E7: Introduction to Computer Programming for Scientists and Engineers
University of California at Berkeley, Spring 2017
Instructor: Lucas A. J. Bastien

Lab Assignment 02: Functions and Arrays
Version: release

Due date: Friday February 3rd 2017 at 12 pm.

General instructions, guidelines, and comments:

- For each question, you will have to write and submit one or more Matlab functions. We provide a number of test cases that you can use to test your function. Note that the fact that your function works for all test cases thus provided does not necessarily guarantee that it will work for all possible test cases relevant to the question. It is your responsibility to test your function thoroughly, to ensure that it will also work in situations not covered by the test cases provided. During the grading process, your function will be evaluated on a number of test cases, some of which are provided here, some of which are not.
- Submit on bCourses one .m file for each function that you have to write. The name of each file must be the name of the corresponding function, with the suffix .m appended to it. For example, if the name of the function is my_function, the name of the file that you have to submit is my_function.m. **Carefully check the name of each file that you submit.** Do not submit any zip file. If you re-submit a file that you have already submitted, bCourses may rename the file by adding a number to the file's name (*e.g.,* rename my_function.m into my_function-01.m). This behavior is okay and should be handled seamlessly by our grading system. Do not rename the file yourself as a response to this behavior.
- A number of optional Matlab toolboxes can be installed alongside Matlab to give it more functionality. All the functions that you have to write to complete this assignment can, however, be implemented without the use of any optional Matlab toolbox. We encourage you to not use optional toolboxes to complete this assignment. All functions of the Matlab base installation will be available to our grading system, but functions from optional toolboxes may not. If one of your function uses a function that is not available to our grading system, you will loose all points allocated to the corresponding part of this assignment. To guarantee that you are not using a Matlab function from an optional toolbox that is not available to our grading system, use one or both of the following methods:
  - ◇ Only use functions from the base installation of Matlab.
  - ◇ Make sure that your function works on the computers of the 1109 Etcheverry Hall computer lab. All the functions available on these computers will be available to our grading system.

- For this assignment, the required submissions are:
  - ◇ `my_sin_approx.m`
  - ◇ `my_projectile.m`
  - ◇ `my_collision.m`
  - ◇ `my_projection.m`
  - ◇ `my_array_metrics_num.m`
  - ◇ `my_array_metrics_lgcl.m`
  - ◇ `my_polygon_perimeter.m` .

1. Approximation of the Sine Function

Write a function with the following header:

```
function [exact, approx, actual_error, relative_error] = my_sin_approx(x)
```

where:

- `x` is a scalar of class `double`.
- `exact` is a scalar of class `double`, and is the "exact" value of $\sin(x)$ as calculated by Matlab's built-in function `sin`. Note that in this problem we describe this value as "exact" even though it may not be rigorously exact, as we will see later in the semester, when discussing binary representations of numbers.
- `approx` is a scalar of class `double`, and is the approximation of $\sin(x)$ calculated using the following approximation:

$$sin(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

- `actual_error` is a scalar of class `double`, and is the error made on $\sin(x)$ when using the approximation instead of the exact value (*i.e.* `approx-exact`).
- `relative_error` is a scalar of class `double`, and is the relative error made on $\sin(x)$ when using the approximation instead of the exact value (*i.e.* `(approx-exact)/exact`).

Note that if `exact` is zero, then the calculation of `relative_error` involves a division by zero. Do not have Matlab handle these cases any differently than the other cases. Matlab will use the values `Inf`, `-Inf`, and `NaN` where appropriate.

Hint: you can use Matlab's built-in function `factorial` to calculate the factorial of a number (*e.g.,* `factorial(3)` to calculate the factorial of 3).

Test cases:

```
>> [exact, approx, actual_error, relative_error] = my_sin_approx(1)

exact =
    0.8415
approx =
```

```
    0.8415
actual_error =
  -2.7308e-06
relative_error =
  -3.2453e-06

>> [exact, approx, actual_error, relative_error] = my_sin_approx(5)

exact =
   -0.9589
approx =
   -5.2927
actual_error =
   -4.3337
relative_error =
    4.5194

>> [exact, approx, actual_error, relative_error] = my_sin_approx(-5)

exact =
    0.9589
approx =
    5.2927
actual_error =
    4.3337
relative_error =
    4.5194

>> [exact, approx, actual_error, relative_error] = my_sin_approx(0)

exact =
     0
approx =
     0
actual_error =
     0
relative_error =
   NaN
```

2. Inelastic Collision

Consider the collision of two imperfect billiard balls of equal mass. They are considered imperfect because when one ball hits another, some amount of kinetic energy is lost in the rebound. A collision where some amount of kinetic energy is lost is categorized as "inelastic".

