The Tacoma Narrows Bridge

By: Adam Headley 998017147 27 May 2015

Abstract

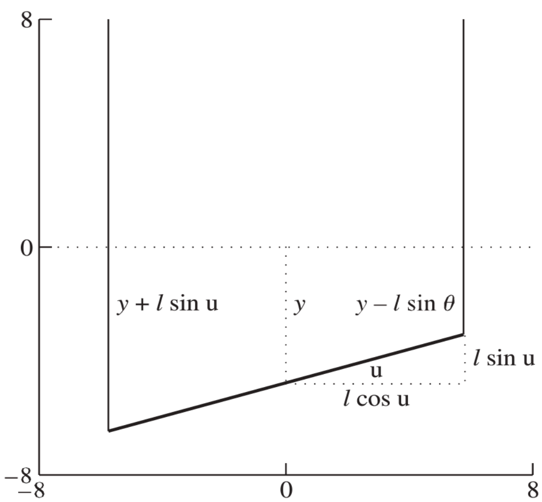
This project is an example of experimental mathematics. The equations are too difficult to derive closed-form solutions, and even too difficult to prove qualitative results. Equipped with reliable ODE solvers, numerical trajectories for various parameter settings can illustrate the types of phenomena available to this model. Differential equation models can predict behavior and shed light on mechanisms in scientific and engineering problems. These problems and the following activities are borrowed from Sauer.

Introduction

McKenna and Tauma recently proposed a mathematical model that attempts to reproduce the Tacoma Narrows Bridge conditions and breakdown. The goal of this model was to explain how torsional (twisting) oscillations can be magnified by strictly vertical forces.

Consider a roadway of width $2l$ hanging between two suspended cables. For this model the bridges links dimension I will be ignored, only taking into consideration a two-dimensional slice of the bridge. The roadway hangs at a certain equilibrium height at rest due to gravity; let y denotes the distance from the roadway center of mass to its equilibrium position and angle θ of the roadway with the horizontal.

Figure 1: (Left) Cross-Section of Modeled Tacoma Narrows Bridge



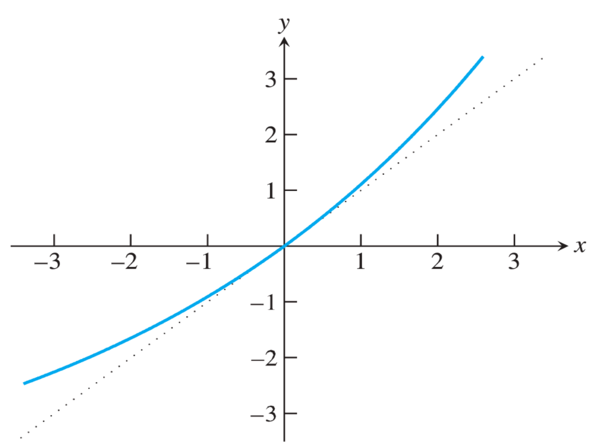


Figure 2: (Right) Exponential Hooke's Law curve $f(y) = \frac{K}{a}(e^{ay} - 1)$.

Hooke's Law postulates a linear response, implying that the restoring force applied by the cables will be proportional to the deviation from equilibrium. Exponential Hooke's Law: $f(y) = \frac{K}{a}(e^{ay} - 1)$. Let θ be the angle the roadway makes with the horizontal. There are two suspension cables, deviates $y - l\sin\theta$ and $y + l\sin\theta$ from equilibrium, respectively. Given a viscous damping term assume that it is proportional to the velocity. Using Newton's law$F = ma$ and denoting Hooke's constant K, the equations of motion for y and θ are as follows:

$$y'' = -dy' - \left[\frac{K}{m}(y - l\sin\theta) + \frac{K}{m}(y + l\sin\theta)\right]$$
$$\theta'' = -dy + \frac{3\cos\theta}{l}\left[\frac{K}{m}(y - l\sin\theta) - \frac{K}{m}(y + l\sin\theta)\right]$$

Hooke's Law is designed for springs, where the restoring force is more or less equal whether the springs are compressed or stretched. McKenna and Tauma hypothesize that the cables pull back with more force when stretched than they push back with when compressed. (Think of a string as an example.) The linear Hooke's Law restoring force $f(y) = Ky$ is replaced with a nonlinear force $f(y) = \frac{K}{a}(e^{ay} - 1)$. Both functions have the same slope K at $y = 0$; but for the nonlinear force, a positive y (stretched cable) causes a stronger restoring force than the corresponding negative y (slackened cable). Making this replacement in the preceding equation yields:

$$y'' = -dy - \frac{K}{ma}\left[e^{a(y - l\sin\theta)} - 1 + e^{a(y + l\sin\theta)} - 1\right]$$
$$\theta'' = -d\theta' + \frac{3\cos\theta}{l}\frac{K}{ma}\left[e^{a(y - l\sin\theta)} - e^{a(y + l\sin\theta)}\right]$$

