Facciamo un dizionario?

Riprendiamo il discorso dell'elaborazione di testi in Mathematica *studiando un problema che ha anche un risvolto teorico: la ricerca delle occorrenze delle parole di una lingua in una raccolta di testi. Come base di partenza ho preso alcuni testi del progetto LiberLiber, rimpolpati con altri testi che avevo a disposizione. Come al solito chiedo scusa agli specialisti (questa volta i linguisti) per la trattazione superficiale e imprecisa*

di Francesco Romani

Alcune definizioni

Un **dizionario** è una raccolta ordinata alfabeticamente delle parole di una lingua corredate da alcune informazioni accessorie: la definizione o traduzione, l'etimologia, la pronuncia, la categoria grammaticale, esempi d'uso, etc. Di solito in un dizionario di ogni parola compare solo la forma principale: il **lemma** ("amare", "casa", "grandiosamente").

Il **lemmario** è l'elenco di tutti i lemmi di una lingua. Grossolanamente si può stimare che l'italiano abbia circa 130.000 lemmi.

Il **formario** è l'elenco di tutte le forme di una lingua ("amai", "amare", "case", "pasticcini"). Grossolanamente si può stimare che l'italiano abbia circa 1.000.000 forme. Il numero può variare grandemente a seconda che si contino solo le forme attestate (cioè effettivamente usate da qualche scrittore antico o moderno) o anche quelle grammaticamente possibili (p. es. "trilinguissimo"). Vi sono anche problemi accessori legati al mutare della lingua attraverso il tempo e alle varianti dialettali.

Una volta i curatori dei dizionari usavano il seguente procedimento per scrivere un dizionario. Analizzando i testi di riferimento per ogni forma si risaliva al lemma (questo procedimento è detto **lemmatizzazione**); per ogni lemma si scriveva un cartellino; tutti i cartellini conservati in ordine alfabetico servivano per formare il testo completo del dizionario. Nell'ultimo quarto di secolo questa procedura è stata automatizzata ed oggi il computer è lo strumento di lavoro più usato da molti linguisti.

Il problema che affrontiamo (in modo puramente didattico) è quello della ricerca del maggior numero di occorrenze di parole diverse in un corpus dato di testi (letterari e non). In assenza di uno specifico programma di lemmatizzazione (ovvero del programma che deriva il lemma a partire dalle forme) ci limiteremo a costruire un formario del campione in considerazione. Verrà anche preso in considerazione il problema (prettamente matematico) di come è correlata la crescita del formario con la crescita del campione.

Il campione

Come più volte segnalato su MCmicrocomputer Il "progetto Manuzio", patrocinato dall'associazione culturale, senza fini di lucro, "Liber Liber" mette a disposizione sul nodo Internet `ftp.dsi.unimi.it` directory: `pub2/papers/basagni/Manuzio` un certo numero di testi italiani. Per informazioni esaurienti e aggiornate si vedano i file README sul medesimo archivio. I testi sono in formato ASCII esteso MS-DOS [codepage 437, USA International].

Per il nostro esperimento ho scelto alcuni testi in prosa di autore italiano e precisamente

"La Costituzione della Repubblica"
"Giorni di guerra in Sicilia", Grazia Pagliaro
"I promessi sposi", Alessandro Manzoni
"Il turno", Luigi Pirandello
"I Malavoglia", Giovanni Verga
"La Giara", Luigi Pirandello
"Sei racconti", Giovanni Verga
"Senilità", Italo Svevo
"Una giornata", Luigi Pirandello.

A questi file (copiati sul Macintosh senza convertire il codice ASCII) se ne sono aggiunti altri 6 (gentilmente concessi da amici linguisti) contenenti contesti di autori italiani. In totale abbiamo 14 file ordinati per lunghezza crescente *(Figura 1)* per un totale di circa 5 Megabyte.

