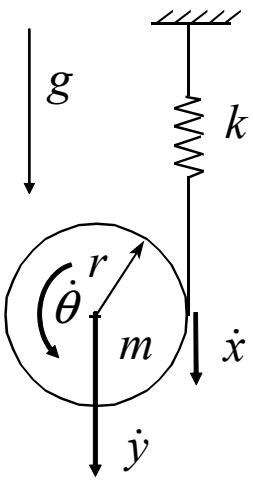
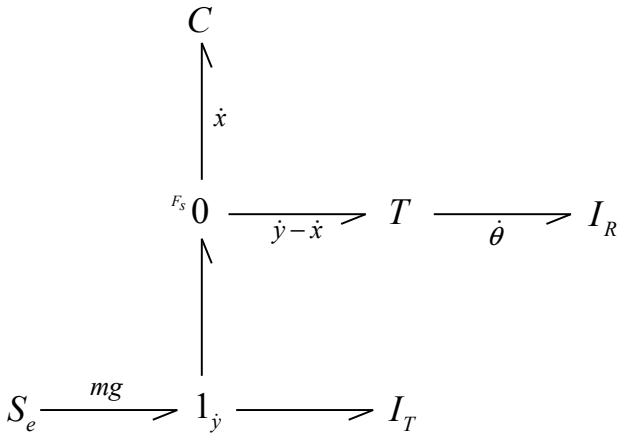
1. A cord with a spring is wrapped around a cylinder. No horizontal excitation is imposed, so the cord and the spring remain vertical.

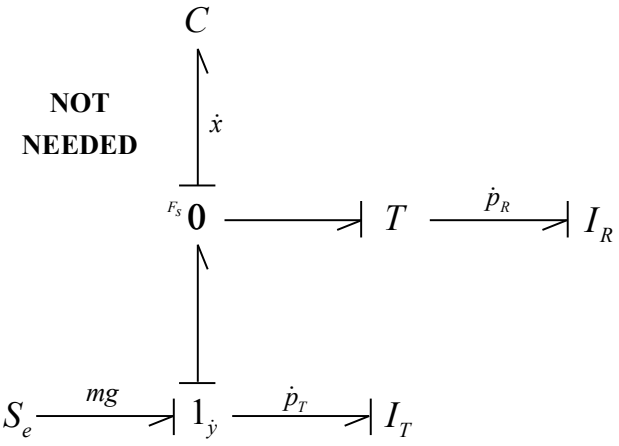


a. Define the key variables and parameters placing them on the physical model shown to the left.

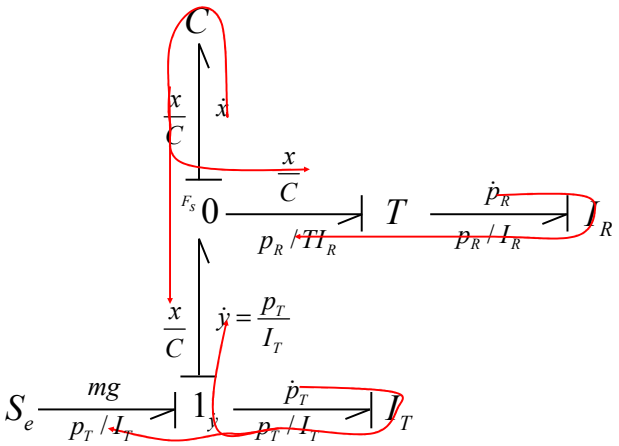
$$C = 1/k \qquad T = 1/r$$
$$I_T = m \qquad I_R = mr^2/2$$

b. Draw a bond graph model below for the system shown above right including all variables and parameters previously defined. (DO NOT WASTE YOUR TIME CONVERTING THE BOND GRAPH MODEL INTO A SET OF DIFFERENTIAL EQUATIONS.)





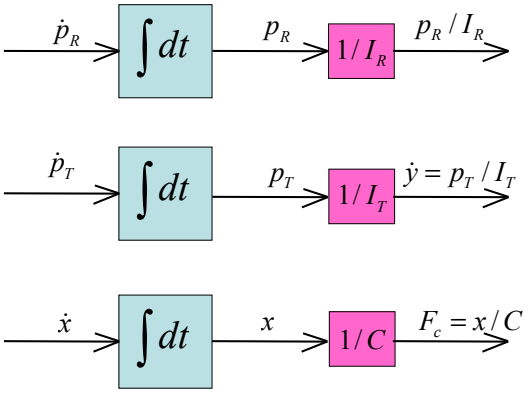
c. What is the order of your model? 3 (there are three state variables; x, p_R and p_T.



