Informe para el usuario:

Este sería un ejemplo de como ejecutar el programa, el flag **-ql** es para la calidad, la **l** significa low, existen otros como **m**, **h**, **k** los cuales corresponden a medium, high y 4k respectivamente. Se debe tener cuidado al usar estos por el consumo de CPU. También podemos añadirle **-p** al flag anterior quedando **-pql** para que al terminar de ejecutar el programa muestre el resultado en un video. Los demás parámetros deben ser el nombre del .py a ejecutar y el nombre de la clase que implementa las clases de Manim. El flag --format=gif es usado para que el archivo que retorne sea en formato gif y no un video.

```
regnod@regnod-HP-ProBook-430-G2:~/Documents/4to/modelos de optimización 2/Proyecto Manim$ manim -ql --format=gif Geo Manim.py Geo Manim
```

Un ejemplo del proceso del programa.

```
                        mp4'
[02/18/22 17:36:28] INFO     Animation 1 : Partial movie file written in            scene_file_writer.py:514
                             '/home/regnod/Documents/4to/modelos de optimización
                             2/Proyecto Manim/media/videos/Penalty_Manim/480p15/parti
                             al_movie_files/Penalty/3833637021_3000509717_463076363.m
                             p4'
[02/18/22 17:36:39] INFO     Animation 2 : Partial movie file written in            scene_file_writer.py:514
                             '/home/regnod/Documents/4to/modelos de optimización
                             2/Proyecto Manim/media/videos/Penalty_Manim/480p15/parti
                             al_movie_files/Penalty/3833637021_2465062817_1672486969.
                             mp4'
[02/18/22 17:36:49] INFO     Animation 3 : Partial movie file written in            scene_file_writer.py:514
                             '/home/regnod/Documents/4to/modelos de optimización
                             2/Proyecto Manim/media/videos/Penalty_Manim/480p15/parti
                             al_movie_files/Penalty/3833637021_683211467_3437384959.m
                             p4'
Waiting 4:  60%|                                                | 9/15 [00:29<00:19,  3.32s/it]
```

Cuando llegue aqui ya terminó el proceso y muestra el path donde se encuentra el archivo.

```
[02/18/22 17:37:53] INFO     Animation 5 : Partial movie file written in            scene_file_writer.py:514
                             '/home/regnod/Documents/4to/modelos de optimización
                             2/Proyecto Manim/media/videos/Penalty_Manim/480p15/parti
                             al_movie_files/Penalty/3511951629_1515588106_895485703.m
                             p4'
                    INFO     Combining to Movie file.                               scene_file_writer.py:608
[02/18/22 17:37:54] INFO                                                            scene_file_writer.py:729
                             File ready at '/home/regnod/Documents/4to/modelos de
                             optimización 2/Proyecto Manim/media/videos/Penalty_Manim
                             /480p15/Penalty_ManimCE_v0.14.0.gif'

                    INFO     Rendered Penalty                                                  scene.py:237
                             Played 6 animations
```

