ANÁLISIS NUMÉRICO

RETO DEL PERRO

Autores

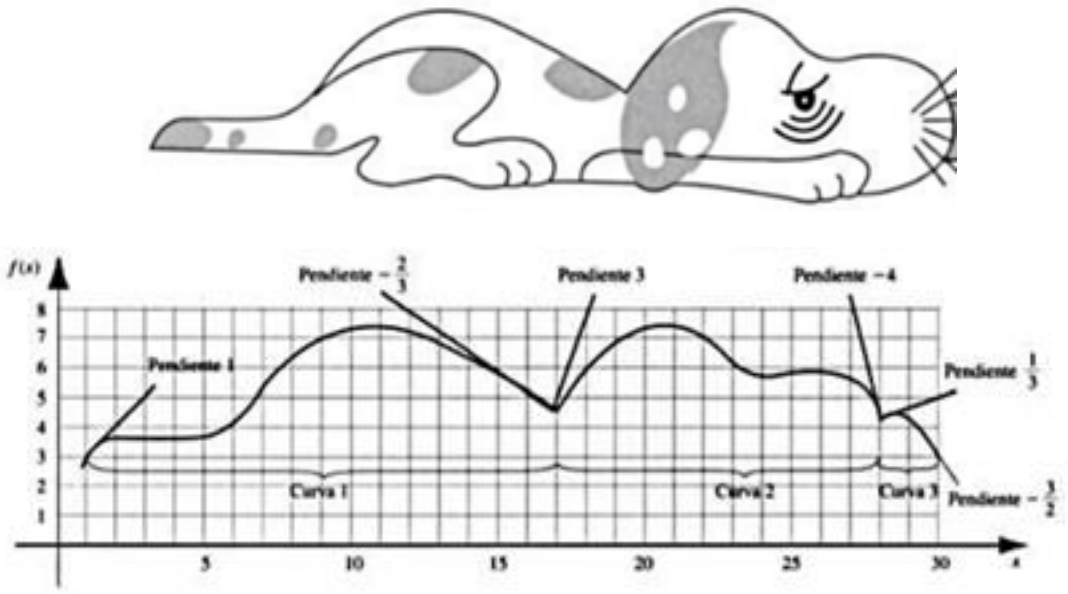
CAMILO EDUARDO CRUZ GUTIERREZ
ESTEBAN CASAS
JOHAN FERNEY GARCIA PACHON

Profesora
EDDY HERRERA

PONTIFICIA UNIVERSIDAD JAVERIANA
FACULTAD DE MATEMATICAS
BOGOTA D.C.

Problema

Construir un Interpolador (no necesariamente en forma polinómica) utilizando la menor cantidad de puntos k (parte superior y/o inferior o en total) y reproducir el dibujo del contorno completo del perrito sin bigotes (mejor exactitud) con la información dada.



coordenadas
y=c(3,3.7,3.9,4.5,5.7,6.69,7.12,6.7,4.45,7,6.1,5.6,5.87,5.15,4.1,4.3,4.1,3)

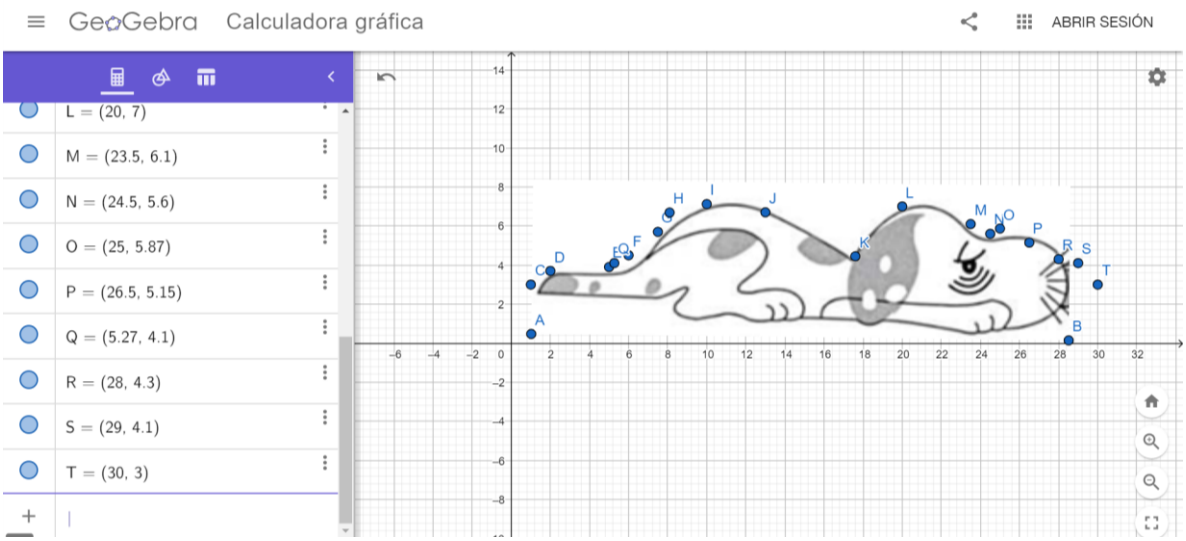
x=c(1,2,5,6,7.5,8.1,10,13,17.6,20,23.5,24.5,25,26.5,27.5,28,29,30)
length(x)
length(y)plot(x,y, pch=19, cex=0.5, col = "red", asp=1,xlab="X", ylab="Y", main="Diagrama ")

Metodología

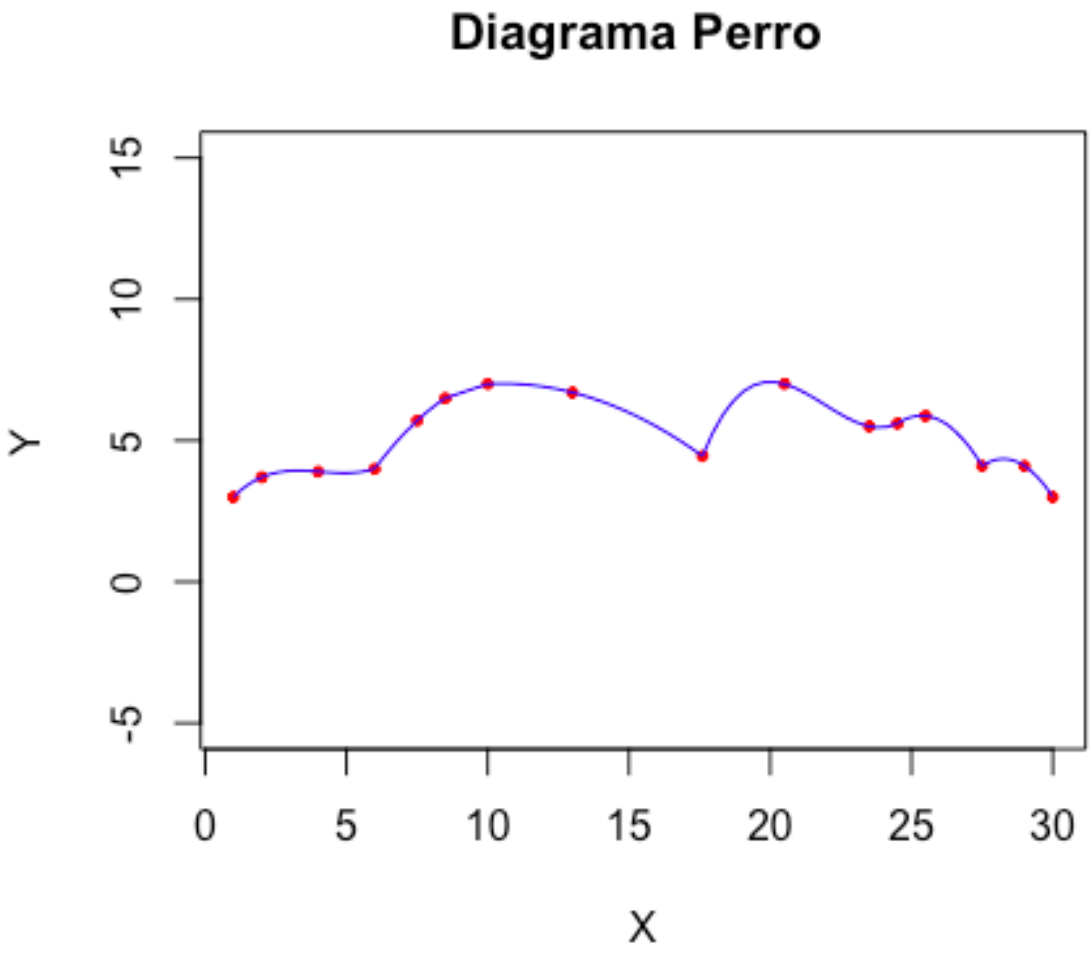
Colocamos las coordenadas dadas por la profesora en Geogebra para luego hallar los puntos de abajo de la figura, y así el perro quedara lo más parecido posible a la imagen.

