Modeling A Simple Toy

Alex Bame

May 24, 2004

Abstract

A model will be developed to study the motion of a common seasonal toy. The Euler-Lagrange equation will be used to obtain a system of differential equations from which the models motion will be calculated using a numerical solver. The behavior of the model will be analyzed utilising the effective potential and a computer graphics rendering program will be used as an aid to visualizing the motion of the model.

1. The Toy

The toy we will model is a common toy which is typically found in shops around Christmas or Easter. The toy usually consists of some sort of base upon which a shell such as an egg or a Christmas tree rests. The shell is usually split into four pieces, each of which are hinged at the base and free to rotate around the base and to open. There is a plunger in the base of the toy which will cause the toy to spin when pushed in, as the toy spins the shell will open up revealing an enclosed figure of Santa or the Easter Bunny.

2. The Model

We will create a simple model of the toy by simulating it as a set of four rods, each of which is attached to a common pivot point P around which the rods may rotate in two directions. First, the rods may rotate some angle θ off the vertical axis. Secondly, the rods may rotate through some angle ϕ in the x-y plane. The rods will have mass m and length R, with a spring some distance b along each rod which connects to the adjacent rod. We will consider the springs to be massless and their equilibrium point will be at length zero, when the rods are completely vertical. The springs will have spring constant k.

3. The Euler-Lagrange Equation

The Euler-Lagrange equation is a relativly simple way to get a system of differential equations describing the motion of a system by using the kinetic and potential energies. In general the Lagrangian is defined as the difference between the kinetic and potential energy of a system:

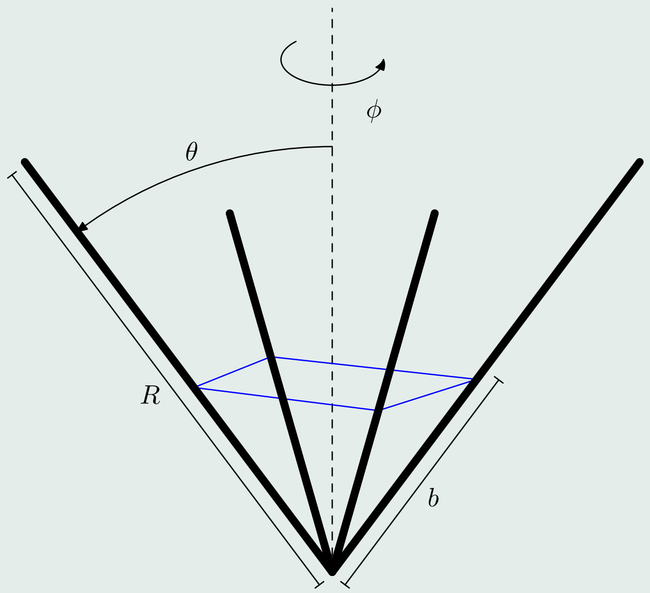

Figure 1: The model

$$L = K - V \tag{1}$$

Where K and V are the kinetic and potential energies, respectivly. In this case, since we are dealing with two variables θ and ϕ, the Euler-Lagrange equation takes the following form:

$$\frac{\partial L}{\partial \theta} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} = 0 \tag{2}$$

$$\frac{\partial L}{\partial \phi} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} = 0 \tag{3}$$

Once we have equations for the kinetic and potential energy of the system we will be able to use Equations (2) and (3) to obtain a system of second order differential equations. The solutions to these differential equations will be minimums of Equation (1) and will describe the motion of the system.

3.1. The Kinetic Energy

In this situation the kinetic energy will be entirely rotational. In general rotational kinetic energy is defined as:

$$K = \frac{1}{2}I\omega^2$$

Where I is the moment of ineria in the plane of rotation and ω is the rotational velocity.

Since the rods can rotate through either θ or ϕ there will be two components to the kinetic energy. The component of kinetic energy in θ is:

$$K_\theta = \frac{1}{2}I\dot{\theta}^2 \tag{4}$$

The component of kinetic energy in ϕ is a little different since only a portion of the bar is rotating in ϕ:

$$K_\phi = \frac{1}{2} I \sin^2(\theta)\dot{\phi}^2 \tag{5}$$

Adding Equations (4) and (5) then multiplying by four (there are four rods) yields the total kinetic energy for the system:

$$\begin{aligned} K &= 4(K_\theta + K_\phi) \\ K &= 2I\dot{\theta}^2 + 2I\dot{\phi}^2 \sin^2(\theta) \\ K &= 2I(\dot{\theta}^2 + \dot{\phi}^2 \sin^2(\theta)) \end{aligned} \tag{6}$$

3.2. The Potential Energy

There are two components to the potential energy of this system, the gravitational and the spring potentials. The general form for gravitational potential energy is:

$$V_G = mgh$$

Where m is the mass of the object, g is the acceleration due to gravity, and h is the height of the object's center of mass. If we assume that each rod has a uniform mass distribution then the center of mass is exactly in the center of the rod. Thus the gravitational potential energy is:

$$V_G = \frac{mgR}{2}\cos(\theta) \tag{7}$$

Recall that we are considering the springs to be massless and so they do not contribute to the gravitational potential.

