IFATEST v.0.0: manuale del programmatore

Emanuele Salerno

ISTI-CNR
Via Moruzzi, 1 – 56124 Pisa

Abstract: IFATEST è un codice in Fortran 90 per la sperimentazione di un algoritmo di analisi ai fattori indipendenti (IFA) ispirato a quello presentato da Attias in [1]. Lo scopo di questo algoritmo è quello della separazione cieca di segnali ottenuti da misture istantanee rumorose di sorgenti indipendenti. La variante qui proposta [2] prevede l'uso di una strategia di annealing simulato piuttosto che l'algoritmo Expectation-Maximization proposto da Attias per l'apprendimento del modello dati. I potenziali vantaggi di questa strategia consistono nell'indipendenza della soluzione dalla stima iniziale e in una maggiore robustezza nei confronti del rumore. I primi risultati ottenuti [2] vanno verso una conferma di queste aspettative. Un altro sicuro vantaggio dell'annealing simulato sull'approccio di Attias è che le eventuali informazioni a priori su alcuni dei parametri incogniti del problema possono essere inserite facilmente nella strategia di ottimizzazione. In questa prima stesura del codice non è sfruttata la possibilità di trattare rumore non stazionario, che, in base ad opportune approssimazioni, può essere un vantaggio dell'approccio IFA sugli algoritmi proposti in precedenza per la separazione cieca di sorgenti [2, 3]. Il codice qui presentato non ha limitazioni sul numero massimo di sorgenti e canali di misura da trattare (salvo, ovviamente, limitazioni di memoria e di tempo di elaborazione) e consente di variare con estrema facilità i parametri del modello che devono venire considerati incogniti, i valori dei parametri da considerare noti e la stima iniziale della soluzione. Anche lo schema di annealing può essere facilmente modificato.

1. Introduzione: descrizione dell'algoritmo

Richiami alle generalità e possibili applicazioni del problema della separazione cieca di segnali sono ormai abbastanza comuni nella letteratura dell'elaborazione di segnali e delle reti neurali. Un'introduzione al problema si trova in [3].
In generale, il modello della combinazione istantanea e rumorosa di segnali è il seguente:

$$\mathbf{y} = \mathbf{H}\mathbf{x} + \mathbf{n} \qquad (1)$$

in cui **y** è un vettore m-dimensionale formato da combinazioni lineari rumorose degli elementi di un vettore n-dimensionale **x**. Sia **x** sia **y** sono anche funzioni vettoriali di una o più variabili (tempo, spazio, ecc.), ma il valore di **y** in ogni punto, a parte il vettore rumore additivo **n**, è dato da una combinazione lineare degli elementi di **x** nello stesso punto (mistura istantanea). Ciò vuol dire che l'operatore **H** sarà di tipo moltiplicativo, e sarà quindi rappresentato da una "matrice di mixing" di dimensioni $m \times n$. L'equazione (1) vale dunque singolarmente per ogni valore delle variabili da cui dipendono **x** e **y**. Il problema della separazione cieca consiste nel ricavare **x** e **H**, nota una collezione di valori di **y** ed eventualmente le statistiche del rumore additivo **n**, indipendente dal segnale. Naturalmente, il problema così come qui posto è sottodeterminato, se non sono specificate alcune proprietà delle soluzioni. Il principio

ICA (da "Independent component analysis") stabilisce che, se le componenti di **x** sono statisticamente indipendenti tra di loro e (eccetto al più una) hanno statistiche non gaussiane, il problema ammette una soluzione unica a meno di fattori di scala e permutazioni tra le componenti di **x**. Considerazioni più generali a proposito di questa indeterminazione residua si trovano in [4]. Tutti gli algoritmi che discendono dal principio ICA assumono l'indipendenza delle sorgenti, ma fanno anche assunzioni implicite sulle loro distribuzioni statistiche, di fatto stabilendo la forma delle densità di probabilità degli elementi di **x**. La maggior parte di questi algoritmi sono poi basati su un modello dati che non prevede la presenza del rumore. Queste caratteristiche portano a una separazione spesso insoddisfacente, specie quando i dati sono molto rumorosi.
Al fine di ovviare a questo inconveniente, l'approccio di Independent factor analysis (IFA) [1] prevede un modello in cui **n** è un vettore gaussiano con matrice di covarianza diagonale $\mathbf{\Lambda}$, e ogni elemento di **x** ha una distribuzione modellata come una mistura di gaussiane, con pesi, medie e varianze incognite. L'apprendimento del modello dati nel caso IFA non consiste quindi nella sola stima di **H**, come avveniva nell'approccio ICA, ma anche nella valutazione dei parametri relativi alle misture di gaussiane e delle varianze del vettore di rumore. Traducendo in formule, la densità di probabilità del generico elemento di **x** è data da

$$p(x_i) = \sum_{q_i=1}^{Q_i} w_{i,q_i} G(x_i - \mu_{i,q_i}, \nu_{i,q_i}) \tag{2}$$

cioè dalla combinazione di Q_i distribuzioni gaussiane con medie $\mu_{i,qi}$ e varianze $\nu_{i,qi}$ pesate con i coefficienti $w_{i,qi}$, la cui somma, per la normalizzazione, deve essere uguale a 1. L'estrazione di un valore x_i da questa distribuzione può anche essere vista come l'estrazione di un intero q_i compreso tra 1 e Q_i, con probabilità $w_{i,qi}$, seguita dall'estrazione di un valore dalla q_i-esima gaussiana. Le n sorgenti x_i sono poi combinate, secondo la (1), per generare le m osservazioni y_i. In generale le incognite del problema sono quindi gli mn elementi di **H**, le m varianze del rumore, e $3Q_i$ parametri per ogni x_i. In totale si hanno quindi al massimo $m(n+1) + 3\sum_{i=1,n} Q_i$ parametri incogniti. Il fatto che queste distribuzioni siano espresse in forma di misture di gaussiane consente di calcolare analiticamente la funzione di verosimiglianza, ovvero la densità di probabilità di **y** dato l'insieme dei parametri incogniti, che raccogliamo nel vettore generalizzato $\mathbf{W}=[\mathbf{H}, (w_{i,qi}, \mu_{i,qi}, \nu_{i,qi})\ \forall(i, q_i), \mathbf{\Lambda}]$. Indicando quindi con **q** il vettore le cui componenti sono gli indici delle gaussiane attive per ogni sorgente, la probabilità che **q** assuma un certo valore è data dal prodotto dei pesi relativi al valore di ognuna delle componenti:

$$p(\mathbf{q}) = \prod_{i=1}^{n} p_i(q_i) = \prod_{i=1}^{n} w_{i,q_i} \tag{3}$$

La probabilità di **y** dato un particolare valore di **q** sarà data da una gaussiana m-dimensionale:

$$p(\mathbf{y}|\mathbf{q}) = G(\mathbf{y} - \mathbf{H}\mu_{\mathbf{q}}, \mathbf{H}\mathbf{V}_{\mathbf{q}}\mathbf{H}^{\mathbf{T}} + \mathbf{\Lambda}) \tag{4}$$

in cui $\mu_{\mathbf{q}}$ è il vettore delle medie dei fattori gaussiani individuati dal vettore **q** e $\mathbf{V}_{\mathbf{q}}$ è la matrice diagonale delle relative varianze. La densità di probabilità di **y** dato **W** sarà data dalla somma su tutti i possibili valori di **q** delle densità in (4) moltiplicate per le probabilità in (3):

$$p(\mathbf{y}|\mathbf{W}) = \sum_{\mathbf{q}} p(\mathbf{q})p(\mathbf{y}|\mathbf{q}) \tag{5}$$

Ricordiamo che **q** è un vettore di *n* elementi, l'*i*-esimo dei quali può assumere Q_i valori, quindi il numero di addendi nella somma in (5) è dato dalla produttoria dei valori Q_i. Ad esempio, se **x** è composto da 3 elementi, e ognuno di questi ha densità modellata dalla combinazione di 5 gaussiane, il vettore **q** potrà assumere $5^3=125$ diverse configurazioni.
L'apprendimento del modello dati avviene minimizzando da divergenza di Kullback-Leibler tra la densità calcolata $p(\mathbf{y}|\mathbf{W})$ e quella, $p_o(\mathbf{y})$, valutata in base ai dati **y** disponibili (proporzionale all'istogramma *m*-dimensionale dei dati). L'espressione della divergenza da minimizzare è la seguente:

$$\mathcal{E}(\mathbf{W}) = \int p_o(\mathbf{y}) \log \frac{p_o(\mathbf{y})}{p(\mathbf{y}|\mathbf{W})} d\mathbf{y} = -E(\log p(\mathbf{y}|\mathbf{W})) - H_{p_o} \tag{6}$$

in cui *H* rappresenta l'entropia, e il simbolo *E*() indica l'operazione di media sui valori di **y** disponibili. Questa quantità è limitata inferiormente dallo zero, che viene raggiunto se e solo se è $p_o(\mathbf{y})=p(\mathbf{y}|\mathbf{W})$. La sua minimizzazione per mezzo di un algoritmo di annealing simulato richiede solo la capacità di calcolare la divergenza per ogni **y** e per ogni **W**. L'entropia dei dati non dipende ovviamente da **W**, quindi la sola parte da minimizzare rimane l'aspettazione del logaritmo di $p(\mathbf{y}|\mathbf{W})$. L'operazione di media sulle **y** disponibili comporta il calcolo dell'argomento dell'aspettazione per ogni punto in cui **y** è nota, attraverso le formule (4) e (5). Come si vede dalla (4), è possibile inserire nel problema una componente di rumore non stazionaria. Infatti, esplicitando la dipendenza di **y** da una generica variabile *t*, la (4) diventa

$$p(\mathbf{y}(t)|\mathbf{q}) = G(\mathbf{y}(t) - \mathbf{H}\mu_{\mathbf{q}}, \mathbf{H}\mathbf{V}_{\mathbf{q}}\mathbf{H}^{\mathbf{T}} + \mathbf{\Lambda}(t)) \tag{7}$$

Se si hanno a disposizione i valori di **y** per un numero *N* di valori della variabile *t*, si può quindi calcolare l'aspettazione come

$$E(\log p(\mathbf{y}|\mathbf{W})) = \frac{1}{N} \sum_{t=1}^{N} \log p(\mathbf{y}(t)|\mathbf{W}(t)) \tag{8}$$

che contiene in qualche modo la dipendenza del rumore dai singoli punti *t*. È comunque da notare che questa aspettazione è solo un'espressione approssimata dell'integrale in (6). A rigore bisognerebbe parlare di una $\mathcal{E}(\mathbf{W}; t)$, ma non saremmo in grado di calcolare questa quantità perché per ogni *t* abbiamo un solo valore disponibile di **y** e non disponiamo della vera densità di probabilità in quel punto, poiché la nostra p_o deriva da una statistica su tutti i valori di *t*, e assumere che sia in ogni punto uguale alla densità delle variabili osservate equivale ad aver assunto implicitamente che il processo di rumore sia stazionario. L'uso dell'approssimazione (8) si è comunque dimostrato sperimentalmente valido [2].

