# Exercises 2b: MAE Unit 2

**Name:** Josep Famadas Alsamora

**Collaboration Info:** I have used some information from mathworks.com.

---

## Exercise 6. Morse code.

Explanation:

(1) The function gets the data from 'morse.mat,', and puts the input argument to capital letters. Then, with a for that iterates the string, it adds the suitable number of spaces. After that, it puts the string in the output parameter with a space between every element.
Finally, using the function strrep it changes the dots/dash/spaces for its number of 0/1.

(2) This function gets the pulse sequence and adapts it in order to be able to use the function 'sound' with it.

(3) This script generates a text, calls both functions created before to play the Morse sound of it and sends out on screen the original text and the text coded.

MATLAB Code:

(1)

```
function [ dashdot_seq, pulse_seq ] = Morse_encoder( input_string )
%UNTITLED Summary of this function goes here
%   Dot                 => '1'
%   Dash                => '111'
%   Separator Dot/Dash  => '0'
%   Next letter         => '000'
%   Next word           => '0000000'  (7 0s)

load('morse.mat');
str = upper(input_string);
string = '';


for i=1:length(str)

    switch str(i)
        case ' '
            string = [string, '  '];       %2 spaces that will be
converted in 7 later
        otherwise
            string = [string, ' ', char(Morse(find(Alpha==str(i))))];
%1 space that will be converted in 3 later
    end

end
```

```
dashdot_seq = blanks(2*length(string)-1); % this is to make the space
for ' '
dashdot_seq(1:2:end) = string;          %we add the text leaving the
spaces




str2 = dashdot_seq;
str2 = strrep(str2,' ','0');
str2 = strrep(str2,'.','1');
str2 = strrep(str2,'-','111');

pulse_seq = str2 - '0';

end
```

(2)

```
function [] = Morse_beep( pulse_seq, tone_freq, dot_duration,
sampling_freq )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

dot_samples=dot_duration*sampling_freq;
pulses = repmat(pulse_seq, [dot_samples,1]);    %it replies the
pulse_seq veticaly.

tone = reshape(pulses,[length(pulse_seq)*dot_samples,1]);   %now this
is the pulse_seq terms repited
tone = tone';   %it was vertical

t=0:1/sampling_freq:(length(tone)/sampling_freq)-(1/sampling_freq);
tone_final = tone.*cos(2*pi*tone_freq*t);
sound(tone_final,sampling_freq);

end
```

(3)

```
%% (3)

text = 'MAE-Unit b2'
dot_duration = 0.05;
t_f = 900;
s_f = 8000;
```

```
[dashdot_seq, pulse_seq] = Morse_encoder(text);

dashdot_seq

Morse_beep(pulse_seq, t_f, dot_duration, s_f);
```

<u>Results:</u>

The written results are:

text =

MAE-Unit b2

dashdot_seq =

```
 -- .- . -....- ..- -. .. -   -... ..---
```

<u>Coments:</u>

To do the second part I asked to David Llaveria for some help because I had no idea of how to adapt a bit sequence to be played as sound.

## Exercise 7. Koch fractal curve.

Explanation:

(1) The function uses two iterations. The big one iterates 'n' times and each of them generates a gap to between every couple of points and applies the triangles to every gap with the little for.
The strategy I used for the small iterator is to define 4 different types of point (A,B,C and D) that corresponds to the points of the original triangle matrix (the $5^{th}$ point of the matrix is the point A from the next triangle). With this idea, I could move in the first iteration all the A-type points, in the second the B-type…

(2) This second function is a "expansion" of the first. The changes are that now I do not create the matrix containing the figure to be plotted, it is now an input parameter. And the other diference is that the small for goes until length(M)-1.

