Contents

- E7 Lab 3 Solutions
- Question 1
- Published Test Case
- Additional Test Case
- Question 2
- Published Test Case
- Additional Test Case
- Question 3
- Published Test Case
- Additional Test Case
- Question 4A
- Published Test Case
- Additional Test Case
- Question 4B
- Published Test Case
- Additional Test Case

E7 Lab 3 Solutions

Spring 2016

```
format compact
format short
clear all
clc
close all
```

Question 1

```
type mySmartMultiply
```

```
function [result] = mySmartMultiply(m1, m2)

size_m1 = size(m1); %Gets the size of the array m1
size_m2 = size(m2); %Gets the size of the array m2

if (size_m1(2) == size_m2(1))  %If the inner dimensions of m1 and m2 are the same
    if all(size_m1 == size_m2)  %If the sizes of m1 and m2 are the same
        result = 'multiplication ambiguous';
    else %If the sizes of m1 and m2 are not the same
        result = m1*m2; %Perform matrix multiplication
    end
else %If the inner dimensions of m1 and m2 are not the same
    if all(size_m1 == size_m2)  %If the sizes of m1 and m2 are the same
        result = m1.*m2; %Performs elementwise multiplication
    else %If the sizes of m1 and m2 are not the same
        result = 'no valid multiplication';
    end
end

if (isscalar(m1) || isscalar(m2)) %If at least one input is a scalar
    result = m1*m2; %Overide the previous answer and perform scalar multiplication
end

end
```

Published Test Case

Additional Test Case

```
result = mySmartMultiply(1,2)

result = mySmartMultiply([2,3,3;4,5,5],[1,2;3,4;5,5])

result = mySmartMultiply([2,3;3,4;5,5],[1,2;3,4;5,5])

result = mySmartMultiply([2,5;8,3],[5,1;3,2])

result = mySmartMultiply([2;8],[5,1;3,2])
```

```
result =
     2
result =
    26    31
    44    53
```

```
result =
     2     6
     9    16
    25    25
result =
multiplication ambiguous
result =
no valid multiplication
```

Question 2

```
type classifyFlow
```

```
function [classification] = classifyFlow(u, h, unitsys)

if strcmpi(unitsys, 'metric')  %If metric units are desired
    g = 9.81; %Define gravitational constant in metric units
else %Otherwise, it can be inferred that imperial units are desired
    g = 32.2; %Define gravitational constant in imperial units
end

froude = u/sqrt(g*h); %Calculate the Froude number
froude = 10^-3*round(froude*10^3); %Rounds to three decimal places

if froude == 1
    classification = 'critical';
elseif froude < 1
    classification = 'subcritical';
else
    classification = 'supercritical';
end

end
```

Published Test Case

Additional Test Case

```
classification = classifyFlow(5, .5, 'metric')

classification = classifyFlow(1, 10, 'imperial')

classification = classifyFlow(15.01, 7, 'imperial')
```

```
classification =
supercritical
classification =
subcritical
classification =
critical
```

