Devon Lafferty
Math 164
Final Project
5/06/07

Inverted Stiff String: the Indian Rope Trick

1.1 INRODUCTION

Many seemingly simple physical systems encountered in life can have very interesting and sometimes non-intuitive dynamics. Pendulums, one of the first systems studied by young students, have been shown to have very interesting dynamics when their pivot point is oscillated at high frequencies. With this modification, pendulums possess a peculiar equilibrium state when they are inverted, or pointing vertically up.

It was later proven that even systems composed of multiple pendulums linked together (double pendulums, triple pendulums, etc.) will also exhibit these dynamics. This evidence suggests the intriguing thought that this inverted equilibrium may even be present in the continuous case, when the number of pendulums goes to infinity – an ideal string. Unfortunately, this hypothesis was mathematically disproved, as the required frequency for stability was found to increase with the pendulum density.

However, the stable equilibrium in inverted strings, playfully named the "Indian Rope Trick," was found experimentally. This is because no real string is ideal, a possessing a "stiffness" that resists bending. The goal in this paper is to numerically model the linearized equations for an inverted "stiff" string and analyze its different properties.

1.2 MATHEMATICAL FORMULATION

We begin with a stiff, inextensible string of length L, held in the near upright position and attached to an oscillator at its base. This oscillator moves according to

$$z = \varepsilon \cos \omega_0 t \,, \tag{1}$$

assuming ε is small.

We will assume that the displacements **u**(s,t) of the string are small, such that

$$\mathbf{u}(s,t) \approx \mathbf{u}(z,t) \tag{2}$$

Although this assumption quickly becomes invalid without a forced base, it should model the equilibrium state for high oscillations.

The forces affecting the string at its base are gravity g and the force due to the oscillating base, giving the equation

$$F = -\rho g - \omega^2 \varepsilon \cos \omega t \, . \tag{3}$$

For a perfectly upright string, the tension should vary along it, with tension being zero at the free end. The function of tension along a string is then

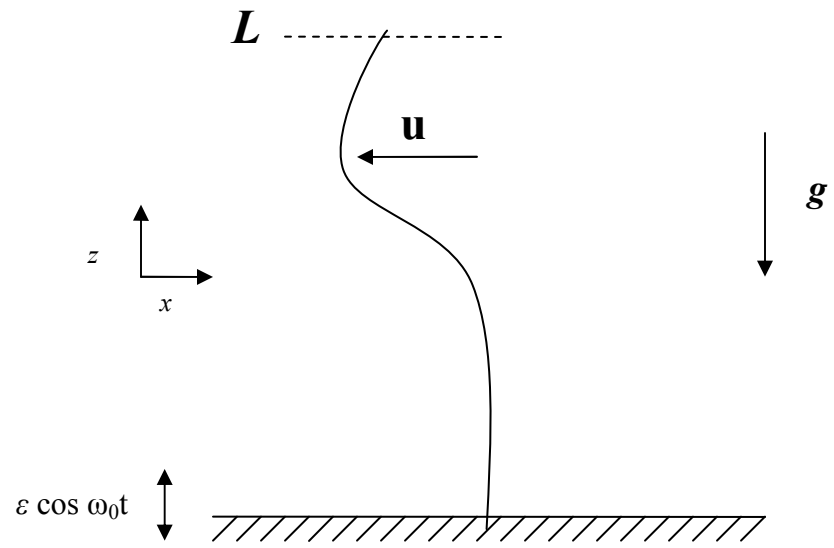

Fig. 1. Inverted string definitions.

$$T = -\left(\rho g - \omega^2 \varepsilon \cos \omega t\right)\left(L - s\right) \tag{4}$$

Therefore, small horizontal perturbations **u** away from vertical will also possess approximately the same tension, since **u** is perpendicular to the tension.

Now we apply the equation that governs the dynamics of inextensible strings,

$$\rho \ddot{\mathbf{u}} = \frac{\partial}{\partial s} T(s,t) \mathbf{u}' \, , \tag{5}$$

In order to find

$$\ddot{\mathbf{u}} = -\left(\frac{-g}{L} - \varepsilon \omega^2 \cos \omega t \right) \left[(1-s) \mathbf{u}' \right]'$$

Also, for our small perturbations, the x or y components donot affect the solution, only total displacement, so $\mathbf{u}(z,t) = u(z,t)$.

