AG2 - Actividad Guiada 2

Nombre: Juan Francisco Vallalta
Link: https://colab.research.google.com/drive/xxxxxxxxxxxxxxxxxxxxxxxxx
Github: https://github.com/xxxxx/AlgoritmosOptimizacion

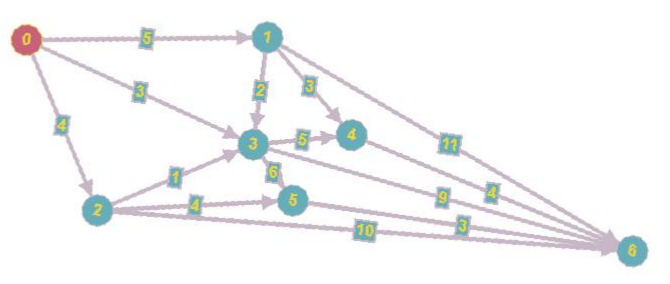
In [1]:
```
import math
```

Programación Dinámica. Viaje por el rio

- **Definición**: Es posible dividir el problema en subproblemas más pequeños, guardando las soluciones para ser utilizadas más adelante.
- **Características** que permiten identificar problemas aplicables:
  -Es posible almacenar soluciones de los subproblemas para ser utilizados más adelante
  -Debe verificar el principio de optimalidad de Bellman: “en una secuencia optima de decisiones, toda sub-secuencia también es óptima” (*)
  -La necesidad de guardar la información acerca de las soluciones parciales unido a la recursividad provoca la necesidad de preocuparnos por la complejidad espacial (cuantos recursos de espacio usaremos)

Problema

En un río hay **n** embarcaderos y debemos desplazarnos río abajo desde un embarcadero a otro. Cada embarcadero tiene precios diferentes para ir de un embarcadero a otro situado más abajo. Para ir del embarcadero i al j, puede ocurrir que sea más barato hacer un trasbordo por un embarcadero intermedio k. El problema consiste en determinar la combinación más barata.



- Consideramos una tabla TARIFAS(i,j) para almacenar todos los precios que nos ofrecen los embarcaderos.
- Si no es posible ir desde i a j daremos un valor alto para garantizar que ese trayecto no se va a elegir en la ruta óptima (modelado habitual para restricciones)

In [2]:
```
#Viaje por el rio - Programación dinámica
################################################################################

TARIFAS = [
[0,5,4,3,float("inf"),999,999],   #desde nodo 0
[999,0,999,2,3,999,11],  #desde nodo 1
[999,999, 0,1,999,4,10], #desde nodo 2
[999,999,999, 0,5,6,9],
[999,999, 999,999,0,999,4],
[999,999, 999,999,999,0,3],
[999,999,999,999,999,999,0]
]

#999 se puede sustituir por float("inf") del modulo math
TARIFAS
```

Out[2]:
```
[[0, 5, 4, 3, inf, 999, 999],
 [999, 0, 999, 2, 3, 999, 11],
 [999, 999, 0, 1, 999, 4, 10],
 [999, 999, 999, 0, 5, 6, 9],
 [999, 999, 999, 999, 0, 999, 4],
 [999, 999, 999, 999, 999, 0, 3],
 [999, 999, 999, 999, 999, 999, 0]]
```

In [3]:
```
#Calculo de la matriz de PRECIOS y RUTAS
#  PRECIOS - contiene la matriz del mejor precio para ir de un nodo a otro
#  RUTAS   - contiene los nodos intermedios para ir de un nodo a otro
################################################################
def Precios(TARIFAS):
################################################################
  #Total de Nodos
  N = len(TARIFAS[0])

  #Inicialización de la tabla de precios
  PRECIOS = [ [9999]*N for i in [9999]*N]  #n x n
  RUTA = [ [""]*N for i in [""]*N]

  #Se recorren todos los nodos con dos bucles(origen - destino)
  #  para ir construyendo la matriz de PRECIOS
  for i in range(N-1):
    for j in range(i+1, N):
      MIN = TARIFAS[i][j]
      RUTA[i][j] = i

      for k in range(i, j):
        if PRECIOS[i][k] + TARIFAS[k][j] < MIN:
            MIN = min(MIN, PRECIOS[i][k] + TARIFAS[k][j] )
            RUTA[i][j] = k
        PRECIOS[i][j] = MIN

  return PRECIOS,RUTA
```

