Reto 2 Análisis Númerico

Edwin Turizo
Juan Pimienta

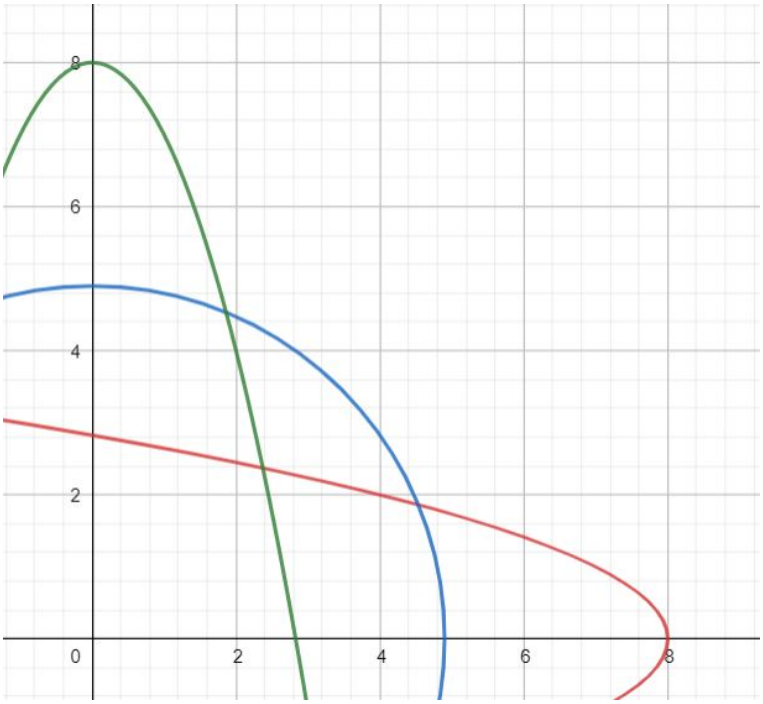
19 de Abril del 2020

1. Superficies de Bézier

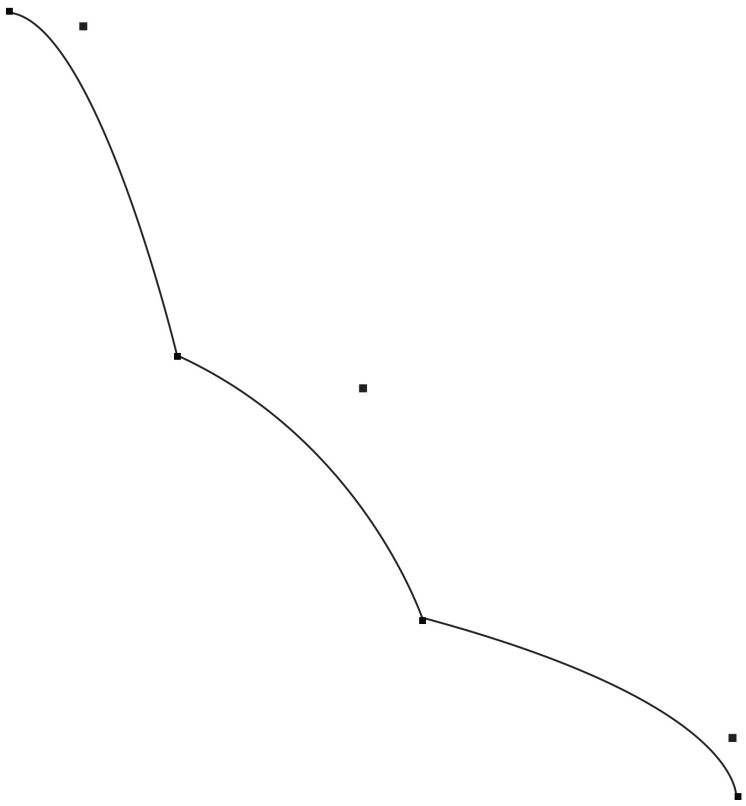
1.1. Puntos de control

Dibujar el mortero valenciano usando superficies de Bezier y otro metodo (BSplines). Para ello se puede utilizar R(PathInterpolatR, gridBezier,vwline) o Python(griddata, matplotlib).