The ball that billiard players must hit with the cue is called the cue ball. In this question, the cue ball is initially at rest. At time $t = 0$, a player hits the cue ball with the cue, thus imparting it with an initial velocity $\vec{V}_{\text{initial}}$. The cue ball then moves towards and eventually collides with another ball, which we call the eight ball. After the collision, the cue ball moves with velocity $\vec{V}_{\text{cue}}$ in a direction that forms an angle θ_{cue} with its original moving direction.

The eight ball then moves with velocity $\vec{V}_{\text{eight}}$ in a direction that makes an angle θ_{eight} with the original moving direction of the cue ball. Figure 1 illustrates the system before (left panel) and after (right panel) the collision. In this problem, the billiard balls are assumed to be point masses (*i.e.* they have a mass but their radius is zero).

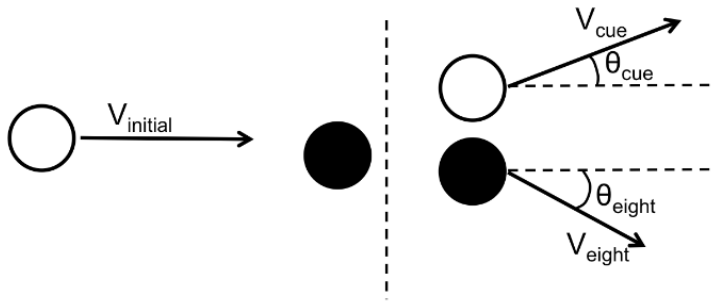

Figure 1: Inelastic collision of two billiard balls. The left panel illustrates the system before the collision. The right panel illustrates the system after the collision. The lengths of the arrows representing the velocities of the balls are not necessarily to scale.

Momentum is conserved throughout the collision, yielding the following two equations:

$$mV_{\text{initial}} = mV_{\text{cue}}\cos(\theta_{\text{cue}}) + mV_{\text{eight}}\cos(\theta_{\text{eight}}) \tag{1}$$
$$mV_{\text{cue}}\sin(\theta_{\text{cue}}) - mV_{\text{eight}}\sin(\theta_{\text{eight}}) = 0 \tag{2}$$

which can be simplified to (since $m \neq 0$):

$$V_{\text{initial}} = V_{\text{cue}}\cos(\theta_{\text{cue}}) + V_{\text{eight}}\cos(\theta_{\text{eight}}) \tag{3}$$
$$V_{\text{cue}}\sin(\theta_{\text{cue}}) - V_{\text{eight}}\sin(\theta_{\text{eight}}) = 0 \tag{4}$$

The amount of kinetic energy lost during the collision, e_{lost}, is:

$$e_{\text{lost}} = \frac{m}{2}V_{\text{initial}}^2 - \frac{m}{2}V_{\text{cue}}^2 - \frac{m}{2}V_{\text{eight}}^2 \tag{5}$$

Write a function with the following header:

```
function [v_cue, v_eight, e_lost] = my_collision(m, v_initial, theta_cue, theta_eight)
```

where:

- `m` is a scalar of class `double` that represents the mass m (in kilograms) of a billiard ball.
- `v_initial` is a scalar of class `double` that represents the magnitude V_{initial} (in m s^{-1}) of the velocity of the cue ball before the collision occurs.
- `theta_cue` is a scalar of class `double` that represents the angle θ_{cue} (in degrees) between the moving direction of the cue ball after the collision and its moving direction before the collision.
- `theta_eight` is a scalar of class `double` that represents the angle θ_{eight} (in degrees) between the moving direction of the eight ball after the collision and the moving direction of the cue ball before the collision.
- `v_cue` is a scalar of class `double` that represents the magnitude V_{cue} (in m s^{-1}) of the velocity of the cue ball after the collision occurs.
- `v_eight` is a scalar of class `double` that represents the magnitude V_{eight} (in m s^{-1}) of the velocity of the eight ball after the collision occurs.
- `e_lost` is a scalar of class `double` that represents the amount of kinetic energy e_{lost} (in Joules) lost during the collision.

The angles θ_{cue} and θ_{eight} are unsigned angles (*i.e.* they are always thought of as positive angles). You can assume that $\theta_{\text{cue}} + \theta_{\text{eight}} \leqslant 90$ degrees.

