

# Mortero valenciano

Construido en base a curvas de Bezier

- Fabián A. Pallares Jaimes
- Juan Sebastián Valderrama Urquijo
- Luis Santiago Jaramillo Espinosa

---

## Paraboloide elíptico

Para crear la base del mortero, graficamos un paraboloide elíptico de revolución con la función `ContourPlot3D`, la cual recibe como parámetros la función a graficar y los intervalos de gráfica para cada uno de los ejes. En nuestro caso decidimos tomar los ejes:

- $x \{0, 1\}$
- $y \{-1, 1\}$
- $z \{0.4, 1\}$

Para lograr que  $z$  comenzara en dicha posición, se implementó la función `ClipPlanes`, la cual aplica un corte a partir de la función que recibe por parámetro, en este caso `InfinitePlane`, función que a su vez traza un plano desde los puntos indicados. Para este caso, `infinite plane` recibe todos sus parámetros de  $z$  en 0.4.

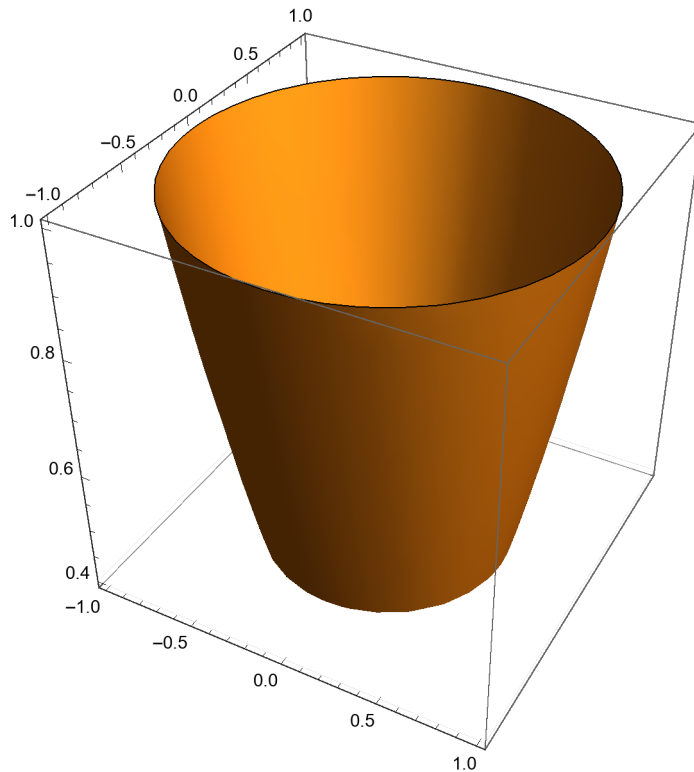
La función `ClipPlanesStyle` configura el estilo del corte, que en este caso pusimos de color blanco para no interferir con la figura objetivo.

Finalmente la función `Mesh -> None` nos permite graficar sin las mallas comunes de Wolfram, dándonos una superficie lisa y mejor presentada.

In[355]:=

```
par = ContourPlot3D[x^2 + y^2 - z == 0, {x, -1, 1}, {y, -1, 1}, {z, 0.4, 1},
ClipPlanes -> InfinitePlane[{{1, 0, 0.4}, {1, 1, 0.4}, {0, 0, 0.4}}],
ClipPlanesStyle -> Directive[Opacity[0.3], White], Mesh -> None]
```

Out[355]=



## Tapa

Para hacer la tapa inferior del mortero, se usó la función `ParametricPlot3D`, con la cual desarrollamos un disco dibujado en  $z = 0.4$ .

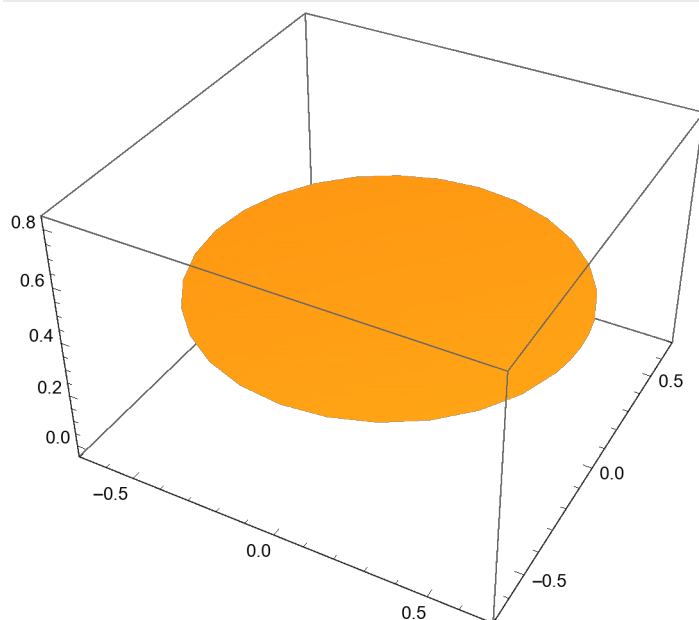
■ Importante para el gráfico del disco:

- Es importante conocer el radio del disco para que se acople en la función del paraboloides cortado en  $z = 0.4$ . Para esto, configuramos el radio entre 0 y la raíz cuadrada de 0.4, valor que fue hallado como radio con las formulas de cálculo correspondientes.
- Uno de los conflictos que tuvimos al desarrollar el disco, fue el hacer que se dibujara en 0.4 y no en cero (como es por defecto en Wolfram). Para esto, finalmente configuramos la segunda expresión que pasamos por parametro a `ParametricPlot3D`, tomando el valor de esta como 0.4.

In[356]:=

```
circ = ParametricPlot3D[{r*Sin[Pi/2]*Cos[phi], r*Sin[Pi/2]*Sin[phi], 0.4},  
{r, 0, Sqrt[0.4]}, {phi, 0, 2*Pi}, Mesh -> None]
```

Out[356]=

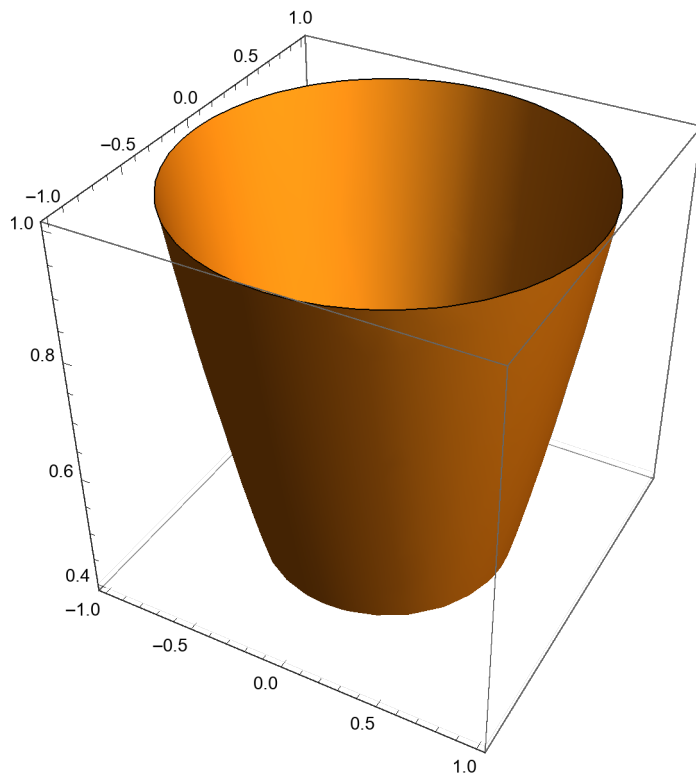


En la siguiente línea de código unimos la tapa con el paraboloides para comprobar que se acoplen sin problema [Las figuras presentadas se pueden mover arrastrando con el mouse].

In[357]:=

`Show[par, circ]`

Out[357]=



## Puntos y aletas

Para realizar las aletas, tomamos una primera como base, cuyos puntos luego se replicarían para las siguientes 3.

Para la toma de puntos tuvimos en cuenta los dos de inicio y final, y los dos de control necesarios para las curvas de Bezier. Así pues, se organizaron en cuatro grupos de cuatro puntos para cada una de las aletas, formando las curvas necesarias.

Al momento de replicar las curvas a los otros 3 lados de la figura, se tuvo en cuenta cambiar los puntos de los ejes x y y multiplicar por -1 ambos ejes teniendo en cuenta la grafica que quisieramos obtener.

