**Array Coding**

Write a function that takes in a square array (an array where #rows=#columns), and adds a perimeter of ascending numbers to it, starting at the top left corner and going clockwise around the array.

function arr=perimeterCount(arr)

 m=length(arr); %remember, m is both the # of cols and # of rows

top=2 : 1+m;

arr=[top; arr];

right=2+m : 2+2.\*m;

arr=[arr right']; %right has to be a column vector

bottom=3+2.\*m : 3+3.\*m;

arr=[arr;bottom(end:-1:1)]; %bottom has to be reversed (to continue clockwise counting)

left=4+3.\*m : 4+4.\*m;

left=left'; %left has to be a column vector

left=[1; left(end:-1:1)]; %left has to be reversed (to continue clockwise counting)

arr=[left arr];

end

Write function that will take in an array. The output array should be such that the bottom row of is the sum of all the columns in the inputted array and the leftmost column is the sum of all the rows. The leftmost number on the bottom row should be the sum of all the elements in the array.

function newArr = sumArr(arr)

newArr = [arr; sum(arr)];

newArr = [newArr sum(newArr,2)];

end

Write a function that takes in an array and inserts an array of zeros between the two halves of the array. The width of the zeros array should be exactly one-third of the final width (or one-half of the original width). You can assume the array you are given will have an even number of columns (and will have at least 2 columns).

function out=arrayFun(arr)

[m n]=size(arr);

middleZs=zeros(m,n./2);

half1=arr(:, 1:n./2); %indexes first half of original vector

half2=arr(:, n./2+1:n); %indexes second half of original vector

out=[half1 middleZs half2];

end

Write function that takes in an array and sorts the values in successive columns. When the modified array is given as an output, it has to have the same dimensions are the original array inputted

function newArr= sortArr(arr)

[ r c] = size(arr);

temp = sort(arr(:));

newArr = reshape(temp,r,c);

end

Write a function ArrayMod which takes in an array and performs the following opperations on it: 1 - remove all the even columns. 2 - switch the first and last rows. 3 - divide each odd number by 4.

function modified = ArrayMod(arr)

    modified= arr;

    modified(:, 2:2:end) = [];

    first\_row= modified(1,:);

    last\_row= modified(end, :);

    modified(1,:)= last\_row;

    modified(end,:)= first\_row;

    modified(mod(modified,2)==1) = modified(mod(modified,2)==1) ./4;

end

**Cell Array**

Write a function that takes in a cell containing numbers either as strings or as doubles. If the element is a double, change it to a vector of the negative of the element then the original. (For example, if in{1}= [8], out{1}=[-8 8] ) If the element is a string, simply convert to double. Refer to test cases.

|  |
| --- |
| function out=numberCell(cA)  for x=1:length(cA)      if strcmp(class(cA{x}), 'char')          out{x}=str2num(cA{x});      elseif strcmp(class(cA{x}), 'double')          out{x}=[(-1).\*cA{x} cA{x}]      end  end |

Given a cell array, delete the first and last cells. Then add 1 to anything that is of type double in the cell.

function modified= modCA(ca)

    modified= ca(2:end-1);

     for ind=1:length(modified)

        if isnumeric(modified{ind})

            modified{ind}= modified{ind}+1;

        end

    end

end

Write a function that takes in a cell containing: 1) a cell of adjectives 2) a cell of nouns 3) a cell of verbs. Output a cell of sentences in the form: The (adjective) (noun) (verb).

function out=basicSentence(in)

out={};

for x=1:length(in{1})

     sentence=sprintf('The %s %s %s.', in{1}{x}, in{2}{x}, in{3}{x});

     out=[out sentence];

end

end

**Conditionals**

The function switchAndIf will indicate when to use switch and ifs based on the following conditions:   
  
If the following strings are inputted:   
'discrete','menu','limited choices' then say:   
'Using switch might be a good idea'   
  
if the following strings are inputted:   
'range, does not check equality' then say:   
'Using if might be a good idea'

function out = switchAndIf(type)

if ischar(type)

    switch type

         case {'discrete','menu','limited choices'}

              out = 'Using switch might be a good idea';

         case {'range, does not check equality'}

              out = 'Using if might be a good idea';

        otherwise

            out = 'Invalid string';

    end

else

    out ='Invalid input';

end

end

The function study will take in a number of hours the person studies on a daily basis and study comments according to these conditions:   
  
If they study less than 4 hours each day say:   
'I wish I had your schedule!';   
  
If they study more than 4 but less than 10 hours say:  
'Keep up the good work!';   
  
If they study more than 10 hours say:   
'You dont sleep do you?'   
  
and if the user does not enter in a valid input, say:   
'Please enter hours of study'

function comment = study(hours)

if hours < 4;

    comment = 'I wish I had your schedule!';

elseif hours <= 10 & hours >=4

    comment = 'Keep up the good work!';

elseif hours > 10

    comment = 'You dont sleep do you?';

else

    comment ='Please enter hours of study';

end

Write a function using a switch statement that takes in a relationship status from a facebook profile and outputs the percent chance you have of getting this person to start dating you. (This assumes that this person is attracted to you.) Here are the (most likely completely inaccurate) statistics:

100% chance if it reads "Single" or "Divorced"

75% chance if it reads "It's complicated" or "In an open relationship"

60% chance if it reads "Separated" or "Widowed"

30% chance if it reads "In a relationship"

5% chance if it reads "Engaged"

1% chance if it reads "Married"

|  |
| --- |
| function out=facebookLUV(str)  switch  str      case {'Single', 'Divorced'}          out=100;      case {'It''s complicated', 'In an open relationship'}          out=75;      case {'Separated', 'Widowed'}          out=60;      case {'In a relationship'}          out=30;      case {'Engaged'}          out=5;      case {'Married'}          out=1;  end |

Write a function that takes in an age and outputs a string of which life stage a person of this age is in. The life stages for this function will be

1) child (0-12yrs)

2) teenager (13-17yrs)

3) adult (18-64yrs)

4) senior citizen (65+yrs)

|  |
| --- |
| function str=lifeStage(age)  if age<13      str='child';  elseif age<18      str='teenager';  elseif age<65      str='adult';  else      str='senior citizen';  end |

Function Description: You are given a string and a vector. If the length of the string is greater than the number, return 'String'. If the number is greater return 'Vector'. If they are both empty, return 'Empty'. If they are the same length, return 'Equal'

function out= Compare(str, vec)

    if length(str)>length(vec)

        out= 'String';

    elseif length(vec)>length(str)

        out= 'Vector';

    elseif isempty(vec) & isempty(str)

        out= 'Empty';

    elseif length(vec)==length(str)

        out= 'Equal';

    end

end

**File I/O**

Function Description: Create a function that will take in a excel file contain student's test grades. You can assume the following about the excel file:

1. the first row is will always contain the student's names
2. the first column will also always contain what test the student took
3. second row and on , also the second column and on, you will find only numbers which represents the student's test scores.