Hint: to calculate V_{cue} and V_{eight}, you are given two equations that feature these two unknown quantities. Derive the expressions of these unknown quantities as a function of the known quantities on paper before starting to code the solution to this question. When applied to engineering and science, computer programming often involves working out some math on paper before starting to code!

Test cases:

```
>> [v_cue, v_eight, e_lost] = my_collision(1, 3, 35, 50)
v_cue =
    2.3069
v_eight =
    1.7273
e_lost =
    0.3473

>> [v_cue, v_eight, e_lost] = my_collision(1, 5, 45, 30)
v_cue =
    2.5882
v_eight =
    3.6603
e_lost =
    2.4519

>> [v_cue, v_eight, e_lost] = my_collision(5, 10, 15, 5)
v_cue =
    2.5483
```

```
v_eight =
    7.5674
e_lost =
   90.6034
```

3. Kinematics of a Projectile

In this question, we consider the motion of a projectile in air, as illustrated by Figure 2. The effects of the resistance of air on the projectile (drag) are neglected. Under this assumption, the only force acting on the projectile is the weight of the projectile. The motion of the projectile is two-dimensional. We use x as the coordinate along the horizontal direction and y as the coordinate along the vertical direction. We assume that the projectile is thrown at time $t = 0$ from the origin, *i.e.* the point of coordinates $(0, 0)$. We assume that the ground is horizontal and located at height $y = 0$.

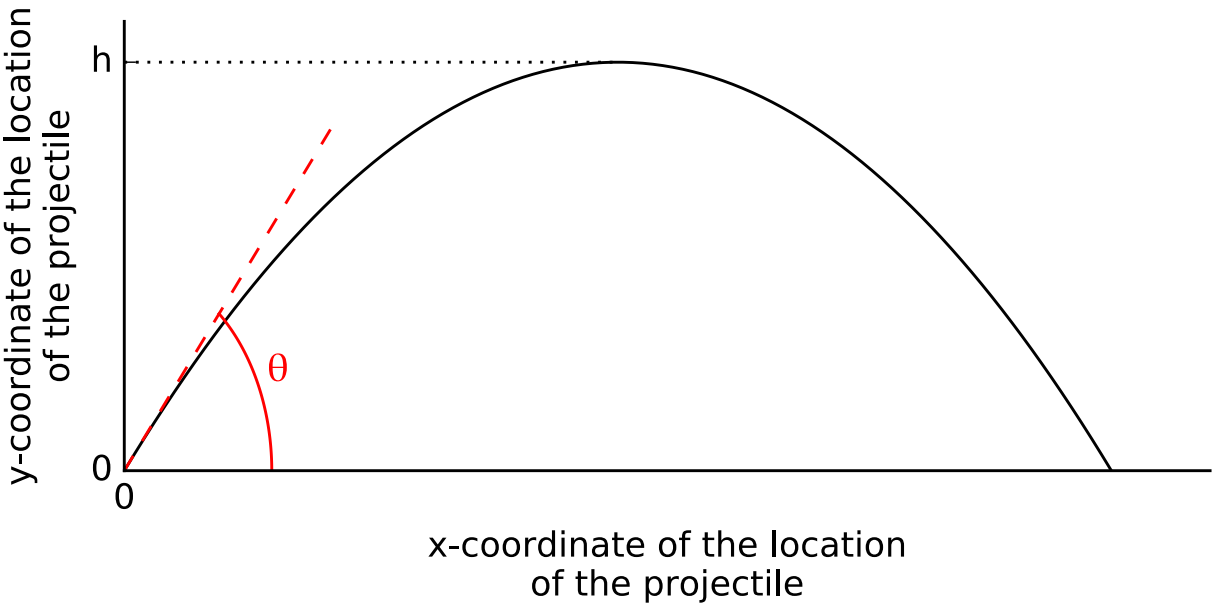

Figure 2: Schematic of the motion of a projectile in air.

