Metodi del gradiente discendente e del gradiente coniugato

Corso di Laurea in Ingegneria Elettronica

"Statistica e Ottimizzazione 1" – III anno, III trimestre, a.a. 2002/2003

Mauro Gaggero

Alessandro Di Fusco

Introduzione

Prima di parlare di questi metodi per la ricerca del minimo di una funzione occorre dare la definizione di minimo. Si possono definire due tipi di minimo, minimo locale e minimo globale.

Data una funzione $f : A \subset \Re^n \to \Re$ e un vettore $\mathbf{x} \in A$ definiamo sviluppo in serie di Taylor della funzione *f* attorno al punto $\mathbf{x}_0 \in A$ l'espressione

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0) \cdot (\mathbf{x} - \mathbf{x}_0) + \frac{1}{2}(\mathbf{x} - \mathbf{x}_0) \cdot \mathbf{H}(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0) + R_2(\mathbf{x}, \mathbf{x}_0)$$

Dove $\nabla f(\mathbf{x}_0)$ indica il gradiente della funzione *f* calcolato in $\mathbf{x}_0$ e $\mathbf{H}(\mathbf{x}_0)$ la matrice hessiana della funzione *f* calcolata in $\mathbf{x}_0$.

In modo analogo al caso monodimensionale è possibile dimostrare che $\mathbf{x}_0$ è un punto di minimo locale se il gradiente in quel punto è nullo e l'hessiano è definito positivo.

Nel caso unidimensionale quanto affermato equivale a dire che un punto $\mathbf{x}_0$ è un punto di minimo locale se la derivata prima della funzione in $\mathbf{x}_0$ è nulla e la derivata seconda è positiva.

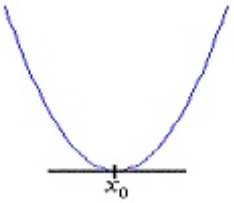


Figura 1

Poiché il calcolo dell'hessiano è molto complesso, soprattutto quando le dimensioni dello spazio in cui si lavora sono molto elevate, si semplifica la definizione precedente definendo il minimo locale come quel punto in cui il gradiente della funzione è uguale a zero. Questa semplificazione porta allo

sviluppo di algoritmi numerici per la ricerca del minimo sicuramente molto più semplici e veloci rispetto al caso in cui si dovesse calcolare anche l'hessiano, ma c'è il rischio che un qualunque punto in cui $\|\nabla f(x_0)\| = 0$ sia scambiato per un minimo.

Un tipico esempio è rappresentato da una sella e dai punti di massimo locale. In tali punti il gradiente della funzione è sì nullo, ma essi non si possono definire certo punti di minimo!
Un esempio in una dimensione potrebbe chiarire le idee:

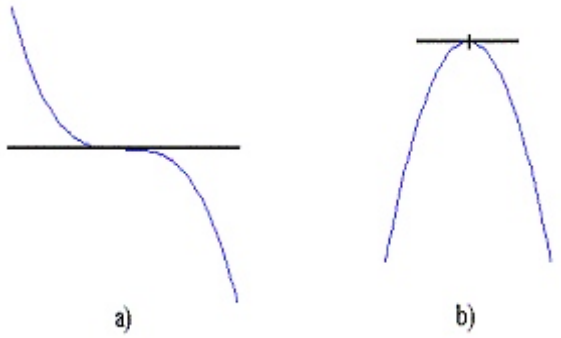


Figura 2

Nell'esempio precedente $\|\nabla f(x_0)\| = 0$ ma x_0 non è un punto di minimo. Si potrebbe scoprire questo andando a calcolare l'hessiano (in questo caso monodimensionale la derivata seconda). Il caso *b)* succede raramente in quanto gli algoritmi sono studiati in modo da andare verso valori di *f* sempre più piccoli (i.e. il metodo del gradiente "discendente") mentre il caso *a)* è molto comune.
Lavorando con i calcolatori per la ricerca del minimo di una funzione occorre definire bene cosa si intende per $\|\nabla f(\mathbf{x}_0)\| = 0$. Il concetto di zero esiste unicamente quando si lavora con gli interi, mentre con valori a virgola mobile a seguito degli errori di approssimazione commessi durante il calcolo lo zero è una pura convenzione. Definiamo zero qualsiasi valore che sia al di sotto di una soglia $\varepsilon > 0$ prefissata, il che vuol dire che un numero qualsiasi che è minore di ε è equiparabile (nel nostro problema) con zero. Questo discorso applicato alla ricerca del minimo significa che viene preso come minimo un punto $\mathbf{x}_0$ tale per cui $\|\nabla f(\mathbf{x}_0)\| \le \varepsilon$.

Per quanto riguarda invece il concetto di minimo globale, un punto $\mathbf{x}^*$ è di minimo globale se $f(\mathbf{x}^*) \le f(\mathbf{x}) \quad \forall \mathbf{x} \in A$. Questo significa che nel punto di minimo globale la funzione *f* assume un valore inferiore al valore che assume in tutti gli altri punti del dominio di definizione.

<u>Metodo del gradiente discendente</u>

Il metodo del gradiente discendente è un algoritmo iterativo per la ricerca del minimo locale di una funzione. Preso un punto iniziale $\mathbf{x}_0$ (la scelta di $\mathbf{x}_0$ avviene casualmente se non si hanno informazioni sulla funzione *f* da minimizzare, in caso contrario secondo qualche criterio comunque indipendente dall'algoritmo stesso) si inizia un ciclo che termina solo quando il gradiente della funzione assume un valore equiparabile con 0.

Detto $\mathbf{x}_k$ il punto ottimo di minimo trovato alla *k*-esima iterazione, andremo a calcolare $\nabla f(\mathbf{x}_k)$. Se $\|\nabla f(\mathbf{x}_k)\| = 0$, $\mathbf{x}_k$ è assunto essere il minimo e l'algoritmo si interrompe. Altrimenti viene calcolata una variazione $\Delta\mathbf{x}_k = -\eta\nabla f(\mathbf{x}_k)$, con η = learning rate o fattore di apprendimento. $\Delta\mathbf{x}_k$ rappresenta lo spostamento compiuto dal punto $\mathbf{x}_k$ per la prossima iterazione. Infatti posto $\mathbf{x} = \mathbf{x}_k + \Delta\mathbf{x}_k$, si definisce un nuovo punto $\mathbf{x}$ e si va a verificare se $f(\mathbf{x}) \le f(\mathbf{x}_k)$.

Scrivere $\Delta\mathbf{x}_k = -\eta\nabla f(\mathbf{x}_k)$ significa muoversi nella direzione opposta rispetto al gradiente di un fattore pari al tasso di apprendimento η.

Il valore di η viene calcolato mediante l'algoritmo di Vogl. η può assumere valori piccoli a piacere, quindi se è sufficientemente piccolo è possibile scrivere l'algoritmo in questo modo:

1) calcolo di $\nabla f(\mathbf{x}_k)$,

2) se $\|\nabla f(\mathbf{x}_k)\| \le \varepsilon \Rightarrow \mathbf{x}_k$ è il minimo, fine ciclo,

3) $\Delta\mathbf{x}_k = -\eta\nabla f(\mathbf{x}_k)$,

4) $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta\mathbf{x}_k$ in quanto certamente $f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k)$ (accettazione implicita del nuovo punto in cui la funzione è sicuramente diminuita),

5) ritorna al punto ***1)***.

Vediamo ora teoricamente la dimostrazione di come con $\eta > 0$ ma molto piccolo $f(\mathbf{x}_{k+1}) \le f(\mathbf{x}_k)$.

Abbiamo:

$$f(\mathbf{x}_{k+1}) = f(\mathbf{x}_k + \Delta\mathbf{x}_k) \cong f(\mathbf{x}_k) + \nabla f(\mathbf{x}_k)^T \Delta\mathbf{x}_k + \mathrm{K}$$

L'espansione può essere interrotta al primo ordine in quanto con η piccolo a piacere i termini di ordine superiore (quali l'hessiano) danno un contributo trascurabile. Andando avanti coi conti abbiamo

$$f(\mathbf{x}_{k+1}) \cong f(\mathbf{x}_k) - \eta \nabla f(\mathbf{x}_k)^T \nabla f(\mathbf{x}_k) = f(\mathbf{x}_k) - \eta \left\| \nabla f(\mathbf{x}_k) \right\|^2 \leq f(\mathbf{x}_k) \text{ in quanto } \eta \left\| \nabla f(\mathbf{x}_k) \right\|^2 \geq 0$$

Nell'algoritmo esposto in precedenza ci sono almeno due punti critici: la scelta del punto iniziale $\mathbf{x}_0$ e la scelta del fattore di qualità ε , ovvero di quella soglia al di sotto della quale tutti i valori sono equiparabili con zero. Per quanto riguarda $\mathbf{x}_0$, come detto in precedenza in assenza di informazioni sulla *f*(**x**), esso viene preso in modo casuale all'interno del dominio di definizione. L'algoritmo esposto può essere migliorato utilizzando la tecnica del "multistart", ovvero se dopo un certo numero di iterazioni non si ottiene nessun miglioramento è possibile ricominciare l'algoritmo con un nuovo punto iniziale $\mathbf{x}_0$. Per quanto riguarda invece la scelta di ε , un ε molto piccolo garantirà maggiore precisione nel risultato, ma implicherà un tempo di convergenza molto maggiore.

