Taylor Rembos
Lab 0 Part 1
EEL 3135
Due May 13 by 11:59pm

Lab 0 Part 1

Section 1.1

MATLAB Code:

```
>> %
>> % 1.1 a
>> %
>> 6^2

ans =

    36

>> ans

ans =

    36

>> ans/6

ans =

     6

>> ans

ans =

     6

>> % the variable ans stores the value of the last calculated output
>> %
>> % 1.1 b
>> %
>> pi*pi - 10

ans =

   -0.1304
```

```
>> sin(pi/4)

ans =

   0.7071

>> ans^2

ans =

   0.5000

>> % MATLAB is being demonstrated here as a "very expensive calculator"
>> % It takes the input from the user, calculates the answer, and stores that value in ans variable
>> x = sin(pi/5);
>> cos(pi/5)

ans =

   0.8090

>> y = sqrt(1 - x*x)

y =

   0.8090

>> ans

ans =

   0.8090

>> % the numerical value of x, as shown in the workspace, is x = 0.5878
```

Section 1.2

```
>> %
>> % 1.2 a
>> %
>> jkl = 0:6

jkl =
```

```
     0     1     2     3     4     5     6

>> % the variable jkl is assigned the array of values ranging from 0 to 6
>> jkl = 2:4:17

jkl =

     2     6    10    14

>> % the variable jkl is assigned the array of values ranging from 2 to 17 in increments of 4
>> jkl = 99:-1:88

jkl =

  Columns 1 through 11

    99    98    97    96    95    94    93    92    91    90    89

  Column 12

    88

>> % the variable jkl is assigned the vector of values ranging from 99 to 88, decresing by 1
>> ttt = 2:(1/9):4

ttt =

  Columns 1 through 6

    2.0000    2.1111    2.2222    2.3333    2.4444    2.5556

  Columns 7 through 12

    2.6667    2.7778    2.8889    3.0000    3.1111    3.2222

  Columns 13 through 18

    3.3333    3.4444    3.5556    3.6667    3.7778    3.8889

  Column 19

    4.0000
```

```
>> % the variable ttt is assigned the vector of values ranging from 2 to 4 in
increments of (1/9)
>> tpi = pi*[0:0.1:2];
>> % the varible tpi is assigned the value of pi multiplied by the vector
>> % ranging from 0 to 2 in increments of 0.1
>> %
```

>> % 1.2b

```
>> %
>> xx = [zeros(1,3), linspace(0,1,5), ones(1,4)]

xx =

  Columns 1 through 6

       0         0         0         0    0.2500    0.5000

  Columns 7 through 12

    0.7500    1.0000    1.0000    1.0000    1.0000    1.0000

>> xx(4:6)

ans =

       0    0.2500    0.5000

>> size(xx)

ans =

     1    12

>> length(xx)

ans =

    12

>> xx(2:2:length(xx))

ans =

       0         0    0.5000    1.0000    1.0000    1.0000

>> % line 2 returns points 4 to 6 in the vector of variable xx
```

```
>> % line 3 returns the size of the list of variable xx
>> % as a two element row vector
>> % line 4 returns the length of the largest array of varible xx
>> % line 5 returns, in regards to the vector of variable xx, the values of the
>> % second point to the length of xx, in this case 12, in increments of 2
>> %
```

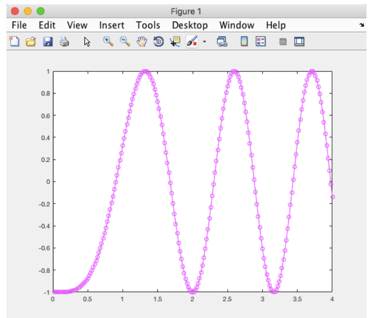
>> %1.2 c

```
>> %
>> yy = xx;
>> % the variable yy is assigned the value of varible xx
>> yy(4:6) = pi*(1:3)

yy =

  Columns 1 through 6

       0         0         0    3.1416    6.2832    9.4248

  Columns 7 through 12

    0.7500    1.0000    1.0000    1.0000    1.0000    1.0000

>> % elements 4 to 6 in yy are assigned the value of pi multiplied by
>> % elements 1 to 3
>> xx(2:2:length(xx)) = pi^pi

xx =

  Columns 1 through 6

       0   36.4622         0   36.4622    0.2500   36.4622

  Columns 7 through 12

    0.7500   36.4622    1.0000   36.4622    1.0000   36.4622

>> % this code takes the even elements of vector xx and replaces them with pi^pi
>> %
```

