PROGETTO FINALIZZATO
SISTEMI INFORMATICI E CALCOLO PARALLELO

SOTTOPROGETTO 1
Calcolo Scientifico per Grandi Sistemi

Coordinatore: Laura Moltedo

Andrea Bosin*

MINIMIZZAZIONE DI FUNZIONI REALI A MOLTE VARIABILI CON IL METODO DEL SIMULATED ANNEALING

N. 1/157 **Aprile 1993**

RAPPORTO TECNICO

*Dipartimento di Scienze Fisiche dell'Università di Cagliari

Sommario

Il programma esegue la minimizzazione di una funzione reale (funzione di costo o energia) a più variabili reali, fornita dall'utente (in forma di una function C o una subroutine FORTRAN). L'algoritmo di base utilizzato è quello del simulated annealing con campionamento Metropolis Monte Carlo dello spazio di definizione della funzione (spazio delle configurazioni). In particolare la distribuzione statistica è quella di Boltzmann mentre la dinamica Monte Carlo è quella proposta da Vanderbilt e Louie, e cioè è regolata da una matrice covarianza. Quest'ultima viene modificata durante la dinamica stessa a partire dalle informazioni accumulate sulla funzione di costo, in modo da massimizzare l'efficienza di campionamento dello spazio. Il tipo di algoritmo implementato è adatto alla minimizzazione di funzioni multimodali (a molti minimi relativi o locali) di molte variabili reali e, in linea di principio, è in grado di determinare il minimo globale (assoluto) della funzione. In pratica, a causa della statistica finita, la probabilità di trovare il minimo globale è però minore di 1. In generale, la bontà del minimo trovato, rispetto a quello assoluto, dipende dall'estensione del campionamento statistico richiesto dall'utente.

Parole chiave: simulated annealing, ottimizzazione, Monte Carlo.

Versione: *SA versione 1.0, 25 febbraio 1993.*

Abstract

The program executes the minimization of a user-supplied real function (cost function or energy) of many real variables (in the form of a C function or a FORTRAN subroutine). The basic algorithm is the simulated annealing with Metropolis Monte Carlo sampling of the space over which the function is defined (configuration space). In particular, Boltzmann's statistical distribution and the Monte Carlo dynamics proposed by Vanderbilt and Louie, regulated by a covariance matrix, are used. The latter is modified during the dynamics itself using the information gained on the cost function, in such a way to maximize the sampling efficiency. The implemented algorithm is well suited to the minimization of multimodal functions (with many relative or local minima) of many real variables and, in principle, it can determine the global (absolute) minimum of such functions. In practice, due to the finite statistics, the probability of finding the global minimum is less than 1. In general, the quality of the minimum reached, compared with the global one, depends on the extension of the statistical sampling generated by the user.

Keywords: simulated annealing, optimization, Monte Carlo.

Version: *SA version 1.0, february 25, 1993.*

Indice

1 Simulated Annealing

1.1 Introduzione

Una situazione che si presenta spesso nei problemi di ottimizzazione è quella di aver a che fare con la minimizzazione di una funzione multimodale, e cioè una funzione che presenta molti punti di minimo relativo o minimi locali. In generale quello a cui siamo interessati non è uno qualunque di questi minimi locali, bensì il punto o i punti di minimo globale (o assoluto, se è unico) dove la funzione assume il suo valore minimo (stiamo supponendo che il minimo della funzione esista e consideriamo solo funzioni limitate su un dominio finito). A volte, anche uno dei minimi locali, purché non troppo distante da quello globale, può essere una soluzione accettabile per il problema.

In questa situazione, gli algoritmi di minimizzazione "tradizionali"[1] (per esempio i metodi basati sul gradiente o il metodo dei simplessi, che a partire da un certo punto iniziale considerano un insieme di punti nel dominio di definizione in cui la funzione è via via più piccola, fino a raggiungere un punto di minimo relativo) non sono utilizzabili. Questo perché non sono in grado di distinguere fra un minimo locale e uno globale e il minimo raggiunto dipende dal punto iniziale scelto. L'idea di un algoritmo completamente diverso, in grado di operare almeno in linea di principio questa distinzione, si basa sull'analogia fra il nostro problema di minimo e il seguente sistema fisico.

Una sostanza che a temperatura $T = 0$ si trovi in uno stato di cristallo perfetto (minimo assoluto della sua energia) può essere riportata in questo stato dopo essere stata riscaldata a $T > 0$. A seconda della velocità di raffreddamento lo stato cristallino finale potrà presentare un certo numero di difetti (stati metastabili che corrispondono a un minimo relativo dell'energia del sistema) ma, se il raffreddamento è sufficientemente lento, lo stato finale sarà quello di cristallo perfetto in quanto esso presenta la massima probabilità. Se lo stato finale è metastabile, è necessario procedere a una serie di riscaldamenti e successivi raffreddamenti (o cicli di annealing) finché non si raggiunge lo stato di cristallo perfetto o uno stato metastabile sufficientemente vicino. Il sistema fisico considerato è in grado di distinguere fra lo

stato di minima energia e gli altri stati metastabili ed è possibile costruire un algoritmo di minimizzazione che simula tale comportamento (da cui il nome di simulated annealing), e cioè in grado di distinguere fra minimi locali e globali.

1.2 Il metodo Metropolis Monte Carlo

La costruzione di un algoritmo che si basa sulle idee esposte nel paragrafo precedente è dovuta a Metropolis *et al.*[2]. Osserviamo che il sistema fisico considerato, all'equilibrio termodinamico a temperatura T, compie un moto nello spazio delle fasi che da un punto di vista statistico è regolato dalla distribuzione di Boltzmann:

$$P(E) = \exp\left(-\frac{E}{T}\right) \tag{1}$$

in un opportuno sistema di unità di misura. E è l'energia del sistema in un certo punto dello spazio delle fasi e $P(E)$ la sua probabilità. Per $T \gg E$ si ha $P(E) \simeq 1$ mentre per $T \to 0$, detta E_0 l'energia minima, risulta:

$$\frac{P(E)}{P(E_0)} = \exp\left(\frac{E_0 - E}{T}\right) \ll 1 \qquad E \neq E_0 \tag{2}$$

e, per un raffreddamento infinitamente lento, il minimo dell'energia è raggiunto con probabilità unitaria[3].

Sia ora E il valore della funzione da minimizzare (che chiameremo anche funzione di costo o energia) nel dominio finito $D \subset \mathbf{R}^N$ e introduciamo una temperatura fittizia $T > 0$; consideriamo poi la successione "Monte Carlo" di punti o passi $\{x^{(n)}\}_n$ nel dominio (o spazio delle configurazioni) così generata: data una stima iniziale $x^{(0)}$ del punto di minimo, ogni configurazione $x^{(n+1)}$ è scelta dall'insieme

$$\{\bar{x} \in \mathbf{R}^N : \bar{x} = x^{(n)} + \mathbf{Q}\xi, \forall \xi \in [-\sqrt{3}, \sqrt{3}]^N\} \tag{3}$$

con probabilità condizionale

$$P(\bar{x}, x^{(n)}) = \begin{cases} \exp\left(-\dfrac{E(\bar{x}) - E(x^{(n)})}{T}\right) & \text{se } E(\bar{x}) > E(x^{(n)}) \\ 1 & \text{se } E(\bar{x}) \leq E(x^{(n)}) \end{cases} \tag{4}$$

data la configurazione precedente $x^{(n)}$, dove $\mathbf{Q}$ è una matrice $N \times N$ proporzionale all'identità (matrice di ampiezza del passo Monte Carlo); se $\bar{x} \notin D$ si può, per esempio, porre $P(\bar{x}, x^{(n)}) = 0$. In pratica, la scelta di $x^{(n+1)}$ si fa generando un numero pseudo-casuale[1] ξ uniformemente distribuito in $[-\sqrt{3}, \sqrt{3}]^N$ (ogni componente ha media nulla e varianza unitaria), da cui si calcola $\bar{x}$ e $P(\bar{x}, x^{(n)})$, e un numero pseudo-casuale χ uniformemente distribuito in $[0, 1]$: se $\chi \leq P(\bar{x}, x^{(n)})$ poniamo $x^{(n+1)} = \bar{x}$, altrimenti, se $\chi > P(\bar{x}, x^{(n)})$, si genera un nuovo ξ e si ripete il procedimento appena discusso. Si può dimostrare[2] che, dopo un numero sufficientemente elevato di passi, la successione $\{x^{(n)}\}_n$ generata (o, in altri termini, la "dinamica" Metropolis Monte Carlo del punto $x^{(0)}$) è distribuita secondo la funzione di Boltzmann e cioè ogni configurazione di energia E è pesata con probabilità $\exp(-E/T)$. In questo modo l'analogia con il sistema statistico è completa e si procede alla minimizzazione in maniera del tutto analoga, eseguendo uno o più cicli di annealing in cui viene variata la temperatura T fino al raggiungimento del minimo globale per la funzione di costo. È importante osservare che, in mancanza di una stima iniziale accurata del punto di minimo, è fondamentale partire da una temperatura T sufficientemente alta in modo da avere un campionamento il più possibile completo dello spazio delle configurazioni (infatti, se $T \gg E(x)$, $\forall x \in D$, risulta $P(E) \simeq 1$ e tutto lo spazio è accessibile); in caso contrario il "congelamento" in un minimo locale è molto probabile.

