Class 24 Group Practice Problem Solutions

Instructions: Type your answers to questions here. If you write code, cut and paste it from MATLAB into this document.

Name:

Partner Names:

1) Explain the difference between these two statements:

result = 9\*2

result = 9\*2;

The first statement shows the value of result; the second suppresses the output.

2) Explain the difference between these:

numb = 5 – 4 \* 2

5 – 4 \* 2

The result is stored in the variable numb in the first, and in the default variable ans in the second.

3) Explain the result of each of the following expressions:

4 == 6 + 1

4 is not equal to 7, so the result is logical 0

'a' < 'b'

The character ‘a’ is less than the character ‘b’ (because it comes before it in the encoding sequence), so the result is logical 1

10 > 5 > 2

This is evaluated from left to right. First, 10>5 is true so the result of that is logical 1. Then 1 > 2 is false so the result overall is logical 0

result = 3^2 - 20;

0 <= result <= 10

The variable result stores -11. 0 <= -11 is false so that is 0. 0 <= 10 is true so the answer is logical 1

xor('c' == 'd'-1, 5 > 3)

The xor function returns true if and only if just one of the expressions is true. In this case, both expressions are true so xor returns logical 0.

4) For a random integer in a variable *num*, e.g.

>> num = randi([1, 10]);

All of the following will test to see whether the value of *num* is less than 5:

>> (num < 5) == true

>> (num < 5) == logical(1)

>> (num < 5) == 1

However, the following expression is simpler. Why is that?

num < 5

The result of this expression will be true or false. Testing to see if that is

equal to true is redundant.

5) Create the following vectors: r = 1:3; c = [2:4]';

Multiply them. OK, so what does that mean? Can you perform array multiplication? Can you perform matrix multiplication? Does the order matter?

Array multiplication cannot be performed because the vectors do not have the same dimensions. The row vector r is 1 x 3. The column vector c is 3 x 1.

However, we can perform matrix multiplication. We can do r\*c because the inner dimensions are the same (3); the result will be the outer dimensions (1 x 1).

>> r\*c

20

We can also do c\*r because the inner dimensions are the same (1); the result will be the outer dimensions (3 x 3).

>> c\*r

2 4 6

3 6 9

4 8 12

6) Create the following vector using the colon operator:

vec =

4 3 2 1 0 -1 -2

vec = 4:-1:-2

7) Create the following vector using **linspace**:

v =

1.0000 1.5000 2.0000 2.5000 3.0000

V = linspace(1,3,5)

8) 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 =

4

>> cols = randi([1, 5])

cols =

2

>> zmat = zeros(rows,cols)

zmat =

0 0

0 0

0 0

0 0

9) Create a 2 x 5 x 3 matrix of all ones.

ones(2,5,3)

10) Write an expression that will sum only the values in a vector variable “v” that are greater than or equal to 4.

sum(v(v>=4))

11) Write an expression that will sum only the values in a matrix variable “mat” that are greater than or equal to 4.

sum(mat(mat>=4))

12) Create a row vector variable that has 5 random integers each in the range from 2 to 12. Add 3 to every element and store the result back in the same variable. Divide every element by 4.

>> rv = randi([2,12], 1, 5)

rv =

6 8 10 2 12

>> rv = rv + 3

rv =

9 11 13 5 15

>> rv/4

ans =

2.2500 2.7500 3.2500 1.2500 3.7500

13) Create a 3 x 4 matrix of random integers, each in the range from 0 to 30. Subtract 10 from every element and store the result back in the same variable. Get the absolute value of every element and store in a new variable. Explore the **sign** function with your original matrix variable. Explain what it does.

>> mat = randi([0, 30], 3, 4)

mat =

16 27 18 14

10 17 6 7

29 19 9 26

>> mat = mat - 10

mat =

6 17 8 4

0 7 -4 -3

19 9 -1 16

>> newm = abs(mat)

newm =

6 17 8 4

0 7 4 3

19 9 1 16

>> sign(mat)

ans =

1 1 1 1

0 1 -1 -1

1 1 -1 1

The sign function returns 1 for positive integers, -1 for negative, and 0 for 0.

14) Create two vector variables veca and vecb, that have the same number of elements, using the colon operator with step values. Add corresponding elements. Multiply corresponding elements.

>> veca = 2:2:8

veca =

2 4 6 8

>> vecb = 1:4:15

vecb =

1 5 9 13

>> veca + vecb

ans =

3 9 15 21

>> veca .\* vecb

ans =

2 20 54 104

15) Create two matrix variables mat1 and mat2 that have the same dimensions. Perform the operation mat1 – mat2.

>> mat1 = [1:4; 2:5; 8 9 3 6]

mat1 =

1 2 3 4

2 3 4 5

8 9 3 6

>> mat2 = mat1 \* 2

mat2 =

2 4 6 8

4 6 8 10

16 18 6 12

>> mat1 - mat2

ans =

-1 -2 -3 -4

-2 -3 -4 -5

-8 -9 -3 -6

16) Reshape your matrix mat1 in as many ways as you can.

>> reshape(mat1, 4, 3)

ans =

1 3 3

2 9 4

8 3 5

2 4 6

>> reshape(mat1, 2, 6);

>> reshape(mat1, 6, 2);

>> reshape(mat1, 1, 12);

>> reshape(mat1, 12,1);

17) Rotate mat2 counterclockwise 180 degrees.

>> mat2

mat2 =

2 4 6 8

4 6 8 10

16 18 6 12

>> rot90(rot90(mat2))

ans =

12 6 18 16

10 8 6 4

8 6 4 2

18) Find the mean of every row of mat1.

>> mat1

mat1 =

1 2 3 4

2 3 4 5

8 9 3 6

>> mean(mat1')

ans =

2.5000 3.5000 6.5000

19) Create a 4 x 4 square matrix. Store the diagonal in a variable. Get the trace of the matrix.

>> mat44 = randi([1, 10], 4, 4)

mat44 =

2 5 2 2

3 4 10 3

2 10 10 5

3 5 5 6

>> diag44 = diag(mat44)

diag44 =

2

4

10

6

>> trace(mat44)

ans =

22

>> sum(diag44)

ans =

22

20) Multiply your 4 x 4 matrix by an identity matrix with appropriate dimensions. What is the result?

>> mat44

mat44 =

2 5 2 2

3 4 10 3

2 10 10 5

3 5 5 6

>> mat44 \* eye(4)

ans =

2 5 2 2

3 4 10 3

2 10 10 5

3 5 5 6

The result is the original matrix

21) Create the following matrix variable:

mat =

1 3 5

2 6 3

Write code that will:

* Find the maximum of each column in the matrix.
* Find the product of numbers in each column of the matrix.
* Find the overall sum of the numbers in the matrix.

>> mat = [1 3 5; 2 6 3]

mat =

1 3 5

2 6 3

>> max(mat)

ans =

2 6 5

>> prod(mat)

ans =

2 18 15

>> sum(sum(mat))

ans =

20

22) Create a row vector variable and pass it to the **diff** function. Then, assume that the diff function does not exist, and create the same result by creating two vectors from the original and subtracting one from the other. Your code must be general, and work for a vector of any length.

>> vec = [5, 2:4, 1, 9]

vec =

5 2 3 4 1 9

>> diff(vec)

ans =

-3 1 1 -3 8

>> vec1 = vec(2:end)

vec1 =

2 3 4 1 9

>> vec2 = vec(1:end-1)

vec2 =

5 2 3 4 1

>> vec1-vec2

ans =

-3 1 1 -3 8

Time to get creative! Put your best creative problems in the GPP Creative chat for Class 24.