Aalto University
School of Electrical
Engineering

# ELEC-E5431 - Large scale data analysis
# D

Homework Assignment 1

Name: Xingji Chen

Student ID: 101659554

E-mail: xingji.chen@aalto.fi

Aalto University
School of Electrical
Engineering

**Problem 1**

The optimization problem to be addressed is a simple quadratic function minimization, that is,

$$\min_{\mathbf{x}} \ \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$$

where the matrix **A** and vector **b** are appropriately generated. Use the same **A** and **b** while comparing different methods.

Hints:

- A should be such that $\mathbf{x}^T\mathbf{A}\mathbf{x}$ is non-negative, that is, **A** is positive semi-definite, and **b** should be in the range of **A**.
- In fact, by solving the above unconstrained minimization problem, you solve a system of linear equations $\mathbf{A}\mathbf{x} = \mathbf{b}$. Indeed, the gradient of the objective function is $\mathbf{A}\mathbf{x}-\mathbf{b}$, and it should be equal to 0 at optimality. Thus, you can find optimal $\mathbf{x}^*$using back-slash (or matrix inversion followed by computing the product $\mathbf{A}^{-1}\mathbf{b}$) operators in MATLAB. The optimal objective value can be then obtained by simply substituting such **x** into the objective of the above optimization problem. It is suitable for small and mid size problems, but the matrix inversion is prohibitively too expensive to be able to solve a system of linear equations for large scale problems. Thus, the only option for large scale problems is the use of algorithms that you implement in this assignment!

To be able to produce convergence figures for the algorithms that you test, let the dimension of **x** be 100 variables or few 100's (but after producing the figures also play with higher dimensions to see when the matrix inversion fails, but the large scale optimization methods still work fine and some also quite fast).

Set the tolerance parameter for the stopping criterion for checking the convergence to $10^{-5}$. For example, check if $\|\nabla f(\mathrm{x})\| \leq 10^{-5}$, and limit the total number of iterations by 5000 if the predefined tolerance is still not achieved.

Aalto University
School of Electrical
Engineering

**Task 1:** *Gradient Descent Algorithm.* Implement Gradient Descent Algorithm for solving the above optimization problem. Use correctly selected fixed step size. Draw the experimental convergence rate, i.e., draw the convergence plot of log $|f(\mathbf{x}^t) - f^*|$ versus iteration count $t$, for the algorithm, and compare it to the theoretically predicted one. Because, we know that

$$f\left(\mathbf{x}^t\right) - f^* \leq \frac{1}{2\mu}\left\|\nabla f\left(\mathbf{x}^t\right)\right\|^2$$

you can equivalently draw

$$\log\left\|\nabla f\left(\mathbf{x}^t\right)\right\| = \log\left\|\mathbf{A}\mathbf{x}^t - \mathbf{b}\right\|$$

versus iteration count $t$.

**Task 2:** *Conjugate Gradient Algorithm.* Implement Conjugate Gradient Algorithm (page 11 in Summary Notes) for solving the above optimization problem. Draw the experimental convergence rate for the algorithm, and compare it to the theoretically predicted one.

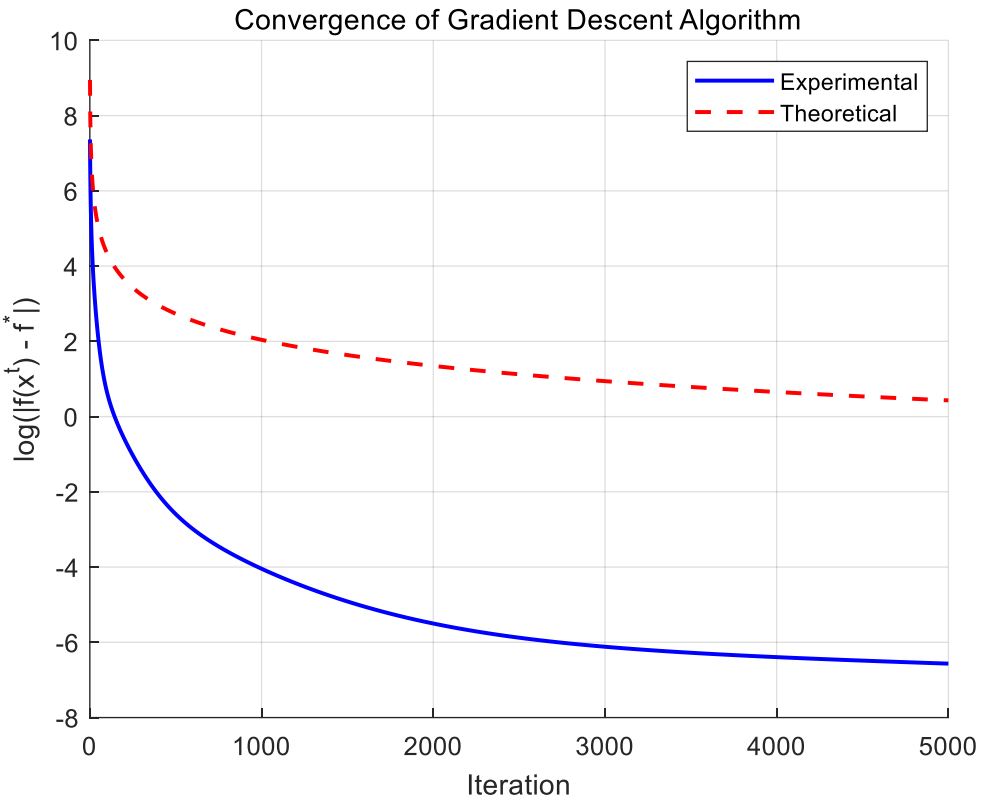
**Task 3:** *FISTA.* Implement FISTA (page 12 in Summary Notes) for solving the above optimization problem. Draw the experimental convergence rate for the algorithm, and compare it to the theoretically predicted one.

