Table of Contents

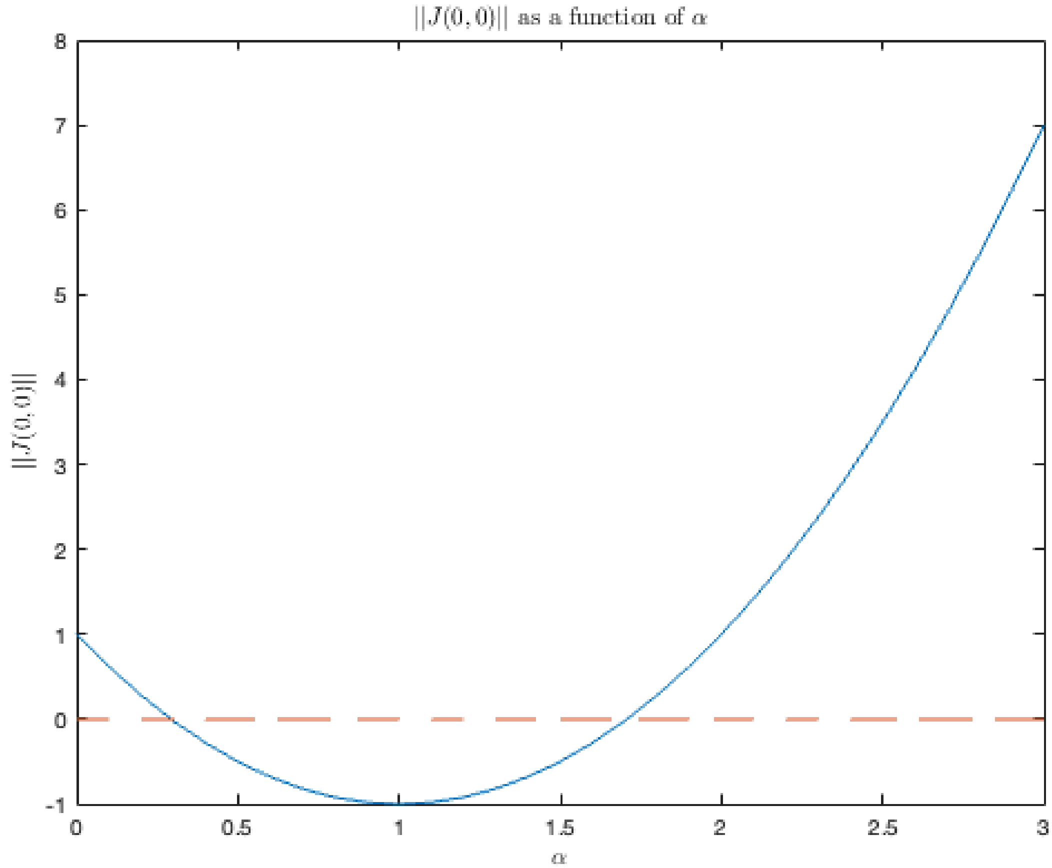
Practical 6 - Cortés & García 1
(a) Branches of equilibrium 1
(b) Newton 2
(c) Continuation 4
Auxiliar Codes 9

Practical 6 - Cortés & García

```
clear all
close all
format long

% Definition of the function we want to optimize
% phi = [phi1; phi2] -> column vector with the two angles phi1 and phi2
% alpha = l*omega^2/(2g) -> as said in the statement
f = @(phi, alpha) [tan(phi(1)) - alpha*(2*sin(phi(1)) + sin(phi(2))) ; ...
    tan(phi(2)) - 2*alpha*(sin(phi(1)) + sin(phi(2)))];
```

(a) Branches of equilibrium

```
alVec0 = 0:0.1:3;
detJVec = alVec0*0;

for i = 1:length(alVec0)
    f_alpha = @(x) f(x, alVec0(i));

    detJVec(i) = det(jac(f_alpha, [0; 0]));
end

figure(1)
plot(alVec0, detJVec, "LineWidth", 0.8)
hold on
plot(alVec0, zeros(size(alVec0)), "--")
xlabel("$\alpha$", "Interpreter","latex")
ylabel("$||J(0,0)||$", "Interpreter","latex")
title("$||J(0,0)||$ as a function of $\alpha$", "Interpreter","latex")
hold off
```



