AG2 - Actividad Guiada 2

Nombre: Esmarlin Julissa Moreno Nivar
Github: https://github.com/julissrock/03MIAR-Algoritmos-de-Optimizacion/blob/main/Algoritmos_AG2.ipynb

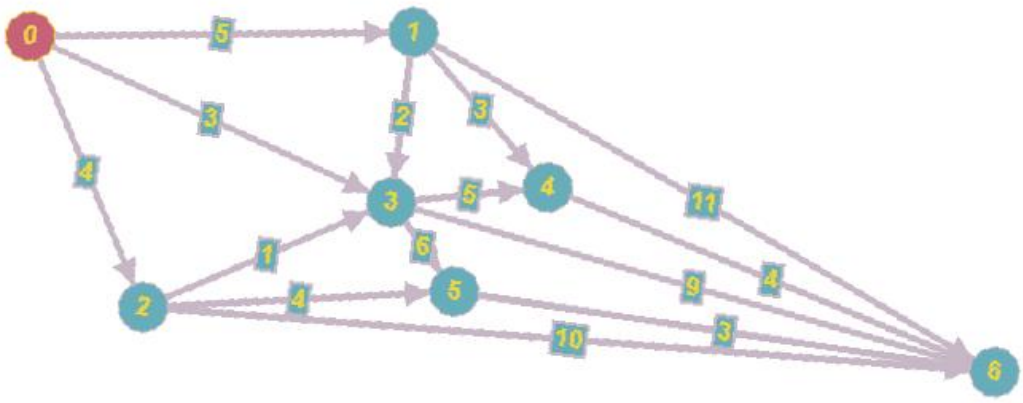
In [1]:
```
import math
```

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[d

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 (fund
#import scipy as sc

import random
```

Vamos a buscar el minimo de la funcion paraboloide :

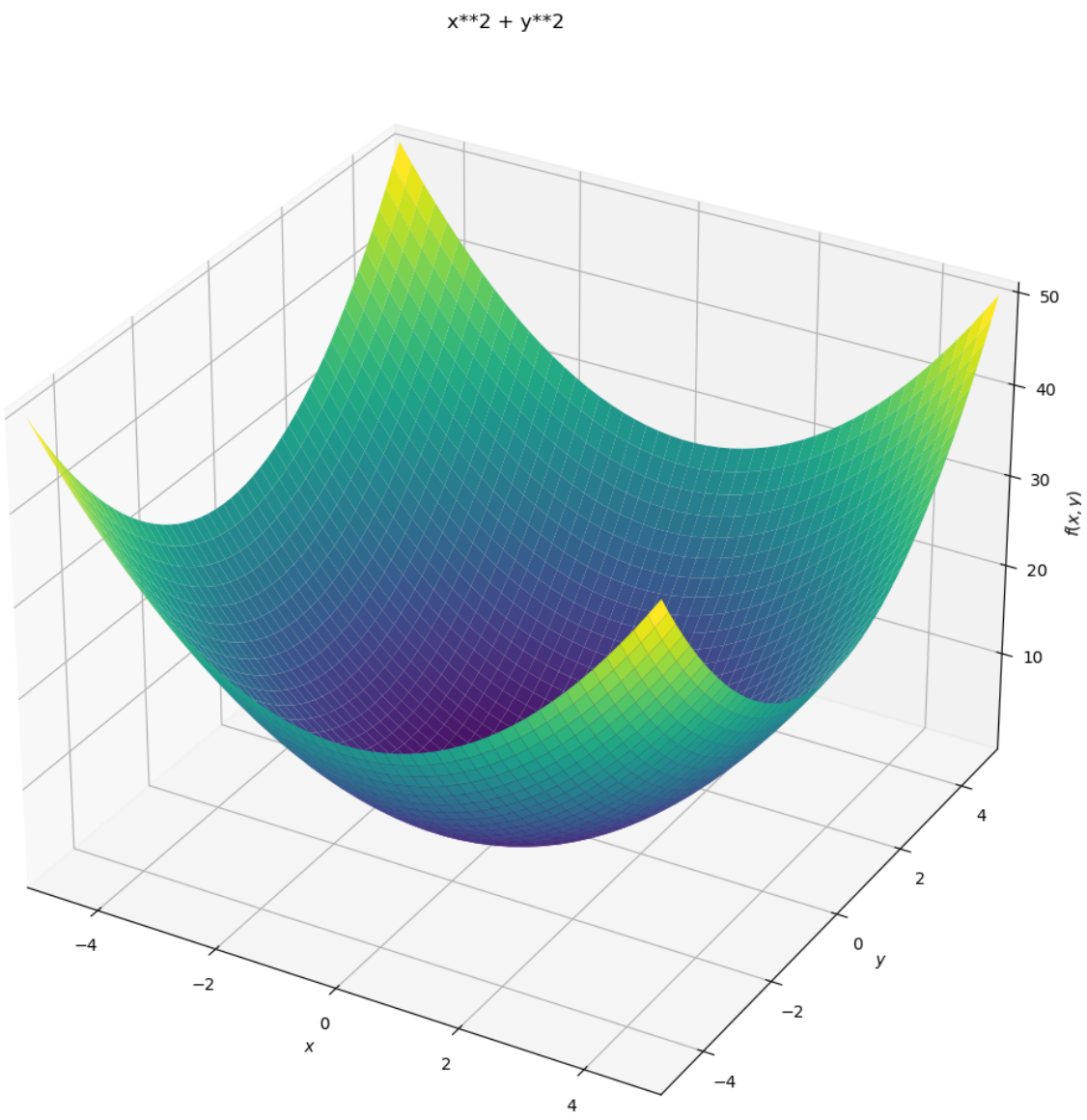
$$f(x) = x^2 + y^2$$

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

In [7]:
```
#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[7]:
```
[2, 4]
```