In[358]:=

```
pts = {{{-0.5, 0.88, 1}, {0, 1.4, 1}, {0, 1.4, 1}, {0.5, 0.88, 1}},
{{-0.45, 0.8, 0.85}, {-0.2, 1.2, 0.85}, {0.2, 1.2, 0.85}, {0.45, 0.8, 0.85}},
{{-0.45, 0.75, 0.7}, {-0.2, 1.15, 0.7}, {0.2, 1.15, 0.7}, {0.45, 0.75, 0.7}},
{{-0.15, 0.76, 0.6}, {-0.08, 0.79, 0.62}, {0.08, 0.79, 0.62}, {0.15, 0.76, 0.6}}};

ptsdos = {{{-0.5, -0.88, 1}, {0, -1.4, 1}, {0, -1.4, 1}, {0.5, -0.88, 1}},
{{-0.45, -0.8, 0.85}, {-0.2, -1.2, 0.85}, {0.2, -1.2, 0.85}, {0.45, -0.8, 0.85}},
{{-0.45, -0.75, 0.7}, {-0.2, -1.15, 0.7}, {0.2, -1.15, 0.7}, {0.45, -0.75, 0.7}},
{{-0.15, -0.76, 0.6}, {-0.08, -0.79, 0.62}, {0.08, -0.79, 0.62}, {0.15, -0.76, 0.6}}};

ptstres = {{{0.88, -0.5, 1}, {1.4, 0, 1}, {1.4, 0, 1}, {0.88, 0.5, 1}},
{{0.8, -0.45, 0.85}, {1.2, -0.2, 0.85}, {1.2, 0.2, 0.85}, {0.8, 0.45, 0.85}},
{{0.75, -0.45, 0.7}, {1.15, -0.2, 0.7}, {1.15, 0.2, 0.7}, {0.75, 0.45, 0.7}},
{{0.76, -0.15, 0.6}, {0.79, -0.08, 0.62}, {0.79, 0.08, 0.62}, {0.76, 0.15, 0.6}}};


ptscuatro = {{{-0.88, -0.5, 1}, {-1.4, 0, 1}, {-1.4, 0, 1}, {-0.88, 0.5, 1}},
{{-0.8, -0.45, 0.85}, {-1.2, -0.2, 0.85}, {-1.2, 0.2, 0.85}, {-0.8, 0.45, 0.85}},
{{-0.75, -0.45, 0.7}, {-1.15, -0.2, 0.7}, {-1.15, 0.2, 0.7}, {-0.75, 0.45, 0.7}},
{{-0.76, -0.15, 0.6}, {-0.79, -0.08, 0.62}, {-0.79, 0.08, 0.62}, {-0.76, 0.15, 0.6}}};
```

## Aplicación de curvas de bezier para la formación de las aletas

Para obtener las curvas mencionadas en el punto anterior, usamos la función BezierFunction, la cual, con una serie de puntos múltiples de 4 (16 en este caso), crea una función paramétrica que se acopla a los inicios, finales, y se ve modificada por los puntos de control.

En los outputs de las funciones podemos ver que crea cada una de las funciones de Bezier y nos muestra la cantidad de puntos y los puntos usados.

```
In[362]:= f = BezierFunction[pts]
          fdos = BezierFunction[ptsdos]
          ftres = BezierFunction[ptstres]
          fcuatro = BezierFunction[ptscuatro]
```

```
Out[362]= BezierFunction[ Argument count: 2  
Output dimension: 3  
Degree: {3, 3}  
Control points: 16  
{-0.5, 0.88, 1.}, {0., 1.4, 1.}, {0., 1.4, 1.}, {0.5, 0.88, 1.}, ...]
```

```
Out[363]= BezierFunction[ Argument count: 2  
Output dimension: 3]
```

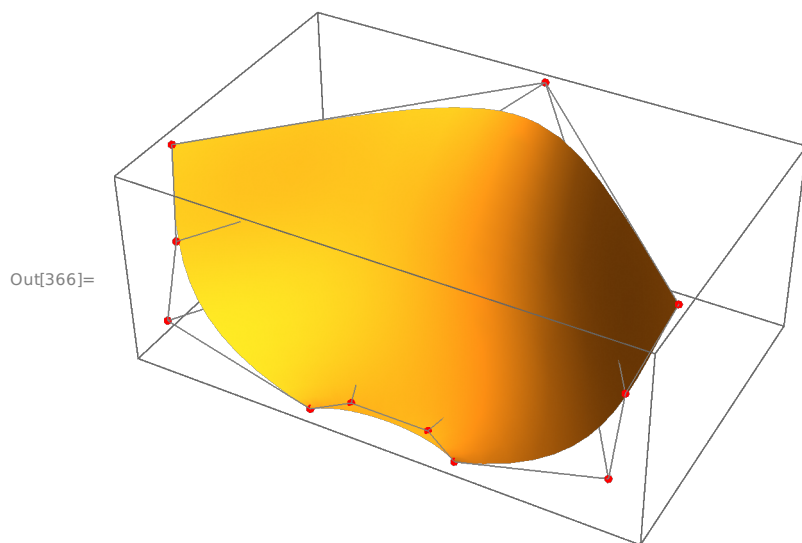
```
Out[364]= BezierFunction[ Argument count: 2  
Output dimension: 3]
```

```
Out[365]= BezierFunction[ Argument count: 2  
Output dimension: 3]
```

