UNIVERSIDAD DE PIURA



**INFORME DE TRABAJO**

“Estudio de mecanismo con Matlab”

**ASIGNATURA :** “Teoría de Máquinas (TMA)”

**PROFESOR :** Dr. Ing. Miguel Castro.

**NOMBRE :** Juan Esteban Paz Jáuregui.

Contenido

[1 Introducción 3](#_Toc372557116)

[2 Objetivos 3](#_Toc372557117)

[3 Cálculo de Posiciones 5](#_Toc372557118)

[4 Simulación del Movimiento del Mecanismo 7](#_Toc372557119)

[5 Cálculo de las Velocidades 10](#_Toc372557120)

[6 Cálculo de las Aceleraciones 17](#_Toc372557121)

[7 Calculo de Fuerzas Dinámicas 26](#_Toc372557122)

[8 Dimensionamiento de los pasadores o Pines 29](#_Toc372557123)

[9 Simulación con vectores de Fuerzas en los pines 30](#_Toc372557124)

[10 Conclusiones 31](#_Toc372557125)

[11 Referencias 31](#_Toc372557126)

[12 Anexos 32](#_Toc372557127)

[Anexo A Intersección Linea – Linea 32](#_Toc372557128)

[Anexo B Intersección Linea – Circunferencia 33](#_Toc372557129)

[Anexo C Intersección Circunferencia - Circunferencia 34](#_Toc372557130)

[Anexo D código distancia Mínima 36](#_Toc372557131)

# Introducción

Para lograr un estudio completo de un mecanismo, es necesario realizar muchas iteraciones las cuales resultarían muy tediosas para ser realizadas a mano; es por esto que se ha recurrido a la programación en MatLab para poder automatizar los cálculos para un rango completo de posibles posiciones.

Para esto ha sido necesario estudiar el mecanismo en una posición y generalizar su solución.

# Objetivos

* Diseñar un conjunto de programas en Matlab que permitan estudiar la cinemática del mecanismo
* Aprender a construir un programa para una simulación dinámica
* Entender el funcionamiento y los principios de cálculo de los simuladores de mecanismos.
* Generar aplicaciones prácticas para el uso de la teoría de máquinas.

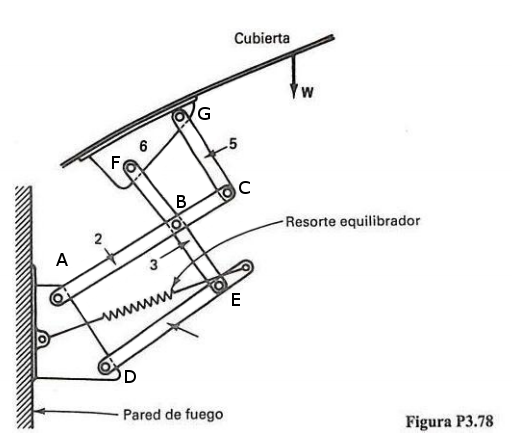
El mecanismo de la figura tiene dimensiones 6 barras, con 7 puntos para el estudio cinemático las condiciones iniciales consideradas son las siguientes:

* A=[0 0 0]
* AD=[-0.5i +0.2j]
* AB=0.5
* BC=0.6
* DE=0.6
* BE=0.5
* EF=1.2
* FG=0.6
* GC=0.5

Condiciones Iniciales

* Φ2 =pi/16;
* ω 2 =[0, 0, 2];
* Ω 2= [0 0 -1 ];

El ángulo inicial es de 11.5° con respecto a la horizontal. Encontrar la posición, velocidad y aceleración de los puntos G y E.



# Cálculo de Posiciones

pos.m

Para el cálculo de las posiciones de todos los puntos del mecanismo se ha recurrido a una solución geométrica por medio de intersecciones de líneas y círculos, implementadas como funciones:

Intersección línea - línea: linlin()[[1]](#footnote-1)

Intersección línea – circunferencia: lincir()[[2]](#footnote-2)

Intersección circunferencia - circunferencia: circir()[[3]](#footnote-3)

Además para poder descartar y escoger soluciones, hace falta discriminarlas con una función de distancia mínima.

Distancia Mínima: distMinima()[[4]](#footnote-4)

clear all

clc

close all

Datos iniciales mencionados anteriormente

%DATOS INICIALES

AB=0.5; BC=0.6; %%barra2

DE=0.6;%barra4

yAD=-0.5;xAD=0.2;%1

BE=0.5;EF=1.2; %barra3

FG=0.6;GC=0.5;%barra5

phi=pi/4;

Partiendo de los datos iniciales hay dos posiciones que ya conocemos las del punto A y D

%CÁLCULO DE LAS POSICIONES

xA=0;

yA=0;

y=0;

xD=xA+xAD;

yD=yA+yAD;

Tambien se puede conocer las coordenadas de B y C debido a que se encuentran en la misma barra, que ya tiene un origen (A) y orientación conocida.

%Calculo Posiciones con datos conocidos

xB=AB\*cos(phi);

yB=AB\*sin(phi);

xC=(AB+BC)\*cos(phi);

yC=(AB+BC)\*sin(phi);

Para poder emplear las funciones de intersección y discriminación mencionadas anteriormente, hace falta introducir puntos de referencia que permitan seleccionar adecuadamente una primera discriminación.

%Puntos de referencia iniciales

xEref=DE;

yFref=BE;

xFref=xAD;

xGref=AB;

%Calculo de posiciones desconocidas

Para calcular la posición del punto E se intersecta dos circunferencias la primera de centro B y radio BE y la segunda de centro D y radio DE, posteriormente se realiza la discriminación de soluciones basada en los puntos de referencia.

%punto E circulo rDE,c rBE

%Para el cálculo de la posición faltante

[ xE1,yE1,xE2,yE2 ] = circir( xB,yB,BE,xD,yD,BE);

% Se escoge una de las dos soluciones

[ xE,yE ] = distMinima( xEref,yD,xE1,yE1,xE2,yE2);

De similar manera para encontrar el punto F se require intersectar una recta BE con una circunferencia de centro E y radio EF, y discriminarla de igual manera.

%punto F circulo rEF, linea BE

%Para el cálculo de la posición faltante

[ xF1,yF1,xF2,yF2 ] = lincir( xE,yE,xB,yB,xE,yE,EF);

% Se escoge una de las dos soluciones

[ xF,yF ] = distMinima( xFref,yFref,xF1,yF1,xF2,yF2);

Y por último para el cálculo del punto G se requiere nuevamente de una intersección entre dos circunferencias, la primera de centro F y radio FG y la segunda de centro C y radio GC

%punto G circulo rFG,c rGC

%Para el cálculo de la posición faltante

[ xG1,yG1,xG2,yG2 ] = circir( xF,yF,FG,xC,yC,GC);

% Se escoge una de las dos soluciones

[ xG,yG ] = distMinima( xGref,yF,xG1,yG1,xG2,yG2);

Por último se requiere presentar los datos encontrados para cada punto, por lo que se procede a mostrar cada vector posición.

