Cognome: _____ Nome: _____ Matricola: _____

Calcolo Numerico - Corsi di Laurea Area Informatica
Docente: P. Causin
Appello del 22/06/2006 - 3 Esercizi da svolgere in 2 ore

Per ogni punto degli esercizi seguenti, scrivere, oltre alla soluzione, i principali comandi Matlab utilizzati; tracciare inoltre, se ve ne sono, una copia qualitativa dei grafici ottenuti.

ESERCIZIO 1 [12 punti]

Si consideri il metodo di Newton applicato alla ricerca dello zero α della funzione $f(x)$.

1. Si applichi il metodo di Newton (codice `newton.m` sviluppato durante il corso) per la ricerca dello zero della funzione $f(x) = \text{atan}(x)$. Si prenda $x_0 = 10$, `toll =1e-6`, `nmax=200`. Come si comporta il metodo? Perché?
2. Si consideri la seguente tecnica, detta *line search*, da applicare ad ogni passo k del metodo di Newton:

   1. $s = -f(x_k)/f'(x_k)$
   2. $guess = x_k + s$
   3. `while` $(|f(guess)| > |f(x_k)|)$
   4. $\quad s = s/2$
   5. $\quad$ `ripetere` 2.
   6. `end`
   7. `porre` $x_{k+1} = guess$

   Si implementi l'algoritmo di *line search*, modificando opportunamente il codice `newton.m` (*Suggerimento. Si scriva su carta il metodo di Newton e si veda come agisce il line search*).
3. Si applichi il metodo di Newton così modificato alla ricerca dello zero di $f(x) = \text{atan}(x)$ con gli stessi parametri usati precedentemente. Cosa si trova?
4. Quante riduzioni dell'incremento s sono necessarie per ogni iterazione di Newton? Quale è il vettore delle iterate successive accettate (vettore generato alla linea 7.)?

SOLUZIONE

Lanciando il codice di Newton semplice con $x_0 = 10$, otteniamo che il metodo diverge. Implementiamo allora la tecnica di *line search* suggerita. Il codice è il seguente:

```
function [zero,xvect,xdif,fx,it]=newtonlinesearch(xv,nmax,toll,fun,dfun)

err=1;
it=0;
```

```
fxv=fun(xv);
xvect=[xv];
fx=[fxv];
xdif=0;
%
while it<nmax & err>toll
   if dfun(xv) == 0
      disp(' Arresto per azzeramento di dfun');
      it=it+1;
      break
   else

      s=-fun(xv)/dfun(xv);
      guess=xv+s;

      while(abs(fun(guess))>abs(fun(xv)))
           s=s/2;
   guess=xv+s;
      end
      xn=guess;

      %test d'arresto ||xn-xv||<toll
      err=abs(xn-xv);
      xdif=[xdif;err];
      fxv=fun(xn);
      xvect=[xvect;xn];
      fx=[fx;fxv];
      it=it+1;
      xv=xn;
   end
end
%

fprintf(' \n Numero di Iterazioni : %d \n',it);
fprintf(' Zero calcolato     : %20.16f \n',xvect(end));
fprintf(' f(zero calcolato)  : %20.16f \n', fx(end));
zero=xvect(end);
%
return
```

Lanciando il codice, otteniamo che il metodo converge in 12 iterazioni:

```
>> fun=inline('atan(x)','x');
>> dfun=inline('1./(1+x.^2)','x');
>> [xvect,xdif,fx,it]=newtonlinesearch(10,200,1e-6,fun,dfun);
```

```
 Numero di Iterazioni : 12
 Zero calcolato     :   0.0000000000000000
 f(zero calcolato)  :   0.0000000000000000
```

Per contare quante riduzioni del passo sono necessarie ad ogni iterazione, introduciamo un contatore `itlinesearch`, che è un vettore la cui i-esima componente contiene il numero di riduzioni effettuate all'iterazione i-esima. Il codice è il seguente

```
function [zero,xvect,xdif,fx,it]=newtonlinesearch(xv,nmax,toll,fun,dfun)
err=1;
it=0;
fxv=fun(xv);
xvect=[xv];
fx=[fxv];
xdif=0;
%

itlinesearch=zeros(nmax,1);

while it<nmax & err>toll
   if dfun(xv) == 0
      disp(' Arresto per azzeramento di dfun');
      it=it+1;
      break
   else

      s=-fun(xv)/dfun(xv);
      guess=xv+s;

      while(abs(fun(guess))>abs(fun(xv)))
           s=s/2;
   guess=xv+s;
           itlinesearch(it+1)=itlinesearch(it+1)+1;
      end

      xn=guess;

      %test d'arresto ||xn-xv||<toll
      err=abs(xn-xv);
      xdif=[xdif;err];
      fxv=fun(xn);
      xvect=[xvect;xn];
      fx=[fx;fxv];
      it=it+1;
      xv=xn;
   end
end
```

```
%

fprintf(' \n Numero di Iterazioni : %d \n',it);
fprintf(' Zero calcolato     : %20.16f \n',xvect(end));
fprintf(' f(zero calcolato)  : %20.16f \n', fx(end));
zero=xvect(end);
%
return
```

