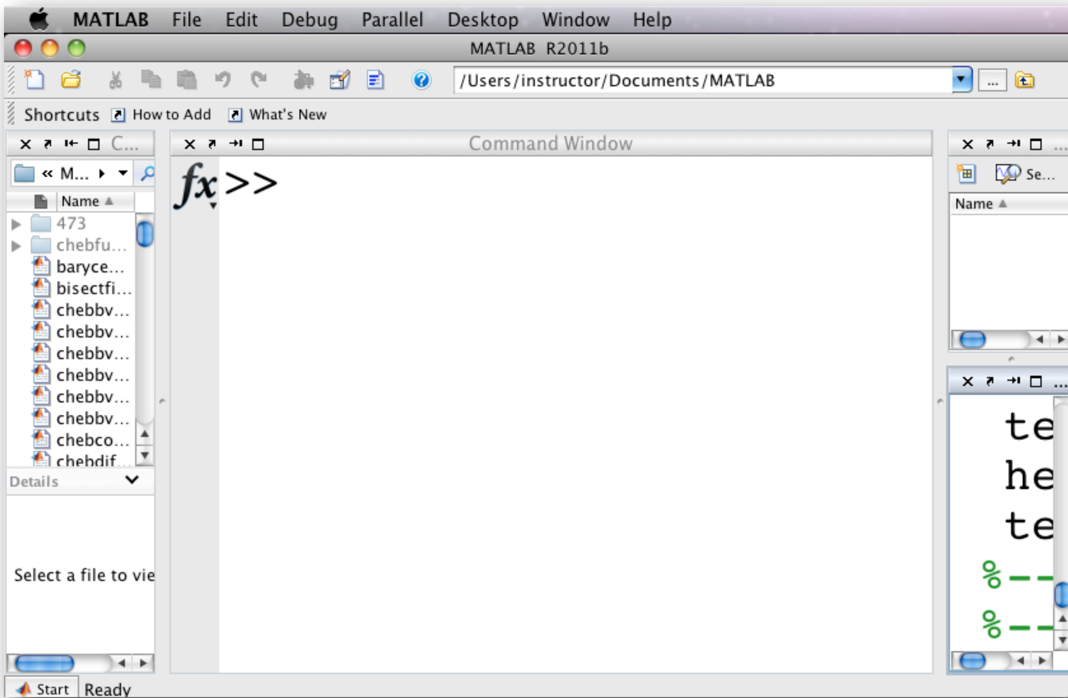
CSUMS Worksheet 01: MATLAB or Octave as a Calculator Version 1 Summer 2013

The following figure shows MATLAB when it first starts up.



Here is an edited log of a demo showing how to use the basic features MATLAB (or Octave)

Enter an arithmetic expression:
```
>> 2*(4+3)^2
```

To see the result DON'T end statements with ';'.
```
>> 2*(4+3)^2;  % No result will be displayed!
```

Use the "Up Arrow Key" **to redisplay the last line entered:**
```
>> 2*(4+3)^2;
```

Use "`format long`" to display 15 digits:
```
>> format long
>> pi
```

Use "`format short`" to display 4 digits:
```
>> format short
>> pi
```

Use "`format short e`" to display 4 digit scientific notation:
```
>> format short e
>> pi
>> 123456789
```

Use "`format long e`" to display 15 digit scientific notation:
```
>> format long e
>> 1234567890/11
```

Use "`format short g`" to display using the least space:
```
>> format short g
>> 1234567890/11 % Note the division operator '/'
```

Use "`format long g`" to display using the least space:
```
>> format long g
>> 1234567890/11 % Note the division operator '/'
```

Standard functions: `sqrt(x), exp(x), abs(x), log(x), log10(x), factorial(x).`

Trig functions: `sin(x), cos(x), tan(x), cot(x).`

Other functions: `round(x), fix(x), ceil(x), floor(x), rem(x,y), mod(x,y), sign(x).`

For `n!` **use:**
```
>> factorial(100)
```

The "`floor`" function:
```
>> floor(5/2)
>> floor(-5/2)
```

The "`mod`" function:
```
>> mod( 27, 4)
>> mod(-27, 4)
```

Scaler variables:
```
>> x = 25
>> x = 3*x - 10
>> a = 15
>> B = 10
>> C = (a + B)*(a - B)+1
```

