# jacobi - Analisi di accuratezza e robustezza

In questo documento sono mostrati i test relativi alla funzione implementata per la risoluzione dei sistemi lineari con matrice dei coefficienti sparsa .L'accuratezza è valutata mediante il calcolo dell'errore relativo creando sistemi **adHoc** di cui si conosce la soluzione reale. La robustezza è valutata effenttuando degli appositi test utilizzando il framework di MatLab per il testing di unità .

# Test di accuratezza

Per testare l'accuratezza dell'algortimo realizzato, si è utilizzata come metrica di valutazione l'errore relativo calcolato come norma della differenza tra la soluzione trovata l'algoritmo ( xRisolve ) e quella reale ( `x` ) normalizzati rispetto la norma della soluzione reale e il residuo relativo calcolato come norma della differenza tra la `b` e il prodotto di `A` per la soluzione trovata , normalizzato rispetto alla norma di `b`.

**Errore relativo :** $\frac{\|\text{xRisolve} - x\|}{\|x\|}$ **Residuo relativo :** $\frac{\|b - A * x\|}{\|b\|}$

**Nota :** Per soluzione reale si intende la soluzione nota a priori del sistema e non quella calcolata con un qualsiasi algoritmo risolutivo.

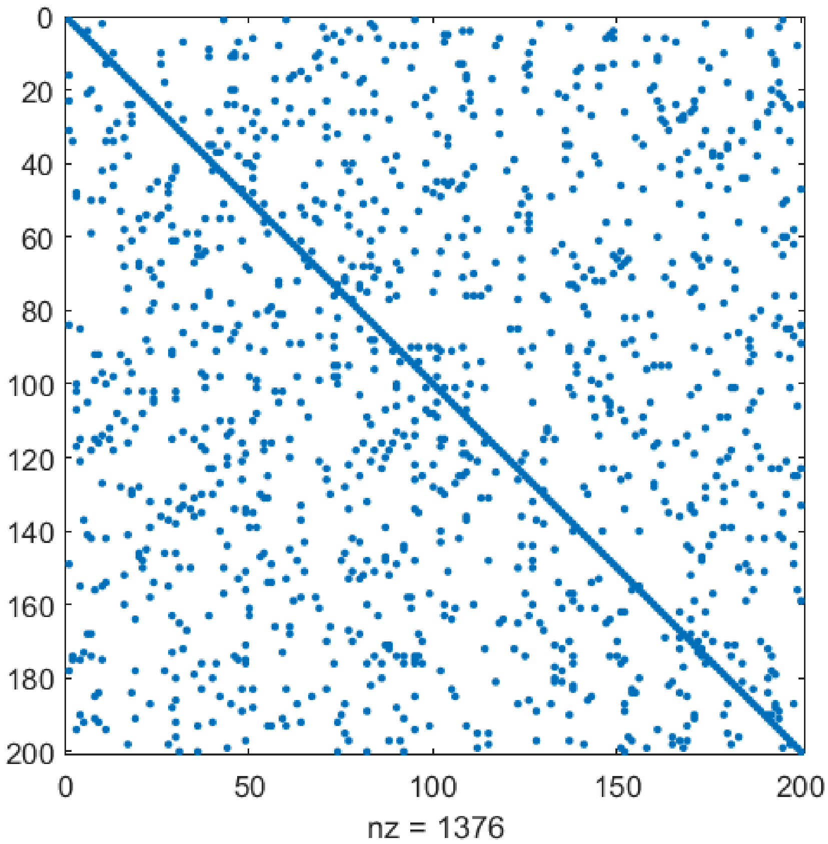
Per effetturare i test si sceglie la matrice `A` ( Matrice dei coefficienti ) e il vettore `x` ( Soluzione reale ) poi si passano questi parametri in input alla funzione jacobi`Accuracy` che calcola il vettore `b` ( Vettore dei coefficienti ) come `A*x` e restituisce una struttura contenete i seguenti campi : **`cond`**, **`erroreRel`** , **`residuoRel`** e **`niter`** . Che contengono rispettivamente i valori dell' l'indice di condizionamento della matrice `A`, l'errore relativo della soluzione ottenuta con `jacobi` , il residuo relativo e il numero di iterazioni eseguite . Si rimanda il lettore alla documentazione delle funzione per maggiori informazioni .

