## Chapter 2: Vectors and Matrices

### Exercises

1) If a variable has the dimensions 3 x 4, could it be considered to be (**bold** all that apply):
**a matrix**
a row vector
a column vector
a scalar

2) If a variable has the dimensions 1 x 5, could it be considered to be (**bold** all that apply):
**a matrix**
**a row vector**
a column vector
a scalar

3) If a variable has the dimensions 5 x 1, could it be considered to be (**bold** all that apply):
**a matrix**
a row vector
**a column vector**
a scalar

4) If a variable has the dimensions 1 x 1, could it be considered to be (**bold** all that apply):
**a matrix**
**a row vector**
**a column vector**
**a scalar**

5) Using the colon operator, create the following row vectors

```
3      4      5      6      7      8

 1.3000    1.7000    2.1000    2.5000

     9      7      5      3
```

```
>> 3:8
ans =
     3     4     5     6     7     8
>> 1.3: 0.4: 2.5
ans =
    1.3000    1.7000    2.1000    2.5000
```

```
>> 9: -2: 3
ans =
     9     7     5     3
```

6) Using a built-in function, create a vector *vec* which consists of 30 equally spaced points in the range from –2*pi to +pi.

```
>> vec = linspace(-2*pi, pi, 30)
```

7) Write an expression using **linspace** that will result in the same as 1:0.5:3

```
>> 1: 0.5: 3
ans =
    1.0000    1.5000    2.0000    2.5000    3.0000
>> linspace(1,3,5)
ans =
    1.0000    1.5000    2.0000    2.5000    3.0000
```

8) Using the colon operator and also the **linspace** function, create the following row vectors:

```
-4     -3     -2     -1     0

     9     7     5

     4     6     8

     >> -4:0
     ans =
         -4    -3    -2    -1     0
     >> linspace(-4, 0, 5)
     ans =
         -4    -3    -2    -1     0
     >> 9:-2:5
     ans =
          9     7     5
     >> linspace(9, 5, 3)
     ans =
          9     7     5
     >> 4:2:8
     ans =
          4     6     8
     >> linspace(4,8,3)
     ans =
          4     6     8
```

9)  How many elements would be in the vectors created by the following expressions?

```
linspace(3,2000)

     100 (always, by default)

logspace(3,2000)

     50 (always, by default - although these numbers
     would get very large quickly; most would be
          represented as Inf)
```

10) Create a variable *myend* which stores a random integer in the inclusive range from 5 to 9. Using the colon operator, create a vector that iterates from 1 to *myend* in steps of 3.

```
>>myend = randi([5, 9])
myend =
     8
>> vec = 1:3:myend
vec =
     1     4     7
```

11) Create two row vector variables. Concatenate them together to create a new row vector variable.

```
>> rowa = 2:4
rowa =
     2     3     4
>> rowb = 5:2:10
rowb =
     5     7     9
>> newrow = [rowa  rowb]
newrow =
     2     3     4     5     7     9
>>
```

12) Using the colon operator and the transpose operator, create a column vector *myvec* that has the values -1 to 1 in steps of 0.5.

```
>> rowVec = -1: 0.5: 1;
>> rowVec'
ans =
   -1.0000
   -0.5000
         0
    0.5000
    1.0000
```

13)Explain why the following expression results in a row vector, not a column vector:

```
colvec = 1:3'
```

```
Only the 3 is transposed; need to put in [] to get a column
vector
```

14) Write an expression that refers to only the elements that have odd-numbered subscripts in a vector, regardless of the length of the vector. Test your expression on vectors that have both an odd and even number of elements.

```
>> vec = 1:8;
>> vec(1:2:end)
ans =
     1     3     5     7

>> vec = 4:12
vec =
     4     5     6     7     8     9    10    11    12
>> vec(1:2:end)
ans =
     4     6     8    10    12
```

15) Generate a 2 x 4 matrix variable *mat*. Replace the first row with 1:4. Replace the third column (you decide with which values).

```
>> mat = [2:5; 1 4 11 3]
mat =
     2     3     4     5
     1     4    11     3
>> mat(1,:) = 1:4
mat =
     1     2     3     4
     1     4    11     3
>> mat(:,3) = [4;3]
mat =
     1     2     4     4
     1     4     3     3
```

16) Generate a 2 x 4 matrix variable *mat*. Verify that the number of elements is equal to the product of the number of rows and columns.

```
>> mat = randi(20,2,4)
mat =
     1    19    17     9
```

```
    13    15    20    16
>> [r c] = size(mat);
>> numel(mat) == r * c
ans =
     1
```

17) Which would you normally use for a matrix: **length** or **size**? Why?

> Definitely **size**, because it tells you both the number of rows and columns.