MATLAB Code:

(1)

```
function [ M ] = koch( n )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

M1 = [0 0; 1/3 0; 1/2 sin(pi/3)/3; 2/3 0; 1 0];
M1x = M1(:,1);
M1y = M1(:,2);
Fx2 = M1x;
Fy2 = M1y;

for i=1:n

    Fx=Fx2;
    Fy=Fy2;
    u0 = Fx(1:4:end-1);
    u1 = Fx(5:4:end);
    v0 = Fy(1:4:end-1);
    v1 = Fy(5:4:end);

    for j = 1:4
    Fx(j:4:end-1) = u0+(u1-u0)*M1x(j)-(v1-v0)*M1y(j);
    Fy(j:4:end-1) = v0+(v1-v0)*M1x(j)+(u1-u0)*M1y(j);
    end

Fx2=zeros(length(Fx)*4-3,1);
Fy2=zeros(length(Fy)*4-3,1);

Fx2(1:4:end)=Fx;
Fy2(1:4:end)=Fy;


end

if n == 0

    M = [0 0; 1 0];
```

```
else

    M = [Fx,Fy];
end

if nargout == 0
    plot(Fx,Fy);
    axis equal;
end
end
```

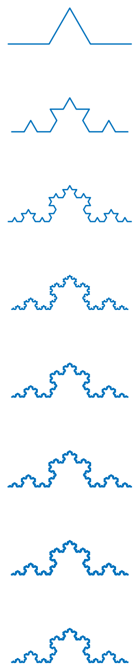
(2)

```
function [ M ] = genkoch( n, M1 )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

M1x = M1(:,1);
M1y = M1(:,2);
Fx2 = M1x;
Fy2 = M1y;

for i=1:n

    Fx=Fx2;
    Fy=Fy2;
    u0 = Fx(1:length(M1)-1:end-1);
    u1 = Fx(length(M1):length(M1)-1:end);
    v0 = Fy(1:length(M1)-1:end-1);
    v1 = Fy(length(M1):length(M1)-1:end);

    for j = 1:length(M1)-1
    Fx(j:length(M1)-1:end-1) = u0+(u1-u0)*M1x(j)-(v1-v0)*M1y(j);
    Fy(j:length(M1)-1:end-1) = v0+(v1-v0)*M1x(j)+(u1-u0)*M1y(j);
    end

Fx2=zeros((length(Fx)-1)*(length(M1)-1)+1,1);
Fy2=zeros((length(Fy)-1)*(length(M1)-1)+1,1);

Fx2(1:length(M1)-1:end)=Fx;
Fy2(1:length(M1)-1:end)=Fy;

if nargout == 0
    figure(1)
    subplot(n,1,i);
    plot(Fx,Fy);
    axis equal;
    axis off;
end

end

if n == 0

    M = [0 0; 1 0];
```

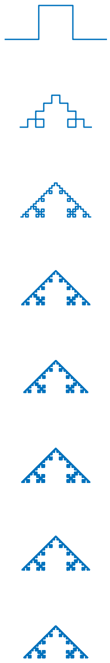
```
else

    M = [Fx,Fy];
end


end
```

Results:

The result for the triangle is:



The result for the rectangle is:



Comments:

I and David Llaveria had some problems to solve this exercise, so we did it together.

## Exercise 8. ULA antenna gain diagram

Explanation:

(1) This function computes the values of AF in function of Psi and returns them in the output parameter.
(2) This function creates the theta vector with Npoints samples, and computes the values of the gain as the absolute value of the return of the previous function.
(3) This function computes different parameters:
    a. Theta_max: the theta value in which the gain has its maximum value.
    b. Delta_theta: calculate the thetas in which the gain values I >= than the max/sqrt(2)
    c. D: calculates the directivity using the Definite_integral function from the previous task.
(4) It uses the function compute gain to get the theta and gain values, then plots gain values in function of 2*pi*d/lambda*cos(theta).
(5) This function first of all transform the gain into dB and cuts its values < min_db. Then uses the function polarplot to plot in polar coordinates the gain and sets the plot limits with rlim.
(6) This is the main function. With a given values of 'a', 'd', 'Npoints' and (min_db) uses the previous functions to plot the gain in lineal and polar, and gives the parameter values.

