# Serie 5 - Soluzione
# Ottimizzazione Numerica



## Esercizio 1.1

1. Tramite i seguenti comandi Matlab® si ottiene il grafico in Figura 1.

```
a = -1;
b = 3;
f = @(y) - y.^4/2 + 4*y.^3 - 7*y.^2 - 4*y + 10;
x = 2;

y = linspace(a,b,100);
figure(1)
plot(y,f(y),"LineWidth",2)
hold on
grid on
plot(x,f(x),"xr","MarkerSize",7,"LineWidth",2)
legend("Funzione obiettivo","Minimo")
xlabel("y")
ylabel("\Phi")
```

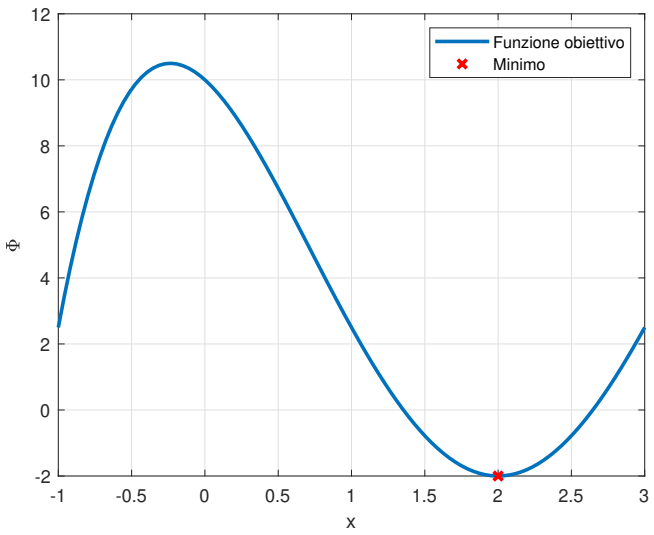


**Figura 1:** Funzione obiettivo con minimo in x = 2

2. Di seguito è riportata una possibile implementazione della function `sezione_aurea.m`:

```
function [xv, k, errv] = sezione_aurea(f, a, b, tol, nmax)
%SEZIONE_AUREA      Approssima il minimo di una funzione scalare
%                   f: [a,b] -> R tramite il metodo della sezione aurea
```

```
%
% Parametri di ingresso:
%
% f: funzione obiettivo
% a: estremo inferiore dell'intervallo
% b: estremo superiore dell'intervallo
% tol:  tolleranza criterio d'arresto basato su stima dell'errore
% nmax: numero massimo di iterazioni ammesse
%
% Parametri di uscita:
%
% xv: vettore delle approssimazioni x_k del minimo
% k: numero di iterazioni eseguite
% errv: vettore delle stime dell'errore e_k
%

k = 0;
phi = (1 + sqrt(5))/2; % rapporto aureo
x = (a + b)/2;
% Vettore delle iterate
xv = x;
err = (b - a)/2;
% Vettore delle stime dell'errore
errv = err;
while (k < nmax && err > tol)
    k = k + 1;
    c = a + (b - a)/(phi + 1);
    d = a + (b - a)/phi;
    if(f(c) > f(d))
        a = c;
    else
        b = d;
    end
    x = (a + b)/2;
    xv = [xv, x];

    err = (b-a)/2;
    errv = [errv, err];
end

if (err <= tol)
         fprintf(['Il metodo della sezione aurea converge in %d iterazioni \n' ...
     ' al punto di minimo x = %f \n '], k, x);
else
     fprintf('Il metodo della sezione aurea non converge in %d iterazioni. \n', k)
end
end
```

3. Approssimiamo il punto di minimo della funzione obiettivo utilizzando il metodo della sezione aurea appena implementato, tramite i comandi Matlab®

```
tol = 1e-6;
nmax = 100;

[xv, k, stimav] = sezione_aurea(f, a, b, tol, nmax);
```

Da cui otteniamo il seguente output

```
Il metodo della sezione aurea converge in 31 iterazioni
 al punto di minimo x = 2.000000
```

L'algoritmo implementato converge correttamente alla soluzione esatta del problema di ottimizzazione in 31 iterazioni.

4. Il confronto dell'errore con la sua stima teorica può essere ottenuto come riportato di seguito

```
err = abs(x - xv(end));
fprintf("\n")
fprintf("Errore dopo %d iterazioni: %2.2e \n", k, err)
fprintf("Stima dell'errore all'iterazione %d: %2.2e \n", k, stimav(end))
```

Coerentemente con i risultati teorici, l'errore $e^{(k)} := |x^{(k)} - x|$ è controllato dalla stima $\frac{|I^{(k)}|}{2}$:

```
Errore dopo 31 iterazioni: 1.27e-07
Stima dell'errore all'iterazione 31: 6.64e-07
```

Possiamo anche verificare che questo valga per ogni $k \geq 0$, andando a rappresentare graficamente l'andamento dell'errore commesso e la sua stima all'aumentare delle iterazioni del metodo.

```
kv = 1:(k+1);
errv = abs(x*ones(1,k+1) - xv);
figure(2)
semilogy(kv, stimav,"LineWidth",2)
hold on
grid on
semilogy(kv, errv,"o--","LineWidth",1)
legend("stima: (b^{(k)} - a^{(k)}) / 2", "errore: | x - x^{(k)} |")
xlabel("iterazioni k")
ylabel("errore e^k")
```

Con i comandi riportati sopra otteniamo il grafico di Figura 2. Come previsto l'errore si mantiene sempre minore della stima e quest'ultima decresce linearmente rispetto al numero di iterazioni, infatti si ha

$$\widetilde{e}^{(k)} = \frac{b-a}{2\,\varphi^k}.$$

Al contrario, la convergenza dell'errore non avviene in modo monotono.

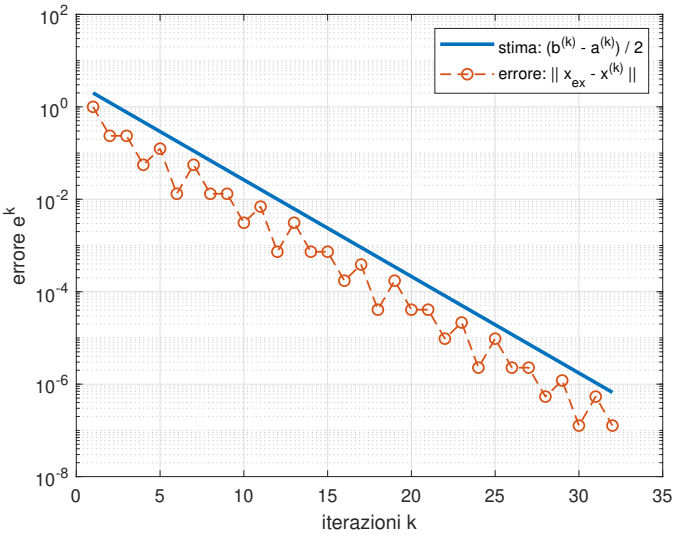


**Figura 2:** Andamento dell'errore e della sua stima rispetto al numero di iterazioni, in scala semilogy.

## Esercizio 2.1

1. Tramite i seguenti comandi Matlab® si definiscono la funzione obiettivo e il suo gradiente.

```
C_L0 = 0;
C_Lalpha = 5;
C_Ldelta = 0.3;
C_D0 = 0.02;
C_Dalpha = 0.5;
C_Ddelta = 0.05;

Phi = @(y1,y2) - (C_L0 + C_Lalpha * y1 + C_Ldelta * y2) ./ ...
    (C_D0 + C_Dalpha * y1.^2 + C_Ddelta * y2.^2);

GradPhi = @(y1, y2) [ 500*(50*y1.^2 + 6*y1*y2 - 5*y2.^2 - 2) ./ ...
                        ( 50*y1.^2 + 5*y2.^2 + 2).^2 ;
                      10*(-150*y1.^2 + 500*y1*y2 + 15*y2.^2 - 6) ./ ...
                        (50*y1.^2 + 5*y2.^2 + 2).^2];
```

   Si rappresenta il funzionale $\Phi$ tramite la funzione `plot_phi.m`, ottenendo i risultati in Figura 3:

```
x1_min_ex = 0.196494373584908;
x2_min_ex = 0.117896623783779;
x_ex   = [x1_min_ex, x2_min_ex]';

a1 = 0;
b1 = 0.4;
a2 = -0.1;
b2 = 0.3;
```

```
y1 = linspace(a1,b1,100);
y2 = linspace(a2, b2, 100);

figure()
plot_phi(y1, y2, Phi, '\Phi(y) = -L(y) / D(y)');
```

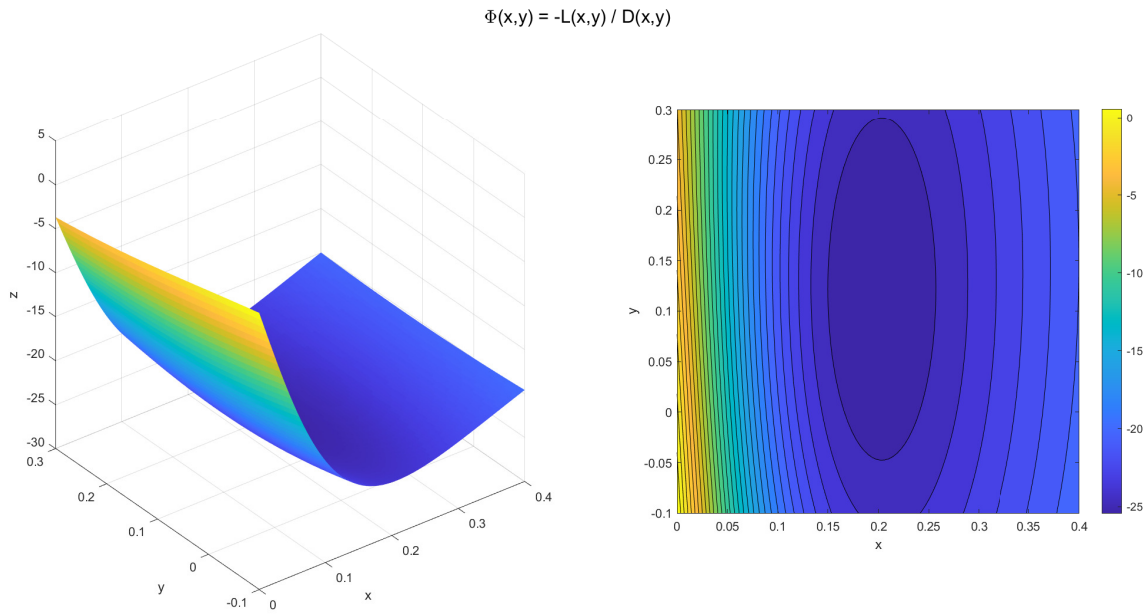


**Figura 3:** Funzione obiettivo con minimo in $\mathbf{x} = (0.196,\, 0.116)^T$

2. Di seguito è riportata una possibile implementazione della function `gradiente_coniugato_opt.m`:

```
function [xvect, it] = gradiente_coniugato_opt(Phi, GradPhi, flag_metodo, x0, toll, nmax)
% function [xvect, it] = gradiente_coniugato_opt(Phi, GradPhi, flag_metodo, x0, toll, nmax )
% Algoritmo gradiente (gradiente coniugato) per la risoluzione di problemi
% di ottimizzazione per funzionali in 2-dimensioni. Tramite il terzo input
% (flag_metodo) e' possibile scegliere se usare il metodo del gradiente
% o il metodo del gradiente coniugato (con \beta_k di Fletcher-Reeves)
%
% INPUT Phi         : (handle function) Funzionale
%       GradPhi     : (handle function) Gradiente del funzionale
%       flag_metodo : (string) flag per scegliere tra gradiente e gradiente
%                     coniugato
%       x0          : (double, vettore) Guess iniziale
%       toll        : (double) tolleranza
%       nmax        : (int) numero massimo di iterazioni
%
% OUTPUT xvect : (double, matrice) matrice che, su ogni colonna, contiene
%                la soluzione a ogni iterazione
%        it    : (int) numero di iterazioni

% Inizializzo variabili per ciclo iterativo
it   = 0;
```

```
err = toll+1;

% Inizializzo la matrice soluzione
xvect = x0;

% Calcolo la direzione iniziale di discesa
d = -GradPhi(x0(1), x0(2));

% Parametri backtracking
c1_bt   = 1e-4;
rho_bt  = 0.3;
nmax_bt = 10;

% Ciclo while per calcolo x^{it+1} (it = 0, 1, ...)
while it < nmax && err > toll

    % Calcolo il nuovo valore di alpha_k
    [alpha_k,~] = backtracking(Phi, GradPhi, x0, d, c1_bt, rho_bt, nmax_bt);

    % Calcolo il nuovo valore di x
    x_new = x0 + alpha_k*d;

    % Swtich per scelta metodo gradiente o gradiente coniugato
    switch flag_metodo
        case 'G' % gradiente
            beta_k = 0;
        case 'FR' % gradiente coniugato (Fletcher-Reeves)
            beta_k = norm(GradPhi(x_new(1),x_new(2)))^2 / norm(GradPhi(x0(1),x0(2)))^2;
        otherwise
            error('metodo CG non definito');
    end

    % Aggiorno la direzione di discesa (se beta_k = 0 -> gradiente)
    d = -GradPhi(x_new(1),x_new(2)) + beta_k*d;

    % Aggiorno quantita' per metodo iterativo
    it    = it+1;
    err   = norm(GradPhi(x_new(1), x_new(2)));
    x0    = x_new;
    xvect = [xvect, x_new];

end

switch flag_metodo
    case 'G' % gradiente
        if err <= toll
             fprintf(['\n Il metodo del gradiente converge in %d iterazioni \n' ...
                 'al punto di minimo x = (%f,%f)^T \n '], it, xvect(1,end), xvect(2,end));
        else
             fprintf('\n Il metodo del gradiente non converge in %d iterazioni. \n', it)
        end

    case 'FR' % gradiente coniugato (Fletcher-Reeves)
        if err <= toll
             fprintf(['\n Il metodo del gradiente coniugato (Fletcher-Reeves)
             converge in %d iterazioni \n' ...
                 'al punto di minimo x = (%f,%f)^T \n '], it, xvect(1,end), xvect(2,end));
```

```
        else
             fprintf('\n Il metodo del gradiente coniugato (Fletcher-Reeves)
             non converge in %d iterazioni. \n',  it)
        end
end

end
```

3. Risolviamo il problema di ottimizzazione con i due metodi richiesti, utilizzando il metodo di backtracking per identificare $\alpha_k$ per ogni $k \geq 0$.

```
x0   = [0.1, 0.05]';
toll = 1e-5;
nmax = 100;

% Gradiente con backtracking
[xvect_G_BT, it_G_BT] = gradiente_coniugato_opt(Phi, GradPhi, 'G', x0, toll, nmax );
% Gradiente coniugato Fletcher-Reeves con backtracking
[xvect_FR_BT, it_FR_BT] = gradiente_coniugato_opt(Phi, GradPhi, 'FR', x0, toll, nmax );

% Visualizzo le soluzioni
figure()
plot_phi(x, y, Phi, 'Gradiente con BT');
plot_soluzioni(Phi, xvect_G_BT, [a1 b1], [a2 b2], x_ex);

figure()
plot_phi(x, y, Phi, 'Gradiente coniugato FR con BT');
plot_soluzioni(Phi, xvect_FR_BT, [a1 b1], [a2 b2], x_ex);
```

Nelle Figure 4 e 5 è riportata la successione delle iterate sulle curve di livello della funzione $\Phi$ per i due metodi.

Gli output restituiti dalla function `gradiente_coniugato_opt` nei tre casi sono i seguenti

```
 Il metodo del gradiente converge in 54 iterazioni
al punto di minimo x = (0.196494,0.117897)^T

 Il metodo del gradiente coniugato (Fletcher-Reeves) converge in 18 iterazioni
al punto di minimo x = (0.196494,0.117897)^T
```

Utilizzando il backtracking, con questa particolare scelta dei parametri dei metodi, riusciamo a raggiungere la convergenza al punto di minimo in entrambi i casi.

4. Risolviamo adesso mettendo come guess iniziale $\mathbf{x_0} = (0, 0.1)^T$.

```
x0_1 = [0,0.1]';
toll = 1e-5;
nmax = 100;

% Gradiente con backtracking
[xvect_G_BT_1, it_G_BT_1] = gradiente_coniugato_opt(Phi, GradPhi, 'G', ...
                                    x0_1, toll, nmax );
```

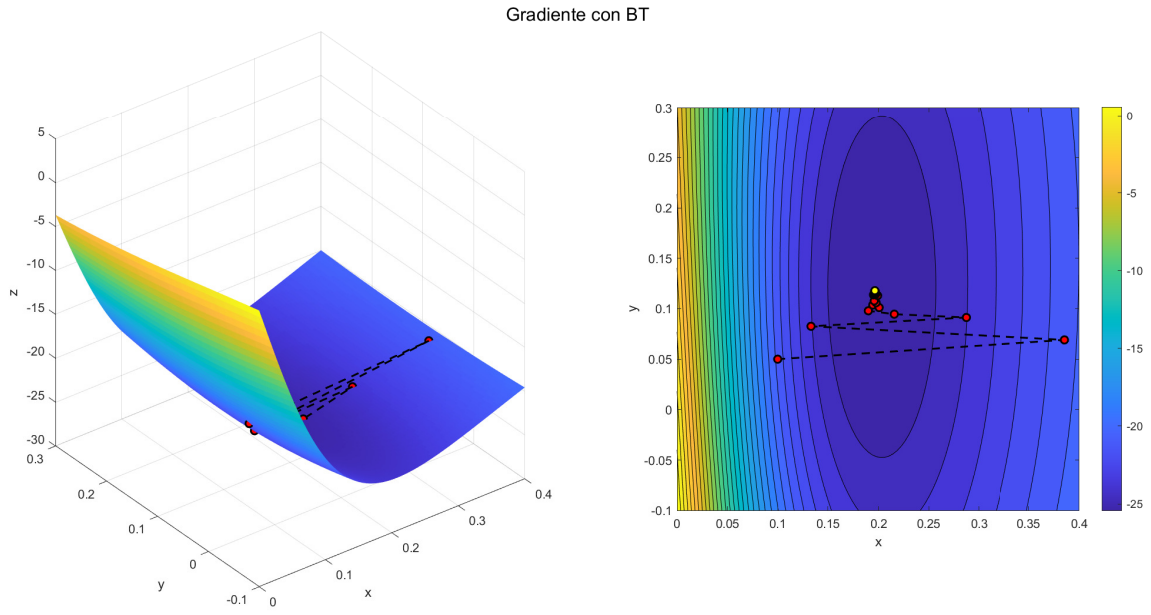


**Figura 4:** Visualizzazione grafica delle iterate per il metodo del gradiente sfruttando il metodo backtracking per $\alpha_k$ , partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

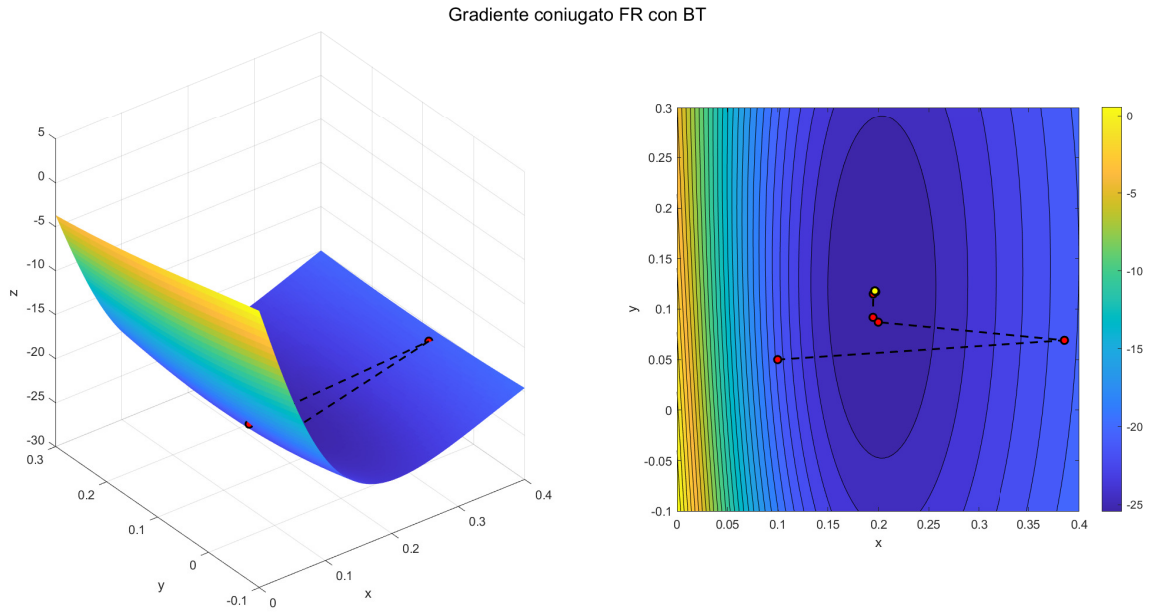


**Figura 5:** Visualizzazione grafica delle iterate per il metodo del gradiente coniugato Fletcher–Reeves sfruttando il metodo backtracking a per $\alpha_k$ , partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

```
% Gradiente coniugato Fletcher-Reeves con backtracking
[xvect_FR_BT_1, it_FR_BT_1] = gradiente_coniugato_opt(Phi, GradPhi, 'FR', ...
x0_1, toll, nmax );

% Qui osserviqmo che anche aumentando a nmax = 1000 non arriviamo
% a convergenza con la tolleranza richiesta
[xvect_FR_BT_1e3, it_FR_BT_1e3] = gradiente_coniugato_opt(Phi, GradPhi, 'FR', ...
                                  x0_1, toll, 1e3);

% Visualizzo le soluzioni
figure()
plot_phi(x, y, Phi, 'Gradiente coniugato FR con BT');
plot_soluzioni(Phi, xvect_FR_BT_1, [a1 b1], [a2 b2], x_ex);
```

I comandi sopra riportati generano l'output seguente:

```
 Il metodo del gradiente converge in 57 iterazioni
al punto di minimo x = (0.196494,0.117897)^T

 Il metodo del gradiente coniugato (Fletcher-Reeves) non converge in 100 iterazioni.

 Il metodo del gradiente coniugato (Fletcher-Reeves) non converge in 1000 iterazioni.
```

Il metodo del gradiente coniugato con $\beta_k$ di Fletcher–Reeves e backtracking per $\alpha_k$ non arriva a convergenza, neanche aumentando da 100 a 1000 il numero massimo di iterazioni. Al contrario, a parità di setup, con $\beta_k = 0$ (gradiente) l'algoritmo con backtracking riesce a convergere al punto di minimo.

Questi risultati dimostrano la difficoltà di convergenza del metodo del gradiente coniugato nel caso in cui non venga scelta una buona iterata iniziale oppure non vengano settati in maniera ottimale i parametri del metodo. Il metodo del gradiente coniugato in questo caso risulta meno robusto del metodo del gradiente

Testiamo quindi il metodo del gradiente coniugato Fletcher–Reeves con un diverso setting dei parametri di backtracking, che andiamo a modificare direttamente nella funzione `gradiente_coniugato_opt.m`.

```
% Parametri backtracking
c1_bt   = 2e-1;
rho_bt  = 0.45;
nmax_bt = 10;
```

In questo caso, otteniamo l'output

```
 Il metodo del gradiente coniugato (Fletcher-Reeves) converge in 21 iterazioni
al punto di minimo x = (0.196494,0.117897)^T
```

Abbiamo così verificato la grande sensibilità dei metodi del gradiente rispetto ai parametri scelti e che sia necessario un settaggio ad-hoc degli stessi a seconda del problema di minimizzazione preso in considerazione.

Nelle Figure 6 e 7 è visualizzata la successione delle iterate con i due diversi set di parametri di backtracking per il gradiente coniugato Fletcher–Reeves.

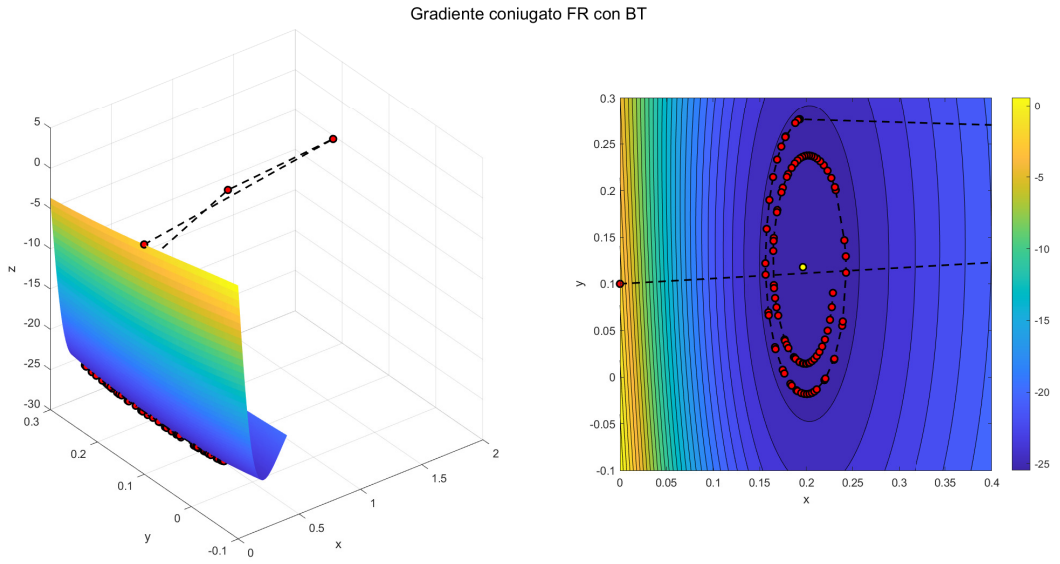


**Figura 6:** Visualizzazione grafica delle iterate per il metodi del gradiente coniugato (FR) sfruttando il metodo backtracking per $\alpha_k$, con due diverse scelta dei parametri $\rho = 0.3$ e $c_1 = 10^{-4}$, partendo dall'iterata iniziale $\mathbf{x_0} = (0, 0.1)^T$

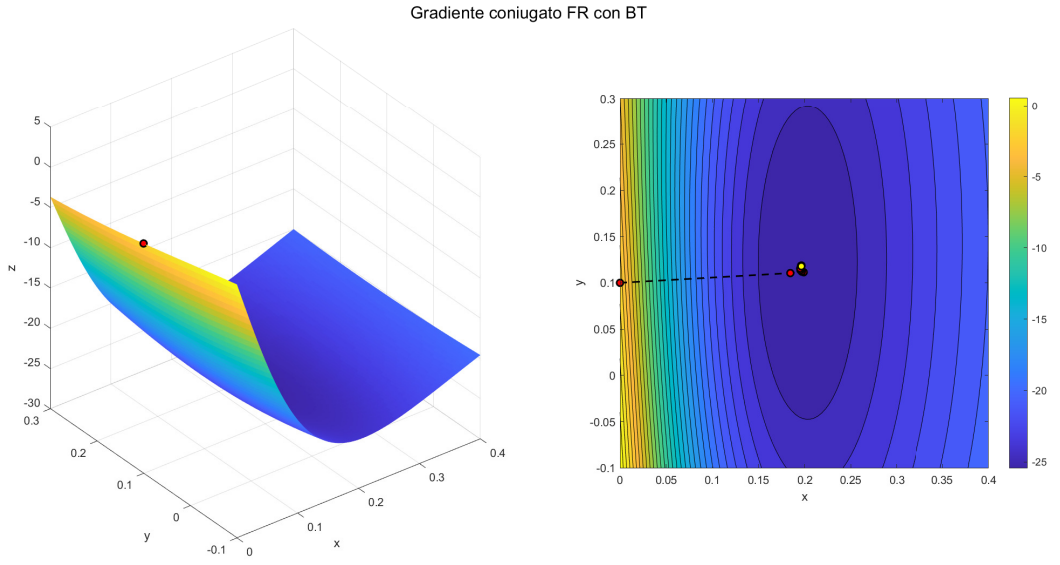


**Figura 7:** Visualizzazione grafica delle iterate per il metodi del gradiente coniugato (FR) sfruttando il metodo backtracking per $\alpha_k$, con due diverse scelta dei parametri $\rho = 0.45$ e $c_1 = 2 \cdot 10^{-1}$, partendo dall'iterata iniziale $\mathbf{x_0} = (0, 0.1)^T$

5. Tramite i seguenti comandi Matlab® rappresentiamo graficamente gli errori contro il numero di iterazioni (in scala semilogaritmica):

```
err_G_BT = zeros(size(xvect_G_BT,2),1);
err_FR_BT = zeros(size(xvect_FR_BT,2),1);

for i = 1:size(xvect_G_BT,2)
    err_G_BT(i) = norm(x_ex - xvect_G_BT(:,i));
end

for i = 1:size(xvect_FR_BT,2)
    err_FR_BT(i) = norm(x_ex - xvect_FR_BT(:,i));
end

figure()
semilogy(err_G_BT,'*-','LineWidth',2.5);
hold on;
grid on;
semilogy(err_FR_BT,'*-','LineWidth',2.5);
legend('Grad BT', 'ConjGrad FR BT');
title('Confronto riduzione errori');
xlabel('#iterazioni');
ylabel('Errori');
title('Metodi del gradiente con backtracking','FontSize',14)
```

E otteniamo gli andamenti riportati in Figura 8.

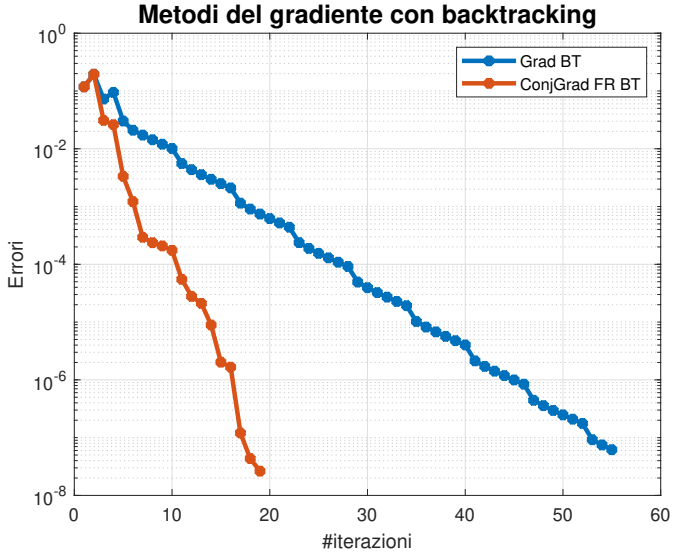


**Figura 8:** Confronto fra la convergenza dei metodi del gradiente e gradiente coniugato, con $\alpha_k$ determinato con backtracking partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

## Esercizio 2.2

1. Tramite i seguenti comandi Matlab® si ottiene il grafico in Figura 9.

```
Phi     = @(y1,y2) 3*y1.^2 + y2.^2 - y1/4 - y2/6 + exp(-2*y1.*y2);

GradPhi = @(y1,y2) [6*y1 - 1/4 - 2*y2.*exp(-2*y1.*y2);
                    2*y2 - 1/6 - 2*y1.*exp(-2*y1.*y2)];

HessPhi = @(y1,y2) [6 + 4*(y2.^2).*exp(-2*y1.*y2),...
                    -2*exp(-2*y1.*y2) + 4*y1.*y2.*exp(-2*y1.*y2);
                    -2*exp(-2*y1.*y2) + 4*y1.*y2.*exp(-2*y1.*y2),...
                    2 + 4*(y1.^2).*exp(-2*y1.*y2)];

% Plotto il funzionale
a1 = -0.15;
b1 = 0.25;
a2 = -0.05;
b2 = 0.35;

y1 = linspace(a1,b1,100);
y2 = linspace(a2,b2,100);

figure()
plot_phi(y1, y2, Phi,'\Phi(x,y) = 3x^2 + y^2 - x/4 - y/6 + exp(-2xy)');
```

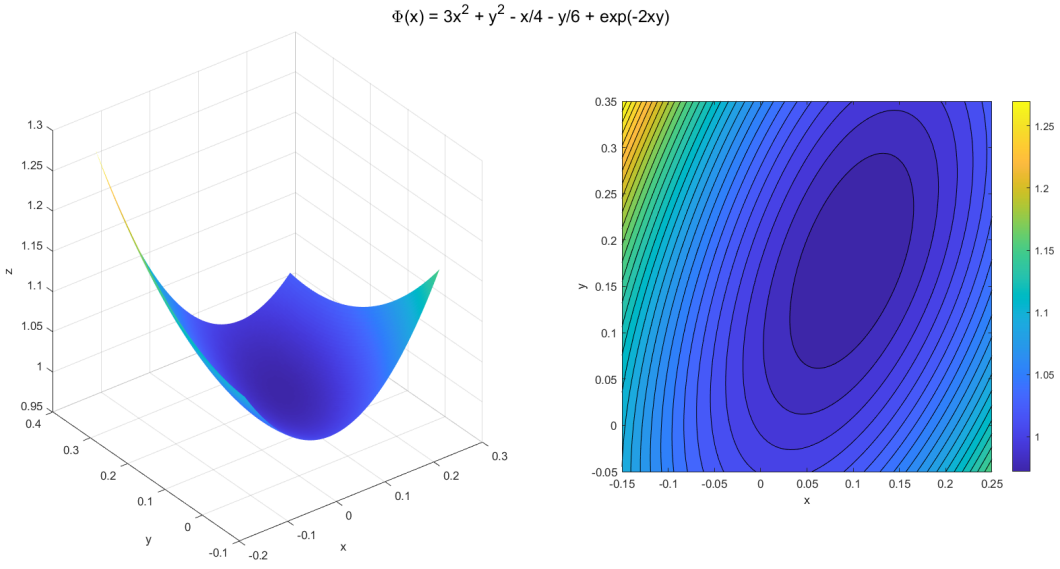


**Figura 9:** Funzione obiettivo con minimo in $\mathbf{x} = (0.099299729019640,\ 0.179161952163217)^T$

e tramite i seguenti comandi si verifica che $\mathbf{x}$ è un punto di minimo per $\Phi$:

```
% Verifico che x_ex sia un minimo locale
x_ex = [0.099299729019640; 0.179161952163217];
```

```
GradPhi_x_ex = GradPhi(x_ex(1),x_ex(2));
HessPhi_x_ex = HessPhi(x_ex(1),x_ex(2));

fprintf('Prima componente Grad(Phi) in x_ex = %.10f\n', GradPhi_x_ex(1));
Prima componente Grad(Phi) in x_ex = -0.0000000000

fprintf('Seconda componente Grad(Phi) in x_ex = %.10f\n', GradPhi_x_ex(2));
Seconda componente Grad(Phi) in x_ex = -0.0000000000

fprintf('Minimo autovalore Hess(Phi) in x_ex = %f\n', min(eig(HessPhi(x_ex(1),x_ex(2)))))
Minimo autovalore Hess(Phi) in x_ex = 1.317222
```

