Esame di Calcolo Numerico — 26 giugno 2023

Corso di Laurea in Ingegneria Chimica
Tempo a disposizione: 2 ore. È consentito consultare appunti e testi (cartacei).

Esercizio 1 (15 punti) Vogliamo risolvere utilizzando il metodo di Jacobi un sistema lineare $T\mathbf{x} = \mathbf{b}$, dove

$$T = \begin{bmatrix} 3 & -1 & & & \\ & 3 & -1 & & \\ & & \ddots & \ddots & \\ & & & 3 & -1 \\ & & & & 3 \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{1}$$

cioè la matrice $n \times n$ tale che

$$T_{ij} = \begin{cases} 3 & i = j, \\ -1 & i = j-1, \\ 0 & \text{altrimenti.} \end{cases}$$

1. Scrivere le equazioni che permettono di calcolare gli elementi del vettore al passo $k+1$ del metodo, cioè $\mathbf{x}^{(k+1)} \in \mathbb{R}^n$, a partire dal vettore al passo k, cioè $\mathbf{x}^{(k)} \in \mathbb{R}^n$.

2. Scrivere una `function y = jac_tri(b, x)` che, dati in ingresso il termine noto $\mathbf{b} \in \mathbb{R}^n$ e $\mathbf{x} = \mathbf{x}^{(k)} \in \mathbb{R}^n$, calcola $\mathbf{y} = \mathbf{x}^{(k+1)} \in \mathbb{R}^n$ eseguendo un passo del metodo. La funzione deve applicare direttamente le formule calcolate al punto precedente, senza fare operazioni superflue. Riportare sul foglio il codice della funzione.

3. Eseguendo la funzione più volte, calcolare i vettori $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}$ ottenuti con i primi tre passi del metodo a partire da $\mathbf{b} = \mathbf{e}_{10} \in \mathbb{R}^{10}$ (l'ultimo vettore della base canonica di $\mathbb{R}^{10}$) e $\mathbf{x}^{(0)} = \mathbf{0} \in \mathbb{R}^{10}$. Riportare sul foglio i comandi Matlab dati e i tre vettori ottenuti.

4. Tra i metodi (diretti o iterativi) studiati nel corso, quale vi aspettate che sia il migliore per risolvere un sistema lineare $T\mathbf{x} = \mathbf{b}$ in cui la matrice T è nella forma (1)?

Esercizio 2 (15 punti) Sia data una funzione continua $g : [0, \pi] \to \mathbb{R}$. Vogliamo risolvere usando il metodo dei trapezi il problema ai valori iniziali

$$\begin{cases} \dfrac{dy}{dt} = y(t) + g(t), \\ y(0) = 1/2, \end{cases} \qquad t \in [0, \pi]. \tag{2}$$

1. Scrivere la formula che lega le approssimazioni y_{n+1} e y_n ottenute in due passi successivi del metodo dei trapezi applicato al problema (2). Risolvere in funzione di y_{n+1}, in modo da ottenere una formula esplicita per calcolare questa quantità a partire da y_n.

2. Scrivere una `function [T, Y] = trap_g(g, N)` che, dati in input la funzione g (come *function handle*) e il numero di intervalli N, risolve il problema (2) con il metodo dei trapezi, restituendo i vettori $T, Y \in \mathbb{R}^{N+1}$ che contengono rispettivamente le approssimazioni t_n e y_n calcolate dal metodo ad ogni passo $n = 0, 1, \dots, N$. Riportare sul foglio il codice Matlab della funzione.

3. Eseguire la funzione con $g(t) = -\cos t$, e, per $N \in \{20, 40, 80\}$, riportare l'errore globale di discretizzazione $E_N = \max_{n=1,2,\dots,N} |y(t_n) - y_n|$ tra la soluzione numerica calcolata e quella esatta, che è $y(t) = \frac{1}{2}(\cos t - \sin t)$. Cosa indicano i risultati ottenuti sull'ordine di convergenza del metodo?

4. (*) Supponiamo di applicare il metodo dei trapezi come sopra per una certa funzione g, e ottenere un errore globale $E_{1000} = 4 \cdot 10^{-6}$. Basandosi sull'ordine di convergenza del metodo, quale valore di N ci aspettiamo di dover usare se vogliamo ottenere $E_N \approx 1 \cdot 10^{-8}$?

Soluzioni

Esercizio 1 (15 punti)

1. Nella scrittura $A = M - N$ che definisce il metodo di Jacobi, notiamo che $M = 3I$, quindi $M^{-1} = \frac{1}{3}I$. Scrivendo esplicitamente un'iterazione del metodo abbiamo

$$
\mathbf{x}^{(k+1)} = \begin{bmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_{n-2}^{(k+1)} \\ x_{n-1}^{(k+1)} \\ x_n^{(k+1)} \end{bmatrix} = M^{-1}(\mathbf{b} + N\mathbf{x}^{(k)}) = \frac{1}{3}I \left(\begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n-2} \\ b_{n-1} \\ b_n \end{bmatrix} + \begin{bmatrix} 0 & 1 & & & & \\ & 0 & 1 & & & \\ & & \ddots & 1 & & \\ & & & 0 & 1 & \\ & & & & 0 & 1 \\ & & & & & 0 \end{bmatrix} \begin{bmatrix} x_1^{(k)} \\ x_2^{(k)} \\ \vdots \\ x_{n-2}^{(k)} \\ x_{n-1}^{(k)} \\ x_n^{(k)} \end{bmatrix} \right)
$$

Confrontando le righe, otteniamo le equazioni

$$
\begin{aligned}
x_n^{(k+1)} &= \frac{1}{3} b_n, \\
x_i^{(k+1)} &= \frac{1}{3}\left(b_i + x_{i+1}^{(k)} \right), \quad i = n-1, n-2, \ldots, 2, 1.
\end{aligned}
$$

2. Una soluzione possibile è la seguente.

```
function y = jac_tri(b, x)
n = length(b);
if not(length(x)==n)
    error('I␣vettori␣devono␣avere␣la␣stessa␣dimensione');
end
y = zeros(n, 1);
y(n) = 1/3 * b(n);
for i = n-1:-1:1
    y(i) = 1/3 * (b(i) + x(i+1));
end
```

3. I risultati ottenuti sono i seguenti.

```
>> n = 10;
>> b = zeros(n, 1); b(end) = 1;
>> x0 = zeros(n, 1);
>> x1 = jac_tri(b, x0)
x1 =
         0
         0
         0
         0
         0
         0
         0
         0
         0
    0.3333
>> x2 = jac_tri(b, x1)
x2 =
         0
         0
```

```
          0
          0
          0
          0
          0
          0
     0.1111
     0.3333
>> x3 = jac_tri(b, x2)
x3 =
          0
          0
          0
          0
          0
          0
          0
     0.0370
     0.1111
     0.3333
```

4. La matrice T è triangolare superiore, e questo suggerisce che l'algoritmo più adatto sia la sostituzione all'indietro. Effettivamente si può vedere che sfruttare la presenza degli zeri sopra la diagonale permette di ridurre il costo dell'intera soluzione a $O(n)$, e questo rende l'algoritmo particolarmente efficiente. La sostituzione all'indietro è quindi l'algoritmo migliore per questo problema, tra quelli visti.

   (In una delle esercitazioni in laboratorio abbiamo visto una versione della sostituzione all'indietro specializzata a matrici bidiagonali superiori, che è esattamente il caso che abbiamo qui.)

   Discutiamo brevemente anche gli altri algoritmi visti. La fattorizzazione LU / eliminazione di Gauss non modifica la matrice T, visto che essa è già triangolare superiore, quindi non ha molto senso considerarla: il costo della soluzione di $T\mathbf{x} = \mathbf{b}$ è solo quello della sostituzione all'indietro, e tutti i passaggi precedenti non hanno effetto. La fattorizzazione LDL e quella di Cholesky non si possono applicare, visto che la matrice non è simmetrica.

   Come visto in questa implementazione, il metodo di Jacobi effettua $O(n)$ operazioni per passo, praticamente facendo in ogni passo lo stesso numero di operazioni della sostituzione all'indietro. Quindi è meno efficiente, visto che tipicamente la convergenza richiede più di un passo. Visto che la matrice T è triangolare superiore, il metodo di Gauss–Seidel coincide con quello di Jacobi, e quindi si applicano le stesse considerazioni.

Esercizio 2 (15 punti)

1. La formula che definisce il metodo è

$$y_{n+1} = y_n + \frac{h}{2}(f_n + f_{n+1}) = y_n + \frac{h}{2}\left(y_n + g(t_n) + y_{n+1} + g(t_{n+1})\right).$$

   Risolvendo in funzione di y_{n+1} otteniamo

$$y_{n+1} = \frac{y_n + \frac{h}{2}\left(y_n + g(t_n) + g(t_{n+1})\right)}{1 - \frac{h}{2}}.$$

2. Una soluzione possibile è la seguente.

```
function [T, Y] = trap_g(g, N)
t0 = 0;
tf = pi;
```

```
h = (tf-t0) / N;
T = t0:h:tf;
Y = zeros(1, N+1);
Y(1) = 1/2;
for n = 1:N
    Y(n+1) = (1-h/2) \ (Y(n) + h/2*(Y(n) + g(T(n)) + g(T(n+1))));
end
```

3. Utilizzando la funzione scritta al punto precedente otteniamo

```
>> g = @(t) -cos(t);
>> [T, Y] = trap_g(g, 20);
>> Y_esatta = 1/2 * (cos(T) - sin(T));
>> E20 = norm(Y - Y_esatta, inf)
E20 =
    0.0250
>> [T, Y] = trap_g(g, 40);
>> Y_esatta = 1/2 * (cos(T) - sin(T));
>> norm(Y - Y_esatta, inf)
ans =
    0.0062
>> E40 = norm(Y - Y_esatta, inf)
E40 =
    0.0062
>> [T, Y] = trap_g(g, 80);
>> Y_esatta = 1/2 * (cos(T) - sin(T));
>> E80 = norm(Y - Y_esatta, inf)
E80 =
    0.0016
>> E20 / E40, E40 / E80
ans =
    4.0230
ans =
    4.0057
```

L'errore si riduce di circa un fattore 4 quando il numero di passi raddoppia, il che indica un ordine di convergenza 2. Questo corrisponde all'ordine noto dalla teoria per il metodo dei trapezi.

4. (*) Poiché il metodo dei trapezi ha ordine di convergenza 2, possiamo aspettarci che l'errore E_N decresca come Ch^2, per una costante C, al crescere del numero di passi. Imponendo che

$$4 \cdot 10^{-6} = C \left(\frac{\pi}{1000}\right)^2$$

otteniamo

$$C = \frac{4}{\pi^2}.$$

Quindi per ottenere

$$1 \cdot 10^{-8} = C \left(\frac{\pi}{N}\right)^2$$

dobbiamo prendere

$$N^2 = \frac{C\pi^2}{10^{-8}} = 4 \cdot 10^8,$$

cioè $N = 2 \cdot 10^4$.

In alternativa, possiamo ottenere lo stesso risultato ragionando in questo modo: far diminuire l'errore da $4 \cdot 10^{-6}$ a 10^{-8} significa ridurlo di un fattore 400; visto che l'errore è inversamente proporzionale al quadrato di N, questo vuol dire che dobbiamo moltiplicare $N = 1000$ per un fattore 20.