a cura di Corrado Giustozzi

Metodi di Approssimazione

Le unità aritmetico-logiche dei computer realizzano direttamente solo le 4 operazioni aritmetiche (e a volte neppure tutte), pertanto le sole funzioni matematiche effettivamente calcolabili sono solo quelle razionali (cioè quelle esprimibili con un numero finito di operazioni aritmetiche). Le altre funzioni (ad esempio l'esponenziale o il coseno) possono essere solo approssimate. In questo articolo, vediamo, con l'aiuto di Mathematica, alcune tecniche di approssimazione di funzioni sia sfruttando specifiche proprietà matematiche delle funzioni stesse che a partire da valori numerici precalcolati

di Francesco Romani

Introduzione

Il problema della approssimazione di funzioni è di grande importanza nella matematica computazionale sia per il suo interesse teorico che per la vastità delle applicazioni. A grandi linee esistono due tipi di situazioni che si possono presentare:

1) trovare algoritmi che, facendo uso delle quattro operazioni, permettano di calcolare in modo semplice, in un intervallo predeterminato, funzioni complicate con un errore più piccolo possibile. Questa esigenza si presenta per esempio nella scrittura di librerie scientifiche o dei programmi da inserire nelle ROM dei coprocessori matematici e dei DSP.

2) trovare funzioni semplici che approssimano con un errore piccolo un insieme di dati tabulati proveniente da sperimentazioni scientifiche, rilevazioni statistiche, misurazioni, calcoli matematici di vario genere, etc.

Lo scopo che si prefigge questo articolo è triplice, si intende:

1) illustrare da un punto di vista teorico alcune delle più semplici tecniche di approssimazione di funzioni e di dati tabulati;

2) mostrare come simili calcoli possono essere effettuati con *Mathematica*, come aiuto per chi deve effettuare calcoli numerici;

3) far vedere alcuni esempi di elaborazione simbolica in *Mathematica*, come aiuto per chi deve effettuare calcoli simbolici.

La trattazione è limitata ad alcuni degli approcci più semplici; la materia è molto vasta e si presta bene ad essere affrontata con *Mathematica*. Gli argomenti tralasciati (approssimazione trigonometrica, splines, approssimazione minimax, fitting non polinomiale, problemi di stabilità, etc.) protanno essere affrontati in (numerose) puntate successive.

Formule di Taylor

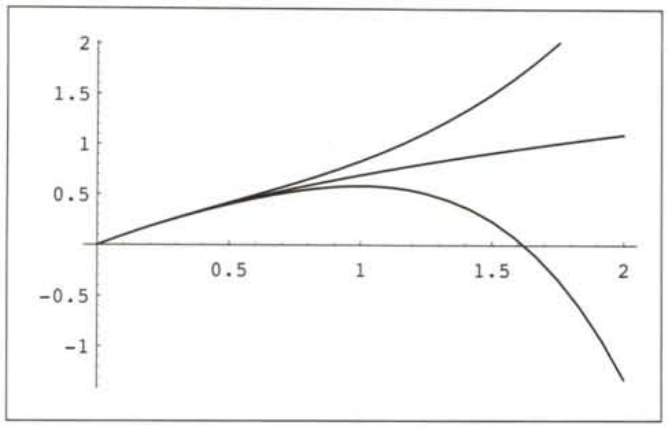
Come insegna l'analisi matematica, se una funzione è derivabile più volte in un punto x_0 è possibile approssimare i suoi valori in un intorno di x_0 in funzione dei valori che la funzione e le sue derivate assumono in x_0 per mezzo della formula di Taylor. In *Mathematica* si può ottenere direttamente lo sviluppo di Taylor. Per esempio vediamo i ben noti sviluppi dell'esponenziale e delle altre funzioni trascendenti elementari:

```
In[1]:=
  Series[Exp[x],{x,0,6}]
Out[1]=
                 2   3   4    5    6
                x   x   x    x    x
1 + x + O[x]^7 + -- + -- + -- + --- + ---
                2   6   24   120  720
In[2]:=
  Series[Log[1+x],{x,0,6}]
Out[2]=
                2   3   4   5   6
               x   x   x   x   x
x + O[x]^7 - -- + -- - -- + -- - --
               2   3   4   5   6
In[3]:=
  Series[Cos[x],{x,0,6}]
Out[3]=
                2   4    6
               x   x    x
1 + O[x]^7 - -- + -- - ---
               2   24   720
In[4]:=
  Series[Sin[x],{x,0,6}]
Out[4]=
                3   5
               x   x
x + O[x]^7 - -- + ---
               6   120
```

