

AG2 – Actividad Guiada 2

03MAIR – Algoritmos de Optimización

Agenda

- Programación dinámica
- Búsqueda en grafos, ramificación y poda.
- Descenso del gradiente

Agenda

1. Repaso de cuestiones Pendientes / Recordatorios / Mejoras /Aportaciones
2. Pivote para quick sort
3. Recursividad no siempre mejora!
4. Dos puntos más cercanos. Fuerza bruta vs Divide y Vencerás
5. Viaje por el rio. Programación Dinámica
6. Problema de asignación de tareas. Ramificación y Poda (Foro)


Foro. Aportaciones en el foro.

- Mejor **calidad** que cantidad
- Nuevo tema 2ª semana. Fichas sobre **problemas tipo**. Buscar problemas generales

<input type="checkbox"/> FORO	DESCRIPCIÓN	PUBLICACIONES TOTALES	RESPUESTAS PARA MÍ NO LEIDAS	PARTICIPANTES TOTALES
<input type="checkbox"/> Cuestiones de la asignatura(No evaluable)	En este foro se recogerán las participaciones que no serán evaluables.	5		3
<input type="checkbox"/> Aportaciones extraordinarias(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al contenido de la asignatura al margen de los temas de debate.	45		10
<input type="checkbox"/> Foro para Tema 1 de debate semana 15-julio(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al tema de debate de la semana 15-jul	12		9
<input type="checkbox"/> Foro para Tema 2 de debate semana 22-jul(Evaluable)	En este foro se recogerán las aportaciones de los alumnos relativas al tema de debate de la semana 22-jul	11		11

Foro. Aportaciones de código en el foro.

- Si es un código completo, añadir enlace de github(en una carpeta diferente de AG1...
- Si es un una porción de código, añadir el código en el texto con otro tipo de letra



Sergio Andrés Gutiérrez Rojas

RE: Los dos puntos más próximos

Dado a que todos empezaron por 1D y para eventualmente no repetir aporte de 2D(pares de puntos) por fuerza fruta:

Aquí lo que llevo de la actividad:

<https://github.com/sergioguti805/03MAIR-Algoritmos-de-optimizacion/blob/master/EVALUABLES/PuntosCercanos.ipynb>

por supuesto me pondre a terminar todos los casos para completar toda la actividad.

✧ Asunto

RE: Los dos puntos más próximos

Mensaje

Rich text editor toolbar with icons for bold, italic, underline, text color, background color, link, unlink, list, indent, outdent, and others. The link icon is highlighted with a red box.

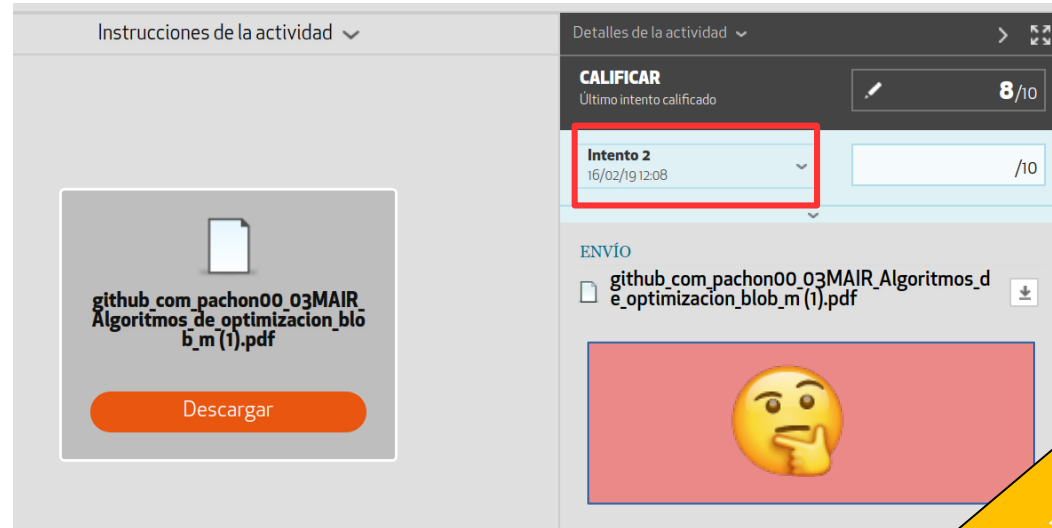
Text input field containing: **Mi algoritmo**

