Algoritmi per l'interpolazione polinomiale
Implementazione MATLAB

Alessandro Gallo

Docente: Prof. Giuseppe Rodriguez

Indice

1 Il problema dell'interpolazione **2**
1.1 Il caso generale 2

2 Interpolazione polinomiale **3**
2.1 canonical.m 4
2.2 pointinteractive.m 5

3 Rappresentazione del polinomio interpolante **5**
3.1 Forma di Lagrange 6
3.2 lagrix.m 6
3.3 lagrange.m 7
3.4 Formula di Neville 7
3.5 neville.m 8
3.6 Formula di Newton 8
3.7 newton.m 9

4 Posizionamento dei nodi **10**
4.1 Nodi di Chebychev 10
4.2 cheby.m 11

5 Interpolazione 2d **12**
5.1 lag2d.m e newnev2d.m 13

1 Il problema dell'interpolazione

Il problema dell'interpolazione può essere presentato in due modi diversi:

1. Dato un insieme di $(n+1)$ punti (x_i, y_i), $i = 0, \ldots, n$ vogliamo trovare una funzione $\Phi(x)$ che passi per ciascun punto, cioè tale che

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

   Una tale funzione prende il nome di **funzione interpolante** e gli x_i sono detti **nodi**.

2. Se $f(x) \in \mathcal{C}_{[a,b]}$ e $(x_i, y_i), i = 0, \ldots, n$ sono $(n+1)$ punti di campionamento della f, ci possiamo chiedere qual è la qualità dell'approssimazione di $f(x)$ fatta dalla funzione interpolante Φ .

Il primo caso è tipico di molte applicazioni in cui si ha a che fare con misure effettuate su un certo fenomeno fisico di cui però non si conosce il modello matematico. In questo caso i nodi rappresentano le misurazioni effettuate e Φ mira a descrivere l'andamento del fenomeno.
Il secondo caso è in sostanza una possibile soluzione al problema dell'approssimazione di una funzione f con una funzione Φ che in genere si vuole più facile da trattare rispetto alla f (si pensi ad esempio a problemi di integrazione e derivazione).

Nel seguito, se non altrimenti specificato, si farà sempre riferimento al caso unidimensionale.

1.1 Il caso generale

In generale, dati $(n+1)$ punti (x_i, y_i), $i = 0, \ldots, n$ cercheremo la funzione interpolante Φ in un sottospazio di funzioni a dimensione finita. In tal modo, fissata una base $\{\phi_i\}$, potremo scrivere Φ come combinazione lineare delle ϕ_i:

$$\Phi(x) = \sum_{j=0}^{n} a_j \phi_j(x) \tag{1}$$

con gli a_j come coefficienti. Facciamo ora vedere che gli a_j sono in realtà le incognite del nostro problema. Affinchè sia interpolante, la Φ deve verificare le condizioni di interpolazione

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

Sostituendo nella (1) otteniamo:

$$\Phi(x_i) = \sum_{j=0}^{n} a_j \phi_j(x_i) = y_i \qquad i = 0, \ldots, n \tag{2}$$

La (2) è in realtà un sistema lineare di $(n+1)$ equazioni in $(n+1)$ incognite che ha la forma

$$\Phi \mathbf{a} = \mathbf{y}$$

dove $\phi_{ij} = \phi_j(x_i)$ è la matrice dei coefficienti e $\mathbf{a}$ è la soluzione del sistema. Quest'ultima esiste se e solo se

$$\det\Phi \neq 0 \qquad \text{(determinante di Haar)} \tag{3}$$

La (3) è detta **condizione di unisolvenza** e assicura che la funzione interpolante esista e sia unica.

2 Interpolazione polinomiale

Un caso molto importante è quello in cui le funzioni $\{\phi_i\}$ vengono scelte nello spazio Π_n dei polinomi di grado $\leq n$.
La scelta naturale è quella della base canonica $\{1, x, x^2, \ldots, x^n\}$ di Π_n, in modo da poter scrivere il **polinomio interpolante** come

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

Per quanto detto prima, il sistema lineare che ci permette di determinare i coefficienti del polinomio interpolante sarà del tipo $X\mathbf{a} = \mathbf{y}$ dove $y = (y_0, \ldots, y_n)$ è il vettore delle ordinate e X è una matrice detta **matrice di Vandermonde**, che si costruisce a partire dai nodi $\{x_i\}$ e ha la forma

$$X = \begin{pmatrix} 1 & x_0 & x_0^2 & \ldots & x_0^n \\ 1 & x_1 & x_1^2 & \ldots & x_1^n \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ 1 & x_n & x_n^2 & \ldots & x_n^n \end{pmatrix}$$

