4. METODI ITERATIVI

Esercizio 4.1

Considerare la matrice A di dimensione 20x20 che ha tutti 5 sulla diagonale principale, tutti 2 sulle diagonali che fanno da contorno a quella principale, e per il resto tutti 0:

$$A = \begin{bmatrix} 5 & 2 & 0 & \dots & \dots & 0 \\ 2 & 5 & 2 & 0 & \dots & 0 \\ 0 & 2 & \ddots & \ddots & 0 & \vdots \\ \vdots & 0 & \ddots & \ddots & 2 & 0 \\ 0 & \dots & 0 & 2 & 5 & 2 \\ 0 & \dots & \dots & 0 & 2 & 5 \end{bmatrix} \qquad \textit{(dim. 20x20)} \qquad \textit{[4.1]}$$

Considerare inoltre il vettore colonna b di tutti 1, di altezza 20:

$$b = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \qquad \textit{(dim. 20x1)} \qquad \textit{[4.2]}$$

Si chiede di risolvere il sistema lineare Ax=b con un errore massimo sulla soluzione di 10^{-6} , utilizzando:

1)il metodo di Gauss scalare;

2) un metodo iterativo così fatto:

$$\begin{cases} x_{k+1} = x_k + \alpha_j P^{-1} v_k \\ v_k = b - A x_k \\ \alpha_j = 1 \\ P = D = diag(diag(A)) \end{cases} \qquad \textit{[4.3]}$$

---

1) Cominciamo a definire in MATLAB la matrice A. Per farlo, è possibile scrivere innanzitutto un vettore colonna v di altezza 19 che contiene solo il valore 2, e poi definire A come una matrice identità che ha tutti 5 sulla diagonale principale e il vettore v opportunamente inserito nelle diagonali di contorno alla principale:

```
>> v=2*ones(19,1);
>> A=5*eye(20)+diag(v,+1)+diag(v,-1);
```

Inoltre definiamo b e calcoliamo la soluzione con il metodo tradizionale:

```
>> b=ones(20,1);
>> x=A\b;
>> r=A*x-b
```

2) Ora vediamo come utilizzare il metodo iterativo. Un metodo di questo tipo ci restituisce una soluzione x* tale per cui vale il sistema lineare Ax*=b. In generale x* è diversa dalla x vera, quello che a noi interessa è trovare una x* plausibile e il più possibile vicina a quella reale. Dobbiamo dunque introdurre due nozioni fondamentali che ci serviranno a definire questa "vicinanza": quella di *consistenza* e quella di *convergenza*.

Def-4.1: Un metodo iterativo si dice *consistente* quando la soluzione del sistema lineare è punto di convergenza del metodo iterativo (ovvero lo soddisfa). Cioè:

$$x^* = x^* + \alpha_j P^{-1}(b - Ax^*) \qquad [4.4]$$

Def-4.2: Un metodo iterativo si dice *convergente* se l'errore rispetto alla soluzione all'infinito si riduce a zero, cioè:

$$\lim_{x\to+\infty} \|x^* - x_k\| = 0 \qquad [4.5]$$

dove $\|x^* - x_k\| = e_k$ è l'errore.

Verifichiamo innanzitutto la consistenza di [4.3] utilizzando MATLAB. Riscriviamo per comodità il nostro sistema come:

$$\begin{aligned} & x = Ix + D^{-1}(b - Ax) = (I - D^{-1}A)x + D^{-1}b \\ & \rightarrow \cancel{Ix - Ix} + D^{-1}Ax = D^{-1}b \\ & \rightarrow \cancel{D^{-1}}Ax = \cancel{D^{-1}}b \\ & \rightarrow Ax = b \end{aligned} \qquad [4.6]$$

Dopo gli opportuni conti, abbiamo ritrovato il classico sistema lineare Ax=b. Dunque il nostro metodo giunge alla medesima soluzione del sistema tradizionale, e dunque è consistente.