In [4]:
```
PRECIOS,RUTA = Precios(TARIFAS)
#print(PRECIOS[0][6])

print("PRECIOS")
for i in range(len(TARIFAS)):
  print(PRECIOS[i])

print("\nRUTA")
for i in range(len(TARIFAS)):
  print(RUTA[i])
```

```
PRECIOS
[9999, 5, 4, 3, 8, 8, 11]
[9999, 9999, 999, 2, 3, 8, 7]
[9999, 9999, 9999, 1, 6, 4, 7]
[9999, 9999, 9999, 9999, 5, 6, 9]
[9999, 9999, 9999, 9999, 9999, 999, 4]
[9999, 9999, 9999, 9999, 9999, 9999, 3]
[9999, 9999, 9999, 9999, 9999, 9999, 9999]

RUTA
['', 0, 0, 0, 1, 2, 5]
['', '', 1, 1, 1, 3, 4]
['', '', '', 2, 3, 2, 5]
['', '', '', '', 3, 3, 3]
['', '', '', '', '', 4, 4]
['', '', '', '', '', '', 5]
['', '', '', '', '', '', '']
```

In [5]:
```
#Calculo de la ruta usando la matriz RUTA
def calcular_ruta(RUTA, desde, hasta):
  if desde == RUTA[desde][hasta]:
  #if desde == hasta:
    #print("Ir a :" + str(desde))
    return desde
  else:
    return str(calcular_ruta(RUTA, desde, RUTA[desde][hasta])) +  ',' + str(RUTA[desde][hasta])

print("\nLa ruta es:")
calcular_ruta(RUTA, 0,6)
```

```
La ruta es:
```

Out[5]:
```
'0,2,5'
```

Descenso del gradiente

In [6]:
```
import math                        #Funciones matematicas
import matplotlib.pyplot as plt  #Generacion de gráficos (otra opcion seaborn)
import numpy as np                 #Tratamiento matriz N-dimensionales y otras (fundamental!)
#import scipy as sc

import random
```

Vamos a buscar el minimo de la funcion paraboloide :

