Total Points: / 100

Submission Instruction: Please submit this homework on Canvas in a single pdf format. The filename should be "HWXX_FullName_RedID.pdf" (ex. HW05_JamesGault_12345678.pdf).
Please copy your Matlab code in the given box. Adjust the box size as needed.
Please also submit all your m files separately. **Don't zip them**.

Full Name: Evan Peres

Red ID: 129964468

Email address: eperes2481@sdsu.edu

Cell Array and Structure (Write your code in the box.)

1. Write a function with header [newStudent] = myNewStudent (name, id, grades) that inputs a name, id, and grades, and generates a 1×1 struct array newStudent. (/ 10)

Test Cases:

```
>> student = myNewStudent('Tim', 1, [100, 98])
student =

  struct with fields:

      name: 'Tim'
        id: 1
    grades: [100 98]
```

```
function [newStudent] = myNewStudent(name, id, grades)
   % Create a struct array with fields: name, id, and grades
   newStudent.name = name;
   newStudent.id = id;
   newStudent.grades = grades;
end
```

2. Let C be a square connectivity matrix containing zeros and ones. We say that point i has a connection to point j, or i is connected to j, if C(i,j) = 1. Note that connections in this context are one-directional, meaning C(i,j) is not necessarily the same as C(j,i). For example, think of a one-way street from point A to point B. If A is connected to B, then B is not necessarily connected to A.

   Write a function with header [node] = myConnectivityMat2Struct(C, names) where C is a connectivity matrix and names is a cell array of strings (i.e., each element of names is a string) that denote the name of a point. That is, names(i) is the name of the i − th point.

   The output variable node should be a struct with fields .name and .neighbors. The i − th element of node is defined as node(i).name = names(i) and node(i).neighbors is a row vector containing the indices, j, such that C(i,j) = 1. In other words, node(i).neighbors is a list of points that point i is connected to.

Warning: Make sure the field names are exactly correct: .name and .neighbors. (/ 10)

Test Cases:

```
>> C= [0 1 0 1; 1 0 0 1; 0 0 0 1; 1 1 1 0];
>> names = {'Los Angeles', 'New York', 'Miami', 'Dallas'};
>> node = myConnectivityMat2Struct(C, names);
>> node(1)
ans =
      name: 'Los Angeles'
  neighbors: [2 4]
>> node(2)
ans =
      name: 'New York'
  neighbors: [1 4]
>> node(3)
 ans =
      name: 'Miami'
  neighbors: 4
>> node(4)
ans =
      name: 'Dallas'
  neighbors: [1 2 3]
```

```
function [node] = myConnectivityMat2Struct(C, names)
  % Initialize the output struct array
  n = length(names); % Number of points (nodes)
  node = struct('name', cell(1, n), 'neighbors', cell(1, n));

  % Populate the struct array
  for i = 1:n
    node(i).name = names{i}; % Set the name
    node(i).neighbors = find(C(i, :) == 1); % Find indices where C(i, j) = 1
  end
end
```

Custom Functions (Write your code in the box.)

3. Let Q(x) be the quadratic equation $Q(x) = ax^2 + bx + c$ for some scalar values a, b, and c. A root of Q(x) is an r such that Q(r) = 0. The two roots of a quadratic equation can be described by the quadratic formula, which is

$$r = \frac{-b \pm \sqrt{(b^2 - 4ac)}}{2a}$$

A quadratic equation has either two real roots (i.e. $b^2 > 4ac$), two imaginary roots (i.e. $b^2 < 4ac$),, or one root, (i.e. $b^2 = 4ac$).

Write a function with header [nRoots, r] = myNRoots(a,b,c) where a, b, and c are the coefficients of the quadratic Q(x), nRoots is 2 if Q has 2 real roots, 1 if Q has 1 root, -2 if Q has two imaginary roots, and r is an array containing the roots of Q. (10)

