Università degli Studi di Cagliari

Facoltà di Ingegneria

Corso di Calcolo Numerico 2

Interpolazione e approssimazione di funzioni

Gianfranco Fancellu (L.M. Ingegneria delle Telecomunicazioni) e
Andrea Picciau (L.M. Ingegneria Elettronica)

Indice

Capitolo 1

Introduzione

Tra gli algoritmi dell'analisi numerica più utilizzati dal punto di vista ingegneristico abbiamo quelli di interpolazione e quelli di approssimazione. Nel campo dell'ingegneria elettronica o delle telecomunicazioni, tali algoritmi vengono applicati a segnali campionati, spesso rumorosi, per determinarne l'andamento in istanti di tempo in cui non si hanno informazioni.

In questa tesina esamineremo i principali algoritmi di interpolazione e approssimazione, confrontandone le prestazioni in termine di errore e il comportamento con la funzione seno, la funzione di Runge e l'onda quadra. Si considereranno sia il caso di assenza che quello di presenza di rumore. Inoltre, tali prestazioni saranno valutate sia con la scelta di nodi equispaziati che di nodi di Chebychev.

Gli algoritmi sono stati implementati in `matlab`.

Capitolo 2

Interpolazione

Siano dati $n+1$ punti

$$(x_i, y_i) \quad i = 0, \ldots, n$$

ottenuti mediante il campionamento di un segnale $f(x)$. Vogliamo determinare una funzione $\Phi(x)$ che passi per tali punti e approssimi il meglio possibile $f(x)$.

Scriviamo la *funzione interpolante* $\Phi(x)$ come combinazione lineare di $n+1$ funzioni di base $\varphi(x)$, ossia

$$\Phi(x) = \sum_{j=0}^{n} a_j \varphi_j(x).$$

I coefficienti di tale combinazione sono incognite da determinare, e corrispondono alle soluzioni del sistema

$$\sum_{j=0}^{n} \varphi_j(x_i) \cdot a_j = y_i \quad i = 0, \ldots, n$$

che si ottiene imponendo l'interpolazione dei punti, ossia

$$\Phi(x_i) = y_i \quad i = 0, \ldots, n.$$

La soluzione è unica se la matrice del sistema è non singolare, cioè se il suo determinante, detto *determinante di Haar* è diverso da zero. Se questa condizione è verificata, il sistema prende il nome di *sistema di Chebychev.*

A seconda delle funzioni interpolanti distinguiamo diversi algoritmi; ad esempio nel caso in cui si scelgano polinomi, si parla di *interpolazione polinomiale.* Tali algoritmi forniscono diverse rappresentazioni dello stesso polinomio che interpola i punti dati.
Una volta calcolati i cofficienti a_j bisogna valutare il polinomio. Per fare questo si utilizza l'*algoritmo di Horner*, che riduce il numero di calcoli da eseguire, in quanto la sua complessità computazionale è $O(n)$. Considerando che il polinomio $p_n(x)$ si può riscrivere nel modo seguente

$$p_n(x) = a_0 + (x - x_0)[a_1 + (x - x_1)[a_2 + (x - x_2)[a_3 + ...[a_{n-1} + (x - x_{n-1})a_n]]]]$$

l'algoritmo è il 2.1. Con questo algoritmo sono sufficienti solo n prodotti ed $2n$ somme per ogni valutazione del polinomio, ed è quello utilizzato dalla funzione `matlab polyval`.

Algoritmo 2.1 Algoritmo di Horner

1. $a_n = valore$
2. for $i = n-1 : -1 : 1$
   (a) $valore = valore * (x - x_i) + a_i$

2.1 Interpolazione polinomiale nella base canonica

In questo tipo di interpolazione si sceglie

$$\varphi_j(x) = x^j$$

e quindi risulta

$$\Phi(x) = p_n(x) = \sum_{j=0}^{n} a_j x^j. \tag{2.1}$$

Gli a_j si possono determinare risolvendo il sistema

$$\underbrace{\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}}_{X} \underbrace{\begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}}_{\mathbf{a}} = \underbrace{\begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} \tag{2.2}$$

in cui la matrice X è detta di *Vandermonde.*

L'implementazione in linguaggio `matlab` da noi realizzata nel corso del nostro lavoro è quella del listato 2.1.

Listato 2.1: Interpolazione con i polinomi della base canonica

```
function [f]=canonica(x,y,t,a,b)
% Funzione che calcola il polinomio interpolante usando la comune base
% canonica dei polonomi. Resituisce la valutazione del polinomio nel range
% passato in ingresso.
%
%   input:  x = vettore delle ascisse di interpolazione.
%       y = vettore delle ordinate di interpolazione
%       t = e' il numero di punti in cui si vuole valutare la funzione
%       a,b = sono gli estremi dell'intervallo in cui si vuole
%             l'interpolazione
%
%   output: f = valutazione del polinomio
%

% Costruiamo la matrice di Vandermonde
X = vander(x);

% Risolviamo il sistema Xc = y
c = X\y;
Z = linspace(a,b,t);
f = polyval(c,Z);
```

La rappresentazione (2.1) e il sistema (2.2) tendono ad essere numericamente instabili. La complessità per la valutazione del polinomio interpolante di Lagrange è pari a $O(n^3)$.

2.2 Interpolazione con i polinomi di Lagrange

In questo tipo di interpolazione si sceglie

$$\varphi_j(x) = L_j(x) = \prod_{\substack{k=0 \\ k \neq j}}^{n} \frac{x - x_k}{x_j - x_k}$$

e quindi risulta

$$\Phi(x) = p_n(x) = \sum_{j=0}^{n} y_j L_j(x). \tag{2.3}$$

Gli $L_j(x)$ prendono il nome di *polinomi caratteristici di Lagrange:* ciascuno di essi è di grado n, è sempre ben definito e inoltre verifica la seguente proprietà:

$$L_j(x_i) = \delta_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}.$$

Per questo motivo la matrice Φ è la matrice identità, e quindi i polinomi di Lagrange permettono di ottenere un sistema di Chebychev.

L'implementazione in linguaggio `matlab` da noi realizzata nel corso del nostro lavoro è quella del listato 2.2. La complessità per la valutazione del polinomio interpolante di Lagrange è pari a $O(n^2)$.

Listato 2.2: Interpolazione con i polinomi di Lagrange

```
function [f]=lagrange(x,y,r,a,b)
% Questa funzione permette di calcolare i valori del polinomio interpolante
% di Lagrange.
%
%   input:  x = vettore che contiene le ascisse di interpolazione
%           y = vettore che contiene le ordinate di interpolazione
%           r = numero di punti su cui si vuole valutare il polinomio
%           a,b = estremi dell'intervallo su cui si vuole calcolare il
%                 polinomio interpolante.
%
%   output: f = vettore che contiene le valutazioni del polinomio
%

n = size(x,1);
m = size(y,1);

% Calcolo i punti su cui effettuare l'interpolazione, questi si suppone
% equi-spaziati nell'intervallo dato
t = linspace (a,b,r);

for i = 1:r % calcolo tutte le valutazioni del polinomio interpolante
    P = 0;
    for j = 1:n
        L = 1;
        for k = 1:n
            if(k ~= j)
                L = L*((t(i)-x(k))/(x(j)-x(k)));
            end
        end
        P = P + (y(j)*L);
    end
    f(i) = P;
end
```

2.3 Formula di Neville

A differenza degli altri algoritmi questa formula consente di calcolare il valore del polinomio interpolante in un punto senza doverlo costruire in modo esplicito. Tale punto può anche essere esterno all'intervallo dei nodi d'interpolazione: in quel caso si parla di *estrapolazione* .

