> All figures in this blog are embedded by Github Image Hosting Service. These figures may not be displayed on mobile devices.
>
> This blog contains a lot of LaTeX code, please use Chrome for reading quality or click here.

Common functions in MATLAB about sparse matrix

Run

```
>> help sparfun
```

or click here.

Create sparse matrix

`sparse`

- `S = sparse(A)` converts a full matrix into sparse form by squeezing out any zero elements. If a matrix contains many zeros, converting the matrix to sparse storage saves memory.
- `S = sparse(m,n)` generates an m-by-n all zero sparse matrix.

```
>> A = eye(10000);
>> SA = sparse(A);
>> whos A SA
```

```
  Name           Size                      Bytes  Class     Attributes

  A          10000x10000             800000000  double
  SA         10000x10000                240008  double    sparse
```

`sprand` , `sprandsym`

- `R = sprandn(S)` creates a sparse matrix that has the same sparsity pattern as the matrix `S` , but with normally distributed random entries with mean 0 and variance 1.
- `R = sprandn(m,n,density)` creates a random m-by-n sparse matrix with approximately `density*m*n` normally distributed nonzero entries for `density` in the interval [0,1].

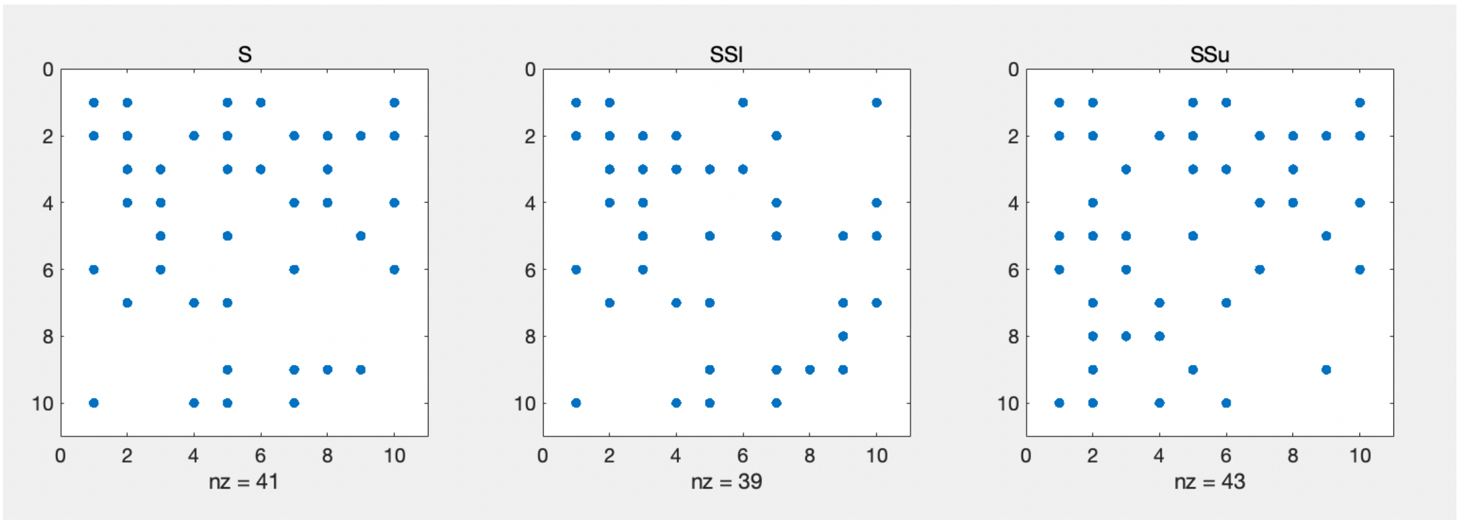
```
>> S = sprandn(5,5,0.5);


   (1,2)       1.7036
   (3,2)      -1.5975
   (3,3)      -0.8973
   (4,3)      -1.7799
   (5,3)       0.5464
   (1,4)       1.9679
   (1,5)      -0.9504
   (2,5)      -0.0473
   (3,5)      -0.0197
   (5,5)      -0.3547
```