Dimostrare la convergenza di [4.3] è invece leggermente più complesso. Riscriviamo il nostro metodo iterativo come segue:

$$x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b \qquad [4.7]$$

Chiamiamo la porzione evidenziata nella [4.7] "matrice di iterazione", ed indichiamola con B.
Come osservato nella Def-4.2, per verificare la convergenza dobbiamo valutare come varia l'errore e_k.
Ricordando che $\|x^* - x_k\| = e_k$ *s*criviamo che:

$$\begin{cases} x_{k+1} = Bx_k + D^{-1}b \\ x^* = Bx^* + D^{-1}b \end{cases} \qquad [4.8]$$

e sottraiamo la seconda dalla prima, giungendo a:

$$e_{k+1} = Be_k \qquad [4.9]$$

Detto $e_0 = x^* - x_0$, possiamo scrivere:

$$\begin{aligned} e_1 &= Be_0 \\ e_2 &= Be_1 = B^2e_0 \\ &\vdots \\ e_k &= B^k e_0 \end{aligned} \qquad [4.10]$$

La [4.10] ci dice che l'errore e_k dipende esclusivamente dalle caratteristiche della matrice B. Quindi anche la convergenza dipenderà da B. Introduciamo il seguente:

Teorema-4.1: Un metodo iterativo è convergente se e solo se il raggio spettrale della matrice di iterazione $\rho(B)$ è minore di 1.

Il raggio spettrale $\rho(B)$ rappresenta l'autovalore di modulo massimo di B. Verifichiamo dunque in MATLAB se tale autovalore è davvero minore di 1.

```
>> D=diag(diag(A));
>> B=eye(20)-D\A;
>> max(abs(eig(B)))                    eig() calcola gli autovalori

ans =

    0.7911
```

Il valore calcolato è minore di 1, quindi abbiamo verificato anche la convergenza di [4.3].

Un criterio di convergenza esclusivamente sufficiente è anche che $\|B\| < 1$. In MATLAB, la norma può essere calcolata mediante la funzione *norm(matrice,tipo-di-norma)*, dove il *tipo-di-norma* può valere 1, 2, inf (per la norma infinito) o 'fro' (per la norma di Frobenius). Per esempio, calcolando la norma 1 e la norma infinito per la matrice B, MATLAB ci restituisce due valori identici e minori di 1 che rafforzano ulteriormente la nostra ipotesi di convergenza:

```
>> norm(B,1)

ans =

    0.8000

>> norm(B,inf)

ans =

     0.8000
```

Metodo di Jacobi

È interessante ora osservare come il metodo iterativo proposto sia né più né meno che il metodo di Jacobi per la risoluzione dei sistemi lineari. Decomponiamo la matrice A in somma di una matrice triangolare inferiore, una matrice diagonale, ed una matrice triangolare superiore:

$$A=L+D+U \qquad [4.11]$$

Con tale decomposizione, possiamo scrivere:

$$\begin{gathered} Dx_{k+1} = (-L-U)x_k + b \\ \rightarrow x_{k+1} = D^{-1}(-L-U)x_k + D^{-1}b \\ \rightarrow x_{k+1} = D^{-1}(D-A)x_k + D^{-1}b \\ \rightarrow x_{k+1} = (I - D^{-1}A)x_k + D^{-1}b \end{gathered} \qquad [4.12]$$

Cioè, dopo opportune semplificazioni abbiamo ritrovato il nostro metodo iterativo. Dunque [4.3] non è che una scrittura alternativa del metodo di Jacobi; metodo che dunque potrebbe essere felicemente applicato al nostro problema per giungere più agevolmente alla soluzione.
Come si osserva nell'ultimo passaggio della [4.12], in questo caso la nostra matrice di iterazione B è costituita da $(I - D^{-1}A) = B$. Utilizziamo allora MATLAB per definire questa nuova matrice:

```
>> L=tril(A,-1);
>> U=triu(A,+1);
>> B=D\(-L-U);
```

