Documentación Segundo Reto

Gabriel Gómez, Juan Pablo Méndez y Simón Dávila

Abril 2020

Nota: Los código están programados en lenguaje R.

1 Curvas y Superficies de Bezier

Se utilizan las curvas de Bezier para interpolar movimiento cuando el objeto ehibe formas de movimiento curvilíneo.

1.1 Problema de Aproximación

Se implementó las curvas de Bezier para graficar el mortero valenciano.

Figure 1: Mortero Valenciano

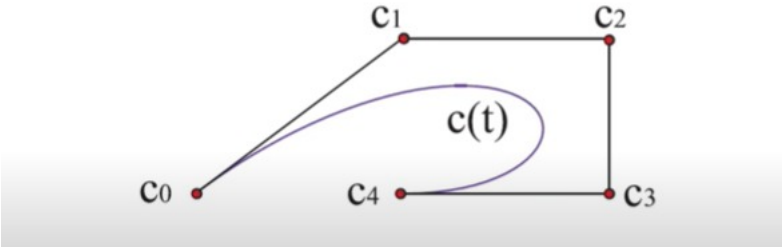
Para dar solución al problema planteado se utilizan las curvas de Bezier y los polinomios de Bernstein. Las curvas de Bezier son curvas polinómicas expresadas con polinomios de Bernstein. Dichos polinomios se utilizan para representar curvas polinómicas.

Para el desarrollo de los polinomios de Bernstein es de vital importancia tener en cuenta los coeficientes binomiales. Para ello contamos con la siguiente formula:

$$B_i^n(t) = \binom{n}{i} t^i (1-t)^{n-i}.$$

En cuánto a la curva de Bezier se debe tomar en cuenta los polinomios de Bernstein y los vértices del polígono de control, estos vértices corresponden a los puntos en el plano cartesiano. Para generar las curvas de Bezier se debe realizar la sumatoria de 0 hasta n particiones para cada vértice.

Representar curvas de grado n, $c(t) = \sum_{i=0}^{n} c_i B_i^n(t)$, $t \in [0,1]$, $c_i \in \mathbb{R}^p$, $\{c_0, \ldots, c_n\}$ **polígono de control**.



Teniendo en cuenta la teoría planteada se realizaron las siguientes funciones en R para el correcto funcionamiento de las curvas de Bezier.

1.1.1 Explicación del código

Implementación: Para la recolección de datos se utilizó la plataforma Tracker. Tomando como un punto de referencia la base del mortero, ubicandola en el origen del plano cartesiano.

x¡-c(0, 20.77, 37.92, 55.08, 73.14, 88.49, 113.8, 130.9, 148.1, 156.2, 162.5, 167.9, 173.4, 176.1, 177, 161.6, 142.7, 124.6, 102.9, 80.36, 49.66, 22.57, 0 , -26.18, -55.98, -78.55, -100.2, -128.2, -143.6, -159.8, -171.6, -176.1, -176.1, -173.4, -170.7, -164.3, -156.2, -144.5, -132.7, -113.8, -95.71, -78.55, -55.8, -28.89, 0 , 0, 23.48, 56.88 , 83.97, 118.3, 140.9, 158, 167, 176.1, -25.28, -49.66, -75.85, -98.42, -116.5, -137.2, -154.4, -170.7, -175.2, 0, 21.67, 46.95, 70.43, 92.10, 114.7, 127.3, 136.3, 128.2, 111.1, 87.58, 60.50, 31.60, 0, -24.38, -53.27, -77.65, -94.81, -114.7, -127.3, -136.3, -136.3, -120.1, -99.32, -80.36, -56.88, -35.21)

y¡-c(0, 3.612, 5.418, 9.932, 15.35, 22.57, 36.12, 50.56, 69.53, 85.78, 102, 126.4, 146.3, 179.7, 209.5, 237.5, 252.2, 270.9, 279.9, 288.9, 294.4, 299.8, 300, 298, 293.5, 287.1, 281.7, 271.8, 260, 246.5, 230.2, 205.9, 178.8, 146.3, 128.2, 108.4,

