ANEXOS

ANEXO I: Cálculo de los momentos de inercia.
ANEXO II: Método Newton-Raphson.
ANEXO III: Código de Matlab.
ANEXO IV: Sistema de ecuaciones dinámica "directa".

ANEXO I:

Cálculo de los momentos de inercia.

A través del Teorema de Steiner, enunciado a continuación, utilizando el momento de inercia de un sólido respecto de su centro de masa y conociendo la posición del centro de masa, es posible calcular cual será el momento de inercia de cada uno de los sólidos respecto del extremo sobre el que giran.

Teorema de Steiner

$$I_{P,eje} = I_{CM,eje} + Md^2$$

Sólido 1:

$I_{Oz} = 101211.52 + 88.21 * 52.5^2 \simeq 344340.3325\ g * mm^2 = 0{,}0003443403325\ Kg * m^2$

Sólido 2:

$I_{Az} = 783941.91 + 177.73 * 110^2 \simeq 2934475\ g * mm^2 = 0{,}002934475\ Kg * m^2$

Sólido 3:

$I_{Fz} = 14101969.44 + 437.58 * 289.49^2 \simeq 50773125\ g * mm^2 = 0{,}050773125\ Kg * m^2$

Sólido 4:

$I_{Cz} = 14041914.46 + 434.52 * 290.47^2 \simeq 50703593\ g * mm^2 = 0{,}050703593\ Kg * m^2$

Sólido 5:

$I_{Dz} = 1916773.41 + 240.13 * 150^2 \simeq\ 7319698 g * mm^2 = 0{,}007319698\ Kg * m^2$

ANEXO II:

El método de Newton-Raphson.

- EL MÉTODO DE NEWTON-RAPHSON -

1 - Introducción -

Hallar los ceros de una función f, es decir, los valores del argumento x para los que $f(x) = 0$, es un problema muy clásico. Existen fórmulas explícitas para la resolución de ecuaciones polinómicas hasta grado cuatro. Para polinomios de grado superior, la regla de Ruffini constituye una herramienta útil cuando el polinomio tiene raíces enteras y únicamente determina éstas.Algunos tipos particulares de ecuaciones logarítmicas, exponenciales o trigonométricas, pueden resolverse de forma exacta.

Sin embargo, en ocasiones se está interesado en la obtención de las soluciones de una ecuación, para las que no se dispone de un método para su obtención exacta. Es el caso de ecuaciones tan sencilla como las siguientes

$$\sin x + x = 1, \quad x^5 + x = 1, \quad x \sin(1/x) = 0.2e^{(-x)}, \quad e^x = 6x.$$

Si se trata de encontrar un cero de una función continua en un intervalo cerrado $[a, b]$ en el que sabemos que dicha función se anula, existen varios procedimientos para determinar de forma aproximada soluciones de ecuaciones de la forma $f(x) = 0$ cuando el cálculo exacto no es posible. La mayor parte de los métodos aproximados se basan en el estudio de la función f.

Previamente hay que asegurarse de que la ecuación $f(x) = 0$ tiene al menos una solución en un cierto intervalo y, en caso de que sea posible, averiguar si es única. Para este propósito, los siguientes resultados suelen ser de interés.

Teorema de Bolzano. *Si f es continua en el intervalo $[a, b]$ y toma valores de signo contrario en los extremos del intervalo, $f(a) \cdot f(b) < 0$, entonces se anula al menos en un punto del interior del intervalo.*

Teorema. *Si f es continua en el intervalo cerrado $[a, b]$, derivable en el intervalo abierto (a, b) y además $f'(x) > 0$, $\forall x \in [a, b]$ ó $f'(x) < 0$, $\forall x \in [a, b]$ entonces podemos asegurar que si existe solución de la ecuación $f(x) = 0$ en $[a, b]$, ésta es única en $[a, b]$.*

Teorema de Rolle. *Si f es continua en el intervalo cerrado $[a, b]$, derivable en el intervalo abierto (a, b) y además $f(a) = f(b)$, entonce existe algún punto $\zeta \in (a, b)$ en el cual $f'(\zeta) = 0$.*

Una vez determinado un intervalo en el que la ecuación $f(x) = 0$ posee una solución, se trata de obtener una aproximación de la misma. Los métodos más utilizados para obtener aproximaciones a las soluciones de ecuaciones no lineales son los denominados *métodos iterativos*. Partiendo de una o varias aproximaciones iniciales $x_0, x_1, \ldots, x_k$ mediante estos métodos es posible construir una sucesión de valores que converge a la solución buscada.

Describamos a continuación el método de *Newton-Raphson*, método aproximado de resolución de ecuaciones y sistemas de ecuaciones no lineales.

Si se desea aproximar una solución de la ecuación $f(x) = 0$, siendo f una función real de variable real cualquiera, primero debe de determinarse un intervalo en el que se encuentre la raíz utilizando los teoremas indicados anteriormente.

