

Decision Models - Assignment B

Fabrizio Cominetti

Si consideri il seguente problema di massimizzazione:

$$\max f(x) = x^3 + 2x - 2x^2 - 0.25x^4$$

Si applichino il metodo di bisezione, delle tangenti (o di Newton) e delle secanti per la risoluzione del problema. Si effettui un'iterazione a mano, dopodiché si realizzino delle procedure in Python o in R che implementino gli algoritmi. La scelta del punto iniziale, della tolleranza, del numero massimo di iterazioni e delle condizioni di terminazione è libera. Idem per la seguente funzione obiettivo utilizzando, stavolta, il metodo del gradiente:

$$\max f(x) = 2x_1x_2 + x_2 - x_1^2 - 2x_2^2$$

Dire, infine, come si possa risolvere lo stesso problema tramite metaeuristiche.

$$\max f(x) = x^3 + 2x - 2x^2 - 0.25x^4$$

```
In [ ]: import math
import numpy as np

import matplotlib.pyplot as plt

from sympy import symbols
```

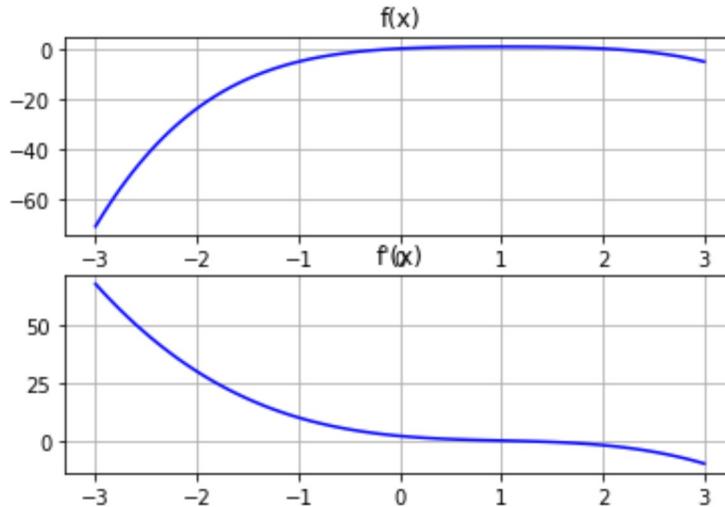
```
In [ ]: x = np.linspace(-3,3,100)
y = x**3 + 2*x - 2*x**2 - 0.25*x**4
y1 = 3*(x**2) + 2 - 4*x - x**3

fig = plt.figure()

ax = fig.add_subplot(2, 1, 1)
plt.title("f(x)")
plt.plot(x, y, 'b')
plt.grid()

ax = fig.add_subplot(2, 1, 2)
plt.title("f'(x)")
plt.plot(x, y1, 'b')
plt.grid()

plt.savefig("funzione.png")
plt.show()
```



```
In [ ]: def f(x):
    y = x**3 + 2*x - 2*(x**2) - 0.25*(x**4)
    return y
```

```
In [ ]: def f1(x):
    y = 3*(x**2) + 2 - 4*x - x**3
    return y
```

```
In [ ]: def f2(x):
    y = 6*x - 4 - 3*(x**2)
    return y
```

dal grafico assumiamo i seguenti valori:

- $x_l = -0.5$
- $x_u = 1.5$

```
In [ ]: x_init = 1.0
a = -0.5
b = 1.5
N = int(10000) # max iter
tolerance = 1e-7
```

```
In [ ]: from scipy.optimize import fsolve

print("***** SCIPY fsolve() *****")
x_star = fsolve(f1, x_init)
x_star = x_star[0]
print("x_star =", x_star)
print("f(x_star) =", f(x_star))
print("f'(x_star) =", f1(x_star))
print()
```

```
***** SCIPY fsolve() *****
x_star = 1.0
f(x_star) = 0.75
f'(x_star) = 0.0
```

