Sistemi di equazioni nonlineari

1 giugno 2008

1 Un esempio introduttivo sul metodo di punto fisso

Il problema della soluzione di sistemi nonlineari è di importanza fondamentale in vari aspetti della modellistica matematica. Ad esempio è fondamentale nella risoluzione di equazioni differenziali nonlineari con metodi impliciti.

Data una funzione $F:\mathbb{R}^n \rightarrow R^n$ si tratta di trovare gli x^* tali che $F(x^*)=0$ (cf. [6, p.254]). Vediamone un esempio. Supponiamo di dover risolvere il sistema nonlineare (cf. [3, p.449])

$$\begin{cases} x - \frac{1}{2}\cos(y) = 0 \\ y - \frac{1}{2}\sin(x) = 0 \end{cases} \tag{1}$$

Per prima cosa ci si chiede quante soluzioni abbia questo problema. Dal punto di vista pratico, sostituendo

$$y = \frac{1}{2}\sin(x)$$

nella prima equazione, otteniamo

$$x = \frac{1}{2}\cos\left(\frac{1}{2}\sin(x)\right).$$

La funzione

$$g(x) = \frac{1}{2}\cos\left(\frac{1}{2}\sin(x)\right)$$

è tale che

$$g^{(1)}(x) = (-1/4)\sin((1/2)*\sin(x))\cos(x).$$

Per convincersene, usando il calcolo simbolico in Matlab

```
>> syms x
>> g=(1/2)*cos((1/2)*sin(x))
g =
1/2*cos(1/2*sin(x))
>>
>> % DERIVATA PRIMA.
>>
>> diff(g)
```

```
ans =
-1/4*sin(1/2*sin(x))*cos(x)
>>
>> % DERIVATA SECONDA.
>>
>> diff(g,2)
ans =
-1/8*cos(1/2*sin(x))*cos(x)^2+1/4*sin(1/2*sin(x))*sin(x)
>>
```

ed essendo

$$0 \leq |\sin\left(\frac{sin(x)}{2}\right) \cdot \cos(x)| \leq 1$$

si ha che

$$|g^{(1)}(x)| \leq \frac{1}{4}, \forall x \in \mathbb{R}.$$

Dalla formula di Taylor, centrata in un arbitrario x_0

$$g(x) = g(x_0) + g^{(1)}(\xi)(x - x_0)$$

e quindi portando $g(x_0)$ a primo membro e calcolando il modulo ad ambo i membri

$$|g(x) - g(x_0)| = |g^{(1)}(\xi)||(x - x_0)| \leq \frac{1}{4}|(x - x_0)|.$$

Ricordiamo ora che se d è una distanza, e (X, d) uno spazio metrico (come ad esempio $(\mathbb{R}, d)$ con $d(x, y) = |x - y|$), M un sottospazio non vuoto di X allora $F : M \to M$ è L contrattiva in M se e solo se

$$|F(x) - F(y)| \leq L|x - y|, \forall x, y \in M.$$

Per quanto visto abbiamo provato che

$$g(x) = \frac{1}{2}\cos\left(\frac{1}{2}\sin(x)\right)$$

è L contrattiva in $M = \mathbb{R}$ per $L = 1/4$.
Banach provò nel 1922 il seguente (cf. [3, p.432])

Teorema 1.1 *Sia (X, d) uno spazio metrico completo ed M un sottospazio non vuoto e chiuso di X. Se $T : M \to M$ è L contrattiva allora*

1. *l'equazione $x = T(x)$ ha una ed una sola soluzione α;*
2. *la successione $x_{k+1} = g(x_k)$ (detta di Banach o di punto fisso) converge alla soluzione α per ogni scelta del punto iniziale x_0;*
3. *per $k = 0, 1, 2, \ldots$ si ha*

$$d(x_k, \alpha) \leq L^k(1 - L)^{-1} d(x_0, x_1), \text{ a priori}$$

$$d(x_{k+1}, \alpha) \leq L(1 - L)^{-1} d(x_{k+1}, x_k), \text{ a posteriori}$$

4. *per* $k = 0, 1, 2, \ldots$ *si ha*
$$d(x_{k+1}, \alpha) \le L d(x_k, \alpha)$$

Dal teorema di Banach deduciamo che
$$g(x) = \frac{1}{2}\cos\left(\frac{1}{2}\sin(x)\right)$$
ha una ed una sola soluzione e quindi pure
$$\begin{cases} x - \frac{1}{2}\cos(y) = 0 \\ y - \frac{1}{2}\sin(x) = 0 \end{cases} \tag{2}$$

Vediamo i dettagli in Matlab. Scriviamo il codice `punto_fisso`

```
function xhist=punto_fisso(x0, maxit, tol, g)

% INPUT.
% x0      : PUNTO INIZIALE.
% maxit   : NUMERO MASSIMO DI ITERAZIONI.
% tol     : TOLLERANZA CRITERIO DI ARRESTO.
% g       : EQUAZIONE DA RISOLVERE x=g(x)

x_old=x0;
xhist=[x_old];

for index=1:maxit
    x_new=feval(g,x_old);
    if norm(x_new-x_old,inf) < tol
       return
   else
      xhist=[xhist; x_new];
      x_old=x_new;
   end
end

fprintf('\n \t [WARNING]: RAGGIUNTO MASSIMO NUMERO DI ITERAZIONI ');
```

Applichiamo la successione descritta nel teorema di Banach per risolvere il problema $x = g(x)$,

```
>> g=inline('(1/2)*cos((1/2)*sin(x))');
>> xhist=punto_fisso(0, 50, 10^(-15), g);
>> format long
>> xhist
xhist =
                  0
   0.50000000000000
   0.48570310513698
   0.48644107261515
   0.48640331614624
```

```
   0.48640524877124
   0.48640514984910
   0.48640515491247
   0.48640515465330
   0.48640515466657
   0.48640515466589
   0.48640515466592
   0.48640515466592
>> feval(g,0.48640515466592)
ans =
   0.48640515466592
>>
```

Nelle ultime due righe dell'esperimento abbiamo verificato che effettivamente per $t = 0.48640515466592$, $t \approx g(t)$.
Vediamo se anche il sistema nonlineare

$$\begin{cases} x - \frac{1}{2}\cos(y) = 0 \\ y - \frac{1}{2}\sin(x) = 0 \end{cases} \tag{3}$$

è risolto correttamente.

- Se $x = 0.48640515466592$, allora dalla seconda equazione

$$y = \frac{1}{2}\sin(0.48640515466592) \approx 0.23372550195872.$$

- Verifichiamo che anche la prima equazione sia risolta. In effetti si ha $x = 0.48640515466592$ e
$$\frac{1}{2}\sin(x) \approx 0.23372550195872$$
che coincide proprio con y.

In Matlab/Octave

```
>> format long;
>> x=0.48640515466592;
>> y=0.5*sin(x);
>> y
y =
   0.23372550195872
>> 0.5*cos(y)
ans =
   0.48640515466592
>>
```

Dal punto di vista pratico abbiamo risolto un sistema nonlineare in due variabili e due incognite come un'equazione lineare in un'incognita. Evidentemente ciò non è sempre possibile. Riproponiamo quanto visto nell'esempio precedente

$$\begin{cases} x - \frac{1}{2}\cos(y) = 0 \\ y - \frac{1}{2}\sin(x) = 0 \end{cases}$$

in modo differente.