The function should add a header called 'Max Average' on the first column, two rows after the last test. The maximum average of the students should be stored in the column to the right. Below 'Max Average' should be 'Min Average' and to the right of that, should be the minimum student average.   
Finally, the function should write the file out to as an excel file with the same name as the original file name except the word 'Edited\_' is now attached to the front of it.

function avgMinMax(fileName)

[ num txt raw ] = xlsread(fileName);

[ r c] = size(raw);

raw(r+2 , 1) = {'Max Average'};

raw(r+3 , 1) = {'Min Average'};

raw(r+2, 2) = num2cell(max(mean(num)));

raw(r+3, 2) = num2cell(min(mean(num)));

xlswrite(['Edited\_' fileName], raw);

Description: Create a function that takes in file name. You can assume the following about the file:

1. the file's 1st row will always contain the students' names as strings.
2. the file's 1st column will always contain the names of the tests each student had to take
3. the 2nd column and on , also the 2nd row and on you can assume this area to be populated with students' scores (doubles).

Your job is to determine the lowest test score for each student and then output a cell array containing the names of each of the students in the first row (should be in the same order as original data), and the test each student should drop in the second row.

function out = dropTest(fileName)

[num txt raw] = xlsread(fileName);

[ r c ] = size(raw);

[minTest index] = min(num);

students = txt(1,2:end);

testNames = txt(index+1);

out = [students; testNames];

Write a function dlmStats which takes in an dlm file and outputs a modified dlm file. The delimiter will be specified in the second input. The inputted file will always have 4 columns. The first column will always be consecutive integers (i.e. 1, 2, 3, etc) that correspond to a given team. The second column will be the number of wins of that team in a given season The third column will be the total losses that season. The fourth column will the the number of ties. The function should create a new array that gives the following statistics all in the first row: 1 - the ID number of the team with the most wins 2 - the ID number of the team with the most losses 3 - the ID number of the team with the least ties Write this new array to delimited file which should have the same base name as the inputted name with '\_new' added to the end. The new delimiter is specified in the third input

function dlmStats(filename, d1, d2)

    arr = dlmread(filename, d1);

    ID = arr(:,1)

    [maxwins ind1] = max(arr(:,2));

    new = ID(ind1);

    [maxloss ind2] = max(arr(:,3));

    new = [new ID(ind2)];

    [mintie ind3] = min(arr(:,4));

    new = [new ID(ind3)];

    new\_name= [filename(1:end-4) '\_new' filename(end-3:end)];

    dlmwrite(new\_name, new, d2)

end

Write a function xlsBaseball which takes in an excel file and outputs a modified excel file. The input excel file will always have 3 columns. This first column will always have 'Team' in the first row and the names of the teams under. The second column will always have 'Runs Scored' in the first row and the amount of runs scored by each team had under. The third column will always have 'Runs Allowed' in the first row and the amount of runs allowed by each team had under. The function should create a new array which is identical to the excel data except with one added row to the very bottom. In this row, the first column should contain 'Totals'. The second column should contain the total amount of runs scored. The third column should contain the total amount of runs allowed. Write this new array to an excel file which should be named the same as the input, but with the prefix 'new\_'

|  |
| --- |
| function xlsBaseball(filename)        [num txt raw]= xlsread(filename);      total= sum(num);      raw{end+1,1}= 'Totals';      raw{end,2}= total(1);      raw{end,3}= total(2);      a= find(filename=='.');      new\_name= [filename(1:a-1) '\_new' filename(a:end)];        xlswrite(new\_name, raw)  end |

**High Level**

Write a function that takes in a filename of either a dlm or csv file and returns the same file with the opposite extension. Assume the delimiter for the file is a comma.

function csvdlmSwitch(filein)

if strcmp(filein(end-2:end), 'csv')

    arr=csvread(filein);

    fileout=sprintf('%sdlm', filein(1:end-3) );

    dlmwrite(fileout, arr)

else

    arr=dlmread(filein);

    fileout=sprintf('%scsv', filein(1:end-3) );

    csvwrite(fileout, arr)

end

end

Write a function that takes in a csv file of phone numbers. In the file, the numbers are separated as follows: ###,###,####.   
  
Change the numbers into strings in the form: (###)###-#### and output the numbers in a column cell array. (Note: None of the numbers will begin with a 0 and all the values will be in a single row. )

function out=phoneNumbers(filein)

nums=csvread(filein);

count=1; %initializer

for x=1:3:length(nums) %each phone number is broken into 3 parts, so go by 3's

    str=sprintf('(%d)%d-%d', nums(x), nums(x+1), nums(x+2));

    out{count, 1}=str;

    count=count+1; %this is going to count where we are in the output cell, since x jumps by 3's

end

end

Write a function that takes in an excel file with a column of symbols (possibly including numbers). There are no outputs, but you should rewrite the file, with the ascii values of each symbol in the second column.

function addingAscii(filein)

[num txt raw]=xlsread(filein);

[r c]=size(raw);

for x=2:r %starts at 2 because titles of columns are in row 1

    if isnumeric(raw{x,1}) % if the symbol at x is a number

        symbol=num2str(raw{x,1}); %then change the number to a string

    else

        symbol=raw{x,1};

    end

    raw{x,2}=double(symbol); %changes symbol to ascii value

end

xlswrite(filein, raw)

end

Write a function that takes in an excel file of boys' names and girls' names and outputs a cell of what each couple's initials would look like if carved in a tree (in the form: AB & CD).

function out=treeCarving(filein)

[num txt raw]=xlsread(filein);

[r c]=size(txt);

out={}; %initializes output cell

for x=2:r %starts at 2 because titles of columns are in row 1

    boy=txt{x,1};

    [first last]=strtok(boy); %separates first and last name

    in=[first(1) last(2)]; %concatenates initials of boy

    girl=txt{x,2};

    [first last]=strtok(girl); %separates first and last name

    in=[in ' & ' first(1) last(2)]; %concatenates initials of boy with & and initials of girl

    out=[out in]; %concatenates couples initials into cell

end

end

**Images**

Write a function that opens an image file and writes a new image file with the original image, a mirror image of it, and a frame around it. The third input tells you the gaze of the person in the original file. The mirror image should be flipped and concatenated on to the original so that the original person and the mirror person are looking at each other. The frame will always be 15 pixels thick. The color of the frame is in RGB format in the 4th input.

function myMirror(img,name,gaze,vec)

img=imread(img);

switch gaze

    case 'Up'

        copy=img(end:-1:1, :,:);    %flip vertically

        img=[copy;img];             %concatenate copy to top

    case 'Down'

        copy=img(end:-1:1, :,:);    %flip vertically

        img=[img;copy];             %concatenate copy to bottom

    case 'Right'

        copy=img(:,end:-1:1,:);     %flip horizontally

        img=[img, copy];            %concatenate copy to right

    case 'Left'

        copy=img(:,end:-1:1,:);     %flip horizontally

        img=[copy, img];            %concatenate copy to left

end

[r c layers]=size(img);

%find new dimensions for final output

    thick=15; %frame is always 15 pixels thick

    r=r+2\*thick;

    c=c+2\*thick;

%create color sheet (image with every pixel having value of input vec)

CS(:,:,1)=ones(r,c)\*vec(1);

CS(:,:,2)=ones(r,c)\*vec(2);

CS(:,:,3)=ones(r,c)\*vec(3);

%note: CS is class double --->change to uint8

CS=uint8(CS);

%insert mirror image into middle of color sheet

CS(thick+1:r-thick, thick+1:c-thick, :)=img;

%write to new filename

imwrite(CS, name)

end

Function description: Write a function that does the following:

1. Reads in the image using the filename
2. Finds each layer where each layer of the pixel is 0
3. When the pixels are found, turn these pixels white
4. Return the modified image array

function img = whiteOutBlack(filename)

img = imread(filename);

mask = img(:, :, 1) == 0 & img(:, :, 2) == 0 & img(:, :, 3) == 0;

mask = cat(3, mask, mask, mask);

img(mask) = 255;

end

Write a function that changes an image to grayscale by averaging. The only difference is that you are going to leave the color contained in the second input alone. In addition, you will not change any color within plus or minus the given factor (input 3). For example, if your second input is [34 155 0], and your third input is 1, the following color combinations would be kept in color and not changed to grayscale

function img=keep1color(img, vec, fac)

img=imread(img);

%change to double b/c we'll be doing math when converting to grayscale

img=double(img);

%make copy of each layer of img

copyR=img(:,:,1);

copyG=img(:,:,2);

copyB=img(:,:,3);

%find where all three color values are within the range of vec plus or

%minus fac

keep=copyR<=vec(1)+fac & copyR>=vec(1)-fac & copyG<=vec(2)+fac ...

    & copyG>=vec(2)-fac & copyB<=vec(3)+fac & copyB>=vec(3)-fac;

%change original image to grayscale using averages method

gray = (img(:, :, 1) + img(:, :, 2) + img(:, :, 3))./3;

% Set each color layer to gray values

grayR = gray;

grayG = gray;

grayB = gray;

%where we found colors close enough to color represented in vec,

%insert color values back in

grayR(keep)=copyR(keep);

grayG(keep)=copyG(keep);

grayB(keep)=copyB(keep);

%edit img to be our new gray & vec-colored version

img(:,:,1)=grayR;

img(:,:,2)=grayG;

img(:,:,3)=grayB;

%change back to class uint8

img=uint8(img);

end

Given the filename of an image, read in the image and perform the following opperations. Find the pixels where the red layer is less than 100 and where the blue layer is less than 150. At this location, set the pixel to be white.

function out= WhiteOut(fn)

    data= imread(fn);

    red= data(:,:,1);

    green= data(:,:,2);

    blue= data(:,:,3);

    ind= red<100 & blue<150;

    mask= cat(3,ind,ind,ind);

    data(mask)= 255;

    out= data;

end

Given the filename of an image, read in the image and perform the following opperations: Swap the right and left halves of the image. After that, rotate the new array by 90 degrees.

function out= Switcher(fn)

    data= imread(fn);

    [r c l]= size(data);

    left= data(:,1:round(c/2),:);

    right= data(:,round(c/2)+1:end,:);

    new= [right left];

    out(:,:,1)= new(:,:,1)';

    out(:,:,2)= new(:,:,2)';

    out(:,:,3)= new(:,:,3)';

    out= out(:,end:-1:1,:);

end

**Iteration**

Write a function that counts all the "um"s and "uh"s in a given string. (Note: you are counting the words "um" and "uh". For instance, the occurrence of "um" within the word "dumb" wouldn't count.)

function [countUm countUh]=UmsandUhs(str)

countUm=0; %sets an initial value

countUh=0; %sets an initial value

while ~isempty(str) %str will be empty once strtok goes through every word in str

    [word str]=strtok(str) %str variable now only contains the leftovers

    word=word((word>='A' & word<='Z') | (word>='a' & word<='z'))

%the step above keeps only the word, and not punctuation that strtok may have overlooked

    if strcmp(word, 'um')

        countUm=countUm+1;

    elseif strcmp(word, 'uh')

        countUh=countUh+1;

    end

end

Given an unsorted vector, sort it without calling the function sort. Use iteration.

function sorted= slowSort(unsorted)

    sorted= [];

    while ~isempty(unsorted)

        min\_values = min(unsorted);

        ind= find(unsorted==min\_values);

        sorted= [sorted unsorted(ind)];

        unsorted(ind)=[];

    end

end

Write a function that takes in a string of a phrase (with no punctation) and finds the first letter of every word in the string and concatenates them together to form an acronym, using iteration.

function out=myAcronym(str)

out=''; %initialize out as an empty string

for x=1:length(str)

    if x==1 %first letter of first word is at index 1

        out=[out str(x)];

    elseif str(x)==' ' %the first letter of the other words will follow a space

        out=[out str(x+1)];

    end

end

Assuming the user inputs a valid number of rows and columns and also a vector, reshape the vector into the array of the specified dimensions of rows and columns.

function arr = createArr(vec,r,c)

count = 1;

for i1 = 1:r

    for i2 = 1:c

        arr(i1,i2) = vec(count);

        count = count +1;

    end

end

Use a loop( you must use it) through a vector and obtain the value and the position of that vector. Then outputs an array in such a way that the first row of that array is are the elements inside that vector and the second row contains the position of that element.

|  |
| --- |
| function arr =  locationsVec(vec)  for i = 1:length(vec)  arr(1,i) = vec(i);  arr(2,i) = i;  end |