### Test con matrice A random

Test con matrice A avente elementi non nulli sulla diagonale principale.

```
% La matrice generata è ben condizionata
n = 200;
A = sprand(n,n,0.03)+spdiags(4*ones(n,1),0,n,n);
x = ones(n,1);

jacobiAccuracy(A,x,eps,1000)
```

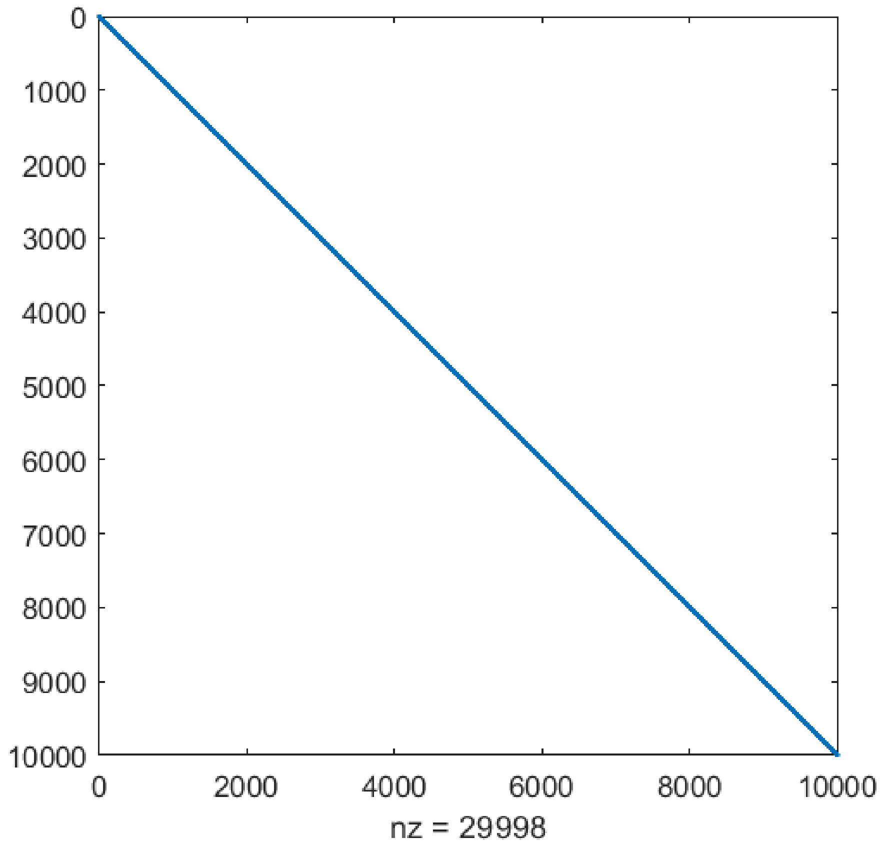


```
ans = struct with fields:

          cond: 9.779263070235286
     erroreRel: 8.881784197001244e-16
    residuoRel: 2.758248405764645e-16
         niter: 131
```

#### Test con matrice tridiagonale

```
n=10000;
A = gallery('tridiag',n,10,31,20);
x=ones(n,1);
% usa eps di default e tol di default
jacobiAccuracy(A,x)
```




```
ans = struct with fields:
          cond: 60.999999999999893
      erroreRel: 4.914566982808700e-07
     residuoRel: 4.914569398143151e-07
         niter: 443
```

### Test con matrice A tale che $\rho(B) < 1$

Test con matrice A tale che il raggio spettrale della matrice di iterazione $\mathrm{Bj} = I - A^{-1}D$ sia minore di 1 ( $\rho(\mathrm{Bj})$ è definito come il massimo autovalore di `Bj` ), Per verificare la condizione richiesta per la convergenza, in questo caso, per come è definita `Bj` è sufficiente che la matrice `A`,in questo caso diagonale, abbia raggio spettrale < 1 ( gli autovalori di $A^{-1}$ coincidono con quelli di A ). La proprietà che il raggio spettrale sia una quantità minore di 1 è una condizione sufficiente per avere la convergenza dell'algoritmo di Jacobi.

