Evaluación Continua 3

Lucas Saurin

13 de junio de 2023

1. Consigna

Se desea conocer la trayectoria de una partícula que se mueve en el plano, dada por la curva $(x(t), y(t))$. Para ello se cuenta con dos sensores, que determinan la posición de la partícula: uno mide la posición en el eje x cada 2 segundos, y otro en el eje y cada 1 segundo. En el inicio de las mediciones ($t = 0$) se sabe que la partícula se encuentra a 2cm del origen en la dirección x y se mueve a una velocidad $\frac{\pi}{2} cm/s$ en la dirección y, y después de 6 segundos llega al origen a la misma velocidad inicial, pero en dirección negativa de y. Las mediciones de posición de los sensores se muestran en la siguiente tabla:

t[s]	Sensor x[cm]	Sensor y[cm]
0	2.0	0.0
1	-	1.0
2	1.5	0.0
3	-	-1.0
4	0.5	0.0
5	-	1.0
6	0.0	0.0

Tabla 1: Valores sensados.

(a) Realice interpolaciones por spline cúbicos sujetos para determinar expresiones de $x(t)$ y de $y(t)$ utilizando los datos de la tabla, y las velocidades inicial y final que describe el problema.

(b) Grafique la trayectoria de la partícula y determine la posición y el vector velocidad a los 3 segundos.

(c) Recuerde que la longitud de la trayectoria de la partícula durante lo T primeros segundos está dada por $\int_0^T \sqrt{x_1'(t)^2 + x_2'(t)^2} dt$. Estimar la distancia recorrida por la partícula durante el proceso. Dar el resultado con 6 cifras exactas.

2. Introducción

En este informe se utilizarán unos pocos datos de posición dados por sensores para aproximar no solo la trayectoria de una partícula, sino también para calcular la distancia recorrida por la misma. Para ello se hará uso de métodos numéricos de interpolación e integración, para cuyos algoritmos se utilizará el software Octave.

3. Interpolación

El primer paso (y consigna) que realizaremos es la aproximación de la trayectoria, a partir de la cual haremos el resto de cálculos.

Como especifica el problema, utilizaremos la técnica de Splines Cúbicos. Estos tienen la ventaja de generar curvas con continuidad C^2, es decir que, además de ser continuas, su derivada primera y segunda también lo son. En nuestro caso, al conocer las velocidades iniciales y finales (condiciones de frontera) de la partícula usaremos Splines Cúbicos Sujetos.

A partir de esta interpolación obtendremos las funciones $S_x(t)$ y $S_y(t)$ definidas por partes, donde cada intervalo estará dado por los puntos interpolantes y que cumplen con las siguientes condiciones:

a. $S_x(t)$ es un polinomio cúbico, denotado $S_{x_j}(x)$, en el subintervalo $[t_j, t_{j+1}]$ para cada $j = 0, 1, ..., n-1$.

b. $S_x(t_j) = x(t_j)$ para cada $j = 0, 1, ..., n$.

c. $S_{x_{j+1}}(t_{j+1}) = S_{x_j}(t_{j+1})$ para cada $j = 0, 1, ..., n-1$.

d. $S'_{x_{j+1}}(t_{j+1}) = S'_{x_j}(t_{j+1})$ para cada $j = 1, 2, ..., n-1$.

e. $S''_{x_{j+1}}(t_{j+1}) = S''_{x_j}(t_{j+1})$ para cada $j = 1, 2, ..., n-1$.

f. $S'_x(t_0) = x'(t_0)$ y $S'_x(t_n) = x'(t_n)$.

Lo mismo para S_y.