La formula di Neville è un algoritmo di tipo ricorsivo, infatti consente di calcolare il polinomio di grado k

$$Q_{r,r+1,\ldots,r+k}(x)$$

che interpola nelle ascisse $x_r, x_{r+1}, \ldots, x_{r+k}$, in funzione di due polinomi di grado $k-1$ che interpolano rispettivamente nelle ascisse $x_r, x_{r+1}, ..., x_{r+k-1}$ed $x_{r+1}, x_{r+2}, ..., x_{r+k}$. La formula è la seguente:

$$Q_{r,r+1,\ldots,r+k}(x) = \frac{(x - x_r)Q_{r+1,r+2,\ldots,r+k}(x) - (x - x_{r+k})Q_{r,r+1,\ldots,r+k-1}(x)}{x_{r+k} - x_r}. \tag{2.4}$$

Si può dimostrare che tale polinomio interpola la funzione in tutti i $k+1$ punti e quindi vale:

$$Q_{r,r+1,...,r+k}(x_i) = y_i \quad i = r, r+1, ..., r+k.$$

La condizione di terminazione sono i polinomi di grado zero che interpolano in un punto quindi $Q_r(x) = y_r$, e partendo da questi si possono costruire polinomi di qualunque grado. Per la valutazione della funzione in un punto si sostituisce la variabile x con il punto richiesto e si calcola lo schema di Neville.

Nella nostra implementazione (listato 2.3), questo meccanismo è stato realizzato creando una matrice triangolare inferiore in cui la prima colonna sono le ordinate d'interpolazione ed a ogni passo gli elementi di una colonna sono calcolati in funzione degli elementi della colonna precedente usando la formula (2.4). Terminato l'algoritmo il valore del polinomio si trova nel vertice destro della matrice triangolare, quindi nell'elemento $Q(n,n)$.

Listato 2.3: Interpolazione con la formula di Neville

```
function [q]=formulaneville(x,y,z)
% Questa funzione calcola il valore del polinomio interpolante nel punto z
% infatti la formula di Neville è utile per calcolare la funzione in un
% punto, appliccando più volte l'algoritmo sarà possibile determinare il
% valore della funzione in più punti.
%   input:  x = ascisse di interpolazione
%           y = ordinate di interpolazione
%           z = punto in cui si vuole valutare la funzione
%
%   output: q = valore del polinomio di Neville interpolante nel punto z
%

% Bisogna costruire la matrice al cui interno ci sono i coefficienti di
% Neville

n=size(x,1);

Q=zeros(n,n);
Q(:,1)=y;    % inizializzo la prima colonna della matrice
for j=2:n
    for i=n:-1:j
        Q(i,j)=(((z-x(i-j+1))*Q(i,j-1))-((z-x(i))*Q(i-1,j-1)))/(x(i)-x(i-j+1));
        %calcolo il generico coefficcente di neville
    end
end
q=Q(n,n);
```

Abbiamo effetuato alcune prove per verificare l'errore che si commette sia nel caso di interpolazione che nel caso di estrapolazione, ottenendo i dati riassunti nella tabella 2.1.
Consideriamo le diverse prove effettuate:

1. Come ci si poteva aspettare l'errore è molto piccolo, confrontabile con la precisione dei numeri di macchina. Questo perchè il punto $x = 1$ è un'ascissa d'interpolazione.

2. In questo caso l'errore che si misura è 0.0103 quindi come ci si poteva aspettare è piccolo. Questo perchè il punto $x = 0.5$ è interno all'intervallo delle ascisse d'interpolazione.

3. L'errore che si misura è 3.6138 quindi come ci si poteva aspettare l'errore è grande. Questo perchè il punto x=4 è esterno all'intervallo delle ascisse d'interpolazione: si tratta di un problema di estrapolazione.

Tabella 2.1: Verifiche sulla formula di Neville

Prova	1	2	3
funzione $f(x)$	$\sin(x)$		
ascisse di interpolazione	equispaziate in $[-2:2]$		
grado del polinomio interpolante n	4		
ascissa del punto di verifica	1	0.5	4
errore	$1.1 * 10^{-16}$	0.0103	3.6138

2.4 Polinomio interpolante di Newton

Come nel caso dell'algoritmo di Neville anche questo è un algoritmo ricorsivo: infatti permette di calcolare il polinomio di grado n che interpola in $n+1$ punti in funzione del polinomio di grado n che interpola in n punti. In questo modo si riescono a sfruttare i risultati ottenuti in precedenza. La condizione d'arresto è il polinomio di grado 0 che interpola in un solo punto $P_0(x) = x_0$.

Il polinomio $p_n(x)$ può essere scomposto in questo modo:

$$p_n(x) = p_{n-1}(x) + g_n(x) \tag{2.5}$$

dove $g_n(x)$ è un polinomio di grado n, della forma

$$g_n(x) = a_n \cdot \omega_{n-1}(x) = a_n \cdot \prod_{i=0}^{n-1}(x - x_i) \tag{2.6}$$

in questo modo il nuovo polinomio interpola nelle stesse ascisse in cui interpola il polinomio di grado inferiore $(x_o, x_1, ..., x_{n-1})$.
Imponendo la condizione d'interpolazione si calcola il coefficiente a_n, quindi si trova:

$$a_n = \frac{y_n - p_{n-1}(x_n)}{\omega_{n-1}(x_n)}. \tag{2.7}$$

In realtà per il calcolo dei cofficienti si preferisce utilizzare un'altra formula che garantisce una stabilità maggiore ed una complessità computazionale inferiore. Per convenzione indichiamo

$$a_n = f[x_0, x_1, ..., x_n]$$

con il nome di *differenza divisa*. Per calcolare questo valore si può usare la formula di Neville. Si consideri il polinomio:

$$Q_{0,1,...,n}(x) = \frac{(x - x_0)Q_{1,2,...,n}(x) - (x - x_n)Q_{0,1,...,n-1}}{x_n - x_0}.$$

a_nè il coefficiente di grado massimo di tale polinomio, essendo $\omega_{n-1}(x)$ un polinomio monico, quindi si può ricavare la formula ricorsiva per il calcolo delle differenze divise. Questa è molto simile alla formula di Neville: la differenza divisa di ordine n si calcola in funzione delle differenze divise di ordine $n-1$, e la condizione d'arresto è la differenza divisa di ordine zero che vale $f[x_i] = y_i$ per $i = 0, 1, ..., n$. La formula generale è:

$$f[x_0, x_1, ..., x_k] = \frac{f[x_1, x_2, ..., x_k] - f[x_0, x_1, ..., x_{k-1}]}{x_k - x_0} \quad k = 1, 2, ..., n. \tag{2.8}$$

Il polinomio risultante è nella forma

$$p_n(x) = f[x_0] + f[x_0, x_1]\omega_0(x) + f[x_0, x_1, x_2]\omega_1(x) + ... + f[x_0, x_1, x_2, ..., x_n]\omega_{n-1}(x). \tag{2.9}$$

I coefficienti di questo polinomio sono gli elementi della diagonale principale dello schema delle differenze divise. Ancora, questo è simile allo schema di Neville visto nel paragrafo precedente: è una matrice triangolare inferiore in cui la prima colonna è inizializzata con le ordinate d'interpolazione e successivamente gli elementi vengono calcolati con la formula (2.8).

Il codice `matlab` per il calcolo del polinomio interpolante di Newton quello del listato 2.4.