Dalla formula di Taylor troncata ad un certo termine si ottiene un polinomio che approssima la funzione sotto esame tanto meglio quanto più siamo vicini al punto x_0. Allontanandosi dal punto x_0 l'approssimazione tende a peggiorare. Disegniamo ora il grafico del logaritmo naturale di 1+x tra 1 e 3 e delle approssimazioni ottenute troncando la formula dopo 3 e 4 termini. La funzione **Normal** trasforma una serie in un polinomio rimuovendo l'espressione **O[...]** che rappresenta il resto.

```
In[5]:=
  f=Log[1+x];
In[6]:=
  a3=Normal[Series[f,{x,0,3}]]
Out[6]=
                2   3
               x   x
x - -- + --
               2   3
In[7]:=
  a4=Normal[Series[f,{x,0,4}]]
Out[7]=
                2   3   4
               x   x   x
x - -- + -- - --
               2   3   4
```

In[8]:=
```
Plot[{f,a3,a4},{x,0,2}];
```



La funzione esatta è quella al centro. Le due approssimazioni polinomiali scappano presto via (verso + ∞ quella di terzo grado e verso - ∞ quella di quarto grado). È evidente come ad una certa distanza dal punto x_0 l'approssimazione divenga totalmente inaccettabile.

Interpolazione polinomiale

L'analisi matematica ci insegna che data una funzione e un insieme di n+1 ascisse **distinte** $\{x_0,x_1,...,x_n\}$ e un insieme di valori $\{f_0, f_1,..., f_n\}$ esiste un unico polinomio p(x) di grado al più n che passa per i punti (x_i, f_i). Vediamo una dimostrazione di questo fatto ottenuta con *Mathematica* per il caso particolare n=2.

Sia **pol** un generico polinomio di secondo grado.
In[1]:=
```
pol=a x^2 + b x + c
```
Out[1]=
```
c + b x + a x^2
```

Nei nodi x_0, x_1, x_2 esso assume i valori
In[2]:=
```
rows=pol/.{{x->x0},{x->x1},{x->x2}}
```
Out[2]=
```
{c+b x0+a x0^2 , c+b x1+a x1^2 , c+b x2+a x2^2 }
```

Noi vogliamo che questi valori coincidano con quelli f_0, f_1, f_2 assegnati dal problema, cioè il vettore **rows** e il vettore **{f0,f1,f2}.** Per questo scriviamo le tre equazioni facendo il prodotto interno (**Inner**) tra le tre espressioni (**rows**) e i termini noti utilizzando al posto della moltiplicazione l'operatore di uguaglianza (**Equal**) e al posto della somma **List**.
In[3]:=
```
ColumnForm[Inner[
  Equal, rows,{f0,f1,f2}, List]]
```
Out[3]=
```
c + b x0 + a x0^2  == f0
c + b x1 + a x1^2  == f1
c + b x2 + a x2^2  == f2
```

Abbiamo così ottenuto un **sistema lineare** di 3 equazioni nelle tre incognite a,b,c. Anche la matrice del sistema si può ricavare da **rows** sostituendo per tre volte 1 al posto di una incognita e 0 al posto delle altre.

In[4]:=
```
M = Transpose[rows/.
          {{a->1,b->0,c->0},
           {a->0,b->1,c->0},
           {a->0,b->0,c->1}}];
MatrixForm[M]
```
Out[4]=
```
x0^2    x0    1

x1^2    x1    1

x2^2    x2    1
```

Questa matrice (nota come matrice di Vandermonde) è una vecchia conoscenza dei matematici.

È noto che un sistema lineare ammette una e una sola soluzione se e solo se il determinante della matrice è diverso da zero. Nel caso di **M** si ha
In[5]:=
```
det = Det [M]
```
Out[5]=
```
x0^2 x1 - x0 x1^2 - x0^2  x2 + x1^2  x2 +

   x0 x2^2 - x1 x2^2
```
In[6]:=
```
Factor[det]
```
Out[6]=
```
(-x0 + x1) (x0 - x2) (-x1 + x2)
```

Questa fattorizzazione ci assicura che il determinante della matrice è non nullo se e solo se i nodi della interpolazione sono distinti, (ipotesi questa che avevamo chiaramente posto all'inizio).

