Table of Contents

Practica 16 Casas Mercadé

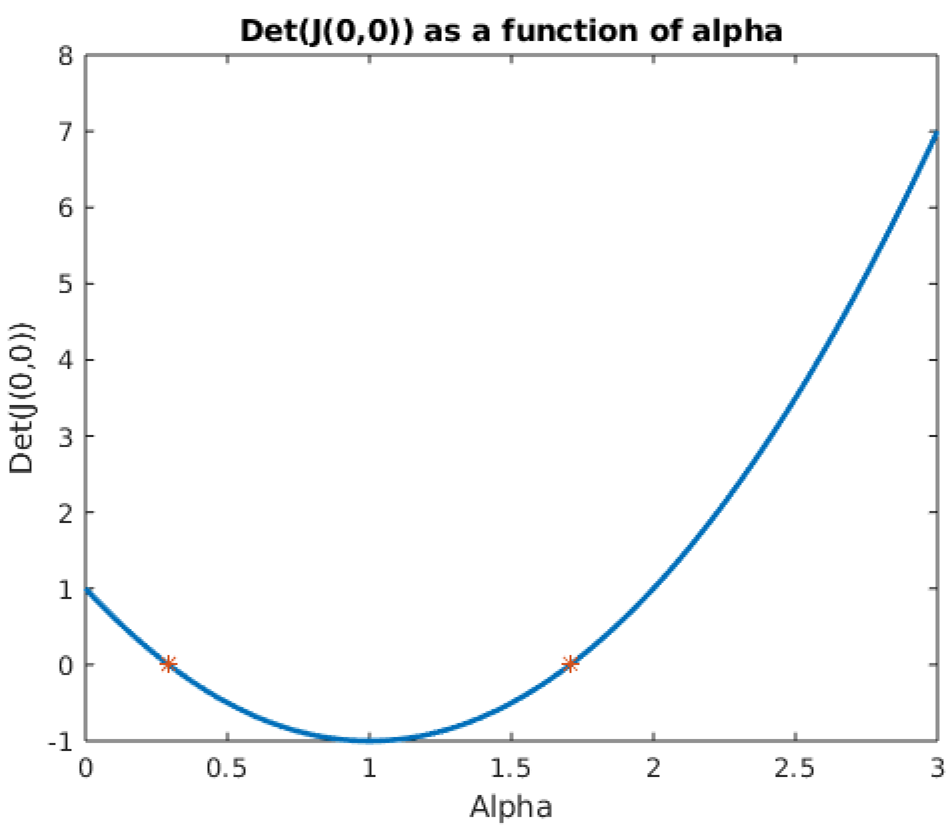
```
clear all
close all
clc
```

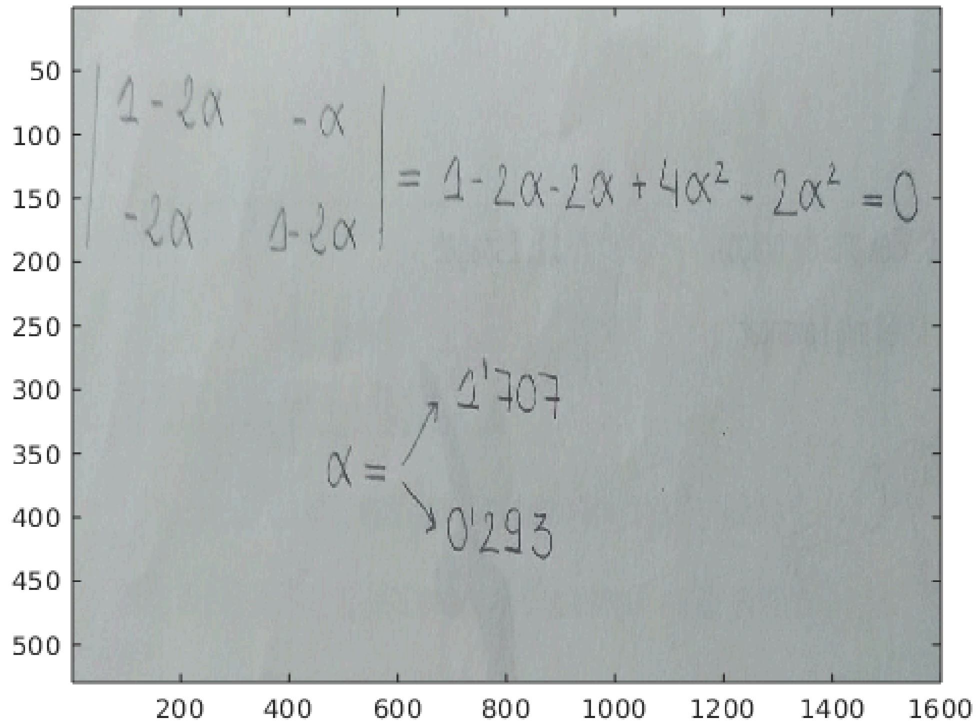
Section A)

```
determinants = [];
alphas = 0:0.01:3;

alphaZeros = [];
posicio = [];

figure(1)
i = 1;

for alpha = alphas

    f = @(phi)([tan(phi(1)) - alpha * (2 * sin(phi(1)) + sin(phi(2)));
 tan(phi(2)) - 2 * alpha * (sin(phi(1)) + sin(phi(2)))]);

    phi = [0, 0];

    j = jaco(f, phi);

    determinants = [determinants, det(j)];

    if abs(det(j)) < 0.01
        alphaZeros = [alphaZeros, alpha];
        posicio = [posicio i];
    end

    i = i + 1;
end

Det0 = [determinants(posicio(1)), determinants(posicio(2))];
plot(alphas, determinants, 'LineWidth', 2)
hold on
title('Det(J(0,0)) as a function of alpha')
xlabel('Alpha')
```

```
ylabel('Det(J(0,0))')
plot(alphaZeros, Det0, '*')
% The implicit function theorem (imft) states that as long as the
 jacobian
% is non-singular (det non zero) the system will define phi(1) and
 phi(2)
% as a unique functions of aplha, so we'll have a unique map between
% the solutions and alphas
% When the determinant is zero the uniqueness will be lost locally
 nearby
% the aplha points which make the determinant 0 and new branches of
% solution may emerge.


disp('The alpha values that make zero the determinant are more less:')
disp(alphaZeros)

% Anallitically:
figure;
img = imread('analitic.jpeg');
image(img);
```

