1 Introduction

Differential equations, especially describing motion are fascinating because while difficult to solve analytically, they just look so realistic when put in a movie or a video game. A lot of research applications require the ability to solve differential equations. You find the forces and MATLAB does the rest!

In this assignment we'll simulate a ball moving along a curved rail. You're given the task of **finding out what initial speed you need to give to the ball to be able to get it from the red point to the yellow point**.

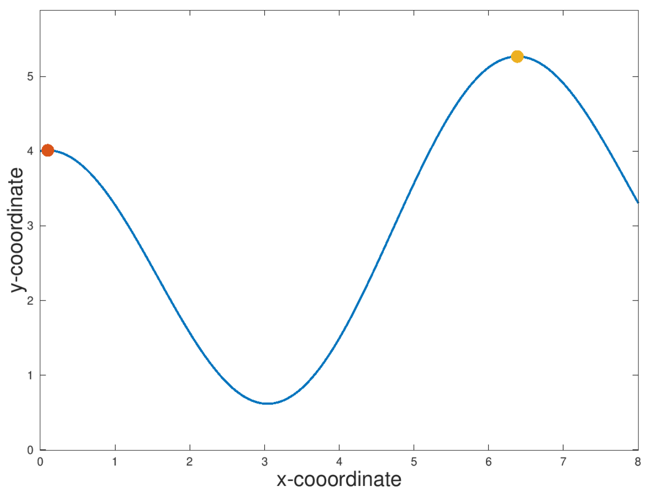

Figure 1: A side view of you track on which the ball is moving. You have to get it from the red point to the yellow point with zero final velocity by choosing an initial velocity.

Your colleague took a laser distance measure and mapped how the rail bends along the track. She's given you a data file called `track.txt`. Unfortunately, the measurements were really noisy. You'll need to approximate the function from the data. To achieve the final goal of getting the ball to stay still at the yellow point by pushing it at the red point, you will:

- read in the noisy data your colleague gave you
- approximate the rail function from the data
- write a function predicting the height of the rail at all x
- get the slope of your rail height function (for gravity component)
- implement the ordinary differential equation of the ball, $f(t, x)$
- simulate the ball movement
- Optional: make a function that computes final velocity at the yellow point
- Optional: solve for the initial velocity of the ball that gets it to the yellow point with zero final velocity
- Optional: make a movie of the ball movement

Format Requirements

- 1-2 page report, includes:
  - work done
  - results
  - discussion of results
  - 500 words of text max (excluding figure captions)
- 1 file of commented, executing code

Implementation Requirements

- function `[x, y] = read_data()` which reads the delimited file `track.txt` and outputs points x and y
- function `th = fit(x, y)` which approximates function approximation coefficients from your data
- function `yp = predict(x, th)` which predicts the rail height at x from best fit coefficients
- function `dyp = slope(x, th)` which gives the slope of the approximate rail function from the best fit coefficients
- function `du = f(t, u, th)` which describes the ordinary differential equation of the ball
- a script that simulates the ball's movement
- optional: function `uf = closest_approach(v0, th)` which sets the ball off with an initial velocity and returns how close and with what final velocity the ball got to the yellow point
- optional: a script that uses `closest_approach` and `fsolve` to solve for the initial velocity getting the ball to the yellow point with zero final velocity
- optional: a movie of the ball motion

General Hints

- Package your work into a script with each of the factions in separate files. Only merge all of your code into a single file for submission.
- Plotting helps a lot. Whatever you get, plot it.
- The website contains many examples of things you need to complete in this assignments. Look at notebooks: `http://stanford.edu/class/cme192/notebooks.html`
- Check out a movie of a similar situation `http://stanford.edu/class/cme192/data/movies/ball_in_a_bowl.mp4`

Reading the data in

Implementation

Write a function that returns reads in the noisy data file `track.txt` (available on the website) and returns vectors of coordinates x and y.

Name:

- `[x, y] = read_data()`

Inputs:

None

Outputs:

- `x` the x coordinates of the rail
- `y` the y coordinates of the rail

Approximating a function

We know that our data, despite being noisy, are represented by the a linear combinations of functions

- 1 (a constant shift, up or down)
- x
- $\cos(x)$

Thus, our rail function looks like this

$$h(x) = \theta_0 1 + \theta_1 x + \theta_2 \cos(x)$$

where $h(x)$ gives the height of the rail at each x.

We can fit the best fit line by solving the equation

$$\begin{bmatrix} 1 & x_1 & \cos(x_1) \\ 1 & x_2 & \cos(x_2) \\ 1 & x_3 & \cos(x_3) \\ \vdots & \vdots & \vdots \\ 1 & x_n & \cos(x_n) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix}$$

Implementation

Write a function that takes x and y data and computes the best fit coefficients to them in the form $h(x) = \theta_0 1 + \theta_1 x + \theta_2 \cos(x)$. You need to find the vector of θ.

$$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$

Name:

- `th = fit(x, y)`

Inputs:

- `x` the x coordinates of the rail
- `y` the y coordinates of the rail

Outputs:

- `th` the θ vector of function approximation

Predicting function value

Our approximate function looks like this.

$$h(x) = \theta_0 1 + \theta_1 x + \theta_2 \cos(x)$$

So, for a given x

We can get the predicted value of a slope by

$$h(x) = \begin{bmatrix} 0 & 1 & -\sin(x_1) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} h(x_1) \end{bmatrix}$$

or for multiple x

$$h(x) = \begin{bmatrix} 1 & x_1 & \cos(x_1) \\ 1 & x_2 & \cos(x_2) \\ 1 & x_3 & \cos(x_3) \\ \vdots & \vdots & \vdots \\ 1 & x_n & \cos(x_n) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} h(x_1) \\ h(x_2) \\ h(x_3) \\ \vdots \\ h(x_n) \end{bmatrix}$$

Implementation

Write a function that takes x and best fit coefficients and computes the function value $h(x) = \theta_0 1 + \theta_1 x + \theta_2 \cos(x)$.