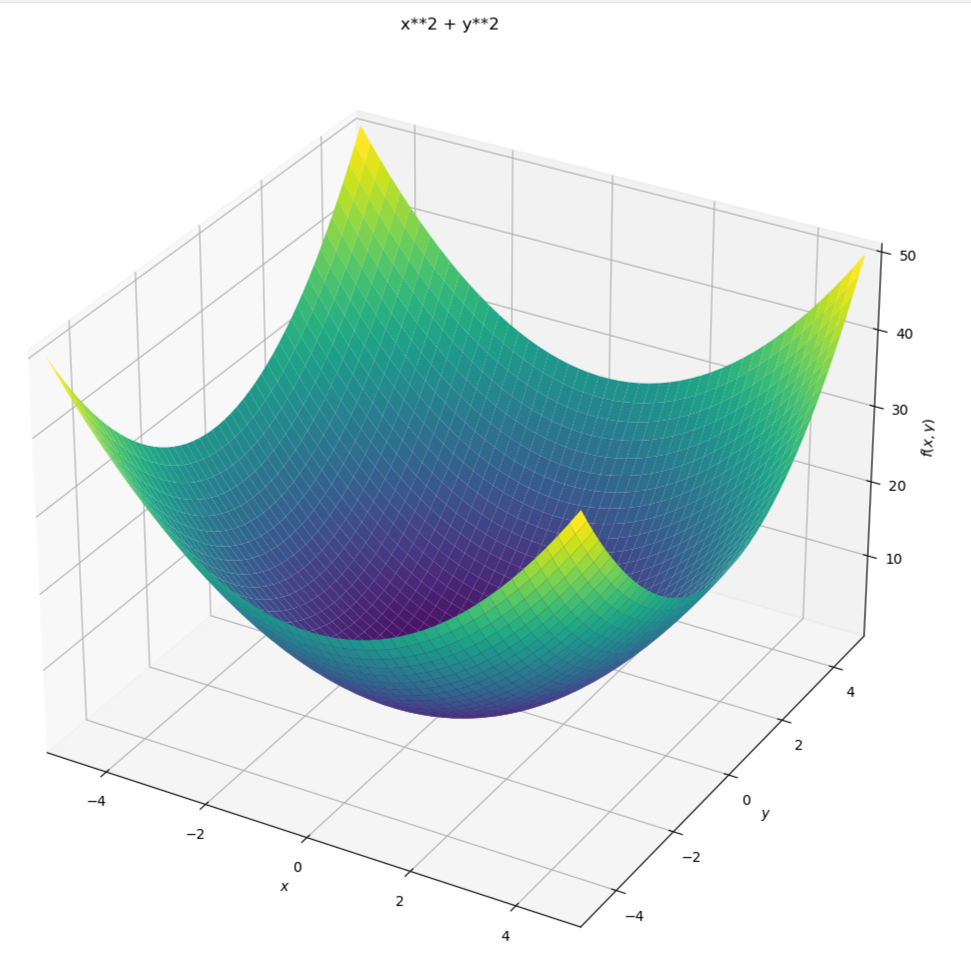
$$f(x) = x^2 + y^2$$

Obviamente se encuentra en (x,y)=(0,0) pero probaremos como llegamos a él a través del descenso del gradiante.

In [13]:
```
#Definimos la funcion
#Paraboloide
f  = lambda X:      X[0]**2 + X[1]**2    #Funcion
df = lambda X: [2*X[0] , 2*X[1]]         #Gradiente

df([1,2])
```

Out[13]:
```
[2, 4]
```

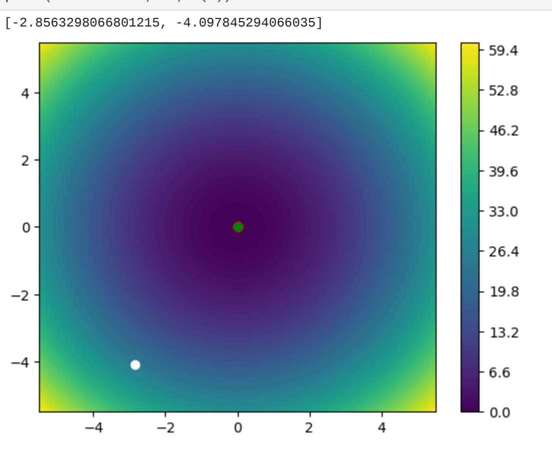
In [14]:
```
from sympy import symbols
from sympy.plotting import plot
from sympy.plotting import plot3d
x,y = symbols('x y')
plot3d(x**2 + y**2,
       (x,-5,5),(y,-5,5),
       title='x**2 + y**2',
       size=(10,10))
```



Out[14]:
```
<sympy.plotting.plot.Plot at 0x1f0fb73b7f0>
```

In [15]:
```
#Prepara los datos para dibujar mapa de niveles de Z
resolucion = 100
rango=5.5

X=np.linspace(-rango,rango,resolucion)
Y=np.linspace(-rango,rango,resolucion)
Z=np.zeros((resolucion,resolucion))
for ix,x in enumerate(X):
  for iy,y in enumerate(Y):
    Z[iy,ix] = f([x,y])

#Pinta el mapa de niveles de Z
plt.contourf(X,Y,Z,resolucion)
plt.colorbar()

#Generamos un punto aleatorio inicial y pintamos de blanco
P=[random.uniform(-5,5  ),random.uniform(-5,5 ) ]
plt.plot(P[0],P[1],"o",c="white")
print(P)

#Tasa de aprendizaje. Fija. Sería más efectivo reducirlo a medida que nos acercamos.
TA=0.5

#Iteraciones:50
for _ in range(1000):
  grad = df(P)
  #print(P,grad)
  P[0],P[1] = P[0] - TA*grad[0] , P[1] - TA*grad[1]
  plt.plot(P[0],P[1],"o",c="red")
  TA=TA*0.9


#Dibujamos el punto final y pintamos de verde
plt.plot(P[0],P[1],"o",c="green")
plt.show()
print("Solucion:" , P , f(P))
```

```
[-2.8563298066801215, -4.097845294066035]
```



