Duke University
Edmund T. Pratt, Jr. School of Engineering

EGR 53L Spring 2005

Test II

Michael R. Gustafson II

Name (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 Judicial 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 Judicial Board and, if found responsible for academic dishonesty or academic contempt, fail the class.

Signature:_____

Problem I: [15 pts.] Validating Inputs

An electrical engineer is writing code to help quickly determine the equivalent resistance of two resistors that are either in series or in parallel. To do this, the engineer has written a program that will ask the user for a resistance, a word, and another resistance, as shown below:

```
1stRes = imput(‘‘First resistance: ’’);
MyWord = imput(‘‘Connection: series or parallel: ’’);
2ndRes = imput(‘‘Second resistance: ’’)
```

The problem is, this code has some flaws. Furthermore, this program is going to be used by other people so the code needs some input validation. All resistances entered must be greater than zero, and the only words that are valid are `series` and `parallel`. Re-write the code so that when it is complete, Matlab knows two proper values for the resistances and has one of the valid words in `MyWord`.

```
Res1=0
while Res1<=0
    Res1=input('First resistance: ');
end
MyWord='blah'
while ~(strcmp(MyWord, 'series') | strcmp(MyWord, 'parallel'))
    MyWord=input('Connection: series or parallel: ', 's');
end
Res2=0
while Res2<=0
    Res2=input('Second resistance: ');
end
```

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

Problem II: [20 pts.] Formatted Output and Files

An engineer is using Matlab in order to produce a text file containing a trig table for common angles. When the script is done, the output file should contain the values of cosine, sine, and tan for angles 0-355 degrees in increments of 5 degrees. The table needs to be properly aligned and should have four digits after the decimal point for the trig columns.

In creating the table, if there are angles for which cosine, sine, or tangent are infinite, the program should print the word "Infinity" in the table instead of the value Matlab returns. The portion of the table from 45 degrees to 100 degrees is shown below with 'X's on the first line to show you spaces for that line (the 'X's are **not** a part of the table, just there to help you with spacing):

```
X45XX0.7071XX0.7071XXXX1.0000
 50  0.6428  0.7660    1.1918
 55  0.5736  0.8192    1.4281
 60  0.5000  0.8660    1.7321
 65  0.4226  0.9063    2.1445
 70  0.3420  0.9397    2.7475
 75  0.2588  0.9659    3.7321
 80  0.1736  0.9848    5.6713
 85  0.0872  0.9962   11.4301
 90  0.0000  1.0000  Infinity
 95 -0.0872  0.9962  -11.4301
100 -0.1736  0.9848   -5.6713
```

The table should be saved to a file called `MyTable.out`. For this program, indicate any spaces in your printing commands with the hat character ^.

```
fid = fopen('MyTable.out', 'w')
for k=0:5:355
    angle = k * pi / 180;
    fprintf(fid, '%3.0f^%7.4f^%7.4f', k, cos(angle), sin(angle))
    if k==90 | k==270
        fprintf('^^Infinity\n')
    else
        fprintf(fid, '^%9.4f\n', tan(angle))
    end
end
fclose(fid)

% Note, for Matlab 7 instead of converting the
% angle to radians, cosd, sind, and tand may be used
% In this case, the question tand(k)==inf is allowed
```

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

Problem III: [25 pts.] Root Finding

(1) Given some function $y(x) = x^3 - 19x - 30$, give the command or commands you would use in Matlab to find the roots of this polynomial:

roots([1 0 -19 -30]) %OR
y=inline('x.^3-19*x-30', 'x'); fzero(y, 10) %OR
fzero('x.^3-19*x-30', 10)

(2) Assuming some function $f(x)$ and some guess x , write the equation that models how Newton's method determines the next guess for the root:

$$x_{+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

(3) Demonstrate your ability to find a root using Newton's Method by finding a root of $y(x) = x^3 - 19x - 30$. Use a starting value of 10, a function tolerance of 0.1, an x tolerance of 0.001, and a maximum number of iterations of 6. Explain what value you found for the root and which of the three conditions above made you stop. Also indicate the total number of iterations, the final x tolerance, and the final f tolerance. Finally, state what value or values for an initial guess would have been disastrous and why.