L'implementazione qui presentata ammette solo un modello dati con rumore uniforme, ovvero la matrice $\mathbf{\Lambda}$ non dipende da t. Per questo caso si è utilizzata una strategia diversa per il calcolo della divergenza. L'integrale

$$-\int p_o(\mathbf{y}) \log p(\mathbf{y}|\mathbf{W}) d\mathbf{y} \tag{9}$$

è stato calcolato approssimando la $p_o(\mathbf{y})$ con l'istogramma normalizzato delle $\mathbf{y}$ disponibili. È stato stabilito un reticolo di discretizzazione m-dimensionale di passo $\Delta\mathbf{y}$ del dominio $\mathbf{y}$, e ad ogni suo nodo è stato assegnato il valore della frequenza relativa dei valori di $\mathbf{y}$ ricadenti nella rispettiva cella. Tale valore approssima la probabilità che un particolare $\mathbf{y}$ appartenga a quella cella; per calcolare il valore corrispondente della probabilità basata sulla $p(\mathbf{y}|\mathbf{W})$, si calcola il valore di $p(\mathbf{y}|\mathbf{W})$ dato dalla (5) in corrispondenza del centro della cella e si moltiplica per il volume di $\Delta\mathbf{y}$, di seguito indicato con $|\Delta\mathbf{y}|$. La funzione (6) è così approssimata dalla seguente formula:

$$\mathcal{E}(\mathbf{W}) = \sum_i p_o(\mathbf{y}_i) \log(p_o(\mathbf{y}_i)) - \sum_i p_o(\mathbf{y}_i) \log(p(\mathbf{y}_i|\mathbf{W})|\Delta\mathbf{y}|) \tag{10}$$

dove i è un indice che corre su tutte le celle su cui l'istogramma normalizzato p_o è non nullo e $\mathbf{y}_i$ sono le coordinate del centro della i-esima cella. Un vantaggio di questo tipo di approccio è che il calcolo va fatto sul numero di celle in cui l'istogramma è diverso da zero, indipendentemente dal numero N di dati a disposizione. Un ulteriore vantaggio è che la funzione descritta dalla (6) può essere effettivamente approssimata nel suo vero valore, evitando i problemi legati al calcolo di entropie differenziali, e quindi essendo sicuri che il minimo della funzione obiettivo è in ogni caso lo zero. La prima sommatoria in (10) può ovviamente essere calcolata una volta per tutte sulla sola base dei dati a disposizione.
L'algoritmo di annealing simulato prevede un numero fisso di tentativi di aggiornamento per ogni temperatura e un numero fisso di iterazioni, con temperatura che diminuisce esponenzialmente al crescere del numero di iterazioni compiute. La temperatura iniziale e il coefficiente di raffreddamento sono fissi ma possono essere facilmente modificati sulla base dei risultati dei test di funzionamento. Lo stesso vale per la legge di raffreddamento, che potrà essere resa se necessario logaritmica. Una successiva versione del codice potrà anche prevedere un controllo da programma di tutti i parametri di annealing.

2. Descrizione del codice

In questo paragrafo si descrive l'organizzazione e il funzionamento di IFATEST. Per comodità di lettura il codice è stato diviso in più parti, comunque riportate nell'ordine in cui appaiono nel codice effettivamente compilato e numerate come Listato 1, Listato 2, ecc. Altre figure o esempi di file di ingresso-uscita sono numerate come Figura 1, Figura 2, ecc.

2.1 Parametri e dati in ingresso

La maggior parte dei pochi parametri utilizzati dal codice sono relativi al modello IFA da considerare, e sono definiti in un modulo chiamato `parametri` di cui si riporta un esempio in Figura 1.

```
module parametri
integer, parameter  :: NSORGENTI=2
integer, parameter  :: NCANALI=2
integer, parameter  :: NRIGHE=256, NCOLONNE=256
integer, parameter  :: NBIN=100
integer, parameter  :: NFATTORI=3
real,    parameter  :: PI=3.141592654,DPI=6.283185307
end module parametri
```

Fig. 1: Modulo `parametri.f90`

I parametri definiti sono:

- `NSORGENTI`: il numero di elementi costituenti il vettore **x** delle sorgenti (al Paragrafo 1 questo numero è indicato con *n*).

- `NCANALI`: il numero di elementi costituenti il vettore **y** degli osservati (al Paragrafo 1 questo numero è indicato con *m*).

- `NRIGHE, NCOLONNE`: in questa versione il codice prevede di trattare dati sotto forma di mappe bidimensionali, di cui questi parametri rappresentano, rispettivamente, il numero di righe e di colonne. Questa scelta è fatta solo per comodità, in quanto le misture sono istantanee e la struttura bidimensionale delle sequenze di dati non ha alcun interesse. Il prodotto `NRIGHE*NCOLONNE` rappresenterà il numero totale di punti dati, che al Paragrafo 1 è indicato con *N*, ed è assegnato in esecuzione alla variabile `npixel` (vedi listati 1 e 3). Questo valore è ovviamente pari al numero di pixel costituenti le mappe **y** date e le mappe **x** da valutare una volta stimate tutte le incognite del modello.

- `NBIN`: il numero di celle di discretizzazione per ognuna delle *m* dimensioni dell'istogramma dati. Tale istogramma sarà quindi contenuto in un ipercubo *m*-dimensionale con un numero totale di `NBIN**NCANALI` celle. Dal momento che i range di variazione delle diverse componenti di **y** non sono tutti uguali, il fatto di discretizzare ognuno di questi range sullo stesso numero `NBIN` di celle implica che le celle non saranno esse stesse degli ipercubi. Naturalmente questa è solo la scelta più immediata, e questa strategia può essere facilmente sostituita con una migliore e/o più efficiente, specialmente nei casi in cui gli osservati hanno dimensione elevata (con la scelta `NBIN=100` basta avere una **y** di dimensione 5 per avere un istogramma con 10^{10} celle, ovviamente intrattabile).

- `NFATTORI`: il numero Q_i (Vedi Paragrafo 1) di fattori gaussiani con cui si modellano le densità di probabilità degli elementi del vettore **x** dei segnali sorgente. In questa versione si è scelto di fare Q_i uguale per tutte le sorgenti, ma, a parte che questo non comporta una vera perdita di generalità, il codice può essere facilmente modificato per attribuire un numero di fattori diverso per ogni sorgente.

- `PI, DPI`: come è ovvio dal listato, i valori di π e 2π.

L'unico altro valore messo a parametro (vedi Listato 1) è il numero di iterazioni dell'algoritmo di annealing, `NITERAZ`.

I dati del problema (vedi Listato 1) sono

- `NCANALI` vettori reali di lunghezza `npixel`, che dovranno essere contenuti in altrettanti file con nomi `y1.txt, y2.txt, y3.txt,...` Questi file vengono letti in un unico array di dimensioni `(NCANALI,npixel)` e nome `osservati` (Listato 3).

- la matrice diagonale delle deviazioni standard del rumore, assunto uniforme (indicata con $\mathbf{\Lambda}$ al Paragrafo 1). Le deviazioni standard dovranno essere contenute in un file di `NCANALI` valori reali, di nome `noisdev`. Questo file viene letto in un vettore reale di dimensione `NCANALI` e nome `sdev` (Listato 3). Notare che in questa versione si assume la conoscenza delle statistiche del rumore, mentre nell'algoritmo originale la matrice delle deviazioni standard era considerata un'altra delle incognite del problema. Anche in questo caso sarebbe facile apportare le modifiche necessarie a trattare il rumore come incognita.

È previsto che le stime iniziali di tutte le incognite del problema siano date come input da file al programma. Questo algoritmo consente di specificare quali sono le effettive incognite e di stabilire il valore di quei parametri che dovessero essere noti, per cui tutte le incognite vengono stabilite nel loro valore noto o in stima iniziale e il programma lascerà invariate quelle note e aggiornerà le incognite secondo quanto specificato nel file `maschera.txt` (vedi più avanti). I file contenenti le stime iniziali sono i seguenti (Listati 1 e 3):

- `mixing.txt`: stima iniziale della matrice **H**. Contiene, ordinati per righe, gli elementi della matrice di mescolamento. Viene letto nell'array reale `mixing`, di dimensione `(NCANALI,NSORGENTI)`.

- `pesi.txt`: stime iniziali dei pesi w_{i,q_i}. È una matrice di dimensione `(NSORGENTI,NFATTORI)` ogni riga della quale contiene i pesi dei fattori gaussiani di una sorgente, definiti al paragrafo precedente. Viene letto nell'array reale `wmat`.

- `medie.txt`: stime iniziali delle medie dei fattori gaussiani, disposte come per i valori in `pesi.txt`. Viene letto nell'array reale `mumat`.

- `varianze.txt`: stime iniziali varianze fattori gaussiani, disposte come per il valori in `pesi.txt`. Viene letto nell'array reale `numat`.

Come accennato sopra, il numero totale di incognite e lo status di parametro fisso o di incognita di ogni singola variabile sono specificati nel file di ingresso denominato `maschera.txt`. In prima linea di questo file è contenuto un intero rappresentante il numero totale di incognite, letto nella variabile intera `nincognite`. I record successivi sono letti nell'array reale `maschera`, di dimensioni `(NCANALI*NSORGENTI+3*NSORGENTI*NFATTORI,5)` in cui il numero di righe eguaglia il numero massimo di incognite indicato al Paragrafo 1, $mn + 3\sum_{i=1,n} Q_i$ (salvo le varianze del rumore, che abbiamo assunto note). Ovviamente, solo le prime

`nincognite` righe di questo array saranno occupate. Ognuna delle righe di `maschera` contiene informazioni su una singola incognita. I suoi 5 elementi reali rappresentano, nell'ordine,

1) Il tipo di incognita: 1 ---> si riferisce a un elemento di `mixing` (**H**)
2 ---> si riferisce a un elemento di `wmat` (w_{iqi})
3 ---> si riferisce a un elemento di `mumat` (μ_{iqi})
4 ---> si riferisce a un elemento di `numat` (ν_{iqi}).

2)L'indice di riga dell'incognita all'interno della matrice di cui fa parte.

3) L'indice di colonna dell'incognita all'interno della matrice di cui fa parte.