Metodo di Bisezione

$$\begin{aligned} \text{Max } f(x) &= x^3 + 2x - 2x^2 - 0,25x^4 \\ f'(x) &= 3x^2 + 2 - 4x - x^3 \\ x_1 &= -0,5 \\ x_u &= 1,5 \\ f(x_1) &= 3(-0,5)^2 + 2 - 4(-0,5) - (-0,5)^3 = \\ &= 4,875 \\ f(x_u) &= 3(1,5)^2 + 2 - 4(1,5) - (1,5)^3 = \\ &= -0,625 \\ f(x_1) * f(x_u) &= -3,046 < 0 \end{aligned}$$

ITERAZIONE 1

$$\begin{aligned} x_m &= \frac{x_1 + x_u}{2} = \frac{-0,5 + 1,5}{2} = \frac{1}{2} = 0,5 \\ f(x_m) &= 3(0,5)^2 + 2 - 4(0,5) - (0,5)^3 = \\ &= 0,625 \end{aligned}$$

$$f(x_1) * f(x_m) = 3,046 > 0$$

Quindi riduce tra x_m e x_u

$$x_1 = 0,5$$

$$x_u = 1,5$$

...

```
In [ ]: def bisezione(func, a, b, N, tolerance):
    if np.sign(func(a)) == np.sign(func(b)):
        raise Exception("sign(f(a)) = sign(f(b))")
    for i in range(N):
        m = (a + b)/2
        auxString = "Iteration" + ' ' + '{:3d}'.format(i) + \
                    ", x = " + '{:.7f}'.format(m) + \
                    ", f(x) = " + '{:.7f}'.format(func(m))
        print(auxString)
        if func(m) == 0 or (b - a)/2 < tolerance:
            return m
        elif np.sign(func(a)) == np.sign(func(m)):
            a = m
        elif np.sign(func(b)) == np.sign(func(m)):
            b = m
    print("Raggiunto limite di iterazioni")
    return m
```

```
In [ ]: print("***** BISECTION METHOD *****")
x_star = bisezione(f1, a, b, N, tolerance)
print("x_star =", x_star)
print("f(x_star) =", f(x_star))
print("f'(x_star) =", f1(x_star))
print()
```

```
***** BISECTION METHOD *****
Iteration  0, x = 0.5000000, f(x) = 0.6250000
Iteration  1, x = 1.0000000, f(x) = 0.0000000
x_star = 1.0
f(x_star) = 0.75
f'(x_star) = 0.0
```

Metodo di Newton

$$\max f(x) = x^3 + 2x - 2x^2 - 0,25x^4$$

$$f'(x) = 3x^2 + 2 - 4x - x^3 \quad g(x)$$

$$f''(x) = 6x - 4 - 3x^2 \quad g'(x)$$

$$x_0 = 1$$

$$a = -0,5$$

$$b = 1,5$$

$$A_0 (x_0, g(x_0)) \quad m = g'(x_0)$$

$$y - y_0 = m(x - x_0)$$

$$\begin{cases} y - g(x_0) = g'(x_0)(x - x_0) \\ y = 0 \end{cases}$$

risolvendo rispetto a x ,

$$x_1 = x_0 - \frac{g(x_0)}{g'(x_0)}$$

$$x_{n+1} = x_n - \frac{g(x_n)}{g'(x_n)}$$

$$x_1 = 1 - \frac{3(1)^2 + 2 - 4(1) - (1)^3}{6(1) - 4 - 3(1)^2} = 1$$

$$f'(x) = 3(1)^2 + 2 - 4(1) - (1)^3 = 0$$

```
In [ ]: def newton(func, func1, x0, N, tolerance):
    x_curr = x0
    for i in range(N):
        f_curr = func(x_curr)
        auxString = "Iteration" + ' ' + '{:3d}'.format(i) + \
                    ", x = " + '{:.7f}'.format(x_curr) + \
                    ", f(x) = " + '{:.7f}'.format(f_curr)
        print(auxString)
        if np.abs(f_curr) < tolerance:
            return x_curr
        f1_curr = func1(x_curr)
        x_next = x_curr - f_curr/f1_curr
        x_curr = x_next
    print("Maximum number of iterations reached")
    return x_curr
```

```
In [ ]: print("***** NEWTON'S METHOD *****")
x_star = newton(f1, f2, x_init, N, tolerance)
print("x_star =", x_star)
print("f(x_star) =", f(x_star))
print("f'(x_star) =", f1(x_star))
print()

***** NEWTON'S METHOD *****
Iteration  0, x = 1.0000000, f(x) = 0.0000000
x_star = 1.0
f(x_star) = 0.75
f'(x_star) = 0.0
```

