Università della Svizzera italiana
Institute of Computing CI

Numerical Computing **2024**

Student: Matteo Ghilardini **Discussed with:** Sasha Toscano

Solution for Project 4 **Due date:** Wednesday, 27 November 2024, 11:59 PM

Project Environment

PC	MacBook Pro 2017
OS	macOS Monterey
CPU	Intel Core i7
RAM	16 GB

1. General Questions [10 points]

1. **What is the size of the matrix A?**
The matrix A indicates the transformation matrix that in our case, produces the blurring effect. The size of the matrix A is $n^2 \times n^2$ where n is the number of pixels per side of the image. So we can say that $A \in \mathbb{R}^{n^2 \times n^2}$.

2. **How many diagonal bands does A have?**
The matrix A is a d^2-banded matrix, where $d \ll n$. This means that A will be a matrix with non-zero values distributed over d^2 diagonals; so A will have d^2 diagonal bands (with $d \ll n$).

3. **What is the length of the vectorized blurred image b?** The length of the vectorized blurred image B, so the length of vector b, will be equal to the total number of pixels of the image. So for an image of size $n \times n$, the length of vector b will be n^2.

2. Properties of A [10 points]

1. **If A is not symmetric, how would this affect $\tilde{A}$?**
Also if A is not symmetric, $\tilde{A}$ will still turn out to be symmetric since $\tilde{A}$ is defined as $A^T A$ and the multiplication of a matrix with its transpose always result in a symmetric matrix; so also if A is not symmetric, $\tilde{A}$ will be.
However, also if the symmetry of A don't affect the symmetry of $\tilde{A}$, it can affect some of its properties, like the eigenvalues.
Indeed if A is not symmetric, $\tilde{A}$ might have different eigenvalues compared to A.

2. **Show why solving $Ax = b$ for x is equivalent to minimizing $\frac{1}{2}x^T Ax - b^T x$ over x, assuming that A is symmetric positive-definite.**
To show this equality we have to minimize the function $\frac{1}{2}x^T Ax - b^T x$, and we do so in two steps separating the two terms of the function:

- derive $\frac{1}{2}x^T Ax$ by x resulting in Ax
- derive $-b^T x$ by x resulting in $-b$

Putting together this two terms we obtain $Ax - b$, and the minimum of the function is when is stops growing, and this is when $Ax - b = 0$, so the same as $Ax = b$.

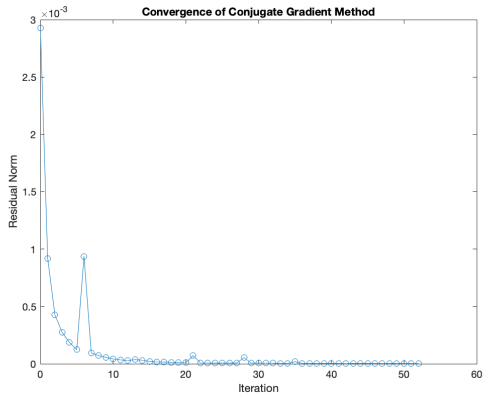
With this steps we have shown that minimizing the function $\frac{1}{2}x^T Ax - b^T x$ we obtain exactly $Ax = b$, so the two resolutions are equivalents.

3. Conjugate Gradient [30 points]

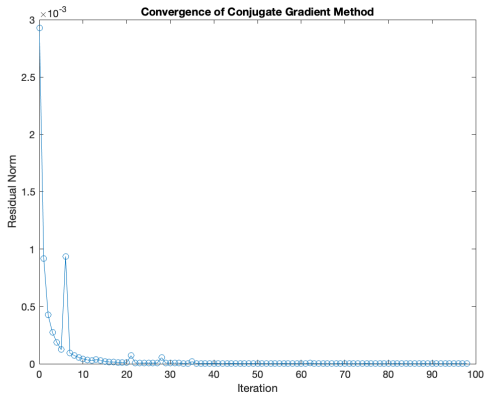
1. To write the function[1] for the conjugate gradient solver `[x,rvec]=myCG(A,b,x0,max itr,tol)` I followed strictly the provided pseudo algorithm given in the document **Project 4 – Conjugate Gradient and Image Deblurring**.

   With respect to the given pseudo-algorithm I've simply added an `if` condition to check for convergence. And I compute the relative residual norm according to the formula of the residual norm ($||r||_2 = \sqrt{\langle r, r \rangle}$) devided by the norm of b resulting in the formula $\sqrt{\langle r, r \rangle}/norm(b)$.

