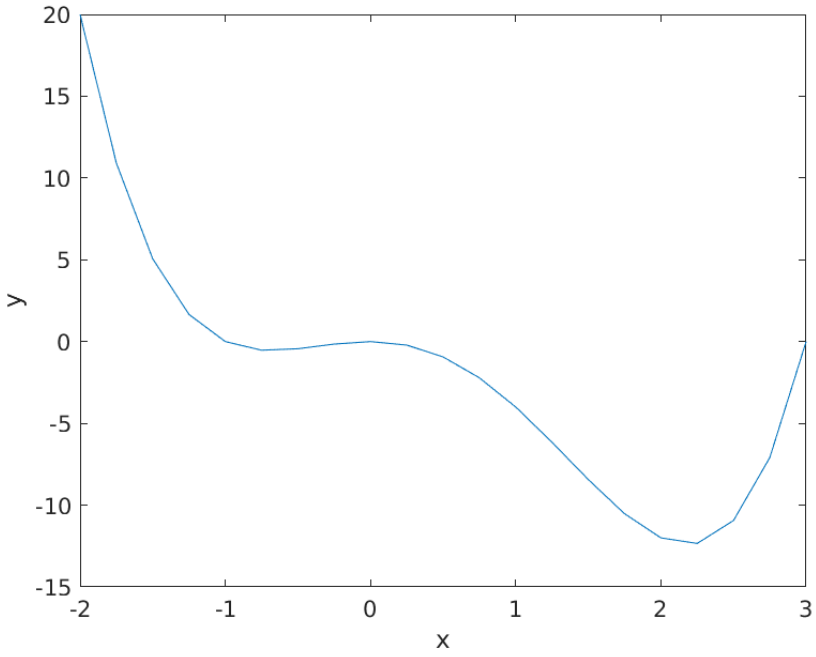
1. (a) Create the vector `x` having values $x = -2, -1.75, -1.5, ..., 2.75, 3.0$ (i.e., x ranges from -2 to 3 and has values spaced 0.25 units apart).

   (b) Create a vector `y` equal to $x^4 - 2x^3 - 3x^2$ where `x` is your vector from part (a).

   (c) Plot `y` versus `x` to reproduce the plot shown below:



2. Exponential decaying sinusoids occur frequently in the analysis of waves and vibrations. An example of an exponential decaying sinusoid is

$$y = Ae^{-ax}\sin(bx)$$

where A, a, and b are scalar constants.

(a) Create a variable `A` having the value 2, a variable `a` having the value 0.246, and a variable `b` having the value 0.806.

(b) Create a vector `x` having 75 equally spaced values between 0 and 20 (inclusive). The values in `x` represent angles in radians.

