Università degli studi di Cagliari

Metodi di Interpolazione Polinomiale e Trigonometrica in Matlab

Fabio Manca
Docente: Prof. Giuseppe Rodriguez

Indice

Capitolo 1

Approssimazione di funzioni

Introduzione

Nel campo dell'analisi numerica può risultare utile effettuare l'approssimazione di una funzione. Tale esigenza scaturisce da varie cause, fra le quali, le principali sono:

- *l'assenza dell' espressione analitica della funzione che descriva un certo fenomeno fisico, disponendo invece soltanto di alcuni valori conseguentemente ad opportune misurazioni effettuate.*
- *pur essendo nota l'espressiona analitica, essa risulta complicata da gestire.*

Nel primo caso si deve trovare una funzione approssimante a partire dai punti noti, mentre nel secondo caso occorre sostituire la funzione nota con una più semplice dal punto di vista operativo.
Esistono due modi diversi di risolvere il problema dell'approssimazione di una funzione, questi prendono il nome di Interpolazione e di Approssimazione ai minimi quadrati. L'interpolazione è un metodo nel quale si richiede alla funzione approssimante di assumere lo stesso valore della funzione da approssimare
Nel metodo ai minimi quadrati si richiede invece che la funzione approssimante sia la più vicina alla funzione approssimata rispetto ad una norma che calcola la distinza delle due funzioni.
Risulta dunque evidente che la scelta dell'uno o dell'altro metodo sia legata ai dati che si possiedono in merito alle funzioni da approssimare ed alle risorse di calcolo di cui si dispone. Infatti, tanto più numeroso è il campione di dati a disposizione, tanto meno saremo interessati a voler conoscere ulteriori informazioni; al contrario ci risulterà utile ridurre i dati, per ridurre di conseguenza la complessità di una eventuale elaborazione. Inoltre, se i dati sono affetti da errore (come succede nella pratica), non è

detto che sia necessariamente utile cercare una funzione che passi per quei punti di interpolazione, perchè, sebbene possa dare calcoli localmente accurati, potrebbe produrre errori intollerabili al di fuori dell'intorno degli stessi punti di interpolazione.

Capitolo 2

Interpolazione

Siano assegnate le (n + 1) coppie di numeri reali

$$(x_i, y_i), \quad i = 0, ..., n. \tag{2.1}$$

una funzione $\phi(x)$ è definita interpolante se

$$\varphi(x_i) = y_i \quad i = 0, ..., n. \tag{2.2}$$

Supponendo che i punti di interpolazione provengano da una funzione incognita $f(x)$, cioè che valga la condizione di interpolazione $y_i = f(x_i)$ e scelte n + 1 funzioni $\psi_j, j = 0, ..., n$ si vuole approssimare $f(x)$ mediante una loro combinazione lineare:

$$\varphi(x) = \sum_{j=0}^{n} \alpha_j \psi_j(x) \tag{2.3}$$

che sia interpolante. I coefficienti della combinazione lineare (2.3) si ottengono imponendo le condizioni di interpolazione (2.2), ottenendo il sistema di equazioni lineari

$$\sum_{j=0}^{n} \psi_j(x_i)\alpha_j = y_i, i = 0, ..., n$$

la cui soluzione $\alpha = (\alpha_0, ..., \alpha_n)^T$ è l'insieme dei coefficienti della combinazione lineare; si dimostra che la funzione di interpolazione esiste ed è unica se e solo se il determinante di ψ, detto ***determinante di Haar***, è diverso da zero $\det(\psi) \neq 0$ (condizione di unisolvenza), ossia se:

$$\det(\Psi) \neq 0 \tag{2.4}$$

con

$$\Psi = \psi_j(x_i), \quad i, j = 0, ..., n$$

Un insieme di funzioni che verificano tale soluzione è anche detto **sistema di Chebychev**.

2.1 Interpolazione Polinomiale

In analisi matematica, i polinomi sono tra le funzioni più semplici da utilizzare; molte funzioni possono essere approssimate con polinomi, in modo che tale approssimazione sia abbastanza precisa.
Il mezzo con il quale si pratica tale approssimazione è il polinomio di Taylor. In effetti il **Teorema di Taylor** afferma che un polinomio di Taylor, ossia una serie di Taylor troncata, può approssimare bene una funzione in prossimità del valore iniziale x_0.

Un altro risultato che afferma le capacità di approssimazione dei polinomi è il **Teorema di Weierstrass**: *Sia* $f \in C[a, b]$. *Per* $\forall \epsilon > 0$ *esiste un intero* n *e un polinomio* p_n *di grado* n *tale che*

$$||f - p_n||_\infty < \epsilon$$

Si capisce, dunque, come i polinomi siano le funzioni più frequentemente utilizzate come base per l'interpolazione.

2.1.1 Il polinomio interpolante in forma canonica

Utilizzando la base canonica

$$span(1, x, x^2, ..., x^n)$$

è possibile costruire la rappresentazione del polinomio interpolante con una combinazione lineare del tipo:

$$p_n(x) = \sum_{j=0}^{n} \alpha_j x^j \tag{2.5}$$

dalla quale, imponendo le condizioni di interpolazione, potrà ricavarsi il sistema lineare

$$p_n(x_i) = y_i = \sum_{j=0}^{n} \alpha_j x_i^j, \quad i = 1, ..., n$$

la cui matrice dei coefficienti è

$$\psi = \begin{pmatrix} 1 & x_0 & x_0^2 & \cdots & x_0^n \\ 1 & x_1 & x_1^2 & \cdots & x_1^n \\ \vdots & \vdots & \ddots & \vdots & \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{pmatrix}$$

che prende il nome di **matrice di Vandermonde**. Dalla 1.4 sappiamo che è sufficiente mostrare che il determinante di Haar sia diverso da zero per mostrare anche l'unicità del polinomio interpolante. Dalla 1.4 consegue che

$$\det(\psi) = \prod_{i,j=0;\quad i>j}^{n} (x_i - x_j) \neq 0$$

cioè, il polinomio interpolante esiste ed è unica se e solo se i nodi sono tutti diversi fra loro.

Implementazione in Matlab

```
xi = -1:2/n:1; %nodi di interpolazione equispaziati
yi = 1./(1+25*xi.^2);
V = vander(xi); %matrice di Vandermonde

%calcolo dei coefficienti del polinomio
c = V \ yi(:);

yy_can = polyval(c, xx); %polinomio interpolante in forma canonica
```

La function polyval valuta il valore di un polinomio in una griglia di punti usando l'algoritmo di Horner Ruffini. `y=polyval(p,x)` restituisce il vettore y contenente i valori di un polinomio di grado n calcolati nei punti x.

E' possibile graficare l'errore che l'interpolante canonica commette rispetto alla funzione di Runge. Si discuterà in seguito in maniera più approfondita dell'Errore di Interpolazione.

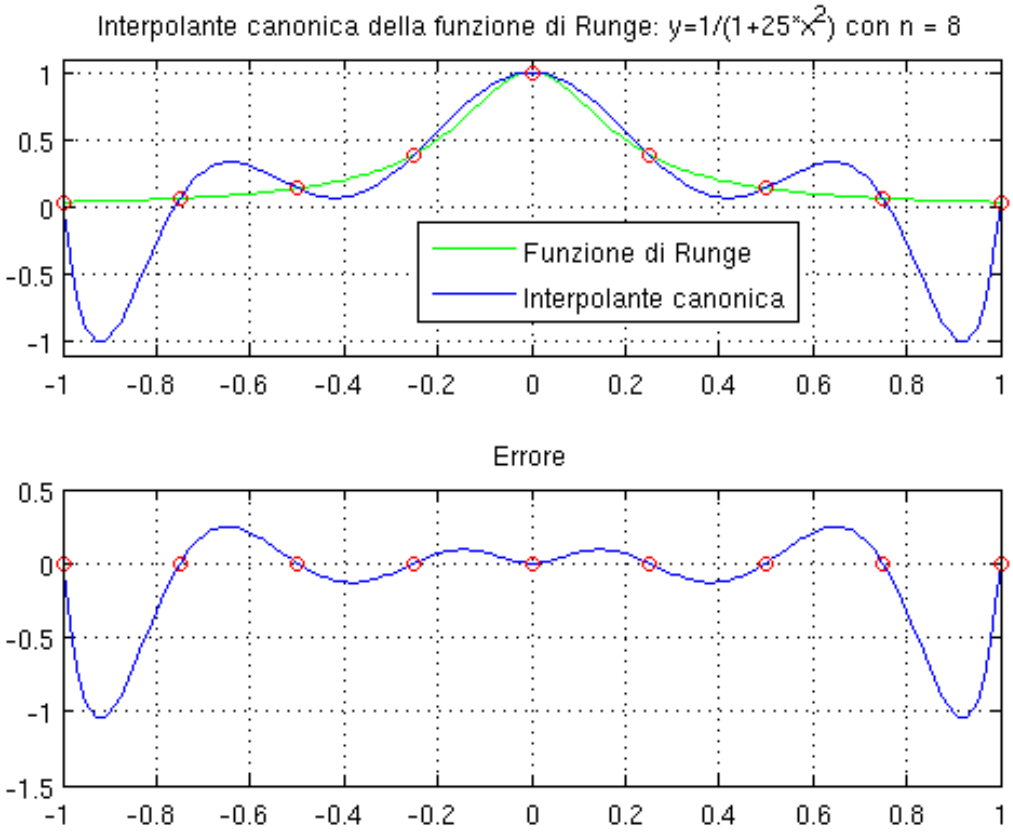

Figura 2.1: Interpolazione con polinomio in forma canonica della funzione di Runge

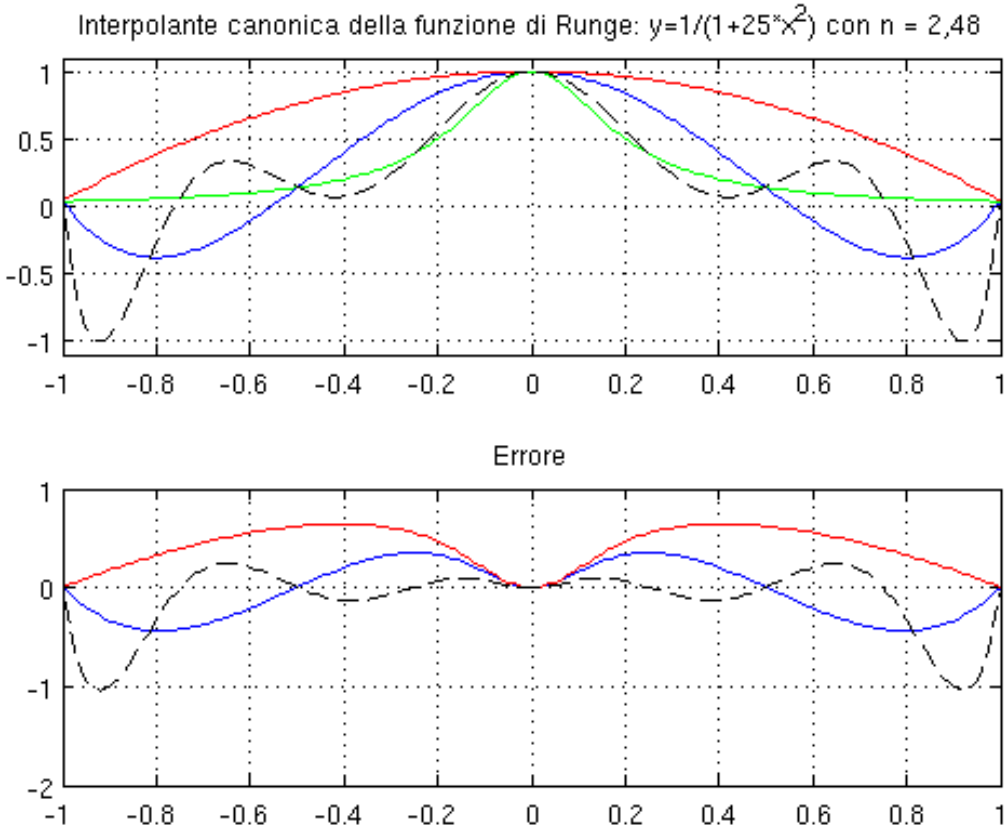

Figura 2.2: Interpolazione con polinomio in forma canonica della funzione di Runge per n=2,4,8

2.1.2 Il polinomio interpolante di Lagrange

Nel caso del polinomio interpolante in forma canonica, si è optato per la scelta della base più semplice (quella canonica), rinunciando però alla semplicità computazionale nel caso dei coefficienti. La rappresentazione di Lagrange opera nel modo inverso, adottando come coefficienti le stesse ordinate di interpolazione al prezzo di una maggiore complessità nel calcolo della base.
Assegnati i dati

$$(x_i, x_j), \quad i = 0, ..., n$$

i polinomi caratteristici di Lagrange $[L_j(x)]_{j=0}^{n}$ sono definiti da

$$L_j(x) = \prod_{k=0;\quad k\neq j}^{n} \frac{x - x_k}{x_j - x_k} \tag{2.6}$$

per i quali è semplice osservare

$$L_j(x_i) = \delta_{ij}$$

essendo δ_{ij} la **delta di Kronecker** che vale 1 per $i = j$, 0 altrimenti.
Il polinomio di Lagrange assume dunque la forma

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

Implementazione in Matlab

```
for i=1:n
   for j=1:n
      if (i~=j)
         L(i,:)=L(i,:).*(x-p_x(j))/(p_x(i)-p_x(j));
      end
   end
end
y=0;
for i=1:n
   y=y+p_y(i)*L(i,:);
end
```

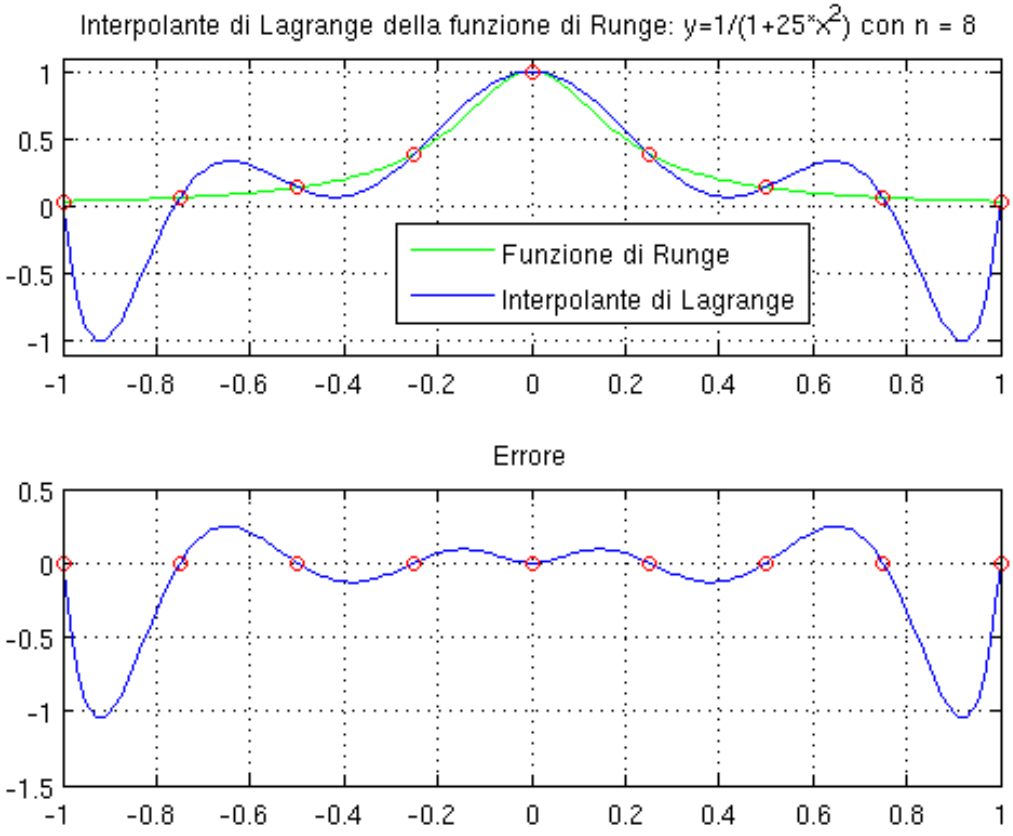

Figura 2.3: Interpolazione con polinomio di Lagrange della funzione di Runge con n=8