**Task 4:** *Coordinate Descent.* Implement the Coordinate Descent method page 20 in Summary Notes) for solving the above optimization problem. Draw the experimental convergence rate for the algorithm, and compare it to the theoretically predicted one.

**Task 5:** *Comparisons and Conclusion.* Compare the results (in terms of the iterations required and the overall computation time) for different methods (including, for example, the standard MATLAB back-slash operator) and draw your overall conclusions. Observe up to which dimension Python or MATLAB still can invert a matrix, that is, define the dimension after which the problem turns to be large scale in the context of your implementation.

Aalto University
School of Electrical
Engineering

**Task 1**




Computation time: 0.403137 s.

For the Gradient Descent Algorithm, the matrix **A** and vector **b** are initially populated with random values, and **A** is made symmetric positive definite by multiplying it with its transpose. The initial guess vector **x** is set to a zero vector. The algorithm calculates the eigenvalues of **A** to determine the step size $\alpha$ and the condition number of **A**, which are used to adjust the steps of the gradient descent for optimal convergence speed.

The main loop of the algorithm iteratively updates the solution vector **x** by moving in the direction opposite to the gradient of the quadratic function $f(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$. This update is based on the calculated step size. The loop continues until the norm of the gradient is less than a set tolerance (*tol*) or the maximum number of iterations (*maxIt*) is reached.

Throughout the iterations, the algorithm records the value of the objective function $f(\mathbf{x})$ at each step. These values are used to evaluate the experimental convergence of the algorithm by plotting the logarithm of the absolute difference between the current function value and the optimal function value $f^*$, where $f^*$ is calculated from the exact solution $\mathbf{x}_{\text{opt}}$ obtained by directly solving $\mathbf{Ax} = \mathbf{b}$, i.e., $f(x) = \frac{1}{2}\mathbf{x}_{opt}^T \mathbf{A}\mathbf{x}_{opt} - \mathbf{b}^T\mathbf{x}_{opt}$.

The algorithm also calculates a theoretical convergence rate based on the step size and initial error, and plots this alongside the experimental convergence for comparison. The goal is to visually assess how well the gradient descent algorithm performs in terms of approaching the minimum function value $f^*$ through iterations.

Finally, the total execution time is measured and displayed using MATLAB's `tic` and `toc` functions, providing a simple benchmark of the algorithm's efficiency.

MATLAB code:

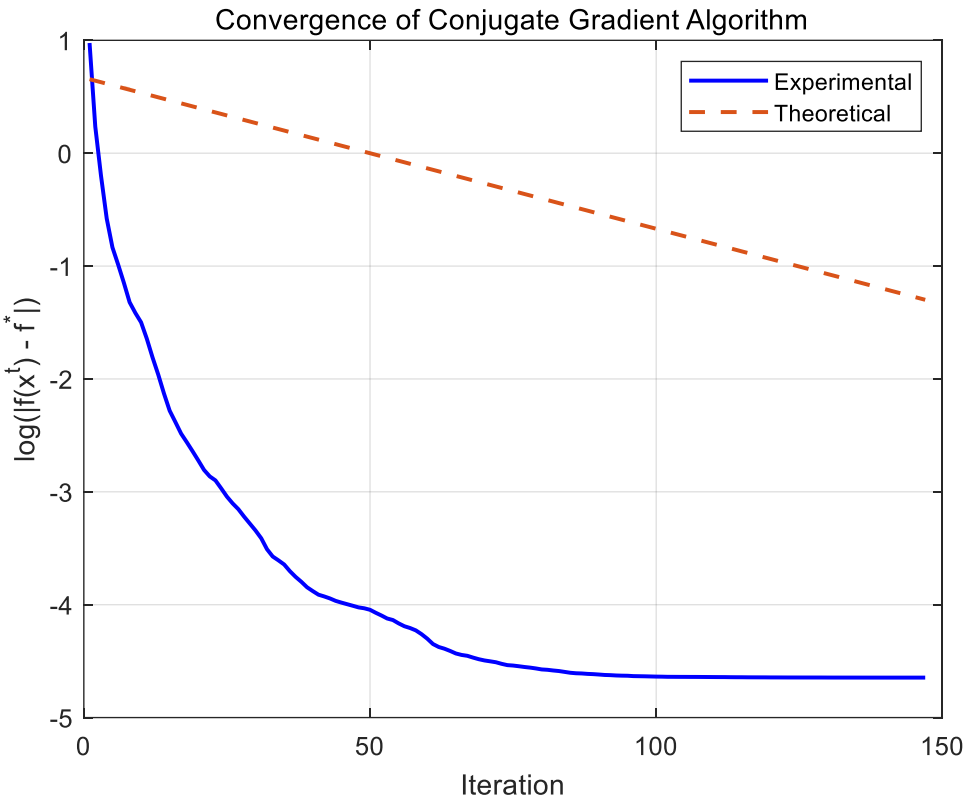
```
clear all;
clc;
tic;

% Initialization
n = 100;
A = randn(n, n);
A = A * A'; % Ensuring A is symmetric positive definite
b = A * randn(n, 1);
x = zeros(n, 1);

% Calculating eigenvalues for step size and condition number
eigVals = eig(A);
minEig = min(eigVals);
maxEig = max(eigVals);
alpha = 1 / maxEig;
kappa = maxEig / minEig;

% Tolerance and max iterations setup
tol = 1e-5;
maxIt = 5000;
```