Listato 2.4: Interpolazione con la formula di Newton

```
function [f]=newton2(x,y,r,a,b)
% Questo algorotimo calcola il polinomio interpolante di Newton ma a
% la valutazione del polinomio la si effettua usando l'algoritmo di
% Horner, ed le differenze divise si calcolano in
% un unica matrice sfruttando il fatto che questi coefficenti
% sono gli elementi della diagonale principale della matrice.
%
%   input:  x = ascisse di interpolazione
%           y = ordinate di interpolazione
%           r = numero di punti in cui si vuole valutare
%               il polinomio interpolante
%           a,b = estremi dell'intervallo di valutazione
%
%   output: f = valutazione del polinomio nell'intervallo
%               richiesto
%

n=size(x,1);

Q=zeros(n,n);
Q(:,1)=y;    % inizializzo la prima colonna della matrice
for j=2:n
    for i=n:-1:j
        Q(i,j)=((Q(i,j-1)-Q(i-1,j-1)))/(x(i)-x(i-j+1));
        %calcolo il generico coefficcente di neville
    end
end

t=linspace(a,b,r);
for k=1:r
    valore=Q(n,n);
    for j=(n-1):-1:1
        valore=valore*(t(k)-x(j))+Q(j,j);
    end
    f(k)=valore;
end
```

2.5 Nodi di Chebychev

Come si vedrà dalle prove sperimentali la scelta dei nodi gioca un ruolo fondamentale nell'errore.

In moltissimi casi la scelta migliore è quella dei nodi di Chebychev. Infatti, il teorema di Bernstein afferma che se le ascisse di interpolazione sono radici del polinomio di Chebychev l'errore tende a zero per n che tende all'infinito, se la funzione $f(x)$ è derivabile nell'intervallo$[a, b]$.

Il polinomio di Chebychev di grado $n + 1$ è il seguente:

$$T_{n+1} = \cos((n + 1)\theta)$$

per $x = \cos(\theta)$ e $\theta \in [0, \pi]$. Imponendo questo uguale a zero si trovano le radici le $n+1$ che saranno: $x_k = \cos\left(\frac{2k+1}{2n+2}\pi\right)$ per $k = 0, 1, ..., n$. Le radici saranno comprese nell'intervallo $[-1, 1]$ quindi per generalizzarlo per qualsiasi intervallo si effettua il cambio di variabili $t_k = \frac{b-a}{2}x_k + \frac{a+b}{2}$.

Per il nostro lavoro di tesina abbiamo implementato una funzione `matlab` che calcola la posizione dei nodi equispaziati o di Chebychev a seconda degli ingressi. Il suo listato è il 2.5.

Listato 2.5: Funzione per il calcolo dei nodi

```
% CREANODI [x] = creaNodi(X,n,mode)
% dato un vettore di ascisse X restituisce le ascisse di n+1 nodi.
% Gli estremi del vettore x coincidono con gli estremi del vettore X in modo
% che non ci sia estrapolazione.
%
%   input:  X = vettore di ascisse
%           n = numero di nodi che si vuole ottenere
%           mode = puo' essere 'equispaziati' o 'chebychev'
%
%   output: x = vettore contenente le ascisse dei nodi
%

function [x] = creaNodi(X,n,mode)
    if (strcmp('equispaziati',mode))
        m = length(X);
        x= linspace(X(1),X(m),n+1)';      %approssima all'intero piu' vicino
    else if (strcmp('chebychev',mode))
        a=X(1);
        b=X(length(X));
        k=(0:n)';
        w=cos((((2*k)+1)*pi)/((2*n)+2));      % calcolo gli zeri del polinomio nell'
            intervallo -1,1
        x=((b-a)*w/2)+((b+a)/2);      % faccio il cambio di variabili
        end
end
```

2.6 Prove sperimentali

Tutte le implementazioni degli algoritmi realizzate, restituiscono un vettore con le valutazioni del polinomio interpolante nell'intervallo richiesto. Questo intervallo può essere lo stesso delle ascisse d'interpolazione oppure può essere diverso: in questo modo si può effettuare estrapolazione.

Gli algoritmi sono stati testati utilizzando campioni della funzione seno, della funzione di Runge e dell'onda quadra: in questo modo si è voluto mettere in luce il comportamento per funzioni molto particolari (Runge) ma anche per segnali tipici delle telecomunicazioni (onda quadra).

2.6.1 Segnali non rumorosi

Sono state effettuate alcune prove sui diversi algoritmi per valutare l'errore ed i tempi d'esecuzione per diverse funzioni (campionate uniformemente nell'intervallo $[-4, 4]$), usando come scelta dei nodi sia quelli equispaziati che quelli di Chebychev. I risultati sono riportati nella tabella 2.2.

Tabella 2.2: Prove sperimentali con segnali non rumorosi
Grado del polinomio interpolante $n = 10$

Nodi	Funzione $f(x)$	Algoritmo	t_c $[s]$[1]	e_{max}[2]	$\bar{e}$[3]	σ^2[4]
equispaziati	$\sin(x)$	Base canonica	4.96e-04	6.51e-04	1.36e-16	3.75e-08
		Lagrange	1.22e-02		7.69e-17	
		Newton	4.02e-03		9.64e-16	
	$\frac{1}{1+25x^2}$	Base canonica	5.67e-04	5.60e+00	6.01e-01	2.56e+00
		Lagrange	1.23e-02			
		Newton	4.20e-03			
	$square(x)$[5]	Base canonica	4.02e-04	8.66e+00	7.13e-01	3.85e+00
		Lagrange	1.32e-02			
		Newton	4.47e-03			
Chebychev	$\sin(x)$	Base canonica	4.86e-04	7.71e-05	1.58e-17	2.78e-09
		Lagrange	1.33e-02		3.94e-17	
		Newton	4.39e-03		-1.01e-15	
	$\frac{1}{1+25x^2}$	Base canonica	4.22e-04	6.18e-01	7.90e-02	5.07e-02
		Lagrange	1.28e-02			
		Newton	4.63e-03			
	$square(x)$	Base canonica	4.43e-04	2.00e+00	1.43e-01	3.33e-01
		Lagrange	1.31e-02			
		Newton	4.38e-03,			

Dai risultati si nota come in generale, i tempi di esecuzione dell'algoritmo di interpolazione nella base canonica siano paragonabili a quelli degli altri algoritmi, mentre ci aspetteremmo essere maggiori. Questo accade presumibilmente perchè nell'implementazione dell'algoritmo (listato 2.1 a pagina 4) si sono utilizzate solo funzioni built-in di `matlab`, particolarmente ottimizzate.
Inoltre l'errore è condizionato soprattutto dalla scelta dei nodi. Si può osservare che l'errore (medio o massimo) che si commette usando i nodi equispaziati è circa un ordine di grandezza maggiore rispetto al caso dei nodi di Chebychev. Analoghe considerazioni si possono effettuare osservando la varianza, che nel secondo caso è minore di circa un ordine di grandezza.
Se consideriamo i casi in cui la funzione $f(x)$ è l'onda quadra, osserviamo che l'errore è maggiore che con le altre funzioni. Questo perchè l'onda quadra è una funzione discontinua, mentre il polinomio interpolante è una funzione continua e derivabile infinite volte.

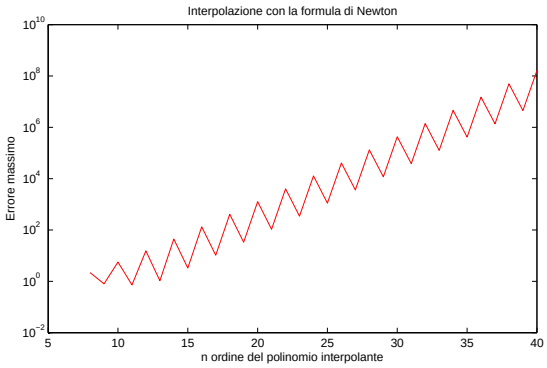
Supponiamo ora di variare l'ordine del polinomio e consideriamo l'errore massimo che si commette. Nel caso dell'algoritmo di Newton otteniamo la figura 2.1. In entrambi i grafici si è utilizzata la scala logaritmica per le ordinate. Possiamo notare come con la scelta dei nodi equispaziati l'errore aumenti esponenzialmente (abbiamo quasi una retta), mentre con la scelta dei nodi di Chebychev l'errore decresca. Con gli altri algoritmi si ottengono risultati simili.