Para realizar el reto de interpolación utilizando el metodo de las superficies de Bezier, primero se dividió la figura en 4 partes iguales (tal como se sugería). Para ello, se graficó un cuarto (1/4) del mortero valenciano desde una vista superior, como se muestra a continuación:



Posteriormente se utilizó Adobe Illustrator para hacer un contorno de la figura para más adelante identificar los puntos de control, que quedaron asignados de la siguiente manera:



Luego con herramientas de Adobe Illustrator primero se midieron los puntos de control previamente asignados, los cuales arrojaron los siguientes resultados (se utilizó una z por defecto de 2, pero con cualquier valor en las z se obtenian unos resultados bastante similares).

X(UN)	Y(UN)
0	8
1,7	7,5
2	4,5
4	4
4,8	1,8
8	0,6
8	0

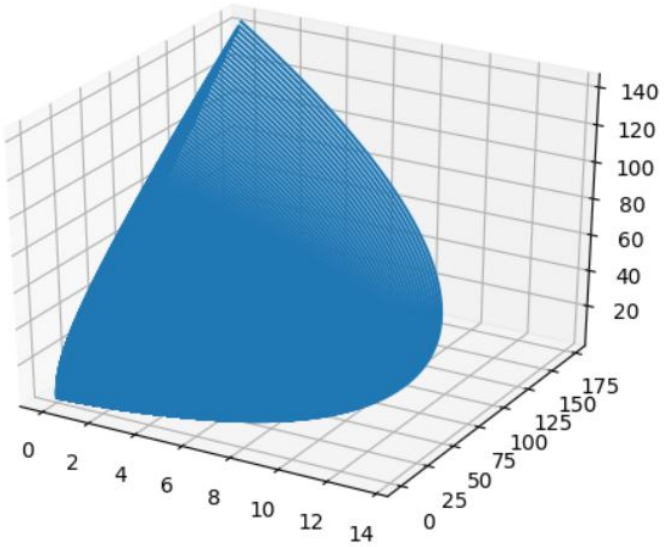
1.2. Implementación del código en Python

$$\alpha(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(t)B_j^m(t)P_{i,}; \forall u,v \in [0,1]$$

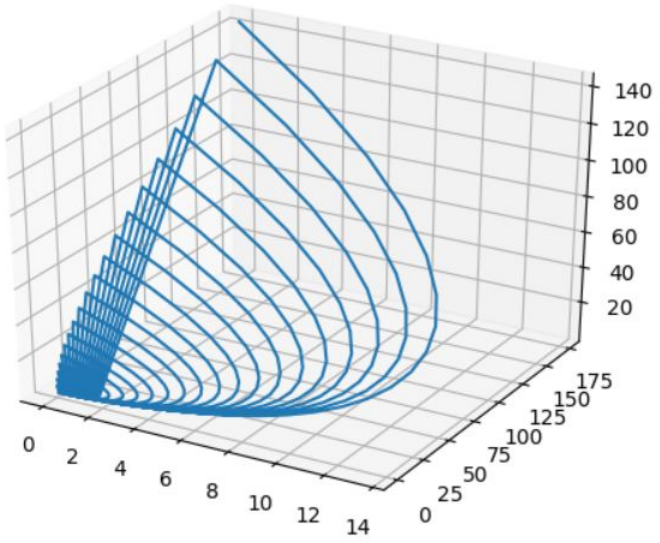
Para la implementación de la ecuación de la superficie de Bézier se utilizó el siguiente algoritmo.

```
def superfBezier(m,n):
  arreglo=np.arange(0,1,0.005)
  arregloV=np.arange(0,1,0.005)
  for u in arreglo:
    for v in arregloV:
      x=y=z=0
      for i in range(len(n)):
        for j in range(len(m)):
          x=x+bernstein(len(n),i,u)*bernstein(len(m),j,v)*m[j][0]*n[i][0]
          y=y+bernstein(len(n),i,u)*bernstein(len(m),j,v)*m[j][1]*n[i][1]
          z=z+bernstein(len(n),i,u)*bernstein(len(m),j,v)*m[j][2]*n[i][2]
      puntosX.append(x)
      puntosY.append(y)
      puntosZ.append(z)
  return puntosX,puntosY,puntosZ
```