For α other than the critical values where the determinant of the Jacobian matrix is 0, we will expect the system to have a unique solution. For those critical values of alpha that cancel the determinant, we may expect new branch solutions to emerge. Graphically speaking, this values of α are approximately $\alpha_1 = 0.3$ and $\alpha_2 = 1.7$ (the values where the determinant of the Jacobian crosses the horizontal line at $y = 0$.

Anallitically, we have that the jacobian of this function at (0, 0) is

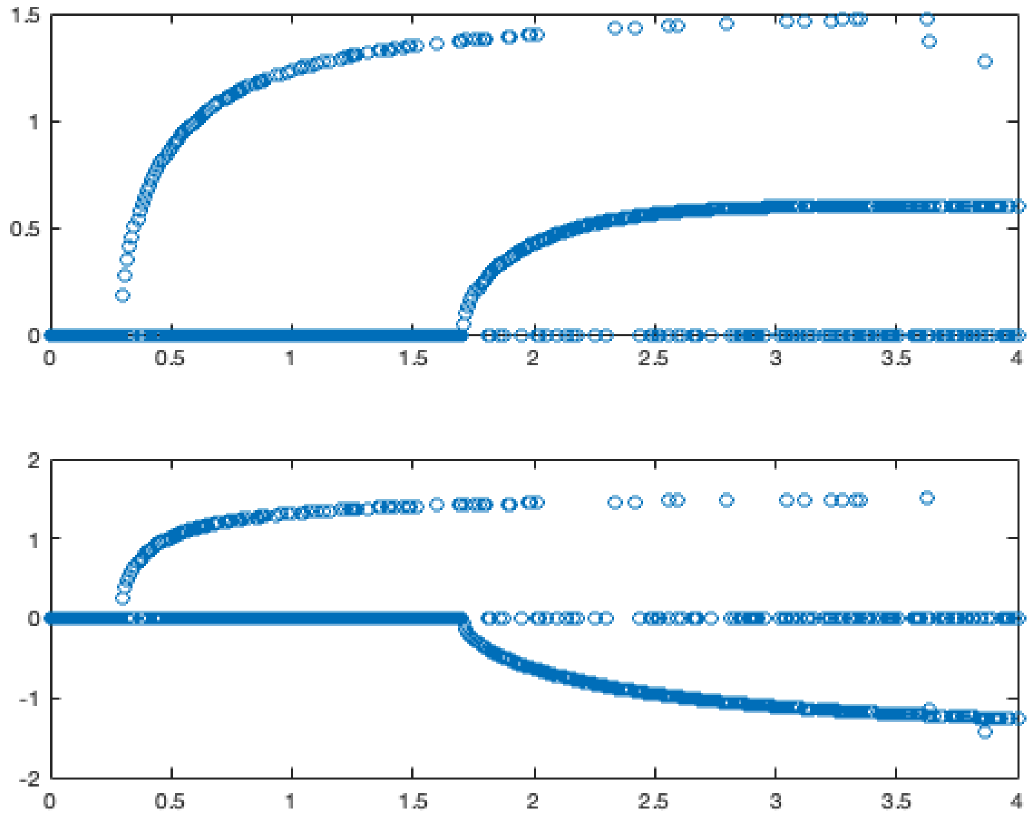
$$J_\alpha(0,0) = \begin{pmatrix} 1-2\alpha & -\alpha \\ -2\alpha & 1-2\alpha \end{pmatrix}$$

We can clearly see that the determinant of this matrix is $|J_\alpha(0,0)| = (1-2\alpha)^2 - 2\alpha = 2\alpha^2 - 4\alpha + 1$. The cases in which this determinant will be 0 are the solutions $\alpha = \frac{2\pm\sqrt{2}}{2}$. We then have that $\alpha_1 = \frac{2-\sqrt{2}}{2} = 0.2928...$ and $\alpha_2 = \frac{2+\sqrt{2}}{2} = 1.7071...$ (approximately the values we observed graphically).

