Curso de Optimización (DEMAT)

Tarea 4

Leslie Janeth Quincosa Ramírez

Descripción:	Fechas
Fecha de publicación del documento:	**Febrero 24, 2022**
Fecha límite de entrega de la tarea:	**Marzo 6, 2022**

Indicaciones

Puede escribir el código de los algoritmos que se piden en una celda de este notebook o si lo prefiere, escribir las funciones en un archivo `.py` independiente e importar la funciones para usarlas en este notebook. Lo importante es que en el notebook aparezcan los resultados de la pruebas realizadas y que:

- **Si se requieren otros archivos para poder reproducir los resultados, para mandar la tarea cree un archivo ZIP en el que incluya el notebook y los archivos adicionales.**
- **Si todos los códigos para que se requieren para reproducir los resultados están en el notebook, no hace falta comprimirlo y puede anexar sólo el notebook en la tarea del Classroom.**
- **Exportar el notebook a un archivo PDF y anexarlo en la tarea del Classroom como un archivo independiente. No lo incluya dentro del ZIP , porque la idea que lo pueda accesar directamente para poner anotaciones y la calificación de cada ejercicio.**

En la descripción de los ejercicios se nombran algunas variables para el algoritmo, pero sólo es para facilitar la descripción. En la implementación pueden nombrar sus variables como gusten.

En los algoritmos se describen las entradas de las funciones. La intención es que tomen en cuenta lo que requiere el algoritmo y que tiene que haber parámetros que permitan controlar el comportamiento del algoritmo, evitando que dejen fijo un valor y que no se puede modificar para hacer diferentes pruebas. Si quieren dar esta información usando un tipo de dato que contenga todos los valores o usar variables por separado, etc., lo pueden hacer y no usen variables globales si no es necesario.

Lo mismo para los valores que devuelve una función. Pueden codificar como gusten la manera en que regresa los cálculos. El punto es que podamos tener acceso a los resultados, sin usar variables globales, y que la función no sólo imprima los valores que después no los podamos usar.

Ejercicio 1 (5 puntos)

Programar el método de descenso máximo con tamaño de paso exacto para minimizar funciones cuadráticas:

$$f(x) = \frac{1}{2} x^\top \mathbf{A} x - b^\top x,$$

donde $\mathbf{A} \in \mathrm{R}^{n \times n}$
y $x \in \mathrm{R}^n$
.

Dado el vector b
, la matriz $\mathbf{A}$
, un punto inicial x_0
, un número máximo de iteraciones N
, la tolerancia $\tau > 0$
. Fijar $k = 0$

y repetir los siguientes pasos:

1. **Calcular el gradiente en el punto** x_k
   ,
   $$g_k = \nabla f(x_k) = \mathbf{A}x_k - b$$
   .
2. **Si** $\|g_k\| < \tau$
   , entonces x_k
   es (casi) un punto estacionario. Hacer $res = 1$
   y terminar el ciclo.
3. **Elegir la dirección de descenso como** $p_k = -g_k$
   .
4. **Calcular el tamaño de paso** α_k
   que minimiza el valor de la función
   $$\phi_k(\alpha) = f(x_k + \alpha p_k)$$
   es decir, calcular
   $$\alpha_k = -\frac{g_k^\top p_k}{p_k^\top \mathbf{A} p_k}$$
5. **Calcular el siguiente punto de la secuencia como**
   $$x_{k+1} = x_k + \alpha_k p_k$$
6. **Si** $k + 1 \geq N$
   , hacer $res = 0$
   y terminar.
7. **Si no, hacer** $k = k + 1$
   y volver el paso 1.
8. **Devolver el punto** x_k
   , $f_k = \frac{1}{2} x_k^\top \mathbf{A} x_k - b^\top x_k$
   , g_k
   , k
   y res
   .

---

1. **Escriba una función que implementa el algoritmo anterior usando arreglos de Numpy.**
2. **Escriba una función para probar el funcionamiento del método de descenso máximo. Esta función debe recibir como parámetros el nombre de un archivo** `.npy` **que contiene las entradas de una matriz cuadrada** $\mathbf{A}$
   , el vector b
   , un punto inicial x_0
   , el número máximo de iteraciones N
   y la tolerancia τ
   .