The alpha values that make zero the determinant are more less:
0.2900   1.7100

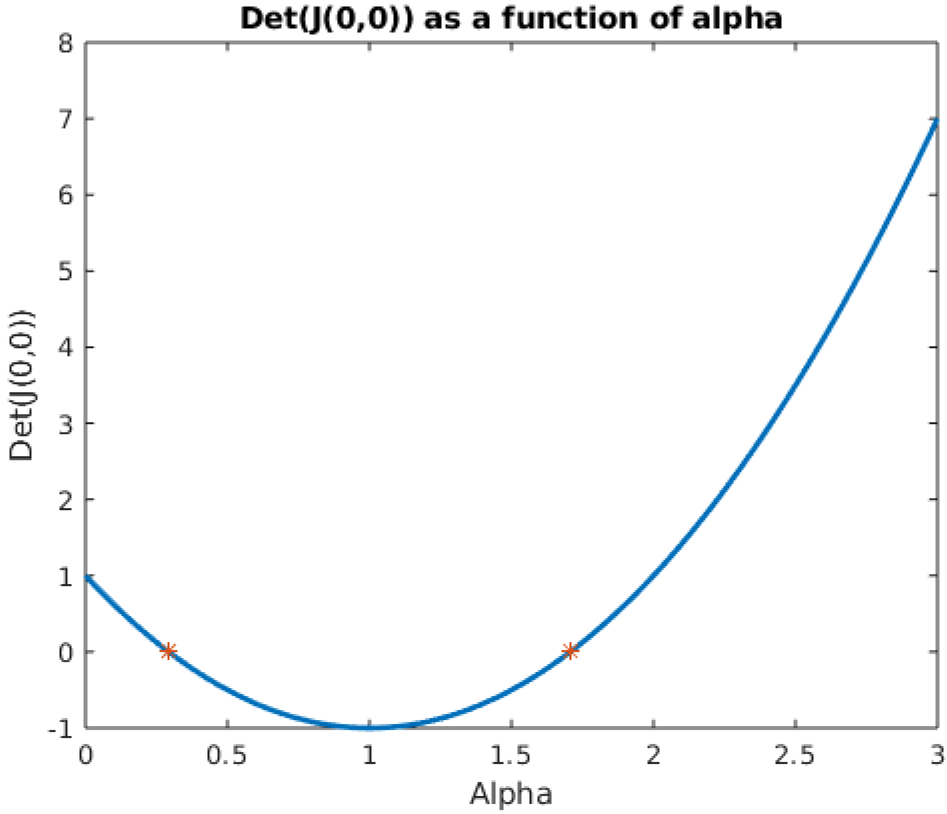


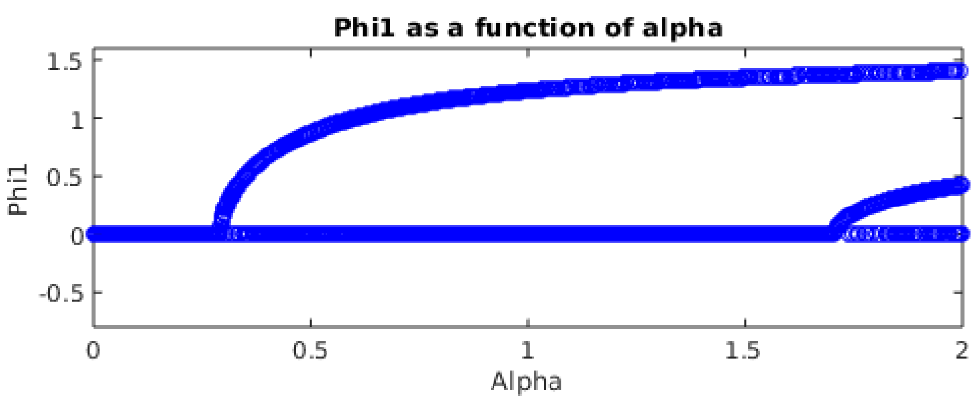


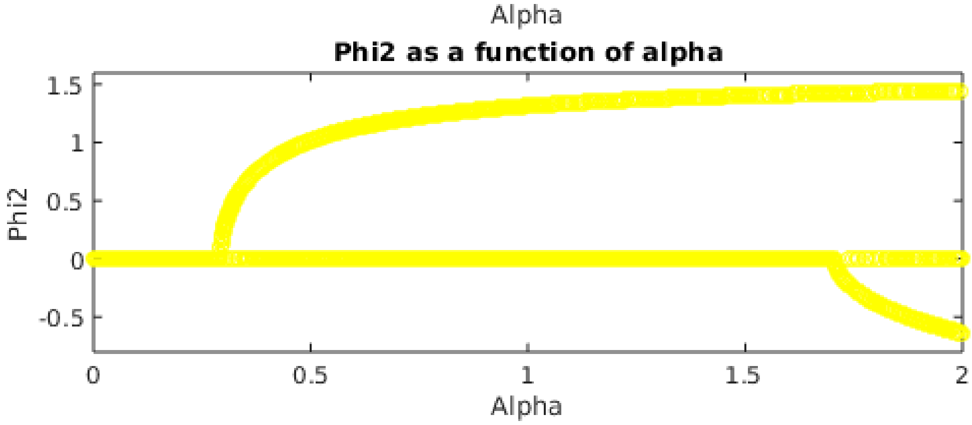
Section B)

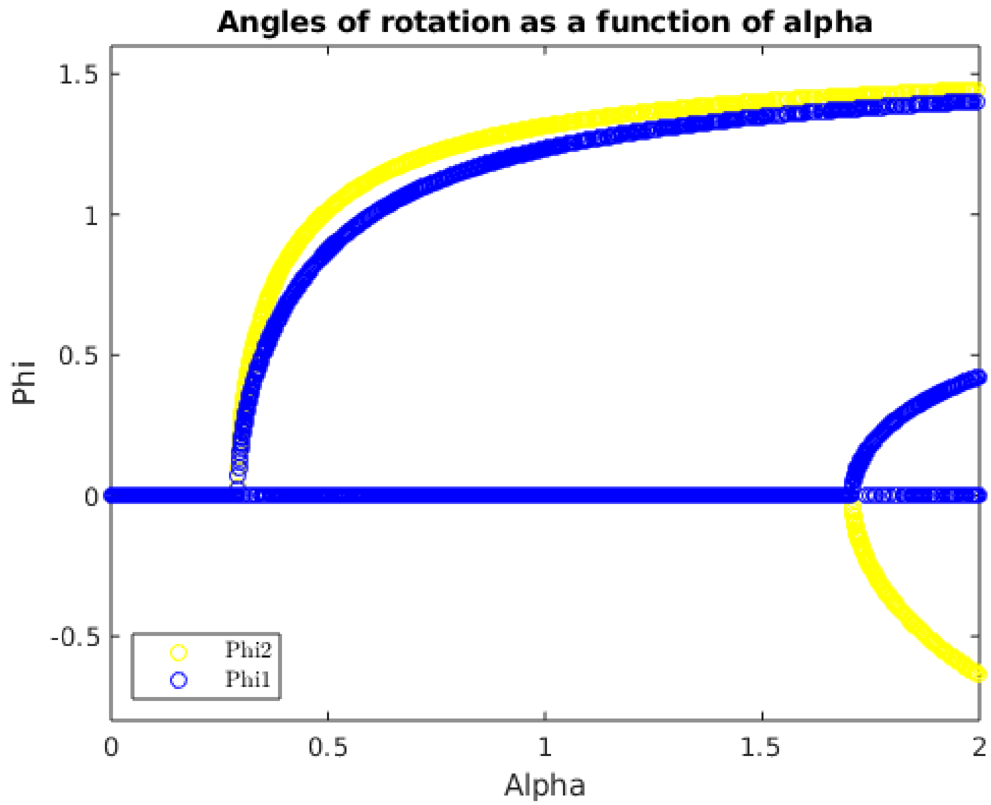
```
x = 0.001;
alphas = 0:x:2;
% Dominis dels angles
dom1 = [0, pi / 2];
dom2 = [-pi / 2, pi / 2];
factor = [dom1(2); (dom2(1) - dom2(2))];
a = [dom1(1); dom2(1)];
aleatoryTimes = 1:20;

sol = [];
alphasol = [];
figure(2)

for alpha = alphas
    f = @(phi)([(tan(phi(1)) - alpha * (2 * sin(phi(1)) +
 sin(phi(2)))), (tan(phi(2)) - 2 * alpha * (sin(phi(1)) +
 sin(phi(2))))]);

    for i = aleatoryTimes
        aleatory = rand(2, 1);
        phi0 = aleatory .* factor - a;
        %Formula per canviar de escala i moure:
        %x*(a-b) - a
```

```
        [XK, resd, it] = newtonn(phi0, 1e-6, 100, f);
        % Comprobar que estigui dintre el domini
        if XK(1, end) > dom1(1) && XK(1, end) < dom1(2) && XK(2, end)
 > dom2(1) && XK(2, end) < dom2(2)
            sol = [sol, XK(:, end)];
            alphasol = [alphasol, alpha];
        end

    end

end

subplot(2, 1, 1)
plot(alphasol, sol(1, :), 'o', 'Color', 'blue')
axis([0 2 -0.8 1.6])
title('Phi1 as a function of alpha')
xlabel('Alpha')
ylabel('Phi1')
subplot(2, 1, 2)
plot(alphasol, sol(2, :), 'o', 'Color', 'y');
axis([0 2 -0.8 1.6])
title('Phi2 as a function of alpha')
xlabel('Alpha')
ylabel('Phi2')

figure(3)
plot(alphasol, sol(2, :), 'o', 'Color', 'y');
hold on
plot(alphasol, sol(1, :), 'o', 'Color', 'blue')
axis([0 2 -0.8 1.6])
title('Angles of rotation as a function of alpha')
legend('Phi2', 'Phi1', 'Location', 'southwest', 'Interpreter', 'latex')
xlabel('Alpha')
ylabel('Phi')
hold off
```


