Cálculo Avanzado proyecto Abril "Optimización"

Samuel Saéz F. - Felipe Pirul Y.

Abril 2021

1. Introducción

Queremos implementar un algoritmo de optimización para el ajuste de curvas, donde la función de costo se define por la comparación de datos con un modelo. Para esto, consideraremos el modelo simplificado de Weibull y el modelo de Higuchi.
Posteriormente mostraremos un ejemplo de la función Rosenbrock. Para la cual obtendremos su gráfico, cálculo de gradiente, dirección del descenso y hasta donde llega.

2. Modelos, datos experimentales y sus gráficos.

El modelo simplificado de Weibull está definido por :

$$u(t) = 1 - e^{-at} \tag{1}$$

Mientras que el modelo de Higuchi está definido por :

$$u(t) = at^{\frac{1}{2}} \tag{2}$$

Los datos experimentales que utilizaremos serán los que son dados en el guión:

x	y
0.1	0.2
0.2	0.3
0.3	0.45
0.4	0.55
0.5	0.6
0.6	0.7
0.7	0.75
0.8	0.8
0.9	0.8
1.0	0.8

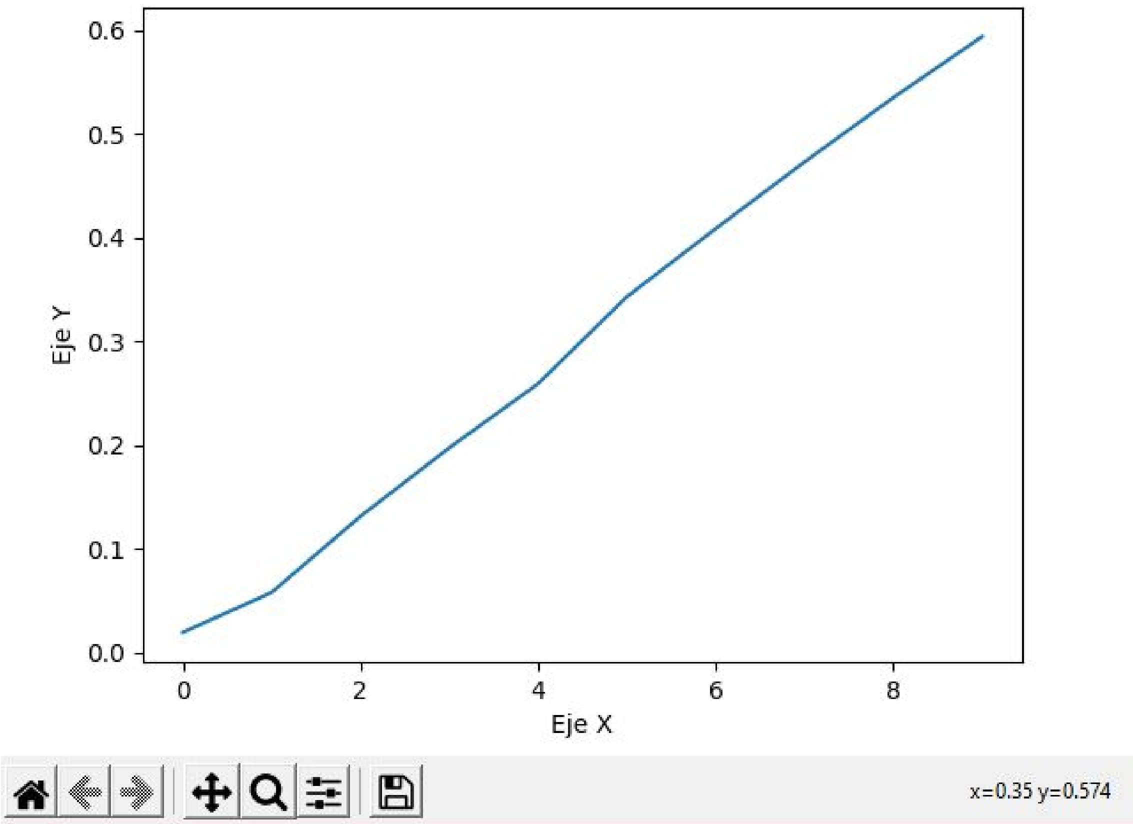
Cuadro 1: Datos experimentales

Comenzaremos graficando el modelo utilizando nuestra implementación en python como se ve a continuación:

```
C: > Users > SAMUEL > Downloads > Calculo > weibull.py > ...
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 30 19:21:24 2021