- `R = sprandsym(S)` returns a symmetric random matrix whose lower triangle and diagonal have the same structure as `S`. Its elements are normally distributed, with mean 0 and variance 1.
- `R = sprandsym(n,density)` returns a symmetric random, n-by-n, sparse matrix with approximately `density*n*n` nonzeros; each entry is the sum of one or more normally distributed random samples, and (0 <= density <= 1).

```
%%

S = sprandn(10,10,0.5);
subplot(1,3,1); spy(S); title('S');
SSl = sprandsym(S);
subplot(1,3,2); spy(SSl); title('SSl');
SSu = sprandsym(S');
subplot(1,3,3); spy(SSu); title('SSu');
```



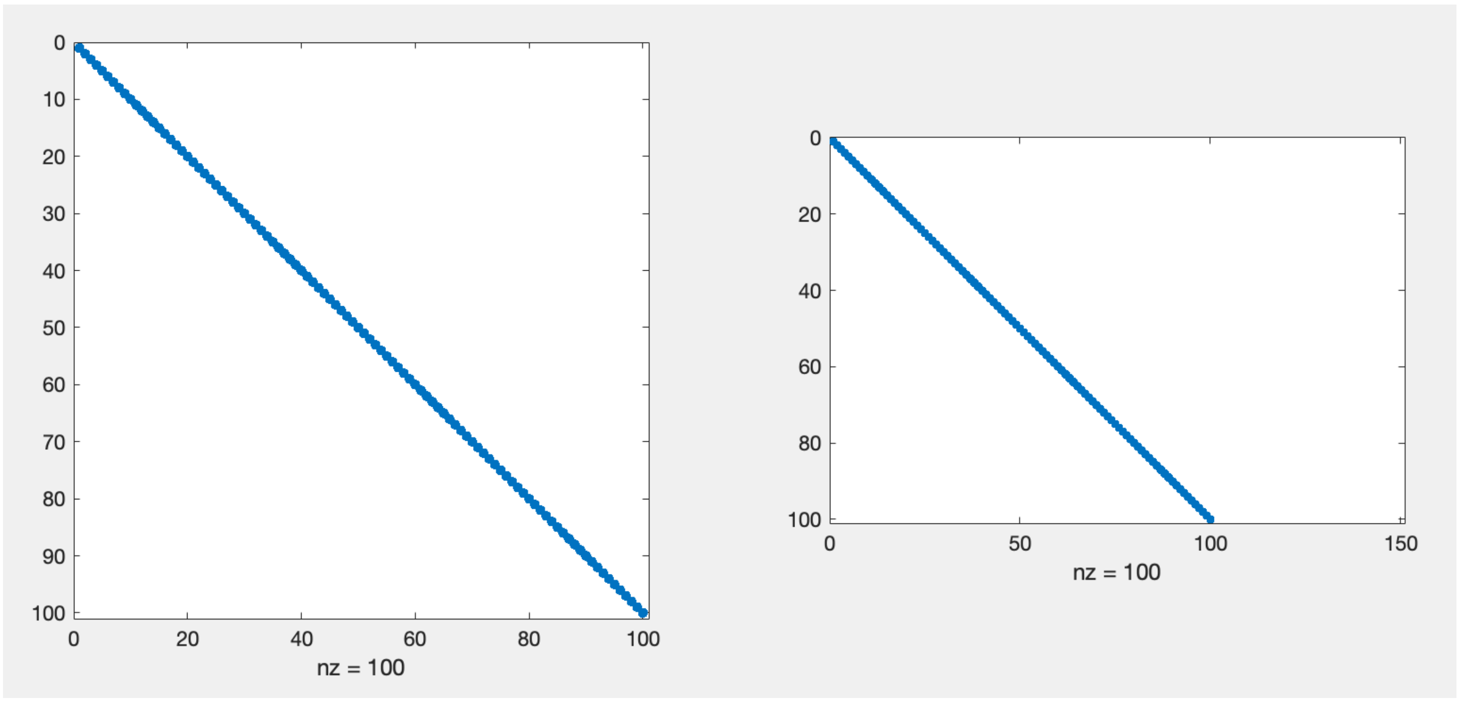
speye

- `S = speye` returns a sparse scalar 1.