Si può notare come questo algoritmo per poter funzionare abbia bisogno dell'informazione analitica della funzione *f* e del suo gradiente. Il gradiente potrebbe essere calcolato anche numericamente facendo uso del rapporto incrementale, ma questa tecnica è sconsigliata per via dell'amplificazione degli errori di arrotondamento nel calcolo di tale rapporto.

Nel programma proposto l'algoritmo viene implementato attraverso la funzione `gradiente()` il cui prototipo è il seguente:

```
double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim), double *xIniziale, double *xmin, int dim, double eps, double etaIniziale, int maxiter);
```

I primi due parametri sono dei puntatori a funzione, rispettivamente alla funzione costo da minimizzare e alla norma del gradiente (il passaggio tramite puntatore permette di inserire la funzione in una libreria e chiamarla con qualunque espressione di *f* e ∇f). `*xIniziale` è l'array contenente il punto $\mathbf{x}_0$ da cui iniziare l'algoritmo, `*xmin` è l'array in cui al termine dell'elaborazione risiederà il minimo trovato. Questi array hanno dimensione pari a `dim`. Infine `eps` è un parametro indicante il fattore di qualità, `etaIniziale` è il valore iniziale del learning rate e `maxiter` un intero indicante il numero massimo di iterazioni consentite al gradiente. Questo parametro potrebbe essere del tutto eliminato, è stato inserito per evitare che il programma giri all'infinito senza trovare un punto di minimo. Se dopo il massimo numero di iterazioni consentito il programma non ha ancora trovato un punto di minimo soddisfacente esso viene interrotto e viene

restituito il valore ottimo trovato sino a quel momento. Il valore restituito dalla funzione `gradiente()` è il valore della funzione costo nel punto di minimo trovato.

Nell'espressione di `f` e `gf` , `*x` è l'array dei punti in cui si vuole calcolare `f` o `gf` di dimensioni pari a `dim`.

Ecco la funzione gradiente completa:

```
/* Funzione per il calcolo del minimo attraverso l'algoritmo del gradiente discendente.
 * Parametri:  -funzione f da minimizzare,
 *             -gradiente della funzione da minimizzare,
 *             -punto iniziale per l'algoritmo,
 *             -array contenente il minimo (passaggio per riferimento, qui ci sarà il
 *              risultato finale),
 *             -numero di variabili (dimensione vettori xIniziale e xmin),
 *             -fattore di qualità: valore al di sotto del quale il gradiente può essere
 *              considerato nullo,
 *             -il valore iniziale del learning rate
 *             -numero massimo di iterazioni concesse. Terminate le iterazioni si ritorna
 *              il valore ottimo sino a quel punto (questo per evitare che il programma
 *              entri in un ciclo infinito senza fornire alcun risultato).
 * Restituisce il valore del minimo.
 *
 * Questo algoritmo NON è multistart e accetta funzioni definite per qualsiasi valore di x.
 * In altre parole non effettua alcun controllo sel range di variazione del punto di minimo
 * durante il calcolo iterativo. Per un algoritmo multistart e di ricerca di minimi entro un
 * ben preciso intervallo si veda il file "gradienteMultistart.cpp"
 */
double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, int dim, double eps, double etaIniziale, int maxiter)
{
	double eta=etaIniziale;

	/*valore minimo della funzione f*/
	double minimo=f(xIniziale, dim);

	/*allocazione dinamica del vettore contenente i minimi di prova, ovvero
	 *quei punti che diventeranno minimi se il valore di f è minore del valore
	 *più piccolo fino a quel momento. E' allocato anche il vettore del gradiente
	 */
	double *xminp=new double[dim];
	double *grad=new double[dim];
	if(!xminp || !grad)
	{
		printf("Errore nell'allocazione dinamica della memoria!\n");
		exit(1);
	}

	/*inizializzazione del minimo al punto iniziale*/
	for(int i=0; i<dim; i++)
	{
		xmin[i]=xIniziale[i];
		xminp[i]=0;
		grad[i]=0;
	}

	/*ciclo principale*/
	for(i=0; i<maxiter; i++)	//iter
	{
		gf(xmin, grad, dim);
		double normaGrad=0;
		for(int j=0; j<dim; j++)
		{
			normaGrad+=grad[j]*grad[j];
		}
		if(normaGrad<0)
		{
			printf("Errore!! Norma del gradiente < 0\n");
			exit(1);
		}
		normaGrad=sqrt(normaGrad);

		/*se il gradiente è zero (ovvero minore di eps) l'algoritmo è finito*/
		if(normaGrad<=eps)	break;
```

```
		/*determina uno spostamento dal minimo trovato sino ad ora*/
		for(j=0; j<dim; j++)
		{
			gf(xmin, grad, dim);
			double deltax = -eta * grad[j];
			xminp[j] = xmin[j] + deltax;
		}

		/*inizio calcolo di eta*/
		double g1=1.05;
		double g2=1.05;
		double g3=0.7;

		double fxminp=f(xminp, dim);

		if( fxminp < minimo )
		{
			eta=eta*g1;	/*ho avuto successo, eta viene aumentato*/
			minimo=fxminp;
			/*accettazione*/
			for(j=0; j<dim; j++)
			{
				xmin[j]=xminp[j];
			}
		}
		else
		{
			if( fxminp <= g2*minimo )
			{
				minimo=fxminp;
				/*accettazione*/
				for(j=0; j<dim; j++)
				{
					xmin[j]=xminp[j];
				}
				eta=eta*g3;	/*eta viene ridotto del 30%*/
			}
			else
			{
				/*rigetto*/
				eta=eta*g3;	/*se non sono entro il 5% riduco eta senza
accettazione*/
			}
		}
		/*stampa valori intermedi*/
		printf("%d\t%lf\t%lf\t%lf\n", i, normaGrad, minimo, eta);
	}	//fine iter

	/*rilascio memoria*/
	delete[] xminp;
	delete[] grad;

	/*se sono arrivato a questo punto ho esaurito il numero massimo di iterazioni,
	 *restituisco il miglior valore trovato sino ad ora.
	 */
	if(i==maxiter) printf("Raggiunto limite massimo di iterazioni\n");
	return minimo;
}
```

Effettuando diversi test con due funzioni costo abbastanza semplici quali $f(x) = x^2$ e $f(\mathbf{x}) = x_1^2 + x_2^2$ si ottengono risultati corretti.

In particolare nel primo caso con il programma

```
/* Gradiente.cpp
 *
 * Questo programma calcola il minimo di una funzione attraverso il metodo
 * del gradiente discendente. Occorre fornire l'espressione analitica sia
 * della funzione da minimizzare che del suo gradiente, l'algoritmo non
 * è multistart e non è possibile ricercare minimi vincolati.
 *
 * by Mauro Gaggero & Alessandro Di Fusco
 * mauro.gaggero1@tin.it
```

```
 * aledifu@libero.it
 *
 * 30/03/2003
 */


#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*Numero massimo di iterazioni concesso al gradiente. Terminate le iterazioni
 *restituisce il minimo valore trovato sino a quel momento
 */
#define MAXITER 10000

/*intestazione funzione di ricerca del minimo*/
double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, int dim, double eps, double etaIniziale, int maxiter);

double f(double*, int);
void gf(double*, double*, int);


int main()
{
        int dim=1;
        double xIniziale[1]={10};
        double xmin[1];
        double eps=1e-6;
        double etaIniziale=0.1;

        double minimo=gradiente(f, gf, xIniziale, xmin, dim, eps, etaIniziale, MAXITER);

        printf("Punto iniziale = %lf\n", xIniziale[0]);
        printf("Minimo = %lf\n", minimo);
        printf("ottenuto per x1 = %lf\n", xmin[0]);

        return 0;
}

//funzione gradiente (come prima)

/* Funzione di cui calcolare il minimo. Vengono passati l'array
 * dei parametri e la dimensione dell'array (numero di variabili).
 * Restituisce il valore della funzione nel punto x.
 */
double f(double *x, int dim)
{
        return x[0]*x[0];
}


/* Gradiente della funzione di cui calcolare il minimo. Vengono passati
 * l'array dei parametri e quello del gradiente in cui sarà salvato il
 * risultato. Gli array hanno dimensione pari a dim(numero di variabili).
 */
void gf(double *x, double *grad, int dim)
{
        grad[0]=2*x[0];
}
```

Si ottiene (i valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta)):

```
0       20.000000       100.000000      0.700000
1       20.000000       16.000000       0.735000
2       8.000000        3.534400        0.771750
3       3.760000        1.044034        0.810338
4       2.043560        0.402201        0.850854
5       1.268387        0.198042        0.893397
.....................
39      0.000005        0.000000        0.927077
40      0.000004        0.000000        0.973431
```

```
41      0.000003        0.000000        1.022102
42      0.000003        0.000000        0.715471
43      0.000003        0.000000        0.751245
44      0.000001        0.000000        0.788807
Punto iniziale = 10.000000
Minimo = 0.000000
ottenuto per x1 = -0.000000
```

Nel secondo caso ($f(\mathbf{x}) = x_1^2 + x_2^2$) con il programma

```
/* Gradiente.cpp
 *
 * Questo programma calcola il minimo di una funzione attraverso il metodo
 * del gradiente discendente. Occorre fornire l'espressione analitica sia
 * della funzione da minimizzare che del suo gradiente, l'algoritmo non
 * è multistart e non è possibile ricercare minimi vincolati.
 *
 * by Mauro Gaggero & Alessandro Di Fusco
 * mauro.gaggero1@tin.it
 * aledifu@libero.it
 *
 * 30/03/2003
 */


#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

/*Numero massimo di iterazioni concesso al gradiente. Terminate le iterazioni
 *restituisce il minimo valore trovato sino a quel momento
 */
#define MAXITER 10000

/*intestazione funzione di ricerca del minimo*/
double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, int dim, double eps, double etaIniziale, int maxiter);

double f(double*, int);
void gf(double*, double*, int);


int main()
{
        int dim=2;
        double xIniziale[2]={50, 20};
        double xmin[2];
        double eps=1e-6;
        double etaIniziale=0.1;

        double minimo=gradiente(f, gf, xIniziale, xmin, dim, eps, etaIniziale, MAXITER);

        printf("Punto iniziale = [%lf, %lf]\n", xIniziale[0], xIniziale[1]);
        printf("Minimo = %lf\n", minimo);
        printf("ottenuto per x1 = %lf      x2 = %lf\n", xmin[0], xmin[1]);

        return 0;
}

//funzione gradiente (come prima)

/* Funzione di cui calcolare il minimo. Vengono passati l'array
 * dei parametri e la dimensione dell'array (numero di variabili).
 * Restituisce il valore della funzione nel punto x.
 */
double f(double *x, int dim)
{
        return x[0]*x[0]+x[1]*x[1];
}


/* Gradiente della funzione di cui calcolare il minimo. Vengono passati
 * l'array dei parametri e quello del gradiente in cui sarà salvato il
 * risultato. Gli array hanno dimensione pari a dim(numero di variabili).
 */
```

```
void gf(double *x, double *grad, int dim)
{
        grad[0]=2*x[0];
        grad[1]=2*x[1];
}
```

Si ottiene (i valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta)):

```
0       107.703296      1856.000000     0.105000
1       86.162637       1158.329600     0.110250
2       68.068483       703.824521      0.115763
3       53.059383       415.646264      0.121551
4       40.774809       238.121964      0.127628
.....................
25      0.000339        0.000000        0.355567
26      0.000109        0.000000        0.373346
27      0.000032        0.000000        0.392013
28      0.000008        0.000000        0.411614
29      0.000002        0.000000        0.432194
Punto iniziale = [50.000000, 20.000000]
Minimo = 0.000000
ottenuto per x1 = 0.000000      x2 = 0.000000
```

I risultati coincidono con quelli attesi teoricamente in quanto le funzioni costo sono molto semplici, con il minimo come il solo punto a gradiente nullo. Da qualsiasi punto iniziale $\mathbf{x}_0$ l'algoritmo si muoverà in discesa verso il minimo.

Esistono però dei casi in cui l'algoritmo non funziona. Consideriamo ad esempio la funzione $f(x) = x^3$, il cui andamento è

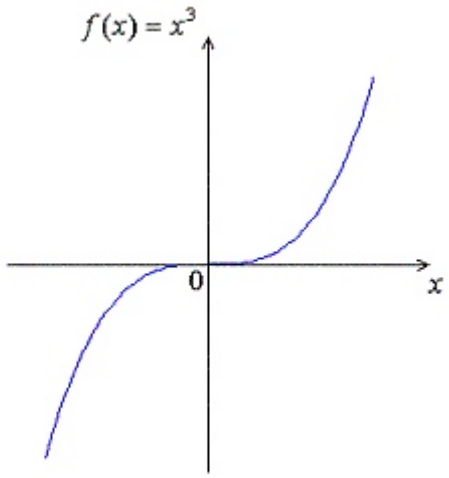


Figura 3

Il punto $x = 0$ è un punto a gradiente nullo.

Scegliendo un punto iniziale $x_0 > 0$, l'algoritmo scende fino a fermarsi in prossimità di $x = 0$, che viene spacciato come minimo. Evidentemente il risultato è errato. Se invece si prende un punto iniziale $x_0 < 0$, viene raggiunta rapidamente una condizione di overflow in quanto l'algoritmo tende

a seguire la discesa di $f(x)$ fino a superare il più grande in modulo numero negativo rappresentabile, causando un overflow. Questo è vero se `etaIniziale` è sufficientemente piccolo (i.e. 0.001), mentre se è grande il flesso viene saltato e l'overflow raggiunto in ogni caso, anche con un punto iniziale $x_0 > 0$. Con questo semplice esempio vengono mostrati due grandi limiti di questo metodo: *a)* considerare qualunque punto a gradiente nullo come minimo e *b)* a volte dare risultati completamente diversi a seconda del punto iniziale. Il programma precedente con $f(x) = x^3$ e $x_0 < 0$ fornisce il seguente risultato come questo: i valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta):

```
0       3.000000        -1.009027       0.001050
1       3.018027        -1.018621       0.001103
2       3.037128        -1.028825       0.001158
3       3.057376        -1.039684       0.001216
……….
60      1482.559556     -124425.720625  0.019613
……….
69      1.#INF00        -1.#INF00       0.030426
70      1.#INF00        -1.#INF00       0.031948
71      1.#INF00        -1.#INF00       0.022363
72      1.#INF00        -1.#INF00       0.015654
……..
9997    1.#INF00        -1.#INF00       0.000000
9998    1.#INF00        -1.#INF00       0.000000
9999    1.#INF00        -1.#INF00       0.000000
Punto iniziale = -1.000000
Minimo = -1.#INF00
ottenuto per x1 = -1.#INF00
```