- **Esta función debe crear la matriz** $\mathbf{A}$
  y el vector b
  leyendo los archivos de datos.
- **Obtener el número de filas** r
  de la matriz e imprimir este valor.
- **Compruebe que la matriz es simétrica y definida positiva calculando e imprimiendo el valor** $\|\mathbf{A} - \mathbf{A}^\top\|$
  y su eigenvalor más pequeño (use la función `numpy.linalg.eig()` **).**
- **Ejecutar la función del Inciso 1.**
- **Dependiendo del valor de la variable** res

Dependiendo del valor de la variable res
, imprima un mensaje que diga que el algoritmo convergió ($res = 1$
) o no ($res = 0$
).

- **Imprimir** k
  , f_k
  , la norma de g_k
  , y los primeros 3 y últimos 3 elementos del arreglo x_k
  .
- **Calcule directamente el minimizador resolviendo la ecuación** $Ax_* = b$
  e imprima el valor del error $\|x_k - x_*\|$
  .

1. **Pruebe la función del Inciso 2 usando** $N = 1000$
   , la tolerancia $\tau = \epsilon_m^{1/3}$
   , donde ϵ_m
   es el épsilon de la máquina, y los arreglos que se incluyen en el archivo datosTarea04.zip, de la siguiente manera:

Matriz	Vector	Punto para iniciar la secuencia
A1.npy	b1.npy	$x_0 = (0, -5)$
A1.npy	b1.npy	$x_0 = (7045, 7095)$
A2.npy	b2.npy	$x_0 = (0, 0, \ldots, 0) \in \mathrm{R}^{500}$
A2.npy	b2.npy	$x_0 = (10000, 10000, \ldots, 10000) \in \mathrm{R}^{500}$

Solución:

In [37]:

```
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importarlas de un archivo .py

import numpy as np
from numpy import linalg as LA
from numpy.linalg import eigvals
import sys

def DescMAxPaso (b, A, N, x0, tol):
    xk = x0
    for k in range(N):
      gk = A@xk - b
      norm = LA.norm(gk)
      if norm < tol:
          res = 1
          break
      else:
          pk = -gk
          ak = - (gk.T@pk)/(pk.T@A@pk)
          xk = xk + ak*pk
          if k+1 >= N:
              res = 0
              break
    fk = (1/2)*xk.T@A@xk-b.T@xk
    return xk, fk, gk, k, res
```

In [38]:

```
# Lectura de datos y pruebas realizadas

def Probar(archivo_A, archivo_b, x0, N, tol):
    A = np.load(archivo_A)
    b = np.load(archivo_b)
    r = A.shape[0]
```

```
    print('Número de filas:',r)
    eigval = np.linalg.eigvals(A)
    eigmin = eigval.min()
    a = LA.norm(A - A.T)
    print('Eigenvalores:', eigval)
    print('||A-AT||:', a)
    xk, fk, gk, k, res = DescMAxPaso (b, A, N, x0, tol)
    if res == 1:
        print('¡Convergió! :)')
    else:
        print('¡No convergió! :(')
    print('Valor k:', k)
    print('fk:', fk)
    print('||gk||:', LA.norm(gk))
    print('Vector xk:', xk[:3], '...', xk[-3:])
    x = np.linalg.solve(A, b)
    print('||xk-x||:', LA.norm(xk-x))
```

In [39]:

```
#Pruebas

Probar('A1.npy', 'b1.npy', np.array([0,-5]), 1000, tol)
```

```
Número de filas: 2
Eigenvalores: [1.  0.1]
||A-AT||: 0.0
¡Convergió! :)
Valor k: 69
fk: -62.749999999933024
||gk||: 4.767340294300877e-06
Vector xk: [-24.49997287  25.49997743] ... [-24.49997287  25.49997743]
||xk-x||: 3.528998540703229e-05
```

