ELEC-E5431- Phung Duc Thao – 913223

Table of Contents

Problem Statement

In this exercise we are going to slove a optimization problem of a simple quadratic function given:

$$minarg(x)\frac{1}{2}x^TAx - b^Tx$$

Initial parameters

```
% Clear the console output
clc
% Clear all the variable space
clear
% Set random seed for random generator
% For this value of RNG, I choose because I can create the condition
 number
% less than 30 unit
rng(860)
```

In this assignment, I choose dimension of variable x to be 100, I have tested with 1000 (seems work fine) and 3000 (getting slow and sometimes crashed on my computer)

Choosing the dimension for variable n

```
n = 100;
% n = 1000;
% n = 3000;
```

Define Matrix A and b

First, define variable b where b should be in range of A given dimension (n,1)

```
b = randn(n,1);
disp("The dimension matrix b is:")
size(b)
```

```
The dimension matrix b is:

ans =

   100     1

```

Next generate Q, the unitary matrix where the multiplication of this matrix with it inverse become the identity matrix and this matrix contains orthonormal basis of eigenvetors of A

```
Q = orth(randn(n));
```

Let define the matrix E as a diagonal matrix where its main diagonal matrix contains the corresponding eigenvalues of A

```
E = diag(abs(randn(n,1)));
```

Then the matrix A which is non negative is defined as followed:

```
A = Q*E*Q';
disp("The dimension matrix A is:")
size(A)
```

```
The dimension matrix A is:

ans =

   100   100

```

The condition number of matrix A is given as:

```
disp("Condition of A is:")
condtionNumber = cond(A);
condtionNumber
% conditional less than 30 will always work fine
```

```
Condition of A is:

condtionNumber =

   26.7129

```

Next we define a random or initial variables for x

```
initial_x = randn(n,1);
disp("The dimension matrix x is:")
size(initial_x)
```

```
The dimension matrix x is:

ans =
   100     1
```

Let's check if the x'*A*x is non negative value

```
disp("The value of x'*A*x is:")
initial_x'*A*initial_x
```

```
The value of x'*A*x is:

ans =

   92.9068
```

To solve the unconstrained minimization problem, we need to solve a system of linear equations Ax=b and the gradient of the objective function is in the form Ax-b and it should equal to 0 The gradient function for the quadratic function is:

$$\frac{1}{2}(A' + A)x - b$$

Since matrix A is non-negative definite, the matrix A is symmetric and A' is equal to A. The result gradient function is:

$$Ax - b$$

To get the minimum value for the function, the gradient of function should be 0 for optimality, and the gradient is solved as follows: Ax-b=0 or Ax=b

```
Xoptimal=mldivide(A,b);
disp("The size matrix Xoptimal is:")
size(Xoptimal)
```

```
The size matrix Xoptimal is:

ans =

   100     1
```

For the above equation, I have tested with different dimensions of x,A and b where it appears to be very slow (or sometimes it crashed) (took around 1 min) to solve the system linear equation above with dimension n=3000. For the n in range 1000 to 2000, my computer was still able to produce a result.

Setting parameters for solving problem

In this section, we are going to solve the optimization problem above using different techniques. The same matrix A and b are used for these techniques with same maximum iteration number and tolerance value defined as following:

Problem 1: Gradient Descent Algorithm

```
tic
```

```
x = initial_x; % Capture the initial solution
iterations = 0; % Initial iteration
max_iterations = 1e4; % maximum number of iterations
tol = 1e-5; % tolerance value
% alpha = 1e-2; % step size
% alpha = 0.5e-1; % step size
% alpha = 0.35; % step size
```

The step size alpha is defined by the Lipschitz constant and the Lipschitz function should be continuous where the derivative of the gradient is finite and the function is Lipschitz continuous gradient so the Lipschitz constant of gradient is A and the step size should be less than or equal to 1/L(gradient of function) which is 1/norm(A). The number of maximum iterations chosen for all problems below are 10000 and the tolerance value of norm(gradient of function) should be less than 0.00001.

```
alpha = 1/norm(A); % step size
f = 0.5*(x'*A*x)-(b'*x); % Function f
% foptimal = 0.5*(Xoptimal'*A*Xoptimal)-(b'*Xoptimal); Value of
 minimum y

gradient_norm = norm(A*x-b); % Define initial gradient norm
log_error = zeros(1,max_iterations); % Matrix for plotting convergence
 rate

while iterations < max_iterations && gradient_norm > tol
% First compute the gradient of function
gradient = grad_function(x,A,b);
% Calculate the norm of the gradient
gradient_norm = norm(gradient); % Gradient Norm
% Update the variable x using the gradient with selected step size
x_ = x - alpha*gradient;
% Decrease the iteration
iterations = iterations + 1;
% Update the variable x for next iteration
x = x_;
% Logging the updated x and compare with the optimal value x
log_error(iterations+1)=log(norm(x-Xoptimal));
end
toc
```

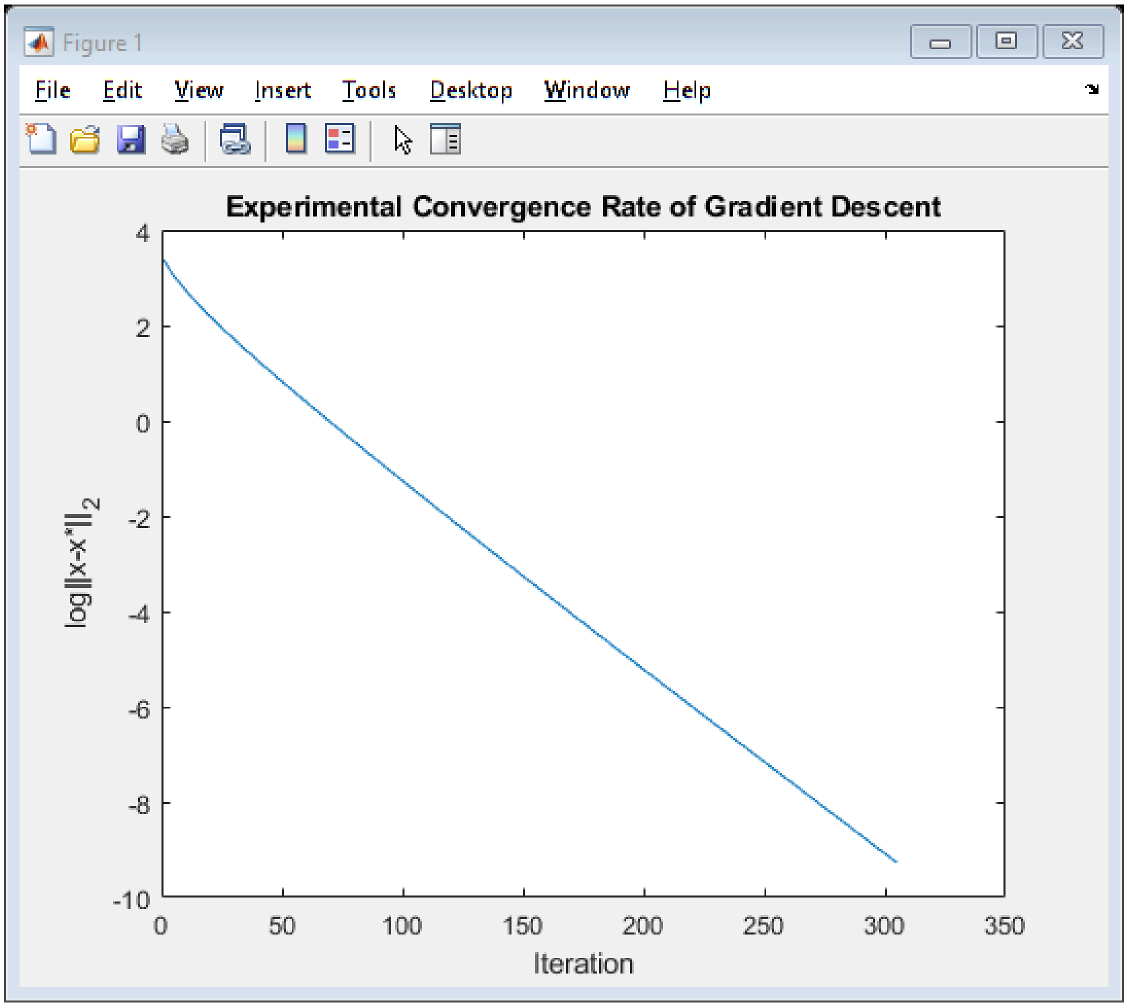
Extract optimal x has just been generated

```
x_optimal_GD = x;
disp('Loop ends in the number of iterations:');
disp(iterations);
disp('Norm of optimal gradient is:');
disp(gradient_norm);
```

```
Loop ends in the number of iterations:
   305

Norm of optimal gradient is:
   9.8566e-06
```

Plotting for Convergence Rate using iteration steps

```
figure(1)
xaxis_GD = 1:iterations;
yaxis_GD = log_error;
[numRowsX,numColsX] = size(xaxis_GD);
[numRowsY,numColsY] = size(yaxis_GD);
% +2 for removing the initial value for log_error and keep the last
 element
% + 2 element
rows_to_delete=(numColsX+2:numColsY); % Delete the 0 rows
yaxis_GD(:,rows_to_delete)=[];
yaxis_GD(:,(1))=[];
plot(xaxis_GD,yaxis_GD)
title("Experimental Convergence Rate of Gradient Descent")
xlabel("Iteration");
ylabel("log||x-x*||_2");
```



Problem 2: Conjugate Gradient Algorithm

```
tic
x = initial_x; % Capture the initial solution
iterations = 0; % Initial iteration
```

```
residual_norm = norm(b-A*x); % Define initial gradient norm
r = b-A*x; % Residual value
p = r;
log_error = zeros(1,max_iterations);

% Reference for code in this link from the Wikipedia page:
% https://en.wikipedia.org/wiki/
Conjugate_gradient_method#Example_code_in_MATLAB_/_GNU_Octave_2
% If residual_norm is sufficiently small, then return optimal x and
 exit the loop
while iterations < max_iterations && residual_norm > tol
% First compute the alpha where r_i is orthogonal to r_j which has the
 property of r^T_i r_j =0
% and p_i is A-orthogonal to p_j where p^T_i A p_i = 0 with i and j is
 not equal.
% Thus input vector x_0 or at iteration 1 can be approximate initial
 solution or 0.
alpha=r'*r/(p'*A*p);
x = x + alpha*p;
r_new = r - alpha*A*p;
residual_norm = norm(r_new);
beta = r_new'*r_new/(r'*r);
p_new = r_new + beta*p;
p = p_new;
r = r_new;
iterations = iterations + 1;
log_error(iterations+1)=log(norm(x-Xoptimal));
end
toc
```

```
Elapsed time is 0.008290 seconds.
```

Extract optimal x has just been generated

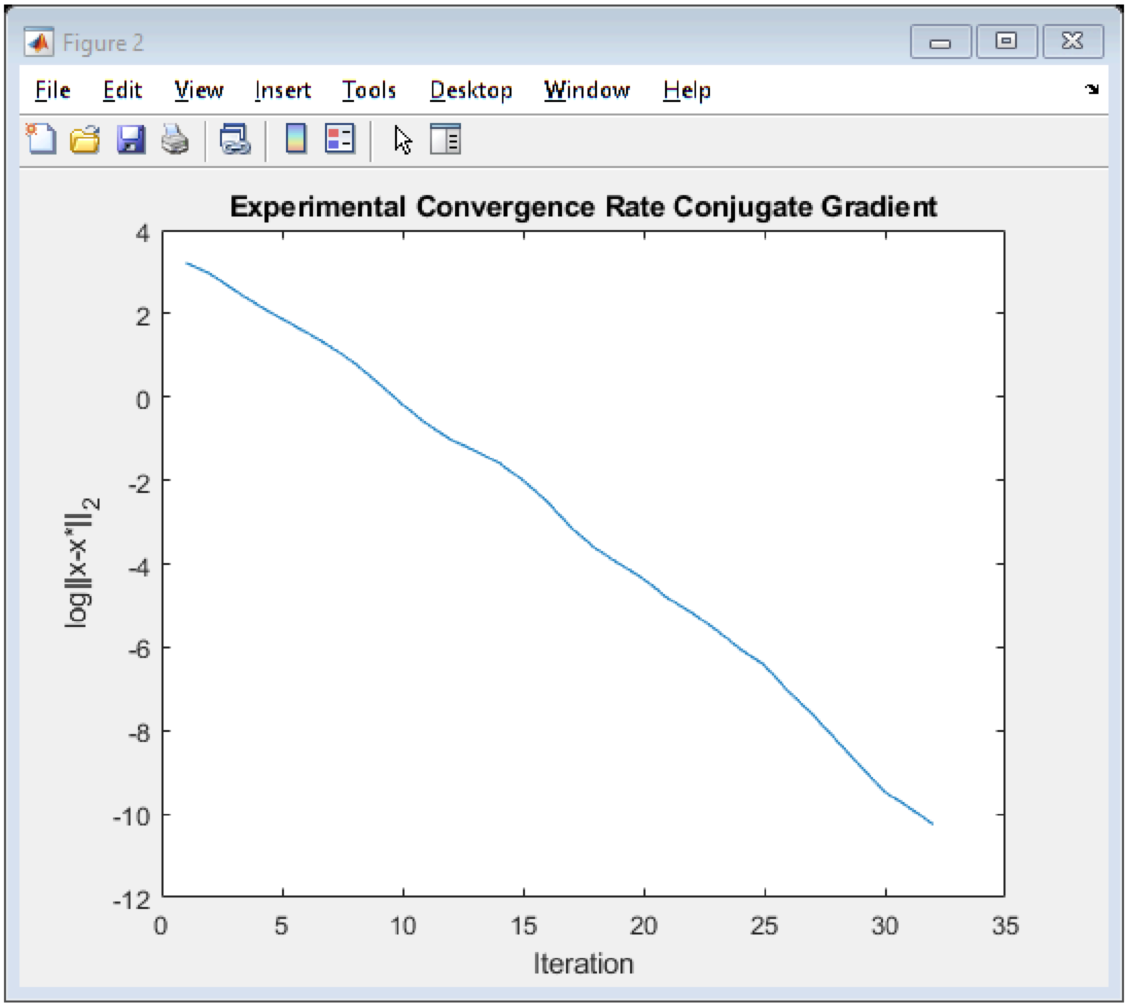
```
x_optimal_CG = x;
disp('Loop ends in the number of iterations:');
disp(iterations);
disp('Norm of optimal gradient is:');
disp(gradient_norm);
```

```
Loop ends in the number of iterations:
    32

Norm of optimal gradient is:
   9.8566e-06
```

Plotting for Convergence Rate using iteration steps

```
figure(2)
xaxis_CG = 1:iterations;
yaxis_CG = log_error;
[numRowsX,numColsX] = size(xaxis_CG);
[numRowsY,numColsY] = size(yaxis_CG);
```

```
% +2 for removing the initial value for log_error and keep the last
 element
% + 2 element
rows_to_delete=(numColsX+2:numColsY); % Delete the 0 rows
yaxis_CG(:,rows_to_delete)=[];
yaxis_CG(:,(1))=[];
plot(xaxis_CG,yaxis_CG)
title("Experimental Convergence Rate Conjugate Gradient")
xlabel("Iteration");
ylabel("log||x-x*||_2");
```



Problem 3: Nesterov's Algorithm

```
tic
x = initial_x; % Capture the initial solution
iterations = 0; % Initial iteration
max_iterations = 1e4; % maximum number of iterations
tol = 1e-5; % tolerance value
alpha = 1/norm(A); % step size

gradient_norm = norm(A*x-b); % Define initial gradient norm
```

```
% Log error for plotting convergence rate
log_error = zeros(1,max_iterations);
t=1; % Initial t = 1 and x0 = y0;
y0 = x;

% The reference formula is taken from this source
% https://www.cs.rochester.edu/u/jliu/CSC-576/class-note-9.pdf

while iterations < max_iterations && gradient_norm > tol
x_new = y0-alpha*grad_function(y0,A,b);
t_new = 0.5+0.5*sqrt(1+4*t^2);
y0 = x_new + (t-1)/t_new*(x_new-x);
% Increase the iteration
iterations = iterations + 1;
% Update the variable x for next iteration
x = x_new;
t = t_new;
gradient_norm = norm(A*x-b);
log_error(iterations+1)=log(norm(x-Xoptimal));
end
toc
```

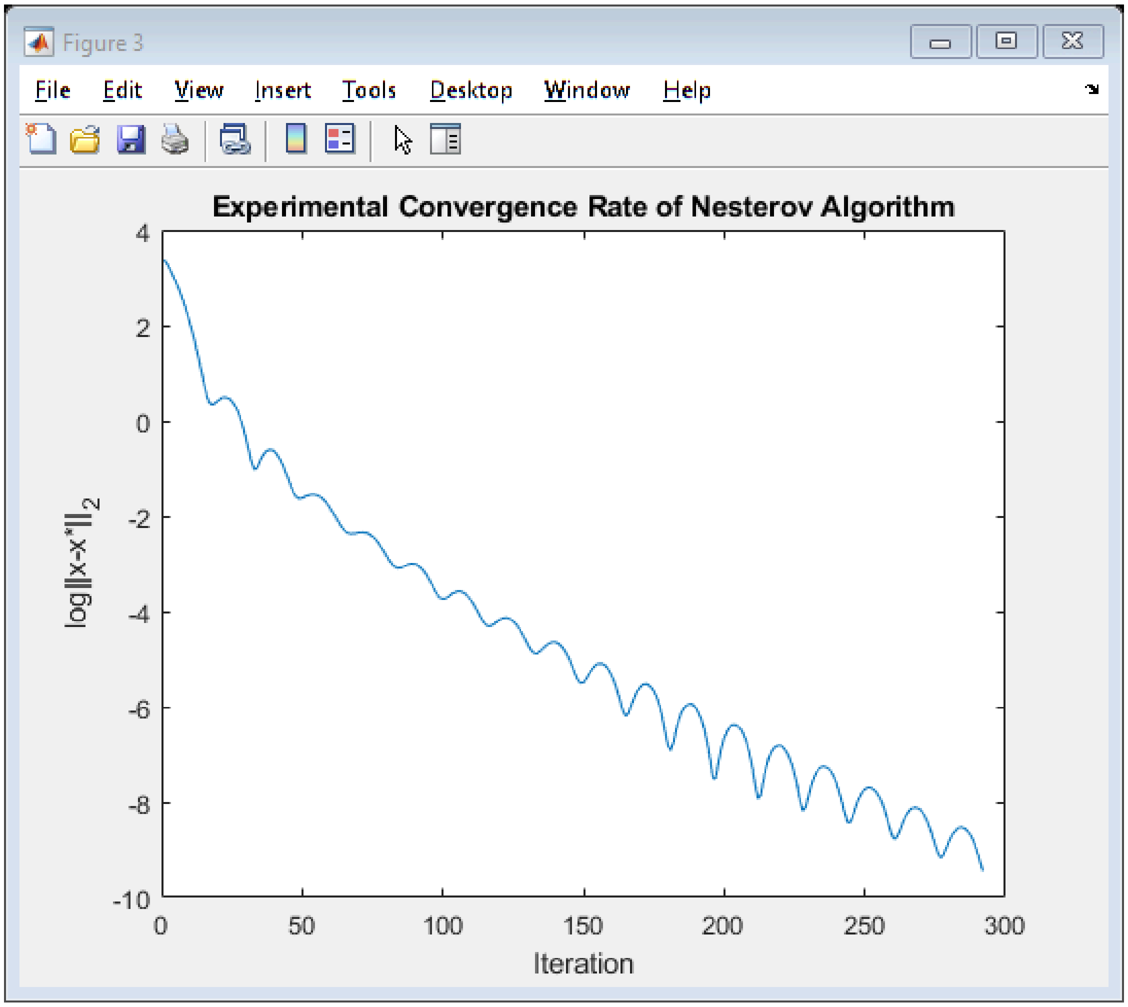
```
Elapsed time is 0.061618 seconds.
```

```
x_optimal_N = x;
disp('Loop ends in the number of iterations:');
disp(iterations);
disp('Norm of optimal gradient is:');
disp(gradient_norm);
```

```
Loop ends in the number of iterations:
   292

Norm of optimal gradient is:
   8.8281e-06
```

Plotting for Convergence Rate using iteration steps

```
figure(3)
xaxis_N = 1:iterations;
yaxis_N = log_error;
[numRowsX,numColsX] = size(xaxis_N);
[numRowsY,numColsY] = size(yaxis_N);
% +2 for removing the initial value for log_error and keep the last
 element
% + 2 element
rows_to_delete=(numColsX+2:numColsY); % Delete the 0 rows
yaxis_N(:,rows_to_delete)=[];
yaxis_N(:,(1))=[];
plot(xaxis_N,yaxis_N)
title("Experimental Convergence Rate of Nesterov Algorithm")
xlabel("Iteration");
ylabel("log||x-x*||_2");
```

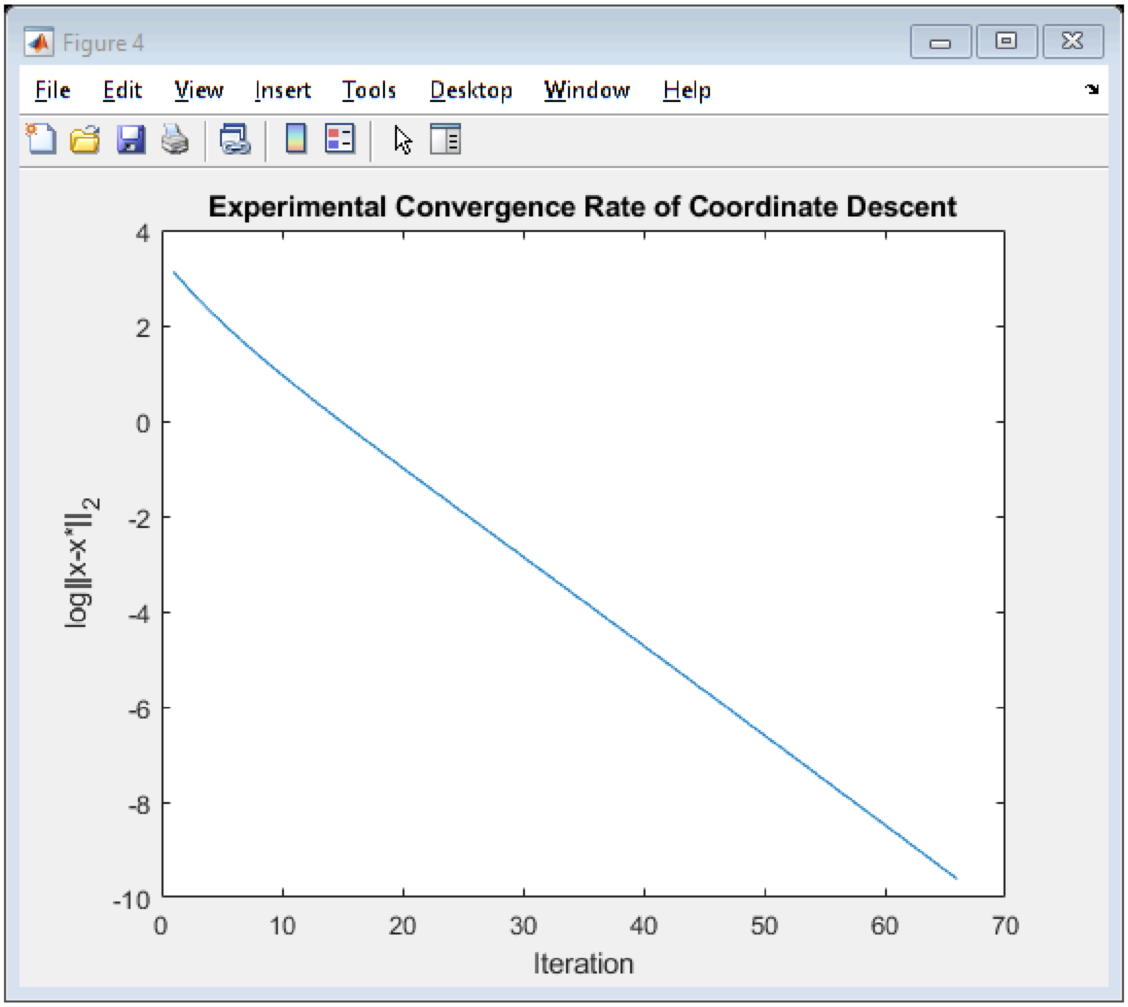


Problem 4: Coordinate Descent

```
tic
x = initial_x; % Capture the initial solution
iterations = 0; % Initial iteration
max_iterations = 1e4; % maximum number of iterations
tol = 1e-5; % tolerance value

gradient_norm = norm(A*x-b); % Define initial gradient norm
% Log error for plotting convergence rate
log_error = zeros(1,max_iterations);
% For this problem, the deterministic approach is used to compute the
% algorithm
while iterations < max_iterations && gradient_norm > tol
    for i = 1 : size(x,1)
        % First compute the gradient of function
        gradient = grad_function(x,A,b);
        % Extract the gradient of single coordinate and update it
        x(i,1) = x(i,1) - gradient(i);
    end
    gradient_norm = norm(A*x-b); % Gradient Norm
    % Increase the iteration
    iterations = iterations + 1;
    log_error(iterations+1)=log(norm(x-Xoptimal));
end
toc

Elapsed time is 1.014685 seconds.

x_optimal_CD = x;
disp('Loop ends in the number of iterations:');
disp(iterations);
disp('Norm of optimal gradient is:');
disp(gradient_norm);

Loop ends in the number of iterations:
    66

Norm of optimal gradient is:
   8.3974e-06
```

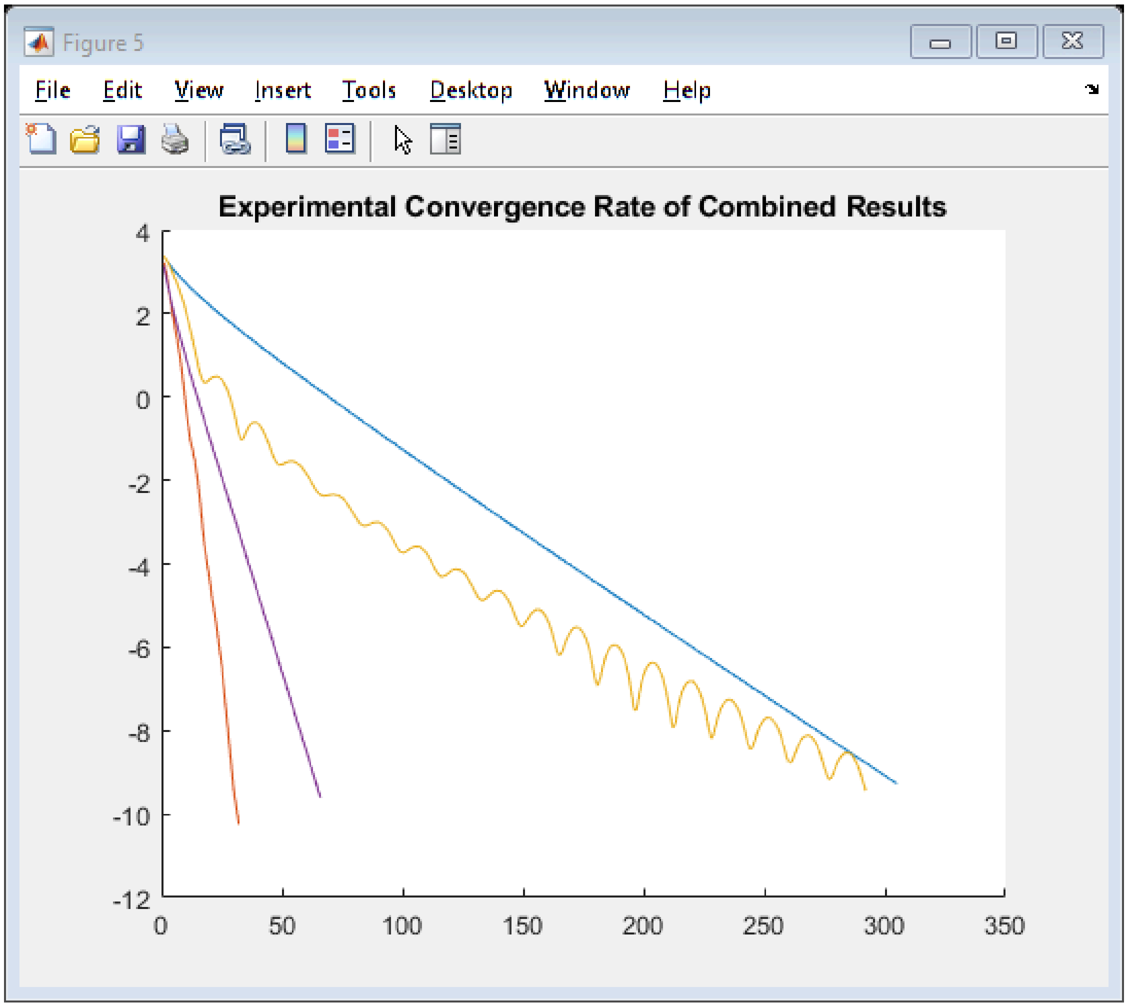
Plotting for Convergence Rate using iteration steps

```
figure(4)
xaxis_CD = 1:iterations;
yaxis_CD = log_error;
[numRowsX,numColsX] = size(xaxis_CD);
[numRowsY,numColsY] = size(yaxis_CD);
% +2 for removing the initial value for log_error and keep the last
 element
% + 2 element
rows_to_delete=(numColsX+2:numColsY); % Delete the 0 rows
yaxis_CD(:,rows_to_delete)=[];
yaxis_CD(:,(1))=[];
plot(xaxis_CD,yaxis_CD)
title("Experimental Convergence Rate of Coordinate Descent")
xlabel("Iteration");
ylabel("log||x-x*||_2");
```



Combined Result Plot

```
figure(5)
title("Experimental Convergence Rate of Combined Results")
hold on
plot(xaxis_GD,yaxis_GD)
plot(xaxis_CG,yaxis_CG)
plot(xaxis_N,yaxis_N)
plot(xaxis_CD,yaxis_CD)
hold off
```



Problem 5: Comparisons.

Iterations required and the overall computation time for different methods

Gradient Descent: require 305 iterations and roughly 0.008s for solving optimization problem above

Conjugate Gradient Algorithm: require 32 iterations and roughly 0.006s for solving optimization problem above

Nesterov's Algorithm: require 292 iterations and roughly 0.008s for solving optimization problem above

Coordinate Descent: require 66 iterations and roughly 0.033s for solving optimization problem above

In conclusion, the coordinate descent seems to stand out of other algorithms since it took shorter iterations to compute the optimal value. Thus, it also converge quickly for coordinatewise minimization to solve the linear systems. However, it needs to trade of with time computation since for each iteration, it have to update each coordinate or single element from variable X individually. This can be improved by using the Block Coordinate Descent(BCD) and other advanced variants of the algorithm.

Futhermore, in the combined plot, the Gradient Descent has a linear convergence rate, whereas for the Conjugate Gradient, the convergence rate this algorithm is faster, and it has the linear convergence rate with formula:

$$1 - \frac{2}{\sqrt{conditionnumber}}$$

Lastly, the Nesterov algorithm has faster convergence rate compared to Gradient Descent and it converges at rate 1/t^2.

Opinion

I really love the problem of assignment which illustrate different approaches and its advantages on the optimization problems. However, it took me a lot of time for learning Matlab commands as well the formula and implementations for different algorithms. Within 1 week, it's really hard to understand all the theory as well as proof of concept. But in the end, I'm glad that I can take away knowledge related to the pros and cons of different algorithms and its convergence rates.

Appendix (Function)

Let define the gradient function

```
function gradient = grad_function(x,A,b)
    gradient = A*x - b;
end
```

```
Elapsed time is 0.055962 seconds.
```

Published with MATLAB® R2019b