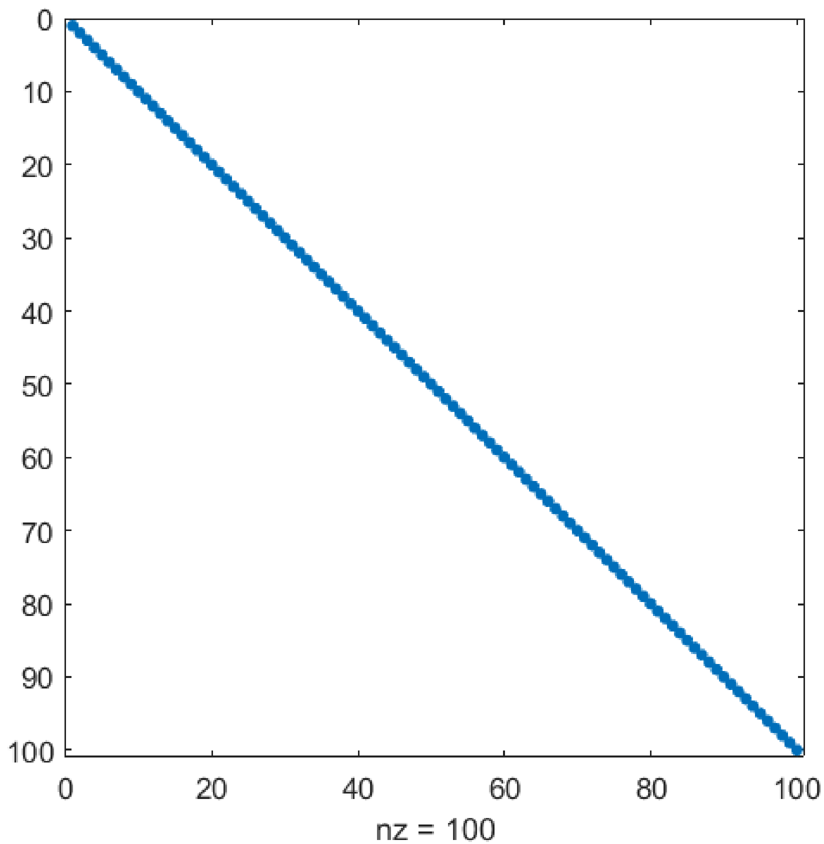
```
% La matrice A è diagonale in modo tale che gli autovalori coincidano con
% gli elementi presenti sulla diagonale principale
n = 100;
A = spdiags(rand(n,1),0,n,n);
x = ones(n,1);
maxAutovaloreA = max(eig(A))
```

```
maxAutovaloreA =
   0.995108548807354
```

```
%Calcolo del raggio di convergenza di Bj
DInv = sparse(1:n,1:n,1./diag(A));
Bj = speye(n,n) - DInv*A;
maxAutovalreBj=max(eig(Bj))
```

```
maxAutovalreBj =
    1.110223024625157e-16
```

```
jacobiAccuracy(A,x,eps,1000)
```




```
ans = struct with fields:
         cond: 1.219427708286496e+03
    erroreRel: 0
   residuoRel: 0
        niter: 2
```

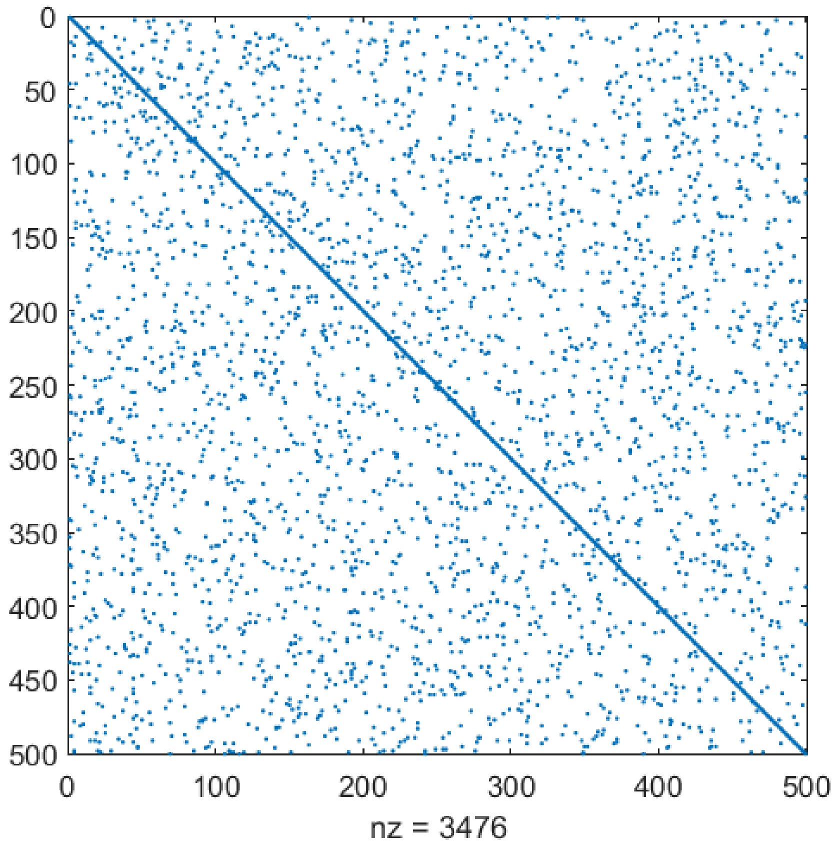
### Test con matrice A a diagonale strettamente dominante

