Midterm A1 / A2 / B1 / B2
Section 4 / 2 / 4 / 2

November 9, 2014

Problem 2 / 3 [10pt]: Finish the following function. For input: f is **a** function so that $f(x_j)$ returns the value of **an** interpolating polynomial at x_j and x is a row vector of values. For output: y is a row vector of the values $f(x_j)$ for each element in x. Assume that $f(x_j)$ only accepts *one value at a time*.

function y = evaluateF(f, x)

end

This is a *very* tricky problem. When I sat down to take this exam, I almost missed this one completely. In the previous problem, you either wrote the function $f(x)$ as a linear system (A1 / A2) or you found a Lagrangian polynomial named $f(x)$ (B1 / B2). In any case, you've done a lot of work to accurately describe $f(x)$. If this problem were stated like this

Problem 2 / 3 [10pt]: Finish the following MATLAB function named "evaluateF." For input: g is a function so that $g(t_j)$ returns the value of **an** interpolating polynomial at t_j and t is a row vector of values. For output: y is a row vector of the values $g(t_j)$ for each element in t. Assume that $g(x_j)$ only accepts one value at a time.

```
function y = evaluateF( g, t )


end
```

then what we were asking may have been a bit more clear. The first part of solving this problem is recognizing that the f in this problem is *not necessarily* the same f from the last problem. With that in mind, let's look back over the statement of **Problem 2 / 3** so that we can identify all of our given information, and decipher what output we want to produce.

The problem statement tells us that we are given two input variables. One is a function f which takes only one value at a time, the other is a row vector, x. Since f takes only one value at a time, we can't do this $f([1, 2])$, or this $f([x_1, x_2, x_3])$, or this $f([x_1, x_2, \ldots, x_n])$, or so on. The best we can hope to do is throw one value into f at a time, like this $f(1), f(2)$, or this $f(x_1), f(x_2), f(x_3)$, or this, $f(x_1), f(x_2), \ldots, f(x_n)$.

We also know that x is a row vector. But we don't know how many entries it has. So as I'm getting my head around this problem, I'm just going to write down what x looks like in general. We have

$$x = [x_1, x_2, \ldots, x_m]$$

for some integer m with $m \geq 1$. That is, x may have 100 entries, it may have 1000 entries, or it may have only one entry. In any case, we want our code to be able to take a given row vector (of any length) and compute ... what? Well, the problem says that the output, y, needs to be a row vector of values $f(x_j)$ for *each* element in x. Since I just wrote $x = [x_1, x_2, \ldots, x_m]$ above, it's pretty obvious how y will look. We'll get

$$y = [f(x_1), f(x_2), \ldots, f(x_m)]$$

Stop! After doing some analysis, let's review the problem statement and make sure we're on the right track.

Have we accounted for every bit of information given in the statement above? We've looked at x, f, and we've got a descent candidate for y. Since the statement tells us that y is a row vector of the values $f(x_j)$ for *each* element in x, and since x has the form $x = [x_1, x_2, \ldots, x_m]$, then we can conclude that the vector y (which we want to make our output) needs to have the form $y = [f(x_1), f(x_2), \ldots, f(x_m)]$. But we know that we can't simply do this

$$y = f(x)$$

because $x = [x_1, x_2, \ldots, x_m]$. We were told *explicitly* that $f(x) = f([x_1, x_2, \ldots, x_m])$ can **not** work because f takes only one value at a time. Then we'll need to come up with another way to fill a row vector y with the m values $f(x_j)$. Notice that we know how big our row vector y has to be. Each entry in y corresponds to a given entry in x. Then y and x must be row vectors of the same size. In summary, we have

$$x = [x_1, x_2, \ldots, x_m]$$

and we want

$$y = [f(x_1), f(x_2), \ldots, f(x_m)].$$

Stop! Did we get all of the information we need from the problem statement? I think so. The two relations above describe the entire problem. We have a row vector with m entries and we want to construct a row vector y with m entries. So we're ready to move on.