Aalto University
School of Electrical
Engineering

```
% Exact solution and its function value for convergence analysis
xOpt = A\b;
fOpt = 1/2 * xOpt' * A * xOpt - b' * xOpt;

% Gradient norm and iteration counter initialization
gradNorm = norm(A * x - b);
iter = 0;

% Arrays for function values and gradient norms
fVals = [];
gNorms = [];

% Gradient descent loop
while(gradNorm > tol && iter < maxIt)
    grad = A * x - b; % Gradient computation
    x = x - alpha * grad; % Solution update
    f = 1/2 * x' * A * x - b' * x; % Current function value
    fVals = [fVals; f];
    gradNorm = norm(grad); % Gradient norm update
    % gNorms = [gNorms; gradNorm];
    iter = iter + 1;
end

% Data preparation for plotting
iters = 1:iter;
expConv = log(fVals - fOpt);
theoConv = log(2 / alpha * norm(zeros(n, 1) - xOpt) ./ iters);

% Plotting convergence analysis
figure;
hold on;
plot(expConv, 'b', 'LineWidth', 1.5);
plot(theoConv, 'r--', 'LineWidth', 1.5);
xlabel('Iteration');
ylabel('log(|f(x^t) - f^*|)');
legend('Experimental', 'Theoretical');
title('Convergence of Gradient Descent Algorithm');
grid on;
hold off;
toc;
```

Aalto University
School of Electrical
Engineering

**Task 2**




Computation time: 0.380546 s.

For the Conjugate Gradient (CG) algorithm, it starts with initializing parameters including the dimension $n$ of the matrix **A** and the solution vector **x**, and generates **A** and a random vector **b**. Then, an initial guess for the solution **x**, the initial residual **r**, and the initial direction **p** based on the residual are set up for the CG algorithm.

The main loop of the CG algorithm iteratively updates the solution **x** by calculating the step size $\alpha$, which is derived using the squared norm of the residual (*rsOld*) and the dot product of **p** and **Ap**. The solution **x** is updated along the direction **p** by this step size. The residual **r** and the squared residual norm (*rsNew*) are accordingly updated to reflect the new solution. The objective function value at the new **x** is calculated, its reciprocal is taken, and then its logarithm is stored for plotting the experimental convergence.

Moreover, a theoretical convergence rate is calculated based on the condition number derived from the eigenvalues of **A**. This rate is used for comparison with the experimental convergence and plotted. The loop terminates when the squared residual norm is less than a set tolerance (*tol*), indicating convergence, or after reaching the maximum number of iterations (*maxIt*).

Finally, the relationship between the logarithm of the absolute difference of the current and optimal objective function values and the iteration number is plotted to demonstrate the experimental and theoretical convergence of the CG algorithm. At the same time, MATLAB's `tic` and `toc` functions are used to measure and display the total execution time of the script.