MATLAB Code:

(1)

```
function [ AF_values ] = compute_AF( a, Psi_values )
%UNTITLED2 Summary of this function goes here
%   Detailed explanation goes here

n = 0:(length(a)-1);

A = zeros(1,length(Psi_values));

for i = 1:length(Psi_values)

    A(i) = sum(a.*exp(1i*n*Psi_values(i)));

end

AF_values = A;


end
```

(2)

```
function [ theta_values, gain_values ] = compute_gain( d_norm, a,
Npoints )
%UNTITLED3 Summary of this function goes here
```

```
%   Detailed explanation goes here

theta_values = -pi:2*pi/(Npoints-1):pi;
Psi_values = 2*pi*d_norm*cos(theta_values);
gain_values = abs(compute_AF(a,Psi_values));


end
```

(3)

```
function [] = radiation_pattern_parameters( theta_values, gain_values
)
%UNTITLED4 Summary of this function goes here
%   theta_max = maximum gain direction
%   delta_theta = beamwidth
%   D = Directivity

    theta_max = theta_values(find(gain_values==max(gain_values)))

    main_lobe =
theta_values(find(gain_values>=max(gain_values)/sqrt(2)));
    main_lobe = main_lobe(1:end/length(theta_max));
    delta_theta = main_lobe(end)-main_lobe(1)


    x = theta_values(theta_values>=0);
    fx = gain_values(theta_values>=0).^2.*sin(x);
    integral = Definite_integral(x,fx);
    D = 2*max(gain_values)^2/integral




end
```

(4)

```
%% (4)

a1 = [1 1 1 1];
a2 = [1 exp(1i*3*pi/4) exp(1i*6*pi/4) exp(1i*9*pi/4)];
d_norm = 1/2;
Npoints = 1000;

[theta_values, gain_values] = compute_gain(d_norm, a1, Npoints);

figure(1)
plot(2*pi*d_norm*cos(theta_values),gain_values);
grid on;
```

(5)

```
function [] = polardB( theta_values, gain_values, min_db )
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here

gain_values_db = 20*log10(gain_values/max(gain_values));

gain_values_db = max(gain_values_db, min_db);
polarplot(theta_values,gain_values_db);
rlim([min_db,0]);

end
```

(6)

```
% (6)

a1 = [1 1 1 1];
a2 = [1 exp(1i*3*pi/4) exp(1i*6*pi/4) exp(1i*9*pi/4)];
d_norm = 1/2;
Npoints = 10001;
min_dB = -20;

[theta_values, gain_values] = compute_gain(d_norm, a2, Npoints);

%Linear
figure(1)
plot(2*pi*d_norm*cos(theta_values),gain_values);
grid on;

%Polar (dB)
figure(2)
polardB(theta_values, gain_values, min_dB);

%Parameters
radiation_pattern_parameters( theta_values, gain_values);
```

Results:

With a = [1 1 1 1] the results are:

theta_max =

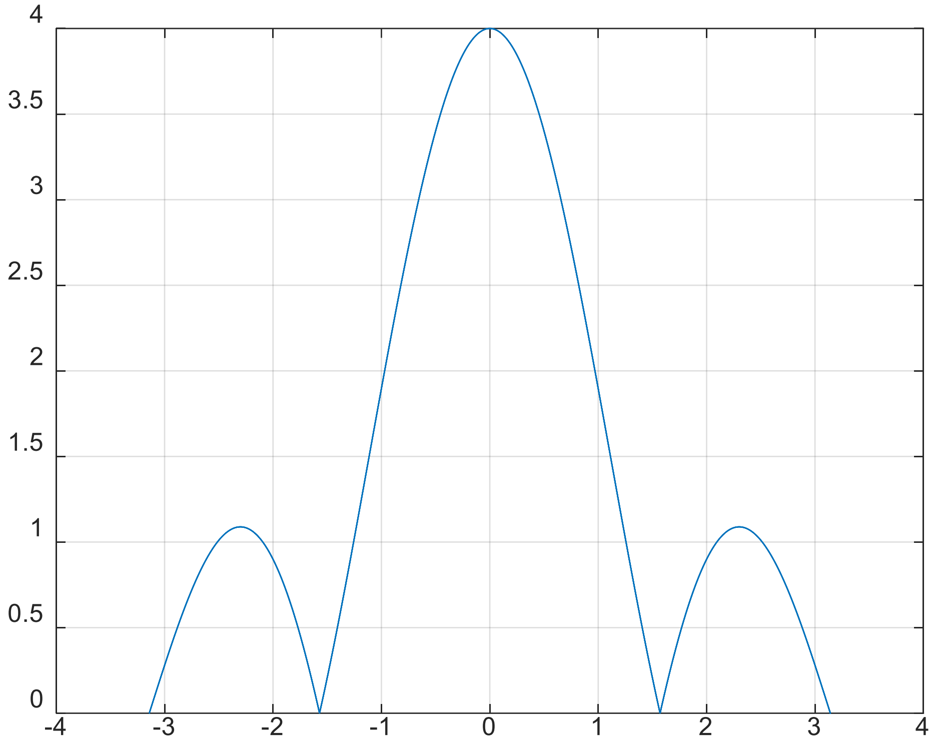
-1.5708    1.5708

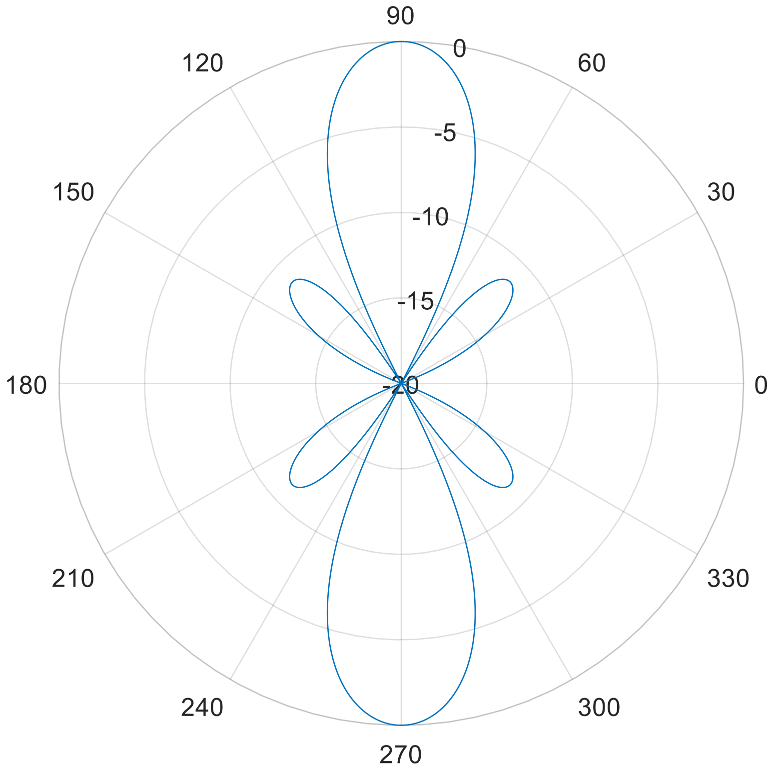
delta_theta =

0.4587

D =

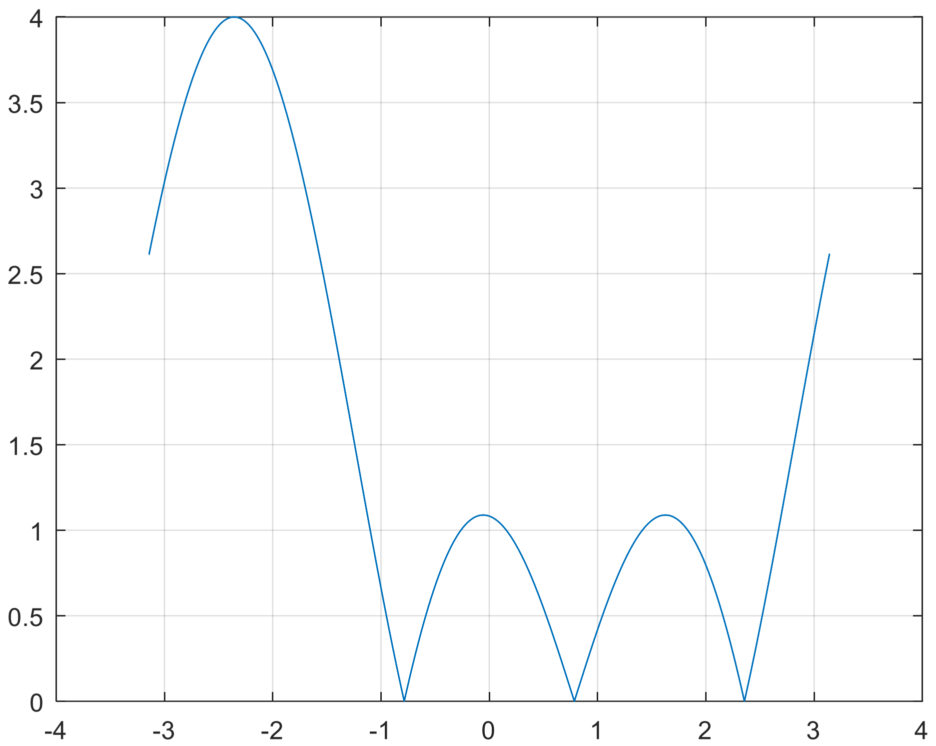
4.0000

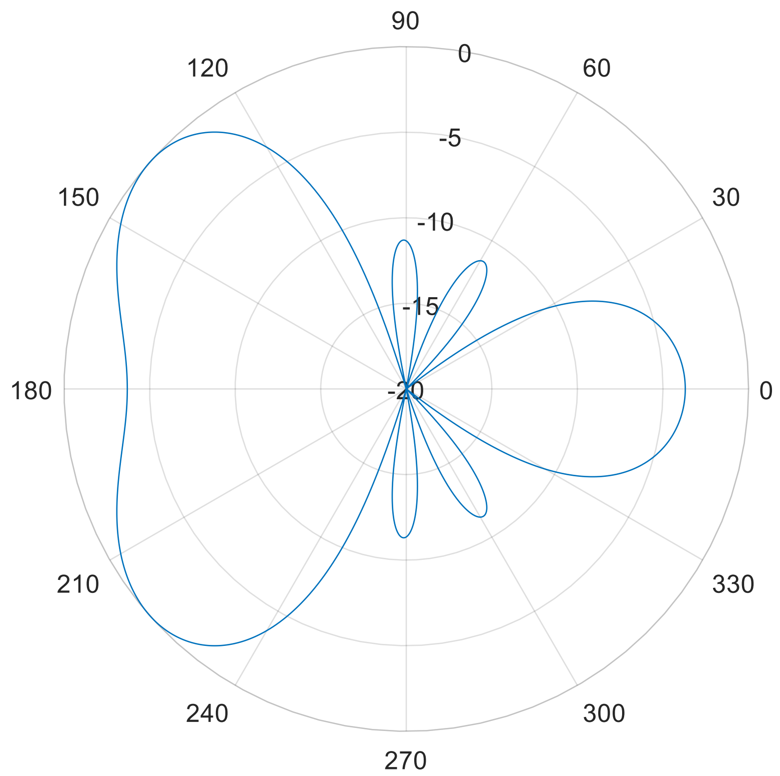





And for a = `[1 exp(1i*3*pi/4) exp(1i*6*pi/4) exp(1i*9*pi/4)] they are:`

```
theta_max =

   -2.4190    2.4190


delta_theta =

    0.8093


D =

    4.0000
```




<u>Comments:</u>

The function to calculate the delta_theta only works if the ratio between main and secondary lobe is higher than 3dB.

## Exercise 9. Input and output argument functions.

Explanation:

The script ‘cones.m’ evaluates the different cases of input arguments with the functions ‘if’ and ‘switch’ and, if the number of input arguments is 2, it differentiates between the case when the second is a number and when the second is a char or char array.

