José Javier Alonso Ramos — Computación y Sistemas Inteligentes —

Aprendizaje Automático

Práctica 1

Ejercicio 1.- BÚSQUEDA ITERATIVA DE ÓPTIMOS

---

1. ***Implementar algoritmo de gradiente Descendente***

Descripción: con este algoritmo tratamos de avanzar hacia el mínimo local de una función más próximo a un punto inicial indicado.

Parámetros:

- **func:** indicamos la función sobre la que se aplicará el gradiente descendiente.

- **grad:** es un array de dos componentes de las cuales, la primera contiene la derivada parcial respecto a la primera variable de *func*, y la segunda la derivada parcial respecto a la segunda variable de *func*.

- **u:** primera coordenada del punto inicial a partir del cual se buscará el mínimo local.

- **v:** segunda coordenada del punto inicial a partir del cual se buscará el mínimo local.

- **maxIter:** número máximo de iteraciones que realizará el algoritmo. Funciona como condición de parada en caso de no encontrar antes el mínimo local.

- **epsilon:** número suficientemente pequeño según el cual consideramos que no ha habido mejora de una iteración a otra del algoritmo. Si la diferencia entre la interpretación de la función *func* en unas coordenas *(x,y)* y la interpretación de otras coordenadas *(x',y')* correspondientes a la siguiente iteración del algoritmo es menor que **epsilon** paramos el cálculo del mínimo. Por defecto le asignamos un valor de 10^{-14}.

$$f(x, y) - f(x', y') < epsilon$$

- **learning_rate:** el gradiente descendente nos indica hacia qué dirección debemos dirigirnos para tomar nuestra siguiente coordenada a considerar como mínimo. El learning_rate (η) indica en qué medida tomamos como válida esa dirección y, por tanto, cuánto avanzamos en esa dirección. Por defecto le asignamos un valor de 0.01.

Funcionamiento: mientras no cumplamos el límite de iteraciones y por cada iteración obtengamos una mejora notable, actualizaremos nuestras coordenadas *(x,y)* restándole el *gradiente* calculado multiplicado por el *learning-rate*.

$$(x', y') = (x, y) - \eta grad$$

. Al finalizar, el algoritmo devolverá:

- **w:** dupla con las coordenadas *(x,y)* del punto mínimo alcanzado.

- **it:** número de iteraciones que han sido necesarias para alcanzar el mínimo.

- **points2min:** conjunto de puntos *(x,y)* que hemos ido obteniendo en cada iteración del algoritmo.

---

2. ***Considerar la función*** $E(u, v) = (u^2e^v - 2v^2e^{-u})^2$ ***. Usar gradiente descendente para encontrar un mínimo de esta función, comenzando desde el punto*** $(u, v) = (1, 1)$ ***usando una tasa de aprendizaje*** $\eta = 0.01$***.***

- a) ***Calcular analíticamente y mostrar la expresión del gradiente de la función*** $E(u, v)$***.***

Gradiente de la función E(u,v):

$$grad(E(u, v)) = (\delta E_u(u, v), \delta E_v(u, v))$$

Derivada de E respecto a 'u'

$$\delta E_u = \frac{\delta}{\delta u}[(u^2e^v - 2v^2e^{-u})^2]$$

Aplicamos la regla de la potencia.

$$\delta E_u = 2(u^2e^v - 2v^2e^{-u})\frac{\delta}{\delta u}[u^2e^v - 2v^2e^{-u}]$$

Derivamos los términos por eseparado.

$$\delta E_u = 2(u^2e^v - 2v^2e^{-u})(e^v\frac{\delta}{\delta u}[u^2] - 2v^2\frac{\delta}{\delta u}[e^{-u}])$$

Resultado de la derivada parcial respecto 'u':

$$\delta E_u = 2(u^2e^v - 2v^2e^{-u})(2e^vu + 2v^2e^{-u})$$

Derivada de E respecto a 'v'

$$\delta E_v = \frac{\delta}{\delta u}[(u^2e^v - 2v^2e^{-u})^2]$$

Aplicamos la regla de la potencia.

$$\delta E_v = 2(u^2e^v - 2v^2e^{-u})\frac{\delta}{\delta u}[u^2e^v - 2v^2e^{-u}]$$

Derivamos los términos por eseparado.

$$\delta E_v = 2(u^2e^v - 2v^2e^{-u})(u^2\frac{\delta}{\delta u}[e^v] - 2e^{-u}\frac{\delta}{\delta u}[v^2])$$

Resultado de la derivada parcial respecto 'v':

$$\delta E_v = 2(u^2e^v - 2v^2e^{-u})(e^vu^2 - 4ve^{-u})$$

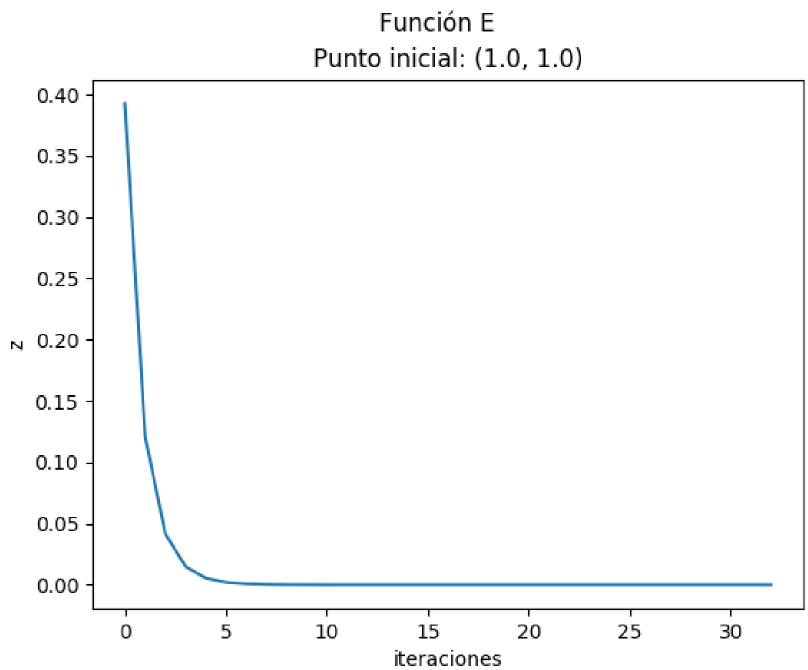
- b) ***¿Cuántas iteraciones tarda el algoritmo en obtener por primera vez un valor de*** $E(u, v)$ ***inferior a*** 10^{-14}***?***

Como mostramos en una tabla más adelante, se realizan ***33*** iteraciones.

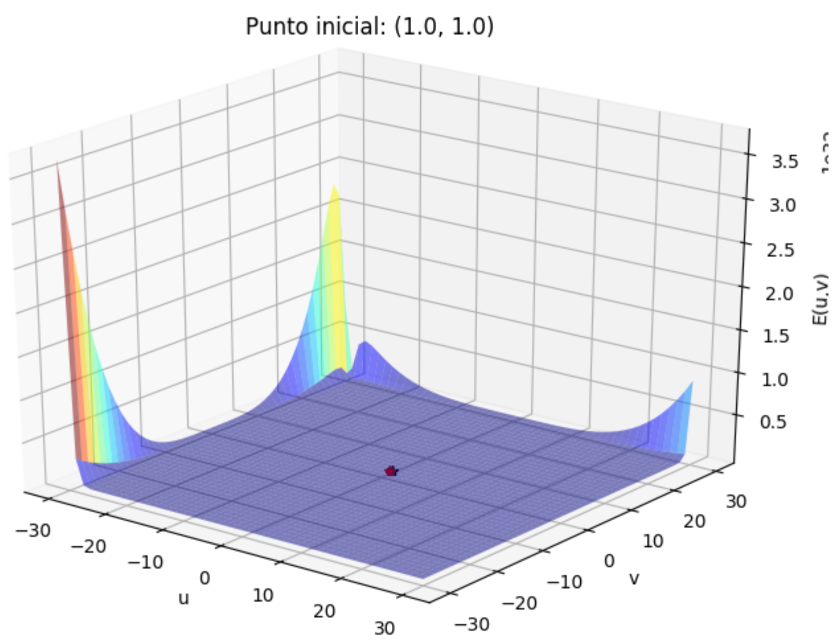
- c) ***¿En qué coordenadas*** (u, v) ***se alcanzó por primera vez un valor igual o menor a*** 10^{-14} ***en el apartado anterior?***

Las coordenadas ***(0.619207678450638, 0.9684482690100487)*** son las primeras en las que obtenemos un valor menor o igual a 10^{-14}.

Initial Point	u	v	lr	F(u,v)	iteraciones
[1.0, 1.0]	0.6192076	0.96844826	0.01	5.9973 x 10ˆ(-15)	33



Vemos como tiende a una asíntota horizontal cuando se aproxima al minímo.



Esta es una representación 3D de la gráfica donde se ven representados el punto incial como una estrella negra y el mínimo alcanzado como una estrella roja.

También marcamos los puntos intermedios recorridos en color verde pero, debido a la proximidad del punto inicial con el mínimo, no se distinguen bien.

---

3. ***Considerar ahora la función*** $F(x, y) = x^2 + 2y^2 + 2sin(2\pi x)sin(2\pi y)$

- a) ***Usar gradiente descendente para minimizar esta función. Usar como punto inicial*** $(x_0 = 0.1, y_0 = 0.1)$***, tasa de aprendizaje*** $\eta = 0.01$ ***y un máximo de 50 iteraciones. Generar un gráfico de cómo desciende el valor de la función con las iteraciones. Repetir el experimento pero usando*** $\eta = 0.1$***, comentar las diferencias y su dependencia de*** η***.***

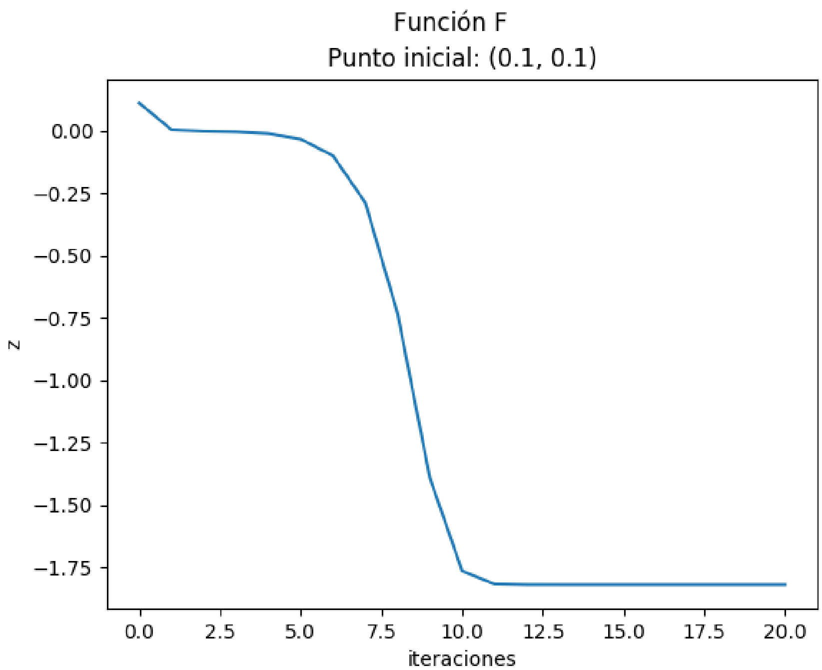
$$\delta F_u(u, v) = 4\pi sin(2\pi v)cos(2\pi u) + 2u$$