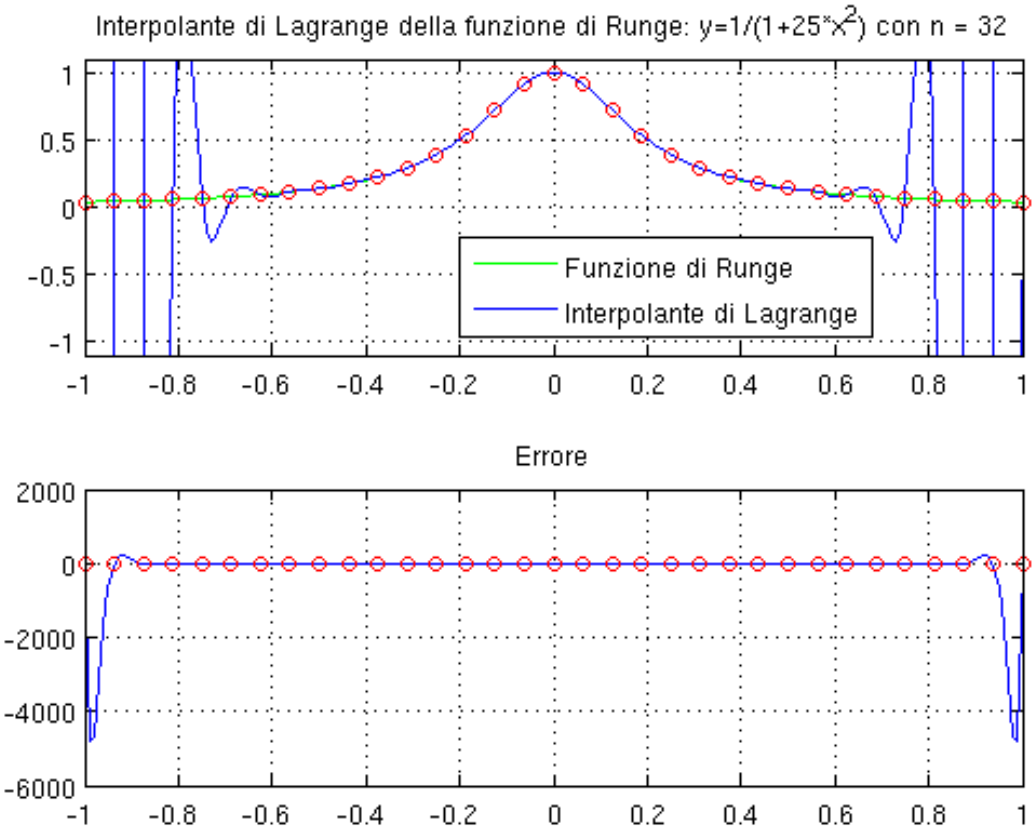

Figura 2.4: Interpolazione con polinomio di Lagrange della funzione di Runge n=32

2.1.3 La formula di Neville

Come era stato introdotto dall'inizio, non sempre il problema dell'interpolazione coincide con quello di dover ricavare l'intero polinomio interpolante (come nei due casi precedenti). La formula di Neville, consente invece di valutare il polinomio interpolante senza costruirlo esplicitamente. In questo caso è sufficiente conoscere il polinomio solo in pochi punti (*estrapolazione*). La formula di Neville permette di valutare il polinomio interpolante in un punto a partire da polinomi interpolanti su un numero inferiore di punti. Si dimostra che un polinomio

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

che interpola k + 1 punti

$$x_r, x_{r+1}, ..., x_{r+k}$$

può essere definito a partire da due polinomi interpolanti k punti mediante la relazione

$$Q_{r,r+1,...,r+k} = \frac{(x - x_r)Q_{r+1,r+2,...,r+k}(x) - Q_{r,r+1,...,r+k-1}(x)}{x_{r+k} - x_r} \tag{2.7}$$

Dati i punti di interpolazione $(x_i, y_i), i = 1, ..., n$ è possibile utilizzare la relazione (1.7) per costruire iterativamente lo schema di Neville (Fig. 2.6) mediante il quale, posti al primo passo $Q_i = y_i$, si può valutare il polinomio interpolante $Q_{0,1,...,n}(x)$.

Implementazione in Matlab

```
for i = n:-1:1
   for j = 1:i
      N(j) = (xInt-x(j))*N(j+1) - (xInt-x(j+n+1-i))*N(j);
      N(j) = N(j)/(x(j+n+1-i)-x(j));
   end
end

y = N(1);
```

2.1.4 Il polinomio interpolante di Newton

Possono esserci casi in cui è già stato calcolato un polinomio $p_{n-1}(x)$ che interpola i punti da $x_0 a x_{n-1}$ e in cui si vuole aggiungere il punto di interpolazione (x_n, y_n) che

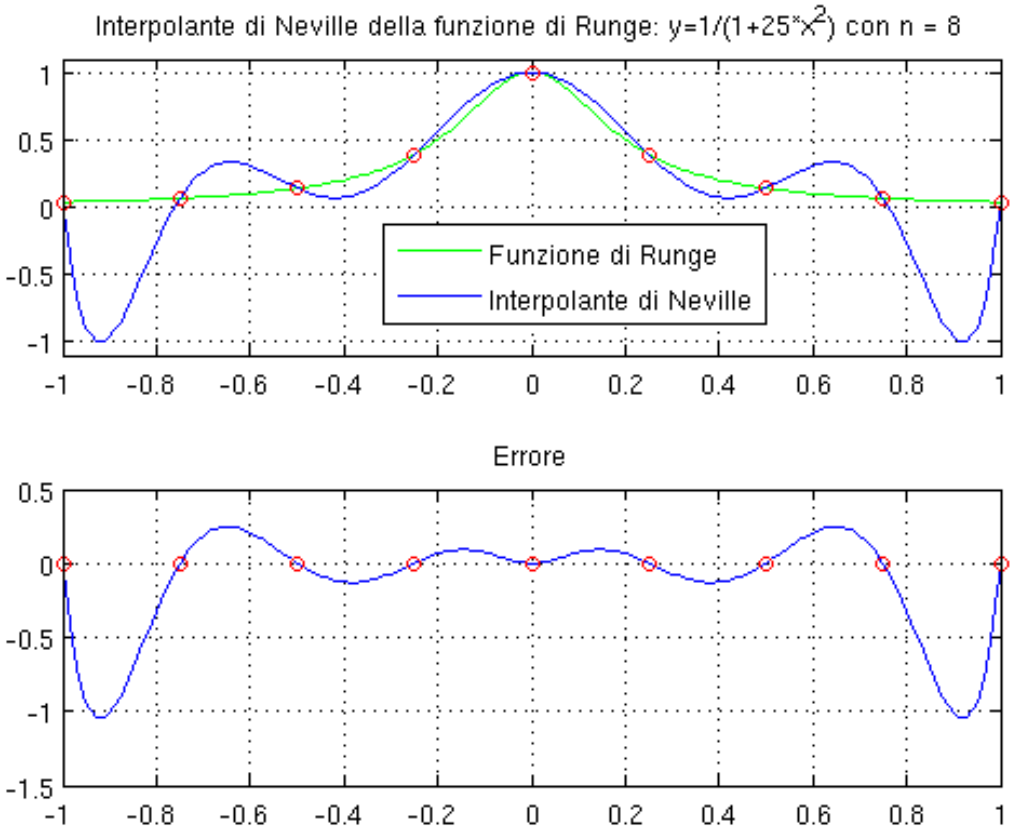

Figura 2.5: Interpolazione con polinomio di Neville della funzione di Runge

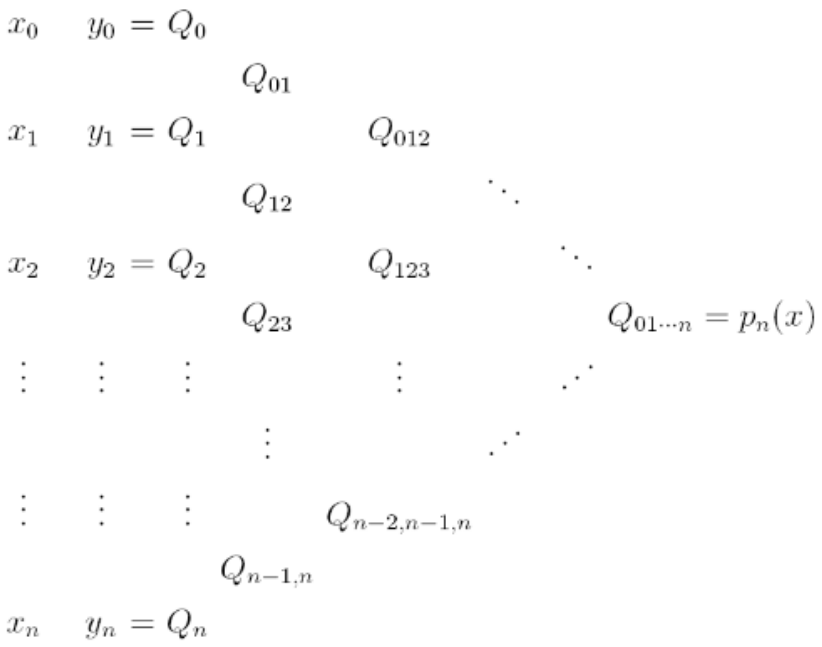

Figura 2.6: Schema delle differenze divise

interpoli tutti i punti $(x_i, y_i), \quad i = 0, ..., n$ nella forma

$$p_{n-1}(x) + g_n(x)$$

dove $g_n(x)$ deve essere anch'esso interpolante gli $n-1$ punti, e quindi nella forma:

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

con

$$a_n = \frac{y_n - p_{n-1}(x_i)}{\omega_{n-1}(x)}$$

determinato imponendo che p_n interpoli anche l'ultimo punto x_n, y_n. In realtà esiste un algoritmo più stabile e con minore complessità per ricavare il coefficiente a_n. Sfruttando la (1.7) possiamo scrivere il coefficiente a_n mediante le differenze divise di ordine n+1:

$$f[x_0, ..., x_n] = \frac{f[x_1, ..., x_n] - f[x_0, ..., x_{n-1}]}{x_n - x_0}$$

La formula ricorsiva ottenuta permette di calcolare qualsiasi differenza divisa ponendo $y_i = f[x_i]$ con $i = 0, ..., n$ Lo schema delle differenze divise è analogo a quello di Neville. A partire dai coefficienti calcolati con tale schema, si ottiene il polinomio interpolante di Newton:

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + ...$$
$$... + f[x_0, x_1, ..., x_n](x - x_0)(x - x_1)...(x - x_{n-1}) \tag{2.8}$$

Implementazione in Matlab

```
for i = 1:(n-1)
    for j = 1:(n-i)
        Y(j) = (Y(j+1) - Y(j))/(X(j+i) - X(j));
    end
    for k = i
        p = p.*(x-X(i));
    end
    y = y + p.*Y(1);
end
```

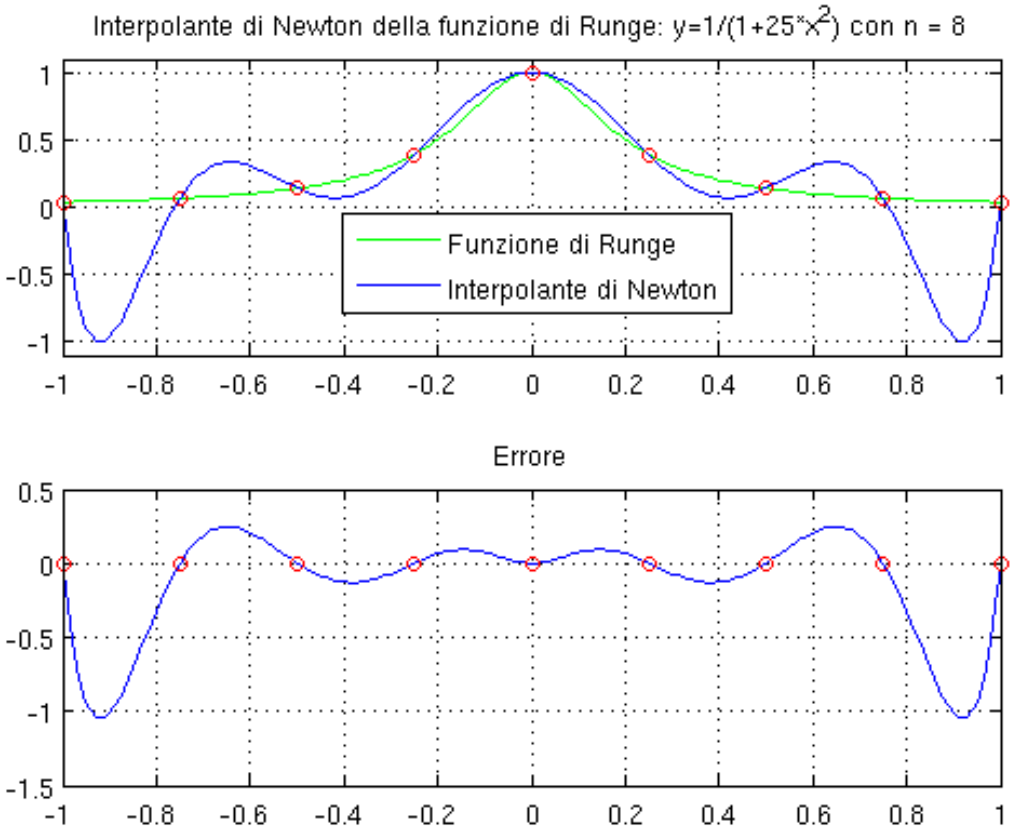

Figura 2.7: Interpolazione con polinomio di Newton della funzione di Runge

2.1.5 Errore di interpolazione

Nel caso dell'interpolazione polinomiale, ha interesse valutare lo scarto tra i valori $f(x)$ e $p_n(x)$ per tutti i punti, eccetto quelli di interpolazione $x = x_i$ in cui per le condizioni di interpolazione i valori di $f(x)$ e $p_n(x)$ coincidono. Tale scarto prende anche il nome di **errore di interpolazione**, e si dimostra che se $f \in \mathcal{C}^{(n+1)}_{[a,b]}$ allora $\exists \xi_x \in \mathcal{I}_{(x_0,x_1,...,x_n)}$ con $x_1, x_2, ..., x_n$ punti di interpolazione, tale che l'errore di interpolazione possa essere espresso come

$$E_n(x) = f(x) - p_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} w_n(x) \tag{2.9}$$

Questa relazione fornisce due informazioni qualitative rilevanti sull'errore. Il valore assoluto del numeratore del secondo membro $|f^{(n+1)}(\xi_x)|$ può essere sempre maggiorato con una costante, essendo la funzione di classe $\mathcal{C}^{n+1}$. Il fattore $\omega_n(x)$ è un polinomio che dipende dalla *disposizione* dei nodi x_i e questo significa che la distribuzione scelta per i nodi di interpolazione influisce sull'entità dell'errore. Ci chiediamo allora quale sia la distribuzione ottimale dei nodi per il problema dell'interpolazione polinomiale. Tale risposta è fornita dal **teorema di Bernstein**: *Se $f \in \mathcal{C}^{\infty}[a,b]$ e se le ascisse di interpolazione $x_{i\,i=0}^{n}$ sono gli zeri del polinomio di Chebychev di grado $n + 1$ allora l'errore di interpolazione tende a zero per $n \to \infty$.*

Il polinomio di Chebychev viene definito come:

$$T_{n+1}(x) = \cos((n+1)\theta), \quad per \quad x = \cos(\theta) \quad e \quad \theta \in [0, \pi]. \tag{2.10}$$

Gli zeri di tale polinomio sono valutabili ponendo:

$$\cos((n+1)\theta) = 0$$

.

Implementazione in Matlab per i nodi di Chebyshev

```
for i=1:n
  x(i) = 0.5*(a + b) + 0.5*(a - b)*cos( ((i-1)/n)*pi )
end
```

Figura 2.8: Confronto delle interpolanti con polinomio in forma canonica della funzione di Runge, utilizzando nodi equispaziati e nodi di Chebyshev con n=16

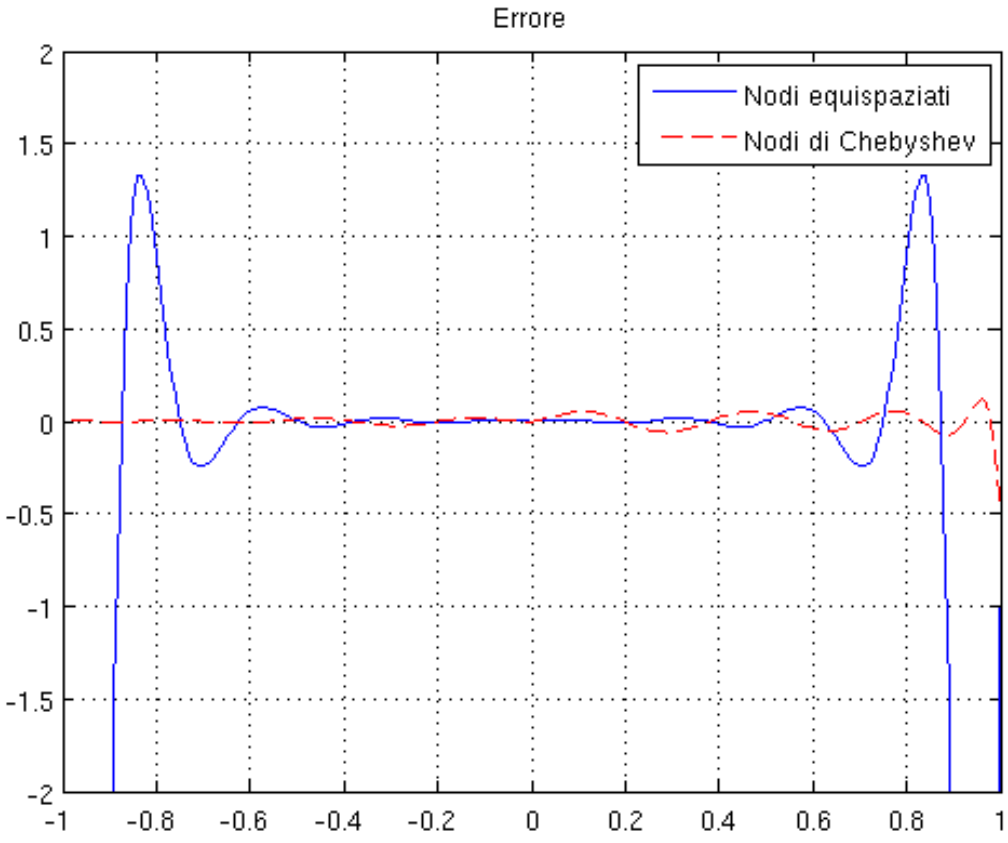

Figura 2.9: Confronto degli errori delle interpolanti con polinomio in forma canonica della funzione di Runge, utilizzando nodi equispaziati e nodi di Chebyshev