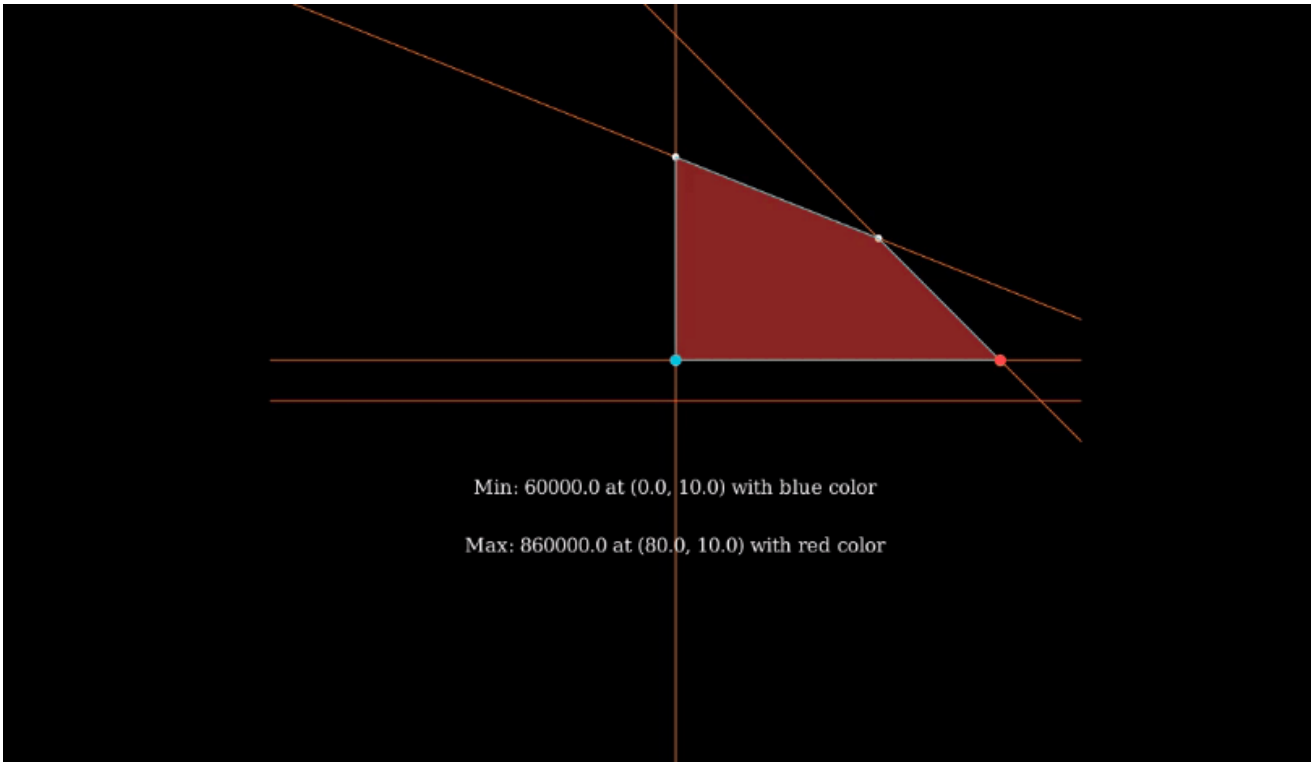
Esta sería la configuración del **penalty_settings.json** la cual consiste en 2 objetos, el primer string corresponde a la función objetivo y el segundo es una lista con las restricciones descritas a través de strings. Los otros parámetros son más específicos del método de penalización, la cantidad de iteraciones como caso de parada, el coeficiente de penalización, el valor para aumentar este, los rangos de x y las y.