Siano $\mathbf{v} = (v_i)_i = [x;\ y]$

$$\begin{aligned} T_1(\mathbf{v}) &= \frac{1}{2}\cos(v_2) = 0 \\ T_2(\mathbf{v}) &= \frac{1}{2}\sin(v_1) = 0. \end{aligned} \tag{4}$$

per $T(\mathbf{v}) = [T_1(\mathbf{v});\ T_1(\mathbf{v})]$ abbiamo $\mathbf{v} = T(\mathbf{v})$. Mostriamo che $T : \mathbb{R}^2 \to \mathbb{R}^2$ è contrattiva. Dalla formula di Taylor in più variabili (cf. [4, p.192]),

$$T(\mathbf{v}) = T(\mathbf{v_0}) + (T'(\xi)) \cdot (\mathbf{v} - \mathbf{v_0}),\ \xi \in S$$

essendo S il segmento che congiunge $\mathbf{v}$ con $\mathbf{v_0}$ e T' la matrice Jacobiana definita per componenti come

$$(T')_{j,k}(\mathbf{v}) = \frac{\partial T_j(\mathbf{v})}{\partial v_k}.$$

Nel nostro esempio quindi,

$$(T')_{j,k}(\mathbf{v}) = \begin{pmatrix} 0 & (-1/2)\sin(v_1) \\ (+1/2)\cos(v_2) & 0 \end{pmatrix}$$

Essendo la norma infinito di una matrice A definita da

$$\|A\|_\infty = \max_j \sum_i |a_{i,j}|$$

si ha

$$\begin{aligned} \|(T')(\mathbf{v})\|_\infty &= \max_j \sum_i |\big((T')(\mathbf{v})\big)_{i,j}| \qquad (5) \\ &= \max(|(-1/2)\sin(v_1)|, |(+1/2)\cos(v_2)|) \leq \frac{1}{2} \qquad (6) \end{aligned}$$

Dalla formula di Taylor si ha così che

$$T(\mathbf{v}) = T(\mathbf{v_0}) + (T'(\xi)) \cdot (\mathbf{v} - \mathbf{v_0}),\ \xi \in S$$

e calcolando la norma infinito di ambo i membri 0

$$\|T(\mathbf{v}) - T(\mathbf{v_0})\|_\infty \leq \|(T'(\xi))\|_\infty \|(\mathbf{v} - \mathbf{v_0})\|_\infty,\ \xi \in S$$

da cui $T : \mathbb{R}^2 \to \mathbb{R}^2$ è una L-contrazione con $L = 1/2$.

Possiamo nuovamente applicare il metodo di Banach (detto anche di *punto fisso*), questa volta però non per risolvere l'equazione $x = g(x)$, bensì direttamente il sistema nonlineare $\mathbf{v} = T(\mathbf{v})$.

Scriviamo il file g1

```
function res=g1(v)

res(1,1)=0.5*cos(v(2));
res(2,1)=0.5*sin(v(1));
```

e quindi dalla shell di Matlab/Octave

```
>> format long;
>> x0=[0; 0];
>> maxit=50;
>> tol=10^(-15);
>> xhist=punto_fisso(x0, maxit, tol, 'g1')
xhist =
                  0                   0
   0.50000000000000                   0
   0.50000000000000    0.23971276930210
   0.48570310513698    0.23971276930210
   0.48570310513698    0.23341513183103
   0.48644107261515    0.23341513183103
   0.48644107261515    0.23374137788239
   0.48640331614624    0.23374137788239
   0.48640331614624    0.23372468931518
   0.48640524877124    0.23372468931518
   0.48640524877124    0.23372554355416
   0.48640514984910    0.23372554355416
   0.48640514984910    0.23372549982964
   0.48640515491247    0.23372549982964
   0.48640515491247    0.23372550206770
   0.48640515465330    0.23372550206770
   0.48640515465330    0.23372550195314
   0.48640515466657    0.23372550195314
   0.48640515466657    0.23372550195901
   0.48640515466589    0.23372550195901
   0.48640515466589    0.23372550195871
   0.48640515466592    0.23372550195871
   0.48640515466592    0.23372550195872
   0.48640515466592    0.23372550195872
>>
```

Vediamo quanto accurato è il teorema di Banach nelle sue stime. Il metodo esegue 24 iterazioni. Calcoliamo

1. Gli errori assoluti compiuti, posto

$$\alpha = [0.486405154665920.23372550195872].$$

2. Essendo $L = 1/2$ dal teorema di Banach si vede che la stima a posteriori afferma

$$d(x_{k+1}, \alpha) \leq d(x_{k+1}, x_k),\ k = 0, 1, \ldots$$

abbiamo

```
>> format short e
>>
>> % CALCOLIAMO ERRORI ASSOLUTI.
>> xhist_err=abs(xhist-repmat([0.48640515466592   0.23372550195872],24,1));
>> xhist_abserr=sqrt((xhist_err(:,1)).^2+(xhist_err(:,2)).^2);
>>
>> % CALCOLIAMO STIMA A POSTERIORI.
>> N=24;
>> xplus=xhist(2:N,:); xminus=xhist(1:N-1,:);
>> xdiff=xplus-xminus;
>> xerr_est=sqrt( (xdiff(:,1)).^2 + (xdiff(:,2)).^2 )
>>
>> % PARAGONIAMO GLI ERRORI A PARTIRE DA k=1. OSSERVIAMO CHE
>> % GLI INDICI IN MATLAB PARTONO DA 1.
>> [xhist_abserr(2:N,:) xerr_est]
ans =
  2.3412e-001  5.0000e-001
  1.4855e-002  2.3971e-001
  6.0283e-003  1.4297e-002
  7.6760e-004  6.2976e-003
  3.1244e-004  7.3797e-004
  3.9270e-005  3.2625e-004
  1.5982e-005  3.7756e-005
  2.0101e-006  1.6689e-005
  8.1807e-007  1.9326e-006
  1.0289e-007  8.5424e-007
  4.1873e-008  9.8922e-008
  5.2664e-009  4.3725e-008
  2.1433e-009  5.0634e-009
  2.6956e-010  2.2381e-009
  1.0971e-010  2.5917e-010
  1.3796e-011  1.1456e-010
  5.6147e-012  1.3266e-011
  7.0775e-013  5.8636e-012
  2.8805e-013  6.7901e-013
  3.4630e-014  3.0012e-013
  1.4144e-014  3.4750e-014
  3.3766e-015  1.5377e-014
  1.9767e-015  1.7764e-015
>>
```

Si osservi che, eccetto nell'ultima riga, la stima dall'alto è effettivamente realizzata. Nell'ultima riga, abbiamo approssimato la soluzione α a 14 cifre dopo la virgola, ragion per cui il risultato non deve essere ivi considerato.

Osservazioni. Si noti che

- in generale un'equazione si scrive come $F(x) = 0$, mentre nella formulazione di punto fisso si descrive come $x = T(x)$; evidentemente non è un problema, in quanto se $F(x) = 0$ allora $x = T(x)$ per $T(x) := F(x) + x$;

- l'esempio mostrato nella sezione è particolarmente semplice, in quanto può essere ricondotto ad un problema unidimensionale; ciononostante abbiamo mostrato che può essere risolto da un metodo che lavora in più variabili.

2 Metodo di Newton

Dalla formula di Taylor (multivariata), se la funzione $F: M \subseteq \mathbb{R}^n \to \mathbb{R}^n$ è sufficientemente regolare, allora per $\mathbf{v_{k+1}}$ sufficientemente vicino a $\mathbf{v_k}$

$$F(\mathbf{v_{k+1}}) \approx F(\mathbf{v}_k) + (F'(\mathbf{v}_k)) \cdot (\mathbf{v_{k+1}} - \mathbf{v}_k)\,. \tag{7}$$

Ricordiamo che $F'(\mathbf{v}_k)$ è una matrice, e il successivo prodotto $\cdot$ è di tipo *matrice per vettore*.

Da (7), supposto che la matrice $F'(\mathbf{v}_0)$ sia invertibile, e che $F(\mathbf{v_{k+1}}) \approx 0$, cioè $\mathbf{v_{k+1}}$ sia una approssimazione di uno zero di F, abbiamo

$$0 \approx F(\mathbf{v_{k+1}}) \approx F(\mathbf{v}_k) + (F'(\mathbf{v}_k)) \cdot (\mathbf{v_{k+1}} - \mathbf{v}_k) \tag{8}$$

da cui ha senso definire la successione (del metodo di Newton)

$$\mathbf{v_{k+1}} := \mathbf{v_k} - (F'(\mathbf{v}_k))^{-1} \cdot F(\mathbf{v}_k). \tag{9}$$