Partiendo entonces de un cierto valor inicial x_0 el método de Newton-Raphson consiste en obtener la tangente a la curva $y = f(x)$ en el punto de coordenadas $(x_0, f(x_0))$, de ecuación

$$y - f(x_0) = f'(x_0)(x - x_0).$$

(*Tangente 1 de la figura*).

A continuación se obtiene el punto de intersección de dicha tangente con el eje OX, es decir, debe de resolverse el sistema

$$\begin{cases} y - y_0 = f'(x_0)(x - x_0) \\ y = 0 \end{cases}$$

lo que proporciona un nuevo punto de coordenadas $(x_1, f(x_1))$.

El siguiente paso consiste en obtener la tangente a la curva en este nuevo punto, (*Tangente 2 de la figura*) y efectuar nuevamente la intersección de dicha tangente con el eje OX lo que proporciona otro nuevo punto de intersección de coordenadas $(x_2, f(x_2)$ y así sucesivamente.

Bajo ciertas condiciones, en cada iteración el método aproxima la solución de la ecuación $f(x) = 0$.

Al cabo de n repeticiones del procedimiento anterior se sigue que el método consiste en calcular las iteraciones

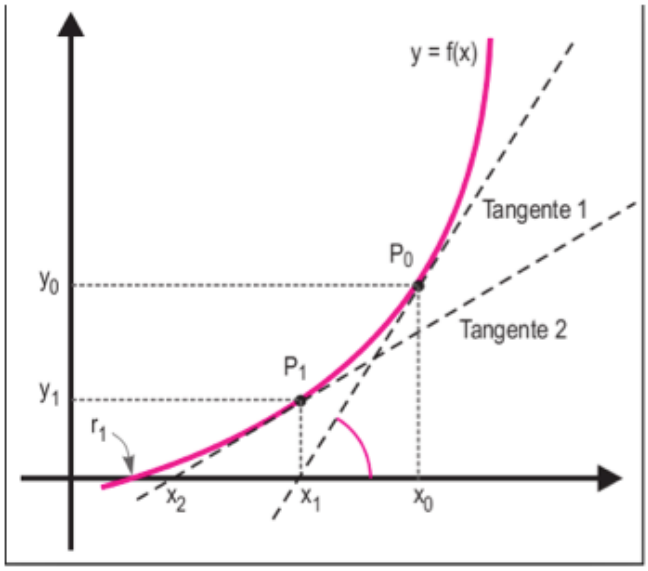
$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}, \qquad n \geq 0,$$

a partir de un valor inicial x_0 y siempre que $f'(x) \neq 0$.

La ventaja del método es que en general, proporciona una convergencia bastante rápida a la solución de la ecuación $f(x) = 0$ para una elección adecuada del punto inicial y siendo f suficientemente derivable con derivada no nula.

Geométricamente, en cada paso estamos calculando la tangente a la curva $y = f(x)$ en el punto $(x_n, f(x_n))$ y haciendo la intersección de dicha tangente con el eje OX, lo que proporciona el punto x_{n+1}.

Los primeros pasos de dicho procedimiento pueden apreciarse en la gráfica siguiente:


y = f(x)
Tangente 1
Tangente 2
P0
P1
y0
y1
r1
x2
x1
x0

3 - Método de Newton-Raphson. Sistemas de ecuaciones no lineales -

En el caso de un sistema de n ecuaciones no lineales con n incógnitas,

$$\begin{cases} f_1(x_1, x_2, \ldots, x_n) & = 0 \\ f_2(x_1, x_2, \ldots, x_n) & = 0 \\ \ldots\ldots & \ldots \\ f_n(x_1, x_2, \ldots, x_n) & = 0 \end{cases}$$

el procedimiento anterior se escribe en términos de la matriz Jacobiana de las n funciones reales de n variables que constituyen el sistema,

$$\begin{pmatrix} x_1' \\ x_2' \\ \ldots \\ x_n' \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \\ \ldots & \ldots & \ldots \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \end{pmatrix}^{-1} \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \ldots\ldots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix}$$

o bien en términos del operador *MATLAB "división por la izquierda"*

$$\begin{pmatrix} x_1' \\ x_2' \\ \ldots \\ x_n' \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \\ \ldots \\ x_n \end{pmatrix} - \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \\ \ldots & \ldots & \ldots \\ \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \ldots \frac{\partial f_1}{\partial x_n} \end{pmatrix} \setminus \begin{pmatrix} f_1(x_1, x_2, \ldots, x_n) \\ f_2(x_1, x_2, \ldots, x_n) \\ \ldots\ldots \\ f_n(x_1, x_2, \ldots, x_n) \end{pmatrix}$$

que ha sido el procedimiento utilizado a lo largo de nuestro TFG.

