4-22-1996

Space Trajectory Simulation for the Internet

Michael Selby

Patrick Newton

Follow this and additional works at: https://louis.uah.edu/honors-capstones

Recommended Citation
Selby, Michael and Newton, Patrick, "Space Trajectory Simulation for the Internet" (1996). Honors Capstone Projects and Theses. 570.
https://louis.uah.edu/honors-capstones/570

This Thesis is brought to you for free and open access by the Honors College at LOUIS. It has been accepted for inclusion in Honors Capstone Projects and Theses by an authorized administrator of LOUIS.
Space Trajectory Simulation for the Internet

Michael Selby and Patrick Newton
MAE 496
Mechanical Engineering Applications/Honors Research Project

Submitted to:
Dr. Robert A. Frederick, Jr.
Assistant Professor
Department of Mechanical and Aerospace Engineering

and

Dr. Joyce Pettus
Honors Program Director

University of Alabama in Huntsville

April 22, 1996
Prologue

Rocket propulsion is a field that has been around since the start of the twentieth century beginning with the writings of K.E. Tsiolkovsky in the *Moscow Technical Review*. Robert Goddard in the United States and Wehrner von Braun in Germany did much of the original work to establish rocket technology. The development of the V-2 rocket in Germany in the 1940s led to the beginnings of the American and Russian space programs that are prevalent today. However, much of the knowledge about rocket propulsion is confined to those people who work in the space program and to major universities who do propulsion research. It would be very useful to find a way to disseminate some of the information about rocket propulsion to the general public, especially to young people considering a career in aerospace. The Internet could be used as a tool to help young people learn more about the field of rocket propulsion.

Introduction

This Honors Project represents one phase of a three phase project that will be completed over a period of one year. The ultimate goal of the project is to place rocket trajectory simulation software on the Internet. The software will be used to allow high school students to learn more about the field of rocket propulsion. The project participants are Dr. Bob Frederick, Jr. of the UAH Propulsion Research Center, Stone Engineering of Huntsville, and Kellie Miller of the Boeing Company. The choice was made to write the programs in the JAVA programming language in order to best accomplish the ultimate project goals.
JAVA is a relatively new programming language developed by Sun Microsystems, Inc. It was developed from the C++ language and has much of the same structure of C and C++. However, JAVA was simplified so that it could be run on many different platforms such as IBM PC's and Macintosh computers and could easily be used for World Wide Web applications. Using a JAVA enabled Web browser, anyone with World Wide Web access can run JAVA applets on their own machine, no matter what type of platform they have. Applets are JAVA applications are written specifically to run in Web pages.

One of the most basic ways to model a rocket in flight is by using a system of masses connected by springs. Each of the springs is assigned a spring constant, and each spring is used to force a mass away from the system. The release of the masses is used to represent fuel leaving the rocket. By applying the principles of conservation of momentum and conservation of energy, the velocities of both the mass that is released and the system of remaining masses can be determined. The goal of this project and phase one of the larger project is to develop and model this system in a Web page using the JAVA language. Figure 1 below shows the system of masses to be modeled.

Figure 1: Mass and Spring System
The system in Figure 1 will be modeled inside a Web page so that it can be accessed over the Internet. The students who access it will be able to input their own values for the weight of each mass and then see the calculated velocities of each of the masses. The students can then check to see if the payload mass reached a velocity high enough to achieve orbit. The program also shows a graph of the position of the payload as a function of time. Once the work on this phase of the project is complete, work will begin on the next two phases. Phase Two will involve a simple rocket flying straight up to a single point in space. Phase Three of the project will involve a rocket flying to meet a moving target which will require the rocket to be constantly changing its trajectory. This phase will require the students to input a variety of different parameters in order to determine if the rocket reaches its target.

**Method of Calculation**

One of the first steps in this project was to gain an understanding of rocket propulsion. To do this, a spreadsheet was set up to calculate the position, velocity, and acceleration of a simple sounding rocket. For the calculations, several assumptions were made. These assumptions included a flat earth, no drag acting on the rocket, and a one dimensional flight path. The simulation was carried out until the rocket reached its apogee, its maximum distance from the surface of the earth.

The next step of the project was to set up the system of masses and springs that would be modeled in the Web page. In order to determine the velocity of the blocks as
they were pushed by the springs, conservation of momentum and conservation of energy were applied to the motion of the blocks.