2. Di seguito è riportata una possibile implementazione della function `newton_opt.m`:

```
function [xvect, it] = newton_opt(GradPhi, HessPhi, x0, toll, nmax )
% function [xvect, it] = newton_opt(GradPhi, HessPhi, xv, toll, nmax )
% Algoritmo di Newton per la risoluzione di problemi di ottimizzazione per
% funzionali in 2-dimensioni. Consideriamo il metodo di Newton esatto,
% quindi fissiamo $\alpha_k = 1$.
%
% INPUT GradPhi: (handle function) Gradiente del funzionale
%       HessPhi: (handle function) Hessiano del funzionale
%       x0     : (double, vettore) Guess iniziale
%       toll   : (double) tolleranza
%       nmax   : (int) numero massimo di iterazioni
%
% OUTPUT xvect : (double, matrice) matrice che, su ogni colonna, contiene
%                la soluzione a ogni iterazione
%        it    : (int) numero di iterazioni

% Inizializzo variabili per ciclo iterativo
it  = 0;
err = toll+1;

% Inizializzo la matrice soluzione
xvect = x0;

% Calcolo la direzione iniziale di discesa
d = - HessPhi(x0(1),x0(2)) \ GradPhi(x0(1),x0(2));

% Definiamo lo step-size alpha_k
alpha_k = 1;

% Ciclo while per calcolo x^{it+1} (it = 0, 1, ...)
while it < nmax && err > toll

    % Calcolo il nuovo valore di x
    x_new = x0 + alpha_k*d;

    % Aggiorno la direzione di discesa
    d = - HessPhi(x_new(1), x_new(2)) \ GradPhi(x_new(1), x_new(2));

    % Aggiorno quantita' per metodo iterativo
    it    = it+1;
    err   = norm(GradPhi(x_new(1), x_new(2)));
    x0    = x_new;
```

```
    xvect = [xvect, x_new];

end

if err <= toll
     fprintf(['\nIl metodo di Newton converge in %d iterazioni \n' ...
         'al punto di minimo x = (%f,%f) \n '], it, xvect(1,end), xvect(2,end));
else
     fprintf('Il metodo di Newton non converge in %d iterazioni. \n', it)
end

end
```

3. Approssimiamo il punto di minimo della funzione obiettivo utilizzando il metodo di Newton tramite i comandi Matlab®

```
x0   = [-0.14; 0.14];
toll = 1e-8;
nmax = 200;
[xv_newton, it_newton] = newton_opt(GradPhi, HessPhi, x0, toll, nmax);

% Visualizzo le soluzioni
figure()
plot_phi(y1, y2, Phi, 'Metodo di Newton');
plot_soluzioni(Phi, xv_newton, [a1 b1], [a2 b2], x_ex);
```

Da cui otteniamo il seguente output

```
Il metodo di Newton converge in 4 iterazioni
al punto di minimo x = (0.099300,0.179162)^T
```

L'algoritmo implementato converge correttamente alla soluzione esatta del problema di ottimizzazione in 4 iterazioni. In Figura 10 riportiamo le iterate del metodo di Newton.

4. Di seguito è riportata una possibile implementazione della function `bfgs.m`:

```
function [xvect, it] = bfgs(Phi, GradPhi, x0, toll, nmax)
% function [xvect, it] = bfgs(Phi, GradPhi, x0, toll, nmax)
% Algoritmo BFGS per la risoluzione di problemi di ottimizzazione per
% funzionali in 2-dimensioni. Consideriamo il metodo di backtracking per
% aggiornare lo step-size alpha_k
%
% INPUT Phi    : (handle function) Funzionale
%       GradPhi: (handle function) Gradiente del funzionale
%       x0     : (double, vettore) Guess iniziale
%       toll   : (double) tolleranza
%       nmax   : (int) numero massimo di iterazioni
%
% OUTPUT xvect : (double, matrice) matrice che, su ogni colonna, contiene
%                la soluzione a ogni iterazione
%        it    : (int) numero di iterazioni

% Inizializzo variabili per ciclo iterativo
it  = 0;
```

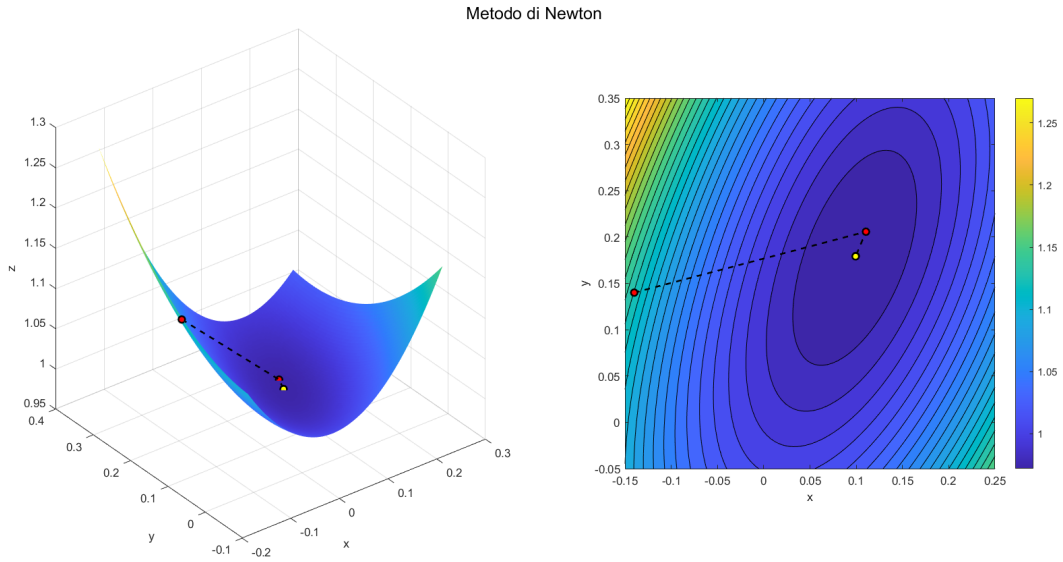


**Figura 10:** Visualizzazione grafica delle iterate per il metodo di Newton

```
err = toll+1;

% Inizializzo la matrice soluzione
xvect = x0;

% Calcolo la direzione iniziale di discesa
I  = eye(length(x0));
Bk = I;
d  = - Bk * GradPhi(x0(1), x0(2));

% Parametri backtracking
c1_bt   = 1e-4;
rho_bt  = 0.5;
nmax_bt = 10;

% Ciclo while per calcolo x^{it+1} (it = 0, 1, ...)
while it < nmax && err > toll

    % Calcolo il nuovo valore di alpha_k con backtracking
    [alpha_k,~] = backtracking(Phi, GradPhi, x0, d, c1_bt, rho_bt, nmax_bt);

    % Calcolo il nuovo valore di x
    x_new = x0 + alpha_k*d;

    % Calcolo Bk (approssimazione inversa Hessiano di Phi)
    delta_k = x_new - x0;
    s_k     = GradPhi(x_new(1), x_new(2)) - GradPhi(x0(1), x0(2));
    rho_k   = 1 / (s_k' * delta_k);

    Bk      = (I - rho_k*delta_k*s_k')*Bk*(I - rho_k*s_k*delta_k') +...
    (rho_k*delta_k)*delta_k';
```

```
    % Aggiorno la direzione di discesa
    d = - Bk*GradPhi(x_new(1), x_new(2));

    % Aggiorno quantita' per metodo iterativo
    it    = it+1;
    err   = norm(GradPhi(x_new(1), x_new(2)));
    x0    = x_new;
    xvect = [xvect, x_new];

end

if err <= toll
     fprintf(['\nIl metodo BFGS converge in %d iterazioni \n' ...
         'al punto di minimo x = (%f,%f) \n '], it, xvect(1,end), xvect(2,end));
else
     fprintf('Il metodo BFGS non converge in %d iterazioni. \n', it)
end

end
```

5. Approssimiamo il punto di minimo della funzione obiettivo utilizzando il metodo BFGS tramite i comandi Matlab®

```
[xv_bfgs,   it_bfgs] = bfgs( Phi, GradPhi, x0, toll, nmax );

% Visualizzo le soluzioni
figure()
plot_phi(y1, y2, Phi, 'Metodo BFGS');
plot_soluzioni(Phi, xv_bfgs, [a1 b1], [a2 b2], x_ex);
```

   Da cui otteniamo il seguente output

```
Il metodo BFGS converge in 29 iterazioni
al punto di minimo x = (0.099300,0.179162)^T
```

   L'algoritmo implementato converge correttamente alla soluzione esatta del problema di ottimizzazione in 29 iterazioni. In Figura 11 riportiamo le iterate del metodo BFGS.

