Problem Set 4 — 2.086 Spring 2012

Released: *Friday, 6 April*
Due: *Tuesday, 24 April, at 2:30 PM, by hardcopy at the beginning of class.*

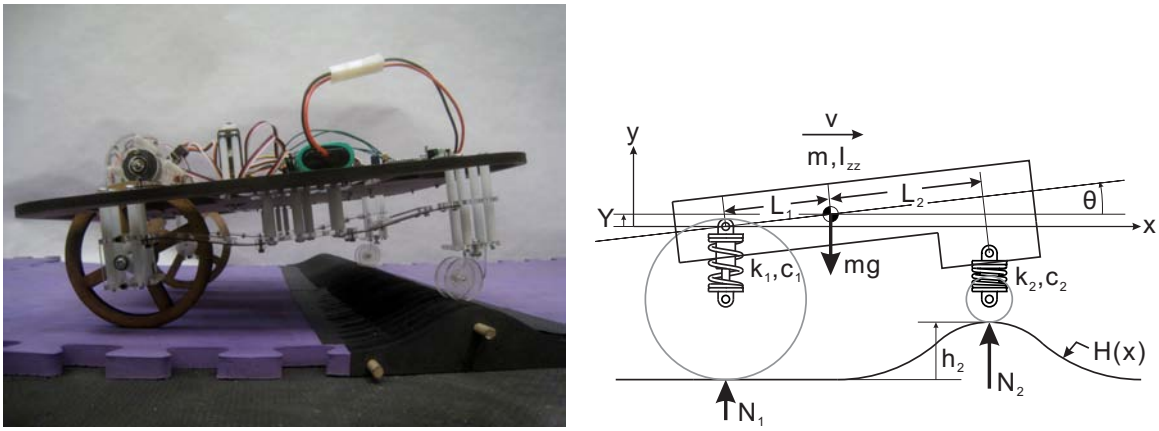
Please also upload to Stellar any MATLAB *function/script files you are asked to supply by hardcopy in your problem set document.*

Recall that this problem set is worth twice as many points as the other problem sets.

Introduction

Although mobile robots operating in flat, indoor environments can often perform quite well without any suspension, in uneven terrain a well-designed suspension can be critical.

An actual robot suspension and its simplified model are shown in Figure 1. The rear and front springs with spring rates k_1 and k_2 serve to decouple the rest of the robot chassis from the (approximated as *massless*) wheels, allowing the chassis and any attached instrumentation to "float" relatively unperturbed while the wheels remain free to follow the terrain and maintain traction. The rear and front dampers with damping coefficients c_1 and c_2 (shown here inside the springs) dissipate energy to prevent excessive chassis displacements (e.g., from excitation of a resonant mode) and oscillations. Note that in our "half-robot" model, k_1 accounts for the *combined* stiffness of both rear wheels, and k_2 accounts for the combined stiffness of both front wheels. Similarly, c_1 and c_2 account for the combined damping coefficients of both rear wheels and both front wheels, respectively.



(a) Actual robot suspension (b) Robot suspension model

Courtesy of James Douglass Penn. Used with permission.

Figure 1: Mobile robot suspension

In this assignment, we are particularly concerned with the possibility of either the front or rear wheels losing contact with the ground, the consequences of which — loss of control and a potentially harsh landing — we wish to avoid.

To aid our understanding of robot suspensions and, in particular, to understand the conditions resulting in loss of contact, we wish to develop a simulation based on the simple model of Figure 1(b).

Specifically, we wish to simulate the transient (time) response of the robot-with-suspension traveling at some constant velocity v over a surface with profile $H(x)$, the height of the ground as a function of x, and to check if loss of contact occurs. To do so, we must integrate the differential equations of motion for the system.

