Iván Quintáns González e Christian Novoa González

Índice

01

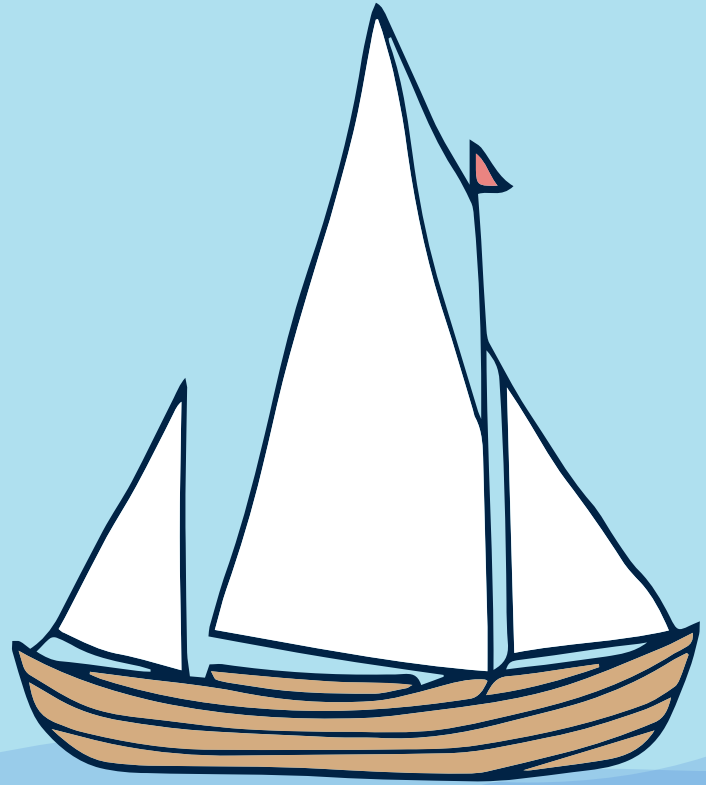
Presentación del problema

Búsqueda de camino más óptimo

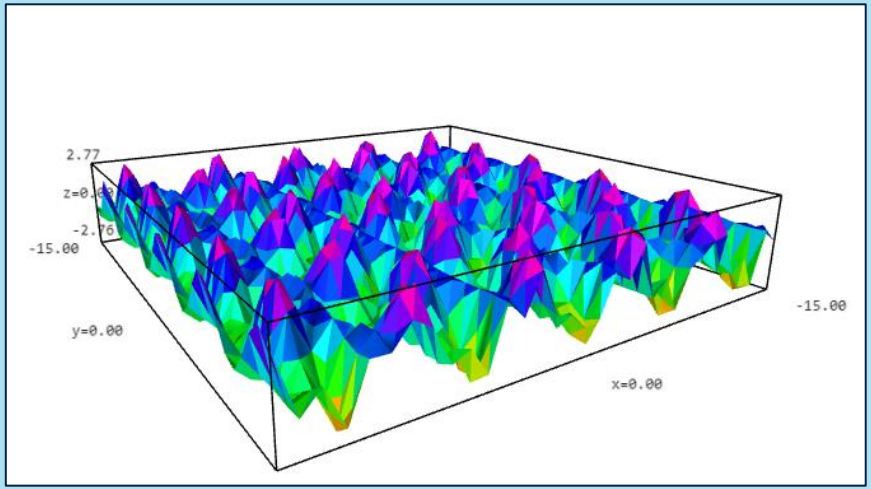
En el equipo de trabajo de nuestra propia empresa Vantian Maps, se les ocurrió la genial idea de implementar un sistema de navegación por mar que nos permita encontrar el camino más óptimo teniendo en cuenta las olas del mar.
Las olas grandes no solo cuestan mucho tiempo para el barco de subir si no que son más peligrosas, mientras que las olas más pequeñas y las zonas que no son un impedimento muy grande para el barco nos proporcionan una gran seguridad y una mayor rapidez a la hora de aproximarse a nuestro destino.

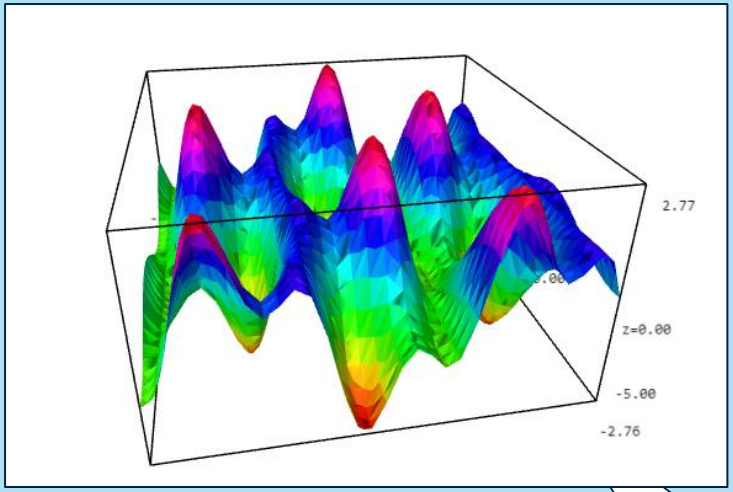
Tras diversas pruebas sobre una función, que modelára el mar, obtuvimos la siguiente:

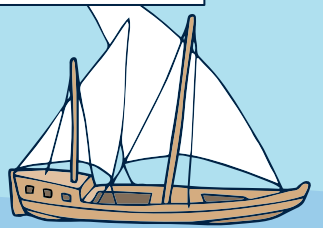
$$h(x, y) = cos(x) * (-2)\, sen(y) + sen(3x)$$