The general form for spring potential energy is:

$$V_S = \frac{1}{2}kx^2$$

Where k is the spring constant and x is the distance the spring has been stretched from its equilibrium point. A little trig reveals that in our case the spring will stretch a distance of $b\sqrt{2}\sin(\theta)$ and so the spring potential is:

$$\begin{aligned} V_S &= \frac{1}{2}k(b\sqrt{2}\sin(\theta))^2 \\ V_S &= kb^2\sin^2(\theta) \end{aligned} \tag{8}$$

We add Equations (7) and (8) then multiply by 4 to get the total potential energy of the system:

$$\begin{aligned} V &= 4(V_S + V_G) \\ V &= 4kb^2\sin^2(\theta) + 2mgR\cos(\theta) \end{aligned} \tag{9}$$

3.3. Solving The Euler-Lagrange Equation

Now that we have equations for the kinetic and potential energy of the system we can plug them into Equation (1):

$$\begin{aligned} L &= K - V \\ L &= 2I(\dot{\theta}^2 + \dot{\phi}^2\sin^2(\theta)) - 4kb^2\sin^2(\theta) - 2mgR\cos(\theta) \end{aligned}$$

Then solve Equations (2) and (3):

$$
\begin{aligned}
\frac{\partial L}{\partial \theta} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\theta}} &= 0 \\
4I\dot{\phi}^2 \sin(\theta)\cos(\theta) + 2mgR\sin(\theta) - 8kb^2 \sin(\theta)\cos(\theta) - \frac{d}{dt}(4I\dot{\theta}) &= 0 \\
4I(\dot{\phi}^2 \sin(\theta)\cos(\theta) - \ddot{\theta}) + 2mgR\sin(\theta) - 8kb^2 \sin(\theta)\cos(\theta) &= 0 \\
I(\ddot{\theta} - \dot{\phi}^2 \sin(\theta)\cos(\theta)) - \frac{mgR}{2}\sin(\theta) + 2kb^2 \sin(\theta)\cos(\theta) &= 0 \qquad (10)
\end{aligned}
$$

$$
\begin{aligned}
\frac{\partial L}{\partial \phi} - \frac{d}{dt}\frac{\partial L}{\partial \dot{\phi}} &= 0 \\
-\frac{d}{dt}(4I\dot{\phi}\sin^2(\theta) &= 0 \\
-4\ddot{\phi}\sin^2(\theta) - 8\dot{\phi}\dot{\theta}\sin(\theta)\cos(\theta) &= 0 \\
-\ddot{\phi}\sin^2(\theta) - 2\dot{\phi}\dot{\theta}\sin(\theta)\cos(\theta) &= 0 \qquad (11)
\end{aligned}
$$

Now that we have solved the Euler-Lagrange equation we have a system of two second order differential equations which describe the motion of the system. Later we will use a numerical solver to calculate values of θ and ϕ versus time which will be solutions to Equations (10) and (11) given a set of initial conditions.

4. Analyzing The Model

The differential equations we obtained from solving the Euler-Lagrange equation are not linear, however we are interested in analyzing the behavior of the model over a large range of θ values so linearization is not an option. Thankfully this model is well suited for analyzation via utilizing the effective potential.

4.1. The Effective Potential

The effective potential energy of an object is the potential energy that you would measure the object as having when you had adopted that object's refrence frame. Thus essentially we need to take the total energy E of the system and remove any terms that vary with velocity. The energy of the system is:

$$\begin{aligned} E &= K + V \\ E &= 2I\dot{\theta}^2 + 2I\dot{\phi}^2\sin^2(\theta) + 4kb^2\sin^2(\theta) + 2mgR\cos(\theta) \end{aligned} \tag{12}$$

It would seem at first that we need to drop the $\dot{\theta}$ and $\dot{\phi}$ terms. However, recall from earlier:

$$-\frac{d}{dt}(4I\dot{\phi}\sin^2(\theta)) = 0$$

Integrating both sides with respect to t and simplifying yields:

$$\dot{\phi}\sin^2(\theta) = h$$

Where h is some constant. Substituting h into Equation (12) yields:

$$E = 2I\dot{\theta}^2 + \frac{2Ih^2}{\sin^2(\theta)} + 2mgR\cos(\theta) + 4kb^2\sin^2(\theta)$$

Therefore only the $\dot{\theta}$ term varies with velocity. Removing this term we are left with the effective potential $U(\theta)$:

$$U(\theta) = \frac{2Ih^2}{\sin^2(\theta)} + 2mgR\cos(\theta) + 4kb^2\sin^2(\theta) \tag{13}$$

Now we we will be able to use the effective potential to analyze the behavior of the model.