**Low Level I/O**

Write a function that reverses the words in each line of a file, and writes all lines to a new file. The new file name should be the original file name with 'reverse\_' added to the beginning. (Make sure there isn't a blank line at the end of the new txt file.)

function backwardTxt(name)

fh=fopen(name);

line=fgetl(fh); %get first line

nameout=sprintf('reverse\_%s', name); %new file name

fhout=fopen(nameout, 'w'); %open new file for writing

while line~=-1 %while there are still lines to go through

    curline=[];

    while~isempty(line)

        [word line]=strtok(line);

        curline=[word ' ' curline]; %reverses the line's words

    end

    line=fgetl(fh); %get next line

    if line~=-1

        fprintf(fhout, '%s\n', curline)

    else %if the previous line was the last line, no \n

        fprintf(fhout, curline)

    end

end

fclose(fh)

fclose(fhout)

end

Given a filename to a a .txt file,write a function txtMod. -This function will delete the last word in each line of the text file. -The function should then write a new .txt file with the suffix \_new added onto the end of the input filename. -Only delete the last word (not the space before it). -Also note, you should not add a new line character to the last line.

|  |
| --- |
| function txtMod(fn)    fh= fopen(fn, 'r');  a= find(fn=='.');  new\_name= [fn(1:a-1) '\_new.txt'];  fh2 = fopen(new\_name, 'w');  line=fgets(fh);    while ischar(line)      if line~=-1          ind= find(line== ' ');          line= line(1:ind(end));          temp= line;          line= fgets(fh);          if line~= -1              fprintf(fh2, '%s\n', temp);          elseif line ==-1              fprintf(fh2, '%s', temp);          end      end  end  fclose(fh);  fclose(fh2);  end |

Write a function reads the indicated txt file and finds the "hidden message" inside it. The letters of the hidden message are all of the uppercase letters in the file. Also, if you reach a number 0-9, this indicates that there should be a space in the hidden message. The output string hidden message) should be in all lowercase.   
  
For example, if a file was one line that read 'gHi5jK', the output string would be 'h k'.

|  |
| --- |
| function str=capitalCode(fname)  fh=fopen(fname);  line=fgetl(fh); %get first line  str=''; %initialies variable  while line~=-1      for x=1:length(line)          if line(x)>='A' & line(x)<='Z'              str=[str lower(line(x))]; %change to lowercase and add to output          elseif line(x)>='0' & line(x)<='9'              str=[str ' ']; %add a space          end      end      line=fgetl(fh); %get next line  end  fclose(fh);  end |

Write a function called replaceAll, similar to the one implemented in Word. It should go through the file and remove all occurrences of the string in the second input and replace them with the string in the third input. These lines should then be written to a new file defined as, the original filename with '\_edited.txt' amended to the end. 

|  |
| --- |
| function replaceAll(file,dlm,dlm2)  fh=fopen(file,'r');  fh2=fopen([file(1:end-4) '\_edited.txt'],'w');  line=fgets(fh);  while ischar(line)      [a b]=strtok(line,dlm);      while ~isempty(b)          a=[a dlm2 b(length(dlm)+1:end)];          [a b]=strtok(a,dlm);      end      fprintf(fh2,a);      line=fgets(fh);  end  fclose(fh)  fclose(fh2) |

Write a function that reads a given file and outputs a column vector of the lengths of each line. You can assume that no line will be empty.

function vec=lengthTxt(fname)

fh=fopen(fname);

line=fgetl(fh);

vec=[];

while line~=-1

    vec=[vec; length(line)];

    line=fgetl(fh);

end

end

**Matrices**

You are given the following system: 2x+4y+z=-3 3x=2 2z+3y= -4x Write your code to solve this specific system for the variables x,y, and z. Store the values of x,y, and z in the variables X,Y,Z.

A= [2 4 1; 3 0 0; 4 3 2];

b= [ -3; 2; 0];

sol = A\b;

% or sol = inv(A) \* b;

% or sol = A^(-1) \*b;

X= sol(1);

Y=sol(2);

Z= sol(3);

Write a function that solves a system of equations inputted as a csv file. Each row of the csv file will have n number of coefficient values and one solution value. In other words, each row is in the form C1x1+ C2x2 + ... + CnXn= Z, where C1, C2, ... , Cn are the coefficients of the n number of variables. The other function input is a cell containing the variable names, for instance... {'x' 'y' 'z'}. See the test cases below to see how the ouput should appear.

function report=solvecsv(file, variables)

arr=csvread(file);

[r c]=size(arr)

coef=arr(:,1:c-1);

vals=arr(:,c);

sols=coef\vals;     %or inv(coef)\*vals

for x=1:length(variables)

    report{x}=sprintf('%s equals %s', variables{x}, num2str(sols(x)))

    %num2str is used above to avoid something like 'x equals 2.000'

end

end

Write a function that plots a pinwheel. A 2D pinwheel consists of 4 congruent triangles. The first input of this function is a cell containing the coordinates of the corners of a starting triangle. Create the other triangles by using the rotation matrix. The pinwheel will always be made of 4 triangles. The other two inputs will be the colors of the pinwheel. The two colors should alternate and will already be in the single letter form that is used in the plot function. Set the axes to equal.

function pinwheel(pts, color1, color2)

pts=[pts{:}];   %turns cell into vector

xcoord=pts(1:2:end);    %odd positions are x-coordinates

xcoord=[xcoord xcoord(1)];

%add the first coordinate at the end so that the triangles will close

ycoord=pts(2:2:end);    %even positions are y-coordinates

ycoord=[ycoord ycoord(1)];

i=1;

while i<=4;

    switch i

        case {1,3}  %if this is the 1st of 3rd triangle, use first color

            plot(xcoord, ycoord, color1)

        case {2,4}  %if this is the 2nd or 4th triangle, use the second color

            plot(xcoord, ycoord, color2)

    end

    matrix=[xcoord; ycoord]; %concatenate into matrix

    rot=[cos(pi./2) -sin(pi./2)

        sin(pi./2) cos(pi./2)]; %rotation matrix for 90 degree rotation

    new=rot\*matrix; %calculate new coordinates

    %separate into x and y coordinates for next loop:

    xcoord=new(1,:);

    ycoord=new(2,:);

    i=i+1;

    hold on

end

hold off

axis equal

end

**Numerical Methods**

Write a function that, using this concept, plots the top edges of the rectangles that determine the area under a curve. Seeing the examples below should help clarify what you will be doing.   
-Interpolate the given x and y values to find 100 x and y values within the bounds of the given x vector.   
-Use the left version of the sum (in other words, a horizontal line should start at every x value except the last.   
-You will find the polyfit() and polyval() functions to be helpful.   
-Plots should be in red with axis set to "square"