## Aplicación de bezier vista desde una aleta

Como podemos ver en el output de la función Show, la grafica de la aleta se ve acoplada y modificada por los puntos definidos anteriormente. Tomando como punto base los de inicio y final, y siendo modificada su forma a nuestra conveniencia por los puntos de control.

```
In[366]:= Show[Graphics3D[{PointSize[Medium], Red, Map[Point, pts]}],  
Graphics3D[{Gray, Line[pts], Line[Transpose[pts]]}],  
ParametricPlot3D[f[u, v], {u, 0, 1}, {v, 0, 1}, Mesh -> None]]
```



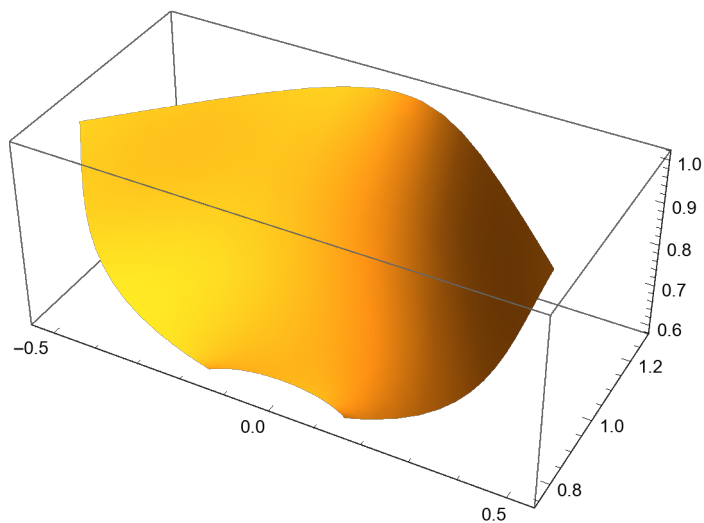
## Formación de todas las aletas con las funciones de bezier halladas

Tal como fue expuesto en el punto anterior, se usa la función `ParametricPlot3D` para graficar las funciones desde las expresiones paramétricas halladas desde `BezierFunction` para cada uno de los arreglos de puntos configurados en el ejercicio.