Ruta: p



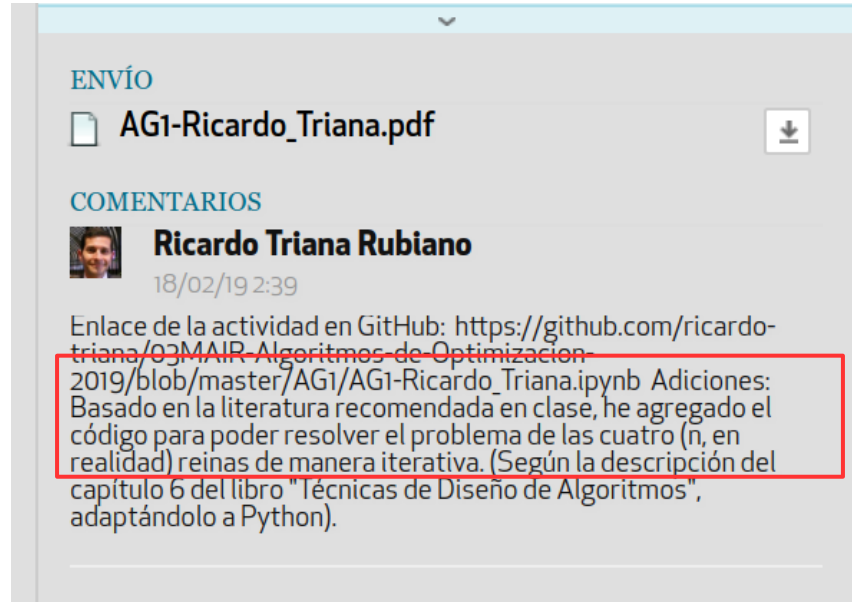
Actividades Guiadas.

- Si hay aportaciones, **añadir explicación**
- Si es la 2ª entrega, **añadir explicación y espera a que te califique la primera**
- Si tienes dudas de que la 2ª entrega sea valida, **enviame un correo.**



Actividades Guiadas.

- Primero practicar.
- Después Investigar, analizar, criticar y referenciar(autor y obra)

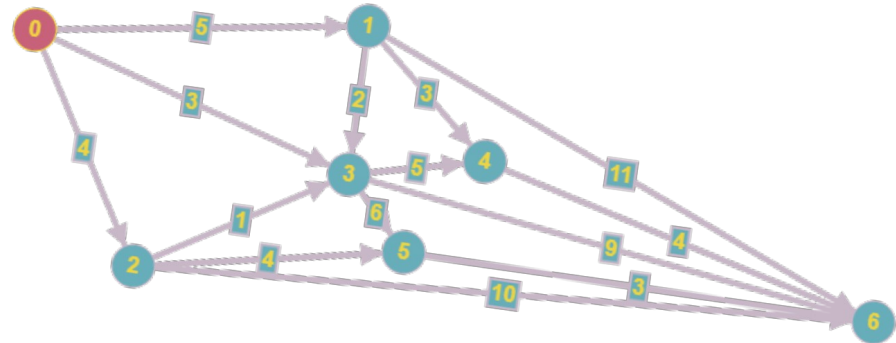


Programación dinámica (I)

Problema: Viaje por el río

- Consideramos una tabla $T(i,j)$ 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)
- Establecer una tabla intermedia para de soluciones óptimas parciales para ir desde i a j .

$$C(i,j) = \min \{T(i,j) , C(i,k)+T(k,j) \text{ para todo } i < k \leq j \}$$



Programación dinámica (II)