Un ulteriore caso di notevole interesse è quando la matrice  A è a diagonale strettamente dominante, ossia il modulo degli elementi sulla diagonale principale sono più grandi della somma dei moduli degli elementi sulla medesima riga. La condizione di diagonale principale strettamente dominante per `A` implica che il raggio spettrale di `Bj` è una quantità minore di uno ed è quindi una condizione sufficiente per avere la convergenza del metodo di Jacobi.

```
n = 500;
```

```
% Aggiungiamo a tutti gli elementi sulla diagonale principale la quantità
% 16, in modo tale da verificare la condizione di dominanza
A=sprand(n,n,0.012)+spdiags(16*ones(n,1),0,n,n);
x = ones(n,1);

jacobiAccuracy(A,x)
```

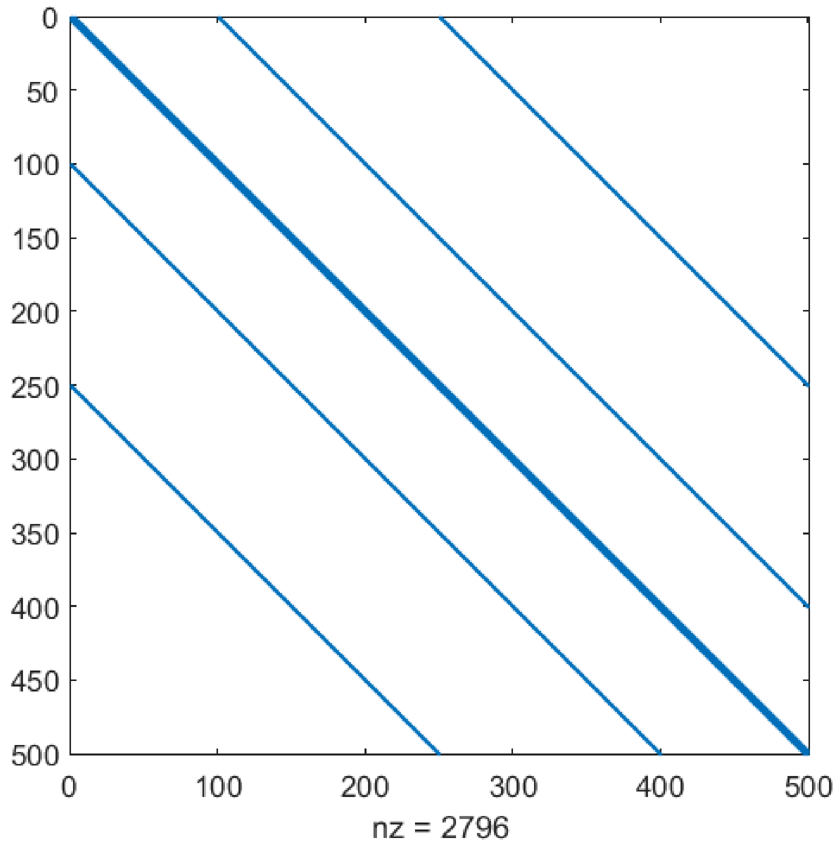



```
ans = struct with fields:

        cond: 2.560530230202991
   erroreRel: 3.174748363328418e-08
  residuoRel: 2.555057897533482e-08
       niter: 11
```

```
n = 500;
onesCol = ones(n,1);
columns = [-onesCol,-2*onesCol, onesCol, 15*onesCol, -onesCol, 2*onesCol,2*onesCol];
d = [-n/2,-n/5, -2, 0, 2, n/5,n/2];
A = spdiags(columns,d,n,n);
x = ones(n,1);

jacobiAccuracy(A,x)
```