The total energy of the system must be conserved. Therefore, the potential energy in the compressed springs which is equal to

\[ PE_{spring} = \frac{1}{2} kx^2 \]

Equation 1. Potential Energy of the Spring

where

- \( k \) = spring constant (force/distance)
- \( x \) = compression distance (distance)

must be equal to the kinetic energy of the blocks after the spring is released. For our program, it was assumed that all the spring constants and compression distances were equal for each spring. These values were \( k = 400 \) lb/in and \( x = 4 \) in. Eventually, the user could be allowed to change these values for each of the springs.

In order to determine the absolute velocity of the blocks, the relative velocities between the blocks were calculated. If all the blocks are held stationary except for the one which is pushed off by the spring (Figure 2), then all of the energy in the spring must be transferred to that one block.

\[ \text{Mass 1} \]

Figure 2. One Mass Released from a Stationary Object

This is true since there is assumed to be no drag or friction on the blocks. If there was drag or friction, then some of the energy of the spring would be lost due to heating from
the drag and/or friction forces. Also, the velocities are assumed to be instantaneous and constant. Therefore,

\[ KE_{\text{block}} = PE_{\text{spring}} \]

Equation 2. Kinetic Equals Potential Energy

where

\[ KE_{\text{block}} = \frac{1}{2} mV^2 \]

Equation 3. Kinetic energy of the Block

with

\( m = \) mass of the block
\( V = \) velocity of the block.

Using equations 1, 2, and 3, the velocity of the block can be found to be

\[ V = \sqrt{\frac{k}{m}} x \]

Equation 4. Relative Velocity of the Single Block

By knowing the velocity of the block with respect to a stationary object, it is assumed that this same velocity will be the relative velocity between two non-stationary objects released by the same spring. In other words, if a person was standing on one of the blocks as the spring was released, the velocity of the other block from that person's point of view would be the same velocity as in equation 4. Therefore, the absolute velocity (that is velocity measured from a fixed reference, i.e. ground) of the single block would be equal to

\[ V_1 = V_{\text{sys}} - V_{\text{rel}} \]

Equation 5. Absolute Velocity of the Single Block

where

\( V_{\text{sys}} = \) absolute velocity of the rest of the blocks
\( V_{\text{rel}} = \) relative velocity between the moving blocks (Equation 4).

Note that if \( V_{\text{rel}} \) is greater than \( V_{\text{sys}} \), then \( V_1 \) will be negative in value. Since \( V_{\text{sys}} \) is assumed to be the positive velocity direction, the single block will be traveling in the
opposite direction from the rest of the system of blocks. As \( V_{sys1} \) continues to increase in value, it will eventually become greater than \( V_{rel} \) making \( V_1 \) a positive quantity. This means that the single block will actually be traveling in the same direction as the rest of the system.

In order to determine the absolute velocities of the blocks, conservation of momentum was used. Conservation of momentum gives the equation

\[
m_T V_T = m_1 V_1 + m_{sys1} V_{sys1} \]  

Equation 6. Momentum of the System

where the subscript ‘T’ represents the initial total system, the subscript ‘1’ represents the single moving block, and the subscript ‘sys1’ represents the rest of the system. Since all of the blocks are connected at the start, as one block is released, then all of the other blocks will travel together with the same velocity. Initially, the total momentum of the system is zero since the masses start from rest. By using equations 5 and 6, it is possible to solve for \( V_1 \) and \( V_{sys1} \). Once \( V_1 \) and \( V_{sys1} \) are calculated, the momentum equation and the absolute velocity equation are used again in the form of

\[
m_{sys1} V_{sys1} = m_2 V_2 + m_{sys2} V_{sys2} \]  

Equation 7. Momentum for System #2

\[
V_2 = V_{sys2} - V_{rel} \]  

Equation 8. Velocity of Mass #2

These equations allow for the calculation of \( V_2 \) and \( V_{sys2} \). Iterations are continued until the final velocity of the payload is calculated. These velocities are shown in Figure 3.
Figure 3. Velocity of Payload

Once the velocities of each of the systems are calculated, the position of the payload can be determined. However, in order to determine the positions, a time scale must be set. The assumption was made that each spring was released in ten second increments. Eventually, the times at which springs would be released could be controlled by the user. The position is then found by multiplying the velocity times the time. However, this must be done in steps since the velocity of the payload is changing. For example, in order to calculate the position of the payload after 16 seconds, a position must be calculated for the first ten seconds using \( V_{sys1} \) and then added to the position calculated for the last six seconds using \( V_{sys2} \). The position of the payload is shown in Figure 4.
These values were first calculated using an Excel spreadsheet which is shown in Appendix A. The spreadsheet was able to take inputs from the user and calculate velocity and position as a function of time.