Un risultato importante è che, siccome

$$\det X = \prod_{i>j} (x_i - x_j)$$

il polinomio interpolante esiste ed è unico se e solo se i nodi sono tutti diversi fra loro. [1]

[1] Indicheremo con n il grado del polinomio interpolante su $(n+1)$ nodi

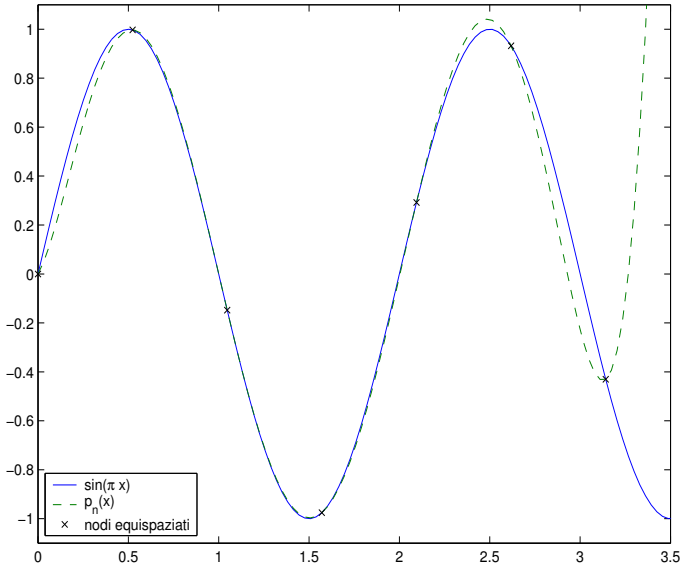

Figura 1: Interpolazione polinomiale canonica

2.1 canonical.m

La `function p = canonical(x, xi, yi)` [2] restituisce il polinomio interpolante $p_n(x)$ in forma canonica. I coefficienti di p_n vengono calcolati risolvendo il sistema lineare che ha come matrice dei coefficienti la matrice di Vandermonde relativa agli $\{x_i\}$ e come termine noto il vettore degli $\{y_i\}$.

```
% costruzione della matrice di Vandermonde
V = vander(xi);

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

% valutazione del polinomio sull'intervallo x
p = polyval(c, x);
```

Nello script `canonicalint.m` viene campionata una funzione test $f = sin(\pi x)$ su 6 nodi equispaziati e successivamente viene calcolato il polinomio interpolante $p_n(x)$ utilizzando la function `canonical`. La funzione test è valutata su 100 punti e interpolata con un polinomio di grado dispari per sfruttare la simmetria della f. Dalla figura 1 si può notare come il polinomio interpolante tenda ad oscillare in

[2] Nelle function implementate in questo documento, `x` è un vettore contentente i punti di valutazione, `xi` è il vettore dei nodi e `yi` è il vettore delle ordinate