function leftRiemann(x,y,n)

xi=linspace(min(x), max(x));

d=xi(2)-xi(1); %distance between x's

C = polyfit(x,y,n); %interpolate based on n (n being the order)

yi = polyval(C,xi); %find vals of new y's

rX=[];

rY=[];

for i=1:length(xi)-1

    rX=[rX xi(i) xi(i)+d]; %builds vector of x-values for horizontal lines

    rY=[rY yi(i) yi(i)]; %builds vector of y-values for horizontal lines

end

plot(rX, rY, 'r') %note: horizontal lines will automatically be connected (look like stairs)

axis square

end

Write a function that takes in a vector of times and a vector of distances and plots three graphs: distance vs. time, velocity vs. time, and acceleration vs. time. Label the graphs and color them as you see below. (Axis is set to "square".) Also find and output average velocity and average acceleration.

|  |
| --- |
| function [avgV avgA]=motionAnalyze(t, d)  subplot(1,3,1)  plot(t,d)  xlabel('time')  ylabel('distance')  title('distance vs. time')  hold on  axis square    v = diff(d)./diff(t); %velocity function  dd/dt  subplot(1,3,2)  plot(t(2:end),v, 'g') %use t(2:end) because length(v)=length(t)-1  xlabel('time')  ylabel('velocity')  title('velocity vs. time')  axis square  avgV=mean(v);    a = diff(v)./diff(t(2:end)); %acceleration function dv/dt  subplot(1,3,3)  plot(t(3:end),a, 'r') %use t(3:end) because length(a)=length(v)-1  xlabel('time')  ylabel('acceleration')  title('acceleration vs. time')  axis square  avgA=mean(a);  end |

Write a function that reads an excel file of x and y data and re-writes the file with a third column of the cumulative area under the curve at each increasing x value. (The x values will already be sorted.)

function xlstrapz(file)

[num txt raw]=xlsread(file);

x=num(:,1);

y=num(:,2);

area=cumtrapz(x,y);

raw{1,3}='Area under curve'

for i=1:length(area)

    raw{i+1, 3}=area(i)

end

xlswrite(file, raw)

end

Write a function called integral which takes in the coefficients of a polynomial to be integrated assuming a constant of integration of zero. That integrated polynomial should then be evaluated by the bounds represented by the x-values in the second input, i.e. [x1 x2]. The two values given represent the lower and upper values of the integration, the difference of which will give the total integral value. For example, a constant polynomial such as y=2 is seen as just [2] in the first input. When integrated, it gives [2 0] or y=2x+0. Evaluated at x=1 and x=2 or [1 2], the integral gives back [2 4] or y=2 and y=4. thus the output values should be [2] or 4-2=2.

function out=integral(coef, x)

int\_coef=[coef./[length(coef):-1:1] 0];

y=polyval(int\_coef,x);

out=y(2)-y(1);

Write a function called compareInterpFit which takes in a vector of x values and a vector of y values. The function will plot three lines all on the same plot. The first line will be the plot of the original x and y values in red pentagrams ('rp'). The second will be a line produced as follows: using curve fitting find a cubic (order 3) polynomial the fitted the data and use the polynomial to evaluate new x-values (100 evenly spaced values from the original first value of x and the original last value of x (i.e. use linspace)). This will be the second line which should be green. The third line will have y values produced by the interp1 function, using the original data and the new x values used in curve fitting, this should be a yellow line. Label the lines using legend, the first being 'Original Points', the second 'Curve Fit' and the third 'Interp1'.

function compareInterpFit(x,y)

newx=linspace(x(1),x(end));

coef=polyfit(x,y,3);

curvefity=polyval(coef,newx);

interpy=interp1(x,y,newx);

plot(x,y,'rp',newx,curvefity,'g',newx,interpy,'y')

legend('Original Points','Curve Fit','Interp1')

Function Description: Create a function that take in two vectors of x and y values respectively. Do the following.

1. Take the derivative given the x and y and plot it as black dashed line.
2. Then take take the second derivative and plot it on a different graph but same figure as red dotted line.
3. The location of second derivative plot should be to the left of the first derivative plot.
4. Also remember to label the title 'First Derivative' and 'Second Derivative' for first derivative and second derivative plot respectively .
5. Label the x-axis and y-axis of each plot 'x-axis' and 'y-axis' respectively.

|  |
| --- |
| function plotDer(x,y)    deriv = diff(y)./diff(x);  subplot(1,2,1)  plot(x(2:end), deriv, 'k--');  title('First Derivative');  xlabel('x-axis');  ylabel('y-axis');    subplot(1,2,2)  deriv2 = diff(deriv)./diff(x(2:end));  plot(x(3:end), deriv2, 'r:');  title('Second Derivative');  xlabel('x-axis');  ylabel('y-axis');    end  Function Description: Create a function that takes in a vector of x-values, a vector y-values, and a vector of new x-values. Do the following:   * 1. Check whether its possible whether to plot a unique 4th order for the x and y-values inputs,      1. if its not possible, output the a\ string saying '4th Order is Not Unique!', and do not plot anything.      2. if its possible to play a unique 4th order, out the string saying '4th Order is Unique' and plot the following on THE SAME PLOT:         1. plot the original x and y as a solid black line         2. plot the 1st order as a magenta solid line         3. plot the 2nd order as a red dashed line         4. plot the 3rd order a blue dashed line         5. plot the 4th order a green solid line         6. Label the x-axis as 'x-axis'         7. Label the y-axis as 'y-axis'         8. Label the title as '4th Order and Below’ |
| function string =  plot4Order(x,y,new\_x)  if length(x)-1 > 4  string = '4th Order is Unique';  coeffs1 = polyfit(x, y, 1);  coeffs2 = polyfit(x, y, 2);  coeffs3 = polyfit(x, y, 3);  coeffs4 = polyfit(x, y, 4);  new\_y1 = polyval(coeffs1, new\_x);  new\_y2 = polyval(coeffs2, new\_x);  new\_y3 = polyval(coeffs3, new\_x);  new\_y4 = polyval(coeffs4, new\_x);  plot(x, y, 'k-');  hold on  plot(new\_x, new\_y1, 'm');  plot(new\_x, new\_y2, 'r--');  plot(new\_x, new\_y3, 'b--');  plot(new\_x, new\_y4, 'g');  title('4th Order and Below');  xlabel('x-axis');  ylabel('y-axis');  else      string = '4th Order is Not Unique!';  end |