Method for Programming

The JAVA language is a C++ based language. It is basically a cut down version of C++. However, it is also an object oriented language. This means that there is a structure of objects, classes, and methods where classes are the physical objects and methods are the functions which are performed on the objects.

Our first approach was to obtain programming material on JAVA. This included programming manuals and a JAVA compiler. As with C++, JAVA programs must be compiled into an executable file. This executable file is what is run when the program is executed. The first compiler obtained was a simple DOS based compiler placed on the Internet by Sun Microsystems, Inc. It required that the program script be written in a text
editor and then compiled using DOS commands. This type of process was slow and tedious considering we were learning the language as we programmed. With a DOS compiler, every line of the program is written from scratch. The compiler does nothing but debug and compile. So in order to have a window or edit box appear on the screen, all of the commands for drawing the box and handling the inputs and outputs must be programmed. This type of programming is generally reserved for experienced programmers for the particular type of language. Since JAVA is a new language, there are not many experienced users from which we could acquire programming support. This lead to the purchase of the second compiler.

The second JAVA compiler obtained was *Symantec Cafe* put out by the Symantec Corporation. This compiler is a windows based compiler which allows for the creation of generic forms. Many software development kits on the market today for C, C++, BASIC, and others are similar to *Symantec Cafe*. Forms are created which allow for the creation of text areas, control buttons, edit boxes, and list boxes. These items are placed inside the form using the mouse to drag and drop them into certain locations (see Figure 5). The code for these items and the whole generic structure of the program is then written by the software. Comparing this to the DOS method of programming, it is easy to see that the time savings would be enormous. Of course, these types of compilers cost more money than the free shareware versions of the DOS compilers, but their cost is justified with time savings.
Figure 5. Symantec Cafe JAVA Compiler

Once the generic form is set up, the programmer must then go into the program and write in the functions necessary to perform the operations. In other words, the compiler may generate the code for the form and the buttons, but it does not generate any code concerning what happens when a particular button is clicked by the user. This code must be developed by the programmer.

Symantec Cafe also aids in the development of the HTML code. HTML is the interface language of the Internet. All Web pages are written in HTML. The HTML code is simple text and can be written using any text editor. It does not have to be compiled like JAVA because Web browsers interpret the HTML into visual commands. The HTML code for the trajectory simulation is shown in Appendix B.
Results

The final web page containing the JAVA applet written for the project can be seen in Figure 6 below. This figure shows the simulation Web page inside of Netscape 2.0. The inputs which are taken from the user are the weights of the fuel masses and the payload mass. Once the ‘Calculate’ button is clicked by the user, the software goes through and solves the equations listed in the above section. The velocities of the payload are shown at the right side of the screen, while the distance traveled by the payload is shown in graphical form at the bottom of the screen. The code for the program is shown in Appendix C.

![Figure 6. Trajectory Simulation Web Page](image-url)
Conclusions

The goal of this project was to model a system of masses and springs inside a World Wide Web page using the JAVA programming language. This system represented a very basic model of a rocket. When someone accesses the page, they can input values for each of the masses, and the JAVA applet imbedded in the page will calculate the velocity of the payload mass after each one of the masses representing the fuel is released. This goal may seem very modest for a project of this nature. However, because JAVA is a new programming language and because the Internet is ever-evolving, there was quite a bit to do in setting up the Web page to model the system of masses. Meeting these goals meant learning about the JAVA programming language as we wrote the applications, as well as learning about HTML programming so that we could create a JAVA enabled Web page for the Internet.

The information gathered in setting up this demonstration will be used to reach the ultimate goals of the larger project. These include simulating a rocket that intercepts a moving target and providing animation of the rocket in flight. By accomplishing these goals we hope that we are able to help more people learn about the field of rocket propulsion.
List of Sources

Primary Sources:


Symantec Cafe. Developed by Symantec Corp. 1996

Secondary Sources:


Appendix A

Calculations in Spreadsheet
### Trajectory Calculations

#### Define and Input Variables

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Gravity =</td>
<td>386.4 in/sec²</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Masses</th>
<th>Spring Rates</th>
<th>Spring Compression</th>
<th>Time of Release of Springs</th>
</tr>
</thead>
<tbody>
<tr>
<td>m₁ = 200 lbs</td>
<td>k₁ = 400 lb/in</td>
<td>x₁ = 4 in</td>
<td>t₁ = 0 sec</td>
</tr>
<tr>
<td>m₂ = 200 lbs</td>
<td>k₂ = 400 lb/in</td>
<td>x₂ = 4 in</td>
<td>t₂ = 10 sec</td>
</tr>
<tr>
<td>m₃ = 200 lbs</td>
<td>k₃ = 400 lb/in</td>
<td>x₃ = 4 in</td>
<td>t₃ = 10 sec</td>
</tr>
<tr>
<td>Payload = 200 lbs</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