f(x) = x^3 - 19 x - 30
f'(x) = 3x^2 - 19

k	xk	f(xk)	f'(xk)	minus f/f'	xk+1
1	10.00000	780.00000	281.00000	-2.77580	7.22420
2	7.22420	209.76435	137.56717	-1.52481	5.69939
3	5.69939	46.84476	78.44898	-0.59714	5.10225
4	5.10225	5.88381	59.09882	-0.09956	5.00269
5	5.00269	0.15073	56.08071	-0.00269	5.00000
6	5.00000	0.00011			

Root at x=5.000
Stopped after 5 iterations due to f tolerance on line 6 of .00011
Final x tolerance therefore .00269 from line 5
Bad initial guess is wherever f' = 0, so (3x^2-19)=0 or f=+/- sqrt(19/3)

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

Problem IV: [20 pts.] Loops

(1) Show the output for the following Matlab code:

```
for k=2:4.5
     for n=k:2:5
          fprintf('%0.0f %0.0f\n', k, n);
     end
end
```

2 2
2 4
3 3
3 5
4 4

(2) Show the output for the following Matlab code:

```
k=1;
Count1=k;
while k<4 | Count1<5
     k=k+Count1;
     Count1=Count1+1;
     fprintf('%0.0f %0.0f\n', k, Count1);
end
```

2 2
4 3
7 4
11 5

(3) Show the output for the following Matlab code:

```
Count2=0;
for m=[3 1 4 2:0]
     Count2=Count2+m;
     fprintf('%0.0f %0.0f\n', m, Count2);
end
```

3 3
1 4
4 8

(4) Show the output for the following Matlab code:

```
for p=1:4:18
     Count3=0;
     q=10-p;
     while q<7
          q=q+3;
          Count3 = Count3+1;
     end
     fprintf('p:%0.0f q:%0.0f c3:%0.0f\n', p, q, Count3);
end
fprintf('P:%0.0f Q:%0.0f C3:%0.0f\n', p, q, Count3);
```

p:1 q:9 c3:0
p:5 q:8 c3:1
p:9 q:7 c3:2
p:13 q:9 c3:4
p:17 q:8 c3:5
P:17 Q:8 C3:5

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

Problem V: [20 pts.] Overloaded Functions

An engineer is running experiments on several different metal samples. At the end of the experiment, a quality ranking - an integer between 0 and 10 - is assigned to the sample. Your task is to write a program that will help analyze the experimental data.

You need to write a function called `Analyzer.m` that will accept up to three arguments. The first argument should be an NxM matrix of quality rankings. If there is only one argument, the function should return the total number of quality rankings greater than 0. If there is a second argument, the function should return the number of quality rankings equal to that second argument. Finally, if there is a third argument, the function should return the number of quality rankings between the second and third argument (inclusive). Note that if the user calls the function with no arguments, the program should state `No Inputs` and set the output to zero. A sample diary is below, while the first line of the function is beneath the diary.

```
>> A = [1 2 0; 0 3 4; 4 5 6];
>> Count1 = Analyzer(A)
Count1 =
         7
>> Count2 = Analyzer(A, 4)
Count2 =
         2
>> Count3 = Analyzer(A, 3, 5)
Count3 =
         4
>> Count4 = Analyzer
No Inputs
Count4 =
         0
```

```
function CountOut = Analyzer(QR, X, Y)
if nargin==0
   fprintf('No Inputs\n')
   CountOut = 0;
elseif nargin==1
   CountOut = sum(sum(QR>0));
elseif nargin==2
   CountOut = sum(sum(QR==X));
else
   CountOut = sum(sum(QR>=X & QR<=Y));
end

% Note: instead of sum(sum()),
% length(find()) may be used
% length() does not work alone, however
```