**Plotting**

Write a function that takes in a request of the form:

'Give me a (circle or square) with (radius/side length) (#).'

and outputs a response in the form:

'Here is your (color) (circle/square) with (radius/side length) (#).'

function out=circlesNsquares(str)

num=str2num(str(end-1)); %index num out of string and change to class double

if mod(num, 2)==0 %if num is even

    color='green';

else %if num is odd

    color='cyan';

end

if ~isempty(strfind(str, 'circle')) %if 'circle' is found within str

    R=num;

    th=linspace(0,2.\*pi, 100);  %assign theta as instructed

    x=R.\*cos(th);

    y=R.\*sin(th);

    plot(x,y,color(1)); %take the first letter because the plot function

%"knows" that 'c' stands for cyan and 'g' stands for green

    out=sprintf('Here is your %s circle with radius %d.', color, R);

elseif ~isempty(strfind(str, 'square')) %if 'square' is found within str

    s=num;

    half=s./2; %need to divide side length in half because need to graph the square centered around the origin

    plot([-half half half -half -half], [-half -half half half -half], color(1));

    out=sprintf('Here is your %s square with side length %d.', color, s);

else %if neither 'circle' nor 'square' was found...

    out='I only draw circles and squares.';

end

axis square

xlabel('x-axis')

ylabel('y-axis')

title(out) %title is same as output string

end

Write a function that takes in the area of the base of a square-based pyramid and the height of that pyramid. Plot the pyramid then flip it over so that two pyramids base to base create a 3D diamond shape. The center of the diamond should be at (0,0,0).

|  |
| --- |
| function plotDiamond(base, h)  s=sqrt(base); %base of pyramid will be a square, so take sqrt to find one side  plot3([-s/2 s/2 s/2 -s/2 -s/2], [-s/2 -s/2 s/2 s/2 -s/2], zeros(1,5), 'm-.')  %line above plots base, remembering to connect back to first coordinate to close shape  hold on  % lines that reach top tip  plot3([-s/2 0 s/2], [-s/2 0 -s/2], [0 h 0], 'm-.')  plot3([s/2 0 -s/2], [s/2 0 s/2], [0 h 0], 'm-.')  %lines that reach bottom tip (only difference is h is now negative)  plot3([-s/2 0 s/2], [-s/2 0 -s/2], [0 -h 0],'m-.')  plot3([s/2 0 -s/2], [s/2 0 s/2], [0 -h 0], 'm-.')  title('He went to Jared!')  hold off  axis equal  end |

Function Description: Function should use the filename given in the first input and the delimiter given in the second input to open the file. The file will be formatted in the following way: the first column will have values of x, the second column will have values of y and the third column will may or may not have values of z. If there are no z values, you will used the plot() function, if there are z values, use the plot3() function. When plotting, use the style specified in the third input.

|  |
| --- |
| function dlmPlot(fn, dlm, style)    arr = dlmread(fn, dlm);    [r c] = size(arr);    if c == 2       x = arr(:, 1);       y = arr(:, 2);       plot(x, y, style);    else       x = arr(:, 1);       y = arr(:, 2);       z = arr(:, 3);       plot3(x, y, z, style);    end |

Function Description: The function takes in two vectors of x values (denoted as x1 and x2 )and two vectors of y values (denoted as y1 and y2).   
  
The function will plot four graphs onto the same figure in a 2 x 2 style.

1. 1st graph - top left should be x1 and y1 plotted with red plus signs titled 'Plot 1'
2. 2nd graph - top right should be x1 and y1 plotted with black line titled 'Plot 2'
3. 3rd graph - bottom left should be x2 and y2 plotted with green stars (\*) titled 'Plot 3'
4. 4th graph - bottom right should be x2 and y2 plotted with blue squares titled 'Plot 4'

Each of the graphs should have their x and y axes labeled 'x-axis' and 'y-axis' respectively.

|  |
| --- |
| function fourQuadrants(x1,y1,x2,y2)    subplot(2,2,1)  plot(x1,y1,'r+')  title('Plot 1')  xlabel('x-axis')  ylabel('y-axis')    subplot(2,2,2)  plot(x1,y1,'k-')  title('Plot 2')  xlabel('x-axis')  ylabel('y-axis')    subplot(2,2,3)  plot(x2,y2, 'g\*')  title('Plot 3')  xlabel('x-axis')  ylabel('y-axis')    subplot(2,2,4)  plot(x2,y2, 'bs')  title('Plot 4')  xlabel('x-axis')  ylabel('y-axis') |

Function Description: You are given a vector or x-coordinates and a vector of y-coordinates. The points to plot are the corresponding positions of each vector. That is, x\_coord(1) and y\_coord(1) represent one point; x\_coord(2) and y\_coord(2) represent another point; etc. (You may assume the two vectors will have the same length). This function should only plot the points where BOTH the x\_coords AND the y\_coords are greater than 3. This should be plotted with black dashed lines and should be titled 'Greater than 3 Plot'

|  |
| --- |
| function plotPoints(x\_coords, y\_coords)        ind= x\_coords<=3;      x\_coords(ind)= [];      y\_coords(ind)= [];        ind2= y\_coords<=3;      x\_coords(ind2)= [];      y\_coords(ind2)= [];        plot(x\_coords, y\_coords, 'k--')      title('Greater than 3 Plot')      xlabel('x-axis')      ylabel('y-axis')    end |

Write a function that takes in a vector of x values and a vector of y values of the four corners of a square. It should create a figure using subplot with four plots in it, in 2x2 format. The upper-left plot should be the original plot in blue. The upper-right should be the original rotated about the origin by 90 degrees in green. The lower-left should be the original rotated by 180 degrees in yellow. The lower-right should be rotated about the origin by 270 degrees in red. Each of these plots should be graphed with the axes -2 to 2 in both the x and y directions.

function rotPlot(x,y)

x=[x x(1)];

y=[y y(1)];

subplot(2,2,1)

plot(x,y)

axis([-2 2 -2 2])

subplot(2,2,2)

rot90=[cos(pi/2) -sin(pi/2); sin(pi/2) cos(pi/2)];

new=rot90\*[x;y];

plot(new(1,:),new(2,:),'g')

axis([-2 2 -2 2])

subplot(2,2,3)

rot90=[cos(pi) -sin(pi); sin(pi) cos(pi)];

new=rot90\*[x;y];

plot(new(1,:),new(2,:),'y')

axis([-2 2 -2 2])

subplot(2,2,4)

rot90=[cos(3\*pi/2) -sin(3\*pi/2); sin(3\*pi/2) cos(3\*pi/2)];

new=rot90\*[x;y];

plot(new(1,:),new(2,:),'r')

axis([-2 2 -2 2])

end

**Recursion**

Write a function using recursion that finds a value inside nested cells and outputs that value. If there is no value inside the cells, output an empty cell (seen in Test case 2). (Note: If the input into your function, isn't a cell, just output the input.