$$\left.\begin{aligned}\dot{p}_R &= \frac{x}{TC}\\ \dot{x} &= \frac{p_T}{I_T} - \frac{p_R}{TI_R}\\ \dot{p}_T &= mg - \frac{x}{C}\end{aligned}\right\}\ \text{State equations}$$

$$\left.\begin{aligned}F_c &= \frac{x}{C}\\ \dot{y} &= \frac{p_T}{I_T}\end{aligned}\right\}\ \text{Output equations}$$

Start of Creating Block Diagram of System



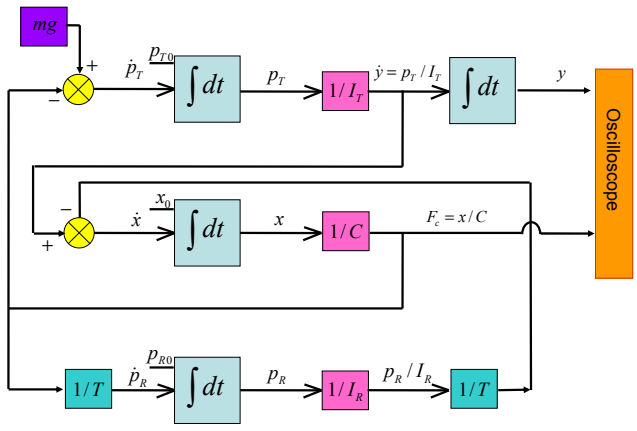
Start of Creating Matrix Form of System Equation

$$\mathbf{x} = \begin{Bmatrix} p_R \\ x \\ p_T \end{Bmatrix} \qquad \mathbf{u} = \{g\} \qquad \mathbf{y} = \begin{Bmatrix} F_c \\ y \end{Bmatrix}$$

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \qquad \mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\mathbf{A} = ? \qquad \mathbf{B} = ? \qquad \mathbf{C} = ? \qquad \mathbf{D} = ?$$

Block Diagram of System



Start of Creating Matrix Form of System

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\left.\begin{aligned}\dot{\mathbf{x}} = \begin{Bmatrix} \dot{p}_R \\ \dot{x} \\ \dot{p}_T \end{Bmatrix} &= \begin{bmatrix} 0 & \frac{1}{TC} & 0 \\ -\frac{1}{TI_R} & 0 & \frac{1}{I_T} \\ 0 & -\frac{1}{C} & 0 \end{bmatrix} \begin{Bmatrix} p_R \\ x \\ p_T \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \\ I_T \end{bmatrix} [g]\end{aligned}\right\}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}$$

$$\left.\begin{aligned}\mathbf{y} = \begin{Bmatrix} F_c \\ \dot{y} \end{Bmatrix} &= \begin{bmatrix} 0 & \frac{1}{C} & 0 \\ 0 & 0 & \frac{1}{I_T} \end{bmatrix} \begin{Bmatrix} p_R \\ x \\ p_T \end{Bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} [g]\end{aligned}\right\}$$

DIMENSIONALITY FOR SIMULINK

Must do dimensional unit checks---I prefer to use English as it forces you to get the units correct. In SI it is possible to think you have units correct and still make unit errors. As a start one has

m=2	initial cylinder mass	lbm
IT=m		lbm
g=386.4	gravity constant acceleration	in/s^2
gc=386.4	unit convertor constant	lbm in / (lbf s^2)
init_cylinder_trans_velocity=0		in/s
pTinit=init_cylinder_trans_velocity*m/gc	initial translational momentum	lbf s
r=2	cylinder radius	in
T=1/r	transformer modulous	1/in
yinit=10.	initial ver loc of cylinder cm	in
xinit=0.	nitial cable stretch	in
C=1	(initial) cable compliance	in/lbf
IR=m*r^2 /2	(initial) cylinder rotary inertia	lbm in^2
init_cylinder_rot_velocity=0		rad/s
pRinit=IR*init_cylinder_rot_velocity/gc		lbf in s

Examine purple block: mg hag units lbm * in/s^2 and is associated with a force equation. It needs to be converted to mg/gc to be made a lbf.

The mg./gc block output is added to a term at the yellow sum block. That term must be in lbf. Check to so if it is.

The result of the summation is integrate with time resulting in a lbf s. The initial condition to that integral block must also be in lbf. The initial condition shown is p_{T0} or pTinit=init_cylinder_trans_velocity*m/gc. Notice how the initial velocity times mass must be scaled by 1/gc.