#### Calculations

<table>
<thead>
<tr>
<th>Msₙext (lb)</th>
<th>Msₜₙext (lb)</th>
<th>Vsₙext (in/sec)</th>
<th>Vsₜₙext (in/sec)</th>
<th>Vexhaust (in/sec)</th>
<th>Momentum (lb*in/sec)</th>
<th>Relative Velocity (in/sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>600</td>
<td>800</td>
<td>27.80</td>
<td>0.00</td>
<td>v₁ = -83.40</td>
<td>0.00</td>
<td>Vᵣ₁ = 111.20</td>
</tr>
<tr>
<td>400</td>
<td>600</td>
<td>64.86</td>
<td>27.80</td>
<td>v₂ = -46.33</td>
<td>0.00</td>
<td>Vᵣ₂ = 111.20</td>
</tr>
<tr>
<td>200</td>
<td>400</td>
<td>120.16</td>
<td>64.86</td>
<td>v₃ = 9.27</td>
<td>0.00</td>
<td>Vᵣ₃ = 111.20</td>
</tr>
<tr>
<td>0</td>
<td>200</td>
<td></td>
<td>vp = 120.46</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A-2
### Trajectory Calculations

#### Position vs. Time

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Payload (in)</th>
<th>Mass #3 (in)</th>
<th>Mass #2 (in)</th>
<th>Mass #1 (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>55.60</td>
<td>55.60</td>
<td>55.60</td>
<td>-166.80</td>
</tr>
<tr>
<td>4</td>
<td>111.20</td>
<td>111.20</td>
<td>111.20</td>
<td>-333.59</td>
</tr>
<tr>
<td>6</td>
<td>166.80</td>
<td>166.80</td>
<td>166.80</td>
<td>-500.39</td>
</tr>
<tr>
<td>8</td>
<td>222.39</td>
<td>222.39</td>
<td>222.39</td>
<td>-667.18</td>
</tr>
<tr>
<td>10</td>
<td>277.99</td>
<td>277.99</td>
<td>277.99</td>
<td>-833.98</td>
</tr>
<tr>
<td>12</td>
<td>333.91</td>
<td>333.91</td>
<td>333.91</td>
<td>-1000.77</td>
</tr>
<tr>
<td>14</td>
<td>383.64</td>
<td>383.64</td>
<td>383.64</td>
<td>-1167.57</td>
</tr>
<tr>
<td>16</td>
<td>433.37</td>
<td>433.37</td>
<td>433.37</td>
<td>-1334.37</td>
</tr>
<tr>
<td>18</td>
<td>483.10</td>
<td>483.10</td>
<td>483.10</td>
<td>-1501.16</td>
</tr>
<tr>
<td>20</td>
<td>532.82</td>
<td>532.82</td>
<td>532.82</td>
<td>-1667.96</td>
</tr>
<tr>
<td>22</td>
<td>582.55</td>
<td>582.55</td>
<td>582.55</td>
<td>-1834.75</td>
</tr>
<tr>
<td>24</td>
<td>632.28</td>
<td>632.28</td>
<td>632.28</td>
<td>-2001.55</td>
</tr>
<tr>
<td>26</td>
<td>681.90</td>
<td>681.90</td>
<td>681.90</td>
<td>-2168.34</td>
</tr>
<tr>
<td>28</td>
<td>731.63</td>
<td>731.63</td>
<td>731.63</td>
<td>-2335.14</td>
</tr>
<tr>
<td>30</td>
<td>781.36</td>
<td>781.36</td>
<td>781.36</td>
<td>-2501.94</td>
</tr>
<tr>
<td>32</td>
<td>831.09</td>
<td>831.09</td>
<td>831.09</td>
<td>-2668.74</td>
</tr>
<tr>
<td>34</td>
<td>880.82</td>
<td>880.82</td>
<td>880.82</td>
<td>-2835.53</td>
</tr>
<tr>
<td>36</td>
<td>930.55</td>
<td>930.55</td>
<td>930.55</td>
<td>-3002.32</td>
</tr>
<tr>
<td>38</td>
<td>980.28</td>
<td>980.28</td>
<td>980.28</td>
<td>-3169.12</td>
</tr>
<tr>
<td>40</td>
<td>1030.01</td>
<td>1030.01</td>
<td>1030.01</td>
<td>-3335.91</td>
</tr>
<tr>
<td>42</td>
<td>1079.74</td>
<td>1079.74</td>
<td>1079.74</td>
<td>-3502.71</td>
</tr>
<tr>
<td>44</td>
<td>1129.47</td>
<td>1129.47</td>
<td>1129.47</td>
<td>-3669.51</td>
</tr>
<tr>
<td>46</td>
<td>1179.20</td>
<td>1179.20</td>
<td>1179.20</td>
<td>-3836.30</td>
</tr>
<tr>
<td>48</td>
<td>1228.93</td>
<td>1228.93</td>
<td>1228.93</td>
<td>-4003.10</td>
</tr>
<tr>
<td>50</td>
<td>1278.66</td>
<td>1278.66</td>
<td>1278.66</td>
<td>-4169.89</td>
</tr>
</tbody>
</table>