$$\delta F_v(u, v) = 4\pi sin(2\pi u)cos(2\pi v) + 4v$$

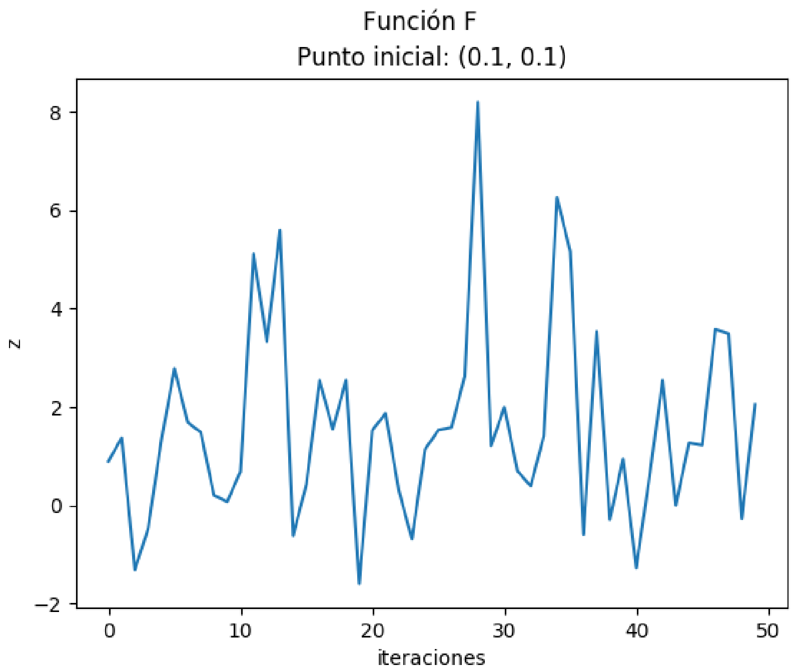
$$grad(F_{(u,v)}) = (\delta F_u, \delta F_v)$$

= =

Para $\eta = 0.01$:



Para $\eta = 0.1$:



Comparación:

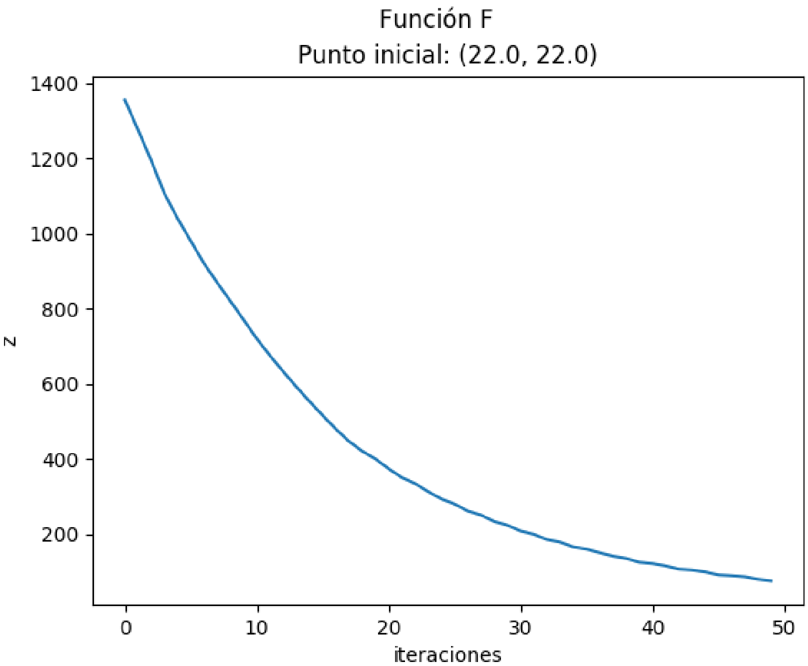
Initial Point	u	v	lr	F(u,v)	iteraciones
[0.1, 0.1]	0.243805	-0.237926	0.01	-1.820079	21
[0.1, 0.1]	0.898964	-0.211171	0.1	2.048279	50

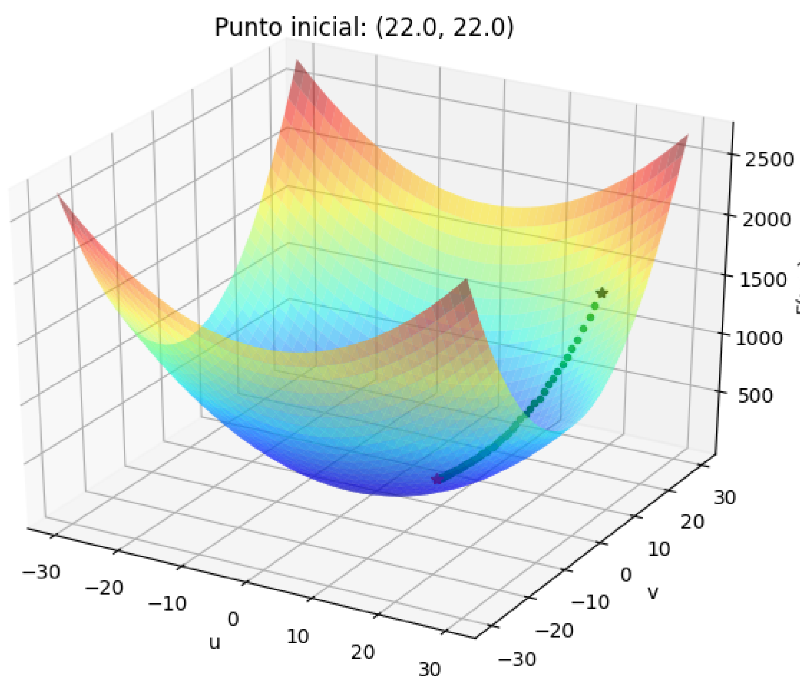
En la gráfica con $\eta = 0.1$ vemos que los valores de la función varían bruscamente. Esto es debido al alto valor de η.
Al usar un $\eta = 0.1$ avanzamos demasiado en la dirección indicada por el gradiente de modo que, cuando se aproxima al mínimo, el algoritmo oscila a su alrededor y no consigue converger. Con $\eta = 0.01$, en cambio, avanzamos distancias más pequeñas que nos permiten ir convergiendo en el mínimo.