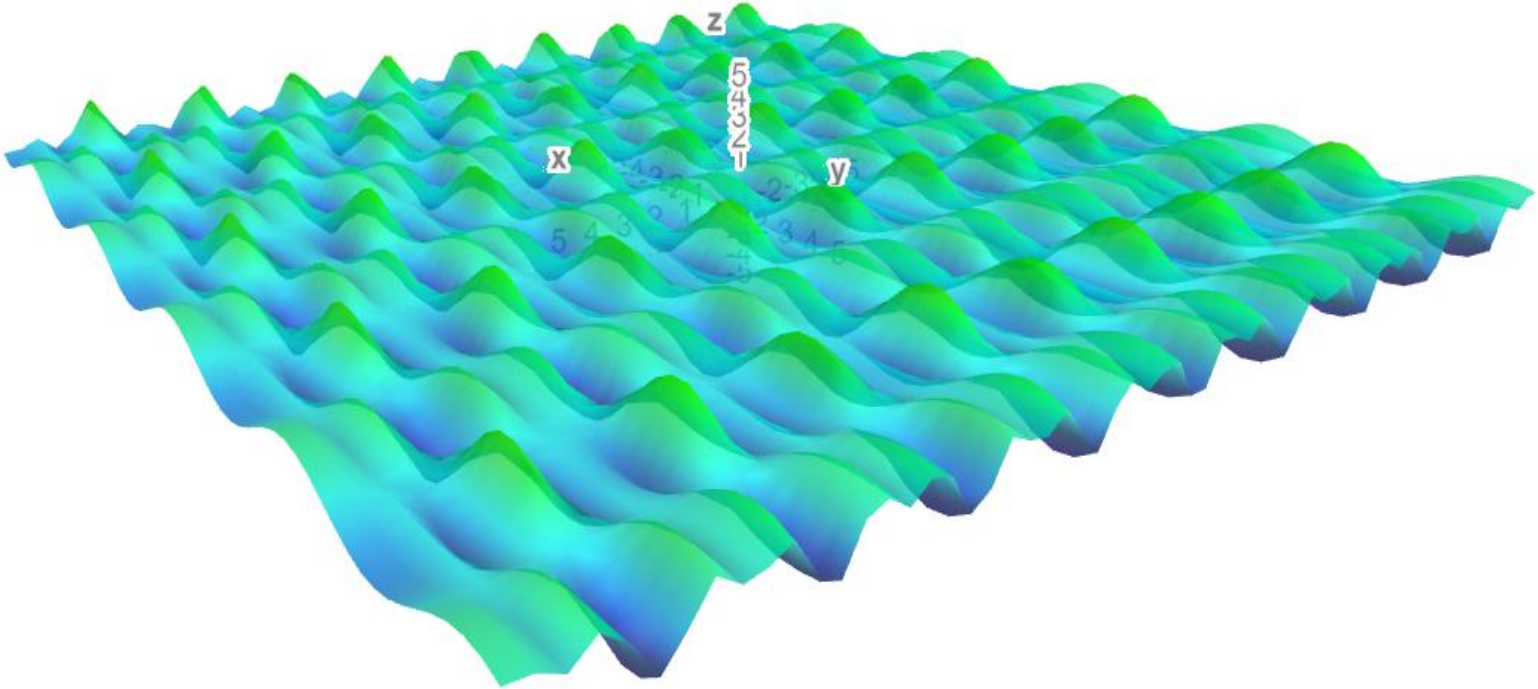


Representación del mar



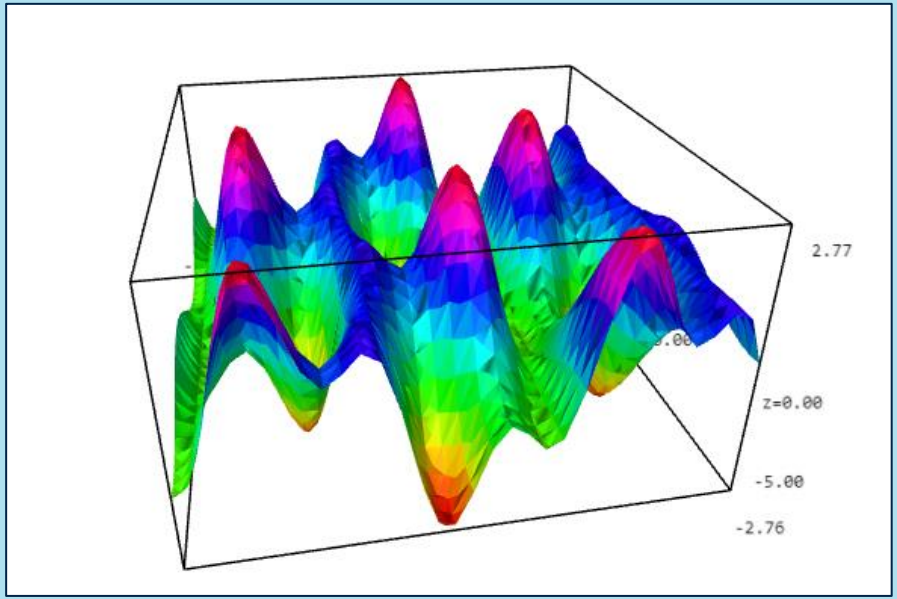


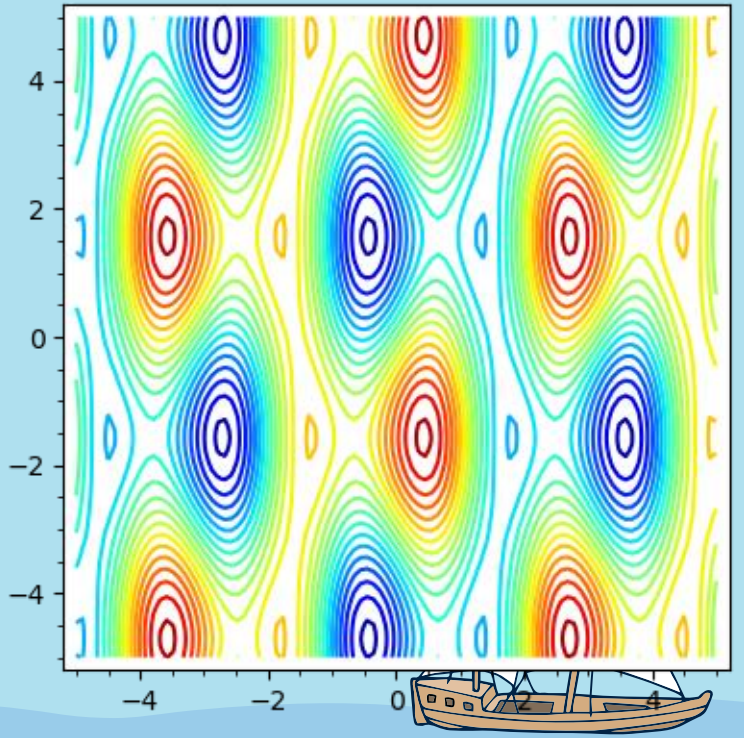



z
5
4
3
2
1
x
y

Representación Función

$$h(x,y) = cos(x) * (-2)\, sen(y) + sen(3x)$$





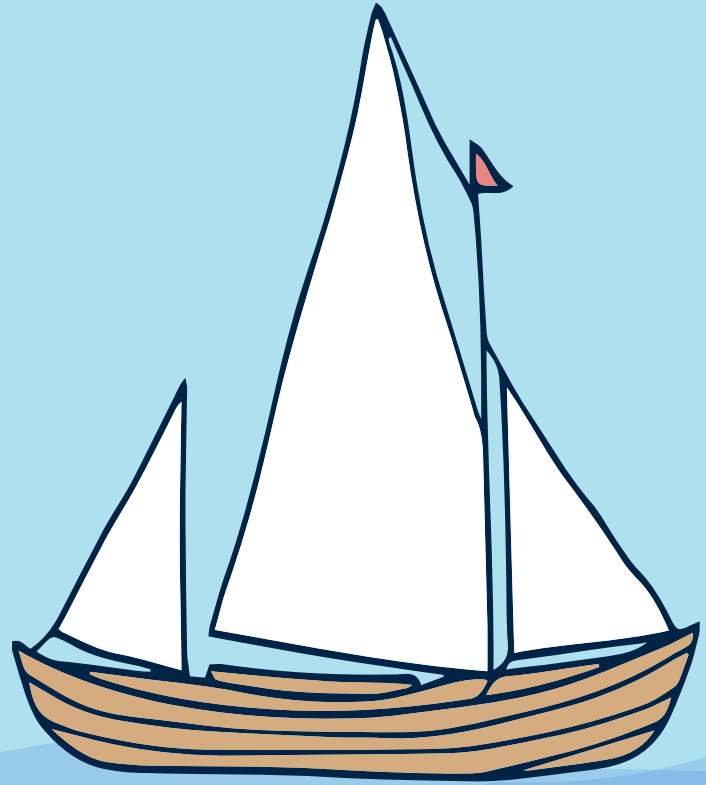
Definimos el vector gradiente como:

$$\nabla h(x, y) = 0$$

$$f(x, y) = (f_1(x, y), f_2(x, y))$$

$$f_1(x, y) = h_x(x, y)$$
$$f_2(x, y) = h_y(x, y)$$

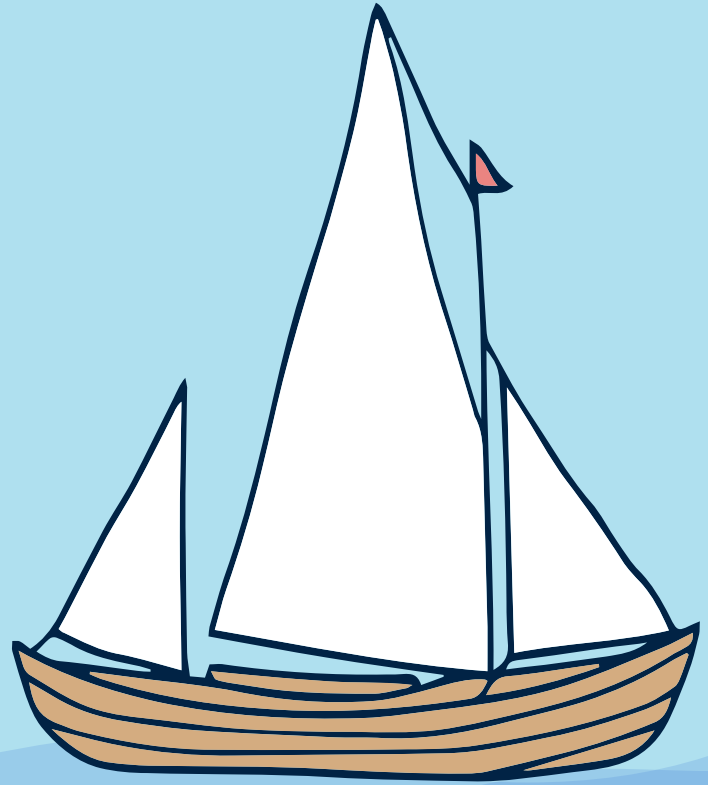
$$\begin{cases} f_1(x, y) = 0 \\ f_2(x, y) = 0 \end{cases}$$



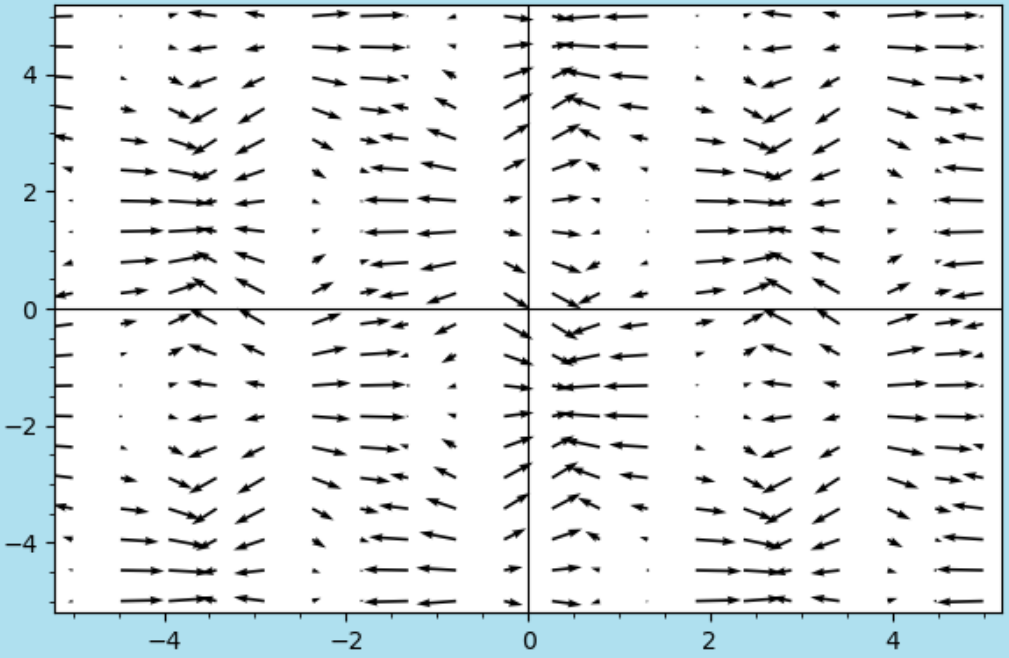
Las derivadas de la función son:

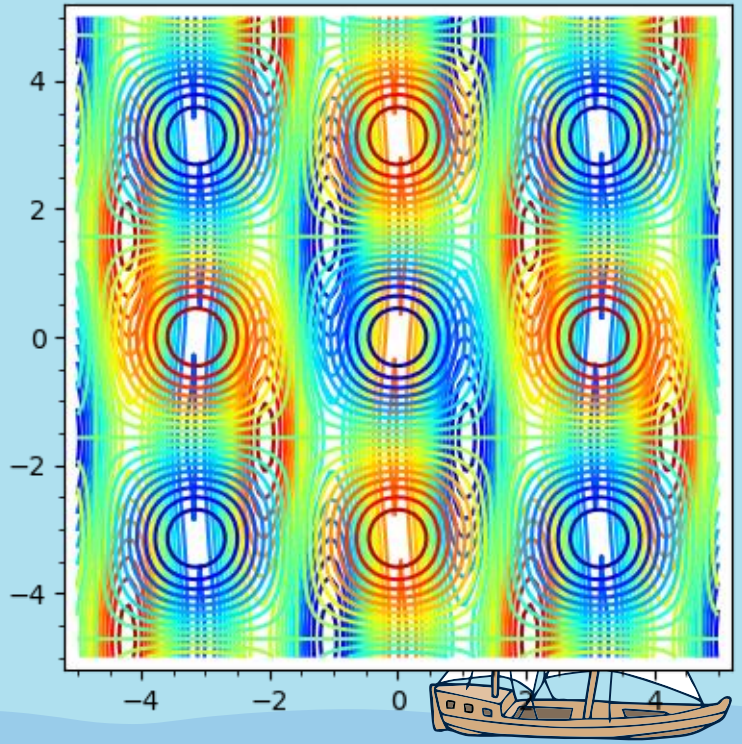
$$h_x(x,y) = 2sen(x)sen(y) + 3\cos(3x)$$
$$h_y(x,y) = -2\cos(x)\,cos(y)$$

El sistema resultante es un sistema no lineal .
Debemos encontrar las raíces del sistema.

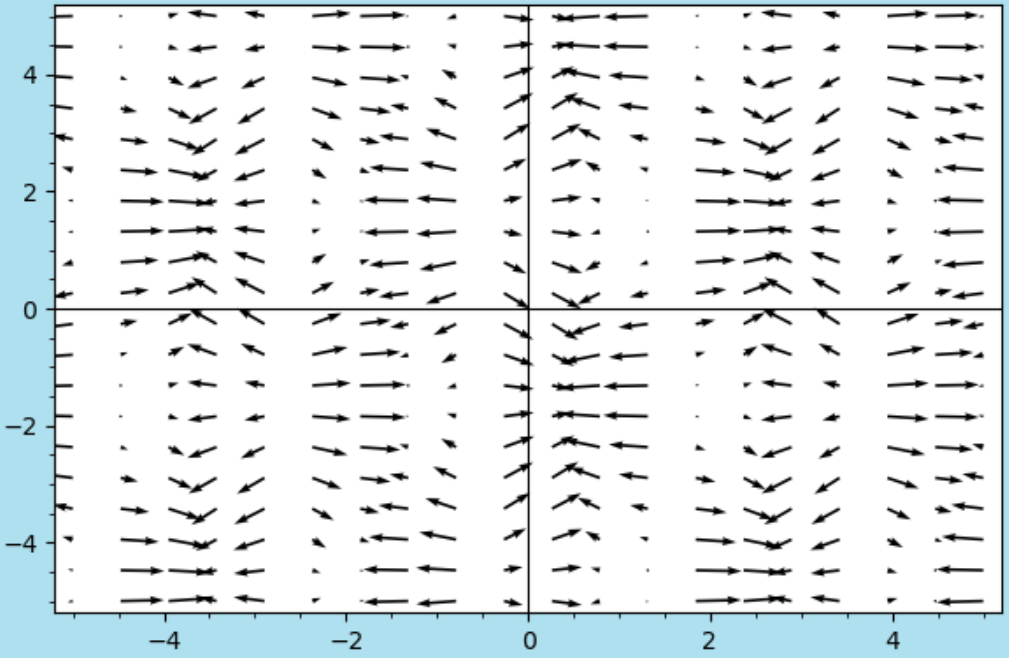


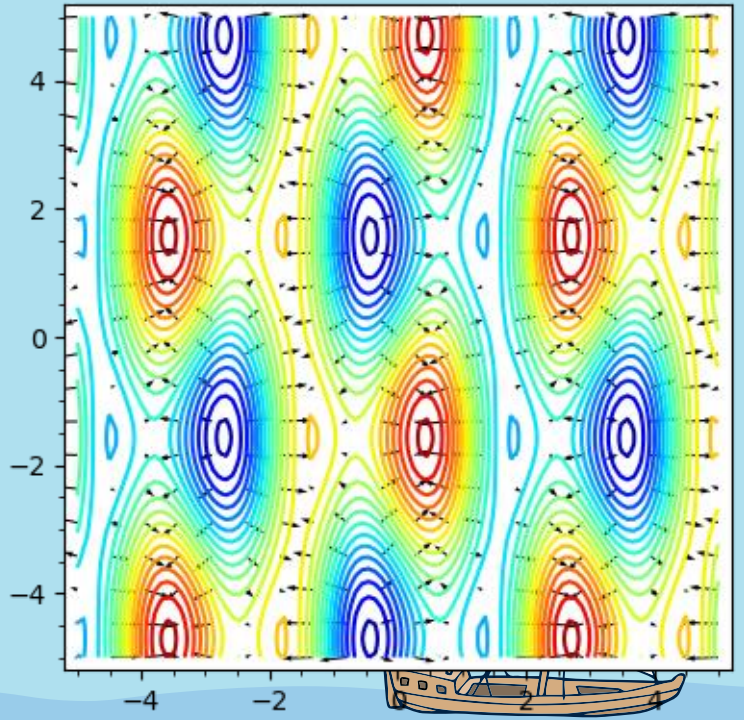
Vector Gradiente





Vector Gradiente





02

Método de Newton

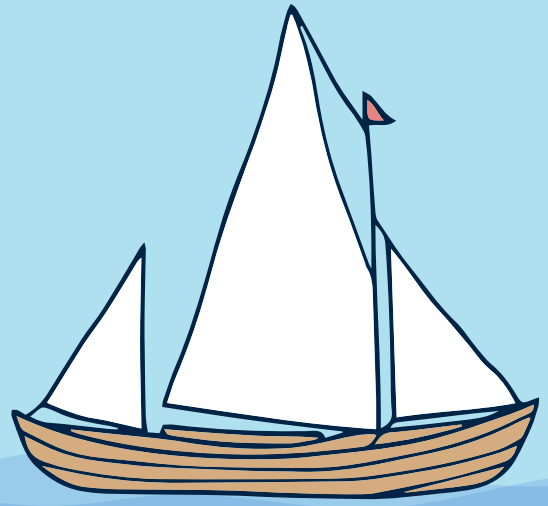
Para poder aplicar el método necesitamos, las derivadas parciales de la función para poder crear la matriz jacobiana.

$$h_{xx}(x,y) = 2cos(x)sen(y) - 9\text{sen}(3x)$$

$$h_{xy}(x,y) = 2sen(x)cos(y)$$

$$h_{yx}(x,y) = 2sen(x)cos(y)$$

$$h_{yy}(x,y) = 2\cos(x)\,sen(y)$$

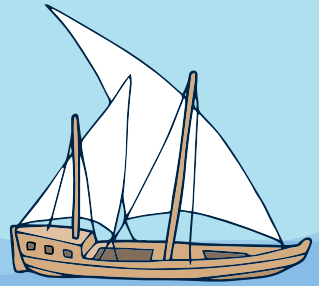


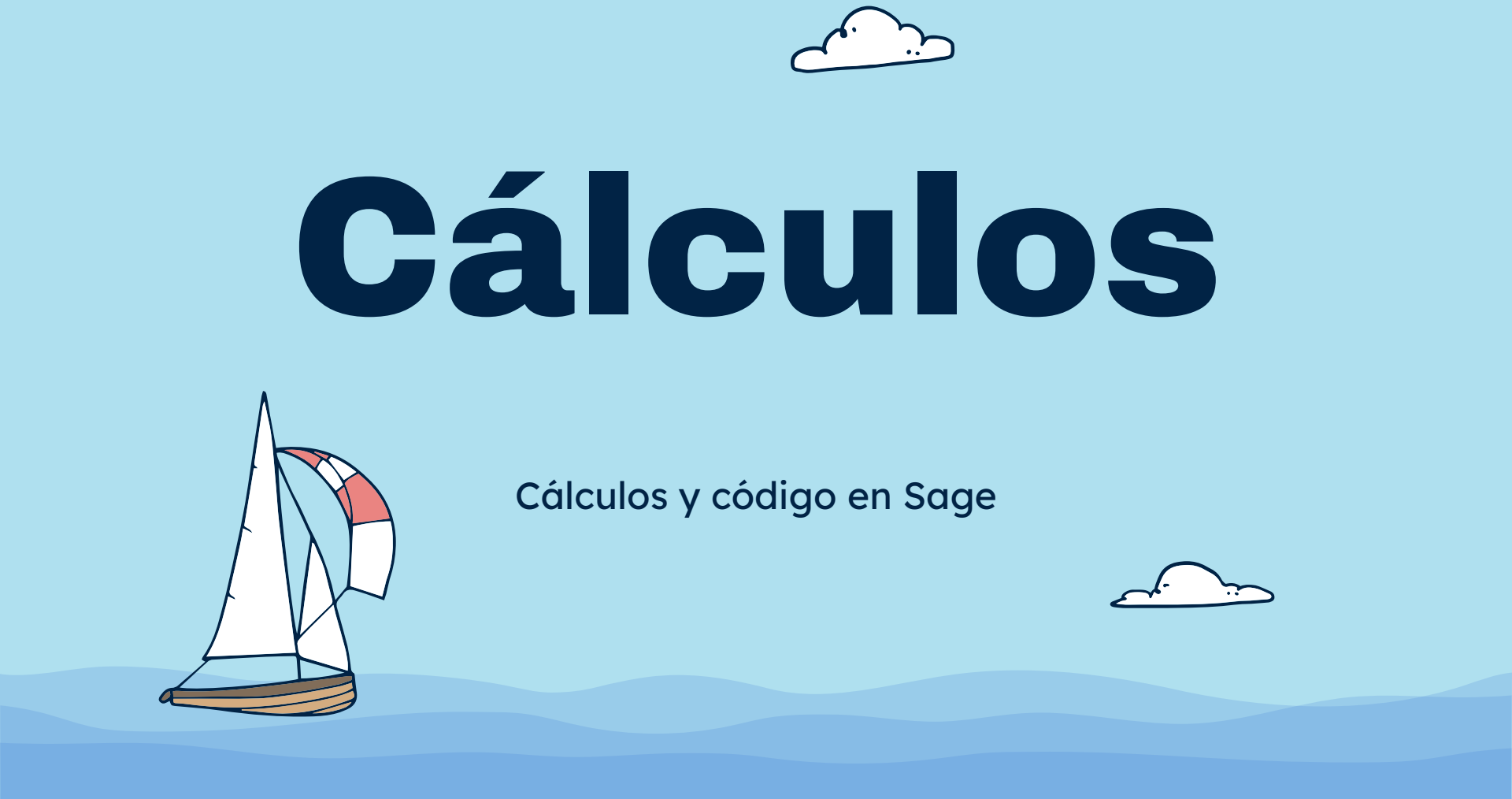
A partir de las derivadas parciales se forma la matriz Jacobiana:

$$Df(x,y) = \begin{pmatrix} f_{1x}(x,y) & f_{1y}(x,y) \\ f_{2x}(x,y) & f_{2y}(x,y) \end{pmatrix}$$

Esta será nuestra matriz Jacobiana, compuesta por las derivadas parciales de nuestra función.

$$Df(x,y) = \begin{pmatrix} 2cos(x)sen(y) - 9\text{sen}(3x) & 2sen(x)cos(y) \\ 2sen(x)cos(y) & 2\cos(x)\,sen(y) \end{pmatrix}$$





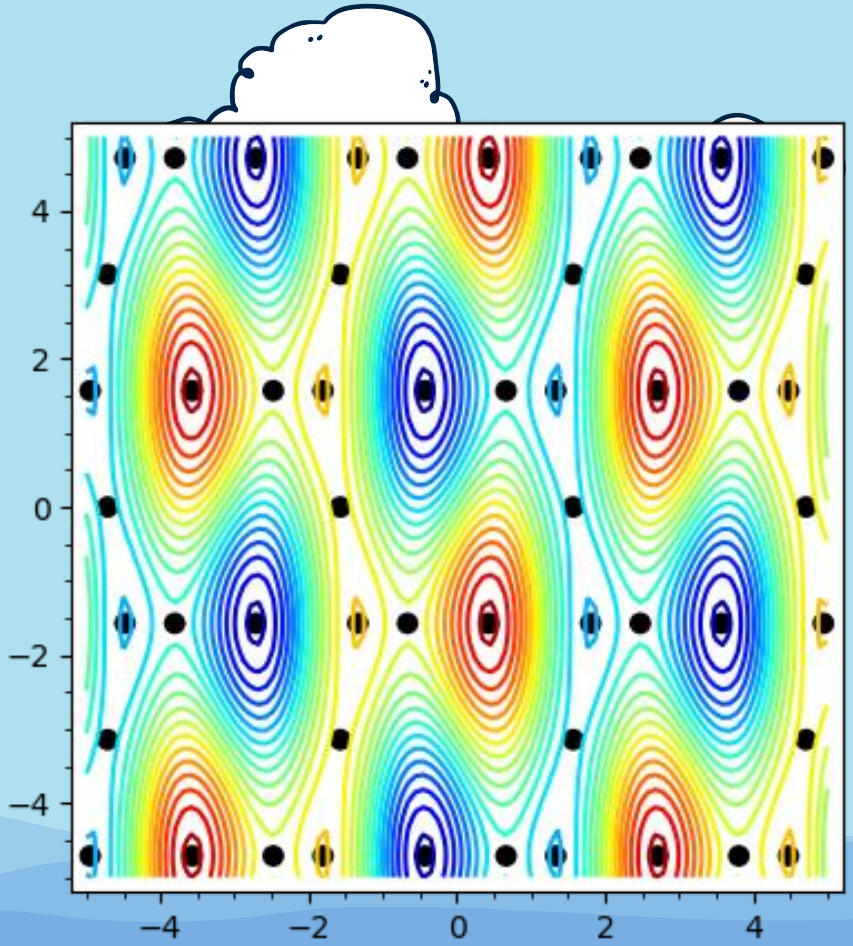
Cálculos

Cálculos y código en Sage

Método de Newton a Gran Escala

```python
def metodoNewtonGranEscala(f, interval_x: list, interval_y: list):
    raices = {}
    raices_rounded = []
    for i in interval_x:
        for j in interval_y:
            try:
                raiz, _, iterantes = newton(diff(f), diff(f,2), (i,j))
                raiz_rounded = [round(x,5) for x in raiz]
                criterio = interval_x[0] <= raiz_rounded[0] <= interval_x[-1] and \
                    interval_y[0] <= raiz_rounded[1] <= interval_y[-1]
                if raiz_rounded not in raices_rounded and criterio:
                    raices_rounded.append(raiz_rounded)
                    raices[tuple(raiz)] = iterantes
            except:
                pass
    return raices
```