Metodo delle Secanti

$$\max f(x) = x^3 + 2x - 2x^2 - 0,25x^4$$

$$f'(x) = 3x^2 + 2 - 4x - x^3 \quad g(x)$$

$$a = -0,5$$

$$b = 1,5$$

$$\frac{y - y_1}{y_2 - y_1} = \frac{x - x_1}{x_2 - x_1} \rightarrow \frac{y - g(b)}{g(a) - g(b)} = \frac{x - b}{a - b}$$

$$\frac{0 - g(b)}{g(a) - g(b)} = \frac{x_1 - b}{a - b} \rightarrow x_1 = b - \frac{b-a}{g(b) - g(a)} \times g(b)$$

$$x_{n+1} = x_n - \frac{x_n - a}{g(x_n) - g(a)} \times g(x_n)$$

$$x_1 = b - \frac{b-a}{g(b) - g(a)} \times g(b) =$$

$$= 1,5 - \frac{(1,5) - (-0,5)}{g(b) - g(a)} \times g(b) =$$

$$3(1,5)^2 + 2 - 4(1,5) - (1,5)^3 = \begin{matrix} \swarrow & \searrow \\ 3(-0,5)^2 + 2 - 4(0,5) - (-0,5)^3 = \\ = -0,625 & = 4,875 \end{matrix}$$

$$= 1,5 - \frac{(1,5) - (-0,5)}{(-0,625) - (4,875)} \times (-0,625) = 1,2727$$

```
In [ ]: def secante(func, a, b, N, tolerance):
    fa = func(a)
    fb = func(b)
    x_curr = b - fb*(b - a)/(fb - fa)
    for i in range(N):
        f_curr = func(x_curr)
        auxString = "Iteration" + ' ' + '{:3d}'.format(i) + \
                    ", x = " + '{:.7f}'.format(x_curr) + \
                    ", f(x) = " + '{:.7f}'.format(f_curr)
        print(auxString)
        if np.abs(f_curr) < tolerance:
            return x_curr
        x_next = x_curr - f_curr*(x_curr - a)/(f_curr - fa)
        x_curr = x_next
    print("Maximum number of iterations reached")
    return x_curr
```

```
In [ ]: print("***** SECANT METHOD *****")
x_star = secante(f1, a, b, N, tolerance)
print("x_star =", x_star)
print("f(x_star) =", f(x_star))
print("f'(x_star) =", f1(x_star))
```

```
***** SECANT METHOD *****
Iteration 0, x = 1.2727273, f(x) = -0.2930128
Iteration 1, x = 1.1722183, f(x) = -0.1773261
Iteration 2, x = 1.1135269, f(x) = -0.1149901
Iteration 3, x = 1.0763445, f(x) = -0.0767895
Iteration 4, x = 1.0518995, f(x) = -0.0520393
Iteration 5, x = 1.0355084, f(x) = -0.0355531
Iteration 6, x = 1.0243911, f(x) = -0.0244056
Iteration 7, x = 1.0167976, f(x) = -0.0168023
Iteration 8, x = 1.0115877, f(x) = -0.0115892
Iteration 9, x = 1.0080027, f(x) = -0.0080033
Iteration 10, x = 1.0055311, f(x) = -0.0055313
Iteration 11, x = 1.0038248, f(x) = -0.0038249
Iteration 12, x = 1.0026459, f(x) = -0.0026459
Iteration 13, x = 1.0018308, f(x) = -0.0018308
Iteration 14, x = 1.0012670, f(x) = -0.0012670
Iteration 15, x = 1.0008769, f(x) = -0.0008769
Iteration 16, x = 1.0006070, f(x) = -0.0006070
Iteration 17, x = 1.0004202, f(x) = -0.0004202
Iteration 18, x = 1.0002909, f(x) = -0.0002909
Iteration 19, x = 1.0002014, f(x) = -0.0002014
Iteration 20, x = 1.0001394, f(x) = -0.0001394
Iteration 21, x = 1.0000965, f(x) = -0.0000965
Iteration 22, x = 1.0000668, f(x) = -0.0000668
Iteration 23, x = 1.0000462, f(x) = -0.0000462
Iteration 24, x = 1.0000320, f(x) = -0.0000320
Iteration 25, x = 1.0000222, f(x) = -0.0000222
Iteration 26, x = 1.0000153, f(x) = -0.0000153
Iteration 27, x = 1.0000106, f(x) = -0.0000106
Iteration 28, x = 1.0000074, f(x) = -0.0000074
Iteration 29, x = 1.0000051, f(x) = -0.0000051
Iteration 30, x = 1.0000035, f(x) = -0.0000035
Iteration 31, x = 1.0000024, f(x) = -0.0000024
Iteration 32, x = 1.0000017, f(x) = -0.0000017
Iteration 33, x = 1.0000012, f(x) = -0.0000012
Iteration 34, x = 1.0000008, f(x) = -0.0000008
Iteration 35, x = 1.0000006, f(x) = -0.0000006
Iteration 36, x = 1.0000004, f(x) = -0.0000004
Iteration 37, x = 1.0000003, f(x) = -0.0000003
Iteration 38, x = 1.0000002, f(x) = -0.0000002
Iteration 39, x = 1.0000001, f(x) = -0.0000001
Iteration 40, x = 1.0000001, f(x) = -0.0000001
x_star = 1.0000000891616163
f(x_star) = 0.7499999999999956
f'(x_star) = -8.916161609029416e-08
```

$$\max f(x) = 2x_1x_2 + x_2 - x_1^2 - 2x_2^2$$

Metodo del Gradiente

$$\text{MAX } f(x) = 2x_1 x_2 + x_2 - x_1^2 - 2x_2^2$$

$$\underline{x}_{k+1} = \underline{x}_k + \gamma \nabla f(\underline{x}_k)$$

\downarrow learning rate

$$\nabla f(x) = \begin{bmatrix} 2x_2 - 2x_1 \\ 2x_1 + 1 - 4x_2 \end{bmatrix}$$

$$\underline{x}_0 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

ITERATION 1 (0)

$$\begin{aligned}\underline{x}_1 &= \underline{x}_0 + t \nabla f(\underline{x}_0) = \\ &= \begin{bmatrix} 0 \\ 0 \end{bmatrix} + t \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ t \end{bmatrix}\end{aligned}$$

$$f(0, t) = t - 2t^2$$

$$f'(0, t) = 1 - 4t = 0 \Rightarrow -4t = -1 \Rightarrow t = 1/4$$

$$\underline{x}_1 = \begin{bmatrix} 0 \\ 1/4 \end{bmatrix} = (0, 1/4)$$

```
In [ ]: import numpy as np
import matplotlib.pyplot as plt
import mpl_toolkits
from mpl_toolkits.mplot3d import Axes3D
```

```
In [ ]: def f(x1, x2):
    z = 2*x1*x2 + x2 - x1**2 - 2*x2**2
    return z
```