6. Tramite i seguenti comandi Matlab® plottiamo gli errori contro il numero di iterazioni (in scala semilogaritmica):

```
err_newton = zeros(size(xv_newton,2),1);
err_bfgs   = zeros(size(xv_bfgs,2),1);

for i = 1:size(xv_newton,2)
    err_newton(i) = norm(x_ex - xv_newton(:,i));
end

for i = 1:size(xv_bfgs,2)
    err_bfgs(i) = norm(x_ex - xv_bfgs(:,i));
end

figure()
```

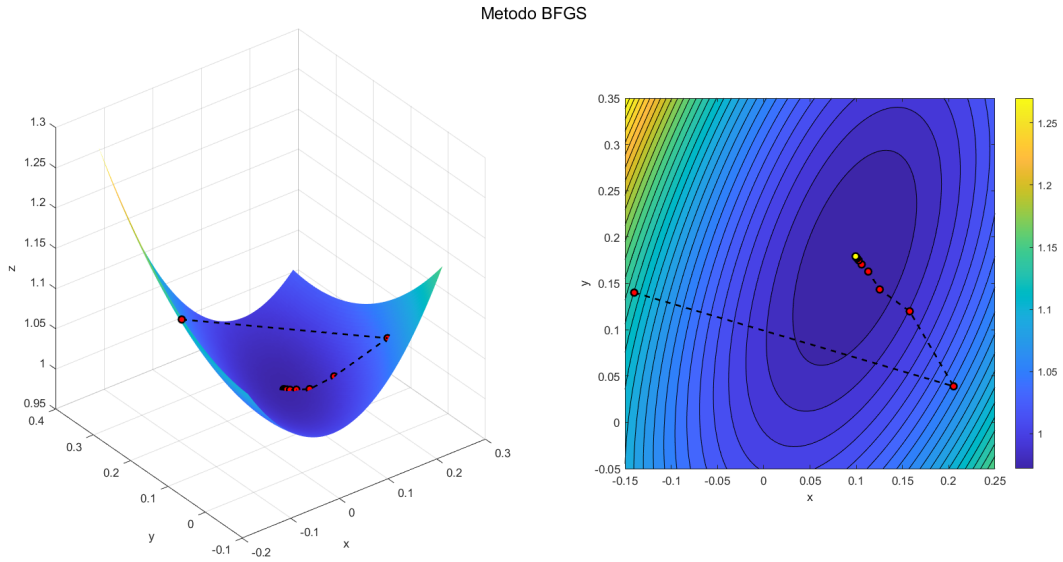


**Figura 11:** Visualizzazione grafica delle iterate per il metodo BFGS

```
semilogy(err_newton,'*-','LineWidth',2.5);
hold on;
grid on;
semilogy(err_bfgs,'*-','LineWidth',2.5);
legend('Newton','BFGS');
title('Confronto riduzione errori');
xlabel('#iterazioni');
ylabel('Errori');
```

In Figura 12 riportiamo i risultati ottenuti: Come atteso, l'errore calcolato per il metodo di Newton decade più rapidamente rispetto al metodo BFGS. Questo è coerente con la teoria poichè Newton è un metodo di ordine 2, mentre BFGS è un metodo (super)lineare.

## Esercizio 3.1

1. Di seguito è riportata una possibile implementazione della function `gradiente_coniugato_opt_BT_SA.m`

```
function [xvect, it] = gradiente_coniugato_opt_BT_SA(Phi, GradPhi, flag_metodo, flag_alpha,
                        x0, toll, nmax )
% function [xvect, it] = gradiente_coniugato_opt(Phi, GradPhi, flag_metodo, x0, toll, nmax )
% Algoritmo gradiente (gradiente coniugato) per la risoluzione di problemi
% di ottimizzazione per funzionali in 2-dimensioni. Tramite il terzo input
% (flag_metodo) si sceglie se usare il metodo del gradiente
% o il metodo del gradiente coniugato (con \beta_k di Fletcher-Reeves)
%
% INPUT Phi         : (handle function) Funzionale
%       GradPhi     : (handle function) Gradiente del funzionale
%       flag_metodo : (string) flag per scegliere tra gradiente e gradiente
%                     coniugato
%       flag_alpha : (string) flag per scegliere tra sezione aurea e
```

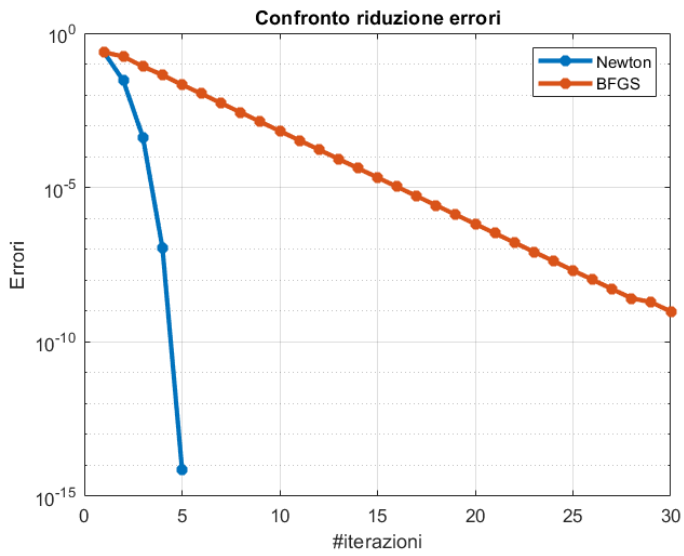


**Figura 12:** Confronto tra la convergenza del metodo di Newton e la convergenza del metodo BFGS

```
%                       backtracking
%       x0          : (double, vettore) Guess iniziale
%       toll        : (double) tolleranza
%       nmax        : (int) numero massimo di iterazioni
%
% OUTPUT xvect : (double, matrice) matrice che, su ogni colonna, contiene
%                la soluzione a ogni iterazione
%        it    : (int) numero di iterazioni

% Inizializzo variabili per ciclo iterativo
it  = 0;
err = toll+1;

% Inizializzo la matrice soluzione
xvect = x0;

% Calcolo la direzione iniziale di discesa
d = -GradPhi(x0(1), x0(2));

% Parametri backtracking
c1_bt   = 1e-4;
rho_bt  = 0.3;
nmax_bt = 10;

% Parametri sezione aurea
a = 0;
b = 1;
tol_sa = toll;
nmax_sa = 20;

% Ciclo while per calcolo x^{it+1} (it = 0, 1, ...)
while it < nmax && err > toll
```

```
    % Calcolo il nuovo valore di alpha_k
    switch flag_alpha
        case 'BT'
            [alpha_k,~] = backtracking(Phi, GradPhi, x0, d, c1_bt, rho_bt, nmax_bt);
        case 'SA'
            f = @(alpha) Phi(x0(1) + alpha*d(1), x0(2) + alpha*d(2));
            [alpha_kv,~, ~] = sezione_aurea(f, a, b, tol_sa, nmax_sa);
            alpha_k = alpha_kv(end);
        otherwise
            error('metodo per calcolo di alphak non definito');
    end

    % Calcolo il nuovo valore di x
    x_new = x0 + alpha_k*d;

    % Swtich per scelta metodo gradiente o gradiente coniugato
    switch flag_metodo
        case 'G' % gradiente
            beta_k = 0;
        case 'FR' % gradiente coniugato (Fletcher-Reeves)
            beta_k = norm(GradPhi(x_new(1),x_new(2)))^2 / ...
                     norm(GradPhi(x0(1),x0(2)))^2;
        otherwise
            error('metodo CG non definito');
    end

    % Aggiorno la direzione di discesa (se beta_k = 0 -> gradiente)
    d = -GradPhi(x_new(1),x_new(2)) + beta_k*d;

    % Aggiorno quantita' per metodo iterativo
    it    = it+1;
    err   = norm(GradPhi(x_new(1), x_new(2)));
    x0    = x_new;
    xvect = [xvect, x_new];

end

switch flag_metodo
    case 'G' % gradiente
        if err <= toll
             fprintf(['\n Il metodo del gradiente con %s converge
                    in %d iterazioni \n' ...
                    'al punto di minimo x = (%f,%f) \n '],
                    flag_alpha, it, xvect(1,end), xvect(2,end));
        else
             fprintf('\n Il metodo del gradiente con %s non converge
                    in %d iterazioni. \n',
                    flag_alpha, it)
        end

    case 'FR' % gradiente coniugato (Fletcher-Reeves)
        if err <= toll
             fprintf(['\n Il metodo del gradiente coniugato (Fletcher-Reeves)
                     con %s converge in %d iterazioni \n' ...
                     'al punto di minimo x = (%f,%f) \n '],
                     flag_alpha, it, xvect(1,end), xvect(2,end));
        else
```

```
            fprintf('\n Il metodo del gradiente coniugato (Fletcher-Reeves)
                    con %s non converge in %d iterazioni. \n',
                    flag_alpha, it)
        end
end

end
```

2. Tramite i seguenti comandi Matlab® si definiscono la funzione obiettivo e il suo gradiente.

```
C_L0 = 0;
C_Lalpha = 5;
C_Ldelta = 0.3;
C_D0 = 0.02;
C_Dalpha = 0.5;
C_Ddelta = 0.05;

Phi = @(y1,y2) - (C_L0 + C_Lalpha * y1 + C_Ldelta * y2) ./ ...
    (C_D0 + C_Dalpha * y1.^2 + C_Ddelta * y2.^2);

GradPhi = @(y1, y2) [ 500*(50*y1.^2 + 6*y1*y2 - 5*y2.^2 - 2) ./ ...
                        ( 50*y1.^2 + 5*y2.^2 + 2).^2 ;
                      10*(-150*y1.^2 + 500*y1*y2 + 15*y2.^2 - 6) ./ ...
                        (50*y1.^2 + 5*y2.^2 + 2).^2];


x1_min_ex = 0.196494373584908;
x2_min_ex = 0.117896623783779;
x_ex   = [x1_min_ex, x2_min_ex]';

a1 = 0;
b1 = 0.4;
a2 = -0.1;
b2 = 0.3;
y1 = linspace(a1,b1,100);
y2 = linspace(a2, b2, 100);

figure()
plot_phi(y1, y2, Phi, '\Phi(y) = -L(y) / D(y)');
```