How are the next two different?
```
>> a = 1, B = 2;
>> a = 1; B = 2;
```

Remember trig function arguments are in radians!
```
>> theta = 0.85;
>> E = sin(theta)^2 + cos(theta)^2
```

Predefined variables: `ans(= previous), pi, eps(= smallest delta), inf, i(or j), NaN.`
```
>> pi
>> eps
>> inf
>> NaN
>> i
```

Useful commands: `who, whos, clear, clear x, y, z`
To see the currently defined objects, enter
```
>> who
```
or
```
>>whos
```

To remove the global variables '`a`**' and '**`B`**', enter**
```
>> clear a, B
>> who
```

To remove all previously defined global variables, enter
```
>> clear
>> who
```

For help, enter
```
>> help
```

Part 2: Creating vectors and arrays

Both MATLAB and Octave and are designed to be array calculators!
Moreover, they are designed to work with data

Representation of population data:
```
>> format short;
```

Use "," or space to create a row vector:
```
>> yrs = [1984, 1986, 1988, 1990, 1992, 1994, 1996]
```

Here is a better way to create the `yrs` **vector!**
```
>> yrs = [1984: 2: 1996]
```

Use ";" to create a column vector.
```
>> pop = [127; 130; 136; 145; 158; 178; 211]
```

Use spaces to create a row vector:
```
>> pt = [1 2 3]  %A row vector
```

Also, You can use RETURN to create a column vector:
```
>> ptV = [1   %  Press Return after entering each coordinate.
 2
 3
] % Type "]" to end the vector
```

The general form to create a row vector is `vName = [startValue: increment: endValue]`
BUT increment defaults to 1 if omitted.
```
>> x = [-5: 5]
```

However, the square brackets can be omitted for vectors!
```
>> x = -5: 5
```

The increment can be negative; hence, it can be used to decrement!
```
>> x = [10: -1 :0]
```

The vector yrs **can also generate by using the number of values:**
```
>> yrs = linspace(1984,  1996,  7) %Note the commas!
>> x = linspace(-1,1, 10)
```

Use v(index) **to access a vector's coordinates:**
```
>> x(1)
```

Note that the indexing of vectors starts at 1 NOT zero!
```
>> x(0)  % Will cause an error!
>> x(3)
>> x(11) % Will cause an error!
```

Creating arrays (matrices).
Use ';' to start a new row. Each must have the same length!
```
>> a = [1 2; 3 4]
>> a =[1 2 3; 4 5 6; 7 8 9]
>> a =[1 2; 3 4; 5 6]
>> a =[1 2 3; 4 5 6]
>> a = 1; b= 10; c = 100;
>> mat = [a b c; a + b + c, a - b - c, a*b*c ]
>> mat = [linspace(1, 5, 4); linspace(10, 15, 4)]
```

Predefined array constructors: zeros, ones, eye (a bad pun)
```
>> zeros(2,3)
>> u = ones(4,4)
>> Identity3by3 = eye(3,3)
```

Use a single quote ' **to transpose an array (matrix)**
```
>> aa = [1 2 3]
>> bb = aa'
>> aa = [1 2; 3 4; 5 6]
>> bb = aa'
```

Accessing vector and array entries:
```
>> v = [1 2 3]
>> v(1)
>> v(2)
>> v(4) % Another invalid index
>> v(1) = 10

>> v(1)*v(3) + v(2)
>> mat
>> mat(1,1)
>> mat(1,2)
>> mat(1,5)
>> mat(1,1) + mat(2,2)
```

```
>> v = linspace(1,100,100); % <= Note ‘; ‘ is used to suppress output!
>> v(2:6) % => The entries v(2) to v(6)
>> a = [1 2 3; 4 5 6; 7 8 9]
>> a(:, 1)       % First column
>> a(1, :)       % First row
>> a(:, 1: 2)    % columns 1 and 2
>> a(1: 2, :)    % rows 1 and 2
>> a(2: 3,:)     % rows 2 and 3
>> a(2: 3, 1: 2) % submatrix of a
>> v = 1: 2: 5
>> v = [1: 2: 5] % same result!
>> v = 1: 100;
>> u = v([3, 5, 7: 10])  % entry selection via a vector
>> a = [10: −1: 4; ones(1, 7); zeros(1, 7)] % mat construction
```

Changing vector and arrays:

```
>> v = [1 2 3 4]
>> v(5: 10) = 10: 5: 35 % Appends entries to a vector
>> mat = [1 2 3 4; 5 6 7 8]
>> mat(3, :)=[10: 4: 22] % Appends a third row
>> mat(1, :)=[] % Removes the first row
>> v=1: 100;
>> v(2: 99)=[]  % Removes entries 2 through 99
>> length(v)
>> length(mat)
>> size(mat)
>> diag(v)    % Create a diagonal matrix from ‘v’
>> diag(mat)  % Create a column vector from the diagonal entries of ‘mat’
>> mat
>> mat =[1 2 3 4 5 6; 7 8 9 10 11 12; 13 14 15 16 17 18]
>> reshape(mat, 2, 9)
>> reshape(mat, 9, 2)
```

Standard Operations on matrices

1. The Determinant

```
>> mat = diag([1 2 3])
>> det(mat)
```

2. Sum and Difference

```
>> v = [1 2 3]; w = [4 5 6];
>> v + w
>> v – w
```

3. Matrix Multiplication

```
>> v*(w') % = v * transpose(w)
>> v'*w   % = transpose(v) * w
>> a = [1 2; 3 4; 5 6]
>> a*[1 1]'  % = a* transpose([1 1])
>> v
```

```
>> v*a % = (1 X 3)*(3 X 2) = 1 X 2
>> a*v % = (3 X 2)*(1 X 3) ==> ERROR
```

4. The Inverse of a Square Matrix

```
>> a = [1 2; 3 4]
>> inv(a); % = inverse of a square matrix
>> ans
>> a*ans % = 2 X 2 identity
>> a = [1 2 3; 0 4 5; 0 0 6]
>> inv(a)
>> a*inv(a)
>> inv(a)*a
>> a*inv(a) - inv(a)*a % Note: This should be all zeros!
```

5. Finding the Solution of a Linear system of Equations

```
     \ <=> left division by a matrix,
     / <=> right division by a matrix
```

Here are two methods for finding the solution of

```
      4x -  2y + 6z = 8
       2x +  8y + 2z = 4
       6x + 10y + 3z = 0
```

Method 1: Use `AX = B => X = A^(-1)B` **(left division by A)**

Method 2: Use `X^t A^t = B^t => X^t = B^t(A^t)^(-1)` **(right division by A^t)**

```
>> A=[4 -2 6; 2 8 2; 6 10 3]
>> B = [8 4 0]'

>> X = A\B  % left division by A is the same as
>> X = inv(A)*B

>> X = B'/A' % right division by A' is the same as
>> X = B'*inv(A')
```

Random matrices can be generated!

```
>> b = rand(3, 3)
```

Elementwise operators “.*”, ".", ".^" **NOTE THE PERIODS!**

```
>> a = [1 2 3]
>> b = [2 4 8]
```

1. Elementwise multiplication

```
>> a.*b
```

2. Elementwise division

```
>> a./b
>> a.\b
```

3. Elementwise power

```
> > a.^b
```

Vector-matrix functions: `mean, max, min, median , sum, std, sort.`

```
>> a = [1 2 3; 0 4 5; 0 0 6], mean(mat)
>> max(a), min(a), sum(a)
>> median(a)
>> a = linspace(1, 100, 100);
>> sum(a)
>> median(a)
>> max(a)
>> std(a)
>> a = [9: −1: 1]
>> sort(a)
```

The dot and cross product functions:

```
>> dot([1 2 3], [3 2 1])
>> cross([1 0 0], [0 1 0])
```

Part 3: Plotting

A. 2D Plotting

1. Simple 2D plotting

```
>> x = [−2: 1: 2] % The x−values
>> y = x.*x       % The y−values = the squares of the x−values
>> plot(x, y)

>>  x = [−2: 0.1: 2]; % Need more points for a good plot!
>> y = x.*x;
>> plot(x, y)
```

2. Setting Plot Attributes:

```
>> plot(x, y, ' r')   % Curve is red
>> plot(x, y, 'g', 'LineWidth', 2) % Curve is green and thicker
>> plot(x, y, 'g:d', 'LineWidth', 2)
>> plot(x, y, 'b:d', 'LineWidth', 2, 'markersize', 4)
>> plot(x, y, 'y:d', 'LineWidth', 2, 'markersize', 2)
```

3. Plotting Several Functions With Labels, Axes, and Grid

A. Setting the plot color and line width

```
>> x = [−−2*pi: 0.1: 2*pi];
>> y = x.*cos(x);    % note use ".*" to generate y's.
>> plot(x, y, 'r', 'lineWidth', 2)
```

