# Anexo I

En este anexo se detalla el código programado para las simulaciones de las ruedas, separando en los casos antes establecidos.

## Caso 1

Listing 1: Caso 1

```
function Caso1(n,fin)

% Resuelve el caso en el que nos dan la carretera en la forma f(t)=(t,y(t))
% Los datos requeridos son nodos que queremos usar para
% obtener numéricamente la rueda y la distancia que queremos que recorra
% dicha rueda en la simulación

c1=1; %Contador para la generación de las imágenes
h=fin/n; x=0:h:fin;
theta(1)=-pi/2;  %Condición inicial%
matriz=zeros(n+1,3); matriz2=zeros(2,n+1);
for i=1:n          %En este bucle se realiza el método Runge-Kutta
  k1=-1/exacta1(x(i));
  k2=-1/exacta1(x(i)+h/2);
  k3=-1/exacta1(x(i)+h/2);
  k4=-1/exacta1(x(i)+h);
  theta(i+1)=theta(i)+h*(k1+2*k2+2*k3+k4)/6;
end

for i=1:n+1
  radio(i)=-exacta1(x(i)); %Calculamos la distancia de los
                           % puntos de la rueda al origen
end

for i=1:n+1
  matriz(i,1)=x(i);
  matriz(i,2)=theta(i);
  matriz(i,3)=radio(i);
end

z=-3:0.01:40;
for i=1:length(z)    %En este bucle calculamos los puntos de la carretera
  c(i)=exacta1(z(i));
end

for j=1:n+1 %Este bucle nos sirve para representar el avance en el eje x
            % la rueda a la vez que representa el giro de la misma

  for i=1:n+1 %En este bucle calculamos los puntos de la rueda
      matriz2(1,i)=matriz(i,3)*cos(matriz(i,2)-(theta(j)-theta(1)))+x(j);
      matriz2(2,i)=matriz(i,3)*sin(matriz(i,2)-(theta(j)-theta(1)));
  end
```

```
43
44    a=matriz2(1,:);    % Coordenadas x de los puntos de la rueda
45    b=matriz2(2,:);    % Coordenadas y de los puntos de la rueda
46
47    e1(j)=matriz2(1,1);
48    e2(j)=matriz2(2,1);
49
50    plot(a,b,z,c,e1,e2,'r-.',x(j),0,'bo', 'MarkerFaceColor','b','MarkerSize',5)
51
52    axis([-3,23, -10, 10.54])
53    saveas(1,sprintf('%d.png',c1));    %Guardamos las imágenes que
54                                       % se van generando
55    c1=c1+1;
56    clf;
57  end
```

Es necesario definir una función de Matlab en la que se especificará la carretera. Se ha denominado a esta función **exacta1**.

Listing 2: Función de la carretera

```
1  function f= exacta1(x)
2
3  f=2*(-1.887365-2*cos(x)/3+sin(x)-sin(2*x)/2);
4
5  return
```

# Caso 2

Listing 3: Caso 2

```
function Caso2(n,fin)

% Este es el caso en el que tenemos la rueda en polares, es decir,
% r=g(theta).
% Las datos requeridos son el número de nodos que queremos usar para
% la obtención numérica de la carretera y el ángulo que queremos que gire
% la rueda en la simulación

c1=1;   %Contador para la generación de las imágenes
h=fin/n; theta=-pi/2:h:fin-pi/2;
x(1)=0;   %Condición inicial
matriz=zeros(n+1,3); matriz3=zeros(2,n+1);

for i=1:n       %En este bucle se realiza el método Runge-Kutta
  k1=exacta2(theta(i));
  k2=exacta2(theta(i)+h/2);
  k3=exacta2(theta(i)+h/2);
  k4=exacta2(theta(i)+h);
  x(i+1)=x(i)+h*(k1+2*k2+2*k3+k4)/6;
end


for i=1:n+1
  matriz(i,1)=x(i);
  matriz(i,2)=theta(i);
  matriz(i,3)=-exacta2(theta(i));
end

matriz2=matriz';
a=matriz2(1,:);    % Coordenadas x de los puntos de la carretera
b=matriz2(3,:);    % Coordenadas y de los puntos de la carretera

for j=1:n+1  % Este bucle nos sirve para representar el avance en el eje x
             % la rueda a la vez que representa el giro de la misma

  for i=1:n+1    % Calculamos los puntos de la rueda
      matriz3(1,i)=exacta2(theta(i))*cos(theta(i)-(theta(j)-theta(1)))+x(j);
      matriz3(2,i)=exacta2(theta(i))*sin(theta(i)-(theta(j)-theta(1)));
  end
  c=matriz3(1,:);    % Coordenadas x de los puntos de la rueda
  d=matriz3(2,:);    % Coordenadas y de los puntos de la rueda

  e1(j)=matriz3(1,1);
  e2(j)=matriz3(2,1);

  plot(a,b,c,d,e1,e2,'r-.',x(j),0,'bo', 'MarkerFaceColor','b','MarkerSize',5)
  axis([-10,55,-25,25])
  saveas(1,sprintf('carretera-%d.png',c1)); %Guardamos las imágenes que
                                            % se van generando
  clf;
  c1=c1+1;
end
```

