Duke University

Edmund T. Pratt, Jr. School of Engineering

EGR 53L Fall 2009

Test I

Rebecca A. Simmons & Michael R. Gustafson II

Name and NET ID (please print)____

In keeping with the Community Standard, I have neither provided nor received any assistance on this test. I understand if it is later determined that I gave or received assistance, I will be brought before the Undergraduate Conduct Board and, if found responsible for academic dishonesty or academic contempt, fail the class. I also understand that I am not allowed to speak to anyone except the instructor about any aspect of this test until the instructor announces it is allowed. I understand if it is later determined that I did speak to another person about the test before the instructor said it was allowed, I will be brought before the Undergraduate Conduct Board and, if found responsible for academic dishonesty or academic contempt, fail the class.

Signature:____

Notes

- You will be turning in each problem in a separate pile. Make sure, then, that you do **not** put work for more than any one problem on any one piece pf paper. For this test, you will be turning in five different sets of work.
- Be sure your name *and* NET ID show up on *every page* of the test. If you are including work on extra sheets of paper, put your name and NET ID on each and be sure to staple them to the appropriate problem.
- This first page should have your name, NET ID, and signature on it. It should be stapled on top of your submission for Problem I.

Name (please print):
Community Standard (print ACPUB ID):

Problem I: [15 pts.] Basic Programming

- Write *all* the MATLAB statements required to generate a graph that shows three functions for height (in meters):

$$y_1(t) = e^t \qquad y_2(t) = t^2 \qquad y_3(t) = \ln(t)$$

for t between 1 and 2 seconds. Your graph should include 120 linearly spaced points between 1 and 2 for t. You should include axis labels, a title, and a reasonable legend placed as far out of the way as MATLAB can handle. You should make sure to use black lines but may choose the style. You should save the graph in a file called `ThreeLines.eps`. The script is started for you

```
clear; format short e; figure(1); clf
t = linspace(1, 2, 120);
plot(t, exp(t), 'k-', t, t.^2, 'k--', t, log(t), 'k:')
legend('e^t', 't^2', 'ln(t)', 0)
xlabel('Time (s)')
ylabel('Height (m)')
title('Height vs. Time for Three Functions (NET ID)')
print -deps ThreeLines.eps
```

- Write a *function* `.m` file called `GeoMean.m` to calculate the *geometric mean* of two numbers:

$$\text{Geometric Mean of } a \text{ and } b = \sqrt{a \cdot b}$$

Your function should check to make sure the user called the functions with two inputs and give an error otherwise. Similarly, your program should make sure the two inputs are the same size and give an error if not. In cases where the user correctly enters two similar-sized matrices, your function should return geometric means of corresponding values in matrices.

```
function MeanOut = GeoMean(a, b)
if nargin<2
    error('Not enough inputs!')
end
if ~(size(a)==size(b))
    error('Not the same size!')
end
MeanOut = sqrt(a.*b)
```

Now write a *script* `.m` file that does all the work necessary to calculate the geometric means of $\cos(x)$ and $\sin(x)$ for values of x between 0 and $\pi/2$ separated by $\pi/14$. The script *must* rely on the function above to perform the calculations. The resulting matrix should be called `Quad1Means`. The script is started for you:

```
clear; format short e;
clear; format short e
x = 0:pi/14:pi/2
Quad1Means = GeoMean(cos(x), sin(x))
```

Name (please print):
Community Standard (print ACPUB ID):

Problem II: [20 pts.] Go 'Canes! (oops..the other kind)

Tropical depressions, tropical storms, and hurricanes are categorized based on wind speeds and the possible storm surge. The following table shows the category name for storms with varying wind ranges and what the storm surge is (in feet above normal):

Category	Wind Ranges (mph)	Storm Surge (ft)
Tropical Depression	0-39	0
Tropical Storm	39-74	0-4
Hurricane	> 74	> 4

Write a script, `StormCat.m`, that will first ask the user to enter either the word `wind` or `surge` - should the user fail to enter a proper response, your program should keep asking until a valid entry is received.

Next the program should ask for a number by using an appropriate prompt - that is to say, the prompt should be tailored to how the user answered the first question:

```
Please enter a wind speed:
```

or

```
Please enter a surge level:
```

You may assume that the user properly enters a single positive number here.