(b) Newton

```
alVec = [];
XkVec = [];

for al = 0:0.01:4
    for it = 1:10
```

```
        x0 = [(pi/2)*rand(); pi*rand()-pi/2];

        f_alpha = @(x) f(x, al);

        [Xk, ~, ~] = newtonn(x0, eps, 50, f_alpha);
        xk = Xk(:, end);

        if xk(1) >= 0 && xk(1) < pi/2 && xk(2) > -pi/2 && xk(2) < pi/2
            XkVec = [XkVec , xk];
            alVec = [alVec al];
        end
    end
end

figure(2)
subplot(2, 1, 1)
plot(alVec, XkVec(1,:), 'o')
subplot(2, 1, 2)
plot(alVec, XkVec(2,:), 'o')
```



To reach the goal of observing what happens through various values of α for various randomly chosen initial conditions, we are setting a random initial guess in the domains of both ϕ_1 and ϕ_2 10 times and we then apply Newton's Method to try and find an equilibrium point.

Graphically, we observe that when $\alpha < \alpha_1$, there is only one solution: the equilibrium position is the trivial solution $\phi_1 = \phi_2 = 0$. However, when $\alpha_1 \leq \alpha < \alpha_2$, the equilibrium position will depend on the initial guess and there is another possibility aside from the trivial solution. When $\alpha_2 \leq \alpha$ there appears another branch and there are actually 3 possible solutions (the trivial one and 2 more). Therefore, we can see graphical proof that the 2 branches we predicted appear.

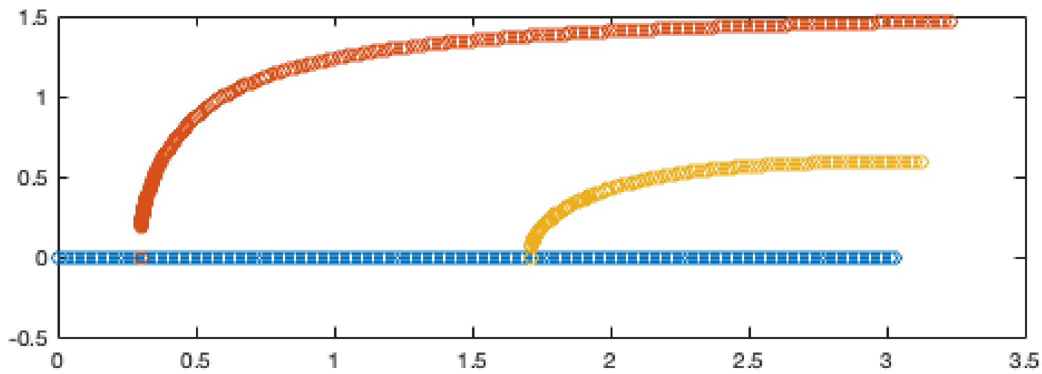
(c) Continuation

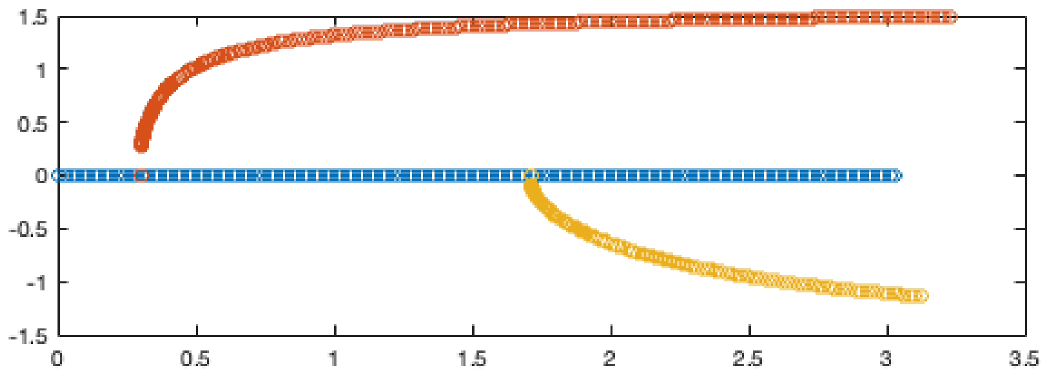
Now we want to plot both $\phi_1(\alpha)$ and $\phi_2(\alpha)$ for the 3 solution branches found. When it comes to the trivial solution, it will be easy: it is only needed to take 2 initial close solutions ($\phi_1(0) = \phi_2(0) = \phi_1(0.01) = \phi_2(0.01) = 0$) and see how the equilibrium positions evolve with α.