Fino ad ora abbiamo ricercato minimi non vincolati: potrebbe accadere però che la funzione costo sia definita solo in un certo intervallo di **x**. Il problema diventa allora un problema di minimo vincolato, e il programma di prima deve essere modificato in modo da garantire che il punto di minimo stia sempre entro i limiti imposti. La funzione `gradiente()` cambia allora in questo modo:

```
/* Funzione per il calcolo del minimo attraverso l'algoritmo del gradiente discendente.
 * Parametri:  -funzione f da minimizzare,
 *             -gradiente della funzione da minimizzare,
 *             -punto iniziale per l'algoritmo,
 *             -array contenente il minimo (passaggio per riferimento, qui ci sarà il
 *              risultato finale),
 *             -limite inferiore del dominio di definizione di f,
 *             -limite superiore del dominio di definizione di f,
 *             -numero di variabili (dimensione vettori xIniziale, xmin, sx e dx),
 *             -fattore di qualità: valore al di sotto del quale il gradiente può essere
 *              considerato nullo,
 *             -il valore iniziale del learning rate
 *             -massimo numero di iterazioni concesse proma di effettuare un nuovo start
 *              con un nuovo punto iniziale,
 *             -numero massimo di iterazioni concesse. Terminate le iterazioni si ritorna
 *              il valore ottimo sino a quel punto (questo per evitare che il programma
 *              entri in un ciclo infinito senza fornire alcun risultato).
 * Restituisce il valore del minimo.
 *
 * Questo algoritmo è multistart e accetta funzioni definite per un certo intervallo di x.
 * In altre parole se durante la ricerca del minimo il punto finisce fuori del dominio di
```

```
 * definizione di f, esso viene bloccato all'estremo del dominio di definizione stesso.
 * Se tutte le componenti sono state bloccate agli estremi l'algoritmo termina e restituisce
 * il risultato (infatti ormai è impossibile spostarsi da quel punto).
 * Per quanto riguarda il multistart, se dopo maxstart iterazioni il valore del minimo non è
 * cambiato, si procede a definire un nuovo punto iniziale scelto casualmente all'interno del
 * dominio di definizione di f e l'algoritmo ricomincia.
 */
double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, double *sx, double *dx, int dim, double eps, double etaIniziale,
int maxstart, int maxiter)
{
	long int SEED=1;	/*per il generatore di numeri casuali*/
	double eta=etaIniziale;

	/*contatore del numero di volte in cui non si ha alcun miglioramento del
	 *valore del minimo. Raggiunto il numero massimo si riparte da un altro
	 *punto iniziale.
	 */
	int start=0;

	/*valore minimo della funzione f*/
	double minimo=f(xIniziale, dim);

	/*allocazione dinamica del vettore contenente i minimi di prova, ovvero
	 *quei punti che diventeranno minimi se il valore di f è minore del valore
	 *più piccolo fino a quel momento. E' allocato anche il vettore del gradiente
	 */
	double *xminp=new double[dim];
	double *grad=new double[dim];
	if(!xminp || !grad)
	{
		printf("Errore nell'allocazione dinamica della memoria!\n");
		exit(1);
	}

	/*inizializzazione del minimo al punto iniziale*/
	for(int i=0; i<dim; i++)
	{
		xmin[i]=xIniziale[i];
		xminp[i]=0;
		grad[i]=0;
		/*controlla che il punto iniziale sia all'interno del dominio*/
		if(xmin[i]<sx[i] || xmin[i]>dx[i])
		{
			printf("Errore!! Il punto iniziale fornito e' fuori dal dominio di definizione
\n");
			exit(1);
		}
	}

	/*ciclo principale*/
	for(i=0; i<maxiter; i++)	//iter
	{
		gf(xmin, grad, dim);
		double normaGrad=0;
		for(int j=0; j<dim; j++)
		{
			normaGrad+=grad[j]*grad[j];
		}
		if(normaGrad<0)
		{
			printf("Errore!! Norma del gradiente < 0\n");
			exit(1);
		}
		normaGrad=sqrt(normaGrad);

		/*se il gradiente è zero (ovvero minore di eps) l'algoritmo è finito*/
		if(normaGrad<=eps)	break;


		/*se sono con una componente all'estremo sinistro con gradiente positivo
		 *non mi muoverò più, così come con una componente all'estremo destro e
		 *gradiente negativo. Tali componenti sono bloccate, cos' come una
		 *componente entro il dominio di definizione ma con componente del gradiente
		 *nulla. Se il numero di queste condizioni è pari alla dimensione delle
		 *incognite tutte le componenti sono bloccate, esco dal ciclo.
		 */
		int count=0, estremo=0;
		for(j=0; j<dim; j++)
```

```
{
        if( xmin[j]==sx[j] && grad[j]>=0.)
        {
                count++;
                estremo++;
        }
        if( xmin[j]==dx[j] && grad[j]<=0.)
        {
                count++;
                estremo++;
        }
        if( xmin[j]>=sx[j] && xmin[j]<=dx[j] && fabs(grad[j])<=eps)       count++;
}
if(count==dim && estremo>0)
{
        printf("Raggiunto estremo del dominio di definizione\n");
        break;
}


/*determina uno spostamento dal minimo trovato sino ad ora*/
for(j=0; j<dim; j++)
{
        gf(xmin, grad, dim);
        double deltax = -eta * grad[j];
        xminp[j] = xmin[j] + deltax;

        /*controlla di rimanere nel dominio corretto*/
        if(xminp[j]<sx[j])
        {
                xminp[j]=sx[j];
        }
        else if(xminp[j]>dx[j])
        {
                xminp[j]=dx[j];
        }
        else;
}


/*inizio calcolo di eta*/
double g1=1.05;
double g2=1.05;
double g3=0.7;

double fxminp=f(xminp, dim);

/*controllo per il multistart*/
if( fxminp == minimo )
{
        /*incrementa il contatore del multistart*/
        start++;
        if(start==maxstart)
        {
                /*si riparte da un nuovo punto iniziale (casuale entro il dominio)*/
                printf("restart alla prossima iterazione\n");
                for(j=0; j<dim; j++)
                {
                        xmin[j]=sx[j]+(dx[j]-sx[j])*randnum(SEED);
                }
                eta=etaIniziale;
                start=0;
        }
}

if( fxminp < minimo )
{
        eta=eta*g1;    /*ho avuto successo, eta viene aumentato*/
        minimo=fxminp;
        /*accettazione*/
        for(j=0; j<dim; j++)
        {
                xmin[j]=xminp[j];
        }
}
else
{
        if( fxminp <= g2*minimo )
        {
```

```
                                minimo=fxminp;
                                /*accettazione*/
                                for(j=0; j<dim; j++)
                                {
                                        xmin[j]=xminp[j];
                                }
                                eta=eta*g3;    /*eta viene ridotto del 30%*/
                        }
                        else
                        {
                                /*rigetto*/
                                eta=eta*g3;    /*se non sono entro il 5% riduco eta senza
accettazione*/
                        }
                }

                /*stampa valori intermedi*/
                printf("%d\t%lf\t%lf\t%lf\t%d\n", i, normaGrad, minimo, eta, start);
        }       //fine iter

        /*rilascio memoria*/
        delete[] xminp;
        delete[] grad;

        /*se sono arrivato a questo punto ho esaurito il numero massimo di iterazioni,
         *restituisco il miglior valore trovato sino ad ora.
         */
        if(i==maxiter)  printf("Raggiunto limite massimo di iterazioni\n");
        return minimo;
}


/*Generatore di numeri pseudo-casuali*/
double randnum(long int &SEED)
{
        long int aa, mm, qq, rr, hh, lo, test;
        double reslt;

        aa = 16807;
        mm = 2147483647;
        qq = 127773;
        rr = 2836;
        hh = (long int)(SEED/qq);
        lo = SEED - hh*qq;
        test=aa*lo-rr*hh;
        if(test>=0)
                SEED=test;
        else
                SEED=test+mm;
        reslt=SEED/(double)mm;
        return reslt;
}
```

Viene aggiunta l'informazione dei limiti superiore e inferiore del dominio di definizione di *f* (attraverso i vettori `*sx` e `*dx`) e il multistart. Se dopo `maxstart` iterazioni il punto di minimo non migliora si riparte da un altro $\mathbf{x}_0$ scelto casualmente entro i limiti fissati. Mentre la funzione `gradiente()` originaria non adottava la tecnica del multistart, questa versione ne fa uso in quanto con l'informazione dei limiti del dominio di definizione di *f* è possibile scegliere casualmente un nuovo punto iniziale all'interno di questo dominio. Nell'implementazione precedente senza limiti del dominio si sarebbe dovuto scegliere casualmente un punto iniziale variabile da $-\infty$ a $+\infty$. In questa versione, se durante la ricerca del minimo si finisce fuori del dominio di definizione, il nuovo punto viene automaticamente bloccato all'estremo relativo. Se l'algoritmo durante la sua

discesa si ferma in un estremo del dominio, questo viene restituito come se fosse il minimo (anche se il gradiente in tale punto non è nullo), in quanto ormai non si troverà più un punto "migliore".

Il problema della funzione $f(x) = x^3$ viene affrontato in questo modo: con limiti ad esempio $-100 \le x \le +100$ e punto iniziale $x_0 = -10$, il minimo viene trovato in prossimità di $x = -100$, ovvero all'estremo inferiore del dominio (notare che questo minimo non ha gradiente nullo). I valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta / start):

```
0       0.030000        -0.001093       0.105000        0
1       0.031827        -0.001203       0.110250        0
2       0.033926        -0.001334       0.115763        0
3       0.036354        -0.001493       0.121551        0
.....................
19      0.973090        -0.542189       0.265330        0
20      1.994749        -2.431459       0.278596        0
21      5.424577        -23.294596      0.292526        0
22      24.469479       -1004.181183    0.307152        0
Raggiunto estremo del dominio di definizione
Punto iniziale = -0.100000
Minimo = -1000000.000000
ottenuto per x1 = -100.000000
```

Se invece $x_0 > 0$ il risultato è lo stesso di prima, con l'algoritmo che si ferma su $x = 0$.

L'algoritmo del gradiente nel caso di variabili digitali

Consideriamo una funzione $f(x_1, x_2 \mathrm{K}\ x_n)$ con $x_1, x_2 \mathrm{K}\ x_n$ variabili digitali, ovvero $x_1, x_2 \mathrm{K}\ x_n \in \{0,1\}$. Prima di spiegare l'algoritmo del gradiente in questo caso occorre definire cosa voglia dire minimo locale in ambito digitale. Data una sequenza di 0 e 1, questa è un punto di minimo se la funzione calcolata nelle sequenze ottenute invertendo un bit alla volta della sequenza di partenza assume valori maggiori rispetto a quello assunto nel presunto minimo. Ad esempio se $n = 3$, supponiamo che il punto di minimo sia 001. Questa sequenza rappresenta effettivamente il minimo se le sequenze 101, 011 e 000 producono valori di *f* maggiori rispetto a $f(001)$. Data una sequenza di *n* bit, ci saranno *n* bit da invertire per effettuare la verifica del minimo. Si potrebbe osservare che così facendo alcune combinazioni (per esempio 111 nell'esempio precedente) non vengono controllate. Questo è sicuramente vero, ma esplorare tutte le possibili combinazioni diventerebbe estremamente complesso da un punto di vista computazionale non appena il numero *n* cresca.