Test Cases:

```
>> [nRoots, r] = myNRoots(1, 0,−9)

nRoots = 2

r=
     3     −3

>> [nRoots, r] = myNRoots(3,4,5)

 nRoots = −2

r=

−0.6667 + 1.1055i        −0.6667 − 1.1055i

>> [nRoots, r] = myNRoots(2,4,2)

nRoots = 1

r=

−1
```

```
function [nRoots, r] = myNRoots(a, b, c)
   % Calculate the discriminant
   discriminant = b^2 - 4*a*c;

   % Check the number of roots based on the discriminant
   if discriminant > 0
      % Two real roots
      nRoots = 2;
      r = [(-b + sqrt(discriminant)) / (2*a), (-b - sqrt(discriminant)) / (2*a)];
```

```
  elseif discriminant == 0
    % One real root
    nRoots = 1;
    r = -b / (2*a);
  else
    % Two imaginary roots
    nRoots = -2;
    realPart = -b / (2*a);
    imaginaryPart = sqrt(-discriminant) / (2*a);
    r = [realPart + imaginaryPart*1i, realPart - imaginaryPart*1i];
  end
end
```

4. Write a function with header [h] = mySplitFunction(f,g,a,b,x) where f and g are handles to functions f(x) and g(x), respectively. The output argument h should be f(x) if $x \leq a$, g(x) if $x \geq b$, and 0 otherwise. You may assume that $b > a$. (10)

Test Cases:
>> h = mySplitFunction(@exp, @sin, 2, 4, 1)
h= 2.7183
>> h = mySplitFunction(@exp, @sin, 2, 4, 3)
h= 0
>> h = mySplitFunction(@exp, @sin, 2, 4, 5)
h= −0.9589

```
function [h] = mySplitFunction(f, g, a, b, x)
  % If x <= a, return f(x)
  if x <= a
    h = f(x);
  % If x >= b, return g(x)
  elseif x >= b
    h = g(x);
  % Otherwise, return 0
  else
    h = 0;
  end
end
```

5. Write a function with header [area, perimeter] = rectInfo (w, l) that outputs the area and perimeter of a rectangle, given that the two inputs are the width and length of the rectangle. If the user provides too few inputs, set both outputs to -1. (10)

Test Cases:
>> [area, perimeter] = rectInfo (10, 20)

area = 200

perimeter = 60

>> [area, perimeter] = rectInfo (10)

area = -1

perimeter = -1

```
function [area, perimeter] = rectInfo(w, l)
  % Check if the user has provided both width and length
  if nargin < 2
    % If not, set area and perimeter to -1
    area = -1;
    perimeter = -1;
  else
    % Otherwise, calculate area and perimeter
    area = w * l;
    perimeter = 2 * (w + l);
  end
end
```

Recursion (Write your code in the box.)

6. Chebyshev polynomials are defined recursively. Chebyshev polynomials of the first kind, Tn(x), is defined by the following recurrence relation:

$$T_n(x) = \begin{cases} 1 & if\ n = 0 \\ x & if\ n = 1 \\ 2xT_{n-1}(x) - T_{n-2}(x) & otherwise \end{cases}$$

   Write a function with header[y] = myChebyshevPoly1(n,x) where y is the n-th Chebyshev polynomial of the first kind evaluated at x. Be sure your function can take **array inputs** for x. You may assume that x is a row vector. The output variable, y, must be a row vector also. (10)

Test Cases:

```
>> myChebyshevPoly1(0,1:5)
ans =
     1     1     1     1     1
>> myChebyshevPoly1(1,1:5)
ans =
     1     2     3     4     5
>> myChebyshevPoly1(3,1:5)
ans =
     1    26    99   244   485
```

```
function [y] = myChebyshevPoly1(n, x)
  % Initialize the first two Chebyshev polynomials
  T0 = 1;  % T0(x) = 1
  T1 = x;  % T1(x) = x

  % For n = 0 or n = 1, return the corresponding Chebyshev polynomial
  if n == 0
    y = T0 * ones(size(x));  % T0(x) = 1 for all x
  elseif n == 1
    y = T1;  % T1(x) = x
  else
    % For n > 1, use the recurrence relation to calculate the nth Chebyshev polynomial
    Tn_2 = T0;  % T(n-2)
    Tn_1 = T1;  % T(n-1)

    for k = 2:n
      % Recurrence relation: Tn(x) = 2*x*T(n-1)(x) - T(n-2)(x)
      Tn = 2 * x .* Tn_1 - Tn_2;
      % Update for the next iteration
      Tn_2 = Tn_1;
      Tn_1 = Tn;
    end

    % Return the nth Chebyshev polynomial
    y = Tn;
  end
end
```

7. The golden ratio, ∅, is the limit of $\frac{F(n+1)}{F(n)}$ as n goes to infinity and $F(n)$ is the n-th Fibonacci number, which can be shown to be exactly $\emptyset = \frac{1+\sqrt{5}}{2}$, and is approximately 1.62. We say that $G(n) = \frac{F(n+1)}{F(n)}$ is the n-th approximation of the golden ratio, and G(1)=1.

