APPM 2460
Vectors & Matrices I

1 Introduction

Last time, we saw a basic example of how to find the numerical solution to first order ODEs. Before getting into systems of ODEs, we need to spend some time familiarizing ourselves with matrices and vectors in Matlab, which are the dominant data structure when writing Matlab code. We'll be focusing today on how to index matrices; that is, how to obtain certain parts of matrices.

2 Review of Vectors and Matrices

2.1 Notation

Recall that a matrix $\mathbf{A}$ is an array of numbers that is said to have *dimension* $m \times n$, meaning that $\mathbf{A}$ has m rows and n columns. The entries of the matrix are identified by a double subscript that gives the row-column position that the number occupies in the matrix. The counting system to identify each position begins in the *top left corner* of the matrix.

- For example, if we define the 2×2 (2 rows, 2 columns) matrix $\mathbf{A}$ as

  $$\mathbf{A} = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$

  we would say that the number 1 is in the $(1, 1)$ position of $\mathbf{A}$ *or* 1 is the a_{11} element of $\mathbf{A}$. Similarly, 2 is the a_{12} (row 1, column 2) element, 3 is the a_{21} element, and 4 is the a_{22} element.

A vector is a special case of a matrix with just one row or one column ($m = 1$ or $n = 1$). The same indexing convention for matrices can be applied to vectors.

2.2 Properties

There are several important properties to keep in mind while working with matrices and vectors in Matlab. These are listed below:

- *Two matrices may be added if their dimensions are the same.*
- *Two matrices may multiplied if the "inner dimensions" of the product match.* For example, suppose $\mathbf{A}$ is 2×3 and $\mathbf{B}$ is 3×4. If we want to compute the product $\mathbf{AB}$, then we should write the matrix dimensions size-by-side $(2 \times \underbrace{3) \cdot (3}\times 4)$ and check that the number of columns in $\mathbf{A}$ matches the number of rows in $\mathbf{B}$.
- *Matrix multiplication usually does not commute.* That is to say if we are considering two matrices $\mathbf{A}$ and $\mathbf{B}$, then usually $\mathbf{AB} \neq \mathbf{BA}$. In some cases, one product will be defined and the other will not. For example, if $\mathbf{A}$ is 2×3 and $\mathbf{B}$ is 3×4 as before, then the product $\mathbf{AB}$ is defined but the product $\mathbf{BA}$ is not. In other cases, both products may be defined but simply give different outcomes.
- *Matrix "division" does not exist. You must use inverses, although they do not always exist.*

3 Review of Indexing for Vectors

Let's first recall how we index vectors. In this context, "how we index" means how we find certain elements of a list, how we assign new values to them, and so on.

- Let's make a list of squared integers starting from 1^2 and ending with 10^2. We could type in

  ```
  >> v = (1:10).^2

  ans =

          1  4  9  16  25  36  49  64  81  100
  ```

  (Remember that the notation " .^" (read "dot-caret") means "apply this operation element-wise to all of the entries of `v`." As desired, the vector `v` is just the squares of the numbers 1 through 10.

- Suppose we want to select several element from this vector that occupy neighboring positions, such as elements 4 through 8. Then we type `v(4:8)`.

- If we want to select elements of `v` that do not occupy neighboring positions in the vector, such as elements 1, 5, 2, and 6 (in that order) we can type `v([1 5 2 9])` where the input vector `[1 5 2 9]` indicate the positions of the desired elements.

- If we want to select all elements occupying positions 2 through the end, we can type `v(2:end)`.

- If we want to select *all* of the entries of the vector `v`, we can use the colon symbol " : " to do this. That is, if we type `v(:)`, Matlab returns **all** of the vector `v`.

  ```
  >> v(:)

  ans =

          1  4  9  16  25  36  49  64  81  100
  ```

  ($\star$) This will become important when we begin to work with matrices.

4 Matrix basics

Now that we remember how to index vectors, let's see how to approach matrix indexing.

4.1 Defining matrices

Entries in a matrix are written down one row at a time. The *matrix columns are separated by commas or spaces*, just like vector elements, *while the row are separated by semicolons.*

- As an example, suppose we want to enter the matrix

  $$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

  into Matlab. One-by-one, we need to enter the rows `1 2 3`, `4 5 6`, and `7 8 9` and we need to separate these row by semicolons.

  Try typing

```
>>  A = [1 2 3; 4 5 6; 7 8 9]

A =
     1     2     3
     4     5     6
     7     8     9
```

You can see how the semicolons separate the rows of the matrix.