- b) ***Obtener el valor mínimo y los valores de las variables*** (x, y) ***en donde se alcanzan cuando el punto de inicio se fija:*** $(0.1, 0.1), (1, 1), (-0.5, -0.5), (-1, -1)$***. Generar una tabla con los valores obtenidos***

Para mostrar como el algoritmo dibuja los puntos intermedios recorridos he añadido a la lista de puntos iniciales $(22.0, 22.0)$ en cuya representación gráfica

veremos que no consigue convergir a un mínimo y agota las 50 posibles iteraciones.
En la ejecución del código se mostrarán las gráficas para todos los puntos.





Initial Point	u	v	lr	F(u,v)	iteraciones
[0.1, 0.1]	0.243805	-0.237926	0.01	-1.820079	21
[1.0, 1.0]	1.218070	0.712812	0.01	0.593269	17
[-0.5, -0.5]	-0.731377	-0.237855	0.01	-1.332481	17
[-1, -1]	-1.218070	-0.712812	0.01	0.593269	17
[22.0, 22.0]	7.604608	3.133128	0.01	76.556035	50

---

4. ***¿Cuál sería su conclusión sobre la verdadera dificultad de encontrar el mínimo global de una función arbitraria?***

La dificultad recae en lo bueno que sea el punto inicial escogido (que el camino marcado por el gradiente descendente sea hacia el mínimo globlal y no hacia un mínimo local), en que el *learning rate* escogido sea acertado (permita converger adecuadamente en el mínimo y a su vez no sea demasiado pequeño y haga que la ejecución del algoritmo sea extremadamente larga) y en ser capaces de solucionar o evitar estancamientos en mínimos locales.

Ejercicio 2.- BÚSQUEDA ITERATIVA DE ÓPTIMOS

Este ejercicio ajusta modelos de regresión a vectores de características extraidos de imágenes de digitos manuscritos. En particular se extraen dos característcas concretas: el valor medio del nivel de gris y simetría del número respecto de su eje vertical. Solo se seleccionarán para este ejercicio las imágenes de los números 1 y 5.

1. ***Estimar un modelo de regresión lineal a partir de los datos proporcionados de dichos números (Intensidad promedio, Simetria) usando tanto el algoritmo de la pseudo-inversa como Gradiente descendente estocástico (SGD). Las etiquetas serán {-1, 1}, una para cada vector de cada uno de los números. Pintar las soluciones obtenidas junto con los datos usados en el ajuste. Valorar la bondad del resultado usando E in y E out (para E out calcular las predicciones usando los datos del fichero de test).***

Para el ***Gradiente Descendente Estocástico*** hemos escogido un tamaño de Minibatch = 64.

$$w_j = w_j - \eta \sum_{n \in Minibatch} [x_{nj}(h(x_n) - y_n)]$$

j hace referencia a las columnas
n hace referencia a las filas dentro del Minibatch

- $h(x_n)$***:*** nuestra aproximación al valor y asociado a x_n.
- y_n***:*** valor real asociado a x_n.
- x_{nj}***:*** columna del atributo j perteneciente al minibatch n.
- η***:*** learning-rate.
- w_j***:*** peso asociado al atributo j.

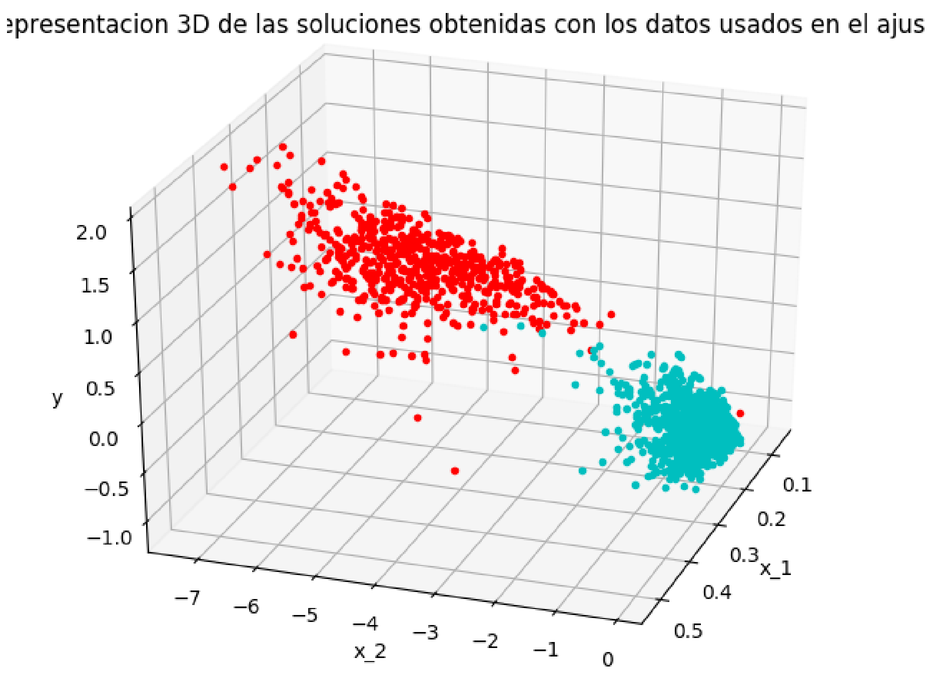
Para realizar la ***Pseudo-inversa*** nos hemos ayudado de una función del módulo *numpy* (*numpy.linalg.pinv(X)*).

$$w = X^{\dagger}Y$$

Para calcular el ***Error medio*** realizamos la sumatoria de la diferencia cuadrática entre nuestra aproximación (Xw) y el valor real asociado a X (Y) dividido por el número de datos.