@author: felip
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.patches as mpatches

#DATOS EXPERIMENTALES TABLA GUIÓN#
y = np.linspace(0.1,1.0,10)
x = [0.2,0.3,0.47,0.55,0.6,0.7,0.75,0.8,0.85,0.9]

## ------------------------- funcion Weibull ------------------- ##

def Weibull(y,x):
    x_1 = 1-np.exp(-(x*y))
    return x_1

plt.plot(Weibull(y,x),label='Weibull')
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()

print("weibull: \n ",Weibull(y,x))
```

Obteniendo los siguientes resultados :

```
SALIDA   TERMINAL   CONSOLA DE DEPURACIÓN   PROBLEMAS 10

PS C:\Users\SAMUEL\Downloads\Calculo>  c:; cd 'c:\Users\SAMUEL\Downloads\Calculo'; & 'python' 'c:\Users\SAMUEL\.vscode\extensions\ms-python.pytho
py'
weibull:
  [0.01980133 0.05823547 0.13151069 0.1974812  0.25918178 0.34295318
 0.40844464 0.47270758 0.53466607 0.59343034]
PS C:\Users\SAMUEL\Downloads\Calculo>
```



Como se puede observar, los datos obtenidos por nuestro modelo Weibull son:
[0.01980133 , 0.05823547 , 0.13151069 0.1974812 , 0.25918178 , 0.34295318 , 0.40844464, 0.47270758 , 0.53466607 , 0.59343034]

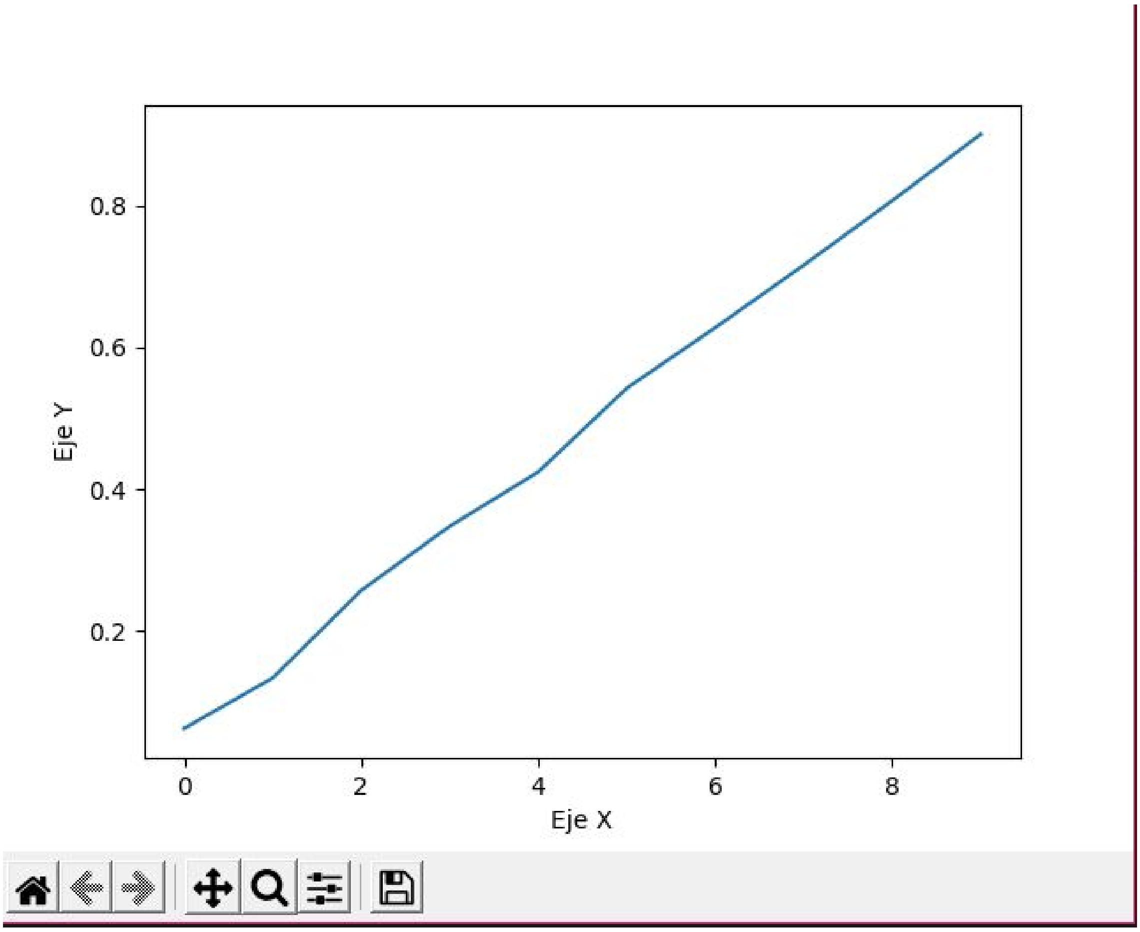
Graficando el modelo Higuchi:

C: > Users > SAMUEL > Downloads > Calculo > Higuchi.py > ...

```python
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 30 19:22:14 2021

@author: felip
"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.patches as mpatches

#DATOS EXPERIMENTALES TABLA GUIÓN#
y = np.linspace(0.1,1.0,10)
x = [0.2,0.3,0.47,0.55,0.6,0.7,0.75,0.8,0.85,0.9]

## ------------------- funcion Higuchi -------------------- ##

def Higuchi(y,x):
    x_2 = x*y**(0.5)
    return x_2

plt.plot(Higuchi(y,x),label='Higuchi')
plt.xlabel("Eje X")
plt.ylabel("Eje Y")
plt.show()

print("higuchi: \n ",Higuchi(y,x))
```

SALIDA TERMINAL CONSOLA DE DEPURACIÓN PROBLEMAS 10

```
PS C:\Users\SAMUEL\Downloads\Calculo>  c:; cd 'c:\Users\SAMUEL\Downloads\Calculo'; & 'python' 'c:\Users\SAMUEL\.vscode\extensions\ms-pyth
py'
higuchi:
  [0.06324555 0.13416408 0.2574296  0.34785054 0.42426407 0.54221767
 0.62749502 0.71554175 0.8063808  0.9       ]
PS C:\Users\SAMUEL\Downloads\Calculo>
```



Como se puede observar, los datos obtenidos por nuestro modelo Higuchi son: [0.06324555 , 0.13416408 , 0.2574296 , 0.34785054 , 0.42426407 , 0.54221767 , 0.62749502 , 0.71554175 , 0.8063808 , 0.9]

3. La función de Rosenbrock

La función de Rosenbrock está definida por :

$$f(x, y) = (x - 1)^2 + b(y - x^2)^2 \tag{3}$$

En donde $b = 10$

Implementación

C: > Users > SAMUEL > Downloads > Calculo > optimizacion.py > ...