In [40]:

```
# Prueba: Descenso máximo con paso exacto

from numpy.core.fromnumeric import partition
#Pruebas
eps = sys.float_info.epsilon
print('epsilon de la maquina', eps)
tol = eps**(1/3)

x0s = [np.array([0, -5]) , np.array([7045,7095])]


print("Descenso máximo con paso exacto. \n")

for x0 in x0s:
      Probar('A1.npy', 'b1.npy', x0, 1000, tol)
      print('\n')
```

```
epsilon de la maquina 2.220446049250313e-16
Descenso máximo con paso exacto.

Número de filas: 2
Eigenvalores: [1.  0.1]
||A-AT||: 0.0
¡Convergió! :)
Valor k: 69
fk: -62.749999999933024
||gk||: 4.767340294300877e-06
Vector xk: [-24.49997287  25.49997743] ... [-24.49997287  25.49997743]
||xk-x||: 3.528998540703229e-05


Número de filas: 2
Eigenvalores: [1.  0.1]
||A-AT||: 0.0
¡Convergió! :)
Valor k: 1
```

```
fk: -62.75
||gk||: 6.425429159208664e-13
Vector xk: [-24.5  25.5] ... [-24.5  25.5]
||xk-x||: 9.131096203816022e-13
```

Nota: el vector x_k
solo tiene dos entradas.