otteniamo i seguenti valori

```
itlinesearch(1:it)

ans =

     3
     3
     2
     2
     0
     0
     0
     0
     0
     0
     0
     0
```

Il vettore delle iterate successive accettate è il vettore `xvect` prodotto dal codice.

ESERCIZIO 2 [12 punti]

Per la risoluzione del sistema lineare $Ax = b$ si consideri il seguente metodo iterativo:

$$(I - \omega D^{-1}E)\mathbf{x}^{(k+1)} = [(1-\omega)I + \omega D^{-1}F]\mathbf{x}^{(k)} + \omega D^{-1}\mathbf{b}$$

dove $A = D - E - F$, D essendo la parte diagonale di A, E la parte triangolare inferiore con segno cambiato, F la parte triangolare superiore con segno cambiato, rispettivamente, e dove ω è un parametro reale.

1. Si scriva la matrice di iterazione B_ω del metodo sopraindicato, tale che

$$\mathbf{x}^{(k+1)} = B_\omega \mathbf{x}^{(k)} + \mathbf{g}_\omega$$

Si tracci un grafico del raggio spettrale di B_ω in funzione di ω. Cosa se ne deduce?

2. Si scriva una funzione Matlab che prenda in ingresso il punto di partenza del metodo iterativo `x0`, il numero massimo di iterazioni `nmax`, la tolleranza sul test d'arresto `toll`, (test d'arresto: $||\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}||_2 \leq$ `toll or niter > nmax`) la matrice del sistema `A`, il termine noto `b` e il parametro ω e restituisca la soluzione calcolata `x` e il numero di iterazioni effettuate `niter`.
3. Si ponga

$$A = \begin{bmatrix} 4 & -1 & 0 & 0 \\ -1 & 4 & -1 & 0 \\ 0 & -1 & 4 & -1 \\ 0 & 0 & -1 & 4 \end{bmatrix}, \quad b = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}.$$

Si usi il codice del punto precedente per risolvere il sistema $Ax = b$. Si prenda `toll=1e-6`, `nmax=200`, `x0` il vettore nullo. Si testi il codice per i valori $\omega = 0.5$, $\omega = 1$, $\omega = 2$. Quanto vale il vettore soluzione per ogni valore di ω? Quante iterazioni sono necessarie?

SOLUZIONE

La matrice di iterazione del metodo iterativo è

$$B_\omega = (I - \omega D^{-1}E)^{-1}[(1-\omega)I + \omega D^{-1}F]$$

e il termine noto è

$$\mathbf{g}_\omega = \omega(I - \omega D^{-1}E)^{-1}D^{-1}\mathbf{b}$$

Tracciamo il grafico di B_ω per la matrice del punto 3., in funzione di ω

```
>> A=[ 4 -1 0 0; -1 4 -1 0; 0 -1 4 -1; 0 0 -1 4];
>> D=diag(diag(A)); E=-tril(A,-1); F=-triu(A,1);
>> omega=0:0.01:5;
>> for i=1:length(omega),
        B=inv((eye(4)-omega(i)*inv(D)*E))*...
((1-omega(i))*eye(4)+omega(i)*inv(D)*F);
       rho(i)=max(abs(eig(B)));
   end
>> plot(omega,rho)
>> grid
>> hold on
>> plot(omega,ones(size(omega)),'r','LineWidth',2)
```

Il raggio spettrale di B_ω è minore di 1 per $\omega \in (0, 2)$, intervallo di valori per i quali il metodo iterativo sarà convergente (condizione necessaria e sufficiente!). Scriviamo una function che implementi il metodo proposto

```
function [x,niter]=metodoiterativo(A,b,xv,omega,nmax,toll)

D=diag(diag(A));
E=-tril(A,-1); F=-triu(A,1);
B=inv((eye(4)-omega*inv(D)*E))*...
  ((1-omega)*eye(4)+omega*inv(D)*F);
g=inv((eye(4)-omega*inv(D)*E))*omega*inv(D)*b;

err=10;
niter=0;

while(err>toll & niter<nmax)
      x=B*xv+g;
      niter=niter+1;
      err=norm(x-xv);
      xv=x;
end

return
```