The same will need to be done for the other 2 branches, but starting at α_1 and α_2, where each branch starts respectively. To do this, we'll benefit from the previous results: we'll want to find the values α closest to the values α_1 and α_2 we obtained analitically. In the previous loop, we used a step between αs of 0.01, so that is the maximum difference we'll want to look at. Then, we'll take the solutions for ϕ_1 and ϕ_2 for those values of α, and start the continuation step from there. We'll iterate the continuation enough to obtain a clear picture of how it evolves with α.

```
f_cont = @(x) f([x(1) ; x(2)], x(3));

% Zero solution
x00 = [0; 0; 0];
x10 = [0; 0; 0.01];

alVecCont0 = zeros(1, 152);
alVecCont0(1) = x00(3); alVecCont0(2) = x10(3);
XkVecCont0 = zeros(2, 152);
XkVecCont0(:, 1) = [x00(1) ; x00(2)]; XkVecCont0(:, 2) = [x10(1) ; x10(2)];

for it = 0:150
    s = .02/norm(x10-x00);
    [y, iconv] = contstep(f_cont, x00, x10, s, eps, 50);

    x00 = x10;
    x10 = y;

    alVecCont0(it+2) = y(3);
    XkVecCont0(:, it+2) = [y(1) ; y(2)];
end

% Solution with branch starting at alpha1
idx1 = find(abs(alVec - (2-sqrt(2))/2) < 1e-2);
x01 = [0; 0; alVec(idx1(end-1))];
x11 = [XkVec(:, idx1(end)) ; alVec(idx1(end))];

alVecCont1 = zeros(1, 202);
alVecCont1(1) = x01(3); alVecCont1(2) = x11(3);
XkVecCont1 = zeros(2, 202);
XkVecCont1(:, 1) = [x01(1) ; x01(2)]; XkVecCont1(:, 2) = [x11(1) ; x11(2)];

for it = 0:200
    s = .02/norm(x11-x01);
```

```
    [y, iconv] = contstep(f_cont, x01, x11, s, eps, 50);

    x01 = x11;
    x11 = y;

    alVecCont1(it+2) = y(3);
    XkVecCont1(:, it+2) = [y(1) ; y(2)];
end

% Solution with branch starting at alpha2
idx2 = find(abs(alVec - (2+sqrt(2))/2) < 1e-2);
x02 = [0; 0; alVec(idx2(end-1))];
x12 = [XkVec(:, idx2(end)) ; alVec(idx2(end))];

alVecCont2 = zeros(1, 102);
alVecCont2(1) = x02(3); alVecCont2(2) = x12(3);
XkVecCont2 = zeros(2, 102);
XkVecCont2(:, 1) = [x02(1) ; x02(2)]; XkVecCont2(:, 2) = [x12(1) ; x12(2)];

for it = 0:100
    s = .02/norm(x12-x02);
    [y, iconv] = contstep(f_cont, x02, x12, s, eps, 50);

    x02 = x12;
    x12 = y;

    alVecCont2(it+2) = y(3);
    XkVecCont2(:, it+2) = [y(1) ; y(2)];
end

figure(3)
subplot(2, 1, 1) %phi1
plot(alVecCont0, XkVecCont0(1,:), 'o')
hold on
plot(alVecCont1, XkVecCont1(1,:), 'o')
plot(alVecCont2, XkVecCont2(1,:), 'o')
ylim([-.5 1.5])
hold off
subplot(2, 1, 2) % phi2
plot(alVecCont0, XkVecCont0(2,:), 'o')
hold on
plot(alVecCont1, XkVecCont1(2,:), 'o')
plot(alVecCont2, XkVecCont2(2,:), 'o')
hold off
```