function thing=nestedCells(cell)

if ~iscell(cell)    %if it's not a cell, we're done

        thing=cell; %and the thing is what we have left

else

    if isempty(cell)    %this happens if cell is still a cell, but it is empty

        thing={};       %if it's empty, we can't go any further and so the nested

                        %thing is just an empty cell

    else

    thing=nestedCells(cell{1}); %call the function for the cell with one of

                                %it's "layers" removed

    end

end

end

Given a vector of numbers, use recursion to sort the numbers in descending order.

function sorted= recurSort(vector)

    if isempty(vector)

        sorted=[];

    else

        [min1 pos]= min(vector);

        vector(pos)= [];

        sorted= [recurSort(vector) min1];

    end

end

Write a function that plots circles of decreasing size recursively, counting the circles as it goes. You are given the space between the circles and the maximum radius. Start by plotting the largest circle using the maximum radius input and continue graphing circles inside themselves. The circles should be equally spaced by the given value in the first input.

function num = bullseye(space,R)

if R < space/2;

    %We want the last circle to have a diameter of at least "space", so if

    %R is less than half of "space", we want to stop recursively going

    %through the function by setting num to 0.

    num=0;

else

th=linspace(0, 2.\*pi,100); %assign theta as instructed

x=R.\*cos(th);

y=R.\*sin(th);

plot(x,y, 'b--')

hold on         %hold on keeps plots from being over-written

axis square

num=bullseye(space,R-space)+1;

%go back through the function with the same "space" value, but reduce R by

%the "space" value. Remember the +1 is here because once you get an answer

%of 0 for "num" that you will want to add 1 to num for every time you

%plotted a new circle.

end

end

Function Description- Given a string: 1) replace all the vowels (a,e,i,o,u) with the letters 'LO'. 2) replace the constants within b-m to the letter 'L'. 3) replace the constants within n-z to the letter 'S'. \*\* you may assume the input will be only lower case letters \*\* you MUST use recursion.

function out= letterReplace(string)

    if isempty(string)

        out = [];

    elseif string(1) == 'a'|string(1)=='e'|string(1)=='i'|string(1)=='o'|string(1)=='u'

        out= ['LO' letterReplace(string(2:end))];

    elseif string(1)>='b' & string(1)<='m'

        out= ['L' letterReplace(string(2:end))];

    elseif string(1)>='n' & string(1)<'z'

        out= ['S' letterReplace(string(2:end))];

    end

end

Given a vector as input recursively sum each element of that vector and return the sum of all the elements of that vector as your output.

function nSum = recurSum(vec)

nSum = 0;

if isempty(vec)

   nSum = nSum;

elseif length(vec) == 1

   nSum = vec(1);

else

   nSum = nSum + recurSum(vec(2:end));

end

Function Description: Write a function that takes in a cell array of plotting values in the style of {x1, y1, str1, x2, y2,...}. There is guaranteed to be a vector of y values for every vector of x values and these should be plotted y vs. x. However, there is not guaranteed be a style string for every plot and there can be any number of x and y combinations. All plots should be plotted on the the same figure.

function recurPlot(in)

if length(in)==2

    plot(in{1},in{2})

elseif length(in)==3

    plot(in{1},in{2},in{3})

elseif ischar(in{3})

    plot(in{1},in{2},in{3})

    hold on

    recurPlot(in(4:end))

else

    plot(in{1},in{2})

    hold on

    recurPlot(in(3:end))

end

Write a function that takes in x values and recursively plots y versus x raised to exponents ranging from 1 to n, where n in the second input. n is guaranteed to positive.

function out=powerPlot(x,n)

if n==1

    plot(x,x)

    hold off

else

    plot(x,x.^n)

    hold on

    powerPlot(x,n-1)

end

**String**

Write a function LetterCmp which takes in two strings. The function returns the letters which are in the same spot (index) for both strings. It is case sensative. You may assume the longer of the two words will always be the first input.

function out = LetterCmp(str1, str2)

    str1= str1(str1>='a' & str1 <= 'z' | str1>='A' & str1<='Z');

    str2= str2(str2>='a' & str2 <= 'z' | str2>='A' & str2<='Z');

    positions= find(str1(1:length(str2))==str2);

    out= str1(positions);

end

Write a function that takes in a command written in all capital letters and ending in exclamation points. Change the command to all lowercase, remove the exclamation points, and insert it into this sentence: 'Would you please \_\_\_\_\_\_\_\_\_\_?'

|  |
| --- |
| function nice=askNicely(str)  logic=str>='A' & str<='Z';  str(logic)=str(logic)+32; %32 is the difference in ascii value b/t uppercase and lowercase  str=strtok(str,'!');  nice=sprintf('Would you please %s?', str);  end |

Write a function that takes a string containing "w/" (a common abbreviation for "with") and transforms it into the word "with". (You can assume that "w/" only occurs once in the input string, and that it is also the only time the "/" symbol occurs.)

function str=withFix(str)

[first second]=strtok(str,'/');

second=second(2:end); %take second(2:end) because second(1) is the '/'

with='ith'; %because the 'w' is already contained in "first"

str=[first with second];

end

**Structure Arrays**

Write a function that takes in a vector and creates a struct with the following fields: (1) Positives (2) Negatives. Make sure the positive field comes first. Also, make sure the numbers are listed in ascending order (within their field).

function out=numCategorize(vec)

Ncount=1;

Pcount=1;

out.Positives=[]; %this guarantees that this field will come first

out.Negatives=[];% This guarantees that this field will exist

vec=sort(vec);

for x=vec

    if x>=0

        out(Pcount).Positives=x;

        Pcount=Pcount+1; %This updates the position for the next number that belongs in the Positives field

    end

    if x<0

        out(Ncount).Negatives=x;

        Ncount=Ncount+1; %This updates the position for the next number that belongs in the Negatives field

    end

end

Write a function that takes in a cell containing a cell of strings and a logical vector and categorizes the strings that are true statements and the strings that are false statements. The first field of the output structure is "Truth", and the second field is "Lies".