4) L'estremo inferiore di un intervallo contenente il valore vero dell'incognita.

5) L'estremo superiore di un intervallo contenente il valore vero dell'incognita.

L'algoritmo di stima lavora ciclicamente su tutte le incognite e le indirizza assegnando di volta in volta a un pointer di nome `incognita` gli array target `mixing`, `wmat`, `mumat`, `numat` (Listato 1). In Figura 2 è mostrato un esempio per il file `maschera.txt` da cui si vede che viene richiesto di operare su un totale di 12 incognite: due elementi della matrice di mixing, tre della matrice dei pesi e della matrice delle medie, e quattro elementi della matrice delle varianze.

```
12
1.  2.  1.  0.      3.
1.  2.  2.  0.      3.
2.  2.  1.  0.      1.
2.  2.  2.  0.      1.
2.  2.  3.  0.      1.
3.  2.  1.  0.      0.1
3.  2.  2.  0.      0.1
3.  2.  3.  0.      0.1
4.  1.  1.  1.E-07  0.01
4.  2.  1.  1.E-07  1.E-04
4.  2.  2.  1.E-07  1.E-04
4.  2.  3.  1.E-07  1.E-04
```

Fig. 2: Esempio di file `maschera.txt`

Listato 1: Dichiarazione variabili programma principale

```
!     IFATEST v0.0
!
!  Separazione delle componenti con algoritmo IFA (Attias 1999) ed
!  apprendimento del modello con simulated annealing.
!  Mappe dati e matrice covarianza rumore lette da file.
!  Tutto il resto calcolato localmente.
!
!  Versione con calcolo entropia discreta 19/11/2002
!  RUMORE STAZIONARIO SPECIFICATO PER OGNI CANALE NELLA MATRICE sdev
!
!  compilazione: f95 parametri.f90
!                f95 siman01.f90 prodmat.o SGEDI.o SGEFA.o obj2.f90
!
```

```
!
      program siman
        use parametri
        implicit none
        real, dimension(NCANALI,NRIGHE*NCOLONNE)  :: osservati
        real, dimension(NCANALI)                  :: sdev
        real, dimension(NCANALI,NRIGHE*NCOLONNE)  :: ossdec
        real, target, dimension(NCANALI,NSORGENTI):: mixing
        real, dimension(NBIN**NCANALI)            :: istogramma
        real, dimension(NCANALI)                  :: &
              binsize,minimi,massimi
        real, target,dimension(NSORGENTI,NFATTORI):: wmat,mumat,numat
        real, pointer, dimension(:,:)             :: incognita
        real, dimension(NFATTORI)                 :: pesiprec
        real, dimension(NCANALI*NSORGENTI+3*NSORGENTI*NFATTORI,5):: &
              maschera
        real, dimension(NBIN**NCANALI,NCANALI)    :: ygrid
        real                                      :: unif, t
        real                                      :: entropia0
        real                                      :: epyw,binvol
        real                                      :: kldiv,kldiv0
        real                                      :: valprec,valnew
        integer, parameter                        :: NITERAZ=2000
        integer                                   :: nagg,nac1,nac2
        integer                                   :: nincognite
        integer                                   :: i,j,it,i1
        integer                                   :: &
                 ipix,ican,isor,npixel
        integer                                   :: &
                 decimazione,pixdec
        integer                                   :: indice,nstati
        integer                                   :: iflag
        integer                                   :: imarkov,nmarkov
        integer                                   :: indne0,k
        integer, dimension(NBIN**NCANALI)         :: indici
!
        character(len=6)                          :: ingresso
!
```

Listato 2: Interfaccia sottoprogrammi (programma principale)

```
        interface
           function gauss(media, devstand)
             real         :: media, devstand
             real         :: gauss, unif
             real, parameter:: DPI=6.283185307
           end function gauss
!
     subroutine obj2(epyw,mixing,wmat,mumat,numat,istogramma,binvol,&
           nstati,indne0,indici,ygrid,Lambda)
        use parametri
        implicit none
        integer, intent(in)              :: nstati,indne0
        integer, dimension(indne0), intent(in)        :: indici
        integer                          :: &
                 ipix,indiceq,isor,info,ican
        integer, dimension(NSORGENTI)    :: q
        integer, dimension(NCANALI)      :: ipvt
        real, dimension(NSORGENTI,NFATTORI),intent(in)::&
                                    wmat,mumat,numat
        real, dimension(NCANALI,NSORGENTI),intent(in) ::&
                                    mixing
```

```
        real, dimension(NCANALI), intent(in)          :: Lambda
        real, dimension(NBIN**NCANALI), intent(in)    :: &
               istogramma
        real, intent(out)                :: epyw
        real, dimension(NCANALI)         :: hmu,argv
        real, dimension(NCANALI)         :: work
        real                             :: pyw,w,arg
        real, dimension(2)               :: det
        real, dimension(NSORGENTI)       :: muvet,nuvet
        real, dimension(NSORGENTI,NCANALI)            :: htv
        real, dimension(NCANALI,NCANALI) :: hht
        real, dimension(NCANALI)         :: hhtinv
        real, intent(in)                 :: binvol
        real, dimension(NBIN**NCANALI,NCANALI), intent(in):: ygrid
     end subroutine obj2
!
     end interface
!
```

Listato 3: Ingresso dati programma principale

```
        npixel=NRIGHE*NCOLONNE
!
      ingresso='y0.txt'
!
! lettura stime iniziali: matrice di mixing
!
        open(11, file='mixing.txt', status='old')
        read(11,*)((mixing(i,j),j=1,NSORGENTI),i=1,NCANALI)
        close(11)
!
!                   matrice pesi
        open(11, file='pesi.txt', status='old')
        read(11,*)((wmat(i,j),j=1,NFATTORI),i=1,NSORGENTI)
        close(11)
!
!                   matrice medie sorgenti
        open(11, file='medie.txt', status='old')
        read(11,*)((mumat(i,j),j=1,NFATTORI),i=1,NSORGENTI)
        close(11)
!
!                   matrice varianze sorgenti
        open(11, file='varianze.txt', status='old')
        read(11,*)((numat(i,j),j=1,NFATTORI),i=1,NSORGENTI)
        close(11)
!
!
! lettura maschera incognite
!
        open(11, file='maschera.txt',status='old')
        read(11,*)nincognite
        do i=1,nincognite
           read(11,*)maschera(i,:)
        end do
        close(11)
!
!
! lettura mappe osservati
!
        do ican=1,NCANALI
           write(ingresso(2:2),200)ican
```

```
           open(11, file=ingresso, status='old')
           read(11,*)(osservati(ican,j),j=1,npixel)
           close(11)
        end do
200   format(I1)
!
! lettura mappe deviazioni standard rumore
!
        open(10, file='noisdev',status='old')
        read(10,*)(sdev(ican),ican=1,NCANALI)
        close(10)
!
!  Nella matrice delle deviazioni standard si mettono
!  le varianze
!
        sdev=sdev**2
!
!  Fine lettura
```

2.2 Decimazione dati

Si suppone che il modello dati (1) sia valido per ogni punto *t* considerato, cioè che la matrice **H** sia costante in tutto il campo di misura considerato. Per questo motivo, e per il fatto che si considera una miscelazione istantanea dei valori delle sorgenti, non ha importanza trattare tutto l'insieme dei dati osservati ma solo un campione significativo per il calcolo dell'istogramma. La sezione del codice qui descritta provvede a prelevare casualmente una frazione dell'insieme dei dati e trasferirla dall'array `osservati` all'array `ossdec`, che viene a contenere un numero `pixdec` di dati, pari a `npixel` diviso per un intero `decimazione` specificato da terminale. Al Listato 4 è mostrata la procedura seguita. La funzione `random_number` genera un numero reale uniformemente distribuito tra 0 e 1. Questo accorgimento è utile per ridurre il numero di dati su cui si lavora e quindi risparmiare sul tempo di elaborazione. Non è particolarmente utile nel caso di questa versione del codice poiché, come già accennato, l'algoritmo non lavora direttamente sull'insieme dei dati ma solo sui centri delle celle in cui l'istogramma dei dati è non nullo. Nel caso in cui si dovesse usare la (8) piuttosto che la (10), la situazione sarebbe diversa. Se si pone `decimazione=0`, `npixel` e `osservati` vengono semplicemente ricopiati così come sono in `pixdec` e `ossdec`, che sono le variabili su cui si va effettivamente a lavorare.

Listato 4: Decimazione dati (programma principale)

```
!
!  decimazione dati (random)
!
  write(*,*)'Fattore di decimazione (0 = no decimazione)'
  read(*,*)decimazione
  if(decimazione/=0)then
     pixdec=npixel/decimazione
     do ican=1,NCANALI
        do ipix=1,pixdec
           call random_number(unif)
           ossdec(ican,ipix)=osservati(ican,int((npixel-1)*unif)+1)
        end do
     end do
  else
```

```
      pixdec=npixel
      ossdec=osservati
   end if
!
```

2.3 Istogramma ed entropia degli osservati

Per calcolare la prima sommatoria nella formula (10), cioè l'opposto dell'entropia dei dati, è necessario calcolare l'istogramma normalizzato $p_o(\mathbf{y})$. Questa è una funzione di m variabili, con m non fissato, per cui non può essere mappata su un array di dimensioni fissate. L'array `istogramma` è quindi un vettore monodimensionale con un numero di elementi uguale al numero totale delle celle dell'istogramma. Ogni cella è indirizzata in base agli indici m-dimensionali seguendo la procedura descritta qui di seguito per il caso generale in cui ogni dimensione dell'istogramma può avere un numero diverso di celle.
Sia

$$(i_1, i_2, \ldots i_m) \tag{11}$$

l'indice m-dimensionale dell'istogramma relativo al valore $(y_1, y_2, \ldots y_m)$. Ogni i_k può assumere valori interi tra 1 e N_k, ognuno relativo a una cella di dimensione Δ_k, pari al rapporto tra il range di y_k e il numero N_k di celle stabilito per la k-esima dimensione dell'istogramma. Sarà

$$i_k = \left[\frac{y_k - \min(y_k)}{\Delta_k} \right] + 1 \tag{12}$$

in cui la parentesi quadra indica l'operatore parte intera, salvo per il caso in cui y_k assume il suo valore massimo, in cui si stabilisce $i_k = N_k$. I valori dell'indice vettoriale (11), calcolati secondo la (12), possono essere ordinati secondo un unico indice *ind*, dato da

$$ind = 1 + \sum_{k=1}^{m} \left\{ (i_k - 1) \prod_{l=1}^{k-1} N_l \right\} \tag{13}$$

in cui si intende che per $k = 1$ la produttoria in parentesi graffa vale 1. L'indice *ind* assume valori che vanno da 1 alla produttoria tra tutti gli N_k. Se, come nel nostro caso, il numero di celle (o *bin*) è lo stesso per ogni dimensione, ossia $N_l = N_{bin}$ per ogni l, la (13) si riduce alla seguente

$$ind = 1 + \sum_{k=1}^{m} (i_k - 1) N_{bin}^{k-1} \tag{14}$$

e *ind* varia tra 1 e N_{bin}^m. Vediamo adesso come, noti N_{bin} e m, si può risalire da *ind* all'indice m-dimensionale del tipo (11) che lo ha generato. Dalla (14) si ottiene:

$$\frac{ind - 1}{N_{bin}^{m-1}} = \sum_{k=1}^{m} N_{bin}^{k-m} (i_k - 1) = (i_m - 1) + \sum_{k=1}^{m-1} N_{bin}^{k-m} (i_k - 1) \tag{15}$$

