  >= gravity * (2 * length_1 * (mass_1 +
mass_2) + length_2 * mass_1) then

            theta_1_maximum = pi

      else

            theta_1_maximum = arc_cos( (gravity * (S * (mass_1 + mass_2) -
mass_2 * length_2) - pendulum_total_energy) / (length_1 * gravity * (mass_1
+ mass_2)) )

      end

      return theta_1_maximum

end


function theta_2_maximum()

      if pendulum_total_energy >= length_2 * gravity * (mass_1 + 2 *
mass_2) then

            theta_2_maximum = pi

      else

            theta_2_maximum = arc_cos( (gravity * (mass_1 + mass_2) * (S -
length_1) - pendulum_total_energy) / (length_2 * gravity * mass_2) )

      end

      return theta_2_maximum

end
```

User interface design and HCI rationale

As this is a teaching aid, the interface should look professional and it should be easy to use. As mentioned before, a way that I can achieve this is if the solution uses widgets that are common to the destination operating system, Windows 7 in this case. This will also avoid using any extreme colour schemes that could bring unwanted attention to the abstraction between the user and the underlying software.

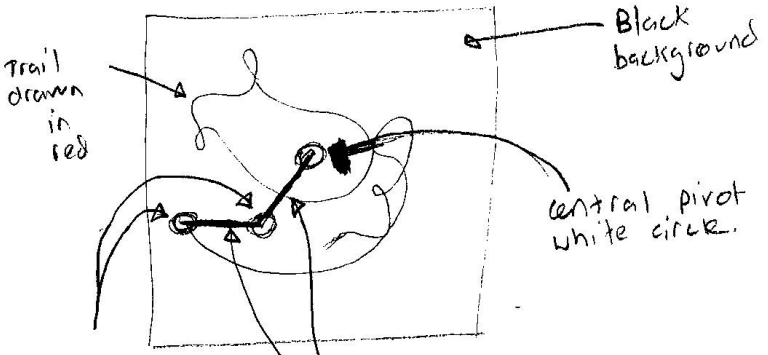
The ideal interface would be one that the user hardly notices that they are using, which is the exact intention of the above points. This will also be aided by intuitive controls that are clearly laid out and are self-explanatory as possible. In case any extra help is needed, there will be a clear and concise user manual that fully describes how the user can interact with the system.

I plan on avoiding using any menus for my interface. The reason is that menus tend to require some exploration and use before the user becomes accustomed to them. Whilst they are necessary in

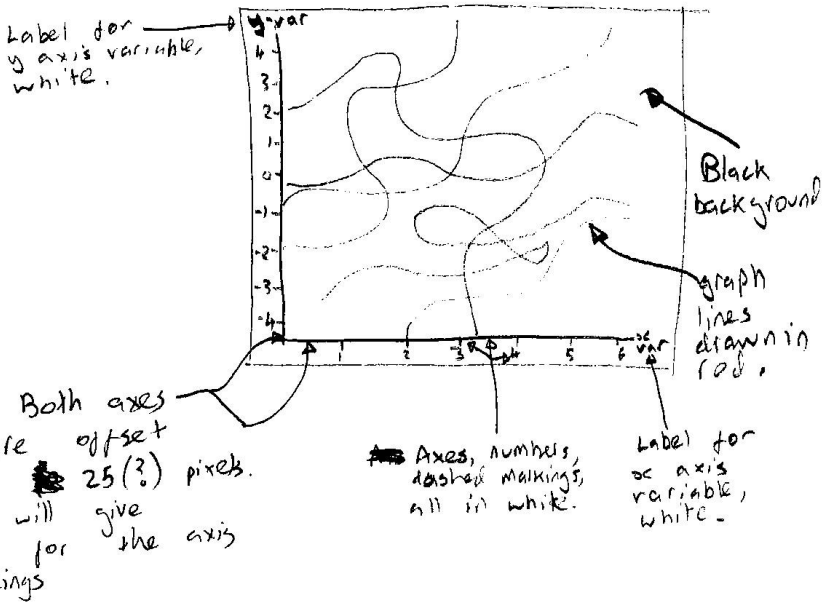
larger programs that have a UI with too many elements to display all at once, my planned UI should be limited enough in size that they won't be necessary. Instead, I plan to use buttons, sliders and radio buttons to allow the user to interact with the simulation. The following images are scanned pages of GUI design mock-ups:

User interface design planning

Pendulum display

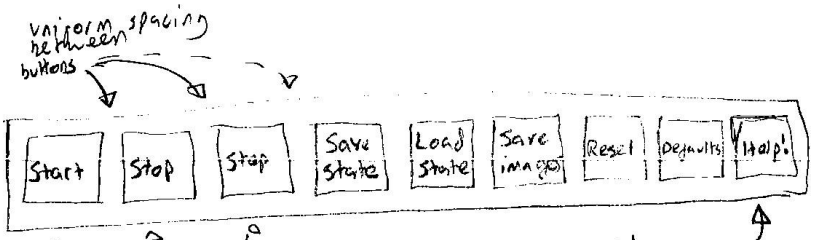


Graph display



(control panel)

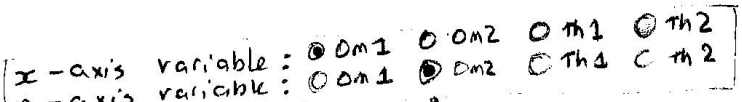
Button strip:



Buttons all same size and colour. colour will be the windows default scheme

the 'Help!' button will be the only button with red text, so it sticks out in case the user needs help. All the other will have black text.

Radio buttons:

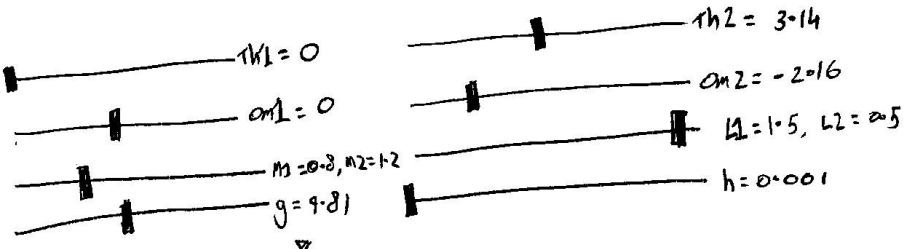


unselected appearance

selected appearance

when the user selects a button, the previous button is unchecked, meaning only 1 button can be checked in each row at a time.

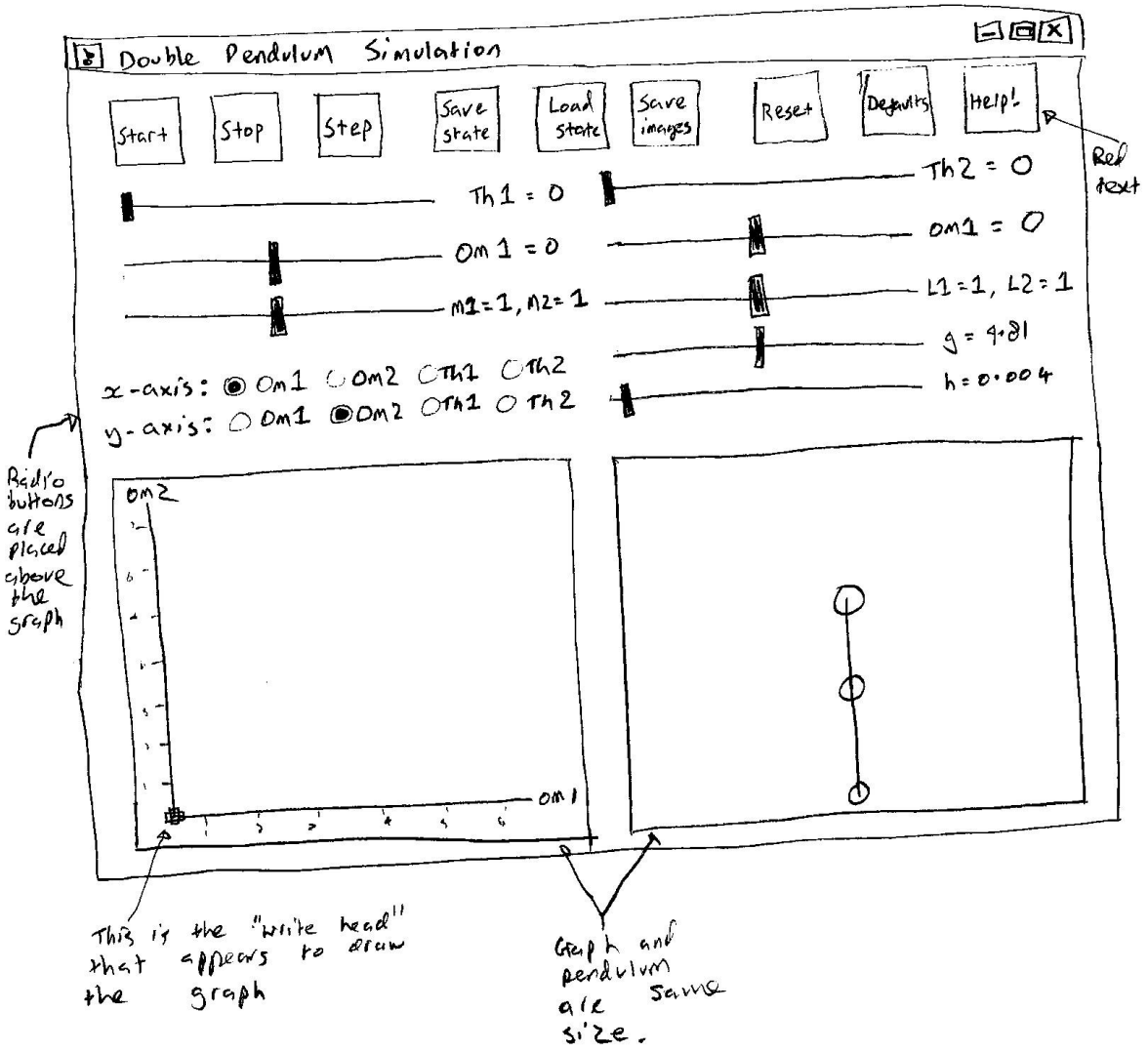
Sliders:



This row of sliders sets the ratio between two values, which means that it needs to have a label that is updated with the 2 new values.

labels should change as the sliders are moved

Full window


Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
Red text
Th1 = 0
Th2 = 0
om1 = 0
om1 = 0
m1=1, m2=1
L1=1, L2=1
g = 9.81
h = 0.004
x-axis: om1 om2 Th1 Th2
y-axis: om1 om2 Th1 Th2
Radio buttons are placed above the graph
om2
om1
This is the "write head" that appears to draw the graph
Graph and pendulum are same size.

Description of measures for security of system and data

As no sensitive data is stored or passed through the system, security is not an issue.

Overall test strategy in relation to the problem being solved and tested

A problem with the testing of my system is that it is mathematically very difficult to check if input values result in the system being in the correct place, as (because this is a chaotic system) a very small change in initial conditions will result in a completely different outcome down the line.

I plan on using this difficulty in testing as a technique *for* testing, in that I will set two different double pendulums with very slightly different initial conditions, and if it is working properly they should start their movements together and very gradually fall out of sync until they are doing completely different movements.

The number of functions in the final program will be of a size where it should still be feasible to test each of them in turn, using the range of values that they will receive for their arguments. The elements of the GUI can be tested to ensure that they properly function and that the user's inputs are correctly passed to the main program.

Another technique I plan on using is the fact that the system in not chaotic for small starting angles, and also that graphs drawn for small starting angles produce Lissajous curves, which I will be able to recognise during testing to confirm that both the graph drawing subsystem and the double pendulum display subsystem are not only working but are working together correctly.

Besides testing the correctness of the system, I will also be checking for non-logical errors that could cause errors or crashes of the program. I will concentrate particularly on the input of data from the user, and the saving of files, as these two can often result in bugs if they are not carefully approached. This should be a feasible technique, as the amount of user input is confined to a small section, so thorough testing will be possible.

System testing

The first set of tests is the two functions that are in the main program's scope. These are functions used in more than one place, so it is more appropriate for them to be available throughout the program.

Test Number	Scope	Function name	Function behaviour	Input values	Expected result	Result
1	Main program	`cutAngle(th)`	Receives an angle 'th', returns its equivalent in the range $-\pi < \theta < \pi$	6.2π	0.2π	**FAILED (see screenshots table)**
1-2				6.2π	0.2π	Passed
2				$-\frac{12\pi}{5}$	$-\frac{2\pi}{5}$	Passed
3				-7	-0.7168…	Passed
4	Main program	`getPictureName(name, ext)`	Receives the name and extension of a file to be saved. It will find the lowest available number that can go between name and ext in order for the file to be saved, i.e. "name0.ext", if that's taken then "name1.ext" etc.	name = "myFile" ext = ".txt" Location: empty folder	"name0.txt"	Passed
5				name = "myPicture" ext = ".bmp" Location: folder with files myPicture0.bmp, myPicture1.bmp, myPicture3.bmp,	"myPicture2.bmp"	Passed

The second table of tests are in the scope of the Double Pendulum widget. This includes the functions that perform the actual calculations for the motion of the double pendulum, the energy calculations that find the maximum values of the pendulum's attributes, the drawing of the pendulum widget itself, and the exporting of the pendulum's image.

Test Number	**Scope**	**Function name**	**Function behaviour**	**Input values**	**Expected result**	**Result**
6	Double Pendulum	`setXY(self)`	Sets the x and y coordinates of each pendulum head mathematically, and then sets their pixel coordinates inside the widget	self.L1 = 1.5 self.L2 = 0.5 self.th1 = pi/3 self.th2 = -pi/2 self.dScale = 100	1.29… -0.75 0.799… -0.75 329.9… 275 279.90… 275	Passed
7				self.L1 = 0.6 self.L2 = 1.4 self.th1 = 1.0023 self.th2 = 0.5 self.dScale = 150	0.505… -0.323… 1.176… -1.55… 375.844… 348.452… 476.52… 532.745…	Passed
8	Double Pendulum	`calculateEnergy(self)` and `omegaMax(self, num)`	Using the pendulum's current attributes, calculates the current kinetic and potential energies, the total energy, the minimum possible potential energy, the maximum possible values for self.th1, self.th2, self.om1, self.om2.	self.th1 = pi/2 self.th2 = pi self.om1 = 0 self.om2 = 0 self.M1 = 1 self.M2 = 1 self.L1 = 1 self.L2 = 1 self.g = 9.81	Kinetic energy = 0 Potential energy = 49.05 Total energy = 49.05 Potential min = 9.81 Om1 max = 8.85… Om2 max = 12.52… Th1 max = pi Th2 max = pi	Passed

9				self.th1 = pi/10 self.th2 = 1.068 self.om1 = -1.02 self.om2 = 2.1 self.M1 = 0.78 self.M2 = 1.22 self.L1 = 0.84 self.L2 = 1.16 self.g = 4.316	Kinetic = 2.49… Potential = 7.42… Total = 9.92… Min potential = 3.90… Om1 max = 4.67… Om2 max = 4.33… Th1 max = 1.39… Th2 max = 1.55…	Passed
10	Double Pendulum	`rungeKutta(self, varList, funcList), multiFuncVect(self, varList, funcList, letList, h), om1d(self, varList), om2d(self, varList), th1d(self, varList), th2d(self, varList), advance(self)`	These are the methods that combine to perform the necessary numerical integration. In order to test that the (mathematical) functions have been properly described, and that the numerical integration is being performed correctly, we can run the simulation for a certain amount of time and compare with an external source (for these tests it will be WolframAlpha)	Th1 = 1.005 Th2 = 1.005 Om1 = 0 Om2 = 0 M1 = 1 M2 = 1 L1 = 1.2 L2 = 0.8 Time duration = 10 s	Th1 = -0.8037 Th2 = -0.9958	Passed
11				Th1 = pi Th2 = pi/2 Om1 = 0 Om2 = 0 M1 = 1.4 M2 = 0.6 L1 = 0.7 L2 = 1.3 Time duration = 6 s	Th1 = -19.32 = -0.470 Th2 = 7.259 = 0.9758	Passed
12	Double pendulum	`paintEvent(self, event)`	Draws the widget. Should draw black background, white central pivot. Colour of heads should vary with mass. Colour of arms should vary with angular velocity. Trail drawn should be	Th1 = pi/2 Th2 = pi Om1 = 0 Om2 = 0 M1 = 1	Arms should be shade of green when pendulum swings through bottom. Trail should be red.	Passed

			red	M2 = 2 L1 = 1 L2 = 1	Heads should be the same shade of blue.	
13				Th1 = pi Th2 = pi Om1 = 0 Om2 = 0 M1 = 1.4 M2 = 0.6 L1 = 1.4 L2 = 0.6	Pendulum should be vertically upwards. Arms should be white. First arm should be longer than second. First head should be dark blue, second head should be light blue.	Passed
14	Double pendulum	`export(self)`	Saves the bitmap image of the pendulum's trail to file. Uses "getPictureName" so it can save multiple files to the same folder; as this function has already been successfully tested (see tests 4&5) I *won't* test to ensure it works in a folder with files already present.	Th1 = 2.136… Th2 = 3.707… Om1 = 0 Om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 Time duration = 10 s	Trail should be the same in the simulation as in the saved image.	Passed
15				Th1 = 0.88… Th2 = 4.021… Om1 = -3 Om2 = -3 M1 = 0.6 M2 = 1.4 L1 = 0.6 L2 = 1.4 Time duration = 30 s	Trail should be the same in the simulation as in the saved image.	Passed

The third table of tests are those in the scope of the graph widget. These include the functions that preform the necessary steps to find the pixel coordinates of the points to be drawn, the function that draws the widget, and the exporting of the graph's image.

Test Number	Scope	Function name	Function behaviour	Input values	Expected result	Result
16	Graph widget	`adjustX(self, x)`, `adjustY(self, y)`, `(adjustPoints(self))`	These functions take mathematical coordinates, and using the minimum value for an axis, the "scale-up" factor for drawing, and the number of pixels the axis is offset by (i.e. 25 pixels in from each side of the widget) to give the pixel coordinate for that point. The y-axis also needs to have taken into account that the pixel coordinates start from the top.	x = 1.2, xMin = -8, xDScale = 100 y = 0.4, yMin = -12. yDScale = 10, sideLength = 500	x = 945 y = 351	Passed
17				x = 0.6, xMin = -3.141, xDScale = 250 y = 1.74, yMin = -12.45, yDScale = 31, sideLength = 700	x = 960 y = 235	Passed