Para los puntos $x_0, x_1, ..., x_n$ entonces tendremos:

$$S_x(t) = \begin{cases} a_0 + b_0 t + c_0 t^2 + d_0 t^3 & t_0 \leq t \leq t_1 \\ a_1 + b_1(t - t_1) + c_1(t - t_1)^2 + d_1(t_1)^3 & t_1 < t \leq t_2 \\ . & \\ . & \\ . & \\ a_{n-1} + b_{n-1}?(t - t_{n-1}) + c_{n-1}(t_{n-1})^2 + d_{n-1}(t_{n-1})^3 & t_{n-1} < t \leq t_n \end{cases}$$

Donde los coeficientes calculados a partir de las condiciones dadas anteriormente son mostrados en las tablas

Vemos que además es fácilmente calculable su derivada, la cual usaremos para calcular la distancia más adelante.

Intervalo	**ax**	**bx**	**cy**	**dx**
[0,2]	2	0	-0.15	0.0125
[2,4]	1.5	-0.45	-0.075	0.025
[4,6]	0.5	-0.45	0.075	0.0125

Tabla 2: Coeficientes de la Spline interpolante de x

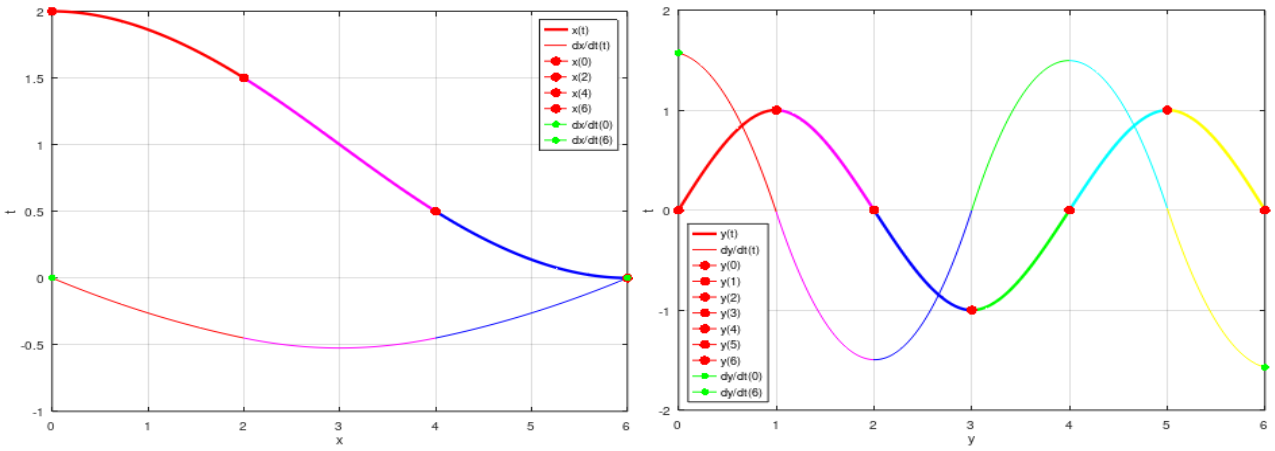
Intervalo	**ay**	**by**	**cy**	**dy**
[0,1]	0	1.57079	-0.12271	-0.44808
[1,2]	1	-0.01887	-1.46696	0.48584
[2,3]	0	-1.49528	-0.00943	0.50471
[3,4]	-1	0	1.50471	-0.50471
[4,5]	0	1.49528	-0.00943	-0.48584
[5,6]	1	0.01887	-1.46696	0.44808

Tabla 3: Coeficientes de la Spline interpolante de y

A partir de estos datos, utilizando la función *funcion_spline*, obtendremos la función correspondiente al Spline Cúbico tanto de $x(t)$ e $y(t)$ como de sus primeras derivadas, los cuales nos servirán para graficar la trayectoria de la partícula y evaluar su posición y velocidad a los 3 segundos.

$$P(3) = (1; -0{,}9999999999999999), \qquad v(3) = (-0{,}525; 2{,}220446049250313 * 10^{-16})$$

Analíticamente, la posición nos debería dar exactamente $P(3) = (1; 0)$ y $v(3) = (-0{,}525; 0)$. Esto es debido al error de redondeo por usar el numero *pi*.