function out=truthNlies(cA)

out.Truth=[];

out.Lies=[];

T=1;

L=1;

logicals=cA{2};

for x=1:length(logicals)

    if logicals(x)==true

        out(T).Truth=cA{1}{x};

        T=T+1; %keeps count for indexing the next true string into the structure

    else

        out(L).Lies=cA{1}{x};

        L=L+1; %keeps count for indexing the next false string into the structure

    end

end

Function Description: -You are given a structure array with the fields 'name' and 'GPA'. -You may assume that no two GPA's will be alike. -Find the person who as the highest GPA. -Then create a new 1 by 1 structure which will contain only the person with the highest GPA in the 'GPA' field and their respective name in the 'name' field.

function new\_sA= findHighestGPA(sA)

    [max\_num ind]= max([sA.GPA]);

    person= sA(ind).name;

    new\_sA= struct('name', person, 'GPA', max\_num);

end

Function Description: Create a function that will take in a structure and a string containing the student's name. Then, using the structure, find out the student's age and see if student is at least 21. If the student is 20 or younger say 'Sorry you are under age.' However if the student is 21 or older say 'Bottoms Up!'.

function out = drinkingAge(s,name)

for i = 1:length(s)

    if strcmp(s(i).names,name)

        if s(i).age >= 21

            out = 'Bottoms Up!';

        else

            out = 'Sorry you are under age.';

        end

    end

end

end

Function Description: Create a function that will take in a structure and string. If the string says 'names' , the structure will be sorted alphabetically from a-z . If the string says 'age;, the structure will be sorted numerically ascending order.

|  |
| --- |
| function out = sortStruct(s,string)  if strcmp(string,'names')      sNames = {s.names};      [dummy ind] = sort(sNames);    else      sAge = [s.age];      [dummy ind] = sort(sAge);  end       out = s(ind);  end |

**Vector**

Write a function that takes in two vectors, reverses the order of their contents, and then concatenates them with a vector of five 5's in between them.

|  |
| --- |
| function out=vecFives(v1, v2)  v1=v1(end:-1:1);  v2=v2(end:-1:1);  fives=ones(1,5).\*5;  out=[v1 fives v2];  end |

Function Description: Given a vector of numbers, write a function "SortRotate" which will eliminate any number in the vector greater than 20. The function will then sort the remaining numbers, and create a mirror image of itself.

function out = SortRotate(vector)

    a= vector<20;

    b=vector(a);

    sort\_b= sort(b);

    c= sort\_b(end:-1:1);

    out= [sort\_b c];

end

Write a function that takes in a vector of test grades. Drop the lowest test score, and then "curve" the grades by adding 5 points to any failing grade (in other words, any grade below 70).

function out=superCurve(grades)

[value spot]=min(grades);

grades(spot)=[];

lowgrades\_logical=grades<70;

grades(lowgrades\_logical)=grades(lowgrades\_logical)+5;

out=grades;

end

Write a function VecMix, which will intertwine two input vectors. That is, the first element in the first vector input vector is the first element in the new vector. The first element in the second input vector is the second element in the new vector. The second element in the first vector is the third element in the new vector and so on. If one vector is longer than the other, put 0's as place holders.

function out = VecMix(arr1, arr2)

    total= [arr1 arr2];

    length\_total= length(total);

    out= zeros(1,length\_total);

    c=length(arr1).\*2;

    d=length(arr2).\*2;

    out(1:2:c)=arr1(1:1:end);

    out(2:2:d)=arr2(1:1:end);

end

Function Description: Write a function that will take in a vector as its input and then it will return a vector whose 1st element will the total number of odd values (NOT POSITIONS) , the 2nd element will be the total number of even values (AGAIN NOT POSITIONS), and for the 3rd and final element , the total number of elements in that vector. The output vector again should have a total of 3 elements.

function newVec = oddsAndEvens(vec)

odds = sum(mod(vec,2)==1);

evens = sum(mod(vec,2)==0);

total = length(vec);

newVec = [ odds evens total];

end

Function Description: Write a function that will take in a vector and will first output a row vector with numbers between 5 and 10 inclusive removed from the original vector. (Inclusive means you include the numbers 5 and 10). The second output should be a vector that remove all the numbers inside the original vector that are *not* in the interval 5 to 10 (exclusive). (Exclusive means you do not include the numbers 5 and 10).

function [out1 out2] = range5and10(vec)

out1 = vec(vec >= 5 & vec <= 10);

out2 = vec(vec < 5 | vec > 10);

end

**Sorting**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Type Sort | Big "O" | In-Place | Form | Page/use |
| Insertion | N^2 | no | Single Loop | Page 372/ Used when adding to sorted data |
| Bubble | N^2 | yes | Nested For Loops | Page 374/ Used for sorting short vectors, in place |
| Merge | NlogN | no | Recursive  Recursive  Merge | Page 378/ used for merging sorted data |
| Quick | NlogN | yes | Partition  Recursive  Recursive | Page 378? Used for random data, not good for sorted. |

🡪set up curve: Z-Axis

u=1:10;

v=u.^2;

%set and angle around which to rotate, we’ll do 360 degrees

th = linspace(0, 2\*pi, 100);

%set up your mesh of the function so it is in 3D

[uu tth] = meshgrid(u, th);

vv=uu.^2;

% Data for rotation around the z axis, remember, when doing z axis straights with straights and curvies with curvies:

rr = uu; %curvy letters go together

xx = cos(tth).\*rr; %when rotating around x and z whether xx or yy

yy = sin(tth).\*rr; is set to cos or sin is interchangeable.

zz = vv; %straight letters go together

mesh(xx, yy, zz) % plot it

X-Axis

u=1:10

v=u.^2;

th = linspace(0, 2\*pi, 100);

[uu tth] = meshgrid(u, th);

vv=uu.^2;

rr = vv;

xx = uu;

yy = sin(tth).\*rr;

zz = cos(tth).\*rr;

surf(xx, yy, zz)

No function just points

u= 1:25;

v = 25:50;

%in this case we set up two mesh grids, one for each variable. Each mesh grid essentially represents that particular u or v coordinate being paired with every angle represented in theta (here from 0 to 2pi) to be plotted.

th = linspace(0, 2\*pi, 120);

[uu tth] = meshgrid(u, th);

[vv tth] = meshgrid(v, th);

rr = uu;

xx = rr .\* cos(tth);

yy = rr .\* sin(tth); %which axis did I rotate around here?

zz = vv;

surf(xx, yy, zz)