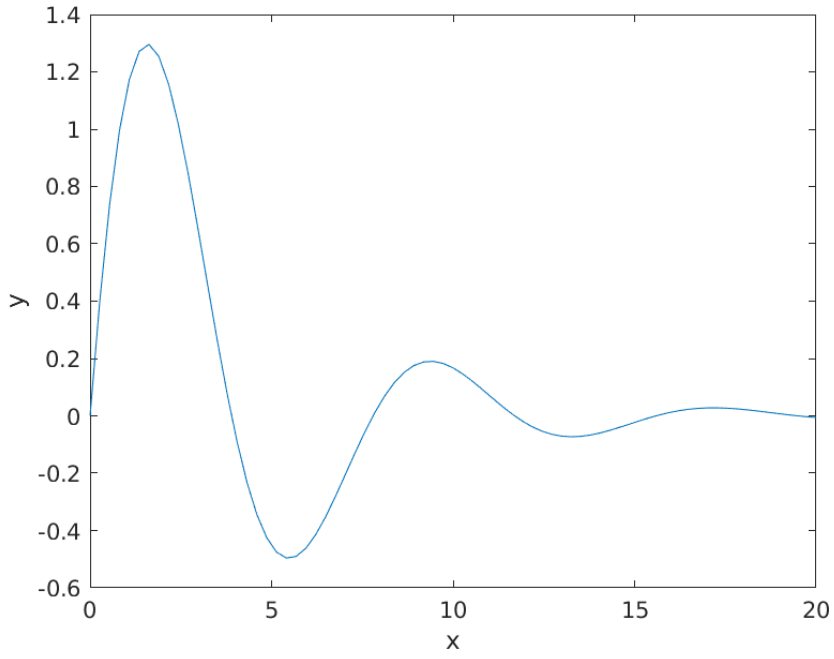
(c) Create a vector `y` equal to

$$y = Ae^{-ax}\sin(bx)$$

where A, a, and b are your variables from part 2(a) and x are the values in your vector from 2(b). You should the variables `A`, `a`, and `b` rather than the numeric values from part 2(a).

Use the MATLAB function `exp` to compute e raised to an exponent.

(d) Plot `y` versus `x` to reproduce the plot shown below:



3. The sinc function is an important function that appears in digital signal processing. It is defined by:

$$y = \frac{\sin(\pi x)}{\pi x}$$

where x is a real scalar value.

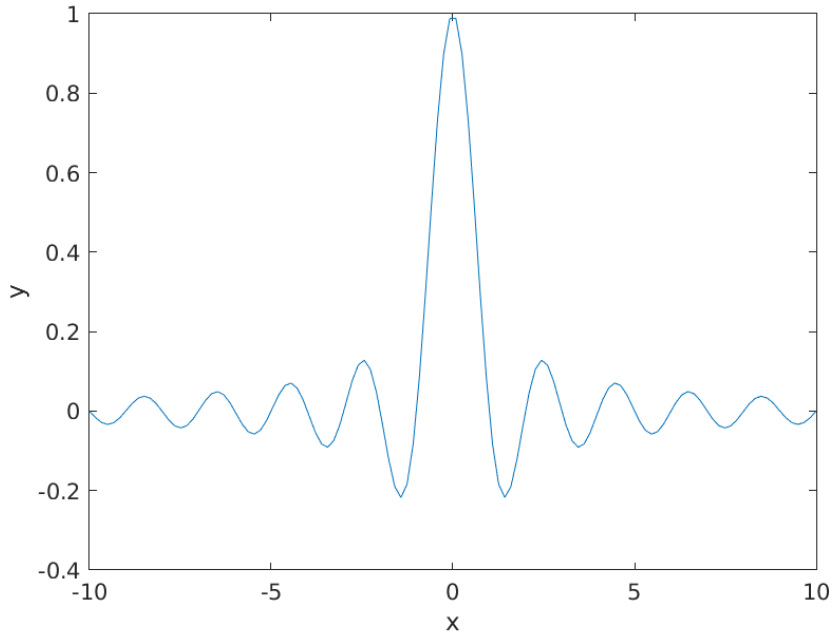
(a) Create a vector `x` having 120 equally spaced values between -10 and 10 (inclusive). The values in `x` represent angles in radians.

(b) Create a vector `y` equal to

$$y = \frac{\sin(\pi x)}{\pi x}$$

where x are the values in your vector from 2(b).

(c) Plot `y` versus `x` to reproduce the plot shown below:



4. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if` statement that sets `y` to 1 if `x` is less than or equal to 5.

5. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if` statement that sets `y` to 1 if `x` is between the values 0 and 10 (including the values 0 and 10).

6. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if` statement that sets `y` to 1 if `x` is outside the range of values 0 and 10 (not including the values 0 and 10).

7. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if` statement that sets `y` to 1 if `x` is between the values 0 and 10 (excluding the values 0 and 10) or between the values 20 and 30 (excluding the values 20 and 30).

8. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if` statement that sets `y` to 1 if `x` is evenly divisible by 3 or 5.

9. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

   Write an `if-elseif` statement that sets `y` to 1 if `x` is outside the range of values 0 and 10 (not including the values 0 and 10) and sets `y` to 2 if `x` is equal to 0 and sets `y` to 3 if `x` is equal to 10.

10. Suppose that you have a numeric variable named `x`. You have a second variable named `y` whose value is equal to 0.

    Write an `if` statement that sets `y` to 1 if `x` is evenly divisible by 3, sets `y` to 2 if `x` is evenly divisible by 5, and sets `y` to 3 if `x` is evenly divisible by 15 (this is a variation of a well-known programming interview question called the fizzbuzz problem).

11. Suppose that you have the following `if` statement:

```
if ~(x < 0 || x > 5)
        % something
end
```

Rewrite the `if` statement replacing the OR (`||`) with logical AND (`&&`) so that the meaning of the `if` statement is unchanged.

12. Consider a circle having radius 1 and centered on the point $(0, 0)$. Suppose you have a point with coordinates (x, y). Write an `if` statement (or an `if-else` statement) that sets the value of a variable `tf` to true if (x, y) is inside the circle or false if (x, y) is outside the circle or on the perimeter of the circle.

13. Consider a circle having radius R and centered on the point (x_c, y_c). Suppose you have a point with coordinates (x, y). Write an `if` statement (or an `if-else` statement) that sets the value of a variable `tf` to true if (x, y) is inside the circle and sets `tf` to false if (x, y) is outside the circle or on the perimeter of the circle.

14. Suppose that you expect to measure a quantity that lies in the range -3 to 3. When you measure the quantity many times, you sometimes get a value that lies outside the range -3 to 3; such a measurement is called an outlier (because it is outside of expected range of values). You want to keep count of the number of outliers using a variable named `nOutlier`.

    Suppose that you have variables `nOutlier`, `nUnusual`, and `meas` having some valid but unknown values.

    (a) Write an `if` statement that increases the current value of `nOutlier` by 1 if the measurement stored in a variable named `meas` is outside the range of -3 to 3

    It does not matter how you treat values exactly equal to -3 or 3.

    (b) Suppose that you purchase better measurement equipment so that you now expect measurements to lie in the range -2 to 2. Values outside the range -3 to 3 are still considered outliers; however, values between -2 to -3, as well as values between 2 to 3 are considered unusual (that is, they might be outliers). You want to keep count of the number of outliers using a variable named `nOutlier` and the number of unusual values using a variable named `nUnusual`

    Write an `if-elseif` statement that increases the current value of `nOutlier` by 1 if the measurement stored in a variable named `meas` is outside the range of -3 to 3, and increases the current value of `nUnusual` by 1 if the measurement stored in a variable named `meas` is inside the ranges of -2 to -3 or 2 to 3.

It does not matter how you treat values exactly equal to -3, -2, 2, or 3.

15. The statement

```
x = 1:10;
```

creates a row vector having the integer values 1, 2, ..., 10.

Suppose that you wanted to write a loop to create the vector `x` instead of using the `:` operator. Starting with the following vector of zeros:

```
x = zeros(1, 10);
```

write a loop that sets the elements of `x` to the values 1 through 10.

16. The statements

```
x = 100:111;
y = reshape(x, [4 3]);
```

creates a matrix equal to:

$$y = \begin{bmatrix} 100 & 104 & 108 \\ 101 & 105 & 109 \\ 102 & 106 & 110 \\ 103 & 107 & 111 \end{bmatrix}$$

Suppose that you wanted to write a pair of nested loops to create the matrix `y` instead of using the code shown above. Starting with the following matrix of zeros:

```
y = zeros(4, 3);
```

write a pair of nested loops that sets the elements of `y` to the appropriate values.

17. Suppose that you have the following matrix:

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

The statement

```
z = y(:);
```

creates a column vector `z` where the elements of `z` are taken from the columns of `y`; i.e.,

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

Starting with the matrix y shown above, write a pair of nested loops that iterate over the elements of y to create the vector z shown above.