18	Graph widget	`paintEvent(self, event)`	Draws the graph widget.	Th1 = 0 Th2 = pi/2 Om1 = 0 Om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 Time duration = 30 s x axis variable = th1 y axis variable = th2	Black background, axes in white. Axes should be offset from the edges of the widget by 25 pixels. Dashes should be drawn from the axis towards the edge. Numbers should match up with each dash. Variable labels should on end of each axis. Drawing head should be two crossed rectangles, the line (in red) should have each	Passed

					point plotted in the central square of the head. Variable labels should be th2 on y and th1 on x	
19				Th1 = 4.84 Th2 = 2.26 Om1 = 2.22 Om2 = 2.16	Same as test 18, except labels: label on x axis should be om1, label on y axis should be om2	Passed
20	Graph widget	`export(self)`	Saves a .bmp file of the graph to disk. Similarly to tests 14&15, there is no need to try test with different files in the folder. Note that this also ensures proper connections for the Save images button	Th1: 1.06… Th2: 1.57… Om1: 0 Om2: 0 L1: 1 L2: 1 M1: 1 M2: 1 g: 9.81 Time duration: 10 s Graph x-axis: om1 Graph y-axis: om2	Image should be the same in the simulation as in the saved file	Passed
21				Th1: 3.141 Th2: 3.141 Om1: 1.02 Om2: 0.42 L1: 0.66 L2: 1.34 M1: 0.6 M2: 1.4 g: 9.81 Time duration: 25 s Graph x-axis: th1	Image should be the same in the simulation as in the saved file.	Passed

				Graph y-axis: th2		
22				Th1: 0 Th2: 0 Om1: 3 Om2: -3 L1: 1 L2: 1 M1: 1 M2: 1 g: 0	Image should be the same in the simulation as in the saved file.	Passed

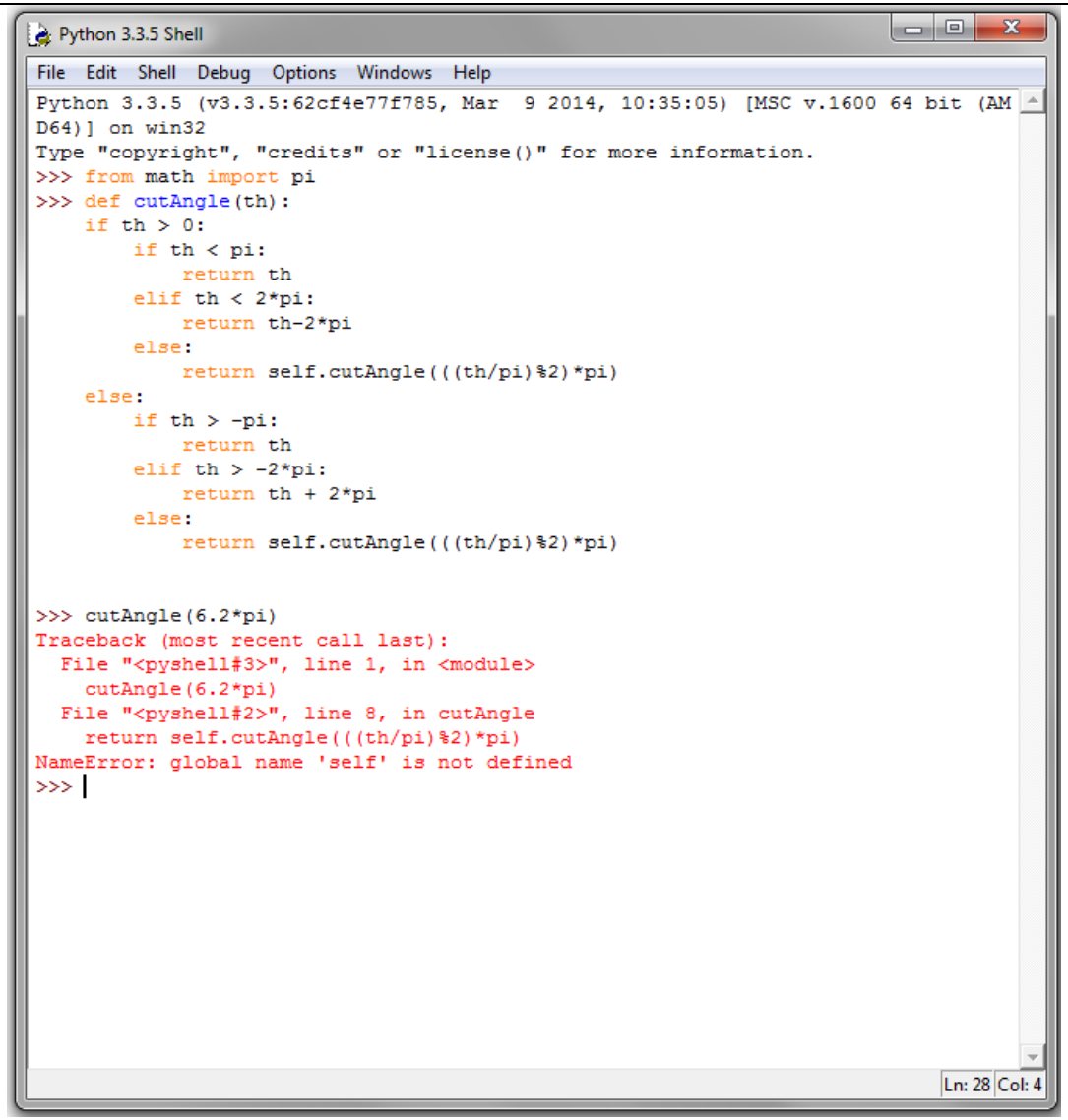
The fourth table of tests are the functions found in the Double Pendulum Window. These have to be tested slightly differently than above. The reason is that for the most part the above functions (the biggest exception being each widget's "paintEvent") are predominately user defined, with limited use of PyQt4's functions. However, in the main window, the large majority of the code written is interacting with the library in detail. The result is that a lot of the functions are connected through PyQt without it being immediately obvious. For example, the sliders are connected (using PyQt4's signal and slot mechanism) to their own function that receives the value of the slider, mathematically translates it to the appropriate attribute, resets the graph and the pendulum widgets with the new value, and reconnects all the sliders and buttons to these new widgets. This makes it difficult to break the functions down into "testable" sizes, so a lot of the tests here have to do multiple functions at once.

Test Number	Scope	Function name	Function behaviour	Input values	Expected result	Result
23	DP Window	`setTh1FromSlider(self, val),`	The function called when the Th1 slider is moved. A “print” has been added for testing to show it works	The slider can vary between 0 and 2π	The printout of self.ith1 should match the label	Passed
24	DP Window	`setTh2FromSlider(self, val)`	The function called when the Th2 slider is moved. A “print” has been added for testing to show it works	The slider can vary between 0 and 2π	The printout of self.ith2 should match the label	Passed
25	DP Window	`setOm1FromSlider(self, val)`	The function called when the om1 slider is moved. A “print” has been added for testing to show it works	The slider starts at 0, is -3 at far left and +3 at far right	The printout of self.iom1 should match the label	Passed
26	DP Window	`setOm2FromSlider(self, val)`	The function called when the om2 slider is moved. A “print” has been added for testing to show it works	The slider starts at 0, is -3 at far left and +3 at far right	The printout of self.iom2 should match the label	Passed
27	DP Window	`setGfromSlider(self, val)`	The function called when the g slider is moved. A “print” has been added for testing to show it works	The slider starts centrally at 9.81, is 0 at far left and 19.62 at far right	The print out of self.ig should match the label.	Passed
28	DP Window	`setHfromSlider(self, val)`	The function called when the h slider is moved. A “print” has been added for testing to show it works	The slider starts close to the left at 0.004, is 0.001 at far left and 0.1 at far right	The printout of self.ih should match the label	Passed
29	DP Window	`setLengthRatioFromSlider(self, ratio)`	The function is called when the length ratio slider is moved. A “print” has been added for testing to show it works	The slider starts centrally with L1 = L2 = 1, at far left L1 = 0.4 and L2 = 1.6, at far right L1 = 1.6 and L2 = 0.4	The print out of self.iL1 and self.iL2 should match the two labels	Passed
30	DP Window	`setMassRatioFromSlider(self, ratio)`	The function is called when the mass ratio slider is moved. A “print” has been added for testing to show it works	The slider starts centrally with M1 = M2 = 1, at far left M1 = 0.4 and M2 = 1.6, at far right M1 = 1.6 and	The print out of self.iM1 and self.iM2 should match the two labels	Passed

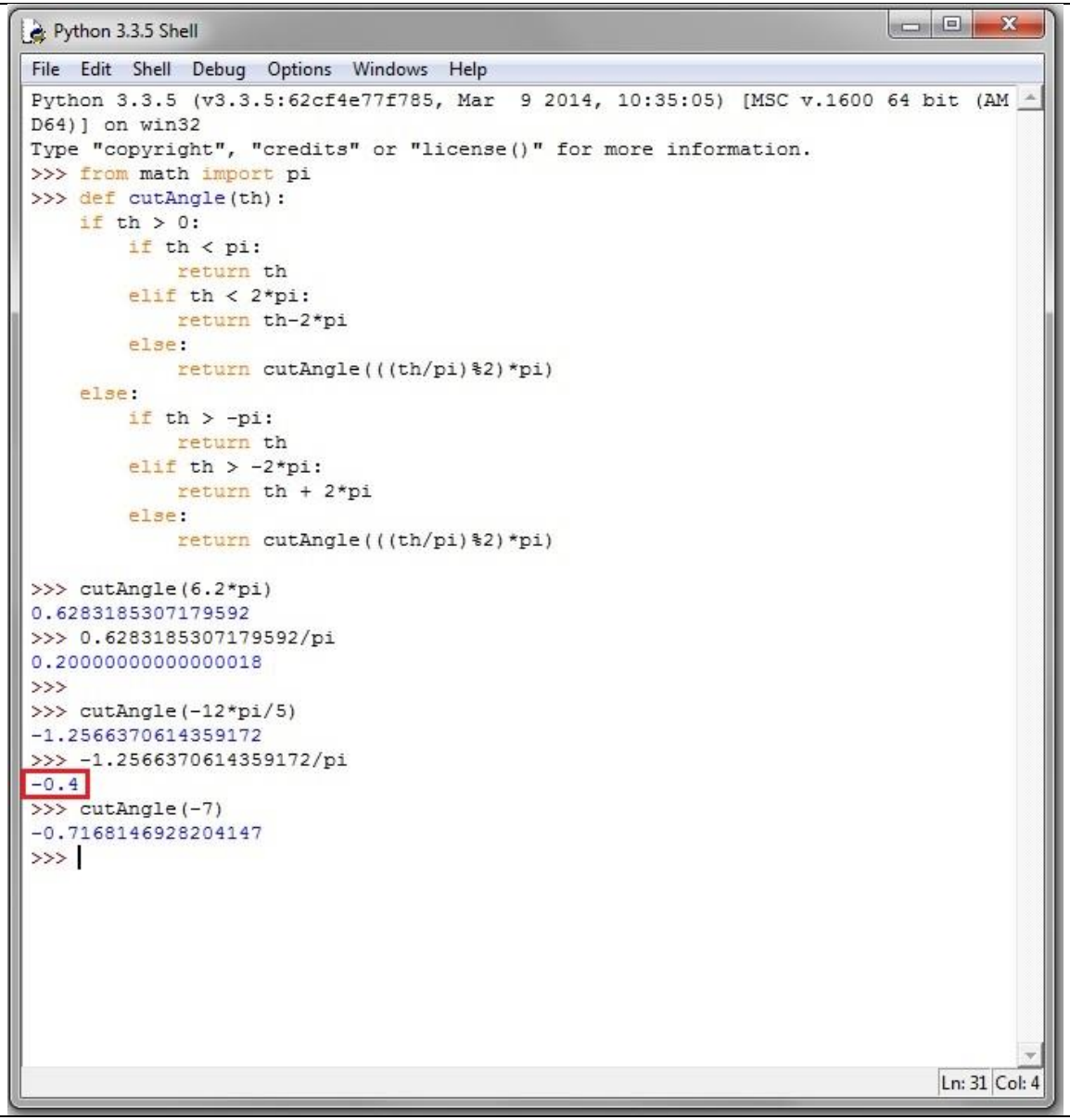
				M2 = 0.4		
31	DP Window	`setGraphYVar(self, yvarID), setGraphXVar(self, xvarID)`	Uses varID to select correct variable from built in list. This will also be checking that the radio buttons are correctly linked to their appropriate functions. A print has been added to show the values being passed to the function are correct	yvarID = 2 (om2) xvarID = 1 (om1)	The variable labels on the graph should match the selections	Passed
32				yvarID = 4 (th2) xvarID = 3 (th1)	The variable labels on the graph should match the selections	Passed
33				yvarID = 1 (om1) xvarID = 3 (th1)	The variable labels on the graph should match the selections	Passed
34	DP Window, Double Pendulum	`DPWindow.pendulumLoadState(self), DoublePendulum.saveState(self)`	Using Python's "pickle" module, the DoublePendulum object saves the pendulum's current attributes to disk. Using the same module, the DPWindow object loads the save from file and recreates the pendulum with the loaded attributes. Note this also tests proper connections for the Save state and Load state buttons.	Start with th1 = pi/2, all other values default. Run for 5 seconds, save state, run for a further 5, load state	Simulation should return to the same position as it was at the end of the first five seconds	Passed
35				Attempt to load state when there is no save file present	Simulation should continue as normal	Passed
	DP Window	`initConnections(self)`	The following tests will ensure that all the buttons are correctly connected to their respective functions			Passed
36			The start button should start the simulation	Set (unimportant) initial conditions, press start	Show that start button starts graph and pendulum widget	Passed
37			The stop button should pause the simulation	Set (unimportant) initial conditions, press stop	Show that simulation stops	Passed
38			The step button should advance the simulation one step	Set (unimportant) initial conditions,	Show that pendulum advances one tick	Passed

				press step		
39			The Reset button should set the pendulum to its initial conditions	Set conditions, allow pendulum to run, press reset	Pendulum should return to state it was in before simulation was run	Passed
40			The defaults button should set the pendulum to its conditions at start up	Vary all attributes (including graph axis), press defaults	Pendulum should return to Th1 = Th2 = Om1 = Om2 = 0, L1 = L2 = M1 = M2 = 1, g = 9.81, h = 0.004, x-axis-var = om1, y-axis-var = om2	Passed
41	n/a	n/a	This is a test to check that the pendulum runs in “real time”	Run pendulum three times for 10 seconds, timing with a stopwatch	Stopwatch time should be close to 10 seconds (bearing in mind human error)	Passed
42	DP Window		The help button should bring up a help window	Press the help button	Help window should be brought up	Passed
						Passed

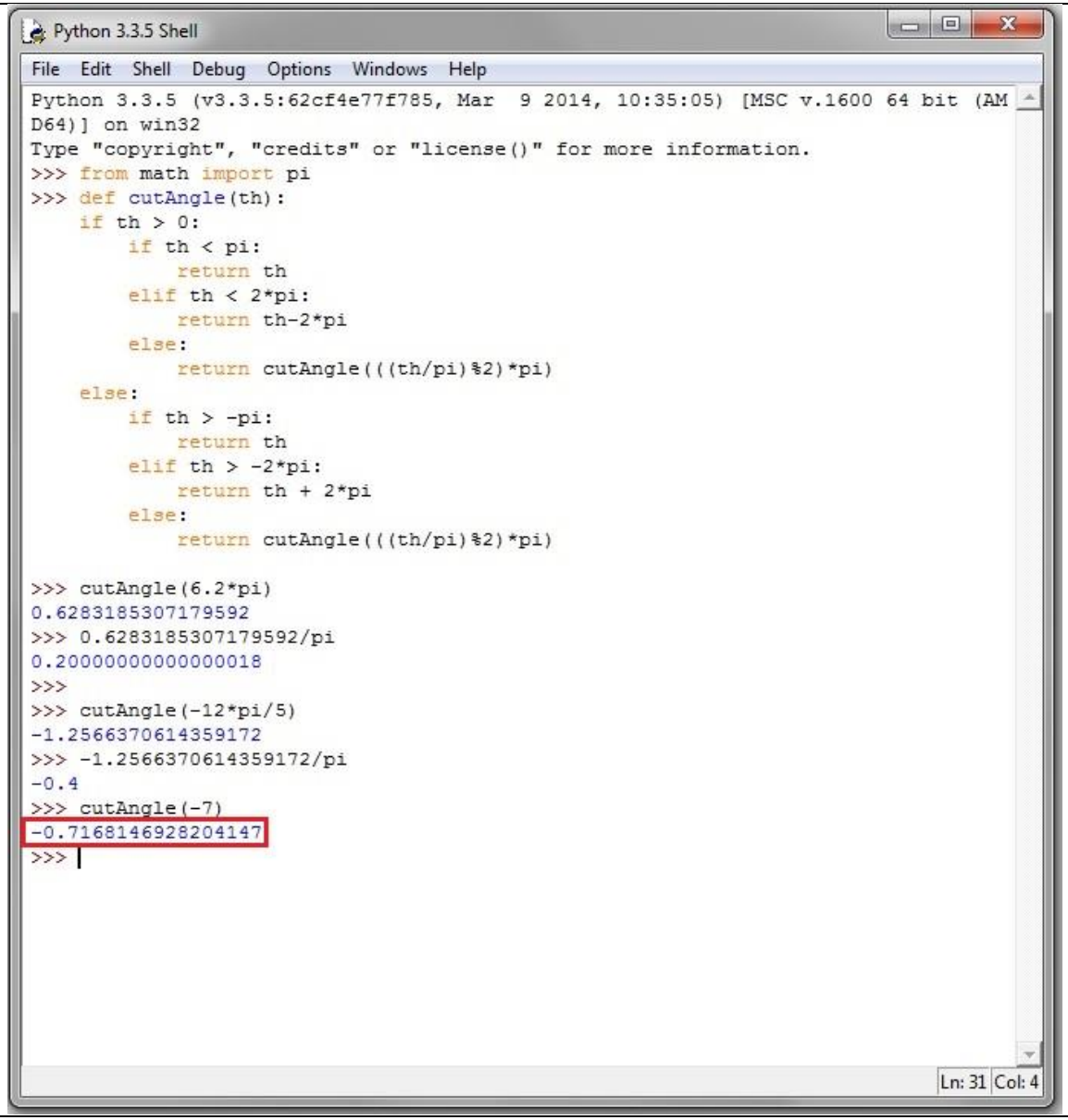
Test Number	**Screenshot**	**Comment**

1		The error caused by this function was a syntax error. The cause of the error is that the function recursively calls itself, but tries to do so by using “return self.cutAngle(…) ”. This is because the function was originally a method of the DoublePendulum object. It was later moved so that other objects could use it without needing their own definition of it. When it was moved however, the recursive calling did not have the “self”s removed from the function.

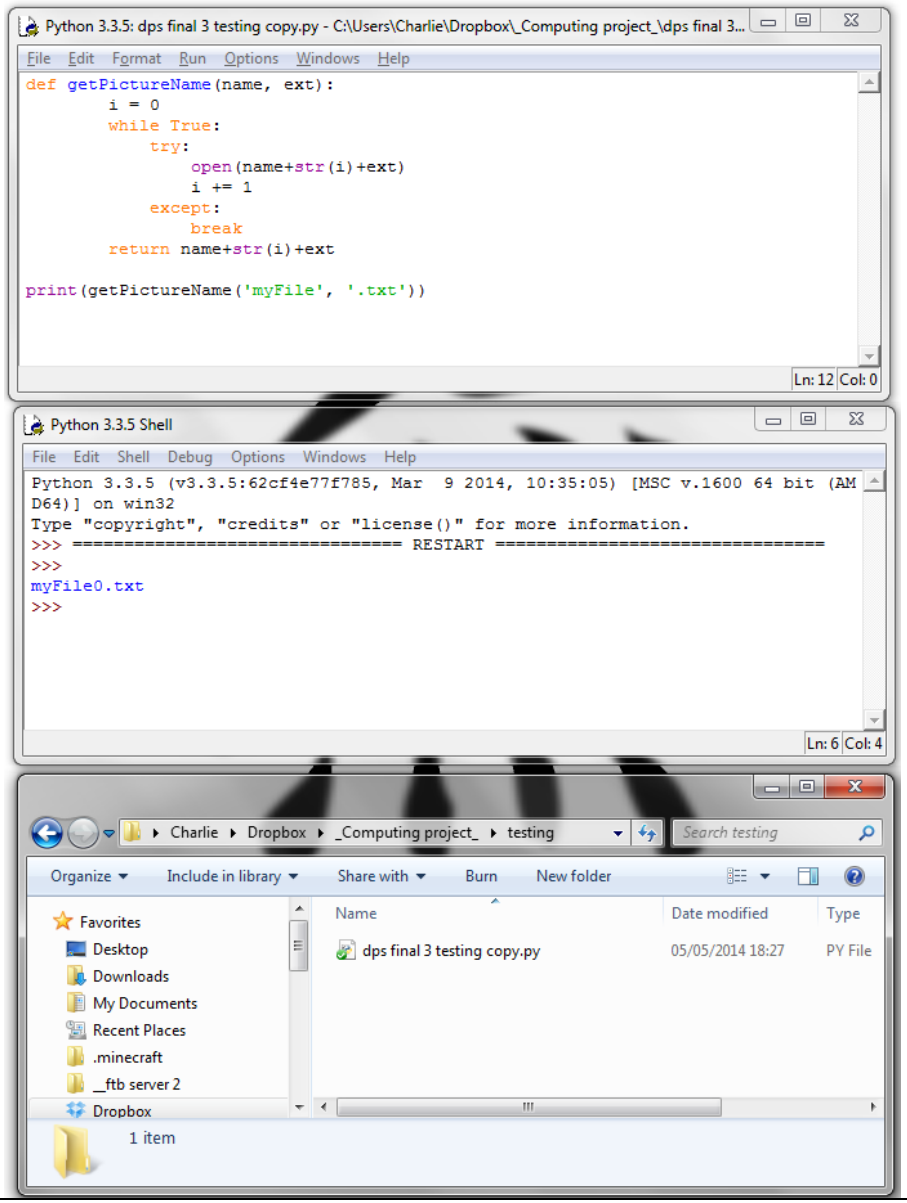
1-2			Amended function provides correct results

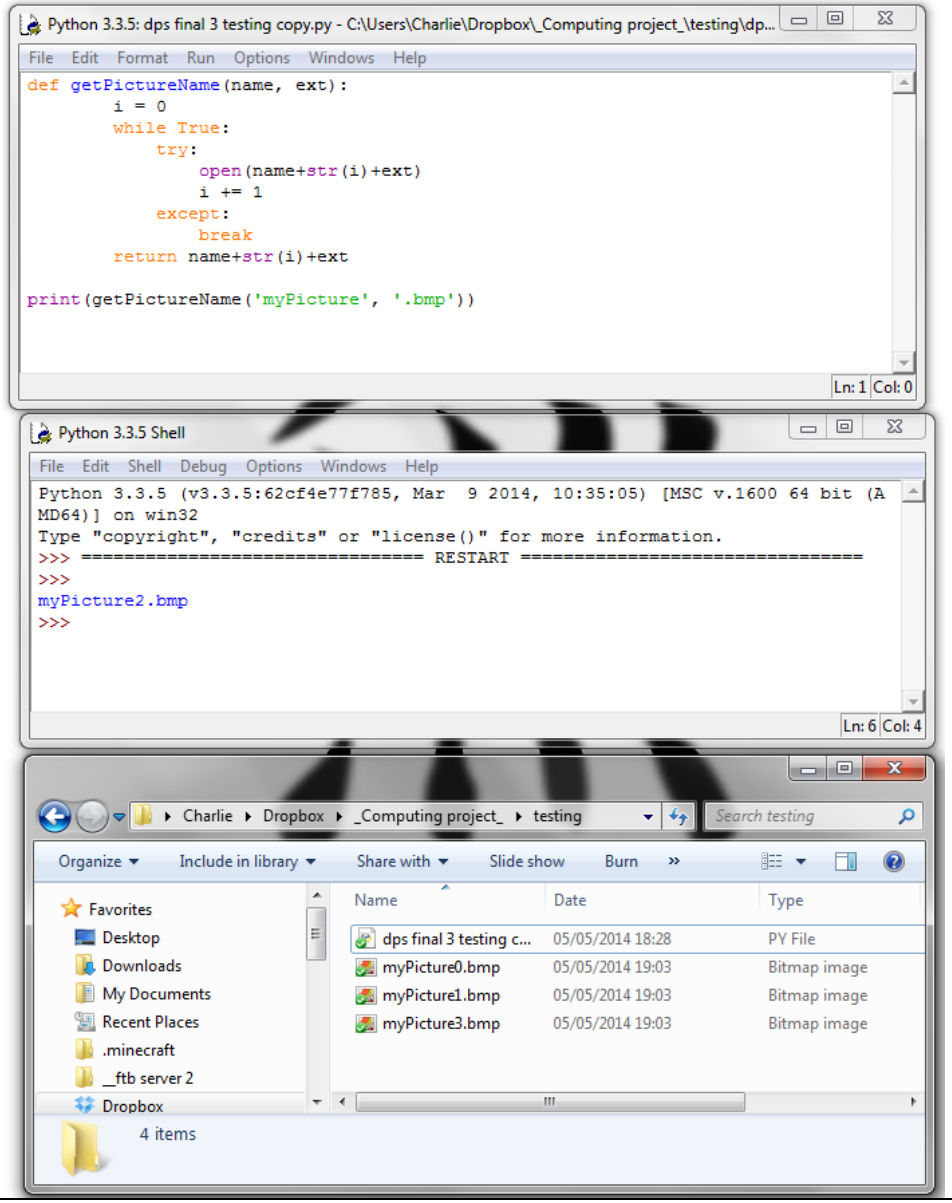
2			Passed

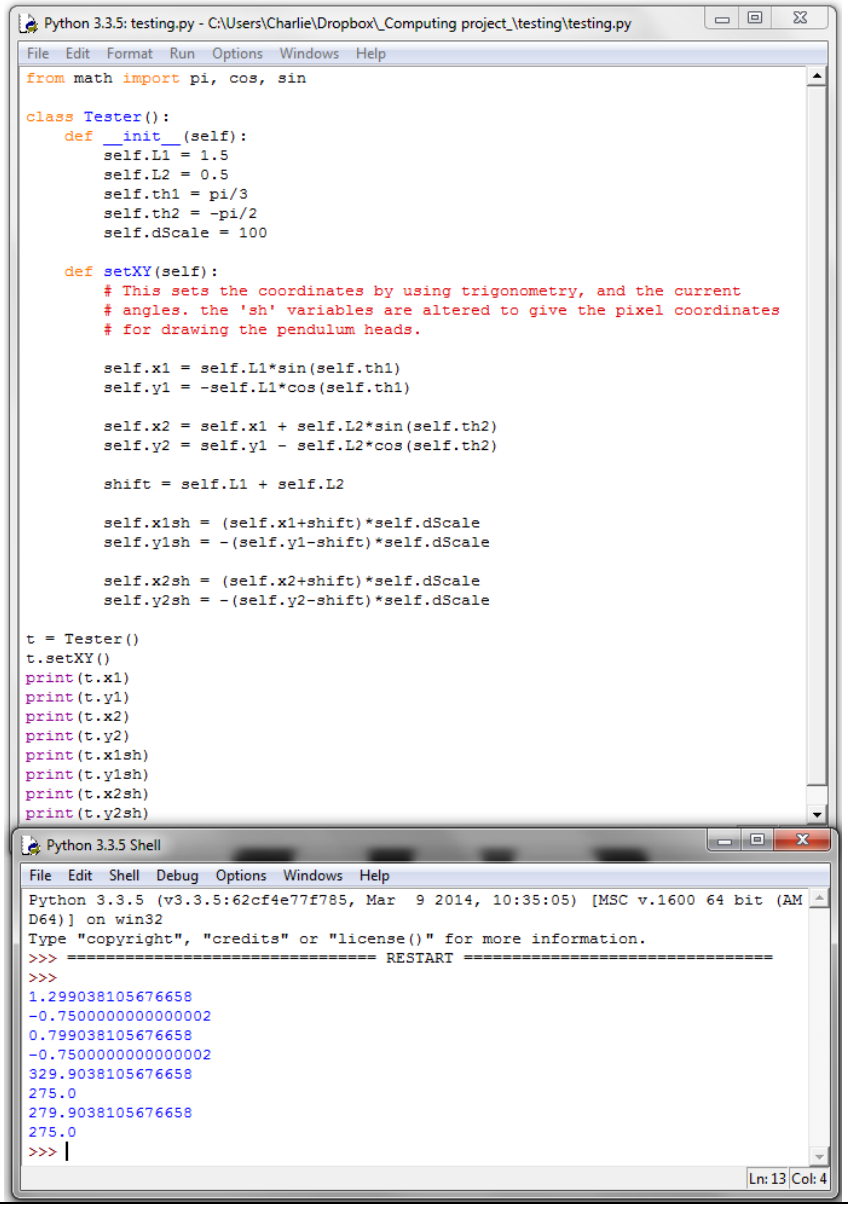
Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> from math import pi
>>> def cutAngle(th):
if th > 0:
if th < pi:
return th
elif th < 2*pi:
return th-2*pi
else:
return cutAngle(((th/pi)%2)*pi)
else:
if th > -pi:
return th
elif th > -2*pi:
return th + 2*pi
else:
return cutAngle(((th/pi)%2)*pi)
>>> cutAngle(6.2*pi)
0.6283185307179592
>>> 0.6283185307179592/pi
0.20000000000000018
>>>
>>> cutAngle(-12*pi/5)
-1.2566370614359172
>>> -1.2566370614359172/pi
-0.4
>>> cutAngle(-7)
-0.7168146928204147
>>>
Ln: 31 Col: 4

3			Passed

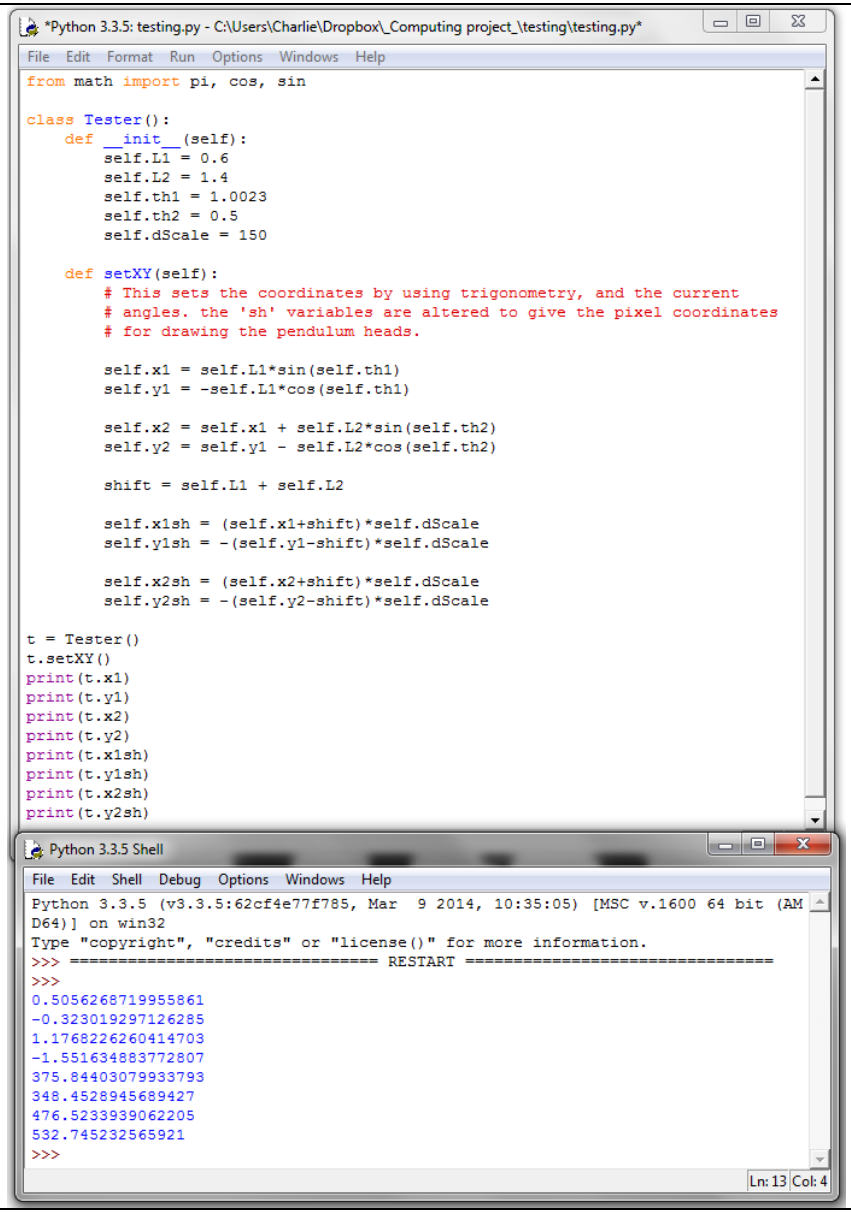
Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> from math import pi
>>> def cutAngle(th):
if th > 0:
if th < pi:
return th
elif th < 2*pi:
return th-2*pi
else:
return cutAngle(((th/pi)%2)*pi)
else:
if th > -pi:
return th
elif th > -2*pi:
return th + 2*pi
else:
return cutAngle(((th/pi)%2)*pi)
>>> cutAngle(6.2*pi)
0.6283185307179592
>>> 0.6283185307179592/pi
0.20000000000000018
>>>
>>> cutAngle(-12*pi/5)
-1.2566370614359172
>>> -1.2566370614359172/pi
-0.4
>>> cutAngle(-7)
-0.7168146928204147
>>>
Ln: 31 Col: 4

4	 Python 3.3.5: dps final 3 testing copy.py - C:\Users\Charlie\Dropbox_Computing project_\dps final 3...
	File Edit Format Run Options Windows Help
	def getPictureName(name, ext):
	i = 0
	while True:
	try:
	open(name+str(i)+ext)
	i += 1
	except:
	break
	return name+str(i)+ext
	print(getPictureName('myFile', '.txt'))
	Ln: 12 Col: 0
	Python 3.3.5 Shell
	File Edit Shell Debug Options Windows Help
	Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AM
	D64)] on win32
	Type "copyright", "credits" or "license()" for more information.
	>>> ========== RESTART ==========
	>>>
	myFile0.txt
	>>>
	Ln: 6 Col: 4
	Charlie Dropbox _Computing project_ testing
	Search testing
	Organize Include in library Share with Burn New folder
	Favorites
	Desktop
	Downloads
	My Documents
	Recent Places
	.minecraft
	_ftb server 2
	Dropbox
	Name
	Date modified
	Type
	dps final 3 testing copy.py
	05/05/2014 18:27
	PY File
	1 item
		Passed

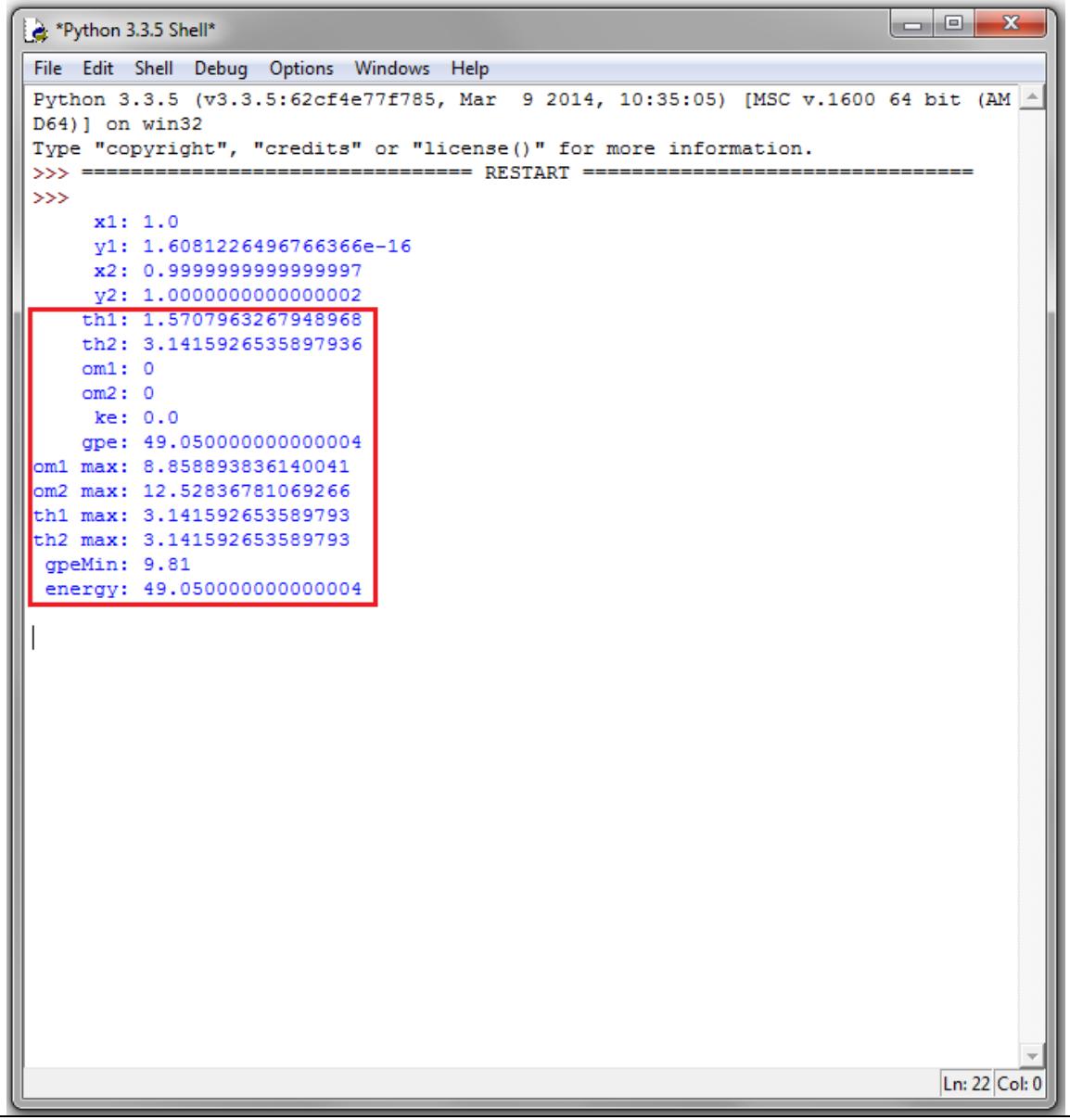
5	 Python 3.3.5: dps final 3 testing copy.py - C:\Users\Charlie\Dropbox_Computing project_\testing\dp...
	File Edit Format Run Options Windows Help
	def getPictureName(name, ext):
	i = 0
	while True:
	try:
	open(name+str(i)+ext)
	i += 1
	except:
	break
	return name+str(i)+ext
	print(getPictureName('myPicture', '.bmp'))
	Ln: 1 Col: 0
	Python 3.3.5 Shell
	File Edit Shell Debug Options Windows Help
	Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
	Type "copyright", "credits" or "license()" for more information.
	>>> ========== RESTART ==========
	>>>
	myPicture2.bmp
	>>>
	Ln: 6 Col: 4
	Charlie ▸ Dropbox ▸ _Computing project_ ▸ testing
	Search testing
	Organize Include in library Share with Slide show Burn
	Favorites
	Desktop
	Downloads
	My Documents
	Recent Places