Initial velocity

The projectile is thrown at time $t = 0$ with a velocity of magnitude v_0 and at an angle θ with the ground. The horizontal and vertical components of the initial velocity vector, v_{0x} and v_{0y}, respectively, are therefore:

$$v_{0x} = v_0 \cos(\theta) \tag{6}$$
$$v_{0y} = v_0 \sin(\theta) \tag{7}$$

Acceleration

Newton's second law of motion indicates that the horizontal and vertical components of the acceleration vector, a_x and a_y, respectively, are:

$$a_x = 0 \tag{8}$$
$$a_y = -g \tag{9}$$

where g is the acceleration of gravity. The horizontal component of the acceleration vector, a_x, is zero because no forces act in that direction. Acceleration along the vertical direction is entirely due to the action of gravity. Note that acceleration is constant throughout the flight of the projectile, and its magnitude a is:

$$a = \sqrt{a_x^2 + a_y^2} \tag{10}$$
$$= |g| \tag{11}$$
$$= g \tag{12}$$

Velocity

The horizontal and vertical components of the velocity vector at time t, $v_x(t)$ and $v_y(t)$, respectively, are:

$$v_x(t) = v_0 \cos(\theta) \tag{13}$$
$$v_y(t) = v_0 \sin(\theta) - gt \tag{14}$$

The magnitude $v(t)$ of the velocity vector at time t is:

$$v(t) = \sqrt{v_x^2(t) + v_y^2(t)} \tag{15}$$

Position

The position of the projectile at time t can be described by its x- and y-coordinates at time t, respectively:

$$x(t) = v_0 \cos(\theta)t \tag{16}$$
$$y(t) = v_0 \sin(\theta)t - \frac{gt^2}{2} \tag{17}$$

The distance $d(t)$ at time t between the projectile and the origin is:

$$d(t) = \sqrt{x(t)^2 + y(t)^2} \tag{18}$$

Time of flight

The time during which the projectile is in motion in the air before hitting the ground is called the time of flight. When the projectile hits the ground, its position along the y-axis is $y = 0$. The time of flight t_f is therefore the strictly positive solution of the equation $y(t_f) = 0$:

$$y(t_f) = 0 \tag{19}$$

$$\iff v_0 \sin(\theta) t_f - \frac{g t_f^2}{2} = 0 \tag{20}$$

$$\iff t_f (v_0 \sin(\theta) - \frac{g t_f}{2}) = 0 \tag{21}$$

The only non-zero solution to this equation, *i.e.* the time of flight t_f, is:

$$t_f = \frac{2 v_0 \sin(\theta)}{g} \tag{22}$$

Maximum height

The projectile reaches its maximum height above ground, h, at time t_h given by $v_y(t_h) = 0$. Therefore:

$$t_h = \frac{v_0 \sin(\theta)}{g} \tag{23}$$

and:

$$h = y(t_h) \tag{24}$$

$$= \frac{v_0^2 \sin^2(\theta)}{2g} \tag{25}$$

Note that $t_h = t_f/2$.

Write a function with the following header:

```
function [solution] = my_projectile(v0, theta, t)
```

where:

- `v0` is a scalar of class `double` that represents the magnitude v_0 (in m s^{-1}) of the velocity of the projectile at time $t = 0$.
- `theta` is a scalar of class `double` that represents the angle θ (in degrees) that the velocity vector of the projectile at time $t = 0$ makes with the ground.
- `t` is a scalar of class `double` that represents the time t (in s). You can assume that this time is smaller than the time of flight of the projectile.
- `solution` is an 1×13 row vector of class `double`. The values in this array are the following quantities, in this order:
  1. Horizontal component v_{0x} (in m s^{-1}) of the initial velocity of the projectile.
  2. Vertical component v_{0y} (in m s^{-1}) of the initial velocity of the projectile.
  3. Horizontal component a_x (in m s^{-2}) of the acceleration vector of the projectile at time t.
  4. Vertical component a_y (in m s^{-2}) of the acceleration vector of the projectile at time t.
  5. Magnitude a (in m s^{-2}) of the acceleration vector of the projectile at time t.
  6. Horizontal component $v_x(t)$ (in m s^{-1}) of the velocity vector of the projectile at time t.
  7. Vertical component $v_y(t)$ (in m s^{-1}) of the velocity vector of the projectile at time t.
  8. Magnitude $v(t)$ (in m s^{-1}) of the velocity vector of the projectile at time t.
  9. Horizontal coordinate $x(t)$ (in m) of the position of the projectile at time t.
  10. Vertical coordinate $y(t)$ (in m) of the position of the projectile at time t.
  11. Distance $d(t)$ (in m) between the projectile and the origin at time t.
  12. Time of flight t_f (in s) of the projectile.
  13. Maximum height h (in m) reached by the projectile during its motion.

Use $g = 9.81$ m s^{-2}.