Usando tale codice per la risoluzione del sistema $Ax = b$, otteniamo i seguenti risultati

```
>> xv=zeros(4,1);
>> omega=0.5;
[x,niter]=metodoiterativo(A,b,xv,omega,200,1e-6);
niter = 30
x =

    0.2727
    0.0909
    0.0909
    0.2727

>> omega=1;
[x,niter]=metodoiterativo(A,b,xv,omega,200,1e-6);
niter = 9
x =

    0.2727
    0.0909
    0.0909
    0.2727

>> omega=2;
[x,niter]=metodoiterativo(A,b,zeros(4,1),2,200,1e-6);
niter = 200
```

```
x =

   -0.0285
    0.0709
   -0.1408
    0.2006
```

Osserviamo che il metodo converge solo per i primi due valori di ω, mentre come previsto per $\omega = 2$ esso non converge (per esempio, confrontiamo la soluzione calcolata con la soluzione prodotta dal comando Matlab $A\backslash b$. Inoltre, dal grafico del raggio spettrale in funzione di ω osserviamo che $\rho(\omega = 1) < \rho(\omega = 0.5)$, da cui il numero di iterazioni necessarie per $\omega = 1$ è minore del numero di iterazioni necessarie per $\omega = 0.5$.

ESERCIZIO 3 [9 punti]

Si vuole approssimare l'integrale $I_f = \displaystyle\int_{-1}^{1} f(x)\,dx$ utilizzando la seguente formula di quadratura

$$I_\gamma = w_0 f(\gamma_0) + w_1 f(\gamma_1) + w_2 f(\gamma_2),$$

con $w_0 = \dfrac{5}{9}, w_1 = \dfrac{8}{9}, w_2 = \dfrac{5}{9}$ e $\gamma_0 = -\dfrac{1488}{1921}, \gamma_1 = 0, \gamma_2 = \dfrac{1488}{1921}$.

1. Si implementi la formula di quadratura qui proposta in un semplice codice Matlab che prenda in ingresso i nodi di quadratura γ_i, i pesi $w_i, i = 1, 2, 3$ e la funzione f assegnata come inline function e restituisca il valore I_γ
2. Si verifichi sperimentalmente il grado di esattezza di tale formula.

SOLUZIONE

Abbiamo che

$$I0ex = \textstyle\int_{-1}^{1} 1\,dx = 2,$$
$$I1ex = \textstyle\int_{-1}^{1} x\,dx = 0,$$
$$I2ex = \textstyle\int_{-1}^{1} x^2\,dx = \frac{2}{3},$$
$$I3ex = \textstyle\int_{-1}^{1} x^3\,dx = 0,$$
$$I4x = \textstyle\int_{-1}^{1} x^4\,dx = \frac{2}{5},$$
$$I5ex = \textstyle\int_{-1}^{1} x^5\,dx = 0,$$
$$I6ex = \textstyle\int_{-1}^{1} x^5\,dx = \frac{2}{7},$$

Il seguente codice Matlab prende in ingresso il vettore dei pesi, il vettore dei nodi, la funzione e restituisce l'approssimazione del valore dell'integrale

```
function Int=quadformula(w,gamma,fun)

Int=w(1)*fun(gamma(1))+w(2)*fun(gamma(2))+w(3)*fun(gamma(3))

return
```

Verifichiamo il grado di esattezza della formula:

```
>> w=[5/9 8/9 5/9];
>> gamma=[-1488/1921 0 1488/1921];

>> fun=inline('1','x');
>> I0=quadformula(w,gamma,fun)
   I0 = 2

>> fun=inline('x','x');
>> I1=quadformula(w,gamma,fun)
   I1 =0

>> fun=inline('x.^2','x');
>> I2=quadformula(w,gamma,fun)
   I2 =2/3

>> fun=inline('x.^3','x');
>> I3=quadformula(w,gamma,fun)
   I3 =0

>> fun=inline('x.^4','x');
>> I4=quadformula(w,gamma,fun)
   I4 =2/5

>> fun=inline('x.^5','x');
>> I5=quadformula(w,gamma,fun)
   I5=0

>> fun=inline('x.^6','x');
>> I6=quadformula(w,gamma,fun)
   I6 =6/25
```

Poiché abbiamo che $I0 = I0ex$,$I1 = I1ex$,$I2 = I2ex$, $I3 = I3ex$,$I4 = I4ex$,$I5 = I5ex$,$I6 \neq I6ex$, deduciamo che il grado di esattezza della formula è 5, ovvero essa è in grado di integrare esattamente tutti i polinomi fino al grado 5 compreso.