	.minecraft
	__ftb server 2
	Dropbox
	Name Date Type
	dps final 3 testing c... 05/05/2014 18:28 PY File
	myPicture0.bmp 05/05/2014 19:03 Bitmap image
	myPicture1.bmp 05/05/2014 19:03 Bitmap image
	myPicture3.bmp 05/05/2014 19:03 Bitmap image
	4 items
		Passed

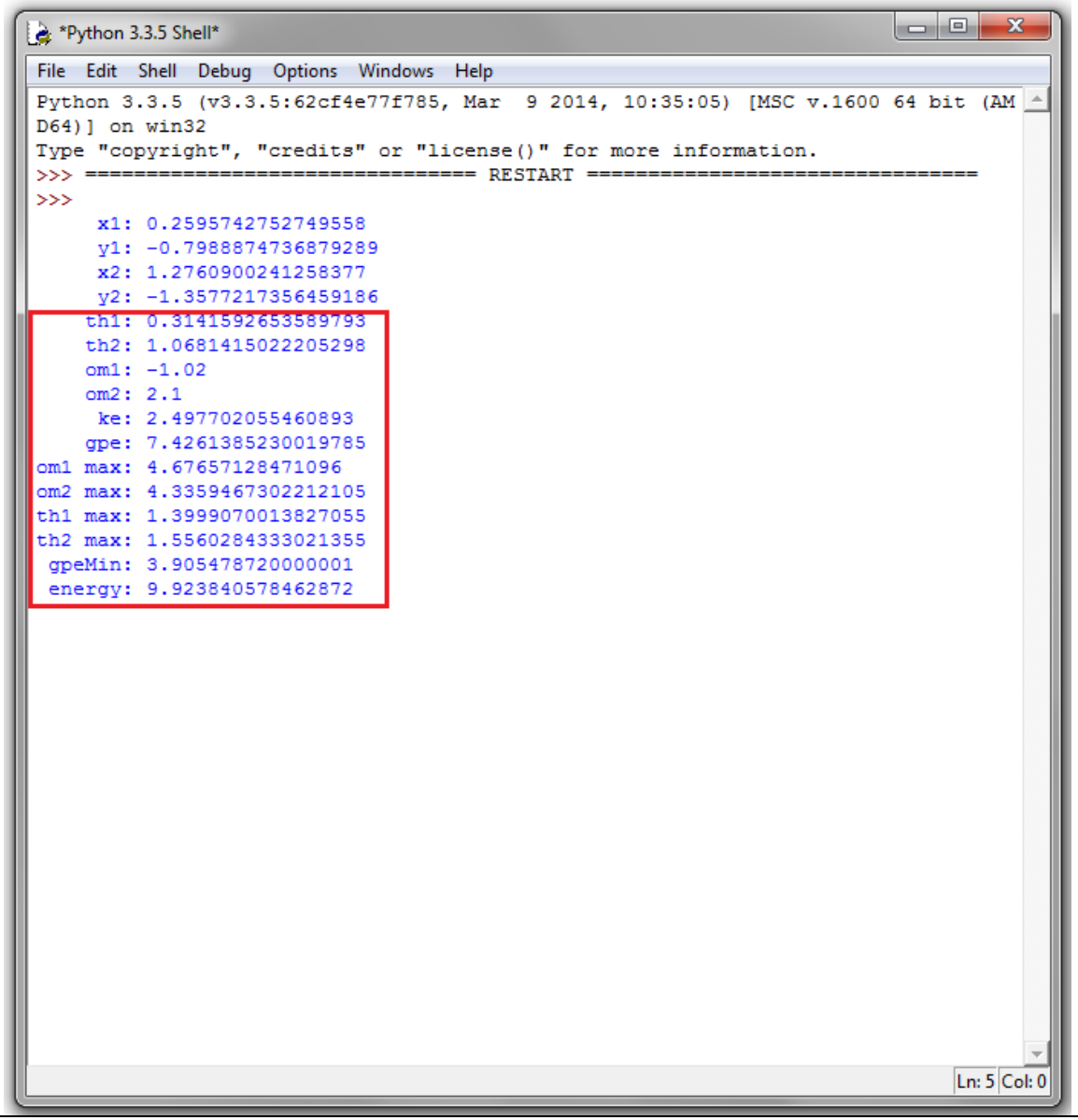
6			Passed

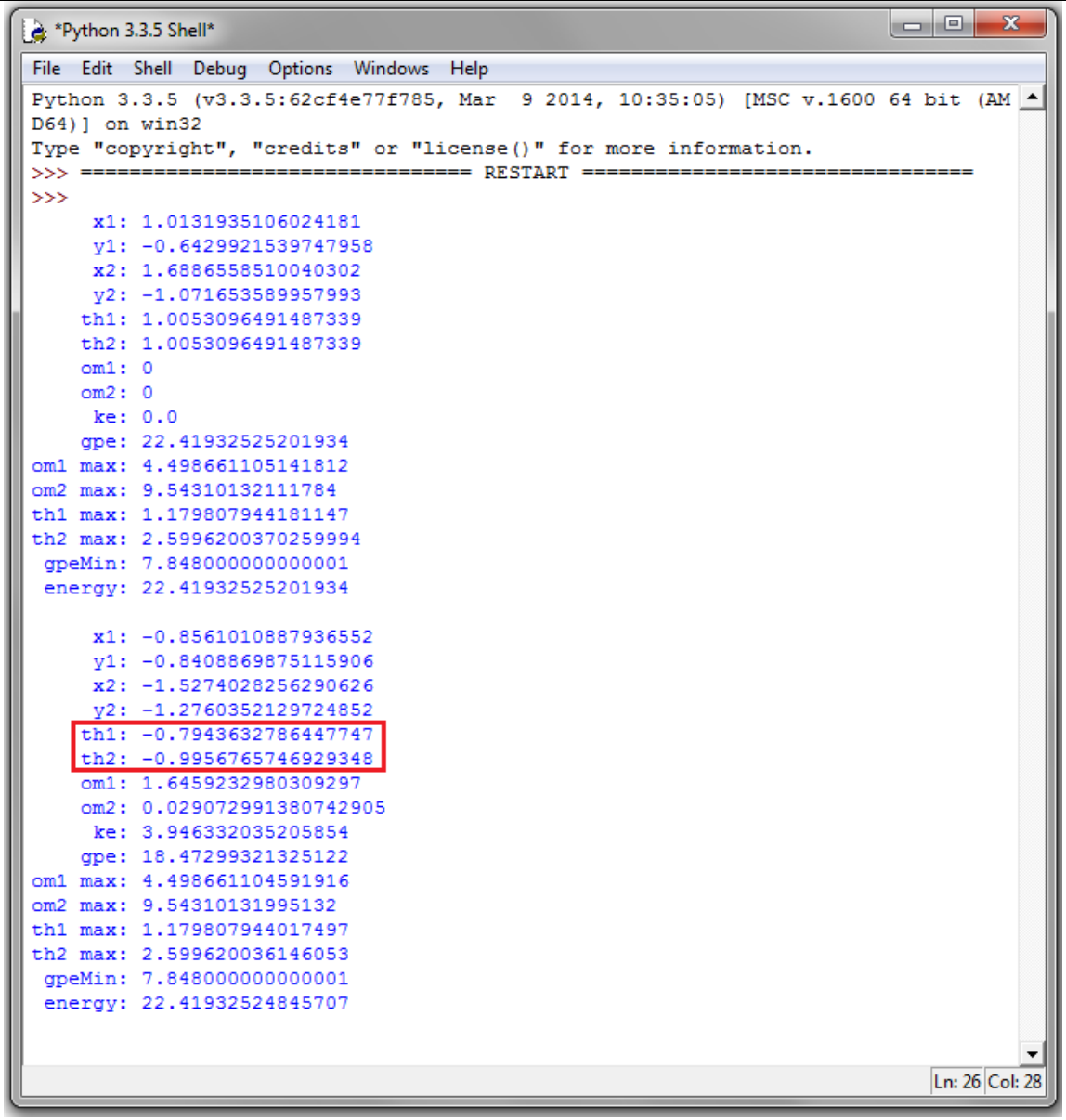
Python 3.3.5: testing.py - C:\Users\Charlie\Dropbox_Computing project_\testing\testing.py
File Edit Format Run Options Windows Help
from math import pi, cos, sin
class Tester():
def __init__(self):
self.L1 = 1.5
self.L2 = 0.5
self.th1 = pi/3
self.th2 = -pi/2
self.dScale = 100
def setXY(self):
# This sets the coordinates by using trigonometry, and the current
# angles. the 'sh' variables are altered to give the pixel coordinates
# for drawing the pendulum heads.
self.x1 = self.L1*sin(self.th1)
self.y1 = -self.L1*cos(self.th1)
self.x2 = self.x1 + self.L2*sin(self.th2)
self.y2 = self.y1 - self.L2*cos(self.th2)
shift = self.L1 + self.L2
self.x1sh = (self.x1+shift)*self.dScale
self.y1sh = -(self.y1-shift)*self.dScale
self.x2sh = (self.x2+shift)*self.dScale
self.y2sh = -(self.y2-shift)*self.dScale
t = Tester()
t.setXY()
print(t.x1)
print(t.y1)
print(t.x2)
print(t.y2)
print(t.x1sh)
print(t.y1sh)
print(t.x2sh)
print(t.y2sh)
Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
1.299038105676658
-0.7500000000000002
0.799038105676658
-0.7500000000000002
329.9038105676658
275.0
279.9038105676658
275.0
>>>
Ln: 13 Col: 4

7			Passed

Python 3.3.5: testing.py - C:\Users\Charlie\Dropbox_Computing project_\testing\testing.py
File Edit Format Run Options Windows Help
from math import pi, cos, sin
class Tester():
def __init__(self):
self.L1 = 0.6
self.L2 = 1.4
self.th1 = 1.0023
self.th2 = 0.5
self.dScale = 150
def setXY(self):
# This sets the coordinates by using trigonometry, and the current
# angles. the 'sh' variables are altered to give the pixel coordinates
# for drawing the pendulum heads.
self.x1 = self.L1*sin(self.th1)
self.y1 = -self.L1*cos(self.th1)
self.x2 = self.x1 + self.L2*sin(self.th2)
self.y2 = self.y1 - self.L2*cos(self.th2)
shift = self.L1 + self.L2
self.x1sh = (self.x1+shift)*self.dScale
self.y1sh = -(self.y1-shift)*self.dScale
self.x2sh = (self.x2+shift)*self.dScale
self.y2sh = -(self.y2-shift)*self.dScale
t = Tester()
t.setXY()
print(t.x1)
print(t.y1)
print(t.x2)
print(t.y2)
print(t.x1sh)
print(t.y1sh)
print(t.x2sh)
print(t.y2sh)
Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
0.5056268719955861
-0.323019297126285
1.1768226260414703
-1.551634883772807
375.84403079933793
348.4528945689427
476.5233939062205
532.745232565921
>>>
Ln: 13 Col: 4

8	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32 Type "copyright", "credits" or "license()" for more information. >>> ========== RESTART ========== >>> x1: 1.0 y1: 1.6081226496766366e-16 x2: 0.9999999999999997 y2: 1.0000000000000002 th1: 1.5707963267948968 th2: 3.1415926535897936 om1: 0 om2: 0 ke: 0.0 gpe: 49.050000000000004 om1 max: 8.858893836140041 om2 max: 12.52836781069266 th1 max: 3.141592653589793 th2 max: 3.141592653589793 gpeMin: 9.81 energy: 49.050000000000004 Ln: 22 Col: 0		Passed

9	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32 Type "copyright", "credits" or "license()" for more information. >>> ========== RESTART ========== >>> x1: 0.2595742752749558 y1: -0.7988874736879289 x2: 1.2760900241258377 y2: -1.3577217356459186 th1: 0.3141592653589793 th2: 1.0681415022205298 om1: -1.02 om2: 2.1 ke: 2.497702055460893 gpe: 7.4261385230019785 om1 max: 4.67657128471096 om2 max: 4.3359467302212105 th1 max: 1.3999070013827055 th2 max: 1.5560284333021355 gpeMin: 3.905478720000001 energy: 9.923840578462872 Ln: 5 Col: 0		Passed

10		Passed. The reason the results from WolframAlpha are slightly different will be because they are using a different value for h. As this can't be changed on their website, the results will always be slightly different.

10 (cont.)

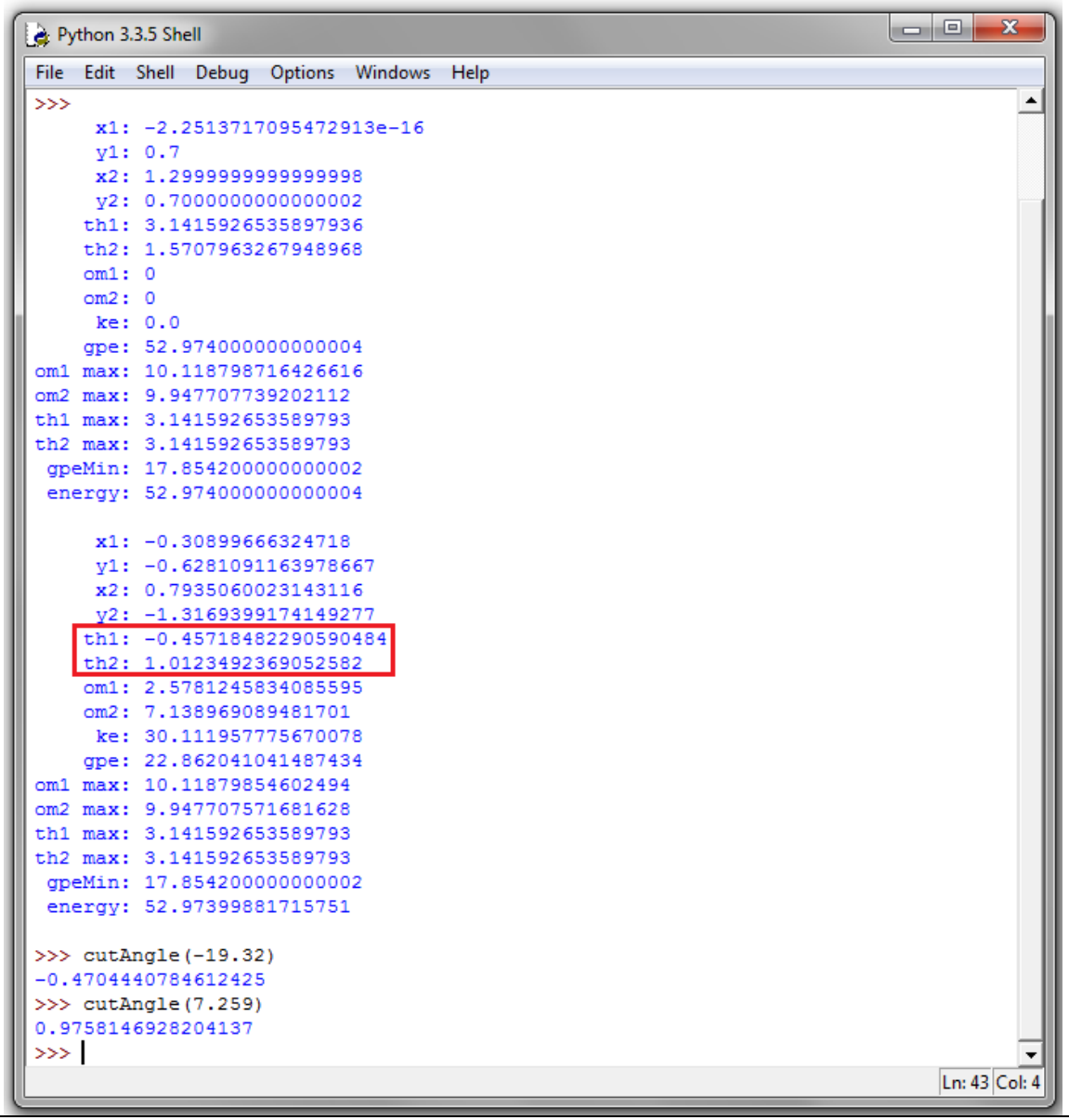
Input information:

double pendulum	
initial angle from vertical 1	1.005309649148734 radians
length 1	1.2 meters
mass 1	1 kg (kilogram)
initial angle from vertical 2	1.005309649148734 radians
length 2	0.8 meters
mass 2	1 kg (kilogram)
time	10 seconds

Result:

More units

final angle from vertical 1	– 803.7 mrad (milliradians) = – 46.05° (degrees) = –46 degrees 2 arc minutes 52.4 arc seconds
final angle from vertical 2	– 0.9958 radians = – 57.05° (degrees) = –57 degrees 3 arc minutes 11.23 arc seconds

11			Passed

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
>>>
x1: -2.2513717095472913e-16
y1: 0.7
x2: 1.2999999999999998
y2: 0.7000000000000002
th1: 3.1415926535897936
th2: 1.5707963267948968
om1: 0
om2: 0
ke: 0.0
gpe: 52.974000000000004
om1 max: 10.118798716426616
om2 max: 9.947707739202112
th1 max: 3.141592653589793
th2 max: 3.141592653589793
gpeMin: 17.854200000000002
energy: 52.974000000000004
x1: -0.30899666324718
y1: -0.6281091163978667
x2: 0.7935060023143116
y2: -1.3169399174149277
th1: -0.45718482290590484
th2: 1.0123492369052582
om1: 2.5781245834085595
om2: 7.138969089481701
ke: 30.111957775670078
gpe: 22.862041041487434
om1 max: 10.11879854602494
om2 max: 9.947707571681628
th1 max: 3.141592653589793
th2 max: 3.141592653589793
gpeMin: 17.854200000000002
energy: 52.97399881715751
>>> cutAngle(-19.32)
-0.4704440784612425
>>> cutAngle(7.259)
0.9758146928204137
>>>
Ln: 43 Col: 4

11 (cont.)

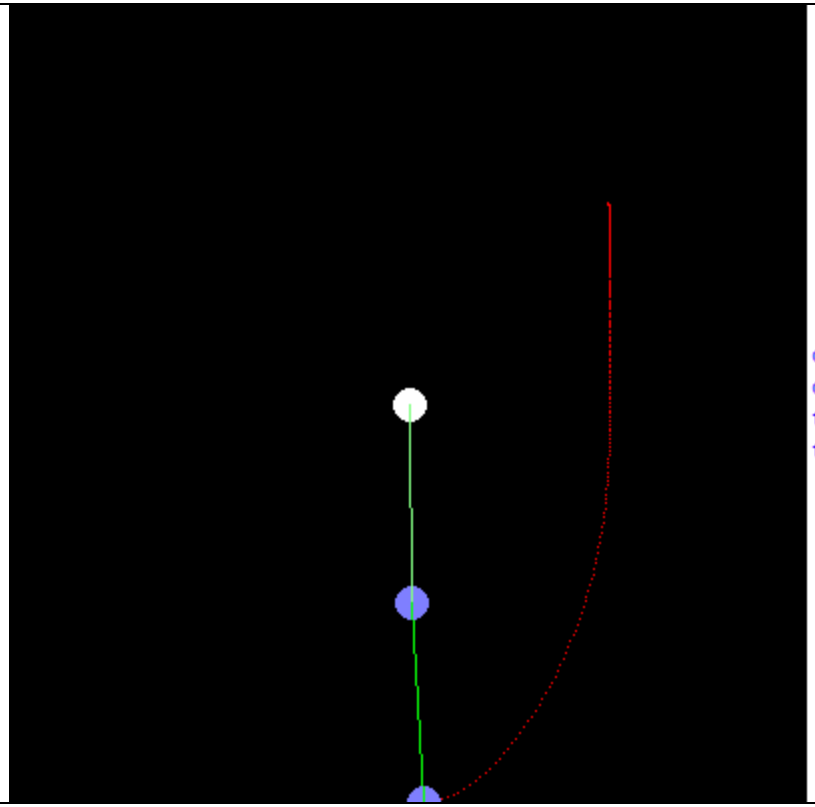
Input information:

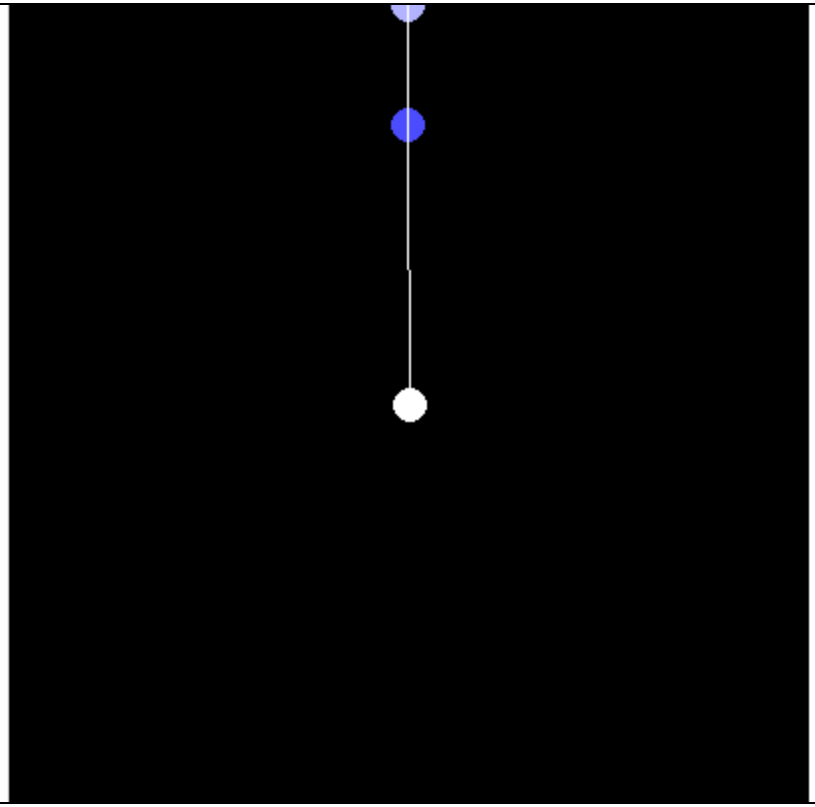
double pendulum	
initial angle from vertical 1	π radians
length 1	0.7 meters
mass 1	1.4 kg (kilograms)
initial angle from vertical 2	$\frac{\pi}{2}$ radians
length 2	1.3 meters
mass 2	0.6 kg (kilograms)
time	6 seconds

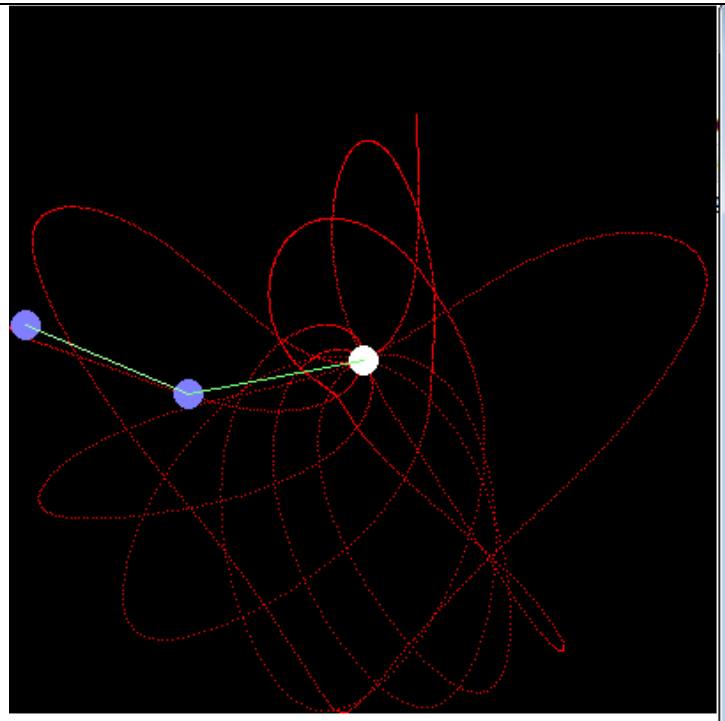
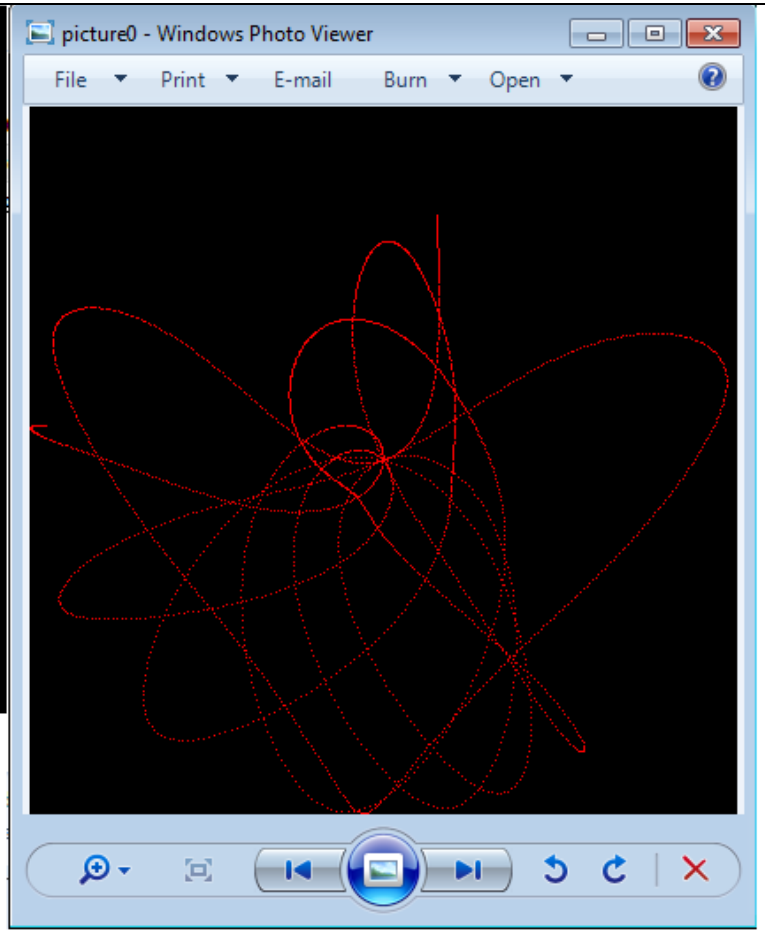
Result:

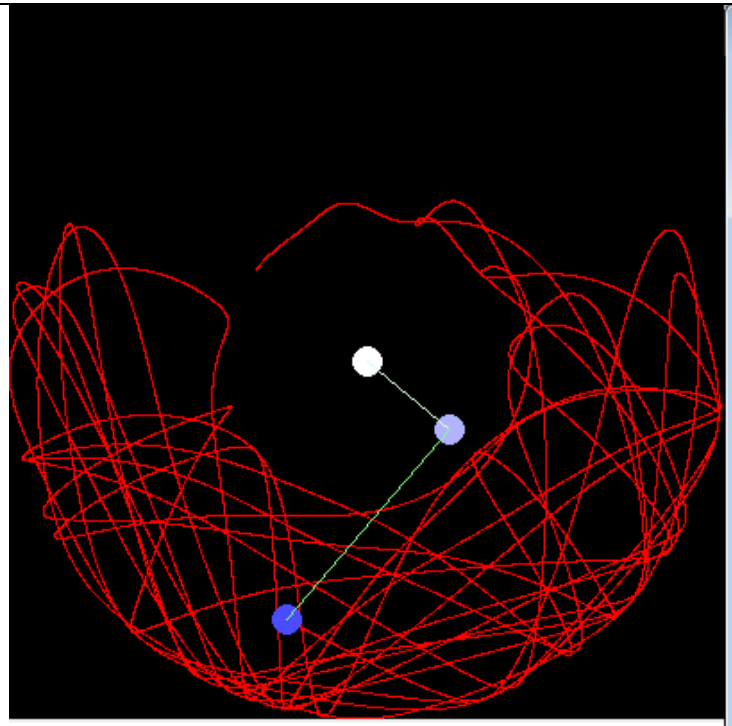
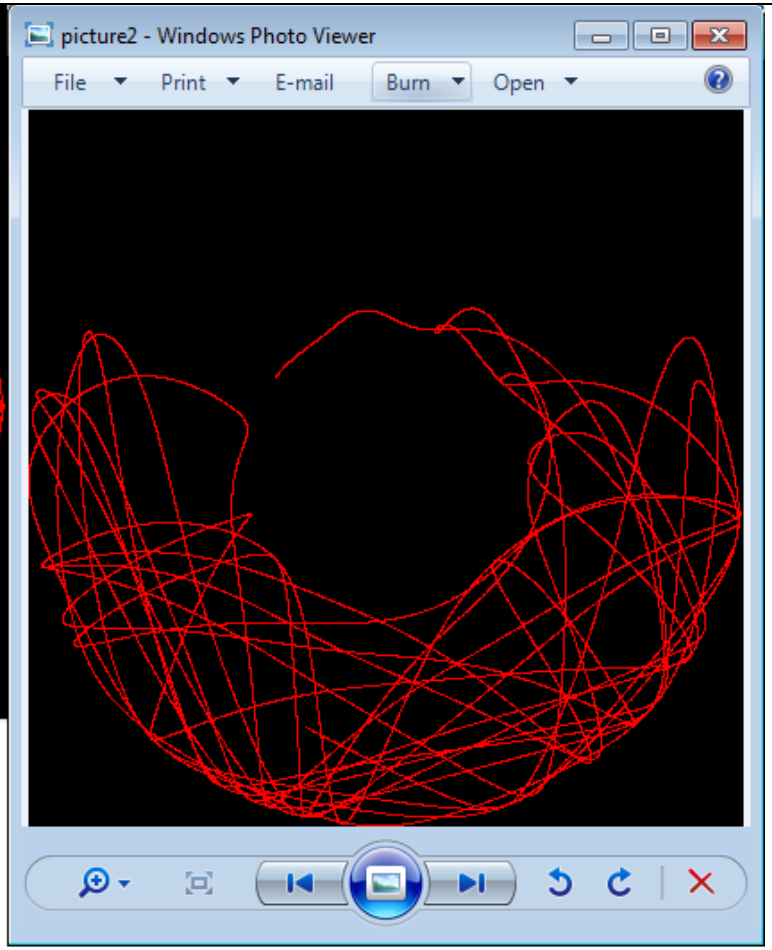
More units

final angle from vertical 1	– 19.32 radians = – 3.075 rev (revolutions)
final angle from vertical 2	7.259 radians = 415.9° (degrees) = 415 degrees 55 arc minutes 25.5 arc seconds

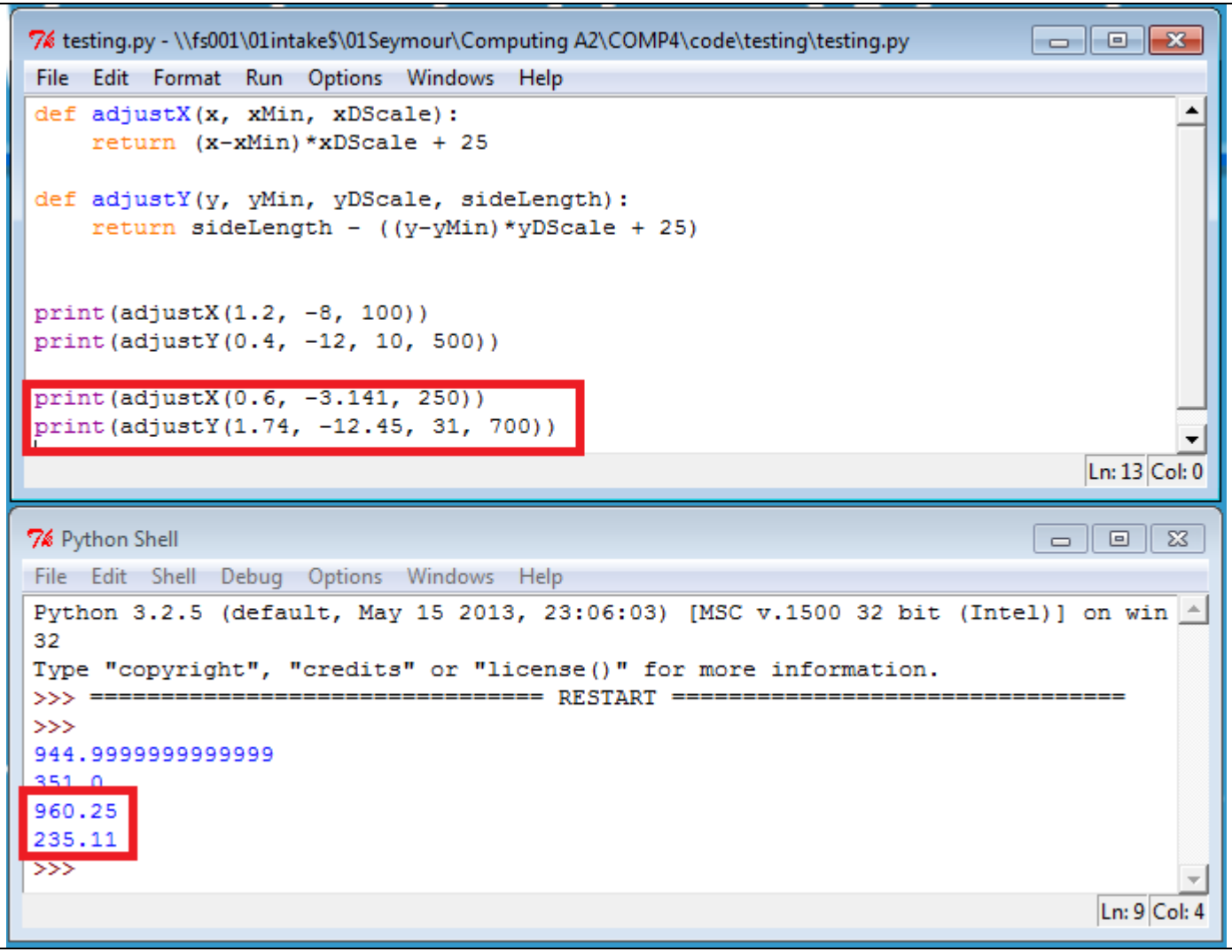
12		 x1: -0.0023201351308974115 y1: -0.999997308482865 x2: 0.10166289560286708 y2: -1.9945763799092755 th1: -0.002320137212460806 th2: 0.10417133396927715 om1: 4.229142030031648 om2: -11.969580512457274 ke: 39.186776309898214 gpe: 9.863232116873101 om1 max: 8.858894787361606 om2 max: 12.528369155923096 th1 max: 3.141592653589793 th2 max: 3.141592653589793 gpeMin: 9.81 energy: 49.05000842677131	Passed.

13		x1: -7.931754456707171e-16 y1: 1.4 x2: -1.133107779529596e-15 y2: 2.0 th1: -3.1415926535897927 th2: -3.1415926535897927 om1: 2.124016845193429e-14 om2: 0.0 ke: 8.842437214984274e-28 gpe: 70.23960000000001 om1 max: 6.722270545195192 om2 max: 28.637305007546335 th1 max: 3.141592653589793 th2 max: 3.141592653589793 gpeMin: 8.240400000000001 energy: 70.23960000000001	Passed. Note that that x1, x2, om1 and ke are considered “negligible”; i.e. they are so small they can be thought of as 0. The fact that they are not 0 will be a combination of floating point inaccuracies and the approximate nature of numerical integration.

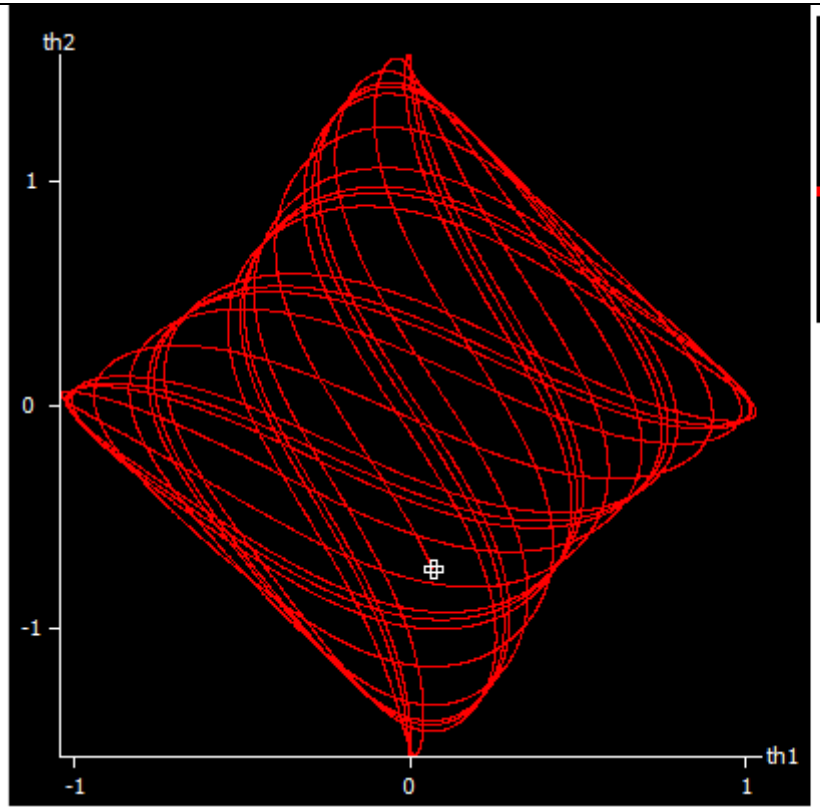
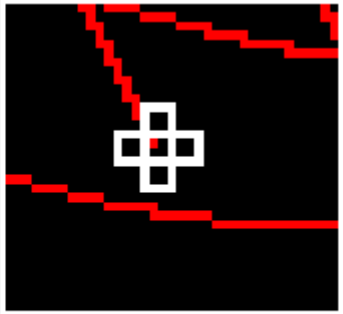
14	  >>> x1: 0.844327925502015 y1: 0.5358267949789969 x2: 0.3085011305230182 y2: 1.3801547204810118 th1: 2.1362830044410597 th2: 3.7070793312359562 om1: 0 om2: 0 ke: 0.0 gpe: 58.035778666662694 om1 max: 9.820975375863917 om2 max: 13.888956572278955 th1 max: 3.141592653589793 th2 max: 3.141592653589793 gpeMin: 9.81 energy: 58.035778666662694	Passed. Screenshot of shell shows starting values

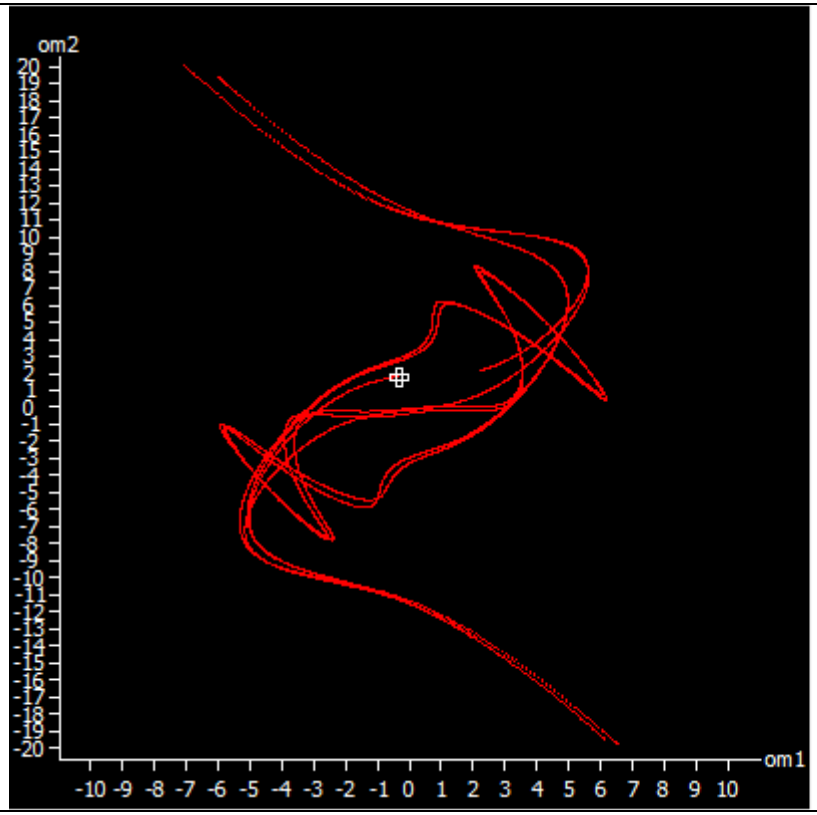
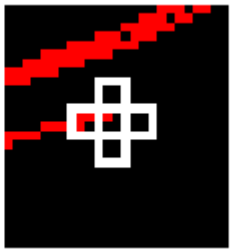
15	  x1: 0.4623079456654735 y1: -0.38245439384921376 x2: -0.6164105942206315 y2: 0.5099391917989515 th1: 0.8796459430051422 th2: 4.0212385965949355 om1: -3.0 om2: -3.0 ke: 5.003999999999996 gpe: 43.992378297970326 om1 max: 19.426017847867104 om2 max: 9.950799555947706 th1 max: 3.141592653589793 th2 max: 3.141592653589793 gpeMin: 8.2404 energy: 48.996378297970324	Passed. Screenshot of shell shows starting values.

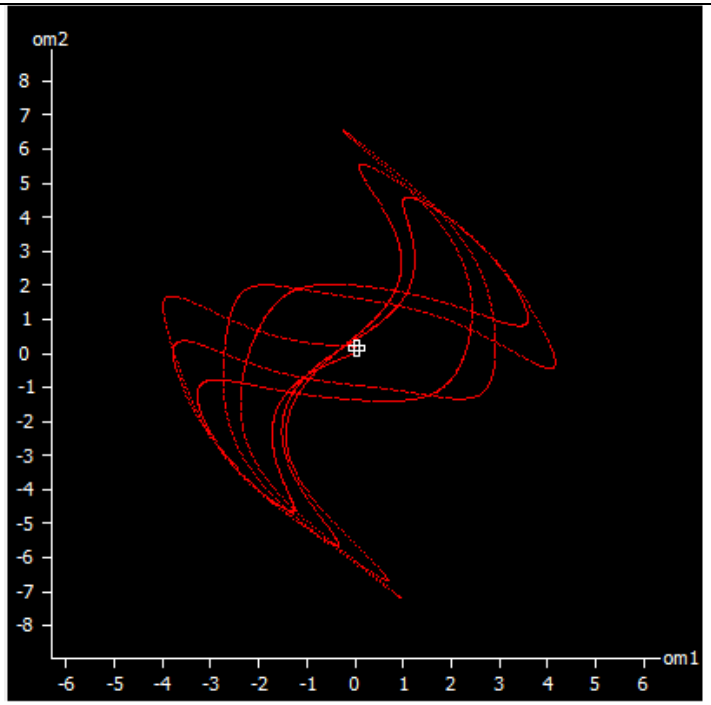
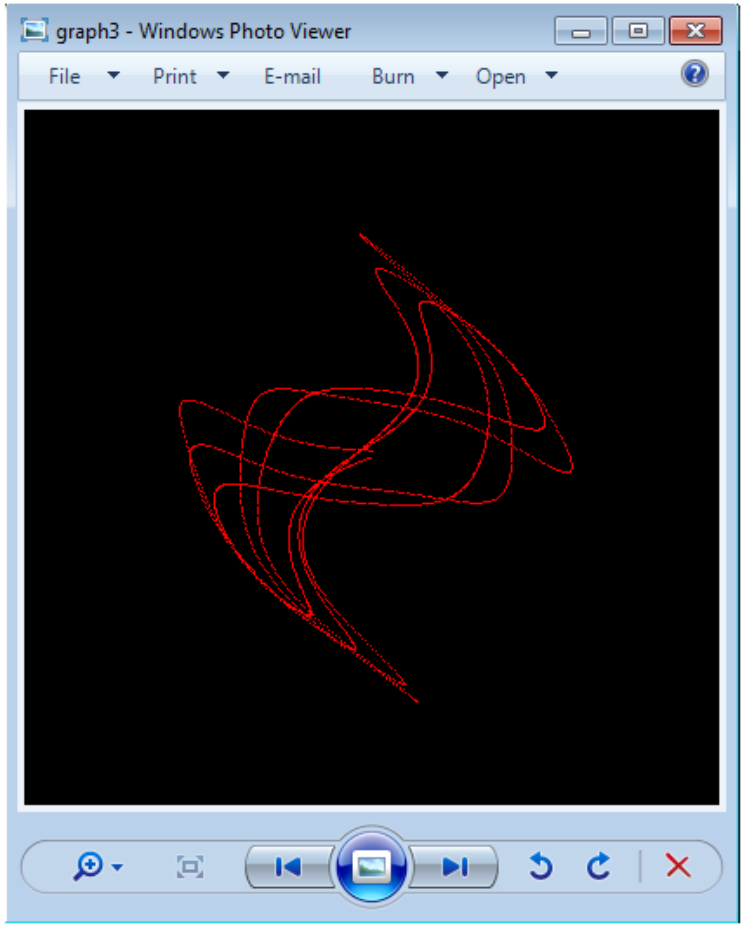
16	testing.py - \\fs001\01intake$\01Seymour\Computing A2\COMP4\code\testing\testing.py
	File Edit Format Run Options Windows Help
	`def adjustX(x, xMin, xDScale):`
	`    return (x-xMin)*xDScale + 25`
	
	`def adjustY(y, yMin, yDScale, sideLength):`
	`    return sideLength - ((y-yMin)*yDScale + 25)`
	
	`print(adjustX(1.2, -8, 100))`
	`print(adjustY(0.4, -12, 10, 500))`
	
	`print(adjustX(0.6, -3.141, 250))`
	`print(adjustY(1.74, -12.45, 31, 700))`
	Ln: 13 Col: 0
	
	Python Shell
	File Edit Shell Debug Options Windows Help
	`Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win32`
	`Type "copyright", "credits" or "license()" for more information.`
	`>>> ========== RESTART ==========`
	`>>>`
	`944.9999999999999`
	`351.0`
	`960.25`
	`235.11`
	`>>>`
	Ln: 9 Col: 4
			Passed. Note that PyQt automatically rounds any pixel coordinate to its nearest value, so these results can be safely entered into the function that draws the pixel

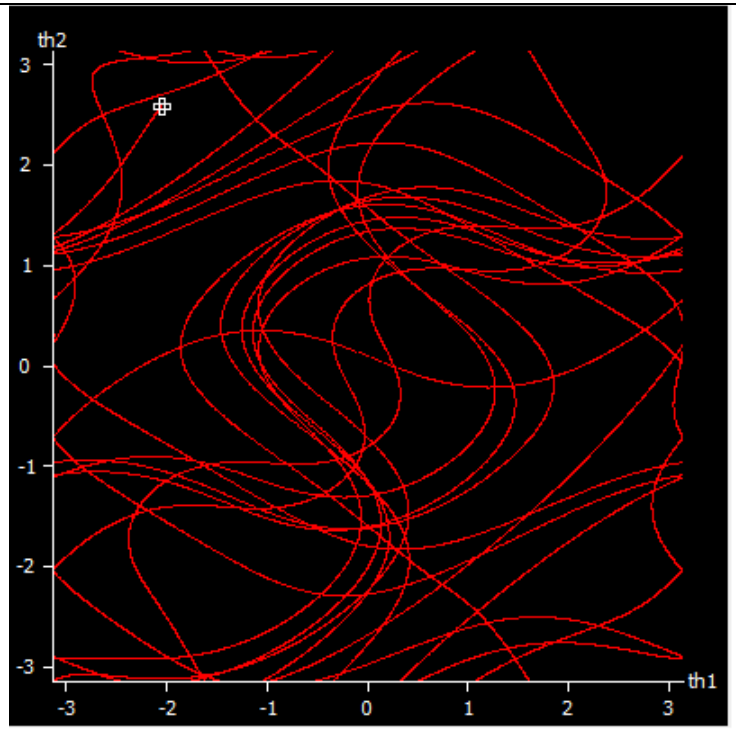
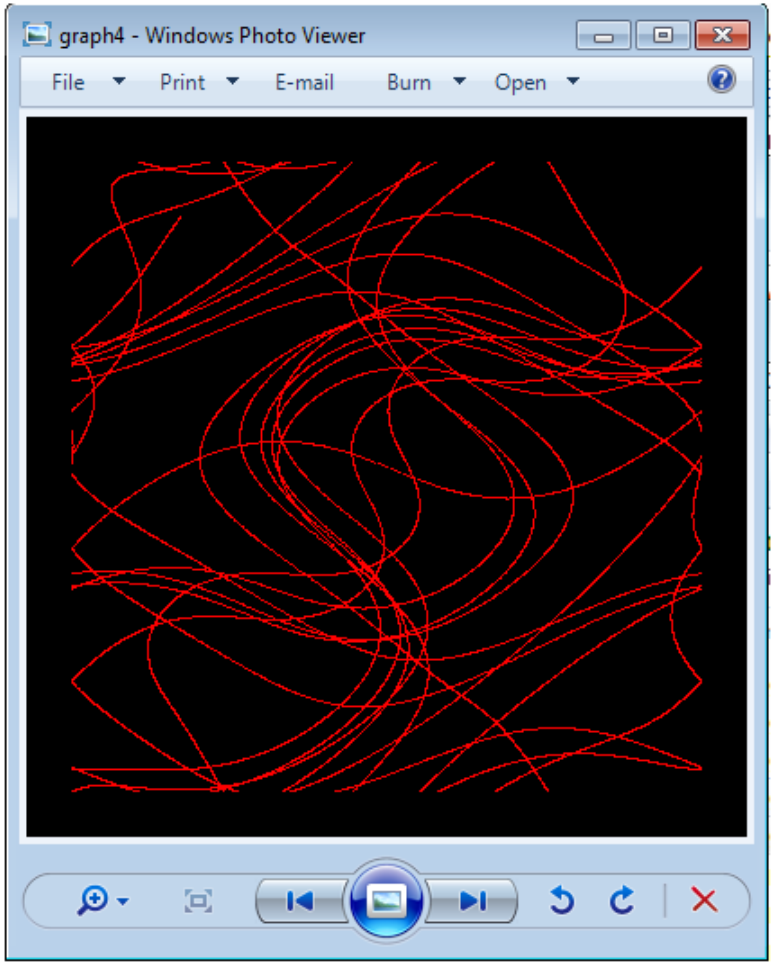
17			Passed

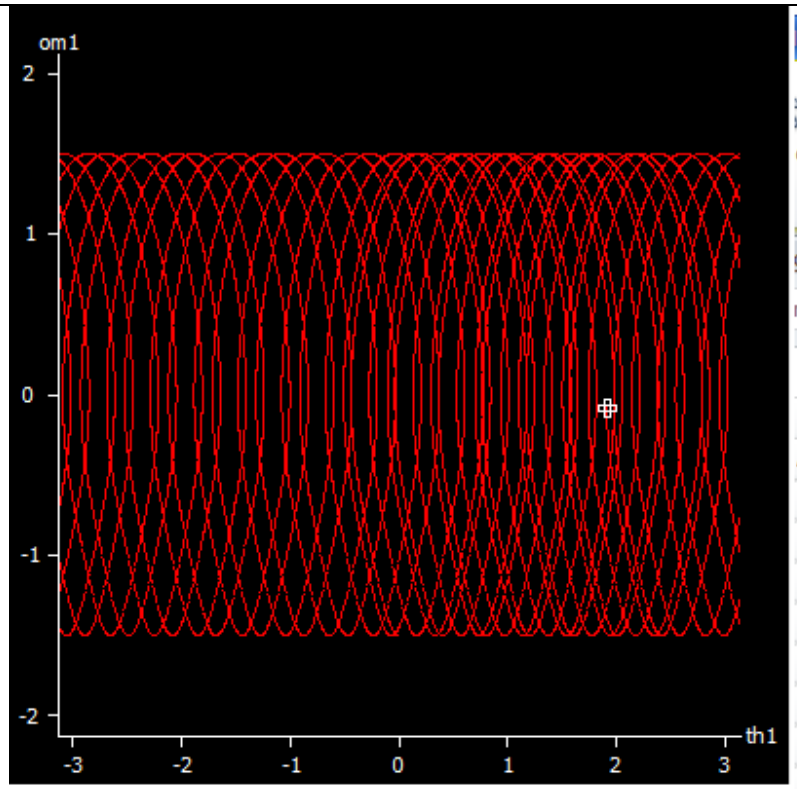
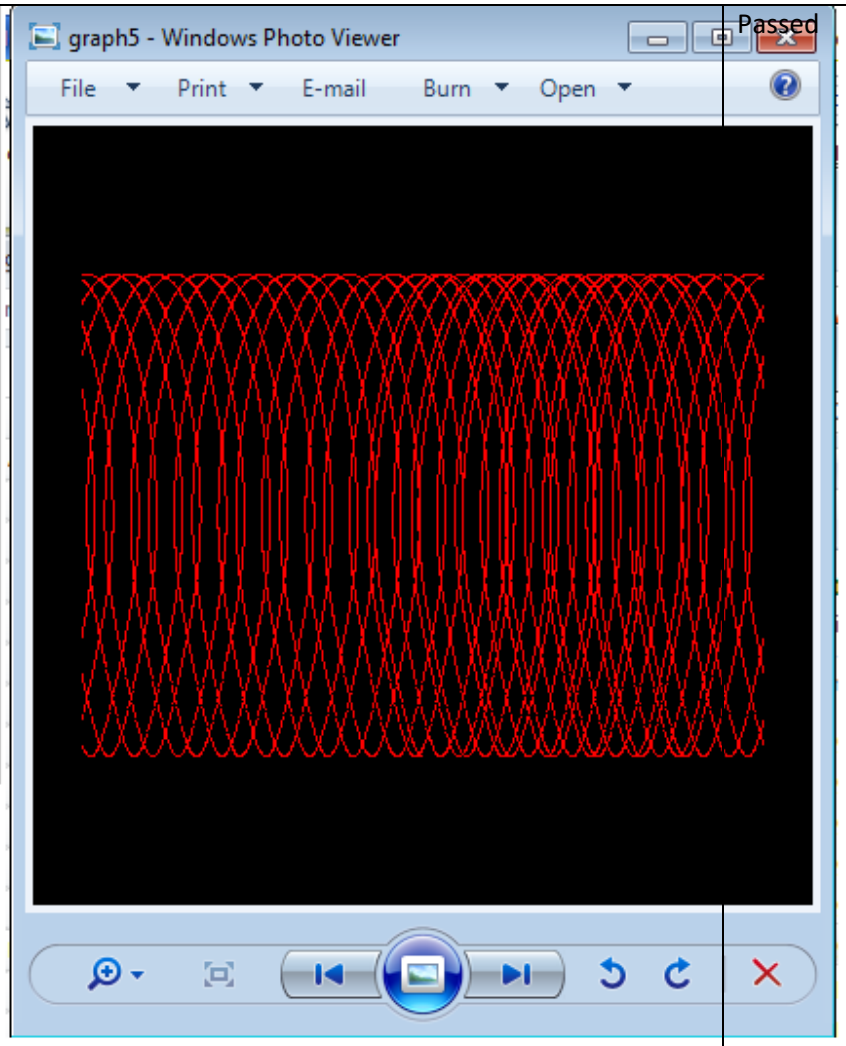
testing.py - \\fs001\01intake$\01Seymour\Computing A2\COMP4\code\testing\testing.py
File Edit Format Run Options Windows Help
def adjustX(x, xMin, xDScale):
return (x-xMin)*xDScale + 25
def adjustY(y, yMin, yDScale, sideLength):
return sideLength - ((y-yMin)*yDScale + 25)
print(adjustX(1.2, -8, 100))
print(adjustY(0.4, -12, 10, 500))
print(adjustX(0.6, -3.141, 250))
print(adjustY(1.74, -12.45, 31, 700))
Ln: 13 Col: 0
Python Shell
File Edit Shell Debug Options Windows Help
Python 3.2.5 (default, May 15 2013, 23:06:03) [MSC v.1500 32 bit (Intel)] on win
32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
944.9999999999999
351.0
960.25
235.11
>>>
Ln: 9 Col: 4

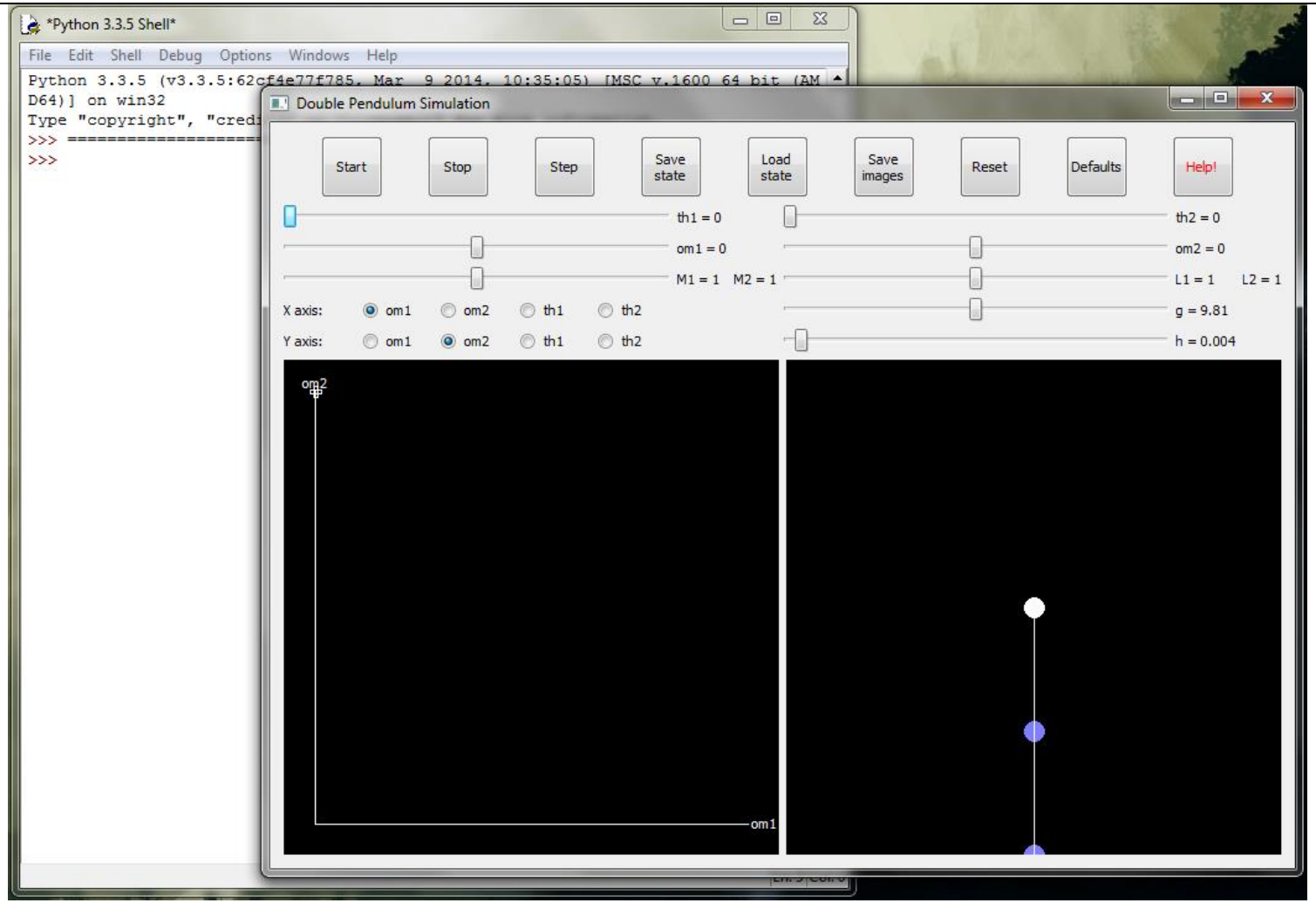
18		 x1: 0.0 y1: -1.0 x2: 1.0 y2: -0.9999999999999999 th1: 0.0 th2: 1.5707963267948968 om1: 0 om2: 0 ke: 0.0 gpe: 19.62 om1 max: 4.4294469180700204 om2 max: 6.26418390534633 th1 max: 1.0471975511965979 th2 max: 1.5707963267948966 gpeMin: 9.81 energy: 19.62	Passed

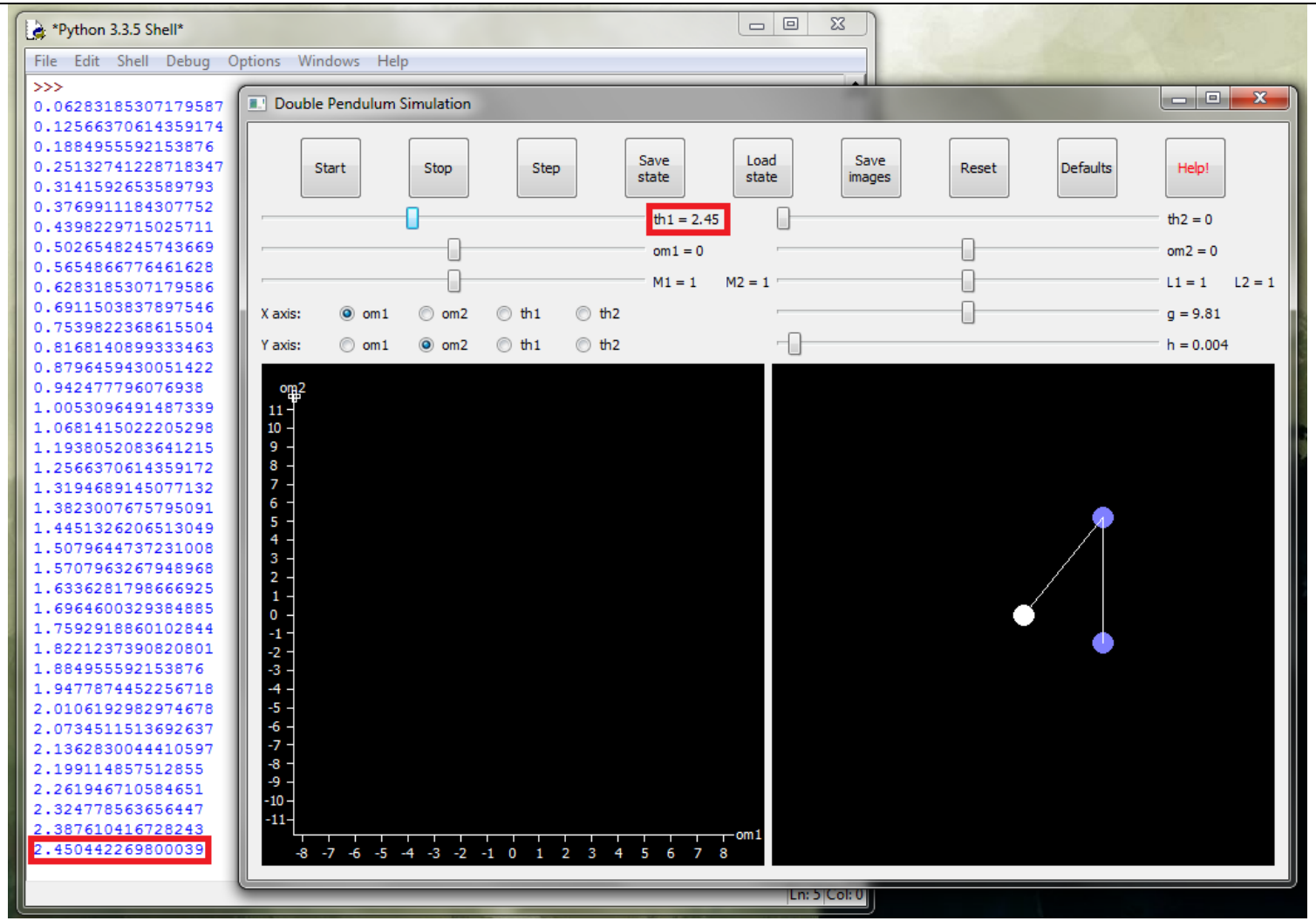
19		 x1: -1.1905376415773734 y1: -0.15039988027716508 x2: -0.5741270473567419 y2: 0.35953931152178675 th1: 4.838052686528282 th2: 2.261946710584651 om1: 2.2199999999999998 om2: 2.16 ke: 4.032857065430307 gpe: 67.4539268922181 om1 max: 10.985303309749446 om2 max: 20.597443705780197 th1 max: 2.320897162728231 th2 max: 3.141592653589793 gpeMin: 8.927884800000001 energy: 71.48678395764841	Passed

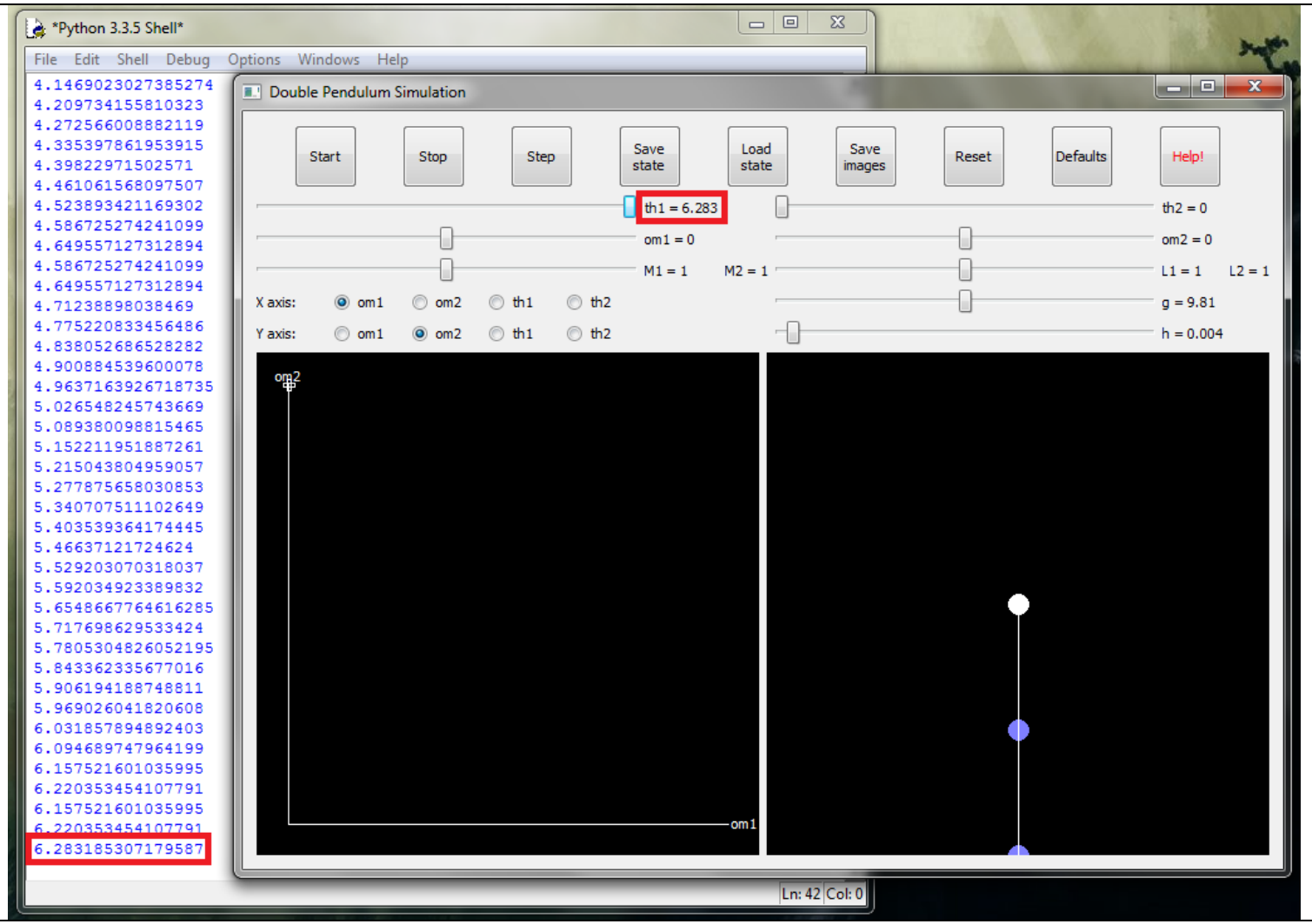
20	 x1: 0.8763066800438637 y1: -0.48175367410171516 x2: 1.8763066800438637 y2: -0.481753674101715 th1: 1.0681415022205298 th2: 1.5707963267948968 om1: 0 om2: 0 L1: 1 L2: 1 M1: 1 M2: 1 g: 9.81		Passed

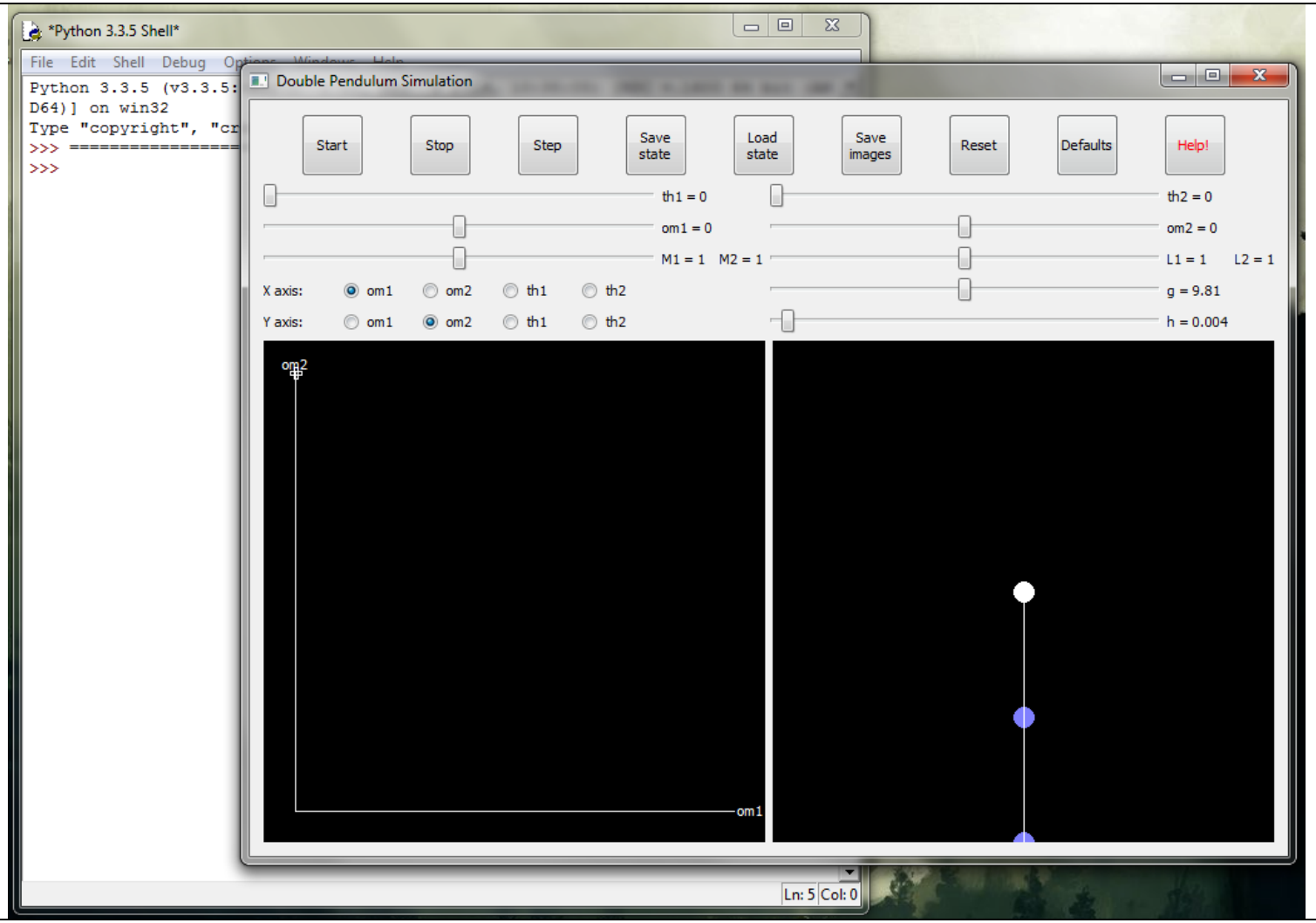
21	 x1: -2.1227218975731604e-16 y1: 0.66 x2: -6.432490598706546e-16 y2: 2.0 th1: 3.1415926535897936 th2: 3.1415926535897936 om1: 1.02 om2: 0.42 L1: 0.66 L2: 1.3399999999999999 M1: 0.6 M2: 1.4 g: 9.81 	Passed. Note that x1 and y1 are negligible

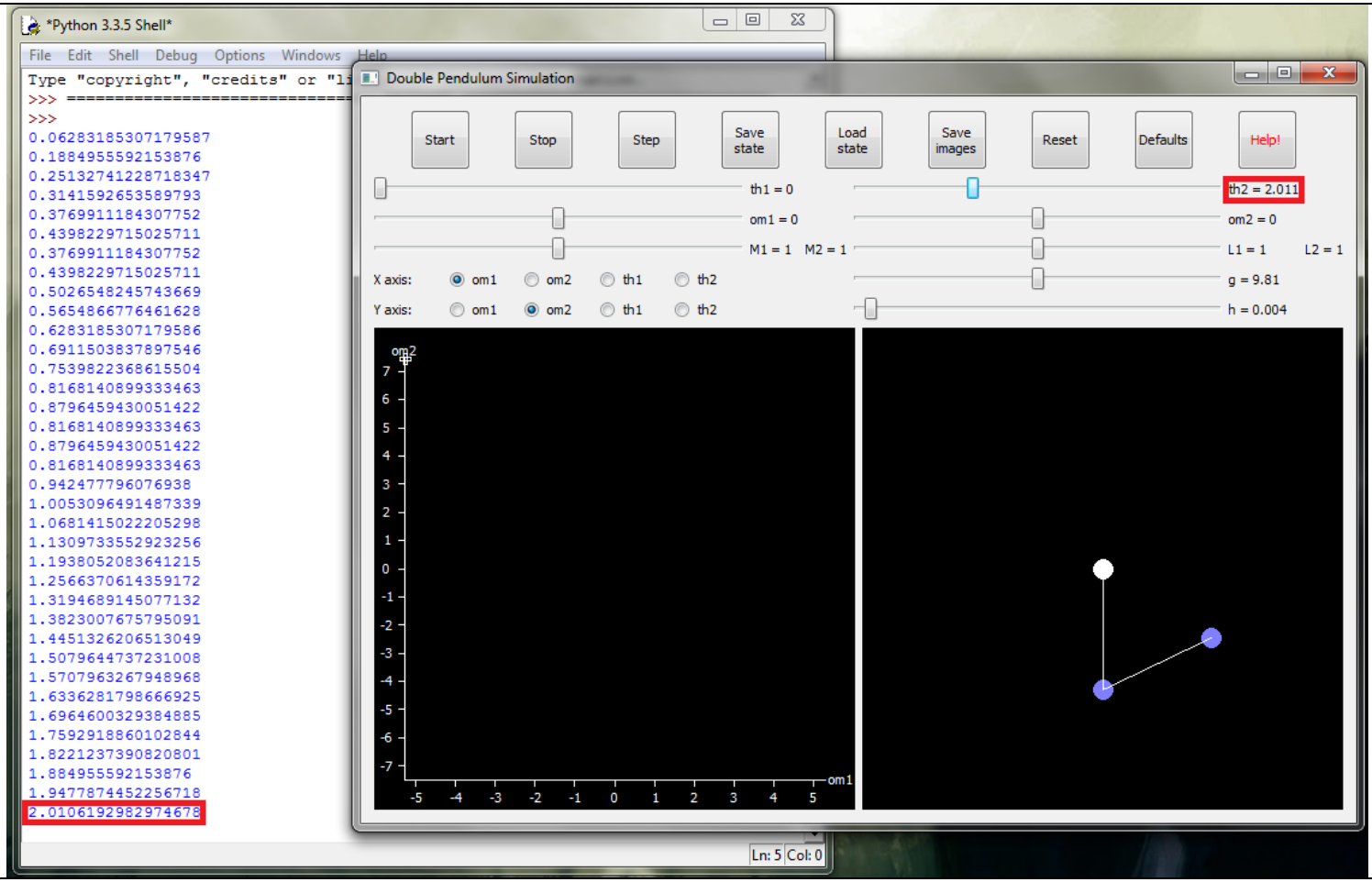
22	 x1: 0.0 y1: -1.0 x2: 0.0 y2: -2.0 th1: 0 th2: 0 om1: 3.0 om2: -3.0 L1: 1 L2: 1 M1: 1 M2: 1 g: 0.0		Passed