Anche nel caso generale in cui i nodi sono n+1 la matrice del sistema è di Vandermonde ed è stato dimostrato per via teorica che il suo determinante è non nullo se e solo se i punti sono distinti.

Vediamo un esempio nel caso particolare in cui i nodi siano x_0 =0, x_1 =1, x_2 =2. La matrice risulta
In[7]:=
```
M0=M/.{x0->0,x1->1,x2->2}
```
Out[7]=
```
{{0, 0, 1}, {1, 1, 1}, {4, 2, 1}}
```

e il sistema può essere risolto simbolicamente ottenendo i valori di a, b e c in funzione di f_0, f_1, f_2.
In[8]:=
```
cofs=LinearSolve[M0, {f0,f1,f2}]
```
Out[8]=
```
 -(-3 f0 + 4 f1 - 2 (-f0 + f1) - f2)
{-------------------------------------,
                    2

  -(3 f0 - 4 f1 + f2)
  -------------------, f0}
          2
```

Moltiplicando i coefficienti per il vettore $\{x^2,x,1\}$ si ottiene il polinomio interpolante corrispondente alla parabola che in 0,1,2 passa rispettivamente per f_0, f_1, f_2.

In[9]:=
```
Expand[cofs.{x^2,x,1}]
```
Out[9]=
```
                3 f0 x                    f2 x
f0 - ------------------ + 2 f1 x - ------------ +
                2                         2

     f0 x^2                    f2 x^2
   ---------- - f1 x^2  + ----------
      2                         2
```

Un altro metodo, più diretto, per calcolare il polinomio interpolante si serve della formula di Lagrange:

$$p(x)=\sum_{i=0}^{n} f_i L_i(x) \quad \text{dove} \quad L_i(x)=\prod_{\substack{j=0\\ j\neq i}}^{n} \frac{x-x_j}{x_i-x_j}$$

Notando che le funzioni $L_i(x)$ sono polinomi di grado n che valgono 1 nel punto x_i e 0 nei punti x_j con $i\neq j$, è possibile verificare facilmente che p(x) vale f_i nei punti x_i. La funzione **Lagrange[asc,i,x]** implementa $L_i(x)$ per l'insieme di punti **asc.**

In[11]:=
```
Lagrange[asc_,i_,x_]:=Product[
 If[i=!=j,
    (x-asc[[j]])/(asc[[i]]-asc[[j]]),
    1],
 {j,Length[asc]}]
```
In[12]:=
```
Lagrange[{x1,x2,x3},2,x]
```
Out[12]=
```
(x - x1) (x - x3)
-----------------
(-x1 + x2) (x2 - x3)
```

La funzione **Interpol[asc,data,x]** calcola p(x) per l'insieme di punti **asc** e di valori **data.**

In[13]:=
```
Interpol[asc_,data_,x_]:=Module[
 {n=Length[data]},
  Expand[Sum[
   Lagrange[asc,i,x] data[[i]],
         {i,n}]]]
```

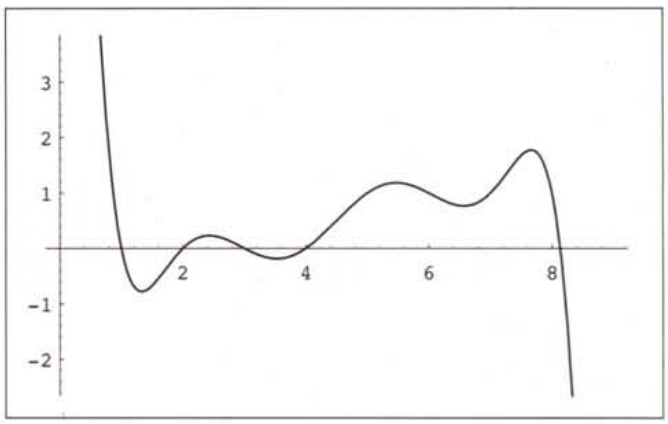
Naturalmente poiché il polinomio interpolante è unico con le ascisse {0,1,2} si ottiene lo stesso risultato di prima.