#### Distance of Payload at Particular Time

<table>
<thead>
<tr>
<th>Time (sec)</th>
<th>Position (in)</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>277.99</td>
</tr>
</tbody>
</table>

#### Position of Payload vs. Time

![Graph showing the position of the payload over time]

#### Velocity of Payload vs. Time

![Graph showing the velocity of the payload over time]
Appendix B

HTML Code
Welcome to The Rocket Page! This page is intended to introduce people to the wonderful world of rocket motion. The following demonstration is a simplified model of a rocket. This model uses masses and springs to simulate the flow of fuel from a rocket. As individual masses are pushed by the compressed springs in one direction, the other masses are forced to go in the opposite direction. This is the same way in which a rocket operates. As fuel is burned and forced out the back through the nozzle, the rest of the rocket goes in the opposite direction:

Instructions: To use the following simulation, type in the weight of the masses you want to use for your rocket. The first five are considered fuel. The last mass is for the payload. After all five of the "fuel" masses are released, the payload will be the last one left and will have the highest velocity. Your job is to decided how to arrange the first five masses so that the payload will have its highest possible velocity.

For example, try setting your payload weight to 200 lbs and your total fuel weight to 1000 lbs. Now, try to decide the value of each of the first five masses which will give the highest final velocity to the payload mass. Remember, the sum total of the first five masses must be 1000 lbs.

The only equations which are necessary for determining the velocity of the masses at each interval are conservation of momentum and conservation of energy. The spring constant of all the springs is 400 lb/in with a compression of 4 inches. The masses are assumed to be moving on a frictionless horizontal plane.
This class is a basic extension of the Applet class. It would generally be used as the main class with a Java browser or the AppletViewer. But an instance can be added to a subclass of Container. You can add controls to rocket with Cafe Studio. (Menus can be added only to subclasses of Frame.)

```java
import java.applet.Applet;
import java.awt.*;
import java.applet.*;
import java.util.Vector;
import java.lang.*;
import java.io.*;
public class rocket extends Applet {

    Image bufferImage; //The graphing image object
    Graphics bufferGC; //The graphing context
    GraphTools gt; //A source of 'tools' to draw graphs
    int graphWidth=600; //The width of the graph area
    int graphHeight=300; //The height of the graph area
    Vector v; //An object to hold graph data

    Panel buttonPanel, b1Panel, b2Panel, b3Panel;
    Button b1, b2, b3; /*
    Dimension d; //Holds applet size information

    public void init() {
        super.init();

        //Cue the graph size to the height specified in the HTML
d = size();
        /*graphHeight = d.height - 25;
        graphWidth = graphHeight;*/

        //Create the drawing context for the graph information
        bufferImage = createImage(graphWidth, graphHeight);
        bufferGC = bufferImage.getGraphics();

        //The main GraphTools object with associated drawing context
        gt = new GraphTools(bufferGC);

        //INIT_CONTROLS
        setLayout(null);
        resize(693,650);
        num1 = new TextField(10);
        add(num1);
        num1 reshape(126,75,84,23);
        num2 = new TextField(10);
    }
```
add(num2);
num2.reshape(126,113,84,22);
num3=new TextField(10);
add(num3);
num3.reshape(126,150,84,23);
num4=new TextField(10);
add(num4);
num4.reshape(126,188,84,22);
num5=new TextField(10);
add(num5);
num5.reshape(126,225,84,23);
num6=new TextField(10);
add(num6);
num6.reshape(126,278,84,22);
label2=new Label("Mass #1");
add(label2);
label2.reshape(21,83,98,15);
label3=new Label("Mass #2");
add(label3);
label3.reshape(21,120,98,15);
label4=new Label("Mass #3");
add(label4);
label4.reshape(21,158,98,15);
label5=new Label("Mass #4");
add(label5);
label5.reshape(21,195,98,15);
label6=new Label("Mass #5");
add(label6);
label6.reshape(21,233,98,15);
label7=new Label("Payload Mass");
add(label7);
label7.reshape(21,285,98,15);
out1=new TextField(10);
out1.disable();
add(out1);
out1.reshape(546,75,84,23);
out2=new TextField(10);
out2.disable();
add(out2);
out2.reshape(546,113,84,22);
out3=new TextField(10);
out3.disable();
add(out3);
out3.reshape(546,150,84,23);
out4=new TextField(10);
out4.disable();
add(out4);
out4.reshape(546,188,84,22);
out5=new TextField(10);
out5.disable();
add(out5);
out5.reshape(546,278,84,22);
label8=new Label("After Release of Spring #1");
add(label8);
Using the tools and services provided by the GraphTools object, set up the starting graph image. Repaint() to display this image.

```
public void start()
{
    gt.drawAxes(graphWidth, graphHeight);
    gt.makeHashMarks(25);
    gt.makeTitle("Position of Payload vs Time");
    repaint();
}
```

**
* Draw the contents of the graph image (which has been manipulated be the GraphTools object).
*

```
public void paint(Graphics g)
{
/* b1.resize(65, 25);
 b2.resize(65, 25);
 b3.resize(65, 25);*/

    g.drawImage(bufferImage, 50, 350, this);
}
```

```
public boolean handleEvent(Event event) {
    if (event.id == Event.ACTIONEVENT && event.target == CalcButton) {
```
Calculate();
return true;
}

return super.handleEvent(event);
}

public void ShowLoanCalc() {
    /*
    GraphIt myGraphIt;
    myGraphIt = new GraphIt();
    myGraphIt.show();*/
}

public void Calculate() {
    double mass1, mass2, mass3, mass4, mass5, payload, masstotal;
    double vrel1, vrel2, vrel3, vrel4, vrel5;
    double vsys1, vsys2, vsys3, vsys4, vpayload;
    Float possys;
    String tempstring = null;
    Vector store = new Vector();
    Vector posvector = new Vector();
    double position;
    int time = 1;

    try {
        mass1 = (Double.valueOf(num1.getText()).doubleValue());
        mass2 = (Double.valueOf(num2.getText()).doubleValue());
        mass3 = (Double.valueOf(num3.getText()).doubleValue());
        mass4 = (Double.valueOf(num4.getText()).doubleValue());
        mass5 = (Double.valueOf(num5.getText()).doubleValue());
        payload = (Double.valueOf(num6.getText()).doubleValue());
    }
    catch (NumberFormatException e) {return;}

    if (mass1 < 0 || mass2 < 0 || mass3 < 0 || mass4 < 0 || mass5 < 0 || payload < 0) return;

    /*IntDec = InterestRate / (12.0 * 100.0);
    NumMonths = LengthLoan * 12.0;
    MonthlyPay = Principal * (IntDec / (1.0 - Math.pow((1.0 + IntDec), -NumMonths)));*/
    masstotal = mass1 + mass2 + mass3 + mass4 + mass5 + payload;
    vrel1 = Math.pow(400/mass1*386.4, .5)*4;
    vrel2 = Math.pow(400/mass1*386.4, .5)*4;
    vrel3 = Math.pow(400/mass1*386.4, .5)*4;
    vrel4 = Math.pow(400/mass1*386.4, .5)*4;
    vrel5 = Math.pow(400/mass1*386.4, .5)*4;
    vsys1 = mass1/masstotal*vrel1;
    out1.setText(String.valueOf(vsys1));
    vsys2 = ((masstotal-mass1)*vsys1 + mass2*vrel2)/(masstotal - mass1);
    out2.setText(String.valueOf(vsys2));
    vsys3 = ((masstotal-mass1-mass2)*vsys2 + (mass3*vrel3))/(masstotal - mass1-mass2);
out3.setText(String.valueOf(vsys3));

vsys4 = ((masstotal-mass1-mass2-mass3)*vsys3 + (mass4*vrel4))/(masstotal - mass1-mass2-mass3);
out4.setText(String.valueOf(vsys4));

vpayload = ((masstotal-mass1-mass2-mass3-mass4)*vsys4 + (mass5*vrel5))/(masstotal - mass1-
masstotal-mass2-mass3-mass4);
out5.setText(String.valueOf(vpayload));

/
*/
*/
tempstring.toString(vsysl);*/
*/
tempstring = String.valueOf(vsys1);
velsys = Float.valueOf(tempstring);
store.addElement(velsys);
tempstring = String.valueOf(vsys2);
velsys = Float.valueOf(tempstring);
store.addElement(velsys);*/

while (time < 50) {
   if (time < 11) {
      position = vsys1 * time;
      tempstring = String.valueOf(position);
      possys = Float.valueOf(tempstring);
      posvector.addElement(possys);
   }
   if (time < 21 && time > 10) {
      position = vsys1 * 10 + vsys2*(time - 10);
      tempstring = String.valueOf(position);
      possys = Float.valueOf(tempstring);
      posvector.addElement(possys);
   }
   if (time < 31 && time > 20) {
      position = (vsys1 + vsys2) * 10 + vsys3*(time - 20);
      tempstring = String.valueOf(position);
      possys = Float.valueOf(tempstring);
      posvector.addElement(possys);
   }
   if (time < 41 && time > 30) {
      position = (vsys1 + vsys2 + vsys3) * 10 + vsys4*(time - 30);
      tempstring = String.valueOf(position);
      possys = Float.valueOf(tempstring);
      posvector.addElement(possys);
   }
   if (time < 51 && time > 40) {
      position = (vsys1 + vsys2 + vsys3 + vsys4) * 10 + vpayload*(time - 40);
      tempstring = String.valueOf(position);
      possys = Float.valueOf(tempstring);
      posvector.addElement(possys);
   }
   time += 1;
}

int count = posvector.size();
gt.clearHashMarks();
gt.clearScreen();
gt.drawAxes(graphWidth, graphHeight);
gt.makeHashMarks(count);
gt.showData(posvector);
/* gt.makeTitle("Position"); */
repaint();
return;
}

} // {DECLARE_CONTROLS

TextField num1;
TextField num2;
TextField num3;
TextField num4;
TextField num5;
TextField num6;
Label label2;
Label label3;
Label label4;
Label label5;
Label label6;
Label label7;
TextField out1;
TextField out2;
TextField out3;
TextField out4;
TextField out5;
Label label8;
Label label9;
Label label10;
Label label11;
Label label12;
Label label13;
Label label14;
Button CalcButton;
} //}

/**
 * Read the HTML data parameters into a Vector object for later
 * use.
 * @param s represents a program-controlled string which is used
 * to get the HTML string (e.g. readData("2") will get the second
 * data set data).
 * readData() is not a part of GraphTools since there are an unlimited
 * number of ways to get data into a Vector. GraphTools assumes that
 * the Vector object already has data.
 */

public Vector readData(String s)
{
    Vector tempVector = new Vector();
    Float param;
    String tempData = null;
    boolean datapresent = true;
    int i = 0;
    Float mass 1;
Float mass2;
Float sys1;

while(datapresent)
{
    try
    {
        tempData = getParameter(s + "_" + (i+1));
    } catch(Exception e)
    {
        System.out.println(e);
    }
    if(tempData == null)
    {
        datapresent = false;
    } else
    {
        param = Float.valueOf(tempData);
        /*tempVector.addElement(param);*/
        i += 1;
    }
}

/*
 sys1 = Float.valueOf(vsys1);*/
/*mass2 = Float.valueOf("20");*/
/* tempVector.addElement(sys1);*/
/* tempVector.addElement(mass2);*/
return tempVector;
}

/**
 * Read the HTML string data (which contains a brief description of the
 * associated data). This string data is the descriptive text which is
 * displayed on the graph.
 * @param s represents the string which is read from the HTML.
 * */

public String readStringData(String s)
{
    String tempString = null;
    Integer param;
    //String temp;
    boolean datapresent = true;
    int i = 0;

    try
    {
        tempString = getParameter(s + "_DESC");
    } catch (Exception e)
    {
        System.out.println(e);
    }
class GraphTools
{

    Graphics myGC;
    Font graphFont;
    int axisH;
    int axisW;
    int xOrigin;
    int yOrigin;
    int xSpacing;
    int ySpacing;

    /**
     * The constructor which sets the font and associates the
     * local graphics context with that created in the main applet.
     */
    public GraphTools(Graphics g)
    {
        myGC = g;
        graphFont = new Font("Helvetica", Font.PLAIN, 10);
        myGC.setFont(graphFont);
    }

    /**
     * Changes the font in general use by the GraphIt services
     */
    public void setFont(Font f)
    {
        graphFont = f;
        myGC.setFont(graphFont);
    }