Finally, the program should determine if the value the user entered is a tropical depression, a storm, or a hurricane. It should then print out a statement accordingly. For example, if the user enters `wind` and `100`, the program should print out

```
That indicates a hurricane.
```

while a user entering `surge` and `0.1` would receive:

```
That indicates a tropical storm.
```

```
clear; format short e
Type = input('Data type (wind or surge): ', 's');
while ~(strcmp(Type, 'wind') | strcmp(Type, 'surge'))
    fprintf('Please enter either ''wind'' or ''surge''\n')
    Type = input('Data type (wind or surge): ', 's');
end

Wind = 0;
Surge = 0;

if strcmp(Type, 'wind')
    Wind = input('Please enter a wind speed: ');
else
    Surge = input('Please enter a surge level: ');
end

if Wind>74 | Surge>4
    fprintf('That indicates a hurricane.\n')
elseif Wind>39 | Surge>0
    fprintf('That indicates a tropical storm.\n')
else
    fprintf('That indicates a tropical depression.\n')
end
```

Name (please print):
Community Standard (print ACPUB ID):

Problem III: [20 pts.] Imma let you finish, but the colon operator is the best operator of all time!

For each of the following sections, show what the matrices created or modified in each block will look like at the end of the snippet of code.

(a)
```
>>A=linspace(-2, 3, 6)
>>B=logspace(-2, 3, 6)
>>C=-2:3:6
```

```
A = [-2 -1  0  1  2  3]
B = [0.01 0.10 1.00 10.00 100.00 1000.00]
C = [-2 1 4]
```

(b)
```
>>D=[1.6 2.4 -8.7 -11.2]
>>E=ceil(D)
>>F=fix(D)
>>G=round(D)
```

```
D = [ 1.6   2.4  -8.7  -11.2]
E = [ 2     3    -8    -11  ]
F = [ 1     2    -8    -11  ]
G = [ 2     2    -9    -11  ]
```

(c)
```
>>H=[6 4 -3; 8 -9 5]
>>I=sum(H)
>>J=min(H)
>>K=I.*J
```

```
H = [ 6   4  -3
      8  -9   5]
I = [14  -5   2]
J = [ 6  -9  -3]
K = [84  45  -6]
```

(d)
```
>>clear
>>L(2, 5) = 10
>>M = [1:5; 6:10]
>>N = M([2 1 2], [1:2:end])
```

```
L = [ 0   0   0   0   0    M = [ 1   2   3   4   5     N = [ 6   8  10
      0   0   0   0  10]         6   7   8   9  10]          1   3   5
                                                             6   8  10]
```

(e)
```
>>Oh = eye(2)
>>P = [Oh 2*Oh(end:-1:1,:)]
```

```
Oh = [ 1  0    P = [ 1  0  0  2
       0  1]         0  1  2  0]
```

Name (please print):
Community Standard (print ACPUB ID):

Problem IV: [20 pts.] The One You Knew Was Coming

A digital thermometer will convert a temperature into a voltage. For the specific thermometer you will be considering, the relationship between the temperature and the voltage is as follows:

$$V(T) = \begin{cases} 0 & T \leq 32^\circ F \\ 5\left(\frac{T-32}{90}\right)^2 & 32^\circ F \leq T \leq 122^\circ F \\ 10 - 5\left(\frac{212-T}{90}\right)^2 & 122^\circ F \leq T \leq 212^\circ F \\ 10 & T \geq 212^\circ F \end{cases}$$

Write code that will do each of the following tasks:

- Create an anonymous function called `Voltage` that accepts a matrix of temperatures and returns the corresponding voltages for the thermometer.
- Generate 200 linearly spaced points between 0 and 300 for the temperatures and stores them in a variable with a reasonable name.
- Plot the voltage versus temperature for those values using a solid black line. You should add an appropriate title and axis labels to this plot. You do not need to save the plot.
- Finally, generate a 7x4 matrix of random integers between 32 and 212. Assuming these are temperatures, calculate the overall maximum, minimum, and average voltages for the thermometer based on those random values. Give these calculations meaningful names such that a person reading your code would understand what they contain.