Question 3

```
type collision
```

```
function [result] = collision(sprite1, sprite2)

result = zeros(1,3); %Initially define the result as an array of zeros

laser = 0; %A count of how many inputs are lasers
rocket = 0; %A count of how many inputs are rockets
player = 0; %A count of how many inputs are players
fighter = 0; %A count of how many inputs are fighters
mothership = 0; %A count of how many inputs are motherships

%Figure out how many of each sprite exists

if strcmpi(sprite1, 'laser')
    laser = 1;
end

if strcmpi(sprite2, 'laser')
    laser = laser + 1;
end

if strcmpi(sprite1, 'rocket')
    rocket = 1;
end

if strcmpi(sprite2, 'rocket')
    rocket = rocket + 1;
end
```

```
if strcmpi(sprite1, 'player')
    player = 1;
end

if strcmpi(sprite2, 'player')
    player = player + 1;
end

if strcmpi(sprite1, 'fighter')
    fighter = 1;
end

if strcmpi(sprite2, 'fighter')
    fighter = fighter + 1;
end

if strcmpi(sprite1, 'mothership')
    mothership = 1;
end

if strcmpi(sprite2, 'mothership')
    mothership = mothership + 1;
end


%Assess results

if (laser == 1) && ((rocket == 1) || (player == 1) || (fighter == 1)) %If a laser goes head to head with either a rocket, player, or fighter

    if strcmpi(sprite1, 'laser') %If the first sprite was the laser
        result(3) = 1; %Sprite two is destroyed
    else
        result(2) = 1; %Sprite one is destroyed
    end

    if (fighter == 1) %If a fighter was called
        result(1) = 1; %A point is given for destroying a fighter
    end

end

if (laser == 1) && (mothership == 1) %If a laser goes head to head with a mothership

    if strcmpi(sprite1, 'laser') %If the first sprite was the laser
        result(2) = 1; %Sprite one is destroyed
    else
        result(3) = 1; %Sprite two is destroyed
    end

end

if (rocket == 2) || ( rocket == 1 && (player == 1 || fighter == 1 || mothership == 1)) %If a rocket goes head to head with either a rocket, player, fighter, or moth

    result(2:3) = [1 1]; %Both are destroyed no matter what

    if (fighter == 1) %If a fighter was called
        result(1) = 1; %A point is given for destroying a fighter
    end

    if (mothership == 1) %If a mothership was called
        result(1) = 20; %Twenty points are given for destroying the mothership
    end

end

if (player == 1) && ((fighter == 1) || (mothership == 1)) %If a player goes head to head with either a fighter or mothership

    result(2:3) = [1 1]; %Both are destroyed no matter what

    if (fighter == 1) %If a fighter was called
        result(1) = 1; %A point is given for destroying a fighter
    end

    if (mothership == 1) %If a fighter was called
        result(1) = 20; %Twenty points are given for destroying the mothership
    end

end

end
```

Published Test Case

```
result = collision('rocket','player')

result = collision('fighter','laser')

result = collision('mothership','fighter')
```

```
result =
     0     1     1
result =
     1     1     0
result =
     0     0     0
```

Additional Test Case

```
result = collision('rocket','mothership')

result = collision('laser','rocket')

result = collision('player','fighter')
```

```
result =
    20     1     1
result =
     0     0     1
result =
     1     1     1
```

