Calcolo Numerico A/B
Esercitazione di Laboratorio 2

Gianna Del Corso <delcorso@di.unipi.it>

31 Ottobre 2013

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

1 Vettori e matrici

Octave è pensato per lavorare con vettori e matrici; pertanto, ha una sintassi specifica e parecchi comandi dedicati, che rendono molto più semplice lavorare con i vettori rispetto a un linguaggio generico come il C.

1.1 Creare vettori e matrici

```
octave:1> A=[1 2 3; 4 5 6]
A =

   1 2 3
   4 5 6
```

```
 octave:1> zeros(3,2)
ans =

   0 0
   0 0
   0 0

octave:2> ones(3,2)
ans =
```

```
   1  1
   1  1
   1  1

octave:3> eye(3)
ans =

   1  0  0
   0  1  0
   0  0  1

octave:4> randn(2,3)
ans =

   0.567178 -0.126397 -0.090664
  -0.678601  0.504481  0.754911
```

1.2 Il *range operator* :

Con la sintassi `a:t:b` creiamo un vettore (riga) che contiene gli elementi `a`, `a+t`, `a+2t`... fino a `b` (o fino all'ultimo che sia minore o uguale a `b`). Se `t`=1, può essere omesso.

```
 octave:6> 1:0.5:4
ans =

   1.0000  1.5000  2.0000  2.5000  3.0000  3.5000  4.0000

octave:7> 1:10
ans =

   1  2  3  4  5  6  7  8  9  10

octave:8> 1:2:10
ans =

   1  3  5  7  9
```

Dove avete già usato l'operatore :?

1.3 Accedere agli elementi

```
octave:16> A=ones(2,3)
A =
```

```
   1 1 1
   1 1 1

octave:17> A(1,2)=2
A =

   1 2 1
   1 1 1

octave:18> A(1,2)
ans = 2
octave:19> A(5,10)
error: invalid row index = 5
error: invalid column index = 10
octave:19> A(5,10)=7
A =

   1 2 1 0 0 0 0 0 0 0
   1 1 1 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 0
   0 0 0 0 0 0 0 0 0 7
```

Se cerco di *leggere* un elemento che non esiste (perché la matrice è troppo piccola), ottengo un errore. Se cerco di *scrivere* un elemento che non esiste, la matrice viene automaticamente ingrandita.

1.4 Operazioni su vettori

```
octave:20> a=1:3
a =

  1 2 3

octave:21> b=4:6
b =

  4 5 6

octave:22> a+b
ans =

   5 7 9

octave:23> sin(a)
ans =
```

```
   0.84147 0.90930 0.14112

octave:24> 2*a+1
ans =

   3 5 7

octave:25> a.*b %operazioni elemento per elemento
ans =

    4 10 18

octave:26> c=a' %matrice trasposta
c =

   1
   2
   3

octave:27> C=a'*b %prodotto matrice-matrice
C =

    4 5 6
    8 10 12
   12 15 18

octave:28> length(a) %lunghezza di un vettore
ans = 3

octave:28> size(C) %dimensioni di una matrice - (righe, colonne)
ans =

   3 3
```

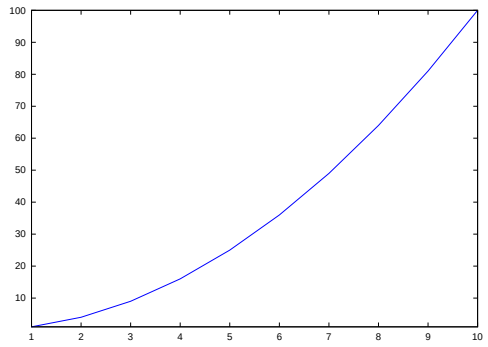
2 Grafici

Il comando `plot(x,y)` prende come argomenti due vettori della stessa lunghezza `x` e `y` e disegna sul piano cartesiano i punti `x(i),y(i)` collegandoli con una linea.

```
octave:28> r=1:10
r =

   1 2 3 4 5 6 7 8 9 10

octave:30> plot(r,r.^2)
```



Il seguente comando disegna un cerchio.

```
octave:23> t=0:.001:2*pi;
octave:24> plot(cos(t),sin(t))
```

Una funzione si può definire con il comando `@`. Ecco un esempio nel quale definisco la funzione $f(x) = x^2 + cos(x)$

```
octave:25>f=@(x)x.^2+cos(x)-1;
```

L'operatore . permette di valutare la funzione direttamente su un vettore. Quanto vale $f(\pi/2)$? Se $x = -6 : 0.5 : 6$ provate a fare $f(x)$. Per disegnare la funzione posso semplicemente fare

```
octave:26>x=-6:0.05:6;
octave:27>plot(x, f(x))
```

3 Metodo delle Tangenti

Esercizio 1. Scrivere una `function y=tangenti(f, d, x0, niter)` che prende una funzione f e la sua derivata d definite con il comando `f=@(x)....` e `d=@(x)....`, il punto iniziale x_0 e il numero di iterazioni niter e restituisce il valore y ottenuto applicando niter iterazioni del metodo delle tangenti applicato a f.

Esercizio 2 (facoltativo). Aggiungere la grafica, disegnando il grafico della funzione, e le varie iterazioni. (Suggerimento: ricordarsi che per disegnare la retta passante per i punti (x_0, y_0) e (x_1, y_1) posso usare il comando `plot([x_0, x_1], [y_0, y_1])`.)

Esercizio 3. Si consideri l'equazione $(x+3)(x-1)(x-4) = x^3+6x^2-11x+12 = 0$ e si applichi la funzione `tangenti` a partire dal punto $x_0 = 4.9$. Si osservi la convergenza alla radice 4, nella quale ad ogni iterazione l'errore tra la soluzione calcolata e la soluzione esatta raddoppia la precisione. Cosa succede all'equazione $(x+3)^2(x-1)(x-4) = 0$ quando si approssima la soluzione -3 a partire da -3.9? Sapete capire il motivo? Cosa osservate partendo dal punto $x_0 = 4.9$?

4 Frattali di Newton

Cercheremo ora di disegnare i frattali che si ottengono disegnando i bacini di attrazione del metodo delle tangenti (sul piano complesso) per un polinomio. Le immagini risultanti dovrebbero assomigliare a quella in figura 4. Notate la simmetria della figura:

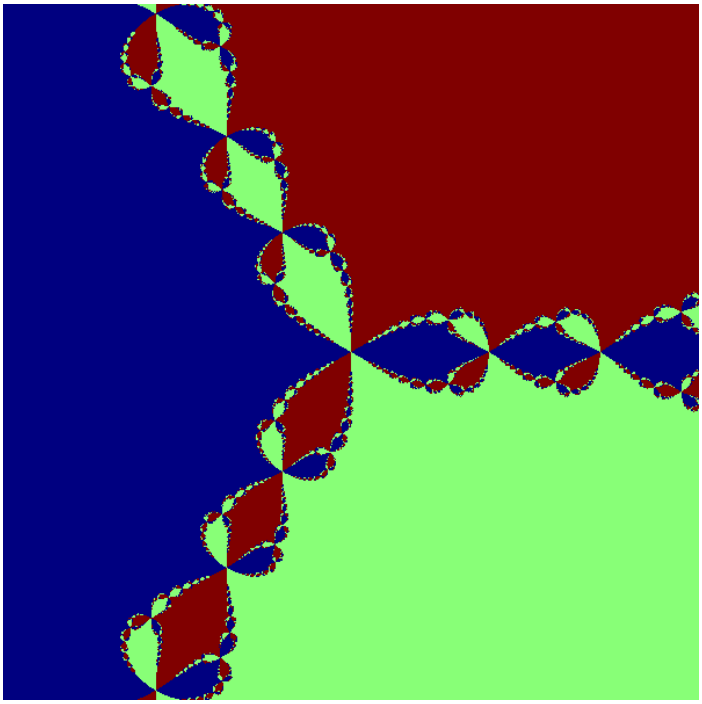


Figura 1: Disegno dei bacini di attrazione del metodo di Newton per il polinomio x^3+1. I tre colori diversi corrispondono ai punti del piano complesso a partire da cui Newton converge alle tre diverse radici.

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.

4.1 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,

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

```
function y=horner(p,x)
n=length(p);
y=p(1);
for i=2:n
 y=y*x+p(i);
endfor
endfunction
```

Esercizio 4. Scrivere una `function dp=derivata(p)` che prende un polinomio p (vettore di coefficienti) e restituisce la sua derivata (vettore di coefficienti).

4.2 Metodo di Newton per polinomi

Il metodo di Newton è l'iterazione

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

Esercizio 5. 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+3I`.

4.3 "Disegnare" una matrice

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

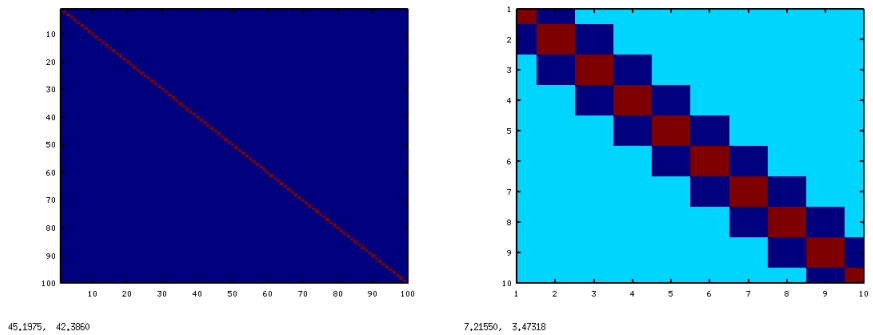


Figura 2: `imagesc(eye(100))` e `imagesc(laplacian(10))` una particolare matrice tridiagonale

```
octave:1> imagesc(eye(100))
```

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

4.4 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 6. Copiare la seguente 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 $\frac{1}{2} - I\frac{\sqrt{3}}{2}$.

```
function val=decidi(x)
 d1=norm(x-1);
 d2=norm(x-(1+sqrt(3)*I)/2);
 d3=norm(x-(1-sqrt(3)*I)/2);
 [inutile, val]=min([d1 d2 d3]);
endfunction
```

Esercizio 7. Scrivere una `function img=newtonfractal()` che non prende alcun argomento e restituisce una matrice 101×101 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 (i, j):
  - calcola il punto z_0 del piano complesso `z0=range(i)+1I*range(j)`;
  - 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(i,j)` a seconda della radice del polinomio a cui si ha convergenza a partire dal valore iniziale `z0`.[1].

- 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 4?