First, we determine the motion at the rear and front wheels in order to calculate the normal forces N_1 and N_2. Because we assume constant velocity v, we can determine the position in x of the center of mass at any time t (we assume $X(t=0)=0$) as

$$X = vt \ . \tag{1}$$

Given the current state $w = [Y, \dot{Y}, \theta, \dot{\theta}]^{\mathrm{T}}$, we can then calculate the positions and velocities in both x and y at the rear and front wheels (assuming θ is small) as

$$\begin{aligned} X_1 &= X - L_1, \quad (\dot{X}_1 = v) \ , \\ X_2 &= X + L_2, \quad (\dot{X}_2 = v) \ , \\ Y_1 &= Y - L_1\theta \ , \\ \dot{Y}_1 &= \dot{Y} - L_1\dot{\theta} \ , \\ Y_2 &= Y + L_2\theta \ , \\ \dot{Y}_2 &= \dot{Y} + L_2\dot{\theta} \ , \end{aligned} \tag{2}$$

where L_1 and L_2 are the distances to the system's center of mass from the rear and front wheels. Here subscript 1 and subscript 2 refer to the rear and front wheels, respectively. Note that we define $Y = 0$ as the height of the robot's center of mass with both wheels in contact with flat ground and both springs at their unstretched and uncompressed lengths, i.e., when $N_1 = N_2 = 0$.

Next, we determine the heights of the ground at the rear and front contact points as

$$\begin{aligned} h_1 &= H(X_1) \ , \\ h_2 &= H(X_2) \ . \end{aligned} \tag{3}$$

Similarly, the rates of change of the ground height at the rear and front are given by

$$\begin{aligned} \frac{dh_1}{dt} = \dot{h}_1 &= v\frac{d}{dx}H(X_1) \ , \\ \frac{dh_2}{dt} = \dot{h}_2 &= v\frac{d}{dx}H(X_2) \ . \end{aligned} \tag{4}$$

Note that we must multiply the spatial derivatives $\dfrac{dH}{dx}$ by $v = \dfrac{dX}{dt}$ to find the desired temporal derivatives.

While the wheels are in contact with the ground, we can determine (assuming massless wheels) the normal forces at the rear and front from the constitutive equations for the springs and dampers as

$$\begin{aligned} N_1 &= k_1(h_1 - Y_1) + c_1(\dot{h}_1 - \dot{Y}_1) \ , \\ N_2 &= k_2(h_2 - Y_2) + c_2(\dot{h}_2 - \dot{Y}_2) \ . \end{aligned} \tag{5}$$

If at any time N_1 or N_2 is calculated from Equations (5) to be less than or equal to zero, we can conclude that the respective wheel has lost contact with the ground. In this problem set we focus on the conditions for "loss of contact" and thus we will terminate the simulation once contact is lost.[1]

Finally, we can determine the rates of change of the state from the linearized ($\cos\theta \approx 1$, $\sin\theta \approx \theta$) equations of motion for the robot-with-suspension, given by Newton-Euler as

$$\begin{aligned} \ddot{Y} &= -g + \frac{N_1 + N_2}{m} \ , \\ \ddot{\theta} &= \frac{N_2 L_2 - N_1 L_1}{I_{zz}}, \end{aligned} \tag{6}$$

where m is the mass of the robot, and I_{zz} is the moment of inertia of the robot about an axis parallel to the Z-axis and passing through the robot's center of mass.

We wish to derive an exact, analytical solution to a simplified version of Equations (6) that we can later use to verify our numerically approximated results. For the special case of $h_1 = \dot{h}_1 = h_2 = \dot{h}_2 = 0$ (i.e., flat ground) and (as in this assignment) $k_1 = k_2 = k$, $c_1 = c_2 = c$, and $L_1 = L_2 = L$, the equations of motion have the solution

$$\begin{aligned} Y(t) &= Y_{\text{eq}} + e^{-\beta_{\text{Y}} t} \left((Y_0 - Y_{\text{eq}}) \cos\omega_{\text{dY}} t + \frac{\beta_{\text{Y}}(Y_0 - Y_{\text{eq}}) + \dot{Y}_0}{\omega_{\text{dY}}} \sin\omega_{\text{dY}} t \right), \\ \theta(t) &= \theta_{\text{eq}} + e^{-\beta_\theta t} \left((\theta_0 - \theta_{\text{eq}}) \cos\omega_{\text{d}\theta} t + \frac{\beta_\theta(\theta_0 - \theta_{\text{eq}}) + \dot{\theta}_0}{\omega_{\text{d}\theta}} \sin\omega_{\text{d}\theta} t \right), \end{aligned} \tag{7}$$