Sfruttando la definizione di minimo data è possibile realizzare l'algoritmo del gradiente in questo modo: data una sequenza iniziale si inizia ad invertire il primo bit. Se la funzione è aumentata

rispetto a prima il bit viene riportato al valore originario, mentre se la funzione è diminuita l'inversione del bit è accettata. In ogni caso si procede all'inversione del 2° bit e si va nuovamente a verificare il valore della funzione, in modo da decidere se accettare o meno questa inversione. Tutto questo procedimento di ripete finchè si sono esauriti tutti i bit della sequenza di partenza.

La funzione qui sotto riportata implementa questo algoritmo nel caso di sequenze lunghe al più quanto un `unsigned int` (nel nostro caso 32 bit). L'inversione bit per bit viene fatta utilizzando l'operatore XOR bit a bit tra la sequenza originaria e una "maschera" composta da tutti zeri tranne un 1 nella posizione del bit da invertire.

Il listato del programma è il seguente:

```
/* Funzione per il calcolo del minimo di una funzione digitale attraverso l'algoritmo del
 * gradiente discendente.
 * Parametri:  -funzione f da minimizzare,
 *             -punto iniziale per l'algoritmo,
 *             -valore per cui si hail  minimo (passaggio per riferimento, qui ci sarà il
 *              risultato finale)
 * Restituisce il valore del minimo.
 *
 * Tutti i parametri sono unsigned int, il che vuol dire che si possono trattare funzioni digitali
 * aventi un numero di incognite pari al massimo alla lunghezza di un unsigned int (32 nella
 * configurazione macchina+compilatore corrente). I bit del punto iniziale fornito vengono invertiti
 * uno ad uno: se il valore della funzione decresce viene accettato il nuovo punto e si procede
 * all'inversione del bit successivo, altrimenti lo switch viene rifiutato e si procede
 * all'inversione del prossimo bit.
 */
double gradiente(double (*f)(unsigned int x), unsigned int xIniziale, unsigned int &xmin)
{
        /*valore minimo della funzione f*/
        double minimo=f(xIniziale);

        /*minimo "di prova" (sarà accettato solo se f(xminp)<f(xmin) )*/
        unsigned int xminp;

        xmin=xIniziale;

        /*inizializzazione della maschera per switchare i bit uno alla volta.
         *per invertire un bit di una sequenza basta fare l'exor con 0..010..0
         *dove l'1 è nella posizione da invertire. Inizialmente l'1 sarà nella
         *prima posizione, poi man mano verrà spostato verso destra tramite
         *shiftamenti.
         */
        unsigned int maschera=0x80000000;    /* 100...00 */

        /*32 è la dimensione di un unsigned int su questa macchina*/
        for(int i=0; i<8*sizeof(unsigned int); i++)
        {
                xminp=xmin^maschera;  /*xor bit a bit*/

                double tmp=f(xminp);
                if(tmp<=minimo)  /*accetto il nuovo punto*/
                {
                        minimo=tmp;
                        xmin=xminp;
                }

                printf("%2d\t0x%8X\t0x%8X\t0x%8X\t%lf\t%lf\n",  i,  maschera,  xmin,  xminp,  minimo,
tmp);

                /*sposto l'1 a destra di uno*/
                maschera = maschera >> 1;
        }

        return minimo;
}
```

Per mostrare un esempio di funzionamento consideriamo il seguente codice:

```
/* GradienteDigitale.cpp
 *
 * Questo programma calcola il minimo di una funzione digitale attraverso
 * il metodo del gradiente. Nel metodo del gradiente "digitale" si procede
 * all'inversione bit per bit del puno iniziale fornito, inversione accettata
 * se il valore della funzione via via decresce. Per semplicità ci si è limitati
 * al caso in cui la funzione può trattare solo 32 variabili digitali.
 *
 *
 * by Mauro Gaggero & Alessandro Di Fusco
 * mauro.gaggero1@tin.it
 * aledifu@libero.it
 *
 * 30/03/2003
 */


#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


/*intestazione funzione di ricerca del minimo*/
double gradiente(double (*f)(unsigned int x), unsigned int xIniziale, unsigned int &xmin);

double f(unsigned int x);


int main()
{
        unsigned int xIniziale=0x00000000;
        unsigned int xmin;

        double minimo=gradiente(f, xIniziale, xmin);

        printf("Minimo = %lf\n", minimo);
        printf("ottenuto per x1 = 0x%8X\n", xmin);

        return 0;
}

//funzione gradienteDigitale (come prima)


/*Funzione di cui calcolare il minimo*/
double f(unsigned int x)
{
        switch(x)
        {
                case 0x00000000:      //0000...0
                        return 0;
                case 0x80000000:      //1000...0
                        return 1;
                case 0x40000000:      //0100...0
                        return -2;
                case 0x20000000:      //0010...0
                        return 3;
                default:
                        return 4;
        }
}
```

Che produce un risultato come questo: i valori visualizzati durante le iterazioni sono (iterazione / maschera / xmin / xminprova / f(xmin) / f(xminprova)):

```
 0      0x80000000     0x        0     0x80000000      0.000000       1.000000
 1      0x40000000     0x40000000     0x40000000     -2.000000       -2.000000
 2      0x20000000     0x40000000     0x60000000     -2.000000       4.000000
.....
29      0x     4       0x40000000     0x40000004     -2.000000       4.000000
30      0x     2       0x40000000     0x40000002     -2.000000       4.000000
31      0x     1       0x40000000     0x40000001     -2.000000       4.000000
```

```
Minimo = -2.000000
ottenuto per x1 = 0x40000000
```

Si nota come vengano controllate solo alcune combinazioni, quindi non è garantito che nel punto trovato la funzione assuma effettivamente il valore minimo. Come detto in precedenza occorrerebbe controllare tutte le combinazioni, al prezzo però di un maggior costo computazionale dell'algoritmo. Questo algoritmo data la lunghezza finita della sequenza non è multistart né permette la ricerca di massimi e minimi entro un determinato intervallo.

L'algoritmo del gradiente coniugato

L'algoritmo del gradiente coniugato è l'algoritmo più robusto e più utilizzato per la ricerca del minimo di una funzione. Questo metodo, come gli altri metodi detti della "direzione coniugata" fu inventato per risolvere il problema riguardante la forma quadratica

$$\min_{\mathbf{x}} \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{b}^T\mathbf{x}$$

dove $\mathbf{Q}$ è una matrice *n*x*n* simmetrica e definita positiva. Risolvere questo problema è equivalente a risolvere il didtema di equazioni $\mathbf{Qx} + \mathbf{b} = \mathbf{0}$, essendo il gradiente della forma quadratica la quantità $\mathbf{g} = \mathbf{Qx} + \mathbf{b}$. Gli algoritmi possono poi essere estesi a problemi più generali in cui non si debba minimizzare necessariamente una forma quadratica, ma qualsiasi altra funzione.

L' algoritmo si compone dei seguenti passi fondamentali: dato un punto iniziale $\mathbf{x}_0 \in \Re^n$, definiamo

$\mathbf{d}_0 = -\mathbf{g}_0 = \mathbf{Q}\mathbf{x}_0 + \mathbf{b}_0$ e

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{d}_k$$

$$\alpha_k = \frac{-\mathbf{g}_k^T \mathbf{d}_k}{\mathbf{d}_k^T \mathbf{Q} \mathbf{d}_k}$$

$$\mathbf{d}_{k+1} = -\mathbf{g}_{k+1} + \beta_k \mathbf{d}_k$$

$$\beta_k = \frac{\mathbf{g}_{k+1}^T \mathbf{g}_{k+1}}{\mathbf{g}_k^T \mathbf{g}_k}$$

dove *k* indica l'iterazione corrente.

Il primo passo è identico al metodo del gradiente discendente. In tutti i passi successivi l'algoritmo si muove in una direzione che è una combinazione lineare del gradiente corrente e del vettore

direzione **d** precedente. Il metodo del gradiente coniugato ritorna ad essere quello del gradiente discendente per $\beta_k = 0 \quad \forall k$.

La funzione `gradienteConiugato()` qui sotto riportata implementa l'algoritmo esposto precedentemente:

```
/* Funzione per il calcolo del minimo attraverso l'algoritmo del gradiente coniugato.
 * L'algoritmo può essere applicato al solo caso in cui la funzione da minimizzare sia
 * una forma quadratica 0.5x*Qx+b*x, e quindi la ricerca del minimo sia equivalente
 * alla soluzione di un sistema lineare di equazioni.
 * Parametri:  -matrice q della forma quadratica, di dimensioni DIMxDIM,
 *             -vettore b della forma quadratica,
 *             -punto iniziale per l'algoritmo,
 *             -array contenente il minimo (passaggio per riferimento, qui ci sarà il
 *              risultato finale),
 *             -numero di variabili (dimensione vettori xIniziale e xmin) (la dimensione
 *              deve essere pari a DIM),
 *             -fattore di qualità: valore al di sotto del quale il gradiente può essere
 *              considerato nullo,
 *             -numero massimo di iterazioni concesse. Terminate le iterazioni si ritorna
 *              il valore ottimo sino a quel punto (questo per evitare che il programma
 *              entri in un ciclo infinito senza fornire alcun risultato).
 * Restituisce il valore del minimo.
 * Questo algoritmo non adotta la tecnica del multistart.
 */
double gradienteConiugato(double q[DIM][DIM], double *b, double *xIniziale, double *xmin, int dim,
double eps, int maxiter)
{
	/*direzione dello spostamento*/
	double *d=new double[dim];
	/*vettore del gradiente*/
	double *grad=new double[dim];

	if(!d || !grad)
	{
		printf("Errore nell'allocazione dinamica della memoria\n");
		exit(1);
	}

	fgrad(xIniziale, q, b, grad, dim);

	double normaGrad2=0, normaGradPrecedente2=0;

	/*inizializzazioni per la prima iterazione*/
	for(int i=0; i<dim; i++)
	{
		xmin[i]=xIniziale[i];
		normaGradPrecedente2+=grad[i]*grad[i];
		d[i]=-grad[i];
	}

	double minimo=0;

	/*ciclo principale*/
	for(i=0; i<maxiter; i++)
	{
		/*****calcolo del coefficiente alfa*****/
		double num=0, den=0;
		for(int j=0; j<dim; j++)
		{
			num-=d[j]*grad[j];
			for(int k=0; k<dim; k++)
			{
				den+=d[j]*q[j][k]*d[k];
			}
		}
		if(den==0)
		{
			printf("Errore nel calcolo del coefficiente alfa. Denominatore=0\n");
			exit(1);
		}
		if(num==0)
		{
			printf("coefficiente alfa=0\n");
```

```
		}
		double alfa=num/den;
		if(alfa<=0)
			printf("Warning! alfa<=0\n");

		/*****fine calcolo coefficiente alfa*****/

		/*****calcolo nuovo punto di minimo*****/
		for(j=0; j<dim; j++)
		{
			xmin[j]+=alfa*d[j];
		}

		fgrad(xmin, q, b, grad, dim); /*calcolo gradiente*/


		/*****calcolo di beta*****/
		normaGrad2=0;
		for(j=0; j<dim; j++)
		{
			normaGrad2+=grad[j]*grad[j];
		}
		double beta;
		if(normaGradPrecedente2==0 || !((i+1)%dim))
			beta=0;
		else
			beta=normaGrad2/normaGradPrecedente2;
		/*****fine calcolo beta*****/

		/*preparo per la prossima iterazione*/
		normaGradPrecedente2=normaGrad2;

		/*controllo che g e d siano ortogonali*/
		double dot=0;
		for(j=0; j<dim; j++)
			dot+=d[j]*grad[j];

		/*****calcolo della nuova direzione*****/
		for(j=0; j<dim; j++)
		{
			d[j]=-grad[j]+beta*d[j];
		}

		minimo=fcosto(xmin, q, b, dim);
		if(normaGrad2<0)
		{
			printf("Errore!! Norma del gradiente < 0\n");
			exit(1);
		}
		/*stampa valori intermedi*/
		printf("%d\t%lf\t%lf\t%lf\t%lf\t%lf\n", i, normaGrad2, minimo, beta, alfa, dot);

		if(sqrt(normaGrad2)<=eps)	break;
	}
	/*rilascio memoria*/
	delete[] d;
	delete[] grad;

	/*se sono arrivato a questo punto ho esaurito il numero massimo di iterazioni,
	 *restituisco il miglior valore trovato sino ad ora.
	 */
	if(i==maxiter) printf("Raggiunto limite massimo di iterazioni\n");
	return minimo;
}


/* Funzione costo della forma quadratica. Questa funzione viene
 * chiamata da gradienteConiugato().
 */
double fcosto(double *x, double q[DIM][DIM], double *b, double dim)
{
	double retval=0;
	for(int i=0; i<dim; i++)
	{
		retval+=b[i]*x[i];
		for(int j=0; j<dim; j++)
		{
			retval+=0.5*x[i]*q[i][j]*x[j];
		}
```

```
        }
        return retval;
}


/* Funzione gradiente della forma quadratica. Questa funzione viene
 * chiamata da gradienteConiugato().
 */
void fgrad(double *x, double q[DIM][DIM], double *b, double *grad, double dim)
{
        for(int i=0; i<dim; i++)
        {
                grad[i]=b[i];
                for(int j=0; j<dim ;j++)
                {
                        grad[i]+=q[i][j]*x[j];
                }
        }
}
```

L'algoritmo del gradiente coniugato converge in n+1 passi, dove con n si indica la dimensione dello spazio considerato. L'algoritmo non converge in n+1 passi quando il problema è mal condizionato, ovvero quando il rapporto tra autovalore massimo e minimo della matrice **Q** è un valore molto grande.

Esempio

Consideriamo ad esempio la forma quadratica $\frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} = \left(\frac{x_1}{\gamma^2}\right)^2 + \left(x_x\right)^2$ con γ ? 1 . Il minimo di questa forma quadratica è in $\begin{bmatrix}0\\0\end{bmatrix}$. Abbiamo:

$\mathbf{Q} = \begin{bmatrix}\frac{1}{\gamma^2} & 0\\ 0 & 1\end{bmatrix}$, i cui autovalori sono $\begin{cases}\lambda_1 = \frac{1}{\gamma^2}\\ \lambda_2 = 1\end{cases}$. Il rapporto tra gli autovalori è $\frac{\lambda_2}{\lambda_1} = \gamma^2$ può essere fatto diventare grande a piacere scegliendo opportunamente γ . Volendo disegnare delle curve di livello abbiamo:

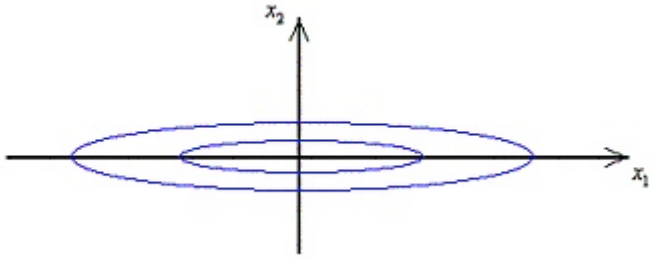


Figura 5

Si può notare come su x_1 la forma si estenda molto di più che su x_2 . Il gradiente della forma quadratica di cui sopra è $\mathbf{g} = \begin{bmatrix} \frac{x_1}{\gamma^2} \\ x_2 \end{bmatrix}$.

Si può utilizzare questo problema per confrontare la velocità di convergenza dell'algoritmo del gradiente coniugato rispetto a quello del gradiente discendente.

Gradiente discendente: poniamo ad esempio $\eta_0 = 0.1$. Scegliendo un punto iniziale $\mathbf{x}_0 = \begin{bmatrix} 50 \\ 1 \end{bmatrix}$,

abbiamo:

$$\mathbf{g}_0 = \begin{bmatrix} 0.005 \\ 1 \end{bmatrix} \Rightarrow \mathbf{x}_1 = \mathbf{x}_0 - \eta \mathbf{g}_0 = \begin{bmatrix} 50 \\ 1 \end{bmatrix} - \eta \begin{bmatrix} 0.005 \\ 1 \end{bmatrix} \cong \begin{bmatrix} 50 \\ 0.9 \end{bmatrix}. \ \eta_0 \text{ viene aumentato a 1.05.}$$

$$\mathbf{g}_1 = \begin{bmatrix} 0.005 \\ 0.9 \end{bmatrix} \Rightarrow \mathbf{x}_2 = \mathbf{x}_1 - 1.05\mathbf{g}_1 \cong \begin{bmatrix} 49.99 \\ 0.8 \end{bmatrix} \text{ e così via per le iterazioni successive.}$$

Si può notare come lungo x_1 l'algoritmo si muova di una quantità irrisoria. Il tempo di convergenza è molto lungo.

Gradiente coniugato: con lo stesso punto iniziale, abbiamo:

$$\mathbf{d}_0 = -\mathbf{g}_0 = \begin{bmatrix} -0.005 \\ -1 \end{bmatrix} \Rightarrow \alpha_0 = -\frac{\mathbf{g}_0^T \mathbf{d}_0}{\mathbf{d}_0^T \mathbf{Q} \mathbf{d}_0} \cong 1 \Rightarrow \mathbf{x}_1 = \mathbf{x}_0 + 1 \cdot \mathbf{d}_0 \cong \begin{bmatrix} 49.995 \\ 0 \end{bmatrix}. \text{ Quindi}$$

$$\mathbf{g}_1 = \begin{bmatrix} 0.005 \\ 0 \end{bmatrix} \Rightarrow \beta_0 = \frac{\|\mathbf{g}_1\|^2}{\|\mathbf{g}_0\|^2} \cong 0.000025 \Rightarrow \mathbf{d}_1 = -\mathbf{g}_1 + \beta_0 \mathbf{d}_0 \cong \begin{bmatrix} -0.005 \\ 0 \end{bmatrix}. \text{ Poiché}$$

$$\alpha_1 = -\frac{\mathbf{g}_1^T \mathbf{d}_1}{\mathbf{d}_1^T \mathbf{Q} \mathbf{d}_1} \cong 10000 \Rightarrow \mathbf{x}_2 = \mathbf{x}_1 + 10000 \mathbf{d}_1 \cong \begin{bmatrix} -0.005 \\ 0 \end{bmatrix}.$$

Si può notare come già al secondo passo siamo già in prossimità del minimo.