Question 4A

```
type vehicleRecommendation
```

```
function [consumerStruct] = vehicleRecommendation(consumerName, state, annualkmTraveled, annualBudget)

%Load the .mat file
load('EV_Comparison.mat');

%Calculate Annual GHG Emissions
annual_GHG_emisions_ca = total_life_cycle_carbon_footprint_ca*annualkmTraveled;
annual_GHG_emisions_ks = total_life_cycle_carbon_footprint_ks*annualkmTraveled;
annual_GHG_emisions_fl = total_life_cycle_carbon_footprint_fl*annualkmTraveled;

%Calculate Annual Cost
annual_cost_ca = normalized_cost_ca*annualkmTraveled;
annual_cost_ks = normalized_cost_ks*annualkmTraveled;
annual_cost_fl = normalized_cost_fl*annualkmTraveled;

%Find Green House Gas Recommendation Index
min_GHG_ca = min(annual_GHG_emisions_ca); %Determines the minimum green house gas emissions
min_GHG_index_ca = find(annual_GHG_emisions_ca == min_GHG_ca); %Determines the index where the minimum green house gas emisions exists

min_GHG_ks = min(annual_GHG_emisions_ks); %Determines the minimum green house gas emissions
min_GHG_index_ks = find(annual_GHG_emisions_ks == min_GHG_ks); %Determines the index where the minimum green house gas emisions exists

min_GHG_fl = min(annual_GHG_emisions_fl); %Determines the minimum green house gas emissions
min_GHG_index_fl = find(annual_GHG_emisions_fl == min_GHG_fl); %Determines the index where the minimum green house gas emisions exists

%Find Cost Recommendation Index

cost_difference_ca = annualBudget - annual_cost_ca; %Create an array of the annual cost minus the annual budget
current_winner_ca = 1e6;  %Create a value for the closest difference without going over that is enourmous (it is designed to be beaten in the for loop)

for i = 1:numel(cost_difference_ca)
    if (cost_difference_ca(i) >= 0) && (cost_difference_ca(i) < current_winner_ca) %If the difference is positive (we didn't exceed the annual budget) and if it is
        current_winner_ca = cost_difference_ca(i); %The smaller difference becomes the current winning difference
    end
end

cost_index_ca = find(cost_difference_ca == current_winner_ca); %Determines the index where the cost was closest to the annual budget without going over

cost_difference_ks = annualBudget - annual_cost_ks; %Create an array of the annual cost minus the annual budget
current_winner_ks = 1e6;  %Create a value for the closest difference without going over that is enourmous (it is designed to be beaten in the for loop)

for i = 1:numel(cost_difference_ks)
    if (cost_difference_ks(i) >= 0) && (cost_difference_ks(i) < current_winner_ks) %If the difference is positive (we didn't exceed the annual budget) and if it is
        current_winner_ks = cost_difference_ks(i); %The smaller difference becomes the current winning difference
    end
end

cost_index_ks = find(cost_difference_ks == current_winner_ks); %Determines the index where the cost was closest to the annual budget without going over

cost_difference_fl = annualBudget - annual_cost_fl; %Create an array of the annual cost minus the annual budget
current_winner_fl = 1e6;  %Create a value for the closest difference without going over that is enourmous (it is designed to be beaten in the for loop)

for i = 1:numel(cost_difference_fl)
    if (cost_difference_fl(i) >= 0) && (cost_difference_fl(i) < current_winner_fl) %If the difference is positive (we didn't exceed the annual budget) and if it is
        current_winner_fl = cost_difference_fl(i); %The smaller difference becomes the current winning difference
    end
end

cost_index_fl = find(cost_difference_fl == current_winner_fl); %Determines the index where the cost was closest to the annual budget without going over
```

```
%Create the Overlying Structure
if strcmpi(state, 'CA') %If the desired state is California

    %Create the GHG_Recommendation Structure
    GHG_vehicle_ca = car_make{min_GHG_index_ca}; %Gives the vehicle with the minimum green house gas emissions in a cell form
    GHG_vehicle_ca_model = car_model{min_GHG_index_ca}; %Gives the vehicle model with the minimum green house gas emissions in a cell form
    GHG_ca.Vehicle = [GHG_vehicle_ca ' ' GHG_vehicle_ca_model]; %Gives the vehicle with the minimum green house gas emissions in a string form
    GHG_ca.Cost = annual_cost_ca(min_GHG_index_ca); %Gives the annual cost
    GHG_ca.GHG = annual_GHG_emisions_ca(min_GHG_index_ca); %Gives the annual green house gas emissions

    %Create the Cost_Recommendation Structure
    Cost_vehicle_ca = car_make{cost_index_ca}; %Gives the vehicle with the closest cost to the annual budget without going over in cell form
    Cost_vehicle_ca_model = car_model{cost_index_ca}; %Gives the vehicle model with the closest cost to the annual budget without going over in cell form
    Cost_ca.Vehicle = [Cost_vehicle_ca ' ' Cost_vehicle_ca_model]; %Gives the vehicle with the closest cost to the annual budget without going over in string form
    Cost_ca.Cost = annual_cost_ca(cost_index_ca); %Gives the annual cost
    Cost_ca.GHG = annual_GHG_emisions_ca(cost_index_ca); %Gives the annual green house gas emissions

    consumerStruct.Name = consumerName;
    consumerStruct.State = state;
    consumerStruct.GHG_Recommendation = GHG_ca;
    consumerStruct.Cost_Recommendation = Cost_ca;

elseif strcmpi(state, 'KS') %If the desired state is Kansas

    %Create the GHG_Recommendation Structure
    GHG_vehicle_ks = car_make{min_GHG_index_ks}; %Gives the vehicle with the minimum green house gas emissions in a cell form
    GHG_vehicle_ks_model = car_model{min_GHG_index_ks}; %Gives the vehicle model with the minimum green house gas emissions in a cell form
    GHG_ks.Vehicle = [GHG_vehicle_ks ' ' GHG_vehicle_ks_model]; %Gives the vehicle with the minimum green house gas emissions in a string form
    GHG_ks.Cost = annual_cost_ks(min_GHG_index_ks); %Gives the annual cost
    GHG_ks.GHG = annual_GHG_emisions_ks(min_GHG_index_ks); %Gives the annual green house gas emissions

    %Create the Cost_Recommendation Structure
    Cost_vehicle_ks = car_make{cost_index_ks}; %Gives the vehicle with the closest cost to the annual budget without going over in cell form
    Cost_vehicle_ks_model = car_model{cost_index_ks}; %Gives the vehicle model with the closest cost to the annual budget without going over in cell form
    Cost_ks.Vehicle = [Cost_vehicle_ks ' ' Cost_vehicle_ks_model]; %Gives the vehicle with the closest cost to the annual budget without going over in string form
    Cost_ks.Cost = annual_cost_ks(cost_index_ks); %Gives the annual cost
    Cost_ks.GHG = annual_GHG_emisions_ks(cost_index_ks); %Gives the annual green house gas emissions

    consumerStruct.Name = consumerName;
    consumerStruct.State = state;
    consumerStruct.GHG_Recommendation = GHG_ks;
    consumerStruct.Cost_Recommendation = Cost_ks;

elseif strcmpi(state, 'FL') %If the desired state is Florida

    %Create the GHG_Recommendation Structure
    GHG_vehicle_fl = car_make{min_GHG_index_fl}; %Gives the vehicle with the minimum green house gas emissions in a cell form
    GHG_vehicle_fl_model = car_model{min_GHG_index_fl}; %Gives the vehicle model with the minimum green house gas emissions in a cell form
    GHG_fl.Vehicle = [GHG_vehicle_fl ' ' GHG_vehicle_fl_model]; %Gives the vehicle with the minimum green house gas emissions in a string form
    GHG_fl.Cost = annual_cost_fl(min_GHG_index_fl); %Gives the annual cost
    GHG_fl.GHG = annual_GHG_emisions_fl(min_GHG_index_fl); %Gives the annual green house gas emissions

    %Create the Cost_Recommendation Structure
    Cost_vehicle_fl = car_make{cost_index_fl}; %Gives the vehicle with the closest cost to the annual budget without going over in cell form
    Cost_vehicle_fl_model = car_model{cost_index_fl}; %Gives the vehicle model with the closest cost to the annual budget without going over in cell form
    Cost_fl.Vehicle = [Cost_vehicle_fl ' ' Cost_vehicle_fl_model]; %Gives the vehicle with the closest cost to the annual budget without going over in string form
    Cost_fl.Cost = annual_cost_fl(cost_index_fl); %Gives the annual cost
    Cost_fl.GHG = annual_GHG_emisions_fl(cost_index_fl); %Gives the annual green house gas emissions

    consumerStruct.Name = consumerName;
    consumerStruct.State = state;
    consumerStruct.GHG_Recommendation = GHG_fl;
    consumerStruct.Cost_Recommendation = Cost_fl;

end
```