1. Generamos trayectorias suaves con funciones polinómicas a trozos(splines) a partir de las coordenadas de geogebra.
2. Determinamos un polinomio distinto entre cada par de puntos consecutivos del conjunto de datos iniciales (mediante el método de spline).

3. Verificamos la primera y segunda derivada (que surgen a medida que hallamos los coeficientes del polinomio interpolador en el método de spline, que será explicado más adelante) en busca de la concavidad y máximos y mínimos en el intervalo, para hallar el punto más adecuado para el contorno del perro.
4. Debido a que el polinomio es de grado 3 se imponen 4 condiciones.
   a. $qi(x_i) = y_i$
   b. $q_i(x_i + 1) = y_i + 1$
   c. $q_i'(x_i + 1) = q_{i+1}'(x_i + 1)$
   d. $q_i''(x + 1) = q_{i+1}''(x_i + 1)$ donde $i = 1,2, \dots, n$



Puntos sacados de geogebra



Parte superior de la figura

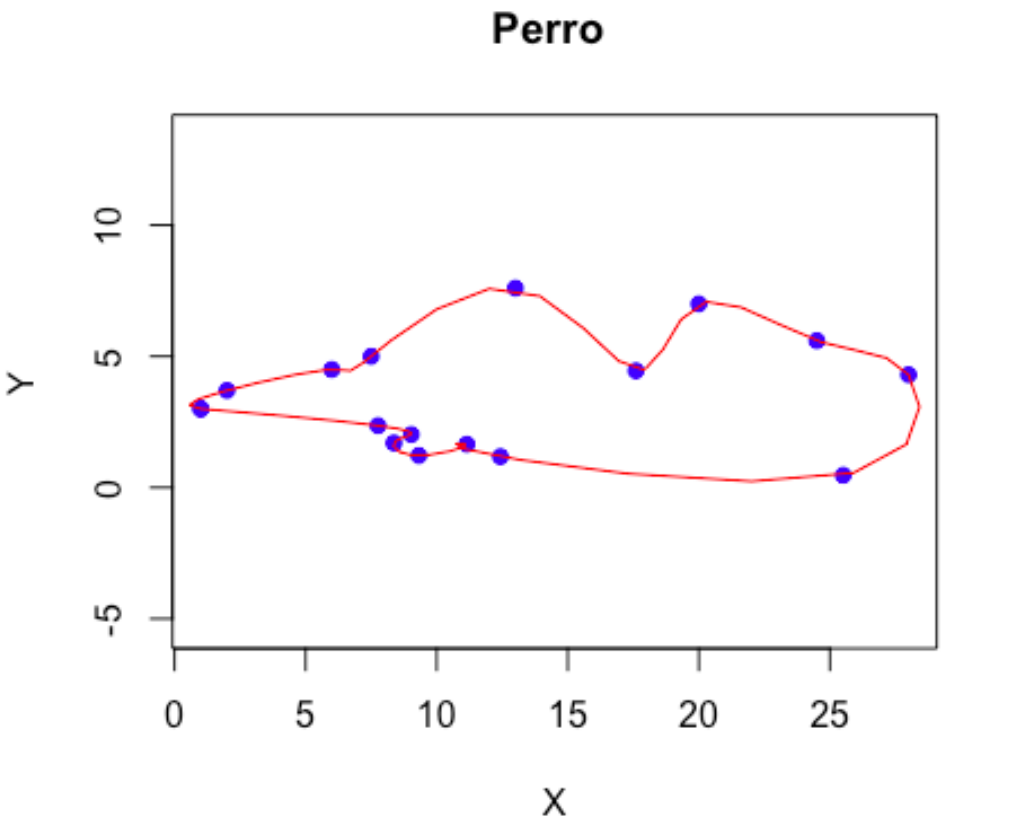


Imagen interpolada con 16 puntos

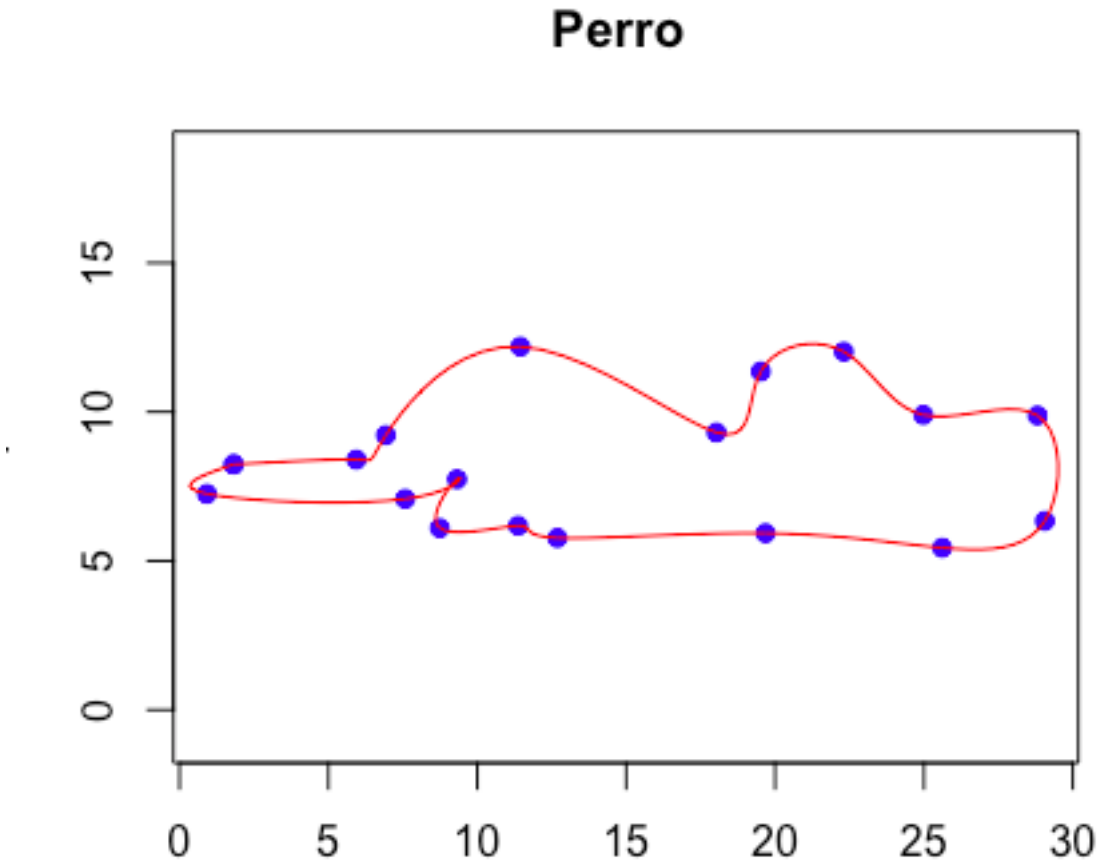


Diagrama final.