where

$$\begin{aligned} \beta_{\text{Y}} &= \frac{c}{m}, & \omega_{0\text{Y}}^2 &= \frac{2k}{m}, & \omega_{\text{dY}} &= \sqrt{\omega_{0\text{Y}}^2 - \beta_{\text{Y}}^2} \ , \\ \beta_\theta &= \frac{cL^2}{I_{zz}}, & \omega_{0\theta}^2 &= \frac{2kL^2}{I_{zz}}, & \omega_{\text{d}\theta} &= \sqrt{\omega_{0\theta}^2 - \beta_\theta^2} \ , \end{aligned}$$

are damping and natural frequency parameters and

$$\begin{aligned} Y_{\text{eq}} &= \frac{-mg}{2k} \ , \\ \theta_{\text{eq}} &= 0 \ , \end{aligned} \tag{8}$$

is the equilibrium position. This exact solution corresponds to a stationary robot ($v = 0$) far from the bump (e.g., $X = 0$) perturbed from equilibrium with initial condition $[Y_0, \dot{Y}_0, \theta_0, \dot{\theta}_0]$ ($\neq w_{\text{eq}} \equiv [Y_{\text{eq}}, 0, \theta_{\text{eq}}, 0]^{\text{T}}$) and then released.

[1] Note that once contact is lost the respective normal force will continue to be zero until $Y_1 \leq h_1$ (in the case of the rear wheel having lost contact) or $Y_2 \leq h_2$ (in the case of the front wheel having lost contact), at which time the respective equation from Equations (5) can once again be used to determine the respective normal force. This switching on and off of the normal forces for contact and non-contact will lead to a nonlinear differential equation. Although the methods we develop in Unit IV can be applied to this nonlinear differential equation, we do not ask you to consider the "airborne" case in this problem set.

Instructions

Please download the file `Pset_4_datafiles.zip` containing `spring_data.mat`, `Hgauss.m`, `rk4sb.mat`, and `animate_robot.m` from Stellar. The appendix describes how to use the files.

Questions

Part I: Regression

1. (20 pts) We assume that the robot suspension springs follow a force-displacement (F-δ) model of the form

$$F = \beta_0 + \beta_1 \frac{\delta}{\delta_{\max}} + \beta_2 \left(\frac{\delta}{\delta_{\max}} \right)^2, \tag{9}$$

where $\delta_{\max}$ is a maximum displacement. Note that F, and hence β_0, β_1, and β_2, all have units of Newtons, and δ and $\delta_{\max}$ have units of meters.

We next conduct an experiment in which we collect data for the force in a single spring from the suspension for several values of $\delta/\delta_{\max}$, as shown in Figure 2 in the appendix. We then multiply the measured force by two (this scaling has *already* been performed for you in the `spring_data.mat` data) since each spring in our model represents the two (left, right) springs in parallel.

We now ask you to perform a least squares regression on the data in `spring_data.mat` for the model given by Equation (9); see the appendix for a description of the `spring_data.mat` file. Note that $k_1 (= k_2$, since front and rear springs are the same for our robot) is given by $\hat{\beta}_1/\delta_{\max}$, where for our experiments $\delta_{\max} = 0.0165$ meters. You may assume that the model Equation (9) is bias-free.

   (*i*) Provide in a single plot ($\delta/\delta_{\max}$ on the horizontal axis, force on the vertical axis) both the experimental data and your (best-fit) model.

   (*ii*) Provide your least-squares/regression estimate for the spring constant k_1, $\hat{\beta}_1/\delta_{\max}$.

   (*iii*) Provide 95% confidence-level joint confidence intervals for β_0^{true} and β_2^{true}.

   (*iv*) Indicate why, based on the confidence intervals for β_0^{true} and β_2^{true}, you can justify a standard linear spring model for the robot suspension springs.

Part II: Crank-Nicolson Approximation

2. (20 pts) Convert the equations of motion Equations (6) to linear state space form ($\dot{w} = Aw + F$ from Equation (19.25) of the text) for state $w = [Y, \dot{Y}, \theta, \dot{\theta}]^{\mathrm{T}}$ by using Equations (1)–(5) to rewrite Equations (6) in terms of the elements of w; note F should not depend on w.