Se A è fortemente diagonalizzata, la convergenza del metodo di Jacobi è automaticamente dimostrata.
In MATLAB è già implementata una routine, denominata *jacob*, che permette di calcolare la soluzione del problema avendo anche impostato una tolleranza sull'errore. Tale routine ha una sintassi del tipo:

```
[xv,iter]=jacob(A,b,x0,nmax,toll)
```

dove:
- A e b sono la matrice ed il vettore del sistema lineare Ax=b impostato;
- `x0` è la matrice iniziale;
- `nmax` è il numero che definisce dopo quante iterazioni deve essere arrestato il metodo (si noti che in realtà una soluzione potrebbe essere trovata già con un numero inferiore di iterazioni, ma in ogni caso nella soluzione MATLAB indicherà chiaramente il numero di iterazioni compiute);
- `toll` rappresenta la tolleranza sull'errore.

Il valore `xv` che viene restituito è la soluzione calcolata con la tolleranza richiesta. `iter` è l'indicazione delle iterazioni compiute dal software per giungere a quella soluzione. Un errore possibile è che il valore `nmax` sia troppo basso per giungere ad una soluzione che rispetti la tolleranza impostata; in questo caso è sufficiente aumentare `nmax`, se il problema lo consente.

Scegliendo un $x_0 = rand(20,1)$ e inserendo `nmax`=100 e `toll`=1e-6, MATLAB giunge alla soluzione in 64 iterazioni:

```
>> [xv,iter]=jacob(A,b,rand(20,1),100,1e-6)

xv =

    0.1667
    0.0833
    0.1250
    0.1042
    0.1146
    0.1094
    0.1120
    0.1107
    0.1113
    0.1111
    0.1111
    0.1113
    0.1107
    0.1120
    0.1094
    0.1146
    0.1042
    0.1250
    0.0833
    0.1667

iter =

    64
```

Teorema-4.2: Se vale per qualche norma $\|B\| < 1$, allora $\|e_k\| \leq \frac{\|B\|^k}{1-\|B\|}\|x_1 - x_0\|$ [4.13]

Il Teorema-4.2 consente di calcolare l'esponente *k*, semplicemente girando la formula:

$$k \geq \frac{\ln\left(\frac{toll\cdot(1-\|B\|)}{\|x_1-x_0\|}\right)}{\ln(\|B\|)} \qquad [4.14]$$

dove *toll* indica la tolleranza sull'errore e la norma è quella per la quale vale $\|B\| < 1$ (p.es., la norma 1). Per eseguire questo calcolo in MATLAB, ci serve conoscere x_1. Se fissiamo $x_0 = rand(20,1)$ e definiamo $x_1 = Bx_0 + D\backslash b$, allora ricaviamo tutti gli elementi utili per calcolare il limite inferiore per *k*. Impostando come tolleranza il valore 10^{-6} e usando la norma 1, otteniamo circa $k \geq 81$.

Tornando al numero di iterazioni necessarie ad ottenere una soluzione, si osserva come questo dipenda essenzialmente dalla grandezza di $\rho(B)$. In particolare, per $\rho(B)$ molto prossimi allo 0 è richiesto un basso numero di iterazioni, mentre questo valore cresce man mano che si avvicina a $\rho(B) = 1$, che rappresenta il limite di separazione tra la convergenza e la non-convergenza.
Sarà allora utile calcolare il valore di α_j che minimizza il numero di iterazioni, cosa che faremo nel prossimo esercizio, e che non era invece richiesto in questo visto che α assumeva un valore fissato.

Esercizio 4.2

Dato il seguente metodo iterativo:

$$x_{k+1} = x_k + \alpha D^{-1}(b - Ax_k) \qquad [4.15]$$

calcolare $\alpha \in (0,2)$ *che minimizza il numero di iterazioni necessarie per giungere alla soluzione del sistema lineare Ax=b.*

Operando opportunamente sul metodo, giungiamo a scrivere che:

$$x_{k+1} = (I - \alpha D^{-1}A)x_k + \alpha D^{-1}b \qquad [4.16]$$

La parte evidenziata nella [4.16] rappresenta la matrice di iterazione B, che questa volta dipende da α.