Test cases:

```
>> [solution] = my_projectile(10, 53, 1.4)

solution =
    6.0182  7.9864  0  -9.8100  9.8100  6.0182  -5.7476  8.3219  8.4254
    1.5671  8.5699  1.6282  3.2509

>> [solution] = my_projectile(10, 53, 1)

solution =
    6.0182  7.9864  0  -9.8100  9.8100  6.0182  -1.8236  6.2884  6.0182
    3.0814  6.7611  1.6282  3.2509

>> [solution] = my_projectile(20, 35, 1)
```

```
solution =
   16.3830  11.4715  0  -9.8100  9.8100  16.3830  1.6615  16.4671  16.3830
   6.5665  17.6500  2.3387  6.7072

>> [solution] = my_projectile(17, 45, 2)

solution =
   12.0208  12.0208  0  -9.8100  9.8100  12.0208  -7.5992  14.2214  24.0416
   4.4216  24.4449  2.4507  7.3649

>> [solution] = my_projectile(15, 85, 3)

solution =
    1.3073  14.9429  0  -9.8100  9.8100  1.3073  -14.4871  14.5459  3.9220
    0.6838  3.9812  3.0465  11.3808
```

4. Vector Projection

Consider an integer $n \geqslant 2$. $\mathbb{R}^n$ is the set of vectors of the form $\vec{x} = (x_1, x_2, \ldots, x_n)$ where all the components x_i $(i = 1, 2, \ldots, n)$ of the vector are real numbers. The dot product $\vec{x} \cdot \vec{y}$ between two vectors $\vec{x} = (x_1, x_2, \ldots, x_n)$ and $\vec{y} = (y_1, y_2, \ldots, y_n)$ of $\mathbb{R}^n$ is defined as:

$$\vec{x} \cdot \vec{y} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n \tag{26}$$

The projection of vector $\vec{y}$ onto a non-zero vector $\vec{x}$ is the vector $\vec{p} \in \mathbb{R}^n$ such that:

$$\vec{p} = \frac{\vec{x} \cdot \vec{y}}{\vec{x} \cdot \vec{x}} \vec{x} \tag{27}$$

Write a function with the following header:

```
function [p] = my_projection(x, y)
```

where `x`, `y`, and `p` are a $1 \times n$ row vectors of class `double` that represent vectors of $\mathbb{R}^n$ and such that `p` is the projection of vector `y` onto vector `x`. You can assume that each of these vectors contains at least two elements (*i.e.* $n \geqslant 2$), and that at least one element of x is non-zero.

Test cases:

```
>> p = my_projection([0, 1], [2, 4])

p =
     0     4

>> p = my_projection([2, 0], [0, 1])
```

```
p =
     0     0

>> p = my_projection([2, 3], [7, 5])

p =
    4.4615    6.6923


>> p = my_projection([2, 3, 4, 9], [7, 5, 0, 1])

p =
    0.6909    1.0364    1.3818    3.1091
```

5. Array Metrics

5.1. Numerical Metrics

Write a function with the following header:

```
function [n_positive, n_negative, n_zero, n_special] = my_array_metrics_num(array)
```

where:

- `array` is an arbitrary $n \times m$ array of class `double`.
- `n_positive`, `n_negative`, `n_zero`, and `n_special` are scalars of class `double` that represent, respectively, the number of strictly positive, strictly negative, zero, and "special" values in `array`. Special values are `NaN`, `Inf`, and `-Inf`.

Note that `Inf` is both positive and "special", and `-Inf` is both negative and "special".

Test cases:

```
>> [n_positive, n_negative, n_zero, n_special] = my_array_metrics_num([10])
n_positive =
     1
n_negative =
     0
n_zero =
     0
n_special =
     0

>> array = [5, 0, 0, 1, -1, 7];
>> [n_positive, n_negative, n_zero, n_special] = my_array_metrics_num(array)
n_positive =
     3
n_negative =
     1
n_zero =
```

```
     2
n_special =
     0

>> array = [-Inf, 5, -1; 0, NaN, 3];
>> [n_positive, n_negative, n_zero, n_special] = my_array_metrics_num(array)
n_positive =
     2
n_negative =
     2
n_zero =
     1
n_special =
     2
```

5.2. Logical Metrics

Write a function with the following header:

```
function [positive, negative, zero, special] = my_array_metrics_lgcl(array)
```

where:

- `array` is an arbitrary $n \times m$ array of class `double`.
- `positive` is a scalar of class `logical` that is true if and only if `array` contains at least one strictly positive value.
- `negative` is a scalar of class `logical` that is true if and only if `array` contains at least one strictly negative value.
- `zero` is a scalar of class `logical` that is true if and only if `array` contains at least one zero value.
- `special` is a scalar of class `logical` that is true if and only if `array` contains at least one "special" value. Special values are `NaN`, `Inf`, and `-Inf`.