18) When would you use **length** vs. **size** for a vector?

> If you want to know the number of elements, you'd use length. If you want to figure out whether it's a row or column vector, you'd use size.

19) Generate a *2 x 3* matrix of random
- real numbers, each in the range (0, 1)
- real numbers, each in the range (0, 5)
- integers, each in the inclusive range from 10 to 50

```
>> rand(2,3)
ans =
    0.5208    0.5251    0.1665
    0.1182    0.1673    0.2944

>> rand(2,3)*5
ans =
    1.9468    2.3153    4.6954
    0.8526    2.9769    3.2779

>> randi([10, 50], 2, 3)
ans =
    16    20    39
    12    17    27
```

20) Create a variable *rows* that is a random integer in the inclusive range from 1 to 5. Create a variable *cols* that is a random integer in the inclusive range from 1 to 5. Create a matrix of all zeros with the dimensions given by the values of *rows* and *cols*.

```
>> rows = randi([1,5])
rows =
     3
>> cols = randi([1,5])
cols =
     2
```

```
>> zeros(rows,cols)
ans =
     0     0
     0     0
     0     0
```

21) Create a vector variable *vec*. Find as many expressions as you can that would refer to the last element in the vector, without assuming that you know how many elements it has (i.e., make your expressions general).

```
>> vec = 1:2:9
vec =
     1     3     5     7     9
>> vec(end)
ans =
     9
>> vec(numel(vec))
ans =
     9
>> vec(length(vec))
ans =
     9
>> v = fliplr(vec);
>> v(1)
ans =
     9
```

22) Create a matrix variable *mat*. Find as many expressions as you can that would refer to the last element in the matrix, without assuming that you know how many elements or rows or columns it has (i.e., make your expressions general).

```
>> mat = [12:15; 6:-1:3]
mat =
    12    13    14    15
     6     5     4     3
>> mat(end,end)
ans =
     3
>> mat(end)
ans =
     3
>> [r c] = size(mat);
>> mat(r,c)
ans =
     3
```

23) Create a 2 x 3 matrix variable *mat*. Pass this matrix variable to each of the following functions and make sure you understand the result: **flip**, **fliplr**, **flipud**, and **rot90**. In how many different ways can you **reshape** it?

```
>> mat = randi([1,20], 2,3)
mat =
    16     5     8
    15    18     1
>> flip(mat)
ans =
    15    18     1
    16     5     8
>>fliplr(mat)
ans =
     8     5    16
     1    18    15
>> flipud(mat)
ans =
    15    18     1
    16     5     8
>> rot90(mat)
ans =
     8     1
     5    18
    16    15
>> rot90(rot90(mat))
ans =
     1    18    15
     8     5    16
>> reshape(mat,3,2)
ans =
    16    18
    15     8
     5     1
>> reshape(mat,1,6)
ans =
    16    15     5    18     8     1
>> reshape(mat,6,1)
ans =
    16
    15
     5
    18
     8
     1
```

24) What is the difference between `fliplr(mat)` and `mat = fliplr(mat)`?

The first stores the result in *ans* so *mat* is not changed; the second changes *mat*.

25) Fill in the following:

The function **flip** is equivalent to the function __**fliplr**____ for a row vector.
The function **flip** is equivalent to the function __**flipud**____ for a column vector.
The function **flip** is equivalent to the function __**flipud**____ for a matrix.

26) Use **reshape** to reshape the row vector 1:4 into a 2x2 matrix; store this in a variable named *mat*. Next, make 2x3 copies of *mat* using both **repelem** and **repmat**.

```
>> mat = reshape(1:4,2,2)
mat =
     1     3
     2     4
>> repelem(mat,2,3)
ans =
     1     1     1     3     3     3
     1     1     1     3     3     3
     2     2     2     4     4     4
     2     2     2     4     4     4
>> repmat(mat,2,3)
ans =
     1     3     1     3     1     3
     2     4     2     4     2     4
     1     3     1     3     1     3
     2     4     2     4     2     4
```

27) Create a *3 x 5* matrix of random real numbers. Delete the third row.

```
>> mat = rand(3,5)
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
    0.1730    0.2523    0.1365    0.1991    0.2844

>> mat(3,:) = []
mat =
    0.5226    0.9797    0.8757    0.0118    0.2987
    0.8801    0.2714    0.7373    0.8939    0.6614
```

28) Given the matrix:

```
>> mat = randi([1 20], 3,5)
mat =
     6    17     7    13    17
    17     5     4    10    12
```

```
     6    19     6     8    11
```

Why wouldn't this work:

```
mat(2:3, 1:3) = ones(2)
```

```
Because the left and right sides are not the same dimensions.
```

