Università della Svizzera italiana

Institute of Computing CI

Numerical Computing **2023**

Student: Harkeerat Singh Sawhney

Due date: Wednesday, 6 December 2023, 11:59 PM

Numerical Computing 2023 — Submission Instructions
(Please, notice that following instructions are mandatory: submissions that don't comply with, won't be considered)

- Assignments must be submitted to iCorsi (i.e. in electronic format).
- Provide both executable package and sources (e.g. C/C++ files, MATLAB). If you are using libraries, please add them in the file. Sources must be organized in directories called:
  Project_number_lastname_firstname
  and the file must be called:
  project_number_lastname_firstname.zip
  project_number_lastname_firstname.pdf
- The TAs will grade your project by reviewing your project write-up, and looking at the implementation you attempted, and benchmarking your code's performance.
- You are allowed to discuss all questions with anyone you like; however: (i) your submission must list anyone you discussed problems with and (ii) you must write up your submission independently.

1. General Questions [10 points]

1.1. Size of Matrix A

From the Background Information given for this Project we do know that $A \in \mathbb{R}^{n^2 \times n^2}$ indicates the transformation matrix coming from the repeated application of what is referred to as the "image kernel", which in our case tends to produce the blurring effect. In the other hand B is the transformed blurred image and X is the original square, grayscale image matrix, in which each matrix entry corresponds to one pixel value. Hence the blurring computation can be defined be following equation:

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

In the above equation x and b are the vectorized repersentation of X and B respectively.

Listing 1: Computing the size of A

```
%% Load Default Img Data
load('blur_data/B.mat');
B=double(B);
n = size(B,1);
sizeA = n * n;
disp(['Size of A: ', num2str(sizeA)]);
```

From the Code Listing 1 we can see that the size of A is 62500×62500. This is computed through the size of B which is 250×250 and then multiplying it by itself.

1.2. How many diagonal bands does A have?

It is understood that A is a d^2-banded symmetric matrix, where $d << n$. Since we know that the size of the kernel image matrix is 7×7 then we can also compute the amount of diagonal bands that A has. Hence the amount of diagonal bands that A has is 49.

1.3. What is the length of the vectorized blurred image b

In order to compute the length of the vectorized blurred image b, we need to compute the size of B and then multiply it by itself. We know that B is a 250×250 matrix, hence the length of the vectorized blurred image b is 62500.

2. Properties of A [10 points]

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

A is used to compute the Conjugate Gradient method, which is an iterative method to solve the linear system $Ax = b$. If A is symmetric of full rank but not positive-definite we can bypass this issue by solving the augmented System.

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

$$\tilde{A}x = \tilde{b} \tag{3}$$

In the above equation the pre-multiplication with A^T ensures that the resulting matrix $\tilde{A}$ is symmetric and positive-definite. Hence even if A is not symmetric, we can still compute $\tilde{A}$ because of the properties of A^T.

2.2. Explain why solving $Ax = b$ for x is equivalent to minimizing $\frac{1}{2}x^TAx - b^Tx$ over x, assuming that A is symmetric positive–definite.

We want to show that by minimizing $\frac{1}{2}x^TAx - b^Tx$ over x is equivalent to solving $Ax = b$. We can do this by taking the derivative of $\frac{1}{2}x^TAx - b^Tx$ with respect to x and setting it to zero. Hence we get the following equation:

$$\frac{d}{dx}\left(\frac{1}{2}x^TAx - b^Tx\right) = 0 \tag{4}$$

$$\frac{1}{2}\left(x^TA + x^TA^T\right) - b^T = 0 \tag{5}$$

$$x^TA - b^T = 0 \tag{6}$$

$$x^TA = b^T \tag{7}$$

$$x^T = b^TA^{-1} \tag{8}$$

$$x = \left(b^TA^{-1}\right)^T \tag{9}$$

$$x = \left(A^{-1}\right)^T b \tag{10}$$

$$x = A^{-1}b \tag{11}$$

$$Ax = b \tag{12}$$

Therefore can see at the end that we get the equation $Ax = b$ which is what we wanted to show.

3. Conjugate Gradient [30 points]

3.1. 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.

In this question we are asked to write a function for the conjugate gradient solver. The function is called `myCG` and it takes in the following parameters: `A`, `b`, `x0`, `max_itr` and `tol`. The function returns the solution value `x` and a vector containing the residual at every iteration `rvec`. The function is implemented in the Code Listing 2. The code is written from the provided Conjugate Gradient Algorithm in the project description.

The function first initializes the solution x with the initial guess $x0$, and computes the initial residual r as $b - A * x0$. the direction d is also initialized as r. Then the function enters a for loop which iterates for the maximum number of iterations. In each iteration, it performs the steps which is provided in the algorithm.

Listing 2: Matlab function for Conjugate Gradient

```
function [x, rvec] = myCG(A, b, x0, maxIter, tol)
    rvec = [];
    x = x0;
    r = b - A * x0;
    d = r;
    pho_old = dot(r, r);


    for i = 1:maxIter
        s = A * d;
        alpha = pho_old / dot(d, s);
        x = x + alpha * d;
        r = r - alpha * s;
        pho_new = dot(r, r);
```

```
15
16          beta = pho_new / pho_old;
17          d = r + beta * d;
18          pho_old = pho_new;
19
20          rvec = [rvec, pho_new];
21
22          if sqrt(pho_new) < tol
23              disp('Converged');
24              break;
25          end
26
27
28      end
29  end
```

3.2. In order to validate your implementation, solve the system defined by `A_test.mat` and `b_test.mat`. Plot the convergence (residual vs iteration)

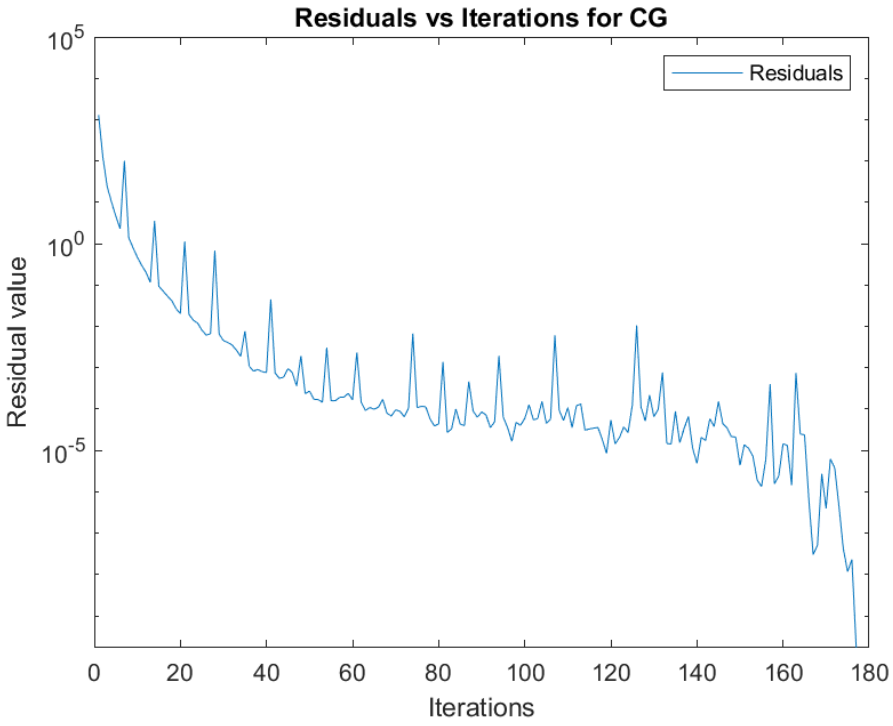


Figure 1: Residuals vs Iterations for the Conjugate Gradient Algorithm

In this question we are asked to validate our implementation of the Conjugate Gradient Algorithm. We are given the matrix A and the vector b. We are then asked to use the function which we implemented in the previous question to solve the system $Ax = b$. When we obtain the solution x we are asked to plot the convergence of the residuals vs the iterations. The plot is shown in Figure 1.

Our maximum number of iterations is set to 200 and the tolerance is set to 10^{-4}. From the Figure 1 we can see that the residuals converge right before the 180th iteration. Also the from the plot we can see that the residual overall decreases, but does has multiple spikes in between.

3.3. Plot the eigenvalues of A `test.mat` and comment on the condition number and convergence rate.

In this question we are asked to plot the eigenvalues of A and comment on the condition number and the convergence rate. Condition Number is defined as the ratio of the largest eigenvalue to the smallest eigenvalue. The condition number $\kappa(A)$ is the relation of sensitivy of the solution x to the changes in the right hand side b. The condition number is computed as follows:

$$\kappa(A) = \frac{\lambda_{max}}{\lambda_{min}} \tag{13}$$

If small changes in b cause large changes in x, then the system is called as ill-conditioned and the condition number of the system is large. However in the other hand, if small changes in b cause small changes in x, then the system is called as well-conditioned and the condition number of the system is small. Therefore the condition number is a measure of the sensitivity of the solution x to the changes in the right hand side b.

Hence in order to compute the condition number of A we need to compute the eigenvalues of A. The eigenvalues of A are computed using the `eig` function in Matlab. The eigenvalues are then sorted in ascending order and plotted. The plot is shown in Figure 2. As it can be seen in the plot the difference between the largest eigenvalue and the smallest eigenvalue is very large. Hence the condition number of A is very large. This means that the system is ill-conditioned and small changes in b will cause large changes in x. This can also be seen by calculating the condition number of A. We can do that by using the inbuilt `cond` function in Matlab. The condition number of A is approximately 1.67×10^6 which is very large. Hence this is aligned with our previous observation that the system is ill-conditioned.

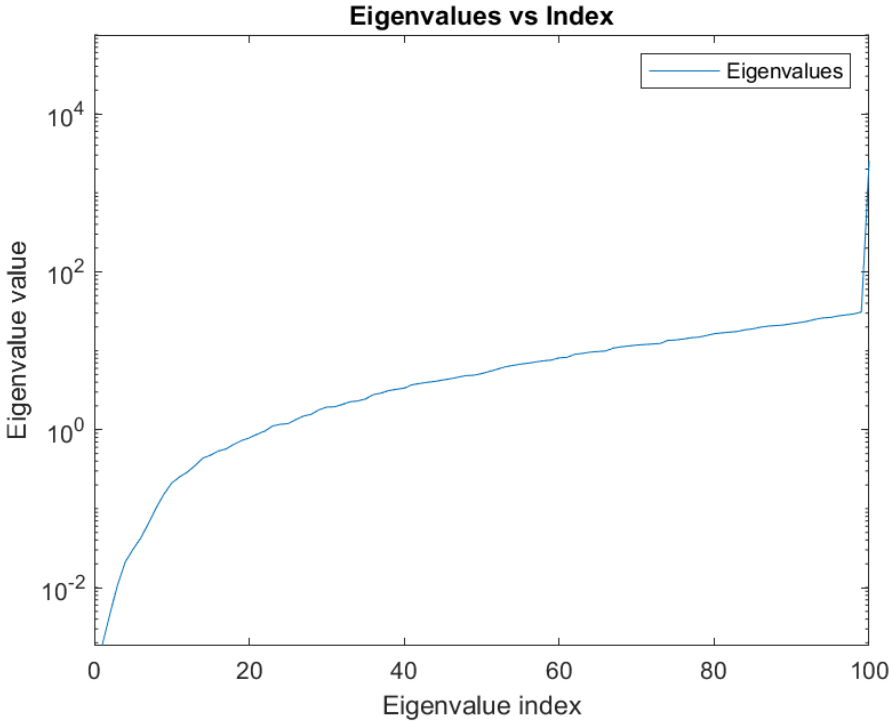


Figure 2: Eigenvalues of A

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

As it can be seen from Figure 1 the residual does not decrease monotonically. The residual decreases overall but has multiple spikes in between. Therefore when we are computing the residuals, we are computing the difference between the actual solution x and the computed solution x. Since the

system is ill-conditioned, small changes in b will cause large changes in x. Hence the residual will have multiple spikes in between, but will overall decrease.

4. Deblurring problem [35 points]

4.1. 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` and `tol = 1e-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.

In this question we have to solve the deblurring problem for the blurred image matrix `B.mat` and transformation matrix `A.mat` using our implemented `myCG` function and Matlab's `pcg` function. We are also asked to use `ichol` as a preconditioner for the `pcg` function

We first load the matrices A and B. A is the blurring matrix and B is the blurred image. We then prepare for the Preconditioned Conjugate Gradient (PCG) method by creating the augmented system $augA$ and $augB$. We compute the preconditioner M by using matlab function `ichol` with the `nofill` option (as asked). The PCG method is then applied to solved the system $augA \times x = augB$. We also make sure to have a diagonal shift of $\alpha = 0.01$ in the PCG method which is done by using the matlab function `speye` in which we create a sparse identity matrix and then multiply it by α and add that to our matrix $augA$. The solution x is then reshaped to the original image size and then displayed.

As for the implementation with our `myCG` function, we call the function with the parameters A, B, $x0$, $maxIter$ and tol. With this we are able to solve the system $Ax = b$. For both the implementation the result is then reshaped to the original image size and then displayed. Also the parameters for both the implementations are set to $maxIter = 200$ and $tol = 1e-6$.
In the Code Listing 3 we can see the implementation of the deblurring problem. The code follows the explanation given above.

Listing 3: Matlab code for solving the deblurring problem

```
%% Excercise 4.1
% Load data
loaded_A = load('blur_data/A.mat');
loaded_B = load('blur_data/B.mat');

A = loaded_A.A;
B = loaded_B.B;


img = B; % Blurred image
n = size(img, 1);
b = B(:); % Vectorized blurred image
guess = ones(size(A, 1), 1);
maxiter = 200; % Maximum number of iterations
tol = 1e-6; % Tolerance

% Display image
imagesc(reshape(img, [n, n]));
colormap('gray');
axis off;
saveas(gcf, '../Template/graphs/blurred.png');
```

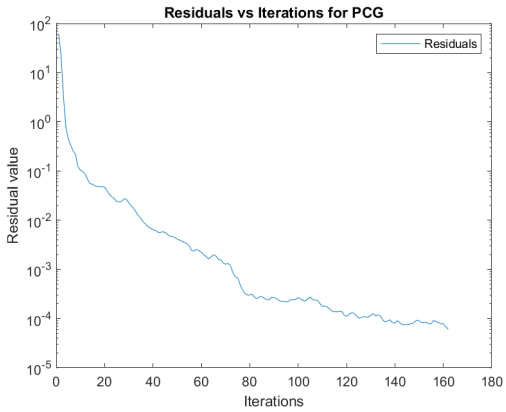
```
22
23      % Solve the system using 'pcg'
24      augA = A' * A; % Augmented matrix of A
25      augA_shifted = augA + 0.01 * speye(size(augA)); % Shifted matrix
26
27      L = ichol(augA_shifted, struct('type', 'nofill')); % Incomplete Cholesky
            factorization
28      M = L * L'; % Preconditioner
29      M1 = L';
30      M2 = L;
31      augB = A' * b; % Augmented vector of b
32
33      [x_pcg, flag, relres, iter, resvec_pcg] = pcg(augA, augB, tol, maxiter, M1, M2); %
            Solve the system
34
35
36      % Draw deblurred image obtained with 'pcg'
37      imagesc(reshape(x_pcg, [n, n]));
38      colormap('gray');
39      axis off;
40      saveas(gcf, '../Template/graphs/deblurred_pcg.png');
41
42      % Draw residuals vs iterations
43      semilogy(resvec_pcg);
44      xlabel('Iterations');
45      ylabel('Residual value');
46      legend('Residuals');
47      title('Residuals vs Iterations for PCG');
48      saveas(gcf, '../Template/graphs/residuals_pcg.png');
49
50      disp(['Convergence flag: ', num2str(flag)]);
51
52      % Solve the system using 'myCG'
53
54      [x, residuals] = myCG(A, b, guess, maxiter, tol); % Solve the system
55
56      % Draw deblurred image obtained with 'myCG'
57      imagesc(reshape(x, [n, n]));
58      colormap('gray');
59      axis off;
60      saveas(gcf, '../Template/graphs/deblurred_mycg.png');
61
62      % Draw residuals vs iterations
63      semilogy(residuals);
64      xlabel('Iterations');
65      ylabel('Residual value');
66      legend('Residuals');
67      title('Residuals vs Iterations for myCG');
68      saveas(gcf, '../Template/graphs/residuals_mycg.png');
```