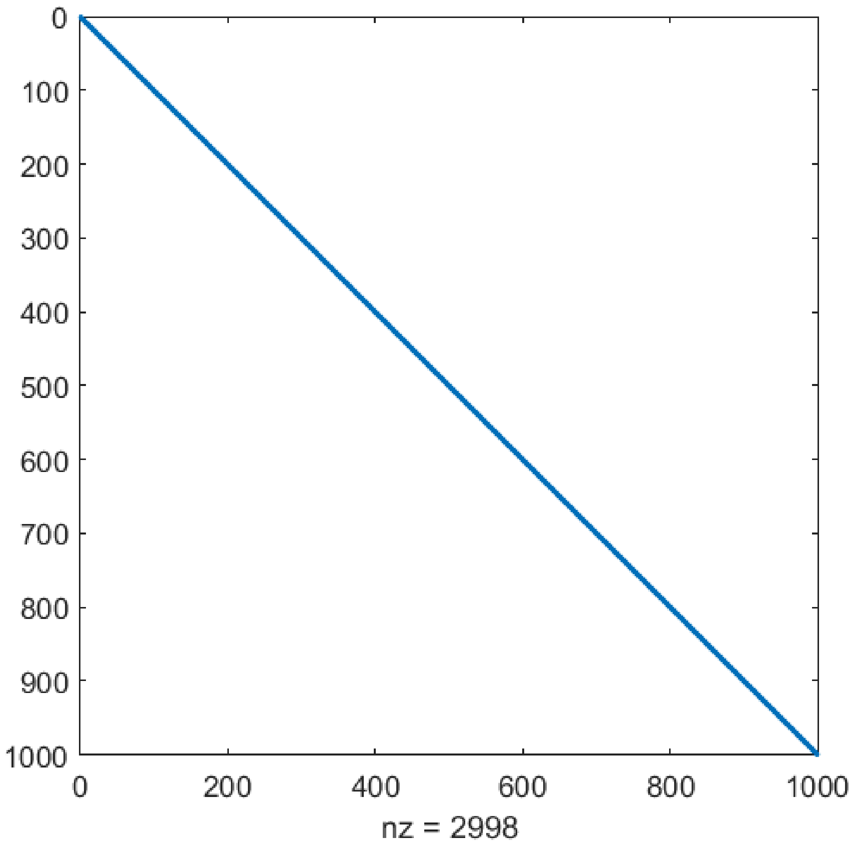

```
ans = struct with fields:

          cond: 2.534910204693912
     erroreRel: 1.645617582290814e-07
    residuoRel: 1.522812210833093e-07
         niter: 15
```

**Test con matrice A tridiagonale con diagonale dominante (dalla gallery)**

```
n = 1000;
A = gallery('tridiag',1000,10,31,20);
x = ones(n,1);

%eps impostato dall'utente
jacobiAccuracy(A,x,1e-4,2000)
```




```
ans = struct with fields:

          cond: 60.999999999999886
     erroreRel: 4.843127632797637e-05
    residuoRel: 4.843362203003932e-05
         niter: 303
```

### Test con matrice A di Poisson

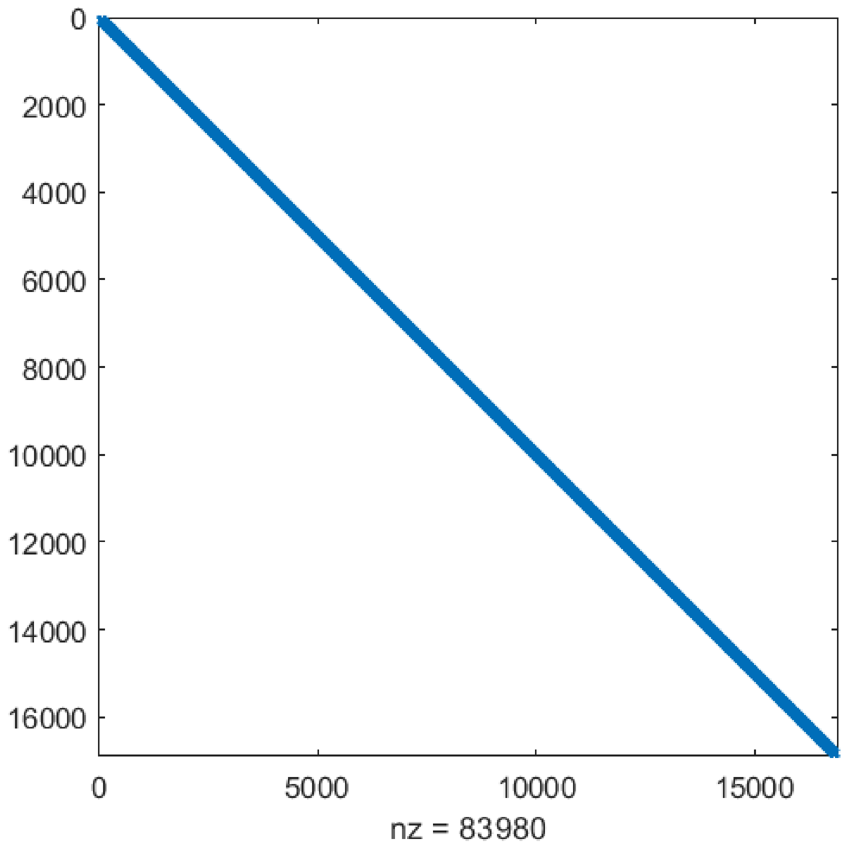
Nel caso in cui la matrice A sia una matrice di Poisson di dimensioni notevoli,anche se l'indice di condizionamento è elevato, si ottiene una soluzione accettabile dal punto di vista dell'errore relativo, ma il numero di iterazioni necessarie all'algoritmo per ottenere la soluzione è elevato, ciò si ripercuote sul tempo necessario per ottenere la soluzione.

```
n = 130;
A = gallery('poisson',n);
x = ones(n^2,1);

% L'algoritmo non riesce a converge utilizzando il numero di MAXITER di
% default

b=A*x;
jacobiAccuracy(A,x)
```

```
Warning: Il numero di iterazioni effettuate non è sufficiente per raggiungere l'accuratezza desiderata.
NITER=500, RESIDUO RELATIVO=1.438791e-03
```




```
ans = struct with fields:
          cond: 1.011272844784519e+04
     erroreRel: 1.002408484469718
    residuoRel: 0.001438790929649
         niter: 500
```