Questi risultati possono essere ricavati anche sperimentalmente utilizzando il seguente programma:

```
/* Ellisse.cpp
 *
 * Questo programma confronta i metodi del gradiente discendente e quello del gradiente
 * coniugato per la ricerca del minimo della forma quadratica (x1/GAMMA^2)^2+(x1)^2.
 *
 *
 * by Mauro Gaggero & Alessandro Di Fusco
 * mauro.gaggero1@tin.it
 * aledifu@libero.it
 *
```

```
 * 02/04/2003
 */

#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define GAMMA 100
#define MAXITER 1000000
#define DIM 2

double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, double *sx, double *dx, int dim, double eps, double etaIniziale,
int maxstart, int maxiter);
double randnum(long int&);
double f(double*, int);
void gf(double*, double*, int);

double gradienteConiugato(double q[DIM][DIM], double *b, double *xIniziale, double *xmin, int dim,
double eps, int maxiter);
double fcosto(double *x, double q[DIM][DIM], double *b, double dim);
void fgrad(double *x, double q[DIM][DIM], double *b, double *grad, double dim);


int main()
{
	double xmin[DIM];

	printf("*****Gradiente discendente*****\n");
	double xIniziale[DIM]={50, 1};
	double sx[DIM]={-1000, -1000};
	double dx[DIM]={+1000, +1000};
	double eps=1.e-6;
	double etaIniziale=0.1;
	int maxstart=1000;
	double minimoGradiente=gradiente(f, gf, xIniziale, xmin, sx, dx, DIM, eps, etaIniziale,
maxstart, MAXITER);
	printf("MinimoGradiente = %lf\n", minimoGradiente);
	printf("ottenuto per x1 = %lf\tx2 = %lf\n", xmin[0], xmin[1]);

	printf("\n\n*****Gradiente conugato*****\n");
	double q[DIM][DIM];
	q[0][0]=1./(GAMMA*GAMMA);
	q[0][1]=0;
	q[1][0]=0;
	q[1][1]=1;
	double r[DIM]={0, 0};
	double minimoConiugato=gradienteConiugato(q, r, xIniziale, xmin, DIM, eps, MAXITER);
	printf("MinimoConiugato = %lf\n", minimoConiugato);
	printf("ottenuto per x1 = %lf\tx2 = %lf\n", xmin[0], xmin[1]);

	return 0;
}


//funzione gradiente() (come prima) nella sua versione multistart (modificata con il salvataggio
su // file

//funzione randnum() (come prima)

//funzione gradienteConiugato() (come prima)


/* Funzione costo della forma quadratica. Questa funzione viene
 * chiamata da gradienteConiugato().
 */
double fcosto(double *x, double q[DIM][DIM], double *b, double dim)
{
	double retval=0;
	for(int i=0; i<dim; i++)
	{
		retval+=b[i]*x[i];
		for(int j=0; j<dim; j++)
		{
			retval+=0.5*x[i]*q[i][j]*x[j];
		}
	}
	return retval;
```

```
}


/* Funzione gradiente della forma quadratica. Questa funzione viene
 * chiamata da gradienteConiugato().
 */
void fgrad(double *x, double q[DIM][DIM], double *b, double *grad, double dim)
{
        for(int i=0; i<dim; i++)
        {
                grad[i]=b[i];
                for(int j=0; j<dim ;j++)
                {
                        grad[i]+=q[i][j]*x[j];
                }
        }
}


/* Funzione di cui calcolare il minimo. Vengono passati l'array
 * dei parametri e la dimensione dell'array (numero di variabili).
 * Restituisce il valore della funzione nel punto x.
 * La funzione qui è chiamata da gradiente().
 */
double f(double *x, int dim)
{
        return 0.5 * ( (x[0]*x[0])/(GAMMA*GAMMA) + x[1]*x[1] );
}


/* Gradiente della funzione di cui calcolare il minimo. Vengono passati
 * l'array dei parametri e quello del gradiente in cui sarà salvato il
 * risultato. Gli array hanno dimensione pari a dim(numero di variabili).
 * La funzione qui è chiamata da gradiente().
 */
void gf(double *x, double *grad, int dim)
{
        grad[0]=x[0]/(GAMMA*GAMMA);
        grad[1]=x[1];
}
```

Questo programma produce come uscita i risultati seguenti. I valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta / start / x1 / x2) per il gradiente discendente e (iterazione / norma gradiente al quadrato / minimo / beta / alfa / dot) per il gradiente coniugato.

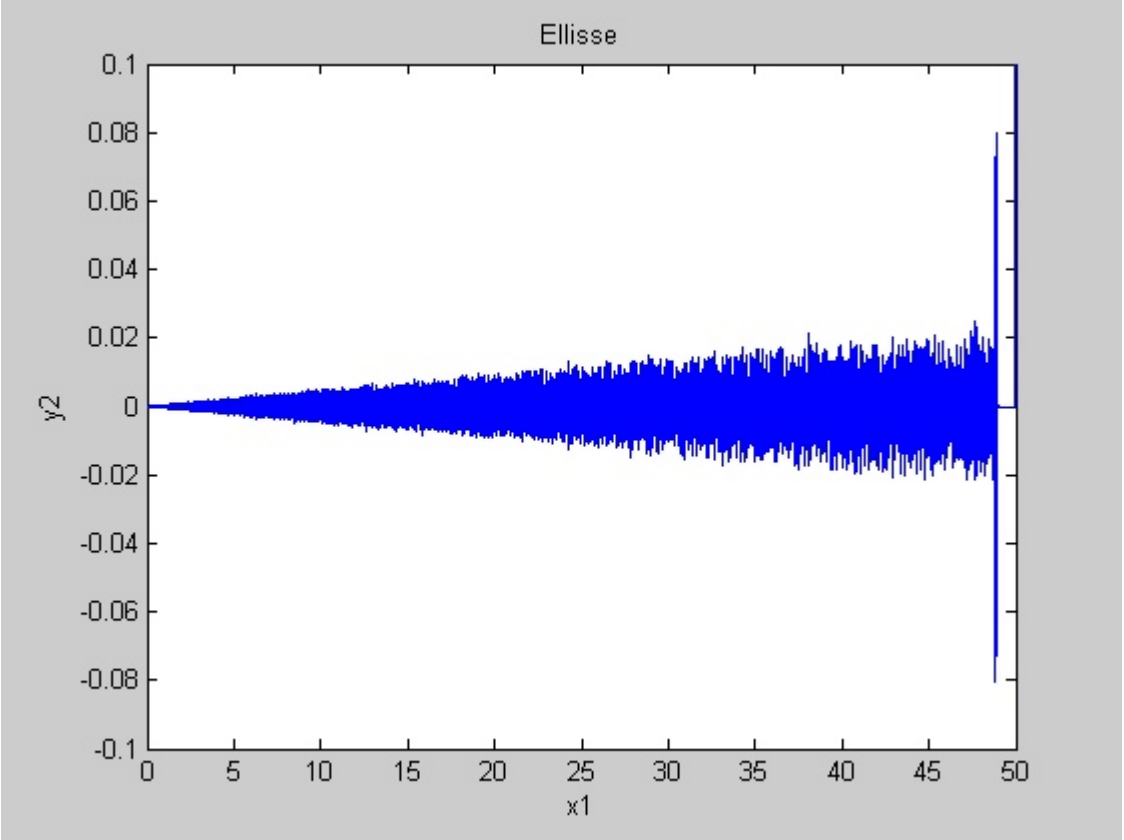
```
*****Gradiente discendente*****
0       1.000012        0.529998        0.105000        0       49.999500       0.900000
1       0.900014        0.449410        0.110250        0       49.998975       0.805500
2       0.805516        0.381817        0.115763        0       49.998424       0.716694
3       0.716711        0.325794        0.121551        0       49.997845       0.633727
4       0.633747        0.279942        0.127628        0       49.997237       0.556697
.....
42638   0.000004        0.000000        1.580926        0       0.008577        -0.000002
42639   0.000002        0.000000        1.659973        0       0.008576        0.000001
42640   0.000001        0.000000        1.742971        0       0.008574        -0.000001
42641   0.000001        0.000000        1.830120        0       0.008573        0.000001
42642   0.000001        0.000000        1.921626        0       0.008571        -0.000000
MinimoGradiente = 0.000000
ottenuto per x1 = 0.008571      x2 = -0.000000

*****Gradiente conugato*****
0       0.000025        0.124975        0.000025        1.000025        0.000000
1       0.000000        0.000000        0.000000        9999.750031     -0.000000
MinimoConiugato = 0.000000
ottenuto per x1 = 0.000000      x2 = 0.000000
```

Analizzando l'uscita si nota immediatamente come il gradiente coniugato converga in un solo passo, contro i ben 42642 del gradiente discendente! Nel caso del gradiente discendente si verifica

il fenomeno dello *zig-zag*, ovvero il punto di minimo continua ad oscillare senza fermarsi stabilmente sul minimo effettivo della funzione. Visualizzando su un grafico l'andamento del punto di minimo alle varie iterazioni si può osservare questo comportamento. Il grafico sotoostante è stato realizzato in Matlab basandosi sul file "data.dat" prodotto dalla funzione `gradiente()` opportunamente modificato per il salvataggio dei punti di minimo alle varie iterazioni. La scala del grafico è stata alterata rispetto all'originale per permettere l'osservazione del fenomeno dello *zig-zag*.