Det(J(0,0)) as a function of alpha
Det(J(0,0))
Alpha
8
7
6
5
4
3
2
1
0
-1
0
0.5
1
1.5
2
2.5
3


Phi1 as a function of alpha
Phi1
Alpha
1.5
1
0.5
0
-0.5
0
0.5
1
1.5
2


Phi2 as a function of alpha
Phi2
Alpha
1.5
1
0.5
0
-0.5
0
0.5
1
1.5
2

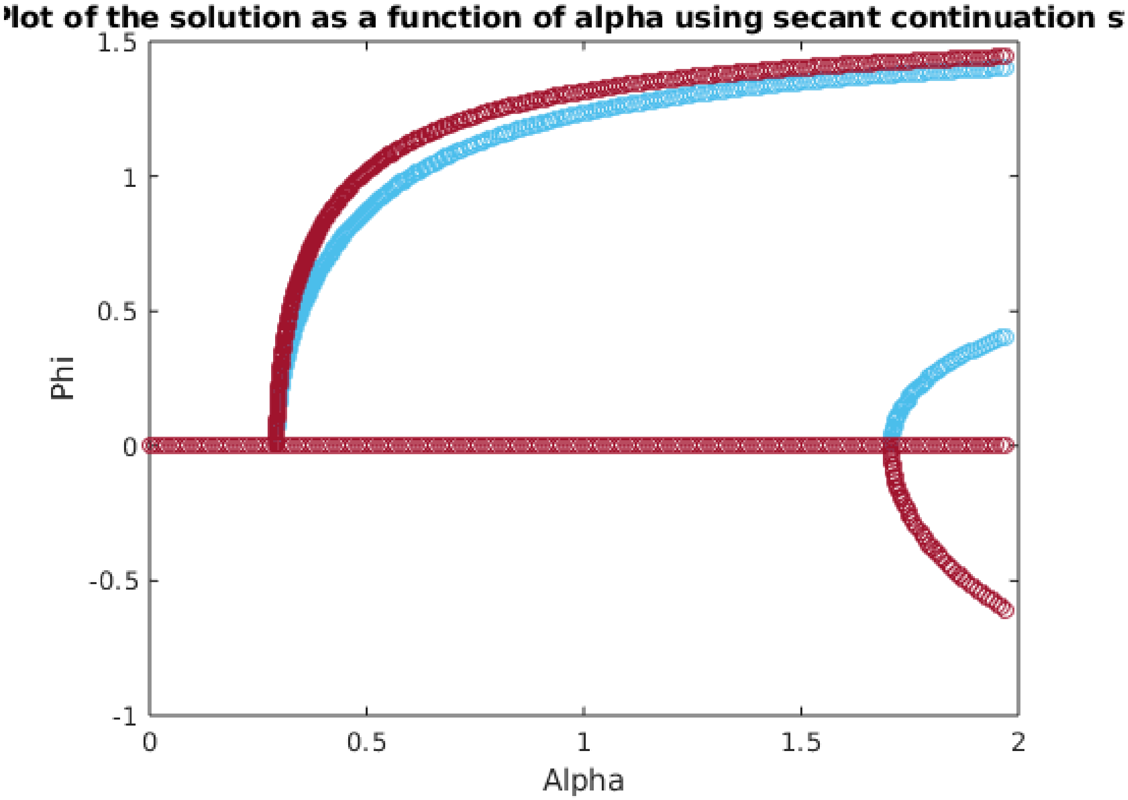


Section C)

```
%As it has been seen analitically there are two alpha's that make zero
 the
%jacobian determinant, this alphas are 0.293 and 1.707, so the in
 order to
%obtand all the solutions with the secant continuation step we will
 take 3
%difrent y0 and y1 at the right plot of the previous exercise.

funAlpha = @(y)([tan(y(1)) - y(3) * (2 * sin(y(1)) + sin(y(2)));
 tan(y(2)) - 2 * y(3) * (sin(y(1)) + sin(y(2)))]);
figure(4)
epsilon = 0.01;
alphas = [2 - epsilon, 2 - 2 * epsilon]; %the two alpha points where
 the secant will start
MP = []; %Matrice where the 3 diffrent pair of solutions needed for
 the secant will be saved.

dom1 = [0, pi / 2]; %domain of phi1
dom2 = [-pi / 2, pi / 2]; %domain of phi2



%As seen in the plot of the previous section for the two alpha chosen
 to start de secant we'll
```

```
%have the 0 solution (trivial), two solution bigger than 0.6 and two
 more
%below 0.6, we use that information to obtain them:

for jj = 1:2
    sol1 = 0; % Si es 0 es que falta trobarla:
    sol2 = 0;
    alpha = 2 - jj*epsilon;
    f = @(phi)([(tan(phi(1)) - alpha * (2 * sin(phi(1)) +
 sin(phi(2)))), (tan(phi(2)) - 2 * alpha * (sin(phi(1)) +
 sin(phi(2))))]);

    while sol1 == 0 || sol2 == 0
        aleatory = rand(2, 1);
        phi0 = aleatory .* factor - a;
        %x*(a-b) - a

        [XK, resd, it] = newtonn(phi0, 1e-16, 100, f);
        % Comprobar que estigui dintre el domini
        if XK(1, end) > dom1(1) && XK(1, end) < dom1(2) && XK(2, end)
 > dom2(1) && XK(2, end) < dom2(2)
            % I a mes que no sigui 0:
            if XK(1, end) > epsilon || XK(1, end) < (-0.001)% El blau
 sempre esta per sobre i nomes cal que comprobem aquest
                %Classificar si es de dalt o de sota
                if XK(1, end) > 0.6
                    MP(:, jj) = [XK(:, end); alpha];
                    sol1 = 1; % He trobat la solucio 1
                else
                    MP(:,jj+2) = [XK(:, end); alpha];
                    sol2 = 1;
                end
            end
        end
    end
end