4.2. Utilizing The Effective Potential

In Figure 2 we see a graph of $U(\theta)$ and each of the individual components of $U(\theta)$ versus θ. Note that as $\theta \to \frac{\pi}{2}$, $\frac{2Ih^2}{\sin^2(\theta)}$ decreases from positive infinity to $2Ih^2$, while $2mgR\cos(\theta)$ decreases from $2mgR$ to 0, and $4kb^2\sin^2(\theta)$ increases from 0 to $4kb^2$. Note also how the bowl shape of the graph is determined by the interplay between the centrifugal term and the spring term. If the spring term is too large the bowl in the graph will be very steep, if it is too small the bowl will dissapear alltogether. We want a compromise between those two extremes.

Recall that our energy equation (12) had a term involving $\dot{\theta}$ in addition to the potential energy:

$$E = 2I\dot{\theta}^2 + U(\theta)$$

Solving for $\dot{\theta}$ we obtain:

$$\dot{\theta} = \pm\sqrt{\frac{E - U(\theta)}{2I}} \tag{14}$$

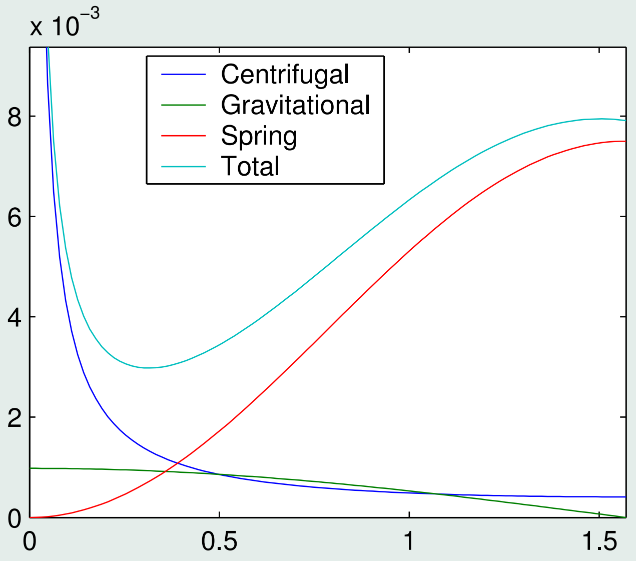

Figure 2: $U(\theta)$ vs. θ

Looking at Equation (14) we can see that as $U(\theta)$ decreases $|\dot{\theta}|$ must increase since we are neglecting friction and E is therefore constant.

In Figure 3 we are once again examining $U(\theta)$, however this time we are also including graphs of various energy levels. The difference between the graph of $U(\theta)$ and the graphs of the different energy levels determines the value of $\dot{\theta}$ as discussed above and as can be seen in Figure 4.

When considering the behavior of the system by analysing the potential energy graph it is usefull to think of a ball rolling across a surface shaped by the graph. If we were to start the ball on the left of the graph at a height greater than the maximum height of the right side of the bowl the ball would roll off the edge of the bowl and go shooting