29) Create a three-dimensional matrix with dimensions 2 x 4 x 3 in which the first "layer" is all 0s, the second is all 1s and the third is all 5s. Use **size** to verify the dimensions.

```
>> mat3d = zeros(2,4,3);
>> mat3d(:,:,2) = 1;
>> mat3d(:,:,3) = 5;
>> mat3d
mat3d(:,:,1) =
     0     0     0     0
     0     0     0     0
mat3d(:,:,2) =
     1     1     1     1
     1     1     1     1
mat3d(:,:,3) =
     5     5     5     5
     5     5     5     5
>> size(mat3d)
ans =
     2     4     3
```

30) Create a vector x which consists of 20 equally spaced points in the range from $-\pi$ to $+\pi$. Create a y vector which is **sin(x)**.

```
>> x = linspace(-pi,pi,20);
>> y = sin(x);
```

31) Create a *3 x 5* matrix of random integers, each in the inclusive range from -5 to 5. Get the **sign** of every element.

```
>> mat = randi([-5,5], 3,5)
mat =
     5     4     1    -1    -5
     4     4    -1    -3     0
     5    -2     1     0     4
>> sign(mat)
ans =
     1     1     1    -1    -1
     1     1    -1    -1     0
```

```
     1    -1     1     0     1
```

32)  Find the sum 2+4+6+8+10 using **sum** and the colon operator.

```
>> sum(2:2:10)
ans =
    30
```

33) Find the sum of the first n terms of the harmonic series where n is an integer variable greater than one.

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\frac{1}{5}+\cdots$$

```
>> n = 4;
>> sum(1./(1:n))
ans =
    2.0833
```

34) Find the following sum by first creating vectors for the numerators and denominators:

$$\frac{3}{1}+\frac{5}{2}+\frac{7}{3}+\frac{9}{4}$$

```
>> num = 3:2:9
num =
     3     5     7     9
>> denom = 1:4
denom =
     1     2     3     4
>> fracs = num ./ denom
fracs =
    3.0000    2.5000    2.3333    2.2500
>> sum(fracs)
ans =
   10.0833
```

35) Create a matrix and find the product of each row and column using **prod**.

```
>> mat = randi([1, 30], 2,3)
mat =
    11    24    16
     5    10     5

>> prod(mat)
```

```
ans =
    55   240    80

>> prod(mat,2)
ans =
        4224
         250
```

36) Create a *1 x 6* vector of random integers, each in the inclusive range from 1 to 20. Use built-in functions to find the minimum and maximum values in the vector. Also create a vector of cumulative sums using **cumsum**.

```
>> vec = randi([1,20], 1,6)
vec =
    12    20    10    17    15    10
>> min(vec)
ans =
    10
>> max(vec)
ans =
    20
>> cvec = cumsum(vec)
cvec =
    12    32    42    59    74    84
```

37) Write a relational expression for a vector variable that will verify that the last value in a vector created by **cumsum** is the same as the result returned by **sum**.

```
>> vec = 2:3:17
vec =
     2     5     8    11    14    17
>> cv = cumsum(vec)
cv =
     2     7    15    26    40    57
>> sum(vec) == cv(end)
ans =
     1
```

38) Create a vector of five random integers, each in the inclusive range from -10 to 10. Perform each of the following:

- subtract 3 from each element
- count how many are positive
- get the cumulative minimum

```
>> vec = randi([-10, 10], 1,5)
vec =
```

```
     1     8     3    -7     7
>> vec - 3
ans =
    -2     5     0   -10     4
>> sum(vec>0)
ans =
     4
>> cummin(vec)
ans =
     1     1     1    -7    -7
```

39) Create a *3 x 5* matrix.  Perform each of the following:

- Find the maximum value in each column.
- Find the maximum value in each row.
- Find the maximum value in the entire matrix.
- Find the cumulative maxima.

```
>> mat = randi([-10 10], 3,5)
mat =
     1    -5     0    -2    10
     2     1     1     6    -3
    -6    10    -3     5     2
>> max(mat)
ans =
     2    10     1     6    10
>> max(mat, [], 2)
ans =
    10
     6
    10
>> max(mat')
ans =
    10     6    10
>> max(max(mat))
ans =
    10
>> cummax(mat)
ans =
     1    -5     0    -2    10
     2     1     1     6    10
     2    10     1     6    10
```

40)  Find two ways to create a *3 x 5* matrix of all 100s (Hint: use **ones** and **zeros**).

```
ones(3,5)*100
```

```
zeros(3,5)+100
```

41) Create variables for these two matrices:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & -1 & 6 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & 0 \end{bmatrix}$$