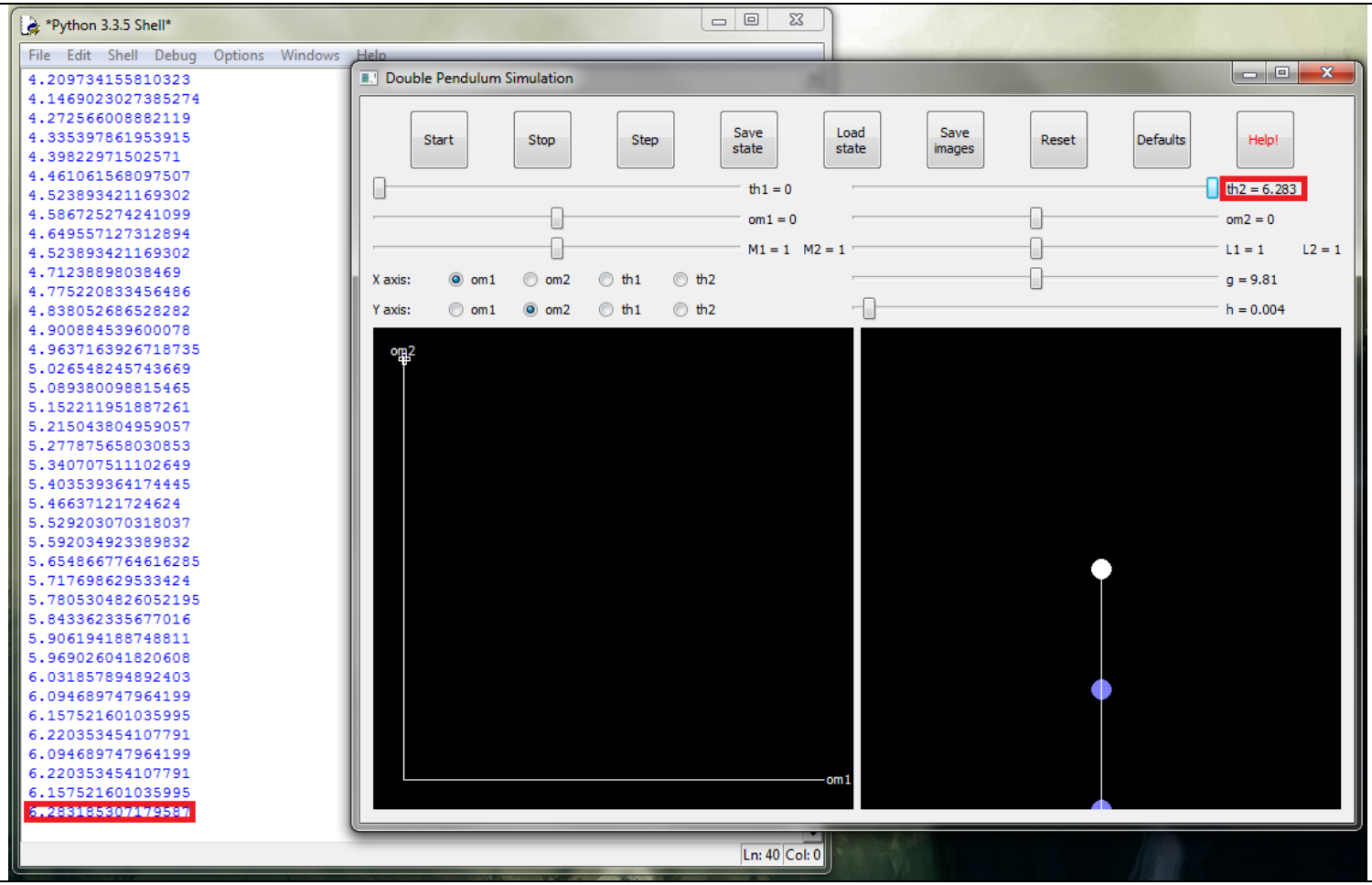
23		Passed. Note that the 6.2831… final value in the third screen shot is 2π, and as this is a full rotation in radians, the pendulum is back to being vertically downwards.

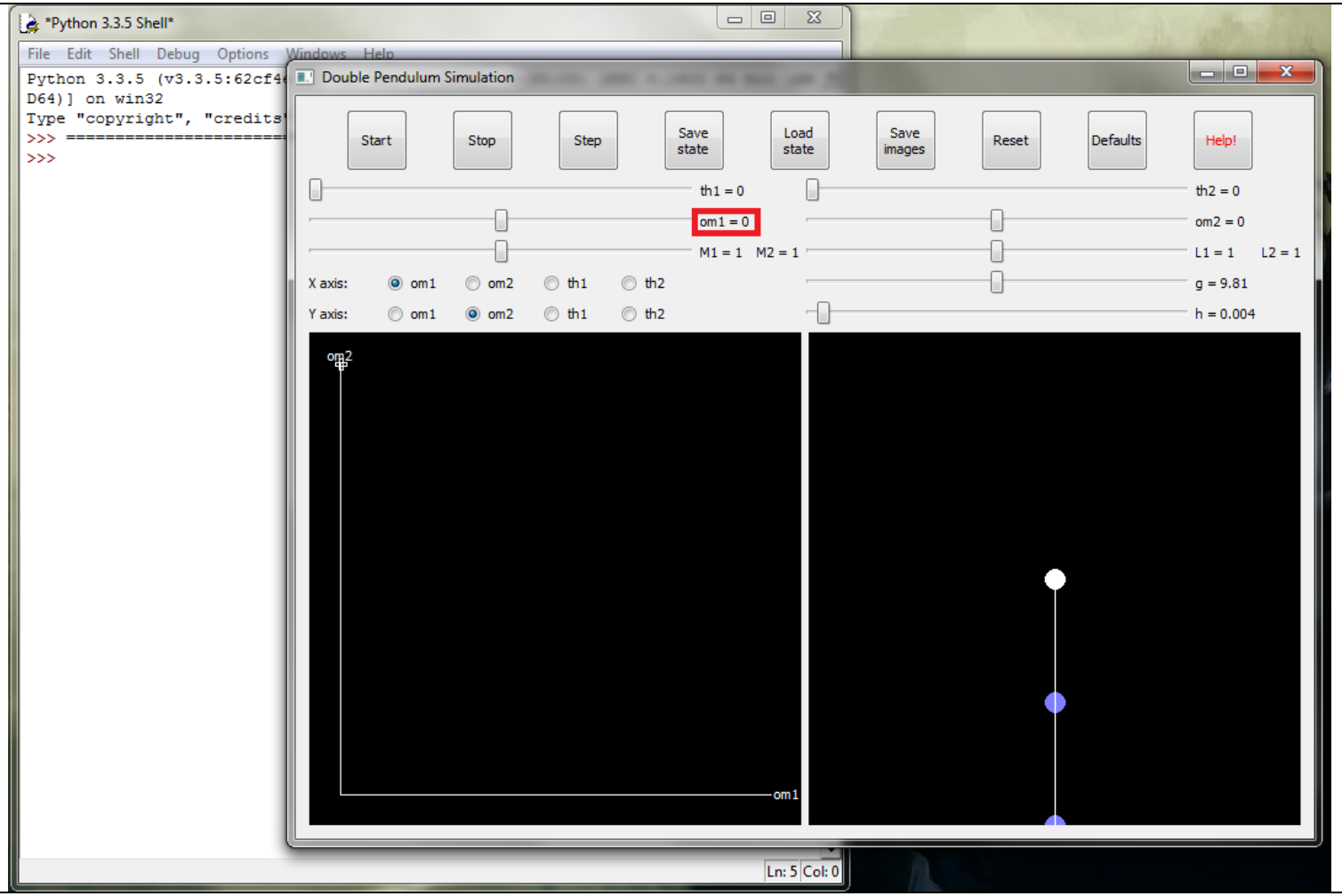

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
>>>
0.06283185307179587
0.12566370614359174
0.1884955592153876
0.25132741228718347
0.3141592653589793
0.3769911184307752
0.4398229715025711
0.5026548245743669
0.5654866776461628
0.6283185307179586
0.6911503837897546
0.7539822368615504
0.8168140899333463
0.8796459430051422
0.942477796076938
1.0053096491487339
1.0681415022205298
1.1938052083641215
1.2566370614359172
1.3194689145077132
1.3823007675795091
1.4451326206513049
1.5079644737231008
1.5707963267948968
1.6336281798666925
1.6964600329384885
1.7592918860102844
1.8221237390820801
1.884955592153876
1.9477874452256718
2.0106192982974678
2.0734511513692637
2.1362830044410597
2.199114857512855
2.261946710584651
2.324778563656447
2.387610416728243
2.450442269800039
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 2.45
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
g = 9.81
h = 0.004
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
om2
om1
Ln: 5 Col: 0

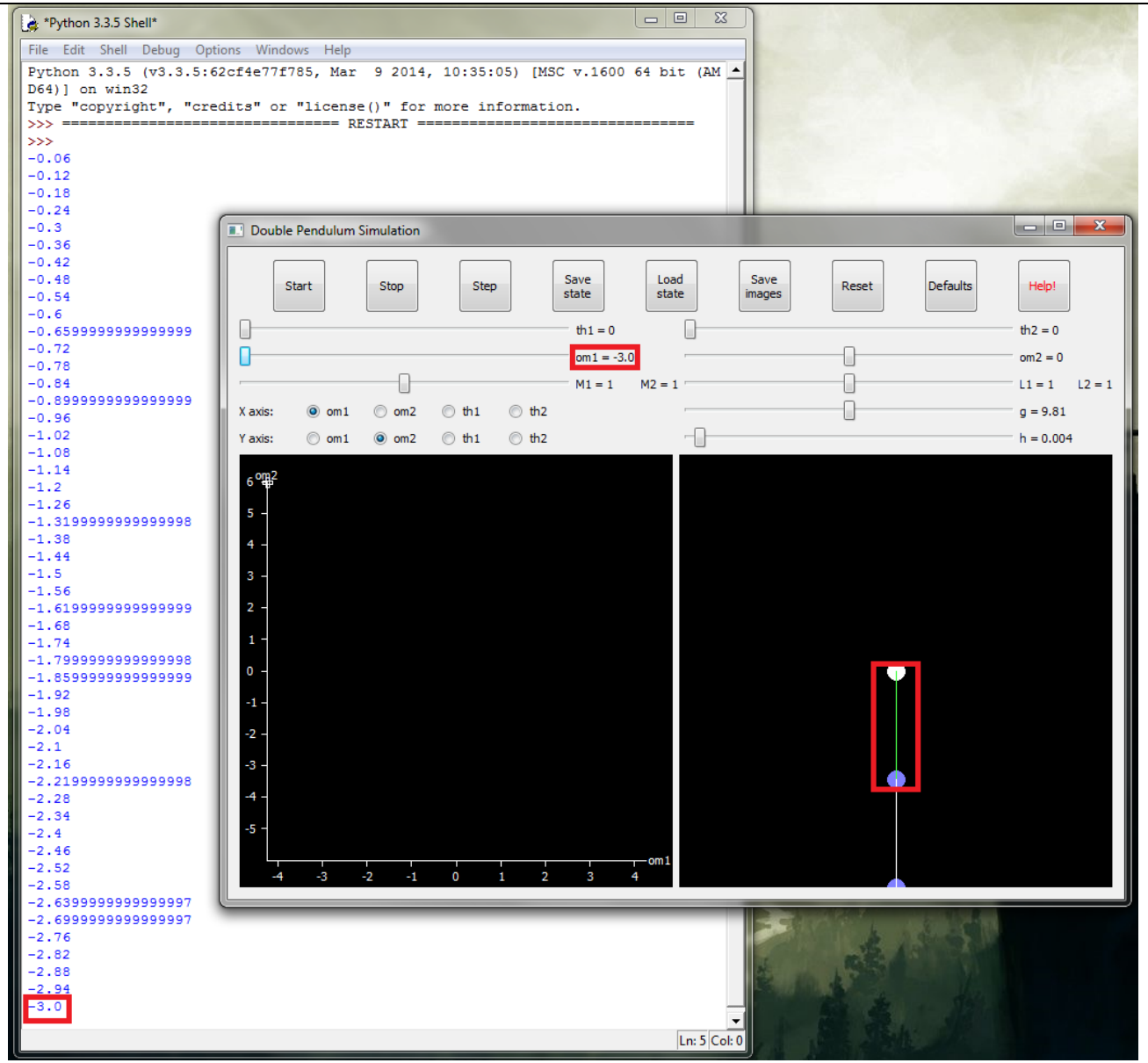

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
4.1469023027385274
4.209734155810323
4.272566008882119
4.335397861953915
4.39822971502571
4.461061568097507
4.523893421169302
4.586725274241099
4.649557127312894
4.586725274241099
4.649557127312894
4.71238898038469
4.775220833456486
4.838052686528282
4.900884539600078
4.9637163926718735
5.026548245743669
5.089380098815465
5.152211951887261
5.215043804959057
5.277875658030853
5.340707511102649
5.403539364174445
5.46637121724624
5.529203070318037
5.592034923389832
5.6548667764616285
5.717698629533424
5.7805304826052195
5.843362335677016
5.906194188748811
5.969026041820608
6.031857894892403
6.094689747964199
6.157521601035995
6.220353454107791
6.157521601035995
6.220353454107791
6.283185307179587
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 6.283
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
g = 9.81
h = 0.004
om2
om1
Ln: 42 Col: 0

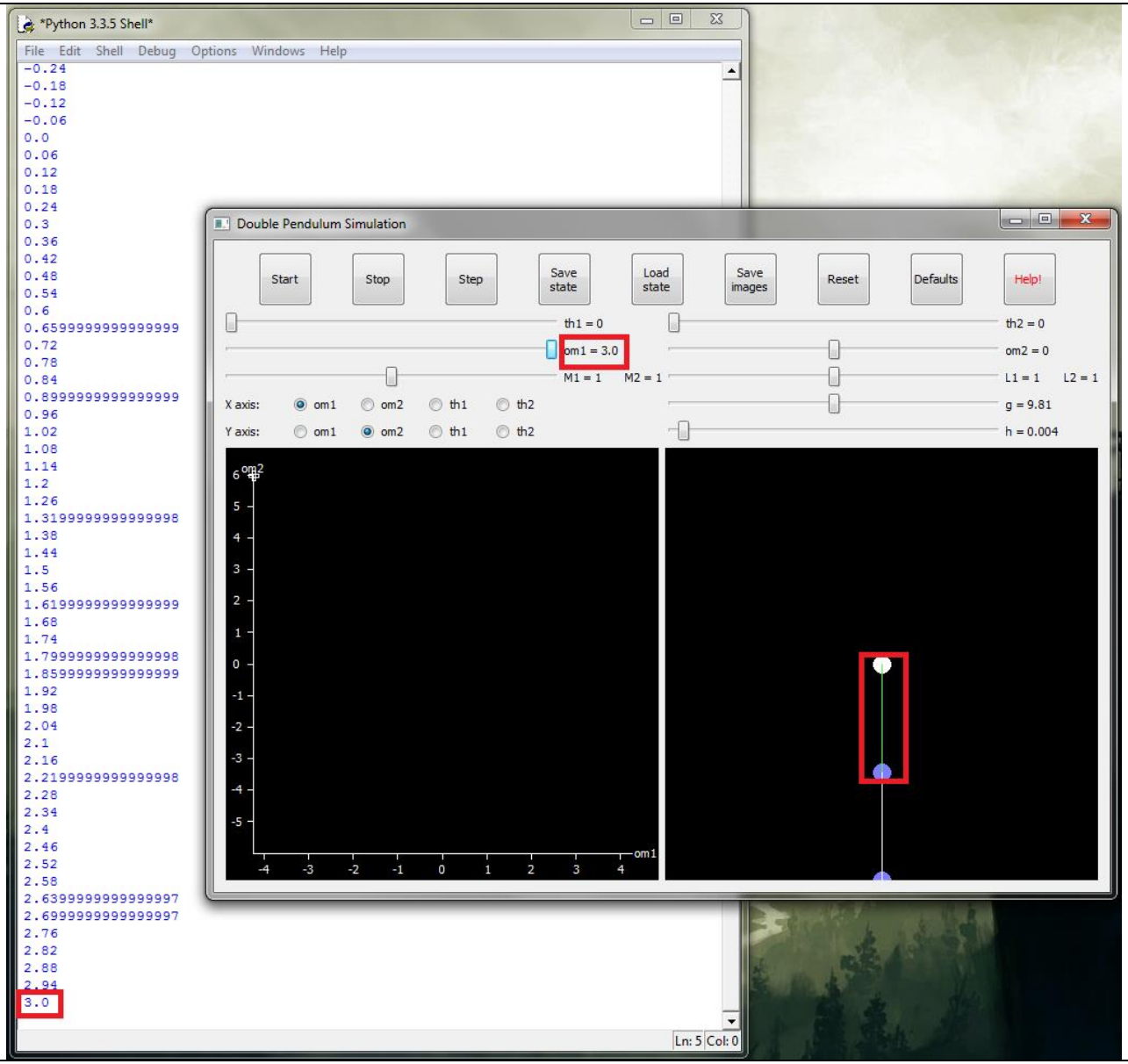
24	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5: D64)] on win32 Type "copyright", "cr >>> ========== >>> Ln: 5 Col: 0 Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 Y axis: om1 om2 th1 th2 g = 9.81 h = 0.004 om2 om1	Passed

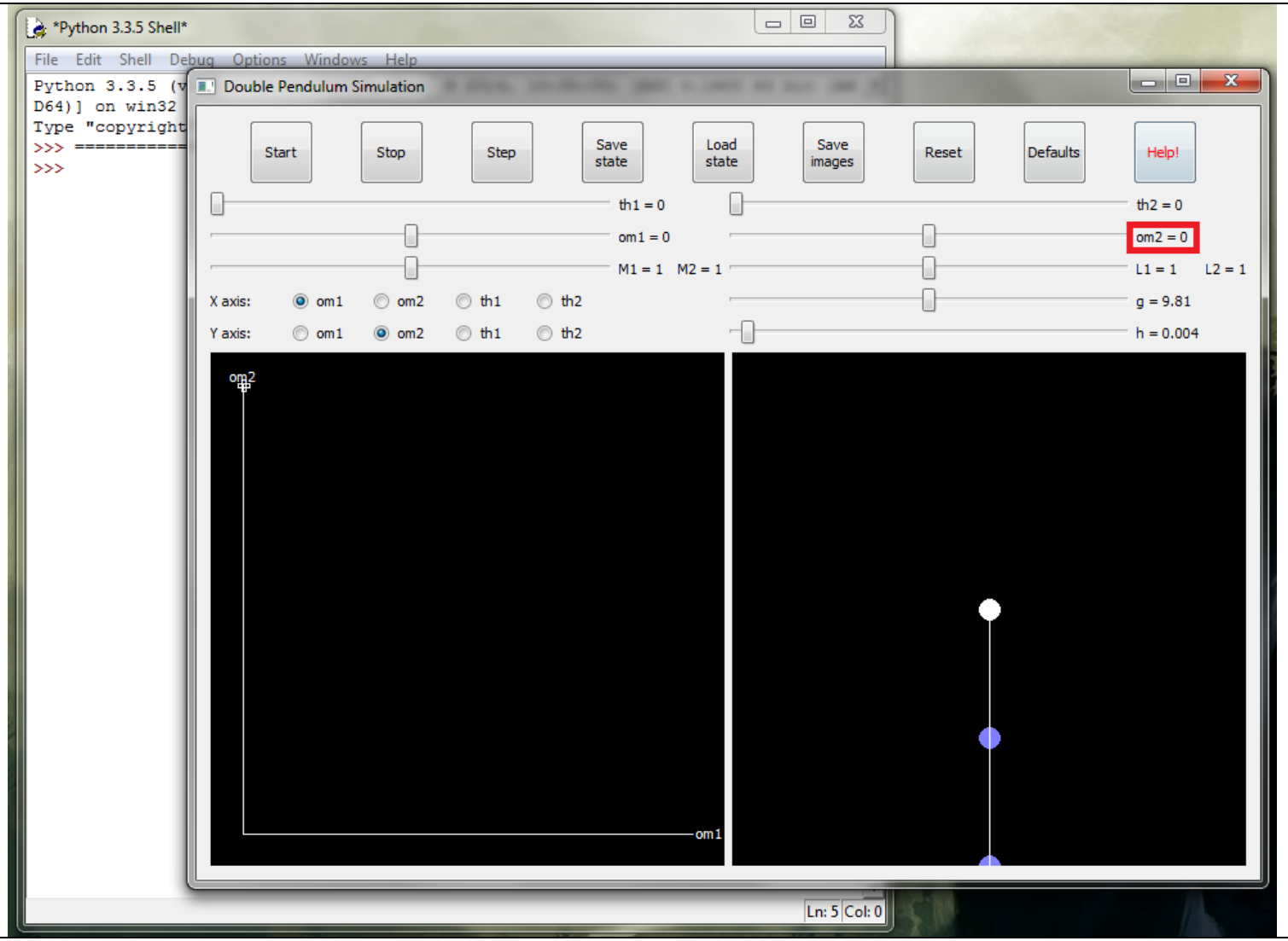

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Type "copyright", "credits" or "li
>>> ==========
>>>
0.06283185307179587
0.1884955592153876
0.25132741228718347
0.3141592653589793
0.3769911184307752
0.4398229715025711
0.3769911184307752
0.4398229715025711
0.5026548245743669
0.5654866776461628
0.6283185307179586
0.6911503837897546
0.7539822368615504
0.8168140899333463
0.8796459430051422
0.8168140899333463
0.8796459430051422
0.8168140899333463
0.942477796076938
1.0053096491487339
1.0681415022205298
1.1309733552923256
1.1938052083641215
1.2566370614359172
1.3194689145077132
1.3823007675795091
1.4451326206513049
1.5079644737231008
1.5707963267948968
1.6336281798666925
1.6964600329384885
1.7592918860102844
1.8221237390820801
1.884955592153876
1.9477874452256718
2.0106192982974678
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 2.011
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 9.81
h = 0.004
om2
om1

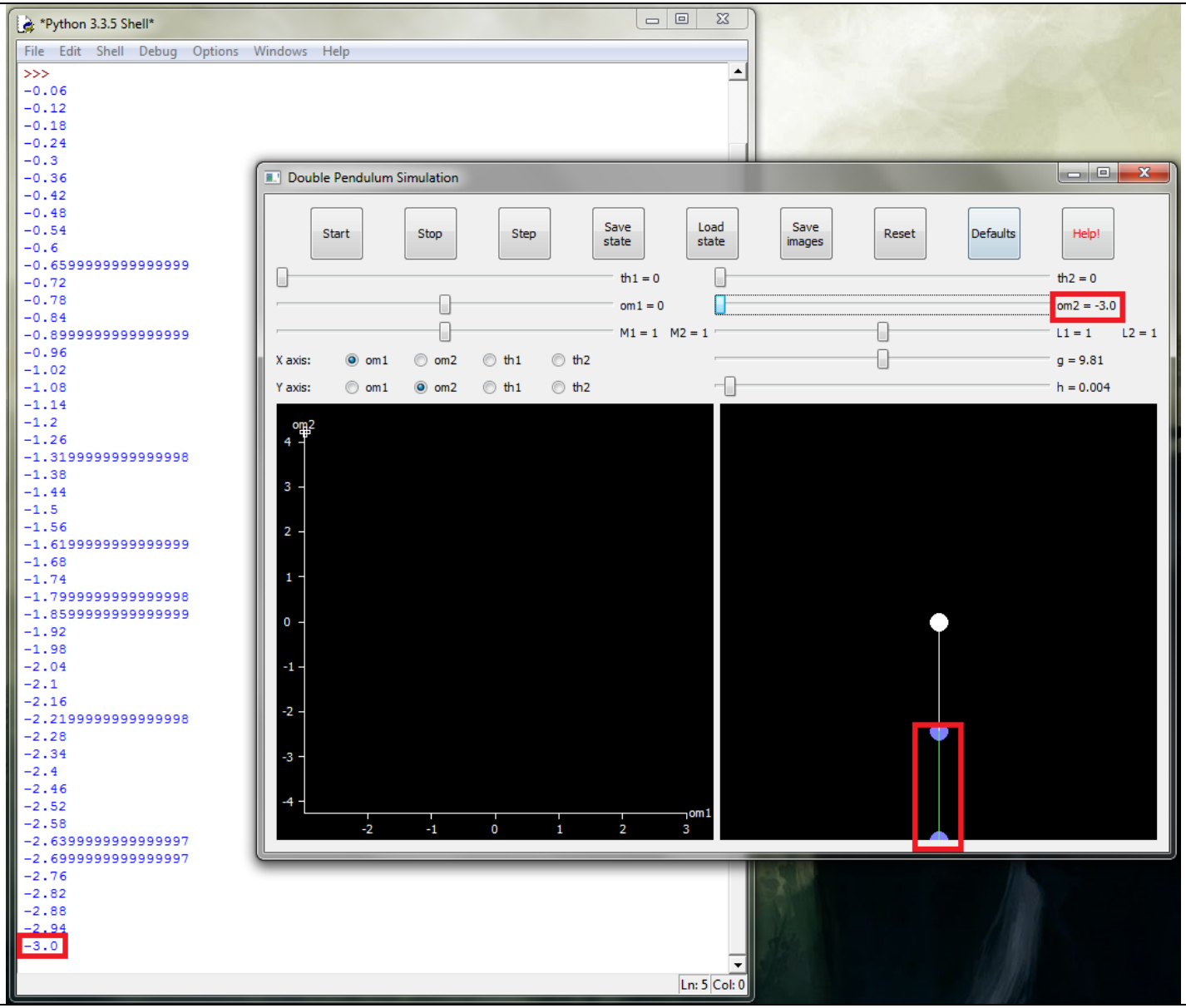

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
4.209734155810323
4.1469023027385274
4.272566008882119
4.335397861953915
4.39822971502571
4.461061568097507
4.523893421169302
4.586725274241099
4.649557127312894
4.523893421169302
4.71238898038469
4.775220833456486
4.838052686528282
4.900884539600078
4.9637163926718735
5.026548245743669
5.089380098815465
5.152211951887261
5.215043804959057
5.277875658030853
5.340707511102649
5.403539364174445
5.46637121724624
5.529203070318037
5.592034923389832
5.6548667764616285
5.717698629533424
5.7805304826052195
5.843362335677016
5.906194188748811
5.969026041820608
6.031857894892403
6.094689747964199
6.157521601035995
6.220353454107791
6.094689747964199
6.220353454107791
6.157521601035995
6.2831853071795871
Ln: 40 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 6.283
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 9.81
h = 0.004
om2
om1

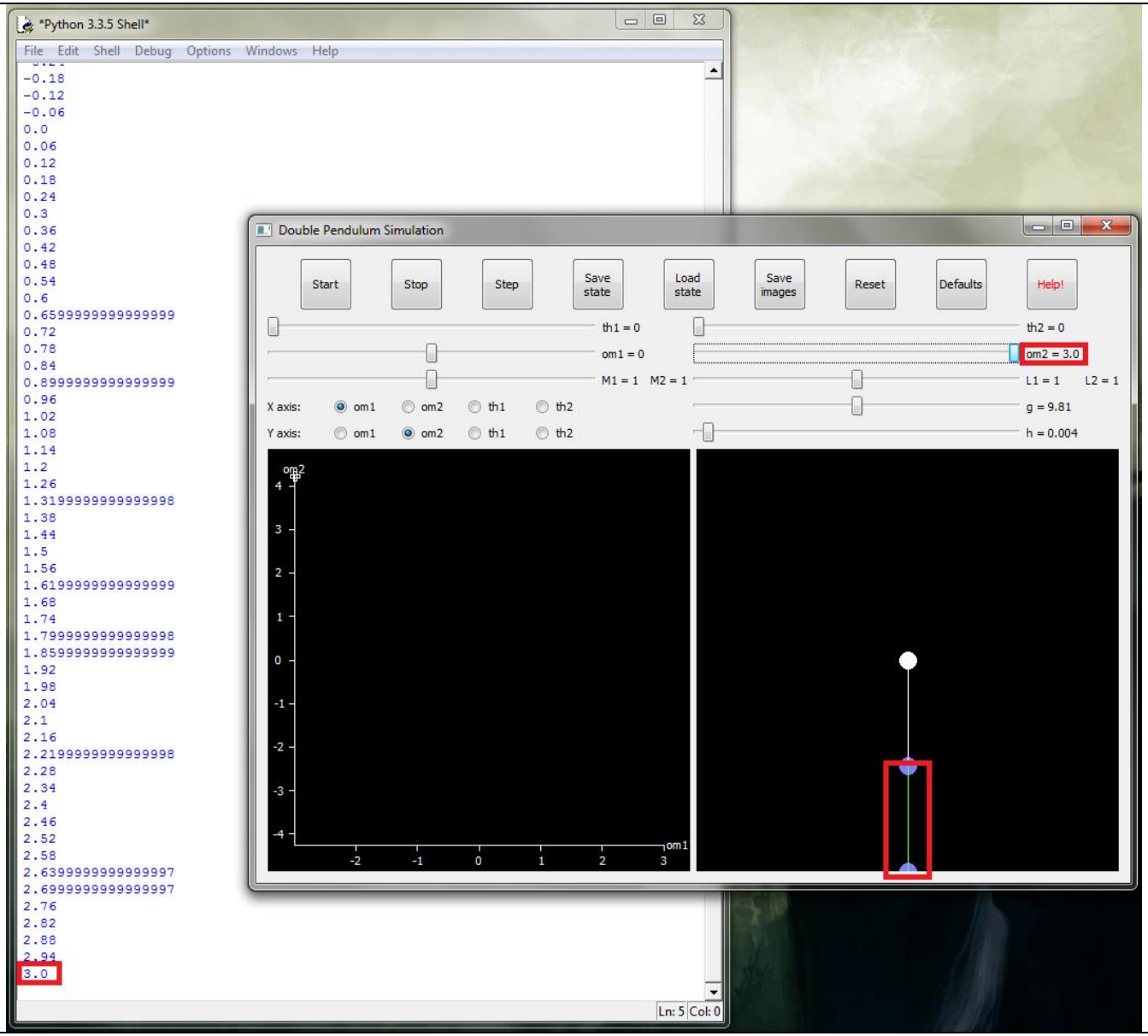
25	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62cf4 D64)] on win32 Type "copyright", "credits" >>> ========== >>> Ln: 5 Col: 0 Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 om1 = 0 M1 = 1 M2 = 1 th2 = 0 om2 = 0 L1 = 1 L2 = 1 g = 9.81 h = 0.004 X axis: om1 om2 th1 th2 Y axis: om1 om2 th1 th2 om2 om1	Passed

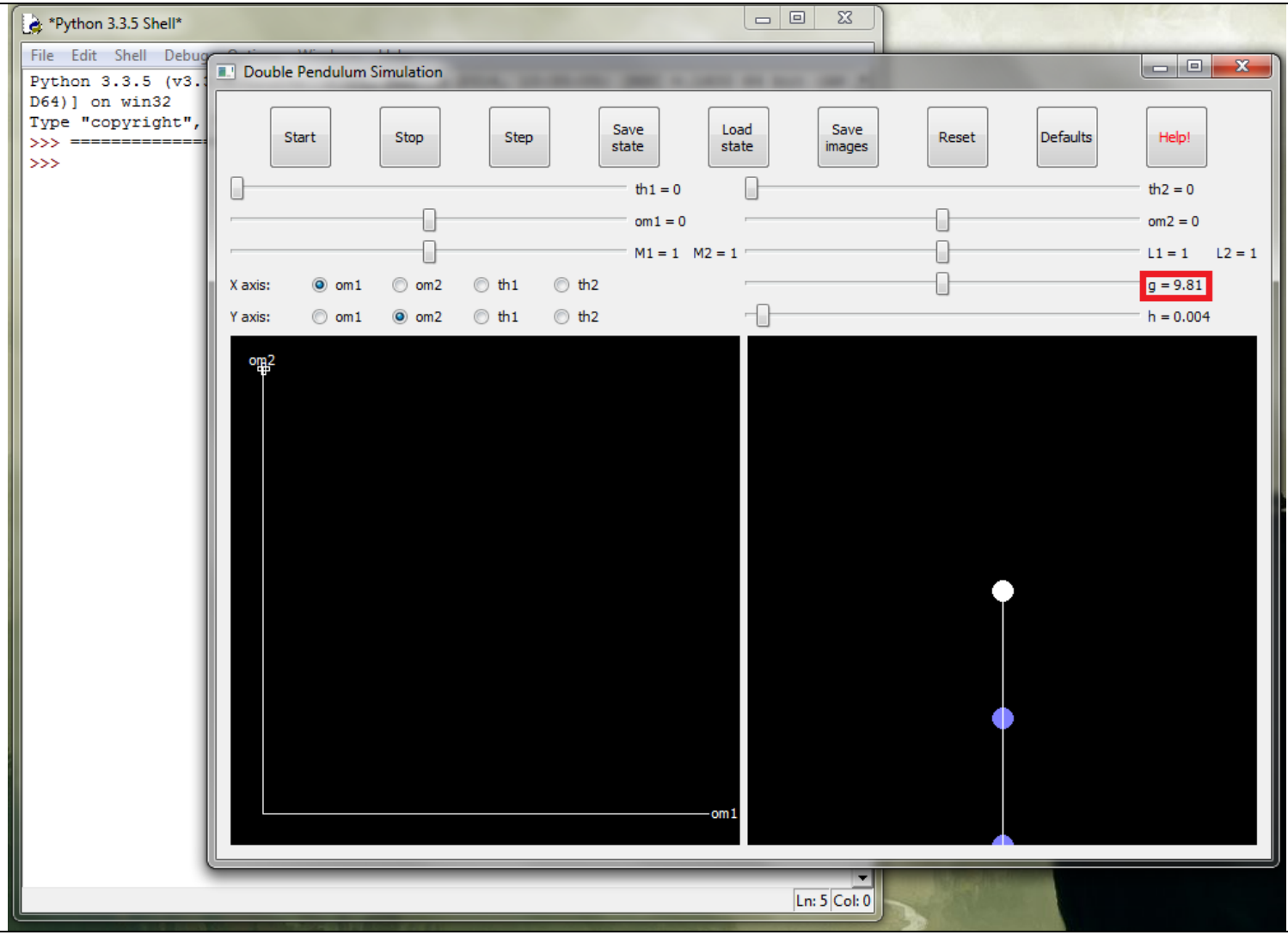

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10:35:05) [MSC v.1600 64 bit (AMD64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
-0.06
-0.12
-0.18
-0.24
-0.3
-0.36
-0.42
-0.48
-0.54
-0.6
-0.6599999999999999
-0.72
-0.78
-0.84
-0.8999999999999999
-0.96
-1.02
-1.08
-1.14
-1.2
-1.26
-1.3199999999999998
-1.38
-1.44
-1.5
-1.56
-1.6199999999999999
-1.68
-1.74
-1.7999999999999998
-1.8599999999999999
-1.92
-1.98
-2.04
-2.1
-2.16
-2.2199999999999998
-2.28
-2.34
-2.4
-2.46
-2.52
-2.58
-2.6399999999999997
-2.6999999999999997
-2.76
-2.82
-2.88
-2.94
-3.0
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
om1 = -3.0
M1 = 1
M2 = 1
th2 = 0
om2 = 0
L1 = 1
L2 = 1
g = 9.81
h = 0.004
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
om2
om1

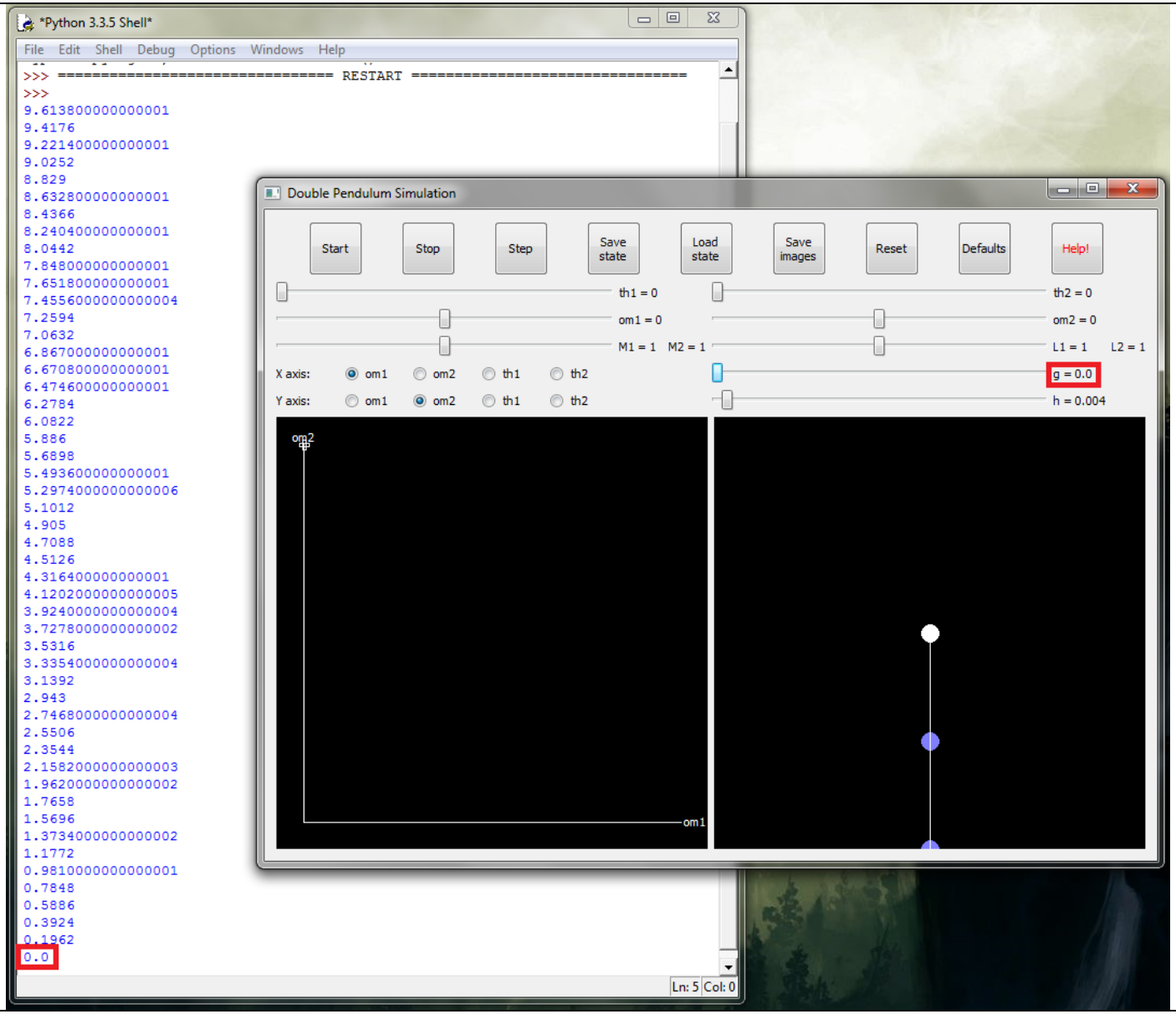

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
-0.24
-0.18
-0.12
-0.06
0.0
0.06
0.12
0.18
0.24
0.3
0.36
0.42
0.48
0.54
0.6
0.6599999999999999
0.72
0.78
0.84
0.8999999999999999
0.96
1.02
1.08
1.14
1.2
1.26
1.3199999999999998
1.38
1.44
1.5
1.56
1.6199999999999999
1.68
1.74
1.7999999999999998
1.8599999999999999
1.92
1.98
2.04
2.1
2.16
2.2199999999999998
2.28
2.34
2.4
2.46
2.52
2.58
2.6399999999999997
2.6999999999999997
2.76
2.82
2.88
2.94
3.0
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
om1 = 3.0
M1 = 1
M2 = 1
th2 = 0
om2 = 0
L1 = 1
L2 = 1
g = 9.81
h = 0.004
X axis:
Y axis:
om1
om2
th1
th2
om2
om1

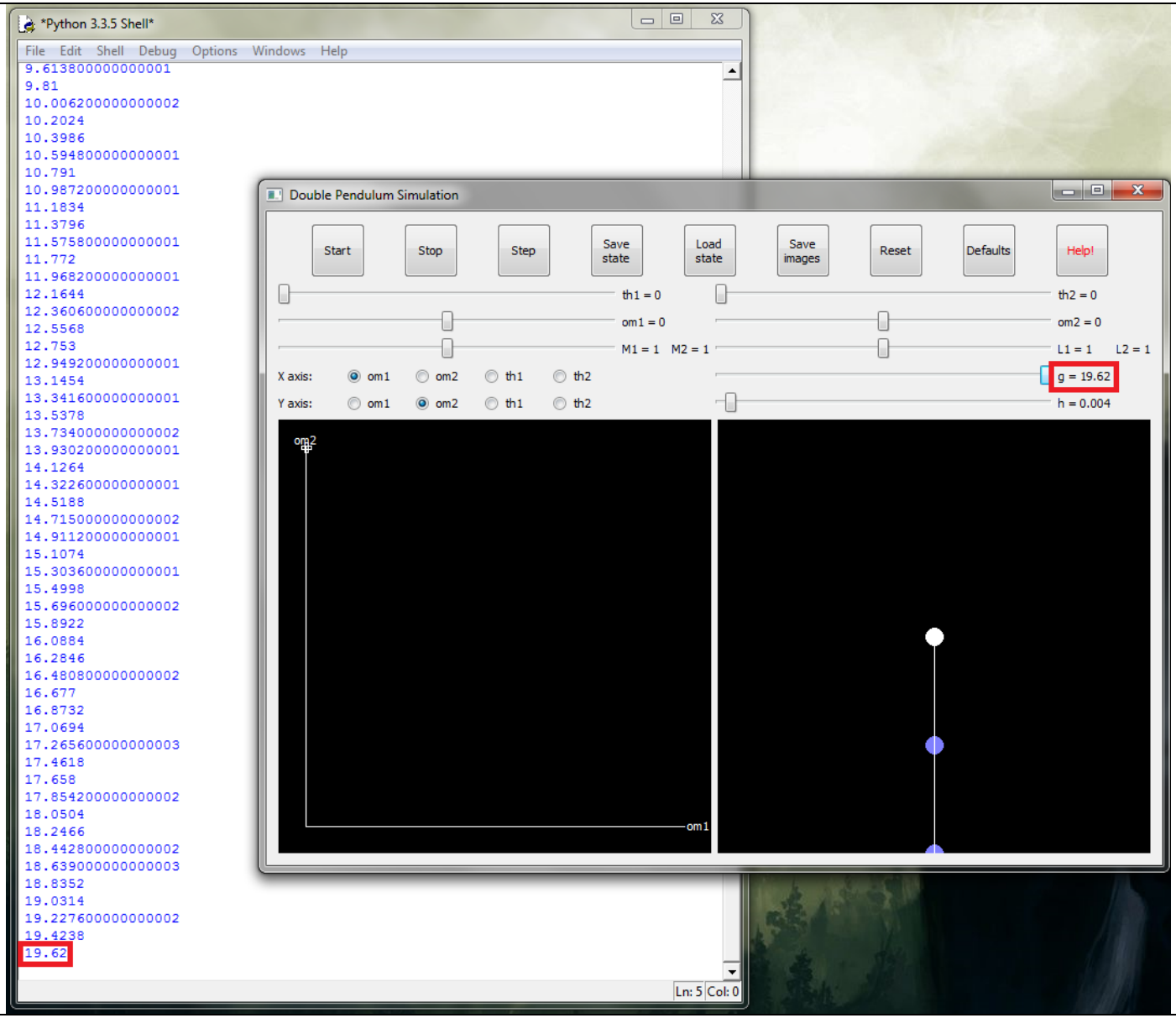
26	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v D64)] on win32 Type "copyright >>> ========== >>> Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 g = 9.81 Y axis: om1 om2 th1 th2 h = 0.004 om2 om1 Ln: 5 Col: 0	Passed

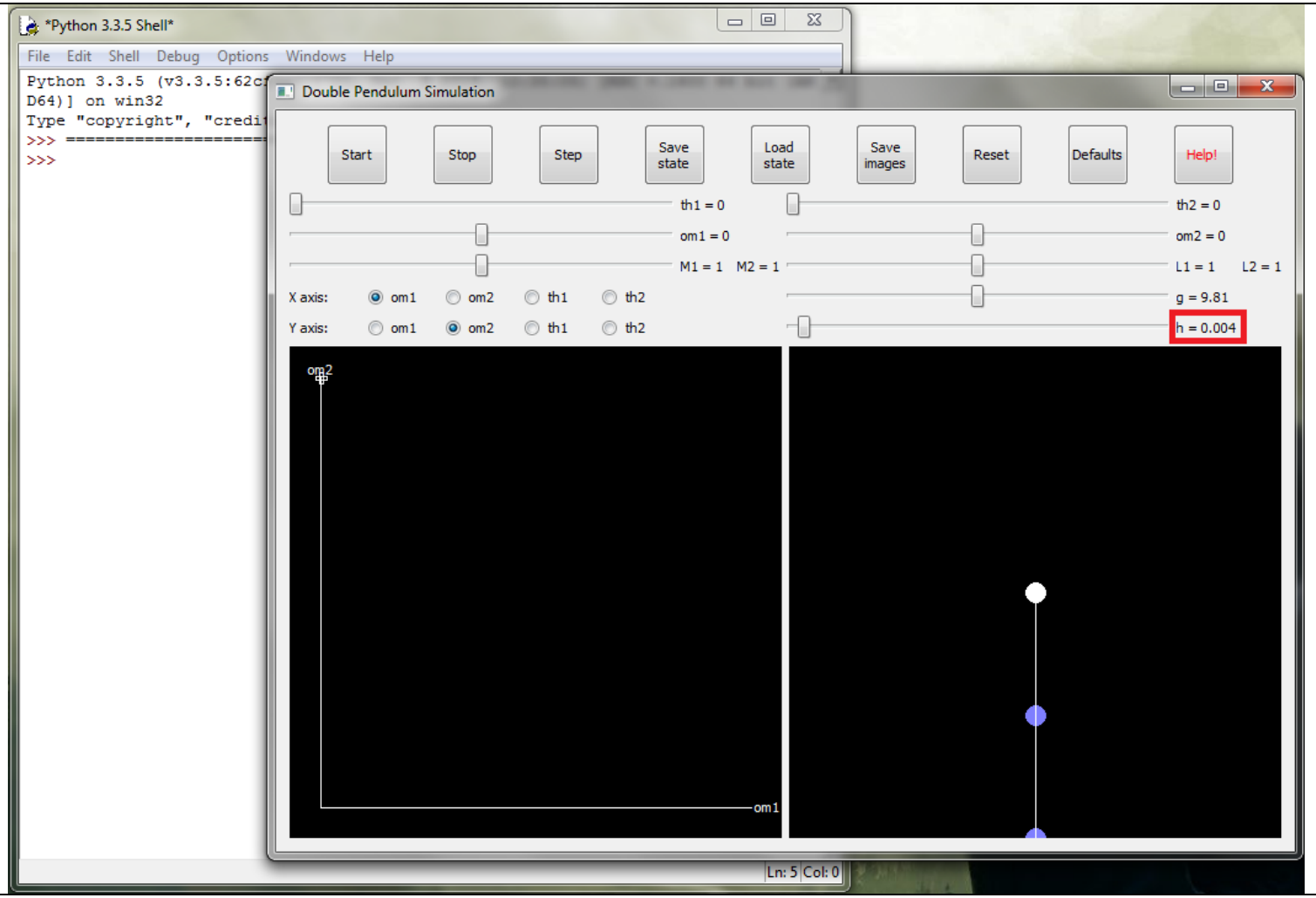

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
>>>
-0.06
-0.12
-0.18
-0.24
-0.3
-0.36
-0.42
-0.48
-0.54
-0.6
-0.6599999999999999
-0.72
-0.78
-0.84
-0.8999999999999999
-0.96
-1.02
-1.08
-1.14
-1.2
-1.26
-1.3199999999999998
-1.38
-1.44
-1.5
-1.56
-1.6199999999999999
-1.68
-1.74
-1.7999999999999998
-1.8599999999999999
-1.92
-1.98
-2.04
-2.1
-2.16
-2.2199999999999998
-2.28
-2.34
-2.4
-2.46
-2.52
-2.58
-2.6399999999999997
-2.6999999999999997
-2.76
-2.82
-2.88
-2.94
-3.0
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = -3.0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 9.81
h = 0.004
om2
om1

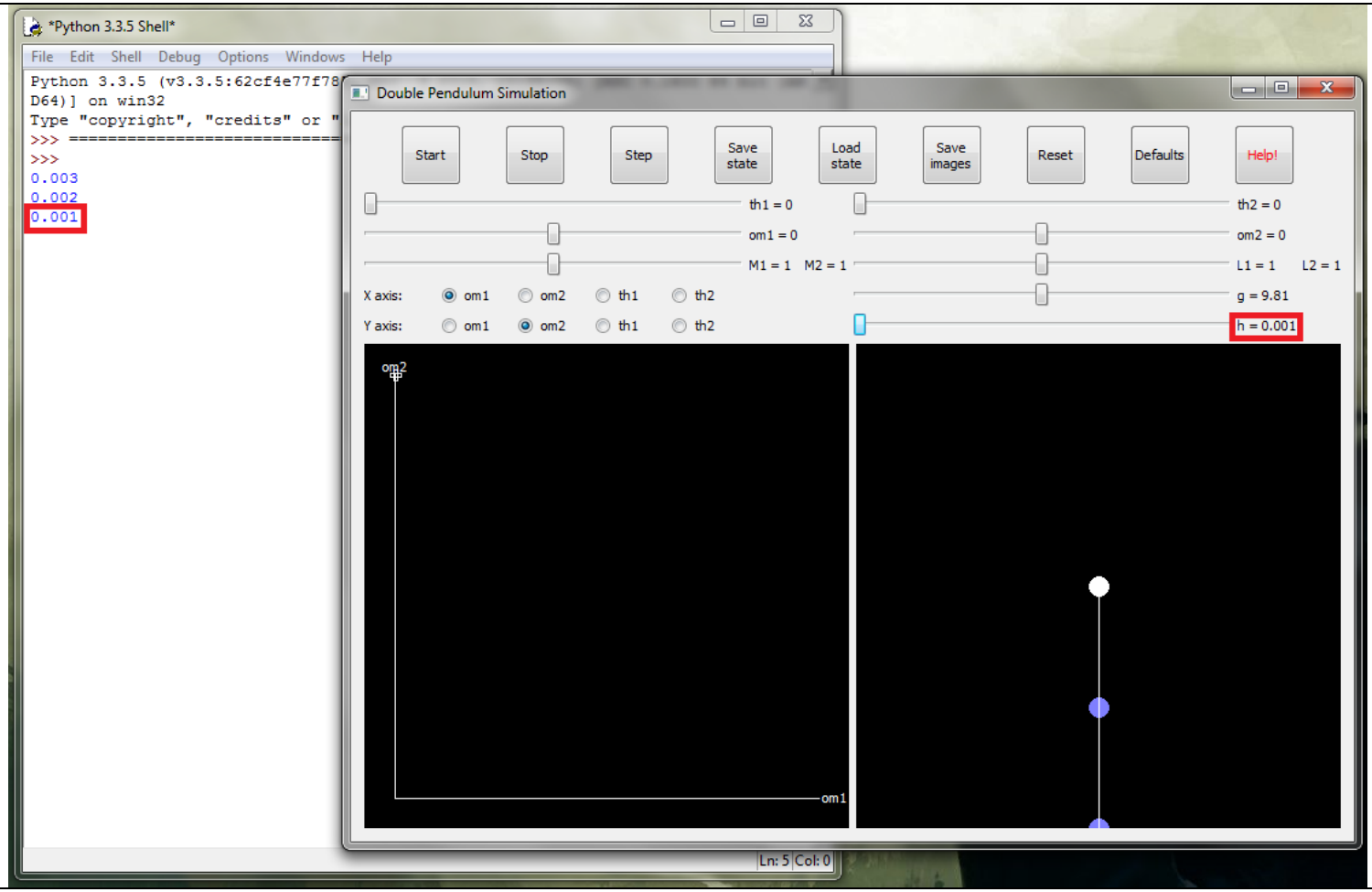

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
-0.18
-0.12
-0.06
0.0
0.06
0.12
0.18
0.24
0.3
0.36
0.42
0.48
0.54
0.6
0.6599999999999999
0.72
0.78
0.84
0.8999999999999999
0.96
1.02
1.08
1.14
1.2
1.26
1.3199999999999998
1.38
1.44
1.5
1.56
1.6199999999999999
1.68
1.74
1.7999999999999998
1.8599999999999999
1.92
1.98
2.04
2.1
2.16
2.2199999999999998
2.28
2.34
2.4
2.46
2.52
2.58
2.6399999999999997
2.6999999999999997
2.76
2.82
2.88
2.94
3.0
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 3.0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 9.81
h = 0.004
om2
om1

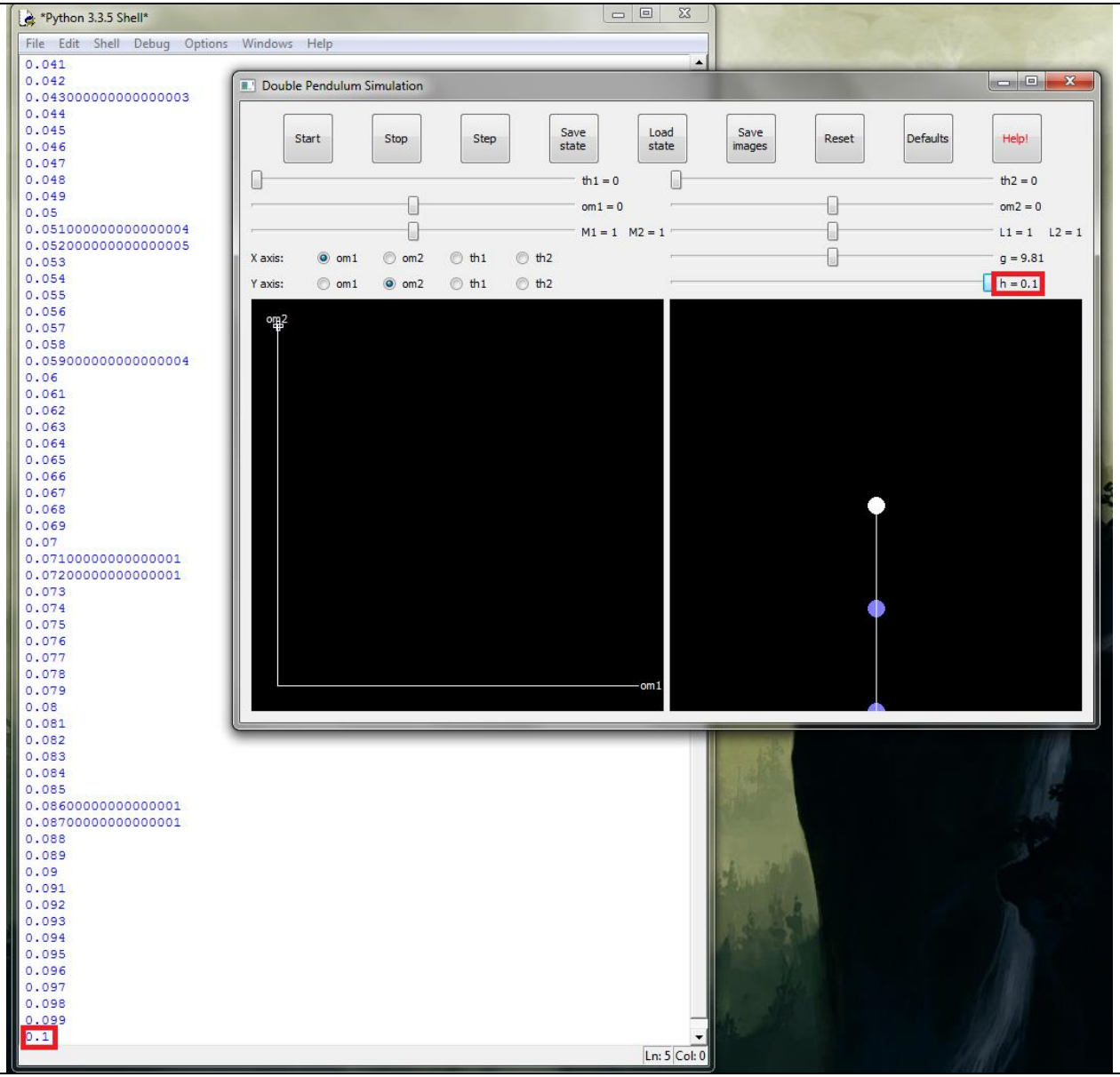
27	 *Python 3.3.5 Shell* File Edit Shell Debug Python 3.3.5 (v3. D64)] on win32 Type "copyright", >>> >>> Ln: 5 Col: 0 Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 Y axis: om1 om2 th1 th2 g = 9.81 h = 0.004 om2 om1	Passed

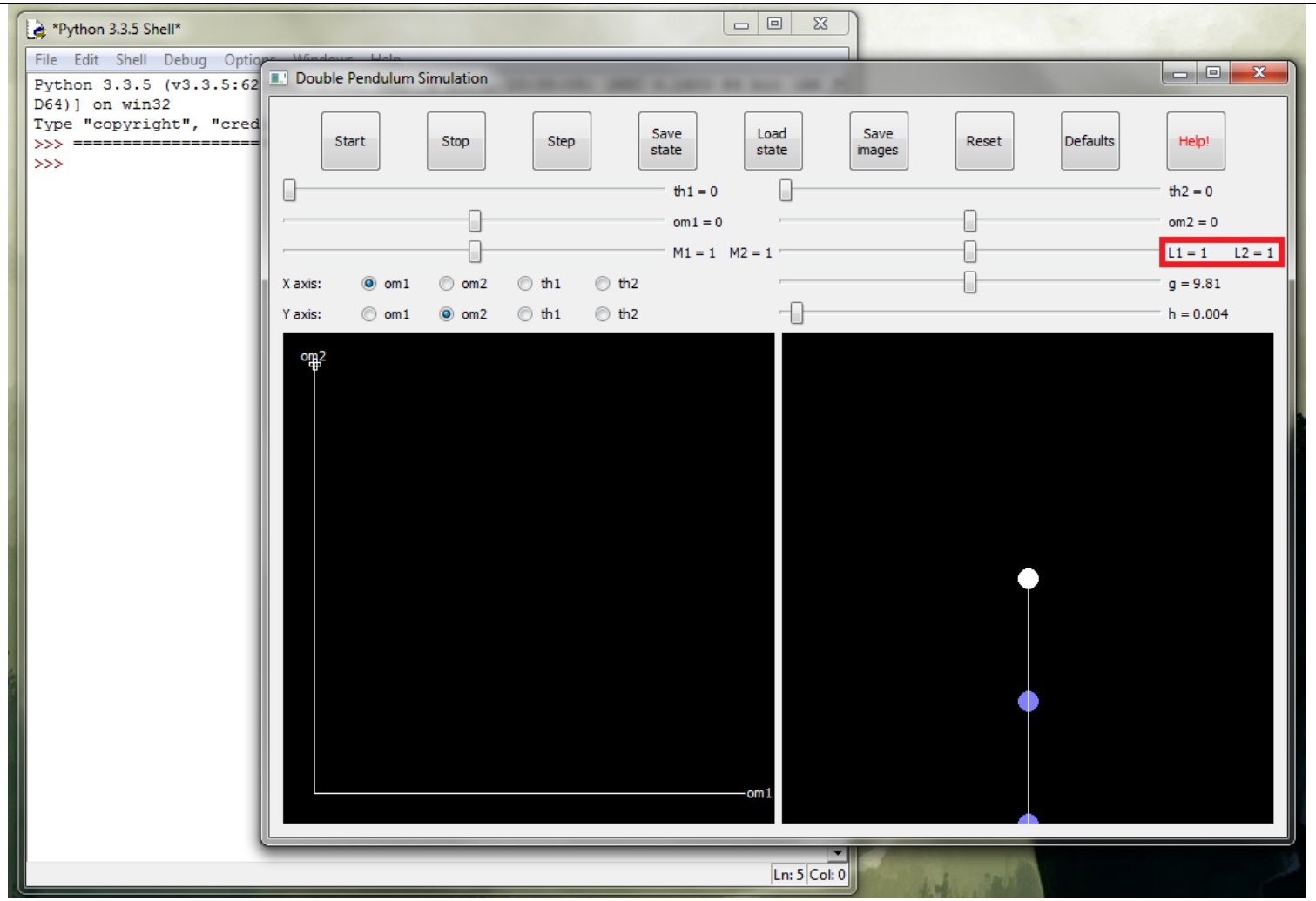

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
>>> ================================ RESTART ================================
>>>
9.613800000000001
9.4176
9.221400000000001
9.0252
8.829
8.632800000000001
8.4366
8.240400000000001
8.0442
7.848000000000001
7.651800000000001
7.4556000000000004
7.2594
7.0632
6.867000000000001
6.670800000000001
6.474600000000001
6.2784
6.0822
5.886
5.6898
5.493600000000001
5.2974000000000006
5.1012
4.905
4.7088
4.5126
4.316400000000001
4.1202000000000005
3.9240000000000004
3.7278000000000002
3.5316
3.3354000000000004
3.1392
2.943
2.7468000000000004
2.5506
2.3544
2.1582000000000003
1.9620000000000002
1.7658
1.5696
1.3734000000000002
1.1772
0.9810000000000001
0.7848
0.5886
0.3924
0.1962
0.0
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
g = 0.0
h = 0.004
om2
om1

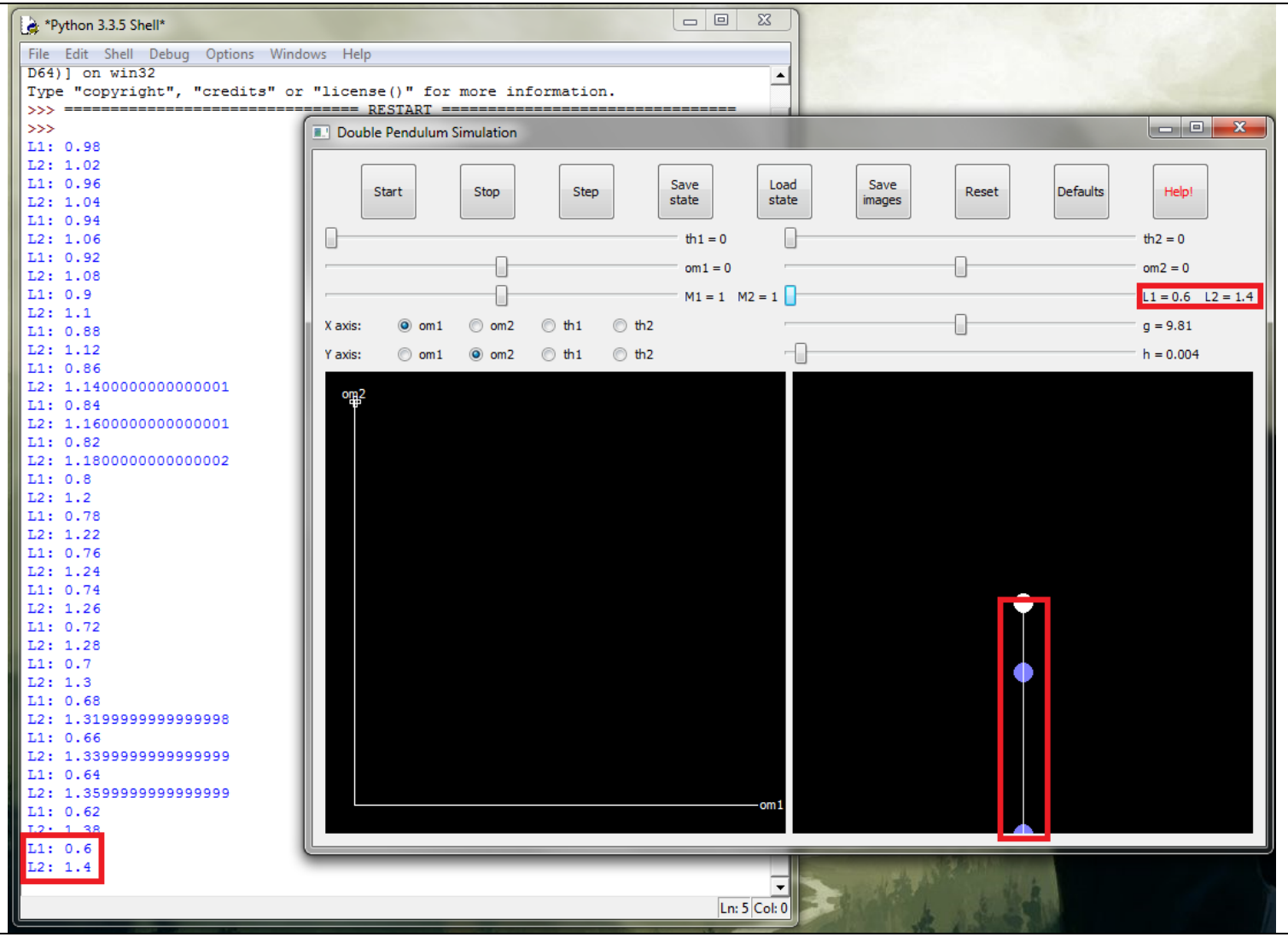

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
9.613800000000001
9.81
10.006200000000002
10.2024
10.3986
10.594800000000001
10.791
10.987200000000001
11.1834
11.3796
11.575800000000001
11.772
11.968200000000001
12.1644
12.360600000000002
12.5568
12.753
12.949200000000001
13.1454
13.341600000000001
13.5378
13.734000000000002
13.930200000000001
14.1264
14.322600000000001
14.5188
14.715000000000002
14.911200000000001
15.1074
15.303600000000001
15.4998
15.696000000000002
15.8922
16.0884
16.2846
16.480800000000002
16.677
16.8732
17.0694
17.265600000000003
17.4618
17.658
17.854200000000002
18.0504
18.2466
18.442800000000002
18.639000000000003
18.8352
19.0314
19.227600000000002
19.4238
19.62
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 19.62
h = 0.004
om2
om1

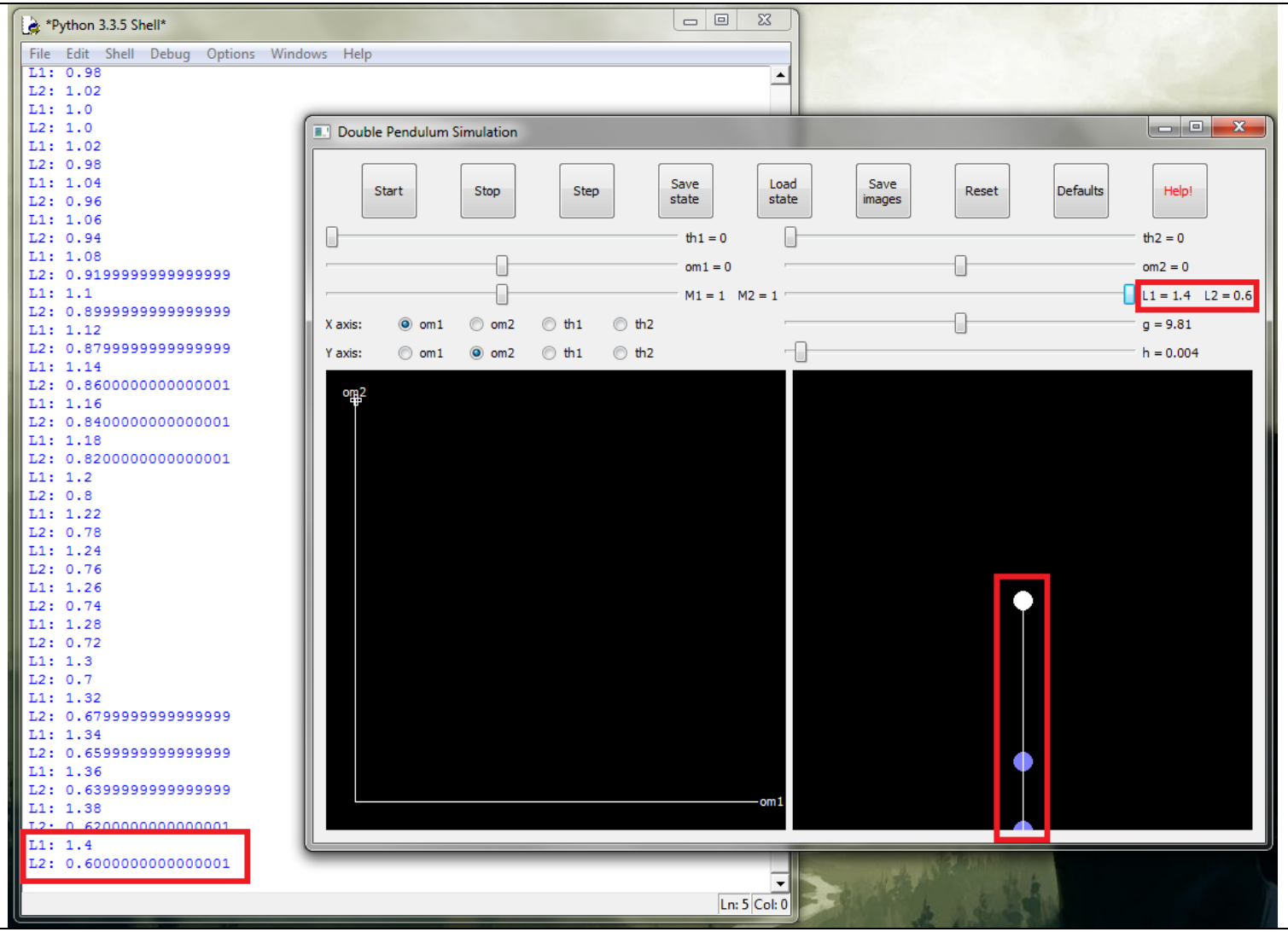
28	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62c D64)] on win32 Type "copyright", "credi >>> ========== >>> Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 g = 9.81 Y axis: om1 om2 th1 th2 h = 0.004 om2 om1 Ln: 5 Col: 0	Passed


Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
Python 3.3.5 (v3.3.5:62cf4e77f78
D64)] on win32
Type "copyright", "credits" or "
>>> ==========
>>>
0.003
0.002
0.001
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 1
L2 = 1
X axis:
om1
om2
th1
th2
Y axis:
om1
om2
th1
th2
g = 9.81
h = 0.001
om2
om1
Ln: 5 Col: 0

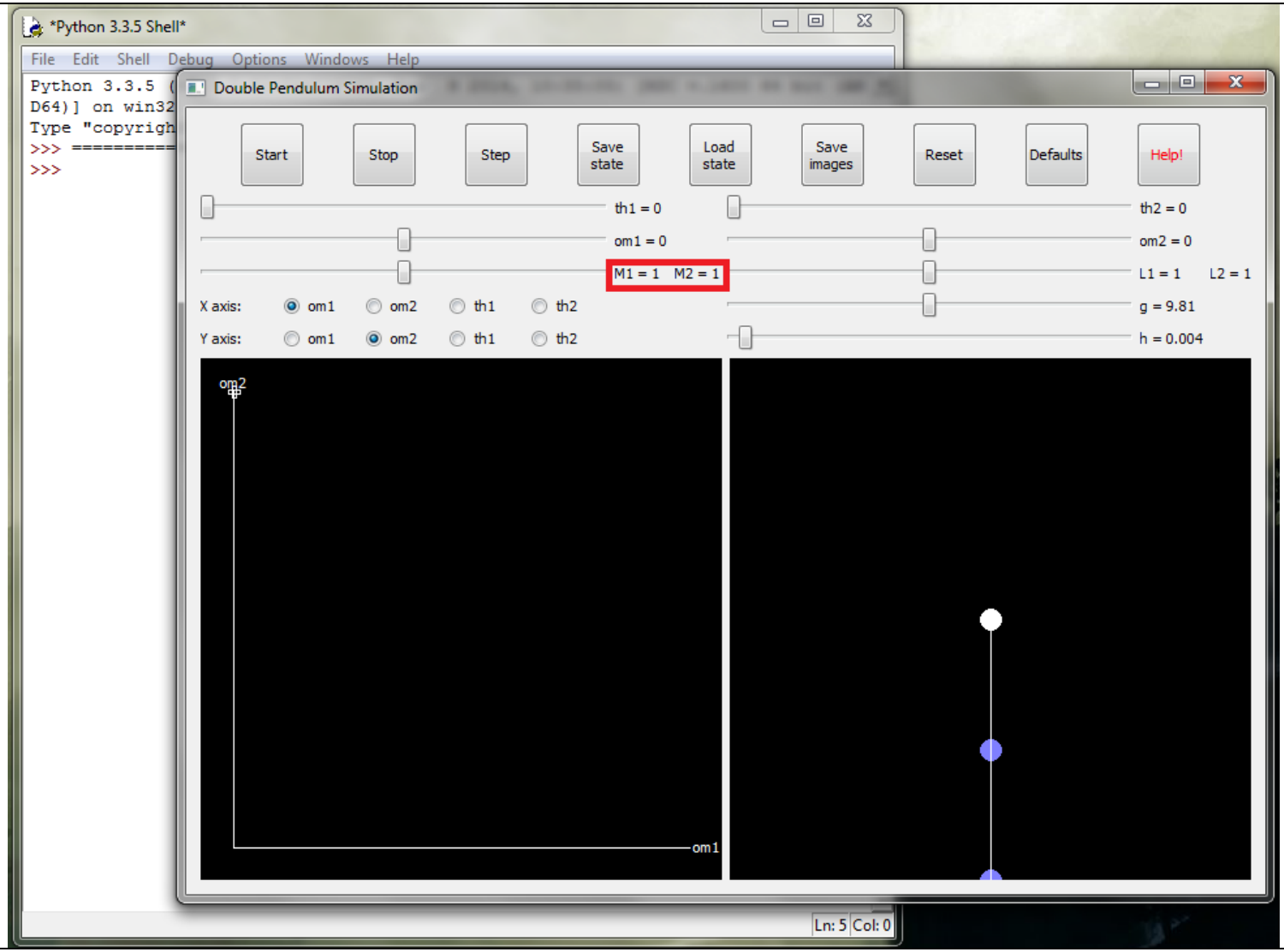

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
0.041
0.042
0.043000000000000003
0.044
0.045
0.046
0.047
0.048
0.049
0.05
0.051000000000000004
0.052000000000000005
0.053
0.054
0.055
0.056
0.057
0.058
0.059000000000000004
0.06
0.061
0.062
0.063
0.064
0.065
0.066
0.067
0.068
0.069
0.07
0.07100000000000001
0.07200000000000001
0.073
0.074
0.075
0.076
0.077
0.078
0.079
0.08
0.081
0.082
0.083
0.084
0.085
0.08600000000000001
0.08700000000000001
0.088
0.089
0.09
0.091
0.092
0.093
0.094
0.095
0.096
0.097
0.098
0.099
0.1
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 1 M2 = 1
L1 = 1 L2 = 1
X axis: om1 om2 th1 th2
g = 9.81
Y axis: om1 om2 th1 th2
h = 0.1
om2
om1

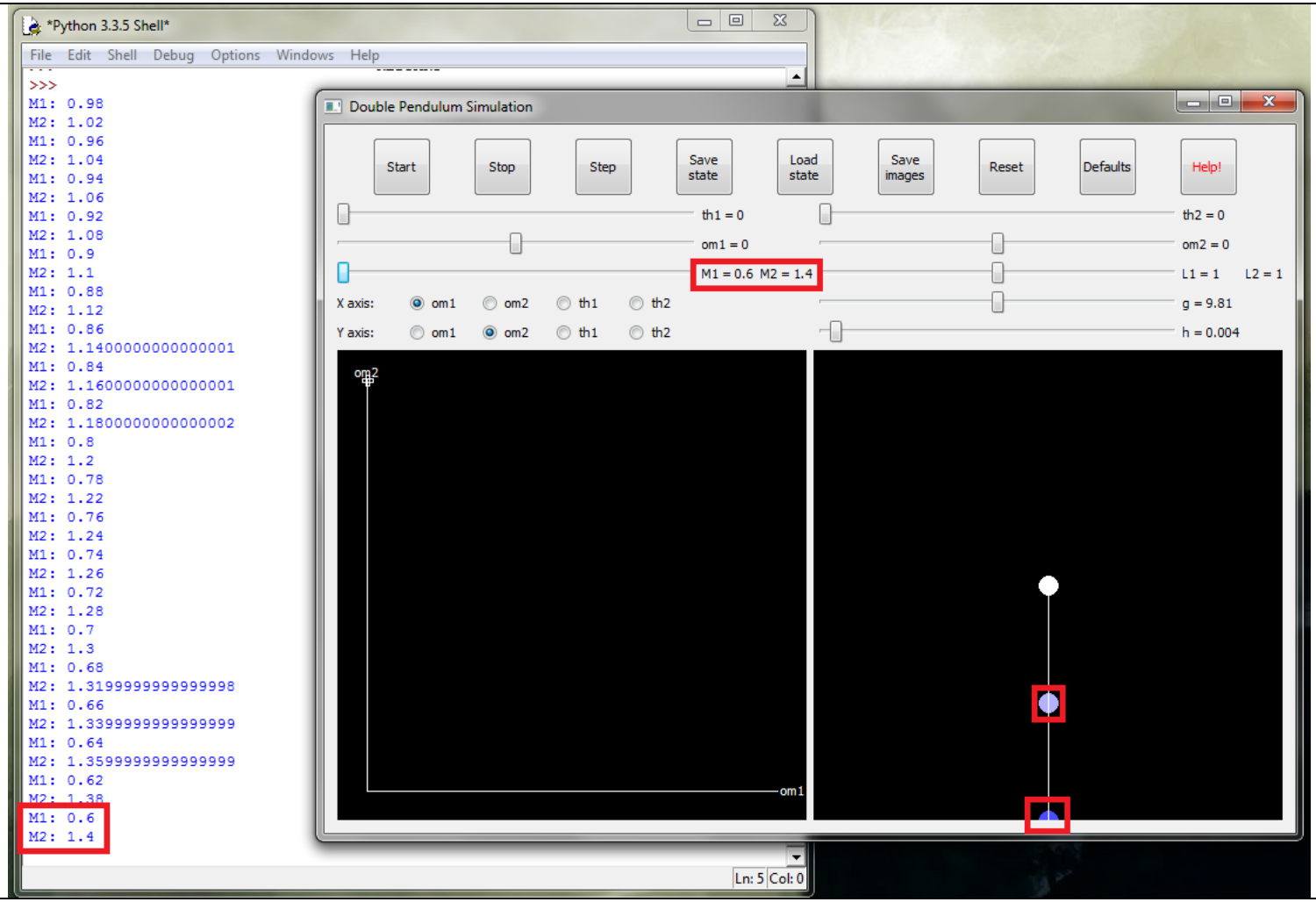
29	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62 D64)] on win32 Type "copyright", "cred >>> ========== >>> Ln: 5 Col: 0 Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 g = 9.81 Y axis: om1 om2 th1 th2 h = 0.004 om2 om1	Passed

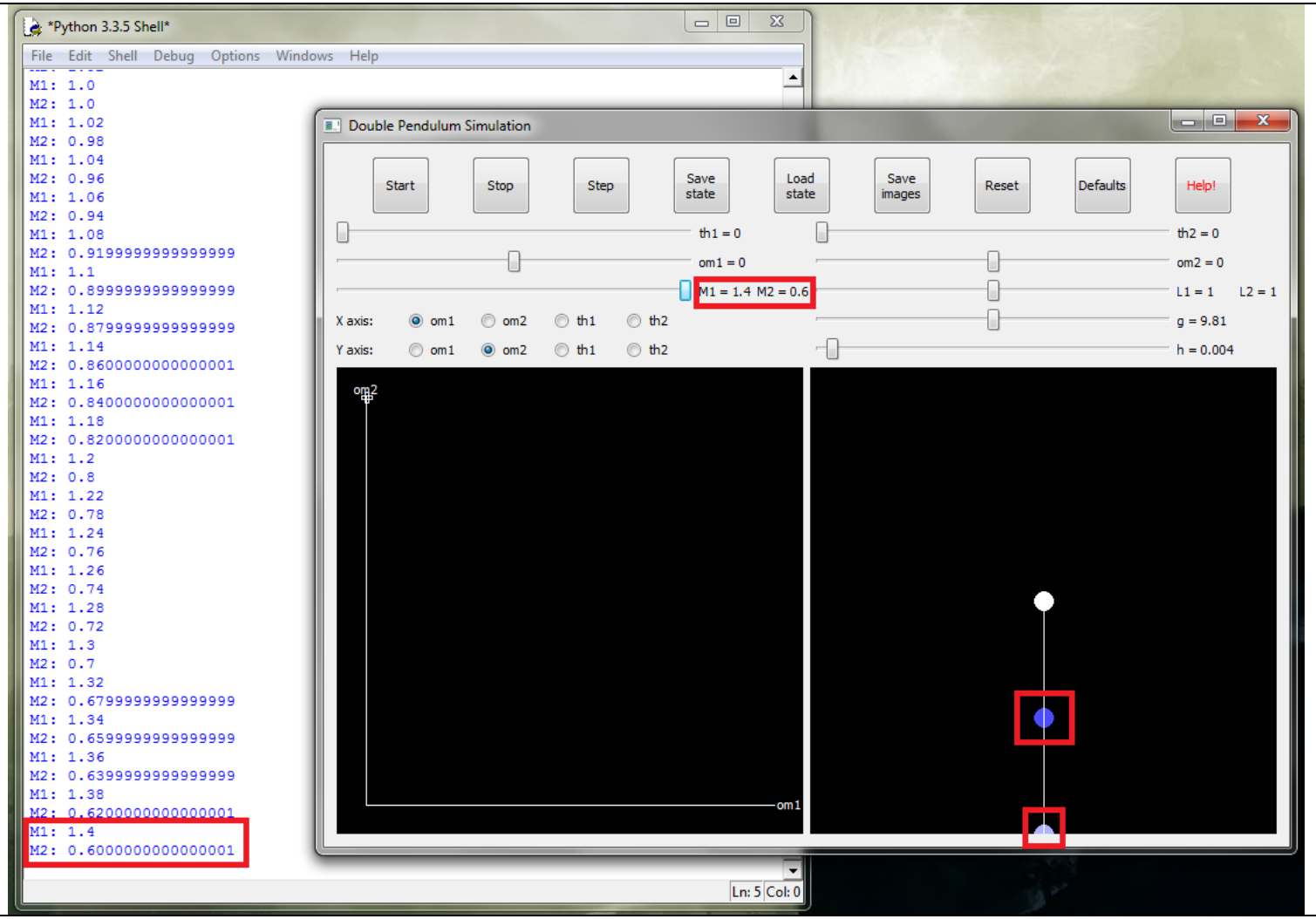