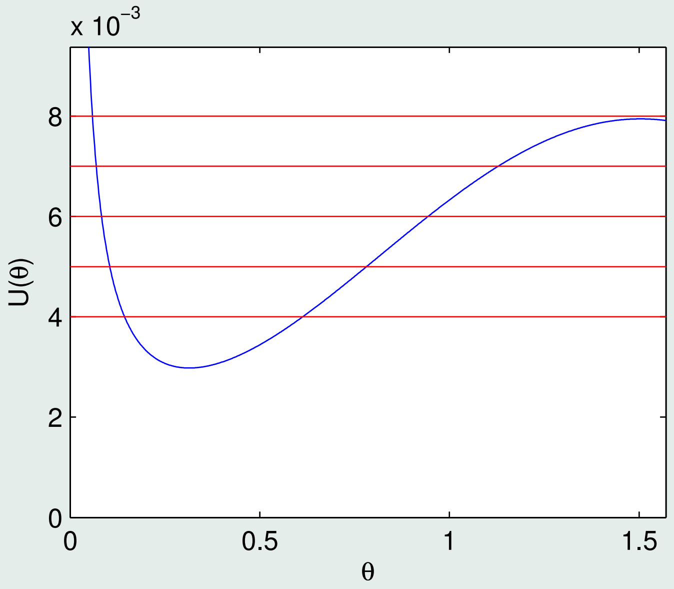

Figure 3: U vs. θ and various energy levels

off to parts unknown. If, however, we were to start the ball at some height less than the maximum height of the bowl on the right of the graph the ball would roll back and forth between the two sides of the bowl forever. The maximum value for the height of the bowl on the right side is the dividing line between where the ball will simply roll back and forth and where the ball will go flying out of the bowl. Also, if we were to start the ball off at exactly the minimum height of the bowl the ball would simply sit in the center.

Our system of rods and springs will behave simmilarly. If we give the system an initial energy level that is lower than the right edge of the bowl the rods will oscillate around some value of θ forever. If we give the system an initial energy level above the

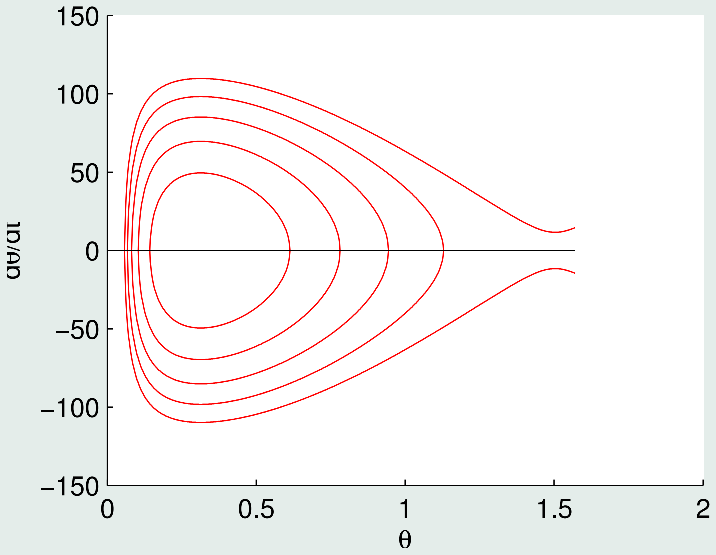

Figure 4: $\dot{\theta}$ vs. θ and energy level curves

right edge of the bowl it will go flying off past $\pi/2$ radians, which is a situation that is prevented in the actual toy by a physical barrier. If we start our system off with just the right level of energy it will sit at one value of θ forever.

5. Software Simulations of the Model

Now that we have solved the Euler-Lagrange equation to get a system of differential equations describing the motion of our system and used our analysis of the effective potential to gain an understanding of how the system will behave we are ready to create a simulation of the system using a numerical solver and a computer graphics rendering program.

For this paper I will be using MatLab's **ode45** variable step numeric solver to get values of θ and ϕ over time, then I will use the Persistence of Vision Raytracer to create an animation of the motion. There are many other software packages that can accomplish the same tasks, these are simply the two applications that I chose to use.

5.1. Setting Up Our DEs

First we must solve Equations (10) and (11) for $\ddot{\theta}$ and $\ddot{\phi}$:

$$\ddot{\theta} = \frac{mgR}{2I}\sin(\theta) - \frac{2kb^2}{I}\sin(\theta)\cos(\theta) + \dot{\phi}^2\sin(\theta)\cos(\theta)$$
$$\ddot{\phi} = \frac{-2\dot{\phi}\dot{\theta}\cos(\theta)}{\sin(\theta)}$$

Then we will use variable substitution to express our system of second order differential equations as a system of first order differential equations:

$$x_1 = \phi$$
$$x_2 = \dot{\phi}$$
$$x_3 = \theta$$
$$x_4 = \dot{\theta}$$

Using these substitutions results in:

$$
\begin{aligned}
\dot{x_1} &= x_2 \\
\dot{x_2} &= \frac{mgR}{2I}\sin(x_3) - \frac{2kb^2}{I}\sin(x_3)\cos(x_3) + (x_2)^2\sin(x_3)\cos(x_3) \\
\dot{x_3} &= x_4 \\
\dot{x_4} &= \frac{-2x_4x_2\cos(x_3)}{\sin(x_3)}
\end{aligned}
$$

This system of first order equations can now be run through the **ode45** numerical routine in order to obtain values for θ and ϕ.