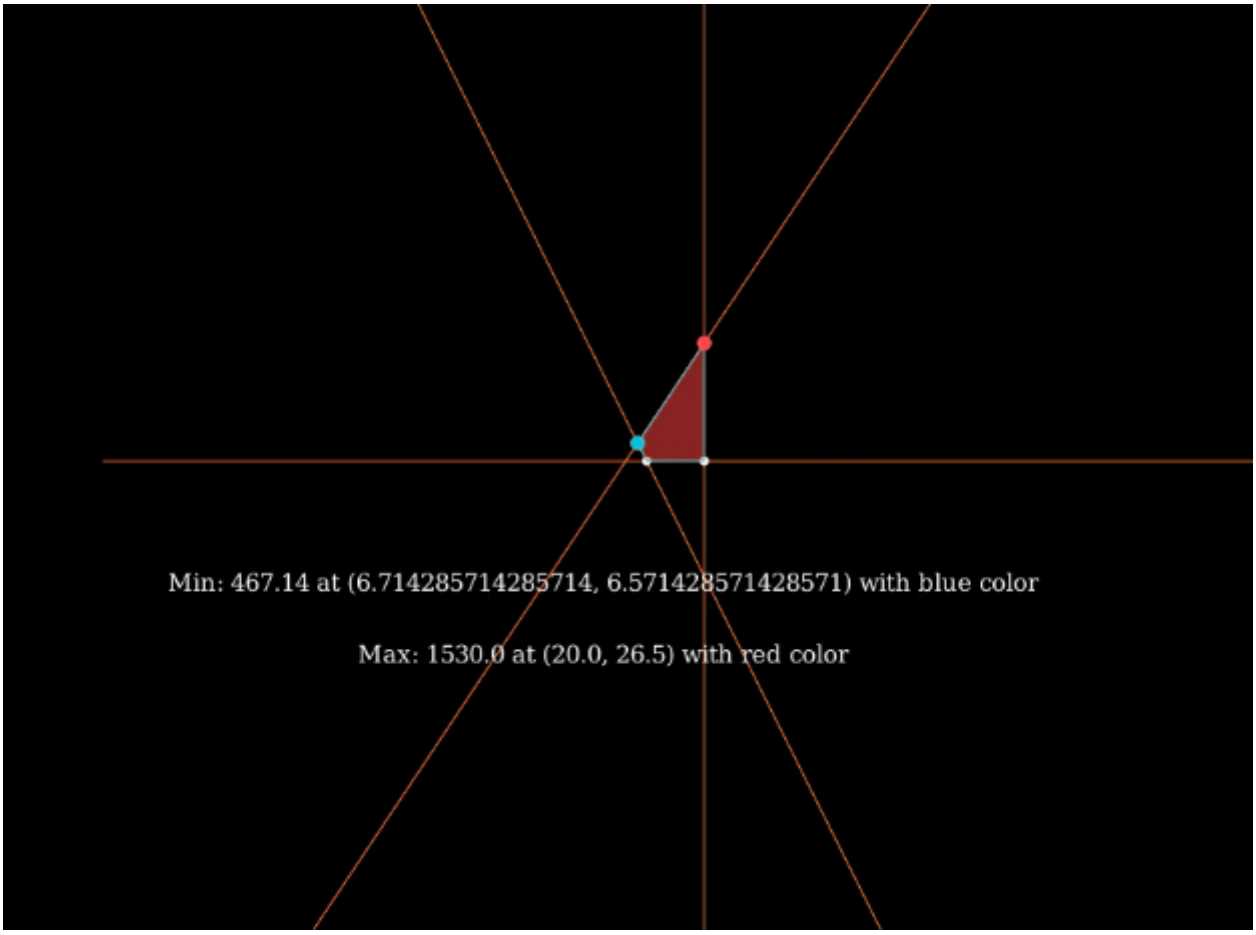
```
{
    "Penalty_number_of_sequence": 20,
    "Penalty_penalty_factor": 0.1,
    "Penalty_update_factor": 0.155,
    "Penalty_constraints" : ["x>=0", "y>=0", "(x-5)**2 + y**2 - 26 >= 0"],
    "Penalty_max_or_min": 0,
    "Penalty_init_point": [0.11, 0.11],
    "Penalty_x_range": [-20, 20, 1],
    "Penalty_y_range": [-20, 20, 1],
    "Penalty_func" : "(x**2 + y - 11)+(x + y**2 - 7)**2",
    "Penalty_vars" : ["x", "y"]
}       You, 4 hours ago • reset
```

Esta sería la configuración del **geometric_aproach.json** la cual consiste en 2 objetos, el primer string corresponde a la función objetivo y el segundo es una lista con las restricciones descritas a través de strings.