Talvolta, la successione di Newton viene descritta come

$$\begin{cases} F'(\mathbf{v}_k)) \cdot h_{k+1} = -F(\mathbf{v}_k), \\ \mathbf{v_{k+1}} := \mathbf{v_k} + h_{k+1} \end{cases} \tag{10}$$

sottolineando che l'inversa di $F'(\mathbf{v}_k))$ non deve necessariamente essere calcolata, bensì bisogna risolvere un sistema lineare la cui soluzione h_{k+1} è la correzione da apportare a $\mathbf{v_k}$ per ottenere $\mathbf{v_{k+1}}$ (cf. [5, p.174] per un esempio in $\mathbb{R}^2$).
L'analisi di convergenza del metodo di Newton in dimensione maggiore di 1 e più in generale in spazi di Banach è un attivo e difficile argomento di ricerca. A tal proposito si consulti [2, p.154].
Vediamo di rifare l'esempio della sezione precedente, cioè

$$\begin{cases} x - \frac{1}{2}\cos(y) = 0 \\ y - \frac{1}{2}\sin(x) = 0 \end{cases} \tag{11}$$

mediante il codice implementante il metodo di Newton

```
function [v_hist, step_hist, residual_hist]=newton_sistemi(v0,maxit,tol)

% INPUT:
% v0      : APPROSSIMAZIONE INIZIALE.
% maxit   : NUMERO MASSIMO DI ITERAZIONI.
% tol     : TOLLERANZA DEL CRITERIO D'ARRESTO.
```

```
% F       : FUNZIONE.
% DF      : FUNZIONE DA CUI VIENE CALCOLATA LA JACOBIANA DI F.

% OUTPUT:
% v_hist : VETTORE CONTENENTE LE APPROSSIMAZIONI DELLA SOLUZIONE.
% step_hist    : VETTORE DEGLI STEP.
% residual_hist: VETTORE DEI RESIDUI.

v_old=v0;
v_hist=[v0'];


JF=DF(v_old);  % CALCOLIAMO LA MATRICE JACOBIANA.
Fv=F(v_old);   % VALUTIAMO LA FUNZIONE.
residual=norm(Fv);
residual_hist=[residual];

h=-JF\Fv;       % CALCOLIAMO LA CORREZIONE.

v_new=v_old+h;  % NUOVA APPROSSIMAZIONE.
v_hist=[v_hist; v_new'];

step_hist=[];

for index=1:maxit

    Fv=F(v_new);  % VALUTIAMO LA FUNZIONE.
    residual=norm(Fv,2);             % RESIDUO.
    residual_hist=[residual_hist; residual];

    loc_step=norm(v_new-v_old,2);    % STEP.
    step_hist=[step_hist; loc_step];

    if max(loc_step,residual) < tol
        return;
    else
        v_old=v_new;
        JF=DF(v_old);  % CALCOLIAMO LA MATRICE JACOBIANA.

        h=-JF\Fv;       % CALCOLIAMO LA CORREZIONE.
        v_new=v_old+h;  % NUOVA APPROSSIMAZIONE.
        v_hist=[v_hist; v_new'];
    end

end

fprintf('\n \t [WARNING]: NUMERO MASSIMO DI ITERAZIONI RAGGIUNTO.')


function res=F(v)
```

```
res(1,1)=0.5*cos(v(2))-v(1);
res(2,1)=0.5*sin(v(1))-v(2);


function res=DF(v)

% res(1,1)=0.5*cos(v(2));
% res(2,1)=0.5*sin(v(1));

res=[-1 -0.5*sin(v(2)); 0.5*cos(v(1)) -1];
```

Quindi scriviamo il programma principale `driver_newton` che setta i parametri della procedura `newton_sistemi`

```
clear;

v0=[0; 0]; maxit=50; tol=10^(-10);

[v_hist, step_hist, residual_hist]=newton_sistemi(v0,maxit,tol);
```