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
D64)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> ========== RESTART ==========
>>>
L1: 0.98
L2: 1.02
L1: 0.96
L2: 1.04
L1: 0.94
L2: 1.06
L1: 0.92
L2: 1.08
L1: 0.9
L2: 1.1
L1: 0.88
L2: 1.12
L1: 0.86
L2: 1.1400000000000001
L1: 0.84
L2: 1.1600000000000001
L1: 0.82
L2: 1.1800000000000002
L1: 0.8
L2: 1.2
L1: 0.78
L2: 1.22
L1: 0.76
L2: 1.24
L1: 0.74
L2: 1.26
L1: 0.72
L2: 1.28
L1: 0.7
L2: 1.3
L1: 0.68
L2: 1.3199999999999998
L1: 0.66
L2: 1.3399999999999999
L1: 0.64
L2: 1.3599999999999999
L1: 0.62
L1: 0.6
L2: 1.4
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 1
M2 = 1
L1 = 0.6 L2 = 1.4
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
g = 9.81
h = 0.004
om2
om1



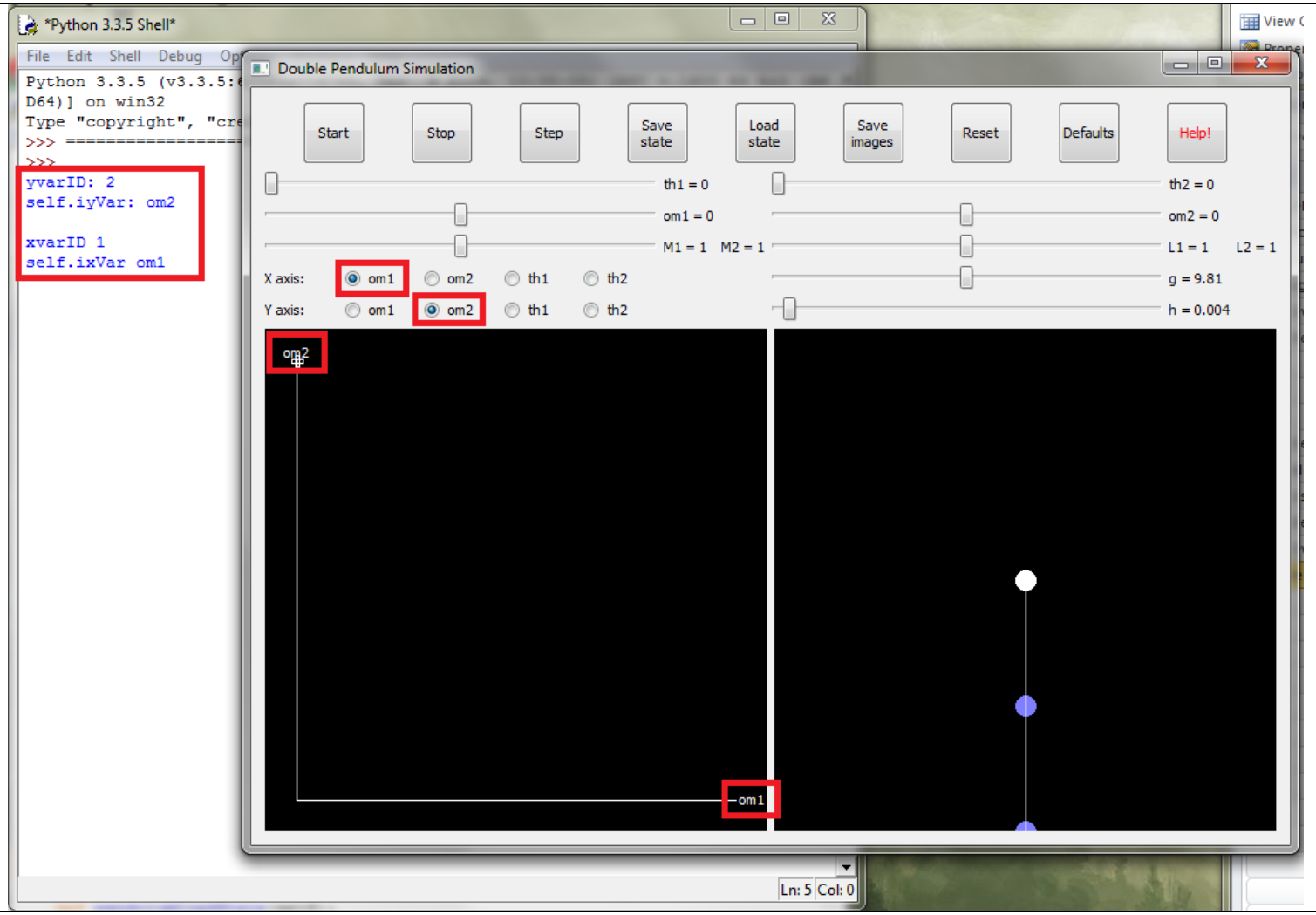
Note that L2 is misrepresented slightly due to floating point inaccuracies; L2 = 0.6

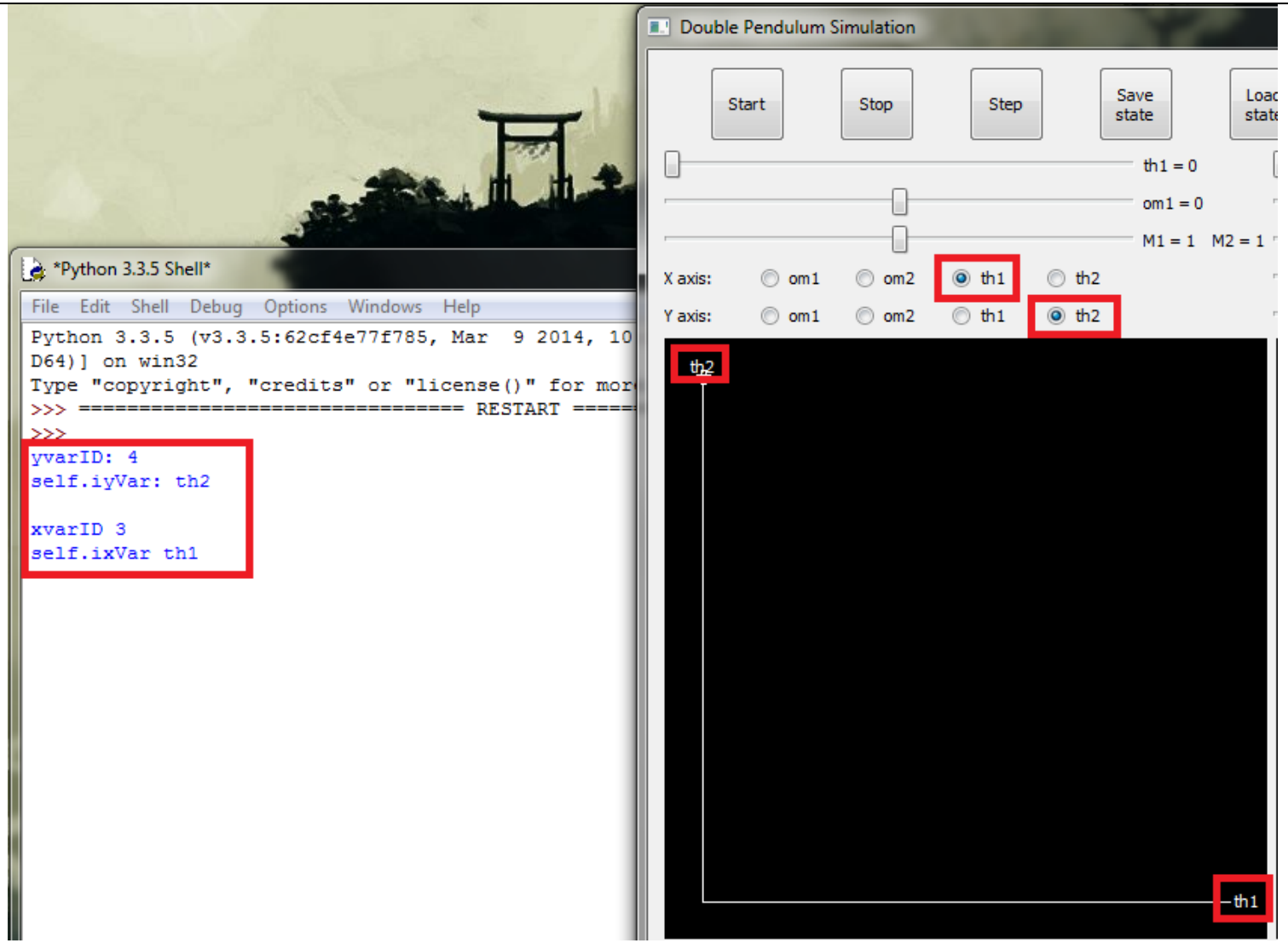
30	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (D64)] on win32 Type "copyrigh >>> ========== >>> Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 g = 9.81 Y axis: om1 om2 th1 th2 h = 0.004 om2 om1 Ln: 5 Col: 0	Passed

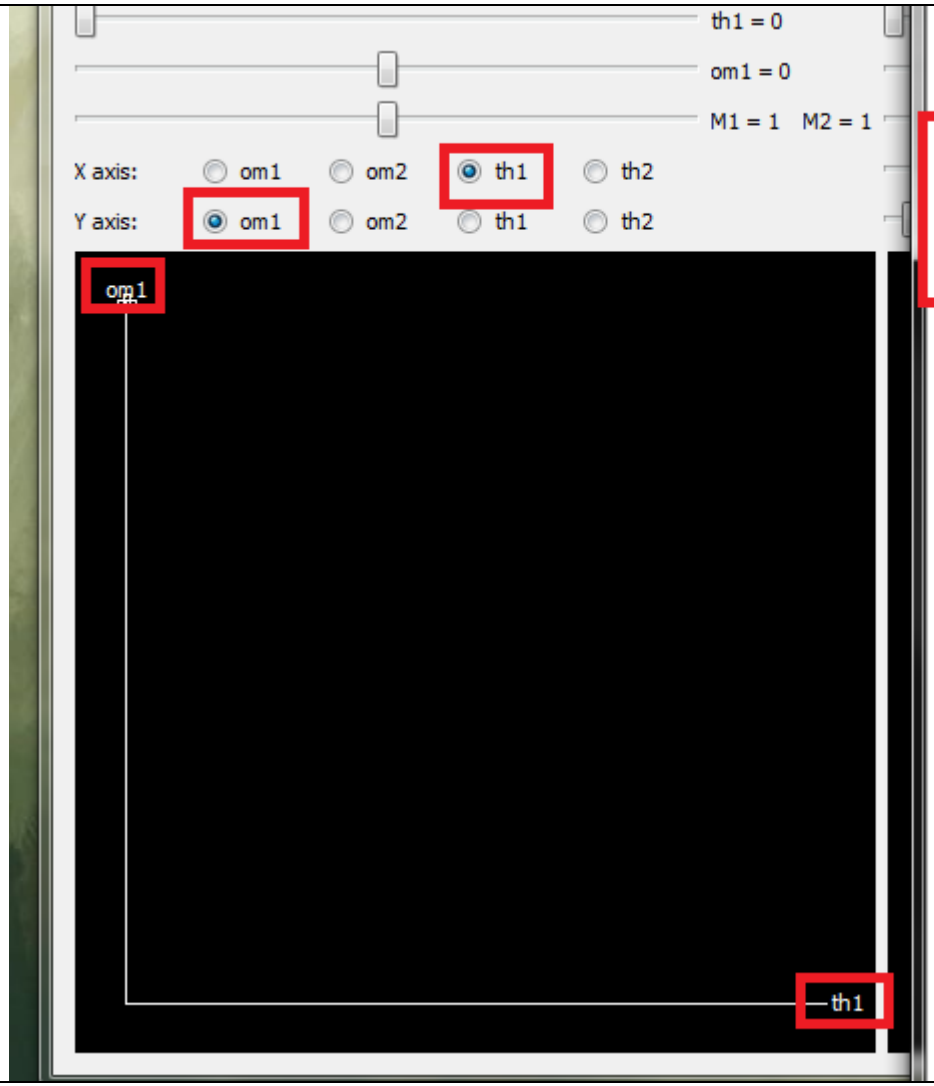

Python 3.3.5 Shell
File Edit Shell Debug Options Windows Help
>>>
M1: 0.98
M2: 1.02
M1: 0.96
M2: 1.04
M1: 0.94
M2: 1.06
M1: 0.92
M2: 1.08
M1: 0.9
M2: 1.1
M1: 0.88
M2: 1.12
M1: 0.86
M2: 1.1400000000000001
M1: 0.84
M2: 1.1600000000000001
M1: 0.82
M2: 1.1800000000000002
M1: 0.8
M2: 1.2
M1: 0.78
M2: 1.22
M1: 0.76
M2: 1.24
M1: 0.74
M2: 1.26
M1: 0.72
M2: 1.28
M1: 0.7
M2: 1.3
M1: 0.68
M2: 1.3199999999999998
M1: 0.66
M2: 1.3399999999999999
M1: 0.64
M2: 1.3599999999999999
M1: 0.62
M2: 1.38
M1: 0.6
M2: 1.4
Ln: 5 Col: 0
Double Pendulum Simulation
Start
Stop
Step
Save state
Load state
Save images
Reset
Defaults
Help!
th1 = 0
th2 = 0
om1 = 0
om2 = 0
M1 = 0.6 M2 = 1.4
L1 = 1
L2 = 1
X axis: om1 om2 th1 th2
Y axis: om1 om2 th1 th2
g = 9.81
h = 0.004
om2
om1

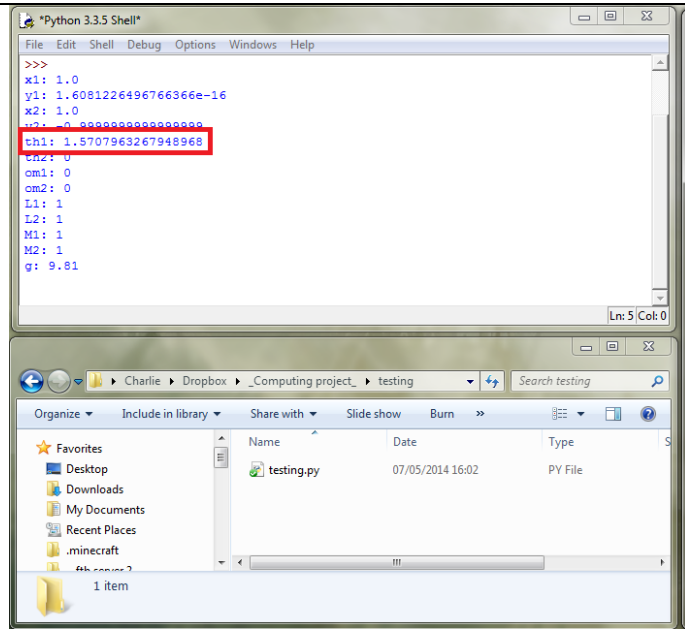
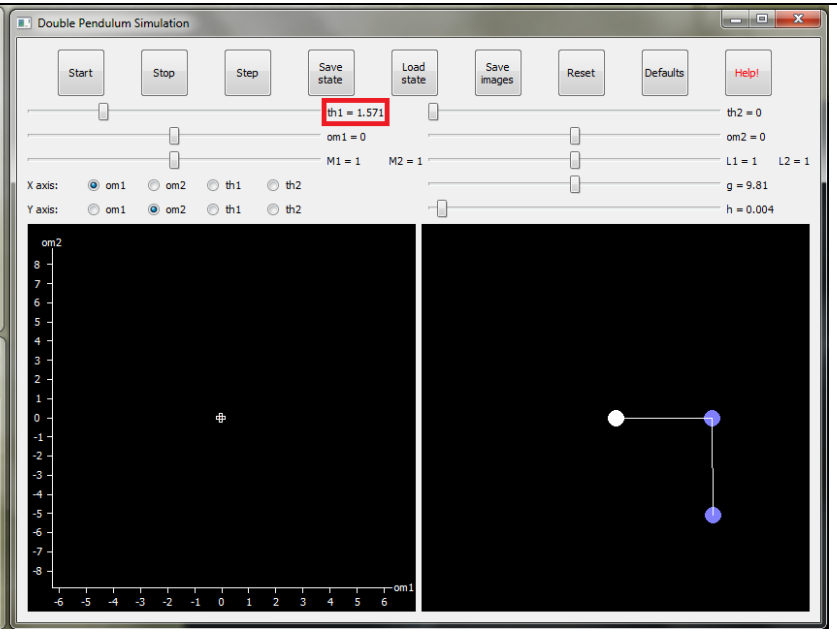


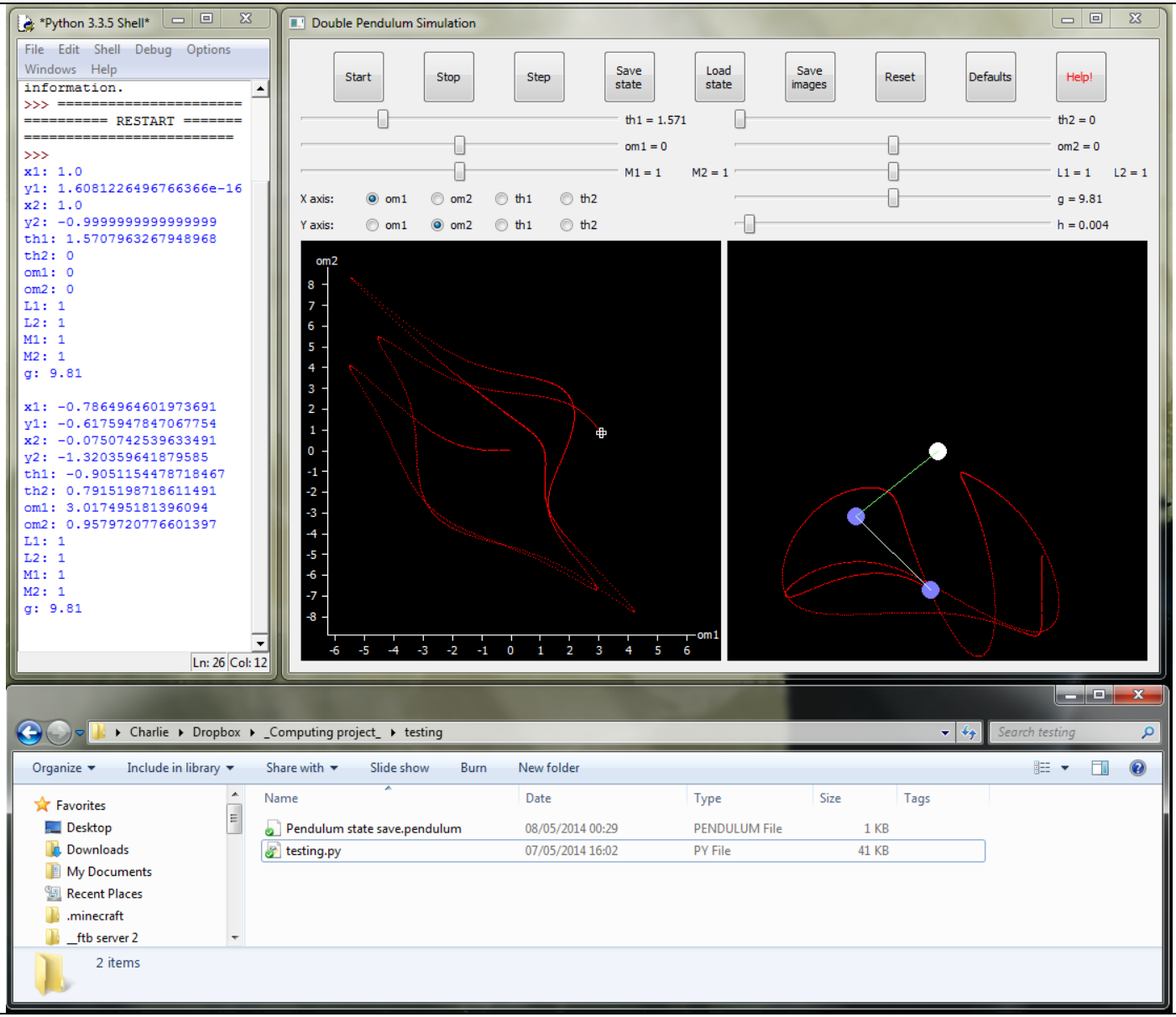
Again, note the floating point inaccuracies

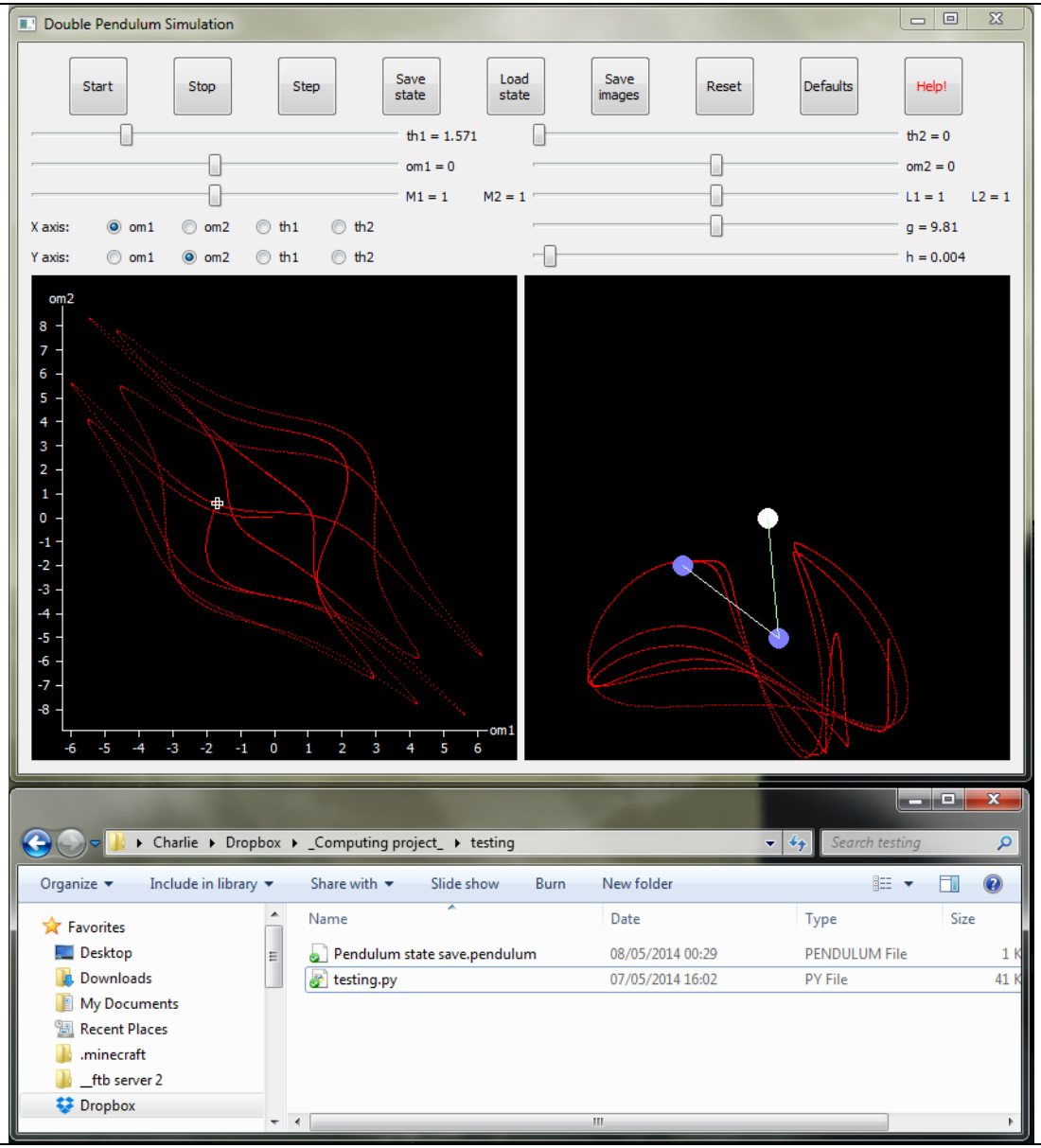
31	 *Python 3.3.5 Shell* File Edit Shell Debug Python 3.3.5 (v3.3.5: D64)] on win32 Type "copyright", "cre >>> ========== >>> yvarID: 2 self.iyVar: om2 xvarID 1 self.ixVar om1 Ln: 5 Col: 0 Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 0 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 g = 9.81 Y axis: om1 om2 th1 th2 h = 0.004 om2 om1	Passed

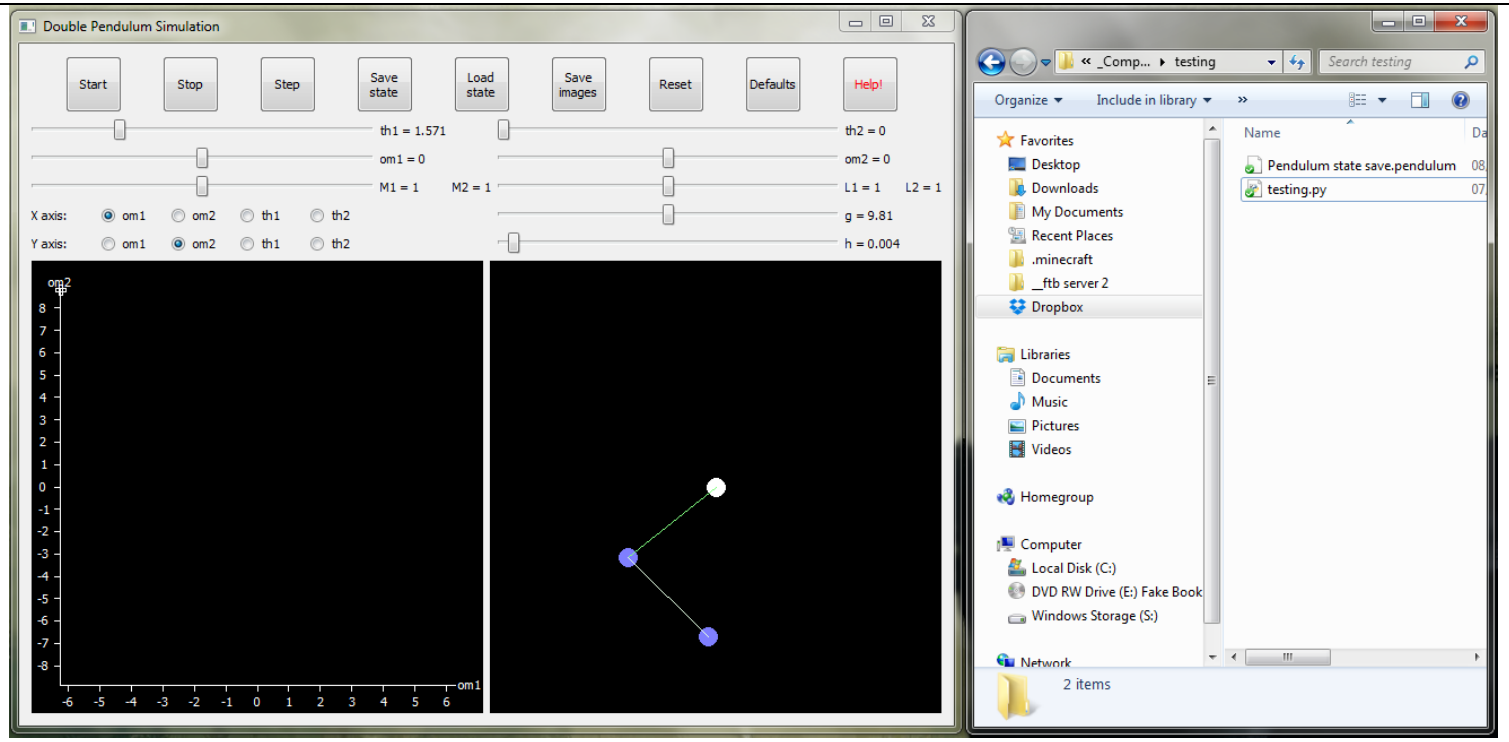
32	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help Python 3.3.5 (v3.3.5:62cf4e77f785, Mar 9 2014, 10 D64)] on win32 Type "copyright", "credits" or "license()" for mor >>> ========== RESTART ===== >>> yvarID: 4 self.iyVar: th2 xvarID 3 self.ixVar th1 Double Pendulum Simulation Start Stop Step Save state th1 = 0 om1 = 0 M1 = 1 M2 = 1 X axis: om1 om2 th1 th2 Y axis: om1 om2 th1 th2 th2 th1	Passed

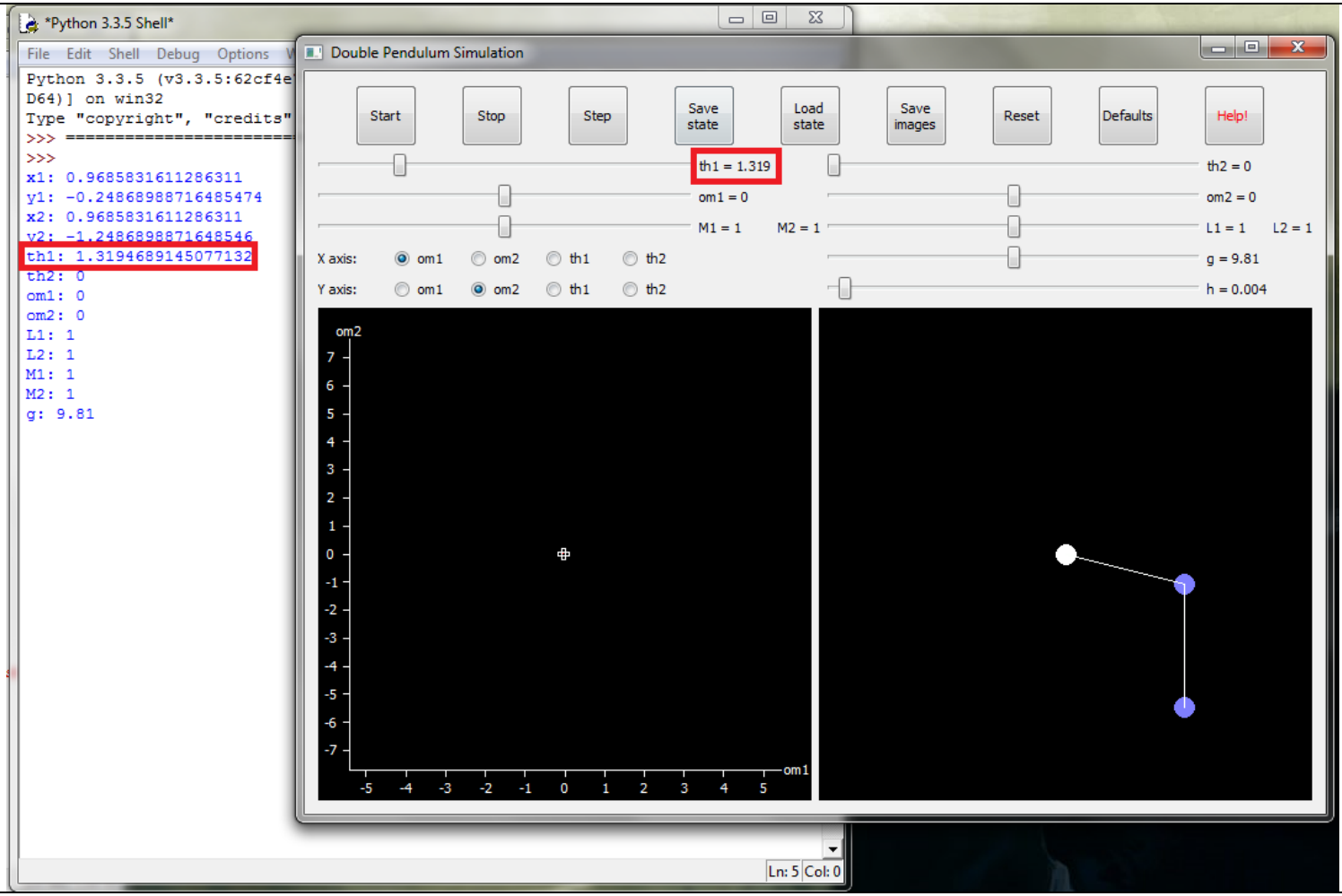
33		D64)] on win32 Type "copyright", " >>> ========== >>> yvarID: 1 self.iyVar: om1 xvarID 3 self.ixVar th1	Passed

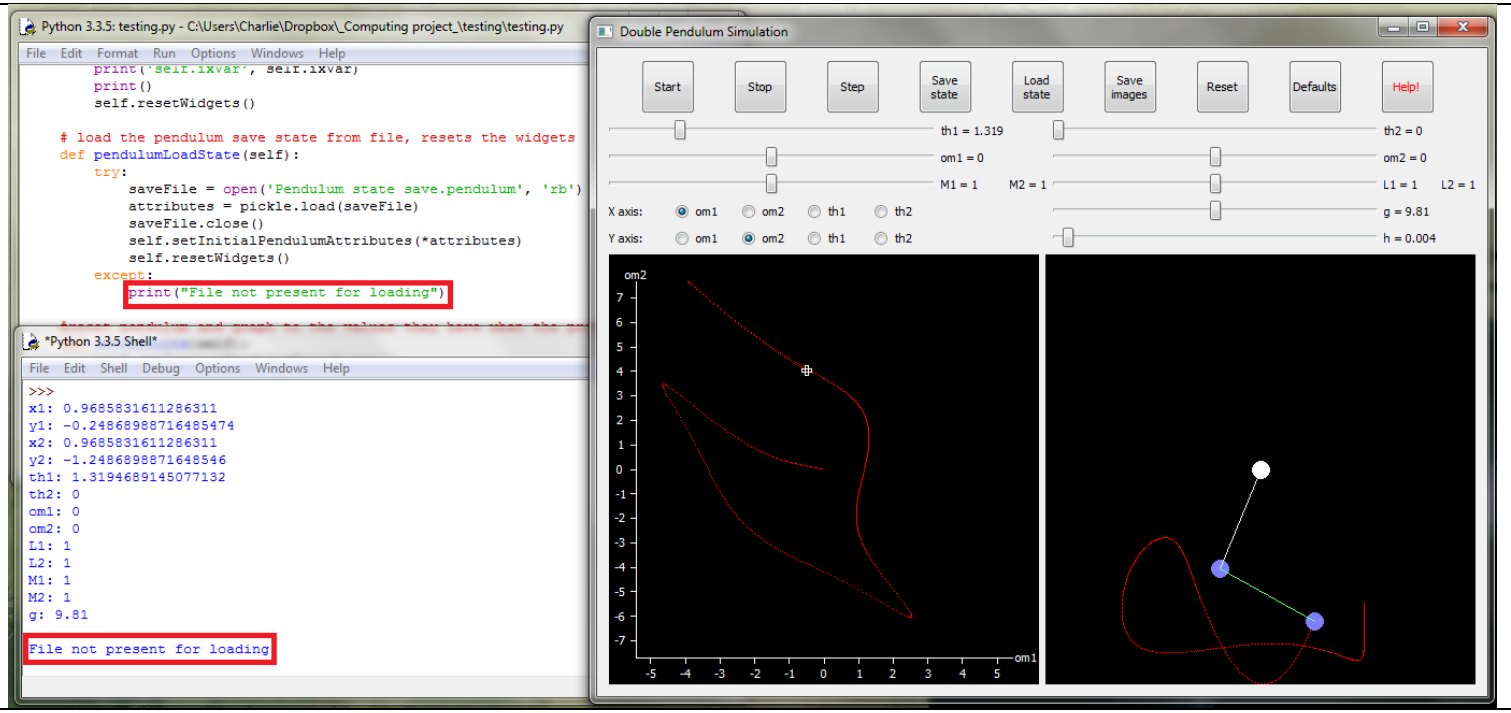
34	 *Python 3.3.5 Shell* File Edit Shell Debug Options Windows Help >>> x1: 1.0 y1: 1.6081226496766366e-16 x2: 1.0 th1: 1.5707963267948968 th2: 0 om1: 0 om2: 0 L1: 1 L2: 1 M1: 1 M2: 1 g: 9.81 Ln: 5 Col: 0 Charlie ▸ Dropbox ▸ _Computing project_ ▸ testing Search testing Organize Include in library Share with Slide show Burn Favorites Desktop Downloads My Documents Recent Places .minecraft Name Date Type testing.py 07/05/2014 16:02 PY File 1 item  Double Pendulum Simulation Start Stop Step Save state Load state Save images Reset Defaults Help! th1 = 1.571 th2 = 0 om1 = 0 om2 = 0 M1 = 1 M2 = 1 L1 = 1 L2 = 1 X axis: om1 om2 th1 th2 Y axis: om1 om2 th1 th2 g = 9.81 h = 0.004 om2 om1	Passed

		After 5 seconds, state saved

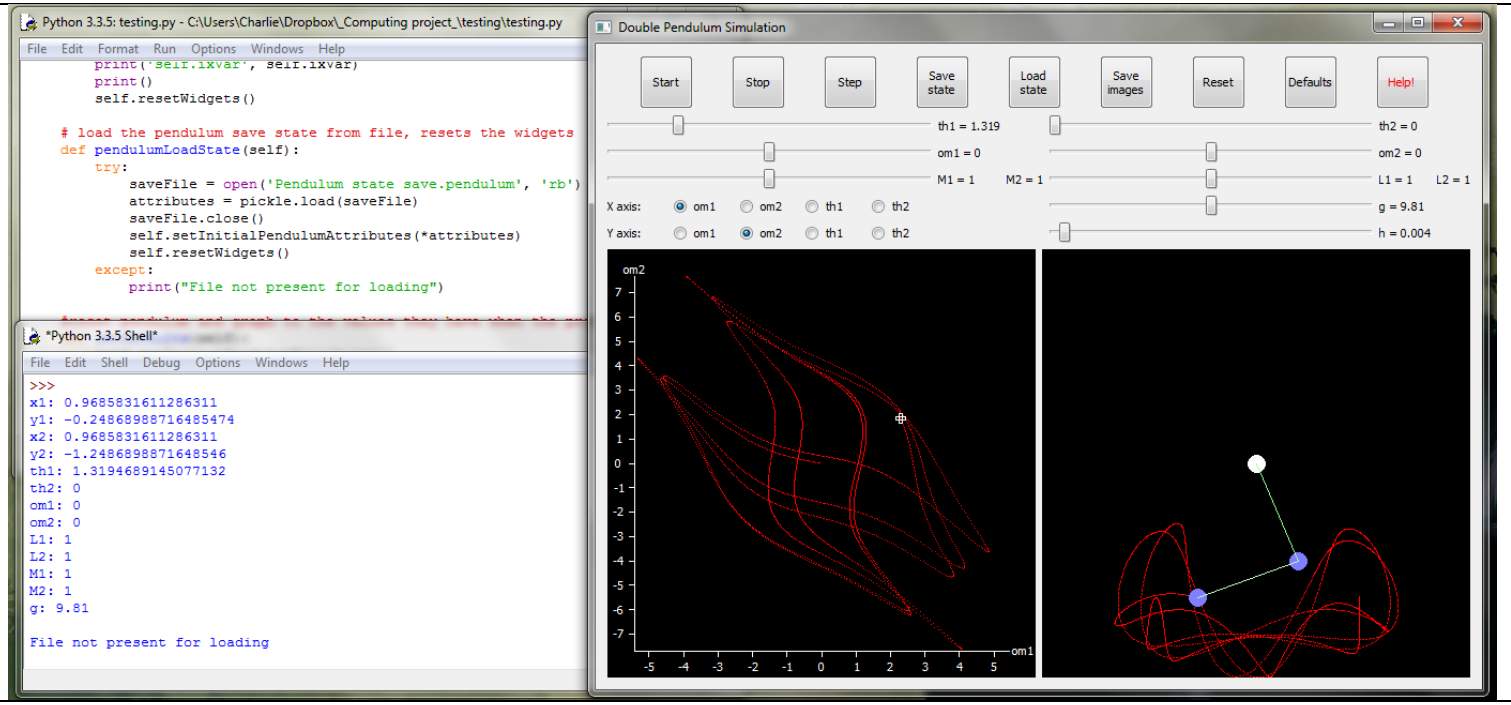
			After 10 seconds, before state load

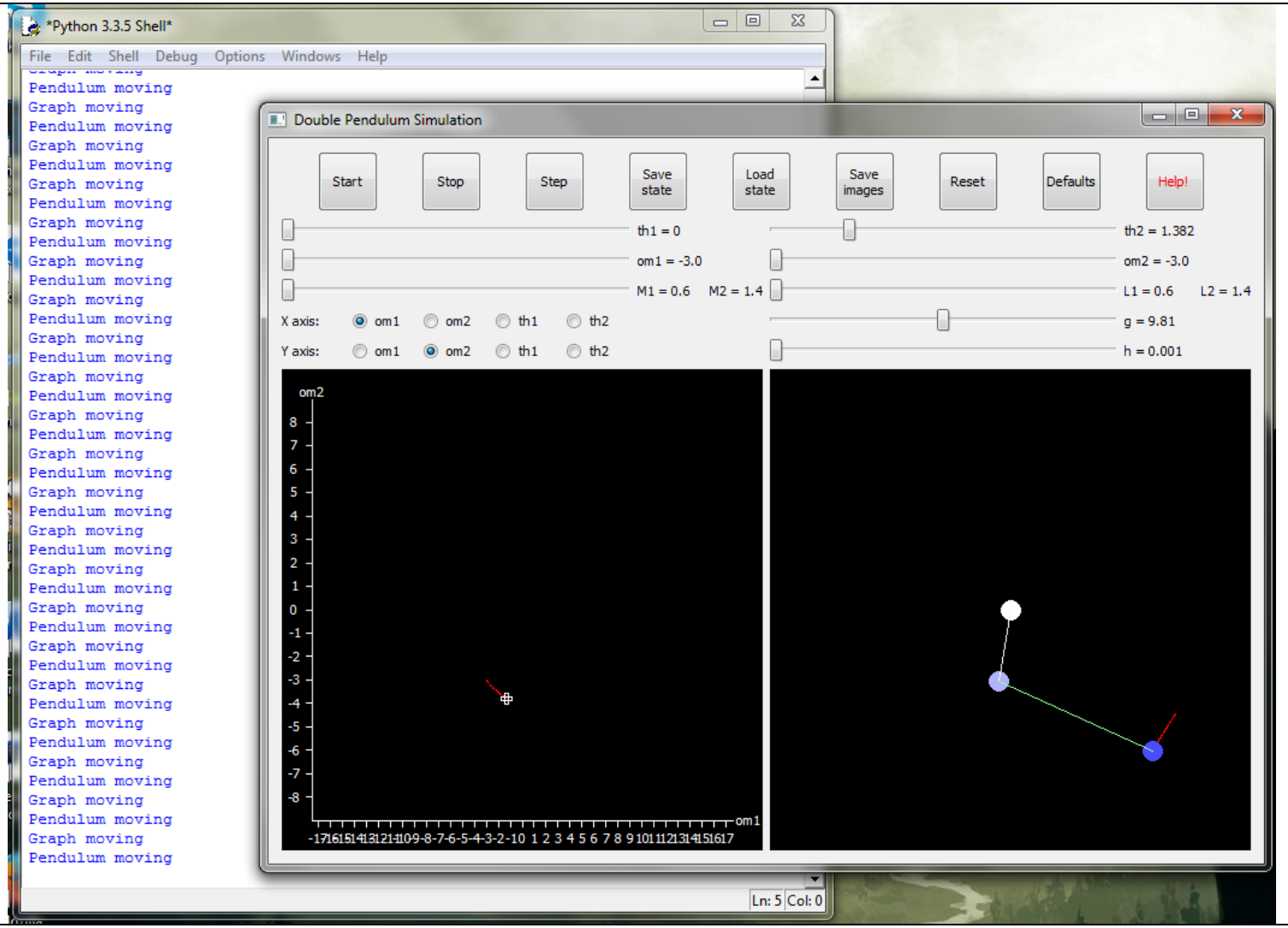
		State successfully loaded

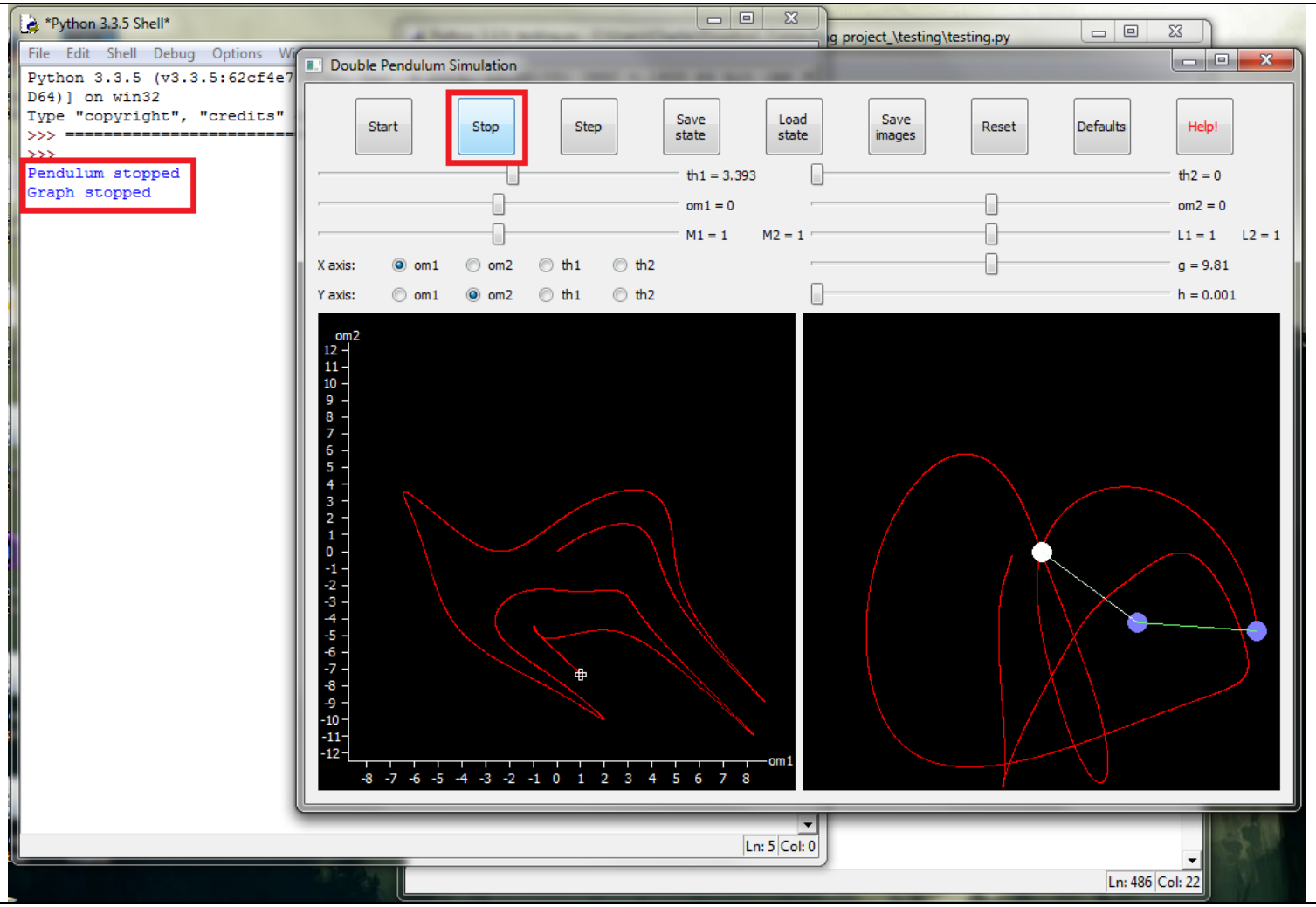
35		Pendulum at start of simulation

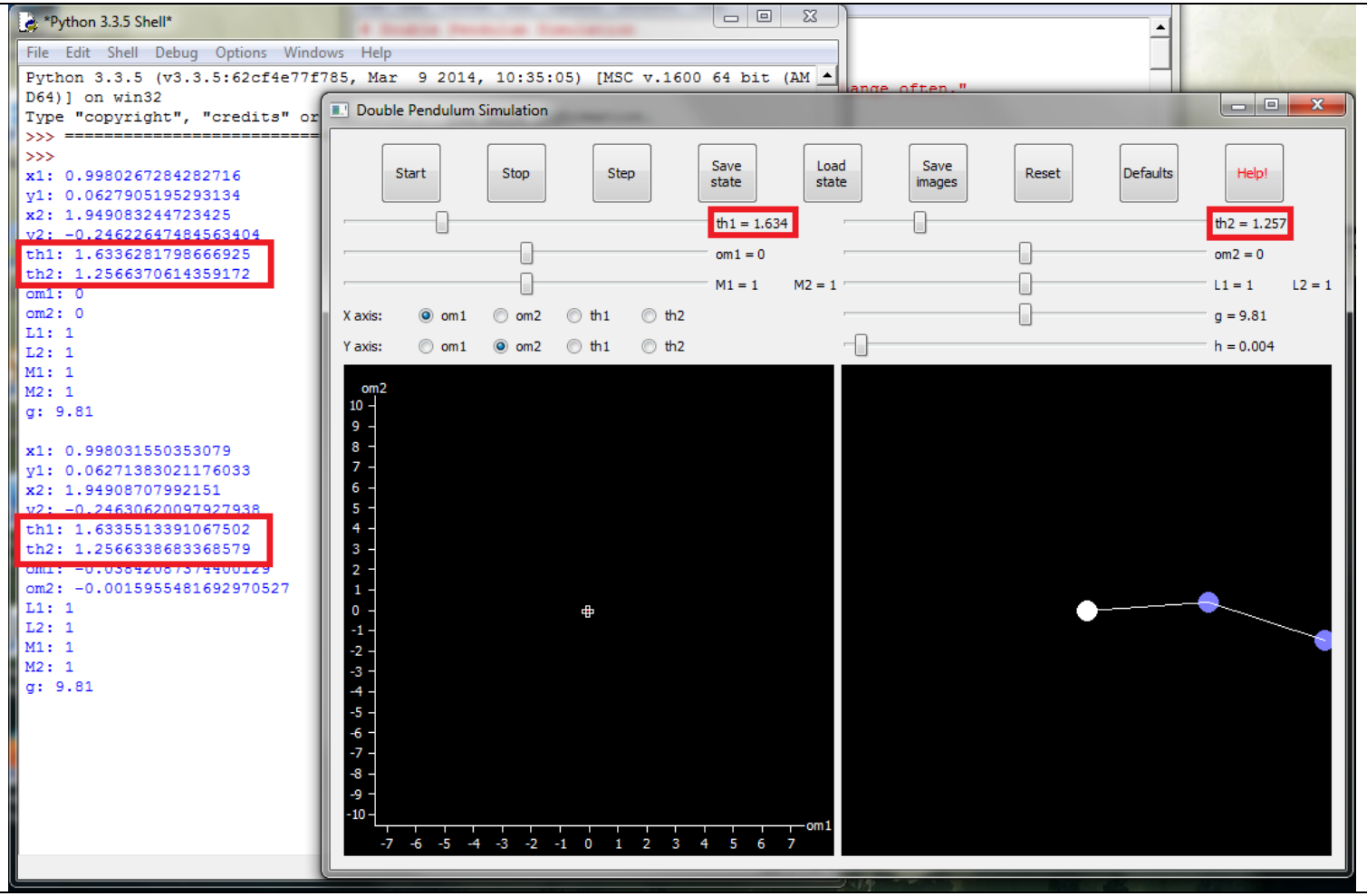


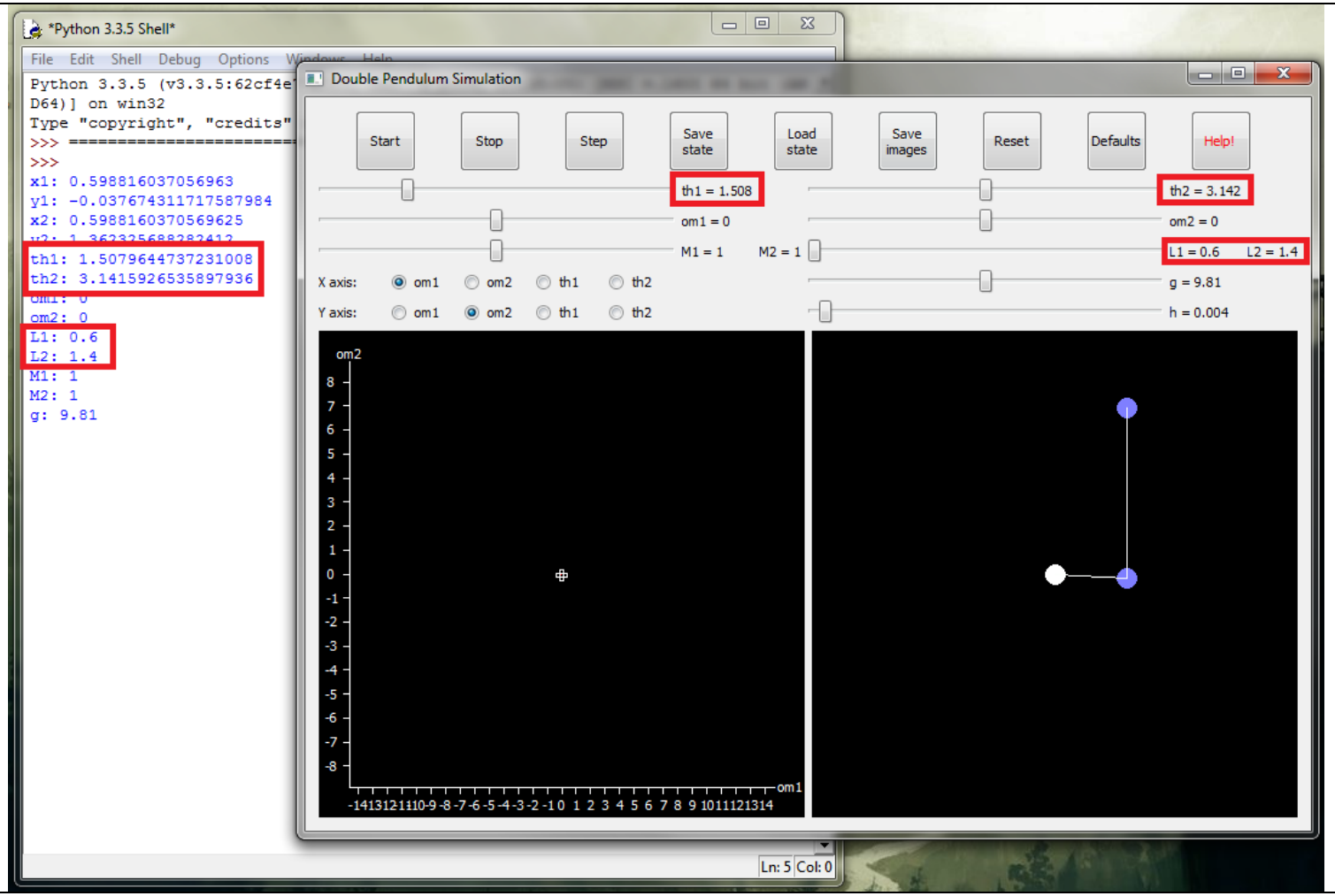
Pendulum after it has been allowed to move, then a load attempt was (intentionally) unsuccessfully made

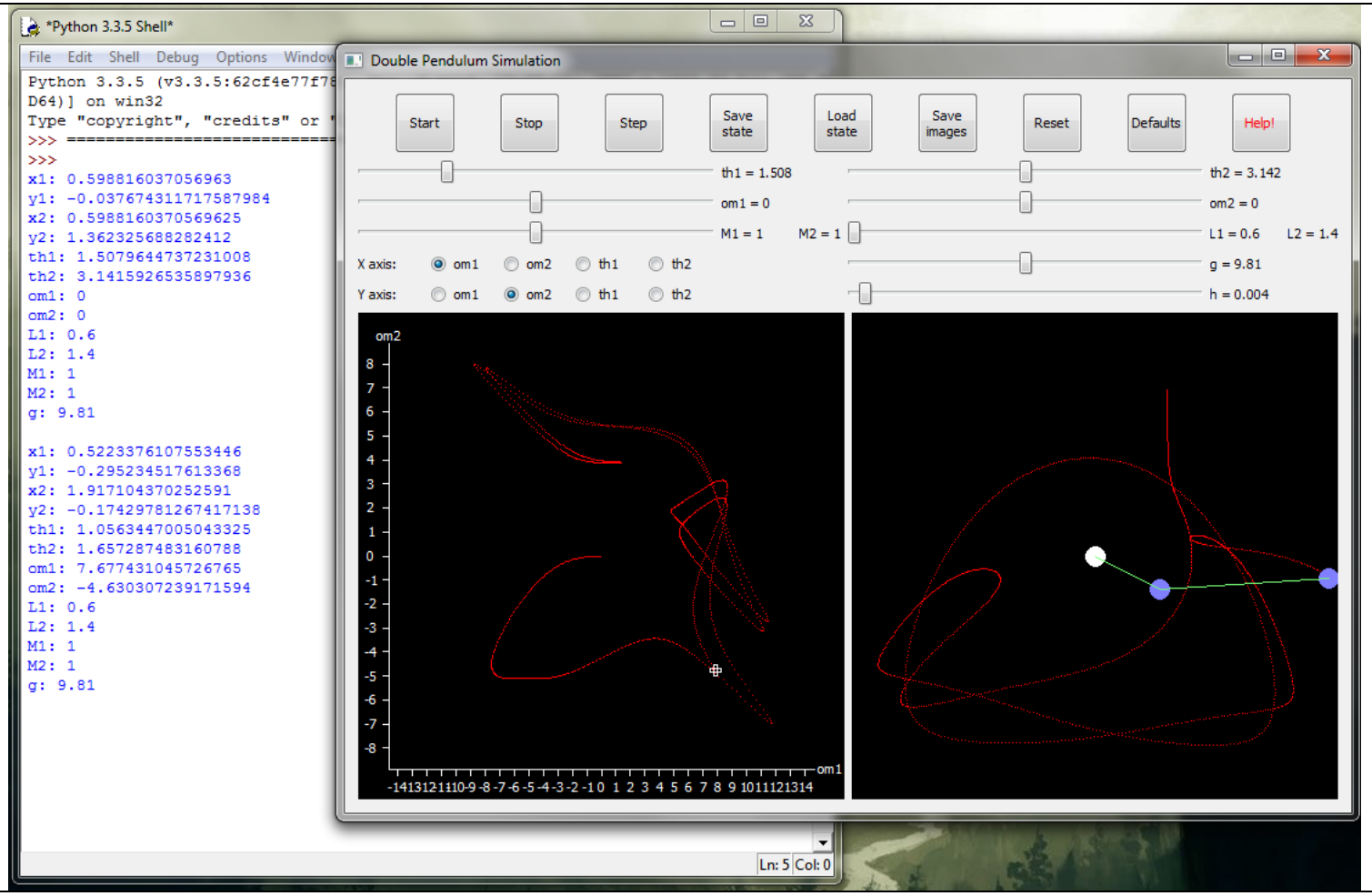
		Pendulum continues as normal even if load attempt fails. Passed.

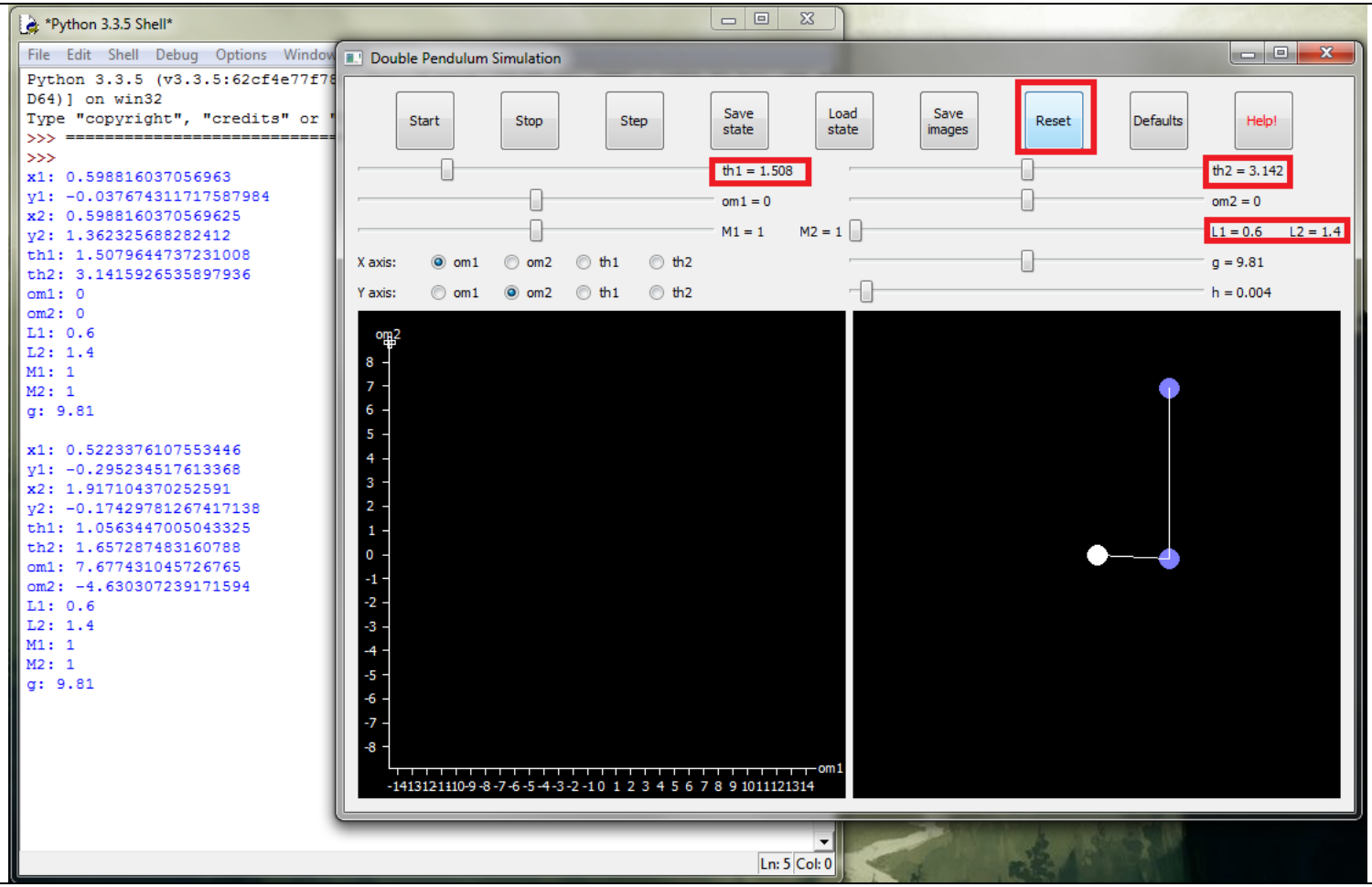
36		Start button sets both graph and pendulum moving

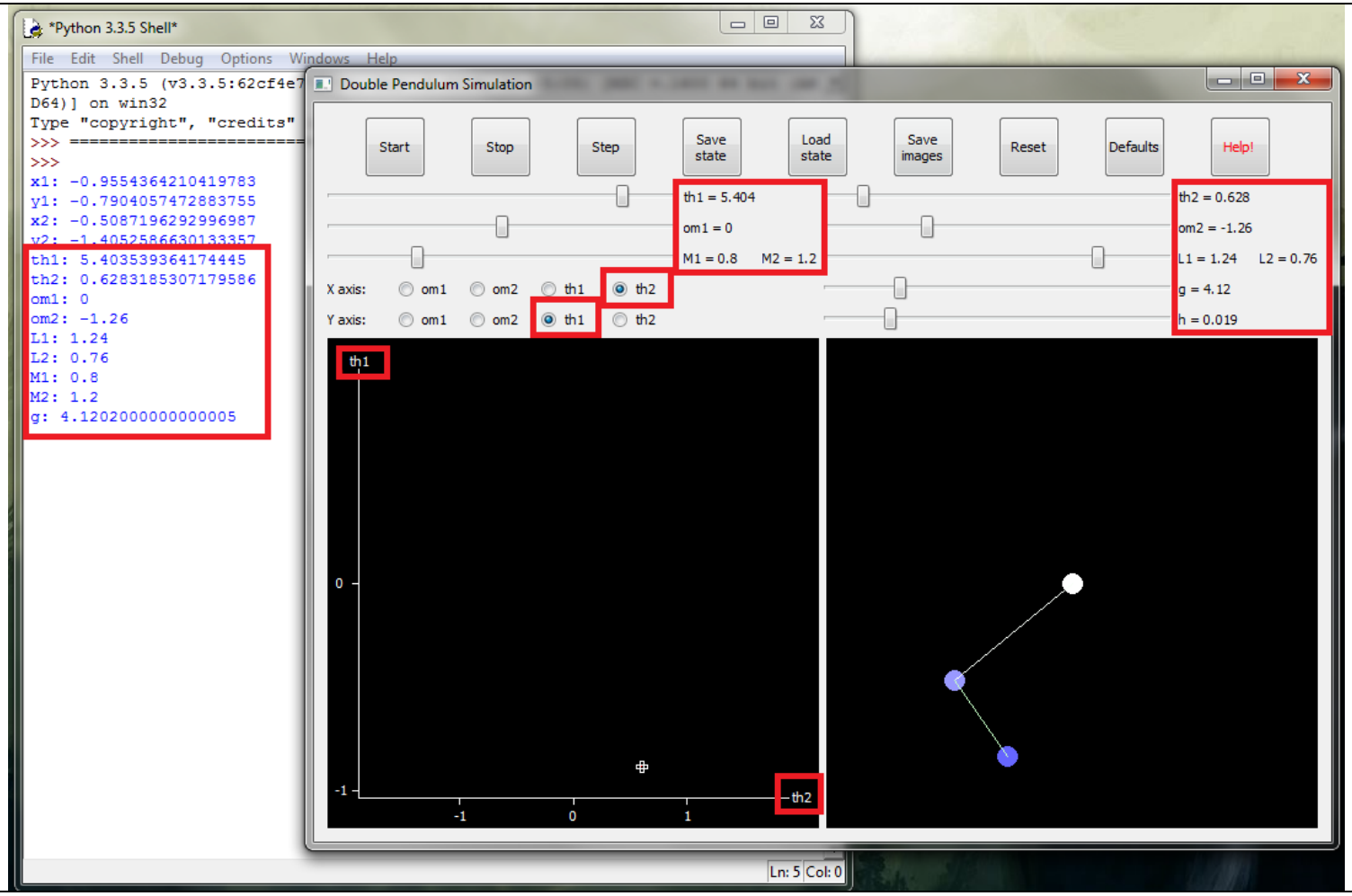
37		Stop button successfully pauses simulation. Passed.

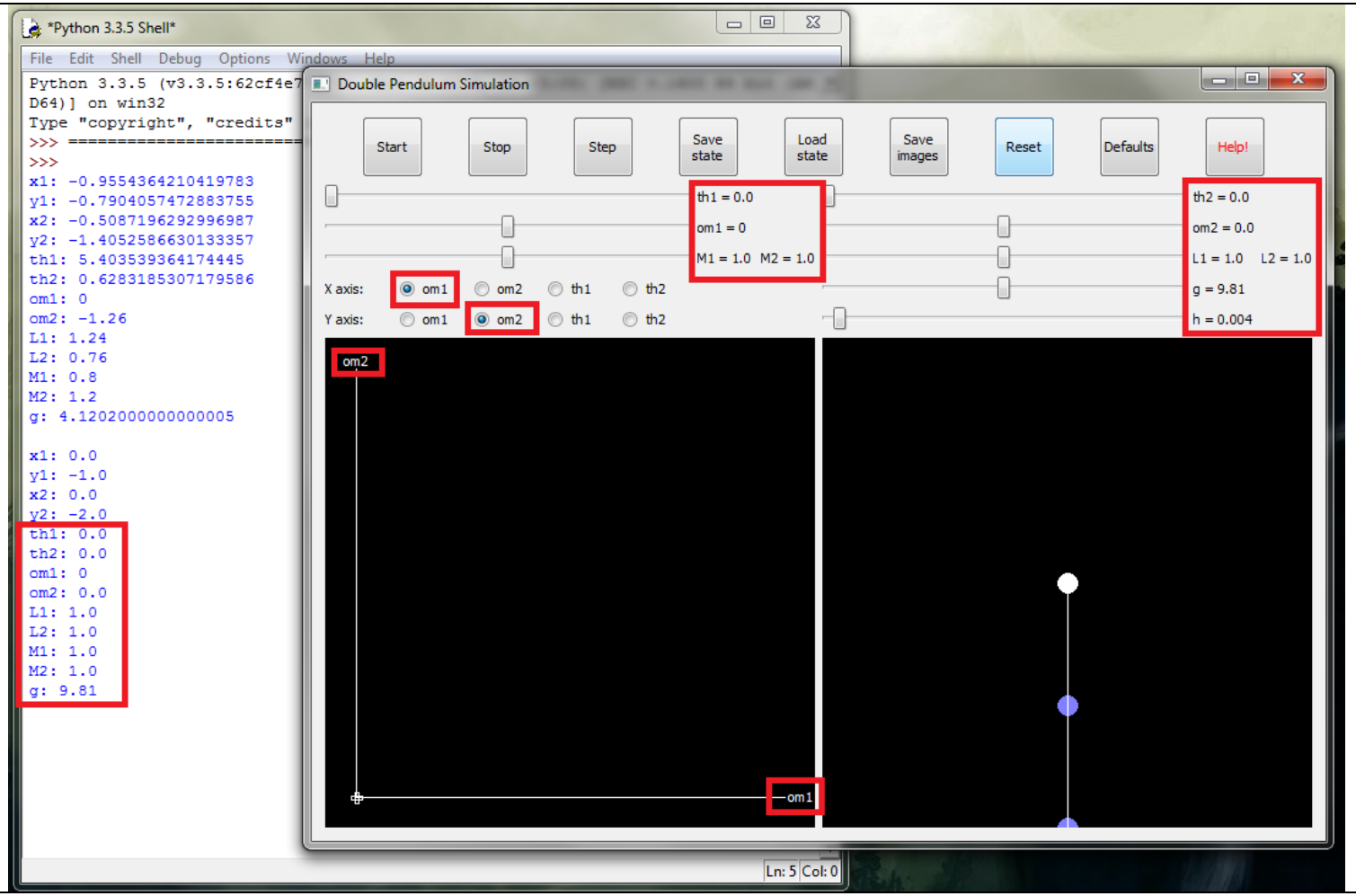
38		First printout is the initial conditions, the second print out is a step later. Both arms have rotated anticlockwise slightly due to gravity. Passed.

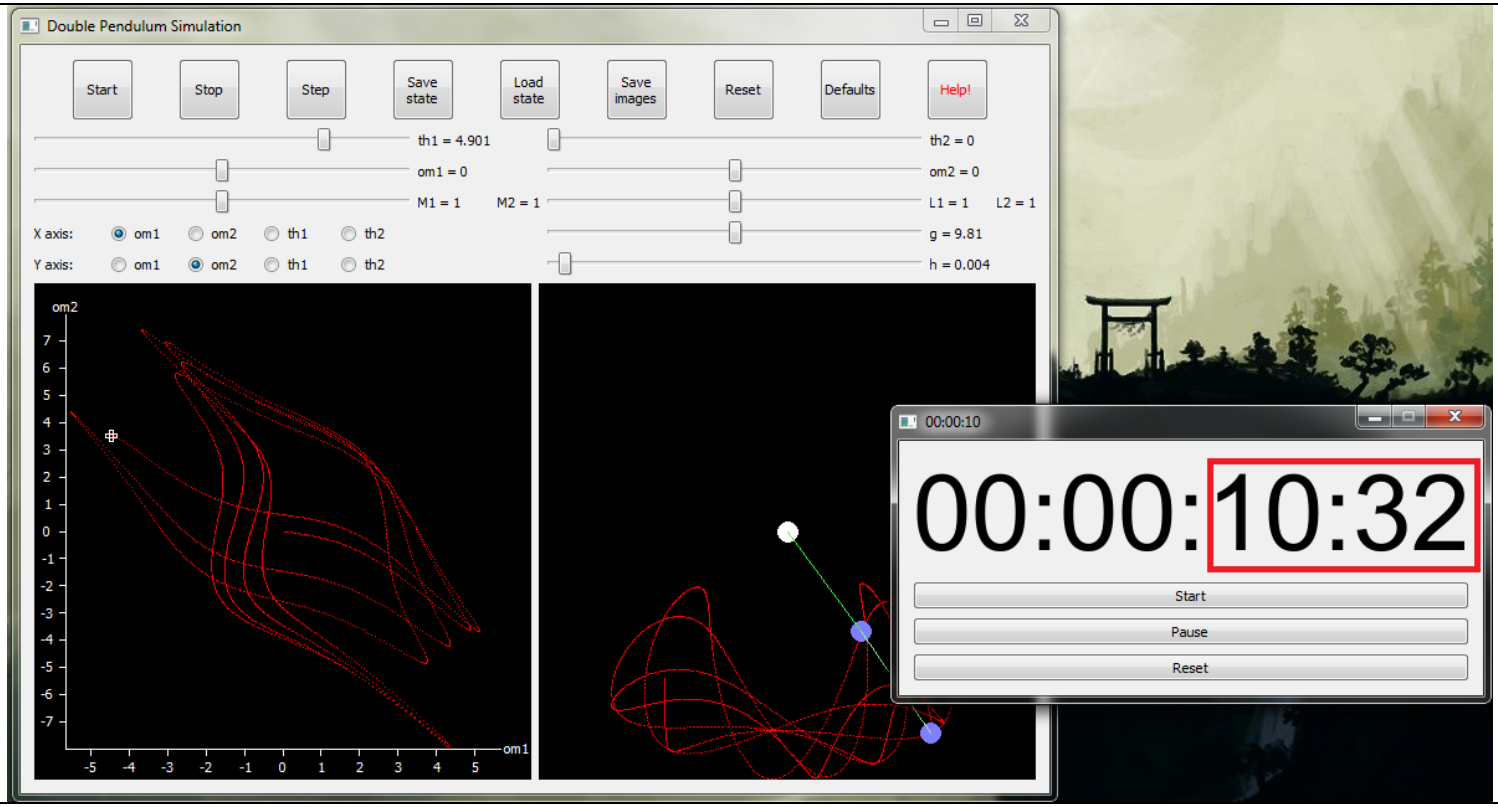
39		Pendulum's initial conditions

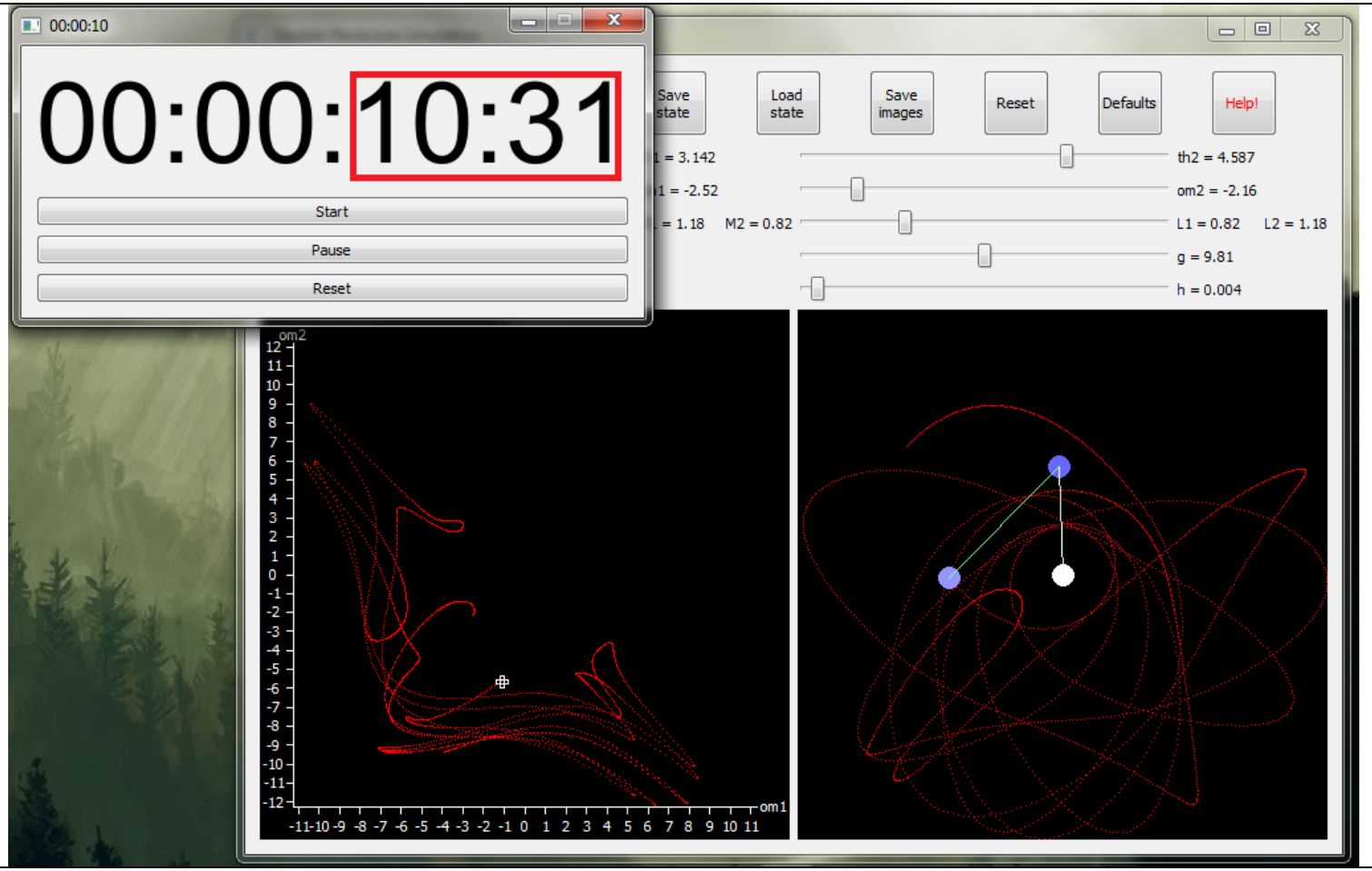
		Pendulum allowed to run

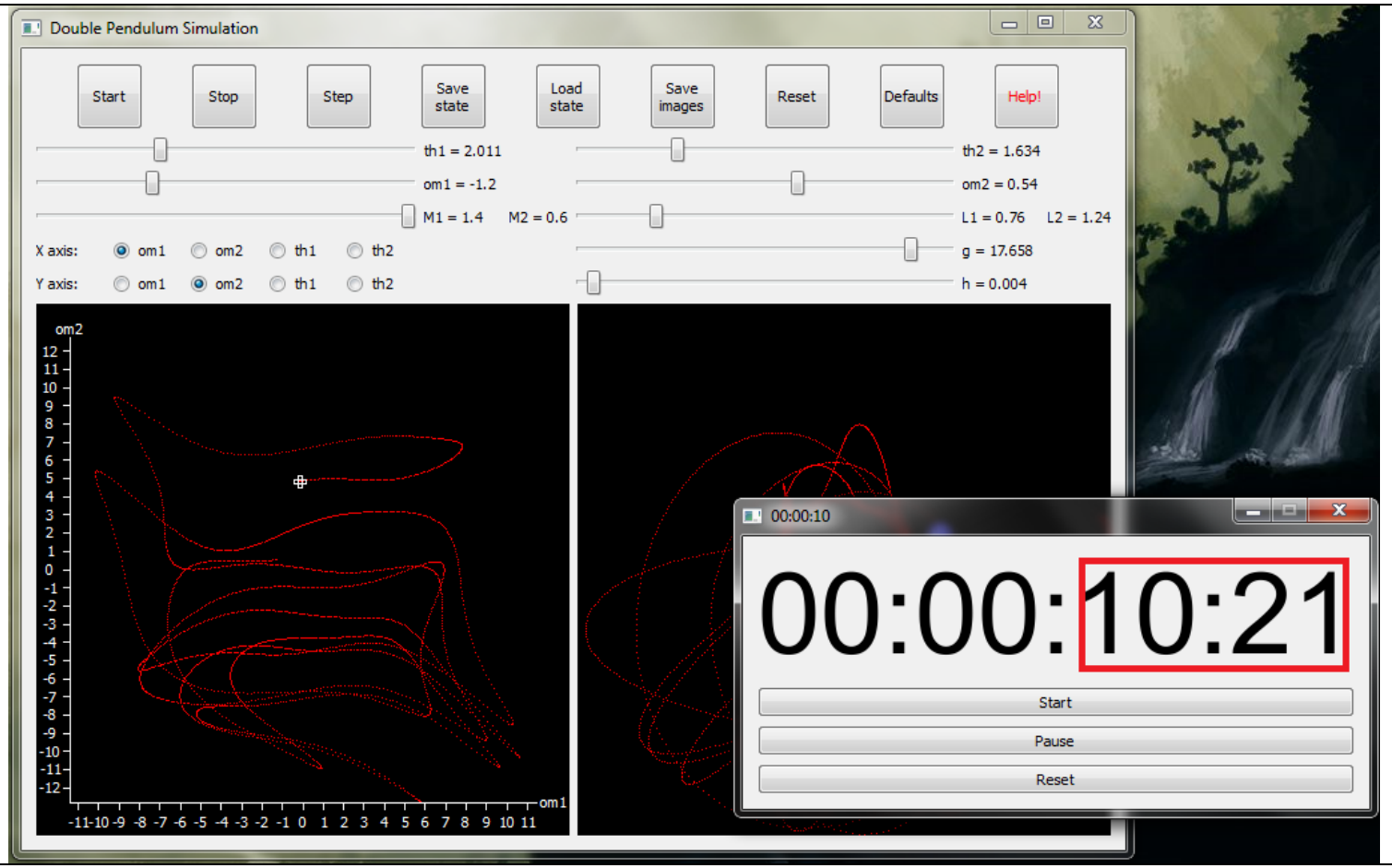
		Pendulum successfully returned to initial conditions. Passed.

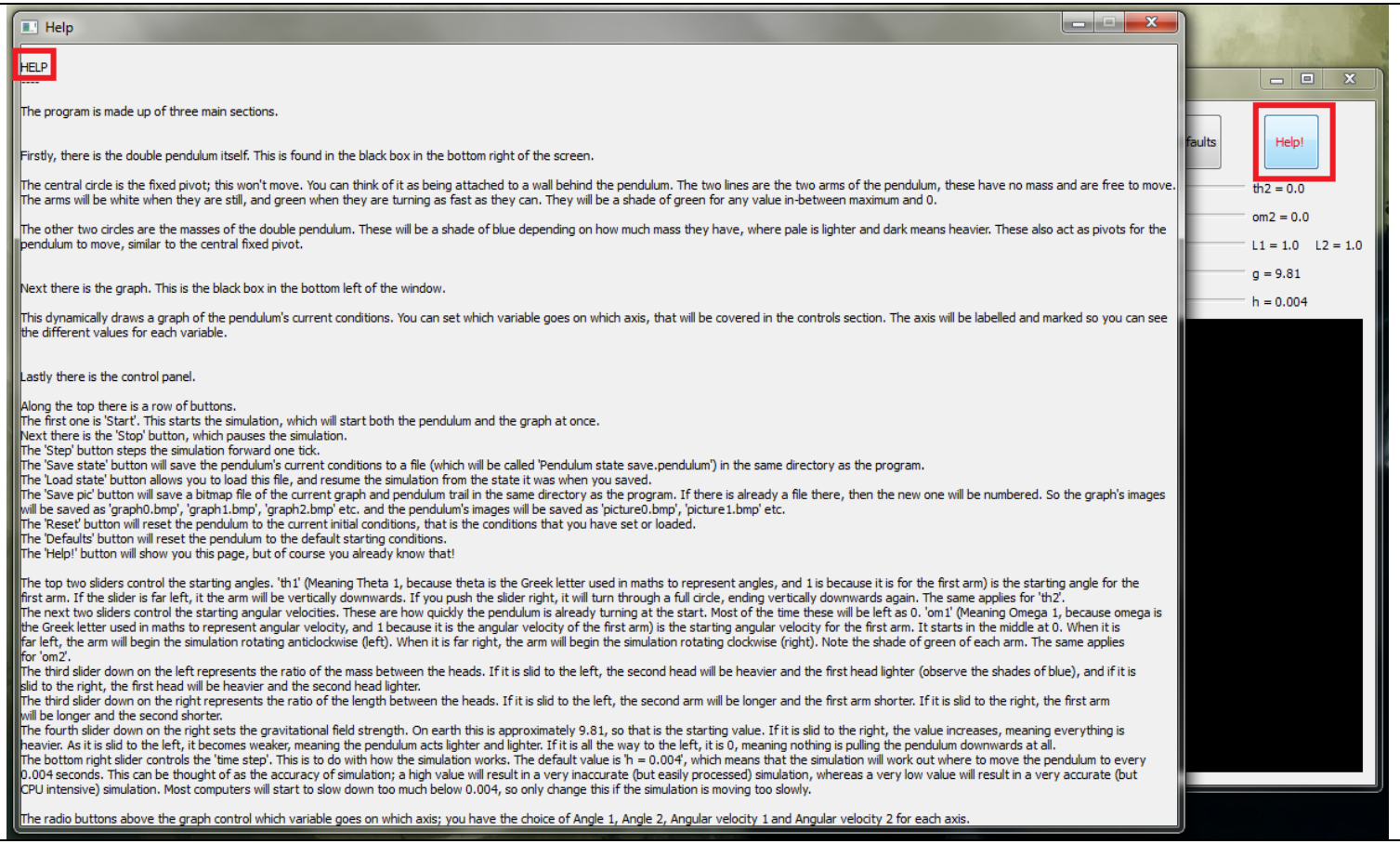
40		Pendulum set to initial conditions

		Pendulum successfully returns to defaults. Passed.

41		Very close to ten, passed.

		Time close to 10 and close to last one, strongly suggesting the ~30ms difference is due to human reaction time. Passed.

		Passed. Note the varying initial conditions of each test.

42	 Help HELP The program is made up of three main sections. Firstly, there is the double pendulum itself. This is found in the black box in the bottom right of the screen. The central circle is the fixed pivot; this won't move. You can think of it as being attached to a wall behind the pendulum. The two lines are the two arms of the pendulum, these have no mass and are free to move. The arms will be white when they are still, and green when they are turning as fast as they can. They will be a shade of green for any value in-between maximum and 0. The other two circles are the masses of the double pendulum. These will be a shade of blue depending on how much mass they have, where pale is lighter and dark means heavier. These also act as pivots for the pendulum to move, similar to the central fixed pivot. Next there is the graph. This is the black box in the bottom left of the window. This dynamically draws a graph of the pendulum's current conditions. You can set which variable goes on which axis, that will be covered in the controls section. The axis will be labelled and marked so you can see the different values for each variable. Lastly there is the control panel. Along the top there is a row of buttons. The first one is 'Start'. This starts the simulation, which will start both the pendulum and the graph at once. Next there is the 'Stop' button, which pauses the simulation. The 'Step' button steps the simulation forward one tick. The 'Save state' button will save the pendulum's current conditions to a file (which will be called 'Pendulum state save.pendulum') in the same directory as the program. The 'Load state' button allows you to load this file, and resume the simulation from the state it was when you saved. The 'Save pic' button will save a bitmap file of the current graph and pendulum trail in the same directory as the program. If there is already a file there, then the new one will be numbered. So the graph's images will be saved as 'graph0.bmp', 'graph1.bmp', 'graph2.bmp' etc. and the pendulum's images will be saved as 'picture0.bmp', 'picture1.bmp' etc. The 'Reset' button will reset the pendulum to the current initial conditions, that is the conditions that you have set or loaded. The 'Defaults' button will reset the pendulum to the default starting conditions. The 'Help!' button will show you this page, but of course you already know that! The top two sliders control the starting angles. 'th1' (Meaning Theta 1, because theta is the Greek letter used in maths to represent angles, and 1 is because it is for the first arm) is the starting angle for the first arm. If the slider is far left, it the arm will be vertically downwards. If you push the slider right, it will turn through a full circle, ending vertically downwards again. The same applies for 'th2'. The next two sliders control the starting angular velocities. These are how quickly the pendulum is already turning at the start. Most of the time these will be left as 0. 'om1' (Meaning Omega 1, because omega is the Greek letter used in maths to represent angular velocity, and 1 because it is the angular velocity of the first arm) is the starting angular velocity for the first arm. It starts in the middle at 0. When it is far left, the arm will begin the simulation rotating anticlockwise (left). When it is far right, the arm will begin the simulation rotating clockwise (right). Note the shade of green of each arm. The same applies for 'om2'. The third slider down on the left represents the ratio of the mass between the heads. If it is slid to the left, the second head will be heavier and the first head lighter (observe the shades of blue), and if it is slid to the right, the first head will be heavier and the second head lighter. The third slider down on the right represents the ratio of the length between the heads. If it is slid to the left, the second arm will be longer and the first arm shorter. If it is slid to the right, the first arm will be longer and the second shorter. The fourth slider down on the right sets the gravitational field strength. On earth this is approximately 9.81, so that is the starting value. If it is slid to the right, the value increases, meaning everything is heavier. As it is slid to the left, it becomes weaker, meaning the pendulum acts lighter and lighter. If it is all the way to the left, it is 0, meaning nothing is pulling the pendulum downwards at all. The bottom right slider controls the 'time step'. This is to do with how the simulation works. The default value is 'h = 0.004', which means that the simulation will work out where to move the pendulum to every 0.004 seconds. This can be thought of as the accuracy of simulation; a high value will result in a very inaccurate (but easily processed) simulation, whereas a very low value will result in a very accurate (but CPU intensive) simulation. Most computers will start to slow down too much below 0.004, so only change this if the simulation is moving too slowly. The radio buttons above the graph control which variable goes on which axis; you have the choice of Angle 1, Angle 2, Angular velocity 1 and Angular velocity 2 for each axis. faults Help! th2 = 0.0 om2 = 0.0 L1 = 1.0 L2 = 1.0 g = 9.81 h = 0.004	Passed

System maintenance

In this section I will show my code in full (with most comments removed) and explain what each part does so that if the program was returned to in the future or if someone else needed to alter it, they could find out how it worked in detail. Each paragraph is written above the relevant piece of code.

Imports: “sys” isn’t technically needed, but the PyQt documentation encourages the practice of passing command line arguments to the QApplication (see fifth last line). As my project has none, this doesn’t strictly matter. “pickle” is used to save the Pendulum’s attributes to file. “PyQt4” is obviously the framework/GUI library; I import the modules I need. Similarly I only import the mathematical functions I need from “math”.

