ECE-210-B Homework #2

Vectors & Matrices

Cat Van West, Spring 2024

Contents

We've now seen some more powerful vector and matrix operations, so let's use 'em! Thinking and writing vectorized code is a powerful way of improving operation efficiency (as you will see). For each computation, save the result to a variable or print it to the screen (omit the semicolon). Bonus points if you use `disp` or `fprintf` to label the output and make really pretty charts (FF loves really pretty charts).

Same submission format as the first assignment, and will be the same going forward. Have fun...

1 The Assignment

Read the rest of this document… as before. Some of it may be review, but hey – lecture notes for this class aren't super formal.

Spatial Awareness Perform the following operations *without* the use of `for`.

1. Use `reshape` and the $:$ operator to create the matrix

$$A = \begin{pmatrix} 0 & 1 & \dots & 7 \\ 8 & 9 & \dots & 15 \\ \vdots & \vdots & \ddots & \vdots \\ 56 & 57 & \dots & 63 \end{pmatrix} \text{ (should be 8 by 8),}$$

   then dot-exponentiate 2 by it to create the matrix

$$B = \begin{pmatrix} 2^0 & \dots & 2^7 \\ \vdots & \ddots & \vdots \\ 2^{56} & \dots & 2^{63} \end{pmatrix}.$$

2. Flatten B into a vector v (row or column – should be quick either way!) and extract the prime-numbered components of v (in other words, $v_2, v_3, \dots, v_{61}$). Save these in their own variable, and name it well!

   Note: MATLAB has a lot of funny little helper functions…

3. Find the geometric mean of those prime components (for some $x_1, x_2, \dots, x_n$, the geometric mean is $\sqrt[n]{x_1 x_2 \dots x_n}$ – see the documentation for `prod` and `nthroot` for how to compute this concisely).
4. Flip *one* row of A end for end (see the course notes for how to do this).
5. Delete one column of A.

Smallest Addition We're going to try out a couple variations of numerical integration and differentiation. Specifically, let's see how accurately we can evaluate

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \int_0^z \exp(-t^2)\, dt$$

and

$$\frac{d}{dt}\left[\exp(-t^2)\right].$$

MATLAB has a function for doing the former (called, unsurprisingly, `erf`), as this is a rather useful but nonelementary integral, but because we enjoy pain we'll do it by hand.

1. Evaluate $\exp(-t^2)$ at 100 evenly spaced points from 0 to 6.66.
2. Approximate the derivative of this function using `diff`. Find the mean squared error (using `mean`) between this derivative and samples of the analytical derivative over this range. (*Note:* you will likely have to truncate or pad these.)
3. Approximate the integral of the original function using `cumsum` and `cumtrapz`. This will result in two estimations of the original. Find the mean squared error against MATLAB's `erf` over the same range for each of them. (Sorry to make you do this before we hit functions, but…)

   Comment on the integral estimations – which is better? How close did they get to `erf`?

A Indexing & How MATLAB Thinks

Vectors and matrices are indexed in multiple ways. Each element in a vector or a matrix has an *index*, which uniquely identifies it (as is probably painfully evident by now). Vectors are indexed by only this number. Matrix indexing can be more complex, so I'll give a refresher here.

All indices in MATLAB start from 1. If this irks your programmer self to no end, Python provides some excellent MATLAB alternatives with zero-indexing (not to mention the expressive power of a real programming language!). Each element of a matrix has two (or more!) numbers which represent it. As in mathematics, matrices are indexed first by row, then by column, then by… whatever other dimensions you have. For a 3×3 matrix, you'd have something like this:

$$\mathtt{A} = \begin{pmatrix} \mathtt{A(1,\ 1)} & \mathtt{A(1,\ 2)} & \mathtt{A(1,\ 3)} \\ \mathtt{A(2,\ 1)} & \mathtt{A(2,\ 2)} & \mathtt{A(2,\ 3)} \\ \mathtt{A(3,\ 1)} & \mathtt{A(3,\ 2)} & \mathtt{A(3,\ 3)} \end{pmatrix}.$$

Each element of a matrix also has a single scalar associated with it, for an alternate method of indexing. Note the row-first order:

$$\mathtt{A} = \begin{pmatrix} \mathtt{A(1)} & \mathtt{A(4)} & \mathtt{A(7)} \\ \mathtt{A(2)} & \mathtt{A(5)} & \mathtt{A(8)} \\ \mathtt{A(3)} & \mathtt{A(6)} & \mathtt{A(9)} \end{pmatrix}.$$

B Notes on Numerical Calculus

The lecture notes on numerical calculus walk you through most of the code you need to know here. One important thing to note, though, is *padding*: the `diff` function returns the difference between subsequent elements of a vector, and thus will return a vector *one element shorter* than the one passed to it. For example:

```
x = [0 1 3 5 9]; % length 5
y = diff(x); % y becomes [1 2 2 4], length 4
```

If you're computing a derivative via `diff`, you might want to pad the resulting vector (depending on what you're doing with it) so that its length matches with other data you're using:

```
% x, y defined above...
dydx = diff(y)./diff(x);
dydx_pad_start = dydx([1 1:end]); % duplicates first element
dydx_pad_end = dydx([1:end end]); % duplicates last element
```

The functions `cumsum` and `cumtrapz` do not need padding and are a little easier to use – `cumtrapz` makes doing numerical integrals really easy if you pass it the right arguments. See its documentation for more details!