Operaciones

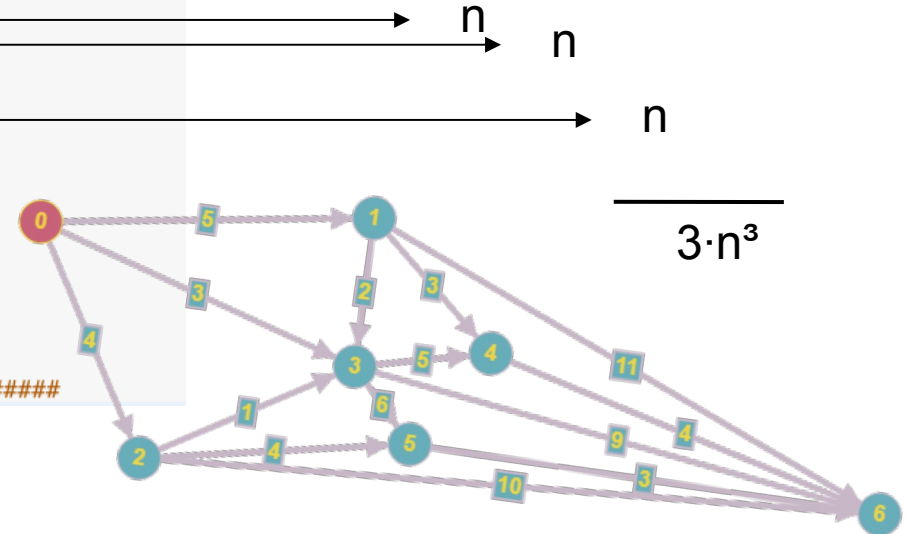
```
#####
def Precios(TARIFAS):
#####
    #Total de Nodos
    N = len(TARIFAS[0])

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

    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
#####
```




$$C(i,j) = \min \{T(i,j) , C(i,k)+T(k,j) \text{ para todo } i < k \leq j \}$$

Ramificación y Poda.

Problema: Asignación de tareas

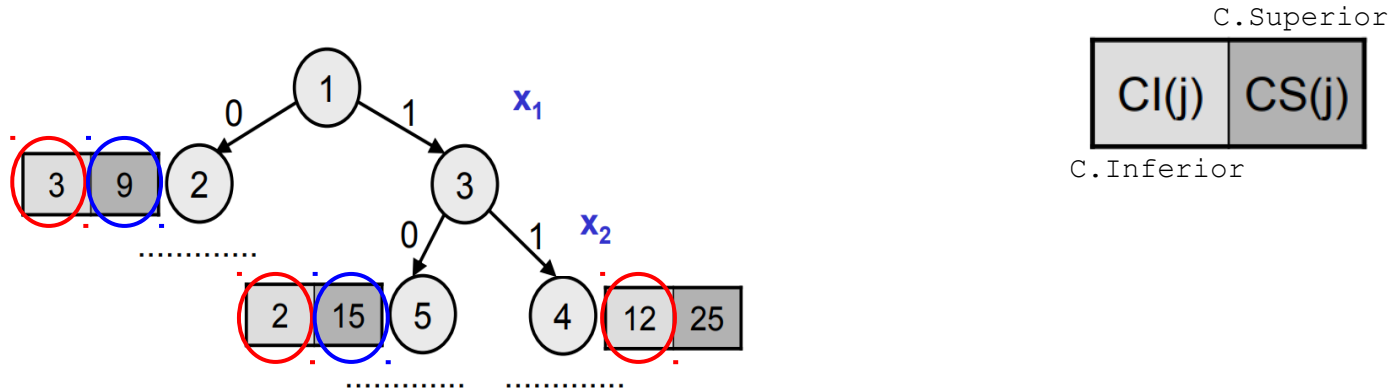
- El problema consiste en maximizar el rendimiento en cuanto a la asignación de N tareas a N agentes. Cada tarea solo puede ser asignado a un agente.
- Los beneficios que se obtienen al realizar la **tarea 1** al **agente A** es 9
- La matriz de beneficios es la que se muestra en la figura
- Aplicando Ramificación y Poda, obtener la asignación que maximice los beneficios.