```
import sys, pickle
from PyQt4 import QtGui, QtCore
from math import sin, cos, acos, pi, sqrt, floor
```

This function receives an angle in radians and returns one that is in the range of –pi to pi. If the received angle “th” is positive, it checks if it is less than pi. If so it can return it. If not, if checks to see if subtracting 2pi will get it into the right form. Although not strictly necessary, as most angles will only be just out of range, this saves having to call the function again recursively. If it is still too large, a recursive call is made. The checks for if th<0 are exactly the same but negative.

```
def cutAngle(th):
    if th > 0:
        if th < pi:
            return th
        elif th < 2*pi:
            return th-2*pi
        else:
            return cutAngle(((th/pi)%2)*pi)
    else:
        if th > -pi:
            return th
        elif th > -2*pi:
            return th + 2*pi
        else:
            return cutAngle(((th/pi)%2)*pi)
```

This function finds the next available file name inside the current folder that is in the form “nameXext” where “name” and “ext” are the arguments, and X is a positive integer. For example, getPictureName(“helloWorld”, “.py”) would check if “helloWorld0.py” was present in the current folder. If it isn’t, that is the first name available and so would be returned. If not, it would try “helloWorld1.py”, then “helloWorld2.py” etc. etc.

```
def getPictureName(name, ext):
        i = 0
        while True:
            try:
                open(name+str(i)+ext)
                i += 1
            except:
                break
        return name+str(i)+ext
```

This is the class that sends out the regular tick to the pendulum and the graph. The connection is made in DPWindow.initConnections. Whilst this could have been defined inside DPWindow, it doesn’t really qualify as being part of the interface and so for the sake of good practice has been given its own class

```
class Communicator(QtCore.QObject):
    def __init__(self):
        super().__init__()
        self.pendulumTick = QtCore.QTimer()
```

This is the double pendulum class. Note that DoublePendulum, GraphWidget and DPWindow are all based on QtGui.QWidget, the base class for all widgets in PyQt.

```
class DoublePendulum(QtGui.QWidget):
```

This is the custom defined signal that will send the pendulum’s attributes to the graph widget.

```
    attributeSignal = QtCore.pyqtSignal(float, float, float, float)
```

This is the function that initialises the double pendulum. It is given all of its starting attributes, which it sets using the initAttributes function. It also defines the attribute “canvas” that will serve as the bitmap that the pendulum’s trail will be drawn on. It sets it to be filled with black.

```
    def __init__(self, L1, L2, M1, M2, g, th1, th2, om1, om2, h):
        super().__init__()
        self.initAttributes(L1, L2, M1, M2, g, th1, th2, om1, om2,
h)
        self.setXY()
        self.calculateEnergy()
```

```
        self.setFixedSize(self.sideLength, self.sideLength)
        self.canvas = QtGui.QImage(self.sideLength, self.sideLength,
QtGui.QImage.Format_RGB32)
        self.canvas.fill(QtGui.QColor(0, 0, 0))
```

Sets all the necessary attributes using the arguments passed to __init__

```
    def initAttributes(self, L1, L2, M1, M2, g, th1, th2, om1, om2,
h):

        # constants
        self.L1  = L1
        self.L2  = L2
        self.M1  = M1
        self.M2  = M2
        self.g   = g
```

dScale (drawing scale) is the number that the mathematical coordinates need to be multiplied by to get the appropriate pixel coordinate within the widget - e.g if the point is (0.8, 0.2) mathematically, then the pixel coordinates will be (80, 20) if dScale is 100

```
        self.dScale = 100

        self.sideLength = self.dScale*2*(self.L1 + self.L2)

        self.x1  = 0.0
        self.x2  = 0.0
        self.y1  = 0.0
        self.y2  = 0.0

        self.th1 = th1
        self.th2 = th2
        self.om1 = om1
        self.om2 = om2

        # delta t
        self.h = h
```

setXY uses basic trigonometry to record each head's mathematical x/y position, using the angle they are inclined at (th1/th2) and their length. It then calculates their screen position. Firstly, it has to take into account that the origin of the pendulum is in the centre. To do this, it adjusts the coordinate values by the sum of the two lengths, this will shift the x coordinate origin to the left of the widget, and the y coordinate to the top (as this is how pixel coordinates are plotted). It also has to take into account that the mathematical y-coordinates were measured with positive upwards rather than downwards. The same is done for the second head, but also taking into account that its position is affected by the first head's position as well.

```
    def setXY(self):
```

```
        self.x1 = self.L1*sin(self.th1)
        self.y1 = -self.L1*cos(self.th1)

        self.x2 = self.x1 + self.L2*sin(self.th2)
        self.y2 = self.y1 - self.L2*cos(self.th2)

        shift = self.L1 + self.L2

        self.x1sh = (self.x1+shift)*self.dScale
        self.y1sh = -(self.y1-shift)*self.dScale

        self.x2sh = (self.x2+shift)*self.dScale
        self.y2sh = -(self.y2-shift)*self.dScale
```

calculateEnergy and omegaMax are the implementation of the energy calculations described in the "Identification of processes and suitable algorithms for data transformation" section of the Design. They are effectively just the formulae with the pendulum's attributes substituted in. The "num" parameter of omegaMax dictates whether it is calculating om1Max or om2Max – the algorithm can be used for both if a small change is made.

```
    def calculateEnergy(self):

        self.ke = 0.5*(self.M1*(self.L1**2)*(self.om1**2) +
self.M2*((self.L1**2)*(self.om1**2) + (self.L2**2)*(self.om2**2) +
2*self.L1*self.L2*self.om1*self.om2*cos(self.th1-self.th2)))
        shift = self.L1 + self.L2
        self.gpe = self.g*(self.M1*(self.y1+shift) +
self.M2*(self.y2+shift))
        self.energy = self.ke + self.gpe
        self.gpeMin = self.g*self.L2*self.M1
        self.om1Max = self.omegaMax(1)
        self.om2Max = self.omegaMax(2)

        if self.energy >=
(self.g*(2*self.L1*(self.M1+self.M2)+self.M1*self.L2)):
            self.th1Max = pi
        else:
            self.th1Max = acos(round((-
self.energy+self.g*(shift*(self.M1 +self.M2)-
self.M2*self.L2))/(self.g*self.L1*(self.M1 + self.M2)), 14))

        if self.energy >= (self.L2*self.g*(self.M1 + 2*self.M2)):
            self.th2Max = pi
        else:
            self.th2Max = acos(round((-
self.energy+self.g*(self.M1+self.M2)*(shift-
self.L1))/(self.M2*self.g*self.L2), 14))

    def omegaMax(self, num):

        A = 0.5*(self.L1**2)*(self.M1 + self.M2)
        B = 0.5*self.M2*(self.L2**2)
```

```
        C = self.L1*self.L2*self.M2
        D = self.energy - self.gpeMin

        if num == 2:
            temp = A
            A = B
            B = temp

        F = C**2 - 4*A*B
        G = 4*A*D
```

Note – this is the value for om2 that will give the maximum value for om1

```
        om2 = -sqrt(round((G*(C**2))/((F**2)-F*(C**2)), 13))
        omMax = (-C*om2+sqrt(round((C**2)*(om2**2)-
(4*A*B*(om2**2))+4*A*D, 13)))/(2*A)
        return omMax
```

om1d, om2d, th1d and th2d are the differential equations that are used with the Runge-Kutta technique. The "d" is just being used to signify "dash" – in the derivation earlier; I used a dash to signify that the variable was the first derivative with respect to time. Thus "om1d" means that it is the function used to calculate the first derivative of omega 1 with respect to time. The parameter varList is a list of the necessary attributes; its contents are th1, th2, om1 and om2 in that order. om1d and om2d are just implementations of the formulae derived in the "Identification of processes and suitable algorithms for data transformation" section of the Design. The first derivative of angle with respect to time is angular velocity, meaning that the function th1d and th2d simply return the value for om1 and om2 respectively from varList.

```
    def om1d(self, varList):
        th1 = varList[0]
        th2 = varList[1]
        om1 = varList[2]
        om2 = varList[3]
        m1 = self.M1
        m2 = self.M2
        L1 = self.L1
        L2 = self.L2
        g = self.g
        return (-g*(2*m1 + m2)*sin(th1) - m2*g*sin(th1-2*th2)-
2*sin(th1-th2)*m2*((om2**2)*L2 + (om1**2)*L1*cos(th1-
th2)))/(L1*(2*m1 + m2 - m2*cos(2*th1-2*th2)))

    def om2d(self, varList):
        th1 = varList[0]
        th2 = varList[1]
        om1 = varList[2]
        om2 = varList[3]
        m1 = self.M1
```

```
        m2 = self.M2
        L1 = self.L1
        L2 = self.L2
        g = self.g
        return (2*sin(th1-
th2)*((om1**2)*L1*(m1+m2)+g*(m1+m2)*cos(th1)+(om2**2)*L2*m2*cos(th1-
th2)))/(L2*(2*m1+m2-m2*cos(2*th1 - 2*th2)))

    def th1d(self, varList):
        return varList[2]

    def th2d(self, varList):
        return varList[3]
```

This is the implementation of the Runge Kutta algorithm described in "Identification of processes and suitable algorithms for data transformation" section of the Design. The "varList" parameter is the same as it was described above. The "funcList" is a list where each indexed function is the derivative of the corresponding index in varList; i.e. varList th1, th2, om1, om2, so funcList is th1d, th2d, om1d, om2d. aList, bList cList dList are lists corresponding to the vectors labelled with letters from the algorithm – i.e. the vector **a** in the algorithm will be" aList" in here. "multiFuncVect" is a function that takes the list of variables, the list of functions, the previous lettered list (as required by the algorithm) and the time step for that calculation (h or h/2 depending on which one (again, as required by the algorithm.))

```
    def rungeKutta(self, varList, funcList):
        newVarList = []
        h = self.h
        aList = []

        for func in funcList:
            aList.append(func(varList))

        bList = self.multiFuncVect(varList, funcList, aList, h/2)
        cList = self.multiFuncVect(varList, funcList, bList, h/2)
        dList = self.multiFuncVect(varList, funcList, cList, h)

        for var, a, b, c, d in zip(varList, aList, bList, cList,
dList):
            newVarList.append(var + (h/6)*(a + 2*b + 2*c + d))

        return newVarList


    def multiFuncVect(self, varList, funcList, letList, h):
        # Part of the Runge Kutta algorithm. See documentation
        newLetList = []
        tempVarList = []

        for var, let in zip(varList, letList):
            tempVarList.append(var + h*let)

        for func in funcList:
            newLetList.append(func(tempVarList))
```

```
        return newLetList
```

This is the function that takes the steps need for the pendulum to advance by one tick.

```
    def advance(self):

        varList = [self.th1, self.th2, self.om1, self.om2]
        funcList = [self.th1d, self.th2d, self.om1d, self.om2d]

        varList = self.rungeKutta(varList, funcList)

        self.th1, self.th2, self.om1, self.om2 = varList
        self.th1 = cutAngle(self.th1)
        self.th2 = cutAngle(self.th2)
        self.setXY()
```

This is the function that is called when the pendulum is drawn to the screen.

```
    def paintEvent(self, event):
        headWidth = 16
        headRadius = headWidth // 2
        s = self.dScale*(self.L1 + self.L2)/2
```

"qp" is the QPainter object that we will control to do the drawing

```
        qp = QtGui.QPainter()
        qp.begin(self)
```

Draw the second pendulum's point onto the trail image

```
        self.canvas.setPixel(self.x2sh, self.y2sh, QtGui.qRgb(255,
0, 0))
        qp.drawImage(0, 0, self.canvas)
```

Calculate the head colour depending on the mass i.e. more blue = more mass, more white = less mass

```
        head0colour = QtGui.QColor(255, 255, 255)
        colFrac = 1-self.M1/2
        head1colour = QtGui.QColor(255*colFrac, 255*colFrac, 255)
        colFrac = 1-self.M2/2
        head2colour = QtGui.QColor(255*colFrac, 255*colFrac, 255)
```

Calculate the arm colour depending on the angular velocity i.e more green = closer to max speed, more white = closer to still if-else check is to stop error being raised if pendulum starts vertically downwards

```
        if self.om1Max == 0:
            line1colour = QtGui.QColor(255, 255, 255)
        else:
```

```
            colFrac = 1 - abs(self.om1)/self.om1Max
            line1colour = QtGui.QColor(255*colFrac, 255,
255*colFrac)
        if self.om2Max == 0:
            line2colour = QtGui.QColor(255, 255, 255)
        else:
            colFrac = 1 - abs(self.om2)/self.om2Max
            line2colour = QtGui.QColor(255*colFrac, 255,
255*colFrac)
```

Draw the arms and heads

```
        qp.setBrush(head0colour)
        qp.setPen(head0colour)
        qp.drawEllipse(2*s - headRadius, 2*s - headRadius,
headWidth, headWidth)

        qp.setBrush(head1colour)
        qp.setPen(head1colour)
        qp.drawEllipse(self.x1sh - headRadius, self.y1sh -
headRadius, headWidth, headWidth)

        qp.setBrush(head2colour)
        qp.setPen(head2colour)
        qp.drawEllipse(self.x2sh - headRadius, self.y2sh -
headRadius, headWidth, headWidth)

        qp.setBrush(line1colour)
        qp.setPen(line1colour)
        qp.drawLine(2*s, 2*s, self.x1sh, self.y1sh)

        qp.setBrush(line2colour)
        qp.setPen(line2colour)
        qp.drawLine(self.x1sh, self.y1sh, self.x2sh, self.y2sh)
        qp.end()
```

Advance pendulum, send signal to graph, repaint

```
    def move(self):
        self.advance()
        self.attributeSignal.emit(self.th1, self.th2, self.om1,
self.om2)
        self.repaint()
```

This function is present for testing, it displays the pendulum's attribute then advances by one tick. As the final program isn't run with a console window, pressing step will simply move the simulation one tick forward

```
    def step(self):
        self.calculateEnergy()
        print('     x1:', self.x1)
```

```
        print('     y1:', self.y1)
        print('     x2:', self.x2)
        print('     y2:', self.y2)
        print('    th1:', self.th1)
        print('    th2:', self.th2)
        print('    om1:', self.om1)
        print('    om2:', self.om2)
        print('     ke:', self.ke)
        print('    gpe:', self.gpe)
        print('om1 max:', self.om1Max)
        print('om2 max:', self.om2Max)
        print('th1 max:', self.th1Max)
        print('th2 max:', self.th2Max)
        print(' gpeMin:', self.gpeMin)
        print(' energy:', self.energy)
        print()
        self.move()
```

Save image of pendulum to file

```
    def export(self):
        name = getPictureName('picture', '.bmp')
        self.canvas.save(name, None, 100)
```

Save the penudlum's current state to file

```
    def saveState(self):
        saveFile = open('Pendulum state save.pendulum', 'wb')
        attributes = (self.L1, self.L2, self.M1, self.M2, self.g,
self.th1, self.th2, self.om1, self.om2, self.h)
        pickle.dump(attributes, saveFile)
        saveFile.close()
```

This is the graph widget.

```
class GraphWidget(QtGui.QWidget):

    def __init__(self, xAxisVar, yAxisVar, xMinMax, yMinMax,
sideLength):

        super().__init__()
        self.setFixedSize(sideLength, sideLength)
```

This is the bitmap that will be drawn on

```
        self.canvas = QtGui.QImage(sideLength, sideLength,
QtGui.QImage.Format_RGB32)
        self.canvas.fill(QtGui.QColor(0, 0, 0))
```

Get attributes from __init__'s arguments

```
        self.xAxisVar = xAxisVar
```

```
        self.xVal = 0.0
        self.xValAdj = 25
        self.xMin = xMinMax[0]
        self.xMax = xMinMax[1]
        self.yAxisVar = yAxisVar
        self.yVal = 0.0
        self.yValAdj = 25
        self.yMin = yMinMax[0]
        self.yMax = yMinMax[1]
        self.sideLength = sideLength
```

Drawing scale (same idea as in the pendulum)

```
        if (self.xMax-self.xMin) == 0:
            self.xDScale = 0
        else:
            self.xDScale = (sideLength-50)/(self.xMax-self.xMin)

        if (self.yMax-self.yMin) == 0:
            self.yDScale = 0
        else:
            self.yDScale = (sideLength-50)/(self.yMax-self.yMin)



    def paintEvent(self, event):

        qp = QtGui.QPainter()
        qp.begin(self)
```

Draw points

```
        qp.drawImage(0, 0, self.canvas)
```

Draw axis

```
        qp.setPen(QtGui.QColor(255, 255, 255))
        topLeft = QtCore.QPoint(25, 25)
        bottomLeft = QtCore.QPoint(25, self.sideLength-25)
        bottomRight = QtCore.QPoint(self.sideLength-25,
self.sideLength-25)
        qp.drawLine(topLeft, bottomLeft)
        qp.drawLine(bottomLeft, bottomRight)
```

Draw markings

Write number markings

x markings

```
        for i in range(floor(self.xMin)+1, floor(self.xMax)+1):
            x = self.adjustX(i)
```

```
            qp.drawLine(x, self.sideLength - 25, x, self.sideLength
- 20)
            qr = qp.fontMetrics().boundingRect(str(i))
            qp.drawText(x-(qr.width()/2), self.sideLength - 6,
str(i))
```

y markings

```
        for j in range(floor(self.yMin)+1, floor(self.yMax)+1):
            y = self.adjustY(j)
            qp.drawLine(25, y, 20, y)
            qr = qp.fontMetrics().boundingRect(str(j))
            qp.drawText((20-qr.width())/2, y+qr.height()/3, str(j))
```

Label axis

```
        qr = qp.fontMetrics().boundingRect(self.xAxisVar)
        qp.drawText(self.sideLength-22, self.sideLength-25 +
qr.height()/3, self.xAxisVar)
        qr = qp.fontMetrics().boundingRect(self.yAxisVar)
        qp.drawText(25-qr.width()/2, 23, self.yAxisVar)
```

Draw head

```
        qp.drawRect(self.xValAdj-1, self.yValAdj-4, 3, 9)
        qp.drawRect(self.xValAdj-4, self.yValAdj-1, 9, 3)

        qp.end()

    def adjustPoints(self):
```

The two functions called here adjust the points so that they lie inside the axis and peform the necessary 'pixel shift' performed earlier

```
        self.xValAdj = self.adjustX(self.xVal)
        self.yValAdj = self.adjustY(self.yVal)

    def adjustX(self, x):
        return (x-self.xMin)*self.xDScale + 25

    def adjustY(self, y):
        return self.sideLength - ((y-self.yMin)*self.yDScale + 25)

    def step(self, th1, th2, om1, om2):
```

This function takes the values from the graph, selects the ones it needs depending on what is on each axis, draws the point onto the bitmap then repaints the graph

```
        if self.xAxisVar == 'th1':
            self.xVal = th1
        if self.xAxisVar == 'th2':
            self.xVal = th2
        if self.xAxisVar == 'om1':
            self.xVal = om1
        if self.xAxisVar == 'om2':
            self.xVal = om2
```

```
        if self.yAxisVar == 'th1':
            self.yVal = th1
        if self.yAxisVar == 'th2':
            self.yVal = th2
        if self.yAxisVar == 'om1':
            self.yVal = om1
        if self.yAxisVar == 'om2':
            self.yVal = om2

        self.adjustPoints()
        self.canvas.setPixel(self.xValAdj, self.yValAdj,
QtGui.qRgb(255, 0, 0))
        self.repaint()

    def export(self):
```
Save the graph image to file
```
        name = getPictureName('graph', '.bmp')
        self.canvas.save(name, None, 100)


class DpWindow(QtGui.QWidget):
```
This is the window that will contain all the widgets
```

    def __init__(self):
        super().__init__()
        self.setInitialPendulumAttributes()
        self.initPendulum()
        self.initComms()
        self.setInitialGraphAttributes()
        self.initGraph()
        self.initUI()
        self.initConnections()

        self.move(100, 100)
        self.show()

    def setInitialPendulumAttributes(self, iL1 = 1, iL2 = 1, iM1 =
1, iM2 = 1, ig = 9.81, ith1 = 0, ith2 = 0, iom1 = 0, iom2 = 0, ih =
0.004):
```
If this is called without arguments, it will set default values. Otherwise, it can be used to set all of a pendulum's initial attributes.

