MELISSA: un package per Mathematica per l'analisi Singular-Spectrum delle serie temporali. Guida per l'utente.

Marco Sandri

Istituto di Scienze Economiche,
Università di Verona, Via dell'Artigliere 19, 37129 Verona;
†msandri@ivruniv.bitnet

1 Maggio 2003

MELISSA **è un pacchetto di comandi in ambiente Mathematica che permette di realizzare l'analisi SSA di serie temporali. Viene qui fornita una breve guida per l'utente.**

1 Introduzione.

Nelle pagine che seguono, in cui si descrivono le funzioni e le procedure contenute nel package *MELISSA*, faremo costante riferimento a [6], mentre per l'uso e lo sviluppo di packages in ambiente Mathematica rinviamo a [8], [4] e [7].

Attenzione! Il file contenente il package *MELISSA* si chiama `melissa.m` e deve risiedere entro la directory

```
c:\math
```

insieme ai file `mel20`, `mel40` e `sval20.p3`. Per rendere operativo il package è prima di tutto necessario entrare in Mathematica (con il comando `math`), e quindi digitare:

```
In[1]:= <<melissa.m
```

Sono a questo punto disponibili tutte le funzioni e le procedure contenute nel package. Ne diamo qui elenco:

```
AddNoise
BetaSearch
Covariance
CovarianceMatrix
EigenSpectrumErrorPlot
EigenSpectrumPlot
FastMean
FastStandardDeviation
FastVariance
kthRC
Ld
LinearPrediction
MultiEigenSpectrumPlot
MultiRC
NMSE
Norm
Normalize
P0
P1
P2
P3
PCs
PhasePlot
RC
RCLinearPrediction
Rebuild
Rescale
ResidualAutocovariance
ShowXY
SurrogateAutocovariance
SurrogatePlot
Sv
Toeplitz
```

Un breve messaggio di help in cui viene descritto l'uso di ciascun comando è disponibile digitando `?<nomecomando>`. Ad esempio, per ottenere un aiuto in relazione al comando Toeplitz, scrivere `?Toeplitz`.

Non va dimenticato che Mathematica fa differenza fra lettere maiuscole e lettere minuscole. Il comando `eigenspectrumplot` non e' infatti la stessa cosa di `EigenSpectrumPlot`. Nel primo caso si possono avere messaggi d'errore o output 'strani'.

Il presente package è stato sviluppato su un PC 486/50 Mhz dotato di 16MByte di RAM, utilizzando la versione 2.2 di Mathematica. Risultati difformi da quelli qui riportati devono essere pertanto imputati alla diversa configurazione del proprio PC, o alla diversa versione di Mathematica posseduta.

2 Comandi di input/output, generazione, trattamento e visualizzazione dati.

Supponiamo di voler caricare in Mathematica una serie bivariata di 150 dati contenuta nel file `henon.dat`. Nonostante Mathematica sia ricco di numerosi comandi per l'input/output di dati alfanumerici, si è preferito creare il comando `Ld`. Nel caso in esame il suo uso è:

```
In[]:= x=Ld["henon.dat",2*150];
```

Da questo momento le serie relative alla prima ed alla seconda variabile prendono rispettivamente il nome di `x[[1]]` e `x[[2]]`.

Volendo invece provvedere a salvare in formato ASCII la lista di dati chiamata y in un file di nome `out.dat`, il comando è:

```
In[]:= Sv["out.dat",y];
```

È importante ricordare che `Sv` consente di salvare in formato ASCII sia serie univariate che multivariate, cioè anche matrici di dati. I comandi che seguono generano una matrice di 20 righe per 10 colonne di valori casuali, e provvedono poi a salvarla nel file "prova.asc":

```
In[]:= xx=Table[Random[],{20},{10}];
In[]:= Sv["prova.asc",xx];
```

Si provi a visualizzare il contenuto del file con un editor.

Nel lavoro [6] vengono usati, al fine di 'testare' l'efficacia della SSA, quattro diversi processi, denominati rispettivamente P1, P2, P3, P4. Lasciando da parte P4, un processo deterministico del tipo 'sistema di Lorenz', abbiamo implementato i primi tre processi. Volendo ad esempio generare una serie storica (che chiameremo x) di 500 osservazioni, proveniente dal processo stocastico P1, dobbiamo ricorrere al comando `P1` (`P2` o `P3`):

```
In[]:= x=P1[500];
```

E' disponibile anche il comando `P0`, relativo al processo deterministico periodico, ma privo di 'noise', presente in P1, P2, P3. È cioè il processo descritto nella (1.5), p. 99, di [6], dove Φ_1 e Φ_2 vengono scelti casualmente nell'intervallo $[0, 2\pi]$.