Rappresentazione alternativa dell'errore

Possiamo utilizzare il formalismo di Newton per fornire un'altra espressione dell'errore di interpolazione, più generale della precedente, dato che non richiede nè la differenziabilità, nè la continuità della funzione f. Dato un polinomio $p_n(x)$ che interpola una certa funzione $f(x)$ sui nodi $x_0, ..., x_n$, supponiamo di voler aggiungere il punto di interpolazione di coordinate $(x, f(x))$:

$$p_{n+1}(z) = p_n(z) + f[x_0, ..., x_n, x]\omega_n(z)$$

valutando questo polinomio in x e calcolando l'errore di interpolazione si ottiene:

$$E_n(x) = f(x) - p_n(x) = f[x_0, ..., x_n, x]\omega_n(x)$$

Questa reppresentazione dell'errore è analoga a quella data precedentemente, ma è più generale perchè, oltre a non richiedere la differenziabilità per la funzione f, fornisce un'espressione effettivamente calcolabile per l'errore di interpolazione in un punto x, visto che essa non dipende dal punto incognito ξ_x.
Confrontando le due rappresentazioni, si arriva alla relazione:

$$f[x_0, x_1, \ldots, x_n] = \frac{f^{(n+1)}(\xi_x)}{(n+1)!}$$

Capitolo 3

I polinomi trigonometrici

Una funzione periodica di periodo T > 0 è una funzione che verifica la proprietà

$$f(x+T) = f(x), \forall x$$

Per approssimare una funzione di questo tipo, riproducendo ad esempio il suo comportamento, dal momento che i polinomi algebrici non sono periodici, è necessario ricorrere a funzioni in un altro spazio.
In particolare è possibile utilizzare lo spazio generato dai monomi trigonometrici

$$T_M = (1, \cos kx, \sin kx, k = 1, 2, ..., M)$$

cui appartengono funzioni periodiche di periodo 2π
Una funzione $\mathcal{T} \in T_M$

$$\mathcal{T}(x) = \frac{a_0}{2} + \sum_{k=1}^{M} (a_k cos kx + b_k sin kx)$$

con $a_k, b_k \in \Re$ o $\mathcal{C}$ viene detta *polinomio trigonometrico* di ordine M.
Ricordando che

$$\cos x = \frac{e^{ix} + e^{-ix}}{2}, \quad \sin x = \frac{e^{ix} - e^{-ix}}{2i}$$

, si ottiene

$$\mathcal{T}(x) = \sum_{k=-M}^{M} c_k e^{ikx}$$

dove

$$c_k = \frac{a_k - ib_k}{2}, \quad c_{-k} = \frac{a_k + ib_k}{2} \quad (b_0 = 0)$$

3.1 Interpolazione con i polinomi trigonometrici

Abbiamo precedentemente discusso il significato di interpolazione polinomiale.
Ne segue che l'interpolazione polinomiale trigonometrica è l'interpolazione che utilizza polinomi trigonometrici, cioè delle funzioni espresse come somma di seni e coseni per un dato periodo. Se il set di punti fornito è equispaziato si presenta un caso importante di Serie di Fourier per cui la soluzione è data dalla *Trasformata Discreta di Fourier* che discuteremo in seguito.
Come abbiamo visti precedentemente, il **teorema di Weierstrass** pone le basi teoriche all'approssimazione di funzioni f. In particolare, ora siamo interessati a funzioni di periodo $T = 2\pi$.
La costruzione del polinomio approssimante può essere effettuata con una distribuzione di nodi arbitrari in $[0, 2\pi]$, distinti, escludendo almeno uno dei due estremi, e si dimostra che comunque essi si scelgano la matrice di collocazione è a rango massimo. La scelta dei nodi equidistanti è comunque privilegiata dal punto di vista applicativo e teorico. Quindi considerando N nodi in $[0, 2\pi]$, equidistanti, con passo $h = 2\pi/N$

$$x_k = \frac{2\pi k}{N}, \quad k = 0, 1, ..., N-1$$

e i valori $f_k \approx f(x_k)$, si vuole determinare il polinomio trigonometrico $\mathcal{T} \in T_L, N > 2L + 1$ tale che

$$\sum_{k=0}^{N-1} [\mathcal{T}(x_k) - y_k]^2 = \min \tag{3.1}$$

cioè si intende risolvere il problema della ricerca di una funzione approssimante nel senso dei minimi quadrati.

Proprietà della base esponenziale

I nodi equidistanti, forniscono alcune proprietà importanti alla base esponenziale. Posto $\omega = e^{ix}$ e $\omega_k = e^{ix_k} = e^{2\pi ik/N}$ si ha la importante proprietà di ortogonalità

$$\sum_{k=0}^{N-1} \omega_k^m \omega_k^{-n} = N\delta_{mn} \tag{3.2}$$

E' facile notare che se $m = n$ ogni componente della serie è unitario, mentre se $m \neq n$ si ottiene La somma parziale:

$$\sum_{k=0}^{N-1} e^{2\pi ik(m-n)/N}$$

e indicando la ragione della serie con z

$$z = e^{2\pi ik(m-n)/N} \tag{3.3}$$

si ottiene

$$\sum_{k=0}^{N-1} e^{2\pi ik(m-n)/N} = \frac{1-z^N}{1-z} = 0 \tag{3.4}$$

3.2 Trasformata di Fourier Discreta (DFT)

La teoria dell'approssimazione trigonometrica discreta è correlata con l'*analisi di Fourier* o *analisi armonica*, basata sull'uso delle serie di Fourier

$$S(x) = \sum_{k\in\mathbb{Z}} \hat{f}(k)e^{ikx}$$

$$\hat{f}(k) = \frac{1}{2\pi}\int_0^{2\pi} f(x)e^{-ikx}dx$$

dove per garantire significato all'integrale, si assume $f \in L[0, 2\pi]$, $L[a, b]$ spazio delle funzioni che hanno integrale finito su $[a, b]$.
Affrontiamo in questa sede il problema generico dell'elaborazione dei segnali nel dominio della frequenza mediante la **Trasformata Discreta di Fourier, (DFT)** e l'inversa di tale trasformata il cui acronimo è appunto **IDFT**.
La *trasformata di Fourier* di una sequenza $f(n)$ di durata finita L, è definita come:

$$F(k) = \sum_{n=0}^{L-1} f(n)e^{-i2\pi kn} \tag{3.5}$$

che è una funzione periodica e continua nella variabile k, e per questo l'elaborazione nel dominio della frequenza risulta difficile da implementare numericamente. Tuttavia, risulta possibile definire una **trasformata *discreta* di Fourier** campionando opportunamente $F(k)$ e cioè in modo che sia soddisfatta la condizione di Nyquist, che richiede che la frequenza di campionamento sia maggiore o uguale alla frequenza del segnale. In formule

$$f_c = \frac{1}{T} \geq 2B$$

dove B indica la banda monolatera.
Se si effettua il campionamento in frequenza, analogamente, deve accadere che l'inverso del passo di campionamento risulti maggiore o uguale della durata temporale del

segnale. Se si utilizza un passo di campionamento pari a 1/N , deve risultare $N \geq L$, cioè bisogna che ci siano un numero di campioni sul periodo maggiore o uguale della durata del segnale.

Scegliendo N = L, la DFT risulta essere:

$$F\left(\frac{k}{N}\right) \equiv \hat{f}_k = \sum_{n=0}^{N-1} f(n) e^{-i2\pi kn/N} = \sum_{n=0}^{N-1} f(n)\omega_N^{-nk} \quad k = 0, ..., N-1 \tag{3.6}$$

Dove

$$\omega_N = e^{2\pi i/N}$$

viene detto radice ennesima dell'unità, in quanto

$$(\omega_N)^N = e^{2\pi i} = 1$$

Inoltre ω_N viene detta *primitiva*, in quanto le sue potenze ω_N^k, $k = 0, .., N-1$ sono tutte radici ennesime dell'unità. Ricordiamo inoltre che tali $(\omega_N)^N$ coincidono con le z definite in (3.3), e pertanto godono della medesima proprietà di ortogonalità.