```
        self.iL1 = iL1
        self.iL2 = iL2
        self.iM1 = iM1
        self.iM2 = iM2
        self.ig = ig
        self.ith1 = ith1
        self.ith2 = ith2
        self.iom1 = iom1
        self.iom2 = iom2
        self.ih = ih
```

```
    def setInitialGraphAttributes(self):
```
Default variables for the x and y axes

```
        self.ixVar = 'om1'
        self.iyVar = 'om2'
        self.setGraphMinMax()
```

```
    def setGraphMinMax(self):
```
Create a dictionary that stores the min and max variables with the name of the variable on the axis as the key. This is simply so that a long if-elif-else clause is not needed

```
        minMaxDict = {'om1':(-self.pendulum.om1Max,
self.pendulum.om1Max),
                      'om2':(-self.pendulum.om2Max,
self.pendulum.om2Max),
                      'th1':(-self.pendulum.th1Max,
self.pendulum.th1Max),
                      'th2':(-self.pendulum.th2Max,
self.pendulum.th2Max)}
        self.ixMinMax = minMaxDict[self.ixVar]
        self.iyMinMax = minMaxDict[self.iyVar]
        self.isideLength = self.pendulum.sideLength
```

```
    def initPendulum(self):
```
Create the pendulum object (widget)

```
        self.pendulum = DoublePendulum(self.iL1, self.iL2, self.iM1,
self.iM2, self.ig, self.ith1, self.ith2, self.iom1, self.iom2,
self.ih)
```

```
    def initComms(self):
```
Create the communicator object

```
        self.comms = Communicator()
```

```
    def initGraph(self):
```
Ccreate the graph object (widget)

```
        self.graph = GraphWidget(self.ixVar, self.iyVar,
self.ixMinMax, self.iyMinMax, self.isideLength)
```

```
    def resetWidgets(self):
```
All the steps needed to re-initialise the widgets if their starting attributes have been changed

```
        self.removeConnections()
        self.pendulum.deleteLater()
        self.graph.deleteLater()
        self.initPendulum()
        self.initComms()
        self.setGraphMinMax()
        self.initGraph()
        self.initConnections()
        self.addGraphToGrid()
        self.addPendulumToGrid()
```

All of the set (x) from slider functions below follow a similar pattern. They multiply the value of the slider by their maximum value divided by the slider's maximum. This effectively scales their value as the slider scales. Some are slightly different to take into accuont things like negatives, or non zero minimums, however the basic idea stays the same

```
def setTh1FromSlider(self, val):
    angle = (2*pi/100)*val
    self.ith1 = angle
    self.th1Label.setText('th1 = ' + str(round(self.ith1, 3)))
    self.resetWidgets()

def setTh2FromSlider(self, val):
    angle = (2*pi/100)*val
    self.ith2 = angle
    self.th2Label.setText('th2 = ' + str(round(self.ith2, 3)))
    self.resetWidgets()

def setOm1FromSlider(self, val):
    self.iom1 = (3/50)*val
    self.om1Label.setText('om1 = ' + str(round(self.iom1, 3)))
    self.resetWidgets()

def setOm2FromSlider(self, val):
    self.iom2 = (3/50)*val
    self.om2Label.setText('om2 = ' + str(round(self.iom2, 3)))
    self.resetWidgets()

def setGfromSlider(self, val):
    self.ig = (19.62/100)*val
    self.gLabel.setText('g = ' + str(round(self.ig, 3)))
    self.resetWidgets()

def setHfromSlider(self, val):
    self.ih = 0.001 * val
    self.hLabel.setText('h = ' + str(round(self.ih, 5)))
    self.resetWidgets()
```

These functions that set ratios are similar to the ones above, except they take steps to ensure that the total mass or length is always 2

```
def setLengthRatioFromSlider(self, ratio):
    """ratio is the fraction of length to be given to L1"""
    ratio /= 100
    self.iL1 = 2*ratio
    self.L1Label.setText('L1 = ' + str(round(self.iL1, 3)))
    self.iL2 = 2*(1-ratio)
    self.L2Label.setText('L2 = ' + str(round(self.iL2, 3)))
    self.resetWidgets()
```

```
    def setMassRatioFromSlider(self, ratio):
        """ratio is the fraction of length to be given to M1"""
        ratio /= 100
        self.iM1 = 2*ratio
        self.M1Label.setText('M1 = ' + str(round(self.iM1, 3)))
        self.iM2 = 2*(1-ratio)
        self.M2Label.setText('M2 = ' + str(round(self.iM2, 3)))
        self.resetWidgets()
```

Setting the graph's axis usign the radio button's group id

```
    def setGraphYVar(self, yvarID):
        if yvarID == 1:
            self.iyVar = 'om1'
        if yvarID == 2:
            self.iyVar = 'om2'
        if yvarID == 3:
            self.iyVar = 'th1'
        if yvarID == 4:
            self.iyVar = 'th2'
        self.resetWidgets()

    def setGraphXVar(self, xvarID):
        if xvarID == 1:
            self.ixVar = 'om1'
        if xvarID == 2:
            self.ixVar = 'om2'
        if xvarID == 3:
            self.ixVar = 'th1'
        if xvarID == 4:
            self.ixVar = 'th2'
        self.resetWidgets()
```

Load the pendulum save state from file, resets the widgets

```
    def pendulumLoadState(self):
        saveFile = open('Pendulum state save.pendulum', 'rb')
        attributes = pickle.load(saveFile)
        saveFile.close()
        self.setInitialPendulumAttributes(*attributes)
        self.resetWidgets()
```

Reset pendulum and graph to the values they have when the program starts

```
    def setDefaults(self):
        self.setInitialPendulumAttributes()
        self.resetSlidersRBs()
        self.setInitialGraphAttributes()
        self.resetSlidersRBs()
        self.resetWidgets()

    def showHelp(self):
```

Opens a new window that has helpText in it

```
        self.helpText = """
HELP
----

The program is made up of three main sections.


Firstly, there is the double pendulum itself. This is found in the
black box in the bottom right of the screen.

The central circle is the fixed pivot; this won't move. You can
think of it as being attached to a wall behind the pendulum. The two
lines are the two arms of the pendulum, these have no mass and are
free to move.
The arms will be white when they are still, and green when they are
turning as fast as they can. They will be a shade of green for any
value in-between maximum and 0.

The other two circles are the masses of the double pendulum. These
will be a shade of blue depending on how much mass they have, where
pale is lighter and dark means heavier. These also act as pivots for
the
pendulum to move, similar to the central fixed pivot.


Next there is the graph. This is the black box in the bottom left of
the window.

This dynamically draws a graph of the pendulum's current conditions.
You can set which variable goes on which axis, that will be covered
in the controls section. The axis will be labelled and marked so you
can see
the different values for each variable.


Lastly there is the control panel.

Along the top there is a row of buttons.
The first one is 'Start'. This starts the simulation, which will
start both the pendulum and the graph at once.
Next there is the 'Stop' button, which pauses the simulation.
The 'Step' button steps the simulation forward one tick.
The 'Save state' button will save the pendulum's current conditions
to a file (which will be called 'Pendulum state save.pendulum') in
the same directory as the program.
The 'Load state' button allows you to load this file, and resume the
simulation from the state it was when you saved.
The 'Save pic' button will save a bitmap file of the current graph
and pendulum trail in the same directory as the program. If there is
already a file there, then the new one will be numbered. So the
graph's images
will be saved as 'graph0.bmp', 'graph1.bmp', 'graph2.bmp' etc. and
the pendulum's images will be saved as 'picture0.bmp',
'picture1.bmp' etc.
The 'Reset' button will reset the pendulum to the current initial
conditions, that is the conditions that you have set or loaded.
```

The 'Defaults' button will reset the pendulum to the default starting conditions.
The 'Help!' button will show you this page, but of course you already know that!

The top two sliders control the starting angles. 'th1' (Meaning Theta 1, because theta is the Greek letter used in maths to represent angles, and 1 is because it is for the first arm) is the starting angle for the
first arm. If the slider is far left, it the arm will be vertically downwards. If you push the slider right, it will turn through a full circle, ending vertically downwards again. The same applies for 'th2'.
The next two sliders control the starting angular velocities. These are how quickly the pendulum is already turning at the start. Most of the time these will be left as 0. 'om1' (Meaning Omega 1, because omega is
the Greek letter used in maths to represent angular velocity, and 1 because it is the angular velocity of the first arm) is the starting angular velocity for the first arm. It starts in the middle at 0. When it is
far left, the arm will begin the simulation rotating anticlockwise (left). When it is far right, the arm will begin the simulation rotating clockwise (right). Note the shade of green of each arm. The same applies
for 'om2'.
The third slider down on the left represents the ratio of the mass between the heads. If it is slid to the left, the second head will be heavier and the first head lighter (observe the shades of blue), and if it is
slid to the right, the first head will be heavier and the second head lighter.
The third slider down on the right represents the ratio of the length between the heads. If it is slid to the left, the second arm will be longer and the first arm shorter. If it is slid to the right, the first arm
will be longer and the second shorter.
The fourth slider down on the right sets the gravitational field strength. On earth this is approximately 9.81, so that is the starting value. If it is slid to the right, the value increases, meaning everything is
heavier. As it is slid to the left, it becomes weaker, meaning the pendulum acts lighter and lighter. If it is all the way to the left, it is 0, meaning nothing is pulling the pendulum downwards at all.
The bottom right slider controls the 'time step'. This is to do with how the simulation works. The default value is 'h = 0.004', which means that the simulation will work out where to move the pendulum to every
0.004 seconds. This can be thought of as the accuracy of simulation; a high value will result in a very inaccurate (but easily processed) simulation, whereas a very low value will result in a very accurate (but
CPU intensive) simulation. Most computers will start to slow down too much below 0.004, so only change this if the simulation is moving too slowly.

```
The radio buttons above the graph control which variable goes on
which axis; you have the choice of Angle 1, Angle 2, Angular
velocity 1 and Angular velocity 2 for each axis."""

        self.helpWindow = QtGui.QWidget()
        self.helpWindow.setWindowTitle("Help")
        self.helpWindow.setWindowIcon(QtGui.QIcon("helpIcon.png"))
        self.helpLabel = QtGui.QLabel(self.helpText)
        self.helpLabel.setParent(self.helpWindow)
        self.helpWindow.setFixedSize(self.helpLabel.sizeHint())
        self.helpWindow.show()

    def resetSlidersRBs(self):
```

Sets all sliders and radio buttons to their default values

```
        self.th1Slider.setValue(0)
        self.th2Slider.setValue(0)
        self.om1Slider.setValue(0)
        self.om2Slider.setValue(0)
        self.massSlider.setValue(50)
        self.lengthSlider.setValue(50)
        self.gSlider.setValue(50)
        self.hSlider.setValue(4)
        self.om1RBX.click()
        self.om2RBY.click()

    def initUI(self):
```

This initialises the UI. It creates, labels and resizes all the buttons, creates and labels all the sliders, creates all the radio buttons, then puts all of them into layouts that will arrange their position on the screen

```
        self.setFixedSize(self.sizeHint())
        self.setWindowTitle("Double Pendulum Simulation")
        self.setWindowIcon(QtGui.QIcon("icon.png"))

        # Buttons
        self.startButton = QtGui.QPushButton('Start')
        self.startButton.setFixedSize(50, 50)
        self.stopButton  = QtGui.QPushButton('Stop')
        self.stopButton.setFixedSize(50, 50)
        self.stepButton = QtGui.QPushButton('Step')
        self.stepButton.setFixedSize(50, 50)
        self.stateSaveButton = QtGui.QPushButton('Save\nstate')
        self.stateSaveButton.setFixedSize(50, 50)
        self.stateLoadButton = QtGui.QPushButton('Load\nstate')
        self.stateLoadButton.setFixedSize(50, 50)
        self.pictureSaveButton = QtGui.QPushButton('Save\nimage')
        self.pictureSaveButton.setFixedSize(50, 50)
        self.resetButton = QtGui.QPushButton('Reset')
        self.resetButton.setFixedSize(50, 50)
        self.defaultsButton = QtGui.QPushButton('Defaults')
        self.defaultsButton.setFixedSize(50, 50)
        self.helpButton = QtGui.QPushButton('Help!')
        self.helpButton.setFixedSize(50, 50)
        self.helpButton.setStyleSheet('QPushButton {color : red}')
```

```
        # Sliders
        self.th1Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.th1Slider.setRange(0, 100)
        self.th2Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.th2Slider.setRange(0, 100)
        self.om1Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.om1Slider.setRange(-50, 50)
        self.om2Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.om2Slider.setRange(-50, 50)
        self.massSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.massSlider.setRange(30, 70)
        self.massSlider.setValue(50)
        self.lengthSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.lengthSlider.setRange(30, 70)
        self.lengthSlider.setValue(50)
        self.gSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.gSlider.setRange(0, 100)
        self.gSlider.setValue(50)
        self.hSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.hSlider.setRange(1, 100)
        self.hSlider.setValue(4)

        # Labels
        self.th1Label = QtGui.QLabel('th1 = ' +
str(round(self.ith1,3)))
        self.th2Label = QtGui.QLabel('th2 = ' +
str(round(self.ith2,3)))
        self.om1Label = QtGui.QLabel('om1 = ' +
str(round(self.iom1,3)))
        self.om2Label = QtGui.QLabel('om2 = ' +
str(round(self.iom2,3)))
        self.M1Label = QtGui.QLabel('M1 = ' +
str(round(self.iM1,3)))
        self.M2Label = QtGui.QLabel('M2 = ' +
str(round(self.iM2,3)))
        self.L1Label = QtGui.QLabel('L1 = ' +
str(round(self.iL1,3)))
        self.L2Label = QtGui.QLabel('L2 = ' +
str(round(self.iL2,3)))
        self.gLabel = QtGui.QLabel('g = ' + str(round(self.ig,3)))
        self.hLabel = QtGui.QLabel('h = ' + str(round(self.ih,5)))

        #   Radio Buttons
        self.RBXGroup = QtGui.QButtonGroup()
        self.RBYGroup = QtGui.QButtonGroup()

        # X axis radio buttons
        self.om1RBX = QtGui.QRadioButton('om1')
        self.RBXGroup.addButton(self.om1RBX, 1)
        self.om2RBX = QtGui.QRadioButton('om2')
        self.RBXGroup.addButton(self.om2RBX, 2)
        self.th1RBX = QtGui.QRadioButton('th1')
        self.RBXGroup.addButton(self.th1RBX, 3)
        self.th2RBX = QtGui.QRadioButton('th2')
        self.RBXGroup.addButton(self.th2RBX, 4)
```

```
# X axis radio buttons
self.om1RBY = QtGui.QRadioButton('om1')
self.RBYGroup.addButton(self.om1RBY, 1)
self.om2RBY = QtGui.QRadioButton('om2')
self.RBYGroup.addButton(self.om2RBY, 2)
self.th1RBY = QtGui.QRadioButton('th1')
self.RBYGroup.addButton(self.th1RBY, 3)
self.th2RBY = QtGui.QRadioButton('th2')
self.RBYGroup.addButton(self.th2RBY, 4)

self.om1RBX.toggle()
self.om2RBY.toggle()


# Radio button boxes
self.RBXbox = QtGui.QHBoxLayout()
self.RBXbox.addWidget(QtGui.QLabel("X axis:"))
self.RBXbox.addWidget(self.om1RBX)
self.RBXbox.addWidget(self.om2RBX)
self.RBXbox.addWidget(self.th1RBX)
self.RBXbox.addWidget(self.th2RBX)

self.RBYbox = QtGui.QHBoxLayout()
self.RBYbox.addWidget(QtGui.QLabel("Y axis:"))
self.RBYbox.addWidget(self.om1RBY)
self.RBYbox.addWidget(self.om2RBY)
self.RBYbox.addWidget(self.th1RBY)
self.RBYbox.addWidget(self.th2RBY)


#   Slider grid
# Th sliders
self.sliderRBGrid = QtGui.QGridLayout()
self.sliderRBGrid.addWidget(self.th1Slider, 0, 0)
self.sliderRBGrid.addWidget(self.th1Label, 0, 1)
self.sliderRBGrid.addWidget(self.th2Slider, 0, 3)
self.sliderRBGrid.addWidget(self.th2Label, 0, 4)

# Om sliders
self.sliderRBGrid.addWidget(self.om1Slider, 1, 0)
self.sliderRBGrid.addWidget(self.om1Label, 1, 1)
self.sliderRBGrid.addWidget(self.om2Slider, 1, 3)
self.sliderRBGrid.addWidget(self.om2Label, 1, 4)

# M/L sliders
self.sliderRBGrid.addWidget(self.massSlider, 2, 0)
self.sliderRBGrid.addWidget(self.M1Label, 2, 1)
self.sliderRBGrid.addWidget(self.M2Label, 2, 2)
self.sliderRBGrid.addWidget(self.lengthSlider, 2, 3)
self.sliderRBGrid.addWidget(self.L1Label, 2, 4)
self.sliderRBGrid.addWidget(self.L2Label, 2, 5)

# g/h sliders
self.sliderRBGrid.addWidget(self.gSlider, 3, 3)
self.sliderRBGrid.addWidget(self.gLabel, 3, 4)
```

```
        self.sliderRBGrid.addWidget(self.hSlider, 4, 3)
        self.sliderRBGrid.addWidget(self.hLabel, 4, 4)

        # Radio buttons
        self.sliderRBGrid.addLayout(self.RBXbox, 3, 0)
        self.sliderRBGrid.addLayout(self.RBYbox, 4, 0)

        # Button box
        self.buttonBox = QtGui.QHBoxLayout()
        self.buttonBox.addWidget(self.startButton)
        self.buttonBox.addWidget(self.stopButton)
        self.buttonBox.addWidget(self.stepButton)
        self.buttonBox.addWidget(self.stateSaveButton)
        self.buttonBox.addWidget(self.stateLoadButton)
        self.buttonBox.addWidget(self.pictureSaveButton)
        self.buttonBox.addWidget(self.resetButton)
        self.buttonBox.addWidget(self.defaultsButton)
        self.buttonBox.addWidget(self.helpButton)


        # Pendulum and graph grid
        self.penGraGrid = QtGui.QGridLayout()

        # Set layouts (overall grid)
        self.grid = QtGui.QGridLayout()
        self.setLayout(self.grid)
        self.grid.addLayout(self.buttonBox, 0, 0)
        self.grid.addLayout(self.sliderRBGrid, 1, 0)
        self.addGraphToGrid()
        self.addPendulumToGrid()
        self.initActions()

    def addGraphToGrid(self):
        self.penGraGrid.addWidget(self.graph, 0, 0)
        self.grid.addLayout(self.penGraGrid, 3, 0)

    def addPendulumToGrid(self):
        self.penGraGrid.addWidget(self.pendulum, 0, 1)
        self.grid.addLayout(self.penGraGrid, 3, 0)

    def initConnections(self):
```

This function connects all the signals to the slots that need them. All the buttons are connected to their desired functions when clicked, all the sliders are connected to the desired functions when they are chagned.

```
        self.comms.pendulumTick.setInterval(self.pendulum.h*1000)

        self.comms.pendulumTick.timeout.connect(self.pendulum.move)
        self.pendulum.attributeSignal.connect(self.graph.step)

        self.th1Slider.valueChanged.connect(self.setTh1FromSlider)
        self.th2Slider.valueChanged.connect(self.setTh2FromSlider)
        self.om1Slider.valueChanged.connect(self.setOm1FromSlider)
        self.om2Slider.valueChanged.connect(self.setOm2FromSlider)
```

```
self.massSlider.valueChanged.connect(self.setMassRatioFromSlider)

self.lengthSlider.valueChanged.connect(self.setLengthRatioFromSlider
)
        self.gSlider.valueChanged.connect(self.setGfromSlider)
        self.hSlider.valueChanged.connect(self.setHfromSlider)

        self.RBXGroup.buttonClicked[int].connect(self.setGraphXVar)
        self.RBYGroup.buttonClicked[int].connect(self.setGraphYVar)


self.startButton.clicked.connect(self.comms.pendulumTick.start)

self.stopButton.clicked.connect(self.comms.pendulumTick.stop)
        self.stepButton.clicked.connect(self.pendulum.step)

self.stateSaveButton.clicked.connect(self.pendulum.saveState)
        self.stateLoadButton.clicked.connect(self.pendulumLoadState)
        self.pictureSaveButton.clicked.connect(self.graph.export)
        self.pictureSaveButton.clicked.connect(self.pendulum.export)
        self.resetButton.clicked.connect(self.resetWidgets)
        self.defaultsButton.clicked.connect(self.setDefaults)
        self.helpButton.clicked.connect(self.showHelp)


def removeConnections(self):
```

This disconnects all the connections made in initConnections, so that when the widgits are recreated, there aren't stray connections causing unwanted results.

```
        self.pendulum.attributeSignal.disconnect(self.graph.step)

self.comms.pendulumTick.timeout.disconnect(self.pendulum.move)


self.th1Slider.valueChanged.disconnect(self.setTh1FromSlider)

self.th2Slider.valueChanged.disconnect(self.setTh2FromSlider)

self.om1Slider.valueChanged.disconnect(self.setOm1FromSlider)

self.om2Slider.valueChanged.disconnect(self.setOm2FromSlider)

self.massSlider.valueChanged.disconnect(self.setMassRatioFromSlider)

self.lengthSlider.valueChanged.disconnect(self.setLengthRatioFromSli
der)
        self.gSlider.valueChanged.disconnect(self.setGfromSlider)
        self.hSlider.valueChanged.disconnect(self.setHfromSlider)


self.RBXGroup.buttonClicked[int].disconnect(self.setGraphXVar)

self.RBYGroup.buttonClicked[int].disconnect(self.setGraphYVar)
```

```
self.startButton.clicked.disconnect(self.comms.pendulumTick.start)

self.stopButton.clicked.disconnect(self.comms.pendulumTick.stop)
        self.stepButton.clicked.disconnect(self.pendulum.step)

self.stateSaveButton.clicked.disconnect(self.pendulum.saveState)

self.stateLoadButton.clicked.disconnect(self.pendulumLoadState)
        self.pictureSaveButton.clicked.disconnect(self.graph.export)

self.pictureSaveButton.clicked.disconnect(self.pendulum.export)
        self.resetButton.clicked.disconnect(self.resetWidgets)
        self.defaultsButton.clicked.disconnect(self.setDefaults)
        self.helpButton.clicked.disconnect(self.showHelp)
```

Simply adds a quit short cut (Ctrl Q)

```
    def initActions(self):
        self.exitAction = QtGui.QAction('&Exit', self)
        self.exitAction.setShortcut('Ctrl+Q')
        self.exitAction.triggered.connect(QtGui.qApp.quit)
        self.addAction(self.exitAction)
```

This creates the application object, creates the main window, then executes the application

```
def main():
    app = QtGui.QApplication(sys.argv)
    win = DpWindow()
    app.exec_()
```

This checks to make sure the program has been started rather than imported

```
if __name__ == "__main__":
    main()
```

User Manual

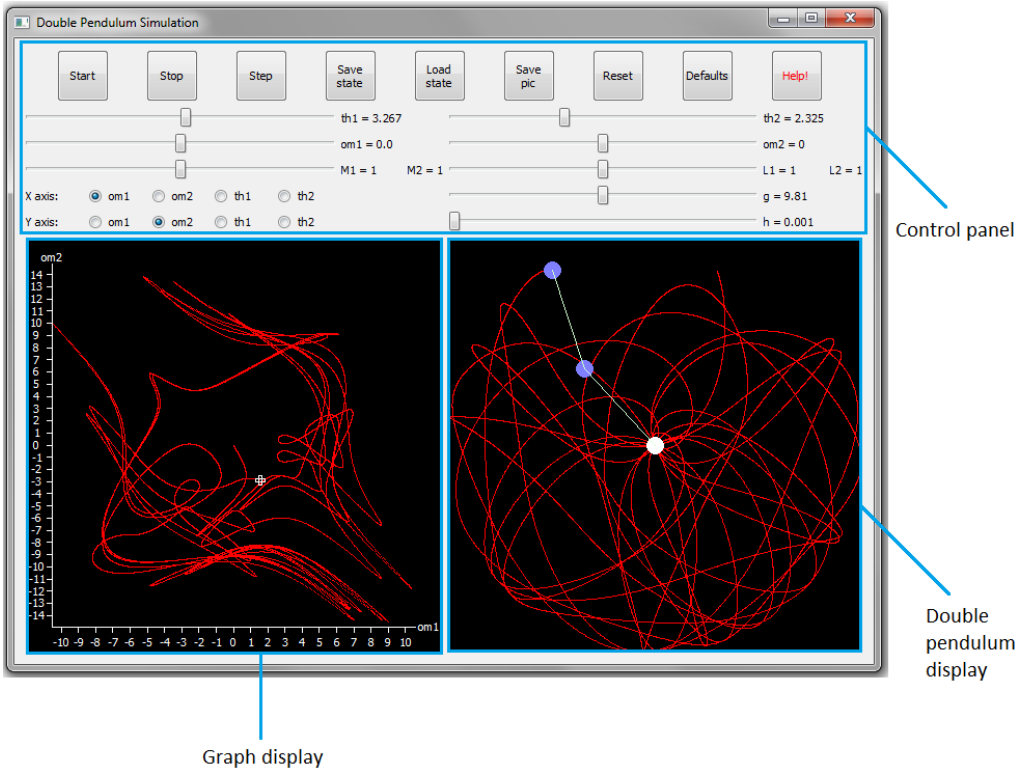
Contents

Installation instructions

In order to run the simulation, your computer must have Python 3.3.x installed, and it must also have PyQt 4.x.x for Python 3.3 installed. All three of these are cross platform, so choose whichever is appropriate for your system from the Downloads page. Python can be downloaded from http://goo.gl/TYPhzI, and PyQt can be downloaded from http://goo.gl/LgVh2. The installations are both self-explanatory, and there are help documents on the websites if you encounter any issues.

Directions for use

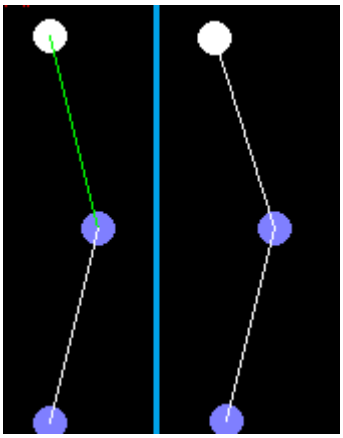
The program is split into three main sections (see image).



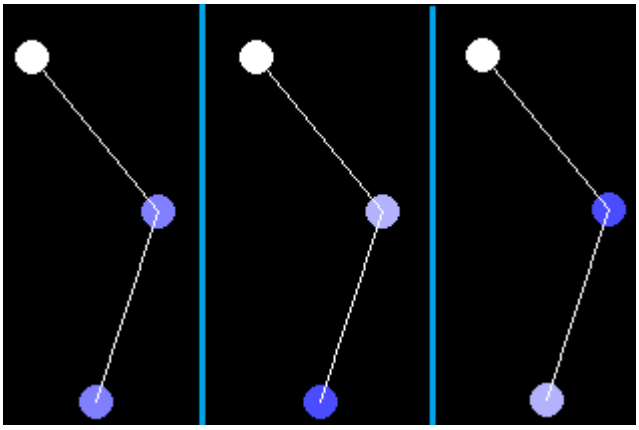
The double pendulum display

This is located at the bottom right of the screen. This is where the pendulum itself will be displayed, and also the traced path of the second head will be drawn. The

traced path is drawn in red, and the background colour is black. The central circle is a fixed pivot that remains in the same place throughout the simulation. There is a rod connected to the central pivot, this is massless. At the end of the first rod is a mass, shown as a circle. Similarly, connected to the first mass is a second (massless) rod, and on the end of the second rod is a second mass. The rods will change colour depending on their angular velocity. When they are stationary, they are white, and when they are moving at maximum angular velocity, they are green (see image). Their colour will scale as their angular speed does; i.e. they will be a paler shade of green when they are rotating slowly and a stronger shade of green when they are rotating quickly.



The heads will change colour depending on their relative mass, that is to say the darker the shade of blue, the heavier the head, and the lighter the shade of blue, the lighter the head (see image).



The graph display

This is found in the bottom left of the screen. As the pendulum is simulated, the graph will draw selected properties of the pendulum at the same time. The options for each axis are:

- The angle of the first pendulum
- The angle of the second pendulum
- The angular velocity of the first pendulum
- The angular velocity of the second pendulum

See the controls section for how to set which variable is placed on which axis.

The control panel

This is how the user interacts with the simulation. Along the top is a row of buttons:

- ❖ "Start"
  - ➢ This button begins the simulation. Both the graph and the pendulum will start simultaneously
- ❖ "Stop"
  - ➢ This button pauses the simulation. Again, both the graph and the pendulum will be stopped by this button
- ❖ "Step"

- This button steps the simulation forward one tick. This allows you to observe in detail how the pendulum is moving

❖ “Save state”
- This button saves the pendulum’s current state to a file on the disk. It will be saved as “Pendulum state save.pendulum” in the same directory as the simulation is being run in.

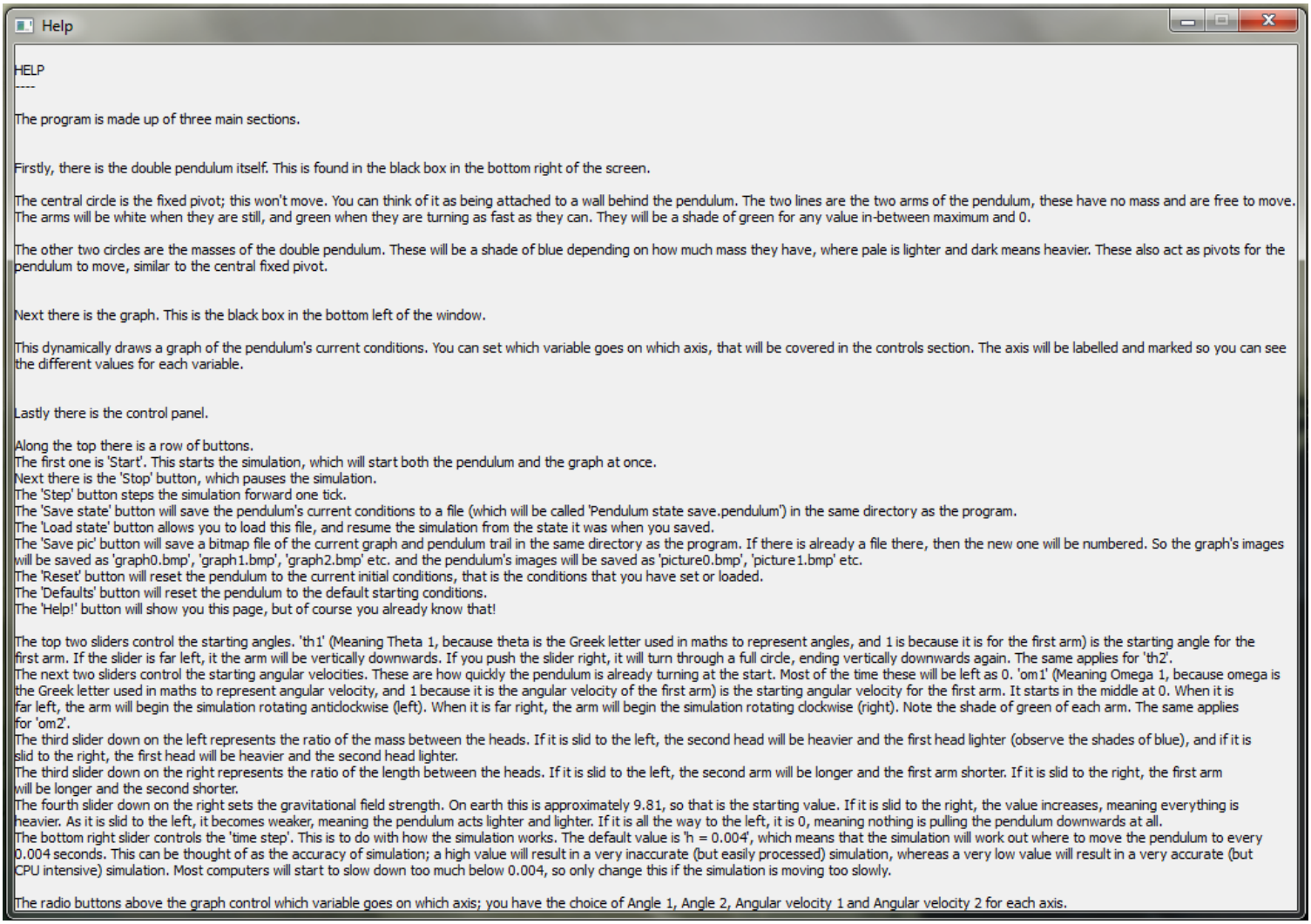
❖ “Load state”
- This button tries to load the “Pendulum state save.pendulum” file from the directory the program was run from. If it is not present, the simulation will be unaffected, i.e. if the simulation is running it will continue to do so, and if the simulation is stopped then it will remain that way.

❖ “Save pic”
- This button saves the traced path image and the graph’s image to disk as “.bmp” files. They will be saved in the same directory as the program is running. If no previous images have been saved, they will be called “picture0.bmp”“graph0.bmp”. From then on, it will keep incrementing the number until a file name is available, i.e. “picture1.bmp” and “graph1.bmp”, “picture2.bmp” and “graph2.bmp”, “picture3.bmp” and “graph3.bmp” etc.

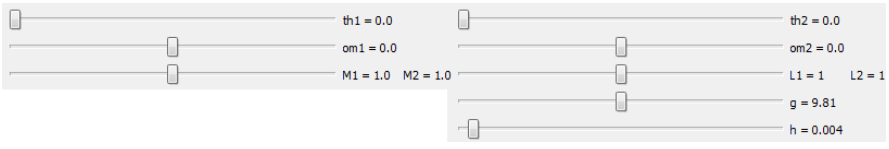
❖ “Reset”
- This button will stop the simulation and return the pendulum to its starting conditions. If a state save has been loaded since the last time the sliders have been adjusted, then this button will have the same effect as “Load state”. If a state hasn’t been loaded since the last time the sliders were adjusted, then the simulation will return to the state it was in after the sliders were last adjusted.

❖ “Defaults”
- This button will stop the simulation and change the pendulum’s attributes to those that it starts the program with.

❖ “Help!”
- This button will bring up an abridged version of this user manual.



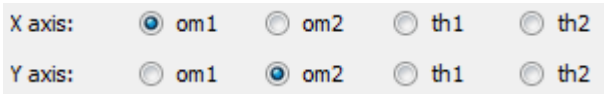
Below the row of buttons is a block of sliders. These are what you will use to enter starting conditions for the pendulum.



Here is a breakdown of what each slider does:

- Top row ("th1" and "th2")
  - These are the sliders that adjust the starting angle of each arm. "th" is shorthand for the Greek letter Theta (θ), as this is the symbol used in algebra to represent an angle. The "1" indicates that you are adjusting the angle for the first arm, and likewise the "2" indicates you are adjusting the second.
  - The starting position of each angle is 0, i.e. vertically downwards. The pendulum will rotate clockwise, until it returns to vertically downwards when the slider is far right. The angles are measured in radians; for an explanation on how they differ from degrees please visit http://goo.gl/k5AKh6.

- Second row (“om1” and “om2”)
  - These sliders adjust the starting angular velocity of each arm. “om” is shorthand for the Greek letter Omega (ω), as this is the symbol used in algebra to represent an angular velocity. The “1” indicates that you are adjusting the angular velocity of the first arm, and similarly “2” indicates you are adjusting the second.
  - These sliders start in the middle at 0, and the angular velocity increases clockwise (positive) as you slide it to the right (to a maximum of 3 radians per second), and increase anticlockwise (negative) as you slide it left (to a minimum of -3 radians per second)
- Third row, first slider (“M1”, “M2”)
  - This slider adjusts the ratio of the mass between the two heads. The M is short hand for mass. It starts in the middle, with both masses equal to 1. At the far right, “M1” will be at its greatest and “M2” at its smallest (M1 = 1.4, M2 = 0.6). When it is slid to the far left, “M2” will be at its greatest and “M1” at its smallest (M1 = 0.6, M2 = 1.4).
- Third row, second slider (“L1”, “L2”)
  - This slider adjusts the ratio of the length between the two arms. The L is short hand for length. It starts in the middle, with both lengths equal to 1. At the far right, “L1” will be at its greatest and “M2” at its smallest (L1 = 1.4, L2 = 0.6). At the far left, “L1” will be at its smallest and “M2” at its greatest (L1 = 0.6, L2 = 0.4).
- Fourth slider, right (“g”)
  - This slider adjusts the gravitational field strength for the simulation. It starts in the middle, with a value of 9.81. This is the earth’s gravitational field strength, and so it is the default value. To the far right, g will be at a maximum of 19.62, which is double earth’s gravitational field strength. To the far left, g will be 0. This will mean that there is no acceleration due to gravity in the simulation. Without gravity, the pendulum ceases to be a chaotic system. In order for the pendulum to be moving, it has to start with some angular velocity. The patterns drawn will become very regular and Spirograph-like.
- Fifth slider, right (“h”)
  - This slider adjusts the length of one “tick” in simulation. This is the length of time between each step in seconds. A high value will result in an inaccurate simulation, as there is a longer amount of time between each step the result is more approximate. A lower value will give a more accurate simulation, but it is also more CPU intensive as more calculations have to be done each second. Most CPUs can’t do the calculations fast enough for the simulation to run in real time if h is below 0.004. This is the default value, you only need to change this if the simulation is running slowly or lagging at all. The maximum value is 0.1 (slider to the far right) and the minimum value is 0.001 (slider to the far left).



Above the graph, there are two rows of radio buttons. These set which variable is on each axis. The top row sets the variable on the x-axis; the bottom row sets the variable on the y-axis. The options for each axis are angular velocity 1 (om1, default for x-axis), angular velocity 2 (om2, default for y-axis), angle 1 (th1), angle 2 (th2).

Evaluation

The objectives listed in the Analysis were as followed:

1. The program should allow the user to set the following initial attributes:
    a. Starting angle of both pendulums
    b. Starting angular velocity of both pendulums
    c. Starting mass of each head
    d. Starting length of each head
    e. Gravitational field strength
    f. Accuracy of simulation
2. The user should be limited in their entry of the initial attributes so that the pendulum does not endure extreme conditions
3. The simulation should move in real time
4. The simulation should have a graph that has customisable axis variables
5. The graph should update at the same time as the pendulum
6. The second pendulum head should draw a traced path of where it has been during the simulation
7. The users should be able to play and pause the simulation
8. The users should be able to step through the simulation
9. The pendulum's current state can be saved to and loaded from the computer's hard drive
10. The user should be able to save the images of the graph and the pendulum's trail to disk
11. The program should have a brief help section that explains to the user how to use the simulation
12. The simulation should be easy to use and user friendly.

Most of these objectives have been shown to have been met in the testing section. Here I will go through the objectives one by one in order:

1. All of these objectives have been met:
    a. Angles were successfully set in tests 23 and 24
    b. Angular velocities were successfully set in tests 25 and 26
    c. The ratio between the masses of the pendulum heads was successfully set in test 30
    d. The ratio between the lengths of the pendulum arms was successfully set in test 29
    e. Gravitational field strength was successfully set in test 27
    f. The length of the smallest step of time (and therefore the accuracy of the simulation) was successfully set in test 28
2. This was met by the use of sliders, which were successfully tested in the same tests mentioned in question 1
3. This was successfully met in test 41
4. This was successfully met in tests 31, 32 and 33

5. This was successfully met. The best example of this is the screenshot in test 34 that shows both the graph and the pendulum drawn after a set period of time
6. This was successfully tested. See test 12, 14 and 15
7. This was successfully shown in tests 36 and 37
8. This was successfully shown in test 38
9. This was successfully shown in tests 34 and 35
10. This was successfully met in tests 14, 15, 20 and 21
11. This was technically met in test 43
12. See email below

In summary, I believe that the important objectives have been met perfectly, and all of the objectives have been met. All of the technical aspects were met exactly, which roughly constitutes objectives 1 to 10. Objective 12 was met exactly, as referred to by Mr Chua's feedback. Here a transcript of the email he sent me for feedback:

"Charlie

I am very pleased with the program.

We ran into some installation issues with Python, but don't worry this was nothing to do with the program itself. The IT administrators solved the problem and it then ran perfectly. The students enjoyed the simulation very much, they were very engaged and it definitely increased their interest in chaos theory and dynamic systems, and so as a teaching aid it was a complete success. They also really liked the regular, non-chaotic paths drawn for small angles – one student went as far as to transfer the image to his phone to use as its background!

The inputs were well limited, I asked all the students to try and break the simulation with extreme input values and none of them came close. However, I did notice that it was possible for the numbers on an axis on the graph to become so densely packed that they started to write on top of each other, meaning you couldn't see what the numbers were saying.

The pendulum and the graph worked perfectly, the pendulum looked realistic and like it was moving in real time, and the graph did exactly what I required also. Changing the variables on the axes worked exactly as desired.

The images exported successfully, and the pendulum saved and loaded its state perfectly.

The help window opened as it should. However, I think there could be a potential improvement with the in-program help. I think the problem is that the help is trying to be two things at once. It is trying to be brief enough that it can be used for quick reference, but it is also trying to convey the detail level of the main User Manual. The result is that it is slightly too verbose to be a quick reference guide, but also lacks the images to make it a useful detailed reference like the user manual. I think that you should shorten it, briefly describe what each thing does and leave anything of more detail to the user manual. Having said that, the students hardly needed to read the help window, as within a short amount of time of using the program they had sussed out what nearly every element of the interface did. Most of them had to look up what the 'h' slider was, and some of them needed to look

up what the ‘g’ slider was, whilst most of them worked this one out or knew already. Perhaps I was a bit harsh about the help window, because all of them successfully found the information they were looking for.

All in all it was a great success and thank you very much for the time and effort you put into making it. I hope you get a good mark!

Mr Chua”

The two points of concern raised in the email were that the numbers on the graph sometimes became too tightly packed to be able to read, and that the help text was too verbose.

These two things could definitely be amended for future improvements of the project.

To solve the problem of the graph axes, I would need to find out how many numbers could comfortably fit, and then make a check each time the axis is drawn. If the amount of numbers to be drawn is too high, I could set it to only write every two. This could then recursively use the same check, and do so every 4, 8 etc. until the numbers comfortably fitted. Another way of doing it would be to change the scale of one “dash” on the axis so that it is more than 1 unit. This would mean that the dashes don’t get bunched up too much for higher ranges as well. However, I prefer the first idea as the input values are sufficiently limited so that the dashes shouldn’t be able to bunch up, and if the dashes stay the same unit, it is easier to compare the two axes at a glance.

The issue of the help text is easily solved. For a future improvement I could refine it so that only the essential details remained, mainly the barebones instructions of how to use the interface.

Both these problems are sufficiently small for me to conclude that overall the project was successful.

Double Pendulum simulation feedback

Mr J Chua

Actions

To: Charlie Seymour

Friday, April 25, 2014 7:21 AM

Dear Charlie,

I am very pleased with the program.

We ran into some installation issues with Python, but don't worry this was nothing to do with the program itself. The IT administrators solved the problem and it then ran perfectly. The students enjoyed the simulation very much, they were very engaged and it definitely increased their interest in chaos theory and dynamic systems, and so as a teaching aid it was a complete success. They also really liked the regular, non-chaotic paths drawn for small angles – one student went as far as to transfer the image to his phone to use as its background!

The inputs were well limited, I asked all the students to try and break the simulation with extreme input values and none of them came close. However, I did notice that it was possible for the numbers on an axis on the graph to become so densely packed that they started to write on top of each other, meaning you couldn't see what the numbers were saying.

The pendulum and the graph worked perfectly, the pendulum looked realistic and like it was moving in real time, and the graph did exactly what I required also. Changing the variables on the axes worked exactly as desired. The images exported successfully, and the pendulum saved and loaded its state perfectly.

The help window opened as it should. However, I think there could be a potential improvement with the in-program help. I think the problem is that the help is trying to be two things at once. It is trying to be brief enough that it can be used for quick reference, but it is also trying to convey the detail level of the main User Manual. The result is that it is slightly too verbose to be a quick reference guide, but also lacks the images to make it a useful detailed reference like the user manual. I think that you should shorten it, briefly describe what each thing does and leave anything of more detail to the user manual. Having said that, the students hardly needed to read the help window, as within a short amount of time of using the program they had sussed out what nearly all of the interface did. Most of them had to look up what the 'h' slider was, and some of them needed to look up what the 'g' slider was, whilst most of them worked this one out or knew already. Perhaps I was a bit harsh about the help window, because all of them successfully found the information they were looking for.

All in all, it was a great success and thank you very much for the time and effort you put into making it. I hope you get a good mark!

Regards,
Mr Chua

Jude Chua
Department of Mathematics
Kingswood School

Tel: +44 (0)1225 734200
Fax: +44 (0)1225 734305

Appendix I: Code

```
# Double Pendulum Simulation
# by Charlie Seymour

# "To improve is to change; to be perfect is to change often."
# - Winston Churchill

import sys, pickle
from PyQt4 import QtGui, QtCore
from math import sin, cos, acos, pi, sqrt, floor

def cutAngle(th):
    # This function ensures that any angle will be kept within the
range
    # of -pi --> pi. This is the standard range for representing an
angle in
    # radians. This is in place to ensure angles do not grow large
enough to
    # slow down the program.
    if th > 0:
        if th < pi:
            return th
        elif th < 2*pi:
            return th-2*pi
        else:
            return cutAngle(((th/pi)%2)*pi)
    else:
        if th > -pi:
            return th
        elif th > -2*pi:
            return th + 2*pi
        else:
            return cutAngle(((th/pi)%2)*pi)

def getPictureName(name, ext):
    # This function will find the next available file name,
following the
    # pattern 'name0.ext', 'name1.ext' etc, to ensure the saved
images
    # do not overwrite old ones.
        i = 0
        while True:
            try:
                open(name+str(i)+ext)
                i += 1
            except:
                break
        return name+str(i)+ext

class Communicator(QtCore.QObject):
```

```
    # This is the object that will send out the simulation's tick

    def __init__(self):

        super().__init__()

        self.pendulumTick = QtCore.QTimer()

class DoublePendulum(QtGui.QWidget):
    # This is the double pendulum widget

    # custom signal to send out attributes to graph each tick
    attributeSignal = QtCore.pyqtSignal(float, float, float, float)

    def __init__(self, L1, L2, M1, M2, g, th1, th2, om1, om2, h):
        # initialises pendulum
        super().__init__()
        self.initAttributes(L1, L2, M1, M2, g, th1, th2, om1, om2,
h)
        self.setXY()
        self.calculateEnergy()

        self.setFixedSize(self.sideLength, self.sideLength)

        #this will be the bmp that the trail will be drawn on
        self.canvas = QtGui.QImage(self.sideLength, self.sideLength,
QtGui.QImage.Format_RGB32)
        self.canvas.fill(QtGui.QColor(0, 0, 0))

    def initAttributes(self, L1, L2, M1, M2, g, th1, th2, om1, om2,
h):
        # Intialises attributes

        # constants
        self.L1  = L1
        self.L2  = L2
        self.M1  = M1
        self.M2  = M2
        self.g   = g

        # This is the pixel scale to draw by - e.g if the point is
(0.8, 0.2)
        # mathematically, then the pixel coordinates will be (80,
20) if
        # dScale is 100
        self.dScale = 100

        self.sideLength = self.dScale*2*(self.L1 + self.L2)

        # variables
        self.x1  = 0.0
        self.x2  = 0.0  # these four will be changed straight away
but
        self.y1  = 0.0  # setting them here will also create them
        self.y2  = 0.0
```

```
        self.th1 = th1
        self.th2 = th2
        self.om1 = om1
        self.om2 = om2

        # delta t
        self.h = h

    def setXY(self):
        # This sets the coordinates by using trigonometry, and the
current
        # angles. the 'sh' variables are altered to give the pixel
coordinates
        # for drawing the pendulum heads.

        self.x1 = self.L1*sin(self.th1)
        self.y1 = -self.L1*cos(self.th1)

        self.x2 = self.x1 + self.L2*sin(self.th2)
        self.y2 = self.y1 - self.L2*cos(self.th2)

        shift = self.L1 + self.L2

        self.x1sh = (self.x1+shift)*self.dScale
        self.y1sh = -(self.y1-shift)*self.dScale

        self.x2sh = (self.x2+shift)*self.dScale
        self.y2sh = -(self.y2-shift)*self.dScale

    def calculateEnergy(self):

        # See documentation for derivation. These values will allow
the
        # graph to know the maximum value it can have on an axis
        self.ke = 0.5*(self.M1*(self.L1**2)*(self.om1**2) +
self.M2*((self.L1**2)*(self.om1**2) + (self.L2**2)*(self.om2**2) +
2*self.L1*self.L2*self.om1*self.om2*cos(self.th1-self.th2)))
        shift = self.L1 + self.L2
        self.gpe = self.g*(self.M1*(self.y1+shift) +
self.M2*(self.y2+shift))
        self.energy = self.ke + self.gpe
        self.gpeMin = self.g*self.L2*self.M1
        self.om1Max = self.omegaMax(1)
        self.om2Max = self.omegaMax(2)

        # if there is enough energy, the theoretical maximum angle
could be
        # larger than vertical. This doesn't make sense, so this
ensures that
        # pi is the the maximum possible angle (180 degrees)
        if self.energy >=
(self.g*(2*self.L1*(self.M1+self.M2)+self.M1*self.L2)):
            self.th1Max = pi
        else:
```

```
            self.th1Max = acos(round((-
self.energy+self.g*(shift*(self.M1 +self.M2)-
self.M2*self.L2))/(self.g*self.L1*(self.M1 + self.M2)), 14))

        if self.energy >= (self.L2*self.g*(self.M1 + 2*self.M2)):
            self.th2Max = pi
        else:
            self.th2Max = acos(round((-
self.energy+self.g*(self.M1+self.M2)*(shift-
self.L1))/(self.M2*self.g*self.L2), 14))

    def omegaMax(self, num):
        # This calculates the maximum value for omega 1 or 2,
depending on
        # whether num = 1 or 2
        # See documentation for derivation

        A = 0.5*(self.L1**2)*(self.M1 + self.M2)
        B = 0.5*self.M2*(self.L2**2)
        C = self.L1*self.L2*self.M2
        D = self.energy - self.gpeMin

        if num == 2:
            temp = A
            A = B
            B = temp

        F = C**2 - 4*A*B
        G = 4*A*D

        om2 = -sqrt(round((G*(C**2))/((F**2)-F*(C**2)), 13))
        # note: this is om2 that will give max om1, this is NOT
self.om2
        omMax = (-C*om2+sqrt(round((C**2)*(om2**2)-
(4*A*B*(om2**2))+4*A*D, 13)))/(2*A)
        return omMax

    def om1d(self, varList):
        # This takes a list of necessary variables and calculates a
value
        # for the current angular acceleration of the first arm.
        # See documentation for derivation.
        th1 = varList[0]
        th2 = varList[1]
        om1 = varList[2]
        om2 = varList[3]
        m1 = self.M1
        m2 = self.M2
        L1 = self.L1
        L2 = self.L2
        g = self.g
        return (-g*(2*m1 + m2)*sin(th1) - m2*g*sin(th1-2*th2)-
2*sin(th1-th2)*m2*((om2**2)*L2 + (om1**2)*L1*cos(th1-
th2)))/(L1*(2*m1 + m2 - m2*cos(2*th1-2*th2)))

    def om2d(self, varList):
```

```
        # This takes a list of necessary variables and calculates a
value
        # for the current angular acceleration of the second arm.
        # See documentation for derivation.
        th1 = varList[0]
        th2 = varList[1]
        om1 = varList[2]
        om2 = varList[3]
        m1 = self.M1
        m2 = self.M2
        L1 = self.L1
        L2 = self.L2
        g = self.g
        return (2*sin(th1-
th2)*((om1**2)*L1*(m1+m2)+g*(m1+m2)*cos(th1)+(om2**2)*L2*m2*cos(th1-
th2)))/(L2*(2*m1+m2-m2*cos(2*th1 - 2*th2)))

    def th1d(self, varList):
        # as th1' = om1, this simply return om1.
        return varList[2]

    def th2d(self, varList):
        # as th2' = om2, this simply return om2.
        return varList[3]

    def rungeKutta(self, varList, funcList):
        # See documentation for explanation of Runge Kutta method
        newVarList = []
        h = self.h
        aList = []

        for func in funcList:
            aList.append(func(varList))

        bList = self.multiFuncVect(varList, funcList, aList, h/2)
        cList = self.multiFuncVect(varList, funcList, bList, h/2)
        dList = self.multiFuncVect(varList, funcList, cList, h)

        for var, a, b, c, d in zip(varList, aList, bList, cList,
dList):
            newVarList.append(var + (h/6)*(a + 2*b + 2*c + d))

        return newVarList


    def multiFuncVect(self, varList, funcList, letList, h):
        # Part of the Runge Kutta algorithm. See documentation
        newLetList = []
        tempVarList = []

        for var, let in zip(varList, letList):
            tempVarList.append(var + h*let)

        for func in funcList:
            newLetList.append(func(tempVarList))
```

```
        return newLetList

    def advance(self):
        # Advances pendulum one step
        # Implements Runge Kutte then assigns the new attributes

        varList = [self.th1, self.th2, self.om1, self.om2]
        funcList = [self.th1d, self.th2d, self.om1d, self.om2d]

        varList = self.rungeKutta(varList, funcList)

        self.th1, self.th2, self.om1, self.om2 = varList
        self.th1 = cutAngle(self.th1)
        self.th2 = cutAngle(self.th2)
        self.setXY()

    def paintEvent(self, event):
        # This describes how the pendulum is drawn to the screen
        headWidth = 16
        headRadius = headWidth // 2
        s = self.dScale*(self.L1 + self.L2)/2 # this is the pixel
shift
                                              # mentioned earlier