Il codice –matlab usato è il seguente:

```
% Ellisse.m
%
% Legge i dati dal file "data.dat" e li grafica
%
% by Mauro Gaggero & Alessandro Di Fusco
% mauro.gaggero1@tin.it
% aledifu@libero.it
%
% 04/04/2003

fid=fopen('data.dat', 'r');
res=fscanf(fid, '%lf');

len=length(res)/2;
x1=zeros(1, len);
x2=zeros(1,len);

j=1;
for i=1:len
    x1(i)=res(j);
    j=j+1;
    x2(i)=res(j);
    j=j+1;
end

plot(x1, x2, '-');
axis([0, max(x1), -0.1, 0.1]);
xlabel('x1');
ylabel('y2');
title('Ellisse');

fclose(fid);
```



Sonar

Vediamo ora l'applicazione dell'algoritmo del gradiente discendente alla soluzione del problema quadratico ricavabile con i valori presenti nel file "sonar.txt". Il programma utilizzato è il seguente:

```
/* Sonar.cpp
 *
 * Confronto tra gli algoritmi del gradiente discendente e del gradiente coniugato
 * per la soluzione del problema quadratico derivante dei dati presenti sul file
 * "sonar.txt".
 *
 *
 * by Mauro Gaggero & Alessandro Di Fusco
 * mauro.gaggero1@tin.it
 * aledifu@libero.it
 *
 * 02/04/2003
 */


#include <iostream.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

#define MAXITER 2000000000

#define ni 60
#define np 52
#define DIM 52

double gradiente(double (*f)(double *x, int dim), void (*gf)(double *x, double *grad, int dim),
double *xIniziale, double *xmin, double *sx, double *dx, int dim, double eps, double etaIniziale,
int maxstart, int maxiter);
double randnum(long int&);
double gradienteConiugato(double q[DIM][DIM], double *b, double *xIniziale, double *xmin, int dim,
double eps, int maxiter);
double fcosto(double *x, double q[DIM][DIM], double *b, double dim);
```

```
void fgrad(double *x, double q[DIM][DIM], double *b, double *grad, double dim);
double f(double*, int);
void gf(double*, double*, int);

double q[np][np], r[np];

int main()
{
	int i, j, l;
	FILE *fp;
	double x[ni][np], y[np];
	double u, tmp;

	/*inizio lettura del file "sonar.txt"*/
	fp=fopen("sonar.txt","r");
	if (fp==NULL)
	{
		printf("Errore in apertura file\n");
		exit(-1);
	}
	for(i=0;i<np;i++)
	{
		for (l=0;l<ni;l++)
		{
			if(fscanf(fp,"%lf",&u)!=1)
			{
				printf("Errore in lettura coordinate dato %d pattern %d \n",l,i);
				exit(-1);
			}
		    x[l][i]=u;
		}
		if(fscanf(fp,"%lf",&u)!=1)
		{
			printf("Errore in lettura target pattern %d \n",i);
			exit(-1);
		}
		y[i]=u;
		if (y[i]<.5)
			y[i]=-1;
	}
	fclose(fp);

	for(i=0;i<np;i++)
	{				/*Inizializzazione matrice q e vettore r*/
		r[i]=-1.;
		for(j=0;j<np;j++)
		{
			tmp=0.;
			for(l=0;l<ni;l++)
				tmp+=x[l][i]*x[l][j];
			q[i][j]=y[i]*y[j]*tmp;
		}
	}
	/*fine lettura file e generazione matrice Q e vettore r*/

	double xmin[np];
	double xIniziale[np];
	double sx[np];
	double dx[np];
	for(i=0; i<np; i++)
	{
		xIniziale[i]=0;
		sx[i]=-1e300;
		dx[i]=+1e300;
	}

	double eps=1e-3;
	double etaIniziale=0.1;
	int maxstart=100000;

	printf("*****Gradiente discendente*****\n");
	double minimoDiscendente=gradiente(f, gf, xIniziale, xmin, sx, dx, np, eps, etaIniziale,
maxstart, MAXITER);
	printf("Minimo = %lf\n", minimoDiscendente);

	printf("\n\n*****Gradiente coniugato*****\n");
	double minimoConiugato=gradienteConiugato(q, r, xIniziale, xmin, np, eps, MAXITER);
	printf("Minimo = %lf\n", minimoConiugato);
```

```
        return 0;
}



/* Funzione di cui calcolare il minimo. Vengono passati l'array
 * dei parametri e la dimensione dell'array (numero di variabili).
 */
double f(double *x, int dim)
{
        double tmp=0;
        for(int i=0; i<np; i++)
        {
                tmp+=r[i]*x[i];
                for(int j=0; j<np; j++)
                        tmp+=0.5*x[i]*q[i][j]*x[j];
        }
        return tmp;
}


/* Gradiente della funzione di cui calcolare il minimo. Vengono passati
 * l'array dei parametri e la dimensione dell'array (numero di variabili).
 * Ritorna la norma euclidea del gradiente.
 */
void gf(double *x, double *grad, int dim)
{
        for(int i=0; i<np;i++)
        {
                grad[i]=r[i];
                for(int j=0; j<np; j++)
                        grad[i]+=q[i][j]*x[j];
        }
}


//funzione gradiente() (come prima) nella sua versione multistart

//funzione randnum() (come prima)

//funzione gradienteConiugato (come prima)

//funzione fcosto() (come prima)

//funzione fgrad() (come prima)
```

L'uscita del programma è la seguente (considero `np=DIM=12` per velocizzare il testing). I valori visualizzati durante le iterazioni sono (iterazione / norma gradiente / minimo / eta / start) per il gradiente discendente e (iterazione / norma gradiente al quadrato / minimo / beta / alfa / dot) per il gradiente coniugato:

```
*****Gradiente discendente*****
0       3.464102        0.000000        70.000000       0
1       3.464102        0.000000        49.000000       0
2       3.464102        0.000000        34.300000       0
3       3.464102        0.000000        24.010000       0
4       3.464102        0.000000        16.807000       0
5       3.464102        0.000000        11.764900       0
.....
3978    0.001479        -12.415748      0.027713        0
3979    0.001635        -12.415748      0.029099        0
3980    0.001961        -12.415748      0.020369        0
3981    0.001961        -12.415748      0.021387        0
3982    0.001387        -12.415748      0.022457        0
3983    0.001129        -12.415748      0.023580        0
3984    0.001012        -12.415748      0.024759        0
Minimo = -12.415748

*****Gradiente coniugato*****
0       166.486141      -2.408152       13.873845       0.401359        0.000000
1       20.279646       -4.376878       0.121810        0.023650        -0.000000
2       10.637357       -6.996313       0.524534        0.258331        0.000000
3       2.617795        -10.090360      0.246094        0.581732        0.000000
```

4	0.190596	-12.204807	0.072808	1.615442	-0.000000
5	0.036733	-12.348379	0.192729	1.506549	-0.000000
6	0.022890	-12.370035	0.623145	1.179091	-0.000000
7	0.005457	-12.390990	0.238406	1.830981	0.000000
8	0.204701	-12.400353	37.510442	3.431277	-0.000000
9	0.005647	-12.401850	0.027585	0.014626	0.000000
10	0.000620	-12.414669	0.109839	4.540306	0.000000
11	0.000042	-12.415672	0.000000	3.234625	-0.000000
12	0.000003	-12.415746	0.081834	3.540336	0.000000
13	0.000008	-12.415751	2.205269	2.924982	-0.000000
14	0.000010	-12.415752	1.249972	0.117330	-0.000000
15	0.000000	-12.415752	0.018002	0.014704	-0.000000

Minimo = -12.415752

Si può notare come i due algoritmi arrivino allo stesso risultato, ma in tempi decisamente diversi. L'algoritmo del gradiente discendente impiega ben 3984 iterazioni prima di raggiungere il valore minimo, mentre il gradiente coniugato solo 15. Questa differenza si può spiegare con il fatto che il problema da cui derivano i dati presenti sul file "sonar.txt" è mal condizionato.