(a) (b)

Figura 1: Gráficas de las componentes x (a) e y (b) con cada tramo diferenciado por color.

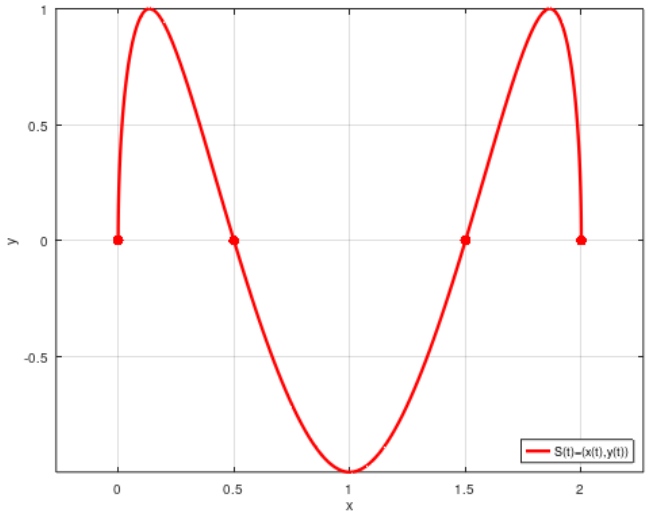


Figura 2: Trayectoria seguida por la partícula.

4. Integración

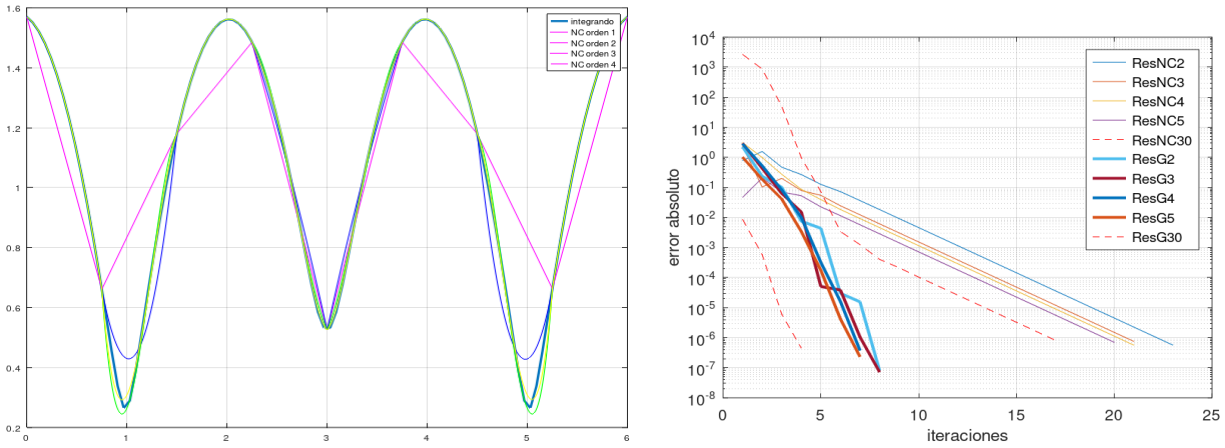
Para calcular la distancia recorrida por la partícula en los 6*segundos* utilizaremos el método de fórmulas de Newton-Cotes y el método de Cuadraturas de Gauss con polinomios de Legendre.

El primero consiste en tomar $n+1$ nodos $x_i = x_0 + ih$ para $i = 0, 1, ..., n$ donde $x_0 = a$, $x_n = b$ y $h = (b-a)/n$ equidistantes dentro del intervalo y construir un polinomio de Lagrange a partir de estos nodos, para luego aproximar la integral a partir de la integral de este polinomio en el intervalo.

El método de Cuadraturas de Gauss con Polinomios se basa en polinomios ortogonales, en este caso específico los de Legendre, los cuales están definidos en el dominio $[-1, 1]$. El mismo encuentra tanto los puntos como los pesos para la evaluación en esos puntos que generen la mejor aproximación. Para los primero, utiliza las raíces de los polinomios ortogonales, y los pesos son determinados a partir de propiedades de los mismos.

Si dividimos el intervalo en L subintervalos y aplicamos estos métodos en cada uno de ellos, podemos obtener una mejor aproximación. Veremos cuántos subintervalos se requieren para lograr 6 dígitos exactos en la integral, empezando desde $L = 1$ intervalos y duplicando en cada iteración, hasta que la diferencia absoluta con la iteración anterior sea menor a $1 * 10^{-6}$

A partir de estos métodos estimamos una distancia recorrida por la partícula (con seis dígitos exactos)



(a) (b)

Figura 3: Gráficas de los polinomios interpolantes de Newton-Cotes en 8 tramos (a) y la curva del error desde $L = 2$ (b).

de $\int_0^T \sqrt{x_1'(t)^2 + x_2'(t)^2}dt = \mathbf{6.51499}$

Método	**tiempo[s]**	**L**
NC n=2	45.34706783294678	8388608
NC n=3	16.92670297622681	2097152
NC n=4	22.4292459487915	2097152
NC n=5	14.13392615318298	1048576
NC n=30	10.41877818107605	131072
Gauss-L n=2	2.973474025726318	256
Gauss-L n=3	2.928060054779053	256
Gauss-L n=4	1.54735803604126	128
Gauss-L n=5	1.502454042434692	128
Gauss-L n=30	0.2123258113861084	16

Tabla 4: Resultados de distintos métodos de integración numérica.

5. Conclusiones

Se comprobó a través de estos pasos que la interpolación por Splines Cúbicos genera no solo una curva considerablemente suave a partir de pocos datos de entrada, sino también que a partir de esta podemos aproximar el comportamiento de las componentes de la posición por separado, y que la curva que generarán ambas también será suave.

Además, vemos que por este método también obtenemos una expresión para las derivadas de las componente, a partir de las cuales podemos evaluar la velocidad instantánea de la partícula, además de usarla para estimar las distancia recorrida por la misma.

Respecto a la estimación de la distancia, la cual requiere el cálculo de una integral, observamos una diferencia considerable en el tiempo de cómputo requerido por cada método para llegar a la precisión requerida. En el caso de las fórmulas de Newton-Cotes, se logra una mejora considerable al aumentar el grado de los polinomios interpolantes, aunque al considerablemente la cantidad de nodos no necesariamente mejoraremos el método de forma considerable, debido a la cantidad de evaluaciones requeridas y a la naturaleza oscilante de los polinomios, además de disminuir la estabilidad (esto lo podemos ver en la curva del residuo). En el caso de el método de Cuadratura de Gauss este parámetro no genera gran diferencia en el tiempo de cómputo, debido a que aumentan la cantidad de evaluaciones de la función.