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Algoritmos de Búsqueda. Técnicas de Ramificación y Poda(V)

- Estrategia de poda: Poda i si $CS(i) \leq CI(j)$ Para algún j ya generado



$C=12$ (max. de las cotas inferiores)


El nodo 2 no puede mejorar(valor 9) las peores predicciones de 4(valor 12 = max{todo CI() })

El nodo 5 quizá pueda mejorar al nodo 4

Ramificación y Poda.

Problema: Asignación de tareas

- P.ej. Si asignamos el agente A a la tarea1
 - Cota Inferior para la tarea2 es 4
 - Cota Superior para la tarea2 es 8



	Job 1	Job 2	Job 3	Job 4
A	9	2	7	8
B	6	4	3	7
C	5	8	1	8
D	7	6	9	4

Ramificación y Poda.

Problema: Asignación de tareas

- ¿Cómo diseñamos un estado?
- ¿Como expandimos nodos (ramificación)?
- ¿Como podamos?
- ¿Como recorreremos el árbol completo de estados?
- ¿Qué complejidad tiene? (*)

(*) es difícil calcular el número de operaciones exacto. Calcular el **mejor** caso y el **peor** caso.

Peor caso = tengo que expandir todos los niveles

Mejor caso = puedo podar todos los nodos de un nivel menos uno.

Ramificación y Poda. Practica

#Del libro Fundamentos de algoritmia - G. Brassard & P. Bratley pg 349, sec. 9.7.1

'''Hay que asignar n tareas a n agentes, de forma que a cada agente le corresponda una tarea. Se dispone de una tabla de costes: el coste para el agente i de realizar la tarea j es $c_{ij} \geq 0$. Se busca la asignación que reduzca el coste total.
Diferentes aplicaciones,

Ejemplo de tabla de costes para $n=4$. Agentes: a, b, c y d . Tareas: 1,2,3 y 4
'''

```
#  TAREAS
#  A
#  G
#  E
#  N
#  T
#  E
#  S
```

```
COSTES= [ [11,12,18,40],
           [14,15,13,22],
           [11,17,19,23],
           [17,14,20,28]]
```

La tupla: (1, 0, 3, 2) representa la solución :
Agente 0 a la tarea 1


"	1	"	0
"	2	"	3
"	3	"	2

Ramificación y Poda. Practica

Función objetivo

```
def valor(S,COSTES):  
    VALOR = 0  
    for i in range(len(S)):  
        VALOR += COSTES[S[i]][i]  
    return VALOR  
  
valor((0, 1, 2, 3),COSTES)
```

```
# TAREAS  
# A  
# G  
# E  
# N  
# T  
# E  
# S  
  
COSTES=[ [11,12,18,40],  
          [14,15,13,22],  
          [11,17,19,23],  
          [17,14,20,28]]
```



Ramificación y Poda. Practica

Fuerza Bruta:

```
@calcular_tiempo
def fuerza_bruta(COSTES):
    #Representacion de la solucion será una tupla donde cada valor en la cordenada i-sima es la tarea asignado al agente i
    # Ejemplo (1,2,3,4) Tiene valor 11+15+19+28=73
    #
    #¿Cuántas posibilidades hay? n! -> Complejidad factorial(exponencial)
    #Con dimension 11 se va a 1 minuto de ejecucion

    mejor_valor = 10e10
    mejor_solucion = ()

    for s in list(itertools.permutations(range(len(COSTES)))):
        #print(s,valor(s,COSTES))
        valor_tmp = valor(s,COSTES)
        if valor_tmp < mejor_valor:
            mejor_valor = valor_tmp
            mejor_solucion = s

    print("La mejor solucion es :", mejor_solucion, " con valor:", mejor_valor )
```


Ramificación y Poda. Practica

Función para estimar un coste inferior para una solución parcial:

```
#Coste inferior para soluciones parciales
# (1,3,) Se asigna la tarea 1 al agente 0 y la tarea 3 al agente 1

def CI(S,COSTES):
    VALOR = 0
    #Valores establecidos
    for i in range(len(S)):
        VALOR += COSTES[i][S[i]]

    #Estimacion
    for i in range( len(COSTES) ):
        if i not in S:
            VALOR += min( [ COSTES[j][i] for j in range(len(S), len(COSTES)) ] )
    return VALOR
```

