Matrices and Loading Files

Leigh J. Little SUNY Brockport October 26, 2022

# Contents



# 1 Matrices

We briefly examined matrices when we looked at vectors. A matrix can be viewed as a set of scalars arranged into a rectangular grid. From before,

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}; \qquad B = \begin{bmatrix} 3 & -2 \\ 0 & 5 \\ -1 & 9 \end{bmatrix}; \qquad C = \begin{bmatrix} 5 & 2 & -1 \\ 0 & 8 & -7 \end{bmatrix}.$$

Here, we say that the matrix $A$ is $3 \times 3$ because it has 3 rows and three columns. Similarly, $B$ is $3 \times 2$ and $C$ is $2 \times 3$.

As with vectors, each individual element of a matrix can be referenced, but this is done using a double subscript. For example,

$$A_{1,3} = 3$$

because the element in row 1 and column 3 of $A$ is 3. Similarly,

$$B_{2,2} = 5$$

and

$$C_{2,3} = -7.$$

## 1.1 Creating Matrices in MATLAB

Matrices can be created in a manner similar to vectors. To create the matrix $A$, enter the following

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

We begin by using the array constructors. Within these, we first enter the first row of elements, the put a semicolon (`;`) to go down to the next row. Then the second row is entered, followed by another semicolon. Finally, the last row of elements is entered.

Similarly, we can create the matrices $B$ and $C$ using

```
>> B = [3 -2; 0 5; -1 9]
B =
   3    -2
   0     5
  -1     9
>> C = [5 2 -1; 0 8 -7]
C =
   5     2    -1
   0     8    -7
```

## 1.2 Creating Matrices from Vectors

As mentioned previously, a matrix can also be viewed as a series of column vectors placed next to each other or a series of stacked row vectors. This idea can be used to create matrices from vectors rather than scalars. Suppose we have the column vectors

$$x = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}; \qquad y = \begin{bmatrix} 2 \\ 5 \\ 8 \end{bmatrix}; \qquad z = \begin{bmatrix} 3 \\ 6 \\ 9 \end{bmatrix};$$

Then we can create the original matrix $A$ by doing

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

Notice that in creating $A$ we have used the same general syntax that we have been using for creating vectors. In this case, rather than scalar elements between the array constructors, we have placed vector elements. This can be done provided $x, y$ and $z$ all have the same number of elements.

If instead we had the row vectors

$$a = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}; \qquad b = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}; \qquad c = \begin{bmatrix} 7 & 8 & 9 \end{bmatrix};$$

we could create $A$ by doing

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

Notice the slight difference in syntax when creating $A$. Here we want to stack the vectors, so we first create the first row of $A$ using $a$, then use a semicolon to go down to the next row. The second row is created using $b$ followed by another semicolon. Finally, we use $c$ to create the last row.

## 1.3 Extracting Matrix Elements

It is often necessary to extract part of a matrix into either a vector or a smaller matrix. This can be done using the special subscripts and array indicies.

The most common operation is of this type is extracting a row or a column. This can easily be done. Suppose we have the $A$ matrix defined as before and we wanted to extract the second column of $A$ into the vector $p$. Then we could do

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> p = A(:,2)
p =
    2
    5
    8
```

We can read the syntax used to extract $p$ as 'get all rows associated with column 2.' The syntax

```
>> p = A(1:3,2)
>> p = A(1:end,2)
```

would also work, but the first method is more convenient if the entire second column is needed.

Similarly, if we needed to extract the third row into the vector $q$ we could use

```
>> q = A(3,:)
q =
    7     8     9
>> q = A(3,1:3);
>> q = A(3,1:end);
```

Other portions of a matrix can be extracted using array indicies or specific vectors of subscripts. For example,

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(1:2,1:2)
ans =
     1     2
     4     5
```

This will extract the $2 \times 2$ submatrix consisting of rows 1 and 2 and columns 1 and 2 of $A$.

```
>> A = [1 2 3; 4 5 6; 7 8 9];
>> ii = [3 1];
>> jj = [2 1];
>> A(ii,jj)
ans =
     8     7
     2     1
```

This will extract the $2 \times 2$ submatrix consisting of rows 3 and 1 (in that order) and columns 2 and 1 (in that order) from $A$.

# 2 Loading Data from Files

Loading data from files is an important part of programming. In real situations, it is very common for the calculation portion of a simulation to be performed using a C or Fortran program. This program outputs the results to data files. These files might then get read into some other program for visualization or other post-processing purposes. This is a very complex topic because data files can be flakey. This is especially true in cases where the data is coming from some kind of physical measuring device like a radiosonde that takes readings of atmospheric data. Very frequently, data from such devices might be missing at certain altitudes (for example, a bird might have flew into the device).

The easiest way to deal with data stored in files is to be the one who creates the file in the first place. This allows you complete control over how the data is presented. However this is not always possible. MATLAB allows for several mechanisms for reading data (including reading data from Excel files), but we will keep this simple for now.

We will assume that the data in the file consists of either a single column of numbers or a rectangular grid of numbers. This means that there are no labels such as column headers or other text in the file. In the event the data file matches this description, the values can easily be read into MATLAB using the `load` command. Consider the file `data1.dat` on the course website. This consists of 4 columns of data. If we want to read this into MATLAB we can do

```
>> load data1.dat
>> data1
data1 =
    0.7577    0.7060    0.8235    0.4387
    0.7431    0.0318    0.6948    0.3816
```

```
0.3922    0.2769    0.3171    0.7655
0.6555    0.0462    0.9502    0.7952
0.1712    0.0971    0.0344    0.1869
```

By examining the workspace pane of the MATLAB environment, we can see that this command created a matrix named `data1`. We can view the contents using either the command line or by double clicking on the variable name in the workspace panel (this will bring up something that looks like an Excel sheet).

In situations like this the data is most often organized columnwise (meaning each column represents a physical quantity). In the above example, the first three columns might be the $(x, y, z)$ coordinates of a point in three-dimensional space and the fourth column might be the value of the temperature at that point. Although not required, it is often helpful to extract the columns into more meaningful variable names. For example, we could do

```
>> x = data1(:,1);
>> y = data1(:,2);
>> z = data1(:,3);
>> T = data1(:,4);
>> clear data1
```

The last statement deletes the `data1` variable to free up memory. Note that this might not be feasible if `data1` is an extremely large matrix. In the event that memory limitations will not permit you to do this, you can always just refer to individual columns using `A(:,1)`, `A(:,2)`, *etc.*.