//SCRIPT1:

fissiamo una tolleranza per usarla come criterio di stop. Possiamo usare 2 metodi:

- richiedendo che la differenza in norma di due iterazioni sia minore di una certa tolleranza (scarto tra due iterazioni assoluto)

-la differenza in norma di due iterazioni diviso la norma del vettore al passo xk sia inferiore sempre di una certa tolleranza(scarto relativo)

differenze tra i due scarti: cambia il fatto che, mentre nella prima non c'è indicazione sull'ordine di grandezza dei vettori che stiamo andando a considerare, nella seconda noi stiamo calcolando il rapporto rispetto alla norma dell'approssimazione e stiamo ignorando l'ordine di grandezza.

Fissata la tolleranza, noi andremo a controllare se il numero di cifre tra xk+1 e xk coincide per il numero che abbiamo fissato noi.

Quindi, se abbiamo fissato una tolleranza di 10^-3 allora controllando lo scarto relativo andiamo a vedere se k+1 e k coincidono per 3 cifre significative. Dunque ciò che fisso è il numero di cifre che voglio che coincidano nei 2 vettori. Questo sarà il nostro criterio di stop.

Per evitare casi in cui questo diventa molto piccolo(e richieando di creare instabilità), alla fine riscriviamo portando la norma di xk al numeratore dall'altra parte, quindi il nostro criterio di stop sarà minore di una certa tolleranza per la norma di xk

Questo è il medoto ideale di quando converge, bisogna affiancare un metodo di sicurezza in caso il metodo non converga, per non cadere in un loop infinito. Inseriamo quindi un numero di stop k basato sul numero di iterazioni. Cicliamo quindi con un while fino a che k è minore di un certo kmax che fissiamo noi

Quindi utilizzeremo 2 criteri di stop:

-quello riferito allo scarto di due iterazioni(convergenza)
-quello riferito al numero di iterazioni(non convergenza)

----------SCRIPT- jacobi----------

//creo una function che mi renda in output l'approssimazione che si ottiene applicando il metodo di jacobi per risolvere il sistema lineare Ax=b

Mi serve in input:

-la matrice dei coefficienti (A)
-il vettore dei termini noti (b)
- un vettore di innesco, richiesto da ogni metodo iterativo (x0)
-criterio di stop /tolleranza di convergenza (tau)
-criterio di stop di non convergenza(kmax)

Cosa mi interessa ricevere in output:

-l'approssimazione della soluzione(x)
-il numero di iterazioni fatte per raggiungere l'approssimazione(k)

```
function [x,k] = jacobi(A,b,x0,tau,kmax)
//posso chiamarla jacobi perché non esiste in matlab nessuna function con questo nome
```

//devo costruire le matrici dello splitting additivo D,E,F

```
D = diag(diag(A)); //matrice diagonale che contiene la diagonale di A
E = -tril(A,-1); //matrice che contiene tutto ciò che c'è sopra la diagonale(-1) di A cambiato di segno
F = -triu(A,1); //matrice che contiene tutto ciò che c'è sotto la diagonale(1) di A cambiato di segno
//implementiamo il passo iterativo
//Noi dobbiamo calcolare xk+1=Bxk+f . A noi non serve memorizzare tutte le approssimazioni di tutti i passi che compio, perché xk+1 dipende solo dall' x precedente, e il criterio di stop prende in considerazione solo i vettori xk+1 e xk


B = D\(E+F); //calcolo il valore di B=D^-1*(E+F)
f = D\b; //calcolo il valore di f=D^-1*b
flag = 1;
k = 0;
x_new = x0; //al primo passo avrò un solo vettore

while flag
//primo passo: ho x0 in ingresso, quindi calcolo x_new semplicemente applicando la formula
    x0 = x_new; //ad ogni iterazione inserisco il valore di x_new nella var x0
    x_new = B*x0 + f;
    k = k+1;
//vado a controllare se son verificate le due condizioni di stop
    flag = (norm(x_new - x0) > tau*norm(x0)) && (k<kmax);
end
x = x_new; //assegno x_new alla soluzione approssimata x


----------------------------------------------------

//creo una matrice random e un vettore soluzioni che conosco

A = rand(10);

x = ones(10,1);

b = A*x; //calcolo il vettore di termini noti

tau = 1e-3; //imposto criterio di stop per lo scarto min

kmax = 100; //imposto criterio di stop per i passi massimi

x0 = zeros(10,1);  //inizializzo il vettore x0 con tutti zero

[x,k] = jacobi(A,b,x0,tau,kmax);

k


k =//in questo caso abbiamo ricevuto il massimo delle iterazioni


  100


x
```

```
x = //la soluzione non corrisponde a quella imposta da noi: il metodo non converge!


  1.0e+124 *


   -0.2518
   -0.1012
   -0.2015
   -2.7112
   -0.7640
   -0.4184
   -0.3889
   -0.5760
   -0.6446
   -0.3226
```