5.2. Utilizing The Numerical Solver

For my own use I wrote a GUI utility so that I could easily change the various parameters to the equations and see the effects by clicking a button, however the extra code to draw the GUI makes the program much too complicated to review here. Therefore I will go over a few critical points here and provide the full source in Section 7. Additionally you will need the gui.fig file which should be linked to off the webpage you downloaded this document from.

The **ode45** routine requires that we provide it with a function that will be used to calculate the derivatives of our variables for whatever values it passes to the function. This is the heart of the program and simply involves typing in the system of first order equations we developed earlier:

```
% This function calculates values for x1', x2', x3',
% and x4' using the values for x1, x2, x3, and x4
% that are passed to it in the vector x by the ode45
% routine
```

```
function xprime=F(t, x, m, R, k, b)
% Some constants
g=9.8;
I=m*R^2/3;

% These are just the coefficents of the spring and
% gravitational terms, I've broken them out here in
% an attempt to make the lines below easier to read
q=m*g*R/(2*I);
w=2*k*b^2/I;

% Calculate the derivatives
xprime=[x(2); ...
        ...
        -2*x(4)*x(2)*cos(x(3))/sin(x(3)); ...
        ...
        x(4); ...
        ...
        q*sin(x(3))-(w-x(2)^2)*sin(x(3))*cos(x(3))];
```

Then we call the **ode45** routine and tell it to use our function F:

```
% Compute values for theta, thetadot, phi, and phidot over
% the time interval 0 to 1 seconds
[t,x]=ode45(@F, ...
            linspace(0,1,500), ...
            [handles.pinit, handles.pdinit, ...
             handles.tinit,handles.tdinit], ...
            options, ...
            handles.m, handles.R, handles.k, handles.b);
```

And finally there is a section to write the computed values for time, θ, and ϕ to a text file so that we can use it later to render the animation:

```
% Write the data returned from ode45 to a text file
fid=fopen('odeoutput.txt', 'w+');
for c = 1:size(t)
    fprintf(fid, '%5f %5f %5f\n', [t(c); x(c,3); x(c,1)]);
end
fclose(fid);
```

Those sections are the meat of the program, all the rest of the code is just formatting and passing variables around to make the graphics work.

5.3. Rendering The Animation

Now that **ode45** has given us a list of values for time, θ, and ϕ we can set up our POV-Ray scene file to simulate what the physical situation is doing. Once again we'll just cover the important parts, the entire source is contained in Section 8. First we must define a base model for one of the rods:

```
#declare Rod= cylinder {
    <0, 0, 0>,          // Center of one end
    <0, 2, 0>,          // Center of other end
    cyl_radius          // Radius
    texture {
        T_Chrome_3A     // Pre-defined texture
        scale 4         // Scale by the same ammount in all
                        // directions
    }
    rotate <0, 0, theta*180/pi>
}
```

This declares a rod of length two with one end at the origin, then rotates the rod down θ radians off the vertical axis.

Next we define a base model for a spring:

```
#declare Spring= isosurface {
    function {
        f_helix1(x, y, z, 1, 20*pi,
                 spr_min_radius, spr_maj_radius,
                 1, 1, 0)
    }
    max_gradient 3
    contained_by { box{-1/2, 1/2} }
    texture {T_Brass_3A}

    scale <1, sqrt(2)*sin(theta), 1>
    rotate <0, 0, 90>
    rotate <0, 90, 0>
    translate <0, 1, 0>
    rotate <0, 0, acos(cos(theta)/
                      sqrt(1-1/2*pow(sin(theta),2)))*180/pi>
}
```

This creates a spring with of the proper length with ten loops, then rotates it off the vertical axis by an angle $\cos^{-1}(\cos(\theta)/\sqrt{1-1/2*\sin^2(\theta)})$. We don't rotate the spring by θ because the spring connects the adjacent rods via a straight line, causing our toy to have the shape of a pyramid.

Now that we have our basic rods and springs defined we can go ahead and make four copies, each of which is rotated the appropriate angle through ϕ:

```
object { Rod rotate <0, phi*180/pi, 0> }
object { Rod rotate <0, (phi+pi/2)*180/pi, 0> }
object { Rod rotate <0, (phi+pi)*180/pi, 0> }
object { Rod rotate <0, (phi+3*pi/2)*180/pi, 0> }

object { Spring rotate <0, (phi+pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+3*pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+5*pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+7*pi/4)*180/pi, 0> }
```

One of the nice things about POV-Ray is that it can handle everything using just vectors and rotations, so it's extremely easy to model most physical situations and then quickly express them as a POV-Ray scene file.