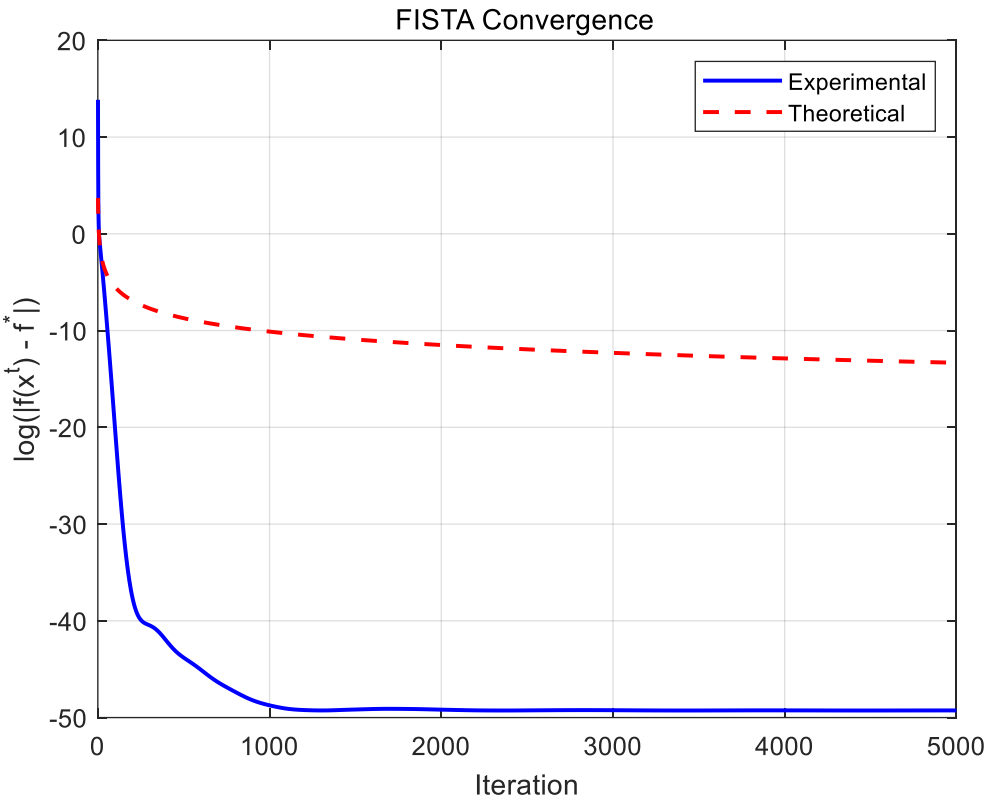
MATLAB code:

```
clear all;
clc;
tic;

% Initialize parameters
n = 100; % Dimension of x and A
A = randn(n, n);
A = A'*A; % Ensuring A is symmetric positive definite
b = randn(n, 1); % Random vector b

% Conjugate Gradient (CG) Algorithm setup
x = zeros(n, 1); % Initial guess for the solution
r = b - A*x; % Initial residual
p = r; % Initial direction
rsOld = r'*r; % Initial squared residual norm
tol = 1e-5; % Tolerance for convergence
maxIt = 5000; % Maximum number of iterations
expConv = []; % To store objective values for plotting

% Theoretical convergence rate setup
eigVals = eig(A);
minEig = min(eigVals);
maxEig = max(eigVals);
alpha = 1 / maxEig;
```

```
kappa = maxEig / minEig; % Condition number of A
theoConv = []; % To store theoretical convergence rates

% CG Algorithm main loop
for i = 1:maxIt
    Ap = A * p;
    alpha = rsOld / (p'*Ap); % Step size
    x = x + alpha * p; % Update solution
    r = r - alpha * Ap; % Update residual
    rsNew = r'*r; % Update squared residual norm
    % Compute objective function value at x
    objVal = 0.5 * x' * A * x - b' * x;
    objVal = 1 / objVal;
    objVal = log(abs(objVal));
    expConv = [expConv; objVal]; % Store for plotting
    % Compute and store theoretical convergence rate
    if i == 1
        e0 = objVal; % Initial error for theoretical rate calculation
    end
    theoConv = [theoConv; log(abs(e0 * (2 * ((sqrt(kappa) - 1) /
(sqrt(kappa) + 1))^i)))];
    if rsNew < tol % Check convergence
        break;
    end
    p = r + (rsNew/rsOld) * p; % Update direction
    rsOld = rsNew; % Update squared residual norm for next iteration
end

% Plotting convergence
figure;
plot(expConv, 'b', 'LineWidth', 1.5); % Experimental convergence
hold on;
plot(theoConv, '--', 'LineWidth', 1.5); % Theoretical convergence
xlabel('Iteration');
ylabel('log(|f(x^t) - f^*|)');
title('Convergence of Conjugate Gradient Algorithm');
legend('Experimental', 'Theoretical');
grid on;
toc;
```

Aalto University
School of Electrical
Engineering

**Task 3**




Computation time: 0.404701 s.

The FISTA process starts with an initial guess $\mathbf{x}_0$ and sets up variables for the algorithm's iterations, including the Lipschitz constant $L$ derived from the eigenvalues of **A**, which is used to calculate the step size for the gradient descent step. The main loop of FISTA iteratively updates the solution vector **x** through a gradient step, followed by a step that updates the variables **y** and $t$, which are used to accelerate the convergence of the algorithm.

At each iteration, the objective function value is calculated and stored, providing a historical record of these values (*expConv*) for subsequent analysis. Similarly, a theoretical convergence rate (*theoConv*) is calculated and stored based on the initial objective function value and the number of iterations. The loop continues until the

solution converges within a specified tolerance (*tol*) or reaches the maximum number of iterations (*maxIt*).

After completing the iterations or achieving convergence, plots of the experimental convergence (objective function values) and the theoretical convergence rate (log-transformed) are drawn to visually assess the performance and efficiency of the FISTA algorithm.