Algoritmo utilizado

- Spline cubico:

Los trazadores cúbicos (cubic splines) son usados para generar una función que interpola un conjunto de puntos de datos. Esto por medio de la unión de polinomios cúbicos -uno por cada intervalo-. Que genera una función con primera y segunda derivada continua. Del mismo modo el spline cubico tiene segunda derivada igual a 0 en la coordenada x del primer punto y el último punto de la tabla de datos.

De este modo supongamos que tenemos $n+1$ puntos (x_0, y_0) ..., (x_n, y_n) con $x_0 < x_1 < \cdots < x_n$, se realiza la interpolación de la función f en cada subintervalo $[x_k, x_{k+1}]$ con un polinomio cúbico (en realidad de grado ≤ 3) $s_k(x)$ de tal manera que el polinomio cúbico (o trazador cúbico) $s_i(x)$ en $[x_i, x_{i+1}]$ y el trazador cúbico $s_{i+1}(x)$ en $[x_{i+1}, x_{i+2}]$, coincidan en x_{i+1} y que también sus derivadas primera y segunda coincidan en este punto. Cada trazador cúbico coincide con f en los extremos de cada intervalo. [1]

Definición formal: Dado el conjunto de datos (t_0, y_0) , (t_1, y_1) , (t_n, y_n), se define función spline de grado k a la función s: $\mathbb{R} \rightarrow \mathbb{R}$ que satisface

i) S es un polinomio de grado menor o igual que k en cada intervalo $[t_{i-1}, t_i]$.

ii) S tiene $k-1$ derivadas continuas en $[t_0, t_n]$.Esto es, sobre cada intervalo S que está definido por un polinomio diferente de grado k.

$$S(x) = \begin{cases} S_0(x) & x \in [t_0, t_1) \\ S_1(x) & x \in [t_1, t_2) \\ \vdots & \\ S_{n-1}(x) & x \in [t_{n-1}, t_n) \end{cases}$$

- *Implementación:*

Con el fin de construir el spline cubico, se necesita encontrar cada $s_J(x)$, lo cual es en realidad encontrar los coeficientes a_J, b_J, c_J, d_J donde $0 \leq J \leq n-1$ y n es el total de datos. De este modo un spline definido en un intervalo que es dividido en n subintervalos, requiere encontrar $4n$ constantes.
De acuerdo a lo anteriormente expuesto se implementó un algoritmo que generaba estos coeficientes para cada x_i. (El algoritmo será agregado como un anexo al final del documento).
Coeficientes generadores del spline cubico:

j	x_J	a_j	b_J	c_J	d_J
0	1	3	0.784	0	-0.0839
1	2	3.7	0.532	-0.251	0.0421
2	6	4.5	0.542	0.254	-0.0479
3	9	7.12	0.774	-0.177	0.00109
4	14	6.7	-0.914	-0.161	0.0669
5	17.6	4.45	0.531	0.562	-0.142
6	20	7	0.775	-0.461	0.04895
7	23	6.5	-0.669	-0.0202	0.0436
8	24.5	5.6	-0.434	0.1760	0.0245
9	26.85	5.87	0.799	0.3487	-0.551
10	28.7	5.05	-3.57	-2.712	-0.887
11	29	3.71	-5.439	-3.510	-0.483
12	25.51	0.47	1.414	1.546	0.100
13	11.15	1.65	19.122	-2.779	-7.160
14	9.32	1.22	-42.63	36.532	85.142
15	8.37	1.7	118.463	-206.121	44.580
16	9.03	2.92	-95.353	-117.853	-33.405
17	7.76	2.36	42.357	9.419	0.464
18	1	3	0	0	0

Luego de esto ya con los coeficientes resueltos, se forma el polinomio interpolador que tiene la siguiente forma:
$s_J(x) = a_J + b_J(x - x_J) + c_J(x - x_J)^2 + d_J(x - x_J)^3$ Donde $0 \leq J \leq n-1$
Finalmente, de acuerdo a cada polinomio interpolador se grafica el contorno del perro.

Validacion de resultado

Es posible verificar el resultado por medio de la grafica generada por el metodo utilizado, debido a que no existe ningun tipo de deformacion en cuanto a la imagen del perro original. Lo cual permite verificar la forma del perro.

Algoritmo

Codificación

```
library(polynom)
library(PolynomF)
x=c(1,2,6,9,14,17.6,20,23,24.5,26.85,28.7,29,25.51,11.15,9.32,8.37,9.03,7.76,1)
y=c(3,3.7,4.5,7.12,6.7,4.45,7,6.5,5.6,5.87,5.05,3.71,0.47,1.65,1.22,1.7,2.92,2.36,3)
plot(x,y, pch=19, cex=0.9, col = "blue", asp=1,xlab="X", ylab="Y", main="Perro ")
n=19
pint<-function(x,y){
  t = 1:length(x)
  sx = spline(t,x)
  sy = spline(t,y)
  lines(sx[[2]],sy[[2]],type='l', col="red")
}
pint(x,y)
```

- Tabla donde está la interpolación en los n-k puntos (no seleccionados):

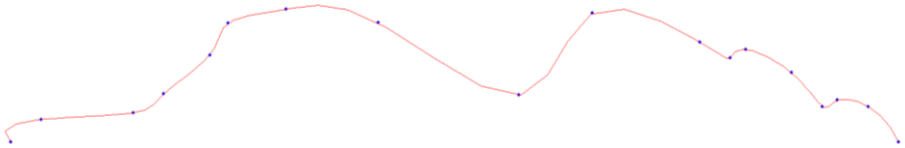
x	y	$f(x)$
5	3.9	1.335817
6	4.5	4.5
7.5	5.7	6.221004
8.1	6.69	-3.003772
10	7.12	17.13745
13	6.7	-0.429965
17.6	4.45	4.45
20	7	7
23.5	6.1	6.94484
24.5	5.6	5.6
25	5.87	2.654599
26.5	5.15	4.001707
27.5	4.1	7.49181
28	4.3	7.189713
29	4.1	3.71
30	3	-0.8941601

- *El polinomio o la función interpolante:*

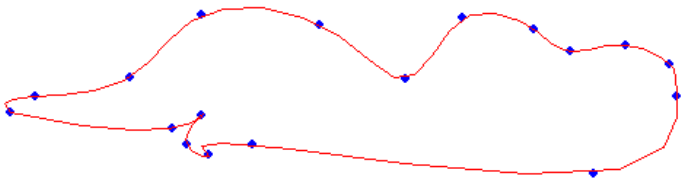
La funcion interpolante tiene la forma :

$s_J(x) = a_J + b_J(x - x_J) + c_J(x - x_J)^2 + d_J(x - x_J)^3$ Donde $0 \le J \le n - 1$