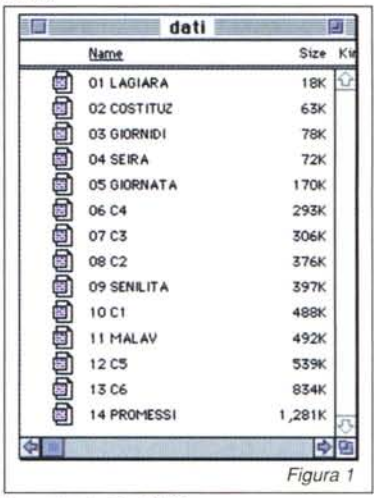


Figura 1

La directory del folder può essere facilmente trasformata in una lista di stringhe

In[1]:=
```
SetDirectory["HDF:MC:  man:dati"];
FileNames[]
```
Out[1]=
```
{"01 LAGIARA","02 COSTITUZ",
 "03 GIORNIDI","04 SEIRA","05 GIORNATA",
 "06 C4","07 C3","08 C2","09 SENILITA",
 "10 C1","11 MALAV","12 C5","13 C6",
 "14 PROMESSI"}
```

La lettura di un file si effettua con il comando **`ReadList`**. Utilizzando come separatori i segni di interpunzione si ottiene una lista di parole con qualche porcheria in mezzo.

In[2]:=
```
testo=ReadList[
 "01 LAGIARA",
 Word,WordSeparators->
 {" ",",",".","?","!",":",";","'"}];
Length[testo]
testo
```
Out[2]=
```
3266
```

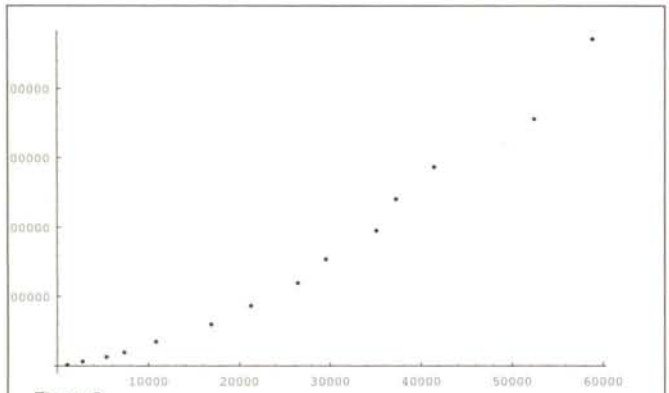


Figura 2

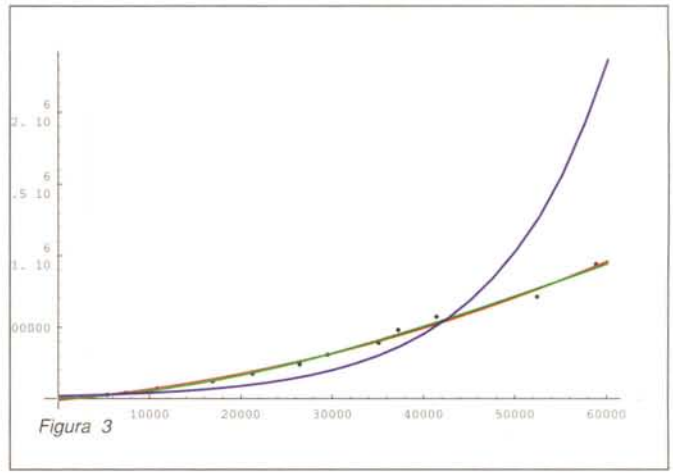


Figura 3

```
{..."prevedendo","che","le","cinque",
"giare","vecchie","di","coccio",
"smaltato","che","aveva","in","cantina",
"non","sarebbero","bastate","a",
"contener","tutto","l","olio","della",
"nuova","raccolta","ne","aveva",
"ordinata","a","tempo","una","sesta",
"pió","capace","a","Santo","Stefano"...}
```

Correzione del testo

Studiando un poco le liste ottenute si evidenziano alcuni problemi: le lettere accentate sono tutte sbagliate a causa del diverso codice ASCII utilizzato, sono presenti sia maiuscole che minuscole, sono presenti caratteri non lettere (parentesi, numeri asterischi, etc). Vogliamo costruire un filtro che effettui le seguenti operazioni:

- corregge gli accenti
- trasforma le maiuscole in minuscole
- elimina i caratteri non lettere.