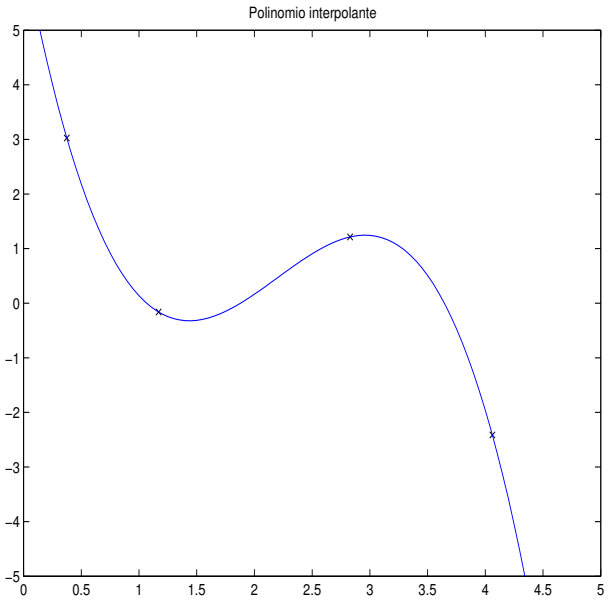

Figura 2: pointinteractive.m

maniera evidente quando non è più vincolato a passare per i punti di campionamento. In figura 1 è anche riportato l'errore di interpolazione nell'intervallo tra il primo e l'ultimo nodo (l'errore risulta pari a 0.2313). Inoltre, provando ad aumentare sensibilmente il numero dei nodi (quindi ad avvicinarli), la matrice di Vandermonde diventa numericamente singolare, mostrando il cattivo condizionamento della base canonica di Π_n.
Provando ad effettuare l'interpolazione su 18 nodi equispaziati, si ottiene da MATLAB il messaggio

```
Warning: Matrix is close to singular or badly scaled.
```

2.2 pointinteractive.m

Si tratta di uno script che permette all'utente di selezionare 4 punti su un piano cartesiano e successivamente plotta il polinomio interpolante $p_n(x)$ (vedi figura 2). Lo script utilizza la function `canonical`.

3 Rappresentazione del polinomio interpolante

La scelta della base canonica di Π_n per la rappresentazione del polinomio interpolante presenta dei problemi di carattere numerico, fra i quali il cattivo condizionamento della matrice di Vandermonde e della stessa base canonica, e un costo

computazionale dell'ordine di $O(n^3)$.
Per ovviare a questi problemi, esistono degli algoritmi che permettono di rappresentare in altra forma il polinomio $p_n(x)$.

3.1 Forma di Lagrange

Anzichè calcolare i coefficienti di $p_n(x)$, questi si scelgono uguali agli $\{y_i\}$ e si cerca una base per rappresentare p_n. Si trova che

$$p_n(x) = \sum_{i=0}^{n} y_i L_i(x)$$

dove gli $\{L_i\}$ sono detti **polinomi cardinali di Lagrange** e hanno la forma

$$L_i(x) = \prod_{\substack{k=0 \\ k \neq i}}^{n} \frac{x - x_k}{x_i - x_k} \tag{4}$$

Inoltre godono della proprietà che $L_i(x_j) = \delta_{ij}$.
Rappresentare $p_n(x)$ in forma di Lagrange richiede un costo computazionale pari a $O(n^2)$.

3.2 lagrix.m

La `function L = lagrix(x, xi)` costruisce la matrice

$$L = [L_0(x) L_1(x) \dots L_n(x)]$$

che ha per colonne le valutazioni degli $L_i(x)$ sui punti contenuti nell'intervallo `x`. La colonna i-esima di L si ottiene invocando la subfunction `l = givelagr(x, i, xi)` che restituisce un vettore `l` contenente il polinomio $L_i(x)$ valutato su `x`.

```
L( :, i ) = givelagr( x, i, xi );
```

La subfunction `givelagr` può operare su vettori `x` e costruisce $L_i(x)$ a partire dalla relazione (4) nel modo seguente:

```
for k = 1 : n
    if i ~= k
        fact = ( x - xi(k) ) ./ ( xi(i) - xi(k) );
        prd = prd .* fact;
    end
end
```

In figura 3 sono plottati i primi 3 polinomi di Lagrange.

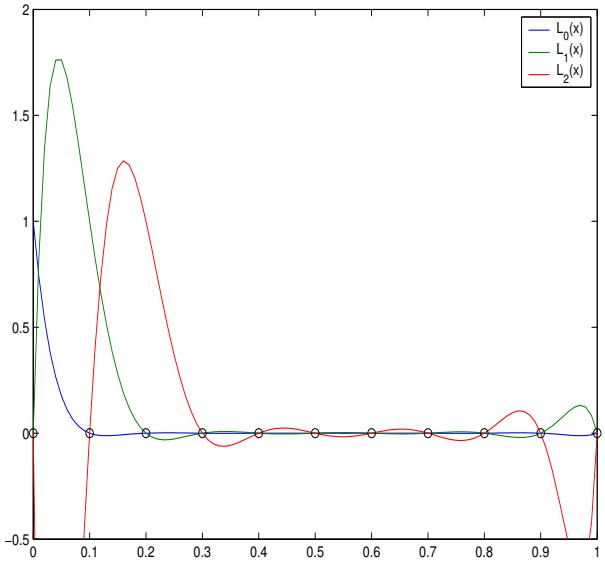

Figura 3: Polinomi cardinali di Lagrange

3.3 lagrange.m

La `function p = lagrange(x, xi, yi)` restituisce il polinomio interpolante $p_n(x)$ rappresentato in forma di Lagrange e valutato sui punti del vettore `x`. Per la costruzione di p_n viene usata la matrice L prodotta dalla function `lagrix`

```
L = lagrix(x, xi);
p = L * yi;
```