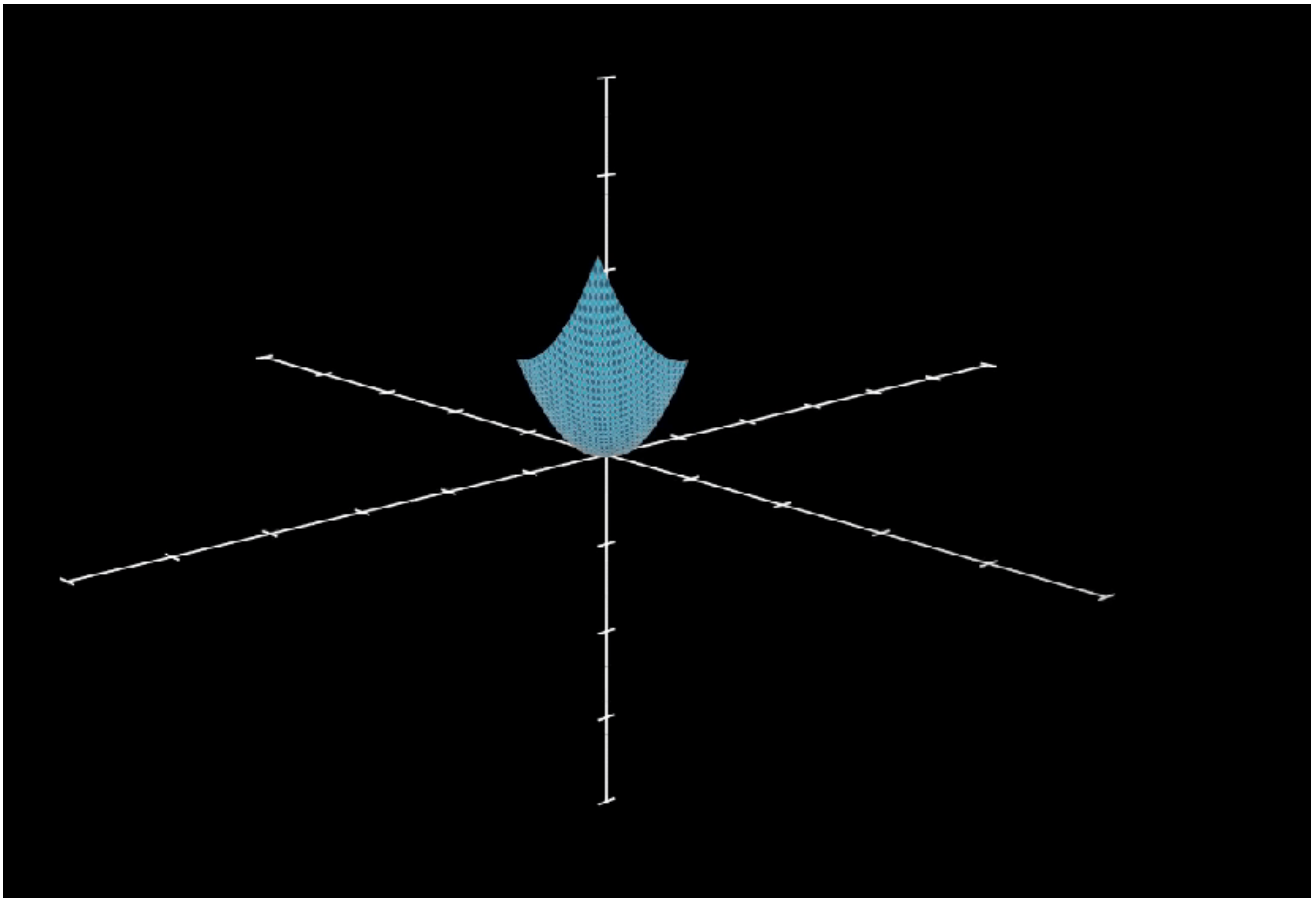
```
{      You, 4 hours ago • reset
    "func": "10000*x + 6000*y",
    "constraints": ["20*x+50*y <= 3000", "x+y <= 90", "y >= 10", "y >= 0", "x >= 0"]
}
```