88.49, 70.43, 55.98, 37.02, 22.57, 16.25, 9.029, 4.515, 0,120. 1, 117.4, 118.3, 127.3, 139.1, 152.6, 164.3, 179.7, 200, 116.5, 121.9, 127.3, 133.6, 144.5, 156.2, 170.7, 186, 199.5, 140.9, 141.8, 144.5, 148.1, 155.3, 170.7 , 180.6, 200, 227.5, 238.4, 253.7, 265.5, 273.6, 275.4, 141.8, 146.3, 152.6, 160.7, 173.4, 186, 200,223, 239.3, 251, 260, 266.4, 269.1)

plot(x,y,xlim=c(-300,300),ylim =c(0,300),xlab = "x", ylab = "y",main = "Mortero puntos Tracker")

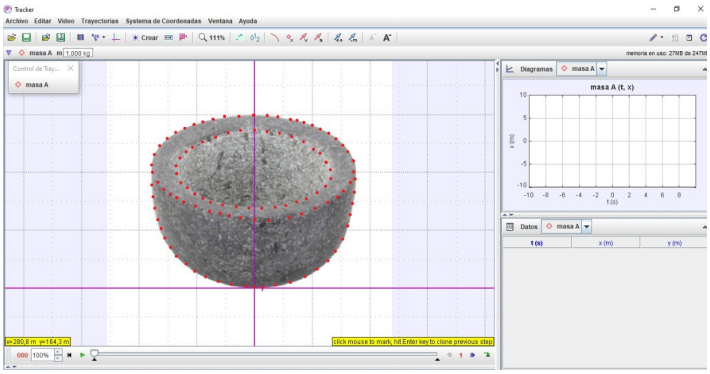


Figure 2: Mortero en plataforma Tracker

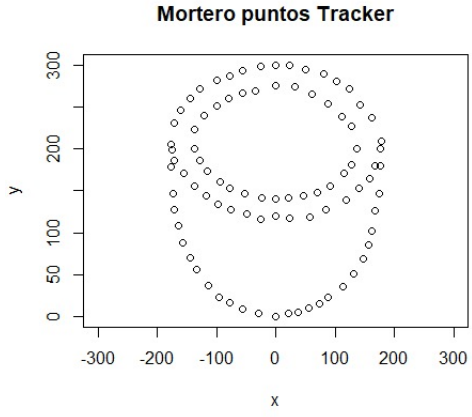


Figure 3: Mortero en graficado en R

Punto	Coordenadas en X	Coordenadas en Y	Punto	Coordenadas en X	Coordenadas en Y	Punto	Coordenadas en X	Coordenadas en Y
1	0	0	31	-171.6	230.2	61	-154.4	170.7
2	20.77	3.612	32	-176.1	205.9	62	-170.7	186
3	37.92	5.418	33	-176.1	178.8	63	-175.2	199.5
4	55.08	9.932	34	-173.4	146.3	64	0	140.9
5	73.14	15.35	35	-170.7	128.2	65	21.67	141.8
6	88.49	22.57	36	-164.3	108.4	66	46.95	144.5
7	113.8	36.12	37	-156.2	88.49	67	70.43	148.1
8	130.9	50.56	38	-144.5	70.43	68	92.10	155.3
9	148.1	69.53	39	-132.7	55.98	69	114.7	170.7
10	156.2	85.78	40	-113.8	37.02	70	127.3	180.6
11	162.5	102	41	-95.71	22.57	71	136.3	200
12	167.9	126.4	42	-78.55	16.25	72	128.2	227.5
13	173.4	146.3	43	-55.8	9.029	73	111.1	238.4
14	176.1	179.7	44	-28.89	4.515	74	87.58	253.7
15	177	209.5	45	0	0	75	60.50	265.5
16	161.6	237.5	46	0	120.1	76	31.60	273.6
17	142.7	252.2	47	23.48	117.4	77	0	275.4
18	124.6	270.9	48	56.88	118.3	78	-24.38	141.8
19	102.9	279.9	49	83.97	127.3	79	-53.27	146.3
20	80.36	288.9	50	118.3	139.1	80	-77.65	152.6
21	49.66	294.4	51	140.9	152.6	81	-94.81	160.7
22	22.57	299.8	52	158	164.3	82	-114.7	173.4
23	0	300	53	167	179.7	83	-127.3	186
24	-26.18	298	54	176.1	200	84	-136.3	200
25	-55.98	293.5	55	-25.28	116.5	85	-136.3	223
26	-78.55	287.1	56	-49.66	121.9	86	-120.1	239.3
27	-100.2	281.7	57	-75.85	127.3	87	-99.32	251
28	-128.2	271.8	58	-98.42	133.6	88	-80.36	260
29	-143.6	260	59	-116.5	144.5	89	-56.88	266.4
30	-159.8	246.5	60	-137.2	156.2	90	-35.21	269.1

