Week 4 - Numeric vs. Symbolic

Reminder - the best place to learn MATLAB (or anything, really) is the internet! StackOverflow and MathWorks' own MATLAB Exchange are filled to the brim with people asking and answering questions about MATLAB. MATLAB's own documentation is also extensive and extremely helpful. It includes descriptions of how to call functions as well as usage examples.

Download this page as a PDF here (this document is generated automatically - print this page to PDF from your browser for the best result).

Return to Lectures

Note: this lesson will require the Symbolic Math Toolbox

What does Numeric mean?

I'll start with an example: everything we have done so far in this course. All of the functions we have written take in a value and return a value, which makes them numeric. In a way, the function itself does not exist and is only a set of directions for changing one value to another. The term Numeric refers, in a way, to the nature of the function. All numerical calculations in a computer are fundamentally just numerical approximations. You can throw as many bytes of memory at a decimal as you want, but you will never be able to represent something like pi with perfect accuracy.

What does Symbolic mean?

Whereas Numeric functions are simply a set of instructions and can be thought of as a set of values mapped to another, symbolic functions are the opposite. When you enter a symbolic function into MATLAB, it is stored as the function itself. It is often difficult or impossible to directly give these functions a value and retrieve an output - but because the function *itself* is stored, you can do things like take the derivative programmatically. Additionally, because you are not storing the function in the same way as a above, you can reach 'better accuracy' (note the quotes). Pi stored numerically is *really* close to pi - nowadays, it is essentially pi. But pi stored symbolically is *exactly pi*.

Expanding on the difference between the two

As alluded to above, but never explicitly stated, storing an expression symbolically or numerically changes what you can evaluate. Let's learn by examples: ```matlab % define a numeric function sin^2(x) numeric_sin_squared = @(x) sin(x).^2; % <- notice the .^ I want this function to operate element wise on matrices. % Note - the function name reflects that it is numeric - good practice!

% I can now evaluate this function over a range of values y = numeric_sin_squared(3:0.1:5); % but if I try to take the derivative using diff() I will get an error (try it!) diff(numeric_sin_squared)

% declare x as a symbol (this will not interfere with numeric_sin_squared because % it has its own function workspace syms x % now declare a symbolic function symbolic_sin_squared = sin(x)^2; % <- we no longer need an @(x), because this function does not accept numeric input % notice the informative name % we did not have to explicitly make symbolic_sin_squared into a symbol - it was done implicitly when we defined it in terms of symbols

% now we can take the derivative with ease dsinsqdx = diff(symbolic_sin_squared); % but attempting to call the function with values will result in an error (try it!) ``` This is a really simple example, but it is easy to imagine that when you are trying to find interrelationships between the various thermodynamic state variables, evaluating partial derivatives with your computer will be immensely helpful.

Interconverting between Numeric and Symbolic

There will be many circumstances where you take a complex partial derivative of a state function and then want to evaluate its behavior at a certain point. In this case, you would want to evaluate the derivative with a symbolic function and then somehow change it to a numefical function. This is possible in MATLAB using the aptly named matlabFunction function. After you have manipulated your symbolic function, pass its variable name (technically called the *function handle*) into *matlabFunction* and it will return the equivalent numeric function.

Moving in the other direction, from numeric to symbolic, is not practical. Part of the reason it's not is because there aren't many circumstances where it would come in handy, and because conversion the other way around exists. If you are defining a function and you are unsure if you will need to differentiate at some point in the future, just leave it as a symbolic.

Black and white is for penguins... this is science!

The line between symbolic and numeric functions and variables is a blurry one, and is becoming more so everyday. In the case of variables, moving from symbolic to numeric is essentially just a semantic difference: `matlab syms x func = sin(x); % solve this function using variable precision algebra out = vpa(func == 1, x); % out is now a symbol, but can be made numeric with a simple call to double value = double(out); % you can just as easily move back and store the number itself as a symbol syms value` *documentation for vpa*

Recently, numeric and symbolic functions have been becoming more similar as well. In the below example, taken from here in the documentation, we define a symbolic function that we can also evaluate numerically. The caveat is that numerically evaluated results are still returned to us as symbols - but as we saw above, that leaves us one quick step away from a purely numeric value. ```matlab % Create a symbolic function f with variables x and y by using syms. Creating f automatically creates x and y. syms f(x,y) % Assign a mathematical expression to f. f(x,y) = x^2*y; % Find the value of f at (3,2). f(3,2)

ans = 18 ``` Such symbolical-numeric hybrid functions come with all the benefits of being symbolic and numeric, including being array-valued by default (this can save you a lot a head ache with dot operators). The drawback to trying to numerically evaluate symbolic functions, though, is the execution time. For simple functions with few points, it probably will not generate any noticeable overhead - but if you start evaluating multi-variate functions with thousands or millions of points, you may start to see the benefit of careful setup of numeric functions.

Ultimately the decision is up to you, but we highly recommend keeping your numeric and symbolic functions separate and well labeled. In simple examples it's easy to handle the mental gymnastics, but when you have functions of functions (Week 7) and are working with iterative integral calculations (Week 5) you will likely find that having them separate in the code makes it *far* easier to keep it straight in your head.