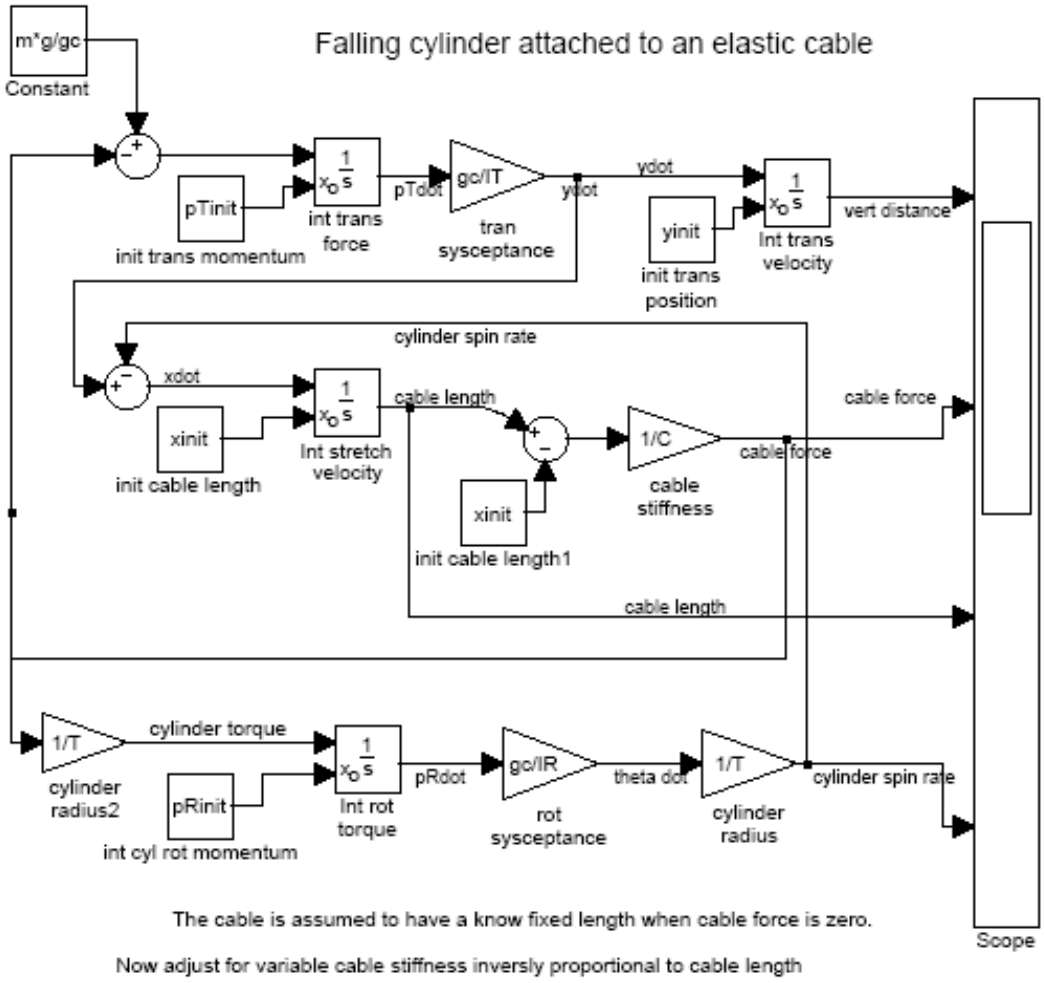
Continue to check for dimensionality of each block. Note that p_{R0} or pRinit=IR*init_cylinder_rot_velocity/gc also is scaled by gc.

DIMENSIONALITY FOR MATLAB

First examine the equation $\dot{p}_R = \frac{x}{TC}$. The dimensionality of the right hand side is lbf. Thus the dimensionality of linear momentum must be lbf s and rotational momentum lbf in^2 s.. Next examine the equation $\dot{x} = \frac{p_T}{I_T} - \frac{p_R}{TI_R}$. As the dimensionality of the right hand side is in/s the required dimensionality of I_T should be lbf s^2/in. If I_T has dimensionality of lbm then I_T / g_c has the dimensionality needed. From this and the previous discussion it is obvious that all calculations utilizing mass require the mass in lbm be scaled by 1/gc. Similarly the weight force requires the mass (in lbm) to be scaled using 1/gc.

SIMULINK CODING

Open Matlab and then Simulink. Make Matlab windows "all tabbed". Attempt, using "Library Explorer" to create the following model.