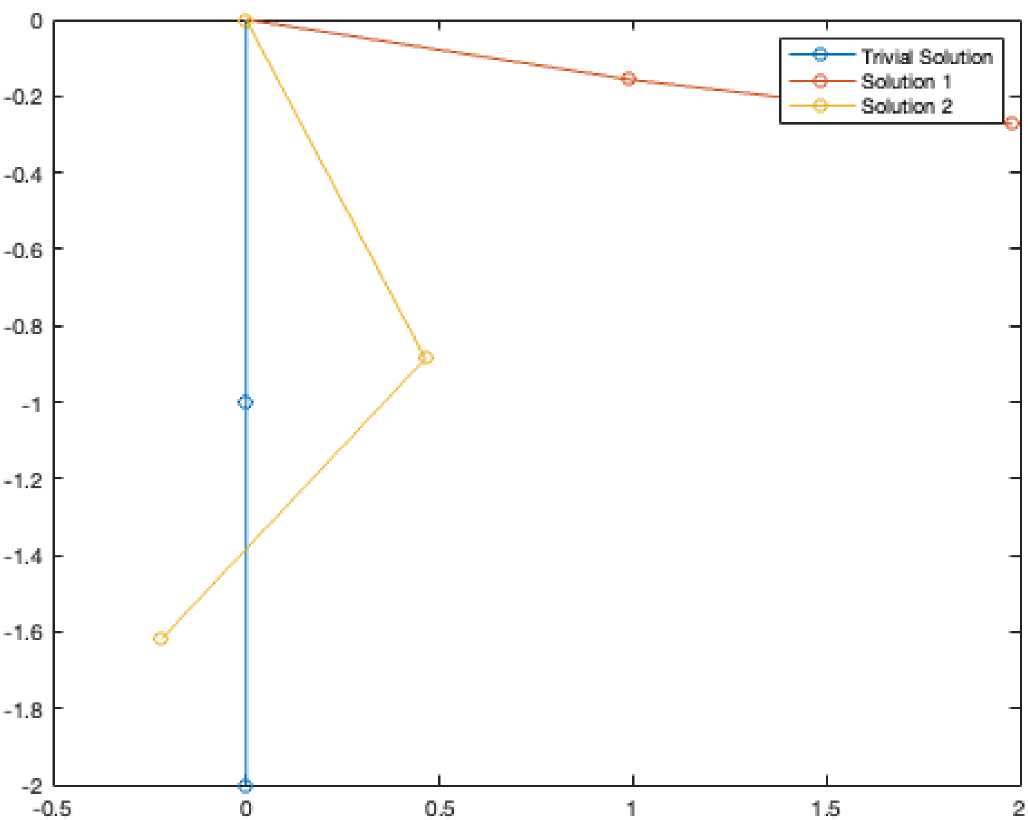




We can now see clearly how ϕ_1 and ϕ_2 evolve as a function of α. We can see that these new solutions start at α_1 and α_2 and they correspond to the ones we had anticipated. We can also see that for a large enough α (from $\alpha = 3$ approximately) both angles seem to converge to a value each, which would physically mean that when ω is large enough (the pendulum is turning fast enough) the equilibrium position will remain the same when increasing the angular velocity depending on the initial conditions of the problem.

```
idxx0 = find(abs(alVecCont0 - 2.14) < 1e-2);
idxx1 = find(abs(alVecCont1 - 2.14) < 1e-2);
idxx2 = find(abs(alVecCont2 - 2.14) < 1e-2);

phi10 = XkVecCont0(1, idxx0(1));
phi20 = XkVecCont0(2, idxx0(1));
phi11 = XkVecCont1(1, idxx1(1));
phi21 = XkVecCont1(2, idxx1(1));
phi12 = XkVecCont2(1, idxx2(1));
phi22 = XkVecCont2(2, idxx2(1));


l = 1; % Assume length of the pendulums of l=1
Sol0 = [[0; 0] , ...
    [l*sin(phi10); -l*cos(phi10)] , ...
    [l*sin(phi10)+l*sin(phi20); -l*cos(phi10)-l*cos(phi20)]];
Sol1 = [[0; 0] , ...
```