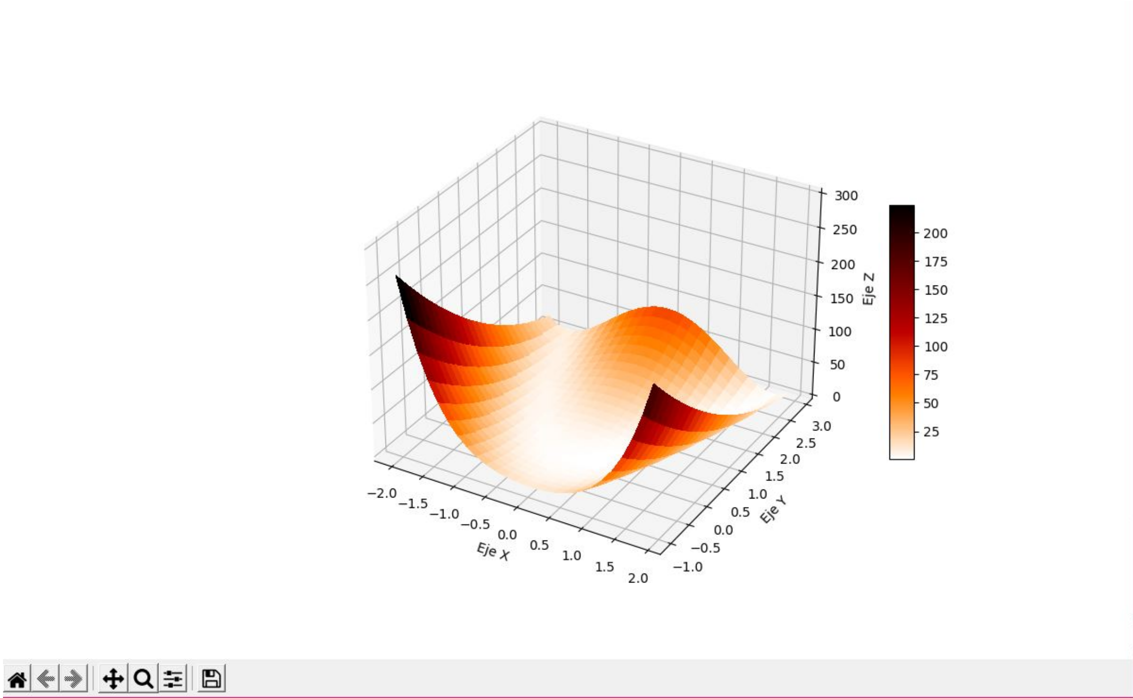
```python
# -*- coding: utf-8 -*-
"""
Created on Fri Apr 30 18:04:02 2021


"""

import numpy as np
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib.ticker import LinearLocator, FormatStrFormatter
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.patches as mpatches

## --------------------------------- Funcion Rosenbrock ------------------------ ##


b=10;

f = lambda x,y: (x-1)**2 + b*(y-x**2)**2;


## -------------------------------- Evaluar la funcion -------------------------- ##
X = np.arange(-2, 2, 0.15)
Y = np.arange(-1, 3, 0.15)
X, Y = np.meshgrid(X, Y)
Z = f(X,Y)
```

```python
## -------------------------------- Graficar ------------------------------------- ##

fig = plt.figure(figsize=(12, 7))
ax = fig.gca(projection='3d')
ros = ax.plot_surface(X, Y, Z, cmap=cm.gist_heat_r, linewidth=0, antialiased=False)

ax.set_xlabel('Eje X')
ax.set_ylabel('Eje Y')
ax.set_zlabel('Eje Z')

ax.set_zlim(0, 300)
fig.colorbar(ros, shrink=0.5, aspect=10)
plt.show()


## -----------------------------  Gradiente -------------------------------- ##

df = lambda x,y: np.array([2*(x-1) - 4*b*(y - x**2)*x, \
                          2*b*(y-x**2)])

F = lambda X: f(X[0],X[1])
dF = lambda X: df(X[0],X[1])

x_0 = np.array([-1.4,1.1])
print(F(x_0))
print(dF(x_0))

plt.figure(figsize=(12, 7))
plt.contour(X,Y,Z,200)
plt.plot([x_0[0]],[x_0[1]],marker='o',markersize=15, color ='r')

### -------------------- Encontrar la direccion de descenso --------- ##

fx = F(x_0);
gx = dF(x_0);
s = -gx;
print(s)

plt.figure(figsize=(12, 7))
plt.contour(X,Y,Z,200)
ns = np.sqrt(s[0]**2+s[1]**2);
plt.plot([x_0[0]],[x_0[1]],marker='o',markersize=15, color ='r')
plt.arrow(x_0[0],x_0[1],s[0]/ns,s[1]/ns, head_width=0.2, head_length=0.1, fc='r', ec='r')
```