Ej. La tupla (2,0,)

Representa un nodo
intermedio con :
Agente 0 asignado a 2
Agente 1 asignado a 0

Ramificación y Poda. Practica

Función para ramificar:

```
#Genera tantos hijos como posibilidades haya para la siguiente elemento de la tupla
#(0,) -> (0,1), (0,2), (0,3)
def crear_hijos(NODO, N):
    HIJOS = []
    for i in range(N ):
        if i not in NODO:
            HIJOS.append({'s':NODO +(i,)    })
    return HIJOS
```

Ramificación y Poda. Practica

Proceso principal (I):

```
@calcular_tiempo
def ramificacion_y_poda(COSTES):
    #Construccion iterativa de soluciones(arbol). En cada etapa asignamos un agente(ramas).
    #Nodos del grafo { s:(1,2),CI:3,CS:5 }
    #print(COSTES)
    DIMENSION = len(COSTES)
    MEJOR_SOLUCION=tuple( i for i in range(len(COSTES)) )
    CotaSup = valor(MEJOR_SOLUCION,COSTES)
    #print("Cota Superior:", CotaSup)

    NODOS=[]
    NODOS.append({ 's':(), 'ci':CI((),COSTES) } )


    iteracion = 0
```



continua

Ramificación y Poda. Practica

Proceso principal (II):



```
while( len(NODOS) > 0):
    iteracion +=1

    nodo_prometedor = [ min(NODOS, key=lambda x:x['ci']) ][0]['s']
    #print("Nodo prometedor:", nodo_prometedor)

    #Ramificacion
    #Se generan los hijos
    HIJOS = [ {'s':x['s'], 'ci':CI(x['s'], COSTES)} for x in crear_hijos(nodo_prometedor, DIMENSION) ]

    #Revisamos la cota superior y nos quedamos con la mejor solucion si llegamos a una solucion final
    NODO_FINAL = [x for x in HIJOS if len(x['s']) == DIMENSION ]
    if len(NODO_FINAL) >0:
        #print("\n*****Soluciones:", [x for x in HIJOS if len(x['s']) == DIMENSION ] )
        if NODO_FINAL[0]['ci'] < CotaSup:
            CotaSup = NODO_FINAL[0]['ci']
            MEJOR_SOLUCION = NODO_FINAL

    #Poda
    HIJOS = [x for x in HIJOS if x['ci'] < CotaSup ]

    #Añadimos los hijos
    NODOS.extend(HIJOS)

    #Eliminamos el nodo ramificado
    NODOS = [ x for x in NODOS if x['s'] != nodo_prometedor ]

print("La solucion final es:" ,MEJOR_SOLUCION , " en " , iteracion , " iteraciones" , " para dimension: " ,DIMENSION )
```

Ramificación y Poda. Practica

Análisis para **mejorar nota**:

- ¿Que complejidad tiene el algoritmo por fuerza bruta?
- Generar matrices con valores aleatorios de mayores dimensiones (5,6,7,...) y ejecutar ambos algoritmos.
- ¿A partir de que dimensión el algoritmo por fuerza bruta deja de ser una opción?
- ¿Hay algún valor de la dimensión a partir de la cual el algoritmo de ramificación y poda deja de ser una opción válida?