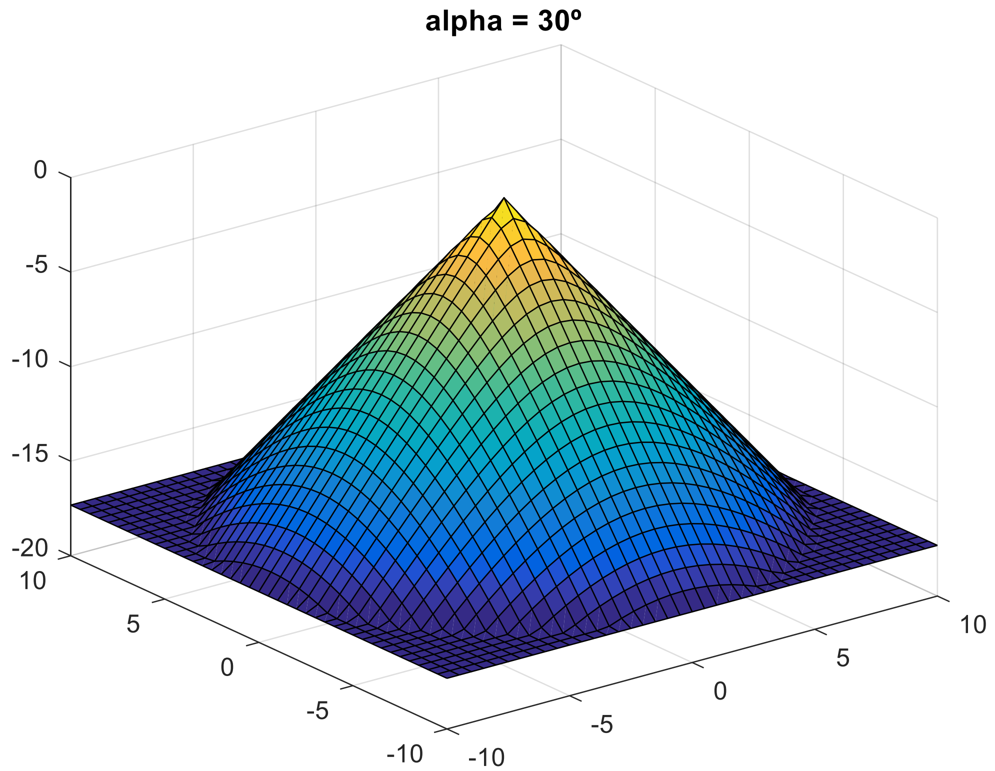
MATLAB Code:

```
function [xx, yy, z ] = cones( arg1, arg2, arg3 )
% » help cones
% CONES: function that plots cones and cone cuts.
% [xx,yy,z]=cones(alpha)     Compute a cone with vertex at the origin
and
%                            vertex angle equal to alpha (degrees)
%                            To plot it use surf(xx,yy,z) or
contour(xx,yy,z)
% [xx,yy,z]=cones(alpha,beta) Compute the cone cut using a plane with
angle
%                            beta (degrees)
% cones(alpha,'3D') or cones(alpha,beta,'3D'): 3D plot of the cone or
the cone cut
% cones(alpha,'C') or cones(alpha,beta,'C'): contour plot of the cone
or the cone cut


    alpha = arg1*pi/180;
    x = -10:0.5:10;
    y = x;

    [xx,yy] = meshgrid(x,y);

    z = -sqrt(xx.^2+yy.^2)/tan(alpha);
    z(find(z<z(1,21))) = z(1,21);





    if(nargin == 1)

        %cone without specific plot
        figure(1)
        surf(xx,yy,z);
        title(sprintf('alpha = %d°', arg1));
        figure(2)
        contour(xx,yy,z);

    elseif(nargin == 2)

        if(ischar(arg2))
            % cone with specific plot (arg2)
            switch arg2
                case '3D'
                    surf(xx,yy,z);
```

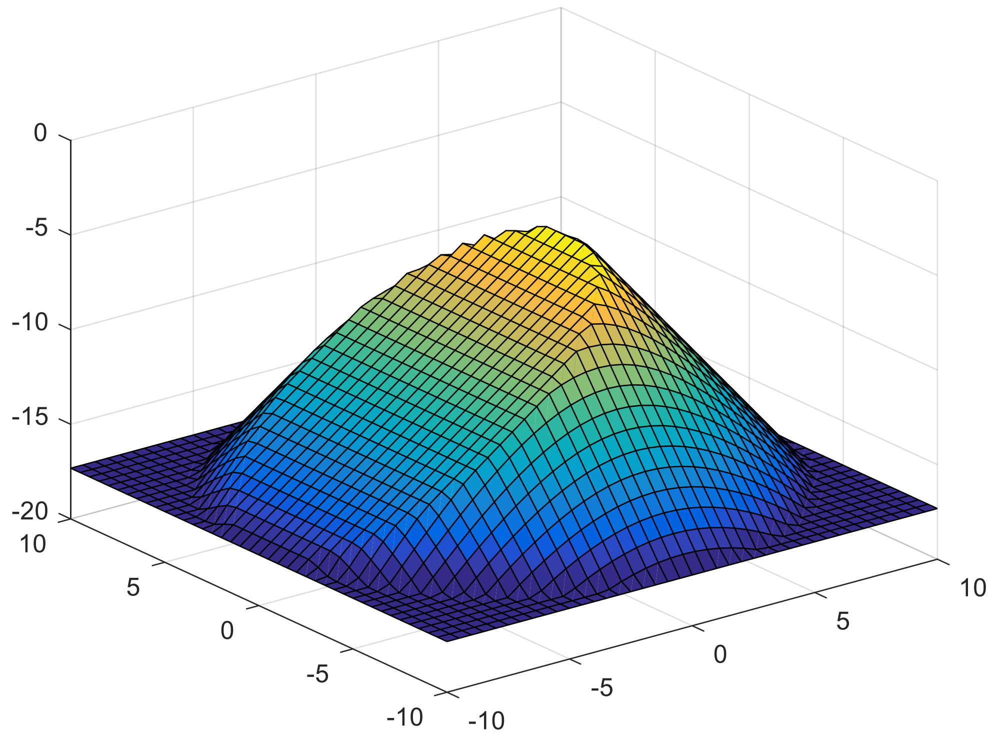
```
                title(sprintf('alpha = %d°', arg1));
            case 'C'
                contour(xx,yy,z);
        end


    elseif(isnumeric(arg2))

        %cut without specific plot
        beta = arg2*pi/180;
        z = min(z,tan(beta)*(abs(min(min(xx)))+xx)+ min(min(z)));



    end


elseif(nargin == 3)

    %cut with specific plot (arg3)
    beta = arg2*pi/180;
    z = min(z,tan(beta)*(abs(min(min(xx)))+xx)+ min(min(z)));
    switch arg3
            case '3D'
                surf(xx,yy,z);
                title(sprintf('alpha = %d° , beta = %d°', arg1,
arg2));
            case 'C'
                contour(xx,yy,z);
    end

end

end
```

Results:

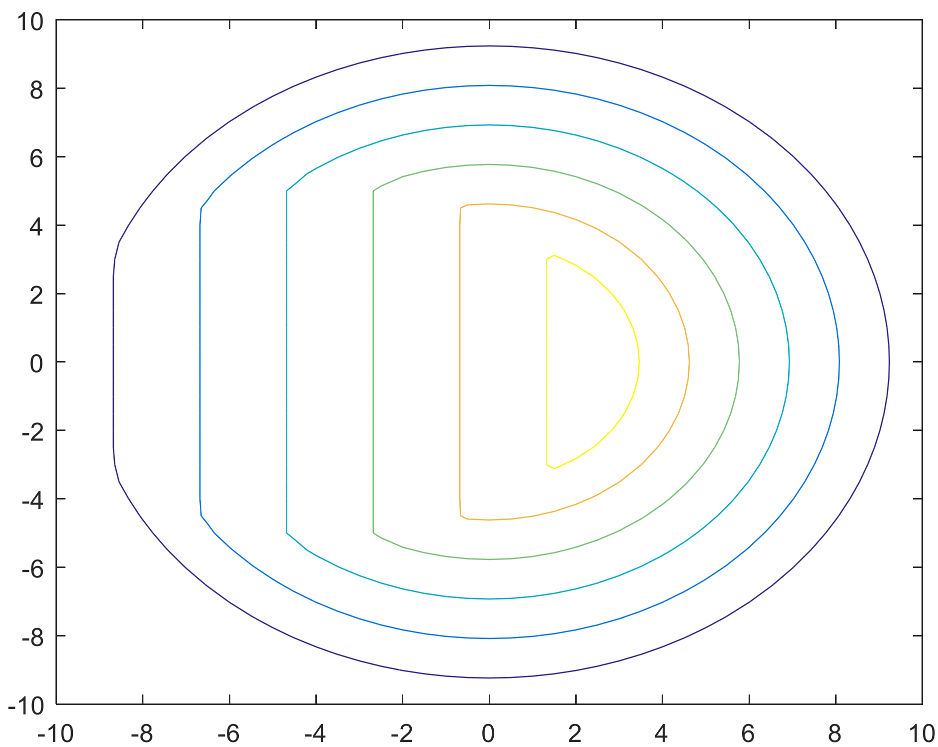
The result for `cones (30,'3D') is:`




For `[xx, yy, z]= cones (30,45);`
```
surf (xx,yy,z);
```




For cones (30,45,'C')




<u>Coments:</u>