Figure 3: Blurred Image

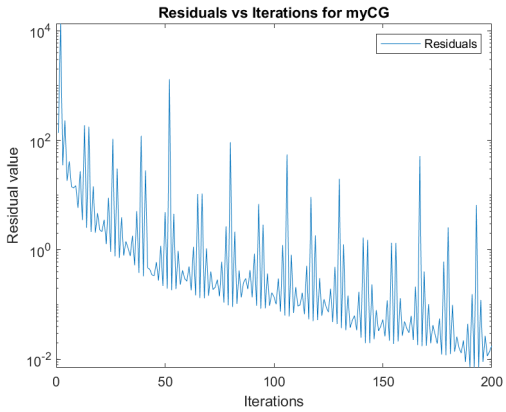
(a) Deblurred Image using myCG

(b) Deblurred Image using PCG

Figure 4: Comparison of Deblurred Images using myCG and PCG



(a) Residuals vs Iterations for PCG



(b) Residuals vs Iterations for myCG

Figure 5: Comparison of Residuals vs Iterations for PCG and myCG

As it can be seen from the Figures 4, the blurred image is deblurred using both the methods.

However in our implementation the blurring using pcg is better than the blurring using myCG. This can be seen by comparing the residual graphs for both of them in Figure 5 which the residual for pcg is much lower than the residual for myCG. The main reason for why this happens is because of the preconditioner. The preconditioner is used to reduce the condition number of the system, which makes the system well-conditioned and hence the solution is more accurate. Hence the blurring using pcg performs better than the blurring using myCG, but it should also be noted that the pcg method is more computationally expensive than the myCG method. Therefore if we want a more accurate solution and we are willing to pay the computational cost, then we should use the pcg method. However if we want a less accurate solution and we are not willing to pay the computational cost, then we should use the myCG method.

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

There are several scenarios in which `pcg` is worth the added computational cost. One of the scenarios is when the system is ill-conditioned. In this case the solution obtained by `pcg` will be more accurate than the solution obtained by `myCG`. Another scenario is when the system is large. In this case the `pcg` will be faster than `myCG` because of the preconditioner. The preconditioner reduces the condition number of the system, hence the system is well-conditioned and the solution is more accurate. Hence in this case `pcg` is worth the added computational cost.

If we are deblurring lots of images with the same blur operator, then `pcg` is worth the added computational cost. This is because the preconditioner is computed only once and then used for all the images. Therefore the computational cost is reduced.

5. Reproducing the results

In order to reproduce the results, Matlab is needed to be installed. The code is in the directory Project4_FilesData and inside that directory there is a folder called `graphs` which contains all the graphs. There would be two more directories called `blur_data` and `deblur_data` which contain the data for the project. The file `code_template.m` must be run in order to reproduce the results. It uses the file `myCG.m` which is the implementation of the Conjugate Gradient Algorithm.

Bellow is the visualization of the directory structure:

```
Project_4_sawhney_harkeerat
├── blur_data
│   ├── A_test.mat
│   ├── b_test.mat
│   └── x_test_exact.mat
├── test_data
│   ├── A.mat
│   └── B.mat
├── graphs
│   ├── blurred.png
│   ├── deblurred_mycg.png
│   ├── deblurred_pcg.png
│   ├── residuals_mycg.png
│   └── residuals_pcg.png
└── code_template.m
```