10/10

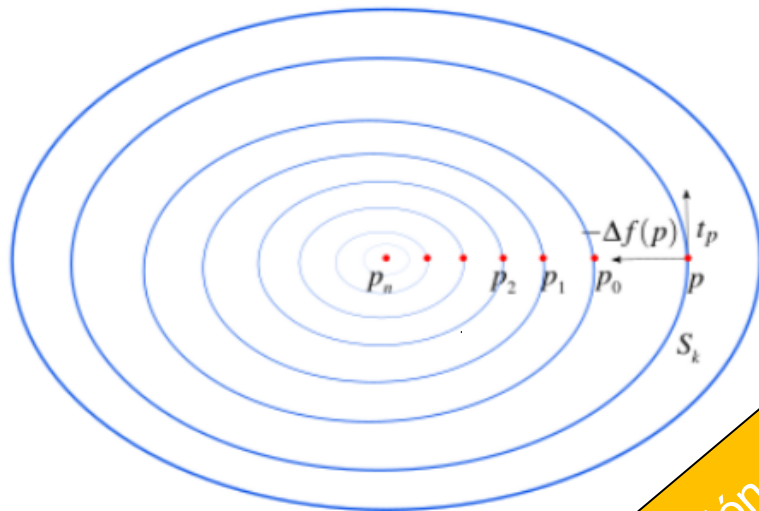
Descenso del Gradiente – AGD

- El procedimiento parte de un punto **p** como **solución aproximada** y da pasos en el sentido opuesto al gradiente (si minimizamos) de la función en dicho punto.

$$p_{t+1} = p_t - \alpha_t \nabla f(p_t)$$

Donde el parámetro α_t se selecciona para que p_{t+1} sea solución

$$f(p_t) \geq f(p_{t+1})$$

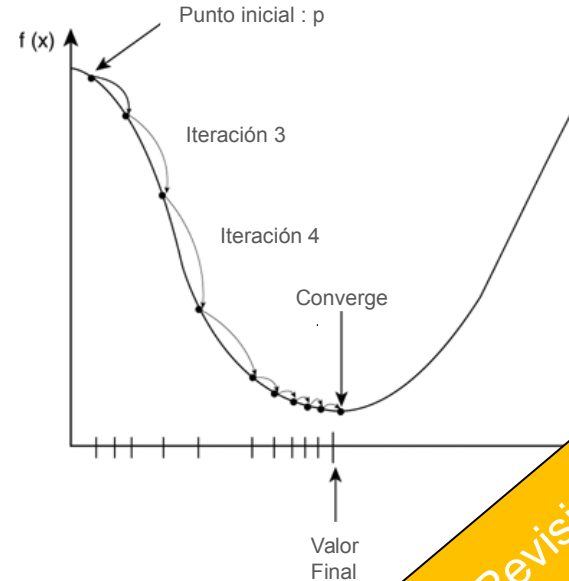


Revisión

Descenso del Gradiente – AGD

- En general se va reduciendo el valor de α_t dinámicamente a medida que nos aproximamos a la solución que podemos deducir por:
 - la magnitud del gradiente
 - cantidad de iteraciones que hemos realizado

$$p_{t+1} = p_t - \alpha_t \nabla f(p_t)$$



Revisión

Descenso del gradiente. Práctica

Preparar entorno

```
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
```


Descenso del gradiente. Práctica

Preparar entorno

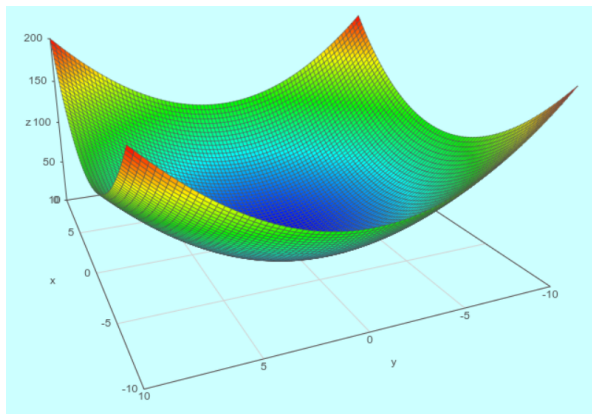
```
import math                #Funciones matematicas
import matplotlib.pyplot as plt #Generacion de gráficos (otra opcion seaborn)
import numpy as np         #Tratamiento matriz N-dimensionales y otras (fundamental!)
#import scipy as sc

import random
```

Descenso del gradiente. Práctica

La función a minimizar. Paraboloide

```
f = lambda X: X[0]**2+X[1]**2      #Funcion  
df = lambda X: [2*X[0] , 2*X[1]]  #Gradiente
```



Descenso del gradiente. Práctica

Prepara los datos para el gráfico

```
#Prepara los datos para dibujar mapa de niveles de Z
resolucion = 100
rango=2.5
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()
```