B. Plotting Several Functions with Labels

```
>> hold on % keep the current plot
>> plot(x, sin(x), ‘g’, 'lineWidth', 10) % Overlay a plot of the sine
>> hold off
```

C. Setting an the box axes

```
>> axis([-6 6 -6 6], 'square')  % Set the axes
```

D. Adding an x,y-axes

```
>> hold on  % keep the current plots
>> plot( [-6 6], [0 0], 'k', 'lineWidth', 3) % draw x-axis
>> plot( [0 0], [-6 6], 'k', 'lineWidth', 3) % draw y-axis
>> grid on % display a grid
```

E. Adding labels

```
>> xlabel('X')
>> ylabel('Y = X COS(X),  Y = SIN(X)')
```

4. 3D Plotting

A. Generate the x and y values

```
>> x = -2: 0.2: 2; y = x;
```

B. Generate a 'meshgrid' from the x and y values.

```
>> [X,Y] = meshgrid(x, y);
```

C. Evaluate the function Z = F(X,Y) on the meshgrid

```
>> Z = 4 - X.^2 - Y.^2;  % Note uppercase X, Y and the periods!
```

D. Plot the surface

```
>> surf(X,Y,Z)
```

E. Plot a different surface over the same meshgrid

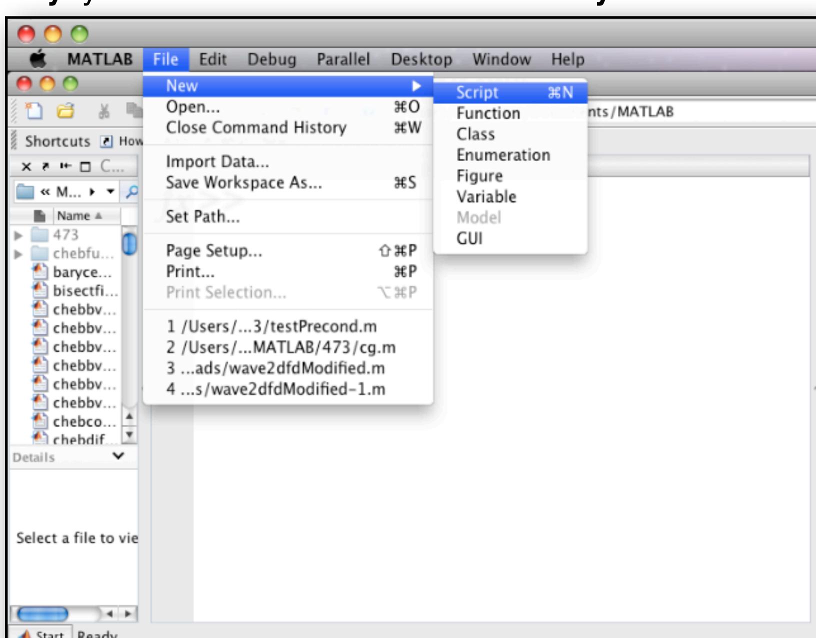
```
>> Z = 1./(X.^2 + Y.^2+1);  % Note uppercase X, Y and the periods!
>> surf(X,Y,Z)
```

Part 4: Simple Script Files

We can create functions and procedures and save them in .m files

A. Select Functions **from the** New **menu (left top of MATLAB IDE)**
Enter the following in the window that opens and save the function in
a file named "piecewise.m" **in a subdirectory** yourNameM **of** Documents **directory.**

```
function result = piecewise(x)
  if (x < -0.5)
    result = x+1.5;
    return
  elseif (x <= 0.5)
    result = 4.0*x*x;
    return;
  else
    result = 1.0;
    return
  end
end
```



B. Select Functions **from the** New **menu (left top of MATLAB IDE)**
Enter the following in the window that opens and save the function in a file named "addAxes.m" in **in the subdirectory** yourNameM **of** Documents **directory.**

```
function addAxes(xMin,xMax, yMin, yMax, color, thickness)
   hold on;
   plot( [xMin xMax], [0 0], color, 'lineWidth', thickness );
   plot( [0 0], [yMin yMax], color, 'lineWidth', thickness );
   grid on;
   hold off
end
```

For Octave, write them using a text editor and saved them as piecewise.m **and** addAxes.m **in the *Documents* directory.**