Published Test Case

```
consumerStruct1 = vehicleRecommendation('Brad', 'CA', 10000, 2000)
GHG_Recommendation1 = consumerStruct1.GHG_Recommendation
Cost_Recommendation1 = consumerStruct1.Cost_Recommendation
```

```
consumerStruct1 =
                  Name: 'Brad'
                 State: 'CA'
     GHG_Recommendation: [1x1 struct]
    Cost_Recommendation: [1x1 struct]
GHG_Recommendation1 =
    Vehicle: 'BMW i3'
       Cost: 2560
        GHG: 1370000
Cost_Recommendation1 =
    Vehicle: 'Toyota Tacoma'
       Cost: 1990
        GHG: 3765000
```

Additional Test Case

```
consumerStruct2 = vehicleRecommendation('Janet', 'KS', 15000, 6090)
GHG_Recommendation2 = consumerStruct2.GHG_Recommendation
Cost_Recommendation2 = consumerStruct2.Cost_Recommendation
consumerStruct3 = vehicleRecommendation('Stacy', 'FL', 18000, 7500)
GHG_Recommendation3 = consumerStruct3.GHG_Recommendation
Cost_Recommendation3 = consumerStruct3.Cost_Recommendation
consumerStruct4 = vehicleRecommendation('Tina', 'CA', 20000, 5000)
GHG_Recommendation4 = consumerStruct4.GHG_Recommendation
Cost_Recommendation4 = consumerStruct4.Cost_Recommendation
```

```
consumerStruct2 =
                  Name: 'Janet'
                 State: 'KS'
     GHG_Recommendation: [1x1 struct]
    Cost_Recommendation: [1x1 struct]
GHG_Recommendation2 =
    Vehicle: 'BMW i3'
       Cost: 3870
        GHG: 3105000
Cost_Recommendation2 =
    Vehicle: 'Tesla Model S'
       Cost: 6090
        GHG: 3490500
consumerStruct3 =
                  Name: 'Stacy'
                 State: 'FL'
     GHG_Recommendation: [1x1 struct]
    Cost_Recommendation: [1x1 struct]
GHG_Recommendation3 =
    Vehicle: 'BMW i3'
       Cost: 4626
        GHG: 3087000
Cost_Recommendation3 =
    Vehicle: 'Tesla Model S'
       Cost: 7.2900e+03
        GHG: 3394800
consumerStruct4 =
                  Name: 'Tina'
                 State: 'CA'
     GHG_Recommendation: [1x1 struct]
    Cost_Recommendation: [1x1 struct]
GHG_Recommendation4 =
    Vehicle: 'BMW i3'
       Cost: 5120
        GHG: 2740000
Cost_Recommendation4 =
    Vehicle: 'Honda Accord Hybrid'
       Cost: 4340
        GHG: 4328000
```