Nella (3.6) è immediato notare la somiglianza con la relazione che definisce i coefficienti di Fourier. In effetti le due equazioni sono identiche a meno di un fattore di scala 1/N.
Usando questa relazione possiamo definire la **trasformata inversa di Fourier** $DFT^{-1} = IDFT$

$$f_n = f(n) = \frac{1}{N} \sum_{k=0}^{N-1} \hat{f}_k \omega_N^{nk} \quad n = 0, ..., N-1 \tag{3.7}$$

In forma matriciale risulta, se $f = (f_0, ..., f_N)^T$ e $\hat{f} = (\hat{f}_0, ..., \hat{f}_N)^T$

$$\hat{f} = \frac{1}{N} F_N f \quad , \quad (F_N)_{kn} = \omega_N^{-nk}$$

$$f = F_N^* \hat{f} \quad , \quad (F_N^*)_{nk} = \omega_N^{nk}$$

La matrice F è una matrice di Vandermonde che prende il nome di **matrice di Fourier**:

$$F = \begin{pmatrix} \omega_N^{0\cdot 0} & \omega_N^{0\cdot 1} & \dots & \omega_N^{0\cdot(N-1)} \\ \omega_N^{1\cdot 0} & \omega_N^{1\cdot 1} & \dots & \omega_N^{1\cdot(N-1)} \\ \vdots & \vdots & \ddots & \vdots \\ \omega_N^{(N-1)\cdot 0} & \omega_N^{(N-1)\cdot 1} & \dots & \omega_N^{(N-1)\cdot(N-1)} \end{pmatrix}$$

Dalla relazione di ortogonalità si ricava:

$$(F_N F_N^*)_{jk} = \sum_{k=0}^{N-1} \omega_N^{-in} \omega_N^{nk} = N\delta_{jk}$$

la matrice F è *quasi unitaria*

$$F_N F_N^* = NI$$

3.3 Fast Fourier Transform (FFT)

Come si è visto la trasformata discreta di Fourier canpiona ad intervalli regolari la trasformata di Fourier del segnale, e precisamente, campiona ogni $2\pi k/N$. Questo significa calcolare N punti della trasformata discreta di Fourier.
Computazionalmente, tali operazioni potrebbero risultare onerose visto che si dimostra, che su N campioni, la complessità della trasformata discreta di Fourier è $\mathcal{O}(N^2)$; questo implica la necesssità di avvalersi di algoritmi a complessità minore, come la **Fast Fourier Transform (FFT)**.
Ad essere precisi, l'acronimo FFT indica una *classe* di algoritmi efficienti per il calcolo della DFT di una sequenza periodica.
La FFT viene normalmente attribuita a Cooley e Tuckey [1]. In realtà questo algoritmo venne scoperto e usato da Gauss nel XIX secolo [2].
Il grosso vantaggio della FFT è che permette di calcolare la trasformata discreta di Fourier in $\mathcal{O}(NLog_2N)$ operazioni.
Ci si rende presto conto del vantaggio in termini di velocità di calcolo; si noti, a titolo di esempio, che per N = 1000, $N^2 = 1,000,000$ mentre $Nlog_2N \approx 9,965.78$; quindi, per N = 1000, il numero di operazioni necessarie viene ridotto di un ordine 100.
Comunque, esistono diversi limiti anche per l'algoritmo FFT.
Infatti, l'algoritmo assume che il numero di punti sia una potenza di due, e che siano equispaziati e periodici. Quindi, se si prova ad eseguire la FFT di un set di punti non periodico il risultato ottenuto sarà affetto da rumore, mentre nel caso in cui il numero di punti non sia una potenza di due, l'algoritmo sarà meno veloce.

3.4 Interpolazione trigonometrica

3.4.1 Sistemi lineari per i coefficienti di Fourier

L'interpolante trigonometrica $y = f(x)$ avrà n coefficienti $[a_k, b_k]$, che devono essere trovati da n equazioni. Per questo si impongono le condizioni di interpolazione:

$y_i = f(x_i)$.
Se $n = 2m$, l'iterpolante trigonometrica $y = f(x)$ prende la seguente forma:

$$f(x) = \frac{1}{2a_0} + \sum_{k=1}^{m-1} \left[a_k \cos\left(\frac{2\pi x}{L} k\right) + b_k \sin\left(\frac{2\pi x}{L} k\right) \right] + a_m \cos\left(\frac{2\pi m x}{L}\right)$$

per cui i coefficienti di Fourier verrano trovati risolvendo il sistema lineare:

$$y_i = f(x_i) = \frac{1}{2a_0} + \sum_{k=1}^{m-1} \left[a_k \cos\left(\frac{2\pi x_i}{L} k\right) + b_k \sin\left(\frac{2\pi x_i}{L} k\right) \right] + a_m \cos\left(\frac{2\pi m x_i}{L}\right)$$

Implementazione in Matlab

```
xx = [x(m+1:n),x(1:m)]';
yy = [y(m+1:n),y(1:m)]';

f_x = 0.5*ones(n,1); % costruzione dei coefficienti per il sistema lineare
cost = 2*pi/N;

for k = 1 : (m-1)
     f_x = [ f_x, cos(k*xx*cost), sin(k*xx+cost) ];

end

f_x = [ f_x, cos(m*xx*cost) ];

c = f_x\yy; % calcolo dei coefficienti di Fourier

a = [c(1); c(2:2:n)]';
b = [0; c(3:2:n)]';

yInt = 0.5*a(1)*ones(1,length(xInt));

for k = 1 : (m-1)
  yInt = yInt + a(1+k)*cos(k*xInt*cost) + b(1+k)*sin(k*xInt*cost);

end
```

```
yInt = yInt + a(m+1)*cos(m*xInt*cost);
```

Quando l'intervallo di partizione $x \in [0, N]$ è equispaziato, i coefficienti di Fourier possono essere calcolati direttamente dalle seguenti:

$$a_k \frac{2}{n} \sum_{i=1}^{n} y_i \cos(2\pi k x_i / N) \quad k = 0, 1, 2, ..L$$

$$b_k \frac{2}{n} \sum_{i=1}^{n} y_i \sin(2\pi k x_i / N) \quad k = 1, 2, ..L - 1$$

che rappresentano la Trasformata Discreta di Fourier definita in un dato set di punti x_i, y_i

Implementazione in Matlab

```
for j = 0 : m

a(j+1) = 2*yy'*cos(j*xx*cost)/n;

b(j+1) = 2*yy'*sin(j*xx*cost)/n;

end

% Coefficienti di Fourier delle serie trigonometriche

c = fft(yy);
c = c';

aF = 2*real(c(1:m+1))/n; bF = -2*imag(c(1:m+1))/n;

aF = aF, bF = bF  %Coefficienti di Fourier derivati dalla FFT

yInt = ifft(c)    % recupero la funzione ->  ifft(fft(y)) = y
```