% We add the (0, 0) trivial solutions
MP(:,5) = [0; 0; 2-epsilon];
MP(:,6) = [0; 0; 2-2*epsilon];


disp('Initial values for the secant')
disp(MP)

s = 1; % In principle we will use s = 1 to mantain an approximate
 regular space between solutions.
% The distrance between solution will be given by the epsilon
 parameter
% defined before.

tol = 1e-6;
itmax = 100;
```

```
Y = [];

for it = 1:2:length(MP)% We launch the countinuation step at the 3
 branches of solutions found
    y0 = MP(:, it);
    y1 = MP(:, it + 1);
    y = y1;

    while y(3) < 2 && y(3) > 0 && s > 0
        [y, iconv] = continuationStep(funAlpha, y0, y1, s, tol,
 itmax);

       if iconv == 1 % No hem aconseguit solució i ajustem s
            s = s - 0.1; % Si la s arriba a 0 desistirem i no buscarem
 mes solucions
        else
            y0 = y1;
            y1 = y;
            % Nomes les guardem si estan dintre el domini
            if y(1, end) >= dom1(1) && y(1, end) <= dom1(2) && y(2,
 end) >= dom2(1) && y(2, end) <= dom2(2)
                Y = [Y, y]; %solucions
            end
        end
        plot(Y(end, :), Y(1:2, :), 'o');
        hold on
    end
end

title('Plot of the solution as a function of alpha using secant
 continuation step');
xlabel('Alpha')
ylabel('Phi')
hold off

Initial values for the secant
    1.4028    1.4020    0.4171    0.4114         0         0
    1.4444    1.4438   -0.6281   -0.6181         0         0
    1.9900    1.9800    1.9900    1.9800    1.9900    1.9800
```



Section C: Optional part

Drawing of the pendulum at alfa = 2.14

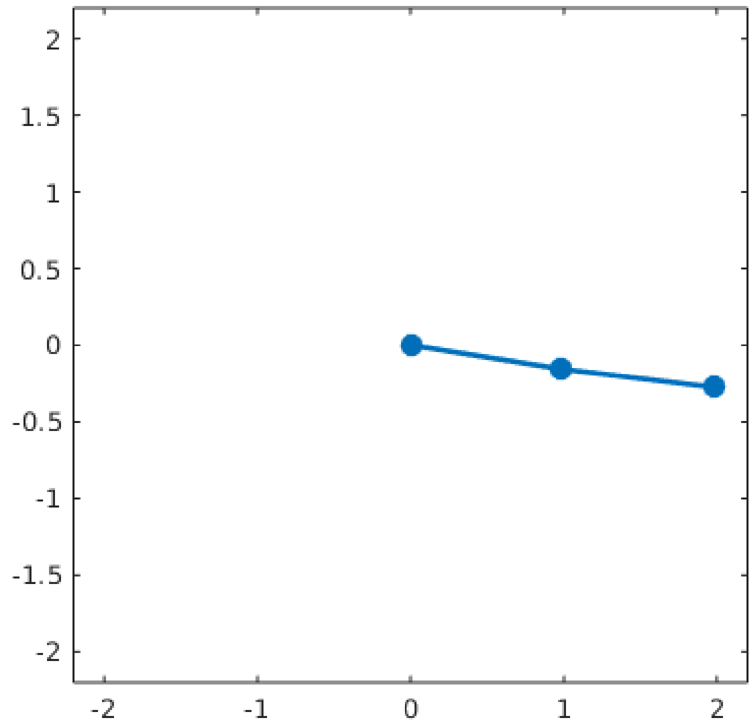
```
%As we want the plot at a specific alpha we'll use the newton method
 in
%order to explore and not the secant continuation step to find
 solutions.

figure(5)
aleatoryTimes = 1:1:1000;
alpha = 2.14;
f = @(phi)([(tan(phi(1)) - alpha * (2 * sin(phi(1)) + sin(phi(2)))),
 (tan(phi(2)) - 2 * alpha * (sin(phi(1)) + sin(phi(2))))]);
for i = aleatoryTimes
    aleatory = rand(2, 1);
    phi0 = aleatory .* factor - a;
    %x*(a-b) - a
    [XK, resd, it] = newtonn(phi0, 1e-16, 100, f);
    if XK(1, end) >= dom1(1) && XK(1, end) <= dom1(2) && XK(2, end) >=
 dom2(1) && XK(2, end) <= dom2(2)
            plot(alpha, XK(1, end), '*');  %phi1
            plot(alpha, XK(2,end), '*') %phi2
            hold on
    end
end
```