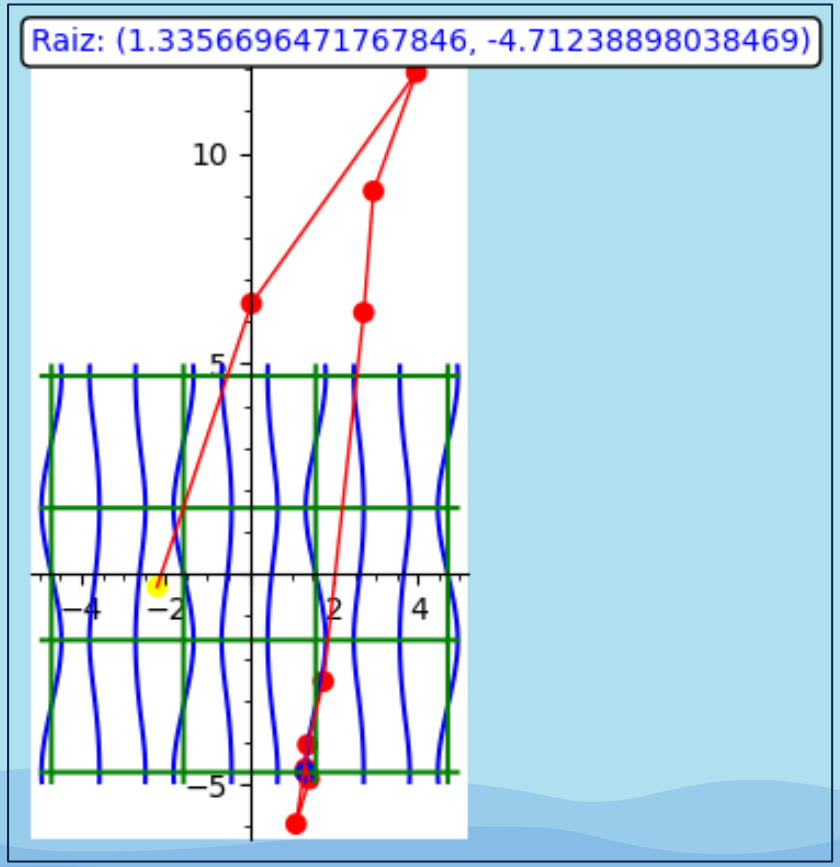
Resultados del Algoritmo

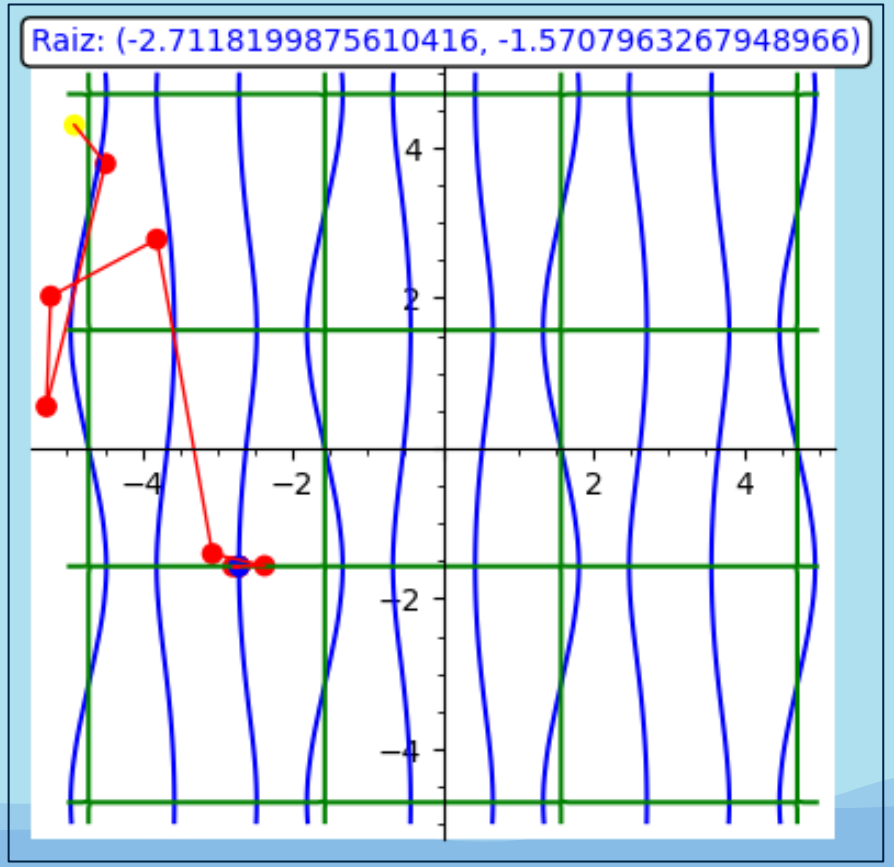


Debuxa Newton

```python
for key, value in raices.items():
    try:
        h1 = contour_plot(diff(f)[0], *args, cmap=['blue'], contours=[0], fill=False)
        h2 = contour_plot(diff(f)[1], *args, cmap=['green'], contours=[0], fill=False)
        h3 = point(value[1:], marker='o', size=50, color='red')+line(value, color='red')
        h3_init = point(value[0], marker='o', size=50, color='yellow')
        h3_end = point(value[-1], marker='o', size=50, color='blue')

        legend = text(f"Raiz: {key}", (0,1), vertical_alignment='bottom',horizontal_alignment='left',
                      bounding_box={'boxstyle':'round', 'fc':'w'}, axis_coords=True)
        show(legend+h1+h2+h3+h3_init+h3_end)
        #a = plot(legend+h1+h2+h3+h3_init+h3_end)
        #a.save(f"Fotos/Debuxa_Newton/debuxa_newton{key}.png")
    except:
        raise
```

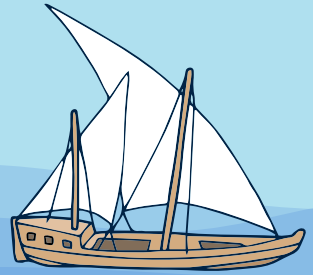

Raiz: (1.3356696471767846, -4.71238898038469)
10
5
−5
−4
−2
2
4


Raiz: (-2.7118199875610416, -1.5707963267948966)
4
2
−2
−4
−4
−2
2
4

Criterio Hessiana

$$Hh(x,y) = \begin{pmatrix} h_{xx}(x,y) & h_{xy}(x,y) \\ h_{yx}(x,y) & h_{yy}(x,y) \end{pmatrix}$$

- Si $|Hh(x_0,y_0)| > 0$ y $h_{xx}(x_0,y_0) < 0$, entonces en el punto (x_0,y_0) la función tiene un máximo relativo.
- Si $|Hh(x_0,y_0)| > 0$ y $h_{xx}(x_0,y_0) > 0$, entonces en el punto (x_0,y_0) la función tiene un mínimo relativo.
- Si $|Hh(x_0,y_0)| < 0$, h tiene un punto de silla.
- Si $|Hh(x_0,y_0)| = 0$ el criterio no decide.



Criterio Hessiana

```
Punto (-4.947515660002802, -4.71238898038469):
H[0][0]=7.318219331386502, det(H)=3.4097952244028247; o punto e un minimo local.
----------------------
Punto (-4.4772623007665775, -1.5707963267948966):
H[0][0]=7.318219331386497, det(H)=3.4097952244028282; o punto e un minimo local.
----------------------
Punto (-2.7118199875610416, 4.71238898038469):
H[0][0]=10.46393358434979, det(H)=19.024697289368994; o punto e un minimo local.
----------------------
Punto (-4.71238898038469, -3.141592653589793):
det(H)=-4.0; o punto e un punto de sela.
----------------------
Punto (-3.806491999236657, -1.570796326794897):
det(H)=-10.43449251377183; o punto e un punto de sela.
----------------------
Punto (-4.71238898038469, -2.1657751547027315e-15):
det(H)=-4.0; o punto e un punto de sela.
----------------------
Punto (-4.947515660002802, 1.5707963267948966):
H[0][0]=7.318219331386502, det(H)=3.4097952244028247; o punto e un minimo local.
----------------------
Punto (-4.71238898038469, 3.141592653589793):
det(H)=-4.0; o punto e un punto de sela.
----------------------
Punto (-4.4772623007665775, 4.71238898038469):
H[0][0]=7.318219331386497, det(H)=3.4097952244028282; o punto e un minimo local.
----------------------
Punto (-3.8064919992366573, 4.71238898038469):
det(H)=-10.434492513771819; o punto e un punto de sela.
----------------------
Punto (-2.7118199875610416, -1.5707963267948966):
H[0][0]=10.46393358434979, det(H)=19.024697289368994; o punto e un minimo local.
----------------------
```

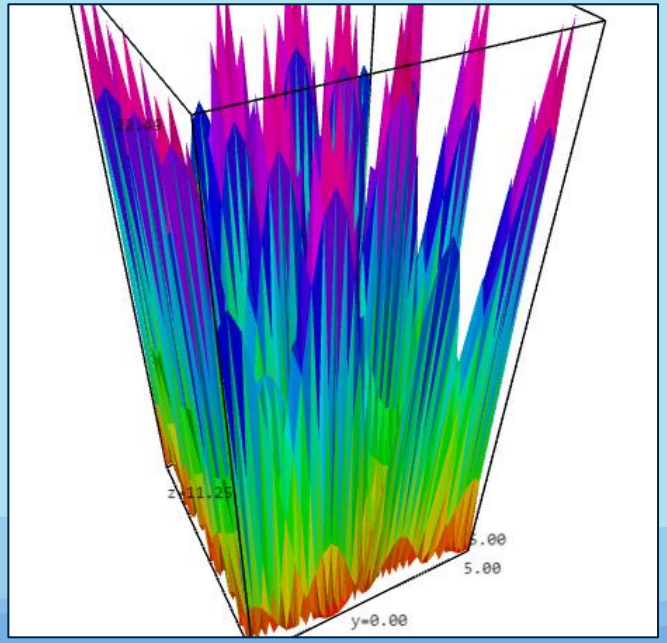
03

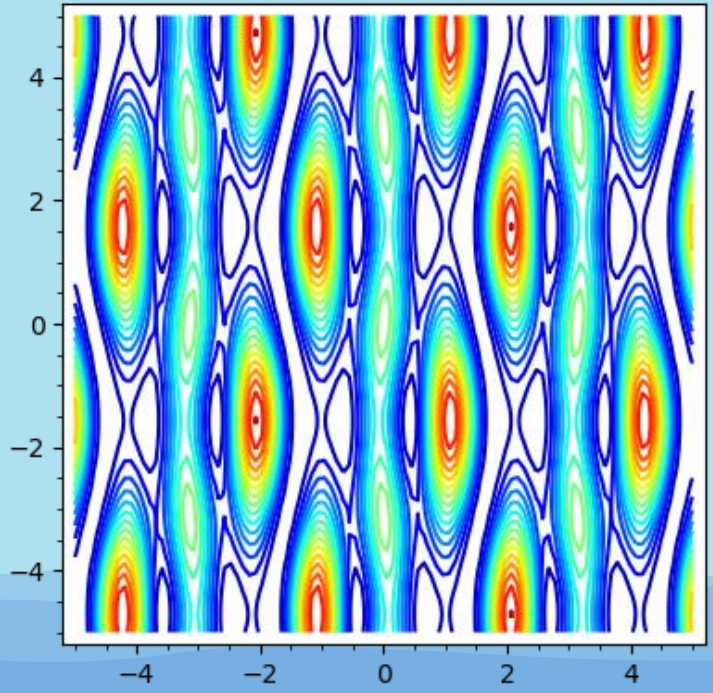
Descenso rápido

Función G

Definimos G como o cuadrado da norma do vector

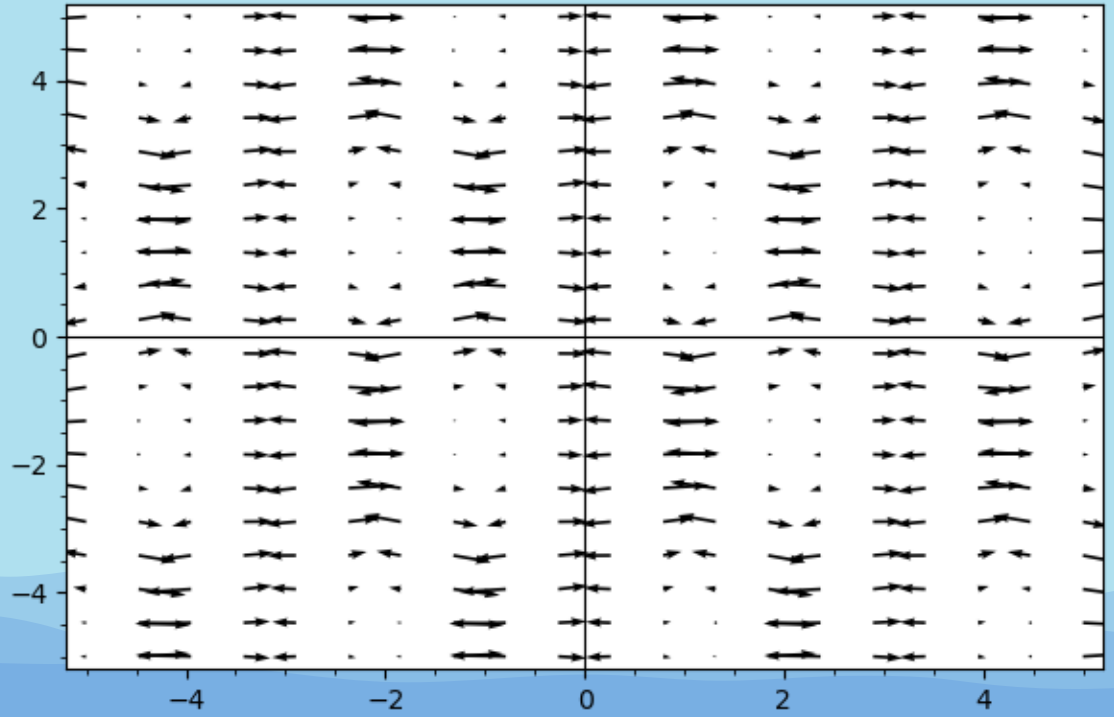
$$G(x,y) = ||f(x,y)||^2 = (f_1(x,y)^2 + f_2(x,y)^2)$$