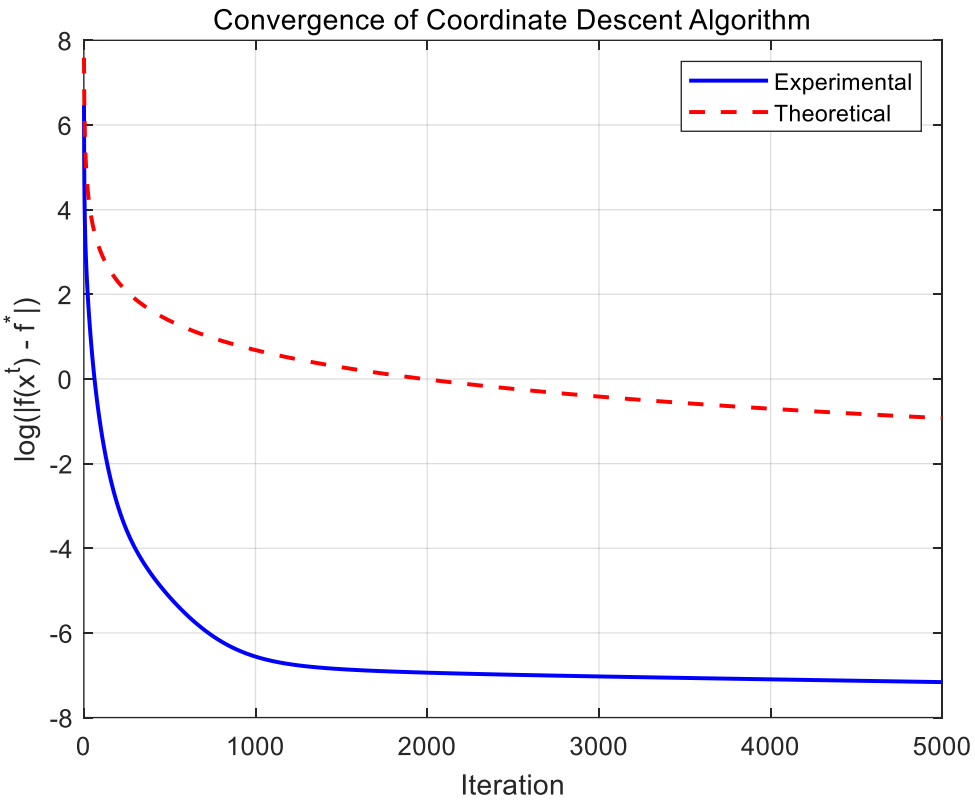
MATLAB code:

```
clear all;
clc;
tic;

% Parameters
n = 100; % Dimension of x
A = randn(n);
A = A'*A; % Making A positive semi-definite
b = randn(n, 1);
x0 = eye(n, 1); % Initial guess
L = max(eig(A)); % Lipschitz constant
maxIt = 5000; % Maximum number of iterations
tol = 1e-5; % Convergence tolerance

% FISTA initialization
y = x0;
t = 1;
x = x0;
expConv = zeros(maxIt, 1); % To store objective function values
theoConv = zeros(maxIt, 1); % For storing theoretical convergence rate

% Initial objective value for theoretical rate calculation
fInitial = 0.5 * x0' * A * x0 - b' * x0;

% FISTA main loop
for k = 1:maxIt
    xOld = x;

    % Gradient step
```

```
    grad = A*y - b;
    x = y - (1/L) * grad;

    % Update t and y for the next iteration
    tOld = t;
    t = (1 + sqrt(1 + 4*t^2)) / 2;
    y = x + ((tOld - 1) / t) * (x - xOld);

    % Record the history (objective function value)
    expConv(k) = 0.5 * x' * A * x - b' * x;

    % Theoretical convergence rate
    theoConv(k) = fInitial * (1 / (k^2));

    % Check convergence
    if norm(A*x - b) <= tol
        expConv = expConv(1:k); % Truncate history to the current
iteration
        theoConv = theoConv(1:k); % Truncate theoretical rate to the
current iteration
        break;
    end

end

% Plot the convergence
figure;
plot(expConv, 'b', 'LineWidth', 1.5);
hold on;
theoConv = log(theoConv);
plot(theoConv, 'r--', 'LineWidth', 1.5);
hold off; % Make sure to turn hold off after plotting
title('FISTA Convergence');
xlabel('Iteration');
ylabel('log(|f(x^t) - f^*|)');
legend('Experimental', 'Theoretical');
grid on;
toc;
```

Aalto University
School of Electrical
Engineering

**Task 4**




Computation time: 1.816308 s.

The Coordinate Descent Algorithm is utilized for optimizing a quadratic function $f(x) = \frac{1}{2}\mathbf{x}^T\mathbf{A}\mathbf{x} - \mathbf{b}^T\mathbf{x}$. The process starts by setting the dimension $n$ of the solution vector $\mathbf{x}$, generating a random matrix $\mathbf{A}$ and ensuring it is positive semi-definite by squaring it, and generating a random vector $\mathbf{b}$. The algorithm calculates the maximum eigenvalue of $\mathbf{A}$ to determine the step size $\alpha$ for updates.

The Coordinate Descent Algorithm iteratively updates each component of the solution vector $\mathbf{x}$ by minimizing the objective function with respect to that component while keeping the others fixed. This is achieved by solving for $\mathbf{x}_i$ in each iteration for every $i$ from 1 to $n$, thus cycling through all coordinates.

After completing a full cycle through all coordinates, the objective value is calculated and stored for plotting the algorithm's convergence towards the optimal solution. Convergence is checked by comparing the logarithm of the absolute difference between the current objective value and the optimal objective value *yOpt*, calculated from the exact solution *xOpt*, against a tolerance (*tol*). The algorithm terminates if this criterion is met or after reaching the maximum number of iterations (*maxIt*).

Finally, the code plots the experimental convergence (the logarithm of the difference between the objective values and *yOpt* and the theoretical convergence rate. This plot demonstrates the speed at which the algorithm approaches the optimal solution throughout the iterations.

MATLAB code:

```
clear all;
clc;
tic;

n = 100; % Dimension of x as specified in the task
A = randn(n, n);
A = A' * A; % Make A positive semi-definite
b = A * randn(n, 1); % Generate a random vector
eigVals = eig(A);
minEig = min(eigVals);
maxEig = max(eigVals);
alpha = 1 / maxEig;

% Coordinate Descent parameters
x = zeros(n, 1); % Initial guess
tol = 1e-5;
maxIt = 5000;
objVals = []; % To store objective values for plotting convergence
xOpt = A \ b;
yOpt = 0.5 * xOpt' * A * xOpt - b' * xOpt;

% Coordinate Descent Algorithm
for iter = 1:maxIt
    for i = 1:n
```

```
        x(i) = (b(i) - A(i, [1:i-1, i+1:n]) * x([1:i-1, i+1:n])) / A(i,
i);
    end

    % Calculate objective value
    objVal = 0.5 * x' * A * x - b' * x;
    objVals = [objVals; objVal];

    % Check for convergence (this is a simple criterion, in practice,
might need a more robust one)
    if iter > 1 && abs(log(abs(objVals(end) - yOpt))) < tol
        break;
    end
end
iters = 1:iter;
theoConv = log((1/alpha * norm(zeros(n, 1) - xOpt)/2./iters));
expConv = log(objVals - yOpt);

% Plot the convergence
figure;
plot(expConv, 'b', 'LineWidth', 1.5);
hold on;
plot(theoConv, 'r--', 'LineWidth', 1.5);
xlabel('Iteration');
ylabel('log(|f(x^t) - f^*|)');
title('Convergence of Coordinate Descent Algorithm');
legend('Experimental', 'Theoretical');
grid on;
toc;
```

**Task 5**

| Method | Computation time |
|---|---|
| Gradient Descent Algorithm | 0.403137 s |
| Conjugate Gradient Algorithm | 0.380546 s |
| FISTA | 0.404701 s |
| Coordinate Descent | 1.816308 s |
| Standard MATLAB back-slash operator | 0.193993 s |

From the convergence plots, it can be observed that the Conjugate Gradient Algorithm and FISTA exhibit a rapid initial error decrease during the initial iterations, indicating a faster convergence towards the optimal solution compared to Gradient Descent and Coordinate Descent. However, as the number of iterations increases, all methods converge to the solution.

In terms of computation time, Coordinate Descent is the slowest, significantly longer than the other methods. This may be due to the fact that Coordinate Descent updates one coordinate at a time, which is inherently a sequential process and may not effectively utilize parallel computation in problems with certain sparse patterns in the data. On the other hand, the standard MATLAB backslash operator (\) is the fastest, which is expected as it is a highly optimized method for solving linear systems.

In conclusion, when applicable, the standard MATLAB backslash operator is very efficient for direct solutions. The Conjugate Gradient Algorithm and FISTA offer a balance between convergence speed and computation time, especially in large-scale problems where direct methods are not feasible. Gradient Descent is simple and widely applicable but may require more iterations to converge, while Coordinate Descent, despite its slow computation time, is effective for problems with specific sparsity patterns in the data.

As the dimension of the matrix increases, the computation time for the program also gradually increases. For matrix inversion operations, when the matrix dimensions are 100, 1000, and 10000, the computation times are respectively 0.010282 seconds, 0.029784 seconds, and 11.148903 seconds. When the matrix dimension increases to 30000, the computation time becomes 369.524536 seconds. With the matrix dimension at 50000, the computation time rises to 1112.056496 seconds. As the matrix dimension continues to increase, inverting the matrix becomes difficult, and beyond these sizes, the problem is considered large-scale, where direct inversion becomes impractical both in terms of memory usage and computation time.

Aalto University
School of Electrical
Engineering

**Problem 2**

Projection onto intersection of convex sets. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be closed convex sets such that $\mathcal{C}_1 \bigcap \mathcal{C}_2 \neq \varnothing$.

Give a complete proof that for the following problem

$$\text{find } \mathbf{x} \in \mathcal{C}_1 \bigcap \mathcal{C}_2$$

the subgradient method with Polyak's stepsize rule is equivalent to the following alternating projections:

$$\mathbf{x}^{t+1} = \mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^t\right),\ \mathbf{x}^{t+2} = \mathcal{P}_{\mathcal{C}_2}\left(\mathbf{x}^{t+1}\right)$$

Almost complete proof can be found in the lecture notes on Subgradient Methods, but explain it in more detail (how you understand it) and prove also (by straightforward calculation) the equality

$$\nabla\left(\frac{1}{2}\text{dist}^2_{\mathcal{C}_i}\left(\mathbf{x}^t\right)\right) = \mathbf{x}^t - \mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^t\right)$$

that is not proven there. Here $\text{dist}_{\mathcal{C}}\left(\mathbf{x}\right) = \min_{\mathbf{z}\in\mathcal{C}} \left\|\mathbf{x}-\mathbf{z}\right\|_2$

➢

For any convex set $\mathcal{C}$ and any $\mathbf{x}$, the projection of $\mathbf{x}$ onto $\mathcal{C}$, $\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)$ is defined as the unique point in $\mathcal{C}$ that is closest to $\mathbf{x}$ in the Euclidean norm. Formally,

$$\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right) = \arg\min_{\mathbf{z}\in\mathcal{C}} \left\|\mathbf{x}-\mathbf{z}\right\|_2$$

The squared distance to a convex set $\mathcal{C}$ from a point $\mathbf{x}$ is given by

$$\text{dist}^2_{\mathcal{C}}\left(\mathbf{x}\right) = \left\|\mathbf{x}-\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)\right\|_2^2$$

The gradient (in the case where it exists) or subgradient (in the general case) of this function can be computed. If $\mathbf{x} \notin \mathcal{C}$, then the subgradient of $\text{dist}^2_{\mathcal{C}}\left(\mathbf{x}\right)$ is given by:

$$\nabla\left(\frac{1}{2}\text{dist}_{\mathcal{C}}^{2}\left(\mathbf{x}^{t}\right)\right)=\mathbf{x}-\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)$$

The subgradient method updates the current point $\mathbf{x}_t$ by moving in the direction of the negative subgradient of the function being minimized. With Polyak's stepsize rule, the update is:

$$\mathbf{x}^{t+1}=\mathbf{x}^{t}-\alpha_{t}\left(\mathbf{x}^{t}-\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}^{t}\right)\right)$$

where $\alpha_t$ is the stepsize chosen according to some rule.

The alternating projections method iteratively projects onto convex sets $\mathcal{C}_1$ and $\mathcal{C}_2$, i.e.,

$$\mathbf{x}^{t+1}=\mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^{t}\right)$$
$$\mathbf{x}^{t+2}=\mathcal{P}_{\mathcal{C}_2}\left(\mathbf{x}^{t+1}\right)$$

To prove equivalence, we need to show that a step of the subgradient method with Polyak's rule is the same as an alternating projection step. Let's consider the update from $\mathbf{x}_t$ to $\mathbf{x}_{t+1}$ and $\mathbf{x}_{t+1}$ to $\mathbf{x}_{t+2}$ in both methods.

Using Polyak's stepsize, we have:

$$\mathbf{x}^{t+1}=\mathbf{x}^{t}-\alpha_{t}\left(\mathbf{x}^{t}-\mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^{t}\right)\right)$$
$$\mathbf{x}^{t+2}=\mathbf{x}^{t+1}-\alpha_{t+1}\left(\mathbf{x}^{t+1}-\mathcal{P}_{\mathcal{C}_2}\left(\mathbf{x}^{t+1}\right)\right)$$

Using Alternating Projections Method, we have:

$$\mathbf{x}^{t+1}=\mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^{t}\right)$$
$$\mathbf{x}^{t+2}=\mathcal{P}_{\mathcal{C}_2}\left(\mathbf{x}^{t+1}\right)$$

To show equivalence, choose $\alpha_t = 1$ for all $t$. Then the subgradient method updates become:

$$\mathbf{x}^{t+1} = \mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^t\right)$$
$$\mathbf{x}^{t+2} = \mathcal{P}_{\mathcal{C}_2}\left(\mathbf{x}^{t+1}\right)$$

which are exactly the steps of the alternating projections method.

➢

In order to prove the equality

$$\nabla\left(\frac{1}{2}\text{dist}^2_{\mathcal{C}_i}\left(\mathbf{x}^t\right)\right) = \mathbf{x}^t - \mathcal{P}_{\mathcal{C}_1}\left(\mathbf{x}^t\right)$$

Let

$$f\left(\mathbf{x}\right) = \frac{1}{2}\text{dist}^2_{\mathcal{C}_2}\left(\mathbf{x}^t\right)$$

Since

$$\text{dist}_{\mathcal{C}}\left(\mathbf{x}\right) = \min_{\mathbf{z}\in\mathcal{C}} \left\|\mathbf{x}-\mathbf{z}\right\|_2$$

So the function $f(\mathbf{x})$ is actually half the squared Euclidean distance from $\mathbf{x}$ to the nearest point in $\mathcal{C}$.

For the convex set $\mathcal{C}$ and $\mathbf{x}\notin\mathcal{C}$, the projection $\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)$ is the unique point in $\mathcal{C}$ such that for all $\mathbf{y}\in\mathcal{C}$:

$$\left\|\mathbf{x}-\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)\right\|_2 \le \left\|\mathbf{x}-\mathbf{y}\right\|_2$$

The gradient $\nabla f(\mathbf{x})$ is the direction of steepest ascent of $f$ at $\mathbf{x}$. For $f(\mathbf{x})$, the direction of steepest ascent should point towards the line connecting $\mathbf{x}$ and $\mathcal{P}_{\mathcal{C}}\left(\mathbf{x}\right)$, because this is the most direct way for $\mathbf{x}$ to increase its distance from $\mathcal{C}$.

More formally, since $f$ is differentiable at $\mathbf{x}$, we can directly compute the gradient of $f$

at $\mathbf{x}$:

$$\nabla f(\mathbf{x}) = \nabla\left(\frac{1}{2}\left\|\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right\|_2^2\right) = \left(\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right) \cdot \nabla\left(\left\|\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right\|_2\right)$$

Since $\left\|\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right\|_2$ is the distance from $\mathbf{x}$ to $\mathcal{P}_{\mathcal{C}}(\mathbf{x})$, its gradient is the unit vector $\frac{\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})}{\left\|\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right\|_2}$.

Thus, we get

$$\nabla f(\mathbf{x}) = \left(\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right) \cdot \frac{\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})}{\left\|\mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})\right\|_2} = \mathbf{x} - \mathcal{P}_{\mathcal{C}}(\mathbf{x})$$

Thus, we have proved that $\nabla\left(\frac{1}{2}\mathrm{dist}_{\mathcal{C}_i}^2\left(\mathbf{x}^t\right)\right) = \mathbf{x}^t - \mathcal{P}_{\mathcal{C}_i}\left(\mathbf{x}^t\right)$.

Aalto University
School of Electrical
Engineering

**Problem 3**

Generalized Pythagorean Theorem states that

$$D_\varphi(\mathbf{z},\mathbf{x}) \geq D_\varphi(\mathbf{z},\mathbf{x}_{\mathcal{C},\varphi}) + D_\varphi(\mathbf{x}_{\mathcal{C},\varphi},\mathbf{x})$$

where $D_\varphi(\mathbf{z}, \mathbf{x})$ is Bregman divergence between points $\mathbf{z}$ and $\mathbf{x}$ for generating function $\varphi$ and $\mathbf{x}_{\mathcal{C},\varphi} = \mathcal{P}_{\mathcal{C},\varphi}(\mathbf{x}) = \arg\min_{\mathbf{z}\in\mathcal{C}} D_\varphi(\mathbf{z}, \mathbf{x})$ is Bregman projection of point $\mathbf{x}$ to set $\mathcal{C}$.