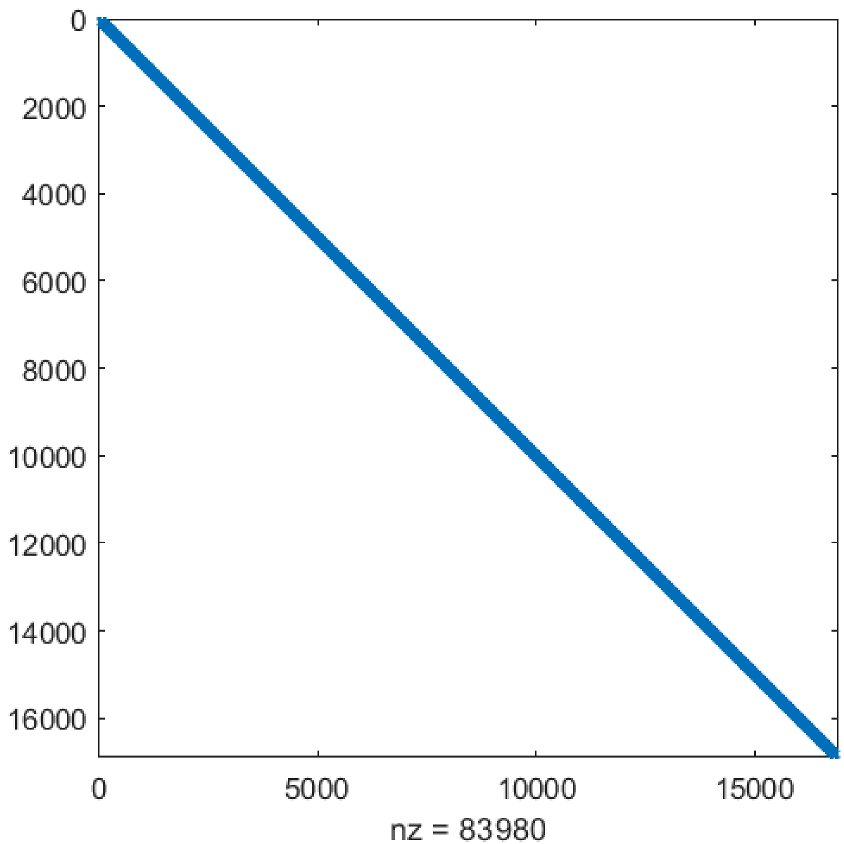
Notiamo che il numero di iterazioni non è necessario per avere la soluzione, quindi lo aumentiamo.

```
n = 130;
A = gallery('poisson',n);
x = ones(n^2,1);

% L'algoritmo non riesce a converge utilizzando il numero di MAXITER
% inserito dall'utente

b=A*x;
jacobiAccuracy(A,x,1e-7,10000)
```

```
Warning: Il numero di iterazioni effettuate non è sufficiente per raggiungere l'accuratezza desiderata.
NITER=10000, RESIDUO RELATIVO=5.253694e-05
```




```
ans = struct with fields:
         cond: 1.011272844784519e+04
    erroreRel: 0.091361436628743
   residuoRel: 5.253694376478046e-05
        niter: 10000
```