- *Puntos y contorno original:*



- *Puntos y contorno interpolado:*



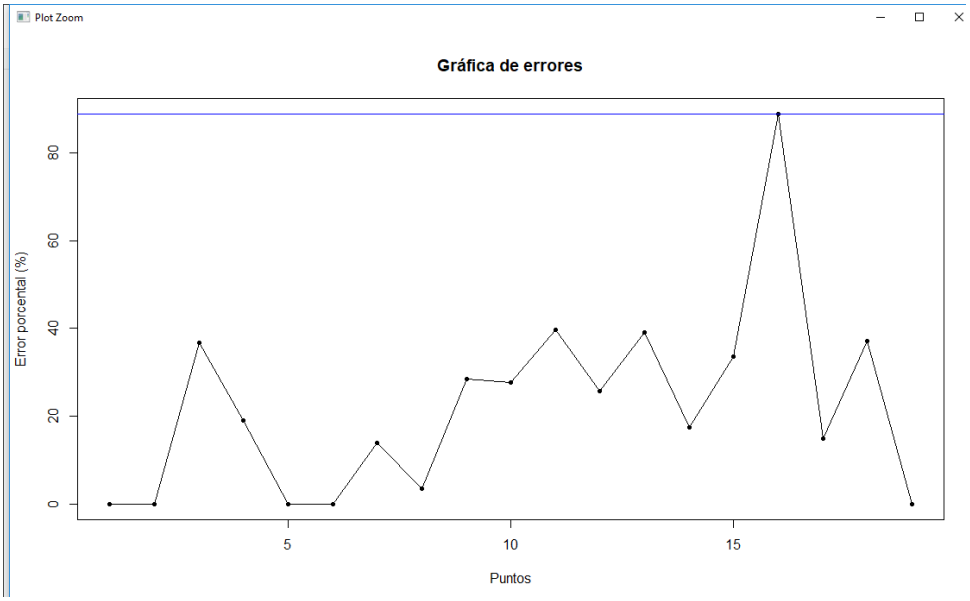
Cota de error

Para calcular la cota de error se calcularon los errores relativos de cada punto donde

- *Valores interpolados:*

x	y	$f(x)$	Error (%)
1	3	3	0
2	3.7	3.7	0
6	7.12	4.5	36.79775
13	6.7	5.429965	18.956
17.6	4.45	4.45	0
20	7	7	0
23.5	6.1	6.94484	13.84983
24.5	5.8	5.6	3.448276
26.5	5.6	4.001707	28.54095
27.5	5.87	7.49181	27.62879
28	5.15	7.189713	39.60608
29	5	3.71	25.8
24.1	4.9	6.81432	39.06775
10.1	2.5	2.06441	17.344
9.2	2	3.010759	33.538
8.37	0.9	1.7	88.88889
9.03	2.54	2.92	14.96063
6.98	2.8	3.840665	37.164
Error Relativo total			318.5889

tabla de errores errores relativos de los puntos interpolados.



Grafica de errores de los puntos interpolados.

De acuerdo a la grafica podemos determinar cual es la cota experimental. Que en este caso es de 0.88 . Adicionalmente, podemos observar que desde el punto 11 aproximadamente hay picos mas pronunciados en comparación a la grafica en general esto es debido a que estos puntos fueron sacados de geogebra y ya teniendo algo de teoria pudimos sacar varios puntos importantes sin necesitar de interpolarlos, pero en el caso del pico que define la cota escogimos un valor que no era tan bueno, por lo que genera ese error tan grande en comparación del resto.

Tabla valores interpolados

- *Los valores interpolados (tenga en cuenta los que no utilizo), los originales y el error relativo, calcule un error relativo total como la suma de los errores relativos.*

Debido a que usamos la función spline en R esta solo nos da los puntos x,y que necesitamos, mas no los valores interpolados como tal. Debido a esto usamos la función splinefun, la cual nos retorna una función en la cual podemos evaluar el interpolador cubico de spline para hallar el valor interpolado. Esta implementación será hallada en el apartado de anexos en la parte final del documento.

- *Valores interpolados:*

x	y	$f(x)$	Error (%)
1	3	3	0
2	3.7	3.7	0
6	7.12	4.5	36.79775

13	6.7	5.429965	18.956
17.6	4.45	4.45	0
20	7	7	0
23.5	6.1	6.94484	13.84983
24.5	5.8	5.6	3.448276
26.5	5.6	4.001707	28.54095
27.5	5.87	7.49181	27.62879
28	5.15	7.189713	39.60608
29	5	3.71	25.8
24.1	4.9	6.81432	39.06775
10.1	2.5	2.06441	17.344
9.2	2	3.010759	33.538
8.37	0.9	1.7	88.88889
9.03	2.54	2.92	14.96063
6.98	2.8	3.840665	37.164
Error Relativo total			318.5889

tabla de errores errores relativos de los puntos interpolados.

- *Valores originales* :

x	y	$f(x)$	Error
1	3	3	0
2	3.7	3.7	0
6	7.12	7.12	58.22222
13	6.7	2.591786	63.59851
17.6	4.45	6.539195	2.400074
20	7	4.45	0
23.5	6.1	7	0
24.5	5.8	7.332235	12.80362
26.5	5.6	5.8	3.571429
27.5	5.87	5.762159	1.837157
28	5.15	4.88978	3.17268
29	5	5	34.77089
24.1	4.9	6.290935	1238.497
10.1	2.5	4.683138	183.8266
9.2	2	1.705541	39.79843
8.37	0.9	0.9	47.05882
9.03	2.54	2.54	13.0137
6.98	2.8	0.3835179	83.74924
Error Relativo total			3310.179

tabla de errores errores relativos de los puntos originales.

$$\varepsilon = \frac{f(x)}{s(x)}$$

$$\varepsilon = \frac{3310.179}{318.5889}$$

$$\varepsilon = 10.39$$

De acuerdo a esto el valor del error relativo total fue menor al de la cota de error.

Eficiencia de su método

Para el análisis de la eficiencia del método implementado se realizó una comparación entre el número de operaciones que realiza nuestro método (Splines cúbicos) respecto al número de puntos con los que se quería interpolar

NUMERO DE PUNTOS	NUMERO DE OPERACIONES
18	126
17	116
16	113
15	105
14	98
13	91

Como Podemos ver la disminución de operaciones dado el número de puntos se puede expresar de la forma ($\frac{1}{3}$On)

Índice Jaccard

El método elaborado consiste en recibir todos los puntos, tanto los originales, como los modificados para contar el número de puntos que se mantuvieron, es decir, los puntos que están presentes tantos en los puntos originales como en los modificados, este método calcula el número de aciertos tanto para los puntos del eje x como para los puntos de eje y

```
Jaccard<-function(x,y,x1,y2){
  buenosx=0
  malosx=0
  buenosy=0
  malosy=0
  for (i in x)
    for(j in x1)
      if(i==j)
        buenosx++
    malosx=length(x)-buenosx
  for (i in y)
    for(j in y1)
      if(i==j)
        buenosy++
    malosy=length(y)-buenosy
}
```

La forma en la que estamos calculando el índice de jaccard es obteniendo el número de aciertos “Puntos que se mantuvieron”, sobre puntos totales.
Obtenemos que:
Puntos Totales: 18
Numero de Aciertos: 7
Incide = 7/18 =38.89%
Como podemos ver el índice nos dio relativamente bajo, pero esto no indica la confiabilidad de los puntos, debido a que, para la elaboración de este proyecto, se agregaron un mayor número de puntos que los que había inicialmente además de cambiar o descartar algunos puntos originales. Este índice encontrado nos expresa el grado de similitud entre los dos conjuntos de puntos.