In [42]:

```
#Prueba: A2.npy b2.npy  x0=(0,0,...,0)∈R500

print("Descenso máximo con paso exacto. \n")
x0s = [np.zeros(500) , 1000*np.ones(500)]

for x0 in x0s:
      Probar('A2.npy', 'b2.npy', x0, 1000, tol)
      print('\n')
```

```
Descenso máximo con paso exacto.

Número de filas: 500
Eigenvalores: [5.         4.99018036 4.98036072 4.97054108 4.96072144 4.9509018
 4.94108216 4.93126253 4.92144289 4.91162325 4.90180361 4.89198397
 4.88216433 4.87234469 4.86252505 4.85270541 4.84288577 4.83306613
 4.82324649 4.81342685 4.80360721 4.79378758 4.78396794 4.76432866
 4.7741483  0.1        0.10981964 0.11963928 0.12945892 0.13927856
 0.1490982  0.15891784 0.16873747 0.17855711 0.18837675 0.19819639
 0.20801603 0.21783567 0.22765531 0.23747495 0.24729459 0.25711423
 0.26693387 0.27675351 0.28657315 0.33567134 0.29639279 0.3258517
 0.31603206 0.30621242 4.75450902 4.74468938 4.73486974 4.7250501
 4.68577154 4.71523046 4.70541082 4.69559118 0.34549098 0.35531062
 0.40440882 0.36513026 0.3749499  0.38476954 0.39458918 0.41422846
 0.4240481  0.43386774 0.45350701 0.44368737 4.62685371 4.63667335
 4.6759519  4.64649299 4.65631263 4.66613226 4.58757515 4.59739479
 4.60721443 4.61703407 0.50260521 0.49278557 0.46332665 0.48296593
 0.47314629 0.51242485 0.54188377 0.52224449 0.53206413 4.53847695
 4.54829659 4.55811623 4.57775551 4.56793587 0.58116232 0.55170341
 0.56152305 0.57134269 4.4991984  4.52865731 4.50901804 4.51883768
 0.62044088 0.61062124 0.59098196 0.6008016  4.45991984 4.48937876
 4.46973948 4.47955912 4.4501002  4.41082164 4.42064128 4.44028056
 4.43046092 0.63026052 0.64008016 0.6498998  0.65971944 0.66953908
 0.67935872 0.68917836 0.698998   0.70881764 0.71863727 0.72845691
 4.401002   4.39118236 4.38136273 4.37154309 4.36172345 4.33226453
 4.34208417 4.35190381 4.32244489 4.31262525 4.30280561 4.29298597
 4.28316633 4.27334669 4.26352705 4.25370741 4.24388778 4.23406814
 0.73827655 0.74809619 0.76773547 0.75791583 0.77755511 0.78737475
 0.79719439 0.80701403 0.84629259 0.83647295 0.82665331 0.81683367
 4.2242485  4.21442886 4.20460922 4.19478958 4.18496994 4.08677355
 4.09659319 4.1751503  4.16533066 4.10641283 4.15551102 4.11623246
 4.1260521  4.14569138 4.13587174 0.85611222 0.86593186 0.8757515
 0.88557114 0.98376754 0.9739479  0.96412826 0.95430862 0.94448898
 0.93466934 0.9248497  0.91503006 0.90521042 0.89539078 4.07695391
 4.06713427 0.99358717 1.00340681 1.01322645 1.02304609 1.03286573
 1.04268537 1.05250501 1.06232465 4.05731463 4.01803607 4.04749499
 4.03767535 4.02785571 4.00821643 3.99839679 3.98857715 3.97875752
 3.96893788 3.95911824 3.9492986  3.93947896 3.92965932 1.07214429
 1.08196393 1.09178357 1.11142285 1.10160321 1.12124248 1.13106212
 1.14088176 1.1507014  1.16052104 1.17034068 1.18016032 1.18997996
 1.1997996  1.28817635 1.20961924 1.27835671 1.26853707 1.25871743
 1.2488978  1.21943888 1.22925852 1.23907816 3.91983968 3.91002004
 3.9002004  3.89038076 3.88056112 3.87074148 3.78236473 3.79218437
 3.86092184 3.8511022  3.84128257 3.80200401 3.81182365 3.83146293
 3.82164329 3.77254509 3.76272545 1.29799599 1.31763527 1.30781563
 3.69398798 3.71362725 3.72344689 3.73326653 3.74308617 3.75290581
 1.36673347 1.35691383 1.33727455 1.34709419 1.32745491 3.70380762
 3.68416834 3.6743487  3.66452906 3.65470942 3.64488978 3.63507014
 3.61543086 3.6252505  1.38637275 1.37655311 1.39619238 1.41583166
 1.40601202 1.4256513  1.43547094 1.44529058 1.45511022 1.46492986
```

```
 1.4747495  1.48456914 1.49438878 1.50420842 1.58276553 1.51402806
 1.57294589 1.5238477  1.53366733 1.56312625 1.55330661 1.54348697
 3.60561122 3.59579158 3.46813627 3.58597194 3.5761523  3.56633267
 3.47795591 3.55651303 3.54669339 3.48777555 3.49759519 3.50741483
 3.53687375 3.51723447 3.52705411 1.60240481 1.59258517 3.45831663
 3.44849699 3.43867735 3.3993988  3.40921844 3.42885772 1.61222445
 1.62204409 1.63186373 1.65150301 1.66132265 1.64168337 3.38957916
 3.37975952 3.36993988 3.3503006  3.36012024 3.34048096 3.33066132