Il comando `Normalize`, applicato ad una serie x, rimuove la media e divide i valori della serie per la deviazione standard. In altri termini, crea una nuova serie con media nulla e varianza unitaria.

```
In[]:=  y=Normalize[x];
In[]:=  Mean[y] // Chop
Out[]:= 0
In[]:=  Variance[y]
Out[]:= 1.
```

Il comando `Rescale` opera invece una trasformazione lineare sui dati in maniera tale che tutti i valori siano compresi nell'intervallo $[-0.9, +0.9]$. L'uso del comando e' molto semplice:

```
In[]:=  z=Rescale[x];
```

Va osservato che:

```
In[]:=  Max[z]
Out[]:= 0.9
In[]:=  Min[z]
Out[]:= -0.9
```

Il comando `Rescale` verrà in generale impiegato per preparare i dati da fornire come input a delle reti neuronali.

L'istruzione `PhasePlot` permette di ottenere il diagramma di fase di una data serie temporale x, essa cioè traccia sul piano l'insieme dei punti aventi coordinate $(x_t, x_{t+\tau})$. Le opzioni che si possono utilizzare sono quelle definite per `ListPlot` (cfr. [8]). È possibile collegare punti temporalmente successivi con una linea continua: l'opzione da utilizzare è `PlotJoined->True`.

Mostriamo un esempio d'uso del comando, dove x è una serie di 1000 osservazioni ottenute dal modello di Lorenz e dove $\tau = 5$:

```
In[]:= z=Ld["lor.dat",1000];
In[]:= a=PhasePlot[z,5]
```

Il risultato è mostrato in Fig.1 a).

Volendo invece collegare i punti con una linea continua:

```
In[]:= b=PhasePlot[z,5,PlotJoined->True]
```

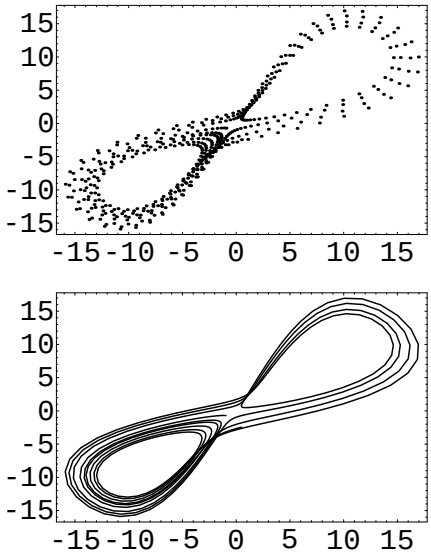

Fig. 1. Esempi di diagramma di fase.

Si veda la Fig.1 b).

Volendo tracciare contemporaneamente l'andamento temporale di due diverse serie temporali x ed y, utilizzando per la prima una linea tratteggiata e per la seconda una linea continua, è disponibile `ShowXY`. La sintassi di questo comando è: `ShowXY[x,y,opts]`, dove `opts` può essere una (o più) delle opzioni disponibili per il comando `Show` (cfr. [8]).

`Rebuild[x,m,`τ`]` esegue la ricostruzione della serie univariata x nello spazio ad m dimensioni, dove τ e' il ritardo temporale di ricostruzione (se non specificato e' posto pari ad uno). Volendo ad esempio ricostruire nello spazio a 3 dimensioni, con $\tau = 2$, la serie z dell'esempio precedente, si dovrà scrivere:

```
In[]:= w=Rebuild[z,3,2];
```

Il comando `AddNoise` consente invece di aggiungere ad una lista univariata o multivariata di dati un rumore avente media nulla e varianza pari ad una percentuale desiderata della varianza del segnale d'origine. Sia ad esempio z una serie trivariata di N osservazioni provenienti da un modello di Lorenz. Volendo aggiungere alle tre componenti un rumore pari rispettivamente al 10%, 5% e 1% della variabilità dei relativi segnali, dovremo scrivere:

```
In[]:= y=AddNoise[{z[[1]],z[[2]],z[[3]]},{10,5,1}];
```

La Fig.2 mostra il diagramma di fase relativo alle serie `y[[1]]`, `y[[2]]`, `y[[3]]` così ottenute.

Nel caso in cui la serie da 'contaminare' sia univariata (considerando ad es. solo `z[[1]]`) la sintassi da utilizzare è:

```
In[]:= y=AddNoise[z[[1]],10];
```

I comandi `FastMean`, `FastVariance` e `FastStandardDeviation` sono versioni velocizzate di `Mean`, `Variance` e `StandardDeviation` rispettivamente, che, diversamente da questi ultimi, possono operare anche su serie multivariate. Essi sono quindi in generale da preferire ai comandi classici forniti da Mathematica.

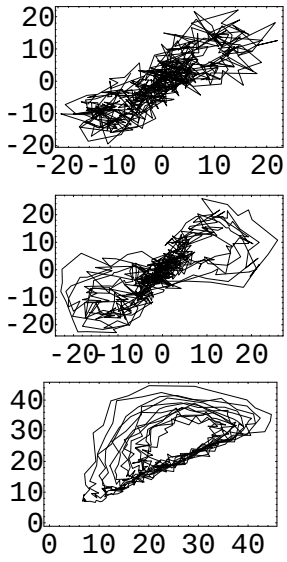

Fig. 2. Variabili del modello di Lorenz con noise additivo.

