1 Introduction

This week we’re going to spend more time working on our ability to slice matrices (to “slice” a matrix or array means to grab certain portions of the matrix via indexing). In particular, we’ll focus on performing row exchanges, and on manipulating and plotting columns of a matrix.

2 Permuting Rows of a Matrix

We’re going to first work on the example of permuting the rows of a matrix. “Permute” is a fancy word for “rearrange” or “reorder.” We’ll first work on the special case of swapping two rows. Eventually, we’ll learn how to perform arbitrary permutations (i.e. make a whole bunch of row swaps at once). Note that the same techniques apply to permuting columns.

We’ll need some matrices to play with. Let’s use the command `magic(4)` to build a $4 \times 4$ matrix:

```matlab
>> A = magic(4)
```

```
A =

16  2  3  13
 5 11 10  8
 9  7  6 12
 4 14 15  1
```

To figure out what a “magic” matrix is, type `help magic` into the command line. In our case, it’s just some matrix we’re going to play around with.

2.1 A clunky way

Now, suppose we wanted to interchange the first and third rows of $A$. There are a few ways we could do this. An effective but somewhat clunky way is to perform the following steps:

- Create a temporary vector that stores the values of row-one.
- Replace (or “overwrite”) the values in row-one with the values in row-three
- Replace the values in row-three with the row-one values in the temporary vector

The commands we might enter are shown below:

```matlab
>> A = magic(4);
>> temp_row = A(1,:); % Make a temporary vector that holds the first row, then
>> A(1,:) = A(3,:); % Replace first row with third row, then
>> A(3,:) = temp_row; % Put vector holding first row into the third row
```
Recall that the notation $A(n,:) \text{ means “fix row } n \text{ and do something to all of }$ 
the column entries in that row.”

### 2.2 A more elegant way

Recall that if we have a vector $\text{vect}$, then we can use an “indexing vector” to reorder its elements. Let’s see how do this using the example from last week’s notes.

- $\text{>> vect} = (1:10).^2$
  
  $\text{vect} =$
  
  \[
  \begin{array}{cccccccc}
  1 & 4 & 9 & 16 & 25 & 36 & 49 & 64 & 81 & 100
  \end{array}
  \]

- An “even-odd” permutation lists all of the elements in the “even” positions of the vector first and then lists all of the elements in the “odd” positions. Applying this to the elements of $\text{vect}$:

  $\text{>> vect} = \text{vect}([2 4 6 8 10 1 3 5 7 9])$

  $\text{vect} =$
  
  \[
  \begin{array}{cccccccc}
  4 & 16 & 36 & 64 & 100 & 1 & 9 & 25 & 49 & 81
  \end{array}
  \]

- Recall from last week that the indexing vector $[2 4 6 8 10 1 3 5 7 9]$ tells Matlab which elements to put in each position. Setting $\text{vect}$ equal to be the permuted version of itself overwrites the original vector, meaning that the original $\text{vect}$ is replaced by the permuted $\text{vect}$. This idea gives us an easy to to perform arbitrary permutations of the vector.

Now, let’s extend this to matrices. We can work much the same way we did with vectors, except we’ll have to be careful about whether we’re indexing the rows or columns.

(*) Remember that we use the syntax $A(\text{row}, \text{column})$ to index matrices in Matlab.

- For example, generate a $4 \times 4$ matrix $A$ using the $\text{magic}$ command:

  $\text{>> A} = \text{magic}(4)$

  $A =$
  
  \[
  \begin{array}{cccc}
  16 & 2 & 3 & 13 \\
  5 & 11 & 10 & 8 \\
  9 & 7 & 6 & 12 \\
  4 & 14 & 15 & 1 \\
  \end{array}
  \]

- Let’s use an indexing vector to permute the rows of $A$ so that they are in the order 3, 2, 1, 4:

  $\text{>> A} = A([3 2 1 4], :)$

  $A =$
  
  \[
  \begin{array}{cccc}
  9 & 7 & 6 & 12 \\
  5 & 11 & 10 & 8 \\
  16 & 2 & 3 & 13 \\
  4 & 14 & 15 & 1 \\
  \end{array}
  \]
• This efficiently and elegantly (one line!) swaps rows 1 and 3 of the matrix \(A\). We could similarly exchange *columns* using the command \(A = A(:,[3 2 1 4])\).