Figure 4: Tabla de puntos graficados

La función polinomiosBernstein tiene como objetivo hallar los coeficientes binomiales para multiplicarlos con los vértices del polígono de control, tal cuál como la fórumla lo indica.

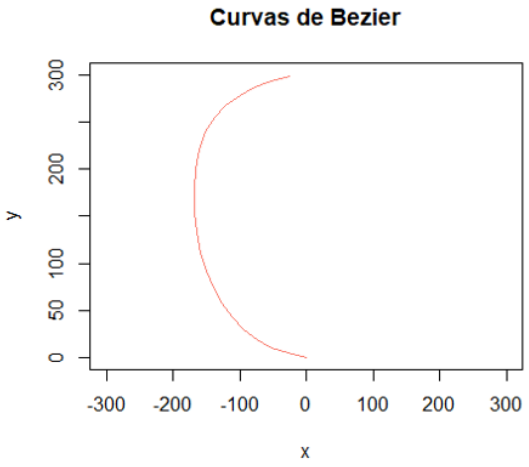
```
polinomios_Bernstein <- function(cord_x, cord_y, t)
{
  x1 = 0
  y1 = 0
  #se le resta 1 al tamaño del vector de coordenadas x
  #porque los polinomios de Bernstein demuestran que
  #una sucesión de puntos converge uniformemente a X ([X,Y])
  n = length(cord_x)-1
  for (i in 0:n)
  {
    # Se utiliza la formula de combinatoria para generar la curva
    # de Bezier tomando como referencia los vertices del poligono  de control
    x1 = x1 + choose(n, i)*((1-t)^(n-i))*t^i*cord_x[i+1]
    y1 = y1 + choose(n, i)*((1-t)^(n-i))*t^i*cord_y[i+1]
  }
  return (list(x=x1, y=y1))
}
```

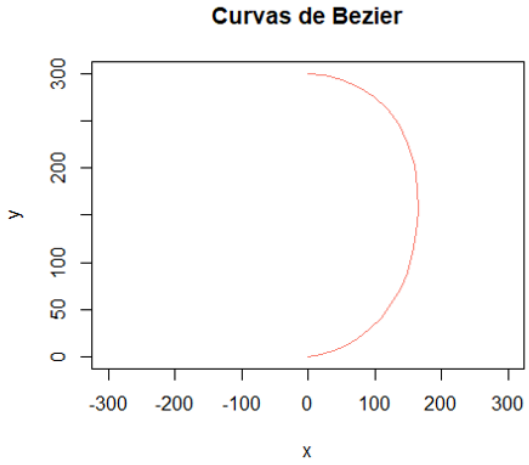
La función curvabezier tiene como objetivo recibir las coordenas resultantes de la sumatoria de los coeficientes binomiales, para esto realizamos n particiones. Tomando n como el tamaño del vector de las coordenadas x. Por último retorna una lista con coordenadas (x,y) para poder graficar el mortero utilizando las curvas de Bezier.