1.3 L'algoritmo di Vanderbilt e Louie

Una delle limitazioni principali nell'uso diretto del metodo introdotto nella sezione precedente è la scarsa efficienza di campionamento dello spazio delle configurazioni. Infatti, la grandezza $\mathbf{Q}$ che regola l'ampiezza di ogni passo Monte Carlo dovrebbe essere tale che approssimativamente la metà degli $\bar{x}$ generati nell'Eq. (3) venga accettata, in modo da avere un guadagno di informazione ottimale e quindi la massima efficienza di campionamento. Il rapporto f fra passi ($\bar{x}$ accettati) e $\bar{x}$ generati, però, dipende dai dettagli della funzione di costo "accessibili" alla temperatura T (se $T \gg E(x)$, $\forall x \in D$, tutto lo spazio è accessibile, e i dettagli non sono importanti; altrimenti solo

le regioni in cui $P(E) \simeq 1$ sono accessibili e dunque per campionarle efficacemente è importante sapere come sono fatte) e quindi è necessario variare $\mathbf{Q}$ di conseguenza, abbandonando la semplice proporzionalità alla matrice identità. In particolare, passi troppo corti sono poco efficienti nell'esplorare lo spazio delle configurazioni (f grande), mentre passi troppo lunghi comportano uno spreco di risorse (f piccola); se la funzione non è isotropa, l'ampiezza del passo deve essere diversa nelle differenti direzioni. Inoltre, con la diminuzione della temperatura T, il passo deve diminuire, dato che diminuisce il volume dello spazio accessibile. Il metodo di Metropolis *et al.*, però, non da alcuna prescrizione su come ciò vada fatto. L'innovazione introdotta da Vanderbilt e Louie[4] è quella di introdurre un meccanismo autoregolante per la distribuzione dei passi che assicura una scelta efficiente per l'ampiezza e l'anisotropia del passo Monte Carlo lungo tutto l'annealing: vediamo come. Dividiamo ogni procedura di annealing in un certo numero di insiemi di M passi (o cicli) fatti a temperatura T fissata. Osserviamo che il numero M deve essere sufficientemente grande per garantire il raggiungimento dell'analogo dell'equilibrio termico del sistema fisico, e cioè permettere un campionamento abbastanza fitto di tutte le regioni accessibili dello spazio delle configurazioni. Su ciascuno di questi cicli calcoliamo le varie componenti del primo e secondo momento del cammino Monte Carlo $\{A_\alpha^{(i)}\}_i$, $\{S_{\alpha\beta}^{(i)}\}_i$, con $i \in \{1, 2, 3, \ldots\}$ indice di ciclo e $\alpha, \beta \in \{1, \ldots, N\}$ indici di coordinata in $\mathbf{R}^N$:

$$A_\alpha^{(i)} = \frac{1}{M} \sum_{n=1}^{M} x_\alpha^{(n,i)} \tag{5}$$

$$S_{\alpha\beta}^{(i)} = \frac{1}{M} \sum_{n=1}^{M} [x_\alpha^{(n,i)} - A_\alpha^{(i)}][x_\beta^{(n,i)} - A_\beta^{(i)}] \tag{6}$$

$\{x^{(n,i)}\}_{n=1}^{M}$ sono le successioni di M passi Monte Carlo per il ciclo i-esimo. La matrice $\mathbf{S}^{(i)}$ contiene informazioni statistiche sulla forma e l'estensione dell'effettivo cammino negli M passi considerati e quindi anche sulla forma della funzione di costo campionata. Si può usare questa informazione per migliorare l'efficienza di campionamento nel successivo ciclo $(i+1)$-esimo, come spiegato in dettaglio nel Rif. [4], costruendo una matrice covarianza $\mathbf{s}$ mediante la formula iterativa (vedi più avanti la relazione fra $\mathbf{s}$ e la genera-

zione del cammino Monte Carlo tramite la matrice $\mathbf{Q}$):

$$s_{\alpha\beta}^{(i+1)} = \gamma \frac{\chi_S}{\beta M} S_{\alpha\beta}^{(i)} + (1-\gamma) s_{\alpha\beta}^{(i)} \tag{7}$$

χ_S (fattore di crescita) e β sono due costanti reali definite nel Rif. [4] e $0 < \gamma \leq 1$ è un fattore che permette di controllare quanta nuova informazione (proveniente dall'ultimo ciclo) inserire nella matrice covarianza per il ciclo successivo ($\gamma < 1$ garantisce una maggiore stabilità numerica). La matrice covarianza iniziale, in mancanza di informazioni specifiche, viene scelta proporzionale alla matrice identità:

$$s_{\alpha\beta}^{(1)} = \lambda^2 \delta_{\alpha\beta} \qquad \lambda > 0 \tag{8}$$

Nel seguito chiameremo covarianza scalare la grandezza $\lambda^2 = \sqrt[N]{\det(\mathbf{s})}$. Matrice covarianza $\mathbf{s}$ e matrice di ampiezza $\mathbf{Q}$ per il cammino Monte Carlo sono per definizione legate dalla relazione:

$$\mathbf{s} = \mathbf{Q}\mathbf{Q}^T \tag{9}$$

e una volta nota $\mathbf{s}$, mediante la sua decomposizione di Cholesky[1], è possibile costruire $\mathbf{Q}$. Osserviamo che $Q_{\alpha\beta}^{(1)} = \lambda \delta_{\alpha\beta}$ e che questa scelta genera una distribuzione isotropa di passi di lunghezza quadratica media $\lambda\sqrt{N}$.

La procedura di annealing, come abbiamo visto, è composta da vari cicli (di M passi ciascuno, a meno che non risulti $f < f_{inf}$, vedere più avanti) durante ognuno dei quali la temperatura T è costante, mentre deve essere diminuita passando da un ciclo al successivo. Questo può essere fatto usando la formula:

$$T^{(i+1)} = \chi_T T^{(i)} \tag{10}$$

$0 < \chi_T < 1$ deve essere scelto empiricamente mediante "prova ed errore" dato che, se χ_T è troppo piccolo, è alta la probabilità di un intrappolamento in un minimo locale, mentre se χ_T è troppo grande stiamo sprecando risorse. Durante l'annealing si possono presentare situazioni di instabilità numerica quali una matrice covarianza singolare o una frazione f di $\bar{x}$ (eventi) generati rispetto a quelli accettati troppo grande ($f > f_{sup}$) o troppo piccola ($f < f_{inf}$), Eq. (3). In questi casi il ciclo successivo è portato avanti alla

stessa temperatura e utilizzando una matrice covarianza ancora proporzionale alla matrice identità; se $\mathbf{s}^{(i+1)}$ calcolata con l'Eq. (7) risulta numericamente singolare poniamo:

$$T^{(i+1)} = T^{(i)} \tag{11}$$

$$s_{\alpha\beta}^{(i+1)} = \sqrt[N]{\det(\mathbf{s}^{(i)})}\delta_{\alpha\beta} \tag{12}$$

Se invece risulta $f > f_{sup}$ poniamo:

$$T^{(i+1)} = T^{(i)} \tag{13}$$

$$s_{\alpha\beta}^{(i+1)} = \sqrt[N]{\sigma \det(\mathbf{s}^{(i)})}\delta_{\alpha\beta} \qquad \sigma > 1 \tag{14}$$

Infine, se $f < f_{inf}$, poniamo:

$$T^{(i+1)} = T^{(i)} \tag{15}$$

$$s_{\alpha\beta}^{(i+1)} = \sqrt[N]{\sigma^{-1} \det(\mathbf{s}^{(i)})}\delta_{\alpha\beta} \qquad \sigma > 1 \tag{16}$$

Per limitare il numero di eventi generati e non accettati quando f è troppo piccola, il controllo su $f < f_{inf}$ viene fatto ogni M eventi e non alla fine del ciclo (M eventi accettati); questo permette di avere abbastanza statistica per il calcolo di f (purchè M venga scelto in maniera ragionevole dall'utente), pur limitando lo spreco di risorse.