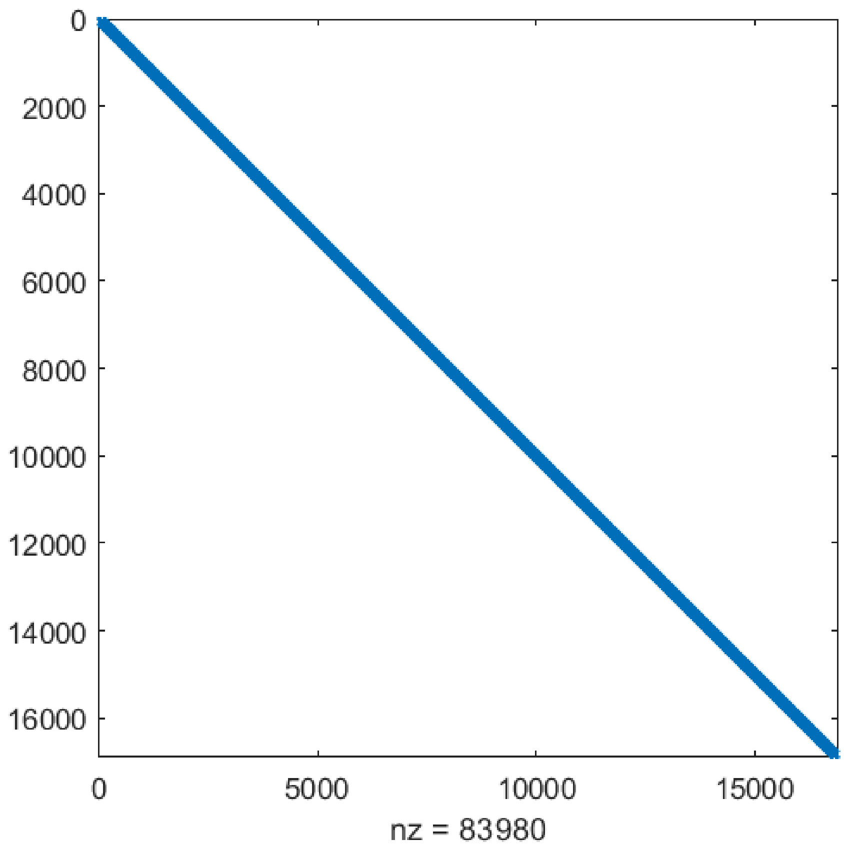
Anche in questo caso il numero di iterazioni non è sufficiente per ottenere l'uscita per effetto del criterio di arresto basato sulla riduzione dell'errore tra le soluzioni del passo corrente e quelle ottenute al passo precedente.

```
n = 130;
A = gallery('poisson',n);
x = ones(n^2,1);

% L'algoritmo riesce a converge utilizzando un numero di MAXITER molto
% grande

b=A*x;
jacobiAccuracy(A,x,1e-7,30000)
```

```
Warning: Il numero di iterazioni specificato è molto alto, l'esecuzione potrebbe essere più lenta
```




```
ans = struct with fields:

         cond: 1.011272844784519e+04
    erroreRel: 3.476221275706391e-04
   residuoRel: 1.999142105746365e-07
        niter: 29373
```

**Nota**: il numero di cifre significative corrette nella soluzione è 4, a fronte delle 7 richieste tramite il parametro TOL. Ciò è dovuto al condizionamento della matrice dei coefficienti.

# Test di robustezza

Per la progettazione dei casi di test è stato utilizzato il metodo di Category-partition testing [1], ampiamente diffuso nell'ambito dell'Ingegneria del Software per il testing funzionale. In particolare, le categorie sono state scelte in base ai parametri (di ingresso ) modificabili dall'utente e in base agli elementi della matrice `A` che potrebbero violare le condizioni di funzionamento della funzione. I valori sono stati scelti in base alle condizioni di errore/warning in modo tale da verificare che in caso di errori in input i warning e gli errori vengono correttamente scatenati .

| Categoria | Valori | Vincolo | Risultato |
| --- | --- | --- | --- |
| A | Matrice quadrata sparsa | Obbligatorio | |
| | Matrice sparsa con NaN o inf | Obbligatorio | ERROR |
| | Matrice di dimensione n*m | Obbligatorio | ERROR |
| | Matrice sparsa con uno 0 sulla diagonale | Obbligatorio | ERROR |
| | Matrice piena | Obbligatorio | ERROR |
| b | Vettore colonna di dimensione n | Obbligatorio | |
| | Vettore colonna di dimensione m | Obbligatorio | ERROR |
| | Vettore colonna con NaN o inf | Obbligatorio | ERROR |
| TOL | Empty | | |
| | 0 < TOL < eps | | |
| | TOL> eps | | |
| | TOL >=1 | | WARNING |
| | TOL < 0 | | WARNING e TOL=eps |
| | !isnumeric(TOL) | | ERROR |
| NMAX | Empty | | |
| | !isnumeric(NMAX) | TOL e x0 non vuoto | ERROR |
| | 10 < NMAX < 10000 | TOL e x0 non vuoto | |
| | 2 < NAMX < 10 | TOL e x0 non vuoto | WARNING |
| | NMAX > 10000 | TOL e x0 non vuoto | WARNING |
| | NMAX < 2 | TOL e x0 non vuoto | ERROR |