Lo script `lagrange_int.m` campiona la stessa funzione test di `canonical_int.m` ($f = sin(\pi x)$) su 6 nodi equispaziati ed interpola con il polinomio $p_n(x)$ posto in forma di Lagrange (vedi figura 4). L'errore di interpolazione risulta anche in questo caso pari a 0.2313, ma aumentando il numero dei nodi equispaziati a 18 (come nell'esempio precedente) non si presentano problemi di instabilità della rappresentazione.

3.4 Formula di Neville

Permette di costruire un polinomio interpolante $(k+1)$ punti.
Definiamo

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

come il polinomio che interpola sui nodi $x_r, x_{r+1}, \ldots, x_{r+k}$. Allora

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

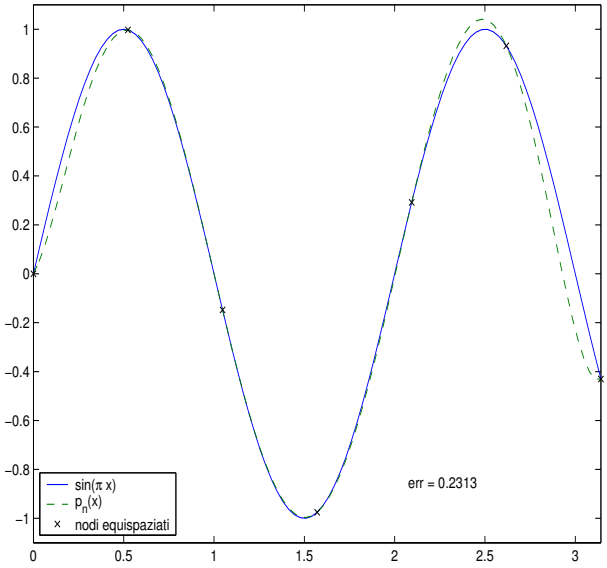

Figura 4: Polinomio interpolante in forma di Lagrange

E' possibile costruire una tabella, detta **tabella di Neville**, mediante la quale, partendo da $Q_r(x) = y_r$, si arriva a $Q_{r,r+1,\ldots,r+k}(x)$.

3.5 neville.m

La `function p = neville(x, xi, yi)` costruisce il polinomio interpolante $p_n(x)$ utilizzando la formula di Neville:

```
p(i) = giveneville(x(i), xi, yi);
```

Per ciascun punto del vettore `x` si invoca la function `s = giveneville(x, xi, yi)` che costruisce la tabella di Neville utilizzando un vettore `q`. L'assegnamento

```
q = yi;
```

corrisponde a porre $Q_r = y_r$. A partire da questo, la tabella viene costruita in maniera ricorsiva e al termine, il primo elemento del vettore `q` contiene il valore $Q_{0,1,\ldots,n}(x)$, che viene assegnato a `p(i)` dalla function `neville`.

3.6 Formula di Newton

Permette di rappresentare $p_n(x)$ nella forma

$$p_n(x) = f[x_0] + f[x_0, x_1](x - x_0) + f[x_0, x_1, x_2](x - x_0)(x - x_1) + \\ + \cdots + f[x_0, x_1, \ldots, x_n](x - x_0)(x - x_1)\ldots(x - x_{n-1})$$

dove i coefficienti sono le **differenze divise**

$$f[x_0, \ldots, x_n] = \frac{f[x_1, x_2, \ldots, x_n] - f[x_0, x_1, \ldots, x_{n-1}]}{x_n - x_0}$$

Anche in questo caso è possibile costruire una tabella, detta **tabella delle differenze divise**, che partendo da $f[x_i] = y_i$ permette di arrivare a $f[x_0, x_1, \ldots, x_n]$. L'algoritmo ha un costo computazionale dell'ordine di $O(n^2)$.