Osserviamo, come suggeriscono Kirkpatrick *et al.*[5], che un picco nel "calore specifico":

$$C(T) = \frac{1}{T^2}[\langle E^2\rangle - \langle E\rangle^2] \tag{17}$$

(la media $\langle\rangle$ è fatta sul cammino Monte Carlo), per analogia con la meccanica statistica, può indicare che la configurazione sta diventando "ordinata" e cioè sta congelando in un minimo; a questo punto può essere importante ridurre χ_T in modo da evitare il congelamento in un minimo che non sia quello globale.

Infine, ci serve un criterio per determinare se l'annealing sia giunto a convergenza e cioè se abbia raggiunto un minimo (locale o globale); per fare questo stabiliamo una certa soglia di accuratezza numerica η e, alla fine di ogni ciclo, interrompiamo la procedura se risulta:

$$\frac{\langle E\rangle - E_{min}}{\langle E\rangle} < \eta \tag{18}$$

dove E_{min} è il valore minimo dell'energia E sul ciclo.

2 Installazione del programma

2.1 Introduzione

Prima di poter utilizzare effettivamente il programma, che è scritto in linguaggio C, è necessario passare attraverso le fasi di configurazione e installazione. La configurazione si divide a sua volta in due punti: da una parte è necessario specificare tutte le informazioni relative al tipo di hardware e di sistema operativo con cui il programma deve funzionare, dall'altra bisogna fornire le informazioni necessarie per adattare il programma alle proprie esigenze. La fase di installazione, che è stata automatizzata il più possibile, permette di generare l'eseguibile vero e proprio, costruito a partire dalle informazioni fornite durante la configurazione. Il programma viene fornito sotto forma di file binario (usate l'opzione *bin* con *ftp*).

Per il sistema operativo (SO) Unix è un file di tipo *tar* compresso di nome *sa-1.0.tar.Z*. Prima di procedere è necessario decomprimerlo con il comando *uncompress sa-1.0.tar.Z* e quindi ripristinarne la struttura con il comando *tar xvf sa-1.0.tar*; quest'ultima operazione crea una directory di nome *sa-1.0* che contiene tutti i file relativi alla distribuzione 1.0 del programma. Per procedere a configurazione e installazione posizionatevi nella directory *sa-1.0* con il comando *cd sa-1.0*.

Per il SO VAX/VMS è un file di tipo *backup* compresso, di nome *SA-1-0.BCK_Z*, che va decompresso con il comando:

DECOMPRESS SA-1-0.BCK_Z SA-1-0.BCK

DECOMPRESS non fa parte del SO VAX/VMS ma è un simbolo definito come:

DECOMPRESS == "$LZDCMP.EXE"

dove *LZDCMP* è un programma di decompressione "public domain" che utilizza l'algoritmo di Lempel-Ziv-Welch; con il comando:

BACKUP SA-1-0.BCK/SAVE [.SA-1-0].**

si genera una directory di nome *[.SA-1-0]* che contiene tutti i file relativi alla distribuzione 1.0 del programma; con *SET DEF [.SA-1-0]* siete pronti per configurazione e installazione.

2.2 Configurazione

Per effettuare la configurazione è necessario modificare alcuni dei file contenuti nella distribuzione. Innanzitutto procuratevi le seguenti informazioni (fra parentesi è riportata la configurazione di default sia per il SO Unix che per VAX/VMS):

- tipo di hardware (HW) utilizzato (default Unix: *Sun SPARCstation 10*; default VAX/VMS: *VAX 6410*);
- sistema operativo (default Unix: *SunOS 4.1.3*; default VAX/VMS: *VMS 5.4-3*);
- compilatore per il linguaggio C (default Unix: *SunOS 4.1.3 C*; default VAX/VMS: *VAXC 3.1*);
- terminale per l'input/output (I/O) (default Unix: qualunque terminale gestito da questo SO; default VAX/VMS: qualunque terminale di tipo ANSI, quali i VT100 e modelli successivi).

Editate il file *Makefile* (per il SO Unix) o il file *make.com* (per VAX/VMS); cercate nell'elenco iniziale il codice pre-definito per il vostro hardware, sistema operativo e compilatore C. I codici pre-definiti sono riportati anche nella Tab. 1. Se il codice non esiste, ma ne esiste uno che si riferisce a HW, SO e compilatore simili ai vostri, provate a utilizzare quello, altrimenti dovete modificare opportunamente il file *sa.h* e aggiungere le vostre definizioni (non è previsto, però, alcun supporto per questa operazione). Seguendo le direttive presenti nei due file e nel seguito, completate le istruzioni:

- **SYSTEM** = con il codice pre-definito;
- **DIR** = con la directory in cui andranno posti i file di input e output;
- **TERMINAL** = con uno dei due codici seguenti che identificano il tipo di terminale utilizzato: **T_UNIX** per qualunque terminale gestito dal SO Unix o **T_ANSI** per un terminale ANSI (quali i terminali VT gestiti da VAX/VMS).

CODICE	HARDWARE	SO	C
VAX6410	Digital VAX 6410	VMS 5.4-3	VAXC 3.1
VAX6320	Digital VAX 6320	VMS 5.5-2	VAXC 3.1
VAX3400	Digital VAX server 3400	VMS 5.3-1	VAXC 3.1
DEC5810	Digital DEC System 5810	Ultrix 4.2	Ultrix C
VAX2000	Digital VAX 2000	Ultrix-32 3.1	Ultrix C
SG320	Silicon Graphics 320/GTX	IRIX System V 4.0.1	MIPS C
SGIRIS	Silicon Graphics IRIS-4D	IRIX System V 4.0.5	MIPS C
SUN4	Sun 4	SunOS 4.1.1	SunOS C
SPARC10	Sun SPARCstation 10	SunOS 4.1.3	SunOS C
NEXT	NeXT Mach	NeXT Mach Unix	GNU C

Tabella 1: codici predefiniti per la configurazione.

- **ENERGY** = con il nome del file e della function C o subroutine FORTRAN in questo contenuta, che calcola la funzione di costo;
- **FC** = con il nome del compilatore FORTRAN che si intende utilizzare, nel caso in cui la funzione di costo sia fornita sotto forma di subroutine FORTRAN;
- **FFLAGS** = con le opzioni per il compilatore FORTRAN (vedi punto precedente).

Una volta completate le modifiche salvate il file.

La seconda fase riguarda la personalizzazione del file *sadef.h* (e *sadef.inc* se la cost function è scritta in FORTRAN). Editate tale file e variate le costanti definite secondo le vostre esigenze. Le più importanti sono (in parentesi il tipo numerico e l'intervallo di valori consentito):

- **PSDIM** massimo numero di variabili da cui dipende la funzione di costo (intero positivo);
- **SAVESTEP** numero di eventi fra un aggiornamento e l'altro dei file temporanei (intero positivo, vedere la Sez. 3.2);

- **GF** fattore di crescita χ_S (reale, $\chi_S > 1$);
- **GB** fattore β (reale positivo);
- **FBK** fattore di feedback γ (reale, $0 < \gamma \leq 1$);
- **ACPRH** frazione massima f_{sup} (reale, $0 < f_{inf} < f_{sup} \leq 1$);
- **ACPRL** frazione minima f_{inf} (reale, $0 < f_{inf} < f_{sup} \leq 1$);
- **SCOVSF** costante di riduzione della matrice covarianza, σ (reale positiva);
- **CEPS** soglia di accuratezza numerica η (reale positiva).

Se la cost function è una subroutine FORTRAN le costanti che compaiono con lo stesso nome in entrambi i file *sadef.h* e *sadef.inc* devono essere definite allo stesso modo. Salvate i file modificati.