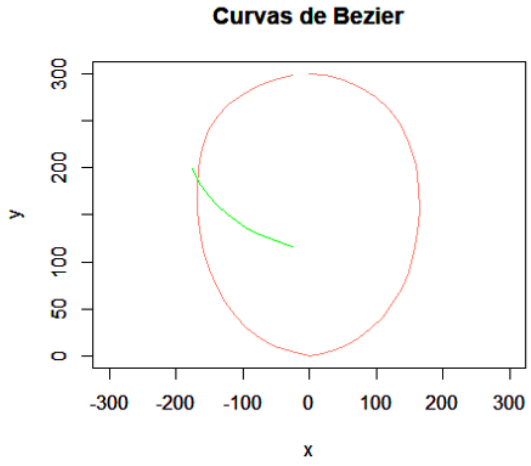
```
curva_bezier<- function(cord_x, cord_y, n)
{
  #se usa NULL para asegurarse que la lista no tenga ningun valor antes de inciar
  x1 = NULL
  y1 = NULL
  cont= 1
  #el for se realiza en una secuencia de 0-1 en n particiones dependiendo del tamano del vector
  for (t in seq(0, 1, length.out = n))
  {
    # se hace el llamado a la funcion bezier
    pivote = funcion_bez(cord_x,cord_y, t)
    # se accede al elemento x de la lista retornada
    x1[cont] = pivote[1]
    # se accede al elemento y de la lista retornada
    y1[cont] = pivote[2]
    cont=cont+1
  }
  return (list(x=x1, y=y1))
}
```

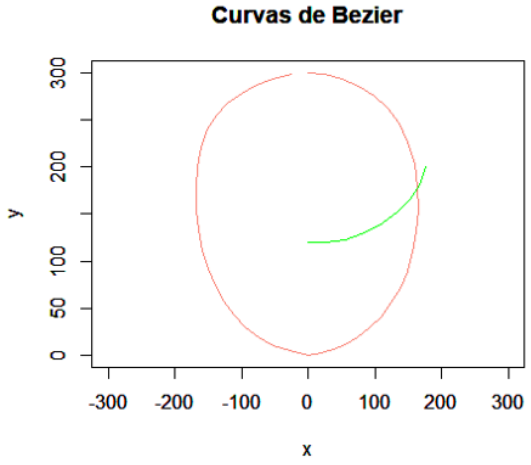
1.2 Resultados

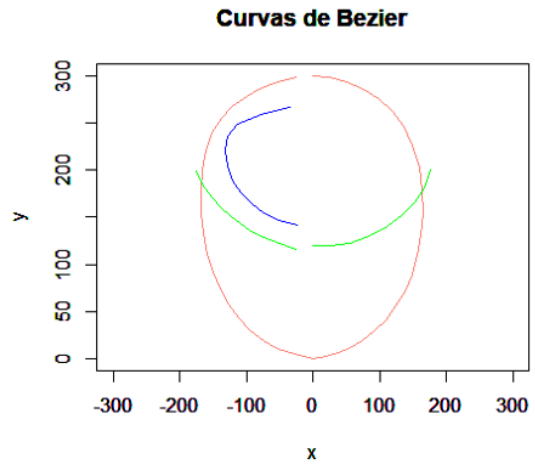
Para graficar el mortero final, tomamos cada una de las secciones con sus respectivos vectores de sus coordenadas x,y pasandolos como argumentos a la función curvasbezier y como último argumento el tamaño del vector de las coordenadas x. Después se utilizó la función predetrminada de R points() pasando como argumento el llamado a la función curvasbezier la cual retorna la secuencia con las coordenadas x,y para graficar adecuadamente.