Notiamo adesso che ogni addendo $N_{bin}^{k-m}(i_k - 1)$ è minore o uguale a $N_{bin}^{k-m}(N_{bin} - 1)$, per cui, posto $l = m - k$, si ottiene:

$$\frac{ind - 1}{N_{bin}^{m-1}} \le (i_m - 1) + (N_{bin} - 1)\sum_{l=1}^{m-1}\left(\frac{1}{N_{bin}}\right)^l = (i_m - 1) + 1 - \left(\frac{1}{N_{bin}}\right)^{m-1} \qquad (16)$$

Il che vuol dire che la parte intera del membro sinistro di (16) è pari a $(i_m - 1)$. Abbiamo così ricavato l'm-esimo indice. Una volta fatto questo, di nuovo dalla (14) otteniamo

$$\frac{(ind - 1) - N_{bin}^{m-1}(i_m - 1)}{N_{bin}^{m-2}} = \sum_{k=1}^{m-1} N_{bin}^{k-m+1}(i_k - 1) = (i_{m-1} - 1) + \sum_{k=1}^{m-2} N_{bin}^{k-m+1}(i_k - 1) \qquad (17)$$

Analogamente a quanto fatto in precedenza, da questa relazione possiamo ottenere l'indice $(m - 1)$-esimo e quindi, ricorsivamente, tutti gli altri indici. In generale, una volta calcolati gli indici i_m, i_{m-1}, ... i_{k+1}, l'indice i_k sarà dato da

$$i_k = \left[\frac{(ind - 1) - \sum_{l=k+1}^{m} N_{bin}^{l-1}(i_l - 1)}{N_{bin}^{k-1}}\right] + 1 \qquad (18)$$

in cui è evidente la possibilità di implementazione ricorsiva. Le parentesi quadre, come prima, rappresentano l'operatore parte intera, e per $k = m$ si intende che la sommatoria valga zero.
Al Listato 5 è riportata le sezione di codice che calcola l'istogramma indirizzandolo secondo la (14), lo normalizza e calcola l'entropia dei dati implementando la prima sommatoria in (10). Nel fare quest'ultima operazione, valuta anche il numero di celle in cui l'istogramma è non nullo (nella variabile intera `indne0`) salvando i relativi indici nel vettore `indici`, per poi risalire usando la (18) agli indici m-dimensionali e porre in un array chiamato `ygrid` le coordinate dei centri delle celle individuate. Gli array `indici` e `ygrid` serviranno al calcolo della seconda sommatoria in (10) durante il processo di annealing (vedi sottoparagrafo *2.8*).
Per facilitare la lettura del listato, si specificano qui di seguito le corrispondenze tra le variabili e gli array del codice e i simboli della notazione qui utilizzata.

- `istogramma(NBIN**NCANALI)` --> versione discreta della densità $p_o(\mathbf{y})$ ordinata in vettore monodimensionale secondo la (14).

- `minimi(NCANALI)` --> vettore contenente i minimi valori delle diverse componenti di **y**.

- `massimi(NCANALI)` --> vettore contenente i massimi valori delle diverse componenti di **y**.

- `binsize(NCANALI)` --> vettore contenente le dimensioni Δ_k delle celle dell'istogramma per ogni sua componente, ottenute dividendo il relativo range di variazione per il numero di celle N_{bin}.

- `binvol` --> (|$\Delta\mathbf{y}$| in (10)) volume della singola cella dell'istogramma, dato dalla produttoria di tutti i Δ_k.

- `indice` --> (*ind*) utilizzato prima per indirizzare il vettore `istogramma`, e calcolato usando (12) e (14), poi come elemento corrente i_k dell'indice m-dimensionale in (11), utilizzato nel calcolo ricorsivo della (18).

- `indne0` --> accumulatore il cui valore finale è pari al numero di bin non vuoti dell'istogramma.

- `indici(NBIN**NCANALI)` --> vettore di puntatori contenente i valori degli indici *ind* relativi a bin non vuoti dell'istogramma.

- `ygrid(NBIN**NCANALI,NCANALI)` --> array contenente i valori di **y** relativi ai centri dei bin dell'istogramma indirizzati dal vettore `indici`.

- `entropia0` --> il valore $-H_{po}$ (opposto dell'entropia degli osservati) che appare nella (6).

Listato 5: Istogramma ed entropia dati (programma principale)

```
!
!  Calcolo entropia degli osservati
!
        istogramma=0.
!
!            calcolo dimensioni bin istogramma
        do ican=1,NCANALI
           minimi(ican)=minval(ossdec(ican,:))
           massimi(ican)=maxval(ossdec(ican,:))
           binsize(ican)=(massimi(ican)-minimi(ican))/NBIN
        end do
!
!            calcolo volume bin
        binvol=product(binsize)
!
!             calcolo istogramma
        do ipix=1,pixdec
           indice=1
           do ican=1,NCANALI
              indice=indice+min(int((ossdec(ican,ipix)-&
                  minimi(ican))/binsize(ican)),NBIN-1)*NBIN**(ican-1)
           end do
           istogramma(indice)=istogramma(indice)+1.
        end do
        istogramma=istogramma/pixdec
!
!              calcolo entropia dati (indne0=numero bin non vuoti)
!              (ygrid=valori osservati al centro dei bin non vuoti)
!              (indici=indici istogramma con valori diversi da zero)
!
        entropia0=0.
        indne0=0
        do i=1,NBIN**NCANALI
           if(istogramma(i).gt.0.)then
              indne0=indne0+1
```

```
               indici(indne0)=i
               i1=i-1
               do k=NCANALI,1,-1
                  indice=int(i1/NBIN**(k-1))+1
                  ygrid(indne0,k)=minimi(k)-&
                           binsize(k)/2.+indice*binsize(k)
                  i1=i1-NBIN**(k-1)*(indice-1)
               end do
               entropia0=entropia0+istogramma(i)*log(istogramma(i))
            end if
         end do
!
!         in realta' e' "-entropia"
         print *,'entropia dati =',entropia0
```

2.4 Valutazione iniziale della divergenza di Kullback-Leibler

Una volta trovato il valore dell'entropia degli osservati, per inizializzare il processo di annealing simulato occorre valutare $E(\log p(\mathbf{y}|\mathbf{W}))$, che appare in (6), per la stima iniziale di **W**. Come detto, tale valore è qui approssimato dalla seconda sommatoria in (10), implementata nella subroutine `obj2`, che sarà descritta in seguito. In questa sezione del codice viene valutato il numero di stati diversi assumibili dal vettore **q**, ognuno dei quali a probabilità data dalla (3). Il numero totale di questi stati è dato, come anticipato al Paragrafo 1, dalla produttoria dei numeri di fattori gaussiani, Q_i, relativi a tutte le sorgenti x_i. Nel nostro caso i Q_i sono stati tutti assunti uguali a `NFATTORI`, per cui il numero di stati distinti di **q** è dato da `NFATTORI**NSORGENTI`. La variabile `nstati`, che viene data in ingresso alla subroutine `obj2`, è pari a questo numero diminuito di 1. Il risultato del calcolo, nella variabile reale `epyw`, viene utilizzato, insieme a `entropia0`, per valutare la divergenza di Kullback-Leibler, nella variabile reale `kldiv`, che viene copiata in una seconda variabile `kldiv0` per l'esigenza di mantenere, durante l'annealing, il valore corrente della funzione obiettivo insieme al valore assunto prima del tentativo di aggiornamento.

Listato 6: Calcolo divergenza iniziale (programma principale)

```
!
!  Calcolo funzione costo divergenza KL
!
         nstati=NFATTORI**NSORGENTI-1
!
!               aspettazione logaritmo densita' calcolata
!
         call obj2(epyw,mixing,wmat,mumat,numat,istogramma,binvol,&
              nstati,indne0,indici,ygrid,sdev)
!
!               divergenza di Kullback-Leibler tra densita' osservata
!                e densita' calcolata
!
         kldiv=-epyw+entropia0
         kldiv0=kldiv
         print *, 'valore aspettazione p(y/W)',epyw
         print *, 'valore iniziale divergenza',kldiv
!!
!  FINE PRIMO CALCOLO DIVERGENZA KL
```