//per poter testare un algoritmo di questo tipo, è necessario applicarlo ANCHE in modo che converga.
mi serve quindi una matrice che so che fa convergere il metodo di jacobi(e gauss-seidel)

Condizioni della matrice per jacobi:
-strettamente diagonalmente dominante (somma degli elementi della riga è minore dell'elemento della diagonale)
-definita positiva(autovalori positivi)

----------SCRIPT: test jacobi----------

//questo script crea una matrice strettamente diagonalmente dominante

```
n = 10;
A = rand(n); //creo una matrice random di n elementi
A = A - diag(diag(A)); //prendo la diagonale di A e la tolgo dalla matrice
s = sum(abs(A'));
```
//al posto della diagonale di A metto un vettore che assicuri la dominanza. Lo creo sommando tutti gli elementi di una stessa riga tra loro, e maggiorando la somma.
//sum di default effettua la somma di tutte le colonne, quindi devo trasporre A per poter sommare le righe
//abs(matrice) restituisce una matrice di valori assoluti!
```
s = s*3; //maggioro il vettore per rendere la matrice strettamente diagonalmente dominante
A = A + diag(s);  //con diag(s) creo una matrice con il vettore s come diagonale, e inserisco la diagonale nuova dentro A semplicemente sommando le due matrici
```

//creo il sistema con la soluzione nota:
```
x = ones(n,1);  //soluzione: vettore di tutti uno di dimensioni opportune
b = A*x;  //b viene calcolato in dipendenza di x

x0 = zeros(n,1);  //inizializziamo il passo base come vettore di zeri
```

```
tau = 1e-3; //inizializzo i criteri di stop
kmax = 100;
//chiamo la function jacobi: notare che xj corrisponderà alla x in output della function
[xj,k] = jacobi(A,b,x0,tau,kmax);
err = norm(xj-x)/norm(x);  //calcolo l'errore relativo (approssimazione-valore vero/valore vero)
```

---

clear

clc

test_jacobi

//posso vedere che l'errore va bene, perché il limite era 10^-3, ed è arrivato a 10^-4, e ci ha messo solo 8 iterazioni

err

err =

  1.5242e-04

k

k =

   8

//xj se uso 3 cifre significative va esattamente ad 1

xj

xj =

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

   0.9998

//ora cambio la precisione, portando il criterio di stop da 10^-3 a 10^-5

----------SCRIPT: test jacobi----------

```
n = 10;
A = rand(n);
A = A - diag(diag(A));
s = sum(abs(A'));

s = s*3;
A = A + diag(s);
x = ones(n,1);
b = A*x;

x0 = zeros(n,1
tau = 1e-5;
kmax = 100;
[xj,k] = jacobi(A,b,x0,tau,kmax);
err = norm(xj-x)/norm(x);
```

---

clear

clc

test_jacobi

//l'errore va bene perché sta sotto 10^-5, ho 12 iterazioni e xj con format long è molto vicino ad uno, con il format short è 1

err

err =

  1.8817e-06

k

k =

   12

xj

xj =

   1.0000

   1.0000

   1.0000

   1.0000

   1.0000

   1.0000

   1.0000

   1.0000

1.0000

1.0000

format long

xj

xj =

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

0.999998118323577

//faccio adesso la function di gauss seidel

----------SCRIPT: gs----------

```
function [x,k] = gs(A,b,x0,tau,kmax)
D = diag(diag(A));
E = -tril(A,-1);
F = -triu(A,1);
//in questa function cambia solo la generazione di B e f rispetto a jacobi
B = (D-E)\F;
f = (D-E)\b;
x_new = B*x0 + f;
k = 1;

while (norm(x_new - x0) > tau*norm(x0)) && (k<kmax)
    x0 = x_new;
    x_new = B*x0 + f;
    k = k+1;
end
x = x_new;
```