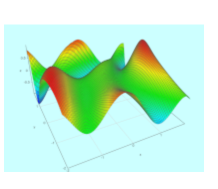

```
Solucion: [0.0, 0.0] 0.0
```

In [16]:
```
Z
```

Out[16]:
```
array([[60.5       , 59.29012346, 58.10493827, ..., 58.10493827,
        59.29012346, 60.5       ],
       [59.29012346, 58.08024691, 56.89506173, ..., 56.89506173,
        58.08024691, 59.29012346],
       [58.10493827, 56.89506173, 55.70987654, ..., 55.70987654,
        56.89506173, 58.10493827],
       ...,
       [58.10493827, 56.89506173, 55.70987654, ..., 55.70987654,
        56.89506173, 58.10493827],
       [59.29012346, 58.08024691, 56.89506173, ..., 56.89506173,
        58.08024691, 59.29012346],
       [60.5       , 59.29012346, 58.10493827, ..., 58.10493827,
        59.29012346, 60.5       ]])
```

Reto

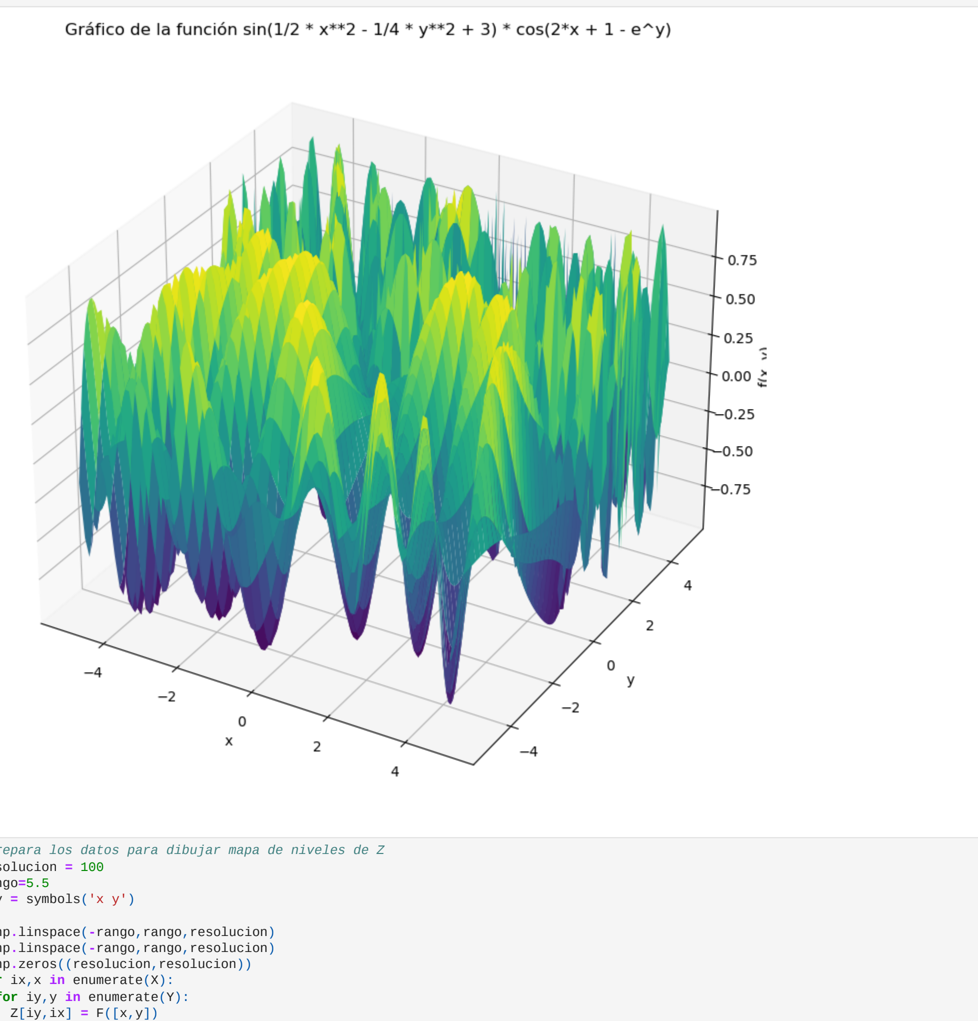
$$f(x) = sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2 * x + 1 - e^y)$$



In [18]:
```
#Definimos la funcion
F= lambda X: math.sin(1/2 * X[0]**2 - 1/4 * X[1]**2 + 3) *math.cos(2*X[0] + 1 - math.exp(X[1]) )
DF = lambda X: [-2*math.cos(1/2+X[0]**2-1/4*X[1]**2+3)*math.sin(2*X[0]+1-math.exp(X[1])),   math.cos(1/2*X[0]**2-1/4*X[1]**2+3)*math.sin(2*X[0]+1-math.exp(X[1]))*math.exp(X[1])]

DF([1,2])
```