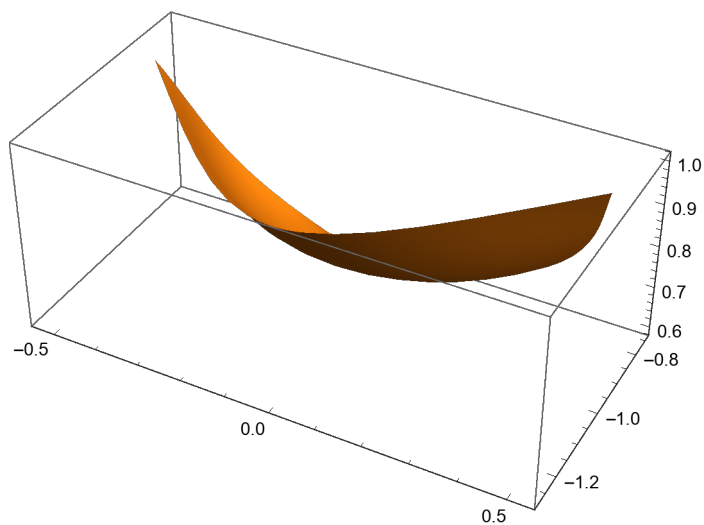
In[367]:=

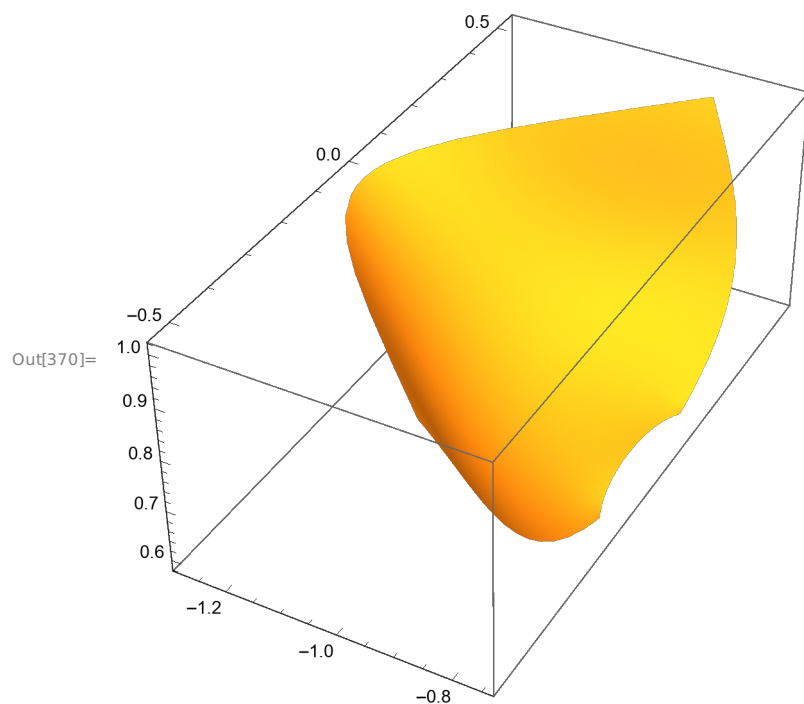
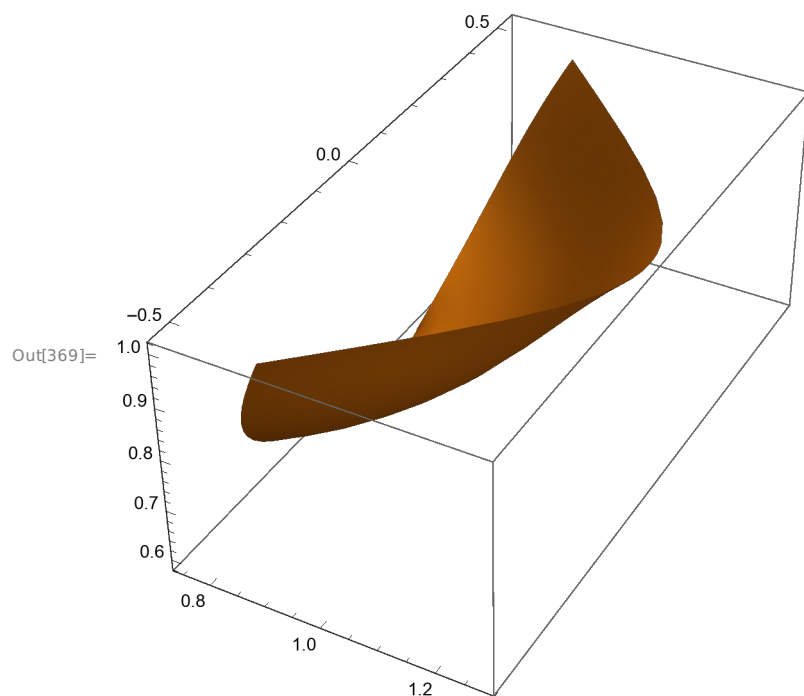
```
p1 = ParametricPlot3D[f[u, v], {u, 0, 1}, {v, 0, 1}, Mesh → None]
p2 = ParametricPlot3D[fdos[u, v], {u, 0, 1}, {v, 0, 1}, Mesh → None]
p3 = ParametricPlot3D[ftres[u, v], {u, 0, 1}, {v, 0, 1}, Mesh → None]
p4 = ParametricPlot3D[fcuatro[u, v], {u, 0, 1}, {v, 0, 1}, Mesh → None]
```

Out[367]=



Out[368]=





---

## Mortero valenciano finalizado

Finalmente, unimos todas las gráficas realizadas anteriormente, y configuramos la función `Show` para que no sea mostrada la caja habitual de Wolfram, así como los ejes de esta.



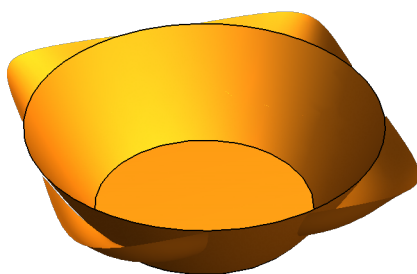
Las funciones a unir son las siguientes:

- p1 -> Aleta frontal
- p2 -> Aleta trasera
- p3 -> Aleta lateral 1
- p4 -> Aleta lateral 2
- par -> Paraboloide elíptico de revolución
- circ -> Tapa inferior del mortero

In[371]:=

```
Show[p1,p2,p3,p4,par, circ, Boxed -> False, Axes ->False]
```

Out[371]=



Versión interactiva (notebook) disponible en:

<https://www.wolframcloud.com/obj/ee6e276e-2f0e-4f29-97b0-07c28c28f015>