Nel caso dell'algoritmo di interpolazione con i polinomi in base canonica, `matlab` segnala tramite una serie di warning che la matrice di Vandermonde risulta estremamente malcondizionata. Questo è uno dei principali problemi che si riscontra con questo algoritmo, come già anticipato nella sezione 2.1. Si osservi la figura 2.2: tale figura è stata ottenuta facendo variare l'ordine del polinomio interpolante per l'onda quadra. I warning compaiono da $n = 22$ in poi. Si noti che l'errore massimo nel caso (a) tende ancora a crescere in maniera esponenziale, mentre il comportamento nel caso (b) è molto diverso dal caso precedente. Si osservano infatti una serie di oscillazioni dall'andamento fortemente irregolare e in generale il valore dell'errore massimo è più alto rispetto a quanto accade quando si utilizza l'algoritmo di Newton (figura 2.1).

[1] Tempo di calcolo.
[2] Errore massimo.
[3] Errore medio.
[4] Varianza dell'errore.
[5] Onda quadra con pulsazione 2π.

Figura 2.1: Variazione dell'errore con l'ordine del polinomio. Algoritmo di Newton.

(a) Nodi equispaziati



(b) Nodi di Chebychev

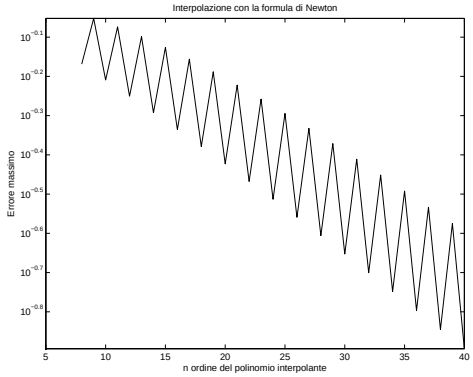
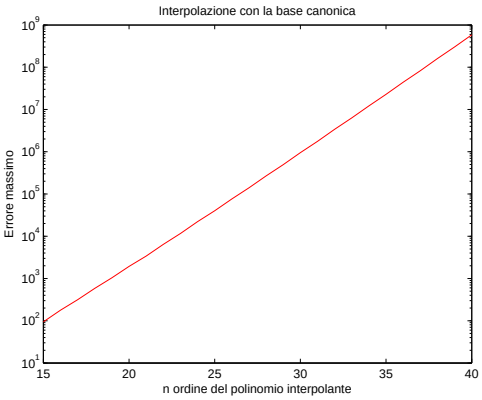
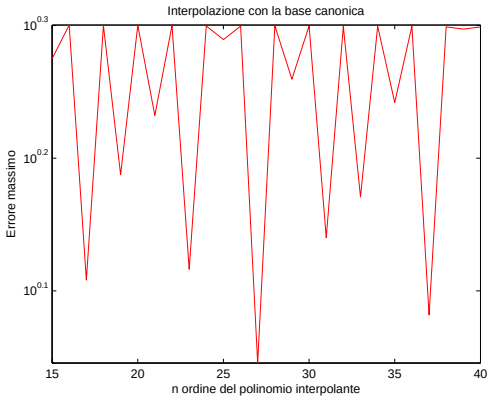


Figura 2.2: Variazione dell'errore con l'ordine del polinomio. Algoritmo di interpolazione nella base canonica.

(a) Nodi equispaziati



(b) Nodi di Chebychev



2.6.2 Segnali rumorosi

Consideriamo il caso di un segnale rumoroso in cui alla funzione "pulita" viene sommato del rumore. Questo è un caso tipico nelle applicazioni: ad esempio un segnale elettrico può essere disturbato da rumore termico, di Schottky, burst noise etc. Si è considerato come rumore quello gaussiano bianco la cui media è nulla, un buon modello per molti tipi di rumore, in particolare per quello termico.

Siamo interessati a vedere come l'errore sui dati iniziali influisca sul risultato dell'interpolazione. Sono state eseguite prove sulle stesse funzioni del caso precedente (sempre campionate uniformemente nell'intervallo $[-4, 4]$) i cui risultati sono schematizzati nella tabella 2.3.

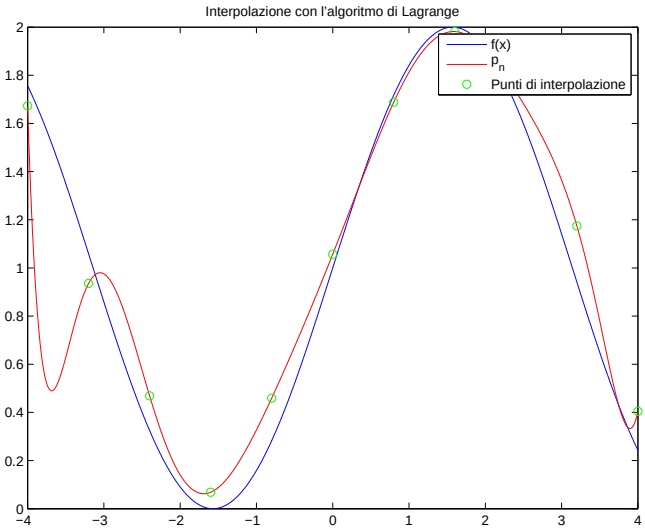
Tabella 2.3: Prove sperimentali con segnali rumorosi
Grado del polinomio interpolante $n = 10$

Nodi	Funzione $f(x)$	Algoritmo	t_c $[s]$	e_{max}	$\bar{e}$	σ^2
equispaziati	$\sin(x)$	Base canonica	4.01e-03	3.70e-01	-4.55e-02	8.00e-03
		Lagrange	1.52e-02			
		Newton	5.48e-03			
	$\frac{1}{1+25x^2}$	Base canonica	4.28e-04	5.60e+00	5.95e-01	2.52e+00
		Lagrange	1.35e-02			
		Newton	4.85e-03			
	$square(x)$	Base canonica	3.87e-04	9.61e+00	7.86e-01	5.04e+00
		Lagrange	1.45e-02			
		Newton	4.15e-03			
Chebychev	$\sin(x)$	Base canonica	4.66e-04	2.19e-01	1.15e-02	9.16e-03
		Lagrange	1.32e-02			
		Newton	4.34e-03			
	$\frac{1}{1+25x^2}$	Base canonica	4.18e-04	6.32e-01	8.43e-02	5.22e-02
		Lagrange	1.32e-02			
		Newton	4.15e-03			
	$square(x)$	Base canonica	3.85e-04	1.89e+00	1.04e-01	3.25e-01
		Lagrange	1.35e-02			
		Newton	4.25e-03			

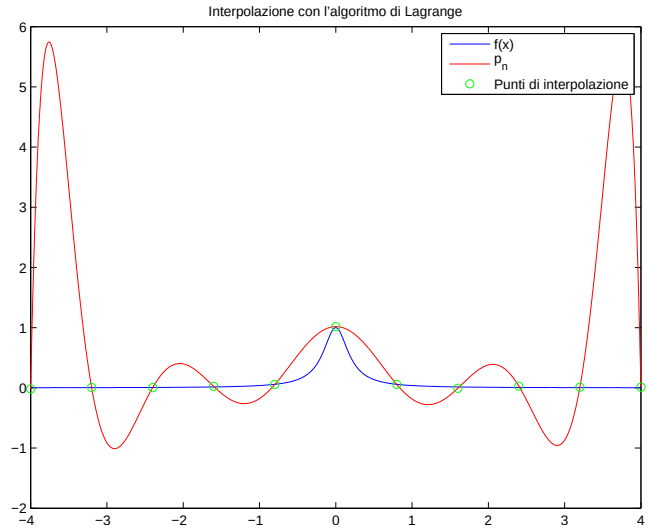
Si può osservare che nel caso della funzione seno l'errore peggiora notevolmente rispetto al caso privo di rumore, invece per le altre due funzioni l'errore rimane dello stesso ordine di grandezza. La vera differenza la gioca la scelta dei nodi, infatti nel caso di nodi equispaziati (figura 2.3) l'errore è almeno un ordine di grandezza superiore di quello che si commette usando i nodi di Chebychev (figura 2.4).