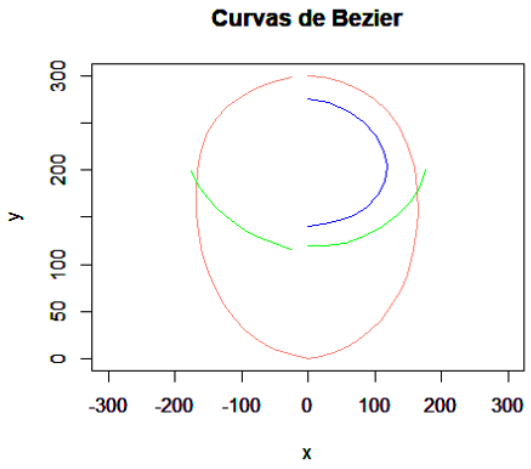


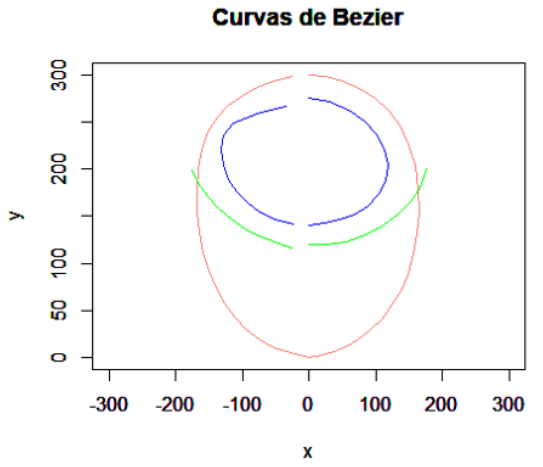


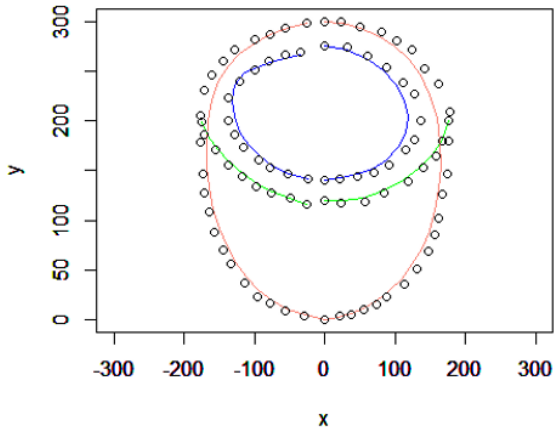

Curvas de Bezier
y
x
300
200
100
50
0
-300
-200
-100
0
100
200
300


Curvas de Bezier
y
x
300
200
100
50
0
-300
-200
-100
0
100
200
300


Curvas de Bezier
y
x
300
200
100
50
0
-300
-200
-100
0
100
200
300


Curvas de Bezier
y
x
300
200
100
50
0
-300
-200
-100
0
100
200
300





2 Conclusión

Las curvas de Bezier nos brindan la posibilidad de observar el comportamiento de una serie de puntos, ya que tenemos la oportunidad de ajustar las particiones para mayor exactitud de las curvas. Para generar estas curvas los polinomios de Bernstein son una herramienta fundamental para el adecuado desarrollo de estas curvas. Por último, se puede evidenciar que estas curvas serán más aproximadas a los vértices del polígono de control dependiendo de la cantida de puntos que se tomen.

3 Referencias

[1] http://fernandorevilla.es/blog/2018/04/12/polinomios-de-bernstein/
[2] http://geogebra.es/gauss/materiales$_d$*idacticos/bach/actividades/geometria/geo*$_2$*D/bezier/actividad.html*
[3]*https* : *//www.youtube.com/watch?v* = *uZI*4*bY*4*uULo*
[4]*https* : *//polipapers.upv.es/index.php/MSEL/article/viewFile*/3071/3162