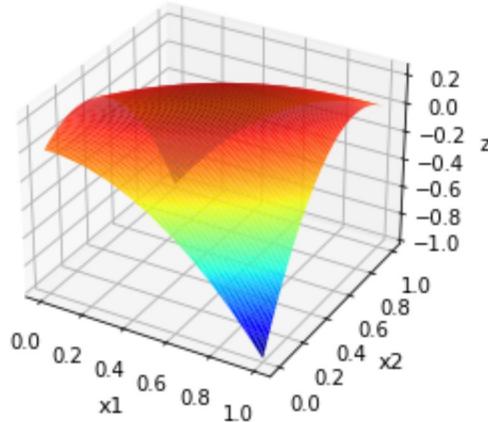
```
In [ ]: def grad_f(x1, x2):
    f1 = 2*x2 - 2*x1
    f2 = 2*x1 + 1 - 4*x2
    return np.array([f1, f2])
```

```
In [ ]: def gradient_batch(func, grad_func, x_init, gamma, MAXITER, eps):
    x1 = []
    x2 = []
    z = []
    x_curr = x_init
    for i in range(MAXITER):
        f_curr = func(x_curr[0], x_curr[1])
        grad_curr = grad_func(x_curr[0], x_curr[1])
        print("it. %3d, x1 = %.6f, x2 =%.6f, f = %.6f" % (i, x_curr[0], \
                                                               x_curr[1], f_curr))
        x1.append(x_curr[0])
        x2.append(x_curr[1])
        z.append(f_curr)
        if np.linalg.norm(grad_curr) < eps:
            return x_curr, x1, x2, z
        x_new = x_curr + gamma*grad_curr
        x_curr = x_new
    print("Maximum number of iterations reached")
    return x_curr, x1, x2, z
```

```
In [ ]: X1 = np.linspace(0, 1, 100)
X2 = np.linspace(0, 1, 100)
x1, x2 = np.meshgrid(X1, X2)
z = f(x1, x2)

fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot_surface(x1, x2, z, rstride = 1, cstride = 1, cmap = 'jet', \
                edgecolor = 'none')
ax.set_title("z = f(x1, x2)", fontsize = 13)
ax.set_xlabel('x1', fontsize = 11)
ax.set_ylabel('x2', fontsize = 11)
ax.set_zlabel('z', fontsize = 10)
plt.show()
```

$$z = f(x_1, x_2)$$



```
In [ ]: x0 = np.array([0, 0])

# SET 1
gamma = 0.1
MAXITER = 10
# SET 2
gamma = 0.2
MAXITER = 10
# SET 3
gamma = 0.25
MAXITER = 15
# SET 4
gamma = .3
MAXITER = 20
# SET 5
gamma = .3
MAXITER = 25
# SET 6
gamma = .3
MAXITER = 30

eps = 1e-3

x_star, x1_g, x2_g, z_g = gradient_batch(f, grad_f, x0, gamma, MAXITER, eps)
```

```

it. 0, x1 = 0.000000, x2 =0.000000, f = 0.000000
it. 1, x1 = 0.000000, x2 =0.300000, f = 0.120000
it. 2, x1 = 0.180000, x2 =0.240000, f = 0.178800
it. 3, x1 = 0.216000, x2 =0.360000, f = 0.209664
it. 4, x1 = 0.302400, x2 =0.357600, f = 0.226675
it. 5, x1 = 0.335520, x2 =0.409920, f = 0.236350
it. 6, x1 = 0.380160, x2 =0.419328, f = 0.241958
it. 7, x1 = 0.403661, x2 =0.444230, f = 0.245244
it. 8, x1 = 0.428003, x2 =0.453350, f = 0.247181
it. 9, x1 = 0.443211, x2 =0.466131, f = 0.248328
it. 10, x1 = 0.456963, x2 =0.472700, f = 0.249007
it. 11, x1 = 0.466406, x2 =0.479638, f = 0.249410
it. 12, x1 = 0.474345, x2 =0.483916, f = 0.249650
it. 13, x1 = 0.480087, x2 =0.487824, f = 0.249792
it. 14, x1 = 0.484729, x2 =0.490488, f = 0.249876
it. 15, x1 = 0.488184, x2 =0.492740, f = 0.249927
it. 16, x1 = 0.490918, x2 =0.494363, f = 0.249956
it. 17, x1 = 0.492985, x2 =0.495678, f = 0.249974
it. 18, x1 = 0.494601, x2 =0.496655, f = 0.249985
it. 19, x1 = 0.495833, x2 =0.497429, f = 0.249991
it. 20, x1 = 0.496791, x2 =0.498014, f = 0.249995
it. 21, x1 = 0.497525, x2 =0.498472, f = 0.249997
it. 22, x1 = 0.498093, x2 =0.498821, f = 0.249998
it. 23, x1 = 0.498530, x2 =0.499092, f = 0.249999
it. 24, x1 = 0.498867, x2 =0.499299, f = 0.249999
it. 25, x1 = 0.499126, x2 =0.499460, f = 0.250000

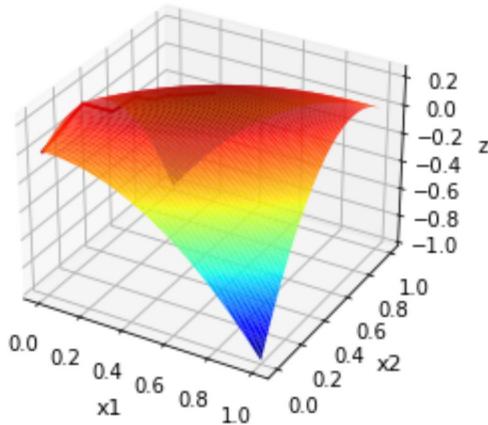
```

```

In [ ]: fig = plt.figure()
ax = plt.axes(projection='3d')
ax.plot(x1_g, x2_g, z_g)
ax.plot_surface(x1, x2, z, rstride = 1, cstride = 1, cmap = 'jet', \
                edgecolor = 'none')
ax.set_title("z = f(x1, x2)", fontsize = 13)
ax.set_xlabel('x1', fontsize = 11)
ax.set_ylabel('x2', fontsize = 11)
ax.set_zlabel('z', fontsize = 10)
ax.plot(x1_g, x2_g, z_g, 'r')
plt.show()