Figura 2.3: Interpolazione di segnali rumorosi con nodi equispaziati

(a) $f(x) = \sin(x)$



(b) $f(x) = \frac{1}{1+25x^2}$



(c) $f(x) = square(x)$

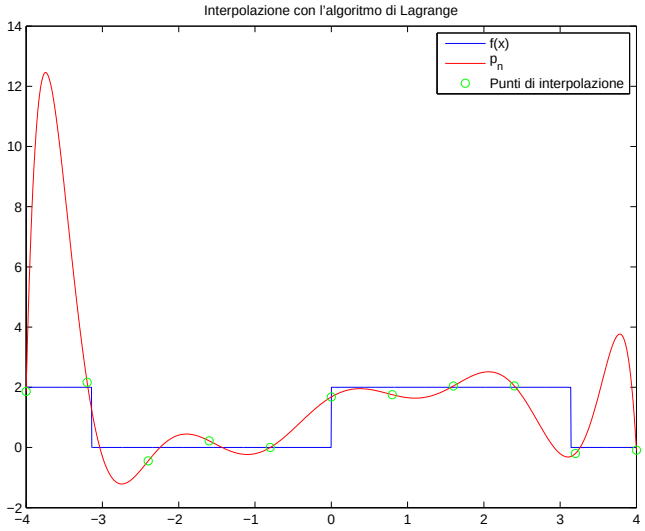
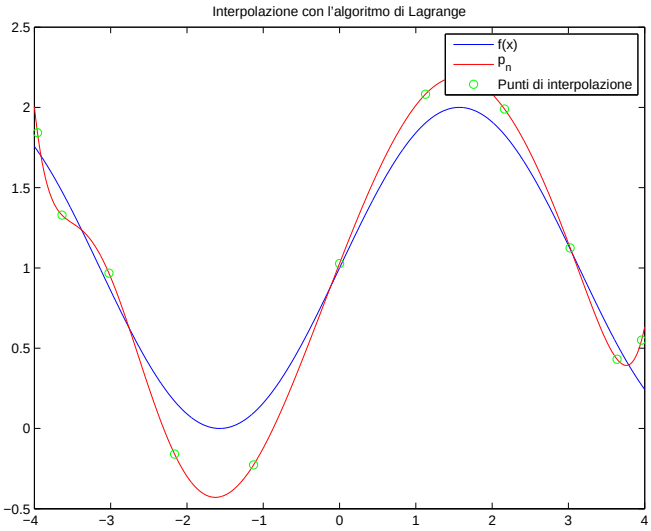
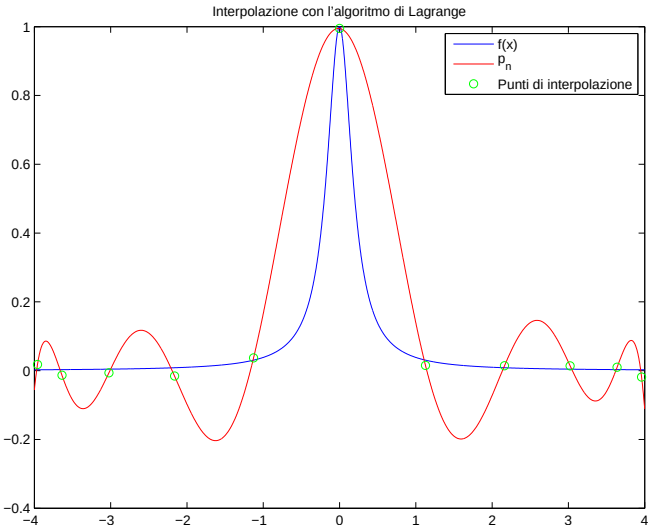


Figura 2.4: Interpolazione di segnali rumorosi con nodi di Chebychev

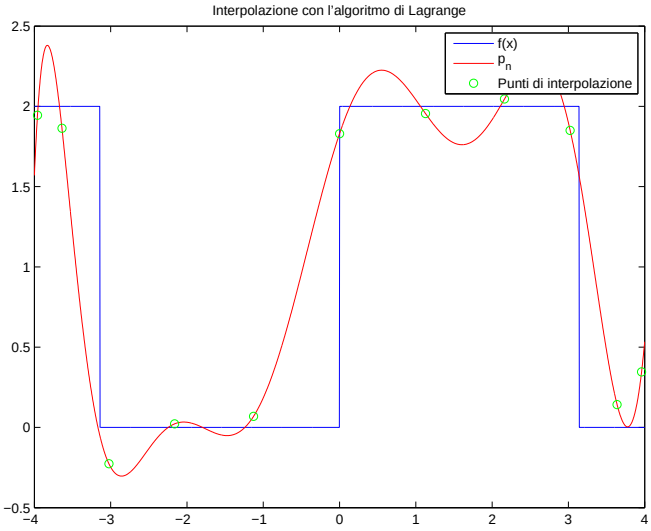
(a) $f(x) = \sin(x)$



(b) $f(x) = \frac{1}{1+25x^2}$



(c) $f(x) = square(x)$

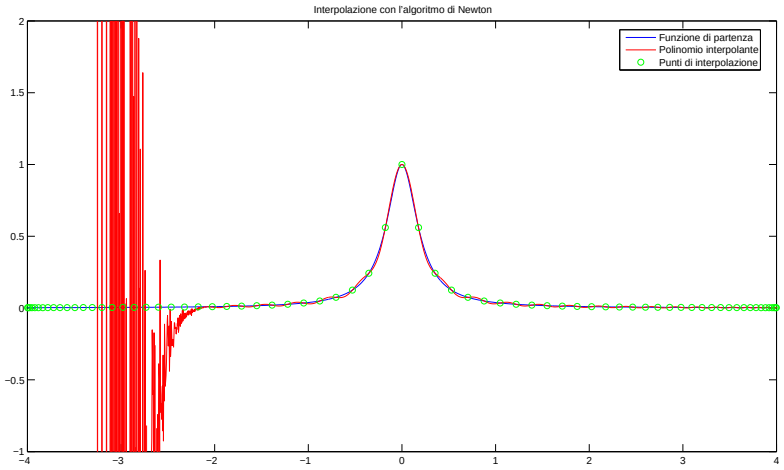


2.6.3 Comportamento *particolare* dell'algoritmo di Newton

Durante le prove si è osservato un comportamento particolare dell'algoritmo di Newton: all'aumentare del grado del polinomio l'errore aumenta in modo considerevole superato un certo valore anche se si utilizzano nodi di Chebychev. Questo va in contrasto con la teoria per cui l'errore dovrebbe tendere a zero all'aumentare del grado del polinomio interpolante quando si utilizza questa particolare scelta di nodi.

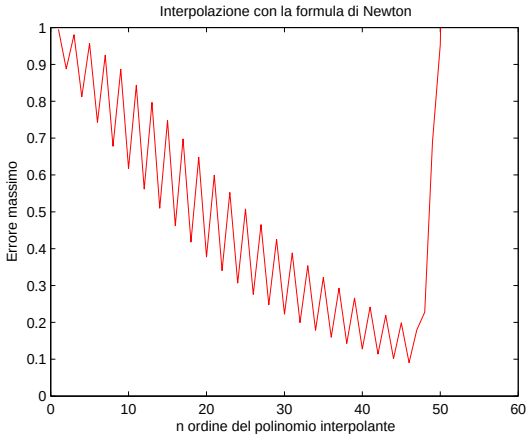
Come si può osservare dalla figura 2.5 l'errore è elevato, inoltre il polinomio non è più interpolante, infatti alcuni campioni a sinistra dell'intervallo non vengono interpolati. Un altra osservazione interessante è l'assimetria: nella parte destra dell'intervallo il polinomio interpola nel modo corretto. In questo esempio il grado del polinomio è 70 ed la scelta dei nodi è quella di Chebychev, la funzione da interpolare è quella di Runge senza rumore. Questo fenomeno invece non si osserva utilizzando l'algoritmo di Lagrange, che in questo caso risulata molto più stabile.