Perform the following operations:

```
A + B
```

$$\begin{bmatrix} 3 & 6 & 4 \\ 5 & 2 & 6 \end{bmatrix}$$

```
A - B
```

$$\begin{bmatrix} -1 & -2 & 2 \\ 3 & -4 & 6 \end{bmatrix}$$

```
A .* B
```

$$\begin{bmatrix} 2 & 8 & 3 \\ 4 & -3 & 0 \end{bmatrix}$$

42) A vector *v* stores for several employees of the Green Fuel Cells Corporation their hours worked one week followed for each by the hourly pay rate. For example, if the variable stores

```
>> v
v =
33.0000  10.5000  40.0000  18.0000  20.0000  7.5000
```

that means the first employee worked 33 hours at $10.50 per hour, the second worked 40 hours at $18 an hour, and so on. Write code that will separate this into two vectors, one that stores the hours worked and another that stores the hourly rates. Then, use the array multiplication operator to create a vector, storing in the new vector the total pay for every employee.

```
>> hours = v(1:2:length(v))
hours =
    33    40    20

>> payrate = v(2:2:length(v))
payrate =
   10.5000   18.0000    7.5000

>> totpay = hours .* payrate
totpay =
  346.5000  720.0000  150.0000
```

43)  Write code that would count how many elements in a matrix variable *mat* are negative numbers.  Create a matrix of random numbers, some positive and some negative, first.

```
>> mat
mat =
     1    -5     0    -2    10
     2     1     1     6    -3
    -6    10    -3     5     2
>> sum(sum(mat < 0))
ans =
     5
```

44)  A company is calibrating some measuring instrumentation and has measured the radius and height of one cylinder 8 separate times; they are in vector variables *r* and *h*. Find the volume from each trial, which is given by $\Pi r^2 h$.  Also use logical indexing first to make sure that all measurements were valid (> 0).

```
>> r = [5.499 5.498 5.5 5.5 5.52 5.51 5.5 5.48];
>> h = [11.1 11.12 11.09 11.11 11.11 11.1 11.08 11.11];

>> all(r>0 & h>0)
ans =
     1
>> vol = pi * r.^2 .* h
```

45)  For the following matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad B = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \quad C = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

- Give the result of 3*A.

$$\begin{bmatrix} 3 & 12 \\ 9 & 6 \end{bmatrix}$$

- Give the result of A*C.

$$\begin{bmatrix} 19 & 6 & 13 \\ 17 & 8 & 19 \end{bmatrix}$$

- Are there any other matrix multiplications that can be performed?  If so, list them.

C*B

46) Create a row vector variable *r* that has 4 elements, and a column vector variable *c* that has 4 elements. Perform r*c and c*r.

```
>> r = randi([1 10], 1, 4)
r =
     3     8     2     9
>> c = randi([1 10], 4, 1)
c =
     4
     9
     7
     8
>> r*c
ans =
   170
>> c*r
ans =
    12    32     8    36
    27    72    18    81
    21    56    14    63
    24    64    16    72
```

47) The matrix variable *rainmat* stores the total rainfall in inches for some districts for the years 2014-2017. Each row has the rainfall amounts for a given district. For example, if *rainmat* has the value:

```
>> rainmat
ans =
  25   33   29   42
  53   44   40   56
     etc.
```

district 1 had 25 inches in 2014, 33 in 2015, etc. Write expression(s) that will find the number of the district that had the highest total rainfall for the entire four year period.

```
>> rainmat = [25 33 29 42; 53 44 40 56];
>> large = max(max(rainmat))
large =
    56
>> linind = find(rainmat== large)
linind =
     8
>> floor(linind/4)
ans =
```

```
     2
```

48) Generate a vector of 20 random integers, each in the range from 50 to 100. Create a variable *evens* that stores all of the even numbers from the vector, and a variable *odds* that stores the odd numbers.

```
>> nums = randi([50, 100], 1, 20);
>> evens = nums(rem(nums,2)==0);
>> odds = nums(rem(nums,2)~=0);
```

49) Assume that the function **diff** does not exist. Write your own expression(s) to accomplish the same thing for a vector.

```
>> vec = [5 11 2 33 -4]
vec =
     5    11     2    33    -4
>> v1 = vec(2:end);
>> v2 = vec(1:end-1);
>> v1-v2
ans =
     6    -9    31   -37
```

50) Create a vector variable *vec*; it can have any length. Then, write assignment statements that would store the first half of the vector in one variable and the second half in another. Make sure that your assignment statements are general, and work whether *vec* has an even or odd number of elements (Hint: use a rounding function such as **fix**).

```
>> vec = 1:9;
>> fhalf = vec(1:fix(length(vec)/2))
fhalf =
     1     2     3     4
>> shalf = vec(fix(length(vec)/2)+1:end)
shalf =
     5     6     7     8     9
```