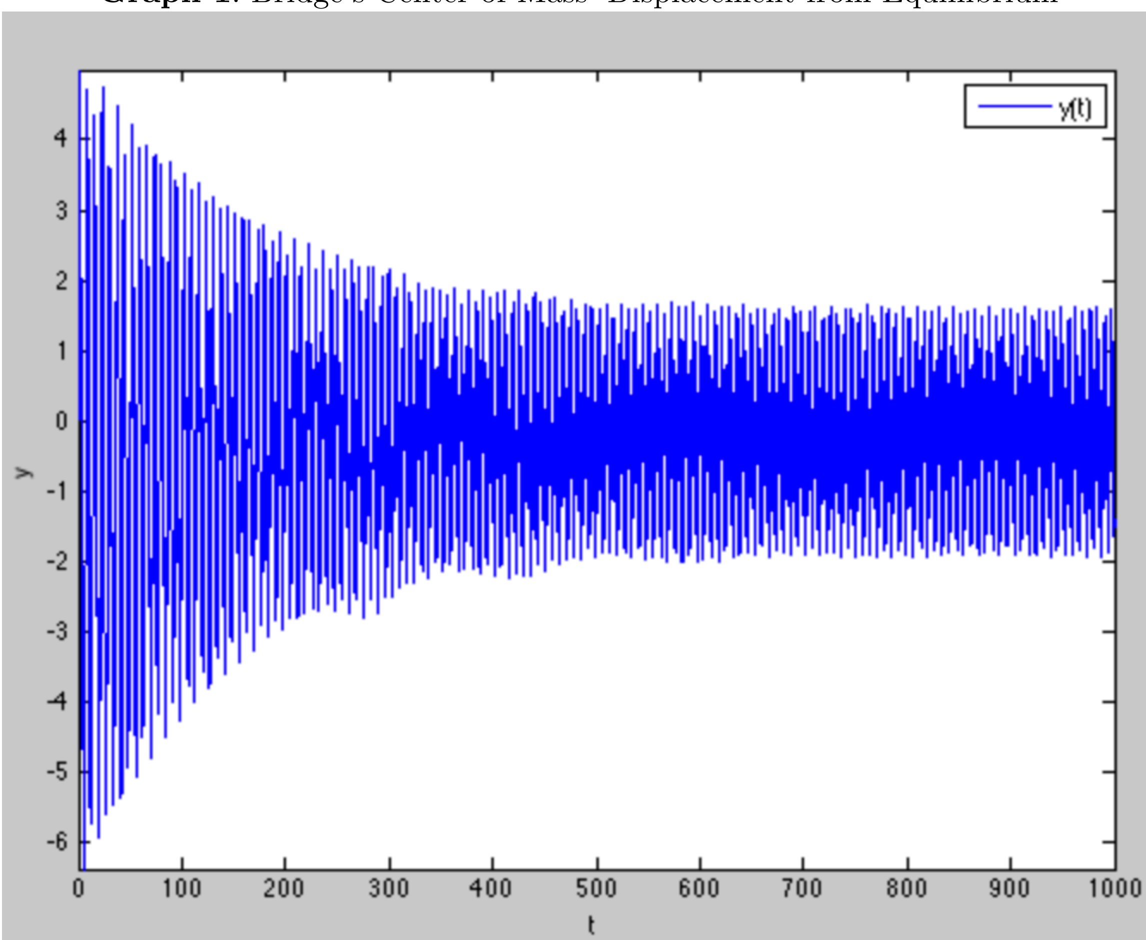
The state $y = y' = \theta = \theta' = 0$ is rests at equilibrium. Turning on the wind and add the forcing term $0.2W\sin(\omega t)$ to the right-hand side of the y equation, where W is the wind speed in km/hr. This adds a strictly vertical oscillation to the bridge.

Useful estimates for the physical constants can be made. The mass of a one-foot length of roadway was about 2500 kg, and the spring constant K has been estimate at 1000 Newtons. The roadway was about 12 meters wide. For this simulation, the damping coefficient was set at $d = 0.01$, and the Hooke's nonlinearity coefficient $a = 0.2$. An observer counted 38 vertical oscillations of the bridge in one minute shortly before the collapse which implies that $\omega = 2\pi\frac{38}{60}$. These coefficients are only guesses, but they suffice to show ranges of motion that tend to match photographic evidence of the bridge's final oscillations. MATLAB code that runs the model is in the Appendix.

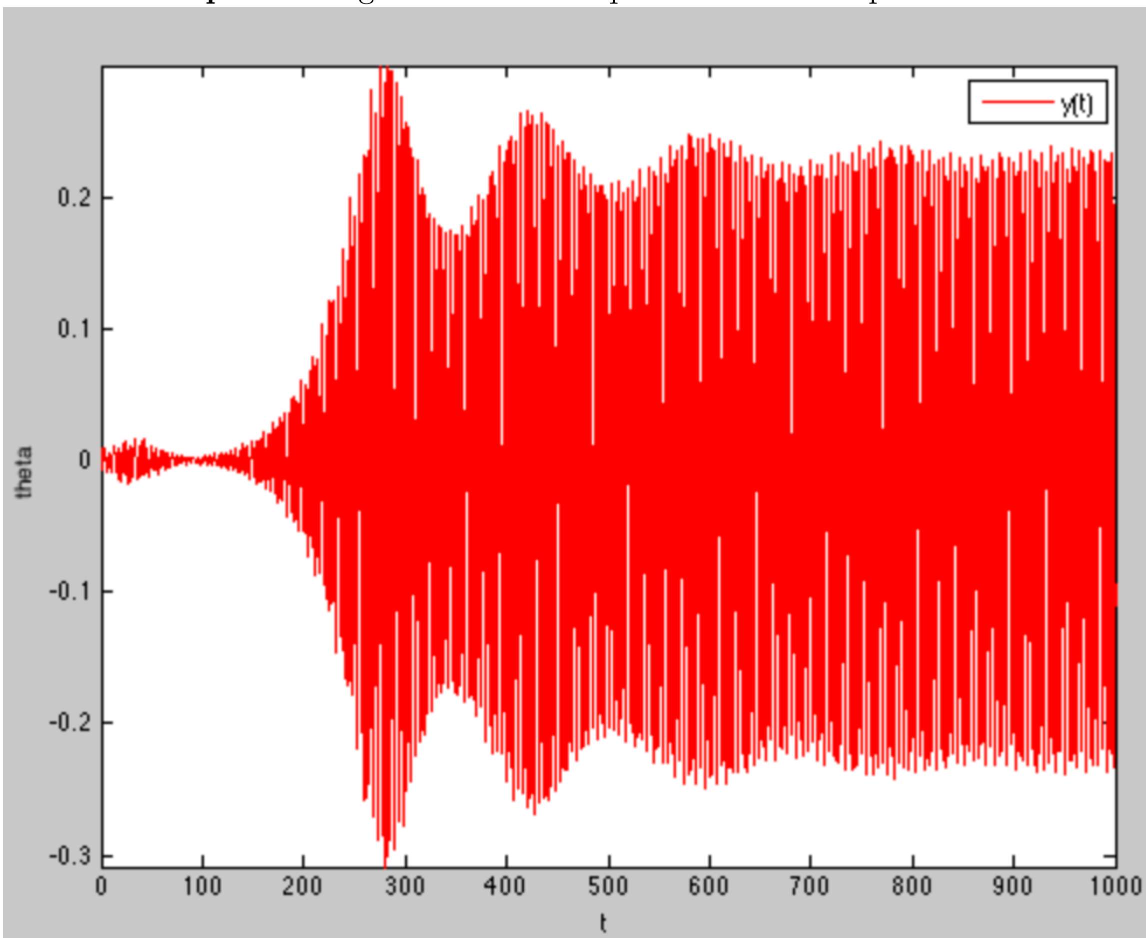
A Running the MATLAB code **tacoma1.m** with wind speed $W = 80$ km/hr and initial conditions $y = y' = \theta' = 0$, $\theta = 10^{-2}$. In the torsional dimension, the bridge is stable if small disturbances in θ die out and unstable if small disturbances grow far beyond the original size. Observing the script.m (%% SCRIPT 1. block), watching the model bridge oscillate, it takes a while however, the small disturbances in θ grow far beyond the original size for $W = 80$ km/hr.

B Replacing the Trapeziod Method with the Runge-Kutta Method of Order 4 (**trapstep.m** changes to **rk4step.m** in **tacoma2.m**) and running the same initial conditions as in **tacoma1.m**, wind speed $W = 80$ km/hr and initial conditions $y = y' = \theta' = 0$, $\theta = 10^{-2}$. The following figures below describe the y over time and θ over time. In the torsional dimension, the bridge is stable if small disturbances in θ die out and unstable if small disturbances grow far beyond the original size. Reference the script.m (%% SCRIPT 2. block).

Graph 1: Bridge's Center of Mass' Displacement from Equillibrium



Graph 2: Bridge's Torsional Displacement from Equillibrium



In Graph 1, the bridge's center of mass' displacement starts oscillating with large amplitude at a low frequency and then oscillates with a smaller amplitude at a higher frequency. The vertical oscillation is extreme in the beginning but then just before $t = 300$ starts to shorten and become more frequent. In Graph 2, the bridge's torsional displacement starts small and then grows extremely large just after $t = 100$ reaching its peak amplitude just before $t = 300$. There seems to be a correlation here. Some of the energy from the vertical forces get transferred to the torsional forces changing θ more drastically. Under close observation one can spot a resonance in the bridge's oscillations of both graphs, in the darker shaded region.

C

The system is torsionally stable for $W = 50\ \text{km/hr}$. The script.m (%% SCRIPT 3. Block) finds the magnification factor for initial conditions $y = y' = \theta' = 0$,

$\theta(0) = 10^{-3}$ by using MATLAB code **tacoma3**.m which takes in **rk4step3**.m which takes in **ydot3**.m. The magnification factor is expressed as

$$MF = \frac{\max(\theta(t))}{\theta(0)}$$

the greatest angle divided by initial angle. For the specified initial conditions, the consistency of the magnification factors found from the initial angles are shown below:

Table 1: Consistency of the Magnification Factor for Initial Angles

Initial Angle θ	**Magnification Factor ≈**	**Consistent?**
10^{-3} to 10^{-16}	$[19.1496572583, 19.6091562004]$	Yes
10^{-17}	9.13686692	No
10^{-18}	202.9913077372	No
10^{-19}	356.2421906764	No
10^{-20} and on	1	Yes

As can be see in Table 1, the magnification factor is not approximately consistent for all values of $\theta(0) = 10^{-3}, 10^{-4}, 10^{-5}, \ldots$, specifically $\theta(0) = 10^{-17}, 10^{-18}, 10^{-19}$. $\theta(0) = 10^{-20}$ is very small to have such a nice looking number, this is most likely from truncation error while using $\theta(0)$

D

To find a minimum wind speed, W, with $\theta(0) = 10^{-3}$ such that the magnification factor of 100 or more is produced, **ydot4**.m and **rk4step4**.m were created to take in one more variable, W, and perform the same as **ydot3**.m and **rk4step3**.m otherwise. The **script**.m (%% SCRIPT 4. Block) finds the magnification factor for small initial angle, $\theta(0) = 10^{-3}$, using MATLAB code **tacoma4**.m. The loop in the script runs through values of W stepping by size s and stops when magnification factor is 100 or more.

Table 2: Tests Ran to Determine a Convergent Wind Speed

Run	Initial Wind Speed	Stepsize	Wind Speed	Magnification Factor
1st	50	1	59	102.2220310286
2nd	58	0.1	59.0	102.2220310286
3rd	58.9	0.01	58.95	100.1476473096
4th	58.94	0.001	58.947	100.0247617444
5th	58.946	0.0001	58.9464	100.0002059828

The wind speed values are converging to $W \approx 58.9463\ \text{km/hr}$ as the magnification factor consistently converges to $MF = 100$.

E Implementing the Secant Method, the minimum wind speed in part D is computed to within 0.5×10^{-3} in the **script**.m (%% SCRIPT 5. Block). Calculates magnification factors using **tacoma4**.m for 2 different wind speeds and combines them using the Secant Method to create a new wind speed using MATLAB code **tacoma5**.m. The Secant Method approximates the derivative of the magnification factor with respect to the wind speed multiplies it by the magnification factor of the second point and then subtracts it from the second wind speed, calculating a new wind speed:

$$W_3 = W_2 - \frac{MF_2(W_2 - W_1)}{MF_2 - MF_1}$$

Where W_3 converges to the minimum wind speed within the given tolerance, $58.9463949662\,\frac{\text{km}}{\text{hr}} \approx 58.9464\ \frac{\text{km}}{\text{hr}}$ after 9 iterations of the Secant Method.

F When trying larger values of W, not all extremely small initial angles (10^{-3}) eventually grow to catastrophic size. The bridge breaks when the magnification factor is greater than 100. This occurs for wind speeds, W, approximately greater than $58.9464\ \text{km/hr}$ as shown in part E. The **script**.m (%% SCRIPT 6. Block) uses **tacoma4**.m to calculate the magnification factor for larger values of W. However, wind speed anomalies that allow the bridge to stay in good condition occur in [78.8,79.6] by increments of $.1$. These anomalies are caused by the damping forces dissipating the energy that is contributed by a resonant excitation to the system at its natural frequency.

Conclusion

If a sinusoidal driving force is applied at the resonant frequency of the oscillator, then its motion will build up in amplitude to the point where it is limited by the damping forces on the system. If the damping forces are small, a resonant system can build up to amplitudes large enough to be destructive to the system. Since a resonant excitation will continue to contribute energy to the system at its natural frequency, the amplitude of oscillation will continue to grow until the damping forces dissipate that energy, or until something destructive happens to the system. (Driven Oscillator & Resonant Excitation, Hyper Physics)

McKenna suggests that perhaps the replacement of the linear small angle approximation for the motion of the bridge by the proper trigonometric functions could provide a sufficient non-linearity to cause the bridge to collapse. By calculating the response of such a non-linear oscillator with a force containing constant frequency and amplitude, it can be argued that the non-linearities of the trigonometric functions alone can allow for the same kind of bimodal response, with large amplitude oscillations when the linear case may not. Although these models provide a more complete analysis of motion, this analysis ignores the cause of the driving forces entirely. The sinusoidal forces applied on the bridge are assumed to have a constant period and amplitude to drive the resonance. Without understanding the origin and fluctuation of these forces, introducing the non-linearities gains an incomplete insight into the cause of the collapse of the bridge. (Tacoma Bridge Failure, Green & Unruh, University of British Columbia)

Appendix

(12 functions, 6 scripts)

SCRIPT (with 6 blocks)

```
%% SCRIPT 1.

% Running tacoma.m with wind speed W = 80 km/hr
% initial conditions y = y' = theta' = 0, theta = 0.001.

% The bridge is stable in the torsional dimension
% if small disturbances in theta die out

% The bridge is unstable in the torsional dimension
% if the small disturbances grow far beyond the originial size

a = 0;
b = 1000;

y1 = 0;
y2 = 0;
theta1 = 10^-2;
theta2 = 0;

tacoma1([a b],[y1 y2 theta1 theta2],.04,3);

% For W = 80,
% The small disturbances in theta grow far beyond the original size.
% Therefore, the bridge is unstable and breaks.


%-----------------------------------------------------------------%
%% SCRIPT 2.

% Replaces the trapeziod method with the Runge-Kutta Method of Order 4
% trapstep.m -> rk4step.m
% Plots y(t) and theta(t)

a = 0;
```

```
b = 1000;

y1 = 0;
y2 = 0;
theta1 = 10^-2;
theta2 = 0;

[time,z1,z3] = tacoma2([a b],[y1 y2 theta1 theta2],.04,3);

% y vs. time
figure
plot(time,z1,'b')
hold on
legend('y(t)')
z1min = min(z1);
z1max = max(z1);
axis([a b z1min z1max])
xlabel('t')
ylabel('y')

% theta vs. time
figure
plot(time,z3,'r')
hold on
legend('y(t)')
z3min = min(z3);
z3max = max(z3);
axis([a b z3min z3max])
xlabel('t')
ylabel('theta')


%-----------------------------------------------------------------%
%% SCRIPT 3.

format long;

% The system is torsionally stable for W = 50 km/hr.
% Finds magnification factor for small initial angle, theta(0) = 10^-3
% Magnification Factor = Maximum Angle theta(t) / Initial Angle theta(0)

a = 0;
b = 1000;

```

```
y1 = 0;
y2 = 0;
theta2 = 0;

tend = 23;
MF2 = zeros(tend-2,1);

% loop
for t = 3:tend
    theta1 = 10^-t;

    [time,z1,z3] = tacoma3([a b],[y1 y2 theta1 theta2],.04,3);
    close

    % Magnification Factor
    MF2(t-2) = max(abs(z3))/z3(1);
    fprintf('Magnification Factor with starting theta = 10^-%d is %.10f \n',t,MF2(t-2))
end

% The Magnification Factor is approximately consistent @ MF = 19 (approx.)
% for initial theta from 10^-3 to 10^-16.

% The Magnification Factor is approximately consistent @ MF = 1
% for initial theta from 10^-20 on.


%-----------------------------------------------------------------%
%% SCRIPT 4.

% Finding minimum W with theta(0) = 10^(-3)
% producing a magnification factor of 100 or more
% ydot4 takes in one more variable, W, and performs the same as ydot

a = 0;
b = 1000;

y1 = 0;
y2 = 0;
theta1 = 10^-3;
theta2 = 0;

% loop looks at values of W stepping by size s
% stops when MF is 100 or more
```

```
W  = 58.9;       % starting value
s  = .001;       % step size of W
MF =  1;         % Initialize MF to run while loop
while MF < 100
    [~,~,z3]=tacoma4([a b],[y1 y2 theta1 theta2],.04,3,W);
    close
    MF = max(abs(z3))/z3(1);
    if MF < 100
        W = W + s;
    end
end
fprintf('For W = %.10f, the Magnification Factor is %.10f < 100 \n', W, MF)

% Tests
% First  run: W = 50,     s = 1      -> W = 59,      …
                                        MF = 102.2220310286
% Second run: W = 58,     s = 0.1    -> W = 59.0,    …
                                        MF = 102.2220310286
% Third  run: W = 58.9,   s = 0.01   -> W = 58.95,   …
                                        MF = 100.1476473096
% Fourth run: W = 58.94,  s = 0.001  -> W = 58.947,  …
                                        MF = 100.0247617444
% Fifth  run: W = 58.946, s = 0.0001 -> W = 58.9464,
                                        MF = 100.0002059828


%---------------------------------------------------------------%
%% SCRIPT 5.

% Method for calculating minimum windspeed to within 0.5x10^-3 km/hr.

W1 = 80;
W2 = 50;

% Secant Method
[W3,iterations] = tacoma5(W1,W2);

% W3 converges to 58.946394966 km/hr in ## iterations.


%---------------------------------------------------------------%
%% SCRIPT 6.

```

```
% When trying larger values of W,
% not all extremely small initial angles (10^-3) eventually grow
% to catastrophic size.

a = 0;
b = 1000;

y1 = 0;
y2 = 0;
theta1 = 10^-3;
theta2 = 0;

s  = .2;         % step size of W
MF =  1;         % Initialize MF to run while loop
for W = 50:s:150
    [~,~,z3] = tacoma4([a b],[y1 y2 theta1 theta2],.04,3,W);
    close
    MF = max(abs(z3))/z3(1);
    if MF < 100
        fprintf('Good Condition, For W = %.1f, the Magnification Factor is %.10f \n', W, MF)
    else
        fprintf('Broke Condition, For W = %.1f, the Magnification Factor is %.10f \n', W, MF)
    end
end

% For an initial he bridge breaks when the Magnification Factor is greater than 100
% This occurs for wind speeds, W, greater than 58.9464 km/hr
% Wind speed anomalies that allow the bridge to stay in
% good condition occur in [78.6,79.8] by increments of .2.  Because the frequency
```

TACOMA#.m

```
function tacoma1(inter,ic,h,p)
%Program 6.6 Animation program for bridge using IVP solver

%Inputs: int = [a b] time interval,
%         ic = [y(1,1) y(1,2) y(1,3) y(1,4)],
%         h = stepsize
%         p = steps per point plotted

%Calls a one-step method: trapstep.m

%Example usage: tacoma([0 1000],[1 0 0.001 0],.04,3)

% clear figure window
clf

% plot n points
a = inter(1);
b = inter(2);
n = ceil((b-a)/(h*p));

% Initial Conditions
y(1,:) = ic;
t(1) = a;
len = 6;
set(gca,'XLim',[-8 8],'YLim',[-8 8], ...
    'XTick',[-8 0 8],'YTick',[-8 0 8], ...
    'Drawmode','fast','Visible','on','NextPlot','add');

% initialize broken criterion
broke = 0;

tic

% clear screen
cla;

% make aspect ratio 1-1
axis square
road    = line('color','b','LineStyle','-','LineWidth',5,...
          'erase','xor','xdata',[],'ydata',[]);
lbroke = line('color','r','LineStyle','-','LineWidth',5,...
          'erase','xor','xdata',[],'ydata',[]);
```

```
rbroke = line('color','r','LineStyle','-','LineWidth',5,...
              'erase','xor','xdata',[],'ydata',[]);
lcable = line('color','k','LineStyle','-','LineWidth',1,...
              'erase','xor','xdata',[],'ydata',[]);
rcable = line('color','k','LineStyle','-','LineWidth',1,...
              'erase','xor','xdata',[],'ydata',[]);

Y = figure;
T = figure;

for k = 1:n

    for i = 1:p
        t(i+1) = t(i)+h;

        % 1 step method
        y(i+1,:) = trapstep(t(i),y(i,:),h);

        time(k) = t(i);
    end
    y(1,:) = y(p+1,:);
    t(1) = t(p+1);
    z1(k) = y(1,1);
    z3(k) = y(1,3);

    c = len*cos(y(1,3));
    s = len*sin(y(1,3));

    % Magnification Factor
    MF = max(z3)/z3(1);

    % Breaking the bridge
    if MF > 100
        broke = 1;
    end

    if broke == 1
        set(road,'xdata',[-c c],'ydata',[0 0])
        set(lbroke,'xdata',[-c -c],'ydata',[-s-y(1,1) s-y(1,1)-len/2])
        set(rbroke,'xdata',[c c],'ydata',[s-y(1,1) s-y(1,1)-len/2])
    else
        set(road,'xdata',[-c c],'ydata',[-s-y(1,1) s-y(1,1)])
    end
    set(lcable,'xdata',[-c -c],'ydata',[-s-y(1,1) 8])
```

```
    set(rcable,'xdata',[c c],'ydata',[s-y(1,1) 8])
    drawnow; pause(h)

    % Graph y(t) and theta(t)
    figure(Y);
    plot(time,z1,'b')
    figure(T);
    plot(time,z3,'r')

end
toc

end
```

```
function [time,z1,z3] = tacoma2(inter,ic,h,p)
%Program 6.6 Animation program for bridge using IVP solver

%Inputs: int = [a b] time interval,
%        ic = [y(1,1) y(1,2) y(1,3) y(1,4)],
%        h = stepsize
%        p = steps per point plotted

%Calls a one-step method: rk4step.m

%Example usage: tacoma([0 1000],[1 0 0.001 0],.04,3)

% clear figure window
clf

% plot n points
a = inter(1);
b = inter(2);
n = ceil((b-a)/(h*p));

% Initial Conditions
y(1,:) = ic;
t(1) = a;
len = 6;

tic
for k = 1:n

    for i = 1:p
        t(i+1) = t(i)+h;

```

```
        % 1 step method
        y(i+1,:) = rk4step(t(i),y(i,:),h);

        % time vector for plots
        time(k) = t(i);
    end

    y(1,:) = y(p+1,:);
    t(1) = t(p+1);
    z1(k) = y(1,1);
    z3(k) = y(1,3);

end
toc

end
```

```
function [time,z1,z3] = tacoma3(inter,ic,h,p)
%Program 6.6 Animation program for bridge using IVP solver

%Inputs: int = [a b] time interval,
%        ic = [y(1,1) y(1,2) y(1,3) y(1,4)],
%        h = stepsize
%        p = steps per point plotted

%Calls a one-step method: rk4step.m

%Example usage: tacoma([0 1000],[1 0 0.001 0],.04,3)

% clear figure window
clf

% plot n points
a = inter(1);
b = inter(2);
n = ceil((b-a)/(h*p));

% Initial Conditions
y(1,:) = ic;
t(1) = a;
len = 6;

tic
for k = 1:n

```

```
    for i = 1:p
        t(i+1) = t(i)+h;

        % 1 step method
        y(i+1,:) = rk4step3(t(i),y(i,:),h);

        % time vector for plots
        time(k) = t(i);
    end

    y(1,:) = y(p+1,:);
    t(1) = t(p+1);
    z1(k) = y(1,1);
    z3(k) = y(1,3);

end
toc

end
```

```
function [time,z1,z3] = tacoma4(inter,ic,h,p,W)
%Program 6.6 Animation program for bridge using IVP solver

%Inputs: int = [a b] time interval,
%         ic = [y(1,1) y(1,2) y(1,3) y(1,4)],
%         h = stepsize
%         p = steps per point plotted

%Calls a one-step method: trapstep.m or rk4.m

%Example usage: tacoma([0 1000],[1 0 0.001 0],.04,3)

% clear figure window
clf

% plot n points
a = inter(1);
b = inter(2);
n = ceil((b-a)/(h*p));

% Initial Conditions
y(1,:) = ic;
t(1) = a;
len = 6;

```

```
for k = 1:n
    for i = 1:p
        t(i+1) = t(i)+h;

        % 1 step method
        y(i+1,:) = rk4step4(t(i),y(i,:),h,W);

        % time vector for plots
        time(k) = t(i);
    end

    y(1,:) = y(p+1,:);
    t(1) = t(p+1);
    z1(k) = y(1,1);
    z3(k) = y(1,3);
end

end
```

```
function [W,iterations] = tacoma5(W1,W2)
%tacoma5 Calculates magnification factors using tacoma4.m
% for 2 different wind speeds.
% Combines them using the Secant Method to create a new wind speed.
% Secant Method computes the derivative of the magnification factor
% with respect to the wind speed
% These wind speeds converge

% Initial Conditions
inter = [0 1000];
ic = [0 0 0.001 0];
h = .04;
p = 3;

% Comparing Iterations
TOL = 100;
iteration = 0;

while TOL > 0.5*10^(-3)

    [~,~,z32] = tacoma4(inter,ic,h,p,W2);
    [~,~,z31] = tacoma4(inter,ic,h,p,W1);
    close

```

```
    MF2 = 100 - max(z32)/z32(1);
    MF1 = 100 - max(z31)/z31(1);

    % Secant Method
    W3 = W2 - ((MF2)*(W2-W1))/(MF2-MF1);
    fprintf('Secant Method produces new wind speed, W = %.10f \n',W3)

    % Update Tolerance
    TOL = max(abs(W3-W1),abs(W3-W2));

    if TOL <= 0.5*10^(-3)
        W = W3;
        iterations = iteration;
        return
    end

    W1=W2;
    W2=W3;

    iteration = iteration + 1;

end

end
```

<u>TRAPSTEP.m</u>

```
function y = trapstep(t,x,h)
% one step of the Trapezoid Method

z1 = ydot(t,x);
g  = x + h*z1;
z2 = ydot(t+h,g);
y = x + h*(z1+z2)/2;
end
```

RK4STEP#.m

```
function y = rk4step(t,w,h)
%rk4step: 1 step of the Runge-Kutta's Method of Order 4

K1 = ydot(t,w);
K2 = ydot(t+h/2,w+h*K1/2);
K3 = ydot(t+h/2,w+h*K2/2);
K4 = ydot(t+h,w+h*K3);
y = w+h*(K1+2*K2+2*K3+K4)/6;
end
```

```
function y = rk4step3(t,w,h)
%rk4step3: 1 step of the Runge-Kutta's Method of Order 4

K1 = ydot3(t,w);
K2 = ydot3(t+h/2,w+h*K1/2);
K3 = ydot3(t+h/2,w+h*K2/2);
K4 = ydot3(t+h,w+h*K3);
y = w+h*(K1+2*K2+2*K3+K4)/6;
end
```

```
function y = rk4step4(t,w,h,W)
%rk4: 1 step of the Runge-Kutta's Method of Order 4

K1 = ydot4(t,w,W);
K2 = ydot4(t+h/2,w+h*K1/2,W);
K3 = ydot4(t+h/2,w+h*K2/2,W);
K4 = ydot4(t+h,w+h*K3,W);
y = w+h*(K1+2*K2+2*K3+K4)/6;
end
```

YDOT#.m

```
function ydot = ydot(t,y)
%ydot Initial Conditions

% damping coefficient
d = 0.01;

len = 6;
a = 0.2;
W = 80;
omega = 2*pi*38/60;

% e^(a(y-l*sin(theta)))
e1 = exp(a*(y(1)-len*sin(y(3))));

% e^(a(y+l*sin(theta)))
e2 = exp(a*(y(1)+len*sin(y(3))));

% y'
ydot(1) = y(2);

% y'' = -d*y' - (K/m)*(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/a
%        + (forcing term) % adds a strictly vertical oscillation to bridge
ydot(2) = -d*y(2) - 0.4*(e1+e2-2)/a + 0.2*W*sin(omega*t);
% d = 0.01, K/m = 0.4, forcing term = 0.2*W*sin(omega*t)

% theta'
ydot(3) = y(4);

% theta'' = -d*theta' - 3*(K/m)*cos(theta')
%                      *(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/l*a
ydot(4) = -d*y(4) + 1.2*cos(y(3))*(e1-e2)/(len*a);
% 3K/m = 1.2 => K/m = 0.4

end
```

```
function ydot = ydot3(t,y)
%ydot3 Initial Conditions

```

```
% damping coefficient
d = 0.01;

len = 6;
a = 0.2;
W = 50; % Torsionally stable
omega = 2*pi*38/60;

% e^(a(y-l*sin(theta)))
e1 = exp(a*(y(1)-len*sin(y(3))));

% e^(a(y+l*sin(theta)))
e2 = exp(a*(y(1)+len*sin(y(3))));

% y'
ydot(1) = y(2);

% y'' = -d*y' - (K/m)*(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/a
%       + (forcing term) % adds a strictly vertical oscillation to bridge
ydot(2) = -d*y(2) - 0.4*(e1+e2-2)/a + 0.2*W*sin(omega*t);
% d = 0.01, K/m = 0.4, forcing term = 0.2*W*sin(omega*t)

% theta'
ydot(3) = y(4);

% theta'' = -d*theta' - 3*(K/m)*cos(theta')
%                      *(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/l*a
ydot(4) = -d*y(4) + 1.2*cos(y(3))*(e1-e2)/(len*a);
% 3K/m = 1.2 => K/m = 0.4

end
```

```
function ydot = ydot4(t,y,W)
%ydot4 Initial Conditions

% damping coefficient
d = 0.01;

len = 6;
a = 0.2;
omega = 2*pi*38/60;

```

```
% e^(a(y-l*sin(theta)))
e1 = exp(a*(y(1)-len*sin(y(3))));

% e^(a(y+l*sin(theta)))
e2 = exp(a*(y(1)+len*sin(y(3))));

% y'
ydot(1) = y(2);

% y'' = -d*y' - (K/m)*(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/a
%       + (forcing term) % adds a strictly vertical oscillation to bridge
ydot(2) = -d*y(2) - 0.4*(e1+e2-2)/a + 0.2*W*sin(omega*t);
% d = 0.01, K/m = 0.4, forcing term = 0.2*W*sin(omega*t)

% theta'
ydot(3) = y(4);

% theta'' = -d*theta' - 3*(K/m)*cos(theta')
%                      *(e^(a(y-l*sin(theta)))-e^(a(y+l*sin(theta))))-2)/l*a
ydot(4) = -d*y(4) + 1.2*cos(y(3))*(e1-e2)/(len*a);
% 3K/m = 1.2 => K/m = 0.4

end
```