Per l'implementazione dei casi di test è stato utilizzato il framework di MatLab per il testing di unità [2]; questa scelta ha permesso di automatizzare completamente l'esecuzione dei test: infatti, è possibile lanciare la simulazione con l'istruzione:

```
results = runtests('testSuite.m')
```

```
Running testSuite
.........
Done testSuite
__________

results =
  1×9 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   9 Passed, 0 Failed, 0 Incomplete.
   1.1547 seconds testing time.
```

Il sistema esegue in automatico i test, riportando in `results` un sommario di quanti hanno avuto successo e dettagliando gli errori in quelli che eventualmente sono falliti.

> **Note**
>
> Non sono stati implementati test per verificare il comportamento della funzione nel caso in cui il numero di parametri in input e di output non siano validi. In quanto MATLAB esegue in automatico questo controllo. Infatti nel caso in cui il numero di parametri in inpu o di output sia maggiore di quelli specificati, viene generato un errore.

**Nota:** Per semplicità non sono stati implementati i test relativi a dati di input che contenessero valori invalidi `NaN/Inf`.

Di seguito viene descritta l'implementazione di alcuni casi di test a titolo esemplificativo.

## Caso di test 2

- A : **zero sulla diagonale**
- b : **valido**
- TOL : **indifferente**
- MAXITER : **indifferente**

```
function testFunctionCase2(testCase)

    A = sprand(15,15,0.1);
    A(1,1) = 0;
    x= ones(15,1);
    b = A*x;

    testCase.verifyError(@()jacobi(A,b),'jacobi:AZeroOnDiag')
end
```

L'algoritmo non verrà eseguito perchè la matrice `A` presenta uno zero sulla diagonale principale. Tale condizione viola le ipotesi di funzionamento dell'algoritmo, quindi si impedisce l'esecuzione dell'algoritmo e viene mostrato un messaggio di errore.

## Caso di test 4

La configurazione dei parametri è:

- A : **valido**
- b : **valido**
- TOL : **TOL > 1**
- MAXITER : **indifferente**

```
function testFunctionCase4(testCase)

    A = sprand(10,10,0.1) + speye(10,10);
    x= ones(10,1);
    b = A*x;
    TOL = 10;
    testCase.verifyWarning(@()jacobi(A,b,TOL),'jacobi:TOLtooHigh')
```

```
end
```

Viene calcolata la soluzione, ma viene mostrato un warning perchè il valore di `TOL` è troppo grande e quindi la soluzione sarà poco accurata.

### Caso di Test 7

- A : **valido**
- b : **non valido**
- TOL : **indifferente**
- MAXITER : **indifferente**

```
function testFunctionCase7(testCase)

    A = sprand(10,10,0.1) + speye(10,10) ;
    b = rand(10);

    testCase.verifyError(@()jacobi(A,b),'jacobi:ColInvalidDimension')
end
```

Essendo `b` una matrice e non un vettore colonna, la funzione non verrà eseguita e verrà mostrato un messaggio di errore.

## Nota underflow

Inoltre è interessante notare come l'algoritmo implementeato si comporti nel caso in cui dovesse verificarsi una condizione di ***underflow*** nel sistema. In particolare, tale condizione si potrebbe verificare nel calcolo del prodotto tra `TOL` e `x`, infatti poichè `x` è calcolato mediate passi successivi e `TOL` è un valore piccolo, potrebbe verificarsi la condizione di ***underflow*** , cioè il prodotto `TOL*x` verrebbe sostituito da 0. Se ciò accedesse l'algoritmo non convergerebbe mai. La funzione implementata è robusto rispetto al verificarsi di quest'errore in quanto è stato implementato un controllo sul prodotto, che nel caso si verifichi la condizione di ***underflow*** sostituisce `TOL*x` con `realmin`.

```
function TOLX = checkUnderflowTOLX(TOL,x)
    TOLX = TOL*norm(x,inf);
    if TOLX < realmin
        TOLX = realmin;
    end
end
```

### Riferimenti

[1] *Robert D. Cameron, 2013,* http://www.cs.sfu.ca/~cameron/Teaching/473/category-partition.html

[2] Matlab Documentation, https://it.mathworks.com/help/matlab/function-based-unit-tests.html

### Autore