 3.32084168 3.3012024  3.31102204 3.29138277 1.68096192 1.69078156
 1.67114228 1.7006012  1.71042084 1.72024048 1.73987976 1.7496994
 1.73006012 1.75951904 3.41903808 3.28156313 3.26192385 3.25210421
 3.24228457 3.23246493 3.14408818 3.22264529 3.15390782 3.21282565
 3.20300601 3.19318637 3.16372745 3.17354709 3.18336673 1.77915832
 1.76933868 1.78897796 1.7987976  1.80861723 1.81843687 1.82825651
 1.92645291 1.83807615 1.91663327 1.84789579 1.90681363 1.85771543
 1.89699399 1.88717435 1.86753507 1.87735471 3.1244489  3.13426854
 1.94609218 1.93627255 3.08517034 3.09498998 3.10480962 3.11462926
 1.95591182 1.99519038 1.98537074 1.9755511  3.0753507  3.05571142
 3.02625251 3.03607214 3.06553106 2.00501002 2.01482966 2.0246493
 2.03446894 2.04428858 1.96573146 3.27174349 3.04589178 3.01643287
 3.00661323 2.98697395 2.99679359 2.96733467 2.95751503 2.94769539
 2.93787575 2.91823647 2.90841683 2.05410822 2.15230461 2.14248497
 2.13266533 2.12284569 2.08356713 2.09338677 2.06392786 2.07374749
 2.88877756 2.87895792 2.86913828 2.85931864 2.83967936 2.78076152
 2.79058116 2.10320641 2.40761523 2.17194389 2.18176353 2.16212425
 2.20140281 2.24068136 2.250501   2.27014028 2.28977956 2.2995992
 2.30941884 2.31923848 2.32905812 2.23086172 2.22104208 2.82985972
 2.81022044 2.77094188 2.75130261 2.74148297 2.34869739 2.36833667
 2.39779559 2.41743487 2.43707415 2.45671343 2.44689379 2.71202405
 2.70220441 2.67274549 2.68256513 2.49599198 2.50581162 2.51563126
 2.53527054 2.54509018 2.55490982 2.64328657 2.63346693 2.60400802
 2.61382766 2.59418838 2.11302605 2.19158317 2.21122244 2.849499
 2.82004008 2.8004008  2.37815631 2.65310621 2.62364729 2.72184369
 2.26032064 2.33887776 2.47635271 2.76112224 2.5745491  2.69238477
 2.66292585 2.73166333 2.42725451 2.56472946 2.89859719 2.97715431
 2.92805611 2.27995992 2.38797595 2.35851703 2.5254509  2.46653307
 2.48617234 2.58436874]
||A-AT||: 1.5063918215255853e-14
¡Convergió! :)
Valor k: 332
fk: -5239.541412076966
||gk||: 5.996601822363623e-06
Vector xk: [-11.61784712   5.44982336   0.96506345] ... [ -1.8852386  -10.34321936  -4.15
697837]
||xk-x||: 3.929751161353029e-05


Número de filas: 500
Eigenvalores: [5.         4.99018036 4.98036072 4.97054108 4.96072144 4.9509018
 4.94108216 4.93126253 4.92144289 4.91162325 4.90180361 4.89198397
 4.88216433 4.87234469 4.86252505 4.85270541 4.84288577 4.83306613
 4.82324649 4.81342685 4.80360721 4.79378758 4.78396794 4.76432866
 4.7741483  0.1        0.10981964 0.11963928 0.12945892 0.13927856
 0.1490982  0.15891784 0.16873747 0.17855711 0.18837675 0.19819639
 0.20801603 0.21783567 0.22765531 0.23747495 0.24729459 0.25711423
 0.26693387 0.27675351 0.28657315 0.33567134 0.29639279 0.3258517
 0.31603206 0.30621242 4.75450902 4.74468938 4.73486974 4.7250501
 4.68577154 4.71523046 4.70541082 4.69559118 0.34549098 0.35531062
 0.40440882 0.36513026 0.3749499  0.38476954 0.39458918 0.41422846
 0.4240481  0.43386774 0.45350701 0.44368737 4.62685371 4.63667335
 4.6759519  4.64649299 4.65631263 4.66613226 4.58757515 4.59739479
 4.60721443 4.61703407 0.50260521 0.49278557 0.46332665 0.48296593
 0.47314629 0.51242485 0.54188377 0.52224449 0.53206413 4.53847695
 4.54829659 4.55811623 4.57775551 4.56793587 0.58116232 0.55170341
 0.56152305 0.57134269 4.4991984  4.52865731 4.50901804 4.51883768
 0.62044088 0.61062124 0.59098196 0.6008016  4.45991984 4.48937876
 4.46973948 4.47955912 4.4501002  4.41082164 4.42064128 4.44028056
 4.43046092 0.63026052 0.64008016 0.6498998  0.65971944 0.66953908
 0.67935872 0.68917836 0.698998   0.70881764 0.71863727 0.72845691
 4.401002   4.39118236 4.38136273 4.37154309 4.36172345 4.33226453
 4.34208417 4.35190381 4.32244489 4.31262525 4.30280561 4.29298597
 4.28316633 4.27334669 4.26352705 4.25370741 4.24388778 4.23406814
```

```
 0.73827655 0.74809619 0.76773547 0.75791583 0.77755511 0.78737475
 0.79719439 0.80701403 0.84629259 0.83647295 0.82665331 0.81683367
 4.2242485  4.21442886 4.20460922 4.19478958 4.18496994 4.08677355