Question 4B

```
type vehicleComparison
```

```
function [comparison] = vehicleComparison(consumerStruct)

%Get Necessary Parameters

name = consumerStruct.Name; %Gets the consumer's name

GHG_cost = consumerStruct.GHG_Recommendation.Cost; %Gets the cost for the green house gas recommendation
GHG_GHG = consumerStruct.GHG_Recommendation.GHG; %Gets the green house gas emissions for the green house gas recommendation
GHG_vehicle = consumerStruct.GHG_Recommendation.Vehicle; %Gets the vehicle for the green house gas recommendation

cost_cost = consumerStruct.Cost_Recommendation.Cost; %Gets the cost for the cost recommendation
cost_GHG = consumerStruct.Cost_Recommendation.GHG; %Gets the green house gas emissions for the cost recommendation
cost_vehicle = consumerStruct.Cost_Recommendation.Vehicle; %Gets the vehicle for the cost recommendation

if (GHG_cost < cost_cost) && (GHG_GHG < cost_GHG) %If the green house gas recommendation is cheaper than the cost recommendation and emits less green house gases

    comparison = sprintf('The %s is the best option for %s because it costs $%.2f per year less and emits %.0f g CO2e per year less than the %s.',...
                         GHG_vehicle,name,cost_cost-GHG_cost,cost_GHG-GHG_GHG,cost_vehicle); %Returns the necessary string

elseif (GHG_cost > cost_cost) && (GHG_GHG > cost_GHG) %If the cost recommendation is cheaper than the green house gas recommendation and emits less green house gas

    comparison = sprintf('The %s is the best option for %s because it costs $%.2f per year less and emits %.0f g CO2e per year less than the %s.',...
                         cost_vehicle,name,GHG_cost-cost_cost,GHG_GHG-cost_GHG,GHG_vehicle); %Returns the necessary string

elseif (GHG_cost < cost_cost) %If the green house gas recommendation is only cheaper than the cost recommendation

    cost_difference = cost_cost - GHG_cost; %Calculates the cost difference
    GHG_difference = GHG_GHG - cost_GHG; %Calculates the green house gas emissions difference

    comparison = sprintf('The %s costs $%.2f per year less but emits %.0f g CO2e per year more than the %s.',...
```

```
                            GHG_vehicle,cost_difference,GHG_difference,cost_vehicle); %Returns the necessary string

    elseif (cost_cost < GHG_cost) %If the cost recommendation is only cheaper than the green house gas recommendation

        cost_difference = GHG_cost - cost_cost; %Calculates the cost difference
        GHG_difference = cost_GHG - GHG_GHG; %Calculates the green house gas emissions difference

        comparison = sprintf('The %s costs $%.2f per year less but emits %.0f g CO2e per year more than the %s.',...
                            cost_vehicle,cost_difference,GHG_difference,GHG_vehicle); %Returns the necessary string

    end

    end
```

Published Test Case

```
comparison1 = vehicleComparison(consumerStruct1)
```

```
comparison1 =
The Toyota Tacoma costs $570.00 per year less but emits 2395000 g CO2e per year more than the BMW i3.
```

Additional Test Case

```
comparison2 = vehicleComparison(consumerStruct2)

comparison3 = vehicleComparison(consumerStruct3)

comparison4 = vehicleComparison(consumerStruct4)
```

```
comparison2 =
The BMW i3 is the best option for Janet because it costs $2220.00 per year less and emits 385500 g CO2e per year less than the Tesla Model S.
comparison3 =
The BMW i3 is the best option for Stacy because it costs $2664.00 per year less and emits 307800 g CO2e per year less than the Tesla Model S.
comparison4 =
The Honda Accord Hybrid costs $780.00 per year less but emits 1588000 g CO2e per year more than the BMW i3.
```

Published with MATLAB® R2014a