Now that we have a text file listing all the values for time, θ, and ϕ and we have a scene file prepared all that remains is to use the values from **ode45** to render our scene file. To accomplish this I have used a UNIX shell script that takes the values from the text file and uses them to render a sequence of images which can later be combined into an animation. The script is a simple one:

```
for a in `cat odeoutput.txt|sed "s/ /_/g"`; do
  TIME=`echo $a|cut -d'_' -f1`;
  THETA=`echo $a|cut -d'_' -f2`;
  PHI=`echo $a|cut -d'_' -f3`;
  cat defs.ini|sed s/var1/$THETA/|sed s/var2/$PHI/>newdefs.ini;
  povray -W640 -H480 +Oanim-$TIME.png anim.pov;
done;
```

This script simply parses the values for θ and ϕ from the **odeoutput.txt** file that was written by our MatLab program then substitutes those values into a prepared defs.ini file which is included from the POV-Ray scene file. Having accomplished that the script then renders the frame using the values for ϕ and θ then moves on to the next set of values. The defs.ini file can be found in Section 9.

6. Conclusion

We have sucesfully analyzed our model of the situation and developed a software simulation of the motion. Using the effective potential we were able to determine the oscilatory motion of the system, which was later confirmed by solving the system of differential equations given by the Euler-Lagrange equation with a numerical solver and rendering the motion with POV-Ray.

After overcoming the initial learning curve POV-Ray should serve as a valuable resource for visualizing complex physical situations. It lends itself easily to modeling situations that use either vectors and rotations or simple x, y, z coordinates. This flexibility should allow for its incorporation into any number of different models, resulting in a more easily understood and visually intuitive analysis.