 4.09659319 4.1751503  4.16533066 4.10641283 4.15551102 4.11623246
 4.1260521  4.14569138 4.13587174 0.85611222 0.86593186 0.8757515
 0.88557114 0.98376754 0.9739479  0.96412826 0.95430862 0.94448898
 0.93466934 0.9248497  0.91503006 0.90521042 0.89539078 4.07695391
 4.06713427 0.99358717 1.00340681 1.01322645 1.02304609 1.03286573
 1.04268537 1.05250501 1.06232465 4.05731463 4.01803607 4.04749499
 4.03767535 4.02785571 4.00821643 3.99839679 3.98857715 3.97875752
 3.96893788 3.95911824 3.9492986  3.93947896 3.92965932 1.07214429
 1.08196393 1.09178357 1.11142285 1.10160321 1.12124248 1.13106212
 1.14088176 1.1507014  1.16052104 1.17034068 1.18016032 1.18997996
 1.1997996  1.28817635 1.20961924 1.27835671 1.26853707 1.25871743
 1.2488978  1.21943888 1.22925852 1.23907816 3.91983968 3.91002004
 3.9002004  3.89038076 3.88056112 3.87074148 3.78236473 3.79218437
 3.86092184 3.8511022  3.84128257 3.80200401 3.81182365 3.83146293
 3.82164329 3.77254509 3.76272545 1.29799599 1.31763527 1.30781563
 3.69398798 3.71362725 3.72344689 3.73326653 3.74308617 3.75290581
 1.36673347 1.35691383 1.33727455 1.34709419 1.32745491 3.70380762
 3.68416834 3.6743487  3.66452906 3.65470942 3.64488978 3.63507014
 3.61543086 3.6252505  1.38637275 1.37655311 1.39619238 1.41583166
 1.40601202 1.4256513  1.43547094 1.44529058 1.45511022 1.46492986
 1.4747495  1.48456914 1.49438878 1.50420842 1.58276553 1.51402806
 1.57294589 1.5238477  1.53366733 1.56312625 1.55330661 1.54348697
 3.60561122 3.59579158 3.46813627 3.58597194 3.5761523  3.56633267
 3.47795591 3.55651303 3.54669339 3.48777555 3.49759519 3.50741483
 3.53687375 3.51723447 3.52705411 1.60240481 1.59258517 3.45831663
 3.44849699 3.43867735 3.3993988  3.40921844 3.42885772 1.61222445
 1.62204409 1.63186373 1.65150301 1.66132265 1.64168337 3.38957916
 3.37975952 3.36993988 3.3503006  3.36012024 3.34048096 3.33066132
 3.32084168 3.3012024  3.31102204 3.29138277 1.68096192 1.69078156
 1.67114228 1.7006012  1.71042084 1.72024048 1.73987976 1.7496994
 1.73006012 1.75951904 3.41903808 3.28156313 3.26192385 3.25210421
 3.24228457 3.23246493 3.14408818 3.22264529 3.15390782 3.21282565
 3.20300601 3.19318637 3.16372745 3.17354709 3.18336673 1.77915832
 1.76933868 1.78897796 1.7987976  1.80861723 1.81843687 1.82825651
 1.92645291 1.83807615 1.91663327 1.84789579 1.90681363 1.85771543
 1.89699399 1.88717435 1.86753507 1.87735471 3.1244489  3.13426854
 1.94609218 1.93627255 3.08517034 3.09498998 3.10480962 3.11462926
 1.95591182 1.99519038 1.98537074 1.9755511  3.0753507  3.05571142
 3.02625251 3.03607214 3.06553106 2.00501002 2.01482966 2.0246493
 2.03446894 2.04428858 1.96573146 3.27174349 3.04589178 3.01643287
 3.00661323 2.98697395 2.99679359 2.96733467 2.95751503 2.94769539
 2.93787575 2.91823647 2.90841683 2.05410822 2.15230461 2.14248497
 2.13266533 2.12284569 2.08356713 2.09338677 2.06392786 2.07374749
 2.88877756 2.87895792 2.86913828 2.85931864 2.83967936 2.78076152
 2.79058116 2.10320641 2.40761523 2.17194389 2.18176353 2.16212425
 2.20140281 2.24068136 2.250501   2.27014028 2.28977956 2.2995992
 2.30941884 2.31923848 2.32905812 2.23086172 2.22104208 2.82985972
 2.81022044 2.77094188 2.75130261 2.74148297 2.34869739 2.36833667
 2.39779559 2.41743487 2.43707415 2.45671343 2.44689379 2.71202405
 2.70220441 2.67274549 2.68256513 2.49599198 2.50581162 2.51563126
 2.53527054 2.54509018 2.55490982 2.64328657 2.63346693 2.60400802
 2.61382766 2.59418838 2.11302605 2.19158317 2.21122244 2.849499
 2.82004008 2.8004008  2.37815631 2.65310621 2.62364729 2.72184369
 2.26032064 2.33887776 2.47635271 2.76112224 2.5745491  2.69238477
 2.66292585 2.73166333 2.42725451 2.56472946 2.89859719 2.97715431
 2.92805611 2.27995992 2.38797595 2.35851703 2.5254509  2.46653307
 2.48617234 2.58436874]
||A-AT||: 1.5063918215255853e-14
¡Convergió! :)
Valor k: 401
fk: -5239.541412076962
||gk||: 5.941558582041861e-06
Vector xk: [-11.6178472    5.44982338   0.96506342] ... [ -1.88523856 -10.3432194   -4.15
697836]
||xk-x||: 4.014900961096634e-05
```