Debe definirse la función de Matlab que contendrá la forma polar de la rueda. En este caso se le ha llamado **exacta2**

Listing 4: Función polar de la rueda

```
function r= exacta2(theta)

r=4*sqrt(5-4*sin(theta)*sin(theta));

return
```

# Caso 3

Listing 5: Caso 3

```
function Caso3(n,fin)

% Resuelve el caso en el que nos dan la carretera en la forma
% f(t)=(x(t),y(t))
% Los datos requeridos son nodos que queremos usar para
% obtener numéricamente la rueda y el tiempo final en la simulación

c1=1; %Contador para la generación de las imágenes
h=fin/n; t=0:h:fin; x=exacta3x(t);
theta(1)=-pi/2;  %Condición inicial%
matriz=zeros(n+1,3); matriz2=zeros(2,n+1);

for i=1:n          %En este bucle se realiza el método Runge-Kutta
  k1=integral3(t(i));
  k2=integral3(t(i)+h/2);
  k3=integral3(t(i)+h/2);
  k4=integral3(t(i)+h);
  theta(i+1)=theta(i)+h*(k1+2*k2+2*k3+k4)/6;
end

for i=1:n+1
  radio(i)=-exacta3y(t(i)); %Calculamos la distancia de los
                            % puntos de la rueda al origen
end

for i=1:n+1
  matriz(i,1)=x(i);
  matriz(i,2)=theta(i);
  matriz(i,3)=radio(i);
end

t2=-x(n+1):0.005:x(n+1);
for i=1:length(t2)   %En este bucle calculamos los puntos de la carretera
 z(i)=exacta3x(t2(i));
 c(i)=exacta3y(t2(i));
end

for j=1:n+1 %Este bucle nos sirve para representar el avance en el eje x
            % la rueda a la vez que representa el giro de la misma

  for i=1:n+1 %En este bucle calculamos los puntos de la rueda
      matriz2(1,i)=matriz(i,3)*cos(matriz(i,2)-(theta(j)-theta(1)))+x(j);
      matriz2(2,i)=matriz(i,3)*sin(matriz(i,2)-(theta(j)-theta(1)));
  end

  a=matriz2(1,:);   % Coordenadas x de los puntos de la rueda
  b=matriz2(2,:);   % Coordenadas y de los puntos de la rueda

  e1(j)=matriz2(1,1);
  e2(j)=matriz2(2,1);

  plot(a,b,z,c,e1,e2,'r-.',x(j),0,'bo',
                              'MarkerFaceColor','b','MarkerSize',5)
  axis([-5,25, -11.8, 11.8])
  saveas(1,sprintf('%d.png',c1));    %Guardamos las imágenes que
                                     % se van generando
  c1=c1+1;
  clf;
end
```

En este caso deben definirse 3 fuciones. Las que contienen las ecuaciones paramétricas de la carretera (que se han denominado **exacta3x** y **exacta3y** respectivamente) y otra en la que estará la función $\frac{-x'(t)}{y(t)}$, la cual se ha nombrado **integral3**

Listing 6: Función parámetrica x de la rueda

```
function f=exacta3x(t)

f=t+sin(t);

return
```

Listing 7: Función parámetrica y de la rueda

```
function g=exacta3y(t)

g=-1-2/3+cos(t);

return
```

Listing 8: Función que se usa en el método Runge-Kutta

```
function f=integral3(t)

f=(1+cos(t))/(1+2/3-cos(t));

return
```

# Caso 4

Listing 9: Caso 4

```
function Caso4(n,fin)

% Resuelve el caso en el que nos dan la rueda en la forma
% f(t)=(x(t),y(t))
% Los datos requeridos son nodos que queremos usar para
% obtener numéricamente la rueda y el valor final del parámetro t

c1=1; %Contador para la generación de las imágenes
h=fin/n; t=0:h:fin;
x(1)=0;  %Condición inicial%
theta(1)=-pi/2;

for i=1:n          % En este bucle se realizan los dos métodos Runge-Kutta
                   % que son necesarios en este caso
  k1=integral4c(t(i));
  k2=integral4c(t(i)+h/2);
  k3=integral4c(t(i)+h/2);
  k4=integral4c(t(i)+h);
  x(i+1)=x(i)+h*(k1+2*k2+2*k3+k4)/6;
  l1=integral4r(t(i));
  l2=integral4r(t(i)+h/2);
  l3=integral4r(t(i)+h/2);
  l4=integral4r(t(i)+h);
  theta(i+1)=theta(i)+h*(l1+2*l2+2*l3+l4)/6;

end

for i=1:n+1
    % Calculamos la distancia de los
    % puntos de la rueda al origen
    radio(i)=sqrt(exacta4x(t(i))^2+exacta4y(t(i))^2);
end

for j=1:n+1 %Este bucle nos sirve para representar el avance en el eje x
            % la rueda a la vez que representa el giro de la misma

  for i=1:n+1 %En este bucle calculamos los puntos de la rueda
      a(i)=radio(i)*cos(theta(i)-(theta(j)-theta(1)))+x(j);
      b(i)=radio(i)*sin(theta(i)-(theta(j)-theta(1)));
  end


  plot(a,b,x,-radio,'c-.',x(j),0,'bo','MarkerFaceColor','b','MarkerSize',5)
  axis([-3,12, -6, 6])
  saveas(1,sprintf('%d.png',c1));    %Guardamos las imágenes que
                                     % se van generando
  c1=c1+1;
  clf;
end
```

Este último caso requiere la programación de 4 funciones de Matlab extra. Dos de ellas serán las ecuaciones paramétricas de la rueda (denominadas **exacta4x** y **exacta4y**) y las otras dos contienen las funciones $\frac{-x'(t)}{y(t)}$ donde $x$ e $y$ son los parámetros de la carretera o de la rueda según la función (denominadas **integral4c** y **integral4r** respectivamente)

Listing 10: Función parámetrica x de la rueda

```
function f=exacta4x(t)

f=sin(t)-sin(2*t)/2;

return
```

Listing 11: Función parámetrica y de la rueda

```
function g=exacta4y(t)

g=-cos(t);

return
```

Listing 12: Función que se usa en el método Runge-Kutta de la carretera

```
function f=integral4c(t)

f=((exacta4x(t))*(sin(t))-(cos(t)-cos(2*t))*...
                    (exacta4y(t)))/sqrt(exacta4x(t)^2+exacta4y(t)^2);

return
```

Listing 13: Función que se usa en el método Runge-Kutta de la rueda

```
function f=integral4r(t)

f=((exacta4x(t))*(sin(t))-(cos(t)-cos(2*t))*...
                    (exacta4y(t)))/(exacta4x(t)^2+exacta4y(t)^2);

return
```

# Anexo II

En este anexo se detalla el código programado para la simulación del botafumeiro. El resultado del mismo es una animación que se reproduce en una ventana de Matlab al compilar y ejecutar el código.

Listing 14: Programa principal

```
function botafumeiro3(tirones,theta0,v0,n,L,L1)

% Los datos requeridos para realizar la simulación son el número máximo de
% tirones que queremos simular, el ángulo inicial del péndulo, la velocidad
% inicial del mismo, el número de nodos que queremos usar para la
% resolución numérica, la longitud máxima de la cuerda y la longitud mínima
% de la misma.

theta(1)=theta0; v(1)=v0;  %Condiciones iniciales

componentes=0; % El número de puntos de la trayectoria del péndulo
               % se inicializa a 0
tiemposubida=0.25; % Se fija el tiempo que dura el proceso de
                   % acortamiento de la cuerda
h=tiemposubida/n; t=0:h:tiemposubida; t2=0:h:tiemposubida; for
j=1:tirones

    % PRIMERA ETAPA, PÉNDULO EN LONGITUD MÁXIMA
    i1=0;
    z=1;
    while (z==1 || theta(i1)*theta(i1+1)>0) % Se aplica el método Runge-Kutta
        z=0;
        i1=i1+1;
        k1=h*v(i1);
        l1=h*f(t(1),theta(i1),v(i1),L);
        k2=h*(v(i1)+l1/2);
        l2=h*f(t(1)+h/2,theta(i1)+k1/2,v(i1)+l1/2,L);
        k3=h*(v(i1)+l2/2);
        l3=h*f(t(1)+h/2,theta(i1)+k2/2,v(i1)+l2/2,L);
        k4=h*(v(i1)+l3);
        l4=h*f(t(1)+h,theta(i1)+k3,v(i1)+l3,L);

        theta(i1+1)=theta(i1)+(k1+2*k2+2*k3+k4)/6;
        v(i1+1)=v(i1)+(l1+2*l2+2*l3+l4)/6;
    end

    i1=i1-1; % Como el último punto calculado es cuando ya
             % se ha cruzado el eje vertical se usará el punto anterior

```

```
40      for k=1:i1
41         x(k+componentes)=L*cos(theta(k)-pi/2);
42         y(k+componentes)=L*sin(theta(k)-pi/2);
43         hold off
44         plot(x(k+componentes),y(k+componentes),'o', ...
               'MarkerFaceColor','b','MarkerSize',10);
45         hold on
46         plot([0;x(k+componentes)],[0;y(k+componentes)]);
47         plot(x,y,'red');
48
49         title(['Tirones : ' num2str(j-1)]);
50         axis([-L*1.264, L*1.264 ,-L, 0]);
51         set(gca,'dataAspectRatio',[1 1 1])
52         pause(0.0001);
53      end
54
55      componentes=componentes+i1; % Actualizamos el número de puntos de la
56                                  % trayectoria recorrida
57
58      if max(abs(theta))>pi/2 % Si se superan los 90° paramos la animación
59          break
60      end
61
62      % SEGUNDA ETAPA, PÉNDULO SUBIENDO
63
64      theta2(1)=theta(i1);  % Las condiciones iniciales de esta etapa son las
65      v2(1)=v(i1);          % finales de la anterior
66
67      for j1=1:n+1        % Se aplica el método Runge-Kutta
68          k1=h*v2(j1);
69          l1=h*f2(t2(j1),theta2(j1),v2(j1),L,tiemposubida);
70          k2=h*(v2(j1)+l1/2);
71          l2=h*f2(t2(j1)+h/2,theta2(j1)+k1/2,v2(j1)+l1/2,L,tiemposubida);
72          k3=h*(v2(j1)+l2/2);
73          l3=h*f2(t2(j1)+h/2,theta2(j1)+k2/2,v2(j1)+l2/2,L,tiemposubida);
74          k4=h*(v2(j1)+l3);
75          l4=h*f2(t2(j1)+h,theta(j1)+k3,v(j1)+l3,L,tiemposubida);
76
77          theta2(j1+1)=theta2(j1)+(k1+2*k2+2*k3+k4)/6;
78          v2(j1+1)=v2(j1)+(l1+2*l2+2*l3+l4)/6;
79      end
80
81      for k=1:j1
82          x(k+componentes)=longitud(t(k),tiemposubida,L)*cos(theta2(k)-pi/2);
83          y(k+componentes)=longitud(t(k),tiemposubida,L)*sin(theta2(k)-pi/2);
84          hold off
85          plot(x(k+componentes),y(k+componentes),'o', ...
                'MarkerFaceColor','b','MarkerSize',10);
86          hold on
87          plot([0;x(k+componentes)],[0;y(k+componentes)]);
88          plot(x,y,'red');
89
90         title(['Tirones : ' num2str(j-1)]);
91         axis([-L*1.264, L*1.264 ,-L, 0]);
92         set(gca,'dataAspectRatio',[1 1 1])
93         pause(0.0001);
94      end
95
96      componentes=componentes+j1; % Actualizamos el número de puntos de la
97                                  % trayectoria recorrida
98
```

```
99      % TERCERA ETAPA, PÉNDULO EN LONGITUD MÍNIMA
100
101     theta3(1)=theta2(j1);   % Las condiciones iniciales de la nueva etapa son
102     v3(1)=v2(j1);           % las finales de la anterior
103
104     i2=0;
105     z=1;
106     while (z==1 || v3(i2)*v3(i2+1)>0) % Se aplica el método Runge-Kutta
107         z=0;
108         i2=i2+1;
109         k1=h*v3(i2);
110         l1=h*f(t(2),theta3(i2),v3(i2),L1);
111         k2=h*(v3(i2)+l1/2);
112         l2=h*f(t(2)+h/2,theta3(i2)+k1/2,v3(i2)+l1/2,L1);
113         k3=h*(v3(i2)+l2/2);
114         l3=h*f(t(2)+h/2,theta3(i2)+k2/2,v3(i2)+l2/2,L1);
115         k4=h*(v3(i2)+l3);
116         l4=h*f(t(2)+h,theta3(i2)+k3,v3(i2)+l3,L1);
117
118         theta3(i2+1)=theta3(i2)+(k1+2*k2+2*k3+k4)/6;
119         v3(i2+1)=v3(i2)+(l1+2*l2+2*l3+l4)/6;
120     end
121
122     i2=i2-1; % Como el último punto calculado es cuando la velocidad ha
123              % cambiado de signo, se usará el punto anterior
124
125     for k=1:i2
126         x(k+componentes)=L1*cos(theta3(k)-pi/2);
127         y(k+componentes)=L1*sin(theta3(k)-pi/2);
128         hold off
129         plot(x(k+componentes),y(k+componentes),'o', ...
                'MarkerFaceColor','b','MarkerSize',10);
130         hold on
131         plot([0;x(k+componentes)],[0;y(k+componentes)]);
132         plot(x,y,'red');
133
134        title(['Tirones : ' num2str(j)]);
135        axis([-L*1.264, L*1.264 ,-L, 0]);
136        set(gca,'dataAspectRatio',[1 1 1])
137        pause(0.0001);
138     end
139
140     componentes=componentes+i2; % Actualizamos el número de puntos de la
141                                 % trayectoria recorrida
142
143     if max(abs(theta3))>pi/2 % Si se superan los 90° paramos la animación
144         break
145     end
146
147     theta(1)=theta3(i2); % Las condiciones iniciales de la siguiente etapa
148     v(1)=0;              % son las finales de la última calculada
149
150 end
151
152 plot(x,y) % La trayectoria se va dibujando en rojo mientras se va recorriendo
153           % y cuando acaba la animación se queda marcada en azul
```

Se usan también 3 funciones aparte del programa principal:

- La función que se usa durante la resolución del problema de valor inicial mediante el método Runge-Kutta en las partes donde la longitud es fija se denota por **f**.
- A la función que devuelve la longitud de la cuerda en función del tiempo en los tramos donde dicha longitud varía se le ha llamado **longitud**
- La función que se usa durante la resolución del problema de valor inicial mediante el método Runge-Kutta en las partes donde la longitud varía se denota por **f2**.

Listing 15: Función usada en el método Runge-Kutta cuando la longitud es fija

```
function f=f(t,theta,v,L)

f=-9.8*sin(theta)/L;

return
```

Listing 16: Función que determina la longitud de la cuerda cuando esta varía

```
function f=f2(t,theta,v,L,tsubida)

f=(-2*v*((exp((-((t-tsubida)/0.2)^2)/2))*((t-tsubida)/0.2^2)*2.9/...
    (exp(0)-exp((-((-tsubida)/0.2)^2)/2)))-9.8*sin(theta))/longitud(t,tsubida,L);

return
```

Listing 17: Función usada en el método Runge-Kutta cuando la longitud varía

```
function f=longitud(t,tsubida,L)

f=L-(exp((-((t-tsubida)/0.2)^2)/2)-exp((-((-tsubida)/0.2)^2)/2))*2.9/...
    (exp((0/0.2)^2/2)-exp((-((0-tsubida)/0.2)^2)/2));

return
```