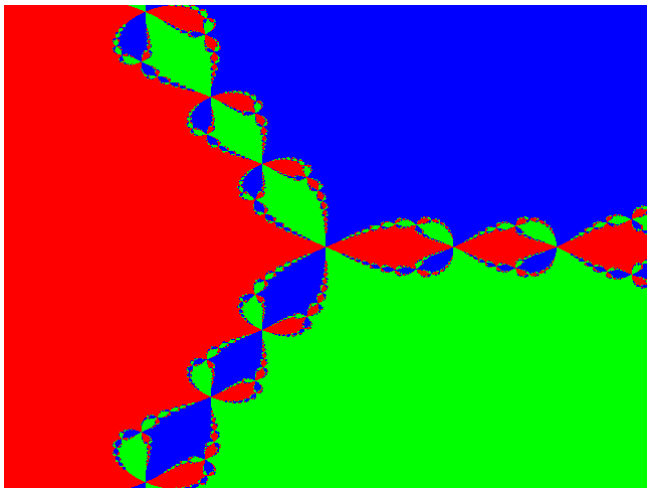
# Laboratorio computazionale numerico
# Lezione 8

f.poloni&sns.it

2008-12-03

## 1 Frattali di Newton

In questa lezione cercheremo di disegnare i "frattali" che si ottengono disegnando i bacini di attrazione del metodo di Newton complesso per un polinomio. Le immagini risultanti dovrebbero assomigliare a quella seguente.



Il problema può essere suddiviso in molti sottoproblemi ognuno abbastanza semplice.

### 1.1 Manipolazione di polinomi

Dato un polinomio, lo rappresentiamo come il vettore dei suoi coefficienti: ad esempio, a $x^3 - 1$ corrisponde il vettore $[1;0;0;-1]$. Notare che se il polinomio ha grado $n$, il vettore ha lunghezza $n + 1$.

Il metodo di Horner per valutare un polinomio corrisponde a fare i prodotti associandoli in questo modo: per esempio, per un polinomio di grado 4,

$$a(x) = (((a_4 * x + a_3) * x + a_2) * x + a_1) * x + a_0.$$

*Esercizio* 1. Scrivere una **function** y=horner(p,x) che prende un polinomio $p$ (rappresentato come il vettore dei suoi coefficienti) e un numero $x$ e calcola $p(x)$ con il metodo di Horner.

*Esercizio* 2. Scrivere una **function** dp=derivata(p) che prende un polinomio $p$ (vettore di coefficienti) e restituisce la sua derivata (vettore di coefficienti).

## 1.2 Metodo di Newton

Il metodo di Newton è l'iterazione

$$x_{k+1} = x_k - \frac{p(x_k)}{p'(x(k)}.$$

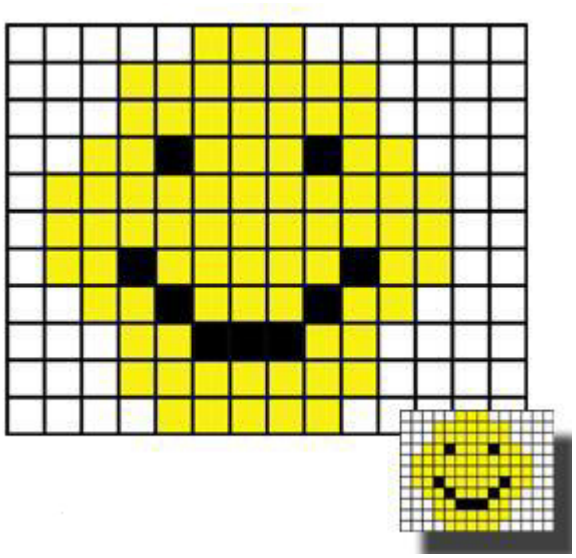
*Esercizio* 3. Scrivere una **function** x=newton(p,x0) che esegue il metodo di Newton sul polinomio $p$ partendo dal punto iniziale x0. Come criterio di arresto, si può usare quello di terminare se $|p(x)| \leq 10^{-12}$:

```
function x=newton(p,x0);
  x=x0;
  px=horner(p,x0);
  while(abs(px)>1E-12)
    %calcola il nuovo x e il nuovo px
  endwhile
endfunction
```

Testare il metodo di Newton sul polinomio $p(x) = x^3 + 1$. Quali sono le sue radici? Riuscite a trovare un valore iniziale per il metodo di Newton che lo faccia convergere ad ognuna di esse? Ricordate che il modo più semplice per inserire un numero complesso in Octave è 2+3*I.

## 1.3 Manipolazione di immagini

Per un computer, un'immagine è formata da un rettangolo di quadratini colorati (pixel). Ogni quadratino viene colorato con luce rossa, verde e blu, ognuna con una certa intensità.



In Octave, un'immagine è rappresentata da una matrice contenente dei numeri interi. Octave traduce questi interi in colori in base a una matrice $n \times 3$, detta

colormap, che contiene nella riga $k$-esima i tre valori di intensità (in una scala da 0 a 1) rispettivamente del rosso, verde e blu da associare al $k$-esimo colore. Per esempio, inizializzando la colormap con le istruzioni

```
octave:24> cm=eye(3)
cm =

   1   0   0
   0   1   0
   0   0   1

octave:25> colormap(cm);
octave:26>
```

Octave colorerà tutti gli elementi contenenti il numero 1 di una matrice immagine di verde (intensità=1 0 0), tutti gli elementi contenenti 2 di rosso, tutti gli elementi contenenti 3 di blu. Per esempio, provate a dare le istruzioni

```
octave:28> A=ones(100,100)+eye(100,100)
octave:29> image(A)
```

(l'istruzione **image** visualizza un'immagine a schermo).

## 1.4 Frattale di Newton

Per disegnare il frattale di Newton relativo al polinomio $p(x) = x^3 + 1$, abbiamo bisogno innanzitutto di una funzione che "decida" a quale valore c'è convergenza.

*Esercizio* 4. Scrivere una funzione **function** val=decidi(x) che restituisca 1, 2 o 3 a seconda se il numero complesso $x$ è più vicino a $-1$, a $\frac{1}{2} + I\frac{\sqrt{3}}{2}$, o a $\frac{1}{2} - I\frac{\sqrt{3}}{2}$.

Prendiamo 101 valori equispaziati nell'intervallo $[-2, 2]$ con l'istruzione range=−2:0.04:2.

*Esercizio* 5. Scrivere una **function** img=disegna() che non prende alcun argomento e restituisce una matrice $101 \times 101$ calcolata in questo modo: per ogni coppia $(i, j)$,

- Calcola il punto del piano complesso z0=range(i)+I∗range(j);
- Esegue il metodo di Newton per il polinomio $x^3 + 1 = 0$;
- Utilizzando la funzione decidi(z), scriva 1, 2 o 3 in img(i,j) se il metodo di Newton con valore iniziale z0 converge rispettivamente a una delle tre radici del polinomio[1].

La funzione appena scritta sarà probabilmente abbastanza lenta (potrebbe metterci un mezzo minuto...) e restituirà una matrice img che potete visualizzare a schermo con l'istruzione **image**(img) (se non l'avete ancora fatto, utilizzate prima l'istruzione **colormap**(**eye**(3)) per inizializzare la colormap).

[1] potete dare per scontato che il metodo di Newton converga per tutti i valori iniziali nel nostro range.

## 1.5 Ottimizzazione

(se avete finito e vi state annoiando) La funzione appena scritta è abbastanza lenta. *Una volta che abbiamo stabilito che è troppo lenta*, e solo allora[2], possiamo pensare a come farla andare più veloce. Per come è fatto Octave (è un linguaggio *interpretato*), le parti critiche sono:

- l'esecuzione di molte istruzioni è lenta: per esempio un ciclo for vs. un'assegnazione fatta direttamente su un vettore:

```
for i=1:100
  v(i)=i+1;
endfor
%e' piu' lento di
v=1+1:100;
```

- la chiamata di funzioni è lenta, in particolare le funzioni definite dall'utente sono più costose di quelle già esistenti dentro Octave.

Un primo passo può essere quello di sostituire le nostre funzioni horner e derivata con due funzioni già esistenti in Octave che fanno lo stesso lavoro: rispettivamente **polyval** e polyderiv. Poi, dovremmo cercare di rimpiazzare le $101 \times 101$ chiamate di funzione con un'esecuzione di Newton "in parallelo" su tutti gli elementi della matrice. Ci possono aiutare l'istruzione ./ di Octave (divisione elemento per elemento tra due matrici/vettori) e il fatto che **polyval** lavora "in parallelo" elemento per elemento se gli passiamo una matrice. La funzione decidi() invece è più complicata, serve utilizzare le variabili di tipo logical (date un'occhiata alla sezione 4.5 del manuale di Octave, lo trovate per esempio sul sito del corso).

Potete passare il resto della lezione a cercare di velocizzare il programma appena scritto. In fondo a questo documento (c'è una pagina bianca in mezzo) trovate una possibile soluzione.

[2] "Premature optimization is the root of all evil", D. Knuth (quello che ha inventato il TEX, tra le altre cose).

## 1.6 Soluzione di “Ottimizzazione”

```
function img=disegna_opt(dimensione)
p=[1;0;0;1];
n=dimensione;
range=-2:4/(n-1):2;

deg=size(p,1)-1;
dp=polyderiv(p);
roots=roots(p);

colormap(eye(3)); %da cambiare se si alza il grado del polinomio...

%matrice dei valori iniziali
X=ones(n,1)*range+range'*ones(1,n)*I;

%esegue 10 passi di Newton "in parallelo", dovrebbero bastare
for k=1:10
  X=X-polyval(p,X)./polyval(dp,X);
endfor

%trova il punto piu' vicino "in parallelo"
img=zeros(n,n);
for i=1:deg
  Map=ones(n,n);
  for j=1:deg
    if (j~=i)
      Map=Map & (abs(X-roots(i)) < abs(X-roots(j)));
    endif
  endfor
  img(Map)=i;
endfor
endfunction
```