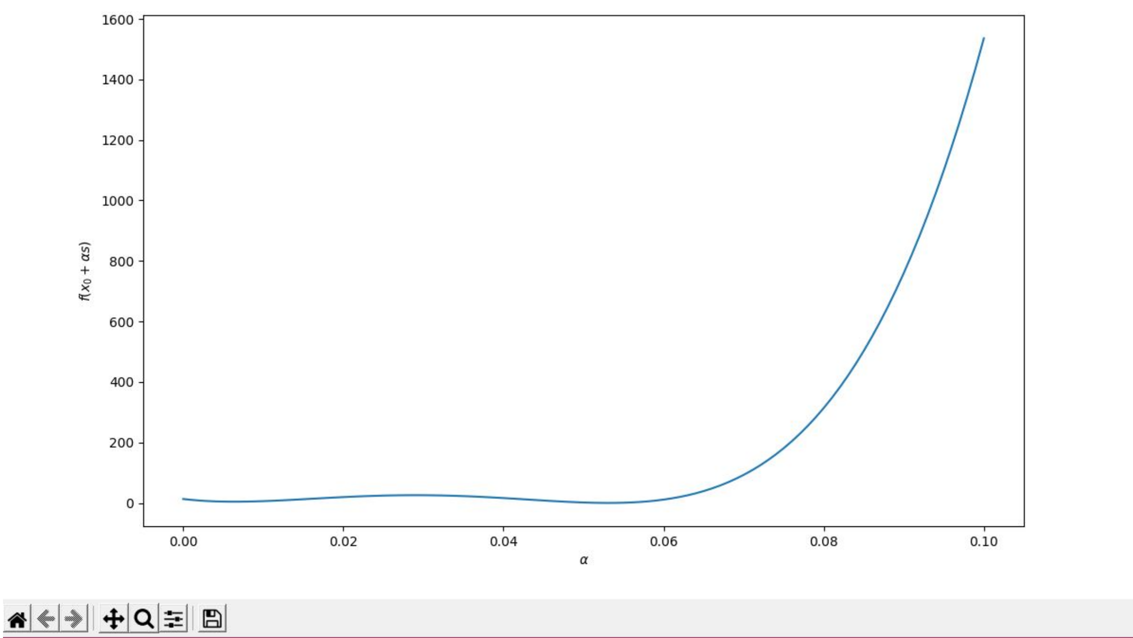
```python
## --------------------- ¿Hasta donde llega? ----------------- ##

al = np.linspace(0,0.1,101)
z = [F(x_0+a*s) for a in al]
figLS = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()

figLS = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.yscale('log')
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()


theta = 0.1
alpha = 1
tol = 1e-10
d = theta*np.dot(gx,s)

print([fx,fx+0.01*d])

figLS1 = plt.figure(figsize=(12, 7))
plt.plot(al,z)
plt.plot(al,[fx+a*d for a in al])

for i in range(10):

    if (alpha<=0.1):
        plt.plot(alpha,F(x_0+alpha*s),marker='x');
        plt.plot(alpha,fx + alpha*d,marker='o')
```

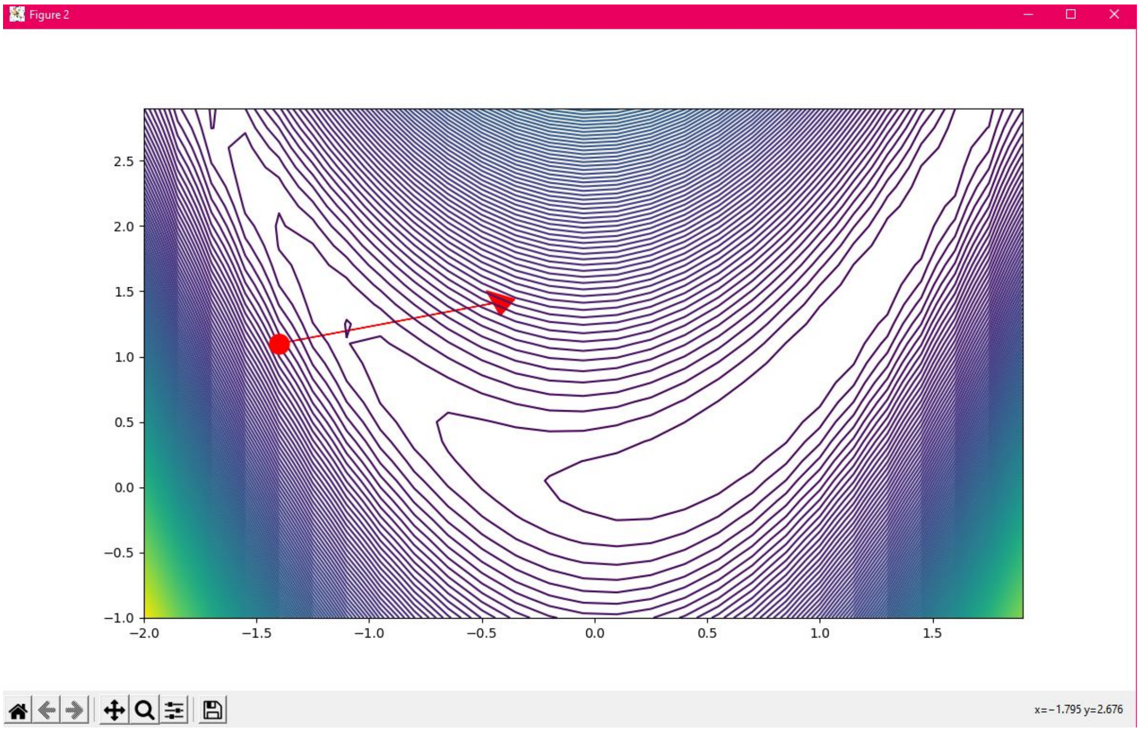
```python
plt.yscale('log')
plt.ylabel('$f(x_0+ \\alpha s)$')
plt.xlabel('$\\alpha$')
plt.show()

print ("Alpha \n", alpha)
```