   (*i*) Identify your 4×4 A matrix (in terms of the symbols provided: g, m, I_{zz}, L_1, L_2, c_1, c_2, k_1, k_2, $H(X)$).

   (*ii*) Identify your 4×1 F vector (in terms of the symbols provided: g, m, I_{zz}, L_1, L_2, c_1, c_2, k_1, k_2, $H(X)$).

Note you should not assume here that $k_1 = k_2$, $c_1 = c_2$, $L_1 = L_2$.

3. (20 pts) Now...

   (*i*) Evaluate and provide the matrix A in your state space formulation of Question 2 in MATLAB for the mobile robot parameters given in Table 1 of the appendix and the value of $k_1 = k_2 (= \hat{\beta}_1/\delta_{\max})$ found in Question 1. Note your result should be in terms of numerical values for the 16 entries of `A`. (Please make sure to clearly indicate which values are associated with which indices of `A`.)

   (*ii*) Find the eigenvalues λ_k, $k = 1, 2, 3, 4$, of A by using the MATLAB built-in function `eig()`. Note your answer should be four complex floating point numbers.

4. (20 pts)

   (*i*) Provide a plot on the Crank-Nicolson absolute stability diagram (Figure 19.12(a) of the text) with $\lambda_k \Delta t$, $k = 1, 2, 3, 4$, indicated for each of the three cases $\Delta t_1 = 0.2$ s, $\Delta t_2 = 0.1$ s, and $\Delta t_3 = 0.001$ s. Note you should provide a *single* plot with all three cases (each case corresponding to a different Δt) on the one plot but with a different symbol for (the four $\lambda_k \Delta t$ points associated with) each case.

   (*ii*) Which Δt is/are absolutely stable for Crank-Nicolson? Clearly indicate for each Δt either stable or unstable.

5. (20 pts) Based on the A and F matrices found in Question 2, implement a Crank-Nicolson integrator script and run it for initial state $w_0 = [0, 0, 0.1, 0]^{\mathrm{T}}$, velocity $v = 0$ (and hence $X(t) = 0$ for all t), final time $t_{\text{final}} = 2$ s, and time steps $\Delta t = 0.001$ s (first run) and $\Delta t = 0.0005$ s (second run); note "s" refers to seconds. In order to fully test your script you should use the `Hgauss()` function provided (described in the appendix) as your $H(x)$ in Equations (3) and (4) (even though `Hgauss(0)` will return an extremely small number, effectively zero).

   (*i*) Please copy-paste your MATLAB Crank-Nicolson script into your problem set (and also upload to stellar).

   (*ii*) Provide a 2×3 table: tabulate (in the three columns) $\theta(t_{\text{final}})$, $\tilde{\theta}(t_{\text{final}})$, and $|\theta(t_{\text{final}}) - \tilde{\theta}(t_{\text{final}})|$ for (in the two rows) $\Delta t = 0.001$ s and $\Delta t = 0.0005$ s. Make sure to clearly label the rows and columns of the table. Note that θ is the exact analytical solution from Equations (7) and $\tilde{\theta}$ refers to your Crank-Nicolson approximation.

   (*iii*) Does your numerical approximation change appreciably when you halve Δt? Does the error in your numerical approximation decrease by the factor you expect when you halve Δt?

6. (20 pts) Run your Crank-Nicolson code again, but now for initial state $w_0 = w_{\text{eq}} \equiv [Y_{\text{eq}}, 0, \theta_{\text{eq}}, 0]^{\mathrm{T}}$ from Equations (8) and non-zero v. You may choose $\Delta t = 0.001$ s except in part (*iii*).

   Test four different cases: $v = 0.25$, $v = 1.2$, $v = 5$, and $v = 10$.

   (*i*) Indicate, for $t_{\text{final}} = 1.25/v$ (you may use `ceil` of $1.25/v$), which velocities result in a loss of contact. Note you may continue your simulations beyond the loss of contact point (or you may terminate the simulation upon loss of contact), however the calculations are not physically relevant once contact is lost — the ground will not generate a negative normal force!