Note that `Inf` is both positive and "special", and `-Inf` is both negative and "special".

Test cases:

```
>> [positive, negative, zero, special] = my_array_metrics_lgcl([10])
positive =
    logical
     1
negative =
    logical
     0
zero =
    logical
     0
special =
    logical
```

```
     0

>> array = [5, 0, 0, 1, -1, 7];
>> [positive, negative, zero, special] = my_array_metrics_lgcl(array)
positive =
    logical
     1
negative =
    logical
     1
zero =
    logical
     1
special =
    logical
     0

>> array = [-Inf, 5, -1; 0, NaN, 3];
>> [positive, negative, zero, special] = my_array_metrics_lgcl(array)
positive =
    logical
     1
negative =
    logical
     1
zero =
    logical
     1
special =
    logical
     1
```

6. Perimeter of a Polygon

Consider a polygon in the (x, y)-plane defined by n vertices $V_1, V_2, \ldots V_n$. Call (x_i, y_i) the coordinates of vertex V_i. Figure 3 shows an example of an arbitrary polygon with $n = 6$ sides (*i.e.* a hexagon).

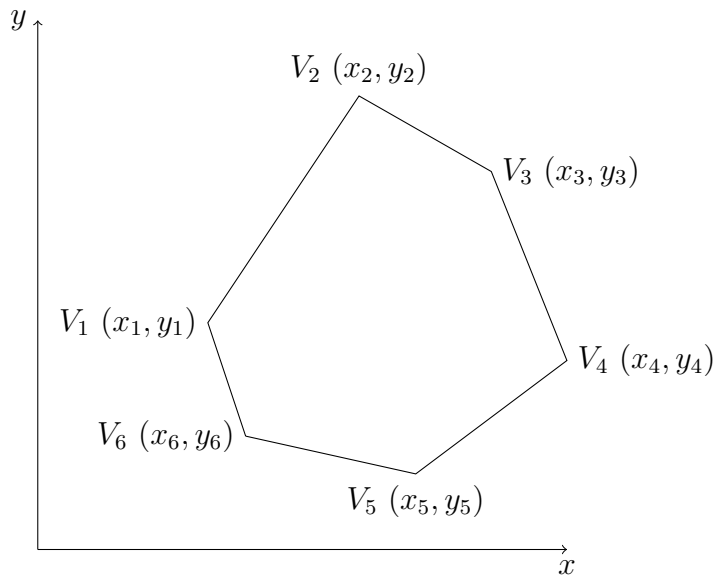

Figure 3: Example of an arbitrary polygon with $n = 6$ sides (*i.e.* a hexagon). The 6 vertices of the hexagon are $V_1, V_2, \ldots, V_6$. The coordinates of vertex V_i are (x_i, y_i).

Write a function with the following header:

```
function [perimeter] = my_polygon_perimeter(x, y)
```

where:

- `x` is a $n \times 1$ column vector of class `double` that represents the x-coordinates of the vertices of the polygon (*i.e.* $x_1, x_2, \ldots x_n$, in that order).
- `y` is a $n \times 1$ column vector of class `double` that represents the y-coordinates of the vertices of the polygon (*i.e.* $y_1, y_2, \ldots y_n$, in that order).
- `perimeter` is a scalar of class `double` that represents the perimeter of the polygon.

You can assume that the vertices described by the inputs `x` and `y` correspond to a valid polygon whose sides do not intersect.

Test cases:

```
>> % Calculate the perimeter of a square
>> x = [0; 0; 2; 2];
>> y = [0; 2; 2; 0];
>> perimeter = my_polygon_perimeter(x, y)
perimeter =
     8

>> % Calculate the perimeter of a triangle
>> x = [1; 4; 7];
```

```
>> y = [1; 5; 1];
>> perimeter = my_polygon_perimeter(x, y)
perimeter =
    16

>> % Calculate the perimeter of a hexagon
>> x = [3; 2; 4; 6; 7; 5];
>> y = [1; 10; 6; 5; 2; 1];
>> perimeter = my_polygon_perimeter(x, y)
perimeter =
   23.1619
```