Referencias

- Richard L. Burden, J. Douglas Faires *"Análisis Numérico"*, Editorial: Cengage Learning. Décima edición.

- http://www.sc.ehu.es/sbweb/fisica3/numerico/raices/raices5.html

- https://sites.google.com/site/procesosnumericosmlg/home/metodos-abiertos/metodo-newton

---

ANEXO III:

Código de Matlab.

initialdatainv.m

```
clear all; clc;

% Se realiza el dimensionado del sistema:
% Longitudes de cada barra (en m)
r0 = 0.245;
r1 = 0.105;
r2 = 0.220;
r3 = 0.150;
a = 0.150;
r4 = r3 + a;
r5 = 0.30;
r6 = r4 + 0.345;
Lo = 0;

rG1 = r1/2;
rG2 = r2/2;
rG3 = 0.29047;
rG4 = 0.28949;
rG5 = r5/2;

% Posición incial de los puntos fijos del sistema
xO = [0;0];
xC = [r0;0];
xF = [r0+r5;0];

% Se define la velocidad de los puntos conocidos
vO = [0;0];
vC = [0;0];
vF = [0;0];
for i=1:361
    w1(i) = 35*2*pi/60;
    % El motor gira a rpm constantes, se pasa a rad/s.
end

% Se define la aceleración de los puntos conocidos
alpha1 = deal(zeros(1,361));
aO = [0;0];
aC = [0;0];
aF = [0;0];

% Masas del los sólidos del sistema (en kg)
m1 = 0.08821;
m2 = 0.17773;
m3 = 0.43452;
m4 = 0.43758;
m5 = 0.24013;

% Momentos de inercia de cada uno de los sólidos (Kg*m^2)
I1 = 0.0003443403325;
I2 = 0.002934475;
I3 = 0.050703593;
I4 = 0.050773125;
I5 = 0.00191677341;
```

```
% Fuerzas externas del sistema
GF = [0;9.81];          % Fuerza de la gravedad
valor = 4.5;
```

DINAMICAINVERSA.m

```
% Se prepara el entorno para trabajar sobre limpio
clear all; clc; clear variables;

% Primero, se carga el fichero con los datos del dimensionado del sistema:

initialdatainv;

% Comprobación de la Ley de Grashof
S = min([r0 r1 r2 r3]);
L = max([r0 r1 r2 r3]);
T = sum([r0 r1 r2 r3]);
PQ = T-S-L;

if (S+L<PQ)
        disp(['El enlace cumple la Ley de Grashof'])
else
        disp('El enlace no cumple la Ley de Grashof')
        return;
end
```

```
N = 361;
fprintf('Resolvemos las ecuaciones de posición a través del metodo NEWTON RAPHSON\n');
for i = 1:N
    fprintf('Iteración número: %0.0f\n', i);
        theta1(i) = (i-1)*(2*pi)/(N-1);

    % Cálculos correspondientes de NR para resolver las ecuaciones de la posición
    xo=[55*pi/180;125*pi/180]; % Valor inicial de theta2 y theta3 para el sistema de
ecuaciones no lineales
    syms x y
    % Se escribe el sistema de ecuaciones la posición
    fpos=[r0-r1*cos(theta1(i))-r2*cos(x)+r3*cos(y);
          r1*sin(theta1(i))+r2*sin(x)-r3*sin(y)];
    dfpos=jacobian(fpos); % Derivada del sistema de ecuaciones
    tolerancia = 0.000000001;
    maxiter = 30;
    iter=0;
    f=inline(fpos,'x','y');
    jf=inline(dfpos,'x','y');
    error=norm(f(xo(1),xo(2)),2);
    while error > tolerancia
        fxo=f(xo(1),xo(2));
        fpxo=jf(xo(1),xo(2));
        x1=xo-fpxo\fxo;
        fx1=f(x1(1),x1(2));
        error=norm((fx1),2);
        if iter > maxiter
            fprintf('Numero máximo de iteraciones alcanzado \n');
            return;
        end
        xo=x1;
        iter=iter+1;
    end
    fprintf('Error=%12.12f\n', error);
    fprintf('%i\t%7.9f\t%7.9f\t \n', iter,x1(1),x1(2));
```

```
    theta1(i) = theta1(i);
    theta2(i) = x1(1);
    theta3(i) = x1(2);
    theta4(i) = theta3(i);


    % Se calculan los vectores unitarios para los angulos calculados que
    % permiten saber la posición de los puntos del sistema
    [e1,n1] = UnitVector(0);
    [e2,n2] = UnitVector(theta1(i));
    [e3,n3] = UnitVector(theta2(i));
    [e4,n4] = UnitVector(theta3(i));

    % Se calculan las posiciones de los puntos en función de los ángulos
    xA(:,i) = FindPos(xO,r1,e2);
    xB(:,i) = FindPos(xC,r3,e4);
    xD(:,i) = FindPos(xC,r4,e4);
    xE(:,i) = FindPos(xF,r4,e4);
    xG(:,i) = FindPos(xC,r6,e4);
    xH(:,i) = FindPos(xF,r6,e4);
end

% Se grafica la trayectoria de cada uno de los enlaces del sistema
figure
plot(xA(1,:),xA(2,:),xB(1,:),xB(2,:),xD(1,:),xD(2,:),xE(1,:),xE(2,:),xG(1,:),xG(2,:),xH(
1,:),xH(2,:))

% Se especifica el ángulo en el que graficar el sistema
hold on;
itheta1 = 350;

% Se grafican todos los sólidos del sistema
hold on;
% Barra 1
plot([xO(1) xA(1,itheta1)],[xO(2) xA(2,itheta1)],'Linewidth',2,'Color','k');
% Barra 2
plot([xA(1,itheta1) xB(1,itheta1)],[xA(2,itheta1)
xB(2,itheta1)],'Linewidth',2,'Color','k');
% Barra 3
plot([xC(1) xB(1,itheta1)],[xC(2) xB(2,itheta1)],'Linewidth',2,'Color','k');
plot([xC(1) xD(1,itheta1)],[xC(2) xD(2,itheta1)],'Linewidth',2,'Color','k');
plot([xC(1) xG(1,itheta1)],[xC(2) xG(2,itheta1)],'Linewidth',2,'Color','k');
% Barra 4
plot([xF(1) xE(1,itheta1)],[xF(2) xE(2,itheta1)],'Linewidth',2,'Color','k');
plot([xF(1) xH(1,itheta1)],[xF(2) xH(2,itheta1)],'Linewidth',2,'Color','k');
% Barra 5
plot([xD(1,itheta1) xE(1,itheta1)],[xD(2,itheta1)
xE(2,itheta1)],'Linewidth',2,'Color','k');

% Se grafican los enlaces
hold on;
plot([xO(1) xC(1) xF(1) xA(1,itheta1) xB(1,itheta1) xD(1,itheta1) xE(1,itheta1)],[xO(2)
xC(2) xF(2) xA(2,itheta1) xB(2,itheta1) xD(2,itheta1)
xE(2,itheta1)],'o','MarkerSize',5,'MarkerFaceColor','k','Color','k');

% Se pinta la etiqueta de cada enlace
text(xO(1)+1,xO(2)-3,'O','HorizontalAlignment','left');
text(xA(1,itheta1)-3,xA(2,itheta1)+1,'A','HorizontalAlignment','right');
```

```
text(xB(1,itheta1)+3,xB(2,itheta1)+1,'B','HorizontalAlignment','left');
text(xC(1)+1,xC(2)-3,'C','HorizontalAlignment','left');
text(xD(1,itheta1)-1,xD(2,itheta1)+3,'D','HorizontalAlignment','right');
text(xE(1,itheta1)+1,xE(2,itheta1)+3,'E','HorizontalAlignment','left');
text(xF(1)+1,xF(2)-3,'F','HorizontalAlignment','left');
text(xG(1,itheta1)-1,xG(2,itheta1)+3,'G','HorizontalAlignment','right');
text(xH(1,itheta1)+1,xH(2,itheta1)+3,'H','HorizontalAlignment','left');