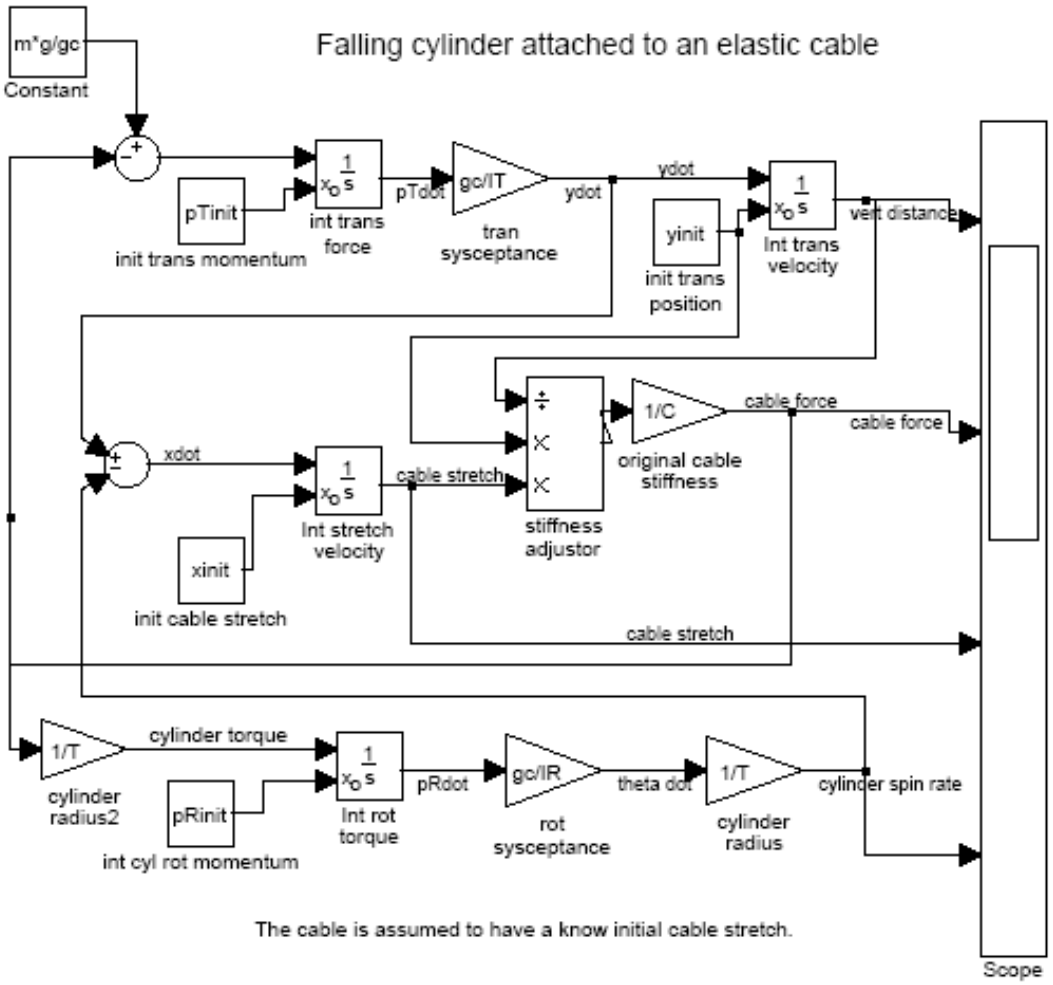


Then use "Model Explorer to input the variables"

m=10 % initial cylinder mass -- lbm
IT=m %
g=386.4 % --- in per s^2
gc=386.4 % --- lbm in / (lbf s^2)
init_cylinder_trans_velocity=0 % --- in/s
pTinit=init_cylinder_trans_velocity*m/gc % initial translational momentum --- lbf s
r=5 % cylinder radius--- in
T=1/r % transformer modulous --- 1/in
yinit=10. % initial vertical location of cylinder center of mass ---- in
xinit=10. % initial cabke length ----- in
C=1/10. % cable compliance ---- in/lbf
IR=m*r^2 /2% initial cylinder rotary inertia ----- lbm in^2
init_cylinder_rot_velocity=0 % ---rad/s
pRinit=IR*init_cylinder_rot_velocity/gc %----lbf in s

This can be done directly into the Model Explorer with explicit code as given above, through the use of variables placed in the mdl, reading a data or mat file or executing an m file. I like thelast choice and created a file called cyl_and_cable_dat.m containing these statements.

