

# Laboratorio di Analisi Numerica

## Lezione 9

Leonardo Robol <leonardo.robol@unipi.it>

Igor Simunec <igor.simunec@sns.it>

26 Novembre 2021

**Quantità di esercizi:** in questa dispensa ci sono *più esercizi* di quanti uno studente medio riesca a farne durante una lezione di laboratorio, specialmente tenendo conto anche degli esercizi facoltativi. Questo è perché è pensata per “tenere impegnati” per tutta la lezione anche quegli studenti che già hanno un solido background di programmazione. Quindi fate gli esercizi che riuscite, partendo da quelli *non* segnati come facoltativi, e non preoccupatevi se non li finite tutti!

### 1 Frattali di Newton

In questa lezione cercheremo di disegnare i frattali che si ottengono disegnando i bacini di attrazione del metodo di Newton (sul piano complesso) per un polinomio. Le immagini risultanti dovrebbero assomigliare a quella in Figura 1. Notate la simmetria della figura: a seconda del quadrante del piano complesso da cui partiamo, si ha convergenza alla più vicina delle tre radici; però, nei punti che sono circa equidistanti da due delle tre radici, si ha un comportamento caotico.

Cominciamo dividendo il problema in molti sottoproblemi più semplici.

#### 1.1 Manipolazione di polinomi

Dato un polinomio, lo rappresentiamo come il vettore dei suoi coefficienti: ad esempio, a  $x^3 + 2x - 1$  corrisponde il vettore  $[1 \ 0 \ 2 \ -1]$ . 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 prenda un polinomio  $p$  (rappresentato come il vettore dei suoi coefficienti) e un numero  $x$  e calcoli  $p(x)$  con il metodo di Horner.

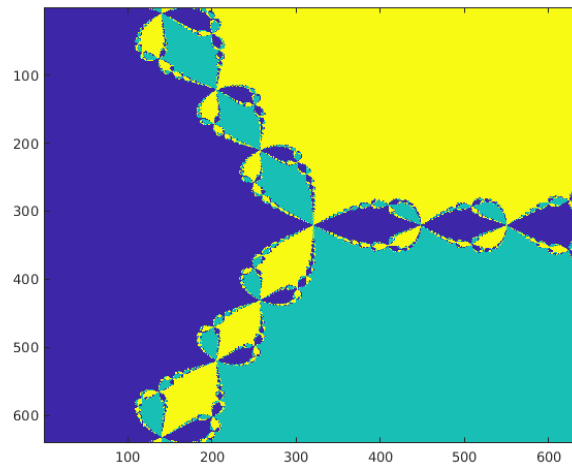


Figura 1: Disegno dei bacini di attrazione del metodo di Newton per il polinomio  $x^3 + 1$ .  
I tre colori diversi corrispondono ai punti del piano complesso a partire da cui Newton converge alle tre diverse radici.

*Esercizio 2.* Scrivere una **function** `dp=derivata(p)` che prenda un polinomio  $p$  (vettore di coefficienti) e restituisca 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 esegua il metodo di Newton sul polinomio  $p$  partendo dal punto iniziale  $x_0$ . 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
    end
end
```

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

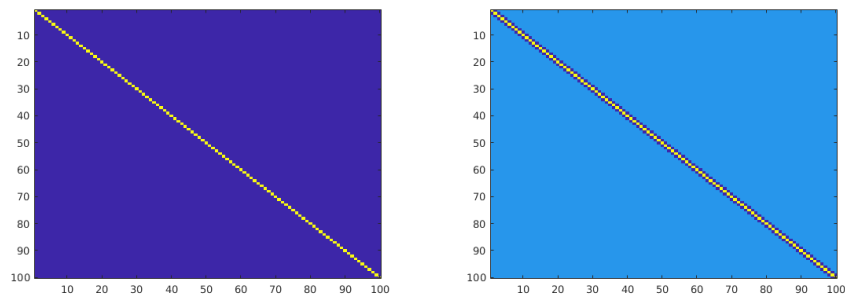


Figura 2: `imagesc(eye(100))` e `imagesc(laplacian(10))`

ognuna di esse? Ricordate che il modo più semplice per inserire un numero complesso in MATLAB è `2+3i`.

### 1.3 “Disegnare” una matrice

La funzione `imagesc` prende come parametro una matrice  $A$  di dimensione  $m \times n$  e genera un'immagine  $m \times n$  in cui il pixel  $(i, j)$  è colorato di un colore che varia su una scala da blu a giallo a seconda di quanto il valore di  $A_{i,j}$  è grande/piccolo rispetto agli altri elementi della matrice. Per esempio, con

```
>> imagesc(eye(100))
```

viene visualizzata un'immagine in cui la diagonale (elementi più grandi) è gialla, e tutti gli altri elementi (elementi più piccoli) sono blu. Provate anche `imagesc(laplacian(10))` o `imagesc(rand(100))`.

### 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  $\frac{1}{2} + \frac{\sqrt{3}}{2}i$ , o a  $\frac{1}{2} - \frac{\sqrt{3}}{2}i$ .

*Esercizio 5.* Scrivere una funzione `img=newtonfractal()` che non prenda alcun argomento e restituisca una matrice  $101 \times 101$  chiamata `img` calcolata in questo modo:

- genera 101 valori equispaziati nell'intervallo  $[-2, 2]$  con l'istruzione `range=-2:0.04:2`.
- per ogni coppia  $(s, t)$ :
  - calcola il punto  $z_0$  del piano complesso `z0=range(s)+1i*range(t)`;

- esegue il metodo di Newton per il polinomio  $x^3 + 1 = 0$  partendo dal punto `z0`;
- utilizzando la funzione `decidi()`, scrive 1, 2 o 3 in `img(t,s)` a seconda della radice del polinomio a cui si ha convergenza a partire dal valore iniziale `z0`.<sup>1</sup>.

- restituisce la matrice `img`

La funzione qui scritta sarà probabilmente abbastanza lenta (potrebbe metterci un mezzo minuto...) e restituirà una matrice `img` che potrete poi visualizzare a schermo con l'istruzione `imagesc(img)`. Assomiglia alla Figura 1?

## 2 Esercizi facoltativi

*Esercizio 6* (facoltativo). Generate l'immagine corrispondente per il metodo di Newton su altri polinomi. Non sapete le radici? Potete farle calcolare a MATLAB: la funzione `roots(p)` calcola le radici di un polinomio (rappresentato come vettore di coefficienti): per esempio,

