Homework 2. Numerical methods for ODES

Valentina Zonta

January 11, 2024

1 Exercise 1

We produce an implementation of PCG following the algorithm using the syntax:

```
[x, resvec, iter] = mypcg(A, b, tol, maxit, L)
```

In the algorithm

```
Input: x_0, A, M, b, k_max, tol
r_0 = b - Ax_0, p_0 = M^(-1)r_0, k = 0, rho_0 = r_0^T M^(-1)r_0

while k <= k_max and ||r_k|| > tol ||b|| do
    1. z_k = Ap_k
    2. alpha_k = rho_k / (z_k^T p_k)
    3. x_{k+1} = x_k + alpha_k p_k
    4. r_{k+1} = r_k - alpha_k z_k
    5. g_{k+1} = M^(-1)r_{k+1}
    6. rho_{k+1} = r_{k+1}^T g_{k+1}
    7. beta_k = rho_{k+1} / rho_k
    8. p_{k+1} = g_{k+1} + beta_k p_k
    9. k = k + 1
end while
```

, we apply the Cholesky preconditioner $r_{k+1} = M^{-1}r_{k+1}$, performing a sequential solution of two sparse triangular linear systems. From $r_{k+1} = (LL^T)^{-1}r_{k+1}$, we obtain $LL^T r_{k+1} = r_{k+1}$ and hence:

$$Ly = r_{k+1}$$

$$L^T r_{k+1} = y$$

We then compare the implementation of the PCG with the Matlab pre-built function. Using the Cholesky preconditioner, the exact solution with all ones b (right handed side), tolerance $1e-8$), and a maximum iteration number of 50, we get the plot in Fig.??:

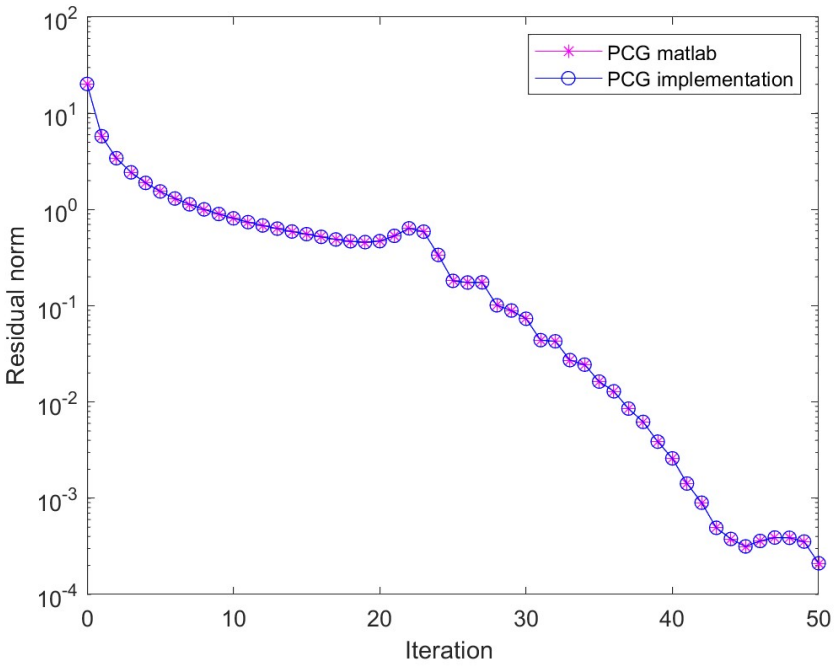


Figure 1: PCG Matlab vs implemented PCG

As we can see, the implemented PCG and the Matlab PCG behave in the same manner.

2 Exercise 2

We create a sparse matrix A representing the discrete Laplacian operator on a 2D grid, using the method of finite differences with $nx = [102, 202, 402, 802]$ grid points along each dimension. We set the r.h.s. b corresponding to the exact solution x with components $x_i = \frac{1}{\sqrt{i}}$, with $i = 1, 2, \ldots$. We use tolerance $= 10^8$ and 5000 as the maximum number of iterations. We calculate the mesh discretization parameter h as $h = 1/(nx(i) - 2)$. Indeed the border of the mesh is missing the four corners, and the spaces between the points are the number of points minus 1.

