Objectives

The tutorial will introduce you to n-dimensional simulations. You'll modify the `evolve()` function you wrote in lab 6 to account for motion in x,y and z directions. You'll then simulate David Beckham's famous goal in 2001 against Greece that secured England a trip to the World Cup.

http://www.youtube.com/watch?v=L2wpH09cOU4

3-Dimensional Differential Equation solver

Background

For motion in three dimensions, the position vector $\vec{\boldsymbol{r}}(t)$ is a function of time and has three components $x(t)$, $y(t)$ and $z(t)$. In mathematical language, this is usually written either using unit vector notation or in coordinate notation..

$$\vec{\boldsymbol{r}}(t) = x(t)\hat{\boldsymbol{\imath}} + y(t)\hat{\boldsymbol{\jmath}} + z(t)\hat{\boldsymbol{k}} = \langle x(t), y(t), z(t)\rangle$$

We are going to use two dimensional arrays in MATLAB to represent time dependent vector functions such as $\vec{\boldsymbol{r}}(t)$, $\vec{\boldsymbol{v}}(t)$, and $\vec{\boldsymbol{a}}(t)$. We will choose the following conventions:

- The (first) row index `I = 1`, `2`, and `3` will correspond to x, y, and z, respectively.
- The (second) column index `J` will correspond to the time t_J.

With these conventions, the three elements in the Jth column will correspond to position vector $\vec{\boldsymbol{r}}(t_J)$. If the time array has size $1 \times N$, then the array `r` that corresponds to $\vec{\boldsymbol{r}}(t)$ should have size $3 \times N$.

$$\begin{bmatrix} x(t_1) & x(t_1) & x(t_3) & \cdots \\ y(t_1) & y(t_2) & y(t_3) & \cdots \\ z(t_1) & z(t_2) & z(t_3) & \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} \texttt{r(1,1)} & \texttt{r(1,2)} & \texttt{r(1,3)} & \cdots \\ \texttt{r(2,1)} & \texttt{r(2,2)} & \texttt{r(2,3)} & \cdots \\ \texttt{r(3,1)} & \texttt{r(3,2)} & \texttt{r(3,3)} & \cdots \end{bmatrix} \tag{1}$$

We will need three such arrays – one for each of $\vec{\boldsymbol{r}}(t)$, $\vec{\boldsymbol{v}}(t)$, and $\vec{\boldsymbol{a}}(t)$.

Building `evolve3D()`

Write a MATLAB function called `evolve3D()` that is based on the midpoint algorithm for numerically solving ordinary differential equations. It should accept three arguments.

- An array `t` with size $1 \times N$ that represents t.
- The initial position `r0` with size 3×1 (column array) that represents $\vec{\boldsymbol{r}}_0$.
- The initial position `v0` with size 3×1 (column array) that represents $\vec{\boldsymbol{v}}_0$.
- The name of the anonymous function that computes the acceleration. This function must have three arguments, that is it must have the form $\vec{\boldsymbol{f}}_a(t, r, v)$.

It should return three arrays `r`, `v`, and `a` that represent position $\vec{\boldsymbol{r}}(t)$, velocity $\vec{\boldsymbol{v}}(t)$, and acceleration $\vec{\boldsymbol{a}}(t)$. Each of these arrays must have size $3 \times N$. The header for your function should look like:

```
function[r, v, a] = evolve(t, r0, v0, f_a)

        insert your code here ....

end
```

The midpoint algorithm performs a time step in two parts. First, we evolve to a time half way between time values using a Euler time step,

$$\begin{aligned} t_{\frac{1}{2}} &= t_{i-1} + \frac{dt}{2} \\ \vec{\boldsymbol{v}}_{\frac{1}{2}} &= \vec{\boldsymbol{v}}_{i-1} + \vec{\boldsymbol{a}}_{i-1}\frac{dt}{2} \\ \vec{\boldsymbol{r}}_{\frac{1}{2}} &= \vec{\boldsymbol{r}}_{i-1} + \vec{\boldsymbol{v}}_{i-1}\frac{dt}{2} \end{aligned} \tag{2}$$

and the use these to find $a_{\frac{1}{2}}$

$$\vec{\boldsymbol{a}}_{\frac{1}{2}} = \vec{\boldsymbol{f}}_a(t_{\frac{1}{2}}, \vec{\boldsymbol{r}}_{\frac{1}{2}}, \vec{\boldsymbol{v}}_{\frac{1}{2}})\,.$$