4.2 Matrix indexing

- Accessing elements of a matrix is similar to a vector, except you have two indices instead of just one. To pick out a specific entry of the matrix **A**, you would type `A(i,j)` where *the first number in the pair identifies the row index and the second identifies the column index.* (That is, if you typed in `A(i,j)` you would be asking Matlab for the number in the i^{th} row and the j^{th} column).

  Using the example matrix A, try typing $A(2,3)$ into the command line. Matlab will return 6, since this is the entry in the 2^{nd} row and 3^{rd} column of **A**.

- Now let's use the colon notation that was introduced in the review of vectors. If you wanted to access the first column of **A**, you would type `A(:,1)` (read: all rows, column 1).

  ```
  >> A(:,1)

  ans =

       1
       4
       7
  ```

- In the same way, you can access the first row of **A** by typing `A(1,:)` (read: row 1, all columns.).

  ```
  >> A(1,:)

  ans =

       1     2     3
  ```

- You could select multiple rows of **A** using syntax that is similar to what we used to pick out multiple **elements** of a vector. Try typing `A(:,1:2)` (read: all rows, columns 1 through 2):

  ```
  >> A(:,1:2)

  ans =

       1     2
       4     5
       7     8
  ```

- Just as we reordered the elements of the vector `v` when we entered `v([1 5 2 6])` above, we can reorder the rows of **A**:

  ```
  >> A([3 1 2], :)
  ```

```
ans =

    7     8     9
    1     2     3
    4     5     6
```

- We can even select a sub-matrix of **A** by selecting a subset of the rows and columns at once:

```
>> A(2:3,1:2)

ans =

     4     5
     7     8
```

You should play with this notation until you get the hang of it.

4.3 Matrix multiplication

Matrix multiplication is a little harder than normal multiplication, and hopefully you learned about it in class! In Matlab, matrix multiplication is performed by using the * (shift 8) symbol.

- As an example, define a matrix of all ones by typing

```
>> B = ones(3,3)

ans =

    1    1    1
    1    1    1
    1    1    1
```

Now multiply the matrix A that was defined above by the new matrix B. You should get the result

```
>> A*B
ans =

     6     6     6
    15    15    15
    24    24    24
```

Basically, the $(i,j)^{th}$ entry in the product is the dot product of the i^{th} row of A and the j^{th} column of B. **Writing `A*B` does NOT give you the first element of `A` times the first element of `B`, and so on. Instead, `A*B` performs matrix multiplication.**

If you want the "element-wise" product of **A** and **B** instead of the matrix product, then you need to use "dot" notation. If you do A "dot-times" B, the answer is quite different:

```
>> A.*B
ans =

     1     2     3
     4     5     6
     7     8     9
```

Again, element `(3,2)` of the matrix `A.*B` is `A(3,2)*B(3,2)`, while element `(3,2)` of `A*B` is the dot-product of the third row of **A** and the second column of **B**.

4.4 Matrix indexing using for loops

Sometimes it's useful to populate a matrix with values using nested for-loops. Let's say you want to build a matrix whose elements were the sum of the row and column index. That means that, for example, $\mathbf{A}(1,1) = 1 + 1 = 2$, $\mathbf{A}(1,2) = 1 + 2 = 3$, ... $\mathbf{A}(i,j) = i + j$. To build such a matrix, write the following function:

```
function A = buildMatrix(n)

A = zeros(n,n);             % Matrix of zeros to pre-allocate memory

for i = 1:n                 % The first loop moves across the rows
    for j = 1:n             % The second loop moves across the columns
        A(i,j) = i+j;
    end
end

end
```

Note that **A** is initialized by writing `zeros(n,n)`. As mentioned before, this is important for speed.

APPM 2460 **Homework** **Week 6**

Submit a published pdf of your script and any other supporting code needed to solve the following problem to Canvas by Monday, Oct. 12at 11:59 p.m. See the 2460 webpage for formatting guidelines.

This is the 5×5 Hilbert matrix.

$$\begin{bmatrix} 1 & 1/2 & 1/3 & 1/4 & 1/5 \\ 1/2 & 1/3 & 1/4 & 1/5 & 1/6 \\ 1/3 & 1/4 & 1/5 & 1/6 & 1/7 \\ 1/4 & 1/5 & 1/6 & 1/7 & 1/8 \\ 1/5 & 1/6 & 1/7 & 1/8 & 1/9 \end{bmatrix}$$

(a) What is the pattern for entries of this matrix*? (Hint: The entries only depends on the row and column index).

(b) Using this pattern, write a function called `myHilb.m` that constructs an $n \times n$ Hilbert matrix.

(c) Write a script that shows the function in action for two different values of n.

(* You do *not* need to explicitly answer question (a) in your homework. We will be able to see if you found the correct pattern from your code.)