axis equal;
grid on;

title('Trayectoria de los enlaces del sistema mecanico')
xlabel('Posicion en x [m]')
ylabel('Posicion en y [m]')
legend('Punto A', 'Punto B','Punto D', 'Punto E', 'Punto G', 'Punto
H','Location','SouthEast')
```

```
N = 361;
for i = 1:N
    % Se calculan las velocidades
    A = [-r2*sin(theta2(i)), r3*sin(theta3(i));
         -r2*cos(theta2(i)), r3*cos(theta3(i))];
    B = [r1*w1(i)*sin(theta1(i));
         r1*w1(i)*cos(theta1(i))];
    X = A\B;
    w2(i) = X(1);
    w3(i) = X(2);
    w4(i) = w3(i);
    W(:,i) = [w1(i);w2(i);w3(i);w4(i)];

    % Se calculan los vectores unitarios para los ángulos calculados
    [e1,n1] = UnitVector(0);
    [e2,n2] = UnitVector(theta1(i));
    [e3,n3] = UnitVector(theta2(i));
    [e4,n4] = UnitVector(theta3(i));

    % Se calculan las velocidades de los puntos en función de los angulos
    vA(:,i) = FindVel(vO,r1,w1(i),n2);
    velA(1,i) = norm(vA(:,i),2);
    vB(:,i) = FindVel(vC,r3,w3(i),n4);
    velB(1,i) = norm(vB(:,i),2);
    vD(:,i) = FindVel(vC,r4,w3(i),n4);
    velD(1,i) = norm(vD(:,i),2);
    vE(:,i) = FindVel(vF,r4,w4(i),n4);
    velE(1,i) = norm(vE(:,i),2);
    vG(:,i) = FindVel(vC,r6,w3(i),n4);
    velG(1,i) = norm(vG(:,i),2);
    vH(:,i) = FindVel(vF,r6,w4(i),n4);
    velH(1,i) = norm(vH(:,i),2);
end
figure
plot(theta1*180/pi,velH(1,:),'Color',[0 73/255 255/255])
legend('velH','Location','Southeast')
title('Velocidad del punto H')
xlabel('Ángulo de theta1 (grados)')
ylabel('Velocidad (m/s)')
grid on
```

```
set(gca,'xtick',0:60:360)
xlim([0 360])
```

```
N = 361;
for i = 1:N
    % Se calculan las aceleraciones
    A = [-r2*sin(theta2(i)), r3*sin(theta3(i));
         -r2*cos(theta2(i)), r3*cos(theta3(i))];
    B =
[r1*alpha1(i)*sin(theta1(i))+r1*(w1(i)^2)*cos(theta1(i))+r2*(w2(i)^2)*cos(theta2(i))-
r3*(w3(i)^2)*cos(theta3(i));
         r1*alpha1(i)*cos(theta1(i))-r1*(w1(i)^2)*sin(theta1(i))-
r2*(w2(i)^2)*sin(theta2(i))+r3*(w3(i)^2)*sin(theta3(i))];
    X = A\B;
    alpha2(i) = X(1);
    alpha3(i) = X(2);
    alpha4(i) = alpha3(i);
    ALPHA(:,i) = [alpha1(i);alpha2(i);alpha3(i);alpha4(i)];

    % Se calculan los vectores unitarios para los angulos calculados
    [e1,n1] = UnitVector(0);
    [e2,n2] = UnitVector(theta1(i));
    [e3,n3] = UnitVector(theta2(i));
    [e4,n4] = UnitVector(theta3(i));

    aa = r1*w1(i)^2;
    ab = r3*w3(i)^2;
    ad = r4*w3(i)^2;
    ae = r4*w4(i)^2;
    ag = r6*w3(i)^2;
    ah = r6*w4(i)^2;

    % Se calculan las aceleraciones de los puntos en función de los angulos
    aA(:,i) = FindAcc(aO,r1,w1(i),alpha1(i),e2,n2);
    accA(1,i) = norm(aA(:,i),2);
    aB(:,i) = FindAcc(aC,r3,w3(i),alpha3(i),e4,n4);
    accB(1,i) = norm(aB(:,i),2);
    aD(:,i) = FindAcc(aC,r4,w3(i),alpha3(i),e4,n4);
    accD(1,i) = norm(aD(:,i),2);
    aE(:,i) = FindAcc(aF,r4,w4(i),alpha4(i),e4,n4);
    accE(1,i) = norm(aE(:,i),2);
    aG(:,i) = FindAcc(aC,r6,w3(i),alpha3(i),e4,n4);
    accG(1,i) = norm(aG(:,i),2);
    aH(:,i) = FindAcc(aF,r6,w4(i),alpha4(i),e4,n4);
    accH(1,i) = norm(aH(:,i),2);
end
figure
plot(theta1*180/pi,accG(1,:),'Color',[89/255 255/255 0])
legend('Aceleración G','Location','Southeast')
title('Aceleración del punto G')
xlabel('Ángulo de theta1 (grados)')
ylabel('Aceleración (m/s^2)')
grid on
set(gca,'xtick',0:60:360)
xlim([0 360])
```

```
N = 361;
for i = 1:N
```

```
    % Se calculan los vectores unitarios para los angulos calculados que
    % permiten saber la posicion de los puntos del sistema
    [e1,n1] = UnitVector(0);
    [e2,n2] = UnitVector(theta1(i));
    [e3,n3] = UnitVector(theta2(i));
    [e4,n4] = UnitVector(theta3(i));

    % Se calculan las posiciones de los puntos en función de los angulos
    xG1(:,i) = FindPos(xO,r1/2,e2);
    xG2(:,i) = FindPos(xA(i),r3/2,e3);
    xG3(:,i) = FindPos(xC,r4/2,e4);
    xG4(:,i) = FindPos(xF,r4/2,e4);
    xG5(:,i) = FindPos(xD(i),r5/2,e4);

    % Se calcula la aceleración de los centros de masa
    gammaabsG1(:,i) = [-rG1*alpha1(i)*sin(theta1(i))-rG1*w1(i)^2*cos(theta1(i));
                       rG1*alpha1(i)*cos(theta1(i))-rG1*w1(i)^2*sin(theta1(i))];
    gammaabsA(:,i) = [-r1*alpha1(i)*sin(theta1(i))-r1*w1(i)^2*cos(theta1(i));
                      r1*alpha1(i)*cos(theta1(i))-r1*w1(i)^2*sin(theta1(i))];
