Optimization methods – Homework 2

Claudio Maggioni

April 11, 2021

1 Exercise 1

1.1 Implement the matrix A and the vector b, for the moment, without taking into consideration the boundary conditions. As you can see, the matrix A is not symmetric. Does an energy function of the problem exist? Consider $N = 4$ and show your answer, explaining why it can or cannot exist.

The implementation of a function that generates A w.r.t. the parameter N can be found in the MATLAB script `main.m` under Section 1.1.

The matrix A and the vector b appear in the following form:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 & \dots & 0 \\ -1 & 2 & -1 & 0 & \dots & 0 \\ 0 & -1 & 2 & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \dots & 1 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ h^2 \\ h^2 \\ \vdots \\ 0 \end{bmatrix}$$

For $N = 4$, we have the following A and b:

$$A = \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 \\ 0 & -1 & 2 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad b = \begin{bmatrix} 0 \\ \frac{1}{9} \\ \frac{1}{9} \\ 0 \end{bmatrix}$$

In order to solve the minimization problem we need to minimize the energy function:

$$\phi(x) = \frac{1}{2} x^T A x - b^T x$$

Computing the gradient of the minimizer, and considering that A is clearly not symmetric as shown above, we find:

$$\Delta\phi(x) = \frac{1}{2} A^T x + \frac{1}{2} A x - b$$

If A would be symmetric, $\Delta\phi(x)$ is equal to $Ax - b$ and therefore the semantic equivalence between this energy function and the solution of $Ax = b$ is straightforward. However, if A is not symmetric then this does not hold and an enegry function therefore does not exist.

1.2 Once the new matrix has been derived, write the energy function related to the new problem and the corresponding gradient and Hessian.

As by the definitions of A and b given in the assignment, we already enforce $x_1 = x_n = 0$, since the first and the last term of the matrix-vector product Ax are x_1 and x_n respectively and the system of equations represented by $Ax = b$ would indeed include the equalities $x_1 = b_1 = 0$ and $x_n = b_n = 0$. Therefore, we can simply alter the matrix A without any need to perform any other transformation in the system.

We therefore define $\overline{A}$ as a copy of A where $\overline{A}_{2,1} = \overline{A}_{n-1,n} = 0$.

The objective function then becomes $\phi(x) = \frac{1}{2}x^T\overline{A}x - b^Tx$. And since the objective is a standard quadratic form, the gradient is $\overline{A}x - b$ while the Hessian is $\overline{A}$.

1.3 Write the Conjugate Gradient algorithm in the pdf and implement it Matlab code in a function called `CGSolve`.

The Conjugate Gradient algorithm is the following:

Algorithm 1: Conjugate Gradient algorithm

Set $r_0 \leftarrow Ax_0 - b$, $p_0 \leftarrow r_0$, $k \leftarrow 0$;
while $r_k \neq 0$ **do**
  $\alpha_k \leftarrow \frac{r_k^T r_k}{p_k^T A p_k}$;
  $x_{k+1} \leftarrow x_k + \alpha_k p_k$;
  $r_{k+1} \leftarrow r_k + \alpha_k A p_k$;
  $\beta_{k+1} \leftarrow \frac{r_{k+1}^T r_{k+1}}{r_k^T r_k}$;
  $p_{k+1} \leftarrow -r_{k+1} + \beta_{k+1} p_k$;
  $k \leftarrow k + 1$;
end

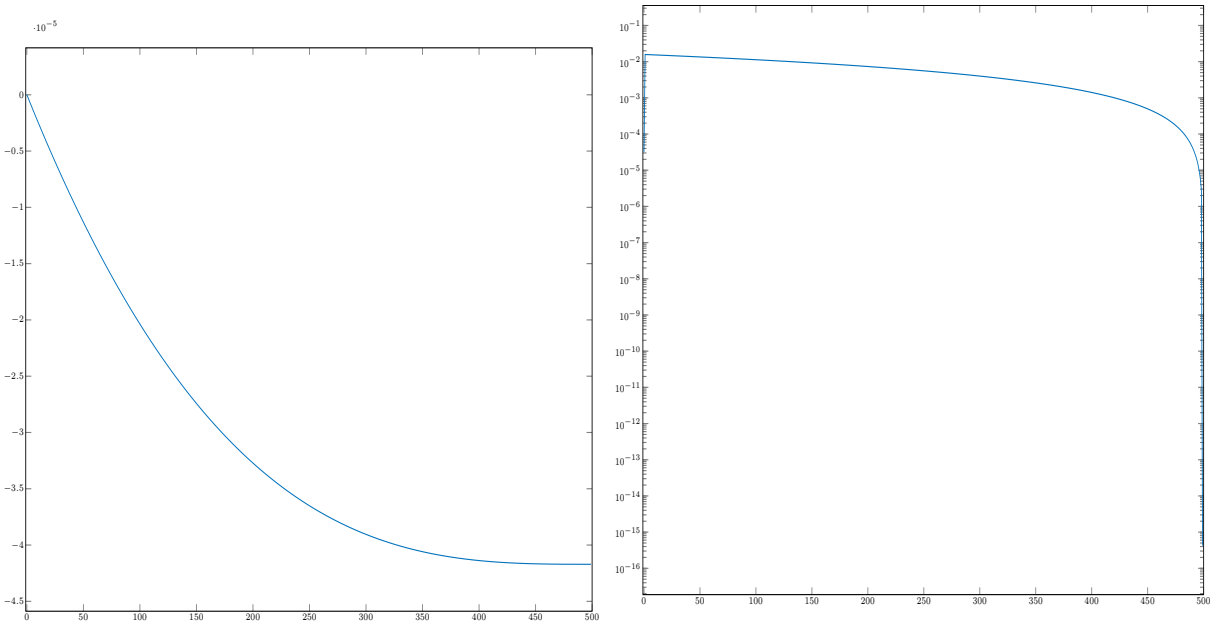
The MATLAB solution of this task can be found in Section 1.3 of the script `main.m` under the function `CGSolve`.

1.4 Solve the Poisson problem.

The solution of this task can be found in Section 1.4 of the script `main.m`. Due to space constraints, the R^{1000} column vector for the solution of x is not shown here but can be easily derived by running the script and checking the variable `x` after the script execution.

1.5 Plot the value of energy function and the norm of the gradient (here, use semilogy) as functions of the iterations.

The code to generate the plots below can be found in Section 1.5 of the script `main.m`.



(a) Objective function values w.r.t. iteration number

(b) Norm of the gradient w.r.t. iteration number (y-axis is log scaled)

Figure 1: Plots for Exercise 1.4.

1.6 Finally, explain why the Conjugate Gradient method is a Krylov subspace method.

We can say that the Conjugate Gradient method is a Krylov subspace method since we can say that both all residuals r_i and all search directions p_i are contained in the span of 0 to k repeated applications of matrix transformer A onto the initial search direction p_0. Broadly speaking, this property is directly connected with the fact that the CG methods performs up to n steps exactly for a n by n matrix, walking steps that are all A-orthogonal with each other (i.e. for all couples of search steps p_i, p_j where $i \neq j$, $\langle Ap_i, p_j \rangle = 0$). To sum up our claim, we can say CG is indeed a Krylov subspace method because:

$$\text{span}\{r_0, r_1, \ldots, r_n\} = \text{span}\{r_0, Ar_0, \ldots, A^k r_0\}$$
$$\text{span}\{p_0, p_1, \ldots, p_n\} = \text{span}\{r_0, Ar_0, \ldots, A^k r_0\}$$

These statements have been already proven in class and the proof can be found in Theorem 5.3 of Nocedal's book.

2 Exercise 2

Consider the linear system $Ax = b$, where the matrix A is constructed in three different ways:

- $A_1 = \text{diag}([1{:}10])$
- $A_2 = \text{diag}(\text{ones}(1{,}10))$
- $A_3 = \text{diag}([1,\ 1,\ 1,\ 3,\ 4,\ 5,\ 5,\ 5,\ 10,\ 10])$
- $A_4 = \text{diag}([1.1,\ 1.2,\ 1.3,\ 1.4,\ 1.5,\ 1.6,\ 1.7,\ 1.8,\ 1.9,\ 2.0])$

2.1 How many distinct eigenvalues has each matrix?

Each matrix has a distinct number of eigenvalues equal to the number of distinct elements on its diagonal. So, in order, each A has respectively 10, 1, 5, and 10 distinct eigenvalues.

2.2 Construct a right-hand side b =rand(10,1) and apply the Conjugate Gradient method to solve the system for each A.

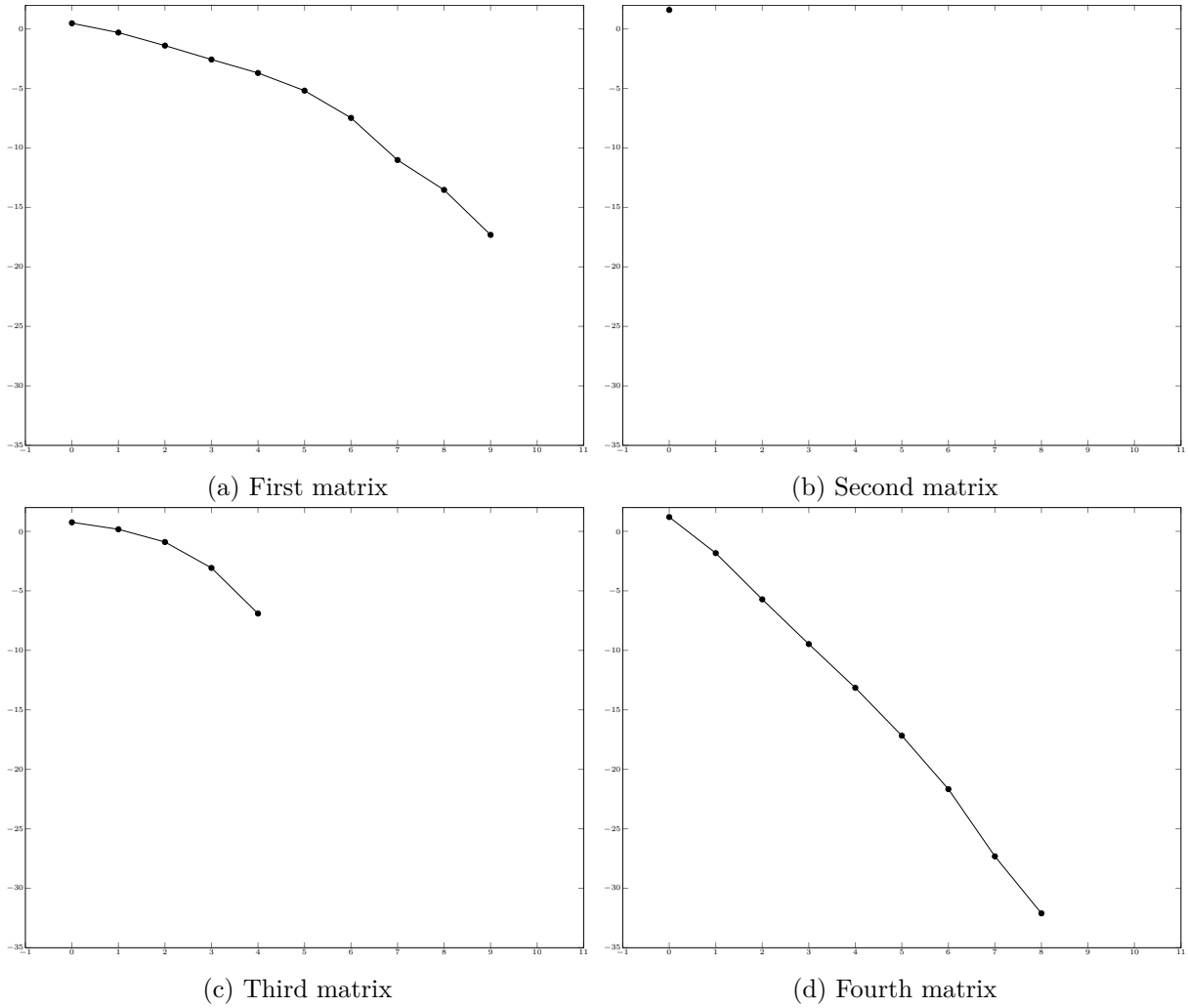
The solution of this task can be found in section 2.2 of the `main.m` MATLAB script. Below are the chosen b vector (which is unchanged between executions due to fixing the random generator seed in the script) and the solutions of x for each matrix respectively.

$$b = \begin{bmatrix} 0.814723686393179 \\ 0.905791937075619 \\ 0.126986816293506 \\ 0.913375856139019 \\ 0.632359246225410 \\ 0.0975404049994095 \\ 0.278498218867048 \\ 0.546881519204984 \\ 0.957506835434298 \\ 0.964888535199277 \end{bmatrix} \quad x_1 = \begin{bmatrix} 0.814723686393179 \\ 0.452895968537810 \\ 0.0423289387645020 \\ 0.228343964034755 \\ 0.126471849245082 \\ 0.0162567341665680 \\ 0.0397854598381500 \\ 0.0683601899006230 \\ 0.106389648381589 \\ 0.0964888535199280 \end{bmatrix} \quad x_2 = \begin{bmatrix} 0.814723686393179 \\ 0.905791937075619 \\ 0.126986816293506 \\ 0.913375856139019 \\ 0.632359246225410 \\ 0.0975404049994095 \\ 0.278498218867048 \\ 0.546881519204984 \\ 0.957506835434298 \\ 0.964888535199277 \end{bmatrix}$$

$$x_3 = \begin{bmatrix} 0.814723686393179 \\ 0.905791937075619 \\ 0.126986816293506 \\ 0.304458618713007 \\ 0.158089811556352 \\ 0.0195080809998819 \\ 0.0556996437734097 \\ 0.109376303840997 \\ 0.0957506835434296 \\ 0.0964888535199280 \end{bmatrix} \quad x_4 = \begin{bmatrix} 0.740657896716011 \\ 0.754826614267239 \\ 0.0976821653904972 \\ 0.652411326111541 \\ 0.421572830214428 \\ 0.0609627567866090 \\ 0.163822480881755 \\ 0.303823066390868 \\ 0.503950965995613 \\ 0.482444267601989 \end{bmatrix}$$

2.3 Compute the logarithm energy norm of the error for each matrix and plot it with respect to the number of iteration.

The code to generate the plots below and to compute the logarithm energy norm of the error can be found in section 2.3 of the `main.m` MATLAB script.



(a) First matrix

(b) Second matrix

(c) Third matrix

(d) Fourth matrix

Figure 2: Plots of logarithm energy norm of the error per iteration. Minus infinity logarithms not shown in the plot.

2.4 Comment on the convergence of the method for the different matrices. What can you say observing the number of iterations obtained and the number of clusters of the eigenvalues of the related matrix?

The method converges quickly for each matrix. The fastest convergence surely happens for $A2$, which is the identity matrix and therefore makes the $Ax = b$ problem trivial.

For all the other matrices, we observe the energy norm of the error decreasing exponentially as the iterations increase, eventually reaching 0 for the cases where the method converges exactly (namely on matrices $A1$ and $A3$).

Other than for the fourth matrix, the number of iterations is exactly equal to the number of distinct eigenvalues for the matrix. That exception on the fourth matrix is simply due to the tolerance termination condition holding true for an earlier iteration, i.e. we terminate early since we find an approximation of x with residual norm below 10^{-8}.