Prove that if $\mathcal{C}$ is affine plane, then Generalized Pythagorean Theorem holds as equality. Prove also that Generalized Pythagorean Theorem is equivalent to Pythagorean Theorem if in addition $D_\varphi(\mathbf{z},\mathbf{x}) = \mathrm{dist}_{\mathcal{C}}^2(\mathbf{x})$.

- Prove that if $\mathcal{C}$ is affine plane, then Generalized Pythagorean Theorem holds as equality.

An affine set $\mathcal{C}$ means that for any points $\mathbf{y}_1, \mathbf{y}_2 \in \mathcal{C}$ and any scalar $\alpha$, the combination $\alpha\mathbf{y}_1 + (1-\alpha)\mathbf{y}_2$ is also in $\mathcal{C}$. For Bregman divergences, the property that characterizes an affine set is the linearity of the generating function $\varphi$ along the set. In an affine space, the divergence $D_\varphi$ simplifies to a difference of function values and their first-order approximations.

Now, since $\mathbf{x}_{\mathcal{C},\varphi}$ is the Bregman projection of $\mathbf{x}$ onto $\mathcal{C}$, by definition of the projection with respect to the Bregman divergence, we have:

$$D_\varphi(\mathbf{z},\mathbf{x}) = \varphi(\mathbf{z}) - \varphi(\mathbf{x}) - \langle \nabla\varphi(\mathbf{x}), \mathbf{z}-\mathbf{x}\rangle$$

Similarly,

$$D_\varphi(\mathbf{z},\mathbf{x}_{\mathcal{C},\varphi}) = \varphi(\mathbf{z}) - \varphi(\mathbf{x}_{\mathcal{C},\varphi}) - \langle \nabla\varphi(\mathbf{x}_{\mathcal{C},\varphi}), \mathbf{z}-\mathbf{x}_{\mathcal{C},\varphi}\rangle$$
$$D_\varphi(\mathbf{x}_{\mathcal{C},\varphi},\mathbf{x}) = \varphi(\mathbf{x}_{\mathcal{C},\varphi}) - \varphi(\mathbf{x}) - \langle \nabla\varphi(\mathbf{x}), \mathbf{x}_{\mathcal{C},\varphi}-\mathbf{x}\rangle$$

Because $\mathcal{C}$ is affine, $\nabla\varphi(\mathbf{x}) = \nabla\varphi(\mathbf{x}_{\mathcal{C},\varphi})$. Adding the last two equalities, we get:

$$D_\varphi\left(\mathbf{z},\mathbf{x}_{\mathcal{C},\varphi}\right)+D_\varphi\left(\mathbf{x}_{\mathcal{C},\varphi},\mathbf{x}\right)=\varphi(\mathbf{z})-\varphi(\mathbf{x})-\left\langle\nabla\varphi(\mathbf{x}),\mathbf{z}-\mathbf{x}\right\rangle$$

Which is exactly $D_\varphi\left(\mathbf{z},\mathbf{x}\right)$, thus proving the equality.

- Proof that Generalized Pythagorean Theorem is equivalent to the Pythagorean Theorem if $D_\varphi\left(\mathbf{z},\mathbf{x}\right)=\mathrm{dist}_{\mathcal{C}}^2\left(\mathbf{x}\right)$

The classic Pythagorean Theorem states that in a right-angled triangle, the area of the square on the hypotenuse (the side opposite the right angle) is equal to the sum of the areas of the squares on the other two sides. If we consider $D_\varphi\left(\mathbf{z},\mathbf{x}\right)=\mathrm{dist}_{\mathcal{C}}^2\left(\mathbf{x}\right)$, we can interpret the Bregman divergence as a measure of squared distance in a possibly non-Euclidean space defined by $\varphi$.

To show equivalence, we can take advantage of the fact that $D_\varphi\left(\mathbf{z},\mathbf{x}\right)=\mathrm{dist}_{\mathcal{C}}^2\left(\mathbf{x}\right)$ implies a Euclidean distance in the space transformed by $\varphi$. That is, the Bregman divergence behaves like a squared Euclidean distance.

If we denote $d$ ($\mathbf{x}$,$\mathbf{y}$) as the Euclidean distance between $\mathbf{x}$ and $\mathbf{y}$, then the Generalized Pythagorean Theorem can be written as:

$$d^2\left(\mathbf{z},\mathbf{x}\right)\geq d^2\left(\mathbf{z},\mathbf{x}_{\mathcal{C},\varphi}\right)+d^2\left(\mathbf{x}_{\mathcal{C},\varphi},\mathbf{x}\right)$$

Since $\mathbf{x}_{\mathcal{C},\varphi}$ is the closest point in $\mathcal{C}$ to $\mathbf{x}$, the triangle formed by $\mathbf{z}$, $\mathbf{x}$, and $\mathbf{x}_{\mathcal{C},\varphi}$ is a right triangle with $\mathbf{x}_{\mathcal{C},\varphi}$ at the right angle. Therefore, the Generalized Pythagorean Theorem directly reduces to the classical Pythagorean Theorem for this right triangle.