How big is m? Well, it could be 1. It could be 2. It could be $2^{16} - 1$. *We don't know how large m is.* The only thing we *do* know is that m is the length of x. On page 2 of the exam, you were given the MATLAB command `size(x)`. The exam states that for a vector `x`, we have $D =$ `size(x)` , where D is the number of elements in `x`. This is actually inaccurate. If `x` is a vector or matrix (of any size), then $D =$ `size(x)`, where D is a 1×2 row vector, whose entries are the dimensions of `x`.

Earlier, I noted that we did a lot of work in the previous problem in order to get information about some $f(x)$, and that the f in this problem is not *necessarily* the same f. Maybe your boss gave you some data and told you to work on getting an f to fit that data. Then later that morning, you get an email from your boss with a vector x and some function f with some weird stipulations on it. It may or may not have anything to do with the f that she told you to work on earlier! I agree that this is a very tricky problem, but that is precisely the point. Computational science and math require careful attention to detail.

Suppose that this is the mysterious f, which happens to have nothing to do with your previous work.

```
function out  = f( x )
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%   FUNCTION:    f(x)
%
%      INPUT:     x    a real or complex
%                      scalar value with
%                      positive real part
%
%     OUTPUT:    out   Gamma(x+1)
%
%DESCRIPTION:
%
% Given a scalar, x, f(x) returns the value
% of the Gamma function evaluated at x+1.
%
% The Gamma function is the continuous
% extension of the factorial function to real
% (and complex) numbers. For any positive
% integer, n, we have
%
%     Gamma(n - 1) = n!
%
% You can think of Gamma as the function
% which "fills in" all the points between
% the integers. For instance, we know that
%
%     3! = 3 * 2 * 1
% and
%
%     4! = 4 * 3 * 2 * 1
%
% But how do we get a (3.5)!, or a (pi)! ?
% It turns out that if x is one of those
% "missing points" between the integers,
% we can compute
%
%     Gamma(x) = (x-1)!
%
% But I don't want that. I want a new
% function f(x) which gives me
%
%         f(x) =  x!
%
% So I want f(x) = x! = Gamma(x+1).
% Looking up the formula for the Gamma
% function on wikipedia, I find that
%
%Gamma(x) = integral(@(t) t.^(x-1).*exp(-t),0,Inf)
%
% Then I want
%
% output = int( t.^(x).*exp(-t),0,Inf) = Gamma(x+1)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
out = integral(@(t) t.^(x).*exp(-t),0,Inf);

end
```

I urge you to either (a) copy and paste everything (except the page number) from "`function out =...`" all the way to "`end`" into a blank file. Name this file "f.m" and make sure it is in your current working directory or (b) in your MATLAB environment, select the drop down arrow under "New" and select "Function." Once you do that, a new file titled `untitledN.m` will appear. `N` will depend on the number of untitled files you're currently working on. If you have none, then your file will be named "`untitled.m`" and will look like this.

```
function [ output_args ] = untitled( input_args )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here


end
```

From here, you can change your new file to look like the one I made. You only have to copy and paste one line and change the first line a little. A MATLAB function must have the same filename as the function name. If you've changed `untitled` to `f`, then when you go to save the file, MATLAB will already suggest that its name should be f.m. If you changed the top line so that it reads "`function out = myWickedCoolFunction( x )`, then MATLAB will suggest that its name should be `myWickedCoolFunction.m`, and so on. You need to name the file so that the two names are the same. So if I did this

```
function [ Y ] = continuousFactorial( X )

% Y = Gamma(X+1) = X!
%               (for X = a + i*b with a > 0)

Y = integral(@(t) t.^(x).*exp(-t),0,Inf);

end
```

then I would need to make sure I save my file as `continuousFactorial.m` But let's move on as though you've saved your function as `f.m`. Making sure that the file `f.m` is in your Current Folder (one of the windows in the MATLAB environment), also called your current or working directory, then you can type `f(3)` into the command window. The returned value should be 6.000, if you're only showing 4 decimal places. You can type `format long` into the command window, hit return, then type `f(3)` again. You'll see that this is a *very* good numerical approximation to the true value, $3! = 6$. Now try this. Define a vector x by typing `x = 0:0.01:3;` Then try `f(x)` and you'll get all kinds of scary errors. This is what we mean by "$f(x_j)$ only accepts one value at a time."

Think about the utility of having a function which can take a vector *of any* length.