ACM 11: Homework 3

Assigned Tuesday October 14, 2008. Due Wednesday October 22 at noon. 50 points. Instructions are identical to Homework 2. Please display your email address so we can email grades. Specifically, here are the instructions:

Create a directory entitled `Firstname_Lastname_3`, where you replace `Firstname` and `Lastname` with your first and last names, respectively. Save your solution to this problem set in this directory; when finished, follow the directions on the course site to submit the directory by FTP. The main program– the one that the TAs will execute– should be named `main.m`; give the auxiliary functions referred to in the assignment reasonable names. If you need to submit revisions of your solution, follow the instructions provided on the course site.

Begin `main.m` with a title comment containing your name and email address. This information should also be displayed in the command window when your script runs. Output the answers to all questions to the command window, properly labeled, including the problem number; if you output a number, indicate what has been measured, and if you provide an explanation, indicate what phenomenon you are explaining. All values in the command windows should be the result of deliberate display statements (e.g. `disp` or `fprintf`) and not MATLAB default evaluations! This documentation is required! Put simply, your output should be easy to read and understandable without reference to your code. Your grade will be almost entirely determined by the **output** of `main.m`, not the code itself.

Each graph that is requested, or which you found useful while solving a problem (e.g. one which supports an explanation), should be created in a new figure (using the `figure` command), properly labeled, and referred to in your output by figure number. Be sure to put `clear`, `clc`, and `close all` as the first executable instructions in `main.m`.

1. Raptors. Scientist/cartoonist Randall Munroe poses an interesting question about optimization in his XKCD comic. We can answer question 2 (see Figure 1) using MATLAB . First, formulate the problem as an ODE. Let the human position be $h(t) \in \mathbb{R}^2$, and a raptor's position $r(t) \in \mathbb{R}^2$. We assume a raptor is not good at predicting the human's location, and at any time t, the raptor runs directly at the human's current position. If the raptor's speed is constant (for simplicity, we also assume instantaneous acceleration at the beginning) at, say, v_r, then we can model the raptor's motion as

$$\frac{dr}{dt} = v_r \frac{h(t) - r(t)}{\|h(t) - r(t)\|_2} \tag{1}$$

   The three raptors each satisfy this ODE separately, i.e. their motions are not coupled. However, the x and y components of a raptor's motion are coupled, so we have 2-dimensional ODEs.

   For simplicity, we assume the person runs in a constant direction and at a constant speed; thus $h(t) = v_h t \frac{\mathbf{c}}{\|\mathbf{c}\|_2} + h(0)$, where $\mathbf{c} \in \mathbb{R}^2$ is an initial direction. Then, substitute $h(t)$ into equation 1 to obtain $\frac{dr}{dt} = F(r, t)$ for some function F.

   We will use Munroe's setup, with a small modification (and assuming instantaneous acceleration). As in the comic, the human is at the center of a 20m equilateral triangle, and the human has a top speed of 6 m/s and the healthy raptors have a top speed of 25 m/s. However, if the injured raptor can only run at 10 m/s, then the best strategy for the human is to run directly at the injured raptor. To make the problem more interesting, assume the injured raptor can run at 20 m/s (bonus question: what is the critical speed of the injured raptor at which the optimal strategy is no longer to run directly towards it?).

   (a) Write a MATLAB function that, given an initial angle of the human, calculates the time until the human is caught and devoured by the raptors (call this t_{devour}). Assume that the human is "caught" when a raptor is within .1 meters. In MATLAB , use an ODE solver such as `ode45`. To improve performance, you may reduce the absolute tolerance of

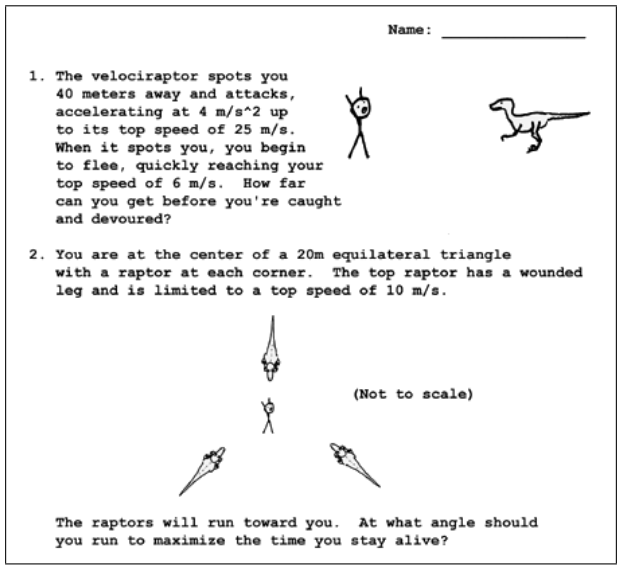

Figure 1: From the XKCD comic. We can use MATLAB to solve problem 2. The full comic can be found at `http://imgs.xkcd.com/comics/substitute.png`

the solver to 10^{-3}. How long will the human live if he/she runs at an angle of 30° degrees above the horizontal? For grading, we will also test your function on an arbitrary angle. Your function should accept angles in degrees; internally, you may convert to radians using `deg2rad` if you wish. Hint: in the function you pass to the ODE solver, you may declare $\frac{dr}{dt} = 0$ when $\|r(t) - h(t)\|_2 < .1$. This should improve performance. 15 points.

