# Alternate Directive method of Multipliers and its Applications

*Arrun Sivasubramanian(CB.EN.U4AIE19103)*
*B. Tech CSE AI*
*Amrita School of Engineering, Ettimadai, Coimbatore*

## Abstract:

ADMM is one of the best methods known at present for solving optimisation problems involving equality constraints. It can handle general constraints and is an effective method to solve problems arising in machine learning including lasso minimization, SVM and some others. It's very effective in statistics and machine learning problems dealing with very large data because it is well suited to distributed(convex) optimisation.

In this project, I have described about ADMM in detail on how it's formulated and its assorted applications in practical areas depending on the type of problem. In the latter parts, I have elaborated some basic concepts needed for speeding up the algorithm by approximately 80 times of the original approach's speed.

## Introduction to ADMM:

ADMM combines the decomposability of dual ascent with the superior convergence properties of the method of multipliers. The basic idea is to split the function F into two independent parts as F(x, z)=f(x)+g(z) and consider the new minimisation problem subject to Ax + Bz = c.

Instead of jointly performing minimization, ADMM first performs an x-minimization step and then a z-minimization step which accounts for it name alternative direction. Because the Lagrangian is augmented, some mild conditions on A and B guarantee termination. Convergence of ADMM is under the following assumptions:

1. The functions f(x) and g(z) are proper and closed functions and domain of f(x) and g(x) is not equal to null.

2. The square matrices $A^TA$ and $B^TB$ of size m and n respectively are invertible. Equivalently, the matrices A and B of dimension (p x m) and (p x n) have ranks m and n respectively.
3. The un-augmented Lagrangian term:

$$L_0 = f(x) + g(z) + \lambda^T (Ax + Bz - c) \text{ has a saddle point}$$

=> There exists x*, z*, $\lambda$*(not necessarily unique) such that:

$$L_0(x^*, z^*, \lambda) \leq L_0(x^*, z^*, \lambda^*) \leq L_0(x, z, \lambda^*) \ \forall \ x, z, \lambda$$

In this project, I experimented with variants of ADMM applied to :

1. Basic Pursuit
2. Least Absolute Deviations (LAD)
3. Linear and Quadratic Programming
4. Sparse Covariance Matrices
5. Hubber Loss
6. Regressor Selector
7. Total Variable Minimisation
8. LASSO

## 1. Basic Pursuit:

Basis pursuit is the equality-constrained L1 minimization problem. It is often used as a heuristic for finding a sparse solution to an underdetermined system of linear equations. It plays a central role in modern statistical signal processing, particularly the theory of compressed sensing.

Problem formulation is as follows:

minimize $\|x\|_1$ subject to $Ax = b$

*ADMM* formulation:

minimize $f(x) + g(z)$ subject to $x - z = \mathbf{0}$ vector

*where,*

$f(x) = \|x\|_1$

$$g(z) = \begin{cases} 0, & Az = b \\ \infty & \text{otherwise} \end{cases}$$

Formation of Lagrangian and solving the problem is as follows:

$$L(x,z,u)=\|x\|_1 + g(z) + \frac{1}{2\lambda}\|x - z + u\|_2^2$$

$$x^{k+1} = \arg\min_x \|x\|_1 + \frac{1}{2\lambda}\|x - z^k + u^k\|_2^2$$

$$= S_\lambda\left(z^k - u^k\right)$$

$$z^{k+1} = \arg\min_x g(z) + \frac{1}{2\lambda}\|x^{k+1} - z + u^k\|_2^2$$

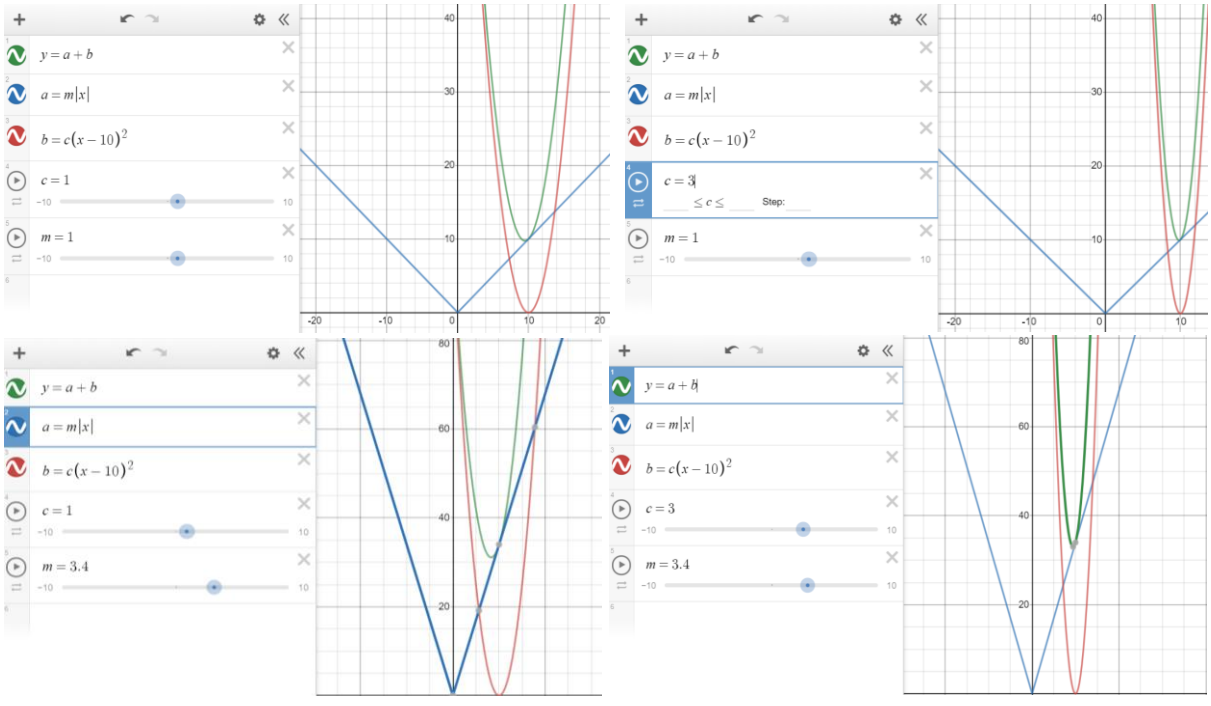
$$z^{k+1} = P_{Az=b}\left(x^{k+1} + u^k\right) = \left(x^{k+1} + u^k\right) + pinv(A) \times \left(b - A\left(x^{k+1} + u^k\right)\right)$$

$$u^{k+1} = u^k + \left(x^{k+1} - z^{k+1}\right)$$

**Function to be aware:** The latter part of the added term in x-update is called the penalty or the regularisation function. The function S(y) is called as the shrinkage function as it shrinks y towards zero. The exact form of shrinkage function depends on the penalty function.

The shrinkage function sets all values of y less than linear multiplier of x(***λ*** in our case) to zero. Thus geometrically, we can define these functions as thresholding functions. But the penalty term can have an additional parameter(***α*** in our case) which geometrically decides the shape(concavity of the function).

Graphical representation of change in shape and intercept of new curve is as follows:



Figures that help in visualising change in function with change in initial parameters. Here c denotes alpha i.e. the coefficient of the penalty function. With variation in c, concavity of function changes and with variation in M(the pulling parameter), the intercepts of the function change.

**Coding basic pursuit is as follows:**

```
clc;clear all;close all;
n = 30;
m = 10;
A = randn(m,n);

x = sprandn(n, 1, 0.1*n);
b = A*x;

xtrue = x;
[X,Z,U] = basis_pursuit(A, b, 1.0, 1.0);

function [x,z,u] = basis_pursuit(A, b, rho, alpha)
% basis_pursuit  Solve basis pursuit via ADMM
%
% Solves the following problem via ADMM:
%   minimize     ||x||_1
%   subject to   Ax = b
%
% The solution is returned in the vector x.
% rho is the augmented Lagrangian parameter and alpha is the
over-relaxation parameter
    MAX_ITER = 1000;
    [m n] = size(A);
    x = zeros(n,1);
    z = zeros(n,1);
    u = zeros(n,1);

    % precompute static variables for x-update (projection on
to Ax=b)
    AAt = A*A';
    P = eye(n) - A' * (AAt \ A);
    q = A' * (AAt \ b);

    for k = 1:MAX_ITER
        % x-update
        x = P*(z - u) + q;

        % z-update with relaxation
        zold = z;
        x_hat = alpha*x + (1 - alpha)*zold;
        z = shrinkage(x_hat + u, 1/rho);
```

```
        u = u + (x_hat - z);
    end
end

function y = shrinkage(a, kappa)
    y = max(0, a-kappa) - max(0, -a-kappa);
end
```

## 2. The Least Absolute Deviations Problem:

Least Absolute Deviations (LAD) is an alternative to traditional Least Squares by using cost function. Least absolute deviations provides a more robust fit than least squares when the data contains large outliers, and has been used extensively in statistics and econometrics. It is similar to an OLS regression model, but it minimizes the absolute deviation instead of the sum of squares. LAD is equivalent to the median regression, a special case of the quantile regression models. LAD is a robust regression technique in the sense that the estimated coefficients are insensitive to outliers.