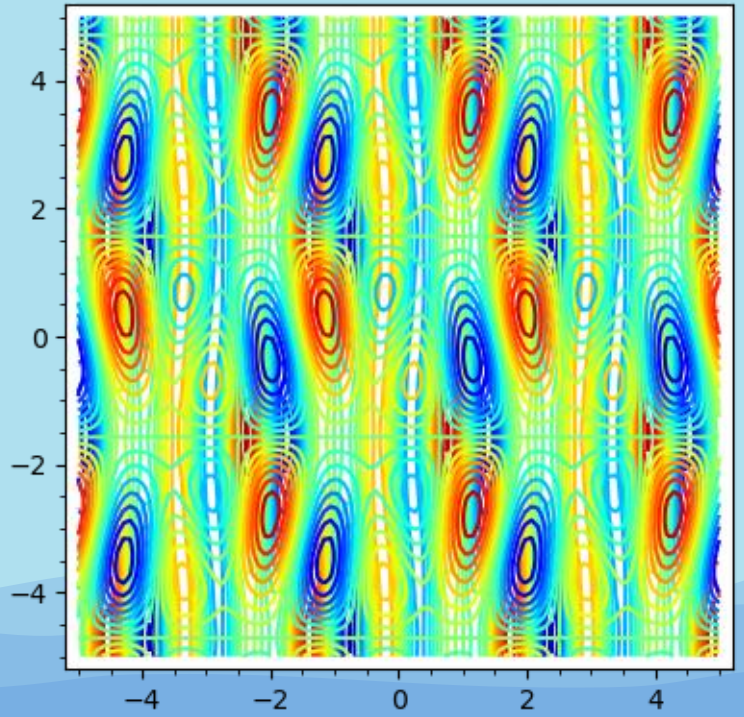




El Gradiente de G es:

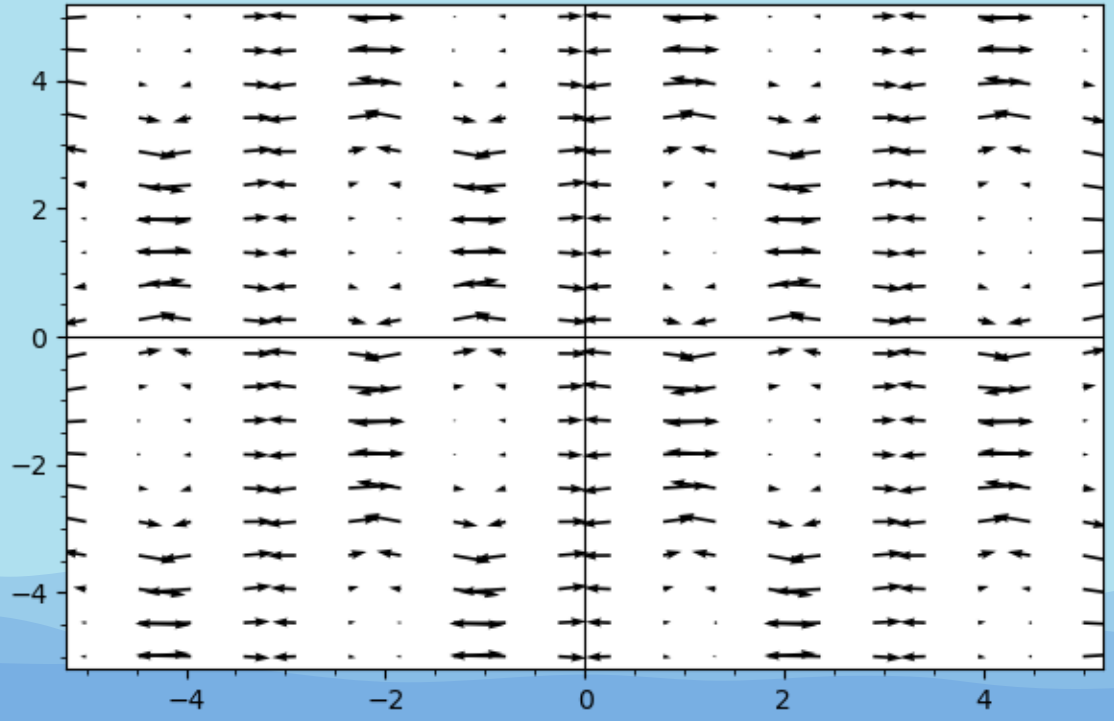
$$\nabla G(x, y) = (G_x(x, y), G_y(x, y))$$

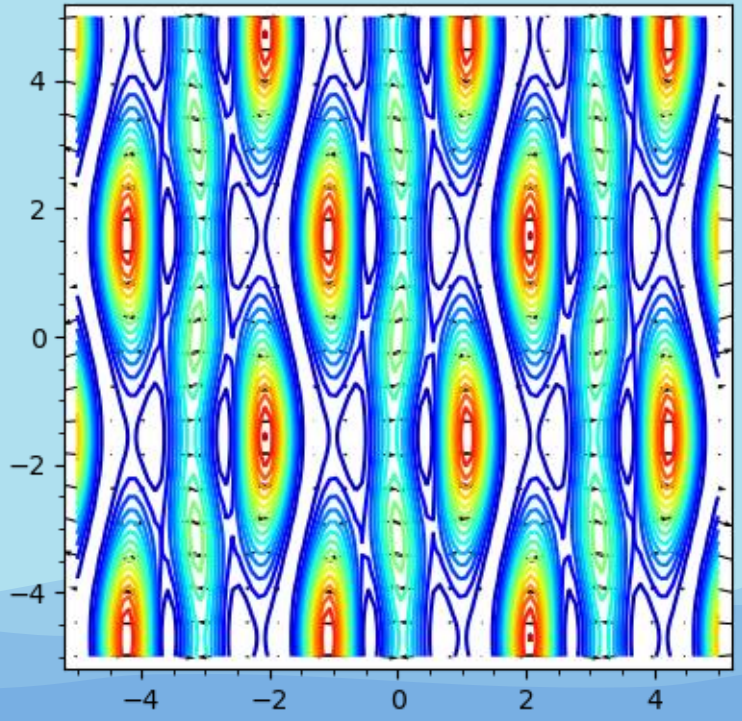


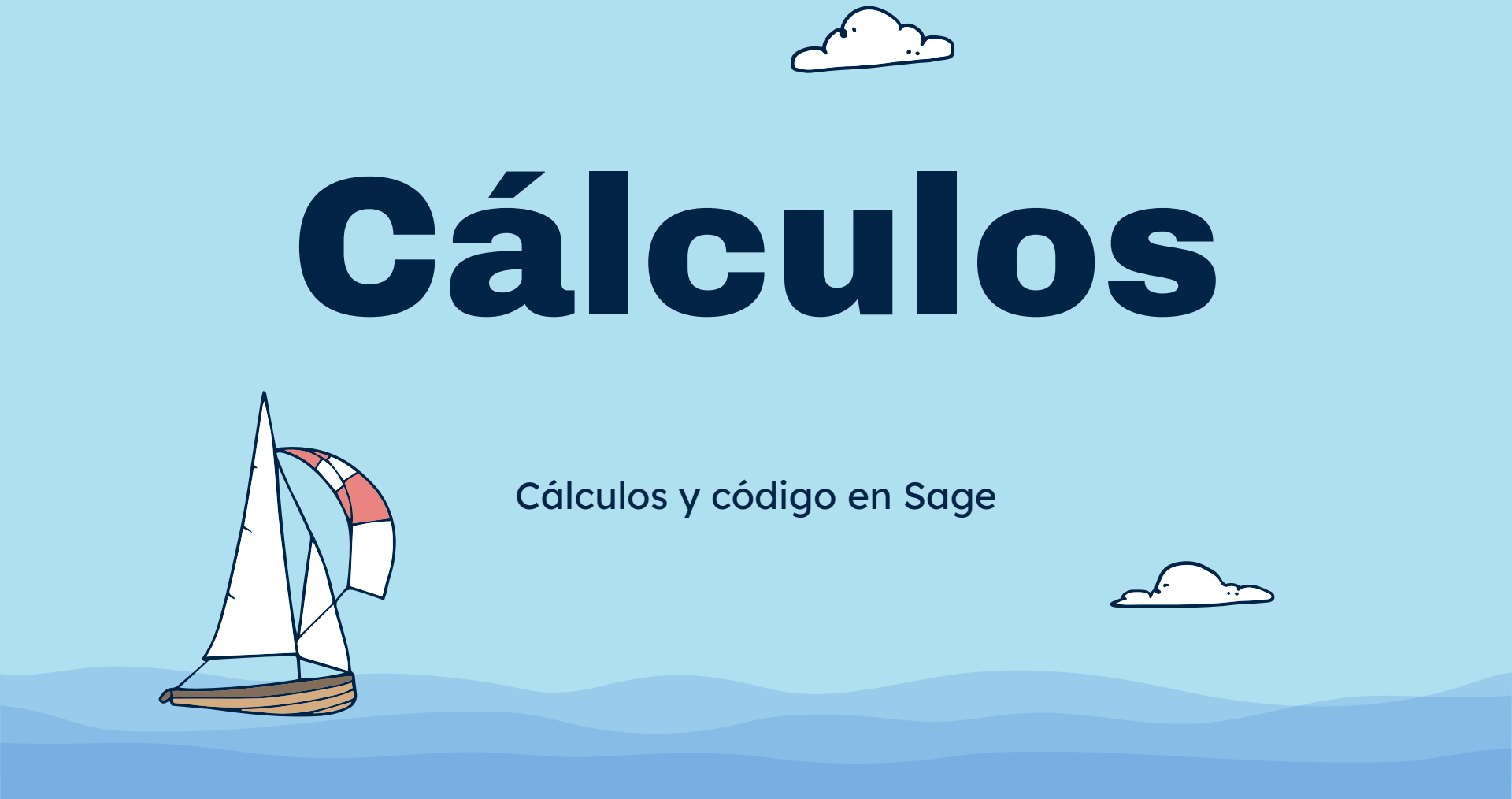


El Gradiente de G es:

$$\nabla G(x,y) = (G_x(x,y), G_y(x,y))$$







Cálculos

Cálculos y código en Sage

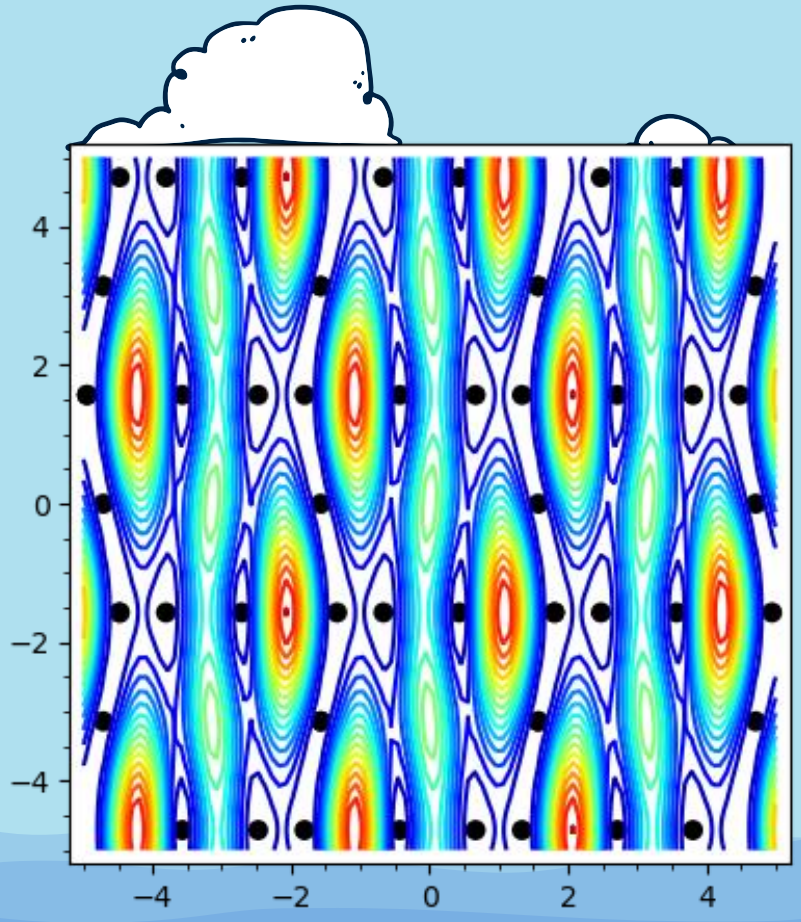
Método de Descenso a Gran Escala

```python
def metodoDescensoGranEscala(f, df, interval):
    minimos = {}
    minimos_rounded = []
    for i in interval:
        for j in interval:
            try:
                minimo, _, iterantes = descenso(f, df, (i,j))
                minimo_rounded = [round(x,5) for x in minimo]
                criterio = interval[0] <= minimo_rounded[0] <= interval[-1] and \
                    interval[0] <= minimo_rounded[1] <= interval[-1]
                if minimo_rounded not in minimos_rounded and criterio:
                    minimos_rounded.append(minimo_rounded)
                    minimos[tuple(minimo)] = iterantes
            except:
                pass
    return minimos
```

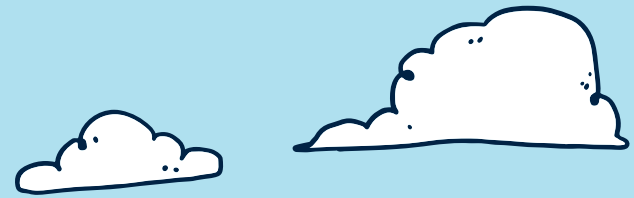
Resultados del Algoritmo

```
Punto (-4.7123897792000085, 3.141588886373728):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-3.8064920589987383, -1.5707964908491387):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-4.9475156579306985, 1.5707923186258386):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-4.477262299998509, -1.5707940737626036):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-3.571365266427268, -4.712388712201756):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-4.477262301529346, -1.5707987071071923):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-4.4772622987878945, 4.7123851487100055):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-3.8064920733432044, 4.712389206288708):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-3.571365320734128, 1.570796610995497):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-2.7118199877806797, -1.5707962286283246):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
Punto (-2.7118200390663687, 4.712389314974105):
Valor entre iterantes moi proximo, posible minimo de G -> Posible Punto de Sela de h
----------------------
```

Resultados del Algoritmo

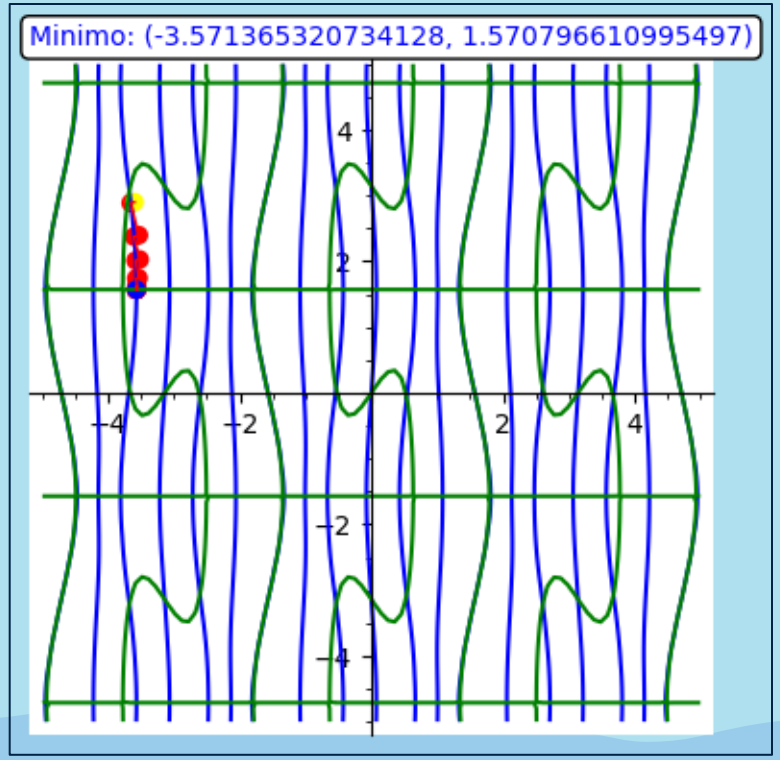


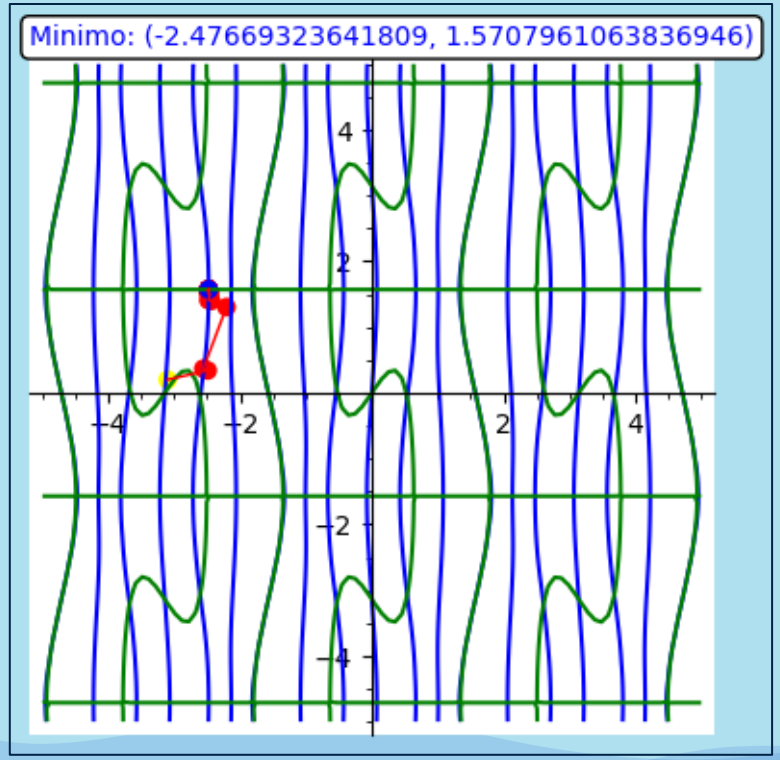
Debuxa Descenso



```python
def debuxaDescensoGranEscala(f, minimos):
    for key, value in minimos.items():
        try:
            h1 = contour_plot(diff(f)[0], *args, cmap=['blue'], contours=[0], fill=False)
            h2 = contour_plot(diff(f)[1], *args, cmap=['green'], contours=[0], fill=False)
            h3 = point(value[1:], marker='o', size=50, color='red')+line(value, color='red')
            h3_init = point(value[0], marker='o', size=50, color='yellow')
            h3_end = point(value[-1], marker='o', size=50, color='blue')

            legend = text(f"Minimo: {key}", (0,1), vertical_alignment='bottom',horizontal_alignment='left',
                          bounding_box={'boxstyle':'round', 'fc':'w'}, axis_coords=True)
            show(legend+h1+h2+h3+h3_init+h3_end)
            # a = plot(legend+h1+h2+h3+h3_init+h3_end)
            # a.save(f"Fotos/Debuxa_Descenso/debuxa_descenso{key}.png")
        except:
            raise
```