6. Scripts Utilizados

6.1. Script Cliente

```
clc; clear all; close all;
format long g;

% - Mediciones
tx = [0 2 4 6];          %tiempo de la medicion
posx = [2.0 1.5 0.5 0.0]; %valor en x
ty = [0 1 2 3 4 5 6];
posy = [0.0 1.0 0.0 -1.0 0.0 1.0 0.0];

a=0;
b=6;

dy0 = pi/2;
dyn = -pi/2;

% - a) Interpolacion
disp("==========================================")
[x,dx] = funcion_spline(tx,posx,0,0);    %velocidad inicial y final nulas en
    x
[a_x,b_x,c_x,d_x] = cubic_spline_clamped(tx,posx,0,0)
disp("==========================================")
[y,dy] = funcion_spline(ty,posy,pi/2,-pi/2);
[a_y,b_y,c_y,d_y] = cubic_spline_clamped(ty,posy,pi/2,-pi/2)
disp("==========================================")

% - b) Grafica
  % x(t) y dx(t)
figure(1);
t = linspace(0,2,50);
plot(t,x(t),'-r','linewidth',2);
hold on;
plot(t,dx(t),'-r');
t = linspace(2,4,50);
plot(t,x(t),'-m','linewidth',2);
plot(t,dx(t),'-m');
t = linspace(4,6,50);
plot(t,x(t),'-b','linewidth',2);
plot(t,dx(t),'-b');

grid on;
legend("x(t)","dx/dt(t)");
xlabel("x");
ylabel("t");

  % puntos (t,posx)
for i=1:length(tx)
  plot(tx(i),posx(i), 'color', 'r', 'markersize', 20, 'displayname', strcat(
      "x(", num2str(tx(i)),")"));
endfor
plot(0,0,'color','g','markersize',15,'displayname','dx/dt(0)');
plot(6,0,'color','g','markersize',15,'displayname','dx/dt(6)');
```

```
figure(2);
  % y(t) y dy(t)
t = linspace(0,1,50);
plot(t,y(t),'-r', 'linewidth',2);
hold on;
plot(t,dy(t),'-r');
t = linspace(1,2,50);
plot(t,y(t),'-m', 'linewidth',2);
plot(t,dy(t),'-m');
t = linspace(2,3,50);
plot(t,y(t),'-b', 'linewidth',2);
plot(t,dy(t),'-b');
t = linspace(3,4,50);
plot(t,y(t),'-g', 'linewidth',2);
plot(t,dy(t),'-g');
t = linspace(4,5,50);
plot(t,y(t),'-c', 'linewidth',2);
plot(t,dy(t),'-c');
t = linspace(5,6,50);
plot(t,y(t),'-y', 'linewidth',2);
plot(t,dy(t),'-y');

grid on;
h = legend("y(t)","dy/dt(t)");
legend(h,"location","southwest");
xlabel("y");
ylabel("t");

  % puntos (t,posy)
for i=1:length(ty)
  plot(ty(i),posy(i), 'color', 'r', 'markersize', 20, 'displayname', strcat(
      "y(", num2str(ty(i)),")"));
endfor
plot(0,dy0,'color','g','markersize',15,'displayname','dy/dt(0)');
plot(6,dyn,'color','g','markersize',15,'displayname','dy/dt(6)');

  % (x(t),y(t))
figure(3);
h=ezplot(x,y,[0 6]);
set(h,'color','r','linewidth',2);
hold on;
  % puntos (posx,posy)
for i=1:length(tx)
  plot(x(tx(i)),y(tx(i)), 'color', 'r', 'markersize', 20, 'displayname',
      strcat("S(", num2str(tx(i)),")"));
endfor

grid on;
h=legend("S(t)=(x(t),y(t))");
legend(h,"location","southeast");

% b) Evaluacion
% S(3)
pos3 = [x(3),y(3)]
```

```
% dS(3)
vel3 = [dx(3),dy(3)]

% c) Integraci n
integrando = @(t) sqrt(dx(t).^2 + dy(t).^2);

  % Grafica del integrando con los polinomios interpolantes por subintervalo
figure(4);
t = linspace(0.0001,6,100);
plot(t,integrando(t),'linewidth',2);
hold on;

h = (b-a)/8;
    % n=2 (Trapecio)
for i=1:8
  p = linspace(a+(i-1)*h,a+i*h,2);
  fp = integrando(p);
  P = PolyLag(p,fp);
  t = linspace(a+(i-1)*h,a+i*h,50);
  plot(t,polyval(P,t),'-m');
endfor
    % n=3 (Simpson)
for i=1:8
  p = linspace(a+(i-1)*h,a+i*h,3);
  fp = integrando(p);
  P = PolyLag(p,fp);
  t = linspace(a+(i-1)*h,a+i*h,50);
  plot(t,polyval(P,t),'-b');
endfor
    % n=4
for i=1:8
  p = linspace(a+(i-1)*h,a+i*h,4);
  fp = integrando(p);
  P = PolyLag(p,fp);
  t = linspace(a+(i-1)*h,a+i*h,50);
  plot(t,polyval(P,t),'-g');
endfor
    % n=5
for i=1:8
  p = linspace(a+(i-1)*h,a+i*h,5);
  fp = integrando(p);
  P = PolyLag(p,fp);
  t = linspace(a+(i-1)*h,a+i*h,50);
  plot(t,polyval(P,t),'-y');
endfor
legend('integrando','NC orden 1','NC orden 2','NC orden 3','NC orden 4');

% Calculo de la integral
tolerancia = 1e-6;
maxit = 30;

[INC2,itNC2,rNC2,tNC2,LNC2] = intNC(integrando,0,6,2,tolerancia,maxit);
INC2
itNC2
tNC2
```

```
LNC2

[INC3,itNC3,rNC3,tNC3,LNC3] = intNC(integrando,0,6,3,tolerancia,maxit);
INC3
itNC3
tNC3
LNC3

[INC4,itNC4,rNC4,tNC4,LNC4] = intNC(integrando,0,6,4,tolerancia,maxit);
INC4
itNC4
tNC4
LNC4

[INC5,itNC5,rNC5,tNC5,LNC5] = intNC(integrando,0,6,5,tolerancia,maxit);
INC5
itNC5
tNC5
LNC5

[INC30,itNC30,rNC30,tNC30,LNC30] = intNC(integrando,0,6,30,tolerancia,maxit)
    ;
INC30
itNC30
tNC30
LNC30
disp("=====================================");

[IG2,itG2,rG2,tG2,LG2] = intGauss(integrando,0,6,2,tolerancia,maxit);
IG2
itG2
tG2
LG2

[IG3,itG3,rG3,tG3,LG3] = intGauss(integrando,0,6,3,tolerancia,maxit);
IG3
itG3
tG3
LG3

[IG4,itG4,rG4,tG4,LG4] = intGauss(integrando,0,6,4,tolerancia,maxit);
IG4
itG4
tG4
LG4

[IG5,itG5,rG5,tG5,LG5] = intGauss(integrando,0,6,5,tolerancia,maxit);
IG5
itG5
tG5
LG5

[IG30,itG30,rG30,tG30,LG30] = intGauss(integrando,0,6,30,tolerancia,maxit);
IG30
itG30
```

```
tG30
LG30

% Graficar residuos
figure(5);
semilogy(rNC2);
hold on;
semilogy(rNC3);
semilogy(rNC4);
semilogy(rNC5);
semilogy(rNC30,'color','r','linestyle','--');
semilogy(rG2,'linewidth',2);
semilogy(rG3,'linewidth',2);
semilogy(rG4,'linewidth',2);
semilogy(rG5,'linewidth',2);
semilogy(rG30,'color','r','linestyle','--');
legend('ResNC2', 'ResNC3', 'ResNC4', 'ResNC5', 'ResNC30', 'ResG2', 'ResG3',
    'ResG4', 'ResG5', 'ResG30');
xlabel('iteraciones');
ylabel('error_absoluto');
```

6.2. cubic_spline_clamped

```
% x: puntos xi, i=1,2,...,n
% y: puntos yi correspondiente a f(xi), i=1,2,...,n
% df1 y dfn: valor de la derivada de f en x0 y xn
function [ai,bi,ci,di] = cubic_spline_clamped(x,y,df1,dfn)
    n = length(x);

    ai = y;

    h(1:n-1) = x(2:n) - x(1:n-1);

    % - Calculamos los terminos independientes
    b(1:n) = 0;
    b(1) = 3*( (y(2) - y(1))/h(1) - df1 );     %fila 1
    b(2:n-1) = 3*( (y(3:n) - y(2:n-1))./h(2:n-1) - (y(2:n-1) - y(1:n-2))./h
        (1:n-2) );   %filas 2...n-1
    b(n) = 3*( dfn - (y(n) - y(n-1))/h(n-1) );     %fila n

    % - Calculamos (metodo de crout)
    l(1) = 2*h(1);
    u(1) = 0.5;
    z(1) = b(1)/l(1);

    for i = 2:n-1
        l(i) = 2 * ( x(i+1)-x(i-1) ) - h(i-1) * u(i-1);
        u(i) = h(i) / l(i);
        z(i) = (b(i) - h(i-1) * z(i-1) ) / l(i);
    endfor

    l(n) = h(n-1) * (2-u(n-1));
    z(n) = (b(n) - h(n-1)*z(n-1) ) / l(n);
    ci(n) = z(n);
```

```
    % Paso 7:
    for i = n-1:-1:1
        ci(i) = z(i) - u(i) * ci(i+1);
        bi(i) = (y(i+1)-y(i))/ h(i) - h(i) * ( ci(i+1) + 2 * ci(i) ) / 3;
        di(i) = (ci(i+1)-ci(i))/(3*h(i));
    endfor

    ai = y(1:n-1)';
    bi = bi';
    ci = ci(1:n-1)';
    di = di';
endfunction
```

6.3. funcion_spline

```
function [S,dS]=funcion_spline(x1,y1,df1=0,df2=0)
    % extremos sujetos
    [a,b,c,d] = cubic_spline_clamped(x1,y1,df1,df2);
    % extremos libres
    %[a,b,c,d] = cubic_spline_natural(x1,y1);

    S=@(x) a(1)*(x==x1(1));

    M=[d c b a];
    dM=[];
    dS= @(x) 0;
    for i=1:length(x1)-1
        dM=[dM;polyder(M(i,:))];
        S=@(x) S(x) + polyval(M(i,:),x-x1(i)).*(x>x1(i)).*(x<=x1(i+1));
        dS=@(x) dS(x) + polyval(dM(i,:),x-x1(i)).*(x>x1(i)).*(x<=x1(i+1));
    endfor
endfunction
```

6.4. gauss_xw

```
function [x, w] = gauss_xw(n)
    % [x, w] = gauss_xw(n)
    % Genera las abscisas y pesos para la Cuadratura Gauss-Legendre.
    % n: numero de puntos de integracion
    % x: abscisas de la cuadratura
    % w: pesos de la cuadratura
    x = zeros(n,1);
    w = x;
    m = (n+1)/2;
    for ii=1:m
        z = cos(pi*(ii-.25)/(n+.5)); % estimado Inicial.
        z1 = z+1;
        while abs(z-z1)>eps
            p1 = 1;
            p2 = 0;
            for jj = 1:n
                p3 = p2;
                p2 = p1;
                p1 = ((2*jj-1)*z*p2-(jj-1)*p3)/jj; % El polinomial. Legendre
```

```
            endfor
            pp = n*(z*p1-p2)/(z^2-1); % La L.P. Derivada.
            z1 = z;
            z = z1-p1/pp;
        endwhile
        x(ii) = -z; % Construye las abscissas.
        x(n+1-ii) = z;
        w(ii) = 2/((1-z^2)*(pp^2)); % Construye los pesos.
        w(n+1-ii) = w(ii);
    endfor
endfunction
```

6.5. intGauss

```
function [I,it,r,t,L] = intGauss(f,a,b,n,tolerancia,maxit)
  % aproxima la integral de f sobre [a,b],
  %subdividiendo en L subintervalos

  tic();

  % calculamos los pesos y los puntos a evaluar t una sola vez
  [xg,w] = gauss_xw(n);
  t = linspace(-1,1,n);

  L=1;
  h=b-a;

  y=linspace(a,b,L+1);
  I=0;
  for i=1:L
    t = h/2*(xg+1)+y(i);
    I += h/2*(w'*f(t));
  endfor
  %----------------
  for it=2:maxit
    % duplicamos los subintervalos
    L=L*2;
    h=h/2;

    y=linspace(a,b,L+1);
    I1=0;
    for i=1:L
      t = h/2*(xg+1)+y(i);
      I1 += h/2*(w'*f(t));
    endfor

    %residuo
    r(it-1) = abs(I1-I);
    if r(it-1) < tolerancia
      I = I1;
      break;
    endif

    I = I1;
  endfor
```

```
  t = toc();

  if it == maxit
    disp("la integral no converge para maxit iteraciones");
  endif
endfunction
```