In both cases, you will need to add a path to this directory to the "search paths".

```
>> addpath( '~/Documents/yourNameM' )
>> x = -2: 0.05: 2; % Generate the x-values
>> y = arrayfun(@piecewise, x); % Apply piecewise(x) to x-values vector
>> plot(x, y, 'r', 'lineWidth', 3)
>> axis( [-3 3 -1 2] )
>> addAxes(-2, 2, -0.5, 1, 'b', 4)
>> addAxes(-3, 3,   -1, 2, 'b', 4)
```

C. An example function to plot Z = F(X,Y) **over a square** [a, b] x [a, b] **using** n **points in each direction. Save the following in the file** plot3d.m .

```
function plot3d(Fhandle, a, b, n)
  x = linspace(a, b, n); y = x;
  [X, Y] = meshgrid(x,y);
  Z = Fhandle(X,Y);
  clf
  surf(X,Y, Z)
end
```

Now call the plot3d **function using** `'@(x,y) expression'` **for the** `Fhandle` **parameter .**

```
plot3d( @(x,y) 9 - x.^2 - y.^2 , -3, 3, 50 )
```

D. An example function to plot the contours of Z = F(X,Y) **over a square** [a, b] x [a, b] **using** n **points in each direction. Save the following in the file contour2d.m**

```
function contour2d(Fhandle, a, b, n)
  x = linspace(a, b, n); y = x;
  [X, Y] = meshgrid(x,y);
  Z = Fhandle(X,Y);
  clf
  [C, h] = contour(X, Y, Z);
  set(h, 'ShowText', 'on', 'TextStep', get(h,'LevelStep')*2)
end
```

Now call the contour2d **function using** `'@(x,y) expression'` **for the** `Fhandle` **parameter .**
```
contour2d(@(x,y) 9 – x.^2 – y.^2 , –3, 3i, 50)
```

Part 5: Additonal Examples

A. Create a text file of data points as follows:

Step 1: Generate the data
```
>> s = 0
>> for i = 0:10
      data(i + 1, 1) = i
      data(i + 1, 2) = s
      s = s + i
   end
```

Step 2: Save the data
```
>> dataFile = fopen('/Users/student/Documents/yourNameData.txt', 'w');
>> fprintf(dataFile, '%6i %6i\n', data);
>> fclose(dataFile);
```

B. Read and plot data points from a text file:

Step 1: Read the data points:
```
>> dataFile = fopen('/Users/student/Documents/yourNameData.txt');
>> A = fscanf(dataFile, '%d %d', [2 inf]);
>> fclose(dataFile);
```

Step 2: Extract the x and y values
```
x = A(1, :)
y = A(2, :)
```

Step 3: Plot the data
```
plot(x, y, '0')
```

C. Plot a solution to the first-order ODE $\frac{dx}{dt} = x$ **over** `[0, 1]` **with** $x(0) = 1$**.**

```
clf
dxdt = @(t,x) x
ode45(dxdt,[0 1], 1)

% Check that the solution is x = exp(t)
hold on
t = 0: 0.1: 1;
plot(t, exp(t),'r')
hold off
```

D. Plot a solution to the first-order ODE $\frac{dx}{dt} = t - x^2$ **over** `[0, 5]` **with** $x(0) = 0$**.**

Note this ODE is known to have no solution in terms of elementary functions!
```
dxdt = @(t,x) t-x^2
```

```
ode45(dxdt,[0 5], 0)
```

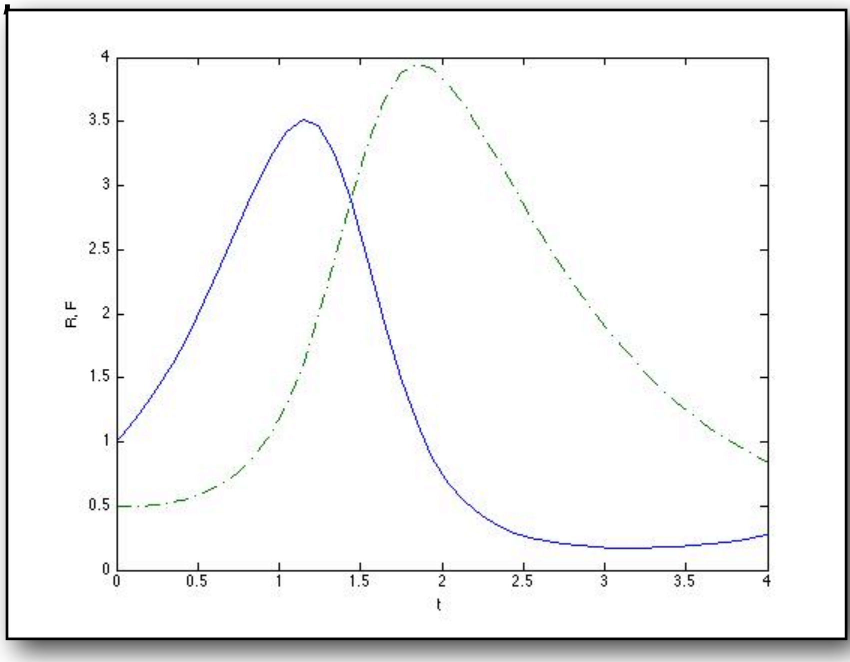
F. Plot a solution to the Foxes and Rabbits first-order Predator-Prey system.