2.5 Stima del modello con annealing simulato

La divergenza di Kullback-Leibler tra la densità dei dati osservati e quella calcolata in base alla stima corrente del modello è certamente non convessa, e viene minimizzata per mezzo di un algoritmo di annealing simulato, riportato al Listato 7. In questa versione, il numero di tentativi di aggiornamento per ogni catena di Markov uniforme, la temperatura iniziale e il numero totale di iterazioni sono fissati da codice, e modificarli richiede una nuova compilazione. Naturalmente questo inconveniente può essere facilmente eliminato in una versione successiva. Lo stesso vale anche per il coefficiente di raffreddamento (qui posto a 0.985) e per la legge di raffreddamento (qui fatta esponenziale rispetto al numero di iterazioni). Il primo blocco di linee di codice dopo l'aggiornamento del valore della temperatura ha il solo scopo di consentire un controllo in esecuzione del processo di minimizzazione. Esso provoca la visulizzazione, per l'iterazione corrente, dei valori del rapporto di accettazione delle transizioni, del rapporto tra il numero di transizioni accettate con aumento di energia e quelle accettate con diminuzione di energia, e del valore corrente dei parametri del modello. I tentativi di aggiornamento sono fatti per singolo parametro incognito. La selezione del parametro da aggiornare viene guidata dall'elemento contenuto nella prima colonna della riga dell'array `maschera` indirizzata dall'indice `iflag`, che assume tutti i valori, da 1 al numero totale di parametri incogniti. La struttura di `maschera` è descritta al sottoparagrafo *2.1* e alla Figura 2. In base al valore di `maschera(iflag,1)`, al puntatore `incognita` viene assegnato uno degli array di parametri (`mixing, wmat, mumat, numat`). Da questo viene estratto l'elemento specificato dal secondo e terzo elemento della riga di `maschera` selezionata e viene copiato nella variabile reale `valprec` che servirà a ripristinare il valore precedente del parametro in caso di non accettazione della transizione proposta. Nel caso in cui l'array puntato sia `wmat`, viene anche estratta l'intera riga cui appartiene il parametro da aggiornare; tale riga viene copiata nel vettore `pesiprec`. Questa operazione è necessaria poiché gli elementi di ogni riga di `wmat` sono vincolati ad avere somma 1 (vedi Paragrafo 1) e l'aggiornamento di uno solo di essi comporta la scalatura di tutti gli altri, effettuata dal gruppo di linee di codice sotto il commento "rinormalizzazione pesi". Il nuovo valore proposto per il parametro da aggiornare viene generato come una variabile gaussiana centrata sul vecchio valore e di deviazione standard pari a un centesimo del range permesso per quel parametro, specificato dal quarto e dal quinto elemento della riga di `maschera` puntata da `iflag`. Per tale generazione viene usata la function `gauss`, il cui funzionamento è descritto al sottoparagrafo *2.7*. Il resto del codice segue la procedura standard dell'annealing simulato, calcolando il nuovo valore della divergenza e aggiornando il parametro se vi è diminuzione di energia oppure aggiornandolo in probabilità con il consueto schema se vi è aumento di energia. L'accettazione della transizione comporta il solo aggiornamento del valore della divergenza e il passaggio al parametro o al ciclo successivo, mentre la non accettazione comporta il ripristino del valore precedente del parametro (o dell'intera riga, se si tratta di `wmat`) e il passaggio al parametro o al ciclo successivo. I controlli da effettuare in esecuzione sulla base delle uscite a terminale comportano che il rapporto di accettazione assuma valori decrescenti, a partire da 1 per temperature alte fino a valori quasi nulli per le temperature finali. Il raggiungimento dell'equilibrio per ogni catena di Markov uniforme è indicato dal valore del rapporto tra le transizioni accettate con aumento e diminuzione di energia, che dovrebbe sempre rimanere attorno all'unità.

Listato 7: Annealing simulato (programma principale)

```
!  CICLO SULLA TEMPERATURA
!
        nmarkov=10          ! (numero passi per catena di Markov)
        t= 1.E-4            ! (valore temperatura iniziale)
        do it=1,NITERAZ     ! (numero valori temperatura)
           t=t*0.985        ! (formula della temperatura corrente)
!
!
           print *, 'iterazione ',it,'temperatura',t
           print *, 'divergenza =',kldiv0
           print *, 'numero transizioni accettate =',nac1+nac2
           print *, 'rapporto acc. =',float(nac1+nac2)/nagg
           print *, 'rapporto acc. aum/dim =',float(nac2)/nac1
           print *,'matrice di mixing'
           do i=1,NCANALI
              print *,mixing(i,:)
           end do
           print *,'matrice pesi'
           do i=1,NSORGENTI
              print *,wmat(i,:)
           end do
           print *,'matrice medie'
           do i=1,NSORGENTI
              print *,mumat(i,:)
           end do
           print *,'matrice varianze'
           do i=1,NSORGENTI
              print *,numat(i,:)
           end do
           print *,''
           print *,'================================================'
           print *,''
!
           nagg=0   !numero transizioni proposte
           nac1=0   !numero transizioni accettate con dim. di energia
           nac2=0   !numero transizioni accettate con aum. di energia
           do imarkov=1,nmarkov
!
!  scelta e ciclo sul parametro da aggiornare
!
              do iflag=1,nincognite
! identificazione parametro
                 if(maschera(iflag,1).eq.1)incognita=>mixing
                 if(maschera(iflag,1).eq.2)then
                    pesiprec=wmat(int(maschera(iflag,2)),:)
                    incognita=>wmat
                 end if
                 if(maschera(iflag,1).eq.3)incognita=>mumat
                 if(maschera(iflag,1).eq.4)incognita=>numat
! generazione nuovo valore
                 valprec=incognita(int(maschera(iflag,2)),&
                      int(maschera(iflag,3)))
                 valnew=gauss(valprec,(maschera(iflag,5)-&
                        maschera(iflag,4))/100.)
! imposizione vincoli sul nuovo valore

if(valnew.lt.maschera(iflag,4))valnew=maschera(iflag,4)

if(valnew.gt.maschera(iflag,5))valnew=maschera(iflag,5)
! tentativo di aggiornamento
                 incognita(int(maschera(iflag,2)),&
```

```
                     int(maschera(iflag,3)))=valnew
                nagg=nagg+1
!      rinormalizzazione pesi
                if(maschera(iflag,1).eq.2)then
                   wmat(int(maschera(iflag,2)),:)=&
                        wmat(int(maschera(iflag,2)),:)/  &
                        sum(wmat(int(maschera(iflag,2)),:))
                end if
!
!      calcolo nuova divergenza
                call obj2(epyw,mixing,wmat,mumat,numat,istogramma,&
                     binvol,nstati,indne0,indici,ygrid,sdev)
                kldiv=-epyw+entropia0

! test di aggiornamento
                if(kldiv.lt.kldiv0)then
!      passato: aggiornamento valore divergenza
                   kldiv0=kldiv
                   nac1=nac1+1
!
                else
!      non passato: aggiornamento in probabilita'
                   call random_number(unif)
!
                   if(unif.gt.exp(-(kldiv-kldiv0)/t))then
!
!          non passato: ripristino valore precedente
                      if(maschera(iflag,1).eq.2)then
                         wmat(int(maschera(iflag,2)),:)=pesiprec
                      else
                         incognita(int(maschera(iflag,2)),&
                              int(maschera(iflag,3)))=valprec
                      end if
                   else
!          passato: aggiornamento valore divergenza
                      kldiv0=kldiv
                      nac2=nac2+1
                   end if   !fine aggiornamento in probabilita'
                end if      !fine aggiornamento
             end do         !fine ciclo nei parametri
          end do            !fine catena a temperatura costante
       end do               !fine ciclo nelle temperature
!
!  FINE ANNEALING SIMULATO
```

2.6 Uscita

I risultati dell'ottimizzazione sono posti in un file di disco di nome `uscita.txt` (in questa versione il nome del file è modificabile solo dopo ricompilazione del codice sorgente). Le quantità registrate sono il numero di iterazioni, il valore finale della divergenza e gli array dei parametri di modello stimati. La relativa sezione di codice è riportata al Listato 8.

Listato 8: Uscita risultati (programma principale)

```
!
!    USCITA RISULTATI
!
```

```
!
        open(10,file='uscita.txt',status='unknown')
        write(10,*)'Iterazioni =',NITERAZ
        write(10,*)'divergenza finale =',kldiv0
        write(10,*)'matrice di mixing stimata'
        do i=1,NCANALI
           write(10,*)mixing(i,:)
        end do
        write(10,*)'matrice pesi fattori'
        do i=1,NSORGENTI
           write(10,*)wmat(i,:)
        end do
        write(10,*)'matrice medie fattori'
        do i=1,NSORGENTI
           write(10,*)mumat(i,:)
        end do
        write(10,*)'matrice varianze fattori'
        do i=1,NSORGENTI
           write(10,*)numat(i,:)
        end do
        close(10)
        end program siman
```

2.7 Generazione di variabili gaussiane

Come visto, l'aggiornamento di ogni singolo parametro all'interno dello ciclo di annealing è generato casualmente con distribuzione gaussiana attorno al valore precedente del parametro. a questo scopo è stato scritto un sottoprogramma di tipo function che sfrutta la subroutine Fortran90 di nome `random_number`, già vista in precedenza. Al Listato 9 è riportato il codice sorgente di questa function, chiamata `gauss(media,devstand)`, che genera una variabile gaussiana di media e deviazione standard specificate. La procedura per la generazione della variabile gaussiana sfrutta la seguente proprietà [5]:

Se x_1 e x_2 sono due variabili uniformemente distribuite tra 0 e 1, la variabile

$$g = \sqrt{-2\ln x_1}\cos 2\pi x_2 \tag{19}$$

ha distribuzione gaussiana con media 0 e deviazione standard 1.
L'interfaccia di `gauss` verso il programma principale è riportata al Listato 2.

Listato 9: Function `gauss(media, devstand)`

```
!
        function gauss(media, devstand)
        real         :: media, devstand
        real         :: gauss, unif
        real, parameter:: DPI=6.283185307
        call random_number(unif)
        gauss=sqrt(-2.*log(unif))
        call random_number(unif)
        gauss=gauss*cos(DPI*unif)
        gauss=gauss*devstand+media
        return
```

```
        end function gauss
!
```

2.8 Calcolo di E(log*p*(**y**|**W**))

Il calcolo della seconda sommatoria nella formula (10) viene implementato dalla subroutine di nome `obj2`. L'interfaccia di questa verso il programma principale è riportata al Listato 2. In questo sottoparagrafo se ne descrive il funzionamento. Al Listato 10 è riportata la sezione di dichiarazione delle variabili. Richiamiamo per adesso solo le variabili e gli array facenti parte della lista di ingresso-uscita, rimandando al seguito la descrizione delle rimanenti. La variabile reale in uscita `epyw` conterrà il risultato del calcolo. Seguono gli array in ingresso `mixing`, `wmat`, `mumat`, `numat`, già descritti, che contengono le stime correnti dei parametri del modello. Il calcolo della sommatoria richiede anche la conoscenza dell'istogramma (`istogramma`) degli osservati, come approssimazione della loro densità di probabilità $p_o(\mathbf{y})$. La lista di ingresso contiene poi il volume dei bin dell'istogramma, `binvol`, in numero di stati del vettore **q** diminuito di 1, `nstati`, il numero dei bin non vuoti dell'istogramma, `indne0`, la lista degli indici monodimensionali (*ind*, equazione (13)) dei bin non vuoti, `indici`, la lista delle coordinate dei centri delle celle di discretizzazione corrispondenti a bin non vuoti, `ygrid`, e la matrice diagonale delle varianze del rumore, contenuta nel vettore `Lambda`.