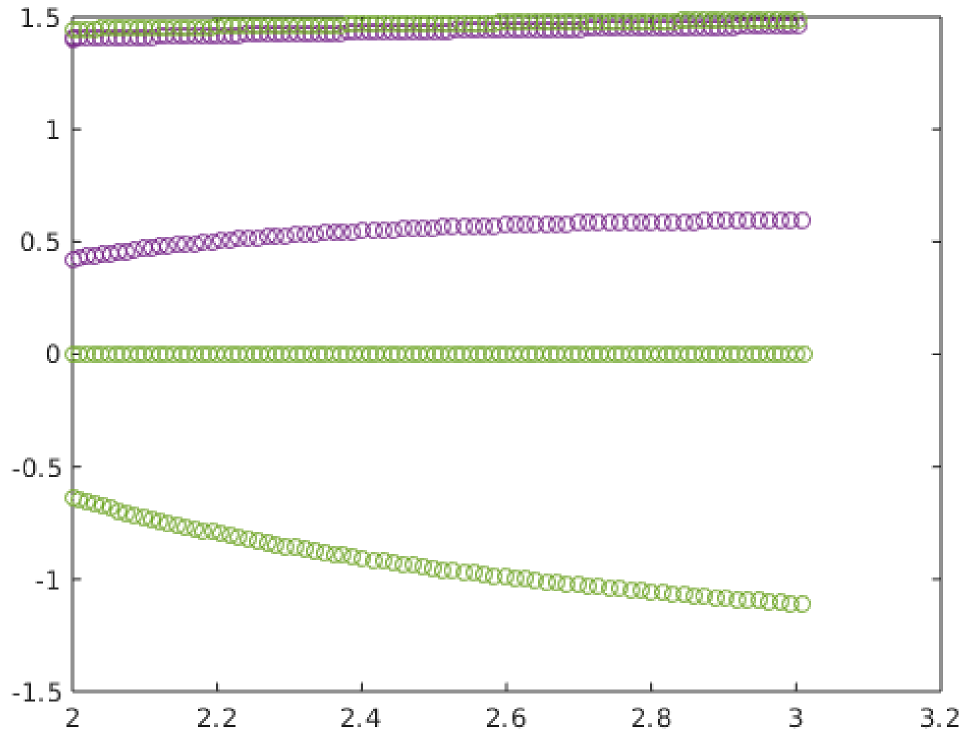
```
title('Exploration at alpha = 2.14 using newton with aleatory initial
 points');
xlabel('Alpha')
ylabel('Phi')
hold off


% We change the direction at we launch the secant continuation step,
 now we
% go to the right
% We will get to alpha = 3;
maxAlfa = 3;
Y = [];
perDibuixar = [];

figure;
title('Exploration from alpha = 2.14 to alpha = 3 using secant
 continuation step');
xlabel('Alpha')
ylabel('Phi')
% Tirarem el continuation step en les 3 branques de solucions trobades
 anteriorment:
for it = 1:2:length(MP)
    y1 = MP(:, it); % Per ferho cal canviar el sentit, ja que ara
 anirem cap a la dreta
    y0 = MP(:, it + 1);
    y = y0;

    sensePintar = 1;

    while y(3) < maxAlfa && y(3) > 0 && s > 0
        [y, iconv] = continuationStep(funAlpha, y0, y1, s, tol,
 itmax);
        if iconv == 1 % No hem aconseguit solució i ajustem s
            s = s - 0.1; % Si la s arriba a 0 desistirem i no buscarem
 mes solucions
        else
            y0 = y1;
            y1 = y;
            Y = [Y, y]; %solucions
            % When we found the pendulum we save the solutions to
 draw.
            if abs(alpha-y(3)) < epsilon && sensePintar == 1
                perDibuixar = [perDibuixar, y];
                sensePintar = 0;
            end
        end

        plot(Y(end, :), Y(1:2, :), 'o');
        hold on
    end

end
```

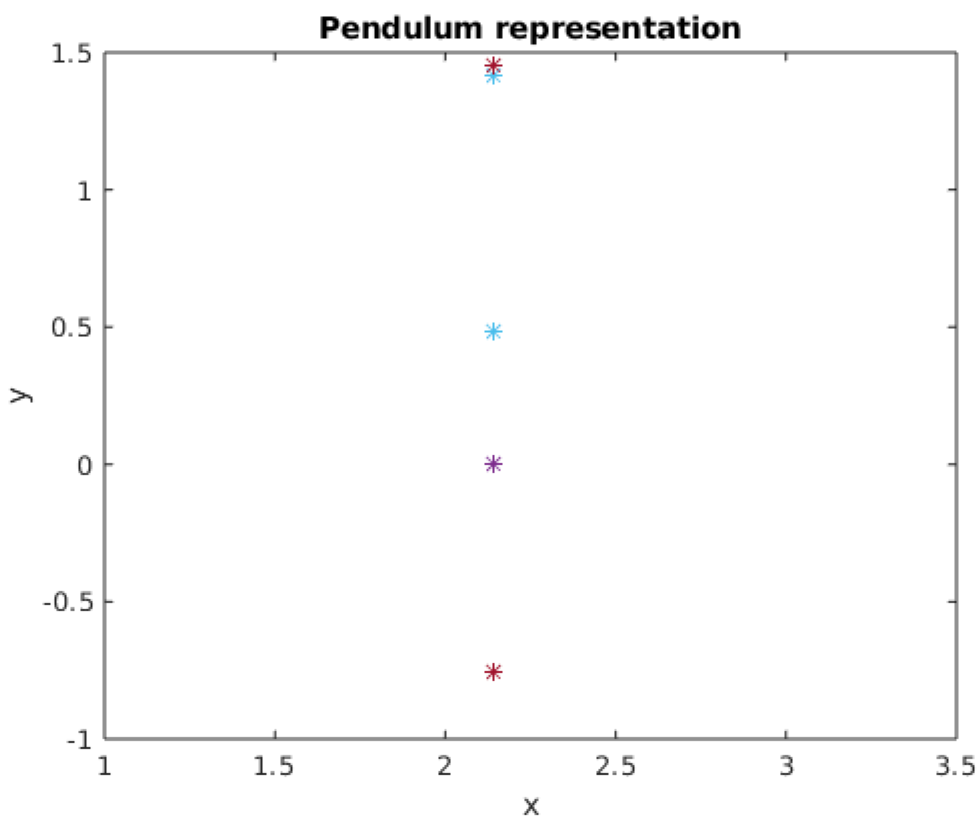
```
hold off

% Dibuixem els pendols:
for ii =perDibuixar
    figure;
    dibuixarPendul(ii(1), ii(2), 1);
    title('Pendulum representation')
    xlabel('x')
    ylabel('y')
    hold off
end

% Codes used:
%{
function succes = dibuixarPendul(phi1, phi2, l)
% Funcio que fa un plot de un pendul doble amb els angles indicats i
 la longitud de les barres indicada (les dos igual)
% a entrar els angels respecte la vertical en radiants

h = plot(0, 0, 'MarkerSize', 30, 'Marker', '.', 'LineWidth', 2);
 %Guardem el objecte plot en una variable per utilitzar les seves
 propietats mes endavant

range = 1.1 * (l + l); axis([-range range -range range]); axis square;

set(gca, 'nextplot', 'replacechildren'); % Diem que en el seguent plot
 es pinti a partir d'on acaba l'anterior:

Xcoord = [0, l * sin(phi1), l * sin(phi1) + l * sin(phi2)];
Ycoord = [0, -l * cos(phi1), -l * cos(phi1) - l * cos(phi2)];
set(h, 'XData', Xcoord, 'YData', Ycoord);
drawnow;

succes = 1;


function [y, iconv] = continuationStep(fun, y0, y1, s, tol, itmax)

    it = 1;
    tolk = 1;
    v = y1 - y0;
    yp = y1 + v * s; % Si s = 1 conseguim que la separaci# entre
 solucions sigui el maxim de "constant"
    xk = yp;
    XK = [];

    % A part de les ecuacions que teniem en el nnewton normal, li
    % imposareem que el preoducte escalar entre v i (xk(punt
    % buscat)- xk(predictor)) sigui 0
    while it < itmax && tolk > tol
        J = jaco(fun, xk); % Jacobia en la posicio anterior

        J = [J; v'];
```

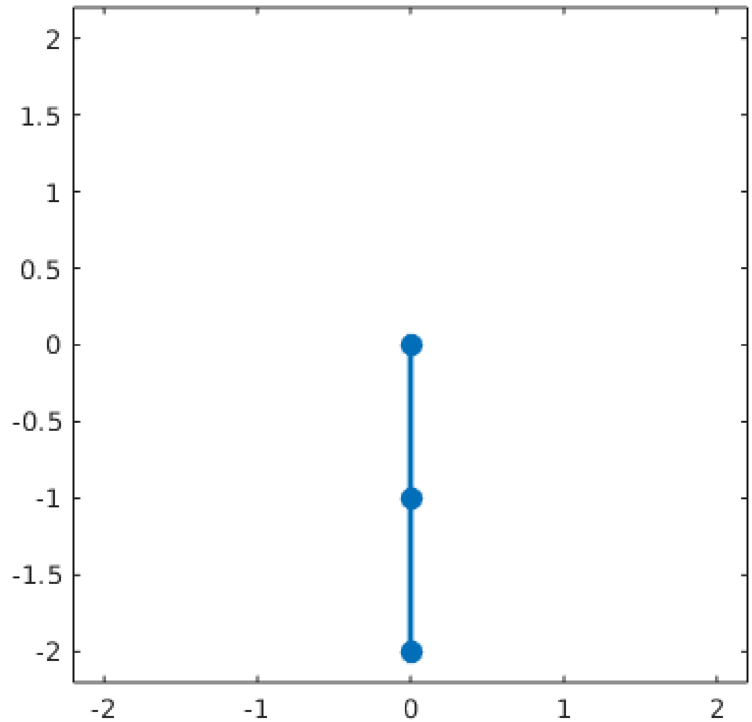
```
        fk = [fun(xk); v' * (xk - yp)]; % TODO: Copiat de teoria
        [P, L, U] = PLU(J);
        Dx = pluSolve(L, U, P, -fk); %Solucio de la ecuacio J*Dx = -fk
        %Dx = J\fk;
        xk = xk + Dx;
        XK = [XK, xk];
        tolk = norm(Dx); % Mirem la distancia entre el anterior i
 l'actual
        it = it + 1;
    end

    y = xk;

    %Retornem si convergeix o no per modificar la s si cal:
    if it <= itmax && tolk < tol
        iconv = 0; %OK
    else
        iconv = 1; %No em arribat a enlloc
    end

end


function [XK, resd, it] = newtonn(x0, tol, itmax, fun)
    % Atencio, pirmer comprobara a a la carpeta actual si hi son

    xk = [x0];
    XK = [x0];
    resd = [norm(feval(fun, xk))];
    it = 1;
    tolk = 1;

    while it < itmax && tolk > tol
        J = jaco(fun, xk); % Jacobia en la posicio anterior
        fk = feval(fun, xk);
        %[P, L, U] = PLU(J);

        %Dx = pluSolve(L, U, P, (-fk)'); %Solucio de la ecuacio J*Dx =
 -fk

        Dx = J\(-fk)';
        xk = xk + Dx;
        XK = [XK, xk];
        resd = [resd, norm(fk)];
        tolk = norm(XK(:, end) - XK(:, end - 1));
        it = it + 1;


    end

%}
```

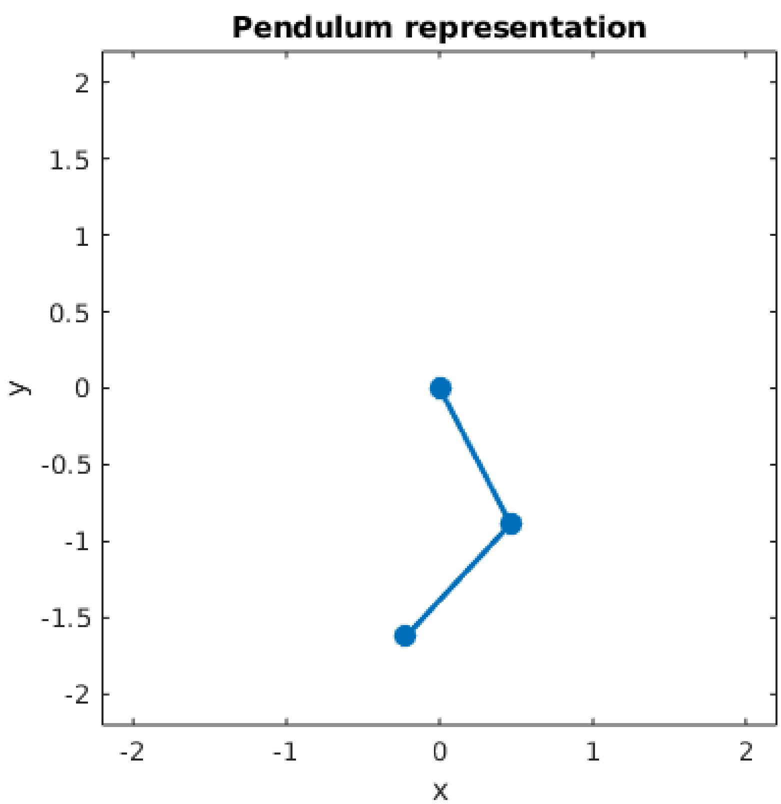
```
%As it can be seen, the solution to the right continue as
 theoretically
%expected. We do not observe not new branches emerged using
 continuationStep and
%neither using newton. This is because new branches appear when the
%jacobian's determinan is zero, and from 2,14 on it is always non
 zero, as
%it has been proved in the graph in section A.
```




1.5
1
0.5
0
-0.5
-1
-1.5
2
2.2
2.4
2.6
2.8
3
3.2


Pendulum representation
1.5
1
0.5
0
-0.5
-1
y
1
1.5
2
2.5
3
3.5
x


2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1
0
1
2


Pendulum representation
y
x
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2
-1
0
1
2

Published with MATLAB® R2019b