(*ii*) Plot N_1 and N_2 as a function of time for each of the four cases. Note that once you obtain your numerical approximation for Y and θ you can then apply Equations (5) to obtain the normal forces. Note you should provide one plot for each of the four cases (each plot will contain two curves, one for N_1 and one for N_2); please provide axis labels, legends, and figure titles.

(*iii*) Now rerun your simulations (for all four velocity cases) for $\Delta t = 0.0005$ s. Indicate which velocities result in a loss of contact for this smaller Δt. When you change Δt from 0.001 s to 0.0005 s do your conclusions change as to which velocities result in loss of contact?

Part III: Runge-Kutta Approximation

Preamble. In Part III we will implement a generic, four-stage Runge-Kutta integrator function `RK4` which can work with a variety of functions $g(w, t)$ through the appropriate MATLAB function handle.

In particular, in Part III we will write a function

```
function [wdot, N1, N2] = gfunc_linear(w, t, v)
```

that computes the rate of change of state $\dot{w}$ (MATLAB `wdot`) (a 4×1 column vector calculated as $Aw + F$), and normal forces N_1 and N_2 (MATLAB `N1`, `N2`) , from the current state w (MATLAB `w`) (a 4×1 column vector), current time t (MATLAB `t`), and velocity v (MATLAB `v`). Note that, although you will not need N_1 and N_2 for your Runge Kutta integrator, you might find these outputs useful for computing the normal forces after running the simulation. You should use the particular function `Hgauss()` provided (see the appendix) as your $H(x)$ in Equations (3) and (4).[2]

The `RK4` integrator function will then be defined as

```
function [W, T] = RK4(gfunc_handle, w0, deltat, tfinal, v)
```

where `gfunc_handle` is the handle (i.e., `@gfunc_linear`) to a function that, given a state w, time t, and velocity v, returns the corresponding rate of change of state $\dot{w}$; `w0` is the initial state (a 4×1 column vector); `deltat` is the desired integration time step Δt; `tfinal` is the desired final time for integration; and `v` is the robot velocity.[3] On output, `T` is a $1 \times (J + 1)$ (where $J \approx t_{\text{final}}/\Delta t$) row vector of the times $t^j = j\,\Delta t$, $0 \le j \le J$, associated with the integration, and `W` is a $4 \times (J + 1)$ array of the (RK4 approximate) states $\tilde{w}^j$, $0 \le j \le J$. □

7. (20 pts)

(*i*) Provide a plot on the four-stage Runge-Kutta absolute stability diagram (Figure 19.14(b) of the text) with $\lambda_k \Delta t$, $k = 1, 2, 3, 4$, indicated for each of the three cases $\Delta t_1 = 0.2$ s, $\Delta t_2 = 0.1$ s, and $\Delta t_3 = 0.001$ s. Note you should provide a *single* plot with all three cases (each case corresponding to a different Δt) on the one plot but with a different symbol for (the four $\lambda_k \Delta t$ points associated with) each case.

(*ii*) Which Δt is/are absolutely stable for the four-stage Runge-Kutta? Clearly indicate for each Δt either stable or unstable.

[2] It would also be possible to pass the handle of the MATLAB function associated with (any) chosen $H(x)$; we do not ask you to provide this generality.

[3] Note it is possible to eliminate the `v` argument from `RK4` — such that all the system information is included in `gfunc_handle` and hence `RK4` is a generic Runga-Kutta integrator — by appeal to anonymous functions. However, for our purposes here you can follow the simpler and more explicit approach suggested.

The file `rk4sb.mat` (see the appendix) contains the absolute stability "neutral boundary" of Figure 19.14(b); you need only add to the plot the $\lambda_k \Delta t$, $k = 1, 2, 3, 4$, for the three cases requested.

8. (20 pts) Based on your (linear) state space equations from Question 2, write a function `gfunc_linear`, with the inputs and outputs as defined above in the Preamble.

   (*i*) Please copy-paste your MATLAB function `gfunc_linear` into your problem set (and also upload to stellar).