7. Appendix A

```
function varargout = gui(varargin)
% GUI M-file for gui.fig
%      GUI, by itself, creates a new GUI or raises the existing
%      singleton*.
%
%      H = GUI returns the handle to a new GUI or the handle to
%       the existing singleton*.
%
%       GUI('CALLBACK',hObject,eventData,handles,...) calls the
%       local function named CALLBACK in GUI.M with the given
%       input arguments.
%
%       GUI('Property','Value',...) creates a new GUI or raises the
%       existing singleton*.  Starting from the left, property
%       value pairs are applied to the GUI before
%       gui_OpeningFunction gets called.  An unrecognized property
%       name or invalid value makes property application stop.  All
%       inputs are passed to gui_OpeningFcn via varargin.
%
%       *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows
%       only one instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui

% Last Modified by GUIDE v2.5 15-May-2004 15:08:04

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_OutputFcn, ...
```

```
                              'gui_LayoutFcn',   []  , ...
                              'gui_Callback',    []);
if nargin & isstr(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before gui is made visible.
function gui_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui (see VARARGIN)

handles.fig1hndl=0;
handles.fig2hndl=0;
handles=setvars(hObject, handles);

% Choose default command line output for gui
handles.output = hObject;
```

```
% Update handles structure
guidata(hObject, handles);

% UIWAIT makes gui wait for user response (see UIRESUME)
% uiwait(handles.figure1);


%--- Outputs from this function are returned to the command line.
function varargout = gui_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


%--- Executes during object creation, after setting all
%--- properties.
function mass_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to mass_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                    CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%         See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor', ...
        get(0,'defaultUicontrolBackgroundColor'));
end



function mass_edit_Callback(hObject, eventdata, handles)
% hObject    handle to mass_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of mass_edit as
%        text str2double(get(hObject,'String')) returns contents
%        of mass_edit as a double


%--- Executes during object creation, after setting all
%--- properties.
function length_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to length_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                  CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor', ...
        get(0,'defaultUicontrolBackgroundColor'));
end



function length_edit_Callback(hObject, eventdata, handles)
% hObject    handle to length_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of length_edit as
%        text str2double(get(hObject,'String')) returns contents
%        of length_edit as a double


%--- Executes during object creation, after setting all
%--- properties.
function spring_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to spring_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                  CreateFcns called
% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor', ...
        get(0,'defaultUicontrolBackgroundColor'));
end



function spring_edit_Callback(hObject, eventdata, handles)
% hObject    handle to spring_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of spring_edit as
%        text str2double(get(hObject,'String')) returns contents
%        of spring_edit as a double


%--- Executes during object creation, after setting all
%--- properties.
function springdist_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to springdist_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                             CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%        See ISPC and COMPUTER.
if ispc
     set(hObject,'BackgroundColor','white');
else
     set(hObject,'BackgroundColor', ...
          get(0,'defaultUicontrolBackgroundColor'));
end



function springdist_edit_Callback(hObject, eventdata, handles)
% hObject     handle to springdist_edit (see GCBO)
% eventdata   reserved: to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of springdist_edit
%         as text str2double(get(hObject,'String')) returns
%         contents of springdist_edit as a double


%--- Executes on button press in calc_equil.
function calc_equil_Callback(hObject, eventdata, handles)
% hObject     handle to calc_equil (see GCBO)
% eventdata   reserved: to be defined in a future version of MATLAB
% handles     structure with handles and user data (see GUIDATA)
```

```
handles=setvars(hObject, handles);
theta=linspace(0,pi/2);
u=2*handles.I*handles.h^2./sin(theta).^2 ...
    + 2*handles.m*handles.g*handles.R*cos(theta) ...
    + 4*handles.k*handles.b^2*sin(theta).^2;
[Y,I]=min(u)
theta(I)


%--- Executes during object creation, after setting all
%--- properties.
function angvel_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to angvel_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor', ...
        get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function angvel_edit_Callback(hObject, eventdata, handles)
% hObject    handle to angvel_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of angvel_edit as
%        text str2double(get(hObject,'String')) returns contents
%        of angvel_edit as a double


%--- Executes during object creation, after setting all
%--- properties.
function rot_edit_CreateFcn(hObject, eventdata, handles)
% hObject    handle to rot_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    empty - handles not created until after all
%                  CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
    set(hObject,'BackgroundColor','white');
else
    set(hObject,'BackgroundColor', ...
        get(0,'defaultUicontrolBackgroundColor'));
end
```

```
function rot_edit_Callback(hObject, eventdata, handles)
% hObject    handle to rot_edit (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of rot_edit as
%        text str2double(get(hObject,'String')) returns contents
%        of rot_edit as a double


%--- Executes on button press in graph_btn.
function graph_btn_Callback(hObject, eventdata, handles)
% hObject    handle to graph_btn (see GCBO)
% eventdata  reserved: to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Grab the values of the various input fields
handles=setvars(hObject, handles);

% Generate 100 equally spaced theta values between 0 and pi/2
theta=linspace(0,pi/2,100);

% Calculate the three components of the potential energy equation
cent=2*handles.I*handles.h^2./sin(theta);
grav=2*handles.m*handles.g*handles.R*cos(theta);
spring=4*handles.k*handles.b^2*sin(theta).^2;
```

```
% If our first figure window has not already been opened we'll
% create one and store the handle
if handles.fig1hndl == 0
    figure
    handles.fig1hndl=gcf;
end

guidata(hObject, handles)

% Ensure that the first figure window is the active figure window
% then clear it of any old plots that might be left over
figure(handles.fig1hndl);
clf;

% Plot the potential energy curves
plot(theta, cent, ...
    theta, grav, ...
    theta, spring, ...
   theta, cent + grav + spring);

% Provide a legend so we can tell which plot is which
legend('Centrifugal', 'Gravitational', 'Spring', 'Total');

% The centrifugal term is coming down from +inf and can
% really mess with the range of the axis, in general all
% of the interesting behavior will occur between 0 and
% just over the maximum height of the spring graph
```

```
top=max(spring);
axis([0, pi/2, 0, top+1/4*top]);

% If the second figure window is not open create a new one
if handles.fig2hndl == 0
     figure
     handles.fig2hndl=gcf;
end

% Store the handle for the second figure window
guidata(hObject, handles);

% Ensure that we the second figure window is active and then
% clear it of any old plots that might be hanging around
figure(handles.fig2hndl);
clf;

% We will not be passing any special options to ode45
options=[];

% Plot values for theta, thetadot, phi, and phidot over
% the time interval 0 to 1 seconds
[t,x]=ode45(@F, ...
           linspace(0,1,500), ...
           [handles.pinit,handles.pdinit, ...
            handles.tinit,handles.tdinit], ...
           options, ...
           handles.m, handles.R, handles.k, handles.b);
```

```
% Write the data returned from ode45 to a text file
fid=fopen('odeoutput.txt', 'w+');
for c = 1:size(t)
    fprintf(fid, '%5f␣%5f␣%5f\n', [t(c); x(c,3); x(c,1)]);
end
fclose(fid);

% The second figure window is broken up into four subplots,
% one for each of theta, thetadot, and phidot then one where
% all of theta, thetadot, phi, and phidot are plotted.

% Create the first subplot, which is a graph of theta vs t
subplot(2,2,1);
plot(t,x(:,3), 'Color', 'r');
xlabel('t');
ylabel('Theta');

% The second subplot is thetadot vs t
subplot(2,2,2);
h=plot(t,x(:,4));
set(h, 'Color', [0,0.75,0.75]);
xlabel('t');
ylabel('Thetadot');

% The third subplot is phidot vs t
subplot(2,2,3);
h=plot(t,x(:,2));
```

```
set(h, 'Color', [0,0.5,0]);
xlabel('t');
ylabel('Phidot');

% The fourth subplot is all of theta, thetadot,
% phi, and phidot vs t
subplot(2,2,4);
plot(t,x);
legend('Phi', 'Phidot', 'Theta', 'Thetadot');
xlabel('t');


% This function calculates values for x1', x2', x3',
% and x4' using the values for x1, x2, x3, and x4
% that are passed to it in the vector x by the ode45
% routine
function xprime=F(t, x, m, R, k, b)
% Some constants
g=9.8;
I=m*R^2/3;

% These are just the coefficents of the spring and
% gravitational terms, I've broken them out here in
% an attempt to make the lines below easier to read
q=m*g*R/(2*I);
w=2*k*b^2/I;

% Calculate the derivatives
```

```
xprime=[x(2); ...
          ...
          -2*x(4)*x(2)*cos(x(3))/sin(x(3)); ...
          ...
          x(4); ...
          ...
          q*sin(x(3))-(w-x(2)^2)*sin(x(3))*cos(x(3))];

% This is a function to get the values from the various
% input fields, convert them into numerical values, then
% store them in the variable storage array that gets
% passed around between functions
function ret=setvars(hObject, handles)
g=9.8;
m=str2double(get(handles.mass_edit,'String'));
R=str2double(get(handles.length_edit,'String'));
k=str2double(get(handles.spring_edit,'String'));
b=str2double(get(handles.springdist_edit,'String'));
I=1/3*m*R^2;
tinit=eval(get(handles.rot_edit,'String'));
tdinit=0;
pinit=0;
pdinit=eval(get(handles.angvel_edit,'String'));
h=pdinit*sin(tinit)^2;

handles.g=g;
handles.m=m;
handles.R=R;
handles.k=k;
handles.b=b;
handles.I=I;
handles.tinit=tinit;
handles.tdinit=tdinit;
handles.pinit=pinit;
handles.pdinit=pinit;
handles.pdinit=pdinit;
handles.h=h;
guidata(hObject, handles);
ret=handles;
```