3 Comandi per la 'Singular-Spectral Analysis' univariata.

Il comando `Covariance` permette di stimare, usando il metodo di Pearson (cfr. (2.3), p. 101, in [6]), le autocovarianze della serie storica data x, fino al 'lag' M. Il comando restituisce una 'Table' di M valori corrispondenti a: $c(0), \ldots, c(M-1)$. Digitando ad esempio:

```
In[]:= x=P0[200];
In[]:= Covariance[x,10]
```

si ottiene:

```
Out[]:= {2.5389, 2.24335, 1.51326, 0.707255, 0.137067,
-0.138357, -0.321053, -0.681897, -1.31664, -2.03776}
```

Il comando `Toeplitz` costruisce invece, a partire dai dati forniti da `Covariance`, la matrice delle autocovarianze per la serie data. Il suo uso ad esempio è :

```
In[]:= Toeplitz[x,5]
```

e il risultato è la matrice simmetrica:

```
Out[]:= {{2.5389, 2.24335,1.51326, 0.707255,0.137067},
>       {2.24335, 2.5389, 2.24335, 1.51326, 0.707255},
>       {1.51326, 2.24335, 2.5389, 2.24335, 1.51326},
>       {0.707255, 1.51326, 2.24335, 2.5389, 2.24335},
>       {0.137067, 0.707255, 1.51326, 2.24335, 2.5389}}
```

Negli esempi sin qui riportati, non e' stato specificato il tipo di algoritmo da impiegare nella stima dell'autocovarianza. Per default è impostato quello di Pearson, che è estremamente veloce. Un metodo alternativo, molto impegnativo sotto il profilo del tempo di computazione richiesto, è quello di Burg (cfr. [5], pp. 568-569 e [6], p. 101). Per selezionarlo basta usare l'opzione `Algorithm`:

```
In[]:= Covariance[x,10,Algorithm->"Burg"]
```

dalla quale si ottiene:

```
Out[]:= {2.5389, 2.24807, 1.52711, 0.724303, 0.14398,
-0.150892, -0.348892, -0.709102, -1.32779, -2.03056}
```

La
medesima opzione è disponibile anche sui comandi `Toeplitz`, `EigenSpectrumErrorPlot`, `EigenSpectrumPlot`, `RC`, `PCs`, `kthRC`, `ResidualAutocovariance`, `SurrogateAutocovariance`.

Si ricordi che in Mathematica, come in molti linguaggi di programmazione, allo scopo di evitare che i risultati di una procedura vengano visualizzati sul monitor, è necessario collocare alla fine del comando un punto e virgola[1]. Ad esempio per calcolare la matrice di Toeplitz del caso precedence, associandola alla variabile ToMat, senza però visualizzarla, dobbiamo scrivere:

```
In[]:= ToMat=Toeplitz[x,5];
```

Per verificare se ToMat è realmente simmetrica, possiamo usare la semplice procedura:

```
In[]:=  ToMat==Transpose[ToMat]
```

La risposta di Mathematica sarà :

```
Out[]:= True
```

Il comando `EigenSpectrumPlot` è in grado di 'plottare' in 4 modi differenti lo spettro di M autovalori (posti in ordine decrescente di ampiezza) relativo ad una data serie x. Se si specifica l'opzione `PlotType->"Eigenvalues"` (data per default), si ottiene il grafico dello spettro SSA su scala logaritmica (Fig.3):

```
In[]:= x=P3[150];
In[]:= EigenSpectrumPlot[x,40]
```

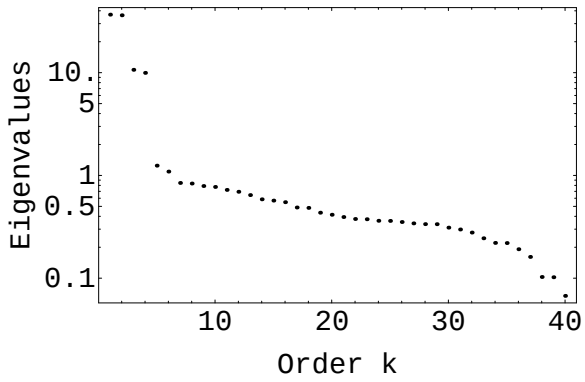

Fig. 3. Esempio di spettro di autovalori ottenuto con il comando `EigenSpectrumPlot` e l'opzione `PlotType->"Eigenvalues"`.

Dagli algoritmi impiegati è spesso possibile ottenere autovalori negativi. Dato che `EigenSpectrumPlot` impiega una scala logaritmica, essi non compaiono sul grafico (vedi ad es. Fig.3).

[1] Si consiglia, se non quando strettamente indispensabile, di evitare sempre la visualizzazione dei risultati, specie se questi occupano diverse schermate. Si potrà risparmiare parecchio tempo.

Specificando `PlotType->"Percentage"` otteniamo il grafico della percentuale della varianza totale del segnale che è associata a (cioè 'spiegata' da) ciascuna delle direzioni principali relativa a ciascuno degli M autovalori della matrice di Toeplitz.

```
In[]:= EigenSpectrumPlot[x,20,PlotType->"Percentage"];
```

Un esempio è fornito in Fig.4.