2.3 Funzione di costo

Una volta configurati i file *Makefile* o *make.com*, *sa.h*, *sadef.h* e *sadef.inc* è necessario scrivere in un file a parte (nella directory che contiene tutta la distribuzione) il codice (in C o FORTRAN) che calcola la funzione di costo. Tale file deve avere il nome specificato da **ENERGY** e terminare in *.c* (function C) o in *.f* (subroutine FORTRAN). La function C deve essere definita come nel file prototipo *energy_proto.c* (vedere anche il file di esempio *energy.c*) e avere il nome specificato da **ENERGY**. Lo stesso vale nel caso di una subroutine FORTRAN, ma con il file prototipo *energy_proto.f* (file di esempio *energy.f*).

La function o subroutine ENERGY (il nome maiuscolo indica che il nome vero è quello specificato dalla costante **ENERGY** definita in *Makefile* o *make.com*) viene chiamata all'interno delle function *init_siman* e *siman* (vedere Sez. 3.1) nel modo seguente (linguaggio C):

ENERGY (&npar,par,&e,&err);

con *npar* (input) numero N di variabili da cui dipende la funzione di costo, *par* (input) vettore che contiene il punto in cui la funzione va calcolata, *e* (output) valore della funzione nel punto specificato e *err* (output) condizione di errore se diversa da zero (vedere più avanti); *npar, par, e, err* sono definite come:

int npar, err;
double e;
double par[PSDIM];

PSDIM è descritto nella Sez. 2.2. Per compatibilità con il FORTRAN tutte le variabili sono passate a ENERGY specificando il loro indirizzo (o puntatore).

Se *err* è diversa da zero per un certo valore di *par* si possono verificare due eventi a seconda che *par* corrisponda a $x^{(0)}$ o a $\bar{x}$. Nel primo caso si ha la possibilità di interrompere l'esecuzione e cambiare il punto di partenza, oppure si può proseguire nella dinamica Monte Carlo (e allora è importante che il valore di *e* ritornato da ENERGY sia ben definito). Nel secondo caso il valore di *e* viene ignorato e $P(\bar{x}, x^{(n,i)})$ viene posto uguale a zero (sia (n, i) l'indice del passo da cui $\bar{x}$ viene generato), il che corrisponde ad associare alla funzione di costo un valore infinito. In questo modo è possibile, per esempio, limitare il dominio di ricerca assegnando a *err* un valore non nullo ogni volta che *par* è fuori da tale dominio.

2.4 Installazione

Dopo aver completato le operazioni di configurazione elencate nelle sezioni precedenti, la generazione dell'eseguibile è automatica. Con il SO Unix basta dare il comando *make c* o *make fortran* per creare l'eseguibile finale *sa*, con VAX/VMS *@make c* o *@make fortran*; *c* e *fortran* si riferiscono rispettivamente a una funzione di costo fornita come function C o subroutine FORTRAN. Nel caso del comando *make fortran* su Sun, ignorate l'eventuale messaggio d'errore "Undefined symbol _MAIN_".

3 Descrizione e utilizzo del programma

3.1 Struttura del codice

Il programma è scritto in linguaggio C e consta di circa 1200 linee di codice. Il disegno è modulare nel senso che il programma è suddiviso in blocchi funzionali o moduli (ogni blocco funzionale è una function C) distribuiti in vari file: *sa.c, cholesky.c, error.c, init.c, move.c, save.c, siman.c, stat.c, trace.c* e *ENERGY.c o ENERGY.f* che è il file fornito dall'utente con le modalità esposte nella Sez. 2.4. Esistono poi due file *sadef.h* e *sadef.inc* che contengono solo le definizioni delle costanti usate nei vari moduli e un file *sa.h* che contiene le rimanenti definizioni e la dichiarazione delle funzioni di libreria utilizzate (vedere la Sez. 2.2).

Segue una breve descrizione di ogni modulo (in parentesi il nome del file che lo contiene) con le funzioni chiamanti e chiamate (escluse quelle di libreria). Per i parametri e i file di I/O vedere la Sez. 3.2.

- **main** (sa.c): programma principale; lettura (da terminale) dei parametri principali della procedura di annealing, inizializzazione e supervisione della stessa mediante costruzione e aggiornamento della matrice covarianza, Eq. (7), diminuzione della temperatura alla fine di ogni ciclo, Eq. (10) e controllo della condizione di convergenza, Eq. (18).
  CHIAMATE A: *init_proc, init_trace, init_siman, error, user_exit, cont_trace, init_cov, init_stat, siman, end_stat, save, update_trace, cholesky.*
  CHIAMATA DA: nessuna.

- **user_exit** (sa.c): uscita dal programma.
  CHIAMATE A: nessuna.
  CHIAMATA DA: *main, onintr, init_proc, init_cycle, save, all, init_trace, cont_trace, update_trace.*

- **cholesky** (cholesky.c): decomposizione di Cholesky.
  CHIAMATE A: nessuna.
  CHIAMATA DA: *main.*

- **error** (error.c): visualizzazione dei messaggi d'errore.

CHIAMATE A: nessuna.
CHIAMATA DA: *main*.

- **onintr** (error.c): gestione del segnale di interruzione.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: gestore dei segnali di sistema.

- **init_cov** (init.c): costruzione della matrice covarianza proporzionale alla matrice identità, Eq. (8).
  CHIAMATE A: *cholesky*.
  CHIAMATA DA: *main*.

- **init_siman** (init.c): inizializzazione del punto di partenza e calcolo della funzione di costo nello stesso.
  CHIAMATE A: *init_cycle, ENERGY*.
  CHIAMATA DA: *main*.

- **init_proc** (init.c): inizializzazione del file di output *anneal.dat*.
  CHIAMATE A: nessuna.
  CHIAMATA DA: *main*.

- **init_cycle** (init.c): lettura dei file *anneal.dat* e *params.dat* all'inizio di ogni run.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *init_siman*.

- **move** (move.c): calcolo di $\bar{x}$, Eq. (3).
  CHIAMATE A: *step*.
  CHIAMATA DA: *siman*.

- **accept_move** (move.c): assegnazione di $x^{(n+1)} = \bar{x}$, Eq. (3).
  CHIAMATE A: nessuna.
  CHIAMATA DA: *siman*.

- **step** (move.c): calcolo di $\mathbf{Q}\xi$, Eq. (3).
  CHIAMATE A: *unif*.
  CHIAMATA DA: *move*.

- **save** (save.c): preparazione dei dati per i file di output *params.dat, params.old, params.min* e *params.tmp* e scrittura di *anneal.dat* e *anneal.tmp*.
  CHIAMATE A: *savx, user_exit*.
  CHIAMATA DA: *siman*.

- **savetmp** (save.c): preparazione dei dati per i file di output *anneal.tmp*.
  CHIAMATE A: *save*.
  CHIAMATA DA: *siman*.

- **savx** (save.c): scrittura dei file di output *params.dat, params.old, params.min* e *params.tmp*.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *save*.

- **siman** (siman.c): generazione del cammino Monte Carlo a temperatura fissata e aggiornamento delle informazioni statistiche associate.
  CHIAMATE A: *all, ENERGY, statis, move, prob, unif, accept_move, savetmp*.
  CHIAMATA DA: *main*.

- **prob** (siman.c): calcolo di $P(\bar{x}, x^{(n)})$, Eq. (4).
  CHIAMATE A: nessuna.
  CHIAMATA DA: *siman*.

- **unif** (siman.c): generazione di un numero pseudo-casuale distribuito uniformemente nell'intervallo $[0, 1]$.
  CHIAMATE A: nessuna.
  CHIAMATA DA: *siman*.

- **all** (siman.c): allocazione della memoria necessaria ad immagazzinare il valore della funzione di costo lungo il cammino Monte Carlo.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *siman*.

- **init_stat** (stat.c): inizializzazione degli array contenenti $A_\alpha^{(i)}$ e $S_{\alpha\beta}^{(i)}$, Eq. (5,6).

CHIAMATE A: nessuna.
CHIAMATA DA: *main*.

- **statis** (stat.c): aggiornamento degli array contenenti $A_{\alpha}^{(i)}$ e $S_{\alpha\beta}^{(i)}$, Eq. (5) ed Eq. (6).
  CHIAMATE A: nessuna.
  CHIAMATA DA: *siman*.