Dopo di che, si risolve il problema di minimizzazione utilizzando il metodo della sezione aurea per identificare $\alpha_k$ per ogni $k \geq 1$.

```
x0 = [0.1,0.05]';
toll = 1e-5;
nmax = 100;

% Gradiente con sezione_aurea
[xvect_G_SA, it_G_SA] = gradiente_coniugato_opt_BT_SA(Phi, GradPhi, 'G', 'SA', ...
                    x0, toll, nmax );

% Gradiente coniugato Fletcher-Reeves con sezione aurea
[xvect_FR_SA, it_FR_SA] = gradiente_coniugato_opt_BT_SA(Phi, GradPhi, 'FR', 'SA', ...
                    x0, toll, nmax );
```

```
% Visualizzo le soluzioni
figure()
plot_phi(x, y, Phi, 'Gradiente con SA');
plot_soluzioni(Phi, xvect_G_SA, [a1 b1], [a2 b2], x_ex);

figure()
plot_phi(x, y, Phi, 'Gradiente coniugato FR con SA');
plot_soluzioni(Phi, xvect_FR_SA, [a1 b1], [a2 b2], x_ex);
```

Nelle Figure 13 e 14 è riportata la successione delle iterate sulle curve di livello della funzione $\Phi$ per i due metodi. Osserviamo come qualitativamente entrambi gli approcci convergono con un andamento analogo al punto di minimo.

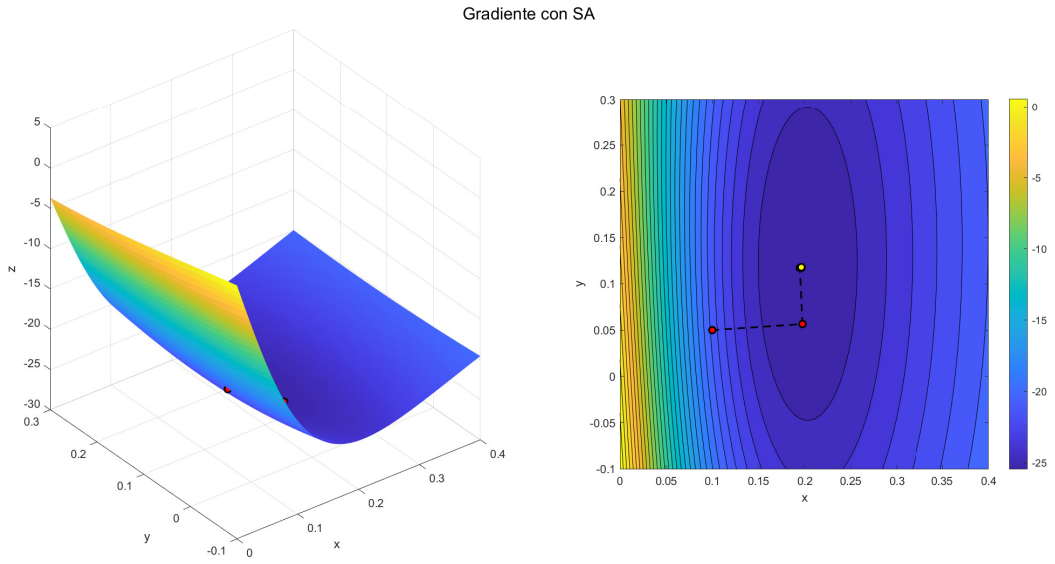


**Figura 13:** Visualizzazione grafica delle iterate per il metodo del gradiente sfruttando il metodo della sezione aurea per $\alpha_k$, partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

Tuttavia, osservando l'output della funzione vediamo che il metodo del gradiente impiega 19 iterazioni per soddisfare il criterio di arresto basato sul gradiente, mentre il metodo del gradiente coniugato FR converge molto più velocemente in sole 6 iterazioni.

```
 Il metodo del gradiente con SA converge in 19 iterazioni
al punto di minimo x = (0.196494,0.117897)^T

 Il metodo del gradiente coniugato (Fletcher-Reeves) con SA converge in 6 iterazioni
al punto di minimo x = (0.196494,0.117897)^T
```

Nell'implementazione del caso `'SA'` nella function `gradiente_coniugato_opt`, i valori di `tol_sa` e `nmax_sa` vengono imposti direttamente all'interno della funzione e sono scelti appositamente per garantire la convergenza del metodo della sezione aurea all'interno di ogni iterata.

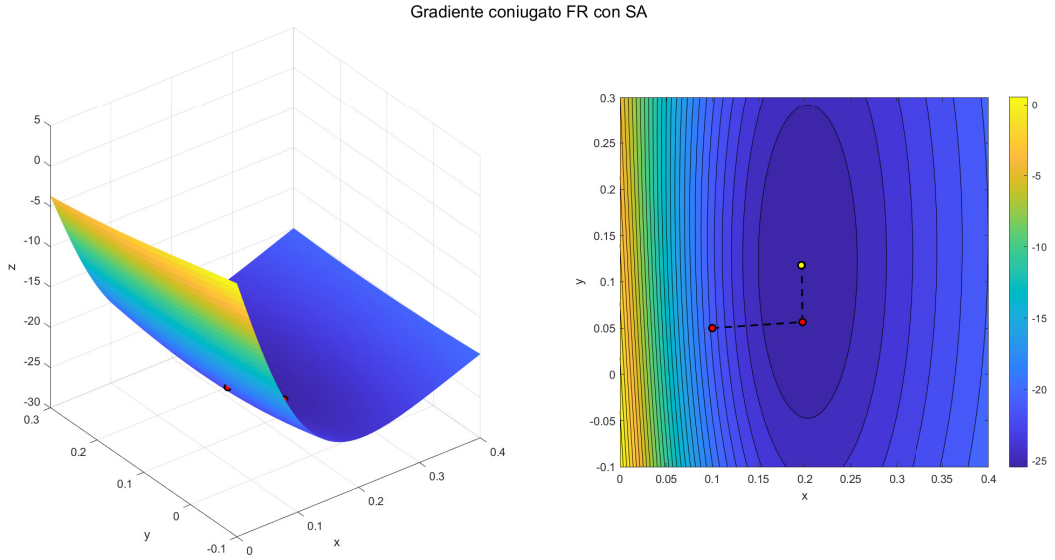


**Figura 14:** Visualizzazione grafica delle iterate per il metodo del gradiente coniugato (FR) sfruttando il metodo della sezione aurea per $\alpha_k$, partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

3. Tramite i seguenti comandi Matlab® plottiamo gli errori contro il numero di iterazioni (in scala semilogaritmica):

```
err_G_SA = zeros(size(xvect_G_SA,2),1);
err_FR_SA = zeros(size(xvect_FR_SA,2),1);


for i = 1:size(xvect_G_SA,2)
    err_G_SA(i) = norm(x_ex - xvect_G_SA(:,i));
end

for i = 1:size(xvect_FR_SA,2)
    err_FR_SA(i) = norm(x_ex - xvect_FR_SA(:,i));
end

figure()
semilogy(err_G_SA,'*-','LineWidth',2.5);
hold on;
grid on;
semilogy(err_FR_SA,'*-','LineWidth',2.5);
legend('Grad SA', 'ConjGrad FR SA');
title('Confronto riduzione errori');
xlabel('#iterazioni');
ylabel('Errori');
title('Metodi con sezione aurea','FontSize',14)
```

E otteniamo gli andamenti riportati in Figura 15.