Next, we use the halfway values to go the who time step

$$\begin{aligned} t_i &= t_{i-1} + dt \\ \vec{\boldsymbol{v}}_i &= \vec{\boldsymbol{v}}_{i-1} + \vec{\boldsymbol{a}}_{\frac{1}{2}}dt \\ \vec{\boldsymbol{r}}_i &= \vec{\boldsymbol{r}}_{i-1} + \vec{\boldsymbol{v}}_{\frac{1}{2}}dt \end{aligned} \tag{3}$$

and then find $\vec{\boldsymbol{a}}_i$:

$$\vec{\boldsymbol{a}}_i = \vec{\boldsymbol{f}}_a(t_i, \vec{\boldsymbol{r}}_i, \vec{\boldsymbol{r}}_i)\,.$$

We repeat for each element of the time array using a `for` loop.

Testing the `evolve3D()` function

Now that we've built an ODE solver, it's time to test it. We will test it by choosing a physical system that has a known analytic solution, and then comparing our numerical solution with the analytic one. Suppose a ball of mass 0.5 kg is placed on the ground and then struck to give it an initial speed 50 m/s with elevation angle 30°. Choose your coordinates so that z is vertical, the ball is confined to the x-z plane, and the ball starts at the origin.

1. Write an anonymous function for the acceleration of this system, evolve it using your ODE solver, and then use `stem3()` to plot your results. You may want to adjust your time array until you get a satisfying figure. Overlay the analytic version of $\vec{\boldsymbol{r}}(t)$ to test that our code works.
2. Repeat using a quadratic drag model with $|\vec{\boldsymbol{F}}_d| = \alpha v^2$ with $\alpha = 5 \times 10^{-3}$ Ns^2/m^2. Overlay this plot with the previous one without drag.
3. Upload your code to the lab dropbox.

Bend it like Beckham

Everyday experience on the playground has taught us that if we add spin to a ball when throwing it, the ball will curve. This effect is due to the ball's surface friction, which drags air along the ball's surface as

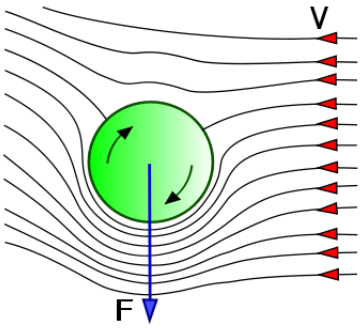

Figure 1: Illustration of the Magnus Force

it spins. This create a pressure differential that gives rise to a force that accelerates the ball. This force is called the **Magnus** force and a pictorial representation of it is shown in figure 1.

In order to model the Magnus force, we need a unit vector that represents the direction of the spin. This unit vector will be called $\hat{\boldsymbol{\omega}}$ and its direction is given by the right hand rule. If you curl your right hand in the same sense as the rotation, then your thumb will point along $\hat{\boldsymbol{\omega}}$. With this convention, the Magnus force is given by

$$\vec{\boldsymbol{F}}_M = \beta |\vec{\boldsymbol{v}}|^2 (\hat{\boldsymbol{\omega}} \times \hat{\boldsymbol{v}}) \, .$$

Download the following simulation of David Beckham's famous free kick. It consists of 3 files:

```
Bend_it_like_Beckham.m
plot_soccer_field_with_wall.m
rgb.m
```

The first two files are self explanatory, and the last file provides interesting colors. You'll need to finish the script `Bend_it_like_Beckham.m`.

1. On line 25, finish the definition of `omega_hat`. It will need to be a 3-dim column (unit) vector.
2. The drag coefficient α is given on line 42 and the Magnus force coefficient β is given on line 48. Using these coefficients, finish the definition of the acceleration function on line 53.
3. Run the code to see if you score a goal. If not, choose another value for `omega_hat` until you do. When you succeed in scoring show your code to the TF and explain how you chose your $\hat{\boldsymbol{\omega}}$.