Scriviamo dapprima la lista delle trasformazioni che correggono gli accenti. Questa lista è stata preparata a mano guardando un po' di testi, ci sono due correzioni diverse per la "ù" perché questa è rappresentata in due modi diversi in testi diversi.

In[1]:=

```
trasfdos={"ó"->"ù",
          "£"->"ù",
          "Ç"->"è",
          "ï"->"ò",
          "°"->"ì",
          "Ö"->"à",
          "ä"->"é",
          "Ö"->"à"};
```

Veniamo ora alle regole che trasformano le maiuscole in minuscole

In[2]:=

```
trasfM={
"A"->"a","B"->"b","C"->"c","D"->"d",
"E"->"e","F"->"f","G"->"g","H"->"h",
```

Un Modello matematico

Il fatto che il fitting migliore sia quello quadratico è un fenomeno curioso, di non facile spiegazione. In attesa di dedicarvi un'intera puntata, vi offro come antipasto un'idea di Giuseppe Fiorentino su cui stiamo lavorando.

Sia **u** il numero di tutte le parole della lingua in esame. Come prima approssimazione si assume che tutte le parole abbiano la stessa probabilità di presentarsi. Allora se ho trovato **t** parole dopo averne scandite **c** mi ne restano da trovare ancora **u-t** e la probabilità di trovarne di nuove non è **1** ma **(u-t)/u**. Si può scrivere allora l'equazione differenziale **dt/dc = (u-t)/u**.

Risolviamola con *Mathematica*

In[1]:=

```
DSolve[{t'[c]==1-t[c]/u,
        t[0]==0},t[c],c]
```

Out[1]=

$$\{\{t[c] \to u - \frac{u}{E^{c/u}}\}\}$$

In[2]:=

```
t[c_]:=Evaluate[t[c]/.%[[1]]]
```

E troviamo lo sviluppo in serie della funzione inversa c(t) che è quella che ci interessa

In[3]:=

```
InverseSeries[Series[t[c],{c,0,5}]]
```

Out[3]=

$$c + \frac{c^2}{2u} + \frac{c^3}{3u^2} + \frac{c^4}{4u^3} + \frac{c^5}{5u^4} + O[c^6]$$

Si vede che il coefficiente del termine cubico è diviso per u^2 e quello del termine di quarto grado è diviso per u^3. Nel nostro caso, con **u** dell'ordine del milione e **c** dell'ordine di qualche decina di migliaia significa che i termini di grado superiore al secondo possono essere trascurati.

Ovviamente l'ipotesi di distribuzione uniforme delle parole è molto imprecisa; possibili sviluppi di ricerca consistono nel trovare modelli più realistici e nel confrontarne le previsioni con i risultati sperimentali ottenuti su un corpus che sia una frazione consistente di una lingua. Con le lingue moderne questo è di difficile realizzazione ma per il Latino e il Greco ci stiamo organizzando...

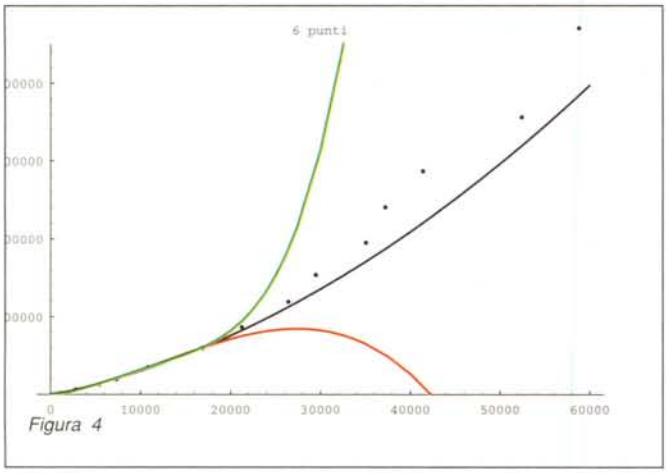


Figura 4

```
"I"->"i","J"->"j","K"->"k","L"->"l",
"M"->"m","N"->"n","O"->"o","P"->"p",
"Q"->"q","R"->"r","S"->"s","T"->"t",
"U"->"u","V"->"v","W"->"w","X"->"x",
"Y"->"y","Z"->"z","È"->"è"};
```