>> % 1.2 d

```
>> %
>> % the dot before the asterisk distinguishes array operations from matrix
operations
>> % condensing the example results in
>> N = 200;
>> sig = exp(j*2*pi*sqrt((((1:N)./50).*((1:N)./50)) + 2.25));
```

```
>> plot((1:N)/50, real(sig), 'mo-')
```




```
>> %
>> xk = cos(pi*(0:11)/4)

xk =

  Columns 1 through 6

    1.0000    0.7071    0.0000   -0.7071   -1.0000   -0.7071

  Columns 7 through 12

   -0.0000    0.7071    1.0000    0.7071    0.0000   -0.7071

>> % this command stores the values of 0 to 11 each multiplied by pi
>> % and divided by 4, and takes the cosine of that value and stores the vector in xk
>> xk(1)

ans =

     1

>> % xk(1) is the value 1. cos(pi*1/4) is 1
>> xk(0)
Subscript indices must either be real positive integers or
logicals.
>> % xk(0) is not defined because it is not a real positive integer
>> %
```

>> % 1.2 e

```
>> %
>> yy = [ ];
>> yy((-5:5) + 6) = cos((-5:5).*pi/3)

yy =

  Columns 1 through 6
```

0.5000   -0.5000   -1.0000   -0.5000   0.5000   1.0000

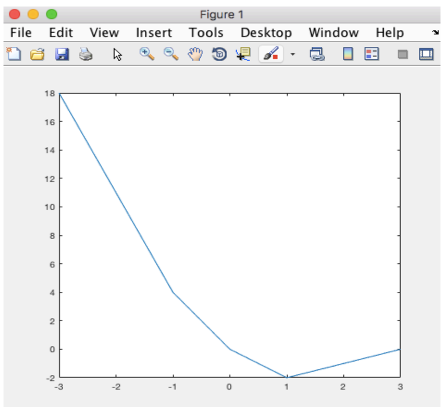
Columns 7 through 11

0.5000   -0.5000   -1.0000   -0.5000   0.5000

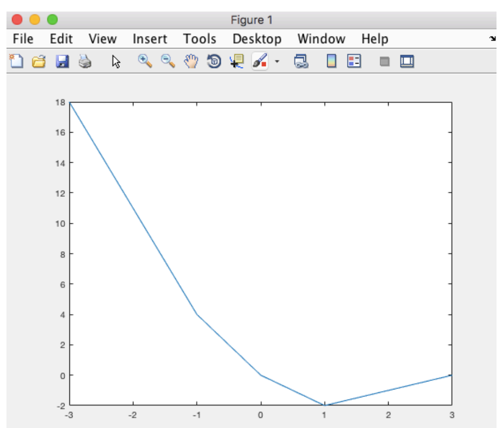
```
>> % the original code uses yy(k+6) in order to start from yy(1)
>> % if we used yy(k), it would start with yy(-5) which does not exist
>> yy(-5)
Subscript indices must either be real positive integers or
logicals.
>>%
```

>> % 1.2 f

```
>>%
>> x = [ -3 -1 0 1 3];
>> y = x.*x - 3*x;
>> plot(x, y)
```




```
>> z = x + y * sqrt(-1)

z =

  Columns 1 through 3

  -3.0000 +18.0000i  -1.0000 + 4.0000i   0.0000 + 0.0000i

  Columns 4 through 5

   1.0000 - 2.0000i   3.0000 + 0.0000i

>> plot(z)
```



>> % the dot product generates the dot product of the vectors at its inputs
>> % the scalar output is equal to y = sum(conj(u1) .* u2) where u1 and u2
>> %, the block's top and bottom inputs, can be vectors, column vectors, or scalars.
>> %
>> % matrix multiplication in matlab is C = A*B, the matrix product of A and B where
>> % C(i,j) is the inner product of the ith row of A with the jth column of B
>> % this operator is defined in matlab as C(i,j) = A(i,:)*B(:,j)
>> %
>> % Matrix multiplication follows the rules of linear algebra while the dot product array operations
>> % execute element by element and supports multidimentional arrays
>>%

>> % 1.2 g

>>%

function oddsummer.m

```
function oddsummer = oddsummer(n)
num = mod(n,2); % check even or odd
    if num == 0 % even number
        n = n - 1; % subtract 1 from even num to get odd num
    end
    oddsummer = 0;
    while n > 0 % if num positive integer, add all
        oddsummer = oddsummer + n; % add all to n
        n = n - 2; % subtract 2 for odd number
    end
end
```

```
>> oddsummer(1)

ans =

     1

>> oddsummer(3)

ans =

     4

>> oddsummer(5)

ans =

     9

>> oddsummer(8)
```

ans =

16

>> oddsummer(0)

ans =

0
>>%

>> % 1.2 h

>>%

function hellos.m

```
function hellos = hellos(h)

while(h>0) % makes sure h is positive integer
    disp('hello'); % displays the world hello
    h = h-1; % subtract 1 to account for 0 place
end
end
```

>> hellos(0)
>> hellos(1)
hello
>> hellos(2)
hello
hello
>> hellos(3)
hello
hello
hello
>> hellos(7)
hello
hello
hello
hello
hello
hello
hello