3.7 newton.m

La `function p = newton(x, xi, yi)` costruisce il polinomio interpolante $p_n(x)$ utilizzando la formula di Newton. Tale polinomio viene poi valutato in ciascun punto del vettore `x`.
La costruzione di p_n avviene in due fasi

```
% costruzione del vettore f delle differenze divise
f = givdifdiv(xi, yi);

% costruzione del polinomio interpolante
p = givehorner(x, xi, f);
```

Nella prima fase si costruisce la tabella delle differenze divise utilizzando un vettore `f`, in modo simile alla tabella di Neville. Questa operazione viene svolta dalla function `f = givdifdiv(x, y)` che costruisce il vettore `f` in modo che i suoi elementi siano le differenze divise

$$f[x_0] \quad f[x_0, x_1] \quad f[x_0, x_1, x_2] \quad \ldots \quad f[x_0, x_1, \ldots, x_n]$$

Nella seconda fase viene costruito il polinomio interpolante utilizzando l'algoritmo di Horner, implementato nella function `h = givehorner(x, xi, c)`, dove `c` è il vettore dei coefficienti del polinomio. L'algoritmo di Horner opera in questo modo:

```
n = length(c);
h = c(n);
for k = n-1 : -1 : 1
    h = h .* ( x - xi(k) ) + c(k);
end
```

4 Posizionamento dei nodi

Data una funzione f e il polinomio $p_n(x)$ interpolante $(n+1)$ nodi, possiamo definire la funzione

$$E_n(x) = f(x) - p_n(x)$$

che fornisce l'errore di interpolazione punto per punto. Andando a studiare la funzione E_n si trova che

$$\forall x \quad E_n(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!}\omega_n(x)$$

dove

$$\omega_n(x) = (x - x_0)(x - x_1)\dots(x - x_n)$$

e gli $\{x_i\}$ sono i nodi di interpolazione.
Questa relazione ci mostra che all'errore di interpolazione contribuiscono sia la funzione f che abbiamo campionato, sia la scelta dei nodi (attraverso il fattore $\omega_n(x)$). Il posizionamento e il numero dei nodi influisce quindi sull'errore $E_n(x)$ che si commette interpolando. D'altra parte un risultato dovuto a Weierstrass ci dice che se $f \in \mathcal{C}_{[a,b]}$ allora esiste sempre un polinomio $p \in \Pi_n$ che rende $||f - p_n||_\infty$ piccolo a piacere, ma non fornisce alcuna informazione sulla scelta dei nodi. Resta quindi il problema di come posizionare i nodi di interpolazione per minimizzare il loro contributo all'errore.

4.1 Nodi di Chebychev

Poichè il contributo all'errore di interpolazione dovuto ai nodi è contenuto nel fattore $\omega_n(x)$, possiamo provare a studiare il problema di minimax

$$\min_{\{xi\}} \max_{x\in[a,b]} \|\omega_n(x)\|$$

Studiando tale problema nell'intervallo $[-1, 1]$ si trova che prendendo come nodi gli zeri del polinomio

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

ottenuto calcolando $\cos(n+1)\theta$ e ponendo $x = \cos\theta$, il contributo all'errore di $\omega_n(x)$ cresce in maniera logaritmica (quindi lenta).
I nodi ottenuti in questo modo prendono il nome di **nodi di Chebychev** e costituiscono una scelta ottimale per l'interpolazione.

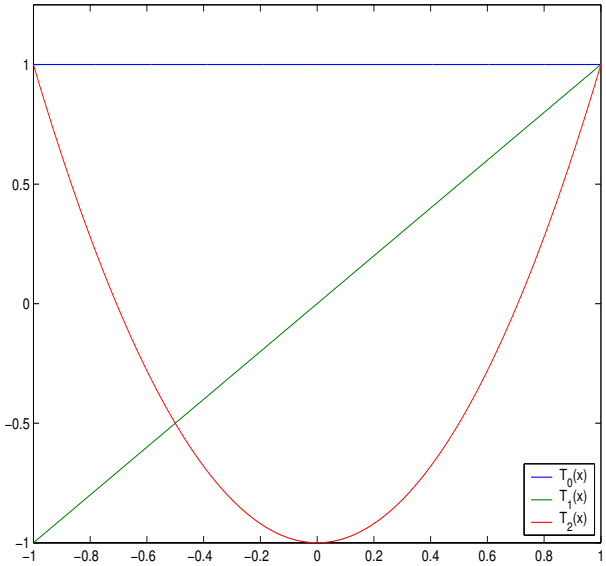

Figura 5: Polinomi di Chebychev

4.2 cheby.m

La `function xi = cheby(n)` restituisce un vettore `x` che contiene i primi n nodi di Chebychev. Il nodo k-esimo viene calcolato implementando la relazione

$$x_k = \cos\left(\frac{2k+1}{2n+2}\pi\right) \qquad k = 0, \ldots, n$$

nel modo seguente:

```
xi = [0:n-1];
xi = cos( (2*xi + 1)/(2*n + 2) * pi );
```

In figura 5 sono plottati i primi 3 polinomi di Chebychev.