Preguntas

1. *¿El origen se puede modificar?*

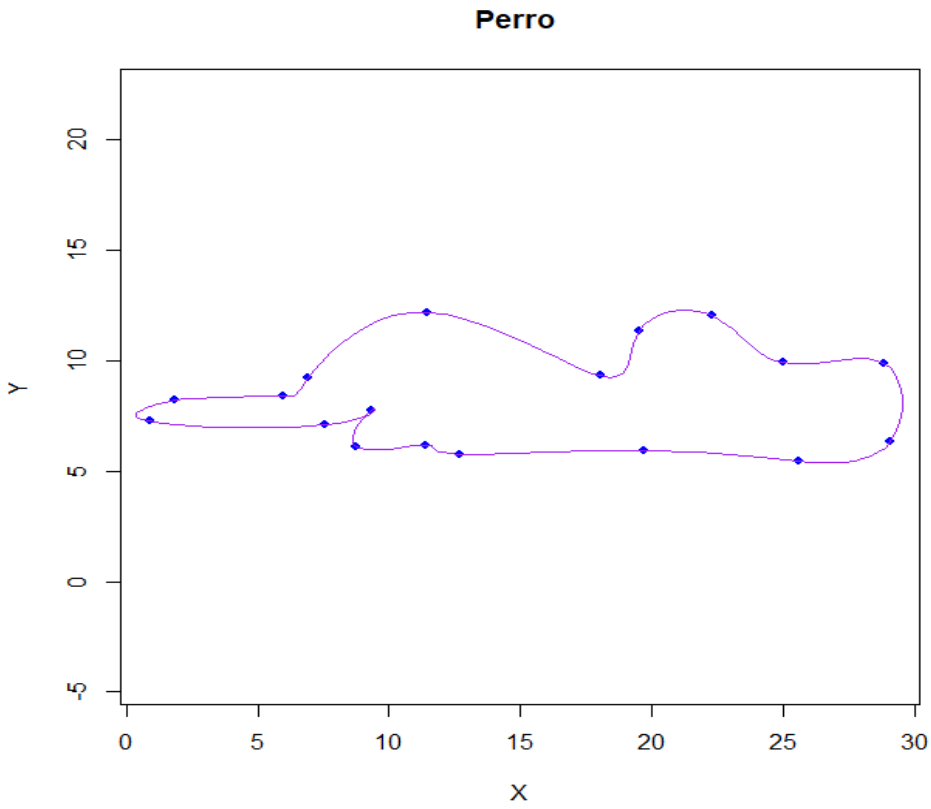
Para la realización de nuestra gráfica, tomamos como punto de inicio, el punto más extremo de la cola del perro, la figura obtenida fue a nuestro parecer muy aproximada a la buscada, a excepción de algunos puntos donde la figura original presentaba uno picos muy pronunciados muy difícil de reproducir por el método que estábamos usando:
Los puntos tomados inicialmente fueron:

x=c(1.,2,6,7.5,13,17.6,20,24.5,28,25.51,4.88,12.43,11.15,9.32,8.37

,9.03,7.76)

y=c(3,3.7,4.5,5,7.6.7,4.45,7,5.6,4.3,0.47,1.37,1.18,1.65,1.22,1.7,2.

02,2.36)



Ahora bien, realizamos varias pruebas modificando el punto de origen.

Primera prueba:
x=c(19.52,22.31,24.98,28.81,29.06,25.61,19.68,12.69,11.37,8.74,9.32,7.58,0.92,1.82,5.94,6.93,11.45,18.03,19.52)
y=c(11.36,12.02,9.90,9.88,6.34,5.44,5.93,5.77,6.18,6.10,7.74,7.08,7.25,8.24,8.40,9.22,12.19,9.31,11.36)

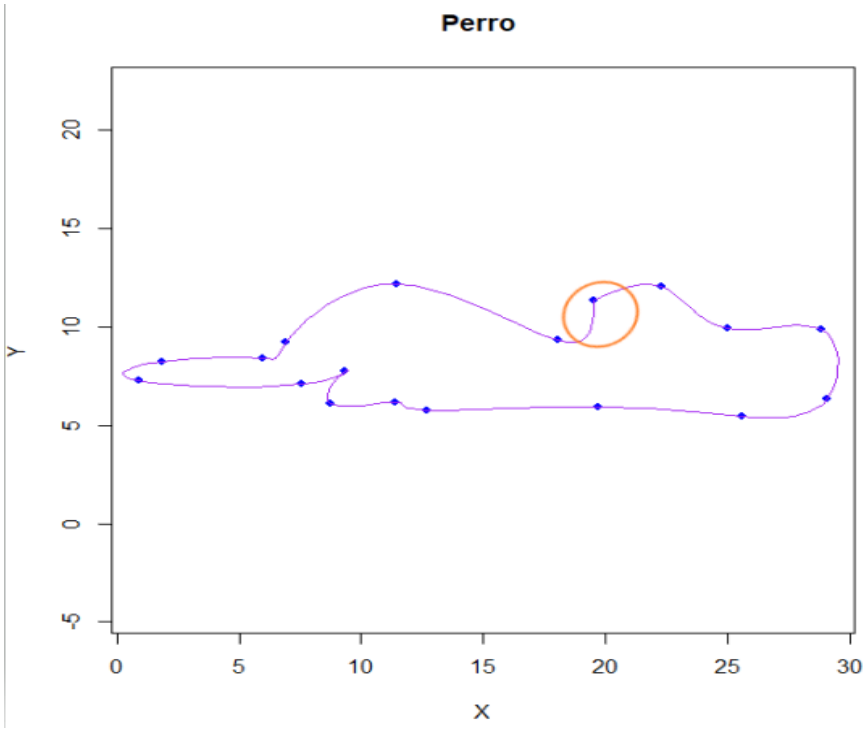


Diagrama del perro con el origen modificado

Segunda prueba:

x=c(12.69,11.37,8.74,9.32,7.58,0.92,1.82,5.94,6.93,11.45,18.03,19.52,22.31,24.98,28.81,29.06,25.61,19.68,12.69)

y=c(5.77,6.18,6.10,7.74,7.08,7.25,8.24,8.40,9.22,12.19,9.31,11.36,12.02,9.90,9.88,6.34,5.44,5.93,5.77)