### 3 More advanced permutations

The method above is nice, but we wouldn’t want to enter the indices individually for long vectors (i.e., if I have 100 elements, I wouldn’t want to enter \(\text{vect}([1 3 2 4 5 6 \ldots])\) by hand. To make this more efficient, we can use the `end` command.

• For example, we can reform the earlier code that computes the “even-odd” permutation vector so that it only permutes the first three elements:

```matlab
>> vect = (1:10).^2
vect =
     1     4     9    16    25    36    49    64    81   100

>> vect = vect([3 2 1 4:end])
vect =
     9     4     1    16    25    36    49    64    81   100
```

We can read the last portion of this indexing as “4 through end.” This would work well regardless of how large the vector `vect` is.

• Applying this idea to matrices, suppose we had a large matrix:

```matlab
>> A = magic(12)
A =
     144     2     3    141    140     6     7   137   136    10    11   133
    131    130     16     17   127   126    20    21   123   122     24
     25    119    118     28     29   115   114     32     33   111   110     36
    108     38     39    105    104     42     43    101    100     46     47     97
     96     50     51     93     92     54     55     89     88     58     59     85
     61     83     82     64     65     79     78     68     69     75     74     72
     73     71     70     76     77     67     66     80     81     63     62     84
     60     86     87     57     56     90     91     53     52     94     95     49
     48     98     99     45     44    102    103     41     40    106    107     37
    109     35     34    112    113     31     30    116    117     27     26    120
    121     23     22    124    125     19     18    128    129     15     14    132
     12    134    135     9     8    138    139     5     4    142    143     1
```

and we wanted to move the first 4 rows the matrix on the end. We know that we can do this via the command \(A = A([5 6 7 8 9 10 11 12 1 2 3 4],:),\) but we’d like to be for efficient. The following code performs the same permutation but is much easier to read and write:
>> A = A([5:end 1:4], :)

A =

```
96  50  51  93  92  54  55  89  88  58  59  85
61  83  82  64  65  79  78  68  69  75  74  72
73  71  70  76  77  67  66  80  81  63  62  84
60  86  87  57  56  90  91  53  52  94  95  49
48  98  99  45  44  102  103  41  40  106  107  37
109  35  34  112  113  31  30  116  117  27  26  120
121  23  22  124  125  19  18  128  129  15  14  132
  12 134  135   9   8  138  139   5   4  142  143   1
144   2   3  141  140   6   7  137  136   10  11  133
  13 131  130  16  17  127  126  20  21  123  122  24
  25 119  118  28  29  115  114  32  33  111  110  36
108  38  39 105 104  42  43 101 100  46  47  97
```
Perform the following steps in a script (no custom functions are necessary for this assignment):

(a) Form a $6 \times 6$ identity matrix using the command `eye(6)`.

(b) Swap the second and fifth rows of the matrix. Call the result $P$.

(c) Form a $6 \times 6$ magic matrix using `magic(6)`. Call it $A$.

(d) Left multiply $A$ by $P$ (that is, form the product $P \times A$). What is the effect on $A$ of this multiplication? Compare to the original matrix $A$.

(e) Right multiply $A$ by $P$ (that is, form the product $A \times P$). What is the effect on $A$ of this multiplication? Compare to the original matrix $A$.

(f) Right and left multiply $A$ by $P$ (that is, form the product $P \times A \times P$). What is the effect on $A$ of this multiplication? Compare to the original matrix $A$. 