You need to return a predicted y or a vector of predicted y's if the input is a vector of x.

Name:

- `yp = predict(x, th)`

Inputs:

- `x` the x coordinates of the rail (a scaler or a vector)
- `th` the best fit coefficients (a vector of 3 elements)

Outputs:

- `yp` the predicted value of the rail height at that location (scalar or vector)

Slope of the approximate function

Our approximate function looks like this.

$$h(x) = \theta_0 1 + \theta_1 x + \theta_2 \cos(x)$$

If we want to get its slope, it's simply a matter of taking a derivative.

$$h'(x) = \theta_0 0 + \theta_1 1 - \theta_2 \sin(x)$$

So, for a given x

We can get the predicted value of a slope by

$$h'(x) = \begin{bmatrix} 0 & 1 & -\sin(x_1) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} h'(x_1) \end{bmatrix}$$

or for multiple x

$$h'(x) = \begin{bmatrix} 0 & 1 & -\sin(x_1) \\ 0 & 1 & -\sin(x_2) \\ 0 & 1 & -\sin(x_3) \\ \vdots & \vdots & \vdots \\ 0 & 1 & -\sin(x_n) \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix} = \begin{bmatrix} h'(x_1) \\ h'(x_2) \\ h'(x_3) \\ \vdots \\ h'(x_n) \end{bmatrix}$$

Implementation

Write a function that takes x and best fit coefficients and computes the function slope $h'(x) = \theta_0 0 + \theta_1 1 - \theta_2 \sin(x)$.

You need to return a predicted slope at that x or a vector of slopes if x is a vector.

Name:

- `dyp = slope(x, th)`

Inputs:

- `x` the x coordinates of the rail (a scaler or a vector)
- `th` the best fit coefficients (a vector of 3 elements)

Outputs:

- `dyp` the predicted slope of the rail at that location (scalar or vector)

2 Implementing the Ordinary Differential Equation

Our differential equation for any dynamic object looks like this

$$\ddot{x} = \frac{1}{m} F(t, x)$$

where F is the force acting on the object and m is the mass. We're going to assume that $m = 1$, for simplicity.

In the case of a ball on the rail, taking x_1 to be the ball's horizontal position along the track, the ODE looks like this.

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \cos(\alpha) \\ -g \sin(\alpha) \end{bmatrix}$$

where $\alpha = \tan^{-1}(h'(x))$ is the angle of the slope at that point x. $g = 9.81$ is the acceleration due to gravity.

x_1 corresponds to the horizontal position of the ball along the track. It's vertical position can be obtained from $h(x)$.

x_2 corresponds to the speed of the ball along the track.

Implementation

Implement the ODE function $f(t, u)$ as given above.

Name:

- `du = f(t, u, th)`

Inputs:

- `t` time t (we won't use it, but MATLAB requires it)
- `u` vector of states x_1 and x_2
- `th` vector of best fit coefficients

Outputs:

- `du` vector of state derivatives $\dot{x}_1$ and $\dot{x}_2$

Hint: your function should call `slope` *to be able to compute* α*.*

Plotting the ball's movement

Pick any initial conditions.

Implement a script that takes the initial position of the ball to be $x_1 = 0.1$ and initial velocity $x_2 = 0$ and computes the ball movement in time. Plot the states x_1, x_2 versus time.

Hint: MATLAB's `ode45` *requires a derivative function to take only two arguments t and u while our derivative function takes 3, t, u and th. However, you can always do*

```
th = % previously computed value
f2 = @(t, u) f(t, u, th)
```

to get around that.

This is the end of mandatory problems

Optional: Getting the final height and speed from initial speed

If we are to get the exact initial velocity, starting at $x_1 = 0.1$ to get to the yellow point with zero final velocity, we need to know how to evaluate how much are we off first.

The yellow point is around $x = 6$. It's the point where the function reaches a local maximum, so it's derivative (slope) must be equal to zero.

Doing that is pretty easy.

- find where the yellow point is (*Hint: use* `fzero` *with a good initial condition*)
- choose an initial velocity
- solve the ball's movement for about 5 seconds in time with 10^3 points
- find the position and index of the solution where the ball is closest to the yellow point
- use the index to get the ball's velocity at that point
- return a vector of distance of closest approach (with a sign) to the red point and the velocity at that moment

Once we have that function, we can just plug it into `fsolve` and solve for the initial velocity getting us to the yellow point with zero final velocity!

Implementation

Always start at $x_1 = 0.1$.

Name:

- `uf = closest_approach(v0, th)`

Inputs:

- `v0` initial velocity
- `th` vector of best fit coefficients

Outputs:

- `uf` a vector of 2 scalars, (1) `dx_cl` distance between closest point in the solution and the yellow point (will be small, make sure it can be both positive and negative), (2) `v_cl` final velocity at the point of closest approach to the red point

Hint: This function builds upon all the previous work.

Optional: Solving for the just-right initial velocity

Implementation

Use `fsolve` to solve for the initial velocity placing the ball at the red point with almost zero final velocity.

Hint: You can package your function `closest_approach` *into a form* `fsolve` *accepts by doing:*

```
th = % computed earlier

to_solve = @(v) closest_approach(v, th)
```

Optional: Make a movie of the ball

Take your just-right initial velocity, solve for the motion at a reasonable number of steps to make a movie of your ball getting to the yellow point with zero final velocity.