jacobi - Analisi di robustezza, accuratezza e prestazioni

In questo documento sono mostrati i test relativi all'algoritmo per la risoluzione dei sistemi lineari, con matrice dei coefficienti sparsa, implementato .L'accuratezza è valutata mediante il calcolo dell'errore relativo creando sistemi **adHoc** di cui si conosce la soluzione reale.

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 , normalizzati rispetto alla norma di `b`.

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

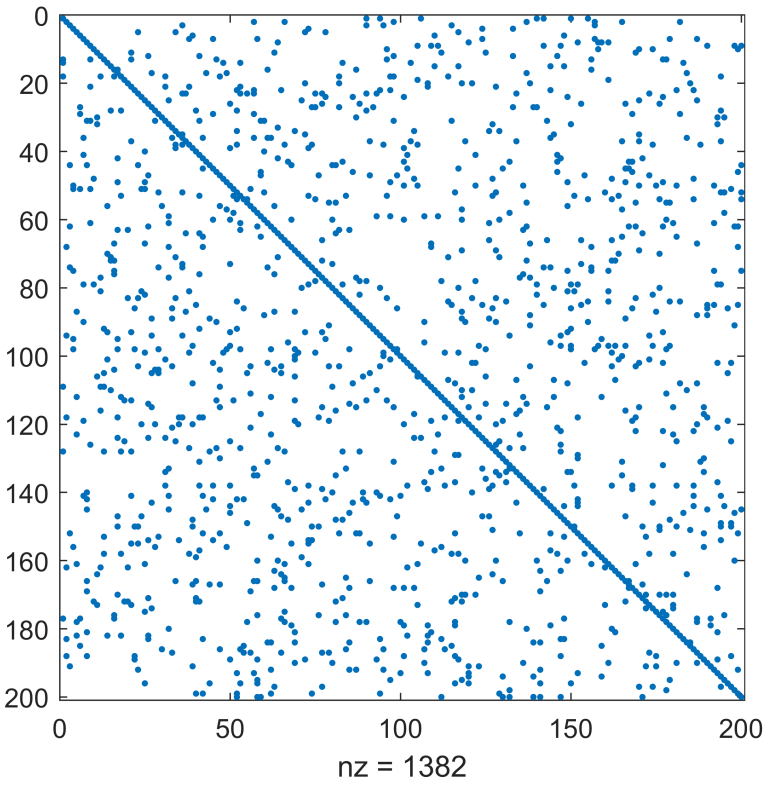
Nota : Per soluzione reale si intende la soluzione nota a priori del sistema e non quella calcolata con un qualsiasi algoritmo risolutivo.

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 si 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 A è diagonale in modo che gli autovalori coincidano con
% gli elementi della diagonale principale
n = 200;
A = sprand(n,n,0.03)+spdiags(4*ones(n,1),0,n,n);
x = ones(n,1);