5 Esercizi facoltativi

Esercizio 8 (facoltativo). Generate l'immagine corrispondente per il metodo di Newton su altri polinomi. Non sapete le radici? Potete farle calcolare a Octave: la funzione `roots(p)` calcola le radici di un polinomio (rappresentato come vettore di coefficienti): per esempio,

```
octave:2> 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.

5.1 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 9 (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 (i, j):
  - calcola il punto z_0 del piano complesso `z0=range(i)+1I*range(j)`;
  - applica per 10 volte la funzione $f(z) = z^2 + c$ a partire dal punto z_0
  - scrive in `img(i,j)` l'*arcotangente* del modulo del numero complesso z_{10} così calcolato. Difatti i numeri hanno variazioni molto grosse (da 0 a 10^{300}...), e disegnarli così come sono non produrrebbe un risultato interessante.
- restituisce la matrice `img`

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

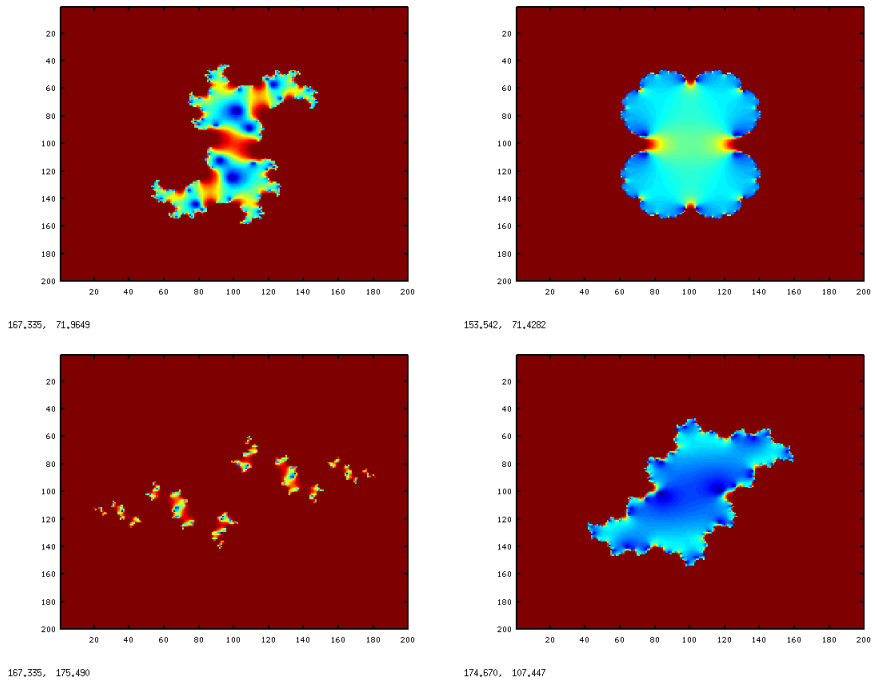


Figura 3: Alcuni frattali di Julia

5.2 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). Un primo passo per velocizzarlo è sostituire `horner` e `derivata` con le funzioni equivalenti che già esistono in Octave, `polyval` e `polyderiv` (controllare la sintassi con `help`).

Un altro miglioramento potete provare a farlo da soli:

Esercizio 10 (facoltativo). Utilizzando le funzioni `polyval` e `polyderiv` 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.

Un po' più complicato infine è rendere efficiente la funzione `decidi` per Newton. Riportiamo qui una versione più veloce delle funzioni che disegnano i frattali di Newton e Julia, che utilizza tutte queste ottimizzazioni. Se volete potete usarli per generare immagini più grandi o per zoomare su alcuni dettagli e studiare la forma dei due frattali.

```
function img=disegna_newton(dimensione)
```

```
%restituisce un frattale di Newton nxn. Visualizzare con imagesc(img)
p=[1;0;0;1]; %coefficienti del polinomio
n=dimensione;
range=-2:4/(n-1):2; %griglia di punti

deg=length(p)-1;
dp=polyderiv(p);
radici=roots(p);

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

%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" con un trucco:
%
%generiamo un "tensore", cioe' un oggetto a tre indici
%M(i,j,k)=X_{j,k}-radici(i)
%
%usiamo la funzione min: [values,positions]=min(A) calcola il minimo
%lungo la prima dimensione (righe)
%e restituisce in positions le /posizioni/ in cui si trovano
%i minimi su ogni riga
%ad es. [v p]=min([4.5 2.5 8 9]) restituisce v=2.5, p=2
%(perche' il minimo sta in posizione 2)
M=zeros(deg,n,n);
for i=1:deg
  M(i,:,:)=abs(X-radici(i));
endfor
[values positions]=min(M);

%ritrasforma positions da un "tensore" 1xnxn a una matrice nxn
img=reshape(positions,[n n]);
endfunction
```

```
function img=julia(c)
%function img=julia(c)
%restituisce il disegno del frattale di Julia con parametro c
%plottare con imagesc(julia(c))
n=200;
range=-2:4/(n-1):2; %griglia di punti
```

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

for k=1:10
  X=X.*X+c;
endfor

img=atan(abs(X));
endfunction
```