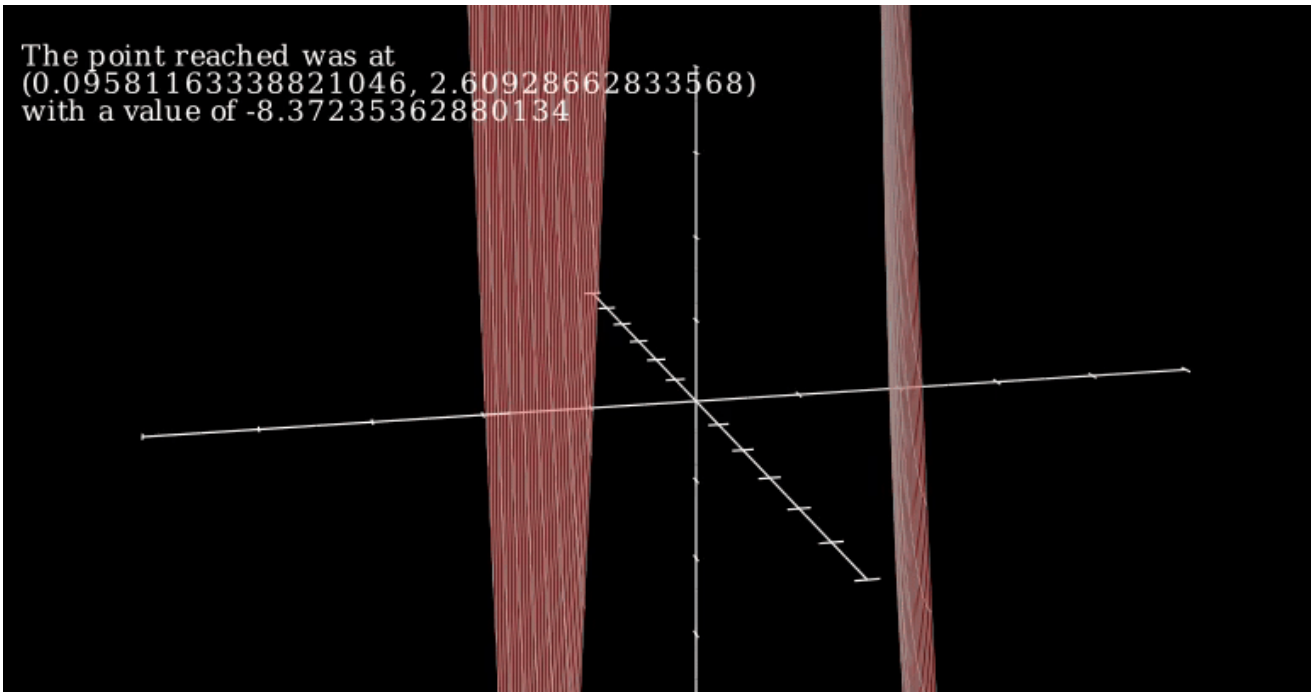
A continuación tenemos ejemplos de lo realizado.





Aquí podemos observar como sería el resultado de graficar una función con esta herramienta en 3d.





Ramificación y acotación:

Para realizar una animación del método de ramificación y acotación se debe definir un json con el nombre `bab.json` que posea la siguiente estructura:

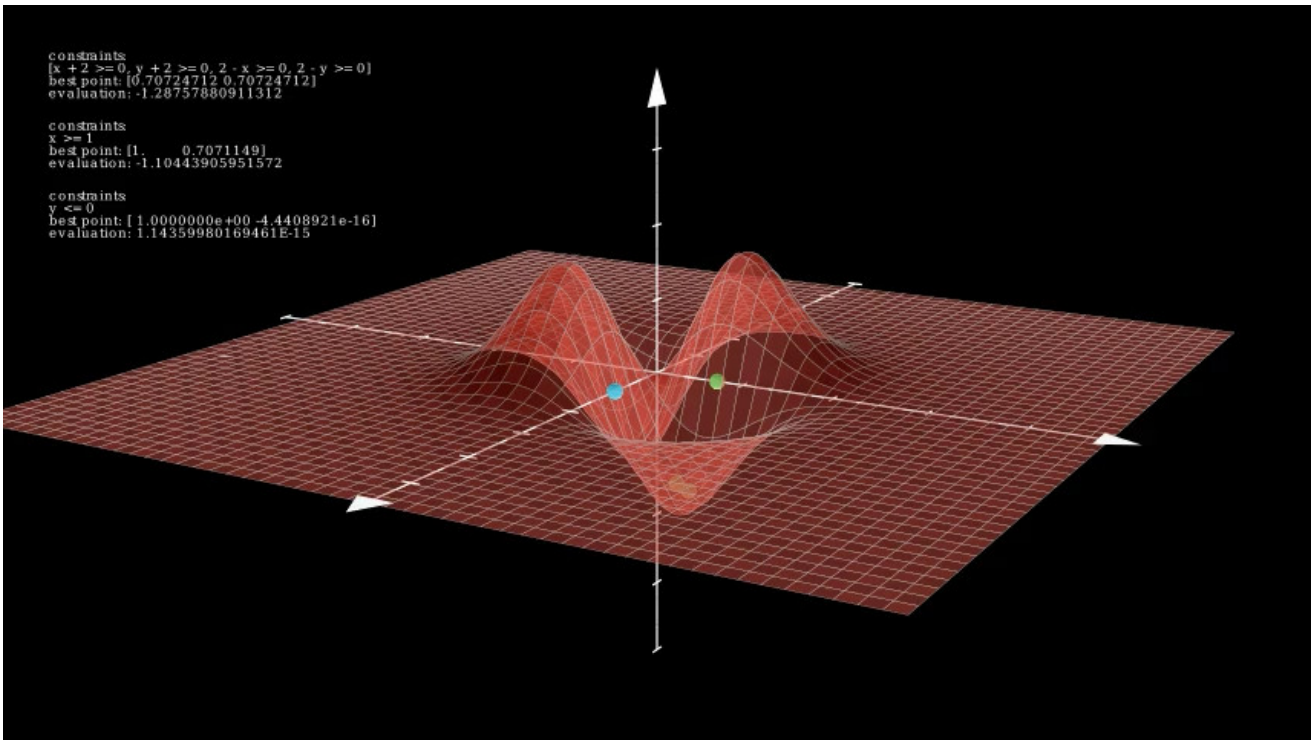
```
{
    "vars": vars,
    "func": func,
    "constraints": constraints,
    "initial_point": initial_point,
    "u_range": u_range,
    "v_range": v_range,
    "stroke_width": stroke_width
}
```

- "vars" (list[string]): las diferentes variables que se encuentran en la función a minimizar
- "func" (string): la función que se desea minimizar
- "constraints" (list[string]): las diferentes restricciones que se desean agregar
- "initial_point" (list[float]): punto inicial que se tomará para la minimización de la función
- "u_range" ([float, float]): el intervalo que se generará en el gráfico en el eje x
- "v_range" ([float, float]): el intervalo que se generará en el gráfico en el eje y
- "stroke_width" (float): grosor de las líneas en el gráfico a generar