6.6. intNC

```
function [I,it,r,t,L] = intNC(f,a,b,n,tolerancia,maxit)
  % aproxima la integral de f sobre [a,b]
  % utilizando la formula de Newton-Cotes compuesta
  % de n puntos, subdividiendo en L subintervalos

  tic();

  % calculamos los pesos una sola vez
  w = pesosNC(n);

  L=1;
  h = b-a;
  y = linspace(a,b,L+1);
  % crea una matriz con los puntos a evaluar
  x = linspace(0, 1, n).' * (y(2:end) - y(1:end-1)) + y(1:end-1);
  x = reshape(x, [], L);

  fx = f(x);
  I = sum(h * (fx .* w)(:));
  % ----------
  for it=2:maxit
    % duplicamos los subintervalos
    L=L*2;
    h = h/2;
    y = linspace(a,b,L+1);
    % crea una matriz con los puntos a evaluar
    x = linspace(0, 1, n).' * (y(2:end) - y(1:end-1)) + y(1:end-1);
    x = reshape(x, [], L);

    fx = f(x);
    I1 = sum(h * (fx .* w)(:));

    %residuo
    r(it-1) = abs(I1-I);
    if r(it-1) < tolerancia
      I = I1;
      break;
    endif

    I = I1;
  endfor

  t = toc();

  if it == maxit
```

```
    disp("la integral no converge para maxit iteraciones");
  endif
endfunction
```

6.7. pesosNC

```
function w = pesosNC(n)
    % function w = pesosNC(n)
    % se calculan los pesos
    % de la formula de Newton-Cotes de n puntos
    x = linspace(0,1,n);
    A = ones(n,n);
    for i=2:n
      A(i,:) = A(i-1,:) .* x;
    end
    b = 1./(1:n)';
    w = A\b;
endfunction
```

6.8. PolyLag

```
%P: vector de coeficientes del polinomio de Lagrange que interpola los
    puntos (x,y)
%x: valores que toman los puntos interpolantes
%y: valores de la funcion interpolada en los puntos x[i]
function [P] = PolyLag (x,y)
  n = length(x);

  % calculamos el primer miembro solo para no enquilombarnos de indices
      despues
  L1 = [1 (-1)*x(2)];                 %(x - x_2)
  div = x(1)-x(2);                    %(x_1 - x_2)
  for i=3:n
    L1 = conv(L1, [1 (-1)*x(i)]);     %(x - x_2)(x - x_3)(x - x_4)...(x - x_n)
    div = div*(x(1)-x(i));            %(x_1 - x_2)(x_1 - x_3)(x_1 - x_4)...(
        x_1 - x_n)
  endfor
  P = L1 * (y(1)/div);                %f(x_1)*(x - x_2)(x - x_3)(x - x_4)...(x
      - x_n) / (x_1 - x_2)(x_1 - x_3)(x_1 - x_4)...(x_1 - x_n)

  % calculamos el resto de miembros del polinomio
  for i=2:n
    % inicializamos el i-esimo miembro del polinomio de Lagrange
    % es lo mismo que para el primero
    Laux = [1 (-1)*x(1)];
    div = x(i)-x(1);

    for j=2:n
      if j!=i                         %evitamos (x - x_i) y (x_i - x_i)
        Laux = conv(Laux, [1 (-1)*x(j)]);
        div = div*(x(i)-x(j));
      endif
    endfor
    P = P + Laux * (y(i)/div);
```

```
    endfor

endfunction
```