Descenso del gradiente. Práctica

Generamos un Punto aleatorio

```
#Generamos un punto aleatorio  
P=[random.uniform(-2,2 ),random.uniform(-2,2 ) ]  
plt.plot(P[0],P[1],"o",c="white")
```

Descenso del gradiente. Práctica

Iteramos el algoritmo

```
#Tasa de aprendizaje
TA=.1

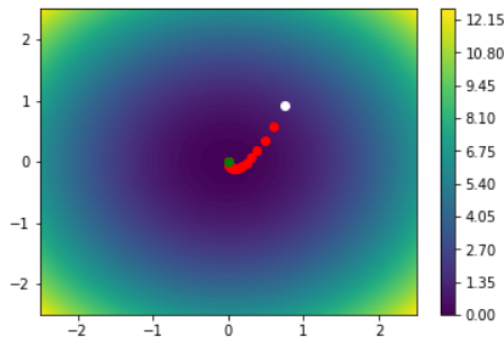
#Iteraciones
for _ in range(500):
    grad = df(P)
    #print(P.grad)
    P[0],P[1] = P[0] - TA*grad[0] , P[1] - TA*grad[1]
    plt.plot(P[0],P[1], "o",c="red")
```

Descenso del gradiente. Práctica

Pintamos el gráfico con las iteraciones

```
plt.plot(P[0],P[1],"o",c="green")  
plt.show()  
print("Solucion:" , P , f(P))
```

↳ Punto Inicial: [0.7569403161678125, 0.9223403659681089]

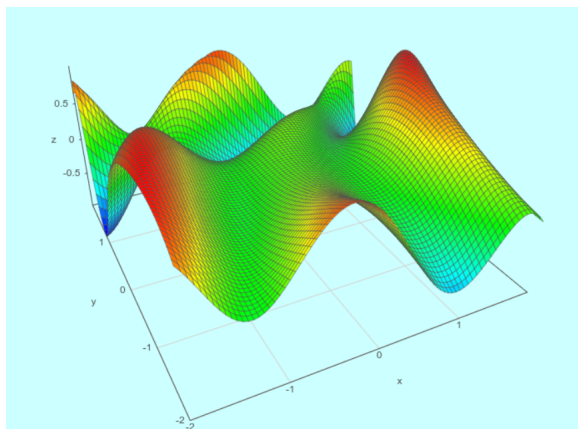


Solucion: [-0.004999999999999998, -0.005000000000000004] 5.000000000000002e-05

Descenso del gradiente. Práctica

Otra función a minimizar.

```
#Definimos la funcion  
#sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2*x + 1 - E^y)  
f = lambda X: np.sin(1/2 * X[0]**2 - 1/4 * X[1]**2 + 3) * np.cos(2 * X[0] + 1 - np.e**X[1])
```



$\sin(1/2 * x^2 - 1/4 * y^2 + 3) * \cos(2*x + 1 - E^y)$

<http://al-roomi.org/3DPlot/index.html>

Descenso del gradiente. Práctica

¿Y el gradiente? !!



```
#Definimos la funcion
#sin(1/2 * x^2 - 1/4 * y^2 + 3) * cos(2*x + 1 - E^y)
f = lambda X: np.sin(1/2 * X[0]**2 - 1/4 * X[1]**2 + 3) * np.cos(2 * X[0] + 1 - np.e**X[1])
```

#Aproximamos el valor del gradiente en un punto por su definición

```
def df(PUNTO):
    h = 0.01
    T = np.copy(PUNTO)
    grad = np.zeros(2)
    for it, th in enumerate(PUNTO):
        T[it] = T[it] + h
        grad[it] = (f(T) - f(PUNTO)) / h
    return grad
```

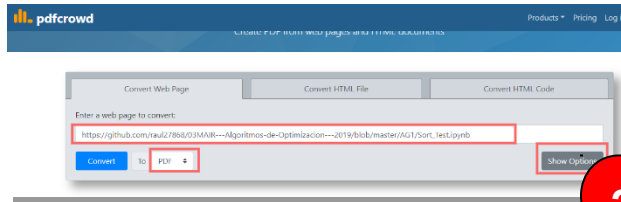
$$f'(a) = \lim_{h \rightarrow 0} \frac{f(a+h) - f(a)}{h}$$