8. Appendix B

```
#include "colors.inc"
#include "metals.inc"
#include "functions.inc"
#include "transforms.inc"

#include "newdefs.ini"

#declare cyl_radius=0.07;
#declare spr_maj_radius=0.05;
#declare spr_min_radius=0.01;

#declare Rod= cylinder {
    <0, 0, 0>,        // Center of one end
    <0, 2, 0>,        // Center of other end
    cyl_radius          // Radius
    texture {
        T_Chrome_3A     // Pre-defined texture
        scale 4         // Scale by the same ammount in all directions
    }
    rotate <0, 0, theta*180/pi>
}

#declare Spring= isosurface {
    function {
        f_helix1(x, y, z, 1, 20*pi,
                  spr_min_radius, spr_maj_radius,
                  1, 1, 0)
    }
    max_gradient 3
    contained_by { box{-1/2, 1/2} }
    texture {T_Brass_3A}

    scale <1, sqrt(2)*sin(theta), 1>
    rotate <0, 0, 90>
    rotate <0, 90, 0>
    translate <0, 1, 0>
    rotate <0, 0,
             acos(cos(theta)/sqrt(1-1/2*pow(sin(theta),2)))*180/pi>
}

camera {
    location <0, 2, -3>
    look_at <0, 1, 0>
}

object { Rod rotate <0, phi*180/pi, 0> }
object { Rod rotate <0, (phi+pi/2)*180/pi, 0> }
object { Rod rotate <0, (phi+pi)*180/pi, 0> }
object { Rod rotate <0, (phi+3*pi/2)*180/pi, 0> }

object { Spring rotate <0, (phi+pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+3*pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+5*pi/4)*180/pi, 0> }
object { Spring rotate <0, (phi+7*pi/4)*180/pi, 0> }

light_source { <2, 4, -3> color White }
light_source { <-2, -4, -3> color White }
```

9. Appendix C

```
#declare theta=var1;
#declare phi=var2;
```

References

[1] R. B. Prigo *American Journal of Physics, Vol. 52 Issue 4, p. 335*

[2] Serway & Beichner *Physics For Scientists and Engineers with Modern Physics Fifth Edition*

[3] Professor Dave Arnold, College of the Redwoods

[4] Professor Scott Pilzer, College of the Redwoods