Figura 2.5: Anomalia algoritmo di Newton



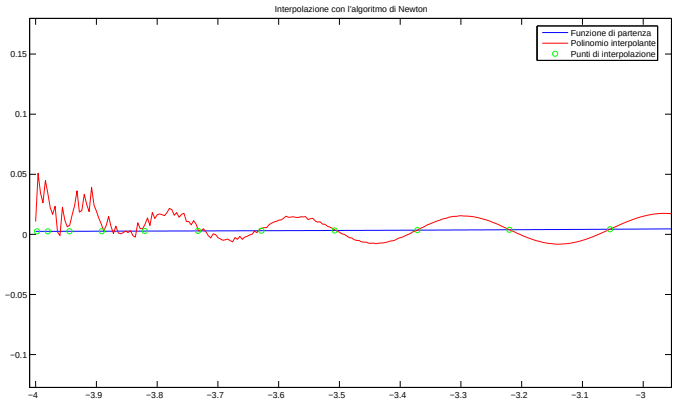
Variando il grado del polinomio, siamo interessati a valutare come varia l'errore. I risultati riportati in figura 2.6 (ottenuti usando come funzione $f(x)$ la funzione di Runge senza rumore, intervallo d'interpolazione [-4 4] e con scelta di nodi di Chebychev) mostrano che l'errore diminuisce fino a $n = 46$ (come è prevedibile dalla teoria, infatti usando nodi di Chebychev l'errore deve tendere a zero), invece per valori maggiori inizia a crescere molto rappidamante.

Figura 2.6: Variazione dell'errore con l'ordine del polinomio usando l'algoritmo di Newton.



La figura2.7 è un particolare della funzione di Runge e del suo polinomio interpolante. Questa mostra che già per $n = 46$ il polinomio si comporta in modo anomalo e non interpola i tre campioni più a sinistra, inoltre l'andamento è molto irregolare.

Figura 2.7: Particolare polinomio interpolante P_n con $n = 46$



2.6.4 Confronto con le funzioni interpft e interp1

Altri algoritmi oltre a quelli da noi implementati sono quelli di interpolazione trigonometrica (funzione `interpft` di `matlab`), di interpolazione *spline* (funzione `interp1` di `matlab`).

La funzione `interpft` calcola i valori del polinomio trigonometrico, i cui coefficienti sono quelli di Fourier, sfruttando la FFT (*Fast Fourier Transform*). Tale base è più stabile rispetto ad altre, in quanto i vettori che la costituiscono sono tra loro ortogonali. La funzione richiede in ingresso un vettore di nodi necessariamente equispaziati.

La funzione `interp1` effettua un'interpolazione polinomiale a tratti, cioè interpola due campioni successivi con un polinomio di grado relativamente basso. Le opzioni che abbiamo utilizzato sono quella lineare e quella cubica (chiamata `SPLINE`). Nel caso di interpolazione lineare, due punti consecutivi vengono uniti tramite un segmento di retta, mentre nella spline cubica sono uniti tramite un polinomio di terzo grado (immagine 2.8). Vengono inoltre imposte condizioni di raccordo fino alla derivata seconda. A differenza degli altri polinomi interpolanti (Lagrange, Newton, etc.) la funzione interpolante dunque non è derivabile infinite volte.

Figura 2.8: Interpolazione *spline*

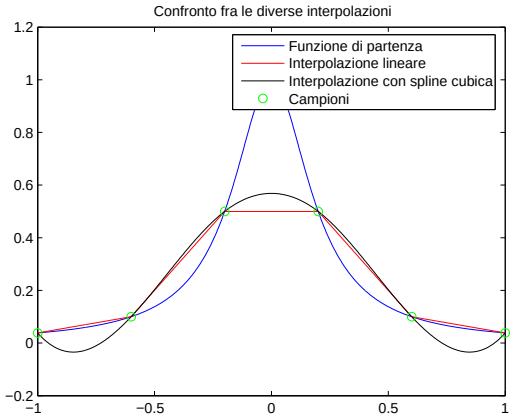


Tabella 2.4: Prove sperimentali con le funzioni *interpft* e *interp1*
Grado del polinomio $n = 10$

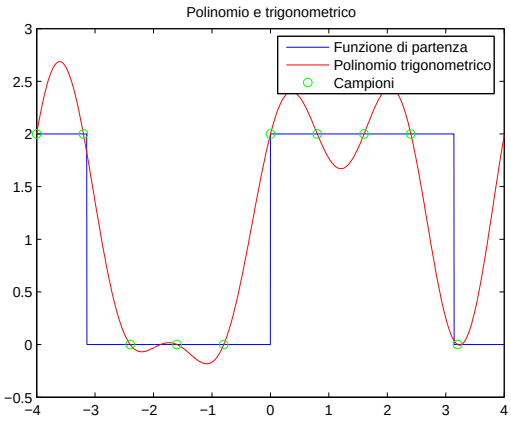
Tipo di segnale	Funzione $f(x)$	Algoritmo	e_{max}	$\bar{e}$	σ^2
senza rumore	$\sin(x)$	interpft	1.51e+00	1.58e-01	7.39e-02
		interp1 (linear)	9.51e-02	7.54e-04	2.97e-07
		interp1 (spline)			
	$\frac{1}{1+25x^2}$	interpft	7.93e-01	4.39e-02	1.76e-02
		interp1 (linear)	8.31e-01	4.63e-02	1.81e-02
		interp1 (spline)			
	$square(x)$	interpft	2.18e+00	4.75e-01	4.40e-01
		interp1 (linear)	1.93e+00	3.75e-01	2.99e-01
		interp1 (spline)			
con rumore	$\sin(x)$	interpft	1.42e+00	1.39e-01	7.72e-02
		interp1 (linear)	2.52e-01	9.98e-02	7.02e-03
		interp1 (spline)			
	$\frac{1}{1+25x^2}$	interpft	7.94e-01	4.84e-02	1.74e-02
		interp1 (linear)	8.39e-01	4.34e-02	1.85e-02
		interp1 (spline)			
	$square(x)$	interpft	2.30e+00	5.23e-01	4.40e-01
		interp1 (linear)	1.98e+00	3.75e-01	2.85e-01
		interp1 (spline)			

Si osservino i dati della tabella 2.4, e li si confronti con i dati nelle tabelle precedenti, nelle righe riferite alla scelta dei nodi equispaziati. Nelle tabelle in esame, il numero dei campioni è sempre $n = 10$.
Possiamo osservare come nel caso della funzione seno in assenza di rumore (cfr. tabelle 2.4 e 2.2) le prestazioni delle funzioni `interpft` e `interp1` siano molto peggiori rispetto alle altre. Nel caso invece della funzione di Runge e dell'onda quadra, le prestazioni di questi algoritmi sono decisamente migliori rispetto alle altre. Ciò può essere dovuto alle particolari caratteristiche di queste funzioni.
Si consideri ora il caso di segnali rumorosi (cfr. tabelle 2.4 e 2.2). Le prestazioni in termini di errore in questo caso sono confrontabili. Nel dettaglio, per la funzione seno l'errore è minore con le funzioni da noi implementate rispetto a quelle di `matlab`, viceversa negli altri casi.

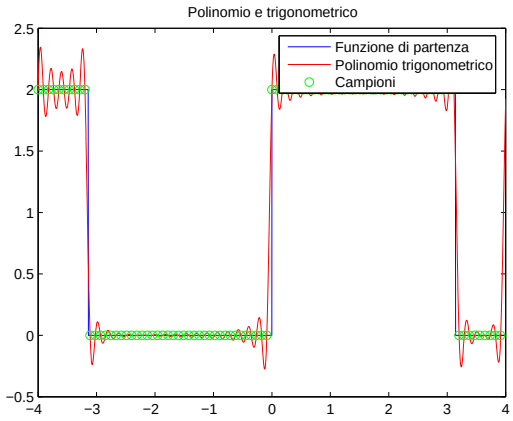
Si può osservare che al crescere dei valori di n l'errore che si commette nel caso di assenza di rumore utilizzando la funzione `interpft` tende a zero. Ciò è illustrato in figura 2.9. Questo comportamento è differente da quanto accade con gli altri algoritmi scegliendo nodi equispaziati, come è indicato dalla figura 2.1 a pagina 11, con cui l'errore cresce.
Un aspetto interessante da osservare dell'interpolazione trigonometrica sta nei nodi che vengono forniti all'algoritmo. La funzione si aspetta in ingresso i campioni relativi a un periodo del segnale, supponendo che questo sia periodico. La funzione interpolante risulta pertanto sempre periodica con periodo pari all'ampiezza della finestra di osservazione. Quindi, se ciò si verifica, la qualità dell'interpolazione è buona, mentre se questo non avviene, come mostra la figura 2.10 a pagina 19 l'andamento del polinomio interpolante si discosta da quello della funzione di partenza. Questo comportamento è collegato allo spettro del segnale, visto che la funzione utilizza la FFT.

Figura 2.9: Comportamento del polinomio trigonometrico al variare di n in assenza di rumore

(a) $n = 11$



(b) $n = 91$



(c) Variazione dell'errore medio in funzione di n

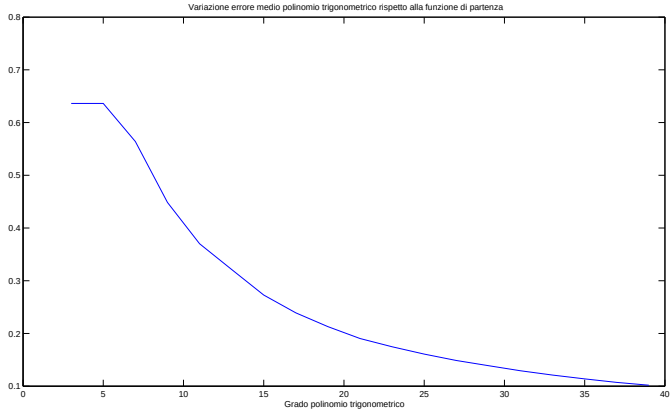
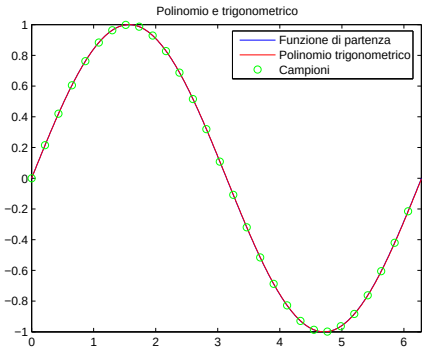
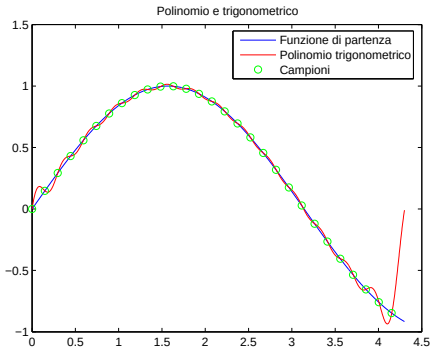


Figura 2.10: Campionamento coerente e incoerente

(a) Campionamento coerente



(b) Campionamento incoerente



Capitolo 3

Approssimazione

Nel caso in cui i dati a disposizione siano affetti da un errore considerevole, l'interpolazione risulta sconveniente. Infatti si imporrebbe al polinomio interpolante il passaggio per punti che sappiamo essere sbagliati. La migliore alternativa è quindi quella di determinare il polinomio di migliore approssimazione, ossia quello che verifica la relazione:

$$\min_{p_n \in \Pi_n} ||p_n - f||. \tag{3.1}$$

3.1 Approssimazione nel senso dei minimi quadrati

Si parla di *migliore approssimazione uniforme* nel caso in cui si utilizza la norma infinito. Utilizzare tale norma risulta in realtà molto difficile, e quindi si preferiai minimi quadratisce utilizzare la norma 2: in questo caso si parla di *migliore approssimazione nel senso dei minimi quadrati*, e si minimizza la varianza dell'errore.

Supponiamo che n sia l'ordine del polinomio approssimante e m sia il numero di punti a disposizione. Se $n = m$ abbiamo un problema di interpolazione, già affrontato nel capitolo precedente. Se invece $n < m$ abbiamo un problema di approssimazione.

Utilizzando la norma 2 abbiamo:

$$||p_n - f||_2^2 = \int_a^b \left((f(x) - p_n(x)\right)^2 dx \approx \sum_{j=0}^{m} \left(f(x_j) - p_n(x_j)\right)^2 \tag{3.2}$$

dove si è approssimato l'integrale con una sommatoria. Inoltre non abbiamo considerato il dx in quanto, essendo una costante, non influsce sul minimo. Usando come base approssimante del polinomio quella canonica, si trova la seguente espressione:

$$p_n(x_i) = \sum_{j=0}^{n} a_j x_i^j = (X\mathbf{a})_i \quad i = 1, \ldots, m. \tag{3.3}$$

dove X è la matrice di Vandermonde già vista nell'equazione (2.2), ma di dimensione $(m+1)\times(n+1)$

$$\begin{bmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_m & x_m^2 & \cdots & x_m^n \end{bmatrix}.$$

Sostituendo la (3.3) nella (3.2) otteniamo

$$||p_n - f||_2^2 = ||X\mathbf{a} - \mathbf{y}||_2^2$$

quindi il problema di approssimazione si riconduce alla soluzione di un sistema rettangolare $X\mathbf{a} = \mathbf{y}$, ossia un sistema sovradeterminato.

L'implementazione in linguaggio `matlab` da noi realizzata è quella del listato 3.1 .

Listato 3.1: Approssimazione nel senso dei minimi quadrati

```
% MINIMIQUADRATI [f] = minimiquadrati(x,y,n,numPunti,a,b)
% Restituisce i valori del polinomio che approssima nel senso dei minimi
% quadrati la funzione campionata nelle ascisse x con valori y.
%
%      input:  x = ascisse dei campioni
%              y = valori dei campioni
%              n = ordine del polinomio di approssimazione
%              numPunti = numero di punti in cui si vuole valutare la funzione
%              a,b = estremi dell'intervallo in cui si vuole effettuare l'ap
%                    prossimazione.
%
%     output:  f = valori del polinomio di approssimazione nell'intervallo
%                  desiderato
%
%

function [f] = minimiquadrati(x,y,n,numPunti,a,b)
    m = length(x);

    % calcolo della matrice di Vandermonde
    X = ones(m,n);
    for i = 1:n-1;
        X(:,i)=x.^(n-i);
    end

    % calcolo dei coefficienti del polinomio
    [Q,R] = qr(X);
    R1 = R(1:n,:);
    b1=Q'*y;
    b1=b1(1:n);
    coeff = R1\b1;

    t = linspace(a,b,numPunti);
    f = polyval(coeff,t);
end
```

