Exercise # 2. Iterative Methods For Linear Systems.

Alexandre Rodrigues (2039952)

January 9, 2022

Question 1

Using as a test the example usage, with $tol = 1 \times 10^{-8}$ and limiting the iterations to $maxit = 250$, I got the following results.

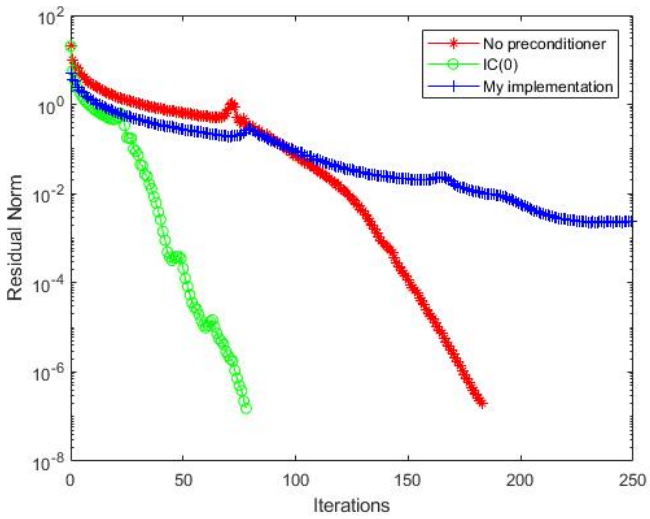

Figure 1: Residual norm vs iteration number for PCG methods, $maxit = 250$

Method	**Iterations**	**Final Residual**	**Computational Time**
Matlab PCG without preconditioning	183	1.9591×10^{-7}	$0.077s$
Matlab PCG IC(0)	78	1.5293×10^{-7}	$0.068s$
My PCG implementation	250	2.3×10^{-3}	$0.151s$

Table 1: Results of PCG methods, $maxit = 250$

When $maxit$ is large enough to guarantee convergence in all implementations we get the following results:

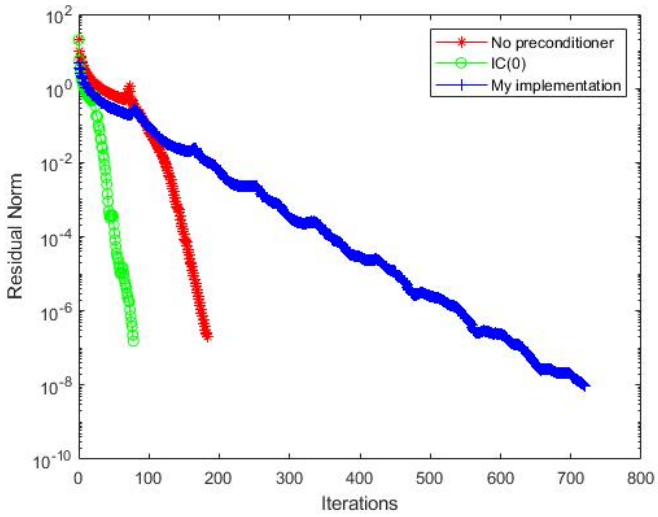

Figure 2: Residual norm vs iteration number for PCG methods, $maxit = 750$

Method	**Iterations**	**Final Residual**	**Computational Time**
Matlab PCG without preconditioning	183	1.9591×10^{-7}	$0.054s$
Matlab PCG IC(0)	78	1.5293×10^{-7}	$0.063s$
My PCG implementation	720	9.6833×10^{-9}	$0.351s$

Table 2: Results of PCG methods, $maxit = 750$

My implementation is slower to converge but produces better final residual values. This can be explained by the simpler implementation relative to Matlab's built-in method. The residuals difference can derive, however, from different normalization techniques.

Question 2

The spectral condition number of A is

$$\kappa(A) = \frac{\lambda_{max}(A)}{\lambda_{min}(A)}. \tag{1}$$

In Matlab, I used the `condest(A)` function to estimate the condition number of a sparse matrix `A`.

The following table shows every data point needed to understand what the number of iterations of the CG method depends on.

n_x	h	$\kappa(A)$	$\sqrt{\kappa(A)}$	CG	$PCG(0)$	$PCG(10^{-2})$	$PCG(10^{-3})$
102	9.9009×10^{-3}	6.0107×10^{3}	77.5288	283	87	45	17
202	4.9751×10^{-3}	2.3810×10^{4}	154.3039	532	159	78	30
402	2.4938×10^{-3}	9.4770×10^{4}	307.8473	948	282	137	53
802	1.2484×10^{-3}	3.7814×10^{5}	614.9304	1792	533	258	97

Table 3: Iterations of PCG methods for each value of n_x and respective values of h and $\kappa(A)$

One can note from the table the inverse proportionality of the number of iterations on $h = \frac{1}{n} = \frac{1}{(nx-1)}$. The number of iterations is halved when n_x approximately doubles.