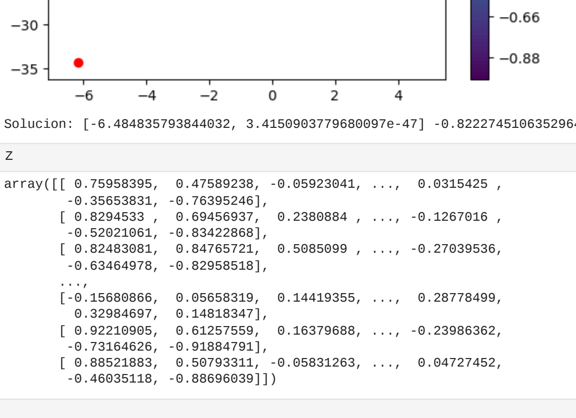
Out[18]:
```
[1.7758622527756773, -5.612946787444883]
```

In [19]:
```
import numpy as np
import matplotlib.pyplot as plt

# Definimos la función a graficar
def f(x, y):
    return np.sin(1/2 * x**2 - 1/4 * y**2 + 3) * np.cos(2*x + 1 - np.exp(y))

# Creamos una malla de puntos (x, y)
x_vals = np.linspace(-5, 5, 100)
y_vals = np.linspace(-5, 5, 100)
x, y = np.meshgrid(x_vals, y_vals)

# Evaluamos la función en cada punto de la malla
z = f(x, y)

# Creamos la figura y la superficie 3D
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')
ax.plot_surface(x, y, z, cmap='viridis')

# Etiquetas de los ejes y título
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('f(x, y)')
ax.set_title('Gráfico de la función sin(1/2 * x**2 - 1/4 * y**2 + 3) * cos(2*x + 1 - e^y)')

# Mostramos el gráfico
plt.show()
```



In [36]:
```
#Prepara los datos para dibujar mapa de niveles de Z
resolucion = 100
rango=5.5
x,y = symbols('x y')

X=np.linspace(-rango,rango,resolucion)
Y=np.linspace(-rango,rango,resolucion)
Z=np.zeros((resolucion,resolucion))
for ix,x in enumerate(X):
  for iy,y in enumerate(Y):
    Z[iy,ix] = F([x,y])

#Pinta el mapa de niveles de Z
plt.contourf(X,Y,Z,resolucion)
plt.colorbar()

#Generamos un punto aleatorio inicial y pintamos de blanco
P2=[random.uniform(-5,5  ),random.uniform(-5,5 ) ]
plt.plot(P2[0],P2[1],"o",c="white")
print(P2)

#Tasa de aprendizaje. Fija. Sería más efectivo reducirlo a medida que nos acercamos.
TA=1

#Iteraciones:50
for _ in range(1000):
  grad = DF(P2)
  #print(P,grad)
  P2[0],P2[1] = P2[0] - TA*grad[0] , P[1] - TA*grad[1]
  plt.plot(P2[0],P2[1],"o",c="red")
  TA=TA*0.9


#Dibujamos el punto final y pintamos de verde
plt.plot(P2[0],P2[1],"o",c="green")
plt.show()
print("Solucion:" , P2 , F(P2))
```

```
[-4.347570458244997, 3.796566516203187]
```




```
Solucion: [-6.484835793844032, 3.4150903779680097e-47] -0.8222745106352964
```

In [26]:
```
Z
```

Out[26]:
```
array([[ 0.75958395,  0.47589238, -0.05923041, ...,  0.0315425 ,
        -0.35653831, -0.76395246],
       [ 0.8294533 ,  0.69456937,  0.2380884 , ..., -0.1267016 ,
        -0.52021061, -0.83422868],
       [ 0.82483081,  0.84765721,  0.5085099 , ..., -0.27039536,
        -0.63464978, -0.82958518],
       ...,
       [-0.15680866,  0.05658319,  0.14419355, ...,  0.28778499,
         0.32984697,  0.14818347],
       [ 0.92210905,  0.61257559,  0.16379688, ..., -0.23986362,
        -0.73164626, -0.91884791],
       [ 0.88521883,  0.50793311, -0.05831263, ...,  0.04727452,
        -0.46035118, -0.88696039]])
```

In [ ]:

In [ ]: