Università della Svizzera italiana | Institute of Computing CI

Numerical Computing **2023**

Student: Hun Rim **Discussed with:** Georgy Batyrev

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

1. General Questions [10 points]

1. What is the size of the matrix A?

   The size of matrix A stored in 'blurred_data' folder is **62500 X 62500**. The result is obtained as following:

```
% Load the matrix struct
load('blured_data/A.m');

% Get the size of matrix A loaded
sizeA = size(A);

... print the matrix size...
```

   It can also be calculated manually as we know the value of $n = 250$, and as A is n^2 X n^2 matrix A is going to be 250^2 X 250^2 which is **62500 X 62500**.

2. How many diagonal bands does A have?

   We have a kernel matrix with the size of 7 X 7. As $(d \leq n)$, and $n = 7$, the d can be set as 7. As the matrix is d^2-matrix, the number of diagonal bands in A is equal to d^2 which is 49.

3. What is the length of the vectorized blurred image in b?

   Matrix A operates on a "row-vectorized" image matrix. In this case, the length of the vectorized blurred image in b would be equivalent to $len(b) = n \times n$. As $n = 250$, length of b would be $250 \times 250 = 625,000$.

2. Properties of A [10 points]

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

$$A^T Ax = A^T b \tag{1}$$
$$\tilde{A}x = \tilde{b} \tag{2}$$

   The operation in equation 1 needs to be performed in case the coefficient matrix A is not positive-definite like the case we have, as the Conjugate Gradient method requires the coefficient matrix to be positive-definite. However, during this operation, conditional number $\kappa(A)$ gets effected because $\kappa(\tilde{A}) = \kappa(A)^2$. The counterpart of A is symmetric as it is a square matrix formed from product of $B^T B$, hence, matrix A also has to be symmetric.

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.

   The minimum of the given equation can be easily computed by computing the derivative of the equation:

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

   Because A is symmetric, we can assume $A = A^T$, and we can replace A^T with A.

$$= \frac{1}{2}Ax + \frac{1}{2}Ax - b$$

   And it can be simplified to $Ax - b$. A function's minimum lies where the $f'(x) = 0$, hence the minimum of the given function can be found when:

$$Ax - b = 0$$

   Which is equivalent to :

$$Ax = b$$

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.

   Conjugate Gradient solver is an iterative method of solving a large linear system to get a approximated solution. Our implementation takes A the system matrix, b the row-vectorized vector, $x0$ the initial guess, *max_itr* the maximum number of iterations, *tol* the tolerance for convergence, and output the x which is the solution to equation 2 with the residual norms of each iteration as *rvec*. The implementation is as following:

```
function [x, rvec] = myCG(A, b, x0, max_itr, tol)
    rvec = [];
    x = x0;
    r = b - A * x;
    d = r;
    p_old = dot(r, r);
    for i = 1:max_itr
        s = A * d;
        alpha = p_old / dot(d, s);
        x = x + alpha * d;
        r = r - alpha * s;
        p_new = dot(r, r);
        beta = p_new / p_old;
        d = r + beta * d;
        p_old = p_new;
        rvec = [rvec, p_new];
        if sqrt(p_new) <= tol
```

```
                disp('Converged');
                break;
            end
        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).

   Implementation is as the following:

```
load('test_data/A_test.mat', 'A_test');
load('test_data/b_test.mat', 'b_test');

m = sqrt(length(b_test));
n = m;

guess=zeros(size(A_test, 1), 1);
max_itr = 200;
tol = 1e-4;

[x, rvec] = myCG(A_test, b_test, guess, max_itr, tol);
...plot the graph...
```

   The resulting graph is as the following:

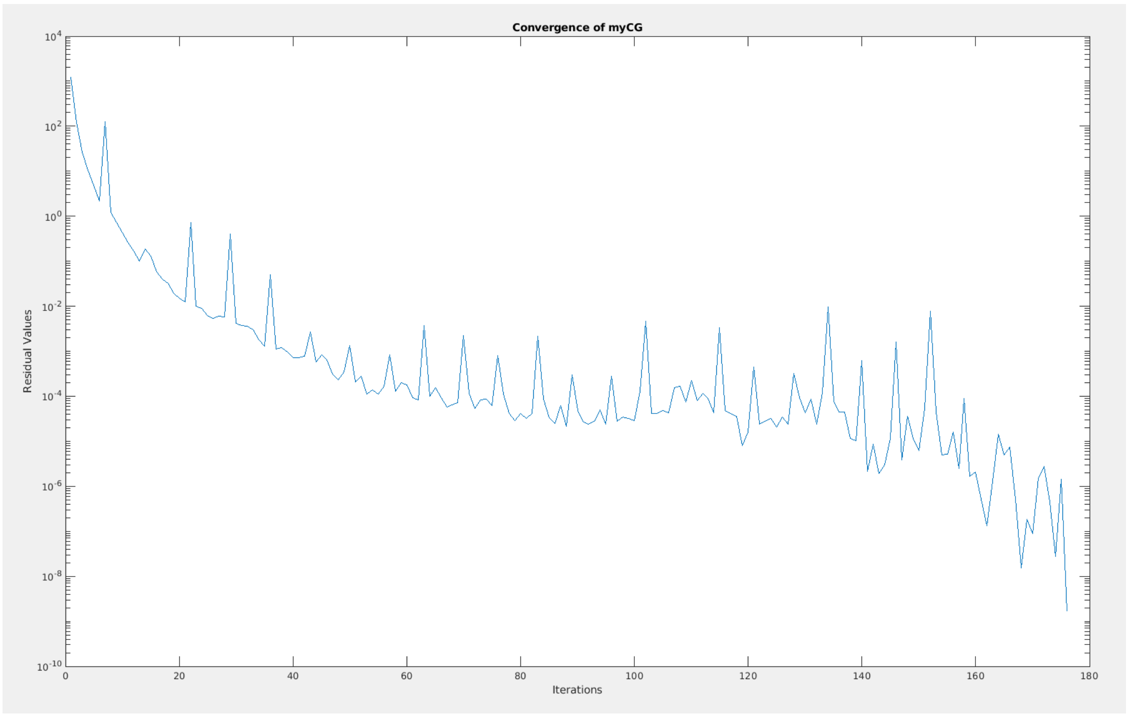

Figure 1: Plotted residual values against iteration using myCG function

The implementation of Conjugate Gradient (myCG) seems to find the convergence of the test data very quickly under 200 iterations.

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

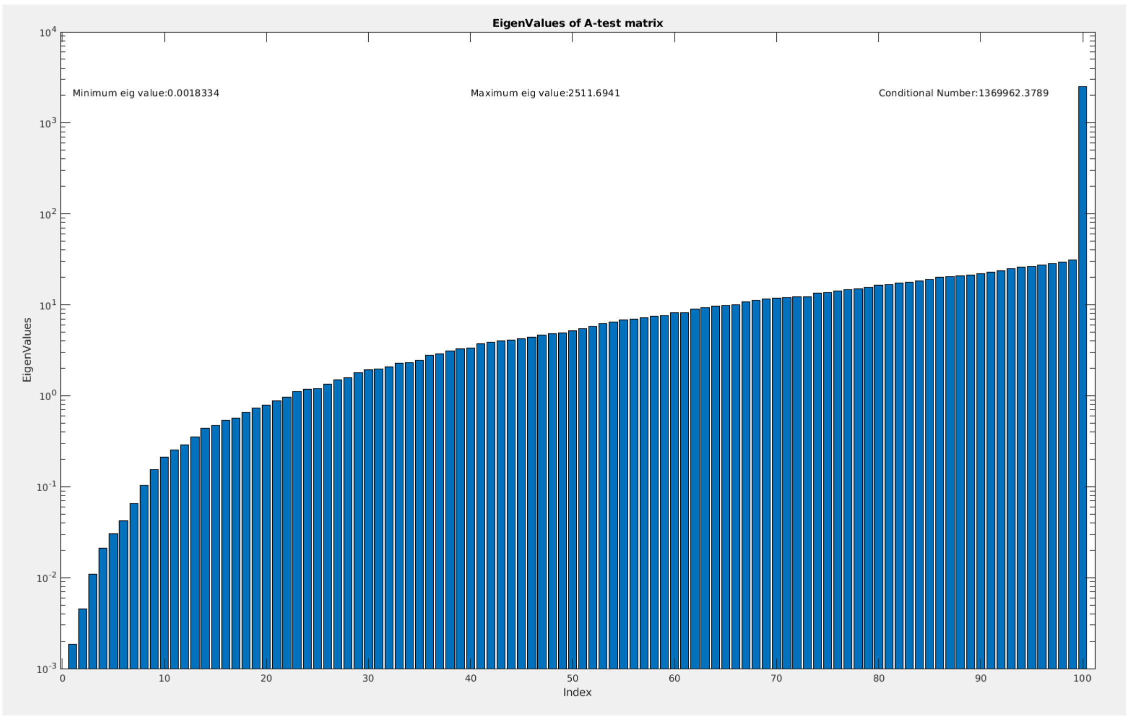

Figure 2: Eigen value distribution in matrix A_test.mat (Eigen values are logged for better visualization)

Implementation can be seen in *ex3c.m* file in the data folder.

As it can be seen from the plotting, there is a significant difference between the smallest and the largest absolute eigenvalues. As a result, it results in a quite significant conditional value as the conditional value is calculated as following:

$$\kappa(A) = \frac{\sigma_{max}}{\sigma_{min}} \tag{3}$$

Condition number has a direct impact on the convergence rate of the CG method.

$$convergenceRate = \sqrt{\kappa(A)} \tag{4}$$

If the condition number is too large, it causes the CG method to converge slower as it can be predicted from the equation above. In case the condition number is too significant, the matrix A is said to be ill-conditioned, whereas, the opposite is said to be well-conditioned. In case the condition number is ill-conditioned, a method called *preconditioning* can be used to make the matrix well-conditioned.

4. Does the residual decrease monotonically? Why or why not?

From the figure 1, it is visible that the residual value is decreasing monotonically. To clarify, monotonically decreasing means it shows a general decreasing trend but doesn't necessarily have to be strict. It is a sign that as the iteration increases, the solution is getting closer and the error is being reduced, which is a positive sign.

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.

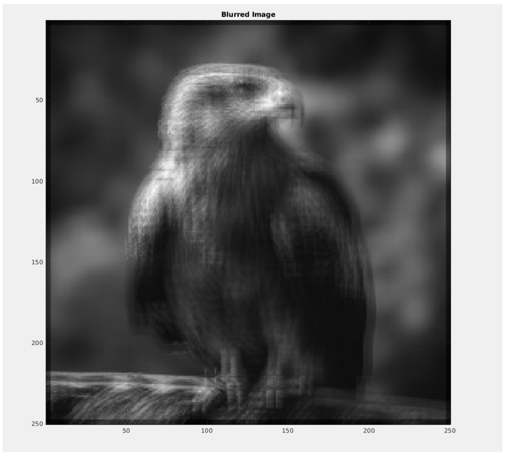

Figure 3: Original Blurred Image

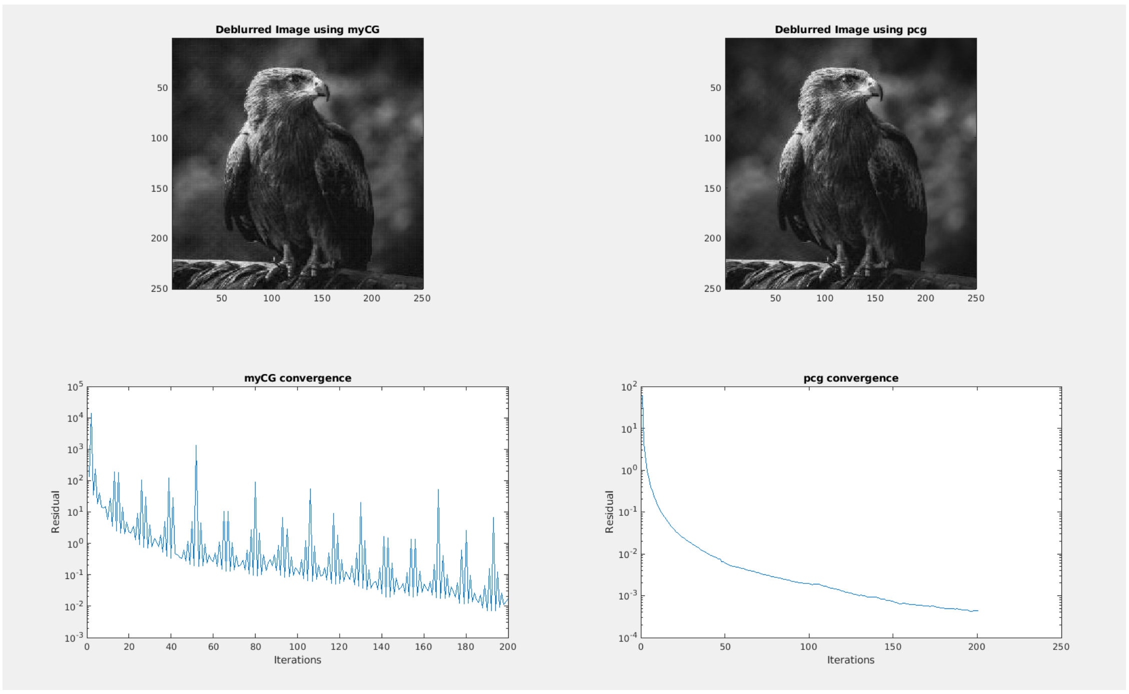

Figure 4: Comparison of myCG function and pcg function (Top: image deblurring, Bottom: Convergence graph)

Script for implementation can be seen in *ex4.m*.

In terms of image deblurring, both methodologies exhibit remarkable effectiveness, yielding outcomes that are almost indistinguishable, leaving only subtle traces of variance. However, a more pronounced differentiation emerges when we observe their convergence behaviors.

The myCG method presents a recognizable, though nonlinear, design with a convergence trajectory that gently falls. As it iteratively moves closer to convergence, it shows a distinctive stabilization phase that is in perfect alignment with the expected behavior.

In contrast, the pcg approach unfolds with an strictly decreasing pattern, carefully following to a declining trajectory. Although the two methods appear to follow a similar inverse logarithmic trend, the PCG approach stands out due to its strict adherence to a continually declining pattern. This contrast with the myCG approach adds another level of complexity to the convergence, as it periodically generates fluctuations followed by a harmonic stabilization as expected and shown from the implementation.

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

   PCG is particularly useful when dealing with ill-conditioned linear systems. It can mitigate the effects of ill-conditioning and improve the convergence rate compared to the standard Conjugate Gradient (CG) method (*myCG* method for example). Also, it is implemented in a way it would be easier to run it in a parallel computing architecture, which in modern world is faster than standard Conjugate Gradient method.

   In case the same blur operator is used, PCG method is very efficient. The computational cost of setting up the preconditioner, (reutilized after calculating once) which is very costly, can be justified by the acceleration in convergence of images afterwards. The preconditioner is applied to the system matrix and helps improve its conditioning, leading to faster convergence of the PCG method.