Trabajo práctico 3: ”Interpolación e integración numérica”

”Cálculo Numérico 2023”

Gareis, Nahuel Julián, DNI 43538493

***Abstract*—En el trabajo a continuación se hará uso de los conceptos de aproximación por funciones interpolantes e integración numérica.**

Las aproximaciones por funciones interpolantes, como el método de Lagrange o Newton para diferencias divididas, logran encontrar una expresión que pueda conectar ciertos puntos de tal manera que sea posible obtener una expresión polinómica que los conecte. Esto es importante ya que en la práctica se presentan muchos casos donde queremos encontrar una aproximación a fenómenos naturales usando instrumentación que no provee una cantidad infinita de datos de entrada, tal que una función matemática sea completamente definida. Para ello, Lagrange y Newton permiten encontrar una función polinómica que se aproxima en comportamiento a la tendencia de comportamiento de las mediciones. En este trabajo usaremos otras formas de interpolación que se denominan spline cúbico natural y spline cúbico sujeto. Estos últimos tienen la particularidad de que es posible definirles condiciones para la derivada de dicha función, haciendo posible un mejor aprovechamiento de fenómenos físicos conocidos como la velocidad y que, debido a la naturaleza de nuestras mediciones, nos será útil para generar curvas con mejor aproximación al comportamiento real de una partícula en movimiento.

Por último, usaremos integración numérica para medir la longitud de arco de la curva generada. El algoritmo seleccionado es el de Simpson.

***Index Terms*—interpolación,integración,newton,simpson.**

I. Introduction

En el desarrollo de la computacion moderna, los métodos de aproximacion numéricos de funciones se utilizan en cada vez más ambitos debido al abaratamiento del poder de computo, haciendo posible el uso de estos métodos de manera más ágil. Los metodos de aproximación numéricos se pueden usar para estimar el comportamiento pasado y futuro de una variable a analizar. Debido a esto han encontrado su lugar en, además de la ingeniería en informática, las finanzas, medicina, la física, entre otras.

En este experimento se provará cómo es posible obtener una expresion matemática para un objeto en movimiento cuyas posiciones fueron obtenidas mediante sensores en intervalos de tiempo. El uso de el metodo de trazadores cubicos nos permite hacer una estimación de la trayectoria que el objeto tomó en los periodos donde los sensores no proveían información acerca de su posición.

Luego, mediante integración numerica obtendremos la medida del espacio recorrido por el objeto haciendo uso de la formula matemática de la longitud de arco. Resaltand así la importacia y utilidad de los trazadores cúbicos y tambien lde la integración numérica.

II. Metodología y el experimento

TRAZADORES CÚBICOS: El experimento fue realizado usando sensores para el trackeo de posicion de el objeto, de los cuales deriamos las siguientes tabalas de información. Tambien además, se registró que la velocidad inicial en y es

Tiempo(s)	Posición en X (cm)	Posición en Y (cm)
0	2	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

TABLE I
Posición X e Y a través del tiempo.

de $\frac{\pi}{2}$ cm/s y la final es $\frac{\pi}{2}$ cm/s

Un spline cúbico es una función polinómica definida por partes que interpola un conjunto de puntos de datos. Consiste en múltiples polinomios cúbicos que están conectados suavemente en los puntos de datos. Supongamos que tenemos los puntos de datos (x_0, y_0) y (x_1, y_1).

El polinomio spline cúbico se puede definir de la siguiente manera:

Para $x \in [x_0, x_1]$, el polinomio spline cúbico se define como una función $S(x)$ tal que:

$$\begin{cases} S_0(x) = a_0 + b_0(x - x_0) + c_0(x - x_0)^2 + d_0(x - x_0)^3 \\ S_1(x) = a_1 + b_1(x - x_1) + c_1(x - x_1)^2 + d_1(x - x_1)^3 \end{cases} \tag{1}$$

que debe cumplir las condiciones

$$S_i(x_i) = f(x_i), S_i(x_{i+1}) = f(x_{i+1}) \tag{2}$$

$$S_i'(x_i) = S_{i+1}'(x_i), S_i'(x_{i+1}) = S_{i+1}'(x_{i+1}) \tag{3}$$

$$S_i''(x_i) = S_{i+1}''(x_i), S_i''(x_{i+1}) = S_{i+1}''(x_{i+1}) \tag{4}$$

Si definimos un parámetro h que represente el intervalo entre dos puntos de datos adyacentes (en el caso de los 2 puntos, h corresponde a $x_1 - x_0$), se puede obtener una expresion matricial. En la forma matricial del sistema de ecuaciones, los valores h_0, h_1, h_2, etc., representan los intervalos entre los puntos de datos. Por ejemplo, $h_0 = x_1 - x_0$, $h_1 = x_2 - x_1$, y así sucesivamente.

Los coeficientes a_0, b_0, c_0, d_0, a_1, b_1, c_1, d_1 son las incógnitas que deben determinarse al resolver el sistema de ecuaciones lineales.

$$[A]u = b$$

donde:

$$[A] = \begin{bmatrix} h_0 & 2(h_0+h_1) & h_1 \\ 0 & 2(h_1+h_2) & h_2 \\ h_2 & 2(h_2+h_3) & h_3 \end{bmatrix} \quad (5)$$

$$u = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \quad (6)$$

$$b = \begin{bmatrix} d_0 \\ d_1 \\ d_2 \end{bmatrix} \quad (7)$$

Sin embargo, esto crea una contradicción en el primer y último nodo de nuestro spline cúbico, ya que las derivadas estarían determinadas por un nodo cuya derivada no está definida.

Por lo tanto, se hace una distinción entre splines cúbicos naturales y splines cúbicos sujetos. Los splines cúbicos sujetos fijan valores para su primera y última iteración, mientras que los naturales dejan las derivadas igual a 0. En nuestro experimento, usaremos ambos casos, ya que hay condiciones específicas para las derivadas en el spline que se generará para el movimiento en el eje y, mientras que en el eje x no hay condiciones específicas, por ende, podemos asumir un spline cúbico natural.

INTEGRACIÓN NUMÉRICA: El método de integración compuesta de Newton-Cotes es una técnica numérica utilizada para aproximar el valor de una integral definida. Consiste en dividir el intervalo de integración en subintervalos y aplicar una fórmula de Newton-Cotes en cada subintervalo. Luego, se suman las aproximaciones obtenidas en cada subintervalo para obtener la aproximación final de la integral.

Existen varias fórmulas de Newton-Cotes, que difieren en la forma en que se interpola la función en cada subintervalo. Las fórmulas más comunes son:

- Regla del punto medio:

$$\int_a^b f(x)\,dx \approx (b-a)\cdot f\left(\frac{a+b}{2}\right)$$

- Regla del trapecio:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{2}\cdot[f(a)+f(b)]$$

- Regla de Simpson:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\cdot\left[f(a)+4f\left(\frac{a+b}{2}\right)+f(b)\right]$$

Estas fórmulas se aplican en cada subintervalo de la partición del intervalo de integración. La precisión de la aproximación depende de la cantidad de subintervalos utilizados y del orden de la fórmula de Newton-Cotes seleccionada.

El método de integración compuesta de Newton-Cotes es ampliamente utilizado debido a su simplicidad y facilidad de implementación. Sin embargo, es importante tener en cuenta que puede haber errores de aproximación significativos cuando se utiliza en intervalos de integración grandes o cuando la función a integrar tiene una variación rápida.

Esta formula de integración numérica será la utilizada para calcular el valor de la longitud de arco, cuya formula a integrar será:

$$L(t) = \sqrt{x'(t)^2 + y'(t)^2} \quad (8)$$

procedemos entonces a analizar y obtener resultados en base a los métodos mencionados.

III. Resultados y análisis

Usando la función del anexo 'funcion_spline' para la tabla (Tab. II) obtenemos la grafica de los siguientes trazadores.

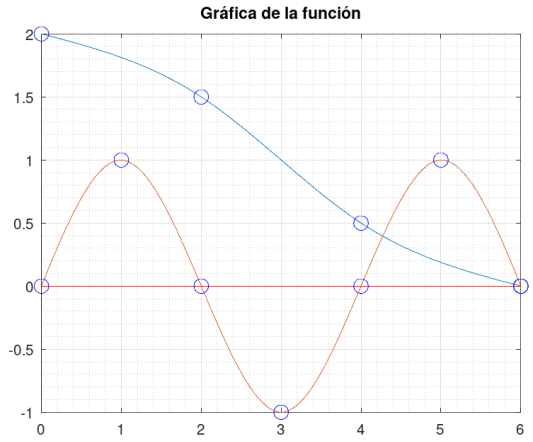

Fig. 1. Trazadores cúbicos para X e Y . S_x es la curva azul y S_y es la curva amrilla, los circulos indican los puntos con los que los trazadores fueron formados.

también proporciona una gráfica de la derivada de (Fig. 1)

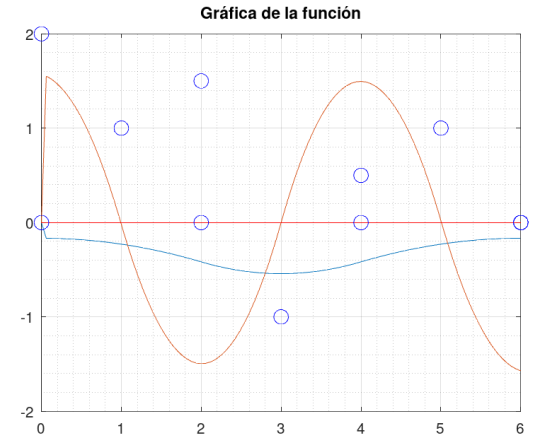

Fig. 2. Derivada de los trazadores cúbicos para X e Y . dS_x es la curva azul y dS_y es la curva amrilla, los circulos indican los puntos con los que los trazadores fueron formados.

Ahora para poder obtener el movimiento real del objetos, se formará la function $p(t) = (x(t), y(t))$ la cual tiene la forma:

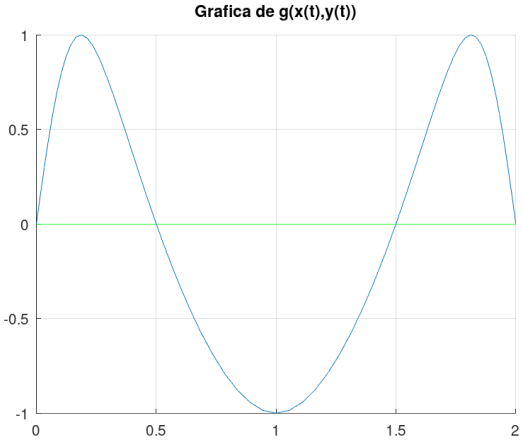

Fig. 3. Función de posición del objeto a través del tiempo.

Por último utilizaremos la funcion 'intNCcompuesta' del anexo junto con (Eq. 8) para obtener el valor de la longitud de arco del recorrido mostrado en (Fig. 3) con 6 cifras de presición. El valor final es obtenido es : 6.4656709

$$\int_0^6 L(t)\ dt = 6.4656709 \tag{9}$$

IV. Conclusión y Resumen

Podemos definir concluir que los métodos numéricos son sumamente útiles para el desarrollo de análisis de fenomenos prácticos. En este caso el uso de trazadores cúbicos nos permitio hacer una una aproximación a los datos faltantes del movimiento del objeto (Fig. 3) devido a la limitante de la frecuencia con la que el sensor tomaba los datos. Por otro lado el uso del cálculo numérico de integrales permitió poder evaluar una expresión complicada de obtener analíticamente logrando así obtener el resultado (Eq. 9).

Appendix A
Rutinas empleadas para el desarrollo del informe

En la siguiente hoja estarán adjuntos los algoritmos utilizados para la sección de resultos.

```
% 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) )
    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

function [a, b, c, d] = cubic_spline_natural(x,y)
    # Pasamos un arreglo de puntos x e y
    # Es una funciÃ³n para el calculo de una Spline CÃºbica Natural, es decir con
    # derivadas segundas nulas en el contorno del dominio.
    # Programado en base a Burden EdiciÃ³n 10 pag 110.
    # Ojo tener cuidado el contador i estÃ¡ inicializado en 1.
    # S(x) = Sj(x) = aj + bj(x-xj) + cj(x-xj)^2 + dj(x-xj)^3 para xj <= x <= xj+1;
    # S''(x1) = 0  , S''(xn+1) = 0 (libre o natural)

    % Medimos la longitud de los datos
    n = length(x);
    alpha = zeros(n,1);
    c = alpha;

    % Paso 1: Calculamos los h de cada Spline.
    h(1:n-1) = x(2:n)-x(1:n-1); # sin usar el lazo

    % Paso 2: Calcula los terminos independientes (alpha)
    alpha(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-

    # Resolvemos el sistema lineal tridiagonal (FactorizaciÃ³n de Crout)
    % Paso 3:
    l = ones(n,1);
    u = zeros(n,1);
    z = zeros(n,1);

    % Paso 4:
    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) = (alpha(i)-h(i-1)*z(i-1))./l(i);
    endfor

    %Paso 6:

    for i = n-1:-1:1
        c(i) = z(i)- u(i)*c(i+1);
        b(i) = (y(i+1)-y(i))./h(i)-(h(i)*(c(i+1)+2*c(i)))/3;
        d(i) = (c(i+1)-c(i))./(3*h(i));
    endfor

    a = y(1:n-1)';
    b = b';
    c = c(1:n-1);
    d = d';
endfunction

function [S,dS]=funcion_spline(x1,y1,df1,df2)

    %la modifique un toque para que ademÃ¡s grafique y
    %que defaultee df1 y df2 a 0 y haga un spline natural
    %si no le pasas esos parametros

    nargin

    if nargin < 3
        [a,b,c,d] = cubic_spline_natural(x1,y1);
    elseif nargin == 3
        [a,b,c,d] = cubic_spline_clamped(x1,y1,df1,df1);
    else
        [a,b,c,d] = cubic_spline_clamped(x1,y1,df1,df2);
    endif

    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

    dots(:,1)=y1';
    dots(:,2)=x1';

    % - THIS IS FOR PLOTTING PORPUSES - %
    plotFunction(S,x1(1),x1(end),100,1);%
    plotDots(dots,1);                   %
    % ------------------------------- %

```

```
127:     % - THIS IS FOR PLOTTING PORPUSES -  %
128:     plotFunction(dS,x1(1),x1(end),100,2);%
129:     plotDots(dots,2);                    %
130:     % ---------------------------------  %
131:
132: end
133:
134: function succesful = plotDots(dotsVector,figNumber)
135:     succesful = 0;
136:     for i = 1 : length(dotsVector')
137:
138:         figure(figNumber)
139:         plot(dotsVector(i,2),dotsVector(i,1),'bo','MarkerSize', 10)
140:         grid on
141:         grid minor
142:         hold on
143:
144:     endfor
145:     succesful = 1;
146: end
147:
148: function res = plotFunction(f_x,a,b,ndots,figNumber)
149:
150:     succesful = 0;
151:     x = linspace(a,b,ndots);
152:     y = f_x(x);
153:
154:     figure(figNumber)
155:     plot(x,y)
156:
157:     grid on
158:     grid minor
159:     title("GrÃ¡fica de la funciÃ³n")
160:     hold on
161:     line([a b],[0 0],'color','r')
162:     succesful = 1;
163:
164: end
165:
166: function [finalVal,it,t,r] = simpson(f_x,a,b,tol,maxIt)
167:     tic();
168:     dist = abs(a-b);
169:     last = 0;
170:     finalVal = 0;
171:     for i = 1 : maxIt
172:         n = 2*i;
173:         w = simpsonWeight(n);
174:         dx = abs(a-b)/n;
175:         it = 0;
176:
177:         for j = a:dx:b
178:             it++;
179:             finalVal = finalVal + f_x(j)*w(it);
180:         endfor
181:         finalVal = finalVal * dx/3;
182:         if abs(finalVal - last) < tol
183:             break;
184:         endif
185:         last = finalVal;
186:     endfor
187:     t = toc();
188: end
189:
```

```
190: function weights = simpsonWeight(n)
191:     if rem(n,2) ~= 0
192:         error("n is not even");
193:     else
194:         weights(1:n) = 2;
195:         weights(2:2:n) = 4;
196:         weights(1) = 1;
197:         weights(n+1) = 1;
198:     endif
199: end
200:
201: function intvalue = tp3()
202:     %Valores iniciales
203:     ty = 0:1:6
204:     tx = 0:2:6
205:     x = [ 2.0 1.5 0.5 0.0 ];
206:     y = [ 0.0 1.0 0.0 -1.0 0.0 1.0 0.0];
207:
208:     %Obtengo los splines
209:     [sX,dX] = funcion_spline(tx,x);
210:     [sY,dY] = funcion_spline(ty,y,pi/2,-pi/2);
211:
212:     %Obtengo un intervalo
213:     plotX = sX(0:0.1:6);
214:     plotY = sY(0:0.1:6);
215:
216:     %Ploteo
217:     figure 3;
218:     hold on;
219:     grid on;
220:     title("Grafica de g(x(t),y(t))");
221:     line([0 2],[0 0],'color','g')
222:     plot(plotX,plotY);
223:
224:     longarc = @(t) sqrt(dX(t).^2 + dY(t).^2)
225:     format long;
226:     intvalue = intNCcompuesta(longarc,0,6,6,19);
227: end
228:
229:
230: function w = pesosNC(n)
231: % function w = pesosNC(n)
232: % se calculan los pesos
233: % de la formula de Newton-Cotes de n puntos
234:
235: x = linspace(0,1,n);
236: A = ones(n,n);
237: for i=2:n
238: A(i,:) = A(i-1,:) .* x;
239: end
240: b = 1./(1:n)';
241: w = A\b;
242: end
243:
244: function Q = intNCcompuesta(f,a,b,L,n)
245: % function Q = intNCcompuesta(f,a,b,L,n)
246: % aproxima la integral de f sobre [a,b]
247: % utilizando la formula de Newton-Cotes compuesta
248: % de n puntos, subdividiendo en L subintervalos
249: y = linspace(a,b,L+1);
250: h = (b-a)/L;
251: % calculamos los pesos una sola vez
252: w = pesosNC(n);
```

```
253: Q = 0;
254: for i=1:L
255: x = linspace(y(i),y(i+1),n);
256: fx = f(x);
257: Q = Q + h*(fx*w);
258: end
259: end
```