Scriviamo infine l'elenco delle lettere

In[3]:=

```
lettere=
{"a","b","c","d","e","f","g","h",
 "i","j","k","l","m","n","o","p",
 "q","r","s","t","u","v","w","x",
 "y","z","A","B","C","D","E","F",
 "G","H","I","J","K","L","M","N",
 "O","P","Q","R","S","T","U","V",
 "W","X","Y","Z","à","è","ì","ò",
 "ù","é"};
```

L'elaborazione di una stringa consiste nel trasformare la stringa in una lista di caratteri (**Characters**), applicare le due trasformazioni, eliminare le non lettere (**Select**) e rimettere tutto insieme (**StringJoin**)

In[3]:=

```
elabs[s_String]:=StringJoin@@
Select[(Characters[s]/.
          trasfM/.trasfdos),
       MemberQ[lettere, #]&]
```

L'elaborazione di una lista di stringhe consiste nell'applicare **elabs** ed eliminare le parole vuote. Poiché **Union[Union[{""},v]** elimina le ripetizioni e ordina lessicograficamente il vettore **v** la stringa vuota viene in testa e viene eliminata prendendo il **Rest** del risultato.

In[4]:=

```
elabl[l_List]:=
  Rest[Union[{""},Map[elabs,l]]]
```

Elaborazione

Abbiamo ora tutti gli elementi per costruire un formario a partire dal nostro campione. **process** prende in input una lista di stringhe (i filename dei dati), per ogni elemento della lista legge il file corrispondente, conta le parole, corregge i caratteri speciali ed elimina le non lettere. Ad ogni passo vengono modificate le seguenti variabili:

all: la lista di tutte le parole trovate (il formario);

pc: il numero di tutte le parole analizzate;

pt: la cardinalità di **all**;

pl: la lista delle coppie **{pl,pc}** da usare in seguito per i plottaggi e i fitting;

time: il tempo di elaborazione in secondi per l'ultimo file.

Per ogni file si stampa in una riga il filename seguito dalla lista **{lt,time,pc,pt}.**

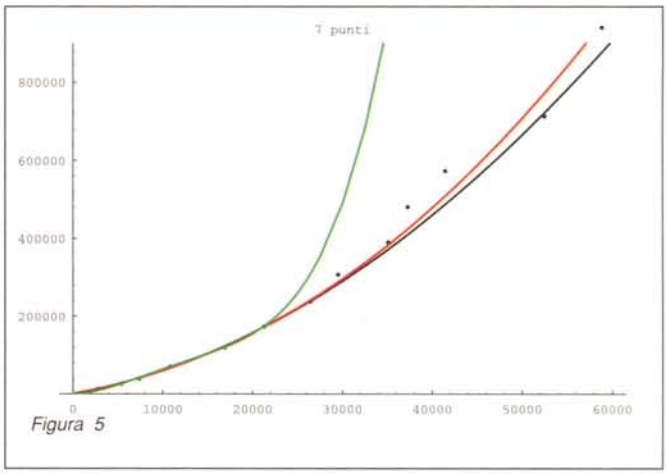


Figura 5

In[1]:=

```
process[t_List]:=Module[
   {i,testo,pc,pt,lt,tr,time},
  all={};
  pl={};
  pc=0;
  pt=0;
  Do[
   testo=ReadList[t[[i]],
      Word,
      WordSeparators->
       {" ",",",".","?",
        "!",":",";","'"}];
   lt=Length[testo];
   pc=pc+lt;
   time=Timing[
     tr=elabl[Union[testo]];
        ][[1,1]];
   all=Union[all,tr];
   pt=Length[all];
   AppendTo[pl, {pt,pc}];
   Print[t[[i]]," ",
        {lt,time,pc,pt}],
  {i,Length[t]}]];
```

Applicando **process** a tutti i file del campione si ottiene il seguente risultato.

In[2]:=

```
process[FileNames[]]
```

Out[2]=

```
01 LAGIARA      {3266,   4.9,   3266,  1106}
02 COSTITUZ     {9458,  10.3,  12724,  2787}
03 GIORNIDI    {13413,  17.6,  26137,  5404}
04 SEIRA       {13167,  14.7,  39304,  7356}
05 GIORNATA    {30605,  29.2,  69909, 10838}
06 C4          {49818,  62.2, 119727, 16906}
07 C3          {53561,  58.1, 173288, 21269}
08 C2          {65100,  71.1, 238388, 26437}
```

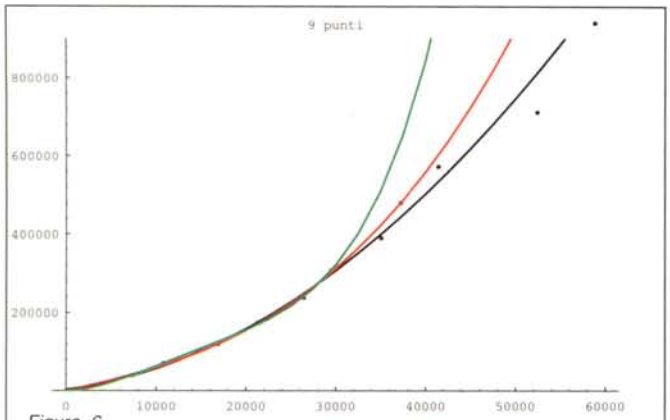


Figura 6

```
09 SENILITA    {69043,   47.9, 307431, 29520}
10 C1          {82749,   90.9, 390180, 35085}
11 MALAV       {90464,   44.5, 480644, 37249}
12 C5          {92495,   88.4, 573139, 41429}
13 C6         {139301, 143.5, 712440, 52432}
14 PROMESSI   {228881, 121.9, 941321, 58834}
```

Si nota che i tempi di elaborazione sono in generale accettabili (2 minuti per elaborare i "Promessi Sposi").

Ovviamente un programma C ottimizzato richiederebbe una frazione di questo tempo di elaborazione ma un tempo uomo almeno dieci volte maggiore per la programmazione.

Fitting

Le ultime due colonne mostrano come variano rispettivamente c la cardinalità dell'insieme delle parole scandite e t la cardinalità dell'insieme delle parole trovate.

Il plottaggio di **pl** dà un'idea di come cresce la funzione c(t).

In[1]:=

```
listplot=ListPlot[pl,
        PlotRange->All];
```

(Figura 2)

Ovviamente la crescita non è lineare. Per avere un idea più precisa di come cresce la funzione c(t) proviamo a fare qualche approssimazione ai minimi quadrati. Iniziamo con un polinomio di secondo grado:

In[2]:=

```
f2=Fit[pl, {1,x,x^2}, x]
```

Out[2]=

$$-10518.2 + 5.58235\ x + 0.000176063\ x^2$$

e di terzo grado:

In[3]:=

```
f3=Fit[pl, {1,x,x^2,x^3}, x]
```

Out[3]=

$$0.000360445 + 2.46747\ t + 0.000318208\ t^2 - 1.62262\ 10^{-9}\ t^3$$

Un altro tentativo può essere la funzione esponenziale a e^{bx}. Per fare questa approssimazione è necessario applicare l'identità $\text{Log}(a\ e^{bx}) = \text{Log}\ a + b\ x$ e quindi ricavare Log a e b da un fit lineare del logaritmo naturale dei dati.

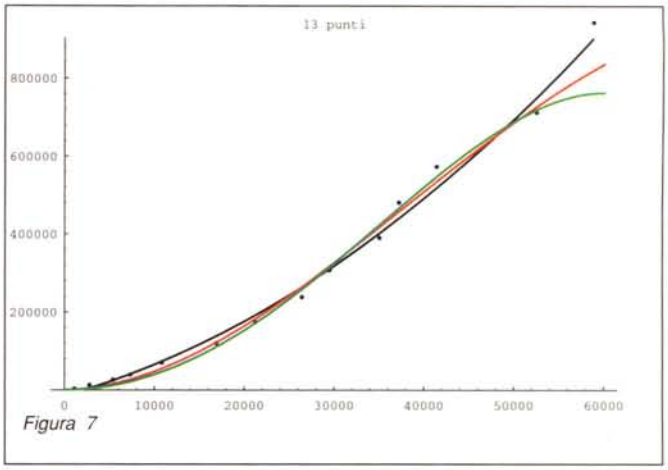


Figura 7

In[4]:=

```
lpl=
 Map[{#[[1]],Log[N[#[[2]]]]}&,pl];
```

In[5]:=

```
fe=Expand[Exp[Fit[lpl, {1,x}, x]]]
```

Out[5]=

$$16562.8\ E^{0.0000826762\ t}$$

Disegniamo insieme i tre grafici e i punti sperimentali:

In[6]:=

```
plot=Plot[{f2,f3,fe},{x,0,60000},
      PlotStyle->{Red,Green,Blue}];
Show[listplot,plot];
```

(Figura 3)

Si nota che il fitting esponenziale sballa completamente mentre i due fitting polinomiali sono abbastanza accurati. Per decidere quale sia il grado più appropriato per il polinomio proviamo ad approssimare solo i primi punti e vediamo come vengono previsti gli altri. Il seguente programma approssima con polinomi di secondo, terzo e quarto grado, partendo con 6 punti e arrivando fino a 13.

In[7]:=

```
Do[
 f2=Fit[Take[pl,npl],
     {1,x,x^2},x];
 f3=Fit[Take[pl,npl],
     {1,x,x^2,x^3}, x];
 f4=Fit[Take[pl,npl],
     {1,x,x^2,x^3,x^4}, x];
 plot=Plot[{f2,f3,f4},{x,0,60000},
       PlotStyle->{Black,Red,Green},
       DisplayFunction->Identity];
 Show[listplot,plot,
       DisplayFunction->
        $DisplayFunction,
       PlotRange->{0,900000},
       PlotLabel->
        ToString[npl]<>"punti"],
{npl,6,13}];
```

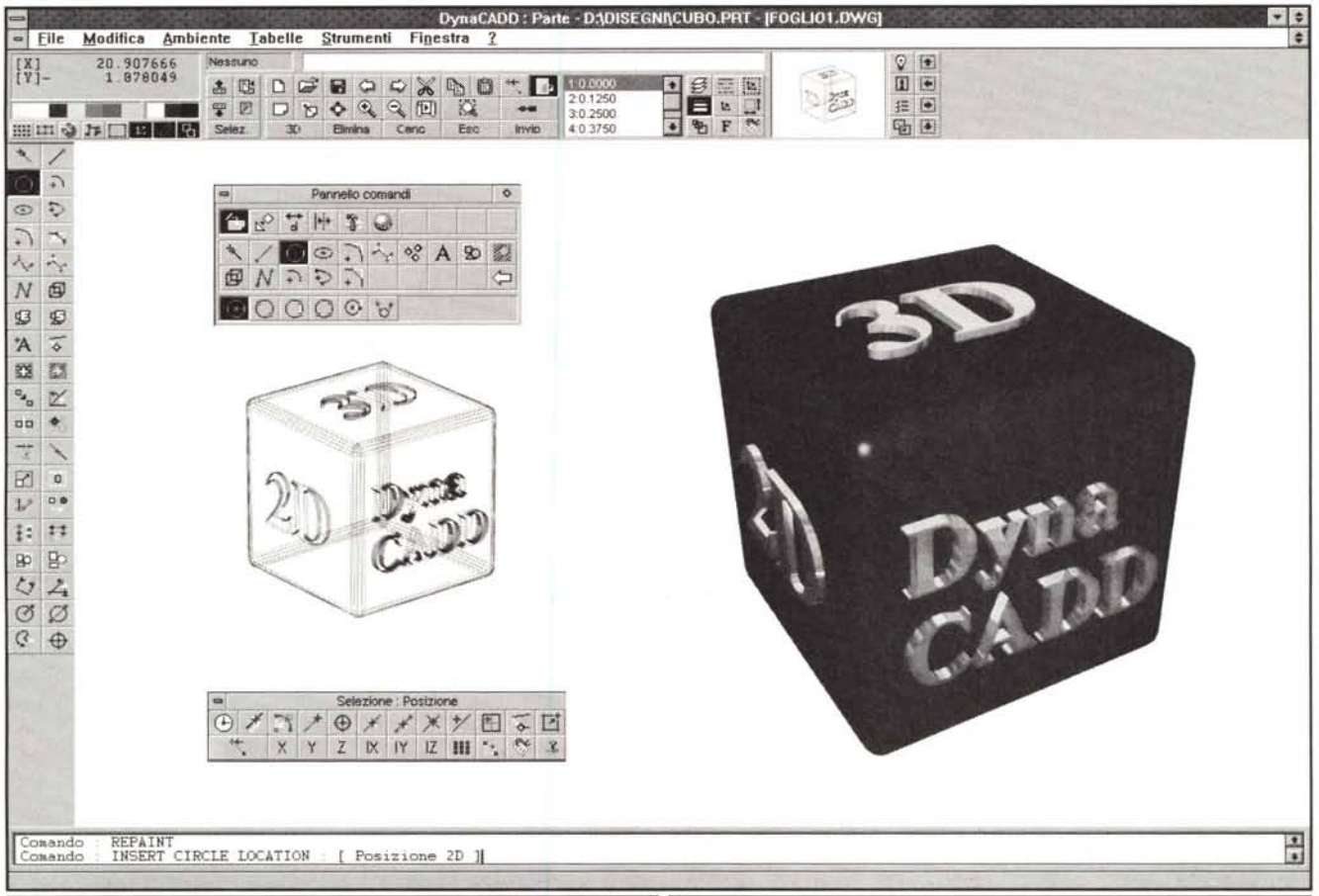
Le figure da 4 a 7 mostrano i risultati per **npl=6,7,9,13**, rispettivamente. È evidente che l'unica approssimazione che permette di fare previsioni ragionevoli è quella quadratica.

MC

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

DynaCADD®

I CAD PROFESSIONALI 2D-3D PER DOS E WINDOWS



DynaDesigner — WIN

CAD 2D per Windows - Associativo - 256 layer - 24 tipi di primitive - Oltre 2000 comandi tramite interfaccia grafica o linea comandi - Undo e Redo infiniti - DXF in/out - Autorecover in grado di recuperare tutto il lavoro in caso di crash - Cursore intelligente con 8 possibilità di snap - Programmabile in C tramite il sistema di sviluppo e conversione dei font True Type tramite il Font Editor (non inclusi).

Lit. 550.000

UPGRADE

DynaCADD — WIN

CAD 2D/3D per Windows - Le funzioni di DynaDesigner e in più:
- Funzioni per la creazione di solidi
- Oltre 2500 comandi
- Viste tridimensionali illimitate
- Rendering a 24 bit con shading, shadowing (con algoritmi di Pixar) e texture mapping
- Programma per l'editing bitmap incluso
- Servizio di Hotline gratuito.

Lit. 1.200.000

UPGRADE

DynaCADD 2D-3D — DOS

Cad 2D e 3D wireframe per DOS - 256 layer - 13 tipi di primitive incluse curve di Bézier e B-splines - Uscita su stampanti, plotter e dispositivi Postscript - Font vettoriali ed editor di font inclusi - Interscambio file DXF 2D e 3D sia in lettura che in scrittura - Help in linea - Viste tridimensionali multiple - Quotature automatiche - Precisione a 16 cifre - Interfaccia utente semplice ed intuitiva - Servizio di Hotline gratuito.

Lit. 250.000

IMPORTATORE E DISTRIBUTORE ESCLUSIVO PER L'ITALIA: STUDIO NUOVE FORME SRL

Via Mancinelli, 19 - 20131 Milano - Tel. 02/26143833 r.a. - Fax 02/26147440 - DynaCADD Hot-line 02/26149649