    /**
     * Draw the X and Y axes on the graphing surface
     */
    public synchronized void drawAxes(int W, int H)
    {
        //The length of the axes should be less than the graph viewing area
axisW = W - 100;
axisH = H - 100;

//Determine the origin of the graph
xOrigin = (W - axisW)/2;
yOrigin = ((H - axisH)/2)+axisH;

//Draw the axes with a 3-D effect
myGC.setColor(Color.lightGray);
myGC.drawLine(xOrigin- 1, yOrigin+l, xOrigin+axisW, yOrigin+l);
myGC.drawLine(xOrigin, yorigin, xOrigin, yorigin-axisH);

//Draw an indented frame around the entire graph (looks cool!)
myGC.drawLine(0, 0, 0, H);
myGC.drawLine(0, 0, W, 0);
myGC.setColor(Color.lightGray);
myGC.drawLine(W, 0, W, H);
myGC.drawLine(W, H, 0, H);

I II*

A method to draw the hash marks on both the X and Y axes.
*/

public synchronized void makeHashMarks(int nXHash)
{
    xSpacing = axisW / nXHash;
ySpacing = axisH / 10;

    //make X-axis hash marks
    myGC.setColor(Color.black);
    int x = xOrigin + (xSpacing/2);
    int y = yOrigin + 1;
    for(int t = 0; t < nXHash; t++)
    {
        myGC.drawLine(x, y, x, y + 5);
        x += xSpacing;
    }
    myGC.setColor(Color.lightGray);

    x = xOrigin + (xSpacing/2);
y = yOrigin + 1;
    for(int t = 0; t < nXHash; t++)
    {
        myGC.drawLine(x+1, y, x+1, y + 5);
        x += xSpacing;
    }

    //make Y-axis hash marks
myGC.setColor(Color.black);
x = xOrigin - 1;
y = yOrigin - ySpacing;
for(int t = 0; t < 10; t++)
{
    myGC.drawLine(x, y, x - 5, y);
y -= ySpacing;
}

myGC.setColor(Color.lightGray);
x = xOrigin - 1;
y = yOrigin - ySpacing;
for(int t = 0; t < 10; t++)
{
    myGC.drawLine(x, y+1, x - 5, y+1);
y -= ySpacing;
}

/**
 * A method to clear the hash marks on the X axis.
 */
public void clearHashMarks()
{
    myGC.setColor(Color.white);
    myGC.fillRect(xOrigin, yOrigin+1, xOrigin+axisW, yOrigin-5);
}

/**
 * Clears the entire graphing area of the screen; adds a buffer
 * on the bottom and right sides of the clearing rectangle of 20
 * pixels to account for any stray string characters which may
 * need cleanup.
 */
public void clearScreen()
{
    myGC.setColor(Color.white);
    myGC.fillRect(xOrigin+1, (yOrigin-axisH)-20, axisW+20, axisH+20);
}

/**
 * A method to take a string and display that string along
 * the top of the graphing area.
 * @param s is the string of text to be displayed.
 */
public void makeTitle(String s)
{
    String temp = s;
    int yTemp = (yOrigin - axisH)/2;
    int xTemp = xOrigin;
}
public synchronized void showData(Vector v) {
    float maxY = 0;
    myGC.setColor(Color.red);
    int xPoint; 
    int yPoint;

    //Determine the maximum y value in the Vector
    for (int i = 0; i < v.size(); i++)
    {
        float temp = ((Float)v.elementAt(i)).floatValue();
        if (temp > maxY)
        {
            maxY = temp;
        }
    }

    //Determine the x coordinate of the first data point
    xPoint = xOrigin + (xSpacing/2);

    int oldX = 0;
    int oldY = 0;

    //Draw a small square at the data point
    for (int j = 0; j < v.size(); j++)
    {
        yPoint = yOrigin - (int)((axisH/maxY) * ((Float)v.elementAt(j)).floatValue());
        myGC.fillRect(xPoint, yPoint, 3, 3);

        //Draw a line between the data points
        if (oldX != 0 && oldY != 0)
        {
            myGC.drawLine(oldX-xSpacing, oldY, xPoint, yPoint);
        }
    }
}
myGC.setColor(Color.blue);

// Write the value of the data point next to the data point
/* String coordString =
Float.toString(((Float)v.elementAt(j)).floatValue());
myGC.drawString(coordString, xPoint+5, yPoint - 4);*/
myGC.setColor(Color.red);

xPoint += xSpacing;

oldX = xPoint;
oldY = yPoint;

} }

C-10