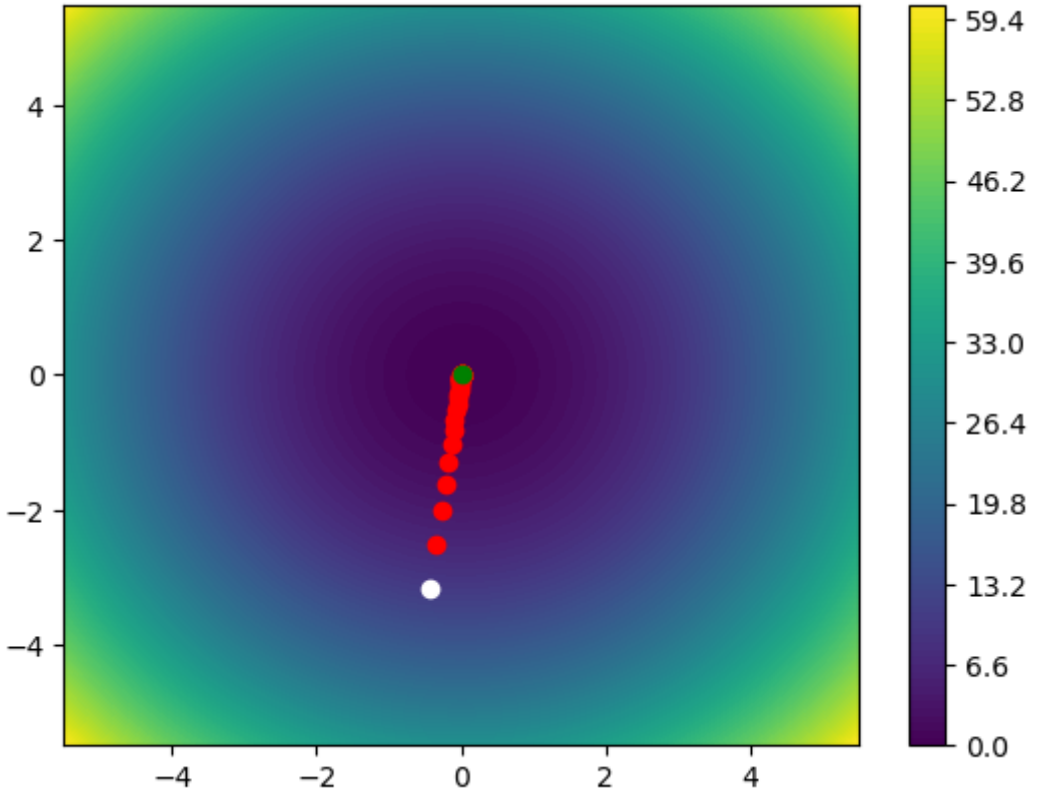
In [8]:
```
from sympy import symbols
from sympy.plotting import plot
from sympy.plotting import plot3d
x,y = symbols('x y')
plot3d(x**2 + y**2,
       (x,-5,5),(y,-5,5),
       title='x**2 + y**2',
       size=(10,10))
```



Out[8]:
```
<sympy.plotting.plot.Plot at 0x1fa8f9a3400>
```

In [9]:
```
#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")

#Tasa de aprendizaje. Fija. Sería más efectivo reducirlo a medida que nos acercamos
TA=0.1

#Iteraciones:50
for _ in range(50):
  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")

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



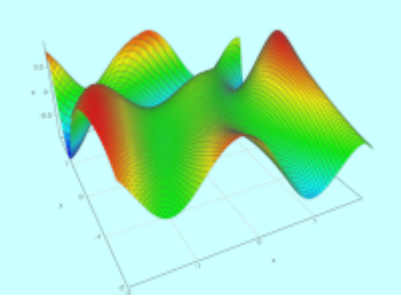

```
Solucion: [-6.1321277845915374e-06, -4.5066230487791225e-05] 2.068568121545283e-09
```

In [10]:
```
Z
```

Out[10]:
```
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

Optimización de la función $f(x) = sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2 * x + 1 - e^y)$ mediante el algoritmo por descenso del gradiente.



In [11]:
```
"""
La función f(x, y) se define en la función f().
El gradiente de la función f(x, y) se define en la función grad_f().
El algoritmo de descenso del gradiente se define en la función gradient_descent().
Los valores de X y Y se inicializan en 0 y 0, respectivamente.
"""

def f(x, y):
    """Calcula f(x, y) = sin(0.5 * x^2 - 0.25 * y^2 + 3) * cos(2*x + 1 - e^y)"""
    return math.sin(0.5 * x**2 - 0.25 * y**2 + 3) * math.cos(2 * x + 1 - math.exp(

def grad_f(x, y):
    """Calcula el gradiente de f(x, y)"""
    return [
        math.cos(0.5 * x**2 - 0.25 * y**2 + 3) * (1 + x),
        -0.25 * math.sin(0.5 * x**2 - 0.25 * y**2 + 3) * math.cos(2 * x + 1 - math
    ]

def gradient_descent(x, y, learning_rate, iterations):
    """Aplica descenso de gradiente a f(x, y)"""
    for _ in range(iterations):
        gradient = grad_f(x, y)
        x -= learning_rate * gradient[0]
        y -= learning_rate * gradient[1]
    return x, y

x, y = gradient_descent(0, 0, 0.01, 1000)
print(x, y)
```

```
1.9070010264178119 0.650961780390601
```