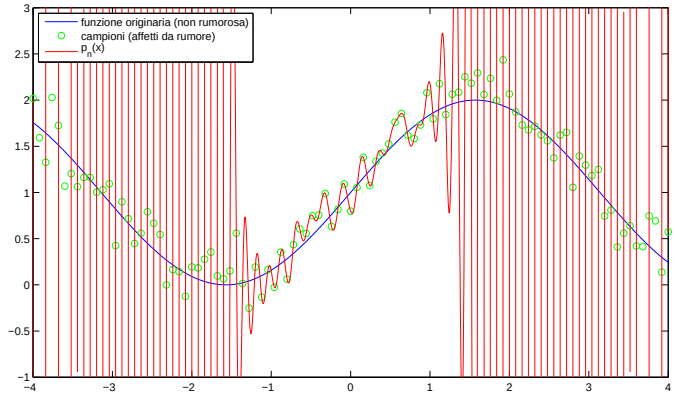
3.2 Prove sperimentali

3.2.1 Confronto tra interpolazione e approssimazione

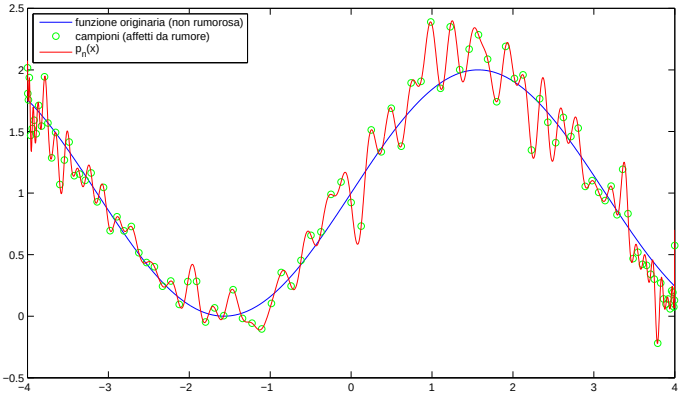
Le prove sperimentali da noi eseguite vogliono mostrare i vantaggi nell'utilizzo degli algoritmi di approssimazione rispetto a quelli di interpolazione nel caso in cui i dati siano affetti da rumore. Si è utilizzata come funzione di partenza $f(x) = \sin(x) + 1$: a questa funzione si è sommato rumore gaussiano bianco. Si è fissato il numero di campioni a disposizione $m = 100$.

Figura 3.1: Confronto tra approssimazione e interpolazione

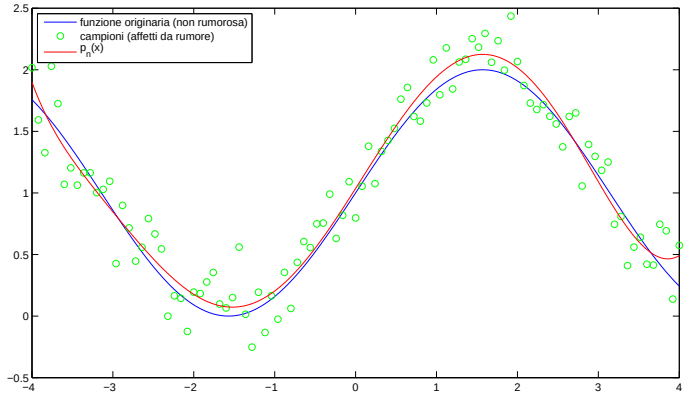
(a) Interpolazione con Lagrange (nodi equispaziati)



(b) Interpolazione con Lagrange (nodi di Chebychev)



(c) Approssimazione ai minimi quadrati



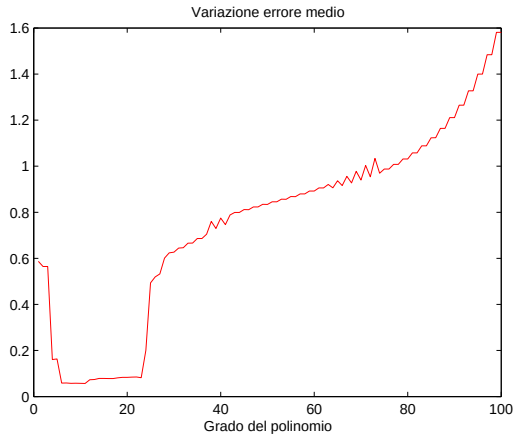
Nella figura 3.1a le ascisse di interpolazione sono equispaziate, e il polinomio interpolante (di ordine $n = 100$) è stato calcolato utilizzando l'algoritmo di Lagrange. Si noti come la qualità del polinomio interpolante sia estremamente bassa. Nella figura 3.1b invece le ascisse di interpolazione sono le radici del polinomio di Chebychev: la qualità del polinomio interpolante è più accettabile,

ma comunque l'andamento si discosta notevolmente dalla funzione di partenza. Infine, nella figura 3.1c si è utilizzata la funzione 3.1, impostando $n = 10$ come grado del polinomio approssimante. Le ascisse dei nodi sono equispaziate.

3.2.2 Scelta dell'ordine del polinomio approssimante

Si sono eseguite una serie di prove con l'obiettivo di osservare come varia l'errore in funzione del grado del polinomio approssimante. Come esempio abbiamo considerato la funzione $\sin(x)$, campionata nell'intervallo $[-4, 4]$. I campioni sono disturbati da rumore gaussiano bianco. Osservando la figura 3.2 si nota la presenza di un minimo dell'errore medio quando il grado del polinomio è pari a 6.
Questa osservazione è confermata dalla figura 3.3. Si osservi che la qualità dell'approssimazione è più alta, come ci si attendeva dalla figura 3.2, nel caso in cui il polinomio ha grado 6.

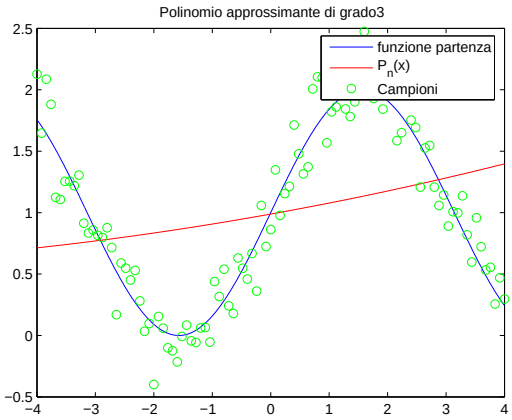
Figura 3.2: Valore medio dell'errore in funzione dell'ordine del polinomio approssimante per $f(x) = \sin(x)$



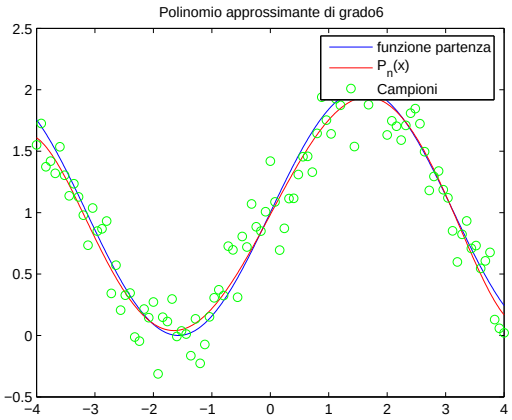
In conclusione possiamo dire che non esiste alcuna regola per determinare a priori il grado ottimo del polinomio approssimante, ma questo va valutato caso per caso.

Figura 3.3: Approssimazione della funzione $f(x) = \sin(x)$ in presenza di rumore e al variare del grado del polinomio approssimante

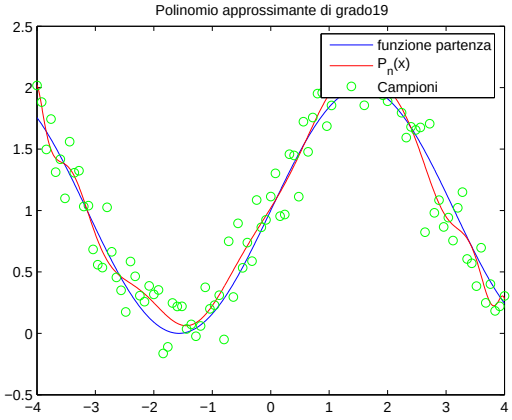
(a) Grado $n = 3$



(b) Grado $n = 6$



(c) Grado $n = 19$



(d) Grado $n = 25$