18. The root mean square (RMS) value is often computed when analyzing vibrations and waves. The formula for the RMS value of a set of n values z_1, z_2, ..., z_n is

$$z_{\text{RMS}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} z_i^2}$$

Create an n-dimensional vector z and assign its elements values of your choice. Without using any MATLAB functions (in particular `sum` and `mean`) except for `sqrt` and functions used to find the number of elements in a vector, write a loop that computes the RMS values of the elements in the vector `z`. Do this by first showing how to obtain the number of elements in the vector by calling an appropriate function; then write the loop to compute the RMS value.

19. Many measured quantities in engineering are rates and ratios. To compute the average value of such quantities, the harmonic mean is often used. The formula for the harmonic mean of a set of n values z_1, z_2, ..., z_n is

$$z_{\text{h}} = \frac{n}{\sum_{i=1}^{n} \frac{1}{z_i}}$$

Create an n-dimensional vector z and assign its elements values of your choice. Without using any MATLAB functions (in particular `sum` and `mean`) except for functions used to find the number of elements in a vector, write a loop that computes the harmonic mean of the elements in the vector `z`. Do this by first showing how to obtain the number of elements in the vector by calling an appropriate function; then write the loop to compute the harmonic mean.

20. Suppose that you have n resistors r_1, r_2, ..., r_n connected in parallel. The total resistance of the n resistors is given by

$$r_{\text{total}} = \frac{1}{\sum_{i=1}^{n} \frac{1}{r_i}}$$

Create an n-dimensional vector r and assign its elements values of your choice. Without using any MATLAB functions (in particular `sum`) except for functions used to find the number of elements in a vector, write a loop that computes the total resistance of the resistor values in the vector `r`. Do this by first showing how to obtain the number of elements in the vector by calling an appropriate function; then write the loop to compute the total resistance.

21. The statement

```
y = randi(100, [1 20]);
```

creates a row vector `y` having 20 random integer values between 1 and 100.

Write a loop that finds the largest value in `y`.

22. The statement

```
y = randi(100, [1 20]);
```

creates a row vector `y` having 20 random integer values between 1 and 100.

Write a loop that finds the smallest value in `y`.

23. The statement

```
y = randi(10, [1 20]);
```

creates a row vector `y` having 20 random integer values between 1 and 10.

Write a loop that finds the *index* of the largest value in `y`. If the largest value appears more than once in `y` then the returned index is the index of the *last* occurrence of the largest value.

24. The statement

```
y = randi(10, [1 20]);
```

creates a row vector `y` having 20 random integer values between 1 and 10.

Write a loop that finds the *index* of the largest value in `y`. If the largest value appears more than once in `y` then the returned index is the index of the *first* occurrence of the largest value.

25. Many science and engineering applications record the time of day using the number of seconds after midnight; in such applications, it is often convenient to convert the number of seconds after midnight to the time on the widely used 12-hour clock format (hours, minutes, seconds, and an AM/PM indicator). For example, 0 seconds after midnight is equal to 12:00:00 AM, 61 seconds after midnight is equal to 12:01:01 AM, $60 \times 60 \times 12$ seconds after midnight is equal to 12:00:00 PM, and $60 \times 60 \times 24 - 1$ seconds after midnight is equal to 11:59:59 PM.

    Implement the following function that converts a number of seconds after midnight to time on a 12-hour clock:

```
function [hr, min, sec, isAM] = fromSeconds(seconds)
%FROMSECONDS Seconds after midnight to 24-hour time
%  [hr, min, sec, isAM] = fromSeconds(seconds) returns
%  the 12-hour clock time hr:min:sec given a
%  number of seconds after midnight. isAM is equal
%  to true if the time is before noon and is equal
%  to false if the time is noon or after noon.
%
%  seconds is expected to be greater than 0 but may
%  be greater than the number of seconds in one day



end
```

    Example usage:

```
[hr, min, sec, isAM] = fromSeconds(60 * 60 * 24 - 1);

% hr should be 11
% min should be 59
% sec should be 59
% isAM should be false
```

26. Dates represented by numbers are ambiguous because there is no universal convention for writing them. For example, the date represented by the vector `[1 4]` could be January 4 (first digit is the month and second digit is the day) or it could be April 1 (first digit is the day and second digit is the month).