$$\frac{dR}{dt} = 2R - 1.2RF$$
$$\frac{dF}{dt} = -F + 0.9RF$$

where $(R_0, F_0) = (1, 0.5)$.

<u>Step 1</u>: Create a function that represents the system and save it in RF.m .

```
function dydt = RF(t,y)
   dydt = zeros(2,1); % a column vector
   dydt(1) = 2*y(1) - 1.2*y(1)*y(2);
   dydt(2) = -y(1) + 0.9*y(1)*y(2);
end
```

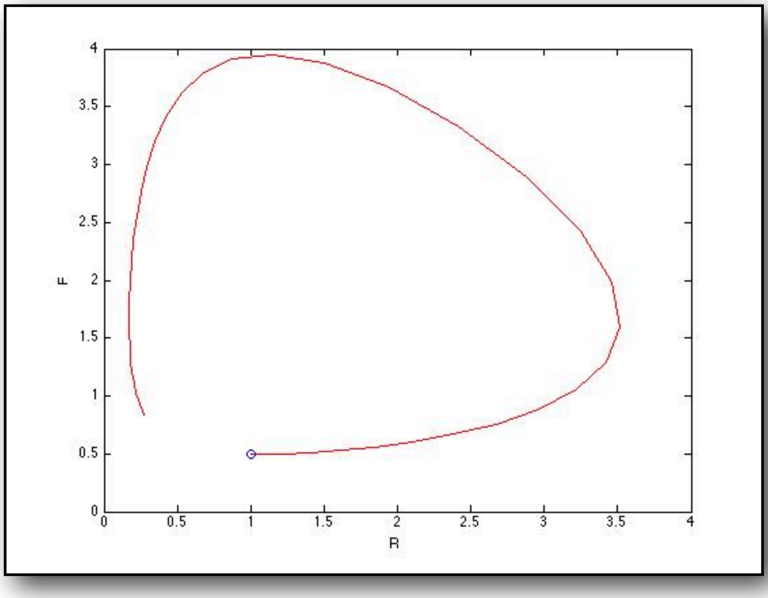


<u>Step 2</u>: Solve the initial-value problem:

```
[T,Y] = ode45(@RF,[0 4],[1 0.5]);
% Plot R(t) and F(t)
```

<u>Step 3</u>: Plot the solution:

```
plot(T,Y(:, 1),'-',T,Y(:, 2),'-.')
% Plot R vs F
figure(2)
plot(Y(:, 1), Y(:, 2))
xLabel('t'); ylabel('R, F')
hold on
plot(1, 0.5, '.')
hold off
```



G. Generate and play a movie.

A. Generate the first frame of the movie

```
>> clf % Clear the figure
>> x = -2: 0.2: 2; y = x;
>> [X,Y] = meshgrid(x, y);
>> Z = 4 - X.^2 - Y.^2;  % Note uppercase X, Y and the periods!
>> figure('Renderer','zbuffer')
>> surf(X,Y,Z)
>> axis tight
```

B. Generate the remaining frames of the movie

```
>> set(gca,'NextPlot','replaceChildren'); % 'gca' = current axes handle
>> for k = 1:20
    % Use -1 ≤ sin(2*pi*k/20) ≤ 1 to create a periodic function!
    surf(X,Y, sin(2*pi*k/20)*Z)
    F(k) = getframe; % Save the frame in F
 end
```

C. Play the movie

```
>> movie(F,20) % Play the movie twenty times
```