In[14]:=
```
Interpol[{0,1,2},{f0,f1,f2},x]
```
Out[14]=
```
                3 f0 x                    f2 x
f0 - ------------------ + 2 f1 x - ------------ +
                2                         2

     f0 x^2                    f2 x^2
   ---------- - f1 x^2  + ----------
      2                         2
```

Quando i punti da interpolare sono molti il polinomio di interpolazione risulta di grado elevato e tende a oscillare molto, cosa che in generale non è opportuna perché una funzione fortemente oscillante non fornisce una buona approssimazione di molte funzioni che si incontrano nella pratica. Vediamo come si riesce ad approssimare un gradino con un polinomio di nono grado:

In[15]:=
```
pol=Interpol[{1,2,3,4,5,6,7,8},
             {0,0,0,0,1,1,1,1},x]
```
Out[15]=
```
                36781 x    669 x^2    2903 x^3    43 x^4
35 - ------------------ + --------- - ---------- + ------ -
                420          8           72          4

     577 x^5    x^6     x^7
   ---------- + --- - ------
      360        8      252
```

In[16]:=
```
Plot[pol,{x,0,9}];
```



Approssimazione ai minimi quadrati di dati tabulati

A volte capita che i valori numerici della funzione da approssimare siano risultati di esperimenti fisici o rilevazione statistiche o comunque dati affetti da errori. In questo caso una interpolazione polinomiale di grado elevato porterebbe a risultati privi di significato. Inoltre, se per lo stesso valore della variabile in ascissa vengono ripetuti più esperimenti che a causa del "rumore" portano dati diversi tra loro, l'interpolazione risulta chiaramente impossibile. Un approccio alternativo è quello di cercare una funzione semplice (ad esempio una retta o un polinomio di grado basso) che passa "vicino" ai punti da approssimare. Tipicamente se p(x) è la funzione approssimante cercata si cerca di minimizzare l'errore quadratico:

$$err=\sum_{i=1}^{n} (p(x_i)-f_i)^2.$$

Anche in questo caso usiamo *Mathematica* per derivare la soluzione nel caso particolare di un polinomio di secondo grado e 4 punti da approssimare. Si costruiscono le quattro espressioni $p(x_i)-f_i$, i=1,2,3,4 nelle tre incognite a, b, c, che sono i coefficienti di p(x)

In[1]:=
```
pol=a x^2 + b x +c;
rows = pol /.
 {{x->x1},{x->x2},{x->x3},{x->x4}};
ColumnForm[rows]
```
Out[1]=

```
c + b x1 + a x1^2
c + b x2 + a x2^2
c + b x3 + a x3^2
c + b x4 + a x4^2
```

Per costruire la funzione quadratica **err** da minimizzare, usiamo la funzione pura **(#1-#2)^2&** che realizza in modo abbreviato la funzione $d(x,y)=(x-y)^2$.

In[2]:=

```
err=Inner[(#1-#2)^2&, rows,
          {f1,f2,f3,f4}, Plus]
```

Out[2]=

```
(c - f1 + b x1 + a x1^2)^2  +

  (c - f2 + b x2 + a x2^2)^2  +

  (c - f3 + b x3 + a x3^2)^2  +

  (c - f4 + b x4 + a x4^2)^2
```

Dalla teoria è noto che una forma quadratica ammette un solo punto stazionario in cui le derivate parziali rispetto alle incognite a, b, c si annullano e che corrisponde al punto di minimo. Ne risulta un sistema lineare di 3 equazioni in tre incognite.

In[3]:=

```
eq1={D[err,a],D[err,b],D[err,c]}
```

Out[3]=

```
{2 x1^2  (c - f1 + b x1 + a x1^2 ) +

   2 x2^2  (c - f2 + b x2 + a x2^2 ) +

   2 x3^2  (c - f3 + b x3 + a x3^2 ) +

   2 x4^2  (c - f4 + b x4 + a x4^2 ),

  2 x1 (c - f1 + b x1 + a x1^2 ) +

   2 x2 (c - f2 + b x2 + a x2^2 ) +

   2 x3 (c - f3 + b x3 + a x3^2 ) +

   2 x4 (c - f4 + b x4 + a x4^2 ),

  2 (c - f1 + b x1 + a x1^2 ) +

   2 (c - f2 + b x2 + a x2^2) +

   2 (c - f3 + b x3 + a x3^2) +

   2 (c - f4 + b x4 + a x4^2)}
```

Si può inoltre dimostrare che questo sistema può essere scritto (a meno di un fattore 2), in forma sintetica come

$$M^T M\, \mathbf{v} = M^T \mathbf{f}$$

dove M è la matrice rettangolare di Vandermonde.

In[4]:=

```
Mat[x_]:= Table[x^i,{i,2,0,-1}];
 SetAttributes[Mat,Listable];
MatrixForm[M=Mat[{x1,x2,x3,x4}]]
```

Out[4]=

```
x1^2    x1    1

x2^2    x2    1

x3^2    x3    1

x4^2    x4    1
```

v = {a,b,c} è il vettore delle incognite e **f** = {f1,f2,f3,f4} il vettore dei valori assegnati della funzione da approssimare. Il sistema può quindi essere scritto così:

In[5]:=

```
eq2=Expand[Transpose[M].(M.{a,b,c}-
{f1,f2,f3,f4})];
```

È facile scrivere un programma che costruisce questo sistema e lo risolve. Il prodotto scalare tra la soluzione del sistema e la lista delle funzioni base $\{x^2, x, 1\}$ dà il polinomio che minimizza la somma dei quadrati degli scarti

In[6]:=

```
fit2[asc_,data_]:=Module[{M,MT},
     M=Mat[asc];
     MT=Transpose[M];
     LinearSolve[MT.M,MT.data].{x^2,x,1}]
```

In[7]:=

```
asc={1,2,3,4,5};
data={3.3,2.4,2.6,4.9,5.5};
```

In[8]:=

```
fit2[asc,data]
```

Out[8]=

```
4.22 - 1.49571 x + 0.364286 x^2
```

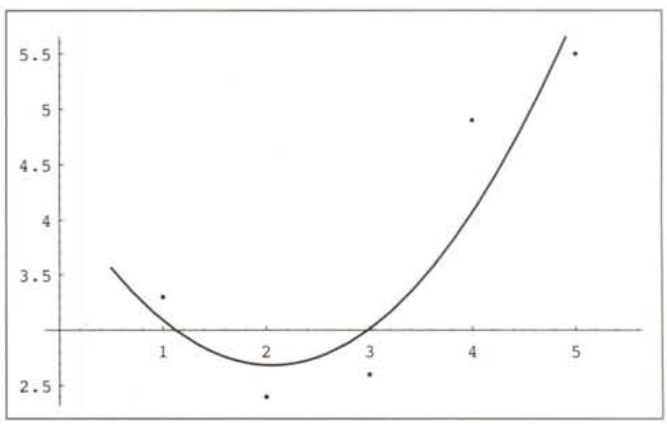
In *Mathematica* esiste una funzione primitiva di nome **Fit** che risolve lo stesso problema in modo più efficiente (perché programmata internamente in C) e permette di specificare una qualunque lista di funzioni base.

In[9]:=

```
polfit=Fit[Transpose[{asc,data}],
{1,x,x^2}, x]
```

Out[9]=

```
4.22 - 1.49571 x + 0.364286 x^2
```



In[10]:=

```
ListPlot[Transpose[{asc,data}],
     DisplayFunction->Identity];
Plot[polfit,{x,0.5,5.5},
```

Bibliografia

R.Bevilacqua, D. Bini, M. Capovani, O.Menchi. **Introduzione alla Matematica Computazionale.** Zanichelli (1987)
R.Bevilacqua, D. Bini, M. Capovani, O.Menchi. **Metodi Numerici.** Zanichelli (1992)
S. Wolfram. *Mathematica.* **A System for Doing Mathematics by Computer.** Addison Wesley,1991 (II Edition).

```
    DisplayFunction->Identity];
Show[%,%%,
    DisplayFunction->$DisplayFunction];
```

Un esempio di applicazione

Per rendersi conto meglio di come vanno le cose con l'interpolazione e con l'approssimazione ai minimi quadrati supponiamo di voler ritrovare l'espressione analitica di una funzione affetta da rumore casuale. Si consideri per esempio la funzione $1+x^2$

In[11]:=
```
f[x_] = 1+x^2;
```

a cui si aggiunge un numero casuale compreso tra ±1/10, ottenendo la funzione "disturbata"

In[12]:=
```
f1[x_] := f[x] +
          Random[Real,{-0.1,0.1}]
```

Calcoliamo la funzione "disturbata" in 11 punti e calcoliamo il polinomio di interpolazione, che viene ovviamente di grado 10.

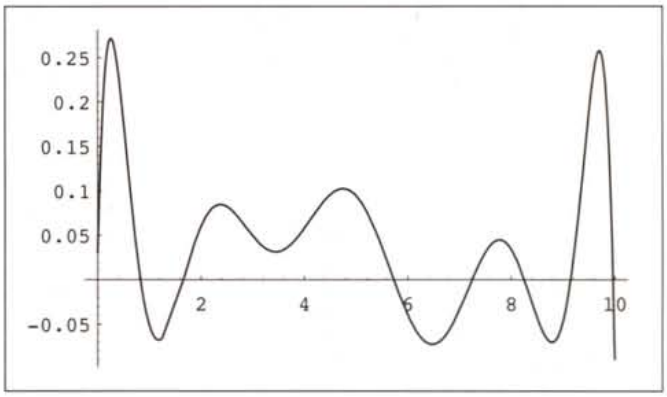
In[13]:=
```
asc  = Table[x, {x,0,10}];
data = Table[f1[x], {x,0,10}]
```
Out[13]=
```
{0.969118, 2.05139, 4.93969, 9.95089,
16.942, 25.9056, 37.0396, 50.034, 64.9658,
82.0472, 101.089}
```
In[14]:=
```
z2=Interpol[asc,data,x]
```
Out[14]=

$$0.969118 - 2.28297\,x + 8.07151\,x^2 - 8.32705\,x^3 + 5.09793\,x^4 - 1.83256\,x^5 + 0.407552\,x^6 - 0.0567247\,x^7 + 0.00480745\,x^8 - 0.000226746\,^{9} + 4.5615\times10^{-6}\,x^{10}$$

Si noti il trucco di calcolare una volta per tutte l'espressione del polinomio interpolante assegnandola ad una variabile con l'operatore "=" e di usarne poi il valore nella definizione di una funzione, evitando così di dover effettuare ogni volta l'interpolazione (lo stesso sistema si può usare per non ripetere inutilmente il calcolo di integrali o derivate simbolici).

In[15]:=
```
g[x_]:=z2
```

La differenza tra la funzione f originaria e il polinomio g presenta numerose oscillazioni.



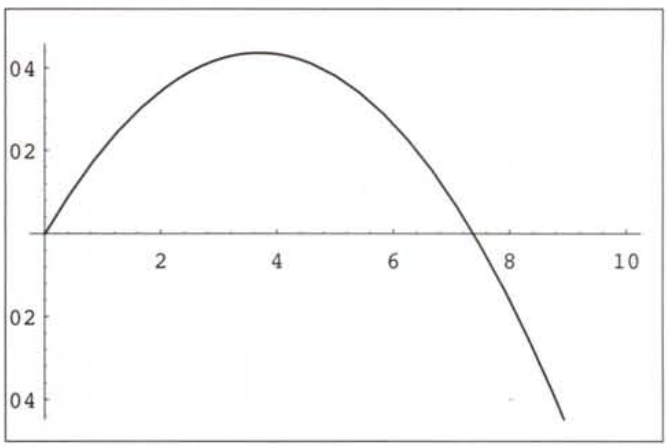
In[16]:=
```
p2=Plot[f[x]-f2[x],{x,0,10}];
```

Invece, approssimando, nel senso dei minimi quadrati, i punti dati con un polinomio di grado 2 si ha:

In[17]:=
```
z3=Fit[Transpose[{asc,data}], {1,x,x^2}, x]
```
Out[17]=

$$1.000226 - 0.0237509\,x + 1.00322\,x^2$$

In[18]:=
```
f3[x_]:=z3
p3=Plot[f[x]-f3[x],{x,0,10}];
```



La presenza del rumore ha impedito di ricostruire esattamente la funzione originaria, ma l'espressione trovata è comunque molto più vicina a f di quanto non fosse la g, infatti l'errore commesso è molto minore di quello del caso precedente.

MC

Francesco Romani è raggiungibile tramite Internet all'indirizzo romani@di.unipi.it