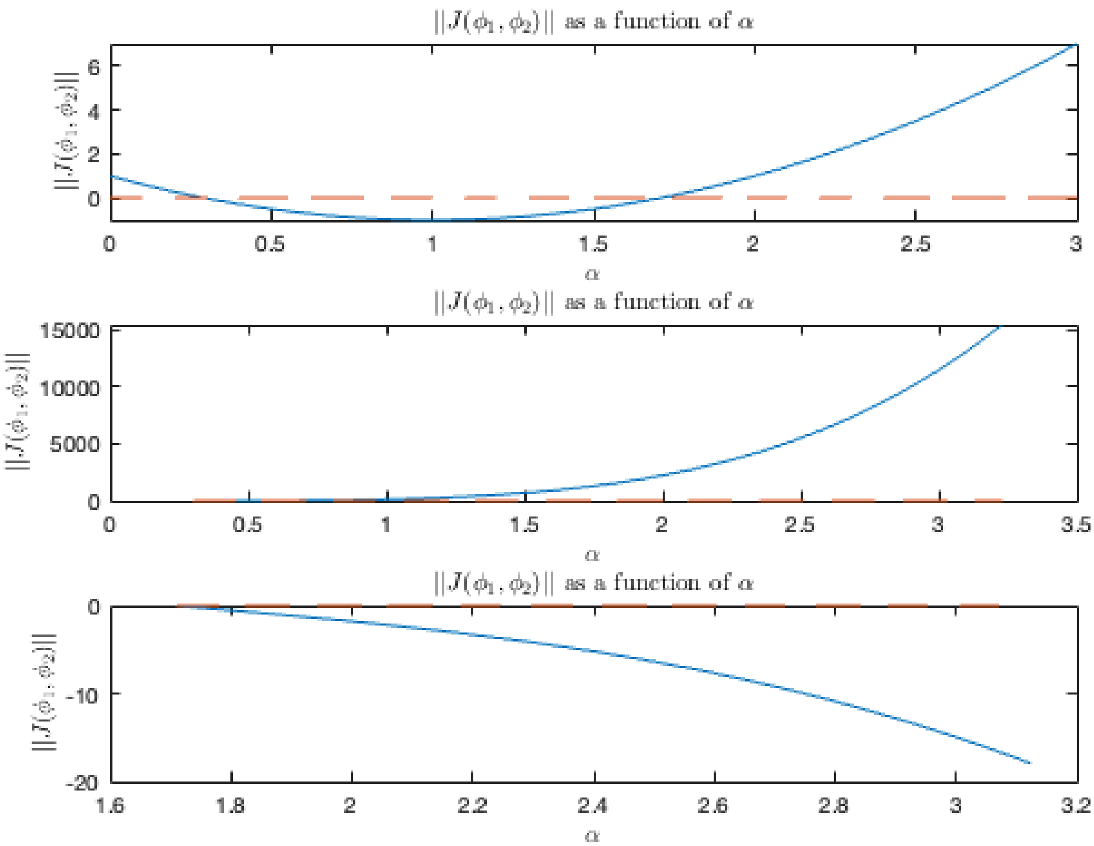
```
    [l*sin(phi11); -l*cos(phi11)] , ...
    [l*sin(phi11)+l*sin(phi21); -l*cos(phi11)-l*cos(phi21)]];
Sol2 = [[0; 0] , ...
    [l*sin(phi12); -l*cos(phi12)] , ...
    [l*sin(phi12)+l*sin(phi22); -l*cos(phi12)-l*cos(phi22)]];

% Jo posaria una linea horitzontal al 0 i que es vegi algo més que només
% els pèndols (en plan centrar-ho verticalment en el 0 i afegir algo de
% marge pels costats i per adalt i per abaix).
figure(5)
plot(Sol0(1,:), Sol0(2,:), '-o')
hold on
plot(Sol1(1,:), Sol1(2,:), '-o')
plot(Sol2(1,:), Sol2(2,:), '-o')
hold off
legend('Trivial Solution', 'Solution 1', 'Solution 2')
```



