

# Interpolazione e minimi quadrati

Laboratorio di Calcolo Numerico

Annalisa Buffa Fausto Cavalli

Corso di Laurea in Chimica

A.A. 2009-2010

# Polinomi

In Matlab esistono molte funzioni che permettono di lavorare con i polinomi (per un elenco `help polyfun`).

Matlab non ha un oggetto particolare di tipo polinomio, ma rappresenta i **polinomi attraverso vettori** che contengono i coefficienti del polinomio stesso, ordinati partendo dal coefficiente del termine di grado maggiore.

Per esempio

$$p(x) = -3x^4 + 2x^3 + x - 5$$

diventa il vettore

```
>> p=[-3 2 0 1 -5];
```

Tutte le funzioni che lavorano su polinomi richiedono che il polinomio sia quindi prima “tradotto” in un vettore. Esistono funzioni che permettono di costruire un polinomio a partire da alcuni dati, di valutare un polinomio su un insieme di punti, di effettuare operazioni fra polinomi e di ottenere il polinomio primitiva o derivata.

# Polinomi

Per esempi la funzione **roots** permette di ricavare le radici di un polinomio, usando metodi numerici di approssimazione.

```
>> roots(p)
```

Al contrario la funzione **poly** restituisce il polinomio (cioè il suo vettore dei coefficienti) che ha come radici un vettore di zeri dato. Se vogliamo per esempio ottenere un polinomio le cui radici siano  $-3, 2, 1, 0$  (che sarà quindi di quarto grado) basta dare

```
>> r=[-3 2 1 0];
```

```
>> p=poly(r)
```

Per valutare un polinomio in un insieme di punti c'è il comando **polyval**, che può essere utile anche per disegnare un polinomio. Per valutare il polinomio  $p$  nei punti  $x$  si scrive

$$y = polyval(p,x)$$

# Interpolazione

## Interpolazione

Con il termine di interpolazione si intende il procedimento che permette di sostituire ad una funzione data un'altra, in genere di tipo più semplice, che passa per alcuni dei suoi punti.

Un esempio molto usato di interpolazione è **l'interpolazione polinomiale**, in cui la classe di funzioni che si usano sono i polinomi. In particolare considereremo **l'interpolazione polinomiale uniforme**, quella in cui i punti che si considerano sono ottenuti suddividendo il dominio in parti uguali.

In matlab esiste il comando **polyfit** che permette di ottenere il polinomio di interpolazione dati i punti di interpolazione. La sua sintassi è

$$p = \text{polyfit}(x,y,n)$$

dove  $n$  è il grado del polinomio che vogliamo ottenere.

# Interpolazione

## Attenzione

Nel comando `polyfit` è necessario che i vettori delle ascisse  $x$  e delle ordinate  $y$  abbiano la stessa lunghezza. Inoltre bisogna prestare attenzione al valore di  $n$ . Infatti se si vuole ottenere un polinomio di interpolazione è necessario che  $n = \text{length}(x) - 1$ : per esempio se uso 2 punti di interpolazione trovo una retta, che è un polinomio di grado 1, se uso 3 punti trovo una parabola, che ha grado 2...

Se scegliessi  $n < \text{length}(x) - 1$  non troverei il polinomio di interpolazione ma il polinomio di migliore approssimazione nel senso dei minimi quadrati. Si noti che anche per  $n = \text{length}(x) - 1$  si ha il polinomio di migliore approssimazione nel senso dei minimi quadrati, che in questo caso particolare coincide con il polinomio di interpolazione.

# Interpolazione

## Esercizio

Si costruisca un file di tipo script in cui si interpoli la funzione  $f = \arctan(x)$  sull'intervallo  $[0, 10]$  su un numero  $n$  di nodi equispaziati. Per  $n = 1, \dots, 6$  e si disegni la funzione e il polinomio interpolatore sovrapposti, indicando anche con un asterisco i nodi di interpolazione.

Per ripetere più volte lo stesso gruppo di istruzioni al variare di un valore (in questo caso al variare di  $n$ ) si usa l'istruzione `for`, che ha la seguente sintassi:

```
for i=inizio:fine  
    comandi da ripetere al variare di i  
end
```

In matlab, come nella maggior parte dei linguaggi di programmazione, **for** e **while** permettono entrambi di ripetere più volte un gruppo di istruzioni: si differenziano dato che **while** si ripete fino a che una data condizione è verificata (a priori non so quante volte si ripeterà il ciclo), mentre con **for** conosco da subito quante volte ripetere il ciclo, in dipendenza da una valore che varia di volta in volta.

# Interpolazione

## Struttura dello script

La prima parte dello script consiste nell'inizializzazione di alcuni valori

- Si definisca la funzione `f=inline('atan(x)')`
- Si crei una nuova suddivisione `xx` di  $[a, b]$  con molti punti (per esempio 300): la userò per disegnare.
- Si inizializzino i valori di `a, b` con i valori degli estremi dell'intervallo  $[0, 10]$
- Si ponga `n=1:6`

La seconda parte dello script si deve occupare di calcolare il polinomio di interpolazione al variare del valore di `n` e sarà tutta contenuta nel ciclo:

- Per `i` che va da 1 a `length(n)`
  - Si costruiscano le ascisse `x` dei nodi di interpolazione con il comando `linspace`, relativo ad `a` e `b` con `n(i)` punti
  - Si calcolino le ordinate dei nodi di interpolazione con `y=f(x)`
  - Si usi il comando `polyfit` per costruire il polinomio di interpolazione relativo a `x, y` di ordine `n(i)-1`