    Complete the following MATLAB function (you can edit the function that is provided to you):

```
function tf = similarDate(date1, date2)
%SIMILARDATE Are two dates similar?
%   tf = similarDate(date1, date2) returns true if
%   the two dates date1 and date2 are similar.
%   date1 and date2 are both vectors of length 2
%   where the elements represent a numeric date
%   (either [day, month] or [month, day]).
%
%   Two dates are similar if they can be interpreted
%   as being the same date.
```

    You should not use any MATLAB functions in your solution.

    Example usage:

```
tf = similarDate([1 4], [1 4])     % should return true
tf = similarDate([2 8], [8 2])     % should return true
tf = similarDate([9 3], [3 21])    % should return false
```

27. A triangle is an isoceles triangle if two sides have the same length. An equilateral triangle is triangle where all three sides have the same length; equilateral triangles are usually also considered to be isoceles triangles, but sometimes it is important to distinguish between the two.

    Complete the following MATLAB function:

```
%ISISOCELES Is a triangle an isoceles triangle?
% tf = isIsoceles(a, b, c) returns true if the triangle
% with side lengths a, b, and c is strictly an
% isocelese triangle (exactly two sides have equal
% length) and false otherwise.
```

    Example usage:

```
tf = isIsoceles(1, 2, 3)         % false
tf = isIsoceles(1, 3, 3)         % true
tf = isIsoceles(1, 5, 1)         % true
tf = isIsoceles(2.5, 2.5, 2.5)   % false
```

28. In MATLAB a scalar is a 2-dimensional array having size 1×1.

    In MATLAB a vector is a 2-dimensional array having either 1 row or 1 column.

    In MATLAB a matrix is a 2-dimensional array that is not a scalar nor a vector.

    Complete the following MATLAB function:

```
function w = whatArray(x)
%WHATARRAY Determines the kind of 2-dimensional array
%   w = whatArray(x) returns:
%
%   1 if x is a scalar
%   2 if x is a vector
%   3 if x is a matrix
%
%   You can assume that x is a 2-dimensional array
%   that is not empty.
```

    The only MATLAB function that you may use is `size`.

    Example usage:

```
whatArray(100)          % should return 1
whatArray(1:5)          % should return 2
whatArray([1:5]')       % should return 2
whatArray(eye(3))       % should return 3
```

29. Complete the following MATLAB function:

```
function n = numUnique(x, y, z)
%NUMUNIQUE Number of unique values
%   n = numUnique(x, y, z) returns the number of different
%   values in the scalars x, y, and z.
```

You should not use any MATLAB functions in your solution.

Example usage:

```
numUnique(1, 1, 1)          % should return 1
numUnique(5, 1, 1)          % should return 2
numUnique(0, 8, 10)         % should return 3
```

30. Write a function that computes the sum of the elements in a vector. Your function should not use any MATLAB functions that compute a sum; instead use a loop to iterate over the elements of the vector and accumulate the sum into a variable.

    You should choose your own function name, and decide what inputs and outputs are required.

31. Write a function that computes the sum of the *positive* elements in a vector (i.e., your function should not include negative elements of the vector in the computed sum). Your function should not use any MATLAB functions that compute a sum; instead use a loop to iterate over the elements of the vector and accumulate the sum into a variable.

    You should choose your own function name, and decide what inputs and outputs are required.

32. Write a function that returns both the largest value and the index of the largest value of a vector. Your function should not sort the input vector and it should not use the function `max`.