For $\alpha = 2.14$, this solutions of the actual pendulum are represented in the previous figure.

```
detJVec1 = alVecCont1*0;
for i = 1:length(alVecCont1)
    f_alpha = @(x) f(x, alVecCont1(i));
    detJVec1(i) = det(jac(f_alpha, XkVecCont1(1:2,i)));
end
```

```
detJVec2 = alVecCont2*0;

for i = 1:length(alVecCont2)
    f_alpha = @(x) f(x, alVecCont2(i));
    detJVec2(i) = det(jac(f_alpha, XkVecCont2(1:2,i)));
end

figure(6)
subplot(3, 1, 1)
plot(alVec0, detJVec, "LineWidth", 0.8)
hold on
plot(alVec0, zeros(size(alVec0)), "--")
hold off
xlabel("$\alpha$", "Interpreter","latex")
ylabel("$||J(\phi_1,\phi_2)||$", "Interpreter","latex")
title("$||J(\phi_1, \phi_2)||$ as a function of $\alpha
$", "Interpreter","latex")
subplot(3, 1, 2)
plot(alVecCont1, detJVec1, "LineWidth", 0.8)
hold on
plot(alVecCont1, zeros(size(alVecCont1)), "--")
hold off
xlabel("$\alpha$", "Interpreter","latex")
ylabel("$||J(\phi_1,\phi_2)||$", "Interpreter","latex")
title("$||J(\phi_1, \phi_2)||$ as a function of $\alpha
$", "Interpreter","latex")
subplot(3, 1, 3)
plot(alVecCont2, detJVec2, "LineWidth", 0.8)
hold on
plot(alVecCont2, zeros(size(alVecCont2)), "--")
hold off
xlabel("$\alpha$", "Interpreter","latex")
ylabel("$||J(\phi_1,\phi_2)||$", "Interpreter","latex")
title("$||J(\phi_1, \phi_2)||$ as a function of $\alpha
$", "Interpreter","latex")
hold off
```



When we evaluate the determinant of the Jacobian matrix for each branch, it looks like it won't be 0 again for any of them, which leads us to think that there will not be more new branches. More specifically, for the branch that starts at α_1, the determinant of the jacobian is incrases with α and for the one that starts at α_2, it decreases with α.