        # qp is the QPainter object that we will control to do the
drawing
        qp = QtGui.QPainter()
        qp.begin(self)

        # draw the second pendulum's point onto the trail image
        self.canvas.setPixel(self.x2sh, self.y2sh, QtGui.qRgb(255,
0, 0))
        qp.drawImage(0, 0, self.canvas)

        # calculate the head colour depending on the mass
        # i.e. more blue = more mass, more white = less mass
        head0colour = QtGui.QColor(255, 255, 255)
        colFrac = 1-self.M1/2
        head1colour = QtGui.QColor(255*colFrac, 255*colFrac, 255)
        colFrac = 1-self.M2/2
        head2colour = QtGui.QColor(255*colFrac, 255*colFrac, 255)

        # calculate the arm colour depending on the angular velocity
        # i.e more green = closer to max speed, more white = closer
to still
        # if-else check is to stop error being raised if pendulum
starts
        # vertically downwards
        if self.om1Max == 0:
            line1colour = QtGui.QColor(255, 255, 255)
        else:
            colFrac = 1 - abs(self.om1)/self.om1Max
            line1colour = QtGui.QColor(255*colFrac, 255,
255*colFrac)
        if self.om2Max == 0:
            line2colour = QtGui.QColor(255, 255, 255)
```

```
        else:
            colFrac = 1 - abs(self.om2)/self.om2Max
            line2colour = QtGui.QColor(255*colFrac, 255,
255*colFrac)


        # draw the arms and heads
        qp.setBrush(head0colour)
        qp.setPen(head0colour)
        qp.drawEllipse(2*s - headRadius, 2*s - headRadius,
headWidth, headWidth)

        qp.setBrush(head1colour)
        qp.setPen(head1colour)
        qp.drawEllipse(self.x1sh - headRadius, self.y1sh -
headRadius, headWidth, headWidth)

        qp.setBrush(head2colour)
        qp.setPen(head2colour)
        qp.drawEllipse(self.x2sh - headRadius, self.y2sh -
headRadius, headWidth, headWidth)

        qp.setBrush(line1colour)
        qp.setPen(line1colour)
        qp.drawLine(2*s, 2*s, self.x1sh, self.y1sh)

        qp.setBrush(line2colour)
        qp.setPen(line2colour)
        qp.drawLine(self.x1sh, self.y1sh, self.x2sh, self.y2sh)
        qp.end()

    def move(self):
        # advance pendulum, send signal to graph, repaint
        self.advance()
        self.attributeSignal.emit(self.th1, self.th2, self.om1,
self.om2)
        self.repaint()

    def step(self):
        # originally implemented for debugging, this function will
display
        # the pendulum's current attributes, and also advance by one
tick
        self.calculateEnergy()
        print('     x1:', self.x1)
        print('     y1:', self.y1)
        print('     x2:', self.x2)
        print('     y2:', self.y2)
        print('    th1:', self.th1)
        print('    th2:', self.th2)
        print('    om1:', self.om1)
        print('    om2:', self.om2)
        print('     ke:', self.ke)
        print('    gpe:', self.gpe)
        print('om1 max:', self.om1Max)
        print('om2 max:', self.om2Max)
```

```
            print('th1 max:', self.th1Max)
            print('th2 max:', self.th2Max)
            print(' gpeMin:', self.gpeMin)
            print(' energy:', self.energy)
            print()
            self.move()

        def export(self):
            # save image of trail
            name = getPictureName('picture', '.bmp')
            self.canvas.save(name, None, 100)

        def saveState(self):
            # save the penudlum's current state to file
            saveFile = open('Pendulum state save.pendulum', 'wb')
            attributes = (self.L1, self.L2, self.M1, self.M2, self.g,
self.th1, self.th2, self.om1, self.om2, self.h)
            pickle.dump(attributes, saveFile)
            saveFile.close()

    class GraphWidget(QtGui.QWidget):
        # This is the graph widget

        def __init__(self, xAxisVar, yAxisVar, xMinMax, yMinMax,
sideLength):
            # intialise the graph

            super().__init__()
            self.setFixedSize(sideLength, sideLength)

            # This is the bitmap that will be drawn on
            self.canvas = QtGui.QImage(sideLength, sideLength,
QtGui.QImage.Format_RGB32)
            self.canvas.fill(QtGui.QColor(0, 0, 0))

            # get attributes from __init__'s arguments
            self.xAxisVar = xAxisVar
            self.xVal = 0.0
            self.xValAdj = 25
            self.xMin = xMinMax[0]
            self.xMax = xMinMax[1]
            self.yAxisVar = yAxisVar
            self.yVal = 0.0
            self.yValAdj = 25
            self.yMin = yMinMax[0]
            self.yMax = yMinMax[1]
            self.sideLength = sideLength
            # drawing scale (same idea as in the pendulum)
            if (self.xMax-self.xMin) == 0:
                self.xDScale = 0
            else:
                self.xDScale = (sideLength-50)/(self.xMax-self.xMin)

            if (self.yMax-self.yMin) == 0:
                self.yDScale = 0
            else:
```

```
            self.yDScale = (sideLength-50)/(self.yMax-self.yMin)

    def paintEvent(self, event):

        qp = QtGui.QPainter()
        qp.begin(self)

        #draw points
        qp.drawImage(0, 0, self.canvas)

        #draw axis
        qp.setPen(QtGui.QColor(255, 255, 255))
        topLeft = QtCore.QPoint(25, 25)
        bottomLeft = QtCore.QPoint(25, self.sideLength-25)
        bottomRight = QtCore.QPoint(self.sideLength-25,
self.sideLength-25)
        qp.drawLine(topLeft, bottomLeft)
        qp.drawLine(bottomLeft, bottomRight)

        #draw markings
        #write number markings
        for i in range(floor(self.xMin)+1, floor(self.xMax)+1):  #x
markings
            x = self.adjustX(i)
            qp.drawLine(x, self.sideLength - 25, x, self.sideLength
- 20)
            qr = qp.fontMetrics().boundingRect(str(i))
            qp.drawText(x-(qr.width()/2), self.sideLength - 6,
str(i))

        for j in range(floor(self.yMin)+1, floor(self.yMax)+1): #y
markings
            y = self.adjustY(j)
            qp.drawLine(25, y, 20, y)
            qr = qp.fontMetrics().boundingRect(str(j))
            qp.drawText((20-qr.width())/2, y+qr.height()/3, str(j))

        #label axis
        qr = qp.fontMetrics().boundingRect(self.xAxisVar)
        qp.drawText(self.sideLength-22, self.sideLength-25 +
qr.height()/3, self.xAxisVar)
        qr = qp.fontMetrics().boundingRect(self.yAxisVar)
        qp.drawText(25-qr.width()/2, 23, self.yAxisVar)

        #draw head
        qp.drawRect(self.xValAdj-1, self.yValAdj-4, 3, 9)
        qp.drawRect(self.xValAdj-4, self.yValAdj-1, 9, 3)

        qp.end()

    def adjustPoints(self):
        # The two functions called here adjust the points so that
they lie
        # inside the axis and peform the necessary 'pixel shift'
performed earlier
        self.xValAdj = self.adjustX(self.xVal)
```

```
        self.yValAdj = self.adjustY(self.yVal)

    def adjustX(self, x):
        return (x-self.xMin)*self.xDScale + 25

    def adjustY(self, y):
        return self.sideLength - ((y-self.yMin)*self.yDScale + 25)

    def step(self, th1, th2, om1, om2):
        # this function takes the values from the graph, selects the
ones
        # it needs depending on what is on each axis, draws the
point onto
        # the bitmap then repaints the graph
        if self.xAxisVar == 'th1':
            self.xVal = th1
        if self.xAxisVar == 'th2':
            self.xVal = th2
        if self.xAxisVar == 'om1':
            self.xVal = om1
        if self.xAxisVar == 'om2':
            self.xVal = om2

        if self.yAxisVar == 'th1':
            self.yVal = th1
        if self.yAxisVar == 'th2':
            self.yVal = th2
        if self.yAxisVar == 'om1':
            self.yVal = om1
        if self.yAxisVar == 'om2':
            self.yVal = om2

        self.adjustPoints()
        self.canvas.setPixel(self.xValAdj, self.yValAdj,
QtGui.qRgb(255, 0, 0))
        self.repaint()

    def export(self):
        # save the graph image to file
        name = getPictureName('graph', '.bmp')
        self.canvas.save(name, None, 100)


class DpWindow(QtGui.QWidget):
    # this is the window that will contain all the widgets

    def __init__(self):
        # intialise everything
        super().__init__()
        self.setInitialPendulumAttributes()
        self.initPendulum()
        self.initComms()
        self.setInitialGraphAttributes()
        self.initGraph()
        self.initUI()
        self.initConnections()
```

```
        self.move(100, 100)
        self.show()

    def setInitialPendulumAttributes(self, iL1 = 1, iL2 = 1, iM1 =
1, iM2 = 1, ig = 9.81, ith1 = 0, ith2 = 0, iom1 = 0, iom2 = 0, ih =
0.004):
        # if this is called without arguments, it will set default
values.
        # otherwise, it can be used to set all of a pendulum's
initial
        # attributes.
        self.iL1 = iL1
        self.iL2 = iL2
        self.iM1 = iM1
        self.iM2 = iM2
        self.ig = ig
        self.ith1 = ith1
        self.ith2 = ith2
        self.iom1 = iom1
        self.iom2 = iom2
        self.ih = ih

    def setInitialGraphAttributes(self):
        # default variables for the x and y axes
        self.ixVar = 'om1'
        self.iyVar = 'om2'
        self.setGraphMinMax()

    def setGraphMinMax(self):
        # create a dictionary that stores the min and max variables
        # with the name of the variable on the axis as the key.
        # This is simply so that a long if-elif-else clause is not
        # needed
        minMaxDict = {'om1':(-self.pendulum.om1Max,
self.pendulum.om1Max),
                      'om2':(-self.pendulum.om2Max,
self.pendulum.om2Max),
                      'th1':(-self.pendulum.th1Max,
self.pendulum.th1Max),
                      'th2':(-self.pendulum.th2Max,
self.pendulum.th2Max)}
        self.ixMinMax = minMaxDict[self.ixVar]
        self.iyMinMax = minMaxDict[self.iyVar]
        self.isideLength = self.pendulum.sideLength

    def initPendulum(self):
        # create the pendulum object (widget)
        self.pendulum = DoublePendulum(self.iL1, self.iL2, self.iM1,
self.iM2, self.ig, self.ith1, self.ith2, self.iom1, self.iom2,
self.ih)

    def initComms(self):
        # create the communicator object
        self.comms = Communicator()
```

```
    def initGraph(self):
        # create the graph object (widget)
        self.graph = GraphWidget(self.ixVar, self.iyVar,
self.ixMinMax, self.iyMinMax, self.isideLength)

    def resetWidgets(self):
        # all the steps needed to re-initialise the widgets if their
        # starting attributes have been changed
        self.removeConnections()
        self.pendulum.deleteLater()
        self.graph.deleteLater()
        self.initPendulum()
        self.initComms()
        self.setGraphMinMax()
        self.initGraph()
        self.initConnections()
        self.addGraphToGrid()
        self.addPendulumToGrid()

    # all of the set (x) from slider functions below follow a
similar pattern.
    # they multiply the value of the slider by their maximum value
divided
    # by the slider's maximum. This effectively scales their value
as the
    # slider scales. Some are slightly different to take into
accuont things
    # like negatives, or non zero minimums, however the basic idea
stays
    # the same

    def setTh1FromSlider(self, val):
        angle = (2*pi/100)*val
        self.ith1 = angle
        self.th1Label.setText('th1 = ' + str(round(self.ith1, 3)))
        self.resetWidgets()

    def setTh2FromSlider(self, val):
        angle = (2*pi/100)*val
        self.ith2 = angle
        self.th2Label.setText('th2 = ' + str(round(self.ith2, 3)))
        self.resetWidgets()

    def setOm1FromSlider(self, val):
        self.iom1 = (3/50)*val
        self.om1Label.setText('om1 = ' + str(round(self.iom1, 3)))
        self.resetWidgets()

    def setOm2FromSlider(self, val):
        self.iom2 = (3/50)*val
        self.om2Label.setText('om2 = ' + str(round(self.iom2, 3)))
        self.resetWidgets()

    def setGfromSlider(self, val):
        self.ig = (19.62/100)*val
        self.gLabel.setText('g = ' + str(round(self.ig, 3)))
```

```
        self.resetWidgets()

    def setHfromSlider(self, val):
        self.ih = 0.001 * val
        self.hLabel.setText('h = ' + str(round(self.ih, 5)))
        self.resetWidgets()

    # these functions that set ratios are similar to the ones above,
    # except they take steps to ensure that the total mass or length
    # is always 2

    def setLengthRatioFromSlider(self, ratio):
        """ratio is the fraction of length to be given to L1"""
        ratio /= 100
        self.iL1 = 2*ratio
        self.L1Label.setText('L1 = ' + str(round(self.iL1, 3)))
        self.iL2 = 2*(1-ratio)
        self.L2Label.setText('L2 = ' + str(round(self.iL2, 3)))
        self.resetWidgets()

    def setMassRatioFromSlider(self, ratio):
        """ratio is the fraction of length to be given to M1"""
        ratio /= 100
        self.iM1 = 2*ratio
        self.M1Label.setText('M1 = ' + str(round(self.iM1, 3)))
        self.iM2 = 2*(1-ratio)
        self.M2Label.setText('M2 = ' + str(round(self.iM2, 3)))
        self.resetWidgets()

    # setting the graph's axis usign the radio button's group id
    def setGraphYVar(self, yvarID):
        if yvarID == 1:
            self.iyVar = 'om1'
        if yvarID == 2:
            self.iyVar = 'om2'
        if yvarID == 3:
            self.iyVar = 'th1'
        if yvarID == 4:
            self.iyVar = 'th2'
        self.resetWidgets()

    def setGraphXVar(self, xvarID):
        if xvarID == 1:
            self.ixVar = 'om1'
        if xvarID == 2:
            self.ixVar = 'om2'
        if xvarID == 3:
            self.ixVar = 'th1'
        if xvarID == 4:
            self.ixVar = 'th2'
        self.resetWidgets()

    # load the pendulum save state from file, resets the widgets
    def pendulumLoadState(self):
        saveFile = open('Pendulum state save.pendulum', 'rb')
        attributes = pickle.load(saveFile)
```

```
            saveFile.close()
            self.setInitialPendulumAttributes(*attributes)
            self.resetWidgets()

        #reset pendulum and graph to the values they have when the
program starts
        def setDefaults(self):
            self.setInitialPendulumAttributes()
            self.resetSlidersRBs()
            self.setInitialGraphAttributes()
            self.resetSlidersRBs()
            self.resetWidgets()

        def showHelp(self):
            # opens a new window that has helpText in it
            self.helpText = """
HELP
----

The program is made up of three main sections.


Firstly, there is the double pendulum itself. This is found in the
black box in the bottom right of the screen.

The central circle is the fixed pivot; this won't move. You can
think of it as being attached to a wall behind the pendulum. The two
lines are the two arms of the pendulum, these have no mass and are
free to move.
The arms will be white when they are still, and green when they are
turning as fast as they can. They will be a shade of green for any
value in-between maximum and 0.

The other two circles are the masses of the double pendulum. These
will be a shade of blue depending on how much mass they have, where
pale is lighter and dark means heavier. These also act as pivots for
the
pendulum to move, similar to the central fixed pivot.


Next there is the graph. This is the black box in the bottom left of
the window.

This dynamically draws a graph of the pendulum's current conditions.
You can set which variable goes on which axis, that will be covered
in the controls section. The axis will be labelled and marked so you
can see
the different values for each variable.


Lastly there is the control panel.

Along the top there is a row of buttons.
The first one is 'Start'. This starts the simulation, which will
start both the pendulum and the graph at once.
Next there is the 'Stop' button, which pauses the simulation.
```

The 'Step' button steps the simulation forward one tick.
The 'Save state' button will save the pendulum's current conditions to a file (which will be called 'Pendulum state save.pendulum') in the same directory as the program.
The 'Load state' button allows you to load this file, and resume the simulation from the state it was when you saved.
The 'Save pic' button will save a bitmap file of the current graph and pendulum trail in the same directory as the program. If there is already a file there, then the new one will be numbered. So the graph's images
will be saved as 'graph0.bmp', 'graph1.bmp', 'graph2.bmp' etc. and the pendulum's images will be saved as 'picture0.bmp', 'picture1.bmp' etc.
The 'Reset' button will reset the pendulum to the current initial conditions, that is the conditions that you have set or loaded.
The 'Defaults' button will reset the pendulum to the default starting conditions.
The 'Help!' button will show you this page, but of course you already know that!

The top two sliders control the starting angles. 'th1' (Meaning Theta 1, because theta is the Greek letter used in maths to represent angles, and 1 is because it is for the first arm) is the starting angle for the
first arm. If the slider is far left, it the arm will be vertically downwards. If you push the slider right, it will turn through a full circle, ending vertically downwards again. The same applies for 'th2'.
The next two sliders control the starting angular velocities. These are how quickly the pendulum is already turning at the start. Most of the time these will be left as 0. 'om1' (Meaning Omega 1, because omega is
the Greek letter used in maths to represent angular velocity, and 1 because it is the angular velocity of the first arm) is the starting angular velocity for the first arm. It starts in the middle at 0. When it is
far left, the arm will begin the simulation rotating anticlockwise (left). When it is far right, the arm will begin the simulation rotating clockwise (right). Note the shade of green of each arm. The same applies
for 'om2'.
The third slider down on the left represents the ratio of the mass between the heads. If it is slid to the left, the second head will be heavier and the first head lighter (observe the shades of blue), and if it is
slid to the right, the first head will be heavier and the second head lighter.
The third slider down on the right represents the ratio of the length between the heads. If it is slid to the left, the second arm will be longer and the first arm shorter. If it is slid to the right, the first arm
will be longer and the second shorter.
The fourth slider down on the right sets the gravitational field strength. On earth this is approximately 9.81, so that is the starting value. If it is slid to the right, the value increases, meaning everything is

```
heavier. As it is slid to the left, it becomes weaker, meaning the
pendulum acts lighter and lighter. If it is all the way to the left,
it is 0, meaning nothing is pulling the pendulum downwards at all.
The bottom right slider controls the 'time step'. This is to do with
how the simulation works. The default value is 'h = 0.004', which
means that the simulation will work out where to move the pendulum
to every
0.004 seconds. This can be thought of as the accuracy of simulation;
a high value will result in a very inaccurate (but easily processed)
simulation, whereas a very low value will result in a very accurate
(but
CPU intensive) simulation. Most computers will start to slow down
too much below 0.004, so only change this if the simulation is
moving too slowly.

The radio buttons above the graph control which variable goes on
which axis; you have the choice of Angle 1, Angle 2, Angular
velocity 1 and Angular velocity 2 for each axis."""

        self.helpWindow = QtGui.QWidget()
        self.helpWindow.setWindowTitle("Help")
        self.helpWindow.setWindowIcon(QtGui.QIcon("helpIcon.png"))
        self.helpLabel = QtGui.QLabel(self.helpText)
        self.helpLabel.setParent(self.helpWindow)
        self.helpWindow.setFixedSize(self.helpLabel.sizeHint())
        self.helpWindow.show()

    def resetSlidersRBs(self):
        # sets all sliders and radio buttons to their default values
        self.th1Slider.setValue(0)
        self.th2Slider.setValue(0)
        self.om1Slider.setValue(0)
        self.om2Slider.setValue(0)
        self.massSlider.setValue(50)
        self.lengthSlider.setValue(50)
        self.gSlider.setValue(50)
        self.hSlider.setValue(4)
        self.om1RBX.click()
        self.om2RBY.click()

    def initUI(self):
        # this initialises the UI. It creates, labels and resizes
all the
        # buttons, creates and labels all the sliders, creates all
the
        # radio buttons, then puts all of them into layouts that
will
        # arrange their position on the screen
        self.setFixedSize(self.sizeHint())
        self.setWindowTitle("Double Pendulum Simulation")
        self.setWindowIcon(QtGui.QIcon("icon.png"))

        # Buttons
        self.startButton = QtGui.QPushButton('Start')
        self.startButton.setFixedSize(50, 50)
        self.stopButton  = QtGui.QPushButton('Stop')
```

```
        self.stopButton.setFixedSize(50, 50)
        self.stepButton = QtGui.QPushButton('Step')
        self.stepButton.setFixedSize(50, 50)
        self.stateSaveButton = QtGui.QPushButton('Save\nstate')
        self.stateSaveButton.setFixedSize(50, 50)
        self.stateLoadButton = QtGui.QPushButton('Load\nstate')
        self.stateLoadButton.setFixedSize(50, 50)
        self.pictureSaveButton = QtGui.QPushButton('Save\nimage')
        self.pictureSaveButton.setFixedSize(50, 50)
        self.resetButton = QtGui.QPushButton('Reset')
        self.resetButton.setFixedSize(50, 50)
        self.defaultsButton = QtGui.QPushButton('Defaults')
        self.defaultsButton.setFixedSize(50, 50)
        self.helpButton = QtGui.QPushButton('Help!')
        self.helpButton.setFixedSize(50, 50)
        self.helpButton.setStyleSheet('QPushButton {color : red}')

        # Sliders
        self.th1Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.th1Slider.setRange(0, 100)
        self.th2Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.th2Slider.setRange(0, 100)
        self.om1Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.om1Slider.setRange(-50, 50)
        self.om2Slider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.om2Slider.setRange(-50, 50)
        self.massSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.massSlider.setRange(30, 70)
        self.massSlider.setValue(50)
        self.lengthSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.lengthSlider.setRange(30, 70)
        self.lengthSlider.setValue(50)
        self.gSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.gSlider.setRange(0, 100)
        self.gSlider.setValue(50)
        self.hSlider = QtGui.QSlider(QtCore.Qt.Horizontal)
        self.hSlider.setRange(1, 100)
        self.hSlider.setValue(4)

        # Labels
        self.th1Label = QtGui.QLabel('th1 = ' +
str(round(self.ith1,3)))
        self.th2Label = QtGui.QLabel('th2 = ' +
str(round(self.ith2,3)))
        self.om1Label = QtGui.QLabel('om1 = ' +
str(round(self.iom1,3)))
        self.om2Label = QtGui.QLabel('om2 = ' +
str(round(self.iom2,3)))
        self.M1Label = QtGui.QLabel('M1 = ' +
str(round(self.iM1,3)))
        self.M2Label = QtGui.QLabel('M2 = ' +
str(round(self.iM2,3)))
        self.L1Label = QtGui.QLabel('L1 = ' +
str(round(self.iL1,3)))
        self.L2Label = QtGui.QLabel('L2 = ' +
str(round(self.iL2,3)))
```

```
self.gLabel = QtGui.QLabel('g = ' + str(round(self.ig,3)))
self.hLabel = QtGui.QLabel('h = ' + str(round(self.ih,5)))

#   Radio Buttons
self.RBXGroup = QtGui.QButtonGroup()
self.RBYGroup = QtGui.QButtonGroup()

# X axis radio buttons
self.om1RBX = QtGui.QRadioButton('om1')
self.RBXGroup.addButton(self.om1RBX, 1)
self.om2RBX = QtGui.QRadioButton('om2')
self.RBXGroup.addButton(self.om2RBX, 2)
self.th1RBX = QtGui.QRadioButton('th1')
self.RBXGroup.addButton(self.th1RBX, 3)
self.th2RBX = QtGui.QRadioButton('th2')
self.RBXGroup.addButton(self.th2RBX, 4)

# X axis radio buttons
self.om1RBY = QtGui.QRadioButton('om1')
self.RBYGroup.addButton(self.om1RBY, 1)
self.om2RBY = QtGui.QRadioButton('om2')
self.RBYGroup.addButton(self.om2RBY, 2)
self.th1RBY = QtGui.QRadioButton('th1')
self.RBYGroup.addButton(self.th1RBY, 3)
self.th2RBY = QtGui.QRadioButton('th2')
self.RBYGroup.addButton(self.th2RBY, 4)

self.om1RBX.toggle()
self.om2RBY.toggle()


# Radio button boxes
self.RBXbox = QtGui.QHBoxLayout()
self.RBXbox.addWidget(QtGui.QLabel("X axis:"))
self.RBXbox.addWidget(self.om1RBX)
self.RBXbox.addWidget(self.om2RBX)
self.RBXbox.addWidget(self.th1RBX)
self.RBXbox.addWidget(self.th2RBX)

self.RBYbox = QtGui.QHBoxLayout()
self.RBYbox.addWidget(QtGui.QLabel("Y axis:"))
self.RBYbox.addWidget(self.om1RBY)
self.RBYbox.addWidget(self.om2RBY)
self.RBYbox.addWidget(self.th1RBY)
self.RBYbox.addWidget(self.th2RBY)


#   Slider grid
# Th sliders
self.sliderRBGrid = QtGui.QGridLayout()
self.sliderRBGrid.addWidget(self.th1Slider, 0, 0)
self.sliderRBGrid.addWidget(self.th1Label, 0, 1)
self.sliderRBGrid.addWidget(self.th2Slider, 0, 3)
self.sliderRBGrid.addWidget(self.th2Label, 0, 4)

# Om sliders
```

```
        self.sliderRBGrid.addWidget(self.om1Slider, 1, 0)
        self.sliderRBGrid.addWidget(self.om1Label, 1, 1)
        self.sliderRBGrid.addWidget(self.om2Slider, 1, 3)
        self.sliderRBGrid.addWidget(self.om2Label, 1, 4)

        # M/L sliders
        self.sliderRBGrid.addWidget(self.massSlider, 2, 0)
        self.sliderRBGrid.addWidget(self.M1Label, 2, 1)
        self.sliderRBGrid.addWidget(self.M2Label, 2, 2)
        self.sliderRBGrid.addWidget(self.lengthSlider, 2, 3)
        self.sliderRBGrid.addWidget(self.L1Label, 2, 4)
        self.sliderRBGrid.addWidget(self.L2Label, 2, 5)

        # g/h sliders
        self.sliderRBGrid.addWidget(self.gSlider, 3, 3)
        self.sliderRBGrid.addWidget(self.gLabel, 3, 4)
        self.sliderRBGrid.addWidget(self.hSlider, 4, 3)
        self.sliderRBGrid.addWidget(self.hLabel, 4, 4)

        # Radio buttons
        self.sliderRBGrid.addLayout(self.RBXbox, 3, 0)
        self.sliderRBGrid.addLayout(self.RBYbox, 4, 0)

        # Button box
        self.buttonBox = QtGui.QHBoxLayout()
        self.buttonBox.addWidget(self.startButton)
        self.buttonBox.addWidget(self.stopButton)
        self.buttonBox.addWidget(self.stepButton)
        self.buttonBox.addWidget(self.stateSaveButton)
        self.buttonBox.addWidget(self.stateLoadButton)
        self.buttonBox.addWidget(self.pictureSaveButton)
        self.buttonBox.addWidget(self.resetButton)
        self.buttonBox.addWidget(self.defaultsButton)
        self.buttonBox.addWidget(self.helpButton)


        # Pendulum and graph grid
        self.penGraGrid = QtGui.QGridLayout()

        # Set layouts (overall grid)
        self.grid = QtGui.QGridLayout()
        self.setLayout(self.grid)
        self.grid.addLayout(self.buttonBox, 0, 0)
        self.grid.addLayout(self.sliderRBGrid, 1, 0)
        self.addGraphToGrid()
        self.addPendulumToGrid()
        self.initActions()

    def addGraphToGrid(self):
        self.penGraGrid.addWidget(self.graph, 0, 0)
        self.grid.addLayout(self.penGraGrid, 3, 0)

    def addPendulumToGrid(self):
        self.penGraGrid.addWidget(self.pendulum, 0, 1)
        self.grid.addLayout(self.penGraGrid, 3, 0)
```

```
    def initConnections(self):
        # This function connects all the signals to the slots that
need
        # them. All the buttons are connected to their desired
functions
        # when clicked, all the sliders are connected to the desired
        # functions when they are chagned.
        self.comms.pendulumTick.setInterval(self.pendulum.h*1000)

        self.comms.pendulumTick.timeout.connect(self.pendulum.move)
        self.pendulum.attributeSignal.connect(self.graph.step)

        self.th1Slider.valueChanged.connect(self.setTh1FromSlider)
        self.th2Slider.valueChanged.connect(self.setTh2FromSlider)
        self.om1Slider.valueChanged.connect(self.setOm1FromSlider)
        self.om2Slider.valueChanged.connect(self.setOm2FromSlider)

self.massSlider.valueChanged.connect(self.setMassRatioFromSlider)

self.lengthSlider.valueChanged.connect(self.setLengthRatioFromSlider
)
        self.gSlider.valueChanged.connect(self.setGfromSlider)
        self.hSlider.valueChanged.connect(self.setHfromSlider)

        self.RBXGroup.buttonClicked[int].connect(self.setGraphXVar)
        self.RBYGroup.buttonClicked[int].connect(self.setGraphYVar)


self.startButton.clicked.connect(self.comms.pendulumTick.start)

self.stopButton.clicked.connect(self.comms.pendulumTick.stop)
        self.stepButton.clicked.connect(self.pendulum.step)

self.stateSaveButton.clicked.connect(self.pendulum.saveState)
        self.stateLoadButton.clicked.connect(self.pendulumLoadState)
        self.pictureSaveButton.clicked.connect(self.graph.export)
        self.pictureSaveButton.clicked.connect(self.pendulum.export)
        self.resetButton.clicked.connect(self.resetWidgets)
        self.defaultsButton.clicked.connect(self.setDefaults)
        self.helpButton.clicked.connect(self.showHelp)

    def removeConnections(self):
        # This disconnects all the connections made in
initConnections, so that
        # when the widgits are recreated, there aren't stray
connections
        # causing unwanted results.
        self.pendulum.attributeSignal.disconnect(self.graph.step)

self.comms.pendulumTick.timeout.disconnect(self.pendulum.move)


self.th1Slider.valueChanged.disconnect(self.setTh1FromSlider)

self.th2Slider.valueChanged.disconnect(self.setTh2FromSlider)
```

```
self.om1Slider.valueChanged.disconnect(self.setOm1FromSlider)

self.om2Slider.valueChanged.disconnect(self.setOm2FromSlider)

self.massSlider.valueChanged.disconnect(self.setMassRatioFromSlider)

self.lengthSlider.valueChanged.disconnect(self.setLengthRatioFromSli
der)
        self.gSlider.valueChanged.disconnect(self.setGfromSlider)
        self.hSlider.valueChanged.disconnect(self.setHfromSlider)


self.RBXGroup.buttonClicked[int].disconnect(self.setGraphXVar)

self.RBYGroup.buttonClicked[int].disconnect(self.setGraphYVar)


self.startButton.clicked.disconnect(self.comms.pendulumTick.start)

self.stopButton.clicked.disconnect(self.comms.pendulumTick.stop)
        self.stepButton.clicked.disconnect(self.pendulum.step)

self.stateSaveButton.clicked.disconnect(self.pendulum.saveState)

self.stateLoadButton.clicked.disconnect(self.pendulumLoadState)
        self.pictureSaveButton.clicked.disconnect(self.graph.export)

self.pictureSaveButton.clicked.disconnect(self.pendulum.export)
        self.resetButton.clicked.disconnect(self.resetWidgets)
        self.defaultsButton.clicked.disconnect(self.setDefaults)
        self.helpButton.clicked.disconnect(self.showHelp)

    def initActions(self):
        self.exitAction = QtGui.QAction('&Exit', self)
        self.exitAction.setShortcut('Ctrl+Q')
        self.exitAction.triggered.connect(QtGui.qApp.quit)
        self.addAction(self.exitAction)

# This creates the application object, creates the main window, then
# executes the application
def main():
    app = QtGui.QApplication(sys.argv)
    win = DpWindow()
    app.exec_()

# This checks to make sure the program has been started rather than
imported
if __name__ == "__main__":
    main()
```

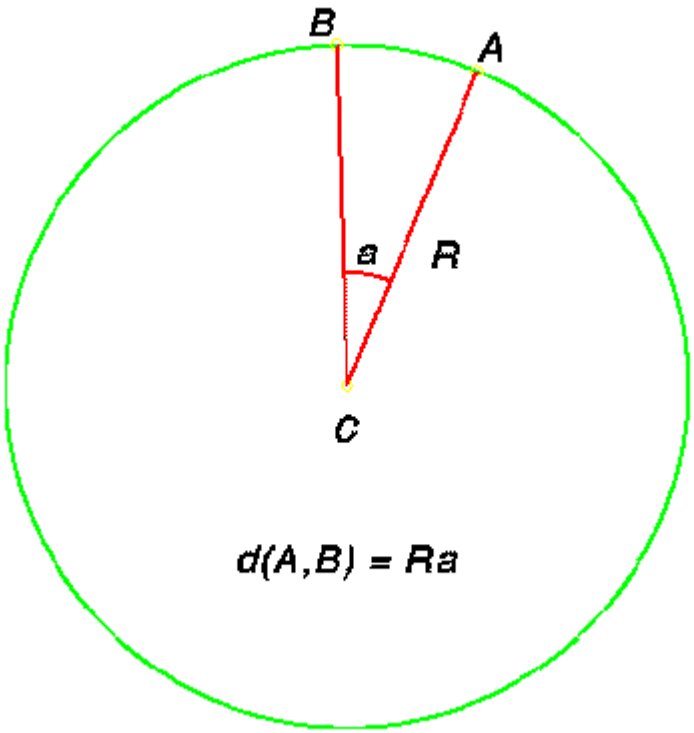
Appendix II: Radians

Here is an explanation of radian measure taken directly from http://goo.gl/k5AKh6.

Degree/Radian Circle

In everyone's experience it is usual to measure angles in degrees. We learn early in childhood that there are 360 degrees in a circle, that there are 90 degrees in a right angle, and that the angle of an equilateral triangle contains 60 degrees. On the other hand, to scientists, engineers, and mathematicians it is usual to measure angles in radians.

The size of a radian is determined by the requirement that there are 2π radians in a circle. Thus 2π radians equals 360 degrees. This means that 1 radian = $180/\pi$ degrees, and 1 degree = $\pi/180$ radians.



The reason for this is that so many formulas become much easier to write and to understand when radians are used to measure angles. A very good example is provided by the formula for the length of a circular arc. If *A* and *B* are two points on a circle of radius R and center C, then the length of the arc

of the circle connecting them is given by

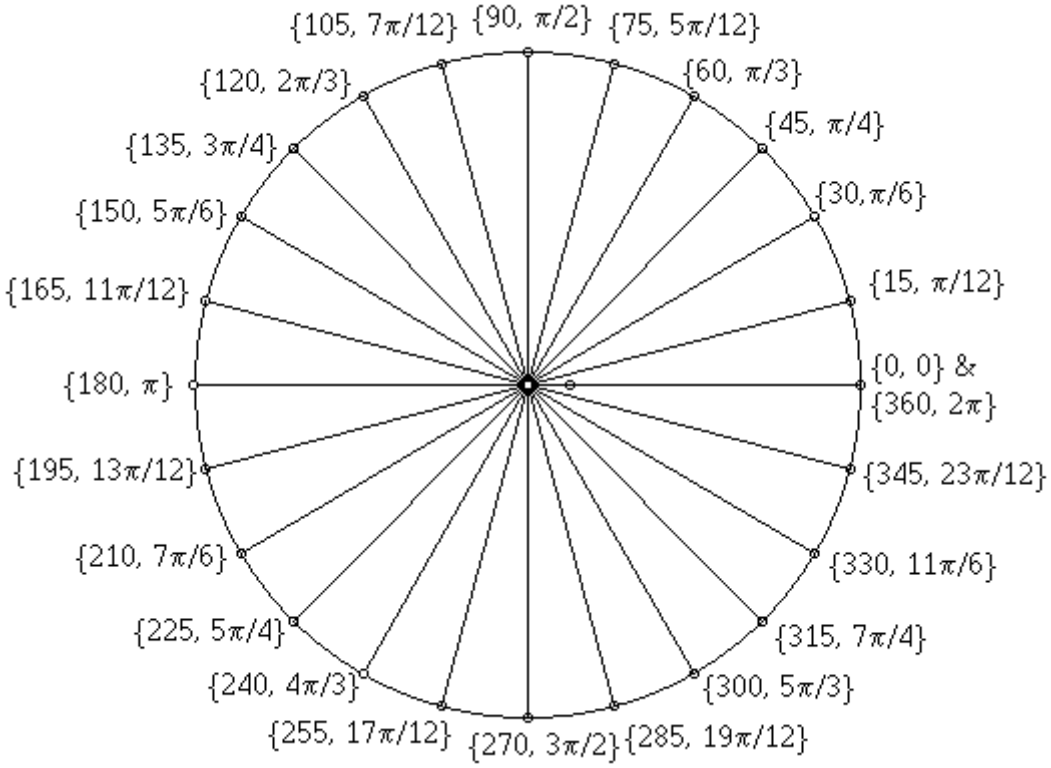
$$d(A,B) = R\,a,$$

where R is the radius of the sphere, and a is the angle ACB measured in radians. If we measure the angle in degrees, then the formula is

$$d(A,B) = R\,a\pi/180,$$

These formulas can be checked by noticing that the arc length is proportional to the angle, and then checking the formula for the full circle, i.e., when a = 2π radians (or 360 degrees).

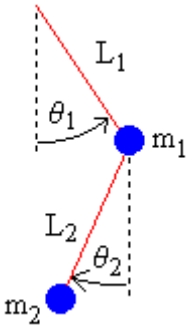
The figure below gives the relationship between degrees and radians for the most common angles in the unit circle measured in the counterclockwise direction from the point to the right of the vertex. The form of the ordered pair is {degree measure, radian measure}



Appendix III: Double pendulum proof

This is a similar derivation to mine from http://goo.gl/CLj9H.

Kinematics of the Double Pendulum



Kinematics means the relations of the parts of the device, without regard to forces. In kinematics we are only trying to find expressions for the position, velocity, & acceleration in terms of the variables that specify the state of the device.

- x = horizontal position of pendulum mass
- y = vertical position of pendulum mass
- ϑ = angle of pendulum (0 = vertical downwards, counter-clockwise is positive)
- L = length of rod (constant)

We place the origin at the pivot point of the upper pendulum. We regard y as increasing upwards. We indicate the upper pendulum by subscript 1, and the lower by subscript 2. Begin by using simple trigonometry to write expressions for the positions x_1, y_1, x_2, y_2 in terms of the angles ϑ_1, ϑ_2.$x_1 = L_1 \sin \vartheta_1 y_1 = -L_1 \cos \vartheta_1 x_2 = x_1 + L_2 \sin \vartheta_2 y_2 = y_1 - L_2 \cos \vartheta_2$The velocity is the derivative with

respect to time of the position.$x_1' = \vartheta_1' L_1 \cos \vartheta_1 y_1' = \vartheta_1' L_1 \sin \vartheta_1 x_2' = x_1' + \vartheta_2' L_2 \cos \vartheta_2 y_2' = y_1' + \vartheta_2' L_2 \sin \vartheta_2$The acceleration is the second derivative.

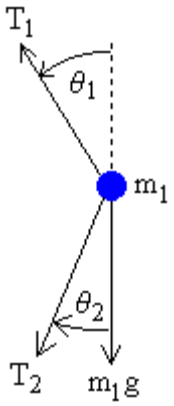
$$x_1'' = -\vartheta_1'^2 L_1 \sin \vartheta_1 + \vartheta_1'' L_1 \cos \vartheta_1$$

$$y_1'' = \vartheta_1'^2 L_1 \cos \vartheta_1 + \vartheta_1'' L_1 \sin \vartheta_1$$

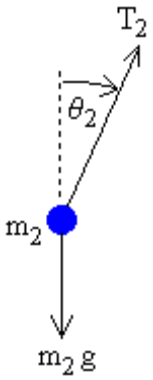
$$x_2'' = x_1'' - \vartheta_2'^2 L_2 \sin \vartheta_2 + \vartheta_2'' L_2 \cos \vartheta_2$$

$$y_2'' = y_1'' + \vartheta_2'^2 L_2 \cos \vartheta_2 + \vartheta_2'' L_2 \sin \vartheta_2$$

Forces in the Double Pendulum



upper mass



lower mass

We treat the two pendulum masses as point particles. Begin by drawing the free body diagram for the upper mass and writing an expression for the net force acting on it. Define these variables:

- T = tension in the rod

- m = mass of pendulum
- g = gravitational constant

The forces on the upper pendulum mass are the tension in the upper rod T_1, the tension in the lower rod T_2, and gravity $-m_1 g$. We write separate equations for the horizontal and vertical forces, since they can be treated independently. The net force on the mass is the sum of these. Here we show the net force and use Newton's law $F = m a$.

$$m_1 x_1'' = -T_1 \sin \vartheta_1 + T_2 \sin \vartheta_2$$

$$m_1 y_1'' = T_1 \cos \vartheta_1 - T_2 \cos \vartheta_2 - m_1 g$$

For the lower pendulum, the forces are the tension in the lower rod T_2, and gravity $-m_2 g$.

$$m_2 x_2'' = -T_2 \sin \vartheta_2$$

$$m_2 y_2'' = T_2 \cos \vartheta_2 - m_2 g$$

In relating these equations to the diagrams, keep in mind that in the example diagram ϑ_1 is positive and ϑ_2 is negative, because of the convention that a counter-clockwise angle is positive.

Direct Method for Finding Equations of Motion

Now we do some algebraic manipulations with the goal of finding expressions for ϑ_1'', ϑ_2'' in terms of ϑ_1, ϑ_1', ϑ_2, ϑ_2'. Begin by solving equations (7), (8) for $T_2 \sin \vartheta_2$ and $T_2 \cos \vartheta_2$ and then substituting into equations (5) and (6).

$$m_1 x_1'' = -T_1 \sin \vartheta_1 - m_2 x_2''$$

$$m_1 y_1'' = T_1 \cos \vartheta_1 - m_2 y_2'' - m_2 g - m_1 g$$

Multiply equation (9) by $\cos \vartheta_1$ and equation (10) by $\sin \vartheta_1$ and rearrange to get

$$T_1 \sin \vartheta_1 \cos \vartheta_1 = -\cos \vartheta_1 (m_1 x_1'' + m_2 x_2'')$$

$$T_1 \sin \vartheta_1 \cos \vartheta_1 = \sin \vartheta_1 (m_1 y_1'' + m_2 y_2'' + m_2 g + m_1 g)$$

This leads to the equation

$$\sin\vartheta_1\,(m_1\,y_1'' + m_2\,y_2'' + m_2\,g + m_1\,g) = -\cos\vartheta_1\,(m_1\,x_1'' + m_2\,x_2'')$$

Next, multiply equation (7) by $\cos\vartheta_2$ and equation (8) by $\sin\vartheta_2$ and rearrange to get

$$T_2\sin\vartheta_2\cos\vartheta_2 = -\cos\vartheta_2\,(m_2\,x_2'')$$

$$T_2\sin\vartheta_2\cos\vartheta_2 = \sin\vartheta_2\,(m_2\,y_2'' + m_2\,g)$$

which leads to

$$\sin\vartheta_2\,(m_2\,y_2'' + m_2\,g) = -\cos\vartheta_2\,(m_2\,x_2'')$$

Next we need to use a program such as Mathematica to solve equations (13) and (16) for ϑ_1'', ϑ_2'' in terms of ϑ_1, ϑ_1', ϑ_2, ϑ_2'. Note that we also include the definitions given by equations (1-4), so that we have 2 equations (13, 16) and 2 unknowns (ϑ_1'', ϑ_2''). The result is somewhat complicated, but is easy enough to program into the computer.

$$\vartheta_1'' = \frac{-g\,(2\,m_1 + m_2)\sin\vartheta_1 - m_2\,g\sin(\vartheta_1 - 2\,\vartheta_2) - 2\sin(\vartheta_1 - \vartheta_2)\,m_2\,(\vartheta_2'^2\,L_2 + \vartheta_1'^2\,L_1\cos(\vartheta_1 - \vartheta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\vartheta_1 - 2\,\vartheta_2))}$$

$$\vartheta_2'' = \frac{2\sin(\vartheta_1 - \vartheta_2)\,(\vartheta_1'^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\vartheta_1 + \vartheta_2'^2\,L_2\,m_2\cos(\vartheta_1 - \vartheta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\vartheta_1 - 2\,\vartheta_2))}$$

These are the equations of motion for the double pendulum.

Numerical Solution

The above equations are now close to the form needed for the Runge-Kutta method. The final step is convert these two 2nd order equations into four 1st order equations. Define the first derivatives as separate variables:

- ω_1 = angular velocity of top rod
- ω_2 = angular velocity of bottom rod

Then we can write the four 1st order equations:$\vartheta_1' = \omega_1\vartheta_2' = \omega_2$

$$\omega_1' = \frac{-g\,(2\,m_1 + m_2)\sin\vartheta_1 - m_2\,g\sin(\vartheta_1 - 2\,\vartheta_2) - 2\sin(\vartheta_1 - \vartheta_2)\,m_2\,(\omega_2^2\,L_2 + \omega_1^2\,L_1\cos(\vartheta_1 - \vartheta_2))}{L_1\,(2\,m_1 + m_2 - m_2\cos(2\,\vartheta_1 - 2\,\vartheta_2))}$$

$$\omega_2' = \frac{2\sin(\vartheta_1 - \vartheta_2)\,(\omega_1^2\,L_1\,(m_1 + m_2) + g(m_1 + m_2)\cos\vartheta_1 + \omega_2^2\,L_2\,m_2\cos(\vartheta_1 - \vartheta_2))}{L_2\,(2\,m_1 + m_2 - m_2\cos(2\,\vartheta_1 - 2\,\vartheta_2))}$$

This is now exactly the form needed to plug in to the Runge-Kutta method for numerical solution of the system.

Appendix IV: Runge-Kutta explanation

Here is a similar explanation of the Runge-Kutta technique found at http://goo.gl/nBPyo.

MyPhysicsLab – Runge-Kutta Algorithm

The Runge-Kutta algorithm is *the* magic formula behind most of the physics simulations shown on this web site. The Runge-Kutta algorithm lets us solve a differential equation numerically (that is, approximately); it is known to be very accurate and well-behaved for a wide range of problems.

Consider the single variable problem $x' = f(t, x)$ with initial condition $x(0) = x_0$. Suppose that x_n is the value of the variable at time t_n. The Runge-Kutta formula takes x_n and t_n and calculates an approximation for x_{n+1} at a brief time later, t_n+h. It uses a weighted average of approximated values of $f(t, x)$ at several times within the interval (t_n, t_n+h). The formula is given by $x_{n+1} = x_n + {}^{h}\!/_{6} (a + 2b + 2c + d)$ where

$a = f(t_n, x_n)$

$b = f(t_n + {}^{h}\!/_{2}, x_n + {}^{h}\!/_{2}\, a)$

$c = f(t_n + {}^{h}\!/_{2}, x_n + {}^{h}\!/_{2}\, b)$

$d = f(t_n + h, x_n + h\, c)$

To run the simulation, we start with x_0 and find x_1 using the formula above. Then we plug in x_1 to find x_2 and so on.

With multiple variables, the Runge-Kutta algorithm looks similar to the above equations, except that the variables become vectors.

Multi-variable Runge-Kutta Algorithm

The Runge-Kutta Algorithm is fairly simple, but to describe it precisely we need to develop some notation. Suppose there are m variables $x_1, x_2, \ldots, x_m$ each of which vary over time. For example, in the single spring simulation, x_1 is position, x_2 is velocity. Suppose further that there are m differential equations for these m variables $x_1' = f_1(x_1, x_2, \ldots, x_m)$

$x_2' = f_2(x_1, x_2, ..., x_m)$

...

$x_m' = f_m(x_1, x_2, ..., x_m)$

Notice there are no derivatives on the right hand side of any of those equations, and there are only first derivatives on the left hand side. These equations can be summarized in vector form as$\mathbf{x}' = \mathbf{f}(\mathbf{x})$

where $\mathbf{x} = (x_1, x_2, ..., x_m)$ and we allow some loose "vector of functions" concept where $\mathbf{f} = (f_1, f_2, ..., f_m)$. Next we label our time states $\mathbf{x}_n$, $\mathbf{x}_{n+1}$which are separated by time interval of length h. That is, $\mathbf{x}_n$ is the value of the m variables at time t_n.

And $x_{1,n}$ is the value of the first variable x_1 at timet_n.$\mathbf{x}_n = (x_{1,n}, x_{2,n}, ..., x_{m,n})$

$\mathbf{x}_{n+1} = (x_{1,n+1}, x_{2,n+1}, ..., x_{m,n+1})$

Suppose we have the state of the simulation at time t_n as $\mathbf{x}_n$. To compute the state a short time h later and put the results into $\mathbf{x}_{n+1}$, the Runge-Kutta algorithm does the following:$\mathbf{a}_n = \mathbf{f}(\mathbf{x}_n)$

$\mathbf{b}_n = \mathbf{f}(\mathbf{x}_n + {}^{h}\!/_{2}\, \mathbf{a}_n)$

$\mathbf{c}_n = \mathbf{f}(\mathbf{x}_n + {}^{h}\!/_{2}\, \mathbf{b}_n)$

$\mathbf{d}_n = \mathbf{f}(\mathbf{x}_n + h\, \mathbf{c}_n)$

$\mathbf{x}_{n+1} = \mathbf{x}_n + {}^{h}\!/_{6}\, (\mathbf{a}_n + 2\, \mathbf{b}_n + 2\, \mathbf{c}_n + \mathbf{d}_n)$

The new vector $\mathbf{x}_{n+1}$ gives you the state of the simulation after the small time h has elapsed. To spell out the above in more detail, we can drop the vector notation and write the Runge-Kutta algorithm like this:$a_{j,n} = f_j(x_{1,n}, \quad x_{2,n}, \ldots, x_{m,n})$

$b_{j,n} = f_j\big((x_{1,n} + {}^{h}\!/_{2}\, a_{1,n}), \quad (x_{2,n} + {}^{h}\!/_{2}\, a_{2,n}), \ldots, (x_{m,n} + {}^{h}\!/_{2}\, a_{m,n}) \big)$

$c_{j,n} = f_j\big((x_{1,n} + {}^{h}\!/_{2}\, b_{1,n}), \quad (x_{2,n} + {}^{h}\!/_{2}\, b_{2,n}), \ldots, (x_{m,n} + {}^{h}\!/_{2}\, b_{m,n}) \big)$

$d_{j,n} = f_j\big((x_{1,n} + h\, c_{1,n}), \quad (x_{2,n} + h\, c_{2,n}), \ldots, (x_{m,n} + h\, c_{m,n}) \big)$

$x_{j,n+1} = x_{j,n} + {}^{h}\!/_{6}\, (a_{j,n} + 2\, b_{j,n} + 2\, c_{j,n} + d_{j,n})$

The above equations are applied for each variable j=(1, ..., m) to get the full set of variables in the vector $\mathbf{x}_{n+1}$.

Time As A Variable

Most of the simulations shown on this website *do not* have differential equations that depend explicitly on time. That is, you won't see the variable t on the right-hand side of the differential equations. One simulation that *does* depend on time is the chaotic driven pendulum because the driving force (which applies the twist to the pendulum) varies over time according to cos(k t).

When time appears explicitly in the differential equations we can add a time variable t to the state vector **x**. Suppose we assign this role to the variable x_2. This new variable has the extremely simple differential equation $x_2' = 1$ That says that the rate of change of the variable x_2 is a constant. Since we are taking derivatives with respect to time we can also write the above equation as $x_2' = \frac{d}{dt} x_2 = 1$ This integrates very easily to give $x_2 = t$, which is what we wanted: time as a variable. Suppose that in the driven pendulum simulation we set up x_2 in this way. Then the driving force is given by $\cos(k\, x_2)$.

You may ask: *Why have time as a variable? We already know the value of t at each time step!* The Runge-Kutta algorithm works by averaging the predicted rates at various points in the time interval from t to $t+h$. Therefore, when the rates (differential equations) depend explictly on t, we also need to know the value of t at those points *within* the time interval. Putting time in as a variable makes for nicer cleaner computer code.

Time Not As A Variable

If you want to, you can avoid keeping time as an additional variable. The following is an equivalent formulation of the Runge-Kutta algorithm where time t is passed in as a variable to each function in $\mathbf{f}.\mathbf{a}_n = \mathbf{f}(t, \mathbf{x}_n)$
$\mathbf{b}_n = \mathbf{f}(t + h/2, \mathbf{x}_n + h/2\, \mathbf{a}_n)$
$\mathbf{c}_n = \mathbf{f}(t + h/2, \mathbf{x}_n + h/2\, \mathbf{b}_n)$
$\mathbf{d}_n = \mathbf{f}(t + h, \mathbf{x}_n + h\, \mathbf{c}_n)$
$\mathbf{x}_{n+1} = \mathbf{x}_n + h/6\, (\mathbf{a}_n + 2\, \mathbf{b}_n + 2\, \mathbf{c}_n + \mathbf{d}_n)$
This is completely equivalent to the formulation where time is kept as one of the variables in **x**. Whether you use this formulation or the earlier (cleaner) one is entirely up to you.