Finalizar la actividad. Grabar, subir a GitHub, Generar pdf (I)

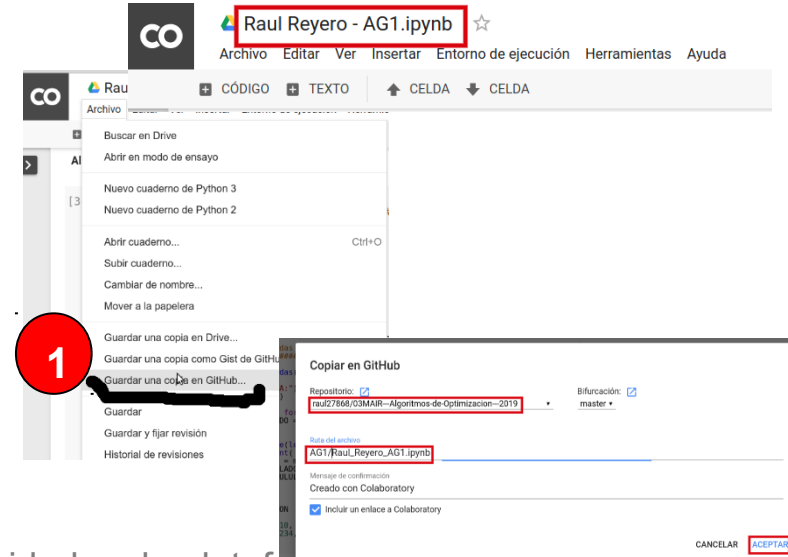
- Guardar en GitHub
Repositorio: 03MAIR ---Algoritmos de Optimizacion
Ruta de Archivo con **AG2**

- Generar pdf (con <https://pdfcrowd.com>)



- Descargar pdf y adjuntar el documento generado a la actividad en la plataforma
 - Adjuntar .pdf en la actividad
 - URL GitHub en el texto del mensaje de la actividad

3



Próxima clase. VC4

Introducción a las metaheurísticas

1. Búsqueda aleatoria
2. Búsqueda basada en trayectorias
3. Métodos basados en trayectorias. Búsqueda Tabú
4. Métodos basados en trayectorias. Recocido Simulado
5. Métodos constructivos. Multiarranque
6. Métodos constructivos. GRASP
7. Métodos constructivos. Colonia de hormigas

Ampliación de conocimientos y habilidades

■ Bibliografía

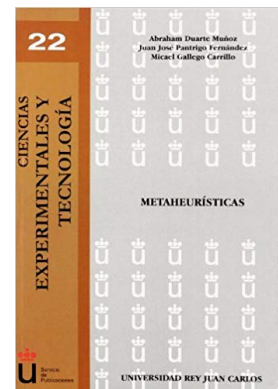
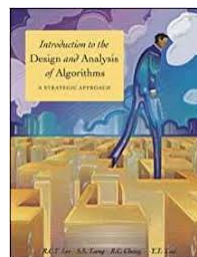
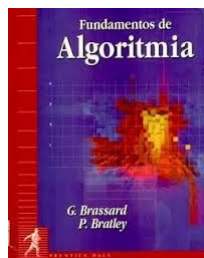
-Brassard, G., y Bratley, P. (1997). Fundamentos de algoritmia. *ISBN 13: 9788489660007*

-Guerequeta, R., y Vallecillo, A. (2000). Técnicas de diseño de algoritmos.(
<http://www.lcc.uma.es/~av/Libro/indice.html>)

-Lee, R. C. T., Tseng, S. S., Chang, R. C., y Tsai, Y. T. (2005). Introducción al diseño y análisis de algoritmos. *ISBN 13: 9789701061244*

-**Abraham Duarte,.. Metaheurísticas. ISBN 13: 9788498490169**

• Practicar



Gracias

raul.reyero@campusviu.es