Examine the results of the above code. One can consider it as a case of linearizing the stiffness of the cable at the stiffness of the original cable length. As the cable gets longer its actual stiffness should decrease. Linearly with the inverse of the cable length. Modifying the above results for this gives:



Cable has length dependent variable stiffness

MATLAB CODING

Coding for Simulation using ODE

Although not strictly necessary I utilize a main m file and a function m file. The function m file is of the same name as the function within it. Alternatively one could put the function inside of the main m file as a local function or one may not wish to use a main m file. I use a main m file so that I do not have to retype commands and to minimize the reproduction of typing errors.

For this particular problem the system is linear and thus it was possible to create system matrics. Thus I use these matrices in the function m file. I use a global statement in both the main m file and function m file to transport calculated numbers so that they do not need to be recalculated over and over again.

I also used a file called cyl_and_cable_dat.m to place all of my input parameter data. This way I could use the same data in all of my programs to make sure I got the same results.

My function m file, recall it has the same name, looks like

```
function f=falling_cylinder(t,x)
global Amatrix Bmatrix Cmatrix Dmatrix u
% note x(1)=rotational momentum of cylinder
%        x(2)=the stretch of cable
%       x(3)=translational momentum of cylinder

f=Amatrix*x+Bmatrix*u;
```

It is tryuly very simple. Notice that the forcing function u was passed using the global statement. Alternatively it could have been calculated repeatedly within this function.

My main m file had a few portions to it. The first portion looks like:

```
close all
clear all

% Mechanical Properties
cyl_and_cable_dat

global Amatrix Bmatrix Cmatrix Dmatrix u
% easier to transport values in the case of linear system through matrices
Amatrix=[0,1/T/C,0;-1/T/(IR/gc),0,1/(IT/gc);0,-1/C,0]
Bmatrix=[0;0;IT/gc]
u=g

% time range used
Ttotal=10            % total time

% print initial conditions
pR0=pRinit
x0=xinit
pT0=pTinit

[t,xcolumn_as_transposed_to_columned_time_sequences]=ode23('falling_cylinder',[0
Ttotal],[pR0 x0 pT0]);
```

Its purpose is to setup the data for the differential equations. The remaining portion of the code is used to present the results. It looks like:

```
Cmatrix=[0,1/C,0;0,0,1/(IT/gc)] %note this C gets ydot not y
Dmatrix=[0;0]

% as system is linear it is easier to use matrices
% matlab places the solutions for each state variable as a column
% to make it easier to see math transposes of the ode result

ncolneeded=size(xcolumn_as_transposed_to_columned_time_sequences',2);
ycolumn_as_time_row_sequences=Cmatrix*xcolumn_as_transposed_to_columned_time_sequen
ces'+g*Dmatrix*ones(1,ncolneeded);
figure
hold on
subplot(3,1,1), plot(t,ycolumn_as_time_row_sequences(1,:),'k-')
title('Falling cylinder held by elastic cable')
ylabel('cable force in lbf')
xlabel('t')
subplot(3,1,2), plot(t,ycolumn_as_time_row_sequences(2,:),'k-')
ylabel('falling velocity')
xlabel('t')
```

I did not bother setting up data for the third plot. Note that the ode routine return the state vector (a 3x1 in the analysis above) as an nx3. The n is associated with the times where the results are kept. Thus when finding the output vector y transposes are required.

Finally notice that the state vector included cable stretch and two momentums. The two plots shown were in cable force and cylinder falling velocity, items that are easily obtained from the states. If we want to know the vertical height of the cylinder with time we need to decide which of two procedures we want to use. If we do not wish to rerun the simulation then the state vector results can be used through integration of the point data they provide. This is done by using a trapezoidal integration of the velocity data and modifying the later part of the code above to

```
cyl_velocity=xcolumn_as_transposed_to_columned_time_sequences(:,3)/(IT/gc);
y0=yinit;
cyl_ver_dist=y0+cumtrapz(t,cyl_velocity)
subplot(3,1,3), plot(t,cyl_ver_dist,'k-')
ylabel('cylinder position')
xlabel('t')
```

The second, and probably a little bit more accurate is to modify the state equations so that the cylinder position becomes a state variable. Although this is not really true, in a dynamical sense as there is no additional potential or kinetic energy stored that is a function of this variable, never-the-less one does this by noting that $\dot{y} = p_T / I_T$. The size of the state vector is thus increased by one. A modification of this type to the system equations is left as an exercise for the student.

Using Matlab Linear System Functions