Obteniendo los siguientes gráficos:

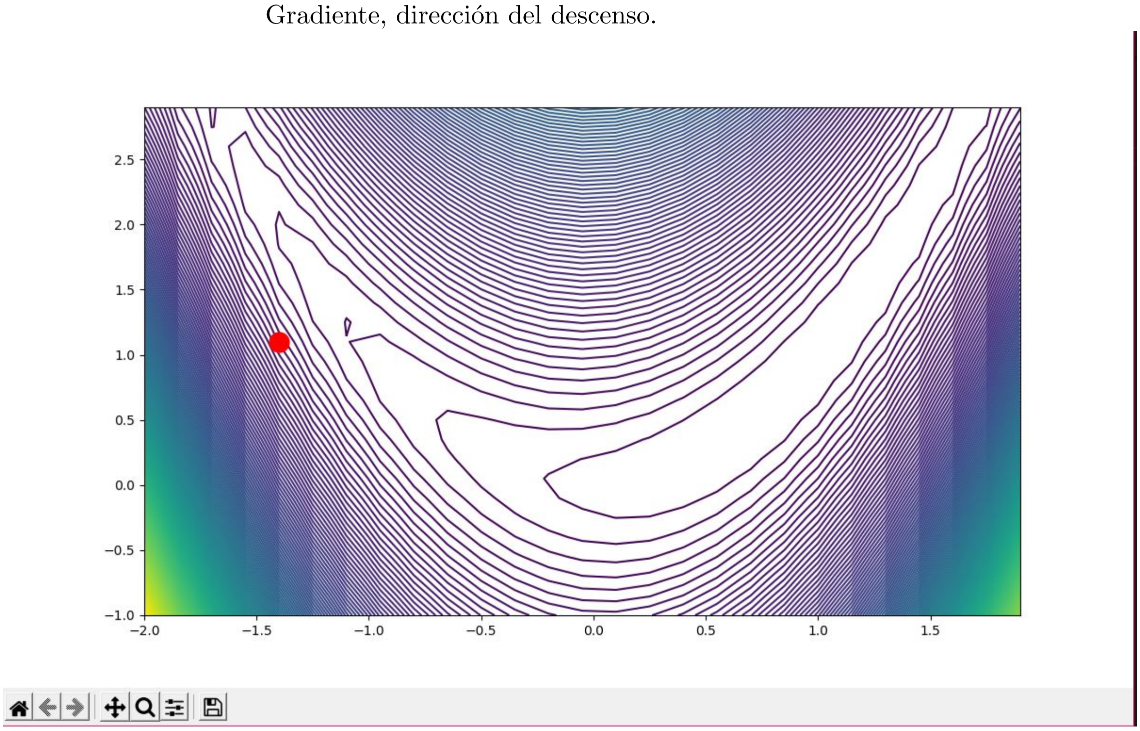


Evaluando función





Gradiente, dirección del descenso.



Hasta donde llega.