Lo script `test_cheby.m` interpola la funzione $f = sin(\pi x)$ utilizzando sia nodi equispaziati che nodi di Chebychev. Per il calcolo dei polinomi interpolanti sono state usate le function `neville` e `newton`.
In figura 6 è mostrato l'errore di interpolazione al variare del grado del polinomio interpolante e del posizionamento dei nodi. Dal plot si nota che sebbene entrambi gli errori decrescano al crescere di n, l'errore dovuto all'utilizzo di nodi di Chebychev si mantiene più basso.
Risulta invece interessante verificare un caso in cui i nodi equispaziati si comportano molto male. Lo script `test_runge.m` interpola la funzione di Runge

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

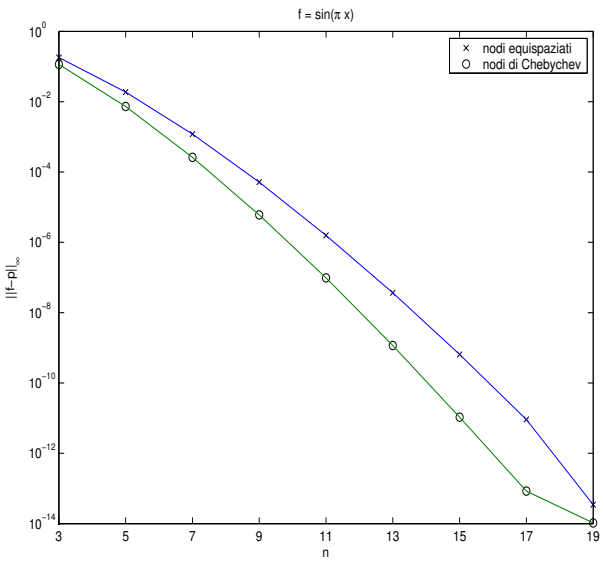

Figura 6: Errore di interpolazione in funzione del grado del polinomio interpolante

su nodi equispaziati e nodi di Chebychev. La funzione è stata valutata su 100 punti nell'intervallo $[-1, 1]$ e per il calcolo dei polinomi interpolanti sono state utilizzate le function `newton` e `neville`.
In figura 7 sono riportati i plot della funzione di Runge con i polinomi interpolanti e il plot dell'errore di interpolazione in funzione del grado del polinomio p_n, che è stato scelto pari per sfruttare la simmetria della funzione f.
Osservando il grafico a sinistra, si può notare come l'interpolazione su nodi di Chebychev si mantiene buona anche in prossimità degli estremi dell'intervallo, dove invece quella su nodi equispaziati tende ad oscillare vistosamente. Dall'esame dell'errore di interpolazione si vede immediatamente che l'errore dovuto all'utilizzo di nodi equispaziati cresce velocemente al crescere di n, al contrario di quello dovuto ai nodi di Chebychev che decresce in maniera rapida.

5 Interpolazione 2d

Data una funzione reale di due variabili reali $z = f(x, y)$ e una griglia equispaziata su $[a, b] \times [c, d]$ ($x \in [a, b]$, $y \in [c, d]$), supponiamo di conoscere i valori

$$f_{ij} = f(x_i, y_j) \quad j = 0, \dots n$$

Allora il polinomio

$$p(x, y) = \sum_{i=0}^{n} \sum_{j=0}^{n} f_{ij} L_i^{(1)}(x) L_j^{(2)}(y) \tag{5}$$