Otteniamo così

```
>> driver_newton
>> v_hist
v_hist =
            0            0
  5.0000e-001  2.5000e-001
  4.8646e-001  2.3377e-001
  4.8641e-001  2.3373e-001
  4.8641e-001  2.3373e-001
  4.8641e-001  2.3373e-001
>> step_hist
step_hist =
  5.5902e-001
  2.1132e-002
  7.5293e-005
  7.7616e-010
            0
>> residual_hist
residual_hist =
  5.0000e-001
  1.8640e-002
  6.7480e-005
  6.7927e-010
            0
            0
>>
```

Dal vettore degli step $|x_{k+1} - x_k|$ e da quello dei residui $|F(x_k)|$ si capisce che il metodo ha convergenza superlineare (in realtà è quadratica).

Se consideriamo quale soluzione il risultato fornito dall'ultima iterazione (il residuo è 0!), possiamo calcolare gli errori assoluti ad ogni iterazione, convincendoci una volta di più della convergenza superlineare del metodo.

```
>> alpha=v_hist(size(v_hist,1),:);
>> err_vett=v_hist-repmat(alpha,size(v_hist,1),1);
>> abs_err_vett=sqrt( (err_vett(:,1)).^2 + (err_vett(:,2)).^2 )
abs_err_vett =
  5.3965e-001
  2.1206e-002
  7.5294e-005
  7.7616e-010
            0
            0
>>
```

Osservazione. Il comando Matlab `repmat` è di uso molto comune. Nel nostro caso volevamo costruire, alla riga

```
err_vett=v_hist-repmat(alpha,size(v_hist,1),1);
```

la differenza tra la matrice `v_hist` e la matrice in cui ogni riga è costituita dal vettore riga soluzione $(\alpha_1\ \alpha_2)$. Per avere le dimensioni corrette per poter eseguire la differenza abbiamo dovuto replicare il vettore riga α tante volte quanto il numero di righe di `v_hist`, cosa eseguita appunto dal comando `repmat`.

Vediamo un esempio di `repmat`.

```
>> v=[1 2];
>> repmat(v,3,1)
ans =
     1     2
     1     2
     1     2
>> repmat(v,3,2)
ans =
     1     2     1     2
     1     2     1     2
     1     2     1     2
>>
```

In pratica, `repmat(v,m,n)` costruisce una matrice $m \times n$ in cui ogni componente è uguale al vettore v, come si può evincere da

```
>> help repmat

 REPMAT Replicate and tile an array.
    B = repmat(A,M,N) creates a large matrix B consisting of an M-by-N
    tiling of copies of A.
```

```
    B = REPMAT(A,[M N]) accomplishes the same result as repmat(A,M,N).

    B = REPMAT(A,[M N P ...]) tiles the array A to produce a
    M-by-N-by-P-by-... block array.  A can be N-D.

    REPMAT(A,M,N) when A is a scalar is commonly used to produce
    an M-by-N matrix filled with A's value.  This can be much faster
    than A*ONES(M,N) when M and/or N are large.

    Example:
        repmat(magic(2),2,3)
        repmat(NaN,2,3)

    See also MESHGRID.

>>
```

References

[1] K. Atkinson, *An Introduction to Numerical Analysis*, Wiley, (1989).

[2] K. Atkinson e W. Han, *Theoretical Numerical Analysis*, Springer Verlag, (2001).

[3] V. Comincioli, *Analisi Numerica, metodi modelli applicazioni*, Mc Graw-Hill, 1990.

[4] G. Gilardi, *Analisi due*, seconda edizione, Mc Graw-Hill, 1996.

[5] J.H. Mathews e K.D. Fink, *Numerical methods using Matlab*, terza edizione, Prentice-Hall, 1999.

[6] A. Quarteroni, F. Saleri *Introduzione al Calcolo Scientifico*, Springer, (2002).

[7] The MathWorks Inc., *Numerical Computing with Matlab*, http://www.mathworks.com/moler.

[8] G. Rodriguez, *Algoritmi numerici*, Pitagora editrice, 2008.