```

$z = f(x_1, x_2)$



Meta-euristiche

Il termine Metaeuristiche si riferisce a una euristica di euristiche, ovvero una procedura di alto livello per migliorare la ricerca.

Le metaeuristiche fanno assunzioni molto più deboli rispetto agli algoritmi visti in precedenza, e perciò sono anche più generiche.

Questa tipologia di algoritmi necessita delle seguenti fasi:

- una o più soluzioni candidate iniziali
- valutazione della qualità di una soluzione candidata
- effettuare una copia della soluzione candidata
- alterare soluzione corrente per ottenere una soluzione candidata differente
- procedura di selezione utile a scegliere le soluzioni candidate

Le metaeuristiche fanno inoltre parte del campo delle ottimizzazioni stocastiche, ovvero una classe di algoritmi che applicano alcuni gradi di casualità per trovare la soluzione ottima per problemi difficili e sono spesso usate per trovare soluzioni a problemi molto complessi.

Tra le procedure di ottimizzazione stocastica troviamo l'"Hill Climbing", che potrebbe essere utilizzato per risolvere i problemi trattati.

Questa procedura sfrutta il gradiente, il gradient ascent, ma non richiede la conoscenza delle sue dimensioni e nemmeno della sua direzione. Testa iterativamente le soluzioni candidate nella regione della nostra attuale soluzione candidata e permette di adottare le nuove se sono migliori. Questo procedimento ci permette di raggiungere la cima fino a raggiungere la soluzione ottimale locale.

Possono inoltre essere testate delle versioni differenti di questo algoritmo, come il "Steepest Ascent Hill-Climbing", o il "Steepest Ascent Hill Climbing with Replacement".

Un altro algoritmo utilizzabile per risolvere i problemi visti in precedenza potrebbe essere il "Random Search", che sfrutta in modo estremo il concetto di esplorazione dello spazio, per questo motivo è utile per funzioni particolari e, nei casi di funzioni unimodali come la prima, farà fatica a trovare una soluzione.

Le metaeuristiche non garantiscono la globalità o la località di una soluzione.

Tra le metaeuristiche principali troviamo il "Simulated Annealing", "Tabu Search" e "Genetic Algorithms".

Algoritmi di ottimizzazione locali, come il metodo di Newton e il Gradiente, riducono il costo della funzione ad ogni iterazione, mentre il Simulated Annealing invece accetta e rifiuta nuove soluzioni in base ad una certa probabilità in ogni iterazione, perciò sarebbe utilizzabile per risolvere i problemi precedenti.

Il principio alla base del tabu search è quello di permettere ai risolutori locali di raggiungere la soluzione ottima evitando la trappola delle soluzioni locali acconsentendo soluzioni non

migliorative, la ricerca a partire da soluzioni già esplorate è evitata usando la memoria. La memoria è strutturata come una tabu list.

Il Tabu Search si basa quindi su una ricerca locale, come il Simulated Annealing, ma ammette spostamenti non migliorativi e, in più, userà della memoria per evitare di ripercorrere negli stessi passaggi, differentemente dal Simulated Annealing.