   It can be shown that is also the limit of the continued fraction:

$$\emptyset = 1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{1+\dots}}}}$$

   Write a recursive function with header [G] = myGoldenRatio(n) where G is the n-th approximation of the golden ratio according to the continued fraction recursive relationship. You should use the continued fraction approximation for the Golden ratio, not the G(n) = F (n + 1)/F (n) definition. However for both definitions, G(1) = 1. (10)

Test Cases:

```
>> format long
>> G = myGoldenRatio(10)

G= 1.618181818181818

>> (1 + sqrt(5))/2

ans =

  1.618033988749895
```

```
function [G] = myGoldenRatio(n)
   % Base case: for n = 1, G(1) = 1
   if n == 1
      G = 1;
   else
      % Recursive case: G(n) = 1 + 1 / G(n-1)
      G = 1 + 1 / myGoldenRatio(n-1);
   end
end
```

8. The greatest common divisor of two integers a and b is the largest integer that divides both numbers without remainder, and the function to compute it is denoted by GCD(a,b). The GCD function can be written recursively.
   If b equals 0, then a is the greatest common divisor. Otherwise, GCD(a,b) = GCD(b, rem(a,b)) where rem(a,b) is the remainder of a divided by b. You may assume that a and b are 1×1 integer doubles. Write a recursive function with header [gcd] = myGCD(a,b) that computes the greatest common divisor of a and b. You may assume that a and b are 1 × 1 integer doubles. (10)

Test Cases:

```
>> gcd = myGCD(10,4)
gcd = 2
>> gcd = myGCD(33,121)
gcd = 11
>> gcd = myGCD (18,0)
gcd =18
```

```
function [gcd] = myGCD(a, b)
   % Base case: if b is 0, then the GCD is a
   if b == 0
      gcd = a;
   else
```

```
    % Recursive case: GCD(a, b) = GCD(b, rem(a, b))
    gcd = myGCD(b, rem(a, b));
  end
end
```

Simulink (20)

9. Consider a small aircraft/drone as a point-mass dynamics model that is moving in a 2D space. The dynamics equations are

$$m\dot{V} = -mg\, sin\gamma - kV^2$$

$$mV\dot{\gamma} = -mg\, cos\gamma$$

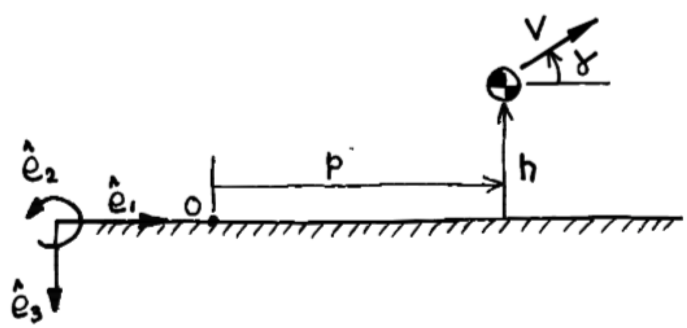
And

$$\dot{p} = Vcos\gamma$$

$$\dot{h} = Vsin\gamma$$

Where p is called the horizontal range and h is the altitude or height of the aircraft. V is the speed and γ is the flight path angle, and $k = \rho S C_D/2$,

The parameters are given as gravity constant: g = 9.81, mass of aircraft: m= 1, air density: rho=0.3809, drag coefficient: Cd=0.4, reference area: S=0.1, kappa = rho*S*Cd/2



Please build a SIMULINK model of this aircraft model with state variables, V, γ, p, h. Test your model with the following initial conditions:

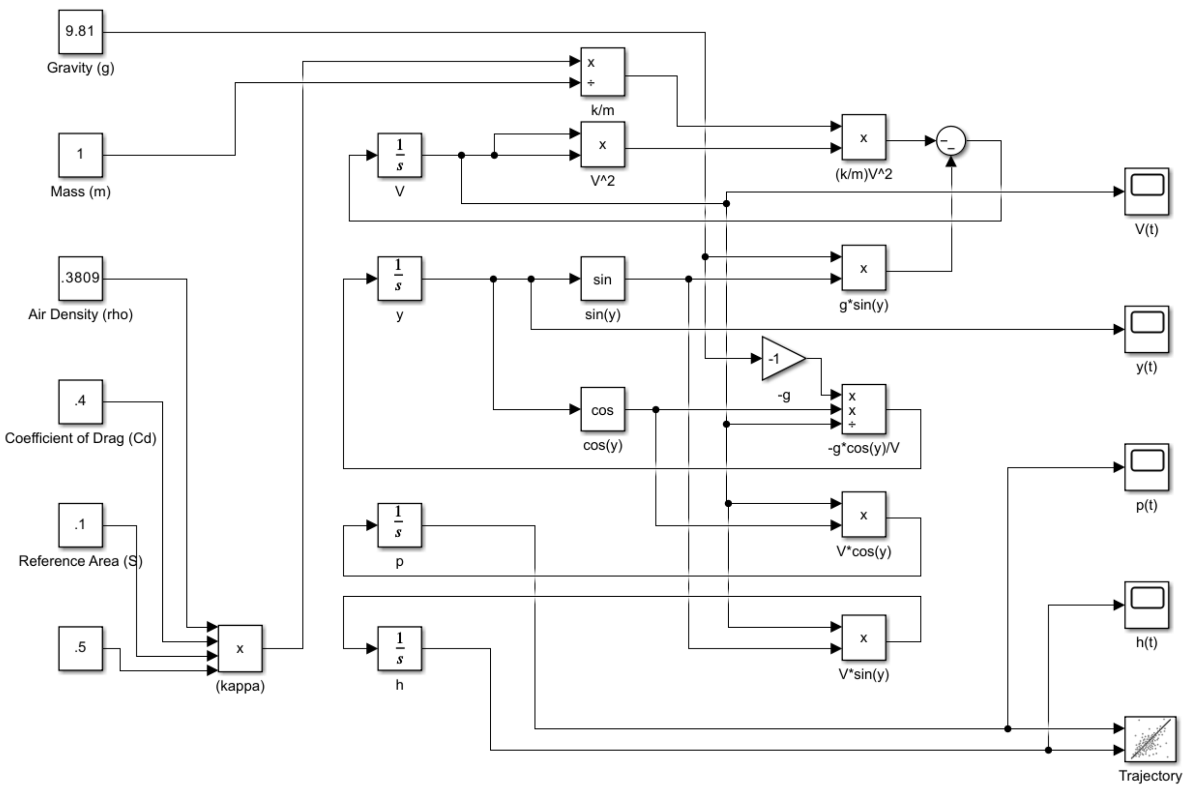
V= 10, $\gamma = pi/4$, p =0, h = 100.

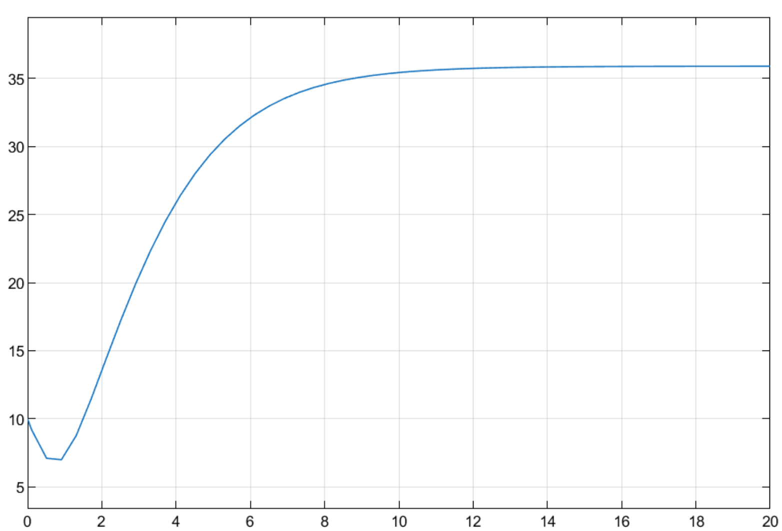
Run your Simulink for 20s.