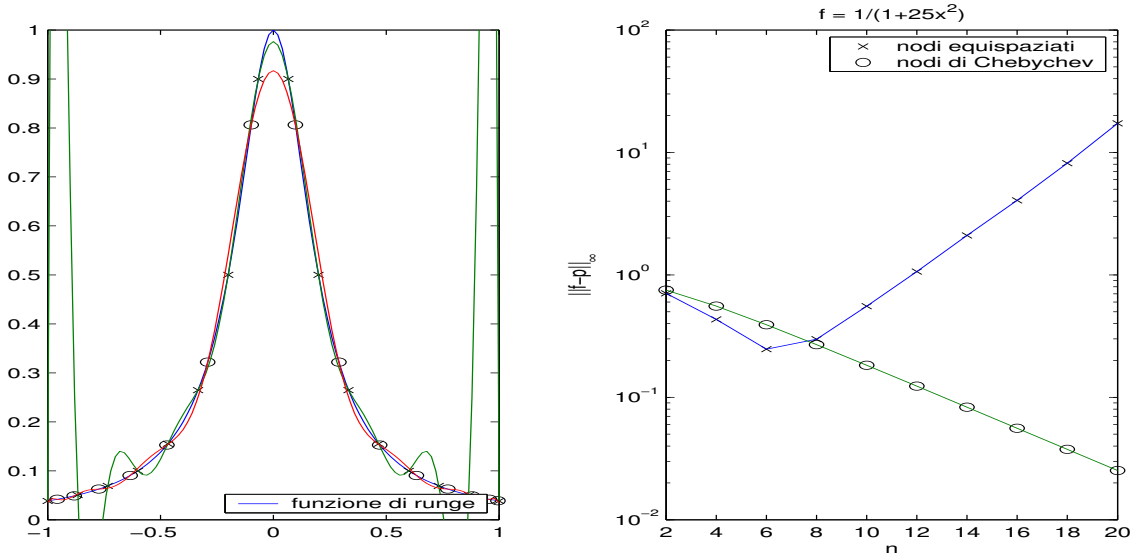

Figura 7: Interpolazione della funzione di Runge e grafico dell'errore in funzione del grado di p_n

interpola f sulla griglia, cioè $p(x, y) = f_{ij}$. I polinomi $L_i^{(1)}(x)$ e $L_j^{(2)}(y)$ sono costruiti rispettivamente sui punti $\{x_i\}$ e $\{y_i\}$. Un problema potrebbe essere che il polinomio $p(x, y)$ è un polinomio di grado n in x e di grado n in y, ma non di grado n^2 in x e in y. La function `lag2d` è una implementazione di questo algoritmo di interpolazione.

Un altro approccio al problema è il seguente: sulla stessa griglia equispaziata si considerano i polinomi $q_j(x)$, $j = 0, \ldots n$. Ciascuno di questi polinomi interpola i valori $\{f_{0j}, \ldots, f_{nj}\}$ sui punti $\{(x_0, y_j), \ldots, (x_n, y_j)\}$. In pratica si tiene y_j fissato e si costruisce un polinomio per ciascuna riga della griglia. Successivamente, per valutare $p(\overline{x}, \overline{y})$ su un punto generico, si valuta $q_j(\overline{x})$, $j = 0, \ldots, n$ e si interpolano su y i punti ottenuti. La function `newnev2d` implementa questo algortimo di interpolazione costruendo i polinomi interpolanti con le formule di Neville e di Newton.

5.1 lag2d.m e newnev2d.m

La `function P = lag2d(x, y, xi, yi, F)` data la matrice `F` che contiene la valutazione di una funzione f sulla griglia equispaziata restituisce la matrice `P` che rappresenta la superficie interpolante f sulla griglia. L'algortimo implementa la relazione (5) che definisce il polinomio $p(x, y)$ utilizzando le variabili `insum` e `outsum` nelle quali vengono memorizzati rispettivamente i risultati della sommatoria interna e di quella esterna:

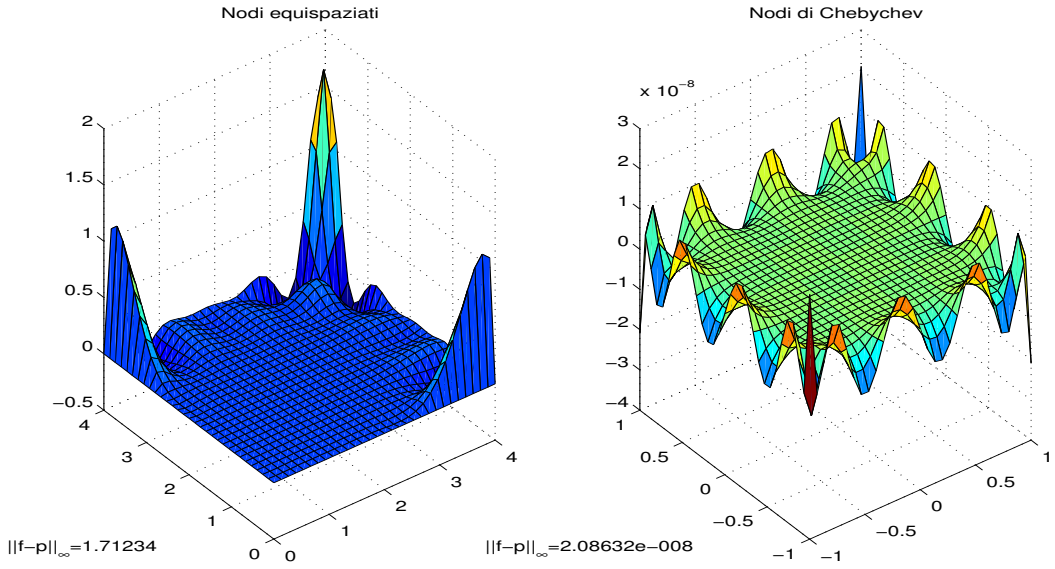

Figura 8: newnev2d, errore di interpolazione punto per punto

```
outsum = 0;
for i = 1 : n
    insum = 0;
    for j = 1 : n
        insum = insum + f(i, j) * Ly(:, j);
    end
outsum = outsum + insum * Lx(:, i);
end
```

Questa function utilizza `lagrix` per costruire le matrici `Lx` e `Ly`:

```
Ly = lagrix(y, yi);
Lx = lagrix(x, xi);
```

La function `P = newnev2d(x, y, xi, yi, F)` costruisce, analogamente a `lag2d`, la superficie interpolante `P` ma lo fa utilizzando polinomi interpolanti costruiti sulla griglia di valutazione. Per la costruzione di questi polinomi si utilizzano le function `neville` e `newton`. La matrice `P` si ottiene in questo modo:

```
n = length(x); ni = length(yi);

% costruzione matrice Q
for i = 1 : ni
    Q(:, i) = newton(x, xi, F(:, i));
```

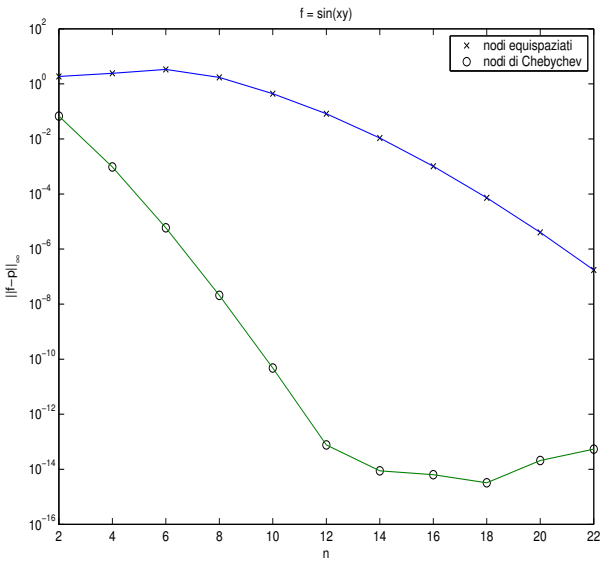

Figura 9: newnev2d, errore di interpolazione in funzione di n

```
end

% calcolo superficie interpolante P
for i = 1 : n
    p = neville(y, yi, Q(i, :));
    P(i, :) = p';
end
```

Lo script `test2d.m` interpola la funzione test $z = sin(xy)$ su una griglia $[0, 4] \times [0, 4]$ costituita da 30 punti di valutazione per lato. La griglia dei nodi è costruita utilizzando sia nodi equispaziati che nodi di Chebychev. Il numero dei nodi è pari a 8. Questo script utilizza la function `newnev2d`. In figura 8 sono raffigurati i plot dell'errore nel caso di nodi equispaziati (a sinistra) e di Chebychev (a destra).
In figura 9 è invece plottato il grafico dell'errore di interpolazione $||f - p||_\infty$ al variare del numero dei nodi n. Come ci si aspetta, l'errore nel caso di nodi equispaziati diminuisce più lentamente rispetto al caso di nodi di Chebychev.