The script is started for you:

```
clear; format short e; figure(1); clf;

Voltage = @(TD) ...
    (TD <= 32)              .* (0) + ...
    ( 32 < TD & TD <= 122) .* (5*((TD-32)/90).^2) + ...
    (122 < TD & TD <= 212) .* (10-5*((212-TD)/90).^2) + ...
    (TD > 212)              .* (10);

T = linspace(0, 300, 200);
plot(T, Voltage(T), 'k-')
xlabel('Temperature (deg. F)')
ylabel('Voltage (V)')
title('Voltage vs. Temperature for a Thermometer')

RandTemps = 32 + floor(181*rand(7, 4));
RandVolts = Voltage(RandTemps)

MinVoltage =  min(RandVolts(:))
MaxVoltage =  max(RandVolts(:))
AvgVoltage = mean(RandVolts(:))
```

Name (please print):
Community Standard (print ACPUB ID):

Problem V: [25 pts.] Takin' What They Givin'

The Green Fuel Cells Corporation has all of its weekly employee data stored in a single text file. The file is set up in one column such that the first entry is the first employee's ID, the second entry is the first employee's pay per hour, and the third entry is the first employee's number of hours for that week. Each employee similarly has three entries in the column. You are going to write a script to process the payroll for this week, which is in a file called `EData38.prl`. Your script must do the following:

- Load the data and create three matrices - one containing the employee IDs, one containing the employee pay rates, and one containing the employee hours for the week. Use an efficient method to do this and give the matrices sensible names.
- Calculate a matrix called `WeekPay` for the amount to be paid to each employee. This should be the same shape as the previously created matrices.
- Print out a formatted table with the employee ID, rate, hours worked, and amount paid for the week. Employee IDs are six-digit integers. You may assume that employees make between \$8.50 and \$32.50 per hour and work no more than 110 hours per week. Rates and salaries should be printed to the nearest cent; hours may be printed to the nearest tenth of an hour. Make sure the decimal points will line up.
- Determine the minimum, average, and maximum amount paid to an employee and print out a formatted table with this information. Make sure the decimal points line up.
- Print out a statement for the total amount paid and the total number of employees.
- Calculate and print out a statement about how many employees worked overtime (that is, how many employees worked more than 40 hours in a week). For grammar purposes, you may assume that you always have at least 2 overtime employees (and thus, always at least two employees as well).

For example - if `EData38.prl` contained the following:

```
513882
 14.19
103.20
129540
 12.99
 20.00
680586
  9.32
 27.40
223276
 20.10
 63.30
201222
 21.28
 43.40
```

here is the output based on that file:

```
    ID  Rate Hours  Salary
513882 14.19 103.2 1464.41
129540 12.99  20.0  259.80
680586  9.32  27.4  255.37
223276 20.10  63.3 1272.33
201222 21.28  43.4  923.55

Minimum Salary:  255.37
Average Salary:  835.09
Maximum Salary: 1464.41

Totals: 4175.46 paid to 5 employees
Includes 3 overtime employees
```

```
%% Prepare the workspace
clear; format short e

%% Load the data
load EData38.prl

%% Split the data into IDs, PayRate, and Hours
IDs     =  EData38(1:3:end)
PayRate =  EData38(2:3:end)
Hours   =  EData38(3:3:end)

%% Calculate Salary
WeekPay = PayRate .* Hours

%% Print table
fprintf('    ID  Rate Hours  Salary\n')
for k=1:length(IDs)
    fprintf('%6d %5.2f %5.1f %7.2f\n', IDs(k), PayRate(k), ...
        Hours(k), WeekPay(k))
end
fprintf('\n')

%% Calculate and print salary data
MinSal =  min(WeekPay);
AvgSal = mean(WeekPay);
MaxSal =  max(WeekPay);
fprintf('Minimum Salary: %7.2f\n', MinSal)
fprintf('Average Salary: %7.2f\n', AvgSal)
fprintf('Maximum Salary: %7.2f\n', MaxSal)

fprintf('\n')

fprintf('Totals: %0.2f paid to %d employees\n', sum(WeekPay), length(IDs))
OTE = find(Hours>40);
fprintf('Includes %d overtime employees\n\n', length(OTE))
```