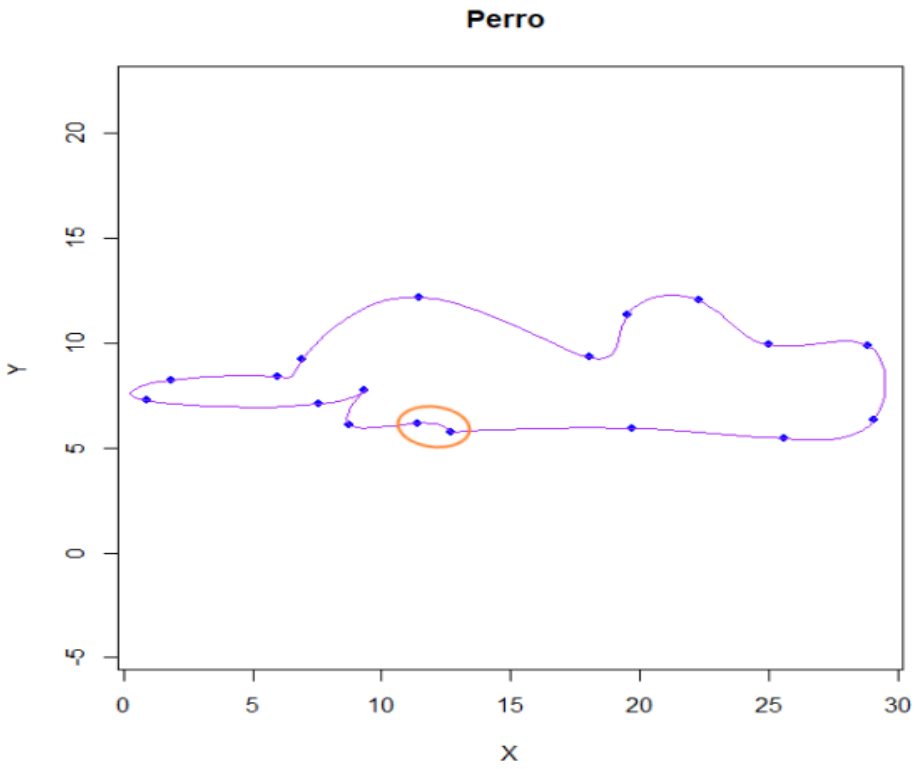


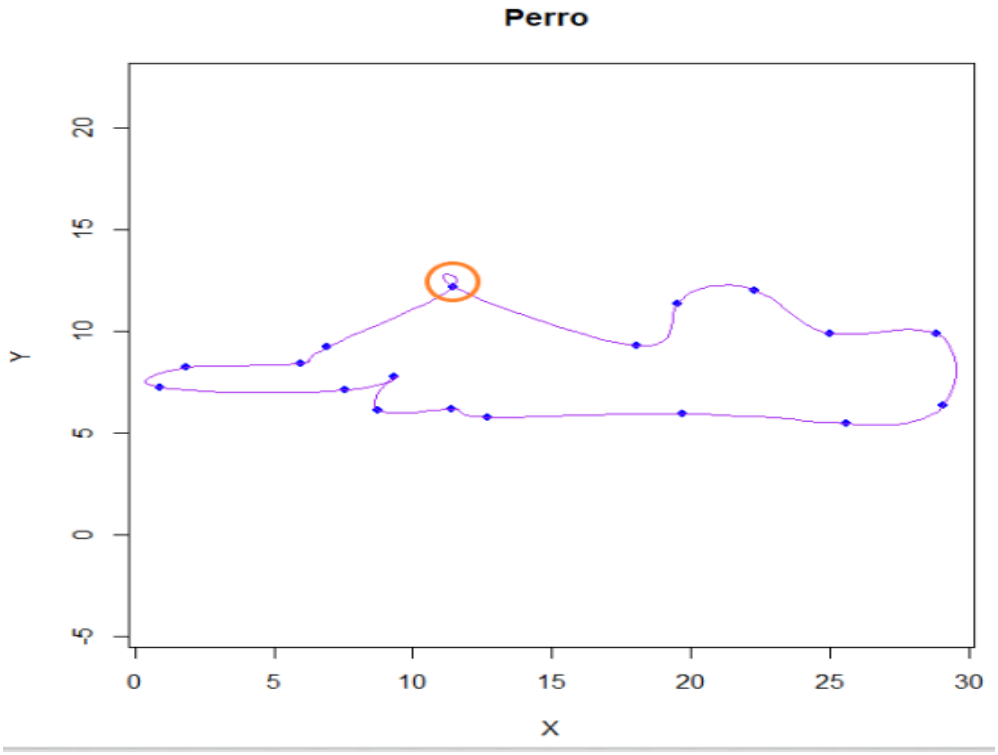
Diagrama del perro con origen modificado 2

Analizando las figuras obtenidas podemos concluir que el origen si se puede modificar, la figura original sufrirá algunas anomalías, pero la idea original del perro se mantiene

2. *¿Si tenemos nueva información ósea nodos como podemos implementar esa información en el algoritmo de interpolación?*

Para responder esta pregunta, realizamos varias pruebas, la primera consiste en modificar la figura insertando nuevos puntos, la primera prueba consistirá el insertar un punto muy cercano a uno existente, los datos del ensayo son:

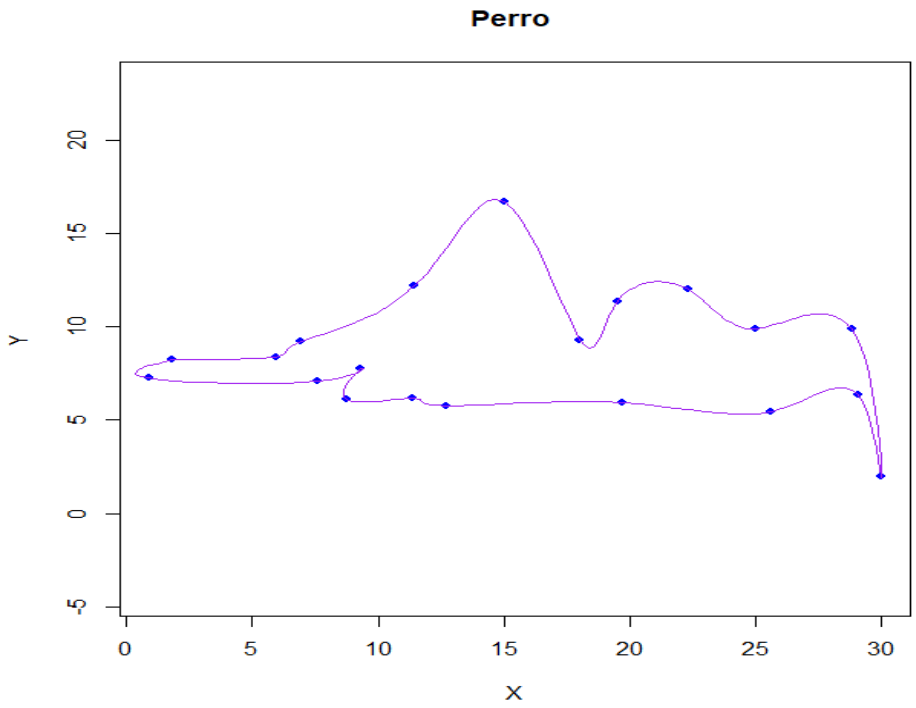
x=c(1.82,5.94,6.93,11.45,11.46,18.03,19.52,22.31,24.98,28.81,29.06,25.61,19.68,12.69,11.37,8.74,9.32,7.58,0.92,1.82)
y=c(8.24,8.40,9.22,12.19,12.20,9.31,11.36,12.02,9.90,9.88,6.34,5.44,5.93,5.77,6.18,6.10,7.74,7.08,7.25,8.24)



Como podemos observar en la figura anterior, al ingresar nuevos puntos muy cercanos a los ya existentes, la figura original presenta anomalías, ahora probaremos con puntos muy lejanos a los existentes, los datos del ensayo son:

x=c(1.82,5.94,6.93,11.45,15.0,18.03,19.52,22.31,24.98,28.81,30,29.06,25.61,19.68,12.69,11.37,8.74,9.32,7.58,0.92,1.82)

y=c(8.24,8.40,9.22,12.19,16.7,9.31,11.36,12.02,9.90,9.88,2,6.34,5.44,5.93,5.77,6.18,6.10,7.74,7.08,7.25,8.24)

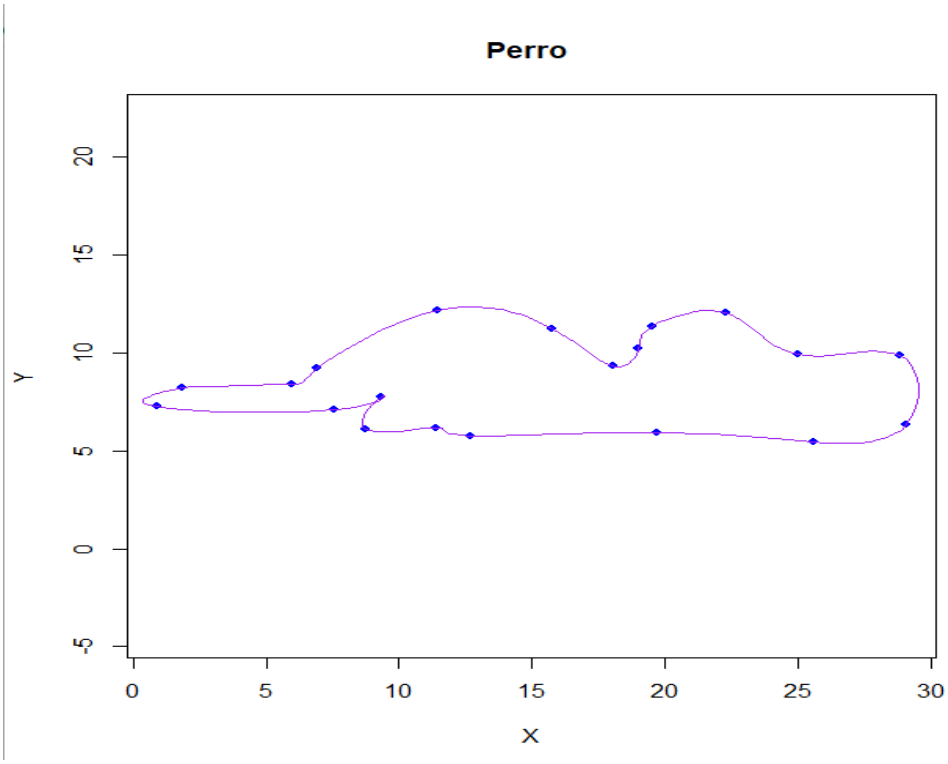


Esta fue la figura obtenida, podemos analizar que al insertar puntos muy distantes la figura pierde su sentido lógico ya que estos puntos no siguen el patrón de los demás puntos para formar la figura del perro.