(b) Now, using one of MATLAB's non-linear optimization functions (see `help funfun` for a list of functions) in combination with the function from part (a), find the optimal angle at which the human should run to maximize t_{devour}. What is the *least* optimal angle (or, most optimal from the raptor's point-of-view)? Is this surprising? 15 points.

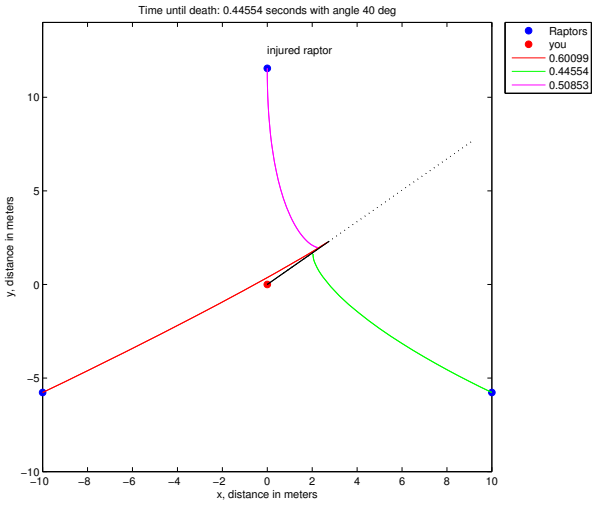

Figure 2: An example of the output from solving the raptor ODE (you do not need to make a plot for your homework). The person lives for less than half-a-second by choosing to run at a 40° angle. It takes .60 seconds for the left raptor to reach the person, .446 seconds for the right raptor, and .509 seconds for the top (injured) raptor. If the top raptor is seriously injured, the optimal strategy is to run straight at it.

(c) Bonus: Solve the ODE analytically. Or, make a more advanced ODE model that accounts for a limited predictive ability of the raptors (i.e. the raptors don't run at the human's current position, but instead run at some predicted location of where the human will be using, e.g., a linear model of the human's movement, and a simple model of the intercept time). Solve this advanced ODE numerically in MATLAB . Or, with this advanced model (perhaps with some time delay added in), allow the human to run in a more complicated pattern. Is there a clear strategy? Note: answering these questions would make for a nice final MATLAB project.

2. Fast convolution. Consider two vectors f and g with lengths N_1 and N_2 respectively. We define the vector $f \star g$ of length $N_1 + N_2 - 1$ as follows:

$$(f \star g)(j) = \sum_{i=-\infty}^{\infty} f(i)g(1+j-i) \quad \text{for} \quad j = 1, \ldots, N_1 + N_2 - 1$$

with the convention that $f(i) = 0$ if $i \notin [1, N_1]$ and $g(i) = 0$ if $i \notin [1, N_2]$ (and hence the sum is not really an infinite sum). If the vectors are assumed to start at 0 ($f(i) = 0$ if $i \notin [0, N_1 - 1]$), then the formula is $(f \star g)(j) = \sum_i f(i)g(j-i)$ for $j = 0, \ldots, N_1 + N_2 - 2$.

(a) Write a MATLAB function that takes as input two vectors f and g and returns the convolution $f \star g$. Implement the convolution however you like, except do not use the FFT. You may test your code for accuracy by using MATLAB's `conv` function, but of course you may not use `conv` within your funcion. To prove that your code works, seed `randn`'s Ziggurat algorithm with the seed 2008, then define `f = randn(80000,1); g = randn(40000,1);`, and display the 30th element of the output from your function (do NOT print out the entire output!). 10 points.

(b) Now, write a new function that takes the same input and gives the same output, except use the FFT in the calculations. Here's how:

The FFT can be used to find the circular convolution of two vectors of the same length; if you haven't seen this result, then just accept it as fact. To perform this, the operation is `ifft( fft(f).*fft(g) )`. We transform f and g into the Fourier domain, where convolution becomes element-wise multiplication, and then transform back to the original domain via the Inverse FFT.

For f and g of different lengths, and if we want a normal convolution and not a circular convolution (a circular convolution assumes the vector entries repeat periodically), we can pad the vectors with zeros. In MATLAB , the `fft` function will automatically pad with zeros if we call the function as `fft(f,N)` for any integer `N >= length(f)`. For the convolution, set $N = N_1 + N_2 - 1$ or larger (you may wish to make it larger, e.g. a power of 2, for speed benefits). Is your code comparable in speed to MATLAB's `conv` function?

Test your code as in part (a). 5 points.

(c) Keeping f fixed as before (with length 80000), vary the length of g from 10 to 10000 with 8 exponentially-spaced points (use MATLAB's `logspace` function). Plot the execution time of both convolution functions as a function of the length of g, and label the plot. The y-axis should be logarithmic. 5 points.