%IMPRIMIR RESULTADOS

fprintf('Posiciones Calculadas para phi=%0.2f\n',phi);

fprintf('A: %0.2fi + (%0.2fj)(m) \n',xA,yA);

fprintf('B: %0.2fi + (%0.2fj)(m) \n',xB,yB);

fprintf('C: %0.2fi + (%0.2fj)(m) \n',xC,yC);

fprintf('D: %0.2fi + (%0.2fj)(m) \n',xD,yD);

fprintf('E: %0.2fi + (%0.2fj)(m) \n',xE,yE);

fprintf('F: %0.2fi + (%0.2fj)(m) \n',xF,yF);

fprintf('G: %0.2fi + (%0.2fj)(m) \n',xG,yG);

Para una ayuda gráfica, también se genera un gráfico de la posición del problema.

%GRAFICAR EL MECANISMO

figure(1);

hold off

plot([xA,xB,xC],[yA,yB,yC],'r-o');

hold on;

plot([xF,xB,xE],[yF,yB,yE],'g-o');

plot([xD,xE],[yD,yE],'b-o');

plot([xC,xG],[yC,yG],'c-o');

text(xA,yA,' A'); text(xB,yB,' B'); text(xC,yC,' C');

text(xD,yD,' D'); text(xE,yE,' E'); text(xF,yF,' F');

text(xG,yG,' G');

title('Gráfica de posiciones')

xlabel('x (m)')

ylabel('y (m)')

axis([-0.8 2 -0.8 2]);

grid

Después de ejecutar este programa se presentan los datos a continuación:

*Posiciones Calculadas para phi=0.79*

D:\UDEP\TMA\Trabajo\POS.epsA: 0.00i + (0.00j)(m)

B: 0.35i + (0.35j)(m)

C: 0.78i + (0.78j)(m)

D: 0.20i + (-0.50j)(m)

E: 0.52i + (-0.12j)(m)

F: 0.12i + (1.01j)(m)

G: 0.66i + (1.26j)(m)

# Simulación del Movimiento del Mecanismo

sim.n

Repitiendo los cálculos anteriores para un rango de ángulos se puede lograr visualizar el movimiento del mecanismo

Una vez que se tienen las coordenadas se puede graficar el movimiento completo del mecanismo para visualizar su funcionamiento correcto. En este caso se debe evitar que el programa se confunda cuando tiene dos puntos de solución para el mecanismo.

Este tipo de situaciones se resuelve considerando también como datos de entrada la posición inicial de todos los puntos del mecanismo que necesitan ser calculador. De esta forma por continuidad en el movimiento del mecanismo se puede llegar a escoger el punto correcto del movimiento. Es decir entre una posición y otra el nuevo punto debe estar cerca del punto anteriormente. Esto se resuelve de forma matemática calculando la distancia más cercana entre dos puntos.

Este programa se mencionó anteriormente distancia Mínima, se puede estudiarlo en el anexo D

A continuación se presenta el programa para la simulación de movimiento.

clear all

clc

close all

%DATOS INICIALES

AB=0.5; BC=0.6; %%barra2

DE=0.6;%barra4

yAD=-0.5;xAD=0.2;%1

BE=0.5;EF=1.2; %barra3

FG=0.6;GC=0.5;%barra5

phi=pi/16;

%CÁLCULO DE LAS POSICIONES

xA=0;

yA=0;

y=0;

xD=xA+xAD;

yD=yA+yAD;

Para poder realizar la primera discriminación de las respectivas intersecciones se introduce valores de referencia para la primera iteración,

%Puntos de referencia iniciales

xEref=DE;

yFref=BE;

xFref=xAD;

xGref=AB;

%Se define el paso de la simulación

Paso=pi/180;

Debido a que el movimiento del mecanismo es oscilante, se agrega un bucle for para conseguir invertir el sentido de incremento de ángulo a decremento para generar la oscilación.

Además se debe definir el angulo completo de recorrido en este caso es 45 grados.

for i=1:-2:-1

for I=phi:Paso\*i:phi+pi\*i/4

Se procede a calcular las posiciones dependientes del ángulo (Los puntos B y C)

Esta parte del programa es igual a los cálculos realizados en la anterior, con la diferencia de que ahora los valores dependen de la variable I que cambia en el rango definido en el bucle.

xB=AB\*cos(I);

yB=AB\*sin(I);

xC=(AB+BC)\*cos(I);

yC=(AB+BC)\*sin(I);

%punto E circulo rDE,c rBE

%Para el cálculo de la posición faltante

[ xE1,yE1,xE2,yE2 ] = circir( xB,yB,BE,xD,yD,BE);

% Se escoge una de las dos soluciones

[ xE,yE ] = distMinima( xEref,yD,xE1,yE1,xE2,yE2);

%punto F circulo rEF, linea BE

%Para el cálculo de la posición faltante

[ xF1,yF1,xF2,yF2 ] = lincir( xE,yE,xB,yB,xE,yE,EF);

% Se escoge una de las dos soluciones

[ xF,yF ] = distMinima( xFref,yFref,xF1,yF1,xF2,yF2);

%punto G circulo rFG,c rGC

%Para el cálculo de la posición faltante

[ xG1,yG1,xG2,yG2 ] = circir( xF,yF,FG,xC,yC,GC);

% Se escoge una de las dos soluciones

[ xG,yG ] = distMinima( xGref,yF,xG1,yG1,xG2,yG2);

Una vez calculados todos los puntos del mecanismo se procede a plotear la nueva posicion de este, utilizando el comando hold off (no mantener los plots anteriores) y hold on (mantener todos los plots anteriores) se puede dibujar una posición que varía para cada iteración resultando en una animación continua.

figure(1);

hold off

plot([xA,xB,xC],[yA,yB,yC],'r-o');

hold on;

plot([xF,xB,xE],[yF,yB,yE],'g-o');

plot([xD,xE],[yD,yE],'b-o');

plot([xC,xG],[yC,yG],'c-o');

text(xA,yA,' A'); text(xB,yB,' B'); text(xC,yC,' C');

text(xD,yD,' D'); text(xE,yE,' E'); text(xF,yF,' F');

text(xG,yG,' G');

title('Simulación de las posiciones')

xlabel('x (m)')

ylabel('y (m)')

axis([-0.8 2 -0.8 2]);

grid

Como se mencionó anteriormente las nuevas referencias para las iteraciones que siguen a la primera se actualizan a las coordenadas encontradas.

%Se actualiza la refrencia a la última coordenada calculada

xEref=xE;

yFref=yF;

xFref=xF;

xGref=xG;

pause(0.01)

end

phi=I;

end

La simulación de forma resumida produce las siguientes imágenes cuyo código fuente se encuentra en el anexo:

D:\UDEP\TMA\Trabajo\sim1.epsD:\UDEP\TMA\Trabajo\sim2.epsD:\UDEP\TMA\Trabajo\sim3.epsD:\UDEP\TMA\Trabajo\sim4.eps

# Cálculo de las Velocidades

veloc.m

Para poder obtener una gráfica de todas las velocidades que se dan en el movimiento del mecanismo es necesario resolver las ecuaciones dinámicas para cada una de las posiciónes del angulo φ, para esto se debe calcular de forma vectorial cada una de las posiciones de los puntos del mecanismo. También es necesario introducir otro valor inicial, la velocidad angular de la barra 2 biela que para este ejemplo se tomará 2 rad/s.