- **end_stat** (stat.c): normalizzazione degli array contenenti $A_{\alpha}^{(i)}$ e $S_{\alpha\beta}^{(i)}$, Eq. (5,6).
  CHIAMATE A: nessuna.
  CHIAMATA DA: *siman*.

- **init_trace** (trace.c): inizializzazione del file di output *trace.dat*.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *main*.

- **cont_trace** (trace.c): aggiornamento del file di output *trace.dat* all'inizio di ogni run.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *main*.

- **update_trace** (trace.c): aggiornamento del file di output *trace.dat* dopo ogni ciclo.
  CHIAMATE A: *user_exit*.
  CHIAMATA DA: *main*.

- **ENERGY** (ENERGY.c): calcolo della funzione di costo.
  CHIAMATE A: definite dall'utente.
  CHIAMATA DA: *init_siman,siman*.

- **ENERGY** (ENERGY.f): calcolo della funzione di costo.
  CHIAMATE A: definite dall'utente.
  CHIAMATA DA: *init_siman,siman*.

3.2 Input e Output

Ogni procedura di annealing è composta da uno o più run in successione del programma (comando *sa* con il SO Unix, *run sa* con VAX/VMS). Questo significa che run successivi ma relativi a una stessa procedura devono essere pensati come un unico run logico in particolar modo nella gestione dell'I/O. Per questa ragione le operazioni di I/O del primo run di una procedura sono un pó diverse da quelle dei run successivi. In particolare, la presenza nella directory di lavoro del file *anneal.dat* indicherà che una procedura di annealing è già in corso. Sarà compito dell'utente decidere se è il caso di portarla avanti o se intraprenderne una nuova, quando richiesto all'inizio del run.

Incominciamo con la descrizione dell'I/O relativo al terminale. Solo per il primo run di ogni procedura verranno chieste due stringhe di identificazione della procedura stessa (identification name) e dell'utente (operator name) che saranno riportate nel nuovo file *trace.dat* assieme a data di creazione dello stesso, HW, SO, compilatore C e directory di lavoro per cui il programma è stato configurato. Per ogni run, dopo aver specificato se si intende proseguire con una procedura esistente o iniziarne una nuova (solo se esiste il file *anneal.dat* nella directory di lavoro), è necessario specificare i valori da assegnare alle quantità seguenti (in parentesi il tipo numerico e l'intervallo di valori ammessi):

- numero dei cicli (intero positivo);
- numero M dei passi per ogni ciclo (intero, $M \geq 2$);
- temperatura iniziale $T^{(1)}$ (reale positiva);
- fattore di riduzione della temperatura χ_T (reale, $0 < \chi_T < 1$);
- covarianza scalare iniziale λ^2 (reale positiva);
- seme del generatore di numeri pseudo-casuali (intero positivo);

Questi dati vengono trascritti in successione nel file *trace.dat* assieme alle costanti definite in *sadef.h*: *GF, GB, FBK, ACPRH, ACPRL, SCOVSF,*

CEPS. A questo punto la simulazione è partita e alla fine di ogni ciclo verrà visualizzata una tabella riassuntiva (oltre a eventuali messaggi di rimozione dei file temporanei, vedere più avanti) con le seguenti informazioni relative al ciclo appena completato:

- numero del ciclo,
- temperatura,
- covarianza scalare,
- numero di iterazioni, e cioè numero di $\bar{x}$ generati e quindi di valutazioni della funzione di costo,
- frazione di $\bar{x}$ accettati rispetto a quelli generati,
- valore iniziale della funzione di costo,
- valore finale,
- valore minimo,
- valore medio,
- deviazione standard,
- calore specifico.

La fine di un run per completamento di tutti i cicli richiesti senza che la convergenza sia stata raggiunta è segnalata dal solo messaggio "Execution successfully terminated"; se invece il run converge a un punto di minimo alla fine dell'ultimo ciclo eseguito, si ha anche il messaggio "Convergence threshold reached" (in tal caso i cicli rimasti non vengono eseguiti). Ogni altra terminazione del programma è anomala e viene segnalata da un opportuno messaggio di errore: vedere più avanti la discussione sui file temporanei.

Quanto visto sopra si può fare anche tramite file; con il SO Unix, per esempio, sia *input_file* il file che contiene i dati di input (uno per riga) e sia *output_file* il file che dovrà contenere l'output: l'istruzione di esecuzione *sa*

$<$ *input_file* $>$ *output_file* al posto di *sa* leggerà da *input_file* e scriverà su *output_file* invece che da e su terminale.

È necessario creare un solo file di input per il primo run di una procedura, *params.dat*, in cui scrivere, un dato per riga e nell'ordine seguente, il numero N di variabili della funzione di costo e le coordinate del punto iniziale $\{x_\alpha^{(0)}\}_\alpha$, a cominciare da $\alpha = 1$ fino a $\alpha = N$.

Per quanto riguarda i file di output essi sono: *trace.dat, anneal.dat, params.dat, params.old, params.min, anneal.tmp* e *params.tmp*. Come abbiamo visto *trace.dat* contiene tutti gli input forniti al programma durante una procedura di annealing e, per ogni ciclo eseguito, anche gli output riportati nella lista precedente. Il file *anneal.dat* contiene una tabella riassuntiva come quella in output su terminale che, però, raggruppa le informazioni su tutti i cicli eseguiti. Sia *trace.dat* che *anneal.dat* vengono creati dal primo run di una procedura e aggiornati in quelli successivi. In *params.dat*, che in input contiene $x^{(0)}$, alla fine di ogni ciclo vengono scritte (nello stesso formato) le coordinate del punto finale del cammino Monte Carlo; una volta raggiunta la convergenza, queste saranno le coordinate del minimo trovato (locale o globale). Lo stesso file viene utilizzato anche come input per il run successivo della stessa procedura, senza bisogno di modificarlo. Simile contenuto ha *params.old*, dove però l'informazione si riferisce non all'ultimo ciclo concluso, ma a quello precedente; *params.min* contiene invece le coordinate del punto di minimo per la funzione di costo lungo il cammino Monte Carlo dell'ultimo ciclo completato. I file temporanei con estensione *.tmp* sono equivalenti ai corrispondenti file *.dat* ma vengono aggiornati con una frequenza stabilita dal valore della costante *SAVESTEP* definita in *simdef.h*. In particolare *params.tmp* viene riscritto ogni *SAVESTEP* iterazioni dall'inizio del ciclo in corso, mentre *anneal.tmp* viene creato dopo *SAVESTEP* iterazioni e aggiornato ogni *SAVESTEP* (il numero di iterazioni per ciclo deve essere maggiore uguale a *SAVESTEP*). Entrambi i file vengono cancellati alla fine di ogni ciclo: la loro presenza indica un run in corso o un run interrotto in modo anomalo. Oltre a permettere una più flessibile supervisione sull'andamento della simulazione (più volte in un ciclo invece che solo alla fine), i file temporanei possono essere utili per proseguire dopo un fermo macchina che abbia

interrotto un run in corso: copiando *params.tmp* in *params.dat* si può ripartire dall'ultimo passo salvato su file (per avere *anneal.dat* aggiornato basta completarlo con l'ultima riga di *anneal.tmp*; non è possibile fare lo stesso con *trace.dat*). Alternativamente, copiando *params.old* in *params.dat*, si può ripartire dall'ultimo ciclo terminato normalmente, ignorando quello interrotto.

3.3 Un esempio di funzionamento

Per verificare il funzionamento del programma, con la distribuzione viene fornita una funzione di esempio da minimizzare, scritta come function C nel file *energy.c* o come subroutine FORTRAN nel file *energy.f*. Facendo riferimento alla Sez. 2.2 definite **SYSTEM** e **TERMINAL** nel file *Makefile* o *make.com*. Se intendete usare la versione FORTRAN, definite anche **FC** e **FFLAGS**. Le altre definizioni in *Makefile, make.com, sadef.h* e *sadef.inc*, come forniti con la distribuzione, sono ottimizzate per l'esempio che stiamo considerando. Generate l'eseguibile *sa*, per esempio, con *make c* o *@make c*. La funzione è definita in $D = [-100, 100] \times [-100, 100]$ ed è costituita da una somma di 10 gaussiane di centro, altezza e larghezza diverse (questi dati sono contenuti in *gauss.dat*). Essa presenta 3 minimi locali nei punti $x_1 \simeq (9.94, -54.3)$, $f(x_1) \simeq -5.2$, $x_2 \simeq (-31.7, 87.2)$, $f(x_2) \simeq -8.0$ e $x_3 \simeq (-85.4, -0.602)$, $f(x_3) \simeq -9.2$ che è anche il minimo assoluto. La stessa è rappresentata in Fig. 1. L'esecuzione del programma su una Sun SPARCstation 10 con i seguenti input:

- numero di cicli: 15,
- numero M di passi per ogni ciclo: 2000,
- temperatura iniziale $T^{(1)}$: 10.0,
- fattore di riduzione della temperatura χ_T: 0.5,
- covarianza scalare iniziale λ^2: 10000.0,
- seme del generatore di numeri pseudo-casuali: 237993317,

T	λ^2	f	E_{fin}	$\langle E \rangle$	$C(T)$
1.0e+01	1.0e+04	29.0%	7.8e-01	-5.7e-01	4.6e-02
5.0e+00	3.1e+03	43.2%	-4.4e-02	-1.0e+00	2.0e-01
2.5e+00	9.7e+02	39.5%	-1.1e-01	-1.8e+00	8.2e-01
1.2e+00	3.4e+02	22.9%	-1.0e+00	-5.1e+00	3.4e+00
6.2e-01	1.4e+02	8.1%	-8.3e+00	-5.1e+00	1.5e+01
6.2e-01	1.4e+01	24.6%	-8.5e+00	-8.2e+00	1.8e+00
3.1e-01	4.2e+00	30.6%	-9.1e+00	-8.8e+00	1.3e+00
1.6e-01	1.3e+00	40.0%	-9.1e+00	-9.0e+00	1.2e+00
7.8e-02	4.1e-01	50.7%	-9.1e+00	-9.1e+00	8.6e-01
3.9e-02	1.3e-01	59.2%	-9.1e+00	-9.1e+00	1.0e+00
2.0e-02	4.2e-02	65.7%	-9.2e+00	-9.2e+00	9.6e-01
9.8e-03	1.4e-02	71.8%	-9.2e+00	-9.2e+00	7.5e-01

Tabella 2: i campi più significativi del file di output *anneal.dat* per l'esempio considerato (E_{fin} è il valore della funzione di costo alla fine del ciclo).

ha portato alla minimizzazione della funzione in 12 cicli e il punto raggiunto è quello di minimo globale, x_3. Nella Tab. 2 riportiamo parte del file di output *anneal.dat* e nelle Fig. 2-7 una rappresentazione grafica su curve di livello del cammino Monte Carlo lungo i primi cicli. È interessante osservare nelle figure come il campionamento dello spazio cambia al diminuire della temperatura (Sez. 1.3) e nella tabella come il calore specifico varia bruscamente quando il sistema comincia a diventare "ordinato".

Al di la dell'esempio riportato, che vuole solo dare un'idea sul funzionamento del metodo, la convergenza al minimo globale, o più realisticamente a un minimo locale di buone caratteristiche, è molto più lenta, dipende dalla complessità della funzione di costo e in generale richiede più di un run. Per rendersene conto basta provare a usare gli stessi input di esempio variando solo il seme per il generatore di numeri pseudo-casuali. Su un campione di 100 run da noi effettuati, 48 convergono a x_1, 43 a x_2 e solo 9 a x_3 (la convergenza è raggiunta in un massimo di 13 cicli). Questo è dovuto alla differente "larghezza" delle regioni di minimo, come si può vedere in Fig. 1. Data l'alta

velocità di raffreddamento e la breve estensione del cammino Monte Carlo durante la fase di "ordinamento", la differenza di profondità fra i vari minimi è meno importante rispetto alla loro "estensione".

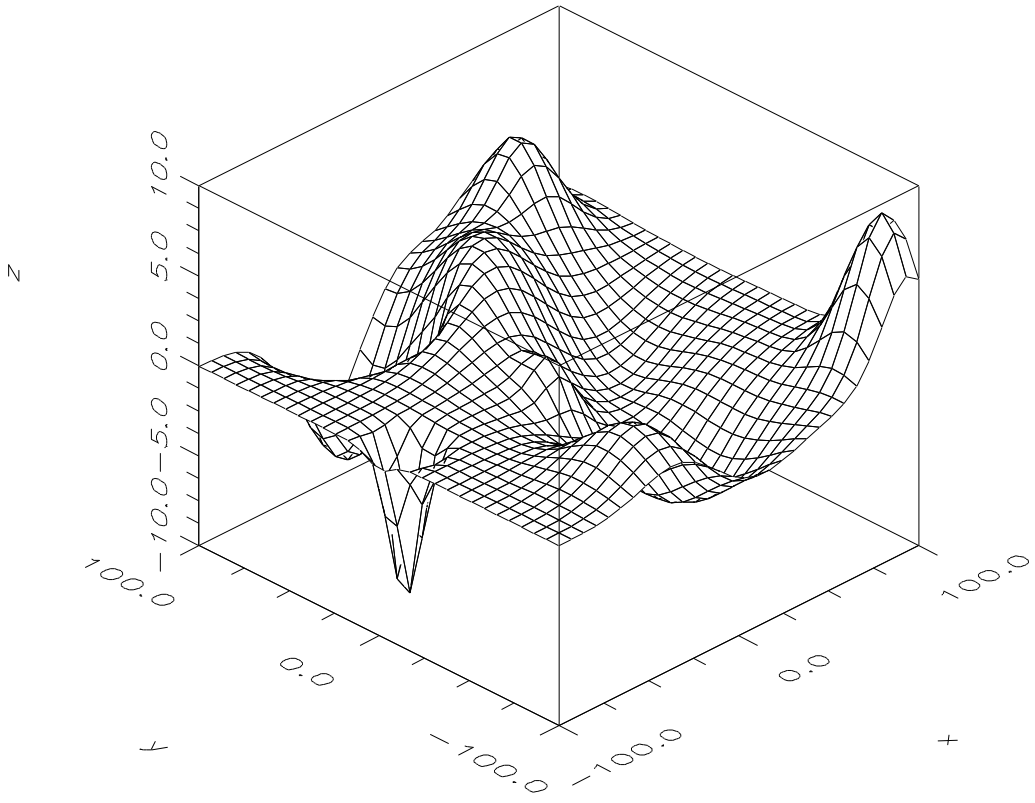

Figura 1: Rappresentazione tridimensionale della funzione considerata nell'esempio.

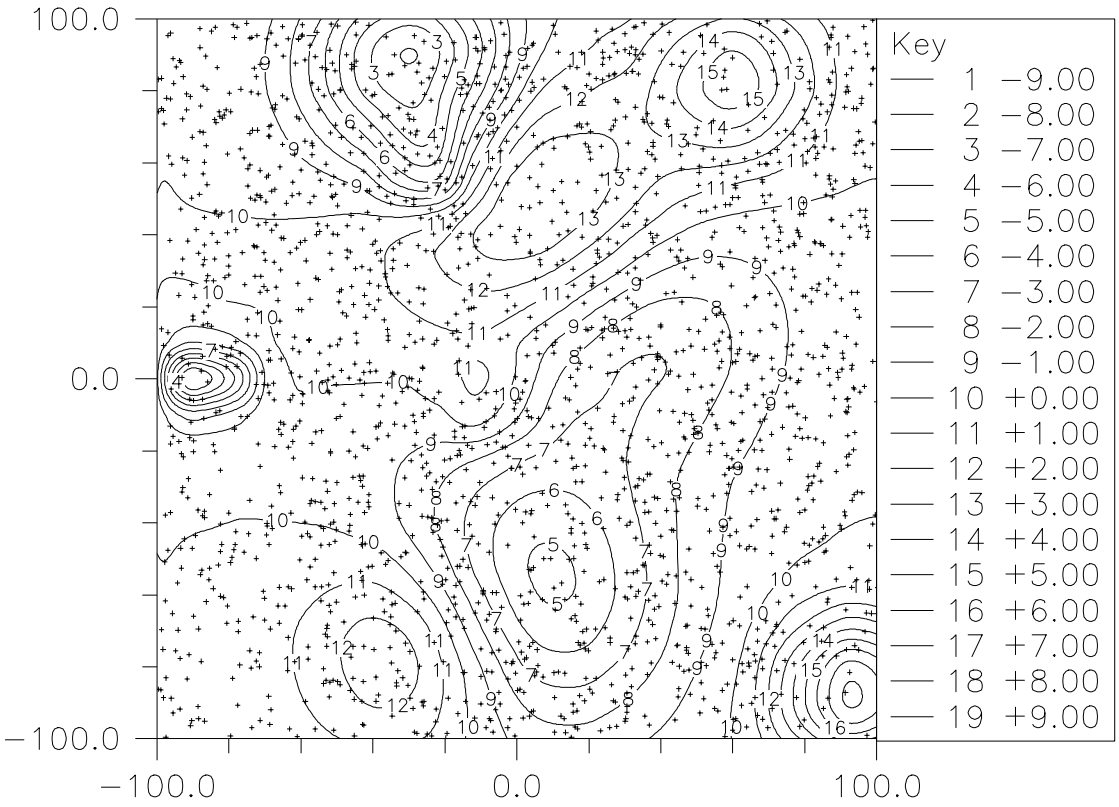

Figura 2: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il ciclo iniziale a temperatura $T^{(1)} = 10.0$.

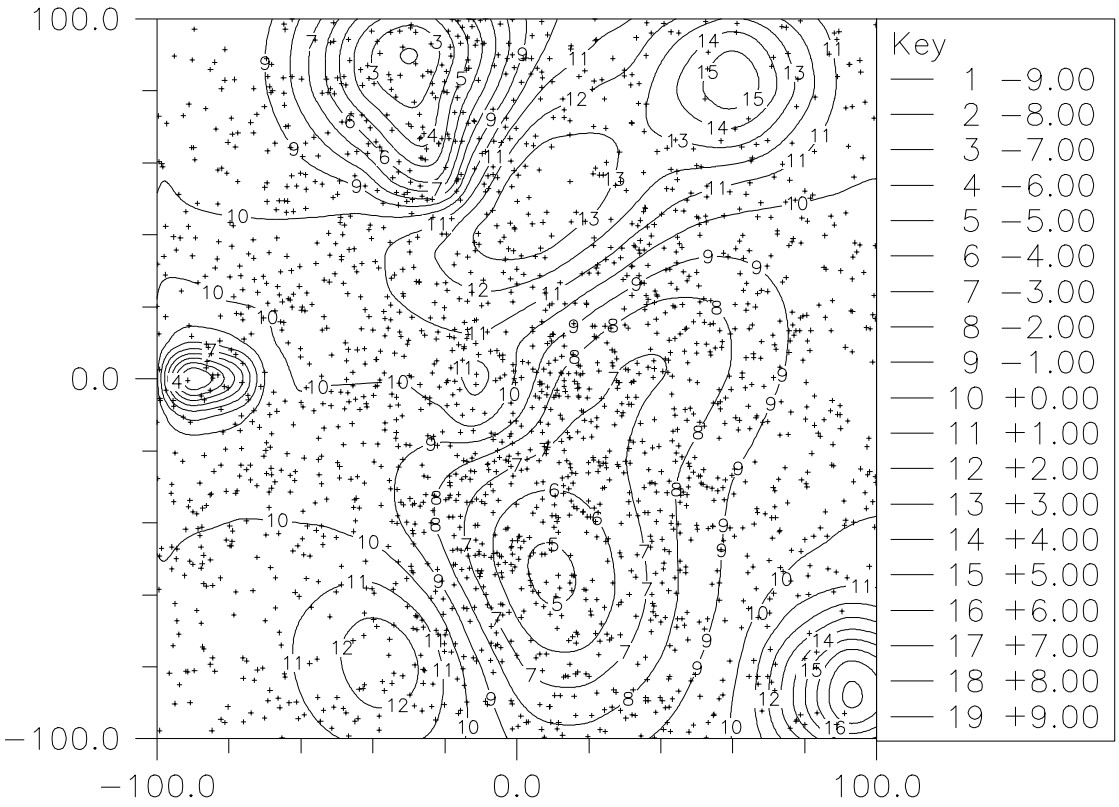

Figura 3: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il 2° ciclo a temperatura $T^{(2)} = 5.0$.

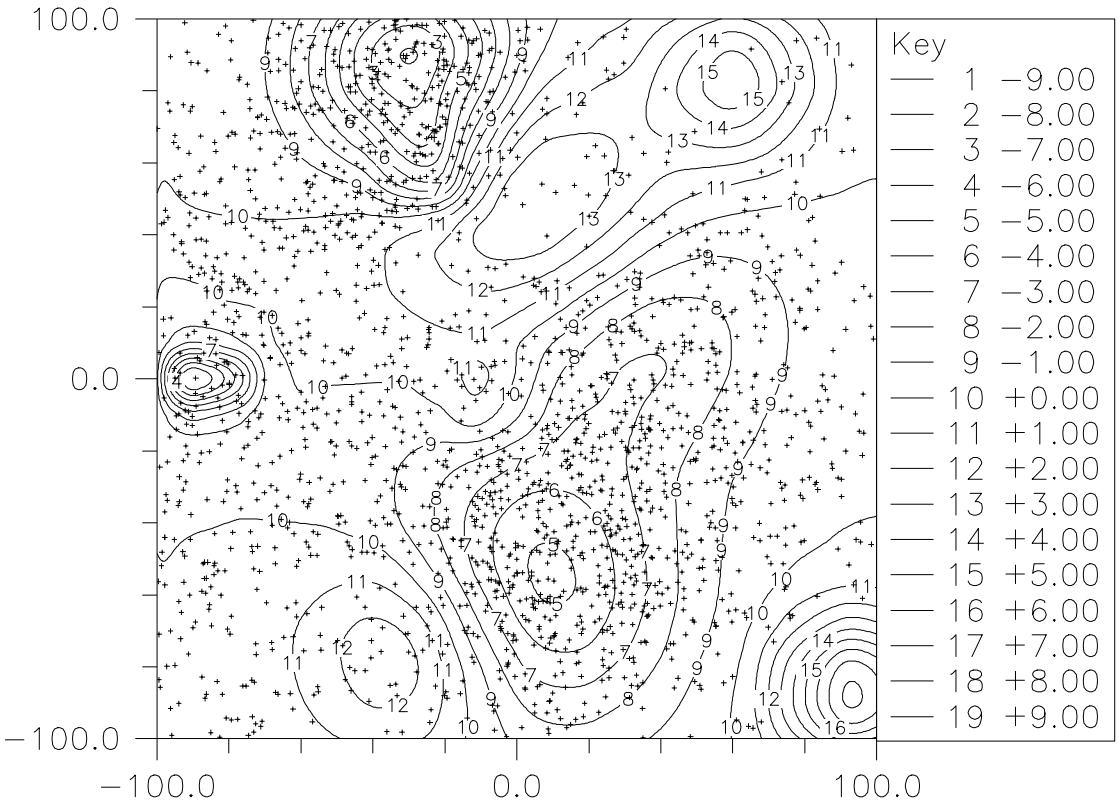

Figura 4: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il 3° ciclo a temperatura $T^{(3)} = 2.5$.

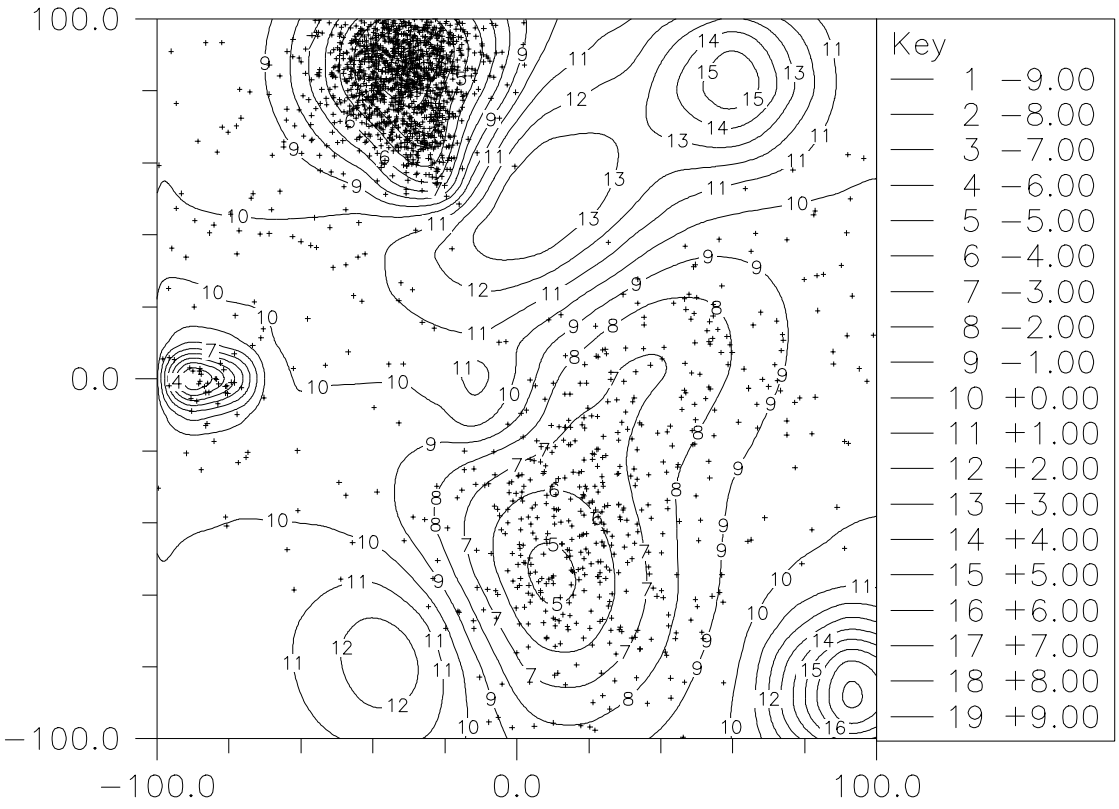

Figura 5: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il 4° ciclo a temperatura $T^{(4)} = 1.25$.

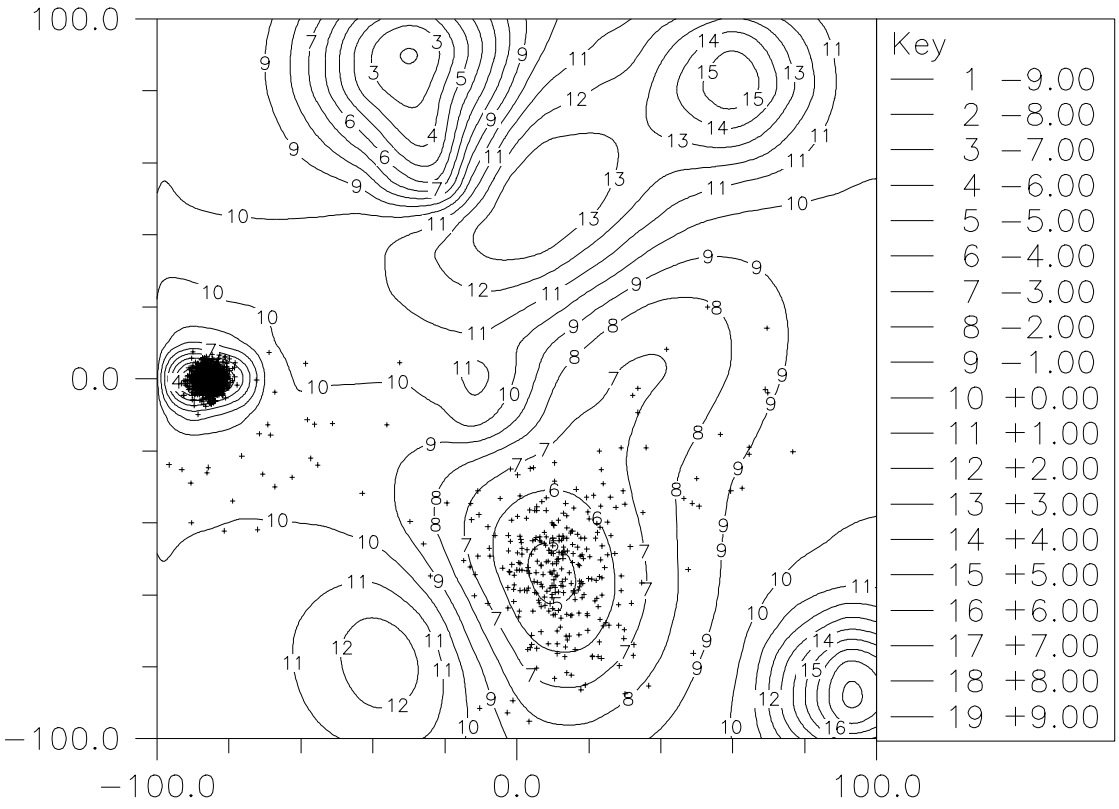

Figura 6: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il 5° e 6° ciclo a temperatura $T^{(5)} = T^{(6)} = 0.625$.

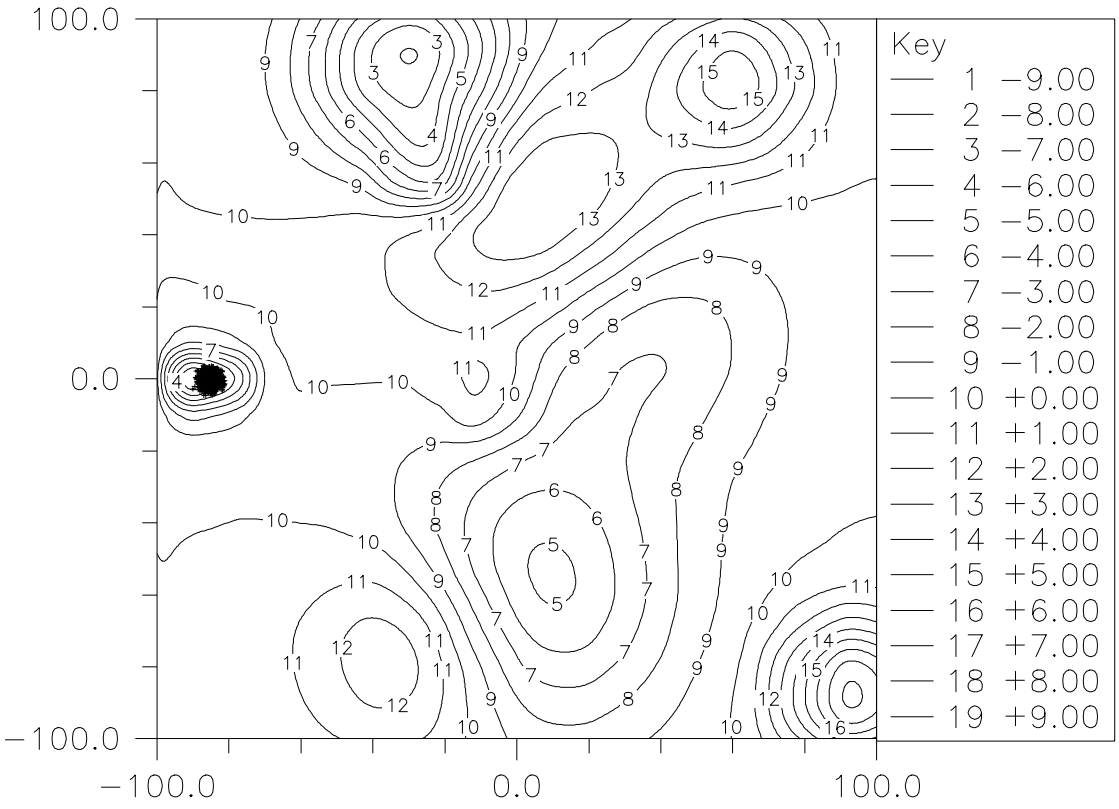

Figura 7: Cammino Monte Carlo (a ogni simbolo (+) corrisponde un passo) sovrapposto alle curve di livello della funzione di costo per il 7° ciclo a temperatura $T^{(7)} = 0.3125$.

Bibliografia

[1] W. H. Press, B. P. Flannery, S. A. Teukolsky e W. T. Vetterling, *Numerical Recipes*, (Cambridge University Press, USA, 1986)

[2] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller and E. Teller, *J. Chem. Phys.* **21**, 1087 (1953)

[3] P. J. M. van Laarhoven and E. H. L. Aarts, *Simulated Annealing: Theory and Applications*, (D. Reidel Publishing Company, Dordrecht, 1987)

[4] D. Vanderbilt and S. G. Louie, *J. Comp. Phys* **56**, 259 (1984)

[5] S. Kirkpatrik, C. D. Gelatt and M. P. Vecchi, *Science* **220**, 671 (1983)