Total Points: / 100

Put a snapshot of the SIMULINK diagram in this pdf file; illustrate your results with plots of the time histories of V, γ and h (height); also plot the aircraft trajectory, that is, h vs p.

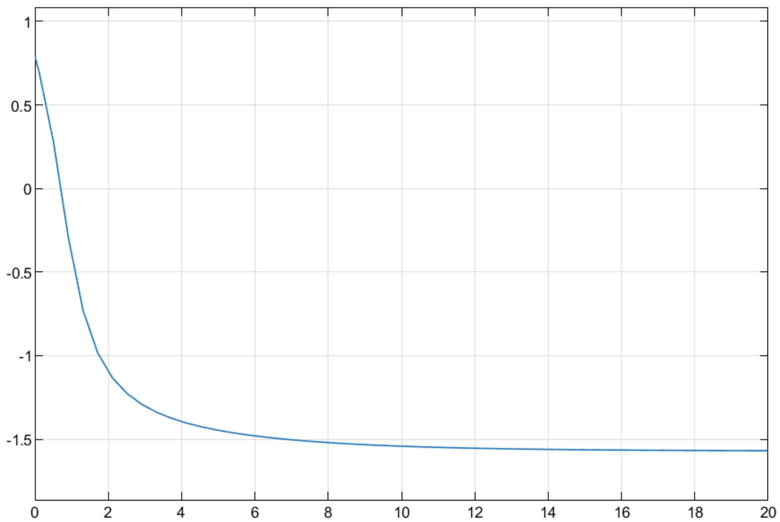
Please also submit your SIMULINK diagram file with Matlab code on canvas(if any)



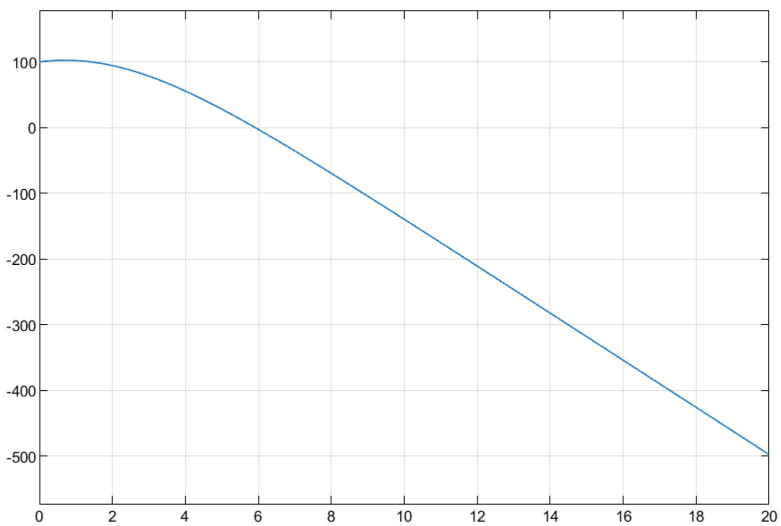


V(t)

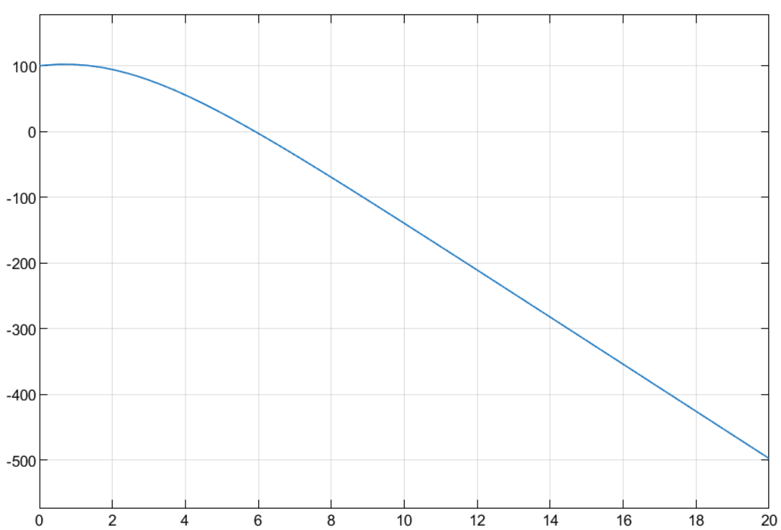
Total Points: / 100



y(t)



h(t)



trajectory