2. To validate my solution of the function `[x,rvec]=myCG(A,b,x0,max itr,tol)`, I've created a simple script[2] that loads the data from the given files (`test_data/A_test.mat` and `test_data/b_test.mat`), validate the sizes (since the length of vector b must be equal to the number of rows in A).
   After this, initialize all variables that must be passed to the function; consider that since the function should *produce the exact solution after a finite number of iterations, which is not larger than the size of the matrix* I set the max number of iterations as `max_itr = length(A)-1;`
   To conclude, I call the actual function `myCG` and plot the resulting data as the number of the iteration on the x-axis, an the residual norm for each of this iterations on the y-axis. This produce the following plots:



(a) myCG with tolerance `tol = 1e-6`



(b) myCG with tolerance `tol = 1e-7`

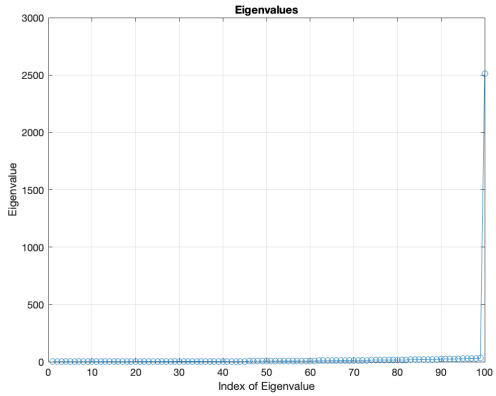
Figure 1: Convergence of Conjugate Gradient Method

We see that in the graph 1a the convergence under the tolerance of `tol = 1e-6` is reached after 53 iterations reaching as relative residual norm `8.79e-07`.
In the second graph 1b, after making the tolerance 10 times "smaller" than the previous one (so `tol = 1e-7`) we notice that the convergence is not reached after the max number of iterations (100); in this case the final relative residual norm is `4.7332e-07`

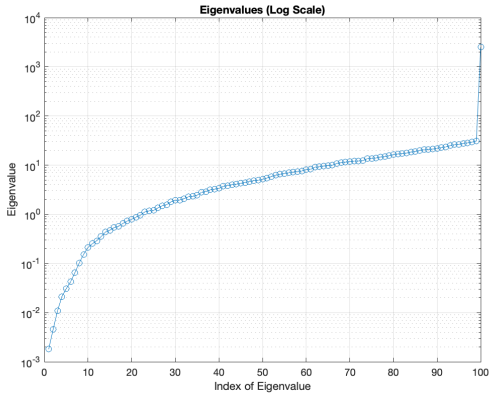
3. In a script[3], I start extracting A from A_test and then simply compute the eigenvalues of A

[1] The code is in the file `myCG.m`
[2] The code is in the file `validateMyCG.m`
[3] The code is in the file `plotEigs.m`

using the command `eig(A)` just before plotting them.



(a) Linear scale



(b) Logarithmic Scale

Figure 2: Eigenvalues of matrix *A_test*

By plotting simply the eigenvalues of the matrix A with a linear scale (figure 2a) we obtain a meaningless plot since many of the eigenvalues are very small, and the last eigenvalue of the matrix is much bigger; this makes plot quite unreadable.

Otherwise, if we plot the eigenvalues of A with a logarithmic scale (using command `semilogy(eigvals, '-o')`) we obtain a plot (figure 2b) that shows us the magnitude of the eigenvalues, and we can notice that is growing accordingly with the index of the eigenvalue.

Anyways, these two graphs (especially the logaritmic one 2b) shows us that there is a big difference of magnitude between the smallest and the largest eigenvalues of A, and this allows us to estimate that the condition number of A will be very large since is computed as $k(A) = \frac{\sigma_{max}}{\sigma_{min}}$

After having plotted the eigenvalues, as requested, I print the condition number of A computed using the command `cond(A)` and result to be `1.37e+06`. Since `cond(A)` is very large, we can confirm that A is **ill-conditioned**, and this will result into a slow convergence of the Conjugate Gradient method.

4. As we can see in the plotted graphs 1, the answer to this question is no.

   The CG algorithm minimizes the error globally in the A-norm (and so the residual will decrease) but does not guarantee that the residual decreases monotonically.

   We can have some outliers (like can be seen well in figure 1a at iterations number 6, or 21,...) that, due to ill-conditioning of the matrix or numerical errors related to the rounding of float values, can further disrupt monotonicity.

4. Deblurring problem [35 points]

1. The provided code[4], shows the blurred image 3 that we will deblur using both the implementation of the Conjugate Gradient Method (`myCG` and `pcg`). The image is the following:

   To complete the script I've separated the implementation into 4 different blocks of code.

   In the first block, I simply initialize all the variables and constants that i will use later. The code is quite intuitive but I want to mention (since it created problems for me during implementation) that i need to compute $\tilde{A}$ and $\tilde{b}$ in order to use it in both Conjugate Gradient Methods (mine and the one provided by Matlab) since we have to use positive-definite augmented transformation matrix, and this is achieved by applying the formula $A^T Ax = A^T b \longrightarrow \tilde{A}x = \tilde{b}$.

[4]The code is in the file `code_template.m`

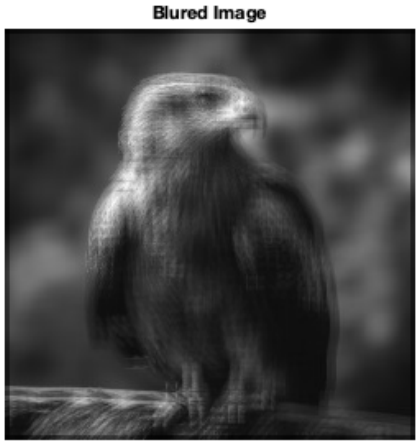
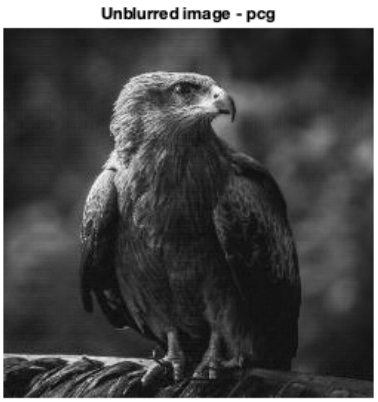


Figure 3: Blurred Image

The second and third blocks of code are very similar; the main difference is that one uses `myCG` and the other uses the Matlab function `pcg`.
In the case of `pcg` we use preconditions, that consist in computing the Cholesky factor of $\tilde{A}$. The first step is to call the Conjugate Gradient function (`myCG` or `pcg`) on $\tilde{A}$ and $\tilde{b}$.
Since we want to visualize also the relative residual norm (`rvec`) for both functions, `pcg` must be called like this:
`[x,∼,∼,∼,rvec] = pcg(A_tilde, b_tilde, tol, max_iter, L, L');` where L and L' specifies factors of the preconditioner matrix
The result of the function is a vectorized form of the deblurred image, so using the function `reshape(...)` we convert it again into a 2D image matrix. Since we have computed b as `b=B'; b=b(:);` (so row-wise) to get the actual 2D image, we need the transpose of the one we have obtained from `reshape(...)`. After this, the deblurred image is shown for both functions:

(a) Unblurred image using `myCG`



(b) Unblurred image using `pcg`

Figure 4: Unblurred images

In the last block of code, I simply plot in a logarithmic scale the residual norms obtained using both functions (`myCG` and `pcg`) to compare it:

From the plot 5 we note that the function `pcg` converges considerably faster than `myCG` and also requires far fewer iterations.

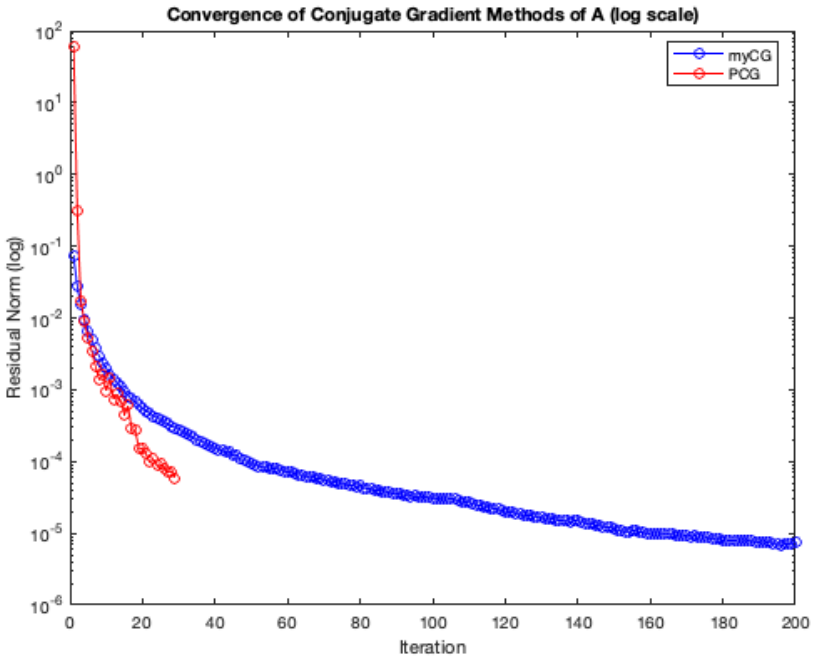


Figure 5: Compared convergence of `myCG` and `pcg`

We also note that `myCG` does not converge after the maximum number of iterations (`max_iter = 200`).
The rapidity of `pcg` may be attributable to the use of precondition showing how much this practice can improve the performance of the algorithm.

However, when analyzing the final result of the two images 4 they appear visually the same, so both implementations produces a satisfactory result.

2. The added computational cost of `pcg` is worth when we have a system that is ill-conditioned (as in our case) and the condition number of the system is large. Since the convergence of the CG is negatively influenced by a large condition number, it is worth to add a computational cost to the algorithm in order to mitigate the slow convergence.

   Another case where we can consider the added computational cost of the `pcg` function is when we have to unblur multiple images with the same blurring operator. This allows us to compute the precondition only one time and then use it for all the images dividing the computational cost over the number of unblurred images.

   In any case, from our example, we can see in figure 5 that using `pcg`, the convergence is around iteration 29, otherwise using `myCG`, convergence is not reached even after 200 iterations; this means that convergence of `pcg` is over 6 times faster than `myCG`.
   So it is easy to see the fact that even despite the higher computational cost, `pcg` is much faster and therefore this cost is quickly amortized.