   (*ii*) What do you expect your function `gfunc_linear` to return for output `wdot` for the case in which input `w0` is the equilibrium position $w_{\text{eq}} \equiv [Y_{\text{eq}}, 0, \theta_{\text{eq}}, 0]^{\text{T}}$ and `t` $= 0$, `v` $= 0$? What does your function `gfunc_linear` in fact return when you run your code for this equilibrium input argument?

9. (30 pts) Write a four-stage Runge-Kutta integrator function, with inputs and outputs as defined in the Preamble above and as described in Example (19.1.14) of the text. Run your `RK4` code with your `gfunc_linear` function, initial state $w_0 = [0, 0, 0.1, 0]^{\text{T}}$, velocity $v = 0$, final time $t_{\text{final}} = 2$ s, and $\Delta t = 0.2$ s (first run) and $\Delta t = 0.1$ s (second run) and $\Delta t = 0.001$ s (third run), using the syntax

   ```
   [W, T] = RK4(@gfunc_linear, w0, deltat, tfinal, v)
   ```

   (*i*) Please copy-paste your MATLAB function `RK4` into your problem set (and also upload to stellar).

   (*ii*) Provide a 3×3 table: tabulate (in the three columns) $\theta(t_{\text{final}}), \tilde{\theta}(t_{\text{final}})$, and $|\theta(t_{\text{final}}) - \tilde{\theta}(t_{\text{final}})|$ for (in the three rows) $\Delta t = 0.2$ s, $\Delta t = 0.1$ s, and $\Delta t = 0.001$ s. Make sure to clearly label the rows and columns of the table. Note that θ is the exact analytical solution from Equations (7) and $\tilde{\theta}$ refers to your Runge-Kutta approximation.

   (*iii*) Do your numerical results agree with your predictions of Question 7 as to which Δt should be stable for RK4?

   (*iv*) Which scheme is more accurate for $\Delta t = 0.001$ s — Crank-Nicolson or RK4?

   Note here you compare the accuracy of Crank-Nicolson and RK4 for the same value of the time step. We can also consider the efficiency — the cost to achieve the same accuracy. Often RK4 will win this competition: RK4 provides higher order convergence than Crank-Nicolson but without the need to solve a linear system (since RK4 is explicit, while Crank-Nicolson is implicit). However, if stability is an issue, Crank-Nicolson could prove the better choice.

10. (10 pts) Run your `RK4` code with your `gfunc_linear` function once again, but now for initial state $w_0 = w_{\text{eq}} = [Y_{\text{eq}}, 0, \theta_{\text{eq}}, 0]^{\text{T}}$ from Equations (8). You may choose $\Delta t = 0.001$ s.

    Test four different cases: $v = 0.25$, $v = 1.2$, $v = 5$, and $v = 10$.

    (*i*) Indicate, for $t_{\text{final}} = 1.25/v$ (you may use `ceil` of $1.25/v$), which velocities result in a loss of contact. Note you may continue your simulations beyond the loss of contact point (or you may terminate the simulation upon loss of contact), however the calculations are not physically relevant once contact is lost — the ground will not generate a negative normal force!

(*ii*) Plot N_1, and N_2 as a function of time for each of the four cases. Note you should provide one plot for each of the four cases (each plot will contain two curves, one for N_1 and one for N_2); please provide axis labels, legends, and figure titles.

Appendix

1. The parameters for the mobile robot — except for k_1 and k_2, which will be found in Part I — are listed in Table 1.

g	9.81	m/s^2
m	1.456	kg
I_{zz}	0.012	kg-m^2
L_1	0.107	m
L_2	0.107	m
c_1	1.5	N-s/m
c_2	1.5	N-s/m

Table 1: Mobile robot parameters.

2. The file `spring_data.mat` contains the data from one load vs. displacement test (see Figure 2) of a plastic, parallel beam flexure spring. Although the flexure spring shown in Figure 2 differs in appearance from the coil spring shown in Figure 1(b), the two springs can be considered to be functionally equivalent. The loads in Newtons are stored in `force`; the nondimensional displacements $\delta/\delta_{\max}$ are stored in `delta_over_delta_max`. Note that for our experiments $\delta_{\max} = 0.0165$ meters.

3. The function `Hgauss` takes one argument `x` corresponding to an x-coordinate or series of x-coordinates and returns the height of the ground $H(x)$ at x and its spatial derivative $H'(x)$ in `H` and `dHdx`. All distances are assumed to be in meters. A plot of $H(x)$ is shown in Figure 3.

```
function [H, dHdx] = Hgauss(x)
A = 0.0254;
sig = 0.0288;
x0 = 0.3;
H = A*exp(-(x-x0).^2/(2*sig^2));
dHdx = -(x-x0)./sig^2.*H;
```

4. The file `rk4sb.mat` contains the real and imaginary coordinates of the "neutral boundary" of the RK4 absolute stability diagram: you may plot as `plot(rk4sb(:,1), rk4sb(:,2))`.

5. The function `animate_robot` animates the robot's states over time and takes four arguments `W`, corresponding to the $4 \times J$ array of states returned by `RK4`, `T`, corresponding to the $1 \times J$ array of times returned by `RK4`, `v`, the velocity of the robot, and `steps_per_frame`, an integer greater than or equal to 1 that can be used to control the number of integration time steps to increment for each successive animation frame. For example, the following call to `animate_robot` would animate the simulation results stored in `W` at a speed of 2 integration steps per frame:

   ```
   animate_robot(W,T,v,2)
   ```

Figure 2: Spring constant test setup showing spring and digital scale.

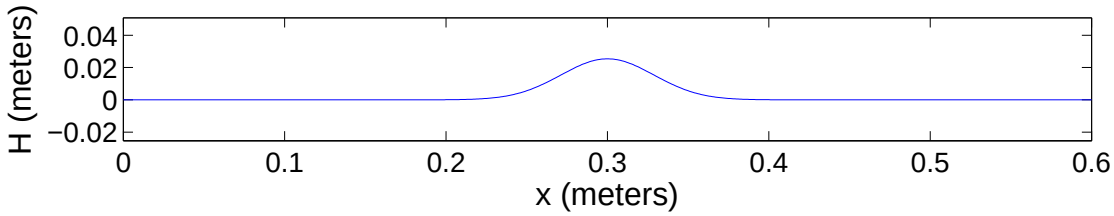


Figure 3: $H(x)$ as defined by `Hgauss.m`.

Note that to slow down the animation below the minimum `speed` of 1, you would need to use a smaller time step.

Figure 4 shows a sample frame from an animation in which the front caster of the robot has just reached the peak of the bump. The `animate_robot` function extracts the Y and θ information from the input W, computes $X = vt$ for each time in T, and draws the robot body as a line with the appropriate center and slope. The wheels are then drawn either in contact with the ground if $Y_1 \leq h_1$ for the rear wheel or $Y_2 \leq h_2$ for the front wheel. Otherwise, in accordance with our assumption of massless wheels, they simply follow the rear

or front of the robot, which we can assume has lost contact with the ground.

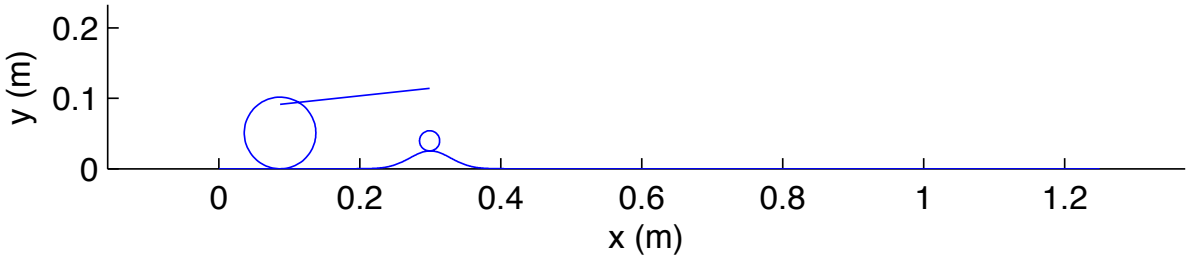


Figure 4: Sample frame from `animate_robot` animation.

Note you are not obligated to use this `animate_robot` function. You might find it useful in confirming that your results look reasonable. Or you may just find it a gratifying graphical fashion by which to visualize your solutions.

2.086 Numerical Computation for Mechanical Engineers
Fall 2012