$$Err = \frac{\sum_{n=1}^{N}[Xw - Y]^2}{N}$$

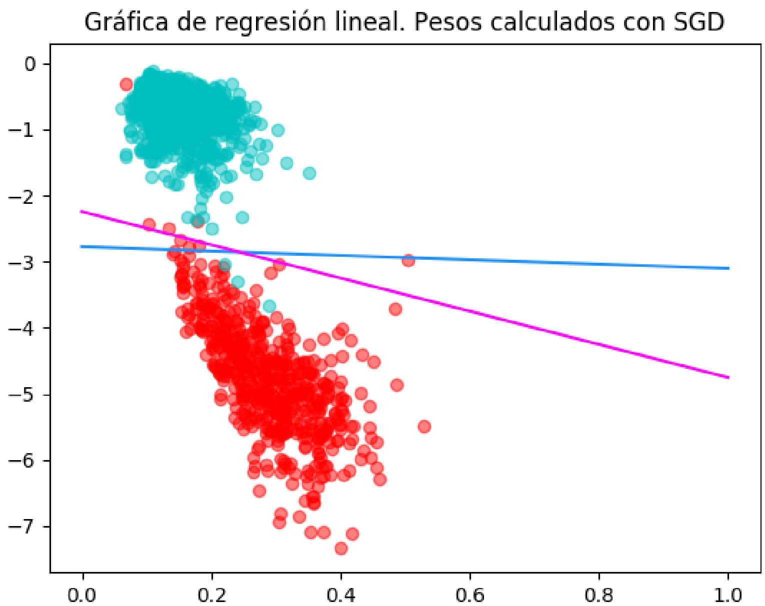
Acontinuación mostramos en una gráfica 3D la distribución de todos los valores de la muestra X leidos. Pintaremos de rojo aquellos a los que nuestro algoritmo les haya asignado la etiqueta 1 y de azul a los que se les haya asignado la etiqueta -1.



Mostramos el error obtenido con SGD y Pseudo-inversa:

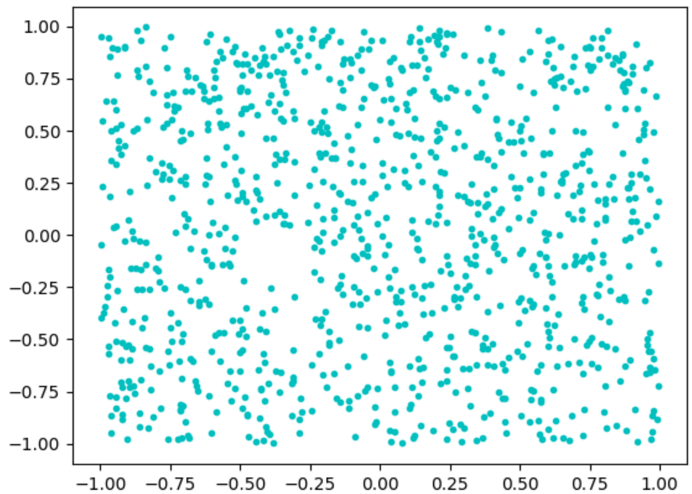
	SGD	Pseudo-inversa
E_{in}	0.08262032686662257	0.07918658628900395
E_{out}	0.13317636750014467	0.13095383720052584

Dibujaremos las rectas asociadas a la regresión:
De color azul la correspondiente al SGD
De color magenta la correspondiente a la Pseudoinversa
Tener en cuenta que los ejes del gráfico indican las dos coordenadas de x



2. ***En este apartado exploramos como se transforman los errores E in y E out cuando aumentamos la complejidad del modelo lineal usado. Ahora hacemos uso de la función simula_unif (N, 2, size) que nos devuelve N coordenadas 2D de puntos uniformemente muestreados dentro del cuadrado definido por [size, size]×[-size,-size]***

- a) ***Generar una muestra de entrenamiento de N = 1000 puntos en el cuadrado X = [-1, 1]×[-1, 1]. Pintar el mapa de puntos 2D.***