    gammaabsG2(:,i) = [-r1*alpha1(i)*sin(theta1(i))-r1*w1(i)^2*cos(theta1(i))-
rG2*alpha2(i)*sin(theta2(i))-rG2*w2(i)^2*cos(theta2(i));
                       r1*alpha1(i)*cos(theta1(i))-
r1*w1(i)^2*sin(theta1(i))+rG2*alpha2(i)*cos(theta2(i))-rG2*w2(i)^2*sin(theta2(i))];
    gammaabsG3(:,i) = [-rG3*alpha3(i)*sin(theta3(i))-rG3*w3(i)^2*cos(theta3(i));
                       rG3*alpha3(i)*cos(theta3(i))-rG3*w3(i)^2*sin(theta3(i))];
    gammaabsG4(:,i) = [-rG4*alpha4(i)*sin(theta4(i))-rG4*w4(i)^2*cos(theta4(i));
                       rG4*alpha4(i)*cos(theta4(i))-rG4*w4(i)^2*sin(theta4(i))];
    gammaabsG5(:,i) = [-r3*alpha3(i)*sin(theta3(i))-r3*w3(i)^2*cos(theta3(i));
                       r3*alpha3(i)*cos(theta3(i))-r3*w3(i)^2*sin(theta3(i))];

    % Se calcula la fuerza de rozamiento de las barras 3 y 4
    vG3(:,i) = [-rG3*w3(i)*sin(theta3(i));
              rG3*w3(i)*cos(theta3(i))];
    vG4(:,i) = vG3(:,i);

    FrG3(:,i) = [valor*(-vG3(1,i)/abs(vG3(1,i)))-vG3(1,i)/norm(vG3(:,i),2);
                 valor*(-vG3(2,i)/abs(vG3(2,i)))-vG3(2,i)/norm(vG3(:,i),2)];
    FrG4(:,i) = [valor*(-vG4(1,i)/abs(vG4(1,i)))-vG4(1,i)/norm(vG4(:,i),2);
                 valor*(-vG4(2,i)/abs(vG4(2,i)))-vG4(2,i)/norm(vG4(:,i),2)];

    % Se calcula el sistema con 15 ecuaciones obtenidas al aplicar los
    % teoremas y se escribe el vector de cada fuerza.
    A = [0,1,0,1,0,0,0,0,0,0,0,0,0,0,0;
         0,0,1,0,1,0,0,0,0,0,0,0,0,0,0;
         1,0,0,-r1*sin(theta1(i)),r1*cos(theta1(i)),0,0,0,0,0,0,0,0,0,0;
         0,0,0,-1,0,1,0,0,0,0,0,0,0,0,0;
         0,0,0,0,-1,0,1,0,0,0,0,0,0,0,0;
         0,0,0,0,0,-r2*sin(theta2(i)),r2*cos(theta2(i)),0,0,0,0,0,0,0,0;
         0,0,0,0,0,-1,0,1,0,1,0,0,0,0,0;
         0,0,0,0,0,0,-1,0,1,0,1,0,0,0,0;
         0,0,0,0,0,r3*sin(theta3(i)),-r3*cos(theta3(i)),0,0,-
r4*sin(theta3(i)),r4*cos(theta3(i)),0,0,0,0;
         0,0,0,0,0,0,0,0,0,0,0,-1,0,1,0;
         0,0,0,0,0,0,0,0,0,0,0,0,-1,0,1;
         0,0,0,0,0,0,0,0,0,0,0,r4*sin(theta4(i)),-r4*cos(theta4(i)),0,0;
         0,0,0,0,0,0,0,0,0,-1,0,1,0,0,0;
         0,0,0,0,0,0,0,0,0,0,-1,0,1,0,0;
         0,0,0,0,0,0,0,0,0,0,0,0,0,r5,0,0];
```

```
    B = [m1*gammaabsG1(1,i);
        m1*gammaabsG1(2,i)+m1*GF(2);
        I1*alpha1(i)+m1*GF(2)*rG1*cos(theta1(i));
        m2*gammaabsG2(1,i);
        m2*gammaabsG2(2,i)+m2*GF(2);
        I2*alpha2(i)+m2*GF(2)*rG2*cos(theta2(i))-m2*rG2*(gammaabsA(1,i)*sin(theta2(i))-
gammaabsA(2,i)*cos(theta2(i)));
        m3*gammaabsG3(1,i)-FrG3(1,i);
        m3*gammaabsG3(2,i)+m3*GF(2)-FrG3(2,i);
        I3*alpha3(i)+(m3*GF(2)-
FrG3(2,i))*rG3*cos(theta3(i))+FrG3(1,i)*rG3*sin(theta3(i));
        m4*gammaabsG4(1,i)-FrG4(1,i);
        m4*gammaabsG4(2,i)+m4*GF(2)-FrG4(2,i);
        I4*alpha4(i)+(m4*GF(2)-
FrG4(2,i))*rG4*cos(theta4(i))+FrG4(1,i)*rG4*sin(theta4(i));
        m5*gammaabsG5(1,i);
        m5*gammaabsG5(2,i)+m5*GF(2);
        m5*r5*(GF(2)+gammaabsG5(2,i))];

    X = A\B;

    par(:,i) = X(1);
    Fo(:,i) = [X(2);X(3)];
    Fa(:,i) = [X(4);X(5)];
    Fb(:,i) = [X(6);X(7)];
    Fc(:,i) = [X(8);X(9)];
    Fd(:,i) = [X(10);X(11)];
    Fe(:,i) = [X(12);X(13)];
    Ff(:,i) = [X(14);X(15)];
    partotal(:,i) = norm(par(:,i),2);
end

% Se representa el par motor
figure
plot(theta1*180/pi,partotal(1,:),'Color',[89/255 255/255 0])
legend('Par motor','Location','Southeast')
title('Par motor')
xlabel('Ángulo de theta1 (grados)')
ylabel('Par motor (N*m)')
grid on
set(gca,'xtick',0:60:360)
xlim([0 360])

% Se representan la fuerza
figure
tiledlayout(1,2)
nexttile([1 1])
plot(theta1*180/pi,Fe(1,:)*(-1),'Color',[89/255 255/255 0])
set(gca,'xtick',0:60:360)
xlim([0 360])
xlabel('Angulo de theta1 (grados)')
ylabel('Fuerza (N)')
title('Fex')
nexttile([1 1])
plot(theta1*180/pi,Fe(2,:)*(-1),'Color',[89/255 255/255 0])
set(gca,'xtick',0:60:360)
xlim([0 360])
```

```
xlabel('Angulo de theta1 (grados)')
ylabel('Fuerza (N)')
title('Fey')

% Se representa la fuerza de rozamiento
figure
plot(theta1*180/pi,abs(FrG3(1,:)),'Color',[89/255 255/255 0])
legend('FrG3','Location','Southeast')
title('Fuerza de rozamiento')
xlabel('Ángulo de theta1 (grados)')
ylabel('Fuerza de rozamiento (N)')
grid on
set(gca,'xtick',0:60:360)
xlim([0 360])
```