Auxiliar Codes

```
% Code 20: Newton’s method for n-dimensional systems
% Input: x0 - initial guess (column vector)
%        tol - tolerance so that ||x_{k+1} - x_{k} || < tol
%        itmax - max number of iterations
%        fun - function’s name
% Output:   XK - iterated
%           resd: resulting residuals of iteration: ||F_k||
%           it:   number of required iterations to satisfy tolerance
function [XK,resd,it] = newtonn(x0,tol,itmax,fun)
    xk = [x0];
    resd = [norm(feval(fun,xk))];
    XK = [x0];
    it = 1;

    tolk = 1.0;
    n = length(x0);
```

```
    while it < itmax && tolk > tol
        Fk = feval(fun, xk);

        DFk = jac(fun, xk);
        [P,L,U] = pplu(DFk);

        dxk = plusolve(L,U,P,-Fk);

        xk = xk + dxk;
        XK = [XK xk];
        resd = [resd norm(Fk)];
        tolk = norm(XK(:, end)-XK(:, end-1));
        it = it + 1;
    end
end

% Code 21: secant continuation step
% Input:    y0 and y1 (two close column vectors)
%           s: pseudo-arclength parameter
%           tol - Newton's tolerance: ||y_[k+1] - y_[k] || < tol % itmax - max
 number of iterations
%           fun - function's name: f(y_1,y_2,...,y_n,y_n+1)
% Output:   y - next point along curve f = 0
%           y belongs to plane orth. to y1-y0
%           passing through secant predictor y1 + s(y1-y0)
%           iconv (0 if y is convergenced to desired tol.)
function [y,iconv] = contstep(fun,y0,y1,s,tol,itmax)
    tolk = 1.0;
    it = 0;
    n = length(y0)-1;

    v = y1-y0;
    yp = y1+s*v;
    xk = yp;

    while tolk > tol && it < itmax
        Fk = [feval(fun,xk); v'*(xk-yp)];
        DFk = [jac(fun,xk); v'];

        [P,L,U] = pplu(DFk);
        dxk = plusolve(L,U,P,-Fk);

        xk = xk + dxk;
        tolk = norm(dxk);
        it = it + 1;
    end

    y = xk;
    if it <= itmax && tolk < tol
        iconv = 0;
    else
        iconv = 1;
    end
end
```

```
% Code 19: Computation of the Jacobian J
% Input:   F(x) : R^m ---> R^n
%          x : (m x 1)-vector ; F: (n x 1)-vector
% Output: DF(x) (n x m) Jacobian matrix at x
function DF = jac(F,x)
    f1 = feval(F,x);
    n = length(f1);
    m = length(x);

    DF = zeros(n,m);
    H = sqrt(eps)*eye(m);

    for j = 1:m
        f2 = feval(F,x+H(:,j));
        DF(:,j) = (f2 - f1)/H(j,j);
    end
end

% Code 13: PA = LU factorization (partial pivoting)
% Input: A (non-singular square matrix)
% Output: L (unit lower triangular matrix)
%         U (upper triangular matrix)
%         P (reordering vector)
function [P, L, U] = pplu(A)
    [m,n] = size(A);

    if m~=n
          error('not square matrix');
    end

    U = A;
    L = eye(n);

    P = [1:n]';

    for k = 1:n-1
        [~, imax] = max(abs(U(k:end,k)));
        imax = imax+k-1;
        i1 = [k, imax];
        i2 = [imax, k];

        U(i1,:) = U(i2,:); % Column k will be column imax and column imax will
 be column k
        P(k) = imax;

        L(i1,1:k-1) = L(i2, 1:k-1);

        for jj = [k+1:n]
            L(jj, k) = U(jj, k)/U(k, k);
            U(jj, k:n) = U(jj, k:n) - L(jj, k)*U(k,k:n);
        end
    end
end
```

```
% Code 14: PA = LU (Solver for Ax = b)
% Input:    L (unit lower triangular matrix)
%           U (upper triangular matrix)
%           P (reordering vector)
%           b (right-hand side)
% Output:   solution x
function x = plusolve(L, U, P, b)
    n = length(b);
    for k = 1:n-1
        b([k P(k)]) = b([P(k) k]);
    end
    y = fs(L, b);
    x = bs(U, y);
end

% Code 11: Forward Substitution for Lower Triangular Systems
% Input:    L: Low Triangular non-singular square matrix
%           b: column right-hand side
% Output:   x: solution of Lx=b
function x = fs(L, b)
    x = 0*b;
    n = length(b);
    x(1) = b(1)/L(1,1);

    for ii = 2:n
        x(ii) = (b(ii)-L(ii, 1:ii-1)*x(1:ii-1))/L(ii,ii);
    end
end

% Code 12: Backward Substitution for Upper Triangular Systems
% Input:    U: Upp. Triangular non-singular square matrix
%           b: column right-hand side
% Output:   x: solution of Ux=b
function x = bs(U, b)
    x = 0*b;
    n = length(b);
    x(n) = b(n)/U(n,n);

    for ii = n-1:-1:1
        x(ii) = (b(ii)-U(ii, ii+1:n)*x(ii+1:n))/U(ii,ii);
    end
end
```

Published with MATLAB® R2023a