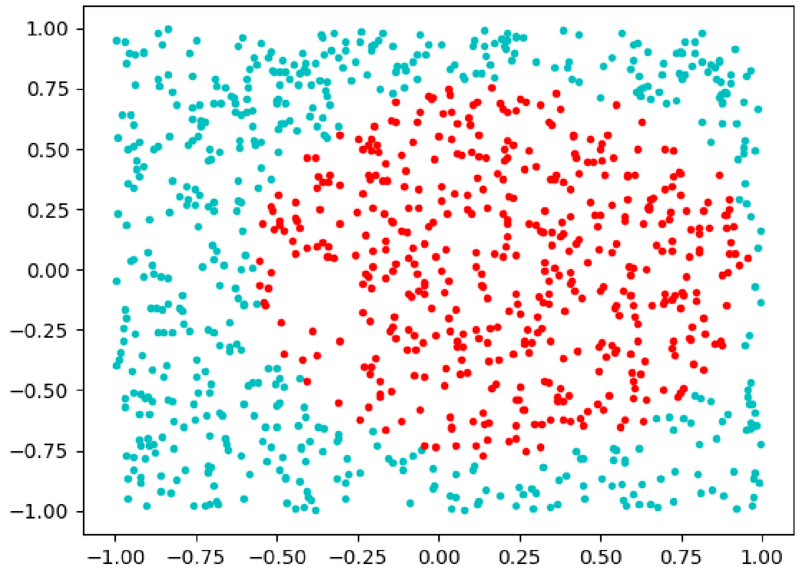


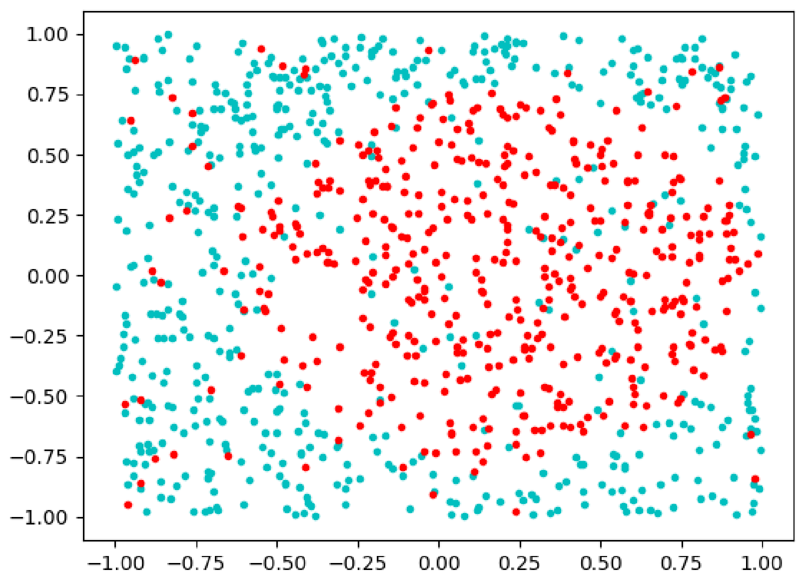
- b) ***Consideremos la función*** $f(x_1, x_2) = sign((x_1 - 0.2)^2 + x_2^2 - 0.6)$ ***que usaremos para asignar una etiqueta a cada punto de la muestra anterior. Introducimos ruido sobre las etiquetas cambiando aleatoriamente el signo de un 10% de las mismas. Pintar el mapa de etiquetas obtenido.***

Para modificar el 10% de la muestra:

- Calculamos las etiquetas correspondientes a cada dato generado en el apartado *2 a)* con la función proporcionada.
- Añadimos a la matriz de puntos 2D una nueva columna que contendrá las etiquetas recién calculadas haciéndolas coincidir con su correspondiente punto 2D.
- Generamos números aleatorios no repetidos en el rango [0, número de datos), es decir, generamos aleatoriamente los índices que referencian a los datos que vamos a alterar.
- Cambiamos de signo las etiquetas referenciadas por estos índices.

Mapa de puntos según etiquetas y mapa de puntos con ruido:





- c) ***Usando como vector de características*** $(1, x_1, x_2)$ ***ajustar un modelo de regresion lineal al conjunto de datos generado y***

estimar los pesos w. Estimar el error de ajuste E_{in} usando Gradiente Descendente Estocástico (SGD).

$$E_{in} = 0.9266173649533643$$

- d) ***Ejecutar todo el experimento definido por (a)-(c) 1000 veces (generamos 1000 muestras diferentes) y***
  - ***Calcular el valor medio de los errores E in de las 1000 muestras.***
  - ***Generar 1000 puntos nuevos por cada iteración y calcular con ellos el valor de E out en dicha iteración. Calcular el valor medio de E out en todas las iteraciones.***

Error medio tras 1000 iteraciones:
E_{in} medio: 0.9273397180289497
E_{out} medio: 0.9320711718563939

- e) ***Valore que tan bueno considera que es el ajuste con este modelo lineal a la vista de los valores medios obtenidos de E_{in} y E_{out}***

Debido a la distribución de los puntos no podemos realizar un buen ajuste con un modelo lineal. El haber alterado un 10% de la muestra no es motivo sufuciente para obtener unos resultados tan malos.
No podemos tomar por bueno un ajuste con un error $E \simeq 0.9$.

Para intentar realizar un mejor ajuste deberíamos probar con un modelo no lineal.

Ejercicio 3.- BONUS - NEWTON'S METHOD

$$\triangle w = -H^{-1} \bigtriangledown E_{in}(w)$$

En este método acompaña al learning-rate (η) la matriz Hessiana que, al igual que η, nos indicará en qué medida debemos hacer caso a lo que nos indica el gradiente.

La matriz Hessiana está compuesta por lo siguiente:

$$H = \begin{pmatrix} \delta F_{uu} & \delta F_{uv} \\ \delta F_{vu} & \delta F_{vv} \end{pmatrix}$$

Siendo:

$$\delta F_{uu} = 2 - 8\pi^2 sin(2\pi v) sin(2\pi u)$$

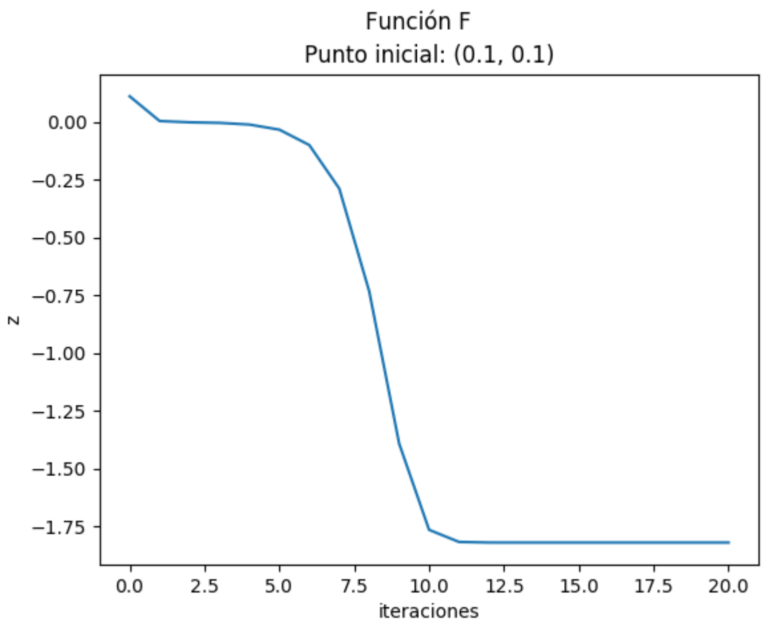
$$\delta F_{uv} = 8\pi^2 cos(2\pi u) cos(2\pi v)$$

