ACM 11: Homework 8

Assigned Wednesday, Nov 26 2008. Due on Wednesday, Dec 3 at noon. 50 pts.

Submission instructions: follow the format of the Mathematica problem set template in the handout section, and submit the notebook file in the format `Firstname_Lastname_4.nb` to `ftp.its.caltech.edu/pub/srbecker/incoming`. Follow the standard instructions for resubmissions.

1. **Raptors, redux.** Recall the raptor problem: you are at the center of a 20m equilateral triangle with a raptor at each corner. The top raptor has a wounded leg and is limited to a top speed of 20 m/s. Write your position as $h(t)$ and that of the raptors as $r_i(t)$ where $i = 1, 2, 3$. Assume that at each time, the raptors run directly at you with constant speed v_i, then we model each raptor's motion with
$$\frac{dr_i}{dt} = v_i \frac{h(t) - r_i(t)}{\|h(t) - r_i(t)\|_2}.$$
We take $v_1 = v_2 = 25$ m/s, and $v_3 = 20$ m/s is the speed of the wounded raptor.

   For simplicity, we'll assume you run in a constant direction with speed $v_h = 6$ m/s, so $h(t) = v_h t\mathbf{d} + h(0)$, where $\mathbf{d} \in \mathbb{R}^2$ is your initial direction ($\mathbf{d}$ is a unit vector).

   (a) Define a variable `d` as the unit vector pointing 30° to the horizon. Write a function `hx` that takes t as an argument and returns your x coordinate at time t. Write a function `hy` that takes t as an argument and returns your y coordinate at time t. Refer to `d` in the definition of these functions.

   (b) The syntax of the `NDSolve` command, which numerically solves a system of differential equations, is much like that of the `DSolve` command, except you must specify what range of values of the independent variable to solve the system over. Specifically, the syntax is `NDSolve[equations, dependentvars, {t, start, stop}]`. Here `equations` is a list of the differential equations and initial conditions of the systems, `dependentvars` is a list of the functions to be solved for, and `{t, start, stop}` specifies the range of values of the independent variable to solve the system over. The result of the `NDSolve` command is a list of substitution rules which replace the `dependentvars` with anonymous functions. *Review the examples in the help* as necessary to familiarize yourself with this syntax. In particular, recall our discussion in Lecture 9 of how to use the rules returned from `DSolve` to define functions.

   We will denote the x and y coordinates of the i-th raptor as `rx1` through `rx3` and `ry1` through `ry3` in Mathematica – these will be the `dependentvars` when we call `NDSolve`. Define variables `r1eqnx, r1eqny, r2eqnx, r2eqny, r3eqnx, r3eqny, r1xinit, r1yinit, r2xinit, r2yinit, r3xinit, r3yinit` as the equations to be satisfied by the appropriate variables. For example, `r1eqnx = r1x'[t] == ...;` where, from the equation governing the motion of raptor one, we know ... is an expression that depends on `hx[t], hy[t], r1x[t], r1y[t]`. Also, as an example of an initial condition, you may take `r1xinit = r1x[0] == -10;`.

   Be careful that you define these equations appropriately. In particular, be careful in your use of `==` and `=`. Remember the former defines an equation, while the latter makes an assignment. Define `equations` as a list of the equations you just defined.

   (c) Now we have `equations, dependentvars`, all that is left to specify is `start, stop`. Unfortunately, we don't know when exactly we'd like to stop– we only know that we'd like to stop when you get eaten by a raptor.

   Fortunately, we can call `NDSolve` in such a way that it will stop on an arbitrary event. We'll use a modification to the syntax: `NDSolve[equations, dependentvars, {t, start, ∞}, Method->{EventLocator, "Event"->stopfun[t]}]` will evaluate the function `stopfun` as the system is being solved, and when `stopfun` first returns `True`, the solver will terminate.

Of course `stopfun` may refer to the `dependentvars` in its definition. Note that we changed `stop` to ∞ since we don't know exactly when the solver should terminate.

Write a function `stopfun` which takes `t` as an argument, and returns `True` if any of the raptors are within .001 meters of you.

(d) Call `NDSolve` with `equations, stopfun, dependentvars`, etc. The result should be a list of substitution rules for the `dependentvars`, like

```
{{r1x->InterpolatingFunction[{{0.,0.4471}},<>],
r1y->InterpolatingFunction[{{0.,0.4471}},<>],...  }
```

Look at the help for `InterpolatingFunction`. It is essentially an anonymous function that Mathematica uses to represent a function that interpolates between some data points– in this case, the data points are the samples generated by `NDSolve` for each of our `dependentvars`.

Use the rules returned by `NDSolve` to define functions `rx1, rx2, rx3, ry1, ry2, ry3` for the coordinates of the raptors.

Now, we also need to know the time at which you were eaten by a raptor. From the help on `InterpolatingFunction`, we can deduce that this time is `rx1[[1,1,2]]` (e.g. in this case, that expression would have value 0.4471). Save this time as the variable `stoptime`.

(e) Use `ParametricPlot` to plot your location and that of the raptors from time $t = 0$ to `stoptime`. Turn off the axes and turn on a frame (`Axes->False, Frame->True`).

(f) Compare the process of solving the raptor problem in Mathematica with that of solving it in Matlab. Which do you prefer, and why?