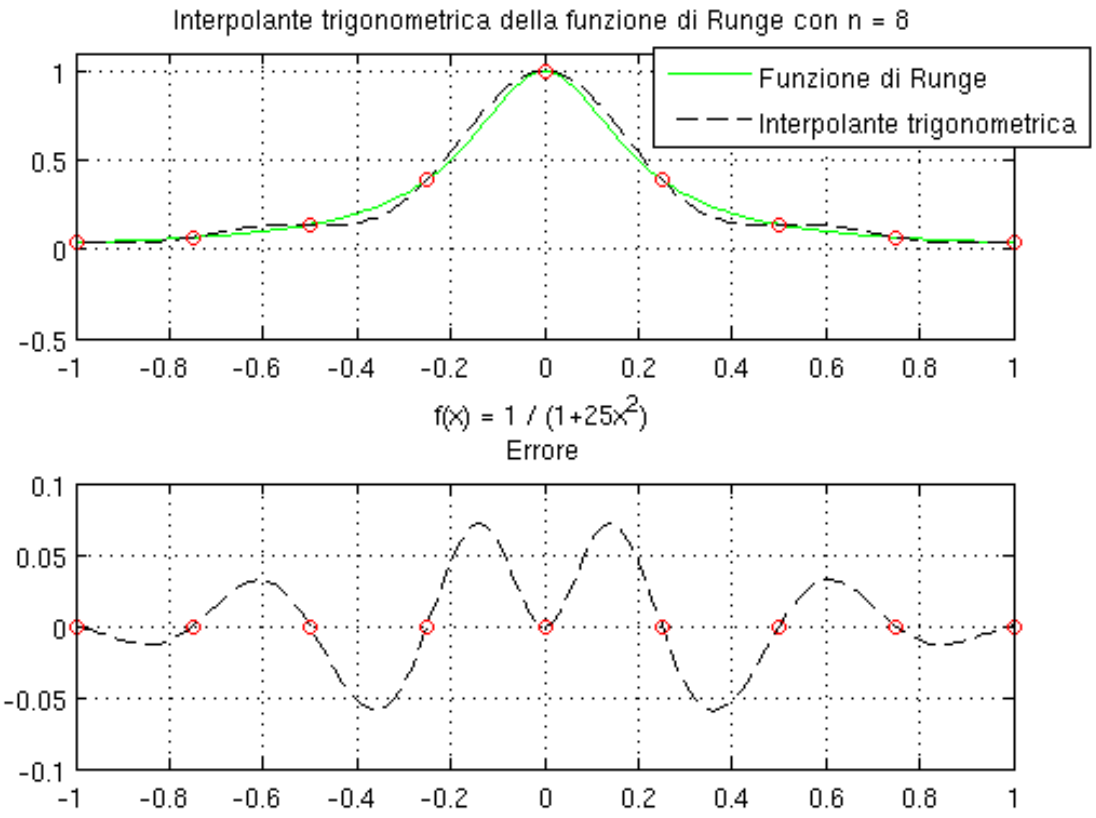

Figura 3.1: Interpolante trigonometrica della funzione di Runge con n=8

3.4.2 Interpolazione trigonometrica con la FFT

Nel precedente esempio in Matlab ci si è avvalsi delle function `fft` e `ifft` che sono l'implementazione in Matlab della *Fast Fourier Transform* e della *Inverse Fast Fourier Transform*.
Vediamo come si affronta il problema dell'interpolazione trigonometrica avvalendosi della FFT.
Sia f 2π-periodica su $[-\pi, \pi]$. Se è periodica su un altro intervallo si opera un cambio di variabile. Siano assegnati i valori

$$f_k = f(x_k), \quad k = 0, 1, ..., N-1$$

sui punti equidistanti

$$x_k = \frac{2\pi k}{N}, \quad k = 0, 1, ..., N-1$$

Vogliamo trovare il polinomio trigonometrico

$$T_{N-1}(x) = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} C_n e^{inx}$$

che interpola i dati, cioè tale che

$$T_{N-1}(x) = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} C_n e^{in\frac{2\pi k}{N}} = f_k$$

ponendo $(\omega_N)^N = e^{2\pi i}$, radice N-esima primitiva dell'unità,

$$f_k = \frac{1}{N} \sum_{n=-N/2}^{N/2-1} C_n \omega_N^{nk} \quad k = -\frac{N}{2}, ..., \frac{N}{2} - 1$$

che è la IDFT shiftata nell'origine.
Per il calcolo dei coefficienti useremo:

$$C_k = \sum_{n=-N/2}^{N/2-1} f_n \omega_N^{-nk} \quad k = -\frac{N}{2}, ..., \frac{N}{2} - 1$$

che è la DFT shiftata nell'origine. Per valutare il polinomio su una griglia più fitta è possibile effettuare lo **zero-padding**. Con l'operazione di zero-padding si estende

una sequenza di lunghezza N alla lunghezza M>N, aggiungendo M-N campioni nulli alla sequenza data. Cioè si valuta il polinomio sulla griglia M>Ne si pone

$$\hat{c}_k = \begin{cases} c_k & k{=}-\frac{N}{2},\ldots,\frac{N}{2}-1 \\ 0 & altrimenti \end{cases}$$

e si calcola

$$T_{N-1}(z_k) = \frac{1}{M} \sum_{n=-M/2}^{M/2-1} \hat{c_n} \omega_M^{nk} \quad k = -\frac{M}{2}, \ldots, \frac{M}{2} - 1$$

dove $\omega_M = e^{2\pi i/M}$ mediante la IDFT shiftata.

Il procedimento è dunque:

$$f \overset{shift}{\longmapsto} \hat{f} \overset{DFT}{\longmapsto} C \overset{shift}{\longmapsto} ZeroPadding \overset{shift}{\longmapsto} \hat{C} \overset{IDFT}{\longmapsto} \hat{t} \overset{shift}{\longmapsto} t$$

Nell'implementazione in Matlab si ricorda che:

$$shift- > \texttt{fftshift}$$

$$DFT- > \texttt{1/N*fft}$$

$$IDFT- > \texttt{N*ifft}$$

Inoltre, potrebbe essere opportuno considerare la parte reale dell'output della FFT per eliminare gli errori di arrotondamento. Infatti la DFT produce in generale risultati complessi, ma in certi casi, come questo, si sa che il risultato è reale. Si deve comunque verificare che $imag(fft(f)) \approx eps$

Implementazione in Matlab

```
n = 8;
x = [0:n-1]'/n*2-1;
f = 1./(1+25*x.^2);

ff = fftshift(f);
c = real(1/n * fft(ff));

N = 256;
t = [0:N-1]'/N*2-1;
```

```
zero_padd = zeros(N-n, 1);
c = [c(1:n/2); zero_padd ; c(n/2+1:n)];
g = real(N*ifft(c));
g = fftshift(g);
```

3.5 Codice Matlab per i test

3.5.1 Test interpolazione in forma: canonica, di Lagrange, di Neville, di Newton

```
clear all
clc

xx=[-1:.01:1];  % uso 100 punti per plottare la reale funzione di Runge
yy=1./(1+25*xx.^2);
%plot(xx, yy, 'g');
hold on;

n=8, %scelgo i punti di interpolazione

%per evidenziare i punti di intersezione con la funzione (x & y che passo
%all'interno di ogni test
x = -1:2/n:1;
y = 1./(1+25*x.^2);

%[yy_can, yy_err_can] = test_can(xx, yy, n); %canonica
%[yy_lagr, yy_err_lagr] = test_lagr(xx, yy, n); %lagrange
[yy_nev, yy_err_nev] = test_nev(xx, yy, n); %neville
%[yy_new, yy_err_new] = test_new(xx, yy, n); %newton

%-----plotta i polinomi
subplot(2,1,1)
plot(xx,yy,'g',xx,yy_nev,'b',x,y,'ro');
%plot(xx,yy_lagr,'r')
%plot(xx,yy_nev,'g')
%plot(xx,yy_new,'k')
```

```
axis([-1, 1, -1.1, 1.1])
legend('Funzione di Runge', 'Interpolante di Neville')
title(['Interpolante di Neville della funzione di Runge:
y=1/(1+25*x^2) con n = ' num2str(n)]);
grid on

subplot(2,1,2)
plot(xx,yy_err_nev,'b');
%plot(xx,yy_err_can,'b',xx,yy_err_lagr,'r', xx,yy_err_nev,
'g',xx,yy_err_new,'k');
%legend('Canonical','Lagrange','Neville','Newton');
hold on
plot(x,0,'ro');
title('Errore');
grid on
```

3.5.2 Test interpolazione in forma trigonometrica

```
clear all
clc

xx = -1 : 0.01 : 1; %per il plot della funzione di Runge
yy = 1./(1 + 25*xx.^2);

n = 8;
x = [-1:2/n:1];
y = 1./(1 + 25*x.^2);

yy_trig = trigon(x,y,xx);
yy_err_trig = yy_trig-yy;

[yy_can, yy_err_can] = test_can(xx,yy,n);
[yy_lagr, yy_err_lagr] = test_lagr(xx,yy,n);

subplot(2,1,1)
plot(xx, yy, 'g',xx,yy_can,'r', xx,yy_lagr, 'b--',
```

```
xx,yy_trig, 'k--',x, y,'or');

%plot(xx, yy, 'g', xx,yy_trig, 'k--',x, y,'or');
axis([-1 1 -0.5 1.1])
legend('Funzione di Runge','Interpolante canonica',
    'Interpolante di Lagrange','Interpolante trigonometrica')

%legend('Funzione di Runge','Interpolante trigonometrica')
title('Confronto interpolazione    ');
xlabel(['f(x) = 1 / (1+25x^2)'])
grid on


subplot(2,1,2)
plot(xx,yy_err_can,'r', xx,yy_err_lagr, 'b--',
     xx,yy_err_trig, 'k--',x, 0,'or');
%plot(xx,yy_err_trig, 'k--',x, 0,'or');
axis([-1 1 -0.1 0.1])
%legend('Interpolante canonica','Interpolante di Lagrange',
'Interpolante trigonometrica')
%title(['Confronto errore di interpolazione della funzione
  di Runge con n = ' num2str(n)]);
title('Errore');
grid on
```

Capitolo 4

Applicazioni e conclusioni

Esistono molte applicazioni della interpolazione trigonometrica e della DFT.
Queste vengono usate soprattutto per il filtraggio del rumore di un segnale applicando in primo luogo la DFT ad un segnale discreto e successivamente settando a zero tutte le frequenze indesiderate. Infine, viene usata la IDFT per ottenere il segnale filtrato.
La FFT può essere usata per valutare polinomi più velocemente.
Nel caso si debba effettuare l'interpolazione polinamiale di un segnale periodico, l'interpolazione trigonometrica risulta essere la scelta più naturale ed efficace. infatti, come si nota dai grafici di Fig. 4.1 e 4.2, a parità di numero di nodi, le prestazioni dell'interpolazione trigonometrica sono nettamente superiori delle prestazioni degli altri metodi di interpolazione discussi.