```
>> roots([1 -1 -1]) %radici di x^2-x-1=0
ans =
    -0.61803
     1.61803
```

Se volete, potete riscrivere le funzioni scritte finora in modo che il polinomio `p` non sia fissato ma sia uno degli argomenti. Inoltre modificando la funzione `decidi(p)` potete assegnare un valore intero da 1 a  $n$ , con  $n$  grado di `p` ad ognuna delle  $n$  radici complesse di `p`.

### 2.1 Frattale di Julia

Il *frattale di Julia* relativo al numero complesso  $c$  è definito come l'insieme dei punti  $z_0$  per cui la successione definita da  $z_{k+1} = z_k^2 + c$  non diverge.

*Esercizio 7* (facoltativo). Scrivete una `function img=julia(c)` che restituisca il frattale di Julia associato a  $c$ . La funzione:

- genera 101 valori equispaziati nell'intervallo  $[-2, 2]$  con l'istruzione `range=-2:0.04:2`.
- per ogni coppia  $(s, t)$ :
  - calcola il punto  $z_0$  del piano complesso `z0=range(s)+1i*range(t)`;
  - applica per 10 volte la funzione  $f(z) = z^2 + c$  a partire dal punto  $z_0$
  - scrive in `img(t,s)` l'*arcotangente* del modulo del numero complesso  $z_{10}$  così calcolato. Difatti i numeri hanno variazioni molto grosse (da 0 a  $10^{300} \dots$ ), e disegnarli così come sono non produrrebbe un risultato interessante.

<sup>1</sup>potete dare per scontato che il metodo di Newton converga per tutti i valori iniziali nel nostro range.

- restituisce la matrice `img`

Provare la funzione con i seguenti valori di  $c$  :  $1 - \phi$ ;  $(\phi - 2) + (\phi - 1) * 1i$ ;  $1i$ ;  $-0.8 + 0.156i$ , dove  $\phi$  è la sezione aurea cioè  $\phi = (1 + \sqrt{5})/2$ .

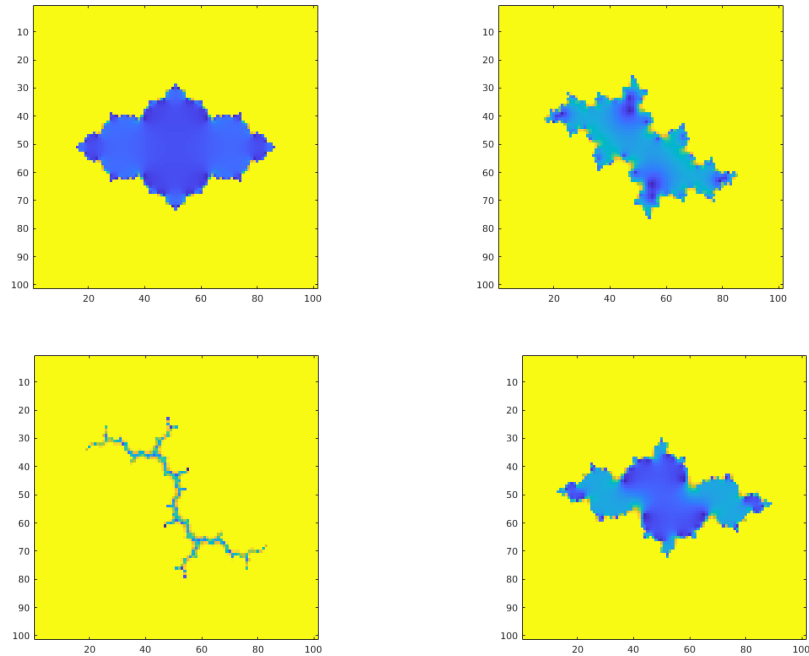


Figura 3: Alcuni frattali di Julia

## 2.2 Versioni vettorizzate

Il programma `newton.m` è abbastanza lento; difatti non abbiamo dedicato alcuna attenzione all'ottimizzazione del numero di istruzioni eseguite (sono questioni tecniche e noiose — non è lavoro per un matematico). Un primo passo per velocizzarlo è sostituire `horner` e `derivata` con le funzioni equivalenti che già esistono in MATLAB, `polyval` e `polyder` (controllare la sintassi con `help`).

Un altro miglioramento potete provare a farlo da soli:

*Esercizio 8* (facoltativo). Utilizzando le funzioni `polyval` e `polyder` applicate con una matrice come secondo argomento (`help polyval`) e le operazioni elemento-per-elemento (ad esempio `.*`, `./`), scrivete un'istruzione che esegua un passo del metodo di Newton *contemporaneamente* su tutti i valori contenuti in una matrice  $X$ , ed implementatelo in

una function `disegna_newton.m`. In alternativa, modificate `horner` per accettare un input  $x$  vettoriale (o matriciale), ed effettuare la valutazione component-wise.

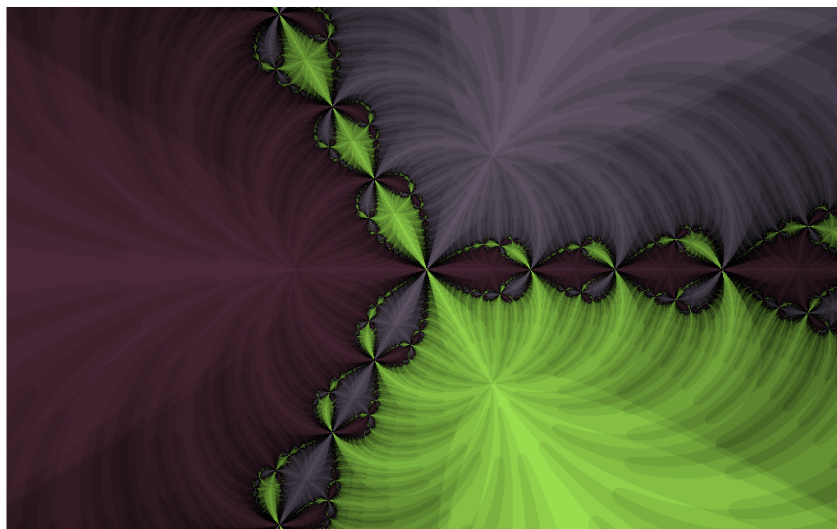
Un po' più complicato infine è rendere efficiente la funzione `decidi` per Newton. Su Moodle trovate delle versioni più veloci delle funzioni che disegnano i frattali di Newton (`disegna_newton.m`) e Julia (`julia2.m`), che utilizzano tutte queste ottimizzazioni. Se volete potete usarle per generare immagini più grandi o per zoomare su alcuni dettagli e studiare la forma dei due frattali.

Un'altra modifica interessante è generare un'immagine a colori, che in MATLAB viene rappresentate da un "tensore"  $n \times n \times 3$ , dove la terza dimensione indica le componenti di rosso, verde, e blu, rispettivamente (formato RGB). Questo tipo di immagine si può visualizzare con il comando `imshow`.

*Esercizio 9* (facoltativo). Modificate `disegna_newton.m` per generare una seconda immagine a colori in cui l'intensità di colore sia determinata dal numero di iterazioni necessarie per arrivare a convergenza. Se riuscite, gestite anche il caso di immagini rettangolari, e provate con altri polinomi.

Quest'ultima versione di `disegna_newton.m` dovrebbe generare un'immagine come la seguente. Se non avete tempo / non riuscite a finire l'esercizio, potete scaricare la soluzione da Moodle, e provarla con le istruzioni:

```
>> [img, img2] = disegna_newton(640, 1024);  
>> imshow(img2);
```



Leggete il codice e cercate di capire come funziona! Potete sperimentare i risultati anche con altri polinomi, e/o con polinomi random (trovate già degli indizi su come fare fra i commenti).