

Fabio Marcuzzi

Numerical Linear Algebra and Learning from Data

- 1st online edition -

Preface

Pordenone and Padova, September 2021

Fabio Marcuzzi
marcuzzi@math.unipd.it

Indice

Preface	0
1 Matrix factorizations	11
1.1 Why Matrix Factorizations ?	12
1.1.1 To solve linear systems	12
1.1.2 To analyze matrices of data	12
1.2 Columns-M-Rows (CMR) factorization	13
1.3 Numerical methods for QR factorization and SVD	13
1.3.1 Modified Gram-Schmidt	15
1.3.2 The Householder method	15
1.3.3 Backward error analysis of the algorithms that compute the QR factorization	17
1.3.4 Updating the QR factorization after row insertion	18
1.3.4.1 Givens transformations	19
1.3.5 Updating the QR factorization after rank-1 perturbation	20
1.4 The Singular Value Decomposition (SVD)	21
1.4.1 Geometric interpretation of the SVD	22
1.4.2 Relevance of the SVD in numerical computing	22
1.4.3 Computing the SVD	24
1.4.4 Analysis of linear transformations with the SVD	26
1.4.5 Truncated SVD (TSVD)	27
1.4.6 Applicative example: eigenfaces	28
1.5 Nonnegative Matrix Factorizations (NMF)	28

2 Linear dimensionality reduction of data and models	33
2.1 Principal Component Analysis (PCA)	34
2.1.1 How to compute principal components	35
2.2 Static linear regression models in the ideal (noiseless) case	36
2.2.1 Linear dimensionality reduction of static linear regression models	37
2.3 Discrete-time Linear Time-Invariant (DLTI) dynamical systems in the ideal (noiseless) case	37
2.3.1 The DLTI model as a convolution sum	37
2.3.2 ARMA models and the theory of difference equations	40
2.3.3 State-space models	42
2.3.4 Explicit conversions from different representations of DLTI systems	43
2.3.5 Model identifiability	45
2.4 Linear dimensionality reduction of state-space DLTI models	45
2.4.1 Balanced truncation	46
2.4.2 Proper Orthogonal Decomposition (POD)	47
2.4.3 Modal truncation	49
3 Linear models estimation and learning	51
3.1 Static linear regression models	51
3.1.1 Standard linear model	51
3.1.2 Generalized linear model (Gauss-Markov)	52
3.1.3 Error in-variable model	52
3.2 Discrete-time Linear Time-Invariant (DLTI) dynamical systems	52
3.2.0.1 Stochastic ARMA model	52
3.2.0.2 Stochastic state-space models	53
3.3 Numerical methods for ordinary least-squares	53
3.3.1 Normal equations method	54
3.3.2 Method with QR factorization	54
3.3.3 A better look at the Pseudo-inverse of a matrix	54
3.3.4 Method with the SVD	55
3.3.5 Choice of the algorithm	56
3.4 Numerical methods for Total Least Squares	56

3.5	Numerical methods for least-squares problems with linear constraints	57
3.5.1	Direct algorithms for equality constrained problems: the null-space method	58
3.5.2	Iterative algorithms for inequality constrained problems: <i>active-set</i> methods	60
3.5.2.1	Active-set methods for least-squares with bounds on variables	62
3.5.2.2	Active-set methods for Non-Negative Least Squares (NNLS)	62
4	Ill-posedness and rank-estimation, Ill-conditioning and regularization	65
4.1	Analysis of singular values of a matrix A	66
4.2	Truncated SVD	67
4.2.1	Solution of ill-conditioned or ill-posed least-squares problems	67
4.3	Tikhonov regularization	68
4.4	Choice of the regularization parameter: Bias vs Variance and the Discrepancy Principle	69
4.4.1	Il metodo dello stimatore unbiased del rischio predittivo . .	70
4.4.2	The Discrepancy Principle	70
4.5	Alternative approaches to the SVD for rank-deficient problems: rank-revealing QR	71
5	Numerical methods for nonlinear least-squares problems	75
5.1	Newton's method	76
5.2	Metodo di Gauss-Newton	78
5.3	The method of Levenberg-Marquardt	78
5.3.1	Choice of $\lambda^{(k)}$	79
5.3.2	Choice of $\Delta^{(k)}$	79
6	Sparse solutions of underdetermined linear systems	81
6.1	Basic concepts and properties	83
6.1.1	Null Space Property (NSP)	83
6.1.2	The <i>spark</i> of a matrix	84
6.1.3	Mutual coherence	84
6.2	Mixed 1- and 2-norm regularization (<i>sparse recovery</i>)	84

7 Analisi di Fourier nel discreto	87
7.1 Sequenze di Dati, Segnali a tempo discreto e Serie storiche	87
7.2 Campionamento di segnali continui	89
7.3 La Trasformata di Fourier Discreta (DFT)	90
7.4 Rappresentazione dei Segnali Discreti mediante DFT	92
7.4.1 Sequenze di Dati Periodiche di Durata Illimitata	92
7.4.2 Sequenze di Dati Aperiodiche di Durata Illimitata	93
7.4.3 Sequenze di Dati di Durata Limitata	94
7.4.4 Sequenze di Dati ad Energia Infinita	95
7.4.5 Correlazione e Covarianza di Segnali Discreti	95
7.4.5.1 Aleatorietá nei Dati	96
7.4.5.2 Stima dell'aspettazione	96
7.4.6 densitá spettrale di potenza	97
7.4.7 Esempi applicativi	97
7.5 Rappresentazione dei Sistemi DLTI mediante DFT	98
7.5.1 Risposta in Frequenza di un Sistema DLTI	98
7.5.2 Trasformata z e Funzione di Trasferimento	100
7.6 Calcolo della DFT: algoritmo della Fast Fourier Transform (FFT) .	100
7.6.1 Algoritmo della FFT per segnali mono-dimensionali	101
7.6.1.1 Una semplice riduzione del numero di operazioni	104
7.6.1.2 Il calcolo <i>sul posto</i>	105
7.6.1.3 La gestione degli indici	106
7.7 Utilizzo in pratica della DFT	108
7.7.1 Rappresentazione grafica della DFT	108
7.7.2 Il problema del <i>leakage</i> e la finestratura dei dati	109
7.7.3 il problema dell'effetto <i>picket fence</i> e la tecnica di <i>zero-padding</i>	110
7.7.3.1 Risoluzione in frequenza	110
7.7.4 Presenza di rumore nei dati	111
7.8 Stima dello Spettro di Potenza di un Segnale	111
7.8.1 Test di bianchezza di Bartlett	113
7.9 La DFT e l'algoritmo FFT per segnali bi-dimensionali	114

8 Analisi tempo-frequenza	117
8.1 Short-Time Fourier Transform (STFT)	117
8.1.1 Esempio di spettrogramma: ricostruzione di una partitura musicale	119
8.1.2 Separazione di sorgenti da uno spettrogramma	119
8.2 La Trasformata Wavelet	121
8.2.1 Le wavelets di Haar	122
8.2.2 Le wavelets della Daubechies [9]	124
8.2.3 Le <i>Coflets</i>	125
8.2.4 Conservazione e compattamento dell'energia	125
8.2.5 Interpretazione della trasformata wavelet come banco di filtri [27]	125
8.2.6 Analisi in frequenza delle wavelets	126
8.2.7 Wavelets 2-d	127
9 Identification of linear systems	129
9.1 Identificazione di modelli lineari statici	129
9.1.1 Stima dei parametri del modello lineare standard	129
9.1.2 Determinazione dell'ordine del modello dai dati sperimentali	130
9.1.2.1 Il caso del modello lineare (regressione lineare)	131
9.1.3 Stima dei parametri del modello lineare generalizzato	132
9.1.4 Stima dei parametri di modelli lineari flessibili	132
9.1.5 Misurare la bontà delle stime	133
9.1.6 Stima di parametri tempo-varianti	134
9.2 Identificazione di sistemi DLTI da dati privi di rumore	135
9.2.1 Caso di risposta all'impulso discreto e rappresentazione ARMA	136
9.2.2 Caso di risposta all'impulso discreto e rappresentazione <i>statespace</i>	137
9.2.3 <i>Eccitazione persistente</i>	138
9.2.4 Caso di coppie ingresso/uscita qualsiasi e rappresentazione ARMA	139
9.2.5 Caso di coppie ingresso/uscita qualsiasi e rappresentazione <i>state-space</i>	140
9.3 Identificazione di modelli DLTI da dati con rumore	141

9.3.1	Analisi dei valori singolari e presenza di rumore nei dati	141
9.3.2	Determinazione dell'ordine di un modello ARX	141
9.3.2.1	L'analisi dei valori singolari	142
9.3.2.2	Criteri di parsimonia	142
9.4	Analisi parametrica di <i>serie storiche</i>	143
9.4.1	Scomposizione della serie storica	143
9.4.2	Predizione lineare	144
10	Stima dello stato di sistemi dinamici	147
10.1	Stima ricorsiva dello stato di modelli DLTI nella rappresentazione <i>state-space</i>	147
10.1.1	costruzione dell'osservatore	148
10.2	Stima dello stato in presenza di rumore di modello e di misura: il Filtro di Kalman	149
10.2.1	Equazioni del filtro di Kalman	149
10.2.2	Accordo sperimentale del filtro	152
10.2.3	Implementazioni Square-Root del filtro di Kalman	152
10.3	Stima dello stato iniziale in sistemi evolutivi nonlineari	153
10.3.1	Discretizzazione nel tempo	154
10.3.2	Stima delle condizioni iniziali (stato iniziale)	155
10.3.3	Il metodo del modello aggiunto	155
10.3.4	Procedura di ottimizzazione con il modello aggiunto	158
10.3.5	Esempio di Lorenz	159
11	Stima dei parametri nelle reti neurali	163
11.1	Reti neurali	163
11.1.1	Modello non-lineare <i>perceptron</i>	163
11.1.2	Modello non-lineare "multi-layer neural network"	165
11.1.3	Modello non-lineare <i>deep neural network</i>	166
11.2	L'algoritmo di <i>back-propagation</i>	167
11.3	Procedura di apprendimento, iper-parametri, convergenza	171
11.4	Metodi del second'ordine: metodo di Newton	171
11.4.1	Regolarizzazione l_2 dei modelli neurali	172
11.5	Metodi del gradiente	172

11.5.1	convergenza del gradiente stocastico	172
11.5.2	<i>Momentum Algorithm</i>	173
11.5.3	<i>Adattamento del learning rate</i>	174
11.6	Metodi per le reti convoluzionali	174
11.6.1	Applicazioni	175
A	Analisi dei sistemi DLTI <i>state-space</i>	177
A.1	Analisi modale	178
A.2	Raggiungibilità	179
A.3	Osservabilità	180
A.4	Realizzazione minima	181
B	Soluzione numerica di problemi differenziali evolutivi	183
B.1	Fenomeni diffusivi	183
B.2	Fenomeni di trasporto e propagazione di onde	185
B.3	Soluzione numerica delle equazioni della dinamica di un sistema meccanico elastico lineare	185
B.3.1	Integrazione nel tempo	185
B.3.2	Metodi per sistemi di equazioni differenziali del primo ordine nel tempo	186
B.3.2.1	Eulero esplicito (forward Euler, $\theta = 0$)	187
B.3.2.2	Eulero implicito (backward Euler, $\theta = 1$)	187
B.3.2.3	Crank-Nicholson ($\theta = \frac{1}{2}$)	187
I	Computing Lab	189
C	Dati quantitativi e modelli	193
C.1	Segnale discreto mono-dimensionale	193
C.2	Oscillazioni dell'esponenziale discreto	195
C.3	Fenomeno dell'aliasing	202
C.4	Osservazione dell'aliasing nel dominio delle frequenze	211
C.5	Esempi di sistemi LTI	220
C.6	Esempio ARMA	220
C.7	Esempio state-space a tempo continuo	220

C.8 Esempio state-space a tempo discreto	221
C.9 Sistema delle tre masse	222
C.10 Sistema delle tre masse (implementazione ad oggetti)	224
D Algoritmi fattorizzazione QR	227
D.1 Esempio di generazione di matrice random con condizionamento assegnato	227
D.2 Esempio di trasformazione di Householder	228
D.3 Confronto tra gli algoritmi di fattorizzazione QR	229
D.4 Esempio di trasformazione di Givens	231
D.5 Rumore e componenti singolari	233
D.6 Confronto algoritmi per il problema dei minimi quadrati	237
E Regolarizzazione	243
E.1 Esempio SVD troncata di un'immagine	243
E.2 Analisi della regolarizzazione con TSVD e Tichonov	249
F Esempio troncamento modale	273
G Esempio PCA	281
G.1 PCA con matrici di correlazione e di covarianza	285
H Minimi quadrati nonlineari	291
H.1 Esempio sulla convergenza globale	291
H.2 Esempio non unicità minimi quadrati nonlineari	297
I Esempi con sistemi sotto-determinati	303
J Analisi di Fourier stazionaria	319
J.1 Esempi risposta in frequenza	319
J.2 Algoritmo FFT	323
J.3 Aspetti pratici DFT 1-d	325
J.4 Finestratura	329
J.5 Effetto picket-fence	333
J.6 DFT in presenza di rumore nei dati	341
J.7 Esempio DFT 2-d	354

J.8	Esempi stima spettro di potenza	356
J.9	Esempi test di Bartlett	361
J.10	Esempio di filtraggio vs regolarizzazione	364
J.11	Esempio ECG	372
K	Esempio segnali non-stazionari	379
L	Aspetti pratici STFT 1-d	385
M	Esempio ricostruzione partitura	389
N	Intro wavelets	403
O	Disegna wavelets	411
P	Multi-risoluzione	415
Q	Analisi fluttuazioni wavelets	423
R	Esempio analisi segnali audio con wavelets	433
S	Esempi wavelets 2-d	445
T	Esempio MRI	451
U	Esempio scelta variabili modello lineare standard	457
U.1	Esempio Hodrick-Prescott	507
V	Stima parametri DLTI state-space dalla risposta all'impulso discreto	511
W	Stima parametri DLTI ARMA tempo-varianti dalla risposta all'impulso discreto	515
X	Stima parametri DLTI ARMA dalla risposta all'impulso discreto	519
Y	Stima parametri DLTI ARMA dalla risposta all'impulso discreto con rumore	523
Y.1	Esempio analisi di serie storiche	689
Z	Esempi di apprendimento neurale	707
Z.1	Esempio stima dello stato	710

Capitolo 1

Matrix factorizations

Let us start with some basic facts about matrix operations. These facts are important for *understanding*, not for *computing*:

- **matrix-vector product** $A x$ is a linear combination of the columns of A with coefficients x_i :

therefore, said $Ax = b$, the vector b is expressed against a “**basis**” composed by the columns of A .

Note that, instead, we compute $Ax = b$ by multiplying x in its *row space* (i.e., dot-products between x and A rows).

- **matrix-matrix product** $A B = C$ is a **generalization**, i.e. several combinations at once: the columns of the left matrix are the “basis” and the columns of the right matrix are the coefficients to form the corresponding column of C . Examples:

$L U$,

$Q R$,

$$(U\Sigma^{1/2}) (\Sigma^{1/2}V^T)$$

- **outer vector product** $uv^T = M$ is a **rank-1** matrix whose columns are the same column u multiplied by v_i :

rank-1 matrices are used in sums of matrices (as we will see in the dyadic decomposition obtained from with the SVD).

1.1 Why Matrix Factorizations ?

1.1.1 To solve linear systems

Linear systems $Ax = b$ arise e.g. in empirical modelling:

in experimental modelling with differential equations:

in nonlinear estimation (the matrix is usually a Jacobian, i.e. a “derivative”, usually ill-conditioned):

In all these contexts, the matrix is expressed in a meaningful column-basis and must be inverted. Therefore, arise also some numerical difficulties: computational cost and stability.

LU factorization helps at reducing the cost of inversion, but the columns of L as a basis are of dubious interpretability:

Often, in data-related applications, the matrix A is rectangular (tall or flat) and, therefore, not invertible.

The immediate solution $A^T A$ is terrible from numerical point of view (as will be clear very soon):

A QR factorization is a stable way to get the least-squares solution (as we will see), with an orthonormal basis of the range of A , without increasing too much the computational cost:

1.1.2 To analyze matrices of data

Data are often put into matrices or even tensors; e.g. facial images:

and matrix factorizations are a main tool to extract some insight from a set of pixels.

1.2 Columns-M-Rows (CMR) factorization

Given a matrix A of known rank r , let us create a matrix C whose columns come directly from A and are mutually independent, i.e. we do not include columns that are linear combinations of previous columns. The number of such columns is equal to the rank of A , the dimension of its column space.

Now we create a matrix R , whose columns are the coefficients that describe each column of A as a linear combination of the columns of C . Its existence is guaranteed from the fact that C contains a basis for the column space of A . Then,

$$A = CR \quad (1.1)$$

and R will contain “ r ” columns of the identity matrix.

Remember that the rank is also the dimension of the row space of A , i.e. the number of independent columns is equal to the number of independent rows. Actually we could do the same by taking the linearly independent rows of A , put them on R^r and compute C^r as the coefficients that describe each row of A as a linear combination of the rows of R^r . Then, again, $A = C^r R^r$.

Now, C has columns extracted from A but R has no rows corresponding to A . This can be achieved by adding a “mixing matrix” M :

$$A = CMR \quad (1.2)$$

Exercise: derive a formula to compute M ...

Note that C has no **orthogonal** columns and R has no **orthogonal** rows. Indeed, this will happen with the SVD!

Despite the optimal numerical properties of QR and SVD, the CMR factorization has gained some importance in data analysis, simply because C and R **keep the original data**, i.e. if A is nonnegative and/or sparse, so are C and R .

1.3 Numerical methods for QR factorization and SVD

Let $A \in \mathbb{R}^{m \times n}$, $m \geq n > 0$. If it has full column rank, it means that the columns of A generate a space of dimension n . These column vectors are therefore linear independent, but not necessarily mutually orthogonal.

Exercise: how to verify this property ? Hint: compute $A^T A$... you get the identity matrix ?

The **QR factorization** [12] [29] [11] [24] [3] determines an orthonormal basis, contained in the matrix Q , for the column space of A , $\text{range}(A)$, and represents in the

matrix R the coordinates of the columns of A with respect to such an orthonormal basis:

$$QR = A$$

where $A \in \Re^{m \times n}$, $Q \in \Re^{m \times n}$ is formed by n columns mutually orthogonal and of unitary length, and $R \in \Re^{n \times n}$ is upper triangular. The QR factorization is unique if A has full column rank.

Exercise: how is it changed from a column permutation ? Hint: build a column permutation matrix P_c ...

This is called *reduced QR*; if we add $m - n$ columns to Q mutually orthogonal, of unitary length and orthogonal also with respect to the previous Q columns, and if we add also $m - n$ null rows to R , we obtain the *complete QR factorization*: the matrix Q is square and orthogonal even if the matrix A be rectangular. Note that the columns added to Q are an orthonormal basis for the orthogonal complement of the column space of A , i.e. $\text{range}(A)^\perp$.

The QR factorization is computed by a **direct** algorithm, not an iterative one. There are two possible approaches:

1. to build explicitly Q with an orthonormalization process (*Modified Gram-Schmidt*), that is a **triangular orthogonalization**; in practice, it consists of right multiplying A times a sequence of upper triangular matrices: $A \cdot R_1 R_2 R_3 \dots R_n = Q$;
2. to build explicitly R by applying orthogonal transformation matrices (typically Householder or Givens matrices), that is a **orthogonal triangulation**; in practice, it consists of left multiplying A times a sequence of orthogonal matrices: $Q_n \dots Q_3 Q_2 Q_1 \cdot A = R$, and then:

$$Q^T = Q_n \dots Q_3 Q_2 Q_1 : \quad (1.3)$$

Let us see an adequate numerical method for both approaches, i.e. Modified Gram-Schmidt (par. 1.3.1) and Householder (par. 1.3.2). Recall that there is a precise relation between the internal (dot) product $x^T y$ of two (column) vectors x e y and the angle α between them [29]:

$$\cos \alpha = \frac{x^T y}{\|x\| \|y\|} . \quad (1.4)$$

Recall the definition and some properties of projection matrices:

- a projection matrix (or *projector*) is a square matrix that satisfies: $P^2 = P$ (i.e. *idempotent*);

- if $v \in Range(P)$, i.e. $v = Px$, then: $Pv = P^2x = Px = v$ and therefore v coincides with its “shadow”;
- if P is a projector, also $I - P$ it is. Indeed: $(I - P)^2 = I^2 - 2P + P^2 = I - P$ and it projects over $Ker(P)$, the null space of P ;

1.3.1 Modified Gram-Schmidt

The Gram-Schmidt is often used in numerical linear algebra, but not in the form usually written in linear algebra textbooks, that here we call the “classic” form, or “CGS”. This is numerically unstable (see sec. 1.3.3). A better alternative is the *modified Gram-Schmidt method*, or “MGS”. The difference is in the way projections over previous directions $q(:, i)$ are computed: from the original column $a(:, j)$ in the classic form and from the current column $q(:, j)$ in the modified form. In the following implementation, this difference is contained in a unique instruction: commenting one of the two alternative instructions $r(i, j) = \dots$ one obtains one form or the other.

```

for j=1:n                                % calcola la j-esima colonna di Q ed R
    q(:,j) = a(:,j);
    for i=1:j-1
        r(i,j) = q(:,i)'*a(:,j);      % G.S.
        r(i,j) = q(:,i)'*q(:,j);      % M.G.S.

        q(:,j) = q(:,j) - r(i,j)*q(:,i);
    end;
    r(j,j) = norm(q(:,j));           % norma-2
    if r(j,j)==0 break; end;         % significa che c'e' dipendenza lineare
    q(:,j) = q(:,j)/r(j,j);         % cosi' Q ha colonne ortogonali e di norma-2
                                    % unitaria, e dunque e' ortonormale
end;

```

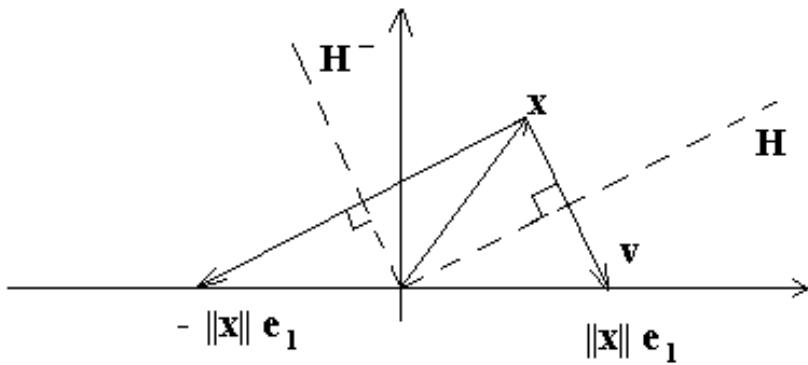
1.3.2 The Householder method

The Householder method defines the matrix Q_k , previously cited, in this way:

$$Q_k = \begin{bmatrix} I_{k-1} & 0 \\ 0 & F_{k,x} \end{bmatrix} \quad (1.5)$$

with $F_{k,x}$ an orthogonal matrix such that, for $x \in \Re^{m-k+1}$, $F_{k,x}x = \begin{bmatrix} \|x\| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = \|x\| \cdot e_1$:

By construction, the hyperplane H is uniquely determined by imposing its orthogonality to the vector $v = \|x\| \cdot e_1 - x$. Then,



$$P = I - \frac{vv^T}{v^Tv} \quad (1.6)$$

projects x su H :

$$Px = x - \frac{vv^T x}{v^Tv} = x - \frac{v \cos \alpha \|v\| \|x\|}{\|v\| \|v\|} = x - \cos \alpha \|x\| \frac{v}{\|v\|} \quad (1.7)$$

, while

$$F_{k,x} = I - 2 \frac{vv^T}{v^Tv} \quad (1.8)$$

reflects it through H .

Exercise: demonstrate that $F_{k,x}$ is an orthogonal matrix.

To avoid cancellation errors, one chooses the plane more distant from x , and therefore

$$v = -\text{sign}(x_1) \|x\| e_1 - x \quad (1.9)$$

The algorithm computes R in-place of A , and the Householder vectors v_k that univocally define the matrices Q_k :

```

for k=1:n
    %
    x = A(k:m,k);
    v_k = sign(x(1))*norm(x)*e_k + x; % "e_k" rappresenta il k-esimo asse
                                                % del sistema di coordinate
                                                % v_k e' stato cambiato di segno per
                                                % maggior efficienza del calcolo
    v_k = v_k / norm(v_k);
    A(k:m,k:n) = A(k:m,k:n) - 2*v_k*v_k'*A(k:m,k:n);
end;

```

Let us see an example: D.2

Usually Q is not explicitly required, but products Qb or $Q^T b$, with an arbitrary (known) b ; to compute these products it is sufficient to apply to b the sequence of Householder vectors that has been applied to A (1.3):

```
for k=1:1:n  b(k:m) = b(k:m) - 2*v_k*(v_k'*b(k:m));    % Q'*b
for k=n:-1:1  b(k:m) = b(k:m) - 2*v_k*(v_k'*b(k:m));    % Q*b
```

Moreover, to eventually compute the matrix Q it is possible to apply these products column-by-column, with b equal to the i -th column of the identity matrix, since $QI = Q$.

1.3.3 Backward error analysis of the algorithms that compute the QR factorization

The *backward error analysis* has been introduced by Wilkinson in the 60s of the XX century. Suppose we want to compute $f(\xi)$, Wilkinson has developed a systematic approach to find $\tilde{\xi}$ such that

$$\tilde{f}(\tilde{\xi}) = f(\tilde{\xi}) = f(\xi + \delta\xi) \quad (1.10)$$

i.e. the result $\tilde{f}(\tilde{\xi})$ of the finite-length computation is found equal to the exact result (infinite precision computation) of a perturbed problem, with an indication of the magnitude of this perturbation. The effectiveness of this approach comes evident in combination with the available theoretical results on sensitivity analysis of the solution with respect to problem data. This usually involves the *condition number* of the problem. As an example, let us remember the result for the solution of a linear system of equations with a perturbed right-hand-side:

$$\frac{\|\delta x\|}{\|x\|} \leq \mathcal{K}_A \frac{\|\delta b\|}{\|b\|} \quad (1.11)$$

where \mathcal{K}_A is the condition number of the matrix A ; the backward error analysis (1.10) gives us for this case $\delta\xi$ ($= \delta b$) and, from (1.11), we have this upper bound for the error $\|\tilde{x} - x\| = \|\delta x\|$.

A numerical algorithm is said *backward stable* if:

$$\|\xi - \tilde{\xi}\| \leq \epsilon \|\xi\| \quad (1.12)$$

where ϵ is the *machine precision*.

With QR factorization, $\tilde{Q}\tilde{R} = A + \delta A$ and it is demonstrated that, with Householder transformations or M.G.S.:

$$\frac{\|\delta A\|}{\|A\|} \leq O(\epsilon) \quad (1.13)$$

i.e. these methods are backward stable, while C.G.S. is unstable.

Looking at the deviation from the orthogonality, it holds [Bjorck]:

$$\tilde{Q}^T \tilde{Q} = I + E$$

with a backward error, E , that satisfies:

$$\begin{aligned}\|E\|_2 &\approx \epsilon \cdot K_2(A) && \text{with M.G.S.} \\ \|E\|_2 &\approx \epsilon && \text{with Householder}\end{aligned}$$

and, therefore, the deviation from orthogonality of the matrix \tilde{Q} with the modified Gram-Schmidt method is small only if the matrix A is not too much ill-conditioned, while the Householder method retains always the machine precision.

To compare these methods, we can operate in the following way: [D.3](#)

1.3.4 Updating the QR factorization after row insertion

Let us suppose to have already computed the *complete* QR factorization of $A_w^N \in \mathcal{R}^{N \times n}$ (it means that we have computed also a block of $N - n$ orthonormal columns to complete a base of R^N , added it to Q and added just as many null rows to R):

$$A_w^{(N)} = Q_w^{(N)} \cdot R_w^{(N)} = Q_w^{(N)} \cdot \begin{bmatrix} R_- \\ 0^T \\ \vdots \\ 0^T \end{bmatrix} \quad (1.14)$$

We want now to compute the QR factorization of A_w^{N+1} :

$$A_w^{(N+1)} = Q_w^{(N+1)} \cdot R_w^{(N+1)} = Q_w^{(N+1)} \cdot \begin{bmatrix} R_+ \\ 0^T \\ \vdots \\ 0^T \end{bmatrix} \quad (1.15)$$

reusing as much as possible the computations already did at the previous step. To this aim, let us note that (here we can set $\lambda = 1$; a different value is used when we desire an exponential forgetting of old data, as we will do later):

$$A_w^{(N+1)} = \left[\frac{\lambda \cdot A_w^{(N)}}{a_{N+1}^T} \right] = \left[\frac{Q_w^{(N)} \quad 0^T}{0 \quad 1} \right] \cdot \left[\frac{\lambda \cdot R_w^{(N)}}{a_{N+1}^T} \right] \quad (1.16)$$

Therefore, to find the factorization QR of A_w^{N+1} it is sufficient to factorize the matrix

$$\tilde{R} = \begin{bmatrix} \lambda \cdot R_- \\ 0^T \\ \vdots \\ 0^T \\ \hline a_{N+1}^T \end{bmatrix} \quad (1.17)$$

which is actually close to an upper triangular matrix, except for the last row. The only necessary operation is therefore to zero all the elements of the last row. This can be done by applying n ($\ll N$) Givens (orthogonal) transformations.

The problem of updating the QR factorization after appending a row can be solved applying n ($\ll N$) times a Givens transformation, in such a way that the n elements of the last row of (1.17) be zeroed.

1.3.4.1 Givens transformations

The Givens transformations, called also rotation matrices, are orthogonal matrices of this kind:

$$Q_{pq} = \begin{pmatrix} 1 & & & & \\ & \cos\theta & & -\sin\theta & \\ & & 1 & & \\ & & & \ddots & \\ & & & & 1 \\ & \sin\theta & & \cos\theta & \\ & & & & 1 \end{pmatrix} \leftarrow p \quad (1.18)$$

and $x' = Q_{pq}x$ is the vector x rotated of θ , counter-clockwise, in the plane p, q . Indeed, the transformation involves only the components of indexes p and q , as is easy to verify, and we have, writing the projection of x in the plane $p - q$:

$$(x_p, x_q) = (l \cos \alpha, l \sin \alpha)$$

where α is the angle between the projection with p axis. We have:

$$(x'_p, x'_q) = (l \cos(\alpha + \theta), l \sin(\alpha + \theta)) = (l(\cos \alpha \cos \theta - \sin \alpha \sin \theta), l(\sin \alpha \cos \theta + \cos \alpha \sin \theta))$$

for the trigonometric addition formulas, and therefore:

$$\begin{bmatrix} x'_p \\ x'_q \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{bmatrix} \begin{bmatrix} x_p \\ x_q \end{bmatrix} \quad (1.19)$$

that coincides with (1.18) produced by the Givens matrix.

Let us see an example: D.4.

It is quite easy to compute the values of $\sin(\theta)$ and $\cos(\theta)$ in such a way that the q-th component of $Q_{pq}x$ be zero, and without the need to apply inverse trigonometric formulas, since it is not necessary to compute θ . We have:

$$\cos(\theta) = \frac{x_p}{\sqrt{x_p^2 + x_q^2}} \quad \sin(\theta) = \frac{-x_q}{\sqrt{x_p^2 + x_q^2}} \quad (1.20)$$

The same can be done with the q-th component of the p-th column of $Q_{pq}\tilde{R}$, when \tilde{R} is the (1.17), $q = N + 1$ and $p = 1, 2, \dots, n$.

An algorithm to compute $\sin(\theta)$ and $\cos(\theta)$, robust against overflow, is the following: `function [c,s] = givens_coeff(xp,xq)`

```

if xq=0
    c=1; s=0;
else
    if abs(xq) > abs(xp)
        r = xp/xq; s=-1/sqrt(1+r^2); c=-s*r;
    else
        r = xq/xp; c=1/sqrt(1+r^2); s=-c*r;
    end;
end;
```

Note that, while Householder matrices put to zero all components below a chosen diagonal, the Givens rotations put to zero a single component. For this reason, they are used in various contexts, e.g. with sparse matrices, other than in perturbed problems.

1.3.5 Updating the QR factorization after rank-1 perturbation

Suppose that the QR factorization $A = QR$ is available, and let \bar{A} be a rank-one modification of A :

$$\bar{A} = A + uv^T \quad (1.21)$$

with $u \neq 0$ and $v \neq 0$.

Multiplying \bar{A} by Q^T , we obtain:

$$Q^T \bar{A} = Q^T (A + uv^T) = R + Q^T uv^T = R + wv^T \quad (1.22)$$

i.e. a rank-one modification of R , where $w = Q^T u$.

Now, we build an orthogonal matrix \tilde{Q}^T that transforms $R + wv^T$ to an upper triangle \bar{R} :

$$\tilde{Q}^T (R + wv^T) = \bar{R} \quad (1.23)$$

so that \tilde{Q} and \bar{R} are the QR factors of $R + wv^T$.

Using (1.22), we have:

$$\bar{R} = \tilde{Q}^T \left(R + wv^T \right) = \tilde{Q}^T Q^T \bar{A} \quad (1.24)$$

and

$$\bar{A} = \bar{Q} \bar{R} \quad , \quad \bar{Q} = Q \tilde{Q} \quad (1.25)$$

To find \tilde{Q} , we first build a sequence of rotations S_1 , in the $(n, n-1)$, $(n, n-2)$, ..., $(n, 1)$ planes, such that

$$S_1 w = \gamma e_n \quad , \quad |\gamma| = \|w\|_2 \quad (1.26)$$

and, therefore

$$S_1 w v^T = \gamma e_n v^T \quad (1.27)$$

which is a matrix of zeros except the last row which is equal to v .

Now, S_1 applied to R leave unchanged the upper-triangular form of the first $n-1$ rows and create an entirely dense n -th row. Therefore, even R' :

$$R' = S_1 R + S_1 w v^T = S_1 \left(R + w v^T \right) \quad (1.28)$$

it is also upper-triangular with a full last row.

At this point, we apply a sequence of rotations S_2 to put to zero each element of the last row, as seen in sec. 1.3.4. Then,

$$\tilde{Q}^T = S_2 S_1 \quad (1.29)$$

1.4 The Singular Value Decomposition (SVD)

Given a matrix $A \in \Re^{m \times n}$, $\text{rank}(A) = r$, it exists at least one factorization of this kind:

$$A = U \Sigma V^T \quad (1.30)$$

where $U \in \Re^{m \times m}$ and $V \in \Re^{n \times n}$ are orthogonal matrices, whose columns are said, respectively, *left/right singular vectors*,

$$\Sigma = \begin{bmatrix} \Sigma_r & 0 \\ 0 & 0 \end{bmatrix}$$

and Σ is a diagonal matrix whose elements (σ_i), univocally determined, are said *singular values* and satisfy:

$$\sigma_i = \sqrt{\lambda_i(A^T A)} \quad (1.31)$$

where $\lambda_i(A^T A)$ is the i -th eigenvalue of the matrix $A^T A$. Note that, since $\lambda_i(A^2) = (\lambda_i(A))^2$, if A is squared and symmetric, singular values and eigenvalues coincide, except possibly the sign.

All singular values are therefore greater than or equal to zero and they are supposed to be ordered in decreasing order. Let us take those strictly bigger than zero in $\Sigma_r = \text{diag}(\sigma_1, \dots, \sigma_r)$, $\sigma_1 > \sigma_2 > \dots > \sigma_r > 0$

1.4.1 Geometric interpretation of the SVD

The meaning of the factorization (1.30) is that for any matrix A it is possible to compute two orthonormal bases, one V for the domain and the other U for the codomain. Moreover, very important, with respect to these two bases the linear transformation defined by A can be represented by a diagonal matrix, Σ . This has a really nice geometric interpretation: the linear transformation defined by the matrix A transforms the hypersphere defined by the left singular vectors $\langle v_1, \dots, v_r \rangle$ in the hyperellipsoid defined by the right singular vectors $\langle \sigma_1 u_1, \dots, \sigma_r u_r \rangle$.

Let us see an example: be A a matrix 2×2 , for simplicity, and its SVD factorization:

$$A = \begin{bmatrix} 0.9501 & 0.6068 \\ 0.2311 & 0.4860 \end{bmatrix}$$

$$U = \begin{bmatrix} -0.9193 & -0.3936 \\ -0.3936 & 0.9193 \end{bmatrix} \quad S = \begin{bmatrix} 1.2212 & 0 \\ 0 & 0.2633 \end{bmatrix} \quad V = \begin{bmatrix} -0.7897 & -0.6135 \\ -0.6135 & 0.7897 \end{bmatrix},$$

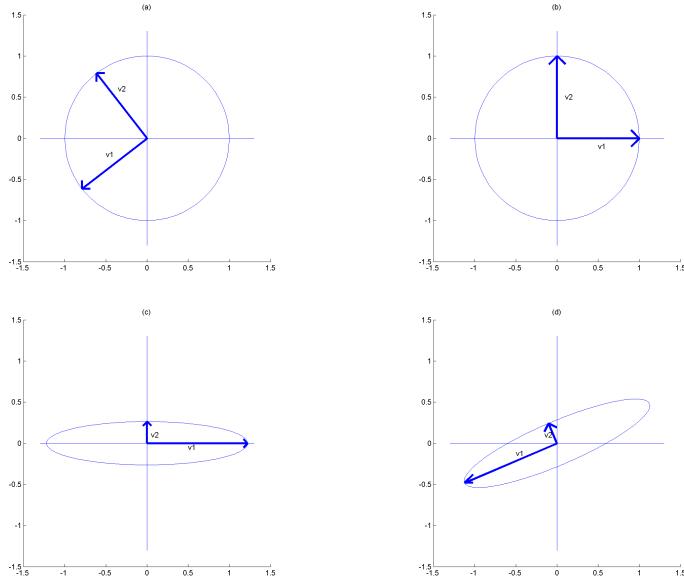
As it can be noted in Figure 1.1, the left singular vectors (a) are rotated by V^T (b), with possible reflections about the origin, then scaled by the singular values (c) and finally rotated by U (d), with possible reflections about the origin.

1.4.2 Relevance of the SVD in numerical computing

The SVD is practically important for numerical computing (despite its high computational cost), other than theoretically for linear algebra:

- the more accurate way to compute the numerical rank of a matrix is to determine the number of singular values greater than “zero”, i.e. a small threshold value chosen according to backward error analysis. Indeed, a backward stable algorithm for the SVD can compute accurately the numerical rank of a matrix. In general, we will operate in the following way: if the i -th singular value is $\hat{\sigma}_i \approx O(\epsilon) \cdot \|A\|_2$, then $\hat{\sigma}_i$ is set to zero, since in a finite precision computer it is indistinguishable from zero. This result is guaranteed by a backward-error analysis made on the algorithm implemented in LAPACK:

$$|\hat{\sigma}_i - \sigma_i| \approx O(\epsilon) \cdot \|A\|_2$$



(a): the unitary “hypersphere” (here a circle since $n = 2$), defined from left singular vectors, i.e. the columns of $V = [v_1 \ v_2]$, (b): $V^T V$, (c): $\Sigma V^T V$, (d): $U \Sigma V^T V \equiv AV$

- the computing of left and right singular vectors is the most accurate way to determine an orthonormal base of the image space of A , i.e. $\langle u_1, \dots, u_r \rangle$ where r is the rank, or of the null-space, i.e. $\langle v_{r+1}, \dots, v_n \rangle$, of a matrix. Moreover, since the space generated by the rows of A is the orthogonal complement of the null-space, it follows that $\langle v_1, \dots, v_r \rangle$ is an orthonormal base for this space. Analogously, $\langle u_1, \dots, u_r \rangle$ is an orthonormal base for the space generated by the columns of A .

Note: for both issues, the QR factorization is a less expensive alternative, but also less accurate.

- the standard method to compute the two-norm of A is $\|A\|_2 = \sigma_1$;
- the *condition number* of a matrix A is $\mathcal{K}_A = \|A\| \|A^{-1}\| = \sigma_1/\sigma_n$. Indeed, $\|A^{-1}\| = \|V\Sigma^{-1}U^T\| = 1/\sigma_n$.

Note that, we can obtain a general matrix with an assigned condition number by assembling it from its SVD: [D.1](#).

- $|det(A)| = \prod_1^r \sigma_i$

1.4.3 Computing the SVD

Now we see that the immediate approach of computing singular values by following (1.31), is unstable. Let us see a simple example:

$$A = \begin{bmatrix} 1 & 1 \\ \mu & 0 \\ 0 & \mu \end{bmatrix}$$

where μ is a real number, such that $\epsilon < \mu < \text{sqrt}(\epsilon)$ and then $fl(1 + \mu^2) = 1$, ϵ is machine precision and $fl(\cdot)$ indicates that the operation has been carried out with finite precision (floating-point precision). We have:

$$fl(A^T A) = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}$$

that would bring to $\hat{\sigma}_1 = \sqrt{2}$ e $\hat{\sigma}_2 = 0$ and the wrong conclusion that the matrix A has rank 1. Instead, with infinite precision:

$$A^T A = \begin{bmatrix} 1 + \mu^2 & 1 \\ 1 & 1 + \mu^2 \end{bmatrix}$$

that brings to $\sigma_1 = \sqrt{2 + \mu^2}$ e $\sigma_2 = |\mu|$ and the right conclusion that the matrix A has rank 2.

Note that this tight relation between singular values and eigenvalues implies that we cannot compute singular values with a direct method. Moreover, computing the SVD can be made in a similar way than that of computing eigenvalues.

There is, anyway, a strict conceptual link between the computation of singular values and of eigenvalues, expressed by (1.31). Therefore, as a consequence of a fundamental theorem in algebra, we know in advance that it is impossible to compute the singular values with a direct method. Instead, it is necessary to proceed by successive approximations. Indeed, let us recall that for eigenvalues are used iterative methods like *the power method* to approximate a single eigenvalue (and corresponding eigenvector), or *subspace iteration* or the *QR method* for a simultaneous approximation of more eigenvalues (and corresponding eigenspaces).

The numerical approximation of eigenvalues/eigenvectors is made by preliminarily transforming the matrix into a **revealing form**, i.e. an equivalent matrix in which the eigenvalues can be directly extracted. Recall that the equivalence can be assured by transforming the matrix by similitude. With real matrices, such a form is the **real Schur form**: for any matrix $A \in \mathcal{R}^{n \times n}$ it exists an orthogonal matrix Q such that:

$$Q^T A Q = S \tag{1.32}$$

where S is in general an upper triangular matrix with possible 2×2 blocks on the diagonal. These correspond to just as many couples of complex-conjugate eigenvalues of A ; therefore, if A has all real eigenvalues, S is upper triangular.

To place near to the Schur form, they use Householder transformations that reduce A in upper Hessenberg form (i.e. all the components below the first sub-diagonal are zeroed):

```
% trasformazione per similitudine di A in forma di Hessenberg superiore,
% mediante trasformazioni ortogonali di Householder;
for k=1:n-2
    x = A(k+1:n,k);
    v_k = sign(x(1))*norm(x)*[1 zeros(1,n-k-1)]' + x;
    v_k = v_k/norm(v_k);      % "v_k" \e il vettore di Householder
    % applico Q_k a sinistra di A :
    A(k+1:n,k:n) = A(k+1:n,k:n) - 2*v_k*v_k'*A(k+1:n,k:n);
    % applico Q_k a destra di A :
    A(1:n,k+1:n) = A(1:n,k+1:n) - 2*A(1:n,k+1:n)*v_k*v_k';
end;
```

Note that, since they are similitude transformations, and therefore must be applied to the left and inversely to the right, it is not possible to obtain an upper triangular matrix by similitude with a direct method, like we made e.g. to obtain the QR factorization by Householder transformations:

But the idea to approach iteratively a revealing form (here the Schur form) using Householder transformations is actually the most efficient algorithm: the QR method for eigenvalues.

Now, when computing the SVD we have more flexibility, since the orthogonal transformations Q and S , applied to the left and to right of A , can be different e quindi $\tilde{A} = QAS$, a differenza di quanto avviene nel caso degli autovalori, in cui la trasformazione di A deve essere per similitudine, ovvero $\tilde{A} = XAX^{-1}$.

Therefore, they tend to produce a bi-diagonal matrix A , with nonzero elements on the main diagonal and the first upper-diagonal. A quel punto, $A^T A$ risulterebbe tri-diagonale.

The standard algorithm to compute the SVD is therefore based upon a *bidiagonalization*, using Householder trasformations, followed by an iteration cycle to diagonalize the matrix, and its most recent formulation, contained in LAPACK, is backward stable.

We have:

$$A = \begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \quad Q_1 A = \begin{bmatrix} x & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix} \quad Q_1 A S_1 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & x & x & x \\ 0 & x & x & x \end{bmatrix}$$

$$Q_2 Q_1 A S_1 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & x \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix} \quad Q_2 Q_1 A S_1 S_2 = \begin{bmatrix} x & x & 0 & 0 \\ 0 & x & x & 0 \\ 0 & 0 & x & x \\ 0 & 0 & x & x \end{bmatrix} \quad \dots$$

Note that $\tilde{A}^T \tilde{A} = S^T A^T Q^T Q A S = S^T A^T A S$ and $\tilde{A}^T \tilde{A}$ has the same eigenvalues of $A^T A$ and, therefore, \tilde{A} has the same singular values of A .

At this point, as is made for eigenvalues/eigenvectors, we use an iterative method to diagonalize A . The QR method is again a good candidate.

1.4.4 Analysis of linear transformations with the SVD

Using the SVD of matrix A , i.e. $A = U \Sigma V^T$, it is possible to do a detailed analysis of the linear transformation represented by A and of its inverse. Let us consider the linear system $Ax = b$, avente quindi A come matrice. Let us see two remarkable situations, where A has one o more singular values much smaller of σ_1 .

In the first case, let us suppose x equal to a linear combination of the last columns of V , i.e. corresponding to the smallest singular values of A : we get a vector $b = U \Sigma V^T x$ of greatly reduced length. In general, by linearity, this means that we can add a quite large perturbation to x and get a small residual, if the perturbation is aligned with the singular vectors corresponding to the smallest singular values of A . In this case the algorithms that adopts the residual as a stopping criterion would be working badly and this is the reason why preconditioners are usually adopted.

In the other case, let us consider the inverse problem of $Ax = b$, e.g. with A square ($n \times n$) and invertible:

$$x = A^{-1}b = V \Sigma^{-1} U^T b \quad (1.33)$$

or rectangular A ($m \times n$, $m > n$), with full column rank:

$$x = V \Sigma^\dagger U^T b \quad . \quad (1.34)$$

Note that, in case of matrix A square and invertible, we have $\Sigma^\dagger = \Sigma^{-1}$. Now, the columns of U are an orthonormal basis that can represent the vector b with a vector of coefficients $d = U^T b$. Its coefficients are then scaled with $c = \Sigma^{-1}d$ or $c = \Sigma^\dagger d$ and, therefore, the components of b aligned with the singular vectors corresponding to the smallest singular values will be relatively much more amplified than the components of b aligned with the biggest singular components. Then, $x = Vc$.

As an example, setting b equal to a linear combination of the columns of U corresponding to the smallest singular values, we obtain a vector $x = V \Sigma^\dagger U^T b$ of length much amplified.

This means that to the right-hand-side b , that typically contains experimental data, it can be summed-up a vector of small magnitude (e.g. the rounding and/or measurement errors) and get a large variation of the solution x .

Let us see this in an **example**: D.5.

It is thus evident the loss of accuracy that can arise when the problem is **ill-conditioned**, i.e. the matrix has a large value of *condition number* $\mathcal{K}_A = \sigma_1/\sigma_n$, see (1.11).

In general, an ill-conditioned problem needs to be *regularized* before it is solved numerically, as we will see in Chapter 4. To this aim it becomes of fundamental importance to know if the components of $d = U^T b$ corresponding to the smallest singular values bring irrelevant information for the problem, or not. In the first case the ill-conditioning can be resolved by cutting them away with the TSVD; otherwise it is necessary to adopt more sophisticated regularization techniques.

1.4.5 Truncated SVD (TSVD)

The standard method to compute the matrix A_μ , of rank $\mu \leq n$, that best approximates $A \in \Re^{m \times n}$ in the 2-norm (i.e. whose distance from A in 2-norm is minimal with respect to all the matrices $B \in \Re^{m \times n}$ of rank less than or equal to μ) is the following (said *dyadic expansion*), see the Eckart-Young-Mirsky theorem ([29] Theorem 5.8 for the 2-norm and Theorem 5.9 for the Frobenius norm; see also [30] theorem 2.3). Since $A = U\Sigma V^T$, placed $\Sigma = \sum_{j=1}^n \Sigma_j$, where:

$$\Sigma_j(k, l) = \begin{cases} \sigma_j & \text{se } k = l = j \\ 0.0 & \text{altrimenti} \end{cases}$$

then, it holds:

$$A = U \sum_{j=1}^n \Sigma_j V^T = \sum_{j=1}^n U \Sigma_j V^T = \sum_{j=1}^n \sigma_j u_j v_j^T$$

, we have:

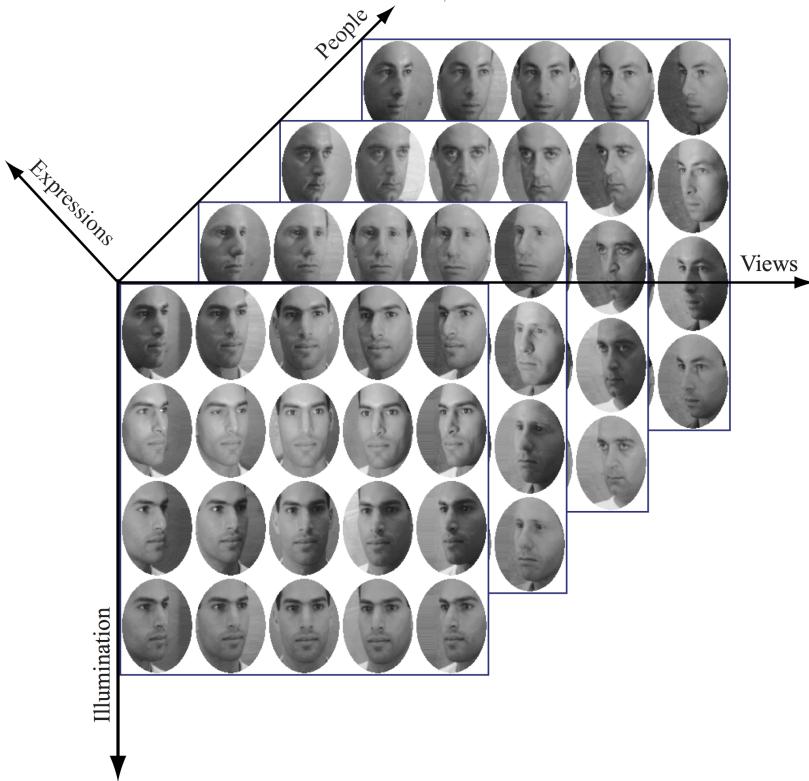
$$A_\mu = \sum_{j=1}^{\mu} \sigma_j u_j v_j^T \quad \|A - A_\mu\|_2 = \sigma_{\mu+1} \quad (1.35)$$

Let us see a practical example of TSVD applied to the matrix corresponding to a gray-scale image: E.1

1.4.6 Applicative example: eigenfaces

Consider the tensor dataset \mathcal{D}

Let us see a face recognition application using the SVD, “eigenfaces”: (notebook) and `animazione_eigenfaces.gif`.



1.5 Nonnegative Matrix Factorizations (NMF)

Problem 1.5.1 (Nonnegative matrix factorization). Given a nonnegative matrix $X \in \mathcal{R}_+^{m \times n}$, a factorization rank r , and a distance measure $D(\dots, \dots)$ between two matrices, compute two nonnegative matrices $W \in \mathcal{R}_+^{m \times r}$ and $H \in \mathcal{R}_+^{r \times n}$, such that $D(X, WH)$ is minimized, i.e. solve:

$$\min_{W \in \mathcal{R}_+^{m \times r}, H \in \mathcal{R}_+^{r \times n}} D(X, WH) \quad (1.36)$$

It builds a nonnegative basis in the columns of W and expresses the columns of M through the coefficients contained in the corresponding columns of H , where $H \geq 0$. It means that the approximation is made in an *additive* way: Lee, D., Seung, H. Learning the parts of objects by non-negative matrix factorization. Nature 401, 788?791 (1999)

Examples (from [Gillis, SIAM 2021]):

- **Urban.mat**: This is a wavelength-by-pixel 162×94249 hyperspectral image of a Walmart in Copperas Cove (Texas); see Figure 5 for an illustration. It is used in many places in the book to compare various NMF models; see Sections 4.1 and 4.5.

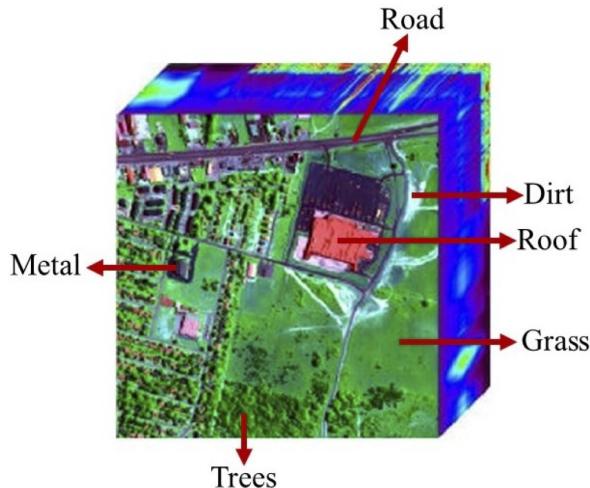


Figure 5: Urban data set.

1) hyperspectral

2) Feature extraction in a set of images (facial set):

See Figure 1.4 and 1.5 and matlab example!

Main difficulties: NP-hard, non uniqueness

A current line of research is to make additional assumptions about the input matrix M , e.g.:

1) separable NMF: *separability* of M means that there are r columns of indices \mathcal{K} that can generate all the columns of M : $M = WH$ where $W = M(:, \mathcal{K})$ and $H \geq 0$. This last constraint makes the effort much bigger than with CMR factorizations. It means that in H there will be all the columns of the identity matrix.

This corresponds to a “pure-pixel assumption” in the hyperspectral example: for each material (i.e. basis element) there is at least one pixel containing only that material.

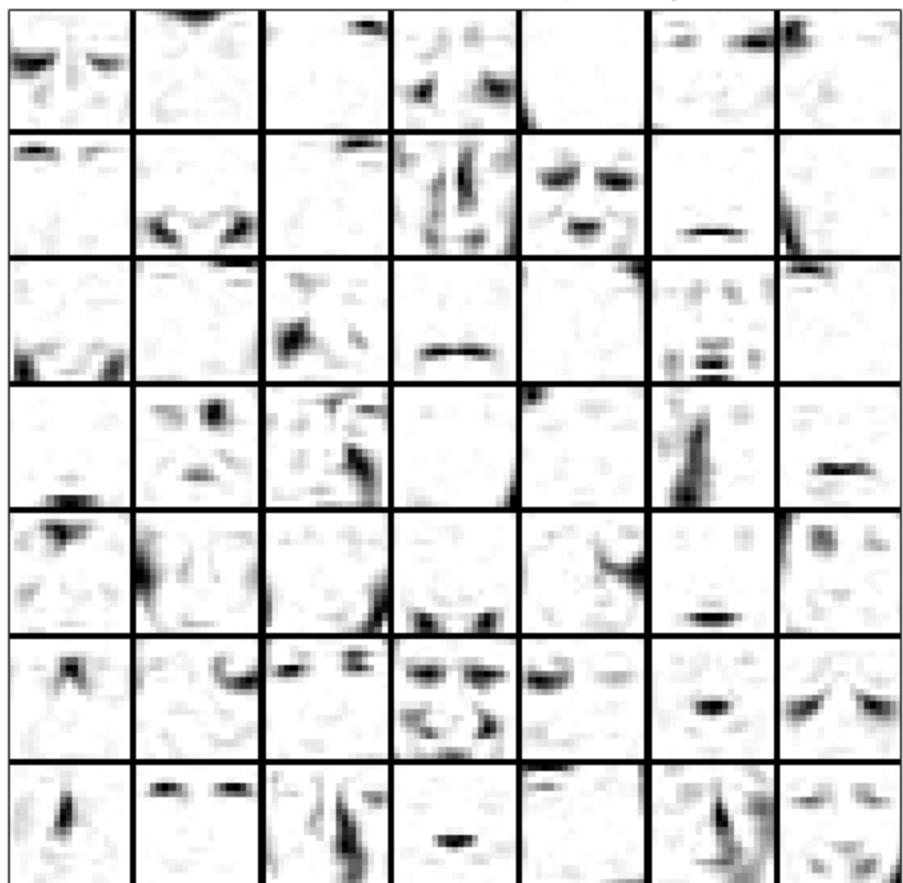
- **CBCL.mat**: This is a 361×2429 pixel-by-image matrix whose columns are gray-scale facial images of size 19×19 ; see Figure 1 for a sample.



Figure 1: Sample images of the CBCL data set.

In feature extraction we cannot expect that there is at least one face corresponding to one feature, so we must use the separability of the transpose, M^T : since facial features are very sparse, we can assume that for each facial feature there is a 'pure' pixel that belongs only to that feature, i.e. an identity-column in the matrix H obtained by factorizing M^T .

Basis elements: columns of W reshaped as images



Capitolo 2

Linear dimensionality reduction of data and models

Data are organized in matrices. Often data are not concise (they may be even redundant), i.e. corresponds to measured variables which bring a common piece of information ... their abundance may be equivalent to a dimensionality higher than necessary. As we will see, data dimensionality is often measured as the rank of a matrix.

A mathematical model of a real systems (physical, socio-economic, etc) may be seen as a set of equations that express in a quantitative way the causal relations owned by the system, or describe as the system responds with an **output** to an applied stimulus at the **input**.

A single real system may be represented by a multitude of mathematical models, depending on which aspects of the system must be taken into account and, therefore, represented in the model, together with the intended analysis to be carried on and the precision that must be attained (the model is always an approximation!). In the model choice, one must consider a balance between the effort necessary to build the model and the real advantages that the model can give. There are two main types of **cost**:

- the first is the cost of model building and of its simulation at the computer (typically, a *computational cost*);
- the second one is related to statistics, i.e. to the cost of acquiring the information (i.e. data, a-priori and measured) needed to validate the model.

Both bring to the fundamental concept of *parsimony*: model complexity (typically measured by the number of parameters contained in the model) must be **the minimum sufficient** to solve the modelling problem with the available data.

Moreover, the input-output relation described in the model can be abstract (**black-box**), mainly data-driven, or given by physical laws (**gray-box**). Therefore, such relations may have very different origins: physical laws (e.g. mechanics, thermodynamics, etc.), socio-economic laws (e.g. from studies about population dynamics, econometrics, finanza, etc.) or abstract laws (e.g. from studies in pattern analysis, statistics, etc.).

Models are constituted, in general, by *functional relations among variables* and by *parameters*. The functional relations define a class of models (the *model structure*) and that a complete set of parameter values identify a univocal member of the class.

Often, during the construction of a model, one introduces some redundancy or an exaggerated level of accuracy, with respect to the quantity and/or quality of data available to validate the model. In this case, it is wise to reduce the complexity of the model, said also *model order*, i.e. to apply techniques of Model Order Reduction (MOR). These are mainly applications of linear algebra, often based upon **low-rank approximations** of the model matrices. More precisely, they are applied to matrix factorizations. A reduced-order model is an approximate model which possesses a lower computational cost and better statistical properties (e.g. less parameters means a lower variance of the estimation error, with the same amount of data).

in the sequel we see some linear models generally used in applications, and corresponding linear dimensionality reduction techniques.

2.1 Principal Component Analysis (PCA)

Let us suppose to have collected a dataset, consisting of m experiments in which n variables $\{q_i\}_{i=1\dots n}$, that we suppose explanatory of the case at hand, organized in a matrix $A \in \mathbb{R}^{m \times n}$ where each row represents an experiment. Therefore, we have just defined the data matrix A from its row vectors A_i (experiments): $A = [A_1; A_2; \dots; A_m]$.

The rationale behind Principal Component Analysis (PCA), see e.g. [15], is that of reducing the dimensionality of the dataset consisting of a high number n of variables $\{q_i\}$ **mutually correlated**, maintaining the biggest part of variance (information) owned by the data. This means also to reduce the number of variables, and it is obtained through a transformation in a new set of variables, said **principal components**, that are mutually uncorrelated, are linear combinations of the original variables and have the property that a small subset of principal components possess a big part of the variability contained in the original set.

Let us see an example: [G](#)

2.1.1 How to compute principal components

Principal components are linear combinations of the original variables. Therefore, each principal component is characterized by a vector α_i of n coefficients. As we will see, they are mutually uncorrelated, i.e. the corresponding vectors of coefficients are mutually orthogonal.

Note the analogy with a similarity transformation of a matrix, to bring it in modal coordinates, i.e. with respect to a basis of eigenvectors. Indeed, the vectors of coefficients that represent the principal components are the eigenvectors of the covariance matrix C of the n variables $\{q_i\}$, i.e. of the matrix whose generic element $C(i, j)$ is the covariance between variables q_i and q_j :

$$\max_{\alpha_1} \text{var} [\alpha_1^T q] = \max_{\alpha_1} \alpha_1^T C \alpha_1 \quad \text{such that } \alpha_1^T \alpha_1 = 1 \quad (2.1)$$

$$\text{Lagrange multipliers: } \max_{\alpha_1} \left(\alpha_1^T C \alpha_1 - \lambda_1 (\alpha_1^T \alpha_1 - 1) \right) \quad (2.2)$$

$$\text{stationary point: } C \alpha_1 - \lambda_1 \alpha_1 = 0 \quad (2.3)$$

and, therefore, (λ_1, α_1) is a couple eigenvalue-eigenvector of matrix C . Moreover, we have:

$$\text{var} [\alpha_1^T q] = \alpha_1^T C \alpha_1 = \lambda_1 \quad (2.4)$$

and, in general:

$$\text{var} [\alpha_i^T q] = \alpha_i^T C \alpha_i = \lambda_i \quad (2.5)$$

hence, to the biggest eigenvalue corresponds the vector of coefficients which gives the biggest variance and corresponds to the first principal component. If, as in this case, α_i has unitary length, we have that the eigenvalues of C are equal to the variance of the corresponding principal components.

Note that *variables scaling* has a fundamental importance on the determination of the principal components: it is really important to be careful at measurements units, especially when heterogeneous variables are involved. Typically, the choice of the measurement unit depends also on the magnitude of the error present in the data.

The matrix C may be known apriori, e.g. from a probabilistic model that describes the n variables and their mutual co-variance, or it may be estimated.

In the second case, one starts from the matrix $A \in \Re^{[m \times n]}$ of data, defined above. Let us subtract to each row A_i the mean value of each variable across the experiments (which is also a row vector) $M = 1/m(A_1 + A_2 + \dots + A_m)$, we obtain the row vectors $X_i = A_i - M$, which are rows of the matrix X of the zero-mean data. At this point, the covariance matrix estimate, becomes:

$$S = \frac{1}{n-1} X^T X \quad (2.6)$$

Let us see an example: [G.1](#)

Important, with the SVD of the matrix A , i.e. $A = U\Sigma V^T$, it is possible to compute the principal components without explicitly computing the covariance matrix S . Indeed, from [\(1.31\)](#) the singular values are the square roots of the eigenvalues of $A^T A$ (variances) and the columns of V are the eigenvectors of $A^T A$, i.e. the singular components.

In multi-variate regression, Principal Component Analysis helps to avoid the *multi-collinearity* problem, i.e. when two or more linear combinations of the original variables are almost linear dependent. Note that such problem is not immediately evident, since its presence is not necessarily pointed out by a simple mutual correlation between couples of model variables. Actually, it is put on evidence by small singular values and a substantial gap between them and the other, bigger, singular values.

2.2 Static linear regression models in the ideal (noiseless) case

Given a vector of *regression variables* q and a vector of *regression parameters* (or *coefficients*) x , both of n components, let us consider the following linear combination of variables q and coefficients x :

$$q \cdot x = q_1 \cdot x_1 + q_2 \cdot x_2 + \dots + q_n \cdot x_n \quad (2.7)$$

which constitutes the structure of the standard, linear static model. The variables q are assumed known without error (deterministic). Let us suppose, for the moment, that the *observations* of the *target variable* (or *model output*), be ideally equal to a deterministic vector $\bar{b} \in R^m$, whose components express an exact linear relation with the regression variables q , i.e. for the i -th experiments it holds $\bar{b}_i = q^{(i)} \cdot x$.

The matrix formulation of the model, becomes: the observation vector b is tied to the regression parameters $x \in R^n$ through the linear model:

$$A \cdot x = b \quad (2.8)$$

where

$$A = \begin{bmatrix} q_1^{(1)} & q_2^{(1)} & q_3^{(1)} & \dots & q_n^{(1)} \\ q_1^{(2)} & q_2^{(2)} & q_3^{(2)} & \dots & q_n^{(2)} \\ \dots & & & \dots & \\ q_1^{(m)} & q_2^{(m)} & q_3^{(m)} & \dots & q_n^{(m)} \end{bmatrix}$$

is the *regression matrix*, whose values are supposed known.

2.2.1 Linear dimensionality reduction of static linear regression models

Let us consider the SVD of the matrix A , $A = U\Sigma V^T$, where $V = [\alpha_1, \dots, \alpha_n]$ is the matrix whose columns are the principal components of A . The linear model $Ax = b$, becomes $U\Sigma V^T x = b$ and then $U\Sigma x^p = b$, where $x^p = V^T x$ are the model parameters represented in the (orthonormal) basis of the principal components (while x where expressed in the original basis). With a matrix factorization, we have changed the system representation into orthogonal coordinates.

Now, let us consider the matrix V_k , formed by the first k columns of V and, therefore, by the first k principal components. The reduced model is obtained by considering the first k principal components and by **projecting** the system on the subspace generated by the first k images of these components, i.e. the first k columns of U :

$$U_k^T U_k \Sigma_k V_k^T x = U_k^T b \quad (2.9)$$

and, therefore:

$$\Sigma_k x_k^p = U_k^T b \quad (2.10)$$

It is a model with mutually uncoupled variables, since the system matrix is now diagonal.

2.3 Discrete-time Linear Time-Invariant (DLTI) dynamical systems in the ideal (noiseless) case

2.3.1 The DLTI model as a convolution sum

A **Discrete-time, Linear, Time-Invariant (DLTI) dynamical system** is a mathematical operator T which transform a discrete time sequence (i.e. a discrete function) $u(:)$, said *input sequence*, into a discrete time sequence $y(:)$, said *output sequence* (the symbol ":" indicates that one consider all the time indexes together):

$$y(:) = T[u(:)]$$

and has these properties:

- **linearity:** if $y_1(:)$ and $y_2(:)$ are the model responses to inputs $u_1(:)$ and $u_2(:)$, then by linearity we mean that (said also *superposition principle*)

$$T[\alpha_1 \cdot u_1(:) + \alpha_2 \cdot u_2(:)] = \alpha_1 \cdot T[u_1(:)] + \alpha_2 \cdot T[u_2(:)] = \alpha_1 \cdot y_1(:) + \alpha_2 \cdot y_2(:)$$

- **time-invariance**, i.e. invariance to time-translations: if the model respond to the input sequence $u(:)$ with the output sequence $y(:)$, it must respond to the translated input sequence $u(: - j)$ with the translated output sequence $y(: - j)$.

Now, let us explicitly construct this operator, by satisfying these two properties. Let us express the generic element $u(k)$ of the input sequence, in relation with the overall sequence $u(:)$:

$$u(k) = \sum_{j=-\infty}^{\infty} u(j) \cdot \delta(k-j) \quad (2.11)$$

where $\delta(:)$ is the *Kronecker's delta* (or *discrete impulse*, the discrete-time counterpart of the Dirac impulse):

$$\delta(k) = \begin{cases} 1 & k = 0 \\ 0 & \text{altrimenti} \end{cases} \quad (2.12)$$

and, therefore:

$$u(:) = \sum_{j=-\infty}^{\infty} u(j) \cdot \delta(: - j) \quad (2.13)$$

We have:

$$y(:) = T \left[\sum_{j=-\infty}^{\infty} u(j) \cdot \delta(: - j) \right]$$

that, applying linearity and time-invariance, becomes::

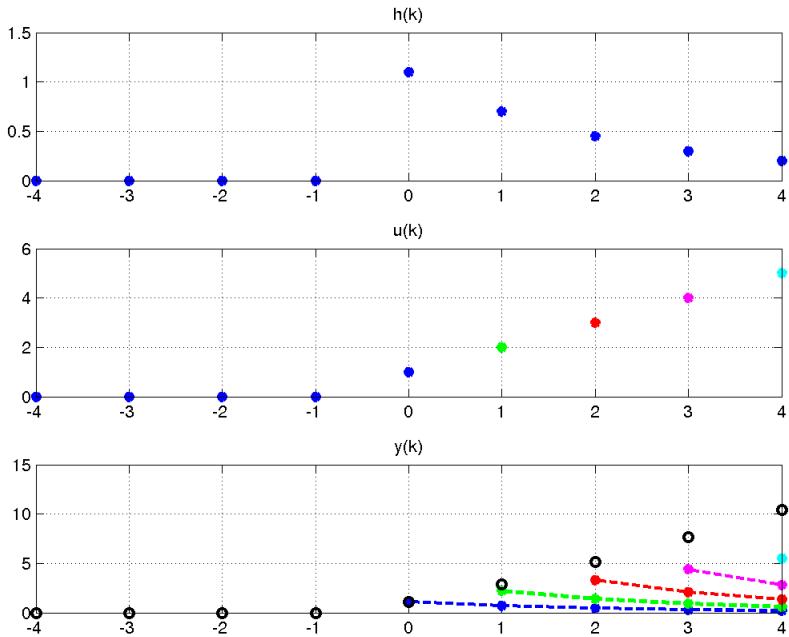
$$y(:) = \sum_{j=-\infty}^{\infty} u(j) \cdot T[\delta(: - j)] = \sum_{j=-\infty}^{\infty} u(j) \cdot h_j(:) = \sum_{j=-\infty}^{\infty} u(j) \cdot h(: - j)$$

and, for the generic instant k :

$$\begin{aligned} y(k) &= \sum_{j=-\infty}^{\infty} u(j) \cdot h(k - j) \\ &= \dots + \\ &\quad u(k - 2) \cdot h(2) + \\ &\quad u(k - 1) \cdot h(1) + \\ &\quad u(k) \cdot h(0) + \\ &\quad u(k + 1) \cdot h(-1) + \\ &\quad u(k + 2) \cdot h(-2) + \\ &\quad \dots \end{aligned} \quad (2.14)$$

The expression (2.14) is commonly named (*discrete convolution* or convolution sum). A numerical example is reported in Figure 2.1. Note that a DLTI system is **univocally identified** by the sequence $h(:)$, said **discrete impulse response**. Note that only if $h(k) = 0$ for $k < 0$ the output at time k , $y(k)$, is not influenced by future inputs $u(j)$, $j > k$; the systems that has this property are called **causal** and are the mostly seen (e.g. a physical system is causal).

The convolution sum (2.14) is not the unique mathematical expression for a DLTI system; actually, there are two more. The first one expresses the functional



Example of convolution sum (2.14)

relation between input and output through a *difference equation* with constant coefficients, i.e. with a recursive algorithm:

$$\sum_{i=0}^{n_a} a_i \cdot y(k-i) = \sum_{j=0}^{n_b} b_j \cdot u(k-j) \quad (2.15)$$

where the output at time k , $y(k)$, is computed by auto-regression (AR) on past values of the output itself, and a moving-average (MA) on past values of the input. Such representation of a DLTI system is called *ARMA*. The constant n_a is said **model order**. Notare che la risposta all'impulso discreto di un modello ARMA puó avere durata *finita*, ed allora si parla di sistema *FIR* e si ha per $n_a = 0$, oppure *infinita*, ed allora si parla di sistema *IIR* e deve essere $n_a > 0$.

Note that in (2.15), the values of the coefficients, i.e. the model parameters, are not univocally identified. Indeed, multiplying both sides by a constant C we get an equivalent input-output expression, but with model parameters $\{C a_i\}$ e $\{C b_i\}$. To get a one-to-one correspondence between model parameters and input-output behaviour usually one parameter is fixed, and precisely set $a_0 = 1$. From now on

we will do this assumption:

$$\sum_{i=0}^{n_a} a_i \cdot y(k-i) = \sum_{j=0}^{n_b} b_j \cdot u(k-j) \quad , \quad a_0 = 1 \quad (2.16)$$

Note that the one-to-one correspondence between model parameters and input-output behaviour is not important in the *direct* problem, i.e. the computation of the output given the input and parameter values, but is fundamental in the *inverse* problem of parameter estimation, that usually is solved by computing parameter estimates from measurement data of input and output sequences: these estimates would be not univocally identified, because this estimation problem would have infinite solutions. This one-to-one correspondence is called *identifiability of the system*.

Let us see an example: C.6

2.3.2 ARMA models and the theory of difference equations

Let us start with a first-order AR model (2.15):

$$y(k) = -a_1 y(k-1) + b_0 u(k)$$

where $b_j = 0$, $j > 0$ ed $a_0 = 1$, $a_i = 0$, $i > 1$. In this case, the output $y(k)$ is the sum of a *geometric progression* (the auto-regression) and of an *arithmetic progression* (due to the input). Let us see the model response to the canonical inputs: Kronecker's delta (2.12) and unit step (2.17):

$$g(k) = \begin{cases} 1 & k \geq 0 \\ 0 & \text{altrimenti} \end{cases} \quad (2.17)$$

In the first case, the arithmetic progression vanishes from $k = 1$ and, therefore, the geometric progression prevails and, in particular: if $|a_1| > 1$ the output sequence diverges, if $|a_1| = 1$ remains at constant amplitude and if $|a_1| < 1$ converges to zero. Moreover, if $a_1 < 0$ the behaviour is monotone, otherwise oscillatory. In the second case (unit step) we have a constant arithmetic progression: if $|a_1| > 1$ the output sequence diverges, if $|a_1| = 1$ remains at constant amplitude or diverges and if $|a_1| < 1$ converges to $y_\infty = b_0 / (1 + a_1)$.

Example 1 [25]: the dynamics of an amount of money, increased by a simple interest rate r :

$$\begin{cases} y(k) = y(k-1) + ru(k) \\ u(k) = y(0) \\ y(0) = Y_0 \end{cases}$$

whose explicit solution is $y(k) = (1 + rk)Y_0$, similar to an arithmetic progression. If a compound interest is used:

$$\begin{cases} y(k) = (1 + r)y(k-1) \\ y(0) = Y_0 \end{cases}$$

whose explicit solution is $y(k) = (1+r)^k Y_0$, similar to a geometric progression.

Example 2: the numerical integration scheme for an ordinary differential equation, with the finite difference method:

$$\begin{cases} y'_c(t) = f(t, y_c(t)) \\ y_c(t_0) = Y_0 \end{cases}$$

if $t_0 = 0$ and one uses e.g. the explicit Euler method, comes out the following recursive expression:

$$y(k) = y(k-1) + hf(kh, y(k-1))$$

Example 3: the discretization of the heat equation, to solve an initial boundary value problem:

$$\begin{cases} \frac{\partial y_c}{\partial t} = \frac{\partial^2 y_c}{\partial x^2} & x \in (x_a, x_b), \quad t > 0 \\ y_c(x, 0) = f(x) & x \in [x_a, x_b] \\ y_c(x_a, t) = y_c(x_b, t) = 0 & t > 0 \end{cases}$$

the explicit Euler method brings to an expression of the kind $Y(k) = BY(k-1)$, while the implicit Euler method brings to $AY(k) = Y(k-1)$. A similar example that we will use is the numerical integration of the linear equations of elastodynamics, see Appendix B.

Let us suppose now that in (2.15) be $b_j = 0, \forall j$ or, equivalently, $u(k) = 0, \forall k$. The second condition indicates that the output $y(k)$ will be the *free response* of the system, which is the response to a null input. The expression (2.15), per $a_0 = 1$ e $a_{n_a} \neq 0$, is a **difference equation, homogeneous linear, of order n_a , with constant coefficients**:

$$y(k) + a_1 y(k-1) + a_2 y(k-2) + \cdots + a_{n_a} y(k-n_a) = 0 \quad (2.18)$$

Theorem (Superposition principle): If the sequences $y' = \{y'(k)\}$ and $y'' = \{y''(k)\}$ are solutions of (2.18), then are also $y' + y''$ and cy' , $\forall c$. Hence, the set of solutions of (2.18) is a vector space. (the proof proceeds by simple substitution).

Definition: the *characteristic polynomial* of (2.18), indicated with $\mathcal{P}(\lambda)$, is the following polynomial of degree n_a and variable λ :

$$\mathcal{P}(\lambda) = \lambda^{n_a} + a_1 \lambda^{n_a-1} + a_2 \lambda^{n_a-2} + \cdots + a_n$$

The equation

$$\mathcal{P}(\lambda) = 0 \quad , \quad \lambda \in \mathbb{C} \quad (2.19)$$

is said *characteristic equation* associated to the homogeneous equation (2.18).

Theorem: The equation (2.18) admits infinite solutions. Among them, it is possible to determine n solutions linear independent. Moreover, the set of solutions of (2.18) is a

subspace of dimension n , spanned by these fundamental solutions. If λ is a solution of multiplicity 1 of the characteristic equation (2.19), it has a corresponding fundamental solution of the difference equation (2.18):

$$v(k) = \lambda^k$$

If λ is a solution of multiplicity m of the characteristic equation (2.19), it has m corresponding fundamental solutions of the difference equation (2.18), linearly independent:

$$v_0(k) = \lambda^k , \quad v_1(k) = k\lambda^k , \quad v_2(k) = k^2\lambda^k , \dots , \quad v_{m-1}(k) = k^{m-1}\lambda^k$$

All the solutions of the difference equation (2.19) are linear combinations of the fundamental solutions $v_j(k)$ associated to the λ s.

Stability analysis: from the previous Theorem, said λ the roots of the characteristic equation (2.19):

- if all of them satisfy the condition $|\lambda| < 1$, the DLTI system represented by the difference equation (2.18) is **asymptotically stable** and the solutions of the homogeneous difference equation (2.18) converge to zero;
- if all of them satisfy the condition $|\lambda| \leq 1$ and that of unitary modulus are of multiplicity 1, the DLTI system is **(simply) stable** and the solutions of the homogeneous difference equation (2.18) remain at constant amplitude;
- if at least one root satisfy $|\lambda| > 1$, the DLTI system is **unstable** and the solutions of the homogeneous difference equation (2.18) may diverge to infinity (depends on the initial conditions);

2.3.3 State-space models

The input-output representation of a DLTI system is said *external*, since it does not describe the internal (unmeasured) dynamics of the system. A common approach to describe the internal dynamics of a DLTI system is to define a vector of state-variables, said *state vector* and to represent the DLTI system in the state space.

Be $x(t)$ a vector of time-dependent functions in \Re^n , $u(t) \in \Re^{n_u}$ and $y(t) \in \Re^{n_y}$ said, respectively, the state, inputs and outputs vectors of out dynamic system. The *representation in the state-space* of an LTI system in continuous-time is the following:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{2.20}$$

$$y(t) = C_c x(t) + D_c u(t) \tag{2.21}$$

where A_c , B_c , C_c e D_c are matrices of adequate dimensions. The number of components in the state-vector is said **model order**. Let us see an example: C.7. To use

this model in a numerical algorithm, it is common to discretize this model. Since (2.20) it is a system of ordinary differential equations, the system (2.20)-(2.21) can be discretized with one of the techniques well known in numerical analysis, see e.g. [24], obtaining the following DLTI system:

$$x(k+1) = Ax(k) + Bu(k) \quad (2.22)$$

$$y(k) = Cx(k) + Du(k) \quad (2.23)$$

where $x(k) \in \mathbb{R}^n$, $u(k) \in \mathbb{R}^{n_u}$ ed $y(k) \in \mathbb{R}^{n_y}$ are, respectively, the state, inputs and outputs vectors at the discrete time k . Let us see an example: C.8.

The **Markov coefficients**, G_k , of (2.26)-(2.23), are defined as:

$$G_0 = D \quad \text{e} \quad G_k = CA^{k-1}B \quad , \quad k = 1, 2, \dots \quad (2.24)$$

and their sequence, for $k = 0, 1, 2, \dots$, correspond to the discrete impulse response of the system, in the sense that if we apply a Kronecker delta (2.12) to the i -th input, we obtain at the output:

$$h^{(i)}(0) = D[:, i] \quad \text{e} \quad h^{(i)}(k) = CA^{k-1}B[:, i] \quad , \quad k = 1, 2, \dots \quad (2.25)$$

that is the i -th column of the (matrix) Markov coefficients (2.25).

The main properties of these systems are briefly treated in Appendix A. For a general treatment, see e.g. [19].

In Appendix B is briefly described the construction of state-space models in some interesting fields involving differential equations: e.g. elasto-mechanics and heat trasfer. Such models are of general use even outside the specific physical applications; for example, the heat equation is used in traffic models, the convection-diffusion is used in finance, etc.

In this lectures, we will often used a simple mechanical model: three masses connected by springs and dampers C.9. Note also an obnject-oriented implementation of the model: C.10

To summarize, the operator T represents the mathematical model of the input-output behaviour of a DLTI dynamical system, and can be represented in a *non-parametric* form as a convolution sum (2.14), characterized by the discrete impulse response $h(k)$, or in *parametric* form, as an ARMA difference equation (2.15), characterized by the parameter vectors $\{a_i\}$ and $\{b_i\}$, or as a state-space model (2.23), characterized by the parameter matrices A, B, C and D .

2.3.4 Explicit conversions from different representations of DLTI systems

Starting from an ARMA model, it is simple to build an equivalent state-space model. Let us consider the generic ARMA model (2.15) and define a state vector with past values of the input u and of the output y :

$$x(k) = \begin{bmatrix} y(k-1) \\ \vdots \\ y(k-n_a) \\ u(k-1) \\ \vdots \\ u(k-n_b) \end{bmatrix}$$

from which is derived a state-space system (2.26)-(2.23)

with matrices:

$$A = \begin{bmatrix} -a_1 & -a_2 & \dots & -a_{n_a} & b_1 & b_2 & \dots & b_{n_b} \\ 1 & 0 & \dots & & & & & \\ 0 & 1 & 0 & \dots & & & & \\ \ddots & \ddots & \ddots & & & & & \\ \dots & 0 & 0 & 0 & \dots & & & \\ & \dots & 0 & 1 & 0 & \dots & & \\ & & \dots & 0 & 1 & 0 & 0 & \\ & & & \dots & 0 & 1 & 0 & \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

$$C = [-a_1 \quad -a_2 \quad \dots \quad -a_{n_a} \quad b_1 \quad b_2 \quad \dots \quad b_{n_b}] \quad , \quad D = b_0$$

This system has order $n_a + n_b$ and it is easy to verify the equivalence with the ARMA model (2.15). Actually, it is possible to build an equivalent state-space model of order n_a , in this way:

$$A = \begin{bmatrix} 0 & 1 & 0 & \dots & \dots \\ 0 & 0 & 1 & 0 & \dots \\ & & \ddots & & \\ & & \dots & 0 & 1 \\ -a_{n_a} & -a_{n_a-1} & \dots & \dots & -a_1 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

$$C = [(b_{n_a} - b_0 a_{n_a}) \quad \dots \quad (b_1 - b_0 a_1)] \quad , \quad D = b_0$$

We leave as an exercise to verify the equivalence, by comparing the discrete impulse responses with symbolic calculation.

Note that the state-space model *I/O equivalent* to an ARMA model is not unique. Actually, there are infinite state-space systems of order n_a algebraically equivalents, see Appendix A.

The converse, i.e. to find an ARMA mode equivalent to a given state-space model, is more complicated and, in general, requires the Z-transform concept.

In practice, there are applications where each of the three representations is preferred against the others.

2.3.5 Model identifiability

The discrete impulse response is in one-to-one relation with a couple of input-output sequences obtained from a DLTI system, see (2.14).

The ARMA representation (2.16) is in one-to-one relation with a couple of input-output sequences obtained from a DLTI system.

The state-space representation has always infinite parametrizations algebraically equivalents; therefore, there is no one-to-one relation with a couple of input-output sequences obtained from a DLTI system.

2.4 Linear dimensionality reduction of state-space DLTI models

Let us see the problem of approximating a state-space DLTI system with a state-space DLTI system of reduced order. A good reference for this subject is, for example, the book of Antoulas [1], where we see some examples motivating the subject: electronic circuits, storm surge forecast, high tower buildings, etc.

Let us recall the general form (2.23) of a state-space representation for a DLTI system:

$$x(k+1) = Ax(k) + Bu(k) \quad (2.26)$$

$$y(k) = Cx(k) + Du(k) \quad (2.27)$$

If the original model order is medium-small, there are optimal techniques:

- balanced truncation
- optimal approximation in Hankel norm

If the dimension of the system is large, these techniques becomes computationally prohibitive.

Two fundamental questions arise:

1. the reduced model must inherit the same fundamental properties owned by the original system, e.g. stability;
2. get an estimate of the reduced model approximation error.

2.4.1 Balanced truncation

A **balanced representation** of a DLTI system is an important concept: it implies that each state $x \in \mathcal{R}^n$ requires the same reachability and observability effort. Mathematically, the balancing corresponds to the simultaneous diagonalization of the reachability and observability **gramians**:

$$G_R = RR^T \quad \text{reachability gramian} \quad (2.28)$$

$$G_O = O^T O \quad \text{observability gramian} \quad (2.29)$$

Their meaning is the energy necessary to bring the system from zero to the state x , $E_R = x^T G_R^{-1} x$, and the energy obtained by observing the outputs from the state x with null inputs, $E_O = x^T G_O x$.

A reasonable approach is to eliminate those states, and possibly an entire subspace of the state-space, that require a high level of energy to be reached and those that produce little energy when observed, i.e. those states that are difficult to reach and to observe. This concept has a strong analogy with the Principal Component Analysis, already seen for static models, and has been introduced by Moore, in "Principal component analysis in linear systems: controllability, observability and model reduction", IEEE Trans. AC-26, 17-32, 1981.

A balanced representation is the best setting to solve this problem: it allows a **truncation** that fulfills both requirements:

$$G_R = G_O = \Sigma = \text{diag}(\sigma_1, \dots, \sigma_n) \quad (2.30)$$

where the σ_i are said *Hankel singular values* of the system S . The balanced truncation corresponds to retain only the first k state variables of the balanced basis. It has been demonstrated that this keeps the original stability of the system and is also available a bound on the truncation error. For example, the limit superior to the H_∞ norm (i.e. the maximum modulus of the frequency response) of the system approximation error, is given by $2(\sigma_{k+1} + \dots + \sigma_n)$, i.e. a result quite similar to the Eckart-Young theorem.

From the point of view of linear algebra, the balancing corresponds to the simultaneous diagonalization of two symmetric, positive definite matrices. Let us now compute a balanced representation of a DLTI system.

In general, a basis change T operates on the gramians in the following way:

$$\tilde{G}_R = TG_R T^T, \quad \tilde{G}_O = T^{-T} G_O T^{-1} \Rightarrow \tilde{G}_R \tilde{G}_O = T(G_R G_O) T^{-1} \quad (2.31)$$

i.e. none of them is transformed by similarity, but their product does. Let us consider the following Cholesky and eigenvalue decompositions:

$$G_R = U^T U, \quad U G_O U^T = K \Sigma^2 K^T \quad (2.32)$$

where K is an orthogonal matrix. Note that the diagonal matrix is precisely Σ^2 of (2.30), because $U G_O U^T$ is a matrix similar to $G_R G_O$, since $U^T (U G_O U^T) U^{-T} =$

$G_R G_O$, and we have just seen that a basis change does not modify the eigenvalues of $G_R G_O$ that, when both are diagonalized (2.30), are evidently Σ^2 . To obtain a balanced representation: given a reachable, observable and stable system S having gramians G_R and G_O , the transformation T that balance the system is:

$$T = \Sigma^{1/2} K^T U^{-T} \quad , \quad T^{-1} = U^T K \Sigma^{-1/2} \quad (2.33)$$

(it holds: $T G_R T^T = \Sigma$ and $T^{-T} G_O T^{-1} = \Sigma$)

Let us consider the numerical computing of a balanced representation. A fundamental problem is that for model orders n not small, G_R e G_O have numerical rank $<< n$ and the above formula turns out to be ill-conditioned.

Let us consider the following matrix factorizations:

$$G_R = U^T U \quad , \quad \text{Cholesky, } U: \text{upper triangular}$$

$$G_O = L L^T \quad , \quad \text{Cholesky, } L: \text{lower triangular}$$

$$U G_O U^T = K \Sigma^2 K^T$$

$$U L = W \Sigma V^T \quad , \quad \text{SVD, the diagonal components of } \Sigma \text{ are the } \textit{Hankel singular values}$$

where the last one come from $U G_O U^T = U L L^T U^T = (U L) (U L)^T$ and from (1.31).

Algorithm 1. The transformation and its inverse are:

$$T = \Sigma^{1/2} K^T U^{-T} \quad , \quad T^{-1} = U^T K \Sigma^{-1/2}$$

Algorithm 2. (square root) The transformation and its inverse are:

$$T = \Sigma^{-1/2} V^T L^T \quad , \quad T^{-1} = U^T W \Sigma^{-1/2}$$

(it is easy to verify that one is the inverse of the other and that the resulting system is balanced)

Algorithm 3. (square root and diagonal scaling) The transformation and its inverse are:

$$T = V^T L^T \quad , \quad T^{-1} = U^T W$$

that produce a balanced system except for a diagonal scaling, but often with a better conditioning.

2.4.2 Proper Orthogonal Decomposition (POD)

This technique, widely used, may be seen as the application of the SVD to the approximation of dynamical systems in general, even nonlinear.

It approximates a generic state-trajectory $\mathcal{T} = \{x(i) \in \Re^n\}$ (obtained given the initial state and the applied inputs) with a trajectory $\mathcal{T}^k = \{x_{POD}(i) \in \Re^k\}$ where $k < n$.

With this aim it is applied a projection method, usually Galerkin or Petrov-Galerkin. Since this projection is obtained from state-trajectories and not from system matrices, this fact creates a different type of approximation with respect to the new trajectories that are different from that used to build the reduced model. Results in this direction say, for example, that a projection in the eigenspace of dominant eigenvalues of $G_R G_O$ produces, for linear systems, an approximation error which is globally bounded.

Let us compute a POD. Suppose to have measured/simulated a trajectory of our system and to have stored it columnwise in the matrix $X \in \Re^{n \times N}$; let us call x_i the i -th column of X .

Let us find now an orthonormal basis U of vectors $u_j \in \Re^n$, $j = 1, 2, \dots, N$ such that:

$$UC = X \quad (2.34)$$

where $U^T U = I$, $C \in \Re^{N \times N}$.

The matrix decomposition (2.34) is said *Proper Orthogonal Decomposition (POD)* and the vectors u_j are the *empirical eigenvectors* (or *principal directions*) of the data matrix X .

The aim of this decomposition is that the reconstruction \hat{X} from $k < n$ empirical eigenvectors be optimal with respect to the norm $\|X - \hat{X}\|_2$. The solution is: given X , let us consider its SVD:

$$U\Sigma V^T = X \quad (2.35)$$

then the columns of U are the empirical eigenvectors and the coefficient matrix is $C = \Sigma V^T$.

Let us consider now the projection. In general, the projection of a DLTI system in a reduced dimension state-space is made with a basis change followed by a truncation. Therefore, chosen the dimension k of the reduced state vector and chosen $Y, W \in \Re^{n \times k}$, let us define the basis change matrix and its inverse:

$$T = \begin{bmatrix} W^T \\ T_2^T \end{bmatrix}, \quad T^{-1} = [Y \quad T_1] \quad (2.36)$$

and partition consequently the state vector:

$$\bar{x} = \begin{bmatrix} \hat{x} \\ \tilde{x} \end{bmatrix}.$$

We have:

$$\begin{cases} \bar{x}(k+1) \\ y(k) \end{cases} = \begin{cases} TAT^{-1}\bar{x}(k) + TBu(k) \\ CT^{-1}\bar{x}(k) \end{cases} \quad (2.37)$$

where the first k equations constitute the system (where we have substituted the expressions (2.36) to T and T^{-1})

$$\begin{cases} \hat{x}(k+1) &= W^T A(Y\hat{x}(k) + T_1\tilde{x}(k)) + W^T B u(k) \\ y(k) &= C(Y\hat{x}(k) + T_1\tilde{x}(k)) \end{cases} \quad (2.38)$$

that is therefore approximated by truncation, putting to zero the variables \tilde{x} .

The possible choices for the matrices Y and W fall into two principal classes:

- $Y = W$, i.e. a Galerkin projection; the matrix T must be orthogonal (and we speak of orthogonal projections);
- $Y \neq W$, i.e. a Petrov-Galerkin projection, or *oblique* projection, and matrices Y and W usually satisfy some constraints.

The matrix $YW^T \in \mathbb{R}^{n \times n}$ is called *projector*.

The Galerkin projection consider the first k singular vectors (columns) of U :

$$Y = W = U_k \in \mathbb{R}^{n \times k} \quad (2.39)$$

and $Y\hat{x} = YY^T x$ is the projection of the state vector x in $\text{span}\{U_k\}$.

2.4.3 Modal truncation

It is greatly used in structural engineering and different from balanced truncation. it is based on the eigenvalue decomposition of the matrix A of the system and not of the gramians product $G_R G_O$.

In structural mechanics, the couples eigenvalue-eigenvector of A have a clear physical meaning: they describe the vibration modes of a structures and their frequencies. Let us consider equation (B.8) homogeneous with $G = 0$

$$M\ddot{d}(t) + Kd(t) = 0 \quad (2.40)$$

and look for a time-harmonic solution: $\bar{d} = d_\omega \cos(\omega t)$. We have:

$$\omega^2 M d_\omega + K d_\omega = 0 \quad (2.41)$$

i.e. a generalized eigenvalue problem. Since matrices M and K are symmetric and at least positive semi-definite, the eigenvalues are all real and non negative, $\lambda_i \geq 0$. The model reduction is made by keeping the eigenspaces corresponding to the lowest vibration frequencies.

Let us see an example: F

Capitolo 3

Linear models estimation and learning

3.1 Static linear regression models

3.1.1 Standard linear model

The *observations* vector $b \in R^m$, referred to the *target variable*, in practice is the sum of a *systematic* component \bar{b} , which expresses the linear and deterministic relation with the regression variables q , i.e. $\bar{b}_i = q^{(i)} \cdot x$, and an error component ϵ , random vector with zero mean and variance $\sigma^2 \cdot I$, i.e. made by mutually uncorrelated variables of variance σ^2 :

$$b = \bar{b} + \epsilon \quad (3.1)$$

Suppose we want to do *statistical inference* [2] on the values of the parameters x :

$$A \cdot x = b \quad (3.2)$$

where

$$A = \begin{bmatrix} q_1^{(1)} & q_2^{(1)} & q_3^{(1)} & \dots & q_n^{(1)} \\ q_1^{(2)} & q_2^{(2)} & q_3^{(2)} & \dots & q_n^{(2)} \\ \dots & & & \dots & \\ q_1^{(m)} & q_2^{(m)} & q_3^{(m)} & \dots & q_n^{(m)} \end{bmatrix}$$

is the *regression matrix*, whose values are known. Note that:

- the presence of the error ϵ implicate that the model (3.2) is never satisfied exactly by the measurement data, whatever be the values of x . For this reason, it is wise to choose $m > n$ and solve the system (3.2) using a global criterion, like least-squares (cfr. par. 3.3).

- this model applies in particular in controlled experiments, since we suppose that the regression variables are known without error.

3.1.2 Generalized linear model (Gauss-Markov)

In the generalized linear model, said Gauss-Markov model, ϵ is a vector of random variables with zero mean and variance $\sigma^2 \cdot W$, i.e. with correlated error components.

3.1.3 Error in-variable model

Often regression variables are measured with non trivial error. To consider this error in the model, said \bar{A} the matrix of true values, we will have a matrix $A \in R^{m \times n}$ affected by an (assumed additive) error term E :

$$A = \bar{A} + E \quad (3.3)$$

where $E \in R^{m \times n}$ is a random matrix whose rows have zero mean and variance $\sigma_E^2 \cdot I$.

The linear model has the same structure as before:

$$A \cdot x = b \quad (3.4)$$

but now A and b have both a random component. For this problem we will seek a solution with Total Least Squares, sec. 3.4.

3.2 Discrete-time Linear Time-Invariant (DLTI) dynamical systems

3.2.0.1 Stochastic ARMA model

When there is noise in data, this can be accounted for in an ARMA model in this way:

- ARMA (e di conseguenza AR, ponendo $n_c = 0$ e $c_0 = 1$):

$$\begin{aligned} a_0 \cdot y(k - n_a) + a_1 \cdot y(k - n_a + 1) + \cdots + a_{n_a} \cdot y(k) \\ c_0 \cdot e(k - n_c) + c_1 \cdot e(k - n_c + 1) + \cdots + c_{n_c} \cdot e(k) \end{aligned} = k = 0, 1, 2, \dots \quad (3.5)$$

- **ARMAX** (e di conseguenza **ARX**, ponendo $n_c = 0$ e $c_0 = 1$):

$$\begin{aligned} a_0 \cdot y(k - n_a) + a_1 \cdot y(k - n_a + 1) + \cdots + a_{n_a} \cdot y(k) &= \\ b_0 \cdot u(k - n_b) + b_1 \cdot u(k - n_b + 1) + \cdots + b_{n_b} \cdot u(k) &+ \\ c_0 \cdot e(k - n_c) + c_1 \cdot e(k - n_c + 1) + \cdots + c_{n_c} \cdot e(k) & \quad k = 0, 1, 2, \dots \end{aligned} \quad (3.6)$$

ovvero un *modello ARMA con ingressi*

where $e(k)$ is a sequence of uncorrelated random variables, identically distributed, of zero mean and variance σ^2 (white noise), and $n_a \geq n_b$, $n_a \geq n_c$, to satisfy the causality condition.

The constant n_a is again the **model order**.

For example, a time-serie is often considered as the output of an ARMA or AR model.

3.2.0.2 Stochastic state-space models

Often, in applications, it is necessary to consider also a **model error** and a **measurement error**:

$$x(k+1) = A \cdot x(k) + B \cdot u(k) + v(k) \quad (3.7)$$

$$y(k) = C \cdot x(k) + D \cdot u(k) + w(k) \quad (3.8)$$

, where $\{v(k)\}$ e $\{w(k)\}$ are the vectors that describe the model error and the measurement error, supposed with zero mean and covariance matrices:

$$E \left\{ \begin{bmatrix} v(k) \\ w(k) \end{bmatrix} \cdot [v(k)' w(k)'] \right\} = \begin{bmatrix} Q & S \\ S' & R \end{bmatrix} \quad (3.9)$$

where Q and R are positive definite matrices.

3.3 Numerical methods for ordinary least-squares

The (Discrete) **Linear Least-Squares problem** looks for the vector $x \in \Re^n$ that gives the minimum of $\|Ax - b\|_2^2$, where $A \in \Re^{m \times n}$ and $b \in \Re^m$.

In general, there exist three solution methods for this problem, when A has full rank:

1. solving the *normal equations*: $A^T Ax = A^T b$;
2. using the QR factorization of A ;
3. using the SVD of A .

3.3.1 Normal equations method

We look for a stationary point for the gradient of

$$f(x) = \|Ax - b\|_2^2 = (Ax - b)^T(Ax - b) . \quad (3.10)$$

We have:

$$\begin{aligned} & \lim_{e \rightarrow 0} \frac{(A(x+e)-b)^T(A(x+e)-b)-(Ax-b)^T(Ax-b)}{\|e\|_2} = \\ &= (\text{noting that } (Ax - b)^T A e = e^T A^T (Ax - b) \text{ is a scalar}) = \\ &= \lim_{e \rightarrow 0} \frac{2e^T(A^T Ax - A^T b) + 2e^T A^T A e}{\|e\|_2} = 0 \end{aligned}$$

it must hold

$$A^T Ax - A^T b = 0 \quad (3.11)$$

, said *normal equations system*.

Moreover, since the Hessian of $f(x)$ is the matrix $A^T A$, which is a *positive definite* matrix (its eigenvalues are $\sigma_i^2 > 0$, where σ_i are the singular values of A) the local minimum found is also the global minimum. Since $A^T A$ is also symmetric, it can be used the Cholesky factorization algorithm.

3.3.2 Method with QR factorization

Since A is supposed to have full (column) rank, the existence of its QR factorization is guaranteed. Starting from the normal equations, we have:

$$\begin{aligned} x &= (A^T A)^{-1} A^T b = \\ &= (R^T Q^T Q R)^{-1} A^T b = \\ &= (R^T R)^{-1} A^T b = \\ &= R^{-1} R^{-T} R^T Q^T b = R^{-1} Q^T b \end{aligned}$$

Note that we have used the orthogonality of Q : it is therefore necessary that the algorithm which computes the QR factorization produces a Q matrix which is really orthogonal.

3.3.3 A better look at the Pseudo-inverse of a matrix

If A is invertible, its inverse is simply derived through the SVD:

$$A^{-1} = V \Sigma^{-1} U^T . \quad (3.12)$$

In general, if A is non-invertible, there is a problem with Σ , since U and V are always orthogonal (and therefore, invertibles).

Note that A and Σ have the same dimensions. Let us consider the case $A \in R^{m \times n}$ with $m > n$ and A has full column rank. In this case Σ will be like:

$$\Sigma = \begin{bmatrix} \Sigma_1 \\ 0 \end{bmatrix}$$

where Σ_1 is a diagonal matrix with all positive entries. Following the geometrical interpretation of the SVD, the action of A is a rotation with V^T , scaling with Σ_1 and addition of $m - n$ null components to the vector, and a final rotation with U .

Let us now take a vector $b \in R^m$ and apply to it the pseudo-inverse, using the SVD of A , the rotation U^T can be applied, the scaling of the first n components can be done with Σ^{-1} , while the addition of $m - n$ null components can be inverted by truncating the same amount of components of $U^T b$. The truncation makes this process irreversible.

More formally:

$$\begin{aligned} A^\dagger &= (A^T A)^{-1} A^T \\ &= (V \Sigma^T U^T U \Sigma V^T)^{-1} V \Sigma^T U^T \\ &= (V \Sigma^T \Sigma V^T)^{-1} V \Sigma^T U^T \\ &= (V \Sigma_1^2 V^T)^{-1} V \Sigma^T U^T \\ &= V \Sigma_1^{-2} V^T V \Sigma^T U^T \\ &= V \Sigma_1^{-2} \Sigma^T U^T \\ &= V \begin{bmatrix} \Sigma_1^{-1} & 0 \end{bmatrix} U^T \\ &\equiv V \Sigma^\dagger U^T \end{aligned} \tag{3.13}$$

3.3.4 Method with the SVD

Using the (reduced) SVD of A , $A = U_r \Sigma_1 V^T$ with U and V orthogonal matrices and Σ_1 diagonal, whose existence is assured for any A , we have (choosing \tilde{U} in such a way that $[U_r \tilde{U}]$ be square and orthogonal) :

$$\begin{aligned} \|Ax - b\|_2^2 &= \|U_r \Sigma_1 V^T x - b\|_2^2 = \\ &= \left\| \begin{bmatrix} U_r^T \\ \tilde{U}^T \end{bmatrix} (U_r \Sigma_1 V^T x - b) \right\|_2^2 = \\ &= \left\| \begin{bmatrix} \Sigma_1 V^T x - U_r^T b \\ -\tilde{U}^T b \end{bmatrix} \right\|_2^2 = \|\Sigma_1 V^T x - U_r^T b\|_2^2 + \|\tilde{U}^T b\|_2^2 \end{aligned}$$

and, since A has full rank, Σ_1 is invertible and we have the least-squares solution $x = V \Sigma_1^{-1} U_r^T b$. The matrix $A^\dagger = V \Sigma_1^{-1} U_r^T$ is called the *pseudo-inverse (Moore-Penrose)* of A .

Note that we have used the well-known fact that the pre- or post-multiplication of a vector by an orthogonal matrix Q does not modify the euclidean norm of the vector. Indeed, considering the generic eigenvalue λ of Q , from the expression $Qv = \lambda v$ and $v^T Q^T = \lambda v^T$, we have, multiplying both sides of the last two expressions: $v^T Q^T Q v = \lambda^2 v^T v$. Now, if Q is orthogonal, the last expression becomes: $v^T v = \lambda^2 v^T v$ and, since $v^T v \neq 0$, $|\lambda| = 1$, i.e. Q transform the unit hypersphere in another unit hypersphere. Thus, it does not change the euclidean length of vectors.

3.3.5 Choice of the algorithm

We can say that:

- the normal equations method is the cheapest, more than QR and much more than SVD;
- if A is well conditioned, normal equations give the same result of the other two;
- if A is ill-conditioned but far from singular, the best choice is QR;
- if A is severely ill-conditioned, or singular, the choice is a compromise between cost and reliability: QR "rank-revealing" is cheaper, while the SVD is more reliable.

Let us see an experimental comparison between these algorithms: [D.6](#)

3.4 Numerical methods for Total Least Squares

The ordinary least squares problem can be formulated also in this way, to find $b' \in \mathbb{R}^m$ such that:

$$\hat{b}' = \operatorname{argmin}_{b'} \|b' - b\|_2 \quad \text{with the constraint:} \quad b' \in \operatorname{Range}(A) \quad (3.14)$$

and we know its solution \hat{b}' is the orthogonal projection of b in $\operatorname{Range}(A)$. The constraint means that the solution must belong exactly to the space generated by the columns of A , supposed exact, i.e. with no error on the regression variables.

If we consider, instead, the model with error in-variables [\(3.4\)](#), the parameter estimation problem, said *Total Least Squares* (TLS), becomes:

$$(\hat{A}', \hat{b}') = \operatorname{argmin}_{A', b'} \| [A \quad b] - [A' \quad b'] \|_F \quad \text{with the constraint:} \quad b' \in \operatorname{Range}(A') \quad (3.15)$$

The computation of the TLS solution is made with the SVD. Let us formulate the linear system in the following way:

$$[A \quad b][x^T \quad -1]^T = 0 \quad (3.16)$$

and

$$U\Sigma V^T = [A \quad b] \quad . \quad (3.17)$$

Now, if $\sigma_{n+1} > 0$, i.e. A has full rank and $b \notin Range(A)$, then the only solution to (3.20) would be zero, but it is incompatible with (3.20). Using the Eckart-Young-Mirsky theorem ([30] theorem 2.3), the minimum perturbation to $[A \quad b]$, such that the perturbed matrix $[\hat{A} \quad \hat{b}]$ has rank n , i.e. not full rank, is obtained through the SVD:

$$[\hat{A} \quad \hat{b}] = U\hat{\Sigma}V^T \quad \text{con} \quad \hat{\Sigma} = diag(\sigma_1, \dots, \sigma_n, 0). \quad (3.18)$$

Then, the TLS solution is given by:

$$[\hat{x}^T \quad -1] = \frac{-1}{v_{n+1,n+1}} v_{n+1} \quad (3.19)$$

where we have considered a proper scaling to get a solution compatible with (3.20), in particular to have -1 as last component.

Note that if it were $\sigma_{n+1} = 0$, then $v_{n+1} \in Ker([A \quad b])$ the solution would be the same, but it would be exact.

By comparison, ordinary least-squares find the solution in case $b \notin Range(A)$ by projecting the system (3.20) in the space generated by the columns of A :

$$A^T[A \quad b][x^T \quad -1]^T = [A^TA \quad A^Tb][x^T \quad -1]^T = 0 \quad (3.20)$$

where $[A^TA \quad A^Tb]$ has not full-rank anymore, since it has been projected on $Range(A)$, that has dimension n .

3.5 Numerical methods for least-squares problems with linear constraints

Let us see how to modify the numerical methods for least-squares problems, when the model is represented, other than by linear equations, by linear constraints that may involve its parameters.

If we take equation (3.10) and set $H = A^TA$ and $s = -2A^Tb$, subtract the term $\frac{1}{2}b^Tb$, which is independent from x , and impose equality and inequality constraints $v_{low} \leq Mx \leq v_{up}$, we obtain a problem said *quadratic programming*, i.e. a problem with a quadratic cost function and linear constraints:

$$\min_x f(x) = \min_x \left(\frac{1}{2}x^T H x + s^T x \right), \quad \text{with constraints} \quad Cx \geq u \quad (3.21)$$

In the class of problems considered in this course, the unknowns of least-squares problems are (some) model parameters, and we can notice four types of constraints that apply:

- equality constraints: e.g. a pendulum, where the suspended body satisfies the minimum of the total energy, with the constraint that the distance from a fixed point be equal to its length;
- inequality constraint: e.g. the shortest path in presence of an obstacle, a contact problem ... etc.;
- validity intervals (i.e. bounds) for parameters;
- non-negativity of parameters: e.g. masses, temperatures (in Kelvin), etc.

The constraints define a *feasible region* for the solution. If the minimum of the unconstrained problem falls into the feasible region, it is sufficient to compute its solution and to ignore the constraints. This would simplify substantially the computations.

Actually, there are many situations where this fact is not known a-priori. For inequality constraints, sometimes it is known, e.g. when using temperatures expressed in Kelvin degrees: here to set the constraint $T > 0$ is unnecessary, unless we are working on extreme experiments. Equality constraints, instead, are never neglected, since it is very unlikely that the minimum of the unconstrained problem falls into their intersection (they produce a substantial dimensionality reduction!).

The first two types of constraints are embedded in the definition of the model.

In general, the bounds and non-negativity constraints (a particular case of bounds) represent extreme conditions that should not be reached. This is because in these models their reaching is an alert of some anomalous condition, or an incorrect model formulation, etc. There is, however, an important class of problems where the non-negativity constraints are used to sparsify the solution and, in these models, to reach the extreme value (here zero) is actually encouraged.

In the next section we see direct algorithms for equality constraints and iterative algorithms for the general case and the particular cases of bounds on single variables (model parameters) and non-negativity.

3.5.1 Direct algorithms for equality constrained problems: the null-space method

Let us suppose that in problem (3.21) there are only n_e equality constraints $Cx = u$ and that the matrix $C \in R^{n_e \times n}$ has full row-rank (for sure it will have less rows than columns).

Let us define the Lagrangian function for this problem:

$$\mathcal{L}(x, \lambda) = \frac{1}{2} x^T H x + s^T x - \lambda^T (Cx - u) \quad (3.22)$$

The well-known Karush-Kuhn-Tucker (KKT) conditions, of the first order, that we will show in their general form in the next paragraph, require in this case that the optimal solution x^* must satisfy:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = Hx^* + s - C^T \lambda = 0 \quad (3.23)$$

$$Cx^* - u = 0 \quad (3.24)$$

e.g., in matrix form:

$$\begin{bmatrix} H & -C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} x^* \\ \lambda^* \end{bmatrix} = \begin{bmatrix} -s \\ u \end{bmatrix} \quad (3.25)$$

For future convenience, we rewrite the system by expressing the solution $x^* = x + p$, i.e. as the sum of an estimate x , not necessarily feasible, plus the step p that brings to the optimum:

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix} \begin{bmatrix} -p \\ \lambda^* \end{bmatrix} = \begin{bmatrix} g \\ h \end{bmatrix} \quad (3.26)$$

where $h = Cx - u$, $g = Hx + s$ e $p = x^* - x$. The following Lemma states the additional conditions (said second order conditions since they involve the Hessian) for the existence and unicity of the solution.

Lemma 3.5.1. *Let us suppose that the matrix C be full row-rank and that the reduced Hessian $Z^T H Z$ be positive definite, where Z is the matrix whose columns are a base for the null-space of C . Then, the matrix*

$$\begin{bmatrix} H & C^T \\ C & 0 \end{bmatrix}$$

is invertible and exists, therefore, a unique solution (x^, λ^*) to the problem (3.26).*

Dimostrazione. See [20] Sec. 16.1 □

Now, let us see a direct algorithm for the solution of (3.26). Let us express the vector p as the sum:

$$p = Yp_Y + Zp_Z \quad (3.27)$$

where Y an arbitrary matrix such that $[Y Z]$ be invertible. By substituting this expression of p in the second equation of the system (3.26) and remembering that by definition of Z it holds that $CZ = 0$, we have:

$$CYp_Y = -h \quad (3.28)$$

and the matrix $CY \in R^{n_e \times n_e}$ is invertible, since C has rank n_e and $C [Y \ Z] = [CY \ 0]$ has consequently rank n_e .

Let us substitute (3.27) now in the first equation of the system (3.26):

$$-HYp_y - HZp_z + C^T\lambda^* = g \quad (3.29)$$

pre-multiply by Z^T :

$$-Z^THYp_y - Z^THZp_z + Z^TC^T\lambda^* = (\text{since } C \ Z = 0) = -Z^THYp_y - Z^THZp_z = Z^Tg \quad (3.30)$$

and we can obtain p_z :

$$(Z^THZ)p_z = -Z^THYp_y - Z^Tg \quad (3.31)$$

using a Cholesky factorization, since Z^THZ is supposed positive definite.

To compute the Lagrange multipliers λ^* , let us pre-multiply the first equation of (3.26) by Y^T :

$$Y^TC^T\lambda^* = (CY)^T\lambda^* = Y^T(Hp + g) \quad (3.32)$$

In conclusion, the presence of linear equality constraints can be solved by simple direct methods.

3.5.2 Iterative algorithms for inequality constrained problems: active-set methods

Let us suppose that in the problem (3.21) there are n_e equality constraints \mathcal{E} and n_d inequality constraints \mathcal{D} , represented together in matrix form as $Cx \geq u$, with $C \in R^{(n_e+n_d) \times n}$.

The Lagrangian function has again the form of (3.22), as follows,

$$\mathcal{L}(x, \lambda) = \frac{1}{2}x^THx + s^Tx - \lambda^T(Cx - u) \quad (3.33)$$

but here there are both equality and inequality constraints.

Let us recall the well known Karush-Kuhn-Tucker (KKT) conditions, of the first order [20], that in this particular problem (3.21) reduces as follows:

Theorem 3.5.2. *Let us suppose that x^* be a solution of the problem (3.21), then there exists a vector of Lagrange multipliers λ^* , having a number of components equal to the total number of constraints, and the following conditions hold in (x^*, λ^*) :*

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = Hx^* + s - C^T\lambda = 0 \quad (3.34)$$

$$c_i^T x^* = u \quad \forall i \in \mathcal{E} \quad (3.35)$$

$$c_i^T x^* \geq u \quad \forall i \in \mathcal{D} \quad (3.36)$$

$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{D} \quad (3.37)$$

$$\lambda_i^* (c_i^T x^* - u_i) = 0 \quad \forall i \in \mathcal{E} \cup \mathcal{D} \quad (3.38)$$

where c_i is the i -th column of C and (3.38) are called complementarity conditions.

We define *active-set* $\mathcal{A}(x^*)$ the set of constraints that the solution x^* satisfy with the equal sign. In general, in $\mathcal{A}(x^*)$ there will be all the equality constraints plus possible inequality constraints that the solution x^* satisfy with the equal sign.

Since at the solution x^* , by complementarity conditions (??), the Lagrange multipliers corresponding to inactive constraints are zero, we can omit the corresponding term of the Lagrangian and the optimality conditions become, therefore:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = Hx^* + s - \sum_{i \in \mathcal{A}(x^*)} \lambda_i^T c_i = 0 \quad (3.39)$$

$$c_i^T x^* = u \quad \forall i \in \mathcal{A}(x^*) \quad (3.40)$$

$$c_i^T x^* > u \quad \forall i \in \mathcal{D} \setminus \mathcal{A}(x^*) \quad (3.41)$$

$$\lambda_i^* \geq 0 \quad \forall i \in \mathcal{D} \cap \mathcal{A}(x^*) \quad (3.42)$$

where c_i is the i -th column of C . One can demonstrate that, when the conditions (3.39) are satisfied and H is positive semi-definite, the solution x^* is a global optimum. One can derive also sufficient conditions of the second order: $Z^T H Z$ must be positive definite, where Z is the matrix whose columns are a basis for the null-space of the Jacobian matrix relative to active constraints (here a row selection from C).

If $\mathcal{A}(x^*)$ were a-priori known, we would arrive simply to the case of equality constraints, but in general it is not known. In *active-set* algorithms [20] [4] the correct active-set is found iteratively, by solving a sequence of problems with only equality constraints, each one corresponding to the current estimate of the active-set. At each step the decrease of the cost function is guaranteed and it can be demonstrated that the method terminates in a finite number of steps (but not guarantee that they are only a few).

In detail, at each iteration all the equality constraints and the selected subset of the inequality constraints form the working set \mathcal{W}_k , which is the current estimate of $\mathcal{A}(x^*)$.

Given the current estimate x_k of the solution, that we suppose feasible, and the working set \mathcal{W}_k , first of all we verify that x_k be the solution of the minimum problem in the subspace defined by the working set. If this is not the case, a step p_k is found, that solve the minimum problem with only equality constraints, corresponding to the working set, and momentarily neglected the other constraints. Such problem is solved e.g. with the null-space method shown before.

Let us call p_k the solution of this subproblem. If $x_k + p_k$ is feasible with respect to all the constraints, we set $x_{k+1} = x_k + p_k$, otherwise $x_{k+1} = x_k + \alpha_k p_k$ with $\alpha_k \in [0, 1]$ the biggest possible that satisfies all the constraints. At such value, at least one more constraint is reached with equal sign and, therefore, the method continues to the next iterate by adding this constraint to the working set. When

$p_k = 0$, one must verify the sign of the Lagrange multipliers: if they are all positive the algorithm finishes, otherwise one removes from the working set the constraint corresponding to the negative Lagrange multiplier of biggest magnitude and go to the next iterate.

This algorithm needs an initial feasible point.

3.5.2.1 Active-set methods for least-squares with bounds on variables

When the constraints are simply upper and lower limits on variables, the algorithm must choose between fixed and free variables. At each iteration, the variables chosen as fixed are simply removed from the problem. Here it can be used a more efficient algorithm, the *gradient projection method*. Each iteration of the algorithm is made by two steps.

In the first step it goes along the steepest descent direction $-g$; when it bumps into a new constraint, the direction is modified in such a way that it remains feasible. In this way, the algorithm stops when along this path it reaches a local minimum x^c , said *Cauchy point*.

In the second step, the algorithm looks for the minimum in all the feasible region around the Cauchy point, i.e. it solves a minimum problem where all the active variables (that take an extremal value in their feasible interval) remain constant. This may be, in principle, as complicated as the original problem, but in practice it is sufficient a step toward the minimum, until a new constraint is reached.

Now H may be also not (semi-)definite positive, since this method can be applied also to non-convex problems.

3.5.2.2 Active-set methods for Non-Negative Least Squares (NNLS)

The problem is:

$$\min_{\mathbf{x} \geq 0} \|A\mathbf{x} - \mathbf{b}\|, \quad (3.43)$$

where $A = [\mathbf{a}_1 \dots \mathbf{a}_n]$ is a matrix $m \times n$ and \mathbf{b} is a vector of length m .

Let us see the Lawson-Hanson algorithm [17]:

be P_k and Z_k , respectively, the index sets of *passive* and *active* constraints at step k , with cardinality n_k and $n - n_k$. The algorithm starts with a working set including all possible constraints, e.g. all variables, hence $n_0 = 0$, and an initial estimate $\hat{\mathbf{x}} = 0$ of the solution, that is feasible.

At each iteration:

1. compute $w = A^T(b - A\hat{\mathbf{x}})$, the steepest descent direction (i.e. opposite to the gradient).

2. chooses the index i corresponding to the component of maximal magnitude of w and inserts i in P_k , that is thus removed from the working set and adds the corresponding column of A in the unconstrained problem that is solved at each iteration;
3. solve the unconstrained problem $y_P = \text{argmin} ||A_k y_k - b|| ; y_Z = 0$;
4. verify if y_P has negative components and, in case, set them to zero and inserts the in Z_k .

Lemma (23.17) in [17] ensures that the variable corresponding to the new index i chosen by the Lawson-Hanson algorithm to be inserted in the passive set will be positive in the solution of the unconstrained problem.

Note that at the end the variables that will be in the active set will be null and, therefore, the resulting solution vector will be sparse. This property becomes very important in the solution of underdetermined systems, and in non-negative matrix factorizations.

Capitolo 4

Ill-posedness and rank-estimation, Ill-conditioning and regularization

Let us consider a generic equation

$$Kx = y \quad (4.1)$$

where K is an application between the topological spaces X and Y , and y is a known data. At the beginning of '900 J. Hadamard defined the *well-posedness* of a problem, in the following way:

"The problem (4.1) is *well-posed* if all these conditions are satisfied:

1. $\forall y \in Y$ a solution $x \in X$ exists;
2. the solution x is unique;
3. the solution x depends continuously from y , i.e. the map $y \rightarrow x$ is continuous.

A problem that does not satisfy one of these conditions is said **ill-posed**."

If a problem is well-posed, it does not imply that it can be solved easily on a computer. Indeed, the continuity of the map $y \rightarrow x$ allows anyway high variations of the result from small variations of the data. The *conditioning* of the problem is an indicator of this phenomenon; it measures the sensitivity of the results to the variation of the data. A problem with a high sensitivity is said **ill-conditioned**.

If a problem is ill-posed or ill-conditioned, it is better to *regularize it* before running a numerical algorithm to solve it. The **regularization** is the approximation of the given problem with a family of approximated, better-conditioned problems, depending on a parameter, said **regularization parameter**).

A class of problems that often brings to ill-conditioned or even ill-posed problems, is that of inverse problems. The definition of *inverse problem* is tied to that of the corresponding *direct problem*. The direct problem is the easiest to solve.

Examples:

- $Ax = b$: matrix-vector product and solution of a linear system;

Exercise: solve these overdetermined systems $Ax = b$, in a least squares sense, with A column vector a_1 with random components and $b = a_1$, then with $A = [a_1 a_2]$ in case a_2 be orthogonal to a_1 and in case a_2 be almost linear dependent from a_1 and comment the solutions found. Hint: the presence of a column vector a_2 , if it is almost linear dependent, disturbs heavily the estimate of the coefficient corresponding to a_1 ...

- computing an integral or a derivative:

$$\begin{aligned} f_n^\delta(x) &= f(x) + \delta \sin \frac{nx}{\delta} \\ (f_n^\delta)'(x) &= f'(x) + n \cos \frac{nx}{\delta} \\ \|f - f_n^\delta\|_\infty &= \delta \\ \|f' - (f_n^\delta)'\|_\infty &= n \end{aligned}$$

Considering f as an exact data and f_n^δ as a perturbed data, we see how an arbitrarily small perturbation can bring to a big error in the computation of the derivative.

In general, given an equation (e.g. a differential equation), we can formulate different problems on it. By convention, the easiest problem is the *direct problem* and the others are the *inverse problems*. If the equation represents a mathematical model, the direct problem is usually the simulation of the model, while parameter estimation, state estimation, input estimation (deconvolution), from output or input/output measurements, are inverse problems. Note that data related problems are usually inverse problems.

4.1 Analysis of singular values of a matrix A

The first quantity that arises from the analysis of singular values of a matrix A is the condition number. It is equal to the ratio between the maximum and the minimum singular values. If it is high, it means that there is an almost linear dependence between the columns of A .

Actually, the ensemble of singular values tells much more. For example, if the ill-conditioning is due to an incorrect formulation of the model, often there is an evident *gap* between the lowest singular values and other, and the model can be improved by eliminating the redundant singular components associated to the smallest singular values.

If the ill-conditioning is intrinsic to the model (as it is e.g. for tomography), singular values decade regularly, without a gap and evident redundancy, but this can happen with quite different velocities, and one usually refers to problems:

- *mildly ill-posed* when the singular values σ_i decay as i^{-k} , for some $k > 0$;
- *severely ill-posed* when the singular values decay faster than i^{-k} , for any k .

4.2 Truncated SVD

4.2.1 Solution of ill-conditioned or ill-posed least-squares problems

In case A has not full rank ($\sigma_{min} = 0$) or "nearly full" ($\sigma_{min} \approx 0$), and therefore its numerical rank is $r < n$, it exist an $(n-r)$ -dimensional X_{n-r} of least-squares solutions.

Due to rounding error propagation, to distinguish the singular values strictly positive, i.e. > 0 , from that exactly zero is a difficult problem, because the rounding error propagation adds small perturbations to the final results. We can say that the rank of A (that is equal to the number of positive singular values) may change discontinuously with arbitrary small perturbations of the components of A , and therefore it is an ill-posed problem, in general. Moreover, it holds:

Theorem 4.2.1. [11] "Let $\sigma_{min} > 0$ the smallest singular value of A . We have:

1. if $x^* = \operatorname{argmin}_x \|Ax - b\|_2$ then $\|x^*\|_2 \geq |u_n^T b| / \sigma_{min}$, where u_n is the last column of U in $A = U\Sigma V^T$;
2. changing b in $b + \delta b$ then x modifies in $x + \delta x$, with $\|\delta x\|_2 \approx \|\delta b\|_2 / \sigma_{min}$."

and therefore, in presence of very small singular values the least squares problem becomes ill-conditioned and its solution grows in magnitude.

Starting from the case $\sigma_{min}(A) = 0$, i.e. $\operatorname{rank}(A) = r < n$, it is demonstrated that:

" Be $A = [U_1 U_2] \begin{bmatrix} \Sigma_1 & 0 \\ 0 & 0 \end{bmatrix} [V_1 V_2]^T = U_1 \Sigma_1 V_1^T$, where $\Sigma_1 \in \Re^{r \times r}$ is non-singular and U_1, V_1 have r columns mutually orthogonal. Let $\sigma = \sigma_{min}(\Sigma_1)$, we have that:

1. all the solutions of $\min_x \|Ax - b\|_2$ can be written as $x^* = V_1 \Sigma_1^{-1} U_1^T b + V_2 z$ with z an arbitrary vector of \Re^{n-r} ;

2. with $z = 0$ we have the minimum-norm solution, where $x^* = V_1 \Sigma_1^{-1} U_1^T b$ e
 $\|x^*\|_2 \geq \|u_n^T b\|_2 / \sigma$;
 3. changing b in $b + \delta b$, the minimum norm solution may change at most of
 $\|\delta b\|_2 / \sigma$.
- "

In general, we will operate in the following way: if the i -th singular value is $\hat{\sigma}_i \approx O(\epsilon) \cdot \|A\|_2$, then $\hat{\sigma}_i$ is set to zero, since in a finite precision computer it is indistinguishable from zero. This result is guaranteed by a backward-error analysis made on the algorithm implemented in LAPACK:

$$|\hat{\sigma}_i - \sigma_i| \approx O(\epsilon) \cdot \|A\|_2$$

Note that in applications often there is an additive error on A and b values, quantifiable e.g. in a $\|\delta A\|_2$ bigger than that determined by the backward error analysis. In this case, the algorithm would be more accurate than the data used for problem definition, and it is a good idea to choose a value of tol according to data accuracy. in this way, we get also a better conditioned problem.

4.3 Tikhonov regularization

Said $\delta > 0$ the *regularization parameter*, we can regularize the problem by modifying the cost function by introducing a term that weights the 2-norm of the solution:

$$\hat{x} = \operatorname{argmin}_x \left(\|b - Ax\|_2^2 + \delta \|x\|_2^2 \right) \quad (4.2)$$

This approach to regularization is due to Tichonov.

Applying the same approach that led to the normal equations, we get now:

$$x = (A^T A + \delta I)^{-1} A^T b \quad (4.3)$$

Tikhonov regularization assures that the regularized problem matrix has its singular values bounded from below by $\sqrt{\delta}$. Indeed, let us reformulate problem (4.2) into:

$$\hat{x} = \operatorname{argmin}_x \left\| \begin{bmatrix} A \\ \sqrt{\delta} I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2 \quad (4.4)$$

Now, taken a generic unitary-length vector v , $\|v\|_2 = 1$, we have:

$$\left\| \begin{bmatrix} A \\ \sqrt{\delta} I \end{bmatrix} v \right\|_2^2 = \|Av\|_2^2 + \left\| \sqrt{\delta} Iv \right\|_2^2 = \|Av\|_2^2 + \delta \|v\|_2^2 = \|Av\|_2^2 + \delta \geq \delta$$

and, remembering the geometric interpretation of the SVD, we have the thesis.

Note that this is a technique of rank completion instead of *truncation*, as TSVD does.

Using the dyadic decomposition, let us compare the singular values of the pseudo-inverse with that of the inverse of the Tichonov-regularized problem. Given $A \in \Re^{m \times n}$ of full rank n , the pseudo-inverse becomes:

$$A^\dagger = V \begin{bmatrix} \Sigma_1^{-1} & 0 \end{bmatrix} U^T = \sum_{j=1}^n \sigma_j^{-1} v_j u_j^T \quad (4.5)$$

and for the Tichonov regularization we have:

$$\begin{aligned} (A^T A + \delta I)^{-1} A^T &= (\text{dato che } A^T A = V \Sigma^T U^T U \Sigma V^T = V \Sigma_1^2 V^T) \\ &= V (\Sigma_1^2 + \delta I)^{-1} V^T A^T \\ &= V (\Sigma_1^2 + \delta I)^{-1} V^T V \Sigma U^T \\ &= (\text{essendo } V \text{ ortogonale}) \\ &= V (\Sigma^2 + \delta I)^{-1} \Sigma U^T \\ &= \sum_{j=1}^n \frac{\sigma_j}{\sigma_j^2 + \delta} v_j u_j^T \end{aligned} \quad (4.6)$$

Note that Tichonov regularization works even when A has rank $r < n$.

Let us analyze the TSVD as a regularization method and make some comparisons with Tichonov: [E.2](#)

4.4 Choice of the regularization parameter: Bias vs Variance and the Discrepancy Principle

Let us suppose that there is a true linear model which can explain the data, $Ax = \bar{b}$, that A is known and ill-conditioned. Let us suppose our measurements are noisy, $b = \bar{b} + \epsilon$, and we want to regularize the problem: $A_\delta x_\delta = b$.

The error $e_\delta = x_\delta - x$ is unknown, since x is unknown, but we may adopt an *error indicator* as the **prediction error** p_δ :

$$p_\delta = Ae_\delta = Ax_\delta - Ax = Ax_\delta - b + \epsilon \quad (4.7)$$

Also p_δ is not directly computable, but it can be estimated if we characterize the noise present in the data.

In questo modo è possibile scegliere un parametro di regolarizzazione ottimale δ che renda minimo l'errore di predizione. A tale scopo è utile considerare la norma-2 al quadrato, o errore quadratico medio (*MSE, Mean Squared Error*) dell'errore di predizione:

$$MSE = \|p_\delta\|_2^2 \quad (4.8)$$

Ora, scomponiamo l'errore nelle sue componenti fondamentali. Tenendo in considerazione TSVD e Tichonov come metodi di regolarizzazione, osserviamo che la soluzione regolarizzata x_δ dipende linearmente dai dati misurati:

$$x_\delta = R_\delta b \quad (4.9)$$

dove R_δ é la matrice inversa regolarizzata, facilmente calcolabile nei due casi, data la decomposizione SVD di $A = U\Sigma V^T$ e la sua suddivisione a blocchi in cui V_1, U_1 e Σ_1 corrispondono alle componenti singolari per le quali $\sigma_i \geq \sqrt{\delta}$:

$$\begin{aligned} R_\delta &= V_1 \Sigma_1^{-1} U_1^T && (\text{TSVD}) \\ R_\delta &= (A^T A + \delta I)^{-1} A^T && (\text{Tichonov}) \end{aligned} \quad (4.10)$$

Definiamo ora per comoditá la matrice $M_\delta = A R_\delta$. Si puó dimostrare [32] che il *MSE* del residuo puó essere scomposto in bias, varianza di modello (*errore riducibile*) e varianza del rumore (*errore irriducibile*):

$$E \left\{ \frac{1}{N} \|b - Ax_\delta\|^2 \right\} = E \left\{ \frac{1}{N} \|\bar{b} - Ax_\delta\|^2 \right\} - \frac{2\sigma^2}{N} \text{trace}(M_\delta) + \sigma^2 \quad (4.11)$$

dove N é il numero di campioni e $\text{trace}(M_\delta)$ é pari alla somma degli elementi sulla diagonale di M_δ . Da qui si ricava immediatamente una formula che d il valore atteso del valore quadratico medio (*MSE*) dell'errore di predizione p_δ (4.8):

$$E \left\{ \frac{1}{N} \|\bar{b} - Ax_\delta\|^2 \right\} = E \left\{ \frac{1}{N} \|b - Ax_\delta\|^2 \right\} + \frac{2\sigma^2}{N} \text{trace}(M_\delta) - \sigma^2 \quad (4.12)$$

Tutti i metodi che vediamo nelle sottosezioni che seguono cercano di stimare $E \left\{ \frac{1}{N} \|\bar{b} - Ax_\delta\|^2 \right\}$.

4.4.1 Il metodo dello stimatore unbiased del rischio predittivo

Lo stimatore da minimizzare é proprio:

$$U(\delta) = \frac{1}{N} \|b - Ax_\delta\|^2 + \frac{2\sigma^2}{N} \text{trace}(M_\delta) - \sigma^2 \quad (4.13)$$

il cui valore atteso é uguale al valore atteso (4.12). Esso richiede la conoscenza di σ^2 . Si veda [32] per l'implementazione. Se é nota la SVD di A , il calcolo di questo stimatore diventa immediato, altrimenti il calcolo di $\text{trace}(M_\delta)$ pu diventare oneroso e si cerca di approssimarla [32].

4.4.2 The Discrepancy Principle

Following the *Discrepancy Principle* of Morozov: said σ the standard deviation od the noise, we choose as a regularization parameter the biggest δ such that:

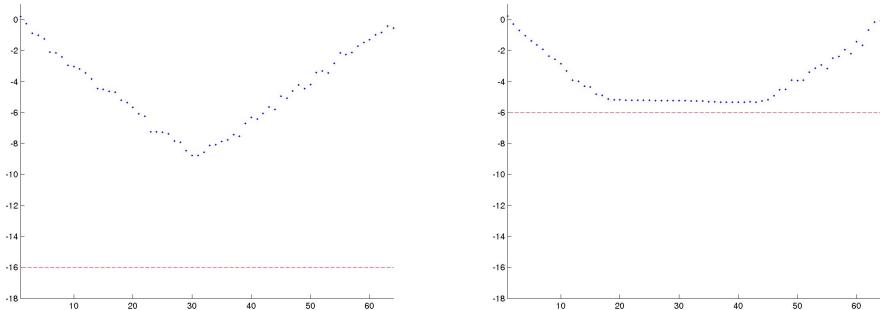
$$\frac{1}{N} \|b - Ax_\delta\|^2 \leq \sigma^2 \quad (4.14)$$

Esso è motivato dal fatto che quando $A_\delta \approx A$, si ha che $\frac{1}{N}||b - Ax_\delta||^2 \approx \frac{1}{N}||\epsilon||^2 = \sigma^2$.

Vediamo un esempio: ??

Costruiamo una matrice A a valori casuali, ma con un andamento dei valori singolari pre-assegnato e dunque con un numero di condizionamento pre-assegnato. A tale scopo, prendiamo due matrici M_U ed M_V a valori casuali e da queste ricaviamo altrettante matrici ortogonali U e V , mediante fattorizzazione QR. A questo punto, definita una matrice diagonale Σ con valori diagonali pari ai valori singolari desiderati, costruiamo la matrice A dalla sua SVD: $A = U\Sigma V^T$.

I risultati sono presentati in Figura 4.1: la scelta dell'indice di troncamento avviene partendo dal valore uno e scegliendo il primo valore per il quale la norma-2 del residuo è prossima alla deviazione standard del rumore di misura.



Residuo della soluzione ai minimi quadrati con SVD troncata, nel caso di rumore basso ($std_{err} = 10^{-16}$, a sinistra) e nel caso di rumore più elevato ($std_{err} = 10^{-6}$, a destra)

4.5 Alternative approaches to the SVD for rank-deficient problems: rank-revealing QR

There are alternative methods to solve rank-deficient or severely ill-posed systems $Ax = b$. A fundamental example is the *rank-revealing QR factorization*.

The term *rank-revealing upper triangular form* denotes the generic $m \times n$ structure of rank r :

$$\tilde{T} = \begin{pmatrix} T_{11} & T_{12} \\ 0 & 0 \end{pmatrix} \quad (4.15)$$

where T_{11} is a $r \times r$ nonsingular upper triangular matrix and T_{12} is $r \times (n - r)$. It explicitly displays its rank: the first r rows are linearly independent, and the remaining rows are zero.

Then, to solve a linear system $Ax = b$ with a rank-revealing procedure, a widely used approach is based on finding nonsingular matrices $C \in \mathcal{R}^{m \times m}$ and $B \in \mathcal{R}^{n \times n}$ such that:

$$CAB = \tilde{T} \quad (4.16)$$

A general matrix A can be reduced to rank-revealing upper-triangular form using orthogonal transformations, e.g. Householder. Unlike the nonsingular case, here there is the need for column interchange to ensure that the rank is fully revealed, i.e. *column selection*; a popular technique is *column pivoting*. In exact arithmetic, when the remaining matrix is zero or null, the reduction is terminated; this happens when the rank r is reached, i.e. after r steps.

The pivot column is taken as the column of largest Euclidean norm in the remaining matrix. A popular alternative is to choose the “least reduced” column in the remaining matrix, i.e. the column with the largest ratio of present to original Euclidean length, i.e. the “most independent column”.

When the pivot column is chosen as the column of largest norm in the remaining matrix, R is a normalized upper triangular matrix with the additional property that its diagonal elements are nonincreasing in magnitude:

$$|r_{11}| \geq \dots \geq |r_{rr}| \gg 0 \quad (4.17)$$

With the “most independent column” strategy, this is not guaranteed.

The 2-norm of the solution is not minimum, as in the pseudo-inverse (SVD), and depends on the chosen permutation P_c .

Now, in exact arithmetic the stopping condition is precise. In finite precision, the “zero” is replaced by “small”, and the difficulty is to define a precise characterization of “small”. By analogy with the case of singular values in the SVD, A is regarded as having rank r if the 2-norm (i.e. the biggest singular value) of the remaining matrix is negligible after r steps. It can be demonstrated that a small remaining matrix is a guaranteed indication of at least one small singular value and, actually:

$$\sigma_{\max}(R_{22}) \geq \sigma_{r+1}(A) \quad (4.18)$$

Conversely, it cannot be guaranteed that small singular values of A are revealed by a small remaining matrix: there is a famous counterexample, the Kahan matrix. Anyway, despite this example, near rank-deficiency is almost always revealed in practice by a small remaining column during Householder reduction and the QR factorization with column interchanges is a reliable and popular technique for estimating the rank of a general matrix.

To determine if the pivot column length can be considered negligible, one usually compare the original length of that column with the length of its sub-column in the current remaining matrix, using a tolerance ϵ_r , i.e. one checks if the original column has been essentially annihilated by the transformations applied so far. Usually, ϵ_r is set comparable to the machine precision ϵ_m or to a more conservative $\sqrt{\epsilon_m}$.

Although the same rank-revealing procedure could be made with the *LU* factorization, it may fail completely to detect the existence of small singular values, hence it is not used.

Capitolo 5

Numerical methods for nonlinear least-squares problems

A well-known method [20] to compute the minimum $\hat{\theta} \in R^{n_\theta}$ of a *cost function* (or *objective function*) $\mathcal{F}_N(\theta) : R^{n_\theta} \rightarrow R$, expressed in the form of the squared sum of nonlinear residuals

$$V_N(\theta) = \left(\frac{1}{N} \sum_{n=1}^N r_n^2(\theta) \right)^{1/2}, \quad \mathcal{F}_N(\theta) = V_N(\theta)^2 \quad (5.1)$$

, where the nonlinear functions $r_n(\theta) : R^{n_\theta} \rightarrow R$ are the *residuals* and are defined by the specific problem to be solved, is that of applied a numerical optimization algorithm, that becomes very attractive here, with a quadratic cost function. Let $\nabla \mathcal{F}_N(\theta) \in R^{n_\theta}$ and $\nabla^2 \mathcal{F}_N(\theta) \in R^{n_\theta \times n_\theta}$ respectively the gradient (vector) and the Hessian (matrix) of the cost function (5.1).

In the nonlinear case we loose the possibility of adopting a direct algorithm and we loose also the global convexity of the problem, where a local minimum is also the global minimum. Here we look for a **local convexity**, that corresponds to a **linearization** of the residuals around a point in the parameter space.

The algorithms for nonlinear optimization are necessarily *iterative*. Therefore, starting from an initial estimate $\hat{\theta}^{(0)}$ they converge by successive approximations to the optimum, precisely a *local minimum*. They are grouped mainly in two families: *line-search* and *trust-region* algorithms. *Line-search* algorithms follow an iterative scheme in which, at each iteration k :

1. a *descent direction* $d^{(k)}$ is chosen; by definition, this can be every direction d for whom the directional derivative $\nabla \mathcal{F}_N(\theta)^T \cdot d$ be negative, i.e.:

$$\nabla \mathcal{F}_N(\theta^{(k)})^T \cdot d^{(k)} < 0$$

2. a minimum along such direction is chosen (from here the term “line search”), that is the step length $\alpha^{(k)}$ to do along $d^{(k)}$ to reach such a minimum point;
3. the estimate is then updated: $\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + \alpha^{(k)}d^{(k)}$.

Trust-region algorithms follow an iterative scheme in which, at each iteration k :

1. a quadratic model function $m^{(k)}$ is build, such that it is a good approximation of the cost function (5.1) in a neighborhood of the current estimate $\hat{\theta}^{(k)}$:

$$m^{(k)}(p) = \mathcal{F}_N(\hat{\theta}^{(k)}) + p^T \nabla \mathcal{F}_N(\hat{\theta}^{(k)}) + \frac{1}{2} p^T B^{(k)} p \quad (5.2)$$

where matrix $B^{(k)}$ is the Hessian $\nabla^2 \mathcal{F}_N(\hat{\theta}^{(k)})$ or its approximation;

2. the constrained optimization problem $p^{(k)} = \arg \min_p m^{(k)}(p)$ is solved, where p must remain inside a (*trust-region*) centered in $\hat{\theta}^{(k)}$;
3. the estimate is then updated: $\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} + p^{(k)}$ and also the dimension of the trust-region.

For the nonlinear least squares problem we will see the well-known Gauss-Newton algorithm (line-search, par. 5.2) and the well-known Levenberg-Marquardt algorithm (trust-region, par. 5.3). Both are based on the Newton’s method (par. 5.1).

Before exploring numerical methods for this problem, let us see two general issues that comes out in the nonlinear case:

- the non-convergence to the global minimum: H.1
- the non-uniqueness of the solution: H.2

5.1 Newton’s method

Let $\psi_\theta \in R^{N \times n_\theta}$ be the sensitivity matrix of the residuals $r(\theta) = [\dots r_n(\theta) \dots]^T_{n=1,\dots,N}$ w.r.t. parameters changes:

$$\psi_\theta = \begin{bmatrix} \vdots \\ \nabla r_n(\theta)^T \\ \vdots \end{bmatrix} \quad (5.3)$$

Use the Newton's method to find a stationarity point for $\mathcal{F}_N(\theta)$, i.e. a point where the gradient $\nabla \mathcal{F}_N(\theta)$ vanishes, and assume that the Hessian in such a point be positive definite. Note that the zeroing of the gradient corresponds to a local minimum, and not to the global one. Global optimization is not guaranteed, in general, a priori. Then, the update equation for the parameters $\hat{\theta}^{(k)}$ at k -th iteration, becomes:

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} - \left[\nabla^2 \mathcal{F}_N(\hat{\theta}^{(k)}) \right]^{-1} \cdot \nabla \mathcal{F}_N(\hat{\theta}^{(k)}) \quad (5.4)$$

where the gradient, thanks to the quadratic structure of the cost function (5.1), becomes:

$$\nabla \mathcal{F}_N(\theta) = \frac{2}{N} \sum_{n=1}^N r_n(\theta) \nabla r_n(\theta) = \frac{2}{N} \psi_\theta^T r(\theta) \quad (5.5)$$

and the Hessian:

$$\begin{aligned} \nabla^2 \mathcal{F}_N(\theta) &= \frac{2}{N} \sum_{n=1}^N \nabla r_n(\theta) \nabla r_n(\theta)^T + \frac{2}{N} \sum_{n=1}^N r_n(\theta) \nabla^2 r_n(\theta) \\ &= \frac{2}{N} \psi_\theta^T \psi_\theta + \frac{2}{N} \sum_{n=1}^N r_n(\theta) \nabla^2 r_n(\theta) . \end{aligned} \quad (5.6)$$

Note that the update in the Newton's direction with a unitary step, if the Hessian $\nabla^2 \mathcal{F}_N(\hat{\theta}^{(k)})$ is positive definite, reaches exactly the minimum of the model function $m^{(k)}(p)$ (5.2). The efficacy of this update depends on how much the model $m^{(k)}(p)$ approximates the cost function $\mathcal{F}_N(\hat{\theta}^{(k)})$ and, in general, it is not guaranteed even that $\mathcal{F}_N(\hat{\theta}^{(k+1)}) < \mathcal{F}_N(\hat{\theta}^{(k)})$. For this reason it is commonly used a variable step $\mu^{(k)} \leq 1$, i.e. the so-called *damped-Newton* method:

$$\hat{\theta}^{(k+1)} = \hat{\theta}^{(k)} - \mu^{(k)} \cdot \left[\nabla^2 \mathcal{F}_N(\hat{\theta}^{(k)}) \right]^{-1} \cdot \nabla \mathcal{F}_N(\hat{\theta}^{(k)}) \quad (5.7)$$

where $\mu^{(k)}$ is set initially to 1 and is halved until the cost function is effectively diminished. In this way it is guaranteed a local convergence of the method.

To compute $\psi_{\hat{\theta}^{(k)}}$, we may approximate the derivative with a centered finite difference:

$$\psi_{\hat{\theta}^{(k)}}(:, j) = \frac{\partial}{\partial \theta_j} r(\hat{\theta}^{(k)}) = \frac{r(\hat{\theta}^{(k)} + \frac{h_{\theta_j}}{2}) - r(\hat{\theta}^{(k)} - \frac{h_{\theta_j}}{2})}{h_{\theta_j}} \quad (5.8)$$

which presents substantially two issues:

- the step h_{θ_j} must satisfy a trade-off: it should be not too small, to avoid cancellation problems, even amplified by a small denominator; it should be not too big, otherwise the approximation of the derivative becomes poor.
- the computation of $\psi_{\hat{\theta}^{(k)}}$ is in general quite heavy.

5.2 Metodo di Gauss-Newton

To simplify the method (especially the computation of the Hessian), we may assume that the residuals be, locally, quasi-linearly dependent from the parameters and find an update that solves the followin (overdetermined) linear system:

$$\psi_{\hat{\theta}^{(k)}} \delta\theta = -r(\hat{\theta}^{(k)}) . \quad (5.9)$$

The solution $\delta\theta$ is a good update is it remains in the region of validity of the linearity assumption. In practice, we did substitute the quadratic model (5.2) of the cost function (5.1) with the following linear model $r(\hat{\theta}^{(k)})$:

$$m_{linr}^{(k)}(p) = r(\hat{\theta}^{(k)}) + \psi_{\hat{\theta}^{(k)}} p \quad (5.10)$$

The system (5.9) is over-determined and it is then solved in a least-squares sense. If the matrix $\psi_{\hat{\theta}^{(k)}}$ has full rank and is well-conditioned, we can use the *normal equations*:

$$\psi_{\hat{\theta}^{(k)}}^T \psi_{\hat{\theta}^{(k)}} \delta\theta = -\psi_{\hat{\theta}^{(k)}}^T r(\hat{\theta}^{(k)}) \quad (5.11)$$

and this is exactly what is done from (5.7) when the Hessian (5.6) is approximated by neglecting the second term, i.e.:

$$\nabla^2 \mathcal{F}_N(\theta) = \frac{2}{N} \psi_{\hat{\theta}^{(k)}}^T \psi_{\hat{\theta}^{(k)}} .$$

If the matrix $\psi_{\hat{\theta}^{(k)}}$ is ill-conditioned or has not full rank, this system can be solved with the QR factorization or the SVD, cfr. sec. 3.3.4.

5.3 The method of Levenberg-Marquardt

When the residual $r(\theta)$ is sufficiently small, we may simplify the Hessian (5.6) and use the Gauss-Newton method, but when it is not so small this method presents practical convergence problems. In this case, the problem can be solved more efficiently with method of Levenberg e Marquardt: at each iteration we solve the problem

$$\delta\theta = \arg \min_p \frac{1}{2} \| \psi_{\hat{\theta}^{(k)}} p + r(\hat{\theta}^{(k)}) \|_2^2 , \quad \| p \| \leq \Delta^{(k)} \quad (5.12)$$

where $\Delta^{(k)}$ is the radius of the *trust-region*. When the Gauss-Newton update falls into the current trust-region, accept the update. Otherwise, it can be demonstrated that, solving the problem (5.12) with an adequate $\lambda^{(k)} > 0$ the solution $\delta\theta$ satisfy $\| \delta\theta \| = \Delta^{(k)}$:

$$\left(\psi_{\hat{\theta}^{(k)}}^T \psi_{\hat{\theta}^{(k)}} + \lambda^{(k)} I \right) \delta\theta = -\psi_{\hat{\theta}^{(k)}}^T r(\hat{\theta}^{(k)}) . \quad (5.13)$$

Note that for $\lambda^{(k)} = 0$ we have the Gauss-Newton method, and that for $\lambda^{(k)} \rightarrow \infty$ we arrive to the steepest-descent, with a step $\delta\theta \rightarrow 0$. Remains to decide how to choose $\lambda^{(k)}$ and $\Delta^{(k)}$.

5.3.1 Choice of $\lambda^{(k)}$

The search for $\lambda^{(k)}$ can be made by a root-finding algorithm, since $\delta\theta$ is a nonlinear function of $\lambda^{(k)}$:

$$\phi(\lambda) = \left\| \delta\theta(\lambda^{(k)}) \right\| - \Delta^{(k)} = 0 \quad (5.14)$$

In particular, given the initial value $\lambda^{(k),0} > 0$, we proceed iteratively in the following way; at the i -th iteration:

1. update the QR factorization (for $i == 0$ it is computed from the scratch)

$$Q_{\lambda^{(k)},i} \begin{bmatrix} R_{\lambda^{(k)},i} \\ 0 \end{bmatrix} = \begin{bmatrix} \psi_{\hat{\theta}^{(k)}} \\ \sqrt{\lambda^{(k),i}} I \end{bmatrix} \quad (5.15)$$

; note that the factor $R_{\lambda^{(k)},i}$ is such that $R_{\lambda^{(k)},i}^T R_{\lambda^{(k)},i} = \psi_{\hat{\theta}^{(k)}}^T \psi_{\hat{\theta}^{(k)}} + \lambda^{(k),i} I$

2. solve the linear system $R_{\lambda^{(k)},i}^T R_{\lambda^{(k)},i} \delta\theta = -\psi_{\hat{\theta}^{(k)}}^T r(\hat{\theta}^{(k)})$;
3. solve the linear system $R_{\lambda^{(k)},i}^T q_i = \delta\theta$
4. update λ : $\lambda^{(k),i+1} = \lambda^{(k),i} + \left(\frac{\|\delta\theta(\lambda^{(k),i})\|}{\|q_i\|} \right)^2 \left(\frac{\|\delta\theta(\lambda^{(k),i})\| - \Delta^{(k)}}{\Delta^{(k)}} \right)$

At the end of the iterations (that in practice limit to 2-3) set $\delta\theta^{(k)} = \delta\theta$.

5.3.2 Choice of $\Delta^{(k)}$

The update of $\Delta^{(k)}$ is made by heuristic arguments, based mainly on the quantity $\rho^{(k)}$, which is equal to the ratio between the actual reduction of the cost function and that was predicted by the model:

$$\rho^{(k)} = \frac{\mathcal{F}_N(\hat{\theta}^{(k)}) - \mathcal{F}_N(\hat{\theta}^{(k)} + \delta\theta^{(k)})}{m^{(k)}(0) - m^{(k)}(\delta\theta^{(k)})} \quad (5.16)$$

If $\rho^{(k)} \approx 1$ it means that the model is a good approximation of the cost function and, therefore, we can expand the *trust region*; if $\rho^{(k)}$ is next to zero or negative the region must be reduced; if $\rho^{(k)}$ stays in the middle, we leave $\Delta^{(k)}$ unchanged.

Let us see some numerical example in `cap7_3_esempi_LevenbergMarquardt.ipynb`.

Capitolo 6

Sparse solutions of underdetermined linear systems

Underdetermined linear systems will have none or infinite solutions. A solution algorithm must choose one of them and, first, the basis in which to express this solution. If the matrix has full row-rank, we can get a unique solution, with the QR factorization or the SVD. This reduces to consider its transpose or to do a proper choice of columns, as we see in the following algorithms.

The QR with column pivoting finds a solution which is not necessarily of minimum norm. We obtain:

$$Q^T AP_c = [R_1 \ R_2] \quad (6.1)$$

where $R_1 \in \mathbb{R}^{m \times m}$ is upper triangular and P_c is a column permutation matrix (and therefore applied to the right of A). The system $Ax = b$, left-multiplied by Q^T , becomes:

$$(Q^T AP_c) (P_c^T x) = [R_1 \ R_2] \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = Q^T b \quad (6.2)$$

where

$$P_c^T x = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

Now, R_1 is non singular since A has full row-rank, and therefore we find the solution $z_1 = R_1^{-1} Q^T b$ and $z_2 = 0$, but its norm depends on the chosen permutation P_c .

As an alternative algorithm, doing the QR of A^T , we get the minimum norm solution (and with a lower operation count):

$$A^T = Q R = Q \begin{bmatrix} R_1 \\ 0 \end{bmatrix} \quad (6.3)$$

and the system $Ax = b$ becomes:

$$(Q R)^T x = \begin{bmatrix} R_1^T & 0 \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} = b \quad (6.4)$$

where

$$Q^T x = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

. The matrix R_1 is non singular since A has full row-rank (and therefore A^T has full column-rank) and we get the solution by setting $z_2 = 0$, and hence

$$x = Q \begin{bmatrix} R_1^{-T} b \\ 0 \end{bmatrix}$$

, that has minimum norm, since every $z_2 \neq 0$ would add a linear combination of columns which are orthogonal to the previous ones.

With the truncated SVD we can get the solution, as is already known: $x^* = V_1 \Sigma_1^{-1} U_1^T b + V_2 z$ with z an arbitrary vector of \Re^{n-r} and by setting $z = 0$ we get the minimum-norm solution.

Note that if $b \in \Re^m$, $x \in \Re^n$ and $m < n$, to solve this system brings us to a number of unknowns (i.e. parameters, in a learning problem) x bigger than the number of data b , i.e. a number of d.o.f. bigger than the information acquired: this is misleading, in general. But, if we add a sparsity condition on the parameters x , in the sense that we assume that some of them will be zero, but we don't know which (otherwise we would simply cancel the corresponding columns of A), then it opens up an interesting new scenario, as shown by these two simple examples taken from image processing and Fourier analysis of signals. I

In recent times became available *wide data-sets* that contains measurements of a lot of variables and represents a big amount of possible regression variables and corresponding parameters. Indeed, very often turns out that only a small subset of variables is capable to describe adequately the target variable: the solution to find is, with respect to the whole *dictionary* of variables, sparse. In this context, the set of columns of a matrix becomes a *dictionary*: a *complete* dictionary generates all the features space, an *overcomplete* dictionary generates the feature space with redundancy, and this redundancy is represented by the matrix null-space.

To get a sparse solution, we must change considerably our numerical methods, since the minimization in the l_2 -norm does not give sparse solutions! For sparsity it is convenient to consider also the l_1 -norm and minimize in mixed norm.

6.1 Basic concepts and properties

To obtain the sparsest solution of a system $Ax = b$ we should minimize in l^0 -“norm”:

$$\min \{||x||_0, \quad x \in \mathcal{R}^n, \quad Ax = b\} \quad (6.5)$$

said the sparse recovery problem (note that the 0-“norm” is not properly a norm and the minimizing function is not continuous), which corresponds to an NP-hard combinatorial problem and it is not tractable, in this form.

Thanks to a recent result, (6.5) is equivalent to a constrained minimization problem in l^1 -norm:

$$\min \{||x||_1, \quad x \in \mathcal{R}^n, \quad Ax = b\} \quad (6.6)$$

which is a convex minimization problem, called *basis pursuit*, tractable by numerical methods. An unconstrained version of (6.6) is:

$$\min_{\mathbf{x}} \|\mathbf{x}\|_1 + \frac{1}{\lambda} \|Ax - \mathbf{b}\|_2^2 \quad (6.7)$$

or, equivalently, the famous LASSO formulation:

$$\min_{\mathbf{x}} \|Ax - \mathbf{b}\|_2^2 + \lambda \|\mathbf{x}\|_1 \quad (6.8)$$

note the regularization character of this form, called “shrinkage”. For small enough values of λ , the solution of (6.8) approximates (6.6).

It is easy to see why the 1-norm sparsify much better than the 2-norm:

In the following subsections we see some theoretical properties that can guarantee existence and unicity of the solution to (6.6).

6.1.1 Null Space Property (NSP)

Theorem 6.1.1. *The vector $x^* \in \mathcal{R}^n$ is a solution of (6.6) iff $\forall v \in \mathcal{N}(A)$, where $\mathcal{N}(A)$ is the null-space of A ,*

$$\left| \sum_{x^* \neq 0} \text{sign}(x_i^*) v_i \right| \leq \sum_{x^* == 0} |v_i| \quad (6.9)$$

and x^* is unique iff

$$\left| \sum_{x^* \neq 0} \text{sign}(x_i^*) v_i \right| < \sum_{x^* == 0} |v_i| \quad (6.10)$$

6.1.2 The *spark* of a matrix

Definition 6.1.2. The *spark* of a matrix A is the smallest number k of columns which are linearly dependent, i.e.

$$\text{spark}(A) = \min_{x \neq 0} \{||x||_0, \quad x \in \mathcal{N}(A)\} \quad (6.11)$$

It is clear that $\text{spark}(A) \leq \text{rank}(A) + 1$.

Theorem 6.1.3 (Donoho and Elad, 2003). *The underdetermined linear system $Ax = b$ has a unique sparse solution x_s if $||x_s||_0 \leq \text{spark}(A)/2$.*

6.1.3 Mutual coherence

Definition 6.1.4. Let $G = A^T A$, which is a square matrix $n \times n$. The *mutual coherence* of A is

$$M(A) = \max_{1 \leq i, j \leq n, i \neq j} |g_{ij}| \quad (6.12)$$

Clearly, $M \leq 1$. Interestingly, $M(A)$ cannot be too small, as stated by the following Lemma:

Lemma 6.1.5. *Assume that $A \in \mathcal{R}^{m \times n}$ is of full rank. Then, $M(A) \geq \sqrt{\frac{n-m}{m(n-1)}}$.*

The bound is called *Welch's bound*.

Theorem 6.1.6 (Gribonval and Nielsen, 2003). *Any solution of the system $Ax = b$, if its sparsity s satisfies*

$$s \leq \frac{1}{2}(1 + 1/M(A)) \quad (6.13)$$

then it is also the (unique) solution to (6.5) and (6.6).

6.2 Mixed 1- and 2-norm regularization (*sparse recovery*)

Therefore, we should solve problems where terms in l^2 -norm are mixed with terms in l^1 -norm. This creates an algorithmic complication, since the solution must be found iteratively and the algorithms that minimize the l^2 terms are in general faster than those for the l^1 terms and to combine them in a single algorithm it is not trivial. An approach recently appreciated is to split the algorithm as done by ADMM (the Alternate Direction of Multipliers Method):

$$\min_{x,z} f(x) + g(z) \quad \text{with the constraint} \quad x = z \quad (6.14)$$

that applied to LASSO becomes:

- $x_{k+1} = (A^T A + \rho I)^{-1} (A^T b + \rho(z_k - u_k))$
- $z_{k+1} = S_{\lambda/\rho}(x_{k+1} + u_k)$
- $u_{k+1} = u_k + x_{k+1} - z_{k+1}$

where ρ and λ are regularization parameters, and S is the *soft-thresholding operator*:

$$S_k(a) = \begin{cases} a - k, & a > k \\ 0 & |a| \leq k \\ a + k & a < -k \end{cases}$$

Vediamo piú in dettaglio gli esempi di compressed sensing fotografico e di analisi spettrale: **I**.

Capitolo 7

Analisi di Fourier nel discreto

Come abbiamo visto al Capitolo 1, spesso i dati sono organizzati in sequenze, dette anche *segnali a tempo discreto*, in cui l'indice ha di solito il significato di *istante temporale*. Le operazioni di calcolo effettuate su queste sequenze prendono il nome di *elaborazione numerica dei segnali* (in inglese, *Digital Signal Processing o DSP*) [21].

Uno strumento matematico fondamentale per il DSP é la *Trasformata di Fourier Discreta* (Discrete Fourier Transform, o brevemente DFT) [5]. Infatti, é spesso vantaggioso analizzare i segnali anche nel dominio delle frequenze, come vedremo nel corso del presente capitolo. Inoltre, esiste un preciso legame tra analisi di Fourier di un segnale a tempo discreto ed analisi di Fourier di una funzione (cioé di un segnale a tempo continuo).

7.1 Sequenze di Dati, Segnali a tempo discreto e Serie storiche

Quando i dati vengono raccolti in funzione di una variabile scalare, si producono una o piú **sequenze di dati** e tale variabile viene detta *indice*. Situazione tipica é quella delle misure prese ad intervalli di tempo successivi: si parla in tal caso di *segnalet a tempo discreto* o *serie-temporale*.

Un **segnalet a tempo discreto** é una sequenza di dati $u(t_k)$, presi agli istanti t_k , dove $k = 0, 1, \dots$ é la variabile (o indice) temporale **discreta**. Spesso tali sequenze vengono ottenute *campionando in modo uniforme* un fenomeno continuo, ovvero $t_k = k \cdot T$, dove T é un intervallo di tempo prefissato, usualmente chiamato *periodo di campionamento*, e quindi misurando il fenomeno ad intervalli di tempo regolari di lunghezza pari a T , producendo in questo modo una sequenza di dati o *campioni*.

Vediamo un esempio in C.1.

In generale, il riferimento alla variabile continua t_k non è necessario all'analisi della sequenza di dati, quindi per semplicità porremo spesso $u(t_k) \equiv u(k)$.

Nelle applicazioni dell'analisi numerica dei dati è molto diffuso l'utilizzo dell'analisi di Fourier di una serie di dati (o *segnaletico discreto*), che vedremo in dettaglio al Capitolo 7. Pertanto, è fondamentale approfondire la rappresentazione e le proprietà dei segnali discreti periodici. Un segnale discreto $u(k)$ è detto **periodico** di periodo N se:

$$u(k + N) = u(k) \quad \forall k$$

Un caso notevole di segnale discreto periodico, proprio perché è l'espressione del generico *modo di Fourier discreto*, è la sequenza di dati, a valori nel campo complesso, detta *esponenziale discreto*:

$$A \cdot e^{j\omega_v k} \quad (7.1)$$

dove A è l'*ampiezza* e ω_v è la *pulsazione discreta*, e le sequenze reali di tipo sinusoidale a tempo discreto:

$$A \cdot \cos(\omega_v k + \phi) \quad , \quad A \cdot \sin(\omega_v k + \phi)$$

dove sia ω_v che ϕ sono espresse in *radiani*. Analogamente al caso continuo, essi sono legati dalle *relazioni di Eulero*

$$e^{j\omega_v k} = \cos(\omega_v k) + j \cdot \sin(\omega_v k)$$

e

$$A \cdot \cos(\omega_v k + \phi) = \frac{A}{2} e^{j\phi} e^{j\omega_v k} + \frac{A}{2} e^{-j\phi} e^{-j\omega_v k} \quad (7.2)$$

A differenza del caso continuo, l'esponenziale discreto è periodico di periodo 2π rispetto alla pulsazione:

$$e^{j(\omega_v + 2\pi)k} = e^{j2\pi k} e^{j\omega_v k} = (\text{essendo } k \text{ intero}) = 1 \cdot e^{j\omega_v k} = e^{j\omega_v k}$$

e dunque richiede di essere analizzato solo in un intervallo di pulsazioni di lunghezza 2π .

Inoltre, l'esponenziale discreto non presenta oscillazioni a frequenza sempre crescente all'aumentare della pulsazione discreta ω_v : esso aumenta da 0 a π e poi diminuisce fino a 2π . Le oscillazioni a più alta pulsazione (cioè la più alta pulsazione continua) si hanno dunque per $\omega_v = \pi$:

vediamolo nell'esempio C.2.

Vediamo ora un aspetto fondamentale. Dato un intero N , vediamo quali sono le pulsazioni discrete ω_ν , $\nu = 1, 2, \dots$, per cui N campioni contengono un numero intero di periodi dell'esponenziale discreto (7.1). Si ha:

$$e^{j\omega_\nu N} = 1$$

ovvero

$$\omega_\nu N = 2\pi\nu$$

e quindi

$$\omega_\nu = \frac{2\pi\nu}{N} \quad \nu = 1, \dots, N \quad (7.3)$$

dove $\frac{\nu}{N}$ è detta *frequenza normalizzata* (adimensionale). Le sequenze $e^{j\omega_\nu k}$, $k = 1, \dots, N$, sono fondamentali in quanto risultano ortogonali rispetto ad un opportuno prodotto scalare, cfr. Lemma 7.13.

Notare che:

- esistono esattamente N pulsazioni discrete con questo requisito;
- non abbiamo messo alcun riferimento alla variabile tempo; ciò verrà fatto parlando di campionamento di segnali continui, alla sezione 7.2;
- per $\nu = N/2$, che corrisponde alla massima oscillazione, si ha una componente di periodo pari a $1/(\nu/N) = N/\nu = N/(N/2) = 2$ campioni, cfr. Figura ??;
- 2 campioni per periodo, presi in corrispondenza del picco positivo e del picco negativo della sinusoide, sono il minimo necessario per riconoscere l'oscillazione, mentre per descrivere bene un'oscillazione sinusoidale sono sufficienti 8 campioni per periodo.

7.2 Campionamento di segnali continui

Accade di frequente che una sequenza di dati sia stata prodotta mediante *campionamento* uniforme di un segnale continuo. La distanza temporale tra l'acquisizione di due campioni consecutivi della sequenza è detta *periodo di campionamento* (T_c).

In base al ben noto *teorema di Shannon* (o *teorema del campionamento*), il periodo di campionamento T_c , ovvero la *frequenza di campionamento* $f_c = 1/T_c$ (espressa in [Hz]), determina la massima frequenza continua f rappresentabile dalla sequenza discreta, che è pari a $f_c/2$.

Notare che il segnale continuo $e^{j2\pi f t}$, campionato con periodo di campionamento T_c , fornisce l'esponenziale discreto $e^{j\omega k}$ con $\omega = 2\pi f T_c$. Eguagliando questa espressione di ω con la (7.3), si ottiene il legame tra frequenza continua e corrispondente frequenza normalizzata (discreta) $\frac{\nu}{N}$:

$$f T_c = \frac{\nu}{N} \quad (7.4)$$

In genere, i segnali fisici hanno un contenuto di frequenze limitato superiormente; l'intervallo che le contiene è detto *banda* del segnale. Per rappresentare correttamente un segnale continuo a banda limitata mediante una sequenza discreta è necessario campionare ad almeno il doppio della massima frequenza posseduta dal segnale, ovvero al doppio della banda, ovvero alla *frequenza di Nyquist*.

Se si campiona un segnale con frequenza di campionamento inferiore alla frequenza di Nyquist, si introduce un errore di *aliasing* nella sequenza discreta: le componenti del segnale continuo a frequenza maggiore di f_c appaiono nella sequenza come componenti **a bassa frequenza**, compromettendo dunque la bontà della rappresentazione discreta.

Vediamo un esempio: C.3.

Quindi, il fenomeno di aliasing può essere evitato campionando il segnale analogico ad una frequenza sufficientemente alta, ovvero con un periodo di campionamento T sufficientemente piccolo. Per ulteriori approfondimenti, cfr. ad esempio [21] al Paragrafo 1.7.

7.3 La Trasformata di Fourier Discreta (DFT)

Classicamente, l'analisi di Fourier è lo studio di come funzioni definite nel continuo (cioè in tutti gli infiniti punti di un intervallo) possano essere rappresentate mediante funzioni periodiche come seni e coseni. Molti problemi applicativi richiedono invece di fare analisi di Fourier su un insieme di campioni di una funzione continua, ovvero su una funzione valutata in un insieme finito di punti, i cui valori sono quindi una sequenza di dati. La DFT risponde dunque a questa esigenza. Per una trattazione completa sulla DFT consigliamo ad esempio [5].

Ci sono vari modi per arrivare alla DFT: qui la ricaviamo mediante approssimazione della Trasformata di Fourier, mentre in [5] la DFT viene presentata anche come risultato dell'approssimazione di una funzione discreta mediante interpolazione con polinomi trigonometrici.

Sia u una funzione definita nell'intervallo $(-\infty, \infty)$ e sia ivi assolutamente integrabile:

$$\int_{-\infty}^{\infty} |u(t)| dt < \infty \quad (7.5)$$

Allora possiamo definire la funzione \hat{u} , detta *Trasformata di Fourier* di u , nel modo seguente:

$$\hat{u}(f) = \int_{-\infty}^{\infty} u(t)e^{-i2\pi ft}dt \quad (7.6)$$

dove f è detta *frequenza*. Vediamo come la DFT emerge come una naturale approssimazione della (7.6). Innanzitutto, per essere utilizzabile in pratica, u deve essere nulla al di fuori di un intervallo finito $(0, T)$ o poter essere considerata tale. In tal caso:

$$\hat{u}(f) = \int_{-\infty}^{\infty} u(t)e^{-i2\pi ft}dt = \int_0^T u(t)e^{-i2\pi ft}dt \quad (7.7)$$

Ora, approssimiamo tale integrale con un metodo di calcolo numerico. Se suddividiamo l'intervallo di integrazione $(0, T)$ in N sottointervalli di lunghezza $\Delta t = T/N = T_c$ ed applichiamo la regola dei trapezi [24], otteniamo, avendo posto $g(t) = u(t)e^{-i2\pi ft}$ e $t_i = i \cdot \Delta t$:

$$\int_0^T g(t)dt \approx \frac{\Delta t}{2} \left\{ g(t_0) + 2 \sum_{k=1}^{N-1} g(t_k) + g(t_N) \right\} \quad (7.8)$$

ed aggiungendo il requisito di periodicità:

$$g(t_0) = g(t_N) \quad (7.9)$$

che corrisponde a richiedere che:

- $u(t)$ debba essere periodica;
- le funzioni $e^{-i2\pi f t_k} = e^{-i2\pi f T_c k} = e^{-i\omega k}$, e dunque anche la loro combinazione lineare (7.10), soddisfino al requisito di periodicità (7.9). Ricordando la (7.3), questo significa restringere le pulsazioni discrete alle sole $\omega = \omega_v$ e dunque le frequenze nel continuo alle sole $f_v = \frac{\omega_v}{2\pi T_c} = \frac{v}{N} f_c$.

Dunque, otteniamo:

$$\hat{u}(f_v) = \int_0^T g(t)dt \approx \Delta t \sum_{k=0}^{N-1} g(t_k) = \frac{T}{N} \sum_{k=0}^{N-1} u(t_k) e^{-i2\pi f_v t_k} \quad (7.10)$$

da confrontare, previo cambio di variabile $t_k = kT/N$, con la (7.16) e con la (7.21), tenendo conto di un fattore di scala:

$$\hat{u}(f_v) = T U(v) \quad . \quad (7.11)$$

7.4 Rappresentazione dei Segnali Discreti mediante DFT

7.4.1 Sequenze di Dati Periodiche di Durata Illimitata

Possiamo, in questo caso, limitare lo studio della sequenza ad un periodo di questa, di lunghezza N campioni. Definiamo il seguente prodotto scalare discreto:

$$(v(k), w(k))_N = \frac{2\pi}{N} \sum_{k=0}^{N-1} v(k) \cdot \overline{w(k)} \quad (7.12)$$

Lemma (ortogonalitá) "Per le sequenze di tipo esponenziale discreto, considerate alle pulsazioni (7.3), vale:

$$\left(e^{i\omega_v k}, e^{i\omega_\mu k} \right)_N = \frac{2\pi}{N} \sum_{k=0}^{N-1} e^{i \cdot (\nu - \mu) \cdot \frac{2\pi}{N} k} = 2\pi \cdot \delta(\nu - \mu) \quad (7.13)$$

".

Dim: per $\nu = \mu$ é immediato. Vediamo il caso $\nu \neq \mu$:

$$\sum_{k=0}^{N-1} e^{i \cdot (\nu - \mu) \cdot \frac{2\pi}{N} k} = \frac{1 - \left(e^{i \cdot (\nu - \mu) \cdot \frac{2\pi}{N}} \right)^N}{1 - e^{i \cdot (\nu - \mu) \cdot \frac{2\pi}{N}}} = 0$$

Infatti, il numeratore diventa $1 - (\cos(2\pi \cdot (\nu - \mu)) + i \cdot \sin(2\pi \cdot (\nu - \mu)))$ ed il denominatore é sempre $\neq 0$ tranne che per $\nu - \mu = N$ che é impossibile in quanto assumono entrambi valori da 0 a $N - 1$. \diamond

Quindi, le N sequenze di valori $e^{i\omega_\nu k}$, $\nu = 0 \dots N - 1$, di lunghezza N , sono a due a due ortogonali tra loro rispetto a $(\cdot, \cdot)_N$ e quindi costituiscono una base di (uno spazio isomorfo a) \mathbb{C}^N . E' facile verificare che ciò accade anche per le N sequenze di valori $e^{-i\omega_\nu k}$, $\nu = 0 \dots N - 1$, di lunghezza N . Una generica sequenza complessa $u(k)$ di periodo N sarà pertanto esprimibile come combinazione lineare di queste N sequenze, secondo un vettore di coefficienti $U(\nu)$ corrispondenti alle frequenze ω_ν , ovvero, in forma matriciale $W(k, \nu) \cdot U(\nu) = u(k)$:

$$\begin{bmatrix} \vdots & \vdots & \vdots \\ \vdots & e^{i\omega_\nu k} & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} \cdot \begin{bmatrix} \vdots \\ U(\nu) \\ \vdots \end{bmatrix} = \begin{bmatrix} \vdots \\ u(k) \\ \vdots \end{bmatrix} \quad (7.14)$$

ovvero, se il segnale $u(k)$ é discreto e periodico di periodo N , può essere rappresentato in modo univoco mediante una **Serie di Fourier Discreta**:

$$u(k) = \sum_{\nu=0}^{N-1} U(\nu) e^{i\omega_\nu k} \quad k = 0 \dots N - 1 \quad (7.15)$$

A questo punto é opportuno osservare nuovamente la somiglianza con la (7.10). La trasformazione inversa, dato che per il precedente Lemma vale $W^H \cdot W = D$, diventa semplicemente $D^{-1} \cdot W^H \cdot u(k) = U(\nu)$, ovvero:

$$U(\nu) = \frac{1}{N} \sum_{k=0}^{N-1} u(k) e^{-i\omega_\nu k} \quad \nu = 0 \dots N-1 \quad (7.16)$$

dove D é una matrice diagonale con tutti N sulla diagonale, $i = \sqrt{(-1)}$, $\omega_\nu = 2\pi\nu/N$ ed N é il periodo della sequenza. La (7.15) é detta *sintesi*, mentre la (7.16) é detta *analisi* della sequenza di dati $u(k)$. I coefficienti $U(\nu)$ sono detti *coefficienti di Fourier discreti*. Il significato di queste relazioni é che la sequenza $u(k)$ é esprimibile come combinazione lineare di N esponenziali discreti, detti *armoniche*, e che dunque i coefficienti (7.16) sono una rappresentazione equivalente della sequenza $u(k)$.

Sono molto importanti queste relazioni duali tra periodicitá e campionamento/discretizzazione:

- **periodicitá nel tempo \Rightarrow campionamento in frequenza**: basta aumentare N e vedere come gli ω_ν si infittiscono tra 0 e 2π ; al limite, per $N \rightarrow \infty$, il periodo della sequenza é infinito, quindi essa é aperiodica, ed i valori delle frequenze tendono ad un continuo;
- **campionamento nel tempo \Rightarrow periodicitá in frequenza**: la periodicitá in frequenza la si vede semplicemente campionando i segnali continui $e^{j2\pi ft}$ ed $e^{j2\pi(f+Mf_c)t}$ con periodo di campionamento T_c e scegliendo M un intero qualsiasi: la sequenza discreta campionata é la stessa in entrambi i casi. Diminuendo il periodo di campionamento, aumenta la massima frequenza continua rappresentabile dalla sequenza campionata (frequenza di Nyquist); quando il periodo di campionamento tende a zero in frequenza non si ha piú periodicitá.

In generale, la sequenza $U(\nu)$ prodotta dalla DFT assume valori nel campo complesso.

7.4.2 Sequenze di Dati Aperiodiche di Durata Illimitata

In generale, la rappresentazione nel dominio delle frequenze puó essere conveniente anche per rappresentare una sequenza di dati aperiodica, se riteniamo che in essa siano contenute delle componenti periodiche. Infatti, nel dominio delle frequenze esse vengono messe in evidenza.

La trasformazione disponibile per questo caso é la **Trasformata Zeta**:

$$U(z) = Z[u(k)] := \sum_{k=-\infty}^{\infty} u(k) z^{-k} \quad (7.17)$$

e la corrispondente *Trasformata Zeta Inversa*:

$$u(k) = \frac{1}{2\pi i} \oint U(z) z^{k-1} dz \quad (7.18)$$

Essa é l'analogo della trasformata di Laplace per i segnali a tempo continuo. Quando la regione di convergenza contiene il cerchio unitario, $z = e^{j\omega}$, essa contiene come caso particolare la **Trasformata di Fourier** (in inglese *Fourier Transform* o *FT*) della sequenza discreta (infinita) $u(k)$, e che si ottiene ponendo $z = e^{j\omega}$:

$$U(\omega) = FT[u(k)] := \sum_{k=-\infty}^{\infty} u(k) e^{-i\omega k} \quad (7.19)$$

e la corrispondente *Trasformata Inversa di Fourier (IFT)*:

$$u(k) = \frac{1}{2\pi} \int_{2\pi} U(\omega) e^{i\omega k} d\omega \quad (7.20)$$

Notare che la (7.19) 'e simile alla precedente (7.15) ma qui l'indice k é illimitato sia a sinistra che a destra, e che qui la frequenza é definita in un continuo di valori (infatti, la sequenza non é periodica). Considerando un supporto limitato per $u(k)$ ed operando il calcolo in un insieme discreto di pulsazioni ω_v riotteniamo la DFT.

7.4.3 Sequenze di Dati di Durata Limitata

Nell'analisi di una sequenza di dati di durata limitata (N), si perde la distinzione sulla periodicità e la sequenza puó essere ricondotta facilmente ad entrambi i casi precedenti: al primo mediante prolungamento per periodicità ed al secondo mediante prolungamento con valori nulli.

La **Trasformata di Fourier discreta (DFT)** di $u(k)$, diventa:

$$U(\nu) = \frac{1}{N} \sum_{k=0}^{N-1} u(k) e^{-i\omega_\nu k} \quad (7.21)$$

dove $U(\nu)$ é l'ampiezza della componente sinusoidale di $u(k)$ alla frequenza $\omega_\nu = 2\pi\nu/N$. Moltiplicando la (7.21) per $e^{i(2\pi\nu/N)k}$ e sommando per $\nu = 0, \dots, N-1$, si ha:

$$u(k) = TIFD(U(\nu)) := \sum_{\nu=0}^{N-1} U(\nu) e^{i(2\pi\nu/N)k} \quad (7.22)$$

La sequenza $u(k)$ é detta in questo caso *anti-trasformata* di $U(\omega)$ (IDFT). Vediamo subito alcune considerazioni fondamentali:

- le sequenze $U(\omega)$ e $u(k)$ sono entrambe periodiche di periodo N e $U(\omega)$ puó essere determinata esattamente da un solo periodo di $u(k)$;

- la sequenza $U(\omega)$ fornisce le componenti di $u(k)$ alle varie frequenze ed è per questo chiamata *spettro* di $u(k)$;

Per le proprietà della DFT consultare, ad es. [7], pag. 649 .

7.4.4 Sequenze di Dati ad Energia Infinita

Definiamo come **energia** di una sequenza, la somma dei quadrati dei campioni della sequenza.

Per una sequenza di dati avente energia infinita la trasformata Zeta, in generale, non converge, mentre una sequenza ad energia finita ammette rappresentazione mediante trasformata Zeta. Le sequenze di durata finita hanno ovviamente energia finita, e così accade considerando un singolo periodo di una sequenza periodica.

Esistono però molti esempi importanti di sequenze che non hanno energia finita e non sono periodici (ad es. vibrazioni generate da sistemi meccanici, il segnale vocale, la musica, ecc...). Per la rappresentazione dei segnali discreti ad energia infinita spesso è conveniente lavorare nell'ambito dei *segnali casuali discreti* (cfr Paragrafo ??): il concetto fondamentale nella rappresentazione di un segnale ad energia infinita è quindi quello di *processo casuale*. Infatti, applicare il modello di processo casuale ai problemi pratici di elaborazione delle sequenze di dati implica che una particolare sequenza venga interpretata come la *realizzazione* di un processo casuale. In questo senso un processo casuale serve come rappresentazione di un segnale ad energia infinita.

Molte delle proprietà delle sequenze ad energia infinita vengono catturate da una sequenza ad energia finita, chiamata sequenza di *autocorrelazione* o *autocovarianza*, per la quale esiste spesso la trasformata Zeta e la trasformata di Fourier.

7.4.5 Correlazione e Covarianza di Segnali Discreti

La *sequenza di correlazione*, $\text{corr}[u, v](\tau)$, tra due sequenze di dati di lunghezza N è una sequenza discreta di indice τ e di lunghezza N , che descrive quanto due sequenze $u(k)$ e $v(k)$ si assomigliano (si correlano). Essa effettua questo confronto traslando una delle due sequenze di τ campioni, per $\tau = 0, \dots, N - 1$ e calcolando:

$$\text{corr}[u, v](\tau) = \frac{1}{N} \sum_{k=0}^{N-1} u(k) \cdot v(k + \tau) \quad (7.23)$$

ovvero è la media dei prodotti delle componenti delle due serie traslate reciprocamente. Le sequenze $u(k)$ e $v(k)$ possono anche coincidere. In tal caso, si parla di *auto-correlazione*. In generale, è utile considerare la sequenza di correlazione normalizzata rispetto al campione iniziale: $\text{corrnorm}[u, v](\tau) = \text{corr}[u, v](\tau) / \text{corr}[u, v](0)$.

Analogamente, la *covarianza* tra due sequenze di dati di periodo N é una sequenza discreta di periodo N , $\text{cov}[u, v](\tau)$, cosí definita:

$$\text{cov}[u, v](\tau) = \frac{1}{N} \sum_{k=0}^{N-1} (u(k) - m_u) \cdot (v(k + \tau) - m_v) \quad (7.24)$$

dove m_u e m_v sono le medie delle sequenze $u(k)$ e $v(k)$, e quindi é la media dei prodotti delle deviazioni dalla media delle due serie, traslate reciprocamente.

7.4.5.1 Aleatorietá nei Dati

Nel caso in cui la sequenza dei dati abbia una componente aleatoria, se adottiamo una trasformazione del segnale in un equivalente deterministico, possiamo ancora applicare i metodi di analisi delle sequenze deterministiche. Queste trasformazioni usano fondamentalmente l'operatore di aspettazione e sono, nel dominio del tempo, la sequenza di *auto-correlazione*:

$$r_{uu}(\tau) := E \{ u(k) \cdot u(k + \tau) \} \quad (7.25)$$

e la sequenza di *auto-covarianza*:

$$c_{uu}(\tau) := E \{ (u(k) - m_u)(u(k + \tau) - m_u) \} \quad (7.26)$$

dove $E \{ \cdot \}$ é l'operatore di *aspettazione* ed m_u é la media del processo u , mentre nel dominio delle frequenze c'é la funzione di *densitá spettrale di potenza*, che verrá introdotta successivamente.

Notare che, a causa dell'operazione di aspettazione, la sequenza di auto-correlazione (ed analogamente succede per la densitá spettrale di potenza) sono funzioni deterministiche, e di fatto anche delle sequenze discrete di dati. In definitiva, come primo passo di analisi nel caso di segnali casuali, quando é possibile si sostituisce il segnale con la sequenza di autocovarianza e ad essa si applicano le tecniche di analisi per le sequenze deterministiche.

L'auto-correlazione (idem per l'auto-covarianza) é una misura della dipendenza tra i valori di un processo casuale in tempi diversi. In questo senso essa descrive la variazione nel tempo di un segnale casuale.

7.4.5.2 Stima dell'aspettazione

Se i dati sono stati prodotti da un processo ergodico (cfr. par. ??), si hanno le ben note formule per la media \bar{u} :

$$\hat{E} \{ u(k) \} = \bar{u} = \frac{1}{n} \sum_{k=1}^n u(k) \quad (7.27)$$

, la varianza s :

$$\hat{E} \left\{ (u(k) - \bar{u})^2 \right\} = s^2 = \frac{1}{n-1} \sum_{i=1}^n (u(k) - \bar{u})^2 \quad (7.28)$$

e, in generale,

$$\hat{E} \{ f(u(k), y(k)) \} = \frac{1}{n} \sum_{k=1}^n f(u(k), y(k)) \quad (7.29)$$

7.4.6 densità spettrale di potenza

Si ha che,

- la trasformata discreta di Fourier della sequenza di autocorrelazione, e cioè lo *spettro di potenza* $P_{uu}(\omega)$, descrive la distribuzione in frequenza della *potenza media* (cioé del valore quadratico medio) della sequenza di dati. Infatti, l'area sottesa da $P_{uu}(\omega)$ per $0 \leq \omega \leq \pi$ è proporzionale alla potenza media del segnale, e l'integrale di $P_{uu}(\omega)$ per una certa *banda* di frequenze è proporzionale alla potenza del segnale in quella banda.
- la sequenza di autocorrelazione e lo spettro di potenza consentono di descrivere l'elaborazione operata da un sistema DLTI su sequenze ad energia infinita, mediante l'effetto che quel sistema opera sulla sequenza di autocovarianza delle sequenze stesse, come vedremo nel prossimo paragrafo 7.5.

7.4.7 Esempi applicativi

La Trasformata di Fourier discreta (DFT) è uno strumento matematico fondamentale, reso estremamente conveniente dall'algoritmo detto Fast Fourier Transform (o FFT). I suoi utilizzi sono i più svariati, ad esempio:

- la telefonia mobile non esisterebbe senza la DFT;
- nel settore biomedico, l'inversione della Trasformata di Radon nella TAC viene fatta generalmente mediante FFT;
- la Magneto-Risonanza per Immagini (MRI) misura sperimentalmente dei coefficienti di Fourier (l'immagine la si ottiene anti-trasformando);
- nell'analisi di immagini, la FFT è spesso utilizzata per filtrare le immagini da sfocamento ed altri disturbi;
- nell'analisi di serie storiche, la FFT è utilizzata per riconoscere e misurare componenti stagionali;

- nella soluzione numerica di equazioni differenziali alle derivate parziali la FFT è utilizzata per risolvere in modo computazionalmente efficiente quei problemi ai limiti la cui soluzione può essere espressa mediante serie di Fourier, cfr. ad esempio il Fast Poisson Solver.

Vediamo un esempio di filtraggio di un segnale di elettro-cardiogramma: J.11

Vediamo un esempio di confronto tra regolarizzazione e filtraggio: J.10

7.5 Rappresentazione dei Sistemi DLTI mediante DFT

7.5.1 Risposta in Frequenza di un Sistema DLTI

Ora, è interessante vedere cosa succede se un segnale del tipo:

$$u(k) = e^{i\omega k}, \quad -\infty < k < \infty \quad (7.30)$$

viene applicato in ingresso ad un sistema DLTI. In uscita, si ha:

$$y(k) = \sum_{j=-\infty}^{\infty} h(j) \cdot e^{i\omega(k-j)} = e^{i\omega k} \cdot \sum_{j=-\infty}^{\infty} h(j) \cdot e^{-i\omega j} = u(k) \cdot H(\omega) \quad (7.31)$$

ovvero viene riproposto il segnale di ingresso $u(k)$, moltiplicato per il fattore complesso $H(\omega)$. La funzione di ω a valori complessi, $H(\omega)$, viene chiamata **risposta in frequenza** del sistema DLTI:

$$H(\omega) = \sum_{j=-\infty}^{\infty} h(j) \cdot e^{-i\omega j} \quad (7.32)$$

Quanto si è detto è facilmente estendibile a sequenze di ingresso a valori reali, dato che una sinusoide può essere espressa come combinazione lineare di esponenziali complessi (7.2). Precisamente, dalla (7.31) segue che la risposta a $u_1(k) = \frac{A}{2} e^{j\phi} e^{j\omega_v k}$ è $y_1(k) = H(\omega_v) \frac{A}{2} e^{j\phi} e^{j\omega_v k}$; se $h(k)$ è reale, dalla (2.14) si ricava che la risposta a $\frac{A}{2} e^{-j\phi} e^{-j\omega_v k}$ è la complessa coniugata della risposta a $\frac{A}{2} e^{j\phi} e^{j\omega_v k}$, e quindi in risposta ad

$$u(k) = A \cdot \cos(\omega_v k + \phi) = \frac{A}{2} e^{j\phi} e^{j\omega_v k} + \frac{A}{2} e^{-j\phi} e^{-j\omega_v k}$$

, si ha:

$$y(k) = \frac{A}{2} \left[H(\omega_v) e^{j\phi} e^{j\omega_v k} + H(-\omega_v) e^{-j\phi} e^{-j\omega_v k} \right]$$

ovvero

$$y(k) = A |H(\omega_v)| \cos (\omega_v k + \phi + \theta) \quad (7.33)$$

dove $\theta = \arg [H(\omega_v)]$ indica lo sfasamento che il sistema introduce alla frequenza ω_v .

La risposta in frequenza è periodica con periodo 2π . Infatti, una sequenza di ingresso avente frequenza $\omega + 2j\pi$ coincide con la sequenza di frequenza ω . In definitiva, possiamo vedere la risposta in frequenza come serie di Fourier in cui i coefficienti sono i campioni della risposta all'impulso $h(k)$, calcolabili da $H(\omega)$ nel modo seguente:

$$h(k) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H(\omega) e^{i\omega k} d\omega \quad (7.34)$$

Per linearità, si ha:

$$Y(\omega) = H(\omega) \cdot U(\omega) \quad (7.35)$$

e quindi la convoluzione nel dominio del tempo diventa un semplice prodotto nel dominio delle frequenze. Questa ed altre proprietà hanno determinato la fortuna della trasformata di Fourier discreta (DFT) nell'analisi dei segnali e dei sistemi DLTI.

Si è detto in precedenza che la sequenza di autocorrelazione e lo spettro di potenza consentono di descrivere l'elaborazione operata da un sistema DLTI su sequenze ad energia infinita, mediante l'effetto che quel sistema opera sulla sequenza di autocovarianza delle sequenze stesse. Infatti, l'autocorrelazione dell'uscita di un sistema lineare è la convoluzione dell'autocorrelazione dell'ingresso con l'autocorrelazione della risposta all'impulso del sistema. Per lo spettro di potenza, vale:

$$P_{yy}(\omega) = |H(\omega)|^2 P_{uu}(\omega)$$

Vediamo alcuni esempi fondamentali di risposta in frequenza: [J.1](#)

L'analisi di Fourier per i sistemi DLTI è rappresentata dalla risposta in frequenza. Nelle applicazioni di tipo fisico-matematico, assume particolare importanza lo studio dei picchi nel diagramma dei moduli, o risonanze; ecco alcuni esempi pratici:

- vetri della casa che vibrano a causa di un camion in strada con il motore al minimo (l'edificio ha risonanze a bassa frequenza);
- zavorra di cemento all'interno della lavatrice, sopra il cesto: durante l'accelerazione della centrifuga vengono eccitate tutte le frequenze, comprese quindi le risonanze;
- orecchio umano: sveglie suonano ad 1kHz (è la frequenza di risonanza principale del nostro sistema uditivo);

- soppressione attiva del rumore generato dalle vibrazioni alla frequenza di risonanza (aggancio di fase e generazione di segnale in controfase): é presente in alcuni modelli di automobile;
- stabilizzazione di una petroliera nel mare del nord contro fenomeni ondosi del second'ordine che eccitano la nave alla frequenza di risonanza.
- effetto distruttivo della risonanza: il ponte di Tacoma (<http://youtu.be/3mclp9QmCGs>).

In ingegneria meccanica c'è un'intera disciplina dedicata all'analisi di Fourier del comportamento dinamico dei sistemi: la meccanica delle vibrazioni.

7.5.2 Trasformata z e Funzione di Trasferimento

Analogamente a quanto avviene con la trasformata di Laplace per i sistemi a tempo continuo, descritti da equazioni differenziali, la trasformata Zeta può descrivere sistemi a tempo discreto, descritti da equazioni alle differenze. Essa rende semplice operare con segnali e sistemi a tempo discreto, in quanto ne descrive la funzione di trasferimento mediante polinomi nella variabile complessa z . Di fatto trasforma equazioni alle differenze in equazioni algebriche. Per un paio di semplici esempi, si veda [7].

In questo modo rende possibile un approccio parametrico all'analisi in frequenza dei sistemi DLTI. Infatti, i sistemi DLTI possono essere descritti in modo **non-parametrico** mediante i valori della risposta al campione unitario, e la loro azione sul segnale in ingresso di conseguenza caratterizzata mediante la convoluzione (oppure, equivalentemente, nel dominio delle frequenze mediante i valori della trasformata di Fourier discreta della risposta al campione unitario, e la loro azione sul segnale in ingresso di conseguenza caratterizzata mediante il prodotto (7.35)). Essi possono essere descritti in modo **parametrico** mediante i coefficienti dell'equazione alle differenze, che sono gli stessi della Funzione di Trasferimento.

7.6 Calcolo della DFT: algoritmo della Fast Fourier Transform (FFT)

Quanto visto nella sezione precedente, ci fa capire che se esiste un algoritmo veloce per calcolare la trasformata discreta di Fourier, esso diventa uno strumento molto potente e di uso generale. Questo algoritmo esiste, ed é l'algoritmo della **Fast Fourier Transform (FFT)**.

Per il calcolo della trasformata (7.16), si nota che esso richiede N^2 moltiplicazioni ed $N \cdot (N - 1)$ addizioni complesse. Questo costo sarebbe impraticabile in molte situazioni pratiche. Il problema é risolto dall'algoritmo della *Fast Fourier*

Transform, che richiede solo $(N/2) \cdot \log_2 N$ moltiplicazioni complesse e $N \cdot \log_2 N$ addizioni complesse.

Ad esempio, nel caso $N = 1024$, che può corrispondere ad un caso bidimensionale di soli 32×32 punti, si ha un fattore di risparmio pari a 100 !

7.6.1 Algoritmo della FFT per segnali mono-dimensionali

Per semplificare la notazione, poniamo $W_N = e^{-j(2\pi/N)}$, da cui ad es. $W_N^{vk} = e^{-j(2\pi v/N)k}$. In questo modo, la trasformata di Fourier discreta (7.21) diventa:

$$U(\nu) = \frac{1}{N} \sum_{k=0}^{N-1} u(k) W_N^{\nu k} \quad (7.36)$$

Gli algoritmi veloci per il calcolo della DFT, che vanno sotto il nome di FFT (*Fast Fourier Transform*), si basano sul principio di calcolare la DFT di una sequenza lunga N dal calcolo di DFT di sotto-sequenze più corte. Applicando ricorsivamente questo principio (cosiddetto del *divide-et-impera*) finché la sottosequenza raggiunge la dimensione $N_s = 2$, la FFT richiede un numero di operazioni in virgola mobile proporzionale a $N \cdot \log N$.

Sceglieremo come esempio rappresentativo di algoritmo FFT un elemento della classe di algoritmi a *decimazione nel tempo*, in cui cioè viene suddivisa la sequenza di campioni nel dominio del tempo (esiste anche la classe di algoritmi a *decimazione in frequenza*).

L'algoritmo sfrutta sia la simmetria che la periodicità dell'esponenziale discreto W_N^{vk} :

- $W_N^{\nu(N-k)} = \overline{W_N^{\nu k}}$ (simmetria)
 - $W_N^{\nu k} = W_N^{\nu(k+N)} = W_N^{(\nu+N)k}$ (periodicità)

Il funzionamento ottimale degli algoritmi FFT si ha quando N è una potenza intera di 2: $N = 2^q$. Infatti, in questo caso possiamo suddividere per due le sottosequenze fino ad arrivare esattamente a sotto-sequenze di lunghezza 2, che vengono calcolate esplicitamente, come vedremo.

Supponiamo dunque di suddividere la sequenza di dati lunga N in due sottosequenze lunghe $N/2$, di cui la prima contiene i dati di indice pari e la seconda quelli di indice dispari. Otteniamo:

$$U(v) = \frac{1}{N} \sum_{k \text{ pari}} u(k) W_N^{vk} + \frac{1}{N} \sum_{k \text{ dispari}} u(k) W_N^{vk} \quad (7.37)$$

ovvero, con la sostituzione $k = 2r$ quando k é pari ed $k = 2r + 1$ quando k é dispari :

$$\begin{aligned} U(\nu) &= \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r) W_N^{2r\nu} + \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r+1) W_N^{(2r+1)\nu} \\ &= \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r) (W_N^2)^{r\nu} + W_N^\nu \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r+1) (W_N^2)^{r\nu} \end{aligned}$$

Ora notiamo che vale:

$$W_N^2 = e^{-2j(2\pi/N)} = e^{-j2\pi/(N/2)} = W_{N/2}$$

e dunque:

$$\begin{aligned} U(\nu) &= \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r) (W_{N/2})^{r\nu} + W_N^\nu \frac{1}{N} \sum_{r=0}^{(N/2)-1} u(2r+1) (W_{N/2})^{r\nu} \\ &= \frac{1}{2} G(\nu) + W_N^\nu \frac{1}{2} H(\nu) \end{aligned}$$

dove $G(\nu)$ e $H(\nu)$ sono due DFT di lunghezza $N/2$. Grazie alla loro periodicitá, anche se ν varia da 0 a $N - 1$ é sufficiente calcolare $G(\nu)$ ed $H(\nu)$ solo per ν tra 0 e $N/2 - 1$.

Dunque, se il calcolo della DFT di una sequenza lunga N richiede N^2 moltiplicazioni e addizioni complesse, aver riorganizzato i calcoli in due DFT di lunghezza $N/2$, la cui ri-combinazione richiede N moltiplicazioni ed addizioni complesse, richiede in totale $2N + 2(N/2)^2 = 2N + N^2/2$ addizioni e moltiplicazioni complesse.

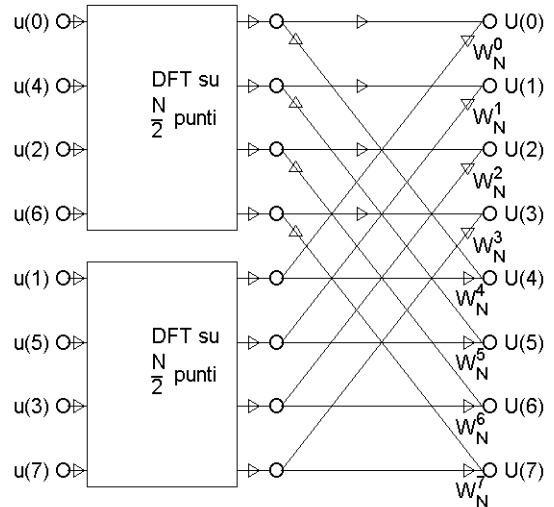
Proseguendo in questo modo per $\log_2 N$ stadi di suddivisione dei calcoli, é facile concludere che il numero complessivo di addizioni e moltiplicazioni complesse é $\mathcal{O}(N \log_2 N)$.

E' da sottolineare la particolare struttura che assume il calcolo della DFT di una sequenza lunga 2:tale struttura viene detta *a farfalla* ed é una struttura che si ripete in tutti gli stadi del calcolo dell'algoritmo. Vedremo nelle sotto-sezioni che seguono alcune proprietá interessanti legate a questa struttura: l'eliminazione di calcoli ridondanti, la realizzazione *sul posto* dei calcoli e la gestione degli indici. Per vedere la struttura a farfalla é necessario rappresentare l'algoritmo mediante i *grafi di flusso di segnale* [21]. Sul sito web del libro sono riprodotti. Per ulteriori approfondimenti sul calcolo della FFT, vedere Capitolo 6 di [21].

Vediamo in Figura 7.1 una rappresentazione mediante *grafo di flusso di segnale* dell'equazione del Par. 5.3.1, che qui riportiamo:

$$U(\nu) = G(\nu) + W_N^\nu \cdot H(\nu) \quad (7.38)$$

e che rappresenta l'applicazione al calcolo della DFT del concetto di *divide-et-impera*, ovvero la DFT di una sequenza lunga N puó essere convenientemente ottenuta mediante il calcolo di due DFT di lunghezza $N/2$ piú una ricombinazione.

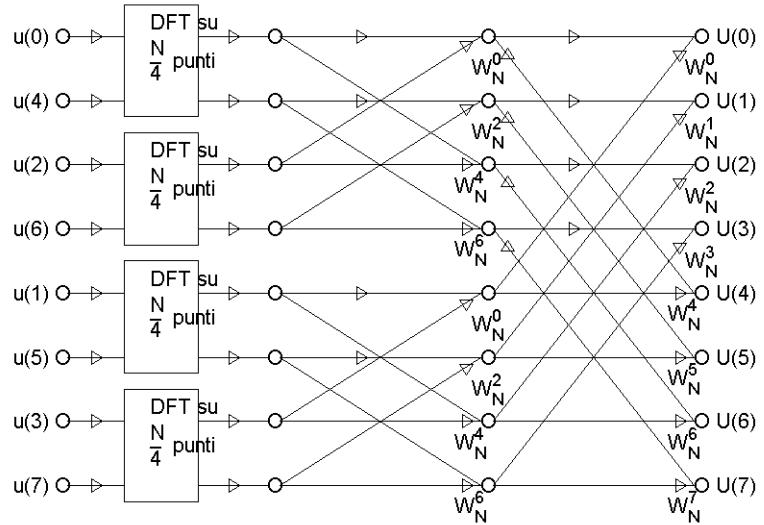


Applicazione del concetto di *divide-et-impera*

Nelle Figure 7.2 e 7.3 vediamo l'applicazione ricorsiva del divide-et-impera, in cui ogni livello di ricorsione aggiuntivo genera un corrispondente livello (o *stadio*) di calcolo. Se N é una potenza di due, si arriva ad un certo punto (qui $N = 8$ e quindi ci arriviamo al terzo stadio) a dover eseguire DFT di due soli campioni.

Quando la DFT é di due soli campioni, il procedimento di divisione del problema si arresta e possiamo allora considerare il calcolo vero e proprio. Qui, l'osservazione fondamentale é che tutto il grafo é riconducibile ad un insieme di operazioni fondamentali che elaborano due campioni di ingresso dello stadio e forniscono due valori in uscita. Tali operazioni fondamentali sono dette *farfalle* e sono del tipo di Figura 7.4:

Dalla Figura 7.4 si osserva che i due pesi complessi W dipendono da un solo parametro s e sono traslati uno rispetto all'altro di $W_N^{N/2} = -1$. Quindi, é possibile calcolare una *farfalla* con una sola moltiplicazione e non due, come mostrato in Figura 7.5, riducendo quindi il costo computazionale di quasi la metá.



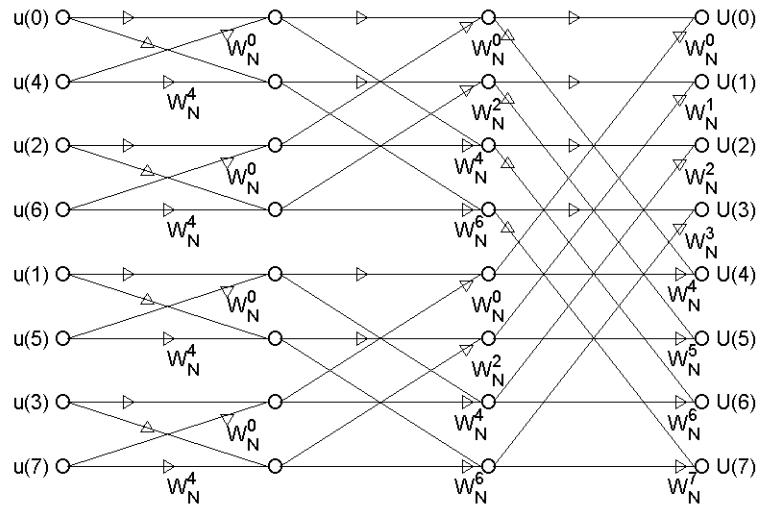
Lo schema di calcolo definitivo diventa dunque quello di Figura 7.6.
L'implementazione è contenuta nel file `algo_fft_1D.py`.

7.6.1.1 Una semplice riduzione del numero di operazioni

Basta osservare che:

$$W_N^{N/2} = e^{-j(2\pi/N) \cdot N/2} = e^{-j\pi} = -1$$

per rendersi conto che il calcolo della farfalla può essere semplificato in modo da richiedere una moltiplicazione in meno.

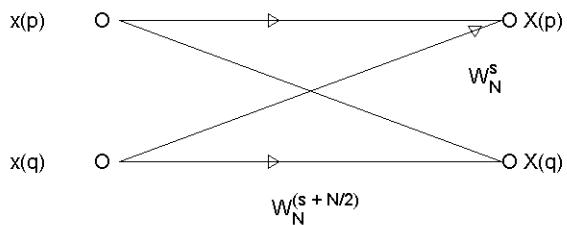


7.6.1.2 Il calcolo sul posto

Prendiamo in considerazione il grafo che descrive il calcolo della FFT. Le cose da tenere presente sono le connessioni (dette *rami*) tra i nodi ed i pesi (o coefficienti di trasmissione) associati a ciascun ramo. Scambiando di posizione i nodi si ottengono algoritmi differenti ma che producono lo stesso risultato, se i pesi e le connessioni rimangono inalterati. A ciascuna disposizione dei nodi possono però corrispondere proprietà differenti dell'algoritmo e produrre risultati differenti nell'efficienza dell'utilizzo della memoria e/o della velocità di esecuzione nel calcolatore.

Ad esempio, ad ogni stadio di calcolo, una sequenza di dati lunga N viene trasformata in una sequenza lunga N . Viene spontaneo allocare due vettori di lunghezza N , uno per la sequenza in ingresso ed uno per la sequenza in uscita.

Se riconosciamo che ogni stadio di calcolo è costituito da $N/2$ farfalle, allora pos-

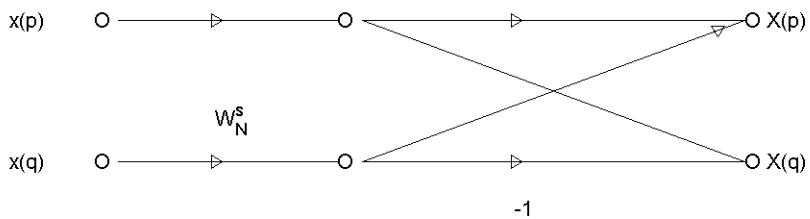


siamo concludere che in realtà è necessario un solo vettore di lunghezza N , in quanto i nuovi valori calcolati possono sostituire quelli vecchi senza che questi ultimi siano più necessari. Questo procedimento di calcolo viene chiamato *sul posto* e porta ad un utilizzo più efficiente della memoria del calcolatore.

Il fatto di poter eseguire l'algoritmo FFT sul posto, dipende dal fatto che tutte le farfalle abbiano nodi di ingresso e di uscita orizzontalmente adiacenti. Si può dimostrare che per poter effettuare il calcolo della FFT sul posto è necessario che la sequenza di ingresso o quella di uscita siano disposti secondo l'ordinamento *a disposizione invertita dei bit*.

7.6.1.3 La gestione degli indici

Gli algoritmi FFT che effettuano i calcoli sul posto, che sono anche quelli di uso più comune, necessitano di avere i dati, all'ingresso o in uscita, disposti secondo

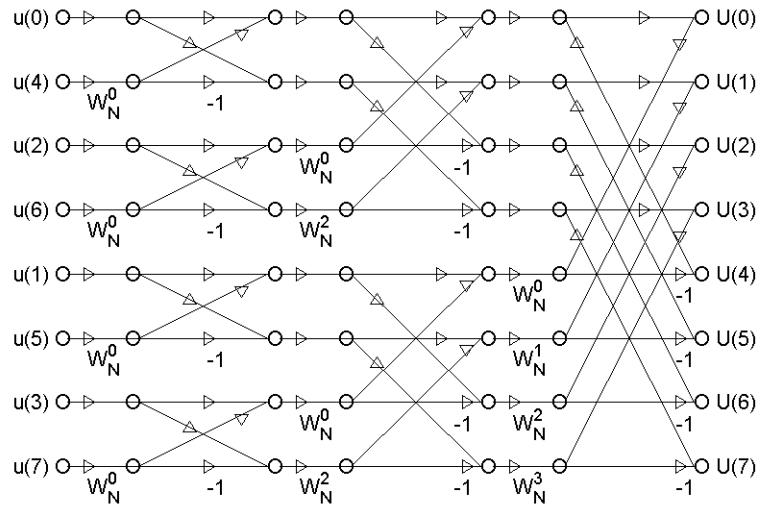


l'ordinamento *a disposizione invertita dei bit*.

Supponiamo di avere una sequenza di dati memorizzata per indici progressivi, allora è possibile riordinare la sequenza in ordine a disposizione invertita dei bit facendo calcoli *sul posto*.

L'algoritmo è il seguente: consideriamo una sequenza $u(k)$, $k = 0, \dots, N - 1$ e facciamo variare un secondo indice j da 0 a $N - 1$ in ordine a disposizione invertita dei bit. Ogniqualvolta si ha che $j > k$, si scambiano $u(k)$ e $u(j)$.

Vediamo un'implementazione Python dell'algoritmo FFT: [J.2](#)



7.7 Utilizzo in pratica della DFT

7.7.1 Rappresentazione grafica della DFT

Come già osservato in precedenza, la sequenza $U(\nu)$ prodotta dalla DFT assume valori nel campo complesso. Per una sua rappresentazione grafica esistono dunque varie opzioni. Una possibilità è graficarne la parte reale $\Re(U(\nu))$ e quella immaginaria $\Im(U(\nu))$ in funzione della frequenza, in due grafici separati.

Molto usata è la rappresentazione in **modulo** e **fase**:

$$\|U(\nu)\| = \sqrt{\Re(U(\nu))^2 + \Im(U(\nu))^2} \quad (7.39)$$

$$\phi_U(\nu) = \tan^{-1} \left(\frac{\Re(U(\nu))}{\Im(U(\nu))} \right) \quad (7.40)$$

in quanto rappresentano, ad esempio, l'amplificazione e lo sfasamento introdotti dalla risposta in frequenza di un sistema DLTI, cfr. (7.34).

E' importante notare che se la sequenza ha valori reali, come spesso avviene, la DFT é **simmetrica** attorno a $\omega_v = \pi$, che é la frequenza di Nyquist.

7.7.2 Il problema del *leakage* e la finestratura dei dati

Consideriamo il caso di una sequenza di dati di durata limitata, corrispondente ad una sinusoide per un numero non intero di periodi. In questo caso, entrambi i tipi di prolungamento indicati al paragrafo 7.4.3 producono una discontinuità nella sequenza, di tipo "salto".

Quindi, la nostra sequenza di dati viene vista, nel calcolo della DFT (mediante FFT), come una sinusoide piú un *salto*. Poiche' la DFT é un operatore lineare, i due spettri corrispondenti si sommano.

La sinusoide ha uno spettro formato da una riga alla frequenza della sinusoide. Il *salto* é prodotto dall'aver sommato alla sinusoide un segnale **discontinuo**, che ha in generale uno spettro (cioe' un contenuto di frequenze) ampio (cioe' non concentrato in un'unica frequenza, come la sinusoide), per cui la sua presenza produce uno spettro complessivo come quello dato dalla sinusoide piú un certo numero di componenti non nulle attorno ad essa ed in particolare una componente continua (che e' sostanzialmente la media della sequenza, che non e' piu' nulla come sarebbe per la sinusoide pura). Notare che i segnali canonici presentati in precedenza (delta di Kronecker e gradino unitario di Heavyside) sono discontinui.

Con la finestratura dei dati si elimina il salto e dunque, al prezzo di una distorsione della sequenza, é possibile effettuare un prolungamento periodico e continuo della sequenza e la stima dello spettro migliora sensibilmente.

Vediamo sperimentalmente: J.3

Come possiamo determinare la finestra ottimale ?

Per fare questo, dobbiamo affrontare il problema in modo piú generale e quindi, ad esempio, riconoscere ([6] pp. 69-72) che una generica sequenza finita di dati $h(k)$ puó essere considerata come una sequenza infinita $h_{inf}(k)$ che viene moltiplicata per una funzione *finestra* a supporto compatto e rettangolare, $w_{rett}(k)$:

$$h(k) = h_{inf}(k) \cdot w_{rett}(k) \quad (7.41)$$

La trasformata di Fourier (continua) della funzione *finestra rettangolare* é il *sinc*: $sinc(x) = \sin(x)/x$ per $x \neq 0$, e $sinc(0) = 1$. Quindi, $W_{rett}(\omega) = \text{sinc}(\omega)$.

Ora, moltiplicazione nel tempo significa convoluzione in frequenza:

$$H(\omega) = \frac{1}{2\pi} \int_{-\pi}^{\pi} H_{inf}(\theta) W_{rett}(\omega - \theta) d\theta \quad (7.42)$$

La finestratura rettangolare genera una dispersione (leakage) dello spettro. Il rimedio al problema del leakage è dunque scegliere finestre con valori dello spettro che decadono più rapidamente possibile. La finestra di Hamming ha uno spettro che decade sensibilmente di più di quello della finestra rettangolare.

Vediamo sperimentalmente: [J.4](#)

7.7.3 il problema dell'effetto *picket fence* e la tecnica di *zero-padding*

Il campionamento in frequenza produce un effetto sali-scendi (chiamato l'effetto dello steccato di paletti, o *picket-fence*).

Se il segnale originario contiene frequenze ai soli valori discreti calcolati dalla DFT non ci sono problemi, altrimenti c'è un effetto di attenuazione, che è massimo a metà tra due lobi consecutivi.

Questo effetto può essere ridotto aumentando il numero di *righe spettrali* a disposizione, e quindi aumentando N . Questo può essere fatto in due modi:

- aumentando la lunghezza dell'intervallo temporale di acquisizione dei campioni, T , oppure
- inserendo zeri in coda alla sequenza, tecnica cosiddetta dello *zero-padding*, avendo prima finestrato i dati in modo da portare l'ultimo campione ad un valore prossimo a zero (altrimenti la discontinuità introduce una componente spuria alla frequenza zero).

7.7.3.1 Risoluzione in frequenza

La *risoluzione in frequenza* è sostanzialmente il reciproco della distanza Δf (con f frequenza nel continuo) tra due righe contigue dello spettro discreto:

$$\Delta f = \frac{1}{T_c \cdot N} \quad (7.43)$$

e quindi più piccolo è Δf e migliore è la risoluzione in frequenza e viceversa. Esso è un parametro importante nella scelta del periodo di campionamento T_c e della lunghezza dell'intervallo temporale di acquisizione dei campioni, T , da cui deriva il numero stesso di campioni, $N = T/T_c$.

Notare che:

- la diminuzione del periodo di campionamento T_c , a parità di numero di campioni N , porta ad una diminuzione della risoluzione in frequenza (infatti, si arrivano a riconoscere frequenze nel continuo di valore più elevato, dato che aumenta la frequenza di Nyquist, però ho sempre N frequenze nello spettro discreto);

- l'aumento della lunghezza dell'intervallo temporale di acquisizione dei campioni T , a parità di periodo di campionamento T_c , porta ad un aumento della risoluzione in frequenza (basta vederlo nell'intervallo $[0, \Delta f]$);
- la tecnica di *zero-padding* **non modifica** la reale risoluzione in frequenza (i valori alle nuove righe spettrali sono sostanzialmente interpolazioni di quelli precedenti).

Vediamo sperimentalmente: [J.5](#)

7.7.4 Presenza di rumore nei dati

Un approccio semplificato al trattamento dei dati in presenza di rumore additivo a media nulla può essere quello, nel caso della DFT, di effettuare una media degli spettri corrispondenti a più sottoperiodi. La media tende in questo caso ad attenuare sostanzialmente il rumore.

Vediamo sperimentalmente: [J.6](#)

7.8 Stima dello Spettro di Potenza di un Segnale

In generale, la stima dello spettro di potenza ricade nell'ambito della *teoria della stima*. In pratica però, le tecniche di stima più attendibili, come quella basata sulla *verosimiglianza*, richiedono una quantità di informazione sul segnale maggiore di quella di solito disponibile. Per questo motivo, le tecniche di stima dello spettro di potenza solitamente usate hanno una notevole base empirica.

Consideriamo una sequenza di dati ad energia infinita e supponiamo di poterla pensare come realizzazione di un processo ergodico, cioè di un processo aleatorio in cui le medie temporali coincidono con le medie d'insieme.

Per questa sequenza di dati esiste la possibilità di costruire uno stimatore dell'autocovarianza e quindi di ottenere una sequenza che approssima la sequenza di autocovarianza del processo aleatorio. Per quanto detto al paragrafo [7.4.4](#), si potrebbe pensare di ottenere una stima consistente dello spettro di potenza semplicemente facendo la trasformata di Fourier discreta della sequenza di stima dell'autocovarianza. Questo non è vero in realtà perché si ha che la varianza della stima dello spettro non tende a zero al crescere della lunghezza della sequenza. È possibile però ottenere una buona stima con degli accorgimenti: la trasformata di Fourier viene *smussata*. Inoltre, l'algoritmo per il calcolo della stima può essere realizzato mediante FFT.

Per semplicità consideriamo che il processo sia a media nulla, per cui auto-correlazione ed auto-covarianza coincidono. Consideriamo la DFT della sequenza

di auto-correlazione r_{uu} :

$$I_N(\omega) = \sum_{m=-(N-1)}^{N-1} r_{uu}(m) e^{-j\omega m}$$

e consideriamo la DFT della sequenza $u(n)$:

$$U(\omega) = \sum_{n=0}^{N-1} u(n) e^{-j\omega n}$$

Si puó dimostrare che vale:

$$|I_N(\omega)| \approx \frac{1}{N} |U(\omega)|^2 \quad (7.44)$$

La stima di $I_N(\omega)$ é chiamata *periodogramma*. L'interesse per il periodogramma é giustificato dal fatto che lo si puó calcolare basandosi sulla FFT. Vediamo ora come ottenere delle stime consistenti dello spettro di potenza a partire dal periodogramma.

Il metodo piú intuitivo per ridurre la varianza delle stime é quello di fare la media di piú stime indipendenti. L'applicazione di questo metodo alla stima di spettri é attribuita a Bartlett. Seguendo questo approccio, la sequenza $u(n)$, $0 \leq n \leq N - 1$, viene suddivisa in K sotto-sequenze $u^{(i)}(l)$ di L campioni ciascuna, da cui $L \cdot K = N$. Si calcolano i k periodogrammi:

$$I_L^{(i)}(\omega) = \frac{1}{L} \left| \sum_{l=0}^{L-1} u(l)^{(i)} e^{-j\omega l} \right|^2$$

Notare che se $r_{uu}(m)$ é sufficientemente piccolo per $m > L$ allora é lecito ritenere che i periodogrammi $I_L^{(i)}(\omega)$ siano tra loro indipendenti. La stima dello spettro diventa:

$$\hat{P}_{uu}(\omega) = \frac{1}{K} \sum_{i=1}^K I_L^{(i)}(\omega) \quad (7.45)$$

Questo stimatore di Bartlett risulta essere polarizzato ed avere varianza inversamente proporzionale al numero di periodogrammi mediati.

Definizioni:

- la *polarizzazione (bias)* di uno stimatore é definita come la differenza tra il valore vero ed il valore atteso della stima;
- uno stimatore é detto *consistente* se la polarizzazione e la varianza delle stime tendono contemporaneamente a zero all'aumentare della lunghezza N della sequenza.

È da notare che nel metodo di Bartlett è necessario raggiungere un compromesso, che varierà a seconda dell'applicazione, tra polarizzazione (ovvero risoluzione dello spettro) e varianza della stima.

Il metodo di Welch, assai utilizzato, ottiene una riduzione della varianza della stima applicando una *finestra* alle sotto-sequenze (ed in questo modo smussando lo spettro). Si calcolano i periodogrammi modificati:

$$J_L^{(i)}(\omega) = \frac{1}{L \cdot S} \left| \sum_{l=0}^{L-1} u^{(i)}(l) w(l) e^{-j\omega l} \right|^2$$

dove

$$S = \frac{1}{L} \sum_{l=0}^{L-1} w^2(l)$$

e la stima dello spettro diventa:

$$\hat{P}_{uu}^{Welch}(\omega) = \frac{1}{K} \sum_{i=1}^K J_L^{(i)}(\omega) \quad (7.46)$$

Ora, la FFT diventa uno strumento efficiente per il calcolo della stima di uno spettro a frequenze equispaziate, cioè:

$$\hat{P}_{uu}^{Welch}(\omega_v) = \frac{1}{K} \sum_{i=1}^K J_L^{(i)}(\omega_v), \quad v = 0, \dots, L-1 \quad (7.47)$$

con $\omega_v = 2\pi v/L$. La procedura di calcolo è la seguente. Si calcola:

$$U_M^{(i)}(v) = \sum_{l=0}^{L-1} u^{(i)}(l) w(l) e^{-j\omega_v l}, \quad v = 0, \dots, L-1$$

mediante un algoritmo FFT, e si ricavano le quantità $|U_M^{(i)}(v)|^2$, che poi vengono sommate l'una all'altra per $i = 0, \dots, K$ ed il risultato finale viene diviso per $K \cdot L \cdot S$.

Per ulteriori approfondimenti, cfr. [21] Capitolo 11 e [26] Sezione 2.7 .

Il metodo di Welch è un classico metodo non-parametrico di stima dello spettro di potenza. Possono essere adottati a questo scopo anche metodi parametrici [6].

Vediamo degli esempi: [J.8](#)

7.8.1 Test di bianchezza di Bartlett

...

Vediamo degli esempi: [J.9](#)

7.9 La DFT e l'algoritmo FFT per segnali bi-dimensionali

La rappresentazione di sequenze bi-dimensionalili nel dominio delle frequenze, mediante la DFT, riveste una certa importanza nelle applicazioni, ad esempio nell'elaborazione delle immagini e dei dati sismici.

La generalizzazione alle sequenze bi-dimensionalili di quanto visto in questo Capitolo per le sequenze mono-dimensionalili è abbastanza immediata. Cominciamo con il definire una sequenza periodica bi-dimensionale:

$$u(m, n) = u(m + qM, n + rN)$$

dove q ed r sono numeri interi arbitrari positivi o negativi. La rappresentazione di tali sequenze mediante trasformata di Fourier discreta, diventa:

$$u(m, n) = \frac{1}{MN} \sum_{\nu=0}^{M-1} \sum_{\mu=0}^{N-1} U(\nu, \mu) W_M^{-\nu m} W_N^{-\mu n}$$

dove

$$U(\nu, \mu) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} u(m, n) W_M^{\nu m} W_N^{\mu n}$$

Si può verificare che:

$$U(\nu, \mu) = U(\nu + qM, \mu + rN)$$

per valori interi di q ed r e dunque $U(\nu, \mu)$ e $u(m, n)$ hanno la stessa periodicità.

Il discorso fatto per la Trasformata di Fourier Discreta riguardo ad una sequenza di lunghezza finita, si applica in modo analogo ad una sequenza bi-dimensionale di area finita nel piano (m, n) . Si ha:

$$U(\nu, \mu) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} u(m, n) W_M^{\nu m} W_N^{\mu n}, \quad \nu = 0, \dots, M-1 \quad \mu = 0, \dots, N-1 \quad (7.48)$$

Per il calcolo della (7.48), possiamo osservare che esso comporta M DFT del tipo:

$$A(m, \mu) = \sum_{n=0}^{N-1} u(m, n) W_N^{\mu n}$$

seguite da N DFT del tipo:

$$U(\nu, \mu) = \sum_{m=0}^{M-1} A(m, \mu) W_M^{\nu m}$$

e quindi puó essere eseguito dagli algoritmi FFT per sequenze monodimensionali. Se M ed N sono potenze di 2, saranno necessarie

$$\frac{NM}{2} \log_2(NM)$$

moltiplicazioni ed addizioni complesse, contro le $M^2 \cdot N^2$ necessarie per il calcolo diretto della 7.48). Per informazioni ulteriori cfr. [21].

Vediamo un'implementazione Python: J.7

Capitolo 8

Analisi tempo-frequenza

Abbiamo visto come analizzare/sintetizzare un segnale discreto dalla sua Trasformata di Fourier, cfr. (7.22). In questo capitolo vediamo alcune tecniche di Calcolo per l'Analisi di segnali non-stazionari.

L'analisi mediante DFT é particolarmente significativa solo se siamo in regime stazionario. Quando invece una sequenza di dati cambia le sue proprietà spettrali al variare del tempo, l'analisi mediante DFT mostra dei limiti fondamentali: non distingue le variazioni nel tempo del contenuto spettrale e quindi non indica né gli istanti in cui tali variazioni sono avvenute né l'andamento di quest'ultime.

Vediamo alcuni esempi: K

8.1 Short-Time Fourier Transform (STFT)

E' possibile fare un'analisi tempo-frequenza seguendo l'approccio dell'analisi di Fourier, visto in precedenza.

L'idea di base é localizzare l'analisi nel tempo e lo strumento fondamentale é la finestratura. Utilizzando una funzione finestra $g(t)$ che pesi significativamente di piú alcuni campioni attorno al suo centro e traslandola con passo a lungo l'asse dei tempi, é possibile calcolare uno spettro che rappresenti maggiormente il contenuto di frequenze presente in questi campioni, e quindi in un intervallo temporale ristretto:

$$\hat{u}^{STFT}(f, \tau) = \int_{-\infty}^{\infty} u(t)g(t - \tau)e^{-i2\pi ft}dt \quad (8.1)$$

Nel discreto, detto $\tau_n = nT$ ed $f_v = \frac{\omega_v}{2\pi}$, diventa:

$$U^{STFT}(v, n) = \frac{1}{N} \sum_{k=0}^{N-1} u(k)g(k - n)e^{-i2\pi f_v k}dt \quad v, n = 0 \dots N - 1 \quad (8.2)$$

Notare che c'è un compromesso fondamentale (legato al principio di indeterminazione di Heisenberg) tra località dell'analisi nel tempo e risoluzione dell'analisi in frequenza (cfr. sezione 7.7.3.1). Lo si può vedere, ad esempio, dal punto di vista del riconoscimento delle basse frequenze: se la finestra è piccola, un segnale a bassa frequenza rimane quasi costante e quindi viene rappresentato alla frequenza zero.

Infatti, questa tecnica di localizzazione diminuisce di fatto il numero di campioni significativi nella sequenza finestrata, e dunque di righe spettrali calcolabili. Pertanto, ha senso definire un passo b di decimazione dello spettro, e calcolare solamente le N/b righe.

La necessità di definire dei passi a e b maggiori di uno è evidente se si pensa che per $a = b = 1$ da N dati si ottengono N^2 coefficienti di Fourier partendo da N campioni nel tempo, che equivale dunque ad una rappresentazione eccessivamente ridondante. Si calcola dunque, detto $u_{per}(k, n_a)$ la sequenza con periodo N/b ottenuta periodizzando¹ la sequenza finestrata $u(k)g(k - n_a)$:

$$U^{STFT}(\nu_b, n_a) = \frac{1}{N} \sum_{k=0}^{N-1} u_{per}(k, n_a) e^{-i2\pi f_{\nu_b} k} dt, \quad \nu_b = 0, b, 2b, \dots, N-b, \quad n_a = 0, a, 2a, \dots, N-a \quad (8.3)$$

Vediamo i dettagli nel seguente algoritmo (presente nel file *stft.m* del sito web):

```
function [stf] = stft(x,w,a,b)
    % x: sequenza di dati
    % w: funzione finestra
    % a: passo di tralazione temporale della finestra
    % b: passo di decimazione dello spettro
    if nargin < 3; a = 1; b = 1; end;    % cos'i ottengo N^2 coefficienti
    N = length(x);
    res = zeros(N/a,N/b);
    w1 = [w zeros(1,N-length(w))]; % finestra prolungata ad N campioni
    ww1 = [w1,w1];    % duplico la finestra per facilitare la traslazione
    for jj = 1 : N/a;
        y = x.* ww1( (N+1-(jj-1)*a) : (2*N - (jj-1)*a) ) ; % finestratura
        y1 = perbas(y,b); % periodizzare y => decimarne lo spettro, cfr. sez. 7.4.1
        v = fft(y1);    % calcolo lo spettro di N/b righe
        res(rem(jj-1+N/a/2,N/a)+1,:) = v;
    end;
    stf = res.';    % \e la matrice che contiene i coefficienti dello spettrogramma
```

Si aprono dunque alcuni problemi fondamentali, ovvero la determinazione:

- della forma ottimale della funzione finestra;

¹Ad esempio, con $N = 9$ e $b = 3$ periodizzando la sequenza $y = [1, 2, 3, 4, 5, 6, 7, 8, 9]$, di lunghezza N , si ottiene la sequenza $u_{per} = [1 + 4 + 7, 2 + 5 + 8, 3 + 6 + 9]$ di lunghezza N/b

- del passo b di decimazione dello spettro di frequenze;
- del passo a di traslazione della finestra nell'asse dei tempi.

A questi problemi ha dato delle risposte molto significative il matematico ungherese Gabor, con la sua teoria delle *Gabor frames*. In particolare, tale teoria dimostra che la forma ottimale della funzione finestra è una gaussiana.

Vediamo alcuni aspetti pratici: L

8.1.1 Esempio di spettrogramma: ricostruzione di una partitura musicale

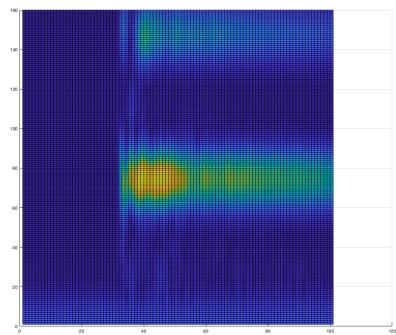
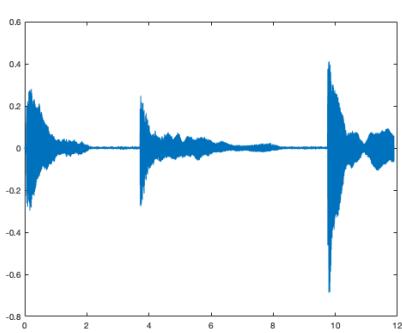
...

Vediamo sperimentalmente: M

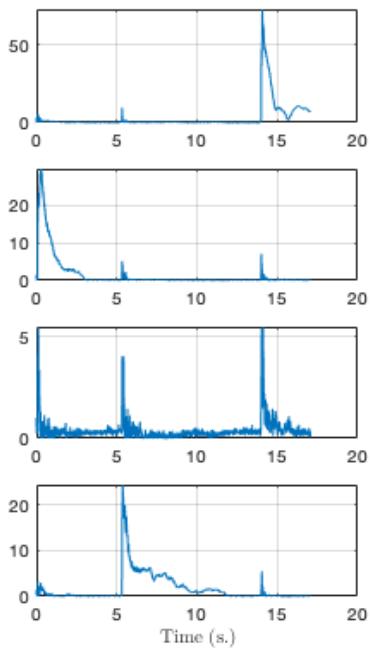
8.1.2 Separazione di sorgenti da uno spettrogramma

Spesso uno spettrogramma è una composizione di sorgenti differenti. Se pensiamo ad esempio a `pianoforte_3.wav` visto in laboratorio, ogni tasto del pianoforte è una sorgente differente. Vogliamo ora individuare l'azione di queste sorgenti e separare ogni azione dalle altre, analizzando lo spettrogramma. Nell'esempio citato ciò significa riconoscere le note suonate e l'istante in cui ciò avviene. Con le seguenti ipotesi, possiamo operare questa separazione di sorgenti mediante una NMF (nonnegative matrix factorization):

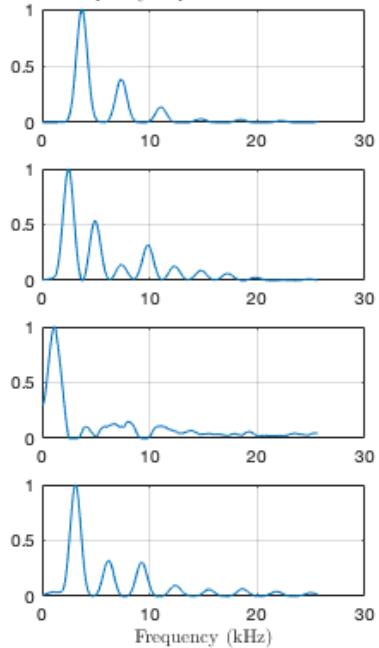
- lo spettrogramma è una combinazione lineare nonnegativa delle sorgenti. Nonnegativa significa che viene ignorata l'eventuale cancellazione di alcune componenti; lineare significa che effetti nonlineari, quali ad esempio la saturazione del microfono, vengono negletti.
- lo spettrogramma delle sorgenti ha rango basso. Effettivamente, il rango di una singola nota è pari a 1!

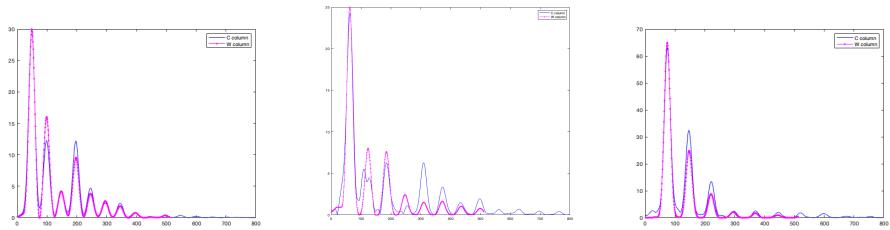


Activations of the sources



Frequency response of the sources





8.2 La Trasformata Wavelet

La trasformata wavelet é nata da un'esigenza fondamentale: molte applicazioni di analisi dei dati non possono permettersi di portare il sistema al regime stazionario e devono funzionare gi dopo un semplice transitorio iniziale. Non a caso le wavelets sono nate nell'ambito delle prospezioni petrolifere, nonostante l'approccio dal punto di vista matematico fosse gi stato studiato all'inizio del secolo scorso.

Un segnale discreto pu anche essere analizzato mediante la somma di sequenze contenenti dettagli a risoluzione via-via decrescente. La trasformazione inversa, sintesi, avviene mediante la somma di una sequenza a bassa risoluzione e di sequenze contenenti dettagli a risoluzione via-via crescente. Questo tipo di analisi prende il nome di *analisi multi-risoluzione (MRA)* ed é il cuore dell'analisi mediante *wavelets*.

Detta $\Psi(t)$ la *mother wavelet*, la trasformata wavelet é cos definita:

$$u^{wav}(s, \tau) = \frac{1}{\sqrt{s}} \int_{-\infty}^{\infty} u(t) \Psi\left(\frac{t - \tau}{s}\right) dt \quad (8.4)$$

Nel discreto, supponendo una sequenza lunga $N = s_0^p$, con p arbitrario, $t = k\tau_0$ e detto $\tau_{n,r} = n\tau_0 s_0^{r+p}$ ed $s_r = s_0^r$, diventa:

$$U^{wav}(r, n) = \frac{1}{\sqrt{s_0^r}} \sum_{k=0}^{N-1} u(k) \Psi\left(\left(\frac{k}{s_0^r} - ns_0^p\right)\tau_0\right), \quad (8.5)$$

dove $r = 0, -1, \dots, -p+1$, $n = 0, 1, 2, \dots, s_0^{-r}-1$.

Una scelta molto nota di mother wavelet é la derivata seconda di una gaussiana:

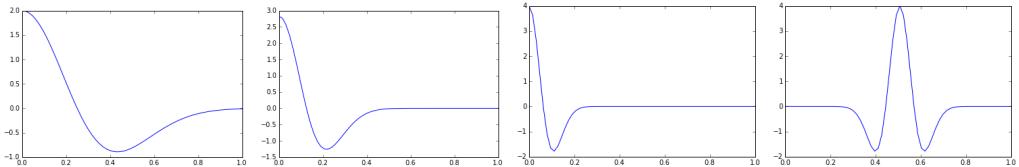
$$\Psi(t) = \left(1 - t^2\right) e^{-t^2/2} \quad (8.6)$$

da cui, scelto $s_0 = 2$, si hanno le $\Psi_{r,n} = \frac{1}{\sqrt{2^r}} \Psi\left(\frac{t}{2^r} - n\tau_0\right)$ come in Figura 8.4. Notare come le $\Psi_{r,n}$ siano ottenute dalla mother wavelet mediante

scalamento e traslazione:

$$\Psi_{r,n} = s_0^{-r/2} \Psi((s_0^{-r} k - n) \tau_0) \quad (8.7)$$

Vediamo l'andamento di alcune wavelets: 



da sinistra a destra ($p = 6$): $r = -2, n = 0$, $r = -3, n = 0$, $r = -4, n = 0$, $r = -4, n = 8$

Ora, il calcolo della trasformata wavelet avviene secondo l'approccio dell'*analisi multi-risoluzione* [9], intendendo con r l'indice del livello di risoluzione. La proiezione della sequenza di dati ad ogni livello di risoluzione avviene mediante prodotti interni con le *scaling functions*, ricavate dalle wavelets. Tale proiezione costituisce un'approssimazione di u al livello di risoluzione r , mentre i prodotti interni con le wavelets rappresentano i dettagli che si aggiungono passando al livello di risoluzione successivo $r + 1$.

8.2.1 Le wavelets di Haar

La mother wavelet di Haar é la seguente:

$$\Psi(t) = \begin{cases} 1 & 0 \leq t \leq 1/2 \\ -1 & 1/2 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (8.8)$$

e la scaling function:

$$\phi(t) = \begin{cases} 1 & 0 \leq t \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad (8.9)$$

Per motivi che saranno chiari nel seguito, useremo qui nel seguito $2^{-p/2} \Psi(t)$, come mother wavelet, e analogamente $2^{-p/2} \phi(t)$ come scaling function. Partendo dal livello di risoluzione piú alto, $r = -p + 1$, la *Trasformata di Haar* decomponne una sequenza $u(k)$ di lunghezza N in due sotto-sequenze di lunghezza $N/2$. La prima é una media mobile ed é detta *trend*, mentre la seconda é una differenza mobile ed é detta *fluttuazione*.

Tutte le trasformate wavelet operano in questo modo. Nel caso della trasformata di Haar, i coefficienti $a(m)$ della sotto-sequenza **trend** sono:

$$a(m) = \frac{u(2m-1) + u(2m)}{\sqrt{2}} \quad m = 1, \dots, N/2 \quad (8.10)$$

mentre quelli $d(m)$ della sotto-sequenza **fluttuazione** sono:

$$d(m) = \frac{u(2m-1) - u(2m)}{\sqrt{2}} \quad m = 1, \dots, N/2 \quad (8.11)$$

Proprietá fondamentale di queste trasformate é di avere *fluttuazioni piccole*: la magnitudo dei campioni della *fluttuazione* é spesso significativamente inferiore di quelli dei campioni della sequenza originale:

- la ragione di questo fatto é legata al fatto che i campioni della sequenza vengono solitamente prodotti mediante campionamento di una funzione continua, con un passo di campionamento solitamente molto breve.
- un'applicazione significativa di questa proprietá é la compressione del segnale.

Definiamo ora le *wavelets di Haar* di livello 1, $\{W_i^1\}$: esse sono $N/2$ sequenze di lunghezza pari a N cosí definite:

$$\begin{aligned} W_1^1 &= (\frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, 0, \dots, 0) \\ W_2^1 &= (0, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}, 0, 0, \dots, 0) \\ &\vdots \\ W_{N/2}^1 &= (0, 0, \dots, 0, \frac{1}{\sqrt{2}}, \frac{-1}{\sqrt{2}}) \end{aligned} \quad (8.12)$$

da cui é immediato osservare le seguenti proprietá:

- energia = 1;
- presenta una rapida fluttuazione tra due valori contigui ed ha media nulla;
- sono una la traslazione dell'altra.

La sotto-sequenza *fluttuazione* é esprimibile mediante prodotti scalari tra la sequenza originale e ciascuna sequenza wavelet. Stessa cosa vale per la sequenza *trend* e le *scaling signals di Haar* di livello 1, cosí definite:

$$\begin{aligned} V_1^1 &= (\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, \dots, 0) \\ V_2^1 &= (0, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}, 0, 0, \dots, 0) \\ &\vdots \\ V_{N/2}^1 &= (0, 0, \dots, 0, \frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}) \end{aligned} \quad (8.13)$$

Ora, ai livelli successivi della trasformata si applicano le formule (8.10) e (8.11) solo alla sotto-sequenza *trend* del livello precedente a quello in considerazione.

Vediamo un esempio: **N**

Vediamo sperimentalmente la multi-risoluzione: P

Notare che in questo modo si possono definire nuove trasformate wavelet semplicemente definendo nuove sequenze wavelet e scaling functions, ovvero la distinzione tra trasformate wavelet differenti sta in questo tipo di scelta.

Definizione di *supporto di un segnale discreto* $u(k)$: è l'insieme di indici $\{k\}$ per cui $u(k) \neq 0$.

Utilizzando questa definizione, la proprietà delle piccole fluttuazioni diventa, nel caso delle wavelets di Haar, la seguente: se il segnale discreto $u(k)$ è pressoché costante in corrispondenza al supporto della wavelet di livello 1, W_i^1 , allora il coefficiente $d^1(i)$ della *fluttuazione* è pressoché nullo.

8.2.2 Le wavelets della Daubechies [9]

Vediamo il caso più semplice, *Daub4*. Definiamo i seguenti *scaling numbers*:

$$\alpha_1 = \frac{1 + \sqrt{3}}{4\sqrt{2}}, \quad \alpha_2 = \frac{3 + \sqrt{3}}{4\sqrt{2}}, \quad \alpha_3 = \frac{3 - \sqrt{3}}{4\sqrt{2}}, \quad \alpha_4 = \frac{1 - \sqrt{3}}{4\sqrt{2}}$$

ed i seguenti *scaling signals* di livello 1:

$$V_1^1 = (\alpha_1, \alpha_2, \alpha_3, \alpha_4, 0, 0, \dots, 0) \quad (8.14)$$

$$V_2^1 = (0, 0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, 0, 0, \dots, 0) \quad (8.15)$$

$$\vdots \quad (8.16)$$

$$V_{N/2}^1 = (\alpha_3, \alpha_4, 0, 0, \dots, 0, \alpha_1, \alpha_2) \quad (8.17)$$

$$(8.18)$$

Definiamo ora i seguenti *wavelet numbers*:

$$\beta_1 = \frac{1 - \sqrt{3}}{4\sqrt{2}}, \quad \beta_2 = \frac{\sqrt{3} - 3}{4\sqrt{2}}, \quad \beta_3 = \frac{3 + \sqrt{3}}{4\sqrt{2}}, \quad \beta_4 = \frac{-1 - \sqrt{3}}{4\sqrt{2}}$$

e le seguenti *wavelets Daub4* di livello 1:

$$W_1^1 = (\beta_1, \beta_2, \beta_3, \beta_4, 0, 0, \dots, 0) \quad (8.19)$$

$$W_2^1 = (0, 0, \beta_1, \beta_2, \beta_3, \beta_4, 0, 0, \dots, 0) \quad (8.20)$$

$$\vdots \quad (8.21)$$

$$W_{N/2}^1 = (\beta_3, \beta_4, 0, 0, \dots, 0, \beta_1, \beta_2) \quad (8.22)$$

$$(8.23)$$

A questo punto si costruisce la trasformata wavelet mediante prodotti scalari della sequenza originaria con gli scaling signals e con le wavelets.

Per le wavelets Daub4 la proprietá delle piccole fluttuazioni diventa la seguente: se il segnale discreto $u(k)$ é pressoché lineare in corrispondenza al supporto della wavelet di livello 1, W_i^1 , allora il coefficiente $d^1(i)$ della *fluttuazione* é pressoché nullo.

In generale, per le wavelets $DaubJ$ della Daubechies la proprietá delle piccole fluttuazioni diventa: se il segnale discreto $u(k)$ é pressoché uguale ad un polinomio di grado $< J/2$ in corrispondenza al supporto della wavelet di livello q , W_i^q , allora il coefficiente $d^q(i)$ della *fluttuazione* é pressoché nullo.

Vediamo un confronto tra wavelets di Haar e della Daubechies: [Q](#)

8.2.3 Le *Coiflets*

Un'altra famiglia di wavelets é stata suggerita da Coifman ed é detta *coiflets*.

La trasformata associata a queste wavelets produce delle sequenze *trend* che aderiscono meglio alla sequenza di dati originaria di quanto faccia qualunque trasformata con wavelets di Daubechies.

Vediamo un esempio di compressione di segnali audio: [R](#)

8.2.4 Conservazione e compattamento dell'energia

Brevemente:

- il primo livello della trasformata di Haar conserva l'energia del segnale originario (e dunque ciò viene mantenuto ai livelli successivi);
- l'energia della sotto-sequenza *trend* di livello 1 corrisponde ad una grossa percentuale dell'energia del segnale originario.

8.2.5 Interpretazione della trasformata wavelet come banco di filtri [27]

Per semplicitá, prendiamo in considerazione la trasformata di Haar. In essa, la sotto-sequenza *trend* $a(m)$ viene ottenuta dalla sequenza originaria $u(k)$ mediante la:

$$y(k) = \frac{1}{\sqrt{2}} \cdot u(k) + \frac{1}{\sqrt{2}} \cdot u(k-1) = b_0 \cdot u(k) + b_1 \cdot u(k-1)$$

da cui

$$a(m) = y(2 \cdot m), \quad m = 1, \dots, N/2$$

e dunque come risposta di un sistema DLTI avente come risposta al campione unitario lo *scaling signal* stesso (come è facile verificare):

$$h(0) = b_0 \quad (8.24)$$

$$h(1) = b_1 \quad (8.25)$$

$$\vdots \quad (8.26)$$

$$h(n_b) = b_{n_b} \quad (8.27)$$

e successiva decimazione (*down-sampling*) ogni 2 campioni .

Analogamente avviene per la sotto-sequenza *fluttuazione* e la sequenza *wavelet*.

In definitiva, la trasformata wavelet agisce sulla sequenza originaria come un banco di filtri DLTI, cioè produce due sotto-sequenze che sono uscite di altrettanti sistemi DLTI.

Notare che la trasformata inversa utilizza gli stessi filtri ma con i coefficienti b presi in ordine invertito. Lo si può verificare notando che scaling-signals e wavelets sono una base ortogonale (le sequenze sono tra loro a due a due ortogonal) e dunque la matrice $T = [V; W]$ è ortogonale. Ne segue che la trasformata inversa è definita dalla matrice trasposta T^T , da cui si verifica che la trasposizione corrisponde all'inversione dell'ordine dei coefficienti. Per una presentazione più dettagliata dell'argomento rinviamo a [27].

8.2.6 Analisi in frequenza delle wavelets

Abbiamo mostrato in precedenza come la DFT metta in evidenza le singole componenti in frequenza di un segnale discreto, ed in particolare ne identifichi le frequenze e le contribuzioni relative in ampiezza.

Esaminando il contenuto in frequenza degli *scaling signals* e delle *wavelets* possiamo dire, in generale, che:

- il contenuto in frequenza degli *scaling signals* è concentrato alle basse frequenze e va diminuendo fino ad annullarsi alle alte frequenze; esso ha dunque un comportamento passa-basso nei confronti della sequenza originaria, per la formazione della sotto-sequenza *trend*;
- il contenuto in frequenza delle *wavelets* è concentrato alle alte frequenze e va diminuendo fino ad annullarsi alle basse frequenze; esso ha dunque un comportamento passa-alto nei confronti della sequenza originaria, per la formazione della sotto-sequenza *fluttuazione*;
- il contenuto in frequenza dei due segnali è dunque complementare; di fatto, per gli scaling signals e le wavelets di livello 1 la somma degli spettri risulta una costante pari a 2.

8.2.7 Wavelets 2-d

...

Vediamo un esempio: **S**

Vediamo un esempio di MRI: **T**

Capitolo 9

Identification of linear systems

In questo capitolo consideriamo modelli lineari statici e sistemi DLTI.

9.1 Identificazione di modelli lineari statici

Scegliere la struttura di un modello lineare statico, ad es. il modello lineare standard (3.2), significa determinare il numero n di variabili esplicative, detto anche *ordine* del modello. Dopodiché, il numero di parametri da stimare risulta anch'esso pari ad n .

9.1.1 Stima dei parametri del modello lineare standard

Richiamiamo quanto visto al par. 3.1.1.

Consideriamo il modello lineare standard (3.2). L'errore residuo tra la predizione del modello, $A x$, e i dati osservati b , detto *errore di predizione*, è pari a:

$$A x - b = \epsilon \tag{9.1}$$

e quindi a valori casuali e media nulla e deviazione standard pari a σ . Dunque il residuo in generale non si annulla nemmeno in presenza del valore vero del vettore di parametri x . Non è quindi ragionevole pensare che il vettore dei parametri x possa essere calcolato prendendo un numero di dati sperimentali uguale al numero di parametri e risolvendo esattamente il sistema lineare a matrice quadrata $A \cdot x = b$, come faremmo invece ad esempio in un problema di interpolazione, in cui i dati sono supposti esatti, cioè in cui $\epsilon = 0$. Dato che i dati osservati sono in generale affetti da rumore, è ragionevole pensare di ridurre l'effetto del rumore considerando un numero sostanzialmente maggiore di dati osservati rispetto al numero di parametri del modello matematico.

Se pensiamo che ad ogni dato osservato corrisponde un'equazione, in cui le variabili sono i parametri del modello matematico, questo corrisponde a dover risolvere un sistema lineare *sovra-determinato*, $A \cdot x = b$, che quindi ha più equazioni (vincoli) che variabili (gradi di libertà) e dunque, in generale, può non avere soluzione.

Piú precisamente, la soluzione del sistema sovra-determinato esiste se il membro destro b appartiene all'immagine di A , cioè $b \in \text{Im}(A)$, altrimenti è comunque possibile cercare quel vettore \hat{x} che minimizza il residuo in norma due:

$$\|b - A x\|_2 \quad (9.2)$$

e cioè la soluzione al problema dei *minimi quadrati lineari*: data una matrice $A \in \mathbb{R}^{m \times n}$, $m > n$ (spesso $m \gg n$), e dato un vettore $b \in \mathbb{R}^m$, si cerca un vettore $\hat{x} \in \mathbb{R}^n$ tale che risolva:

$$\hat{x} = \underset{x}{\operatorname{argmin}} \|A x - b\|_2^2 \quad (9.3)$$

Se A ha rango pieno (ovviamente per colonne), allora la soluzione del problema (9.3) esiste ed è unica. Il caso di A con rango non-pieno è trattato nel paragrafo 4.2.1.

Questo modo di procedere può essere spesso motivato anche da ragioni statistiche, e da ragioni computazionali. Infatti, esistono ormai algoritmi agevoli ed accurati per il calcolo numerico della soluzione del problema (9.3), cfr. paragrafo 3.3.

Esempio: fitting di un insieme di N dati sperimentali, presi alle ascisse $\{\xi_i\}$, mediante un polinomio di secondo grado, avente coefficienti $x = \{x_1, x_2, x_3\}$. Il modello (deterministico) è:

$$x_1 + x_2 \xi_i + x_3 \xi_i^2 = b_i$$

problema dei minimi quadrati: $\hat{x} = \underset{x}{\operatorname{argmin}} \sum_{i=1}^N [b_i - (x_1 + x_2 \xi_i + x_3 \xi_i^2)]^2$

ovvero, in termini matriciali: $b = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} \quad x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad A = \begin{bmatrix} 1 & \xi_1 & \xi_1^2 \\ \vdots & \vdots & \vdots \\ 1 & \xi_N & \xi_N^2 \end{bmatrix}$

e $\hat{x} = \underset{x}{\operatorname{argmin}} \|b - Ax\|_2^2$.

9.1.2 Determinazione dell'ordine del modello dai dati sperimentali

La scelta delle variabili esplicative in un modello lineare, e quindi anche del loro numero, che rappresenta la complessità o *ordine* del modello lineare, è in genere

dettata da fattori legati all'applicazione specifica. Il procedimento di identificazione del modello puó peró fornire degli strumenti generali per verificare se tale scelta risulti essere adeguata a far sí che il modello descriva bene i dati sperimentali utilizzati, oltreché per stimare i parametri del modello.

Riprendiamo il concetto di errore riducibile ed errore irriducibile del par. 4.4. Quando il modello non é noto e va identificato, operare su un insieme di dati affetto da rumore introduce il problema dell'*overfitting*: si puó arrivare a complicare il modello per descrivere il rumore presente nei dati, tentando quindi di ridurre l'errore irriducibile (!). Non é solo inutilmente costoso, ma anche dannoso: la varianza dell'errore di stima di tutti i parametri aumenta.

9.1.2.1 Il caso del modello lineare (regressione lineare)

Le quantitá fondamentali ed i criteri da adottare per validare un modello lineare statico, sono i seguenti:

- **parsimonia**: il modello deve avere il numero di parametri minimo possibile e l'aggiunta di un parametro deve corrispondere ad un calo significativo del residuo (9.2), che é la deviazione standard dell'errore di predizione (9.1). La stima della varianza dell'errore di predizione σ^2 , é: $\hat{\sigma}^2 = \frac{\sum_i(b_i - A\hat{x})^2}{m-n}$. E' un concetto analogo a quello di bias-vs-varianza del par. 4.4.
- **significativitá dei parametri**: ogni parametro deve essere significativamente non nullo; la significativitá viene in genere misurata in senso statistico, tramite la costruzione di un *test di ipotesi*;
- **varianza delle stime dei parametri**: $\sigma^2(A^T A)^{-1}$; bisogna considerare quanto l'ingresso di ogni nuovo parametro faccia aumentare la varianza delle stime dei parametri e, se l'aumento é eccessivo, non considerare tale parametro;

Vediamo un esempio sperimentale: U

Per i modelli lineari statici non esistono *indici di parsimonia* che rivelino automaticamente l'ordine del modello, come é invece per i sistemi DLTI, cfr. par. 9.3.2.2. Esiste una vasta letteratura su come utilizzare le quantitá fondamentali ed i criteri ora elencati, ed altri ancora, cfr. [2].

Per quanto riguarda la selezione delle variabili da inserire nel modello, cosa che puó essere fatta anche a priori, é da sottolineare il concetto di *shrinkage* [28], che equivale a selezionare a posteriori le variabili tramite un termine aggiuntivo al criterio di identificazione del modello, che va a mettere a zero i coefficienti delle variabili risultanti meno significative; un metodo molto noto é il Lasso [28]. Notare le analogie con i metodi di regolarizzazione.

9.1.3 Stima dei parametri del modello lineare generalizzato

Consideriamo il modello lineare generalizzato (par. 3.1.2). Se W é definita positiva e $W = BB^T$ é la sua fattorizzazione di Cholesky allora il problema diventa trovare:

$$\hat{x} = \operatorname{argmin}_x \|B^{-1}(b - Ax)\|_2 = \operatorname{argmin}_x (b - Ax)^T W^{-1} (b - Ax)$$

che puó essere risolto come un problema lineare standard, considerando $\bar{A} = B^{-1}A$ e $\bar{b} = B^{-1}b$.

Il caso particolare in cui W é una matrice diagonale positiva viene detto dei *mínimi quadrati pesati*. Spesso W é una matrice di covarianza di stima dei parametri, vedere ad esempio il Filtro di Kalman 10.2.

9.1.4 Stima dei parametri di modelli lineari flessibili

Nelle situazioni in cui il modello lineare non descrive adeguatamente bene il comportamento del sistema/fenomeno da modellare, una buona stima dei parametri non puó comunque essere risolutiva. Si puó pensare in tali casi ad introdurre flessibilitá nel modello, effettuando una stima dei parametri che non imponga in maniera forte la struttura del modello, bensí in maniera debole o, appunto, *flessibile*. Un metodo molto usato é quello di Hodrick-Prescott, il quale invece di richiedere che, come in (9.2):

$$\hat{b}' = \operatorname{argmin}_{b'} \|b' - b\|_2^2 \quad \text{soggetto al vincolo: } b' \in \operatorname{Range}(A) \quad (9.4)$$

richiede che:

$$\hat{x}^{hp} = \operatorname{argmin}_x \|x - b\|_2^2 \quad \text{soggetto al vincolo: } Dx = 0 \quad (9.5)$$

dove D é una matrice tri-diagonale tale che il generico vincolo risulti: $x_{i-1} + 2x_i + x_{i+1} = 0$. In questo metodo i vincoli non vengono applicati in maniera forte, altrimenti ci sarebbero solo soluzioni esattamente lineari, bensí trasformando il vincolo in una funzione di *penalizzazione*:

$$\hat{x}^{hp} = \operatorname{argmin}_x \frac{1}{2} \sum_{i=1}^N (b_i - x_i)^2 + \lambda \sum_{i=2}^{N-1} (x_{i-1} + 2x_i + x_{i+1})^2 \quad (9.6)$$

ovvero

$$\hat{x}^{hp} = \operatorname{argmin}_x \|x - b\|_2^2 + \lambda \|Dx\|_2^2 \quad (9.7)$$

dove D é la matrice delle differenze finite del second'ordine:

$$D = \begin{bmatrix} -1 & 1 & & & \\ 1 & -2 & 1 & & \\ & \ddots & \ddots & \ddots & \\ & & 1 & -2 & 1 \\ & & & 1 & -1 \end{bmatrix} \quad (9.8)$$

la cui soluzione é:

$$\hat{x}^{hp} = \left(I + 2\lambda D^T D \right)^{-1} b \quad (9.9)$$

Quindi i parametri non sono piú i coefficienti del modello ma i singoli campioni. In sostanza il modello viene applicato imponendo che l'andamento della soluzione sia localmente lineare. Questo impone un certo grado di regolaritá alla soluzione, ovvero il metodo agisce sui dati da *smoother*.

Notare che la flessibilitá rende però il modello piú difficilmente interpretabile; si parla infatti di trade-off tra flessibilitá ed interpretabilitá del modello [28]. Qui, la scelta molto dipende dallo scopo per cui si sta costruendo il modello: un modello flessibile porta ad una descrizione piú accurata del fenomeno (facendo ovviamente attenzione all'over-fitting !) e migliora quindi potenzialmente le capacità predittive del modello, mentre un modello piú restrittivo migliora l'inferenza, ad esempio i parametri possono avere un preciso significato fisico.

Vediamo un esempio sperimentale: [U.1](#)

9.1.5 Misurare la bontá delle stime

Come misuriamo la bontá del modello stimato? Possiamo calcolare l'MSE su N dati utilizzati per la stima dei parametri del modello (ovvero per il *training* del modello):

$$MSE_{training} = \frac{1}{N} \sum_{n=1}^N (b_i - A_{i,:} x_\delta)^2 \quad (9.10)$$

come stima di $E \{b - Ax_\delta\}$, che é un valore che vale in generale per il fenomeno osservato e non solo per l'insieme di dati considerato durante il training del modello.

C'è una difficoltà fondamentale in questo procedimento: assunto che il criterio siano i minimi quadrati (MSE), possiamo facilmente misurare il $MSE_{training}$ ma in realtà vogliamo il modello che renda minimo il MSE_{test} , ovvero che generalizzi a tutti i dati che possono essere generati dal sistema/fenomeno modellato. Dall'analisi congiunta di queste due quantitá possiamo capire se siamo in over-fitting.

Vediamo due metodi di stima del MSE_{test} basati sul ricampionamento (*resampling methods*): una parte dell'insieme di dati viene esclusa dal procedimento di fitting del modello, ed usata successivamente per testare il modello su nuovi dati. Consideriamo:

1. *il metodo dell'insieme di validazione (validation set)*: si divide semplicemente l'insieme di dati in due parti, la prima da utilizzare per il training e la seconda per il test;

2. la *validazione incrociata* (*cross-validation*): si fanno piú training, ognuno dei quali esclude un piccolo insieme di dati (al limite uno solo), calcolando ogni volta il MSE_{test} sui dati esclusi; la stima del MSE_{test} globale é la media di quelli calcolati nei singoli training. Il modello stimato é la media dei modelli di training.

Un vantaggio principale del secondo metodo rispetto al primo é di avere meno *bias* in quanto ad ogni training vengono utilizzati molti piú dati dell'insieme di partenza. Lo svantaggio di dover fare molti training puó essere annullato in alcune situazioni di rilevanza pratica (vedi esempi).

Relativamente al secondo metodo, si tratta di creare k sottoinsiemi da escludere individualmente ad ogni training. Al limite $k = N - 1$, quando escludiamo un solo dato alla volta. Per alti valori di k si ha meno bias ma piú varianza, quindi esiste un valore di k da ottimale scegliendo in base alla scomposizione bias-varianza, analogamente a (4.4).

9.1.6 Stima di parametri tempo-varianti

Esistono molte situazioni in cui é necessario elaborare una stima dei parametri prima di avere raccolto l'intera sequenza di dati, e poi di aggiornare la stima ad ogni nuovo dato ricevuto. Se utilizziamo le tecniche viste fin qui per il calcolo della soluzione ai minimi quadrati, ogni volta che vogliamo aggiornare la stima ci troviamo a dover ripetere daccapo tutti i calcoli, con evidente spreco di risorse di calcolo.

Per evitare questo spreco sono disponibili algoritmi (detti *minimi quadrati ricorsivi*) che, supposto di conoscere la soluzione con $N - 1$ dati, la riutilizzano per calcolare quella relativa a N dati con un costo di calcolo ottimale.

Consideriamo il seguente sistema sovra-determinato, $A_w^{(N)} \cdot x = b_w^{(N)}$, da risolvere ai minimi quadrati pesati :

$$\begin{bmatrix} w_0 \cdot a_0^T \\ w_1 \cdot a_1^T \\ \vdots \\ w_N \cdot a_N^T \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} w_0 \cdot b_0 \\ w_1 \cdot b_1 \\ \vdots \\ w_N \cdot b_N \end{bmatrix} \quad (9.11)$$

dove a_i^T é la i -esima riga della matrice di (9.17) e b_i é la i -esima componente del suo membro destro, e $w_i = \lambda^{N-i}$, $|\lambda| < 1$.

Questo modo usuale di pesare le osservazioni passate con i pesi w_i rende le osservazioni recenti piú importanti rispetto a quelle acquisite nel passato. Per questo λ é comunemente detto *fattore di oblio* (*forgetting factor*) e rappresenta di fatto la lunghezza della memoria con la quale opera l'algoritmo di stima dei parametri.

All'arrivo di un nuovo dato, per aggiornare le stime dei parametri, e dunque la soluzione del problema ai minimi quadrati, é possibile utilizzare la forma ricorsiva della fattorizzazione QR, cfr. sec. 1.3.4 . Notare che l'aggiornamento della (1.17) mediante trasformazioni di Givens é l'unica operazione necessaria, in quanto per risolvere i minimi quadrati é sufficiente aver applicato tutte le trasformazioni ortogonali anche al membro destro, e non é invece necessario formare esplicitamente la matrice Q .

9.2 Identificazione di sistemi DLTI da dati privi di rumore

Scegliere la struttura di un modello DLTI significa principalmente determinare l'ordine n del modello. Dopodiché, ci sono alcune differenze a seconda che si cerchi una rappresentazione ARMA (2.15) o state-space (2.26)-(2.23) del modello. Nel caso ARMA risulta $n = n_a$ ed é necessario determinare anche n_b , dopodiché il numero di parametri da stimare risulta, come vedremo nel seguito, pari a $n_a + n_b$. Nel caso state-space l'ordine n é pari al numero di variabili di stato ed il numero di parametri da stimare dipende, come vedremo nel seguito, dalla parametrizzazione scelta per le matrici A, B, C, D e puó essere, al piú, pari a $n^2 + n * n_u + n_y * n + n_y * n_u$, ovvero al numero di componenti di dette matrici.

Per costruire l'algoritmo di stima, operiamo in modo analogo a quanto fatto per il modello lineare standard (3.2) quando ne abbiamo stimato i coefficienti con i minimi quadrati (9.3), cioè cerchiamo quell'insieme di valori dei parametri che rende minimo l'errore di predizione in norma-2. Questo approccio fondamentale é chiamato *Prediction Error Method (PEM)* [18].

Nel caso di un sistema DLTI, questo significa porre

$$y(k) - \hat{y}(k) = 0 \quad , \quad k = 0, 1, 2, \dots, N \quad (9.12)$$

dove $y(k)$ e' il dato di indice k , $\hat{y}(k)$ e' la predizione (??) ottenuta in base ai dati relativi agli indici precedenti, e risolvere il sistema di equazioni lineari risultante nel senso dei minimi quadrati. Questo approccio é basato su una visione *ingresso-uscita* del sistema, ovvero non si tiene conto esplicitamente del vettore di stato.

Per i sistemi state-space (2.26)-(2.23) ha notevole importanza anche l'appuccio dei *metodi subspace* [31]: invece di trovare il minimo di un funzionale dell'errore di predizione (metodo PEM), i metodi subspace enfatizzano la presenza di un vettore di variabili di stato e procedono alla stima dei parametri del modello in due passi: calcolo della sequenza di stati dai dati ingresso-uscita e successiva determinazione delle matrici del sistema dagli stati stimati.

I metodi numerici utilizzati dagli algoritmi di *subspace identification* sono in particolare la fattorizzazione QR e la decomposizione in valori singolari (SVD), mentre il metodo PEM fa ricorso ai metodi numerici per minimi quadrati, lineari e nonlineari (cfr. par. ??).

Cominciamo con l'analizzare il caso deterministico, in cui cioè supponiamo che i dati siano stati generati da un modello ARX in cui $e(k) = 0$ per $k = 0, 1, 2, \dots$ (ovvero un modello ARMA deterministico), dopodiché alla sezione (9.3.1) vedremo l'effetto introdotto da una sequenza di rumore.

9.2.1 Caso di risposta all'impulso discreto e rappresentazione ARMA

Supponiamo di possedere $N \gg n_a$ campioni della *risposta all'impulso discreto* $h(k), k = 0, 1, 2, \dots, N - 1$. Per semplicità, poniamo $n_b = n_a$, da cui risulterà ovvio il caso $n_b < n_a$. Riscrivendo la (2.15) per ogni valore di k , sostituendo $h(k)$ ad $y(k)$, otteniamo il seguente sistema lineare (ovviamente sovra-determinato):

$$\left[\begin{array}{cccccc} 0 & \dots & \dots & 0 & h_0 \\ \vdots & \ddots & \ddots & h_0 & h_1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & h_0 & h_1 & \ddots & h_{n_a-1} \\ h_0 & h_1 & h_2 & \ddots & h_{n_a} \\ \hline h_1 & h_2 & h_3 & \ddots & h_{n_a+1} \\ h_2 & h_3 & h_4 & \ddots & h_{n_a+2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{N-n_a} & h_{N-n_a+1} & \dots & \dots & h_N \end{array} \right] \cdot \begin{bmatrix} a_{n_a} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n_b-1} \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (9.13)$$

che spezziamo nei due seguenti sistemi lineari, $B \cdot a = b$:

$$\left[\begin{array}{cccccc} 0 & \dots & \dots & 0 & h_0 \\ \vdots & \ddots & \ddots & h_0 & h_1 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & h_0 & h_1 & \ddots & h_{n_a-1} \\ h_0 & h_1 & h_2 & \ddots & h_{n_a} \end{array} \right] \cdot \begin{bmatrix} a_{n_a} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n_b-1} \\ b_{n_b} \end{bmatrix} \quad (9.14)$$

e $A \cdot a = 0$:

$$\left[\begin{array}{cccccc} h_1 & h_2 & h_3 & \ddots & h_{n_a+1} \\ h_2 & h_3 & h_4 & \ddots & h_{n_a+2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{N-n_a} & h_{N-n_a+1} & \dots & \dots & h_N \end{array} \right] \cdot \begin{bmatrix} a_{n_a} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \quad (9.15)$$

Vale il seguente

Theorem 9.2.1. : “ se n_a è l’ordine del sistema da cui è stata acquisita la risposta all’impulso discreto $h(k)$, la matrice (di Hankel):

$$\begin{bmatrix} h_1 & h_2 & h_3 & \ddots & h_k \\ h_2 & h_3 & h_4 & \ddots & h_{k+1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \end{bmatrix} \quad (9.16)$$

ha rango n_a per $k \geq n_a$ ”.

Quindi, se ne deduce che la matrice A della (9.15) non ha rango pieno. Allora, sfruttiamo il fatto che si sia posto $a_0 = 1$ ed il sistema (9.15), portando a membro destro l’ultima colonna di A , diventa:

$$\begin{bmatrix} h_1 & h_2 & h_3 & \ddots & h_{n_a} \\ h_2 & h_3 & h_4 & \ddots & h_{n_a+1} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{N-n_a} & h_{N-n_a+1} & \dots & \dots & h_{N-1} \end{bmatrix} \cdot \begin{bmatrix} a_{n_a} \\ \vdots \\ a_2 \\ a_1 \end{bmatrix} = \begin{bmatrix} -h_{n_a+1} \\ -h_{n_a+2} \\ \vdots \\ -h_N \end{bmatrix} \quad (9.17)$$

la cui matrice ha questa volta, sempre per il precedente teorema, rango pieno.

Allora, siamo in grado di formulare l’algoritmo di stima dei parametri: si risolve il sistema sovra-determinato (9.17), nel senso dei minimi quadrati, e si trovano i coefficienti $\{a_i\}$, dopodiché si trovano i coefficienti $\{b_i\}$ mediante il prodotto matrice-vettore (9.14).

Vediamo un esempio sperimentale: X

Vediamo un esempio nel caso di parametri tempo-varianti: W

9.2.2 Caso di risposta all’impulso discreto e rappresentazione *statespace*

Quello che abbiamo appena visto per i sistemi DLTI nella rappresentazione AR-MA lo possiamo ripetere anche per la rappresentazione nello spazio degli stati (*statespace*), data la stretta analogia tra le due rappresentazioni, vedi par. 2.3.4. Consideriamo qui quella classe di problemi inversi chiamata *problem di realizzazione*: da una sequenza di dati sperimentali che sappiamo corrispondere ai coefficienti di Markov del sistema (risposta all’impulso discreto), trovare la realizzazione A, B, C, D di questa sequenza. In particolare, ci interessa la realizzazione del sistema minimo (cfr. [10] per approfondimenti), e cioè che sia raggiungibile ed osservabile (cfr. Appendice A). Consideriamo inizialmente per semplicità il generico sistema DLTI ad un ingresso ed un’uscita (*single-input-single-output* o *SISO*), nella forma *state-space* (2.26)-(2.23):

$$\begin{aligned}x(k+1) &= A \cdot x(k) + b \cdot u(k) \\y(k) &= c \cdot x(k) + d \cdot u(k)\end{aligned}$$

E' facile verificare la relazione seguente tra la forma state-space e la risposta all'impulso:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 & \dots \\ h_2 & h_3 & h_4 & \ddots & \dots \\ h_3 & h_4 & \ddots & \ddots & \dots \\ h_4 & \ddots & \ddots & \ddots & \dots \\ \vdots & \vdots & \vdots & \vdots & \end{bmatrix} = \begin{bmatrix} c \\ cA \\ cA^2 \\ cA^3 \\ \vdots \end{bmatrix} \cdot [\begin{bmatrix} b & Ab & A^2b & A^3b & \dots \end{bmatrix}] \quad (9.18)$$

cioé la matrice di Hankel (infinita) H puó essere fattorizzata nel prodotto delle matrici di *osservabilitá* e *raggiungibilitá* del sistema, aventi rango n_x , dove n_x é la dimensione della piú piccola realizzazione del sistema, cioé la lunghezza del vettore di stato del sistema *minimo*, che é per definizione sia raggiungibile che osservabile.

Dunque, se il sistema é *minimo*, le due matrici R ed O hanno rango n_x , cosí come le matrici R_{n_x} ed O_{n_x} :

$$O_{n_x} = \begin{bmatrix} c \\ cA \\ cA^2 \\ cA^3 \\ \vdots \\ cA^{n_x-1} \end{bmatrix}, \quad R_{n_x} = [\begin{bmatrix} b & Ab & A^2b & A^3b & \dots & A^{n_x-1}b \end{bmatrix}] \quad (9.19)$$

Questo significa che il minore principale di ordine n_x della matrice di Hankel, é invertibile. In generale, per il rango della matrice H vale ancora il teorema 9.2.1.

Vediamo un esempio sperimentale: **V**

Nel caso MIMO, applicando un impulso unitario a ciascun ingresso si costruiscono per colonne i coefficienti di Markov (risposta all'impulso discreto), e da questi si stima la realizzazione minima mediante fattorizzazione QR o SVD della matrice di Hankel a blocchi, analogamente a quanto visto sopra per il caso SISO (per i dettagli si veda ad es. [10]).

9.2.3 Eccitazione persistente

Ci sono molte situazioni nella pratica in cui non é possibile imporre l'ingresso desiderato al sistema, però é comunque possibile osservarne sia l'ingresso che l'u-

scita. In tal caso, perché l'identificazione del sistema sia corretta, é necessario che la coppia di sequenze ingresso-uscita $\{u(k), y(k)\}$ sia *persistentemente eccitante*, ovvero soddisfi alla seguente definizione (intuitiva):

" una coppia di sequenze ingresso-uscita $\{u(k), y(k)\}$ é detta *persistentemente eccitante* per il modello in esame, di ordine n_a , se non potrebbe essere prodotta da un modello di ordine inferiore".

Una definizione rigorosa é in [18]: *la sequenza di ingresso $u(k)$ é persistentemente eccitante di ordine $2i$ se la matrice di covarianza dell'ingresso C_{uu} ha rango pieno, che é pari a $2i$ se $i = m$, con m pari al numero di ingressi (se $m > 1$ la sequenza di ingresso é ovviamente multi-variata).*

9.2.4 Caso di coppie ingresso/uscita qualsiasi e rappresentazione ARMA

Il metodo di stima risulta sostanzialmente simile a quello visto nel caso precedente, ma con una differenza fondamentale: mentre la coppia di sequenze $\{\delta(k), h(k)\}$, ovvero delta di Kronecker (2.12) e relativa risposta del sistema, determina univocamente il sistema DLTI (cfr. par 2.3.1), una generica coppia di sequenze $\{u(k), y(k)\}$ potrebbe in generale essere riprodotta anche da un sistema di complessità (*ordine*) inferiore a quella reale, ovvero puó essere non rappresentativa di tutte le dinamiche presenti nel sistema¹. La coppia di sequenze $\{u(k), y(k)\}$ deve quindi essere persistentemente eccitante (par. 9.2.3). Vale allora il seguente

Theorem 9.2.2. : " se n_a é l'ordine del sistema da cui é stata acquisita la coppia di sequenze ingresso-uscita e questa é persistentemente eccitante, la matrice :

$$\left[\begin{array}{cccccc|cccccc} 0 & \dots & \dots & 0 & u_0 & 0 & \dots & \dots & 0 & y_0 \\ \vdots & \ddots & \ddots & u_0 & u_1 & \vdots & \ddots & \ddots & y_0 & y_1 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & u_0 & u_1 & \ddots & u_{n_a-1} & 0 & y_0 & y_1 & \ddots & y_{n_a-1} \\ u_0 & u_1 & u_2 & \ddots & u_{n_a} & y_0 & y_1 & y_2 & \ddots & y_{n_a} \\ u_1 & u_2 & u_3 & \ddots & u_{n_a+1} & y_1 & y_2 & y_3 & \ddots & y_{n_a+1} \\ u_2 & u_3 & u_4 & \ddots & u_{n_a+2} & y_2 & y_3 & y_4 & \ddots & y_{n_a+2} \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ u_{N-1-n_a} & u_{N-1-n_a+1} & \dots & \dots & u_{N-1} & y_{N-1-n_a} & y_{N-1-n_a+1} & \dots & \dots & y_{N-1} \end{array} \right] \quad (9.20)$$

¹Un semplice esempio dalla meccanica pratica é il caso in cui si registri la posizione nel tempo di un sistema mantenuto a velocitá costante e della forza che lo mantiene in moto: essendo l'accelerazione nulla non vi é alcun effetto inerziale nel moto ed infatti la forza applicata serve solo a vincere l'attrito all'avanzamento; l'identificazione da questa coppia di sequenze porterebbe ad un modello mancante della parte inerziale, quindi sarebbe impossibile stimare le masse in gioco!

é singolare, mentre la matrice:

$$\left[\begin{array}{cccccc|cccccc} 0 & \dots & \dots & 0 & u_0 & 0 & \dots & \dots & 0 \\ \vdots & \ddots & \ddots & u_0 & u_1 & \vdots & \ddots & \ddots & y_0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & y_1 \\ 0 & u_0 & u_1 & \ddots & u_{n_b-1} & 0 & y_0 & y_1 & \vdots \\ u_0 & u_1 & u_2 & \ddots & u_{n_b} & y_0 & y_1 & \ddots & y_{n_a-1} \\ u_1 & u_2 & u_3 & \ddots & u_{n_b+1} & y_1 & y_2 & y_3 & \vdots \\ u_2 & u_3 & u_4 & \ddots & u_{n_b+2} & y_2 & y_3 & y_4 & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \vdots \\ u_{N-1-n_b} & u_{N-1-n_b+1} & \dots & \dots & u_{N-1} & y_{N-1-n_a} & y_{N-1-n_a+1} & \dots & y_{N-2} \end{array} \right] \quad (9.21)$$

ha rango pieno per colonne.

L'algoritmo di stima dei parametri, mediante risoluzione di un problema ai minimi quadrati, é analogo a precedente. Sul sito web c'è l'implementazione e qualche esempio.

9.2.5 Caso di coppie ingresso/uscita qualsiasi e rappresentazione *state-space*

In questo caso un approccio fondamentale é dato dai metodi subspace (che nel caso SISO si riducono a quanto visto nel paragrafo 9.2.2).

La teoria dei metodi subspace é piuttosto complessa ed articolata: l'approccio si sviluppa in due passi:

1. calcolo della sequenza di stati futuri e della matrice di osservabilità estesa dalla coppia di sequenze ingresso-uscita $\{u(k), y(k)\}$, mediante fattorizzazioni QR ed SVD;
2. da queste informazioni intermedie, calcolo dei coefficienti delle matrici A, B, C, D risolvendo un problema ai minimi quadrati lineari.

Rimandiamo a [31] oppure [16] per una descrizione dettagliata della teoria ed al sito web del presente libro per alcuni esempi significativi ed implementazioni degli algoritmi.

9.3 Identificazione di modelli DLTI da dati con rumore

9.3.1 Analisi dei valori singolari e presenza di rumore nei dati

Vediamo ora di riscrivere la (9.15) tenendo conto anche del termine random $e(k)$ presente nel modello ARX; si ha:

$$\begin{bmatrix} h_1 & h_2 & h_3 & \ddots & h_{n_a+1} \\ h_2 & h_3 & h_4 & \ddots & h_{n_a+2} \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ h_{N-n_a} & h_{N-n_a+1} & \dots & \dots & h_N \end{bmatrix} \cdot \begin{bmatrix} a_{n_a} \\ \vdots \\ a_1 \\ a_0 \end{bmatrix} = \begin{bmatrix} e_{n_a+1} \\ e_{n_a+2} \\ \vdots \\ e_N \end{bmatrix} \quad (9.22)$$

In questo caso, per la presenza del termine random $e(k)$, le colonne della matrice risultano sempre linearmente indipendenti e non c'è valore n_a per cui essa diventi singolare. In pratica, però, è sicuramente utile poter capire dai dati di che entità, valutata ad esempio in norma-2, deve essere ammesso questo errore random $e(k)$. L'analisi ai valori singolari di A ci aiuta proprio in questo.

Supponiamo dunque di non conoscere i valori del vettore dei parametri a (che è la tipica situazione all'inizio di un procedimento di identificazione) e consideriamo il caso normalizzato $\|a\|_2 = 1$. Ora, per definizione di SVD, si ha:

$$Av_{n_a+1} = U\Sigma V^T v_{n_a+1} = \sigma_{n_a+1} u_{n_a+1} \quad (9.23)$$

e cioè scegliendo $a = v_{n_a+1}$, vettore singolare destro (che quindi ha effettivamente norma-2 unitaria) corrispondente al più piccolo valore singolare σ_{n_a+1} della matrice A , si ottiene a membro destro $\sigma_{n_a+1} \cdot u_{n_a+1}$, che è il vettore di lunghezza minima (in norma-2) ottenibile quando si applica un vettore di norma-2 unitaria a destra di A .

Dunque σ_{n_a+1} è la deviazione standard del rumore quando il vettore di parametri ha lunghezza unitaria. Nel caso generale, in cui $\|a\|_2 \neq 1$ e vogliamo peraltro che sia $a_0 = 1$, la deviazione standard diventa $\sigma_{n_a+1} * |a_0| / \|a\|_2$.

9.3.2 Determinazione dell'ordine di un modello ARX

Come si può vedere dal passo 3 della procedura delineata all'inizio della Parte ??, insieme ai valori dei parametri è necessario determinare anche il loro numero, ovvero la complessità del modello da utilizzare. Questo aspetto, particolarmente delicato, non può essere svolto senza adeguata conoscenza del sistema che si intende modellare, e senza un corredo di tecniche per la valutazione della complessità riscontrabile direttamente sui dati acquisiti.

Di seguito vediamo due strumenti fondamentali: l'analisi dei valori singolari della matrice di Hankel ed i criteri di parsimonia.

9.3.2.1 L'analisi dei valori singolari

I valori singolari della matrice di Hankel danno un'indicazione chiara sull'ordine del sistema, come si può capire dai teoremi visti ai paragrafi 9.2.2 e 9.2.4.

Inoltre, è stato visto l'effetto del rumore presente nei dati sulla magnitudo del più piccolo valore singolare. In particolare, la presenza di rumore nei dati riduce di fatto la distanza tra il valore singolare più piccolo della matrice 9.16 nel caso $k = na$ e quello della stessa matrice nel caso $k = na + 1$, che in assenza di rumore dovrebbe essere riconosciuta come singolare, per riuscire a determinare correttamente l'ordine del sistema.

9.3.2.2 Criteri di parsimonia

Un buon modello identificato è quello che produce la minima varianza dell'errore di predizione sui dati prodotti dal sistema ed i cui parametri hanno la minima varianza d'errore. Variando il numero di parametri del modello, queste due esigenze sono tra di loro in conflitto: aumentando il numero di parametri diminuisce la varianza dell'errore di predizione ma aumenta in generale la varianza d'errore di stima di ogni singolo parametro.

Su questo spirito nascono i *criteri di parsimonia*, anche se la motivazione teorica ha origini decisamente più profonde. Vediamo ora due criteri di parsimonia comunemente utilizzati:

- AIC (Akaike's Information Criterion): $\log \left(V \cdot \left(1 + \frac{d \cdot 2}{N} \right) \right) \approx \log(V) + \frac{d \cdot 2}{N}$
- MDL (Minimum Description Length, dovuto a Rissanen): $V \cdot \left(1 + \frac{d \cdot \log(N)}{N} \right)$

dove :

- V viene detta *funzione di perdita*, ad es. $V = \frac{1}{N} \sum_{k=1}^N \left(y(k) - \hat{y}(k) \right)^2$;
- d è il numero totale di parametri del modello ;
- N è il numero di osservazioni utilizzate per stimare i parametri del modello.

In generale, aumentando il numero di parametri diminuisce il valore della funzione di perdita e viceversa. Il compromesso principale espresso dal criterio di parsimonia è di fatto tra queste due quantità.

Vediamo un esempio sperimentale, in cui andiamo anche a confrontare l'andamento dei valori singolari, il residuo ed i criteri di parsimonia per la determinazione dell'ordine del modello: \mathbf{Y}

9.4 Analisi parametrica di serie storiche

Finora abbiamo visto la modellazione di un sistema DLTI in cui si è voluto rappresentare esplicitamente una relazione causa-effetto tra ingresso ed uscita. Non sempre è possibile applicare questo paradigma all'analisi di sequenze di dati mediante modelli matematici. Ci sono casi in cui è disponibile una *serie storica* (o più serie storiche), che rappresenta l'andamento temporale della variabile di interesse mediante una sequenza di dati (indicizzata da un indice temporale), ma il cui andamento non è spiegabile in termini di uscita di un sistema il cui ingresso sia (almeno) parzialmente noto e deterministico.

Un approccio comunemente usato è pensare la serie storica come risposta di un sistema dinamico ad un ingresso aleatorio. Consideriamo dunque la serie storica, che è una sequenza di dati di durata finita, come parte finita di una realizzazione di un processo stocastico [22].

La giustificazione nell'utilizzo di un modello DLTI (mediante rappresentazione ARX o state-space) nell'analisi di serie storiche sta nel fatto di voler *dare una previsione dell'andamento futuro in base all'osservazione della dinamica passata*, che è uno degli scopi fondamentali per cui si cerca di modellare una serie storica.

Le serie storiche possono essere scalari, per cui vengono modellate mediante un'equazione ricorsiva alle differenze di tipo ARMA, oppure vettoriali. In questo caso è conveniente modellare anche le relazioni reciproche tra ciascuna coppia di serie storiche e lo si può fare utilmente con un modello nello spazio degli stati (2.26)-(2.23). Anche le serie storiche possono essere utilmente rappresentate nel dominio del tempo o in quello delle frequenze.

Qui prendiamo in considerazione solo la rappresentazione nel dominio nel tempo di serie storiche scalari. Come vedremo nella sottosezione seguente, analizzare una serie storica significa scomporla in componenti notevoli.

9.4.1 Scomposizione della serie storica

Da questo punto di vista, un risultato fondamentale è dato dal *teorema di Wold* [22]: "Ogni processo stazionario X può sempre decomporsi univocamente come somma di una componente puramente deterministica $X_{p.d.}$ e di una componente puramente non-deterministica $X_{p.n.d.}$ (in pratica, rumore bianco), ovvero:

$$X = X_{p.d.} + X_{p.n.d.} \quad . \quad (9.24)$$

Ovviamente, possiamo pensare di modellare, ad uso predizione, ecc., solo la componente puramente deterministica, mentre la componente puramente non-deterministica può essere osservata arbitrariamente a lungo, senza poter dire nulla di preciso sul suo futuro. Questo risultato è fondamentale: possiamo pensare di modellare la serie storica scomponendo la serie in modo additivo, fino ad ottenerne un errore di predizione prossimo ad un rumore bianco. Si dice che ci fermiamo con la modellizzazione quando abbiamo *sbiancato* la serie storica.

Dunque, in generale, supponiamo la generica serie storica composta dalla somma di componenti del tipo:

- un **trend**, che esprime l'andamento globale (costante, lineare, quadratico, ..., esponenziale) della sequenza di dati;
- delle **componenti stagionali**, che esprimono oscillazioni nei valori riconducibili a fenomeni ciclici (su base giornaliera, mensile, annua, ...) tipici dei sistemi socio-economici, naturali, ecc. .
- **fluttuazioni** attorno al trend, dovute alla risposta del sistema a fenomeni solo in parte o per nulla noti in senso deterministico (cfr. Par. 9.4.2);
- **random**, dovute ad una molteplicitá di fenomeni in parte ignoti.

Vediamo un esempio sperimentale: Y.1

Notare che la suddivisione di una serie storica nelle sue componenti é comunque decisa con un certo grado di soggettivitá. Ad esempio, l'analisi di una serie storica puó essere svolta mediante scomposizione del trend dalle fluttuazioni, che poi vengono modellate come sistema DLTI, oppure includendo anche il trend in un unico sistema dinamico (es. ARIMA oppure nello spazio degli stati).

9.4.2 Predizione lineare

L'analisi delle fluttuazioni attorno al trend viene spesso studiata come risposta di un sistema DLTI ad un processo stazionario di tipo rumore bianco, ovvero da una sequenza $u(k)$ che ha autocorrelazione:

$$r_{uu}(\tau) = E \{u(k) \cdot u(k + \tau)\} = \delta(\tau) \quad (9.25)$$

mentre l'autocorrelazione dell'uscita $y(k)$ sará in generale:

$$r_{yy}(\tau) = E \{y(k) \cdot y(k + \tau)\} = r_{yy}(-\tau) \quad (9.26)$$

da cui $r_{yy}(\tau) = r_{yy}(-\tau)$, che possiamo stimare dai dati sperimentali con la (7.29), ovvero la (7.25).

Ipotizzando dunque un modello AR del tipo:

$$y(k) = - \sum_{i=1}^{n_a} a_i \cdot y(k - i) + u(k) \quad (9.27)$$

si ha che, ponendo $r(\tau) \equiv r_{yy}(\tau)$:

$$r(\tau) = - \sum_{i=1}^{n_a} a_i \cdot r(\tau - i) \quad , \quad \tau \geq 1 \quad (9.28)$$

e

$$r(0) = 1 - \sum_{i=1}^{n_a} a_i \cdot r(i) \quad (9.29)$$

In forma matriciale, diventa:

$$\begin{bmatrix} r(0) & r(1) & r(2) & \dots & r(n_a - 1) \\ r(1) & \ddots & \ddots & \ddots & r(n_a - 2) \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ r(n_a - 1) & r(n_a - 2) & \dots & r(1) & r(0) \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ \vdots \\ a_{n_a} \end{bmatrix} = \begin{bmatrix} -r(1) \\ -r(2) \\ \vdots \\ -r(n_a) \end{bmatrix} \quad (9.30)$$

in cui la matrice è detta di Toeplitz ed è definita positiva, in quanto matrice di autocorrelazione di $y(k)$. Risolvendo il sistema (9.30) si ottengono i coefficienti a_1, \dots, a_{n_a} del modello.

Nel caso di modelli più complessi, l'algoritmo di stima comunemente usato è dovuto a Levinson.

Capitolo 10

Stima dello stato di sistemi dinamici

10.1 Stima ricorsiva dello stato di modelli DLTI nella rappresentazione *state-space*

Consideriamo un sistema nello spazio degli stati, del tipo (2.26)-(2.23):

$$x(k+1) = A \cdot x(k) + B \cdot u(k) \quad (10.1)$$

$$y(k) = C \cdot x(k) + D \cdot u(k) \quad (10.2)$$

In generale, le sequenze di ingresso e di uscita, $\{u(k)\}$ e $\{y(k)\}$, sono misurabili, ma non necessariamente lo sono le variabili del vettore di stato, $\{x(k)\}$. Tuttavia, esso è composto in genere da variabili che hanno un preciso significato fisico, e dunque poterne conoscere l'andamento durante il funzionamento del sistema è di rilevante interesse applicativo. Quindi è desiderabile poter almeno stimare il vettore di stato $\{x(k)\}$.

L'algoritmo numerico che compie la stima dello stato è detto *osservatore*, ed è esso stesso un sistema dinamico che riceve in ingresso $\{u(k)\}$ ed $\{y(k)\}$ e produce in uscita la stima dello stato, $\{\hat{x}(k)\}$.

Condizione necessaria e sufficiente affinché si possa stimare lo stato di un sistema dinamico è che il sistema sia *osservabile*, cfr. Appendice A.

Lo vediamo in un esempio preliminare: la stima dello stato iniziale. Poiché dalle sequenze $\{u(k)\}$ ed $\{y(k)\}$ ci possiamo facilmente ricavare la risposta libera

$\{y_l(k)\}$, cfr. la (A.1), abbiamo che:

$$\begin{bmatrix} y_l(0) \\ y_l(1) \\ y_l(2) \\ \vdots \\ y_l(k) \end{bmatrix} = \begin{bmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ \vdots \\ C \cdot A^k \end{bmatrix} x(0) \quad (10.3)$$

e dunque $x(0)$ può essere stimato risolvendo questo sistema sovradeterminato nel senso dei minimi quadrati, la cui matrice (di osservabilità) ha rango pieno per colonne se il sistema è osservabile.

10.1.1 costruzione dell'osservatore

Il sistema dinamico complessivo, cioè compreso l'osservatore, diventa:

$$\begin{cases} x(k+1) = A \cdot x(k) + B \cdot u(k) \\ y(k) = C \cdot x(k) + D \cdot u(k) \\ \hat{x}(k+1) = A_o \cdot \hat{x}(k) + B_o \cdot u(k) + K_o \cdot y(k) \end{cases} \quad (10.4)$$

L'errore di stima dello stato, la cui espressione può essere ottenuta sottraendo la prima dalla terza delle (10.4), con la scelta delle matrici $A_o = A - K_o \cdot C$, $B_o = B - K_o \cdot D$, e K_o opportuna, diventa:

$$\begin{aligned} x(k+1) - \hat{x}(k+1) &= A \cdot x(k) + B \cdot u(k) - A_o \cdot \hat{x}(k) - B_o \cdot u(k) - K_o \cdot y(k) \\ &= A \cdot x(k) - K_o \cdot C \cdot x(k) - A_o \cdot \hat{x}(k) \\ &\quad + B \cdot u(k) - K_o \cdot D \cdot u(k) - B_o \cdot u(k) \\ &= A_o \cdot (x(k) - \hat{x}(k)) \end{aligned} \quad (10.5)$$

e dunque A_o regola la dinamica dell'errore di stima dello stato.

Le matrici dell'osservatore vengono determinate, ad esempio, nel modo seguente:

1. K_o viene calcolata in modo che A_o abbia gli autovalori in posizioni scelte in modo arbitrario (è un *problema inverso agli autovalori*);
2. la scelta degli autovalori di A_o viene fatta in modo che la dinamica dell'osservatore sia veloce (cioè converga velocemente al valore vero della stima); se viene scelta *nilpotente*, l'errore si annulla in numero finito di passi;

Per la risoluzione del punto 1, dal punto di vista numerico possiamo fare un discorso analogo a quello fatto al paragrafo precedente: affinché il calcolo di K_o risulti numericamente efficace, devono essere utilizzate trasformazioni che si ottengono applicando matrici ortogonali (tipo Householder o Givens, cfr. parr. 1.3.2).

e 1.3.4.1), nonostante la teoria dei sistemi abbia prodotto anche altri tipi di trasformazioni, utili da un punto di vista concettuale e semplici, tanto da permettere il calcolo a mano di K_o , ma decisamente malcondizionate per sistemi già di ordine > 10 .

Vediamo un esempio sperimentale: Z.1

10.2 Stima dello stato in presenza di rumore di modello e di misura: il Filtro di Kalman

Consideriamo il sistema dinamico lineare discreto stocastico (3.8):

$$x(k+1) = A \cdot x(k) + B \cdot u(k) + v(k)$$

$$y(k) = C \cdot x(k) + D \cdot u(k) + w(k)$$

Ciò a cui aspiriamo è calcolare la *stima a minima varianza* del vettore di variabili stocastiche $\{x(k)\}$, date le misure $y(1), \dots, y(j)$:

$$\hat{x}_{k|j} = \hat{x}_{k|y(1), \dots, y(j)} \quad (10.6)$$

Quando $j = k$ la stima prende il nome di *filtro (stocastico)*, mentre quando $j = k - p$ prende il nome di *predittore (stocastico) a p-passi*. Il *Filtro di Kalman* è un algoritmo ricorsivo che risolve questo problema. Deriveremo le equazioni del filtro di Kalman come applicazione del metodo di Newton, già visto varie volte in precedenza, seguendo l'approccio presentato in [14].

10.2.1 Equazioni del filtro di Kalman

Ricordiamo che, per il teorema di Gauss-Markov, lo stimatore lineare *unbiased* a minima varianza per il modello (3.2) è dato dalla soluzione del sistema delle equazioni normali (3.11), che la matrice di covarianza dell'errore di stima dei parametri è:

$$C_{\hat{x}} = \sigma^2 (A^T A)^{-1} \quad (10.7)$$

e che per il modello di Gauss-Markov generalizzato (par. 3.1.2) ci si riconduce al caso standard, come descritto al Par. 9.1.3. E' facile verificare che tale soluzione risulta anche applicando un passo di Newton alla ricerca del minimo di (3.10), partendo da una qualunque approssimazione iniziale x . Infatti, applicando Newton alla (3.10):

$$f(x) = \|Ax - b\|_2^2 = (Ax - b)^T (Ax - b)$$

che ha gradiente $A^T Ax - A^T b$ ed hessiana $A^T A$, si ha la k-esima iterazione:

$$x(k+1) = x(k) - (A^T A)^{-1} A^T Ax(k) + (A^T A)^{-1} A^T b = (A^T A)^{-1} A^T b$$

che é indipendente in generale da $x(k)$ e quindi in particolare da $x(0)$.

Vediamo ora le equazioni del filtro nel caso di rumori scorrelati ($S = 0$). Supponiamo che lo stato iniziale sia $x(0) = \mu(0) + v(0)$, e $\mu(0)$ sia noto. Scriviamo allora un sistema lineare con le equazioni (3.8) calcolate per $k = 0, 1, \dots, m$:

$$\begin{array}{rcl} \mu(0) & = & x(0) \\ Bu(0) & = & x(1) - Ax(0) \\ y(1) & = & Cx(1) \\ & \vdots & \vdots \\ Bu(k) & = & x(k+1) - Ax(k) \\ y(k+1) & = & Cx(k+1) \end{array} \quad \begin{array}{rcl} - & v(0) \\ - & v(1) \\ + & w(1) \\ & \vdots \\ - & v(k+1) \\ + & w(k+1) \end{array} \quad (10.8)$$

che indichiamo in forma matriciale con:

$$b_{k+1|k+1} = A_{k+1|k+1}z_{k+1} + \epsilon_{k+1|k+1} \quad (10.9)$$

, ovvero un modello di Gauss-Markov generalizzato, dove $A_{k+1|k+1}$ ha per costruzione rango pieno per colonne, $z_k = [x(0); x(1); \dots; x(k)] \in \mathcal{R}^{(k+1)n}$ ed $\epsilon_{k+1|k+1}$ é un vettore di variabili aleatorie la cui covarianza é la seguente matrice a blocchi:

$$W_{k+1|k+1} = \text{diag}(Q, Q, R, \dots, Q, R) \quad . \quad (10.10)$$

Se definiamo dunque la seguente funzione costo:

$$\begin{aligned} J_{k+1|k+1}(z_{k+1}) &= \frac{1}{2} \left\| A_{k+1|k+1}z_{k+1} - b_{k+1|k+1} \right\|_{W_{k+1|k+1}}^2 \\ &= (\text{dato che } W_{k+1|k+1} \text{ é diagonale a blocchi}) \\ &= \frac{1}{2} (\|x(0) - \mu(0)\|_Q^2 + \sum_{i=1}^{k+1} \|y(i) - Cx(i)\|_R^2 + \sum_{i=1}^{k+1} \|x(i) - Ax(i-1) - Bu(i)\|_Q^2) \end{aligned} \quad (10.11)$$

il cui minimo é dato da (minimi quadrati pesati):

$$\hat{z}_{k+1|k+1} = \left(A_{k+1|k+1}^T W_{k+1|k+1}^{-1} A_{k+1|k+1} \right)^{-1} A_{k+1|k+1}^T W_{k+1|k+1}^{-1} b_{k+1|k+1} \quad (10.12)$$

che é il nostro stimatore, la cui varianza d'errore risulta:

$$\mathbb{E} \left[(\hat{z}_{k+1|k+1} - z_{k+1})(\hat{z}_{k+1|k+1} - z_{k+1})^T \right] = \left(A_{k+1|k+1}^T W_{k+1|k+1}^{-1} A_{k+1|k+1} \right)^{-1} \quad (10.13)$$

Ora, ad ogni istante discreto $k+1$ il calcolo di $\hat{z}_{k+1|k+1}$ deve ripartire daccapo in un problema che aumenta di dimensione passo dopo passo. Per ovviare a questo, esprimiamo tale soluzione in una conveniente forma ricorsiva, che utilizza un passo di ottimizzazione alla Newton con un'opportuna condizione iniziale, arrivando cosí al Filtro di Kalman. A tale scopo, definiamo $\mathcal{F}_k = [0 \dots 0 \ A] \in \mathcal{R}^{n \times kn}$

in modo tale che $\mathcal{F}_{k+1}z_k = Ax(k)$ ed esprimiamo la funzione costo $J_{k+1|k}(z_{k+1})$ in maniera ricorsiva:

$$J_{k+1|k}(z_{k+1}) = J_{k|k}(z_k) + \frac{1}{2} \|x(k+1) - \mathcal{F}_{k+1}z_k - Bu(k)\|_Q^2 \quad (10.14)$$

il cui gradiente ed Hessiano diventano, rispettivamente:

$$\nabla J_{k+1|k}(z_{k+1}) = \begin{bmatrix} \nabla J_{k|k}(z_k) + \mathcal{F}_{k+1}^T Q^{-1} (\mathcal{F}_{k+1}z_k - x(k+1) + Bu(k)) \\ -Q^{-1} (\mathcal{F}_{k+1}z_k - x(k+1) + Bu(k)) \end{bmatrix} \quad (10.15)$$

e

$$\nabla^2 J_{k+1|k}(z_{k+1}) = \begin{bmatrix} \nabla^2 J_{k|k}(z_k) + \mathcal{F}_{k+1}^T Q^{-1} \mathcal{F}_{k+1} & -\mathcal{F}_{k+1}^T Q^{-1} \\ -Q^{-1} \mathcal{F}_{k+1} & Q^{-1} \end{bmatrix} \quad (10.16)$$

e, dato che $\nabla^2 J_{k+1|k}(z_{k+1})$ risulta essere definita positiva (é facile verificarlo), una sola iterazione di Newton conduce al minimo:

$$\hat{z}_{k+1|k} = z_{k+1} - \nabla^2 J_{k+1|k}(z_{k+1})^{-1} \nabla J_{k+1|k}(z_{k+1}) \quad (10.17)$$

per qualunque z_{k+1} . Però, dato che per definizione di $\hat{z}_{k|k}$ si ha $\nabla J_{k|k}(\hat{z}_{k|k}) = 0$ ed $\mathcal{F}_{k+1}\hat{z}_{k|k} = A\hat{x}_{k|k}$, una scelta conveniente per z_{k+1} risulta essere:

$$z_{k+1} = \begin{bmatrix} \hat{z}_{k|k} \\ A\hat{x}_{k|k} + Bu(k) \end{bmatrix} \quad (10.18)$$

da cui $\nabla J_{k+1|k}(z_{k+1}) = 0$ e quindi la miglior stima di x_{k+1} date le misure y_1, \dots, y_k é l'ultima riga di z_{k+1} , e la covarianza $P(k+1|k)$, che é il blocco in basso a destra dell'Hessiana inversa $\nabla^2 J_{k+1|k}^{-1}(z_{k+1})$, e per il Lemma di Schur si ottengono le equazioni (10.19) e (10.20) (cfr. [14]). Procedendo analogamente quando si é misurato $y(k+1)$, si arriva al Filtro di Kalman.

In sintesi, il *Filtro di Kalman* é dato dalle seguenti:

- equazioni di *aggiornamento nel tempo*:

$$\hat{x}(k+1|k) = A \cdot \hat{x}(k|k) + B \cdot u_{known}(k) \quad (10.19)$$

$$P(k+1|k) = A \cdot P(k|k) \cdot A^T + Q \quad (10.20)$$

- equazioni di *aggiornamento nelle misure*:

$$\hat{x}(k+1|k+1) = \hat{x}(k+1|k) + L(k+1) \cdot [y(k+1) - C \cdot \hat{x}(k+1|k)] \quad (10.21)$$

$$P(k+1|k+1) = P(k+1|k) - P(k+1|k) \cdot C^T \cdot \Lambda(k+1)^{-1} \cdot C \cdot P(k+1|k) \quad (10.22)$$

dove $\Lambda(k) = C \cdot P(k+1|k) \cdot C^T + R$, $L(k) = P(k+1|k) \cdot C^T \cdot \Lambda^{-1}(k)$, $P(k+1|k)$ e $P(k|k)$ sono, rispettivamente, le matrici di covarianza di predizione e di filtraggio.

10.2.2 Accordo sperimentale del filtro

Un teorema importante asserisce che *la bianchezza dell'errore di predizione è una condizione sufficiente per l'ottimalità del filtro.*

Dal punto di vista pratico, il modello approssimato va *accordato* (tuned) variando Q in modo tale che l'errore di predizione risulti il più possibile assimilabile a rumore bianco. Da questo punto di vista può essere utile il test del periodogramma cumulato di Bartlett.

10.2.3 Implementazioni Square-Root del filtro di Kalman

Se si implementano le equazioni del filtro di Kalman come visto al paragrafo 10.2.1, l'aggiornamento (10.20) (10.22) della matrice di covarianza dell'errore di stima dello stato può portare, da un certo k in poi, ad una matrice $P(k)$ che non è più simmetrica e definita positiva (SPD).

Ciò può accadere, ad esempio, quando paradossalmente alcune misure sono particolarmente accurate, per cui alcuni valori diagonali di R sono ≈ 0 ed il calcolo di $P(k+1|k+1)$ diventa malcondizionato e dunque molto più sensibile agli errori di arrotondamento.

Dal punto di vista della stabilità numerica, una scelta robusta sono gli algoritmi *Square-Root*, così chiamati perché usano le fattorizzazioni di Cholesky delle matrici di covarianza o delle loro inverse, per implementare le equazioni del filtro di Kalman. Ciò è possibile in quanto le matrici Q ed R sono definite positive.

In questo modo:

- dato che si aggiornano i fattori di Cholesky L delle matrici di covarianza, è implicitamente garantito che le matrici di covarianza risultino sempre SPD;
- il numero condizionamento delle quantità usate per il calcolo, e cioè i fattori di Cholesky, è molto migliore di quello delle matrici di covarianza, in quanto ne è la radice quadrata. Nelle applicazioni di calcolo in tempo reale, in cui la precisione aritmetica è generalmente più limitata di quella disponibile in un PC, l'uso di algoritmi square-root può dunque permettere di mantenere un livello di accuratezza accettabile nelle stime del filtro di Kalman anche quando le formule convenzionali non lo consentirebbero.

Consideriamo il fattore di Cholesky $L_{k|k}$ (triangolare inferiore) di $P(k|k)$, ovvero $P(k|k) = L_{k|k}L_{k|k}^T$, e riscriviamo la (10.20). A tale scopo, consideriamo la seguente equazione di aggiornamento di $L_{k|k}$:

$$\begin{bmatrix} L_{k+1|k}^T \\ 0 \end{bmatrix} = T \begin{bmatrix} L_{k|k}^T A^T \\ (Q^{1/2})^T \end{bmatrix} \quad (10.23)$$

dove T é una matrice ortogonale tale che $L_{k+1|k}^T$ risulti triangolare superiore (e quindi la si ottiene mediante fattorizzazione QR). E' immediato verificare che, prendendo $L_{k+1|k}$ come fattore di Cholesky di $P(k+1|k)$, si ha:

$$P(k+1|k) = L_{k+1|k} L_{k+1|k}^T = \begin{bmatrix} L_{k+1|k} & 0 \end{bmatrix} \begin{bmatrix} L_{k+1|k}^T \\ 0 \end{bmatrix} = A \cdot P(k|k) \cdot A^T + Q \quad (10.24)$$

ovvero la (10.20). Inoltre, riscriviamo la (10.22) mediante il seguente aggiornamento ricorsivo di $L_{k+1|k+1}$:

$$\begin{bmatrix} (\Lambda^{1/2}(k+1))^T & \tilde{K}^T \\ 0 & L_{k+1|k+1}^T \end{bmatrix} = \bar{T} \begin{bmatrix} (R^{1/2})^T & 0 \\ L_{k+1|k}^T C^T & L_{k+1|k}^T \end{bmatrix} \quad (10.25)$$

dove \bar{T} é una matrice ortogonale e la si ottiene analogamente alla T . E' immediato verificare che questo schema é equivalente alla (10.22).

10.3 Stima dello stato iniziale in sistemi evolutivi nonlineari

Abbiamo visto in precedenza quanto sia immediato stimare lo stato iniziale di un sistema lineare una volta noto l'andamento della sequenza di ingresso e di uscita. E' sufficiente risolvere il sistema sovradeterminato (10.3).

Ora, discretizzando nello spazio e nel tempo un modello retto da equazioni differenziali lineari si ottiene un sistema DLTI, cf. Appendice B. Ci sono però importanti applicazioni che vengono modellate con equazioni differenziali nonlineari. In tal caso il modello discreto sará costituito da un sistema di equazioni alle differenze nonlineari. Per stimare lo stato iniziale da misure prese in un insieme di istanti si procede analogamente a quanto fatto per la stima di parametri con il metodo PEM (??), ovvero vogliamo rendere minima la distanza tra i dati simulati e quelli sperimentali, misurata da una funzione costo J . Le equazioni dinamiche del modello entrano qui come vincoli da soddisfare. I parametri da stimare sono ora i valori assunti dalla soluzione al tempo $t = 0$, $u(0)$.

Una differenza fondamentale con il caso della stima dei parametri del modello é che in quel caso si puó spesso ridurre il numero dei parametri da stimare, facendo delle ipotesi di omogeneitá dei coefficienti in zone relativamente vaste del dominio, mentre qui occorre stimare la soluzione iniziale, per cui é molto difficile effettuare una qualunque riduzione del numero di parametri da stimare.

La procedura risolutiva é necessariamente iterativa. Ad ogni iterazione, partendo da una stima iniziale delle condizioni iniziali:

- si calcola il gradiente $\nabla_{u(0)} J$ della funzione costo rispetto alle condizioni iniziali;

- si usa un passo di aggiornamento delle stime del tipo "quasi-Newton" (come in PEM) oppure "gradiente coniugato".

Ora, l'elevato numero di variabili da stimare rende particolarmente oneroso il calcolo del gradiente, tanto che per modelli di grandi dimensioni il metodo diventa inapplicabile. Per fare un esempio, un modello di utilizzo reale nelle previsioni atmosferiche puó avere $O(10^6)$ variabili. Per ovviare a questo problema, é stato proposto il *metodo del modello aggiunto* per ridurre il costo necessario al calcolo del gradiente $\nabla_{u(0)} J$ e il procedimento di stima in questo caso va sotto il nome di *Variational Data Assimilation* (VDA).

In generale, a partire dal modello non lineare viene formulato, tramite un procedimento di *linearizzazione*, il modello lineare tangente, dalla cui trasposizione si ottiene il modello aggiunto. Il gradiente si ottiene con la risoluzione iterativa all'indietro nel tempo di tale modello, imponendo all'ingresso le *variabili aggiunte*, definite dalla differenza tra le predizioni del modello e le osservazioni (in sostanza sono l'errore di predizione).

10.3.1 Discretizzazione nel tempo

In generale, le equazioni del modello nonlineare evolutivo forniscono la mappa di aggiornamento dello stato in forma implicita, cioé come soluzione di un sistema di equazioni differenziali ordinarie non lineari di primo grado. Come anticipato al paragrafo precedente, l'obiettivo é quello di determinare le condizioni iniziali la cui traiettoria evolutiva approssimi i dati derivanti dall'osservazione. Ovviamente, se la condizione iniziale $\mathbf{y}(0)$ é nota, é possibile con un metodo che discretizzi il modello nonlineare evolutivo, calcolare la soluzione agli istanti successivi $\mathbf{Y}(k)$.

Dato il passo temporale (o periodo di campionamento) T , prendiamo in considerazione il θ -metodo (B.11):

$$\frac{\mathbf{Y}(k+1) - \mathbf{Y}(k)}{T} = (1 - \theta)f(k, \mathbf{Y}(k)) + \theta f(k + 1, \mathbf{Y}(k + 1))$$

La mappa di aggiornamento di un passo é:

$$\mathbf{Y}(k + 1) = N(\mathbf{Y}(k)) \quad (10.26)$$

dove N indica che l'applicazione da risolvere al passo k , per essere in grado di determinare la soluzione al passo successivo, é non lineare. Applicando iterativamente questa equazione, si vede che la traiettoria futura é unicamente determinata dalle condizioni iniziali. Noto il valore all'istante iniziale $\mathbf{Y}(0)$, é possibile calcolare il valore dello stato all'istante k , $\mathbf{Y}(k)$:

$$\mathbf{Y}(k) = N(\dots(N(\mathbf{Y}(0)))) \quad (10.27)$$

In particolare, mediante opportuna scelta del passo di discretizzazione nel tempo T é possibile mantenere l'errore di discretizzazione ad un livello confrontabile con l'errore presente nei dati sperimentali, in modo da poter assumere nel seguito che $\mathbf{Y}(kT) \approx \mathbf{y}(kT)$, $\forall k$.

10.3.2 Stima delle condizioni iniziali (stato iniziale)

Si consideri ora il caso in cui lo stato iniziale esatto $\mathbf{Y}(0)$ é sconosciuto, e invece sono note alcune osservazioni in alcuni istanti, $\bar{\mathbf{Y}}(k)$, dove k indica l'istante temporale discreto. Lo scopo é quello di utilizzare le osservazioni disponibili per ottenere soluzioni del modello piú vicine possibili alla traiettoria esatta. Introduciamo la funzione costo J , che misura la distanza tra le soluzioni del modello e le osservazioni. In questo caso particolare la funzione J é data dalla somma in tutti gli istanti della discretizzazione, della differenza in norma $\|\cdot\|_2$ tra gli stati calcolati dal modello numerico (predizioni) e i dati sperimentali (osservazioni):

$$J(\mathbf{Y}(0)) = \frac{1}{2} \sum_{k=0}^{k_{max}} (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)) \quad (10.28)$$

dove T é l'operatore di trasposizione, e $\bar{\mathbf{Y}}(k)$ ha la stessa dimensione di $\mathbf{Y}(k)$. Nel caso di lacune nei dati sperimentali, le rispettive componenti di $\bar{\mathbf{Y}}(k)$ vengono poste uguali a quelle di $\mathbf{Y}(k)$ e non danno alcun contributo alla sommatoria in (10.28). In sostanza, la funzione costo indica solo la discrepanza tra dati simulati e dati sperimentali disponibili e non la copertura dei dati sperimentali, ovvero la quantitá di informazione disponibile. Lo stato iniziale che meglio approssima i dati osservati sará quello che minimizza la funzione costo, ed é in generale incognito. La sua ricerca, da effettuarsi per via numerica, deve partire necessariamente da una stima dello stato iniziale $\hat{\mathbf{Y}}(0)$.

Partendo da un punto iniziale $\mathbf{Y}^0(0)$, l'algoritmo iterativo del gradiente diventa:

$$\mathbf{Y}^{n+1}(0) = \mathbf{Y}^n(0) - \alpha_n \nabla J(\mathbf{Y}^n(0)) \quad (10.29)$$

dove n rappresenta il numero dell'iterazione, α_n é un parametro scelto in modo da assicurare la convergenza e ∇J e' il gradiente della funzione costo J rispetto allo stato iniziale $\mathbf{Y}^n(0)$.

10.3.3 Il metodo del modello aggiunto

Vediamo ora come, in problemi applicativi caratterizzati da un enorme numero di parametri, quali i problemi atmosferici, il modello aggiunto permetta di ottenere una buona approssimazione del gradiente della funzione costo (10.28).

I sistemi dinamici lineari vengono spesso utilizzati per studiare in modo approssimato la dinamica di sistemi dinamici non lineari; il procedimento secondo il

quale si associa un sistema lineare ad uno non lineare viene detto linearizzazione [19].

Si consideri una generica soluzione del modello non lineare $\mathbf{Y}(k)$, e si indichi con $\mathbf{X}(k)$ una perturbazione dello stato all'istante k dal valore $\mathbf{Y}(k)$. Si definisca inoltre lo stato perturbato $\tilde{\mathbf{Y}}(k) = \mathbf{Y}(k) + \mathbf{X}(k)$. L'equazione di aggiornamento di un passo del sistema non lineare in un intorno dello stato $\mathbf{Y}(k)$ si approssima sviluppando secondo la formula di Taylor, arrestata al primo ordine, la funzione N introdotta in (10.26). Si ha:

$$\tilde{\mathbf{Y}}(k+1) = N(\tilde{\mathbf{Y}}(k)) = N(\mathbf{Y}(k) + \mathbf{X}(k)) = N(\mathbf{Y}(k)) + \frac{\partial N(\mathbf{Y}(k))}{\partial \mathbf{Y}} \mathbf{X}(k) + \epsilon. \quad (10.30)$$

Il termine ϵ è infinitesimo di ordine superiore rispetto a $\|\mathbf{X}(k)\|$ e con $\frac{\partial N(\mathbf{Y}(k))}{\partial \mathbf{Y}}$ si intende la matrice Jacobiana di N valutata in $\mathbf{Y}(k)$. Tenendo conto della definizione di $\tilde{\mathbf{Y}}(k)$ e trascurando ϵ si ottiene

$$\mathbf{X}(k+1) = \frac{\partial N(\mathbf{Y}(k))}{\partial \mathbf{Y}} \mathbf{X}(k). \quad (10.31)$$

Il modello sopra definito è detto modello lineare tangente ed il vettore \mathbf{X} , detto vettore lineare tangente, è il vettore di stato di tale modello.

Poiché è evidente che la soluzione al passo $k+1$ dipende unicamente dalla risoluzione del modello lineare tangente con input la variabile di stato del modello non lineare $\mathbf{Y}(k)$, in forma generale è possibile scrivere

$$\mathbf{X}(k+1) = L(\mathbf{Y}(k))\mathbf{X}(k) \quad (10.32)$$

in cui $L(\mathbf{Y}(k))$ è detto operatore lineare tangente (TL). Utilizzando questa equazione è facile stabilire la relazione tra $\mathbf{X}(k)$ e $\mathbf{X}(0)$

$$\mathbf{X}(k) = L(\mathbf{Y}(k-1))\mathbf{X}(k-1) = L(\mathbf{Y}(k-1))L(\mathbf{Y}(k-2))\mathbf{X}(k-2) = \dots = L_k \mathbf{X}(0) \quad (10.33)$$

dove

$$L_k = L(\mathbf{Y}(k-1))L(\mathbf{Y}(k-2))\dots L(\mathbf{Y}(1))L(\mathbf{Y}(0)).$$

Si consideri ora la funzione costo definita in (10.28). La variabile lineare tangente \mathbf{X} appena definita è una perturbazione dalla soluzione del modello non lineare \mathbf{Y} . Considerando lo sviluppo in serie di Taylor della funzione costo J nel punto iniziale $\mathbf{Y}(0)$, si ottiene

$$J(\mathbf{Y}(0) + \mathbf{X}(0)) = J(\mathbf{Y}(0)) + \nabla J(\mathbf{Y}(0))^T \mathbf{X}(0) + o(\|\mathbf{X}(0)\|)$$

da cui, definendo

$$J^{tl}(\mathbf{Y}(0)) = J(\mathbf{Y}(0) + \mathbf{X}(0)) - J(\mathbf{Y}(0)) \quad (10.34)$$

, dalla formula sopra scritta si ottiene:

$$J^{tl}(\mathbf{Y}(0)) \approx \nabla J(\mathbf{Y}(0))^T \mathbf{X}(0) \quad (10.35)$$

Inoltre, utilizzando la definizione della funzione costo (10.28) e sostituendola nella definizione di J^{tl} (10.34), si ottiene:

$$\begin{aligned} J^{tl}(\mathbf{Y}(0)) &= \frac{1}{2} \sum_{k=0}^{kmax} (\mathbf{Y}(k) + \mathbf{X}(k) - \bar{\mathbf{Y}}(k))^T (\mathbf{Y}(k) + \mathbf{X}(k) - \bar{\mathbf{Y}}(k)) \\ &\quad - \frac{1}{2} \sum_{k=0}^{kmax} (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)) \\ &= \frac{1}{2} \sum_{k=0}^{kmax} (\mathbf{X}(k)^T (\mathbf{Y}(k) + \mathbf{X}(k) - \bar{\mathbf{Y}}(k)) + (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k)) \\ &= \frac{1}{2} \sum_{k=0}^{kmax} (\mathbf{X}(k)^T (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)) + (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k) + \mathbf{X}^T(k) \mathbf{X}(k)) \\ &= (\mathbf{X}(k)^T (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))) = ((\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k) \text{ perché scalari}) \\ &= \sum_{k=0}^{kmax} ((\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k) + O(\|\mathbf{X}(k)\|^2)) \\ &\approx \sum_{k=0}^{kmax} (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k) \end{aligned} \quad (10.36)$$

dove con $\bar{\mathbf{Y}}$ si intende il vettore delle osservazioni. Uguagliando i risultati trovati in (10.35) e (10.36) si ottiene l'equazione

$$[\nabla J(\mathbf{Y}(0))]^T \mathbf{X}(0) \approx \sum_{k=0}^{kmax} (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T \mathbf{X}(k) \quad (10.37)$$

che, una volta trovata la relazione tra $\mathbf{X}(0)$ e $\mathbf{X}(k)$ permette di calcolare il gradiente della funzione costo. La relazione cercata è espressa dall'equazione (10.33), ottenuta risolvendo il modello lineare tangente

$$[\nabla J(\mathbf{Y}(0))]^T \mathbf{X}(0) \approx \sum_{k=0}^{kmax} (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k))^T L_k \mathbf{X}(0) \quad (10.38)$$

da cui si ricava una formula per il calcolo del gradiente

$$\nabla J(\mathbf{Y}(0)) \approx \sum_{k=0}^{kmax} L_k^T (\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)) \quad (10.39)$$

dove L_k^T è il trasposto di L_k , è definito da

$$L_k^T = L^T(\mathbf{Y}(0)) L^T(\mathbf{Y}(1)) \dots L^T(\mathbf{Y}(k-2)) L^T(\mathbf{Y}(k-1))$$

ed è chiamato **operatore aggiunto**, ed il corrispondente modello aggiunto (AD) definito da

$$\mathbf{Y}^{ad}(k-1) = L^T(\mathbf{Y}(k-1)) \mathbf{Y}^{ad}(k) \quad (10.40)$$

dalla cui risoluzione all'indietro nel tempo è possibile ricavare la soluzione al tempo 0

$$\mathbf{Y}^{ad}(0) = L_k^T \mathbf{Y}^{ad}(k) \quad (10.41)$$

dove $\mathbf{Y}^{ad}(k)$ indica la variabile aggiunta al passo k , cioè la soluzione del modello aggiunto all'istante k . L'equazione aggiunta ha la stessa forma dell'equazione lineare tangente con due differenze: l'operatore lineare che compare in (10.41) e' il trasposto dell'operatore lineare che compare in (10.33), e la sequenza temporale e' rovesciata. Inizializzando il modello aggiunto con input la differenza tra $\mathbf{Y}(k)$ e l'osservazione al tempo k , $\bar{\mathbf{Y}}(k)$

$$\mathbf{Y}^{ad}(k) = \mathbf{Y}(k) - \bar{\mathbf{Y}}(k) \quad (10.42)$$

si ottiene dall'equazione 10.41

$$L_k^T(\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)) \quad (10.43)$$

che e' esattamente l'espressione che compare nella somma (10.39); quindi la risoluzione del modello aggiunto dall'istante k_{max} a quello iniziale, con input $\mathbf{Y}(k) - \bar{\mathbf{Y}}(k)$ da' il k -esimo contributo al gradiente della funzione costo. In generale il procedimento iterativo consiste in:

- a partire dalla variabile $\mathbf{Y}^{ad}(k)$ definita in (10.42) si risolve il modello aggiunto (10.40);
- sfruttando la linearità del modello, alla soluzione ottenuta si somma la nuova variabile aggiunta $\mathbf{Y}^{ad}(k-1) = \mathbf{Y}(k-1) - \bar{\mathbf{Y}}(k-1)$ e si applica nuovamente il modello aggiunto;
- procedendo iterativamente si ottiene la somma cercata: grazie alla linearità del modello aggiunto, il procedimento è equivalente a risolvere il modello aggiunto per tutte le variabili aggiunte $\mathbf{Y}^{ad}(k)$ dal loro istante a quello iniziale e sommare i risultati ottenuti.

E' possibile osservare quindi che una sola risoluzione del modello aggiunto dal passo k_{max} al passo 0 con input le variabili aggiunte definite in (10.42) permette di calcolare una buona approssimazione del gradiente della funzione costo con un notevole risparmio del tempo di esecuzione.

10.3.4 Procedura di ottimizzazione con il modello aggiunto

Col metodo della Variational Data Assimilation, che utilizza il modello aggiunto per il calcolo del gradiente della funzione costo, è possibile ottenere una buona approssimazione dello stato iniziale $\mathbf{y}(0)$ tramite un processo iterativo. Partendo da un insieme di osservazioni $\bar{\mathbf{Y}}(k)$, $k = 0, 1, \dots, k_{max}$, ed uno stato iniziale $\mathbf{Y}^{(0)}(0)$ si risolve il modello non lineare dal tempo 0 al tempo k_{max} , producendo in tal modo la traiettoria evolutiva $\mathbf{Y}^{(0)}(k)$, per $k = 0, 1, \dots, k_{max}$, rispetto allo stato iniziale dato. Dopo aver determinato le variabili aggiunte come differenza tra le soluzioni del modello non lineare calcolate e le osservazioni, $\mathbf{Y}^{(0)}(k) - \bar{\mathbf{Y}}(k)$, si risolve il modello aggiunto, che permette di calcolare il gradiente della funzione

costo rispetto allo stato iniziale, $\nabla J(\mathbf{Y}^{(0)}(0))$. A questo punto, l'algoritmo di ottimizzazione produrrà una nuova approssimazione dello stato iniziale $\mathbf{Y}^1(0)$. Ripetendo iterativamente la procedura è possibile ottenere il minimo della funzione costo J desiderato ad un costo computazionale ragionevole.

Oltre alle previsioni atmosferiche, un'applicazione interessante è la localizzazione di una sorgente di inquinamento.

10.3.5 Esempio di Lorenz

1.1 Il modello di Lorenz

Nelle scienze applicate, i modelli matematici utilizzati assumono spesso la forma di equazioni differenziali, la cui possibilità di risoluzione è notevolmente aumentata negli ultimi decenni grazie allo sviluppo di computer potenti e di algoritmi efficienti in grado di risolvere tali problemi.

Il modello analizzato in questo caso è esattamente un sistema di tre equazioni differenziali non lineari elaborato da Lorenz nel 1963 [1].

$$\frac{dy_1(t)}{dt} = -py_1(t) + py_2(t), \quad (1.1)$$

$$\frac{dy_2(t)}{dt} = [r - y_3(t)]y_1(t) - y_2(t), \quad (1.2)$$

$$\frac{dy_3(t)}{dt} = y_1(t)y_2(t) - by_3(t), \quad (1.3)$$

dove t indica che si tratta di un fenomeno che evolve nel tempo, p è il numero di Prandtl¹, r è il numero modificato di Rayleigh², b è il rapporto di forma, $y_1(\cdot)$ l'intensità

¹Il numero di Prandtl è importante nello studio di fenomeni di scambio termico per convezione

²Il numero di Rayleigh è importante nei fenomeni in cui si ha convezione naturale così come il numero di Reynolds lo è per la convezione forzata

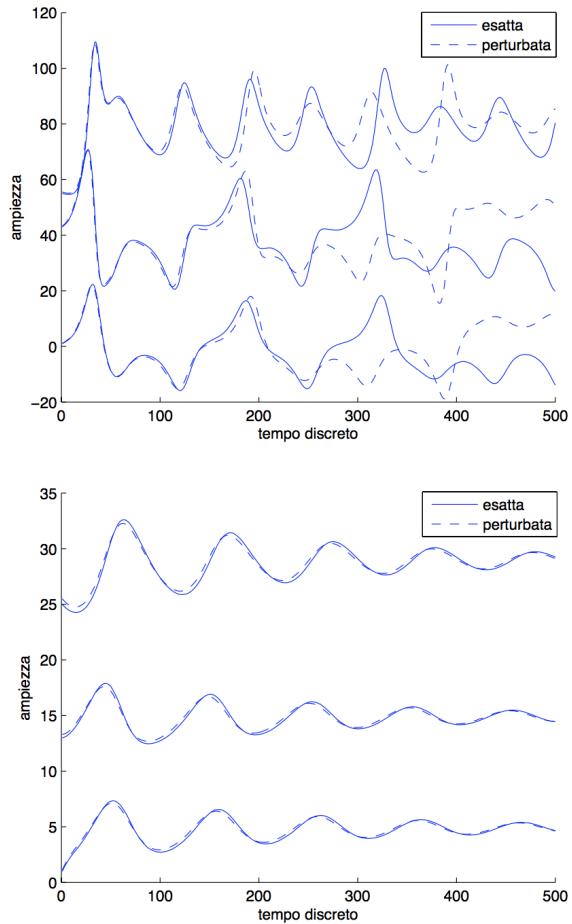


Figura 3.2: instabilità del sistema rispetto a piccole perturbazioni con numero di Rayleigh 32 e sua stabilità con numero di Rayleigh 10

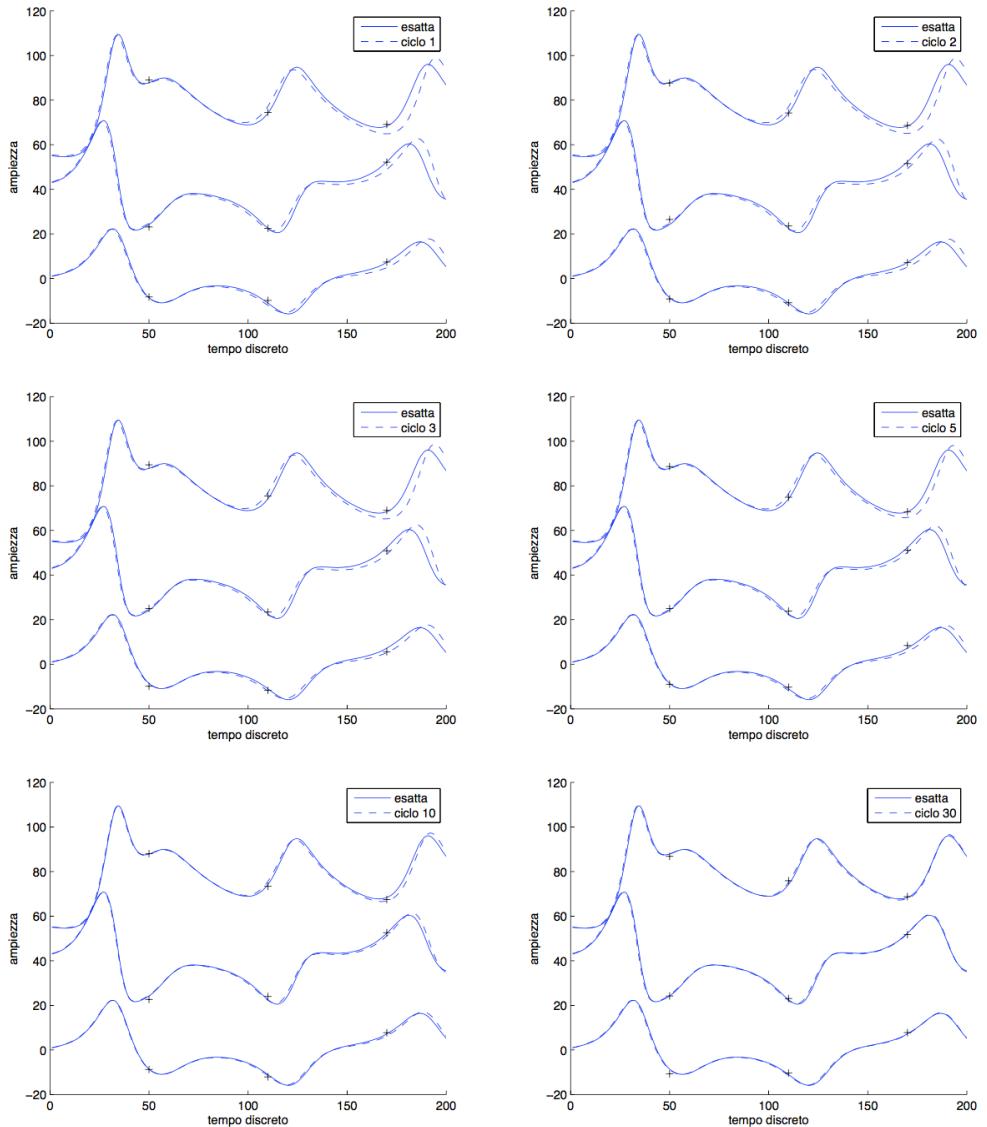


Figura 3.3: traiettoria esatta e traiettoria vda a partire da tre vettori di osservazioni a disposizione

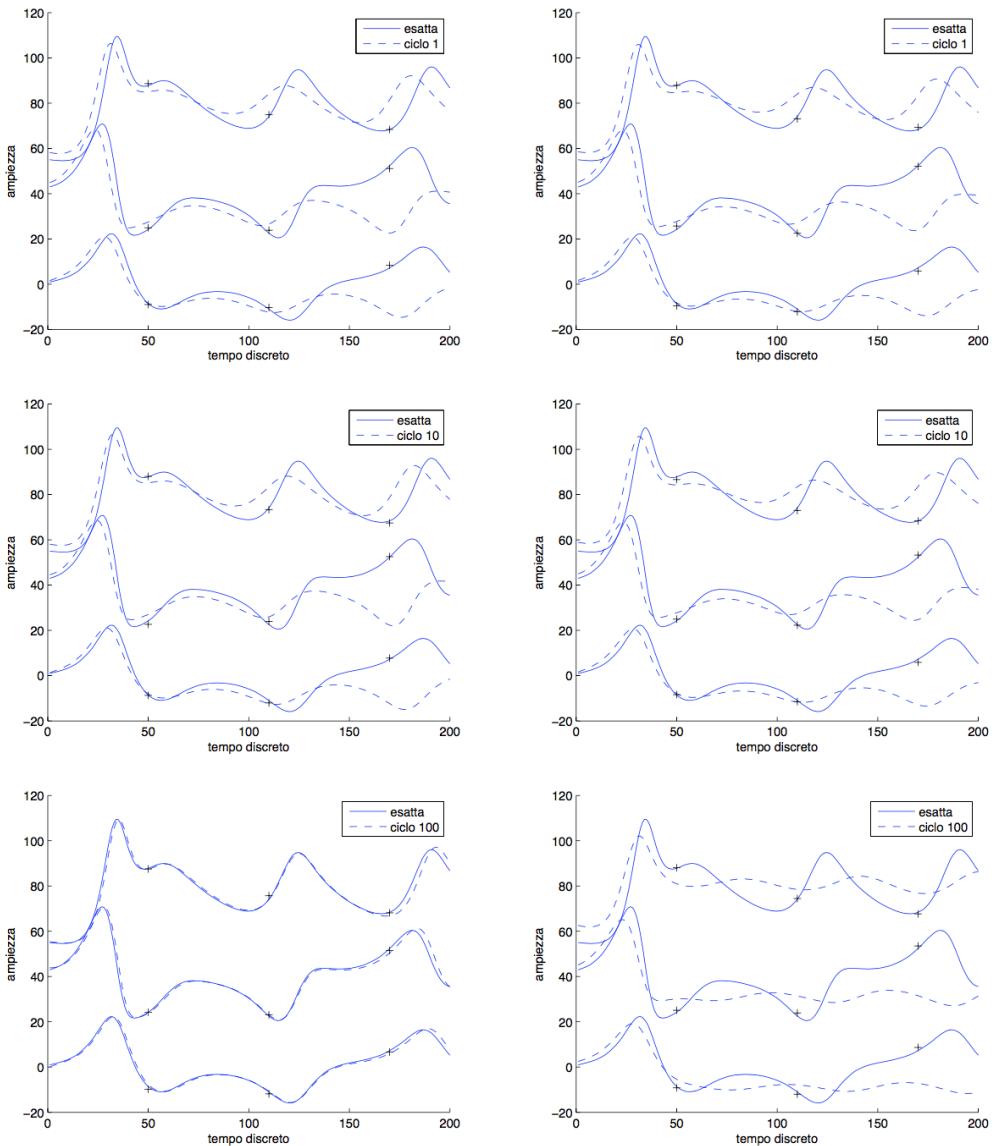


Figura 3.5: traiettoria esatta e soluzione vda a partire da punto con errore del 65% (prima colonna) e del 70% (seconda colonna) rispetto al punto iniziale esatto

Capitolo 11

Stima dei parametri nelle reti neurali

Nell'analisi dei dati si utilizzano anche modelli nonlineari nelle variabili. Lo sviluppo di modelli nonlineari è avvenuto principalmente nell'ambito della cosiddetta *pattern analysis*, che ricerca relazioni tra i dati del tipo *regole di classificazione* (classification rules), *funzioni di regressione* (regression functions) o strutture *cluster*. Vedremo di seguito i metodi numerici fondamentali che riguardano:

1. la ricerca di relazioni lineari tra i dati (*perceptron*, 1957);
2. la ricerca di relazioni nonlineari (*backpropagation multilayer neural networks*, metà anni '80);
3. il Deep Learning.

11.1 Reti neurali

11.1.1 Modello non-lineare *perceptron*

Un limite fondamentale, dal punto di vista pratico, del modello lineare (3.2) è che la variabile risposta dipende linearmente dalle variabili esplicative senza limiti. Nella realtà, invece, il comportamento lineare ha in generale un intervallo di validità ristretto. E' naturale dunque pensare di estendere il modello lineare introducendo una saturazione, come è stato fatto nel *Perceptron*¹:

$$y_i = f(q_1 \cdot x_1 + q_2 \cdot x_2 + \dots + q_n \cdot x_n) \quad (11.1)$$

¹Il *Perceptron* è stato sviluppato da Rosenblatt (1958) ispirandosi ai modelli che allora venivano studiati sul funzionamento del cervello umano, in particolare della cellula neuronale, o *neurone*.

dove ora la variabile risposta y_i dipende dalle variabili eplicative q_i in modo non-lineare, grazie alla *funzione di attivazione* $f(\cdot)$, che é nonlineare. In un problema di *apprendimento* dei valori dei parametri x , in cui le informazioni contenute nel vettore di osservazioni y vengono utilizzate per indurre i valori dei suoi parametri x , il vettore delle osservazioni y é associato al vettore di parametri di regressione $x \in R^n$ mediante il modello non lineare:

$$A(x) = \begin{bmatrix} \vdots \\ f(q^{(i)} \cdot x) \\ \vdots \end{bmatrix} = y \quad (11.2)$$

Per la soluzione del problema di apprendimento vengono solitamente utilizzate delle varianti dei minimi quadrati nonlineari, cfr. in particolare l'algoritmo di *backpropagation* al par. 11.2.

Ci sono varie possibilità per la scelta della funzione di attivazione f :

- $f(s) = \begin{cases} 1 & , s \geq 0 \\ -1 & , s < 0 \end{cases}$, detta *hard limiter*, produce un modello che **classifica** i dati in base al fatto di essere a sinistra o a destra dell'iperpiano di separazione individuato dai coefficienti x_i del modello.
- $f(s) = \begin{cases} 1 & , s \geq \alpha \\ \frac{s}{\alpha} & , -\alpha \leq s \leq \alpha \\ -1 & , s < -\alpha \end{cases}$, lineare a tratti, produce un modello lineare nell'intervallo $s \in [-\alpha, \alpha]$, dopodiché satura immediatamente; é una funzione solo continua (C^0).
- $f(s) = \tanh(s) = \frac{2}{1+e^{-2s}} - 1$, detta *sigmoide*, produce un modello pressoché lineare in un intervallo di s centrato in zero, dopodiché tende asintoticamente a saturare, ma con un andamento molto regolare.
- $f(s) = \begin{cases} 0 & , s \leq 0 \\ x & , s > 0 \end{cases}$, detta *REctified Linear Unit (ReLU)*, che ha dimostrato di funzionare bene in pratica su reti grandi.

Oltre all'andamento delle funzioni di attivazione, é importante considerarne anche il valore per $x = 0$, che diventa approssimativamente il valor medio dell'uscita del neurone, e quindi un segnale in ingresso ad un neurone del layer successivo della rete. Sperimentalmente é stato osservato che, quando i neuroni hanno segnali in ingresso con valori medi diversi da zero, l'apprendimento rallenta sensibilmente. Recentemente, la *ReLU* ha dimostrato di funzionare meglio delle altre: la nonlinearitá é quindi data dall'avere una funzione **lineare a tratti**.

Il *perceptron* é il modello di un singolo neurone della rete. Puó effettuare la classificazione di configurazioni in ingresso che siano *linearmente separabili*, tali cioé

che configurazioni appartenenti a classi differenti stiano dalle parti opposte di un iperpiano. Questo modello ha mostrato dei limiti notevoli, vedi il caso XOR: non riesce a riprodurne la regola, ed il fatto che questa sia una regola di base della logica ha creato in passato una certa titubanza nel ritenerlo un buon modello *neurale*. In realtà è una pietra miliare importante del cosiddetto *machine learning*. Infatti, aggiungendo neuroni e collegandoli a rete, è possibile discriminare un numero maggiore di classi, a patto che siano linearmente separabili.

11.1.2 Modello non-lineare “multi-layer neural network”

Il modello *rete neurale* si basa sulla connessione a rete di neuroni come il *Perceptron*. Esistono molte varianti; qui vediamo in dettaglio uno schema fondamentale. Pensiamo di suddividere i neuronidella rete in *strati* (o *layers*), attraverso i quali i dati in ingresso si propagano producendo i valori in uscita della rete. Ogni layer avrà un certo numero di neuroni. Ogni neurone è collegato ad uno o più neuroni del layer precedente (o direttamente alle variabili in ingresso, se è un neurone del primo layer) e del successivo (o direttamente alle variabili di uscita, se è un neurone dell'ultimo layer).

In termini matematici, definiamo una rete a q strati nel modo seguente²:

- Ingressi: $u = [u_1, u_2, \dots, u_n]^T$; Nota: ad ogni strato viene aggiunto un ingresso fittizio, pari a 1;

- uscite: $y = [y_1, y_2, \dots, y_m]^T$;

- strato di ingresso, $j = 1$: $\text{neuron}(1, i)$, $i = 1, 2, \dots, p_1$ con peso generico $\gamma_{ik}^{(1)}$, $k = 0, 1, 2, \dots, n$; in forma matriciale, detta $f(\cdot)$ la funzione di attivazione, $w_i^{(1)} = [\{\gamma_{ik}^{(1)}\}]^T$ e $W^{(1)} = [\{w_i^{(1)}\}]$:

$$\begin{cases} s^{(1)} &= W^{(1)} \begin{bmatrix} 1 \\ u \end{bmatrix} \\ z^{(1)} &= f(s^{(1)}) \end{cases}$$

- strato intermedio, $1 < j < q$: $\text{neuron}(j, i)$, $i = 1, 2, \dots, p_j$ con peso generico $\gamma_{ik}^{(j)}$, $k = 0, 1, 2, \dots, p_{j-1}$; in forma matriciale, detta $f(\cdot)$ la funzione di attivazione, $w_i^{(j)} = [\{\gamma_{ik}^{(j)}\}]^T$ e $W^{(j)} = [\{w_i^{(j)}\}]$:

$$\begin{cases} s^{(j)} &= W^{(j)} \begin{bmatrix} 1 \\ z^{(j-1)} \end{bmatrix} \\ z^{(j)} &= f(s^{(j)}) \end{cases}$$

²Qui seguiremo il linguaggio comunemente usato per le reti neurali, dove i coefficienti x della combinazione lineare (11.1) vengono detti *pesi* (*weights*) e non *parametri*, anche se di fatto sono i parametri del modello, che vengono stimati durante la fase di apprendimento della rete.

- strato di uscita, $j = q$: $neuron(q, i)$, $i = 1, 2, \dots, p_q$ con peso generico $\gamma_{ik}^{(q)}$, $k = 0, 1, 2, \dots, p_{q-1}$; in forma matriciale, detta $f(\cdot)$ la funzione di attivazione, $w_i^{(q)} = [\{\gamma_{ik}^{(q)}\}]^T$ e $W^{(q)} = [\{w_i^{(q)}\}]$:

$$\begin{cases} s^{(q)} &= W^{(q)} \begin{bmatrix} 1 \\ z^{(q-1)} \end{bmatrix} \\ y &= f(s^{(q)}) \end{cases}$$

- relazione ingresso-uscita complessiva:

$$y = f \left(W^{(q)} \begin{bmatrix} & & 1 \\ & \ddots & \\ \dots & f \left(W^{(j)} \begin{bmatrix} & & 1 \\ & \ddots & \\ \dots & f \left(W^{(1)} \begin{bmatrix} 1 \\ u \end{bmatrix} \right) & \dots \end{bmatrix} \right) & \dots \end{bmatrix} \right) \in \Re^m$$

Ci sono alcuni risultati teorici interessanti, come il Teorema dell'approssimazione universale, inizialmente dimostrato da Cybenko [8] per le funzioni multi-variate continue e poi successivamente esteso.

11.1.3 Modello non-lineare *deep neural network*

Le reti neurali deep sono caratterizzate dall'avere molti layer interni. Questo porta ad un numero di parametri molto alto e quindi ad un rischio elevato di overfitting. Per ridurre questo rischio in modo accettabile ci sono alcuni accorgimenti fondamentali:

- riduzione del numero di parametri mediante approccio *convoluzionale*: "stencil" di coefficienti e shift degli stessi. Buoni risultati dove i dati hanno una struttura locale, es. in suoni ed immagini;
- *pooling*: di risultati intermedi vicini ne prendo solo uno: il massimo (nonlineare) o la media (lineare);
- algoritmi con stochastic (minibatch) gradient descend e momentum, scelta del passo di apprendimento e stop anticipato (vedi Capitolo ??).

Consideriamo i modelli neurali (11.1) e del par. 11.1.2. Spesso una rete neurale viene utilizzata per riconoscere l'appartenenza di una configurazione (*pattern*) di valori in ingresso ad una determinata classe, e quindi per assegnare un ben preciso valore in uscita a tutte le configurazioni di una stessa classe. per questo motivo si parla sovente di *classificazione di configurazioni in ingresso* (input pattern classification) quando si parla di reti neurali. Ma una rete neurale può anche riprodurre il

comportamento di una funzione nonlineare multi-variata di elevata complessità. Se la funzione è ignota, la rete può riprodurla mediante *apprendimento supervisato* su un insieme significativo di esempi (in questo caso, valori della funzione in un insieme discreto di punti del dominio, ovvero dello spazio di ingresso alla rete).

11.2 L'algoritmo di *back-propagation*

L'apprendimento di una rete neurale come descritta al par. 11.1.2, viene solitamente svolto mediante l'algoritmo di *back-propagation*. Consideriamo per semplicità una rete a due strati:

- Ingressi: $u = [u_1, u_2, \dots, u_n]^T$; Nota: ad ogni strato viene aggiunto un ingresso fittizio, pari a 1;
- uscite: $y = [y_1, y_2, \dots, y_m]^T$;
- strato di ingresso: $\text{neuron}(1, i)$, $i = 1, 2, \dots, p$ con peso generico $\gamma_{ik}^{(1)}$, $k = 0, 1, 2, \dots, n$; in forma matriciale, detta $f(\cdot)$ la funzione di attivazione, $w_i^{(1)} = [\{\gamma_{ik}^{(1)}\}]^T$ e $W^{(1)} = [\{w_i^{(1)}\}]$:

$$\begin{cases} s^{(1)} &= W^{(1)} \begin{bmatrix} 1 \\ u \end{bmatrix} \\ z &= f(s^{(1)}) \end{cases}$$

- strato di uscita: $\text{neuron}(2, i)$, $i = 1, 2, \dots, m$ con peso generico $\gamma_{ik}^{(2)}$, $k = 0, 1, 2, \dots, p$; in forma matriciale, detta $f(\cdot)$ la funzione di attivazione, $w_i^{(2)} = [\{\gamma_{ik}^{(2)}\}]^T$ e $W^{(2)} = [\{w_i^{(2)}\}]$:

$$\begin{cases} s^{(2)} &= W^{(2)} \begin{bmatrix} 1 \\ z \end{bmatrix} \\ y &= f(s^{(2)}) \end{cases}$$

- relazione ingresso-uscita complessiva:

$$y = f \left(W^{(2)} \begin{bmatrix} 1 \\ f \left(W^{(1)} \begin{bmatrix} 1 \\ u \end{bmatrix} \right) \end{bmatrix} \right) \in \Re^m$$

L'apprendimento è basato sulla cosiddetta *legge delta generalizzata* (generalized delta rule) [Rumelhart e McClelland (1986)]. Dato un insieme di vettori target, $d(k) = [d_1(k) \ d_2(k) \ \dots \ d_m(k)]^T$, consideriamo l'errore di predizione:

$$E(k) = \frac{1}{2} \sum_{j=1}^m (d_j(k) - y_j(k))^2 = \frac{1}{2} \sum_{j=1}^m e_j^2(k) \quad , \quad k = 1, 2, \dots \quad (11.3)$$

dove il fattore $1/2$ è messo per convenienza nel calcolo delle derivate che seguono. L'apprendimento avviene cercando il minimo di $E(k)$ iterativamente per $k = 0, 1, 2, \dots$. Ad ogni iterazione, tale minimo viene cercato utilizzando l'algoritmo di *back-propagation*. Dunque, per non appesantire troppo la notazione, descriviamo qui nel seguito l'algoritmo di backpropagation omettendo l'indice k , ad esempio porremo $E(k) = E$.

Per trovare il minimo della funzione costo, le correzioni ai pesi sono fatte lungo una direzione di discesa; ad esempio, nella direzione opposta al gradiente:

$$\delta w_i^{(1)} = -\eta \frac{\partial E}{\partial w_i^{(1)}} \quad , \quad i = 1, 2, \dots, p \quad (11.4)$$

e

$$\delta w_j^{(2)} = -\eta \frac{\partial E}{\partial w_j^{(2)}} \quad , \quad j = 1, 2, \dots, m \quad (11.5)$$

dove η è chiamato *tasso di apprendimento* (learning rate constant).

Vediamo l'adattamento (stima) dei parametri relativi allo strato di uscita della rete:

$$\frac{\partial E}{\partial w_j^{(2)}} = \sum_{l=1}^m \frac{\partial E}{\partial s_l^{(2)}} \frac{\partial s_l^{(2)}}{\partial w_j^{(2)}} = (\text{essendo } s_l^{(2)}, \text{ per } l \neq j, \text{ indipendente da } w_j^{(2)}) = \frac{\partial E}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial w_j^{(2)}} \quad (11.6)$$

ed è immediato vedere che:

$$\frac{\partial s_j^{(2)}}{\partial w_j^{(2)}} = \begin{bmatrix} 1 \\ z \end{bmatrix} \quad (11.7)$$

mentre per l'altro fattore definiamo le variabili di comodo $\delta_j^{(2)}$, dette variabili *delta* per lo strato di uscita:

$$\begin{aligned}
\delta_j^{(2)} \equiv -\frac{\partial E}{\partial s_j^{(2)}} &= -\frac{1}{2} \frac{\partial \sum_{l=1}^m (d_l - y_l)^2}{\partial s_j^{(2)}} = -\frac{1}{2} \frac{\partial \sum_{l=1}^m (d_l - f(s_l^{(2)}))^2}{\partial s_j^{(2)}} \\
&= (\text{per linearit\'a della derivata}) = -\frac{1}{2} \sum_{l=1}^m \frac{\partial (d_l - f(s_l^{(2)}))^2}{\partial s_j^{(2)}} \\
&= (\text{essendo } s_l^{(2)} \text{ e } s_j^{(2)}, \text{ per } (l \neq j), \text{ indipendenti}) \\
&= (d_j - f(s_j^{(2)})) \frac{\partial f(s_j^{(2)})}{\partial s_j^{(2)}} \\
&= e_j f'(s_j^{(2)})
\end{aligned} \tag{11.8}$$

Quindi,

$$\frac{\partial E}{\partial w_j^{(2)}} = -\delta_j^{(2)} \begin{bmatrix} 1 \\ z \end{bmatrix} = -e_j f'(s_j^{(2)}) \begin{bmatrix} 1 \\ z \end{bmatrix} \tag{11.9}$$

e, in definitiva:

$$\begin{aligned}
w_j^{(2)}(k+1) &= w_j^{(2)}(k) - \eta \frac{\partial E}{\partial w_j^{(2)}} \\
&= w_j^{(2)}(k) + \eta e_j f'(s_j^{(2)}) \begin{bmatrix} 1 \\ z \end{bmatrix}.
\end{aligned} \tag{11.10}$$

Procedendo all'indietro, dall'uscita all'ingresso, vediamo l'adattamento (stima) dei parametri relativi allo strato di ingresso della rete. Qui c'\'e un problema fondamentale: l'errore di predizione dello strato di ingresso non \'e calcolabile direttamente, in quanto il suo valore esatto non \'e noto. Quindi, non \'e possibile aggiornare i pesi $W^{(1)}$ dello strato di ingresso in modo autonomo, ma \'e necessario farlo prendendo in considerazione anche lo strato di uscita. Cominciamo applicando la regola di derivazione di una funzione composta (*chain rule*):

$$\frac{\partial E}{\partial w_i^{(1)}} = \sum_{j=1}^m \sum_{q=1}^p \frac{\partial E}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial s_q^{(1)}} \frac{\partial s_q^{(1)}}{\partial w_i^{(1)}} \tag{11.11}$$

ed essendo

$$\frac{\partial s_q^{(1)}}{\partial w_i^{(1)}} = \begin{cases} 0 & q \neq i \\ [1 \ u]^T & q = i \end{cases} \tag{11.12}$$

si ha,

$$\frac{\partial E}{\partial w_i^{(1)}} = \sum_{j=1}^m \frac{\partial E}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial s_i^{(1)}} \frac{\partial s_i^{(1)}}{\partial w_i^{(1)}} \tag{11.13}$$

Ora (per convenzione $z_0 = 1$):

$$\begin{aligned}\frac{\partial s_j^{(2)}}{\partial s_i^{(1)}} &= \frac{\partial \sum_{l=0}^p (\gamma_{jl}^{(2)} z_l)}{\partial s_i^{(1)}} \\ &= \sum_{l=0}^p \frac{\partial (\gamma_{jl}^{(2)} f(s_l^{(1)}))}{\partial s_i^{(1)}} \\ &= \gamma_{ji}^{(2)} f'(s_i^{(1)})\end{aligned}\tag{11.14}$$

Definiamo le variabili-delta per lo strato all'ingresso, $\delta_i^{(1)}$:

$$\begin{aligned}\delta_i^{(1)} &= \sum_{j=1}^m \frac{\partial E}{\partial s_j^{(2)}} \frac{\partial s_j^{(2)}}{\partial s_i^{(1)}} \\ &= \sum_{j=1}^m -\delta_j^{(2)} \frac{\partial s_j^{(2)}}{\partial s_i^{(1)}} \\ &= \sum_{j=1}^m -\delta_j^{(2)} \gamma_{ji}^{(2)} f'(s_i^{(1)})\end{aligned}\tag{11.15}$$

Definiamo l'*errore di predizione propagato all'indietro* (back-propagation error):

$$e_i^{(1)} = \sum_{j=1}^m \delta_j^{(2)} \gamma_{ji}^{(2)}\tag{11.16}$$

esso rappresenta l'errore di predizione allo strato di ingresso, dovuto all'errore di predizione rilevato allo strato di uscita. Dunque:

$$\delta_i^{(1)} = -e_i^{(1)} f'(s_i^{(1)})\tag{11.17}$$

$$\frac{\partial E}{\partial w_i^{(1)}} = \delta_i^{(1)} \begin{bmatrix} 1 \\ u \end{bmatrix} = -e_i^{(1)} f'(s_i^{(1)}) \begin{bmatrix} 1 \\ u \end{bmatrix}\tag{11.18}$$

e, in definitiva:

$$\begin{aligned}w_i^{(1)}(k+1) &= w_i^{(1)}(k) - \eta \frac{\partial E}{\partial w_i^{(1)}} \\ &= w_i^{(1)}(k) + \eta e_i^{(1)} f'(s_i^{(1)}) \begin{bmatrix} 1 \\ u \end{bmatrix}.\end{aligned}\tag{11.19}$$

Nota: per le reti con strati interni, si generalizza la costruzione delle funzioni-delta utilizzando la stessa tecnica vista per lo strato di ingresso (generalized delta rule).

Vediamo un esempio sperimentale nel notebook `cap12_1_esempi_di_apprendimento_neurale.ipynb`

Ora, si sarebbe tentati dal sostituire il metodo del gradiente con un metodo che converga piú velocemente, come l'algoritmo di Newton o Gauss-Newton. In verità, il mondo nonlineare é fatto di situazioni a sé e in questo caso tale strategia, da sola, si rivelerebbe controproducente. Infatti, uno dei problemi principali nell'apprendimento neurale é la presenza di molti minimi locali (dovuti alla presenza delle funzioni di attivazione nonlineari in ogni neurone). Quindi, buoni risultati si ottengono con degli accorgimenti che evitano di finire in un minimo locale o almeno che permettono di uscirne e, per fare questo, é addirittura conveniente rallentare o non velocizzare troppo (!) la convergenza dei parametri W , come vediamo nella sottosezione 11.5.2.

11.3 Procedura di apprendimento, iper-parametri, convergenza

Sintetica descrizione dell'algoritmo: dato un insieme di esempi ingresso-uscita di numerositá K ,

1. selezione della struttura della rete: numero di neuroni e loro raggruppamento in strati;
2. inizializzazione: selezione pseudo-casuale dei valori iniziali dei parametri W , $k = 0$;
3. considero il k -esimo esempio: calcolo dell'errore di predizione e e della sua propagazione all'indietro $e^{(1)}$;
4. aggiornamento dei pesi $W^{(2)}$ e $W^{(1)}$;
5. se $k < K$ torno al passo 3;
6. se $\text{sum}(E(k)) > \text{toll}$ allora $k = 0$ e torno al passo 3;

11.4 Metodi del second'ordine: metodo di Newton

Vediamo se é possibile utilizzare qualche declinazione del metodo di Newton, come abbiamo visto al Capitolo 5, per l'apprendimento neurale. In linea generale, e quindi nell'utilizzo in vari tipi di problemi, non necessariamente nell'apprendimento neurale, esso risulta:

- veloce ed accurato
- costoso

come si puó verificare nel primo esempio:

`cap12_1_esempi_di_apprendimento_neurale_con_Gauss-Newton_1.ipynb`

Nello specifico delle reti neurali ci sono però alcune criticitá che non emergono nel caso generale:

- il malcondizionamento dell'Hessiana;
- minimi locali: non identificabilitá per simmetrie e scalamenti; si puó verificare se cé stato un accorciamento importante del gradiente (condizione necessaria ma non sufficiente: anche i punti di sella producono gradienti piccoli).

11.4.1 Regolarizzazione l_2 dei modelli neurali

Se l'Hessiana risulta mal-condizionata, si puó procedere aggiungendo una costante alla sua diagonale.

Si arriva all'algoritmo di Levenberg-Marquardt per le reti neurali, che però puó portare a passi di avanzamento estremamente piccoli per riuscire a regolarizzare sufficientemente il problema, da renderlo inefficiente rispetto al metodo del gradiente, come si puó verificare nell' esempio:
`cap12_1_esempi_di_apprendimento_neurale_con_Gauss-Newton_2.ipynb`.

Questo perché in presenza di punti di sella, il metodo di Newton puó prendere direzioni sbagliate.

Una soluzione possibile é modificare il metodo di newton in modo che eviti i punti di sella, ad esempio modificando l'Hessiana prendendo i valori assoluti degli autovalori.

11.5 Metodi del gradiente

In generale, un problema sono i minimi locali:

`cap12_1_esempi_di_apprendimento_neurale_con_Gauss-Newton_3.ipynb`

11.5.1 convergenza del gradiente stocastico

Il metodo del gradiente stocastico é basato sull' osservazione/dimostrazione che é possibile ottenere una stima non distorta (unbiased) del gradiente come media del gradiente ottenuto da sottoinsiemi estratti in modo i.i.d dalla distribuzione che genera gli esempi (in pratica si andrá ad operare sull'insieme di darti raccolto, eventualmente online).

Nota: con un batch di lunghezza pari a 1 é spesso necessario utilizzare un learning rate piccolo per mantenere la stabilitá, data l'elevata varianza (in questo caso) nella stima del gradiente.

11.5.2 Momentum Algorithm

Una considerazione iniziale va fatta sulla scelta di η : tale coefficiente influisce sulla velocità di convergenza ad un minimo ed anche sull'effettiva convergenza a tale minimo, basti ricordare il discorso fatto riguardo all'analogico coefficiente μ nel metodo damped-Newton (5.7). Tra l'altro, dato che qui si considera l'algoritmo della discesa più ripida (steepest-descent), è noto che la convergenza presenta pericolose oscillazioni. L'argomento è delicato ed esistono vari approcci proposti, che danno buoni risultati.

Più in generale, il problema della convergenza globale, di cui si è parlato al Capitolo 5, diventa qui particolarmente cruciale, in quanto i valori iniziali dei parametri W non sono affatto noti e vengono praticamente scelti come sequenza di numeri pseudo-casuali (notare che inizializzando a zero i pesi W l'algoritmo di apprendimento non parte). Di conseguenza, non avremo mai quella buona approssimazione iniziale dei parametri auspicata in un procedimento di ottimizzazione nonlineare. Inoltre, il percorso da fare per raggiungere i valori ottimi dei parametri può essere piuttosto lungo. E' dunque elevata la probabilità di trovare lungo questo percorso alcuni minimi locali che sono ben lontani dall'ottimo globale.

Per risolvere questo problema, che si presenta sovente in pratica, Rumelhart e McClelland (1986) hanno proposto il cosiddetto *Momentum Algorithm*:

cominciamo con il sostituire $E(k)$ con $E_m(k) = \sum_{r=1}^k E(r)$. Allora, con semplici passaggi si arriva ad esprimere (ponendo $x^{(0)} = u$ e $x^{(1)} = z$)

$$-\frac{\partial E_m(k)}{\partial w_i^{(q)}} = \delta_i^{(q)}(k) \left[\begin{array}{c} 1 \\ x^{(q-1)}(k) \end{array} \right] + \sum_{r=1}^{k-1} \delta_i^{(q)}(r) \left[\begin{array}{c} 1 \\ x^{(q-1)}(r) \end{array} \right] \quad (11.20)$$

e, dato che

$$-\frac{\partial E_m(k-1)}{\partial w_i^{(q)}} = \sum_{r=1}^{k-1} \delta_i^{(q)}(r) \left[\begin{array}{c} 1 \\ x^{(q-1)}(r) \end{array} \right] \quad (11.21)$$

si ha:

$$-\frac{\partial E_m(k)}{\partial w_i^{(q)}} = \delta_i^{(q)}(k) \left[\begin{array}{c} 1 \\ x^{(q-1)}(k) \end{array} \right] - \frac{\partial E_m(k-1)}{\partial w_i^{(q)}} \quad (11.22)$$

Ora, introducendo un coefficiente $0 < \alpha < 1$, si è di fatto introdotto un meccanismo di inerzia nella variazione della direzione di aggiornamento dei parametri:

$$-\frac{\partial E_m(k)}{\partial w_i^{(q)}} = \delta_i^{(q)}(k) \left[\begin{array}{c} 1 \\ x^{(q-1)}(k) \end{array} \right] - \alpha \frac{\partial E_m(k-1)}{\partial w_i^{(q)}} \quad (11.23)$$

Si chiama *inerzia* perché funziona proprio come l'inerzia in meccanica: arrivato al minimo locale, se questo non è sufficientemente profondo, riesce ad uscirne ed a proseguire verso altri minimi, finché non raggiunge il minimo globale (oppure un minimo locale sufficientemente attrattivo).

11.5.3 Adattamento del learning rate

Il learning va modificato nel corso della procedura di learning. Ad esempio, é spesso utile diminuirlo a causa del rumore prodotto dallo stimatore.

Esistono delle tecniche euristiche per imporre a-priori una legge di adattamento, oppure per adattare il suo valore ad ogni iterazione, in base a degli indicatori. Non vi é dimostrazione di quale tra le tecniche di adattamento sia quella ottimale (es. in TensorFlow di Google é preferito ADAMS).

11.6 Metodi per le reti convoluzionali

Un limite fondamentale delle reti neurali con neuroni del tipo visto finora é con dati organizzati a griglia, in cui é presente una correlazione spaziale o temporale. Tipicamente, ogni esperimento presenta una grande mole di dati di questo tipo (es. pixels in una foto o valori in una serie temporale, ad esempio da un segnale audio) e questo crea un layer di ingresso alla rete di dimensione tale da creare un numero enorme di parametri nella rete. Le reti convoluzionali sono una buona risposta nel mantenere trattabile (seppur elevato) il numero di parametri. Hanno poi dimostrato anche altre qualità e sono pertanto uno strumento principale nella modellazione di questo tipo di dati con reti neurali.

La struttura piú usata prevede un'insieme di strati convoluzionali collegati all'ingresso, seguiti da alcuni strati come quelli visti finora.

Riprendiamo la formula della convoluzione discreta 2.14 in termini piu' generali, ovvero consideriamo la convoluzione discreta tra due vettori (non necessariamente considerati sequenza di ingresso e risposta all'impulso discreto di un sistema DLTI, come nel caso della 2.14). La componente $k - \text{esima}$ della convoluzione discreta tra i vettori c e d , é:

$$(c * d)_k = \sum_{i+j=k} c_j d_i = \sum_j c_j d_{k-j} \quad (11.24)$$

Nota: c'è un legame molto interessante tra la somma di convoluzione discreta e i polinomi. Se c e d sono i vettori dei coefficienti di due polinomi $p_c(x)$ e $p_d(x)$, di grado n , e consideriamo il prodotto dei due polinomi

$$p_c(x) \cdot p_d(x) = (c_0 + c_1x + c_2x^2 + \dots c_nx^n) (d_0 + d_1x + d_2x^2 + \dots d_nx^n)$$

, il coefficiente del monomio di grado k nel polinomio prodotto é dato dalla 11.24:

$$(p_c(x) \cdot p_d(x))_k = \sum_j c_j x^j d_{k-j} x^{k-j} = \left(\sum_j c_j d_{k-j} \right) x^k$$

Ancora, questo prodotto di polinomi diventa equivalente alla risposta del sistema DLTI se consideriamo la Trasformata Zeta delle sequenze di ingresso e di risposta all'impulso discreto.

Nella terminologia delle reti neurali convoluzionali, le due funzioni di cui si fa la convoluzione discreta sono dette “ingresso” (input) e “nucleo” (kernel).

Inoltre, nelle reti convoluzionali, i dati in ingresso sono spesso multidimensionali. Ad esempio, se diamo in ingresso un’immagine I , ovvero una sequenza bidimensionale, dobbiamo pensare anche ad una convoluzione e ad un kernel K bi-dimensionali:

$$(I * K)_{i,j} = \sum_m \sum_n I(m,n)K(i-m, j-n) \quad (11.25)$$

Nota: molte librerie di machine learning parlano di convoluzione ma in realtà implementano la *mutuacorrelazione*:

$$(I * K)_{i,j} = \sum_m \sum_n I(i+m, j+n)K(m,n) \quad (11.26)$$

Oltre alla convoluzione, che è un’operatore lineare, le reti neurali convoluzionali applicano alcuni operatori anche nonlineari:

- pooling

11.6.1 Applicazioni

- classificazione di serie temporali a scopo predittivo (esempio: durata di un processo di asciugatura). Recentemente si parla di “Temporal Convolutional Networks”.
- riconoscimento di features in spetrogrammi, con applicazione alla meccanica delle vibrazioni.
- ricostruzione di mappe nonlineari: ad esempio, della mappa tra i dati del problema nel continuo (coefficienti delle equazioni differenziali del modello, condizioni iniziali, condizioni al contorno) ed i coefficienti/parametri di un modello di ordine ridotto. Per la costruzione del modello di ordine ridotto possiamo utilizzare una delle tecniche del Capitolo ??, in particolare la tecnica POD. Questa applicazione è di fondamentale importanza nella costruzione di “digital twins”.

Appendice A

Analisi dei sistemi DLTI *state-space*

Consideriamo il sistema lineare discreto (2.26)-(2.23) nello spazio degli stati. Le variabili del vettore di stato x , che possono essere scelte in base ad un significato fisico evidente oppure in base a considerazioni di tipo matematico o statistico inferenziale, godono però in ogni caso della **proprietà di separazione**: *i valori assunti dalle variabili del vettore di stato ad ogni istante di tempo, contengono complessivamente tutta l'informazione relativa alla storia passata del sistema e sufficiente a determinare l'andamento futuro delle variabili di stato e di uscita, se è noto l'andamento delle variabili di ingresso negli istanti futuri.* Notare che, se ci mettiamo in un contesto probabilistico, c'è uno stretto legame tra la rappresentazione di un sistema DLTI nello spazio degli stati ed una *catena di Markov*.

L'evoluzione dello stato può essere scomposta nella somma di un'evoluzione libera e di un'evoluzione forzata. Infatti, la soluzione $\mathcal{T}(u, x_0, k)$ dell'equazione di stato, detta anche **traiettoria**, è data da:

$$\mathcal{T}(u, x_0, k) = A^k x_0 + \sum_{j=0}^{k-1} A^{k-1-j} B u(j) \quad , \quad k \geq 0 \quad (\text{A.1})$$

Quando si scrivono le equazioni di un sistema nello spazio degli stati, è necessario aver scelto una base per lo spazio degli stati, quello degli ingressi e quello delle uscite, rispetto alla quale scrivere le relazioni tra le variabili di stato, di ingresso e di uscita, ovvero le matrici A , B , C e D . Tale scelta è totalmente arbitraria, anche se può essere suggerita spesso da considerazioni di tipo fisico.

Detta e_1, e_2, \dots, e_n la base dello spazio degli stati usata per scrivere il sistema (2.26)-(2.23), consideriamo una nuova base v_1, v_2, \dots, v_n , in cui ogni v_i è

combinazione lineare degli e_i :

$$v_j = \sum_{i=1}^n t_{i,j} e_i \quad , \quad j = 1, \dots, n \quad .$$

Detta $T = \{t_{i,j}\}$ la matrice di cambiamento di base, il vettore di componenti rispetto alla nuova base è dato da:

$$\bar{x} = T^{-1}x \tag{A.2}$$

ed il sistema, diventa:

$$\begin{aligned} \bar{x}(k+1) &= T^{-1}x(k+1) = T^{-1}AT\bar{x}(k) + T^{-1}Bu(k) = \bar{A}\bar{x}(k) + \bar{B}u(k) \\ y(k) &= CT\bar{x}(k) + Du(k) = \bar{C}\bar{x}(k) + Du(k) \end{aligned} \tag{A.3}$$

ed è *algebricamente equivalente* a (2.26)-(2.23), avendo posto:

$$\begin{aligned} \bar{A} &= T^{-1}AT \\ \bar{B} &= T^{-1}B \\ \bar{C} &= CT \end{aligned} \tag{A.4}$$

Notare che la trasformazione effettuata alla matrice A è una trasformazione per similitudine, per cui gli autovalori rimangono gli stessi, e dunque la dinamica evolutiva dello stato rimane immutata. Di fatto, si è cercata una diversa rappresentazione per descrivere la stessa dinamica. L'analisi di questa dinamica viene affrontata nella sezione seguente.

A.1 Analisi modale

Con la trasformazione di base appena introdotta, è possibile scegliere diverse rappresentazioni del sistema (2.26)-(2.23), che magari mettano in evidenza alcune proprietà notevoli del sistema. Una di queste la si ha scegliendo come base v_1, v_2, \dots, v_n una base di Jordan per la matrice A , rispetto alla quale, la matrice \bar{A} risulta espressa in *forma canonica di Jordan*. In tal caso, risultano evidenti i *modi* del sistema, ovvero l'insieme di funzioni che caratterizzano l'evoluzione libera del sistema (2.26)-(2.23), da cui tale operazione prende il nome di *analisi modale*. Consideriamo qui solo il caso in cui A sia diagonalizzabile. Otteniamo:

$$\bar{x}(k) = \bar{A}^k \bar{x}(0) = \begin{bmatrix} \lambda_1^k \bar{x}_1(0) \\ \lambda_2^k \bar{x}_2(0) \\ \vdots \\ \lambda_n^k \bar{x}_n(0) \end{bmatrix} \tag{A.5}$$

A questo punto, l'analisi modale procede ad individuare il carattere di convergenza dei modi ed in particolare:

- la stabilità dell'evoluzione libera
- i modi dominanti.

In generale, l'evoluzione libera rispetto ad una base qualsiasi sarà una combinazione lineare dei modi del sistema.

A.2 Raggiungibilità

La *raggiungibilità* ci permette di stabilire quanto sia possibile modificare il vettore di stato x agendo sul vettore di ingresso u . Tale concetto coinvolge solamente l'equazione di stato.

Definizione: "Dato un sistema state-space $\Sigma = (A, B, \cdot, \cdot)$, uno stato \bar{x} è *raggiungibile dallo stato zero* (o *raggiungibile*) se esiste una sequenza di ingresso $\bar{u}(k)$, ad energia finita, ed un istante discreto $\bar{k} < \infty$, tale che

$$\bar{x} = \mathcal{T}(\bar{u}(k), 0, \bar{k}) .$$

" Il sottospazio *raggiungibile* $\mathcal{X}_r \subset \mathcal{X}$ di Σ è l'insieme contenente tutti gli stati raggiungibili di Σ . Il sistema Σ è (completamente) *raggiungibile* se $\mathcal{X}_r = \mathcal{X}$. Inoltre,

$$\mathcal{R}(A, B) = [B \ AB \ A^2B \ \dots \ A^iB \ \dots]$$

è detta *matrice di raggiungibilità* di Σ ".

Per il teorema di Cayley-Hamilton [19] [13], il rango della matrice di raggiungibilità e lo spazio generato dalle sue colonne sono determinati al più dai primi n termini A^iB , e quindi in pratica ci si riferisce alla matrice di raggiungibilità in n passi:

$$\mathcal{R}_n(A, B) = [B \ AB \ A^2B \ \dots \ A^iB \ \dots \ A^{n-1}B]$$

Il risultato fondamentale riguardante la raggiungibilità è quindi:

$$\mathcal{X}_r = \text{range}(\mathcal{R}_n(A, B)) . \quad (\text{A.6})$$

Notare che:

- la raggiungibilità è un concetto analitico ma, parlando di sistemi DLTI finito-dimensional, essa equivale ad un concetto *algebrico* che dipende unicamente da $\mathcal{R}_n(A, B)$, e quindi da A e B , ma non dal tempo o dagli ingressi u ; la conseguenza di questo fatto, e di fatti ad esso correlati (cfr. ad esempio l'*osservabilità*), è che gli strumenti per l'analisi di tali sistemi dinamici provengono dall'algebra, in particolare l'algebra lineare, e vengono realizzati mediante algoritmi di algebra lineare numerica.
- lo spazio generato dalle colonne di $\mathcal{R}_n(A, B)$ viene detto *spazio di raggiungibilità* nella comunità di teoria dei sistemi, mentre è detto comunemente *spazio di Krylov* dagli analisti numerici.

A.3 Osservabilità

Definizione [19]: "Dato un sistema state-space $\Sigma = (A, B, C, \cdot)$, due stati \bar{x}^I e \bar{x}^{II} sono *indistinguibili nel futuro in k passi* se per ogni sequenza di ingresso $u(i)$, $i = 0, \dots, k - 1$, le rispettive sequenze di uscita $y^I(k)$ ed $y^{II}(k)$, ottenute a partire dagli stati iniziali \bar{x}^I e \bar{x}^{II} , coincidono nei primi k passi".

Scrivendo esplicitamente le espressioni delle uscite $y^I(k)$ ed $y^{II}(k)$, si vede facilmente che la condizione di indistinguibilità riguarda esclusivamente le risposte libere del sistema e corrisponde alla seguente relazione algebrica:

$$\bar{x}^I - \bar{x}^{II} \in \ker \begin{bmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ \vdots \\ C \cdot A^k \end{bmatrix} \quad (\text{A.7})$$

Per il teorema di Cayley-Hamilton,

$$\ker \begin{bmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ \vdots \\ C \cdot A^k \end{bmatrix} = \ker \begin{bmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ \vdots \\ C \cdot A^{n-1} \end{bmatrix} \quad k \geq n - 1$$

e dunque due stati sono indistinguibili nel futuro (per ogni k) se lo sono in $n - 1$ passi.

Definizione [19]: "Uno stato \bar{x} si dice *non osservabile* se è indistinguibile nel futuro dallo stato zero."

Quindi la teoria dei sistemi richiede una semplice verifica algebrica per garantire che il sistema di ordine n (e quindi $A \in \mathbb{R}^{n \times n}$) sia osservabile: la matrice di osservabilità:

$$O = \begin{bmatrix} C \\ C \cdot A \\ C \cdot A^2 \\ \vdots \\ C \cdot A^{n-1} \end{bmatrix} \quad (\text{A.8})$$

deve avere rango n . Questa condizione, facile da verificare anche mediante conti fatti a mano, non è però numericamente stabile e di fatto si creano problemi già per sistemi di ordine > 10 (dovuti principalmente all'instabilità numerica del calcolo delle potenze di A).

Per una verifica numericamente efficace della raggiungibilità e dell'osservabilità di un sistema, sono stati sviluppati negli ultimi anni dei criteri che utilizzano

come operazioni fondamentali le trasformazioni ortogonali, in particolare le matrici di Givens ed Householder, già viste in questo corso, con cui costruire una trasformazione ortogonale H dello spazio degli stati, invece che semplicemente invertibile. L'obbiettivo di queste trasformazioni è quello di raggiungere una forma in cui l'osservabilità può essere verificata per semplice ispezione (ad esempio la *staircase form*).

In particolare, in questo caso è necessario fare attenzione a mantenere inalterate le proprietà dinamiche del sistema trasformato. Per ottenere questo è sufficiente operare *trasformazioni per similitudine* della matrice A , che nel caso di matrici di trasformazione ortogonali, diventano del tipo:

$$A_t = H^T \cdot A \cdot H \quad , \quad B_t = H^T \cdot B \quad (\text{A.9})$$

A.4 Realizzazione minima

Definizione: " (A, B, C, D) è detta **realizzazione** della sequenza $\{G_k\}_{k=0}^{\infty}$ se vale la (2.25)".

Definizione: "La realizzazione (A, B, C, D) è **minima** se l'ordine del modello è il minimo".

Teorema (Kalman): *Una realizzazione è minima se e solo se è raggiungibile ed osservabile.*

Notare che raggiungibilità ed osservabilità sono duali, nel senso che una realizzazione (A, B, C, D) è osservabile se e solo se la realizzazione (A^T, C^T, B^T, D) è raggiungibile e viceversa.

Vediamo ora dunque il calcolo numerico della realizzazione minima, al par. 9.2.2.

Appendice B

Soluzione numerica di problemi differenziali evolutivi

Tratteremo qui alcuni esempi notevoli di equazioni differenziali lineari alle derivate parziali ed evolutive, e di modelli dinamici a parametri concentrati.

Il primo riguarda la modellazione di fenomeni diffusivi (equazione del calore), il secondo la modellazione di fenomeni di trasporto e propagazione di onde (equazione delle onde) e il terzo la modellazione a parametri concentrati di sistemi meccanici.

B.1 Fenomeni diffusivi

Consideriamo il seguente problema ai limiti ed ai valori iniziali; trovare $u(x, t)$ tale che:

$$\begin{cases} \frac{\partial u}{\partial t} - \frac{\partial^2 u}{\partial x^2} = f(x, t) & , \quad 0 \leq x \leq 1 & , \quad t \geq 0 \\ u(x, 0) = g(x) & , \quad 0 \leq x \leq 1 \\ u(0, t) = \phi_0(t) & , \quad u(1, t) = \phi_1(t) & , \quad t \geq 0 \end{cases} \quad (\text{B.1})$$

Approssimiamo la soluzione $u(x, t)$ mediante il metodo delle differenze finite. A tale scopo, scegliamo un intero positivo d e definiamo una griglia rettangolare $\{(i\Delta x, k\Delta t) \quad , \quad i = 0, 1, \dots, d+1 \quad , \quad k \geq 0\}$, dove $\Delta x = 1/(d+1)$ e $\Delta t > 0$.

Ora applichiamo il metodo di *Eulero esplicito*, ovvero sostituiamo la derivata seconda nello spazio e la derivata prima nel tempo, rispettivamente, con le differenze finite *centrate* e le differenze finite *in avanti*:

$$\frac{\partial^2 u(x, t)}{\partial x^2} = \frac{1}{(\Delta x)^2} [u(x - \Delta x, t) - 2u(x, t) + u(x + \Delta x, t)] + \mathcal{O}((\Delta x)^2) \quad , \quad \Delta x \rightarrow 0 \quad (\text{B.2})$$

$$\frac{\partial u(x, t)}{\partial t} = \frac{1}{\Delta t} [u(x, t + \Delta t) - u(x, t)] + \mathcal{O}(\Delta t) \quad , \quad \Delta t \rightarrow 0 \quad (\text{B.3})$$

ed arriviamo al seguente modello discreto:

$$u_{\Delta x}(k+1) = A_{\Delta x}u_{\Delta x}(k) + b_{\Delta x}(k) \quad , \quad k = 0, 1, \dots \quad (\text{B.4})$$

ovvero all'equazione di aggiornamento dello stato di un sistema DLTI, in cui le componenti del vettore di stato $u_{\Delta x}(k)$ hanno il significato di concentrazioni della grandezza fisica che diffonde (temperature, nel caso si stia considerando un problema di conduzione del calore), il vettore $b_{\Delta x}(k)$ contiene i contributi del termine forzante e delle condizioni al contorno, e la matrice di aggiornamento dello stato risulta:

$$A_{\Delta x} = \begin{bmatrix} 1 - 2\mu & \mu & 0 & \dots & 0 \\ \mu & 1 - 2\mu & \mu & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & 0 \\ \vdots & \ddots & \mu & 1 - 2\mu & \mu \\ 0 & \dots & 0 & \mu & 1 - 2\mu \end{bmatrix}$$

dove $\mu = \frac{\Delta t}{(\Delta x)^2}$ é detto *numero di Courant*.

Notare che:

- il passo Δx determina la dimensione del vettore di stato e dunque l'ordine del sistema DLTI;
- l'errore di troncamento locale risultante, $\mathcal{O}((\Delta x)^2, \Delta t)$, ottenuto quando $\Delta x \rightarrow 0$ $\Delta t \rightarrow 0$ mantenendo μ pressoché costante, é effettivamente un *rumore di modello*.

Ora, gli autovalori della matrice $A_{\Delta x}$ sono dati da:

$$1 - 4\mu \sin^2[\pi j / (d+1)] \quad , \quad j = 1, 2, \dots, d \quad (\text{B.5})$$

da cui il raggio spettrale risulta: $\rho(A_{\Delta x}) = |1 - 4\mu|$

e quindi il sistema é instabile per $\mu > 1/2$, ovvero il metodo di Eulero esplicito é solo condizionatamente stabile.

B.2 Fenomeni di trasporto e propagazione di onde

Consideriamo l'equazione del trasporto:

$$\begin{cases} \frac{\partial u}{\partial t} + a \frac{\partial u}{\partial x} = 0 & , \quad 0 \leq x \leq 1 & , \quad t \geq 0 & , \quad a > 0 \\ u(x, 0) = g(x) & , \quad 0 \leq x \leq 1 \\ u(0, t) = \phi_0(t) & , \quad t \geq 0 \end{cases} \quad (\text{B.6})$$

le cui soluzioni viaggiano lungo le linee caratteristiche a velocità costante, senza modificarsi.

L'equazione delle onde 1D è formata da due fenomeni di trasporto che viaggiano in direzioni opposte:

$$\begin{cases} \frac{\partial^2 u}{\partial t^2} - \frac{\partial^2 u}{\partial x^2} = 0 & , \quad 0 \leq x \leq 1 & , \quad t \geq 0 \\ u(x, 0) = g_0(x) & , \quad 0 \leq x \leq 1 \\ \frac{\partial u(x, 0)}{\partial t} = g_1(x) & , \quad 0 \leq x \leq 1 \\ u(0, t) = \phi_0(t) & , \quad u(1, t) = \phi_1(t) & , \quad t \geq 0 \end{cases} \quad (\text{B.7})$$

Discretizzando la derivata spaziale analogamente a quanto fatto al par. B.1 per l'equazione del calore, otteniamo un sistema di equazioni differenziali lineari ordinarie del tipo (B.8).

E' importante osservare che i modelli differenziali di natura iperbolica, come le equazioni del trasporto e delle onde, presentano un *dominio di dipendenza* dalle condizioni iniziali locale e che invade progressivamente tutto o solo in parte il dominio del problema, seguendo la direzione di propagazione del fronte d'onda.

Nota: nelle applicazioni si parla di onde e di dominio di dipendenza anche per i problemi parabolici, ad esempio di *onde termiche* (*thermal waves*) per i fenomeni di trasmissione del calore, in quanto la soluzione fondamentale ha velocità infinita, ma decade esponenzialmente, quindi in un modello discreto che considera dati quantizzati di fatto la velocità risulta essere finita.

B.3 Soluzione numerica delle equazioni della dinamica di un sistema meccanico elastico lineare

B.3.1 Integrazione nel tempo

In generale, integrare nel tempo le equazioni di moto di un sistema meccanico elastico lineare significa risolvere numericamente il seguente sistema di equazioni

differenziali ordinarie:

$$\begin{aligned} M\ddot{d}(t) + G\dot{d}(t) + Kd(t) &= f(t) \\ y(t) &= Hd(t) \end{aligned} \tag{B.8}$$

dove le matrici M , G e K descrivono, rispettivamente, la massa, lo smorzamento e la rigidezza delle componenti del sistema fisico, e $f_h(t)$ rappresenta il vettore di forze esterne (il termine forzante) e supponiamo dunque che le misure sperimentali siano esprimibili mediante combinazioni lineari degli spostamenti $d(t)$ descritte dalla matrice H .

Il significato fisico della soluzione può essere molto differente da caso a caso e dipende dalla formulazione del modello utilizzato. Ad esempio, l'equazione (B.8) può descrivere le piccole deformazioni attorno ad un punto di equilibrio per un sistema elastico continuo (nello spazio) vincolato da adeguate condizioni al contorno, e dunque il modello utilizzato è tipicamente una discretizzazione agli elementi finiti delle equazioni dell'elasticità lineare. Oppure, può descrivere il moto nello spazio 3D di un sistema di corpi rigidi, e dunque il modello utilizzato è tipicamente l'assemblamento di componenti a parametri concentrati (es. un modello *multi-body*). Ovviamente esistono moltissime varianti e combinazioni.

I metodi numerici per l'integrazione nel tempo di questo sistema di equazioni differenziali possono essere suddivisi in due classi:

- metodi che risolvono il sistema del second'ordine nel tempo;
- metodi che trasformano il sistema (B.8) in un equivalente sistema del primo ordine.

Ognuna di queste classi comprende metodi *impliciti* e metodi *esplicativi*. I metodi impliciti sono in generale più onerosi dal punto di vista computazionale ma incondizionatamente stabili, mentre i metodi esplicativi sono più leggeri ma se non sono garantite le condizioni di stabilità possono produrre soluzioni numeriche aberranti.

Nel caso in cui il sistema di equazioni differenziali risultì *stiff*, gli unici metodi numerici applicabili risultano quelli impliciti.

B.3.2 Metodi per sistemi di equazioni differenziali del primo ordine nel tempo

Vediamo innanzitutto la riformulazione di (B.8) come sistema del primo ordine nel tempo:

$$\dot{x}(t) = A_c x(t) + B_c u(t) = f(x(t), u(t)) \tag{B.9}$$

$$y(t) = C_c x(t) + D_c u(t) \tag{B.10}$$

dove $x(t) = \begin{bmatrix} \dot{d}(t) \\ d(t) \end{bmatrix}$ é detto anche *vettore di stato*, $u(t) = \begin{bmatrix} f(t) \\ 0 \end{bmatrix}$ é detto anche *vettore degli ingressi*, e:

$$A_c = \begin{bmatrix} -M^{-1}G & -M^{-1}K \\ I & 0 \end{bmatrix}, \quad B_c = \begin{bmatrix} M^{-1} \\ 0 \end{bmatrix}$$

$$C_c = [0 \quad H], \quad D_c = 0$$

Tra i metodi possibili per discretizzare la (B.9) nel tempo, indichiamo il *θ -metodo* [23] ed i metodi *multi-step*. Detto τ il passo temporale (o periodo di campionamento) ed $x(k) = x(k\tau)$, il θ -metodo approssima la derivata nel tempo con il rapporto incrementale e sostituisce il membro destro con una media pesata (da $0 \leq \theta \leq 1$) dei suoi valori agli istanti k e $k+1$:

$$\frac{x(k+1) - x(k)}{\tau} = [(1-\theta)f(x(k), u(k)) + \theta f(x(k+1), u(k+1))] \quad (\text{B.11})$$

B.3.2.1 Eulero esplicito (forward Euler, $\theta = 0$)

Si ha:

$$x(k+1) = x(k) + \tau f(x(k), u(k)) = x(k) + \tau(A_c x(k) + B_c u(k))$$

$$x(k+1) = (I + \tau A_c)x(k) + \tau B_c u(k) \quad (\text{B.12})$$

B.3.2.2 Eulero implicito (backward Euler, $\theta = 1$)

Si ha:

$$x(k+1) = x(k) + \tau f(x(k+1), u(k+1)) = x(k) + \tau(A_c x(k+1) + B_c u(k+1))$$

$$(I - \tau A_c)x(k+1) = x(k) + \tau B_c u(k+1) \quad (\text{B.13})$$

B.3.2.3 Crank-Nicholson ($\theta = \frac{1}{2}$)

Si ha:

$$\begin{aligned} x(k+1) &= x(k) + \frac{1}{2}\tau [f(x(k), u(k)) + f(x(k+1), u(k+1))] \\ &= x(k) + \frac{1}{2}\tau [A_c(x(k) + x(k+1)) + B_c(u(k) + u(k+1))] \end{aligned}$$

$$(I - \frac{1}{2}\tau A_c)x(k+1) = (I + \frac{1}{2}\tau A_c)x(k) + \frac{1}{2}\tau B_c(u(k) + u(k+1)) \quad (\text{B.14})$$

Riprendiamo ora l'esempio delle tre masse: C.9.

Parte I

Computing Lab

Introduction

...

Appendice C

Dati quantitativi e modelli

```
In [1]: %matplotlib inline
# Effettuiamo il caricamento del modulo "numpy", assegnandogli un nome a scelta;
# ad es. qui "np"; digita help("import") per vedere i dettagli di questo comando.
# il modulo "numpy" contiene le routine per la manipolazione di vettori e matrici,
# e l'implementazione dei principali algoritmi di algebra lineare numerica.
# Digita help(numpy) per vedere una (lunga) descrizione di questo modulo.
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
from libreria_NLALD.sistema_meccanico_3gdl import *
from libreria_NLALD.SistemaMeccanico3gdl import *

M1 = 27.0
M2 = 120.0
M3 = 11246.0
K1 = 180000000.0
K2 = 60000000.0
K3 = 120000000.0
C1 = 50000.0
C2 = 46000.0
C3 = 240000.0
coordinate di partenza: y1=0.06, y2=0.06, y3=0.05
```

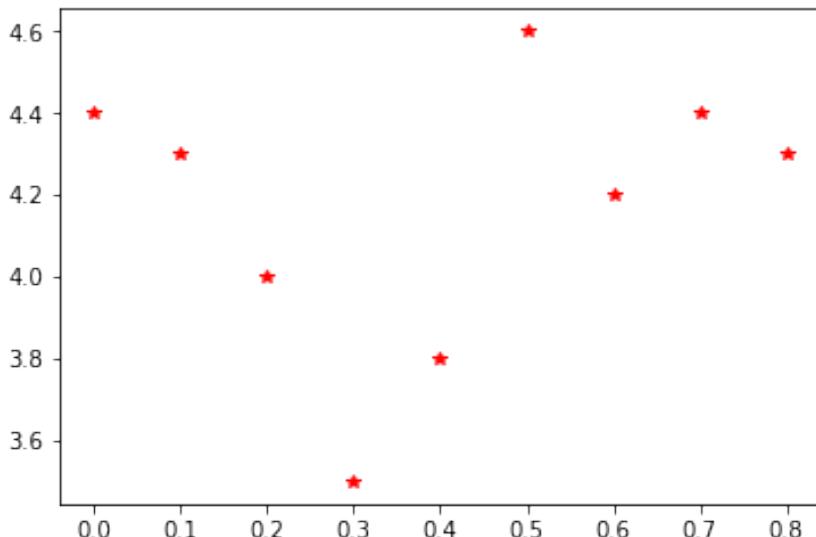
C.1 Segnale discreto mono-dimensionale

```
In [2]: # Un segnale discreto 1-dimensionale viene rappresentato con un vettore:
s = np.array([4.4,4.3,4.0,3.5,3.8,4.6,4.2,4.4,4.3])
```

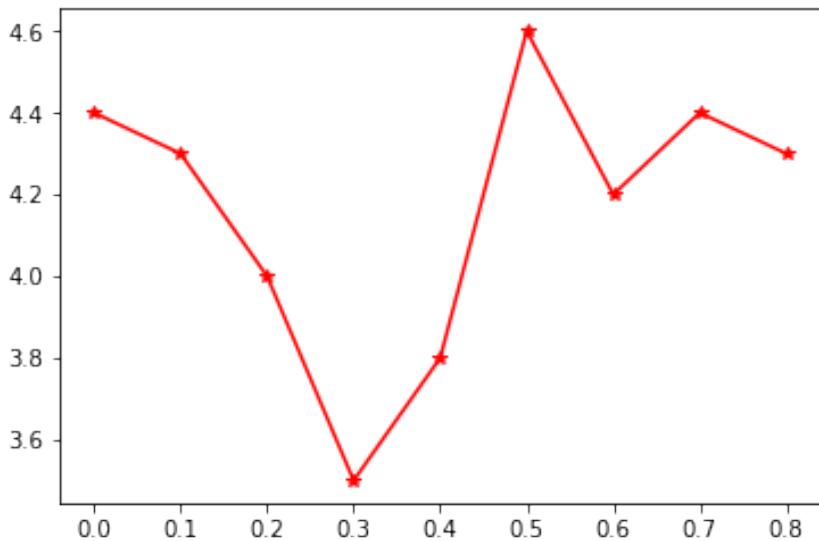
```
In [3]: # se esso è stato campionato con periodo uniforme Tc secondi, allora è utile  
# creare un vettore con gli istanti di campionamento:  
Tc = 0.1;  
t = np.arange(0.0,np.size(s)*Tc,Tc) # digita "help(np.arange)" per i dettagli  
print("t = ",t) # per vedere il contenuto di una variabile utilizzare "print".
```

```
t = [0. 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8]
```

```
In [4]: # per fare il grafico del segnale:  
plt.figure(1); plt.plot(t,s,'r*'); plt.show()
```



```
In [5]: # spesso si collegano i campioni della sequenza di dati con una linea spezzata,  
# per capirne meglio l'andamento: ricordarsi però che la sequenza (discreta)  
# è definita solo nell'insieme di punti e non nel continuo:  
plt.figure(2); plt.plot(t,s,'r-*'); plt.show()
```



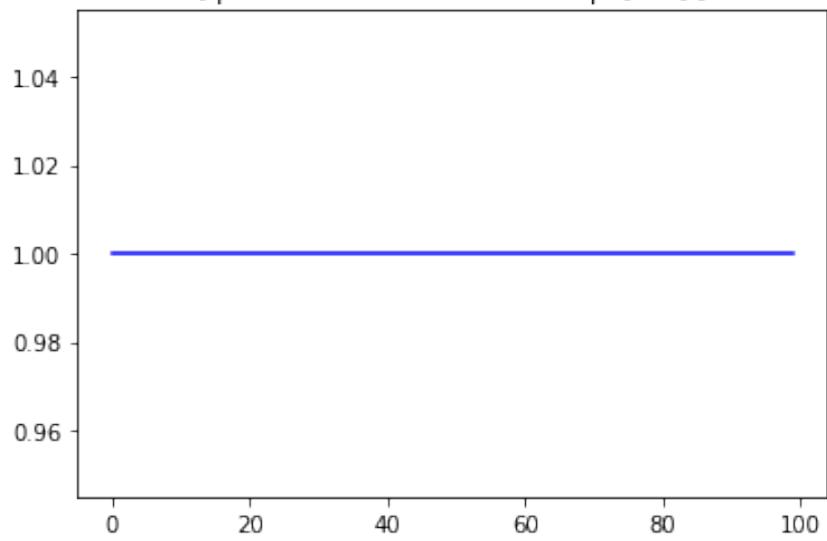
```
In [6]: # Quando la sequenza di dati è indicizzata dalla variabile tempo, viene chiamata
# "serie temporale" o "segnale a tempo discreto".
# è possibile generare un arbitrario segnale discreto deterministico:
sd = np.sin(2*np.pi*1.5*t) + 2*np.sin(2*np.pi*4*t);
# o stocastico:
ss = np.sin(2*np.pi*t) + np.random.randn(len(t));
# la funzione "randn()" genera sequenze di dati pseudo-casuali secondo una
# distribuzione gaussiana normalizzata.
```

Ritorniamo al par. 7.1.

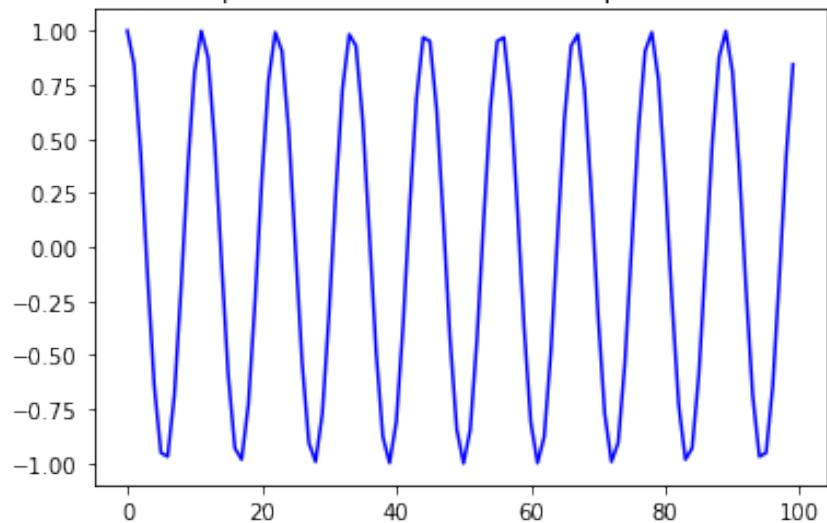
C.2 Oscillazioni dell'esponenziale discreto

```
In [7]: # Vediamo le oscillazioni dell'esponenziale discreto al variare della
# pulsazione discreta "omega_ni":
N = 100;
for ni in range(0,N,9):
    omega_ni = 2*np.pi*ni/N;
    wk = np.array([0.+(omega_ni*(0.+1.j))]) * np.arange(0,N)
    plt.figure(ni); plt.plot(np.real(np.exp(wk)), 'b-')
    plt.title(['pulsazione discreta = ' + str(omega_ni/np.pi) + ' * pi [rad]']);
    plt.show()
#endif
```

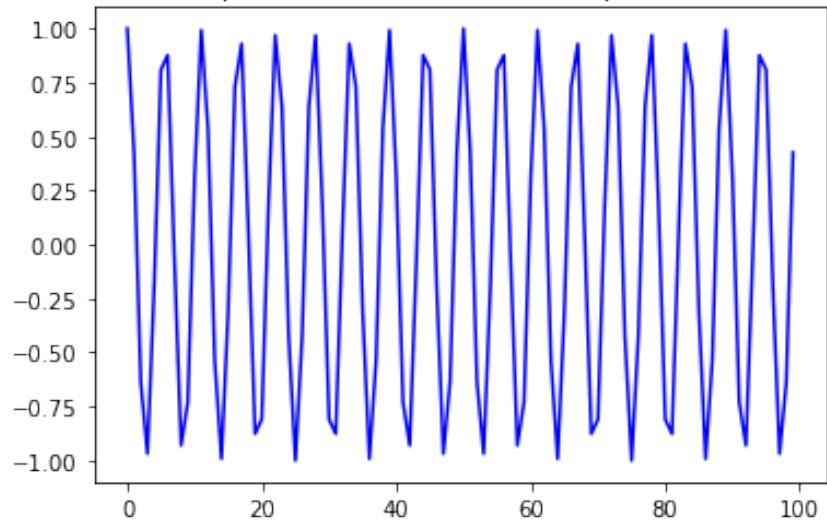
['pulsazione discreta = 0.0 * pi [rad]']



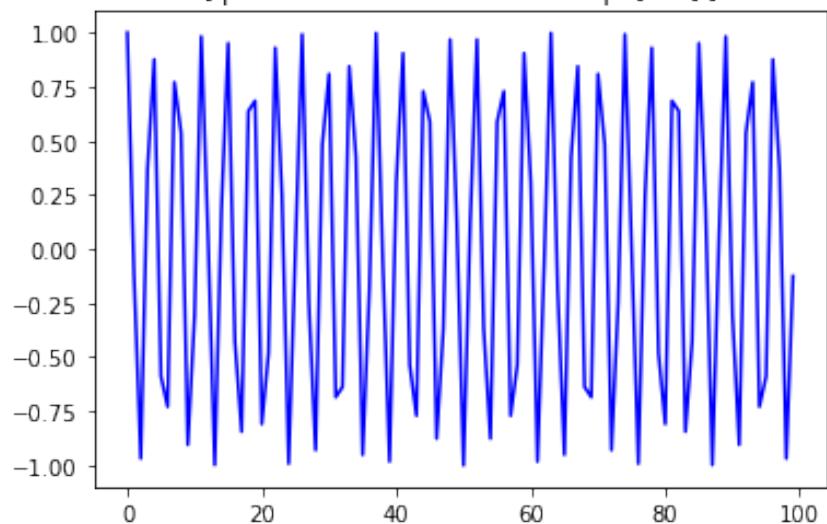
['pulsazione discreta = 0.18 * pi [rad]']



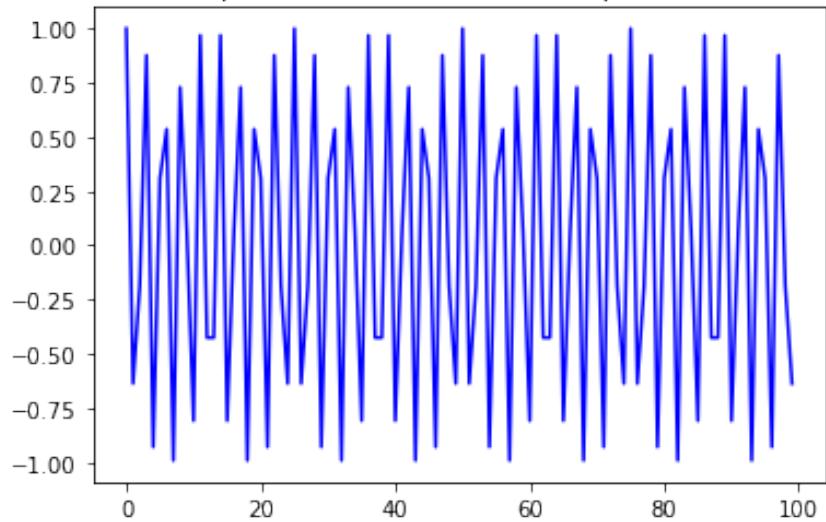
['pulsazione discreta = 0.36 * pi [rad]']



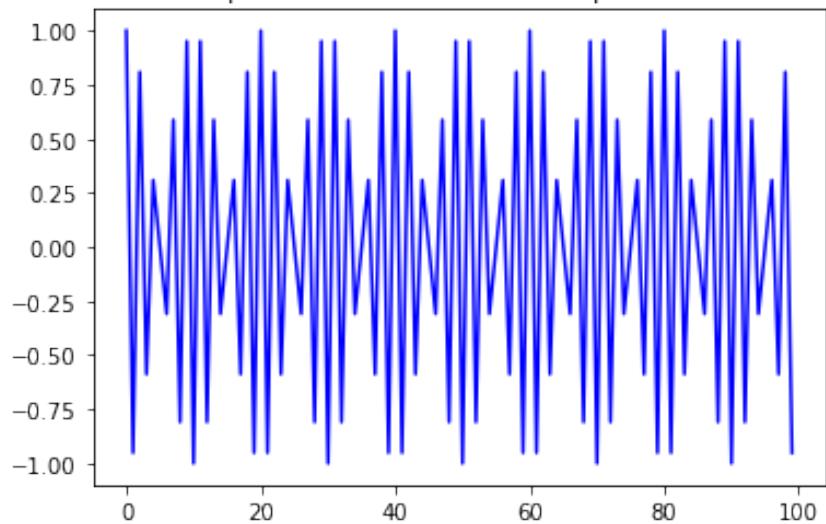
['pulsazione discreta = 0.54 * pi [rad]']



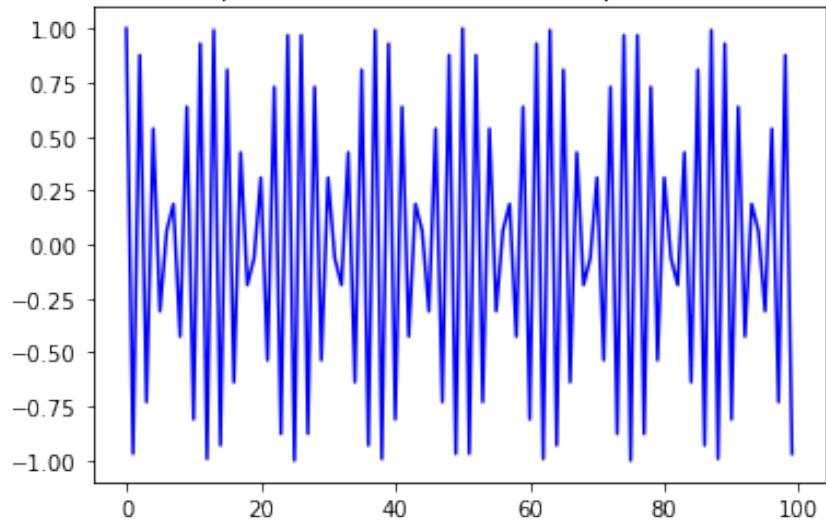
['pulsazione discreta = 0.72 * pi [rad]']



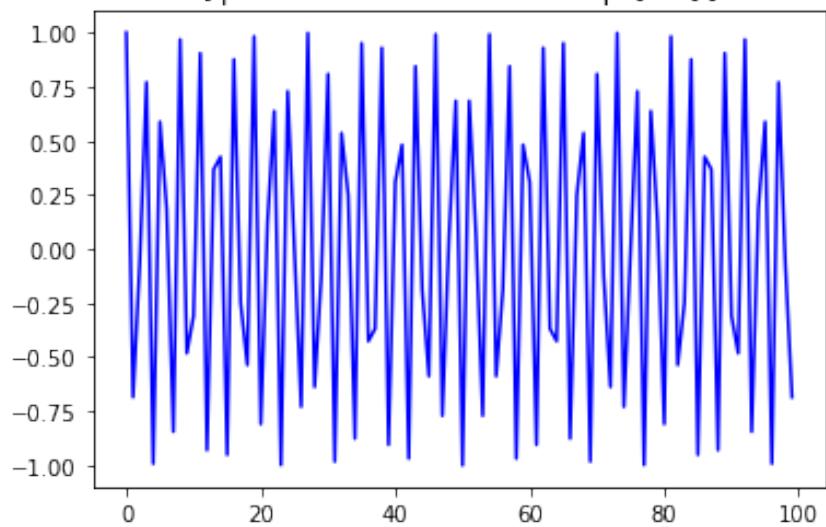
['pulsazione discreta = 0.9 * pi [rad]']



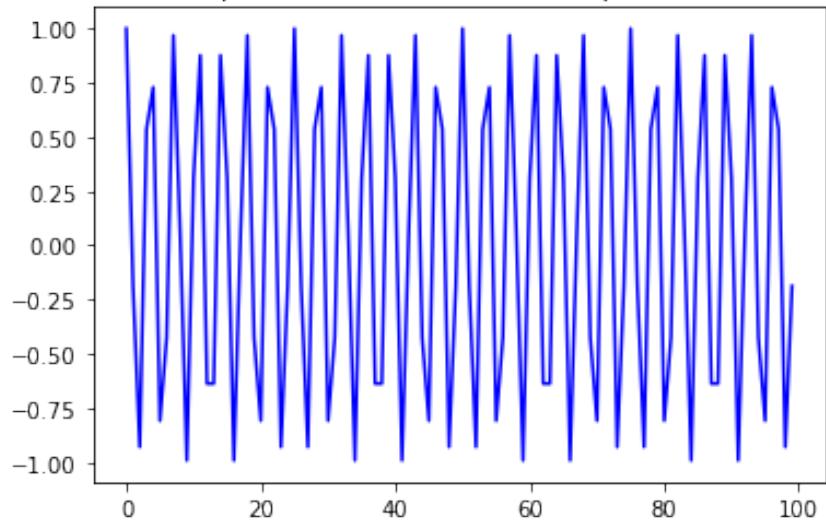
['pulsazione discreta = 1.08 * pi [rad]']



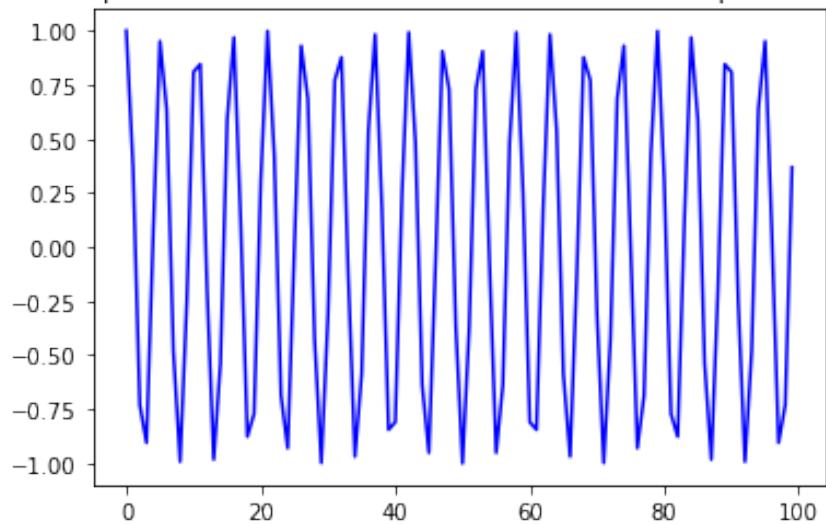
['pulsazione discreta = 1.26 * pi [rad]']



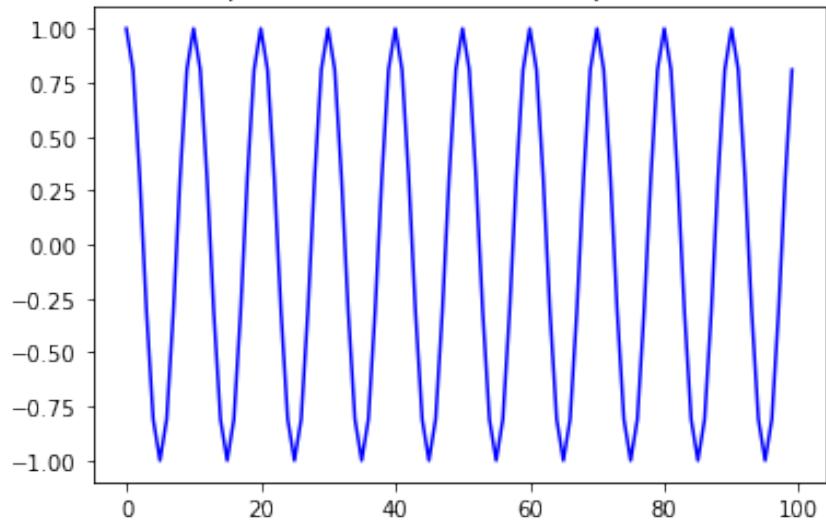
['pulsazione discreta = 1.44 * pi [rad]']



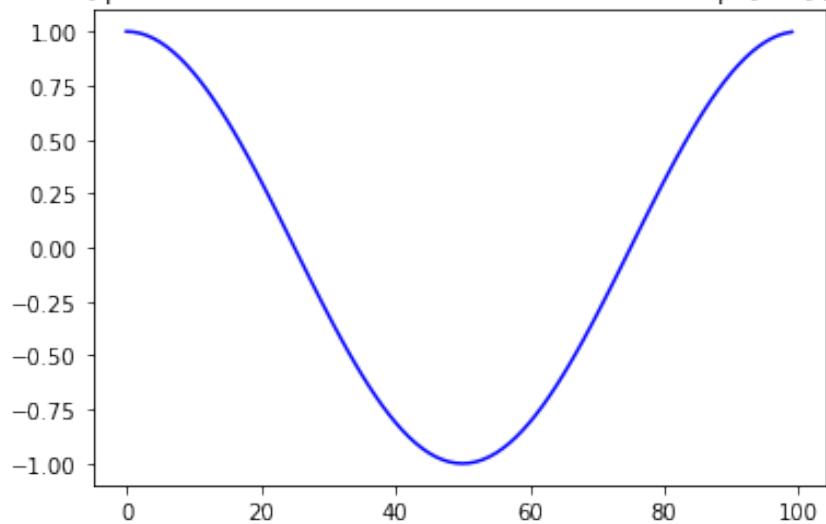
['pulsazione discreta = 1.619999999999999 * pi [rad]']



[*'pulsazione discreta = 1.8 * pi [rad]'*]



[*'pulsazione discreta = 1.9800000000000002 * pi [rad]'*]

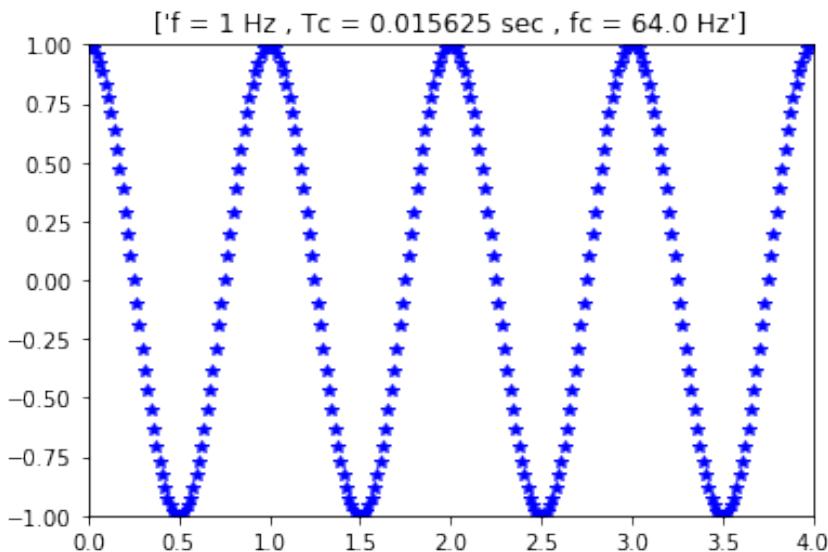


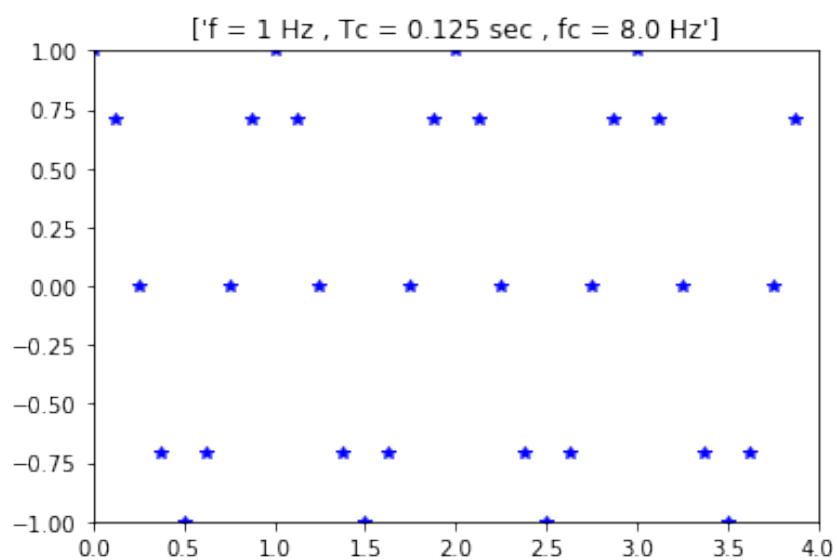
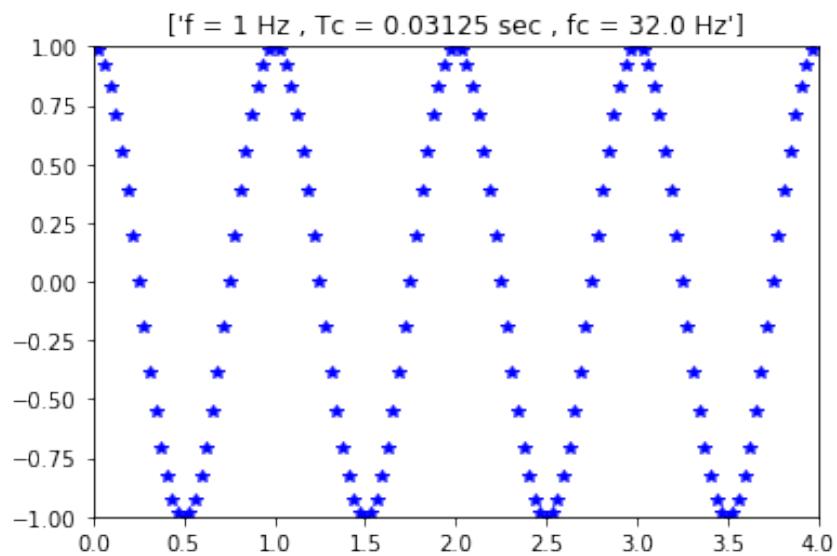
torna al par. 7.1

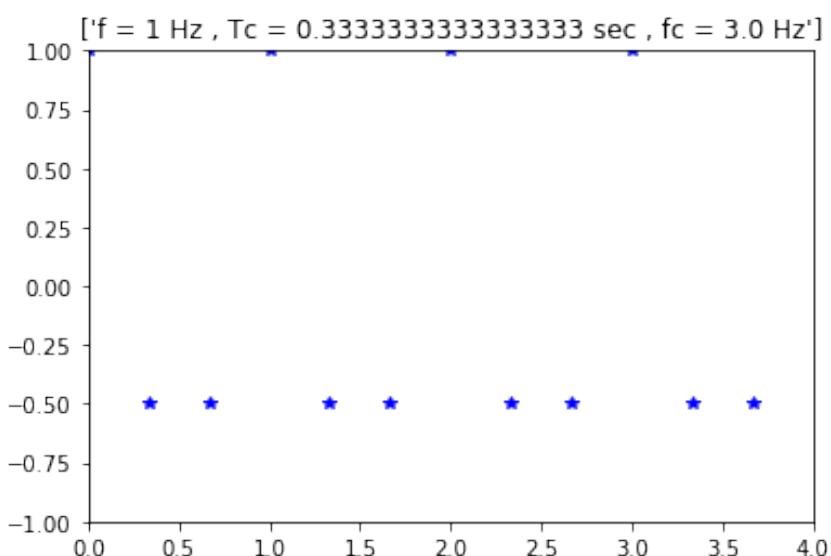
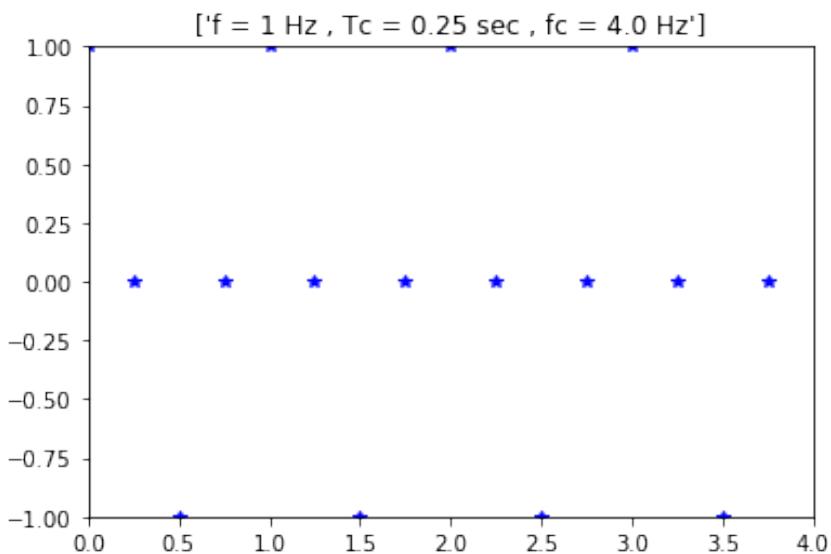
C.3 Fenomeno dell'aliasing

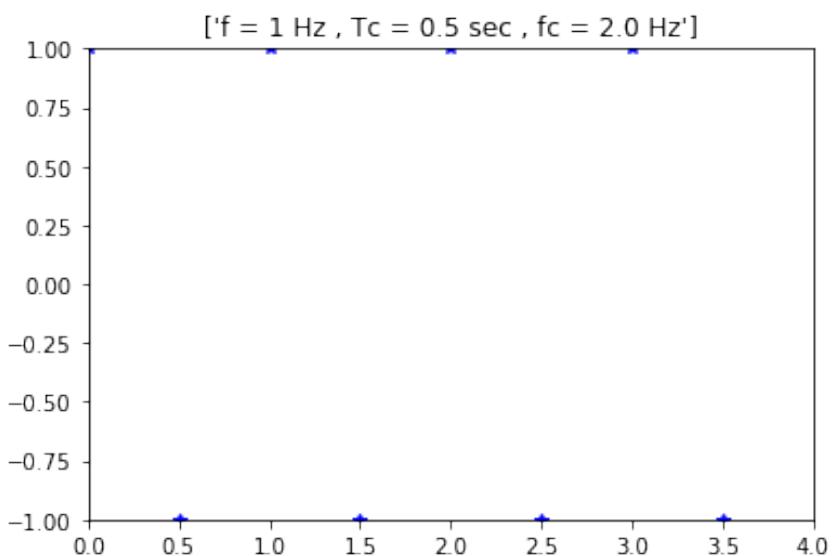
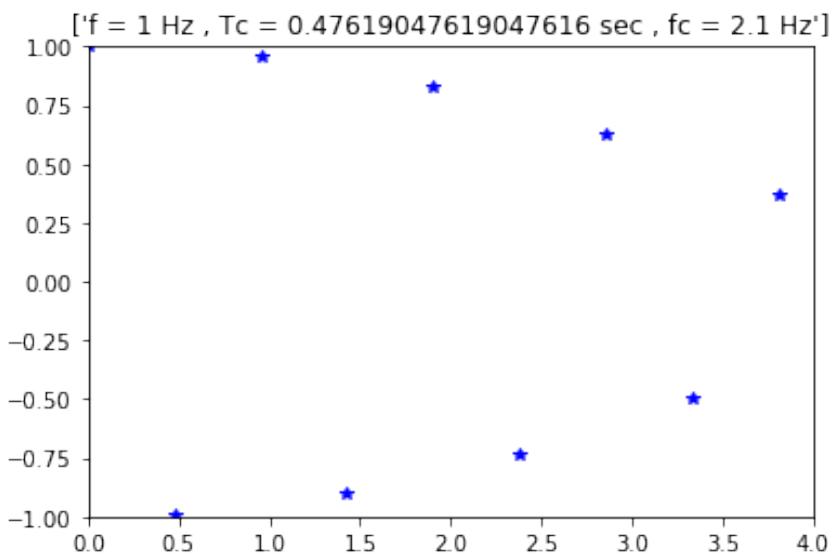
In [8]: # osservazione empirica del fenomeno di "aliasing" :

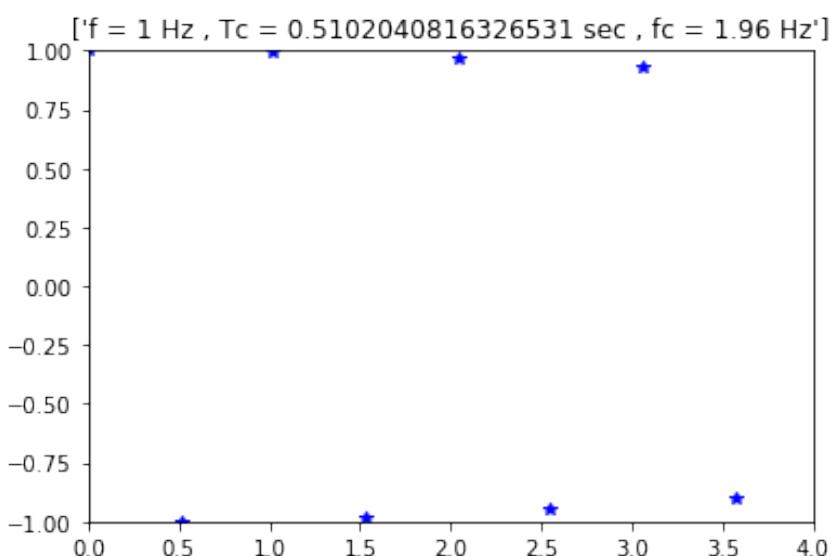
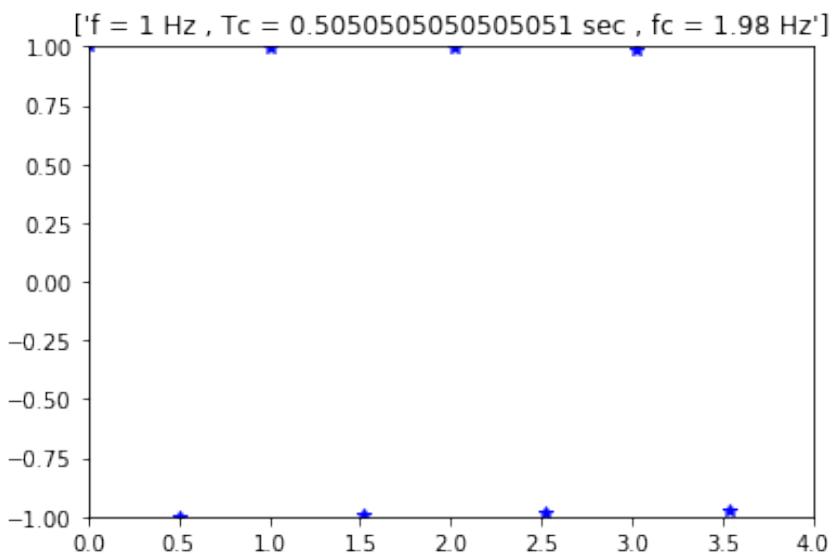
```
Tc_val = [1./64, 1./32, 1./8, 1./4, 1./3, 1./2.1, 1./2, 1./1.98, 1./1.96, 1./1.94,
           1./1.92, 1./1.9, 1./1.88, 1./1.86, 1./1.15, 1./1.05, 1.];
T = 4;
f = 1; # frequenza [Hz]
t = np.arange(0,T,Tc_val[0]);
s = np.cos(2*np.pi*f*t);
for i in range(0,len(Tc_val)):
    Tc = Tc_val[i];
    t = np.arange(0,T,Tc);
    plt.figure(200+i)
    plt.plot(t,np.cos(2*np.pi*f*t), 'b*'), plt.axis([0., T, -1., 1.]);
    plt.title(['f = ' + str(f) + ' Hz , Tc = ' + str(Tc) + ' sec , ' +
               'fc = ' + str(1./Tc) + ' Hz']); plt.show()
#endif
```

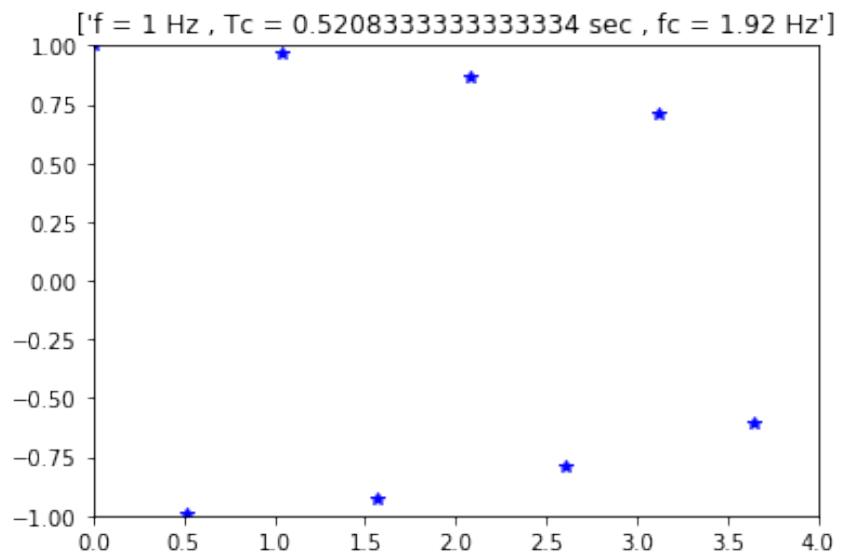
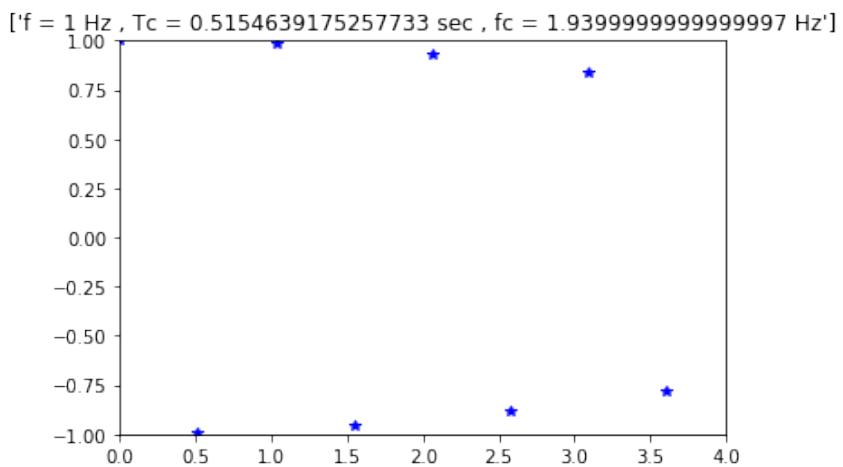




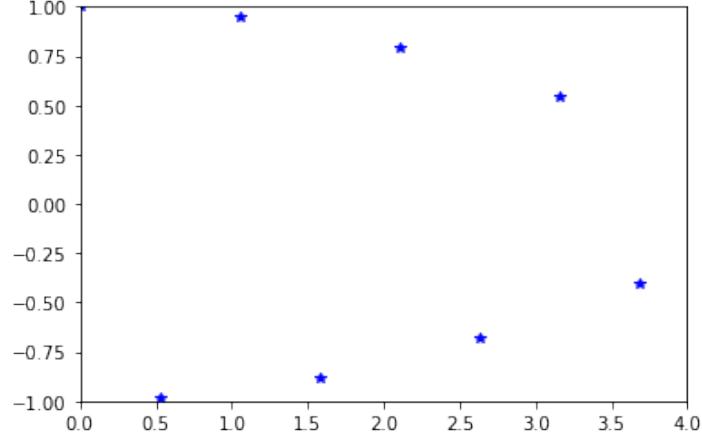




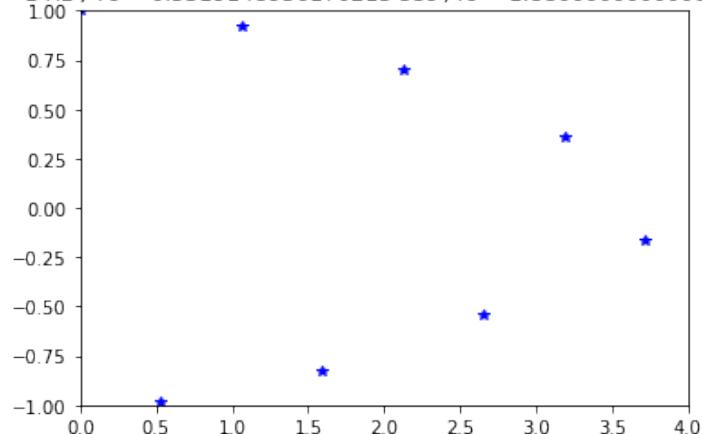


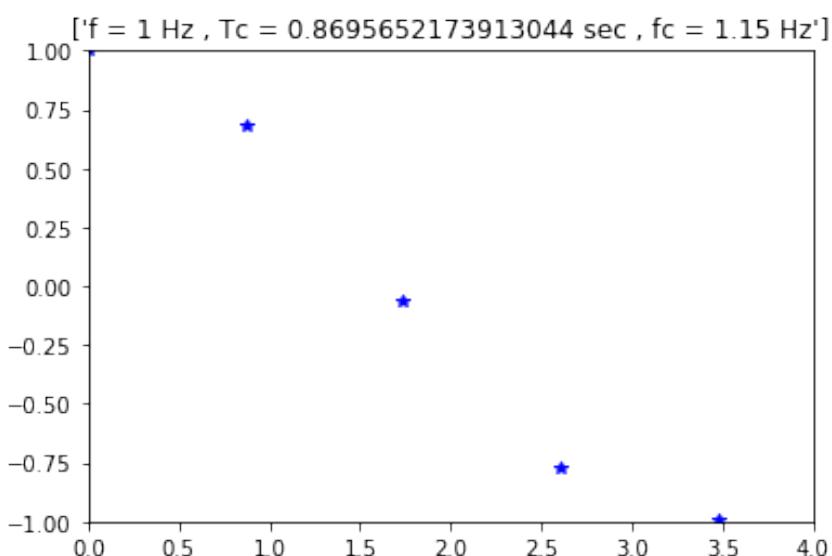
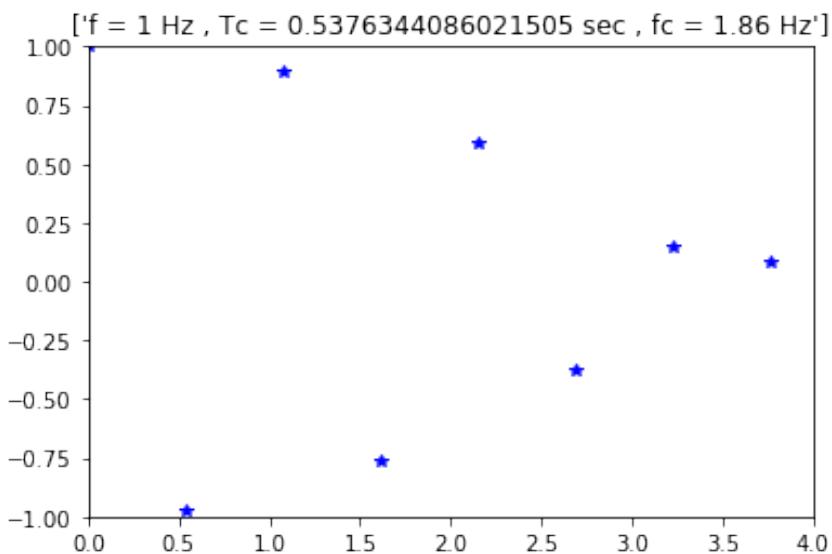


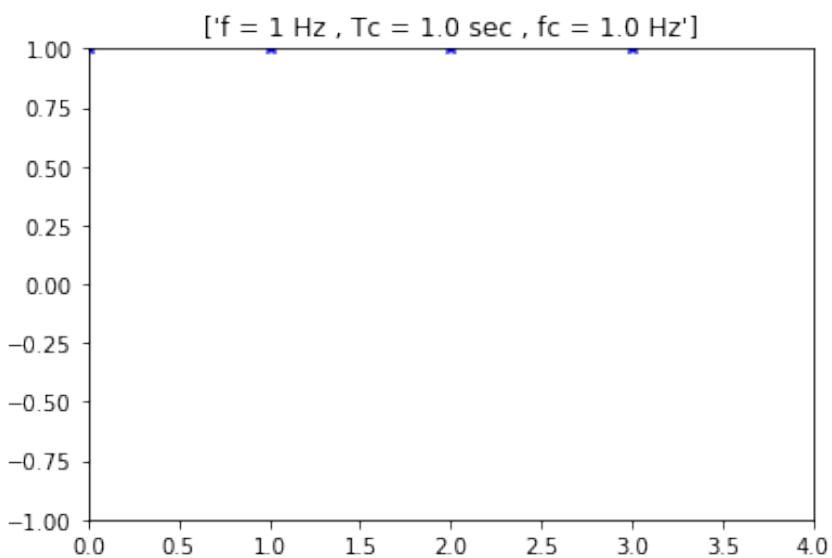
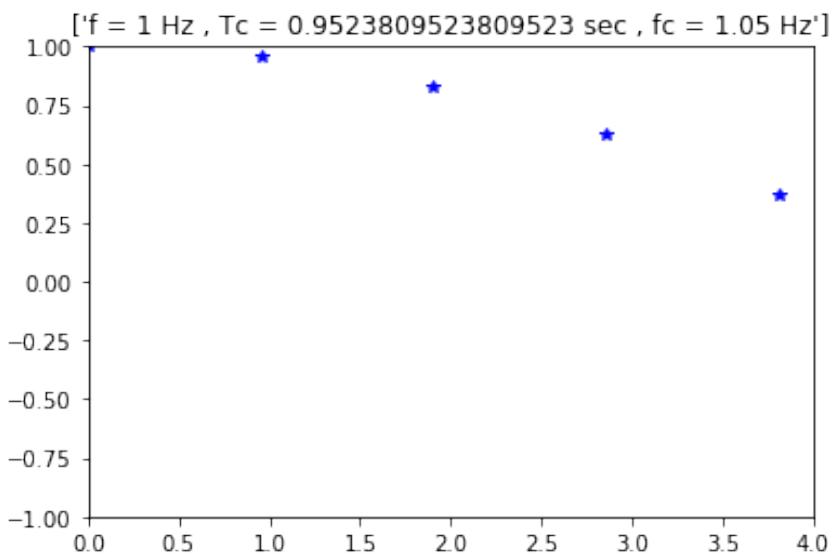
[$f = 1 \text{ Hz}$, $T_c = 0.5263157894736842 \text{ sec}$, $f_c = 1.9000000000000001 \text{ Hz}$]



[$f = 1 \text{ Hz}$, $T_c = 0.5319148936170213 \text{ sec}$, $f_c = 1.8800000000000001 \text{ Hz}$]



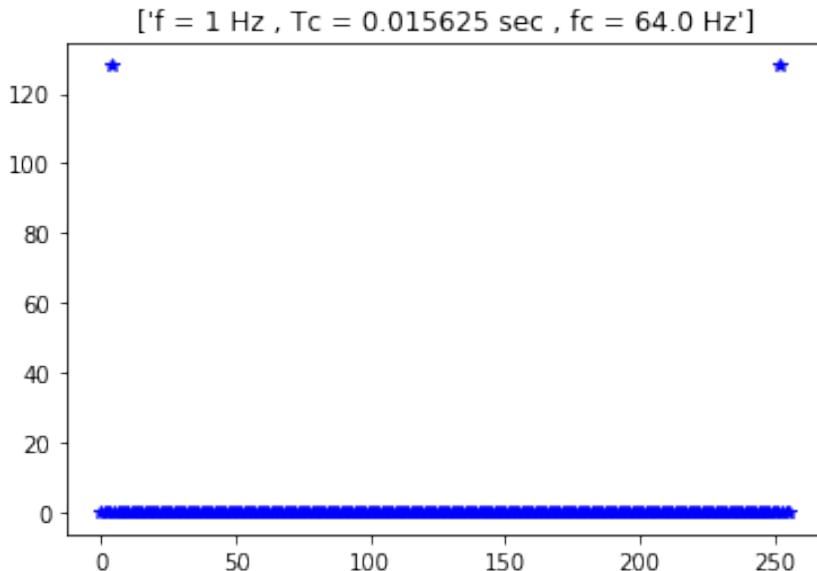




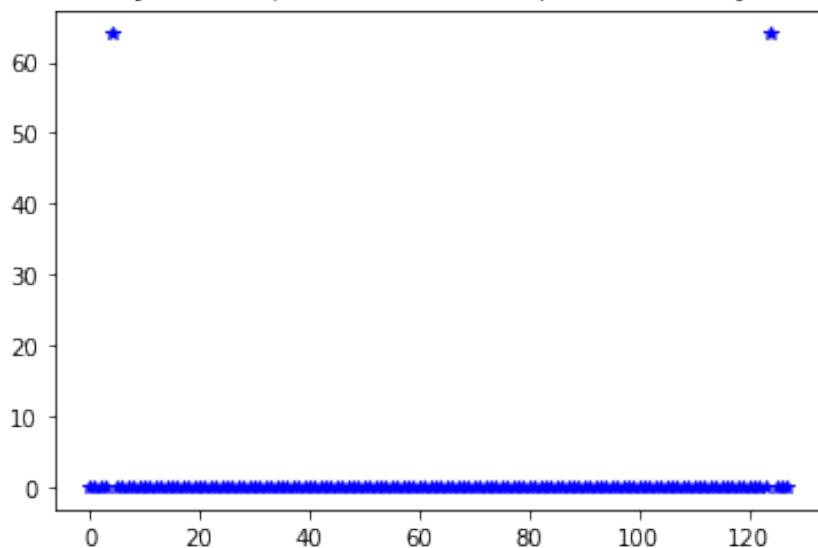
torna al par. 7.2

C.4 Osservazione dell'aliasing nel dominio delle frequenze

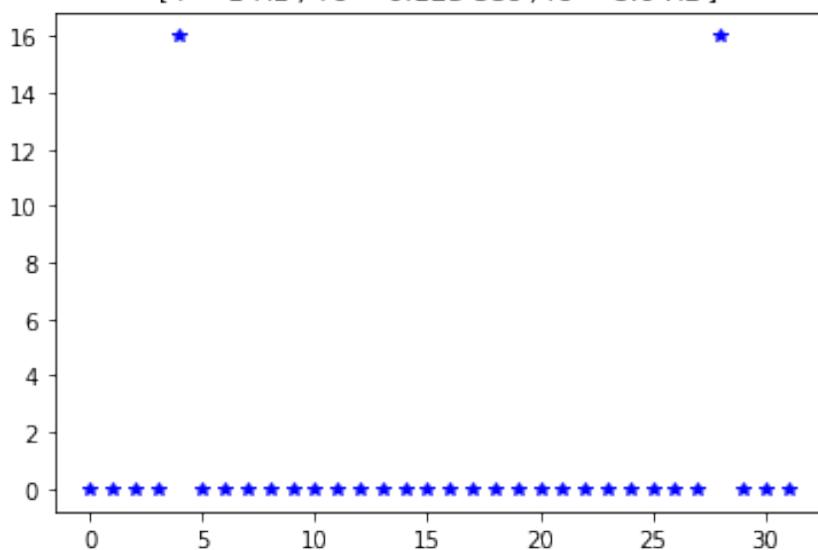
```
In [9]: # NB: tornare qui dopo aver visto la Trasformata di Fourier Discreta:  
# osservazione empirica del fenomeno di "aliasing" nel dominio delle frequenze:  
from numpy.fft import *  
Tc_val = [1./64, 1./32, 1./8, 1./4, 1./3, 1./2.1, 1./2, 1./1.98, 1./1.96, 1./1.94,  
          1./1.92, 1./1.9, 1./1.88, 1./1.86, 1./1.15, 1./1.05, 1.];  
T = 4;  
f = 1; # frequenza [Hz]  
for i in range(0,len(Tc_val)):  
    Tc = Tc_val[i];  
    t = np.arange(0,T,Tc);  
    plt.figure(200+i)  
    plt.plot(np.abs(fft(np.cos(2*np.pi*f*t))), 'b*'), #p.axis([0., T, -1., 1.]);  
    plt.title(['f = ' + str(f) + ' Hz , Tc = ' + str(Tc) + ' sec , ' +\n               'fc = ' + str(1./Tc) + ' Hz']); plt.show()  
#endfor
```



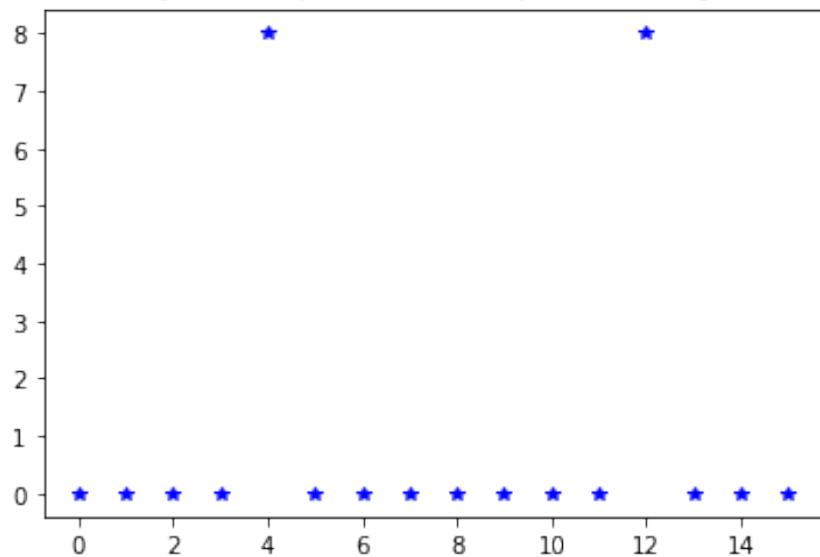
[$f = 1 \text{ Hz}$, $T_c = 0.03125 \text{ sec}$, $f_c = 32.0 \text{ Hz}$]



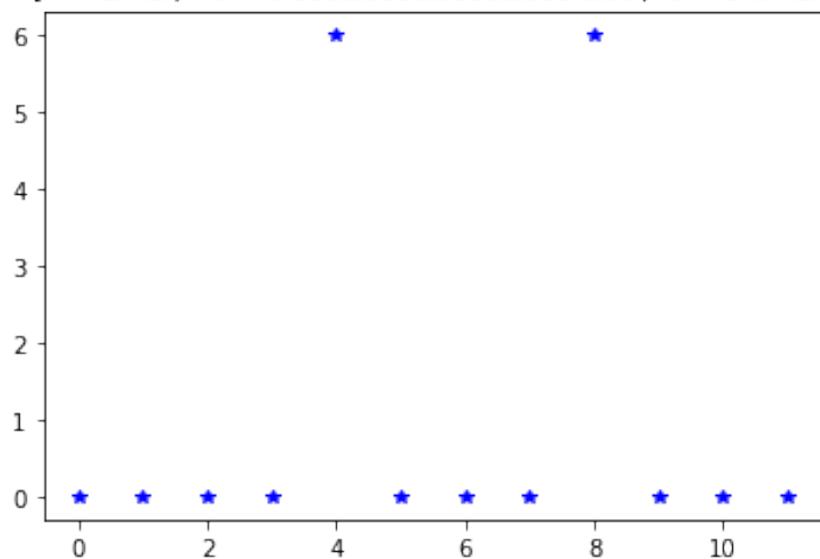
[$f = 1 \text{ Hz}$, $T_c = 0.125 \text{ sec}$, $f_c = 8.0 \text{ Hz}$]



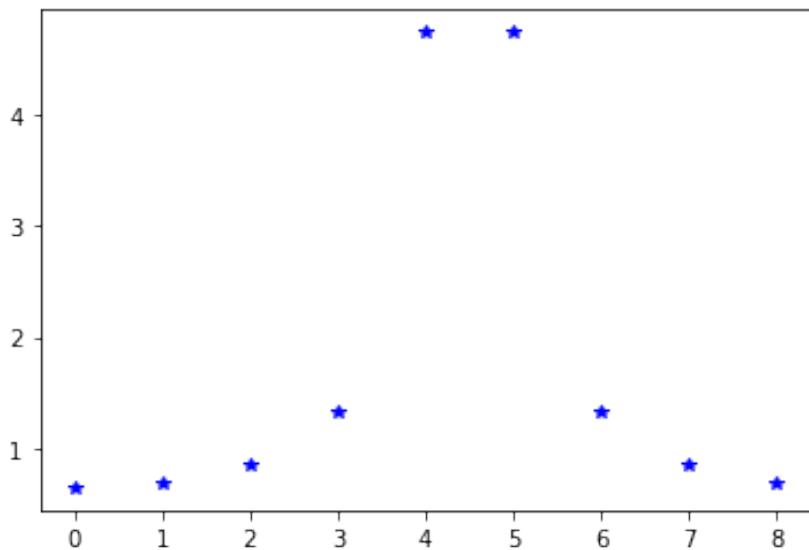
[$f = 1 \text{ Hz}$, $T_c = 0.25 \text{ sec}$, $f_c = 4.0 \text{ Hz}$]



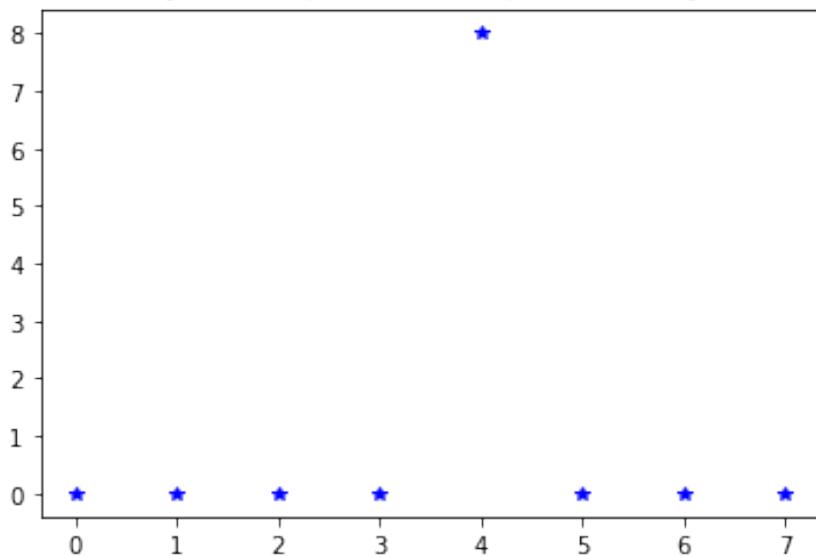
[$f = 1 \text{ Hz}$, $T_c = 0.3333333333333333 \text{ sec}$, $f_c = 3.0 \text{ Hz}$]



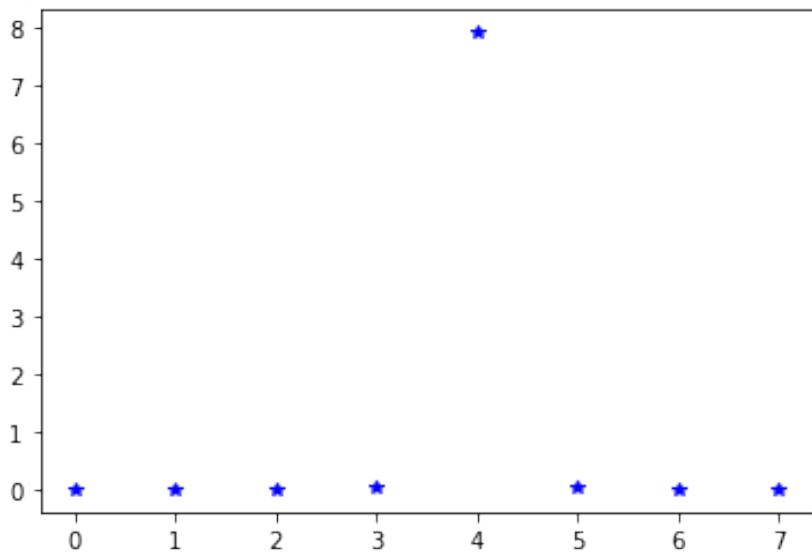
[$f = 1 \text{ Hz}$, $T_c = 0.47619047619047616 \text{ sec}$, $f_c = 2.1 \text{ Hz}$]



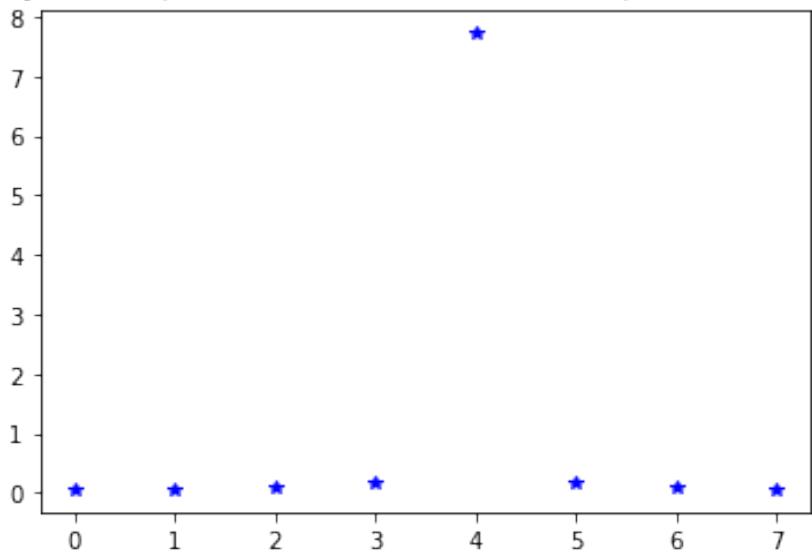
[$f = 1 \text{ Hz}$, $T_c = 0.5 \text{ sec}$, $f_c = 2.0 \text{ Hz}$]



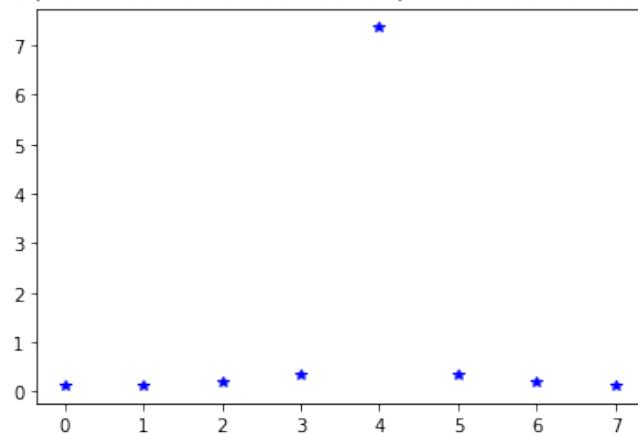
[$f = 1 \text{ Hz}$, $T_c = 0.5050505050505051 \text{ sec}$, $f_c = 1.98 \text{ Hz}$]



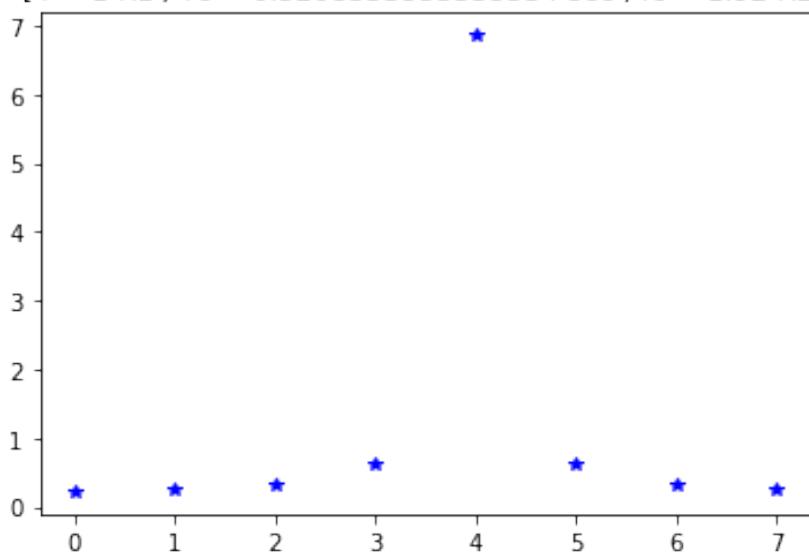
[$f = 1 \text{ Hz}$, $T_c = 0.5102040816326531 \text{ sec}$, $f_c = 1.96 \text{ Hz}$]



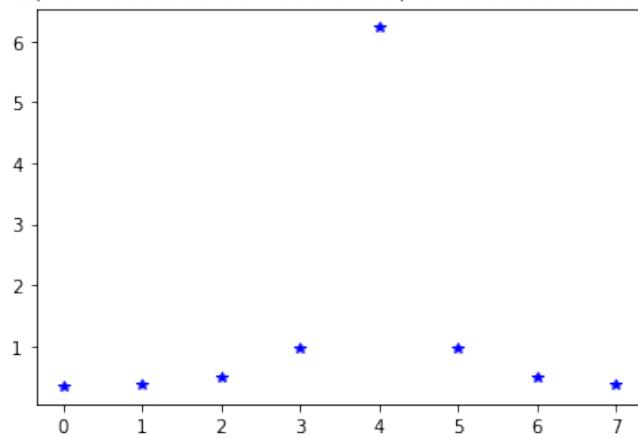
[$f = 1 \text{ Hz}$, $Tc = 0.5154639175257733 \text{ sec}$, $fc = 1.9399999999999997 \text{ Hz}$]



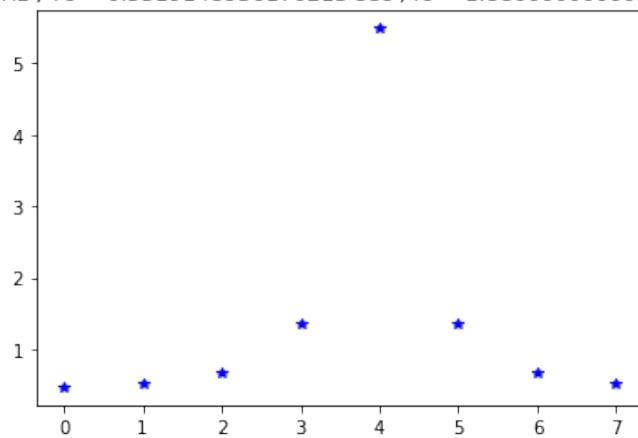
[$f = 1 \text{ Hz}$, $Tc = 0.5208333333333334 \text{ sec}$, $fc = 1.92 \text{ Hz}$]



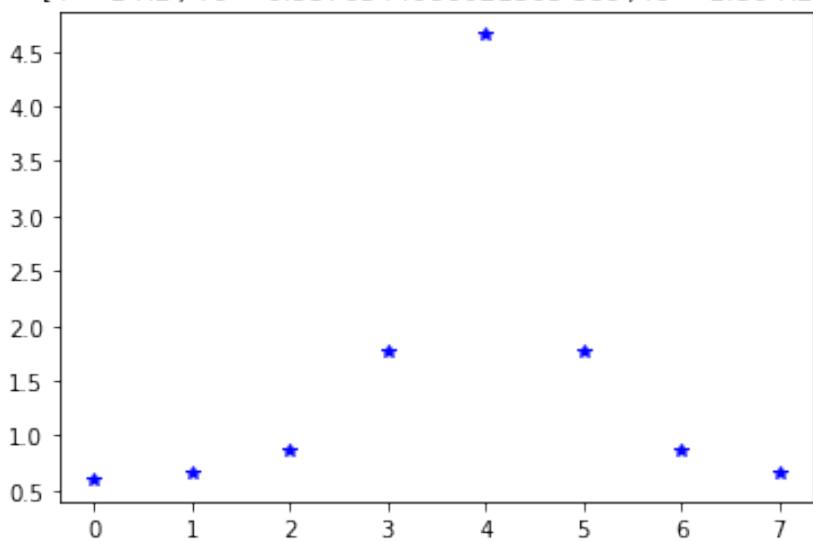
[$f = 1 \text{ Hz}$, $T_c = 0.5263157894736842 \text{ sec}$, $f_c = 1.9000000000000001 \text{ Hz}$]



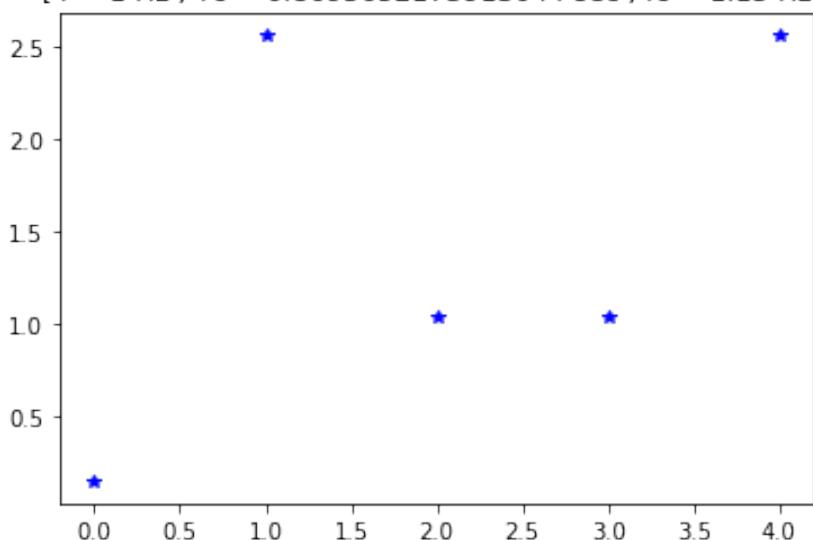
[$f = 1 \text{ Hz}$, $T_c = 0.5319148936170213 \text{ sec}$, $f_c = 1.8800000000000001 \text{ Hz}$]



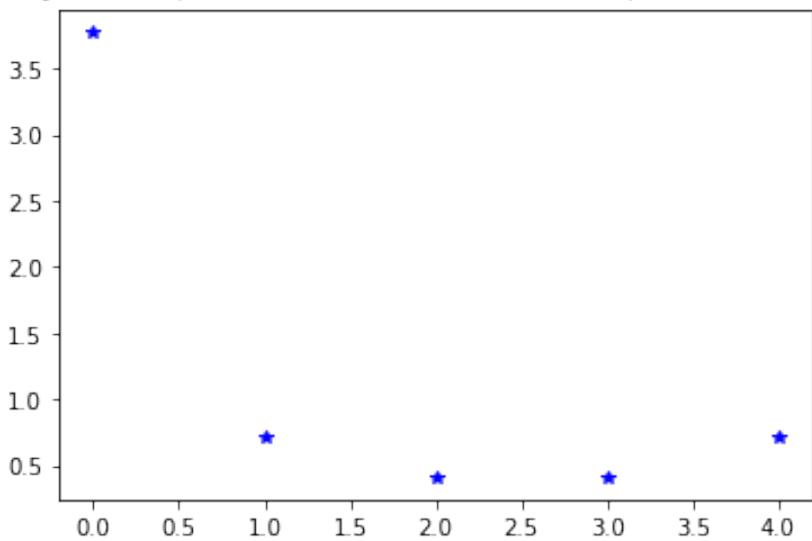
[$f = 1 \text{ Hz}$, $Tc = 0.5376344086021505 \text{ sec}$, $fc = 1.86 \text{ Hz}'$]



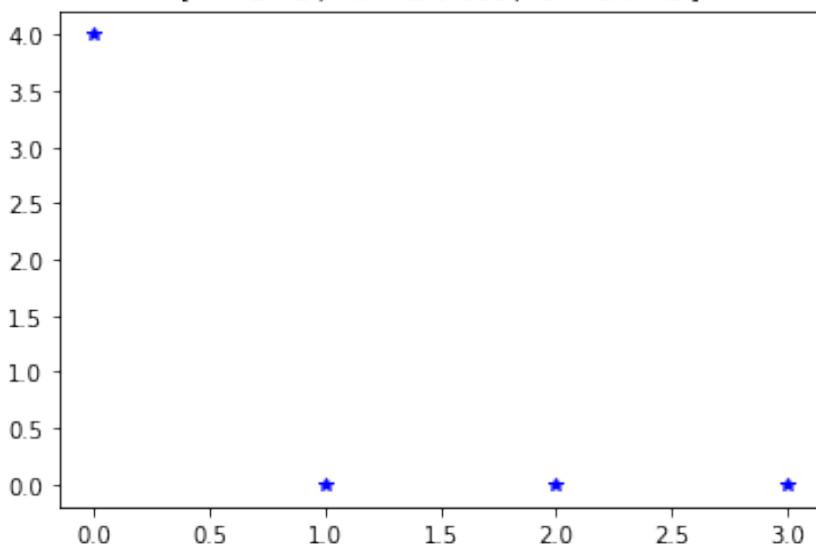
[$f = 1 \text{ Hz}$, $Tc = 0.8695652173913044 \text{ sec}$, $fc = 1.15 \text{ Hz}'$]



[$f = 1 \text{ Hz}$, $Tc = 0.9523809523809523 \text{ sec}$, $fc = 1.05 \text{ Hz}$]



[$f = 1 \text{ Hz}$, $Tc = 1.0 \text{ sec}$, $fc = 1.0 \text{ Hz}$]

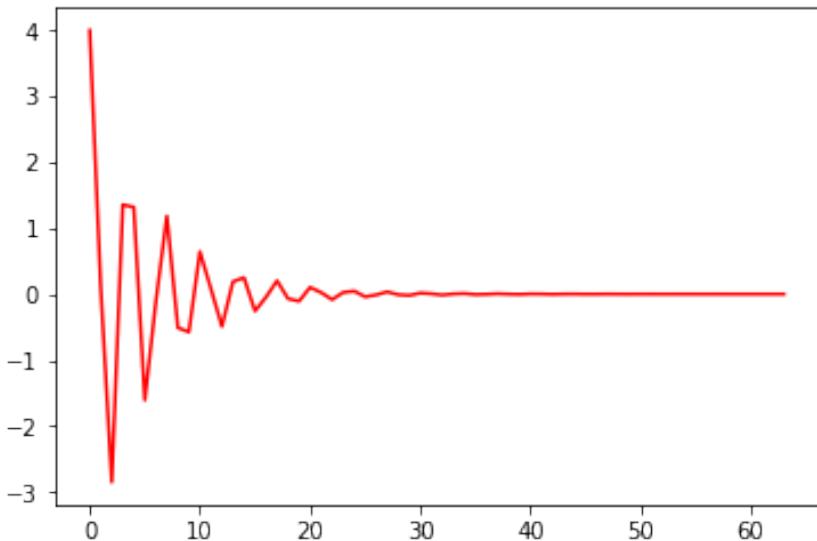


torna al par. ??

C.5 Esempi di sistemi LTI

C.6 Esempio ARMA

```
In [10]: # calcolo della risposta al campione unitario per la rappresentazione ARMA:  
a = np.array([1., 0.5, 0.7])  
b = np.array([4., 2.1])  
N = 64  
u = np.zeros(N); u[0] = 1.0  
y = simula_DLTI(b,a,u)  
plt.figure(1); plt.plot(y,'r-'); plt.show()
```



Torna alla (2.16)

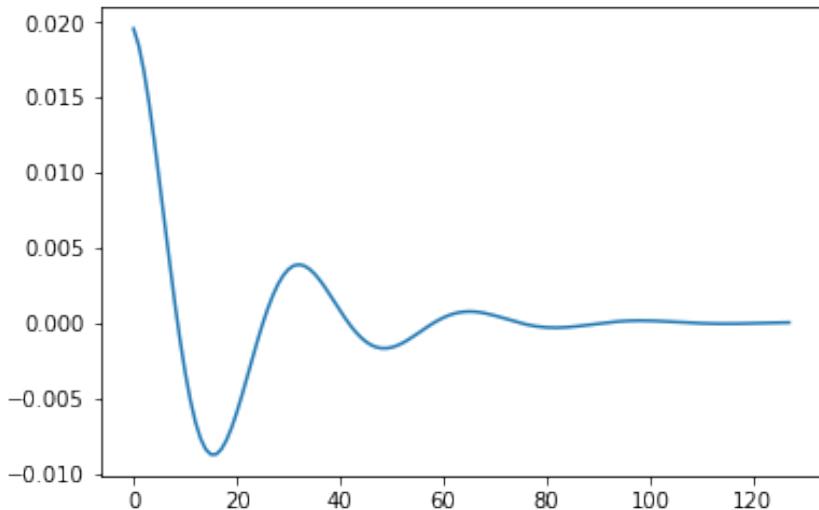
C.7 Esempio state-space a tempo continuo

```
In [11]: # calcolo della risposta al campione unitario per la rappresentazione state-space  
N = 128  
A = np.array([[.8, -10.2],[10.2, -4.]])  
B = np.array([[1.],[0.]])  
C = np.array([[1., 0.]])  
D = np.array([[0.]])  
u = np.zeros((1,N)); u[0,0] = 1.0
```

```

x0 = np.zeros(A.shape[0])
Ts = 0.02
y,X_hist,Ad = simula_DLTI_StateSpace_continuo(A,B,C,D,u,x0,Ts)
plt.figure(2); plt.plot(y[0,:].T); plt.show()

```



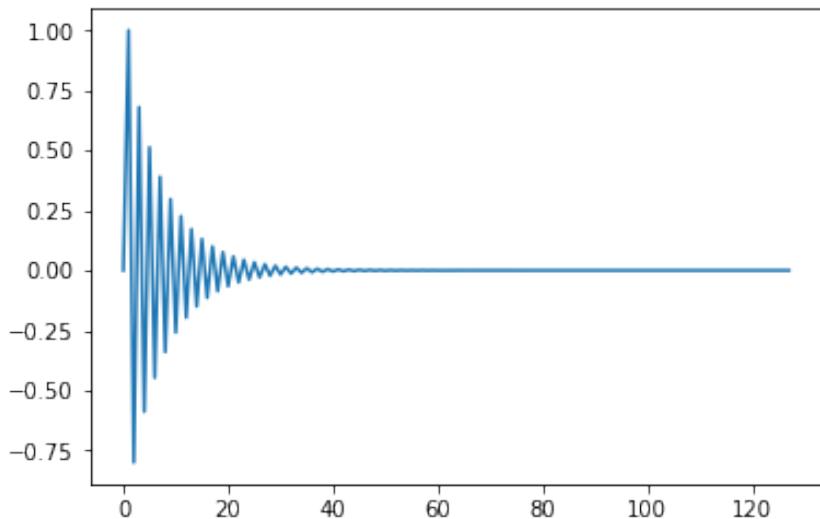
Torna alla (2.20)

C.8 Esempio state-space a tempo discreto

```

In [12]: # calcolo della risposta al campione unitario per la rappresentazione state-space
N = 128
A = np.array([[-0.8, 0.2], [0.2, -0.32]])
B = np.array([[1.], [0.]])
C = np.array([[1., 0.]])
D = np.array([[0.]])
u = np.zeros((1,N)); u[0,0] = 1.0
x0 = np.atleast_2d(np.zeros(A.shape[0])).T
y,X_hist = simula_DLTI_StateSpace_discreto(A,B,C,D,u,x0)
plt.figure(3); plt.plot(y[0,:].T); plt.show()
aA,vA=np.linalg.eig(A)
print(aA, abs(aA))

```



$[-0.87240999 \text{ } -0.24759001] \text{ } [0.87240999 \text{ } 0.24759001]$

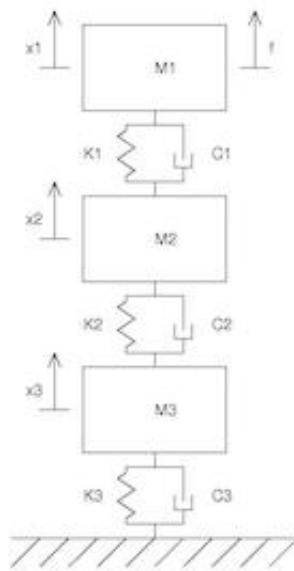
Torna alla (2.26)

C.9 Sistema delle tre masse

Vediamo in Figura un esempio di sistema meccanico a parametri concentrati con tre masse, molle e smorzatori.

Dati i valori dei parametri fisici, il sistema è supposto all'equilibrio quando le distanze tra i terminali delle tre molle, considerate partendo dall'alto, assumono i valori $\delta x_g = 0.005$, $\delta x_b = 0.005$ e $\delta x_s = 0.05$, che supponiamo essere la condizione iniziale del sistema dinamico. Questo corrisponde a supporre che la forma geometrica delle molle sia tale per cui l'equilibrio statico con la forza peso esercitata dalle masse avvenga per tali valori di compressione delle molle stesse (detti valori sono assunti anch'essi come dati del problema).

Le equazioni del sistema a tempo continuo possono essere ricavate semplicemente imponendo il soddisfacimento della legge di Newton, $F = m \cdot a$, ai tre gradi di



ex 2.13

libertà del nostro sistema, ovvero alle tre masse:

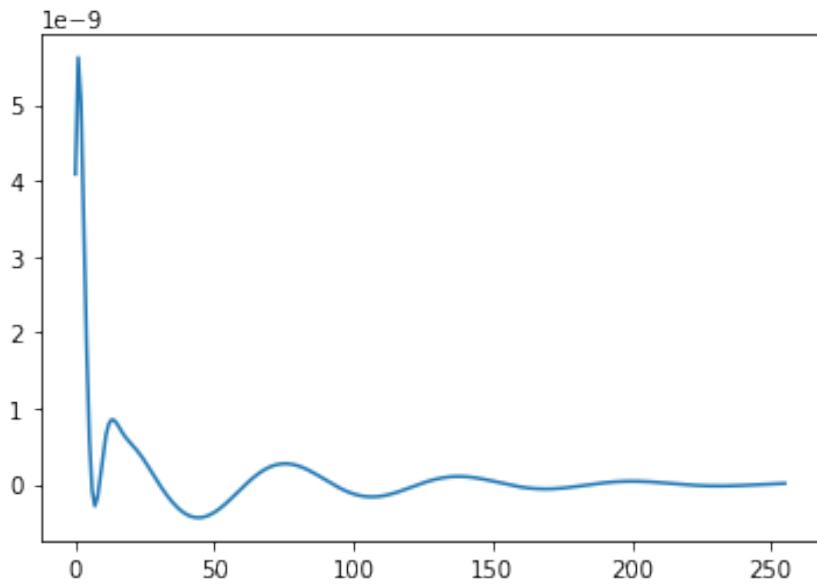
$$\left\{ \begin{array}{l} M_1 \ddot{x}_1(t) = C_1 (\dot{x}_2(t) - \dot{x}_1(t)) + K_1 (x_2(t) - x_1(t) + \delta x_g) + f \\ M_2 \ddot{x}_2(t) = C_2 (\dot{x}_3(t) - \dot{x}_2(t)) + C_1 (\dot{x}_1(t) - \dot{x}_2(t)) + K_2 (x_3(t) - x_2(t) + \delta x_b) + \\ \quad K_1 (x_1(t) - x_2(t) - \delta x_g) \\ M_3 \ddot{x}_3(t) = C_3 (-\dot{x}_3(t)) + C_2 (\dot{x}_2(t) - \dot{x}_3(t)) + K_3 (-x_3(t) + \delta x_s) + \\ \quad K_2 (x_2(t) - x_3(t) - \delta x_b) \end{array} \right.$$

Per simulare questo modello o per utilizzarlo per il confronto con i dati, è necessario discretizzarlo, cfr. Appendice B.3.

```
In [13]: A,B,C,D = build_sistema_meccanico_3gdl()
A,B,C,D = build_sistema_meccanico_3gdl(M1_stimato=10.0)
```

```
In [14]: N = 256
u = np.zeros(N); u[0] = 1.0
```

```
In [15]: Ts = 0.001
y, X_hist = simula_sistema_meccanico_3gdl(A, B, C, D, u, Ts)
plt.figure(1); plt.plot(y[0,:].T); plt.show()
```



Torna al par. [2.3.3](#)

C.10 Sistema delle tre masse (implementazione ad oggetti)

In [16]: `s1 = SistemaMeccanico3gdl()`

In [17]: `s1.write()`

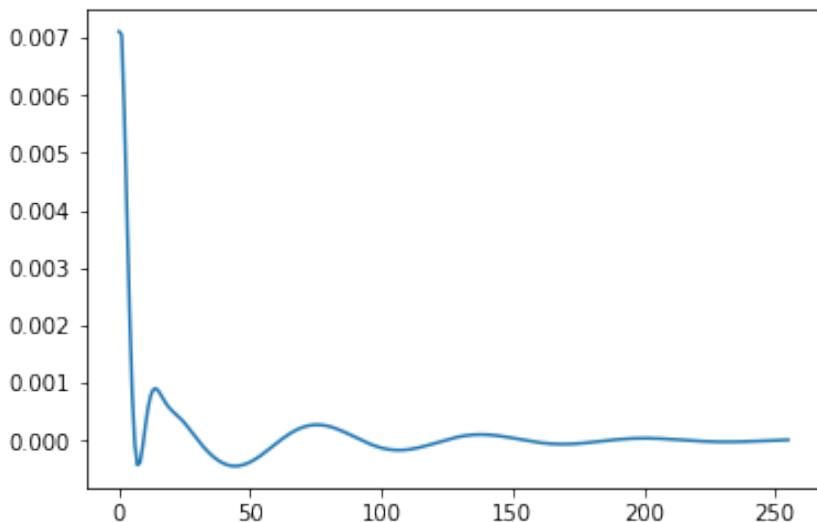
```
M1 = 27.0
M2 = 120.0
M3 = 11246.0
K1 = 180000000.0
K2 = 60000000.0
K3 = 120000000.0
C1 = 50000.0
C2 = 46000.0
C3 = 240000.0
coordinate di partenza: y1=0.06, y2=0.06, y3=0.05
```

In [18]: `A,B,C,D = s1.build_model()`

```
In [19]: N = 256
u = np.zeros(N); u[0] = 1.0e6
print("u.shape: ", u.shape)
```

```
u.shape: (256,)
```

```
In [20]: Ts = 0.001
y1, X_hist = s1.simula_sistema(A, B, C, D, u, Ts)
plt.figure(1); plt.plot(y1[0,:].T); plt.show()
```

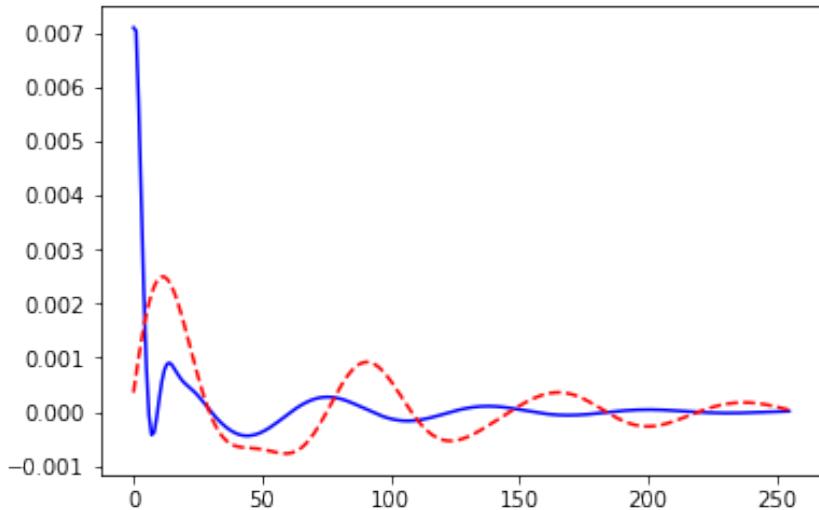


```
In [21]: s2 = SistemaMeccanico3gdl()
A,B,C,D = s2.build_model(M1_stimato=2700.0,\n                           M2_stimato=None,\n                           M3_stimato=None,\n                           K1_stimato=1.8e8,\n                           K2_stimato=None,\n                           K3_stimato=None,\n                           C1_stimato=5.0e4,\n                           C2_stimato=None,\n                           C3_stimato=None)
```

```
In [22]: s2.write()
```

```
M1 = 2700.0
M2 = 120.0
M3 = 11246.0
K1 = 180000000.0
K2 = 60000000.0
K3 = 120000000.0
C1 = 50000.0
C2 = 46000.0
C3 = 240000.0
coordinate di partenza: y1=0.06, y2=0.06, y3=0.05
```

```
In [23]: y2, X_hist = s2.simula_sistema(A, B, C, D, u, Ts)
plt.figure(2); plt.plot(y1[0,:].T, 'b-'); plt.plot(y2[0,:].T, 'r--'); plt.show()
```



Torna al par. 2.3.3

```
In [24]:
```

Appendice D

Algoritmi fattorizzazione QR

```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matplotlib inline"
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD import qr
```

D.1 Esempio di generazione di matrice random con condizionamento assegnato

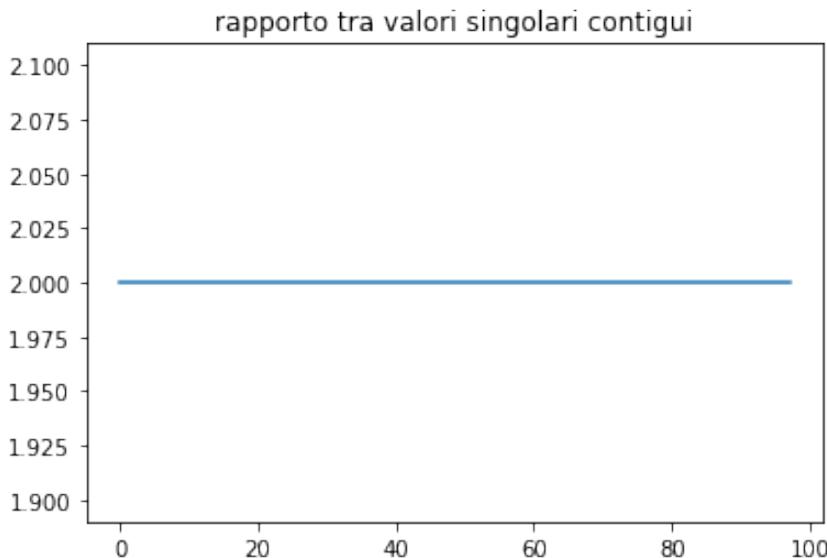
```
In [2]: n = 100
[U,_] = np.linalg.qr(np.random.randn(n,n))
[V,_] = np.linalg.qr(np.random.randn(n,n))
S = np.diag(2.*np.arange(0,-n,-1))
A = U @ S @ V.T;

In [3]: print("np.linalg.cond(A) = ",np.linalg.cond(A))
print("S[0,0]/S[-1,-1] = ",S[0,0]/S[-1,-1])
print("S[0,0] = ",S[0,0])
print("S[-1,-1] = ",S[-1,-1])
normA = np.linalg.norm(A); print("||A|| = ",normA)
norminvA = np.linalg.norm(np.linalg.inv(A)); print("||A^{-1}|| = ",norminvA)
print("||A||*||A^{-1}|| = ",normA*norminvA)
vsigma = np.diag(S)
plt.figure()
plt.plot(vsigma[0:-2]/vsigma[1:-1]); plt.title ("rapporto tra valori singolari controllati")
print("Ora calcoliamo la SVD di A:")
Utilde,Stilde,Vtilde = np.linalg.svd(A); Vtilde = Vtilde.T
print("Scalc[0]/Scalc[n] = ",Stilde[0]/Stilde[-1])
print("Scalc[0] = ",Stilde[0])
print("Scalc[n] = ",Stilde[-1])
```

```

np.linalg.cond(A) = 2.3578012923433533e+18
S[0,0]/S[-1,-1] = 6.338253001141147e+29
S[0,0] = 1.0
S[-1,-1] = 1.5777218104420236e-30
||A|| = 1.154700538379253
||A^{-1}|| = 5.31126663084633e+18
||A|| * ||A^{-1}|| = 6.132922438114018e+18
Ora calcoliamo la SVD di A:
Scalc[0]/Scalc[n] = 2.5936991458276028e+16
Scalc[0] = 0.9999999999999993
Scalc[n] = 3.855497279276457e-17

```



Torna al par. [1.4.2](#)

D.2 Esempio di trasformazione di Householder

```

In [4]: A = np.random.randn(4, 4)
        print("A iniziale = ",A)
        V, R = qr.Householder(A.copy(),verbose=True)
        print(V)

A iniziale = [[ 0.77298068 -1.56926606 -1.6105384   0.71237834]
 [-1.17491956 -0.68746114  0.33567113  0.06450657]

```

```

[ 0.49930126 -1.76364567  1.36444164  0.11404897]
[ 0.12196486 -0.58648184  0.06974355 -0.35754757]]
A = [[-1.49736845e+00  9.06538058e-01  6.34131130e-01 -3.26039478e-01]
[ 0.00000000e+00 -1.96870491e+00 -8.25959079e-01  6.01894138e-01]
[-5.55111512e-17 -1.21916020e+00  1.85809537e+00 -1.14322652e-01]
[-1.38777878e-17 -4.53479785e-01  1.90328878e-01 -4.13332152e-01]]
A = [[-1.49736845e+00  9.06538058e-01  6.34131130e-01 -3.26039478e-01]
[ 0.00000000e+00  2.35961745e+00 -3.07489049e-01 -3.63676129e-01]
[-5.55111512e-17  0.00000000e+00  2.00413302e+00 -3.86295195e-01]
[-1.38777878e-17  5.55111512e-17  2.44649157e-01 -5.14495273e-01]]
A = [[-1.49736845e+00  9.06538058e-01  6.34131130e-01 -3.26039478e-01]
[ 0.00000000e+00  2.35961745e+00 -3.07489049e-01 -3.63676129e-01]
[-5.55111512e-17  0.00000000e+00 -2.01901024e+00  4.45791592e-01]
[-1.38777878e-17  5.55111512e-17  2.77555756e-17 -4.63895700e-01]]
A = [[-1.49736845e+00  9.06538058e-01  6.34131130e-01 -3.26039478e-01]
[ 0.00000000e+00  2.35961745e+00 -3.07489049e-01 -3.63676129e-01]
[-5.55111512e-17  0.00000000e+00 -2.01901024e+00  4.45791592e-01]
[-1.38777878e-17  5.55111512e-17  2.77555756e-17  4.63895700e-01]]
[[ 0.87069688  0.          0.          0.        ]
 [-0.45059096 -0.9576879   0.          0.        ]
 [ 0.19148599 -0.26975231  0.99815616  0.        ]
 [ 0.04677449 -0.10033728  0.06069833 -1.        ]]

```

Torna al par. 1.3.2

D.3 Confronto tra gli algoritmi di fattorizzazione QR

```
In [5]: m = 128
n = 12
A = np.random.randn(m, n)
print("K(A) = ", np.linalg.cond(A))
```

```
K(A) = 1.6694532157943038
```

```
In [6]: Q, R = np.linalg.qr(A.copy())
print("||A - Q*R|| = ", np.linalg.norm(A - Q @ R, 2))
```

```
||A - Q*R|| = 1.0462790500372156e-14
```

```
In [7]: Q, R = qr.CGS(A.copy())
print("||A - Q*R|| = ", np.linalg.norm(A - Q @ R, 2))
```

```
||A - Q*R|| = 2.330908589596488e-15
```

```
In [8]: Q, R = qr.MGS(A.copy())
print("||A - Q*R|| = ", np.linalg.norm(A - Q @ R,2))
```

```
||A - Q*R|| = 2.577018849736898e-15
```

```
In [9]: A = np.array([[0.44, 0.44, 0.44, 0.44+1e-15],[1.12, 1.12, 1.12-1e-15, 1.12],[0.88,
print("K(A) = ", np.linalg.cond(A))
Q, R = qr.CGS(A.copy())
print(R)
print("||A - Q*R|| = ", np.linalg.norm(A - Q@R,2))
print("||I - Q'*Q|| = ", np.linalg.norm(np.eye(A.shape[0]) - Q.T@Q,2))
print(" ")
Q, R = qr.MGS(A.copy())
print(R)
print("||A - Q*R|| = ", np.linalg.norm(A - Q@R,2))
print("||I - Q'*Q|| = ", np.linalg.norm(np.eye(A.shape[0]) - Q.T@Q,2))
print(" ")
Q, R = np.linalg.qr(A.copy())
print(R)
print("||A - Q*R|| = ", np.linalg.norm(A - Q@R,2))
print("||I - Q'*Q|| = ", np.linalg.norm(np.eye(A.shape[0]) - Q.T@Q,2))
print(" ")
V, R = qr.Householder(A.copy())
print(R)
print(" ")
```

```
K(A) = 1.2037163922483046e+16
```

```
[[ 1.59248234e+00 1.59248234e+00 1.59248234e+00 1.59248234e+00]
```

```
[ 0.00000000e+00 8.47340949e-16 -1.28403869e-01 -1.28403869e-01]
```

```
[ 0.00000000e+00 0.00000000e+00 1.28403869e-01 -1.28403869e-01]
```

```
[ 0.00000000e+00 0.00000000e+00 0.00000000e+00 2.56807739e-01]]
```

```
||A - Q*R|| = 1.6213750489334986e-16
```

```
||I - Q'*Q|| = 2.009705008327854
```

```
[[ 1.59248234e+00 1.59248234e+00 1.59248234e+00 1.59248234e+00]
```

```
[ 0.00000000e+00 8.47340949e-16 4.50945398e-16 -2.03652760e-16]
```

```
[ 0.00000000e+00 0.00000000e+00 5.92719219e-16 4.25283403e-16]
```

```
[ 0.00000000e+00 0.00000000e+00 0.00000000e+00 8.23075969e-16]]
```

```
||A - Q*R|| = 2.543840524313801e-16
```

```
||I - Q'*Q|| = 0.35345406818360975
```

```
[[ -1.59248234e+00 -1.59248234e+00 -1.59248234e+00 -1.59248234e+00]]
```

```

[ 0.00000000e+00  8.15843973e-16  5.74112426e-16 -1.66190439e-16]
[ 0.00000000e+00  0.00000000e+00  6.00539936e-16  3.23075282e-16]
[ 0.00000000e+00  0.00000000e+00  0.00000000e+00 -4.74821754e-16]
||A - Q*R|| =  1.135363022810288e-15
||I - Q'*Q|| =  8.695092970894871e-16

[[-1.59248234e+00 -1.59248234e+00 -1.59248234e+00 -1.59248234e+00]
 [ 0.00000000e+00  8.30814836e-16  5.63767210e-16  1.48359792e-16]
 [-1.11022302e-16  0.00000000e+00  5.34918838e-16  4.19705550e-16]
 [ 0.00000000e+00 -2.46519033e-32  1.47911420e-31  1.04692958e-15]]

```

In [10]: `np.linalg.norm(A[:,0],2)`

Out[10]: 1.5924823389915508

Torna al par. 1.3.3

D.4 Esempio di trasformazione di Givens

```

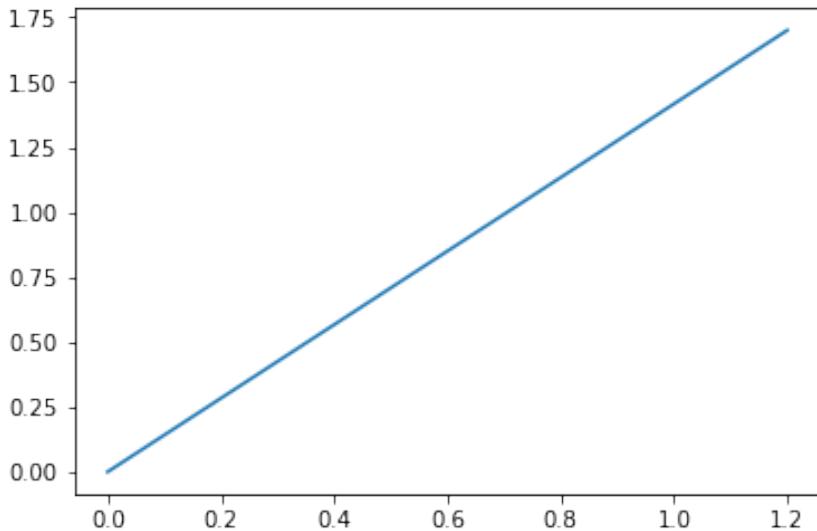
In [11]: # trasformazioni di Givens: esempio di rotazione
x = np.array([[1.2],[1.7]])
print("||x||_2 = ",np.linalg.norm(x))
plt.figure(1); plt.plot([0, x[0]],[0, x[1]]);

```

$||x||_2 = 2.080865204668481$

```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/numpy/core/_as
```

 $\text{return array(a, dtype, copy=False, order=order, subok=True)}$



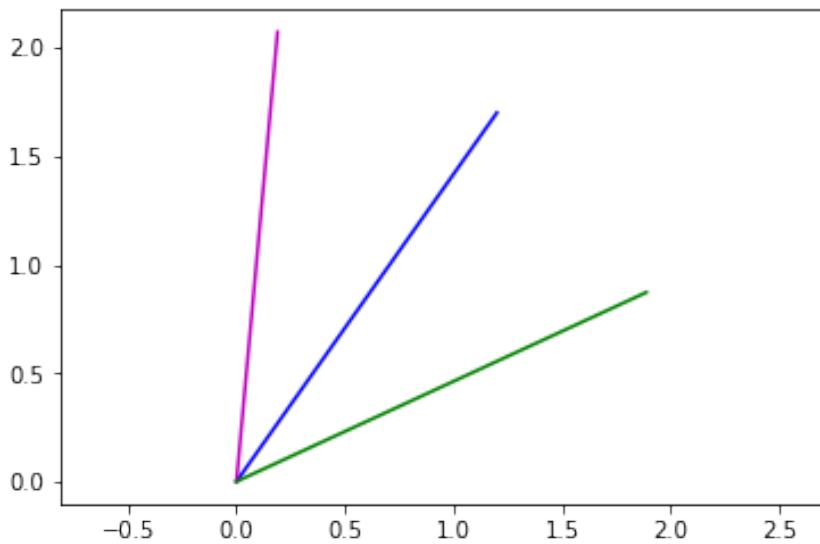
```
In [12]: # trasformazioni di Givens: esempio di rotazione
x = np.array([[1.2],[1.7]])
print("||x||_2 = ",np.linalg.norm(x))
plt.figure(1); plt.plot([0, x[0]],[0, x[1]],'b-');
# ruotiamo "x" di pi/6 in senso antiorario:
theta = np.pi/6.
G = np.array([[np.cos(theta), -np.sin(theta)],[np.sin(theta), np.cos(theta)]])
xra = G @ x
print("||xra||_2 = ",np.linalg.norm(xra))
plt.plot([0, xra[0]],[0, xra[1]],'m-');
xro = G.T @ x
print("||xro||_2 = ",np.linalg.norm(xro))
plt.plot([0, xro[0]],[0, xro[1]],'g-');
plt.axis([0, 2, 0, 2.5]); plt.axis('equal'); plt.show();

||x||_2 =  2.080865204668481
||xra||_2 =  2.0808652046684806
```

```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/numpy/core/_as
```

$$\text{return array}(a, \text{dtype}, \text{copy=False}, \text{order=order}, \text{subok=True})$$

```
||xro||_2 =  2.080865204668481
```

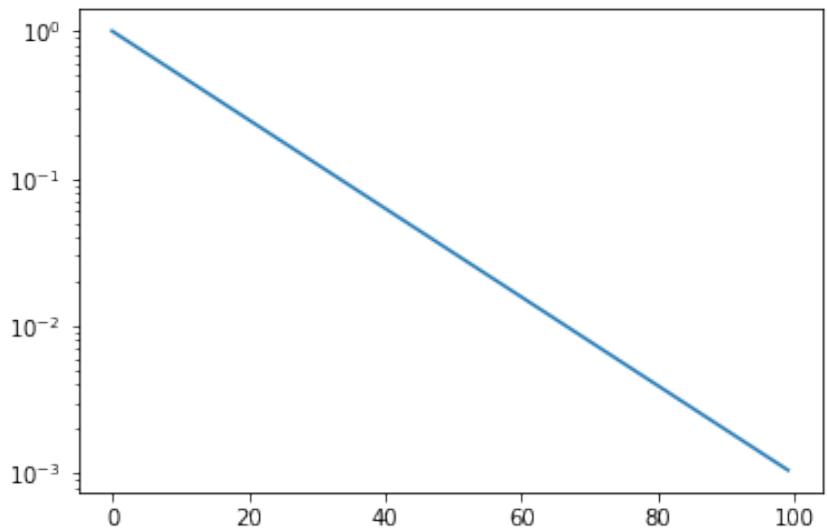


Torna al par. 1.3.4.1

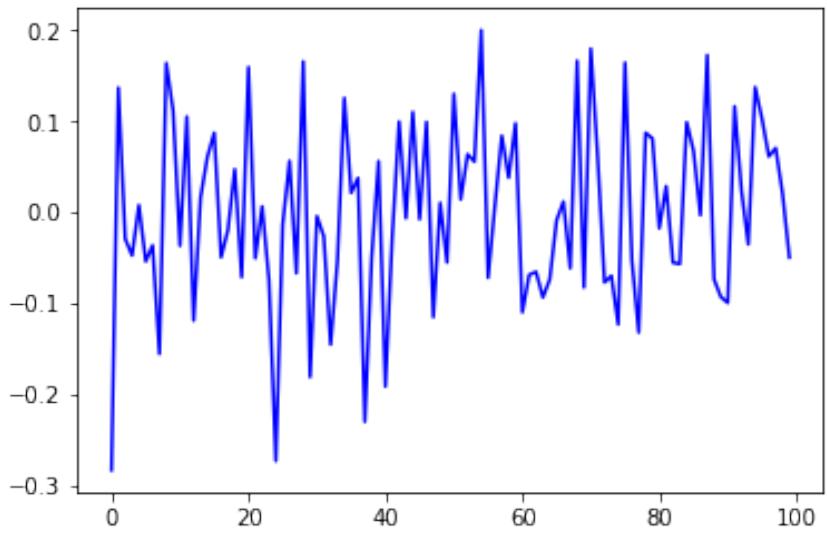
D.5 Rumore e componenti singolari

```
In [13]: n = 100
S = np.diag(2.**((0.1*np.arange(0,-n,-1))))
U,R = np.linalg.qr(np.random.randn(n,n))
V,R = np.linalg.qr(np.random.randn(n,n))
plt.figure(1); plt.semilogy(np.diag(S))
A = U @ S @ (V.T)
K2_A = np.linalg.cond(A,2);
print("numero di condizionamento di A = ", K2_A)
```

numero di condizionamento di A = 955.4257833336687

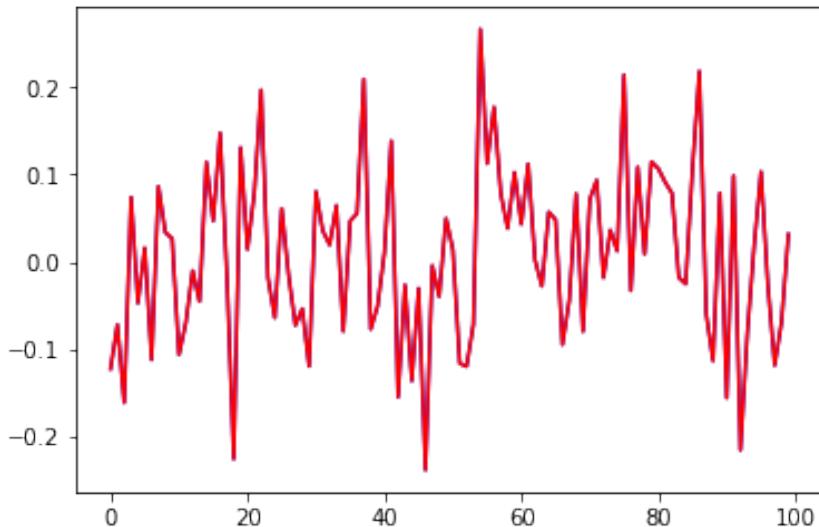


```
In [14]: x_vero = V[:,0]
b = A @ x_vero
plt.figure(2); plt.plot(b, 'b-'); plt.show()
```

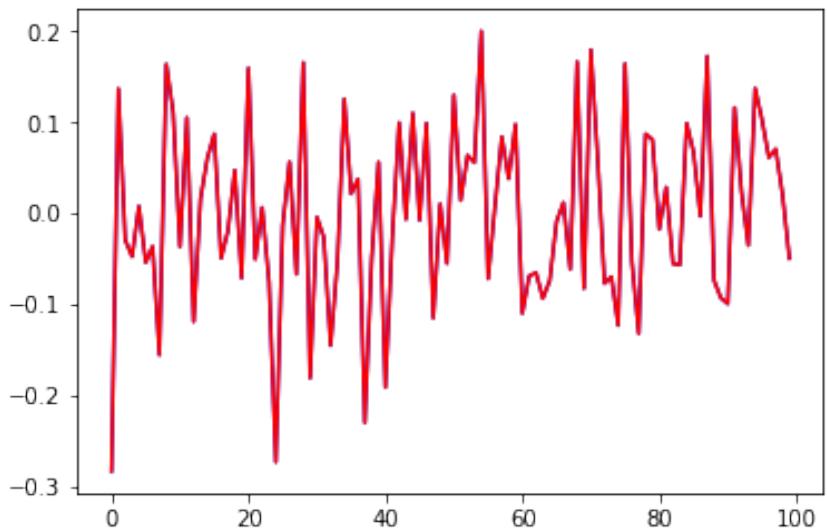


```
In [15]: x_LU = np.linalg.solve(A, b)
print("errore relativo LU = ", \
      np.linalg.norm(x_LU - x_vero, 2)/np.linalg.norm(x_vero, 2))
plt.figure(3); plt.plot(x_vero,'b-'); plt.plot(x_LU,'r'); plt.show()
```

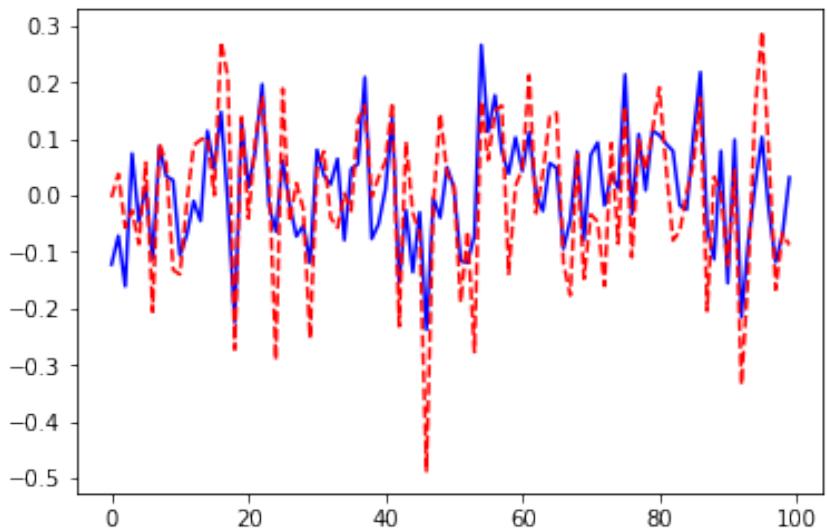
errore relativo LU = 8.999793543790034e-14



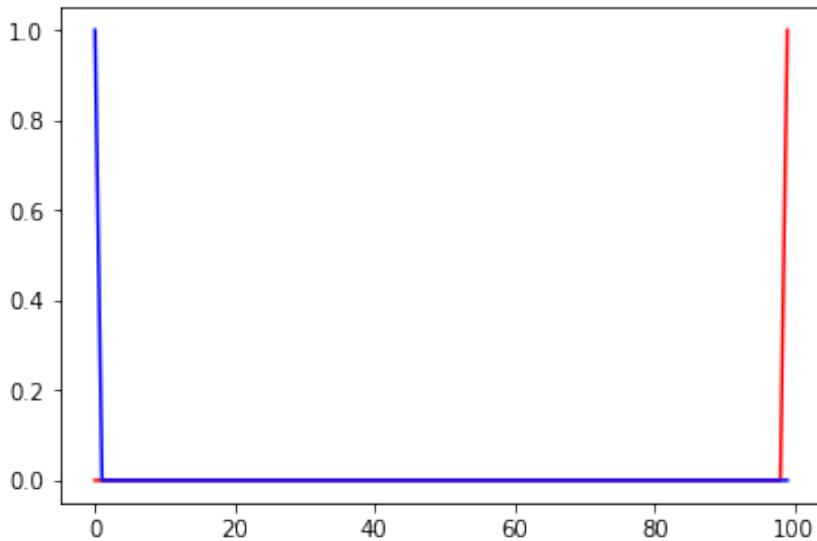
```
In [16]: ncv = n-1
delta_b = 0.001*U[:,ncv]
br = b + delta_b
plt.figure(3); plt.plot(b,'b-'); plt.plot(br,'r'); plt.show()
x_LU = np.linalg.solve(A, br)
print("errore relativo LU = ", \
      np.linalg.norm(x_LU - x_vero, 2)/np.linalg.norm(x_vero, 2))
plt.figure(4); plt.plot(x_vero,'b-'); plt.plot(x_LU,'r--'); plt.show()
```



errore relativo LU = 0.9554257833336502



```
In [17]: plt.figure(5); plt.plot(U.T@delta_b*1.e3, 'r-'); plt.plot(U.T@b, 'b-'); plt.show()
```



Torna al par. 1.4.4

D.6 Confronto algoritmi per il problema dei minimi quadrati

```
In [18]: # confronto tra algoritmi per i minimi quadrati discreti:  
coord = np.arange(50.,0.,-1)  
nc = len(coord)  
# "x_veri" - valori veri dei coefficienti del polinomio che genera i dati osservati  
x_veri = np.array([[2.987654321, 5.987654321, 4.987654321, 1.987654321, 3.987654321  
nx = len(x_veri)  
gradopol = 5 # "gradopol" - grado del polinomio che genera i dati osservati:  
x_veri = x_veri[:,range(nx-(gradopol+1),nx)] # NB: prendo solo quelli fino a "gradopol"
```

```
In [19]: # "A" - matrice del modello lineare:  
A = np.vander(coord)  
print("K_Aint = ",np.linalg.cond(A,2))  
A = A[:,range(nc-(gradopol+1),nc)]  
U,S,V = np.linalg.svd(A)  
print("K_Als = ",S[0]/S[-1])  
print("K_ATA = ",np.linalg.cond(A.T@A,2))  
U,S,V = np.linalg.svd(A.T@A)
```

```
K_Aint = 7.048659608320864e+90
K_Als = 761242009.2355455
K_ATA = 5.794893965912059e+17
```

```
In [20]: # caso di misure perfette :
    b = np.atleast_2d(np.polyval(x_veri,coord)).T
    Qh,Rh = np.linalg.qr(A.copy())
    x1 = np.linalg.solve( Rh , Qh.T@b )
    Qm,Rm = qr.MGS(A.copy())
    x2 = np.linalg.solve( Rm , Qm.T@b )
    x3 = np.linalg.solve( A.T@A , A.T@b )
    print('risultato con misure perfette:')
    print((x1-x_veri)/x_veri)
    print(" ")
    print((x2-x_veri)/x_veri)
    print(" ")
    print((x3-x_veri)/x_veri)
    print(" ")
    print("=> QR_Householder e' nettamente il migliore !")

risultato con misure perfette:
[[ 7.67504872e-14]
 [-2.17820296e-11]
 [ 4.32987875e-10]
 [-2.87833942e-08]
 [ 5.85815895e-08]
 [-1.15690112e-07]]

[[ 4.01917905e-11]
 [-1.31036276e-08]
 [ 3.05016286e-07]
 [-2.47261178e-05]
 [ 6.50275158e-05]
 [-2.03747889e-04]]

[[ -1.02937896e-10]
 [ 3.36173490e-08]
 [-7.83729987e-07]
 [ 6.35790670e-05]
 [-1.66825773e-04]
 [ 5.15995257e-04]]

=> QR_Householder e' nettamente il migliore !
```

```
In [21]: # calcolo i residui:  
print(np.linalg.norm(A@x1 - b, 2))  
print(np.linalg.norm(A@x2 - b, 2))  
print(np.linalg.norm(A@x3 - b, 2))
```

```
4.032686826617651e-06  
0.0005352660704647317  
0.0013736323884751358
```

```
In [22]: # caso di misure rumorose :  
stderr = 1.e-3;  
br = np.matrix(np.polyval(x_veri,coord)).T + stderr*np.matrix(np.random.randn(nc,1))  
Qh,Rh = np.linalg.qr(A.copy())  
x1 = np.linalg.solve( Rh , Qh.T@br )  
Qm,Rm = qr.MGS(A.copy())  
x2 = np.linalg.solve( Rm , Qm.T@br )  
x3 = np.linalg.solve( A.T@A , A.T@br )  
print('risultato con misure rumorose: stderr = ', stderr)  
print((x1-x_veri)/x_veri)  
print(" ")  
print((x2-x_veri)/x_veri)  
print(" ")  
print((x3-x_veri)/x_veri)  
print(" ")  
print("=> per effetto del rumore nelle misure, i tre algoritmi sono adesso equivalenti")
```

```
risultato con misure rumorose: stderr =  0.001
```

```
[[ -1.39720255e-10]  
[ 4.25880930e-08]  
[ -9.23841272e-07]  
[ 7.05956219e-05]  
[ -1.79109177e-04]  
[ 4.08027503e-04]]
```

```
[[ -9.96103794e-11]  
[ 2.95076762e-08]  
[ -6.19290983e-07]  
[ 4.59013769e-05]  
[ -1.14151079e-04]  
[ 2.04445962e-04]]
```

```
[[ -1.12144571e-10]  
[ 3.33883283e-08]  
[ -7.03827589e-07]  
[ 5.21857569e-05]]
```

```
[ -1.28921392e-04]  
[ 2.45364559e-04]]
```

=> per effetto del rumore nelle misure, i tre algoritmi sono adesso equivalenti !

```
In [23]: # verifico l'equivalenza del residuo con la deviazione standard dell'errore "sigma"  
print(np.linalg.norm(A@x1 - br, 2))  
print(np.linalg.norm(A@x2 - br, 2))  
print(np.linalg.norm(A@x3 - br, 2))  
# se conoscessi "b", la stima di "sigma" sarebbe piu' precisa:  
print(np.linalg.norm(A@x1 - b, 2))  
print(np.linalg.norm(A@x2 - b, 2))  
print(np.linalg.norm(A@x3 - b, 2))
```

```
0.0071950570912077795  
0.0072150236794666376  
0.007205964718808048  
0.002919015095584948  
0.002647294881287113  
0.002751713938930293
```

```
In [24]: # vediamo che risultati darebbe l'interpolazione :  
print("interpolazione con misure perfette (primi campioni) :")  
xif = np.linalg.solve( A[range((nc-1)-gradopol,nc),:], b[range((nc-1)-gradopol,nc)] )  
print((xif-x_veri)/x_veri)  
print("interpolazione con misure perfette (ultimi campioni) :")  
xil = np.linalg.solve( A[0:gradopol+1,:], b[0:gradopol+1] )  
print((xil-x_veri)/x_veri)
```

```
interpolazione con misure perfette (primi campioni) :  
[[ -2.13690451e-15]  
[ 1.31149749e-13]  
[ -5.24645343e-13]  
[ 7.15074250e-12]  
[ -3.39526754e-12]  
[ 2.47877495e-12]]  
interpolazione con misure perfette (ultimi campioni) :  
[[ 2.77224487e-09]  
[ -1.65565374e-06]  
[ 7.85462459e-05]  
[ -1.50879274e-02]  
[ 1.18618836e-01]  
[ -1.69566131e+00]]
```

```
In [25]: # il motivo di questa differenza è il condizionamento della matrice:
print("condizionamento di A (primi campioni) = ", np.linalg.cond(A[range((nc-1)-gradopol+1, nc), :])
print("condizionamento di A (ultimi campioni) = ", np.linalg.cond(A[0:gradopol+1, :])

condizionamento di A (primi campioni) = 731200.9387889808
condizionamento di A (ultimi campioni) = 1.9540968000705612e+16

In [26]: print("interpolazione con misure rumorose (primi campioni) : stderr=", stderr)
xifr = np.linalg.solve( A[range((nc-1)-gradopol, nc), :], br[range((nc-1)-gradopol, nc)])
print((xifr-x_veri)/x_veri)
print("l'interpolazione e' piu' sensibile al rumore di quanto lo siano i minimi quadrati")
print("(i minimi quadrati risolvono un sistema sovradeterminato, in cui la ridondanza di

interpolazione con misure rumorose (primi campioni) : stderr= 0.001
[[ 3.68643386e-05]
 [-1.59222969e-03]
 [ 5.10285157e-03]
 [-6.09086918e-02]
 [ 2.66741223e-02]
 [-1.82952626e-02]]
l'interpolazione e' piu' sensibile al rumore di quanto lo siano i minimi quadrati!
(i minimi quadrati risolvono un sistema sovradeterminato, in cui la ridondanza di
```

Torna al par. [3.3.5](#)

In [27] :

Appendice E

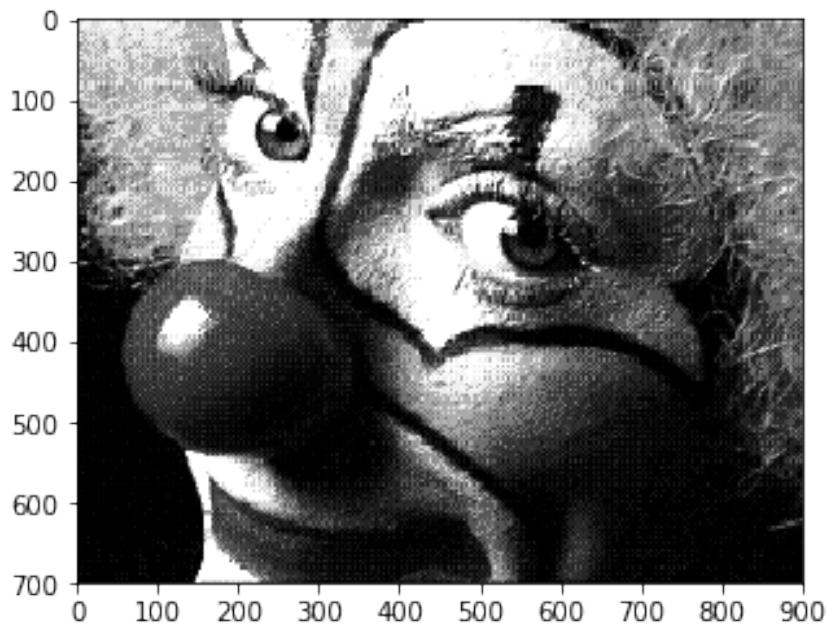
Regolarizzazione

E.1 Esempio SVD troncata di un'immagine

```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matplotlib inline"
import numpy as np
import matplotlib.pyplot as plt
import PIL.Image as Image

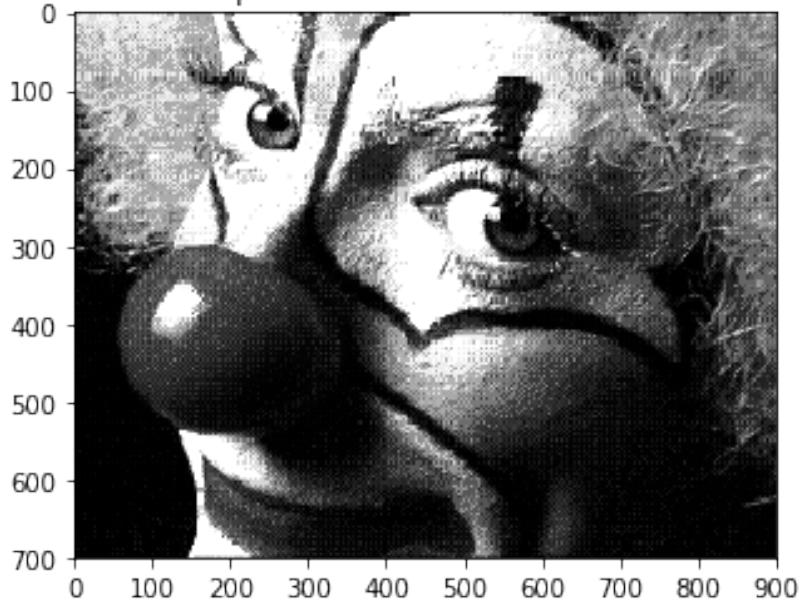
In [2]: im = Image.open("clown.png")
im = im.convert("L")
print(im.format, im.size, im.mode)
#im.save("prova.png", "L")
X = np.reshape(np.matrix(im.getdata()), (im.size[1],im.size[0]))
X = X[100:800,160:1060]
print("X.shape =", X.shape)
plt.figure(1); plt.imshow(X, 'gray'); plt.show()

None (1200, 900) L
X.shape = (700, 900)
```

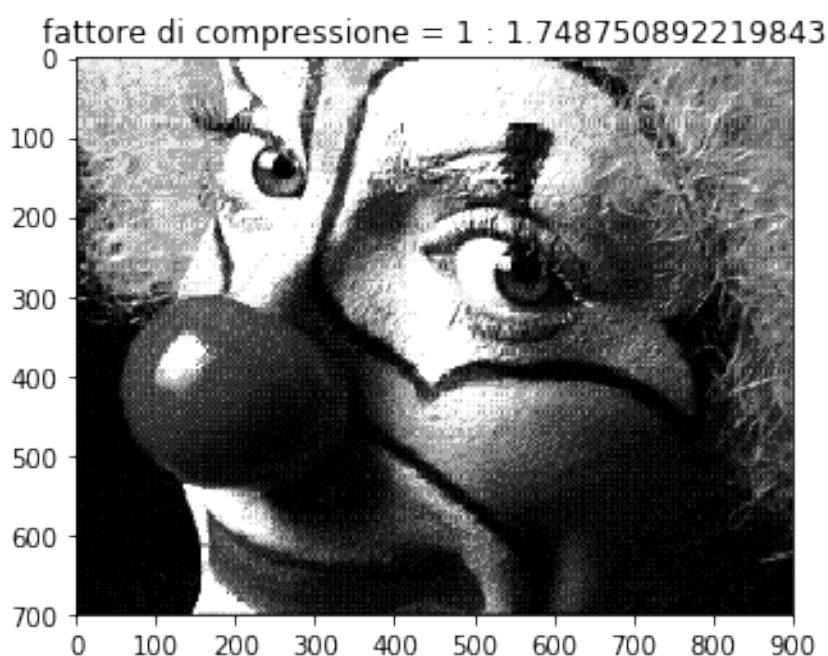


```
In [3]: [U,S,V]=np.linalg.svd(X,full_matrices=0); V = V.T
S = np.diag(S)
i = 0
for k in [700, 200, 50, 30, 10]:
    i = i+1
    m = U.shape[0]; n = V.shape[1]
    plt.figure(i); plt.imshow(U[:,0:k]*S[0:k,0:k]*V[:,0:k].T,'gray')
    plt.title('fattore di compressione = 1 : ' + str(float(m*n)/float(m*k+k*k*n)))
    print('fattore di compressione = 1 : ', float(m*n)/float(m*k+k*k*n))
#endfor
```

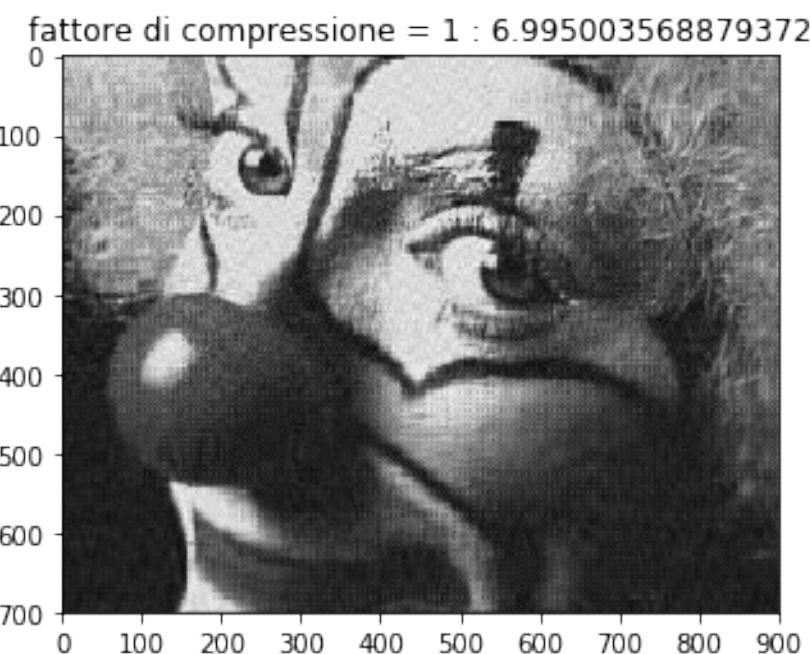
fattore di compressione = 1 : 0.49964311206281226



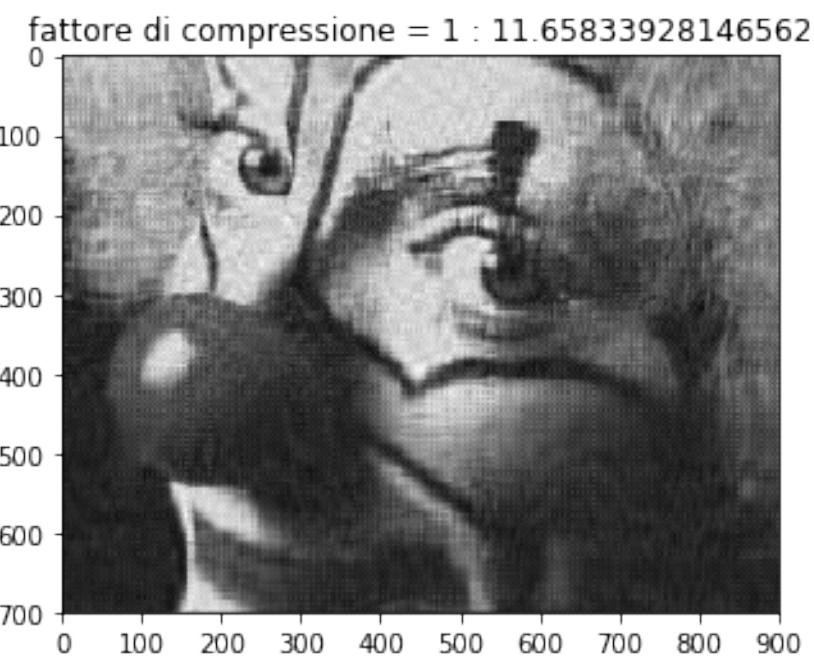
fattore di compressione = 1 : 0.49964311206281226



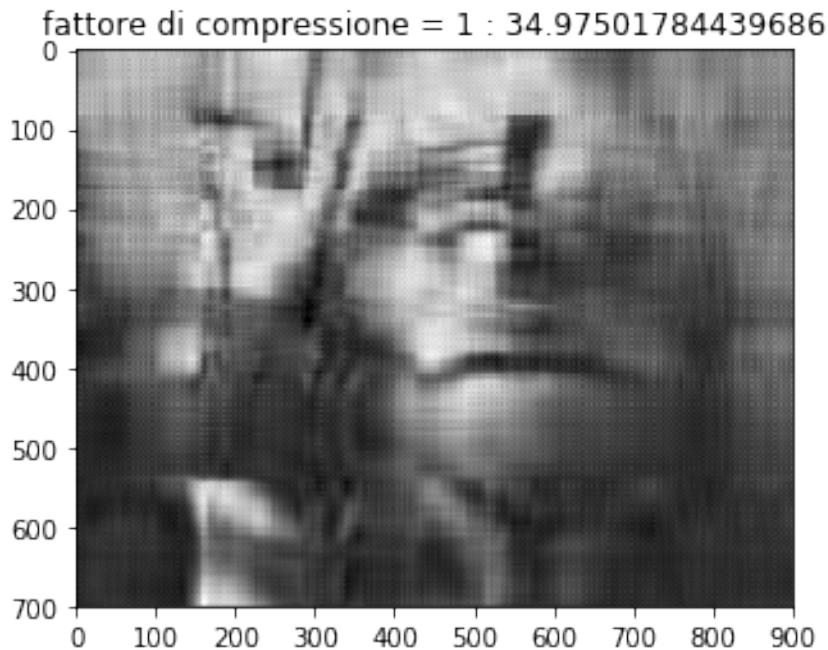
fattore di compressione = 1 : 1.748750892219843



fattore di compressione = 1 : 6.995003568879372



fattore di compressione = 1 : 11.65833928146562



fattore di compressione = 1 : 34.97501784439686

Torna al par [1.4.5](#)

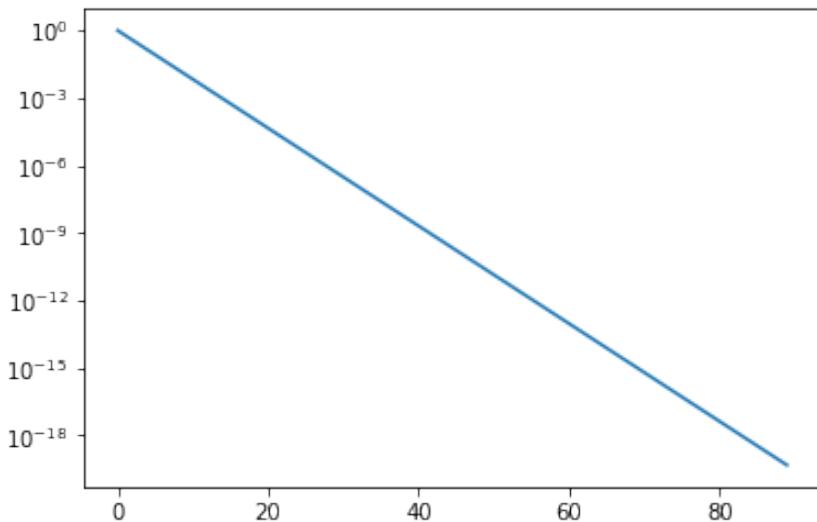
E.2 Analisi della regolarizzazione con TSVD e Tichonov

```
In [4]: # Attenzione: la routine "svd" di NumPy usa una definizione di SVD un po' diversa:
n = 100
A = np.matrix(np.random.randn(n,n))
U,S,V = np.linalg.svd(A)
#V = V.T # NB: per riportarsi a quella solita, basta scommentare questa istruzione
S = np.matrix(np.diag(S))
print(np.linalg.norm(A - U@S@V))
print(np.linalg.norm(A - U@S@V.T))
```

2.6291808756960314e-13

141.42938053091876

```
In [5]: # risoluzione di un sistema lineare fortemente mal-condizionato
esempio = 2
if esempio==1:
    n = 90;
    x_vero = np.zeros(n)
    x_vero[5:20] = 0.75*np.ones(15)
    x_vero[25:28] = 0.2*np.ones(3)
    x_vero[35:85] = np.sin(np.array(np.pi*np.arange(0.,1.,0.02)))
    S = np.diag(2.**((0.5*np.arange(0,-n,-1))))
    stde = 0.e-6 # deviazione standard del rumore additivo aggiunto
elif esempio==2:
    n = 90;
    m = n #1000;
    x_vero = np.ones(n) # soluzione vera (imposta da noi)
    S = np.diag(np.exp(0.5*np.arange(0,-n,-1)))
    stde = 0.0; # deviazione standard del rumore additivo aggiunto
else:
    print("esempio errato!")
#endif
x_vero = np.asmatrix(x_vero).T
S = np.asmatrix(S)
U,R = np.linalg.qr(np.matrix(np.random.randn(m,n)))
V,R = np.linalg.qr(np.matrix(np.random.randn(n,n)))
plt.figure(1); plt.semilogy(np.diag(S)); plt.show()
A = U@S@V.T;
if m == n:
    K2_A = np.linalg.cond(A,2);
    print("numero di condizionamento di A = ", K2_A)
#endif
```



```
numero di condizionamento di A = 1.211283131340705e+19
```

```
In [6]: if m == n:  
    print(np.linalg.norm(U.T@U - np.eye(n)))  
    print(np.linalg.norm(V.T@V - np.eye(n)))  
    Sinv = np.matrix(np.diag(1./np.diag(S[:, :])))  
    print(np.linalg.norm(Sinv*S - np.eye(n)))  
    psinv_A = V@Sinv@U.T  
    print(np.linalg.norm(psinv_A@A - np.eye(n)))  
    print(np.linalg.norm(np.linalg.inv(A)@A - np.eye(n)))  
    print("Notare che, essendoci valori singolari <1.e-16, la 'psinv' puo' essere inesatta")  
#endif
```

```
8.400230681027567e-15
```

```
8.798087985556654e-15
```

```
4.002966042486721e-16
```

```
2150.373363480594
```

```
1094.2378045273217
```

```
Notare che, essendoci valori singolari <1.e-16, la 'psinv' puo' essere meno accurata
```

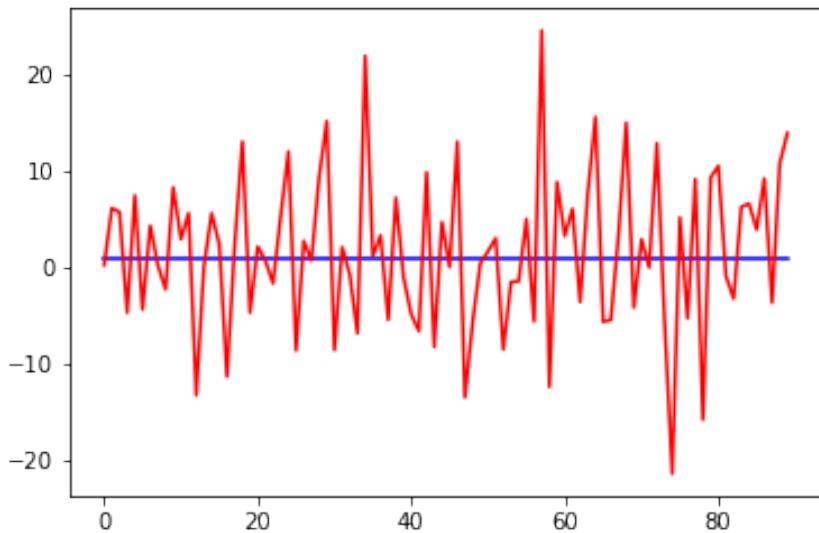
```
In [7]: print(A.shape)  
print(x_vero.shape)
```

```
(90, 90)
```

```
(90, 1)
```

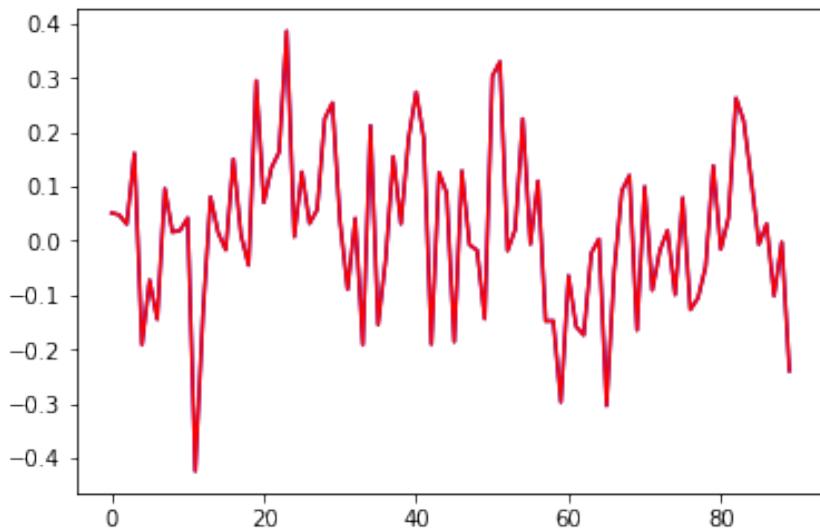
```
In [8]: # soluzione senza rumore aggiunto:  
#b = np.matrix(randn(m)).T  
b = A @ x_vero  
if m == n:  
    x_LU = np.linalg.solve(A, b)  
    print(np.linalg.norm(x_LU - x_vero, 2) / np.linalg.norm(x_vero, 2))  
    plt.figure(2); plt.plot(x_vero, 'b-'); plt.plot(x_LU, 'r'); plt.show()  
#endif
```

```
8.123830223037158
```

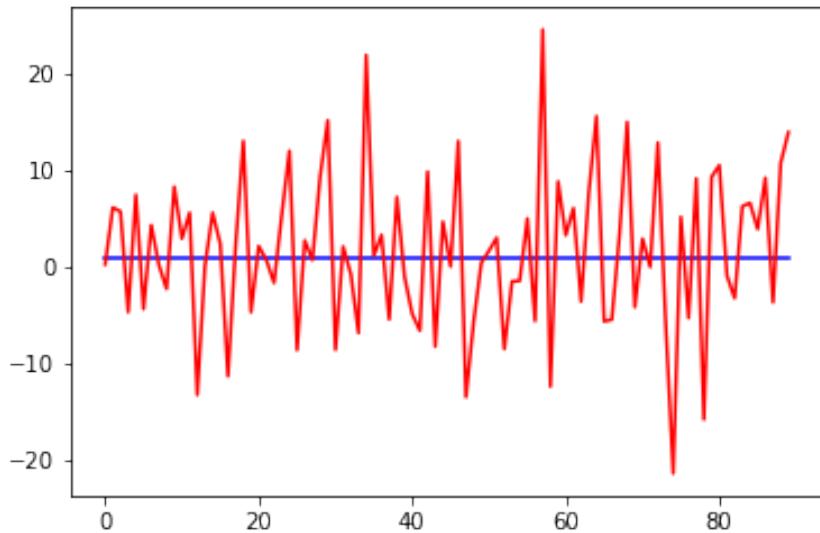


```
In [9]: # soluzione con rumore aggiunto:  
    print("stde = ", stde)  
    br = b + stde*np.matrix(np.random.randn(m,1))  
    plt.figure(3); plt.plot(b,'b-'); plt.plot(br,'r'); plt.show()  
    if m == n:  
        x_LU = np.linalg.solve(A, br)  
        n2err_LU = np.linalg.norm(x_LU - x_vero, 2) / np.linalg.norm(x_vero, 2);  
        print("n2err_LU = ",n2err_LU)  
        plt.figure(4); plt.plot(x_vero,'b-'); plt.plot(x_LU,'r'); plt.show()  
    #endif
```

```
stde =  0.0
```



n2err_LU = 8.123830223037158



```

In [10]: analisi_TSVD = False
confronto_TSVD_Tichonov = False

In [11]: I = np.eye(A.shape[1])
n2res_TSVD_hist = np.zeros(n);
n2restrunc_TSVD_hist = np.zeros(n);
n2rescompl_TSVD_hist = np.zeros(n);
n2bias_TSVD_hist = np.zeros(n);
n2res_Tich_hist = np.zeros(n);
n2resvero_TSVD_hist = np.zeros(n);
n2reserr_TSVD_hist = np.zeros(n);
n2reserr_Tich_hist = np.zeros(n);
n2err_TSVD_hist = np.zeros(n);
n2err_Tich_hist = np.zeros(n);
n2A_TSVD_hist = np.zeros(n);
varf_TSVD_hist = np.zeros(n);
varf_Tich_hist = np.zeros(n);
ifig = 5
for k in range(0,n):
    #>>> CALCOLO:
    # soluzione con SVD Troncata (TSVD):
    Sinv = np.matrix(np.diag(1./np.diag(S[0:k+1,0:k+1])))
    R_delta_TSVD = V[:,0:k+1]@Sinv@U[:,0:k+1].T
    x_TSVD = R_delta_TSVD @ br
    #n2xTSVD = norm(x_TSVD, 2)
    #
    # soluzione con metodo di Tichonov:
    if k < n-1:
        sqrt_delta = S[k+1,k+1]
    else:
        sqrt_delta = 0.0
    #endif
    calcolo_Tichonov_con_eq_normali = False # "True":soluzione derivata da equazioni
    if calcolo_Tichonov_con_eq_normali:
        x_Tich = np.linalg.solve(A.T@A + (sqrt_delta**2)*I, A.T@br)
    else:
        AT = np.vstack((A,sqrt_delta*I))
        brT = np.vstack((br,np.zeros((I.shape[0],1))))
        [Q,R] = np.linalg.qr(AT)
        x_Tich = np.linalg.solve(R, Q.T@brT)
        [UT,ST,VT] = np.linalg.svd(AT)
        #print("sqrt_delta = ",sqrt_delta," K_ATich = ", ST[0]/ST[-1]," (s_1=",ST[0]
    #endif
    #
    #>>> ANALISI (per la scelta ottimale di "delta"):
    M_delta_TSVD = A @ R_delta_TSVD
    #print("k=",k," sqrt_delta=",sqrt_delta," TSVD: trace(M_delta_TSVD) = ",np.sum(varf_TSVD_hist[k] = 2*stde**2/m * np.sum(np.diag(M_delta_TSVD)))
    #print("varf_hist[k] = ",varf_hist[k])

```

```

# scomposizione diadica: A = A_TSVD + A_mc
A_TSVD = U[:,0:k+1]@S[0:k+1,0:k+1]@V[:,0:k+1].T # matrice di rango "k"
A_compl = U[:,k+1:]@S[k+1:,k+1:]@V[:,k+1: ].T # matrice di rango "n-k"
n2A_TSVD = np.linalg.norm(A_TSVD, 2)
n2A_TSVD_hist[k] = n2A_TSVD
n2bias_TSVD = np.linalg.norm(A@x_TSVD - b, 2)
n2bias_TSVD_hist[k] = n2bias_TSVD
n2res_TSVD = np.linalg.norm(A@x_TSVD - br, 2)
n2res_TSVD_hist[k] = n2res_TSVD
n2restrunc_TSVD = np.linalg.norm(A_TSVD@x_TSVD - br, 2)
n2restrunc_TSVD_hist[k] = n2restrunc_TSVD
n2rescompl_TSVD = np.linalg.norm(A_compl@x_TSVD, 2)
n2rescompl_TSVD_hist[k] = n2rescompl_TSVD
n2res_Tich = np.linalg.norm(A@x_Tich - br, 2)
if False: # se "True", verifica che x_TSVD sta tutto nello spazio sorgente di A_
    n2resT = np.linalg.norm(A_TSVD@x_TSVD - br, 2)
    print(n2res_TSVD - n2resT)
#endif
n2res_Tich_hist[k] = n2res_Tich
n2reserr_TSVD = np.linalg.norm(A_TSVD@(x_vero - x_TSVD), 2) # e' la componente
n2reserr_TSVD_hist[k] = n2reserr_TSVD
if False:
    n2reserr_Tich = np.linalg.norm(A_TSVD@(x_vero - x_Tich), 2) # e' la componente
    n2reserr_Tich_hist[k] = n2reserr_Tich
#endif
n2resvero_TSVD = np.linalg.norm(A_TSVD@x_vero - br, 2)
n2resvero_TSVD_hist[k] = n2resvero_TSVD
n2err_TSVD = np.linalg.norm(x_TSVD - x_vero, 2) / np.linalg.norm(x_vero, 2);
n2err_TSVD_hist[k] = n2err_TSVD
n2err_Tich = np.linalg.norm(x_Tich - x_vero, 2) / np.linalg.norm(x_vero, 2);
n2err_Tich_hist[k] = n2err_Tich
if ((k+1) % 10) == 0:
    # scommentare l'istruzione seguente se si vogliono graficare le soluzioni vere
    #figure(ifig); clf, hold(True), title("dim="+str(k+1)), plot(x_vero, 'b-'); plt.show()
    ifig += 1
#endif
#endiffor

```

In [12]: # residuo ed errore per la TSDV: hanno entrambi un minimo, ma per valori di "k" magici non esiste

```

print("curva 'r--': livello di rumore nei dati (membro destro 'b')")
print("curva 'b.' : residuo TSVD || A*x_TSVD - br ||")
print("curva 'm+' : errore TSVD || x_TSVD - x_vero ||")
plt.figure(100); plt.axis([0, n, 1.e-20, 1.e2]); #plt.show()
if stde > 0.0:
    plt.semilogy([1, n],[stde, stde], 'r--') # livello di rumore nei dati (membro destro 'b')
else:
    plt.semilogy([1, n],[1e-16, 1e-16], 'r--')
#endif

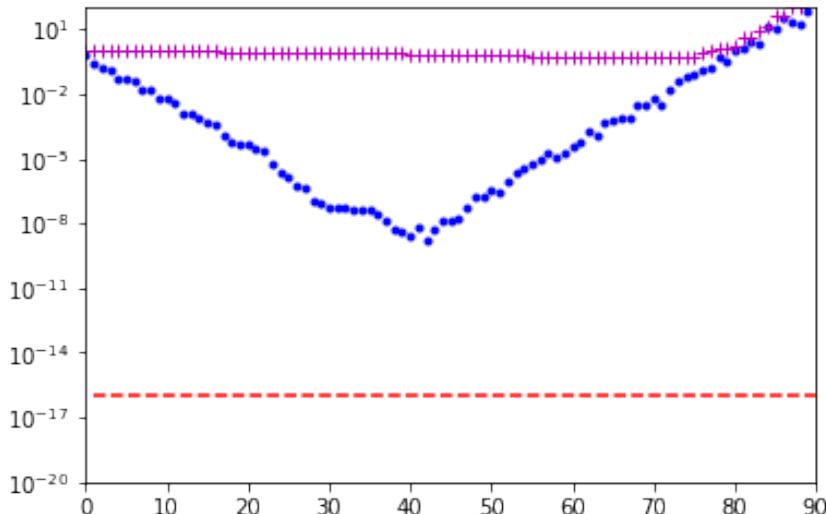
```

```

plt.semilogy(n2res_TSVD_hist,'b.');// e' il residuo
plt.semilogy(n2err_TSVD_hist,'m+');// e' l'errore

curva 'r--': livello di rumore nei dati (membro destro 'b')
curva 'b.' : residuo TSVD || A*x_TSVD - br ||
curva 'm+' : errore TSVD || x_TSVD - x_vero ||

```



```

In [13]: # il "residuo TSVD" e' composto da due addendi: "A_TSVD*x_TSVD - br" e "A_compl*x_"
         # grazie alla particolare decomposizione di A operata:
print("curva 'r--': livello di rumore nei dati (membro destro 'b')")
print("curva 'b.' : residuo TSVD || A*x_TSVD - br ||")
print("curva 'm+' : residuo TSVD troncato || A_TSVD*x_TSVD - br ||")
print("curva 'c.' : residuo TSVD complementare || A_compl*x_TSVD ||")
plt.figure(100); plt.axis([0, n, 1.e-20, 1.e2]); #plt.show()
if stde > 0.0:
    plt.semilogy([1, n],[stde, stde],'r--') # livello di rumore nei dati (membro de
else:
    plt.semilogy([1, n],[1e-16, 1e-16],'r--')
#endif
plt.semilogy(n2res_TSVD_hist,'b.');// e' il residuo
plt.semilogy(n2restrunc_TSVD_hist,'m+');
plt.semilogy(n2rescompl_TSVD_hist,'c.');

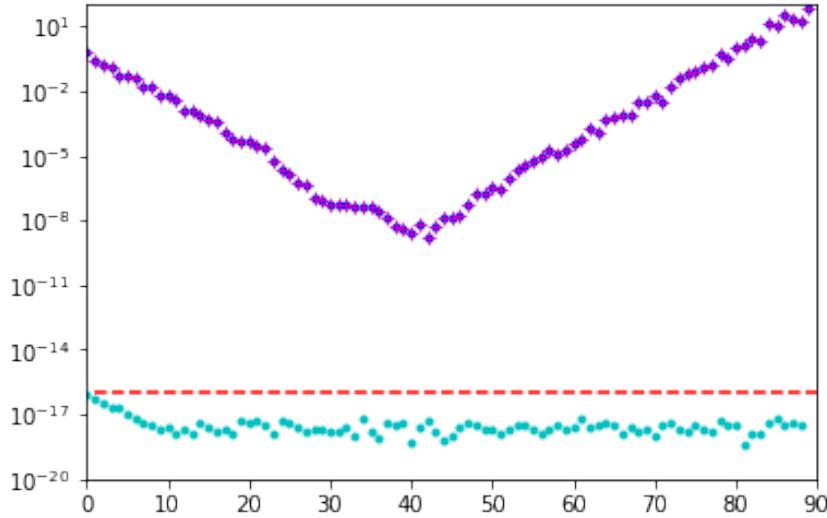
curva 'r--': livello di rumore nei dati (membro destro 'b')
curva 'b.' : residuo TSVD || A*x_TSVD - br ||

```

```

curva 'm+' : residuo TSVD troncato || A_TSVD*x_TSVD - br ||
curva 'c.' : residuo TSVD complementare || A_compl*x_TSVD ||

```



```

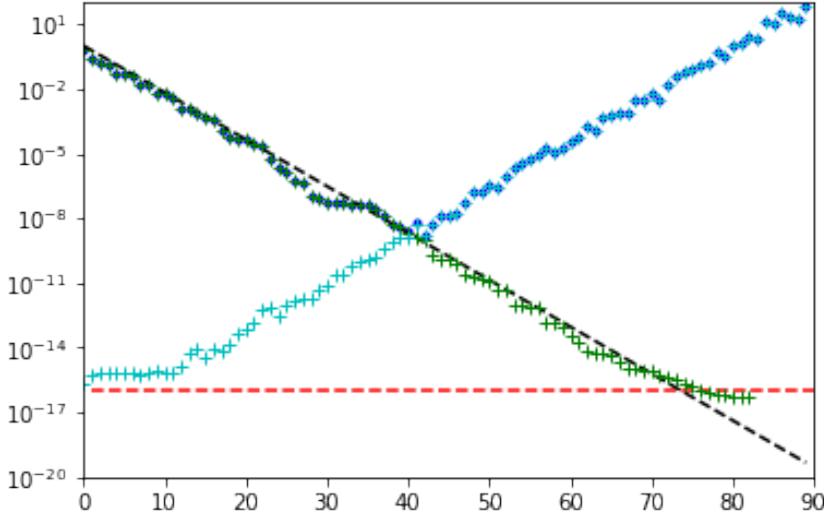
In [14]: # la risalita del residuo TSVD e' dovuta al crescente malcondizionamento di A_TSVD
          # lo vedo dall'immagine troncata dell'errore.
print("curva 'k--' : valori singolari di A")
print("curva 'r--': livello di rumore nei dati (membro destro 'b')")
print("curva 'b.' : residuo TSVD troncato || A_TSVD*x_TSVD - br || =(approx)= || A
print("curva 'g+' : residuo troncato soluzione vera || A_TSVD*x_vero - br ||")
print("curva 'c+' : immagine troncata dell'errore || A_TSVD*(x_TSVD - x_vero) ||")
plt.figure(100); plt.axis([0, n, 1.e-20, 1.e2]); #plt.show()
if stde > 0.0:
    plt.semilogy([1, n],[stde, stde], 'r--') # livello di rumore nei dati (membro de
else:
    plt.semilogy([1, n],[1e-16, 1e-16], 'r--')
#endif
plt.semilogy(n2restrunc_TSVD_hist,'b.');// e' il residuo
plt.semilogy(np.diag(S),'k--')
plt.semilogy(n2resvero_TSVD_hist,'g+');// e' il residuo con la soluzione vera
plt.semilogy(n2reserr_TSVD_hist,'c+');// cresce perche' il modello incorpora semp
curva 'k--' : valori singolari di A
curva 'r--': livello di rumore nei dati (membro destro 'b')
curva 'b.' : residuo TSVD troncato || A_TSVD*x_TSVD - br || =(approx)= || A*x_TSVD

```

```

curva 'g+' : residuo troncato soluzione vera || A_TSVD*x_vero - br ||
curva 'c+' : immagine troncata dell'errore || A_TSVD*(x_TSVD - x_vero) ||

```



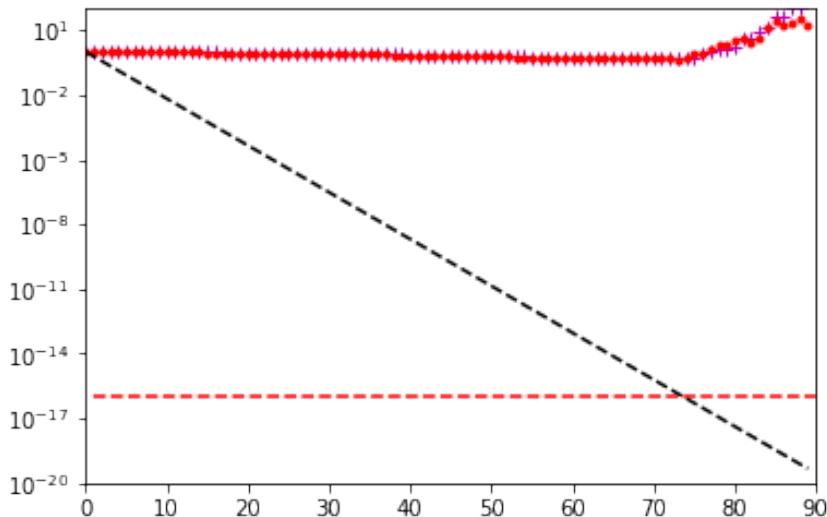
```

In [15]: # confronto con Tichonov: ad un certo punto Tichonov e' migliore
          # perche' il calcolo dei valori singolari non e' piu' accurato
print("curva 'k--' : valori singolari di A")
print("curva 'r--': livello di rumore nei dati (membro destro 'b')")
print("curva 'm+' : errore TSVD || x_TSVD - x_vero ||")
print("curva 'r.' : errore Tich || x_Tich - x_vero ||")
plt.figure(100); plt.axis([0, n, 1.e-20, 1.e2]); #plt.show()
if stde > 0.0:
    plt.semilogy([1, n],[stde, stde], 'r--') # livello di rumore nei dati (membro destro)
else:
    plt.semilogy([1, n],[1e-16, 1e-16], 'r--')
#endif
plt.semilogy(n2err_TSVD_hist, 'm+'); # e' l'errore TSVD
plt.semilogy(n2err_Tich_hist, 'r.');// e' l'errore Tich
plt.semilogy(np.diag(S), 'k--')

curva 'k--' : valori singolari di A
curva 'r--': livello di rumore nei dati (membro destro 'b')
curva 'm+' : errore TSVD || x_TSVD - x_vero ||
curva 'r.' : errore Tich || x_Tich - x_vero ||

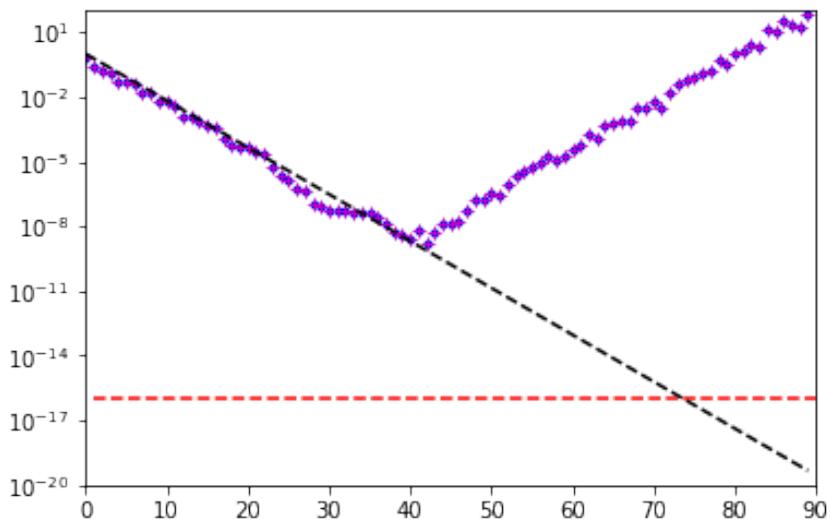
```

```
Out[15]: [<matplotlib.lines.Line2D at 0x7fecec86f7f0>]
```



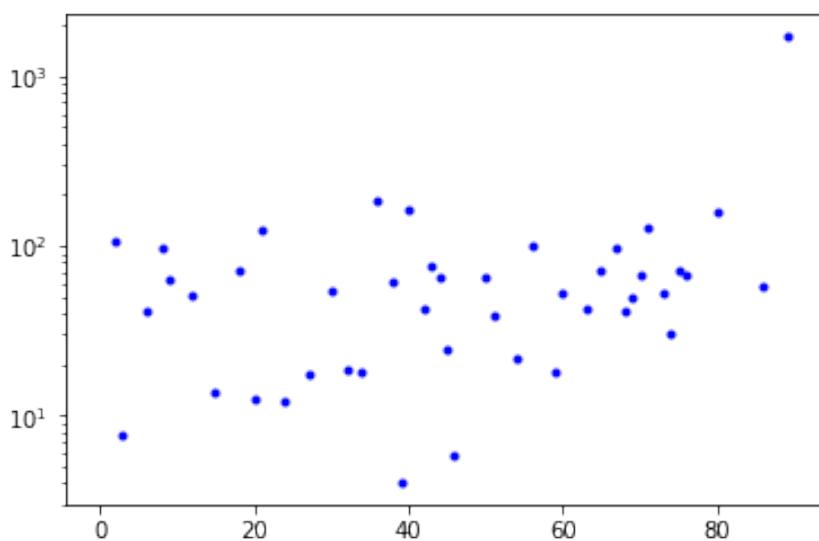
```
In [16]: # scelta del punto di troncamento per TSVD:  
# NB: attenzione, sono le radici quadrate delle quantita' che compaiono nel teorema  
print("curva 'k--' : valori singolari di A")  
print("curva 'r--': deviazione standard del rumore nei dati (membro destro 'b')")  
print("curva 'b.' : sqrt(MSE) TSVD || A*x_TSVD - br ||")  
print("curva 'm+' : sqrt(bias) TSVD || A*x_TSVD - b ||")  
print("curva 'c-' : deviazione standard di modello TSVD: 2*stde**2/m * trace(M_delta_TSVD)  
plt.figure(100); plt.axis([0, n, 1.e-20, 1.e2]); #plt.show()  
if stde > 0.0:  
    plt.semilogy([1, n],[stde, stde], 'r--') # livello di rumore nei dati (membro de  
else:  
    plt.semilogy([1, n],[1e-16, 1e-16], 'r--')  
#endif  
plt.semilogy(n2res_TSVD_hist,'b.');// e' il residuo  
plt.semilogy(n2bias_TSVD_hist,'m+');// e' il bias  
plt.semilogy(np.sqrt(varf_TSVD_hist),'c-');// e' var(f)  
plt.semilogy(np.diag(S), 'k--')  
  
curva 'k--' : valori singolari di A  
curva 'r--': deviazione standard del rumore nei dati (membro destro 'b')  
curva 'b.' : sqrt(MSE) TSVD || A*x_TSVD - br ||  
curva 'm+' : sqrt(bias) TSVD || A*x_TSVD - b ||  
curva 'c-' : deviazione standard di modello TSVD: 2*stde**2/m * trace(M_delta_TSVD)
```

```
Out[16]: [<matplotlib.lines.Line2D at 0x7fecec298eb8>]
```



Torna al par. 4.3

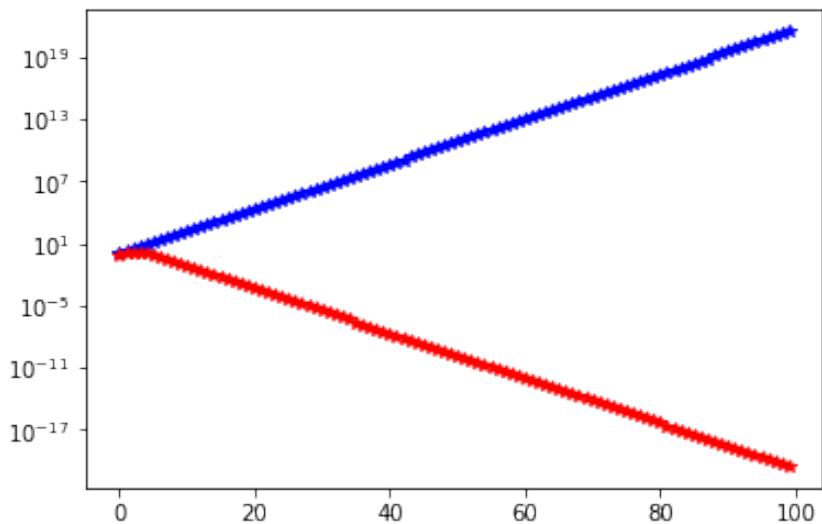
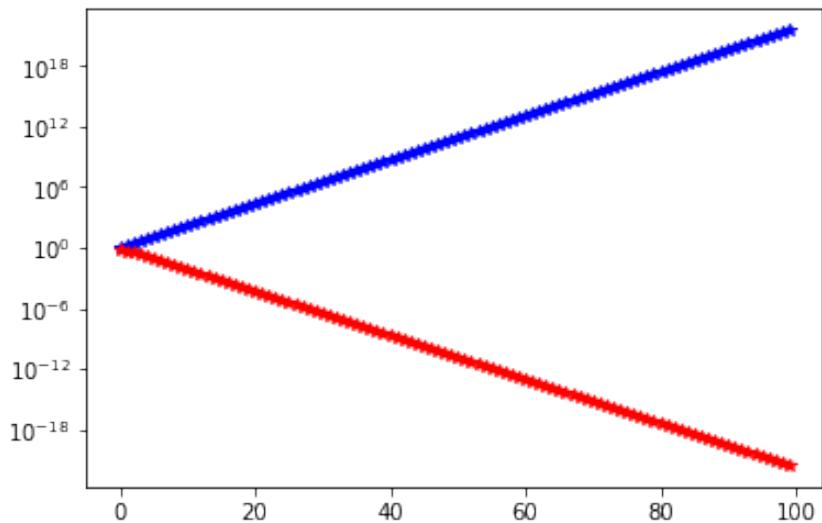
```
In [17]: errpV = V.T@(x_TSVD - x_vero)
plt.figure(200); plt.semilogy(errpV, 'b.');?>
plt.show()
```

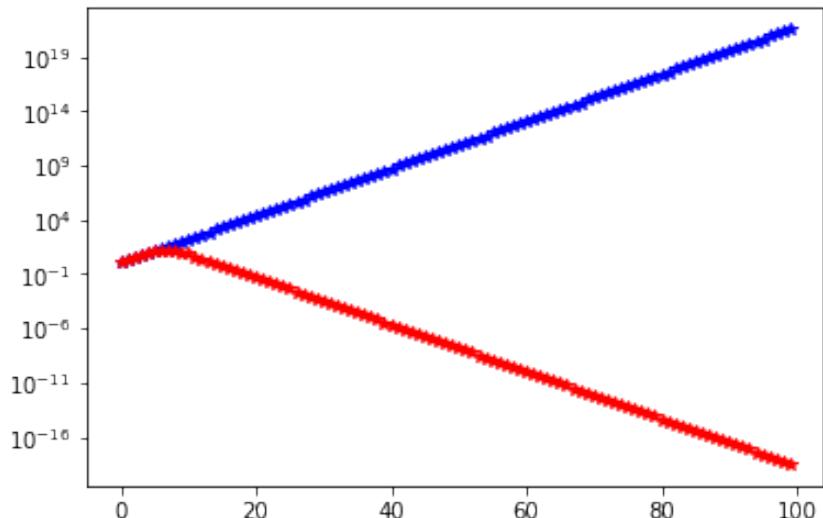
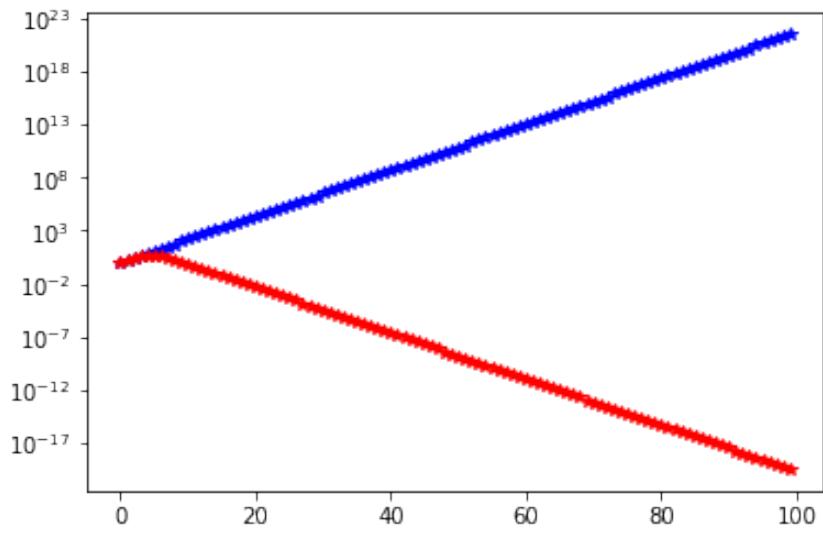


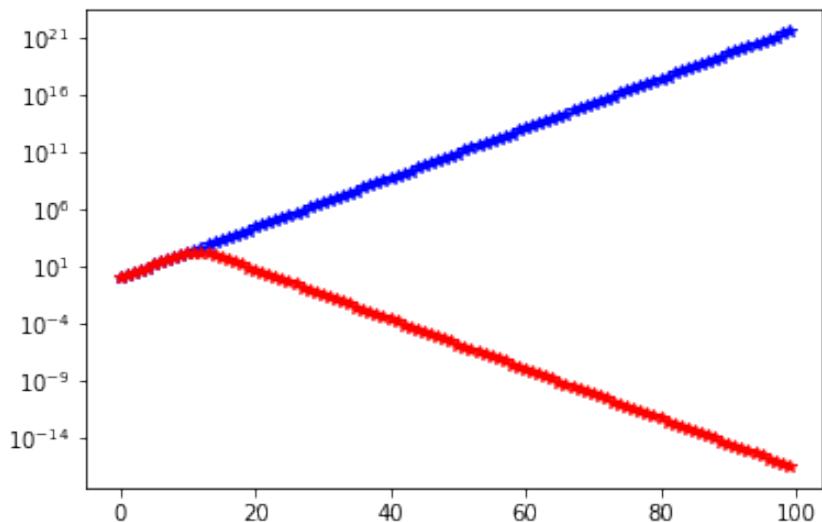
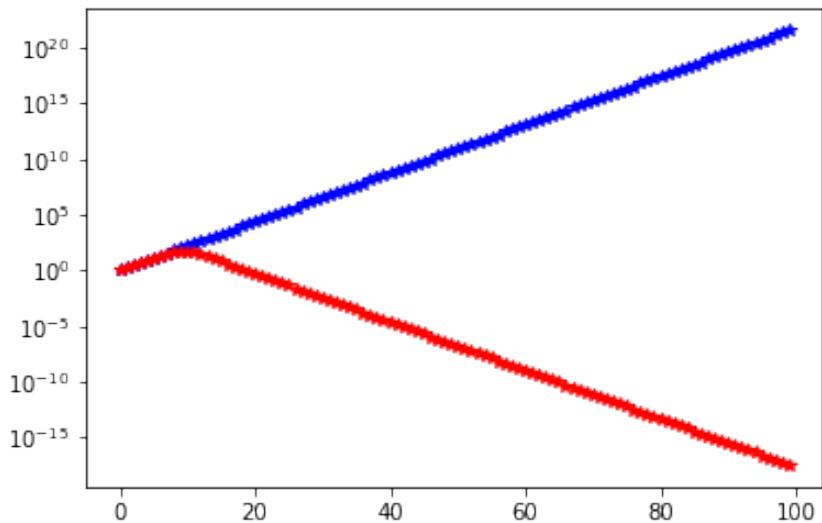
```
In [18]: # conditioning of random matrices:
n = A.shape[0]
n2A = np.linalg.norm(A)
print(n2A)
for i in range(-20,0):
    s = 10**i
    print("i=",i,"      K_A=",np.linalg.cond(A/n2A + s*np.random.randn(n,n)))
#endfor
```

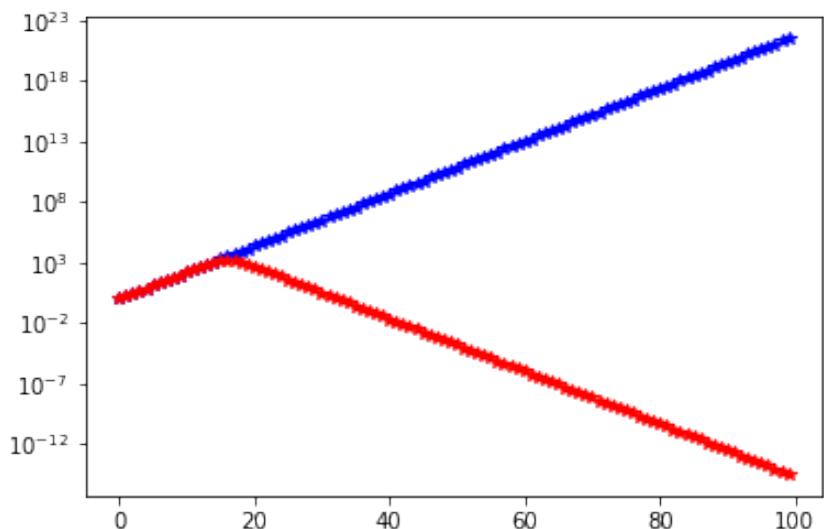
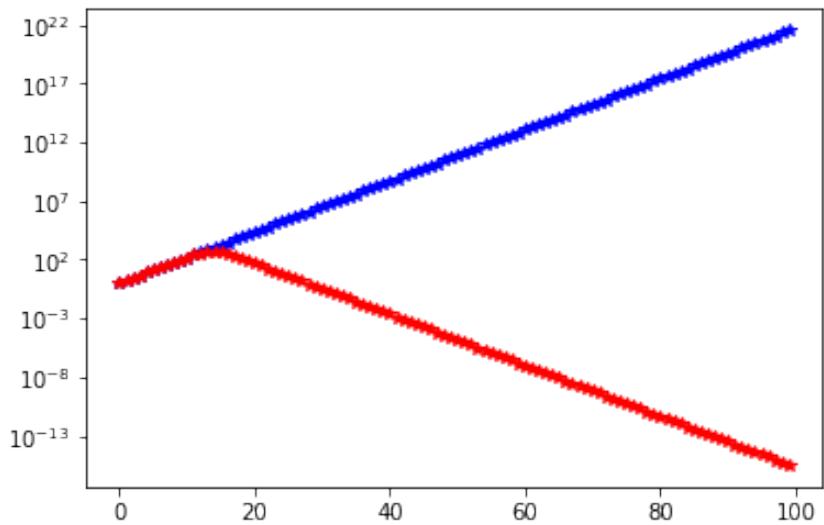
```
1.2577665549971198
i= -20      K_A= 3.2023624308356014e+18
i= -19      K_A= 4.0101593297880166e+18
i= -18      K_A= 1.6559854528194865e+18
i= -17      K_A= 8.357326000380321e+17
i= -16      K_A= 5.14494346773246e+16
i= -15      K_A= 2.7145478407540176e+16
i= -14      K_A= 3395652033177559.5
i= -13      K_A= 41245698201279.3
i= -12      K_A= 14816745466463.516
i= -11      K_A= 422842727284.24115
i= -10      K_A= 53504410109.83731
i= -9       K_A= 3157170003.1930695
i= -8       K_A= 605459127.1783403
i= -7       K_A= 107581683.01198865
i= -6       K_A= 44898024.667474665
i= -5       K_A= 606562.9993746404
i= -4       K_A= 46229.06545830429
i= -3       K_A= 5696.30546259251
i= -2       K_A= 15569.503119569074
i= -1       K_A= 116.28825015099825
```

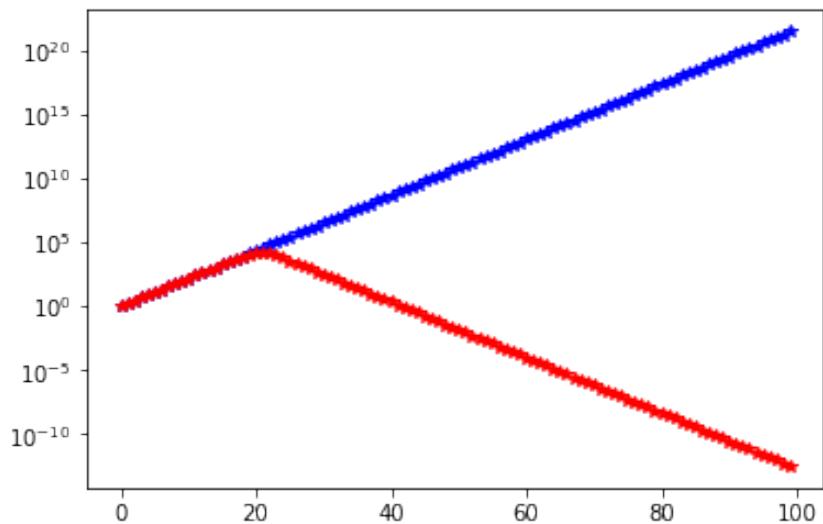
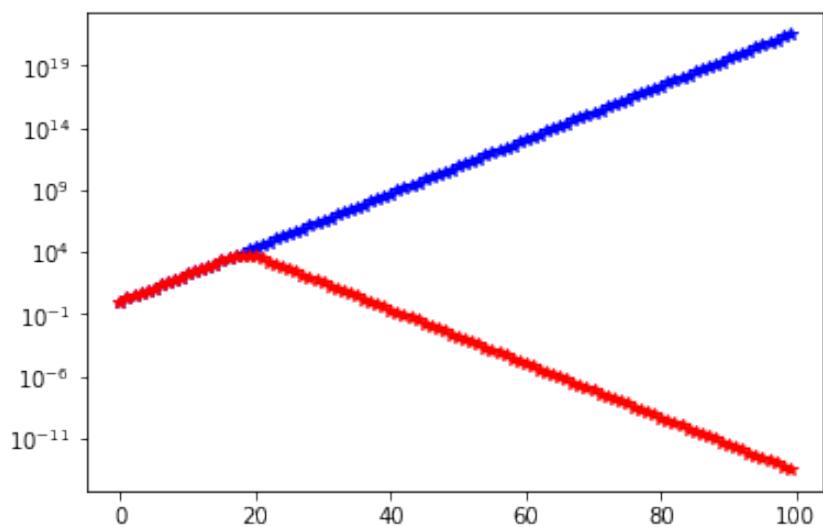
```
In [19]: # "Tichonov" effect on singular values:
n = 100
S = np.diag(np.exp(0.5*np.arange(0,-n,-1)))
for i in range(22):
    alpha = 10**(-i)
    plt.figure(300+i); plt.semilogy(1/np.diag(S), 'b*'); plt.semilogy(np.diag(S)/(n+alpha), 'r*')
#endfor
```

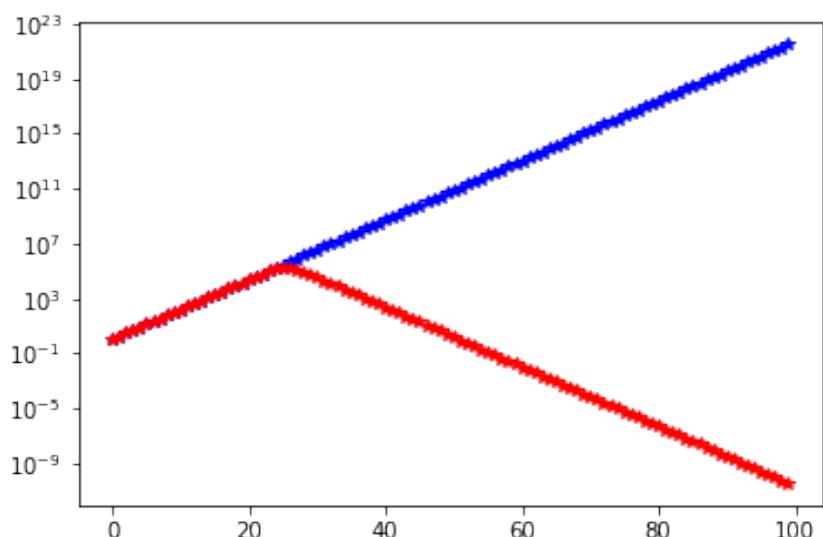
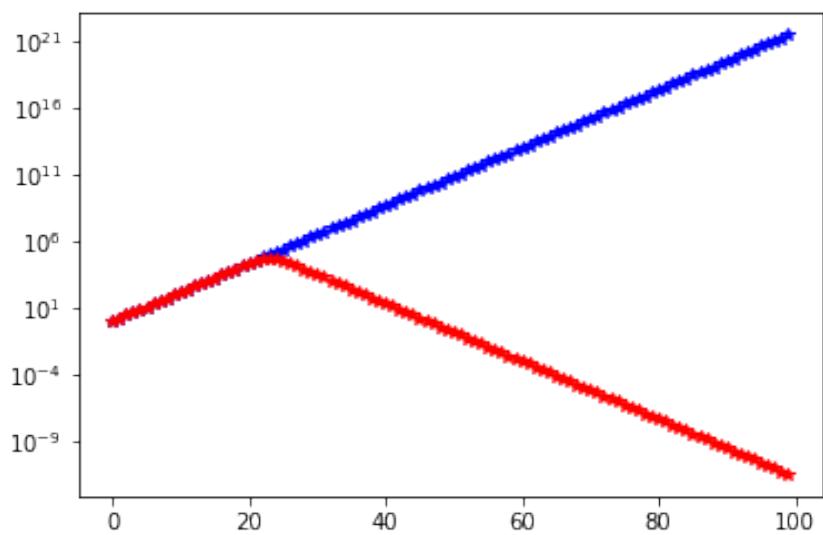


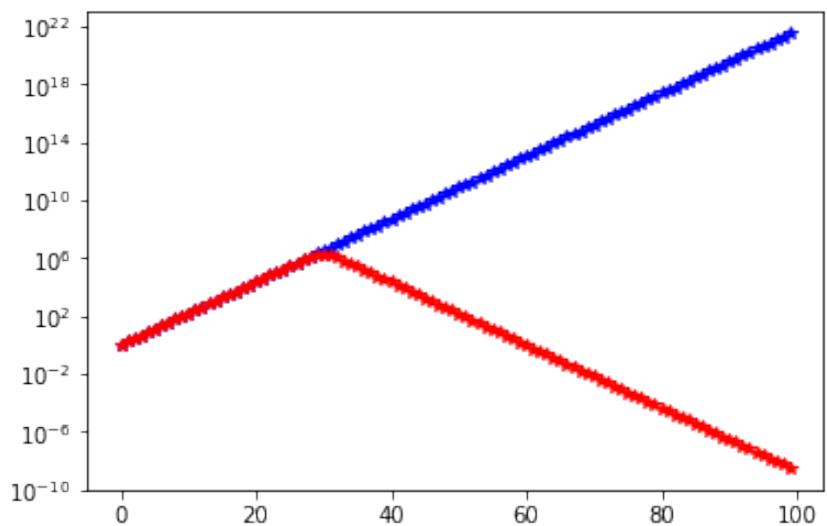
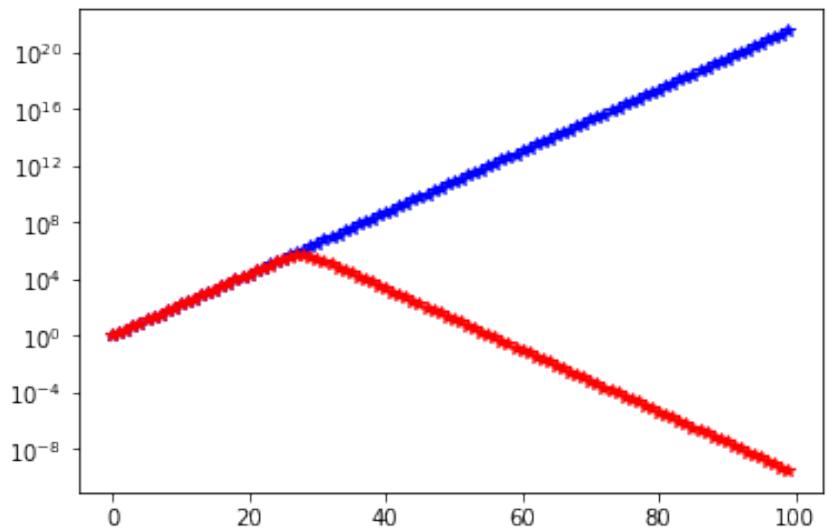


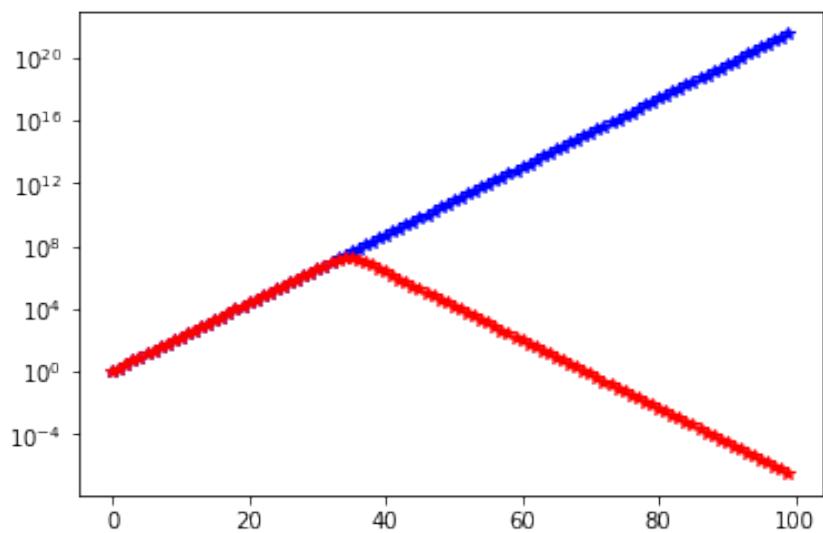
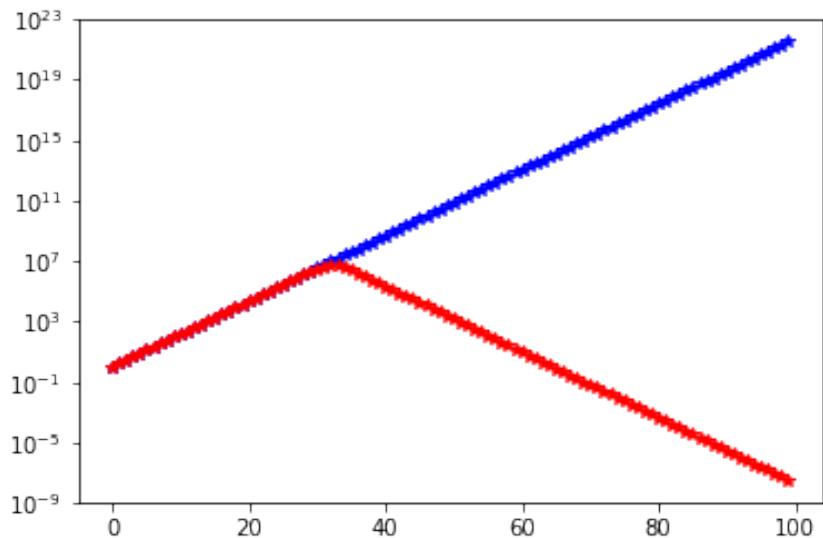


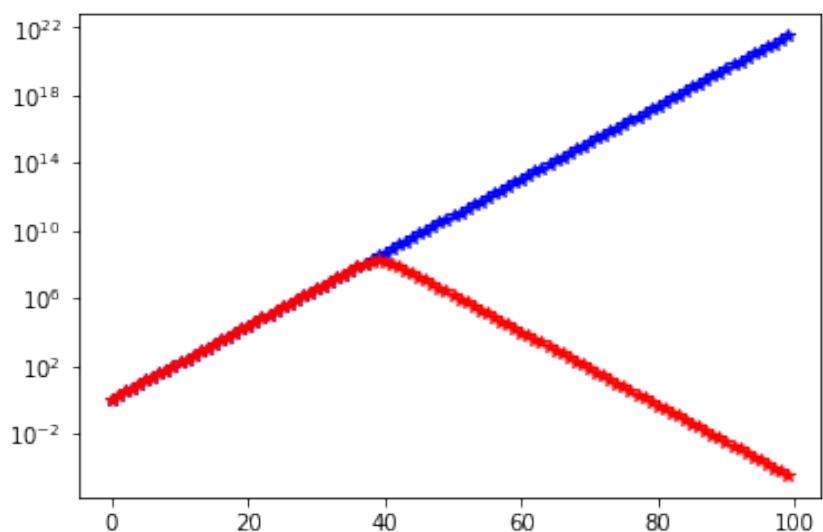
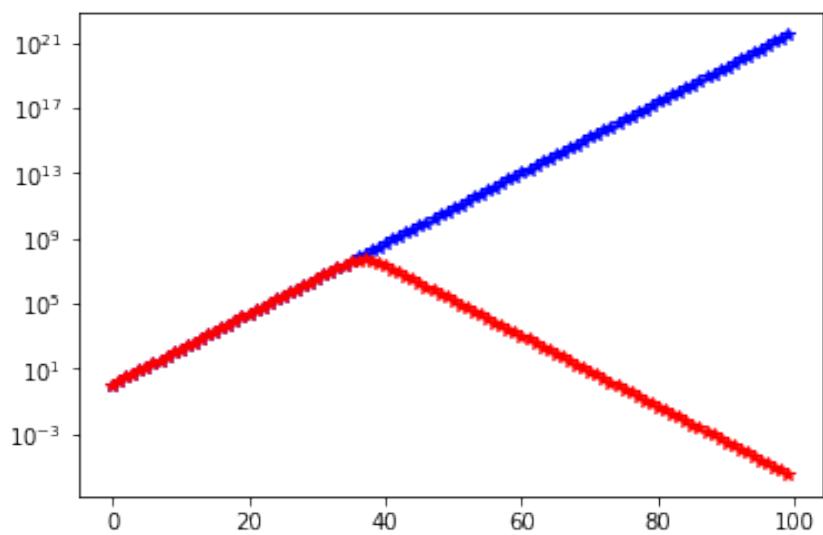


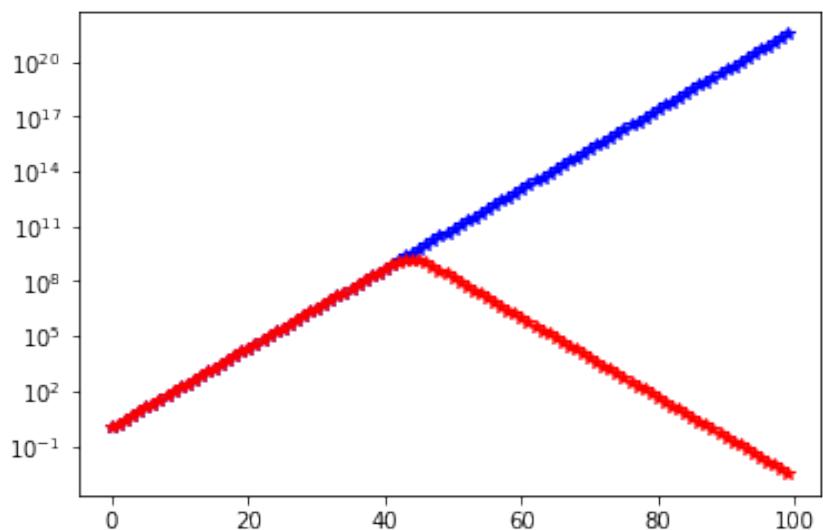
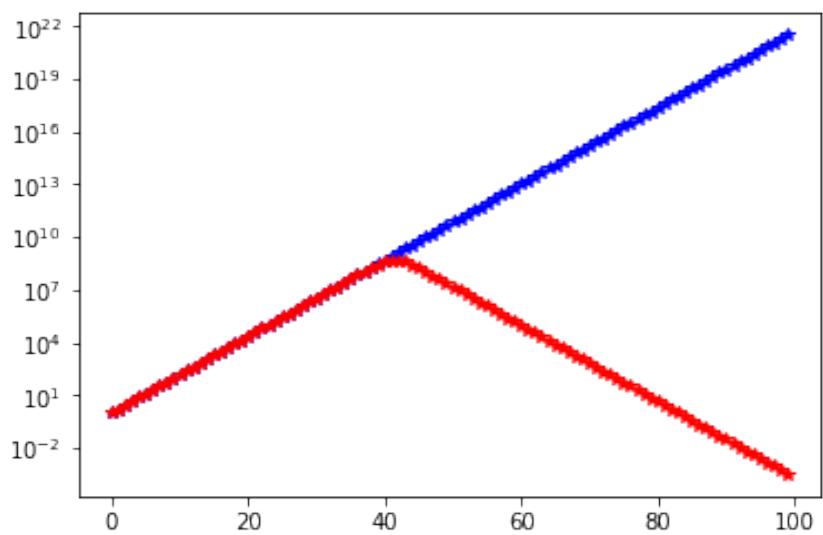


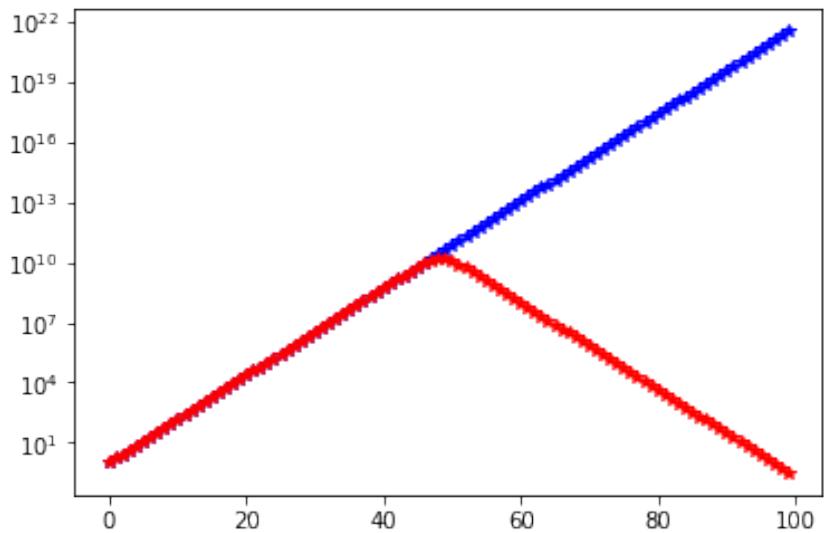
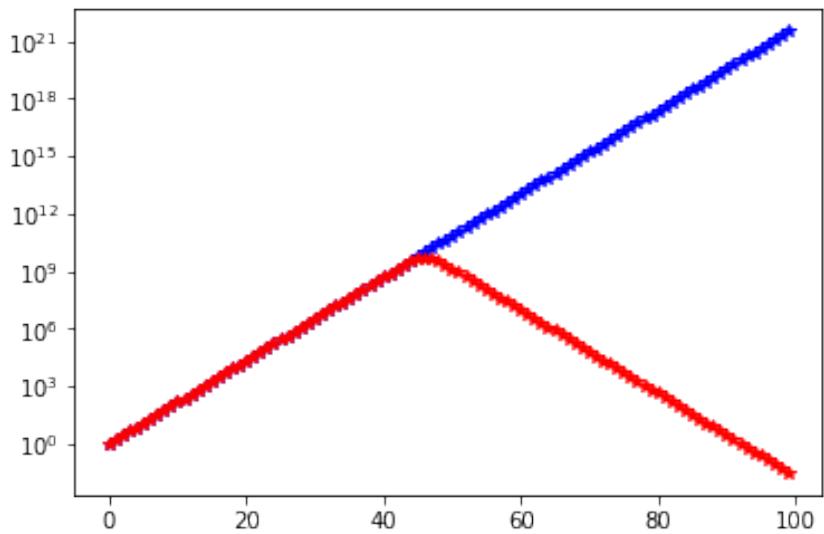












Torna al par. 4.3

In [20] :

Appendice F

Esempio troncamento modale

```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matplotlib inline"
import numpy as np
import matplotlib.pyplot as plt
from scipy.sparse import *
from scipy.sparse.linalg import *
```



```
In [2]: L=1;
h = 0.01
x = np.arange(0,L+h,h) # discretizzazione nello spazio
N = len(x); # numero di nodi ed N-1 e' il numero di elementi
dx = np.diff(x) # contiene la lunghezza dei segmenti della discretizzazione
# numero di modi presi in considerazione
nmodi = 5; # 10 -> 5 -> 2 -> 1
# posizione iniziale della forzante
load_position = round(N/3.); # indice del nodo su cui e' applicata
# discretizzazione dei coefficienti :
#
# "u": coefficiente di convezione
u_c = 0.0
u = u_c + np.asmatrix(np.zeros(N-1)).T
#
# "c": coefficiente di rigidezza
c_c = 5.0
c = c_c + np.asmatrix(np.zeros(N-1)).T
#
# "d": coefficiente di densita'
d_c = 1.0
d = d_c + np.asmatrix(np.zeros(N-1)).T
#
# "f": coefficiente del membro destro (la forzante)
f_c = 0.0
```

```

f = f_c + np.asmatrix(np.zeros(N)).T # "f" e' applicato ai nodi (e quindi varia in modo costante)
forzante = lambda t: 100*np.sin(2*np.pi*t*5) # essendo dipendente dal tempo, f=forzante(t)
#
ydn=0;
# condizioni iniziali
y = np.asmatrix(np.zeros(N*2)).T # [spostamento; velocità]
yndgN = np.asmatrix(np.zeros(N*2)).T
yndgN_m1 = np.zeros((N*2,1))
yndmN_m1 = np.asmatrix(np.zeros(nmodi*2)).T
yndgD = np.asmatrix(np.zeros((N-1)*2)).T
yndgD_m1 = np.asmatrix(np.zeros((N-1)*2)).T
#
Mg = lil_matrix((N,N)); Kg = lil_matrix((N,N)); Fg = np.asmatrix(np.zeros(N)).T
for i in range(0,N):
    ic=i; icm1=i-1;
    if i==0:
        Kg[i,i] = c[ic]/dx[i] - u[ic]/2.
        Kg[i,i+1] = -c[ic]/dx[i] + u[ic]/2.
    elif i<N-1:
        Kg[i,i-1] = -c[icm1]/dx[i-1] - u[icm1]/2.
        Kg[i,i] = c[icm1]/dx[i-1] + u[icm1]/2. + c[ic]/dx[i] - u[ic]/2.
        Kg[i,i+1] = -c[ic]/dx[i] + u[ic]/2.
    else:
        Kg[i,i-1] = -c[icm1]/dx[i-1] - u[icm1]/2.
        Kg[i,i] = c[icm1]/dx[i-1] + u[icm1]/2.
    #endif
    if i==0:
        Mg[i,i] = d[ic]*dx[i]/3.
        Mg[i,i+1] = d[ic]*dx[i]/6.
    elif i<N-1:
        Mg[i,i-1] = d[icm1]*dx[i-1]/6.
        Mg[i,i] = d[icm1]*dx[i-1]/3. + d[ic]*dx[i]/3.
        Mg[i,i+1] = d[ic]*dx[i]/6.
    else:
        Mg[i,i-1] = d[icm1]*dx[i-1]/6.
        Mg[i,i] = d[icm1]*dx[i-1]/3.
    #endif
    if i==0:
        Fg[i] = f[1]*dx[i]/3. + f[2]*dx[i]/6.
    elif i<N-1:
        Fg[i] = f[i-1]*dx[i-1]/6. + f[i]*(dx[i-1]+dx[i])/3. + f[i+1]*dx[i]/6.
    else:
        Fg[i] = f[i-1]*dx[i-1]/6. + f[i]*dx[i-1]/3.
    #endif
#endfor
iMg = np.linalg.inv(Mg.todense());
invMxK = iMg*Kg;
invMxF = iMg*Fg;
FDiriN = 0;

```

```

Bg = lil_matrix((N,N-1));
for i in range(0,N-1):
    Bg[i,i]=1.0;
#endfor

In [3]: D,V = np.linalg.eig(invMxK)
I = np.argsort(D)
D = D[I]
V = V[:,I]

In [4]: ### discretizzazione nel tempo
if d_c==0:
    T = dt; # cosi` finisce in una iterazione e si risole il problema stazionario
else:
    T = 0.8;
#endif
AsN = np.asmatrix(np.zeros((2*N,2*N)))
AsN[0:N,N:2*N] = np.eye(N)
AsN[N:2*N,0:N] = -invMxK
BsN = np.asmatrix( np.zeros((2*N,1)) )
BsN[N:2*N,0] = invMxF+FDiriN
#
V = np.asmatrix(V[:,0:nmodi])
Fm = V.T @ Fg
Mm = V.T @ Mg @ V
Km = V.T @ Kg @ V
iMm = np.linalg.inv(Mm)
invMmxKm = iMm @ Km
invMmxFm = iMm @ Fm
AmN = np.asmatrix(np.zeros((2*nmodi,2*nmodi)))
AmN[0:nmodi,nmodi:2*nmodi] = np.eye(nmodi)
AmN[nmodi:2*nmodi,0:nmodi] = -invMmxKm
BmN = np.asmatrix( np.zeros((2*nmodi,1)) )
BmN[nmodi:2*nmodi,0] = invMmxFm
# creo le matrici per Eulero implicito:
dt = 0.01
BN = np.eye(N*2) - dt*AsN
cN = dt*BsN
BmodaleN = np.eye(nmodi*2) - dt*AmN
cmN = dt*BmN

In [5]: it = 0
nframes = 5.
dtf = T/nframes;
serie_t = np.arange(dt,T+dt,dt);    len_serie_t=len(serie_t);
spost_ricev = np.zeros(len_serie_t)
load_exact_position = load_position;
for t in serie_t:

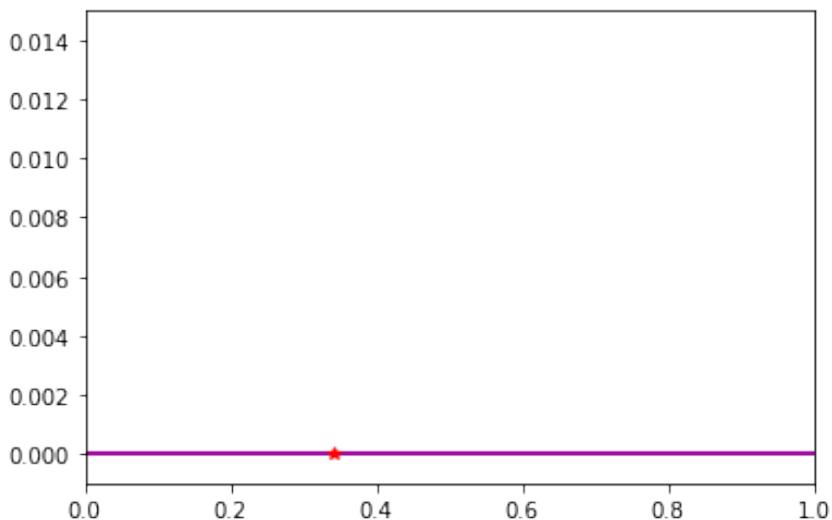
```

```

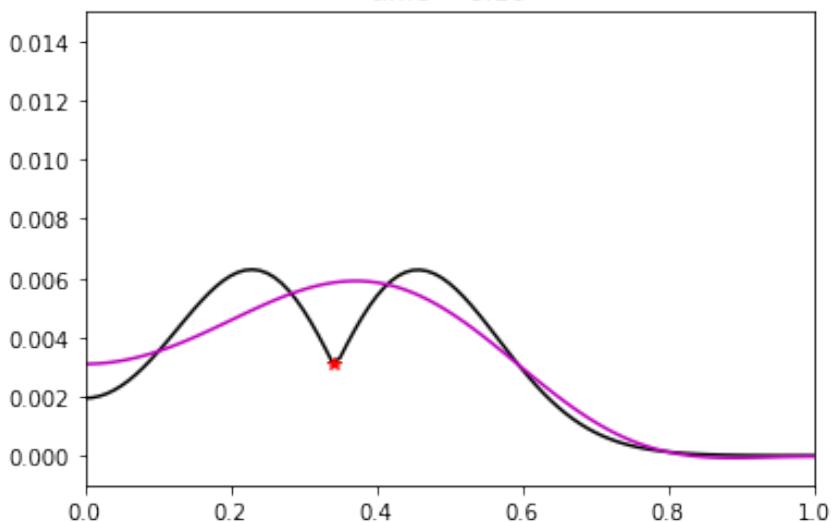
it = it+1
#print("cN = ", cN.T)
#print("yndgN_m1 = ", yndgN_m1.T)
#b = yndgN_m1 + cN
#print("b = ", b.T)
yndgN = np.linalg.solve(BN, yndgN_m1 + cN)
#print("yndgN = ", yndgN.T)
# NB: la matrice "BN" sarebbe sparsa: se avessimo usato per lei la rappresentazio
yndmN = np.linalg.solve(BmodaleN, yndmN_m1+cmN)
syN = yndgN[0:N]
smyN = yndmN[0:nmodi]
yndgN_m1 = yndgN.copy()
yndmN_m1 = yndmN.copy()
# visualizzazione dei risultati
if (t % dtf)<=dt:
    plt.figure(it);
    plt.plot(x,syN, 'k-')
    plt.plot(x,V@smyN, 'm-');
    plt.plot(load_position*dx[0],syN[load_position], 'r*');
    plt.axis([0., L, -0.001, 0.015])
    plt.title('time = ' + str(t))
    plt.show()
#endif
if t>0.:
    # evoluzione della forzante
    load_intensity = forzante(t)
    Fg = np.asmatrix(np.zeros(Fg.shape[0])).T
    Fg[load_position+1-1] = load_intensity*dx[0]/3.
    Fg[load_position+2-1] = load_intensity*dx[0]/6.
    invMxF = iMg@Fg
    Fm = V.T@Fg
    invMmxFm = iMm@Fm
#endif
BsN[N:2*N,0:N] = invMxF+FDiriN
cN = dt*BsN
BmN[nmodi:2*nmodi,0:nmodi] = invMmxFm
cmN = dt*BmN
#endiffor

```

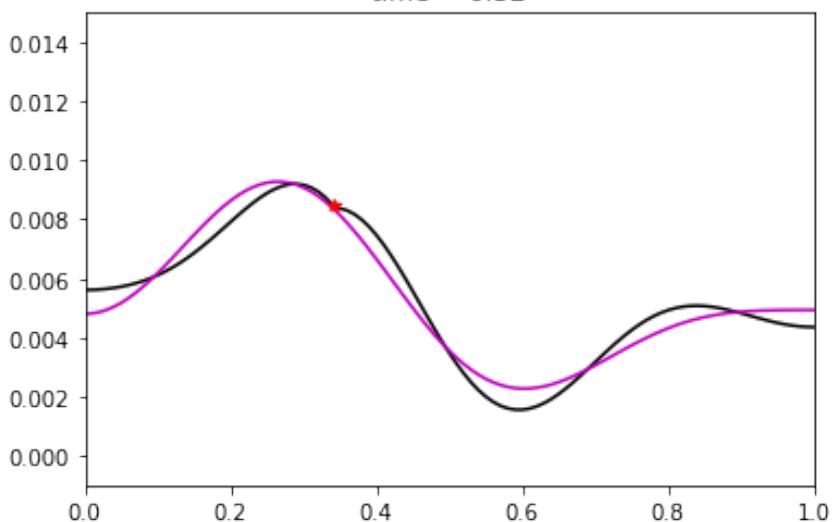
time = 0.01



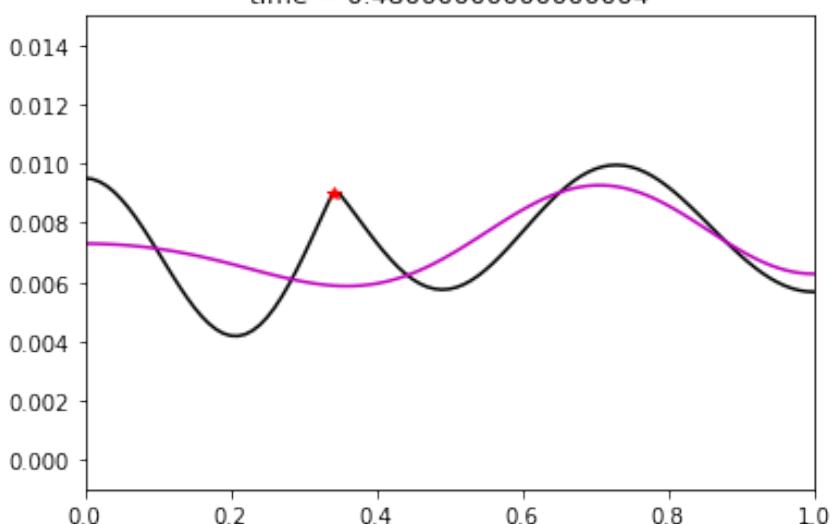
time = 0.16



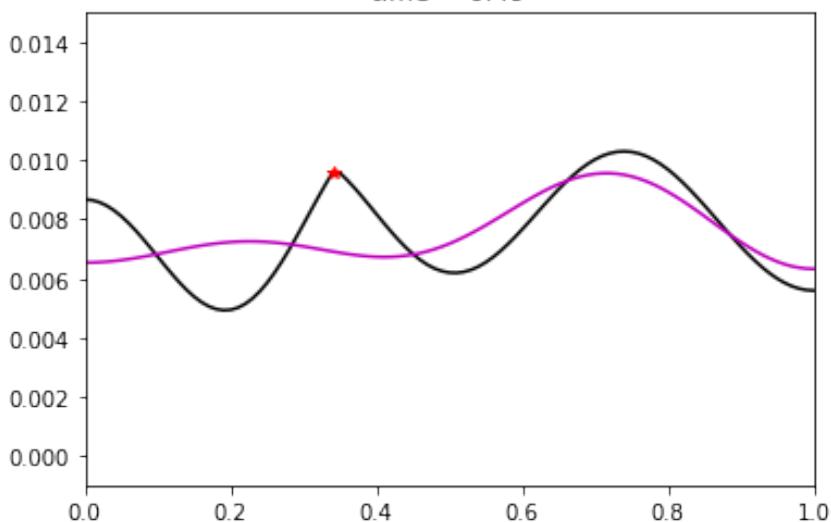
time = 0.32



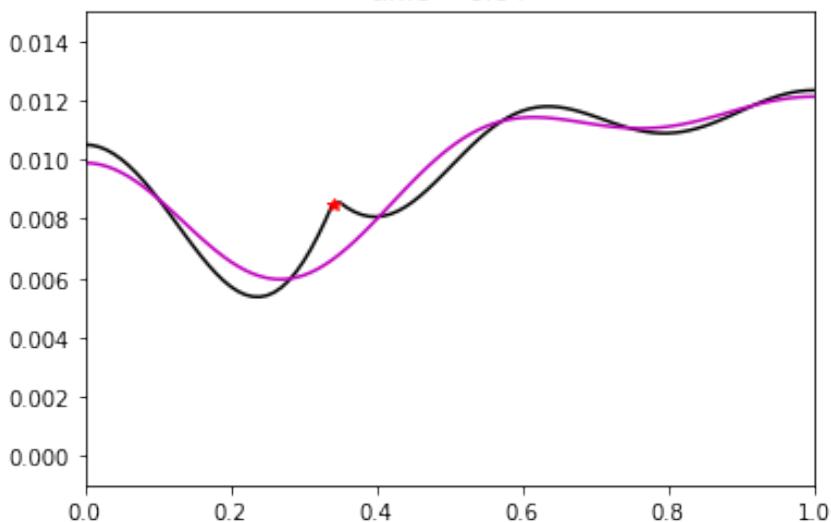
time = 0.48000000000000004

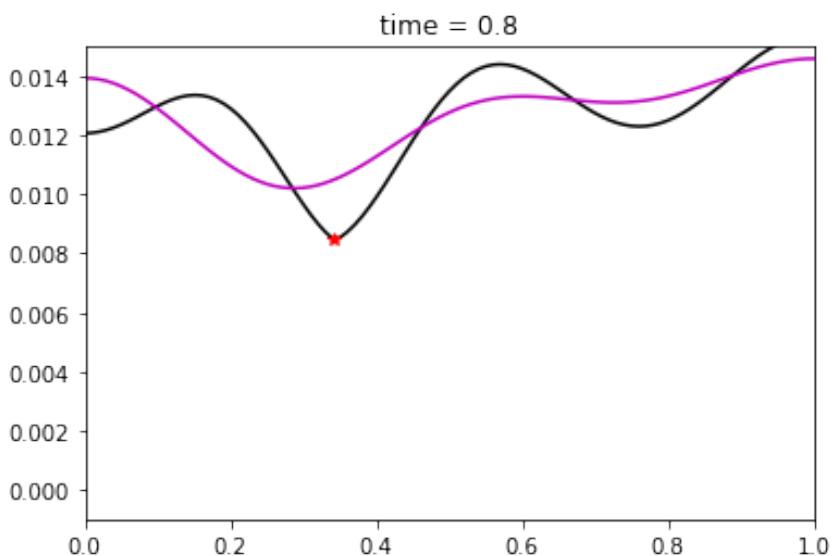


time = 0.49



time = 0.64





Torna al par. [2.4.3](#)

Appendice G

Esempio PCA

```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matplotlib inline"
import numpy as np
import matplotlib.pyplot as plt

In [2]: N = 100

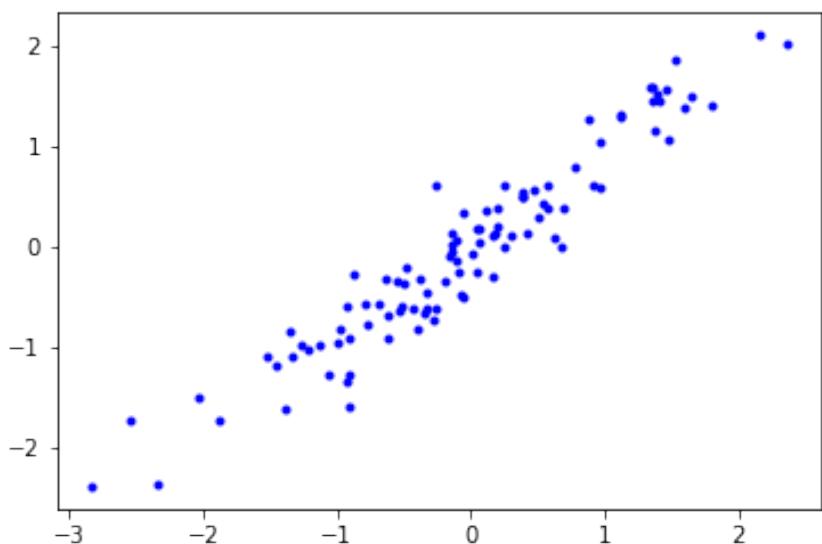
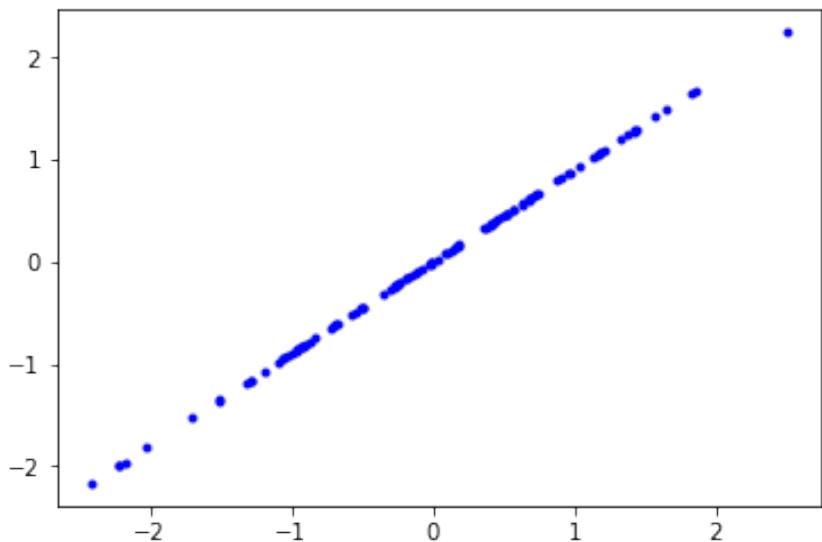
In [3]: c1 = np.random.randn(N)
c2 = c1 * 0.9
A1 = np.squeeze(np.array([[c1],[c2]])).T
print(A1.shape)

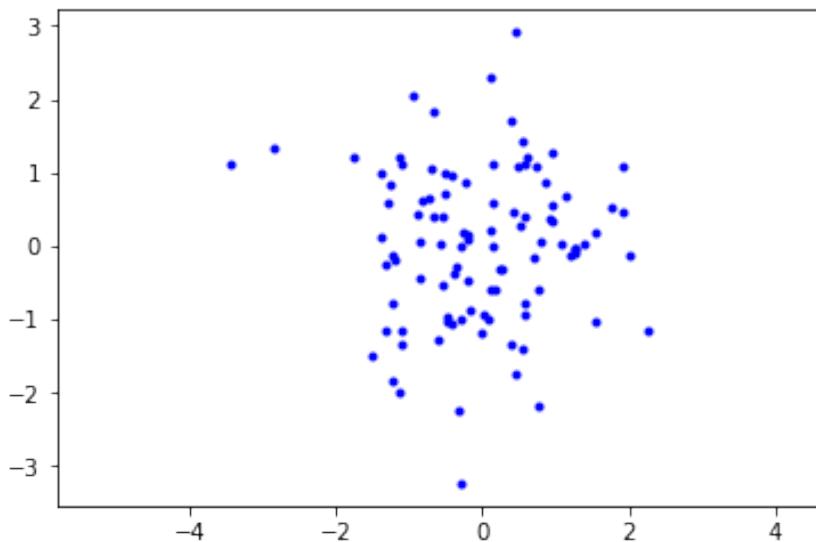
(100, 2)

In [4]: c1 = np.random.randn(N)
c2 = c1 * 0.9 + 0.3*np.random.randn(N)
A2 = np.squeeze(np.array([[c1],[c2]])).T

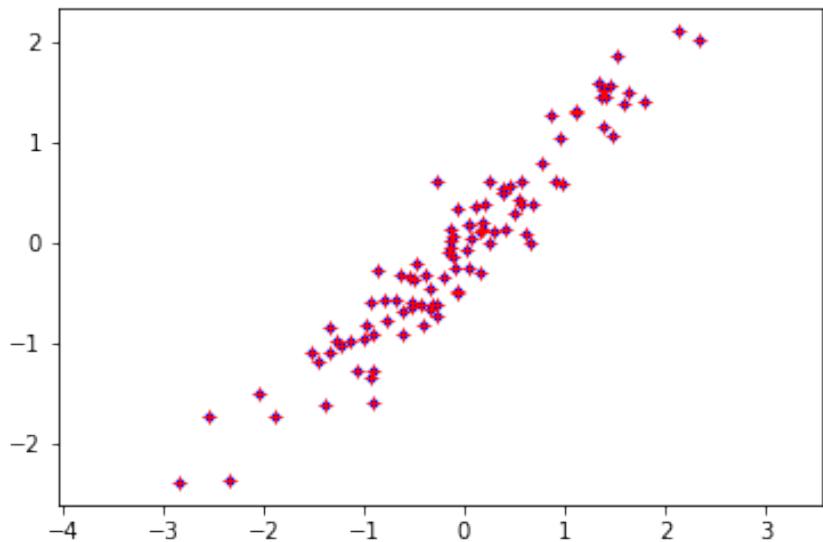
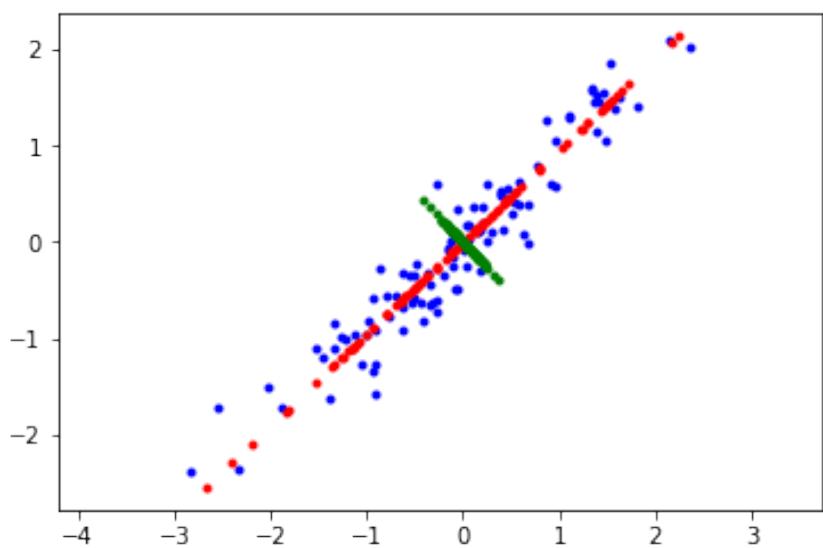
In [5]: c1 = np.random.randn(N)
c2 = np.random.randn(N)
A3 = np.squeeze(np.array([[c1],[c2]])).T

In [6]: plt.figure(1), plt.plot(A1[:,0],A1[:,1], 'b.'); plt.show()
plt.figure(2), plt.plot(A2[:,0],A2[:,1], 'b.'); plt.show()
plt.figure(3), plt.plot(A3[:,0],A3[:,1], 'b.'); plt.axis('equal'); plt.show()
```





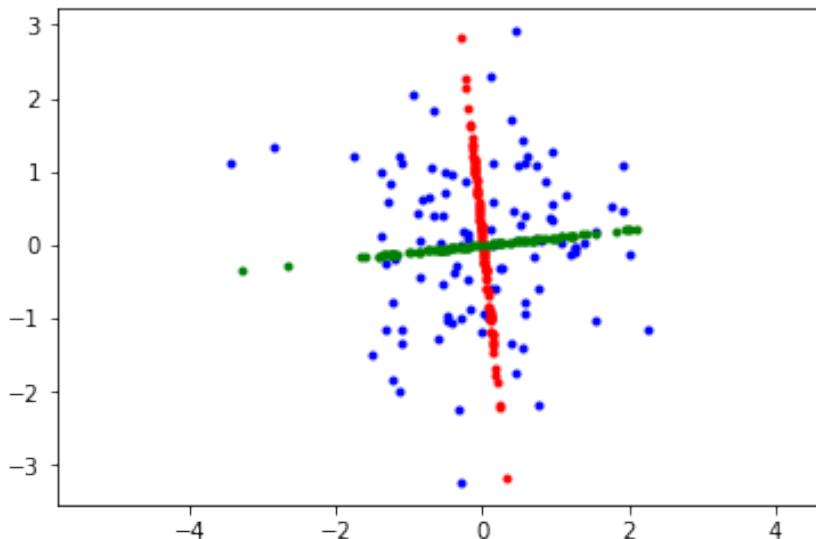
```
In [7]: C = A2.T @ A2
D,V = np.linalg.eig(C)
I = np.argsort(D)
D = D[I]
V = V[:,I]
#figure(1), axis('equal'), plot(A2[:,0],A2[:,1],'b.'); plot(A2[:,0],A2*V[:,1],'r.')
U,S,V = np.linalg.svd(A2)
V = V.T # NB: per riportarsi a quella solita, basta scommentare questa istruzione
PC1 = U[:,range(0,1)]@np.diag([S[0]])@V[:,range(0,1)].T
PC2 = U[:,range(1,2)]@np.diag([S[1]])@V[:,range(1,2)].T
#figure(2), axis('equal'), plot(A2[:,0],A2[:,1],'b.'); plot(A2[:,0],-A2*V[:,0],'r.')
# le componenti principali "assomigliano" agli andamenti dominanti presenti nei dati
plt.figure(4), plt.axis('equal'), plt.plot(A2[:,0],A2[:,1],'b.); plt.plot(PC1[:,0]
PC12 = PC1 + PC2
plt.figure(5), plt.axis('equal'), plt.plot(A2[:,0],A2[:,1],'b.); plt.plot(PC12[:,0]
```



```
In [8]: print(S)
```

```
[13.8137274  2.07612263]
```

```
In [9]: U,S,V = np.linalg.svd(A3)
V = V.T # NB: per riportarsi a quella solita, basta scommentare questa istruzione
PC1 = U[:,range(0,1)]@np.diag([S[0]])@V[:,range(0,1)].T
PC2 = U[:,range(1,2)]@np.diag([S[1]])@V[:,range(1,2)].T
plt.figure(6), plt.axis('equal'), plt.plot(A3[:,0],A3[:,1],'b.'); plt.plot(PC1[:,0]
```



```
In [10]: print(S)
```

```
[10.55041014 10.15396595]
```

Torna al par. 2.1

G.1 PCA con matrici di correlazione e di covarianza

```
In [11]: N = 100
```

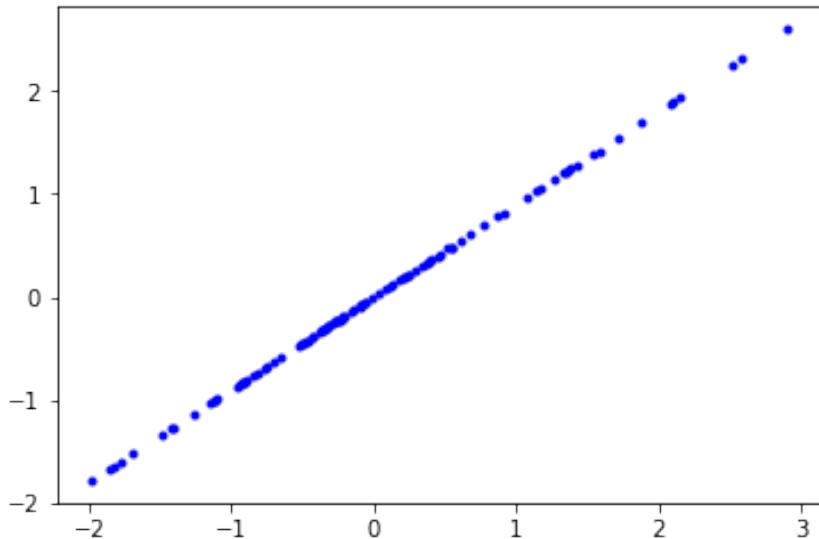
```
In [12]: c1 = np.random.randn(N)
c2 = c1 * 0.9
A1 = np.squeeze(np.array([[c1],[c2]])).T
print(A1.shape)
```

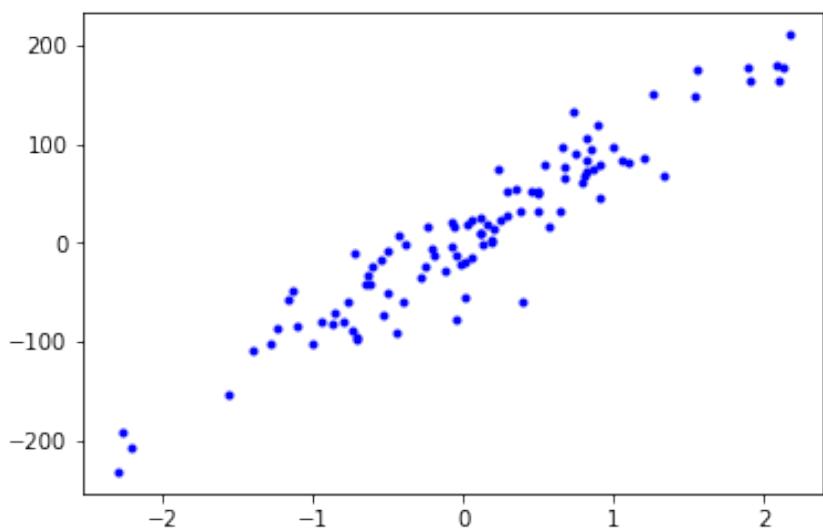
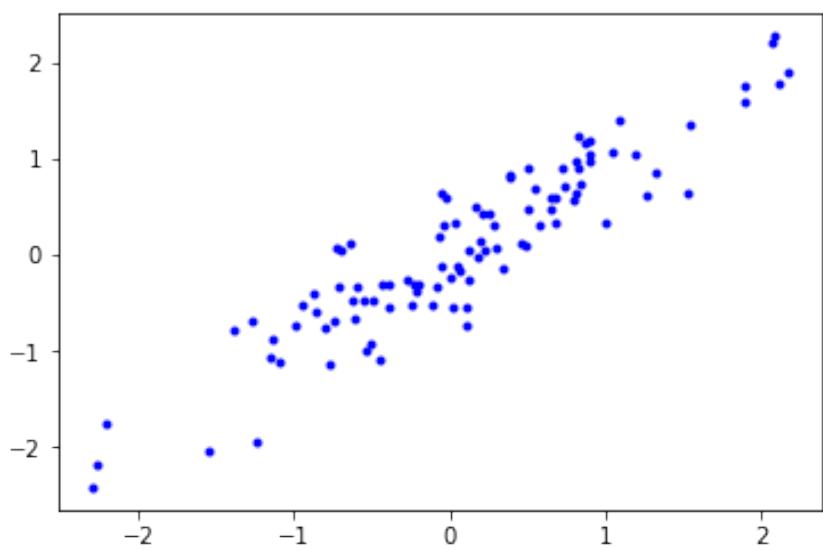
```
(100, 2)
```

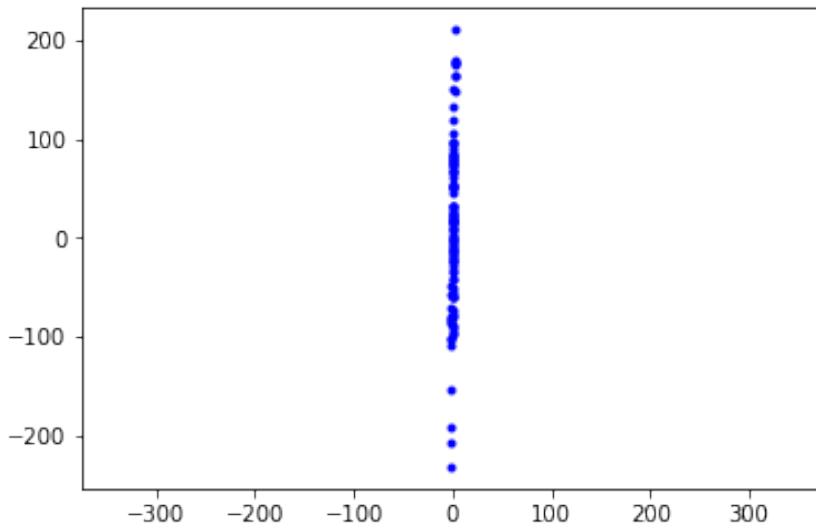
```
In [13]: c1 = np.random.randn(N)
c2 = c1 * 0.9 + 0.3*np.random.randn(N)
A2 = np.squeeze(np.array([[c1],[c2]])).T
```

```
In [14]: # now, let us change the unity measurement for x2 in data matrix A2,
# e.g. considering as it would be a length, from "m" to "cm":
c2u = (c1 * 0.9 + 0.3*np.random.randn(N))*100.0
A2u = np.squeeze(np.array([[c1],[c2u]])).T
```

```
In [15]: plt.figure(1), plt.plot(A1[:,0],A1[:,1],'b.'); plt.show()
plt.figure(2), plt.plot(A2[:,0],A2[:,1],'b.'); plt.show()
plt.figure(3), plt.plot(A2u[:,0],A2u[:,1],'b.'); plt.show()
plt.figure(4), plt.plot(A2u[:,0],A2u[:,1],'b.'); plt.axis('equal'); plt.show()
```







```
In [16]: print("PCA with covariance matrices:")
    # compute the principal components of A2 data matrix:
C = np.cov(A2,rowvar=0)    # covariance matrix
D,V = np.linalg.eig(C)
I = np.argsort(D)
I = I[-1::-1] # reverse indices: I[0] indexes the maximum eigenvalue
#print I
D = D[I]
#print "D = ",D
V = V[:,I]
# verify that eigenvectors have unit-norm:
if False:
    print("2-norms of eigenvectors:")
    print(np.linalg.norm(V[:,0]))
    print(np.linalg.norm(V[:,1]))
#endif
print("first principal component of A2 is: ",V[0,0],"*x1 + ",V[1,0],"*x2")
# now, compute the principal components of A2u, with changed x2 unity measurement:
Cu = np.cov(A2u,rowvar=0)    # covariance matrix
D,V = np.linalg.eig(Cu)
I = np.argsort(D) # sort in ascending order
I = I[-1::-1] # reverse indices: I[0] indexes the maximum eigenvalue
#print I
D = D[I]
#print "D = ",D
V = V[:,I]
```

```

# verify that eigenvectors have unit-norm:
if False:
    print("2-norms of eigenvectors:")
    print(np.linalg.norm(V[:,0]))
    print(np.linalg.norm(V[:,1]))
#endif
print("first principal component of A2u is: ",V[0,0],"*x1 + ",V[1,0],"*x2")
print("")
print("PCA with correlation matrices:")
# compute the principal components of A2 data matrix:
R = np.corrcoef(A2,rowvar=0) # correlation matrix
D,V = np.linalg.eig(R)
I = np.argsort(D)
I = I[-1::-1] # reverse indices: I[0] indexes the maximum eigenvalue
#print I
D = D[I]
#print "D = ",D
V = V[:,I]
# verify that eigenvectors have unit-norm:
if False:
    print("2-norms of eigenvectors:")
    print(np.linalg.norm(V[:,0]))
    print(np.linalg.norm(V[:,1]))
#endif
print("first principal component of A2 is: ",V[0,0],"*x1 + ",V[1,0],"*x2")
# now, compute the principal components of A2u, with changed x2 unity measurement:
Ru = np.corrcoef(A2u,rowvar=0) # correlation matrix
D,V = np.linalg.eig(Ru)
I = np.argsort(D) # sort in ascending order
I = I[-1::-1] # reverse indices: I[0] indexes the maximum eigenvalue
#print I
D = D[I]
#print "D = ",D
V = V[:,I]
# verify that eigenvectors have unit-norm:
if False:
    print("2-norms of eigenvectors:")
    print(np.linalg.norm(V[:,0]))
    print(np.linalg.norm(V[:,1]))
#endif
print("first principal component of A2u is: ",V[0,0],"*x1 + ",V[1,0],"*x2")

```

PCA with covariance matrices:

first principal component of A2 is: 0.7148652990449166 *x1 + 0.6992621856081035
 first principal component of A2u is: -0.010219168892975078 *x1 + -0.999947782930

PCA with correlation matrices:

first principal component of A2 is: -0.7071067811865475 *x1 + -0.707106781186547

first principal component of A2u is: -0.7071067811865475 *x1 + -0.7071067811865475 *x2

Torna al par. [2.1.1](#)

Appendice H

Minimi quadrati nonlineari

H.1 Esempio sulla convergenza globale

In questo esempio vediamo un problema di stima di parametri mediante minimi quadrati nonlineari in un semplice modello, costituito dalla funzione (statica) “y”, che dipende linearmente dalla variabile indipendente “x” ma nonlinearmente dai parametri “theta” (vettore di due componenti, e quindi il modello ha due parametri). Notare come la convergenza dei parametri vada a buon fine, osservando l'avvicinarsi delle predizioni del modello (curva rossa dei grafici) ai valori obiettivo “y_meas” (curva blu dei grafici). Notare però come il problema non sia affatto convesso: la mappa di colori mostra come la funzione costo “F_N” ha un gran numero di minimi locali. Notare anche quale è stato l'andamento delle stime, i cui valori numerici sono riportati in tabella e successivamente indicati nel grafico sotto la mappa di colori. Possiamo dire che:

- 1) non è stato raggiunto il minimo globale: l'algoritmo si è fermato in un minimo locale;
- 2) la convergenza delle stime non è stata monotona per quanto riguarda l'errore di stima dei parametri;
- 3) l'algoritmo è uscito inizialmente da un minimo locale grazie solo al settaggio (euristico) delle costanti nell'algoritmo stesso.

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.nonlinear_LS import *
```

```

In [2]: # il modello vero:
    xmax = 1.0
    dx = 0.1
    x = np.arange(0.0,xmax+dx,dx)
    N = len(x)
    theta = np.array([-2.0, sqrt(2.)])
    n_theta = len(theta)

    y = lambda x,theta: theta[0]*np.sin(theta[0])*x + theta[1]*np.cos(3.0*theta[1])*(1.0 - x*x)

    def y_param(theta):
        global x
        return y(x,theta)

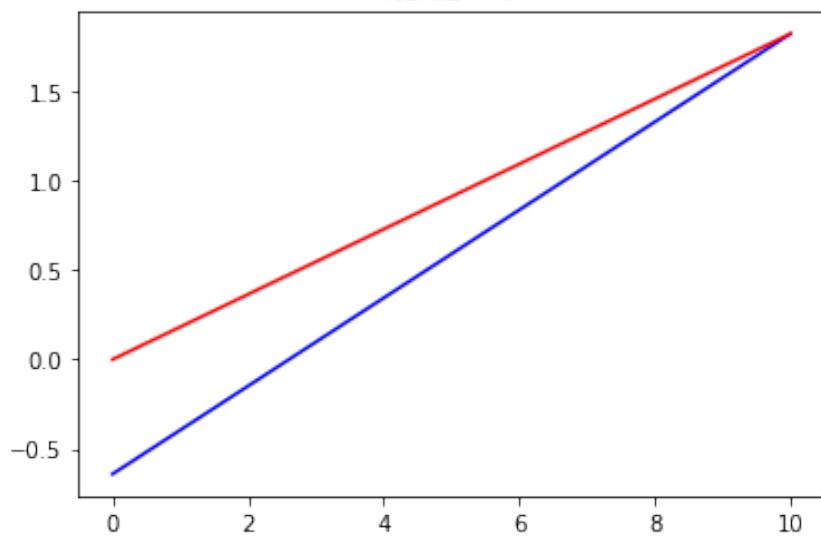
In [3]: std_mod = 0.0 # deviazione standard dell'errore di modello
        std_mis = 0.0 # deviazione standard dell'errore di misuray_meas = y(x,theta) + std_mod*x
        y_meas = y(x+std_mod*np.random.randn(N),theta) + std_mis*np.random.randn(N)
        test = 1 # 1,2
        if test==1:
            theta_est = np.array([-2.0, 0.0]) #np.random.randn(n_theta)
        elif test==2:
            theta_est = np.array([-2.0, -2.0]) #np.random.randn(n_theta)
        #endif
        E = GN_LM(y_meas,y_param,theta_est)

*****
metodo_stima = GN

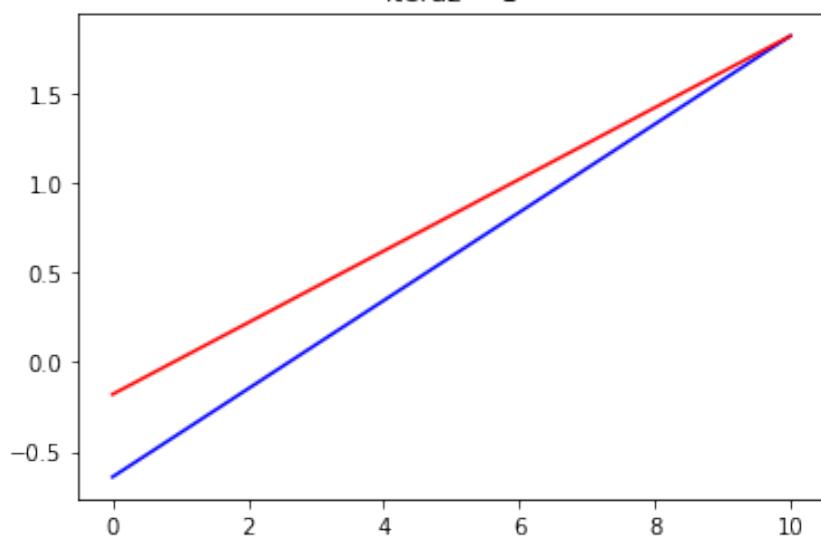
opzione_solve = LS
-----
start: theta_est = [-2. 0.] FN = 0.14343192993315315
mu = 1.0 temp_FN = 0.29963523830907923
mu = 0.5 temp_FN = 0.07424845760110874
iterazione 1 : theta_est = [-2.          -0.33504453] FN = 0.07424845760110874
mu = 1.0 temp_FN = 0.06750727757687498
iterazione 2 : theta_est = [-2.          1.08713139] FN = 0.06750727757687498
mu = 1.0 temp_FN = 0.23857099215945063
mu = 0.5 temp_FN = 0.029649195296150663
iterazione 3 : theta_est = [-2.          0.69814962] FN = 0.029649195296150663
mu = 1.0 temp_FN = 5.149279201391117e-05
iterazione 4 : theta_est = [-2.          0.82668299] FN = 5.149279201391117e-05
mu = 1.0 temp_FN = 4.110309232973441e-08
iterazione 5 : theta_est = [-2.          0.82130323] FN = 4.110309232973441e-08

```

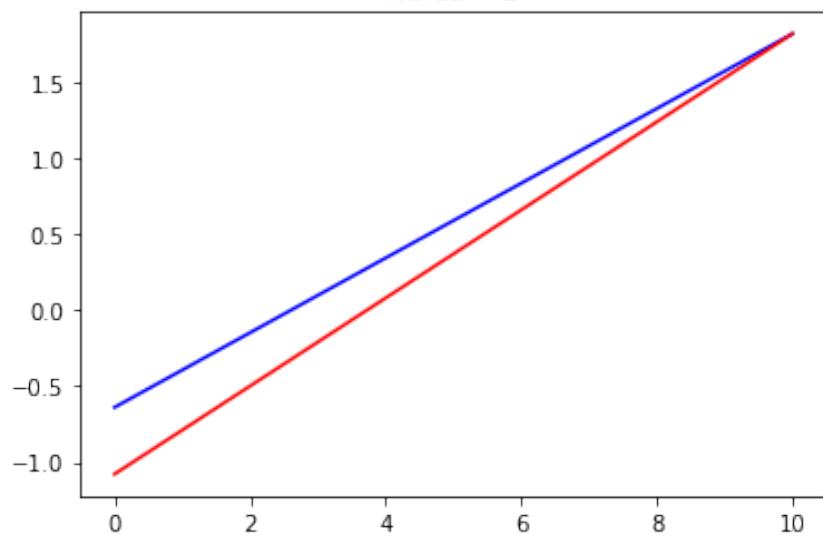
iteraz = 0



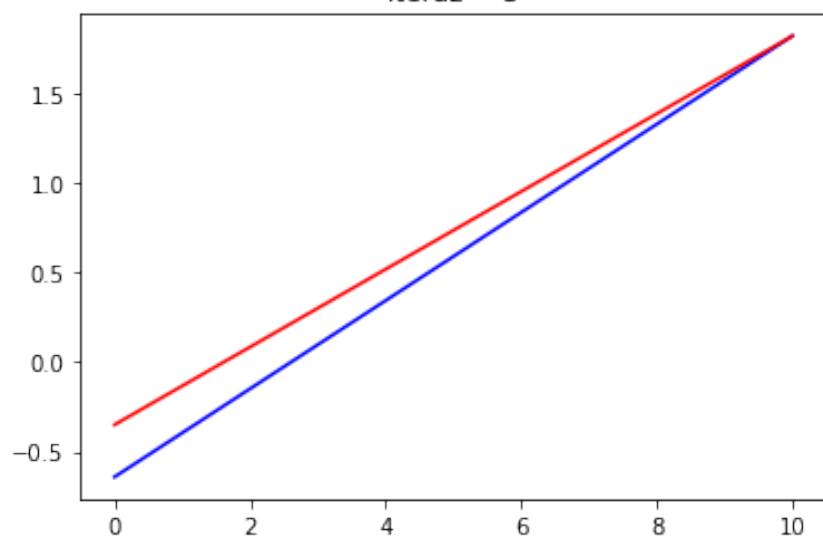
iteraz = 1



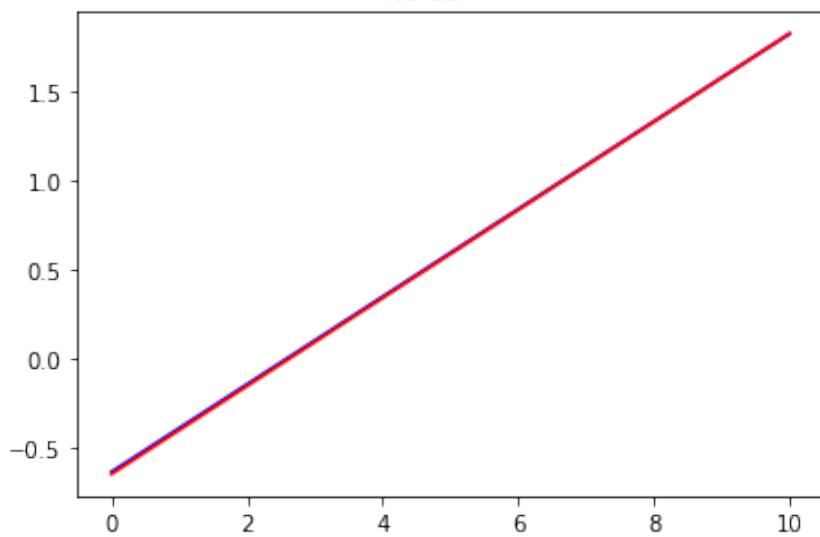
iteraz = 2



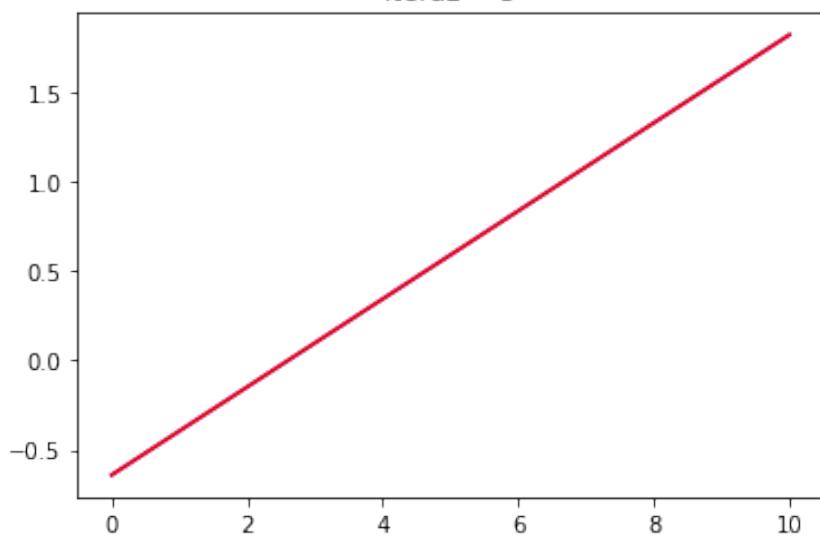
iteraz = 3



iteraz = 4



iteraz = 5



In [4]: theta_lims = (-6.0,6.0,-4.0,4.0)

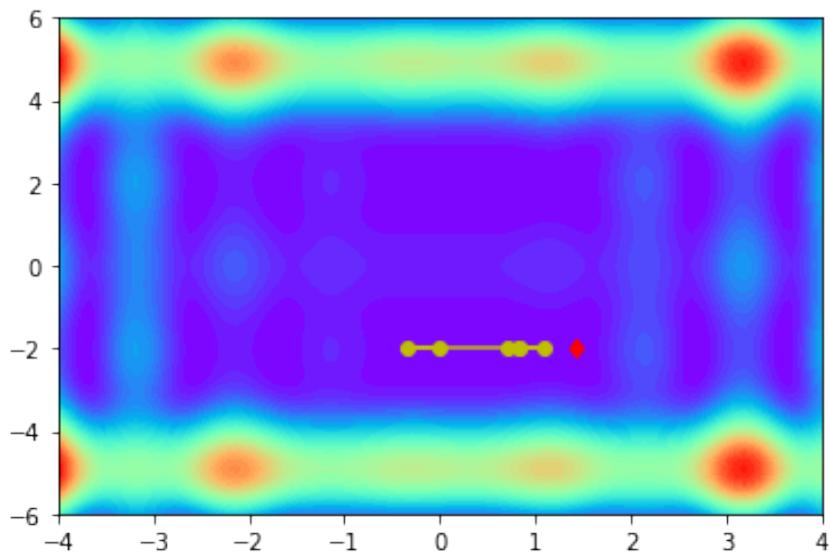
N1 = 100

```

N2 = 100
print(E)
plot_FN(y_meas,y_param,theta_lims,N1,N2,E,theta)
#
theta_lims = (-2.0,-2.0,-4.0,4.0)
plot_FN(y_meas,y_param,theta_lims,N1,N2,E,theta)

[[[-2.          ,  0.          ],
 [-2.          , -0.33504453],
 [-2.          ,  1.08713139],
 [-2.          ,  0.69814962],
 [-2.          ,  0.82668299],
 [-2.          ,  0.82130323]]]
x2.shape = (100,)
x1.shape = (100,)
V.shape = (100, 100)

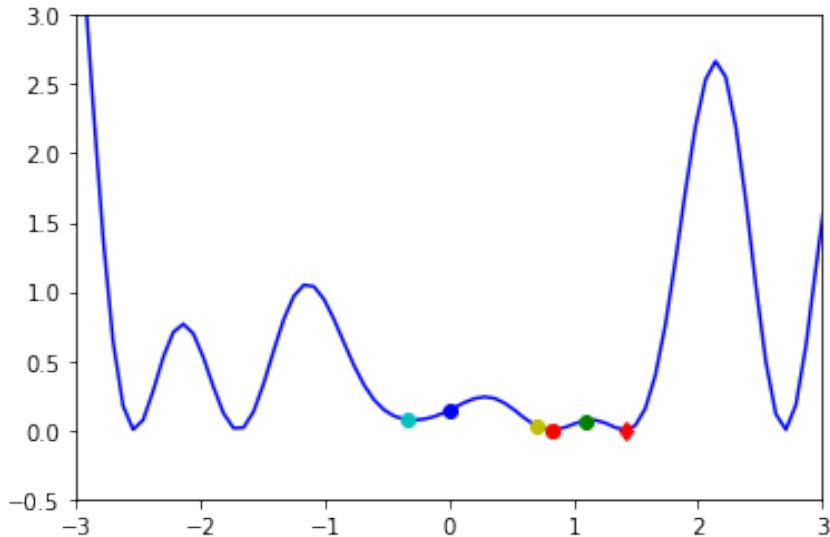
```



```

x2.shape = (100,)
V.shape = (100, 100)

```



In [5]: # NB: si e' fermato in un minimo locale! non e' arrivato al minimo globale.

Torna al par. 5

H.2 Esempio non unicità minimi quadrati nonlineari

In questo esempio consideriamo un modello "y" leggermente più complicato: è una funzione quadratica di "x". La presenza di un parametro elevato al quadrato crea un problema di identificabilità dei parametri: non vi è più una corrispondenza biunivoca tra valori in uscita "y" e valori di theta, perché "theta[1]" e "-theta[1]" danno la stessa "y". A questo punto, quali dei due valori verrà stimato dipende dalla stima iniziale.

```
In [6]: # il modello vero:
xmax = 1.0
dx = 0.1
x = np.array(np.arange(0.0,xmax+dx,dx))
N = len(x)
theta = np.array([-2.0, np.sqrt(2.)])
n_theta = len(theta)
```

```

y = lambda x,theta: theta[0]*x + (theta[1]*x)**2

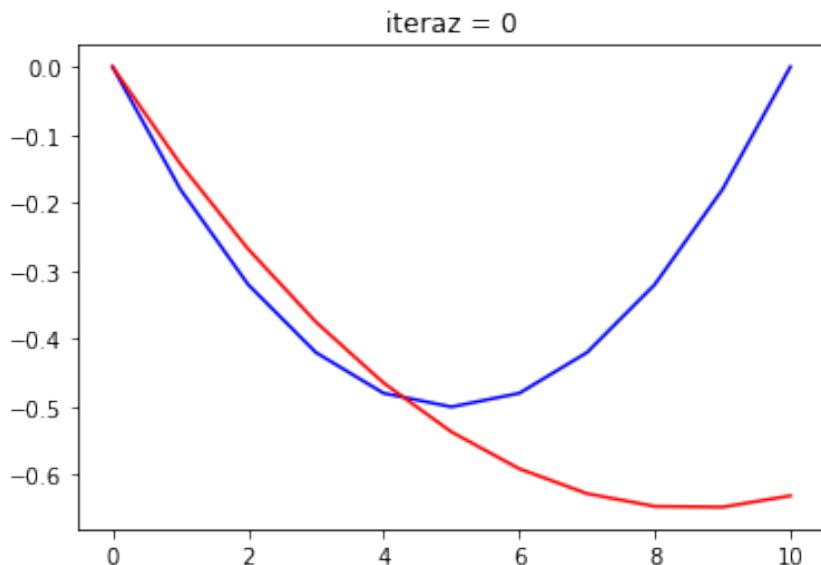
# NB: questo problema di stima ha due soluzioni ( e non una sola!):
#      [-2., -1.41421411] e [-2., 1.41421411]
def y_param(theta):
    global x
    return y(x,theta)

In [7]: std_mod = 0.0 # deviazione standard dell'errore di modello
        std_mis = 0.0 # deviazione standard dell'errore di misura
        y_meas = y(x+std_mod*np.random.randn(N),theta) + std_mis*np.random.randn(N)
        theta_est = np.random.randn(n_theta) #np.array([1.0, -3.0])
        E = GN_LM(y_meas,y_param,theta_est)

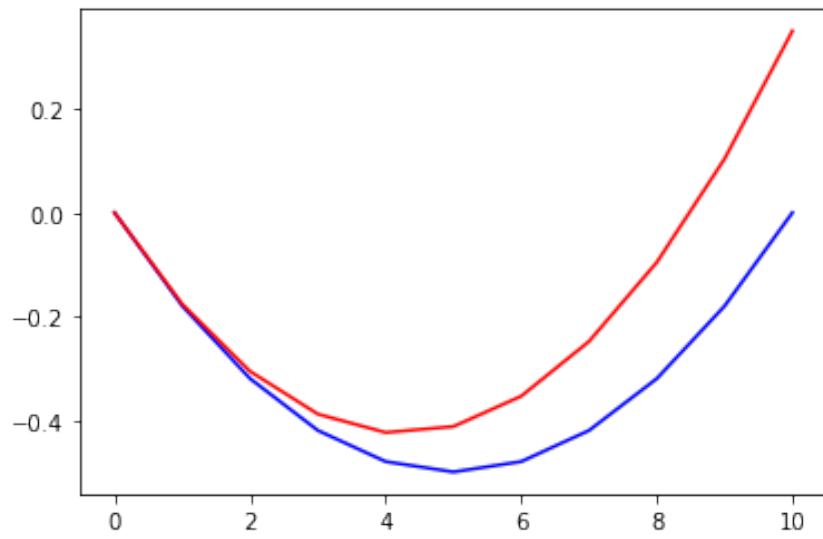
*****
metodo_stima = GN

opzione_solve = LS
-----
start: theta_est = [-1.51643544 -0.94113084] FN = 0.07143093824478404
mu = 1.0 temp_FN = 0.02828394377437294
iterazione 1 : theta_est = [-2. -1.53311694] FN = 0.02828394377437294
mu = 1.0 temp_FN = 3.929749663168559e-05
iterazione 2 : theta_est = [-2. -1.41882443] FN = 3.929749663168559e-05
mu = 1.0 temp_FN = 1.0341920946520708e-10
iterazione 3 : theta_est = [-2. -1.41422105] FN = 1.0341920946520708e-10

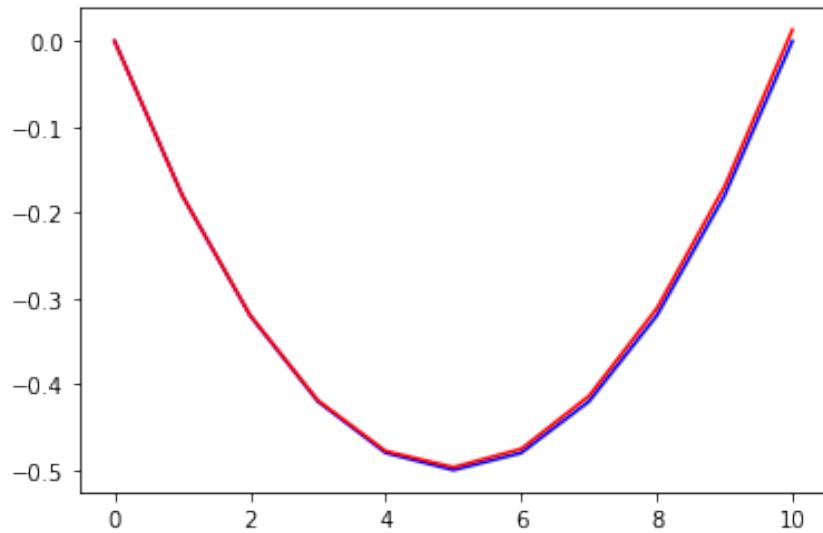
```



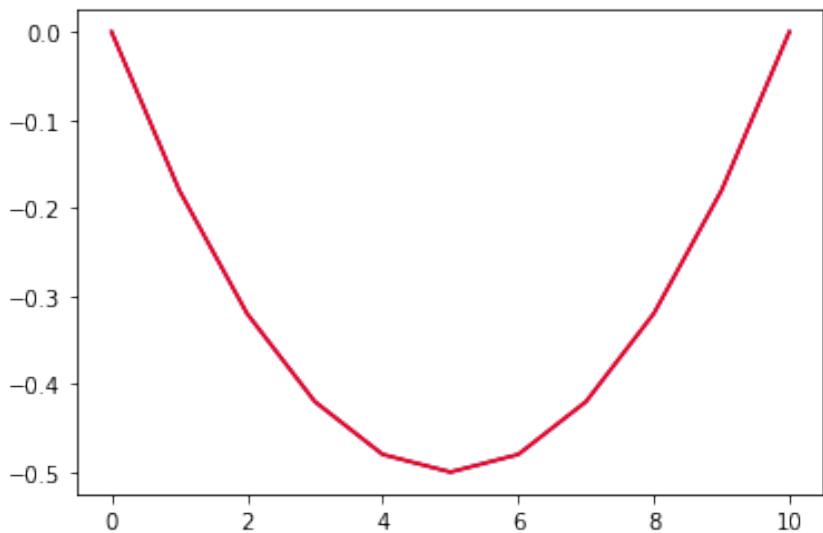
iteraz = 1



iteraz = 2

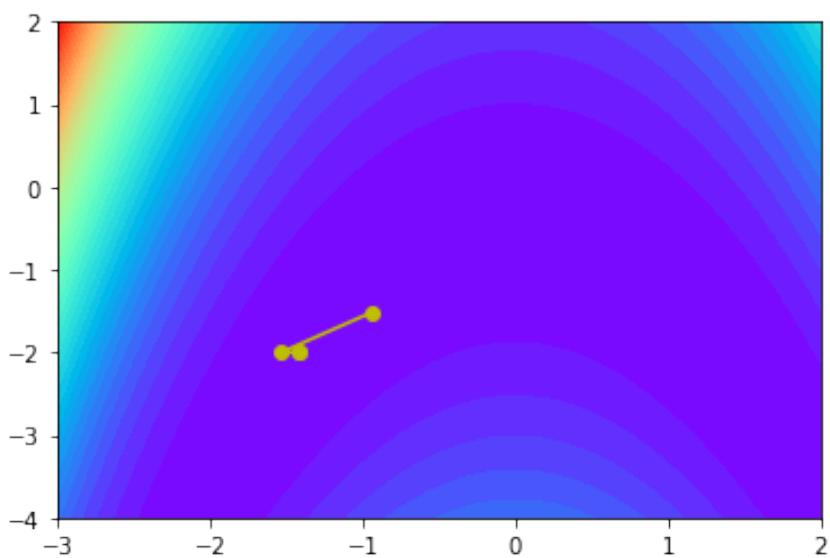


iteraz = 3



```
In [8]: theta_lims = (-4.0,2.0,-3.0,2.0)
N1 = 50
N2 = 50
plot_FN(y_meas,y_param,theta_lims,N1,N2,E)
```

```
x2.shape = (50,)
x1.shape = (50,)
V.shape = (50, 50)
```



Torna al par. 5

In [9]:

Appendice I

Esempi con sistemi sotto-determinati

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menù "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#%get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import PIL.Image as Image
from scipy.signal import gaussian

In [2]: def soft_thresholding(x,c):
         return np.clip(1-c/np.abs(x),0.0,np.Inf) * x

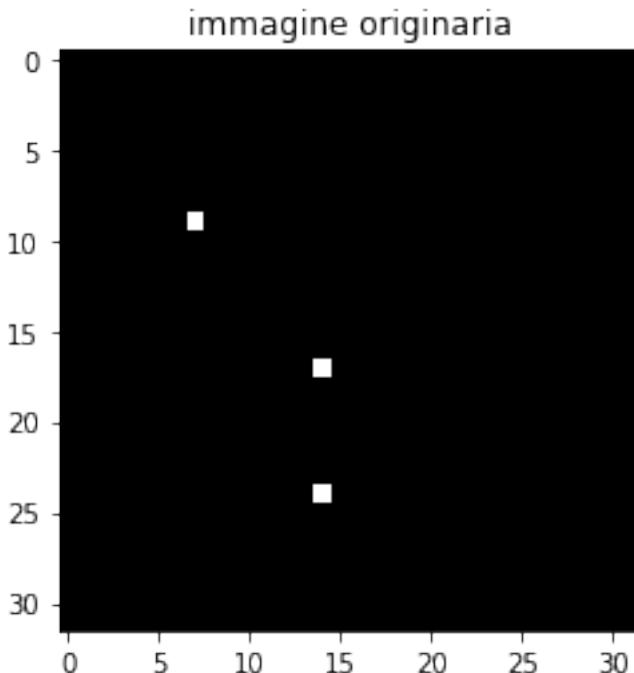
In [3]: def LASSO(A,b,x,rho,lambda,maxiter=10):
         b = np.squeeze(b)
         n = A.shape[1]
         ATb = A.T @ b;
         z = np.zeros(n)
         u = np.zeros(n)
         for i in range(maxiter):
             # scaled ADMM:
             x = np.linalg.solve( A.T @ A + rho*np.eye(n) , ATb + rho*(z-u) )
             # soft thresholding:
             z = soft_thresholding(x+u,lambda/rho)
             # dual variable:
             u = u + x - z
         return x

In [4]: # esempio delle tre stelle:
Nx = 32
```

```

Ny = Nx
stars = np.zeros((Nx,Ny))
stars[9,7] = 255
stars[24,14] = 255
stars[17,14] = 255
plt.figure(1); plt.imshow(stars,'gray'); plt.title("immagine originaria"); plt.show()
# trasformo l'immagine in un vettore, costruito concatenando le righe della matrice
# il valore vero del vettore incognito nel problema inverso:
n = Nx*Ny
x_veri = np.reshape(stars,(n,1))

```



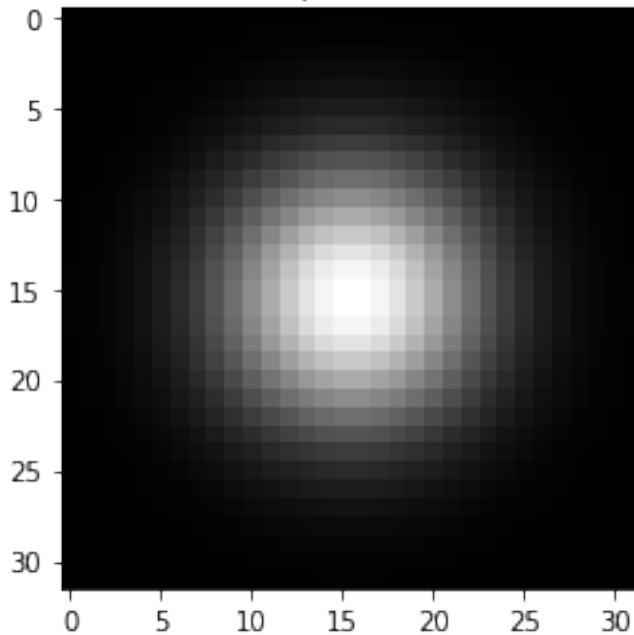
```

In [5]: # creo una matrice che rappresenti lo sfocamento creato dal mezzo che separa l'oggetto
sigma = 5
g = gaussian(Nx, sigma) # costruzione della gaussiana:
                        # e' una gaussiana di centro c e deviazione standard sigma
                        # calcolata nei punti [0:N-1];
                        # e' infatti ipotizzato che la sequenza di dati lunga N abbia
                        # che corrono da 0 a N-1
G = 255 * np.atleast_2d(g).T @ np.atleast_2d(g)
print(np.max(np.max(G)))
plt.figure(1); plt.imshow(G,'gray'); plt.title("sfocamento di un punto luminoso centrale")

```

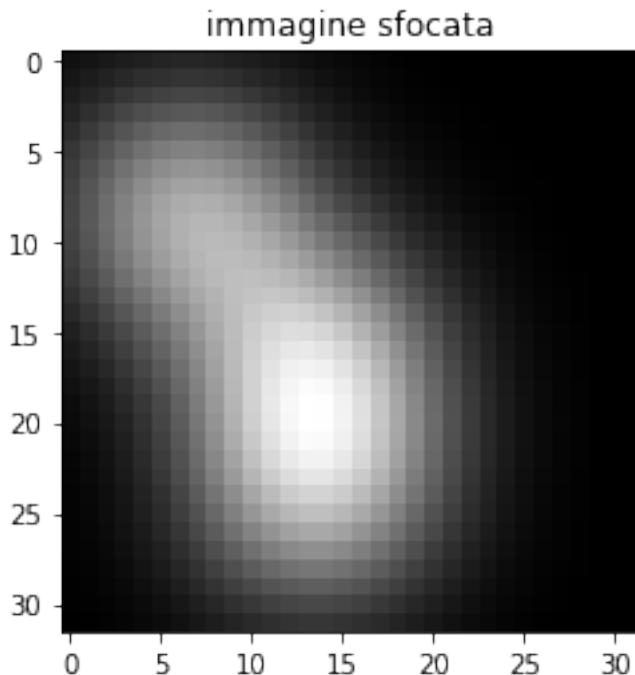
252.46270760603784

sfocamento di un punto luminoso centrale



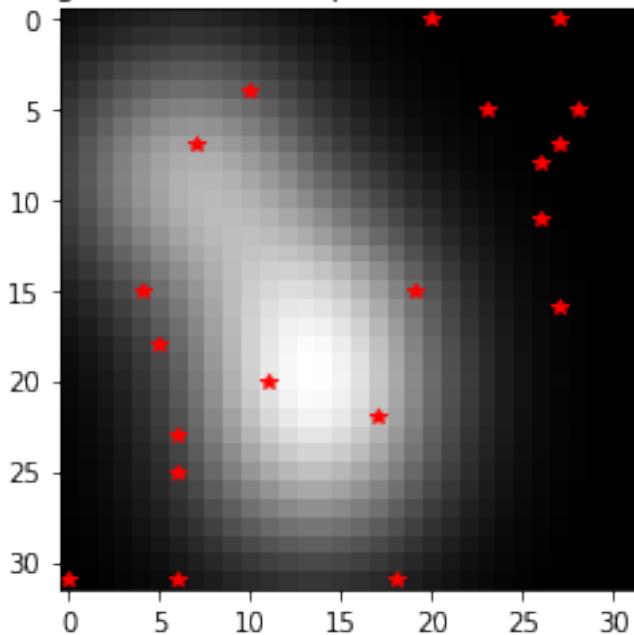
```
In [6]: # creo una base "T" di funzioni di sfocamento, una colonna per ogni pixel:  
T = np.zeros((n,n))  
gg = np.concatenate((g, g))  
ind = 0  
for ix in range(Nx):  
    for iy in range(Ny):  
        W = np.ones((Nx,Ny))  
        Gxmin = np.max([-Nx,-int(Nx/2)-ix]); #print("Gxmin = ",Gxmin)  
        Gymin = np.max([-Ny,-int(Ny/2)-iy]); #print("Gymin = ",Gymin)  
        #plt.figure(jj+1); plt.imshow(G[Gxmin:,Gymin:], 'gray'); plt.show()  
        Wxstart = ix+(Gxmin+int(Nx/2)); #print("Wxstart = ",Wxstart)  
        Wystart = iy+(Gymin+int(Ny/2)); #print("Wystart = ",Wystart)  
        tmpG = G[Gxmin:np.min([-Wxstart,-1]),Gymin:np.min([-Wystart,-1])]; #print(''  
        W[Wxstart:Wxstart+tmpG.shape[0],Wystart:Wystart+tmpG.shape[1]] = tmpG  
        #plt.figure(jj+1); plt.imshow(W, 'gray'); plt.show()  
        T[:,ind] = np.reshape(W,(n))  
        ind = ind + 1  
#endfor  
#endfor
```

```
In [7]: # costruisco l'immagine sfocata:  
s = T @ x_veri;  
plt.figure(1); plt.imshow(np.reshape(s,(Nx,Nx)), 'gray'); plt.title("immagine sfocata")  
plt.show()
```



```
In [8]: # ora prendo le misure, come campionamento random dell'immagine sfocata (che è quel  
Nb = round(n / 50)  
Ib = np.round(n*np.random.rand(Nb)+0.5); #print("ib = ",ib)  
plt.figure(1); plt.imshow(np.reshape(s,(Nx,Nx)), 'gray'); plt.title("immagine sfocata")  
irib = Ib//Ny  
plt.plot(irib,Ib-irib*Ny, 'r*')  
plt.show()  
# costruisco il vettore delle misure, facendo attenzione a rimuovere i doppiioni:  
b = s[np.int32(Ib)]  
Nb = len(b); print("numero di punti di misura = ",Nb)  
A = T[np.int32(Ib),:]; print("A.shape = ",A.shape)
```

immagine sfocata con punti di misura (in rosso)

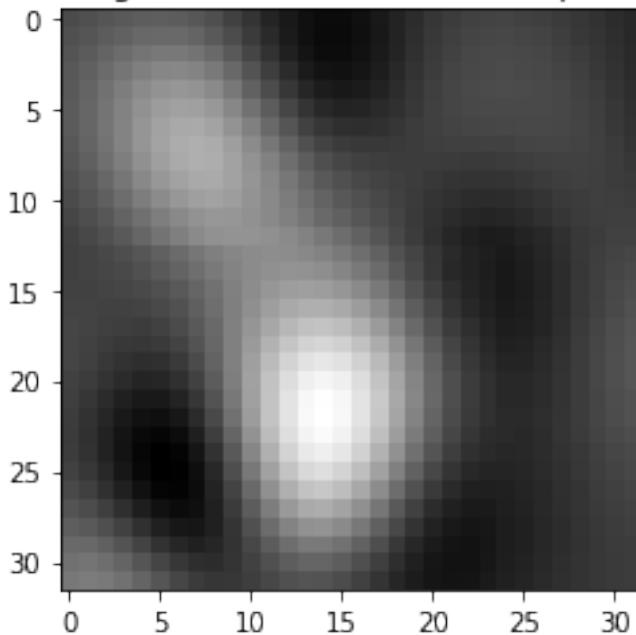


```
numero di punti di misura = 20
A.shape = (20, 1024)
```

```
In [9]: # provo a ricostruire con i minimi quadrati:
```

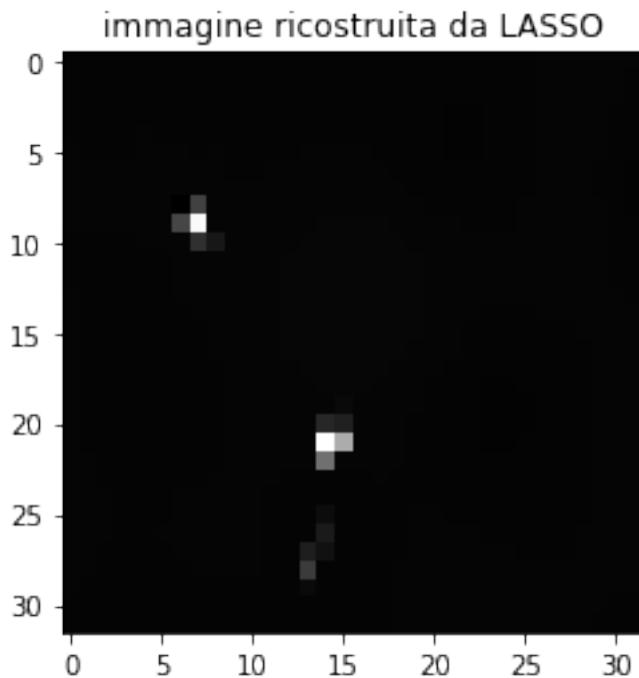
```
Q,R = np.linalg.qr(A.T)
z1 = np.linalg.solve(R.T,b)
x_LS = Q[:,0:Nb] @ z1
plt.figure(1); plt.imshow(np.reshape(x_LS,(Nx,Nx)), 'gray'); plt.title("immagine ricostruita")
plt.show()
```

immagine ricostruita dai minimi quadrati



In [10]: # ricostruisco "x" con ADMM-LASSO:

```
lam = 30.0
x_LASSO = LASSO(A,b,np.zeros(n),rho=0.2,lambda=lam,maxiter=100)
plt.figure(1); plt.imshow(np.reshape(x_LASSO,(Nx,Nx)), 'gray'); plt.title("immagine")
plt.show()
```



Chi vuole provare ADMM su delle foto reali, può guardare questa libreria:
<http://people.csail.mit.edu/jrk/proximal.pdf>

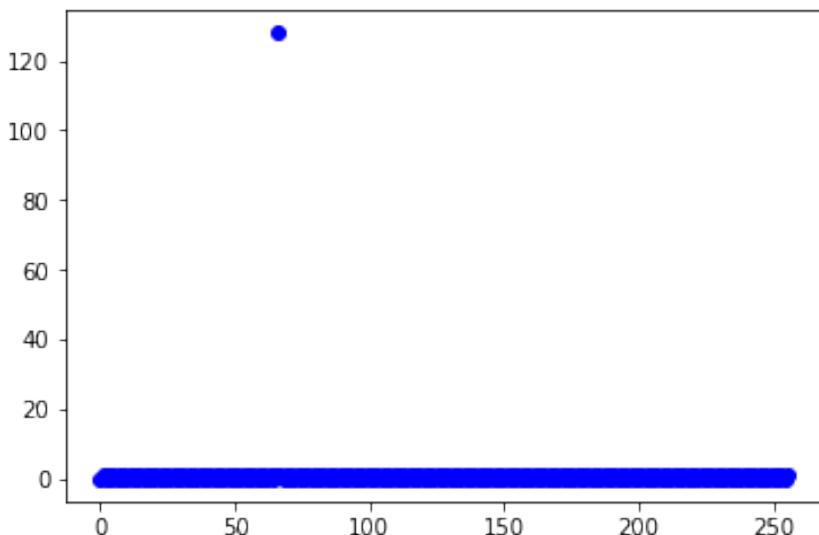
Inoltre, utilizzando una trasformata che sparsifichi, come la trasformata wavelet, questa tecnica può essere utilizzata anche con immagini che nella loro base naturale sono dense, vedere ad esempio l'esperimento della fotocamera ad un pixel dell'Università di Rice: <https://www.ams.org/publicoutreach/math-history/hap7-pixel.pdf>

e, come vedremo più avanti, nella Magneto-Risonanza per Immagini (MRI).

```
In [11]: # esempio dello spettro di una sinusoida
N = 256;
nu = 33;
omega_nu = 2*np.pi*nu/N;
u = np.cos(omega_nu*np.arange(0,N));
# costruisco la matrice relativa alla trasformata Coseno Discreta (DCT):
W = np.zeros((N,N));
for nu in range(N):
    #la (5.1.3) sarebbe: W(:,nu+1) = exp(-sqrt(-1)*2*pi*nu/N*(0:N-1));
    W[:,nu] = np.cos(np.pi*nu/N*np.arange(0,N));
#endfor
```

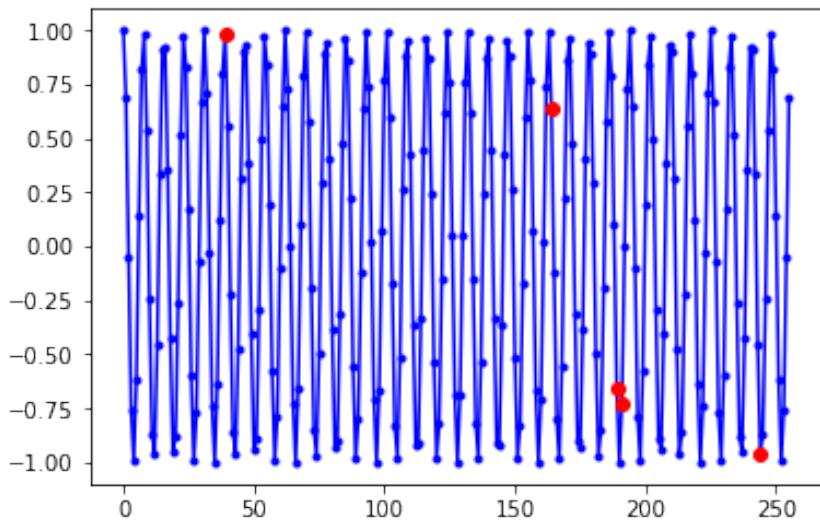
```
# e mi calcolo lo spettro U:  
U = W.T @ u  
plt.figure(); plt.plot(np.abs(U), 'bo')
```

Out[11]: [`<matplotlib.lines.Line2D at 0x7febd9c60080>`]

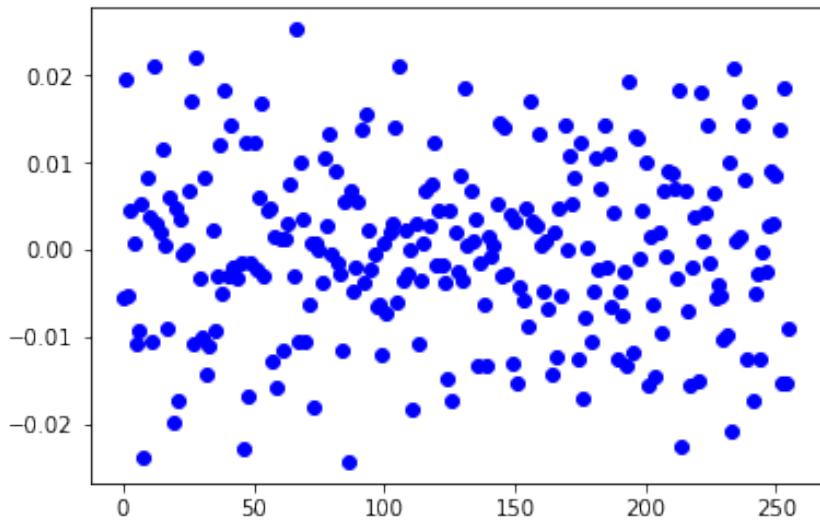


Torna al par. 6.2

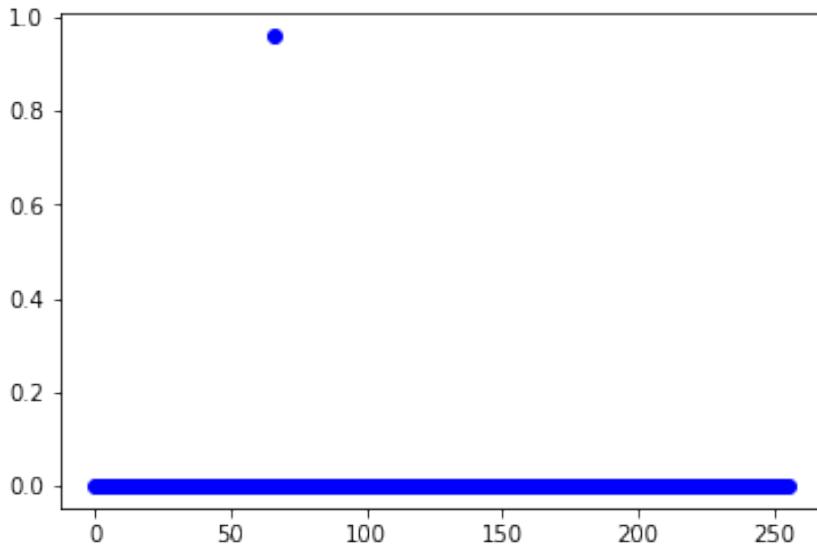
```
In [12]: # pero' in questo modo ho dovuto raccogliere tutti i campioni "U(k)";  
# vediamo se con il compressed sensing ottengo lo stesso risultato,  
# avendo raccolto solo un numero Nb < N di campioni, presi a caso:  
Nb = round(N / 50)  
Ib = np.round(N*np.random.rand(Nb)+0.5); #print("ib = ",ib)  
# costruisco il vettore delle misure, facendo attenzione a rimuovere i doppioni:  
b = u[np.int32(Ib)]  
Nb = len(b); print("numero di punti di misura = ",Nb)  
plt.figure(); plt.plot(u,'b.-'); plt.plot(Ib,b,'ro')  
A = W[np.int32(Ib),:]; print("A.shape = ",A.shape)  
  
numero di punti di misura = 5  
A.shape = (5, 256)
```



```
In [13]: # provo a ricostruire con i minimi quadrati:  
Q,R = np.linalg.qr(A.T)  
z1 = np.linalg.solve(R.T,b)  
x_LS = Q[:,0:Nb] @ z1  
plt.figure(); plt.plot(x_LS, 'bo');  
plt.show()
```



```
In [14]: # ricostruisco lo spettro con ADMM-LASSO:
lam = 0.1
x_LASSO = LASSO(A,b,np.zeros(N),rho=0.25,lambda=lam,maxiter=200)
plt.figure(); plt.plot(x_LASSO, 'bo');
plt.show()
```



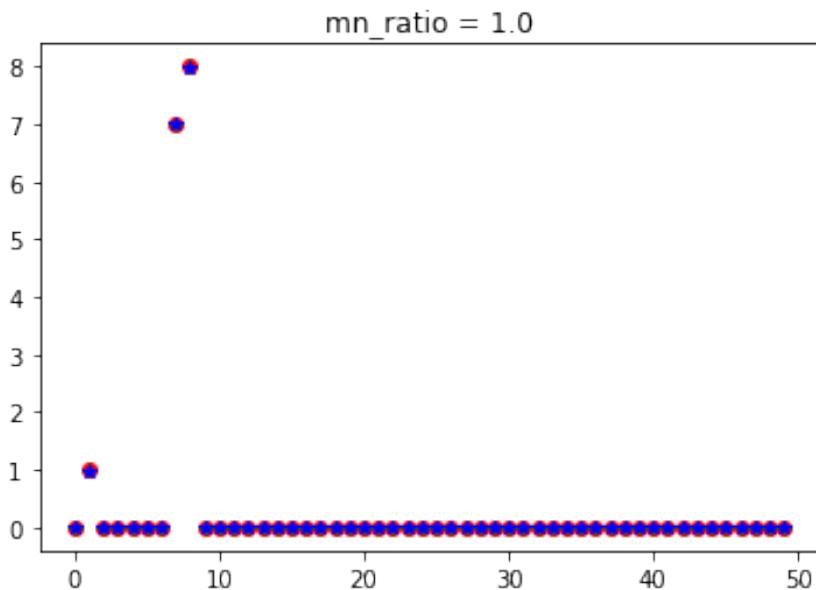
```
In [15]: # analisi di un caso generico:
n = 50
x_veri = np.zeros(n); x_veri[1] = 1.0; x_veri[7] = 7.0; x_veri[8] = 8.0;
v_mn_ratio = [1.0,0.95,0.5,0.45,0.2,0.1]
for mn_ratio in v_mn_ratio:
    m = int(mn_ratio * n)
    A = np.random.rand(m,n)
    b = A @ x_veri
    lam = 0.1
    x_LASSO = LASSO(A,b,np.zeros(n),rho=0.2,lambda=lam,maxiter=50)
    if m==n:
        x_LS = np.linalg.solve( A.T @ A , A.T @ b )
    elif m<n:
        Q,R = np.linalg.qr(A.T)
        z1 = np.linalg.solve(R.T,b)
        x_LS = Q[:,0:m] @ z1
```

```

#endif
plt.figure(); plt.plot(x_veri, 'ro'); plt.plot(x_LS, 'g+'); plt.plot(x_LASSO, 'b*')
plt.title("mn_ratio = "+str(mn_ratio)); plt.show()
print("||A x_LS - b||_2 = ",np.linalg.norm(A @ x_LS - b))
print("||A x_LASSO - b||_2 = ",np.linalg.norm(A @ x_LASSO - b))
print("||A x_LS - b||_2 + lam*||x_LS||_1 = ",np.linalg.norm(A @ x_LS - b)+lam*
print("||A x_LASSO - b||_2 + lam*||x_LASSO||_1 = ",np.linalg.norm(A @ x_LASSO
print("lam*||x_LS||_1 = ",lam*np.linalg.norm(x_LS,1))
print("lam*||x_LASSO||_1 = ",lam*np.linalg.norm(x_LASSO,1))

#endiffor

```

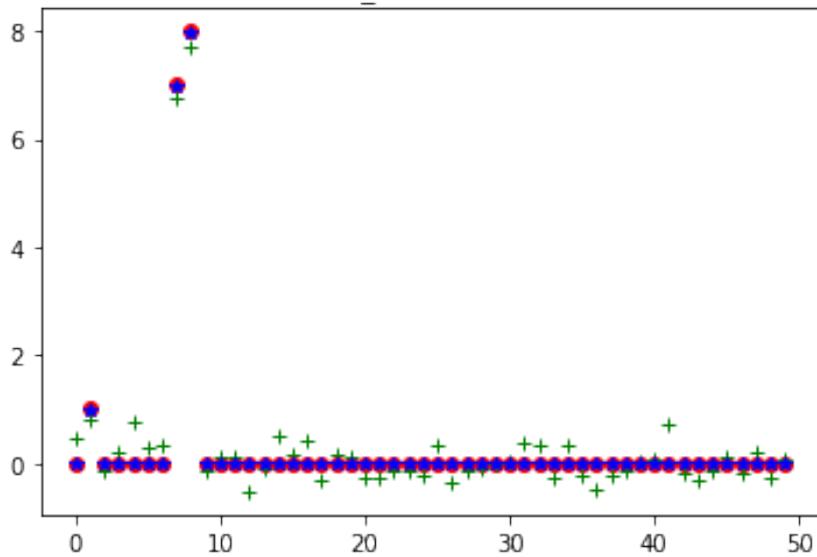


```

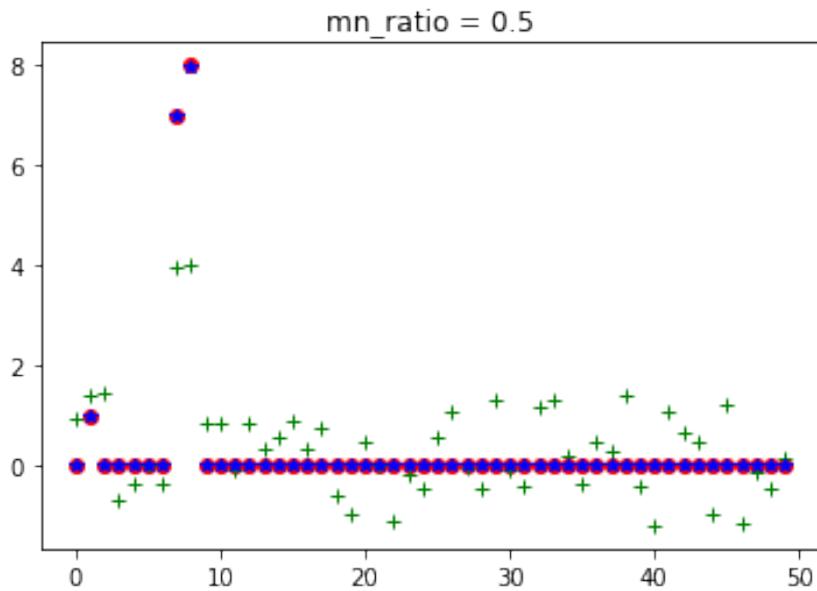
||A x_LS - b||_2 =  4.3324724241999994e-12
||A x_LASSO - b||_2 =  0.07230622955158089
||A x_LS - b||_2 + lam*||x_LS||_1 =  1.6000000001284669
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 =  1.671865802426628
lam*||x_LS||_1 =  1.6000000001241343
lam*||x_LASSO||_1 =  1.5995595728750471

```

mn_ratio = 0.95

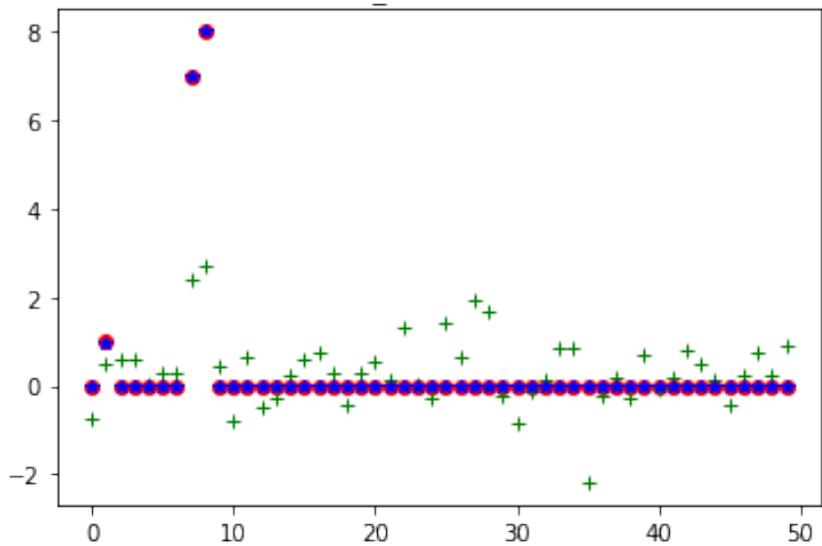


```
||A x_LS - b||_2 =  2.969758394275688e-14
||A x_LASSO - b||_2 =  0.06846250653185937
||A x_LS - b||_2 + lam*||x_LS||_1 =  2.6803894187133124
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 =  1.6683449488279454
lam*||x_LS||_1 =  2.6803894187132826
lam*||x_LASSO||_1 =  1.599882442296086
```



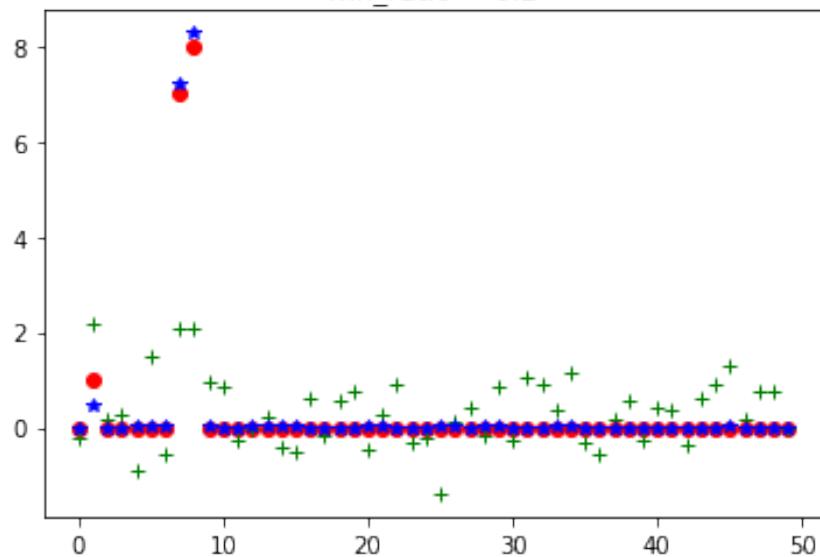
```
||A x_LS - b||_2 =  2.4940265110194155e-14
||A x_LASSO - b||_2 =  0.08096118516890448
||A x_LS - b||_2 + lam*||x_LS||_1 =  3.971054176553493
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 =  1.6814112888934436
lam*||x_LS||_1 =  3.971054176553468
lam*||x_LASSO||_1 =  1.600450103724539
```

mn_ratio = 0.45

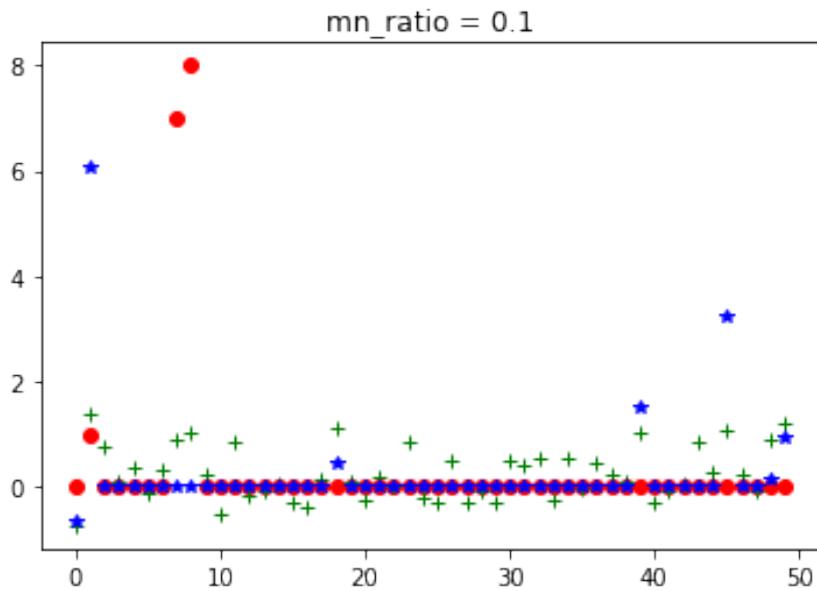


```
||A x_LS - b||_2 = 1.4438022517112667e-14
||A x_LASSO - b||_2 = 0.06354298127001896
||A x_LS - b||_2 + lam*||x_LS||_1 = 3.2252503693903134
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 = 1.6646101221801617
lam*||x_LS||_1 = 3.2252503693902987
lam*||x_LASSO||_1 = 1.6010671409101427
```

mn_ratio = 0.2



```
||A x_LS - b||_2 = 7.717447657366157e-15
||A x_LASSO - b||_2 = 0.09077461619290755
||A x_LS - b||_2 + lam*||x_LS||_1 = 3.1724183718797425
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 = 1.739540390521157
lam*||x_LS||_1 = 3.172418371879735
lam*||x_LASSO||_1 = 1.6487657743282496
```



```
||A x_LS - b||_2 =  6.720289191984444e-15
||A x_LASSO - b||_2 =  0.15982759882762568
||A x_LS - b||_2 + lam*||x_LS||_1 =  2.1702041999051853
||A x_LASSO - b||_2 + lam*||x_LASSO||_1 =  1.470808593817838
lam*||x_LS||_1 =  2.1702041999051787
lam*||x_LASSO||_1 =  1.3109809949902123
```

Torna al par. [6.2](#)

In [16]:

Appendice J

Analisi di Fourier stazionaria

J.1 Esempi risposta in frequenza

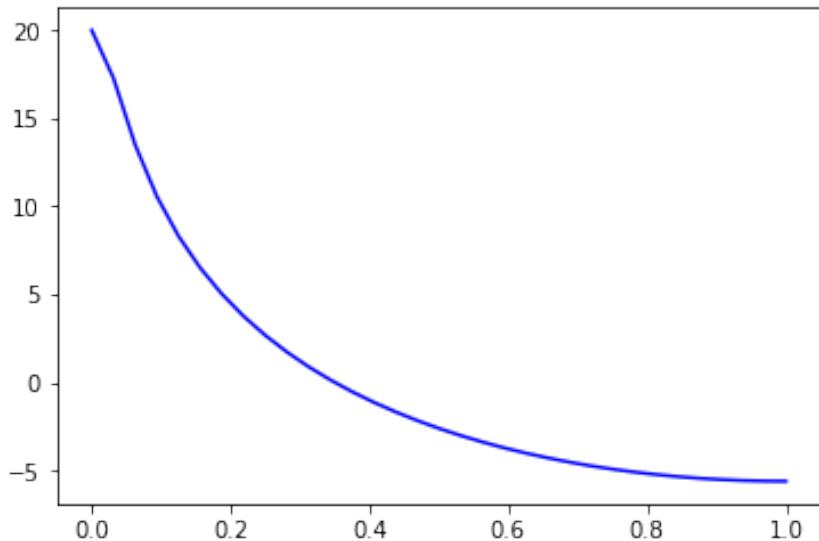
```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matplotlib inline"
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
from libreria_NLALD.freqresp import *
from libreria_NLALD.hamming import *
from numpy.fft import *
from libreria_NLALD.algo_fft_1D import *
from libreria_NLALD.psd Welch import *
from matplotlib.pyplot import psd
from libreria_NLALD.sistema_meccanico_3gdl import *
from libreria_NLALD.testBartlett_wn import *

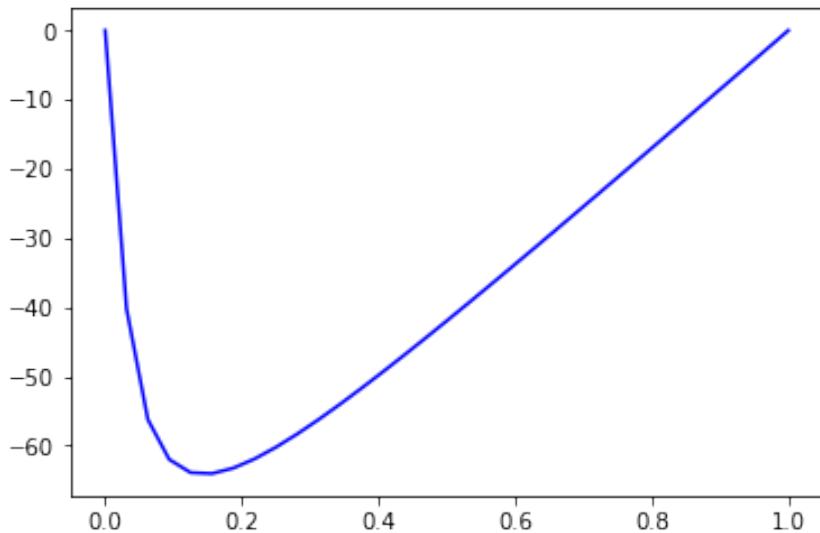
M1 = 27.0
M2 = 120.0
M3 = 11246.0
K1 = 180000000.0
K2 = 60000000.0
K3 = 120000000.0
C1 = 50000.0
C2 = 46000.0
C3 = 240000.0
coordinate di partenza: y1=0.06, y2=0.06, y3=0.05
```

```
In [2]: a = np.array([1., -0.9])
b = np.array([1.])
```

```
N = 64
u = np.zeros(N); u[0] = 1.0
h = simula_DLTI(b,a,u)
H = freqresp(h, 'b-', False)
```

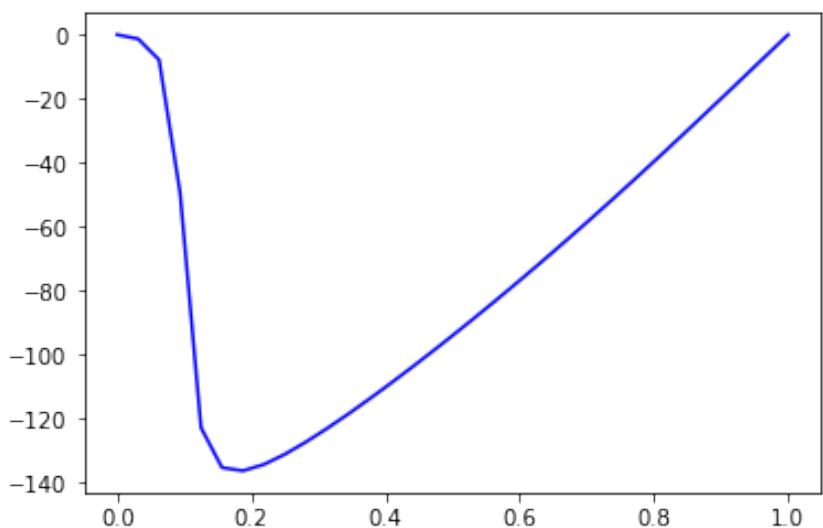
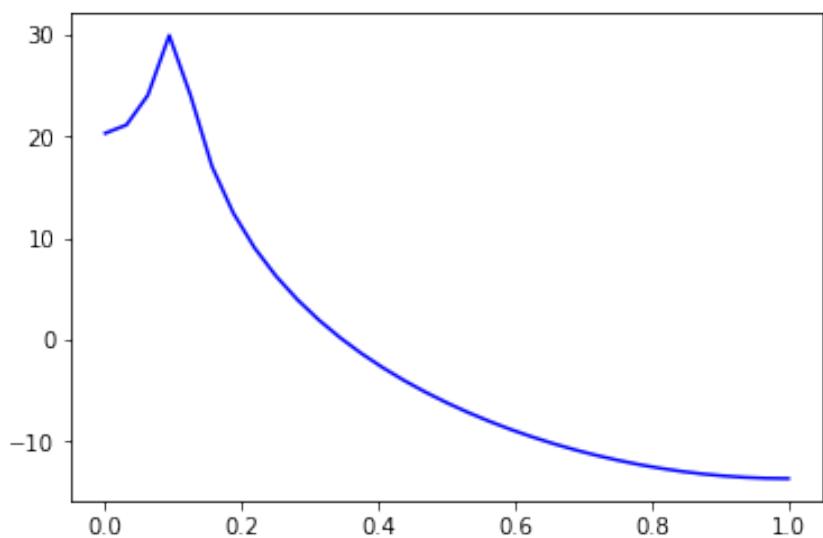
```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOLibro_MNAD/sito_web_MNAD
if nb==1: b=np.array([b, 0]); nb += 1; #endif # serve per non inserire "if" ne
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOLibro_MNAD/sito_web_MNAD
y[n] = 1./a[0] * np.sum(np.array([ np.dot(-a[1:na],y_past), vb ]));
```





```
In [3]: r = 0.95; # modulo delle radici
phi = 0.1*np.pi; # fase delle radici
a = np.array([1., -2.*r*np.cos(phi), r**2])
b = np.array([1.])
N = 64
u = np.zeros(N); u[0] = 1.0
h = simula_DLTI(b,a,u)
H = freqresp(h,'b-',False)
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPRIO/libro_MNAD/sito_web_MNAD
if nb==1: b=np.array([b, 0]); nb += 1; #endif # serve per non inserire "if" ne
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPRIO/libro_MNAD/sito_web_MNAD
y[n] = 1./a[0] * np.sum(np.array([ np.dot(-a[1:na],y_past), vb ]));
```



Torna al par. [7.5.1](#)

J.2 Algoritmo FFT

```
In [4]: # sequenza di dati complessi
xi = np.array([1.0+1.0j, 1.3+1.8j, 1.7+4.0j, 2.0+1.3j, 1.0+4.0j, 4.4+1.0j, 1.1+3.0j
              2.2+2.0j, 3.3+2.2j, 2.0+4.5j, 1.5+6.0j, 6.0+1.4j, 1.4+4.2j, 1.7+1.7j, 1.3+1.0j])
N = len(xi)
# calcolo della DFT mediante sommatoria:
u = xi.copy()
F = np.zeros((N,N),dtype=np.complex128)
U = np.zeros(N,dtype=np.complex128)
for k in range(0,N):
    for n in range(0,N):
        Wkn = np.exp(-1.j*(2*np.pi/N)*k*n);
        U[k] = U[k] + u[n]*Wkn
        F[k,n] = Wkn
    #endfor
#endfor
Uref = fft(xi)
print("verifica U: ", np.sum(np.abs(np.real(U - Uref))) + np.sum(np.abs(np.imag(U - Uref))))
```



```
# calcolo della DFT mediante FFT
#
u = xi.copy()
Uf = fft_1D(u) # calcolo della FFT "sul posto".
print("verifica Uf: ", np.sum(np.abs(np.real(Uf - Uref))) + np.sum(np.abs(np.imag(Uf - Uref))))
```



```
verifica U: 4.2870151872875795e-12
n= 0000      j= 0000
n= 0001      j= 1000
n= 0010      j= 0100
n= 0011      j= 1100
n= 0100      j= 0010
n= 0101      j= 1010
n= 0110      j= 0110
n= 0111      j= 1110
n= 1000      j= 0001
n= 1001      j= 1001
n= 1010      j= 0101
n= 1011      j= 1101
n= 1100      j= 0011
n= 1101      j= 1011
n= 1110      j= 0111
n= 1111      j= 1111
verifica Uf: 1.8207657603852567e-13
```

```
In [5]: # calcolo della DFT mediante prodotto matrice-vettore
UM = F * np.matrix(xi).T
print(UM.shape)
print(Uref.shape)
#err = UM - Uref
#print(err)
print("verifica: ", np.sum(np.abs(np.real(UM.T - Uref))) + np.sum(np.abs(np.imag(UM.T - Uref))))
verifica: 4.2870151872875795e-12
```

```
In [6]: u = (1/N) * np.conj(F.T) @ np.atleast_2d(U).T
#print(u.shape)
#print(xi.shape)
err = u.T - xi
#print(err)
u = np.squeeze(np.asarray(u))
print("verifica: ", np.sum(np.abs(np.real(u.T - xi))) + np.sum(np.abs(np.imag(u.T - xi))))
verifica: 1.602273869139026e-12
```

```
In [7]: np.linalg.cond(F)
```

```
Out[7]: 1.0000000000000001
```

```
In [8]: for i in range(0,N):
    print(np.linalg.cond(F[0:i+1,0:i+1]))
#endif
```

```
1.0
10.153170387608856
55.25096203306495
185.3804643113365
422.32430633912713
714.1782099981486
961.3784698113866
1059.5396645604515
961.4559695870241
715.8178879532614
432.2903634983653
207.29294840288514
76.20161789583885
```

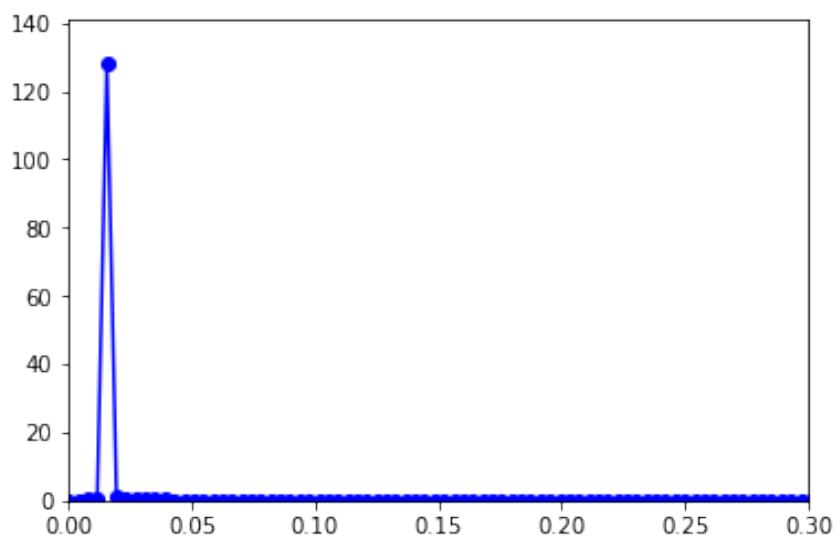
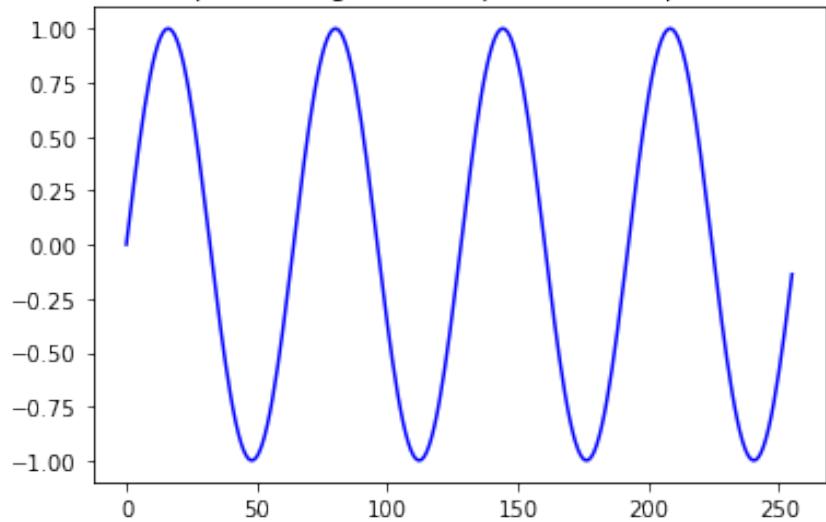
```
20.404594474756678
4.0000000000000002
1.0000000000000001
```

Torna al par. [7.6.1.3](#)

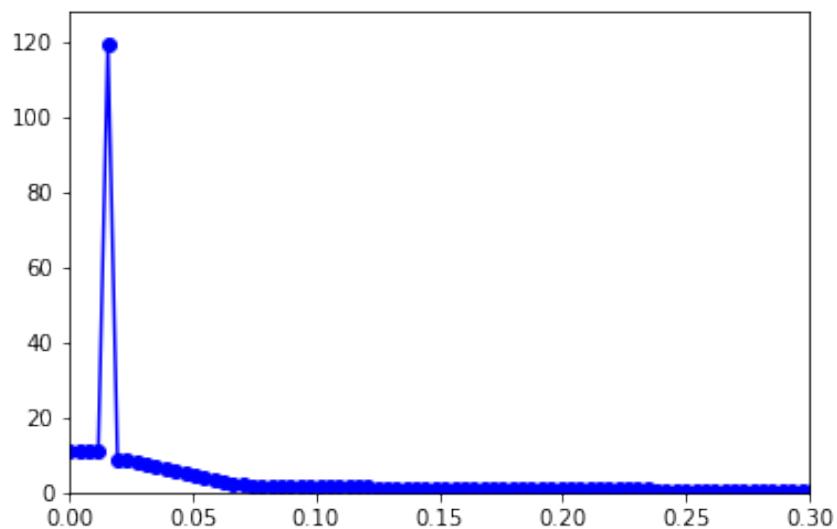
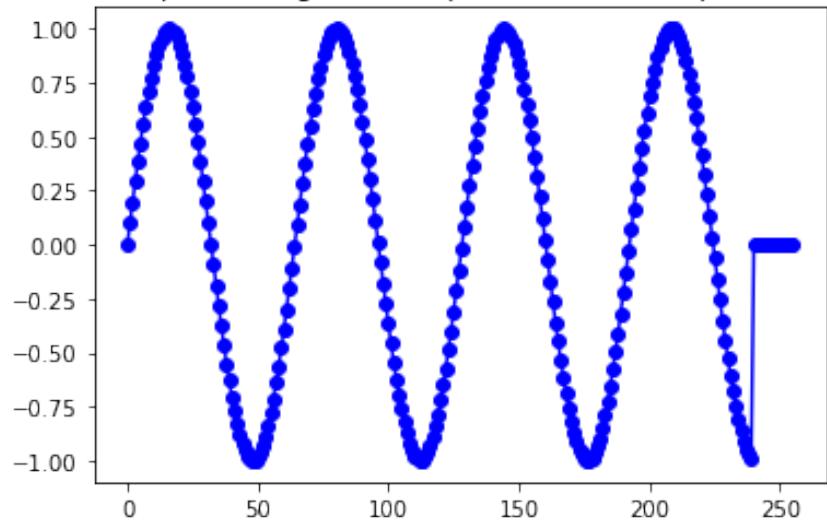
J.3 Aspetti pratici DFT 1-d

```
In [9]: # troncamento di una sequenza sinusoidale pura ad un istante che non è un multiplo
# sinusoidale:
fc = 1.
Tc = 1./fc
N = 256
t = np.arange(0.,N*Tc,Tc)
f = 0.0156
s = np.sin(2*np.pi*f*t);    # segnale sinusoidale alla frequenza "f" (Hz)
# FFT di un multiplo intero del periodo:
S = fft(s);
plt.figure(1); plt.plot(t, s, 'b-'); plt.title('sequenza lunga un multiplo intero')
vf = np.arange(0,1/Tc,1/(Tc*N))
plt.figure(2); plt.plot(vf, abs(S), 'b-o'); plt.axis([0., 0.3, 0., max(abs(S))*1.1])
# FFT di un multiplo NON intero del periodo:
tail = np.concatenate((np.zeros(240), s[N-1-np.arange(15,-1,-1)]))
sc = s - tail
Sc = fft(sc)
plt.figure(3); plt.plot(t, sc, 'b-o'); plt.title('sequenza lunga un multiplo NON-intero')
plt.figure(4); plt.plot(vf, np.abs(Sc), 'b-o'); plt.axis([0., 0.3, 0., np.max(np.abs(Sc))])
plt.figure(5); plt.plot(t, sc, 'b-o'); plt.plot(t, -tail, 'r-+'); plt.title('sequenza lunga un multiplo NON-intero')
plt.figure(6); plt.plot(vf, abs(Sc), 'b-o'); plt.axis([0., 0.3, 0., max(abs(Sc))]);
# notare che il prolungamento per periodicità introduce un "salto", discontinuità in
# larga banda; questo spiega il risultato.
```

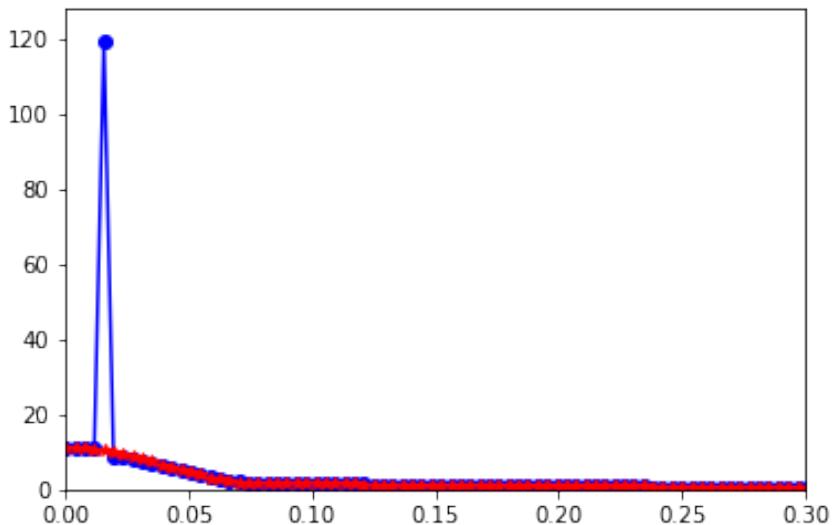
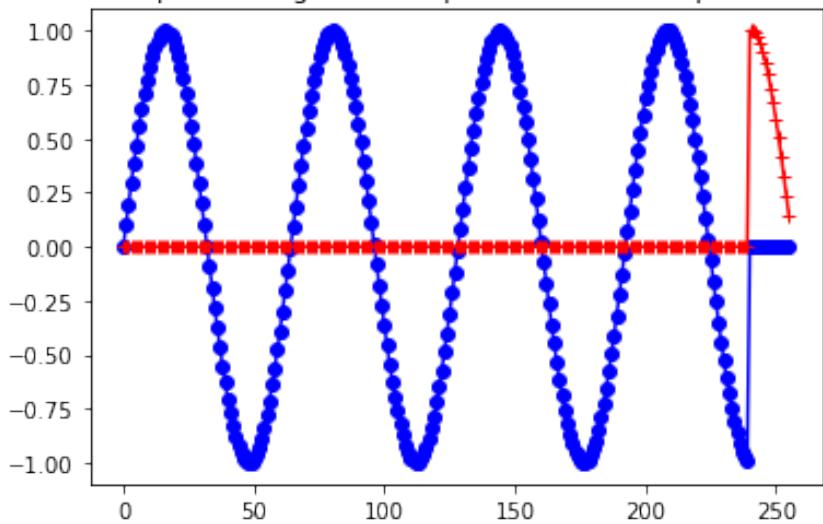
sequenza lunga un multiplo intero del periodo



sequenza lunga un multiplo NON-intero del periodo



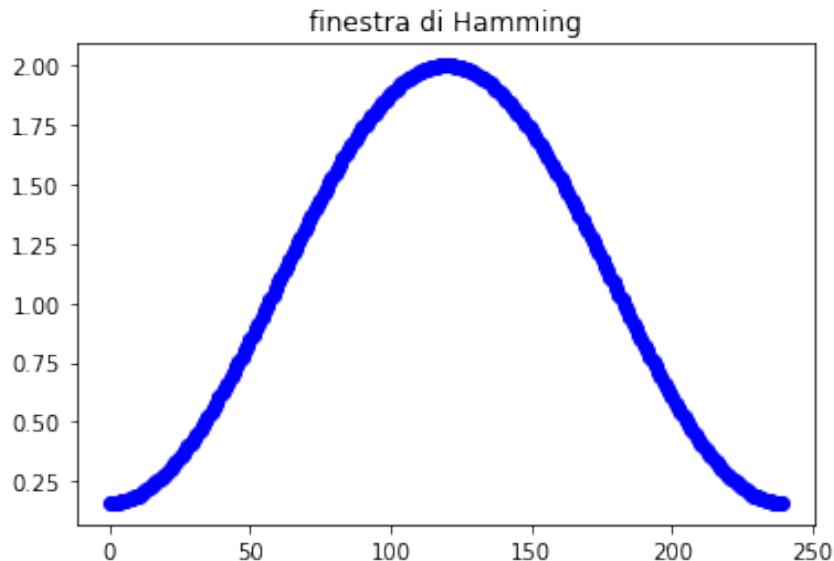
sequenza lunga un multiplo NON-intero del periodo

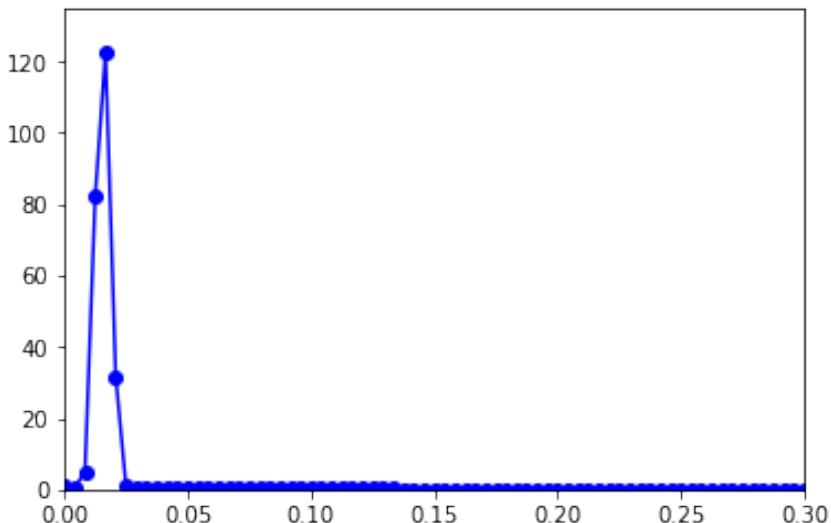
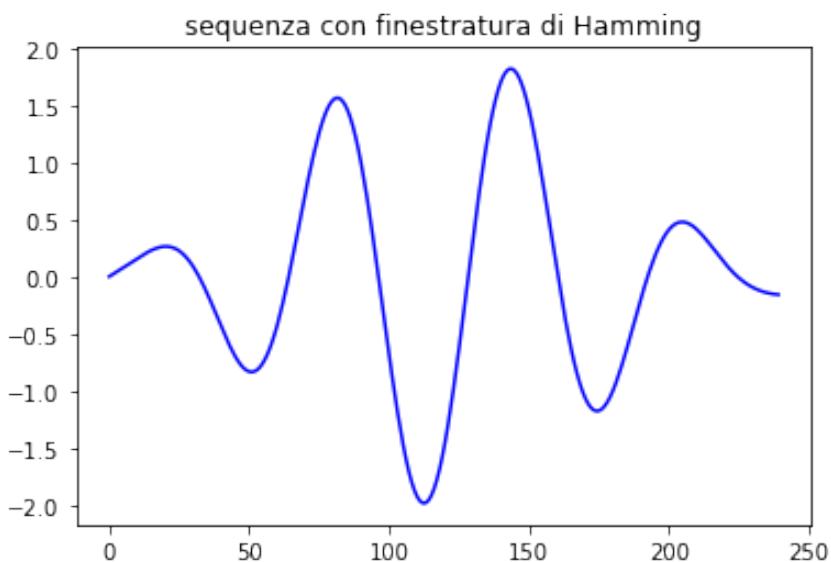


Torna al par. [7.7.2](#)

J.4 Finestratura

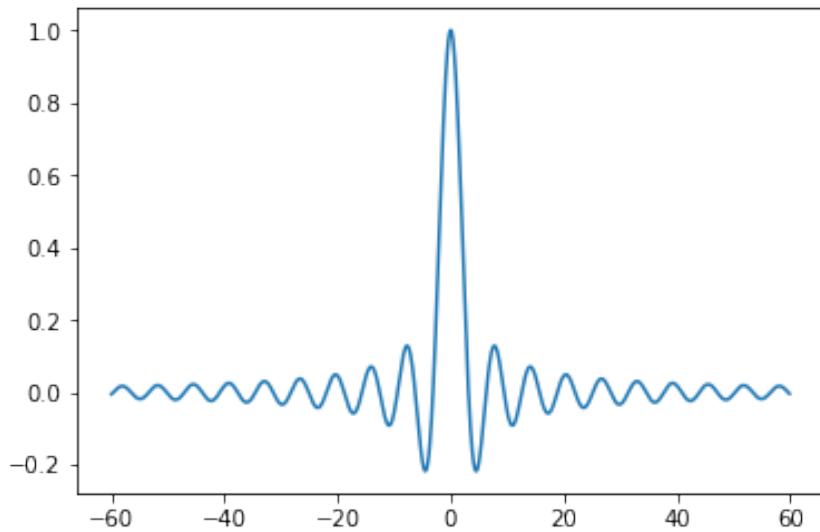
```
In [10]: # rimedio: FFT di un multiplo NON intero del periodo ma CON FINESTRATURA OPPORTUNA
N = 240
t = np.arange(0.,N*Tc,Tc)
w = 2*hamming(N); plt.figure(31); plt.plot(w,'b-o'); plt.title('finestra di Hamming')
s = np.sin(2*np.pi*f*t) * w
S = fft(s)
plt.figure(11); plt.plot(t, s, 'b-'); plt.title('sequenza con finestratura di Hamming')
vf = np.arange(0,1/Tc,1/(Tc*N))
plt.figure(12); plt.plot(vf, np.abs(S), 'b-o'); plt.axis([0., 0.3, 0., np.max(np.abs(S))])
```





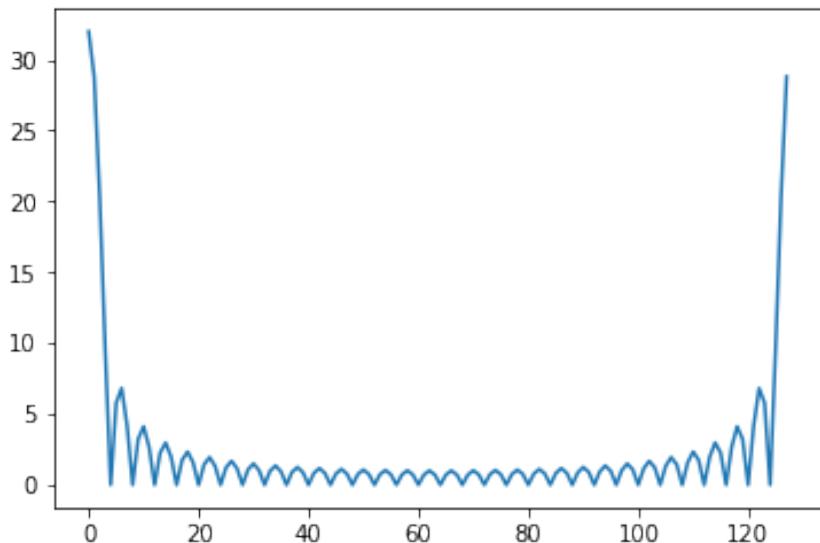
```
In [11]: # grafico della funzione sinc:  
x = np.arange(-60.0,60.0,0.1)  
plt.figure(21); plt.plot(x, np.sin(x)/x)
```

```
Out[11]: [<matplotlib.lines.Line2D at 0x7f93f1d40ac8>]
```



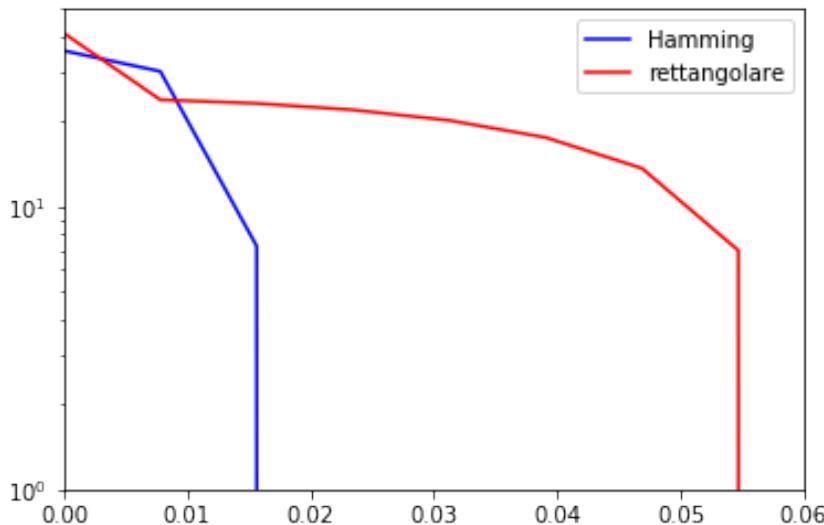
```
In [12]: N = 128
fr = np.zeros(N); fr[0:32]=1.0
FR = fft(fr)
plt.figure(31); plt.plot(abs(FR))
```

```
Out[12]: [<matplotlib.lines.Line2D at 0x7f93f1d04630>]
```



```
In [13]: #
    # DFT della finestra rettangolare e di quella di Hamming
    #
N = 128
nc = int(N/8.)
w = np.concatenate((hamming(7.*nc), np.zeros(nc)))
r = np.concatenate((np.ones(7*nc), np.zeros(nc)))  # finestra rettangolare
WdB = 20*np.log10( np.abs( fft(w) ) )
RdB = 20*np.log10( np.abs( fft(r) ) )
vf = np.arange(0,1/Tc,1/(Tc*N))
plt.figure(41); plt.semilogy(vf, WdB, 'b-'); plt.semilogy(vf, RdB, 'r-'); plt.axis
```

```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_laun
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_laun
if __name__ == '__main__':
```



Torna al par. [7.7.2](#)

J.5 Effetto **picket-fence**

```
In [14]: #
    # effetto "picket-fence" e tecnica di "zero-padding" (rimedio):
    #
Tc = 0.1
N = 256
t = np.arange(0., N*Tc, Tc)
#
f = 1/(Tc*N) * 12    # la frequenza cade ad una frequenza discreta risoluta dalla D
s = np.sin(2*np.pi*f*t)
S = fft(s)

plt.figure(51); plt.plot(t, s, 'b-'); plt.title('sinusoide a frequenza coincidente')
vf = np.arange(0, 1/Tc, 1/(Tc*N))
plt.figure(52); plt.plot(vf, np.abs(S), 'b-o'); plt.axis([0., 2., 0., np.max(np.abs(S))])
#
f = 1/(Tc*N) * 12.5    # la frequenza cade giusto a metà tra due frequenze discrete
s = np.sin(2*np.pi*f*t)
S = fft(s)

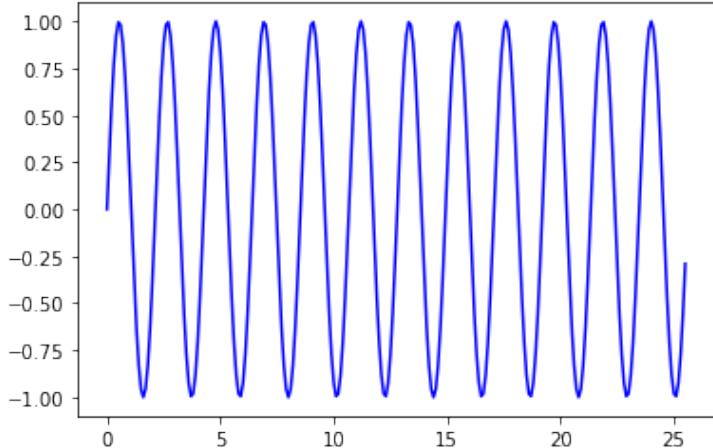
plt.figure(53); plt.plot(t, s, 'b-'); plt.title('sinusoide a metà' tra due frequenze discrete')
plt.figure(54); plt.plot(vf, np.abs(S), 'b-o'); plt.axis([0., 2., 0., np.max(np.abs(S))])
# NB: ci sono due valori consecutivi dello spettro che indicano la sinusoide
# rimedio: "zero-padding" (cioè aggiunta di zeri alla sequenza, previa finestratura)
```

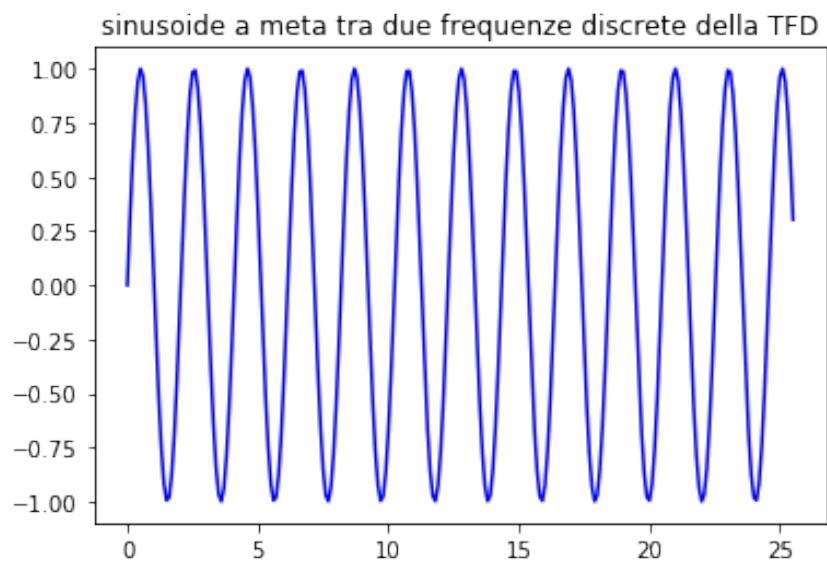
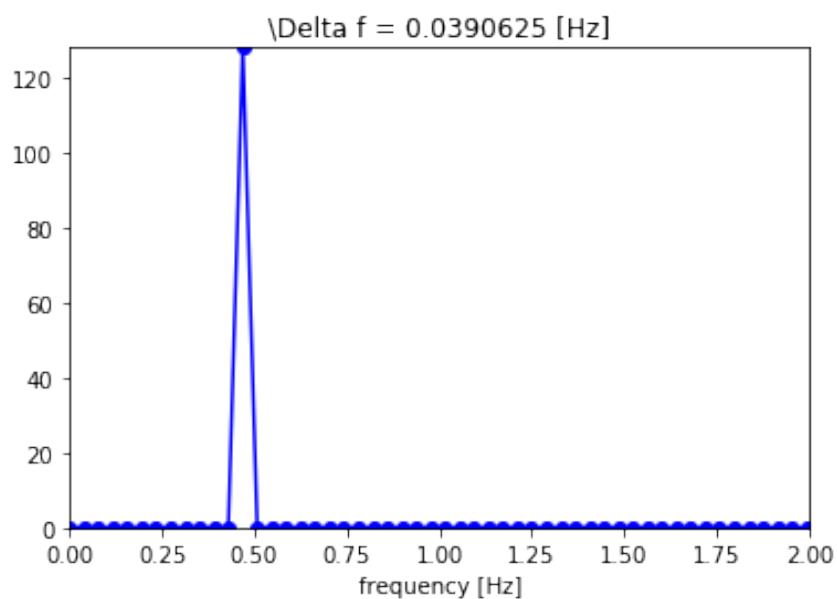
```

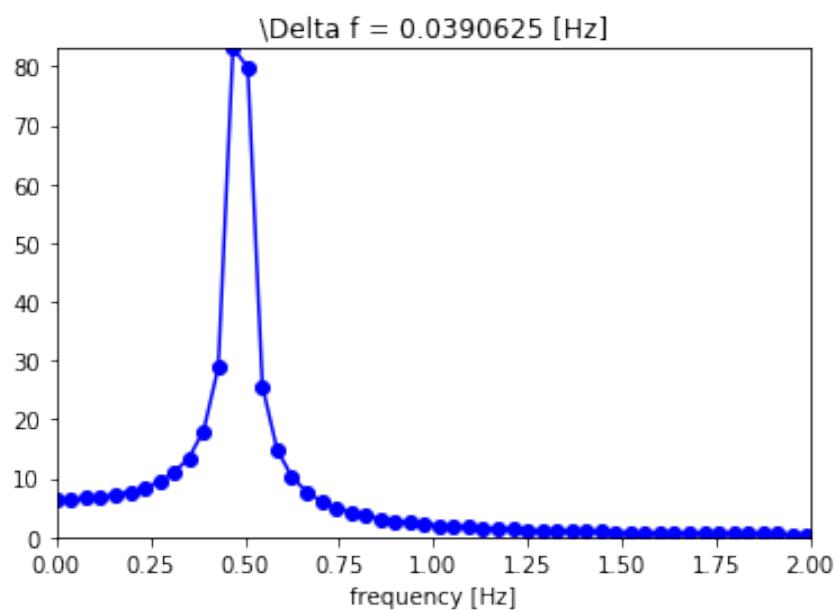
f = 1/(Tc*N) * 12.5    # la frequenza cade giusto a metà tra due frequenze discrete
w = hamming(N)
s = np.sin(2*np.pi*f*t) * w
s = np.concatenate((s, np.zeros(N)))
N = len(s)
t = np.arange(0., N*Tc, Tc)
S = fft(s)
vf = np.arange(0., 1/Tc, 1/(Tc*N))
plt.figure(55); plt.plot(t, s, 'b-'); plt.title('sinusoide a metà' tra due frequenze discrete')
plt.figure(56); plt.plot(vf, np.abs(S), 'b-o'); plt.axis([0., 2., 0., np.max(np.abs(S))])

```

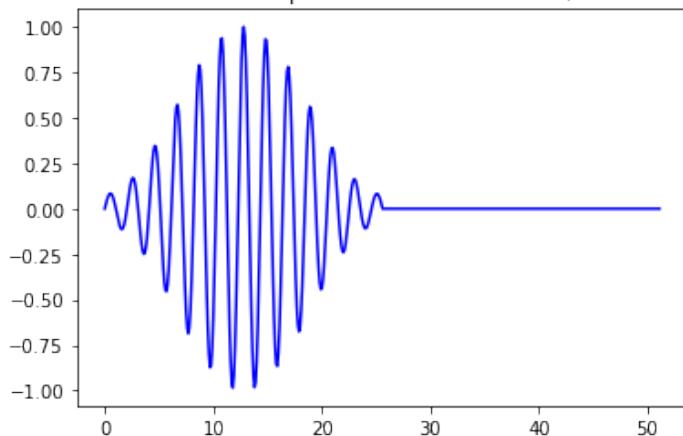
sinusoide a frequenza coincidente con una frequenza discreta della TFD

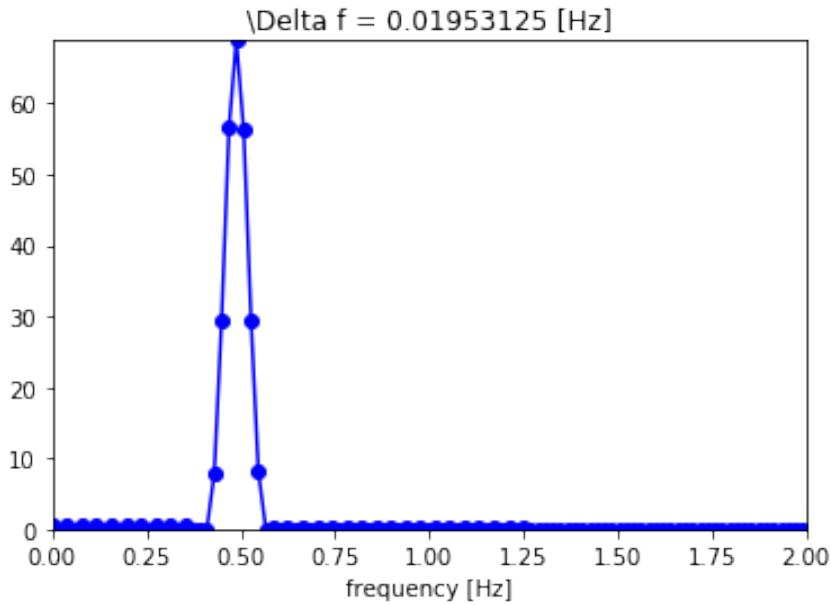




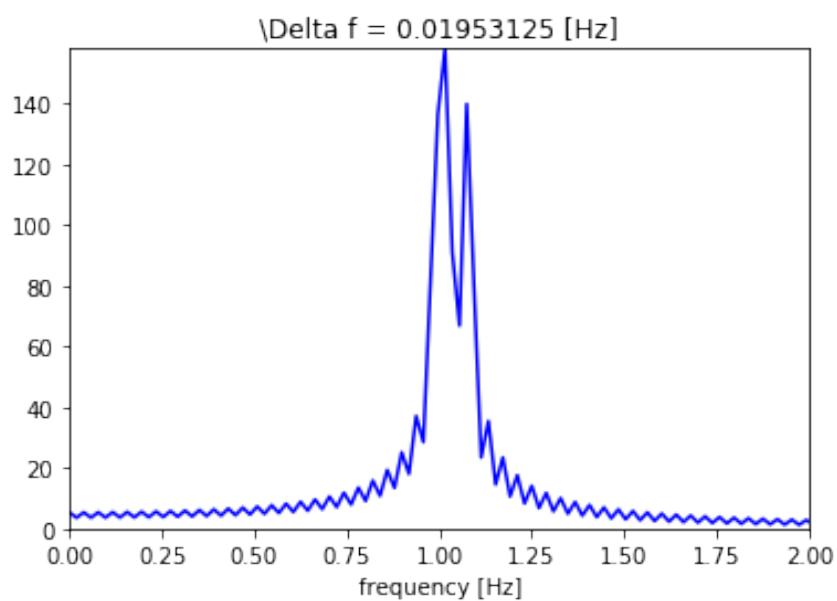
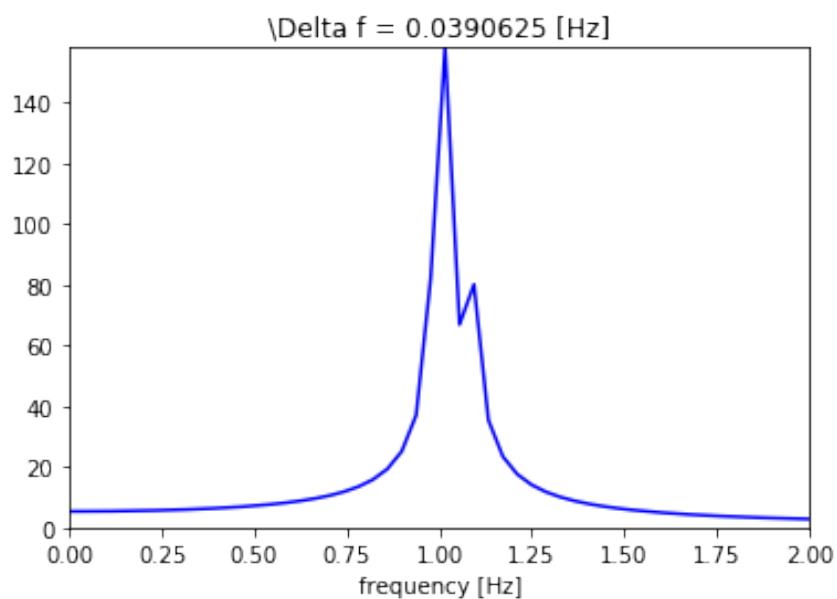


sinusoide a metà tra due frequenze discrete della TFD, con ZERO-PADDING

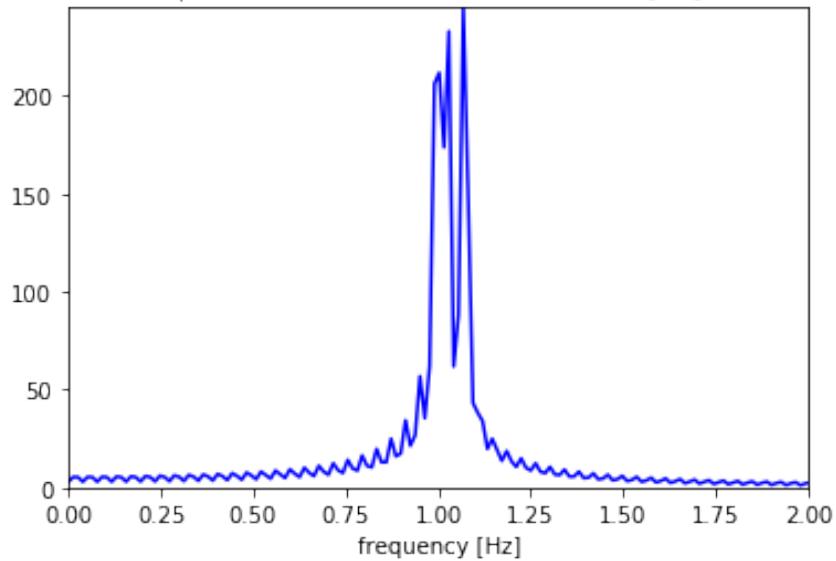




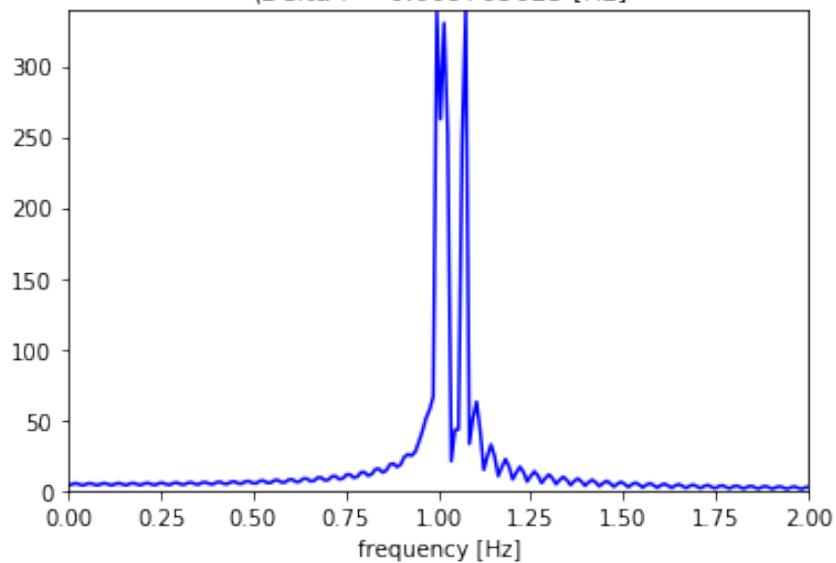
```
In [15]: #
    # risoluzione in frequenza: aumentando il numero di campioni, a parità di frequenz
    #
Tc = 0.1
n = 256
for i in range(1,7+1):
    N = n*i
    t = np.arange(0.,N*Tc-1e-9,Tc)
    s = np.sin(2*np.pi*1*t) + np.sin(2*np.pi*1.02*t) + np.sin(2*np.pi*1.07*t);
    if True and i>1:
        s[n*(i-1):N] = 0
    #endif
    S = fft(s);
    vf = np.arange(0,1./Tc-1e-9,1./(Tc*N))
    plt.figure(60+i), plt.plot(vf, np.abs(S), 'b-'), plt.axis([0., 2., 0., np.max(np
#endfor
```



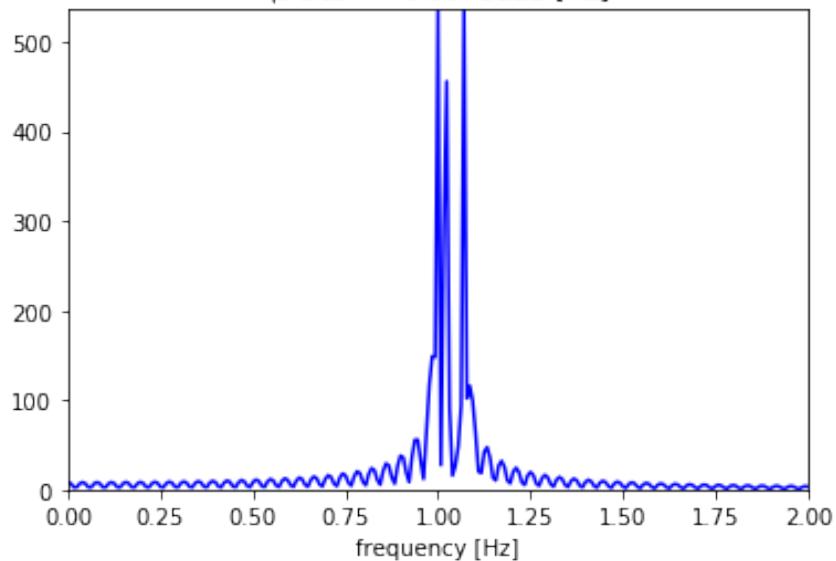
\Delta f = 0.01302083333333332 [Hz]



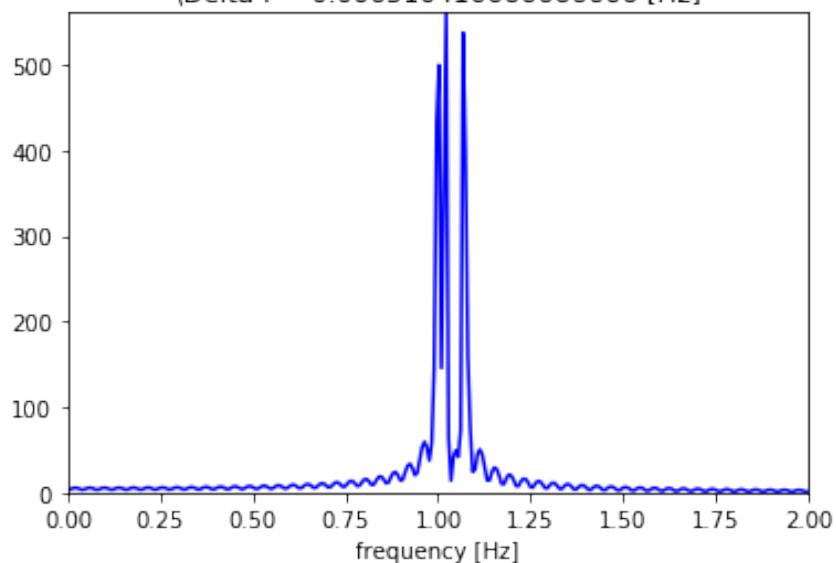
\Delta f = 0.009765625 [Hz]

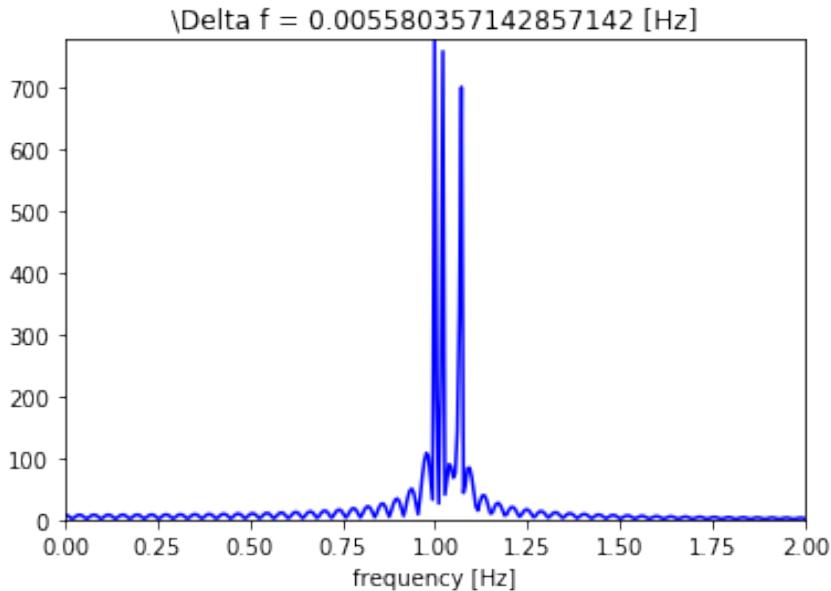


\Delta f = 0.0078125 [Hz]



\Delta f = 0.006510416666666666 [Hz]





Torna la par. 7.7.3.1

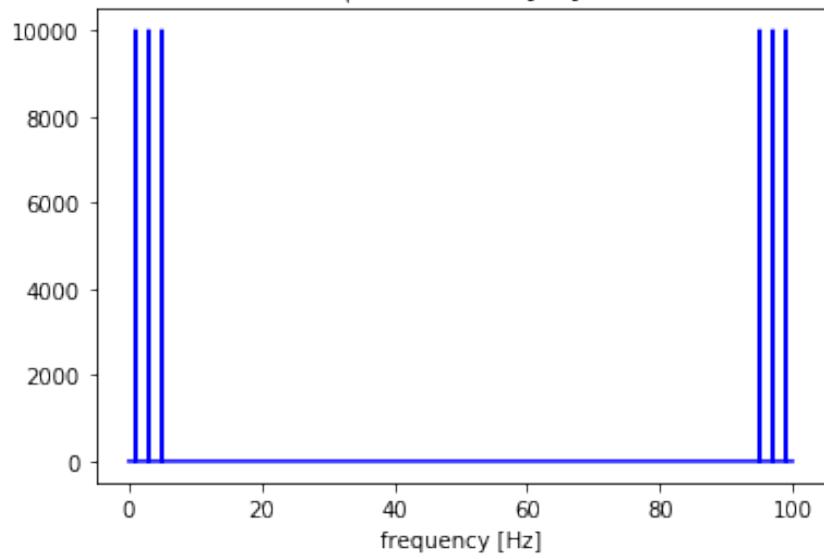
J.6 DFT in presenza di rumore nei dati

```
In [16]: #
    # presenza di rumore bianco a media nulla nei dati:
    #
    Tc = 0.01
    N = int(1/Tc)
    nper = 200
    t = np.arange(0.,nper,Tc)
    s = np.sin(2*np.pi*t) + np.sin(2*np.pi*3*t) + np.sin(2*np.pi*5*t)
    plt.figure(71); plt.plot(t,s,'b-'); plt.axis([0., nper/10., -8., 8.]), plt.title('S = fft(s)')
    vf = np.arange(0,1/Tc,1/(Tc*N*nper))
    plt.figure(72), plt.plot(vf, np.abs(S), 'b-'), plt.xlabel('frequency [Hz]'); plt.t
    # aggiungiamo il rumore:
    s = s + 6.*np.random.randn(len(s))
    plt.figure(73); plt.plot(t,s,'b-'); plt.axis([0., nper/10., -8., 8.]), plt.title('
    # vediamo la FFT mediata su più periodi:
    sm = np.reshape(s,(nper,N)).T
    for i in range(nper):
        sm[:,i] = np.abs(fft(sm[:,i].T).T)  # calcolo la FFT di sm "sul posto".
```

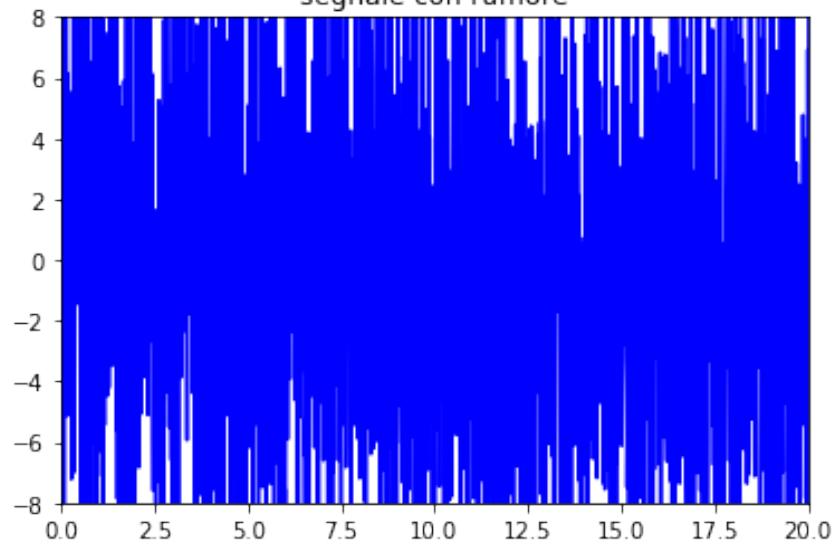
```
#endfor
ifig = 0
for i in range(10,nper+10,10):
    plt.figure(100+ifig), plt.plot(range(N), np.mean(sm[:,0:i],axis=1), 'b-')
    ifig = ifig + 1
#endfor
```



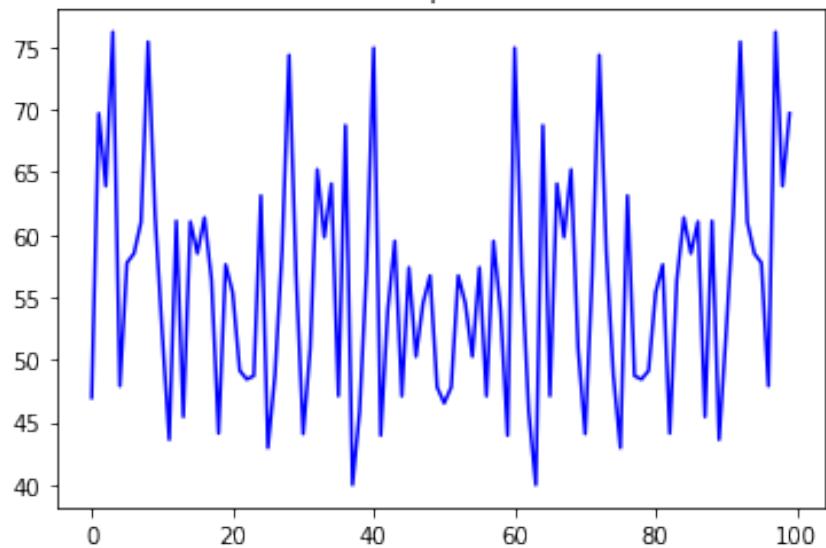
\Delta f = 1.0 [Hz]



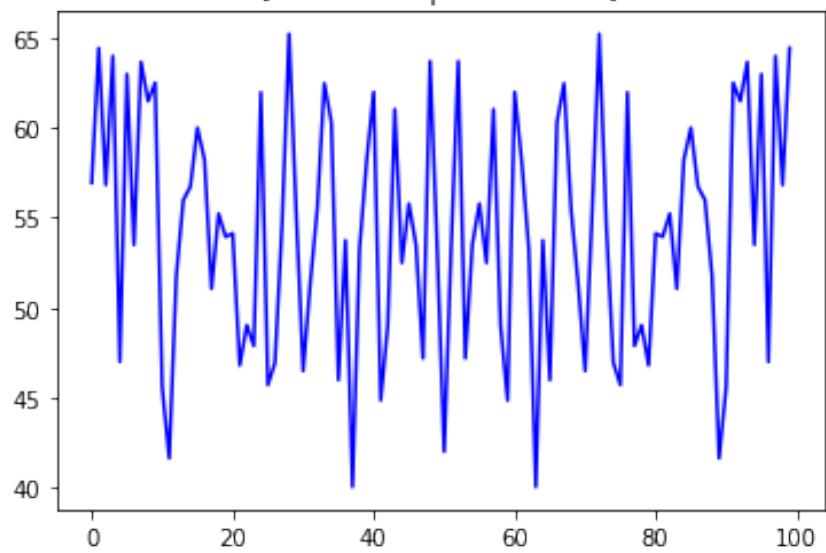
segnalet con rumore



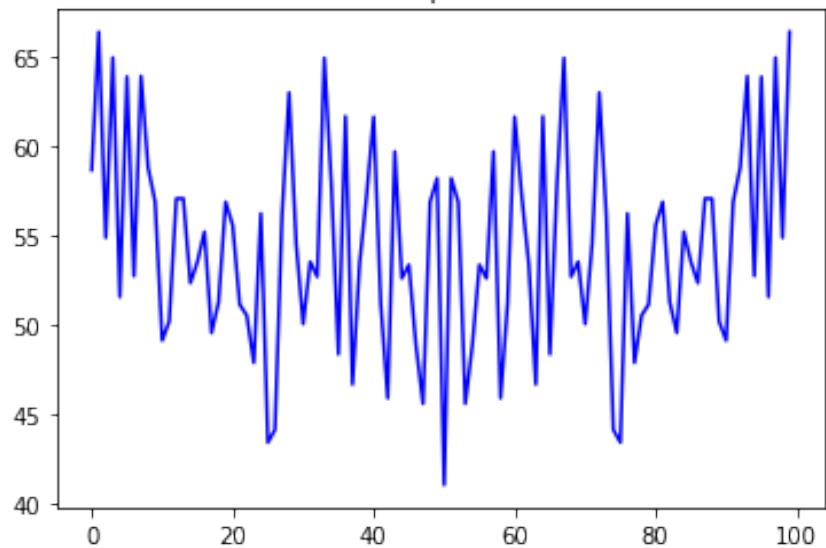
['numero di periodi = 10']



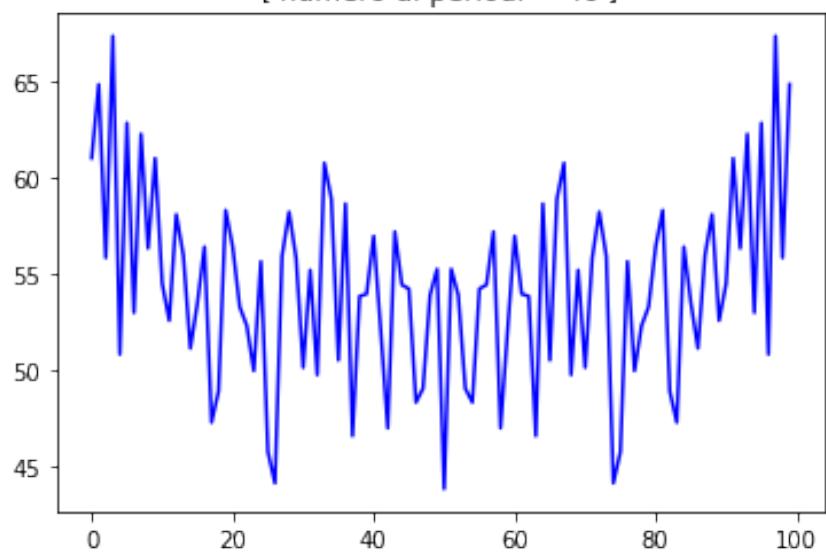
['numero di periodi = 20']



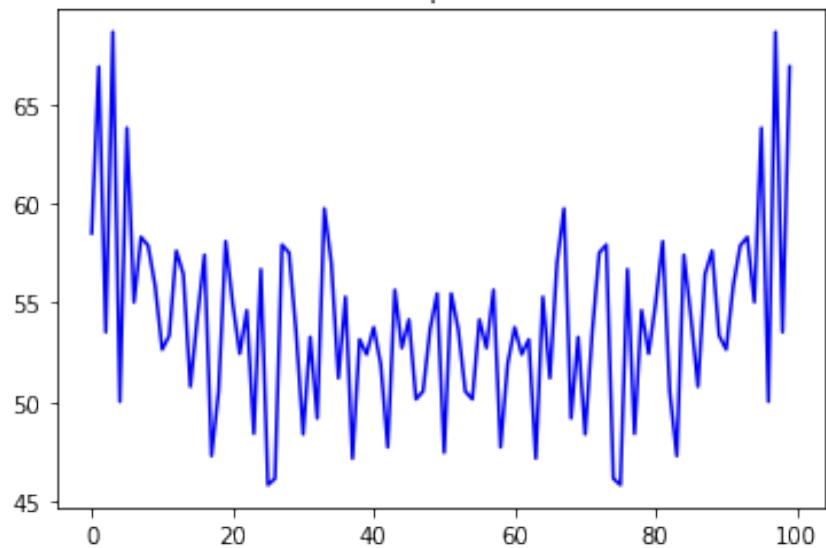
['numero di periodi = 30']



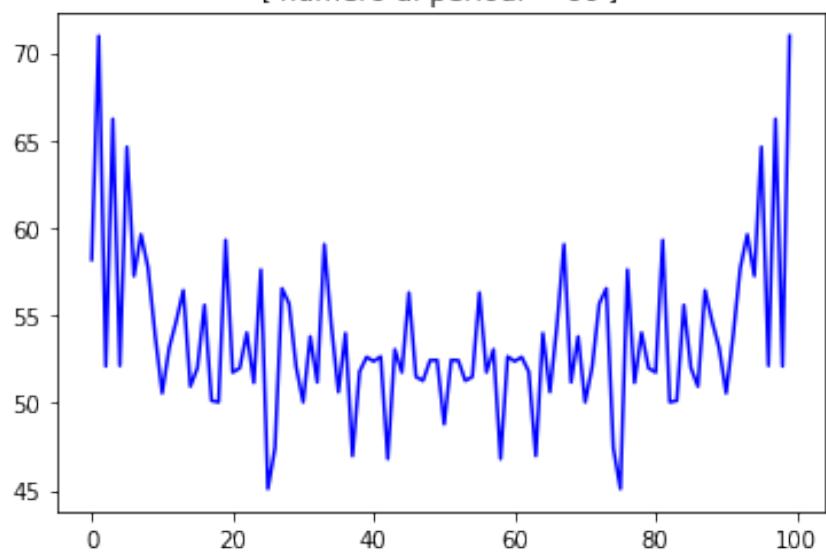
['numero di periodi = 40']



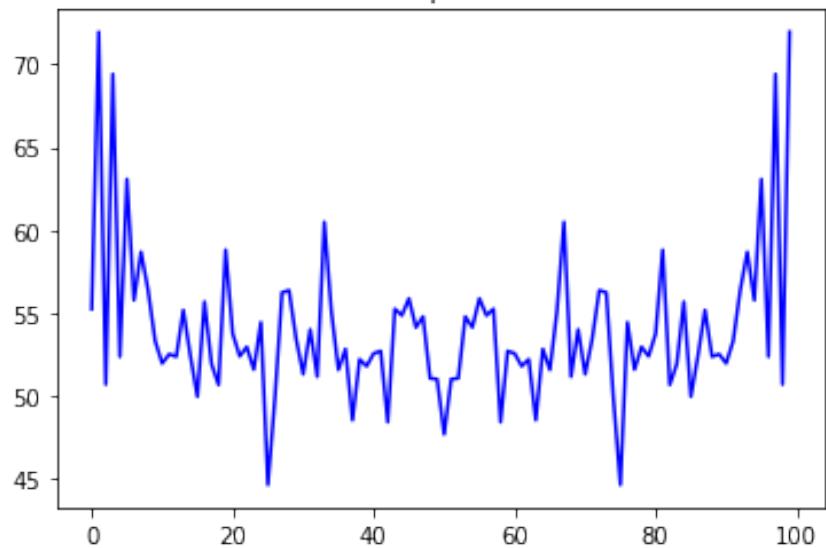
[*'numero di periodi = 50'*]



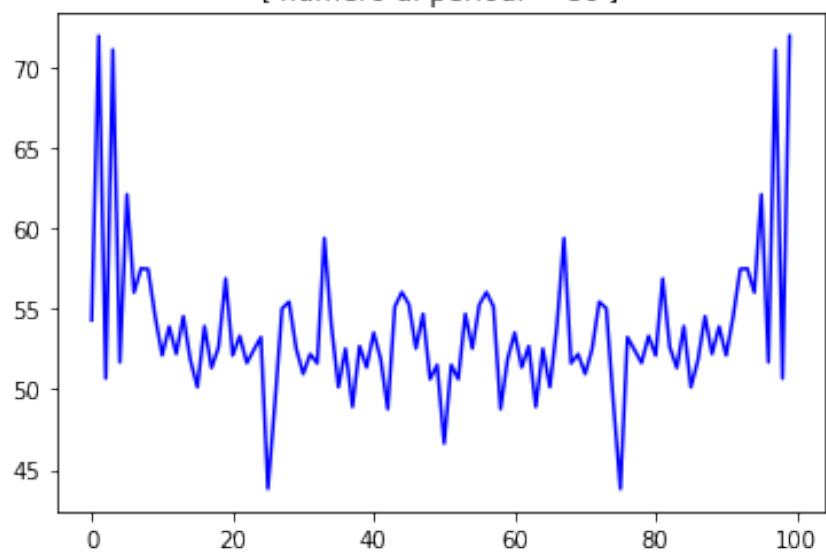
[*'numero di periodi = 60'*]



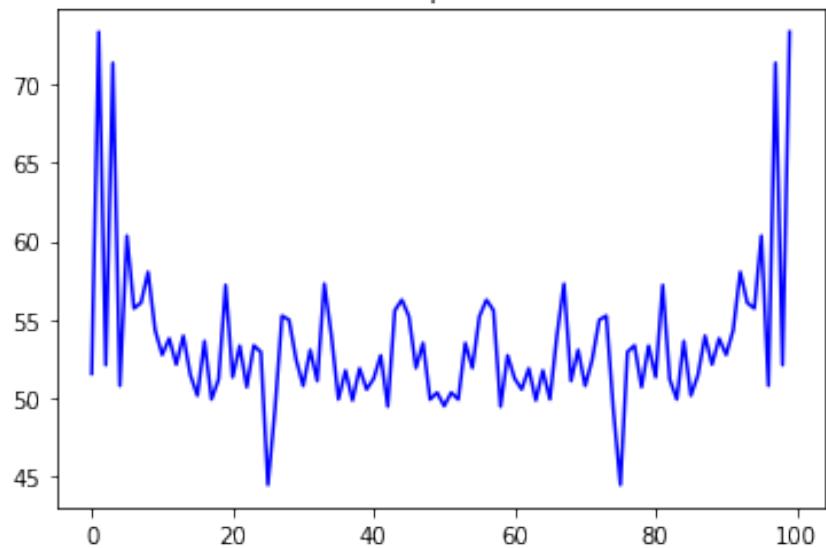
[*'numero di periodi = 70'*]



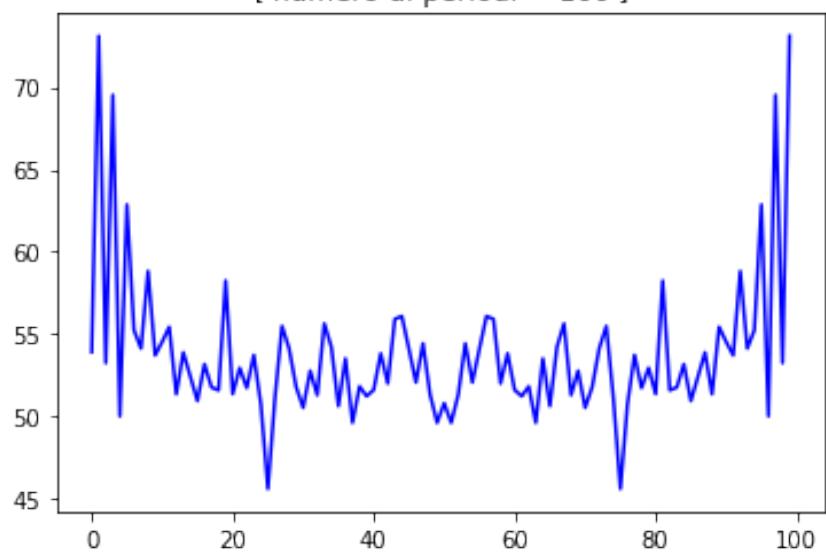
[*'numero di periodi = 80'*]



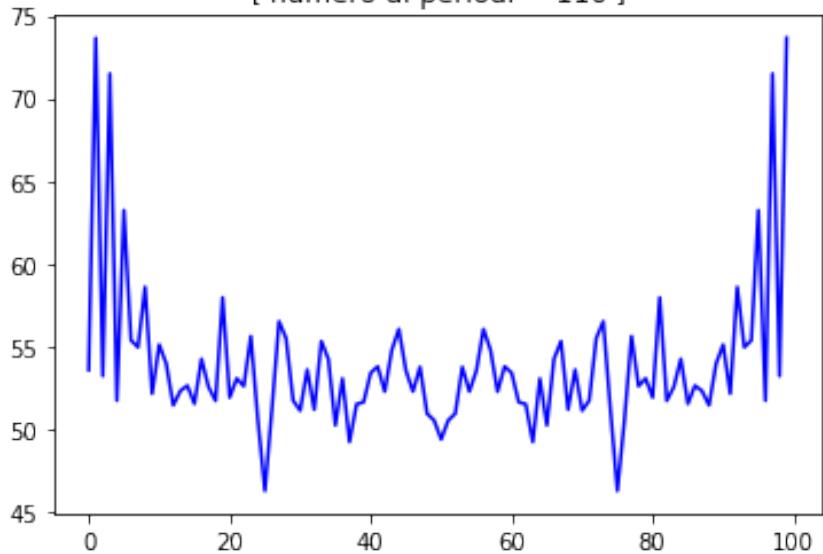
[*'numero di periodi = 90'*]



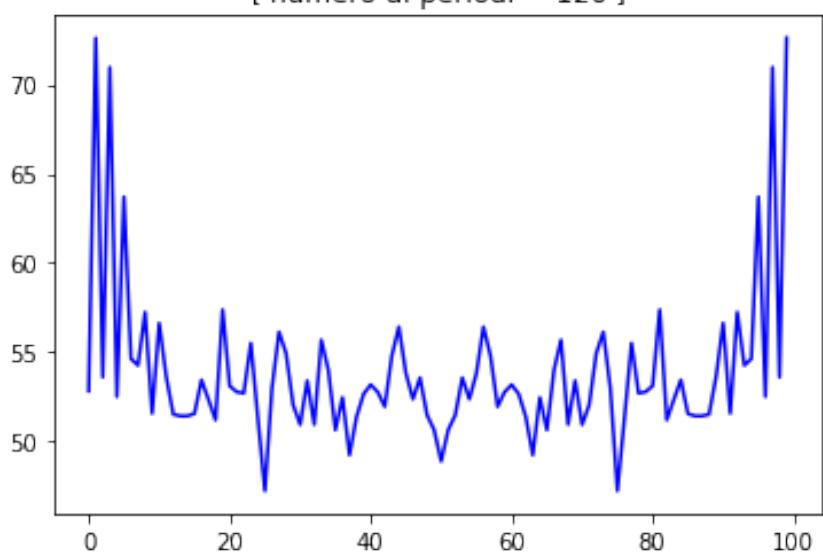
[*'numero di periodi = 100'*]



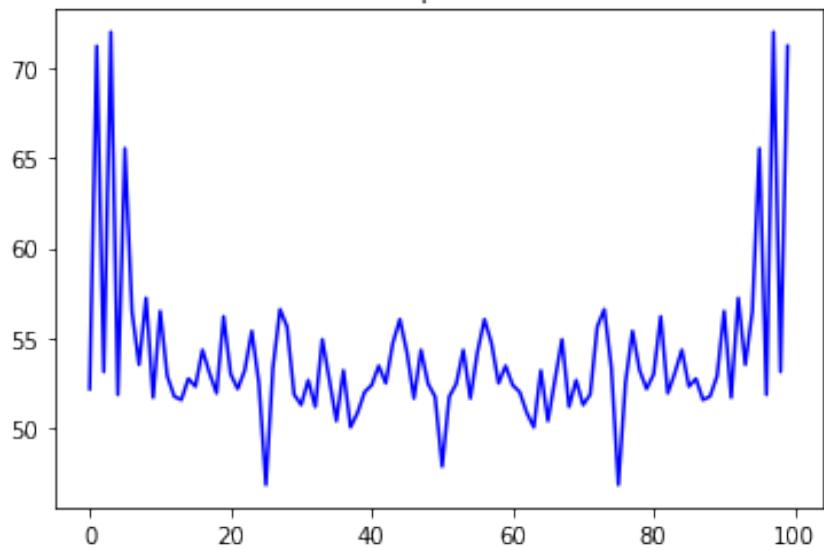
['numero di periodi = 110']



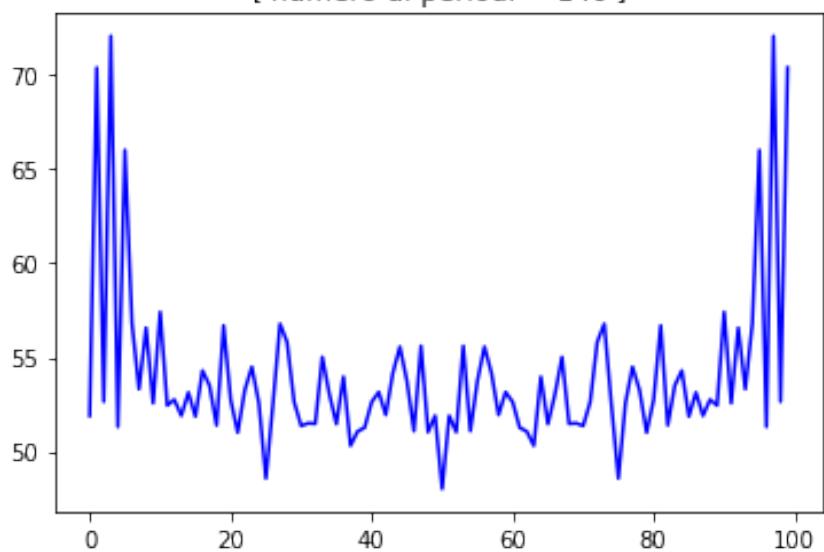
['numero di periodi = 120']



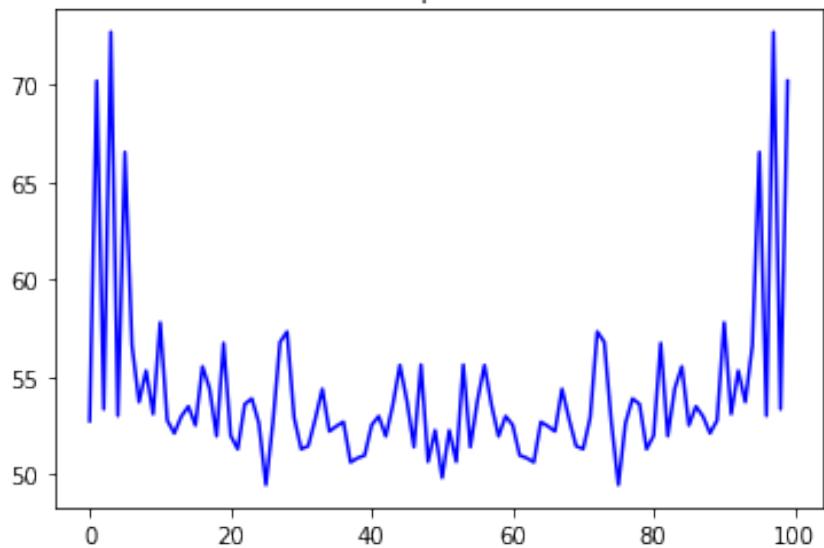
['numero di periodi = 130']



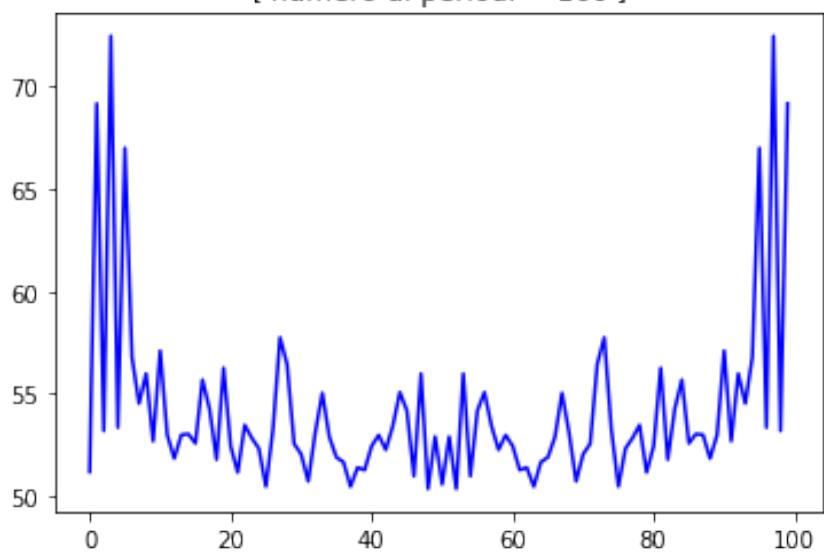
['numero di periodi = 140']



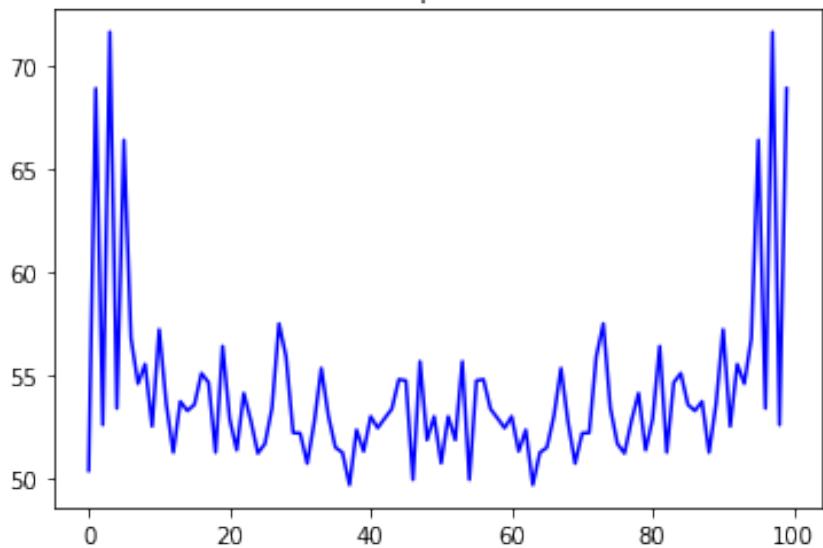
['numero di periodi = 150']



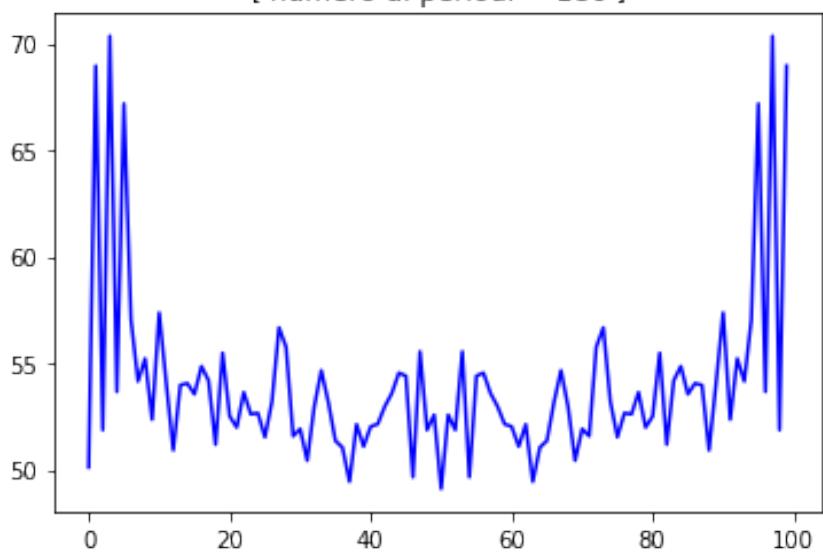
['numero di periodi = 160']



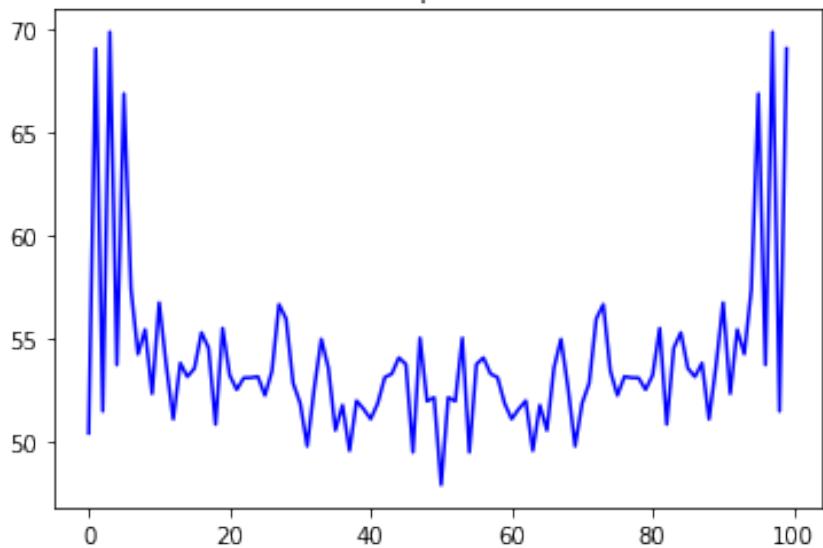
['numero di periodi = 170']



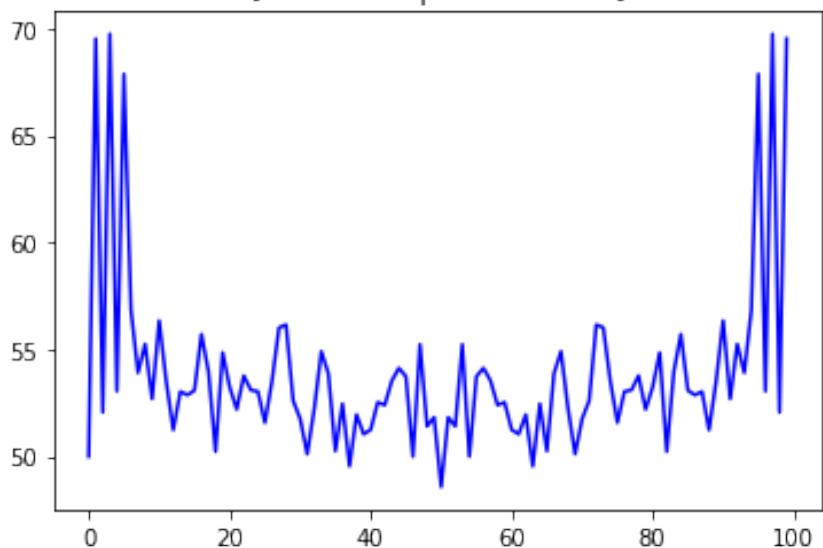
['numero di periodi = 180']



['numero di periodi = 190']

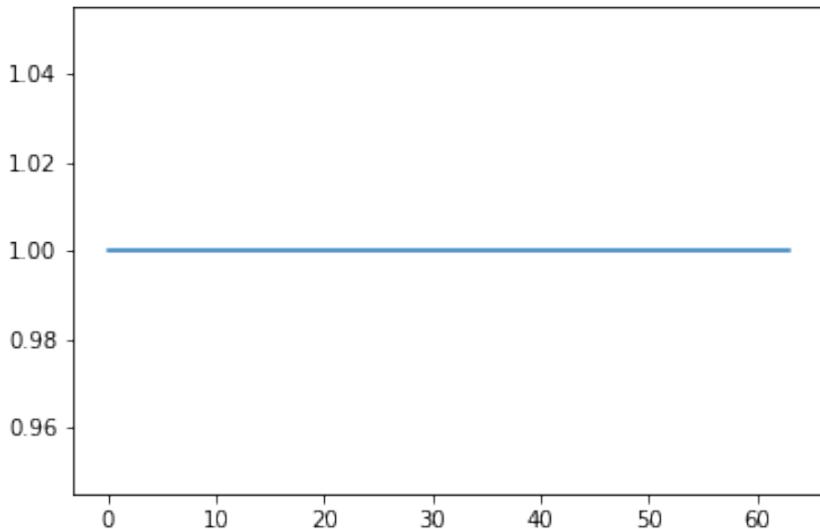


['numero di periodi = 200']



Torna al par. [7.7.4](#)

```
In [17]: # DFT del delta di Kronecker:
N = 64
u = np.zeros(N); u[0]=1.0
U = fft(u)
plt.figure(300), plt.plot(np.abs(U));
```

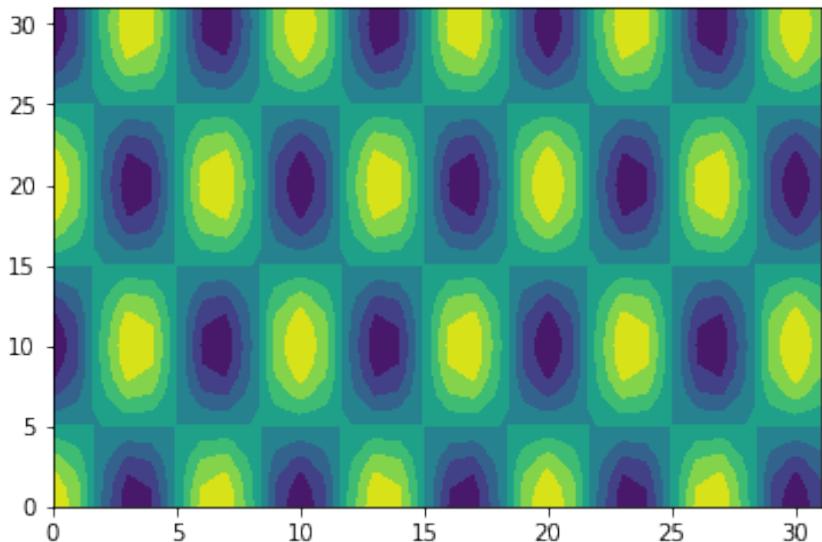


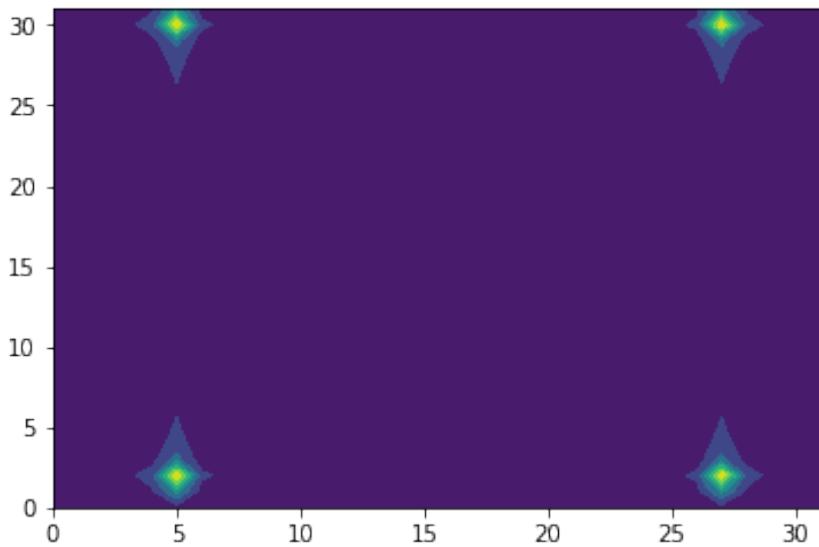
Torna al par. [7.7.2](#)

J.7 Esempio DFT 2-d

```
In [18]: #
# algoritmo FFT 2D
#
# sequenza sinusoidale bi-dimensionale
Nx = 32
Ny = 32
omega_nu_x = (2*np.pi/Nx) * 0.3*(Nx/2.)
omega_nu_y = (2*np.pi/Ny) * 0.1*(Ny/2.)
xi = np.atleast_2d(np.cos(omega_nu_y*np.arange(Ny))).T @ np.atleast_2d(np.cos(omega_nu_x*np.arange(Nx)))
plt.figure(1); plt.contourf(np.array(xi)); plt.show()
#
# calcolo della FFT 2D
#
x = np.zeros((Nx,Ny),dtype=complex)
```

```
for ix in range(Nx):
    x[ix,:] = fft_1D( xi[ix,:])
#endfor
for iy in range(Ny):
    x[:,iy] = fft_1D( np.squeeze(x[:,iy]) )
#endfor
plt.figure(2); plt.contourf(np.array(np.abs(x))); plt.show()
print('verifica: ' + str(np.sum(np.sum(np.abs(np.real(x - fft2(xi)))))) + np.sum(np
```





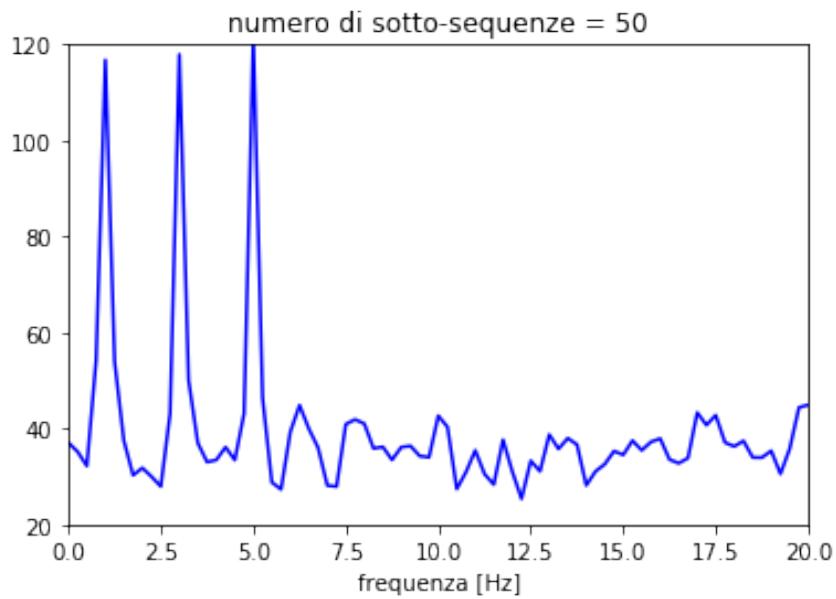
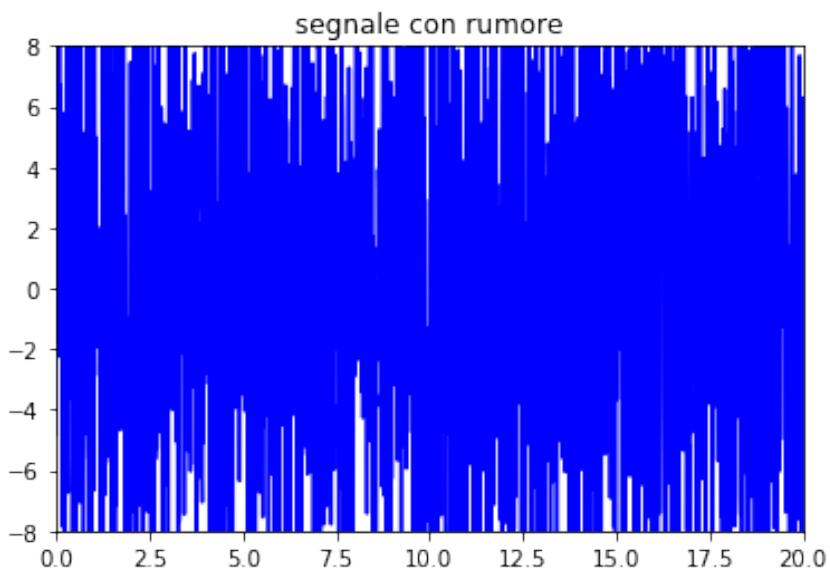
verifica: 4.905187367398867e-12

Torna al par. [7.9](#)

J.8 Esempi stima spettro di potenza

In [19]: $T_c = 0.01$

```
nper = 4 # numero di periodi considerati in ciascuna sotto-seguenza
N = int(1/Tc * nper)
nsseq = 50 # numero di sotto-seguenze
t = np.arange(0.,nper*nsseq,Tc)
si = np.sin(2*np.pi*t) + np.sin(2*np.pi*3*t) + np.sin(2*np.pi*5*t) + 6.*np.random.
plt.figure(1); plt.plot(t,si,'b-'); plt.axis([0., (nper*nsseq)/10., -8., 8]), plt.
[I,f] = psd_Welch(si,Tc,nsseq)
ifn = int(N/2)
plt.figure(2); plt.plot(f[0:ifn], I[0:ifn], 'b-'); plt.xlabel('frequenza [Hz]'), p
#
# NB: confrontare la bonta' di questo risultato, che considera la sequenza come se
# dunque ne valuta lo spettro di potenza, rispetto a quello ottenuto in precedenza
# la sequenza come segnale deterministico e si calcolava semplicemente la media de
#
print("Notare che il valor medio della psd e' pari alla varianza del rumore bianco")
```



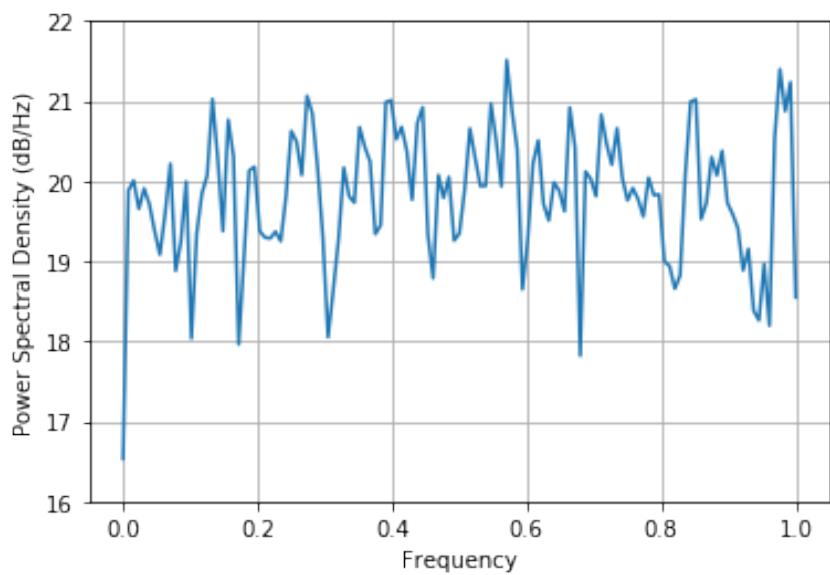
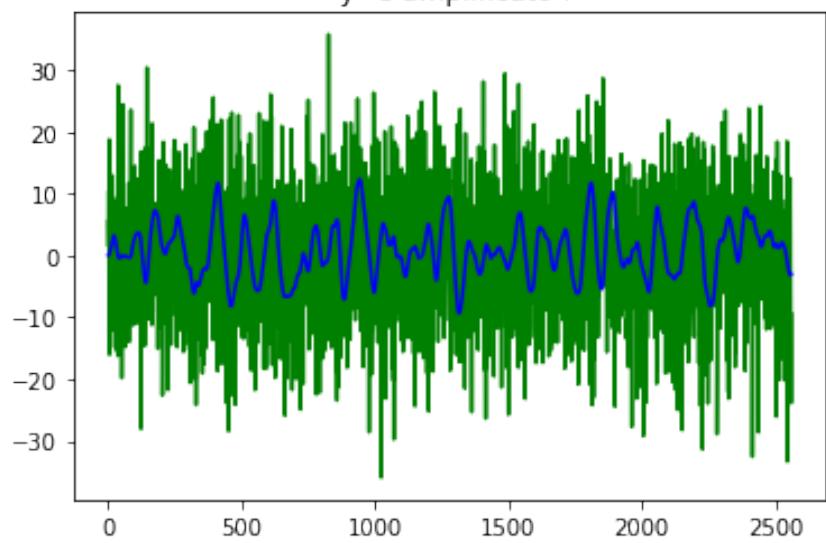
Notare che il valor medio della psd e' pari alla varianza del rumore bianco aggiunto.

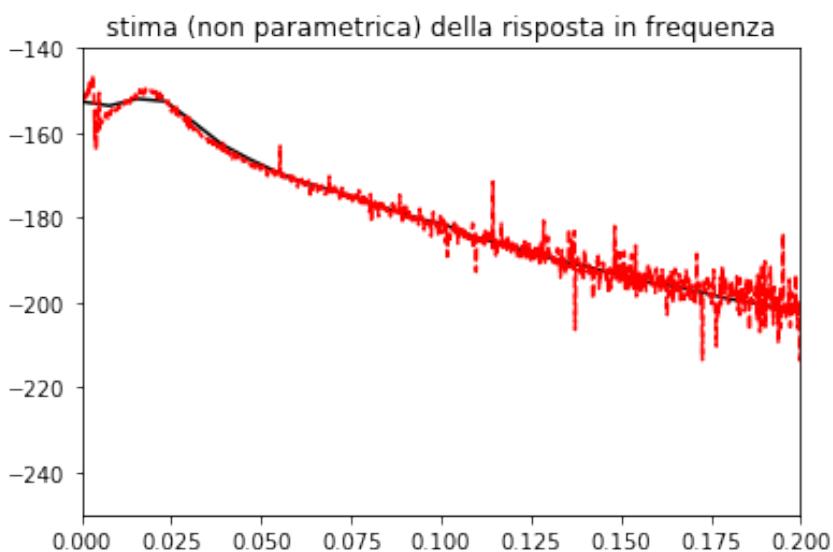
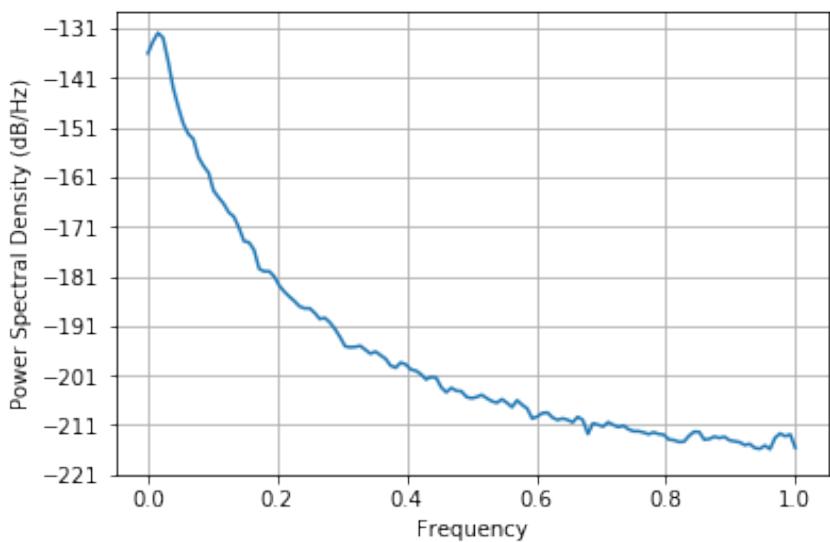
```
In [20]: #
# Verifica sperimentale della relazione di Parseval, e cioè che l'integrale (somma
# spettro di potenza è una misura della potenza media del segnale:
#
print('valore quadratico medio del segnale = ' + str(np.sum(np.sum(si**2))/(N*nsse
print('valor medio della stima dello spettro di potenza = ' + str(sum(I)/N))
```

valore quadratico medio del segnale = 37.49484370461335
 valor medio della stima dello spettro di potenza = 37.65890785890954

```
In [21]: # Calcoliamo in vari modi la risposta in frequenza del sistema meccanico con le t
N = 1024*10 # N=1024 sono pochi per DFT, con 10240 la varianza si riduce anche al
# costruisco il modello continuo del sistema meccanico:
A,B,C,D = build_sistema_mecchanico_3gdl()
# definisco il periodo di campionamento:
Ts = 0.0001;
# calcolo analitico della risposta in frequenza:
# TODO
#
# Stima della risposta in frequenza da una sequenza di dati "sperimentali":
#
# risposta ad un ingresso aleatorio a largo spettro, es. un rumore bianco:
u = 1.e1*np.random.randn(N)
y, X_hist = simula_sistema_mecchanico_3gdl(A, B, C, D, u, Ts)
ifr = int(N/4)
vk = range(ifr)
plt.figure(10); plt.plot(vk,u[0:ifr], 'g-', vk,np.squeeze(np.array(y[0,0:ifr]))*1.e
U = fft(u)
Y = fft(np.squeeze(np.array(y)))
H = Y / U
plt.figure(12); Puu, f = psd(u);
plt.figure(13); Pyy, f = psd(np.squeeze(y));
plt.figure(14); plt.plot(f, 20*np.log10( np.sqrt(Pyy / Puu) ), 'k-'); plt.plot(np.
```

"y" è amplificato !

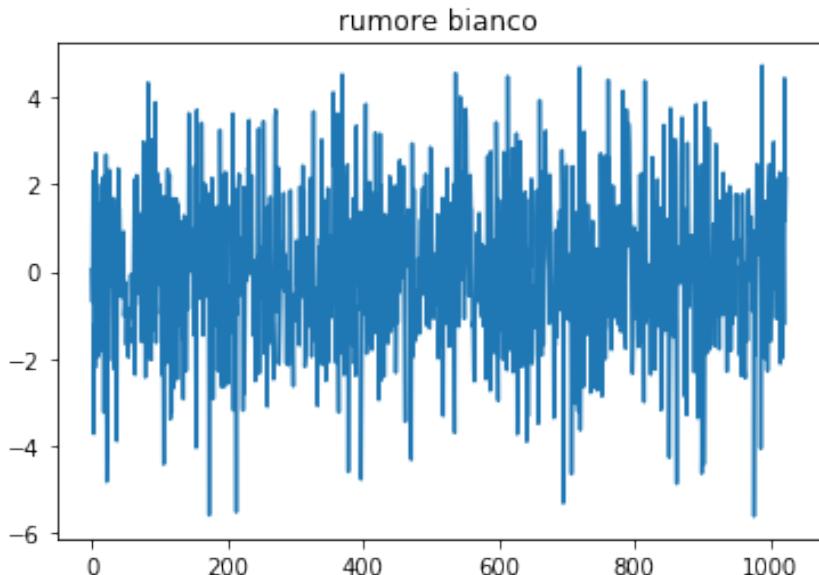




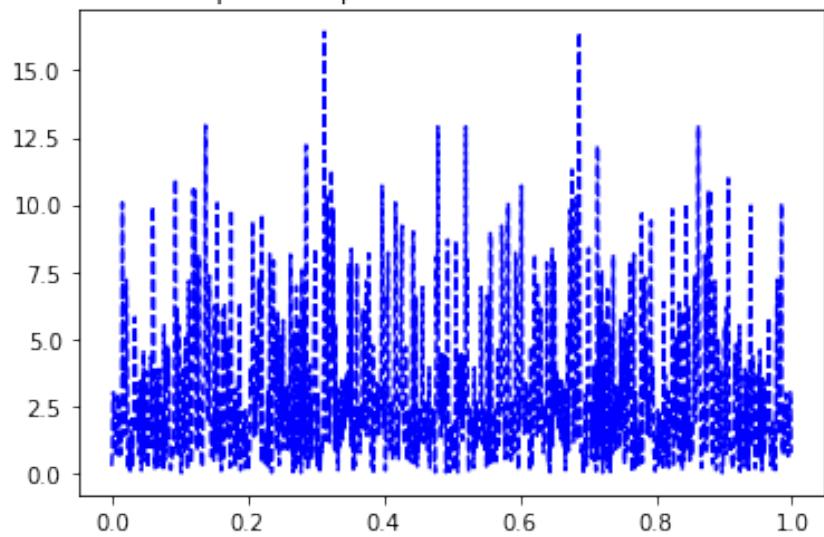
Torna al par. [7.8](#)

J.9 Esempi test di Bartlett

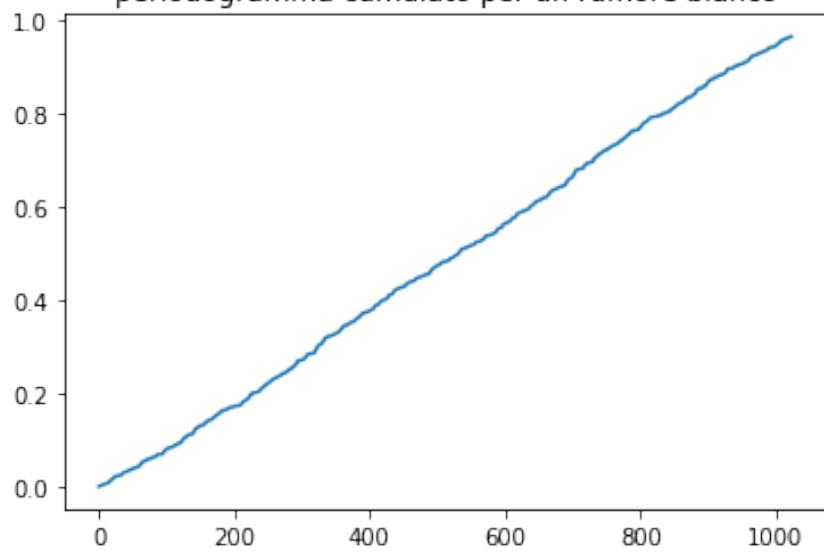
```
In [22]: #
    # test di Bartlett (o del periodogramma cumulato): serve per verificare se una seq
    #
N = 1024
var_s = 3.0
s = np.sqrt(var_s)*np.random.randn(N)
plt.figure(1); plt.plot(s); plt.title('rumore bianco'); plt.show()
Pss, f = psd Welch(s);
plt.figure(2); plt.plot(f, Pss, 'b--'); plt.title('spettro di potenza del rumore b
CPss_wn,f = testBartlett_wn(s)
plt.figure(4); plt.plot(CPss_wn); plt.title('periodogramma cumulato per un rumore
s = np.sqrt(var_s)*np.random.randn(N) + np.sin(2.*np.pi/(N/15)*np.arange(N))
plt.figure(5); plt.plot(s); plt.title('rumore bianco piu'' componente sinusoidale'
Pss, f = psd Welch(s);
plt.figure(2); plt.plot(f, Pss, 'b--'); plt.title('spettro di potenza del rumore b
CPss,f = testBartlett_wn(s);
plt.figure(6); plt.plot(CPss_wn, 'r--'); plt.plot(CPss); plt.title('periodogramma
plt.show()
```



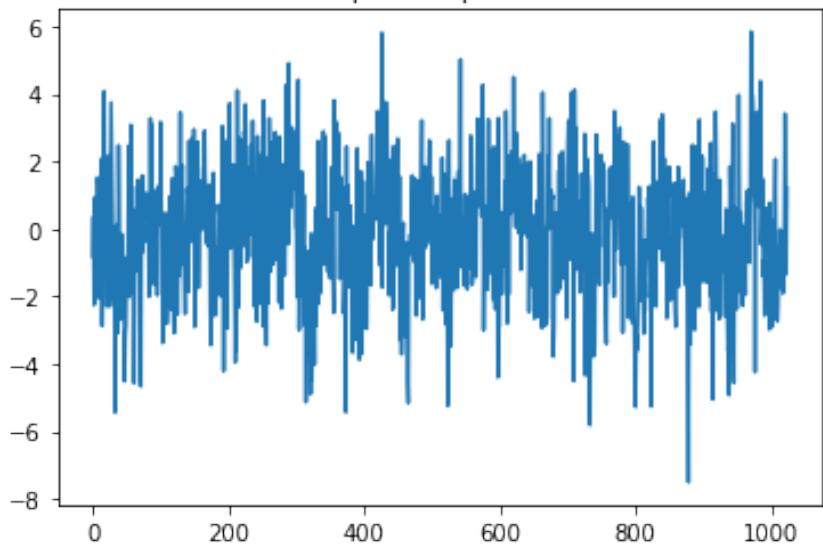
spettro di potenza del rumore bianco



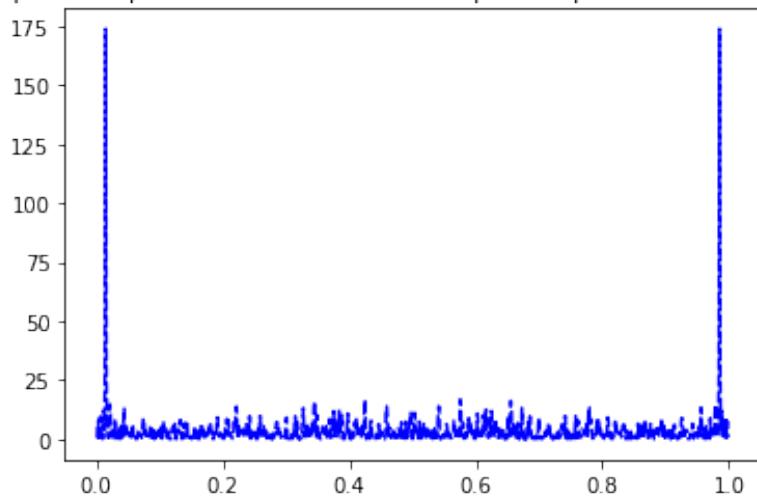
periodogramma cumulato per un rumore bianco



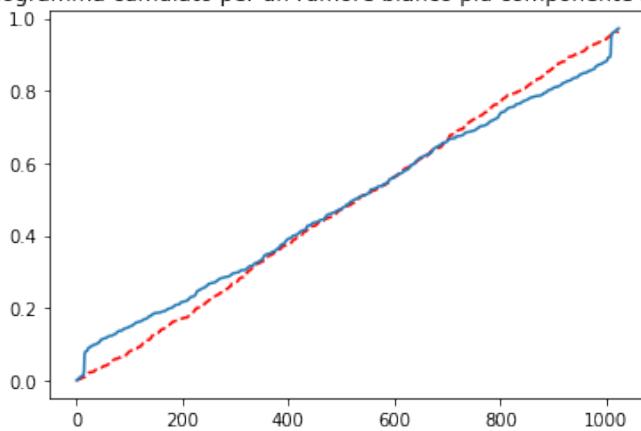
rumore bianco piu componente sinusoidale



spettro di potenza del rumore bianco piu componente sinusoidale



periodogramma cumulato per un rumore bianco piu componente sinusoidale

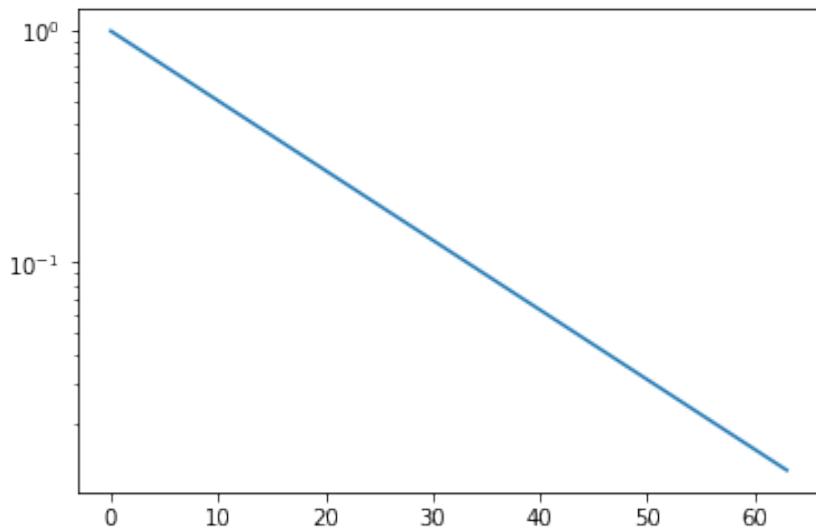


Torna al par. 7.8.1

J.10 Esempio di filtraggio vs regolarizzazione

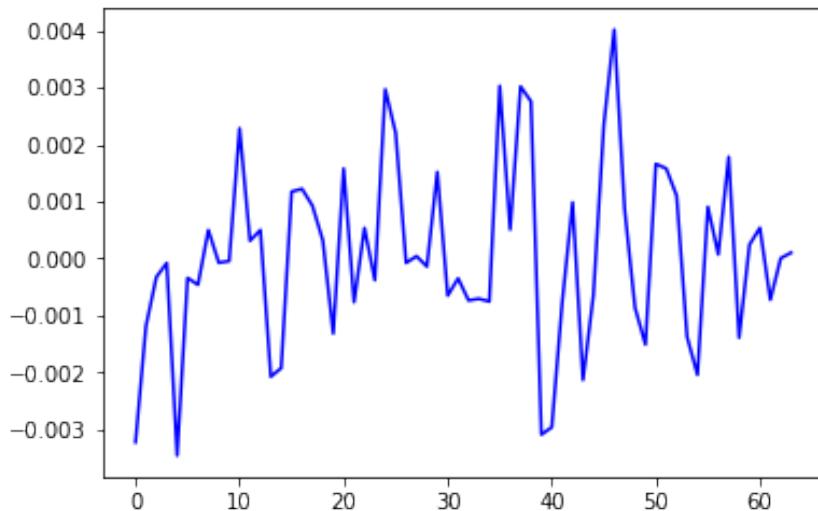
In [23]: $n = 64$

```
S = np.diag(2.**((0.1*np.arange(0,-n,-1)))
U,R = np.linalg.qr(np.random.randn(n,n))
V,R = np.linalg.qr(np.random.randn(n,n))
plt.figure(1); plt.semilogy(np.diag(S)); plt.show()
A = U@S@V.T;
K2_A = np.linalg.cond(A,2);
print("numero di condizionamento di A = ", K2_A)
```



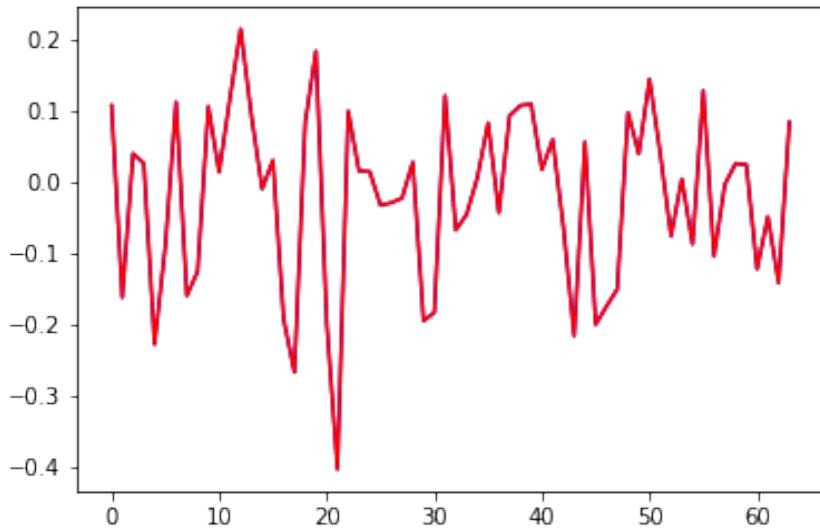
numero di condizionamento di $A = 78.79324245407467$

```
In [24]: # prendo come soluzione vera un vettore che sta nella componente principale relativa  
x_vero = V[:,n-1]  
x_vero = np.atleast_2d(x_vero).T  
b = A @ x_vero;  
plt.figure(2); plt.plot(b,'b-'); plt.show()
```



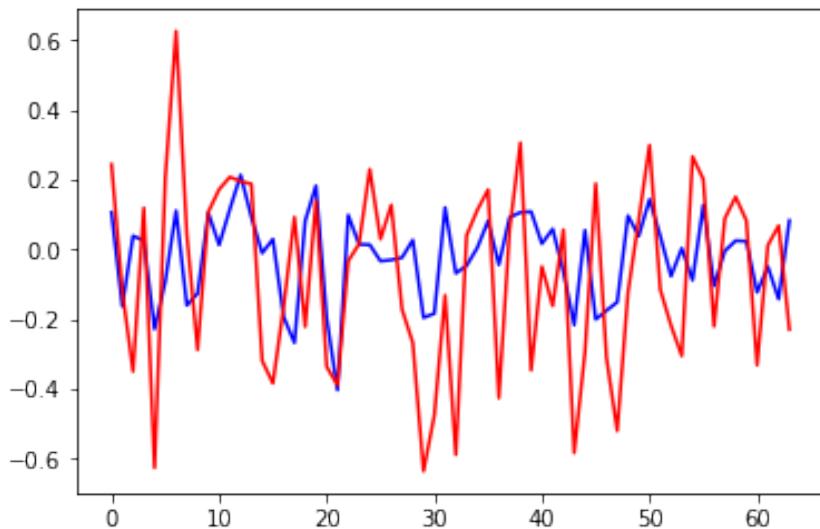
```
In [25]: x_LU = np.linalg.solve(A, b)
        print("errore relativo LU = ", np.linalg.norm(x_LU - x_vero, 2)/np.linalg.norm(x_v
plt.figure(3); plt.plot(x_vero,'b-'); plt.plot(x_LU,'r'); plt.show()
```

```
errore relativo LU =  3.1350897704723374e-15
```



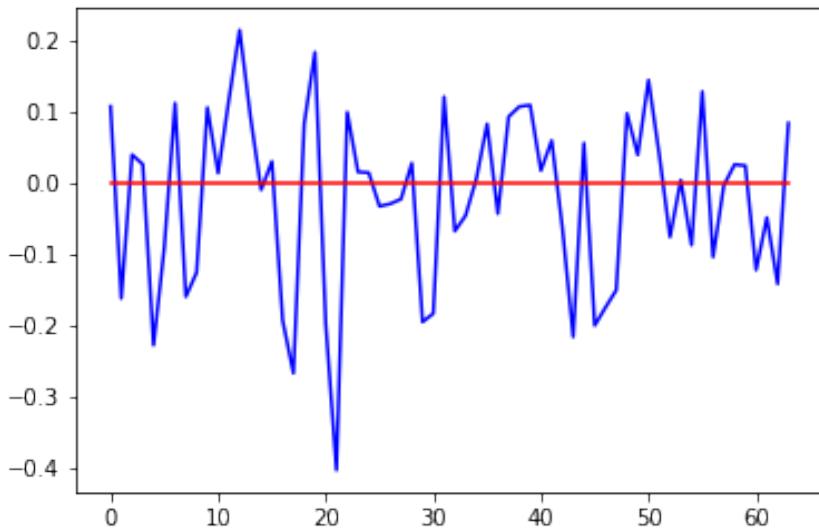
```
In [26]: print("puo' bastare pero' una perturbazione di b perche' il risultato x_LU venga corrotto")
nu = 28
delta_b = 0.01*np.sin(2*np.pi*nu/n*np.arange(n))
delta_b = np.atleast_2d(delta_b).T
br = b + delta_b
x_LU = np.linalg.solve(A, br)
print("errore relativo LU = ", np.linalg.norm(x_LU - x_vero, 2)/np.linalg.norm(x_v
plt.figure(4); plt.plot(x_vero,'b-'); plt.plot(x_LU,'r');
```

```
puo' bastare pero' una perturbazione di b perche' il risultato x_LU venga corrotto
errore relativo LU =  1.9373648214048096
```

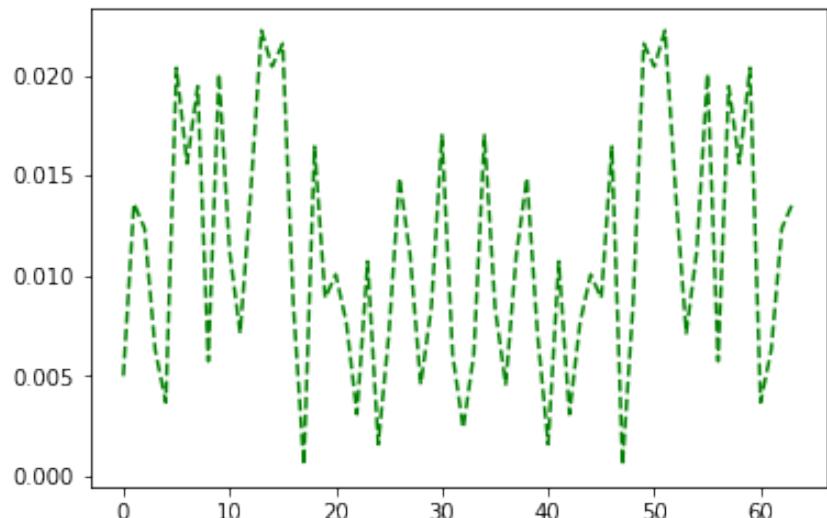
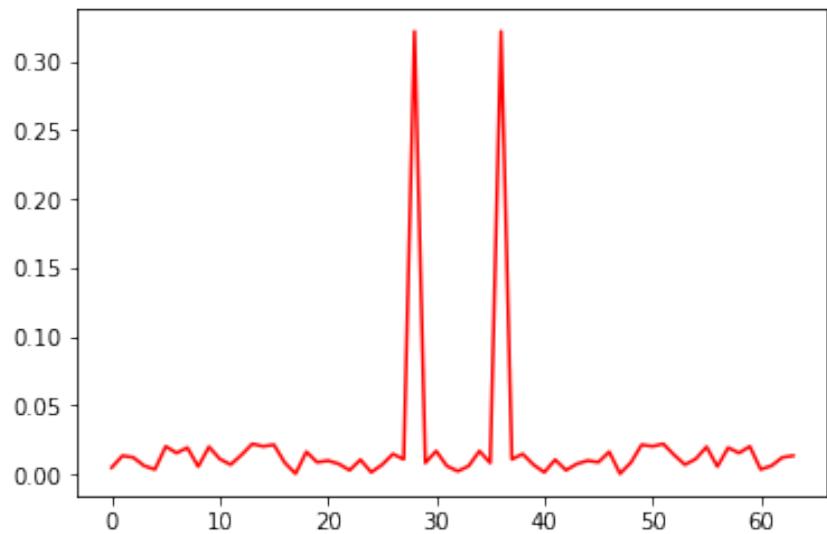


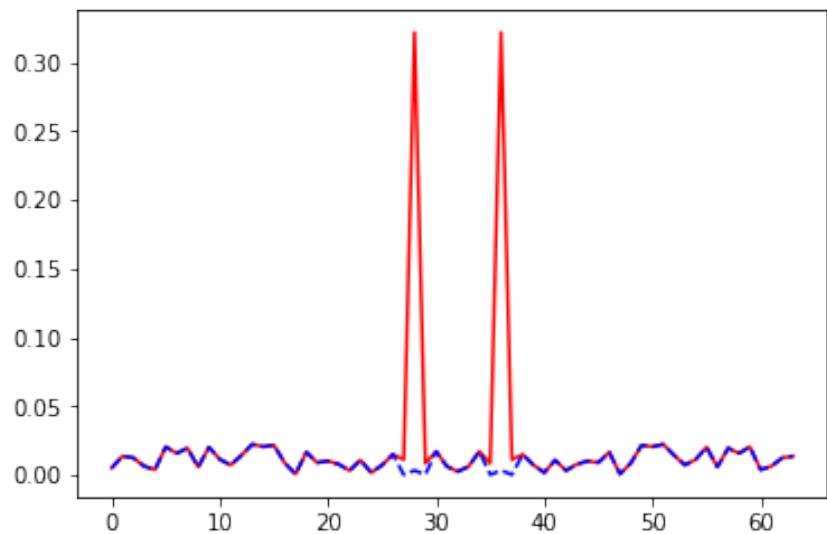
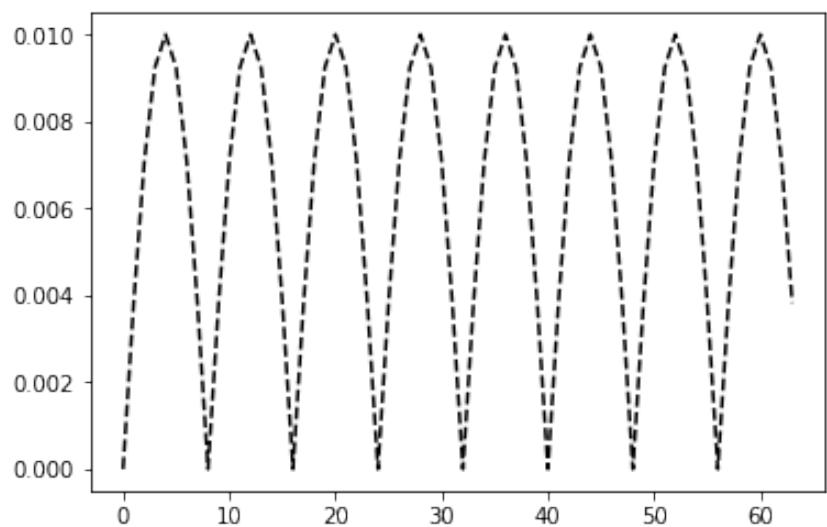
```
In [27]: k = n-2
Sinv = np.diag(1./np.diag(S[0:k+1,0:k+1]))
x_TSVD = V[:,0:k+1] @ Sinv @ U[:,0:k+1].T @ b
plt.figure(5); plt.plot(x_vero, 'b-'); plt.plot(x_TSVD, 'r')
print("errore relativo = ", np.linalg.norm(x_TSVD - x_vero, 2)/np.linalg.norm(x_vero))
print("come si puo' vedere, e' stata tagliata la parte utile del modello ... in quanto")
```

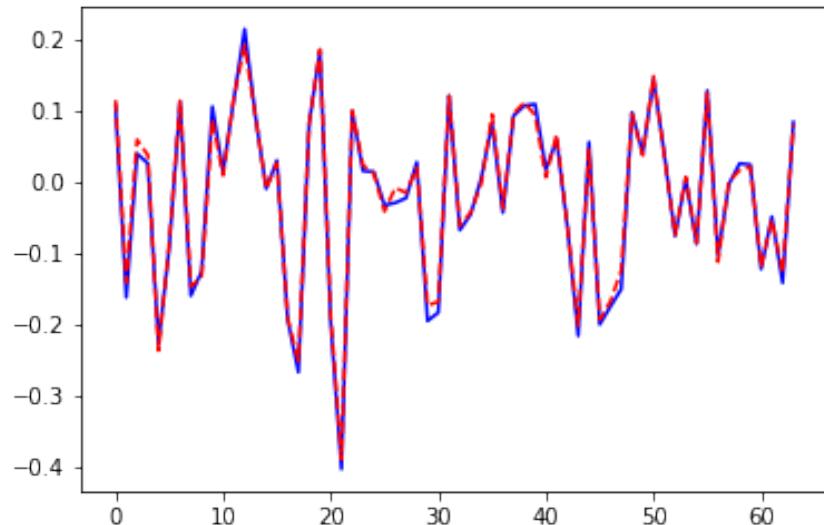
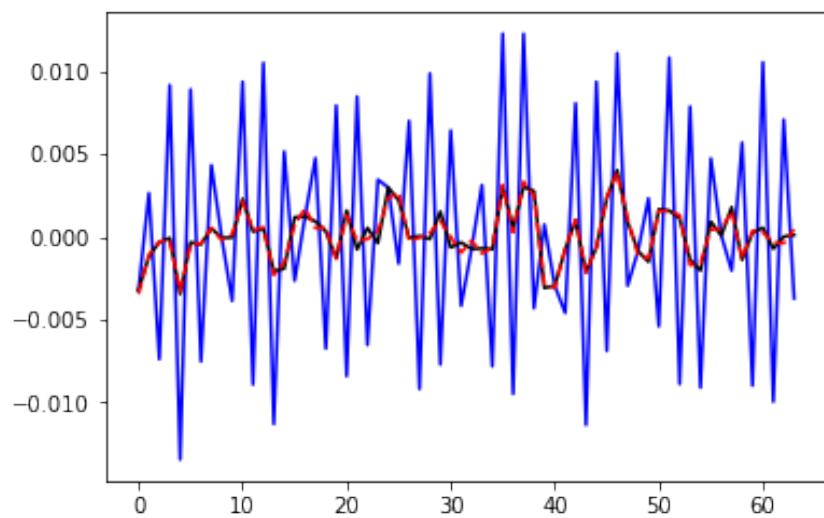
```
errore relativo =  1.0
come si puo' vedere, e' stata tagliata la parte utile del modello ... in questo caso
```



```
In [28]: # regolarizzo mediante filtraggio:
Fb_mis = fft(np.squeeze(br))
plt.figure(6); plt.plot(np.abs(Fb_mis), 'r-'); plt.show()
plt.figure(7); plt.plot(np.abs(fft(np.squeeze(b))), 'g--'); plt.show()
Fdelta_b = fft(delta_b)
plt.figure(8); plt.plot(abs(Fdelta_b), 'k--'); plt.show()
nu_approx = nu
# creo il filtro:
c_atten = 1.e-2
Ffiltro = np.ones(n)
Ffiltro[nu_approx+np.arange(-1,1+1)] = c_atten # NB: non "0", altrimenti si divide per zero
Ffiltro[n-(nu_approx+np.arange(-1,1+1))] = c_atten # NB: non "0", altrimenti si divide per zero
#
Fb_filtrato = Fb_mis * Ffiltro
plt.figure(9); plt.plot(np.abs(Fb_mis), 'r-'); plt.plot(np.abs(Fb_filtrato), 'b--');
b_filtrato = np.real(ifft(Fb_filtrato))
plt.figure(10); plt.plot(br, 'b-'); plt.plot(b, 'k-'); plt.plot(b_filtrato, 'r--'); plt.show()
#
x_LU = np.linalg.solve(A, np.asmatrix(b_filtrato).T)
plt.figure(12); plt.plot(x_vero, 'b-'); plt.plot(x_LU, 'r--'); plt.show()
print('norm(x_vero-x_LU)/norm(x_vero) = ' + str(np.linalg.norm(x_vero-x_LU,2)/np.linalg.norm(x_vero,2)))
```





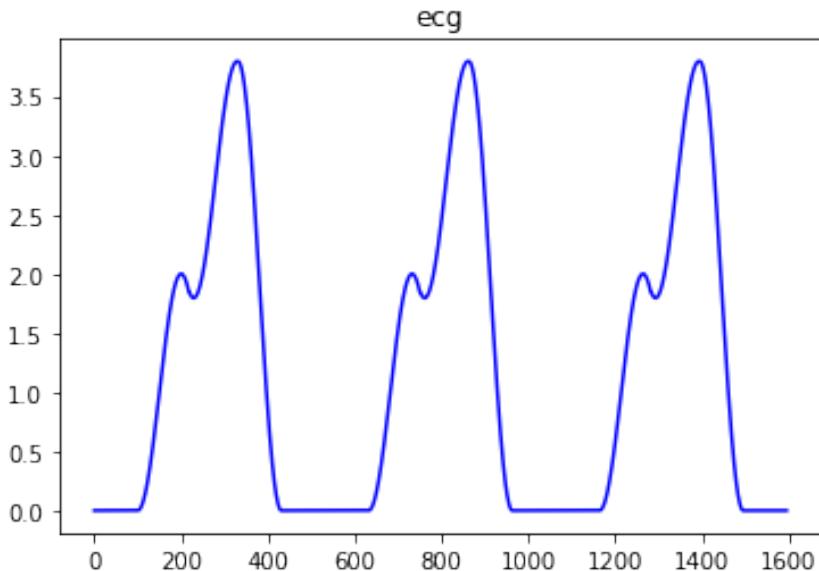


```
norm(x_vero-x_LU)/norm(x_vero) = 0.08030466193562008
```

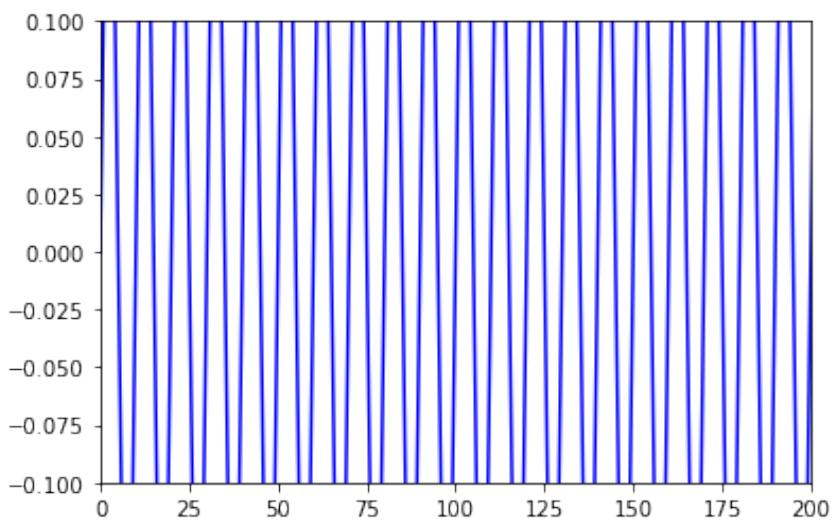
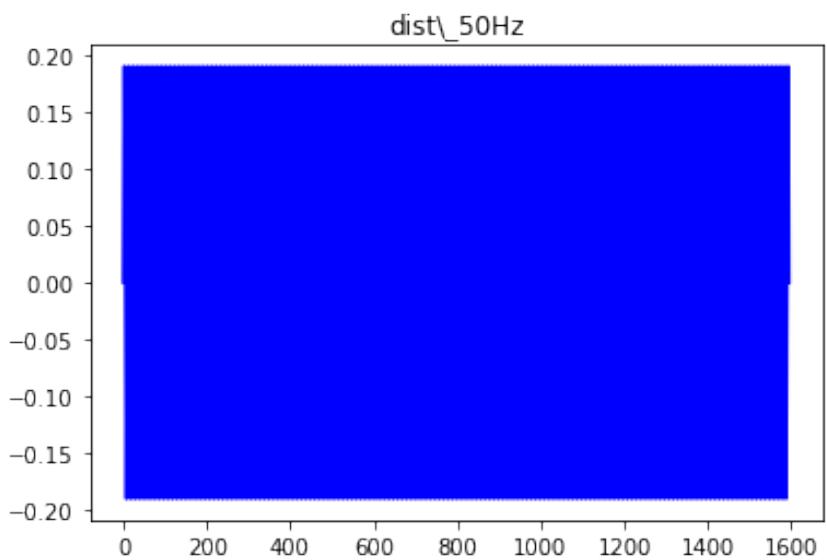
Torna al par. [7.4.7](#)

J.11 Esempio ECG

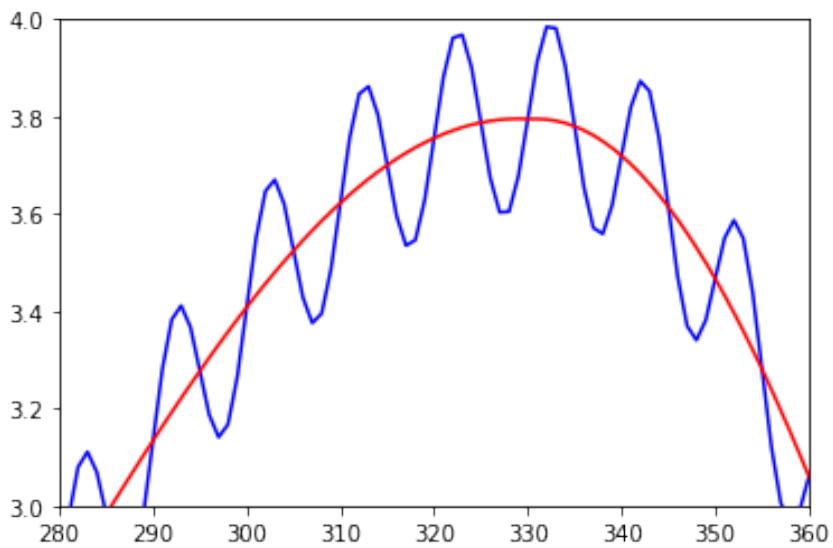
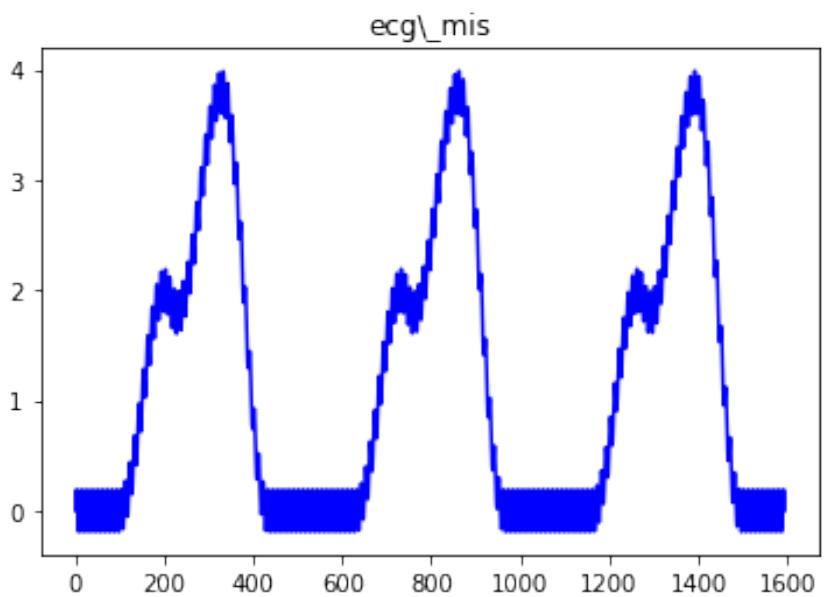
```
In [29]: # ecg campionato a 500Hz:  
fc = 500.  
t2 = np.sin(np.pi*np.arange(-0.5,0.68,0.01)) + np.abs(np.sin(np.pi*-0.5))  
t3 = (t2[len(t2)-1]-np.sin(np.pi*-0.6)) + np.sin(np.pi*(-0.5+np.arange(-0.1,1.01,0.01)))  
t4 = t3[len(t3)-1]/2.*np.sin(np.pi*np.arange(0.5,1.51,0.01)) + t3[len(t3)-1]/2  
ecg = np.tile(np.concatenate((np.zeros(100), t2, t3, t4, np.zeros(100))),3)  
plt.figure(1); plt.plot(ecg, 'b-'), plt.title('ecg'); plt.show()  
N= len(ecg);
```



```
In [30]: # rumore di misura a 50Hz (dovuto al circuito di alimentazione elettrica):  
fr = 50. # [Hz]  
dist_50Hz = 0.2 * np.sin(2*np.pi*(fr/fc)*np.arange(0,N))  
plt.figure(2); plt.plot(dist_50Hz, 'b-'), plt.title('dist\50Hz'); plt.show()  
plt.figure(3); plt.plot(dist_50Hz, 'b-'), plt.axis([0, 200, -0.1, 0.1]); plt.show()
```



```
In [31]: # segnale misurato:  
ecg_mis = ecg + dist_50Hz;  
plt.figure(4); plt.plot(ecg_mis, 'b-'); plt.title('ecg\mis'); plt.show()  
plt.figure(5); plt.plot(ecg_mis, 'b-'); plt.plot(ecg, 'r-'), plt.axis([280, 360, 3.0])
```

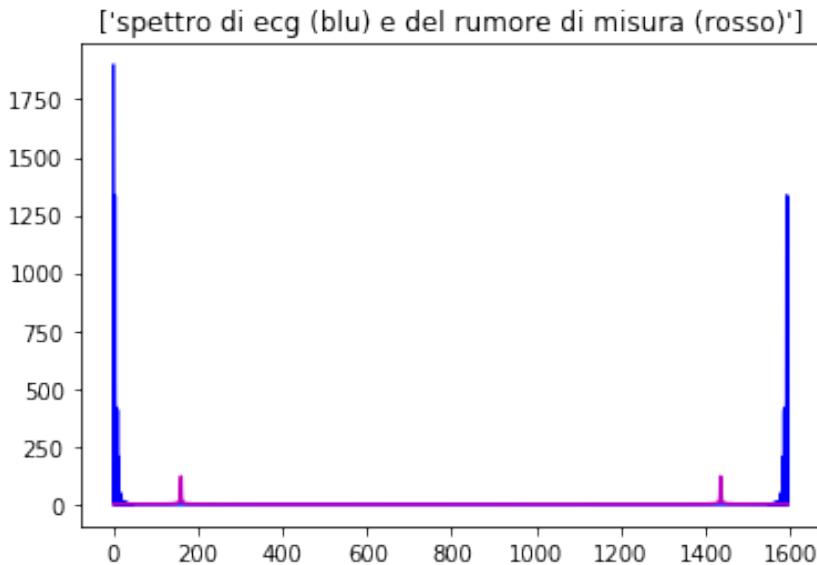


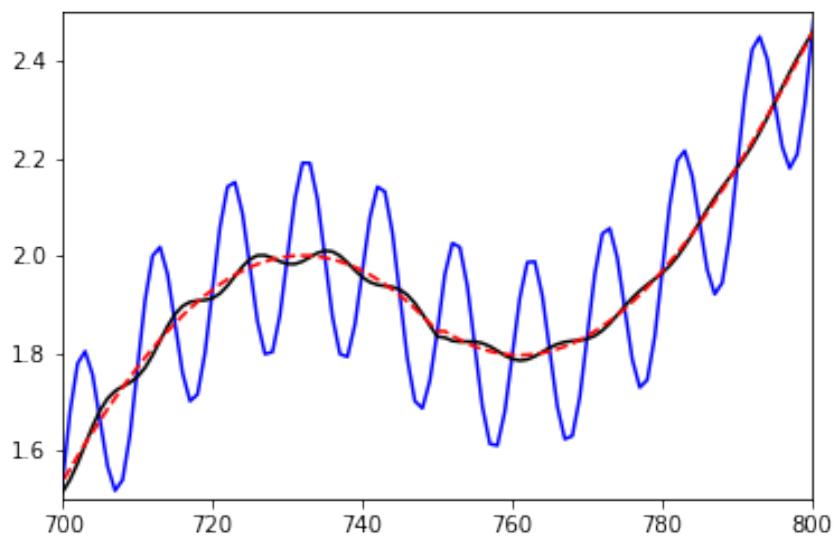
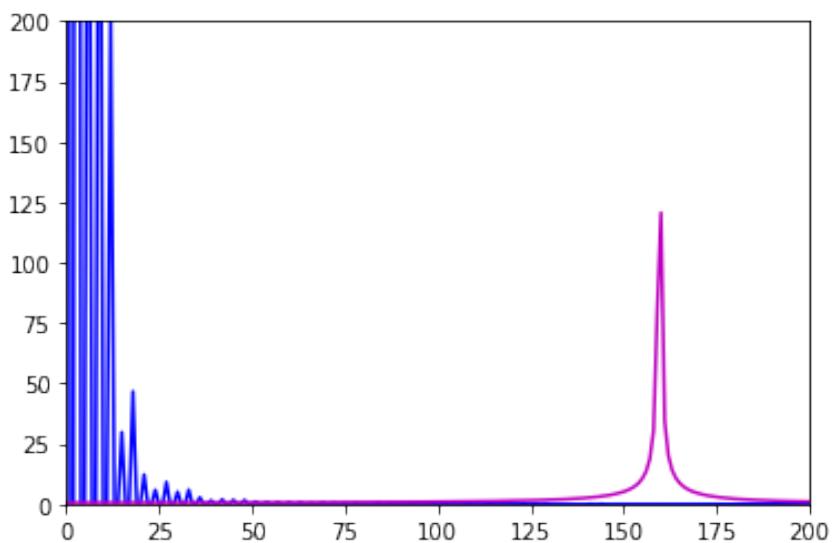
```
In [32]: # analisi di Fourier:  
Fecg = fft(ecg)
```

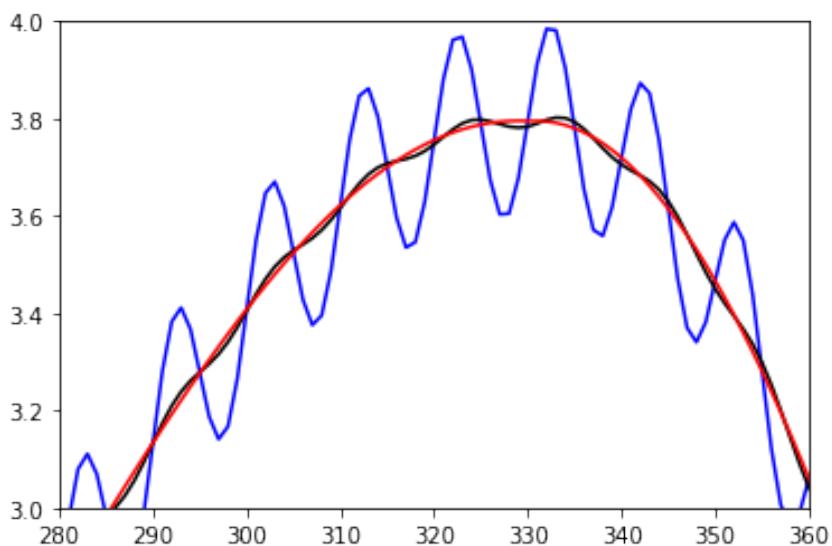
```

Fdist_50Hz = fft(dist_50Hz)
plt.figure(6); plt.plot(abs(Fecg), 'b-'); plt.plot(abs(Fdist_50Hz), 'm-'); plt.title('spettro di ecg (blu) e del rumore di misura (rosso)')
plt.figure(7); plt.plot(abs(Fecg), 'b-'); plt.plot(abs(Fdist_50Hz), 'm-'); plt.axis([0, 1600, 0, 1800])
# dalla figura si vede che lo spettro del segnale utile e del rumore di
# misura hanno bande disgiunte, quindi posso separarli facilmente mediante
# filtraggio:
Fecg_mis = fft(ecg_mis)
nu_approx_ecg = int(round(fr/fc*N))
# creo il filtro:
c_atten = 1.e-2
Ffiltro = np.ones(N)
fmask = np.array(range(-1,2))
Ffiltro[nu_approx_ecg+fmask] = c_atten    # NB: non "0", altrimenti si divide per zero
Ffiltro[N-(nu_approx_ecg+fmask)] = c_atten # NB: non "0", altrimenti si divide per zero
#
Fecg_filtrato = Fecg_mis * Ffiltro
ecg_filtrato = np.real(ifft(Fecg_filtrato))
plt.figure(8); plt.plot(ecg_mis, 'b-'); plt.plot(ecg_filtrato, 'k-'); plt.plot(ecg, 'r-')
plt.figure(9); plt.plot(ecg_mis, 'b-'); plt.plot(ecg_filtrato, 'k-'); plt.plot(ecg, 'r-')

```







Torna la par. [7.4.7](#)

In [33] :

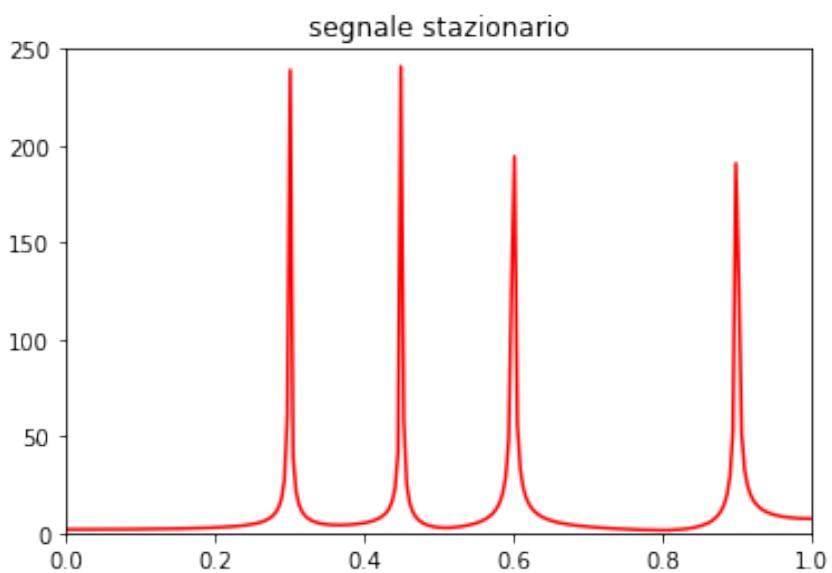
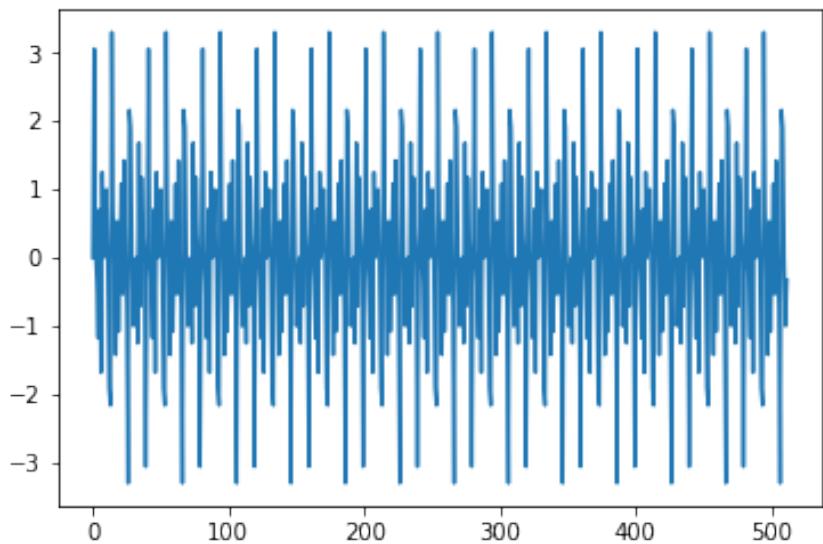
Appendice K

Esempio segnali non-stazionari

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from numpy.fft import *

In [2]: #
# comportamento della DFT per segnali non-stazionari:
#
N = 512
omega_nu = (2*np.pi/N) * 0.3*(N/2)

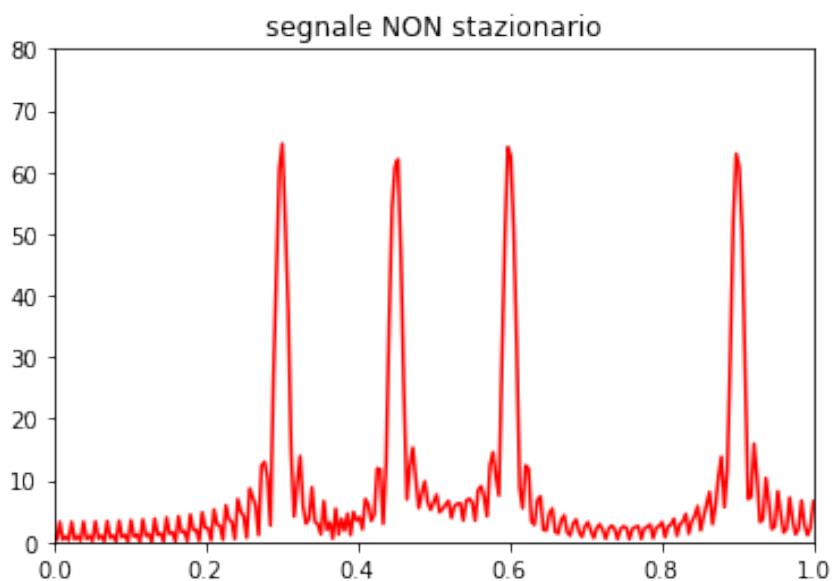
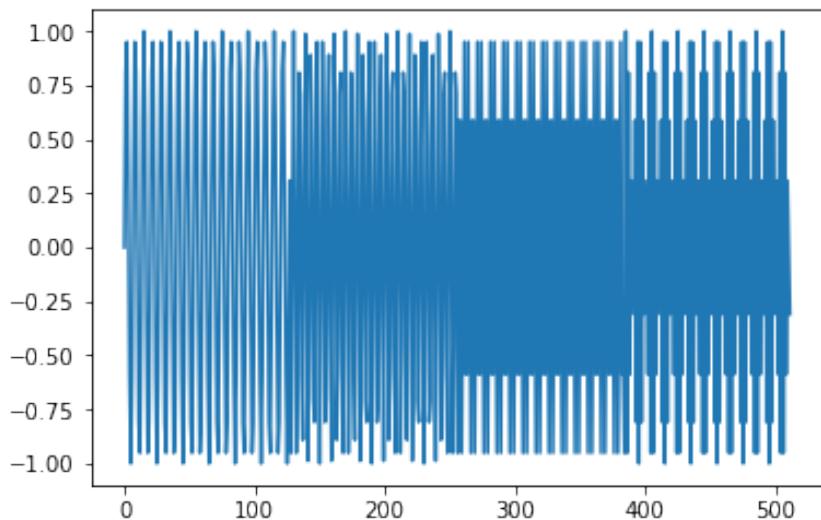
In [3]: # segnale stazionario:
u = np.sin(omega_nu * np.arange(N)) + np.sin(1.5*omega_nu * np.arange(N)) + np.sin(
U = fft(u)
plt.figure(1); plt.plot(u); plt.show()
fn = int(N/2)
plt.figure(2); plt.plot(np.arange(0.,1.+2./N,2./N), np.abs(U[0:fn+1]), 'r-'); plt.a
```



In [4]: # segnale *NON* stazionario:

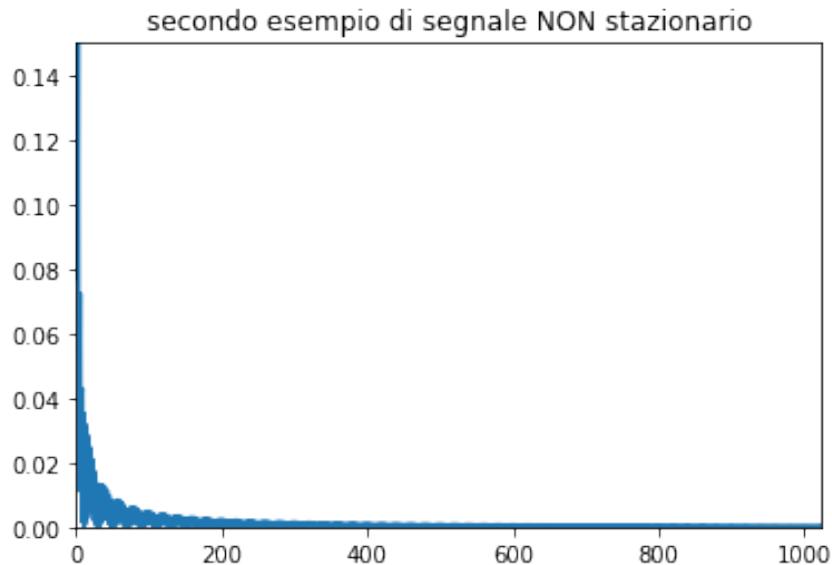
```
u = np.concatenate((np.sin(omega_nu * np.arange(N/4)), np.sin(1.5*omega_nu * np.arange(N/4)))
```

```
U = fft(u)
plt.figure(11); plt.plot(u); plt.show()
fn = int(N/2)
plt.figure(12); plt.plot(np.arange(0.,1.+2./N,2./N), np.abs(U[0:fn+1]), 'r-'); plt.
```

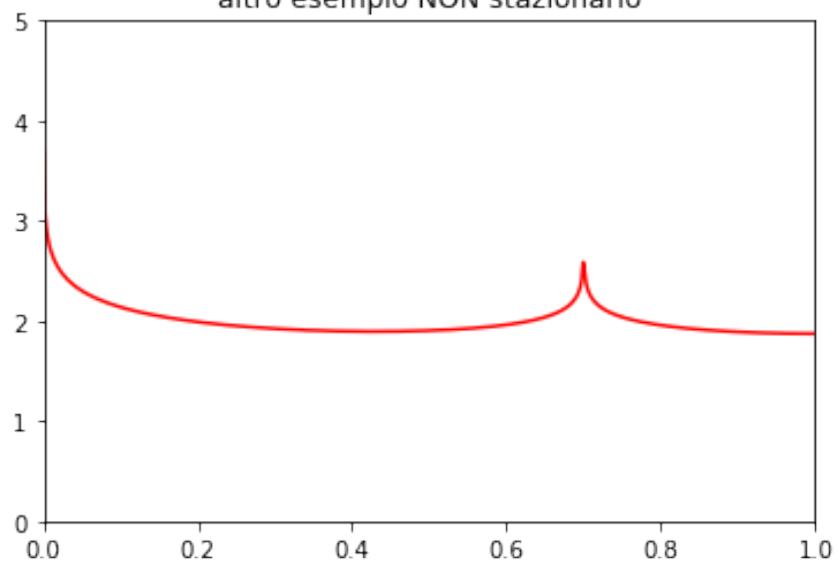


In [5]: # altro esempio:

```
N = 1024
fn = int(N/2)
omega_nu = (2*np.pi/N) * 0.7*fn;
u = (1. + np.cos(omega_nu * np.arange(N))) / (1. + 4 * np.arange(N))
U = fft(u)
plt.figure(21); plt.plot(u); plt.title('secondo esempio di segnale NON stazionario')
plt.figure(22); plt.plot(np.arange(0.,1.+2./N,2./N), np.abs(U[0:fn+1]), 'r-'); plt.
```



altro esempio NON stazionario



Torna al par. 8

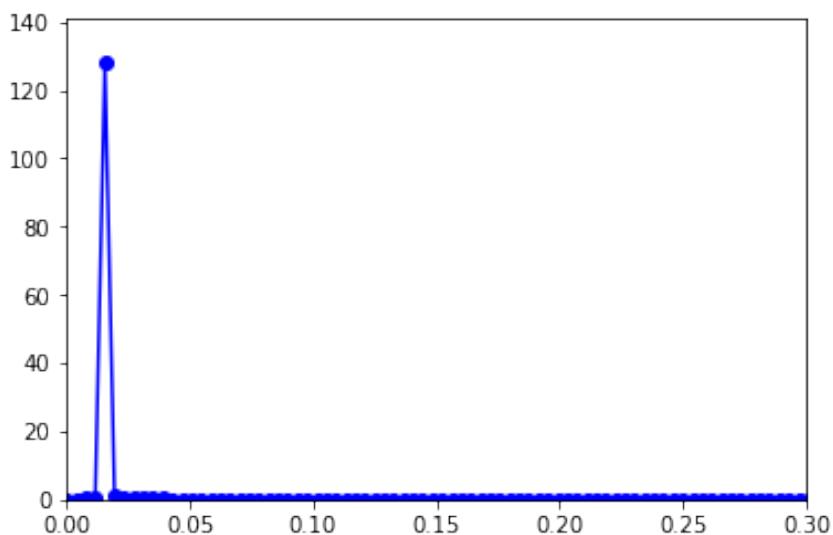
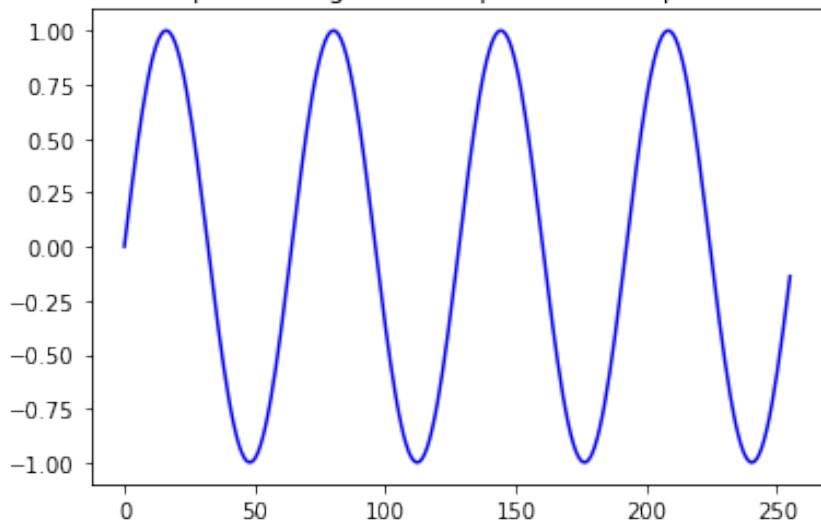
Appendice L

Aspetti pratici STFT 1-d

```
In [1]: %matplotlib inline
# NB: per eseguire questo notebook come file Python, commentare l'istruzione "%matp
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.hamming import *
from libreria_NLALD.stft import *
from numpy.fft import *
from scipy.signal import gaussian
```

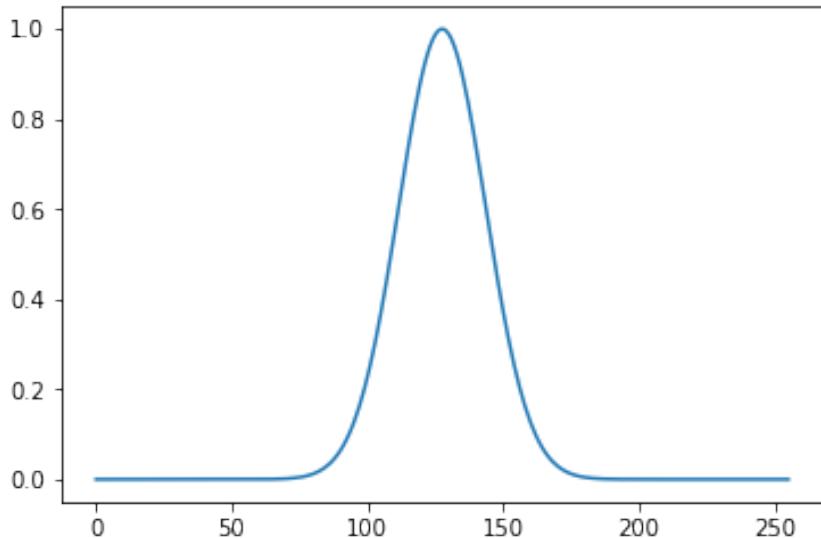
```
In [2]: #
fc = 1.
Tc = 1./fc
N = 256
t = np.arange(0.,N*Tc,Tc)
f = 0.0156
s = np.sin(2*np.pi*f*t);    # segnale sinusoidale alla frequenza "f" (Hz)
# FFT di un multiplo intero del periodo:
S = fft(s);
plt.figure(1); plt.plot(t, s, 'b-'); plt.title('sequenza lunga un multiplo intero d
vf = np.arange(0,1/Tc,1/(Tc*N))
plt.figure(2); plt.plot(vf, abs(S), 'b-o'); plt.axis([0., 0.3, 0., max(abs(S))*1.1]
```

sequenza lunga un multiplo intero del periodo

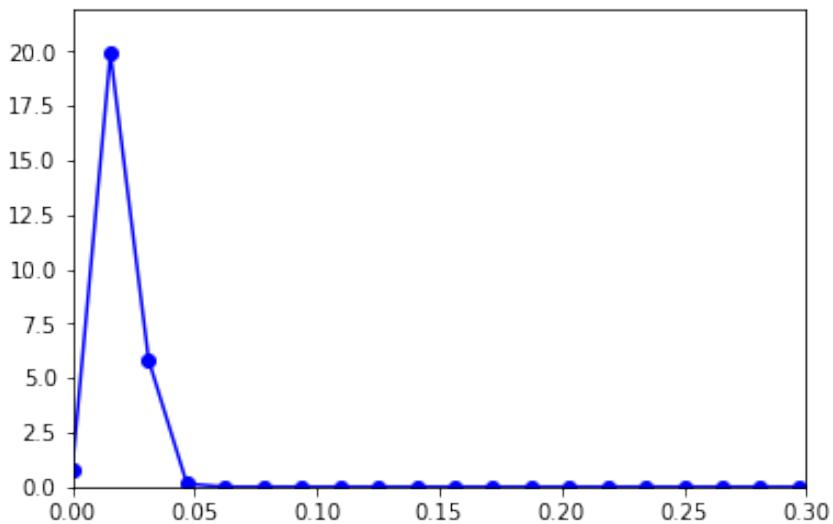
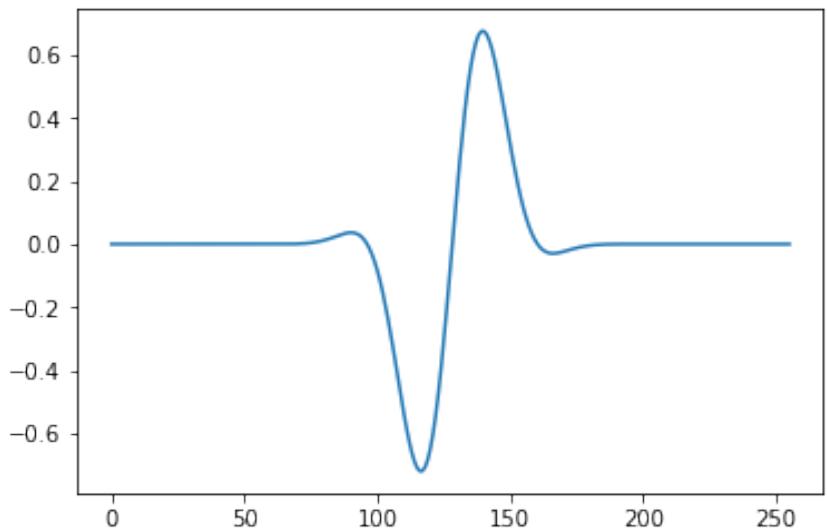


```
In [3]: sigma = 16
g = gaussian(N, sigma) # costruzione della gaussiana:
# e' una gaussiana di centro c e deviazione standard sigma
```

```
# calcolata nei punti [0:N-1];
# e' infatti ipotizzato che la sequenza di dati lunga N abbia
# che corrono da 0 a N-1
plt.figure(21); plt.plot(g); plt.show()
```



```
In [4]: y = g * s
        plt.figure(31); plt.plot(y); plt.show()
        b = 4
        y1 = perbas(y,b)
        V = fft(y1)
        vf = np.arange(0,1/Tc,b/(Tc*N))
        plt.figure(35); plt.plot(vf, abs(V), 'b-o'); plt.axis([0., 0.3, 0., max(abs(V))*1.1])
```



Torna al par. **8.1**

Appendice M

Esempio ricostruzione partitura

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
from scipy.signal import gaussian
from libreria_NLALD.stft import *
from libreria_NLALD.hamming import *
```



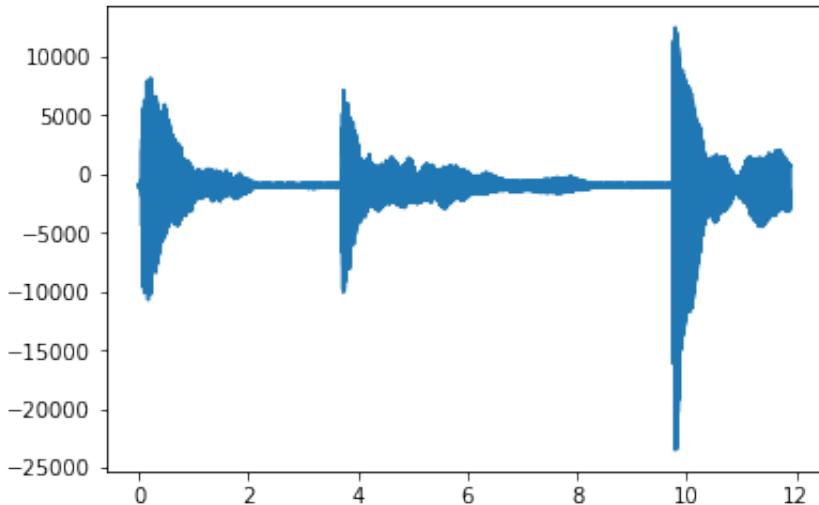
```
In [2]: # ATTENZIONE: SISTEMARE IL PATH !
selezione = 3
if selezione==1:
    fc,ytot = wavfile.read('../MATERIALE_DA_TENERE_SU_HOMEPAGE/intro_KeithJarrett_
    I = np.where(ytot[:,0] >= 10)
    istart = I[0][0] # ci possono essere fino a due secondi di silenzio all'inizio de
    N = 65536 # numero di campioni del segnale campionato a 44.1 KHz
    a = 128 # passo di traslazione nel tempo
    b = 16 # passo di campionamento in frequenza
elif selezione==2:
    fc,ytot = wavfile.read('../MATERIALE_DA_TENERE_SU_HOMEPAGE/pianoforte_4.wav')
    istart = 46000
    N = 65536*4 # numero di campioni del segnale campionato a 44.1 KHz
    a = 128*4 # passo di campionamento nel tempo
    b = 32*2 # passo di campionamento in frequenza
elif selezione==3:
    fc,ytot = wavfile.read('../pianoforte_3.wav') # note "staccate"
    istart = 50000
```

```

N = 65536*8 # numero di campioni del segnale campionato a 44.1 KHz
a = 128*8 # passo di campionamento nel tempo
b = 32*2 # passo di campionamento in frequenza
#endif
print("fc = ", fc)
print("istart = ", istart)
y = ytot[istart+np.arange(N),:]
#wavefile.play(y, fc)
y = y[:,0]
plt.figure(1); plt.plot(np.arange(float(N))/fc, y); plt.show()

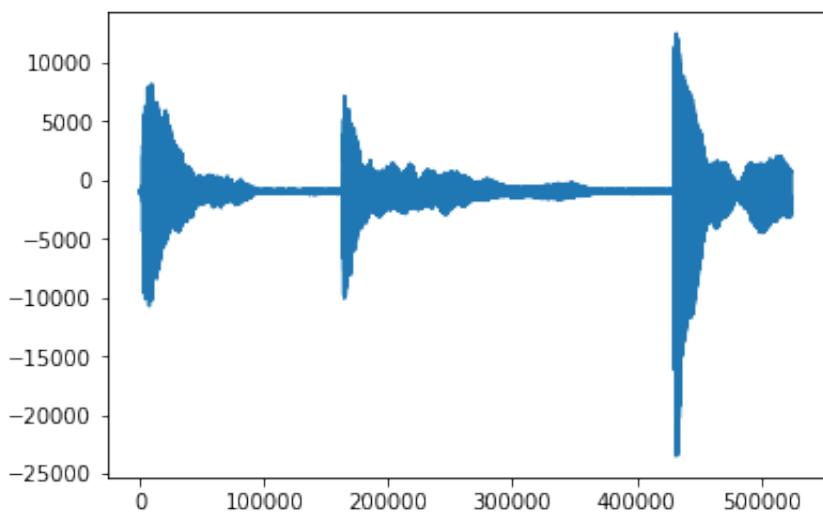
fc = 44100
istart = 50000

```

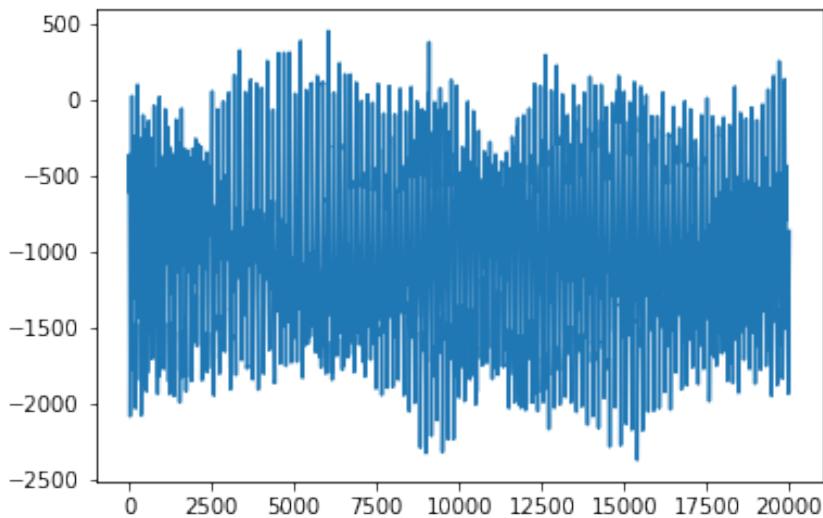


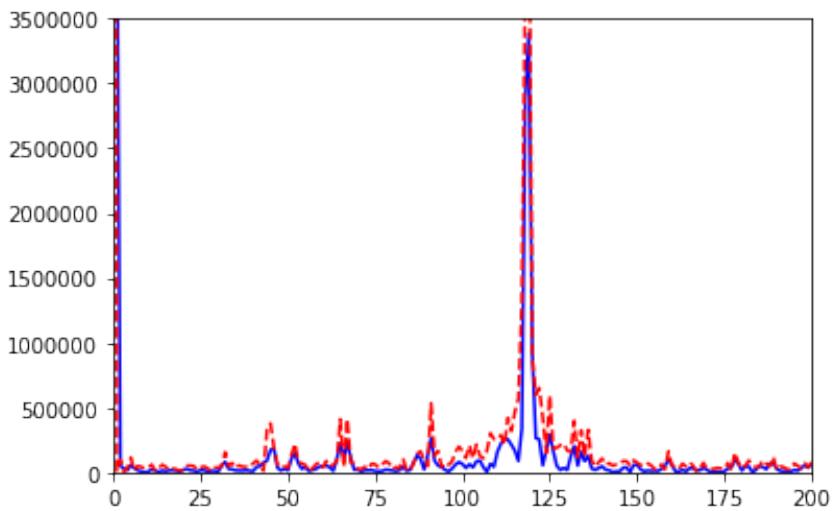
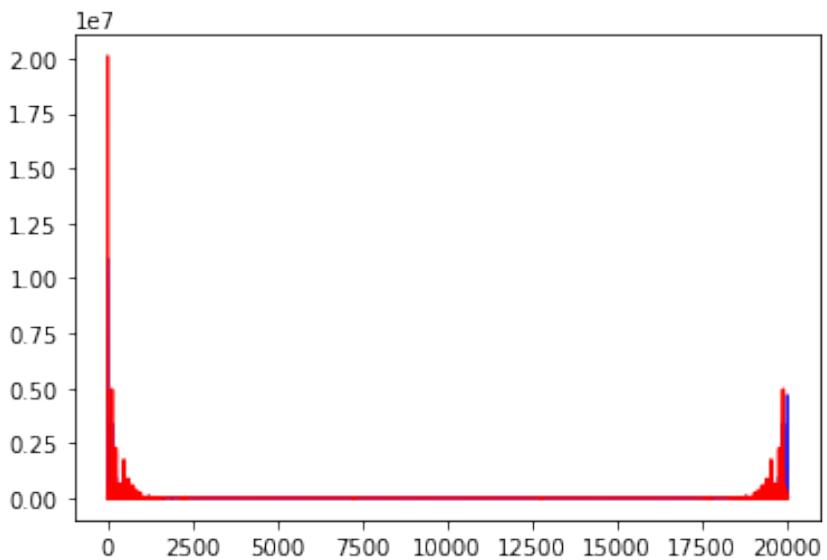
```

In [3]: plt.figure(11); plt.plot(y); plt.show()
start1 = 50000; N1 = 20000; y1 = y[start1:start1+N1]
print("df = ",fc/N1, " [Hz]")
plt.figure(12); plt.plot(y1); plt.show()
w1 = hamming(N1)
Y1 = fft(y1)
WY1 = fft(w1 * y1)
plt.figure(13); plt.plot(np.abs(WY1), 'b-'); plt.plot(np.abs(Y1), 'r-'); plt.show()
plt.figure(14); plt.axis([0., 200., 0., 3500000.]); plt.plot(np.abs(WY1[0:2000]), 'b-
```



df = 2.205 [Hz]



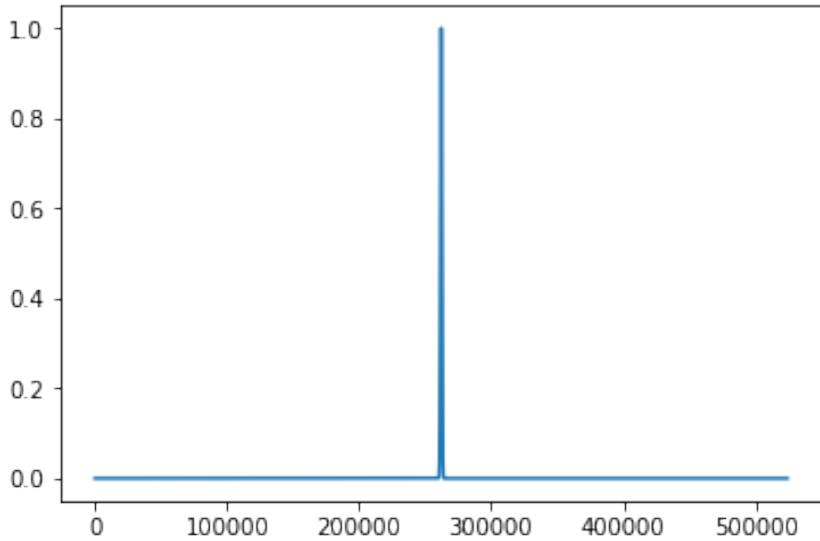


```
In [4]: sigma = 512
g = gaussian(N, sigma) # costruzione della gaussiana:
# e' una gaussiana di centro c e deviazione standard sigma
# calcolata nei punti [0:N-1];
```

```

# e' infatti ipotizzato che la sequenza di dati lunga N abbi
# che corrono da 0 a N-1
plt.figure(21); plt.plot(g); plt.show()

```



```

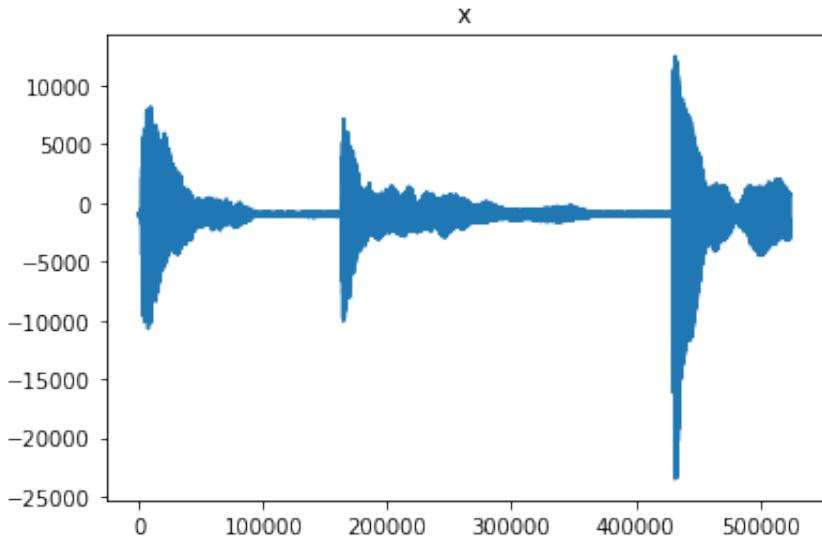
In [5]: # C = stft(y,g,a,b) :
x = y.copy()
plt.figure(39); plt.plot(x); plt.title('x'); plt.show()
w = g
n = len(x)
print("n = ",n)
print("a = ",a)
print("n/a = ",n/a)
print("b = ",b)
print("n/b = ",n/b)
nt = int(n/a)
res = np.zeros((nt,int(n/b)))
w1 = np.concatenate((w, np.zeros(n-len(w))))
plt.figure(40); plt.plot(w1)
ww1 = np.concatenate((w1, w1))
plt.figure(42); plt.plot(ww1); plt.title('ww1'); plt.show()
n2y = np.zeros(nt)
for jj in range(nt):
    wf = ww1[ (n - jj*a) : (2*n - jj*a) ]
    y = x * wf
    n2y[int(jj+nt/2) % nt] = np.linalg.norm(y,2)

```

```

y1 = perbas(y,b)
v = fft(y1)
if jj==0:
    plt.figure(100+jj); plt.plot(wf); plt.title('wf'); plt.show()
    plt.figure(200+jj); plt.plot(y); plt.title('y'); plt.show()
    plt.figure(300+jj); plt.plot(y1); plt.title('y1'); plt.show()
    plt.figure(400+jj); plt.plot(v); plt.title('v'); plt.show()
#endif
res[int(jj+nt/2) % nt] [:] = v
#endiffor
plt.figure(999); plt.plot(n2y); plt.title('n2y'); plt.show()
C = res.T

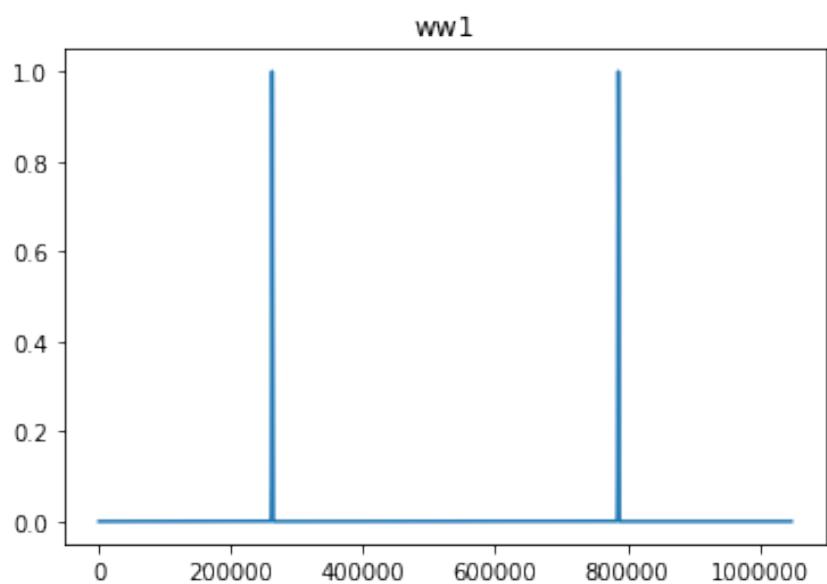
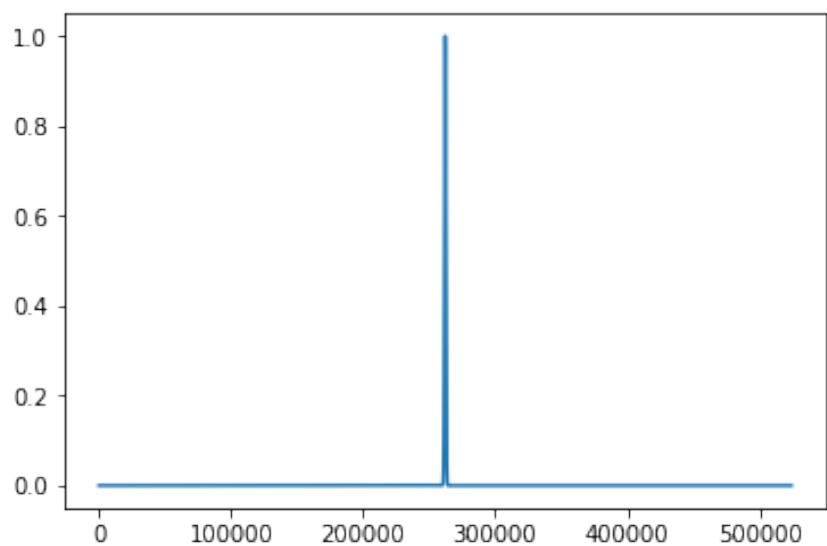
```

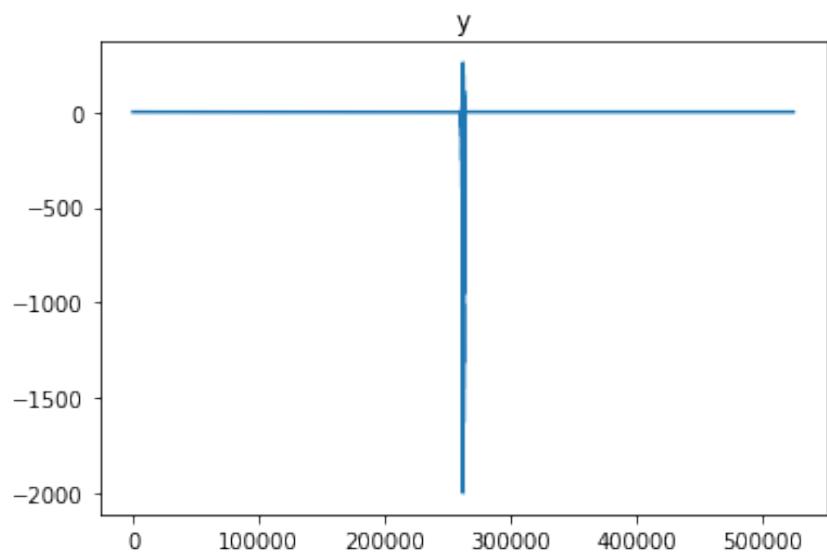
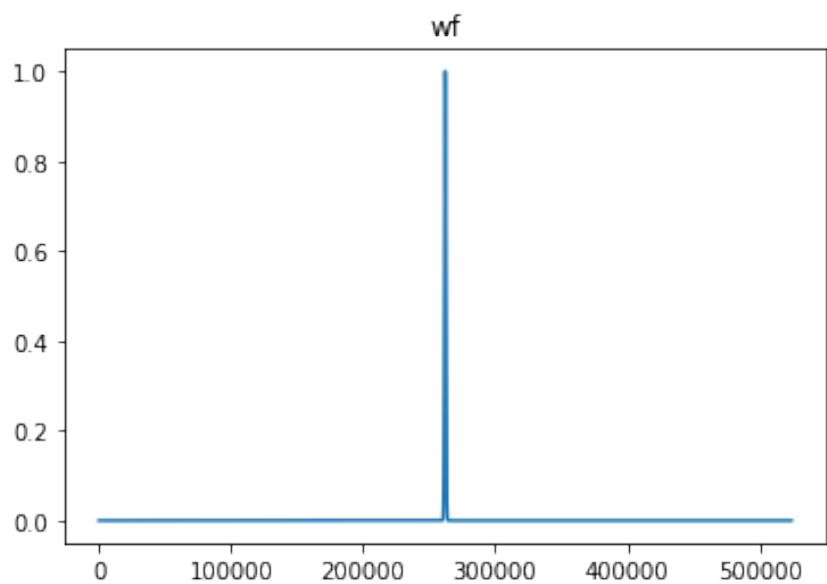


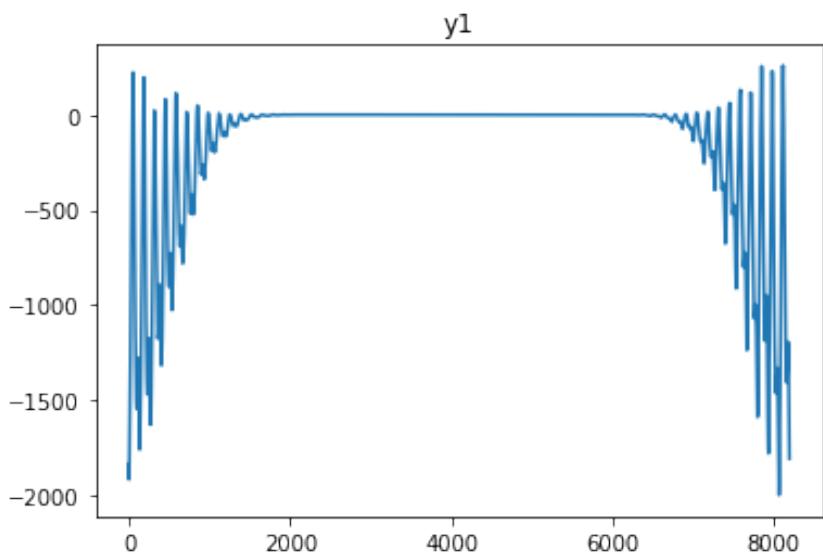
```

n = 524288
a = 1024
n/a = 512.0
b = 64
n/b = 8192.0

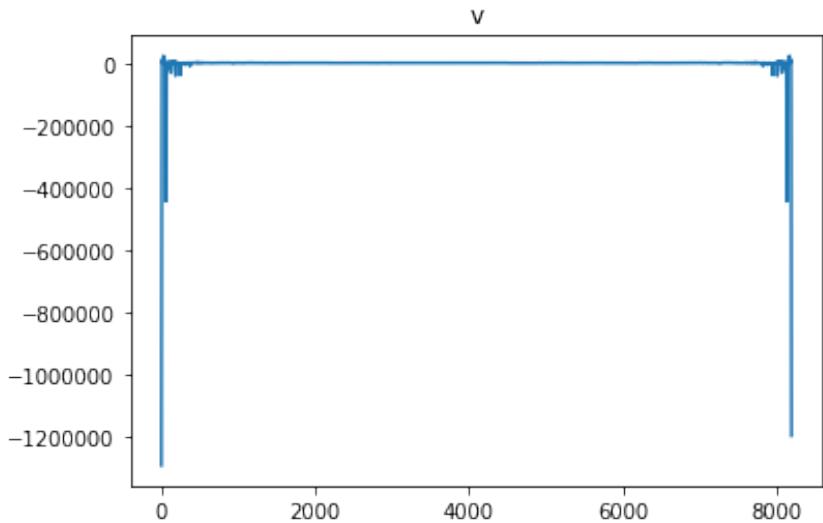
```



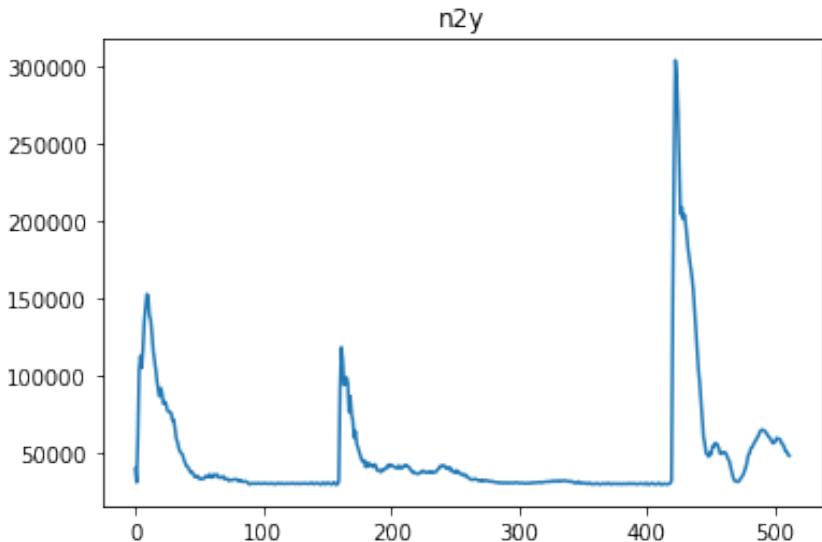




```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/numpy/core/_as
    return array(a, dtype, copy=False, order=order)
```



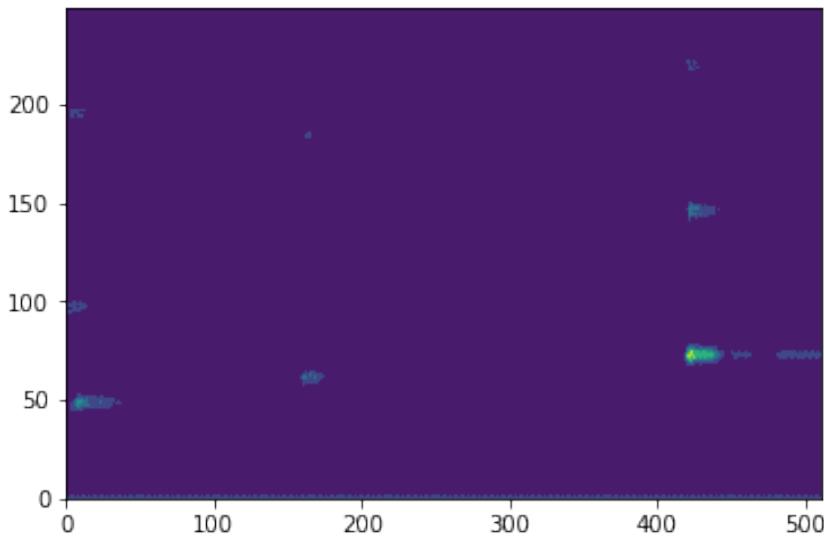
```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/ipykernel_laun
```



```
In [6]: print(C.shape)
      plt.figure(2000); plt.contourf(np.array(np.abs(C[0:250,0:1000])))
      #spettro_g_musica(C,N,a,b);
```

(8192, 512)

Out[6]: <matplotlib.contour.QuadContourSet at 0x7fd58e030e10>

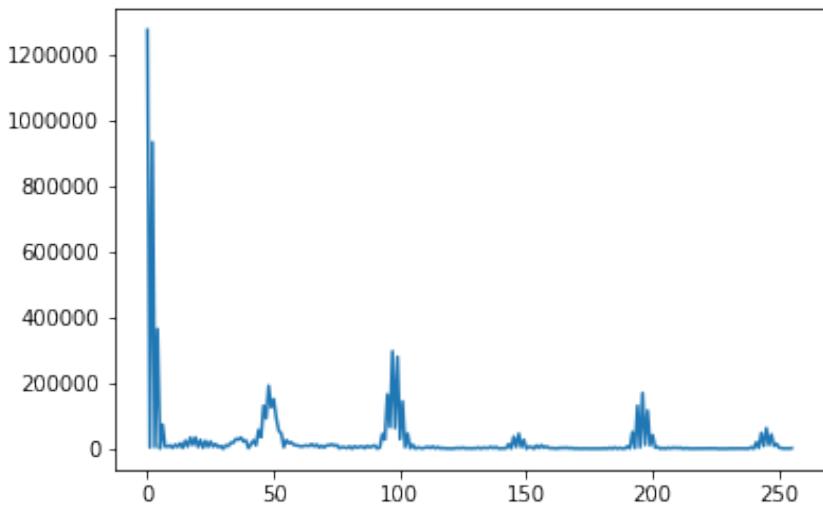
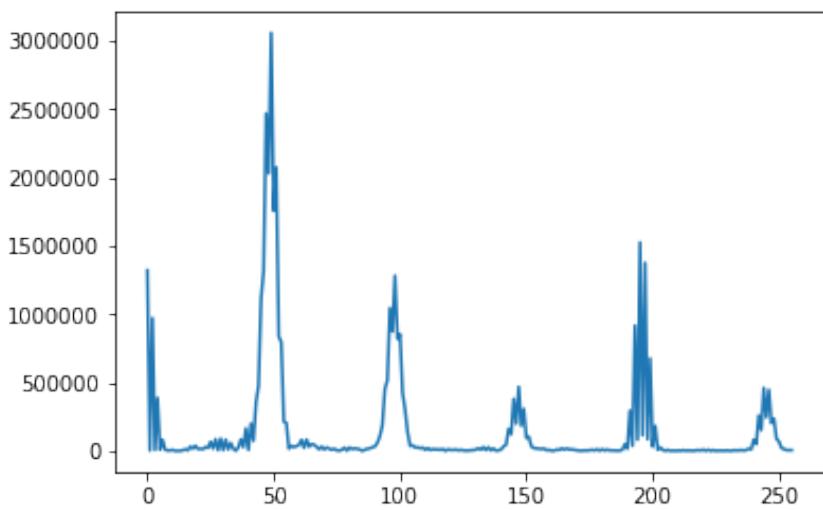


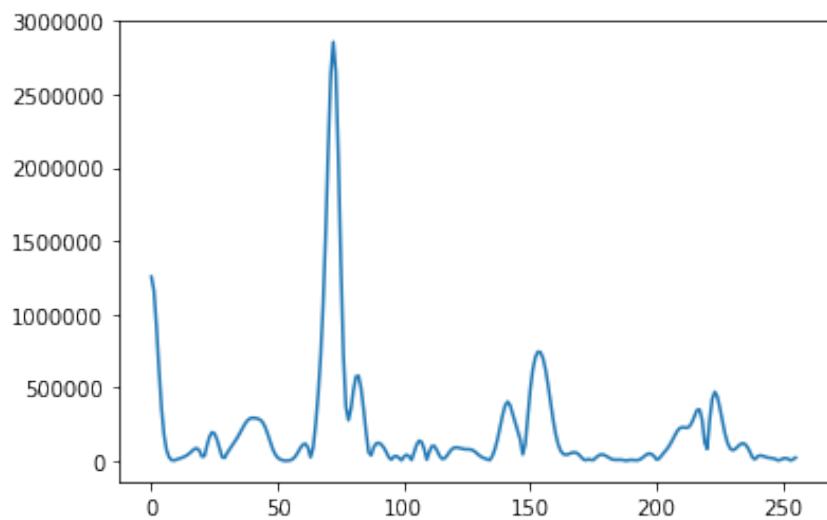
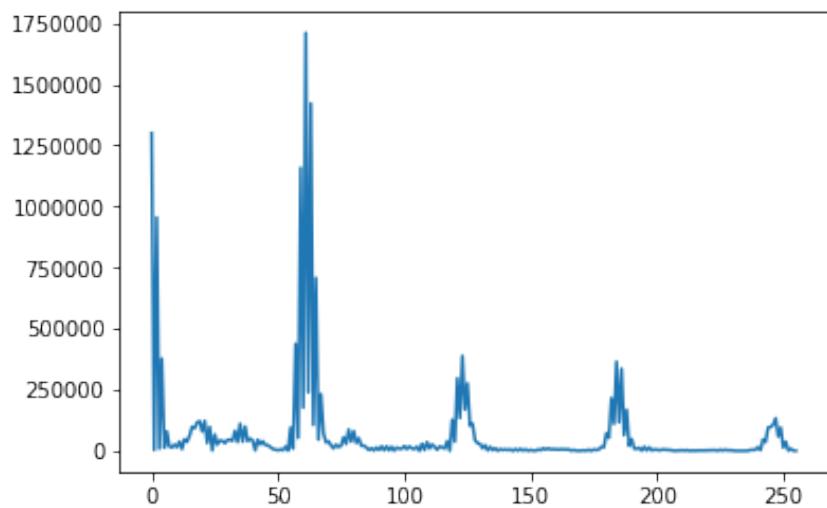
In [7]: `np.abs(C[0:80,0:50])`

Out [7]: `array([[1.30758131e+06, 1.30366714e+06, 1.27032292e+06, ...], [1.28721038e+06, 1.30794391e+06, 1.30689802e+06], [1.21040265e+06, 8.60001592e+05, 2.18583823e+03, ...], [8.45673861e+05, 1.21096320e+06, 8.60086856e+05], [9.59862846e+05, 1.47959739e+04, 9.32147197e+05, ...], [7.94867130e+03, 9.60809911e+05, 9.03961265e+03], ..., [7.96118243e+04, 2.59266299e+03, 4.59325568e+04, ...], [4.96724125e+02, 4.14135705e+03, 9.72124412e+03], [1.05934970e+04, 1.56853064e+04, 4.79679406e+04, ...], [1.13839788e+03, 4.23053191e+03, 7.32952226e+03], [2.32106924e+04, 1.72742475e+04, 1.66188273e+05, ...], [3.55578436e+03, 3.16084772e+03, 2.20108082e+03]])`

In [8]: `plt.figure(1); plt.plot(abs(C[0:256,10])) # prima nota`
`plt.figure(2); plt.plot(abs(C[0:256,50])) # rumore sottofondo prevale`
`plt.figure(3); plt.plot(abs(C[0:256,170])) # seconda nota`
`plt.figure(4); plt.plot(abs(C[0:256,420])) # terza nota`

Out [8]: [`<matplotlib.lines.Line2D at 0x7fd5884707f0>`]





In [9]: `perbas(np.array([1, 2, 3, 4, 5, 6, 7, 8, 9]),3)`

Out [9]: `array([12, 15, 18])`

Torna al par. 8.1.1

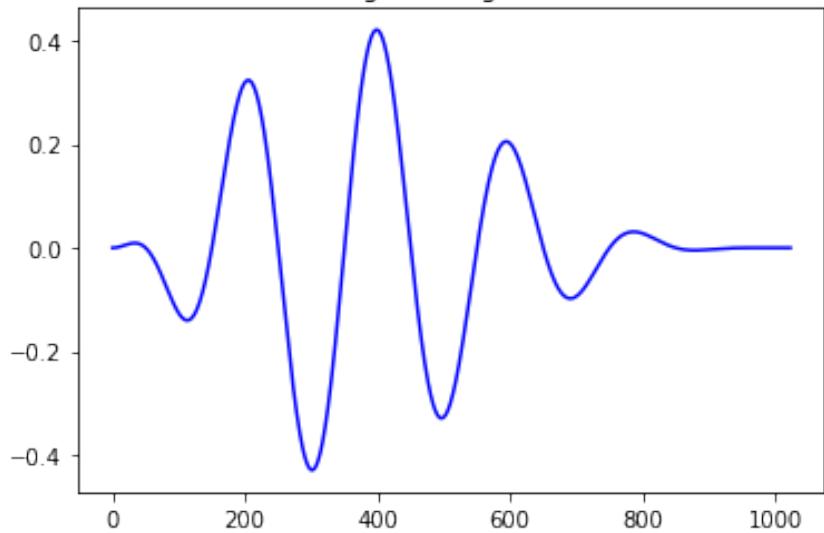
Appendice N

Intro wavelets

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
from scipy.sparse import *
from libreria_NLALD.upsample import *
from numpy.fft import *
```

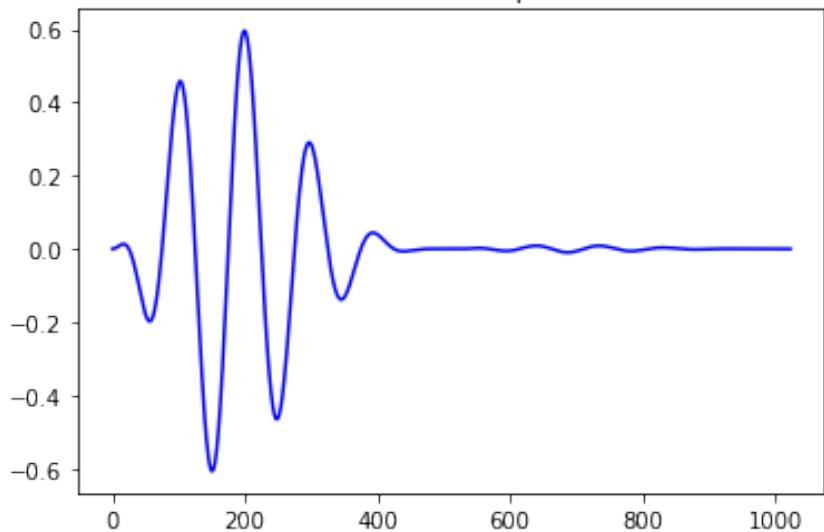
```
In [2]: # Trasformata di Haar :
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
N = 1024
if True:
    omega_nu = (2*np.pi/N) * 0.01*(N/2)
    u = 20.0 * (np.arange(float(N))/N)**2 * (1.0 - (np.arange(float(N))/N))**4 * np
else:
    u = 20.0*np.random.randn(N)
#endif
plt.figure(7); plt.plot(u,'b-'); plt.title('segnaile originario'); plt.show()
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(8); plt.plot(u_Haar1,'b-'); plt.title('trasformata di Haar - primo livel
temp = simula_DLTI(scalnum,np.array([1.]),trend)
trend2 = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),trend)
flutt2 = temp[1:N+1:2]
plt.figure(9); plt.plot(np.concatenate((trend2, flutt2)), 'b-'); plt.title('t
```

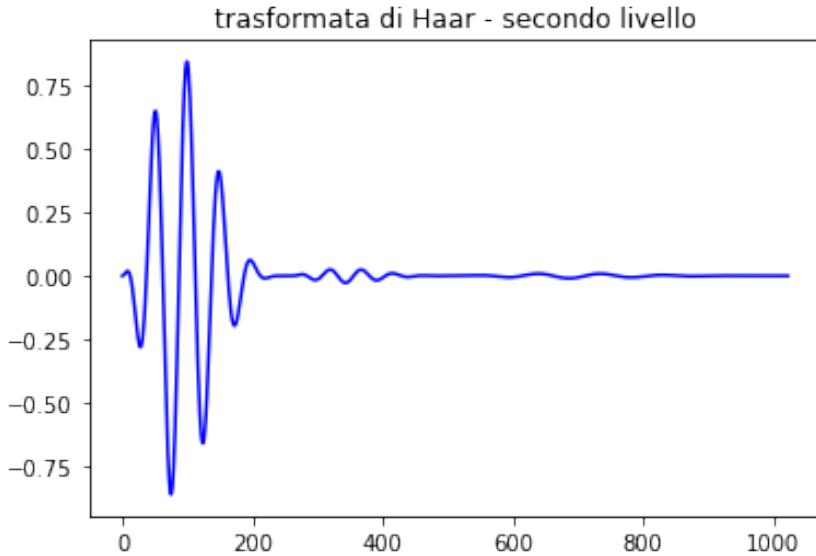
segnale originario



```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOP/libro_MNAD/sito_web_MNAD  
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

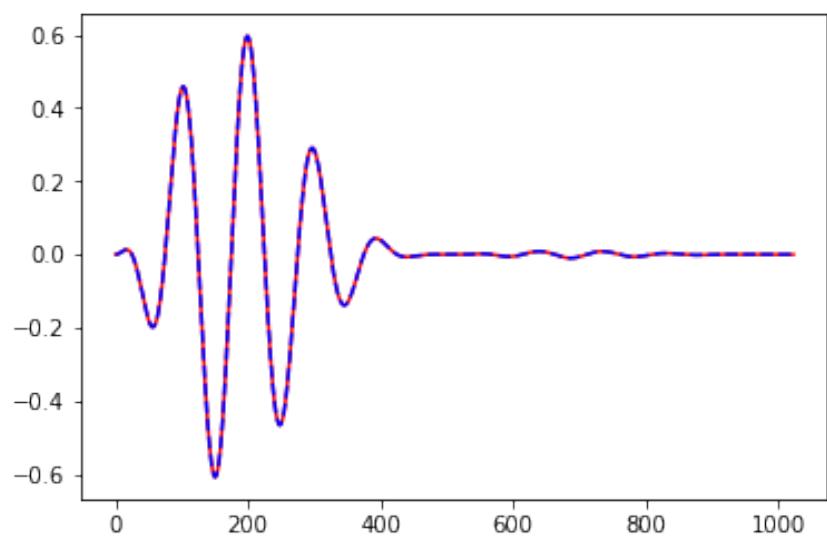
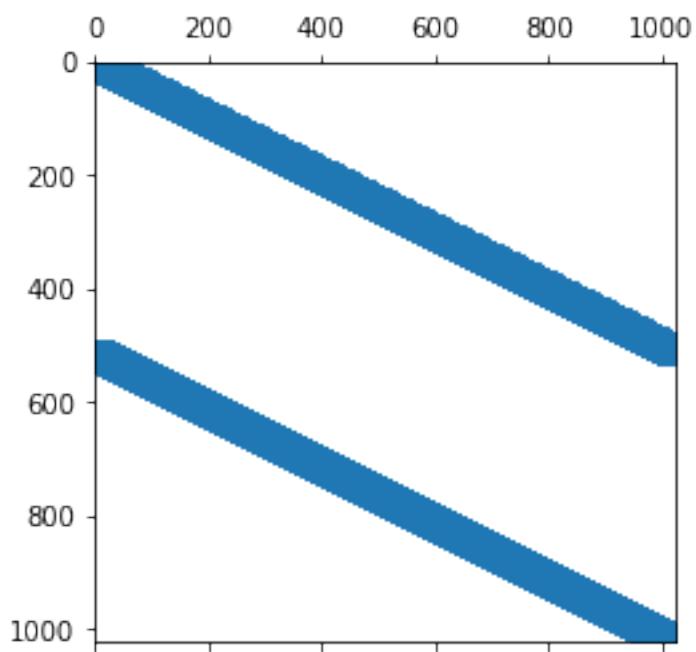
trasformata di Haar - primo livello

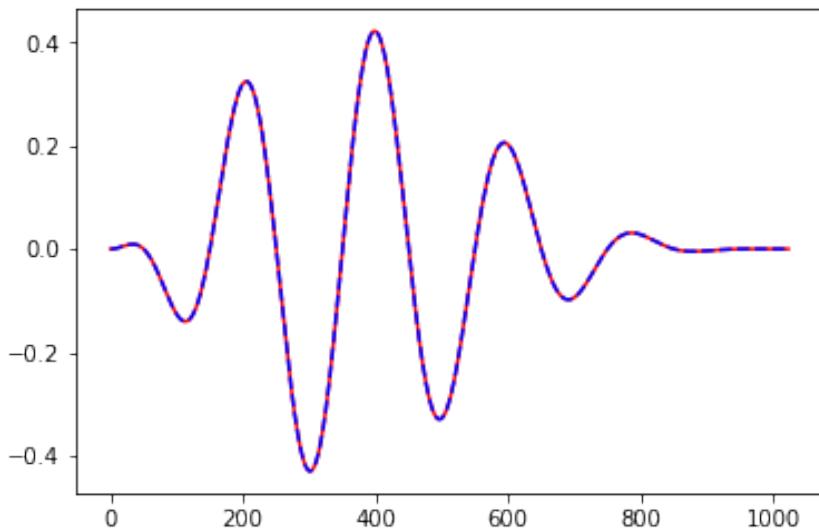




```
In [3]: # facciamo la adesso con una matrice sparsa (NB: la sparsità è dovuta al supporto lo
T1 = lil_matrix((N,N))
for i in range(int(N/2)):
    T1[i,i*2] = 1./np.sqrt(2)
    T1[i,i*2+1] = 1./np.sqrt(2)
#endif
for i in range(int(N/2),N):
    T1[i,(i-N/2)*2] = 1./np.sqrt(2)
    T1[i,(i-N/2)*2+1] = -1./np.sqrt(2)
#endif
plt.figure(11); plt.spy(T1); plt.show()
u_haar = T1 @ np.atleast_2d(u).T
plt.figure(12); plt.plot(u_Haar1, 'r-'); plt.plot(u_haar, 'b--');
# T1 e' una matrice ortogonale, perchè costituita da wavelets ortogonali, quindi:
u_ric = T1.T @ u_haar
plt.figure(13); plt.plot(u, 'r-'); plt.plot(u_ric, 'b--');
```

```
/Users/marcuzzi/opt/anaconda3/envs/py36/lib/python3.6/site-packages/scipy/sparse/l
if not x.flags.writeable:
```

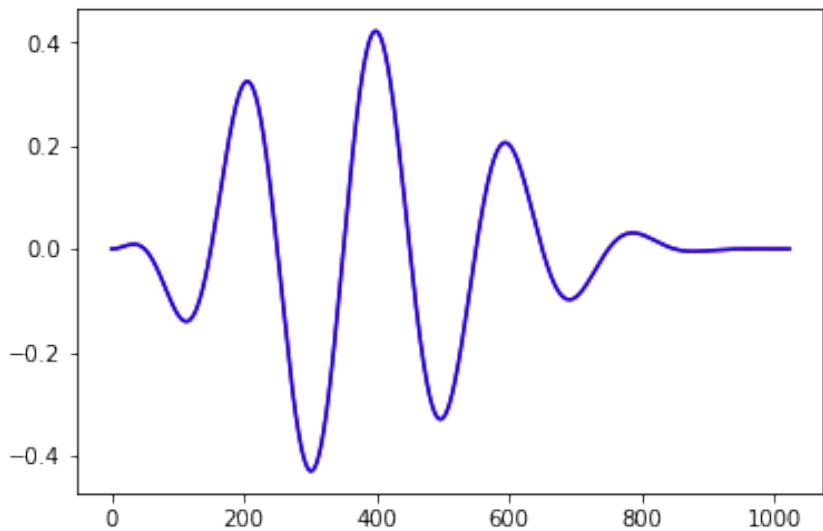




In [4]: # trasformata inversa con i filtri:

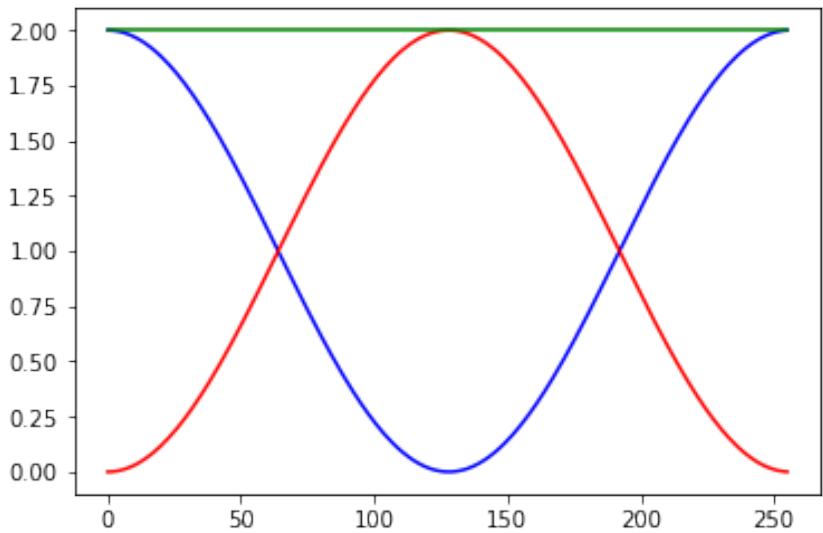
```
u_ric_filt = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(trend,
u_ric_filt += simula_DLTI(wavenum[range(Nc-1,-1,-1)],np.array([1.]), upsample(flutt
plt.figure(20); plt.plot(u,'r-'); plt.plot(u_ric_filt,'b-');
```

/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n



In [5]: # contenuto in frequenza delle wavelets :

```
haarscalfun = np.concatenate(([1./np.sqrt(2), 1./np.sqrt(2)]), np.zeros(256-2))
haarwavelet = np.concatenate(([1./np.sqrt(2), -1./np.sqrt(2)]), np.zeros(256-2))
plt.figure(10); plt.plot(np.abs(fft(haarscalfun))**2, 'b-'); plt.plot(np.abs(fft(haa
```



Torna al par. [8.2.1](#)

Appendice O

Disegna wavelets

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

In [2]: def Psi(tipo,k,r,n,s0,tau0,tf):
    N = len(k)
    p = np.log(N) / np.log(s0)
    print("p = ",p)
    formulazione_Daub = 1 # =0 indici r positivi (Strang-Nguyen), =1 indici r negati
    if formulazione_Daub:
        ts = (k/(s0**r) - n*N)*tau0
    else:
        ts = (s0**r) * (k - n*N)*tau0
    #endif
    #print ts
    out = np.zeros(N)
    if tipo=='Haar':
        for i in range(N):
            if ts[i] >= 0.0 and ts[i] < 0.5:
                out[i] = 1.0 / p
            elif ts[i] >= 0.5 and ts[i] <= 1.0:
                out[i] = -1.0 / p
            #endif
        #endfor
    elif tipo=='MexicanHat':
        out = (1 - ts**2) * np.exp(-ts**2/2.0)
    #endif
    if formulazione_Daub:
        out /= s0**(r/2.0)
    else:
        out *= s0**(-r/2.0)
```

```

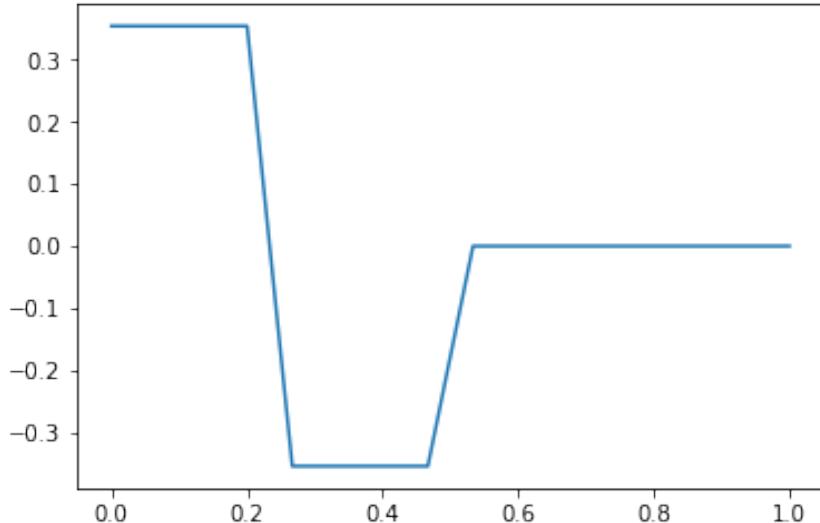
#endif
#print out
return out

In [3]: N = 2**4
dt = 1.0 / (N-1)
tf = (N-1)*dt
k = np.array(range(0,N))
t = np.arange(0.0,tf+dt/2,dt)
s0 = 2
r = -1
n = 0
print("n = ",n)
#print t.shape
plt.figure(1); plt.plot(t,Psi('Haar',k,r,n,s0,dt,tf))

n = 0
p = 4.0

```

Out[3]: [`<matplotlib.lines.Line2D at 0x7f8b2f46c828>`]



In [4]: N = 2**6
dt = 1.0 / (N-1)

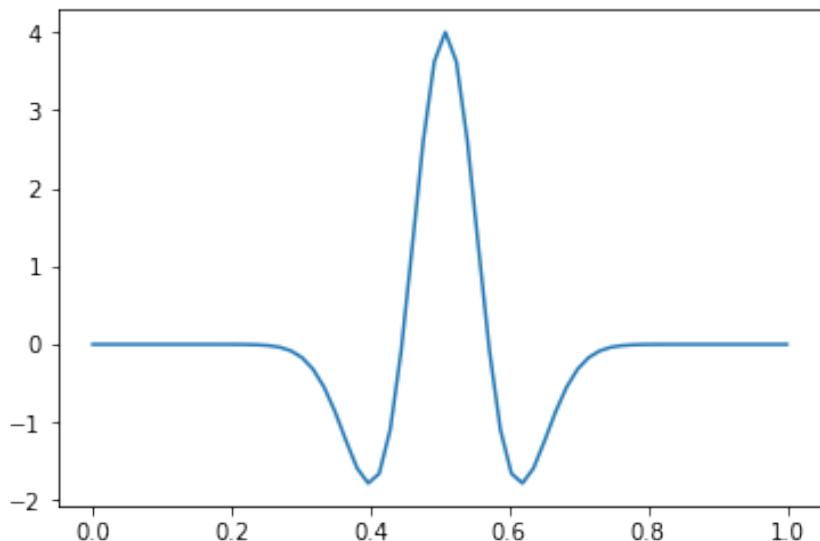
```

tf = (N-1)*dt
k = np.array(range(0,N))
t = np.arange(0.0,tf+dt/2,dt)
s0 = 2
r = -4
n = 8
#print t.shape
plt.figure(1); plt.plot(t,Psi('MexicanHat',k,r,n,s0,dt,tf))

p = 6.0

```

Out [4]: [`<matplotlib.lines.Line2D at 0x7f8b2f55b828>`]



In [5]: `1./np.sqrt(2.0)`

Out [5]: 0.7071067811865475

```

In [6]: N = 2**4
        dt = 1.0 / (N-1)
        tf = (N-1)*dt
        k = np.array(range(0,N))
        t = np.arange(0.0,tf+dt/2,dt)
        s0 = 2

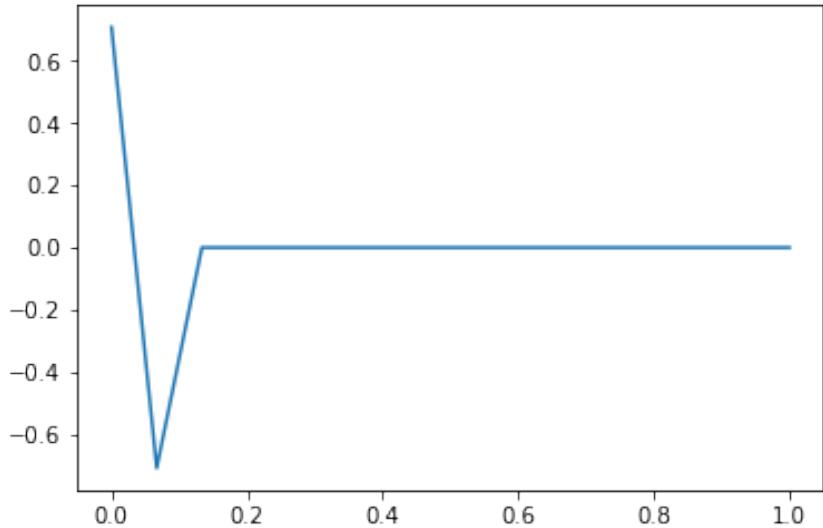
```

```

r = -3
n = 0
print("n = ",n)
#print t.shape
print(Psi('Haar',k,r,n,s0,dt,tf))
plt.figure(1); plt.plot(t,Psi('Haar',k,r,n,s0,dt,tf));

n = 0
p = 4.0
[ 0.70710678 -0.70710678  0.          0.          0.          0.
  0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.          ]]
p = 4.0

```



Torna al par. 8.2

Appendice P

Multi-risoluzione

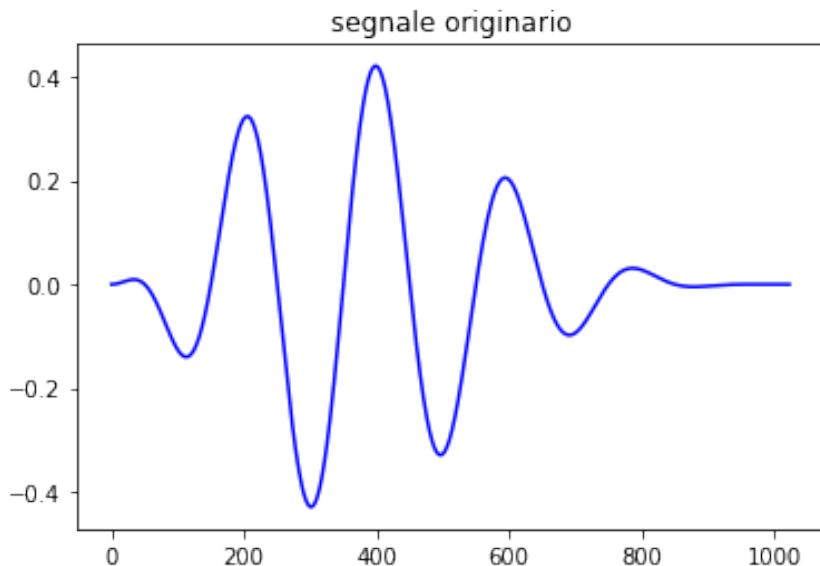
```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
from scipy.sparse import *
from libreria_NLALD.upsample import *
from numpy.fft import *

In [2]: # Trasformata di Haar :
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
N = 1024
omega_nu = (2*np.pi/N) * 0.01*(N/2)
u = 20.0 * (np.arange(float(N))/N)**2 * (1.0 - (np.arange(float(N))/N))**4 * np.cos(omega_nu * np.arange(float(N))/N)
plt.figure(7); plt.plot(u,'b-'); plt.title('segnale originario'); plt.show()
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(8); plt.plot(u_Haar1,'b-'); plt.title('trasformata di Haar - primo livel
# trasformata inversa con i filtri:
trend_0 = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(trend,2)
flutt_0 = simula_DLTI(wavenum[range(Nc-1,-1,-1)],np.array([1.]), upsample(flutt,2)
plt.figure(10); plt.plot(u,'r-',label='segnale originario'); plt.plot(trend_0,'b.'
plt.figure(11); plt.plot(flutt_0,'g.',label='flutti1 anti-trasformato'); plt.axis([5
plt.figure(12); plt.plot(u,'r-',label='segnale originario'); plt.plot(trend_0,'b.',
#
temp = simula_DLTI(scalnum,np.array([1.]),trend)
trend2 = temp[1:N+1:2]
```

```

temp = simula_DLTI(wavenum,np.array([1.]),trend)
flutt2 = temp[1:N+1:2]
u_Haar2 = np.concatenate((trend2, flutt2, flutt))
plt.figure(19); plt.plot(u_Haar2,'b-'); plt.title('trasformata di Haar - secondo li
# trasformata inversa con i filtri:
trend_0 = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(trend,2)
trend2_1 = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(trend2,2)
trend2_0 = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(trend2_1
flutt_0 = simula_DLTI(wavenum[range(Nc-1,-1,-1)],np.array([1.]), upsample(flutt,2)
flutt2_1 = simula_DLTI(wavenum[range(Nc-1,-1,-1)],np.array([1.]), upsample(flutt2,2
flutt2_0 = simula_DLTI(scalnum[range(Nc-1,-1,-1)],np.array([1.]), upsample(flutt2_1
plt.figure(20); plt.plot(u,'r-',label='segnale originario'); plt.plot(trend2_0,'b.'
plt.figure(21); plt.plot(flutt2_0,'g.',label='flutt2 anti-trasformato'); plt.axis([
plt.figure(22); plt.plot(u,'r-',label='segnale originario'); plt.plot(trend2_0,'b.'
plt.figure(24); plt.plot(u,'r-',label='segnale originario'); plt.plot(trend2_0,'b.'

```

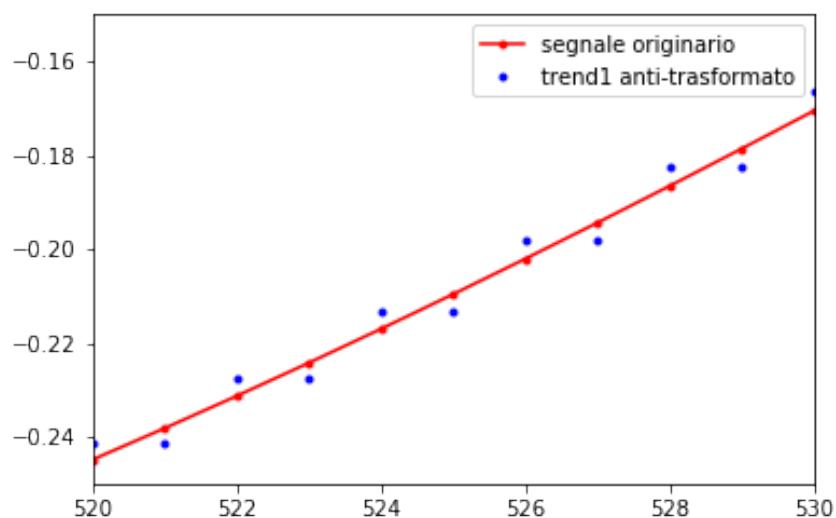
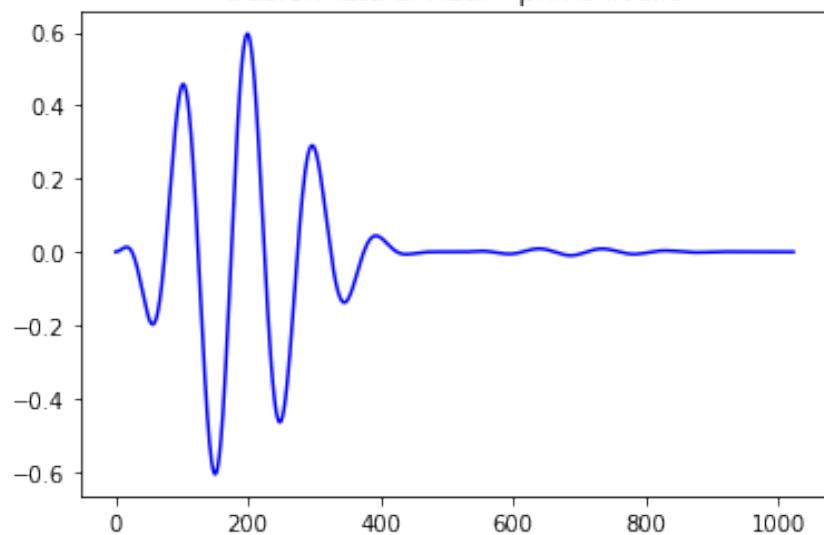


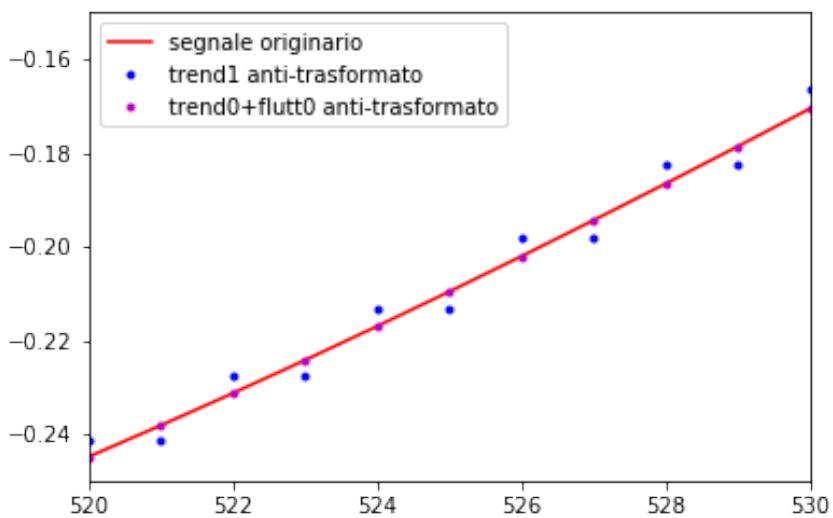
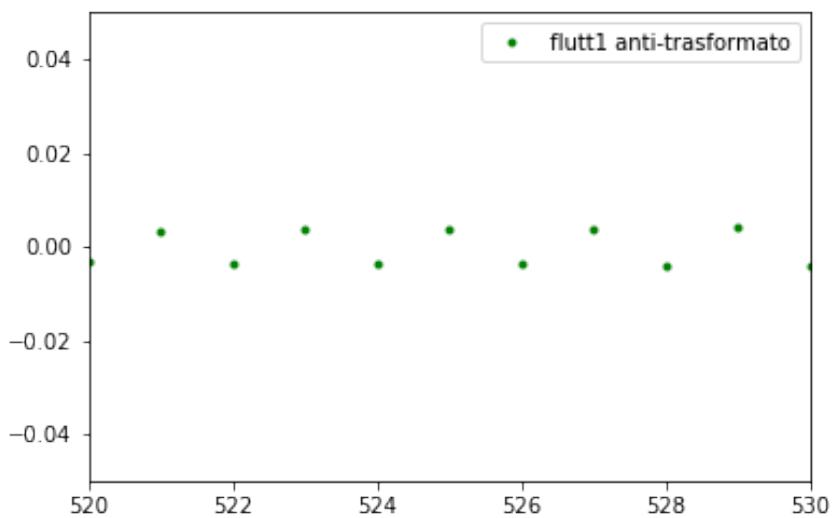
```

/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n

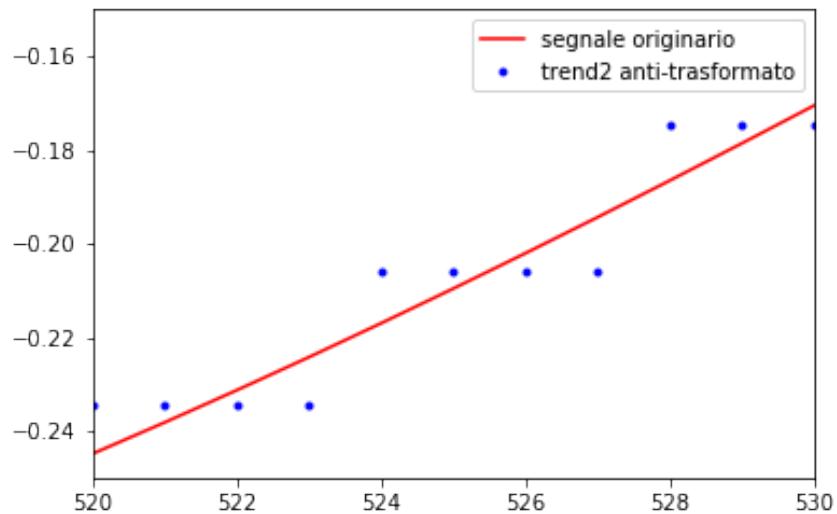
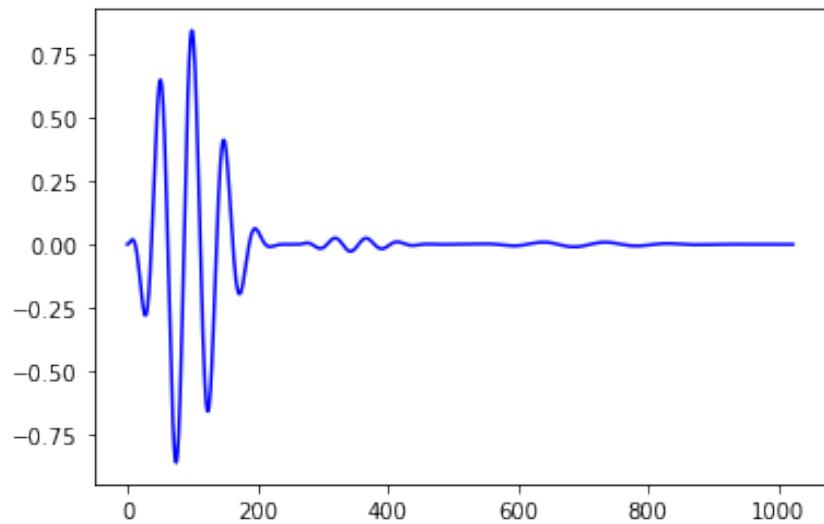
```

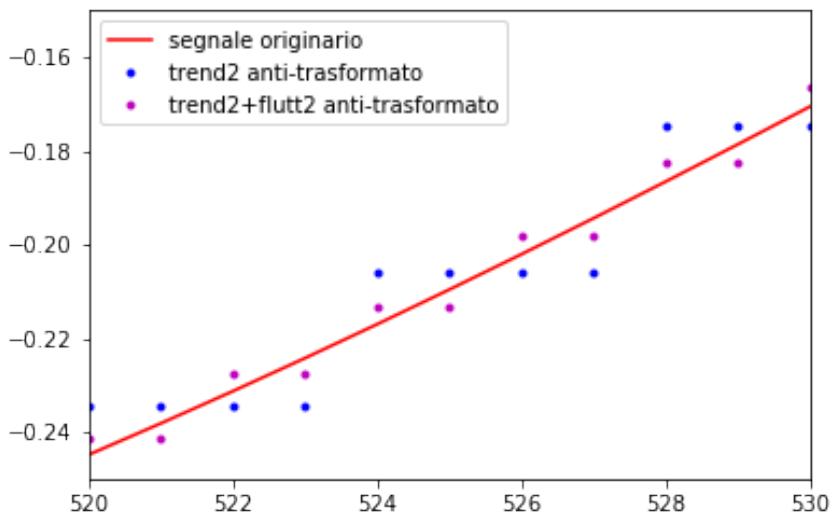
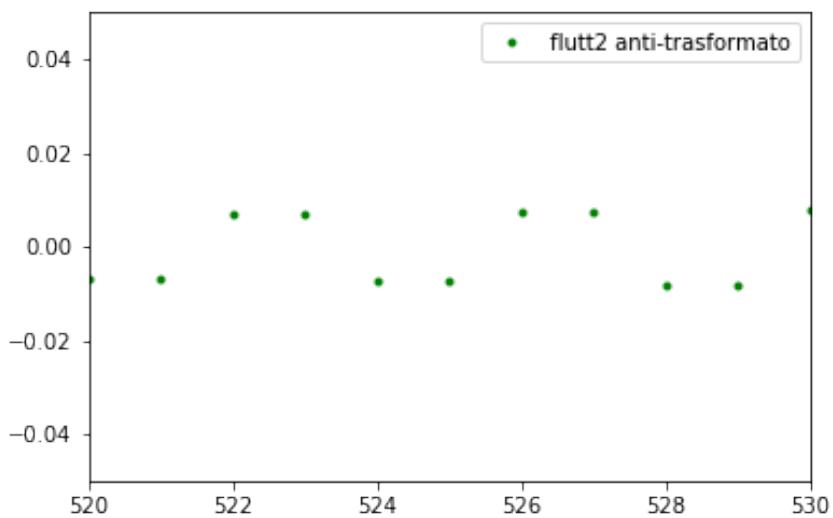
trasformata di Haar - primo livello

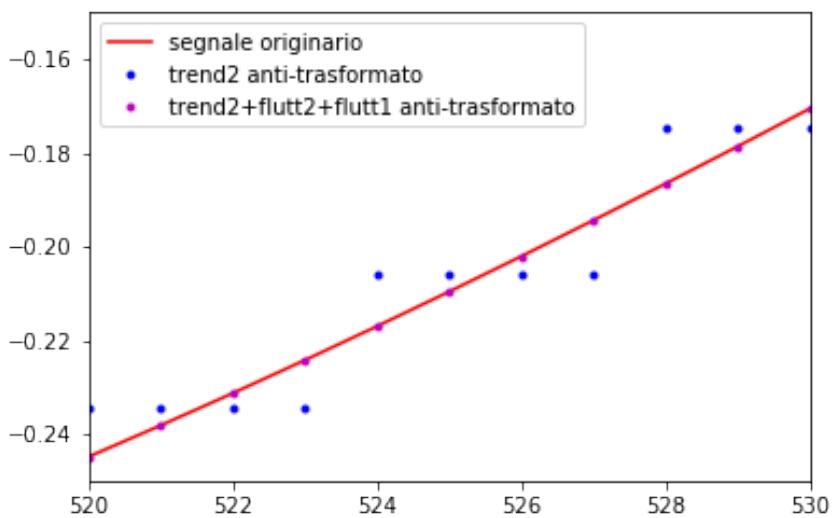




trasformata di Haar - secondo livello







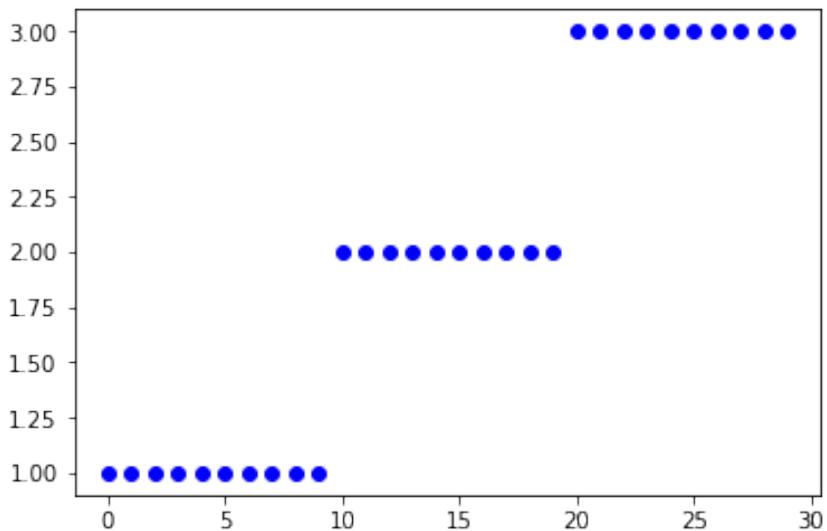
Torna al par. 8.2.1

Appendice Q

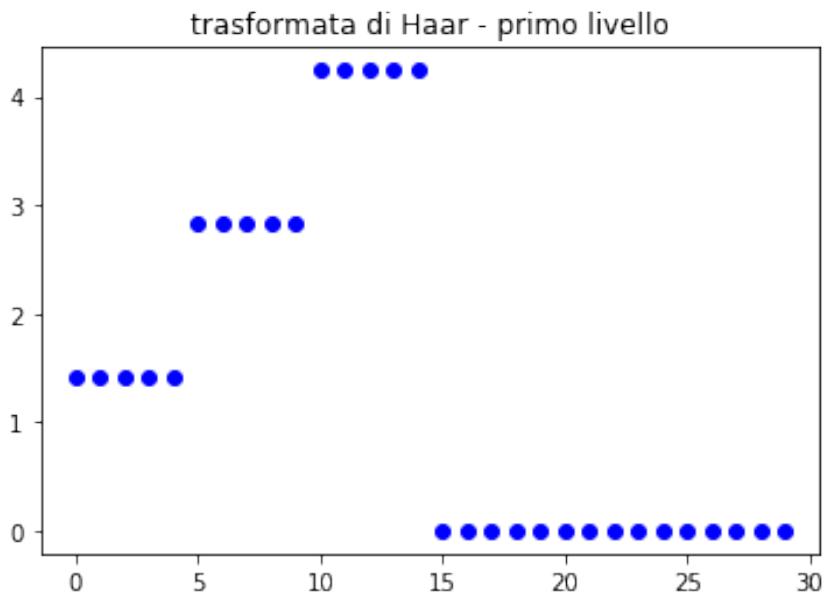
Analisi fluttuazioni wavelets

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
```

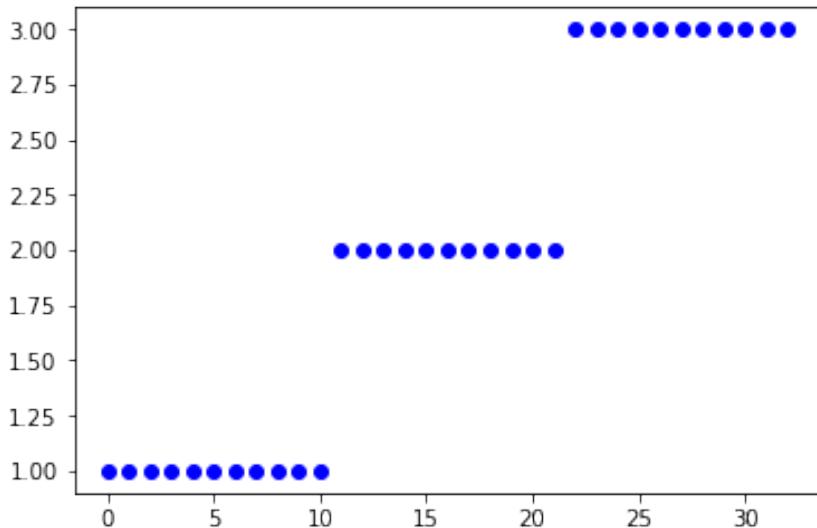
```
In [2]: Np = 10
u = np.concatenate((np.repeat(np.array([1.]),Np), np.repeat(np.array([2.]),Np), np.
N = len(u)
plt.figure(1); plt.plot(u,'bo'); plt.show()
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(2); plt.plot(u_Haar1,'bo'); plt.title('trasformata di Haar - primo livel
```



```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD  
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

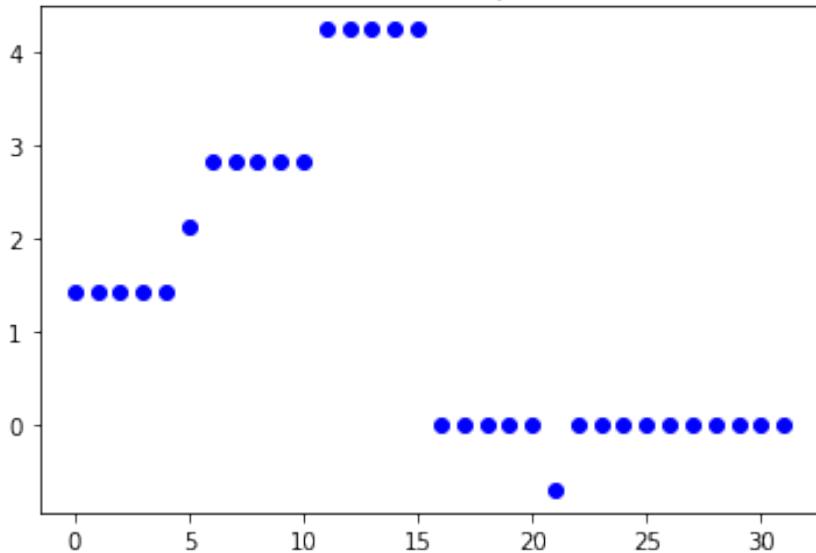


```
In [3]: Np = 11; # qui si vede la "fragilita'" del meccanismo ...
u = np.concatenate((np.repeat(np.array([1.]),Np), np.repeat(np.array([2.]),Np), np.
N = len(u)
plt.figure(11); plt.plot(u,'bo'); plt.show()
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(12); plt.plot(u_Haar1,'bo'); plt.title('trasformata di Haar - primo live
```



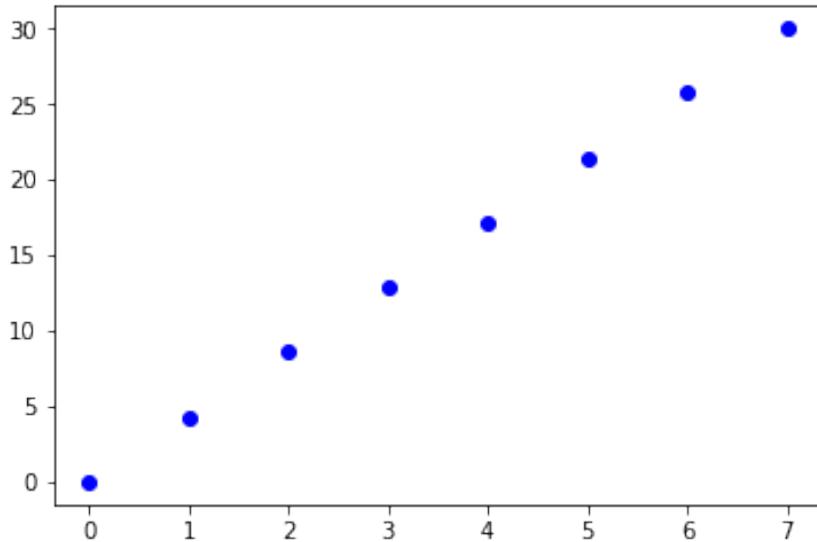
```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

trasformata di Haar - primo livello

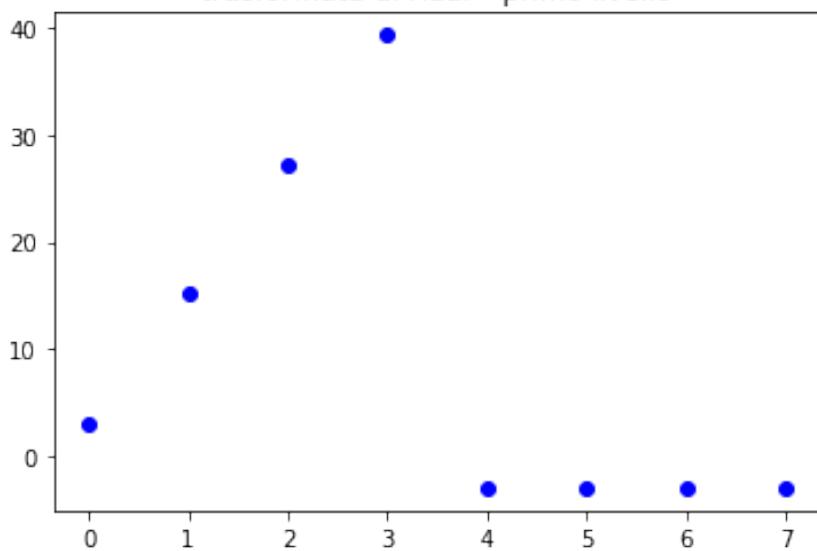


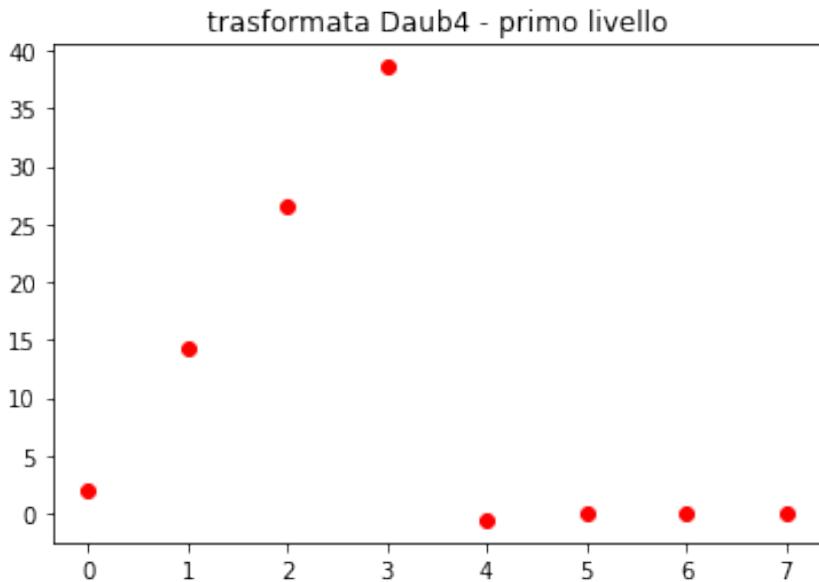
```
In [4]: u = np.linspace(0,30,8)
N = len(u)
plt.figure(21); plt.plot(u,'bo');
#
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(22); plt.plot(u_Haar1,'bo'); plt.title('trasformata di Haar - primo live
#
scalnum = np.array([1.+np.sqrt(3), 3.+np.sqrt(3), 3.-np.sqrt(3), 1.-np.sqrt(3)]) /
wavenum = np.array([1.-np.sqrt(3), -3.+np.sqrt(3), 3.+np.sqrt(3), -1.-np.sqrt(3)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Daub4 = np.concatenate((trend, flutt))
plt.figure(23); plt.plot(u_Daub4,'ro'); plt.title('trasformata Daub4 - primo livell
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOSito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```



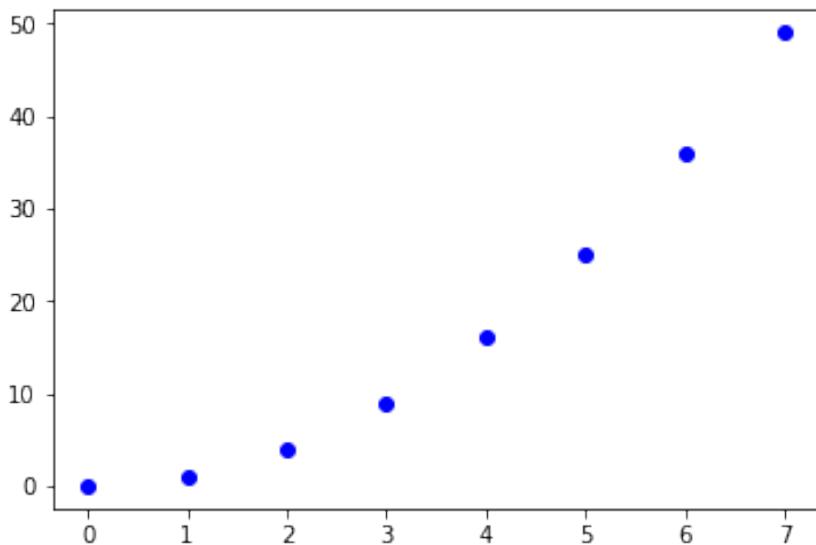
trasformata di Haar - primo livello



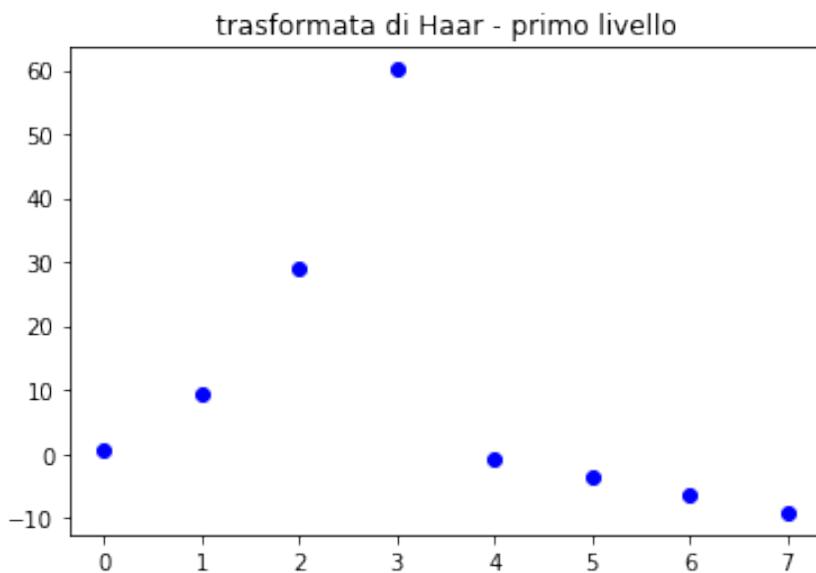


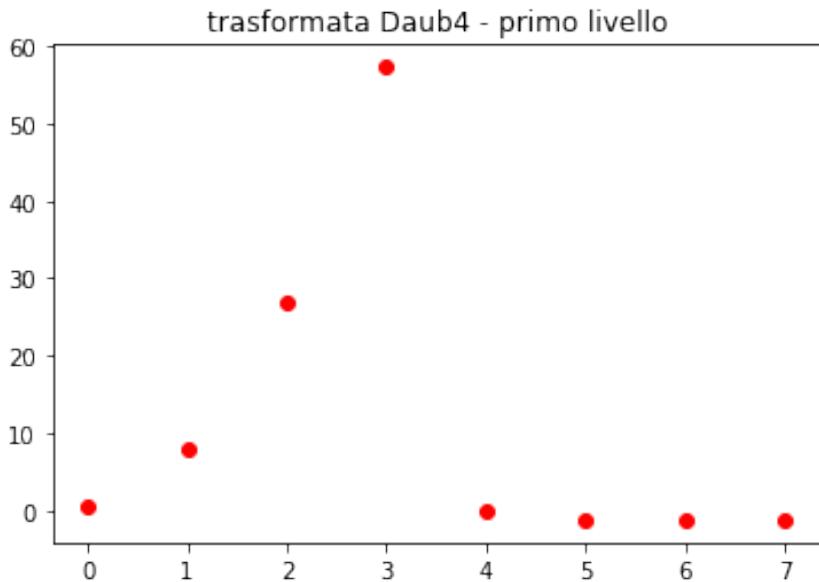
```
In [5]: u = np.arange(0.,8.)**2
N = len(u)
plt.figure(31); plt.plot(u,'bo'); plt.show()
#
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(32); plt.plot(u_Haar1,'bo'); plt.title('trasformata di Haar - primo live')
#
scalnum = np.array([1.+np.sqrt(3), 3.+np.sqrt(3), 3.-np.sqrt(3), 1.-np.sqrt(3)]) /
wavenum = np.array([1.-np.sqrt(3), -3.+np.sqrt(3), 3.+np.sqrt(3), -1.-np.sqrt(3)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
```

```
u_Daub4 = np.concatenate((trend, flutt))
plt.figure(33); plt.plot(u_Daub4, 'ro'); plt.title('trasformata Daub4 - primo livello')
```



```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOP/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

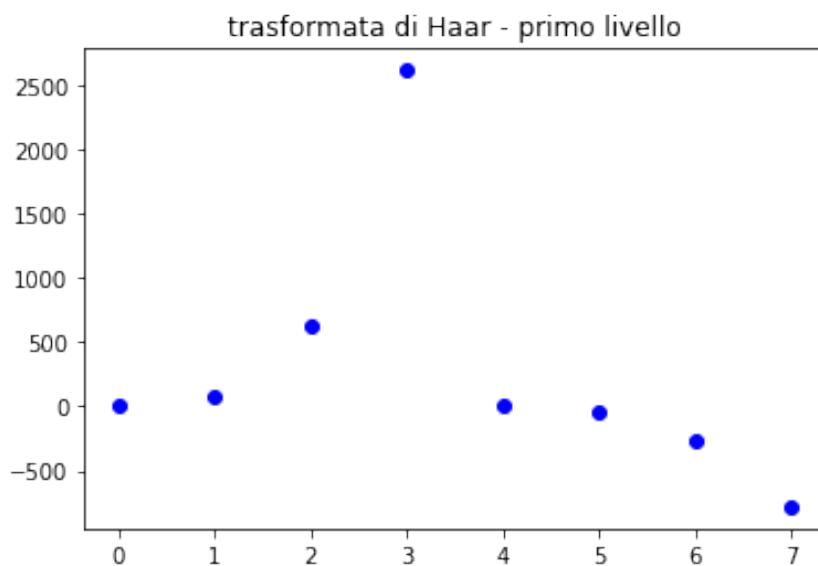
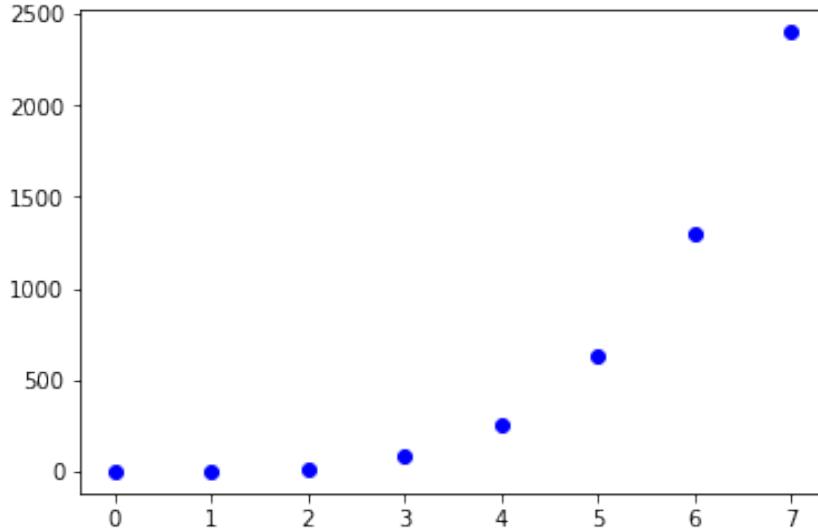




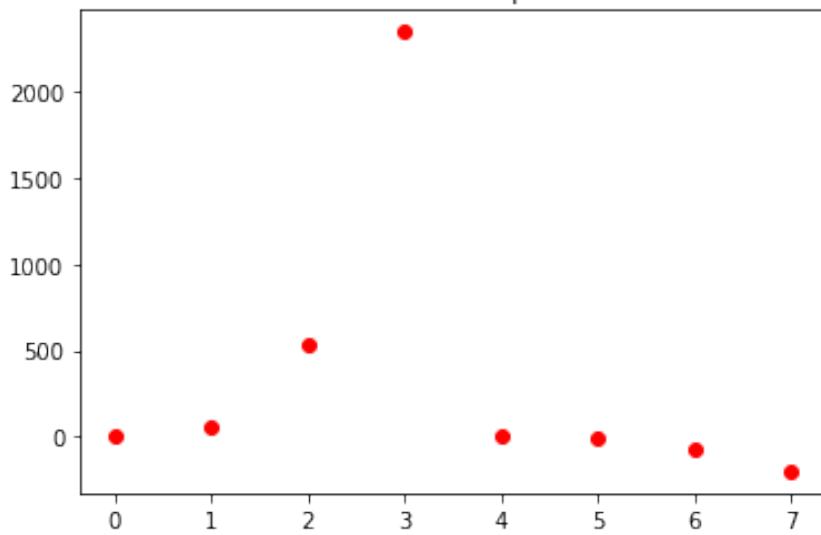
```
In [6]: u = np.arange(0.,8.)**4
N = len(u)
plt.figure(41); plt.plot(u,'bo');
#
scalnum = np.array([1/np.sqrt(2), 1/np.sqrt(2)])
wavenum = np.array([-1/np.sqrt(2), 1/np.sqrt(2)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
u_Haar1 = np.concatenate((trend, flutt))
plt.figure(42); plt.plot(u_Haar1,'bo'); plt.title('trasformata di Haar - primo live
#
scalnum = np.array([1.+np.sqrt(3), 3.+np.sqrt(3), 3.-np.sqrt(3), 1.-np.sqrt(3)]) /
wavenum = np.array([1.-np.sqrt(3), -3.+np.sqrt(3), 3.+np.sqrt(3), -1.-np.sqrt(3)])
Nc = len(scalnum)
temp = simula_DLTI(scalnum,np.array([1.]),u)
trend = temp[1:N+1:2]
temp = simula_DLTI(wavenum,np.array([1.]),u)
flutt = temp[1:N+1:2]
```

```
u_Daub4 = np.concatenate((trend, flutt))
plt.figure(43); plt.plot(u_Daub4, 'ro'); plt.title('trasformata Daub4 - primo livello')

/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPUBBLICATI/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```



trasformata Daub4 - primo livello



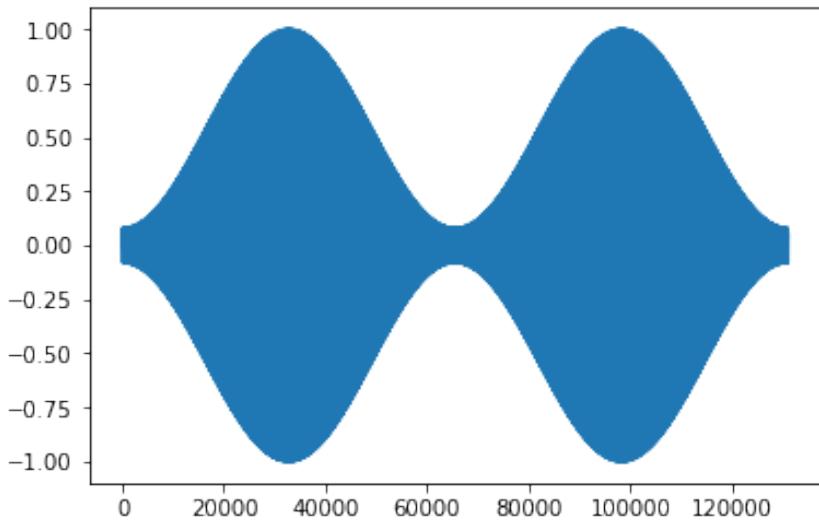
Torna al par. 8.2.2

Appendice R

Esempio analisi segnali audio con wavelets

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dou
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.hamming import *
from libreria_NLALD.trasformata_wavelet import *
from scipy.io import wavfile
```

```
In [2]: # compressione (lossy) di segnali audio
N = 2**16
u_ini = np.concatenate((hamming(N)*np.sin(2*np.pi*(1./164)*np.arange(N)), hamming(N)
N = len(u_ini)
plt.figure(1); plt.plot(u_ini);
```



```
In [3]: def audio_processing(u, wavelet_type='Haar', compress=1, Nlevels=5, write_files=False):
    # 'compress': 0=perfect reconstruction, 1=compression, 2=denoising
    N = len(u)
    C = 30000.0 / np.max(u)
    for i in range(1,Nlevels+1):
        u[0:int(N/2**i-1)] = trasformata_wavelet(u[0:int(N/2**i-1)],wavelet_type)
        if compress==1:
            # compressione del segnale: azzero tutti i coefficienti delle fluttuazioni
            u[int(N/2**i):int(N/2**i-1)] = np.zeros(int(N/2**i))
        elif compress==2:
            u[int(N/2**i):int(N/2**i-1)] = np.where(np.abs(u[int(N/2**i):int(N/2**i-1)])>C, 1, 0)
        #endif
    plt.figure(2*i-1); plt.plot(u); plt.title(wavelet_type+' '+livello +'lev'+str(i)); plt.show()
    ttemp = u[0:int(N/2**i-1)]
    # eseguo la trasformata inversa
    for j in range(1,i+1):
        ttemp = trasformata_wavelet(ttemp, wavelet_type, -1)
        if j < i:
            ttemp = np.concatenate((ttemp, u[int(N/2**i-j):int(N/2**i-j-1)]))
        #endif
    #endfor
    print("reconstruction error = ", np.sum(np.abs(u_ini-ttemp)))
    if write_files:
        Fc = 44100;
        wavarray = np.vstack((ttemp, ttemp)).T
        wavfile.write(wavelet_type+"_lev"+str(i)+"_ann"+".wav", Fc, np.asarray(C*ttemp))
```

```

#endif
if analyze==True:
    plt.figure(2*i); plt.plot(u_ini[4000:4200], 'b-'); plt.plot(ttemp[4000:4200], 'r-')
#endif
#endiffor

```

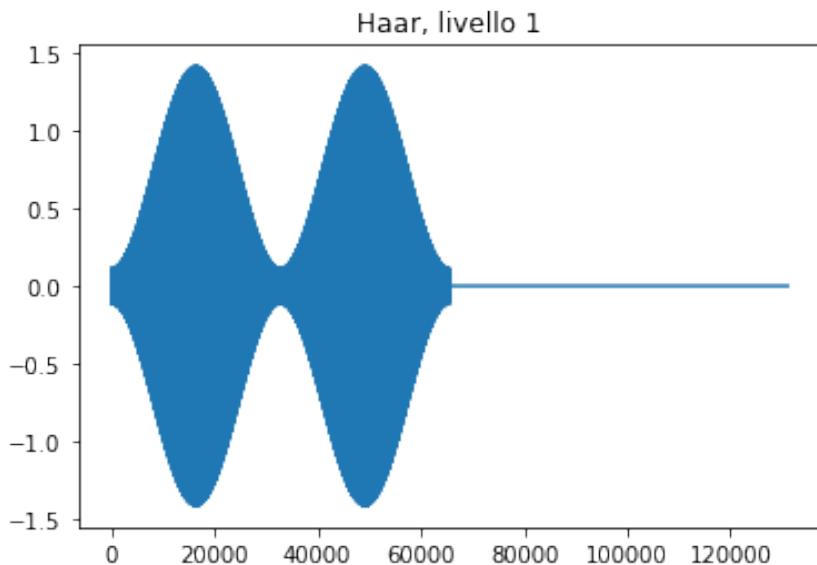
```

In [4]: u = u_ini.copy()
# compressione con wavelets di Haar:
compress = 1 # 0=no compressione 1=compressione
write_files = False
audio_processing(u, 'Haar', compress, 4, write_files, "compression", 1.0, True)

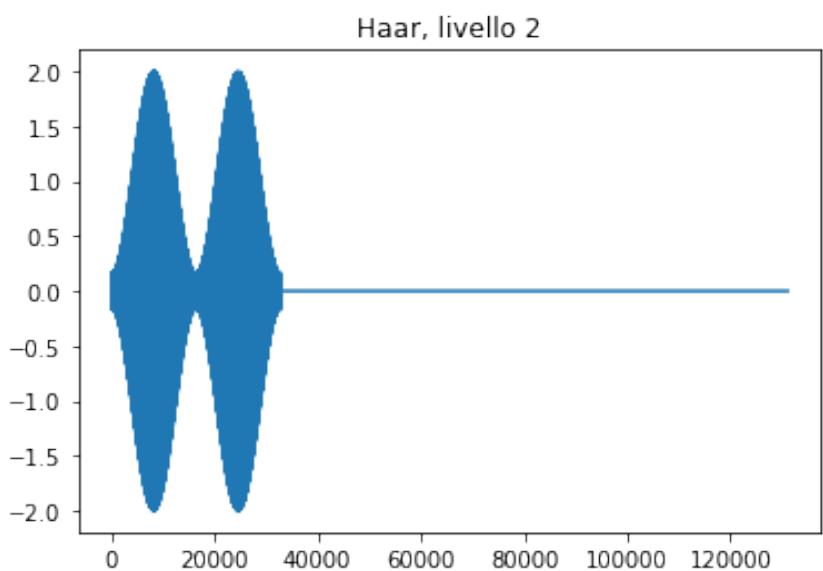
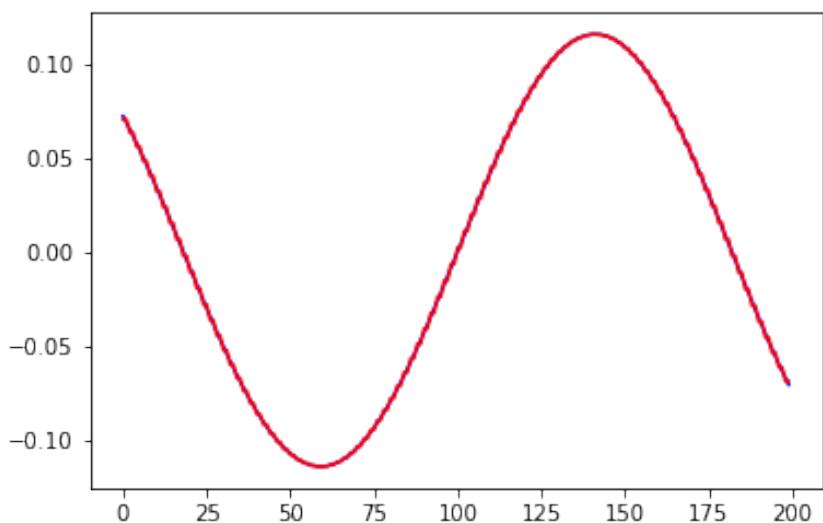
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPUBBLICAZIONI/BOOKS/BOOKS_MNAD/sito_web_MNAD
```

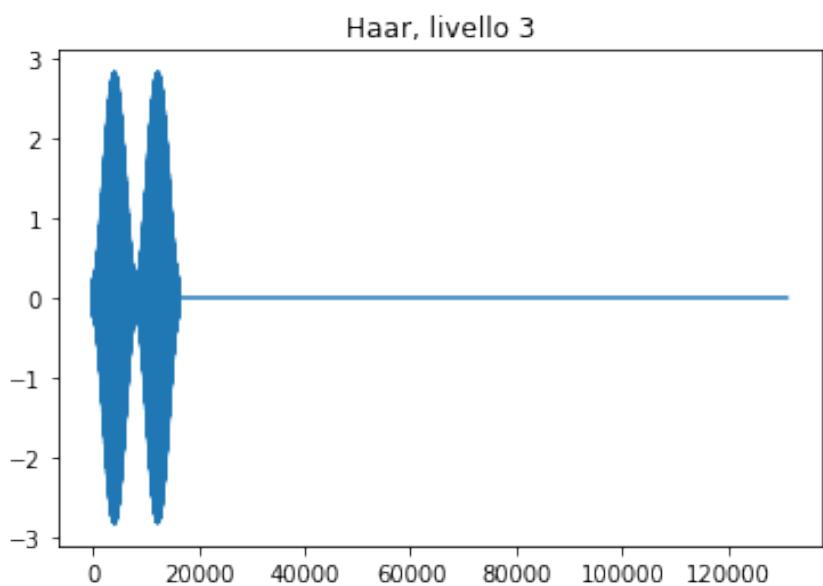
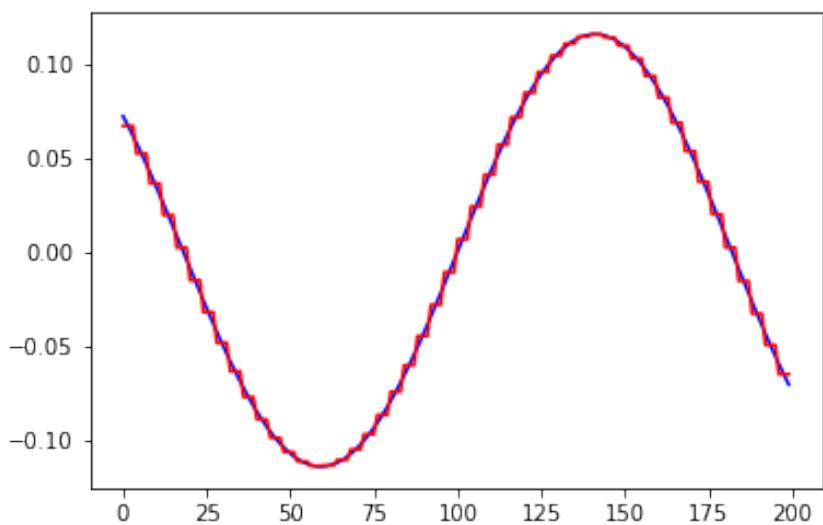
```
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```



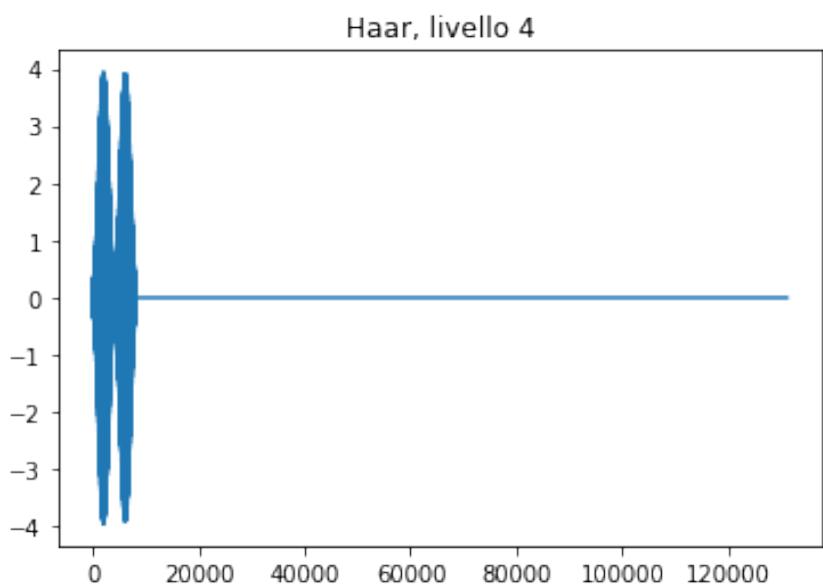
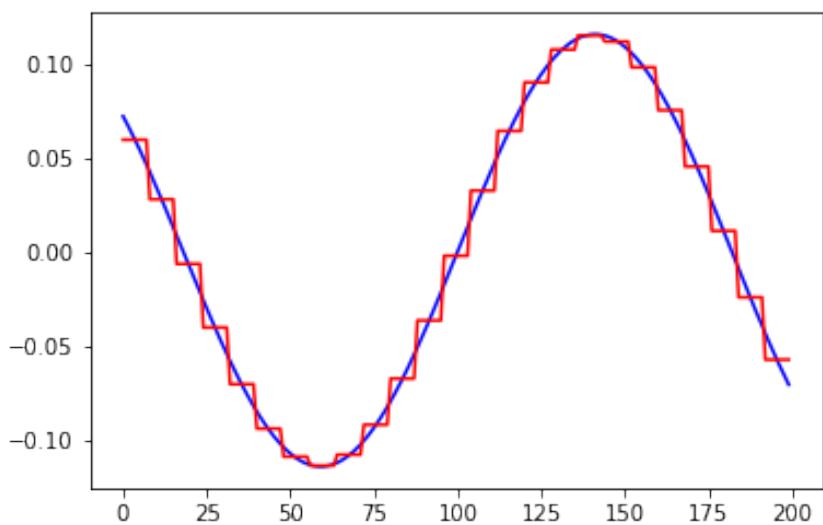
```
reconstruction error = 937.146101686212
```



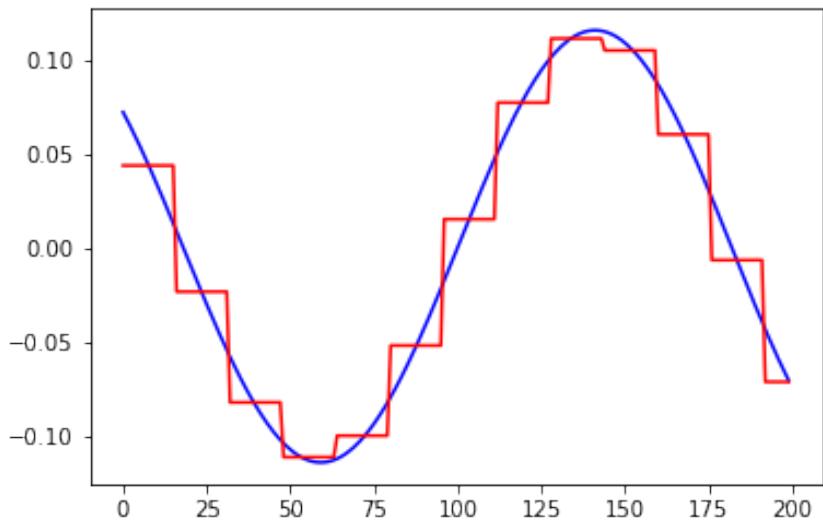
reconstruction error = 1873.8789248174191



reconstruction error = 3747.482593317246



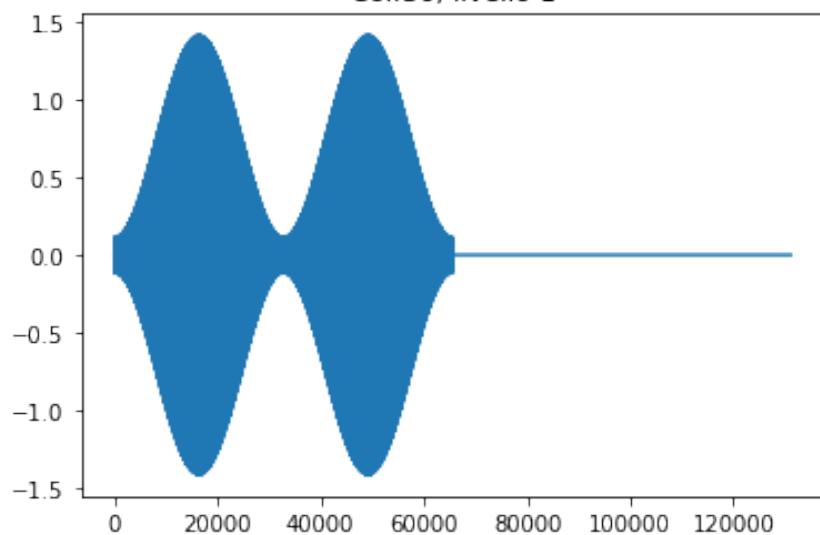
reconstruction error = 7496.582055274537



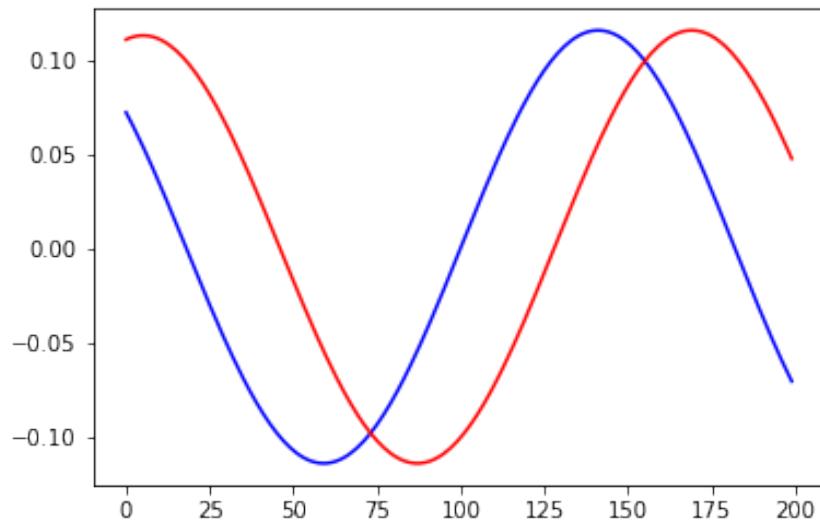
```
In [5]: u = u_ini.copy()
        # compressione con wavelets di tipo Coiflets:
compress = 1 # 0=no compressione 1=compressione
write_files = False
audio_processing(u, 'Coif30', compress, 4, write_files, "compression", 1.0, True)
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

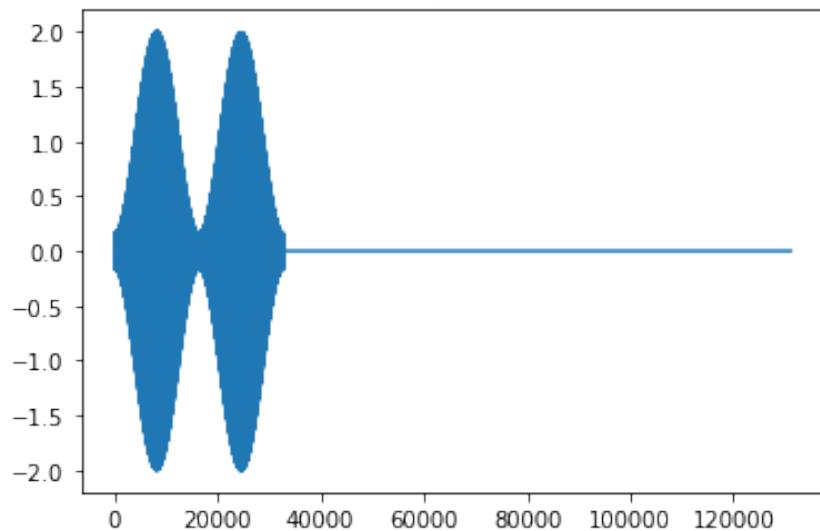
Coif30, livello 1



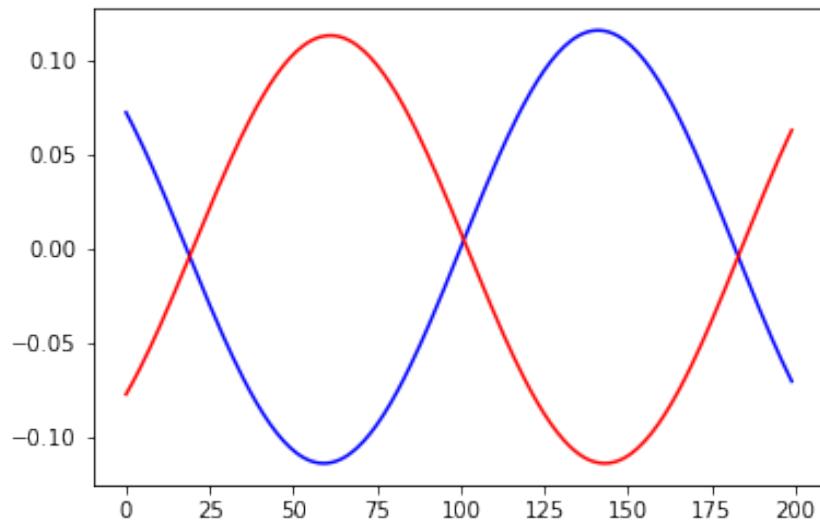
reconstruction error = 49503.480270110565



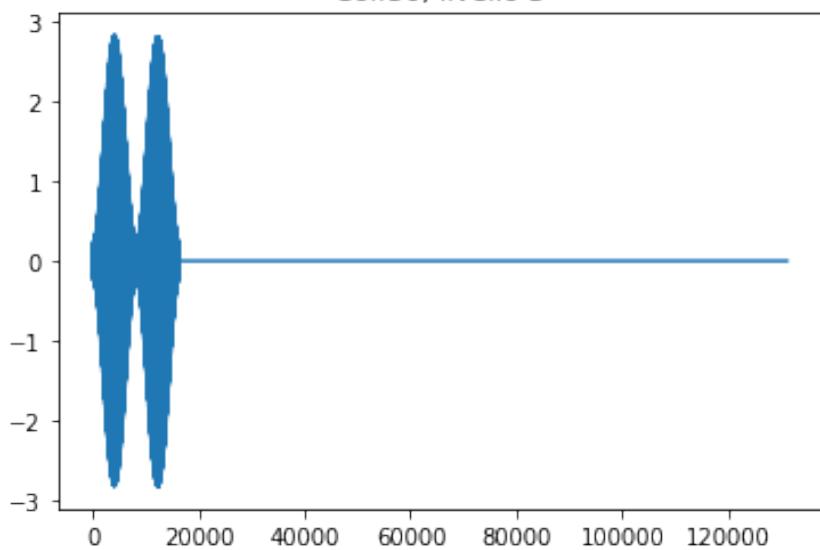
Coif30, livello 2



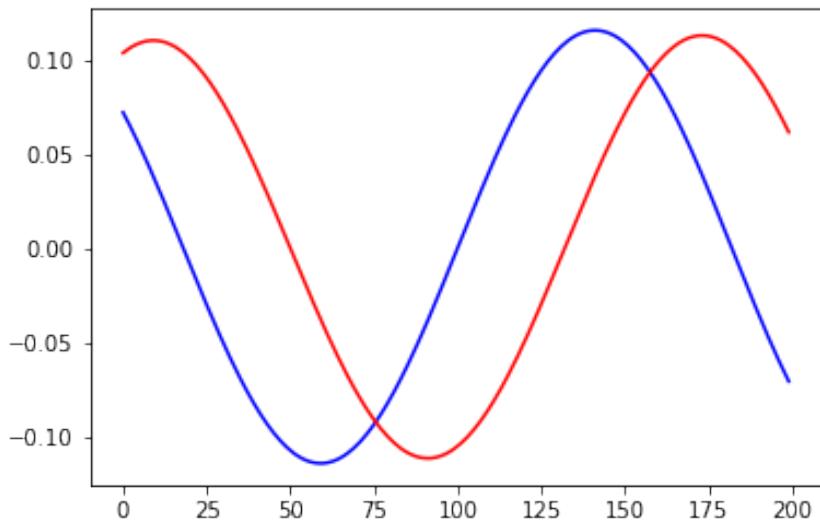
reconstruction error = 87854.90788703768



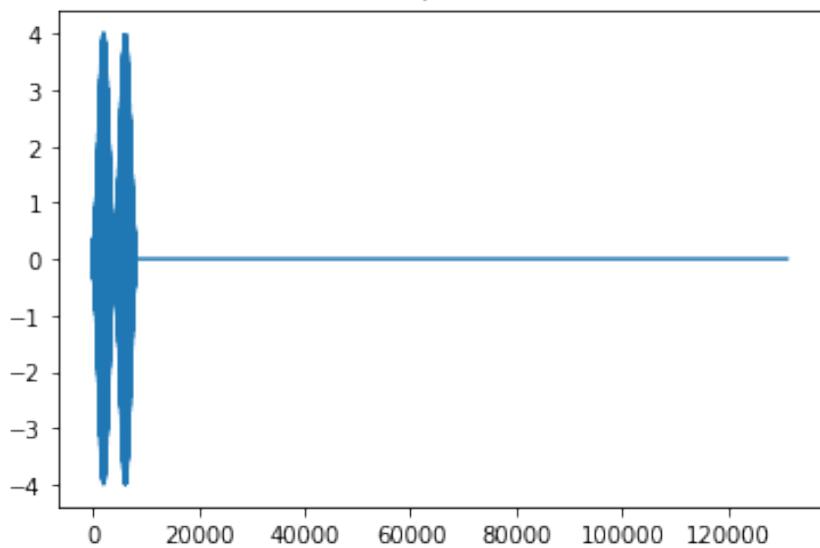
Coif30, livello 3



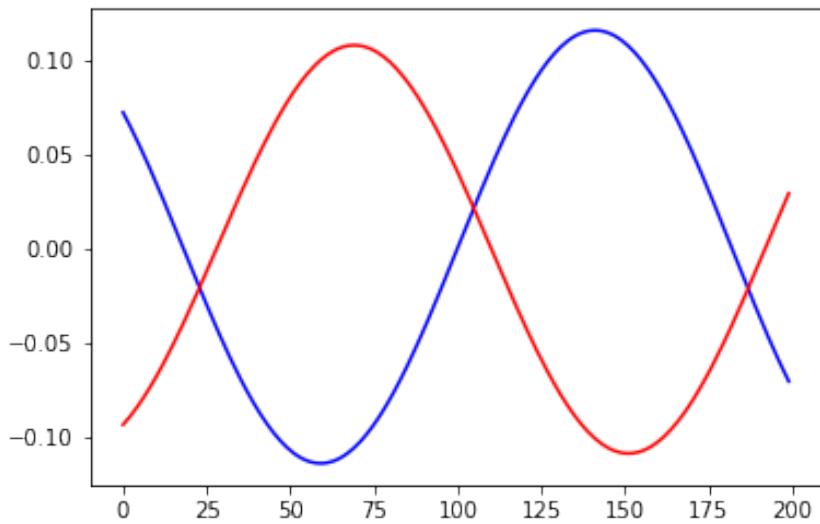
reconstruction error = 68764.73262487129



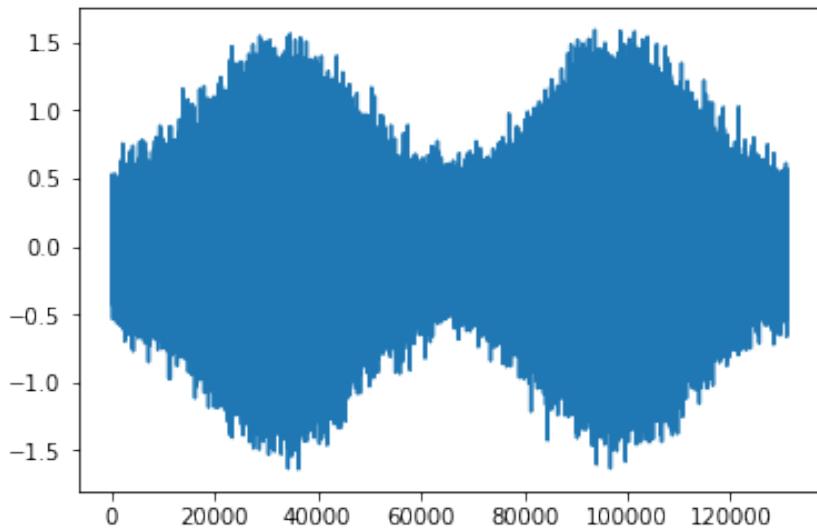
Coif30, livello 4



reconstruction error = 44721.55487651983



```
In [6]: # rimozione del rumore (de-noising)
sigma = 0.2;
n = sigma * np.random.randn(N);
un = u_ini + n;
soglia = 5*sigma;
plt.figure(103); plt.plot(un);
```



```
In [7]: u = un.copy()
# denoising con wavelets di Haar:
operation = 2; # 0=no compressione 1=compressione 2=denoising
write_files = False
analyze = True
thresh = 1.0
#audio_processing(u, 'Haar', operation, 5, write_files, "denoising", thresh, analyze)

In [8]: u = un.copy()
# denoising con wavelets Coiflets:
operation = 2; # 0=no compressione 1=compressione 2=denoising
write_files = False
analyze = True
thresh = 1.0
#audio_processing(u, 'Coif30', operation, 5, write_files, "denoising", thresh, analyze)
```

Torna al par. 8.2.3

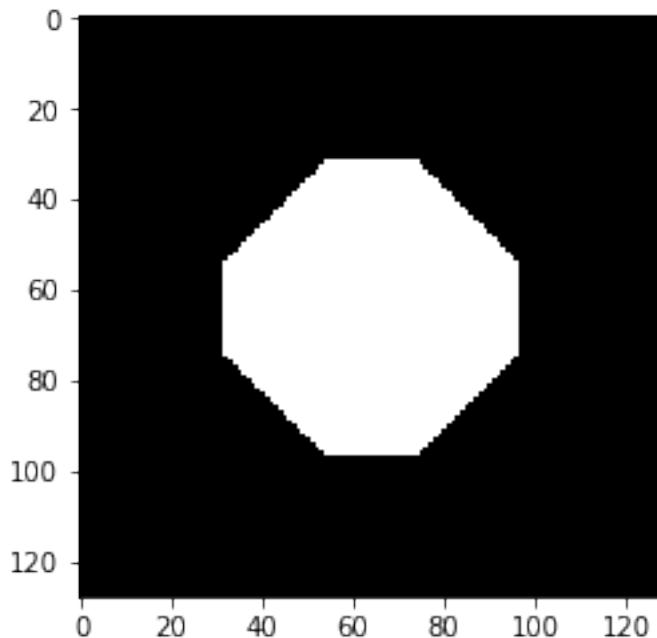
Appendice S

Esempi wavelets 2-d

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dou
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#%get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import PIL.Image as Image
from numpy.fft import *
from libreria_NLALD.trasformata_wavelet import *
```

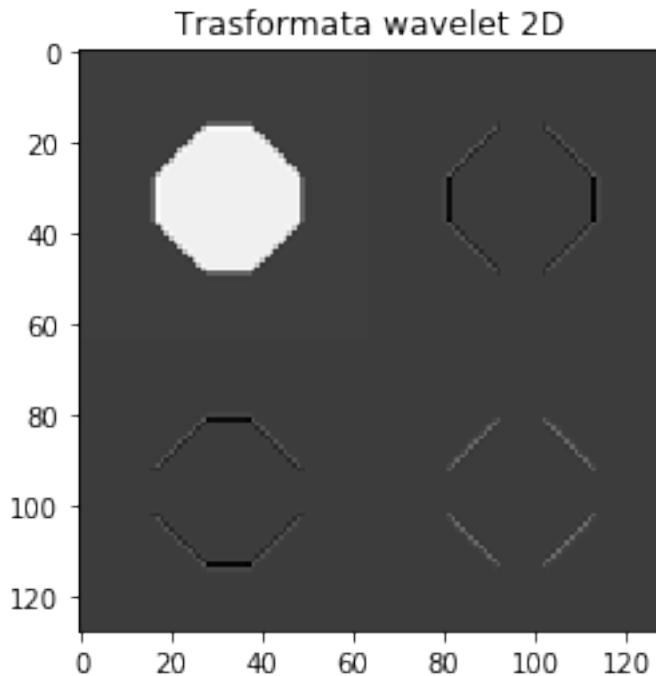
```
In [2]: # Data una sequenza bi-dimensionale, la trasformata wavelet puo' essere prodotta in
# utilizzando l'algoritmo 1D ed agendo al primo passo sulle righe (o sulle colonne)
# sulle colonne (o sulle righe). La sequenza trasformata possiede dunque una suddiv
# del tipo:          ThTv      FhTv
#                  ThFv      FhFv
# in cui ogni blocco è dunque una combinazione di trend e fluttuazioni orizzontali
#
# I livelli successivi della trasformata procedono considerando la sequenza trend "
```

```
In [3]: # costruiamo un'immagine a ottagono:
Nx = 128
Ny = Nx
otta = np.ones((Nx,Ny))
for x in range(int(Nx/2-Nx/4),int(Nx/2+Nx/4+1)):
    for y in range(int(Ny/2-Ny/4),int(Ny/2+Ny/4+1)):
        if ((x-Nx/4+y-Ny/4 > Nx/6) and (3*Nx/4-x+y-Ny/4 > Nx/6) and (x-Nx/4+3*Ny/4-y >
            otta[x,y] = 255
        #endif
    #endfor
#endfor
plt.figure(1); plt.imshow(otta,'gray'); plt.show()
```

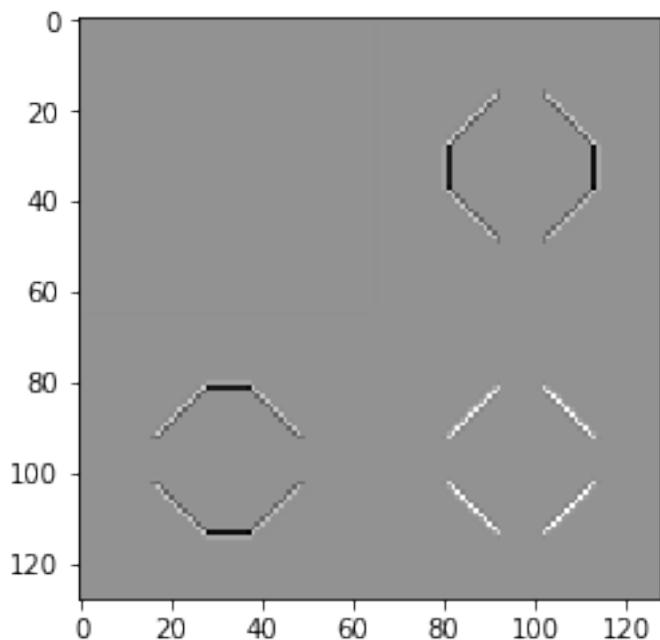


```
In [4]: # calcolo del primo livello della trasformata wavelet 2D
x = otta.copy()
x = trasformata_wavelet(x, 'Coif06')
plt.figure(2); plt.imshow(x,'gray'); plt.title('Trasformata wavelet 2D'); plt.show()
```

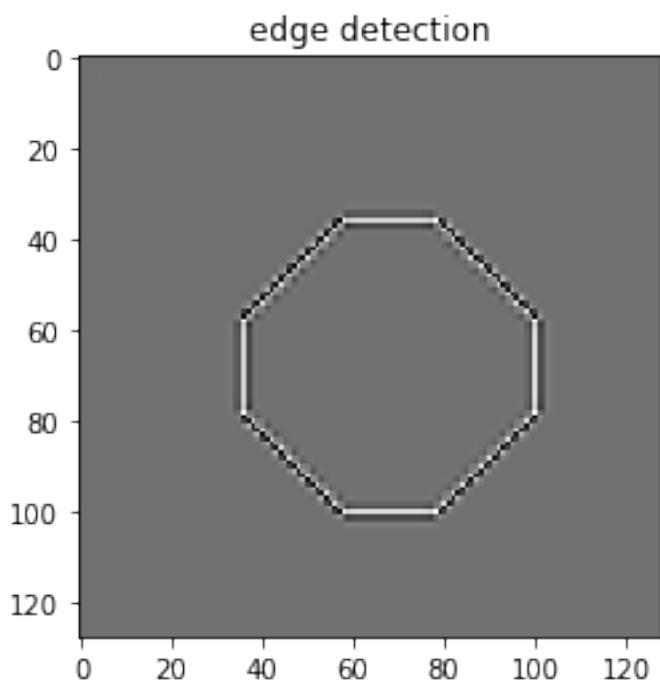
```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIO/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```



```
In [5]: # ora, azzeriamo la sotto-seguenza trend "ThTv" e calcoliamo la trasformata inversa
trasf_w = x.copy()      # memorizzo la trasformata wavelet
x[0:int(Nx/2),0:int(Ny/2)] = np.zeros((int(Nx/2),int(Ny/2)))
plt.figure(3); plt.imshow(x,'gray'); plt.show();
x = trasformata_wavelet(x,'Coif06',-1)
plt.figure(4); plt.imshow(x,'gray'); plt.title('edge detection'); plt.show()
# -> si puo' notare come le fluttuazioni mettano bene in evidenza i bordi (edges) !
```



```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPROPRIO/libro_MNAD/sito_web_MNAD  
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```



Torna al par. [8.2.7](#)

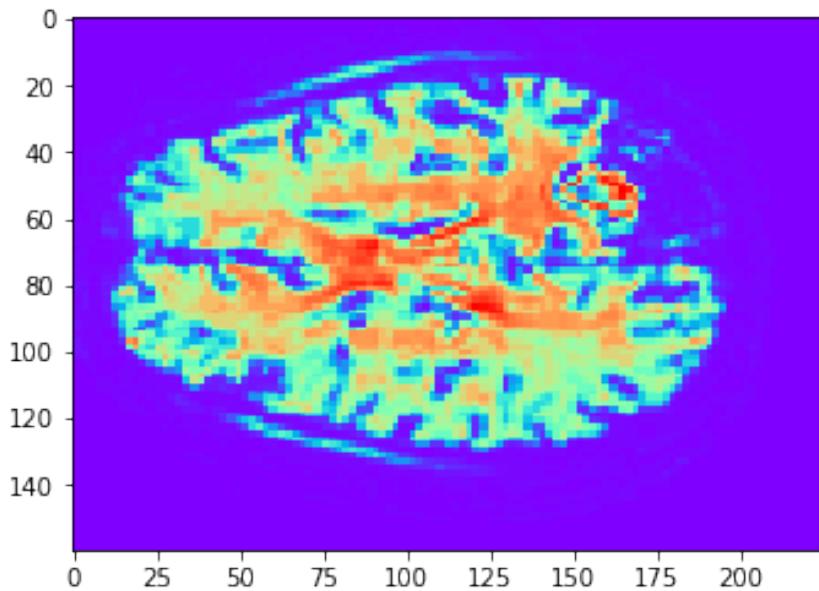
Appendice T

Esempio MRI

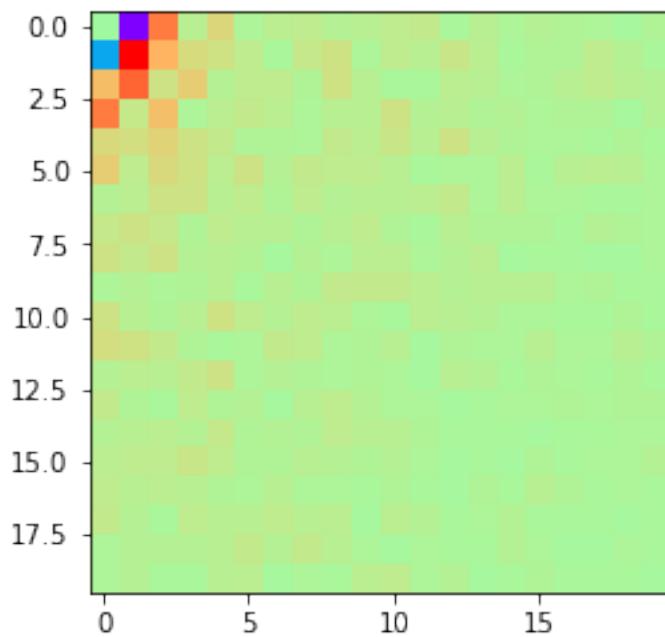
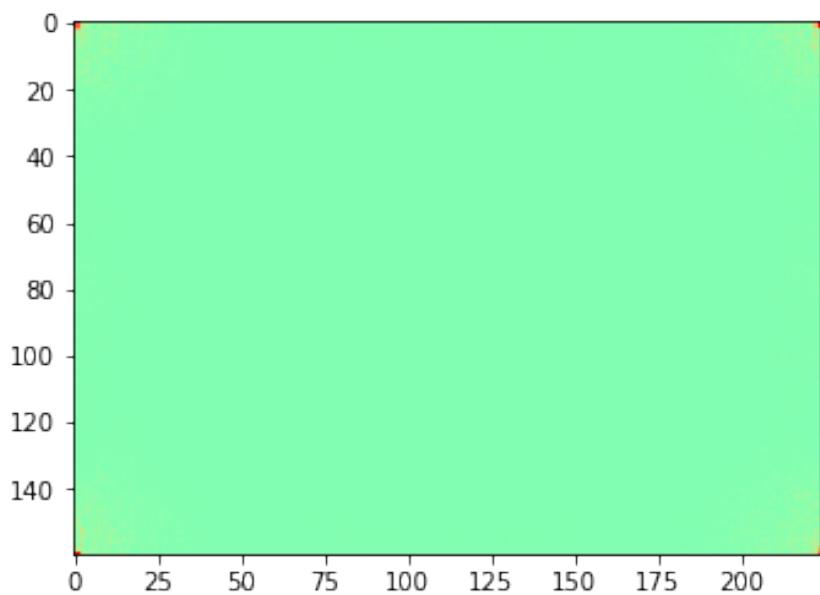
```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow  
%matplotlib inline  
import numpy as np  
import matplotlib.pyplot as plt  
import PIL.Image as Image  
from numpy.fft import *  
from libreria_NLALD.trasformata_wavelet import *
```

```
In [2]: im = Image.open("../MRI.png")
im = im.convert("L")
print(im.format, im.size, im.mode)
#im.save("prova.png", "L")
X = np.reshape(im.getdata(), (im.size[1],im.size[0]))
X = X[:,0:224]
Nx,Ny = X.shape
print("X.shape =", X.shape)
plt.figure(1); plt.imshow(X,cmap='rainbow');
```

None (225, 160) L
X.shape = (160, 224)



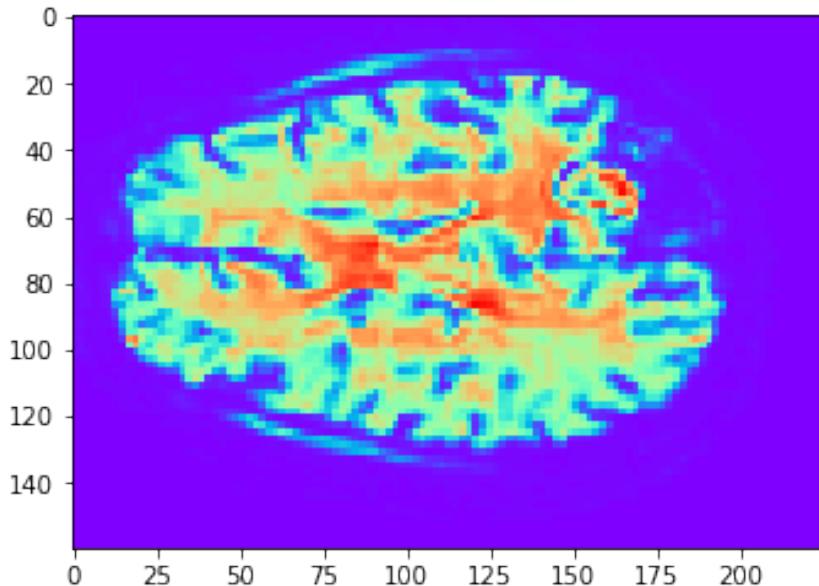
```
In [3]: # "mFX" sono le misure sperimentali:  
mFX = np.abs(fft2(X))  
mFX = 2* 255. / np.max(mFX) * mFX  
mFX = np.asarray(mFX,dtype=np.int8)  
plt.figure(11); plt.imshow(mFX,cmap='rainbow'); plt.show()  
plt.figure(12); plt.imshow(mFX[0:20,0:20],cmap='rainbow'); plt.show()
```

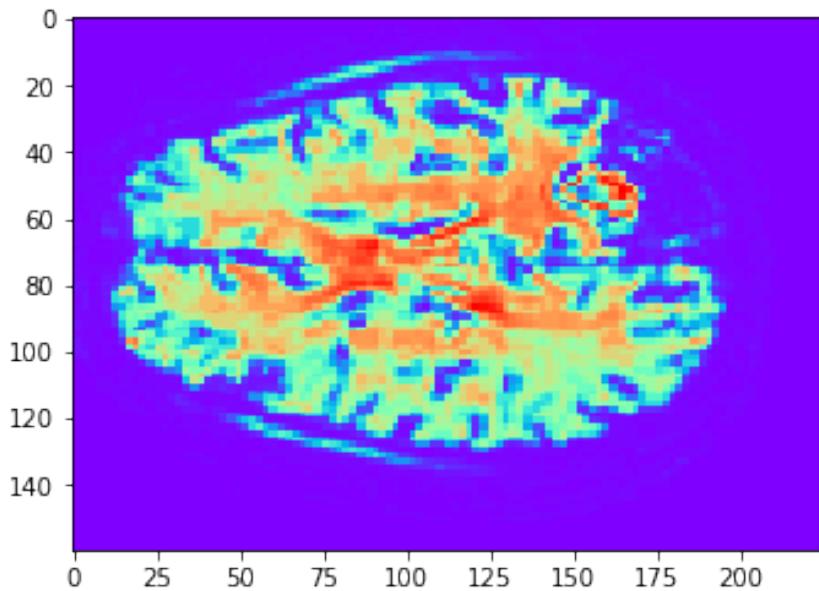


```
In [4]: mWX = trasformata_wavelet(X.copy(), 'Haar');
# metto a zero le fluttuazioni:
Nxh = int(Nx/2)
Nyh = int(Ny/2)
mWX[0:Nxh,Nyh:Ny] = np.zeros((Nxh,Nyh))
mWX[Nxh:Nx,0:Nyh] = np.zeros((Nxh,Nyh))
mWX[Nxh:Nx,Nyh:Ny] = np.zeros((Nxh,Nyh))
#print mWX.shape
XrecW = trasformata_wavelet(np.asarray(mWX), 'Haar', -1);
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPUBBLICAZIONI/libro_MNAD/sito_web_MNAD
if na==1: a=np.array([a, 0.]); na += 1; #endif # serve per non inserire "if" n
```

```
In [5]: plt.figure(21); plt.imshow(XrecW,cmap='rainbow'); plt.show()
plt.figure(22); plt.imshow(X,cmap='rainbow'); plt.show()
```





```
In [6]: print(Nx,Ny)
      print(Nx*Ny)
```

```
160 224
35840
```

```
In [7]: I3 = np.where(abs(XrecW[:]) > np.min(abs(X)))
      print(len(I3[0]))
```

```
19708
```

```
In [8]: print(np.min(abs(X)))
```

```
16
```

```
In [9]: print(np.min(abs(X-XrecW)))
```

```
0
```

Torna al par. [8.2.7](#)

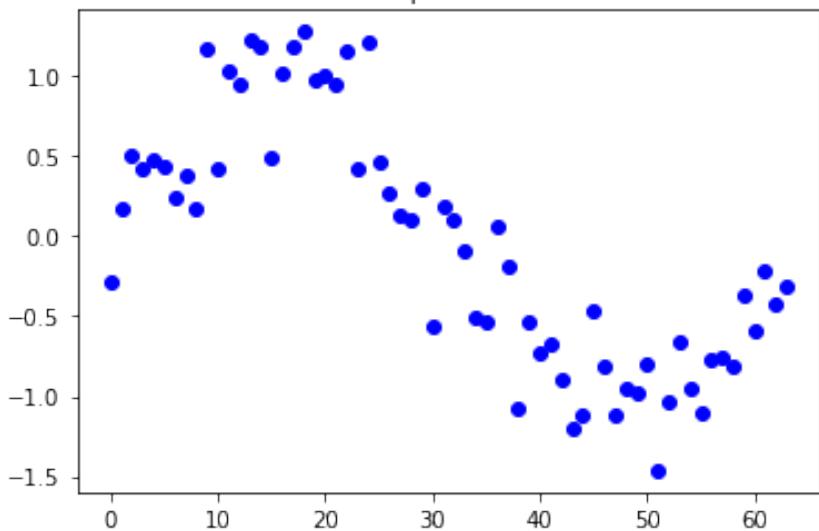
Appendice U

Esempio scelta variabili modello lineare standard

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.parsimonia_LS import *
from libreria_NLALD.validation_set import *
```

```
In [2]: N = 64
maxgp = 12 #N-1
stderr = 0.3
vk = np.arange(0.,1.,1./N)
b = np.sin(2*np.pi*vk)
br = b + stderr*np.random.randn(N);
plt.figure(1), plt.plot(br,'bo'); plt.title('dati sperimentali'); plt.show()
br = np.atleast_2d(br).T
```

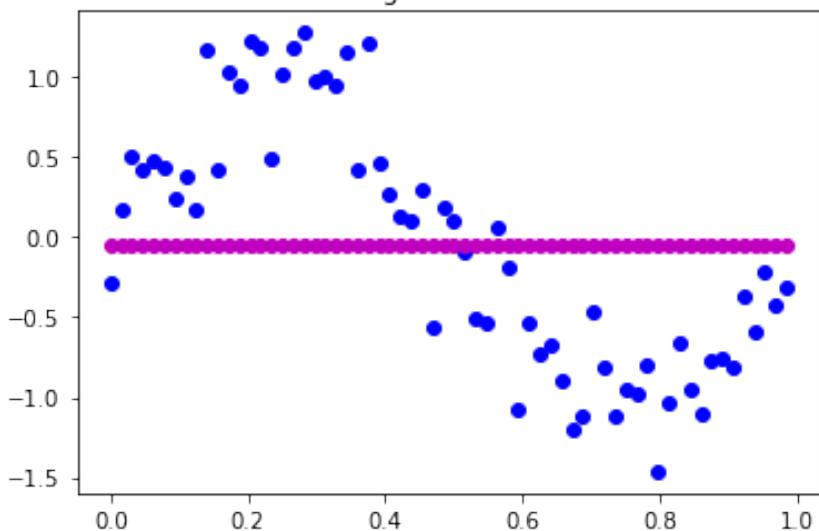
dati sperimentali



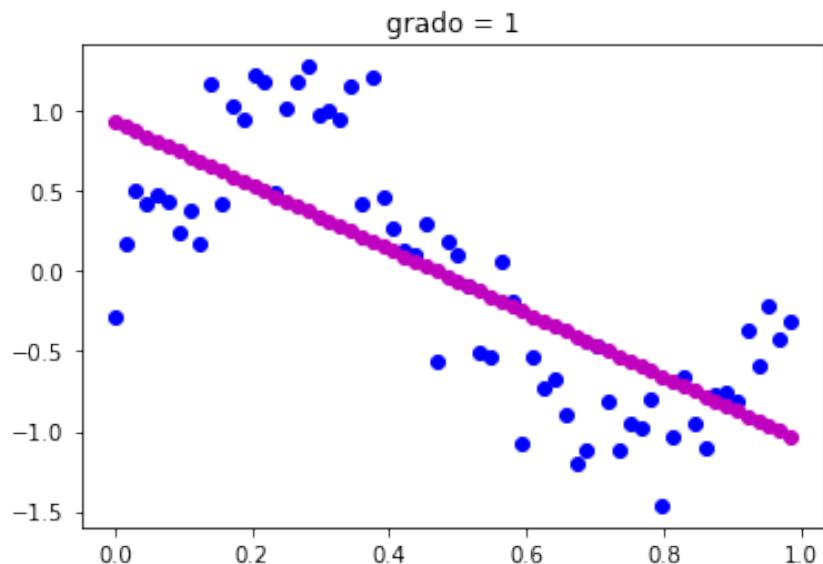
```
In [3]: parsimonia_LS(vk,br,maxgp,stderr)
```

```
# aumentando il grado, la stima della varianza dell'errore di predizione (estvar_errpred)
# in sostanza il residuo scalato, scende (cfr. Figura) e al limite (interpolazione)
# pero' la covarianza dell'errore di stima dei parametri aumenta vistosamente.
# Notare che al grado 3 (che è quello giusto per approssimare un periodo di
# una sinusoide) si ha l'avvicinamento di estvar_errpred al valore vero di varianza
```

grado = 0

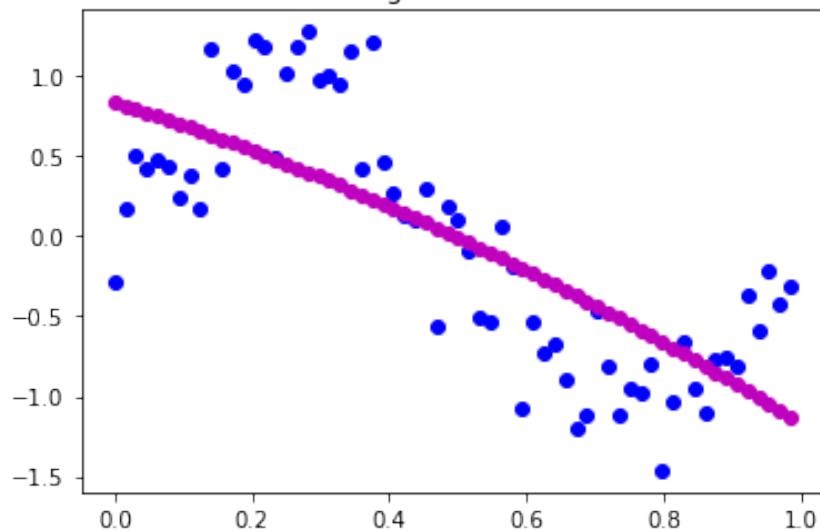


```
parametri stimati: [[-0.04508311]]  
varianza delle stime dei parametri: [0.00940167]  
valori singolari di [A b]: [8.01977515 6.14172911]
```



```
parametri stimati: [[-1.99536151  0.93700888]]  
varianza delle stime dei parametri: [0.05045055 0.01642476]  
valori singolari di [A b]: [9.06401553 6.24003698 1.33383974]
```

grado = 2

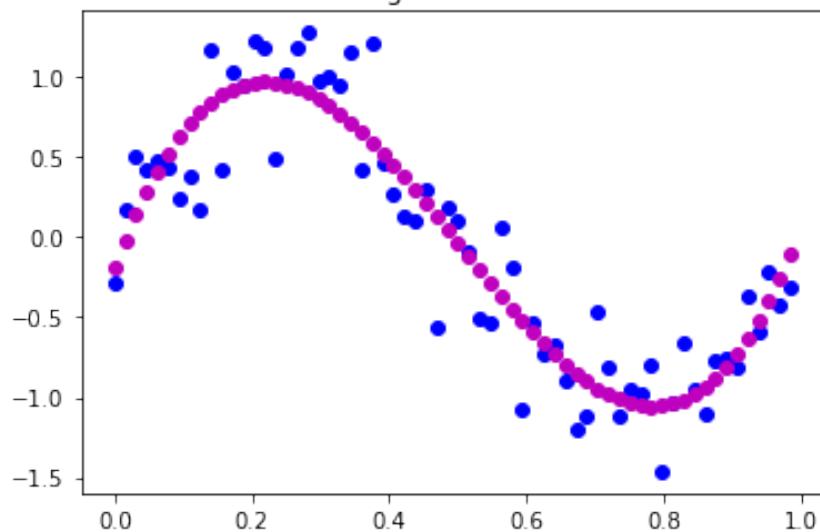


parametri stimati: [[-0.62740929 -1.37775549 0.83729124]]

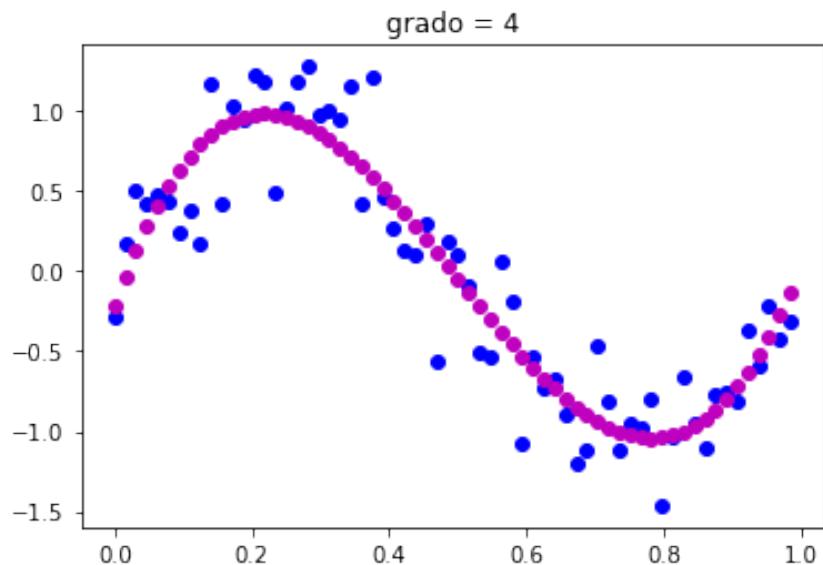
varianza delle stime dei parametri: [0.76346279 0.79063878 0.03583955]

valori singolari di [A b]: [9.57985529 6.33240853 1.76970492 0.4170072]

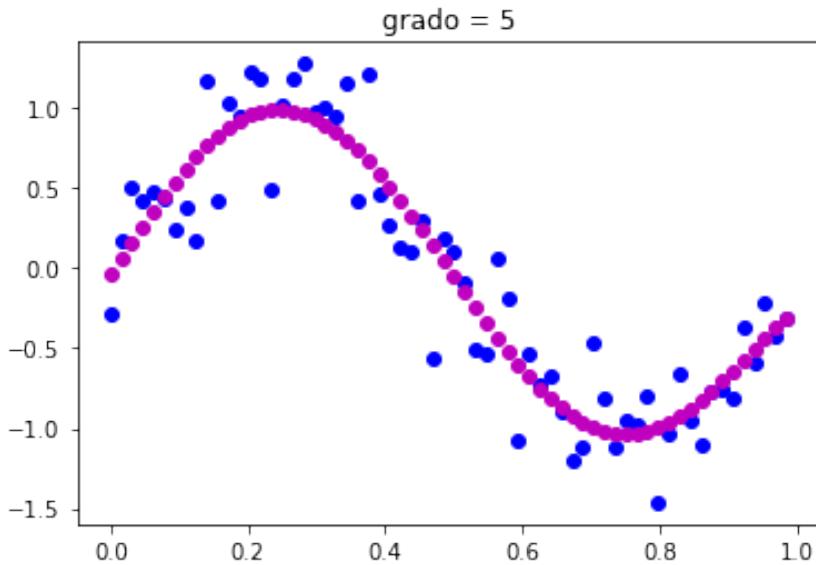
grado = 3



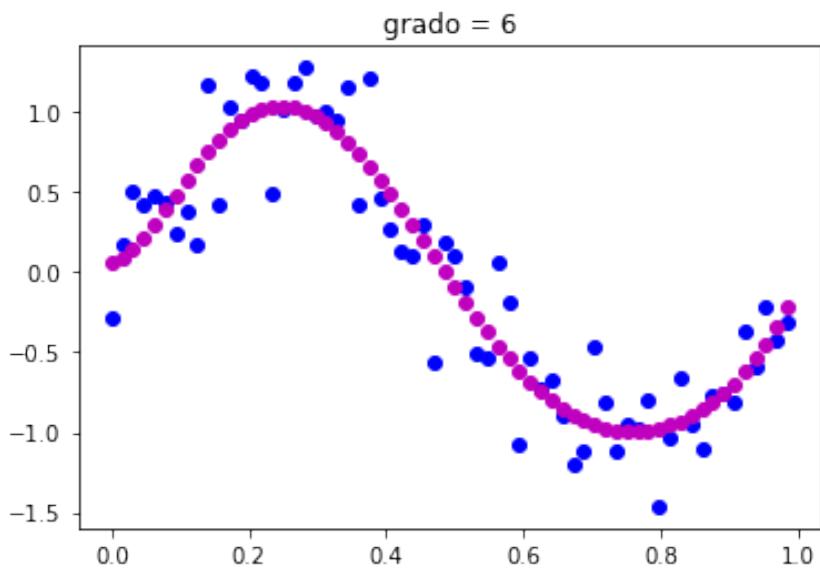
```
parametri stimati: [[ 22.61971344 -34.02682992  11.66999767 -0.19067595]]  
varianza delle stime dei parametri: [3.57308696 8.01936949 1.4262398 0.01813865]  
valori singolari di [A b]: [9.89934351 6.39775666 2.12271789 0.62715622 0.0435468]
```



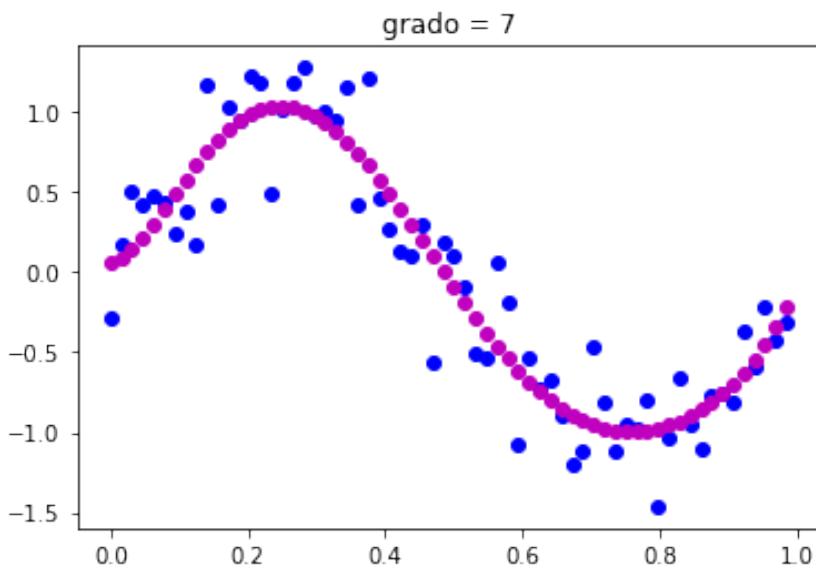
```
parametri stimati: [[ -2.40452869  27.3536293 -37.0070802   12.31011318 -0.2199  
varianza delle stime dei parametri: [5.73563875e+01 2.25939464e+02 9.62518148e+01  
2.69137183e-02]  
valori singolari di [A b]: [10.12000696 6.44289947 2.43755314 0.75960067 0.08
```



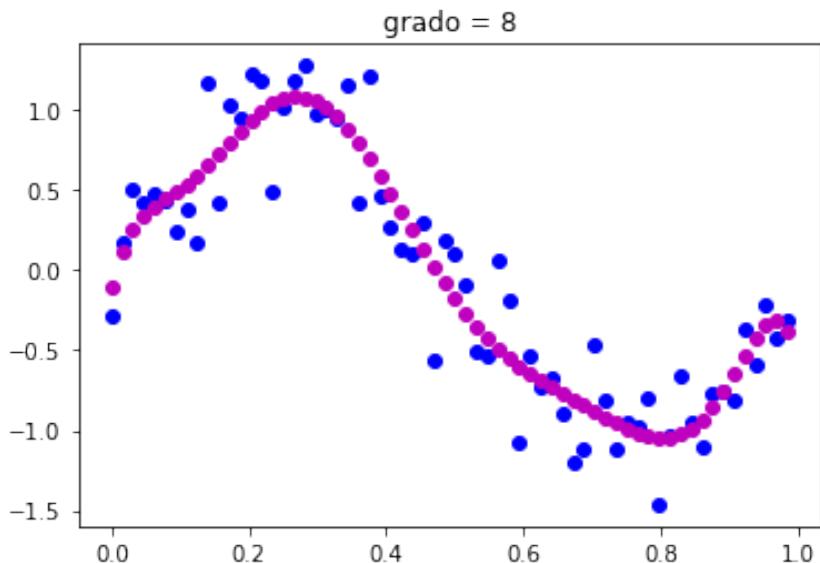
```
parametri stimati: [-5.89418430e+01  1.42647663e+02 -9.90873686e+01  9.13623180e+00
 6.01994263e+00 -3.62125069e-02]
varianza delle stime dei parametri: [8.69965870e+02 5.32328581e+03 4.21844481e+03
 1.51543999e+01 3.40675717e-02]
valori singolari di [A b]: [1.02829782e+01 6.47501473e+00 2.71733649e+00 8.463223e-01
 1.26024847e-01 2.53825291e-02 2.61425550e-03]
```



```
parametri stimati: [[ 1.28602269e+02 -4.38720420e+02  5.66222254e+02 -3.19661152e+02
   6.24114710e+01  1.01232103e+00  6.28684397e-02]]
varianza delle stime dei parametri: [1.38998631e+04 1.22087101e+05 1.56096099e+05
 3.00815976e+03 3.61803825e+01 4.22071841e-02]
valori singolari di [A b]: [1.04090147e+01 6.49878169e+00 2.96380595e+00 9.069936e+00
 1.76157143e-01 3.56085901e-02 5.56559903e-03 4.74063534e-04]
```

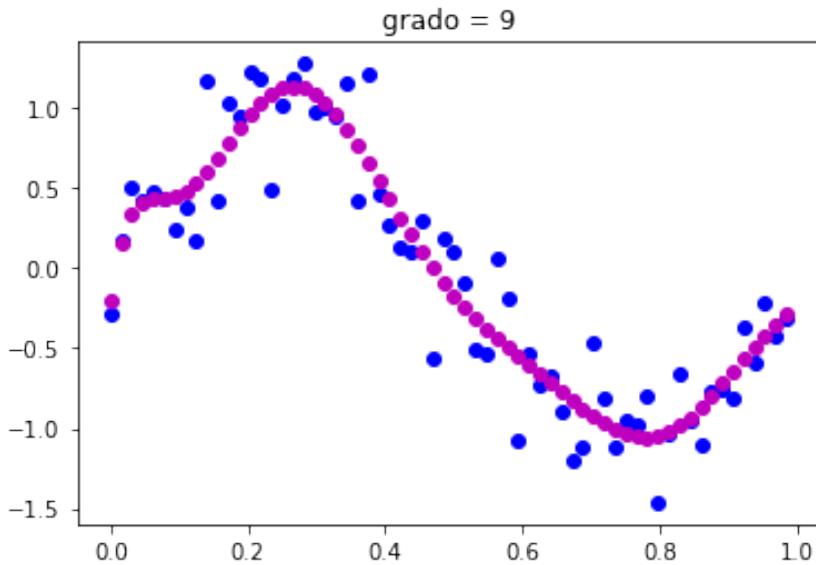


```
parametri stimati: [[ 1.88706596e+01  6.35869501e+01 -3.50315413e+02  5.06161716e-02
-2.98438826e+02  5.87506343e+01  1.26288843e+00  5.93822729e-02]]
varianza delle stime dei parametri: [2.27934740e+05 2.71977366e+06 5.12679821e+06
3.34184233e+05 1.16400093e+04 7.70124257e+01 5.07388452e-02]
valori singolari di [A b]: [1.05098261e+01 6.51705591e+00 3.18044004e+00 9.525897e-01
2.29205041e-01 4.51735791e-02 9.09533972e-03 1.06128481e-03
8.58495377e-05]
```



```

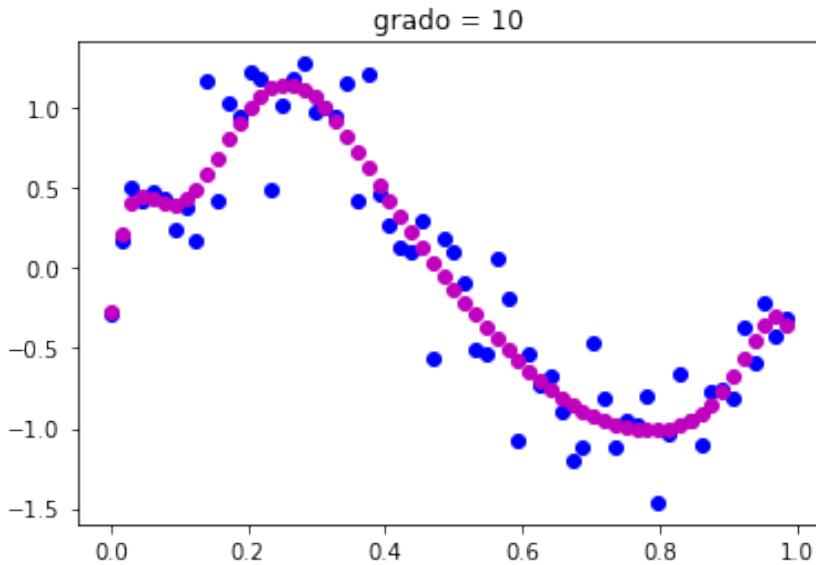
parametri stimati: [[-3.89095198e+03  1.53394941e+04 -2.45212259e+04  2.02921300e
-9.16944248e+03  2.19590204e+03 -2.64384122e+02  1.80063720e+01
-1.08341161e-01]]
varianza delle stime dei parametri: [3.48221753e+06 5.42030561e+07 1.41586602e+08
2.38614271e+07 1.74638765e+06 3.50000408e+04 1.37149994e+02
5.43471933e-02]
valori singolari di [A b]: [1.05925566e+01 6.53159319e+00 3.37127732e+00 9.892134
2.83040933e-01 5.44658426e-02 1.29948050e-02 1.82193585e-03
2.04205305e-04 1.46549429e-05]
```



```

parametri stimati: [[ 9.71480855e+03 -4.69245180e+04  9.49492473e+04 -1.04205095e
 6.70059142e+04 -2.53776267e+04  5.39312835e+03 -5.86878099e+02
 3.10335287e+01 -2.03602883e-01]]
varianza delle stime dei parametri: [5.59708603e+07 1.10170957e+09 3.81215436e+09
 1.39574237e+09 1.79368161e+08 7.78734389e+06 9.62494803e+04
 2.36108896e+02 5.90595122e-02]
valori singolari di [A b]: [1.06618301e+01 6.54350247e+00 3.54013149e+00 1.020376
 3.36143773e-01 6.37653264e-02 1.70702446e-02 2.74106667e-03
 3.66751150e-04 3.73798378e-05 2.61061862e-06]

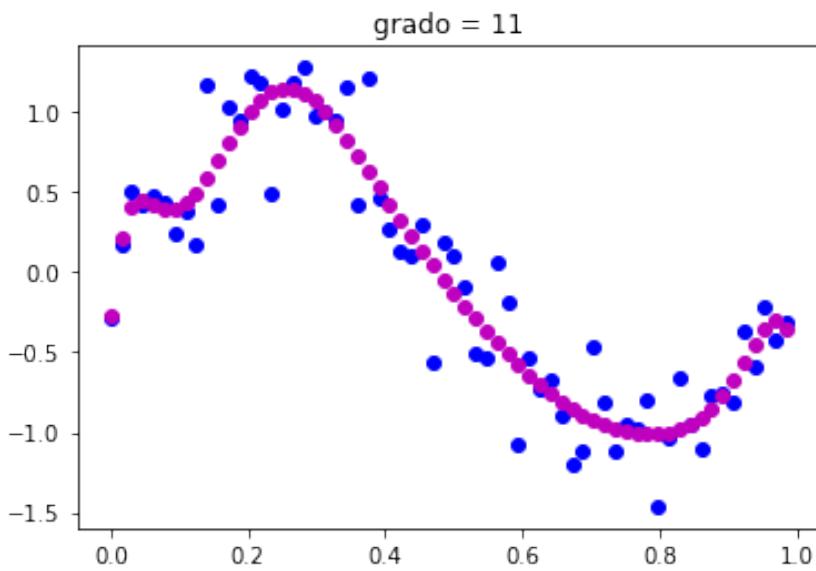
```



```

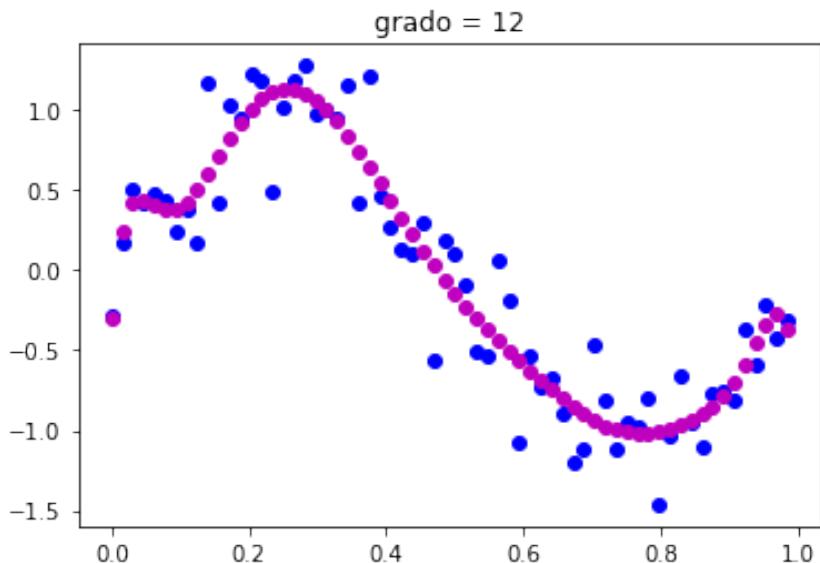
parametri stimati: [[-3.26047010e+04  1.70191071e+05 -3.83191919e+05  4.85998788e
-3.79952664e+05  1.87938802e+05 -5.78994304e+04  1.04724059e+04
-9.96917551e+02  4.42502446e+01 -2.74592460e-01]]
varianza delle stime dei parametri: [9.13761191e+08 2.21922873e+10 9.82999972e+10
6.92623136e+10 1.39651834e+10 1.08822451e+09 2.99482346e+07
2.40550881e+05 3.85642323e+02 6.32161284e-02]
valori singolari di [A b]: [1.07207819e+01 6.55350631e+00 3.69031660e+00 1.048193
3.87486710e-01 7.32346635e-02 2.11748192e-02 3.80008457e-03
5.71476489e-04 7.15598493e-05 6.90715457e-06 4.60012541e-07]

```



```

parametri stimati: [[ 8.24801959e+03 -7.72599946e+04  2.74733253e+05 -5.21750728e
 6.00312267e+05 -4.40742677e+05  2.08798722e+05 -6.23924198e+04
 1.10426198e+04 -1.03457781e+03  4.52377209e+01 -2.78487423e-01]]
varianza delle stime dei parametri: [1.55052008e+10 4.55437817e+11 2.51373604e+12
3.11665748e+12 9.13013650e+11 1.13432502e+11 5.71136789e+09
1.04652239e+08 5.68528315e+05 6.15353533e+02 6.78848902e-02]
valori singolari di [A b]: [1.07716194e+01 6.56209009e+00 3.82461405e+00 1.073994
4.36400519e-01 8.29458646e-02 2.52143222e-02 4.97864901e-03
8.15768589e-04 1.17951362e-04 1.36916525e-05 1.26797375e-06
8.12839931e-08]
```

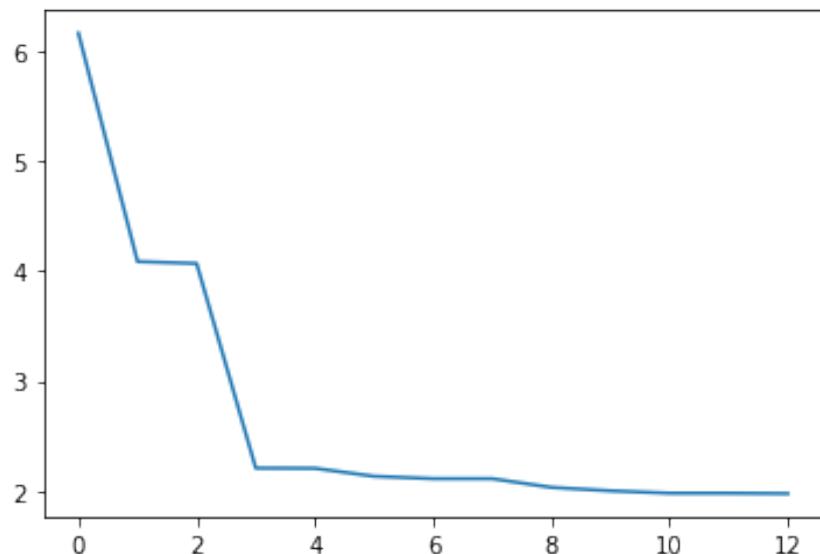


```

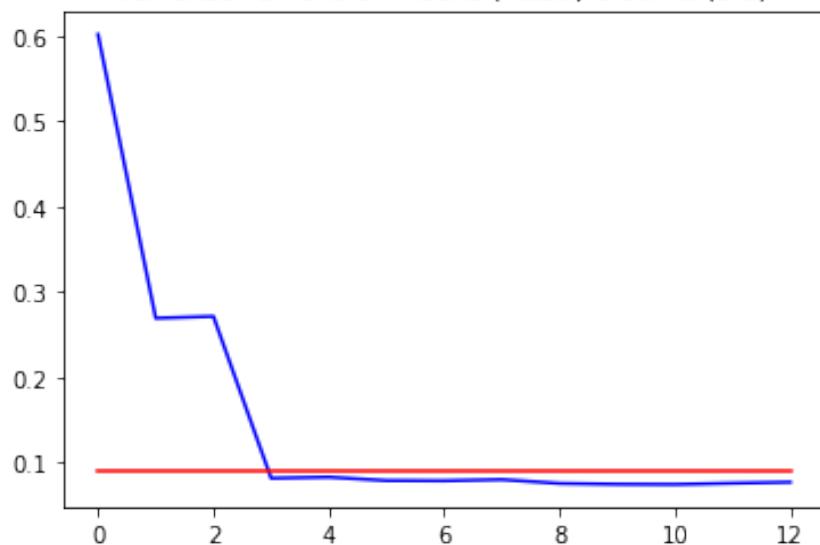
parametri stimati: [[-2.16986764e+05 1.28982610e+06 -3.39255232e+06 5.20863926e
-5.18716989e+06 3.52155427e+06 -1.66511258e+06 5.48184268e+05
-1.22415454e+05 1.73621796e+04 -1.38233688e+03 5.27803304e+01
-3.00206116e-01]]
varianza delle stime dei parametri: [-1.78849905e+11 -6.33787181e+12 -4.29051486e
-8.53052539e+13 -3.29885640e+13 -5.60868381e+12 -4.03277182e+11
-1.10076532e+10 -8.77462120e+07 -3.83916356e+04 3.18117785e+02
6.60985464e-02]
valori singolari di [A b]: [1.08159460e+01 6.56958921e+00 3.94532725e+00 1.098646
4.82468257e-01 9.29097468e-02 2.91392488e-02 6.25587487e-03
1.09709944e-03 1.76683517e-04 2.33511706e-05 2.60175526e-06
2.34473691e-07 1.40505275e-08]

```

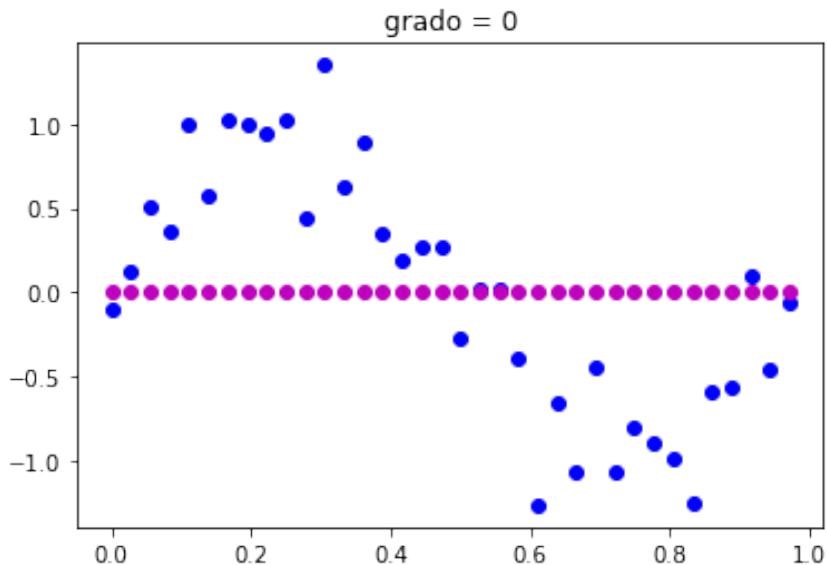
norma-2 del residuo



varianza rumore di misura (rosso) e stima (blu)

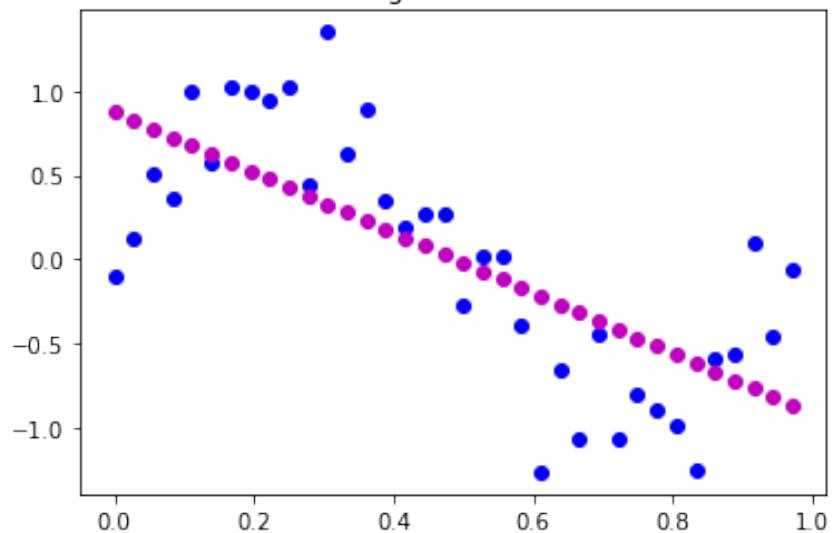


```
In [4]: # avvicinarsi all'interpolante è numericamente instabile:
N = 36
maxgp = N-1
vk = np.arange(0.,1.,1./N)
br = np.asmatrix( np.sin(2*np.pi*vk) + stderr*np.random.randn(N) ).T
parsimonia_LS(vk,br,maxgp,stderr)
```



```
parametri stimati: [[0.00546623]]
varianza delle stime dei parametri: [0.0148953]
valori singolari di [A b]: [6.00018725 4.3320793 ]
```

grado = 1

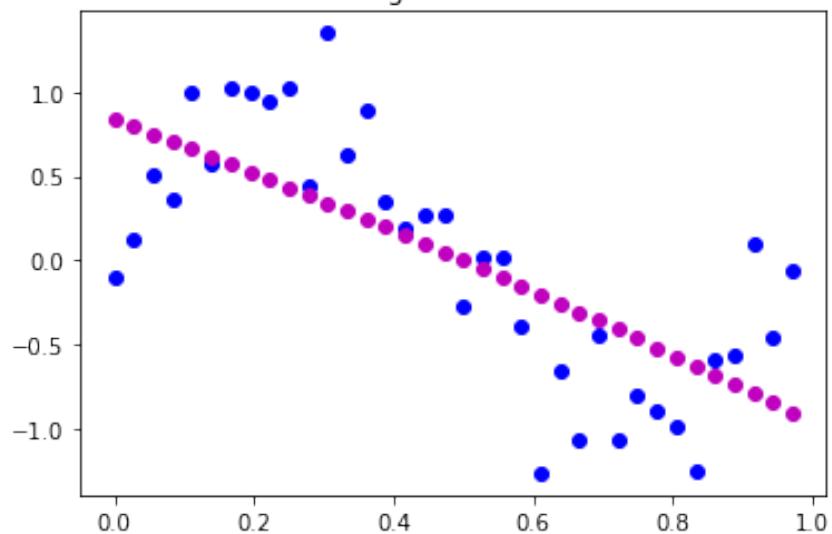


parametri stimati: [[-1.78556821 0.87345078]]

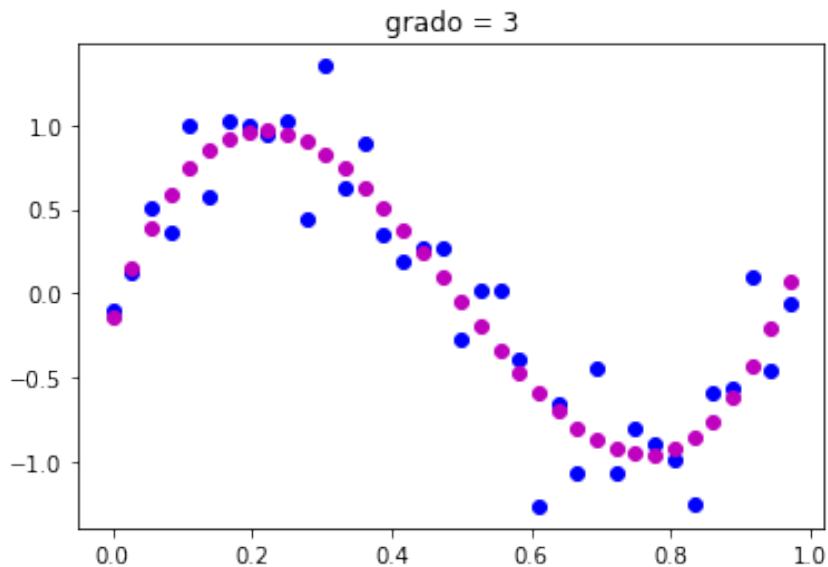
varianza delle stime dei parametri: [0.09037072 0.02888005]

valori singolari di [A b]: [6.73109679 4.45662938 1.0509892]

grado = 2

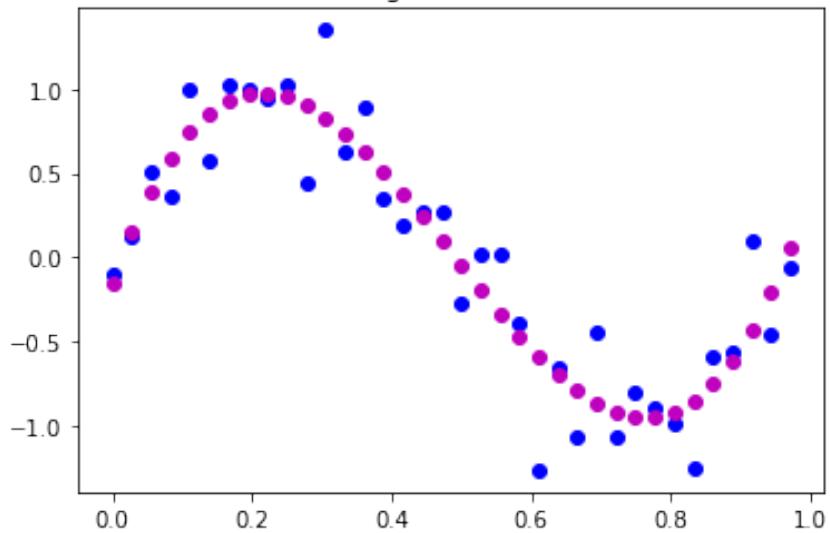


```
parametri stimati: [[-0.24889315 -1.54358876  0.83536142]]  
varianza delle stime dei parametri: [1.39908519 1.41542224 0.06248149]  
valori singolari di [A b]: [7.08823916 4.55028369 1.39164238 0.31330773]
```



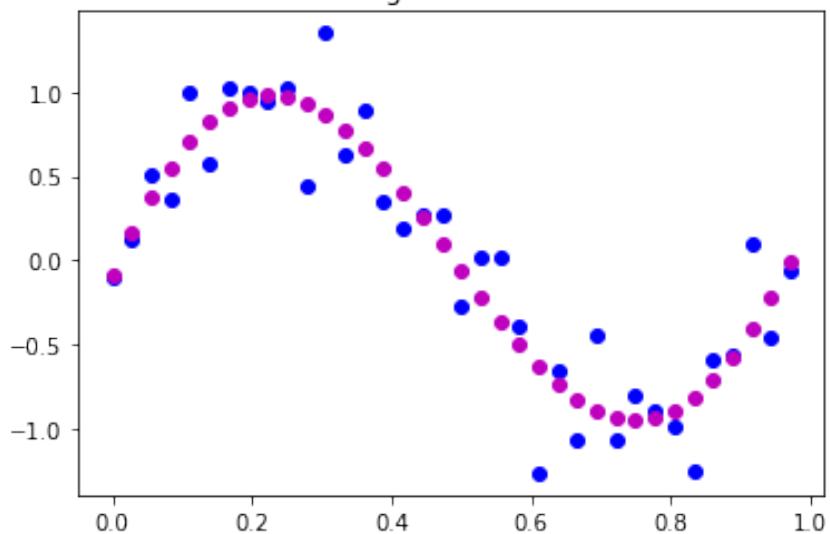
```
parametri stimati: [[ 23.10583287 -33.94489942  11.37677936 -0.13703876]]  
varianza delle stime dei parametri: [ 5.91687215 12.96135456  2.23225453  0.02734  
valori singolari di [A b]: [7.30748553 4.61186081 1.65929793 0.46577202 0.0311558]
```

grado = 4



parametri stimati: [[-1.14476316 25.33176124 -35.32348601 11.66508027 -0.1492
varianza delle stime dei parametri: [9.73570766e+01 3.74198973e+02 1.54563784e+02
3.93400933e-02]
valori singolari di [A b]: [7.45762157 4.65274048 1.89093916 0.56128826 0.0580830

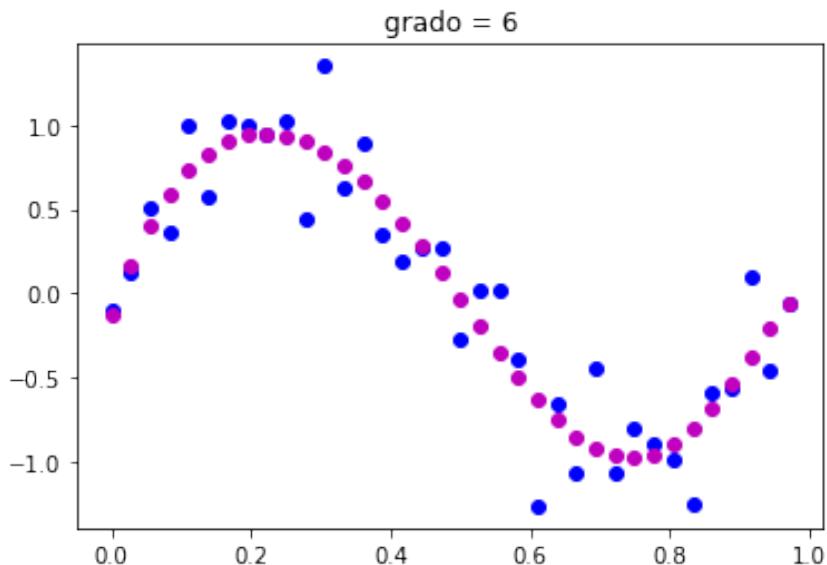
grado = 5



```

parametri stimati: [[-26.97237365  64.41308946 -30.95318171 -15.207612      9.0205
                     -0.08031711]]
varianza delle stime dei parametri: [1.60063456e+03 9.55501012e+03 7.35099033e+03
                                     2.40171329e+01 5.05066514e-02]
valori singolari di [A b]: [7.56747778e+00 4.68099999e+00 2.09240846e+00 6.237308
                           8.99305010e-02 1.91191976e-02 2.01033996e-03]

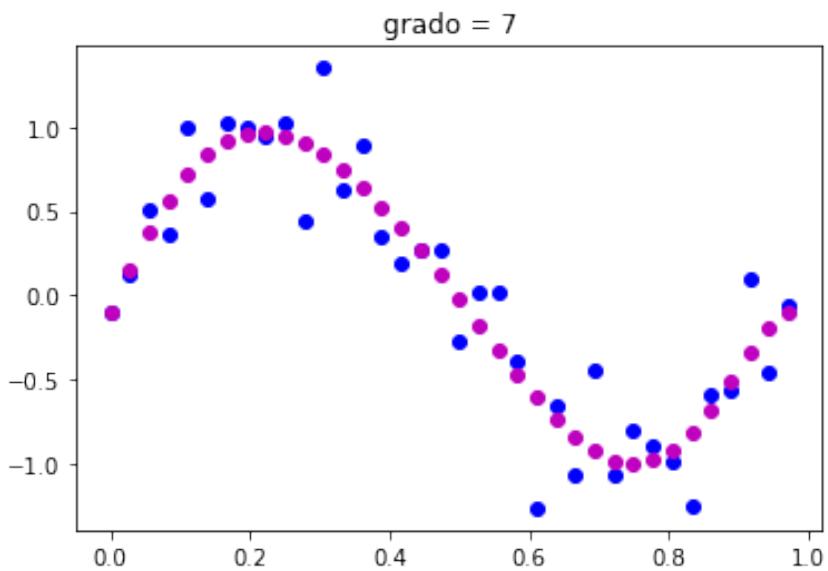
```



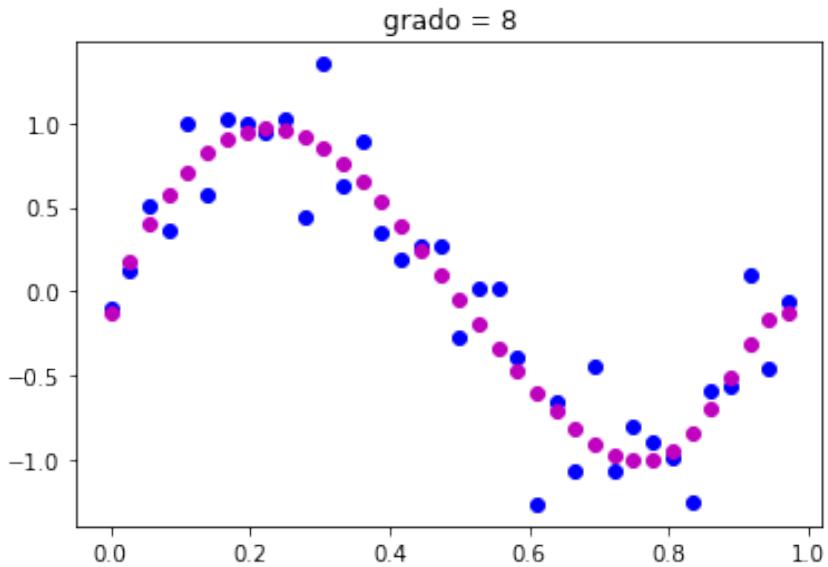
```

parametri stimati: [[-8.71474775e+01  2.27207769e+02 -2.15013903e+02  1.11951830e
                     -4.88488471e+01  1.20380674e+01 -1.30953379e-01]]
varianza delle stime dei parametri: [2.67990888e+04 2.29618164e+05 2.85305007e+05
                                       5.06676458e+03 5.67352571e+01 6.07902287e-02]
valori singolari di [A b]: [7.65156813e+00 4.70141093e+00 2.26706989e+00 6.672369
                           1.25767951e-01 2.67316904e-02 4.20083754e-03 3.59685616e-04]

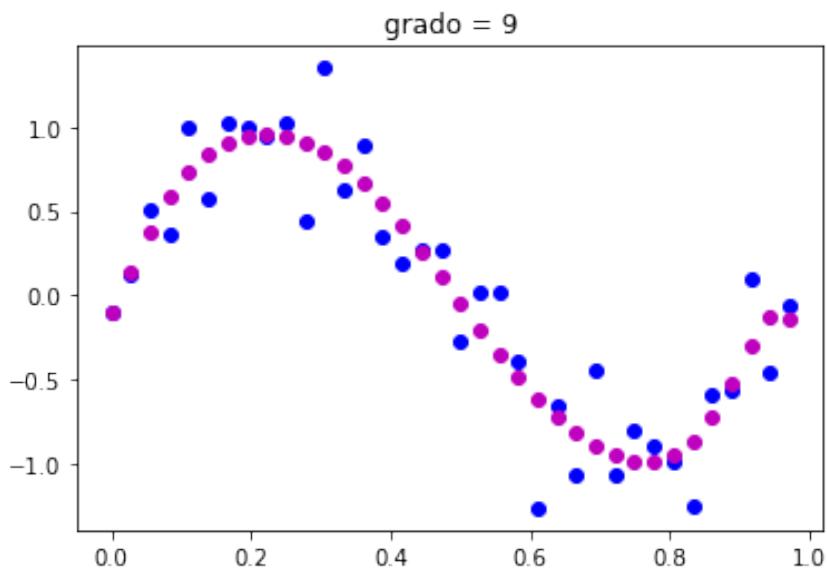
```



```
parametri stimati: [[-2.83539876e+02  8.77675712e+02 -1.06658596e+03  6.49707736e-02  
                     -1.87315489e+02  1.23651090e+00  8.80185740e+00 -9.52227072e-02]]  
varianza delle stime dei parametri: [4.56323619e+05  5.31130663e+06  9.73745043e+06  
      5.90270338e+05  1.94534723e+04  1.17839465e+02  6.98141036e-02]  
valori singolari di [A b]: [7.71807835e+00  4.71675978e+00  2.41853566e+00  6.996957e-01  
    1.63813974e-01  3.37109272e-02  6.77523308e-03  8.03433160e-04  
    6.35915206e-05]
```



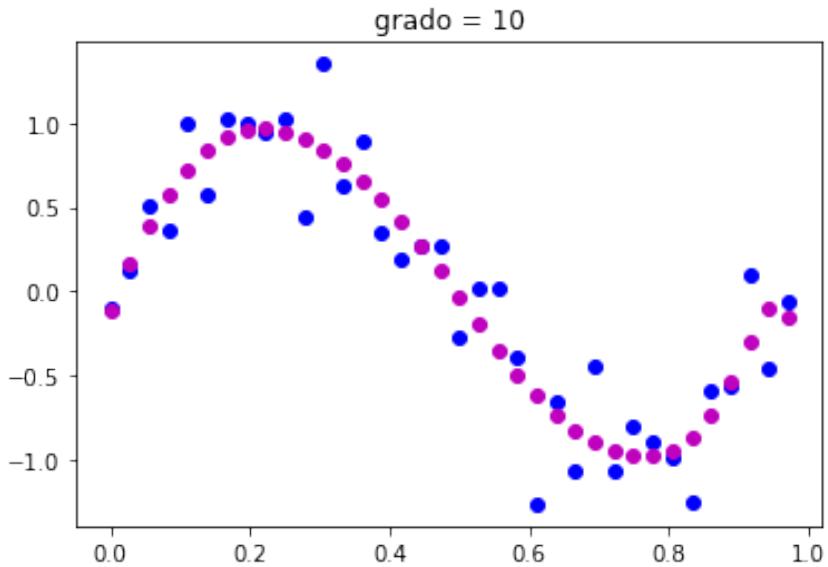
```
parametri stimati: [[-1.04884192e+03  3.79528981e+03 -5.57971595e+03  4.27365957e
-1.80895434e+03  4.32129713e+02 -7.63235187e+01  1.25581096e+01
-1.22636009e-01]]
varianza delle stime dei parametri: [7.89306229e+06  1.19841180e+08  3.04664660e+08
4.80551290e+07  3.36214720e+06  6.32323502e+04  2.22812311e+02
7.74199872e-02]
valori singolari di [A b]: [7.77200598e+00  4.72871605e+00  2.55031233e+00  7.255277
2.02518502e-01  4.02920606e-02  9.61135503e-03  1.37073683e-03
1.51014872e-04  1.11047911e-05]
```



```

parametri stimati: [[-4.13157446e+03  1.70267964e+04 -2.92055487e+04  2.69830148e
                     -1.45128844e+04  4.58762868e+03 -7.99451434e+02  4.34601067e+01
                     8.07006078e+00 -1.01197639e-01]]
varianza delle stime dei parametri: [1.38793507e+08 2.66474986e+09 8.97883245e+09
                                     3.09152704e+09 3.82351749e+08 1.58077728e+07 1.82016684e+05
                                     3.94067006e+02 8.37560651e-02]
valori singolari di [A b]: [7.81658869e+00 4.73831483e+00 2.66549814e+00 7.472906
                           2.40740006e-01 4.66619533e-02 1.25977714e-02 2.04389047e-03
                           2.71518787e-04 2.78595333e-05 1.91729477e-06]

```

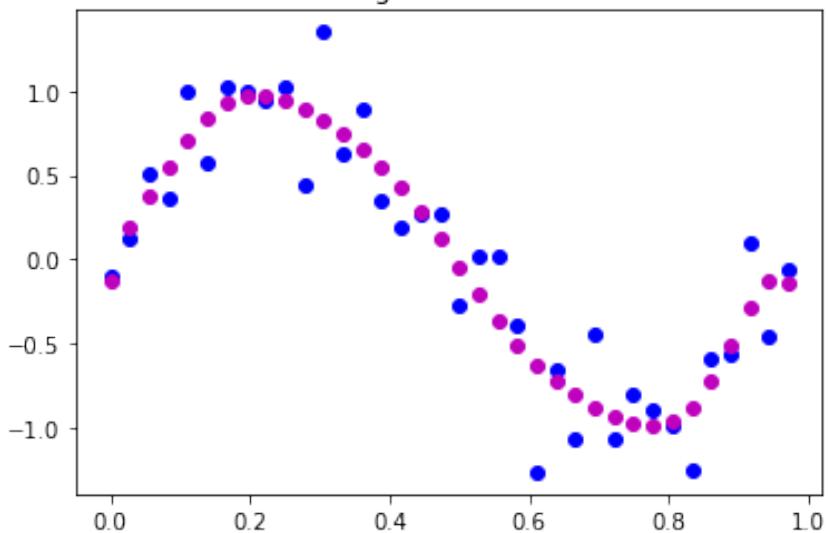


```

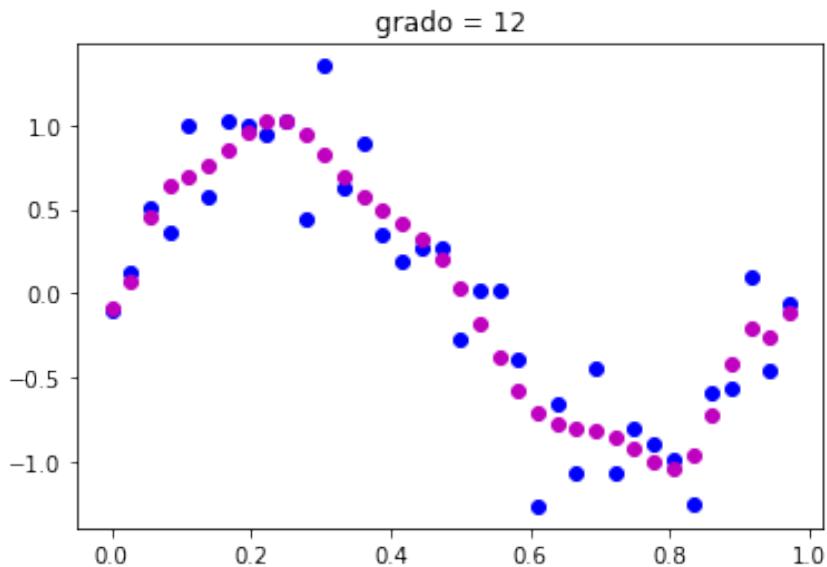
parametri stimati: [[-1.11689139e+04  5.01617570e+04 -9.52814771e+04  9.96352785e
-6.25521238e+04  2.41242012e+04 -5.60867091e+03  7.55088950e+02
-7.75373566e+01  1.16748114e+01 -1.12212382e-01]]
varianza delle stime dei parametri: [2.48720525e+09 5.89155030e+10 2.54231169e+11
1.69081777e+11 3.29655688e+10 2.46902581e+09 6.45687313e+07
4.80655140e+05 6.67894480e+02 8.93478675e-02]
valori singolari di [A b]: [7.85402370e+00 4.74622044e+00 2.76670573e+00 7.665243
2.77690650e-01 5.29478976e-02 1.56432298e-02 2.80463390e-03
4.23780141e-04 5.25319151e-05 5.05786687e-06 3.27608938e-07]

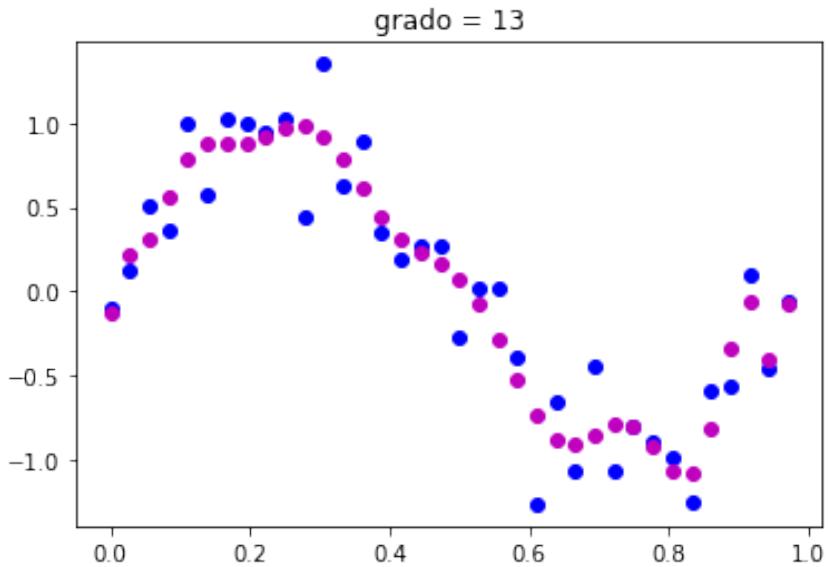
```

grado = 11



parametri stimati: [[5.90962685e+04 -3.27169794e+05 7.80630156e+05 -1.05091245e+06
8.77473424e+05 -4.70346551e+05 1.61914326e+05 -3.47693019e+04
4.37511401e+03 -3.08757468e+02 1.72929109e+01 -1.22812346e-01]]
varianza delle stime dei parametri: [4.28503664e+10 1.22659113e+12 6.59446292e+12
7.74168082e+12 2.20298848e+12 2.65262752e+11 1.28880313e+10
2.25726915e+08 1.14432502e+06 1.07275451e+03 9.41027272e-02]
valori singolari di [A b]: [7.88585908e+00 4.75287332e+00 2.85609412e+00 7.841921e+00
3.12850510e-01 5.92283511e-02 1.86786807e-02 3.63615500e-03
6.05460251e-04 8.54877512e-05 9.96805384e-06 9.05167555e-07
5.51932876e-08]

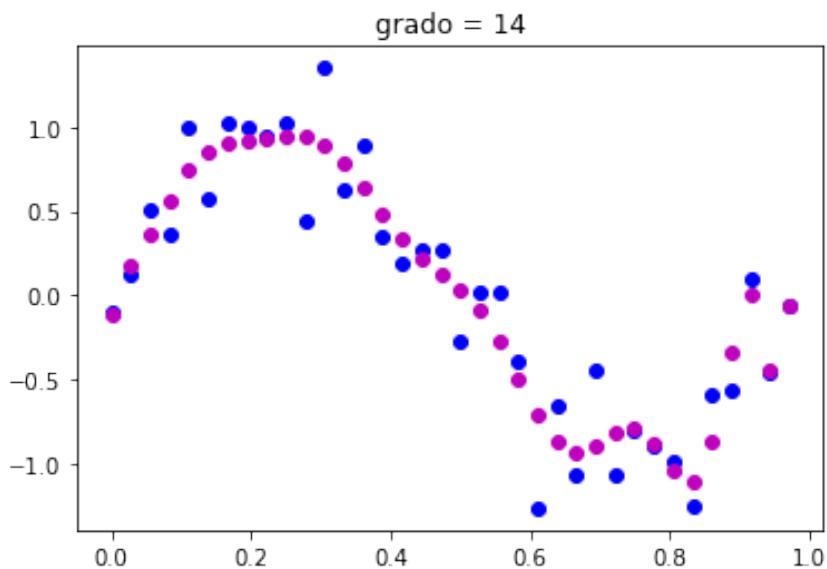




```

parametri stimati: [[ 6.59783236e+06 -4.05205059e+07  1.09880864e+08 -1.73179036e
                      1.75713951e+08 -1.20116971e+08  5.62634957e+07 -1.80094996e+07
                      3.85883810e+06 -5.30560545e+05  4.34287015e+04 -1.86686404e+03
                      3.96543531e+01 -1.20374279e-01]]
varianza delle stime dei parametri: [2.83464857e+11 8.03903624e+12 4.85467801e+13
                                    2.07650377e+14 1.79611298e+14 7.56487656e+13 1.47171957e+13
                                    1.26798519e+12 4.51315621e+10 5.85141196e+08 2.18962717e+06
                                    1.46568068e+03 8.38932535e-02]
valori singolari di [A b]: [7.93694473e+00 4.76353407e+00 3.00618469e+00 8.170983
                           3.76644161e-01 7.19141036e-02 2.45458013e-02 5.45374586e-03
                           1.04603052e-03 1.76102278e-04 2.59044628e-05 3.25814412e-06
                           3.36166796e-07 2.64687781e-08 1.43532229e-09]

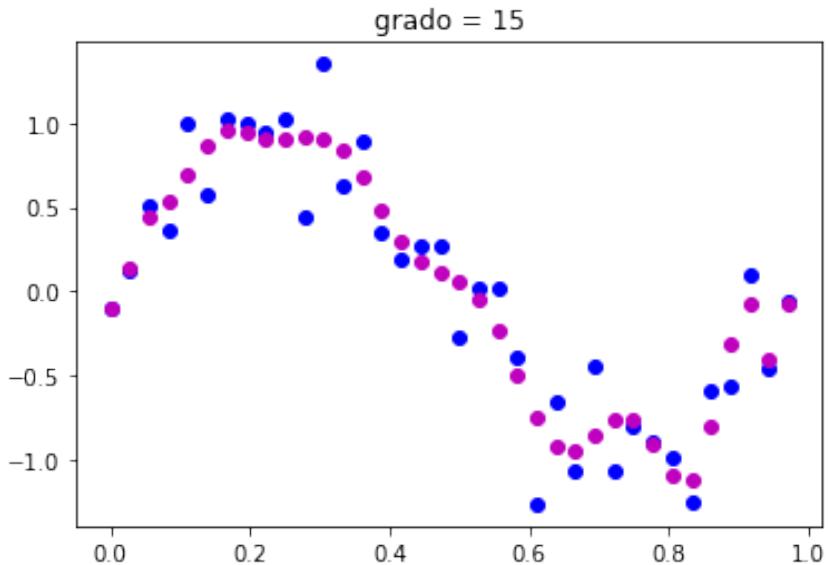
```



```

parametri stimati: [[ 9.84878299e+06 -6.04286075e+07  1.63356508e+08 -2.55793648e
                     2.56473407e+08 -1.71721854e+08  7.75789434e+07 -2.32698543e+07
                     4.39555850e+06 -4.52663034e+05  1.14467572e+04  2.19595037e+03
                     -2.17126811e+02  1.45106816e+01 -1.12288790e-01]]
varianza delle stime dei parametri: [6.32585430e+11 1.64648041e+13 7.92852137e+13
                                    4.74174588e+14 7.23776658e+14 5.07301493e+14 1.64260405e+14
                                    2.49774571e+13 1.74906729e+12 5.28774846e+10 6.08461326e+08
                                    2.11293490e+06 1.38474556e+03 8.58579413e-02]
valori singolari di [A b]: [7.95767346e+00 4.76790446e+00 3.06954053e+00 8.330062e
                           4.05020205e-01 7.83335855e-02 2.73314947e-02 6.41561991e-03
                           1.29928399e-03 2.33203682e-04 3.71396138e-05 5.16070280e-06
                           6.06433586e-07 5.80590762e-08 4.51108114e-09 2.44818420e-10]

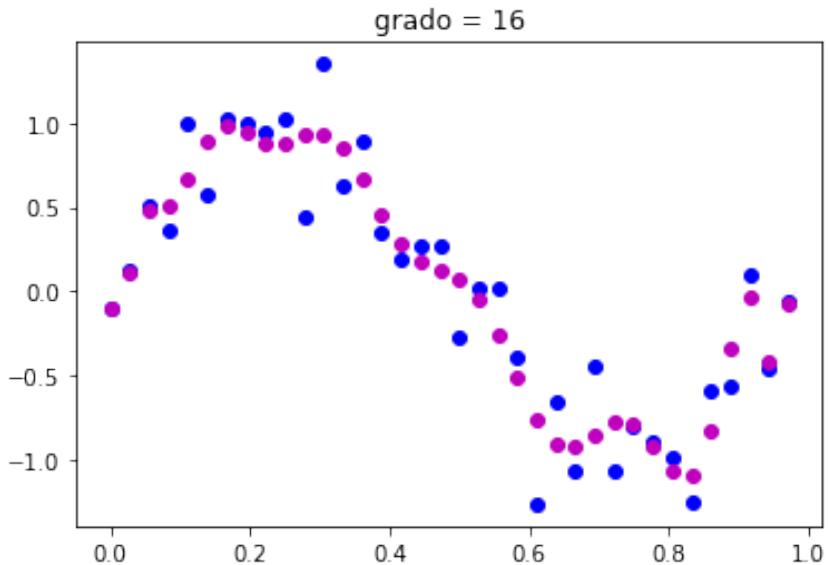
```



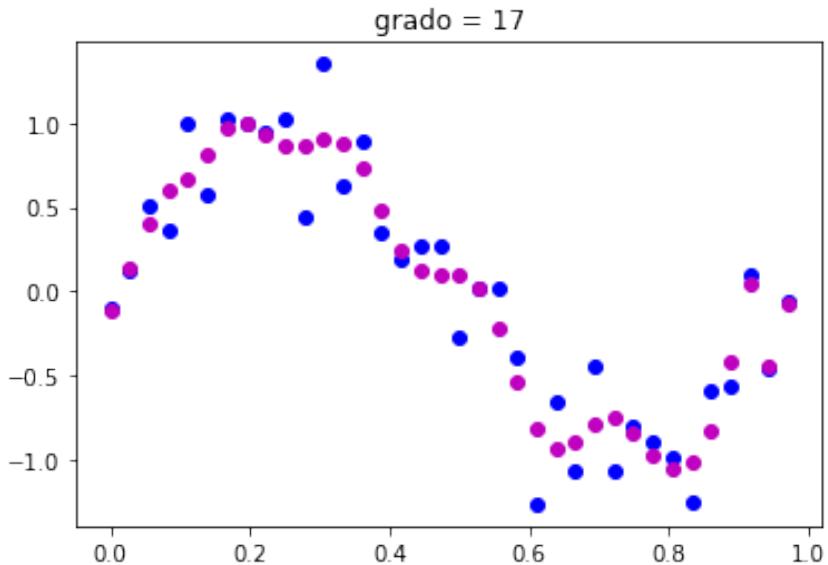
```

parametri stimati: [[-5.36326557e+07  4.00920232e+08 -1.34524566e+09  2.67637349e
-3.51314328e+09  3.20365138e+09 -2.08138532e+09  9.72244941e+08
-3.25493961e+08  7.69597617e+07 -1.24961374e+07  1.33032835e+06
-8.60033585e+04  2.91364887e+03 -2.82099000e+01 -1.05646363e-01]]
varianza delle stime dei parametri: [-5.85633006e+12 -1.35591356e+14 -4.34017231e
1.27682051e+14  1.54783277e+14  2.57098470e+14 -4.11432423e+13
-1.13289928e+14 -3.00208644e+13 -2.59569216e+12 -7.88805002e+10
-7.50359534e+08 -1.55770526e+06 -3.20739094e+00  8.64939664e-02]
valori singolari di [A b]: [7.97590288e+00 4.77179672e+00 3.12649622e+00 8.488020
4.31025629e-01 8.47913324e-02 3.00070619e-02 7.39962940e-03
1.57091384e-03 2.97598249e-04 5.05867027e-05 7.60252220e-06
9.85062278e-07 1.08491059e-07 1.04386886e-08 7.92133767e-10
3.85968854e-11]

```



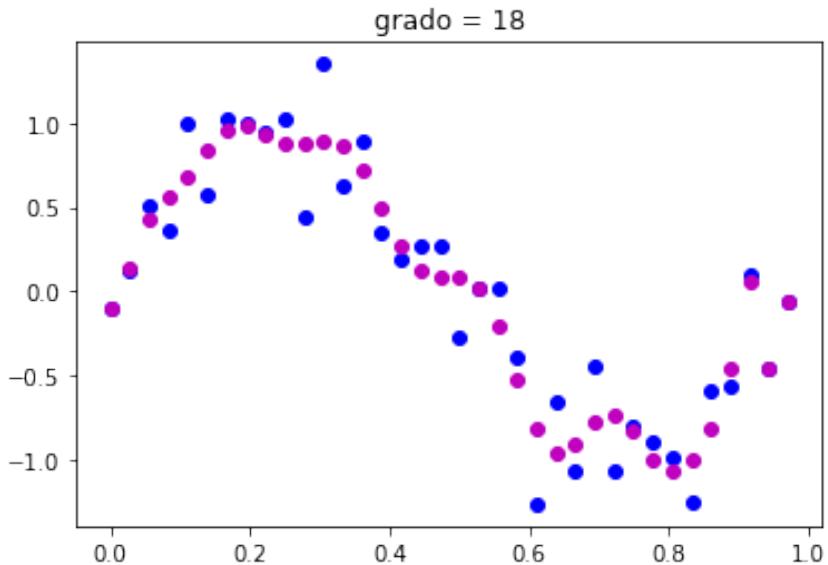
parametri stimati: [[1.34085199e+08 -1.09651768e+09 4.08138912e+09 -9.14222570e+09
 1.37305447e+10 -1.45825633e+10 1.12572230e+10 -6.39436832e+09
 2.67595495e+09 -8.18218098e+08 1.79485273e+08 -2.73949108e+07
 2.77213712e+06 -1.71994256e+05 5.66398607e+03 -6.24454747e+01
 -1.03263675e-01]]
 varianza delle stime dei parametri: [1.93338449e+12 1.96780565e+13 -9.93254084e+13
 1.49375770e+14 2.52208132e+14 7.49402379e+13 1.08797018e+14
 2.17134973e+13 -9.26137496e+12 -1.59886384e+12 3.69935102e+10
 9.15056264e+09 2.21724269e+08 1.21878178e+06 1.12031183e+03
 9.13365972e-02]
 valori singolari di [A b]: [7.99202544e+00 4.77529421e+00 3.17787981e+00 8.645767e-01
 4.54721094e-01 9.12693744e-02 3.25726323e-02 8.39775980e-03
 1.85840688e-03 3.68769322e-04 6.62067103e-05 1.06046254e-05
 1.48558461e-06 1.82847059e-07 2.04402053e-08 1.86777527e-09
 1.29096985e-10 6.11458030e-12]



```

parametri stimati: [[ 1.36052888e+09 -1.11091756e+10  4.13239492e+10 -9.26882898e
1.39843622e+11 -1.49911812e+11  1.17586027e+11 -6.84760531e+10
2.97343127e+10 -9.58767695e+09  2.26752985e+09 -3.84978636e+08
4.53846058e+07 -3.53242234e+06  1.67735647e+05 -4.26224315e+03
5.20401812e+01 -1.06570362e-01]]
varianza delle stime dei parametri: [ 7.29683310e+12  1.37851877e+14  2.46752077e
7.27204165e+14  1.99119774e+15  3.95470140e+14 -6.56862356e+13
1.18932001e+15  1.08807918e+15  3.56424944e+14  5.25726457e+13
3.57477065e+12  1.05033405e+11  1.15969653e+09  3.74062393e+06
2.11854432e+03  9.32739781e-02]
valori singolari di [A b]: [8.00635559e+00 4.77846055e+00 3.22438611e+00 8.803674
4.76210472e-01 9.77474422e-02 3.50320125e-02 9.40333701e-03
2.15941608e-03 4.46159325e-04 8.39203507e-05 1.41749051e-05
2.11987891e-06 2.86379084e-07 3.57863246e-08 3.70932108e-09
3.15772137e-10 2.09022274e-11 9.32500453e-13]

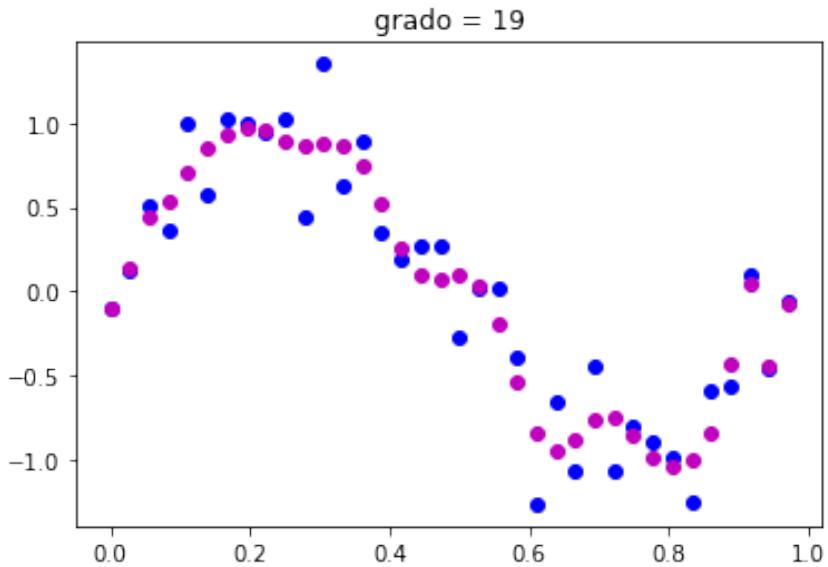
```



```

parametri stimati: [[ 2.26704011e+09 -1.84760761e+10  6.85726734e+10 -1.53426497e
                     2.30925263e+11 -2.47104523e+11  1.93787011e+11 -1.13233751e+11
                     4.96911381e+10 -1.64206539e+10  4.08764186e+09 -7.67602131e+08
                     1.09260638e+08 -1.18292359e+07  9.55580109e+05 -5.33243767e+04
                     1.70875529e+03 -1.25669603e+01 -1.05853041e-01]]
varianza delle stime dei parametri: [ 1.00456690e+12  6.36691030e+13  2.81760983e
                     -1.43021482e+15 -5.81680541e+14  1.17430101e+14  8.51015684e+13
                     -6.70259165e+14 -1.22059619e+15 -1.03034068e+15 -3.74321603e+14
                     -5.59924484e+13 -3.35080314e+12 -7.33335656e+10 -4.61122812e+08
                     -2.31293865e+05  7.29572979e+02  9.68716635e-02]
valori singolari di [A b]: [ 8.01914850e+00  4.78134510e+00  3.26660198e+00  8.961717
                     4.95628009e-01  1.04205136e-01  3.73908429e-02  1.04108638e-02
                     2.47175644e-03  5.29187420e-04  1.03617757e-04  1.83116055e-05
                     2.89873894e-06  4.24270035e-07  5.77025297e-08  6.58447143e-09
                     6.50312369e-10  5.19855117e-11  3.32212854e-12  1.43659688e-13]

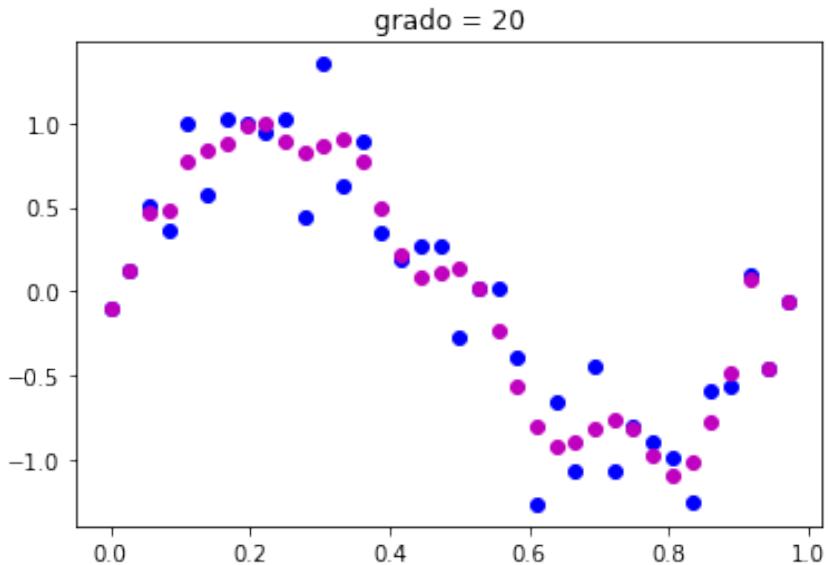
```



```

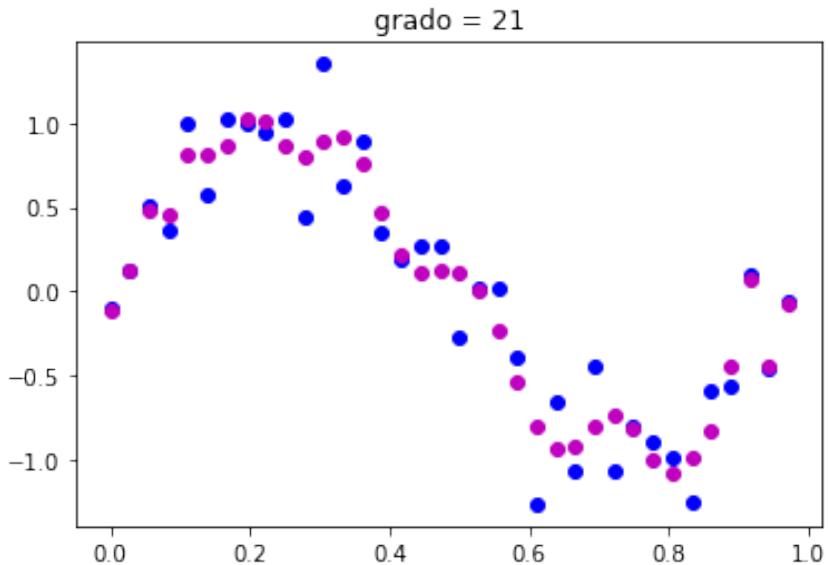
parametri stimati: [[-1.08346662e+10  1.02337628e+11 -4.44872513e+11  1.18025785e
                     -2.13667605e+12  2.79595602e+12 -2.73307762e+12  2.03432524e+12
                     -1.16487917e+12  5.15031374e+11 -1.75514232e+11  4.57838938e+10
                     -9.02845504e+09  1.32008958e+09 -1.39013086e+08  1.00835838e+07
                     -4.68513887e+05  1.21806234e+04 -1.19869686e+02 -1.05855265e-01]]
varianza delle stime dei parametri: [ 3.76819064e+11  6.58410629e+12 -7.38321854e
                                     4.15374975e+14  1.19492680e+14 -2.16519379e+14  9.11943183e+12
                                     -1.31309340e+14  1.45750337e+14  1.24866172e+14  8.93197852e+13
                                     4.30051207e+13  8.54275712e+12  7.33106528e+11  2.72187266e+10
                                     3.89802780e+08  1.68420283e+06  1.36214700e+03  1.02419974e-01]
valori singolari di [A b]: [8.03061358e+00 4.78398672e+00 3.30502616e+00 9.119596
                           5.13127290e-01 1.10623189e-01 3.96554194e-02 1.14158616e-02
                           2.79339610e-03 6.17263617e-04 1.25168091e-04 2.30067627e-05
                           3.83220949e-06 6.01301437e-07 8.73105575e-08 1.07883258e-08
                           1.19289437e-09 1.08692744e-10 8.60962285e-12 5.26190399e-13
                           2.12043091e-14]

```



```

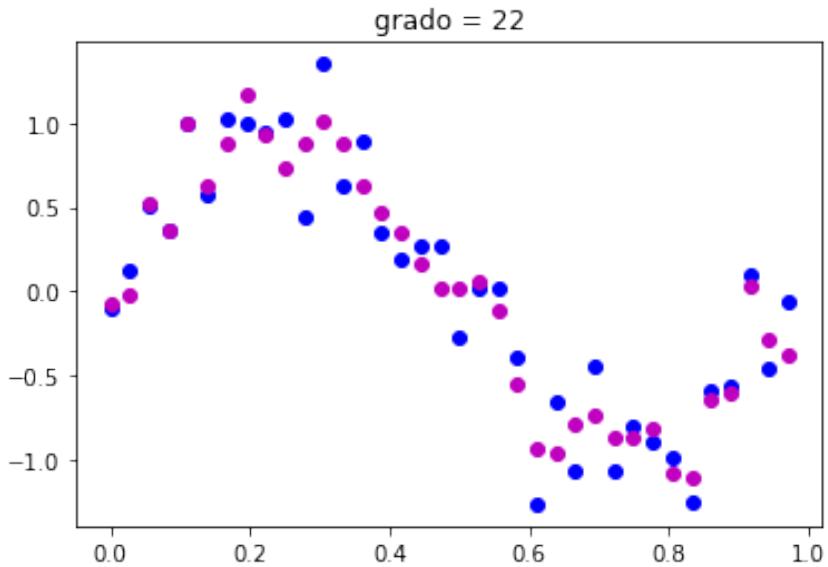
parametri stimati: [[ 1.02577875e+11 -1.00806819e+12  4.59470506e+12 -1.28896977e
                     2.49053310e+13 -3.51424284e+13  3.74655540e+13 -3.07995995e+13
                     1.97527102e+13 -9.93536447e+12  3.92000291e+12 -1.20771012e+12
                     2.87841577e+11 -5.22900774e+10  7.08447967e+09 -6.93654547e+08
                     4.68430240e+07 -2.02660334e+06  4.92013612e+04 -4.82307492e+02
                     -9.88761733e-02]]
varianza delle stime dei parametri: [ 8.87061256e+12  7.29830197e+13 -2.04810423e
                     -5.52776648e+14 -2.13139582e+14 -9.14344365e+12 -1.05598152e+14
                     1.98647217e+14  8.80032581e+13  1.68957014e+14  1.91294647e+14
                     1.12524519e+13 -2.86066364e+13 -5.74356327e+12 -2.37807502e+11
                     3.80726618e+09  2.10612681e+08  1.36554303e+06  1.31965355e+03
                     1.06688378e-01]
valori singolari di [A b]: [8.04092439e+00 4.78641652e+00 3.34008511e+00 9.276837
                     5.28871776e-01 1.16984125e-01 4.18320009e-02 1.24147208e-02
                     3.12244673e-03 7.09800765e-04 1.48428515e-04 2.82491040e-05
                     4.92958357e-06 8.21516169e-07 1.25615712e-07 1.66377483e-08
                     2.00894603e-09 2.02694527e-10 1.87070399e-11 1.39499334e-12
                     8.00831174e-14 3.06135853e-15]
```



```

parametri stimati: [[-3.13925978e+11  3.30729466e+12 -1.62282932e+13  4.92464415e
-1.03497903e+14  1.59854783e+14 -1.87909035e+14  1.71768779e+14
-1.23697261e+14  7.06658070e+13 -3.20940865e+13  1.15640802e+13
-3.28540112e+12  7.28212555e+11 -1.23951529e+11  1.58403463e+10
-1.47173166e+09  9.48290030e+07 -3.93497949e+06  9.21416811e+04
-8.84201647e+02 -1.12179697e-01]]
varianza delle stime dei parametri: [-7.62690919e+12 -1.52120957e+14 -4.97192026e
-3.00093513e+14 -5.86195623e+14 -3.65677461e+14 -4.52400766e+14
-3.03294967e+14  9.96375860e+13  2.24960357e+14  1.65604423e+13
-2.91612883e+13  1.68491235e+13  2.94339472e+13  1.08099053e+13
1.29964765e+12  5.47903582e+10  7.66778996e+08  2.94236431e+06
1.97279244e+03  1.13725237e-01]
valori singolari di [A b]: [8.05022612e+00 4.78865970e+00 3.37214550e+00 9.432871
5.43026992e-01 1.23272512e-01 4.39264449e-02 1.34045626e-02
3.45715582e-03 8.06224957e-04 1.73251974e-04 3.40261387e-05
6.19908431e-06 1.08797312e-06 1.73526577e-07 2.44609009e-08
3.16605209e-09 3.48097933e-10 3.60810865e-11 3.09279521e-12
2.18296472e-13 1.18139183e-14 4.35565443e-16]

```

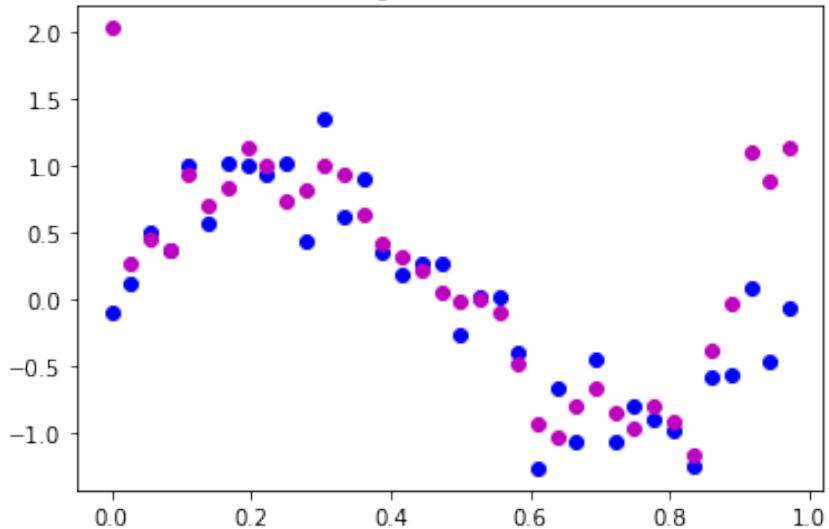


```

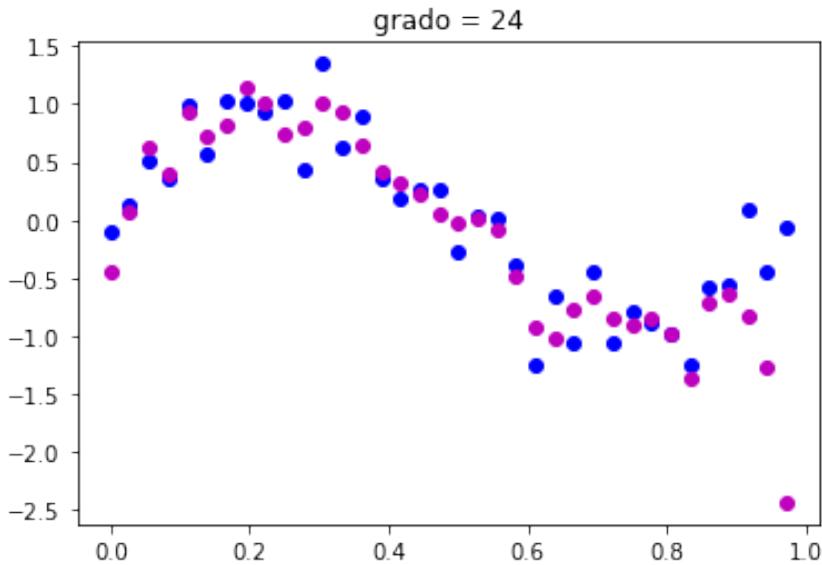
parametri stimati: [[ 7.99274346e+12 -8.56865967e+13  4.29094032e+14 -1.33333301e
 2.88030546e+15 -4.59300779e+15  5.60308096e+15 -5.34774302e+15
 4.05006877e+15 -2.45427870e+15  1.19466254e+15 -4.67177363e+14
 1.46272816e+14 -3.64130146e+13  7.12807371e+12 -1.07989686e+12
 1.23803601e+11 -1.04044085e+10  6.11825250e+08 -2.34051636e+07
 5.11549274e+05 -4.69790901e+03 -7.04333415e-02]]
varianza delle stime dei parametri: [-3.71693090e+13 -6.63218250e+14 -1.33282969e
 1.96716791e+13 -3.55592816e+14 -4.02488182e+14 -2.93384052e+14
 -2.28317400e+14 -2.62719384e+14  2.13258271e+13  4.47582052e+13
 1.97219175e+14  1.26476804e+14 -5.74365328e+13 -5.72785633e+13
 -9.38527717e+12 -3.14706741e+11  8.15686033e+09  3.18900028e+08
 1.80467140e+06  1.53324386e+03  1.07050543e-01]
valori singolari di [A b]: [8.05864117e+00 4.79073696e+00 3.40152429e+00 9.587098
 5.55754432e-01 1.29474997e-01 4.59440485e-02 1.43831136e-02
 3.79590103e-03 9.05984378e-04 1.99493404e-04 4.03253853e-05
 7.64734574e-06 1.40265337e-06 2.31885088e-07 3.45823561e-08
 4.73191917e-09 5.61417123e-10 6.37146240e-11 6.08152697e-12
 4.96784022e-13 3.29386530e-14 1.75344594e-15 5.34506335e-17]

```

grado = 23



parametri stimati: [[1.50703127e+13 -1.61214430e+14 8.04304576e+14 -2.48486900e+15
5.32309114e+15 -8.38881388e+15 1.00681192e+16 -9.39669536e+15
6.90131661e+15 -4.00822437e+15 1.83806011e+15 -6.59462482e+14
1.81327497e+14 -3.65511253e+13 4.80724398e+12 -2.19390440e+11
-6.11069798e+10 1.63532232e+10 -2.05212416e+09 1.54921860e+08
-7.05091206e+06 1.75940699e+05 -1.84971802e+03 2.02917542e+00]
varianza delle stime dei parametri: [-3.67547851e+14 -4.49726479e+15 -3.71052024e+15
-6.39809498e+15 6.40556461e+13 -1.73137872e+15 -9.46277387e+15
4.06003552e+15 -1.57247158e+15 -1.55443384e+15 -3.36837266e+14
-6.46150270e+14 -2.63880099e+14 1.44505549e+15 3.39027743e+14
-7.31797099e+14 -2.33851997e+14 -2.02477084e+13 -5.61737978e+11
-4.37065206e+09 -4.73250950e+06 4.95098054e+03 8.51063531e-01]
valori singolari di [A b]: [8.06627350e+00 4.79266553e+00 3.42849687e+00 9.738940e+00
5.67207184e-01 1.35580205e-01 4.78895111e-02 1.53485960e-02
4.13718683e-03 1.00855646e-03 2.27014234e-04 4.71348239e-05
9.27884947e-06 1.76652152e-06 3.01489004e-07 4.73063695e-08
6.77367948e-09 8.61331474e-10 1.05036805e-10 1.09501161e-11
1.00015361e-12 7.67961717e-14 5.00140023e-15 1.95693618e-16
2.64746482e-17]

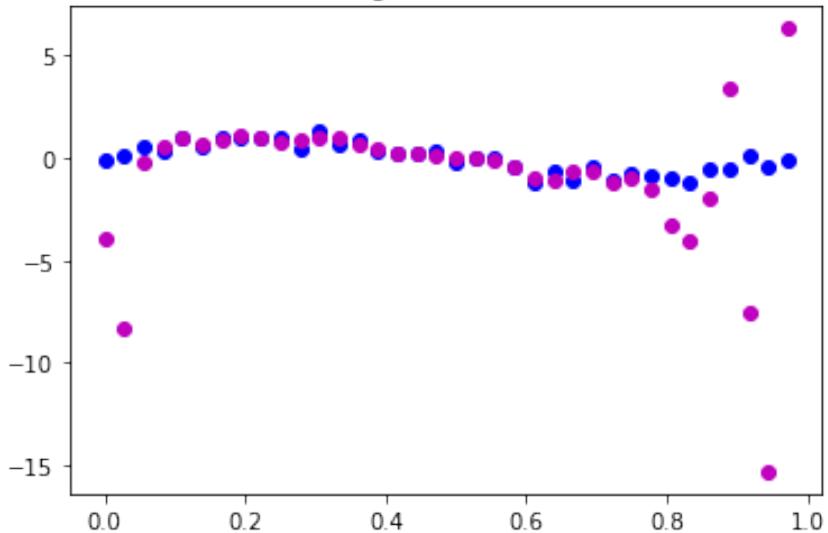


```

parametri stimati: [[-5.95168183e+12  8.54211784e+13 -5.51960396e+14  2.15979551e
-5.77749409e+15  1.12745873e+16 -1.66925202e+16  1.92242144e+16
-1.74999371e+16  1.27148976e+16 -7.40930596e+15  3.46473825e+15
-1.29533660e+15  3.83869156e+14 -8.88151701e+13  1.56355876e+13
-1.99816395e+12  1.66610812e+11 -5.84024772e+09 -4.57495512e+08
7.41702400e+07 -4.35014736e+06  1.21752871e+05 -1.28328025e+03
-4.41969846e-01]]
varianza delle stime dei parametri: [-1.81761591e+14 -2.29740756e+14  5.53491098e
1.65419352e+15  1.00863673e+15 -1.08084157e+15 -3.01042518e+15
-2.30209226e+15  5.93220009e+15  1.40584775e+15 -4.84339648e+14
-2.10832972e+14  7.92349465e+14  8.72393559e+14  6.76274327e+14
1.71661936e+15  1.14301367e+15  2.45274579e+14  1.99838018e+13
6.32535490e+11  7.18069429e+09  2.37241639e+07  1.42910095e+04
7.62168624e-01]
valori singolari di [A b]: [8.07321203e+00  4.79445990e+00  3.45330360e+00  9.887867
5.77527127e-01  1.41578576e-01  4.97669621e-02  1.62996325e-02
4.47964232e-03  1.11345317e-03  2.55685248e-04  5.44427513e-05
1.10954667e-05  2.17970049e-06  3.83097835e-07  6.29017819e-08
9.35821570e-09  1.26818160e-09  1.63849349e-10  1.84158164e-11
1.83953436e-12  1.58107002e-13  1.18275432e-14  5.51269399e-16
5.31015154e-16  5.31015154e-16]

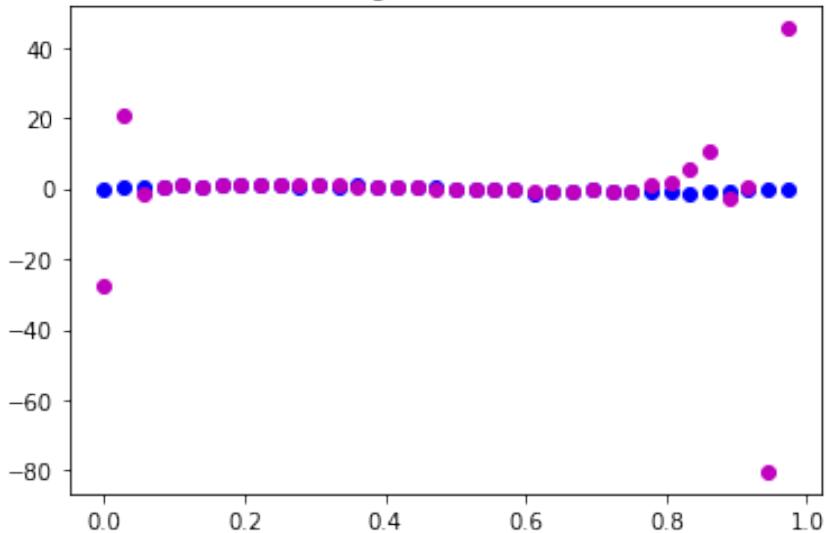
```

grado = 25

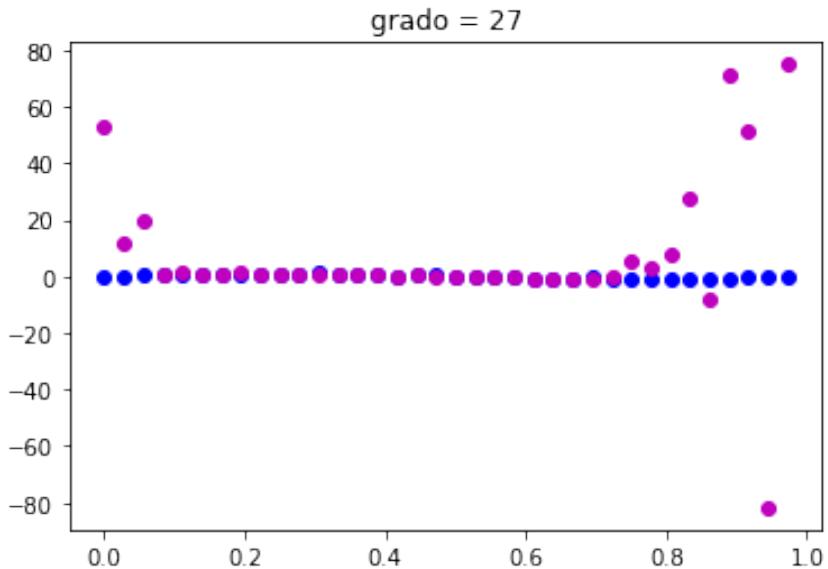


parametri stimati: [[1.05580801e+13 9.76678682e+13 -1.82205565e+15 1.09522672e+16
-3.84482606e+16 9.22691378e+16 -1.62942798e+17 2.20833529e+17
-2.35820190e+17 2.01858446e+17 -1.40066404e+17 7.93330521e+16
-3.68100141e+16 1.40019893e+16 -4.35807906e+15 1.10480626e+15
-2.26404637e+14 3.70934224e+13 -4.78409886e+12 4.75411297e+11
-3.53181339e+10 1.87707901e+09 -6.66269912e+07 1.39630153e+06
-1.28072211e+04 -3.92752290e+00]]
varianza delle stime dei parametri: [2.74187359e+13 -4.88986646e+16 -1.07471244e+17
1.27752797e+17 6.40952524e+16 8.58990966e+16 -2.90914005e+16
-6.08001651e+16 -1.13445351e+17 -6.28100068e+15 -1.24550400e+17
5.95505864e+16 -1.90644664e+16 7.57566356e+16 2.79739618e+16
3.28112695e+16 2.11153627e+16 2.46122826e+15 1.03871915e+14
4.94443139e+13 4.63808854e+12 1.05469346e+11 5.79612223e+08
5.34070640e+05 4.33143855e+01]
valori singolari di [A b]: [8.07953320e+00 4.79613241e+00 3.47615519e+00 1.003341e+00
5.86843450e-01 1.47462182e-01 5.15800201e-02 1.72351659e-02
4.82201977e-03 1.22022448e-03 2.85388034e-04 6.22372440e-05
1.30962105e-05 2.64169892e-06 4.77421494e-07 8.15915071e-08
1.25528966e-08 1.80322751e-09 2.44268807e-10 2.93172027e-11
3.15260991e-12 2.96873730e-13 2.48979466e-14 1.46547541e-15
5.28740582e-16 5.28740582e-16 4.72320933e-16]

grado = 26



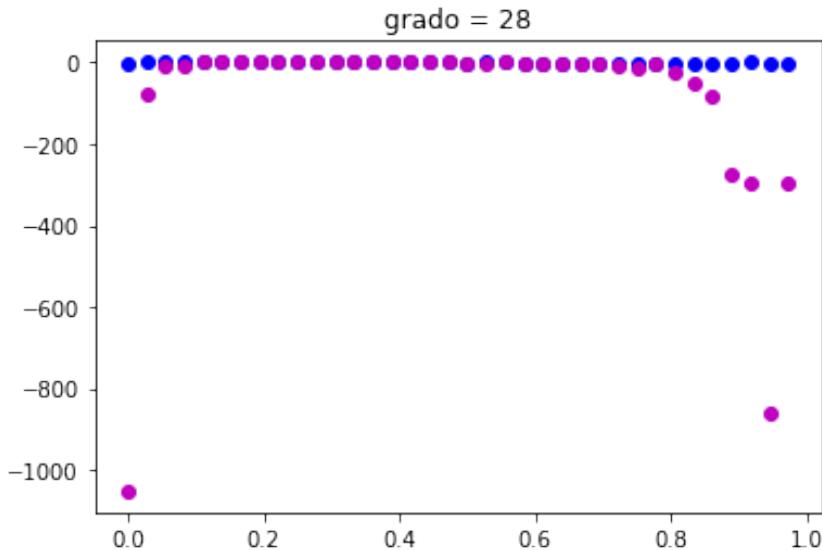
parametri stimati: [[6.05505127e+14 -7.11818092e+15 3.93321050e+16 -1.35743501e+17
3.27914853e+17 -5.88795786e+17 8.14283030e+17 -8.86476277e+17
7.69377884e+17 -5.35173187e+17 2.97671188e+17 -1.30694832e+17
4.37895783e+16 -1.01802589e+16 1.02182008e+15 3.51711667e+14
-2.20207554e+14 6.64512256e+13 -1.37818392e+13 2.10805927e+12
-2.41507377e+11 2.05507670e+10 -1.26310268e+09 5.30626746e+07
-1.36752689e+06 1.67988711e+04 -2.75302120e+01]]
varianza delle stime dei parametri: [2.81670367e+16 -1.68645515e+18 -9.99861673e+17
3.28008594e+18 -3.84532553e+17 1.27881096e+18 6.76601586e+18
7.05058283e+18 -6.03609550e+18 3.67975939e+18 5.17560663e+18
-6.25134500e+18 -2.42970571e+16 -6.11131463e+17 1.05859249e+18
-1.11782382e+18 1.30427008e+18 2.65804708e+18 1.11749154e+18
1.71892896e+17 1.07431272e+16 2.87278585e+14 3.29918309e+12
1.36297371e+10 1.20726629e+07 1.07266703e+03]
valori singolari di [A b]: [8.08530315e+00 4.79769369e+00 3.49723715e+00 1.017520e+00
5.95272184e-01 1.53224544e-01 5.33318626e-02 1.81543932e-02
5.16319292e-03 1.32846002e-03 3.16015301e-04 7.05054301e-05
1.52772416e-05 3.15163533e-06 5.85096135e-07 1.03548131e-07
1.64261147e-08 2.48777334e-09 3.50686125e-10 4.45996822e-11
5.10329489e-12 5.18877380e-13 4.74943029e-14 3.24795396e-15
5.26506942e-16 5.26506942e-16 5.26506942e-16 4.68234755e-16]



```

parametri stimati: [[-5.62091316e+15  6.37387808e+16 -3.35845494e+17  1.08761356e
                     -2.40774490e+18  3.81784584e+18 -4.37344441e+18  3.45982879e+18
                     -1.47777094e+18 -4.53595064e+17  1.46173539e+18 -1.48357148e+18
                     1.00679167e+18 -5.12940521e+17  2.03593620e+17 -6.36626675e+16
                     1.56342951e+16 -2.96585125e+15  4.18409729e+14 -3.99884979e+13
                     1.76891937e+12  1.34626678e+11 -3.13171620e+10  2.71097902e+09
                     -1.30133712e+08  3.35519398e+06 -3.63705190e+04  5.27980483e+01]]
varianza delle stime dei parametri: [ 7.09149975e+18  2.56892052e+19 -3.52304983e
                                     -1.27066950e+20 -1.41604720e+19  8.24937433e+19  5.27330500e+18
                                     3.16206021e+18  1.58740187e+18 -2.24059541e+19  1.47753145e+19
                                     -5.32267224e+19  7.19729386e+18 -1.59734036e+19  5.54001950e+18
                                     4.71451310e+18  1.00801427e+19  1.39134799e+19 -1.14425426e+19
                                     -1.61390279e+19 -3.89024882e+18 -2.96805352e+17 -7.62602575e+15
                                     -5.87283561e+13 -9.23683491e+10  4.55167265e+06  3.03569169e+03]
valori singolari di [A b]: [8.09057938e+00 4.79915295e+00 3.51671337e+00 1.031293
                           6.02916466e-01 1.58860452e-01 5.50252947e-02 1.90567110e-02
                           5.50215469e-03 1.43778925e-03 3.47470381e-04 7.92327428e-05
                           1.76321111e-05 3.70842181e-06 7.06653898e-07 1.28895482e-07
                           2.10472081e-08 3.34230213e-09 4.87738213e-10 6.52920589e-11
                           7.88003917e-12 8.55550842e-13 8.45277594e-14 6.40721628e-15
                           5.24313038e-16 5.24313038e-16 5.24313038e-16 5.24313038e-16
                           4.53078231e-16]

```

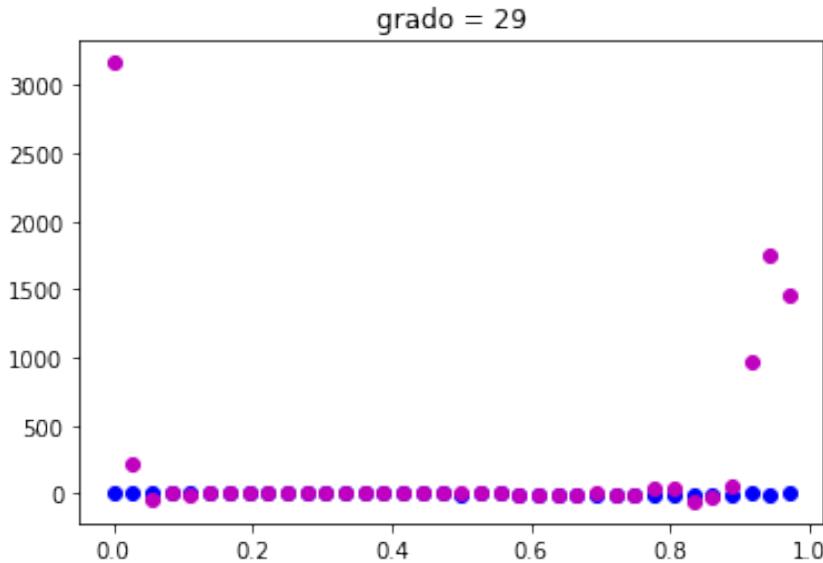


```

parametri stimati: [[-1.03660884e+16  1.20983054e+17 -6.61428740e+17  2.24500741e
-5.27709371e+18  9.04439519e+18 -1.15113466e+19  1.06849594e+19
-6.48416146e+18  1.02910778e+18  3.06171786e+18 -4.55628139e+18
3.96771309e+18 -2.56305324e+18  1.31052247e+18 -5.44042941e+17
1.85253548e+17 -5.18618791e+16  1.19009983e+16 -2.22206304e+15
3.33576173e+14 -3.95706876e+13  3.61837175e+12 -2.45873349e+11
1.17129047e+10 -3.51029674e+08  4.94054436e+06  1.89034862e+04
-1.05233313e+03]]
varianza delle stime dei parametri: [ 7.17276956e+20  2.15689273e+21  5.47140888e
1.72404167e+22 -1.75411897e+22 -4.58445042e+21  7.59886779e+20
1.38384196e+21  1.80027275e+21  1.81147360e+21 -9.64354028e+20
2.25338119e+21  7.90274513e+21  1.32369944e+22 -1.87328876e+20
1.05976338e+21  1.08808018e+21  8.99699913e+20 -1.86382341e+21
-4.75482935e+21 -1.95910070e+21 -2.45232310e+20 -9.44376068e+18
-4.79304866e+16  1.47210881e+15  8.90207572e+12  6.30785402e+09
3.00351647e+05]
valori singolari di [A b]: [8.09541211e+00 4.80051829e+00 3.53472921e+00 1.044636
6.09867273e-01 1.64365812e-01 5.66628124e-02 1.99416732e-02
5.83801387e-03 1.54788037e-03 3.79666198e-04 8.84022855e-05
2.01521827e-05 4.31089122e-06 8.42493370e-07 1.57714742e-07
2.64855971e-08 4.38575958e-09 6.60275012e-10 9.24758628e-11
1.16934470e-11 1.34484048e-12 1.41570169e-13 1.18300871e-14
3.59386898e-15 5.22157898e-16 5.22157898e-16 5.22157898e-16

```

5.22157898e-16 4.40169568e-16]

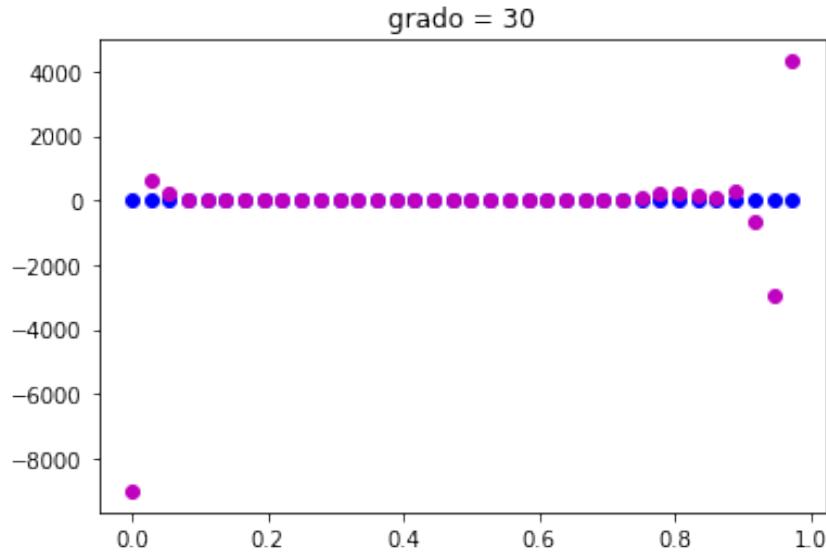


parametri stimati: [[1.66344065e+16 -1.93974152e+17 1.03940542e+18 -3.36023983e
7.17919253e+18 -1.02267747e+19 8.56627791e+18 -3.84199276e+17
-1.08684634e+19 1.85891815e+19 -1.87702087e+19 1.29951440e+19
-5.95564944e+18 1.16743039e+18 7.46526087e+17 -9.15890815e+17
5.48161919e+17 -2.31877592e+17 7.51910404e+16 -1.92095135e+16
3.89623329e+15 -6.25453451e+14 7.85437187e+13 -7.55644726e+12
5.38351762e+11 -2.68274798e+10 8.37144821e+08 -1.19138589e+07
-7.11917160e+04 3.16585747e+03]]
varianza delle stime dei parametri: [5.34903853e+21 3.67674081e+22 3.65747606e
-7.22474109e+22 2.77927912e+21 9.99365361e+21 -2.80936100e+22
2.36646146e+21 -3.82832360e+22 -1.19757332e+21 -6.31419767e+21
2.58275049e+22 5.17763678e+21 1.84983952e+22 -4.55675953e+21
-1.25259410e+22 -1.97140749e+22 -9.08186150e+20 -6.64260545e+21
7.44391210e+20 2.65208158e+21 1.77898437e+21 5.30968896e+20
5.83807063e+19 2.28684090e+18 2.94822001e+16 1.02086768e+14
5.92849996e+10 2.68935346e+06]
valori singolari di [A b]: [8.09984537e+00 4.80179685e+00 3.55141401e+00 1.057532
6.16204414e-01 1.69737502e-01 5.82466567e-02 2.08089561e-02
6.16999068e-03 1.65843830e-03 4.12523943e-04 9.79943941e-05
2.28271577e-05 4.95786801e-06 9.92856234e-07 1.90053085e-07

```

3.28091899e-08 5.63508838e-09 8.73309313e-10 1.27253827e-10
1.67734177e-11 2.03011553e-12 2.25828806e-13 2.07694241e-14
3.59481266e-15 5.20040720e-16 5.20040720e-16 5.20040720e-16
5.20040720e-16 5.20040720e-16 4.30301869e-16]

```



```

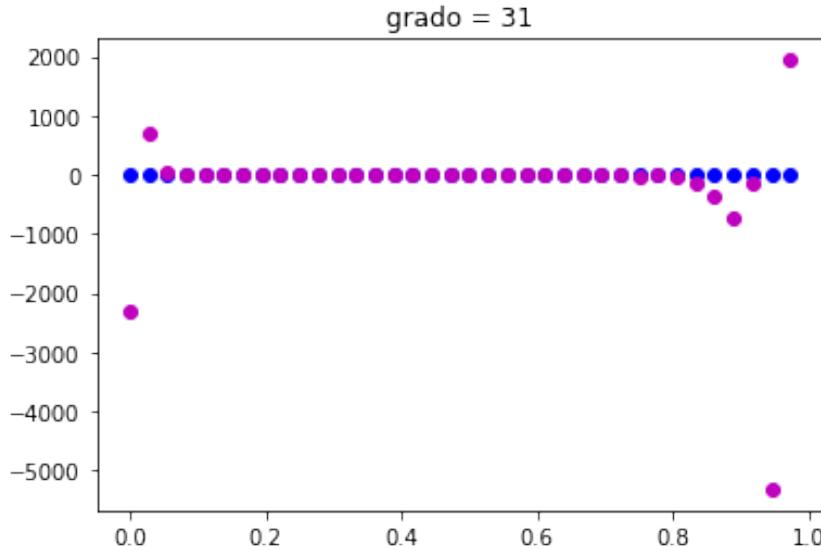
parametri stimati: [[ 2.44933310e+17 -2.75567943e+18  1.42959154e+19 -4.50924911e
9.57119566e+19 -1.42028400e+20  1.46100441e+20 -9.50841970e+19
2.25524200e+19  2.18431294e+19 -2.01272392e+19 -5.47731185e+18
2.44705276e+19 -2.58047213e+19  1.67387921e+19 -7.45362425e+18
2.19697812e+18 -2.91926556e+17 -9.19250346e+16  7.34331928e+16
-2.65056650e+16  6.54378051e+15 -1.19923815e+15  1.66924234e+14
-1.76626614e+13  1.40358302e+12 -8.18005796e+10  3.36393993e+09
-9.14284824e+07  1.43814931e+06 -9.00829994e+03]]
varianza delle stime dei parametri: [-3.53159203e+22 -5.85830211e+22  6.41337738e
3.08212890e+23  1.87918116e+22 -5.84674535e+23 -7.72276106e+23
-7.40100402e+23  2.17647003e+23 -2.61480966e+23 -6.05233227e+22
-2.56749382e+23 -9.06635585e+22 -6.83378597e+21  1.88138273e+23
3.84425641e+23  9.91976570e+21  1.37210833e+22  6.89403772e+22
6.40033437e+22  1.66340517e+22  2.33568751e+22 -7.52888047e+20
-1.44519213e+21 -1.02218730e+20 -2.32087041e+17  4.83947978e+16
3.49273905e+14  3.20916032e+11  2.17733168e+07]
valori singolari di [A b]: [8.10391794e+00 4.80299503e+00 3.56688320e+00 1.069972

```

```

6.21997651e-01 1.74973249e-01 5.97788600e-02 2.16583328e-02
6.49741139e-03 1.76920203e-03 4.45971645e-04 1.07986436e-04
2.56456277e-05 5.64819272e-06 1.15781357e-06 2.25932827e-07
4.00822675e-08 7.10504111e-09 1.13193674e-09 1.70716660e-10
2.33664403e-11 2.96073165e-12 3.46005027e-13 3.45703571e-14
3.59504269e-15 5.17960834e-16 5.17960834e-16 5.17960834e-16
5.17960834e-16 5.17960834e-16 5.17960834e-16 4.24641425e-16]

```



```

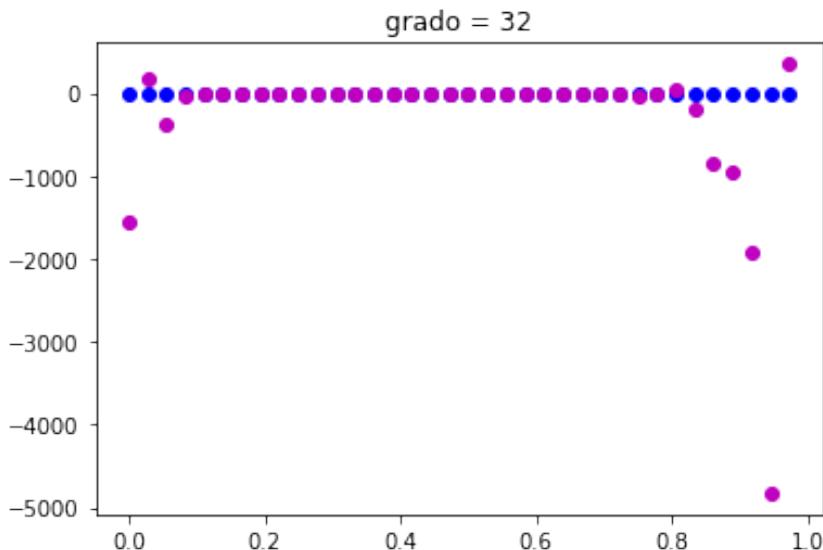
parametri stimati: [[-2.07813074e+17  2.33903829e+18 -1.21772181e+19  3.87132061e
-8.33668502e+19  1.26891431e+20 -1.36772147e+20  9.85512607e+19
-3.56554412e+19 -1.15499764e+19  2.37950868e+19 -1.32730690e+19
1.74015261e+18  1.09462368e+18  1.51096655e+18 -3.78483173e+18
3.75464952e+18 -2.44988027e+18  1.18845075e+18 -4.50383108e+17
1.36456620e+17 -3.33922407e+16  6.61444467e+15 -1.05645303e+15
1.34758293e+14 -1.35107626e+13  1.03915532e+12 -5.91051485e+10
2.34631234e+09 -5.87865432e+07  7.51422106e+05 -2.30105553e+03]]
varianza delle stime dei parametri: [-4.81699018e+22 -3.99600351e+23 -1.94811747e
6.55455648e+22  9.73282696e+21 -1.55019146e+23 -4.77204922e+22
1.09519682e+23 -6.97124937e+22  5.51405668e+21 -6.73316553e+22
1.63422806e+22  2.17963127e+21  7.01543613e+22 -1.25747067e+22
-3.97273417e+22  4.89070116e+21  3.52071088e+21 -2.13725852e+22
-5.82330320e+21 -9.72854028e+21  5.72973894e+21 -5.07535201e+21

```

```

-4.42787183e+21 -5.78269060e+20 -1.93645646e+18  1.56987788e+18
 3.68220189e+16  1.81626884e+14  1.43613912e+11  9.52219665e+06]
valori singolari di [A b]:  [8.10766411e+00 4.80411855e+00 3.58124013e+00 1.081951
 6.27307824e-01 1.80071520e-01 6.12612830e-02 2.24896525e-02
 6.81970186e-03 1.87994156e-03 4.79942772e-04 1.18352846e-04
 2.85955931e-05 6.38071353e-06 1.33726311e-06 2.65359575e-07
 4.83631536e-08 8.80824198e-09 1.44122579e-09 2.23916148e-10
 3.17320586e-11 4.19117524e-12 5.12125074e-13 5.51991682e-14
 5.49927276e-15 5.15917665e-16 5.15917665e-16 5.15917665e-16
 5.15917665e-16 5.15917665e-16 5.15917665e-16 5.15917665e-16
 4.09648586e-16]

```



```

parametri stimati: [[ 1.63510034e+17 -1.75719608e+18  8.58713166e+18 -2.49798975e
 4.71330628e+19 -5.74131989e+19  3.72139067e+19  9.88235345e+18
-5.60927985e+19  7.57761414e+19 -6.72018155e+19  4.58437251e+19
-2.54750747e+19  1.09400325e+19 -2.29435236e+18 -1.43341008e+18
 1.86767071e+18 -1.00971062e+18  2.37905517e+17  7.38901216e+16
-1.00166947e+17  5.29279129e+16 -1.86524734e+16  4.84169610e+15
-9.54656528e+14  1.43851446e+14 -1.64252830e+13  1.39089669e+12
-8.40964801e+10  3.40363753e+09 -8.19499891e+07  9.03070127e+05
-1.55407992e+03]]

```

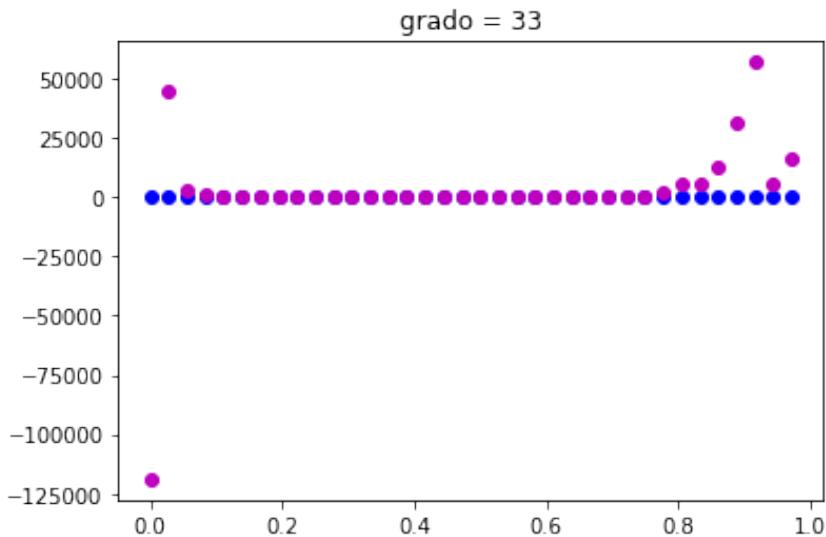
```
varianza delle stime dei parametri: [ 2.47218035e+22  2.46119861e+23  1.86177284e
```

```

-2.31819089e+23 -5.20857617e+22 -1.05480745e+23 -9.87792947e+22
6.11158420e+22  1.01471192e+23  5.13581171e+22  1.79328410e+22
7.62387910e+22  1.03408731e+23  1.08112917e+23  -9.03784194e+22
-2.54787872e+22  1.37620822e+23  1.90379654e+23  6.48008431e+20
-2.17363222e+22  3.49246089e+22  1.81870651e+21  6.13223184e+22
6.04154263e+22  1.24164795e+22  8.02672782e+20  3.26840252e+19
1.66467784e+18  3.49518003e+16  1.80462702e+14  1.50463420e+11
1.03562791e+07]

valori singolari di [A b]: [8.11111426e+00 4.80517259e+00 3.59457759e+00 1.093467
6.32187942e-01 1.85031427e-01 6.26956452e-02 2.33028253e-02
7.13638054e-03 1.99045479e-03 5.14374965e-04 1.29065380e-04
3.16649077e-05 7.15425829e-06 1.53093684e-06 3.08328468e-07
5.77019820e-08 1.07554197e-08 1.80608290e-09 2.87843890e-10
4.21412048e-11 5.78088249e-12 7.35848267e-13 8.44892821e-14
8.58396881e-15 3.59827616e-15 5.13910716e-16 5.13910716e-16
5.13910716e-16 5.13910716e-16 5.13910716e-16 5.13910716e-16
5.13910716e-16 3.65942120e-16]

```



```

parametri stimati: [[ 1.39842894e+18 -1.44454440e+19  6.92800222e+19 -2.04393787e
4.10357620e+20 -5.68202750e+20  4.65141190e+20  6.88123091e+19
-9.55840278e+20  1.78167962e+21 -2.02934850e+21  1.52505639e+21
-6.24010176e+20 -1.15673959e+20  4.04765912e+20 -3.42051712e+20
1.75122427e+20 -5.85459062e+19  1.28610660e+19 -3.34748974e+18

```

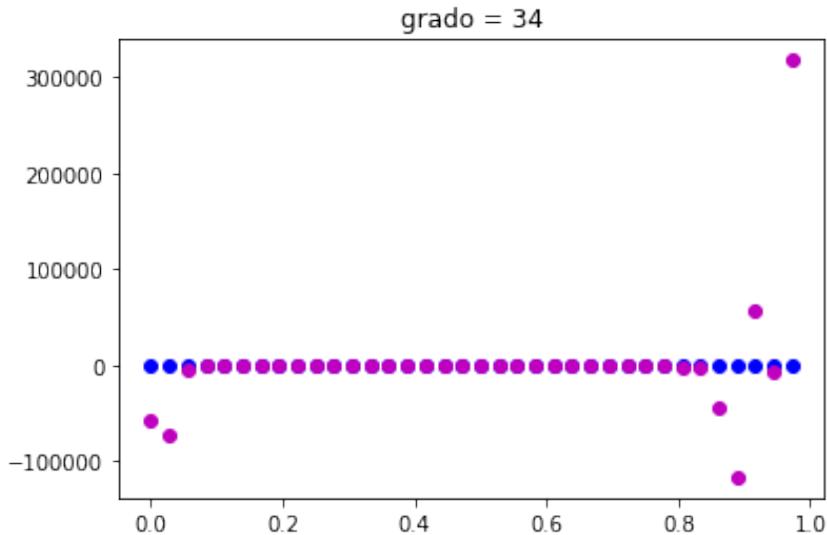
```

2.63717062e+18 -1.96586117e+18 9.91951986e+17 -3.57515594e+17
9.62732830e+16 -1.97872153e+16 3.12048035e+15 -3.75014123e+14
3.37164420e+13 -2.19438271e+12 9.78159114e+10 -2.70479156e+09
3.74963179e+07 -1.18904997e+05]]]

varianza delle stime dei parametri: [-6.68193472e+24 9.14579272e+24 -5.12826952e+24
-2.78977717e+25 -7.89815419e+25 4.14618548e+25 4.55872474e+25
2.04806196e+25 -1.93357147e+25 -7.11434285e+24 6.03910696e+25
2.18476512e+25 -2.74375205e+25 1.82153069e+25 1.14643331e+25
-2.37228852e+25 -7.74564019e+25 -3.79434366e+24 1.12490793e+25
-7.03529600e+24 -5.77348975e+24 1.12692701e+25 -7.05456565e+24
-1.04873617e+25 -2.59060853e+25 -1.34708164e+25 -2.22734461e+24
-1.24263353e+23 -1.79808677e+21 7.37542974e+18 1.22239995e+17
1.33804043e+14 1.03376516e+10]

valori singolari di [A b]: [8.11429542e+00 4.80616186e+00 3.60697909e+00 1.104522e+00
6.36684186e-01 1.89852643e-01 6.40835491e-02 2.40978109e-02
7.44705091e-03 2.10056427e-03 5.49208950e-04 1.40093524e-04
3.48416287e-05 7.96759832e-06 1.73841715e-06 3.54828178e-07
6.81388324e-08 1.29557177e-08 2.23110100e-09 3.63419699e-10
5.48732364e-11 7.79389529e-12 1.03049442e-12 1.25005625e-13
1.30355190e-14 3.59766035e-15 5.11939540e-16 5.11939540e-16
5.11939540e-16 5.11939540e-16 5.11939540e-16 5.11939540e-16
5.11939540e-16 5.11939540e-16 3.95868440e-16]

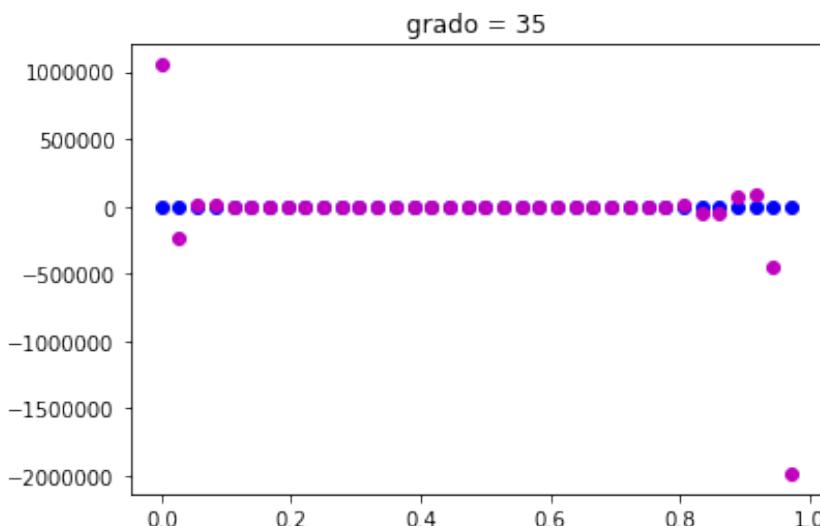
```



```

parametri stimati: [[-1.83946816e+19  2.07965540e+20 -1.09103217e+21  3.51948760e
-7.79582945e+21  1.25325002e+22 -1.50429099e+22  1.35866513e+22
-9.00685554e+21  3.81433051e+21 -8.39665073e+19 -1.52802274e+21
1.44613806e+21 -5.60370567e+20 -2.64685975e+20  5.82851590e+20
-4.60078960e+20  2.00081832e+20 -2.05331524e+19 -4.07483804e+19
3.73406574e+19 -1.96670911e+19  7.56945816e+18 -2.28239654e+18
5.55765649e+17 -1.10908962e+17  1.82316974e+16 -2.45899914e+15
2.68197618e+14 -2.30076625e+13  1.48338576e+12 -6.67480017e+10
1.82989168e+09 -2.14835144e+07 -5.77459487e+04]]
varianza delle stime dei parametri: [-2.94495221e+25 -2.42244984e+26 -7.03162970e
-4.60702881e+26 -1.81306978e+26 -1.34157270e+27  1.11927084e+26
-7.35184671e+26 -5.05974150e+26  1.33515636e+26 -9.55947888e+25
-5.92289562e+26  1.65194580e+25 -7.58085696e+26 -7.67004054e+25
2.04953288e+26 -9.49686050e+25 -4.30535122e+25 -4.98990049e+26
1.05084326e+26 -1.60207270e+26  2.70898837e+25  3.29786971e+26
1.77349768e+26  9.63684923e+25 -8.08579782e+24 -1.51162941e+25
2.16008657e+24  8.22659840e+23  4.80451137e+22  7.75965250e+20
3.18612931e+18  2.19850566e+15  1.27244171e+11]
valori singolari di [A b]: [8.11723172e+00  4.80709065e+00  3.61851997e+00  1.115124
6.40836813e-01 1.94535334e-01 6.54264991e-02 2.48746094e-02
7.75139371e-03 2.21011426e-03 5.84387664e-04 1.51405028e-04
3.81142685e-05 8.81941220e-06 1.95915889e-06 4.04842822e-07
7.97024264e-08 1.54169990e-08 2.72040822e-09 4.51487243e-10
7.02132645e-11 1.02977816e-11 1.41023960e-12 1.79190450e-13
1.91555043e-14 3.59620239e-15 5.10003734e-16 5.10003734e-16
5.10003734e-16 5.10003734e-16 5.10003734e-16 5.10003734e-16
5.10003734e-16 5.10003734e-16 5.10003734e-16 3.80885245e-16]

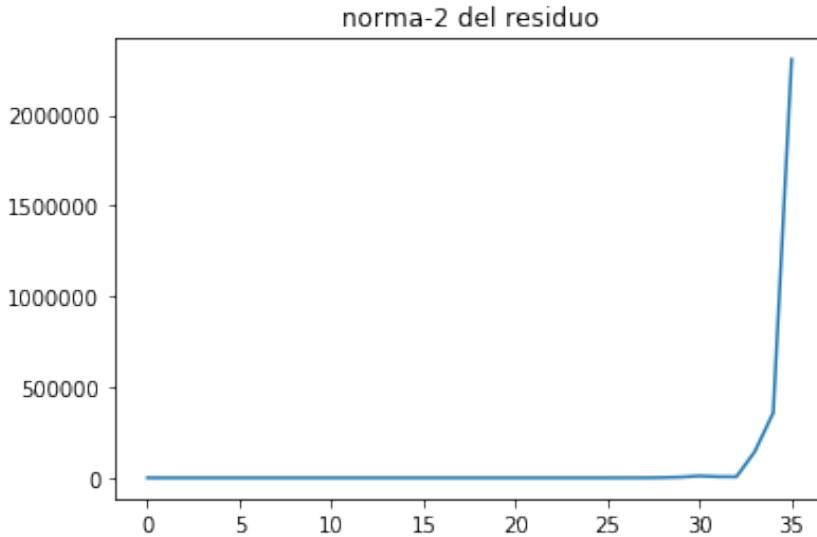
```

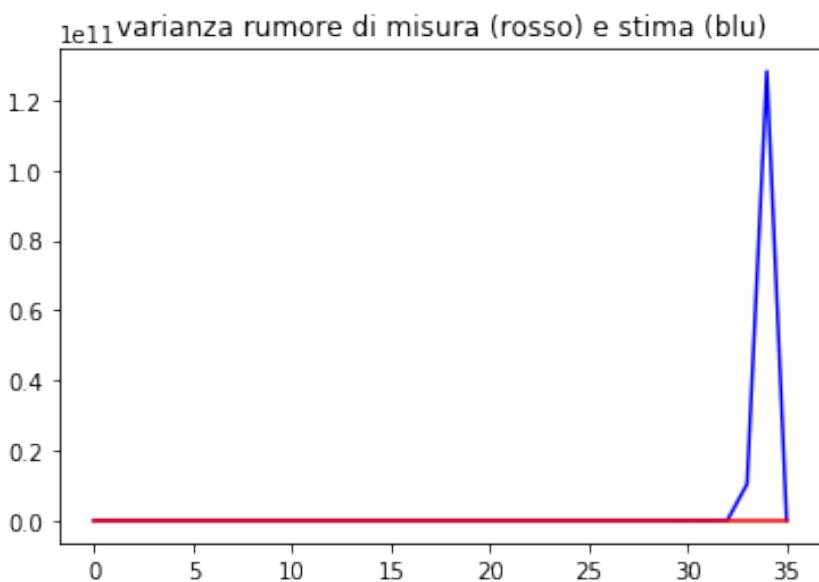


```

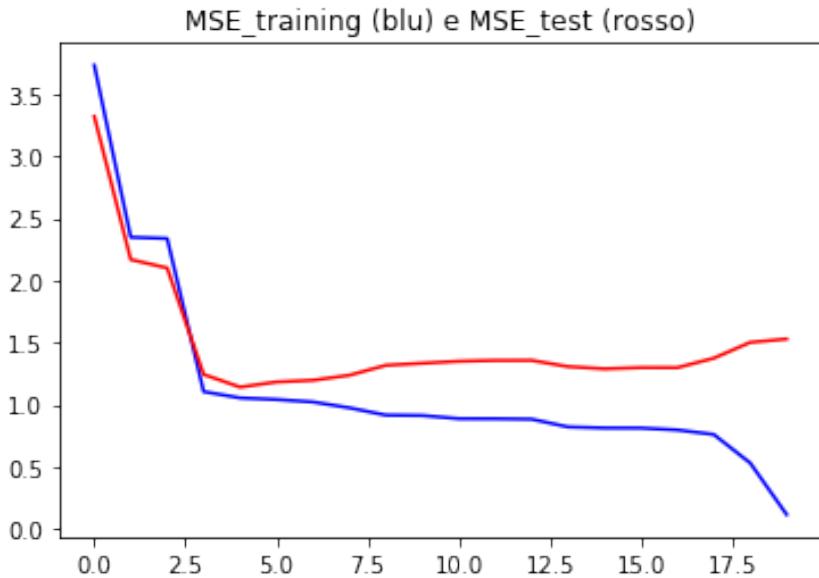
parametri stimati: [[ 1.47051826e+19 -1.90326507e+20  1.13883634e+21 -4.17737003e
 1.04911174e+22 -1.90561989e+22  2.57285729e+22 -2.60165335e+22
 1.93638588e+22 -9.74093341e+21  1.79747280e+21  2.55613812e+21
-3.90537194e+21  3.40564551e+21 -1.94410798e+21  3.12551481e+20
 8.14343994e+20 -1.18191595e+21  9.91933572e+20 -6.10467005e+20
 2.90929599e+20 -1.07466722e+20  2.92502343e+19 -4.73419068e+18
-2.68364718e+17  4.87557491e+17 -1.88609172e+17  4.70815538e+16
-8.55847296e+15  1.16304610e+15 -1.17477665e+14  8.58591625e+12
-4.31260430e+11  1.35576815e+10 -2.20114153e+08  1.05081760e+06]]
varianza delle stime dei parametri: [ 0.  0.  0.  0.  0.  -0.  -0.  -0.  0.  -0.  -0.  -0.
-0.  0.  -0.  0.  -0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
valori singolari di [A b]: [8.11994474e+00 4.80796291e+00 3.62926839e+00 1.125278
 6.44680960e-01 1.99080091e-01 6.67259169e-02 2.56332554e-02
 8.04915918e-03 2.31896781e-03 6.19855594e-04 1.62966488e-04
 4.14719564e-05 9.70825534e-06 2.19251469e-06 4.58352124e-07
 9.24094731e-08 1.81460815e-08 3.27753054e-09 5.52816653e-10
 8.84499365e-11 1.33626628e-11 1.89065882e-12 2.49727698e-13
 2.75829911e-14 3.27292322e-15 5.97627611e-16 5.97627611e-16
 5.97627611e-16 5.97627611e-16 5.97627611e-16 5.97627611e-16
 5.97627611e-16 5.97627611e-16 5.97627611e-16 5.97627611e-16]

```





```
In [5]: # validazione del modello: "validation set"
# consideriamo due insiemi di dati, formati ciascuno da una sola serie storica (in
# il primo, "br", lo usiamo per il training del modello, mentre il secondo, "br_test"
N = 20
maxgp = N-1
vk = np.arange(0.,1.,1./N)
b = np.sin(2*np.pi*vk)
stderr = 0.3
br = np.asmatrix( b + stderr*np.random.randn(N) ).T
br_test = np.asmatrix( b + stderr*np.random.randn(N) ).T
validation_set(vk,br,br_test,maxgp)
```

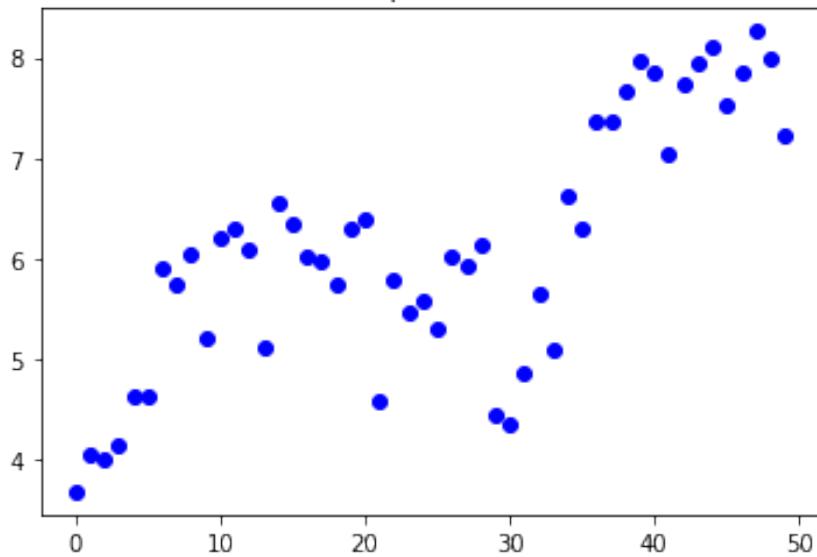


Torna al par. 9.1.2.1

U.1 Esempio Hodrick-Prescott

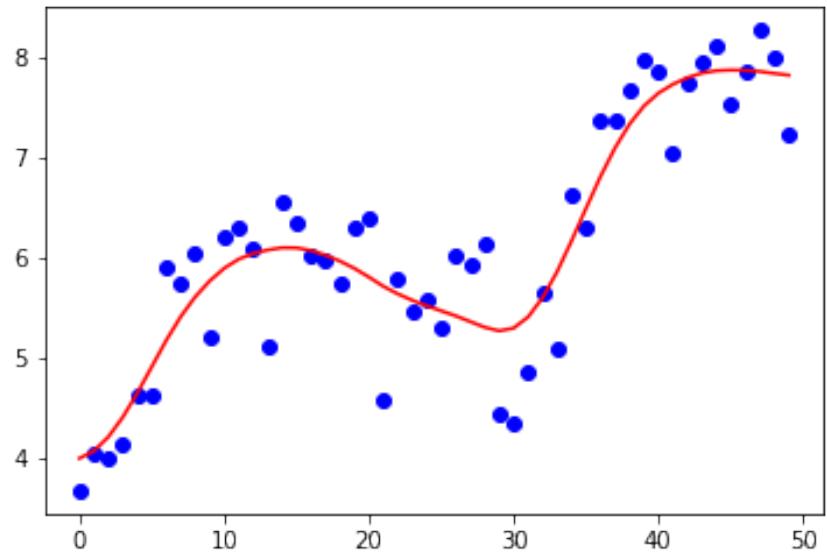
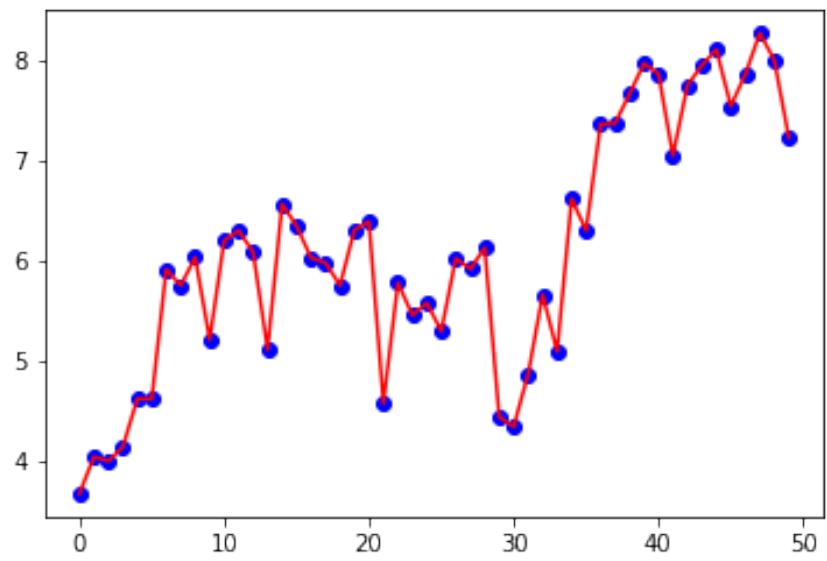
```
In [6]: stderr = 0.5
br = np.concatenate((3.0+3.0*np.arange(0.1,1.1,0.1) + stderr*np.random.randn(10), \
                     6.0+0.1*np.arange(0.1,1.1,0.1) + stderr*np.random.randn(10), \
                     6.1-1.1*np.arange(0.1,1.1,0.1) + stderr*np.random.randn(10), \
                     5.0+3.0*np.arange(0.1,1.1,0.1) + stderr*np.random.randn(10), \
                     8.0+0.0*np.arange(0.1,1.1,0.1) + stderr*np.random.randn(10)))
N = len(br)
plt.figure(1), plt.plot(br,'bo'); plt.title('dati sperimentali'); plt.show()
br = np.atleast_2d(br).T
```

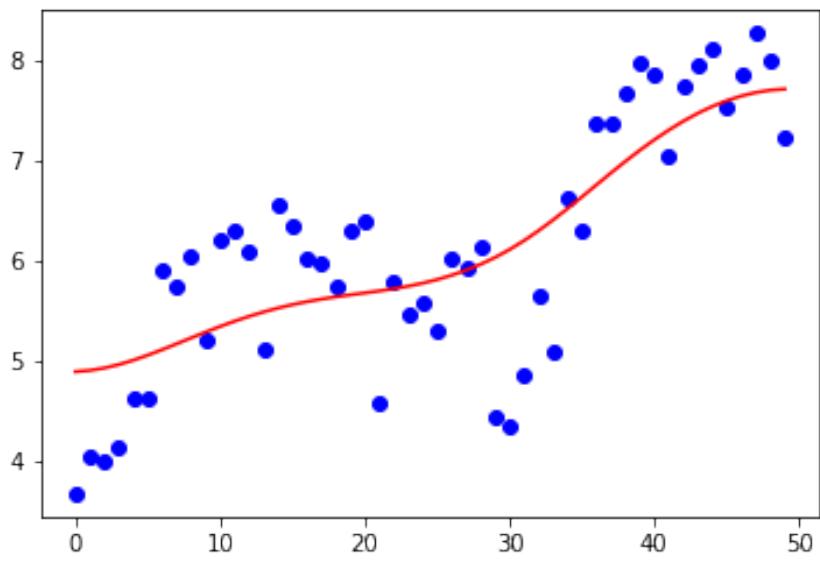
dati sperimentali



```
In [7]: d0 = -2.0 + np.zeros(N)
d1 = 1.0 + np.zeros(N-1)
D = np.diag(d0) + np.diag(d1,-1) + np.diag(d1,1)
D[0,0] = -1.0
D[N-1,N-1] = -1.0
```

```
In [8]: for lam in [0, 20, 1000]:
    x_hp = np.linalg.solve( np.eye(N)+2*lam*D, br)
    plt.figure(1); plt.plot(br, 'bo'); plt.plot(x_hp, 'r-')
#endfor
```





Torna al par. 9.1.4

Appendice V

Stima parametri DLTI state-space dalla risposta all'impulso discreto

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#%get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.freqresp import *
from libreria_NLALD.sistemi_DLTI import *
import numpy.matlib as npmat
from scipy.linalg import hankel

In [2]: # stima dei parametri di un modello state-space dalla risposta al campione unitario
N = 32
# sequenza "campione unitario" di lunghezza N:
u=np.zeros(N); u[0]=1.0
if False:
    # calcolo la risposta del sistema meccanico precedentemente visto:
    param_sistema_meccanico_3gdl;
    build_sistema_meccanico_3gdl;
    load = u.T
    simula_sistema_meccanico_3gdl; # NB: calcola "response".
    h = response; # in questo caso la risposta del sistema è la risposta al gradino
    nx=6;
else:
    nx=6
x0 = np.atleast_2d(np.zeros(nx)).T
```

```

A = np.random.rand(nx,nx)
b = np.atleast_2d(np.random.rand(nx)).T
c = np.atleast_2d( np.random.rand(nx) )
h, X_hist = simula_DLTI_StateSpace_discreto(A,b,c,[0],np.atleast_2d(u),x0);
h = np.squeeze(h)
#endif

```

```

In [3]: for na in range(1,10):
    H = hankel(h[1:na+2], h[na+1:na+1+na])
    print("na=",na," rank(H)=",np.linalg.matrix_rank(H))
#endif
# uso la "h" per identificare un modello state-space di ordine nx=6:
H = hankel(h[1:nx+2], h[nx+1:2*nx+1])

```

```

na= 1      rank(H)= 1
na= 2      rank(H)= 2
na= 3      rank(H)= 3
na= 4      rank(H)= 4
na= 5      rank(H)= 5
na= 6      rank(H)= 6
na= 7      rank(H)= 6
na= 8      rank(H)= 6
na= 9      rank(H)= 6

```

```

In [4]: # metodo di realizzazione in forma controller-Hessenberg:
O,R = np.linalg.qr(H.copy())
O_1 = O[0:nx,0:nx]
O_2 = O[1:nx+1,0:nx]
# Dato che O_1 * A_c = O_2 e le matrici hanno rango pieno, si ha:
A_c = np.linalg.solve(O_1, O_2)
b_c = np.atleast_2d(R[0:nx,0]).T
c_c = np.atleast_2d(O[0,0:nx])
#
x0 = np.atleast_2d(np.zeros(nx)).T
h_c, X_hist = simula_DLTI_StateSpace_discreto(A_c,b_c,c_c,[0],np.atleast_2d(u),x0);
h_c = np.squeeze(h_c)

```

```

In [5]: # metodo di realizzazione in forma bilanciata:
# Nota: dato che calcola la SVD di H, è dei due il miglior metodo per determinare "
U,S,V = np.linalg.svd(H.copy())
V = V.T
sq = np.sqrt(S[0:nx])
O = U[0:nx+1,0:nx] @ np.diag(sq)
O_1 = O[0:nx,0:nx]
O_2 = O[1:nx+1,0:nx]
R = np.diag(sq) @ V[0:nx,0:nx].T

```

```

A_b = np.linalg.solve(0_1, 0_2)
b_b = np.atleast_2d(R[0:nx,0]).T
c_b = np.atleast_2d(0[0,0:nx])
#
x0 = np.atleast_2d(np.zeros(nx)).T
h_b, X_hist = simula_DLTI_StateSpace_discreto(A_b,b_b,c_b,[0],np.atleast_2d(u),x0);
h_b = np.squeeze(h_b)

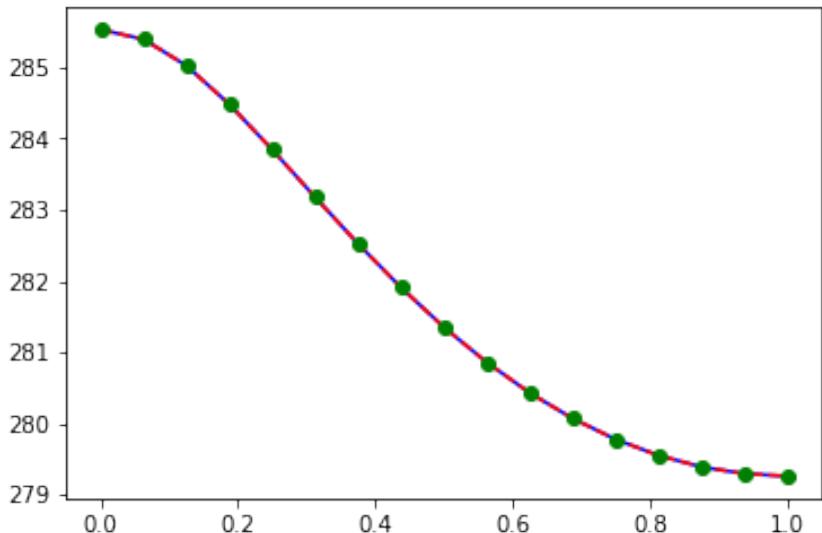
```

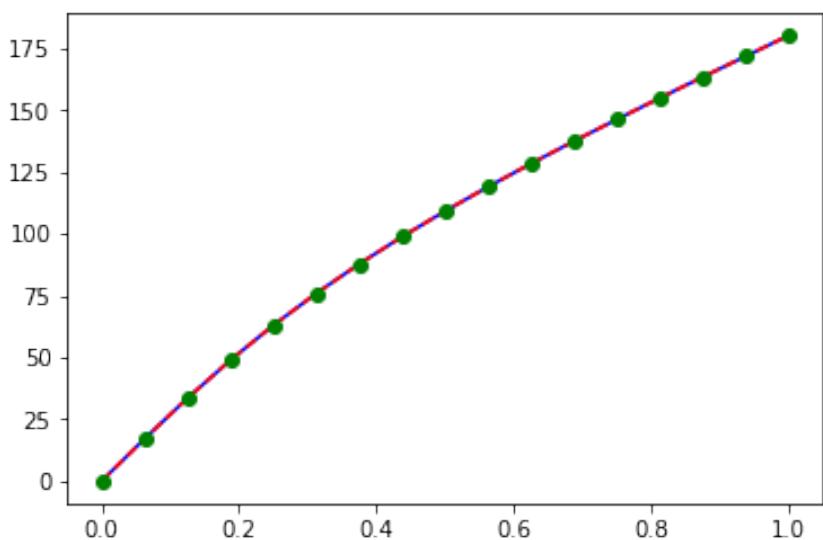
In [6]: # verifico le risposte in frequenza:

```

plt.figure(1);
freqresp(h, 'b-',True);
freqresp(h_c, 'r--',True);
freqresp(h_b, 'go',True);

```





Torna al par. [9.2.2](#)

Appendice W

Stima parametri DLTI ARMA tempo-varianti dalla risposta all'impulso discreto

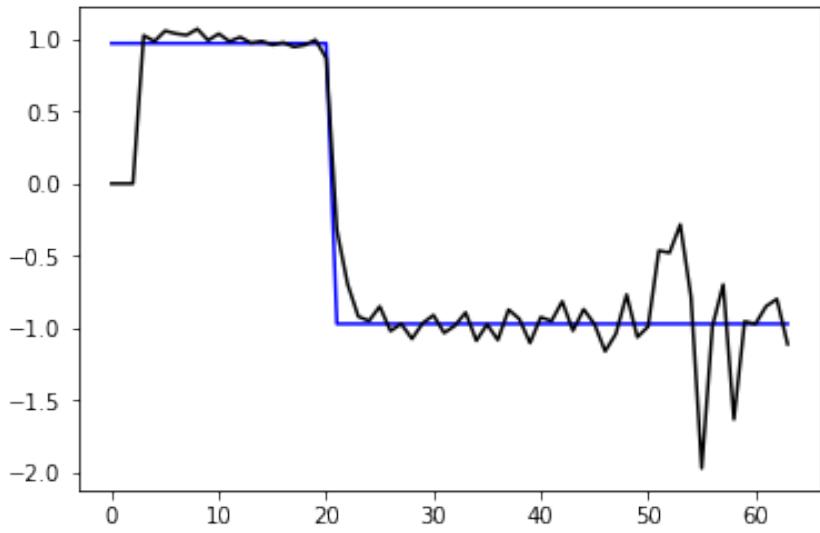
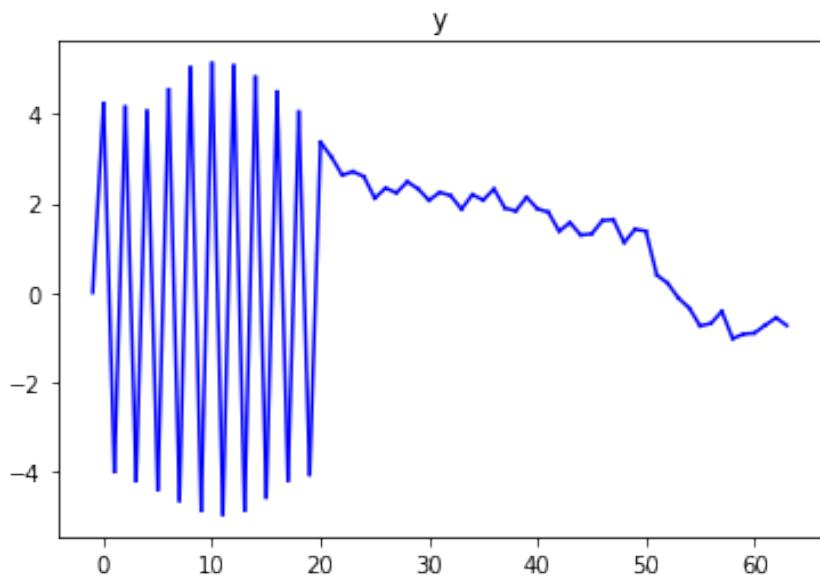
```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

In [2]: # definiamo un sistema del primo ordine:
a = np.array([1., 0.97])
na = len(a) - 1
b = np.array([4.])
# condizioni iniziali :
u_m1 = 0.
y_m1 = 0.
y_m2 = 0.
# scelgo se aggiungere il rumore ai dati:
aggiungo_rumore = True
# coefficiente di oblio (lambda):
lam = 0.5 # 0.5 - 0.99
Ndati = 64
u = np.zeros(Ndati)
y = np.zeros(Ndati)
plt.figure(1); plt.title('y');
a_est = np.zeros((na,Ndati))
a_hist = np.zeros((na+1,Ndati))
w = np.random.randn(Ndati)
```

```

for N in range(Ndati):
    #print("N = ",N)
    # legge di variazione dei parametri:
    if N>20: a[1] = -0.97 #endif
    a_hist[1,N] = a[1]
    # calcolo l'ingresso (sequenza di kronecker):
    if N==0:
        u[N]=1.
    else:
        u[N]=0.
    #endif
    # aggiorno l'uscita del sistema
    y[N] = 1./a[0] * ( -a[1]*y_m1 + b[0]*u[N] )
    if aggiungo_rumore:
        y[N] = y[N] + 0.3*w[N]      # 0.3
    #endif
    plt.plot([N-1, N],[y_m1, y[N]],'b-');
    # calcolo/aggiorno le stime
    istante_prima_stima = (na+1)+na
    if N > istante_prima_stima: # aggiorno
        A = lam * A; d = lam * d;
        A = np.vstack((A, y[N-na:N-1+1]))
        d = np.vstack((d, -y[N]))
        Q,R = np.linalg.qr(A.copy())
        a_est[:,N] = np.linalg.solve( R , Q.T@d )
    elif N == istante_prima_stima: # calcolo la prima volta, risolvendo il sistema (NB: è >= al primo "N" per cui la matrice del sistema risulta sempre quadrata)
        A = np.vstack((A, y[N-na:N-1+1]))
        d = np.vstack((d, -y[N]))
        Q,R = np.linalg.qr(A.copy())
        a_est[0,N] = np.linalg.solve( R , Q.T@d )
    elif N > na+1:
        A = np.vstack((A, y[N-na:N-1+1]))
        d = np.vstack((d, -y[N]))
    elif N == na+1:
        A = y[N-na:N-1+1]
        d = np.array(-y[N])
    #endif
    # aggiorno gli elementi di ritardo (NB: da quello più lontano nel tempo !!!)
    u_m1 = u[N]
    y_m2 = y_m1
    y_m1 = y[N]
#endfor
plt.show()
plt.figure(10); plt.plot(np.squeeze(np.array(a_hist[1,:])), 'b-'); plt.plot(np.squeeze(np.array(a_hist[2,:])), 'r-');

```



Torna al par. 9.2.1

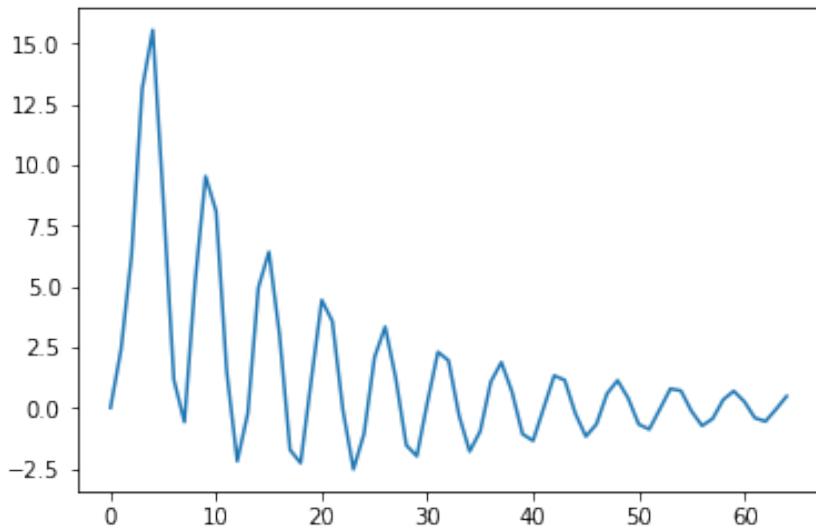
Appendice X

Stima parametri DLTI ARMA dalla risposta all'impulso discreto

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#%get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.freqresp import *
from libreria_NLALD.sistemi_DLTI import *
import numpy.matlib as npmat
from scipy.linalg import hankel
```

```
In [2]: # stima dei parametri di un modello ARMA dalla risposta al campione unitario
#
# definiamo il modello ARMA in modo arbitrario:
a = np.poly([0.2, 0.4+0.87j, 0.4-0.87j, -0.1+0.1j, -0.1-0.1j, -0.89, 0.88])
na = len(a) - 1
b = np.array([0.0, 2.4, 4.4, 8.4, 7.4])
nb = len(b) - 1
```

```
In [3]: # calcolo la risposta al campione unitario:
N = 64
u = np.zeros(N+1); u[0] = 1.0
h = simula_DLTI(b,a,u)
plt.figure(1); plt.plot(h);
```



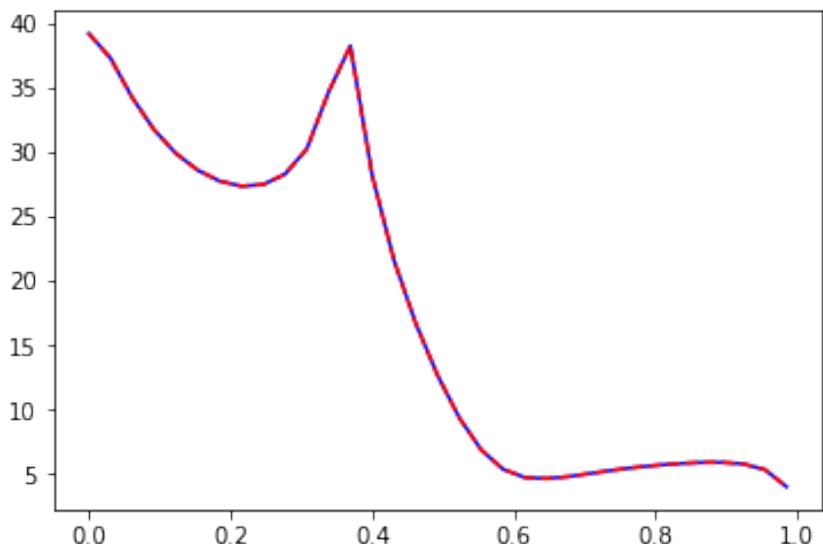
```
In [4]: # costruisco le matrici di Hankel:
A = hankel(h[1:N-na+1], h[N-na:N-1+1])
B = hankel(np.concatenate((np.zeros(na), np.array([h[0]]))), h[0:na+1])
#print A

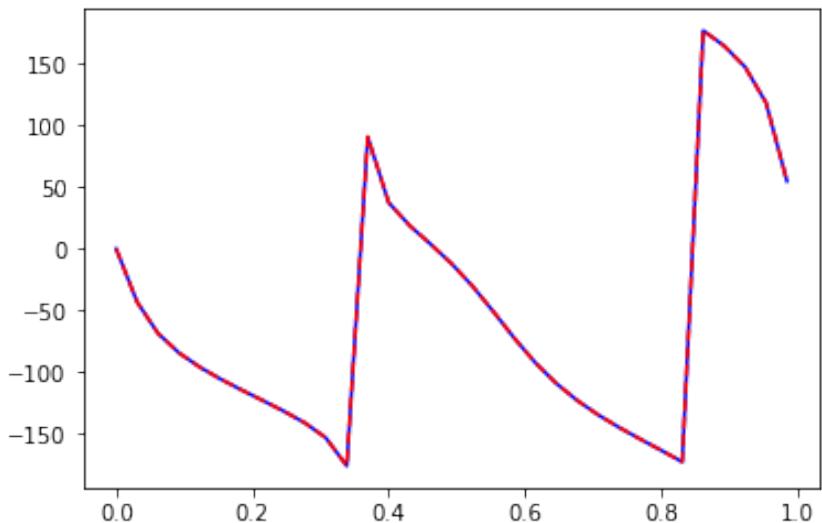
In [5]: # ora stimo i coefficienti "a" e "b" a partire da "h(n)":
d = np.atleast_2d( - h[na+1:N+1] ).T
Q,R = np.linalg.qr(A.copy())
# risolvo il sistema (11.2.10) :
a_est = np.ones(na+1)
a_est[0:na] = np.squeeze( np.linalg.solve( R , Q.T@d ) )
b_est = B @ np.atleast_2d(a_est).T
b_est = np.squeeze( b_est[0:nb+1] )
a_est = a_est[range(na,-1,-1)]
print('a = ', a)
print('a_est = ', a_est)
print('b = ', b)
print('b_est = ', b_est)

a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
  0.01181941  0.00287246]
a_est = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
  0.01181941  0.00287246]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.  2.4 4.4 8.4 7.4]
```

```
In [6]: H = freqresp(h,'b-',False)
        print('confronto delle risposte in frequenza:')
        h_est = simula_DLTI(b_est,a_est,u)
#h_est = simula_DLTI(b_est,a_est[range(na,-1,-1)],u)
H2 = freqresp(h_est,'r--',True)
```

confronto delle risposte in frequenza:





Torna al par. 9.2.1

Appendice Y

Stima parametri DLTI ARMA dalla risposta all'impulso discreto con rumore

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dow
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
#get_ipython().magic(u'matplotlib inline')
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.freqresp import *
from libreria_NLALD.sistemi_DLTI import *
import numpy.matlib as npmat
from scipy.linalg import hankel
from libreria_NLALD.testBartlett_wn import *

In [2]: # stima del livello di rumore dalla risposta all'impulso discreto [par.11.3.1]:
a = np.poly([0.2, 0.4+0.87j, 0.4-0.87j, -0.1+0.1j, -0.1-0.1j, -0.89, 0.88])
na = len(a) - 1
print("na = ", na)
b = np.array([0.0, 2.4, 4.4, 8.4, 7.4])
nb = len(b) - 1
N = 64
stde = 1.0*1.e-4
u = np.zeros(N); u[0]=1.0
noise = stde*np.random.rand(N)
y = simula_DLTI(b,a,u) + noise;
na_est = na+1  # costruisco una matrice di Hankel che ha "na_est" colonne pari al v
                # nel caso deterministico la matrice risulterebbe singolare.
jfig = 1
A = hankel(y[1:N-na_est], y[N-na_est-1:N-1])
```

```

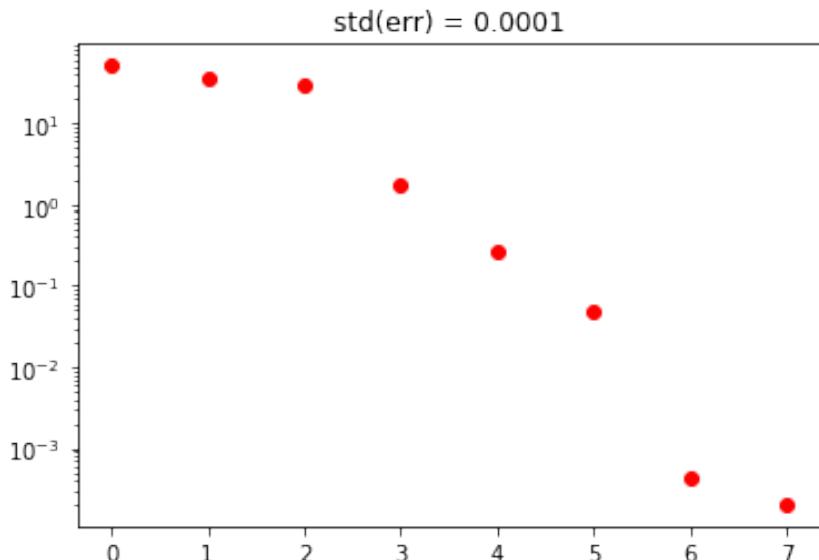
print("A.shape = ",A.shape)
U,S,V = np.linalg.svd(A.copy()); V = V.T
print("Sigma = ",S)
plt.figure(jfig*20+na_est); plt.semilogy(S,'ro'); plt.title('std(err) = ' + str(std))
d = - np.atleast_2d( noise[na_est+1:N] ).T
Q,R = np.linalg.qr(A.copy())
a_est = np.linalg.solve( R , Q.T@d )
print("||a_est|| = ",np.linalg.norm(a_est))
print("a_est = ",a_est)
print(V.T * (a_est / np.linalg.norm(a_est)))
print("stima std rumore = ",S[-1] * (abs(a_est[-1])/np.linalg.norm(a_est)))

```

```

na = 7
A.shape = (55, 8)
Sigma = [5.16097243e+01 3.63376169e+01 3.00495838e+01 1.71539640e+00
2.58874822e-01 4.89805846e-02 4.28886207e-04 2.03053452e-04]

```



```

||a_est|| = 0.11045001651729604
a_est = [[-6.21483304e-04]
          [ 4.25672377e-05]
          [-1.05080246e-03]
          [ 4.98068355e-02]
          [-4.28007704e-02]]

```

```

[-9.59260593e-03]
[ 5.60725572e-02]
[-6.81831162e-02]]
[[ 2.58996858e-03  3.14320397e-03  2.51576599e-03  1.25681628e-03
  5.55944026e-04  9.47407987e-04  1.68766519e-03  1.76351801e-03]
[-1.81780761e-04 -2.13787690e-05  1.87709235e-04  2.07625813e-04
  2.46678669e-05 -1.40447426e-04 -1.11100781e-04  6.33235722e-05]
[ 2.06684969e-03 -2.45438577e-03 -1.00577430e-03  4.10330668e-03
  6.42400590e-03  3.45249438e-03 -1.40782679e-03 -2.68305664e-03]
[ 1.65418698e-01 -2.00685355e-01  2.18759606e-01 -1.54763287e-01
  9.52299955e-02 -2.03795596e-01  7.94613188e-02 -8.36068908e-02]
[-2.08559328e-01 -8.70212067e-02  7.47793423e-02 -6.44956108e-02
  8.68357388e-02  1.49094288e-01  2.34057798e-01  6.92248034e-02]
[-2.66594830e-02  5.21967731e-02  8.29631631e-03 -2.69779320e-02
  3.33953372e-02 -1.19264151e-02 -5.93858897e-03 -4.49258667e-02]
[ 2.11102885e-03 -4.69043382e-02  2.63549014e-01 -9.61820064e-02
  -1.73156001e-01  2.93194106e-01 -2.10931753e-01 -1.27988118e-01]
[-9.04848993e-04 -6.34296058e-03  1.39062419e-02  2.73294275e-01
  -2.55432684e-01 -3.08227148e-02  2.97257412e-01 -3.89362522e-01]]
stima std rumore = [0.00012535]
```

In [3]: `np.linalg.norm(a_est)`

Out [3]: 0.11045001651729604

```

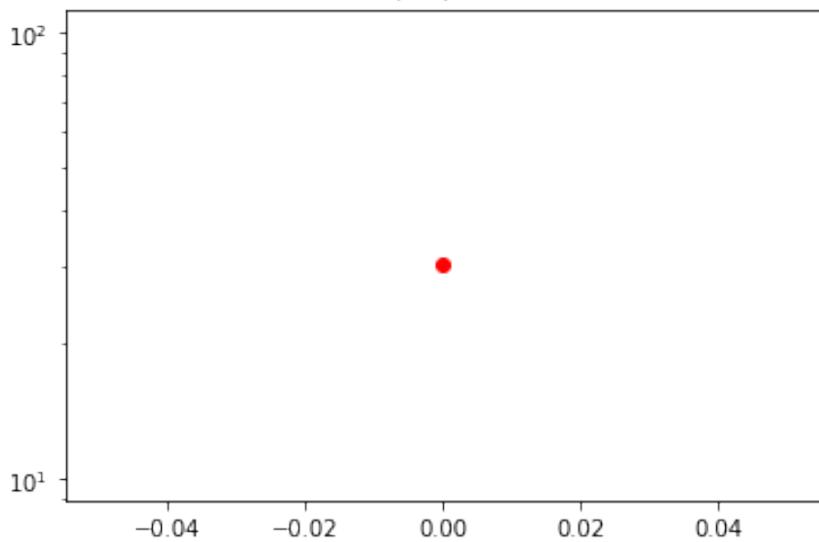
In [4]: a = np.poly([0.2, 0.4+0.87j, 0.4-0.87j, -0.1+0.1j, -0.1-0.1j, -0.89, 0.88])
na = len(a) - 1
print("na = ", na)
b = np.array([0.0, 2.4, 4.4, 8.4, 7.4])
nb = len(b) - 1
N = 64
vstderr = np.array([0., 1e-16, 1e-12, 1e-6, 1e-4, 1e-2, 1e0, 1e1])
jfig = 0
for stde in vstderr:
    print('std = ' + str(stde) + ' :')
    u=np.zeros(N); u[0]=1.
    y = simula_DLTI(b,a,u) + stde*np.random.rand(N) ;
    if stde >= 0.:
        vna = range(1,na+7)
        n2res = np.zeros(len(vna))
        MDL = np.zeros(len(vna))
        for na_est in vna:
            print('### na_est = ',na_est,':')
            A = hankel(y[1:N-na_est], y[N-na_est-1:N-1])
            U,S,V = np.linalg.svd(A.copy())
```

```

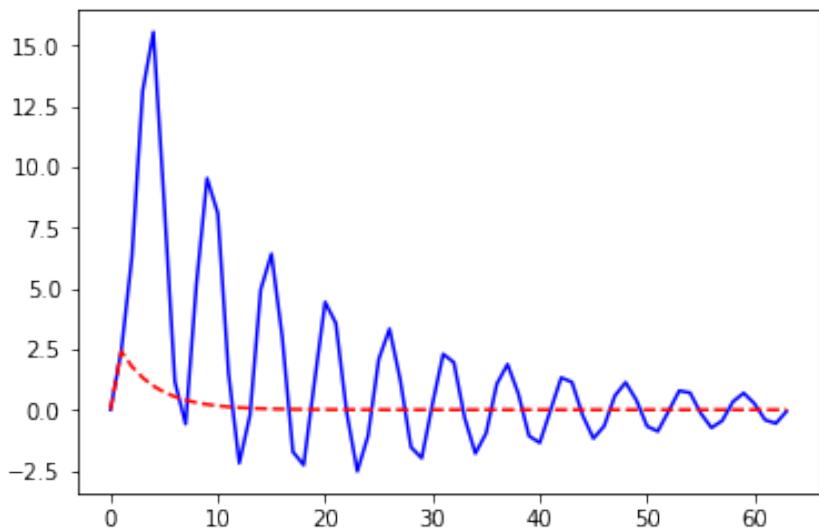
V = V.T
plt.figure(jfig*20+na_est); plt.semilogy(S,'ro'); plt.title('std(err) = ' + s
# ora stimo i coefficienti "a" e "b" a partire da "h(n)" (NB: n = k+1 !) :
d = - np.atleast_2d( y[na_est+1:N] ).T
Q,R = np.linalg.qr(A.copy())
B = hankel(np.concatenate((np.zeros(na_est), np.array([y[0]]))), y[0:na_est+1]
# risolvo il sistema (10.2.10) :
a_est = np.ones((na_est+1,1))
a_est[0:na_est] = np.linalg.solve( R , Q.T@d )
b_est = B @ a_est
b_est = np.squeeze(b_est[0:na_est+1])
a_est = np.squeeze(a_est[range(na_est,-1,-1)])
print('a = ', a)
print('a_est = ', a_est)
print('b = ', b)
print('b_est = ', b_est)
y_est = simula_DLTI(b_est, a_est, u);
n2res[na_est-1] = 1./(N-na_est)*np.sum((y - y_est)**2)
MDL[na_est-1] = n2res[na_est-1]*(1 + (na_est+na_est+1)*np.log(N)/N)
plt.figure(jfig*20+19); plt.plot(range(N),y,'b-'); plt.plot(range(N),y_est,'r-
#endif
plt.figure(); plt.semilogy(vna,n2res,'b*-',label='residuo'); plt.semilogy(vna,M
#if std

```

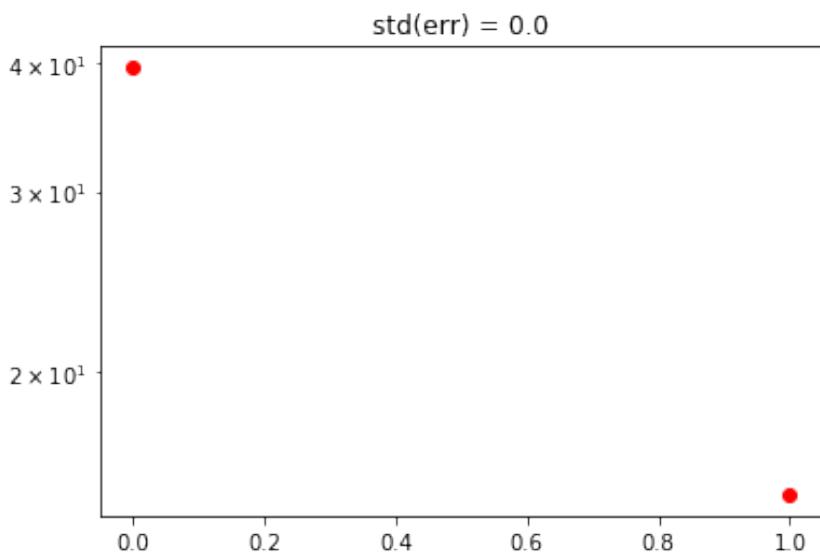
$\text{std}(\text{err}) = 0.0$



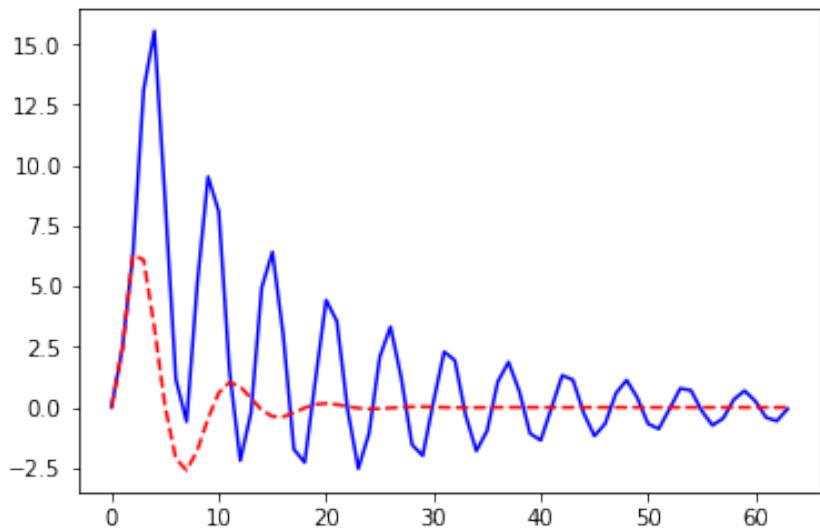
```
a = [ 1.           -0.79           0.1057           0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74094934]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.  2.4]
```



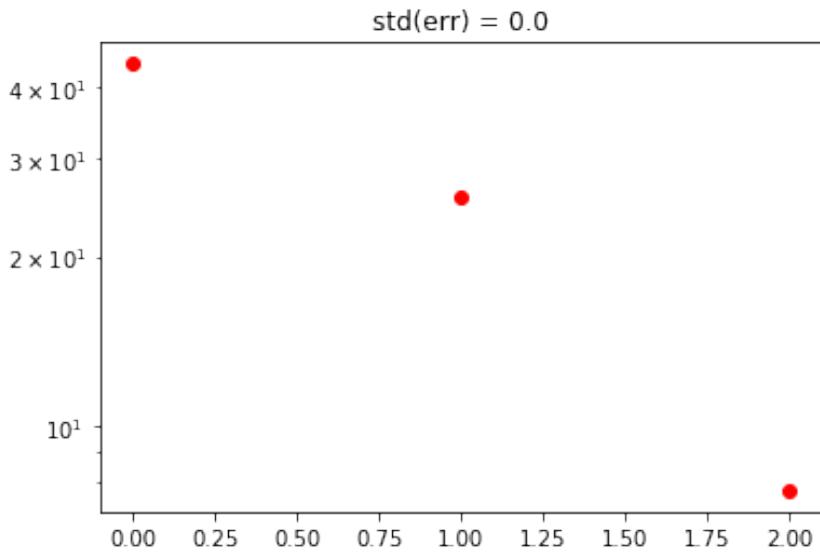
```
### na_est = 2 :
```



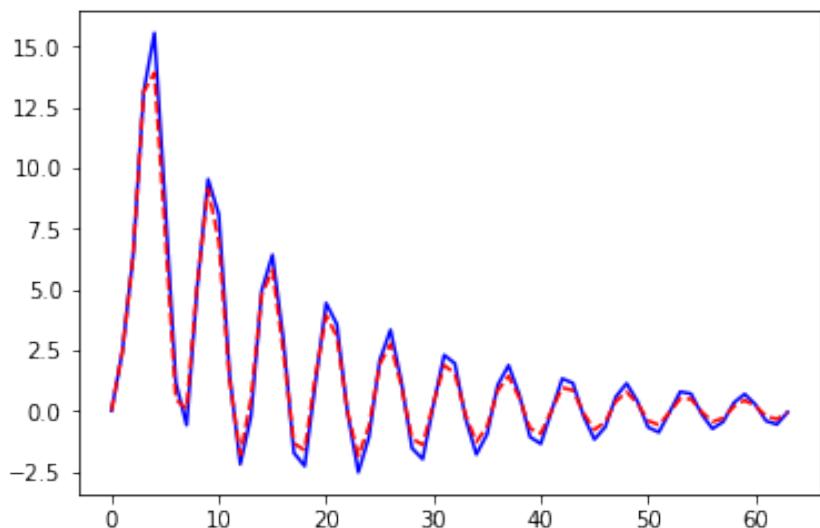
```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21625056  0.65362331]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.          2.4          3.37699865]
```



```
### na_est = 3 :
```

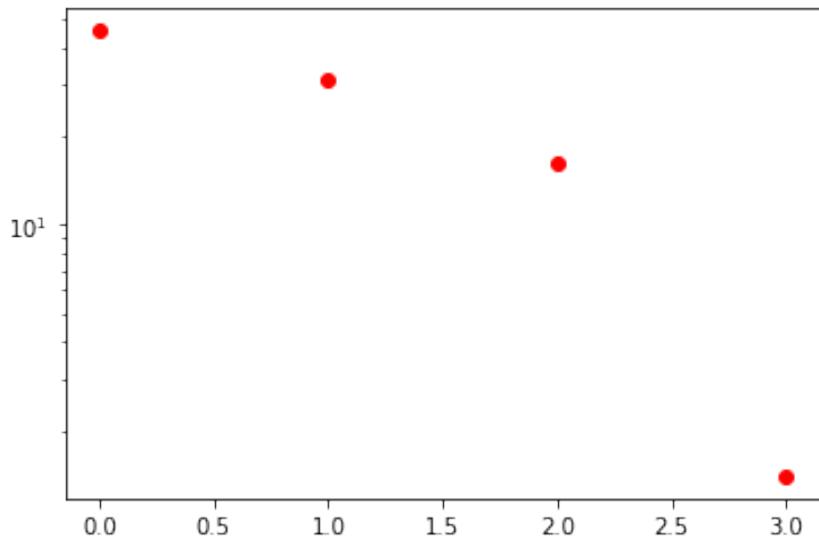


```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -1.68740899  1.61319279 -0.80094766]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.          2.4          2.24621842 6.36789569]
```

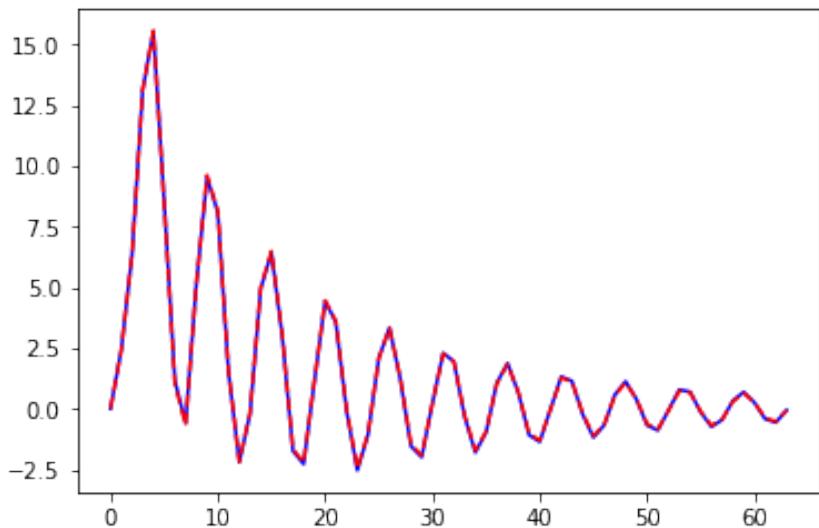


```
### na_est = 4 :
```

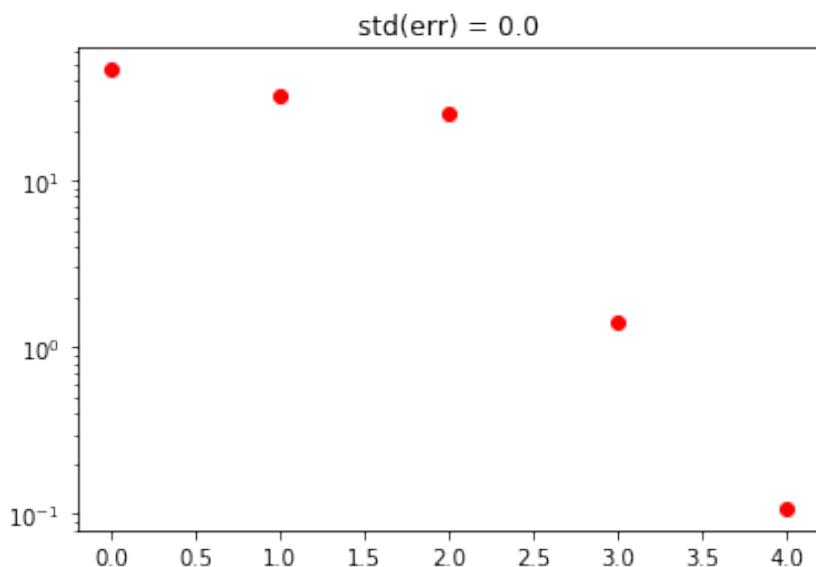
$\text{std}(\text{err}) = 0.0$



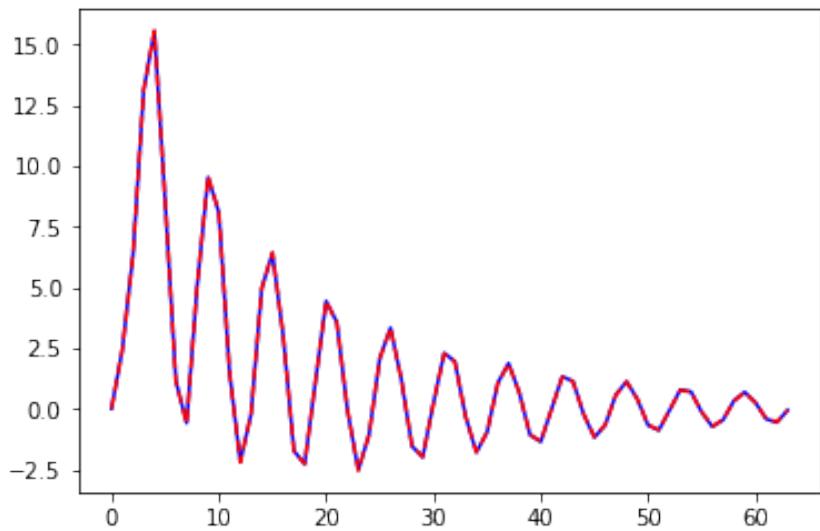
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.79406241  0.12323547  0.6340996   -0.71855887]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.          2.4          4.39025022 8.4165082  7.42487331]
```



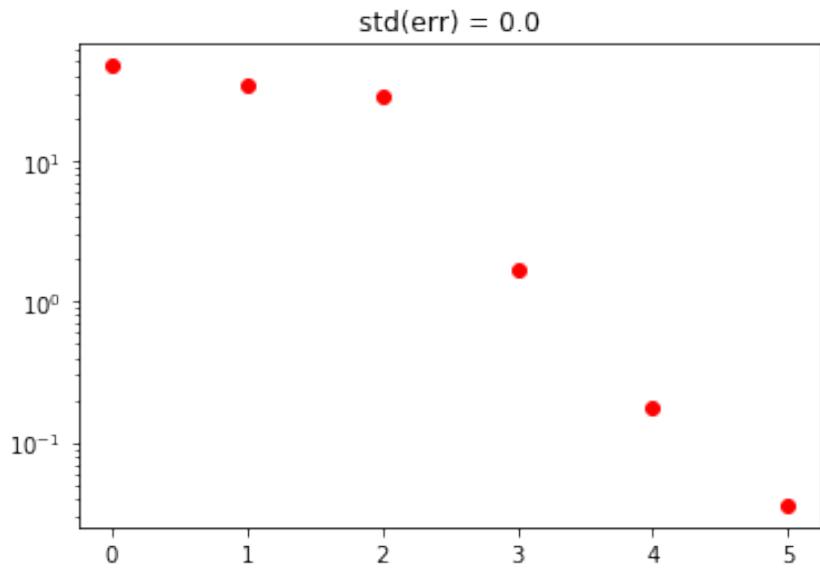
```
### na_est = 5 :
```



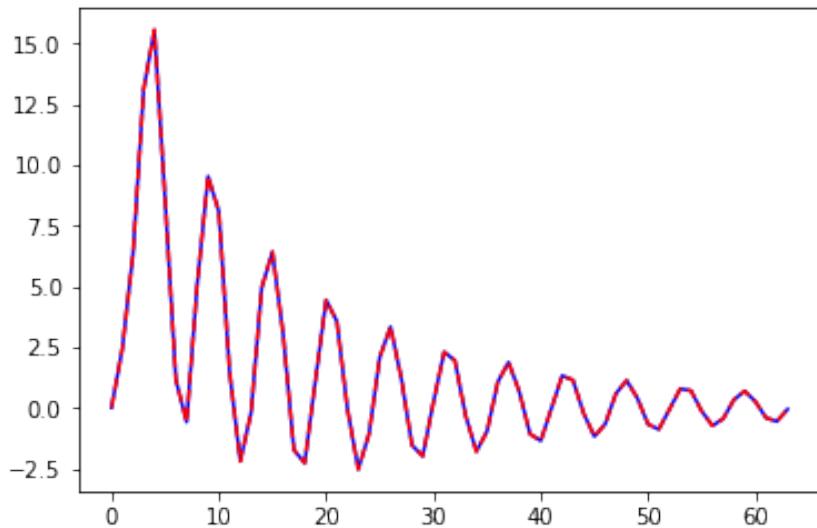
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -1.37353837  0.57138116  0.58688411 -1.11125735  0.43032752]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.          2.4          2.9995079   5.84367718  2.53026402 -4.28841806]
```



```
### na_est = 6 :
```

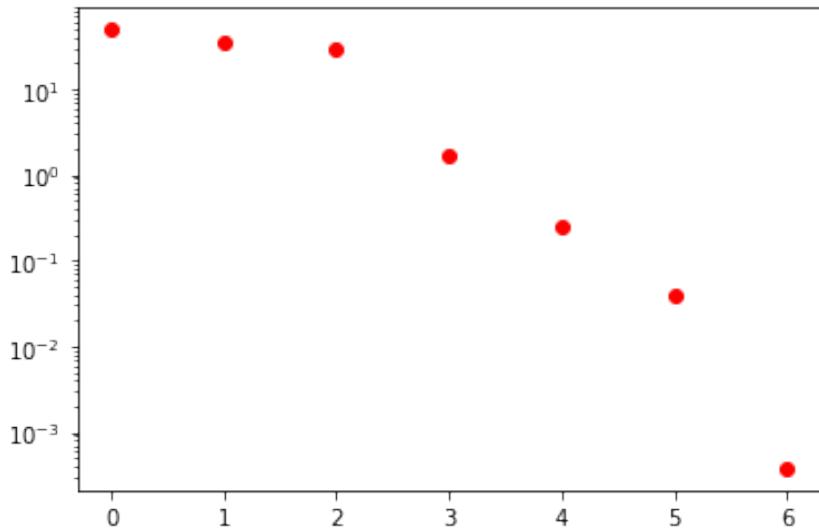


```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.98269039  0.28884512  0.60274659 -0.83901822  0.14508689
      -0.00773825]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.          2.4          3.93754306  7.62636959  5.91747515 -1.16621562
      0.22876174]
```

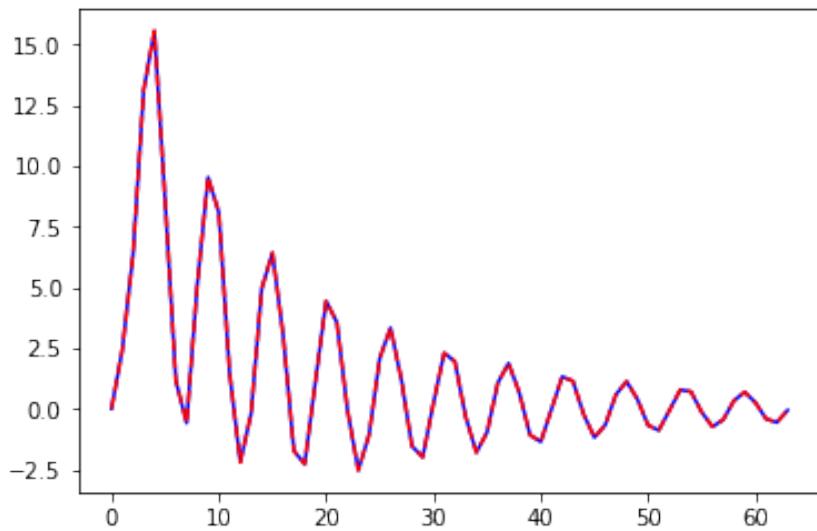


```
### na_est = 7 :
```

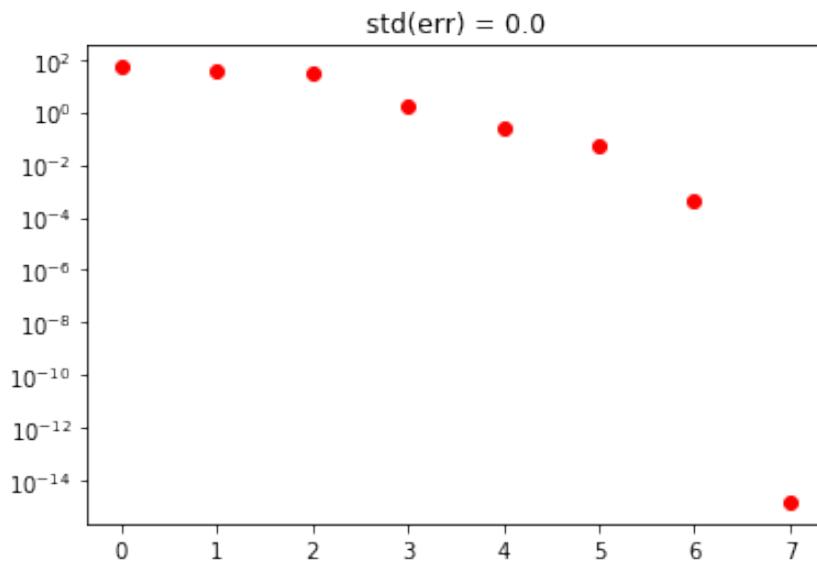
`std(err) = 0.0`



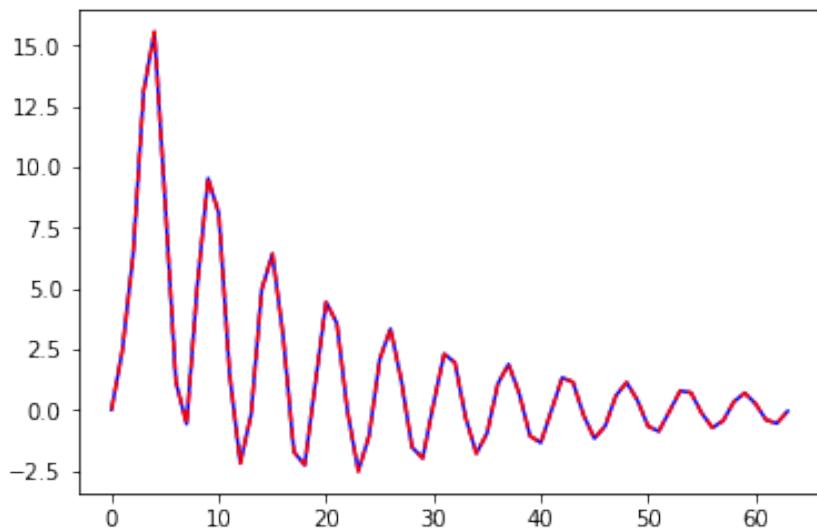
```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  4.40000000e+00  8.40000000e+00
      7.40000000e+00 -4.35900205e-11  8.79341044e-12 -2.05002681e-12]
```



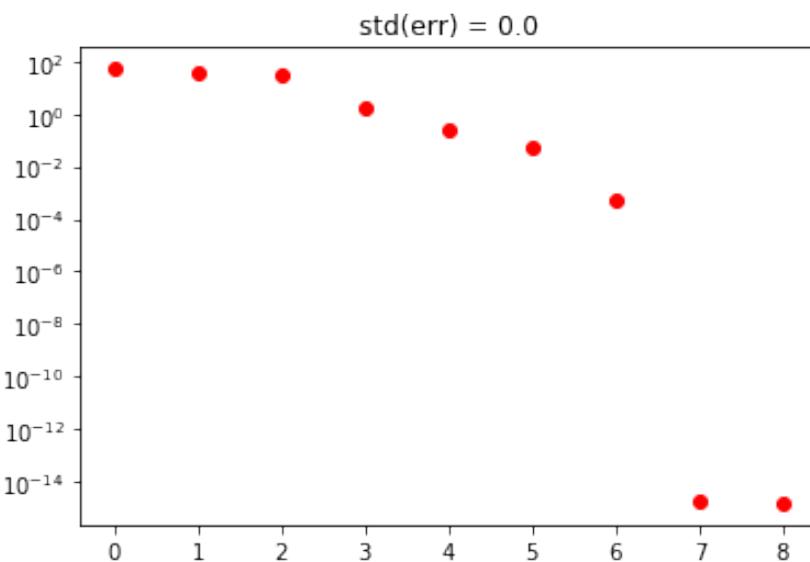
```
### na_est = 8 :
```



```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.15746488 -0.39400275  0.71438796 -0.30788525 -0.4670424
      0.00345895  0.01034865  0.00181693]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  5.91808429e+00  1.11831545e+01
      1.27132950e+01  4.68075989e+00 -9.73436887e-11  1.90533145e-11
      -3.62021524e-12]
```



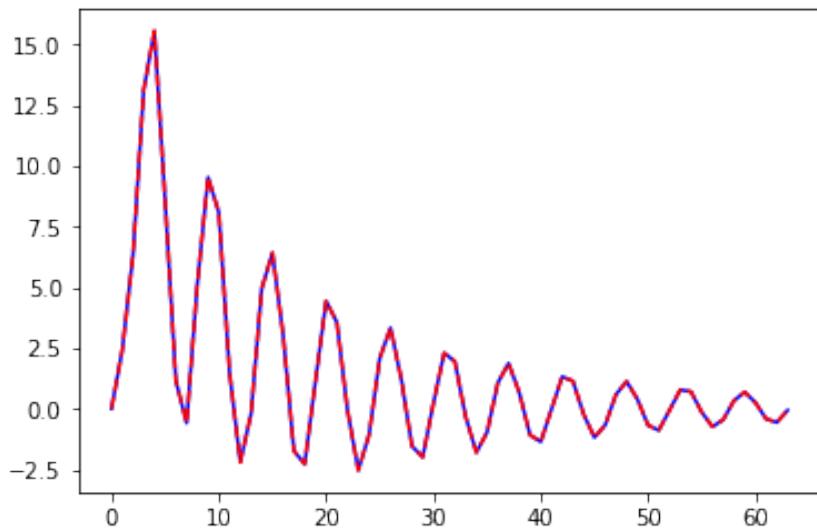
```
### na_est = 9 :
```



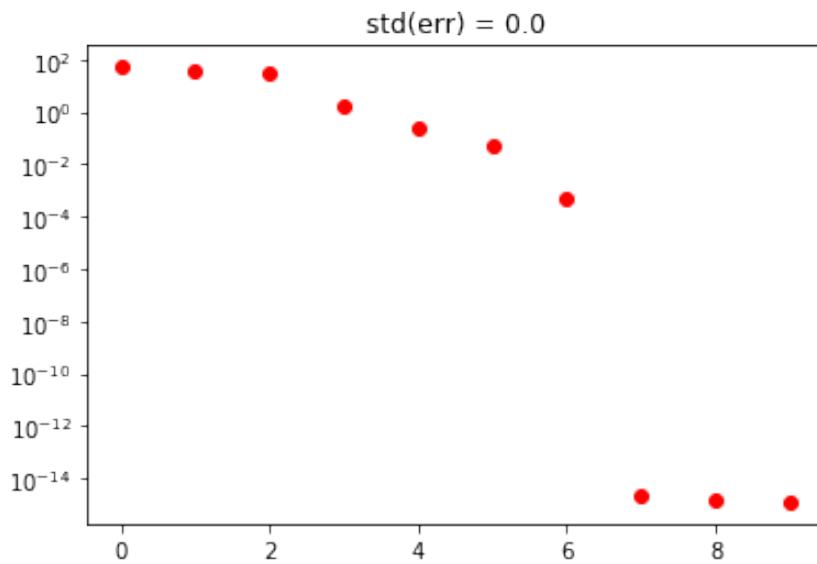
```

a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -7.98116068e-01 -5.00698334e-02  7.74794538e-01
      -7.39868057e-01 -1.12411586e-01  1.28287072e-01  4.92015209e-03
      -1.94020237e-03 -4.65860650e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  4.38052144e+00  7.97505364e+00
      6.61822631e+00 -1.42238373e+00 -1.20014330e+00 -3.30324657e-12
      7.78932474e-13  7.46069873e-14]

```



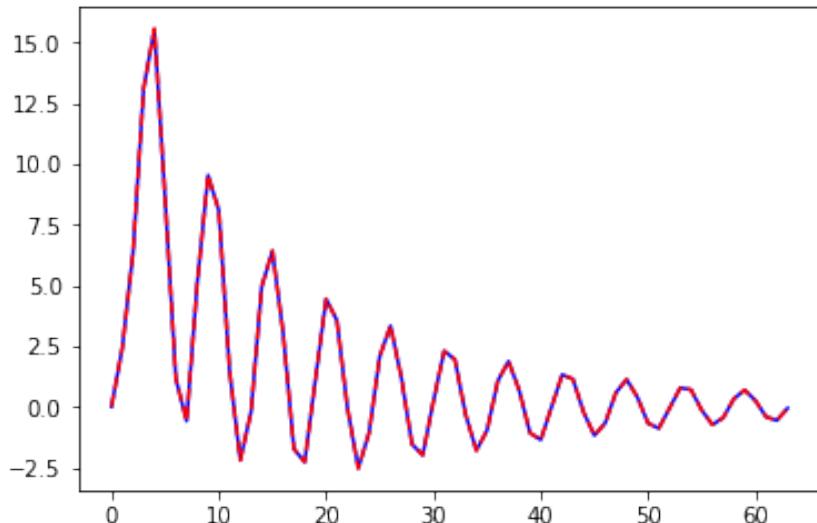
```
### na_est = 10 :
```



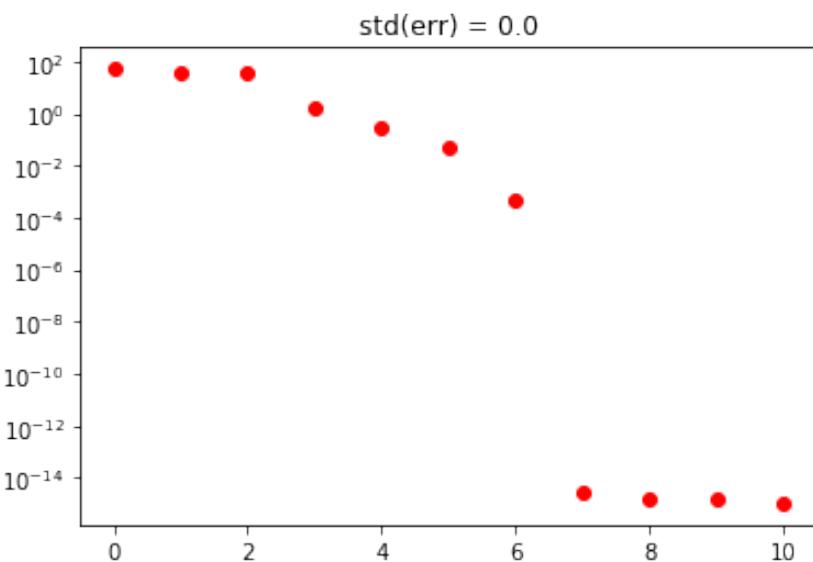
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.04344669e-01  1.11877889e-01  1.19458430e-01
      -1.79510301e-01 -1.22801250e-01 -4.25624678e-01  2.35256025e-01
      9.05452513e-03 -2.92792279e-03 -9.28570954e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  5.32557279e+00  1.08429129e+01
      1.12313863e+01  4.04258071e+00 -4.15179598e-01 -2.39217073e+00
      5.59552404e-12 -1.18838273e-12  1.70530257e-13]

```



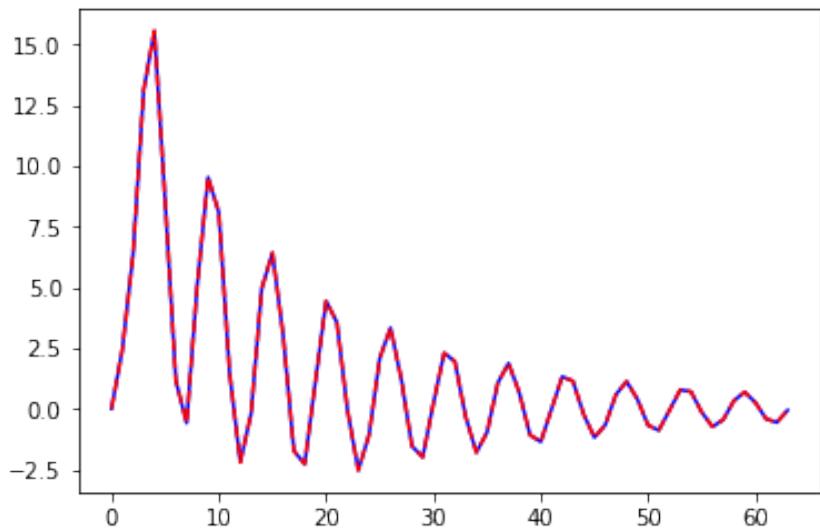
```
### na_est = 11 :
```



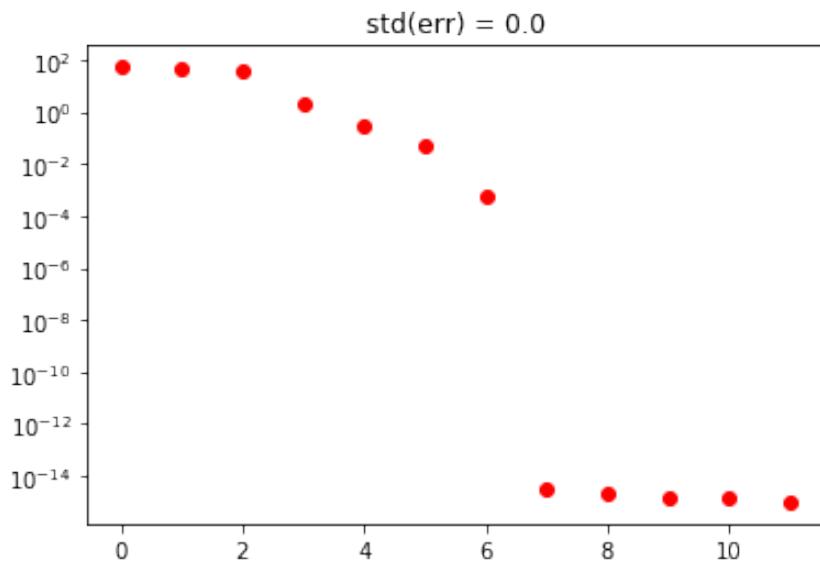
```

a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -5.64718725e-01  8.60139991e-02  2.07065983e-01
      -1.64164984e-01 -2.05150144e-01 -3.11414413e-01  3.26286168e-01
      -8.10369627e-02 -5.17668123e-03  4.75900902e-04  3.52471653e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  4.94067506e+00  9.77112451e+00
      9.17467207e+00  1.79857427e+00 -1.13829583e+00 -1.47955637e+00
      9.08032248e-01  8.49453841e-12 -1.45838897e-12  1.32116540e-13]

```



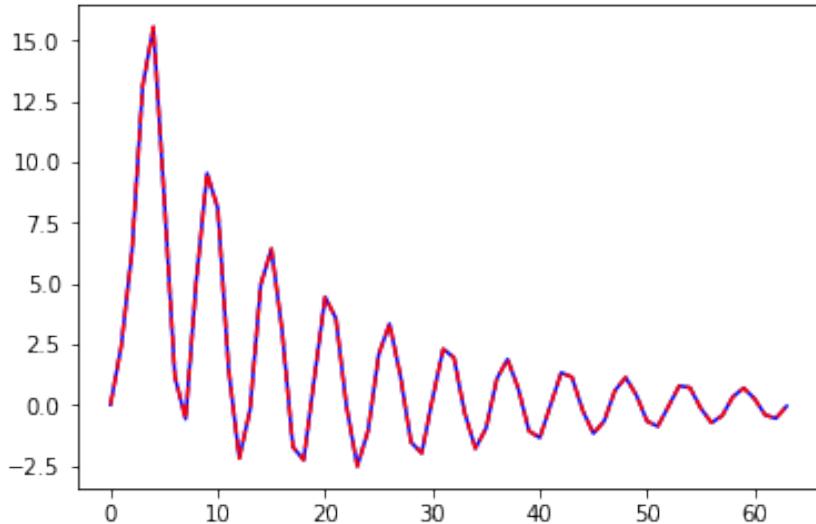
```
### na_est = 12 :
```



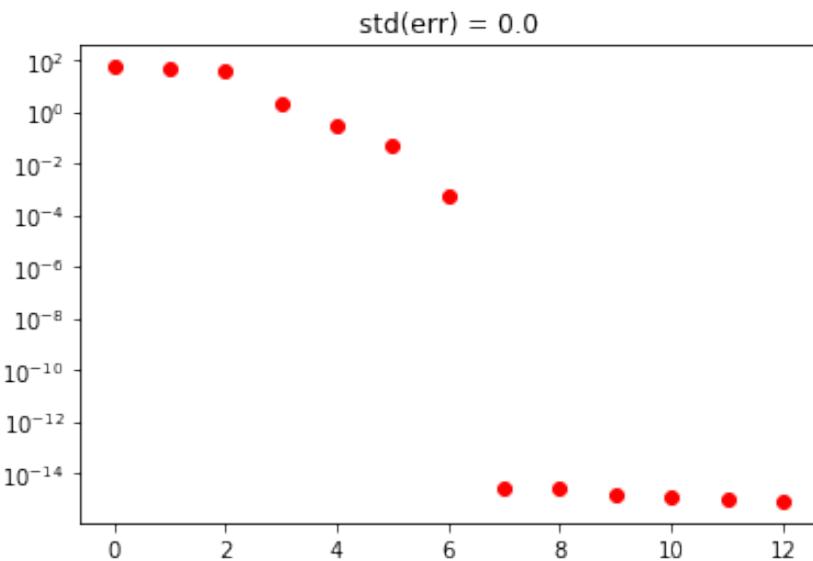
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -8.66720644e-01  3.71581669e-01  4.90406548e-01
      -6.85530419e-01  1.75795338e-01 -1.62106175e-01  4.11799845e-02
      -1.26681585e-02 -3.97914241e-02  1.39165866e-04  8.54444704e-04
      1.58571365e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  4.21587045e+00  8.55508283e+00
      7.69030998e+00  1.38317150e+00  2.07135244e+00  9.30838226e-01
      9.84017221e-01  4.08509200e-01 -7.81241738e-12  1.51501034e-12
      -4.09006162e-13]

```



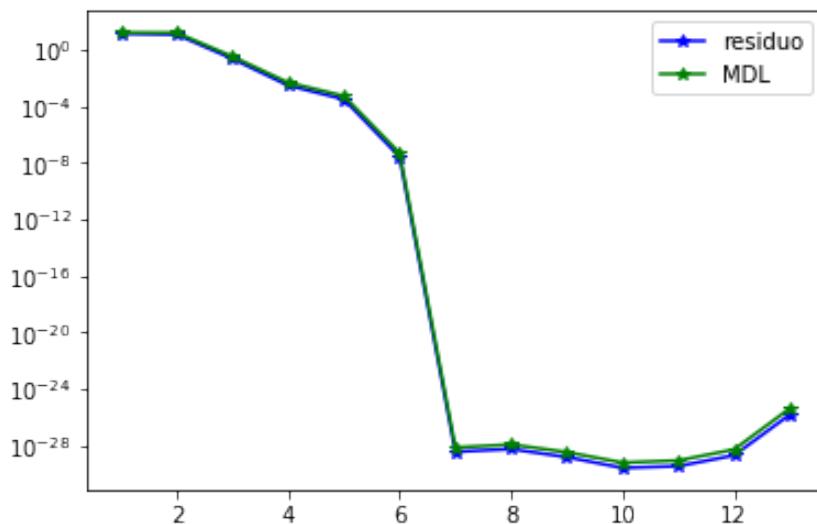
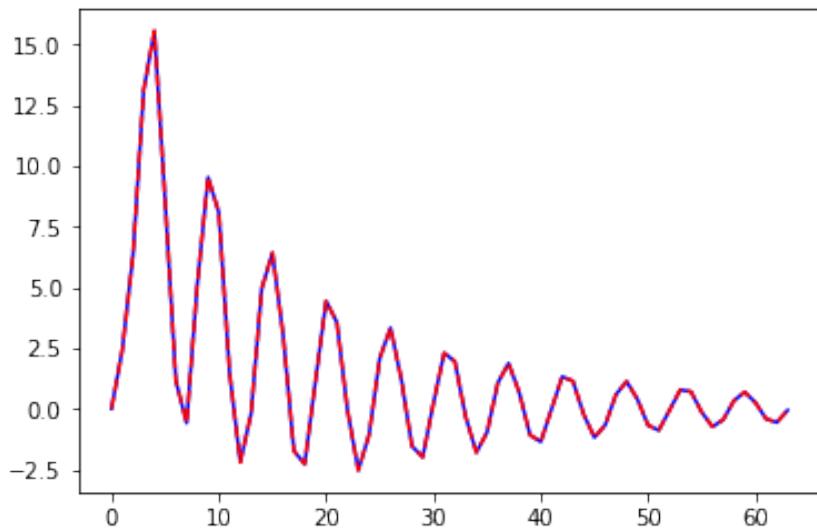
```
### na_est = 13 :
```



```

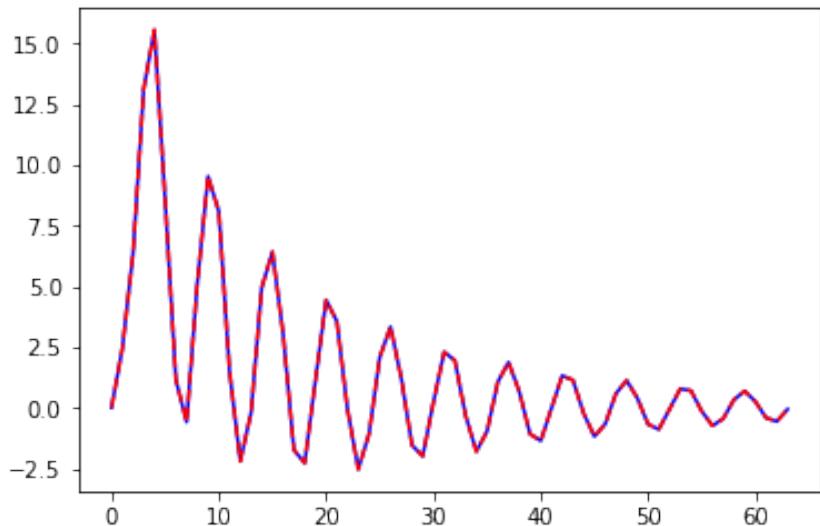
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -1.41487000e+00  5.05528391e-01  7.26220129e-02
      -5.30676477e-01  2.01003163e-01 -1.76269431e-01  4.28249577e-01
      -8.61233030e-02  5.51603616e-02 -7.07063290e-02 -9.04215432e-04
      1.16324973e-03  2.83064782e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.00000000e+00  2.40000000e+00  2.90031200e+00  5.42540662e+00
      3.39148403e-01 -7.63933238e+00 -4.97305379e+00 -2.89722084e+00
      1.47103532e+00  8.23947219e-01  7.29227295e-01  1.38091760e-11
      -2.86437540e-12  4.59576821e-13]

```

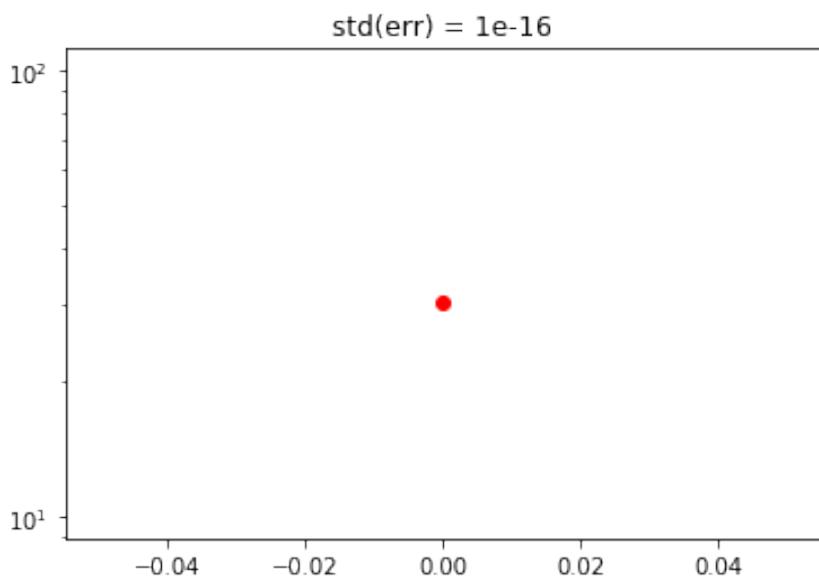


```
a = [ 1.           -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.           -0.79          0.1057         0.647529      -0.71747008 -0.01321738]
```

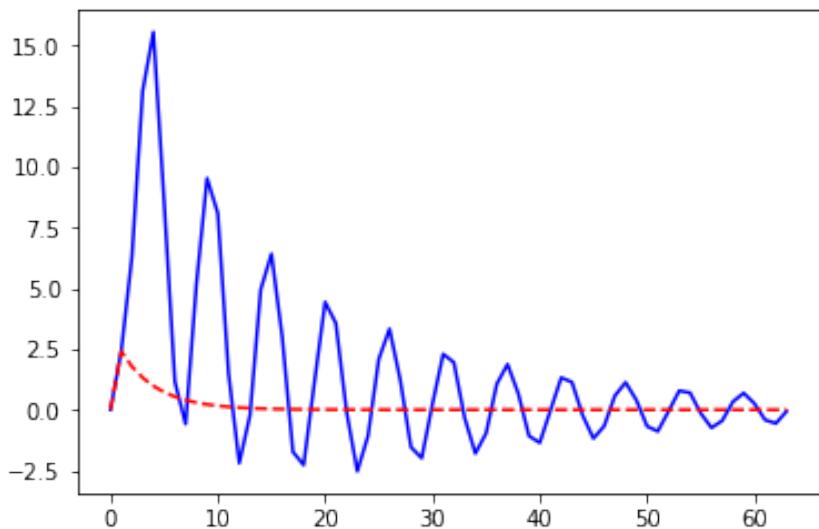
```
0.01181941  0.00287246]  
b = [0.  2.4 4.4 8.4 7.4]  
b_est = [0.  2.4 4.4 8.4 7.4]
```



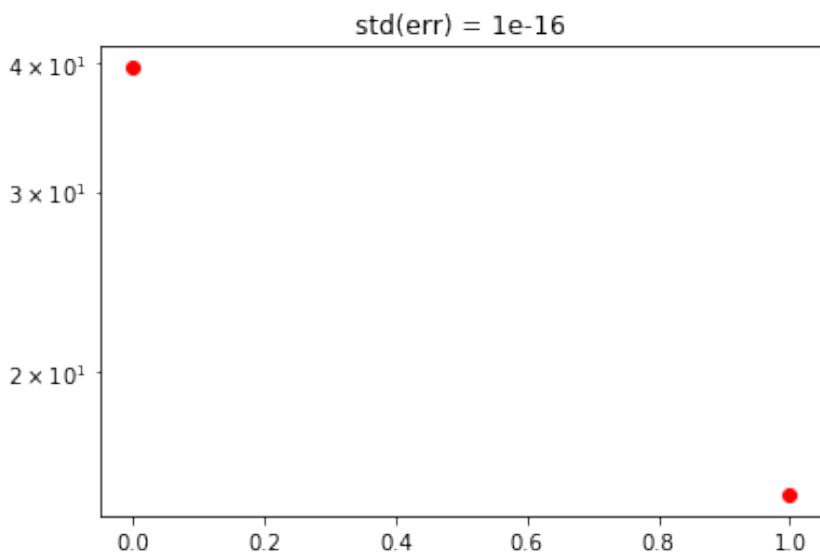
```
std = 1e-16 :  
### na_est = 1 :
```



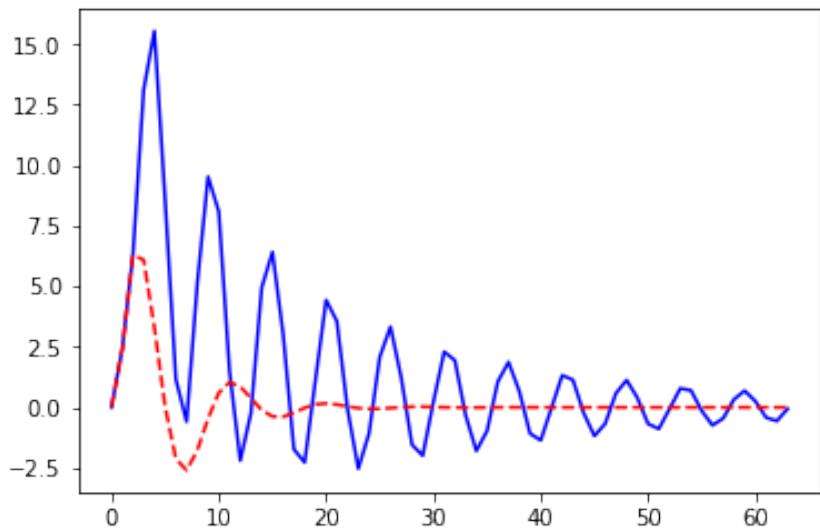
```
a = [ 1.           -0.79           0.1057           0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74094934]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [5.47254777e-17 2.40000000e+00]
```



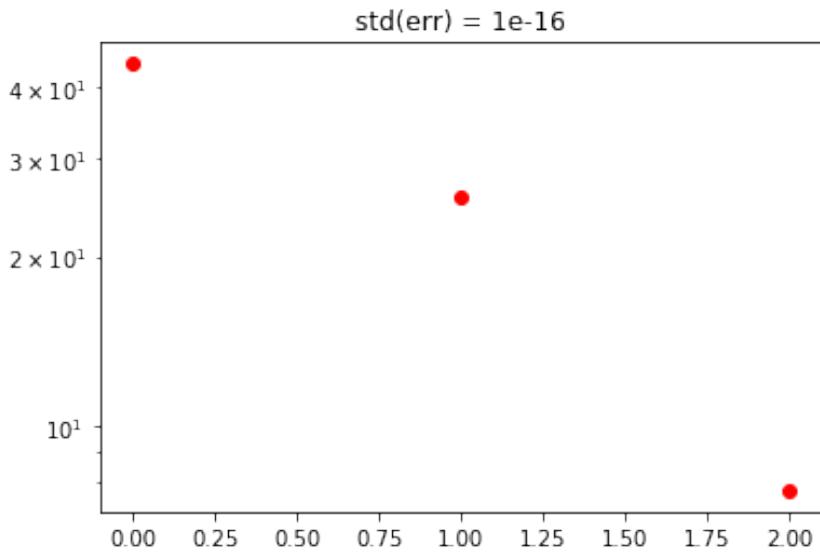
```
### na_est = 2 :
```



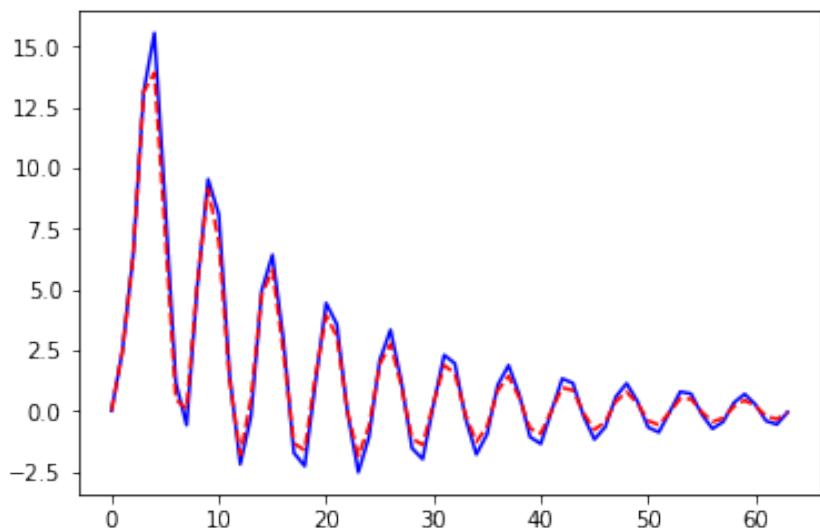
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21625056  0.65362331]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [5.47254777e-17 2.40000000e+00 3.37699865e+00]
```



```
### na_est = 3 :
```

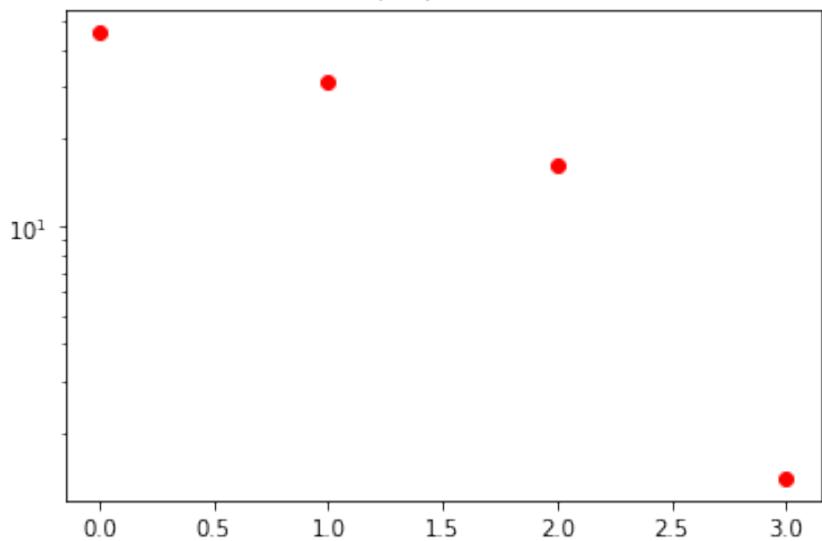


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.68740899  1.61319279 -0.80094766]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [5.47254777e-17 2.40000000e+00 2.24621842e+00 6.36789569e+00]
```

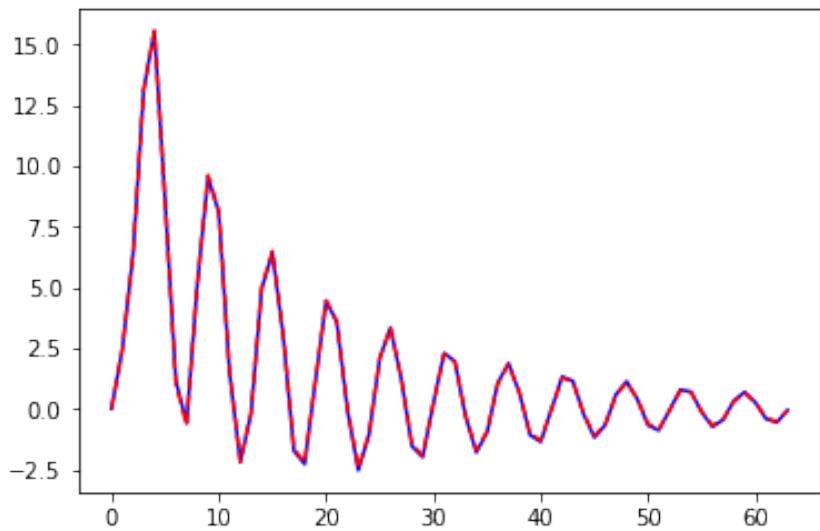


```
### na_est = 4 :
```

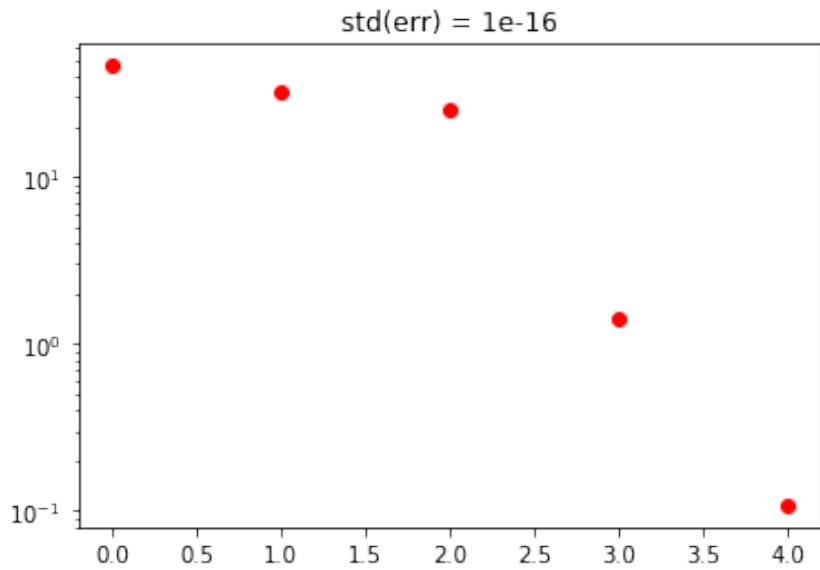
std(err) = 1e-16



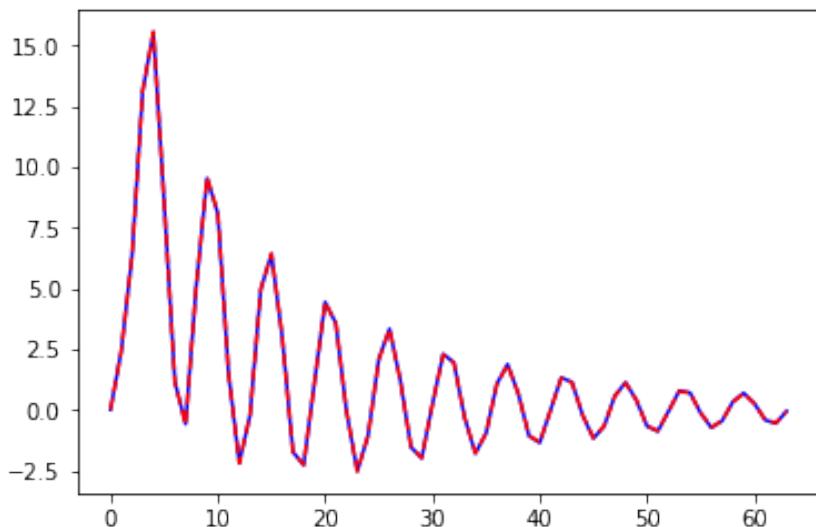
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.79406241  0.12323547  0.6340996   -0.71855887]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [5.47254777e-17 2.40000000e+00 4.39025022e+00 8.41650820e+00
    7.42487331e+00]
```



```
### na_est = 5 :
```

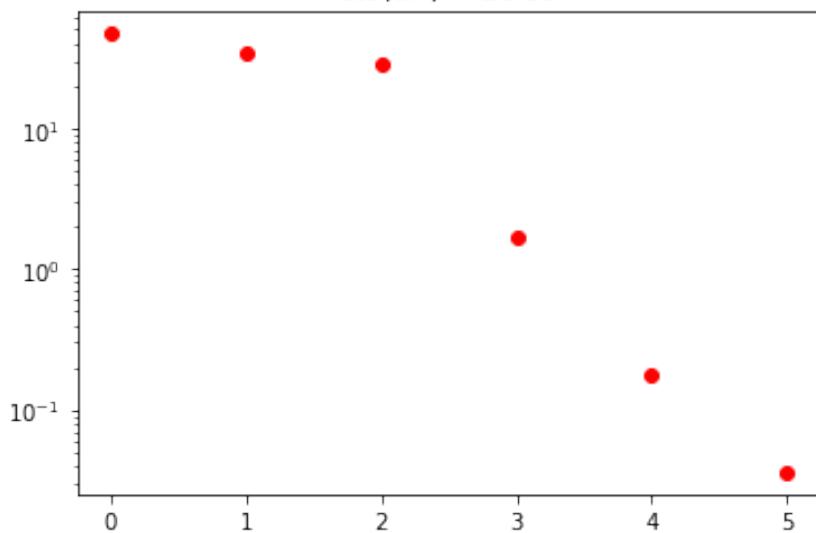


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.37353837  0.57138116  0.58688411 -1.11125735  0.43032752]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  2.99950790e+00  5.84367718e+00
      2.53026402e+00 -4.28841806e+00]
```

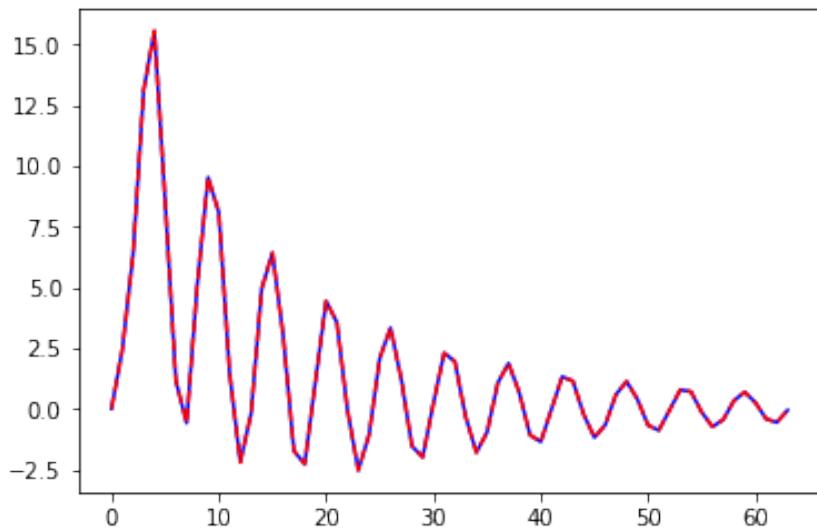


```
### na_est = 6 :
```

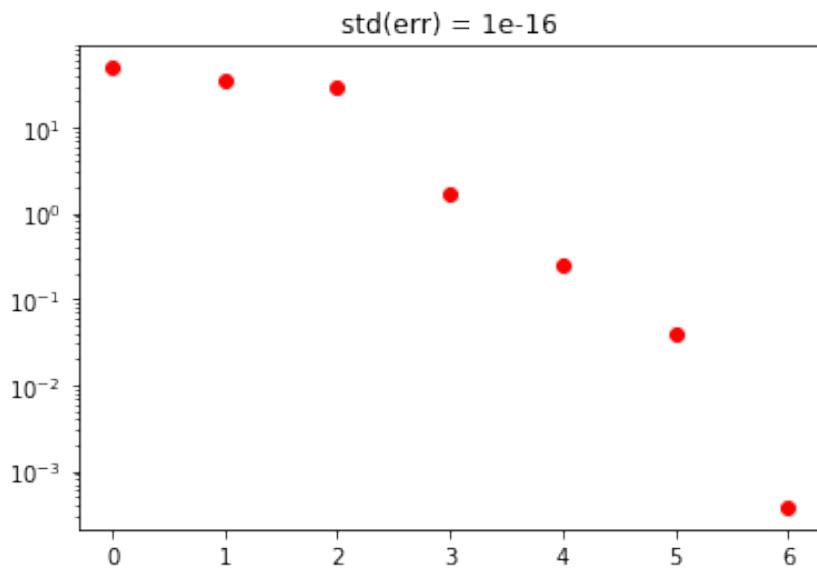
`std(err) = 1e-16`



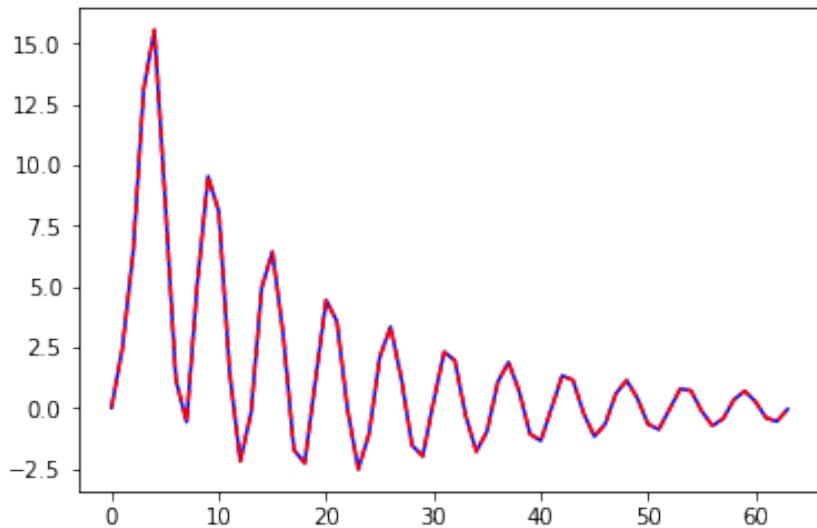
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.98269039  0.28884512  0.60274659 -0.83901822  0.14508689
   -0.00773825]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  3.93754306e+00  7.62636959e+00
   5.91747515e+00 -1.16621562e+00  2.28761736e-01]
```



```
### na_est = 7 :
```

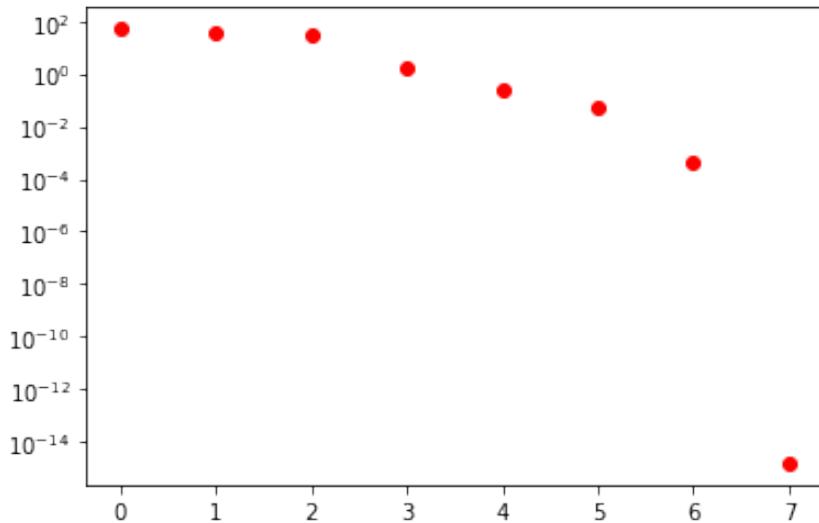


```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  4.40000000e+00  8.40000000e+00
      7.40000000e+00 -3.46567219e-11  6.70086209e-12 -1.31950006e-12]
```

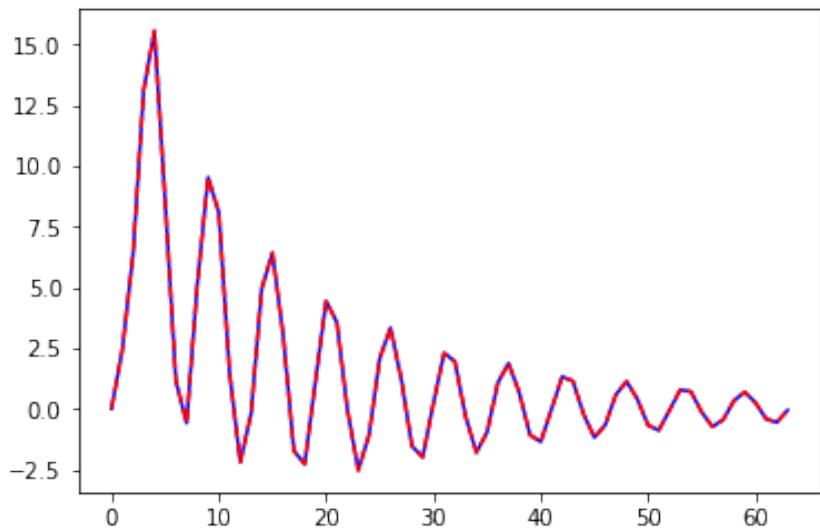


```
### na_est = 8 :
```

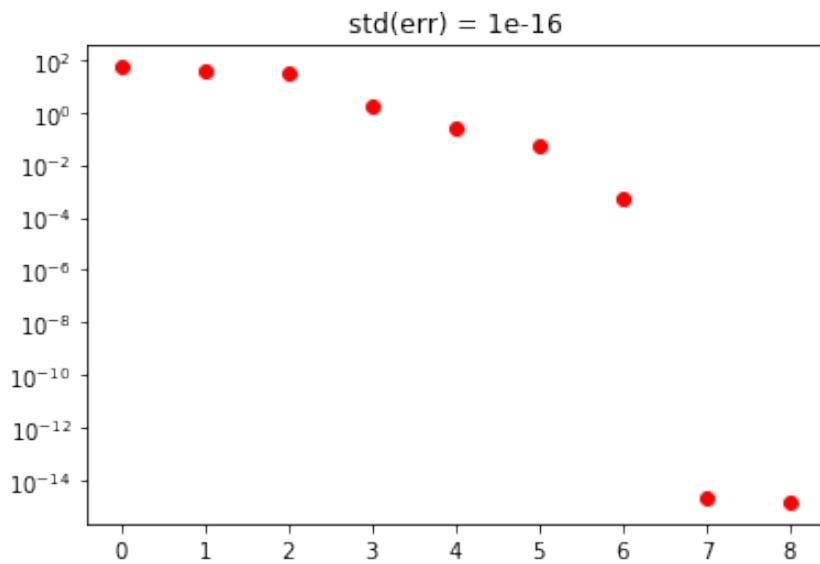
std(err) = 1e-16



```
a = [ 1.           -0.79          0.1057        0.647529     -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -7.86532435e-01  1.02960624e-01  6.47895522e-01
      -7.15224731e-01 -1.57052540e-02   1.17735735e-02  2.91344887e-03
      9.96045632e-06]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  4.40832216e+00  8.41525729e+00
      7.42912754e+00  2.56599796e-02 -7.01185776e-11  1.38311584e-11
      -2.56861199e-12]
```



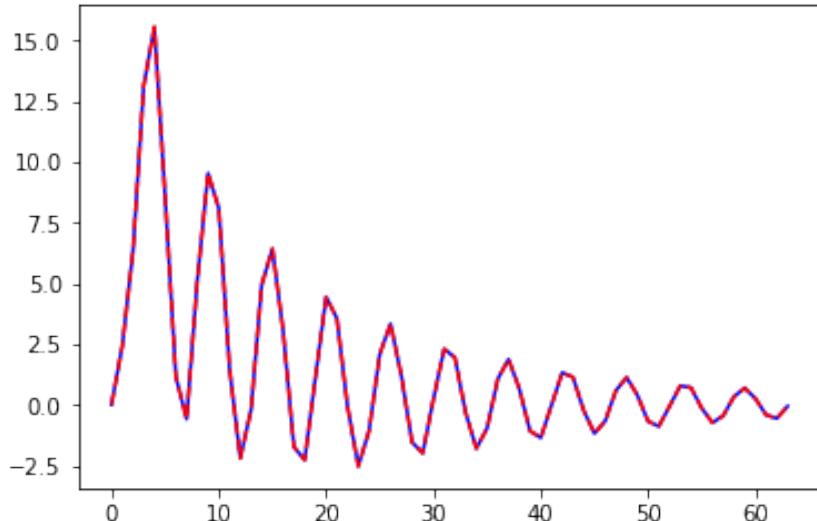
```
### na_est = 9 :
```



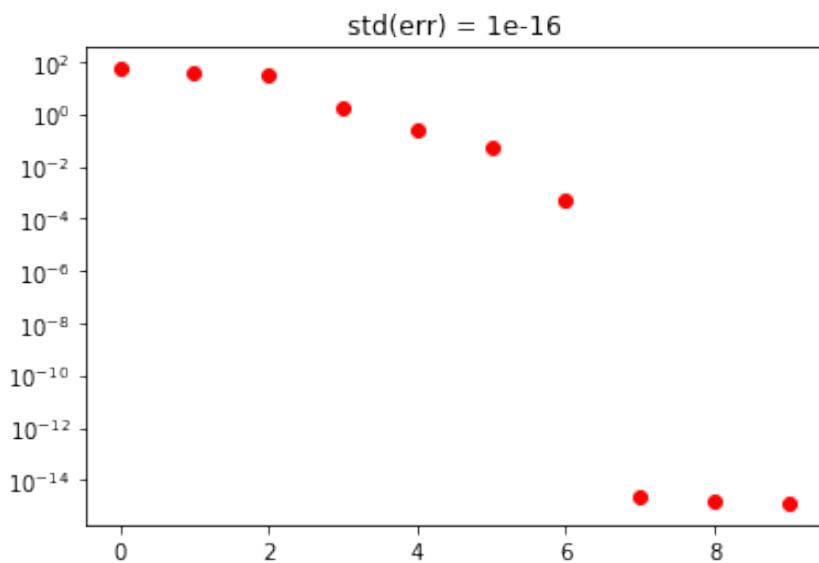
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -8.47744575e-01  3.09973683e-01  5.16087578e-01
      -7.38091484e-01  1.30946642e-01 -1.01247914e-01  9.29481541e-05
      1.70934410e-03  4.55732172e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  4.26141302e+00  8.52669699e+00
      7.61302963e+00  9.05396079e-01  1.17405047e+00 -3.02399217e-11
      6.30695496e-12 -1.53299595e-12]

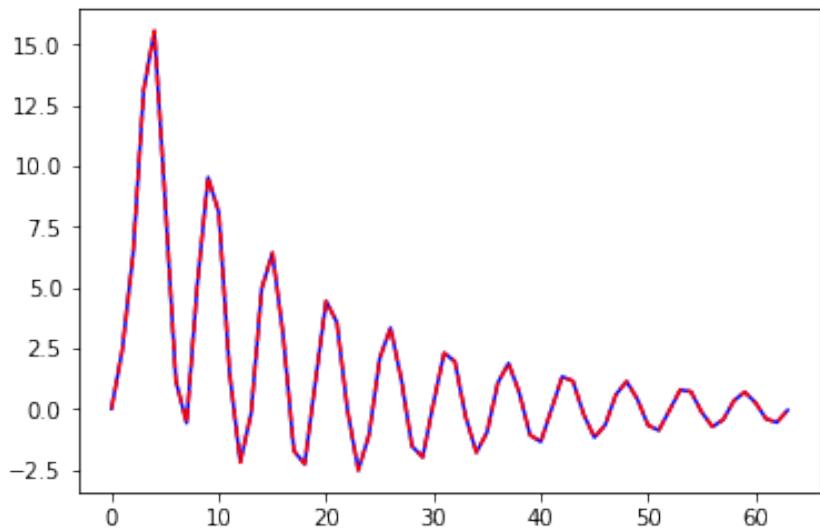
```



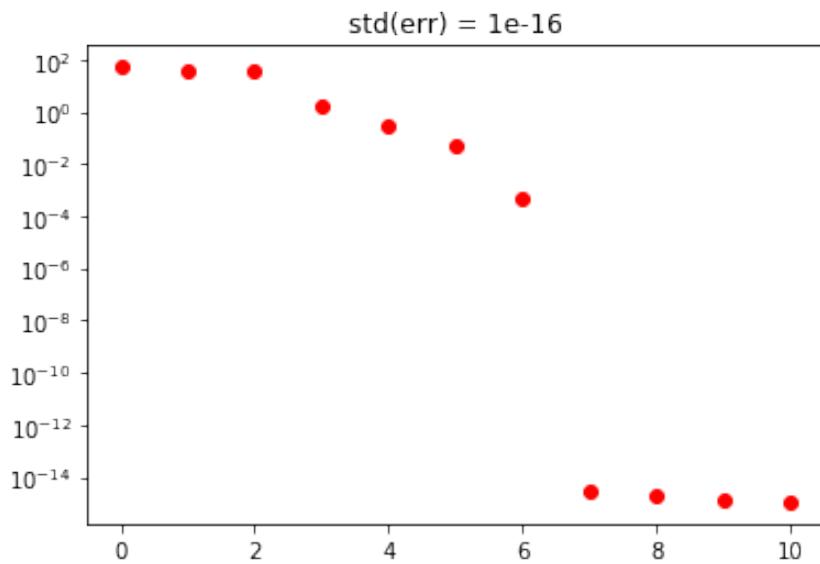
```
### na_est = 10 :
```



```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.58230923  0.61509897 -0.2443068  -0.21022047  0.23351478
      -0.72131378  0.27031537  0.01360231 -0.00257745 -0.00109655]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  4.89845784e+00  1.09301786e+01
      1.11917061e+01  5.51442661e+00  1.77706538e+00 -2.82490353e+00
      1.91615612e-11 -3.77120557e-12  6.64357458e-13]
```



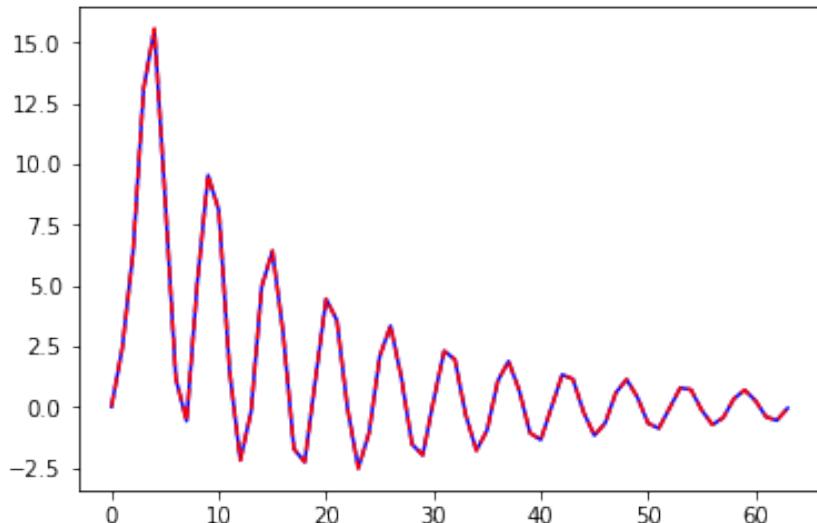
```
### na_est = 11 :
```



```

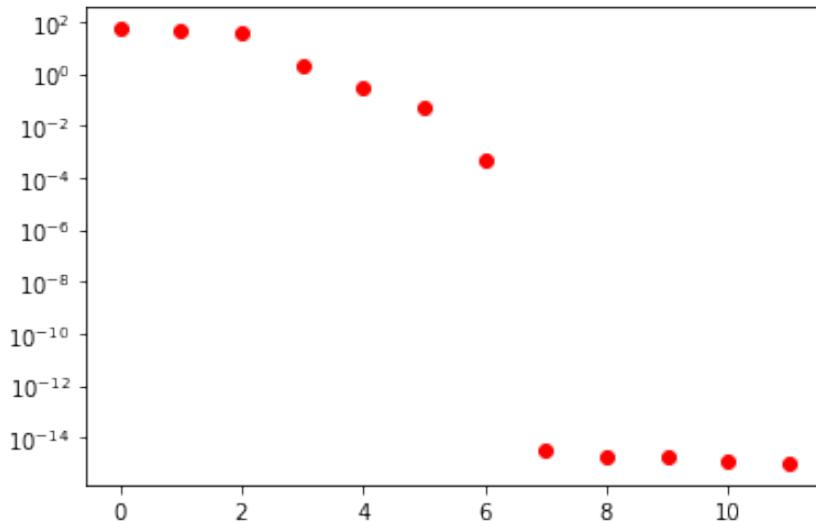
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -7.58056652e-01  9.60760047e-04  3.75570976e-01
      -4.82677647e-01 -8.81080902e-02 -1.55234626e-01  2.18014724e-01
      3.56355766e-02 -3.63524561e-03 -1.49869489e-03 -1.28171906e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  4.47666404e+00  8.34974114e+00
      6.50696433e+00 -2.02637031e+00 -3.62505924e+00 -2.87707063e+00
      -3.30194563e-01 -2.03552730e-11  3.91331412e-12 -9.91429161e-13]

```

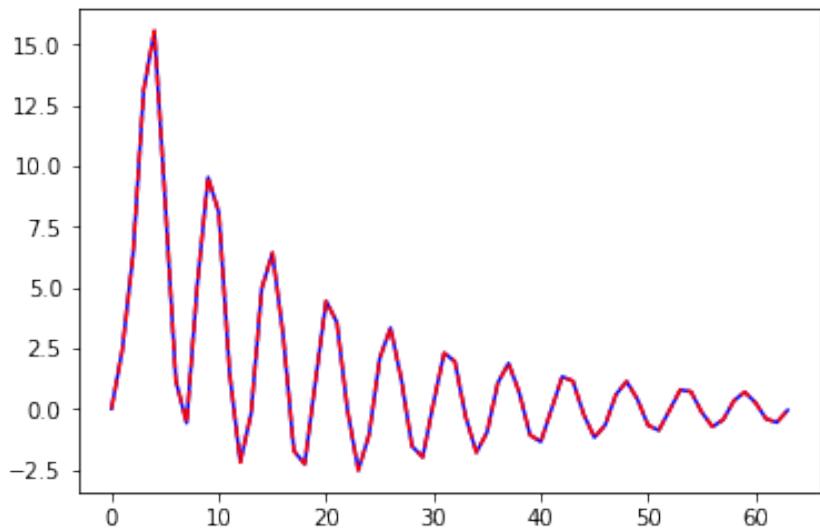


```
### na_est = 12 :
```

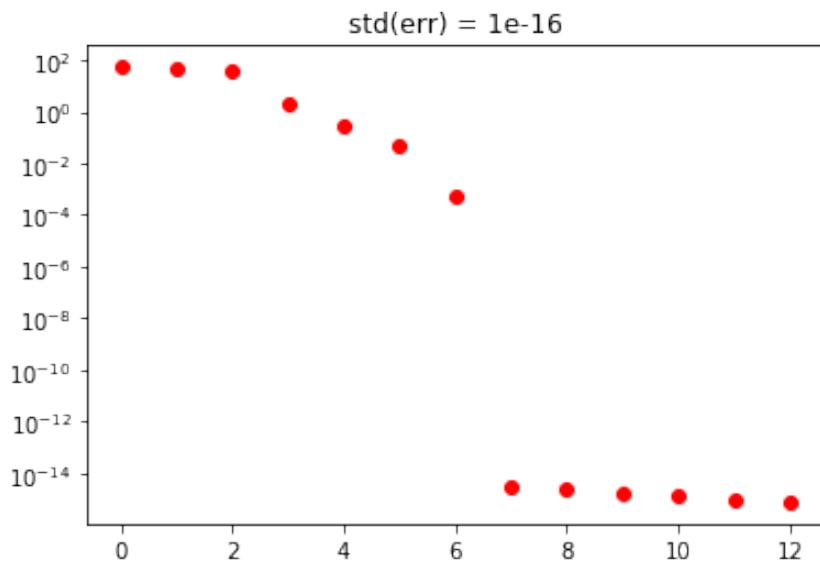
std(err) = 1e-16



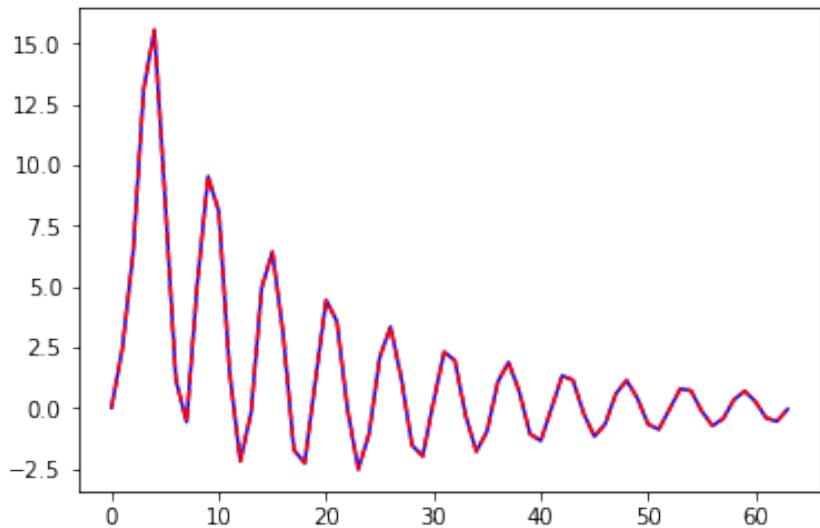
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -3.43063836e-01 -3.36369576e-02  4.53897720e-02
      1.39018084e-01 -4.46799485e-01 -3.85647243e-01  4.49425907e-01
     -1.53640853e-01  4.31779866e-02  1.49736788e-03 -3.57045651e-04
     -2.01872698e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  5.47264679e+00  1.08795014e+01
      1.09414744e+01  3.38416770e+00 -1.89791056e+00 -2.48012850e+00
      6.29754997e-01 -5.20061452e-01 -3.26476624e-11  6.21858121e-12
     -1.05826459e-12]
```

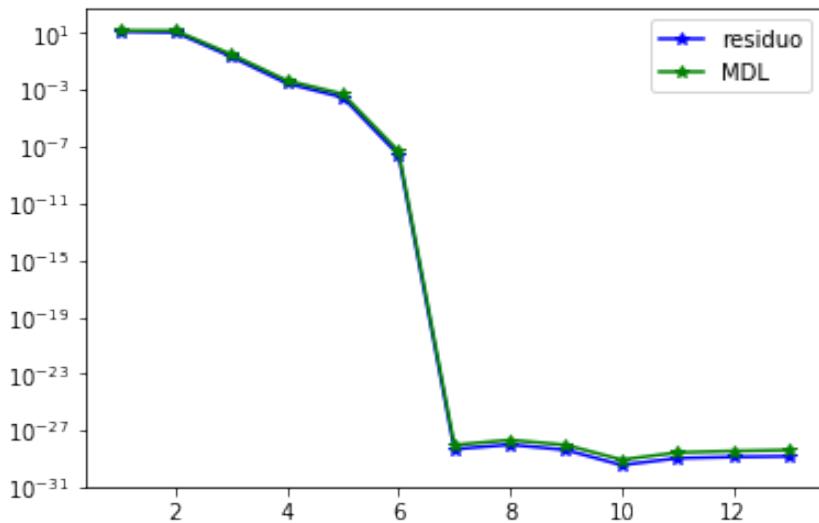


```
### na_est = 13 :
```

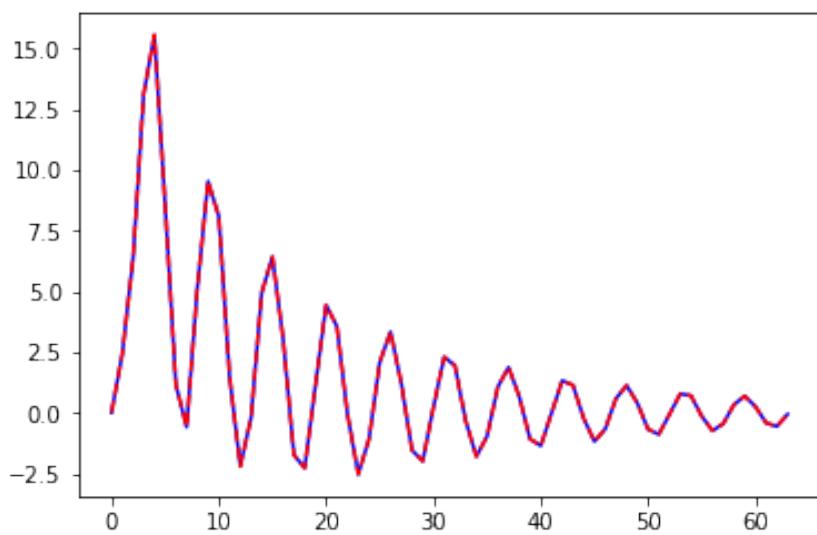


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.22332919 -0.24491736  0.40437658 -0.30285759 -0.20234162
      0.73328958 -0.71399982  0.25755518  0.6074253  -0.72993326 -0.01300455
      0.01216383  0.00290953]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 5.47254777e-17  2.40000000e+00  5.76000995e+00  1.11262778e+01
      1.20437590e+01  3.67313365e+00 -1.64529927e+00 -1.38208039e-01
      3.97144518e+00  9.00276599e+00  7.49549131e+00  2.41731080e-11
     -4.87032636e-12  8.95644670e-13]
```

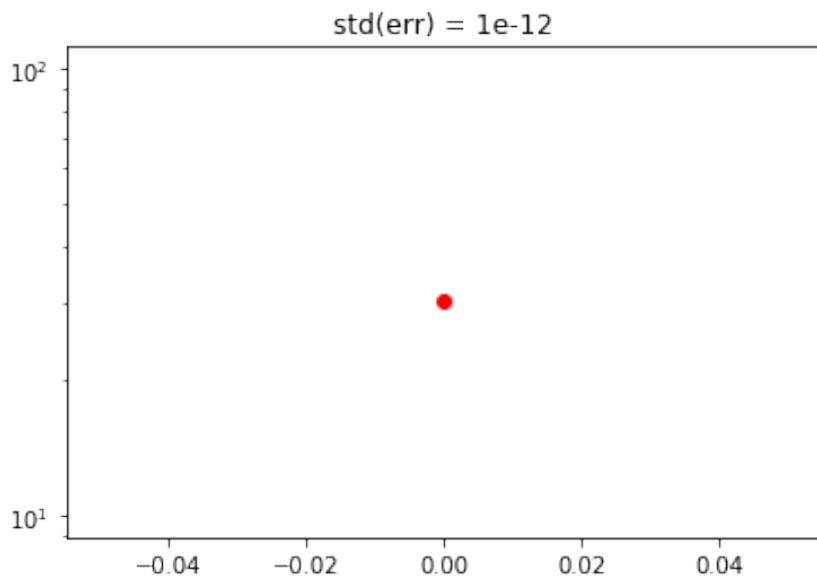




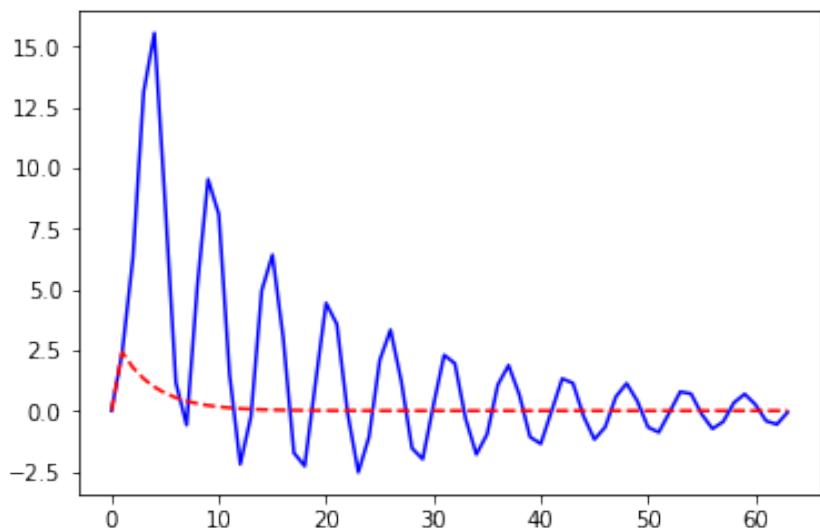
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [5.47254777e-17 2.40000000e+00 4.40000000e+00 8.40000000e+00
      7.40000000e+00]
```



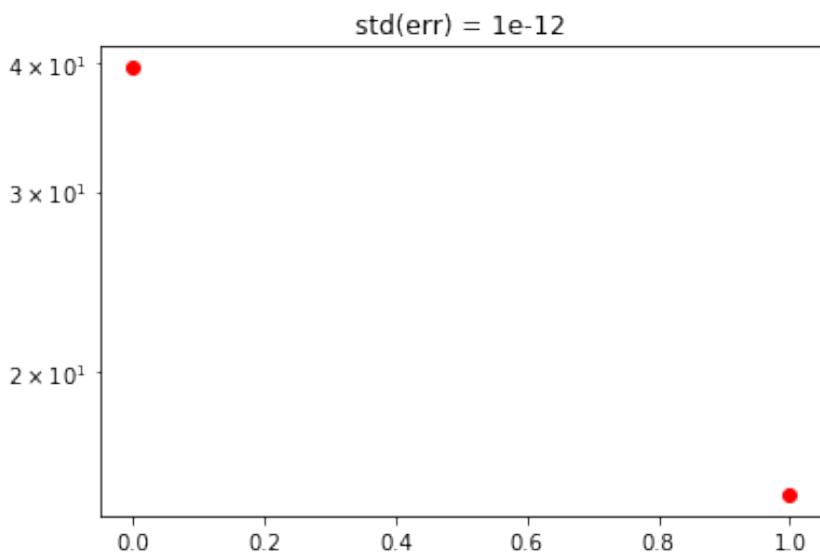
```
std = 1e-12 :  
### na_est = 1 :
```



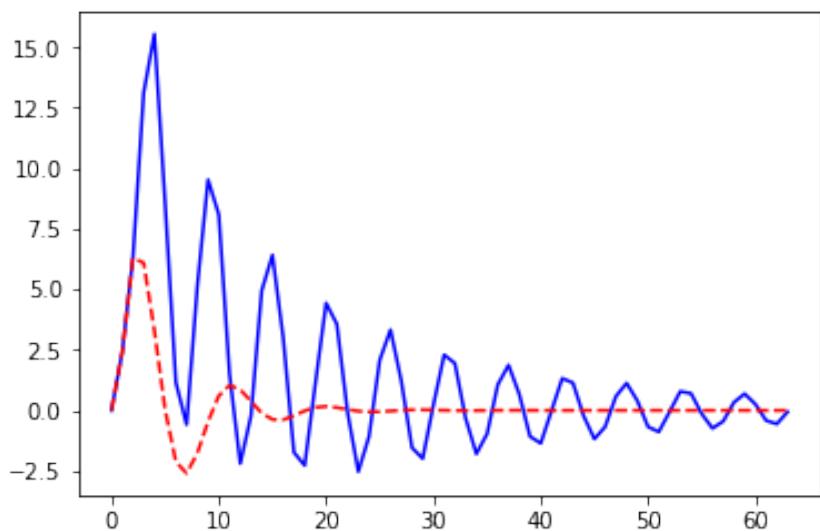
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74094934]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.31810915e-13 2.40000000e+00]
```



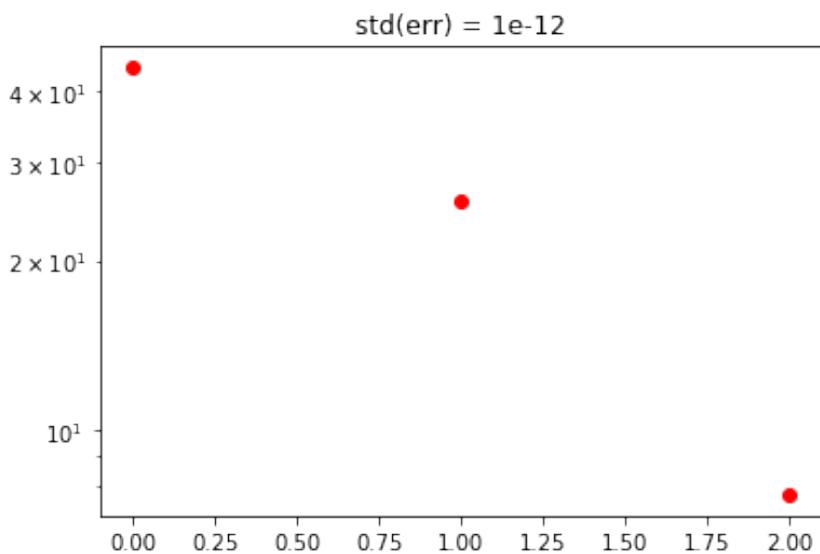
```
### na_est = 2 :
```



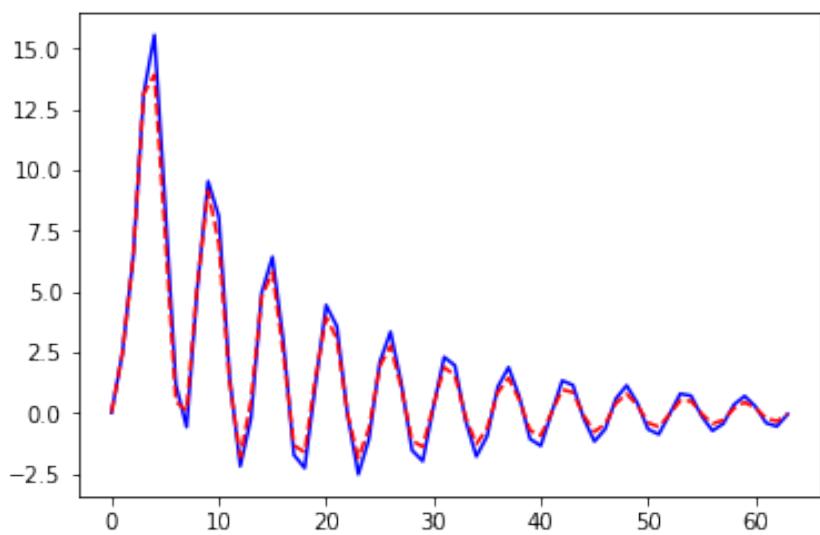
```
a = [ 1.           -0.79           0.1057         0.647529      -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21625056  0.65362331]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.31810915e-13 2.40000000e+00 3.37699865e+00]
```



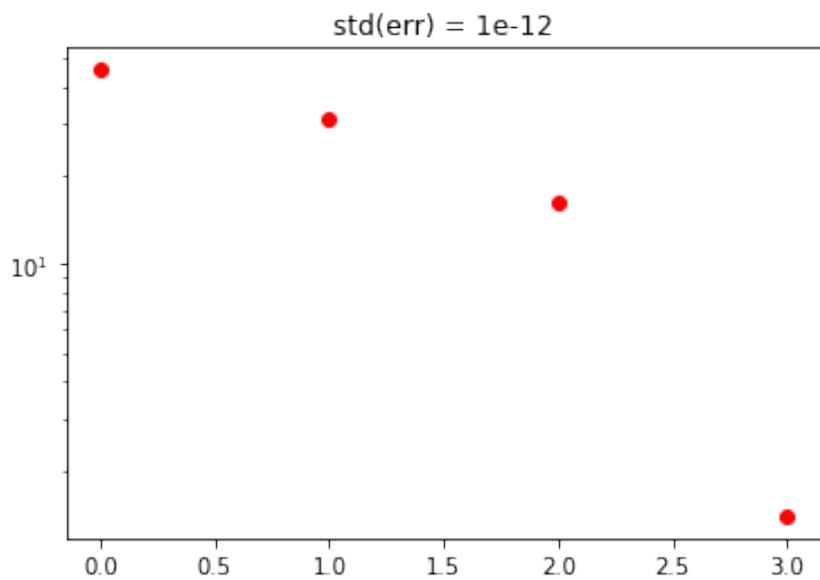
```
### na_est = 3 :
```



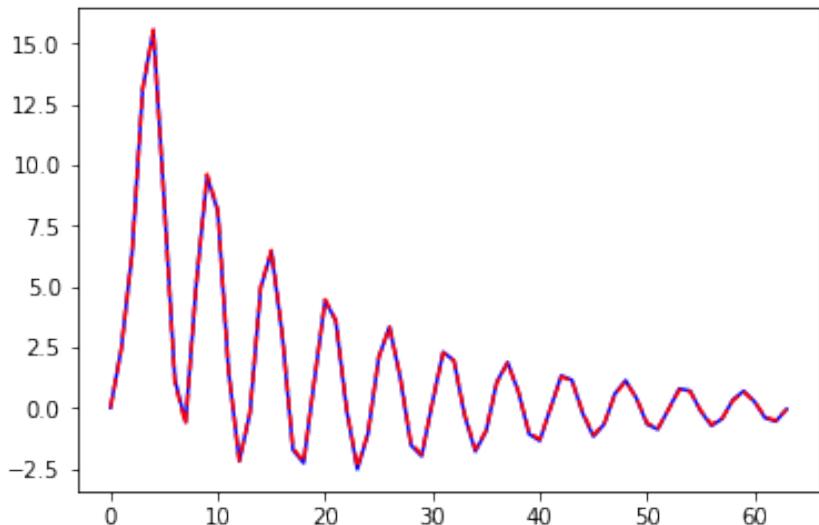
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.68740899  1.61319279 -0.80094766]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.31810915e-13 2.40000000e+00 2.24621842e+00 6.36789569e+00]
```



```
### na_est = 4 :
```

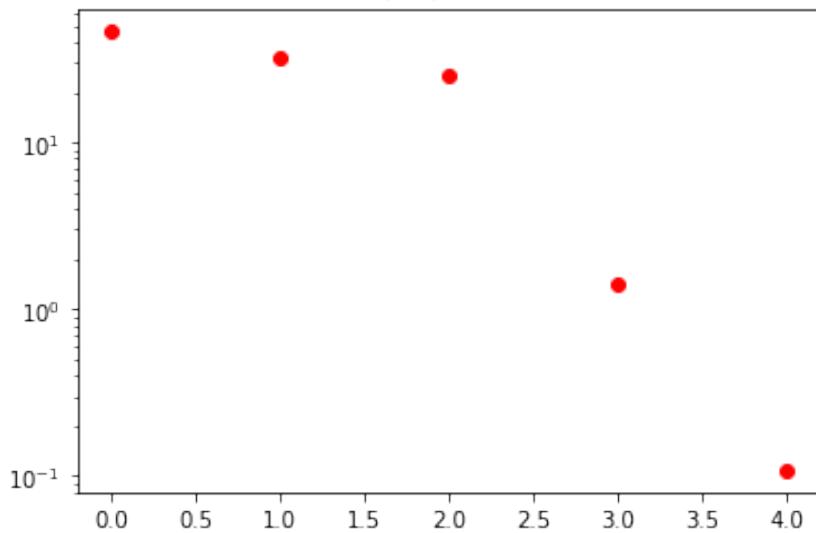


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79406241  0.12323547  0.6340996  -0.71855887]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.31810915e-13 2.40000000e+00 4.39025022e+00 8.41650820e+00
         7.42487331e+00]
```

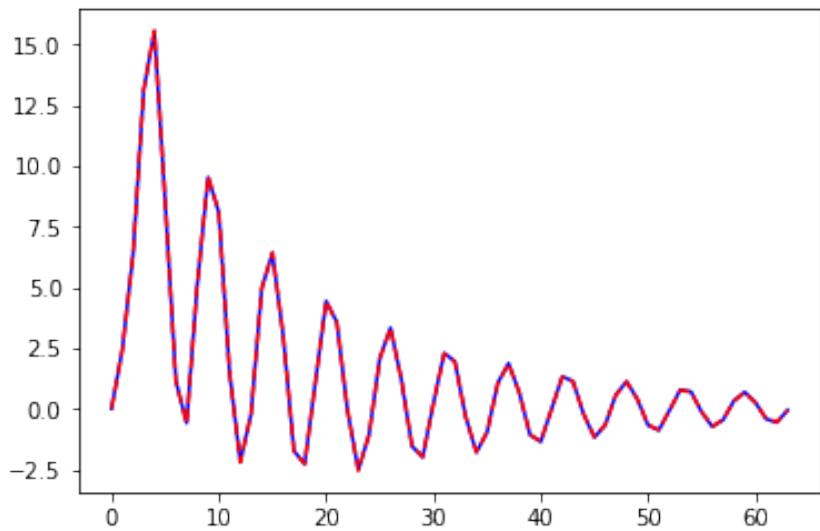


```
### na_est = 5 :
```

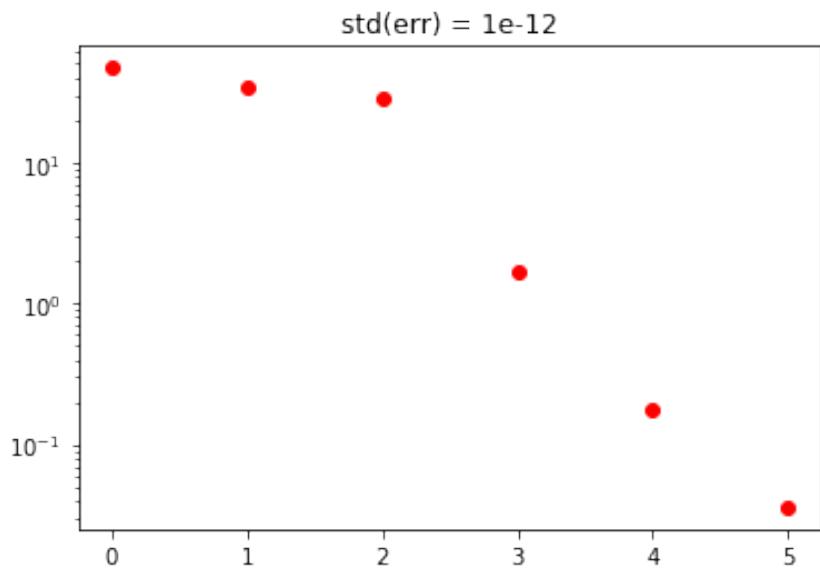
$\text{std}(\text{err}) = 1e-12$



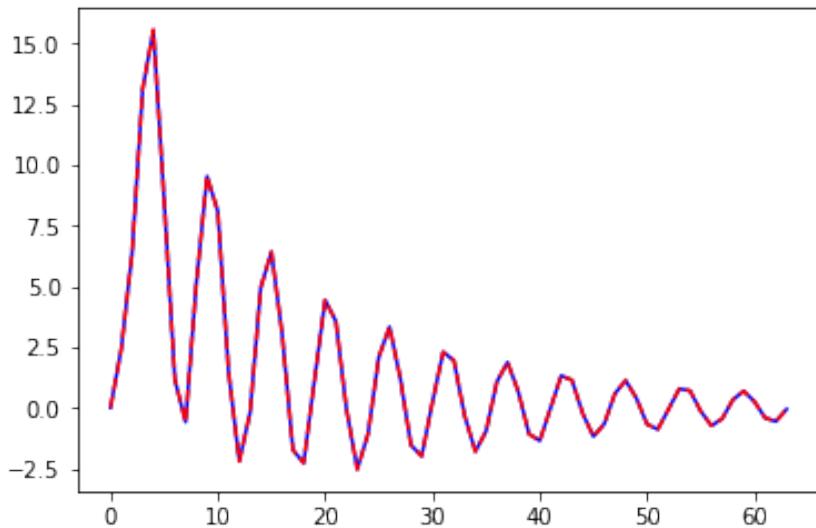
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -1.37353837  0.57138116  0.58688411 -1.11125735  0.43032752]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  2.99950790e+00  5.84367718e+00
     2.53026402e+00 -4.28841806e+00]
```



```
### na_est = 6 :
```

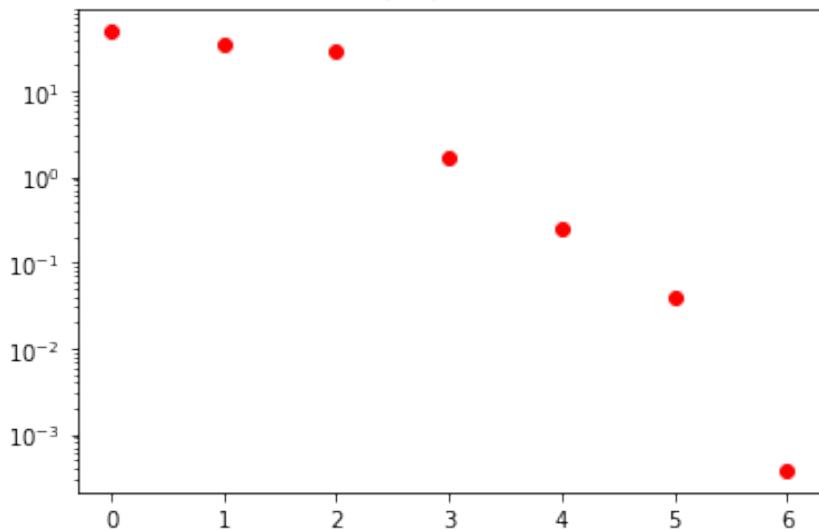


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.98269039  0.28884512  0.60274659 -0.83901822  0.14508689
      -0.00773825]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  3.93754306e+00  7.62636959e+00
      5.91747515e+00 -1.16621562e+00  2.28761736e-01]
```

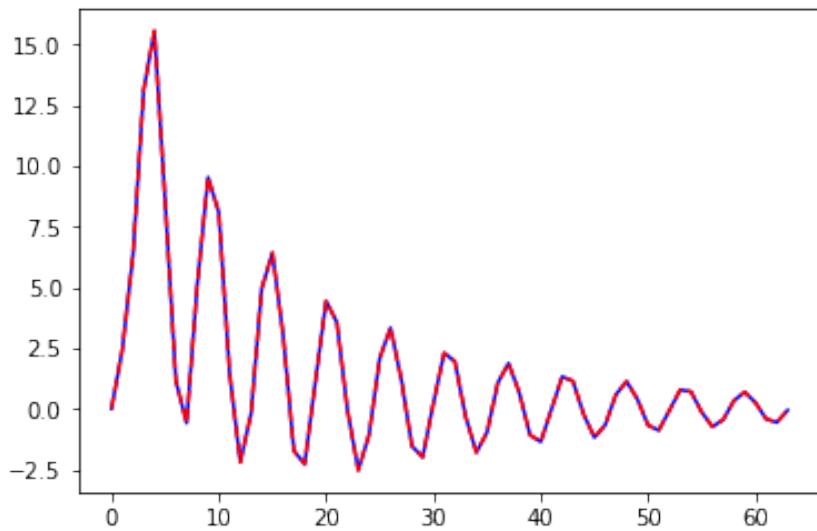


```
### na_est = 7 :
```

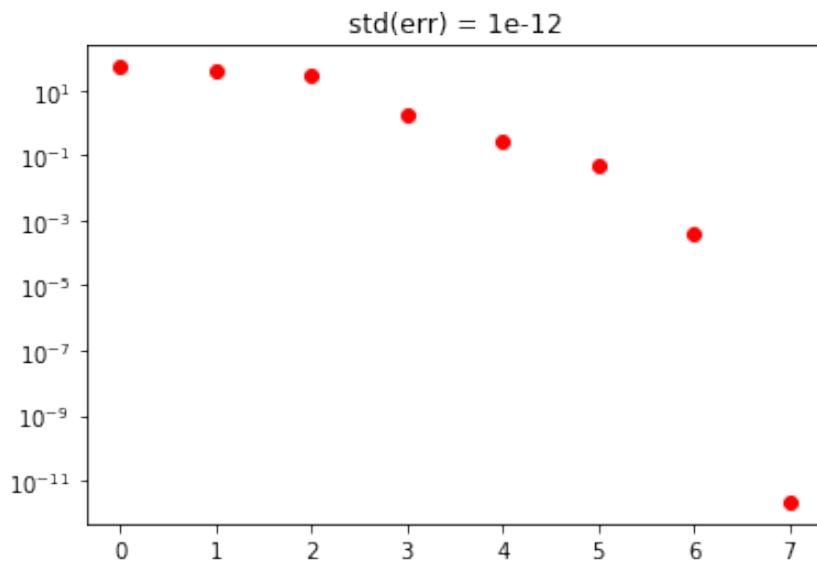
`std(err) = 1e-12`



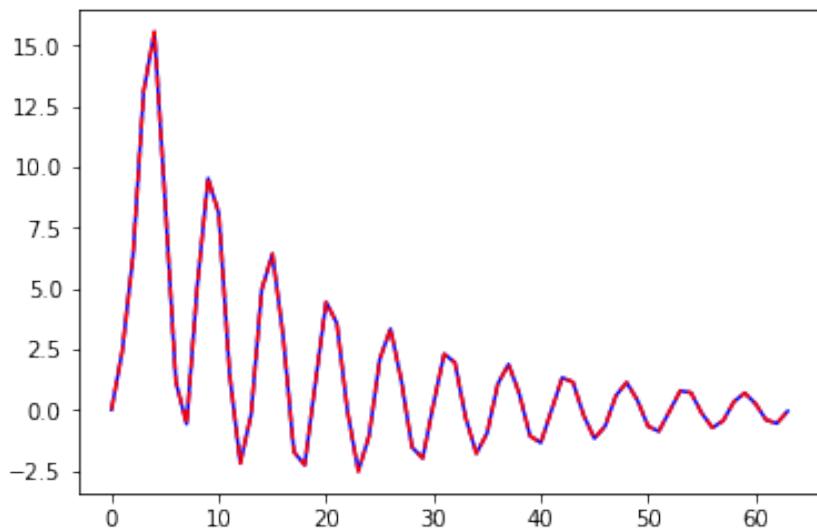
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  4.40000000e+00  8.40000000e+00
      7.40000000e+00  1.05155884e-09 -2.03783657e-10  6.82670587e-11]
```



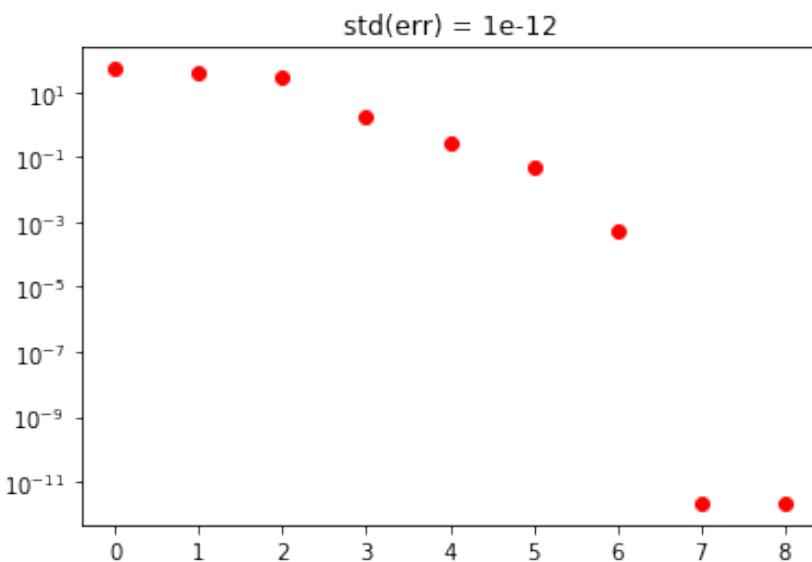
```
### na_est = 8 :
```



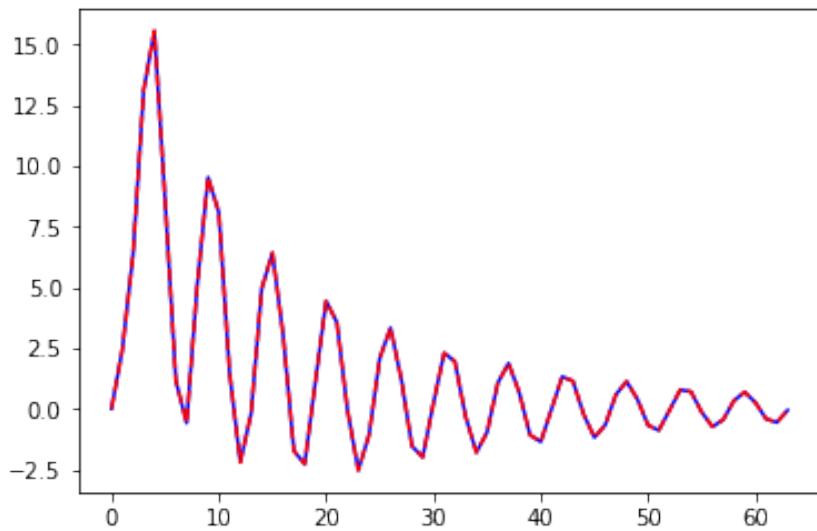
```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.32033824 -0.26533279  0.69717225 -0.41335047 -0.35018564
      0.00561171  0.00842359  0.00134909]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  5.52718823e+00  1.04665118e+01
      1.13451588e+01  3.47549705e+00 -7.45283835e-11  4.41804371e-11
      1.76987314e-11]
```



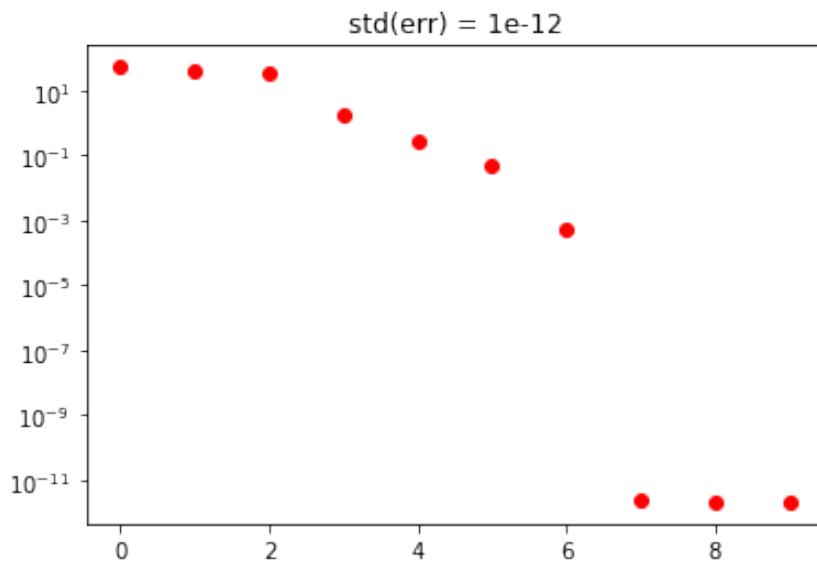
```
### na_est = 9 :
```



```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.07019212  0.09654328  0.2816144  -0.19223535 -0.16737102
      -0.39911298  0.00398515  0.00868048  0.00160712]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  6.12753890e+00  1.29099343e+01
      1.59081488e+01  1.00263069e+01  4.14023715e+00  3.21286397e-09
      -5.97911942e-10  1.43783652e-10]
```



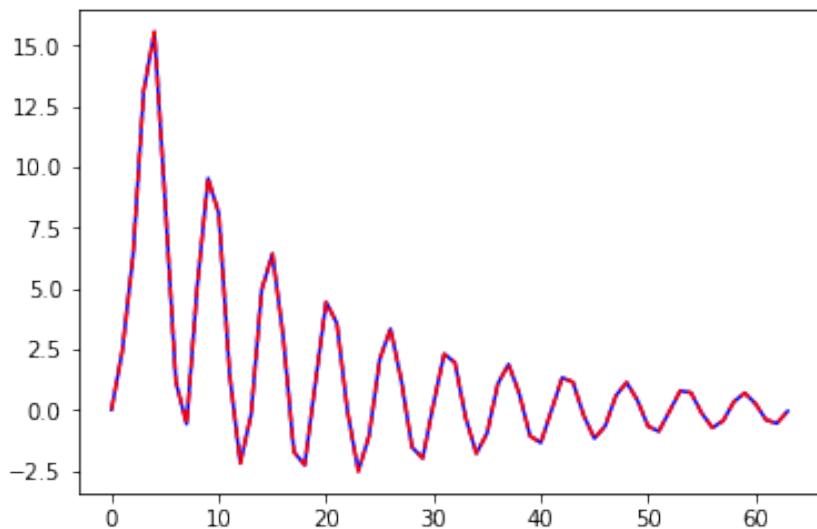
```
### na_est = 10 :
```



```

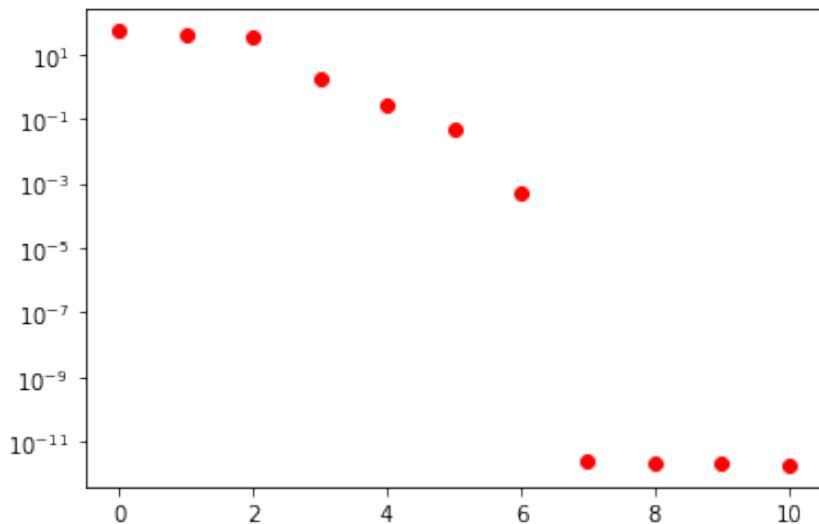
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.28177899 -0.00389963  0.09373541 -0.05976567 -0.22868026
      -0.44838758  0.27544686  0.00989171 -0.00361646 -0.00108268]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  5.61973042e+00  1.13367203e+01
      1.20487970e+01  4.55432480e+00 -1.00606773e+00 -2.78917493e+00
      -3.22448290e-09  6.49402310e-10 -1.15642607e-10]

```

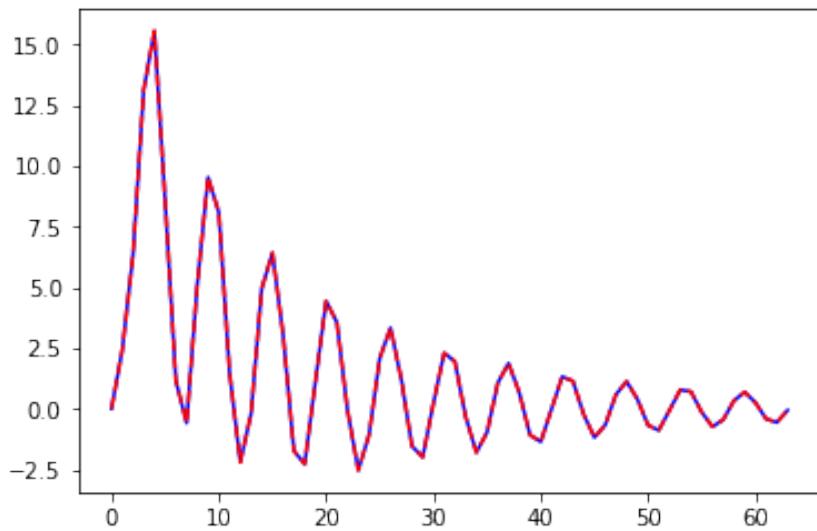


```
### na_est = 11 :
```

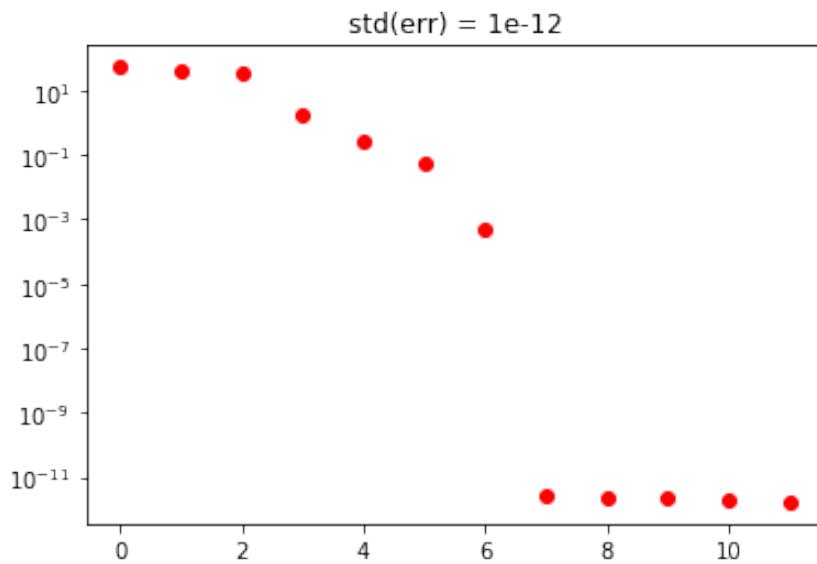
std(err) = 1e-12



```
a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -1.77473691e-01 -1.69476561e-01  1.44148465e-02
      -1.87148289e-01 -1.29432263e-01 -5.23845047e-01  1.89388490e-01
      2.33266638e-01 -1.61019903e-03 -5.18803483e-03 -8.88782624e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  5.87006314e+00  1.15960419e+01
      1.21844576e+01  3.19827295e+00 -4.29385031e+00 -6.54306128e+00
      -2.28966862e+00  2.54878074e-09 -4.96843455e-10  6.98403557e-11]
```



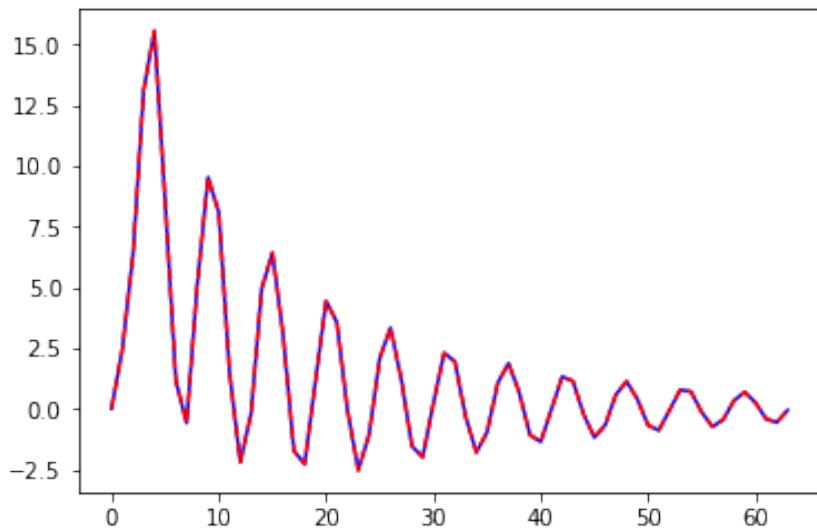
```
### na_est = 12 :
```



```

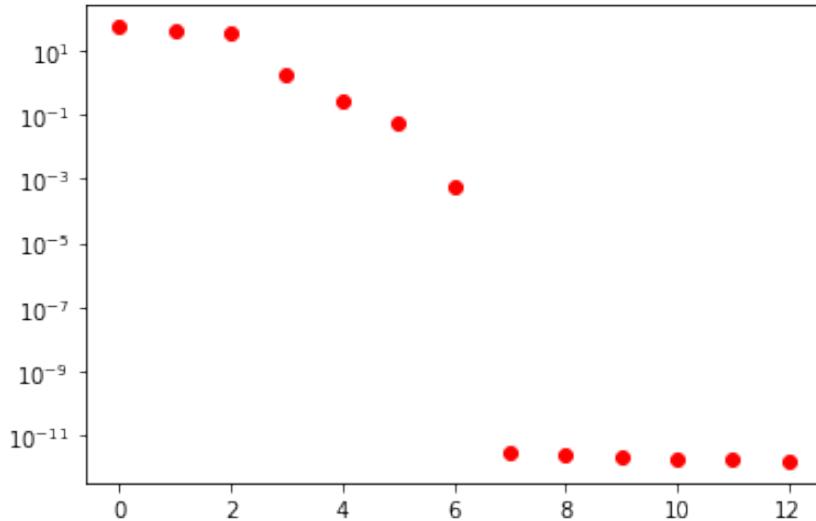
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -2.00857666e-01 -1.27871174e-01 -3.28545134e-02
      -1.87090393e-01 -1.34797999e-01 -5.25945533e-01  1.98215050e-01
      2.13289738e-01  2.75204951e-02 -4.82192013e-03 -1.39450918e-03
      -1.17182896e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  5.81394160e+00  1.15486693e+01
      1.20261571e+01  3.08316081e+00 -4.47945273e+00 -6.98770323e+00
      -2.69301993e+00 -3.01884865e-01 -6.25261620e-09  1.20298438e-09
      -2.30886421e-10]

```

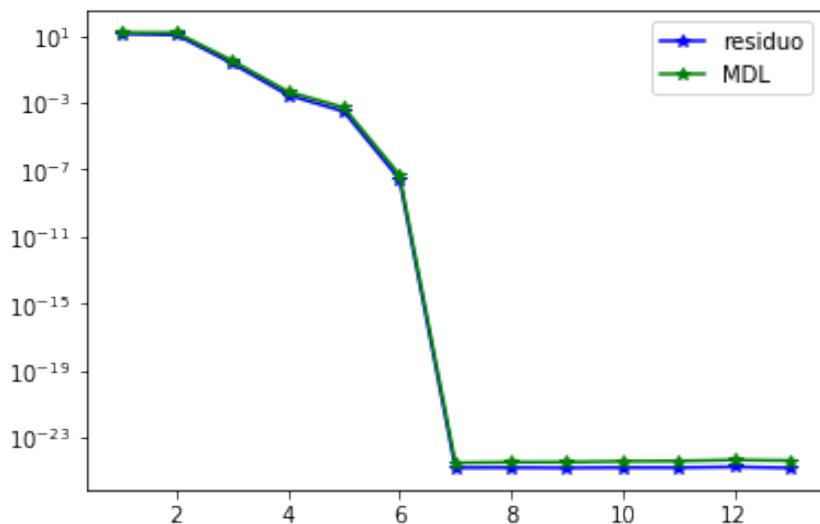
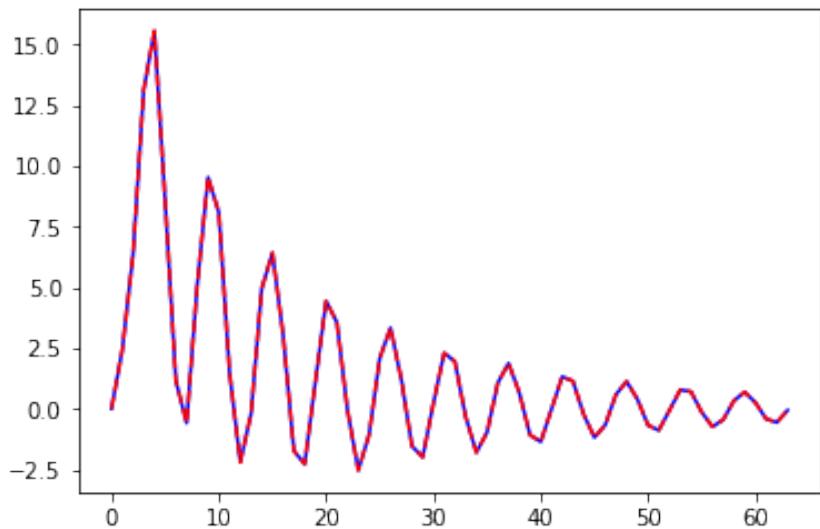


```
### na_est = 13 :
```

std(err) = 1e-12

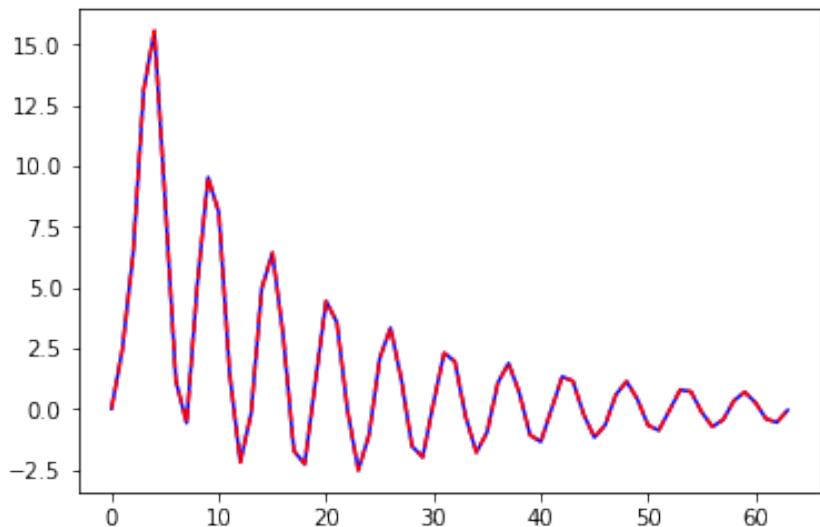


```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -2.63693589e-01 -1.42378236e-01 -1.01335744e-01
      -6.48583677e-02 -8.96754869e-02 -4.52497074e-01  1.98742683e-01
      2.51225399e-01  5.96475985e-02 -1.37457987e-01 -2.20234488e-03
      2.38966949e-03  5.27344266e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.31810915e-13  2.40000000e+00  5.66313539e+00  1.11182374e+01
      1.09460484e+01  1.77821722e+00 -5.26215212e+00 -6.18471348e+00
      -3.01804733e-01  2.10834693e+00  1.35853650e+00  5.63381608e-09
      -1.09917231e-09  1.80635229e-10]
```



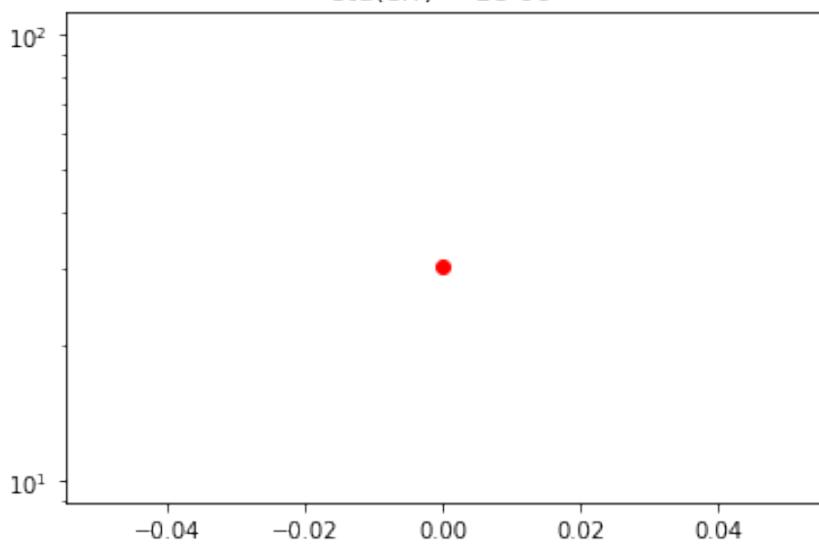
```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738]
```

```
0.01181941  0.00287246]  
b = [0.  2.4 4.4 8.4 7.4]  
b_est = [9.31810915e-13 2.40000000e+00 4.40000000e+00 8.40000000e+00  
7.40000000e+00]
```

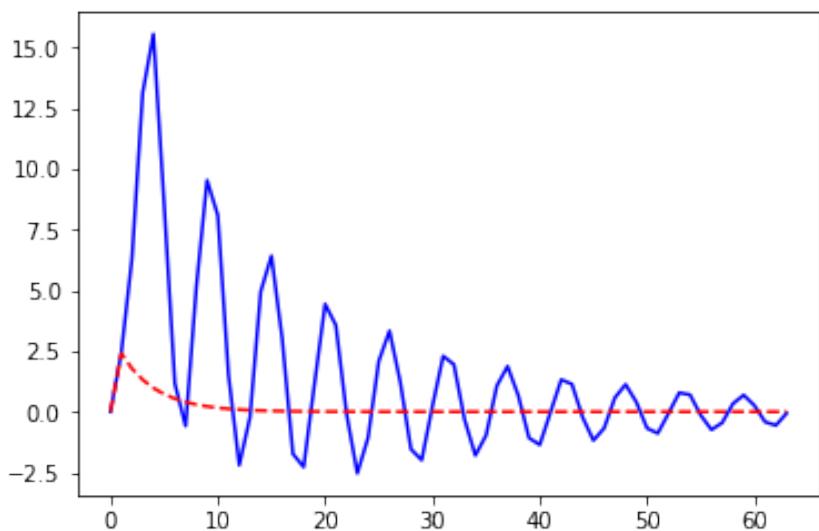


```
std = 1e-06 :  
### na_est = 1 :
```

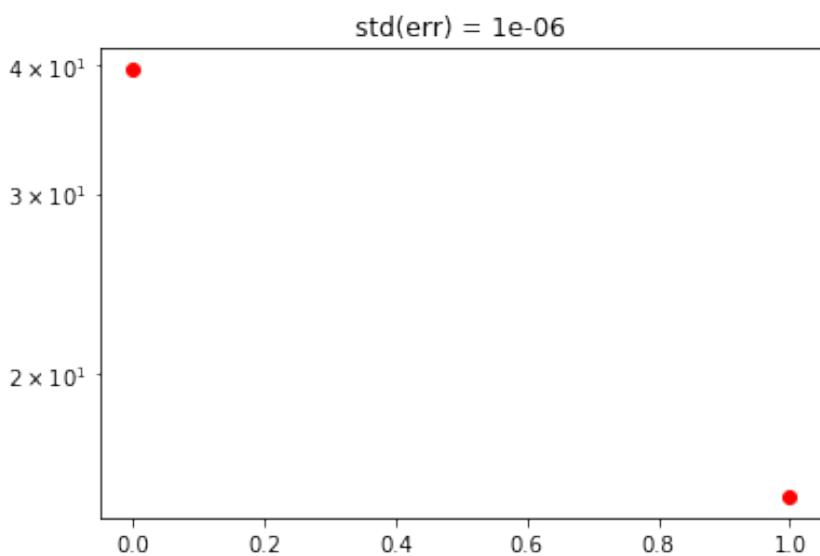
std(err) = 1e-06



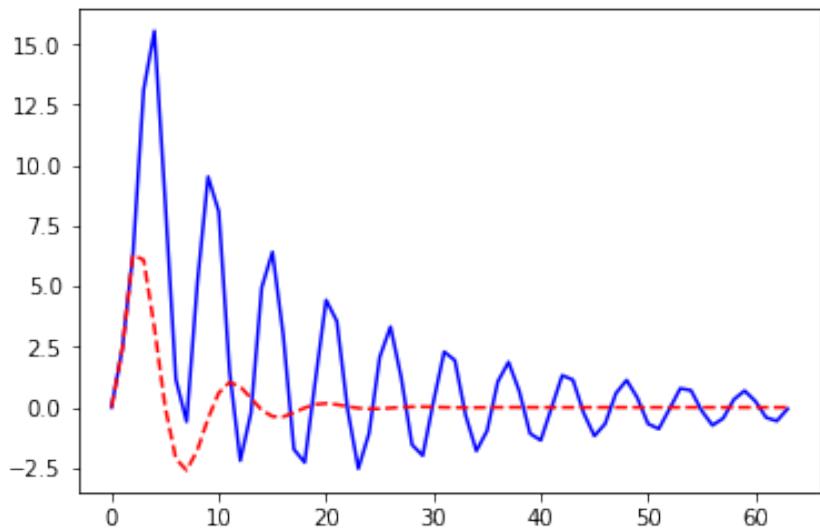
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74094937]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [8.92750227e-07 2.39999943e+00]
```



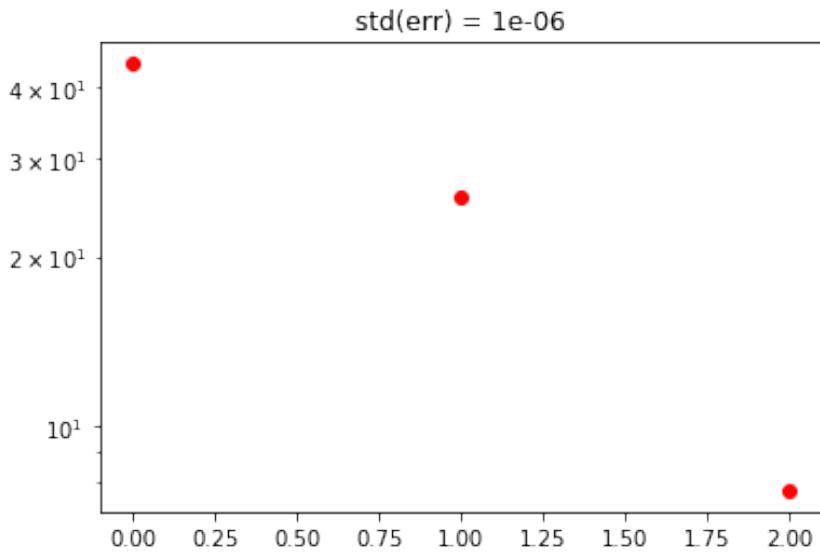
```
### na_est = 2 :
```



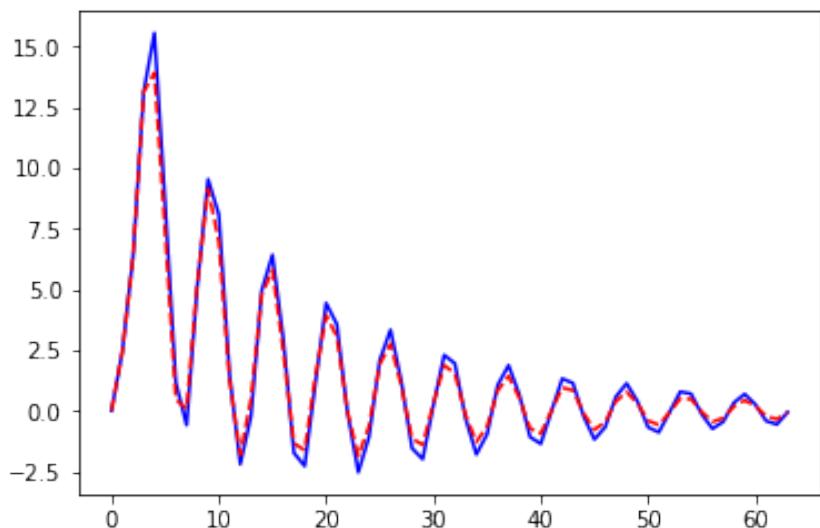
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.2162506    0.65362329]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [8.92750227e-07 2.39999901e+00 3.37699983e+00]
```



```
### na_est = 3 :
```

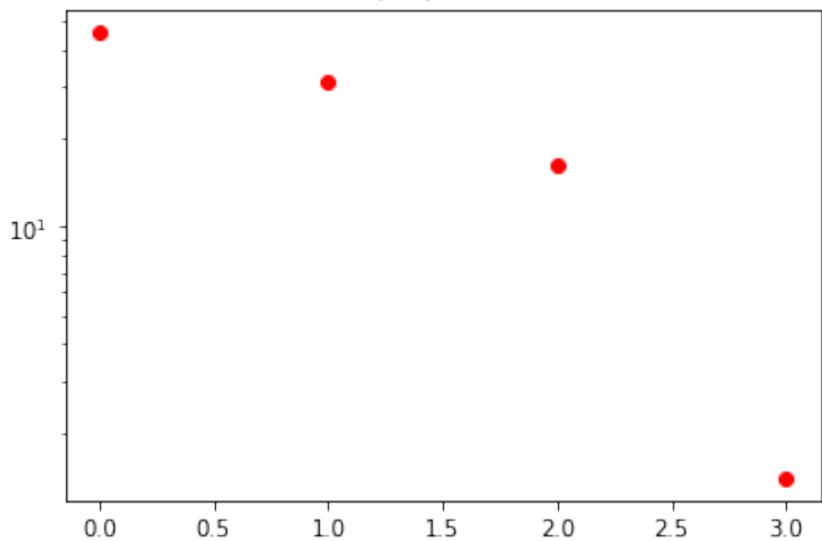


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.68740904  1.61319284 -0.80094769]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [8.92750227e-07 2.39999859e+00 2.24622039e+00 6.36789457e+00]
```

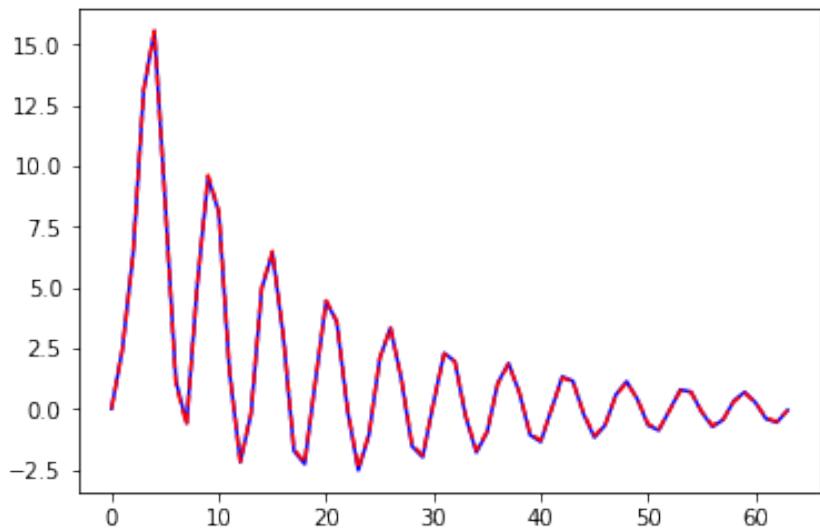


```
### na_est = 4 :
```

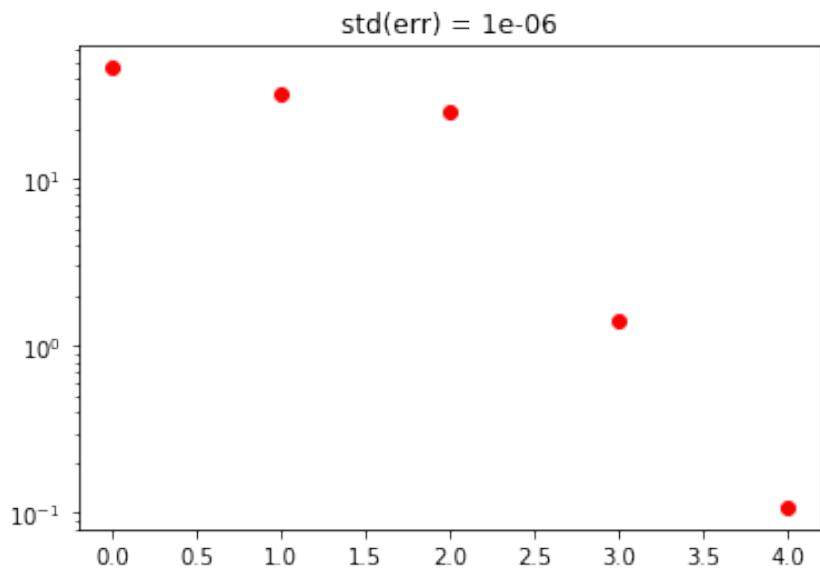
std(err) = 1e-06



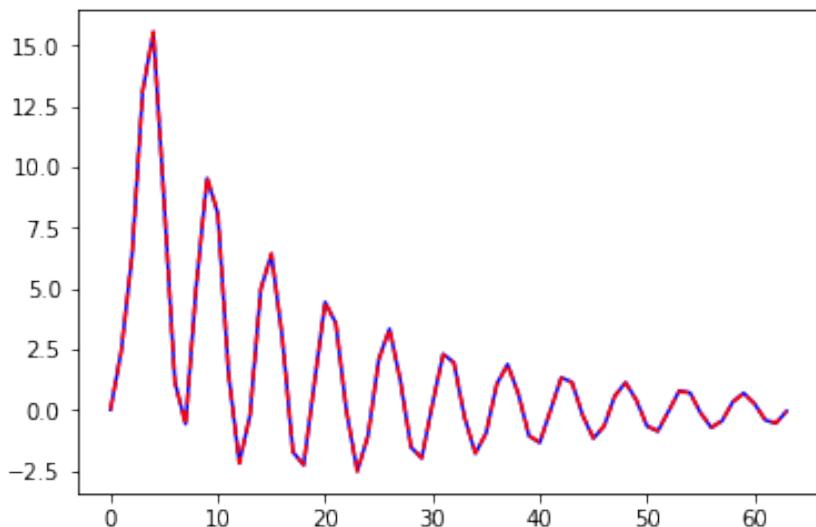
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.79406252  0.12323564  0.63409944      -0.71855879]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [8.92750227e-07 2.39999939e+00 4.39025078e+00 8.41650878e+00
    7.42487194e+00]
```



```
### na_est = 5 :
```

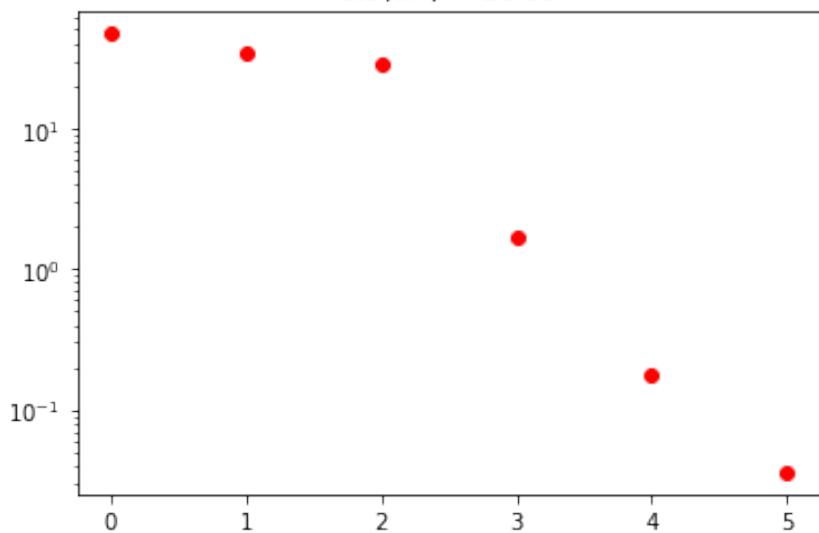


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.37353798  0.571381     0.58688389 -1.11125683  0.4303271 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999887e+00  2.99951002e+00  5.84367970e+00
      2.53026656e+00 -4.28841360e+00]
```

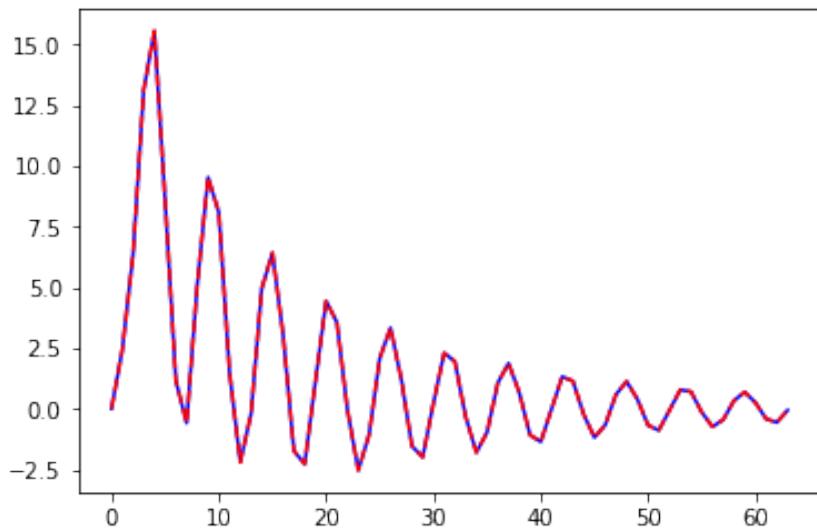


```
### na_est = 6 :
```

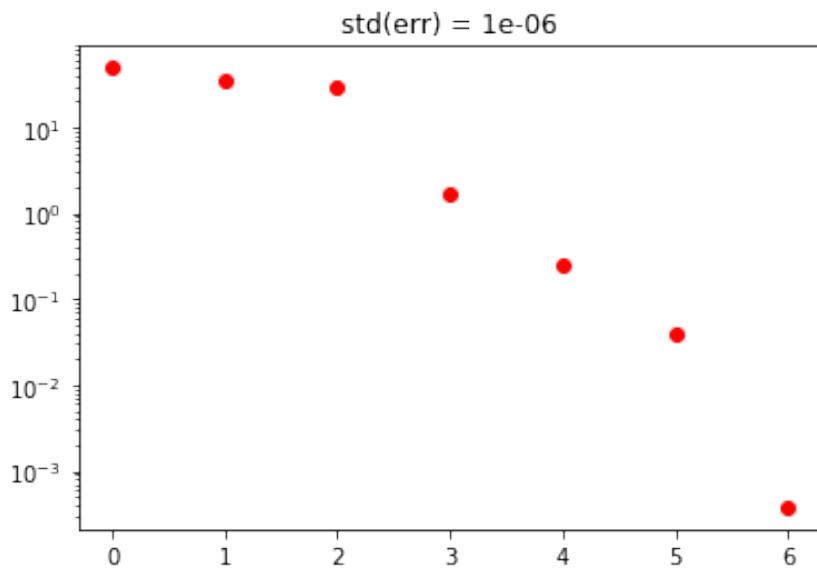
$\text{std}(\text{err}) = 1e-06$



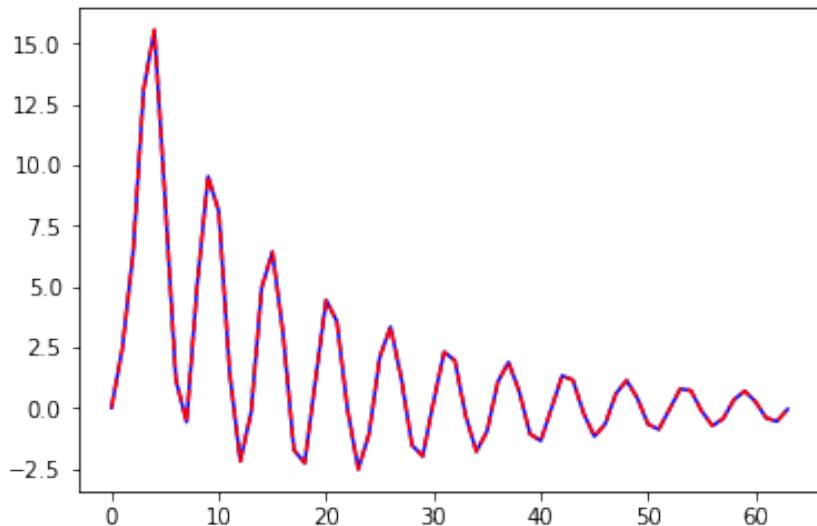
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.98269116  0.28884769  0.60274479 -0.83901823  0.14508846
     -0.00773956]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999922e+00  3.93754219e+00  7.62637167e+00
     5.91747629e+00 -1.16620475e+00  2.28775443e-01]
```



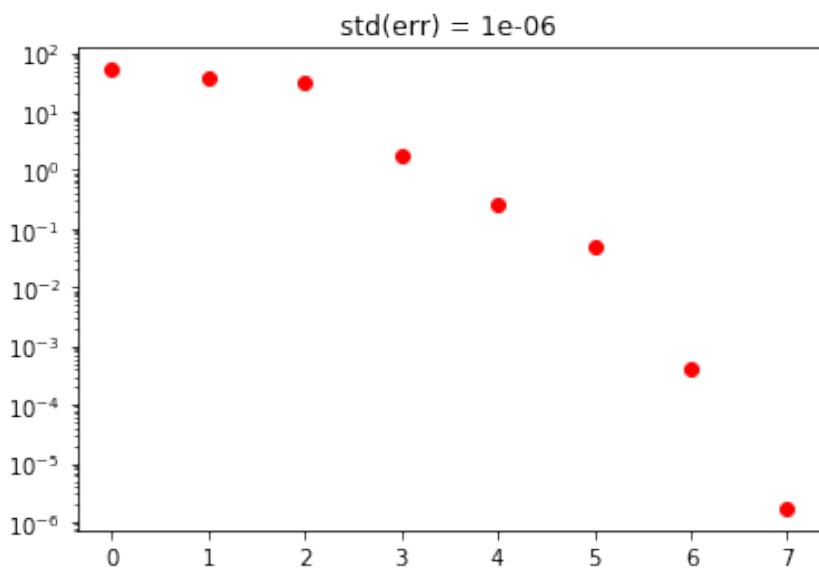
```
### na_est = 7 :
```



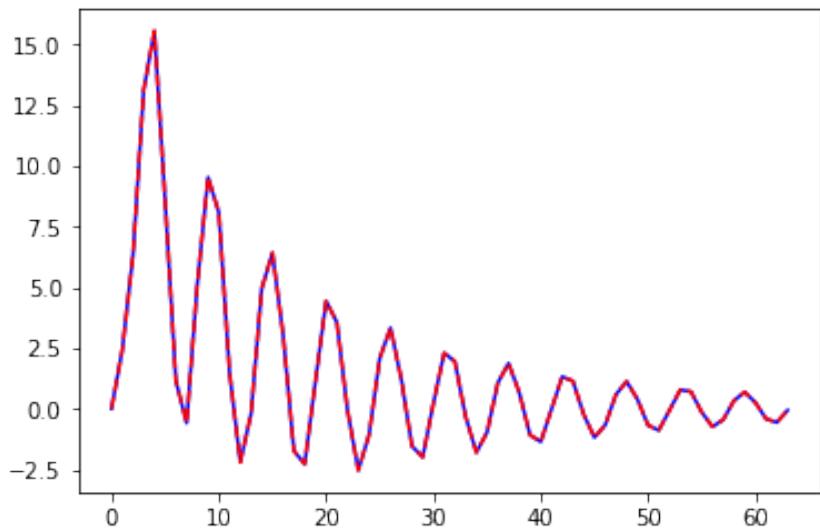
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79009168  0.10578772  0.64750848 -0.71752872 -0.01314173
      0.01181066  0.0028701 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999939e+00  4.39978078e+00  8.39963422e+00
      7.39929967e+00 -5.43926754e-04  1.24194142e-04  1.03065551e-05]
```



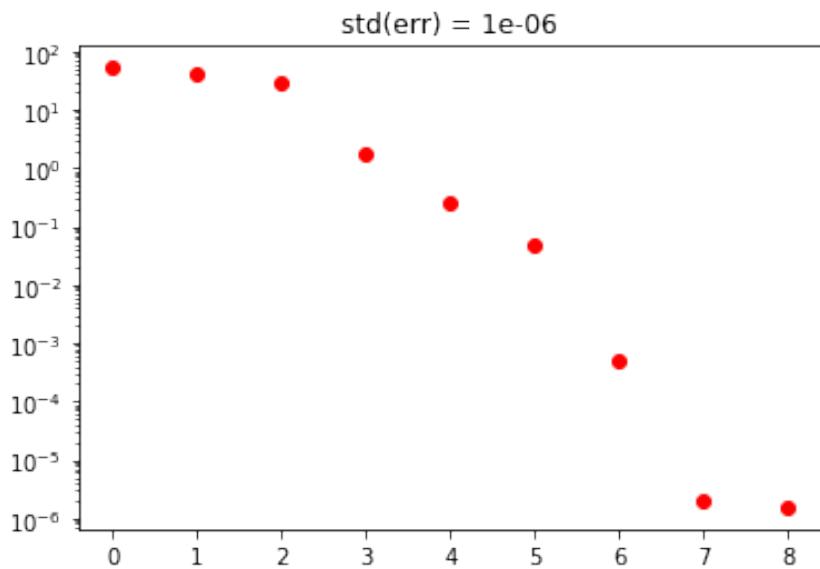
```
### na_est = 8 :
```



```
a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.94225568e-01 -1.27842551e-01  6.78678524e-01
      -5.25914809e-01 -2.25355264e-01   7.81240142e-03  6.38367211e-03
      8.48064088e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999965e+00  5.10985927e+00  9.70169485e+00
      9.88498258e+00  2.18965742e+00  7.38230308e-04 -1.05916992e-04
      3.39063984e-05]
```



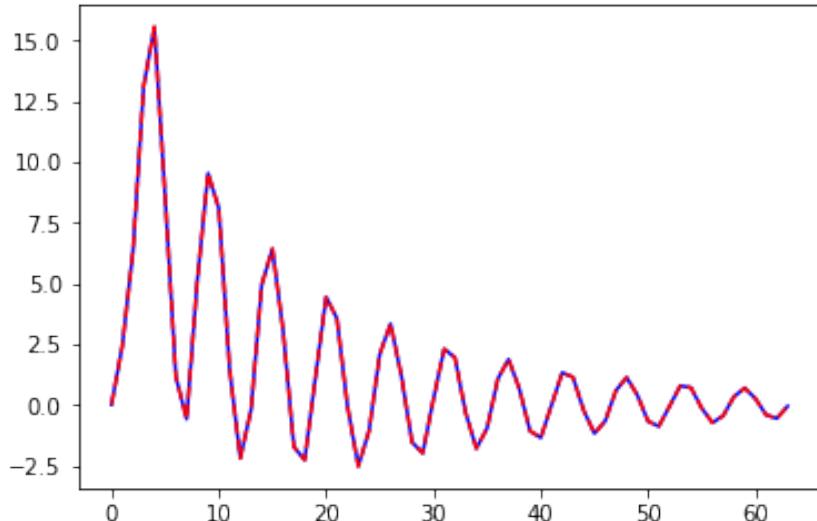
```
### na_est = 9 :
```



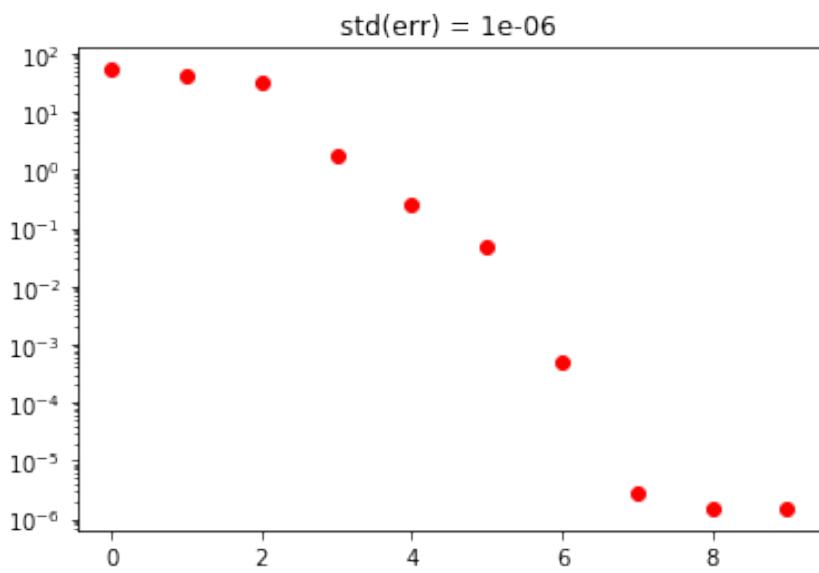
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.46543909e-01 -2.60354573e-02  5.74830487e-01
      -4.81573974e-01 -1.68872626e-01 -9.21070309e-02  4.01401579e-03
      2.82445873e-03  3.89054660e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999970e+00  5.22429535e+00  1.02462356e+01
      1.09023159e+01  3.71920267e+00  1.04275217e+00  7.56830102e-03
      -1.47455385e-03  3.12506784e-04]

```



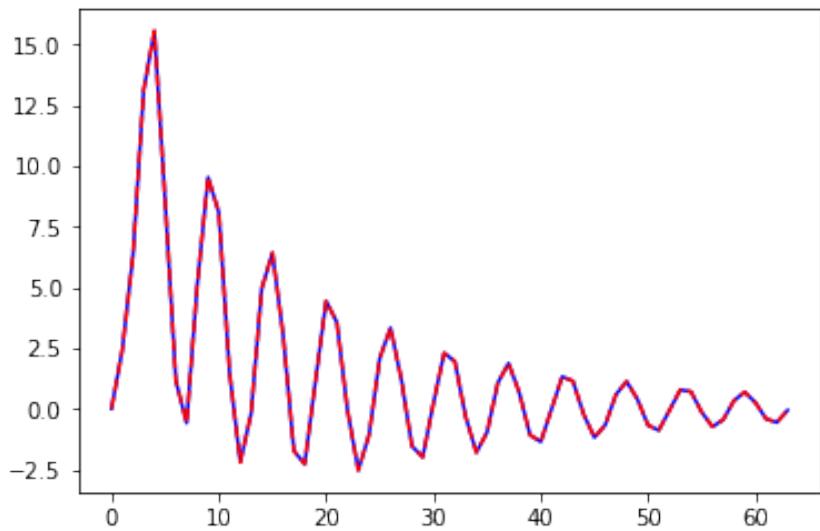
```
### na_est = 10 :
```



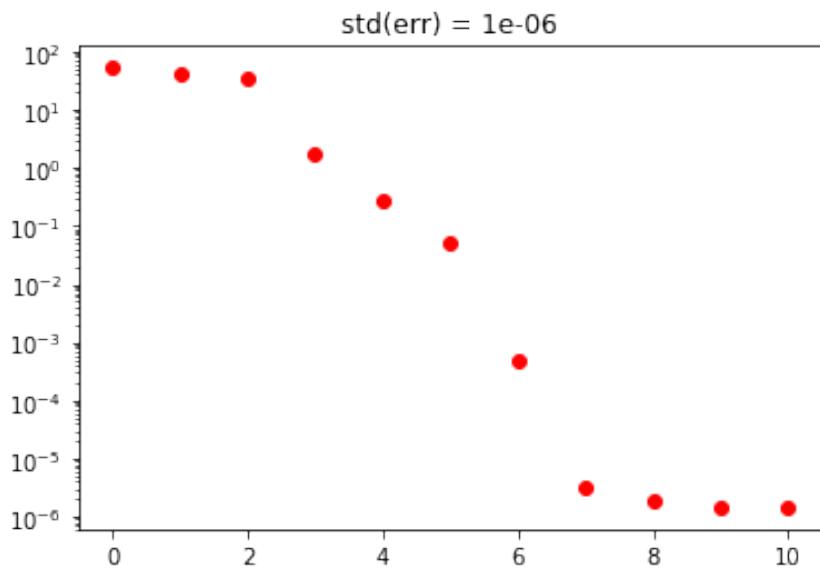
```

a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -5.00979257e-01 -8.17089202e-02  3.01416315e-01
      -2.54204583e-01 -2.30256108e-01 -2.44383838e-01  2.52651893e-01
      6.06018207e-03 -3.97891522e-03 -9.91812674e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999965e+00  5.09365046e+00  9.76989400e+00
      9.18140138e+00  9.66810892e-01 -2.59058335e+00 -2.54989755e+00
      -1.40740964e-03  2.89955598e-04 -7.94536783e-06]

```



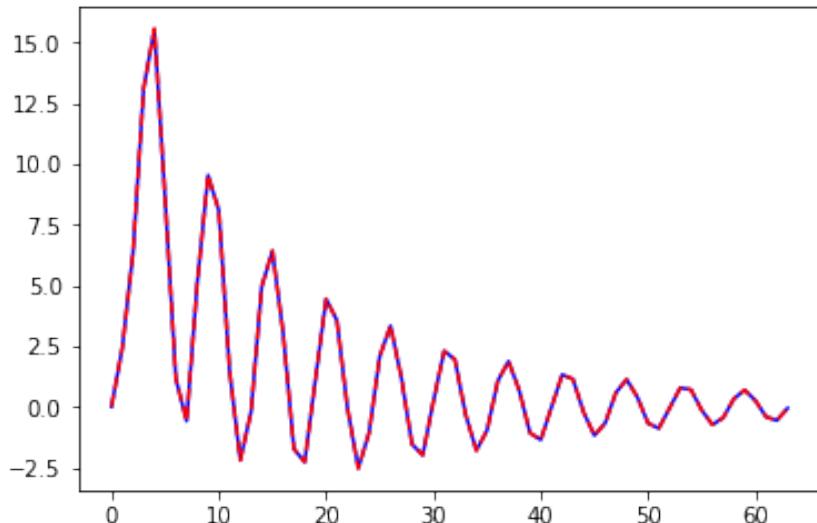
```
### na_est = 11 :
```



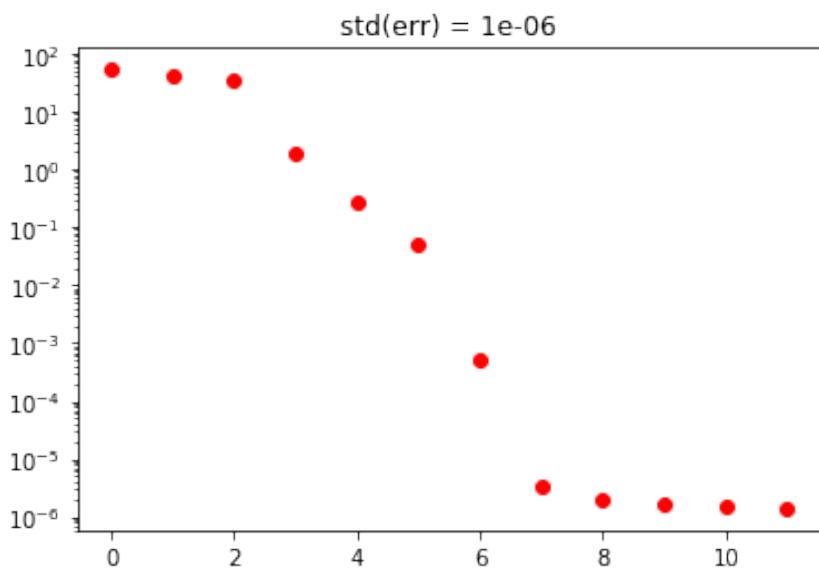
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.40562152e-01 -7.91236826e-02  2.53527222e-01
      -3.52360860e-01 -1.20225911e-01 -3.07579405e-01  1.63536139e-01
      1.17933221e-01 -2.48668780e-03 -2.76426234e-03 -4.47778346e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999970e+00  5.23865152e+00  1.01564847e+01
      9.87542626e+00  1.40285115e+00 -3.01671772e+00 -3.94917914e+00
      -1.13684904e+00  3.92909438e-03 -7.27015193e-04  1.45320433e-04]

```



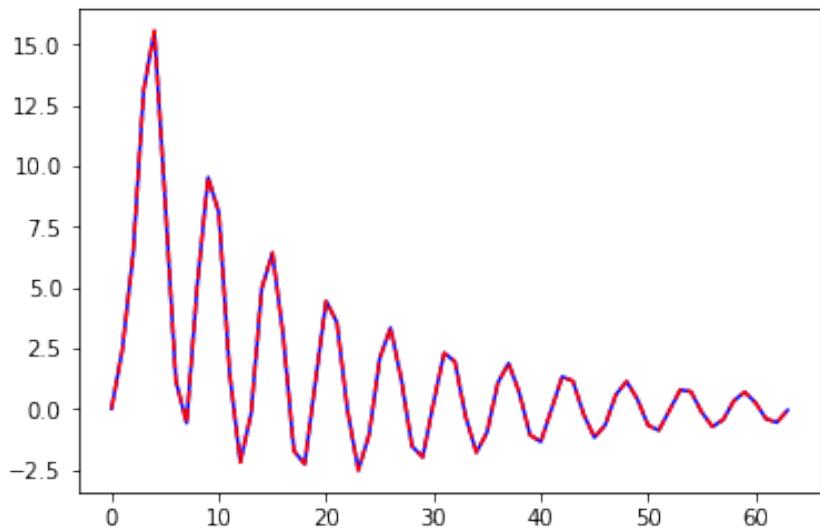
```
### na_est = 12 :
```



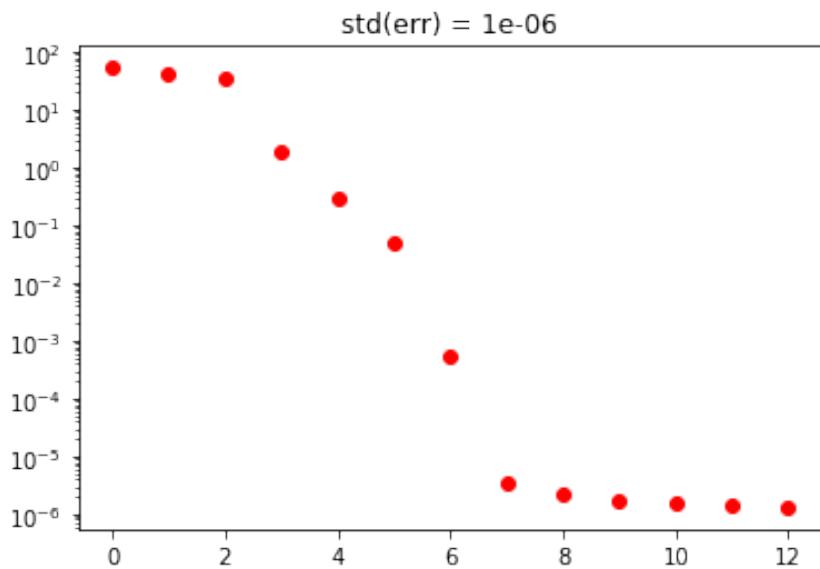
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.56458026e-01  4.72321337e-03  1.12784505e-01
      -3.50184616e-01 -2.11782100e-01 -2.70691248e-01  1.63563383e-01
       4.11315374e-02  1.32014799e-01 -5.40250642e-04 -2.96528047e-03
      -5.33134750e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999969e+00  5.20050150e+00  1.02576367e+01
      9.85698743e+00  1.37493532e+00 -3.90162385e+00 -5.89882662e+00
     -3.16749574e+00 -1.39441933e+00 -5.58776341e-03  1.09810876e-03
     -1.96763361e-04]

```



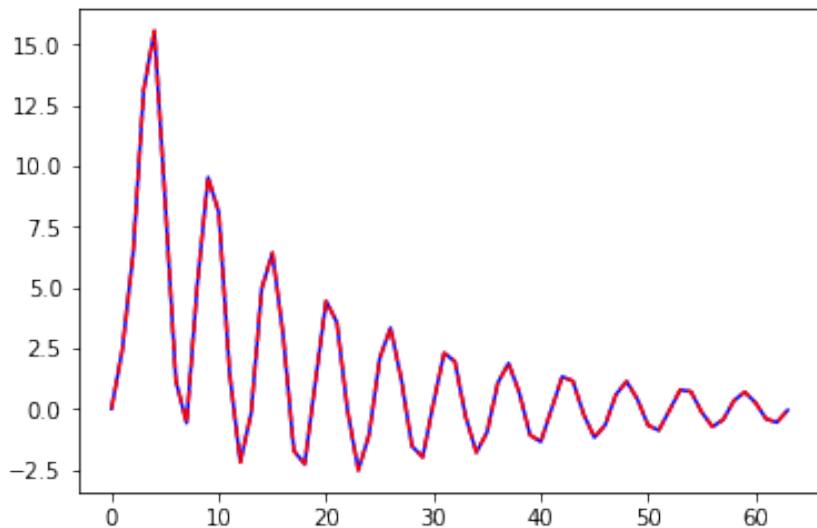
```
### na_est = 13 :
```

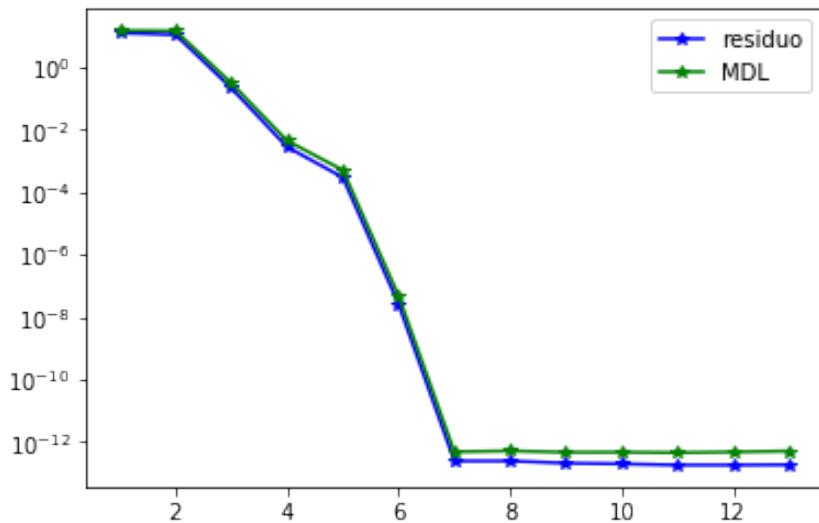


```

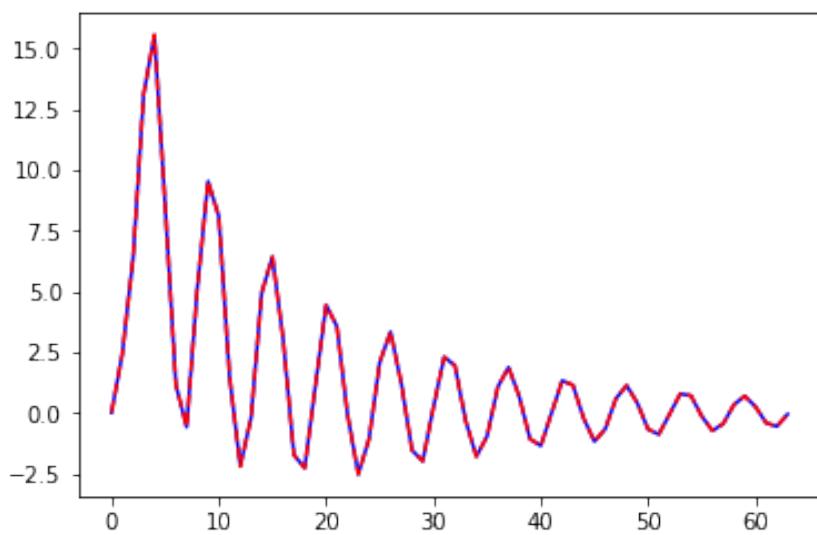
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -4.68329538e-01  9.77134121e-03  1.00253968e-01
      -3.29770548e-01 -2.07182932e-01 -2.49814869e-01  1.57939063e-01
      4.17447352e-02  1.45200227e-01 -2.65066732e-02 -3.33743627e-03
      -4.65122800e-05  1.03082616e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 8.92750227e-07  2.39999968e+00  5.17200987e+00  1.01950091e+01
      9.70294103e+00  1.22672202e+00 -3.94935884e+00 -5.71733973e+00
      -2.76606299e+00 -9.86993356e-01  2.63706472e-01  1.83845338e-03
      -3.41904612e-04  2.97513642e-05]

```

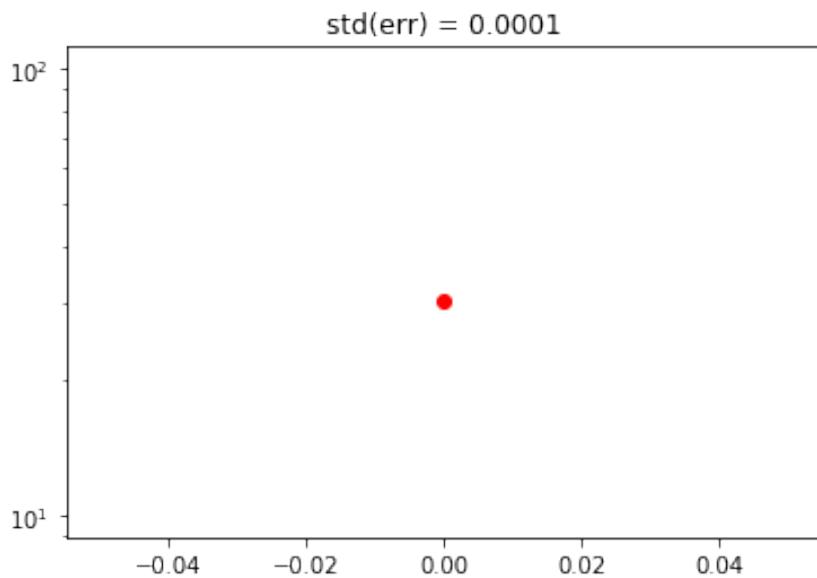




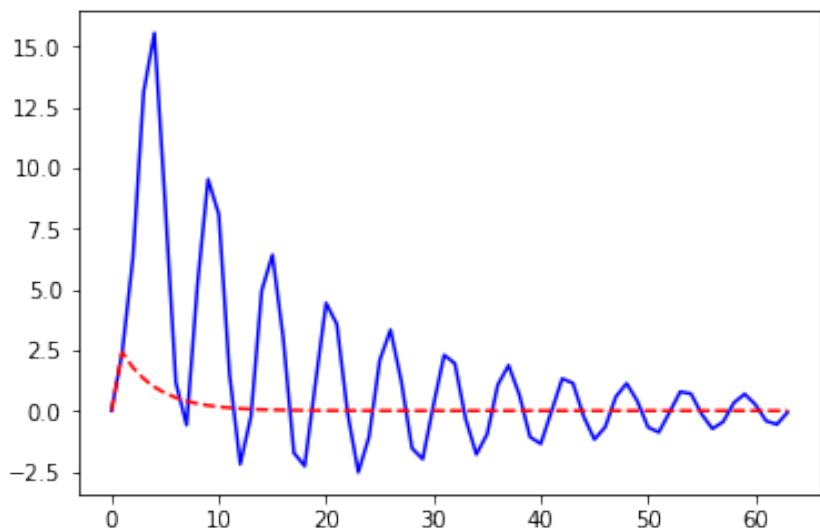
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79009168  0.10578772  0.64750848 -0.71752872 -0.01314173
      0.01181066  0.0028701 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [8.92750227e-07 2.39999939e+00 4.39978078e+00 8.39963422e+00
      7.39929967e+00]
```



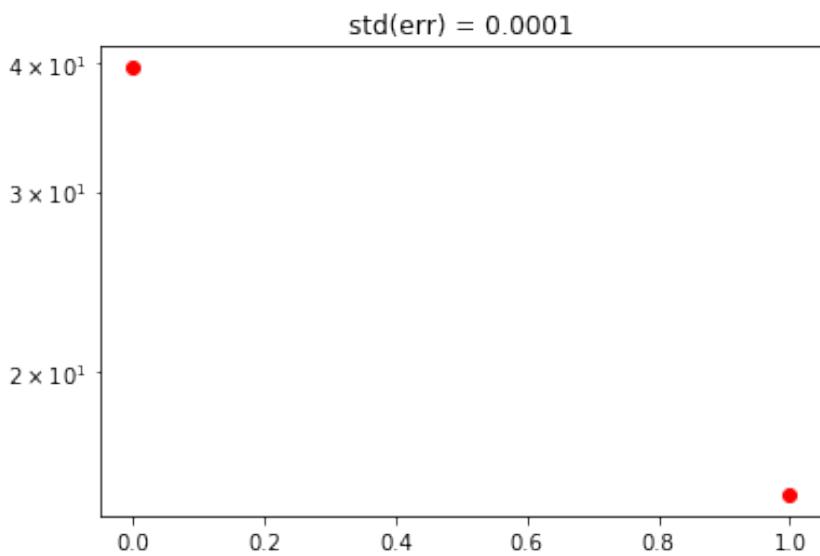
```
std = 0.0001 :  
### na_est = 1 :
```



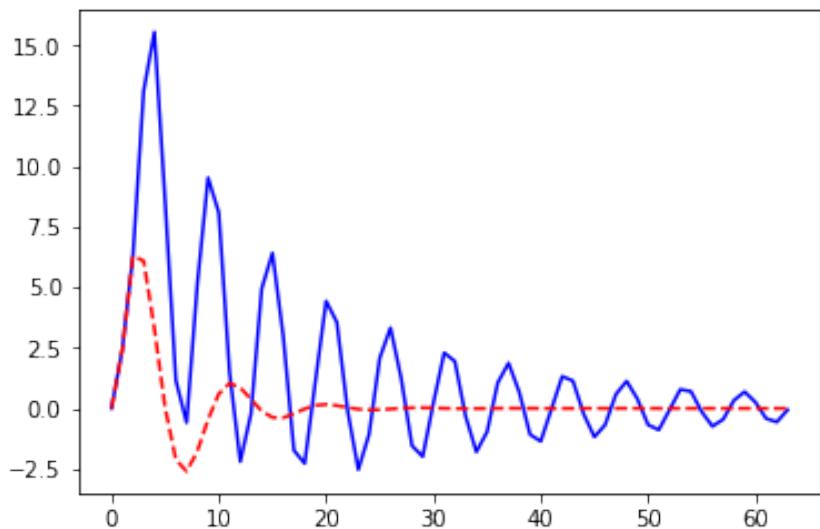
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74095189]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.86284713e-05 2.39992767e+00]
```



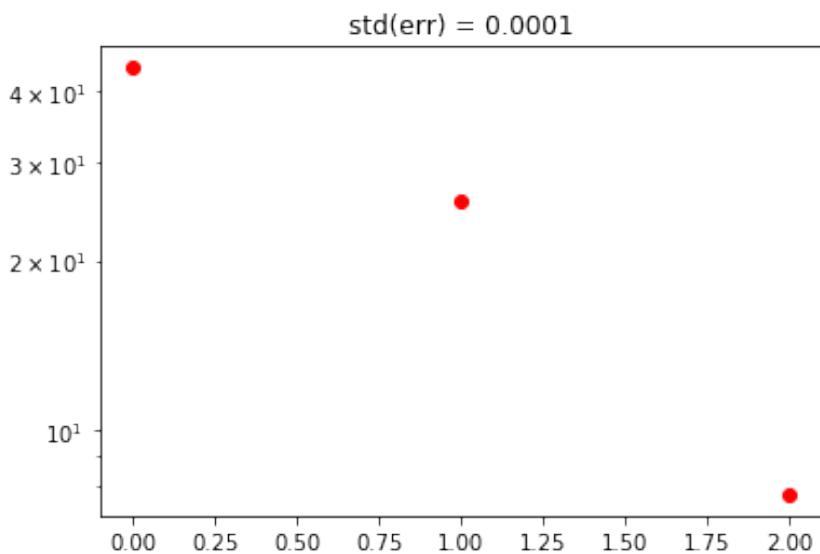
```
### na_est = 2 :
```



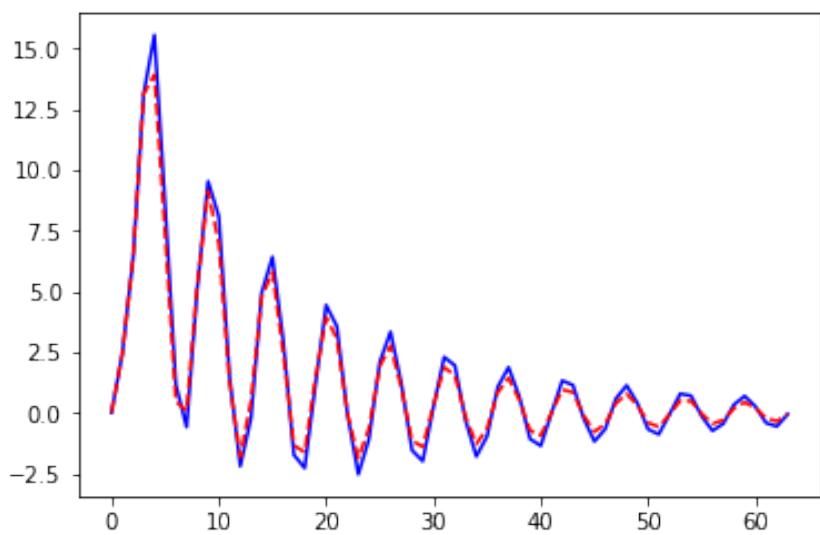
```
a = [ 1.           -0.79           0.1057       0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21625391  0.65362222]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.86284713e-05 2.39988079e+00 3.37712282e+00]
```



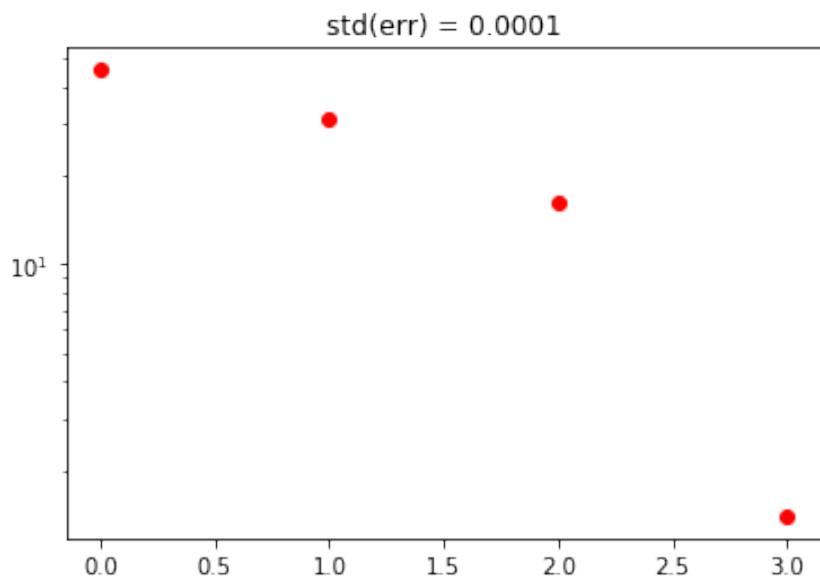
```
### na_est = 3 :
```



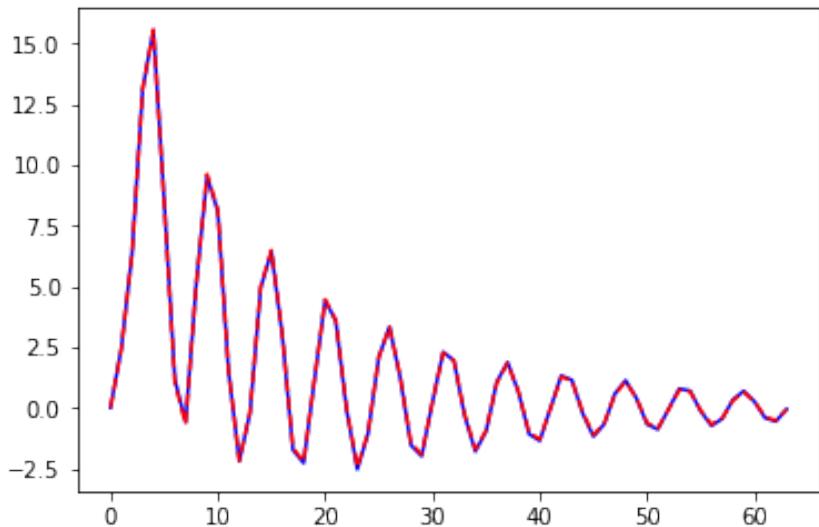
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -1.68741537  1.6132003   -0.80095245]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.86284713e-05 2.39983432e+00 2.24642960e+00 6.36769927e+00]
```



```
### na_est = 4 :
```

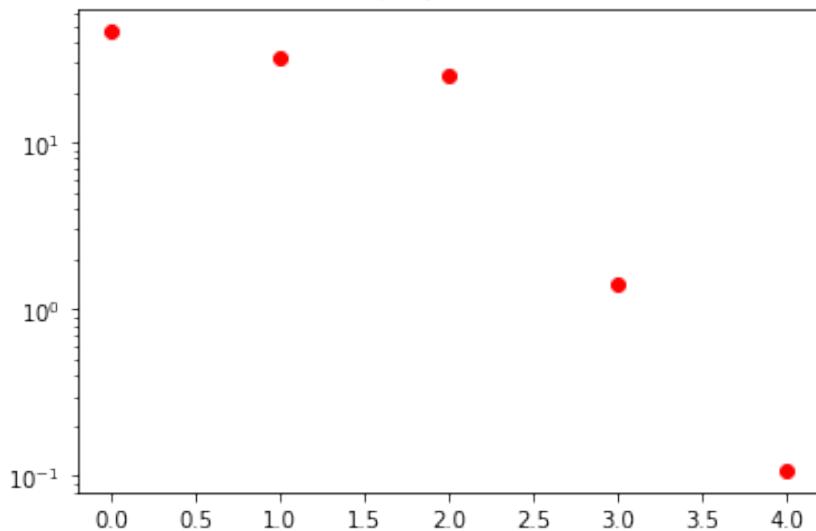


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.79407973  0.12326033  0.63407655 -0.71854899]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [9.86284713e-05 2.39992243e+00 4.39028886e+00 8.41648629e+00
         7.42468793e+00]
```

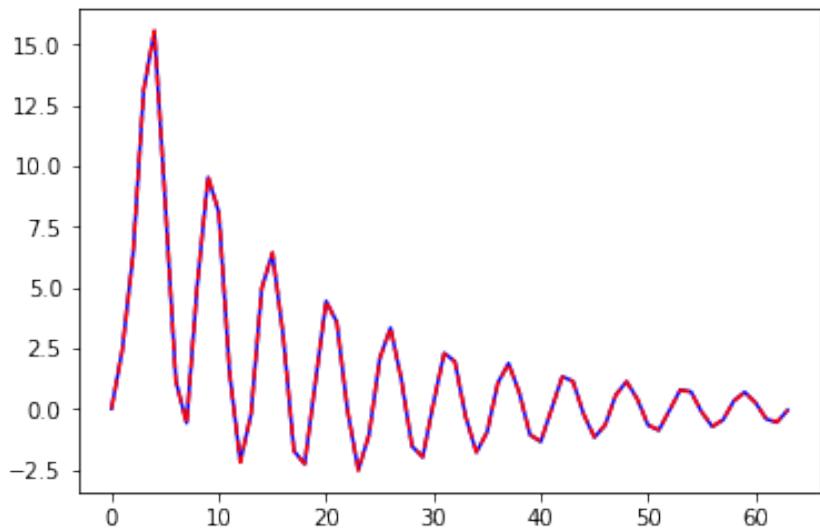


```
### na_est = 5 :
```

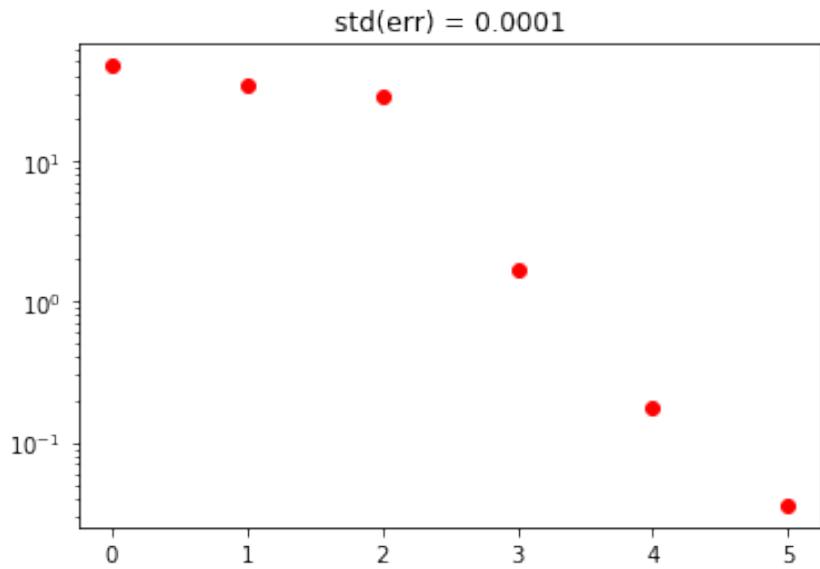
std(err) = 0.0001



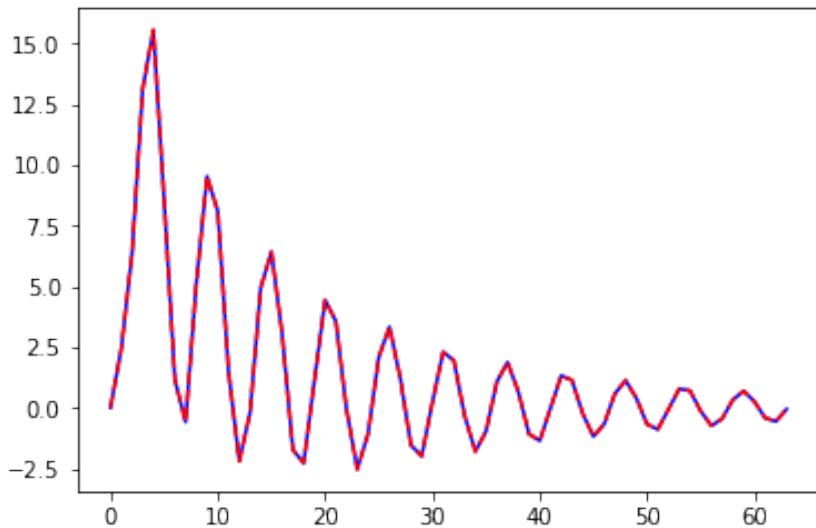
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -1.37383466  0.5716275   0.58683247  -1.11143201  0.43053378]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39986525e+00  2.99892082e+00  5.84238633e+00
        2.52772517e+00 -4.29037057e+00]
```



```
### na_est = 6 :
```

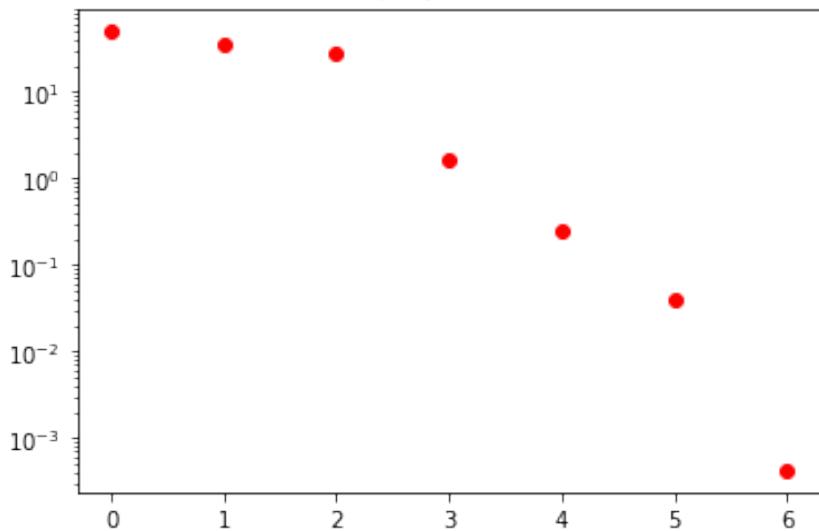


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.98424709  0.29090362  0.60193133 -0.83996525  0.14678968
      -0.00836411]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39990368e+00  3.93390359e+00  7.62152038e+00
      5.90799119e+00 -1.17067254e+00  2.34769967e-01]
```

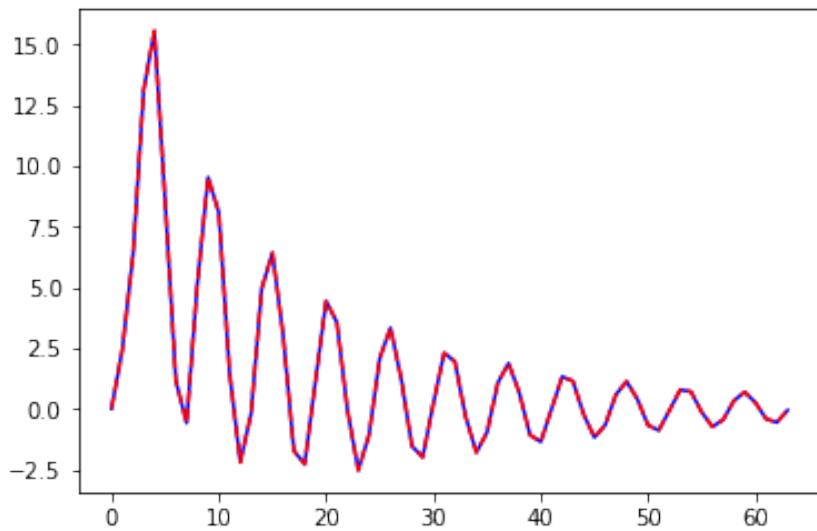


```
### na_est = 7 :
```

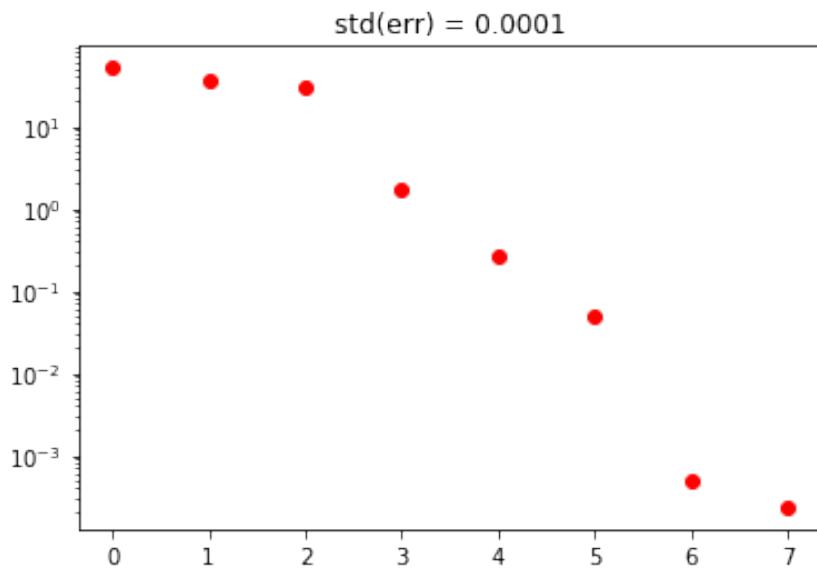
std(err) = 0.0001



```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008   -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.81547666  0.13075051  0.64060723  -0.73320308   0.00826049
     0.00835999  0.00276439]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39992032e+00  4.33893695e+00  8.29974681e+00
    7.20678721e+00 -1.48586648e-01  3.34443791e-02 -2.55874459e-03]
```



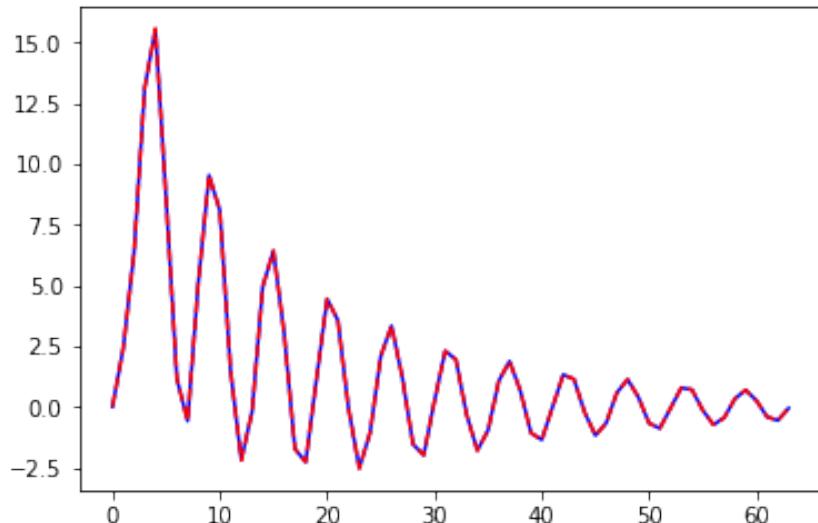
```
### na_est = 8 :
```



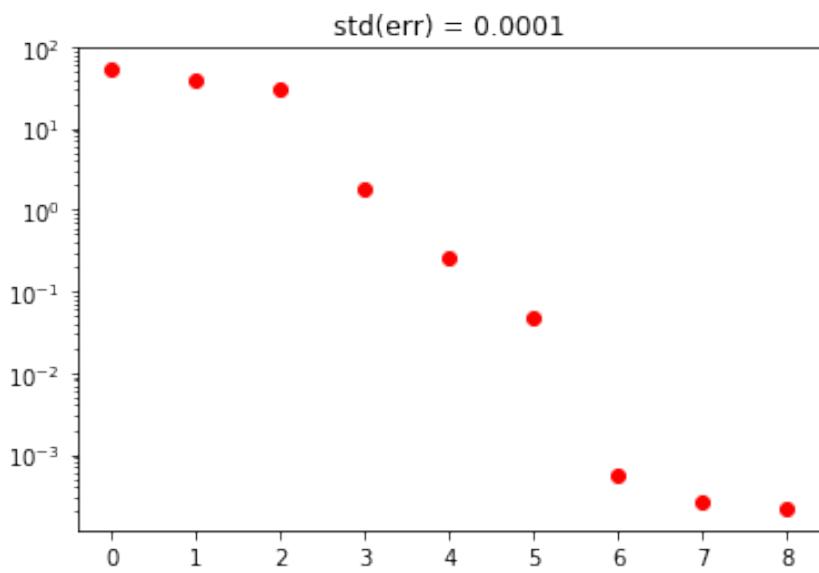
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -3.83770501e-01  1.35465929e-02  4.66595639e-01
      -3.89523039e-01 -1.66320202e-01 -1.84960307e-01  3.99248431e-02
      -2.96225460e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39996290e+00  5.37502050e+00  1.07364918e+01
      1.17153315e+01  4.75394269e+00  1.35965659e+00 -2.63406272e-01
      4.92380978e-02]

```



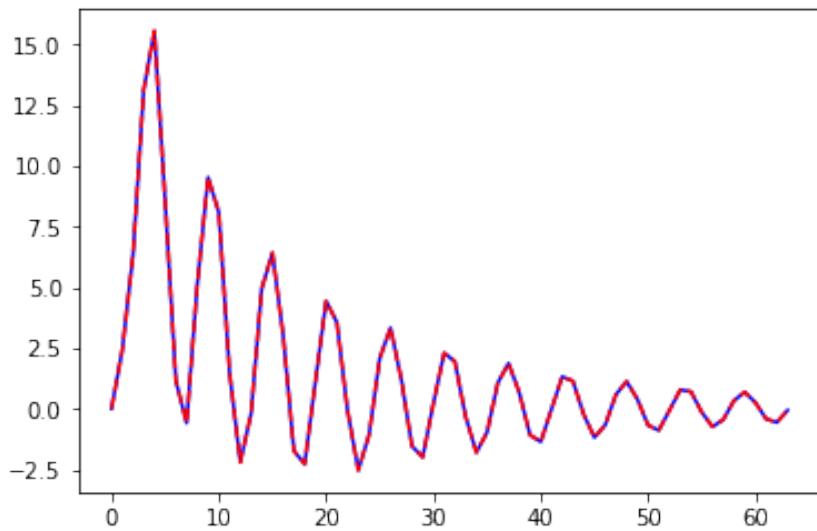
```
### na_est = 9 :
```



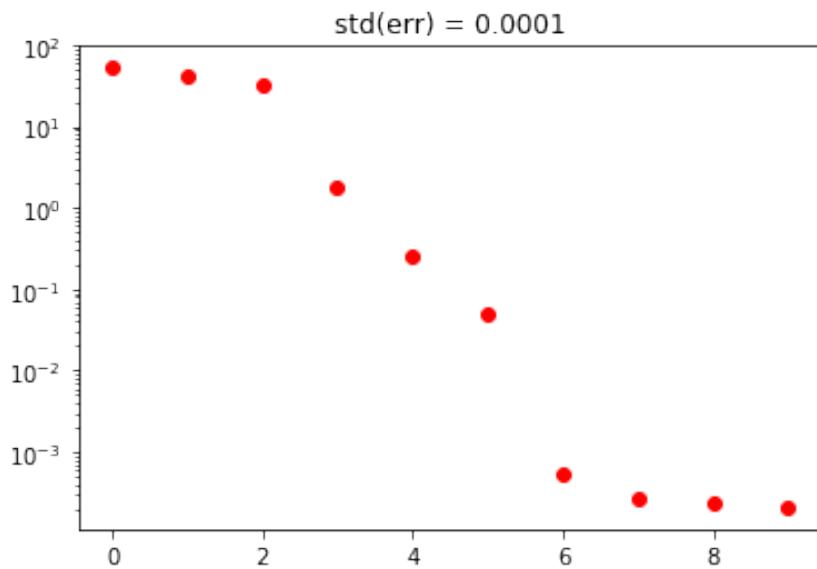
```

a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -2.09331365e-01  1.49186359e-01  2.91742228e-01
      -2.69324071e-01 -1.08850340e-01 -3.69310668e-01  1.96929075e-02
       5.93078910e-03  9.55294101e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39998010e+00  5.79368793e+00  1.21602908e+01
      1.44383651e+01  8.43287892e+00  3.55838655e+00 -1.25236411e-01
      2.38150023e-02  1.75030481e-03]

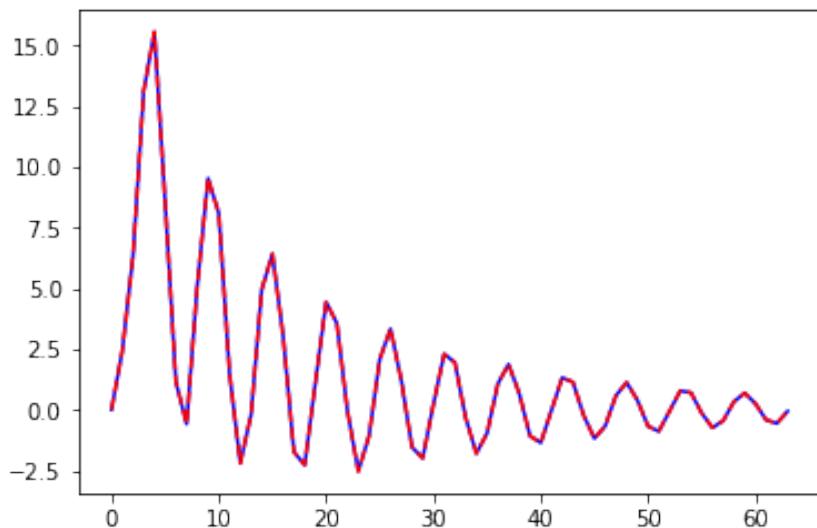
```



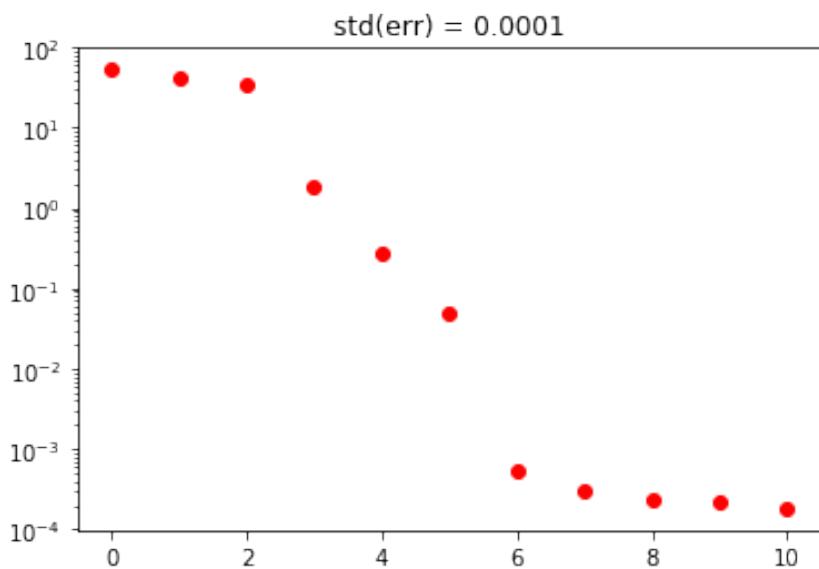
```
### na_est = 10 :
```