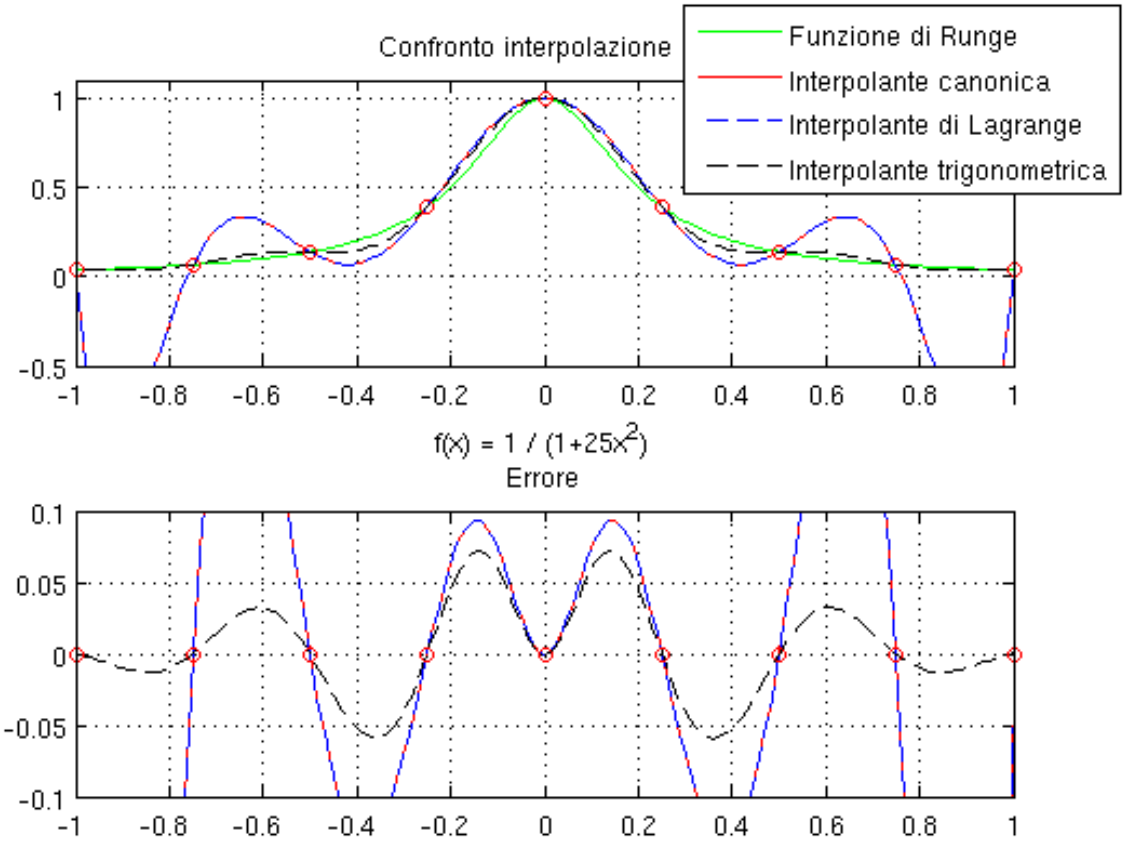

Figura 4.1: Confronto delle interpolanti in forma canonica, di Lagrange e trigonometrica con n=8 nodi

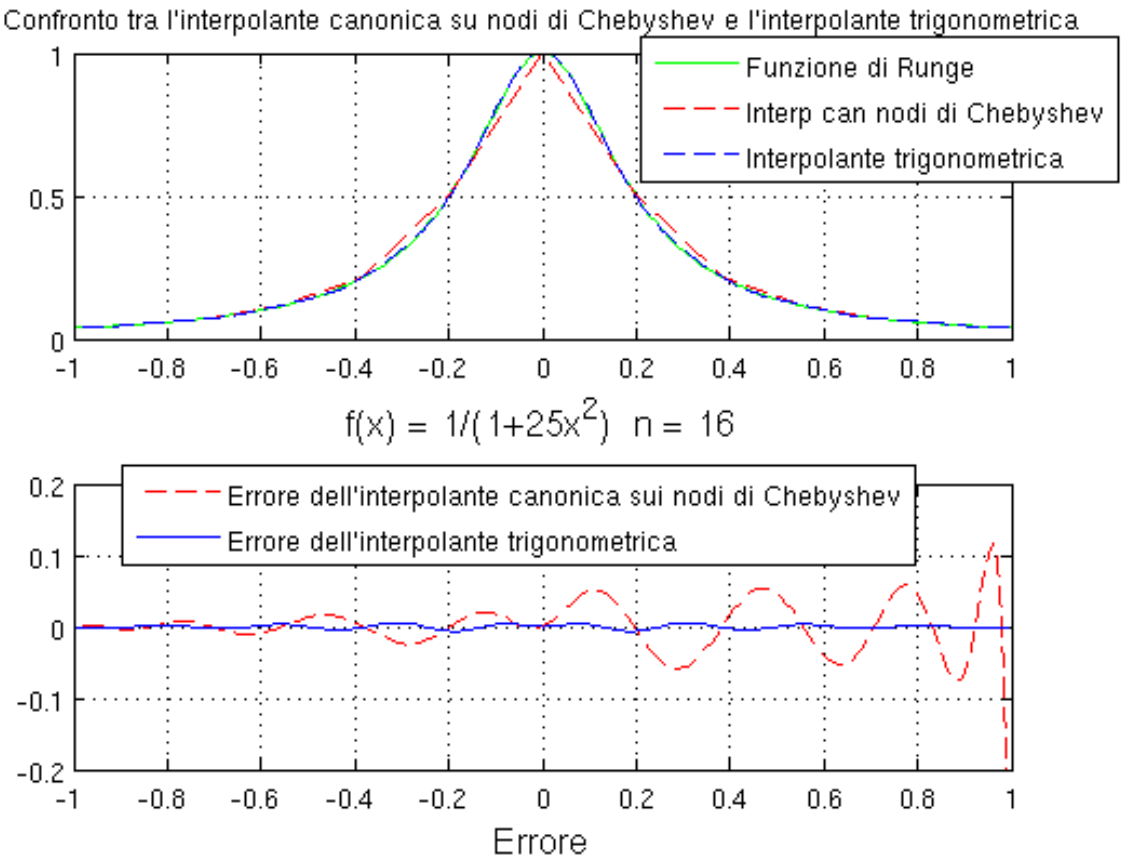

Figura 4.2: Confronto delle interpolanti in forma canonica con nodi di Chebyshev, e trigonometrica con n=16 nodi

Bibliografia

J. W. Cooley and J. W. Tukey. An algorithm for the machine calculation of complex fourier series.

M. T. Heideman, D. H. Johnson, and C. S. Burrus. Gauss and the history of the fast Fourier transform. IEEE ASSP Magazine, 4:14-21, 1984.

A. Quarteroni, R. Sacco e F. Saleri Matematica Numerica, Springer, (1998).

G. Rodriguez, Algoritmi Numerici, Pitagora Editrice.

M. L. Lo Cascio, www.dmmm.uniroma1.it/~locascio/Didattica/Elettrica_08_09 /Materiale_Didattico/D1_App_trigonometrica.pdf

A. Murli, www.dma.unina.it/~murli/didattica/2006_2007/cs/ estrattoCap7Vol2_FFT.pdf

Wikipedia, (Polynomial Interpolation), http://en.wikipedia.org/wiki/ Polynomial_interpolation

Wikipedia, (Fourier Series), http://en.wikipedia.org/wiki/Fourier_series

Wikipedia, (Trigonometric Interpolation), http://en.wikipedia.org/wiki/ Trigonometric_interpolation

Wikiversity, (Trigonometric Interpolation), http://en.wikiversity.org/ wiki/Trigonometric_interpolation