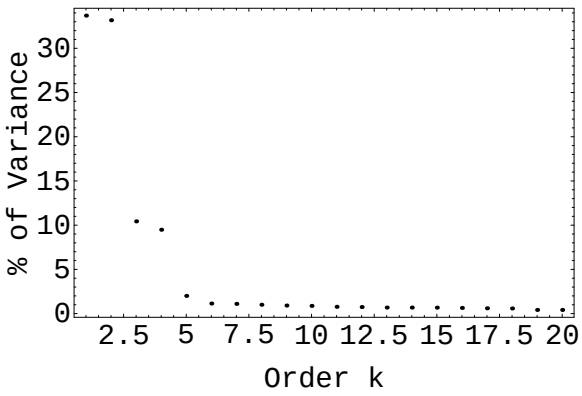

Fig. 4. Percentuale della varianza totale associata a ciascun autovalore: l'opzione `PlotType->"Percentage"`.

Volendo invece ottenere la percentuale della varianza totale del segnale spiegata dalle prime n direzioni principali, si può utilizzare l'opzione `PlotType->"CumulativePercentage"`.

```
In[]:= EigenSpectrumPlot[x,20,PlotType->"CumulativePercentage"];
```

Un esempio è fornito in Fig.5.

Da ultimo, è disponibile l'opzione `PlotType->"Differences"` che consente di tracciare l'entità della differenza relativa fra un autovalore e quello successivo. Esso cioè porta a tracciare i valori:

$$\frac{\lambda_{i+1} - \lambda_i}{\lambda_i}, \qquad i = 1, \ldots, M-1,$$

dove λ_i è l'i-esimo autovalore dello spettro SSA. Un esempio è fornito in Fig.6.

Va da ultimo ricordato che `EigenSpectrumPlot` accetta tutte le opzioni previste per il comando `ListPlot`. Esso può quindi essere largamente personalizzato, con l'aggiunta ad es. di diciture sugli assi, cambi di scala, impostazione di font, etc.

Volendo ottenere le figure mostrate a p. 103 di [6], cioè lo spettro degli autovalori di x unitamente ai relativi intervalli di confidenza al 95%, è disponibile il comando `EigenSpectrumErrorPlot`. È necessario specificare nell'ordine:

1. il tipo di processo (P0, P1, P2, P3, od altri che in futuro verranno implementati);
2. il numero di 'lags' M;
3. la lunghezza di ogni singola realizzazione del processo (150, ad esempio, in [6]);

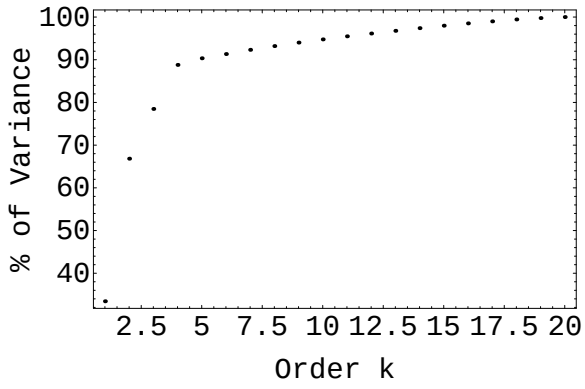

Fig. 5. Percentuale della varianza totale associata ai primi n autovalori: l'opzione `PlotType->"CumulativePercentage"`.

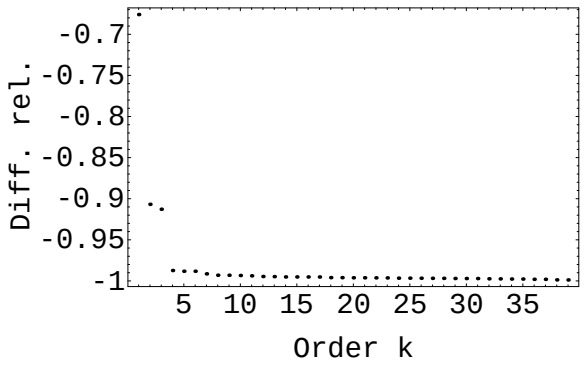

Fig. 6. Differenze relative fra autovalori successivi: l'opzione `PlotType->"Differences"`.

4. il numero complessivo di realizzazioni del processo (100 in [6]).

Ad esempio:

```
In[]:= EigenSpectrumErrorPlot[P1,40,150,100]
```

Questa procedura, dato il notevole impegno computazionale, in generale richiede alcune decine di minuti prima di essere completata. Il risultato viene mostrato in Fig.7. Va osservato che questo comando non traccia gli intervalli di confidenza, ma si limita ad indicare per ogni 'lag' l'estremo superiore ed inferiore dell'intervallo ed il corrispondente valore medio. Ciò è dovuto ad una limitazione di Mathematica che non dispone di un comando `ErrorListPlot` con coordinate logaritmiche.