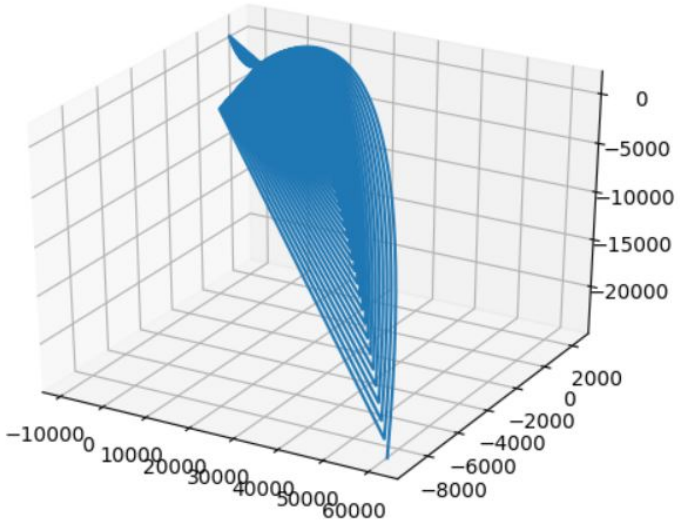
Primero dividimos los puntos de control en 2 grupos para satisfacer la ecuacion.
Guardamos cada coordenada en 3 arreglos llamados, puntosX, puntosY y puntosZ, allí se añadía un punto nuevo cada vez que se cambiaba el valor de v en el ciclo de este mismo.
Para la ejecución del algoritmo, y después de probar con varios u y v posibles, consideramos que los valores aproximadamente mejores para estas variables era de 0,005 en 0,005 hasta 1.
Observamos que el resultado puede cambiar de manera importante de acuerdo a la variación de estas dos variables.
Se observaron los siguientes resultados.



Ya si cambiabamos la forma de variar de las u y v por ejemplo, de 0,05 en 0,05 se observaba el siguiente resultado.



En cambio, si exagerabamos un poco más el número de iteraciones, se seguía obteniendo un resultado no deseado, como el que se aprecia a continuación.



Ya volviendo al tema principal, que es la ecuación de las superficies de Bezier, se puede observar que se utiliza el Polonomio de Bernstein:

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

Este polinomio se implemento en Python con el siguiente código:

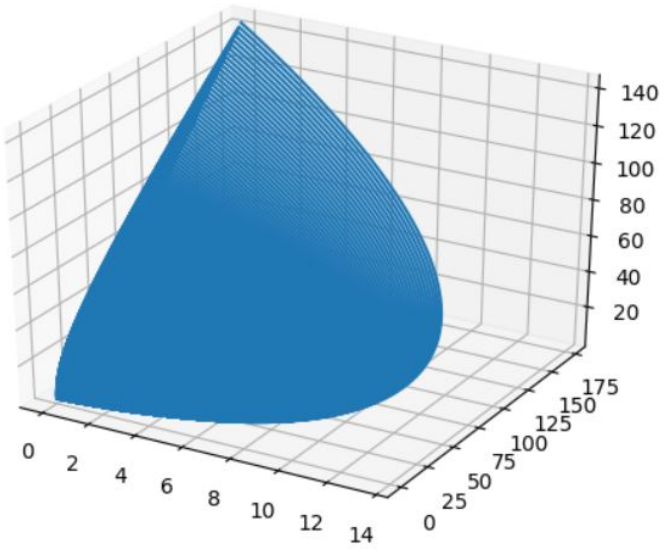
```
23 def bernstein(n,i,t):
24     return combinatoria(n,i)* t**i * (1-t)**(n-i)
```

Adicionalmente, se puede observar que se utiliza una funcion que calcula la combinatoria, que a su vez necesita una función que calcule el factorial de un número dado, la implementación de estas dos funciones se puede observar a continuación.