    You should choose your own function name, and decide what inputs and outputs are required.

33. Write a function that returns a vector containing a value repeated n times. For example

    ```
    n = 5;
    y = repeat(1.25, n);
    ```

    would return a vector `y` equal to `[1.25  1.25  1.25  1.25  1.25]`.

    Try to implement the function in three different ways. For the first way, use a loop to create the returned vector. For the second way, do not use a loop to create the returned vector; instead use the function `ones`. For the third way, do not use a loop to create the returned vector; instead use the function `repmat`.

34. Complete the following MATLAB function:

```
function tf = contains(value, x)
%CONTAINS Does a vector contain a specified value?
%   tf = contains(value, x) returns true if the vector x
%   has at least one element equal to value and returns false
%   otherwise.
```

You should not use any MATLAB functions in your solution that search for a value in an array (such as `find`).

Example usage:

```
contains(1, [0 0 0])          % should return false
contains(5, 3:10)             % should return true
contains(0, zeros(1, 5))      % should return true
```

35. Complete the following MATLAB function:

```
function idx = indexOf(value, x)
%INDEXOF Index of a specified value in a vector
%   idx = indexOf(value, x) returns the index of the first
%   element in x equal to value. The index 0 is returned
%   if there is no element in x equal to value.
```

You should not use any MATLAB functions in your solution that search for a value in an array (such as `find`).

Example usage:

```
i = indexOf(1, [0 0 0])          % should return 0
i = indexOf(5, 3:10)             % should return 3
i = indexOf(0, zeros(1, 5))      % should return 1
```

36. Complete the following MATLAB function:

```
function IDX = allIndexesOf(value, x)
%ALLINDEXESOF Indexes of a specified value in a vector
%   IDX = allIndexesOf(value, x) returns the vector of
%   indexes of all of the elements in x equal to
%   value. The index 0 is returned if there is no
%   element in x equal to value.
```

Note that this is similar to the MATLAB function `find`. You should not use any MATLAB functions in your solution that search for a value in an array.

Example usage:

```
I = allIndexesOf(1, [0 0 0])          % should return 0
I = allIndexesOf(5, 3:10)             % should return 3
I = allIndexesOf(0, zeros(1, 5))      % should return [1 2 3 4 5]
```

37. Given a matrix `A` the statement

```
y = sum(A);
```

will return a row vector containing the sum of each column of `A`; for example,

```
A = [1 2 3;  7 8 9];
y = sum(A);
```

computes `y` equal to `[8 10 12]`.

Write a function that computes the sum of the columns of a matrix. Your function should not use any MATLAB functions that compute a sum but you may use your solution to Question 30 if you wish.

You should choose your own function name, and decide what inputs and outputs are required.

38. Given a matrix `A` the statement

```
y = sum(A, 2);
```

will return a column vector containing the sum of each row of `A`; for example,

```
A = [1 2 3;  7 8 9];
y = sum(A);
```

computes `y` equal to `[6;  24]`.

Write a function that computes the sum of the rows of a matrix. Your function should not use any MATLAB functions that compute a sum but you may use your solution to Question 30 if you wish.

You should choose your own function name, and decide what inputs and outputs are required.

39. A square matrix is a matrix having the same number of rows and columns. A square matrix where all of the elements are equal to zero except possibly for the elements on the main diagonal is said to be a *diagonal* matrix. Write a function that returns true if the input matrix is a diagonal matrix, and returns false otherwise.

You should choose your own function name, and decide what inputs and outputs are required.

40. See the previous question. Write a function that returns a vector containing the elements of the diagonal elements of a square matrix (the matrix does not need to be a diagonal matrix). For example, if the input matrix is

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

then your function should return the vector `[8  5  2]`. Do not use the MATLAB function `diag` in your solution.

You should choose your own function name, and decide what inputs and outputs are required.

41. You normally use two indexes to get or set an element in a matrix; however, it is possible to use a single index instead (MATLAB refers to this as *linear indexing*).

In linear indexing, the elements of a matrix are numbered from top to bottom and then left to right. For example, the linear indexes of a 5×3 matrix are:

$$\begin{bmatrix} 1 & 6 & 11 \\ 2 & 7 & 12 \\ 3 & 8 & 13 \\ 4 & 9 & 14 \\ 5 & 10 & 15 \end{bmatrix}$$

In the above example, the element at row 2 and column 3 has the linear index 12.

Write a function that converts a row and column index for a matrix to a linear index. You do not require any loops to solve this problem; there is a simple formula for calculating the linear index.

You should choose your own function name, and decide what inputs and outputs are required. Note that one of the inputs is either the matrix that you want the linear index for, or the size of the matrix that you want the linear index for.

42. See the previous question. Write a function that converts a linear index for a matrix to a row and column index for the matrix.

You should choose your own function name, and decide what inputs and outputs are required. Note that one of the inputs is either the matrix that you want the linear index for, or the size of the matrix that you want the linear index for.

43. Suppose that you have two vectors `x` and `y` where the elements in `x` are unique (have no duplicates) and the elements in `y` are unique. The number of elements in `x` and `y` are not necessarily the same, but both vectors have at least 1 element.

    The intersection of `x` and `y` is the vector `z` made up of elements that are common to both `x` and `y`. For example, consider the vectors `x` and `y`:

```
x = [3  2  4];
y = [1  9  8   4   2];
```

    then the intersection of `x` and `y` is the vector

```
z = [2  4];     % z = [4  2] is also acceptable
```

    because both `x` and `y` contain the elements `2` and `4`.

    The intersection is the empty vector if `x` and `y` have no elements in common; the intersection of:

```
x = [3  2  4  8  9  10  12  15  18];
y = [100  -500  300];
```

    is the empty vector `[]`

    Complete the following MATLAB function:

```
function z = intersect(x, y)
%INTERSECT The set intersection of two vectors.
%   z = intersect(x, y) returns the vector z containing all
%   of the elements that are common to both x and y.
%
%   z is the empty vector if there is no element that
%   is in both x and y.
```

    You may use the MATLAB function `isempty` and any function that determines the size of a vector, but do not use any other MATLAB functions in your solution.

    Example usage:

```
z = intersect([3 2 4], [1 9 8 4 2])     % should return [2 4]
z = intersect([1 2 3 4], [5])           % should return []
```

    Hint: Start with an empty vector `z`. Loop over the elements of `x`. Inside that loop, loop over the elements of `y`. If you find an element in `y` that is equal to an element in `x` then concatenate that element to the vector `z`.

44. A common task in image processing is to find the location of a pattern in an image. Consider the following grayscale image `A`:

$$\texttt{A} = \begin{bmatrix} 17 & 24 & 1 & 8 & 15 \\ 23 & 5 & 7 & 14 & 16 \\ 4 & 6 & 13 & 24 & 22 \\ 17 & 24 & 1 & 5 & 3 \\ 23 & 5 & 7 & 2 & 9 \end{bmatrix}$$

Consider the following pattern `P`:

$$\texttt{P} = \begin{bmatrix} 17 & 24 & 1 \\ 23 & 5 & 7 \end{bmatrix}$$

`P` occurs in `A` two times.

Consider the following pattern `P`:

$$\texttt{P} = \begin{bmatrix} 24 \\ 5 \end{bmatrix}$$

`P` occurs in `A` 3 times.

The pattern $\texttt{P} = [100]$ does not appear in `A`.

If `P` is larger than `A` then `P` does not appear in `A`.

Complete the following MATLAB function:

```
function n = countPattern(P, A)
%COUNTPATTERN Count occurrences of a pattern in a matrix.
%   n = countPattern(P, I) returns the number of times
%   a pattern P appears in a matrix A.
```

Note that in the statement

```
tf = all(P == A)
```

the value of `tf` will be true if the matrix `P` is equal to the matrix `A` and `tf` will be false if `P` is not equal to `A`. This is very useful in this particular problem.

You may use the MATLAB function `all` and MATLAB functions that determine the size of a matrix, but do not use any other MATLAB functions in your solution.

Example usage:

```
A = [ 1  2  3  4;
      2  3  0  0;
      3  4  5  8];
P = [2; 3];
n = countPattern(P, A)       % should return 2
P = [2 3];
n = countPattern(P, A)       % should return 2
P = 100;
n = countPattern(P, A)       % should return 0
```

45. The game tic-tac-toe is usually played on a 3×3 grid but it can be played on any $n \times n$ grid where $n > 0$. Suppose that you have a square matrix that represents a completed $n \times n$ tic-tac-toe board. The elements of the matrix are 0 (the Os), 1 (the Xs), or -1 (empty). For example:

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 1 \\ -1 & 1 & 0 \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & 1 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

X wins the game if any row, any column, or any main diagonal is all 1s. In the first board, X wins because the middle column is all 1s. In the second board, X wins because the main diagonal from top-left to bottom-right is all 1s. In the third board, X wins because the second row is all 1s. In the fourth board, X wins because the main diagonal from top-right to bottom-left is all 1s.

Complete the following MATLAB function:

```
function tf = xwins(A)
%XWINS Does X have tic-tac-toe?
%   tf = xwins(A) returns true if X wins on the
%   tic-tic-toe board A. A is a square matrix.
%
%   X wins if any row, any column, or either main
%   diagonal of A is made up of all 1s.
```

You may use the MATLAB functions `all`, `any`, `sum`, and any function that determines the size of a matrix, but do not use any other MATLAB functions in your solution.

46. In one of your labs you computed the moving average on a vector. It is also possible (and common) to compute the moving average on a matrix. The moving average on a matrix `A` is the matrix `B` having the same dimensions as `A` where element `B(row, col)` is equal to the average of the 9 values formed by the 3×3 neighbourhood around `A(row, col)`. If there is no 3×3 neighbourhood around `A(row, col)` then `B(row, col)` is equal to `0`.

    For example, consider the matrix `A`:

$$\texttt{A} = \begin{bmatrix} 2 & 1 & 4 \\ 5 & 6 & 2 \\ 1 & 1 & 5 \end{bmatrix}$$

    then the moving average of `A` is

$$\texttt{B} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

    The value 3 is the average of of the 9 elements of `A`; the 0s in `B` occur because there is no 3×3 neighbourhood around the corresponding elements in `A`.

    Consider the matrix `A`:

$$\texttt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

    then the moving average of `A` is

$$\texttt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

    On the next page you can find a series of figures that show you how elements of `B` are computed by averaging elements from `A`.

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

$$\mathtt{B} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 3 & 3 & 2 & 0 \\ 0 & 5 & 6 & 7 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathtt{A} = \begin{bmatrix} 2 & 1 & 4 & 2 & 1 & -5 \\ 5 & 6 & 2 & 5 & 6 & 2 \\ 1 & 1 & 5 & 1 & 1 & 5 \\ -8 & 14 & 19 & 1 & 23 & 1 \end{bmatrix}$$

Complete the following MATLAB function:

```
function B = movingAvg(A)
%MOVINGAVG Moving average on a matrix
%   B = movingAvg(A) computes the 3x3 moving average on
%   a matrix A.
%
%   B(row, col) is zero where a 3x3 neighbourhood does not
%   exist at A(row, col).
```

You may use the MATLAB functions `sum`, `mean`, `mean2`, and any function that determines the size of a matrix, but do not use any other MATLAB functions in your solution.

Example usage:

```
A = [2     1     4     2     1    -5;
     5     6     2     5     6     2;
     1     1     5     1     1     5;
    -8    14    19     1    23     1];
B = movingAvg(A)
```