We solve the linear system $Ax = b$ in four different ways: without preconditioner, PCG with IC(0), PCG with IC droptol 10^{-2} and last IC droptol 10^{-3}. The results are shown in the following table:

n	h	CG	PCG IC(0)	PCG droptol 10^{-2}	PCG droptol 10^{-3}
102	0.01000	283	87	45	17
202	0.00500	532	159	78	30
402	0.00250	948	282	137	53
802	0.00125	1792	533	258	9

To verify the dependence between the number of iterations and the root of the condition number (proportional to h^{-1}) we write $h^{-1} \propto \alpha \cdot iter$. In the following table, we show the constant α for each CG. The constants get a little smaller depending on h, probably due to the machine's precision. However, there is proportionality for each method.

$\frac{1}{h}$	$\alpha_{\text{PCG \& IC(0)}}$	$\alpha_{\text{IC(0)}}$	$\alpha_{\text{PCG droptol } 10^{-2}}$	$\alpha_{\text{PCG droptol } 10^{-3}}$
100	2.8	0.8	0.4	0.2
200	2.7	0.8	0.4	0.2
400	2.4	0.7	0.3	0.1
800	2.2	0.7	0.3	0.1

3 Exercise 3

The matrix described in the exercise is:

$$A = \begin{bmatrix} 200 & 0 & 0 & 0 & 0 & 0 & \dots \\ 0 & 400 & 0 & 0 & 0 & 0 & \dots \\ 0 & 0 & 600 & 0 & 0 & 0 & \dots \\ 0 & 0 & 0 & 800 & 0 & 0 & \dots \\ 0 & 0 & 0 & 0 & 1000 & 0 & \dots \\ 0 & 0 & 0 & 0 & 0 & 1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

.

For our purpose we use the following theorem, proved at https://pages.di.unipi.it/gemignani/lecture2.pdf.

Theorem 1 *Let A be symmetric positive definite (spd). Assume that there are exactly $k \leq N$ distinct eigenvalues of A. Then the Conjugate Gradient (CG) iteration terminates in at most k iterations.*

We modify the prove of previous theorem in order to demonstrate the theorem based on having 6 eigenvalues with degeneracy.

Let λ_m be the eigenvalues associated with the eigenvectors $|\lambda_{m,n'}\rangle$ where n' is the degeneracy index. Using the Spectral Theorem we can write the solution

$$x^* = \sum_{m}^{6} \sum_{n'}^{d(\lambda_m)} \frac{\gamma_{m,n'}}{\lambda_m} |\lambda_{m,n'}\rangle = \sum_{m}^{6} |v_m\rangle \,, \tag{1}$$

where we wrote the vector v as a linear combination of the degenerate eigenvectors living in the eigenspaces of dim 6. Then with the same reasoning we can expand the Matrix A and $P(A)$ as:

$$A = \sum_{k=1}^{6} \lambda_k \sum_{n=1}^{d(\lambda_n)} |\lambda_{k,n}\rangle \langle \lambda_k, n| = \sum_{k=1}^{6} \lambda_k |w_k\rangle \langle w_k| \,, \tag{2}$$

where $|w_k\rangle\langle w_k| = \sum_{n=1}^{d(\lambda_n)} |\lambda_{k,n}\rangle\langle\lambda_k, n|$ is a projector, since sum of orthogonal projectors.

$$P(A) = \sum_{k=1}^{6} \prod_{l=1}^{6} \frac{(\lambda_l - \lambda_k)}{\lambda_l} |w_k\rangle \langle w_k| \,. \tag{3}$$

Then we get :

$$\begin{aligned} P(A)x^* &= \sum_{k} \prod_{l=1}^{6} (\lambda_l - \lambda_k) |\omega_k\rangle \cdot \text{const}, \qquad (\langle w_k \mid v_m\rangle = \text{const} \cdot \delta_{m,k}\,) \\ &= \sum_{k=1}^{6} (\lambda_1 - \lambda_k)(\lambda_2 - \lambda_k) \dots \quad |w_k\rangle \cdot \text{const} = 0 \end{aligned}$$

Because $\|x^* - x_k\|_A = \min_{p \in P_k} \|p(A)(x^* - x_0)\|_A$ and $x_0 = 0$, we get:

$$\|x_k - x^*\|_A \left\|\bar{P}(A)x^*\right\|_A = 0 \tag{4}$$

So the algorithm ends at the iteration number k=6. As we can see from the plot underneath the total iterations necessary for the CG method are exactly 6.

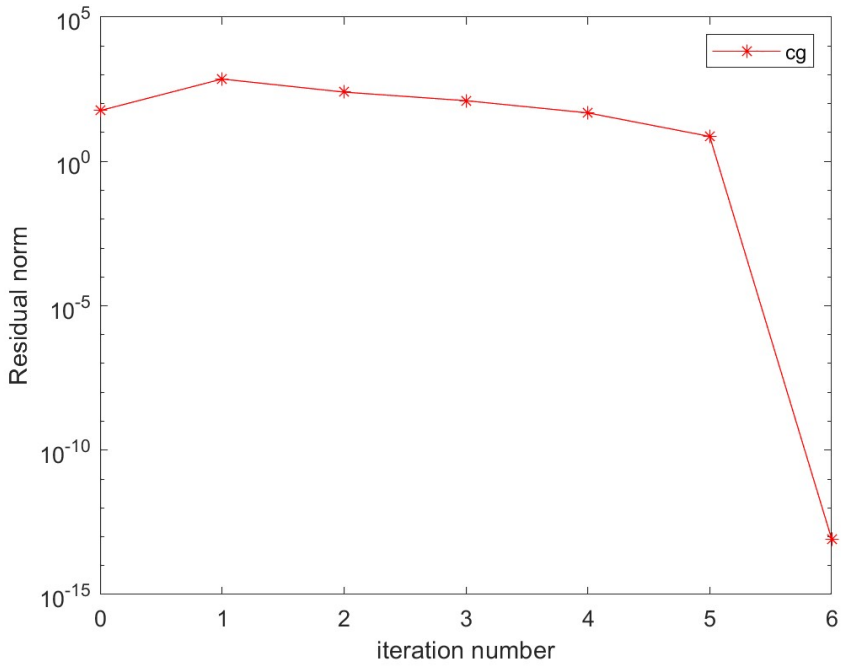


Figure 2: PCG Matlab vs implemented PCG

4 Exercise 4

To generate a symmetric positive definite (SPD) matrix, one approach is to use the MATLAB command:

```
A = gallery('wathen', 100, 100);
```

The matrix A serves as the coefficient matrix, and b corresponds to the right-hand side generated from a random vector solution. To assess the performance, we compare the results obtained with the non-preconditioned Conjugate Gradient (CG) method against those achieved with Jacobi and IC(0) preconditioners. The following image shows the results:

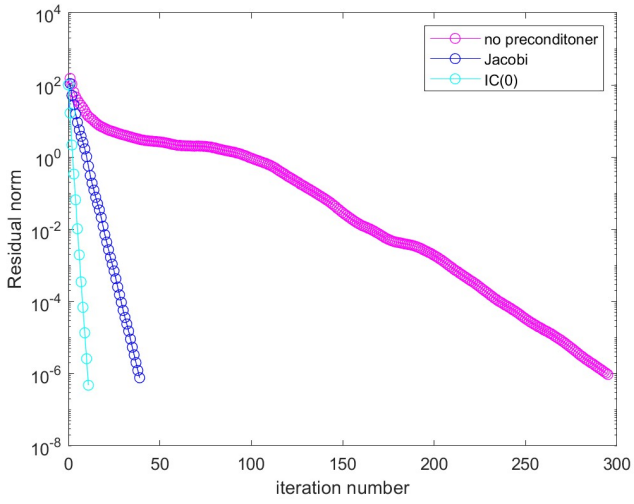


Figure 3: PCG Matlab vs implemented PCG

The fact that the results obtained without any preconditioner require 300 iterations, significantly more than the other two methods, underscores the impact of preconditioning on convergence. Among the preconditioners employed, IC(0) demonstrates superior efficiency and accuracy, making it a favorable choice for a linear system.

5 Exercise 5

In this exercise, the implementation of the GMRES method (without restart) is presented as a MATLAB function following the algorithm provided in the slides. The function has the following syntax:

```
function [x, iter, resvec, flag] = mygmres(A, b, tol, maxit, x0)
```

We follow the algorithm studied during lessons, in particular computing the QR factorization, to reduce the computational cost.

```
Input: x0, A, b, kmax, tol
r0 = b - Ax0, k = 0, rho0 = ||r0||^2, beta = rho0, v1 = r0 / beta

while rho_k > tol * ||b||^2 and k < kmax do
    1. k = k + 1
    2. v_k+1 = Av_k
    3. for j = 1 to k do
         h_jk = vv_j
         v_k+1 = v_k+1 - h_jk v_j
       end for
    4. h_k+1,k = k ||v_k+1||^2
    5. v_k+1 = v_k+1 / h_k+1,k
    6. Compute the QR factorization of H_k+1,k: H_k+1,k = QR.
    7. rho_k = |beta * q,k|
end while

y_k = argmin beta * e - H_k+1,k y_k by using the QR factorization of H_k+1,k
x_k = x0 + V_k
y_k
```

In particular, the minimization of the residual can be written as

$$\|r_k\|_2 = \min_{y_k} \|\beta e_1 - H_{m+1,m} y_k\|_2 \tag{5}$$

We factorize $H_{m+1,m} = QR$, with Q being orthogonal of size $((m+1) \times (m+1))$ and R of size ((m + 1)× m), where the$(m \times m)$ submatrix of R is upper triangular and the last row is equal to zero.

After computing the QR factorization:
$H_{m+1,m} = QR$ (computational cost $O(m^3)$), we have

$$\min_k \|\beta e_1 - H_{m+1,m} y_k\|_2 = \min_k \|\beta Q^T e_1 - R y_k^2\|_2 = \min_k \|g - R y_k\|_2, \tag{6}$$

where $g = \beta Q^T e_1$.

We solve the linear system $Ax = b$ with b corresponding to an exact solution with components $x_i = \sqrt{\frac{1}{x_i}}$ using the GMRES implementation with tolerance $1e^{-10}$, maximum iteration 550, and x0 all zero vector. The following graph shows the solutions:

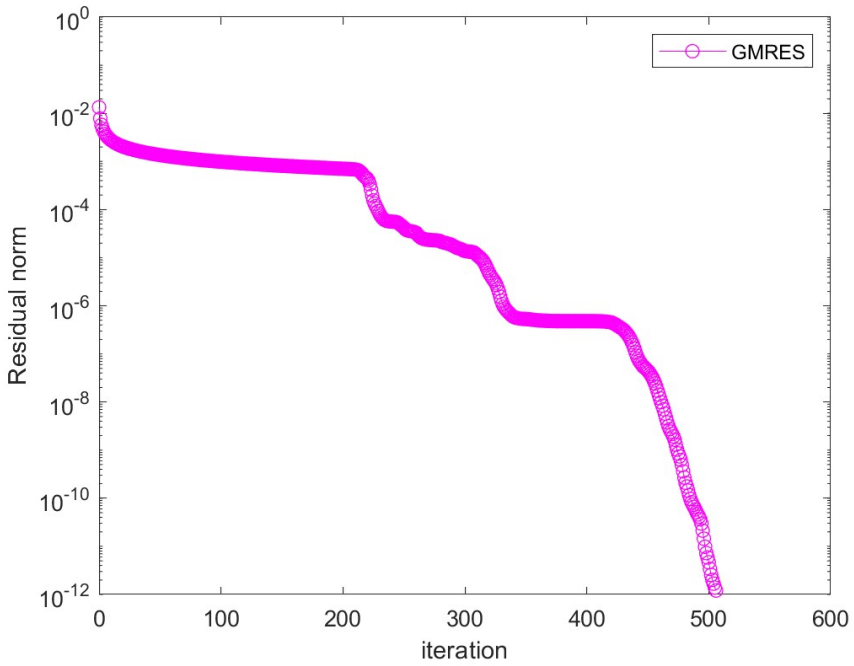


Figure 4: implemented GMRES

To check the correctness of the function We compare the implemented non-preconditioned GMRES with the preconditioned one in the following section. As expected the non-preconditioned one takes more iteration to get to the final results.

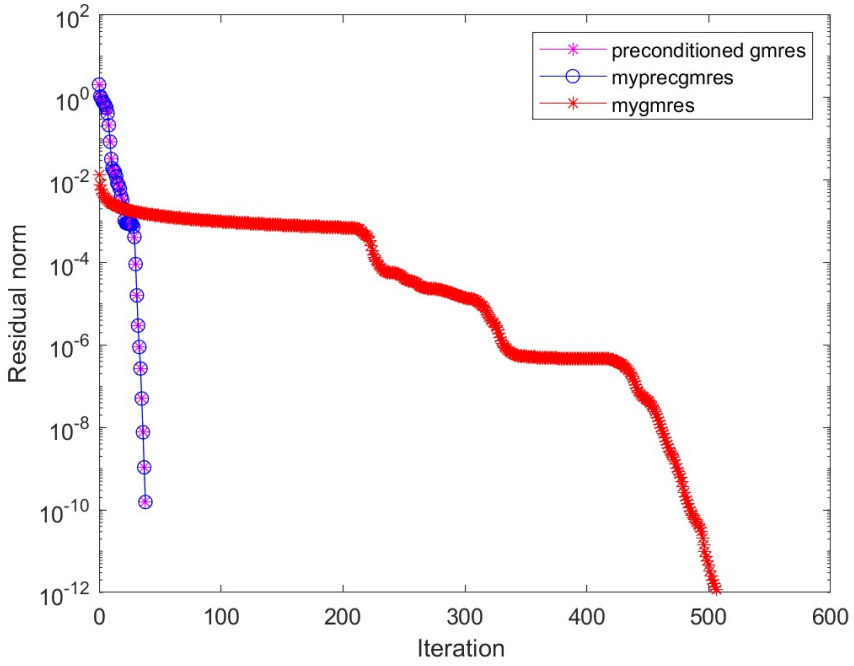


Figure 5: implemented GMRES

6 Exercise 6

We implement the preconditioned GMRES method without restarting as a Matlab function with the following syntax

```
function[x,iter,resvec,flag]=myprecgmres(A,b,tol,maxit,x0,L,U)
```

We implement the program modifying the non-preconditioned GMRES, using as preconditioner a matrix M such that

$$M^{-1}Ax = M^{-1}b \tag{7}$$

In this case the initial residual becomes $r_0 = M^{-1}(b - Ax_0)$ and $v_{k+1} = M^{-1}Av_k$. This last step is implemented as

$$\begin{aligned} z &= Av_k \\ w &= L\backslash z \\ v_{k+1} &= U\backslash w \end{aligned}$$

We compute, using the matrix A given for the exercise, the ILU preconditioner with drop tolerance 0.1. We use b as the corresponding to an exact solution with components $x_i = \frac{1}{\sqrt{i}}$. As for the other impute parameters We use maximum iteration 550 and tolerance 1e-10.
We compare the results obtained with the function myprecgmres with the Matlab function gmres. The comparisons are shown in the Table and graph below:

Method	Iterations	Last Residual Vector
MATLAB GMRES	38	1.5592×10^{-10}
Implemented GMRES	38	1.5592×10^{-10}

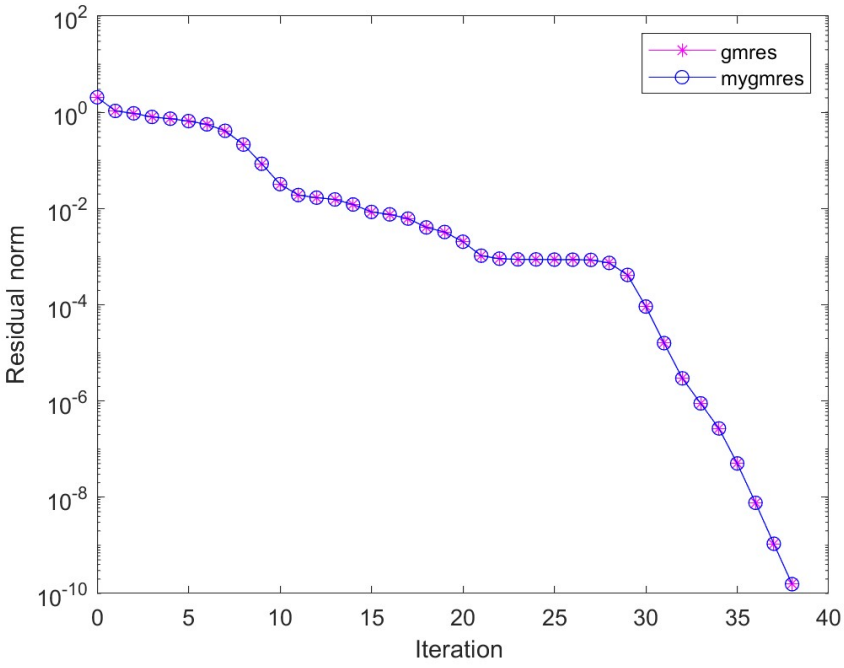


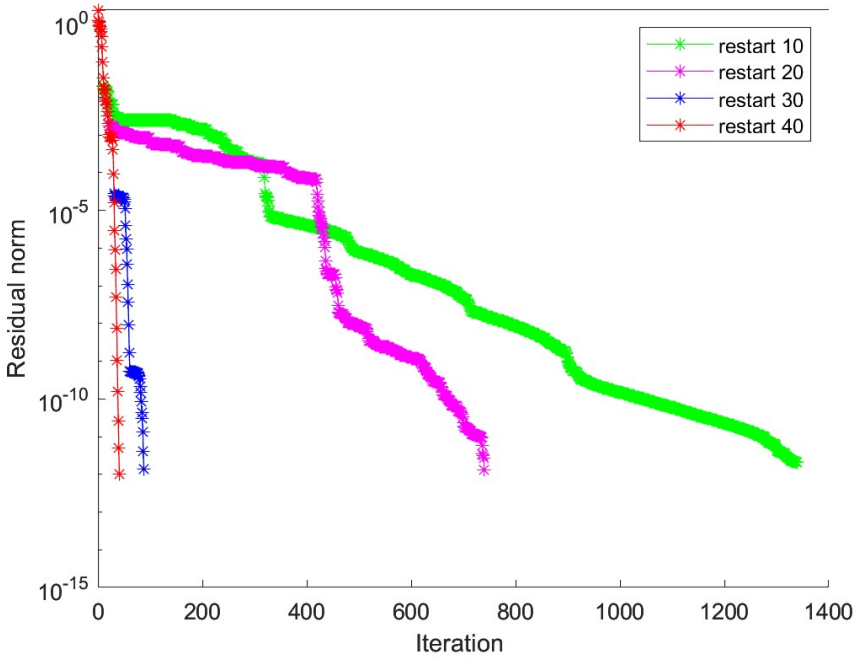
Figure 6: implemented GMRES

7 Exercised 7

Using the same matrix A and vector b from the previous section we solve the linear system Ax=b. In this case, using Matlab GMRES with tolerance 1e-12, different restart 10,20,30,50, and the ILU preconditioner with droptol 1e-2.
In the next table, we analyze the difference between the various restart methods:

Restart	Total Iterations	Relres	CPU Time (s)
10	1339	9.9×10^{-13}	1.6
20	720	6.4×10^{-13}	0.9
30	88	6.7×10^{-13}	0.1
50	41	4.8×10^{-13}	0.06
190	41	4.8×10^{-13}	0.07
5000	41	4.8×10^{-13}	0.1

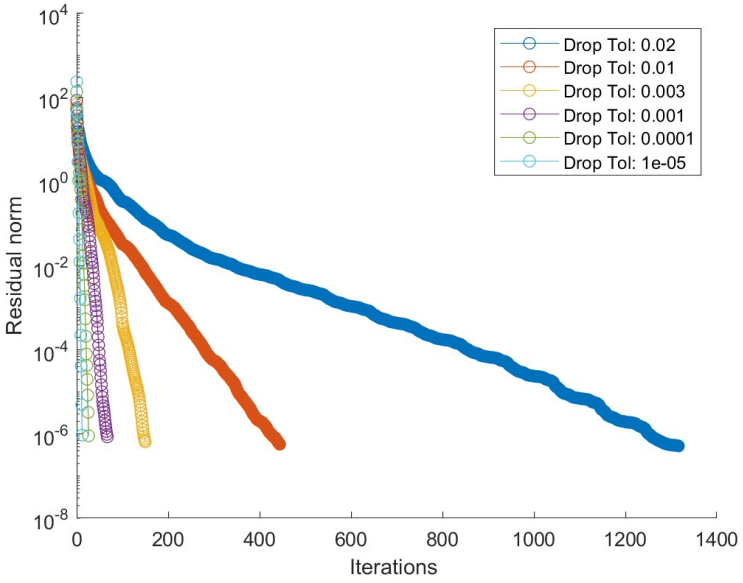
We solve the system with other two restarts of 190 and 5000 to better analyze the method. In general larger restart value allows for a larger Krylov subspace, potentially improving convergence, but it comes with increased memory requirements. When the number of iterations becomes large, order of a hundred, the cost may be prohibitive. In this case, the total iteration is not enough high, and is better to use no restart.
As we can see the total iteration needed to get to the final vector x is 41, so increasing the iterations from 41 will increase the CPU time. This is shown in the CPU time of restart 190 and 5000 which are higher than the one in 50. On the other hand, using a too-short restart number has a high computational cost. In conclusion, because the total iteration without restart is relatively small, is more efficient and precise to use GMRES without restart.
In the next plot, we can see the convergence and iteration numbers of the thirst five restarts in comparison.



8 Exercise 8

In this exercise, We compute several ILU preconditioned GMRES with different drop tolerances ($droptol = 2 \times 10^{-2}, 10^{-2}, 3 \times 10^{-3}, 10^{-3}, 10^{-4}, 10^{-5}$) keeping the restart parameter fixed to 50 .

For Incomplete LU, the Drop tolerance field is used to tune the maximum allowed sizes of dropped (neglected) elements. A smaller drop tolerance means that the preconditioner drops fewer elements and so the preconditioner becomes more accurate.
In the next plot, we show the convergence profile of the six runs on the same figure.



Droptol	Iterations	tprec (s)	tsol (s)	ttot (s)	Res. Norm	ρ
2×10^{-2}	1316	35.2	37.9	73.1	9.9×10^{-9}	0.5
10^{-2}	444	40.9	11.4	52.3	9.9×10^{-9}	0.6
3×10^{-3}	150	40.7	7.4	48.1	9.3×10^{-9}	0.9
10^{-3}	67	39.7	2.8	42.5	9.9×10^{-9}	1.5
10^{-4}	26	45.3	1.6	47.0	6.6×10^{-9}	3.5
10^{-5}	12	87.3	1.5	88.8	4.0×10^{-9}	9.1

If ρ is close to zero, it indicates that the preconditioner introduces very few non-zero entries compared to the original matrix. If ρ is significantly larger, it implies that the preconditioner introduces a notable number of non-zero entries. In fact, when the tolerance gets smaller the time spent to calculate the preconditioner is higher, and the density increases. The norm of residual as expected get smaller with the precision of the preconditioner.