jacobiAccuracy(A,x,eps,zeros(n,1),1000)
```




```
ans = struct with fields:
          cond: 9.3467
     erroreRel: 6.6613e-16
    residuoRel: 1.6158e-16
         niter: 131
```

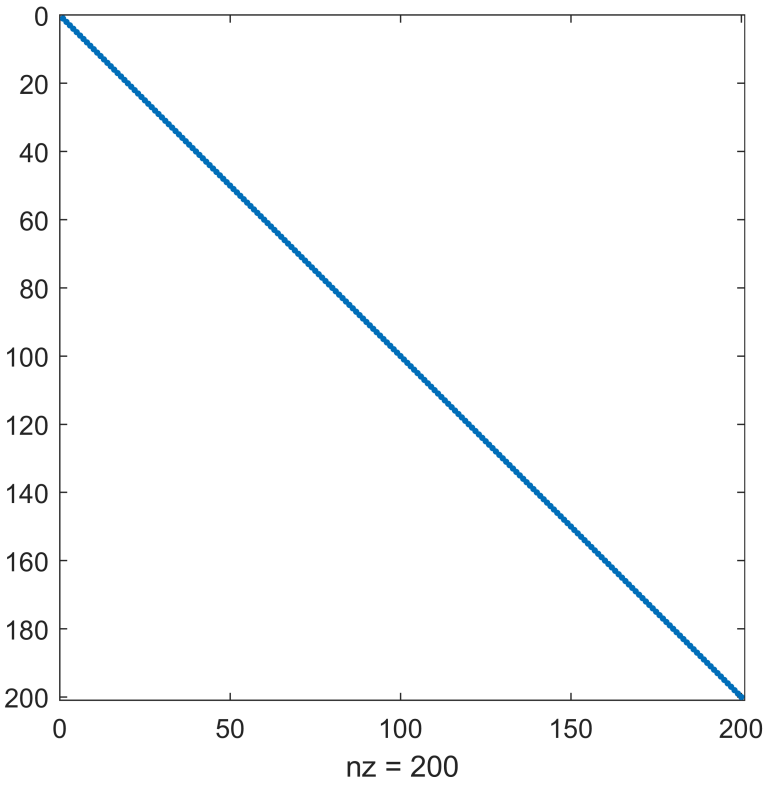
Test con matrice A tale che $\rho(A) < 1$

Test con matrice A tale che il raggio spettrale sia minore di 1 ($\rho(A)$ è definito come il massimo autovalore 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 = 200;
A = spdiags(rand(n,1),0,n,n);
x = ones(n,1);
max_autovalore = max(eig(A))
```

```
max_autovalore = 0.9905
```

```
jacobiAccuracy(A,x,eps,zeros(n,1),1000)
```

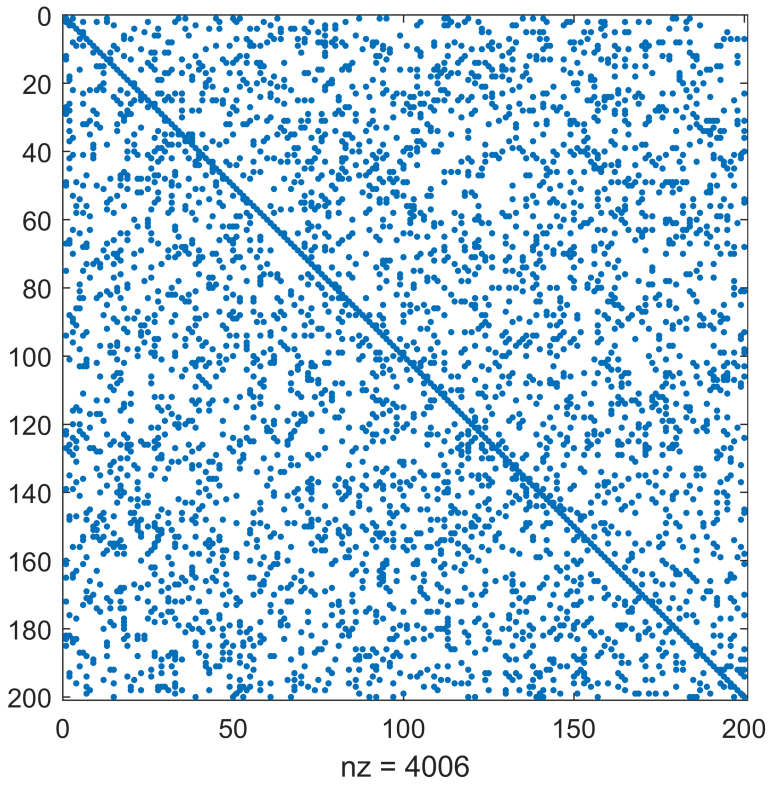



```
ans = struct with fields:
          cond: 200.1160
     erroreRel: 0
    residuoRel: 0
         niter: 2
```

Test con matrice A a diagonale strettamente dominante

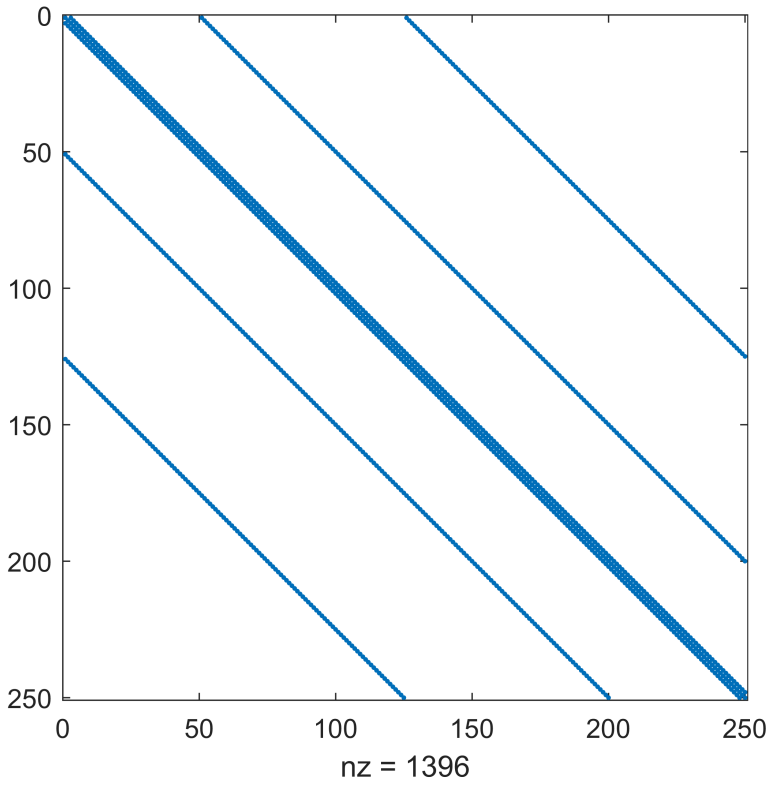
Un ulteriore caso di notevole interesse è quando la matrice A è a diagonale strettamente dominante, ossia il modulo degli elementi sulla diagonale principale sono più grandi della somma dei moduli degli elementi sulla medesima riga. Dato che $\rho(A) \leq \|A\|$, anche la diagonale principale strettamente dominante è una condizione sufficiente per avere la convergenza del metodo di Jacobi

```
n = 200;
% Aggiungiamo a tutti gli elementi sulla diagonale principale la quantità
% 16, in modo tale da verificare la condizione di dominanza
A=sprand(n,n,0.10)+spdiags(16*ones(n,1),0,n,n);
x = ones(n,1);

jacobiAccuracy(A,x)
```




```
ans = struct with fields:
          cond: 5.0475
     erroreRel: 2.5808e-07
    residuoRel: 2.0515e-07
         niter: 29
```

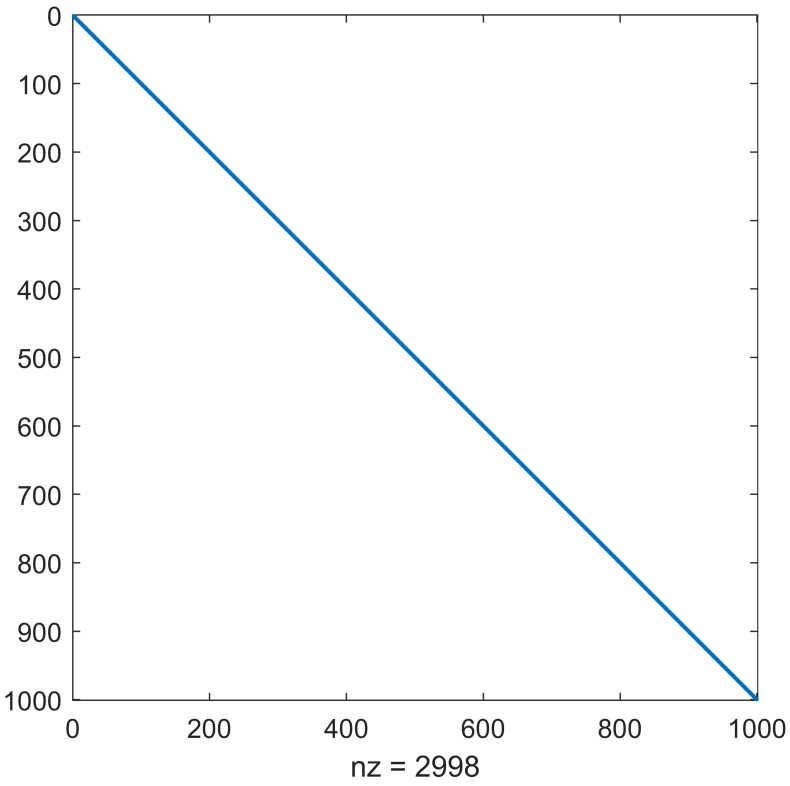
```
n = 250;
onesCol = ones(n,1);
columns = [-onesCol,-2*onesCol, onesCol, 15*onesCol, -onesCol, 2*onesCol,2*onesCol];
d = [-n/2,-n/5, -2, 0, 2, n/5,n/2];
A = spdiags(columns,d,n,n);
x = ones(n,1);

jacobiAccuracy(A,x)
```




```
ans = struct with fields:
          cond: 2.5283
     erroreRel: 1.6456e-07
    residuoRel: 1.5228e-07
         niter: 15
```

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

```
n = 1000;
A = gallery('dorr',1000);
x = ones(n,1);

jacobiAccuracy(A,x,1e-4,zeros(n,1),2000)
```




```
ans = struct with fields:
          cond: 9.2089e+09
     erroreRel: 1.0535
    residuoRel: 1.9397e-04
         niter: 1614
```

Test con matrice A di Poisson

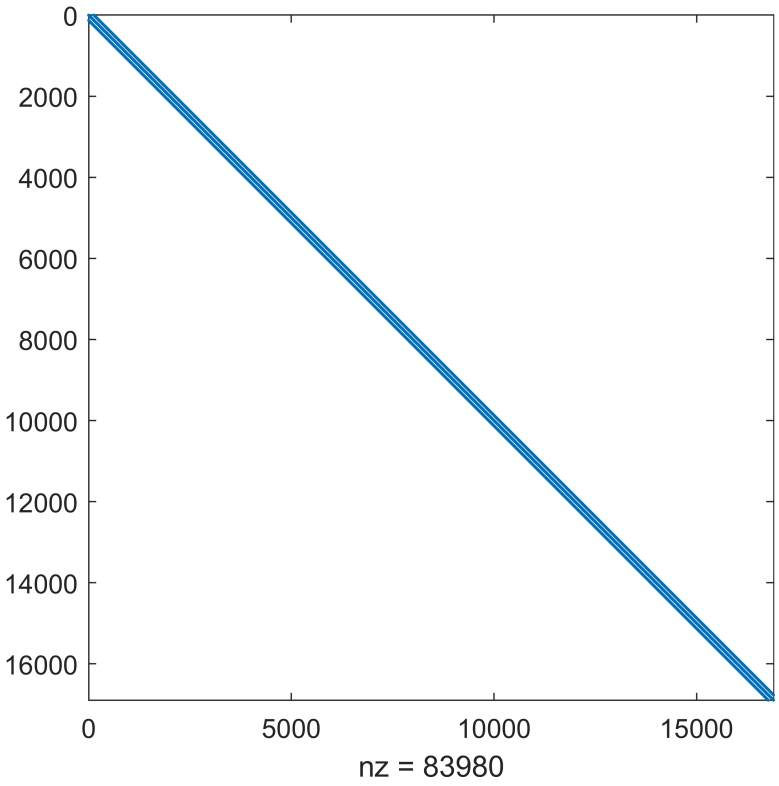
Nel caso in cui la matrice A sia una matrice di Poisson di dimensioni notevoli, si ottiene una soluzione accettabile dal punto di vista dell'errore relativo, ma il numero di iterazioni necessarie all'algoritmo per ottenere la soluzione è elevato, ciò si ripercuote sul tempo necessario per ottenere la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

jacobi(A,A*x);
```

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

```
jacobiAccuracy(A,x,1e-7,zeros(n^2,1),35000)
```

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




```
ans = struct with fields:
          cond: 1.0113e+04
     erroreRel: 3.4762e-04
    residuoRel: 1.9991e-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
	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
x0	Empty		
	Vettore colonna di dimensione n	TOL non vuoto	
	Vettore colonna di dimensione m	TOL non vuoto	ERROR
	Vettore colonna con NaN o inf	TOL non vuoto	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 =
  1×10 TestResult array with properties:

    Name
    Passed
    Failed
    Incomplete
    Duration
    Details

Totals:
   10 Passed, 0 Failed, 0 Incomplete.
   0.35472 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**
- x0 : **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**
- x0 : **indifferente**
- 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**
- x0 : **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
```

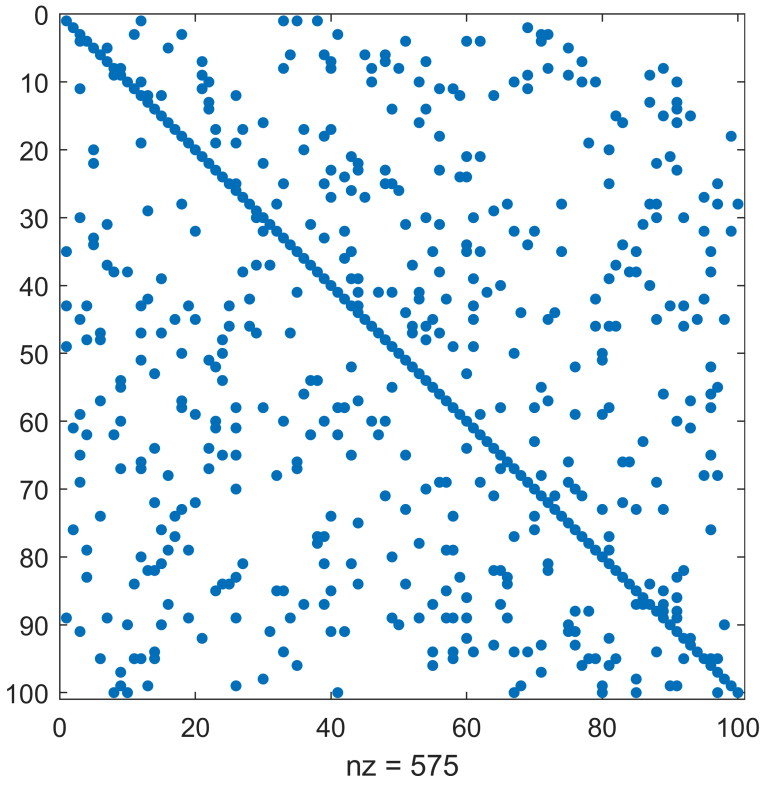
Essendo `b` una matrice e non un vettore colonna, la funzione non verrà eseguita e verrà mostrato un messaggio di errore.

Test underflow

Inoltre è interessante notare come l'algoritmo implementeato si comporti nel caso in cui viene forzato il verificarsi di una condizione di ***underflow*** nel sistema. In particolare, tale condizione si verifica nel calcolo del prodotto tra `TOL` e `x`, infatti poichè `x` è calcolato mediate passi successivi e `TOL` è un valore piccolo, può verificarsi la condizione di ***underflow*** , cioè il prodotto `TOL*x` viene sostituito da 0. Se ciò accede l'algoritmo non convergerebbe mai per quanto rigurarda la convergenza per il raggiungimento della tolleranza richiesta. Per tastare il comportamento della funzione poniamo il punto di partenza specificato `x0` pari a un vettore i cui elementi concidono con `realmin,` e quindi l'operazione `TOL*x0` nella prima iterazione del ciclo genera sicuramente un underflow. Si può osservare che la funzione implementata è robusto rispetto al verificarsi di quest'errore in quanto è stato implementato un controllo del prodotto e che nel caso di ***underflow*** sostituisce `TOL*x` con `realmin`.

```
n=100;
A=sprand(n,n,0.05)+spdiags(ones(n,1),0,n,n)*4;
x=ones(100,1);
x0=x*realmin;
b=A*x;
```

```
jacobiAccuracy(A,x,eps,x0)
```




```
ans = struct with fields:
          cond: 8.0508
     erroreRel: 4.4409e-16
    residuoRel: 1.7780e-16
         niter: 72
```

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

Gabriele Previtera