$$\delta F_{vv} = 4 - 8\pi^2 sin(2\pi u) sin(2\pi v)$$

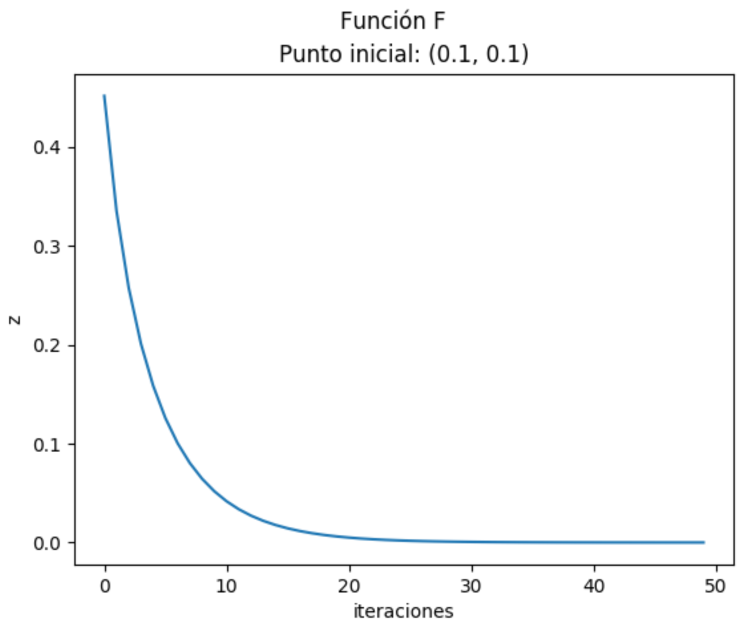
$$\delta F_{vu} = 8\pi^2 cos(2\pi v) cos(2\pi u)$$

Vamos a comparar los resultados con los obtenidos en el ejercicio 1.3:

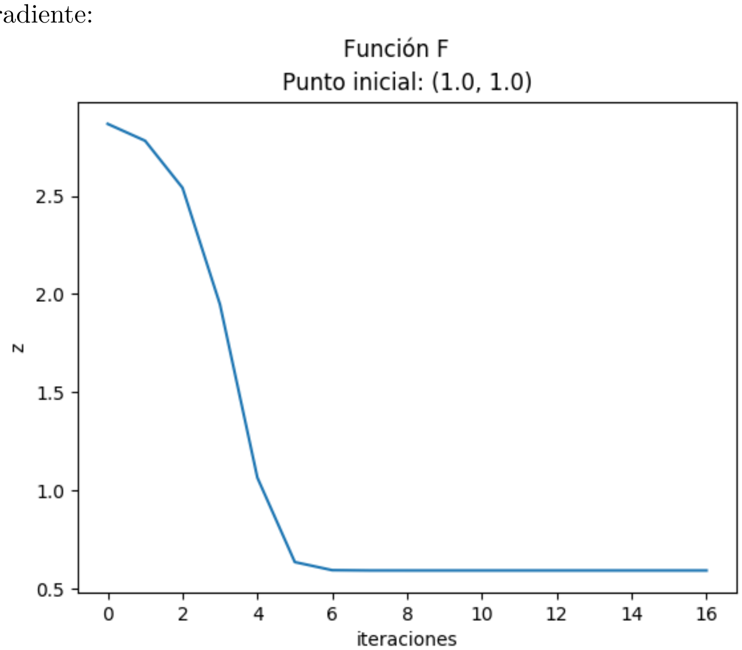
- ***Para el punto (0.1, 0.1):***
  Gradiente:



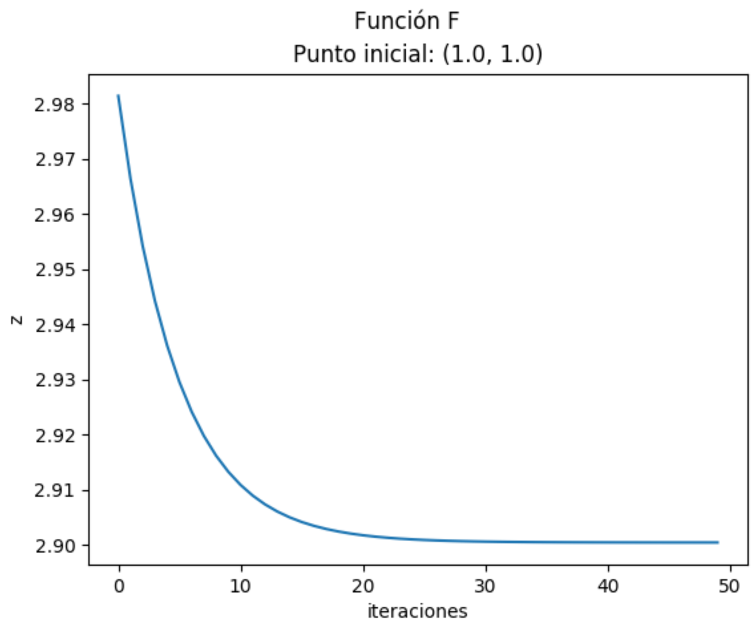
  Newton:



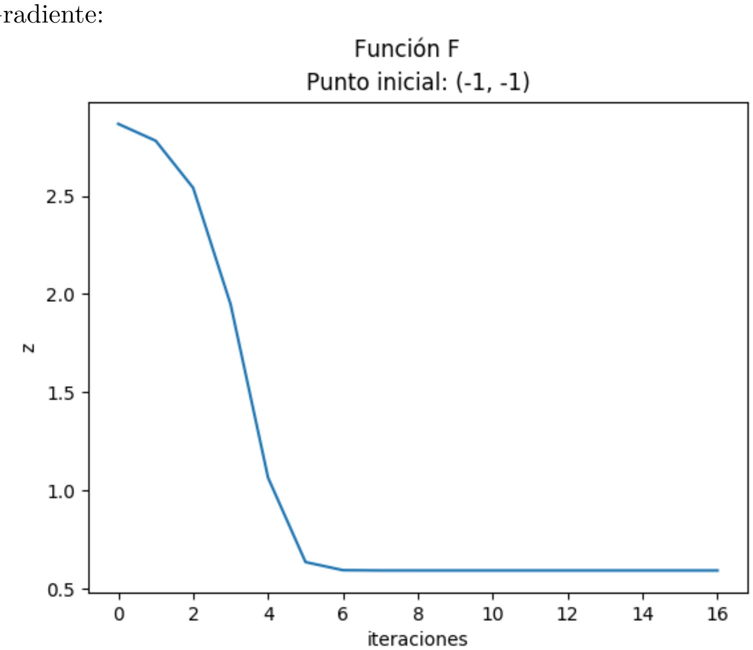
- ***Para el punto (1, 1):***
  Gradiente:



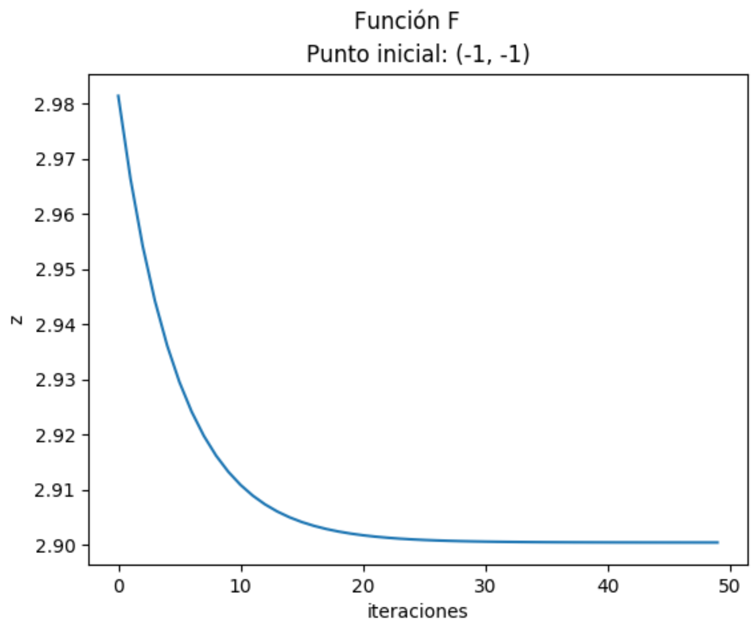
Newton:



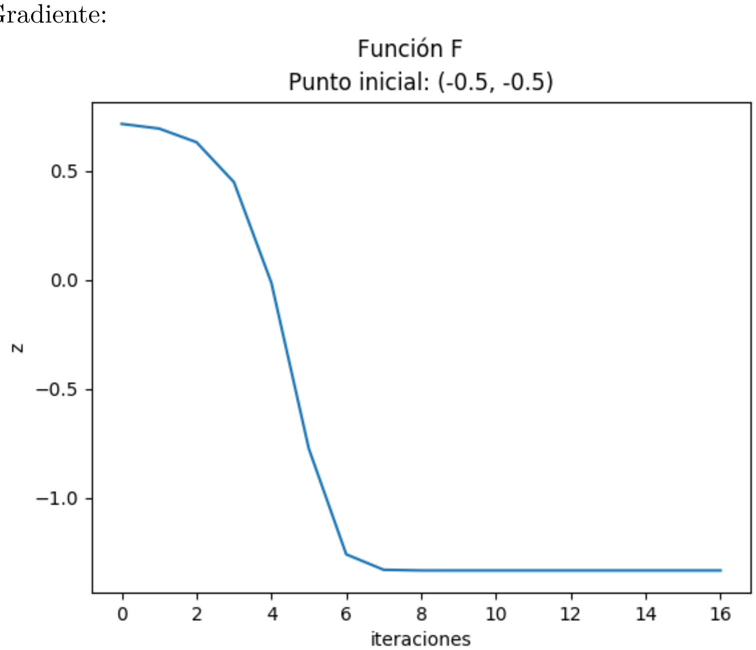
- ***Para el punto (-1, -1):***
  Gradiente:



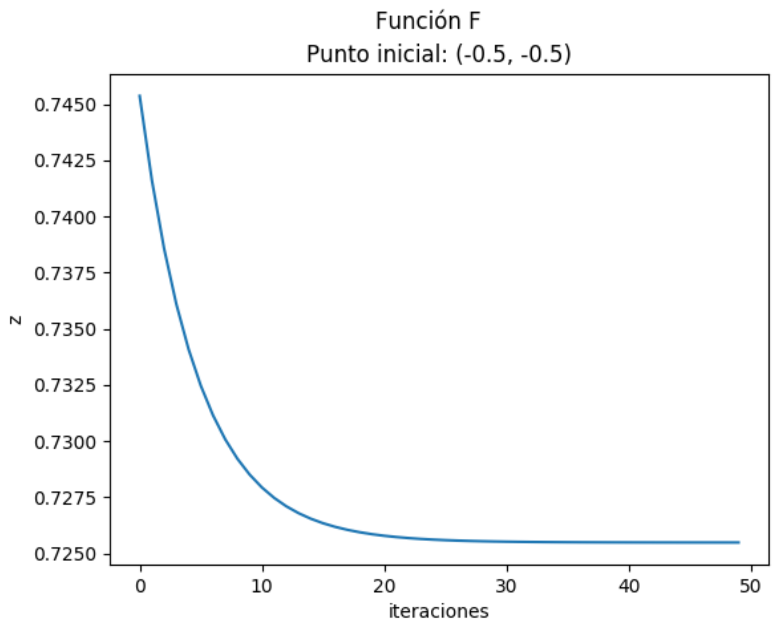
Newton:



- ***Para el punto (-0.5, -0.5):***
  Gradiente:



Newton:



Initial Point	u	v	lr	F(u,v)	iteraciones
[0.1, 0.1]	0.000368	0.000364	0.1	0.000011	50
[1.0, 1.0]	0.949409	0.974715	0.1	2.900408	50
[-0.5, -0.5]	-0.475244	-0.487869	0.1	0.725483	50
[-1, -1]	-1.949409	-0.974715	0.1	2.900408	50

El método de Newton tiende a converger a 0 aunque en los puntos $(\pm 1, \pm 1)$ no lo logre. Por lo tanto el gradiente descendente consigue mejores resultados pudiendo alcanzar valores negaivos.