Nuestro último ensayo consiste en probar puntos no muy cercanos ni muy lejanos a los originales, estos puntos hacen parte de la figura original del perro, los datos del ensayo son:

x=c(1.82,5.94,6.93,11.45,15.74,18.03,19,19.52,22.31,24.98,28.81,29.06,25.61,19.68,12.69, 11.37,8.74,9.32,7.58,0.92,1.82)

y=c(8.24,8.40,9.22,12.19,11.25,9.31,10.20,11.36,12.02,9.90,9.88,6.34,5.44,5.93,5.77,6.18, 6.10,7.74,7.08,7.25,8.24)



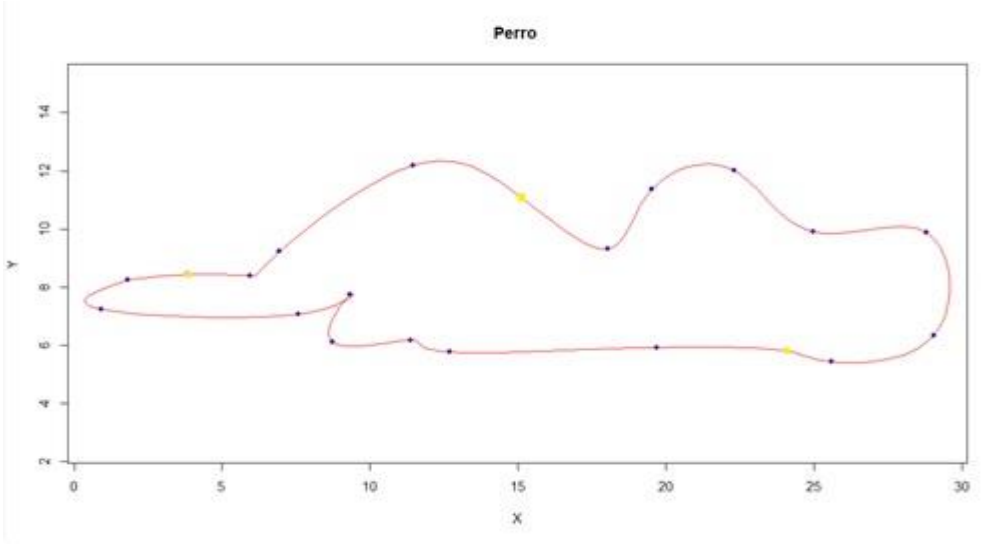
Observando nuestra última gráfica, observamos que esta tiene una aproximación mayor a la imagen del perro, debido a que poseemos más puntos y estos hacen parte del conjunto de puntos que conforman la línea que describe el contorno del perro. Podemos concluir que, si se pueden agregar nuevos puntos, siempre y cuando estos estén a una distancia prudente unos de otros y sigan el patrón lógico de los puntos que conforman la figura del perro

3. *¿Su método es robusto, en el sentido que si se tienen más puntos la exactitud no disminuye?*

La exactitud no disminuiría si se tienen más puntos que antes, solo decaería la exactitud si los puntos no estuvieran en orden, las abscisas no estuvieran de menor a mayor, por ejemplo, otra forma la cual decaería la exactitud seria si utilizamos un único polinomio que pase exactamente por dichos puntos, lo que puede ocurrir es que al representar dicha trayectoria se observen oscilaciones indeseadas.

Por ejemplo, si le añadimos esas coordenadas.

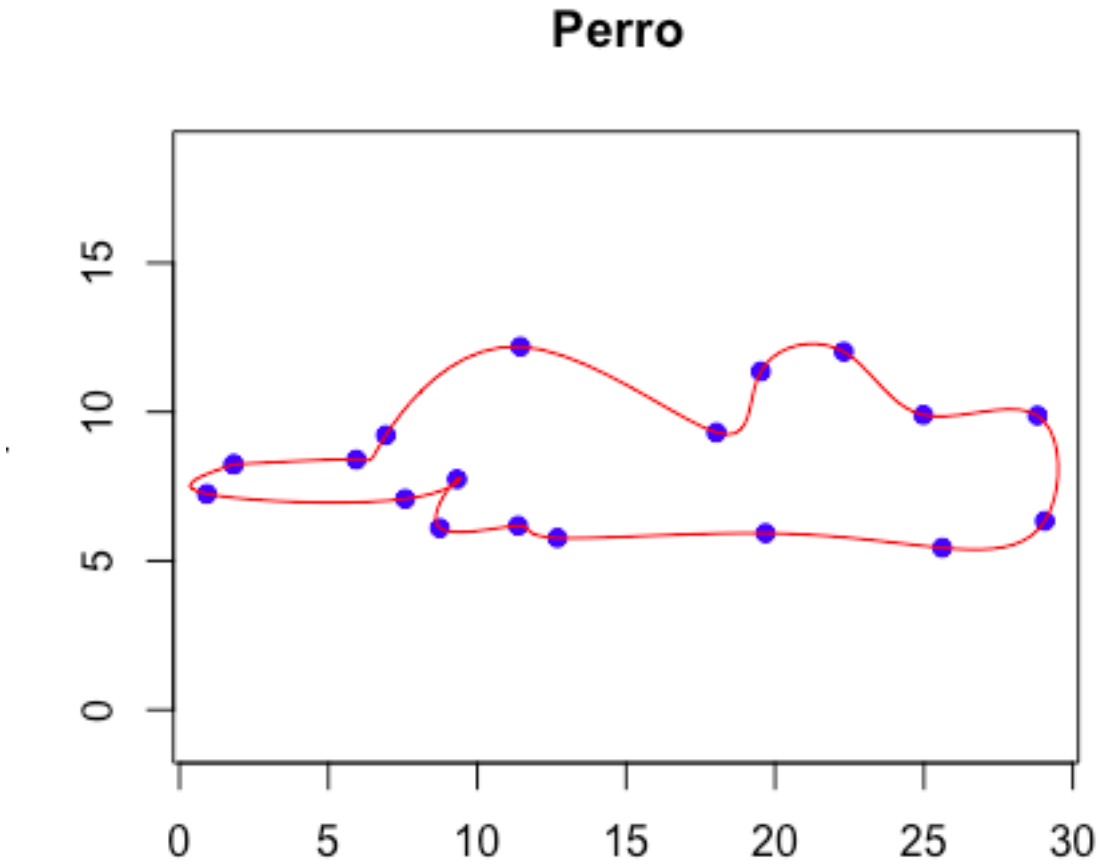
x	y
3.83	8.44
24.11	5.80
11.10	15.10



Como se puede observar en la imagen los nuevos puntos están marcados con color amarillo. Se puede notar como la precisión aumenta debido a la nueva información añadida, eso quiere decir que el método utilizado es un método robusto.

4. *¿Suponga que tiene más puntos con más cifras significativas como se comporta su algoritmo? la exactitud decae?*

La exactitud no decaería, al contrario, la exactitud aumentaría debido a que la ubicación de los puntos en el grafica fueran más precisas por lo que la imagen del perro podría ser aún más parecida. Si se tienen aún más puntos puede mejorar la exactitud de la interpolación si se sabe bien dónde ubicar los puntos de forma estratégica en el Spline que le corresponde, y si además los puntos tienen más cifras significativas podría mejorarse un poco más la exactitud. Si ahora utilizamos más cifras significativas tales como 5 así quedaría la imagen.
x=c(1.8167819,5.9345678,6.9278916,11.44567899,18.0299999,19.5154321,22.3012345,24.9765432,28.8099988,29.05678998,25.6055566,19.6765891,12.6812345,11.3689012,8.7345678,9.3156489,7.5789123,0.9123456,1.8189756)
y=c(8.242345,8.3912345,9.2123455,12.1845673,9.30456789,11.3578966,12.0145678,9.8998765,9.8786543,6.33456789,5.43567891,5.9234567,5.7612345,6.1798765,6.0987656,7.7345678,7.0765432,7.2467895,8.2381793)



Como se puede observar la exactitud no decae, la imagen del perro no presenta ninguna oscilación indeseada por el contrario mejora, aunque no sea muy notorio en la imagen se vuelve cada vez más preciso.