UnitVector.m

```
% Función utilizada para crear vectores unitarios según los ángulos del
% sistema
function [e,n] = VectorUnitario(theta)
    e = [cos(theta);sin(theta)];
    n = [-sin(theta); cos(theta)];
end
```

FindPos.m

```
% Esta función calcula la posición de un punto conocido un vector
% unitario y la longitud que debe tener.
% x0 = Punto inicial desde el que empieza el vector
% L = Longitud del vector
% e = Vector unitario que indica la dirección y sentido
% x = Punto final del vector

function x = EncontrarPosicion(x0,L,e)
    x = x0+L*e;
end
```

FindVel.m

```
% Esta función calcula la velocidad de un punto conocido el vector normal,
% la velocidad inicial, la velocidad angular y la distancia al punto de
% giro.
% v0 = Velocidad inicial del punto a estudiar
% L = Distancia al punto de giro
% n = Vector normal que indica la dirección y sentido
% w = Velocidad angular del punto
function y = calcularvelocidad(v0,L,w,n)
    y = v0+L*w*n;
end
```

FindAce.m

```
% Esta función calcula la aceleración de un punto conocido el vector
% normal y el vector unitario, la aceleración inicial, la velocidad
% angular, la aceleración angular y la distancia al punto de giro.
% a0 = Aceleración inicial del punto a estudiar
% L = Distancia al punto de giro
% n = Vector normal que indica la dirección y sentido
% w = Velocidad angular del punto
function a = calcularaceleracion(a0,L,w,alpha,e,n)
    a = a0+L*alpha*n-L*(w^2)*e;
end
```

ANEXO IV:

Sistema de ecuaciones dinámica “directa”.

Para resolver el mecanismo por dinamica “directa” conociendo el par motor, se añaden las ecuaciones de la aceleración para tener un sistema de17 ecuaciones y 17 incognitas.