Finally, we add in stiffness, which is a fourth order spatial derivative term, proportional to the stiffness factor B.

We now have the governing equation:

$$\ddot{u} = -\left(\frac{-g}{L} - \varepsilon\omega^2 \cos\omega t\right)\left[(1-s)u'\right]' - Bu'''' . \tag{6}$$

and the boundary conditions

$$u = u' = 0\,, \text{ at s } = 0 \tag{7}$$

$$u'' = u'''' = 0\,, \text{ at s } = 1. \tag{8}$$

COMPUTATIONAL RESULTS

Numerical simulations were run using Matlab™ 7.3.0.

It was difficult to determine which numerical method to use in order to model the equations above, for a variety of reasons. The most important aspect of the model is generating stable inverted solutions. Since we are dealing with a fourth order derivative term, it seems necessary for the model to preserve terms of higher order. Thus, a method made for preserving these terms like the pseudospectral method would have been ideal. However, this problem lacks the periodic boundary conditions necessary for implementation. Thus, a finite difference method was used.

Naïvely, a second-order finite difference method was first implemented to model the dynamics. Stable solutions were found for $\varepsilon > 0$, so the numerical model does agree that certain stable solutions exist. A sample graph is shown in Fig. 1. The frequency ω controlled whether the solution was stable, with frequency below some threshold allowing the string to fall.

For $\varepsilon = 0$, the string would fall to the left or right depending on initial conditions. In addition, increasing the stiffness B caused the solutions to try and avoid bending in parts of the string, which agrees with theory.

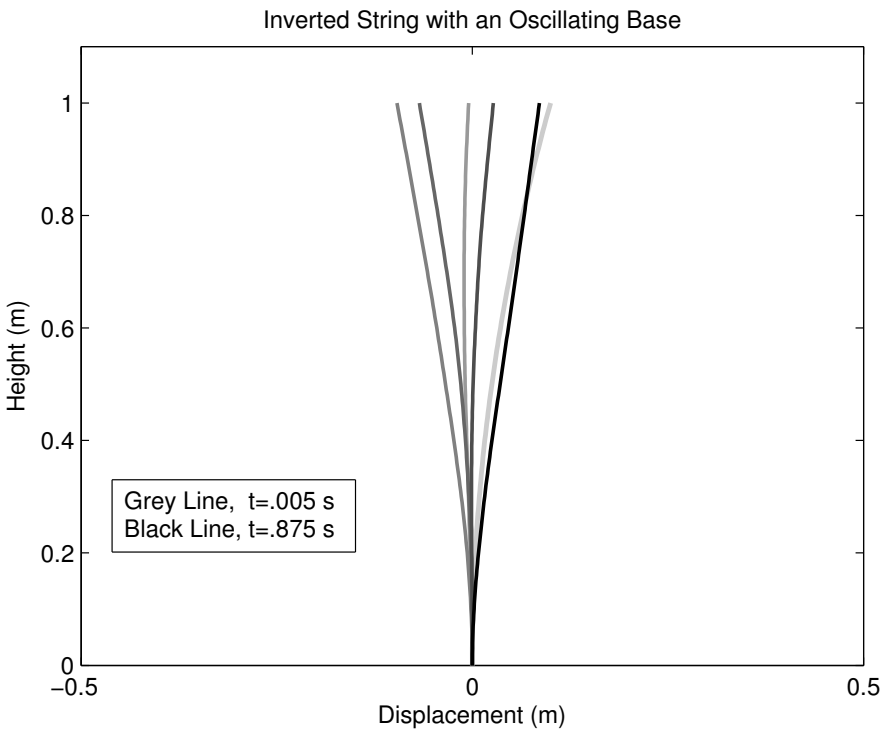

Fig. 1. We see that for an oscillating base, the string has a stable equilibrium in its upright position.

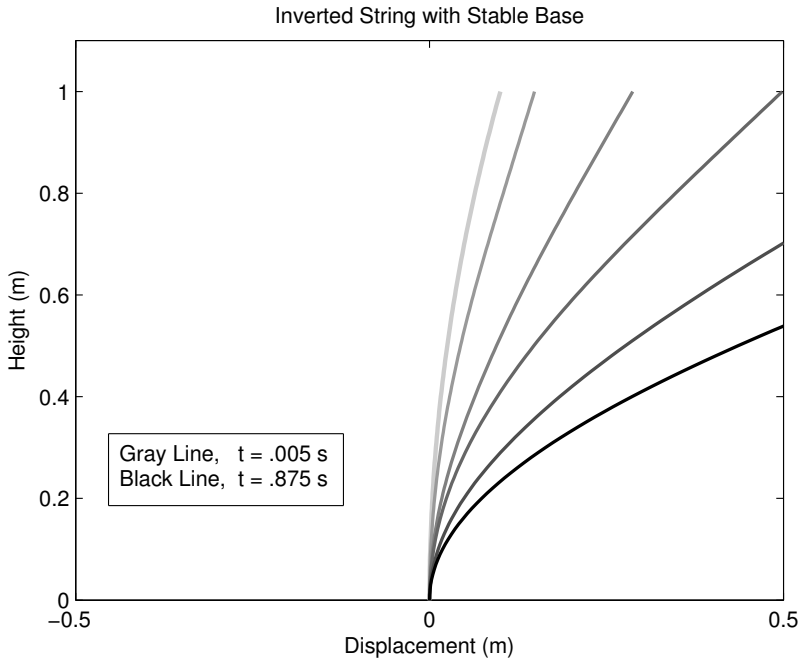

Fig. 2. In this image, we see that for a base that does not oscillate, the string "fall" off.

However, there were some problems in the model that were nonphysical.

- No matter how large stiffness B was increased, with $\varepsilon = 0$ the string would always fall over to one side. Physically, a very rigid object that was held stationary at its bottom without letting it swing would cause the object to remain upright. This is most likely caused by the stiffness being modeled using a fourth order derivative term u''''. Although this term creates stiffness for very quickly changing u, it still allows the string to bend, as long as that bending is not fourth order. For example, even with an infinite B, the string would still be allowed to bend into a parabola.

- For stable solutions, increasing the B caused the string to oscillate *farther* away from equilibrium. This is counter to the mathematical prediction that stiffness stabilizes a solution. Most likely, this is due to the nonphysical model of stiffness.

- For $\varepsilon > 0$, stable solutions were found even with $B = 0$. This again does not agree with theory. Most likely, since a second-order finite difference method has error $\boldsymbol{O}(h^2)$, then fourth order derivative terms can arise. These error terms would then act as a an extra B, letting B=0 does not truly remove stiffness.

- When the time step size was changed, it altered the overall results, increasing the amount of total time an oscillation would take. This is most likely also due to the extra fourth order derivative terms, since these terms are proportional to the ratio of Δx and Δt.

The model itself had several caveats. First of all, the model required an extremely small time step in order to avoid the Courant-Friedrichs-Levy condition. For fourth order PDEs, this condition can be as stringent as

$$\frac{\Delta t}{\Delta x^4} < C. \tag{9}$$

C for this model also was affected by the input frequency ω_0 and the stiffness B. Thus, any practical time resolved data required an enormous number of time steps, and the number was sensitive to any changes in B and ω_0.

A spatial fourth-order finite difference method was coded to attempt to analyze the dynamics, but the required number of steps could no be determined. After attempting extremely large time steps over an increasingly small total time, this method was abandoned.

CONCLUSIONS

Despite the disappointing shortcomings of the model, it did predict that vertically oscillating a string's pivot point creates stable solutions in the inverted position, as long as the frequency was above some threshold. This is promising, and was the original goal

of this project. If this project were to be continued, the next step would be to re-examine the mathematical model (particularly the stiffness component). The model also needs to be revised: the fourth-order finite difference method could be attempted on a faster or more powerful computer capable of handling the required time steps, and other PDE solving methods could be attempted.

REFERENCES

Acheson, D.J. "A Pendulum Theorem." Proceedings: Mathematical and Physical Sciences, Vol. 44, No. 1917, pp. 239-245. Oct 1993.

"Courant-Friedrichs-Lewy condition." Wikipedia, the Free Encyclopedia. Accessed May 4, 2007. http://en.wikipedia.org/wiki/Courant-Friedrichs-Levy_condition

Champneys, Alan R. and Fraser, W. Barrie. "The 'Indian Rope Trick' for a Parametrically Excited Flexible Rod: Linearized Analysis." Proceedings: Mathematical and Physical Sciences, Vol. 456, No. 1995, pp. 530-570. March 200.

Yong, Darryl. "Strings, Chains, and Ropes." SIAM Review, Vol. 48, No. 4, pp. 771-781. November 2006.

APPENDIX : MATLAB CODE

```
% 2ndstring.m 5/6/2007 Devon Lafferty
%
% This program tries to solve the equation governing an upright stiff
% string with an oscillating base.
% A second-order space finite diffference method was used, with
equations
% gained from Maple.
% Since only second order, problems arised due to higher order effects.
%
% It uses a second-order central differences in time.


clear;
L=1;
Nx=100;
dx=L/Nx;
T=5.0;  % final time
Nt=800000;  % %number of actual time steps
dt=T/Nt;
alpha=(dt/dx)^2;
tstep=.005;              %time step to actually be shown
nstep=int32(tstep*Nt/T);
solstep=int32(T/tstep);



%set physical constants
g=9.8;          % acceleration due to gravity in m/s^2
w=20;          % frequency of oscillation in radians/s
m=.4;           % mass per unit length in kg/m



%find the appropriate constants
%B=realB/(m*g*L^3);
B=0.2;
e=.1;
del=g/L;

C=alpha*del*B*alpha/dt^2;



% set up spatial and temporal grid
x=[0:Nx]*dx;
t=[0:Nt]*dt;

% solution will be arranged in a matrix Sol, with intermediate matrix
% v.
```

```
v=zeros(nstep,Nx+1);
sol=zeros(solstep,Nx+1);

f=.1*x.^2;

% initial velocity u_t(x,0)=g(x)
deri=0*x;
% use the initial conditions to figure out solution
% for t=0, t=dt.
v(1,:)=f;
sol(1,:)=f;
v(2,:)=f+dt*deri;

v(2,1)=0;


breaker=0;                      %breaks the solution when it gets too
big

% This uses 2 for loops, one to calculate the main time step, the other
% to calculate the intermediate time steps.
lastk=0;
for k=2:solstep
    for j=3:nstep
      % find solution at next time
      for i=3:(Nx-1)
          G=-(del-e*w^2*cos(w*(double(j)-1)*dt));
          Q=(1-i*dx);
          v(j,i)=-C*(v(j-1,i+2)+v(j-1,i-2))+(alpha*G*(Q-dx/2)+4*C)*v(j-
1,i+1)+...
          (alpha*G*(Q+dx/2)+4*C)*v(j-1,i-1)+(2-2*alpha*Q*G-6*C)*v(j-
1,i)-v(j-2,i);
          if abs(v(j,i))>2
            breaker=1;      %set up to break for nonphysical answers
over 1
          end
      end
      % Dirichlet and Neumann boundary condition at x=0
      v(j,1)=0;
      v(j,2)=1/4*v(j,3);
      %2nd and 4th order Neumann boundary condition at x=L
      v(j,Nx)=1/13*(40*v(j,Nx-1)-45*v(j,Nx-2)+22*v(j,Nx-3)-4*v(j,Nx-
4));
      v(j,Nx+1)=5/2*v(j,Nx)-2*v(j,Nx-1)+1/2*v(j,Nx-2);
      if breaker==1
          break;
      end
    lastj=j;
    end

    if breaker==1
        break;
    end;
    sol(k,:)=v(nstep,:);    %gives sol the data from v
    v(1,:)=v(nstep-1,:);    %resets v
```

```
    v(2,:)=v(nstep,:);

    lastk=k;            %saves the point it stops
end



figure(1)
hold off
%The data is then plotted as a movie
for j=1:lastk
  plot(sol(j,:),x)
  axis([-.5 .5 0 L+.1])
  text(.1,1,[' Time = ',num2str(t(j*nstep))]);
  M(j)=getframe;
end
```

```
% Fourthstring.m 5/6/2007 Devon Lafferty
%
% This program tries to solve the equation governing an upright stiff
% string with an oscillating base.
% A fourth order space finite diffference method was used, with
equations
% gained from Maple. However, the required timestep to
% overcome the CFL condition seems to be too large for any reasonable
data.
% The memory is overflowed if you use only one matrix, so two matrices
were
% used, one to store the data, sol[t,x), the other to compute v[t,x].
% It uses a second-order central differences in time.

clear;
L=1;

Nx=10;
dx=L/Nx;
T=.01;       % final time
Nt=3500000;      %number of actual time steps computed
dt=T/Nt;
alpha=(dt/dx)^2;
tstep=.0001;    %time step to actually be shown
nstep=int32(tstep*Nt/T);
solstep=int32(T/tstep);



%set physical constants
g=9.8;          % acceleration due to gravity in m/s^2
w=30;          % frequency of oscillation in radians/s
m=.4;           % mass per unit length in kg/m



%find the appropriate constants
B=1.2;
e=0;
del=g/L;

C=alpha*B*alpha/dt^2;
a=alpha;



% set up spatial and temporal grid
x=[0:Nx]*dx;
t=[0:Nt]*dt;

% solution will be arranged in a matrix Sol, with intermediate matrix
% v.
v=zeros(nstep,Nx+1);
sol=zeros(solstep,Nx+1);
```

```
f=.001*x.^2;

% initial velocity u_t(x,0)=g(x)
deri=0*x;
% use the initial conditions to figure out solution
% for t=0, t=dt.
v(1,:)=f;
sol(1,:)=f;
v(2,:)=f+dt*deri;

v(2,1)=0;

breaker=0;                          %breaks the solution when it gets too
big

% This uses 2 for loops, one to calculate the main time step, the other
% to calculate the intermediate time steps.
lastk=0;
for k=2:solstep
    for j=3:nstep
      % find solution at next time
      for i=4:(Nx-2)
          G=-(del-e*w^2*cos(w*(double(j)-1)*dt));
          Q=(1-i*dx);
          v(j,i)=C*v(j-1,i+3)/(6*dx^2)+(-G*dx^3*a+G*dx^2*a*Q-24*C)*v(j-
1,i+2)...
              /(12*dx^2)+(8*G*dx^3*alpha-16*G*dx^2*alpha*Q+78*C)*v(j-
1,i+1)/(12*dx^2)...
              +(24*dx^2-112*C+30*G*dx^2*alpha*Q)*v(j-1,i)/(12*dx^2)+...
              (-16*G*dx^2*alpha*Q+78*B*alpha-8*G*dx^3*alpha)*v(j-1,i-
1)/(12*dx^2)...
              +(G*dx^3*a+G*dx^2*a*Q-24*C)*v(j-1,i-2)/(12*dx^2)...
              +C*v(j-1,i+3)/(6*dx^2)-v(j-2,i);
          if abs(v(j,i))>1
            breaker=1;      %set up to break for nonphysical answers
over 1
          end
      end
      % Dirichlet and 2nd Neumann boundary condition at x=0
      v(j,1)=0;
      v(j,2)=1/4*v(j,3);
      G=-(del-e*w^2*cos(w*(double(j)-1)*dt));
      Q=(1-3*dx);
      v(j,3)=-C*(v(j-1,3+2)+v(j-1,3-2))+(alpha*G*(Q-dx/2)+4*C)*v(j-
1,3+1)+...
          (alpha*G*(Q+dx/2)+4*C)*v(j-1,3-1)+(2-2*alpha*Q*G-6*C)*v(j-
1,3)-v(j-2,3);
      %2nd and 4th order Neumann boundary condition at x=L
      Q=(1-(Nx-1)*dx);
      v(j,Nx-1)=-C*(v(j-1,3+2)+v(j-1,3-2))+(alpha*G*(Q-dx/2)+4*C)*v(j-
1,3+1)+...
          (alpha*G*(Q+dx/2)+4*C)*v(j-1,3-1)+(2-2*alpha*Q*G-6*C)*v(j-
1,3)-v(j-2,3);
      v(j,Nx)=1/13*(40*v(j,Nx-1)-45*v(j,Nx-2)+22*v(j,Nx-3)-4*v(j,Nx-
4));
```

```
      v(j,Nx+1)=5/2*v(j,Nx)-2*v(j,Nx-1)+1/2*v(j,Nx-2);
      if breaker==1
          break;
      end
    lastj=j;
    end

    if breaker==1
        break;
    end;
    sol(k,:)=v(nstep,:);    %gives sol the data from v
    v(1,:)=v(nstep-1,:);    %resets v
    v(2,:)=v(nstep,:);

    lastk=k;                %saves the point it stops
end

%The data is then plotted as a movie
figure(1)
hold off

for j=1:lastk
  plot(sol(j,:),x)
  axis([-.5 .5 0 L+.1])
  text(.1,1,[' Time = ',num2str(t(j*nstep))]);
  M(j)=getframe;
end
```