Università della Svizzera italiana
Institute of Computing CI

Numerical Computing – Fall Semester 2024
Lecturer: Dr. Edoardo Vecchi
Assistants: Gianmarco De Vita, Lorenzo Varese

Project 4 – Conjugate Gradient and Image Deblurring

Due date: Wednesday, 27 November 2024, 11:59 PM

The purpose of this mini-project is to gain insight into the theoretical and numerical properties of the Conjugate Gradient method. Here we use this method in an image processing application with the goal of deblurring an image given the exact (noise-free) blurred image and the original transformation matrix. Note that the "noise-free" simplification is essential for us to solve this problem in the scope of this mini-project[1].

1. Formulation of the image deblurring problem

Suppose we have a blurred image B which we want to refine (or deblur) and we know the transformation that lead to this blurred image. Then how can we efficiently reconstruct the original image? Let $X \in \mathbb{R}^{n\times n}$ be the original square, grayscale image matrix, where each matrix entry corresponds to one pixel value. Hence X corresponds to an image with n pixels per side. We can convert the original image matrix X into a column vector $x \in \mathbb{R}^{n^2}$, where the entries of X get stacked into a vector, either row- or column-wise. This process is referred to as *vectorization*[2]. If we now define as b the vectorized form of the blurred image B, we can write the following system of equations:

$$Ax = b, \tag{1}$$

Here, $A \in \mathbb{R}^{n^2\times n^2}$ indicates the transformation matrix coming from the the repeated application of what is referred to as the "image kernel" which in our case produces a blurring effect. An excellent visual explanation of the kernel operation can be found at [4]. Our objective will be to solve the linear system in Eq.[1] for the original vectorized image x, where the transformation matrix A and the blurred image B (and hence b) are known. By solving the system, we can recover the original image, getting rid of the blurring effect applied on top of it.

1. Visualization and vectorization

The image in consideration is black and white and the corresponding image matrix can be visualized with the command `imshow` (see Matlab documentation). We can vectorize the blurred image matrix with the command `B(:)`. The vectorization ordering can be arbitrary, by column, row or whatever fits best with the data access pattern; however, it must correspond with A. In our case we require row by row vectorize (Matlab default is by column). This can be achieved using the command `B=B'; b=B(:);`.

2. Transformation matrix

A blurred image pixel is the weighted average of the surrounding pixels. These weights are defined by the kernel matrix $K \in \mathbb{R}^{d\times d}$. The much bigger transformation matrix A is constructed from the kernel matrix, such that the non-zero elements of each row of A correspond to the values of K. The detailed construction of the transformation matrix is omitted for brevity, see [4]. The following attributes should be noted:

Borders: The transformation matrix ignores the elements outside of borders of the matrix, $(\max(i, j) > n)$.

[1] For the interested reader, a Google search of *"image deblurring* or *inverse problem image processing"* may be enlightening
[2] For more details on vectorization check, e.g., `https://en.wikipedia.org/wiki/Vectorization_(mathematics)`, while for its importance in Matlab check `https://ch.mathworks.com/help/matlab/matlab_prog/vectorization.html`.

Banded Matrix: A is a d^2-banded matrix, where $d \ll n$. This means that A is sparse, and thus one should use sparse data structures. Recall that A is of dimension $n^2 \times n^2$, defining a dense matrix of that size will not fit on your system. For example, if $n = 250$ we would need 31 GB (i.e., $8n^2 \cdot n^2$ Bytes) of memory. For this mini-project, you will use the provided transformation matrix `A.mat`, which is generated from the normalized blur image kernel

$$K = \frac{1}{605}\begin{bmatrix} 100 & 9 & 9 & 9 & 9 & 9 & 100 \\ 9 & 1 & 1 & 1 & 1 & 1 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 9 \\ 9 & 1 & 1 & 1 & 1 & 1 & 9 \\ 100 & 9 & 9 & 9 & 9 & 9 & 100 \end{bmatrix}, \tag{2}$$

and operates on a "row-vectorized" image matrix.

2. Conjugate gradient

The conjugate gradient method (CG) [1, 3] is widely used for solving systems of linear equations $Ax = b$, where A is both symmetric and positive-definite. The conjugate gradient method can theoretically be viewed as a direct method, as it produces the exact solution after a finite number of iterations, which is not larger than the size of the matrix, in the absence of round-off error. However, the conjugate gradient method is unstable with respect to even small perturbations, e.g., most directions are not in practice conjugate, and the exact solution is never obtained. Fortunately, the conjugate gradient method can be used as an iterative method as it provides monotonically improving approximations $\mathbf{x}_k$ to the exact solution, which may reach the required tolerance after a relatively small (compared to the problem size) number of iterations. The improvement is typically linear and its speed is determined by the condition number $\kappa(A)$ of the system matrix A: the larger $\kappa(A)$ is, the slower the improvement.
The CG algorithm starts from an initial guess x_0 and applies a series of matrix-vector and vector-vector operations until the desired tolerance is reached:

Algorithm 1 *
Conjugate Gradient Algorithm

$r \leftarrow b - Ax_0$
$d \leftarrow r$
$\rho_{\text{old}} \leftarrow \langle r, r \rangle$
for $i = 0, 1, \ldots$ **do**
  $s \leftarrow Ad_i$
  $\alpha \leftarrow \rho_{\text{old}} / \langle d, s \rangle$
  $x \leftarrow x + \alpha d$
  $r \leftarrow r - \alpha s$
  $\rho_{\text{new}} \leftarrow \langle r, r \rangle$
  $\beta \leftarrow \rho_{\text{new}} / \rho_{\text{old}}$
  $d \leftarrow r + \beta d$
  $\rho_{\text{old}} \leftarrow \rho_{new}$
end for

1. Symmetry and positivity

In our case A is symmetric and of full rank but not positive-definite. We can bypass this issue by solving the augmented system Eq. [3]. The pre-multiplication with A^T, results in the positive-definite augmented transformation matrix $\tilde{A}$.

$$\begin{aligned} A^T Ax &= A^T b \\ \tilde{A}x &= \tilde{b}. \end{aligned} \tag{3}$$

2. Condition number

An important attribute of a linear system $Ax = b$ is its condition number $\kappa(A)$. This is the relation of the sensitivity of the solution x, to changes in b. If small changes in b results in large changes in x, the system is called *ill-conditioned* and the condition number of the system is large. The convergence rate of CG is hindered when the system has a high condition number. Notice that the number of iterations needed to find the exact solution is not related to $\kappa(A)$, but rather equal to the number of unique eigenvalues. The condition number of a square matrix A is

$$\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}} \tag{4}$$

with σ indicating the singular values of A. For a real symmetric matrix, the singular values are equal to the absolute value of the eigenvalues: $\sigma = |\lambda|$.

In our case, matrix A is symmetric and real, and this implies that the condition number is given by the ratio of the magnitudes of the maximum and minimum eigenvalues. We can check an estimate of the condition number of A by using the Matlab command `condest` (or `cond` for a dense matrix). Notice that this command will take a long time on large matrices. Furthermore, the transformation matrix is ill-conditioned and - to make matters worse - we will be solving the augmented system in Eq.[3] with condition number $\kappa(A)^2 = \kappa(\tilde{A})$. These factors will unfortunately deteriorate the convergence rate of our deblurring program, but this problem can be partially mitigated through preconditioning, as explained in the next section.

3. Preconditioning

A symmetric positive-definite preconditioner P is selected such that $P^{-1}\tilde{A} \approx I$, that is P^{-1} approximates $\tilde{A}^{-1}$. We can decompose the preconditioner such that $P = LL^T$, where L is the Cholesky factor (see [2]). Our intention is to solve the preconditioned augmented system

$$P^{-1}\tilde{A}x = P^{-1}\tilde{b} \tag{5}$$

$$\underbrace{(L^{-1}\tilde{A}L^{-1})}_{\bar{\tilde{A}}}\underbrace{(Lx)}_{\bar{x}} = \underbrace{L^{-1}\tilde{b}}_{\bar{\tilde{b}}}. \tag{6}$$

This operation is done to both decrease the condition number $\kappa(\bar{\tilde{A}}) < \kappa(\tilde{A})$ and to decrease the range of the eigenvalues. The underlying caveat here is that the preconditioner should be computationally inexpensive to find.

To meet the desired properties of the preconditioner, a common approach is to use incomplete Cholesky (IC) factorization (see [1]). With IC, we only compute the Cholesky factorization of the non-zero elements of $\tilde{A}$. This cheap factorization of $\tilde{A}$ gives us the preconditioner $P = F^T F$, where F is the sparse IC factors. This routine can fail since the existence of F is not guaranteed (note that F is forced to be sparse). To amend this issue, a heuristic approach is used to apply a diagonal shift of P, enforcing positive-definiteness and making F computable. The detailed explanation of this routine is outside the scope of this assignment. We can use the Matlab `ichol` command for the IC factorization (please refer to the Matlab documentation for more information).

The assignment

Read this document and Section 7.4 of the textbook [1] in order to answer the following questions.

1 . General questions [10 points]

1. What is the size of the matrix A?

2. How many diagonal bands does A have?
3. What is the length of the vectorized blurred image b?

2. Properties of A [10 points]

1. If A is not symmetric, how would this affect $\tilde{A}$?
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.

3. Conjugate gradient [30 points]

1. Write a function for the conjugate gradient solver `[x,rvec]=myCG(A,b,x0,max_itr,tol)`, where `x` and `rvec` are, respectively, the solution value and a vector containing the residual at every iteration.
2. In order to validate your implementation, solve the system defined by `A_test.mat` and `b_test.mat`. Plot the convergence (residual vs iteration).
3. Plot the eigenvalues of `A_test.mat` and comment on the condition number and convergence rate.
4. Does the residual decrease monotonically? Why or why not?

4. Deblurring problem [35 points]

1. Solve the deblurring problem for the blurred image matrix `B.mat` and transformation matrix `A.mat` using your routine `myCG` and Matlab's preconditioned conjugate gradient `pcg`. As a preconditioner, use `ichol` to get the incomplete Cholesky factors and set routine type to `nofill` with $\alpha = 0.01$ for the diagonal shift (see Matlab documentation). Solve the system with both solvers using $max\_iter = 200$ $tol = 10^{-6}$. Plot the convergence (residual vs iteration) of each solver and display the original and final deblurred image. Comment on the results that you observe.

   Hint : Check the Matlab description of the pcg() routine to make sure that you compute the residual norm in the same way to be able to properly compare them.
2. When would `pcg` be worth the added computational cost? What about if you are deblurring lots of images with the same blur operator?

Quality of the code and of the report [15 Points]

The highest possible score for each project is 85 points and up to 15 additional points can be awarded based on the quality of your report and code (maximum possible grade: 100 points). Your report should be a coherent document, structured according to the template provided on iCorsi. If there are theoretical questions, provide a complete and detailed answer. All figures must have a caption and must be of sufficient quality (include them either as .eps or .pdf). If you made a particular choice in your implementation that might be out of the ordinary, clearly explain it in the report. The code you submit must be self-contained and executable, and must include the set-up for all the results that you obtained and listed in your report. It has to be readable and, if particularly complicated, well-commented.

Additional notes and submission details

Summarize your results and experiments for all exercises by writing an extended LaTeX report, by using the template provided on iCorsi. Submit your gzipped archive file (tar, zip, etc.) **on iCorsi strictly before the deadline** in the dedicated section and use the following standard naming: `project_4_lastname_firstname.zip` (or `tgz`). Submission by email or through any other channel will not be considered. Late submissions will not be graded and will result in a score of 0 points for that project. You are allowed to discuss all questions with anyone you like, but: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently. Please remember that plagiarism will result in a harsh penalization (0 points) for all involved parties and that the usage of generative AI, even for rephrasing, is strictly forbidden.

In-class assistance

If you experience difficulties in solving the problems above, you can receive in-class assistance either on Tuesdays (13:30-15:00, in room D1.14) or on Wednesdays (13:30-15:00, in room D0.03). Please refer to this schedule for any eventual change in the allocated room.

References

[1] Linear Systems: Iterative Methods, Chapter 7, SIAM Book A First Course on Numerical Methods, C. Greif, U. Ascher.

[2] The Cholesky decomposition, Chapter 5, Section 5.5, SIAM Book A First Course on Numerical Methods, C. Greif, U. Ascher.

[3] An Introduction to the Conjugate Gradient Method Without the Agonizing Pain, `http://www.cs.cmu.edu/~./quake-papers/painless-conjugate-gradient.pdf`

[4] Image Kernels, `http://setosa.io/ev/image-kernels/`