//creo il test dedicato per l'algoritmo di gauss-seidel

----------SCRIPT: test gs----------

```
n = 10;
A = rand(n);
A = A - diag(diag(A));
s = sum(abs(A'));
```

```
s = s*3;
A = A + diag(s);

x = ones(n,1);
b = A*x;

x0 = zeros(n,1);
tau = 1e-5;
kmax = 100;

[xgs,k] = gs(A,b,x0,tau,kmax);
err = norm(xgs-x)/norm(x);
```

---

clear

clc

test_gs

//noto che il vettore delle soluzioni xgs, calcolato con gauss seidel, è differente da quello precedente calcolato con jacobi. Questo perché la matrice di origine è diversa!
è necessario costruire un test per poter testare insieme i due algoritmi

err

err =

2.063074675103854e-07

k

k =

6

xgs

xgs =

0.999999736276536

0.999999569672641

0.999999788867923

0.999999680078398

0.999999888216049

0.999999965770601

1.000000048013477

1.000000064475666

1.000000053560844

1.000000031160092

----------SCRIPT: test metodi iter----------

```
n = 10;
A = rand(n);
A = A - diag(diag(A));
s = sum(abs(A'));
s = s*3;
A = A + diag(s);

x = ones(n,1);
b = A*x;

x0 = zeros(n,1);
tau = 1e-5;
kmax = 100;
//in questo modo i due algoritmi si basano esattamente sugli stessi dati di partenza
//ho 2 approssimazioni, xj e xgs, e 2 contatori, kj e kgs
[xj,kj] = jacobi(A,b,x0,tau,kmax);
[xgs,kgs] = gs(A,b,x0,tau,kmax);
err_j = norm(xj-x)/norm(x);
err_gs = norm(xgs-x)/norm(x);
```

---

clear

clc

test_metodi_iter

err_gs

err_gs = //errore di gauss-seidel

1.208702855311383e-07

err_j

err_j = //errore di jacobi

1.881676423154399e-06

kj

kj = //iterazioni jacobi

12

kgs

kgs = //iterazioni gauss seidel

6

//l'errore calcolato con GS è inferiore, quindi l'approssimazione è più precisa

//GS per ottenere una miglior precisione impiega circa la metà delle iterazioni di jacobi e questo dipende dal fatto che intrinsecamente gs utilizza informazioni aggiornate del vettore approssimazione

//più si diminuisce il valore di tolleranza, più il divario tra i 2 si fa marcato

//Come assegnare valori di default: uso un comando che va a guardare quanti argomenti in input vengono dati durante la chiamata alla funzione. Quando io chiamo la funzione, se non gli dico nulla, lui si aspetta che io gli dia 5 input. Se non glieli do mi avvisa che il numero di input è insufficiente.
Se vado a calcolare quanti input assegno alla chiamata, posso dire "se il numero di input è 4 e non 5, vuol dire che kmax non mi interessa dartelo, usa il valore di default"
quindi dovrò usare una combinazione tra controllo di numero input e if numero input:

----------SCRIPT- jacobi----------

```
function [x,k] = jacobi(A,b,x0,tau,kmax)
//inserisco dentro la function queste condizioni tramite la funzione nargin che misura il numero di argomenti in input
if (nargin < 5),kmax = 100;end
if (nargin < 4),tau = 1e-5;end
if (nargin < 3),x0 = zeros(size(b));end
//nell'ultimo if ci serve la lunghezza oppertuna di x0, che ha la stessa dimensione di b, che sto inserendo in input. Inserisco quindi la lunghezza di b come riferimento.
//posso scrivere gli if sulla stessa riga, mettendo una virgola dopo le condizioni

D = diag(diag(A));
E = -tril(A,-1);
F = -triu(A,1);

B = D\(E+F);
f = D\b;
flag = 1;//imposto il valore di flag a 1 per entrare all'interno del ciclo while
k = 0;
x_new = x0;

while flag
    x0 = x_new;
    x_new = B*x0 + f;
    k = k+1;
    flag = (norm(x_new - x0) > tau*norm(x0)) && (k<kmax); //imposto un controllo in uscita dal ciclo con una flag, per evitare di ripetere operazioni all'interno del while
end
x = x_new;
```

//ESERCIZIO(CONSIGLIATO CALDAMENTE): FARE LE STESSE MODIFICHE SU GS, E MODIFICARE IL TEST PER ENTRAMBI GLI ALGORITMI IN MODO CHE FACCIA DEI TEST SU MATRICI DI DIMENSIONE CRESCENTE:SI FA UN CICLO FOR CHE MODIFICA n, SI MEMORIZZA IN UN VETTORE DI ERRORI.