Notemos que en todas las pruebas los vectores convergieron. El numero de iteraciones para llegar al mínimo depende del punto inicial. En el primer ejercicio tuvimos que en el primer caso convergió en 69 iteraciones y con otro punto inicial fue con una sola iteración.

Ejercicio 2 (5 puntos)

Programar el método de descenso máximo con tamaño de paso seleccionado por la estrategia de backtracking:

Algoritmo de descenso máximo con backtracking:

Dada una función $f: \mathrm{R}^n \to \mathrm{R}$
, su gradiente $g: \mathrm{R}^n \to \mathrm{R}^n$
, un punto inicial x_0
, un número máximo de iteraciones N
, una tolerancia $\tau > 0$
. Fijar $k = 0$
y repetir los siguientes pasos:

1. **Calcular el gradiente en el punto** x_k
   :

$$g_k = \nabla f(x_k) = g(x_k)$$

2. **Si** $\|g_k\| < \tau$
   , x_k
   es un aproximadamente un punto estacionario, por lo que hay que hacer $res = 1$
   y terminar el ciclo.
3. **Eligir la dirección de descenso como** $p_k = -g_k$
   .
4. **Calcular el tamaño de paso** α_k
   mediante la estrategia de backtraking, usando el algoritmo que describe más adelante.
5. **Calcular el siguiente punto de la secuencia como**

$$x_{k+1} = x_k + \alpha_k p_k$$

6. **Si** $k + 1 > N$
   , hacer $res = 0$
   y terminar.
7. **Si no, hacer** $k = k + 1$
   y volver el paso 1.
8. **Devolver el punto** x_k
   , $f_k = f(x_k)$
   , g_k
   , k
   y res
   .

Algoritmo de backtracking

Backtracking(f
, f_k
, g_k
, x_k
, p_k
, α_{ini}
, ρ
, c
)

El algoritmo recibe la función f

El algoritmo recibe la función f
, el punto x_k
, $f_k = f(x_k)$
, la dirección de descenso p_k
, un valor inicial α_{ini}
, $\rho \in (0,1)$
, $c \in (0,1)$
.

Fijar $\alpha = \alpha_{ini}$
y repetir los siguientes pasos:

1. **Si se cumple la condición**

$$f(x_k + \alpha p_k) \leq f_k + c\alpha g_k^\top p_k,$$

   terminar el ciclo devolviendo α
   .
2. **Hacer $\alpha = \rho\alpha$**
   y regresar al paso anterior.

---

1. **Escriba una función que implementa el algoritmo de backtracking.**
2. **Escriba la función que implementa el algoritmo de máximo descenso con búsqueda inexacta, usando backtraking. Tiene que recibir como paramétros todos los elementos que se listaron para ambos algoritmos.**
3. **Escriba una función para probar el funcionamiento del método de descenso máximo. Esta función debe recibir la función f**
   , la función g
   que devuelve su gradiente, el punto inicial x_0
   , el número
   máximo de iteraciones N
   , la tolerancia $\tau > 0$
   y el factor ρ
   del algoritmo de backtracking.

- **Fijar los parámetros $\alpha_{ini} = 2$**
  y $c = 0.0001$
  del algoritmo de backtracking.
- **Ejecutar la función del Inciso 2.**
- **Dependiendo del valor de la variable res**
  , imprima un mensaje que diga que el algoritmo convergió ($res = 1$
) o no ($res = 0$
)
- **Imprimir k**
  , x_k
  , f_k
  y la norma de g_k
  .

1. **Pruebe la función del Inciso 3 usando $N = 10000$**
   , $\rho = 0.8$
   , la tolerancia $\tau = \epsilon_m^{1/3}$
   , donde ϵ_m
   es el épsilon de la máquina. Aplique esta función a:

- **La función de Rosenbrock, descrita en la Tarea 3, usando como punto inicial $x_0 = (-1.2, 1)$**
  y $x_0 = (-12, 10)$
  . Como referencia, el minimizador de la función es $x_* = (1,1)$
  .

1. **Repita el inciso anterior con** $\rho = 0.5$
   .