```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.38604402 -0.01477966  0.1037847  -0.11219536 -0.20329277
      -0.41883306  0.30297483  0.02327247 -0.00686763 -0.00131738]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39996268e+00  5.36956125e+00  1.06541587e+01
      1.06364393e+01  2.72825484e+00 -2.20292776e+00 -3.30341706e+00
      -9.53729522e-02  2.32615743e-02 -1.54027038e-03]
```



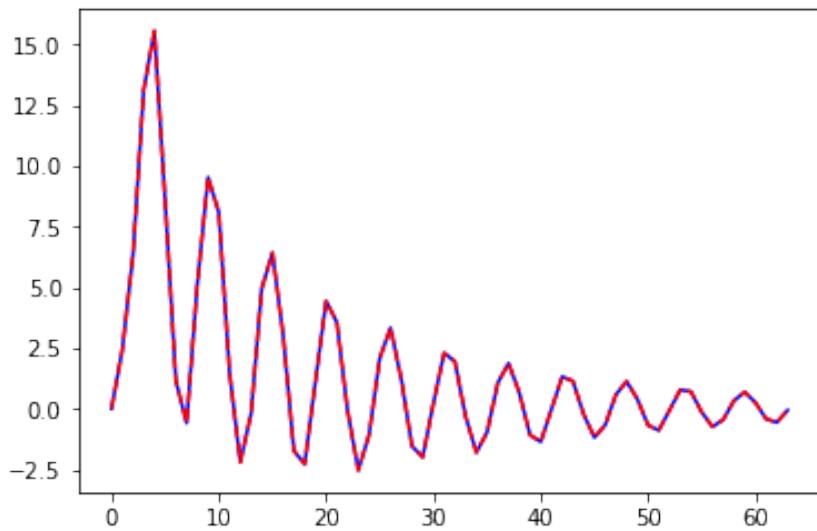
```
### na_est = 11 :
```



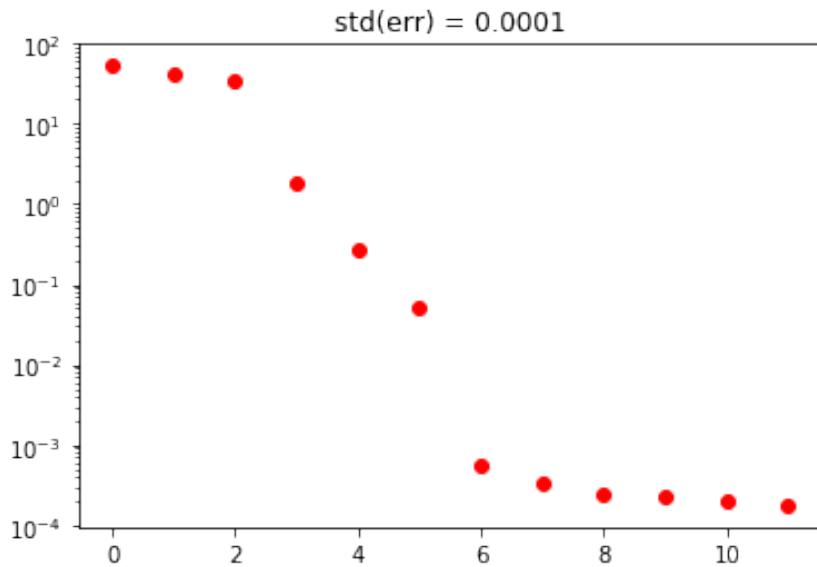
```

a = [ 1.           -0.79          0.1057        0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -2.92950685e-01 -1.03971040e-01  2.97580364e-03
      -1.34710289e-01 -2.10839030e-01 -4.49942658e-01  2.60310722e-01
      1.17601287e-01  4.30189093e-02 -1.21583065e-02 -1.09334151e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39997186e+00  5.59297653e+00  1.10262114e+01
      1.10543418e+01  2.31648358e+00 -4.27699799e+00 -5.94211832e+00
      -1.86123488e+00 -3.23287786e-01  6.66126803e-02 -1.28653217e-02]

```



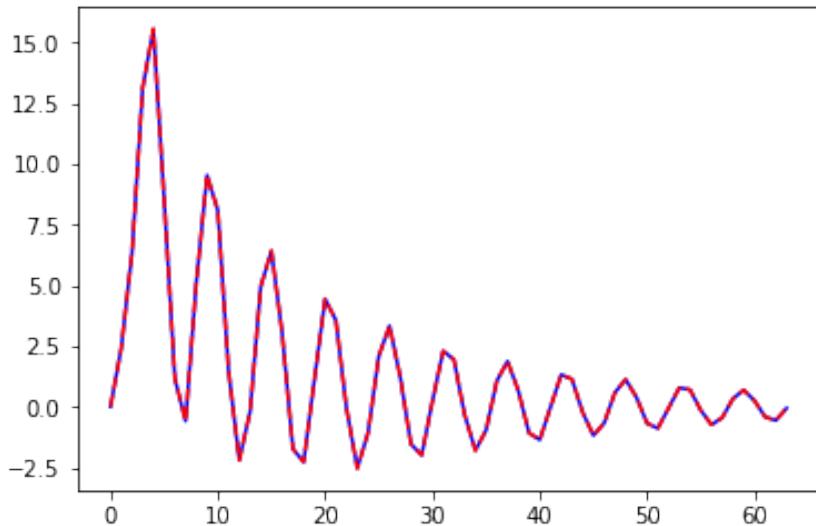
```
### na_est = 12 :
```



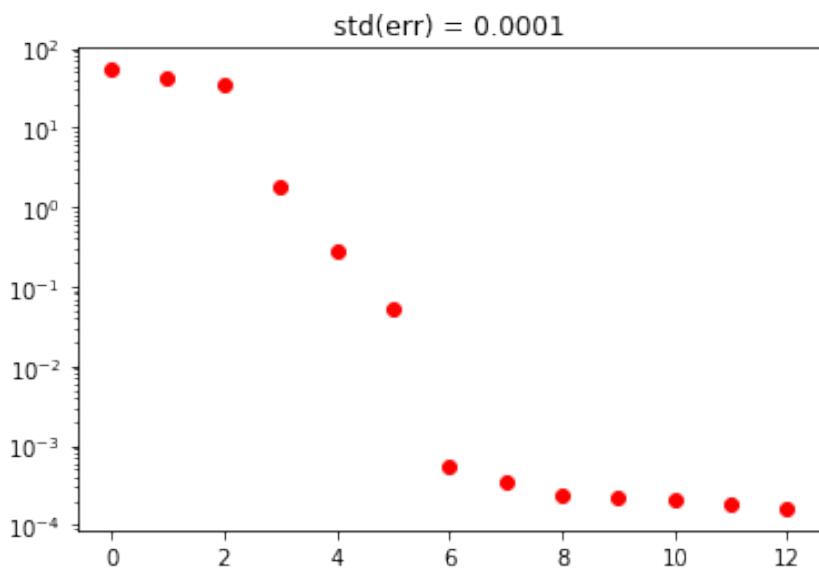
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.22262123  0.0602838  -0.18115924 -0.22410928 -0.31482428
      -0.38409942  0.13066226  0.02920091  0.27838361 -0.00462479 -0.00501339
      -0.00111759]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39997879e+00  5.76178349e+00  1.18632041e+01
      1.25693037e+01  4.19094048e+00  -4.35133900e+00  -8.99027702e+00
      -5.93568062e+00  -2.81623170e+00  2.68074447e-02  -5.18377703e-03
      7.31972116e-04]

```



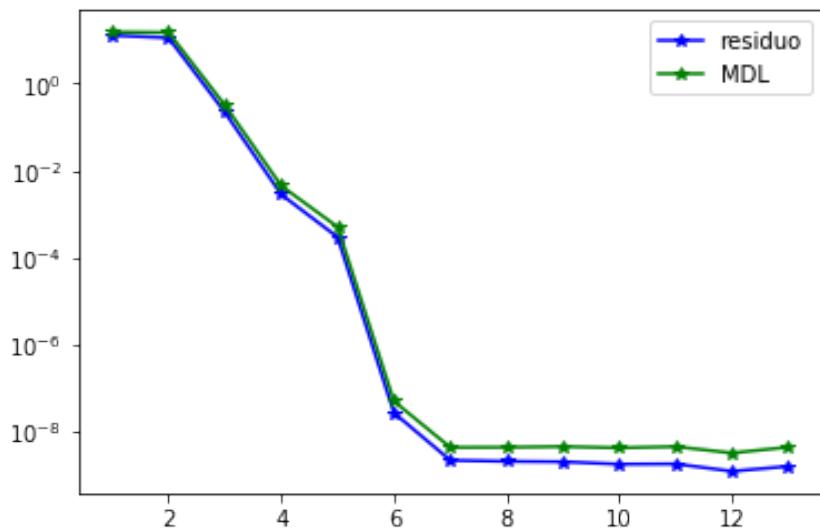
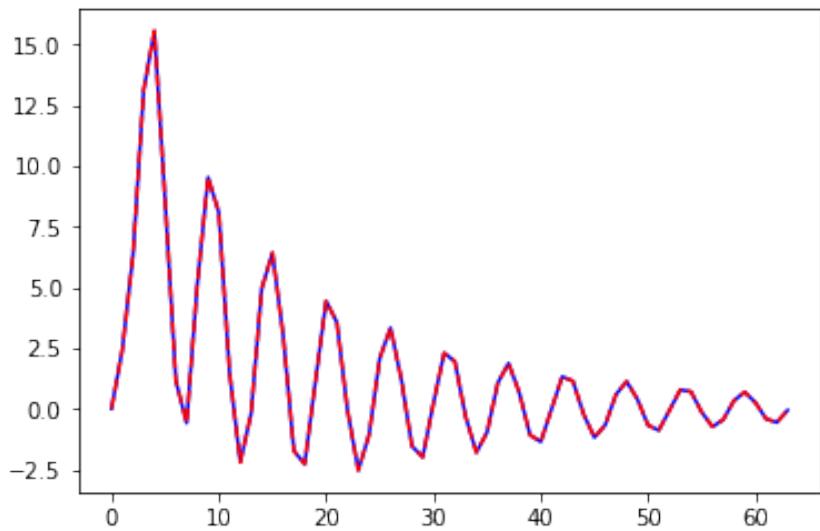
```
### na_est = 13 :
```



```

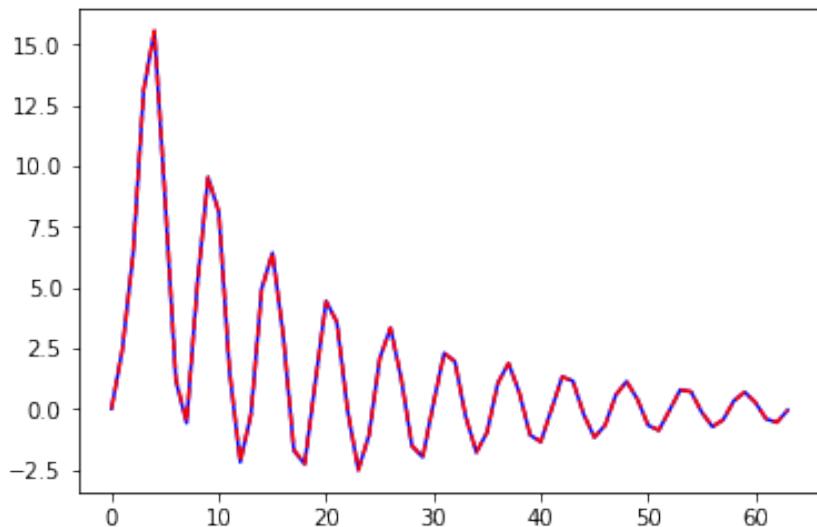
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -3.44259275e-01  3.03222355e-02 -3.00861951e-01
      -7.45430439e-02 -2.06924541e-01 -2.82573262e-01  1.50637699e-01
      1.00854437e-01  3.30084690e-01 -1.94517412e-01 -2.61649197e-02
      5.95453435e-03  7.51270984e-04]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 9.86284713e-05  2.39996680e+00  5.46984912e+00  1.10254430e+01
      1.04974787e+01  1.51224199e+00 -6.22566995e+00 -8.36221975e+00
      -2.49316857e+00  1.02328420e+00  2.29459915e+00  1.36324167e-01
      -2.70969996e-02  4.17574675e-03]

```



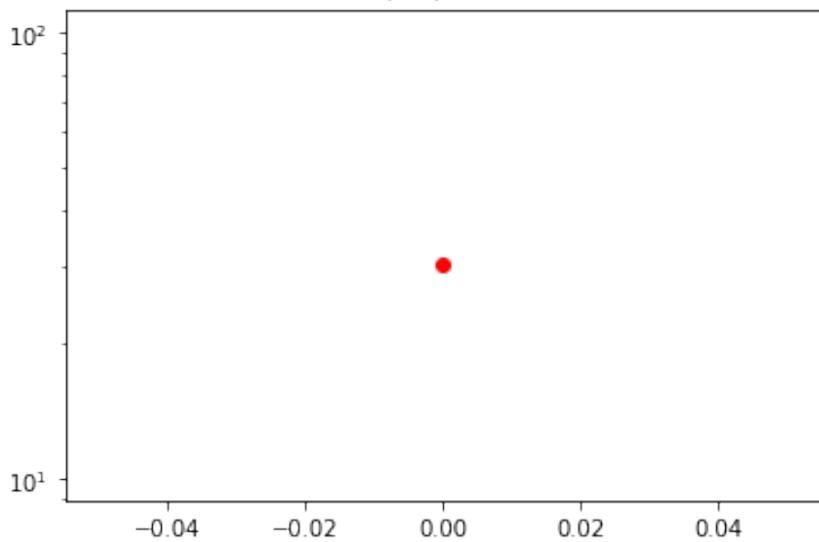
```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.81547666  0.13075051  0.64060723 -0.73320308  0.00826049]
```

```
0.00835999  0.00276439]  
b = [0.  2.4 4.4 8.4 7.4]  
b_est = [9.86284713e-05 2.39992032e+00 4.33893695e+00 8.29974681e+00  
7.20678721e+00]
```

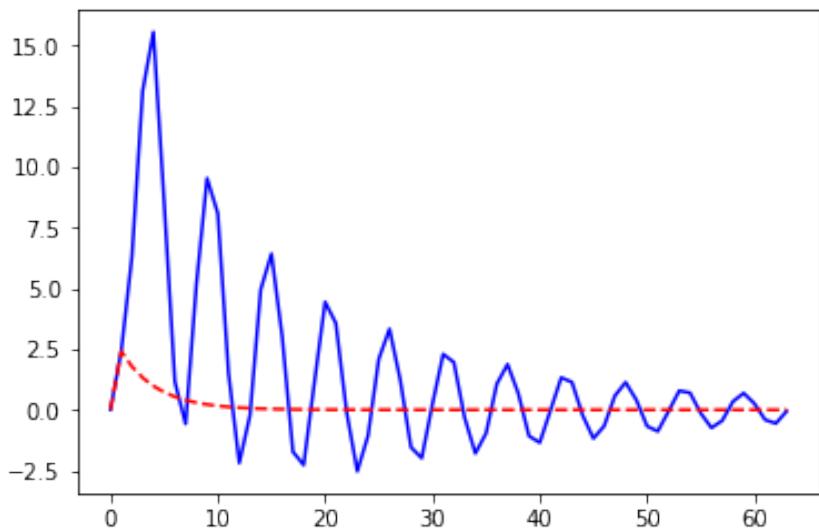


```
std = 0.01 :  
### na_est = 1 :
```

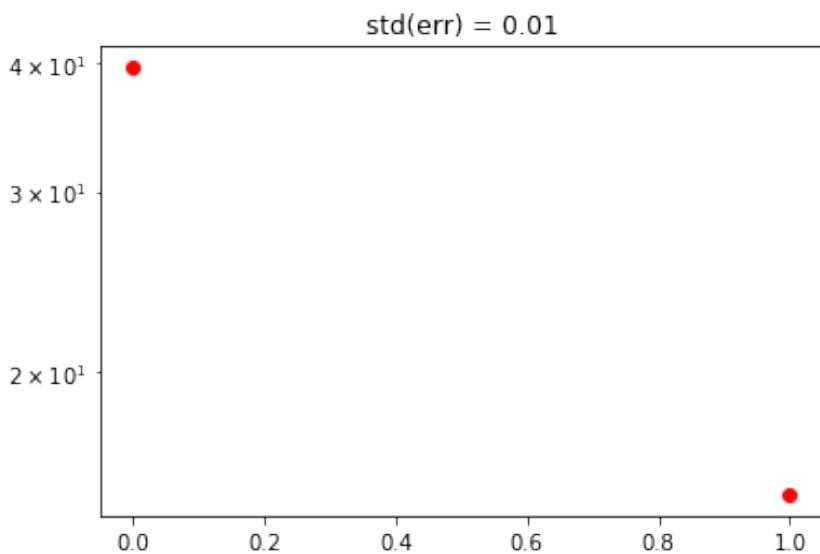
$\text{std}(\text{err}) = 0.01$



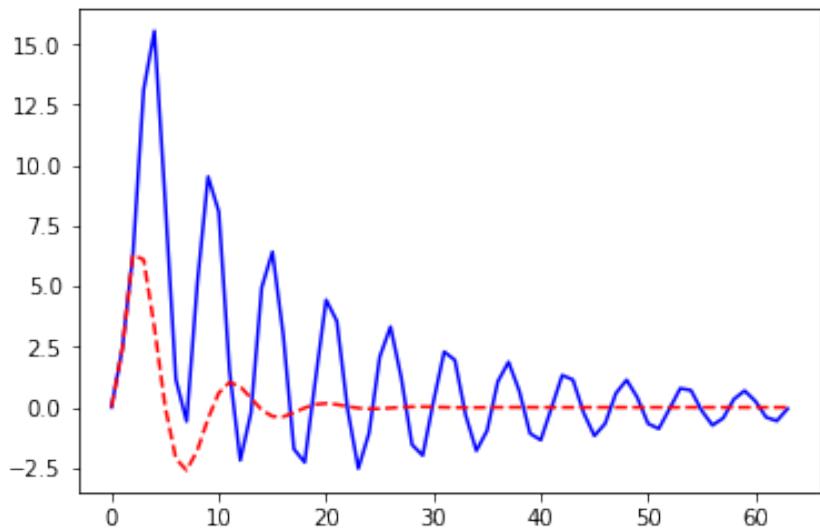
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.74124613]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.0071883  2.39866642]
```



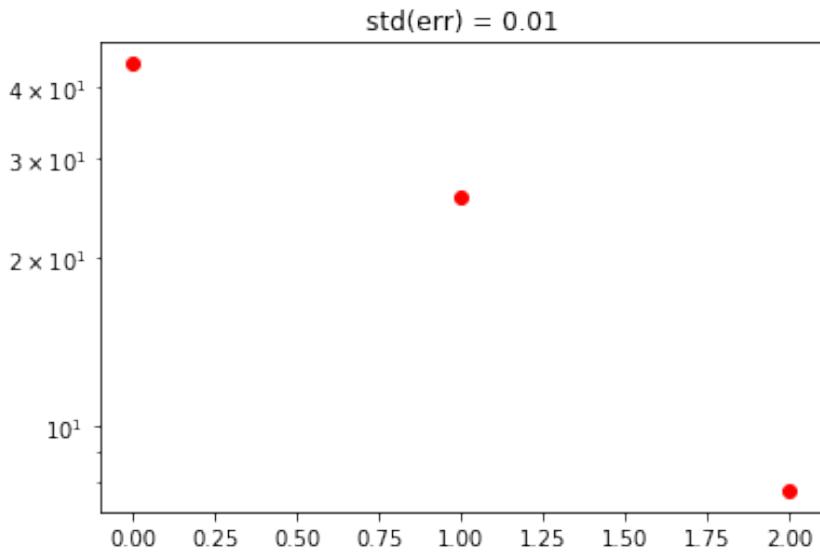
```
### na_est = 2 :
```



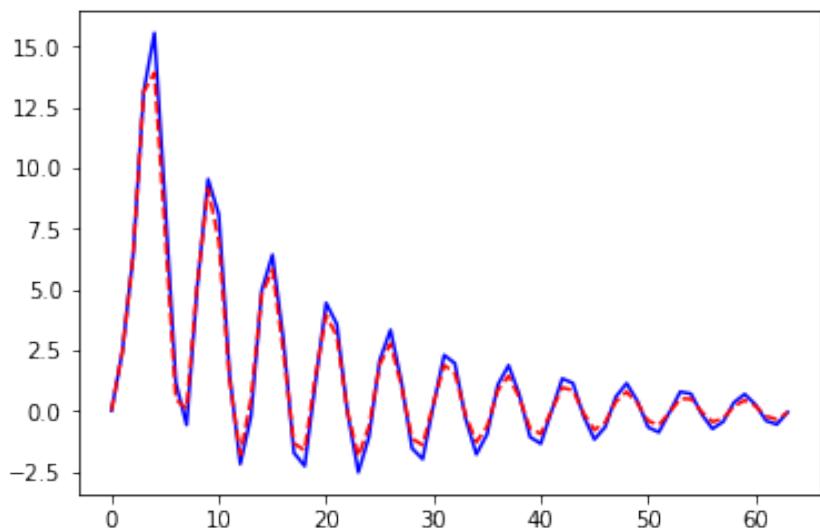
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21662819  0.65348587]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.0071883  2.39524923  3.38529462]
```



```
### na_est = 3 :
```

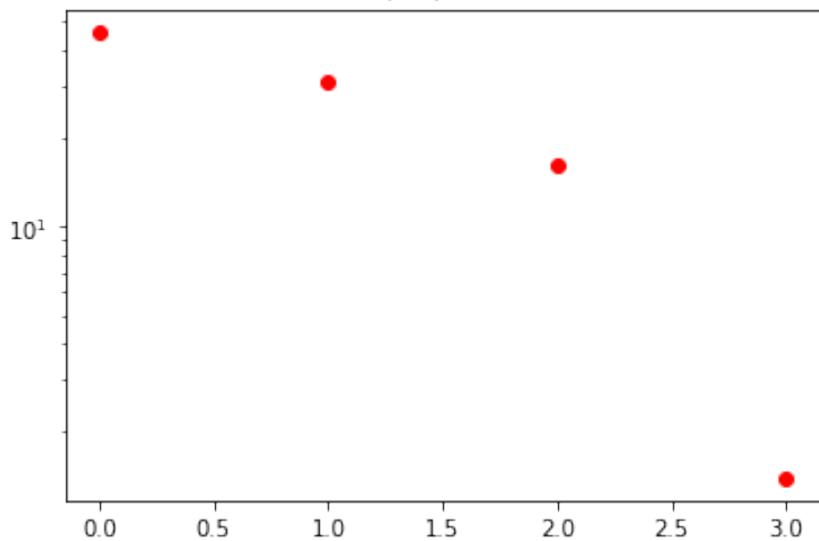


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.68807144  1.61405653 -0.80149556]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.0071883  2.39186035 2.25885243 6.3592147 ]
```

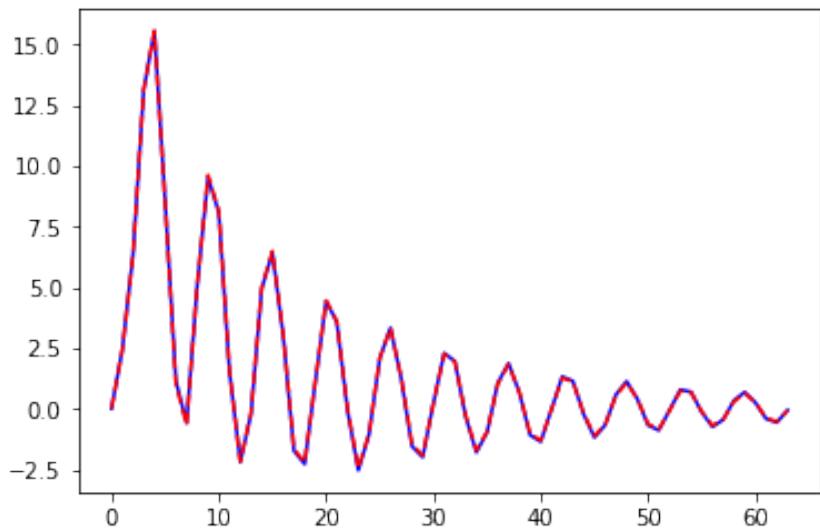


```
### na_est = 4 :
```

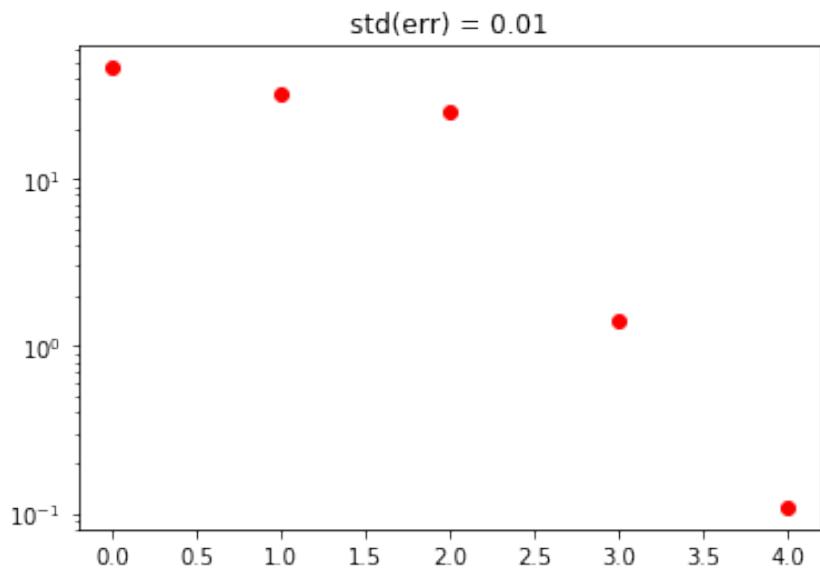
$\text{std}(\text{err}) = 0.01$



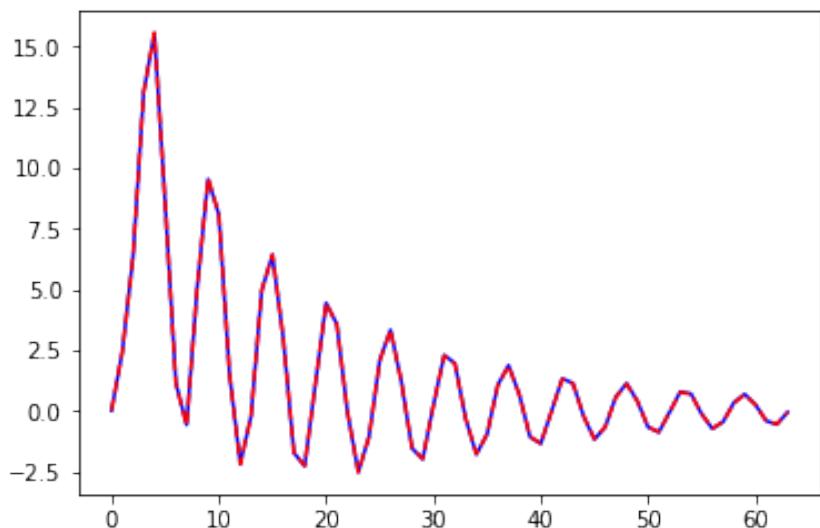
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.79580588  0.12591017  0.63152995      -0.71735568]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [7.18830028e-03 2.39827422e+00 4.39315686e+00 8.41807965e+00
    7.41090698e+00]
```



```
### na_est = 5 :
```

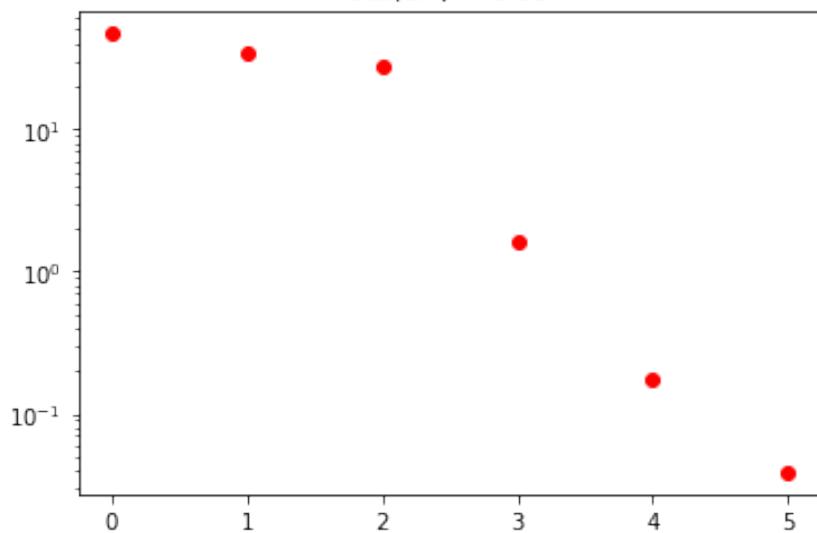


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.33280311  0.54094827  0.5887229   -1.08236098  0.39948849]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.0071883   2.39441413  3.10520178  6.02955785  2.87226691 -3.96560904]
```

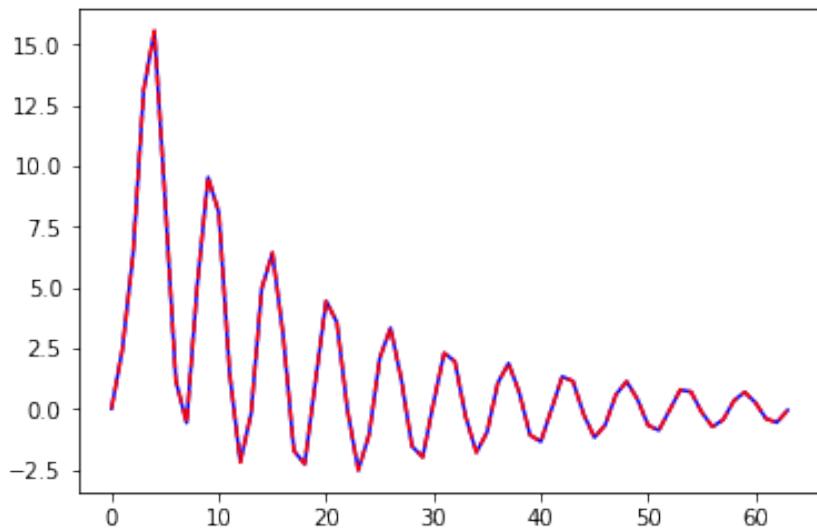


```
### na_est = 6 :
```

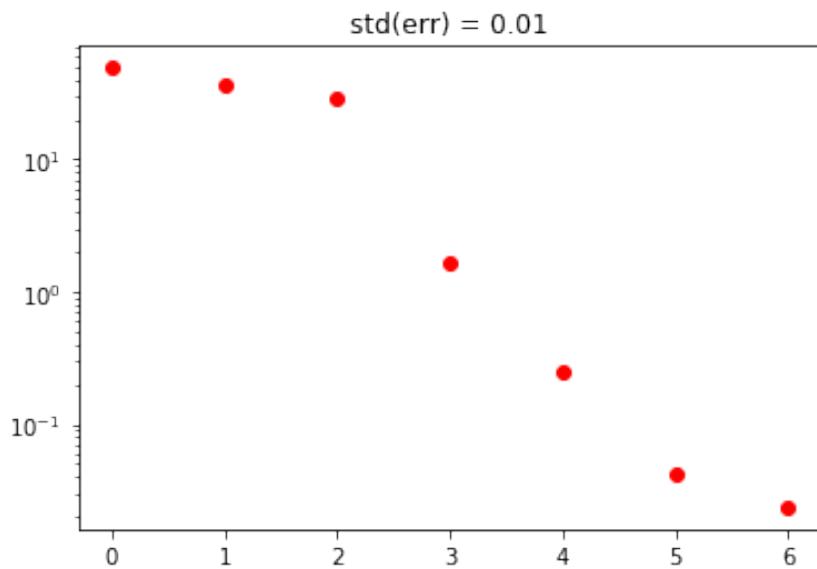
$\text{std}(\text{err}) = 0.01$



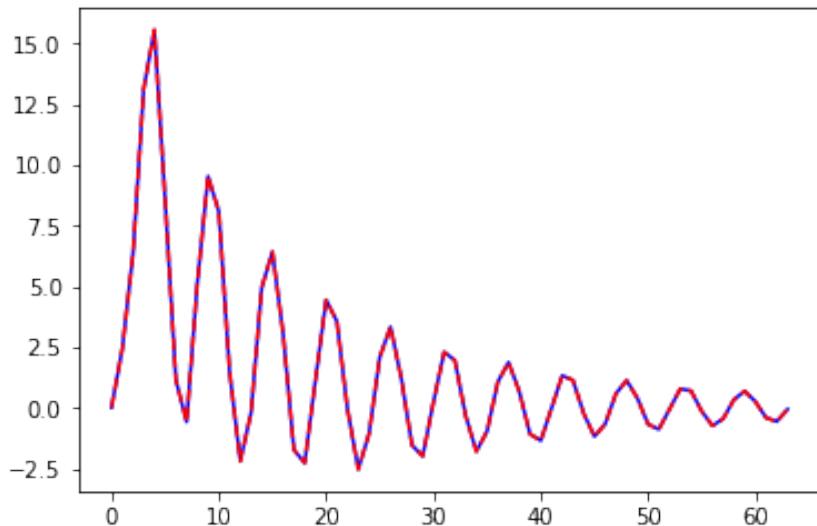
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.93073881  0.2238832   0.6268425    -0.80721371  0.09100802
     0.01007053]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.39730429e+00  4.06948308e+00  7.80277129e+00
     6.24525441e+00 -9.75821897e-01  4.18346378e-02]
```



```
### na_est = 7 :
```

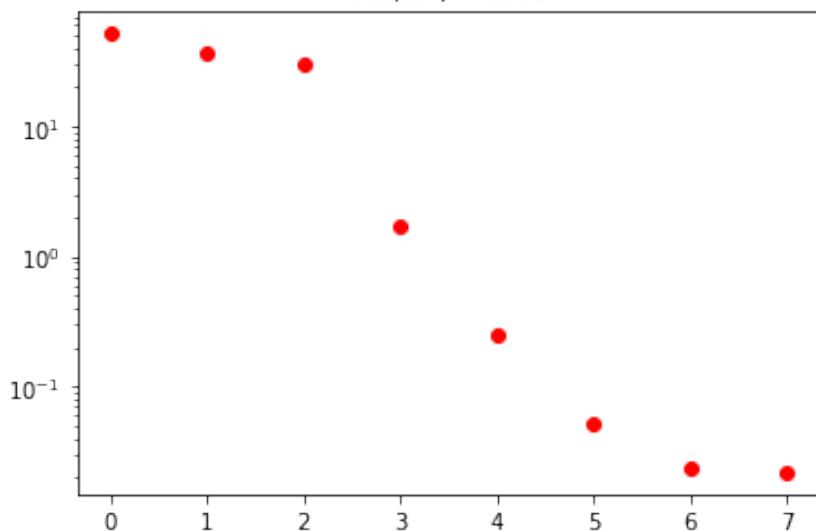


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.39718273 -0.16911766  0.62388791 -0.43007646 -0.27488921
      -0.04553047  0.0365879 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40113965e+00  5.34932407e+00  1.02222438e+01
      1.07677459e+01  3.04775094e+00 -5.05650589e-02 -2.70308659e-01]
```

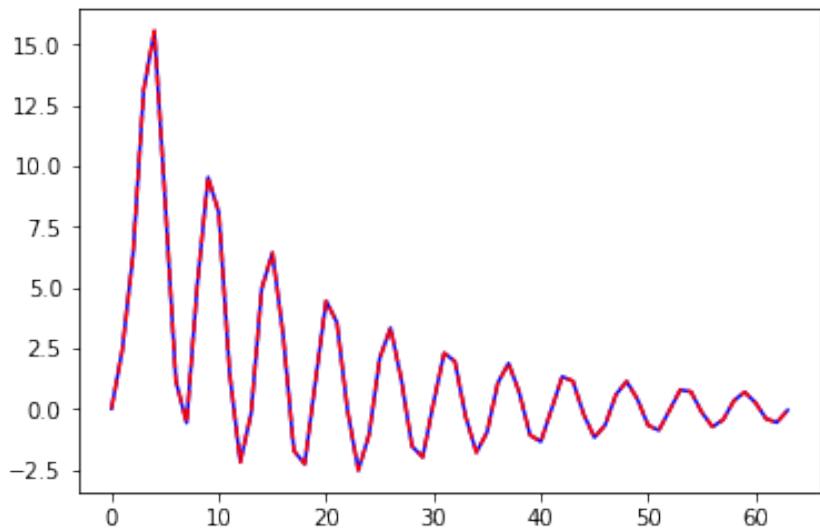


```
### na_est = 8 :
```

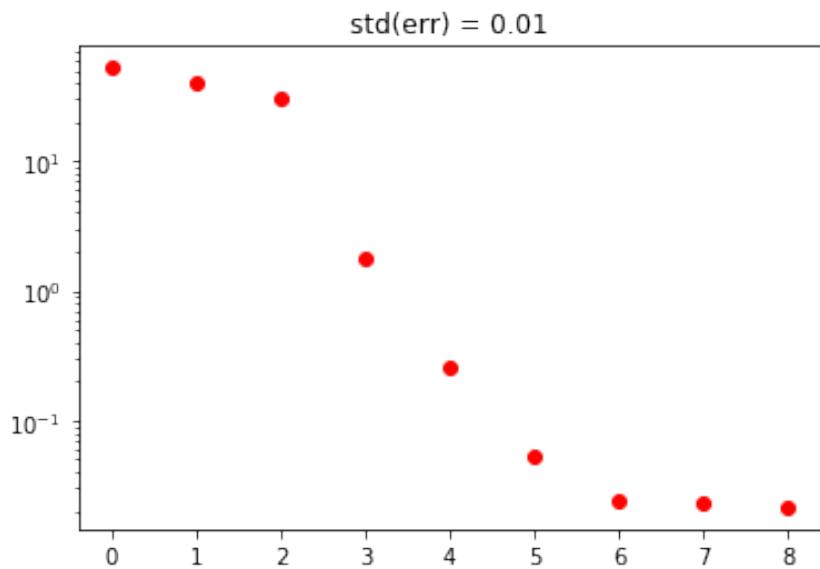
$\text{std}(\text{err}) = 0.01$



```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.17012102  0.04001716  0.21948321 -0.03780454 -0.28588756
      -0.39193753  0.18978764 -0.07133251]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40277183e+00  5.89668255e+00  1.21538028e+01
      1.40980752e+01  7.71767130e+00  2.27774825e+00 -2.60496195e-01
      7.96898010e-01]
```



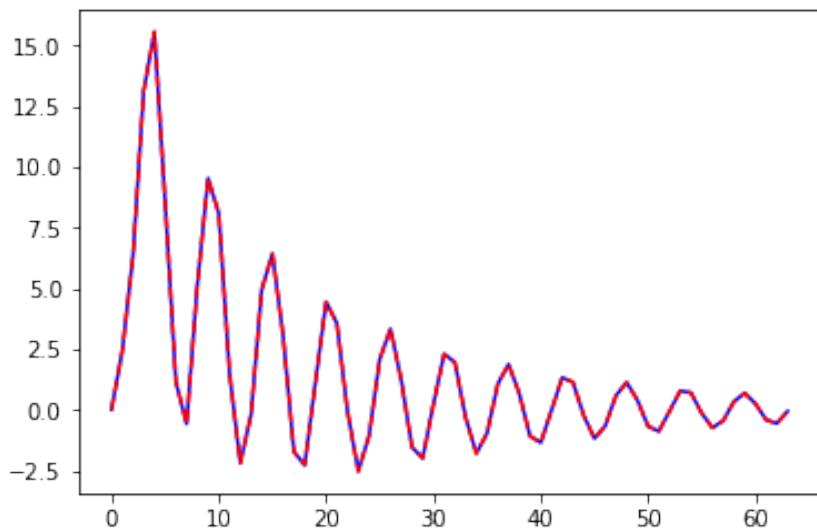
```
### na_est = 9 :
```



```

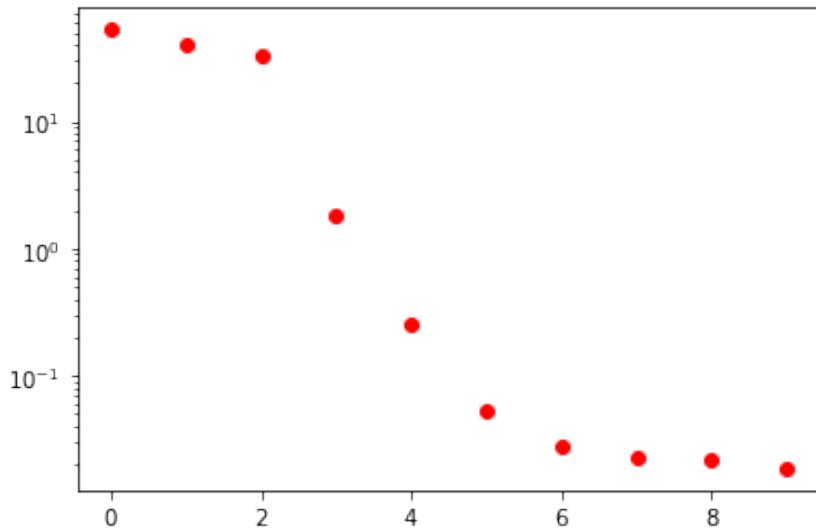
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.27570239  0.03660506  0.18288884  0.01288446 -0.36557392
      -0.34619761  0.28664326 -0.14045752  0.03226795]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40201288e+00  5.64284095e+00  1.14796080e+01
      1.26028062e+01  5.92143104e+00  9.70218027e-01 -7.07176581e-01
      8.04230848e-01 -3.43508387e-01]

```

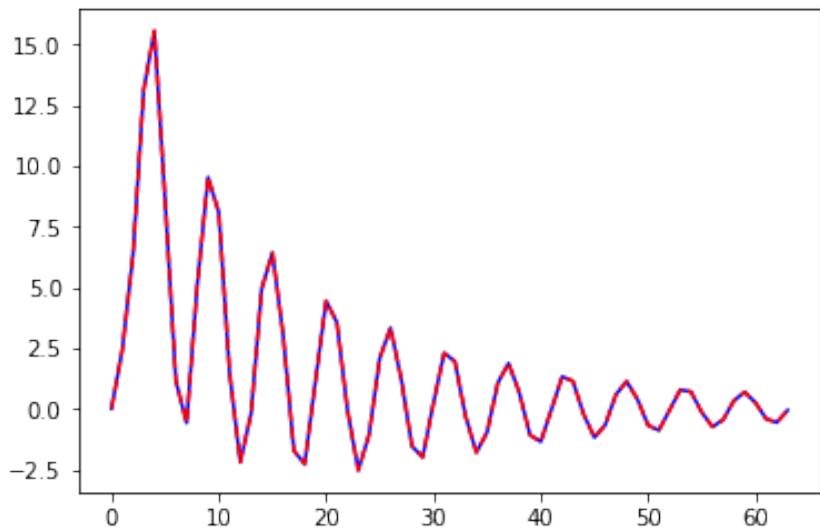


```
### na_est = 10 :
```

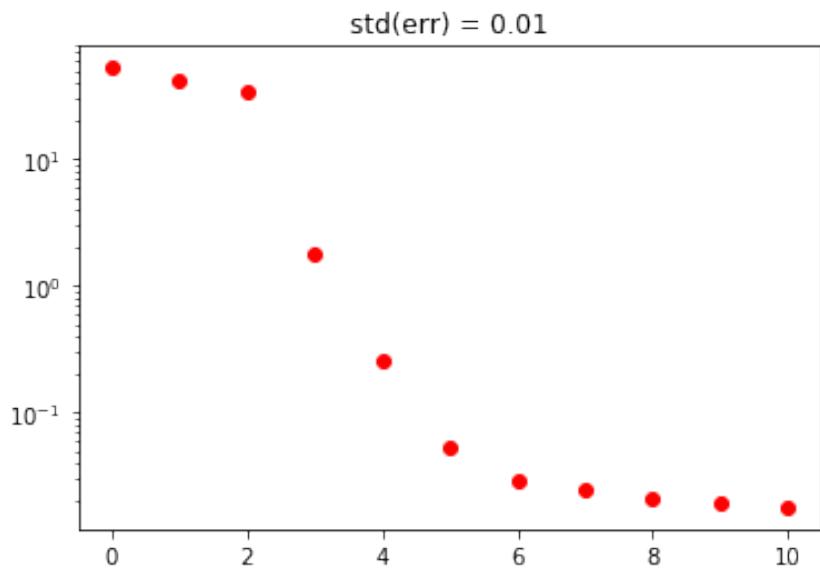
$\text{std}(\text{err}) = 0.01$



```
a = [ 1.           -0.79           0.1057          0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.           -0.28787367  0.04016413  0.10481893 -0.0170035   -0.33649298
     -0.33982344  0.25707363 -0.0967393   0.0817446   -0.02811059]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40192539e+00  5.61360685e+00  1.14108584e+01
     1.22775599e+01  5.21497031e+00 -2.21914543e-01 -2.09887686e+00
     3.45357924e-02 -2.98821551e-01  2.66184027e-01]
```



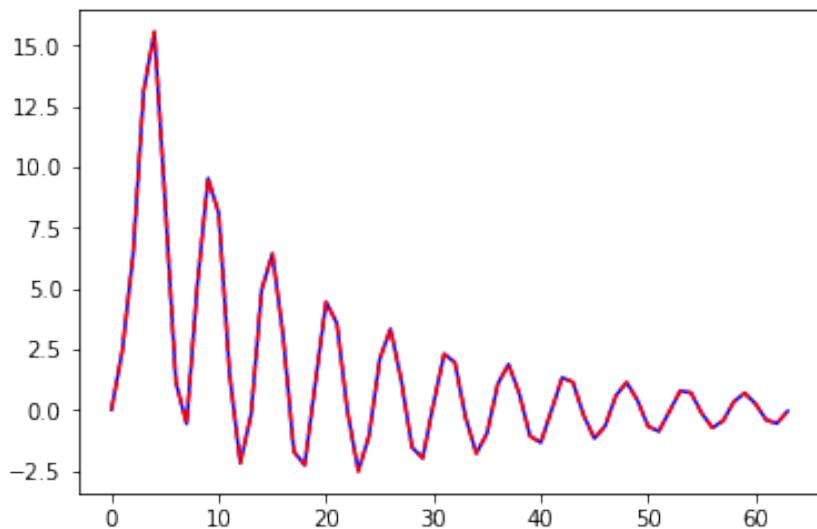
```
### na_est = 11 :
```



```

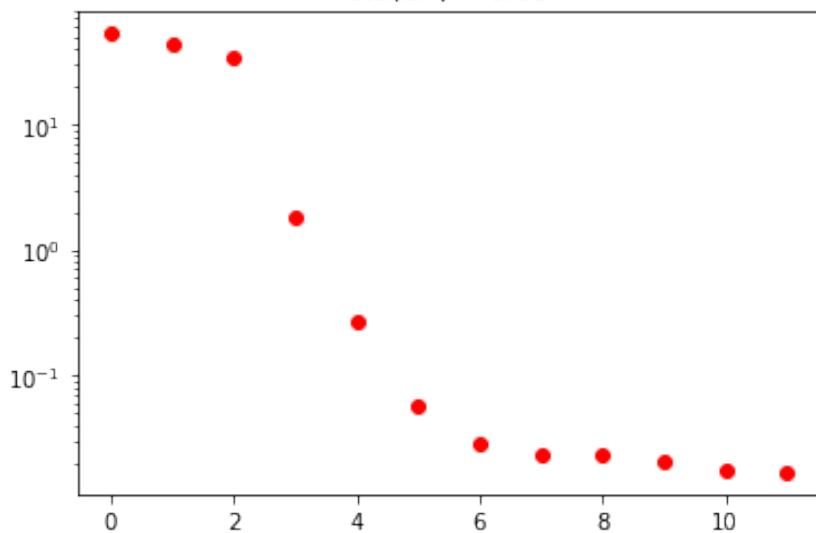
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.21725299  0.00790799  0.08905967 -0.15943743 -0.40539916
      -0.28571774  0.21168061 -0.12690636  0.18997027  0.04685652 -0.04421746]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40243304e+00  5.78314671e+00  1.17784907e+01
      1.29624217e+01  5.44749803e+00 -1.39021937e+00 -4.71270018e+00
      -3.06631982e+00 -1.87249168e+00  3.45228858e-01  4.06859939e-01]

```

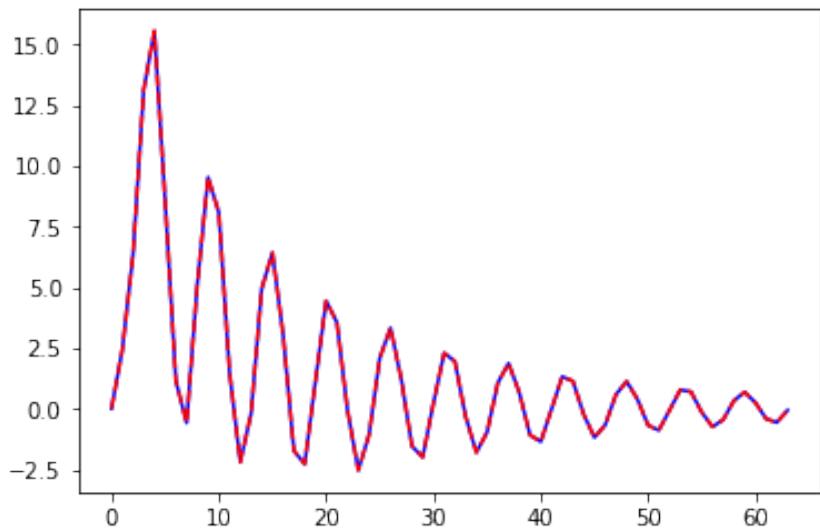


```
### na_est = 12 :
```

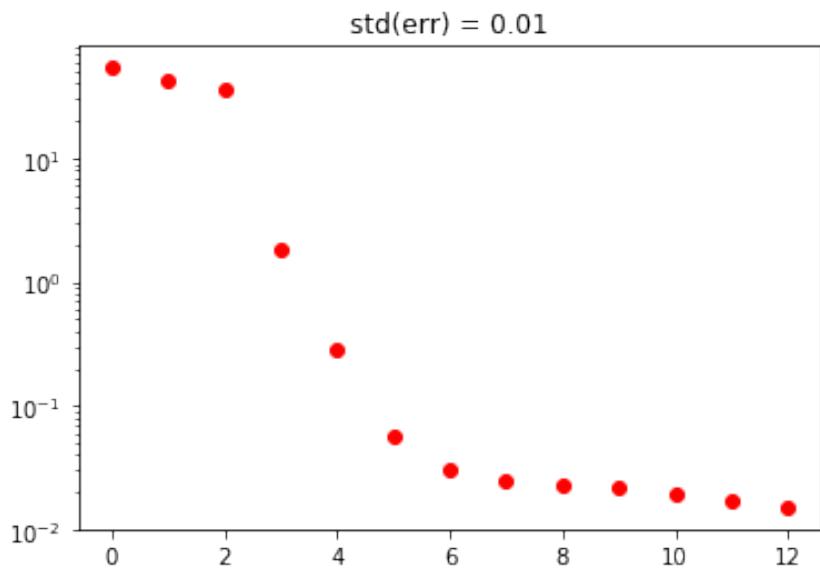
$\text{std}(\text{err}) = 0.01$



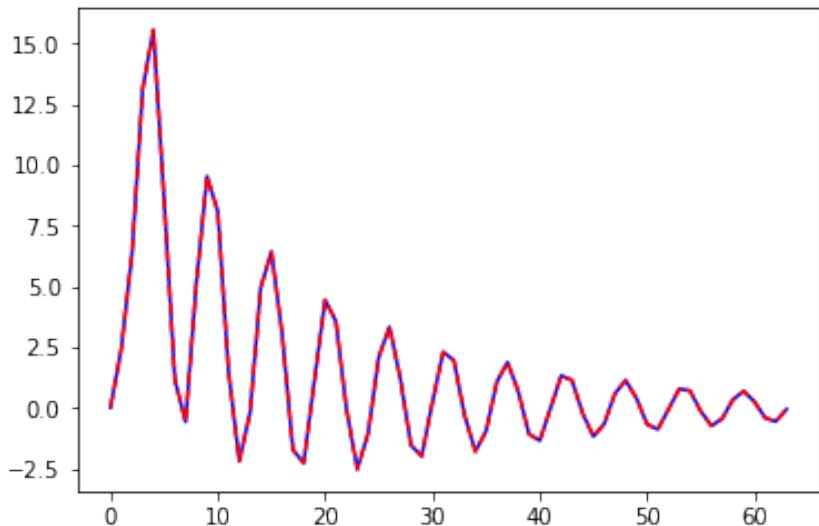
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.25320053 -0.00211528  0.10678935 -0.17520398 -0.37382373
      -0.25499442  0.21666155 -0.14017179  0.20583489  0.03158294 -0.07214602
      0.01546825]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40217463e+00  5.69665698e+00  1.15278599e+01
      1.24697858e+01  4.83097113e+00 -1.64386190e+00 -4.49821338e+00
      -2.53060188e+00 -1.27141259e+00  6.90090665e-01  3.86389694e-01
      -1.48679050e-01]
```

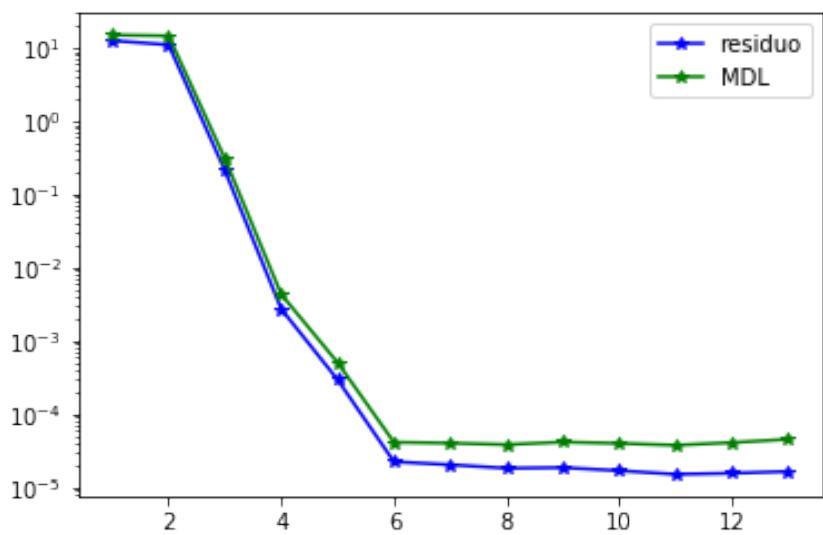


```
### na_est = 13 :
```

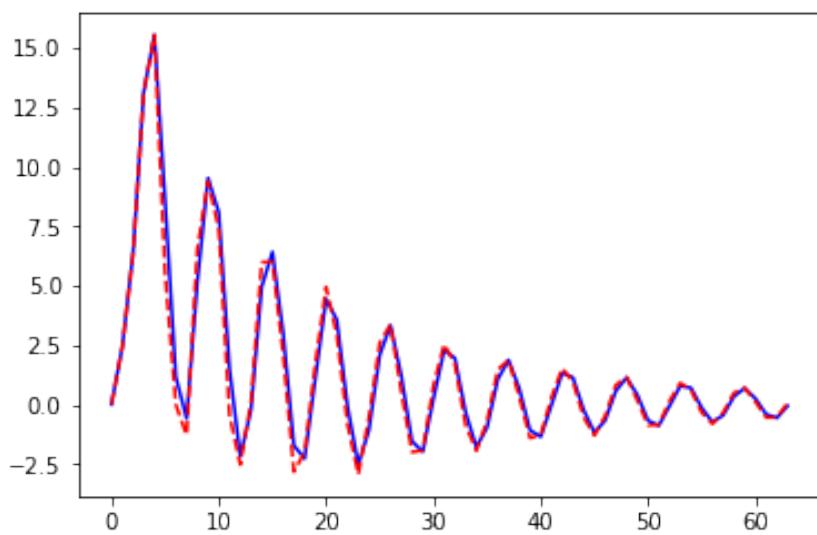


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.24069781 -0.05372112  0.09878982 -0.14798723 -0.40909693
      -0.2173099   0.2518783  -0.12968473  0.17843706  0.05880812 -0.08294254
      -0.02714605  0.0227869 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.18830028e-03  2.40226451e+00  5.72634248e+00  1.14825764e+01
      1.22895013e+01  4.36262934e+00 -2.35750525e+00 -4.82348221e+00
      -2.38336221e+00 -7.60712752e-01  1.32786972e+00  7.98021578e-01
      -1.69674935e-01 -2.22003851e-01]
```

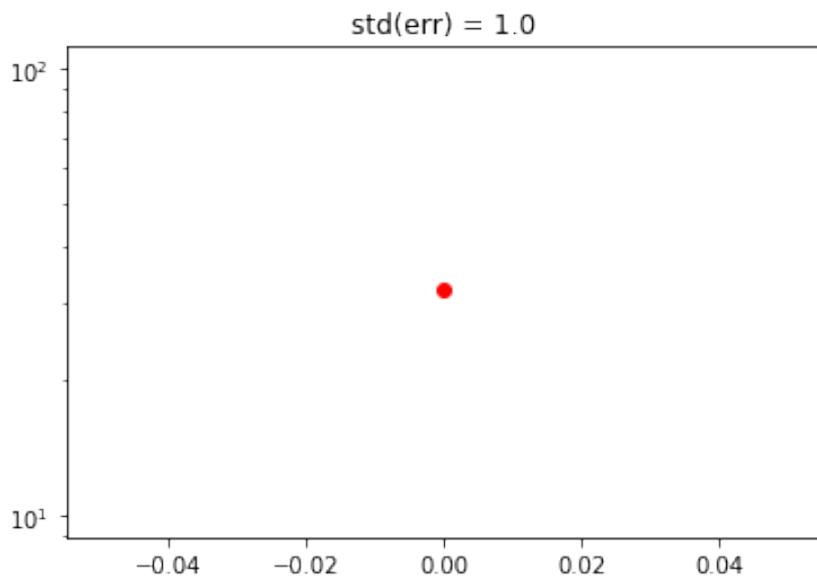




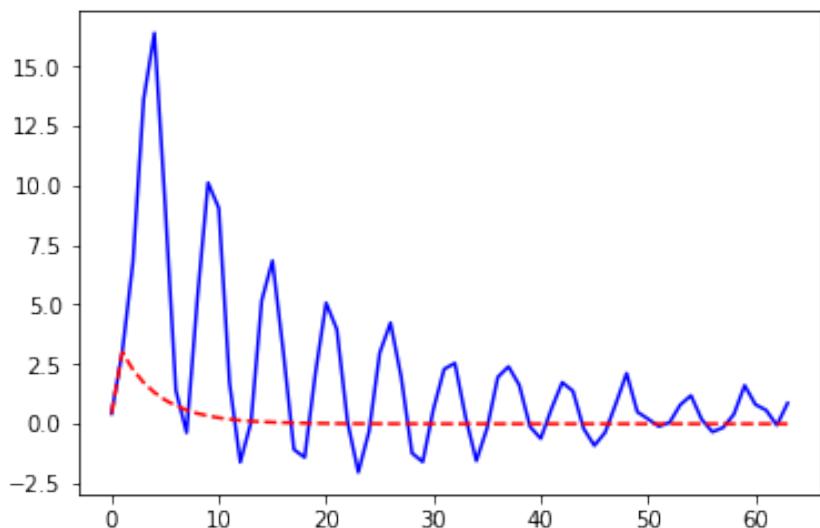
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.39718273 -0.16911766  0.62388791 -0.43007646 -0.27488921
      -0.04553047  0.0365879 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [7.18830028e-03 2.40113965e+00 5.34932407e+00 1.02222438e+01
      1.07677459e+01]
```



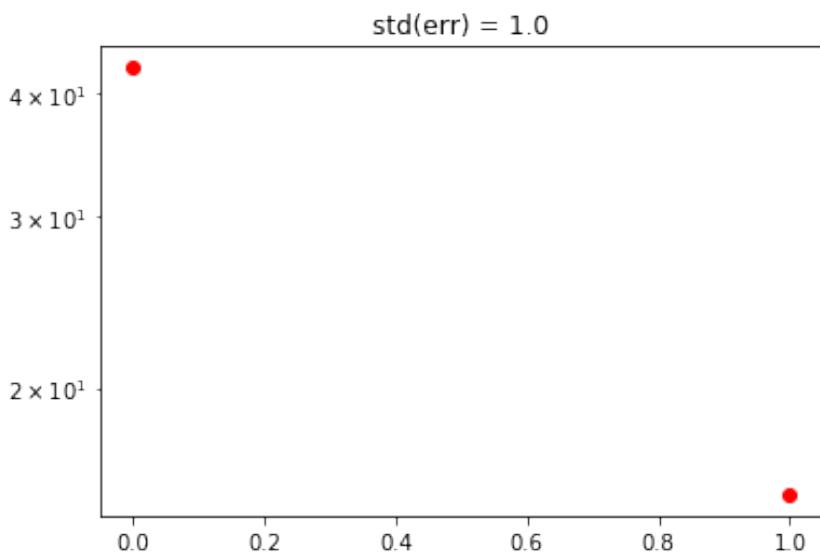
```
std = 1.0 :  
### na_est = 1 :
```



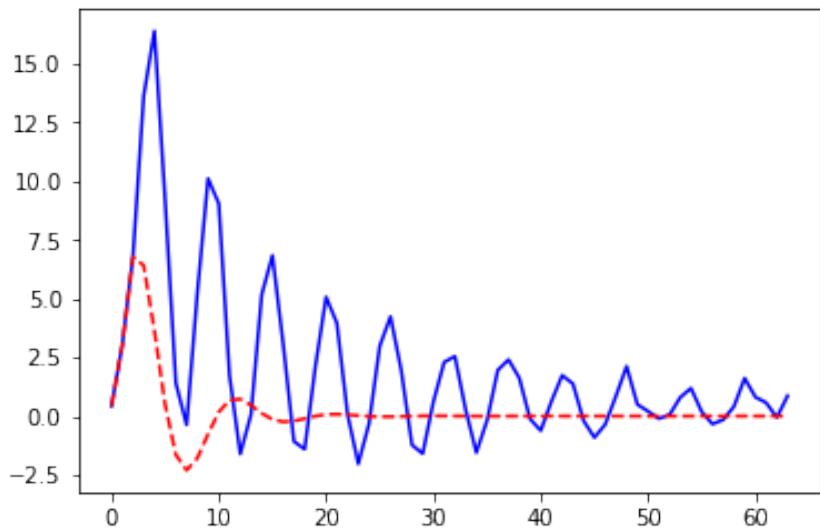
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.75956406]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.42317093 2.6728177 ]
```



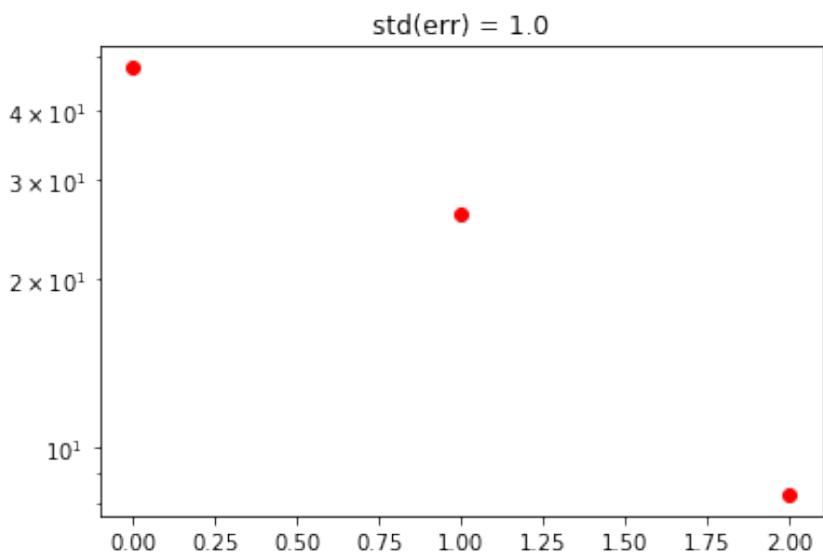
```
### na_est = 2 :
```



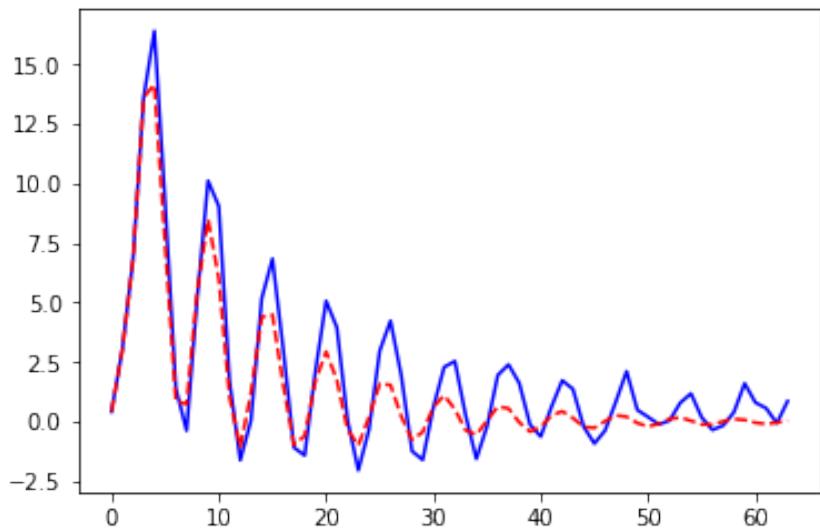
```
a = [ 1.           -0.79           0.1057         0.647529      -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -1.21799515  0.61550273]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.42317093 2.47882299 3.38887334]
```



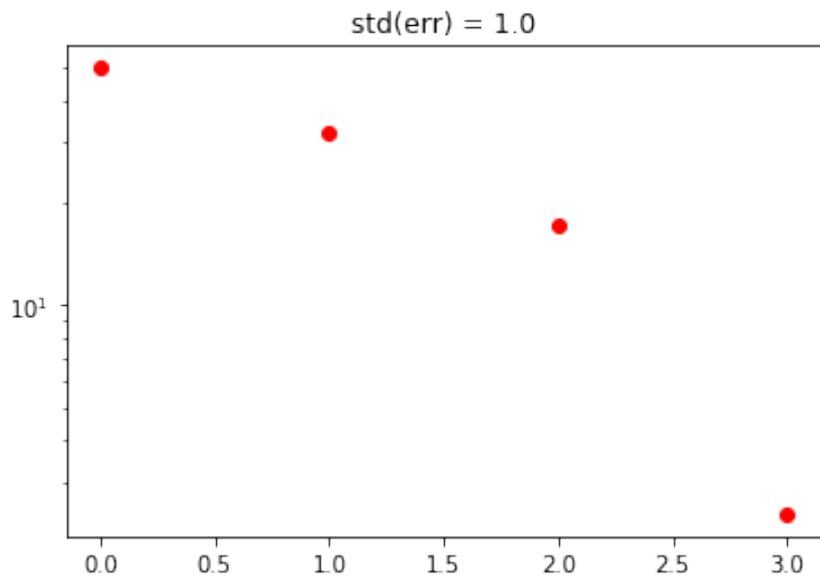
```
### na_est = 3 :
```



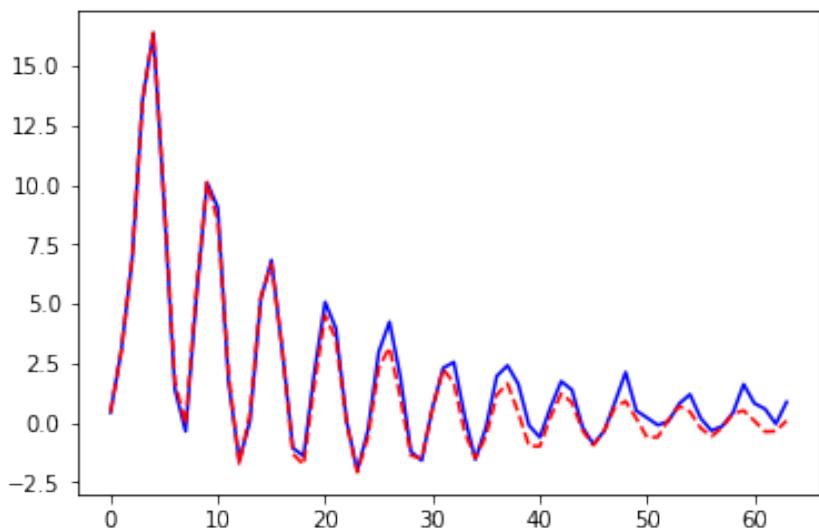
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -1.62971732  1.52138867 -0.75345723]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.42317093 2.30459414 2.53942166 6.81203863]
```



```
### na_est = 4 :
```

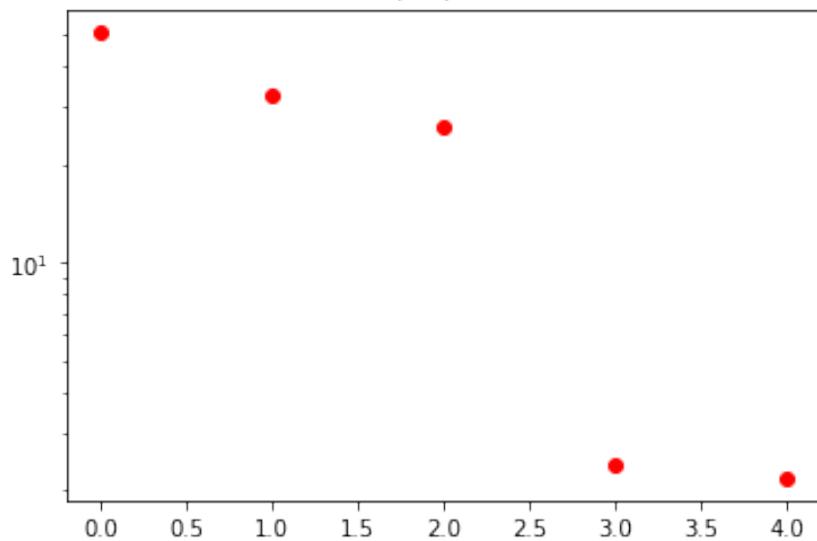


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.86690786  0.2531258   0.49061928 -0.65002381]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.42317093 2.62739293 4.28676666 8.70933529 7.49425888]
```

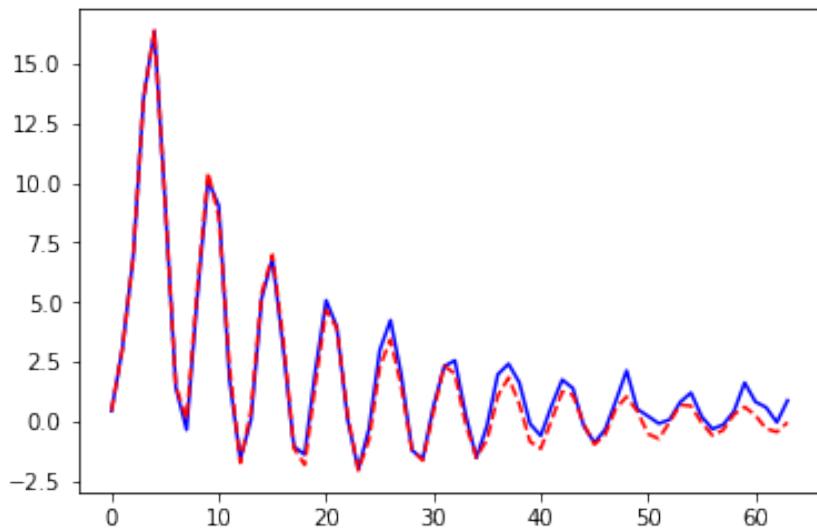


```
### na_est = 5 :
```

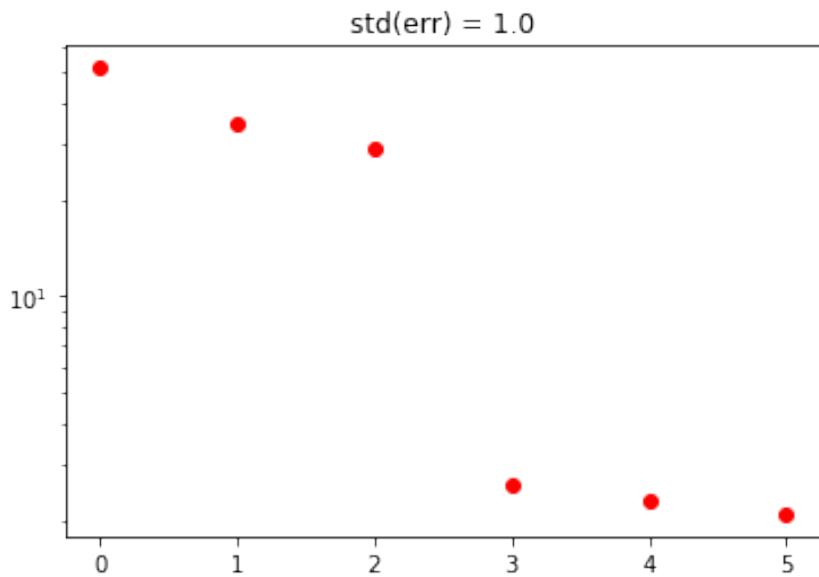
$\text{std}(\text{err}) = 1.0$



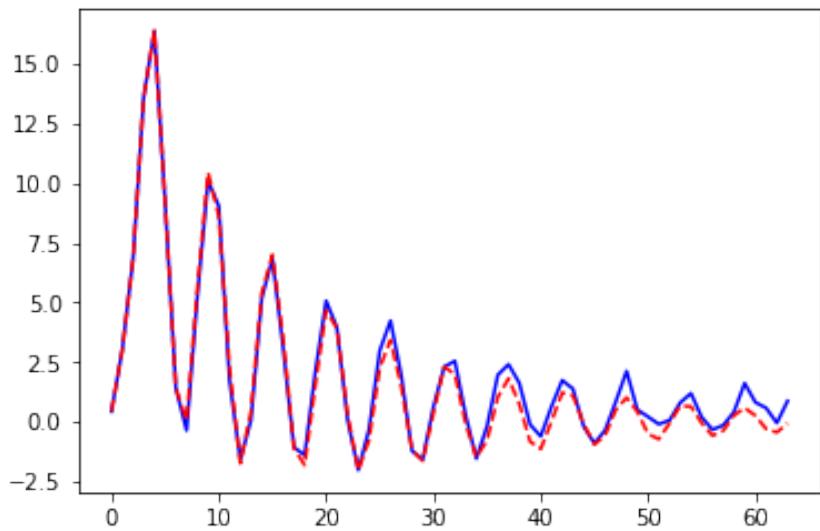
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.49541052  0.06579154  0.33816785 -0.21464039 -0.38030119]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.7845998   5.3198456   10.6009351  11.01160234  3.43787267]
```



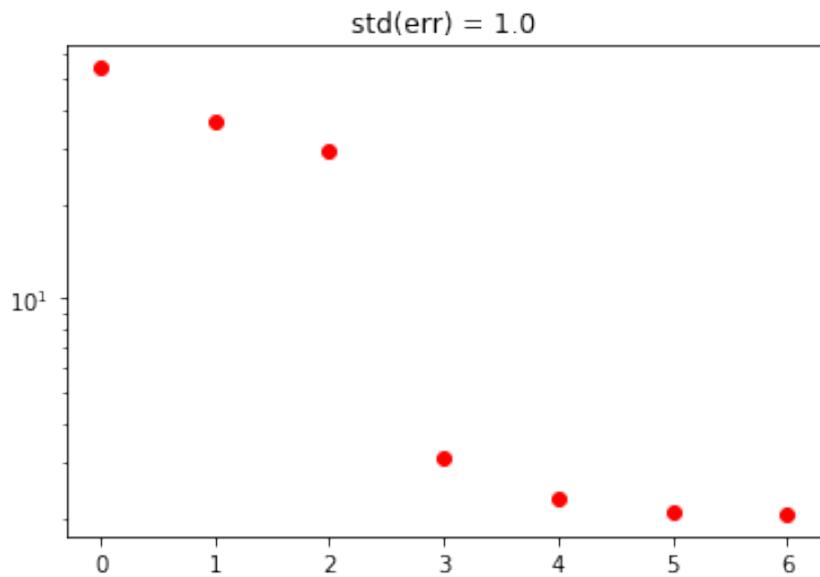
```
### na_est = 6 :
```



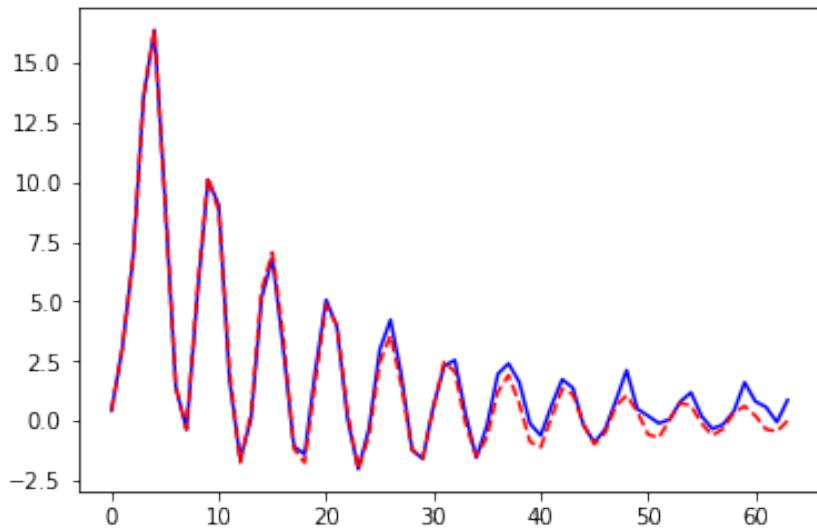
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.46916462  0.07564219  0.33177373 -0.22272596 -0.35074788
      -0.03824362]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.7957063   5.40260073 10.80555064 11.41317906  3.94716578
      0.27659727]
```



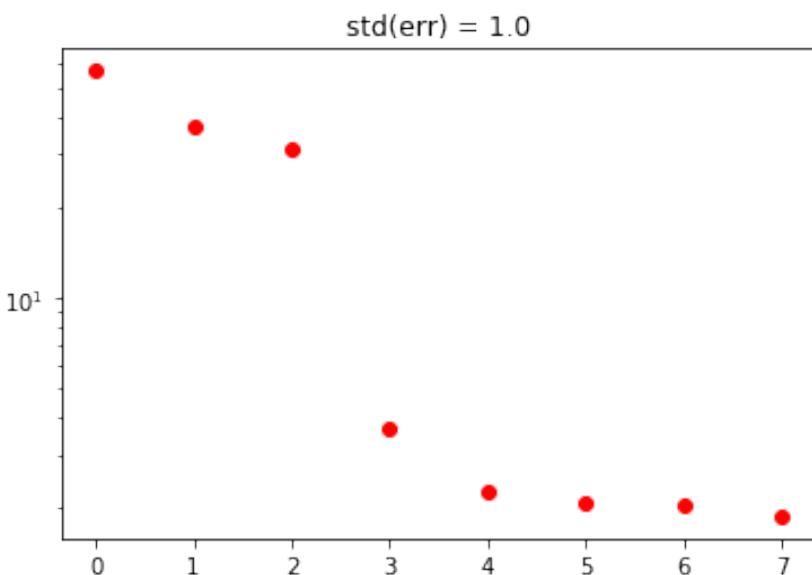
```
### na_est = 7 :
```



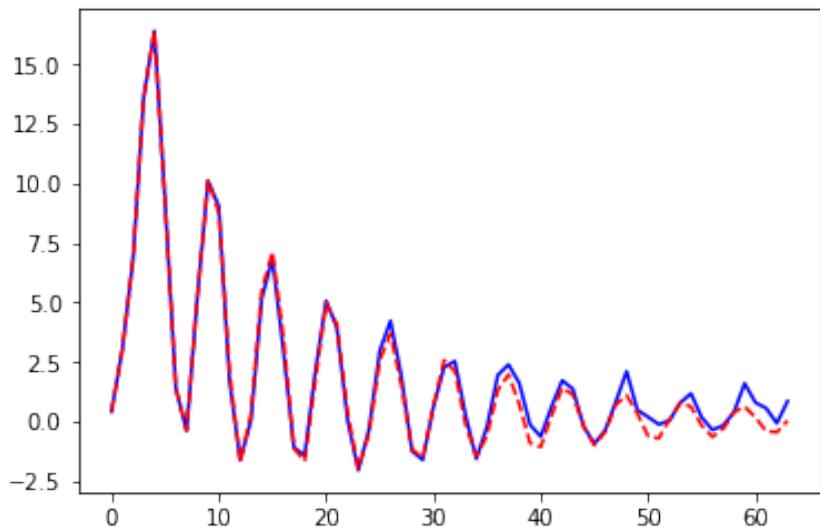
```
a = [ 1.           -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.4988663  -0.13972488  0.11301121 -0.05508023 -0.35535342
      -0.28541534  0.37666573]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.78313742  5.22252959  9.86687503  8.9654384  -0.45457825
      -5.48733851 -4.35144534]
```



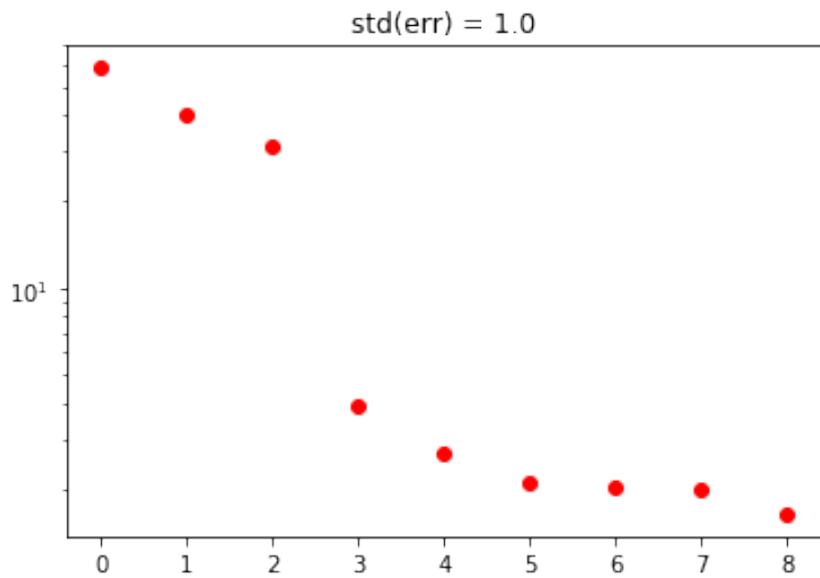
```
### na_est = 8 :
```



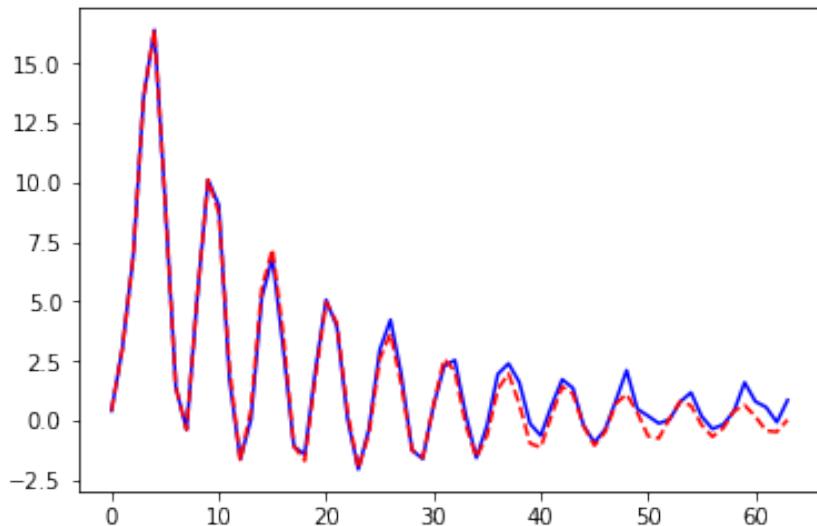
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008   -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.47852298 -0.14700195  0.13038869 -0.08277692  -0.32612603
     -0.32087392  0.39068198 -0.00950183]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.79174611  5.28036297  9.9902731   9.23346907  -0.17306306
     -5.29848668 -4.38430384 -0.40280155]
```



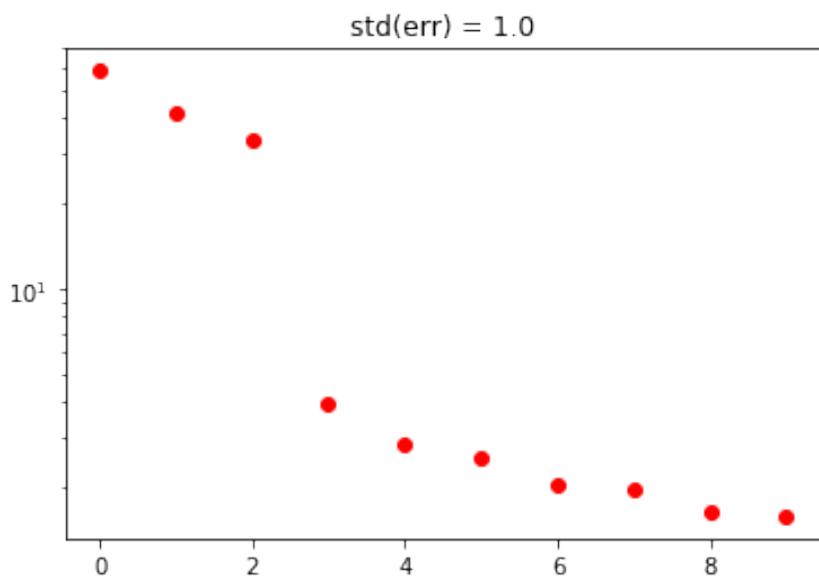
```
### na_est = 9 :
```



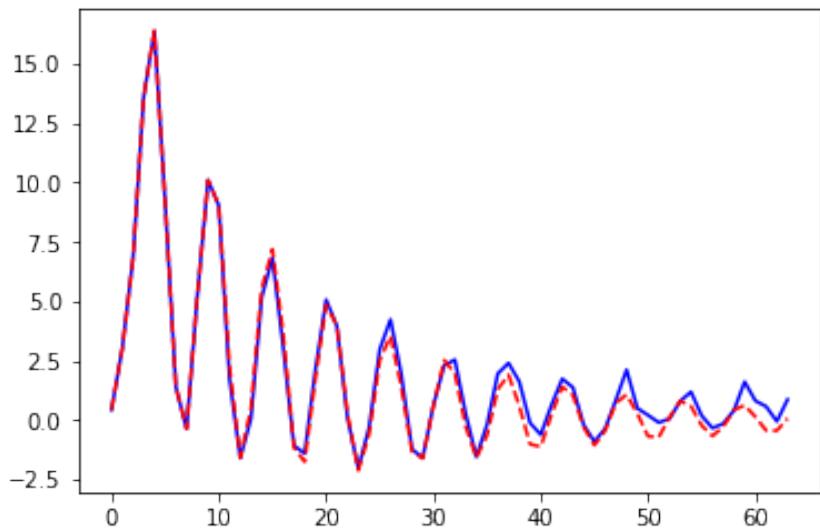
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.46484233 -0.24763363  0.19450577  0.00674606 -0.27109582
      -0.36602392  0.40873497  0.11344127 -0.17151207]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.79753537  5.27874177  9.80878155  8.96781183 -0.59341917
      -5.19703253 -2.77312197  2.05645887  1.70760052]
```



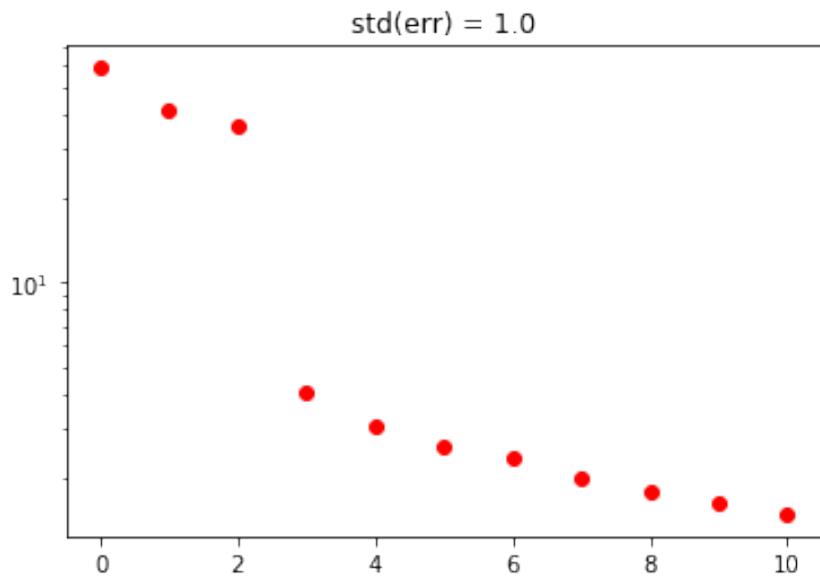
```
### na_est = 10 :
```



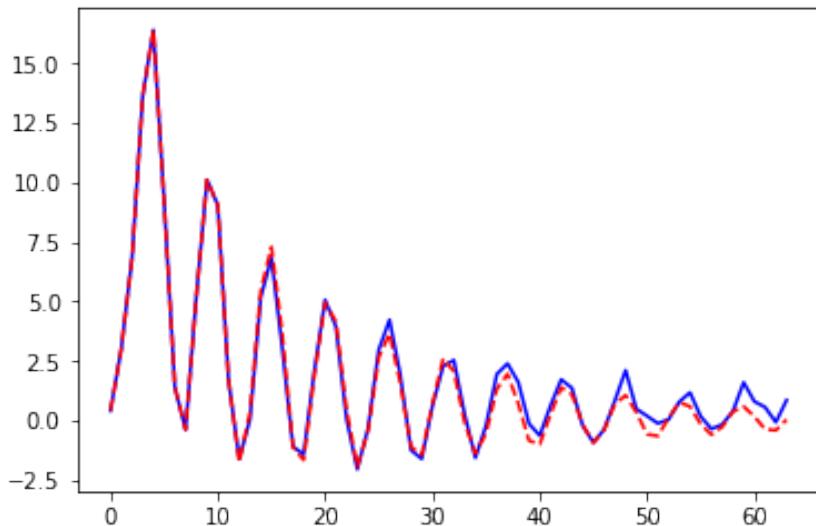
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.44972411 -0.2702525   0.12761704  0.05432888 -0.25411984
   -0.3070764   0.35396563  0.17612431 -0.13397955 -0.07459858]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.80393296  5.31443775  9.81518153  8.84028618 -0.95717604
   -5.94174998 -3.13951722  2.6785466   3.05157153  1.32565158]
```



```
### na_est = 11 :
```

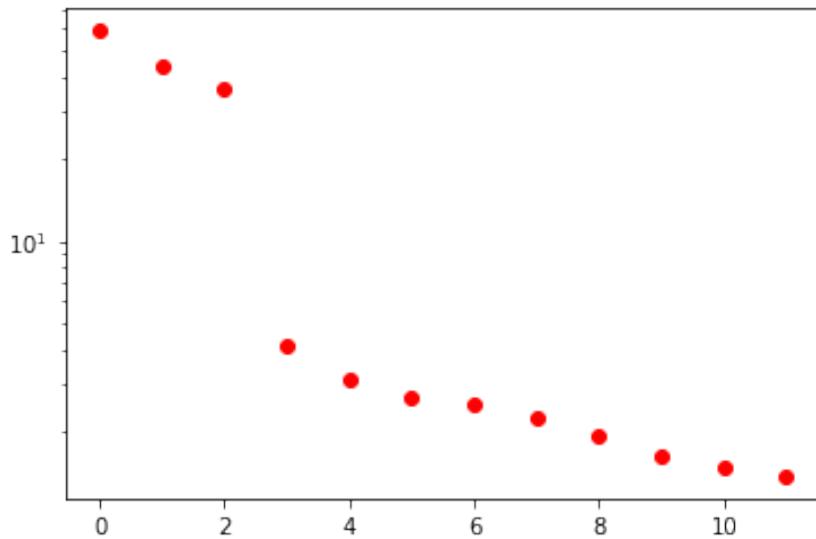


```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.51202098 -0.2561187   0.17492638  0.13378916 -0.32452421
      -0.29510854  0.2837346   0.23038837 -0.23642165 -0.0999143   0.08712829]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.77757074  5.13388679  9.45543629  8.26300563 -1.257107
      -5.30476036 -1.71041942  3.39288505  2.16767192 -0.49229384 -1.8677018 ]
```

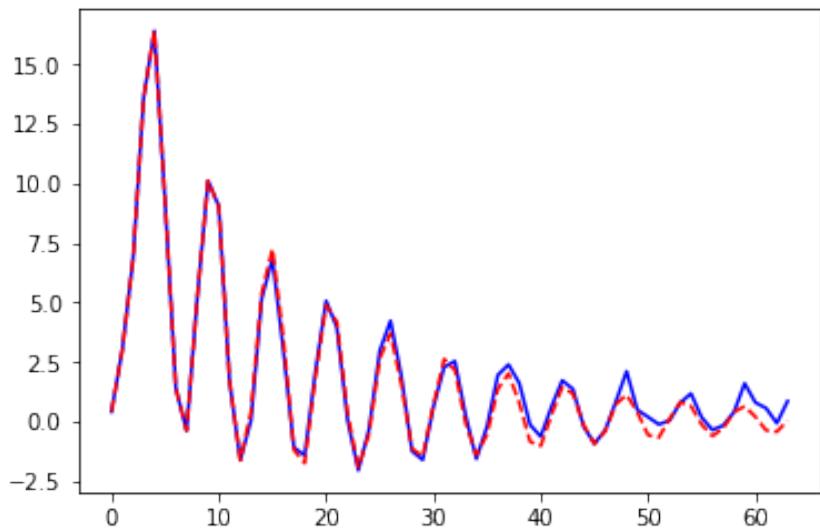


```
### na_est = 12 :
```

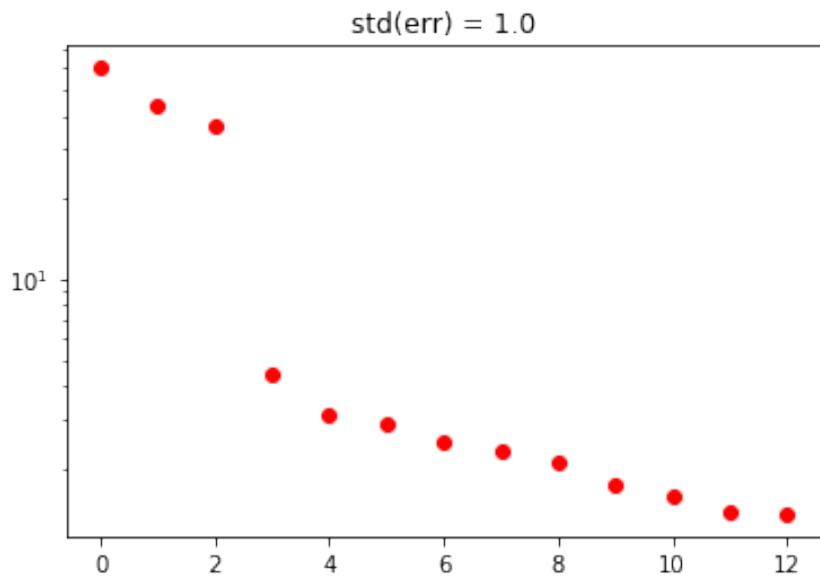
$\text{std}(\text{err}) = 1.0$



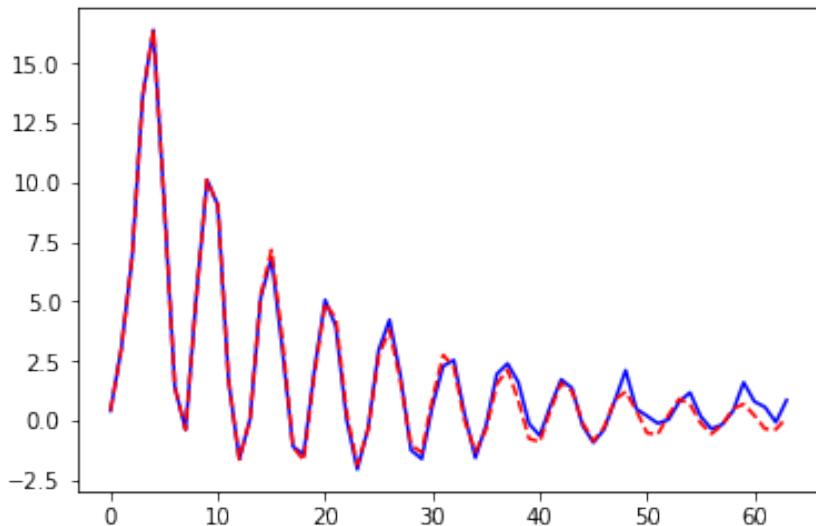
```
a = [ 1.           -0.79          0.1057        0.647529     -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.50619639 -0.27455625  0.14432085  0.12852337 -0.23007361
      -0.36391023  0.21046415  0.21077128 -0.19937262 -0.11261811 -0.02747647
      0.1536627 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.78003553  5.1435248   9.42674226  8.12353139 -1.5958747
      -5.75227475 -2.04184202  3.59016932  2.18557126 -1.81222571 -3.96232778
      -1.36475929]
```

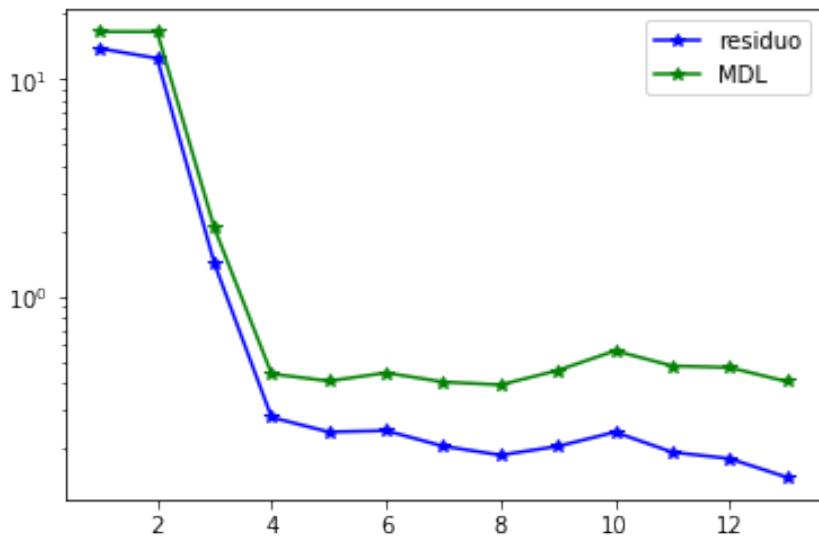


```
### na_est = 13 :
```



```
a = [ 1.          -0.79          0.1057         0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.49955871 -0.1982388   0.06314046  0.12398926 -0.18358723
      -0.29329352  0.1321162   0.19944188 -0.22680556 -0.06203692 -0.1156339
      0.1251098   0.06700671]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 0.42317093  2.7828444   5.19569494  9.66587489  8.48600704 -0.99176601
      -5.40762463 -2.23146321  3.74744797  3.18402756 -0.9423147  -4.35158852
      -2.63742894 -1.43183318]
```

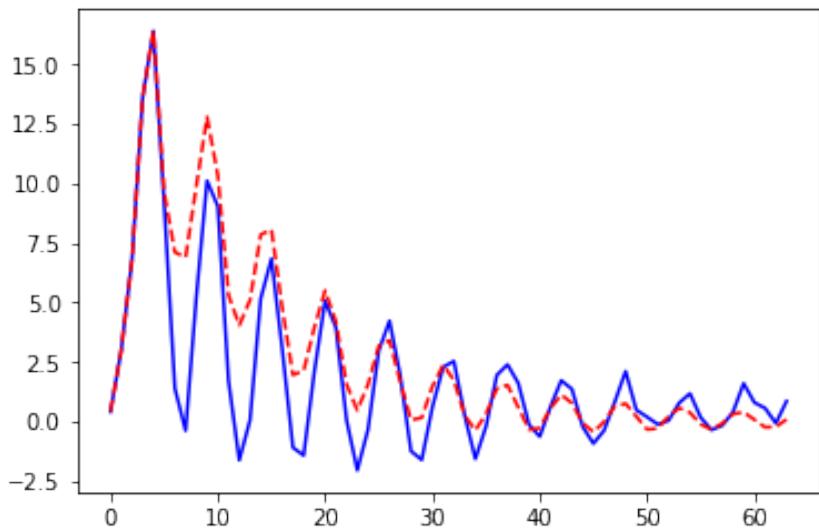




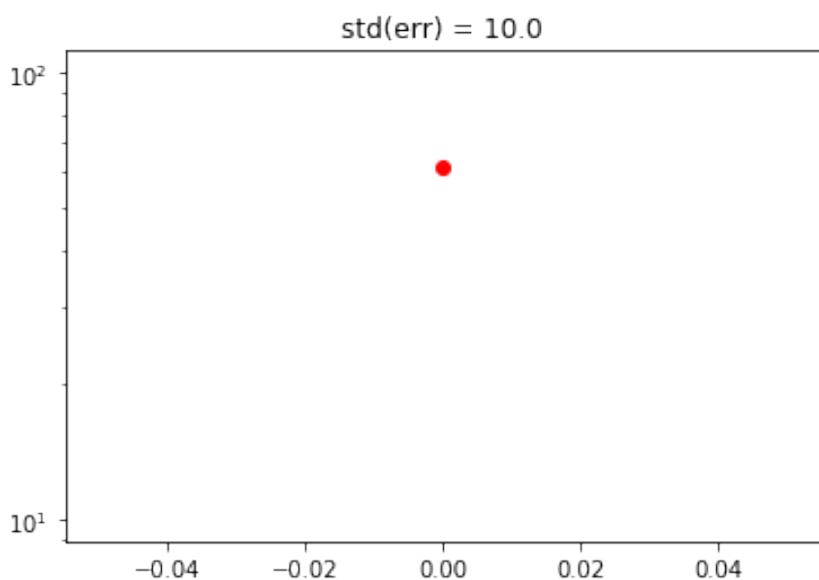
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.4988663  -0.13972488  0.11301121 -0.05508023 -0.35535342
      -0.28541534  0.37666573]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [0.42317093 2.78313742 5.22252959 9.86687503 8.9654384 ]

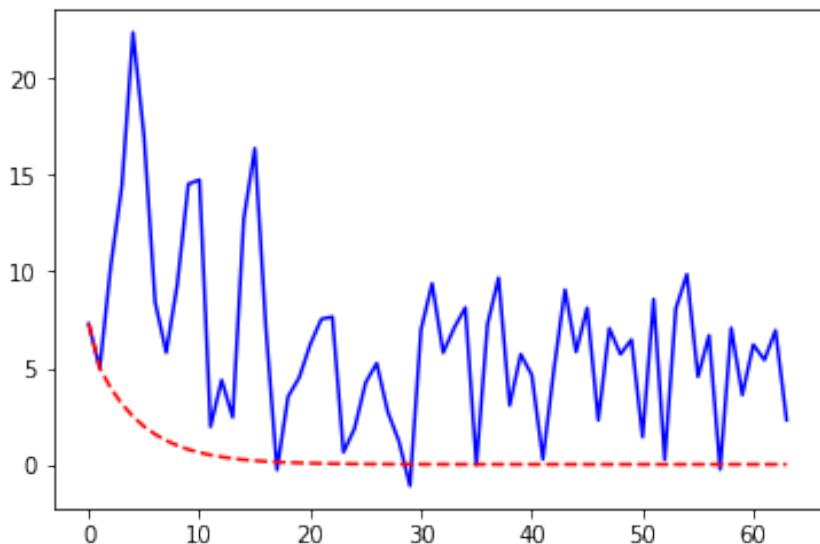
```



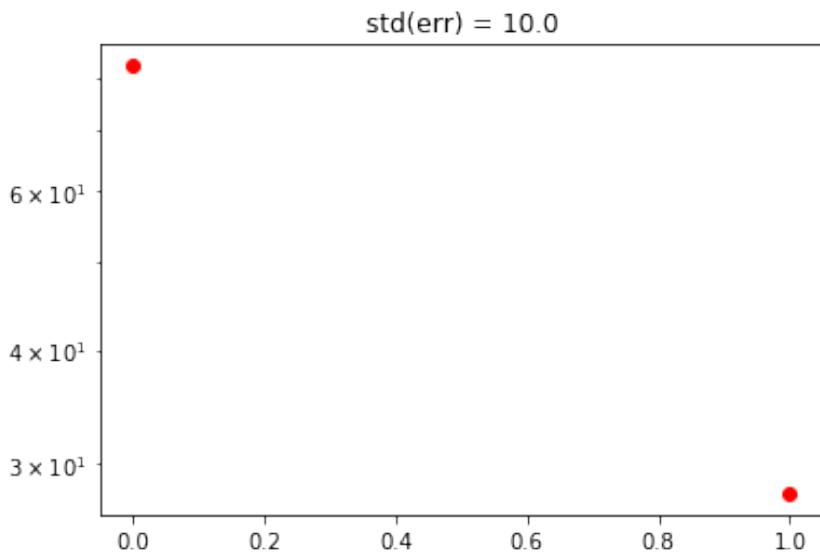
```
std = 10.0 :  
### na_est = 1 :
```



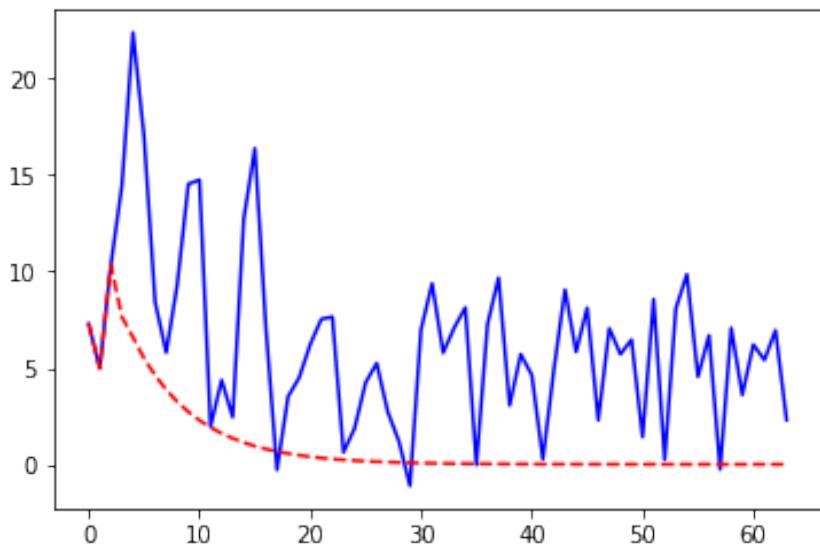
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738  
     0.01181941  0.00287246]  
a_est = [ 1.           -0.79220168]  
b = [0.  2.4 4.4 8.4 7.4]  
b_est = [ 7.28992442 -0.79010848]
```



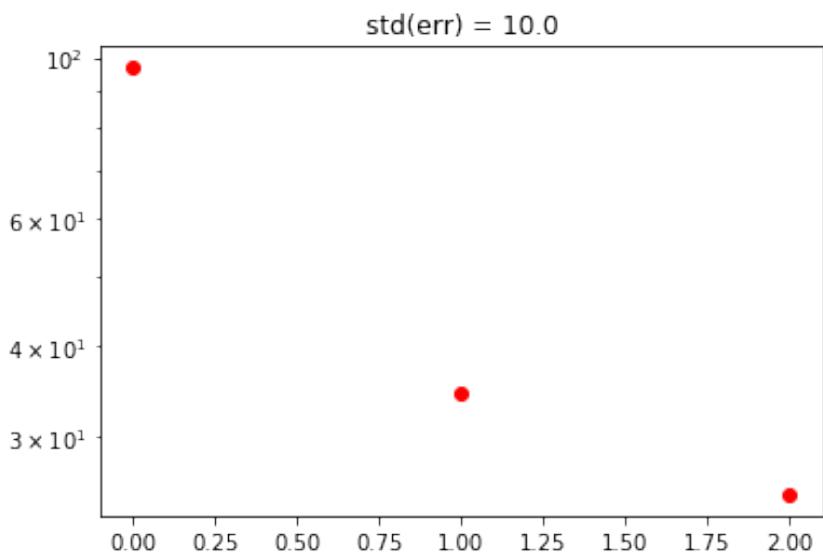
```
### na_est = 2 :
```



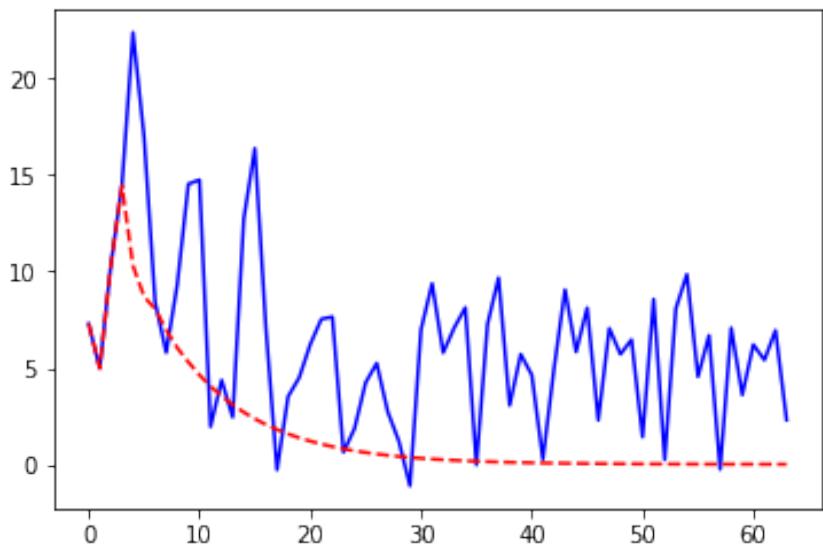
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.67467356 -0.13761492]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [7.28992442 0.06666267 5.95524265]
```



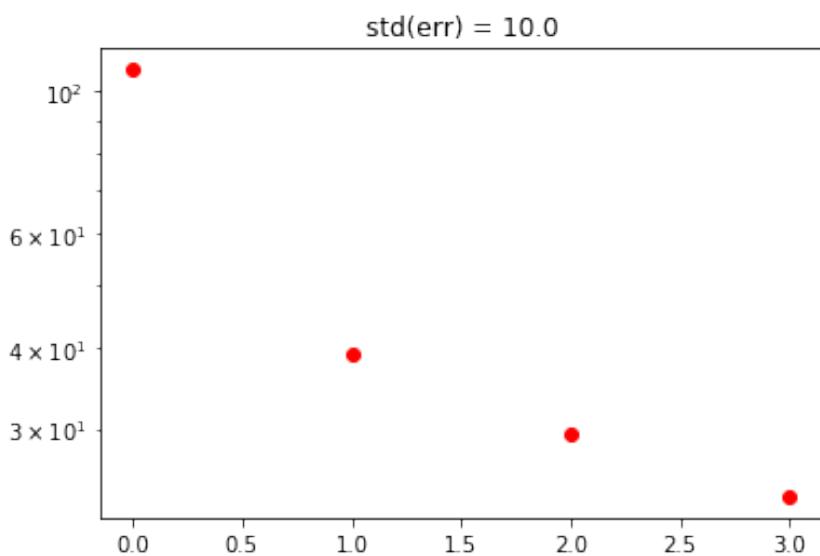
```
### na_est = 3 :
```



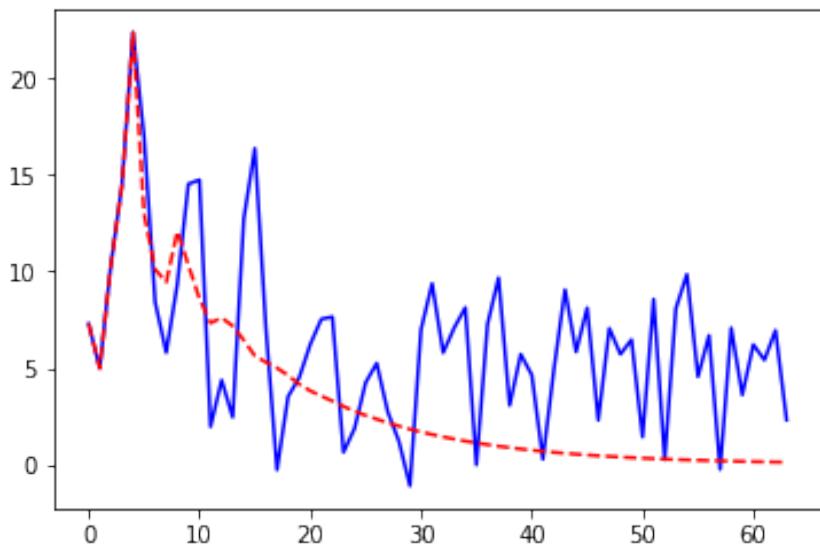
```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.62253339 -0.05687656 -0.1422509 ]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [7.28992442 0.44676054 6.80373691 6.70540784]
```



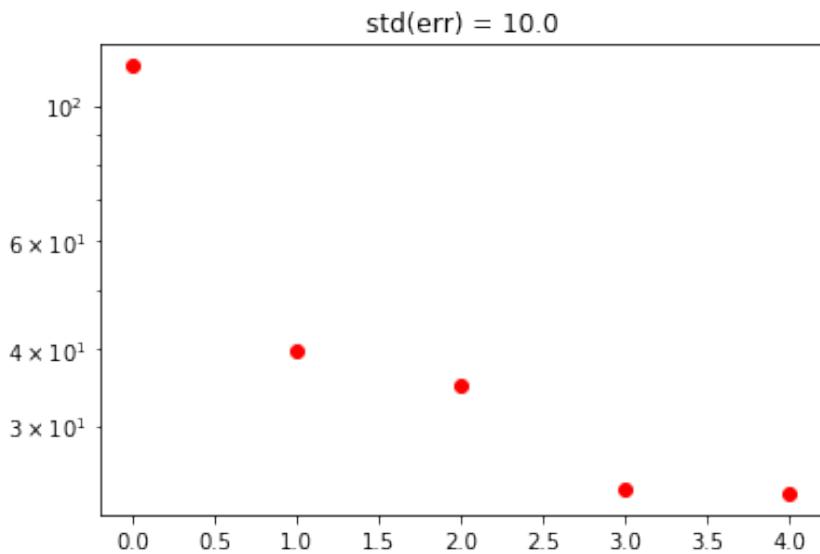
```
### na_est = 4 :
```



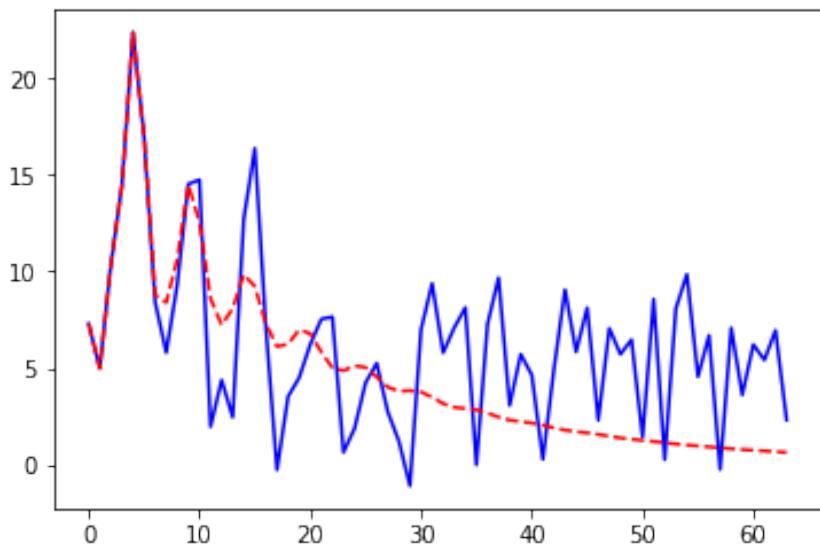
```
a = [ 1.          -0.79          0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.50436326 -0.02963137  0.04002493 -0.33623752]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  1.3082119   7.59142834  9.38971605 12.54430078]
```



```
### na_est = 5 :
```

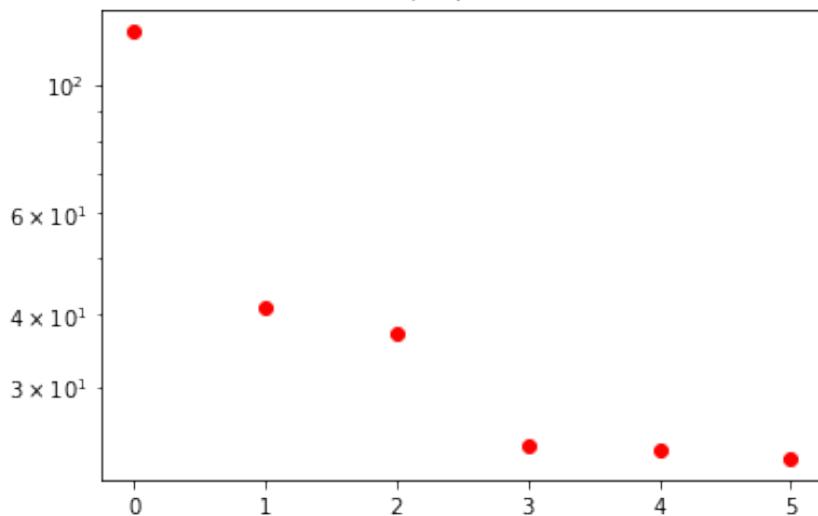


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.2863236  -0.06299163  0.05132874 -0.10883252 -0.42460329]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  2.89770453  8.43515829 11.55635534 17.06508699  6.55357275]
```

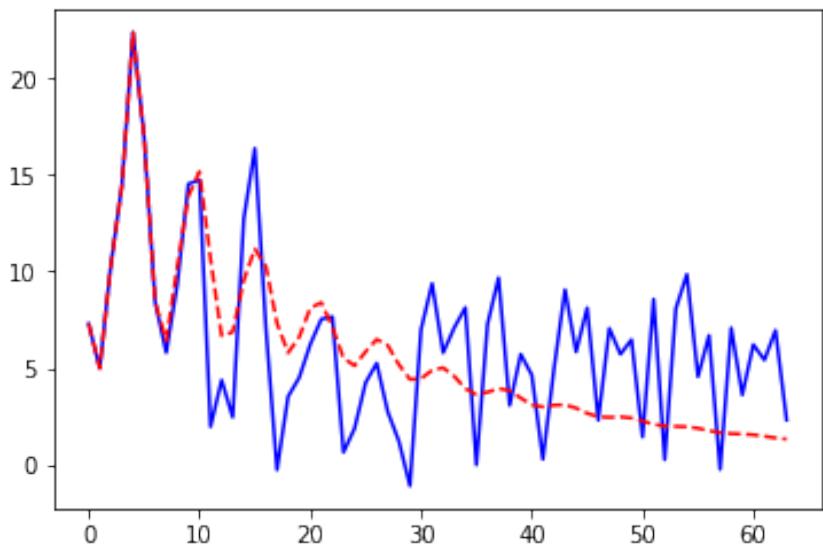


```
### na_est = 6 :
```

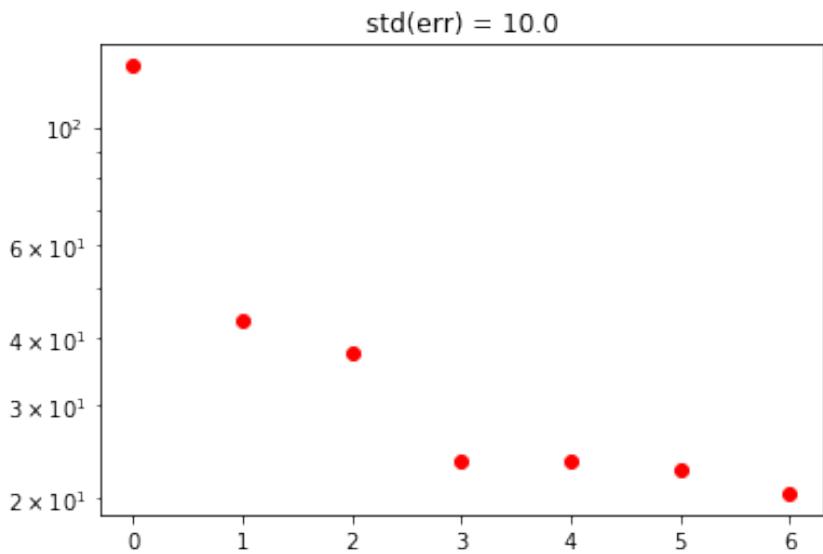
$\text{std}(\text{err}) = 10.0$



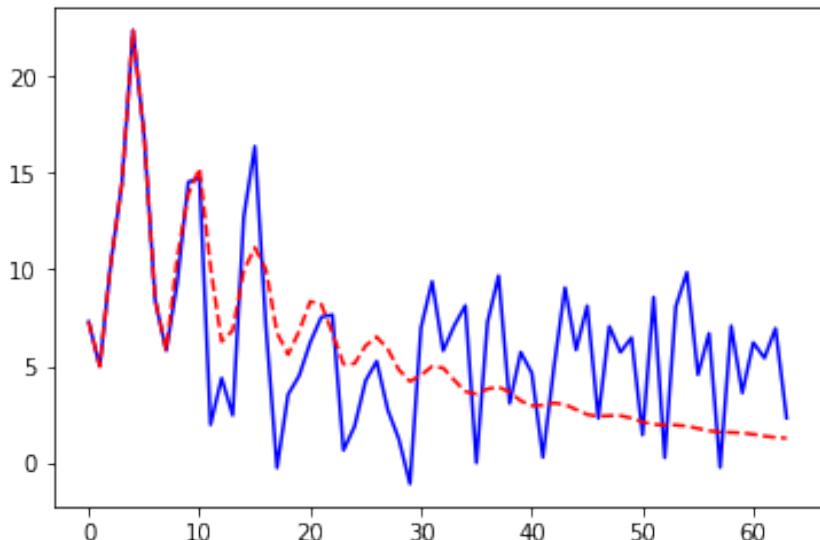
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.          -0.0787228   -0.00669426   0.01898117  -0.09519149  -0.26214904
      -0.39643411]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.41109862  9.88044804 13.74397442 20.58450862 12.93385746
      2.01706411]
```



```
### na_est = 7 :
```

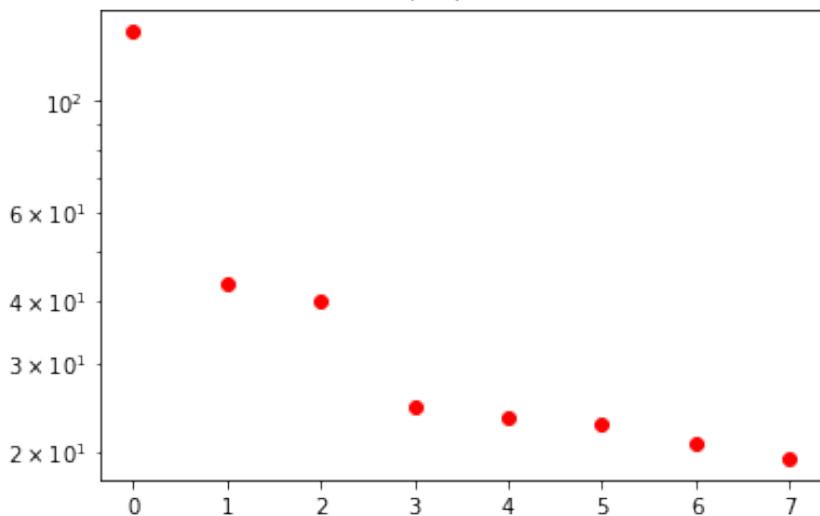


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.12119061 -0.04883063  0.00190074 -0.08983096 -0.25986018
      -0.41330064  0.09467733]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.10151156  9.36157586 12.97107121 19.48979817 11.24115263
      0.04932492 -1.36300901]
```

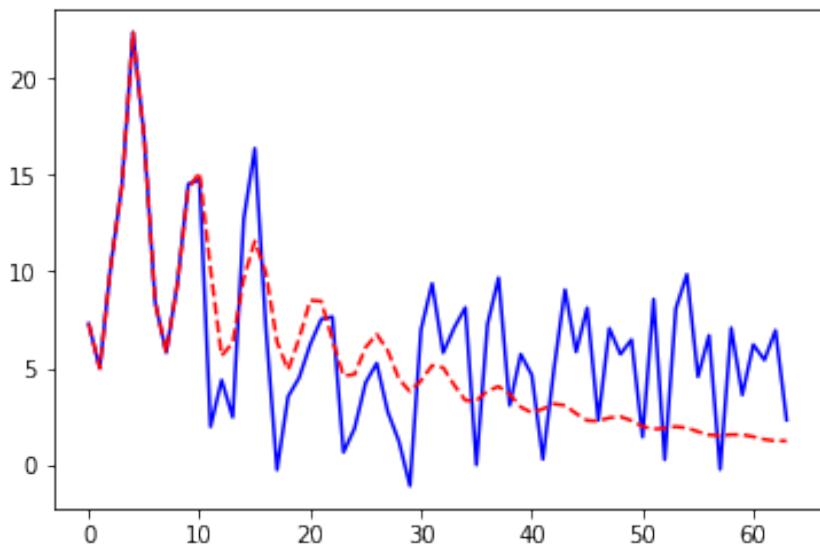


```
### na_est = 8 :
```

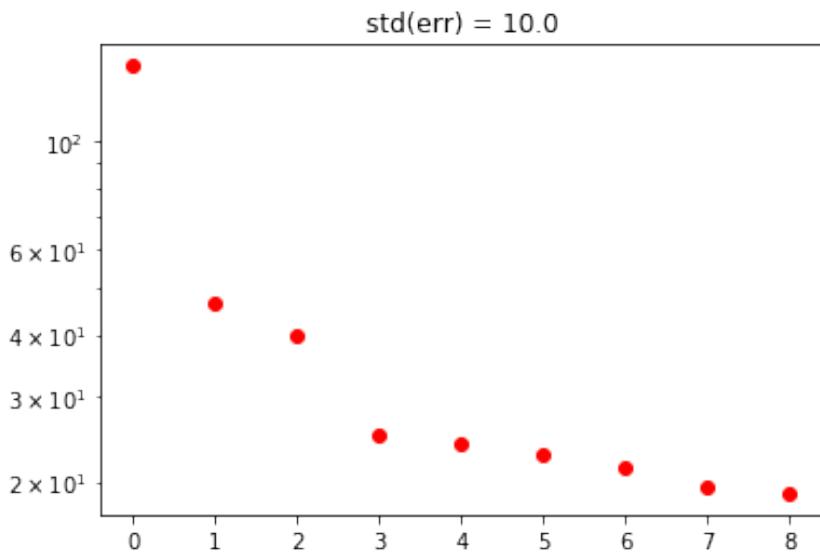
`std(err) = 10.0`



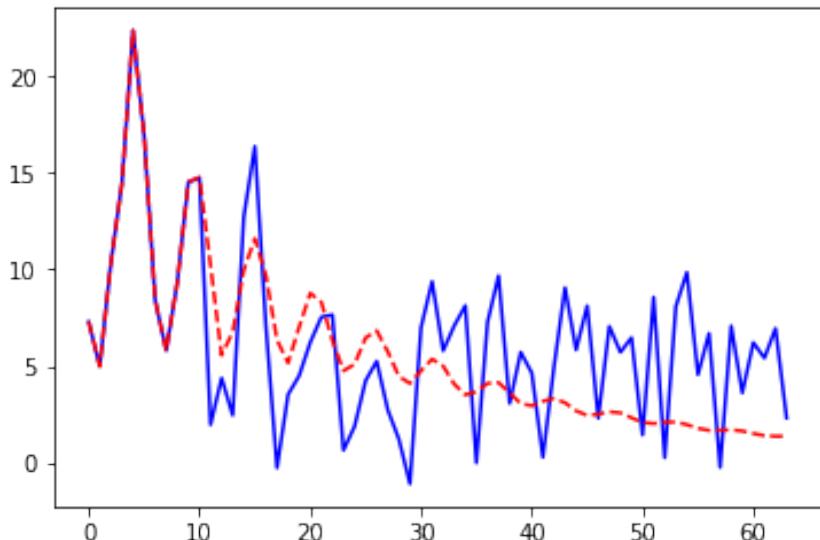
```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008   -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.10754558 -0.09854134 -0.05014811 -0.11511136 -0.25415352
     -0.41488922  0.07399859  0.10938099]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.20098274  9.06720876 12.48467163 18.73013674 10.20662432
     -1.82824939 -3.72300943 -2.33130321]
```



```
### na_est = 9 :
```

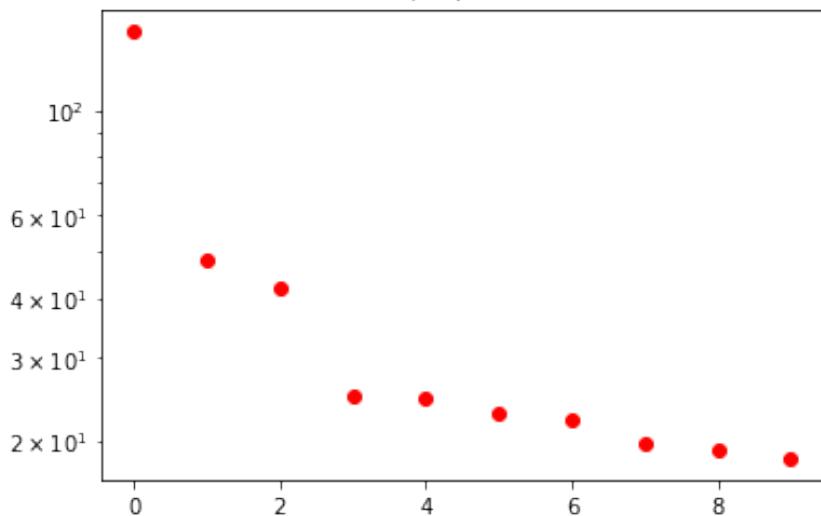


```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.11926227 -0.10599043 -0.0095826  -0.07770048 -0.24135755
      -0.4143199   0.07701138  0.13109281 -0.08771814]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.11556899  8.95449799 12.62232165 18.95886644 10.53510858
      -1.15370765 -2.34223433 -0.57115438  0.82249473]
```

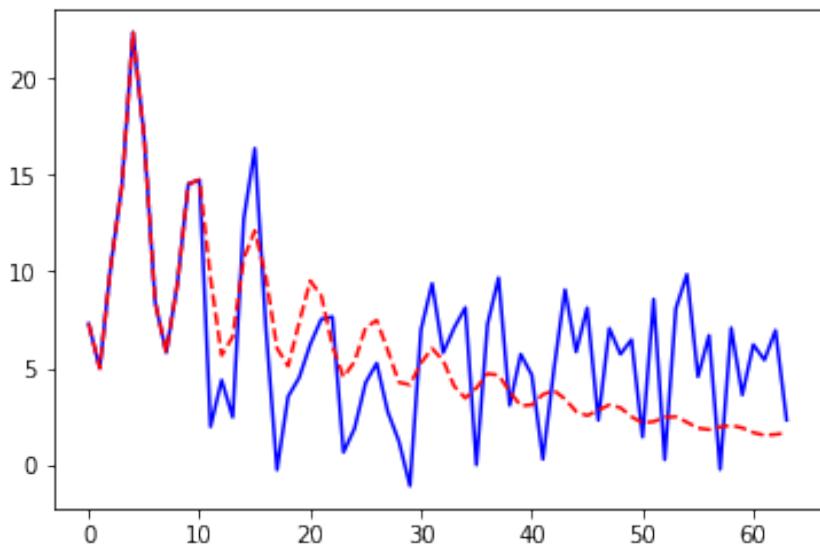


```
### na_est = 10 :
```

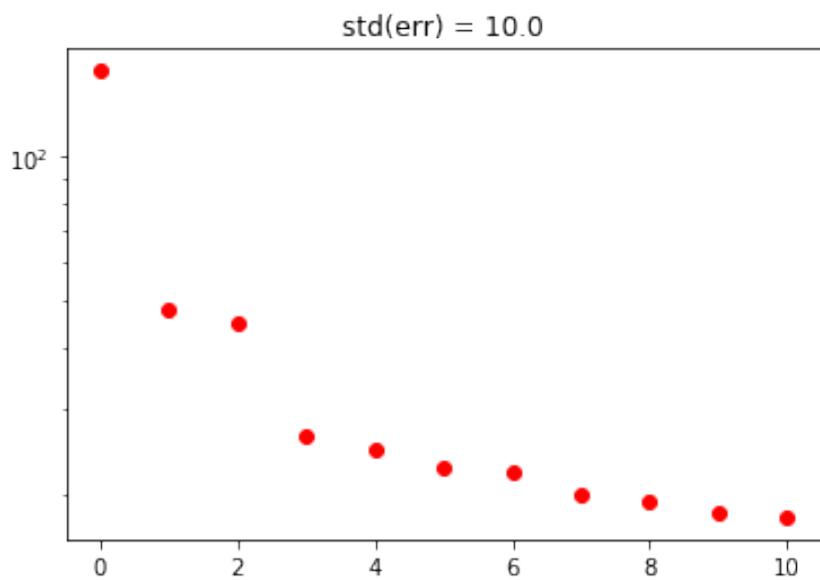
`std(err) = 10.0`



```
a = [ 1.          -0.79          0.1057         0.647529      -0.71747008 -0.01321738
     0.01181941  0.00287246]
a_est = [ 1.          -0.10479861 -0.12881132 -0.0236004   -0.01620881 -0.18431755
     -0.40533085  0.0786387   0.13416922 -0.04707622 -0.13887617]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.22100793  8.86023646 12.55566044 19.31072947 11.10682647
     -0.63705433 -1.38797236  1.40714662  3.48726087  0.8184019 ]
```



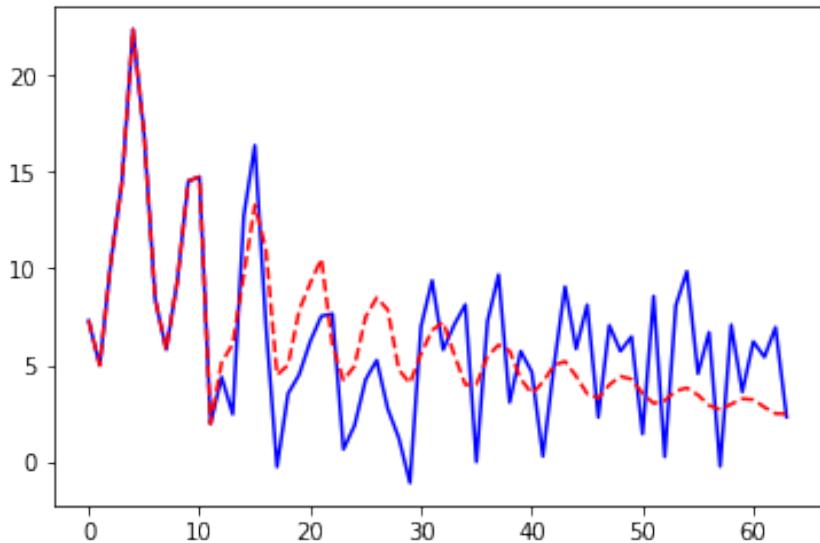
```
### na_est = 11 :
```



```

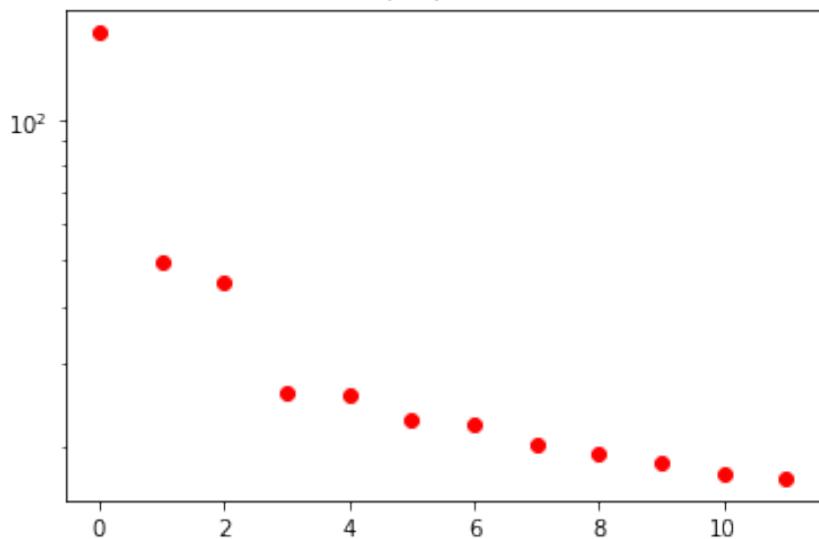
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.00000000e+00 -6.78063792e-02 -1.11304534e-01 -4.67068773e-02
      -3.32223240e-02 -8.39254297e-02 -3.57791714e-01 -7.84600419e-04
      1.50748014e-01 -6.20908782e-02 -1.35958451e-02 -2.37771924e-01]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.49067853  9.17226525 12.85630904 19.78681063 12.59664542
      0.72064641 -0.84862336  2.66174638  5.53915236  3.87372385 -8.35704005]

```

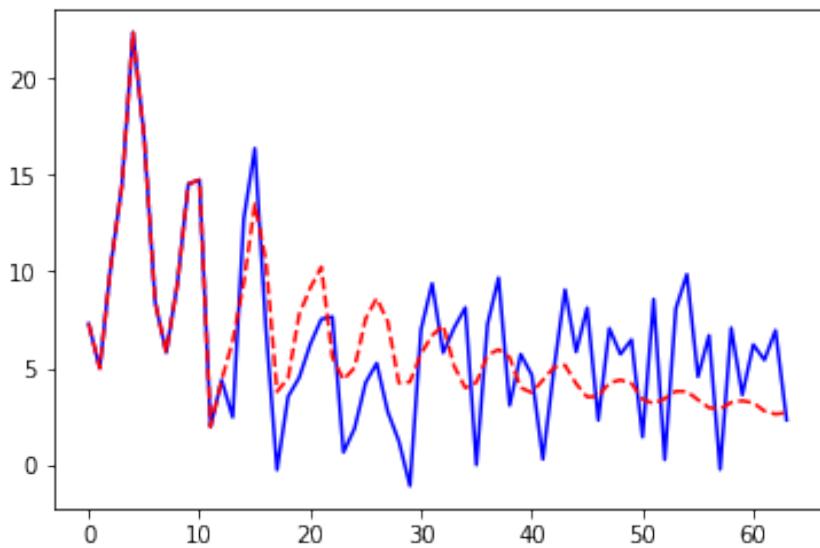


```
### na_est = 12 :
```

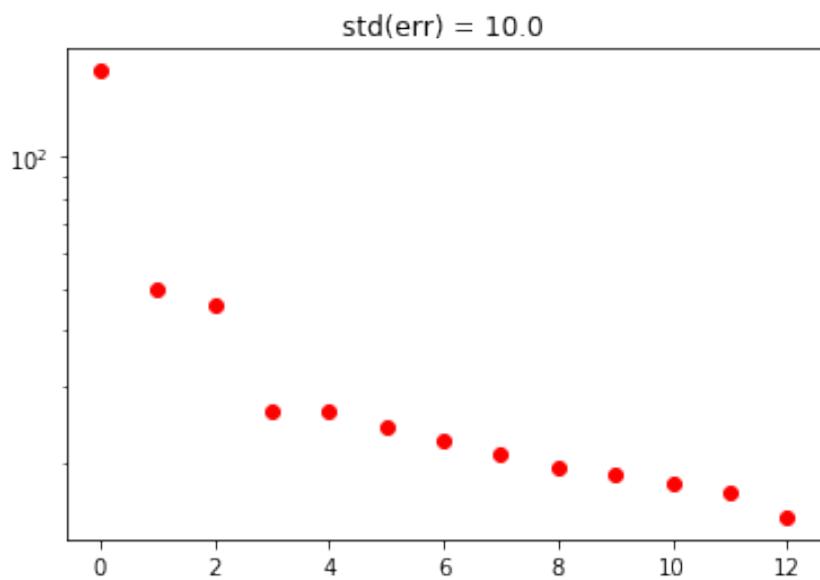
$\text{std}(\text{err}) = 10.0$



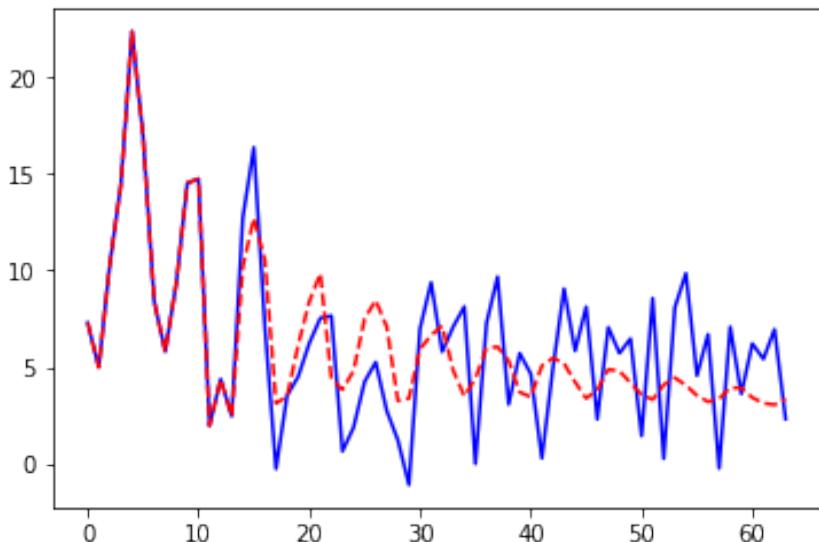
```
a = [ 1.           -0.79          0.1057        0.647529     -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.08618974 -0.13211053 -0.0552218   -0.00920362 -0.06668214
      -0.40906529 -0.03224699  0.13870698 -0.06221997 -0.02597691 -0.26289538
      0.11736985]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.35666524  8.92895043 12.50077124 19.43903808
      12.04189967 -0.22037802 -1.50707727  2.24882008  4.8426475
      2.03983343 -10.72137154 -1.53081323]
```

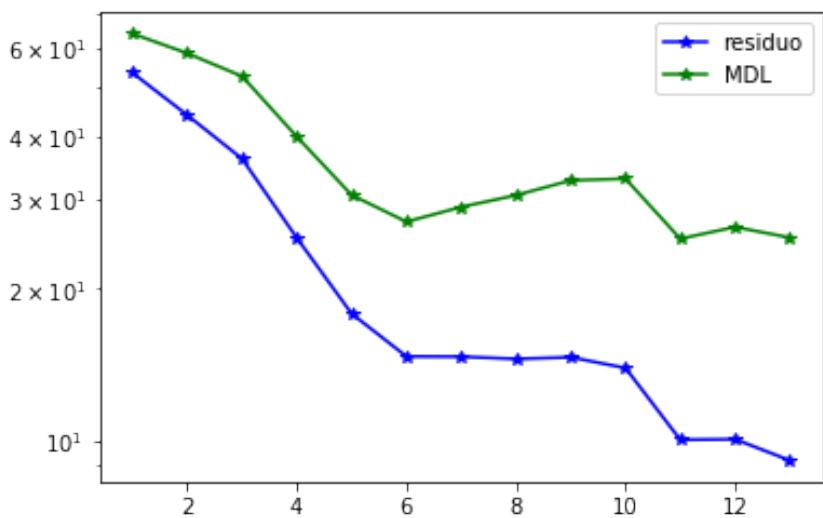


```
### na_est = 13 :
```



```
a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.04753141 -0.10193544 -0.09751875 -0.03532479 -0.02791308
      -0.37132455 -0.15252733  0.05097299 -0.11213293 -0.00741664 -0.29376231
      0.07225354  0.19886514]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.63848152   9.34163556  12.74187079  19.90789647
      13.05939255  0.69931076  -2.28262273   1.13397522   3.94994814
      1.16490038 -13.90074054  -6.34660323  -6.58228185]
```

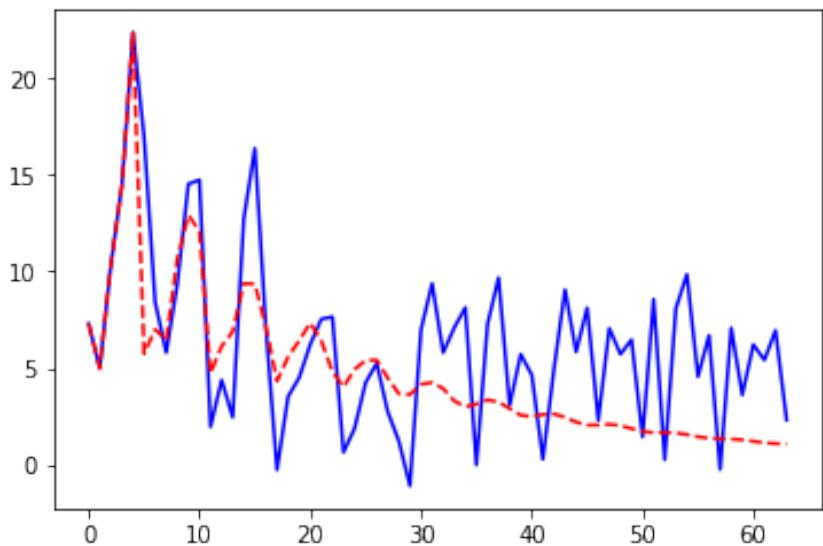




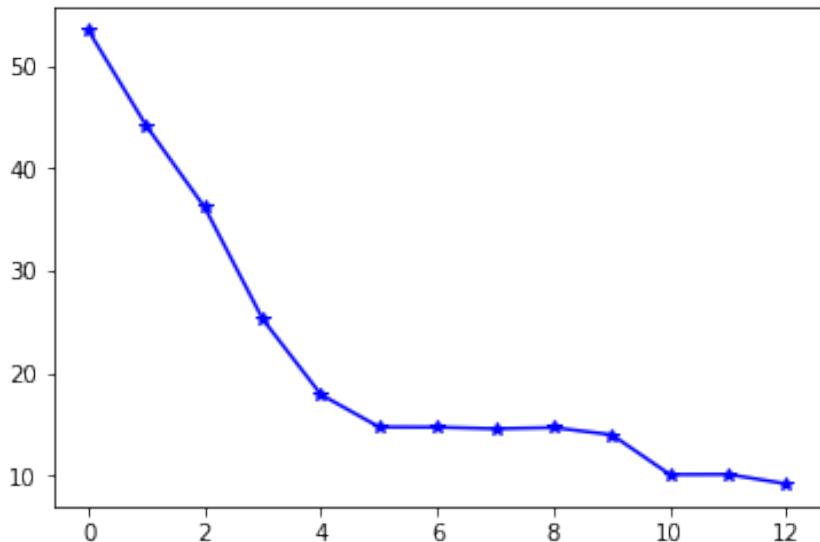
```

a = [ 1.           -0.79           0.1057          0.647529   -0.71747008 -0.01321738
      0.01181941  0.00287246]
a_est = [ 1.           -0.12119061 -0.04883063  0.00190074 -0.08983096 -0.25986018
      -0.41330064  0.09467733]
b = [0.  2.4 4.4 8.4 7.4]
b_est = [ 7.28992442  4.10151156  9.36157586 12.97107121 19.48979817]

```



```
In [5]: plt.figure()
plt.plot(n2res, 'b*-')
plt.show()
```



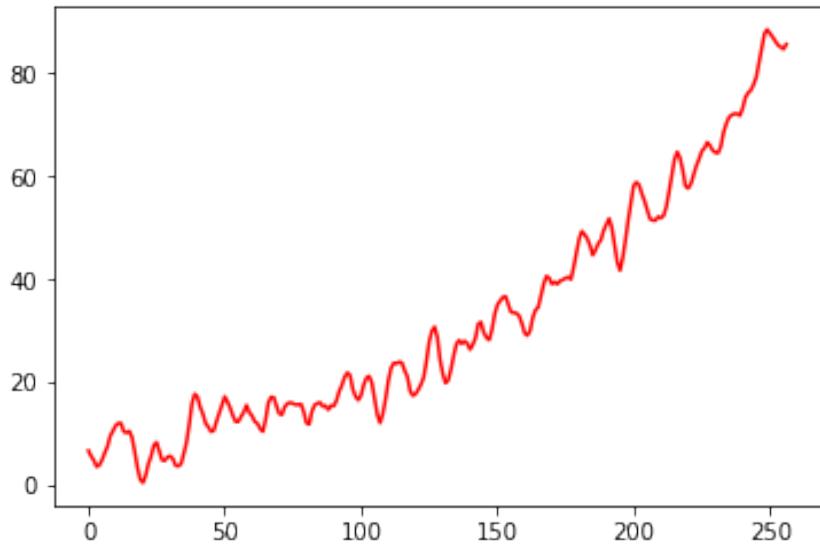
Torna al par. 9.3.2.2

Y.1 Esempio analisi di serie storiche

```
In [6]: # serie-storiche:
N = 256
a = np.poly([0.3, 0.7+0.4j, 0.7-0.4j])
print(a)
a /= a[0]
na = len(a) - 1
stdw = 0.9
print("na = ", na)
y = 7.345*np.exp(0.0097*np.arange(N+1)) + simula_DLTI(np.array([1.]),a,stdw*np.random.randn(N))
plt.figure(3); plt.plot(y, 'r-');
plt.show()

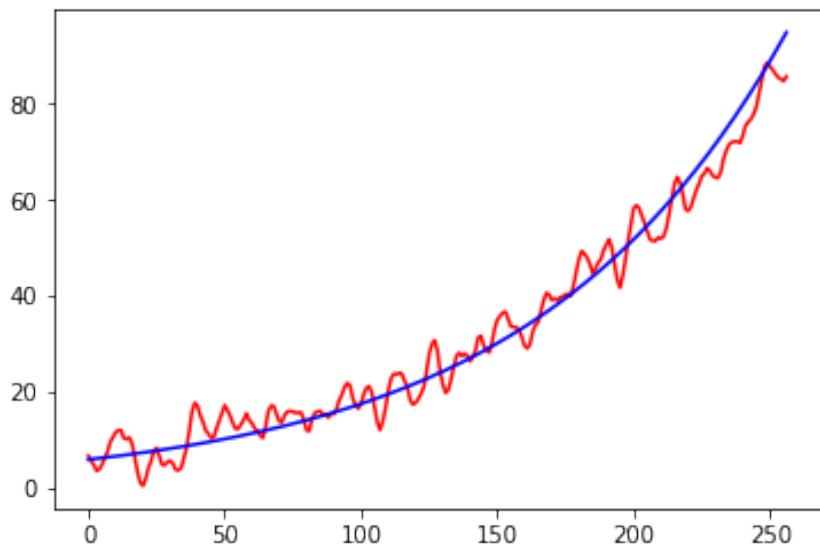
[ 1.      -1.7      1.07   -0.195]
na =  3
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
if nb==1: b=np.array([b, 0]); nb += 1; #endif # serve per non inserire "if" ne
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIETARIO/libro_MNAD/sito_web_MNAD
y[n] = 1./a[0] * np.sum(np.array([ np.dot(-a[1:na],y_past), vb ]));
```

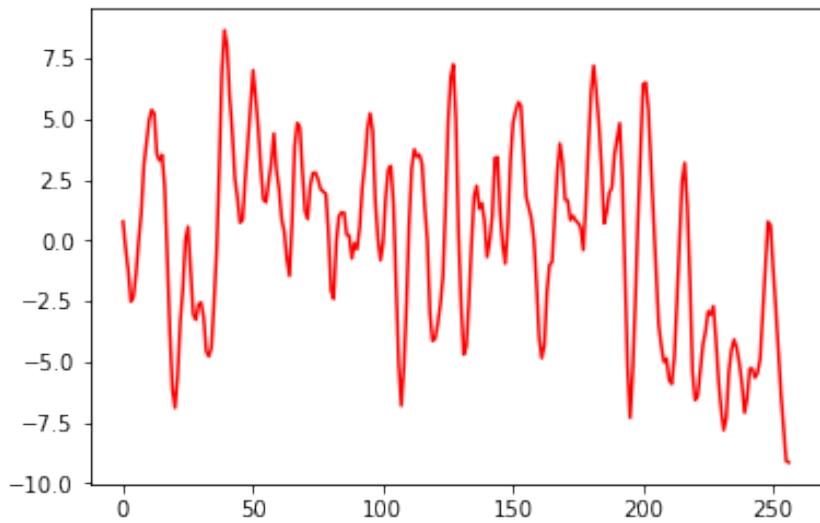


```
In [7]: # stimo il trend
vk = np.arange(0.,N+1)
Ai = np.vander(vk)
A = Ai[:,range(N-1,N+1)]
Qt,Rt = np.linalg.qr(A.copy())
b = np.atleast_2d( np.log(y) ).T
xtrend = np.linalg.solve( Rt , Qt.T@b )
trend = np.exp(xtrend[1])*np.exp(xtrend[0]*np.arange(N+1))
plt.figure(11); plt.plot(y,'r-'); plt.plot(trend,'b-');
print("trend = ", np.exp(xtrend[1]), "*exp(", xtrend[0], "*k)")
```

```
trend = [5.95525494] *exp( [0.0108098] *k)
```



```
In [8]: # elimino il trend  
yflutt = y - trend  
plt.figure(21); plt.plot(yflutt, 'r-');
```



```

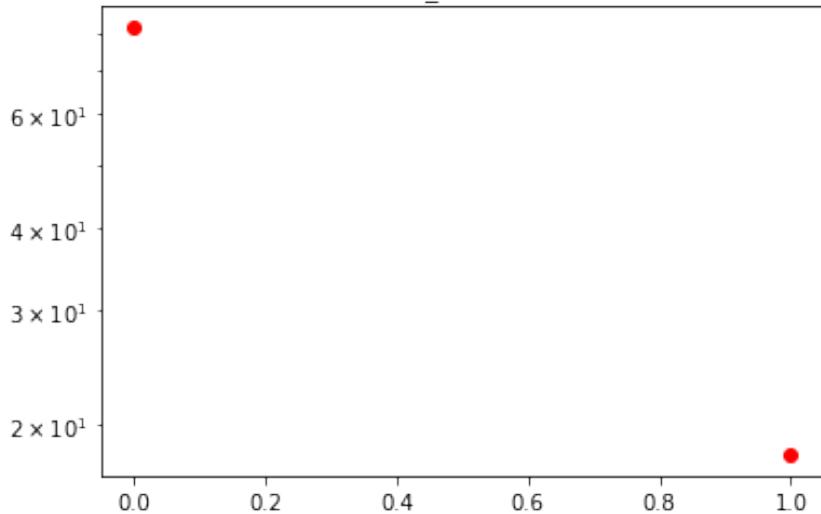
In [9]: max_na = na+10
        # calcolo le correlazioni
        corrval = np.zeros(max_na)
        for i in range(0,max_na):
            corrval[i] = np.sum(np.dot(yflutt, np.roll(yflutt,-i))) / N
        #endifor
        #T = np.asmatrix( toeplitz(corrval[0:5], corrval[0:5]) )
        #print T

In [10]: print('a = ', a)
         jfig = 0
         for na_est in range(2,max_na):
             #A = np.asmatrix( hankel(yflutt[1:N-na_est+1], yflutt[N-na_est:N-1+1]) ) #attualmente non serve
             A = hankel(yflutt[1:N-na_est], yflutt[N-na_est-1:N-1])
             U,S,V = np.linalg.svd(A.copy()); V = V.T
             plt.figure(jfig*20+na_est); plt.semilogy(S,'ro'); plt.title('na_est = ' + str(na_est))
             b = np.atleast_2d( -yflutt[na_est+1:N] ).T
             Q,R = np.linalg.qr(A.copy())
             # risolvo il sistema (10.2.10) :
             a_est = np.ones((na_est+1,1))
             a_est[0:na_est] = np.linalg.solve( R , Q.T@b )
             a_est = a_est[range(na_est,-1,-1)]
             print('a_est = ', a_est)
             print("S = ",S)
             if na_est == na:
                 a_est_ok = np.squeeze( a_est.copy() )
             #endif
             if na_est == na+1:
                 print("S[na] / np.linalg.norm(a) = ",S[na] / np.linalg.norm(a))
             #endif
             jfig += 1
         #endfor
         # errore di predizione a un passo:
         pe = np.zeros(N+1)
         y_est = yflutt.copy()
         for i in range(na_est,N+1):
             y_est[i] = 1./a_est_ok[0] * np.sum( -a_est_ok[1:na+1]*yflutt[i-1:i-na-1:-1] )
             pe[i] = yflutt[i] - y_est[i]
         #endfor
         y_est = y_est + trend
         plt.figure(303); plt.plot(y,'r--'); plt.plot(y_est,'b-'); plt.show()
         plt.figure(333); plt.plot(pe,'b-'); plt.show()
         print("na = ",na)
         print("a = ",a)
         print("a_est_ok = ",a_est_ok)
         print("varianza dell'errore di predizione a un passo: ", np.var(pe))

a =  [ 1.      -1.7     1.07   -0.195]

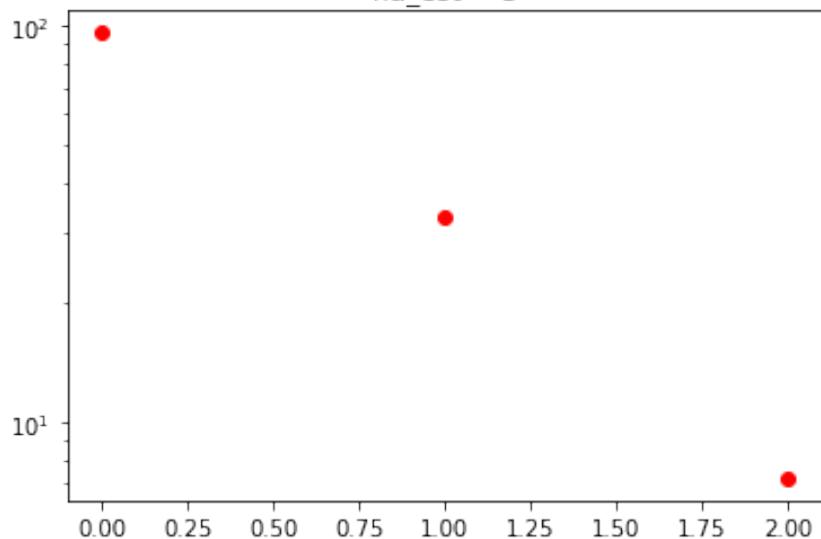
```

na_est = 2

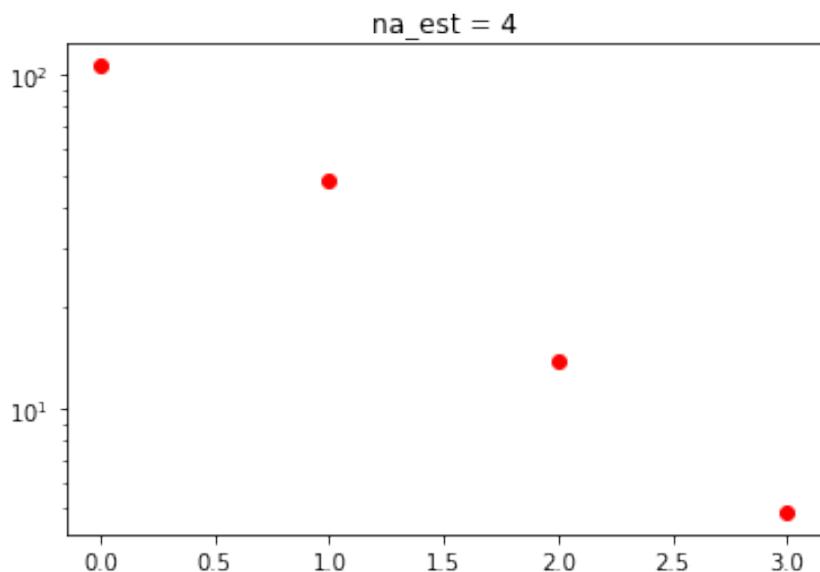


```
a_est = [[ 1.          ]
[-1.6224694 ]
[ 0.78115401]]
S = [81.60981867 17.97457883]
```

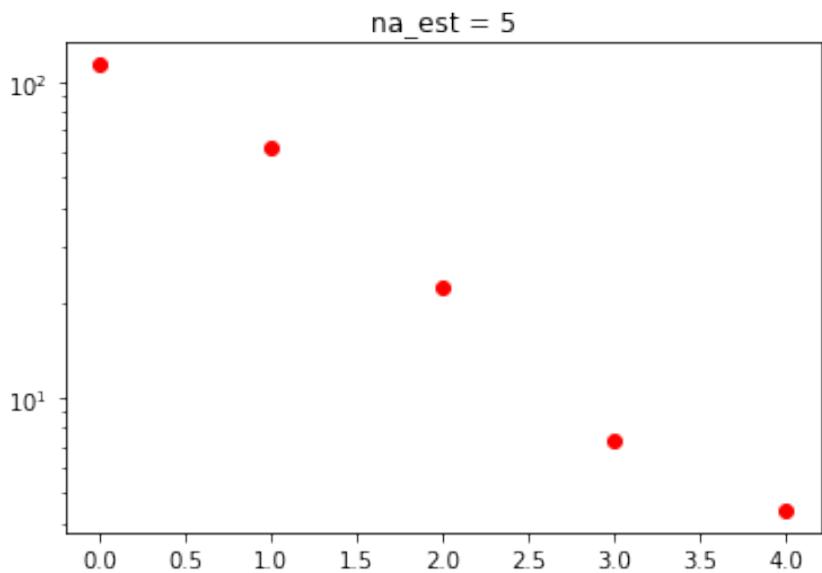
na_est = 3



```
a_est = [[ 1.          ]
[-1.86656739]
[ 1.29273095]
[-0.31972858]]
S = [96.22885923 33.041267    7.22370611]
```



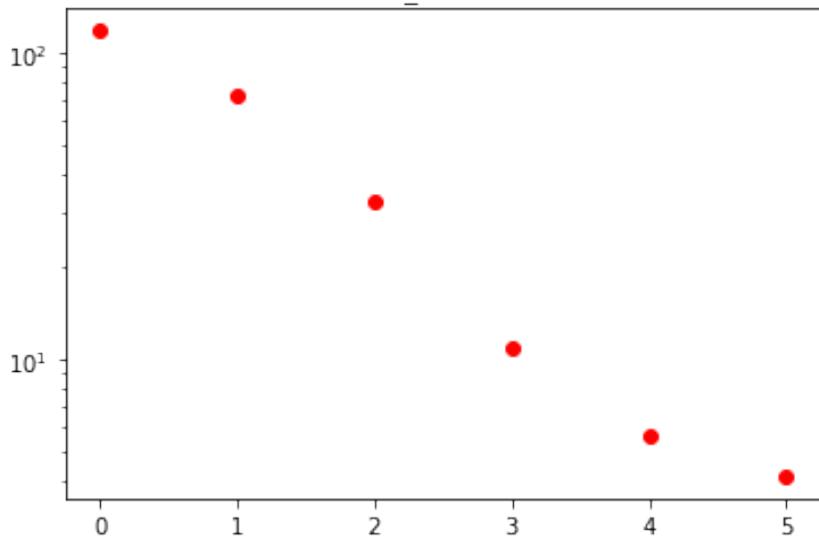
```
a_est = [[ 1.          ]
[-1.81569149]
[ 1.07729583]
[-0.00380028]
[-0.17161088]]
S = [106.24975132 47.90893598 13.7432028   4.87098982]
S[na] / np.linalg.norm(a) = 2.1626587699428543
```



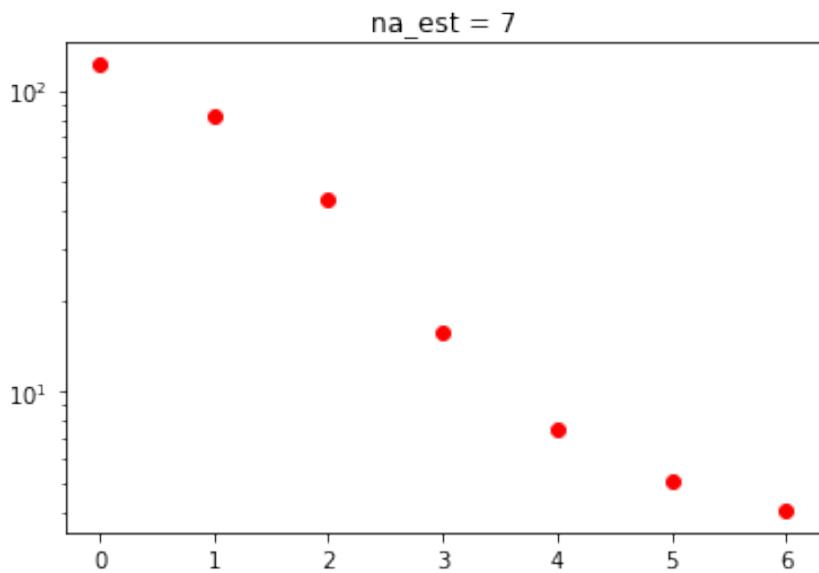
```
a_est = [[ 1.          ]
[-1.80589153]
[ 1.07494834]
[-0.05771597]
[-0.0774414 ]
[-0.05279457]]
```

```
S = [113.21456438  61.53511702  22.37655671  7.26159335  4.39191481]
```

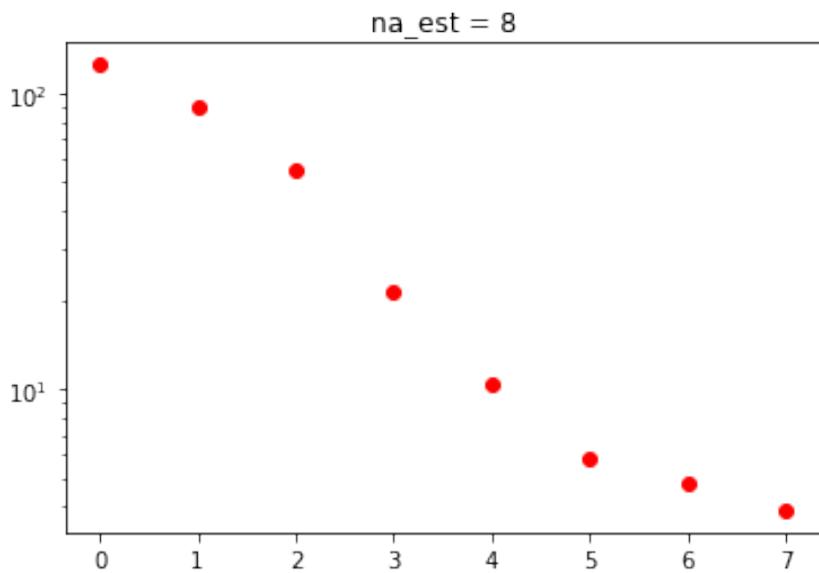
na_est = 6



```
a_est = [[ 1.          ]
[-1.80654121]
[ 1.07424197]
[-0.06037641]
[-0.059492  ]
[-0.08081683]
[ 0.01542598]]
S = [118.16489844  73.24745599  32.52840395  10.84430922  5.68872747
 4.16950685]
```

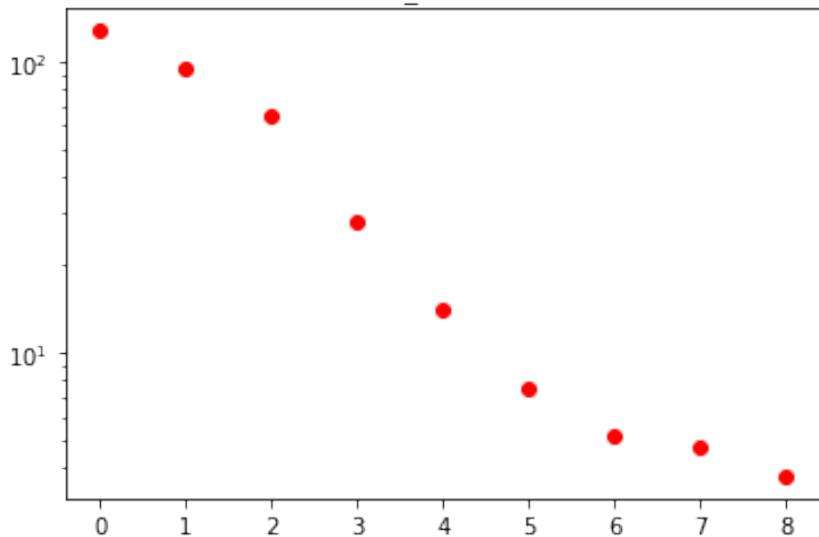


```
a_est = [[ 1.          ]
[-1.80807336]
[ 1.08740925]
[-0.04919367]
[-0.05083722]
[-0.26741289]
[ 0.33294829]
[-0.17820951]]
S = [121.85632357  82.73203673  43.48784096  15.72104001  7.50893144
      5.02903736   4.04441536]
```



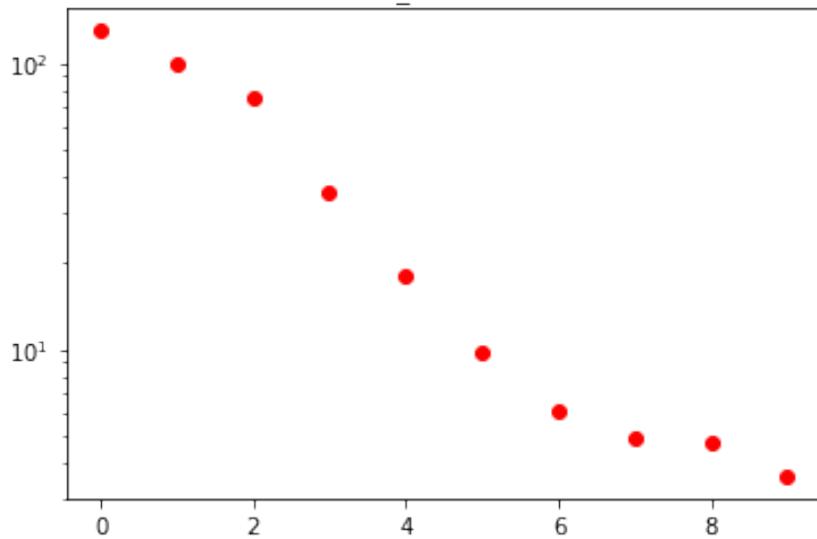
```
a_est = [[ 1.          ]
[-1.80868551]
[ 1.09000069]
[-0.05156164]
[-0.04628714]
[-0.28256748]
[ 0.35472495]
[-0.19660015]
[ 0.00800134]]
S = [124.80412441  89.97179535  54.69215018  21.4608375   10.46436829
 5.80522059   4.82129409   3.89975038]
```

na_est = 9



```
a_est = [[ 1.          ]
[-1.81162121]
[ 1.10531056]
[-0.07413298]
[-0.03023325]
[-0.27996242]
[ 0.36015356]
[-0.26498524]
[ 0.11913212]
[-0.06160429]]
S = [127.35061636  95.20309224  65.4681326   28.01419127  14.03063815
    7.43806352   5.18107796   4.73263875   3.74887365]
```

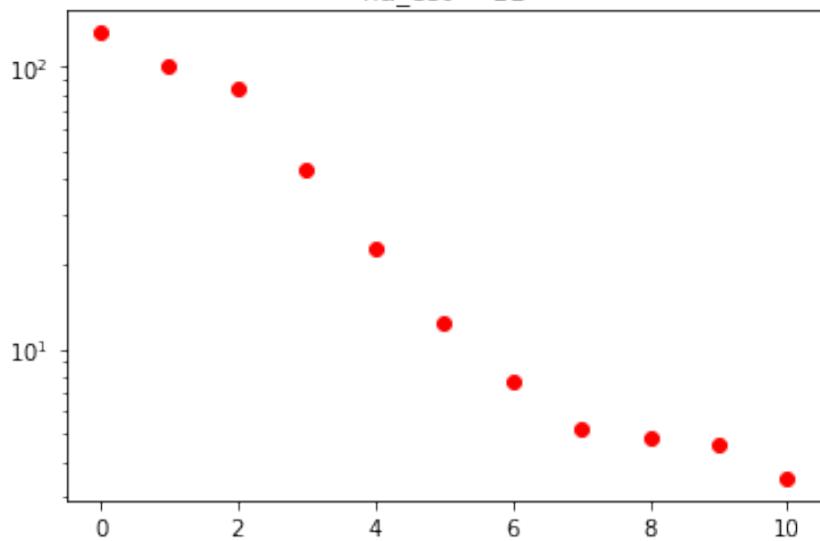
na_est = 10



```
a_est = [[ 1.          ]
[-1.80814979]
[ 1.10581428]
[-0.06902565]
[-0.04781515]
[-0.26052221]
[ 0.36672877]
[-0.27337118]
[ 0.04688753]
[ 0.07444791]
[-0.07845725]]
```

```
S = [129.7284336   98.74951317   75.24728789   35.25411512   18.15637568
     9.73588031    6.0518744    4.92172946    4.68554833    3.59781571]
```

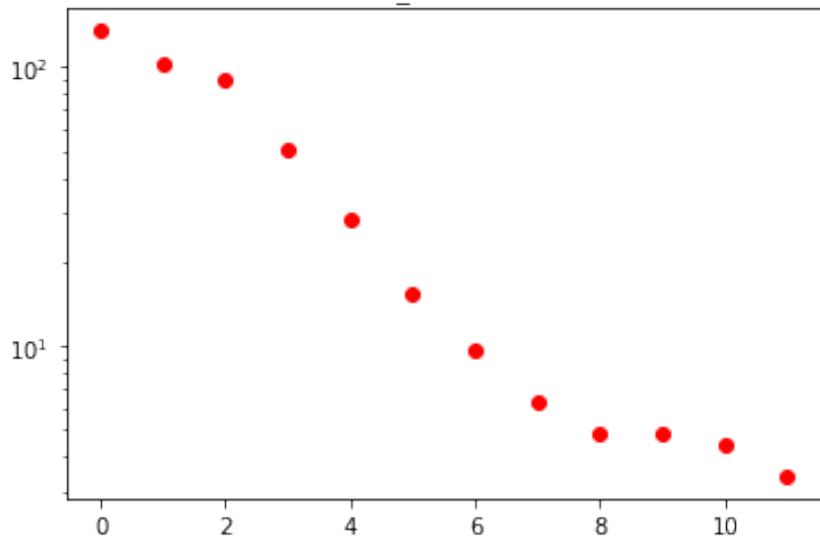
na_est = 11



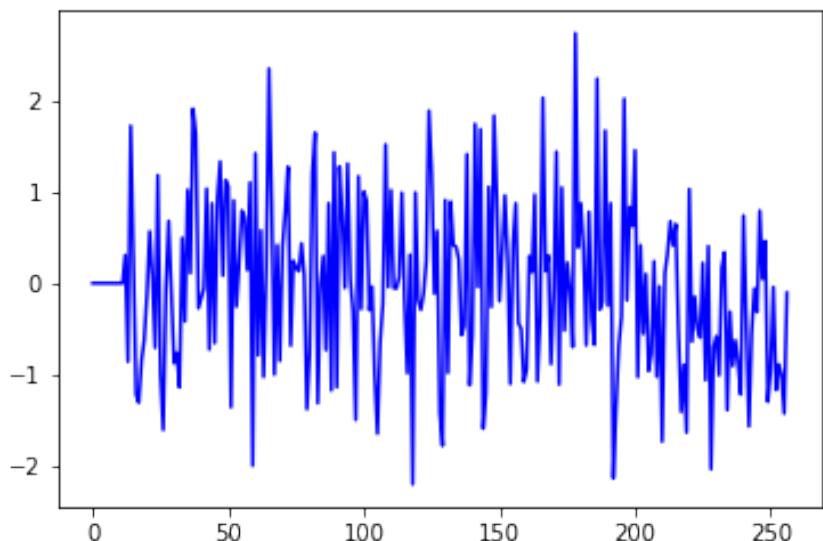
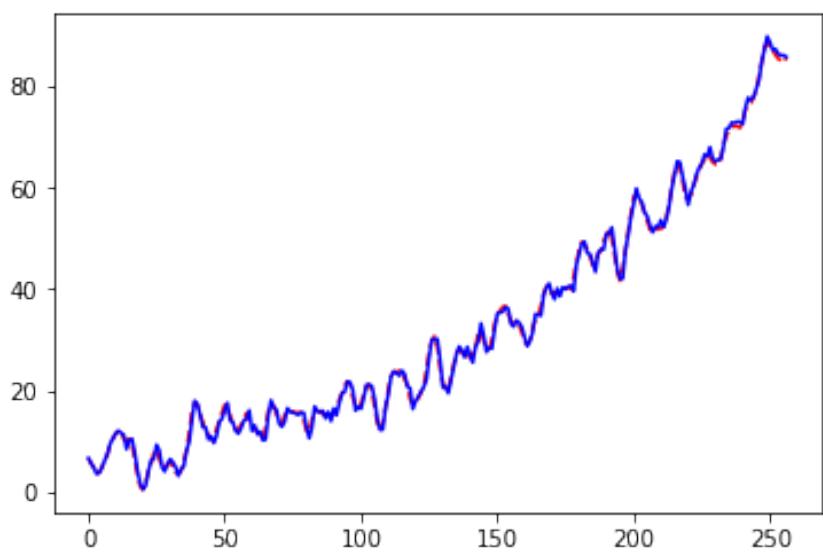
```
a_est = [[ 1.  
[-1.8027648 ]  
[ 1.09717436]  
[-0.0611548 ]  
[-0.04881276]  
[-0.26595748]  
[ 0.37258532]  
[-0.26835032]  
[ 0.04006152]  
[ 0.0495688 ]  
[-0.02758124]  
[-0.02993917]]
```

```
S = [132.08572649 100.95261163 83.61289175 42.96294057 22.91192957  
12.34620089 7.69946361 5.20067842 4.85222695 4.60765092  
3.49388129]
```

na_est = 12



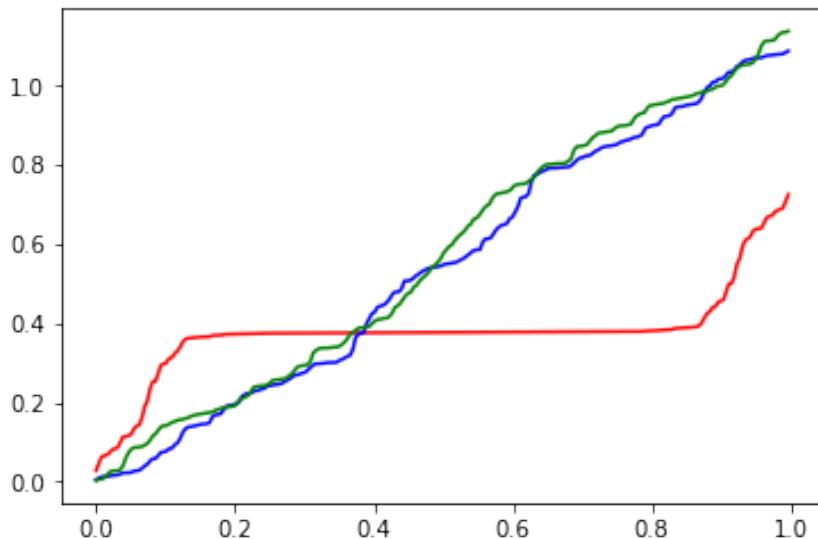
```
a_est = [[ 1.          ]
[-1.79729684]
[ 1.10011608]
[-0.07060486]
[-0.04763036]
[-0.23850518]
[ 0.3291351 ]
[-0.23573011]
[ 0.05006746]
[ 0.04911006]
[-0.16762627]
[ 0.21158458]
[-0.13677645]]
S = [134.50303032 102.13154082 90.38137113 50.90680884 28.26183485
15.33191306   9.59850656   6.29226999   4.88947711   4.84272751
4.44270944   3.41759743]
```



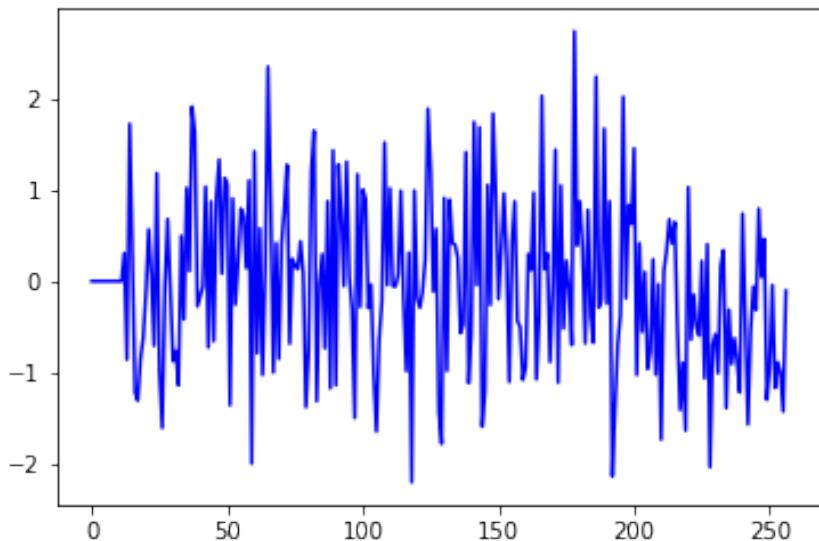
```
na = 3
a = [ 1.      -1.7      1.07   -0.195]
```

```
a_est_ok = [ 1.           -1.86656739  1.29273095 -0.31972858]
varianza dell'errore di predizione a un passo:  0.8637795561694229
```

```
In [11]: # vediamo se ho "sbiancato" la serie; applico il periodogramma cumulato di Bartlett
CP_yf,f = testBartlett_wn(yflutt)
plt.figure(401); plt.plot(f, CP_yf, 'r-');
CP_pe,f = testBartlett_wn(pe)
plt.plot(f, CP_pe, 'b-')
# per confronto riporto il periodogramma cumulato del rumore bianco:
var_s = 10
wn = np.sqrt(var_s) * np.random.randn(N)
CP_wn,f = testBartlett_wn(wn)
plt.plot(f, CP_wn, 'g-');
plt.show()
```

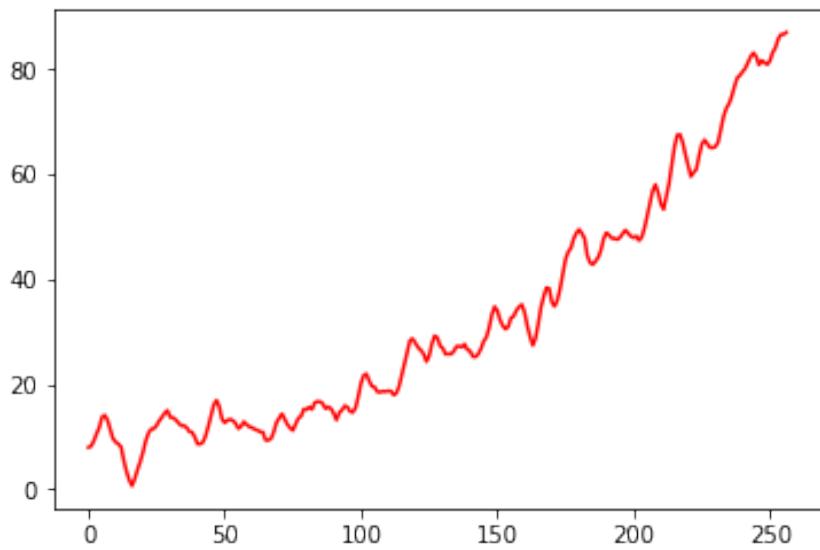


```
In [12]: plt.figure(501); plt.plot(pe, 'b-'); plt.show()
```



```
In [13]: # ripetiamo il calcolo dell'errore di predizione per una diversa realizzazione:
y = 7.345*np.exp(0.0097*np.arange(N+1)) + simula_DLTI(np.array([1.]),a,stdw*np.random.randn(N))
plt.figure(601); plt.plot(y,'r-'); plt.show()
b = np.atleast_2d( np.log(y) ).T
xtrend = np.linalg.solve( Rt , Qt.T@b )
trend = np.exp(xtrend[1])*np.exp(xtrend[0]*np.arange(N+1))
yflutt = y - trend
for i in range(na_est,N+1):
    y_est[i] = 1./a_est_ok[0] * np.sum( -a_est_ok[1:na]*yflutt[i-na:-1] )
    pe[i] = yflutt[i] - y_est[i]
#endfor
print("varianza dell'errore di predizione a un passo: ", np.var(pe), " (stdev=", s
```

```
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPRIO/libro_MNAD/sito_web_MNAD
if nb==1: b=np.array([b, 0]); nb += 1; #endif # serve per non inserire "if" ne
/Users/marcuzzi/Documents/MATERIALE_BIBLIOGRAFICO_PROPRIOPRIO/libro_MNAD/sito_web_MNAD
y[n] = 1./a[0] * np.sum(np.array([ np.dot(-a[1:na],y_past), vb ]));
```



varianza dell'errore di predizione a un passo: 1.493710222666754 (stdev= 1.222

Torna al par. [9.4.1](#)

In [14]:

Appendice Z

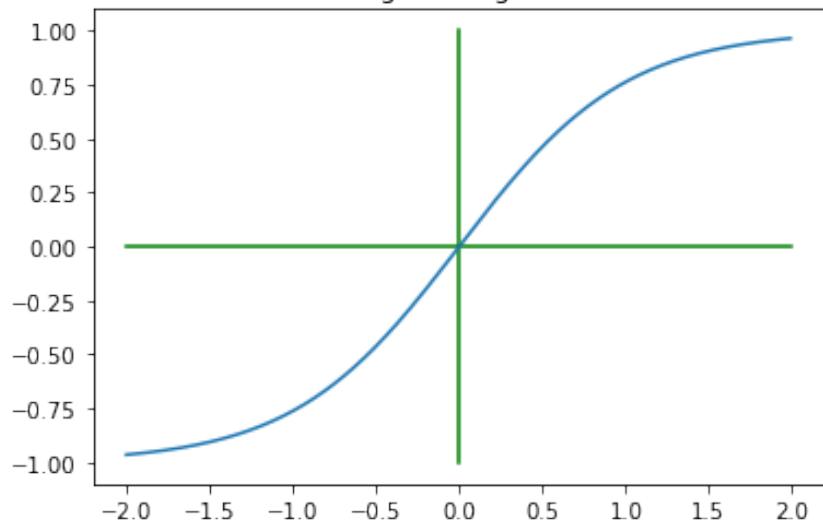
Esempi di apprendimento neurale

```
In [1]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Dou  
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non  
#get_ipython().magic(u'matplotlib inline')  
%matplotlib inline  
import numpy as np  
import matplotlib.pyplot as plt  
from libreria_NLALD.apprendimento_neurale import *
```

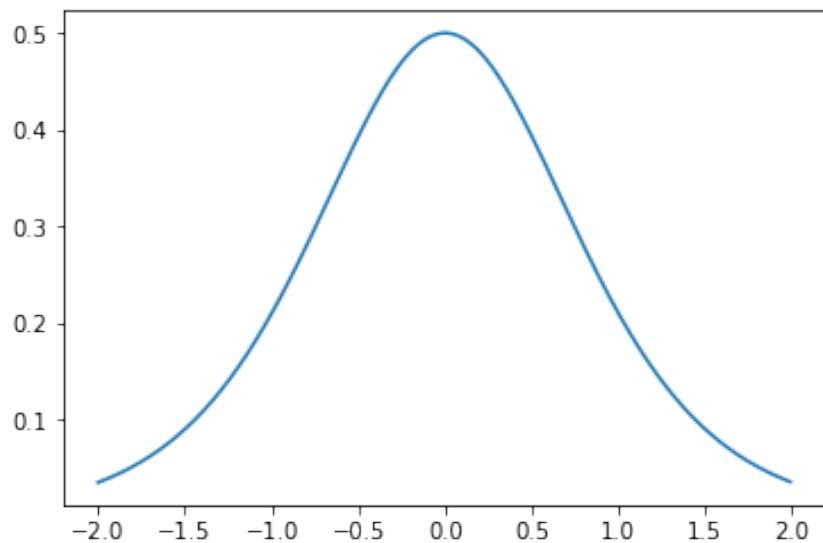
```
In [2]: x = np.arange(-2.,2.,0.01)
    s = sigmoid_logistic(x)
    plt.figure(1), plt.plot([-2., 2.],[0., 0.],'g-'); plt.plot([0., 0.],[-1., 1.],'g-')
    plt.title("Sigmoid logistic")
```

Out [2]: Text(0.5,1,'Sigmoid logistic')

Sigmoid logistic



```
In [3]: Ds = sigmoid_logistic_derivative(x)
plt.figure(1), plt.plot(x,Ds);
```



```

In [4]: # apprendimento legge AND:
n_neurons_first_layer = 1
input = np.array([[0, 0, 1, 1],[0, 1, 0, 1]])
target = np.array([[0, 0, 0, 1]])
toll = 1e-3
#W1,W2,err_pred_hist,target_pred_hist = apprendimento_neurale(n_neurons_first_layer)
#plt.figure(1); plt.semilogy(err_pred_hist);

In [5]: #print(np.squeeze(np.array(target_pred_hist[:, -1])))

In [6]: # apprendimento legge XOR:
n_neurons_first_layer = 1
input = np.array([[0, 0, 1, 1],[0, 1, 0, 1]])
target = np.matrix([[0, 1, 1, 0]])
toll = 1e-3
#W1,W2,err_pred_hist,target_pred_hist = apprendimento_neurale(n_neurons_first_layer)
#plt.figure(1); plt.semilogy(err_pred_hist);

In [7]: #np.squeeze(target_pred_hist[:, -1])

In [8]: # apprendimento legge XOR con più neuroni:
input = np.array([[0, 0, 1, 1],[0, 1, 0, 1]])
target = np.array([[0, 1, 1, 0]])
n_neurons_first_layer = 5
toll = 1e-3
#W1,W2,err_pred_hist,target_pred_hist = apprendimento_neurale(n_neurons_first_layer)
#plt.figure(4); plt.semilogy(err_pred_hist);

In [9]: #np.squeeze(np.array(target_pred_hist[:, -1]))

In [10]: # approssimazione di una funzione:
h = 0.8
y_veri = np.concatenate((np.sin(np.arange(0.,2.*pi+h,h)), -np.sin(np.arange(0.,2.*pi+h,h))))
maxy = max(y_veri)
miny = min(y_veri)
y_veri = (y_veri - (maxy+miny)/2.) / ((maxy-miny)/1.8)

In [11]: n_neurons_first_layer = 18
toll = 1e-3
input = np.atleast_2d(np.arange(len(y_veri)))
#W1,W2,err_pred_hist,target_pred_hist = apprendimento_neurale(n_neurons_first_layer)
#plt.figure(1); plt.semilogy(err_pred_hist);
#u_target_pred = np.squeeze(np.array(target_pred_hist[:, -1]))

In [12]: #plt.figure(1); plt.plot(y_veri, 'b.-'); plt.plot(target_pred_hist[:, -1], 'r.');


```

Torna al par. ??

Z.1 Esempio stima dello stato

```
In [13]: # NB: per eseguire questo notebook come file Python, scegliere il menu "File -> Do
# Da Canopy, scommentando questa istruzione si hanno i grafici nella console e non
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from libreria_NLALD.sistemi_DLTI import *
import libreria_NLALD.control as control

In [14]: N = 16
nx=6
x0 = np.atleast_2d( np.zeros(nx) ).T
A = np.random.rand(nx,nx)
b = np.atleast_2d( np.random.rand(nx) ).T
c = np.atleast_2d( np.random.rand(nx) )
# vediamo per quali combinazioni di misura degli stati il sistema e' osservabile :
# caso di 1 uscita:
for i in range(nx):
    c = np.zeros(nx); c[i]=1.0
    OSS = c.copy()
    for k in range(nx-1):
        OSS = np.vstack((c, OSS@A))
    #endfor
    S = np.linalg.svd(OSS,compute_uv=False)
    print(str(i) + ' : indice di osservabilita' = ' + str(len(np.where(S>np.linalg
#endiffor

0 : indice di osservabilita = 6
1 : indice di osservabilita = 6
2 : indice di osservabilita = 6
3 : indice di osservabilita = 6
4 : indice di osservabilita = 6
5 : indice di osservabilita = 6

In [15]: nx=6;
u = 0.0 * np.atleast_2d( np.random.rand(N) )
x0 = np.atleast_2d( np.zeros(nx) ).T
Q,R = np.linalg.qr(np.random.randn(nx,nx))
Q = np.asmatrix(Q)
D = 0.1 + np.diag(0.1*np.arange(nx)) # scegliendo gli autovalori di A, impongo un
print("autovalori di A = ",np.diag(D))
A=Q@D@Q.T
b = np.atleast_2d( np.random.rand(nx) ).T
c = np.atleast_2d( np.random.rand(nx) )
```

```
h, X_hist = simula_DLTI_StateSpace_discreto(A,b,c,[0],u,x0);
h = np.squeeze(np.array(h))
```

```
autovalori di A = [0.1 0.2 0.3 0.4 0.5 0.6]
```

```
In [16]: # calcola il guadagno dell'osservatore
desired_eigs = 0.33*np.ones(nx) + 0.005*np.arange(nx)
Ko = np.atleast_2d( control.acker(A.T, c.T, desired_eigs) ).T
Ao = A - Ko*c;
#print np.linalg.eig(Ao)
```

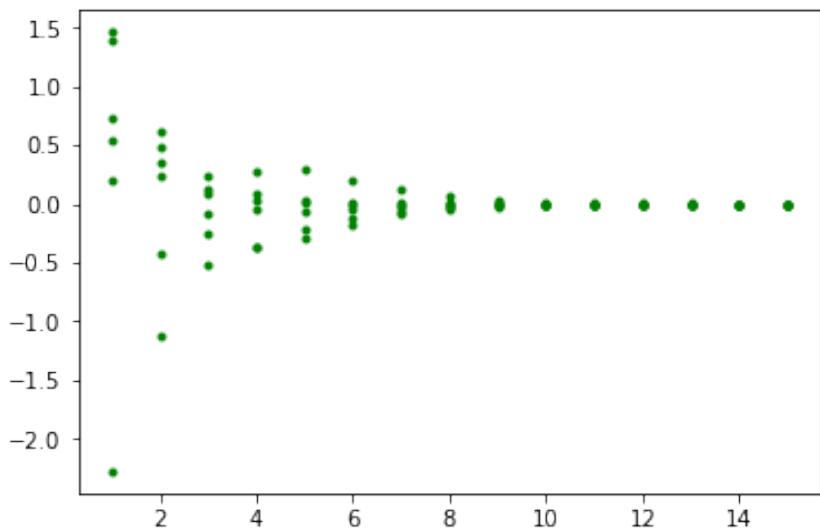
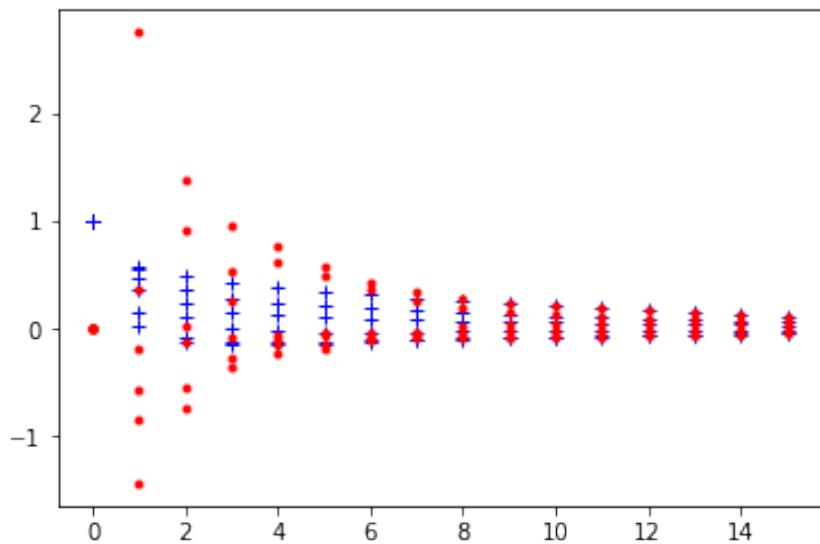
```
In [17]: L,V = np.linalg.eig(Ao)
print("desired eigs modulus = ",np.abs(np.sort(desired_eigs)))
print("actual eigs modulus = ",np.abs(np.sort(L)))
```

```
desired eigs modulus = [0.33 0.335 0.34 0.345 0.35 0.355]
actual eigs modulus = [0.32999998 0.33500015 0.33999964 0.34500042 0.34999976 0.35]
```

```
In [18]: x0_est = np.atleast_2d( np.zeros(nx) ).T
x = 1.0 + x0.copy()
x_est = x0_est.copy()
plt.figure(1); plt.plot(0.,x.T,'b+'); plt.plot(0.,x_est.T,'r.');
plt.figure(2);
y = np.zeros(N)
for i in range(1,N):
    y[i] = c @ x
    x = A @ x + b @ np.atleast_2d(u[0,i])
    x_est = Ao @ x_est + b * u[0,i] + Ko * y[i]
    plt.figure(1); plt.plot(i,x.T,'b+'); plt.plot(i,x_est.T,'r.');
    plt.figure(2); plt.plot(i,x.T-x_est.T,'g.');
```

#endfor

```
plt.show()
```



Torna al par. 10.1.1

In [19]:

Bibliografia

- [1] A. Antoulas. *Approximation of large-scale dynamical systems*. SIAM, 2002.
- [2] A. Azzalini. *Inferenza Statistica*. Springer - UNITEXT, 2001.
- [3] D. Bini, M. Capovani, and O. Menchi. *Metodi Numerici per L'Algebra Lineare*. Zanichelli, 1988.
- [4] A. Bjorck. *Numerical Methods for Least Squares Problems*. SIAM, 1996.
- [5] Henson V.E. Briggs, W.L. *The DFT*. SIAM, 1995.
- [6] Candy. *Signal Processing: the modern approach*. McGraw-Hill, 1989.
- [7] V. Comincioli. *Metodi Numerici e Statistici per le Scienze Applicate*. Casa Editrice Ambrosiana, Milano, 1992.
- [8] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems*, 7:303–314, 1989.
- [9] I. Daubechies. *Ten Lectures on Wavelets*. SIAM, 1992.
- [10] B. De Schutter. Minimal state-space realization in linear system theory: an overview. *JCAM*, 121:331–354, 2000.
- [11] J. Demmel. *Applied Numerical Linear Algebra*. SIAM, 1997.
- [12] G. Golub and C. Van Loan. *Matrix Computations*. John Hopkins University Press, 1996.
- [13] E. Gregorio and L. Salce. *Algebra Lineare*. Libreria Progetto, 2012.
- [14] J. Humpherys, P. Redd, and J. West. A fresh look at the kalman filter. *SIAM Review*, 54:801–823, 2012.
- [15] I.T. Jolliffe. *Principal Component Analysis*. Springer, 2002.
- [16] T. Katayama. *Subspace Methods for System Identification*. Springer, 2005.
- [17] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. SIAM, 1995.

- [18] L. Ljung. *System Identification: theory for the user*. Prentice Hall, 1987.
- [19] G. Marchesini and E. Fornasini. *Teoria dei Sistemi*. Libreria Progetto, 2000.
- [20] J. Nocedal and S.J. Wright. *Numerical Optimization*. Springer, 1999.
- [21] A.V. Oppenheim and R.W. Schafer. *Elaborazione Numerica dei Segnali*. Franco Angeli, 1988.
- [22] D. Piccolo. *Introduzione all'analisi delle serie storiche*. La Nuova Italia Scientifica, 1990.
- [23] A. Quarteroni. *Modellistica numerica per problemi differenziali*. Springer, UNITEXT, 3 ed., 2006.
- [24] A. Quarteroni, R. Sacco, and F. Saleri. *Matematica Numerica*. Springer - UNITEXT, 2002.
- [25] F. Salinelli, E. e Tomarelli. *Modelli Dinamici Discreti*. Springer - UNITEXT, 2005.
- [26] Shumway. *Applied Statistical Time-Series Analysis*. Prentice-Hall.
- [27] G. Strang and T. Nguyen. *Wavelets and filter banks*. Wellesley-Cambridge Press, 1996.
- [28] ... Tibshirani and ...
- [29] L. Trefethen and J. Bau. *Numerical Linear Algebra*. SIAM, 1997.
- [30] S. Van Huffel and J. Vandewalle. *The Total Least Squares Problem: Computational Aspects and Analysis*. SIAM, 1991.
- [31] P. Van Overschee and B. De Moor. *Subspace identification for linear systems*. Kluwer, 1996.
- [32] C.R. Vogel. *Computational methods for Inverse Problems*. SIAM, 2002.