- `S = speye(n)` returns a sparse n-by-n identity matrix, with ones on the main diagonal and zeros elsewhere.
- `S = speye(n,m)` returns a sparse n-by-m matrix, with ones on the main diagonal and zeros elsewhere.

```
%% speye

S1 = speye(100); subplot(1,2,1); spy(S1);
S2 = speye(100,150); subplot(1,2,2); spy(S2);
```



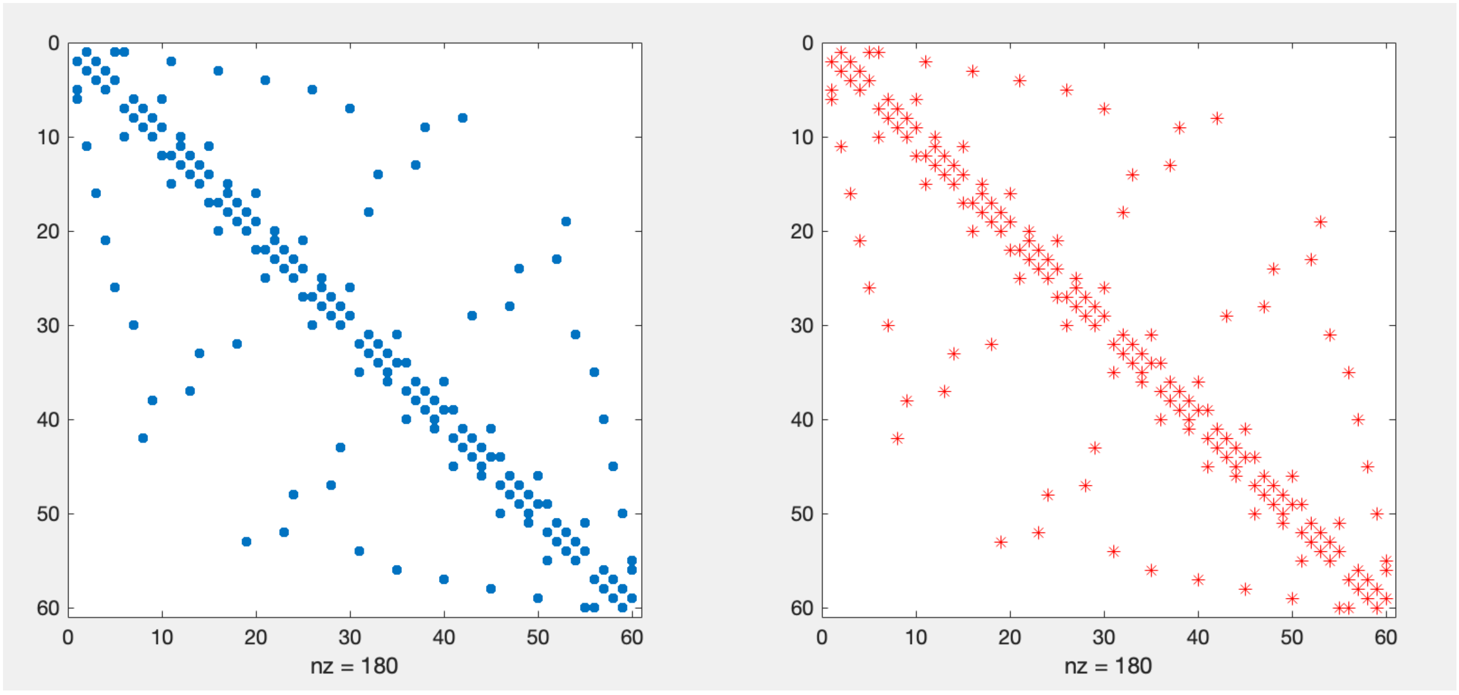
Display sparse matrix

spy

- `spy(S)` plots the sparsity pattern of matrix S. Nonzero values are colored while zero values are white. The plot displays the number of nonzeros in the matrix, `nz = nnz(S)` .
- `spy(S,LineSpec)` additionally specifies `LineSpec` to give the marker symbol and color to use in the plot.

```
%% spy

S = bucky;
subplot(1,2,1); spy(S);
subplot(1,2,2); spy(S, 'r*')
```



`full`

`A = full(S)` converts sparse matrix `S` to full storage organization, such that `issparse(A)` returns logical `0` (false).

```
%% full

S = speye(5);
A = full(S);
disp(issparse(A))
disp(A)
```

```
   0

     1     0     0     0     0
     0     1     0     0     0
     0     0     1     0     0
     0     0     0     1     0
     0     0     0     0     1
```

Conjugate Gradients

Algorithm

Let $A(N \times N)$ be symmetric positive definite (SPD), that is $A^T = A$ and all eigenvalues is positive, or say $x^T Ax > 0$ for any $x \neq \mathbf{0}$. Let b be an $N \times 1$ vector, we want to solve a x_* s.t. $Ax_* = b$, since A is SPD A is non singular, and hence x_* is existed and unique.

Conjugate Gradients (CG) find x_* iteratively starting from x_0, x_0 is could be any vector, and we will use $\mathbf{0}$. And x_n belongs to the Krylov space:

$$K_n = \langle b, Ab, \cdots, A^{n-1}b \rangle$$

which is spanned by vectors $b, Ab, \cdots, A^{n-1}b$, and any $k \in K_n$ is a linear combination of these vectors. It is direct to see that $K_n \subseteq \mathbb{R}^N$, so conjugate gradient is searching in bigger and bigger space for a good approximation to the solution.

Let define a special way to measure the size of a vector: define "A - norm" of a vector x as

$$\|x\|_A = \sqrt{x^T Ax}$$

So $\|x\|_A$ is a positive real number if $x \neq \mathbf{0}$, because by definition, A is symmetric positive definite, and therefore $x^T Ax$ is a positive number.

And define the error of each steps: define n-th error as

$$e_n = x_* - x_n$$

and define n-residual as

$$r_n = Ae_n = b - Ax_n$$

In principle, we like a small error, but in practice, the thing we can measure is the residual. Thus we minimize the residual as a mean of minimizing.

THM 1. Among all $x \in K_n$, there exists one unique x_n such that minimizes $\|e_n\|_A$.

So at every step of conjugate gradient, we can find the best approximation in a certain measure. And from this theorem, we obtain a corollary: $\|e_0\|_A \geq \|e_1\|_A \geq \cdots \geq 0$. That is the error keep going down monotonically.

Define $f(x) = \frac{1}{2}x^T Ax - x^T b$, thus $\nabla f(x) = Ax - b$, now we want $Ax = b$, which means we want $\nabla f(x) = 0$, i.e minimize of $f(x)$.

Now we can introduce the conjugate gradient algorithm. First, we start form an initial guess $x_0 = \mathbf{0}$, corresponding to this is a residual $r_0 = b - Ax_0 = b - A\mathbf{0} = b$. And then we define search direction vector $p_0 = r_0$ which indicate the search direction in Krylov space.

For $n = 1, 2, 3, \cdots$:

- $\alpha_n = r_{n-1}^T r_{n-1} / p_{n-1}^T A p_{n-1}$
- $x_n = x_{n-1} + \alpha_n p_{n-1}$
- $r_n = r_{n-1} - \alpha_n A p_{n-1}$
- $\beta_n = r_n^T r_n / r_{n-1}^T r_{n-1}$
- $p_n = r_n + \beta_n p_{n-1}$

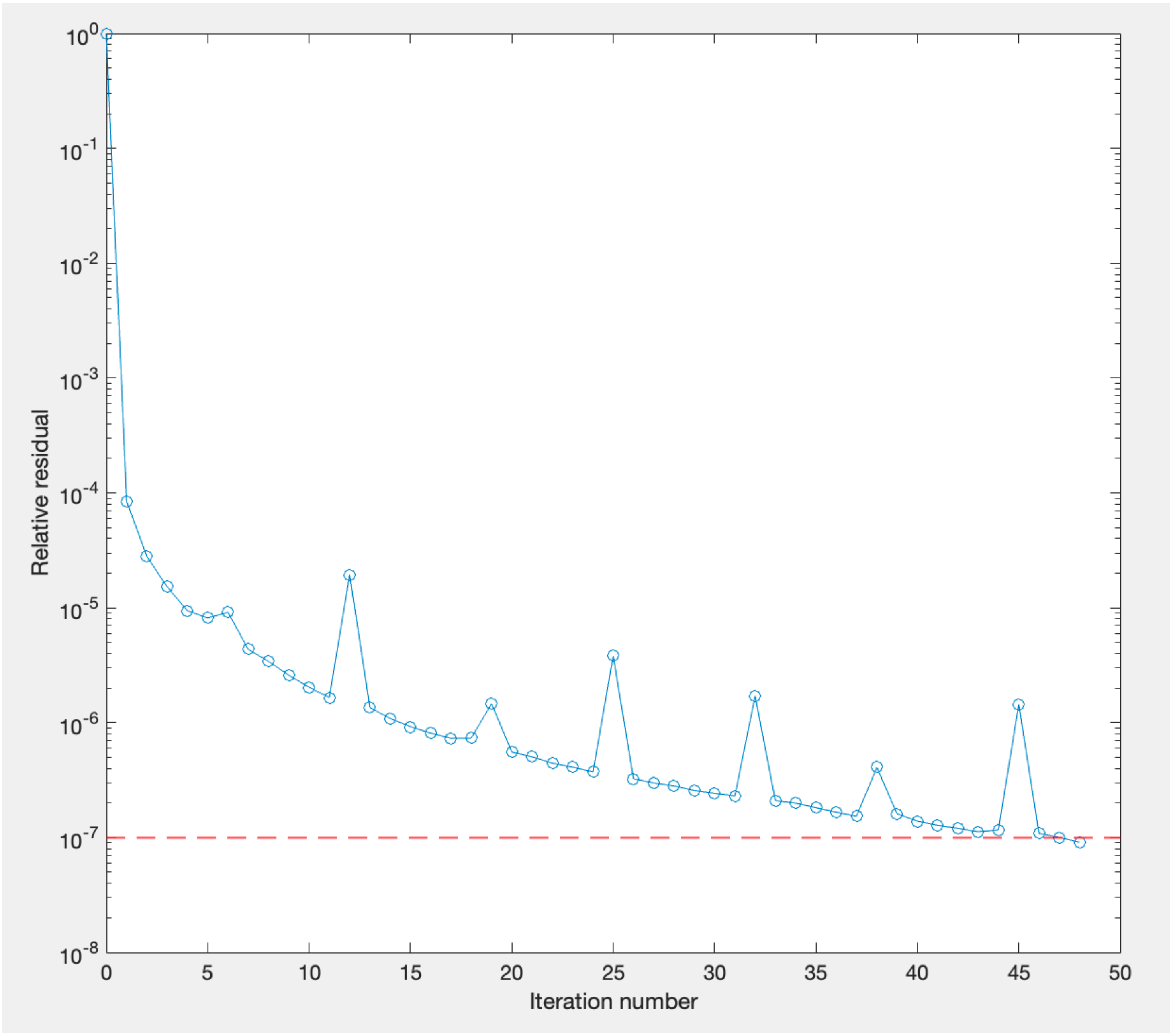
Here for each step n, α_n which is a scalar, represents the step length, it is how far we are going along a search direction to improve our current guess. x_n is the approximate solution at step n, and r_n is the corresponding residual. And β_n measures the improvement in this step. And finally, p_n is the search direction.

```
%%

clc; clear; clf
rng default
A = sprand(1000,1000,.75);
A = A'*A;
b = sum(A,2);

tol = 1e-7; maxit = 150;
[x,fl0,rr0,it0,rv0] = pcg(A,b,tol,maxit);

semilogy(0:length(rv0)-1,rv0/norm(b),'-o')
yline(tol,'r--', 'LineWidth', 1);
xlabel('Iteration number')
ylabel('Relative residual')
```



Convergence of CG

Note that

$$e_0 = x_* - x_0 = x_*$$

and

$$e_n = x_* - x_n = e_0 - x_n$$

and since $b = Ax_* = Ae_0$ we have that

$$x_n \in K_n = \langle b, Ab, \cdots, A^{n-1}b \rangle = \langle Ae_0, A^2e_0, \cdots, A^ne_0 \rangle$$

And hence x_n is the linear combination of $Ae_0, A^2e_0, \cdots, A^ne_0$ and can be interpreted as a polynomial of A. So in conjugate gradient, we implicitly dealing with n-degree polynomial of A timing a constant e_0.

Thus $e_n = e_0 - x_n$ where x_n is a polynomial of A can be rewritten as

$$e_n = e_0 P_n(A)$$

where P_n is a polynomial of (at most) degree n with the property $p_n(0) = 1$. The THM 1. tells us that $\|P_n(A)e_0\|$ is the minimal among polynomials P of degree n with $P(0) = 1$.

Since A is a symmetric positive defined matrix thus it has real valued, orthogonal eigenvectors and hence has an orthogonal basis. Let A has positive eigenvalues $\lambda_1, \cdots, \lambda_n$ and eigenvectors $v_1, \cdots, v_n$, then $e_0 = \sum_{k=1}^N a_k v_k$ and

$$e_n = P_n(A)e_0 = P_n(A)\sum_{k=1}^N a_k v_k = \sum_{k=1}^N a_k P_n(\lambda_k) v_k$$

Hence if there exists polynomials that are small at eigenvalues, then CG converges quickly. Thus the whole logic of the argument is that hte existence of the polynomial. Note that the $P_n(0) \equiv 0$, thus if some λ are too close to 0, then the polynomial would be non exist which means CG will not convergent, there is a theorem shows this fact:

THM 2. The ratio of the error at the end step to the initial error is bounded by the minimal overall polynomials with the restriction of maximal of the eigenvalues of the matrix:

$$\frac{\|e_n\|_A}{\|e_0\|_A} \le \min_{P_n} \max_{\lambda_k} |P_n(\lambda_k)|.$$

where $P_n(0) = 1$.

For example, if we create a SPD matrix which has very small eigenvalues:

```
%%

clc; clear; clf

A = randn(1000);
A = A'*A;
e = eig(A);
format long
disp([min(e), max(e)])
format short
```

```
   1.0e+03 *

   0.000000557765389   2.000311897319554
```

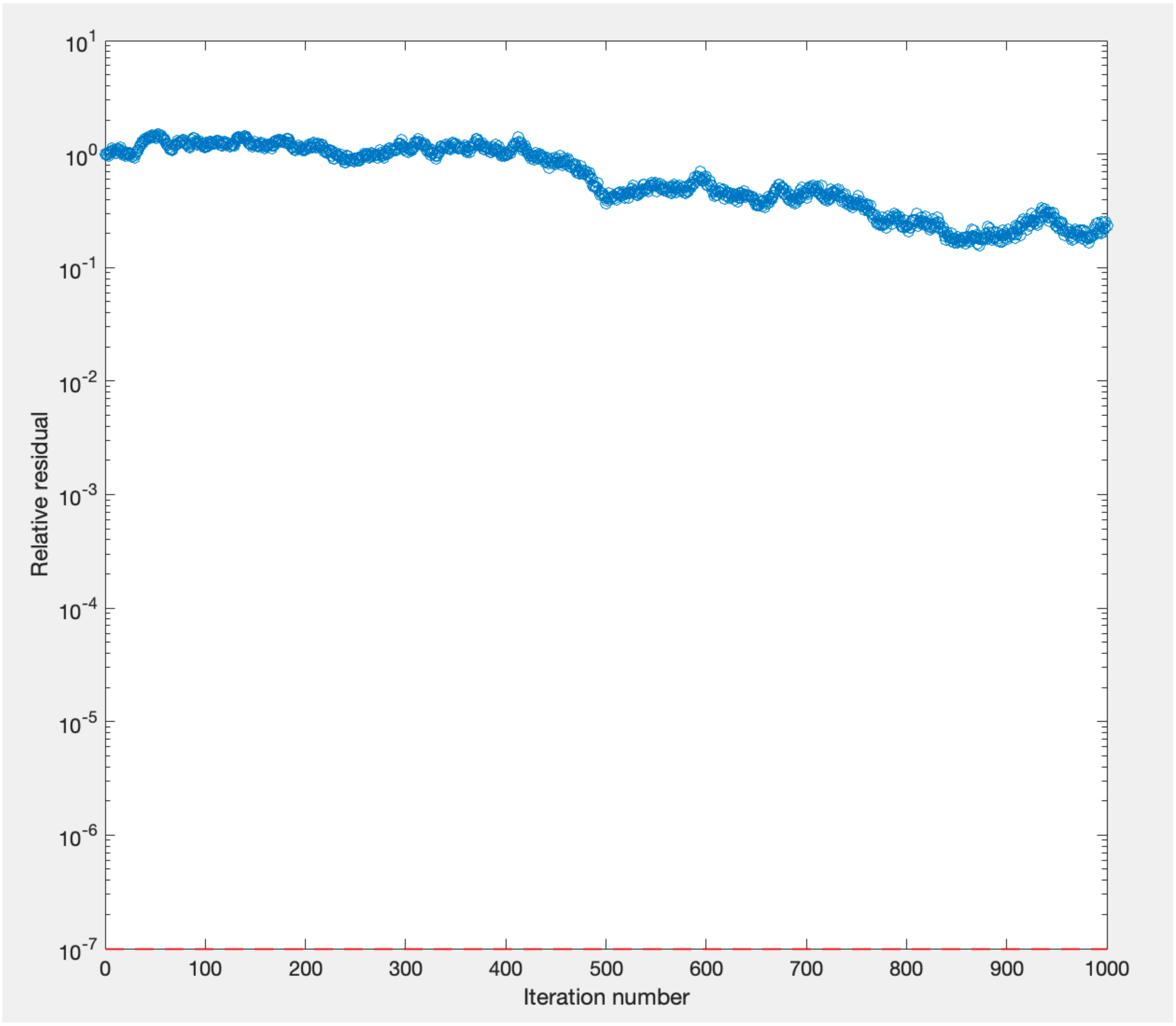
As we can see that matrix A has a very small eigenvalue. And the CG on A performs badly:

```
%%

clc; clear; clf
A = randn(1000);
A = A'*A;
b = ones(1000,1);

tol = 1e-7; maxit = 1000;
[~,~,~,~,rv0] = pcg(A,b,tol,maxit);

semilogy(0:length(rv0)-1,rv0/norm(b),'-o')
yline(tol,'r--', 'LineWidth', 1);
xlabel('Iteration number')
ylabel('Relative residual')
```



We can modify A like

```
%%

clc; clear; clf

A = randn(1000);
A = A'*A;
A = A + 1000*eye(1000);
e = eig(A);
format long
disp([min(e), max(e)])
format short
```

```
   1.0e+03 *

     1.000000394127676   4.946870550127192
```

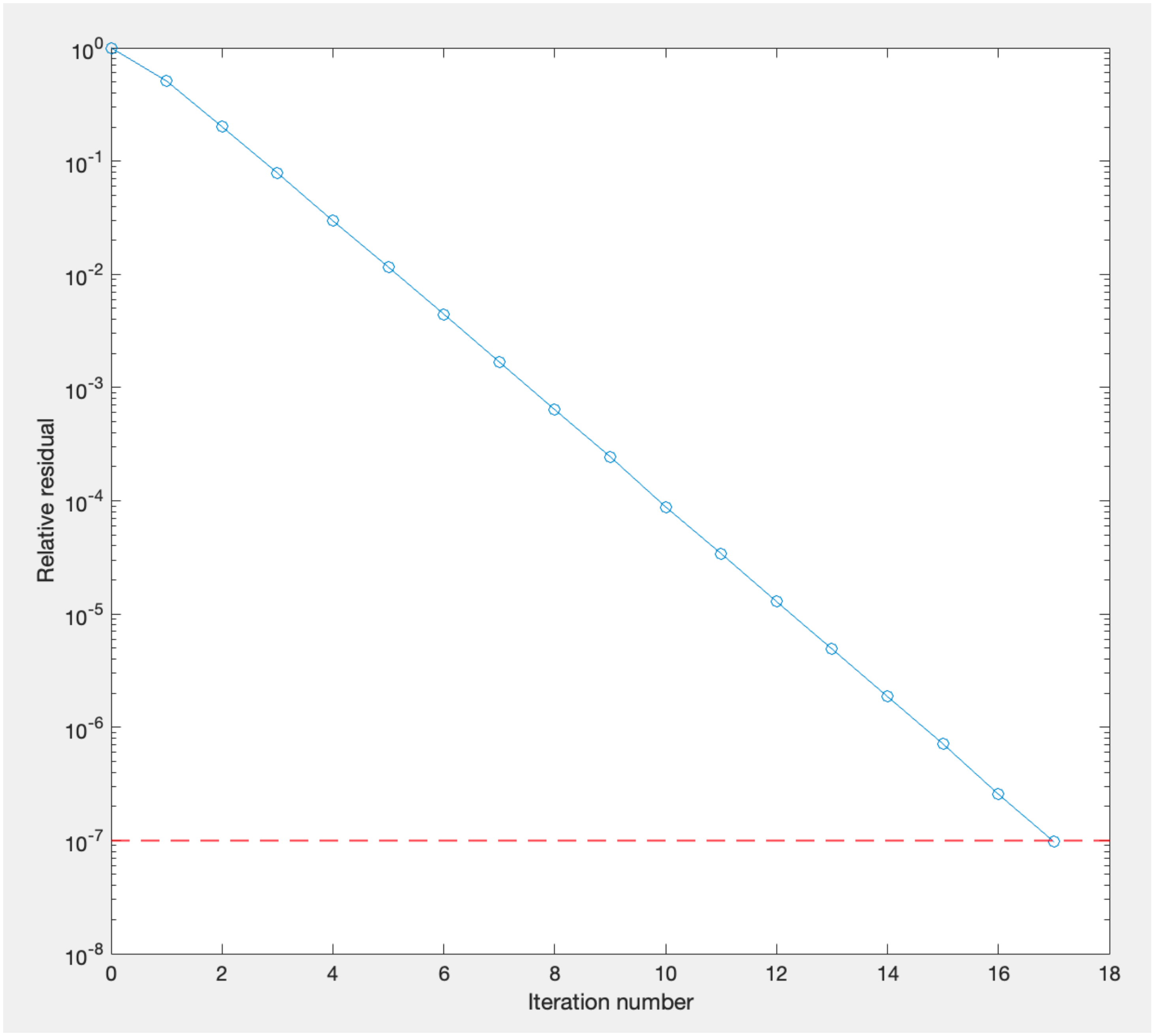
and then CG converges quickly (in 17 steps):

```
%%

clc; clear; clf
A = randn(1000);
A = A'*A;
A = A + 1000*eye(1000);
b = ones(1000,1);

tol = 1e-7; maxit = 1000;
[~,~,~,~,rv0] = pcg(A,b,tol,maxit);

semilogy(0:length(rv0)-1,rv0/norm(b),'-o')
yline(tol,'r--', 'LineWidth', 1);
xlabel('Iteration number')
ylabel('Relative residual')
```



Preconditioned CG