El programa para su cálculo se muestra a continuación:

%CALCULO DE LAS VELOCIDADES EN CADA POSICIÓN

clear all

clc

close all

%DATOS INICIALES

AB=0.5; BC=0.6; %%barra2

DE=0.6;%barra4

yAD=-0.5;xAD=0.2;%1

BE=0.5;EF=1.2; %barra3

FG=0.6;GC=0.5;%barra5

phi=pi/16;

omega2=[0, 0, 2];

%CÁLCULO DE LAS POSICIONES

xA=0;

yA=0;

y=0;

xD=xA+xAD;

yD=yA+yAD;

%Puntos de referencia iniciales

xEref=DE;

yEref=yD;

yFref=BE;

xFref=xAD;

xGref=AB;

yGref=yA+FG;

%Calculo Posiciones con datos conocidos

xB=AB\*cos(phi);

yB=AB\*sin(phi);

xC=(AB+BC)\*cos(phi);

yC=(AB+BC)\*sin(phi);

%Se define el paso de la simulación

Paso=pi/180;

J=1;

Se incrementa variables simbólicas para la solución de las ecuaciones dinámicas

%variable simbólicas para el eslabón 2

omega3z = sym('omega3z','real');

omega4z = sym('omega4z','real');

omega5z = sym('omega5z','real');

omega6z = sym('omega6z','real');

for I=phi:Paso:phi+pi/4

%Almacenar los ángulos

ang(J)=I;

%Calculo Posiciones con datos conocidos

El cálculo de posiciones se mantiene igual para este caso, en realidad es el punto de partida para el cálculo de las velocidades.

xB=AB\*cos(I);

yB=AB\*sin(I);

xC=(AB+BC)\*cos(I);

yC=(AB+BC)\*sin(I);

%punto E circulo rDE,c rBE

%Para el cálculo de la posición faltante

[ xE1,yE1,xE2,yE2 ] = circir( xB,yB,BE,xD,yD,BE);

% Se escoge una de las dos soluciones

[ xE,yE ] = distMinima( xEref,yEref,xE1,yE1,xE2,yE2);

%punto F circulo rEF, linea BE

%Para el cálculo de la posición faltante

[ xF1,yF1,xF2,yF2 ] = lincir( xE,yE,xB,yB,xE,yE,EF);

% Se escoge una de las dos soluciones

[ xF,yF ] = distMinima( xFref,yFref,xF1,yF1,xF2,yF2);

%punto G circulo rFG,c rGC

%Para el cálculo de la posición faltante

[ xG1,yG1,xG2,yG2 ] = circir( xF,yF,FG,xC,yC,GC);

% Se escoge una de las dos soluciones

[ xG,yG ] = distMinima( xGref,yGref,xG1,yG1,xG2,yG2);

%Actualizacion de Referencias

xEref=xE;

yFref=yF;

xFref=xF;

xGref=xG;

yGref=yG;

%Contrucción de los vectores posición

rA=[xA, yA, 0];

rB=[xB, yB, 0];

rC=[xC, yC, 0];

rD=[xD, yD, 0];

rE=[xE, yE, 0];

rF=[xF, yF, 0];

rG=[xG, yG, 0];

Partiendo de que el punto A está fijo y que omega2 es conocido se puede calcular fácilmente la velocidad en B con la ecuación

vA=[0, 0, 0 ]; %en m/s

%Barra 1

%B y C Estan en la misma barra

%Cálculo de la velocidad en B

vB(J,:) = vA + cross(omega2,rB);

vBmod(J)=norm(vB(J,:));

%cálculo de la velocidad en C

vC(J,:) = vA + cross(omega2,rC);

vCmod(J)=norm(vC(J,:));

De similar manera Partiendo de que el punto D está fijo y que omega3 y 4 son desconocidos se puede calcular la velocidad en E planteando un sistema de ecuaciones con las ecuaciones

Para esto la ecuación que se introduce en Matlab sería

Una vez obtenidos se puede calcular VE con una de las ecuaciones anteriores y se calcula los módulos de las 3 dentro de una matriz que contiene cada uno de los vectores para J posiciones del mecanismo.

%Barra 2

vD=[0, 0, 0 ]; %en m/s

%Barra 2 y Barra 3

%F, B y E Estan en la misma barra

%Para esto es necesario crear dos variable simbólicas cuya declaración

%se pone fuera del for porque se hace una sola vez

Omega3 = [ 0 0 omega3z ];

Omega4 = [ 0 0 omega4z ];

%Se usa las ecuaciónes vE = vD + w4×(rE -rD)

%y vE = vB + w3×(rE -rB) que en Matlab es

eqvE= -( vD+ cross(Omega4,rE- rD))+ vB(J,:) + cross(Omega3,rE-rB);

%como la ecuación anterior es vectorial la convertimos en dos

%algebraicas

eqvEx = eqvE(1); % Ecuación en X

eqvEy = eqvE(2); % Ecuación en Y

solvE = solve(eqvEx,eqvEy);

omega3zs = eval(solvE.omega3z);

omega4zs = eval(solvE.omega4z);

omega3(J,:) = [0, 0, omega3zs];

omega3mod(J)=norm(omega3(J,:));

omega4(J,:) = [0, 0, omega4zs];

omega4mod(J)=norm(omega4(J,:));

vE(J,:) = vD + cross(omega4(J,:),rE-rD);

vEmod(J)=norm(vE(J,:));

vF(J,:) = vE(J,:) + cross(omega3(J,:),rF-rE);

vFmod(J)=norm(vF(J,:));

De similar manera Partiendo de que se conoce las el punto F y C está fijo y que omega3 y 4 son desconocidos se puede calcular la velocidad en E planteando un sistema de ecuaciones con las ecuaciones

Para esto la ecuación que se introduce en Matlab sería

Una vez obtenidos se puede calcular VG con una de las ecuaciones anteriores y se calcula los módulos de las 3 dentro de una matriz que contiene cada uno de los vectores para J posiciones del mecanismo.

Y J se incrementa en 1

%Barra 4 y Barra 5

%Fy G barra 5, C y G barra 4

Omega5 = [ 0 0 omega5z ];

Omega6 = [ 0 0 omega6z ];

%Se usa las ecuaciónes vG = vF + w6×(rG -rF)

%y vG = vC + w5×(rG -rF) que en Matlab es

eqvG=-( vF(J,:)+cross(Omega6,rG- rF)) + vC(J,:) + cross(Omega5,rC-rF);

%como la ecuación anterior es vectorial la convertimos en dos

%algebraicas

eqvGx = eqvG(1); % Ecuación en X

eqvGy = eqvG(2); % Ecuación en Y

solvG = solve(eqvGx,eqvGy);

omega5zs = eval(solvG.omega5z);

omega6zs = eval(solvG.omega6z);

omega5(J,:) = [0, 0, omega5zs];

omega5mod(J)=norm(omega5(J,:));

omega6(J,:) = [0, 0, omega6zs];

omega6mod(J)=norm(omega6(J,:));

vG(J,:) = vF(J,:)+cross(omega6(J,:),rG- rF);

vGmod(J)=norm(vG(J,:));

J=J+1;

End

Se procede a graficar todos los valores de los módulos de las velocidades agrupando los puntos de cada una de las barras

%%BARRA 2

figure(1);

title('Velocidades Barra 2')

subplot(2,1,1);

plot(ang\*180/pi,vBmod);

title('Modulo de la velocidad en B'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad (m/s)')

grid

subplot(2,1,2);

plot(ang\*180/pi,vCmod);

title('Modulo de la velocidad en C'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

ylabel('Velocidad (m/s)')

grid

%%BARRA 3

figure(2);

title('Velocidades Barra 3')

subplot(3,1,1);

plot(ang\*180/pi,vEmod);

title('Modulo de la velocidad en E'...