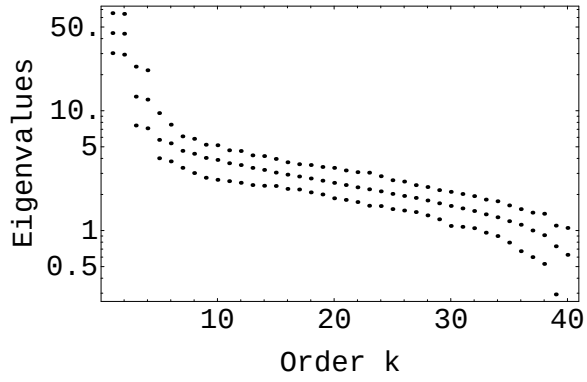

Fig. 7. Esempio di spettro di autovalori, con i relativi intervalli di confidenza al 95%, ottenuto con `EigenSpectrumErrorPlot`.

La procedura `RC` consente di calcolare la serie ricostruita $y = R_{\mathcal{A}}x$, dove $\mathcal{A}$ è l'insieme dei primi p autovalori dello spettro, secondo il metodo suggerito da [6], pp. 105-106. I parametri da specificare sono, nell'ordine: la serie temporale su cui operare la ricostruzione, il numero di 'lags' M, il numero p di autovalori che si vogliono considerare nella ricostruzione. Ovviamente, se $p = M$, `RC` restituisce la serie originaria.

```
In[]:= x=P2[150];
In[]:= y=RC[x,20,4];
```

Per visualizzare la serie ottenuta insieme alla serie originaria (vedi Fig.8), conviene utilizzare il comando `ShowXY`:

```
In[]:= ShowXY[x,y]
```

La procedura `kthRC` è nella sostanza identica a quella appena qui descritta. Essa infatti permette di calcolare la serie ricostruita $x^k = R_{\mathcal{A}}x$, con la sola differenza che ora l'insieme $\mathcal{A}$ è costituito da un solo indice k, cioè $\mathcal{A} = \{k\}$, e non $\mathcal{A} = \{1, \ldots, k\}$ come nel caso precedente per il comando `RC`. Non va dimenticata la proprietà additiva di cui godono queste componenti ricostruite:

$$R_{\mathcal{A}}x = \sum_{k\in\mathcal{A}} x^k.$$

La si può facilmente verificare con i comandi:

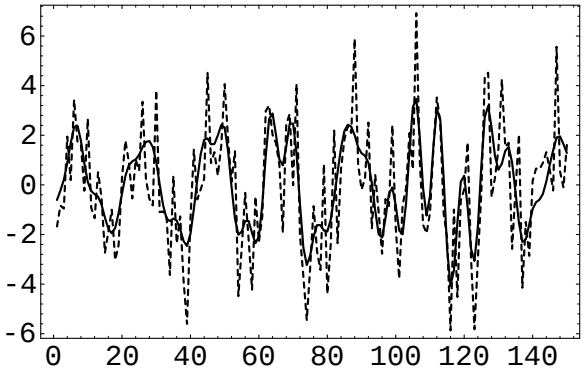

Fig. 8. Serie storica vera (linea tratteggiata) e serie ricostruita con la procedura `RC` (linea continua).

```
In[]:=  p=4;
In[]:=  For[i=1,i<p+1,i++, z[i]=kthRC[x,20,i];]
In[]:=  y==Sum[z[i],{i,p}]
Out[]:= True
```

dove x e y sono state definite nei passi precedenti, durante la descrizione di `RC`.

L'istruzione `PCs` calcola le prime p componenti principali, cioè i coefficienti delle proiezioni ortogonali della serie temporale sulle prime p funzioni ortogonali (EOF), come indicato in [1] e [6]:

$$a_i^k = \sum_{j=1}^{M} x_{i+j} E_j^k, \qquad 0 \le i \le N - M.$$

Usando il comdano `PCs` è necessario specificare nell'ordine: la serie temporale x, il numero di lag M e il numero delle prime p componenti che si vogliono ottenere:

```
In[]:= x=P3[250];
In[]:= w=PCs[x, 20, 4];
```

dove w è una matrice a 4 righe, corrispondenti alle prime 4 componenti principali della serie x. Volendo ora tracciare il diagramma di fase della prima componente principale, assunto un ritardo $\tau = 5$, si scriverà:

```
In[]:= PhasePlot[w[[1]],5];
```

`ResidualAutocovariance` calcola la funzione di autocovarianza (con il metodo di Pearson) dei residui ottenuti dopo la ricostruzione, applica cioè `Covariance` sulla serie $x - R_{\mathcal{A}}x = R_{\mathcal{A}'}x$.

```
In[]:= cp=ResidualAutocovariance[x,20,5];
```

Per accettare o respingere l'ipotesi di residui incorrelati, o comunque di una 'corretta' e 'sufficiente' ricostruzione attraverso l'uso delle prime p 'reconstructed components' (*RCs*), è necessario confrontare, seguendo [6] pp. 107-110, i valori di $c_p(j)$ con quelli ottenuti dai 'dati surrogati', cioè quelli che [6] chiama $c_v(j)$ e $s_v(j)$.

I valori di questi ultimi possono essere generati, per un dato valore di p, attraverso `SurrogateAutocovariance`. Esso richiede i seguenti parametri:

1. il numero di 'lags' M;
2. la lunghezza di ogni singola realizzazione del processo 'white noise';
3. il numero complessivo di realizzazioni;
4. il numero di autovalori da considerare nella fase di ricostruzione.

Anche questa procedura richiede tempi di computazione elevati.

```
In[]:= {cv,sv}=SurrogateAutocovariance[x,150,100,5];
```

L'istruzione `Surrogate Plot` permette di tracciare contemporaneamente gli M valori dell'autocovarianza dei residui e della autocovarianza dei dati surrogati, questi ultimi corredati dei relativi 'error bars' ed in più moltiplicati per il coefficiente β definito in [6], p. 107. La sintassi da seguire è: specificare il valore di β, la lista contenente le autocovarianze dei residui $c_p(j)$, la lista dei valori medi delle autocovarianze dei dati surrogati $c_v(j)$, la lista delle varianze delle stime dell'autocovarianza dei dati surrogati $s_v(j)$. Vediamo un esempio il cui risultato è riportato in 9::

```
In[]:= SurrogatePlot[2.4,cp,cv,sv];
```

`SurrogatePlot` è utile per ricercare 'ad occhio' l'eventuale valore di β in corrispondenza del quale tutti i valori $c_p(j)$ cadono negli intervalli:

$$\left[\beta^2\left(cv(j)-1.96\sqrt{s_v(j)}\right),\beta^2\left(cv(j)+1.96\sqrt{s_v(j)}\right)\right].$$

Il comando `BetaSearch` consente invece di calcolare automaticamente il valore di β soddisfacente la succitata proprietà. Meglio ancora, esso fornisce una lista contenente il valore minimo, indicato da [6] come γ_s, il valore massimo (δ_s) di β, e una variabile binaria (che può assumere cioè i due soli stati `True` o `False`) indicante se la procedura è andata 'in porto' in modo corretto. Nel caso non fosse stato trovato nessun valore di β, `BetaSearch` restituisce {`False, False, False`} . I parametri richiesti sono:

1. il valore minimo di β a partire dal quale si vuole far iniziare la ricerca;
2. il valore massimo di β in corrispondenza del quale arrestare la ricerca;
3. il numero di passi da compiere in fase di ricerca muovendo dal valore minimo a quello massimo (in linea di massima 100-150 possono andar bene);
4. la lista contenente le autocovarianze dei residui $c_p(j)$;
5. la lista dei valori medi dell'autocovarianza dei dati surrogati $c_v(j)$;
6. la lista delle varianze delle stime dell'autocovarianza dei dati surrogati $s_v(j)$.

È superfluo ricordare che l'intervallo entro cui l'algoritmo deve effettuare la ricerca (specificato dai primi due parametri) deve necessariamente includere l'intervallo $[\gamma_s,\delta_s]$. Se ciò non dovesse avvenire, `BetaSearch` tenta di segnalarlo ponendo `False` la variabile binaria. Conviene comunque essere molto accorti nella sua scelta. L'uso preventivo, per tentativi, di `SurrogatePlot` può essere d'aiuto.

Vediamo un caso pratico. Supponiamo che γ_s e δ_s siano:

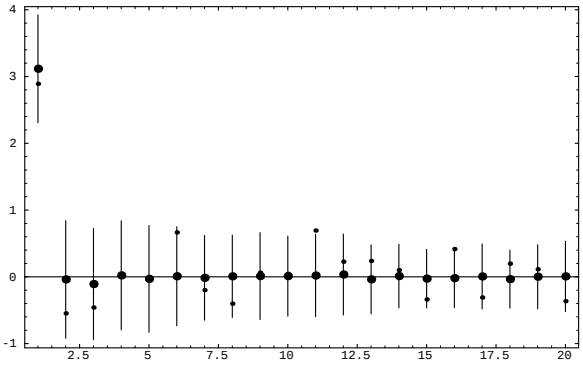

Fig. 9. Grafico ottenuto con la procedura `SurrogatePlot`.

```
In[]:=  BetaSearch[2.3,2.7,100,cp,cv,sv]
Out[]:= {2.35657, 2.56667, True}
```

La risposta che si ottiene nei casi in cui $[\gamma_s,\delta_s]$ non è incluso in $[\beta_{\min},\beta_{\max}]$ è la seguente:

```
In[]:=  BetaSearch[2.4,2.7,100,cp,cv,sv]
Out[]:= {2.4, 2.56667, False}
In[]:=  BetaSearch[2.3,2.5,100,cp,cv,sv]
Out[]:= {2.35657, 2.5, False}
In[]:=  BetaSearch[2.4,2.5,100,cp,cv,sv]
Out[]:= {2.4, 2.5, False}
In[]:=  BetaSearch[1.3,1.7,100,cp,cv,sv]
Out[]:= {False, False, False}
```

Per comodità e per (notevole) risparmio di tempo sono stati costruiti i files `mel20` e `mel40`, i quali contengono i valori medi delle autocovarianze (e relative varianze) dei dati surrogati per $M=20$ ed $M=40$ rispettivamente, e per un numero di autovalori che va da $p=2$ a $p=15$. Sono state impiegate 100 realizzazioni di un processo 'white noise', ciascuna di lunghezza pari a 150 dati (cfr. [6], p. 107). La procedura seguita per costruire `mel20` è stata la seguente[2]:

```
In[]:= Do[For[p=2,p<=15,p++,
{cv[p],sv[p]}=SurrogateAutocovariance[20,150,100,p];];
Save["mel20",cv,sv];]
```

Desiderando ottenere i grafici ad istogrammi che compaiono a p. 108 in [6], si possono utilizzare le istruzioni:

[2]La realizzazione del file `mel20` ha richiesto parecchie ore di calcolo!

```
In[]:= <<mel20;
In[]:= Do[x=P3[150];For[s=2,s<=15,s++,
  cp=ResidualAutocovariance[x,20,s];
  b=BetaSearch[.6,1.1,150,cp,cv[s],sv[s]];
  If[b[[3]],f=Join[{s},{b[[1]]},{b[[2]]}];
  PutAppend[TextForm[TableForm[f,TableDirections->{Row},
  TableSpacing->{2}]],"sval20.p3"];s=15]],{i,100}]
```

L'esempio si riferisce per la precisione al grafico c) di p. 108. Le 100 realizzazioni provengono infatti dal processo P3 con $M = 20$. Nel file `sval20.p3` vengono salvati i valori di S (definito sempre a p. 108), di γ_s e di δ_s. Si può quindi passare a realizzare il grafico ad istogrammi di Fig.10:

```
In[]:= <<statistics`datamanipulation`
       y=ReadList["sval20.p3",Table[Number,{3}]];
       s=Table[y[[i]][[1]],{i,Length[y]}];
       freq=BinCounts[s,{0.5,15.5,1}];
       BarChart[Transpose[{freq,Table[i,{i,1,15}]}],
                PlotRange->All,Frame->True,FrameLabel->
                {"Valori di S","Numerosita'"}, Axes->
                {True,None}]
```

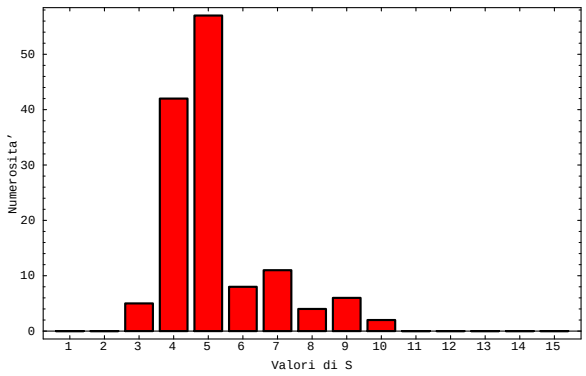

Fig. 10. Istogrammi della stima della dimensione statistica S per $M = 20$ e per il processo P3.

4 Comandi per la 'Singular-Spectral Analysis' multivariata.

I comandi relativi alla 'Singular-Spectral Analysis' multivariata sono sostanzialmente `MultiEigenSpectrumPlot` e `MultiRC`. Il loro uso è del tutto analogo a quello dei relativi comandi univariati, con le medesime opzioni `PlotType->"Eigenvalues"`, `PlotType->"Percentage"`, `PlotType->"CumulativePercentage"`, e `PlotType->"Differences"`.

Supponendo di lavorare su dati provenienti da un modello di Lorenz, i comandi da utilizzare diventano:

```
In[]:= x=Ld["lor.dat",3*500];
In[]:= MultiEigenSpectrumPlot[
                  {x[[1]],x[[2]],x[[3]]},{40,40,40}];
```

Il comando `MultiRC` permette di ricostruire la prima componente della lista (nell'esempio che segue `x[[1]]`), tenendo conto delle prime p (nell'esempio qui sotto, 12) componenti principali:

```
In[]:= y=MultiRC[{x[[1]],x[[2]],x[[3]]},{40,40,40},12];
```

Volendo ricostruire invece la terza componente, sarà quindi sufficiente scrivere:

```
In[]:= z=MultiRC[{x[[3]],x[[1]],x[[2]]},{40,40,40},12];
```

5 Previsione.

`LinearPrediction` adatta un modello autoregressivo alla serie x stimando i parametri con il metodo di Burg (1968) (cfr. [5]). Il comando richiede che venga specificata la serie x su cui 'fittare' il modello AR, l'ordine L della stima ed il numero di previsioni che si desiderano. Nell'esempio che segue generiamo una serie x di 250 valori utilizzando il processo P1 definito in precedenza: sui primi 220 'fittiamo' il modello AR e prevediamo i successivi 30 (il parametro L è stato posto pari a 15):

```
In[]:=  x=P1[250];
In[]:=  y=LinearPrediction[Take[x,220],15,30];
```

Mettiamo ora graficamente a raffronto le stime ottenute (`y`) con i valori veri (`Take[x,-30]`):

```
In[]:=ShowXY[Take[x,-30],y]
```

La Fig.11 mostra il risultato ottenuto.