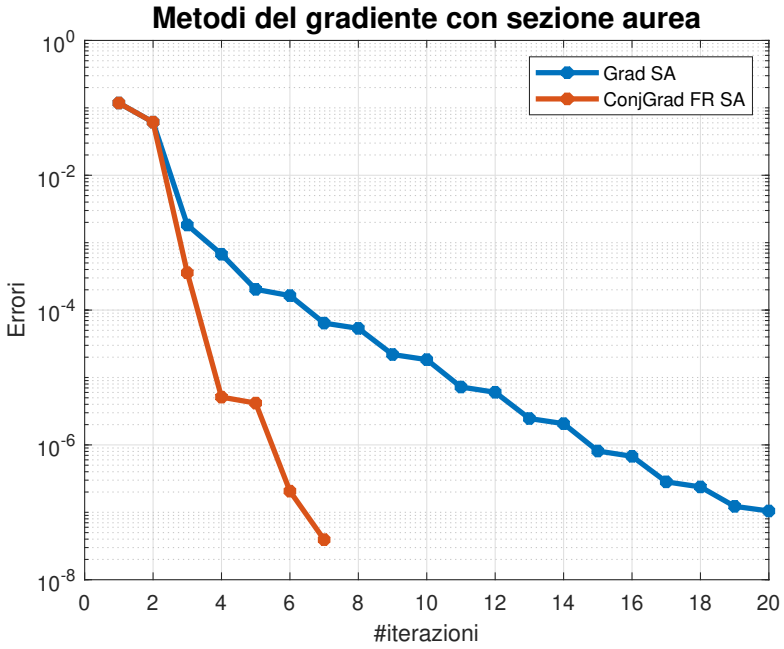


**Figura 15:** Confronto fra la convergenza dei metodi del gradiente e gradiente coniugato, con $\alpha_k$ determinato con sezione aurea partendo dall'iterata iniziale $\mathbf{x_0} = (0.1, 0.05)^T$

### Esercizio 3.2

Tramite i seguenti comandi Matlab® si definisce il funzionale $\Phi$, il suo gradiente e il suo Hessiano.

```
Phi     = @(y1,y2) 3*y1.^2 + y2.^2 - y1/4 - y2/6 + exp(-2*y1.*y2);

GradPhi = @(y1,y2) [6*y1 - 1/4 - 2*y2.*exp(-2*y1.*y2);
                    2*y2 - 1/6 - 2*y1.*exp(-2*y1.*y2)];

HessPhi = @(y1,y2) [6 + 4*(y2.^2).*exp(-2*y1.*y2),...
                    -2*exp(-2*y1.*y2) + 4*y1.*y2.*exp(-2*y1.*y2);
                    -2*exp(-2*y1.*y2) + 4*y1.*y2.*exp(-2*y1.*y2),...
                    2 + 4*(y1.^2).*exp(-2*y1.*y2)];

% Definisco x_ex il punto di minimo locale
x_ex = [0.099299729019640; 0.179161952163217];
```

Tramite i seguenti comandi Matlab® risolviamo il problema di ottimizzazione con i quattro metodi e stimiamo gli ordini di convergenza. In Figura 16 riportiamo i risultati relativi alle stime degli ordini di convergenza.

```
% Condizione iniziale, tolleranza e numero max di iterazioni
x0   = [-0.14; 0.14];
toll = 1e-8;
nmax = 200;

% Newton
```

```
[xv_newton, ~] = newton_opt(GradPhi, HessPhi, x0, toll, nmax);
% BFGS
[xv_bfgs, ~] = bfgs( Phi, GradPhi, x0, toll, nmax );
% Gradiente
[xv_g, ~] = gradiente_coniugato_opt(Phi, GradPhi, 'G', x0, toll, nmax);
% Gradiente Coniugato (FR)
[xv_gc, ~] = gradiente_coniugato_opt(Phi, GradPhi, 'FR', x0, toll, nmax);


% Calcolo errori
err_newton = zeros(size(xv_newton,2),1);
err_bfgs   = zeros(size(xv_bfgs,2),1);
err_g      = zeros(size(xv_g,2),1);
err_gc     = zeros(size(xv_gc,2),1);

for i = 1:size(xv_newton,2)
    err_newton(i) = norm(x_ex - xv_newton(:,i));
end

for i = 1:size(xv_bfgs,2)
    err_bfgs(i) = norm(x_ex - xv_bfgs(:,i));
end

for i = 1:size(xv_g,2)
    err_g(i) = norm(x_ex - xv_g(:,i));
end

for i = 1:size(xv_gc,2)
    err_gc(i) = norm(x_ex - xv_gc(:,i));
end

% Stima ordini di convergenza
conv_ord1_newton = err_newton(2:end) ./ err_newton(1:end-1);
conv_ord2_newton = err_newton(2:end) ./ (err_newton(1:end-1).^2);

conv_ord1_bfgs = err_bfgs(2:end) ./ err_bfgs(1:end-1);
conv_ord2_bfgs = err_bfgs(2:end) ./ (err_bfgs(1:end-1).^2);

conv_ord1_g = err_g(2:end) ./ err_g(1:end-1);
conv_ord2_g = err_g(2:end) ./ (err_g(1:end-1).^2);

conv_ord1_gc = err_gc(2:end) ./ err_gc(1:end-1);
conv_ord2_gc = err_gc(2:end) ./ (err_gc(1:end-1).^2);

figure()
subplot(2,2,1)
plot(conv_ord1_newton,'o-','LineWidth',2);
hold on;
plot(conv_ord2_newton,'o-','LineWidth',2);
legend('ordine 1', 'ordine 2');
xlabel('#iterazioni')
ylabel('\mu')
ylim([0,2]);
title('Newton');
subplot(2,2,2)
plot(conv_ord1_bfgs,'o-','LineWidth',2);
hold on;
```

```
plot(conv_ord2_bfgs,'o-','LineWidth',2);
legend('ordine 1', 'ordine 2');
xlabel('#iterazioni')
ylabel('\mu')
ylim([0,5]);
title('BFGS');
subplot(2,2,3)
plot(conv_ord1_g,'o-','LineWidth',2);
hold on;
plot(conv_ord2_g,'o-','LineWidth',2);
legend('ordine 1', 'ordine 2');
xlabel('#iterazioni')
ylabel('\mu')
ylim([0,2]);
title('Gradiente');
subplot(2,2,4)
plot(conv_ord1_gc,'o-','LineWidth',2);
hold on;
plot(conv_ord2_gc,'o-','LineWidth',2);
legend('ordine 1', 'ordine 2');
xlabel('#iterazioni')
ylabel('\mu')
ylim([-1,3]);
title('Gradiente Coniugato');

Il metodo di Newton converge in 4 iterazioni
al punto di minimo x = (0.099300,0.179162)^T

Il metodo BFGS converge in 29 iterazioni
al punto di minimo x = (0.099300,0.179162)^T

Il metodo del gradiente con BT converge in 54 iterazioni
al punto di minimo x = (0.099300,0.179162)^T

Il metodo del gradiente coniugato (Fletcher-Reeves) con BT converge in 38 iterazioni
al punto di minimo x = (0.099300,0.179162)^T
```

In Figura 16 vediamo il comportamento dei fattori di convergenza rispetto al numero di iterazioni. Nonostante alcune oscillazioni nelle iterate iniziali e in quelle finali, possiamo concludere che il metodo di Newton ha ordine di convergenza pari a 2, mentre il metodo BFGS, il metodo del gradiente e il metodo del gradiente coniugato hanno ordini di convergenza (almeno) pari a 1 in questo caso.

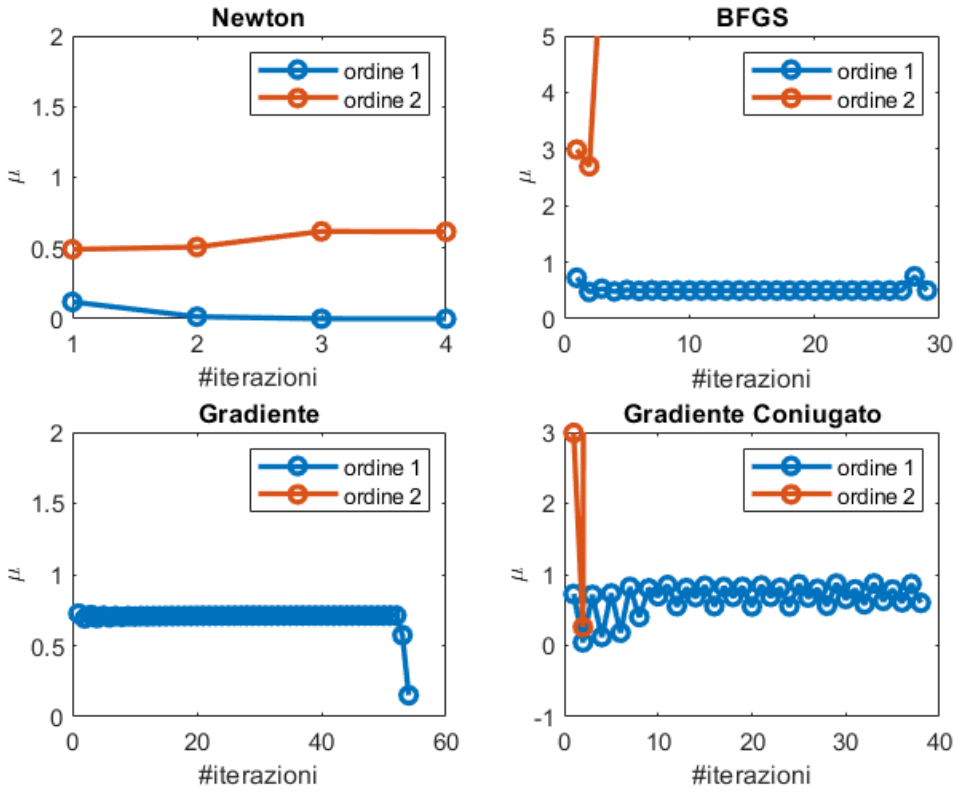


**Figura 16:** Visualizzazione grafica della stima del fattore di convergenza asintotico