,'Color','b','FontWeight','bold')

grid

subplot(3,1,2);

plot(ang\*180/pi,vBmod);

title('Modulo de la velocidad en B'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad (m/s)')

grid

subplot(3,1,3);

plot(ang\*180/pi,vFmod);

title('Modulo de la velocidad en F'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

grid

%%BARRA 5 GC

figure(3);

title('Velocidades Barra 5')

subplot(2,1,1);

plot(ang\*180/pi,vGmod);

title('Modulo de la velocidad en G'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad (m/s)')

grid

subplot(2,1,2);

plot(ang\*180/pi,vCmod);

title('Modulo de la velocidad en C'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

ylabel('Velocidad (m/s)')

grid

De la misma manera se grafica todas las velocidades angulares del mecanismo agrupadas.

%Graficas de velocidades angulares

figure(4);

title('Velocidades angulares')

subplot(4,1,1);

plot(ang\*180/pi,omega3mod);

title('Modulo de la velocidad angular en la barra3'...

,'Color','b','FontWeight','bold')

grid

subplot(4,1,2);

plot(ang\*180/pi,omega4mod);

title('Modulo de la velocidad angular en la barra4'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad angular (rad/s)')

grid

subplot(4,1,3);

plot(ang\*180/pi,omega5mod);

title('Modulo de la velocidad angular en la barra5'...

,'Color','b','FontWeight','bold')

grid

subplot(4,1,4);

plot(ang\*180/pi,omega6mod);

title('Modulo de la velocidad angular en la barra6'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

grid

Las gráficas generadas son:

D:\UDEP\TMA\Trabajo\vel1.eps

Figura 5‑1 Velocidades en la Barra 2

D:\UDEP\TMA\Trabajo\vel4.eps

Figura 5‑2 Velocidades Barra 3

D:\UDEP\TMA\Trabajo\vel2.eps

Figura 5‑3 Velocidades en la barra 5

D:\UDEP\TMA\Trabajo\vel3.eps

Figura 5‑4 velocidades angulares

# Cálculo de las Aceleraciones

acel.m

Las aceleraciones se calculan en cada una de las diferentes posiciones del ángulo φ y es necesario conocer la velocidad angular de todas las barras. Además se considera que la barra 2 tiene una aceleración angular de -1 rad/s2.

Este programa requiere las soluciones del anterior, por esto dentro de este programa se encuentran los códigos para cálculo de posiciones y velocidades mostrados anteriormente por separado.

El programa para su cálculo se muestra a continuación:

%CALCULO DE LAS VELOCIDADES EN CADA POSICIÓN

clear all

clc

close all

%DATOS INICIALES

AB=0.5; BC=0.6; %%barra2

DE=0.6;%barra4

yAD=-0.5;xAD=0.2;%1

BE=0.5;EF=1.2; %barra3

FG=0.6;GC=0.5;%barra5

phi=pi/16;

omega2=[0, 0, 2];

Se introduce la aceleración angular de la barra 2.

alpha2 = [0 0 -1 ]; % (rad/sˆ2)

%CÁLCULO DE LAS POSICIONES

xA=0;

yA=0;

y=0;

xD=xA+xAD;

yD=yA+yAD;

%Puntos de referencia iniciales

xEref=DE;

yEref=yD;

yFref=BE;

xFref=xAD;

xGref=AB;

yGref=yA+FG;

%Calculo Posiciones con datos conocidos

xB=AB\*cos(phi);

yB=AB\*sin(phi);

xC=(AB+BC)\*cos(phi);

yC=(AB+BC)\*sin(phi);

%Se define el paso de la simulación

Paso=pi/180;

J=1;

%variable simbólicas para el eslabón 2

omega3z = sym('omega3z','real');

omega4z = sym('omega4z','real');

omega5z = sym('omega5z','real');

omega6z = sym('omega6z','real');

Se genera una nueva serie de variables simbólicas para las aceleraciones angulares

%variable simbólicas para el eslabón 2: Aceleración

alpha3z=sym('alpha3z','real');

alpha4z=sym('alpha4z','real');

alpha5z=sym('alpha5z','real');

alpha6z=sym('alpha6z','real');

for I=phi:Paso:phi+pi/4

Cabe recalcar que los procedimientos para el cálculo de las posiciones y velocidades son los mismos que se han empleado en los códigos anteriores.

%Almacenar los ángulos

ang(J)=I;

%Calculo Posiciones con datos conocidos

xB=AB\*cos(I);

yB=AB\*sin(I);

xC=(AB+BC)\*cos(I);

yC=(AB+BC)\*sin(I);

%punto E circulo rDE,c rBE

%Para el cálculo de la posición faltante

[ xE1,yE1,xE2,yE2 ] = circir( xB,yB,BE,xD,yD,BE);

% Se escoge una de las dos soluciones

[ xE,yE ] = distMinima( xEref,yEref,xE1,yE1,xE2,yE2);

%punto F circulo rEF, linea BE

%Para el cálculo de la posición faltante

[ xF1,yF1,xF2,yF2 ] = lincir( xE,yE,xB,yB,xE,yE,EF);

% Se escoge una de las dos soluciones

[ xF,yF ] = distMinima( xFref,yFref,xF1,yF1,xF2,yF2);

%punto G circulo rFG,c rGC

%Para el cálculo de la posición faltante

[ xG1,yG1,xG2,yG2 ] = circir( xF,yF,FG,xC,yC,GC);

% Se escoge una de las dos soluciones

[ xG,yG ] = distMinima( xGref,yGref,xG1,yG1,xG2,yG2);

%Actualizacion de Referencias

xEref=xE;

yFref=yF;

xFref=xF;

xGref=xG;

yGref=yG;

%Contrucción de los vectores posición

rA=[xA, yA, 0];

rB=[xB, yB, 0];

rC=[xC, yC, 0];

rD=[xD, yD, 0];

rE=[xE, yE, 0];

rF=[xF, yF, 0];

rG=[xG, yG, 0];

vA=[0, 0, 0 ]; %en m/s

%Barra 1

%B y C Estan en la misma barra

%Cálculo de la velocidad en B

vB(J,:) = vA + cross(omega2,rB);

vBmod(J)=norm(vB(J,:));

%cálculo de la velocidad en C

vC(J,:) = vA + cross(omega2,rC);

vCmod(J)=norm(vC(J,:));

%Barra 2

vD=[0, 0, 0 ]; %en m/s

%Barra 2 y Barra 3

%F, B y E Estan en la misma barra

%Para esto es necesario crear dos variable simbólicas

%cuya declaración se pone fuera del for porque

%se hace una sola vez

Omega3 = [ 0 0 omega3z ];

Omega4 = [ 0 0 omega4z ];

%Se usa las ecuaciónes vE = vD + w4×(rE -rD)

%y vE = vB + w3×(rE -rB) que en Matlab es

eqvE=-( vD+cross(Omega4,rE- rD))...