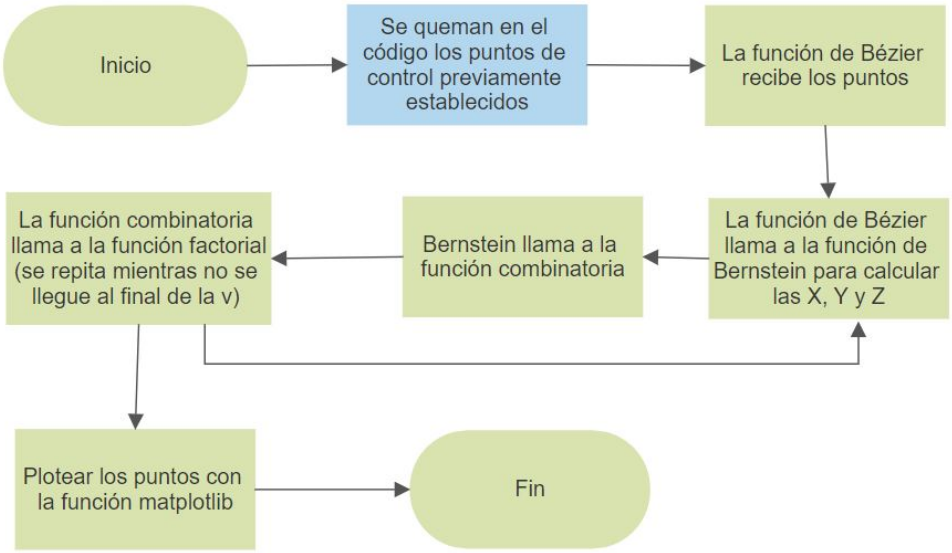
```
16 def combinatoria (n,i):
17     if(i>=0 and i<=n):
18         return (factorial(n)/(factorial(i)*factorial(n-1)))
```

```
10 def factorial(num):
11     if(num==0):
12         return 1
13     else:
14         return num* factorial(num-1)
```

Finalmente con la libreria Matplotlib de Python, se dibujaron las figuras previamente observadas, esta librería permitió el proceso de graficar tanto en 2D como en 3D, para realizar las pruebas de los algoritmos implementados. Matplotlib permite la rotación y traslación de la figura y sus ejes cartesianos, lo que facilita al interpretación de los resultados y la correcta verificación de los procesos desarrollados.
Finalmente nuestra mayor aproximación al 1/4 de mortero valenciano, es la que se muestra en la siguiente figura:



2. Diagrama de flujo



3. Conclusiones

- Las curvas de bezier son una forma de establecer una union entre puntos sin la necesidad de realizar un simple trazado recto, utilizando polinomios de mayores grados y formando así trayectorias curvas, esto se logra interpolando uno o dos puntos de control que son los que van dando la forma deseada de la figura.
- Para el desarrollo de las curvas de bezier es muy importante establecer los coordenadas de entrada, coordenadas intermedias y coordenadas de salida para que se logre una correcta grafica y no se vean deformaciones en cada sector o cuadrante del plano, esto dependiendo claramente del modelo que se desea dibujar.
- El proceso aplicado en este reto dio a conocer cuales son los parametros que se necesitan y como se deben aplicar mostrando que existen otros metodos al graficar superficies o solidos, y aunque se tiene una complejidad, con ciertas tecnicas matematicas y estadisticas se puede lograr adecuadamente.