Listato 10: Dichiarazioni variabili subroutine `obj2`

```
      subroutine &
        obj2(epyw,mixing,wmat,mumat,numat,istogramma,binvol,&
           nstati,indne0,indici,ygrid,Lambda)
        use parametri
        implicit none
        integer, intent(in)              :: nstati,indne0
        integer, dimension(indne0), intent(in)        :: indici
        integer                     :: ipix,indiceq,isor,info,ican,ibin
        integer, dimension(NSORGENTI)    :: q
        integer, dimension(NCANALI)      :: ipvt
        real, dimension(NSORGENTI,NFATTORI),intent(in)::&
                                           wmat,mumat,numat
        real, dimension(NCANALI,NSORGENTI),intent(in) :: mixing
        real, dimension(NCANALI), intent(in)          :: Lambda
        real, dimension(NBIN**NCANALI), intent(in)    :: istogramma
        real, dimension(NBIN**NCANALI)   :: istpyw
        real, intent(out)                :: epyw
        real, dimension(NCANALI)         :: hmu,argv
        real, dimension(NCANALI)         :: work
        real                             :: pyw,w,arg
        real, dimension(2)               :: det
        real, dimension(NSORGENTI)       :: muvet,nuvet
        real, dimension(NSORGENTI,NCANALI)            :: htv
        real, dimension(NCANALI,NCANALI) :: hht
        real, dimension(NCANALI)         :: hhtinv
        real, intent(in)                 :: binvol
        real, dimension(NBIN**NCANALI,NCANALI), intent(in):: ygrid
```

Al Listato 11 è riportata l'interfaccia verso i sottoprogrammi richiamati da `obj2`. Si tratta della subroutine `prodmat`, che opera un prodotto matriciale e sarà descritta in seguito, de delle subroutine `SGEFA` e `SGEDI`, che sono due procedure di pubblico dominio, rispettivamente per la fattorizzazione e l'inversione di una matrice. Queste due ultime subroutine sono utilizzate nel calcolo della (7), in cui occorre valutare l'inversa e il determinante di un matrice. I codici sorgente sono riportati in appendice senza nessun commento.

Listato 11: Interfaccia sottoprogrammi (subroutine `obj2`)

```
        interface
           SUBROUTINE prodmat(a,b,c,nrb,ncb,ncc)
             IMPLICIT NONE
             integer, intent(in)                    :: nrb,ncb,ncc
             integer                                :: i,j,k
             real, dimension(nrb,ncc), intent(out) :: a
             real, dimension(nrb,ncb), intent(in)  :: b
             real, dimension(ncb,ncc), intent(in)  :: c
           end subroutine prodmat
!
           SUBROUTINE SGEDI(A,LDA,N,IPVT,DET,WORK,JOB)
             INTEGER LDA,N,IPVT(*),JOB
             REAL A(LDA,*),DET(2),WORK(*)
             REAL T
             REAL TEN
             INTEGER I,J,K,KB,KP1,L,NM1
           end subroutine SGEDI
!
           SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
             INTEGER LDA,N,IPVT(*),INFO
             REAL A(LDA,*)
             REAL T
             INTEGER ISAMAX,J,K,KP1,L,NM1
           end subroutine SGEFA
!
        end interface
```

La seconda sommatoria in (10) viene calcolata nel ciclo esterno con indice `ibin`, e si estende su tutti i bin non vuoti dell'istogramma, indirizzati, appunto, da `ibin`. I corrispondenti valori di **y**, in corrispondenza dei quali deve essere calcolata $p(\mathbf{y}|\mathbf{W})$, sono contenuti nell'array `ygrid`. Quest'ultima densità va calcolata mediante la somma in **q** in (5). A questo scopo si usa il ciclo interno con indice `indiceq`, che va da zero a `nstati`. Da questo indice vengono estratti gli stati del vettore **q** per mezzo del ciclo in `isor`, e i vettori delle medie e delle varianze delle gaussiane attive per ogni stato, `muvet` e `nuvet`. La probabilità dello stato, `w`, è calcolata come produttoria dei pesi relativi a tutte le gaussiane attive (vedi (3)). Per ogni **q** viene calcolata la gaussiana (7), quindi la sommatoria in (5). Il valore della densità così calcolato, `pyw`, viene moltiplicato per il volume del bin, `binvol`, ottenendo la probabilità che **y** appartenga al bin considerato. Questa probabilità va confrontata con la frequenza relativa contenuta nello stesso bin dell'istogramma dei dati. Questo valore viene moltiplicato per il logaritmo della probabilità `pyw`, a formare l'addendo della seconda sommatoria in (10) relativo alla **y** corrente. Fatto questo, il ciclo e completato e si passa al successivo **y**. Le probabilità `pyw` vengono salvate nel vettore `istpyw` semplicemente per consentire un eventuale confronto visivo tra questo e `istogramma`. Attraverso la

minimizzazione della divergenza tra questi due vettori si spera appunto di giungere a una stima utile dei parametri del modello. L'intera procedura è riportata al Listato 12.

Listato 12: Calcolo di $E(\log p(\mathbf{y}|\mathbf{W}))$ (subroutine `obj2`)

```
        epyw=0.
!
!  Calcolo aspettazione logaritmo densita' calcolata
!
        do ibin=1,indne0        !ciclo su bin non vuoti dell'ist.
           pyw=0.
           do indiceq=0,nstati  !ciclo su tutti gli stati q
              w=1.
!
!                estrazione vettore q da indice stato e calcolo p(q)
              do isor=1,NSORGENTI
                 q(isor)=modulo(indiceq/(NFATTORI**(isor-1)),&
                                        NFATTORI)+1
                 muvet(isor)=mumat(isor,q(isor))
                 nuvet(isor)=numat(isor,q(isor))
                 w=w*wmat(isor,q(isor)) !probabilita' stato q
              end do
!
!   Argomento gaussiana
!
!               y - H\mu\sub(q)
              call prodmat(hmu,mixing,muvet,NCANALI,NSORGENTI,1)
              argv=ygrid(ibin,:)-hmu
!
!               H'
              htv=transpose(mixing)
!
!               V*H'
              do isor=1,NSORGENTI
                 htv(isor,:)=nuvet(isor)*htv(isor,:)
              end do
!
!               H*V*H'
              call prodmat(hht,mixing,htv,NCANALI,NSORGENTI,NCANALI)
!
!               H*VH' + \Lambda
              do ican=1,NCANALI
                 hht(ican,ican)=hht(ican,ican)+Lambda(ican)
              end do
!
!               (H*VH' + \Lambda)**-1; |H*V*H'+\Lambda|
!
!                 fattorizzazione matrice HVqHT+Lambda
              call SGEFA(hht,NCANALI,NCANALI,ipvt,info)
!                 inversione matrice HVqHT+Lambda
              call SGEDI(hht,NCANALI,NCANALI,ipvt,det,work,11)
!
!               (H*V*H'+\Lambda)**-1*(y-H*\mu)
              call prodmat(hhtinv,hht,argv,NCANALI,NCANALI,1)
!
!                 (y-H*\mu)'*(H*V*H'+\Lambda)**-1*(y-H*\mu)
              call prodmat(arg,argv,hhtinv,1,NCANALI,1)
!
!                 argomento esponenziale gaussiana
              arg=-0.5*arg
!
!             calcolo gaussiana e somma in q
!
```

```
            pyw=pyw+w/sqrt(DPI**NCANALI*det(1)*10.**det(2))*exp(arg)
          end do   ! fine ciclo sugli stati
!
          pyw=pyw*binvol   ! probabilita' del valore y dato W
          istpyw(indici(ibin))=pyw
          epyw=epyw+istogramma(indici(ibin))*log(pyw)!incr. asp. log
        end do    ! fine ciclo sui bin
!
        return
      end subroutine obj2
```

2.9 Prodotto matriciale

Si riporta, al listato 13, anche la semplice procedura per il prodotto matriciale (subroutine `prodmat`), al solo scopo di sottolineare che, anche se si opera su vettori o addirittura su scalari, il risultato rimane valido. Se ciò avviene, come nelle ultime due chiamate di `prodmat` al Listato 12, si produce solo un avviso in fase di compilazione. L'operazione eseguita è il prodotto riga per colonna tra gli array bidimensionali reali `b` e `c`, che viene posto nell'array reale in uscita `a`. Ovviamente, bisognerà solo curare che le dimensioni negli array definiti nella routine chiamante siano tali che il numero di colonne di `b`, `ncb`, sia uguale al numero di righe di `c`, e che `a` abbia lo stesso numero di righe di `b` e lo stesso numero di colonne di `c`.

Listato 13: Prodotto matriciale (subroutine `prodmat`)

```
        SUBROUTINE prodmat(a,b,c,nrb,ncb,ncc)
      IMPLICIT NONE
        integer, intent(in)                    :: nrb,ncb,ncc
        integer                                :: i,j,k
        real, dimension(nrb,ncc), intent(out) :: a
        real, dimension(nrb,ncb), intent(in)  :: b
        real, dimension(ncb,ncc), intent(in)  :: c
        a=0.
        do i=1,nrb
           do j=1,ncc
              do k=1,ncb
                 a(i,j)=a(i,j)+b(i,k)*c(k,j)
              end do
           end do
        end do
        return
        end subroutine prodmat
```

3. Considerazioni

Molte delle scelte operate nella stesura di IFATEST in questa versione sono dettate dal particolare problema di elaborazione di immagini astrofisiche descritto in [2] e nelle relative citazioni bibliografiche, anche se si è cercato di produrre un codice il più possibile di uso generale, in particolare nel lasciare che i numeri di osservati e di sorgenti, come quelli dei fattori gaussiani, possano essere scelti arbitrariamente modificando solo il valore di alcuni parametri numerici.
La costruzione del codice non ha tenuto conto di possibili ottimizzazioni per quanto riguarda il tempo di calcolo e lo spazio di memoria richiesto.

È da notare che la scelta di calcolare la funzione obiettivo servendosi della (10) piuttosto che della (8), oltre che impedire di tener conto di eventuali conoscenze circa la variabilità del rumore, comporta l'impiego di uno spazio di memoria che cresce esponenzialmente con il numero dei canali di osservazione. Ciò vuol dire che questa implementazione non consentirà di operare su un numero di canali superiore a tre-quattro. Il vantaggio che offre, di contro, è la possibilità di controllare meglio l'andamento del processo di stima dei parametri e quindi di valutare l'efficacia dell'approccio IFA in diverse situazioni. Una versione successiva dovrebbe tornare a utilizzare la (8) per il calcolo dell'aspettazione, anche allo scopo di reintrodurre la possibilità di trattare rumore non stazionario.

La stima del modello non costituisce ancora la soluzione completa del problema della separazione cieca di segnali. Rimane da valutare la $\mathbf{x}$, noti $\mathbf{y}$, $\mathbf{H}$ e $\mathbf{\Lambda}$. Questo è un problema classico di ricostruzione di immagini che può essere risolto con uno dei metodi standard o ricorrendo anche per esso a una strategia di stima bayesiana.
Un altro problema da prendere in considerazione, una volta completata la sperimentazione di questo codice, è quello di trasformarlo in modo da poter essere integrato in ambienti di elaborazione quali IDL e relativi pacchetti dedicati, al fine di poter essere distribuito alla comunità scientifica interessata.