Possiamo considerare l'intervallo (0,2) all'interno del quale dovrà cadere α e discretizzarlo. Calcoleremo dunque diversi α_j che ci forniranno altrettanti $B(\alpha_j)$. Poiché il numero di iterazioni è tanto più piccolo quanto più lo è $\rho(B)$, andremo a cercare il $B(\alpha_j)$ che minimizza il raggio spettrale, cioè quello per cui $\rho\big(B(\alpha_j)\big)$ è minimo. Trovare il minimo non è difficile, basterà usare un grafico.

```
>> alfa=linspace(0.1,1.9,20);
>> k=1;
>> for j=alfa
   B=eye(20)-j*(D\A);
   rho(k)=max(abs(eig(B)));
   k=k+1;
   end
>> plot(alfa,rho)
```

Cerco il ρ minimo e l'indice dello stesso:

```
>> [rhomin,ind]=min(rho)

rhomin =

    0.8010

ind =

    10
```

Quindi calcolo il valore di α in corrispondenza di quell'indice:

```
>> alfa(10)

ans =

    0.9526
```

Torniamo al sistema lineare Ax=b. Sappiamo che vale la relazione:

$$\frac{\|\delta x\|}{\|x\|} \leq Cond(A) \cdot \frac{\|\delta b\|}{\|b\|} \qquad [4.17]$$

Un modo per calcolare il condizionamento di A è

$$Cond(A) = \|A\| \cdot \|A^{-1}\| \qquad [4.18]$$

La strada che percorre la [4.18] non è affatto agevole, perché il calcolo dell'inversa di A può essere computazionalmente molto oneroso. Ci interessa allora fissare un metodo che permetta di calcolare il condizionamento di A senza calcolare l'inversa.

La strategia che dovremo adottare passerà attraverso il calcolo di una soluzione affetta da errore:

$$\begin{aligned} A\bar{x} &= b + \delta b \\ \bar{x} &= x + \delta x \end{aligned} \qquad [4.19]$$

Noti i valori δb e δx potremo effettuare una stima del condizionamento come:

$$Cond(A) \geq \frac{\frac{\|\delta x\|}{\|x\|}}{\frac{\|\delta b\|}{\|b\|}} \qquad [4.20]$$

Per effettuare una buona stima, occorre calcolare una giusta perturbazione. Una prima strada percorribile è la seguente:

```
>> X=A\b;
>> B=b+1e-6*rand(20,1);
>> db=1e-6*rand(20,1);
>> B=b+db;
>> X=A\B;
>> dx=X-x;
>> K=(norm(dx)/norm(x))/(norm(db)/norm(b))

K =

    1.7810
```

K fornisce una stima inferiore per il condizionamento. Questo significa che il valore reale deve essere maggiore del *K* calcolato con MATLAB.
Utilizzando la funzione cond(), che calcola il condizionamento con il metodo tradizionale, il valore restituito è in effetti:

```
>> cond(A)

ans =

    8.5723
```

Una seconda strada, che come vedremo porterà ad un valore ancora più vicino a quello reale, è quella che impone di fissare il vettore colonna *x* a priori, riempiendolo con valori il più possibile prossimi ad 1, se non con 1 stesso.
Operando in questo modo e procedendo come nel caso precedente, MATLAB permette di calcolare:

```
>> x=ones(20,1);
>> b=A*x;
>> db=1e-6*rand(20,1);
>> B=b+db;
>> X=A\B;
>> dx=X-x;
>> K=(norm(dx)/norm(x))/(norm(db)/norm(b))

K =

    1.9759
```

Questo nuovo valore di K è prossimo a quello calcolato precedentemente, ma più grande e dunque più prossimo al valore reale. In effetti, questo secondo modo di procedere, usando *x* fissato, è preferibile.