Solución:

In [43]:

```
# En esta celda puede poner el código de las funciones
# o poner la instrucción para importarlas de un archivo .py

def Backtraking(f, fk, gk, xk, pk, a0, rho, c):
    a = a0
    while f(xk + a*pk) > fk +c*a*gk.T@pk:
        a = rho*a
    return a


def DescMAxBack(f, g, x0, N, tol, a0, rho, c):
    xk = x0
    for k in range(N):
      gk = g(xk)
      norm = LA.norm(gk)
      if norm < tol:
          res = 1
          break
      else:
          pk = -gk
          fk = f(xk)
          ak = Backtraking(f, fk, gk, xk, pk, a0, rho, c)
          xk = xk + ak*pk
          if k+1 >= N:
              res = 0
              break
    return xk, fk, gk, k, res
```

In [44]:

```
# Lectura de datos y pruebas realizadas

def Probar2(f, g, x0, N, tol, rho):
    a0 = 2
    c = 0.0001
    print('N:', N, 'x0:', x0, 'rho:', rho)
    xk, fk, gk, k, res = DescMAxBack (f, g, x0, N, tol, a0, rho, c)
    if res == 1:
        print('¡Convergió! :)')
    else:
        print('¡No convergió! :(')
    print('Valor k:', k)
    print('xk:', xk)
    print('fk:', fk)
    print('||gk||:', LA.norm(gk))

def function(x):
    xT = x.T
    return 100 * (xT[1] - xT[0]**2)**2 + (1-xT[0])**2

def gradient(x):
    xT = x.T
    return np.array([-400*xT[0]*(xT[1]- xT[0]**2) -2*(1-xT[0]), 200*(xT[1]-xT[0]**2)])
```

In [45]:

```
from numpy.core.fromnumeric import partition
#Pruebas
eps = sys.float_info.epsilon
tol = eps**(1/3)

x0s = [np.array([-1.2,1]) , np.array([-12,10])]
```

```
rhos = [0.8, 0.5]

Ns = [10000, 50000]

print("Descenso máximo con backtracking. \n")

for x0 in x0s:
    for rho in rhos:
        for N in Ns:
            Probar2(function, gradient, x0, N, tol, rho)
            print('\n')
```

```
Descenso máximo con backtracking.

N: 10000 x0: [-1.2  1. ] rho: 0.8
¡No convergió! :(
Valor k: 9999
xk: [1.00079208 1.00158391]
fk: 6.284994359483611e-07
||gk||: 0.0019653292817284995


N: 50000 x0: [-1.2  1. ] rho: 0.8
¡Convergió! :)
Valor k: 17005
xk: [1.00000294 1.0000059 ]
fk: 8.709467898400525e-12
||gk||: 5.999000737744113e-06


N: 10000 x0: [-1.2  1. ] rho: 0.5
¡No convergió! :(
Valor k: 9999
xk: [0.99998357 0.99996704]
fk: 2.7141545777021076e-10
||gk||: 2.377226272021971e-05


N: 50000 x0: [-1.2  1. ] rho: 0.5
¡Convergió! :)
Valor k: 11525
xk: [0.99999522 0.99999044]
fk: 2.286837854150557e-11
||gk||: 6.004300577326651e-06


N: 10000 x0: [-12  10] rho: 0.8
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: RuntimeWarning: overflow encountered in long_scalars
```

```
¡No convergió! :(
Valor k: 9999
xk: [ 3.92075554 15.37404809]
fk: 8.531123402708898
||gk||: 2.823173926047487


N: 50000 x0: [-12  10] rho: 0.8
¡No convergió! :(
Valor k: 49999
xk: [3.13859163 9.85233986]
fk: 4.573871935931183
||gk||: 2.0410061941286526


N: 10000 x0: [-12  10] rho: 0.5
¡No convergió! :(
Valor k: 9999
xk: [2.85989511 8.18266786]
fk: 3.460691855385309
||gk||: 1.121519863216604
```

```
||gk||: 1.1215190632100e-
```

```
N: 50000 x0: [-12  10] rho: 0.5
¡Convergió! :)
Valor k: 42287
xk: [1.00000469 1.00000939]
fk: 2.206627769944216e-11
||gk||: 6.021268237021556e-06
```

Notemos que para $N = 10000$
, el método no converge para ningún punto inicial.

Para un N
más grande sí converge. Vemos que este método es más inexacto que el de la tarea 3. Es muy útil saber el punto donde converge para que converja rapidamente y elegir esto para nuestra conveniencia. Sin embargo, en la práctica este método puede ser más costoso computacionalmente.