De este modo, en primer lugar, se define

$a = -m_1 g r_{G1} cos\theta_1$

$b = -m_2 g\, r_{G2} cos\theta_2 + m_2\, r_{G2}\left(a_{Ax} sen\theta_2 - a_{Ay} cos\theta_2\right)$

$c = \left(-m_3 g + F_{RG3y}\right) r_{G3} cos\theta_3 - F_{RG3x} r_{G3} sen\theta_3$

$d = \left(-m_4 g + F_{RG4y}\right) r_{G4} cos\theta_4 - F_{RG4x} r_{G4} sen\theta_4$

obteniendo un sistema tal que:

- Variables: $\ddot{\theta}_1, \ddot{\theta}_2, \ddot{\theta}_3, F_{Ox,}\, F_{Oy,} F_{Ax,} F_{Ay,} F_{Bx,} F_{By,} F_{Cx,} F_{Cy,} F_{Dx,} F_{Dy,}\, F_{Ex,}\, F_{Ey,} F_{Fx,} F_{Fy}$

Ec (1) $F_{Ox} + F_{Ax} = m_1 a_{G1x} \rightarrow F_{Ox} = m_1 a_{G1x} - F_{Ax}$

Ec (2) $F_{Oy} + F_{Ay} = m_1 a_{G1y} + m_1 g \rightarrow F_{Oy} = m_1 a_{G1y} - F_{Ay}$

Ec (3) $\Gamma + \mathrm{a} + r_1\left(-F_{Ax} sen\theta_1 + F_{Ay} cos\theta_1\right) = I_{Oz}\ddot{\theta}_1$

Ec (4) $-F_{Ax} + F_{Bx} = m_2 a_{G2x} \rightarrow F_{Ax} = -m_2 a_{G2x} + F_{Bx}$

Ec (5) $-F_{Ay} + F_{By} - m_2 g = m_2 a_{G2y} \rightarrow F_{Ay} = -m_2 a_{G2y} + F_{By} - m_2 g$

Ec (6) $b - r_2\left(F_{Bx} sen\theta_2 - F_{By} cos\theta_2\right) = I_A \ddot{\theta}_2 \rightarrow F_{Bx} = \left[F_{By} cos\theta_2 - \left(I_A\ddot{\theta}_2 - b\right)/r_2\right]/sen\theta_2$

Ec (7) $F_{Cx} - F_{Bx} + F_{Dx} + F_{RG3x} = m_3 a_{G3x} \rightarrow F_{Cx} = m_3 a_{G3x} + F_{BX} - F_{Dx} - F_{RG3x}$

Ec (8) $F_{Cy} - F_{By} + F_{Dy} - m_3 g + F_{RG3y} = m_3 a_{G3y} \rightarrow F_{Cy} = m_3 a_{G3y} + F_{By} - F_{Dy} + m_3 g - F_{RG3y}$

Ec (9) $c + r_3\left(F_{Bx} sen\theta_3 - F_{By} cos\theta_3\right) - (r_3 + a)\left(F_{Dx} sen\theta_3 - F_{Dy} cos\theta_3\right) = I_{Cz}\ddot{\theta}_3$

Ec (10) $F_{Fx} - F_{Ex} + F_{RG4x} = m_4 a_{G4x} \rightarrow F_{FX} = m_4 a_{G4x} - F_{RG4x} + F_{Ex}$

Ec (11) $F_{Fy} - F_{Ey} - m_4 g + F_{RG4y} = m_4 a_{G4y} \rightarrow F_{Fy} = m_4 a_{G4y} - F_{RG4y} + F_{Ey} + m_4 g$

Ec (12) $d + (r_4 + a)\left(F_{Ex} sen\theta_4 - F_{Ey} cos\theta_4\right) = I_{Fz}\ddot{\theta}_4 \rightarrow F_{EX} = \left[F_{Ey} cos\theta_4 + \left(I_{Fz}\ddot{\theta}_4 - d\right)/(r_4 + a)\right]/sen\theta_4$

Ec (13) $-F_{Dx} + F_{Ex} = m_5 a_{G5x} \rightarrow F_{DX} = F_{Ex} - m_5 a_{G5x}$

Ec (14) $-F_{Dy} + F_{Ey} = m_5 a_{G5y} + m_5 g \rightarrow F_{Dy} = F_{Ey} - \left(m_5 a_{G5y} + m_5 g\right)$

Ec (15) $r_5 F_{Ey} = m_5\left(a_{Dy} + g\right) r_{G5} \rightarrow F_{Ey} = m_5\left(a_{Dy} + g\right) r_{G5}/r_5$

Ec (16) $-r_2\ddot{\theta}_2 sen\theta_2 + r_3\ddot{\theta}_3 sen\theta_3 = r_1\ddot{\theta}_1\, sen\theta_1 + r_1\dot{\theta}_1^2 cos\theta_1 + r_2\dot{\theta}_2^2 cos\theta_2 - r_3\dot{\theta}_3^2 cos\theta_3$

Ec (17) $-r_2\ddot{\theta}_2 cos\theta_2 + r_3\ddot{\theta}_3 cos\theta_3 = r_1\ddot{\theta}_1\, cos\theta_1 - r_1\dot{\theta}_1^2 sen\theta_1 - r_2\dot{\theta}_2^2 sen\theta_2 + r_3\dot{\theta}_3^2 sen\theta_3$