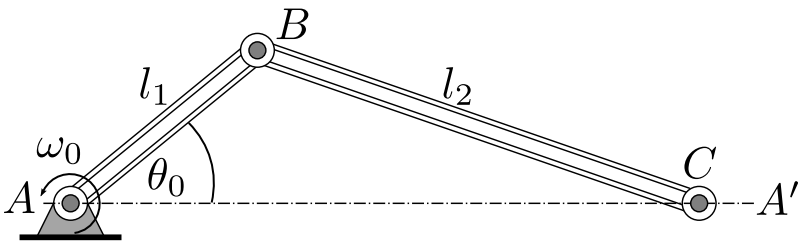
APLICACIONES COMPUTACIONALES [15063]
INGENIERÍA DE EJECUCIÓN MECÁNICA

Fecha de entrega: 02.05.2014

Sistemas de Ecuaciones: lineales y no lineales

Problema 1 (2 pts): Para el mecanismo biela-manivela de la figura escribir un programa que permita animar el movimiento a partir de un fichero de datos que contiene la posición en el tiempo de los extremos A, B y C.



El formato del fichero de datos es el siguiente:

Time	xA	yA	xB	yB	xC	yC
0.000e+00	0.000e+00	0.000e+00	7.071e-01	7.071e-01	2.578e+00	0.000e+00
1.000e-02	-6.140e-06	-1.080e-05	7.383e-01	6.745e-01	2.621e+00	-1.077e-05
⋮	⋮	⋮	⋮	⋮	⋮	⋮
5.00e+00	-2.954e-06	2.646e-06	-7.366e-01	6.763e-01	1.146e+00	-2.486e-07

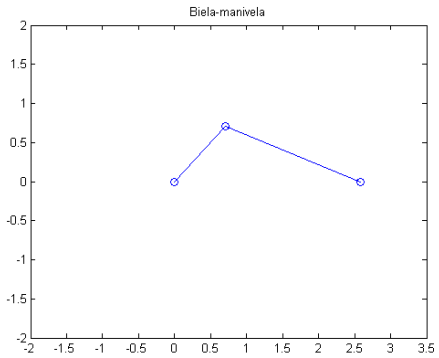
Cuadro 1: Tabla de resultados

Solución:

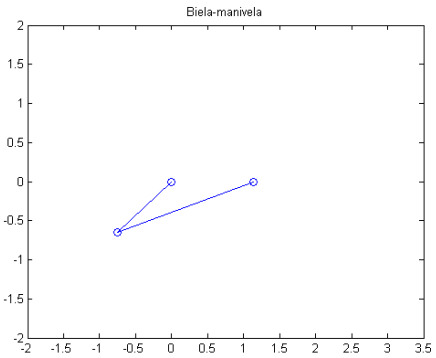
```
%
% Mecanismo Biela-manivela
%
% Leer el fichero con datos y guadar en una matriz "datos"
datos = load('Mecanismo.dat');

% Guardar vector temporales por columna
time=datos(:,1);
xA=datos(:,2);
yA=datos(:,3);
xB=datos(:,4);
yB=datos(:,5);
xC=datos(:,6);
yC=datos(:,7);

% Animar el movimiento mediante un ciclo "for"
for i = 1:length(time)

    % Posisiones "[x]" y "[y]" para un instante t(i)
    x = [xA(i),xB(i),xC(i)];
    y = [yA(i),yB(i),yC(i)];
```

```
    % Plot de las posciones del mecanimos para un
    % instante t(i)
    plot(x,y,'-o');

    % Titulo del grafico
    title('Biela-manivela')

    % Fijar limites del grafico
    axis([-2 3.5 -2 2])

    % Pause del grafico duarante 0.01 segundos
    % para dar el efecto de animacion
    pause(0.01)
end
%
%
```

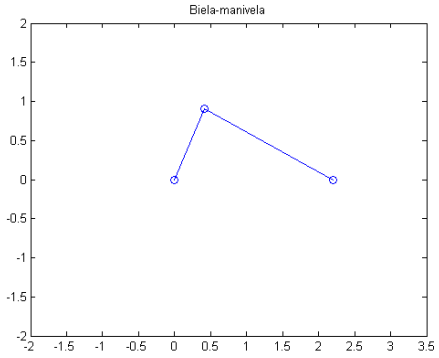
El resultado por pantalla en Matlab es el siguiente para algunos instante de tiempo:



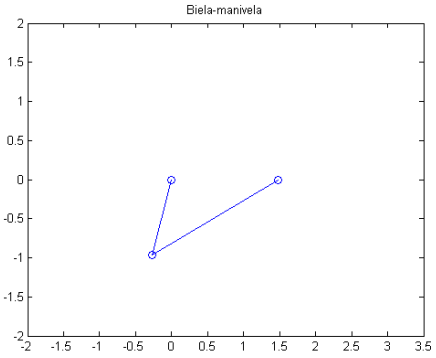
$t(1) = 0{,}0\,\mathrm{s}$



$t(50) = 0{,}5\,\mathrm{s},$

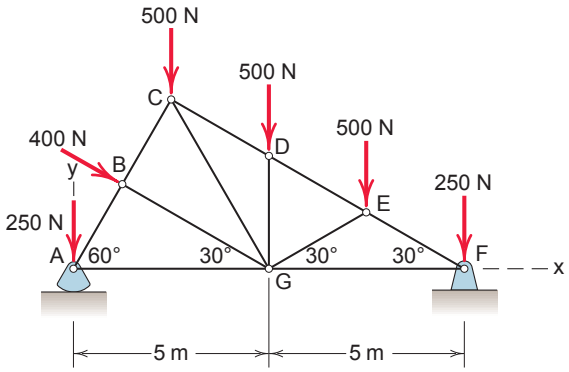


$t(100) = 0{,}1$



$t(150) = 1{,}5\,\mathrm{s}$

Problema 1 (2 pts): Para la estructura de la figura se pide obtener las fuerzas en cada una de las barras. Para ello escriba las ecuaciones de equilibrio de cada uno de los nodos, plantee el sistema de ecuaciones que resulta y resuelva mediante el método de descomposición [L][U] programado en Matlab.



Use su programa para resolver la estructura en diferentes estados de carga:

A) Todas las cargas aplicadas.

B) Sólo las cargas superiores a 450 N.

C) Sólo las cargas inferiores a 450 N.

Muestres sus resultados y los tiempos de cálculo para procesos independientes usando eliminación de Gauss y descomposición [L][U] (para reducir el tiempo de cálculo).

Solución:

La rutina a continuación compara los métodos de eliminación de Gauss con el método de descomposición [L][U], para un sistema de ecuaciones compuesto de una matriz A de dimensión n (con valores aleatorios), y m vectores del lado derecho b.

```
%
clc, clear all, close all
%
n = 800;          % dimension de la matriz
m = 50;           % numero vectores b
A = rand(n);      % matriz random A
b = rand(n,m);    % m vectores b almacenados en una matriz

%%
% Metodo de eliminacion de Gauss para resolver [A]{x}={b}
% Se resuelve para cada vector b(:,i)
%
tic
n = length(A);
[row_b col_b] = size(b);

for p = 1:col_b
```

```
     U = A;
     c = b(:,p);
    % 1) Eliminacion hacia adelante para obtener [U]
    for k = 1:n-1
        for i = k+1:n
            f = U(i,k) / U(k,k);
            U(i,:) = U(i,:) - f * U(k,:);
            c(i) = c(i) - f * c(k);
        end
    end
    % 2) Sustitucion hacia atras para resolver [U]{x}={c}
    %
    x(n) = c(n) / U(n,n);
    for i = n-1:-1:1
        sum = c(i);
        for j = i+1:n
            sum = sum - U(i,j) * x(j);
        end
        x(i) = sum / U(i,i);
    end
end
toc

%%
% Metodo descomposicion [L][U] resolver [A]{x}={b}
% Transformando [A]{x}={b} en  [L][U]{x}={}
%
%
tic
% Definir dimension de [L] y [U]
n = length(A);
U = A;
L = tril(ones(n));
[row_b col_b] = size(b);
%
% Eliminacion de Gauss hacia adelante
%
for k = 1:n-1
    for i = k+1:n
        f = U(i,k) / U(k,k);
        L(i,k) = f;
        U(i,:) = U(i,:) - f * U(k,:);
    end
end

% La idea es resolver en dos etapas:
% 1) Resolver [L]{y}={b}, mediante sustitucion
%    hacia adelante
%    Donde {y} es un vector incognita intermedio.
%
```

```
% 2) Resolver [U]{x}={y}, mediante sustitucion
%    hacia atras
%    Donde {x} es el vector incognita buscado.

for p = 1:col_b

    c = b(:,p);
    % 1) Sustitucion hacia adelante
    %    para resolver [L]{y}={b}
    %
    y(1) = c(1) / L(1,1);
    for i = 2:n
        sum = c(i);
        for j = 1:i-1
            sum = sum - L(i,j) * y(j);
        end
        y(i) = sum / L(i,i);
    end

    % 2) Sustitucion hacia atras
    %    para resolver [U]{x}={y}
    %
    x(n) = y(n) / U(n,n);
    for i = n-1:-1:1
        sum = y(i);
        for j = i+1:n
            sum = sum - U(i,j) * x(j);
        end
        x(i) = sum / U(i,i);
    end
end
toc
%
```

El resultado por pantalla en Matlab es el siguiente para algunos instante de tiempo:

```
Elapsed time is 193.447598 seconds.
Elapsed time is 38.655265 seconds.
```

La diferencia de tiempo entre un método y otro, es importante cuando el número de elementos en la matriz A es elevado; y también cuando existe una gran cantidad de vectores del lado derecho b a resolver.

Problema 1 (2 pts):

Solución: Procedimiento general en Matlab del método de Newton-Raphson para varias variables:

```
%
%
clc, clear all, close all
%
% Funciones F1 y F2
F1 = @(x,y) 5*x^2 + 3*x*y - 2;
F2 = @(x,y) x^2 + 7*y^2 + 3*x*y - 10;

% Jacobiano
F1dx = @(x,y) 10*x^2 + 3*y;
F1dy = @(x,y) 3*x;
F2dx = @(x,y) 2*x + 3*y;
F2dy = @(x,y) 14*y + 3*x;

err = 1;
v = [1;1];
k = 1;
while err > 1e-2

    x_n = v(1);
    y_n = v(2);

    F = [ F1(x_n,y_n)
          F2(x_n,y_n)];
    J = [ F1dx(x_n,y_n) F1dy(x_n,y_n)
          F2dx(x_n,y_n) F2dy(x_n,y_n)];
    v = v - inv(J)*F;

    err = norm(F);
    k = k+1;
end
disp(['Iteraciones: ' num2str(k)])
disp(['La funcion se hace cero para (x,y): ('...
      num2str(v(1)) ',' num2str(v(2)) ')'])%
```

El resultado por pantalla en Matlab es el siguiente para algunos instante de tiempo:

```
Iteraciones: 10
La funcion se hace cero para (x,y): (0.38272,1.1073)
```