Anexo

- Pruebas realizadas antes de escoger el método de spline cubico:

Para la solución del problema del perro, nuestro grupo encontró conveniente usar una interpolación por el método de spline cúbico, esto debido a que se realizaron varias pruebas con el fin de reducir el número de puntos originalmente dados a través de tres métodos distintos, el primero que se usó fue el método de Lagrange baricéntrico, este método no nos daba una aproximación muy exacta a la figura real, con el mismo número de puntos generaba una versión muy modificada del perro (véase al final del párrafo). Por lo tanto, no optamos por este método ya que, además, uno de los criterios para la realización de este taller, era trabajar con el polinomio de menor grado posible, y encontramos que el método de la Lagrange baricéntrico maneja un polinomio muy alto igual al número de datos manejados menos uno (al final en el apartado de anexos se encontrará el resultado generado con este

método), otro método utilizado fue el de newton, pero a este no le hicimos muchas pruebas y no logramos avanzar mucho.

```
x=c(1,2,6,9,14,17.6,20,23,24.5,26.85,28.7,29,25.51,11.15,9.32,8.37,9.03,7.76,1)
y=c(3,3.7,4.5,7.12,6.7,4.45,7,6.5,5.6,5.87,5.05,3.71,0.47,1.65,1.22,1.7,2.92,2.36,3)
require(pracma)
plot(x,y, pch=19, cex=0.5, col = "red", asp=1,xlab="X", ylab="Y", main="Diagrama Perro ")
Graficar<-function(x0, xn){
  xi = x[x0:xn]
  yi = y[x0:xn]
  x <- seq(x[x0], x[xn], len=20)
  y <- barylag(xi, yi, x)
  lines(x, y, col="blue")
}
Graficar (1,4)
Graficar (4,6)
Graficar (6,7)
Graficar (7,9)
Graficar (9,12)
Graficar (12, 14)
Graficar (14, 16)
```

Código de la implementación con Lagrange baricentrico

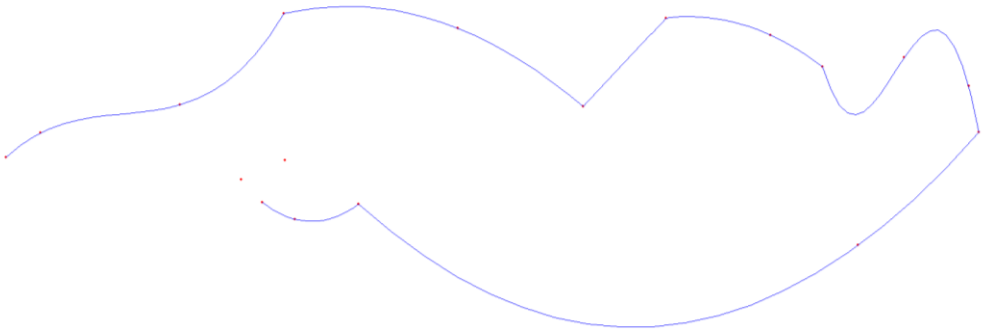


Imagen generada del perro por el método de Lagrange baricentrico

- Código de como se generaron los coeficientes del polinomio $s_J(x)$:

```
n=19
x=c(1,2,6,9,14,17.6,20,23,24.5,26.85,28.7,29,25.51,11.15,9.32,8.37,9.03,7.76,1)
a=c(3,3.7,4.5,7.12,6.7,4.45,7,6.5,5.6,5.87,5.05,3.71,0.47,1.65,1.22,1.7,2.92,2.36,3)
h=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
l=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
alpha=c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19)
z=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
b=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
c =c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
d=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)
mu=c(0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)

for(i in 1:n){
  h[c(i)]=x[c(i+1)]-x[c(i)]
}
aux1=((3/h[c(2)]))*(a[c(2+1)]-a[c(2)]))- ((3/h[c(2-1)])*(a[c(2)]-a[c(2-1)]))
for(i in 2:n){
  aux1=((3/h[c(i)]))*(a[c(i+1)]-a[c(i)]))- ((3/h[c(i-1)])*(a[c(i)]-a[c(i-1)]))
  alpha[c(i)]=aux1
}
l[c(1)]=1
mu[c(1)]=0
z[c(1)]=0
for(i in 2:n){
  l[c(i)]=  (2*(x[c(i+1)] - x[c(i-1)]))-(h[c(i-1)]*mu[c(i-1)])
  mu[c(i)]=h[c(i)]/l[c(i)]
  z[c(i)]=(alpha[c(i)]-(h[c(i-1)]*z[c(i-1)]))/l[c(i)]
}

l[c(n)]=1
z[(n)]=0
c[(n)]=0
j=n-1
while(j>0){
  c[c(j)]=z[c(j)]- (mu[c(j)]*c[c(j+1)])
  b[c(j)]=((a[c(j+1)]-a[c(j)])/h[c(j)])- (h[c(j)]*(c[c(j+1)]+2*c[c(j)]))/3
  d[c(j)]=(c[c(j+1)]-c[c(j)])/(3*h[c(j)])
  j=j-1;
  }
cat("x[]", "   " , "a[]" , "   " , "b[]" , "   ", "c[]" , "   ", "d[]", "\n")
val=0
for (i in 1:n){
  cat(i-1,",",x[c(i)], "," , a[c(i)] , "," , b[c(i)] , ",", c[c(i)] , ",", d[c(i)],"\n")
}
```

- Código de cómo se generaron los errores relativos con los valores originales e interpolados:

```
yx=c(3,3.7,3.9,4.5,5.7,6.69,7.12,6.7,4.45,7,6.1,5.6,5.87,5.15,4.1,4.3,4.1,3)
xx=c(1,2,5,6,7.5,8.1,10,13,17.6,20,23.5,24.5,25,26.5,27.5,28,29,30)
#error relativo valores interpolados
cat("x   ","y    ","s(x)   ","   error relativo")
inter = splinefun(x,y,method = "natural")
acumerrorrela=0
for(i in 2:n-1){
  valorinter = inter(xx[i])
  errabs=abs(yx[i]-valorinter)
  error = errabs/yx[i] * 100
  acumerrorrela=acumerrorrela+error
  cat(xx[i],",",yx[i],",",valorinter,",",error,"\n")
}
cat("error total:   ", acumerrorrela)
#error relativo valores originales
inter = splinefun(xx,yx,method = "natural")
for(i in 2:n-1){
  valorinter = inter(x[i])
  errabs=abs(y[i]-valorinter)
  error = errabs/y[i] * 100
  acumerrorrela=acumerrorrela+error
  cat(x[i],",",y[i],",",valorinter,",",error,"\n")
}
cat("error total:   ", acumerrorrela)
```

- Código de cómo se sacaron los valores no utilizados:

```
#valores no seleccionados
inter = splinefun(x,y,method = "natural")
i=0
cat(x[c(i)],"      ",xx[c(i)])
for(i in 2:n-1){
if(x[c(i)]!=xx[c(i)]){
  valorinter = inter(xx[i])
  cat(xx[i],",",yx[i],",",valorinter,"\n")
}
```

Bibliografía

[1] (Mora,Walter).2017. Introducción a los Métodos Numéricos. pg 203.