# Interpolazione

L'ultima parte deve disegnare il tutto: per poter disegnare è necessario utilizzare un dominio più ricco di punti rispetto alla variabile  $x$ , quello creato prima nella variabile  $xx$

- Si calcolino le ordinate dei punti  $xx$  relativamente alla funzione  $f$  con  $fy=f(xx)$
  - Si usi `polyval` per calcolare il valore delle ordinate  $py$  dei punti  $xx$  relativamente al polinomio  $p$
  - Si disegni il tutto: si può usare il comando `plot(xx,fy,'b',xx,py,'r',x,y,'k*');`
  - All'interno di un ciclo, per vedere di volta in volta un grafico, è necessario "fermare" l'esecuzione con il comando `pause`, altrimenti vedrei solo l'ultimo grafico disegnato. Quando eseguo lo script, l'esecuzione si blocca dopo il disegno e per riprenderla devo premere un tasto.
- `end`, a concludere il ciclo `for`

# Interpolazione

Al crescere del valore di  $n$  si vede che il polinomio interpolatore diventa sempre più vicino alla funzione. Ci si può chiedere se questa sia una caratteristica generale: purtroppo no, come mostra il seguente esempio

## Esercizio

Si ripeta l'esercizio precedente considerando come funzione

$f(x) = \frac{1}{1+x^2}$  (detta funzione di Runge) relativamente all'intervallo  $[-5, 5]$ . Si considerino valori di  $n = 1 : 2 : 15$

Quali considerazioni si possono fare in questo caso? Come si comporta il polinomio di interpolazione vicino agli estremi dell'intervallo quando il valore di  $n$  cresce?

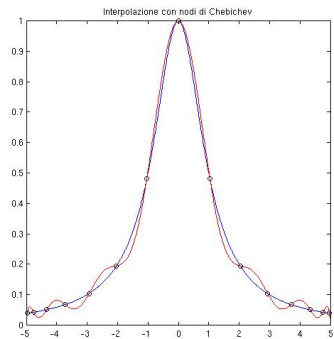
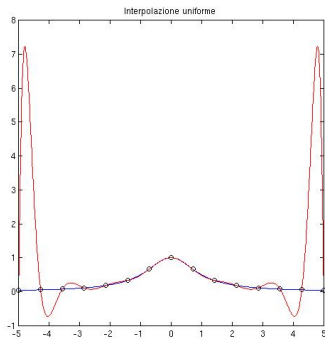
# Interpolazione

Come visto l'interpolazione usando nodi equispaziati non sempre dà buoni risultati al crescere del grado del polinomio. Per fornire delle approssimazioni che siano sempre più accurate al crescere del valore di  $n$  è necessario “scegliere” in maniera più accurata la disposizione dei nodi. Fortunatamente esiste una disposizione di nodi indipendente dalla funzione data che permette di ottenere ciò: sono i nodi di Chebichev, la cui espressione è

$$x_i = (a + b)/2 + \frac{(a - b)}{2} \cos \frac{2i - 1}{2n} \pi$$

Nella pagina seguente c'è un confronto sulla funzione di Runge fra l'interpolazione uniforme e quella con i nodi di Chebichev, usando in entrambi i casi polinomi di grado 15.

# Interpolazione



## Migliore approssimazione

Il problema della migliore approssimazione nel senso dei minimi quadrati può essere risolto in Matlab ancora con il comando `polyfit`. In questo caso il risultato non è un polinomio che passa per i punti dati (interpolazione), ma un polinomio, in genere di grado basso, che cerca di “ricostruire” secondo un dato criterio l’insieme dei dati, magari grazie al fatto che si conosce a priori che tipo di andamento i dati devono avere. Vediamo di costruire con Matlab un esempio di applicazione della migliore approssimazione nel senso dei minimi quadrati: supponiamo di avere delle misurazioni di un dato  $y$  effettuate ad intervalli regolari  $x$  di ampiezza 0.01 su  $[0,1]$ , cioè

```
>> x=0:0.01:1;
```

Supponiamo che la legge che lega  $y$  ad  $x$  sia

$$y = 1.5x - 2$$

e simuliamo la presenza di errori sperimentali attraverso il comando **`rand(1,length(x))`**, che crea un vettore di numeri casuali (tra 0 ed 1) della stessa lunghezza di  $x$ .

## Migliore approssimazione

Per avere numeri casuali compresi tra -0.5 e 0.5 basta sottrarre 0.5 al vettore precedente. Infine moltiplichiamo il vettore ottenuto per 0.2 per evitare che la perturbazione sia “troppo grande” rispetto ai dati:

```
>> pert=((rand(1,length(x)))-0.5)*0.2;  
>> y=1.5*x-2+pert;
```

A questo punto per ottenere la migliore approssimazione con un polinomio di grado 1 (i dati di partenza stavano su una retta) basta dare il comando

```
>> p=polyfit(x,y,1)  
p =  
1.4890    -1.9906
```

dove i risultati saranno ovviamente sempre diversi dipendendo dal comando rand. Si ha un buon accordo con i valori “giusti” 1.5 e -2.

# Migliore approssimazione

Ecco un esempio di un esito dell'applicazione precedente