Question 3

Theoretical proof

One can estimate the number of iterations needed for convergence of a CG method as

$$k \approx \frac{log10}{2} p(\sqrt{\kappa} + 1), \tag{2}$$

with $tol = 10^{-p}$, so, for this example,

$$k \approx \frac{log10}{2} 8(\sqrt{10^3} + 1) \tag{3}$$

$$k \approx 300.44. \tag{4}$$

This result is for a non-preconditioned method.
For PCG we can find the optimal polynomial

$$P_k(t) = \left(1 - \frac{t}{200}\right)\left(1 - \frac{t}{400}\right)\left(1 - \frac{t}{600}\right)\left(1 - \frac{t}{800}\right)\left(1 - \frac{t}{1000}\right)\hat{T}_{k-4}(t) \tag{5}$$

This is only relevant for $t \in \lambda(A) = \{1, 200, 400, 600, 800, 1000\}$. The error reduction is then given by the maximum value of $\hat{T}_{k-4}(t)$, $k = 4, 5, \ldots$. So the method converges when $\frac{1}{max(\hat{T}_{k-4}(t))||b||} < tol$, with $tol = 10^{-8}$ in this case.
These calculations produced the following results:

Iteration (k)	**Expected Error**
4	1.2×10^{-3}
5	6.2×10^{-7}
6	3.1×10^{-10}
...	...

Table 4: Results for each value of implementation, no preconditioning

It is hereby proven that the method converges after 6 iterations.

Testing my implementation

When using the Choledsky precontionier with no fill-in, I didn't get the expected results. Both Matlab's and my implementation converged in only one iteration.

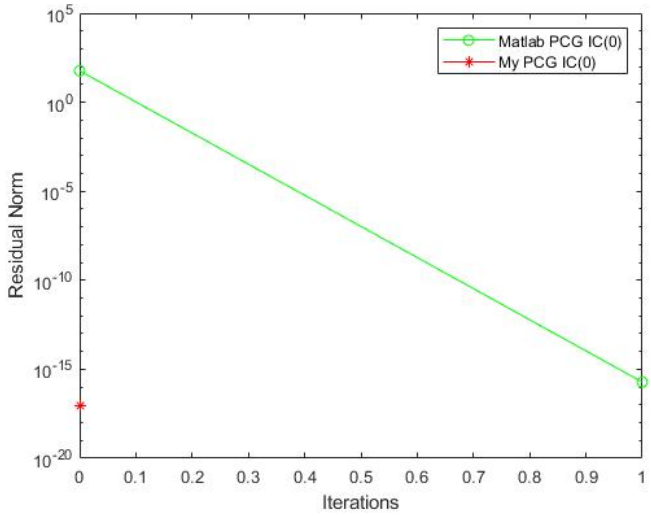

Figure 3: Residual norm vs iteration number for PCG methods with $IC(0)$ preconditioner

Method	**Iterations**	**Final Residual**	**Computational Time**
Matlab PCG	1	5.6843×10^{-14}	$0.058s$
My PCG	1	0	$0.020s$

Table 5: Results for each preconditioned PCG implementation

These results can be due to the optimized preconditioner usage in this methods, making this example solved in almost only preprocessing.

Due to the bad results, I tried to remove preconditioning from my implementation by setting L as the identity matrix, `L = speye(size(L))`.

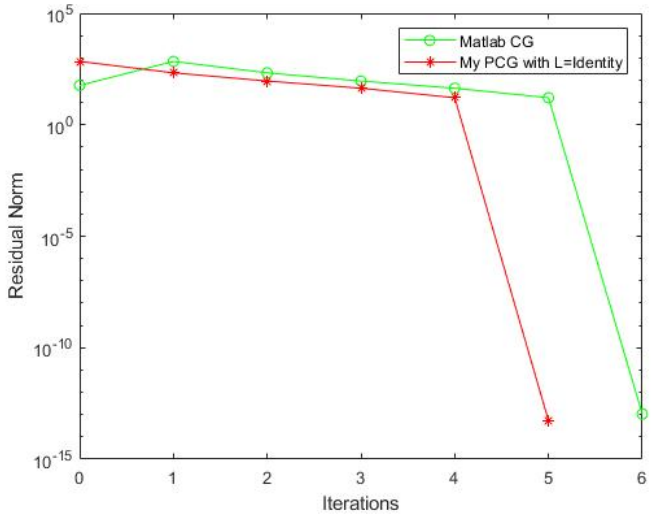

Figure 4: Residual norm vs iteration number for PCG methods without preconditioning

Method	Iterations	Final Residual	Computational Time
Matlab CG	6	9.2128×10^{-14}	$0.021s$
My PCG	5	1.2744×10^{-13}	$0.012s$

Table 6: Results for each value of implementation, no preconditioning

These results show the theoretical calculations, my implementation is still better than expected.

Question 4

Solving with PCG a linear system with `A` as the coefficient SPD matrix and `b` corresponding to a random vector solution, I got the following results.

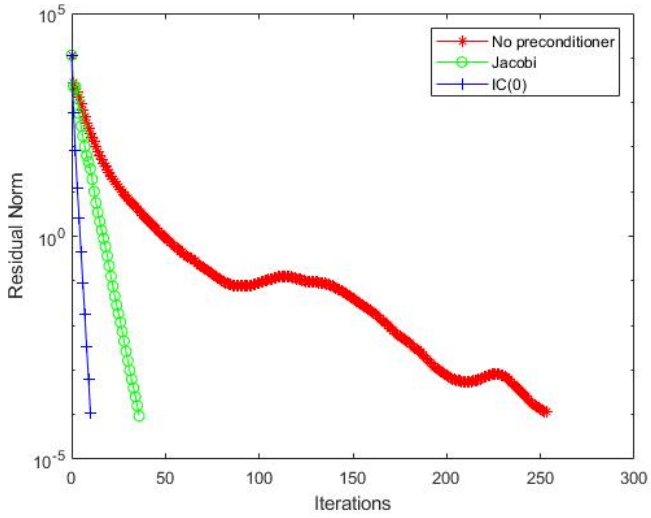

Figure 5: Residual norm vs iteration number for PCG methods without preconditioning

Preconditioner	Iterations	Final Residual	Computational Time
None	253	1.1367×10^{-4}	$0.254s$
Jacobi	36	9.3198×10^{-5}	$0.053s$
IC(0)	10	1.1155×10^{-4}	$0.046s$

Table 7: Results for each preconditioner

There is a very clear improvement when using preconditioning. It is also noticeable the superior characteristics of the incomplete Choledsky preconditioner relative to Jacobi.

Question 5

Solving the linear system based on the matrix `mat13041.rig` with the GMRES method, I got the following results.

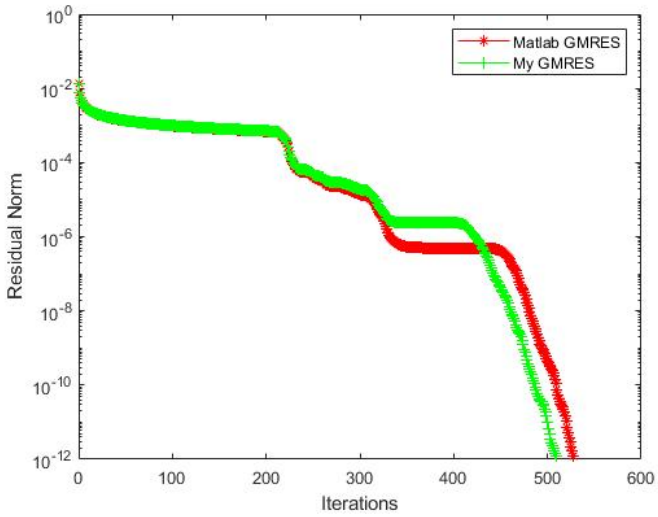

Figure 6: Residual norm vs iteration number for GMRES methods

Method	**Iterations**	**Final Residual**	**Computational Time**
Matlab GMRES	527	1.2073×10^{-12}	$9.097s$
My GMRES	509	1.2231×10^{-12}	$10.032s$

Table 8: Results for each GMRES implementation

These results show that the methods have very similar convergence characteristics. My implementation has a smaller number of iterations but the other results are slightly worse than the ones achieved with Matlab's implementation.

Question 6

Solving the same linear system based on the matrix `mat13041.rig` with the preconditioned GMRES method, I got the following results.

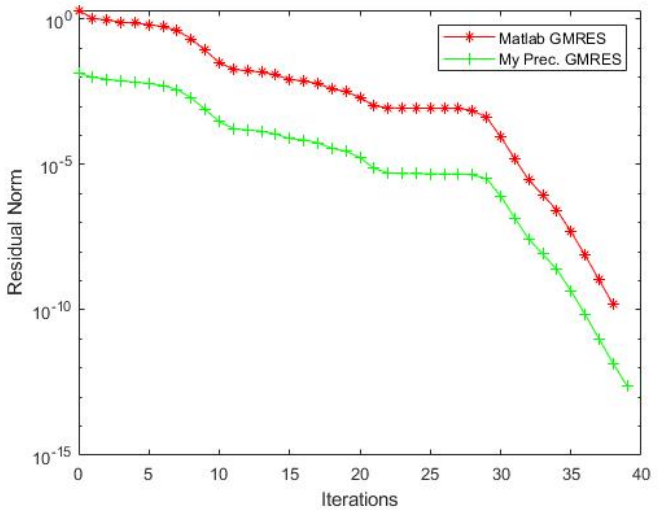

Figure 7: Residual norm vs iteration number for preconditioned GMRES methods

Method	Iterations	Final Residual	True Residual	Computational Time
Matlab GMRES	38	1.5592×10^{-10}	4.5350×10^{-13}	$0.121s$
My GMRES	39	2.3797×10^{-13}	7.1893×10^{-14}	$4.943s$

Table 9: Results for each preconditioned GMRES implementation

There are clear differences in the residuals and computational time values. My implementation is 40 times slower but produces a true residual 5 times smaller. This can be due to the simplicity of my implementation and consequent unnecessary calculations.

Solving the linear system in 3.(b) produced the same unexpected results as in that question. When using the Choledsky precontionier with no fill in both Matlab's and my implementation converged in only one iteration.

Method	Iterations	Final Residual	Computational Time
GMRES	1	3.8481×10^{-16}	$0.027s$
My PCG	1	1.7554×10^{-17}	$0.003s$

Table 10: Iterations for each value of nx

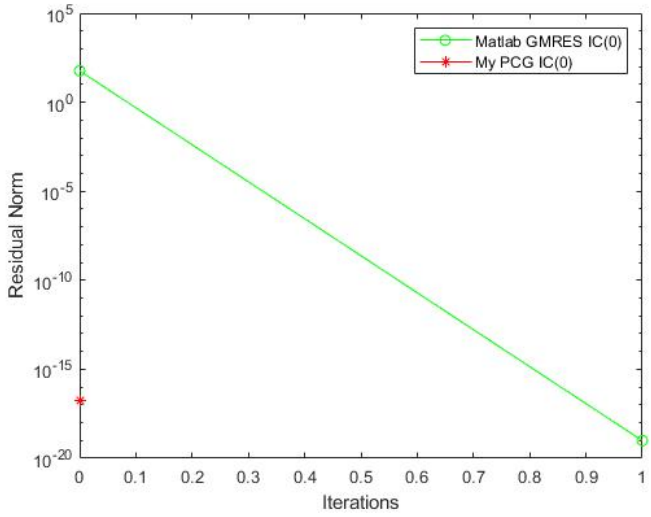

Figure 8: Residual norm vs iteration number for GMRES methods

I then removed preconditioning from my implementation by setting L as the identity matrix, `L = speye(size(L))`.

Method	Iterations	Final Residual	Computational Time
GMRES	6	3.0413×10^{-14}	$0.102s$
My PCG	5	1.0468×10^{-13}	$0.012s$

Table 11: Results for each value of implementation, no preconditioning

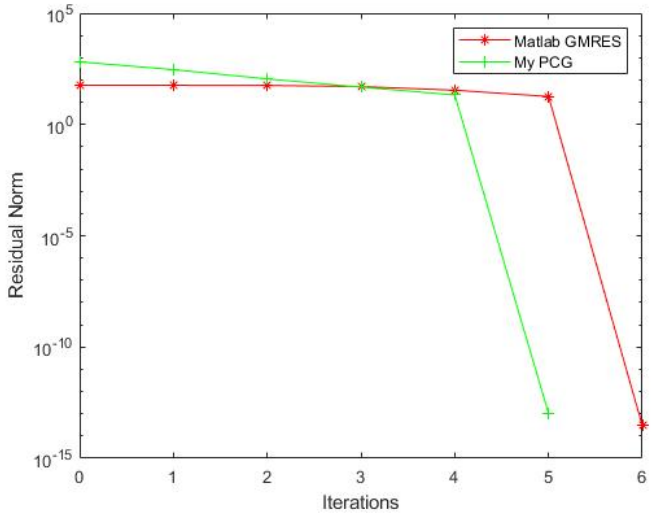

Figure 9: Residual norm vs iteration number for GMRES methods without preconditioning

As in 3.(b), these results show the theoretical calculations, my implementation is still better than expected.

Question 7

Using the same linear system as in the previous question, we will now study the effect of the `restart` value.

`restart`	**Iterations**	**Final Residual**	**Computational Time**
10	1149	1.9901×10^{-12}	$1.735s$
20	739	1.9741×10^{-12}	$1.443s$
30	88	1.3800×10^{-12}	$0.242s$
50	41	9.9416×10^{-13}	$0.135s$

Table 12: Results for each value of `restart`

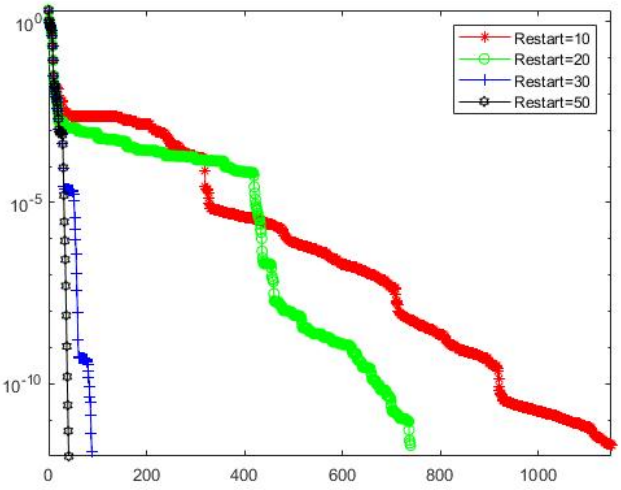

Figure 10: Residual norm vs iteration number for each value of `restart`

One can notice a clear improvement in convergence with the increase of the `restart` value. A larger `restart` value implies the method restarts less times, this is clearly better but implies a larger memory usage and computational cost. For `restart`= 50 the method did not restart and the results are optimal. This shows that this example does not benefit from restarting.

Question 8

Solving the linear system based on the matrix `ML_laplace.mtx` with Matlab's GMRES method, `maxit=550`, `tol=1e-8`, I got the following results.

`droptol`	**Iterations**	**Prec. Time**	**Solving Time**	**Total Time**	**Final Residual**	ρ
2×10^{-2}	1316	$37.92s$	$44.53s$	$82.45s$	5.2065×10^{-7}	0.4537
1×10^{-2}	4444	$40.55s$	$22.86s$	$63.42s$	5.7213×10^{-7}	0.5807
3×10^{-3}	150	$51.78s$	$7.39s$	$59.17s$	6.4998×10^{-7}	0.9401
1×10^{-3}	67	$47.74s$	$3.63s$	$51.37s$	8.5337×10^{-7}	1.4544
1×10^{-4}	26	$43.30s$	$2.30s$	$45.60s$	9.1517×10^{-7}	3.5140
1×10^{-5}	12	$96.73s$	$2.69s$	$99.42s$	9.4359×10^{-7}	9.0720

Table 13: Results for each value of `droptol`

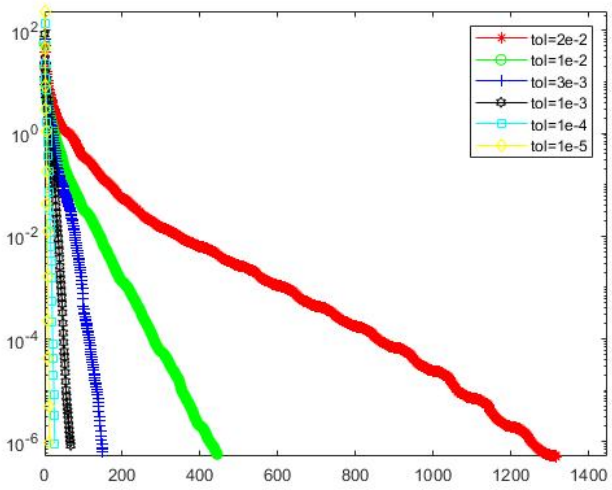

Figure 11: Residual norm vs iteration number for each value of `droptol`

There is a clear reduction in the number of iterations with the decrease of `droptol` which is due to the more complete factorization. This has a higher computational cost to compute (larger Prec. Time) but a smaller solving time. Despite of this, the smaller `droptol` produces a reduction in total computational time until `droptol`= 1×10^{-5}. For `droptol`= 1×10^{-5} the Prec. time increases significantly and the solving time stagnates relative to 1×10^{-4}. The final residual does not have a noticeable change with the reduction of `droptol`. The density (ρ) has a very clear increase with the reduction of `droptol`.