+ vB(J,:) + cross(Omega3,rE-rB);

%como la ecuación anterior es vectorial

%la convertimos en dos algebraicas

eqvEx = eqvE(1); % Ecuación en X

eqvEy = eqvE(2); % Ecuación en Y

solvE = solve(eqvEx,eqvEy);

omega3zs = eval(solvE.omega3z);

omega4zs = eval(solvE.omega4z);

omega3(J,:) = [0, 0, omega3zs];

omega3mod(J)=norm(omega3(J,:));

omega4(J,:) = [0, 0, omega4zs];

omega3mod(J)=norm(omega4(J,:));

vE(J,:) = vD + cross(omega4(J,:),rE-rD);

vEmod(J)=norm(vE(J,:));

vF(J,:) = vE(J,:) + cross(omega3(J,:),rF-rE);

vFmod(J)=norm(vF(J,:));

%Barra 5 y Barra 6

%Fy G barra 6, C y G barra 5

Omega5 = [ 0 0 omega5z ];

Omega6 = [ 0 0 omega6z ];

%Se usa las ecuaciónes vG = vF + w6×(rG -rF)

%y vG = vC + w5×(rG -rF) que en Matlab es

eqvG=-( vF(J,:)+cross(Omega6,rG- rF))...

+ vC(J,:) + cross(Omega5,rC-rF);

%como la ecuación anterior es vectorial

%la convertimos en dos algebraicas

eqvGx = eqvG(1); % Ecuación en X

eqvGy = eqvG(2); % Ecuación en Y

solvG = solve(eqvGx,eqvGy);

omega5zs = eval(solvG.omega5z);

omega6zs = eval(solvG.omega6z);

omega5(J,:) = [0, 0, omega5zs];

omega5mod(J)=norm(omega5(J,:));

omega6(J,:) = [0, 0, omega6zs];

omega6mod(J)=norm(omega6(J,:));

vG(J,:) = vF(J,:)+cross(omega6(J,:),rG- rF);

vGmod(J)=norm(vG(J,:));

El Cálculo de las aceleraciones inicia partiendo de que los puntos A y D son fijos y que ya se conoce el valor de alfa2 con lo que se puede conocer la aceleración en B y en C

%Cálculo de la aceleración en B

aA = [0 0 0 ];

aD = [0 0 0 ];

aB(J,:) = aA + cross(alpha2,rB-rA)...

- dot(omega2,omega2)\*(rB-rA);

aBmod(J)=norm(aB(J,:));

%Cálculo de la aceleración en C

aC(J,:) = aA + cross(alpha2,rC-rA)...

- dot(omega2,omega2)\*(rC-rA);

aCmod(J)=norm(aC(J,:));

Una vez que se conoce las aceleraciones en los puntos B y D se plantea un sistema de ecuaciones para la aceleración de E.

Para esto la ecuación que se introduce en Matlab sería

Una vez obtenidos se puede calcular VG con una de las ecuaciones anteriores y se calcula los módulos de las 3 dentro de una matriz que contiene cada uno de los vectores para J posiciones del mecanismo.

%%Aceleración en E

Alpha3 = [0 0 alpha3z ]; % alpha3z unknown

Alpha4 = [0 0 alpha4z ]; % alpha3z unknown

eqaE=-(aB(J,:)+cross(Alpha3,rE-rB)...

-dot(omega3(J,:),omega3(J,:))\*(rE-rB))...

+aD+cross(Alpha4,rE-rD)...

-dot(omega4(J,:),omega4(J,:))\*(rE-rD);

eqaEx = eqaE(1); % Ecuación en X

eqaEy = eqaE(2); % Ecuación en Y

solaE = solve(eqaEx,eqaEy);

alpha3zs=eval(solaE.alpha3z);

alpha4zs=eval(solaE.alpha4z);

alpha3(J,:) = [0 0 alpha3zs];

alpha3mod(J)=norm(alpha3(J,:));

alpha4(J,:) = [0 0 alpha4zs];

alpha4mod(J)=norm(alpha4(J,:));

aE(J,:)=aB(J,:)+cross(alpha3(J,:),rE-rB)...

-dot(omega3(J,:),omega3(J,:))\*(rE-rB);

aEmod(J)=norm(aE(J,:));

aF(J,:)=aB(J,:)+cross(alpha3(J,:),rF-rB)...

-dot(omega3(J,:),omega3(J,:))\*(rF-rB);

aFmod(J)=norm(aF(J,:));

Una vez que se conoce las aceleraciones en los puntos F y C se plantea un sistema de ecuaciones para la aceleración de G.

Para esto la ecuación que se introduce en Matlab sería

Una vez obtenidos se puede calcular VG con una de las ecuaciones anteriores y se calcula los módulos de las 3 dentro de una matriz que contiene cada uno de los vectores para J posiciones del mecanismo.

%%Acel G

Alpha5 = [ 0 0 alpha5z ]; % alpha5z unknown

Alpha6 = [ 0 0 alpha6z ]; % alpha6z unknown

eqaG=-(aF(J,:)+cross(Alpha6,rG-rF)...

-dot(omega6(J,:),omega6(J,:))\*(rG-rF))+...

aC(J,:)+cross(Alpha5,rG-rC)...

-dot(omega5(J,:),omega5(J,:))\*(rG-rC);

eqaGx = eqaG(1); % Ecuación en X

eqaGy = eqaG(2); % Ecuación en Y

solaE = solve(eqaGx,eqaGy);

alpha5zs=eval(solaE.alpha5z);

alpha6zs=eval(solaE.alpha6z);

alpha5(J,:) = [0 0 alpha5zs];

alpha5mod(J)=norm(alpha5(J,:));

alpha6(J,:) = [0 0 alpha6zs];

alpha6mod(J)=norm(alpha6(J,:));

aG(J,:)=aF(J,:)+cross(alpha6(J,:),rG-rF)...

-dot(omega6(J,:),omega6(J,:))\*(rG-rF);

aGmod(J)=norm(aG(J,:));

J=J+1;

end

Por último se procede a graficar las variaciones en las aceleraciones para los puntos de cada barra.

%%BARRA 1 BC

figure(1);

title('Aceleraciones en la Barra 1')

subplot(2,1,1);

plot(ang\*180/pi,aBmod);

title('Módulo de la aceleración en B'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

ylabel('Aceleración (m/s2)')

grid

subplot(2,1,2);

plot(ang\*180/pi,aCmod);

title('Módulo de la aceleración en C'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

ylabel('Aceleración (m/s2)')

grid

%%BARRA 2 EBF

figure(2);

title('Aceleraciones en la Barra 2')

subplot(3,1,1);

plot(ang\*180/pi,aEmod);

title('Módulo de la aceleración en E'...

,'Color','b','FontWeight','bold')

grid

subplot(3,1,2);

plot(ang\*180/pi,aBmod);

title('Módulo de la aceleración en B'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad (m/s)')

grid

subplot(3,1,3);

plot(ang\*180/pi,aFmod);

title('Módulo de la aceleración en F'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

grid

%%BARRA 5 GC

figure(3);

title('Velocidades Barra 5')

subplot(2,1,1);

plot(ang\*180/pi,aGmod);

title('Módulo de la aceleración en G'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad (m/s)')

grid

subplot(2,1,2);

plot(ang\*180/pi,aCmod);

title('Módulo de la aceleración en C'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

ylabel('Velocidad (m/s)')

grid

Se Muestra las gráficas de las aceleraciones angulares

%Graficas de aceleraciones angulares

figure(4);

title('Aceleraciónes angulares')

subplot(4,1,1);

plot(ang\*180/pi,alpha3mod);

title('Modulo de la aceleración angular en la barra 2'...

,'Color','b','FontWeight','bold')

grid

subplot(4,1,2);

plot(ang\*180/pi,alpha4mod);

title('Modulo de la aceleración angular en la barra 3'...

,'Color','b','FontWeight','bold')

ylabel('Velocidad angular (rad/s)')

grid

subplot(4,1,3);

plot(ang\*180/pi,alpha5mod);

title('Modulo de la aceleración angular en la barra 4'...

,'Color','b','FontWeight','bold')

grid

subplot(4,1,4);

plot(ang\*180/pi,alpha6mod);

title('Modulo de la aceleración angular en la barra 5'...

,'Color','b','FontWeight','bold')

xlabel('Angulo \phi (Grados)')

grid

Las gráficas generadas son:

D:\UDEP\TMA\Trabajo\acel1.eps

Figura 6‑1 Aceleraciones en la barra 2

D:\UDEP\TMA\Trabajo\acel2.eps

Figura 6‑2 Aceleraciones en la barra 3

D:\UDEP\TMA\Trabajo\acel3.eps

Figura 6‑3 aceleraciones en la barra 5

D:\UDEP\TMA\Trabajo\acel4.eps

Figura 6‑4 aceleraciones angulares en el mecanismo

# Calculo de Fuerzas Dinámicas

Forces.m

Una vez obtenidas las velocidades y aceleraciones angulares de cada barra se procede a calcular las fuerzas por medio del principio de dAlembert

Como punto de partida se requiere conocer todas las masas e inercias de cada elemento, para simplificar el proceso se ha escogido una densidad lineal, para geometrías complicadas se debe ingresar los datos de masa e inercia para cada elemento. Las inercias para las barras 2 y 4 deben estar calculadas respecto al eje de giro.

%Calculo Masas,Inercias

lAC=AB+BC;

lDE=DE;

lFE=BE;

lGC=GC;

lFG=FG;

pesol=1;%Kg/m

m2=pesol\*lAC;

m3=pesol\*lFE;

m4=pesol\*lDE;

m5=pesol\*lGC;

m6=pesol\*lFG;

Iac=1/12\*m2\*lAC^2;

IacA=Iac+m2\*lAC^2/4;

Ide=1/12\*m4\*lDE^2;

IdeD=Ide+m4\*lDE^2/4;

Ife=1/12\*m3\*lFE^2;

Igc=1/12\*m5\*lGC^2;

Ifg=1/12\*m6\*lFG^2;

Posteriormente se define las variables simbólicas, Fuerzas en los pines (A,B,C,D,E,F,G)

%%Fuerzas Incognitas

FA = [sym('FAx','real') sym('FAy','real') 0 ];

F32 = [ sym('F32x','real') sym('F32y','real') 0 ];

F52 = [ sym('F52x','real') sym('F52y','real') 0 ];

F63 = [sym('F63x','real') sym('F63y','real') 0 ];

F43 = [sym('F43x','real') sym('F43y','real') 0 ];

F65 = [sym('F65x','real') sym('F65y','real') 0 ];

FD = [sym('FDx','real') sym('FDy','real') 0 ];

Se puede apreciar que no se ha considerado como incognita al valor del torque de entrada, en realidad si es una incognita pero se calcula despejando una ecuacion, con el fin de que para la solucion del Sistema de ecuaciones se tenga N incognitas y N ecuaciones.

Para crear el sistema de ecuaciones se debe realizar un diagrama de cuerpo libre de cada elemento, y plantear las ecuaciones de equilibrio de fuerzas y momento dinámicos, cabe aclarar que se debe calcular las aceleraciones en los centros de gravedad o en los ejes de giro (A,D), a continuación se encuentra el análisis para cada una de las barras.

%%FUERZAS%%

%BARRA 2 %FA,F32,F52,m2 a2CG,IacA

a2cg(J,:)=cross(alpha2,(rF+rE)/2)-dot(omega2,omega2)\*(rF+rE)/2;

eqF2=FA+F32+F52-m2\*a2cg(J,:);

eqF2x = eqF2(1);

eqF2y = eqF2(2);

eqM2 = cross(rB-rA,F32)+cross(rC-rA,F52)-IacA\*alpha2;

eqM2z = eqM2(3);

%BARRA 3 %F63,23,43,m3,a3cg,Ife

a3cg(J,:)=aE(J,:)+cross(alpha3s(J,:),(rF+rE)/2)...

-dot(omega3s(J,:),omega3s(J,:))\*(rF+rE)/2;

F23 = -F32;

eqF3 = F23+F63+F43-m3\*a3cg(J,:);

eqF3x = eqF3(1);

eqF3y = eqF3(2);

eqM3 = cross(rB-(rE+rF)/2,F32)+cross(rF-(rE+rF)/2,F52)-Ife\*alpha2;

eqM3z = eqM3(3);

%BARRA 5 F25,65,m5,a5cg

a5cg(J,:)=aC(J,:)+cross(alpha5s(J,:),(rG+rC)/2)...

-dot(omega5s(J,:),omega5s(J,:))\*(rG+rC)/2;

F25 = -F52;

eqF5 = F25+F65+-m5\*a5cg(J,:);

eqF5x = eqF5(1);

eqF5y = eqF5(2);

eqM5 = cross(rG-(rC+rG)/2,F65)-Igc\*alpha5s(J,:);

eqM5z = eqM5(3);

%BARRA 6 F56,F36,m6,a6cg,Ifg

a6cg(J,:)=aF(J,:)+cross(alpha6s(J,:),(rG+rF)/2)...

-dot(omega6s(J,:),omega6s(J,:))\*(rG+rF)/2;

F56 = -F65;

F36 = -F63;

eqF6 = F56+F36-m6\*a6cg(J,:);

eqF6x = eqF6(1);

eqF6y = eqF6(2);

eqM6 = cross(rG-(rF+rG)/2,F56)-Ifg\*alpha6s(J,:);

eqM6z = eqM6(3);

%BARRA 4 F34, FD,m4,a4cg,

a4cg(J,:)=cross(alpha4s(J,:),(rE+rD)/2)...

-dot(omega4s(J,:),omega4s(J,:))\*(rE+rD)/2;

F34 = -F43;

eqF4 = FD+F34-m4\*a4cg(J,:);

eqF4x = eqF4(1);

eqF4y = eqF4(2);

\*\*En las ecuaciones de la barra 4 se ha omitido la ecuación de equilibrio de momentos dinámicos, como se mencionó anteriormente esta se operará después de resolver el sistema

Una vez obtenidas todas las ecuaciones de equilibrio requeridas se resuelve el sistema de ecuaciones y posteriormente se construye los vectores de fuerzas

sol=solve(eqM3z,eqF3y,eqF3x,eqF2x,eqF2y,eqF3y,eqM2z,eqF4x,eqF4y,eqF6x,eqF6y,eqM6z,eqF5x,eqF5y,eqM5z);

F32ys=eval(sol.F32y);

F32xs=eval(sol.F32x);

F52ys=eval(sol.F52y);

F52xs=eval(sol.F52x);

F63ys=eval(sol.F63y);

F63xs=eval(sol.F63x);

F43xs=eval(sol.F43x);

F43ys=eval(sol.F43y);

F65xs=eval(sol.F65x);

F65ys=eval(sol.F65y);

FAxs=eval(sol.FAx);

FAys=eval(sol.FAy);

FDxs=eval(sol.FDx);

FDys=eval(sol.FDy);

F32s(J,:) = [ F32xs, F32ys, 0 ];

F52s(J,:) = [ F52xs, F52ys, 0 ];

F63s(J,:) = [ F63xs, F63ys, 0 ];

F43s(J,:) = [ F43xs, F43ys, 0 ];

F65s(J,:) = [ F65xs, F65ys, 0 ];

F23s(J,:) = -F32s(J,:);

F25s(J,:) = -F52s(J,:);

FAs(J,:) = [FAxs,FAys,0];

FDs(J,:) = [FDxs,FDys,0];

T4s = -cross(rE-rD,F32s(J,:)) + IdeD\*alpha4s(J,:);

xAv(J)=xA;

xBv(J)=xB;

xCv(J)=xC;

xDv(J)=xD;

xEv(J)=xE;

xFv(J)=xF;

xGv(J)=xG;

yAv(J)=yA;

yBv(J)=yB;

yCv(J)=yC;

yDv(J)=yD;

yEv(J)=yE;

yFv(J)=yF;

yGv(J)=yG;

# Dimensionamiento de los pasadores o Pines

Debido a que el tipo de movimiento es oscilante, el dato importante para el cálculo de los pines se requiere únicamente un valor máximo de fuerza en ese punto, una vez concluido el bucle de cálculos, se calcula las dimensiones de los pasadores considerando un factor de seguridad y el esfuerzo último a tracción.

esfuerzoUltimo=400;%MPa

cortanteUltimo=esfuerzoUltimo/sqrt(3);

FS=2;

Tmax=max(T4s);

FBmax =norm(max(F32s));

FCmax =norm(max(F52s));

FFmax =norm(max(F63s));

FEmax =norm(max(F43s));

FGmax =norm(max(F65s));

FAmax=norm(max(FAs));

FDmax=norm(max(FDs));

dB=sqrt(FBmax\*FS/(pi\*cortantefluencia));

dC=sqrt(FCmax\*FS/(pi\*cortantefluencia));

dF=sqrt(FFmax\*FS/(pi\*cortantefluencia));

dE=sqrt(FEmax\*FS/(pi\*cortantefluencia));

dG=sqrt(FGmax\*FS/(pi\*cortantefluencia));

dA=sqrt(FAmax\*FS/(pi\*cortantefluencia));

dD=sqrt(FDmax\*FS/(pi\*cortantefluencia));

fprintf('Requerimiento de Torque:%0.3f\n',Tmax);

fprintf('diametro Pin A:%0.3f\n',dA);

fprintf('diametro Pin B:%0.3f\n',dB);

fprintf('diametro Pin C:%0.3f\n',dC);

fprintf('diametro Pin D:%0.3f\n',dD);

fprintf('diametro Pin E:%0.3f\n',dE);

fprintf('diametro Pin F:%0.3f\n',dF);

fprintf('diametro Pin G:%0.3f\n',dG);

# Simulación con vectores de Fuerzas en los pines

Esta simulación está basada en el programa de simulación de posiciones, a esta se le incrementa los vectores de fuerza con el comando quiver3 (xo,yo,∆x,∆y) y posteriormente se etiqueta cada vector.

figure(1);

for J=1:size(F23s,1)

plot([xAv(J),xBv(J),xCv(J)],[yAv(J),yBv(J),yCv(J)],'r-o');

hold on

plot([xFv(J),xBv(J),xEv(J)],[yFv(J),yBv(J),yEv(J)],'g-o');

plot([xDv(J),xEv(J)],[yDv(J),yEv(J)],'b-o');

plot([xCv(J),xGv(J)],[yCv(J),yGv(J)],'c-o');

text(xAv(J),yAv(J),' A');

text(xBv(J),yBv(J),' B');

text(xCv(J),yCv(J),' C');

text(xDv(J),yDv(J),' D');

text(xEv(J),yEv(J),' E');

text(xFv(J),yFv(J),' F');

text(xGv(J),yGv(J),' G');

quiver3(xAv(J),yAv(J),0,FAs(J,1)\*0.01,FAs(J,2)\*0.01,0);%FA

text(xAv(J)+FAs(J,1)\*forcef,yAv(J)+FAs(J,2)\*forcef,'FA');

quiver3(xBv(J),yBv(J),0,F32s(J,1)\*forcef,F32s(J,2)\*forcef,0);%FB

text(xBv(J)+F32s(J,1)\*forcef,yBv(J)+F32s(J,2)\*forcef,'FB');

quiver3(xCv(J),yCv(J),0,F52s(J,1)\*forcef,F52s(J,2)\*forcef,0);%FC

text(xCv(J)+F52s(J,1)\*forcef,yCv(J)+F52s(J,2)\*forcef,'FC');

quiver3(xDv(J),yDv(J),0,FDs(J,1)\*0.01,FDs(J,2)\*0.01,0);%FD

text(xDv(J)+FDs(J,1)\*forcef,yDv(J)+FDs(J,2)\*forcef,'FD');

quiver3(xEv(J),yEv(J),0,F43s(J,1)\*forcef,F43s(J,2)\*forcef,0);%FE

text(xEv(J)+F43s(J,1)\*forcef,yEv(J)+F43s(J,2)\*0.01,'FE');

quiver3(xFv(J),yFv(J),0,F63s(J,1)\*forcef,F63s(J,2)\*forcef,0);%FF

text(xFv(J)+F63s(J,1)\*forcef,yFv(J)+F63s(J,2)\*forcef,'F63');

quiver3(xGv(J),yGv(J),0,F65s(J,1)\*0.01,F65s(J,2)\*0.01,0);%FG

text(xGv(J)+F65s(J,1)\*forcef,yGv(J)+F65s(J,2)\*forcef,'FG');

hold off

pause(0.1)

J=J+1;

end

# Conclusiones

* Se ha logrado profundizar los métodos de cálculo para mecanismos.
* Se ha aprendido otra aplicación muy útil de matlab para la solución de ecuaciones vectoriales
* Se ha destacado la importancia de un código eficiente, en realidad una simulación en tiempo real es muy difícil conseguir eso, por eso es importante también tener en cuenta que las animaciones no podrán ser en tiempo real.
* La aplicación interesante de este tipo de programas es que se podría además optimizar la geometría de los elementos para reducir las inercias, exigencias a los componentes, etc.

# Referencias

**Dynamic Force Analysis with Matlab.** Chapter 4 Dynamic Forces

# Anexos

## Anexo A Intersección Linea – Linea

% Progarama para calcular el punto de intersección de dos líneas

% DATOS DE INGRESO

% Puntos de la primera línea (Ax,Ay) y (Bx,By)

% Puntos de la segunda línea (Cx,Cy) y (Dx,Dy)

% SALIDA

% Devuelve el punto de intersección (Px,Py)

function [ Px,Py ] = linlin( Ax,Ay,Bx,By,Cx,Cy,Dx,Dy)

% calcular si la primera o segunda recta son verticales

if(Bx-Ax==0) recVertA=1; else recVertA=0; end

if(Dx-Cx==0) recVertB=1; else recVertB=0; end

if(recVertA ==1 && recVertB==1)

error('Las rectas son pareleas');

end

if(recVertA ==1 && recVertB==0) %primera recta vertical

m2=((Dy-Cy)/(Dx-Cx));

Px=Ax;

Py=m2\*(Px-Dx)+Dy;

elseif(recVertA ==0 && recVertB==1)%segunda recta vertical

m1=((By-Ay)/(Bx-Ax));

Px=Cx;

Py=m1\*(Px-Bx)+By;

else %Rectas en cualquier posición

m1=((By-Ay)/(Bx-Ax));

m2=((Dy-Cy)/(Dx-Cx));

%Vericar nuevamente si las recatas no son paralelas

if(m1 == m2)

error('Las rectas son pareleas');

end

Px=(m1\*Ax-m2\*Dx+Dy-Ay)/(m1-m2);

Py=m1\*(Px-Ax)+ Ay;

end

end

## Anexo B Intersección Linea – Circunferencia

% Progarama para calcular el punto de intersección de dos líneas

% DATOS DE INGRESO

% Puntos de la primera línea (Ax,Ay) y (Bx,By)

% Puntos del dentro del circulo (Cx,Cy) y Radio R

% SALIDA

% Devuelve los puntos de intersección (Px,Py) y (Qx,Qy)

function [ Px,Py,Qx,Qy ] = lincir( Ax,Ay,Bx,By,Cx,Cy,R)

% Verificar si la recta es vertical o no

if(Bx-Ax==0) % La recta es vertical

%verificar que exista intersección

tmp=R^2-(Ax-Cx)^2;

if(tmp<0)

error('No existe intersección entre línea y círculo');

end

Px=Ax;

Qx=Ax;

Py=Cy + tmp^0.5;

Qy=Cy - tmp^0.5;

else % La recta no es vertical

m=(By-Ay)/(Bx-Ax);

b=Ay-m\*Ax;

%verificar que exista intersección

tmp=- Cx^2\*m^2 + 2\*Cx\*Cy\*m - 2\*Cx\*b\*m - Cy^2 + 2\*Cy\*b + R^2\*m^2 + R^2 - b^2;

if(tmp<0)

error('No existe intersección entre línea y círculo');

end

Px=(Cx - (tmp)^(1/2) + Cy\*m - b\*m)/(m^2 + 1);

Qx=(Cx + (tmp)^(1/2) + Cy\*m - b\*m)/(m^2 + 1);

Py=m\*Px+b;

Qy=m\*Py+b;

end

end

## Anexo C Intersección Circunferencia - Circunferencia

% Progarama para calcular el punto de intersección de dos líneas

% DATOS DE INGRESO

% Puntos del centro del circulo 1 (Ax,Ay) y Radio R1

% Puntos del dentro del circulo 2 (Bx,By) y Radio R2

% SALIDA

% Devuelve los puntos de intersección (Px,Py) y (Qx,Qy)

function [ Px,Py,Qx,Qy ] = circir( Ax,Ay,R1,Bx,By,R2)

% Verificar si hay intersección

dist = ((Bx-Ax)^2+(By-Ay)^2)^2;

if(dist>R1+R2)

error('No existe intersección entre los círculos');

end

% calcular la intersección

Px=Ax/2 + Bx/2 - (Ax\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ax\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Bx\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (Bx\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ay\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (By\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2);

Qx=Ax/2 + Bx/2 - (Ax\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ax\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Bx\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (Bx\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (Ay\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (By\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2);

Py=Ay/2 + By/2 - (Ay\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ay\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (By\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (By\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (Ax\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Bx\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2);

Qy=Ay/2 + By/2 - (Ay\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ay\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (By\*R1^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (By\*R2^2)/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) + (Ax\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2) - (Bx\*(- Ax^4 + 4\*Ax^3\*Bx - 2\*Ax^2\*Ay^2 + 4\*Ax^2\*Ay\*By - 6\*Ax^2\*Bx^2 - 2\*Ax^2\*By^2 + 2\*Ax^2\*R1^2 + 2\*Ax^2\*R2^2 + 4\*Ax\*Ay^2\*Bx - 8\*Ax\*Ay\*Bx\*By + 4\*Ax\*Bx^3 + 4\*Ax\*Bx\*By^2 - 4\*Ax\*Bx\*R1^2 - 4\*Ax\*Bx\*R2^2 - Ay^4 + 4\*Ay^3\*By - 2\*Ay^2\*Bx^2 - 6\*Ay^2\*By^2 + 2\*Ay^2\*R1^2 + 2\*Ay^2\*R2^2 + 4\*Ay\*Bx^2\*By + 4\*Ay\*By^3 - 4\*Ay\*By\*R1^2 - 4\*Ay\*By\*R2^2 - Bx^4 - 2\*Bx^2\*By^2 + 2\*Bx^2\*R1^2 + 2\*Bx^2\*R2^2 - By^4 + 2\*By^2\*R1^2 + 2\*By^2\*R2^2 - R1^4 + 2\*R1^2\*R2^2 - R2^4)^(1/2))/(2\*Ax^2 - 4\*Ax\*Bx + 2\*Ay^2 - 4\*Ay\*By + 2\*Bx^2 + 2\*By^2);

end

## Anexo D código distancia Mínima

% Progarama para calcular la distancia mínima entre dos puntos

% DATOS DE INGRESO

% Puntos de refererencia (Px,Py)

% Puntos a evaluar (Qx,Qy) y (Rx,Ry)

% SALIDA

% Devuelve el punto (Qx,Qy) o (Rx,Ry) mas cercano

function [ Sx,Sy ] = distMinima( Px,Py,Qx,Qy,Rx,Ry )

%Cálculo de las distancias

dist1=((Px-Qx)^2+(Py-Qy)^2)^2;

dist2=((Px-Rx)^2+(Py-Ry)^2)^2;

if(dist1<dist2)

Sx=Qx;

Sy=Qy;

else

Sx=Rx;

Sy=Ry;

end

end

1. Anexo A function linlin() [↑](#footnote-ref-1)
2. Anexo B function lincir() [↑](#footnote-ref-2)
3. Anexo C function circir() [↑](#footnote-ref-3)
4. Anexo D function distMinima() [↑](#footnote-ref-4)