Appendice: subroutine `SGEFA` e `SGEDI`

Listato A.1: Fattorizzazione di matrice reale generale (subroutine `SGEFA`)

```
      SUBROUTINE SGEFA(A,LDA,N,IPVT,INFO)
C***BEGIN PROLOGUE  SGEFA
C***DATE WRITTEN   780814   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C***CATEGORY NO.  D2A1
C***KEYWORDS  FACTOR,LINEAR ALGEBRA,LINPACK,MATRIX
C***AUTHOR  MOLER, C. B., (U. OF NEW MEXICO)
C***PURPOSE  Factors a real matrix by Gaussian elimination.
C***DESCRIPTION
C
C     SGEFA factors a real matrix by Gaussian elimination.
C
C     SGEFA is usually called by SGECO, but it can be called
C     directly with a saving in time if  RCOND  is not needed.
C     (Time for SGECO) = (1 + 9/N)*(Time for SGEFA) .
C
C     On Entry
C
C        A       REAL(LDA, N)
C                the matrix to be factored.
C
C        LDA     INTEGER
C                the leading dimension of the array  A .
C
C        N       INTEGER
C                the order of the matrix  A .
C
C     On Return
C
C        A       an upper triangular matrix and the multipliers
C                which were used to obtain it.
```

```
C                The factorization can be written  A = L*U , where
C                L  is a product of permutation and unit lower
C                triangular matrices and  U  is upper triangular.
C
C        IPVT    INTEGER(N)
C                an integer vector of pivot indices.
C
C        INFO    INTEGER
C                = 0  normal value.
C                = K  if  U(K,K) .EQ. 0.0 .  This is not an error
C                     condition for this subroutine, but it does
C                     indicate that SGESL or SGEDI will divide by zero
C                     if called.  Use  RCOND  in SGECO for a reliable
C                     indication of singularity.
C
C     LINPACK.  This version dated 08/14/78 .
C     Cleve Moler, University of New Mexico, Argonne National Lab.
C
C     Subroutines and Functions
C
C     BLAS SAXPY,SSCAL,ISAMAX
C***REFERENCES  DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,
C                  *LINPACK USERS  GUIDE*, SIAM, 1979.
C***ROUTINES CALLED  ISAMAX,SAXPY,SSCAL
C***END PROLOGUE  SGEFA
      INTEGER LDA,N,IPVT(*),INFO
      REAL A(LDA,*)
C
      REAL T
      INTEGER ISAMAX,J,K,KP1,L,NM1
C
C     GAUSSIAN ELIMINATION WITH PARTIAL PIVOTING
C
C***FIRST EXECUTABLE STATEMENT  SGEFA
      INFO = 0
      NM1 = N - 1
      IF (NM1 .LT. 1) GO TO 70
      DO 60 K = 1, NM1
         KP1 = K + 1
C
C        FIND L = PIVOT INDEX
C
         L = ISAMAX(N-K+1,A(K,K),1) + K - 1
         IPVT(K) = L
C
C        ZERO PIVOT IMPLIES THIS COLUMN ALREADY TRIANGULARIZED
C
         IF (A(L,K) .EQ. 0.0E0) GO TO 40
C
C           INTERCHANGE IF NECESSARY
C
            IF (L .EQ. K) GO TO 10
               T = A(L,K)
               A(L,K) = A(K,K)
               A(K,K) = T
   10       CONTINUE
C
C           COMPUTE MULTIPLIERS
C
            T = -1.0E0/A(K,K)
            CALL SSCAL(N-K,T,A(K+1,K),1)
C
C           ROW ELIMINATION WITH COLUMN INDEXING
```

```
C
            DO 30 J = KP1, N
               T = A(L,J)
               IF (L .EQ. K) GO TO 20
                  A(L,J) = A(K,J)
                  A(K,J) = T
   20          CONTINUE
               CALL SAXPY(N-K,T,A(K+1,K),1,A(K+1,J),1)
   30       CONTINUE
         GO TO 50
   40    CONTINUE
            INFO = K
   50    CONTINUE
   60 CONTINUE
   70 CONTINUE
      IPVT(N) = N
      IF (A(N,N) .EQ. 0.0E0) INFO = N
      RETURN
      END
      INTEGER FUNCTION ISAMAX(N,SX,INCX)
C***BEGIN PROLOGUE  ISAMAX
C***DATE WRITTEN   791001   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C
C***CATEGORY NO.  D1A2
C***KEYWORDS  BLAS,LINEAR ALGEBRA,MAXIMUM COMPONENT,VECTOR
C***AUTHOR  LAWSON, C. L., (JPL)
C           HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS)
C           KROGH, F. T., (JPL)
C***PURPOSE  Find largest component of s.p. vector
C***DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       SX  single precision vector with N elements
C     INCX  storage spacing between elements of SX
C
C     --Output--
C   ISAMAX  smallest index (zero if N .LE. 0)
C
C     Find smallest index of maximum magnitude of single precision SX.
C     ISAMAX =  first I, I = 1 to N, to minimize  ABS(SX(1-
INCX+I*INCX)
C***REFERENCES  LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
C                *BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN USAGE*,
C                 ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
C                SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323
C***ROUTINES CALLED  (NONE)
C***END PROLOGUE  ISAMAX
C
      REAL SX(*),SMAX,XMAG
C***FIRST EXECUTABLE STATEMENT  ISAMAX
      ISAMAX = 0
      IF(N.LE.0) RETURN
      ISAMAX = 1
      IF(N.LE.1)RETURN
      IF(INCX.EQ.1)GOTO 20
```

```
C
C        CODE FOR INCREMENTS NOT EQUAL TO 1.
C
      SMAX = ABS(SX(1))
      NS = N*INCX
      II = 1
          DO 10 I=1,NS,INCX
          XMAG = ABS(SX(I))
          IF(XMAG.LE.SMAX) GO TO 5
          ISAMAX = II
          SMAX = XMAG
    5     II = II + 1
   10     CONTINUE
      RETURN
C
C        CODE FOR INCREMENTS EQUAL TO 1.
C
   20 SMAX = ABS(SX(1))
      DO 30 I = 2,N
         XMAG = ABS(SX(I))
         IF(XMAG.LE.SMAX) GO TO 30
         ISAMAX = I
         SMAX = XMAG
   30 CONTINUE
      RETURN
      END
      SUBROUTINE SAXPY(N,SA,SX,INCX,SY,INCY)
C***BEGIN PROLOGUE  SAXPY
C***DATE WRITTEN   791001   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C
C***CATEGORY NO.  D1A7
C***KEYWORDS  BLAS,LINEAR ALGEBRA,TRIAD,VECTOR
C***AUTHOR  LAWSON, C. L., (JPL)
C           HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS)
C           KROGH, F. T., (JPL)
C***PURPOSE  S.P. computation y = a*x + y
C***DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       SA  single precision scalar multiplier
C       SX  single precision vector with N elements
C     INCX  storage spacing between elements of SX
C       SY  single precision vector with N elements
C     INCY  storage spacing between elements of SY
C
C     --Output--
C       SY  single precision result (unchanged if N .LE. 0)
C
C     Overwrite single precision SY with single precision SA*SX +SY.
C     For I = 0 to N-1, replace  SY(LY+I*INCY) with SA*SX(LX+I*INCX) +
C      SY(LY+I*INCY), where LX = 1 if INCX .GE. 0, else LX = (-INCX)*N
C       and LY is defined in a similar way using INCY.
C***REFERENCES  LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
C                 *BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN
USAGE*,
```

```
C                 ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
C                 SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-
323
C***ROUTINES CALLED  (NONE)
C***END PROLOGUE  SAXPY
C
      REAL SX(*),SY(*),SA
C***FIRST EXECUTABLE STATEMENT  SAXPY
      IF(N.LE.0.OR.SA.EQ.0.E0) RETURN
      IF(INCX.EQ.INCY) IF(INCX-1) 5,20,60
    5 CONTINUE
C
C        CODE FOR NONEQUAL OR NONPOSITIVE INCREMENTS.
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        SY(IY) = SY(IY) + SA*SX(IX)
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C        CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C        CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 4.
C
   20 M = MOD(N,4)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        SY(I) = SY(I) + SA*SX(I)
   30 CONTINUE
      IF( N .LT. 4 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,4
        SY(I) = SY(I) + SA*SX(I)
        SY(I + 1) = SY(I + 1) + SA*SX(I + 1)
        SY(I + 2) = SY(I + 2) + SA*SX(I + 2)
        SY(I + 3) = SY(I + 3) + SA*SX(I + 3)
   50 CONTINUE
      RETURN
C
C        CODE FOR EQUAL, POSITIVE, NONUNIT INCREMENTS.
C
   60 CONTINUE
      NS = N*INCX
          DO 70 I=1,NS,INCX
          SY(I) = SA*SX(I) + SY(I)
   70     CONTINUE
      RETURN
      END
      SUBROUTINE SSCAL(N,SA,SX,INCX)
C***BEGIN PROLOGUE  SSCAL
C***DATE WRITTEN   791001   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C
C***CATEGORY NO.  D1A6
C***KEYWORDS  BLAS,LINEAR ALGEBRA,SCALE,VECTOR
```

```
C***AUTHOR  LAWSON, C. L., (JPL)
C           HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS)
C           KROGH, F. T., (JPL)
C***PURPOSE  S.P. vector scale x = a*x
C***DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       SA  single precision scale factor
C       SX  single precision vector with N elements
C     INCX  storage spacing between elements of SX
C
C     --Output--
C       SX  single precision result (unchanged if N .LE. 0)
C
C     Replace single precision SX by single precision SA*SX.
C     For I = 0 to N-1, replace SX(1+I*INCX) with  SA * SX(1+I*INCX)
C***REFERENCES  LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
C                 *BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN
USAGE*,
C              ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
C              SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-323
C***ROUTINES CALLED  (NONE)
C***END PROLOGUE  SSCAL
C
      REAL SA,SX(*)
C***FIRST EXECUTABLE STATEMENT  SSCAL
      IF(N.LE.0)RETURN
      IF(INCX.EQ.1)GOTO 20
C
C        CODE FOR INCREMENTS NOT EQUAL TO 1.
C
      NS = N*INCX
          DO 10 I = 1,NS,INCX
          SX(I) = SA*SX(I)
   10     CONTINUE
      RETURN
C
C        CODE FOR INCREMENTS EQUAL TO 1.
C
C
C        CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 5.
C
   20 M = MOD(N,5)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        SX(I) = SA*SX(I)
   30 CONTINUE
      IF( N .LT. 5 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,5
        SX(I) = SA*SX(I)
        SX(I + 1) = SA*SX(I + 1)
        SX(I + 2) = SA*SX(I + 2)
        SX(I + 3) = SA*SX(I + 3)
        SX(I + 4) = SA*SX(I + 4)
   50 CONTINUE
      RETURN
      END
```

Listato A.2: Inversione di matrice reale fattorizzata (subroutine SGEDI)

```
      SUBROUTINE SGEDI(A,LDA,N,IPVT,DET,WORK,JOB)
C***BEGIN PROLOGUE  SGEDI
C***DATE WRITTEN   780814   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C***CATEGORY NO.  D2A1,D3A1
C***KEYWORDS  DETERMINANT,FACTOR,INVERSE,LINEAR
ALGEBRA,LINPACK,MATRIX
C***AUTHOR  MOLER, C. B., (U. OF NEW MEXICO)
C***PURPOSE  Computes the determinant and inverse of a matrix
C            using the factors computed by SGECO or SGEFA.
C***DESCRIPTION
C
C     SGEDI computes the determinant and inverse of a matrix
C     using the factors computed by SGECO or SGEFA.
C
C     On Entry
C
C        A       REAL(LDA, N)
C                the output from SGECO or SGEFA.
C
C        LDA     INTEGER
C                the leading dimension of the array  A .
C
C        N       INTEGER
C                the order of the matrix  A .
C
C        IPVT    INTEGER(N)
C                the pivot vector from SGECO or SGEFA.
C
C        WORK    REAL(N)
C                work vector.  Contents destroyed.
C
C        JOB     INTEGER
C                = 11   both determinant and inverse.
C                = 01   inverse only.
C                = 10   determinant only.
C
C     On Return
C
C        A       inverse of original matrix if requested.
C                Otherwise unchanged.
C
C        DET     REAL(2)
C                determinant of original matrix if requested.
C                Otherwise not referenced.
C                Determinant = DET(1) * 10.0**DET(2)
C                with  1.0 .LE. ABS(DET(1)) .LT. 10.0
C                or  DET(1) .EQ. 0.0 .
C
C     Error Condition
C
C        A division by zero will occur if the input factor contains
C        a zero on the diagonal and the inverse is requested.
C        It will not occur if the subroutines are called correctly
C        and if SGECO has set RCOND .GT. 0.0 or SGEFA has set
C        INFO .EQ. 0 .
C
C     LINPACK.  This version dated 08/14/78 .
C     Cleve Moler, University of New Mexico, Argonne National Lab.
```

```
C
C     Subroutines and Functions
C
C     BLAS SAXPY,SSCAL,SSWAP
C     Fortran ABS,MOD
C***REFERENCES  DONGARRA J.J., BUNCH J.R., MOLER C.B., STEWART G.W.,
C                 *LINPACK USERS  GUIDE*, SIAM, 1979.
C***ROUTINES CALLED  SAXPY,SSCAL,SSWAP
C***END PROLOGUE  SGEDI
      INTEGER LDA,N,IPVT(*),JOB
      REAL A(LDA,*),DET(2),WORK(*)
C
      REAL T
      REAL TEN
      INTEGER I,J,K,KB,KP1,L,NM1
C
C     COMPUTE DETERMINANT
C
C***FIRST EXECUTABLE STATEMENT  SGEDI
      IF (JOB/10 .EQ. 0) GO TO 70
         DET(1) = 1.0E0
         DET(2) = 0.0E0
         TEN = 10.0E0
         DO 50 I = 1, N
            IF (IPVT(I) .NE. I) DET(1) = -DET(1)
            DET(1) = A(I,I)*DET(1)
C        ...EXIT
            IF (DET(1) .EQ. 0.0E0) GO TO 60
   10       IF (ABS(DET(1)) .GE. 1.0E0) GO TO 20
               DET(1) = TEN*DET(1)
               DET(2) = DET(2) - 1.0E0
            GO TO 10
   20       CONTINUE
   30       IF (ABS(DET(1)) .LT. TEN) GO TO 40
               DET(1) = DET(1)/TEN
               DET(2) = DET(2) + 1.0E0
            GO TO 30
   40       CONTINUE
   50    CONTINUE
   60    CONTINUE
   70 CONTINUE
C
C     COMPUTE INVERSE(U)
C
      IF (MOD(JOB,10) .EQ. 0) GO TO 150
         DO 100 K = 1, N
            A(K,K) = 1.0E0/A(K,K)
            T = -A(K,K)
            CALL SSCAL(K-1,T,A(1,K),1)
            KP1 = K + 1
            IF (N .LT. KP1) GO TO 90
            DO 80 J = KP1, N
               T = A(K,J)
               A(K,J) = 0.0E0
               CALL SAXPY(K,T,A(1,K),1,A(1,J),1)
   80       CONTINUE
   90       CONTINUE
  100    CONTINUE
C
C        FORM INVERSE(U)*INVERSE(L)
C
         NM1 = N - 1
         IF (NM1 .LT. 1) GO TO 140
```

```
         DO 130 KB = 1, NM1
            K = N - KB
            KP1 = K + 1
            DO 110 I = KP1, N
               WORK(I) = A(I,K)
               A(I,K) = 0.0E0
  110       CONTINUE
            DO 120 J = KP1, N
               T = WORK(J)
               CALL SAXPY(N,T,A(1,J),1,A(1,K),1)
  120       CONTINUE
            L = IPVT(K)
            IF (L .NE. K) CALL SSWAP(N,A(1,K),1,A(1,L),1)
  130    CONTINUE
  140    CONTINUE
  150 CONTINUE
      RETURN
      END
      SUBROUTINE SSWAP(N,SX,INCX,SY,INCY)
C***BEGIN PROLOGUE  SSWAP
C***DATE WRITTEN   791001   (YYMMDD)
C***REVISION DATE  820801   (YYMMDD)
C***REVISION HISTORY  (YYMMDD)
C   000330  Modified array declarations.  (JEC)
C
C***CATEGORY NO.  D1A5
C***KEYWORDS  BLAS,INTERCHANGE,LINEAR ALGEBRA,VECTOR
C***AUTHOR  LAWSON, C. L., (JPL)
C           HANSON, R. J., (SNLA)
C           KINCAID, D. R., (U. OF TEXAS)
C           KROGH, F. T., (JPL)
C***PURPOSE  Interchange s.p vectors
C***DESCRIPTION
C
C                B L A S  Subprogram
C    Description of Parameters
C
C     --Input--
C        N  number of elements in input vector(s)
C       SX  single precision vector with N elements
C     INCX  storage spacing between elements of SX
C       SY  single precision vector with N elements
C     INCY  storage spacing between elements of SY
C
C     --Output--
C       SX  input vector SY (unchanged if N .LE. 0)
C       SY  input vector SX (unchanged if N .LE. 0)
C
C     Interchange single precision SX and single precision SY.
C     For I = 0 to N-1, interchange  SX(LX+I*INCX) and SY(LY+I*INCY),
C     where LX = 1 if INCX .GE. 0, else LX = (-INCX)*N, and LY is
C     defined in a similar way using INCY.
C***REFERENCES  LAWSON C.L., HANSON R.J., KINCAID D.R., KROGH F.T.,
C                 *BASIC LINEAR ALGEBRA SUBPROGRAMS FOR FORTRAN
USAGE*,
C                 ALGORITHM NO. 539, TRANSACTIONS ON MATHEMATICAL
C                 SOFTWARE, VOLUME 5, NUMBER 3, SEPTEMBER 1979, 308-
323
C***ROUTINES CALLED  (NONE)
C***END PROLOGUE  SSWAP
C
      REAL SX(*),SY(*),STEMP1,STEMP2,STEMP3
C***FIRST EXECUTABLE STATEMENT  SSWAP
```

```
      IF(N.LE.0)RETURN
      IF(INCX.EQ.INCY) IF(INCX-1) 5,20,60
    5 CONTINUE
C
C       CODE FOR UNEQUAL OR NONPOSITIVE INCREMENTS.
C
      IX = 1
      IY = 1
      IF(INCX.LT.0)IX = (-N+1)*INCX + 1
      IF(INCY.LT.0)IY = (-N+1)*INCY + 1
      DO 10 I = 1,N
        STEMP1 = SX(IX)
        SX(IX) = SY(IY)
        SY(IY) = STEMP1
        IX = IX + INCX
        IY = IY + INCY
   10 CONTINUE
      RETURN
C
C       CODE FOR BOTH INCREMENTS EQUAL TO 1
C
C
C       CLEAN-UP LOOP SO REMAINING VECTOR LENGTH IS A MULTIPLE OF 3.
C
   20 M = MOD(N,3)
      IF( M .EQ. 0 ) GO TO 40
      DO 30 I = 1,M
        STEMP1 = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP1
   30 CONTINUE
      IF( N .LT. 3 ) RETURN
   40 MP1 = M + 1
      DO 50 I = MP1,N,3
        STEMP1 = SX(I)
        STEMP2 = SX(I+1)
        STEMP3 = SX(I+2)
        SX(I) = SY(I)
        SX(I+1) = SY(I+1)
        SX(I+2) = SY(I+2)
        SY(I) = STEMP1
        SY(I+1) = STEMP2
        SY(I+2) = STEMP3
   50 CONTINUE
      RETURN
   60 CONTINUE
C
C     CODE FOR EQUAL, POSITIVE, NONUNIT INCREMENTS.
C
      NS = N*INCX
        DO 70 I=1,NS,INCX
        STEMP1 = SX(I)
        SX(I) = SY(I)
        SY(I) = STEMP1
   70   CONTINUE
      RETURN
      END
```

Riferimenti

[1] H. Attias, "Independent Factor Analysis", *Neural Computation*, Vol. 11, pp. 803-851, 1999.

[2] E. Kuruoglu, L. Bedini, M.T. Paratore, E. Salerno, A. Tonazzini, "Source separation in astrophysical maps using independent factor analysis", *Neural Networks*, to appear, 2003.

[3] A. Hyvärinen and E. Oja, "Independent Component Analysis: Algorithms and Applications", *Neural Networks*, Vol. 13, pp. 411-430, 2000.

[4] L. Bedini, E. Salerno, A. Tonazzini, "Blind Source Separation from noisy data using Bayesian Estimation and Gibbs Priors", *Proc. IASTED-SIP'2001*, pp. 108-112, 2001.

[5] http://www.npac.syr.edu/users/gcf/cps615D/foilsepimagedir/010IMAGE.html