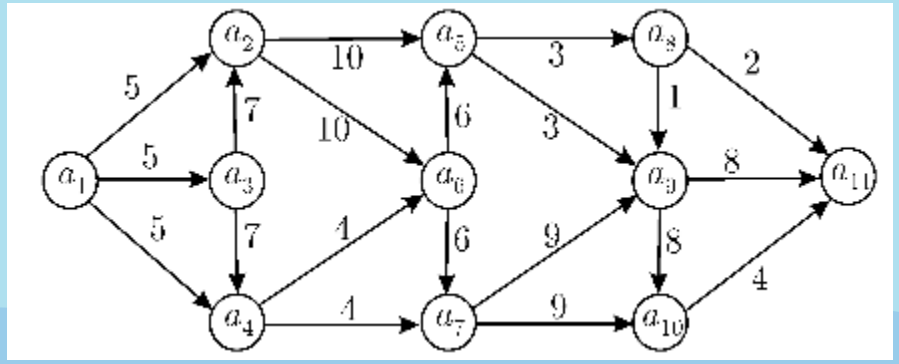

Minimo: (-3.571365320734128, 1.570796610995497)
4
2
−4
−2
2
4
−2
−4

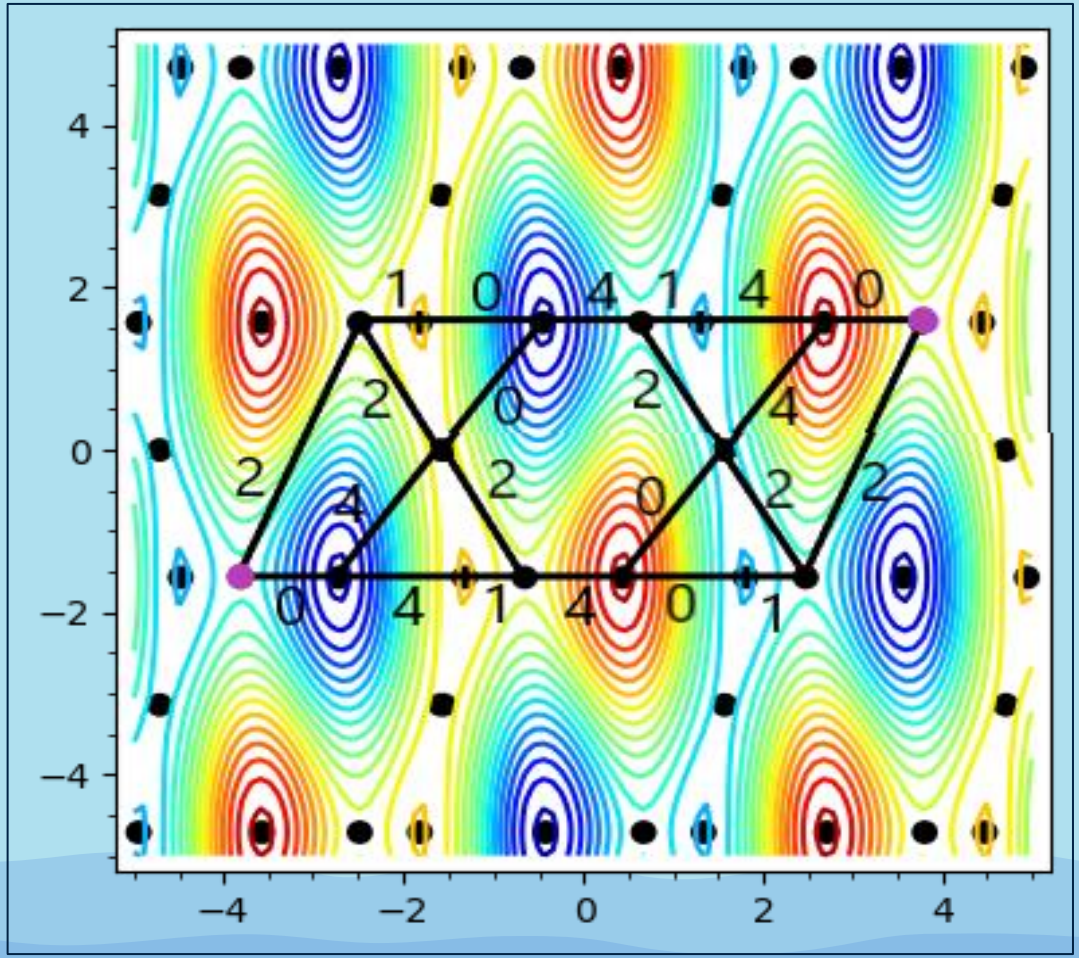

Minimo: (-2.47669323641809, 1.5707961063836946)
4
2
−4
−2
2
4
−2
−4

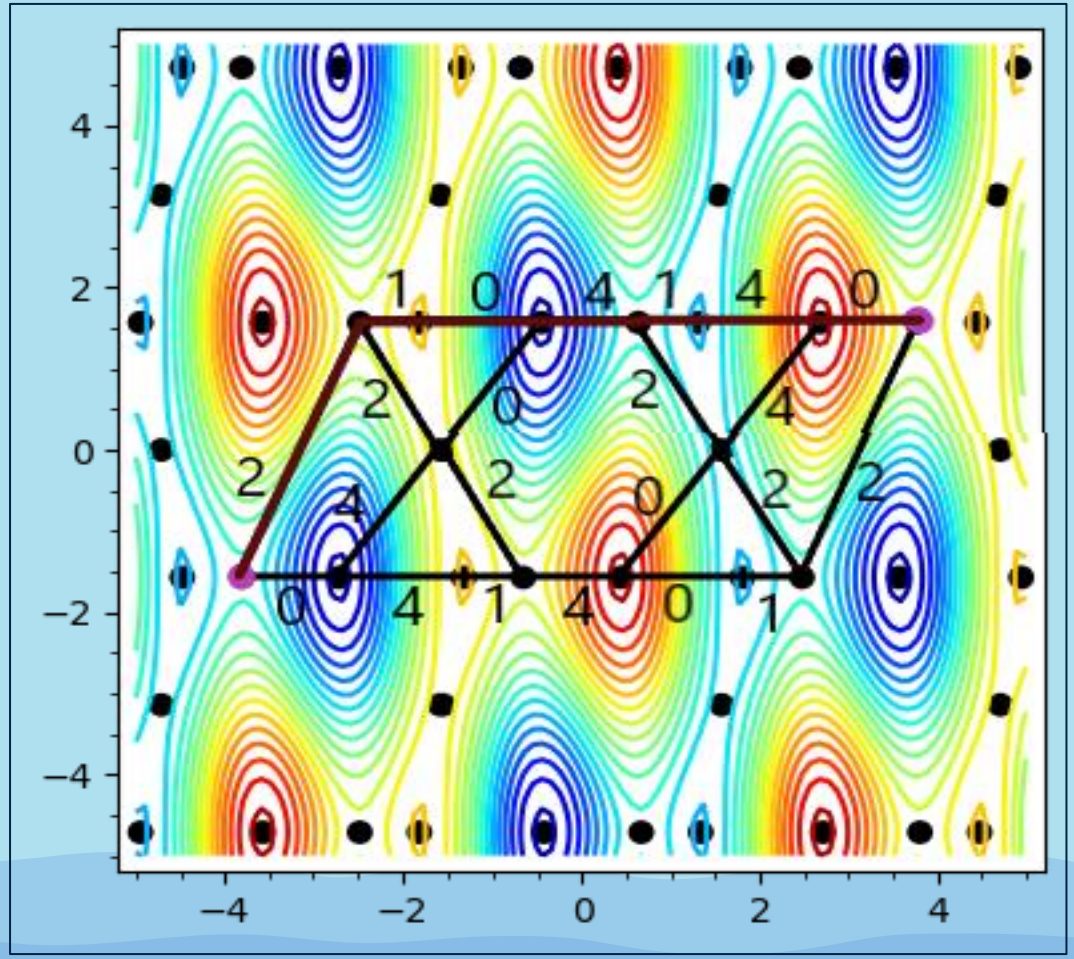
04 Conclusión

Una vez obtenemos todos los máximos, mínimos y puntos de silla, ya tenemos todos los datos para poder elegir el camino óptimo para nuestra embarcación.

Para ello podemos crear grafos ponderados suponiendo lo que cuesta subir, bajar o ir recto sobre el mar y buscar en nuestro caso el camino de menos peso para llegar desde nuestro punto de partida hasta el punto final.




4
2
0
−2
−4
−4
−2
0
2
4
1
0
4
1
4
0
2
0
2
4
2
4
2
0
2
2
0
4
1
4
0
1


4
2
0
−2
−4
−4
−2
0
2
4
1
0
4
1
4
0
2
0
2
4
2
2
4
2
0
2
2
0
4
1
4
0
1

05 Bibliografia

1. GeoGebra. GeoGebra for Teaching and Learning Math.Disponible en: https://www.geogebra.org/?lang=en
2. SageMath. Version 9.5 [Local]. Disponible en: https://www.sagemath.org/