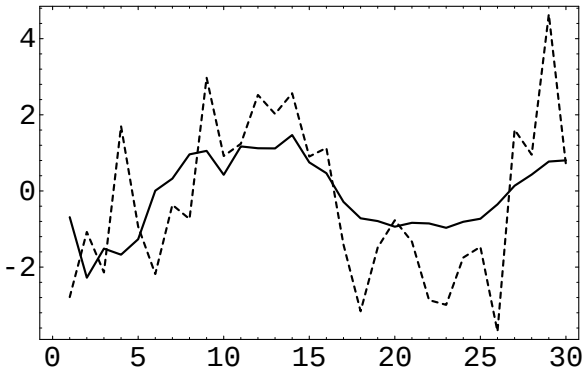

Fig. 11. Valori veri e stime ottenute con `LinearPrediction`.

L'errore quadratico medio di previsione normalizzato è la quantità:

$$\text{NMSE} = \frac{1}{\sigma^2} \cdot \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2, \qquad \text{dove} \quad \sigma^2 = \frac{\sum_{i=1}^{N} y_i^2}{N},$$

dove y_i è l'i-esima osservazione della grandezza y e $\hat{y}_i$ è il corrispondente valore stimato. L'errore NMSE è un indicatore del grado di 'bontà ' del metodo di previsione impiegato: se NMSE ≈ 1 significa che il metodo impiegato è in grado di prevedere solamente la media di y, mentre valori tanto più vicini allo zero indicano una capacità di previsione tanto migliore. Il comando `NMSE` calcola questo indicatore:

```
In[]:=  NMSE[Take[x,-30],y];
Out[]:= 0.6215
```

Il metodo proposto da [2], basato sulla SSA, per la previsione di serie temporali si compone delle seguenti fasi:

1. adattamento di un modello AR alle prime p componenti ricostruite utilizzando le stime presentate in Burg (1968);
2. previsione, per ciascuna delle singole componenti, del valore x^k_{t+T} che la variabile x^k assumera' dopo T periodi dall'ultima rilevazione;
3. stima del valore di x_{t+T} come somma dei p valori precedentemente ottenuti.

Le 3 fasi del metodo di [2] sono raccolte nel comando `RCLinearPrediction[x,M,L,p,T]`, dove x è la serie da analizzare, M il numero di lag impiegati nella SSA, L l'ordine del modello AR (che [6] suggerisce, per l'analisi SSA, di porre pari a M), p il numero di componenti principali da considerare, e T il numero di previsioni che si vogliono ottenere (cioè le previsioni per T periodi dopo l'ultima osservazione x_{t_f}).

Facendo uso dei risultati ottenuti nell'esempio precedente e ponendo $M = 15$, $L = 15$, $p = 4$, $T = 30$, possiamo scrivere:

```
In[]:=  z=RCLinearPrediction[Take[x,220],15,15,4,30];
```

Il valore di NMSE si ottiene agevolmente digitando:

```
In[]:=  NMSE[Take[x,-30],z];
Out[]:= 0.522451
```

La Fig.12 confronta i valori ottenuti con questo metodo di previsione e i valori veri.

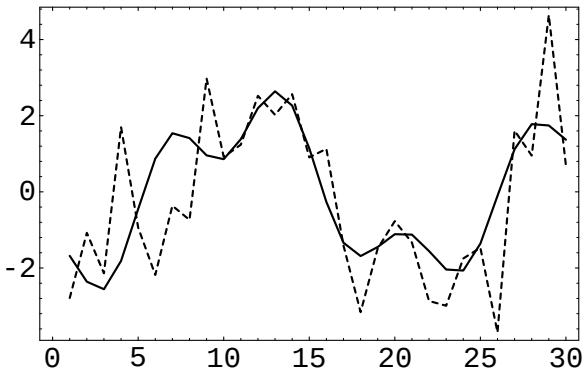

Fig. 12. Valori veri e stime ottenute con `RCLinearPrediction`.

References

1. Broomhead, D.S. and King, G.P. (1986), *Extracting Qualitative Dynamics from Experimental Data*, Physica D, 20, pp. 217-236.
2. Keppenne, C.L. and Ghil, M. (1992), *Adaptive Filtering and Prediction of the Southern Oscillation Index*, Journal of Geophysical Research, 97 (D18), pp. 449-454.
3. Keppenne, C.L. and Ghil, M. (1993), *Adaptive Filtering and Prediction of Noisy Multivariate Signals: an Application to Subannual Variability in Atmospheric Angular Momentum*, Internation Journal of Bifurcation and Chaos, 3 (3), pp. 625-634.
4. Maeder, R. (1991), *Programming in Mathematica. Second Edition*, Addison Wesley, Redwood City.
5. Press, W.H., Teukolsky, S.A., Vetterling, W.T. and Flannery, B.P. (1992), *Numerical Recipes in C. The Art of Scientific Computing. Second Edition*, Cambridge University Press, Cambridge.
6. Vautard, R., Yiou, P. and Ghil, M. (1992), *Singular-Spectrum Analysis: A Toolkit for Short, Noisy and Chaotic Signals*, Physica D, 58, pp. 95-126.
7. Wagon, S. (1991), *Mathematica in Action*, W.H. Freeman and Co., Inc., New York.
8. Wolfram, S. (1991), *Mathematica. A System for Doing Mathematics by Computer. Second Edition*, Addison Wesley, Redwood City, California.