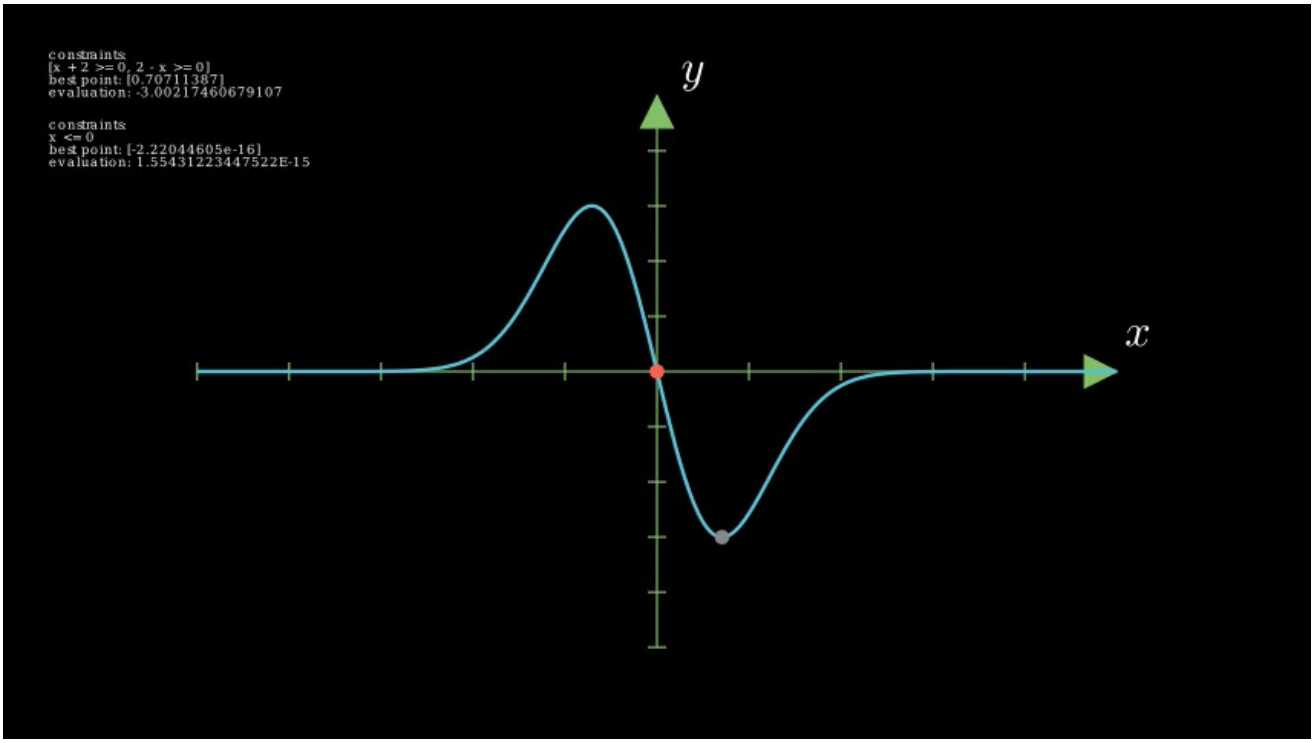
Podemos poner como ejemplo:

```
{
    "vars": ["x", "y"],
    "func": "- (7 * x * y / 2.71828 ** ( x ** 2 + y ** 2))",
    "constraints": ["x >= -2", "y >= -2", "x <= 2", "y <= 2"],
    "initial_point": [1, 1],
    "u_range": [-5, 5],
    "v_range": [-5, 5],
    "stroke_width": 0.5
}
```

Este json anterior nos daría como resultado una animación como se muestra en la siguiente imagen:



En cambio si lo modificamos para que solo posea una sola variable obtendremos algo como el siguiente ejemplo:



Métodos numéricos para la optimización no lineal

Para realizar una animación del Método del gradiente, Métodos del gradiente conjugado y el Método de Newton se debe definir un json con el nombre `numerical_optimization.json` que posea la siguiente estructura:

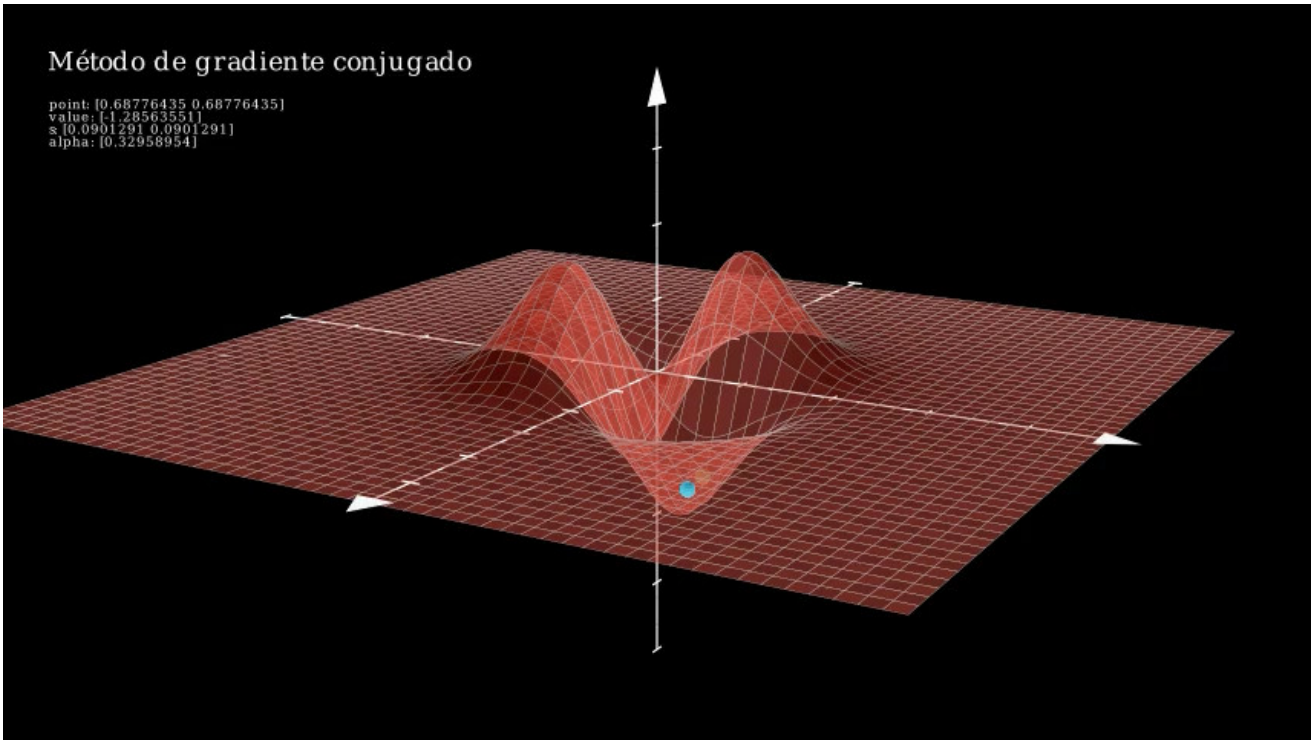
```
{
    "vars": vars,
    "func": func,
    "initial_point": initial_point,
    "u_range": u_range,
    "v_range": v_range,
    "stroke_width": stroke_width
    "cycles": cycles
}
```

- "vars" (list[string]): las diferentes variables que se encuentran en la función a minimizar
- "func" (string): la función que se desea minimizar
- "initial_point" (list[float]): punto inicial que se tomará para la minimización de la función
- "u_range" ([float, float]): el intervalo que se generará en el gráfico en el eje x
- "v_range" ([float, float]): el intervalo que se generará en el gráfico en el eje y
- "stroke_width" (float): grosor de las líneas en el gráfico a generar
- "cycles" (int): cantidad de iteraciones máximas que se desean realizar para obtener una aproximación del mínimo valor.

Podemos poner como ejemplo:

```
{
    "vars": ["x", "y"],
    "func": "- (7 _ x _ y / 2.71828 ** ( x ** 2 + y \*\* 2))",
    "constraints": ["x >= -2", "y >= -2", "x <= 2", "y <= 2"],
    "initial_point": [1, 1],
    "u_range": [-5, 5],
    "v_range": [-5, 5],
    "stroke_width": 0.5,
    "cycles": 500
}
```

Este json anterior nos daría como resultado una animación como se muestra en la siguiente imagen:



En cambio si lo modificamos para que solo posea una sola variable obtendremos algo como el siguiente ejemplo: