Università della Svizzera italiana | Institute of Computing CI

Numerical Computing **2023**

Student: Birindelli Leonardo **Discussed with:** Morales Jeferson

Solution for Project 4 **Due date:** Wednesday, 6 December 2023, 11:59 PM

1. General Questions [10 points]

a) What is the size of the matrix A?
In the image deblurring problem can be formulate as the system of equation :

$$Ax = b$$

where $A \in R^{n^2 \times n^2}$ is the transformation matrix, which represents the blurring effect applied to the vectorized column vector version $x \in R^{n^2}$ of the original image matrix $X \in R^{n \times n}$, all equals to the vectorized column vector version $b \in R^{n^2}$ of the blurred image matrix $B \in R^{n \times n}$ From the data ,it is possible to notice that the size of the transformation matrix A is $n^2 \times n^2 = 62500 \times 62500$ where $n^2 = 62500$ which represents exactly the size of column vector b . I use the following code to obtain that information:

```
% Load matrix A data
load('A.mat')
%returns the row size of matrix A
size(A,1) % it is equal to 62500
```

b) How many diagonal bands does A have?

The matrix A has 49 diagonal bands because, knowing that, A is a d^2-banded matrix where d represents the size of the kernel matrix $K \in R^{d \times d}$ and, analyzing the data, it is possible to see that $d = 7$,therefore the transformation matrix A has $d^2 = 7^2 = 49$ bands.

c) What is the length of the vectorized blurred image b?

The length of the vector b is equals to $n^2 = 62500$. This conclusion is obtained from the fact that $b \in R^{n^2}$ represents the vectorized version of the matrix $B \in R^{n \times n}$ in which the initial information show that $n = 250$,thus basically the dimension of the vector is $n^2 = 250^2 = 62500$

2. Properties of A [10 points]

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

   Since, A is not symmetric , the matrix $\tilde{A}$,which is equal to $A^T A$, is not guaranteed to be necessarily symmetric but only positive-define because the pre-moltiplication with A^T guaranteens only the positive-defined behavior of matrix $\tilde{A}$

2. Explain 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.

   First of all , I know from the beginning, as initial conditions, that $A = A^T$ and $\langle Ax, x\rangle > 0$ if x is not a zero vector because matrix A is symmetric and positive-definite .

   I notice that the function $\frac{1}{2}x^T Ax - b^T x$ can be written in terms of scalar products achieving the expression $\frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle$ and, due to the symmetric characteristic of matrix A, I know that $\langle x, Ax\rangle = \langle Ax, x\rangle$ so the final expression I obtain it is $\frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$. Subsequently, I compute the derivative of $\frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$ in terms of the unknown variable x .

$$f(x) = \frac{1}{2}x^T Ax - b^T x = Ax - b$$

$$= \frac{1}{2}\langle x, Ax\rangle - \langle b, x\rangle \quad \text{where} \quad \langle x, Ax\rangle = \langle Ax, x\rangle$$

$$= \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle$$

   Taking the derivative of $f(x)$ it will lead to :

$$f'(x) = \frac{1}{2}A^T x + \frac{1}{2}Ax - b \quad \text{where} \quad A^T = A$$

$$= Ax - b$$

   Setting the equation to zero, I achieve the equation $Ax = b$ which is what I need to proof:

$$Ax - b = 0$$

$$Ax = b$$

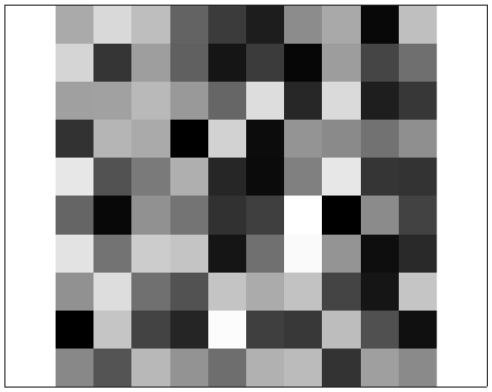
3. Conjugate Gradient [30 points]

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.
   I created the following matlab function in order to realize the conjugate gradient algorithm :

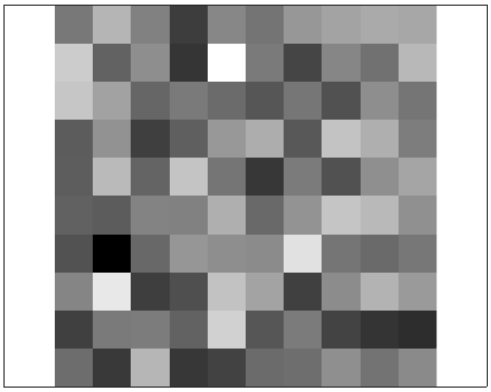
```
function [x,rvec] = myCG(A,b,x0,max_itr,tol)
% The function computes the Conjugate Gradient Method
% A and b are the elements of the system Ax =b
% x0 is an initial vector
% max_itr represents the maximum number of iterations to do in the
    iteration
%method
% tol is the tollerance for which the convergence is reached in the
    iteration method
% x is the solution value
% rvec is the residual at every iteration

rvec = zeros(1, max_itr); %initilize the vector of residuals

%starting point
x = x0;
r = b - A*x;
p = r;
    for i = 1:max_itr

        scalar_prod_m=dot(r,r); % save the scalar product <r_m,r_m>
        alpha = scalar_prod_m / dot(A*p, p);    % alpha_m
        x = x + alpha * p;  % x_m+1
        r = r - alpha * (A*p);  %r_m+1

        beta = dot(r,r) / scalar_prod_m;
        p= r + beta * p;

        rvec(i) = dot(r,r); % save the residual of each iteraction
    end
end
```

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).

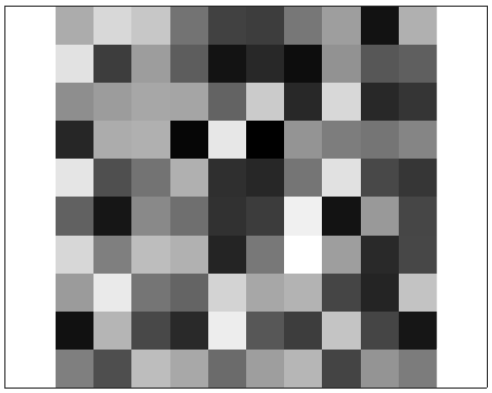
   The next pictures show the difference between the initial data obtained from `x_test_exact` and `b_test` datasets and compare them with the results obtained from the computation of "`myCG.m`" function :



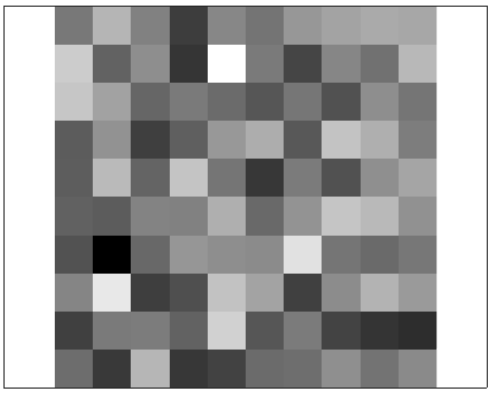
Original image "x_test_exact"



Blurred image "b_test"

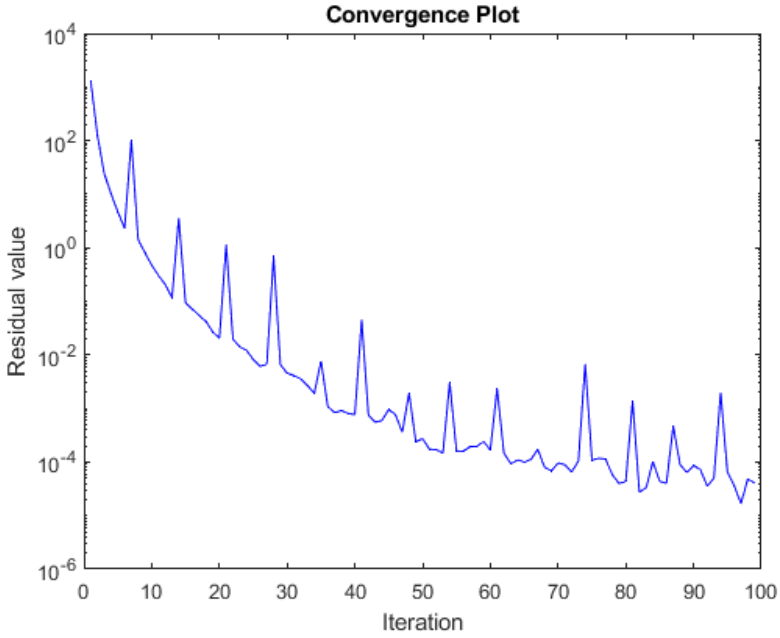


Deblurred image obtained from "myCG" results



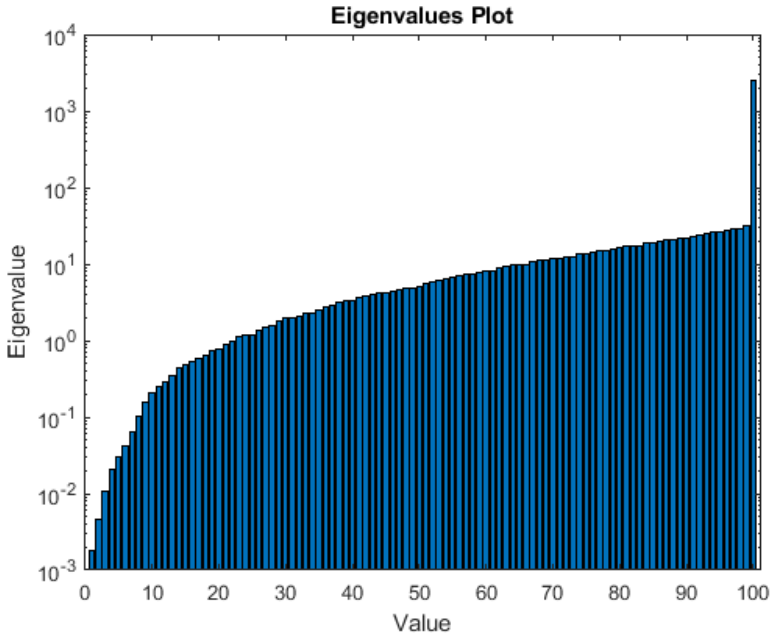
Blurred image obtained from "myCG" results

On the other hand, the convergence graph of "`myCG.m`" result is the following:



3. Plot the eigenvalues of `A_test.mat` and comment on the condition number and convergence rate.

   The subsequent picture show the information related to the eigenvalues of matrix "`A_test.mat`":



$$\kappa(A) = \frac{\sigma_{\max}}{\sigma_{\min}} = \frac{|\lambda_{max}|}{|\lambda_{min}|} = \frac{2511.69}{0.0018} \approx 1.37 \times 10^6$$

Since knowing that matrix A is symmetric the singular values of A correspond directly to the absolute value of its eigenvalues $\sigma = |\lambda|$, the value of $\kappa(A)$ is obtained computing the division between the biggest and the smallest eigenvalue of matrix A. It is also possible to notice that $\kappa(A) \gg 1$ so it means that the convergence rate is slow and basically the solution of the system are reached with a low rate. Whenever this condition afflicts a matrix, it is said that the matrix is "ill-conditioned" .In this particular case , the matrix contained in the file "`A_test.mat`" is positive-defined but if it would not , the condition number can get worst and it deteriorates the convergence rate value. Indeed $\kappa(A)^2 = \kappa(\tilde{A})$ where matrix $\tilde{A}$ is computed to guarantee the positive-define condition by the pre-multiplication with A^T for the matrix A :

$$\tilde{A}x = \tilde{b} \quad \text{where} \quad \tilde{A} = A^T A \quad \text{and} \quad \tilde{b} = A^T b$$

4. Does the residual decrease monotonically? Why or why not?
   No,the residual values are not decreasing monotonically as it possible to see from the picture 2 because for many iterations the corresponding residual value do not decrease compared to the iteration before. This result is due to the fact that the function `myCG` aims to reach the closest solution x and each time a solution closer to solving the system is reached, the residual value between the most 'correct' solution and the estimated solution will decrease more and more at every iteration

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.

   Before start solving the deblurring problem on blurred image matrix `B.mat` and transformation matrix `A.mat` using the `pcg` function I need to calculate the incomplete Cholesky factorization: the IC factors are necessary to define a preconditioner for the preconditioned conjugate gradient function. Firstly, I start computing the pre-multiplication with A^T on the matrix A so as to guarantee to be positive-define obtaining the system $\tilde{A}x = \tilde{b}$. Then , I configure the `ichol` function so that the preconditioner is created from compensated version of incomplete Cholesky factors with zero-fill. The compensated condition is necessary for the IC factor because it may not exist and doing a diagonal shift of an α on the preconditioner matrix $P = F^T F$, where F is the incomplete Cholesky factor, can make F computable.

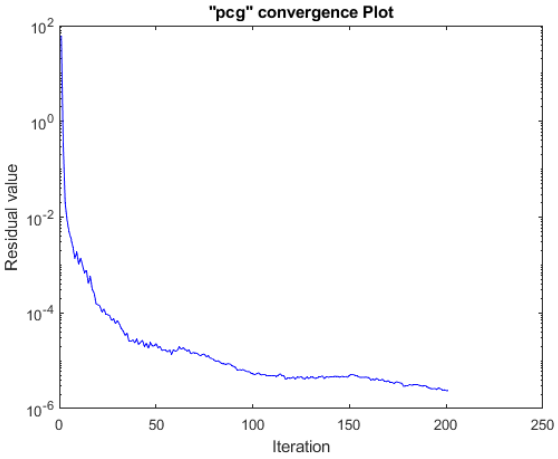
```
% Incomplete Cholensky factorial parameters

%pre-multiplication with A^T
A_positive=A'*A; % necessary to guarantee a positive-defined matrix
b_positive=A'*b;

options.type = 'nofill'; % incomplete Cholesky factor with zero-fill
options.diagcomp=0.01; % diagonal shift of 0.01 used to guarantee the
    existance of IC factor

%Compute the incomplete Cholesky factorization
IC_matrix = ichol(A_positive, options);
```

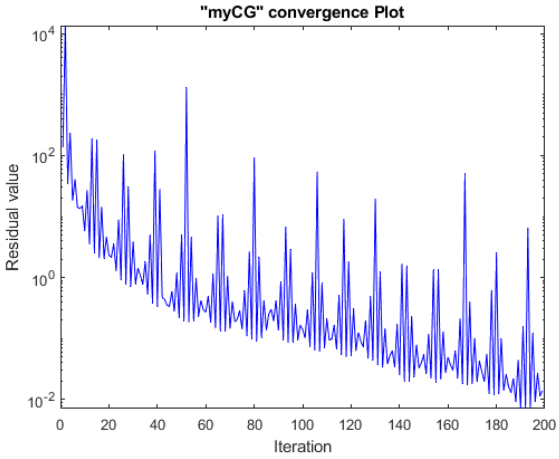
Finally, I compute the `pcg` and `myCG` functions and I plot the resulting images.
More information about the implementation are available on the matlab file "`code_template.m`"

Initilal blurred image

"pcg" result deblurred image

"myCG" result deblurred image





As it is possible to notice from plots, both function obtain a similar final result in the deblurred image. Instead in the convergence plots, there are some differences between each other. Indeed in the `pcg` the graph , the residual values get lower at each iteration than the ones computed by `myCG` function.Naturally this behaviour is due to the application of a preconditioning factor which leads to obtain a reduced condition number of system . On the other hand, `myCg` graph is characterized by a slower convergence in which the residual values decrease only after many iterations compared to the other plot. Notice that in both plots even if the convergence is

not reached in 200 iterations, the final solutions bring to a correct deblurred version of the picture

2. When would `pcg` be worth the added computational cost? What about if you are deblurring lots of images with the same blur operator?

   In my opinion, the `pcg` function can justify the extra computation of a preconditioner like the incomplete Cholesky factorization because despite the computation of an high-quality preconditioner for the problem, it can reduce the condition number of the system which leads to reach a faster convergence. Indeed, in the case in which there are many images where the blur operator remains the same, the use of a preconditioning process has a strong impact on the computational time which leads to require less waiting time to compute the system compared to simply use a pre-multiplication operation on the transformation matrix

5. References

1. "Steepest Descent & Conjugate Gradient" slides of the lesson of 22nd November 2023 available on iCorsi platform

2. Jonathan Richard Shewchuk,1994 "An introduction to the Conjugate Gradient Method Without the Agonizing Pain"

3. "A First Course in Numerical Methods, Chapter 7 ,Linear Systems: Iterative Solvers" available