This problem differs slightly from Basis Pursuit in that we do not have the constraint Ax = b anymore. We are given observation in the rows of A and corresponding noisy measurements b. Our goal here is to actually recover the original, minimal signal x despite the noise in b. For this, we don't need to denoise it.

The problem statement is:

$$x^* = \underset{x}{\operatorname{argmin}} \|Ax-b\|_1 \quad , \qquad \left( \|Ax-b\|_1 \equiv \sum_{i=1}^{n} \left| a_i^T x - b_i \right|, \quad a_i^T \text{ is ith row of A} \right)$$

Formulation and solving:

$$\min \ \|z\|_1$$
$$subj\ Ax - b - z = 0$$

$$L(x,y,z) = \|z\|_1 + \frac{\beta}{2}\|Ax - b - z + u\|^2$$
$$\Delta L_x(x,y,z) = -A^T\left(Ax - b - z + u\right) = 0$$
$$\Rightarrow x_{k+1} = \left(A^T A\right)^{-1} A^T (b + z^k - u^k)$$

Instead of naively solving this system, we can use a more efficient approach by finding the Cholesky Decomposition using a upper and lower triangular matrix.

$$\Delta L_z(x,y,z) = 1 - \beta\left(Ax - b - z + u\right) = 0$$

$$\hat{Ax} = \alpha Ax_{k+1} + (1-\alpha)(z_k + \beta)$$

$$\Rightarrow z_{k+1} = S_\lambda\left(\hat{Ax} - b + u^k\right)$$

$$\text{where } S_\lambda(c) = \begin{cases} c - \lambda(sign(c)) & \lambda \le c \\ 0 & \lambda > c \end{cases}$$

Updating z with relaxation, here S is elementwise shrinkage of lambda towards zero.

$$\Delta L_u(x,y,z) = Ax^{k+1} - b - z^{k+1} = 0$$

$$u_{k+1} = u_k + Ax^{k+1} - b - z^{k+1}$$

**MATLAB code to execute this is as follows:**

```
function [x,z,u] = lad(A, b, rho, alpha)

% lad  Least absolute deviations fitting via ADMM
% Solves the following problem via ADMM:
%
%   minimize     ||Ax - b||_1
%
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter

    MAX_ITER = 1000;
    [m n] = size(A);
    x = zeros(n,1);
    z = zeros(m,1);
    u = zeros(m,1);

    for k = 1:MAX_ITER
        if k > 1
            x = R \ (R' \ (A'*(b + z - u)));
        else
            R = chol(A'*A);
            x = R \ (R' \ (A'*(b + z - u)));
        end

        zold = z;
        Ax_hat = alpha*A*x + (1-alpha)*(zold + b);
        z = shrinkage(Ax_hat - b + u, 1/rho);
```

```
        u = u + (Ax_hat - z - b);
    end
end

function y = shrinkage(a, kappa)
    y = max(0, a-kappa) - max(0, -a-kappa);
end
```

## 3. Basic ADMM for Linear and Quadratic Programming:

Quadratic Programming is the process of solving a specific type of mathematical problem(specifically linear constrained) i.e. optimizing a quadratic function od several variables subject to linear constraints on these variables. A quadratic problem having n variables and m constraints is formulated as follows:

$n \times 1$ vector q

n $\times$ n symmetric matrix P

$m \times$ n real matrix A

$m \times 1$ vector b

For linear programming(a special case of quadratic programming with P=0, We reformulate the primal LPP problem to solve using ADMM as follows:

$$\min c^T x_1$$

$$\text{subj to: } Ax_1 = b \ ;$$

$$x_1 - x_2 = 0;\ x_2 \geq 0$$

The augmented Lagrangian Function is as follows

$$L_P(x_1, x_2, y, s) = c^T x_1 - y^T (Ax_1 - b) - s^T (x_1 - x_2) + \frac{\beta}{2}(\|Ax_1 - b\|^2 + \|x_1 - x_2\|^2)$$

Using the first order gradient condition, we get:

$$\nabla L_{P,x_1}(x_1, x_2, y, s) = c^T - A^T y - s + \beta(A^T (Ax_1 - b) + (x_1 - x_2)) = 0$$

This can also be written as:

$$x_1^{k+1} = \arg\min_{x:Ax=b} f(x_1) + g(x_2) + \frac{1}{2\lambda}\left\|x_1^k - x_2^k + u^k\right\|^2;$$

$$i.e.\ \text{prox}_f\left(x_2^k - u^k\right)$$

$$\left(\text{here } u^k = \frac{1}{\rho}y^k; \rho = \frac{1}{\lambda}\right)$$

We obtain first update step for $x_1$ as:

$$x_1^{k+1} = (A^TA + I)^{-1} + \frac{1}{\beta}\left(A^Ty^k + s^k - c + A^Tb^k + x_2^k\right)$$

Similarly, we update $x_2$ as:

$$x_2^{k+1} = \arg\min_{x:Ax=b}\left(f(x_1) + g(x_2) + \frac{1}{2\lambda}\left\|x_1^{k+1} - x_2^k + u^k\right\|^2\right);$$

$$i.e.\ \text{prox}_g\left(x_1^k - u^k\right)$$

This expression is same as using shrinkage function used in SVM Updating y and s using steepest ascent, we get:

$$y^{k+1} = y^k + \beta\left(Ax_1 - b\right)$$

$$s^{k+1} = s^k - \beta\left(x_1^{k+1} - x_2^{k+1}\right)$$

We terminate the iterative when solution meets primal feasible i.e.

$$Ax_1^{k+1} - b \approx 0 \text{ and } x_1^{k+1} - x_2^{k+1} \approx 0$$

For Quadratic Programming, we formulate:

$$\min\left(\frac{1}{2}\right)x^TPx + q^Tx$$

$$subj: Ax = b; x \geq 0$$

$$\text{here } f(x) = \min\left(\frac{1}{2}\right)x^TPx + q^Tx;$$

$g(z)$ is Indicator Function for $z > 0$

$$L_\rho(x,z,u) = f(x) + g(z) + \left(\frac{1}{2\lambda}\right)\left\|x - z + u\right\|_2^2 + v^T\left(Ax - b\right)$$

$$\frac{\partial L}{\partial x} = Px^{k+1} + q + \frac{1}{\lambda}\left(\left\|x^{k+1} - z^k + u^k\right\|\right) + A^Tv$$

$$\left(P+\frac{1}{\lambda}I\right)x^{k+1}+q+A^{T}v-\frac{1}{\lambda}\left(z^{k}-u^{k}\right)=0$$

$$\frac{\partial L}{\partial v}=0\Rightarrow Ax^{k+1}-b=0$$

In the form of a matrix, it can be written and solved as follows:

$$\begin{bmatrix} P+\frac{1}{\lambda}I & A^{T} \\ A & 0 \end{bmatrix}\begin{bmatrix} x^{k+1} \\ v \end{bmatrix}+\begin{bmatrix} q-\frac{1}{\lambda}(z^{k}-u^{k}) \\ -b \end{bmatrix}=\begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

$$\begin{bmatrix} x^{k+1} \\ v \end{bmatrix}=\begin{bmatrix} P+\frac{1}{\lambda}I & A^{T} \\ A & 0 \end{bmatrix}^{-1}\begin{bmatrix} \frac{1}{\lambda}(z^{k}-u^{k})-q \\ b \end{bmatrix}$$

From:

$$x_{2}{}^{k+1}=\arg\min_{x:Ax=b}\left(f\left(x_{1}\right)+g(x_{2})+\frac{1}{2\lambda}\left\|x^{k+1}-z^{k}+u^{k}\right\|^{2}\right)$$

$$u^{k+1}=\arg\min_{x:Ax=b}f\left(x_{1}\right)+g(x_{2})+\frac{1}{2\lambda}\left\|x^{k+1}-z^{k+1}+u^{k}\right\|^{2}$$

We finally arrive at:

$$z^{k+1}=\left(x^{k+1}+u^{k}\right)_{+}$$

$$u^{k+1}=u^{k}+x^{k+1}-z^{k+1}$$

Another key point to note is that both Linear and Quadratic programming can be solved using the same Quadratic programming problem where P=0 as mentioned earlier.

### Code to perform both LP and QP is as follows:

```
clc;clear all;close all;

n = 500;  % dimension of x
m = 400;  % number of equality constraints

c  = rand(n,1) + 0.5;     % create nonnegative price vector with mean 1
x0 = abs(randn(n,1));     % create random solution vector

A = abs(randn(m,n));      % create random, nonnegative matrix A
b = A*x0;
[X,Z,U] = linprog(c, A, b, 1.0, 1.0);%function call
```

```
function [x,z,u] = linprog(c, A, b, rho, alpha)
% Linear Programming Solves the following problem via ADMM:
%
%   minimize     c'*x
%   subject to   Ax = b, x >= 0
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter

    MAX_ITER = 1000;

    [m n] = size(A);
    x = zeros(n,1);
    z = zeros(n,1);
    u = zeros(n,1);

    for k = 1:MAX_ITER

        % x-update
        tmp = [ rho*eye(n), A'; A, zeros(m) ] \ [ rho*(z - u) - c; b ];
        x = tmp(1:n);%extract 1st 500 values of temp

        % z-update with relaxation
        zold = z;
        x_hat = alpha*x + (1 - alpha)*zold;
        z = pos(x_hat + u);

        u = u + (x_hat - z);
    end
end
```

## 4. Interior point ADMM for sparce covariance:

Given a dataset consisting of samples from a zero mean Gaussian distribution in $R^n$ , consider the task of estimating the covariance matrix $\Sigma$ under the prior assumption that the inverse of it is sparse. Since $\Sigma^{-1}_{ij}$ is zero if and only if the $i^{th}$ and $j^{th}$ components of the random variable are conditionally independent, this problem is equivalent to the *structure learning* problem of estimating the topology of the undirected graphical model representation of the Gaussian. Determining the sparsity pattern of the inverse covariance matrix $\Sigma^{-1}$ is also called the *covariance selection problem*.

For a very small value of n, it is feasible to search over all sparsity patterns in $\Sigma^{-1}$ since for a fixed sparsity pattern, determining the maximum likelihood estimate of $\Sigma$ is a tractable (convex optimization) problem. A good experiment that scales to much larger values of *n* is to minimize the negative log-likelihood (with respect to $\Sigma^{-1}$) with an l1 regularization term to promote sparsity of the estimated inverse covariance matrix .

If c is the empirical covariance matrix $\frac{1}{N}\sum_{i=1}^{N} a_i a_i^T$,

The estimation problem can be written as:

$\min(c^T x) - \log(\det(x)) + \lambda |x|_1$.

This is a special case of the $l_1$ regularised problem with the initial part $(c^T x) - \log(\det(x))$ as the convex loss function.

Hence we reformulate primal LPP using log barrier function as:

$$\min_{x_1, x_2} \; c^T x_1 - \mu \sum_j \ln(x_{2,j})$$

$$subj.\;\; Ax_1 = b;\;\; x_1 - x_2 = 0\{\text{Implicit}\}$$

Thus, Lagrangian function with augmented term and equations on partial derivative can be written as:

$$L_{P,\mu}(x_1, x_2, y, s) = c^T x_1 - \mu \sum_j \ln(x_{2,j}) - y^T (Ax_1 - b) - s^T (x_1 - x_2) + \frac{\beta}{2}\left( \|Ax_1 - b\|^2 + \|x_1 - x_2\|^2 \right)$$

$$\Delta L_{P,\mu,x_1}(x_1, x_2, y, s) = c^T - A^T y - s + \beta(A^T(Ax_1 - b) + (x_1 - x_2)) = 0$$

$$\Delta L_{P,\mu,x_2}(x_1, x_2, y, s) = s - \frac{\mu}{x_2} + \beta(x_2 - x_1)$$

Update becomes:

$$x_1^{k+1} = (A^T A + I)^{-1} + \frac{1}{\beta}\left( A^T y^k + s^k - c + A^T b^k + x_2^k \right)$$

$$x_2 = \frac{1}{2\beta}\left( \beta x_1 - s_j + \sqrt{\beta^2 x_{2\,i,j}^2 - 2\beta s_j x_{1,j} + s_j^2 + 4\beta u} \right) \; for \; \text{j=1,...,n}$$

The final variable update is same as the primal LPP:

$$y^{k+1} = y^k + \beta\left( Ax_1 - b \right)$$

$$s^{k+1} = s^k - \beta\left( x_1^{k+1} - x_2^{k+1} \right)$$

Here $\boldsymbol{\beta}$ is the step size

At the end of iteration we check for $\mu_{k+1} = \omega \mu_k$; w<<1

## MATLAB Snippet is as follows:

```
clc;clear all;close all;
n = 100;   % number of features
N = 10*n;  % number of samples

% generate a sparse positive definite inverse covariance matrix
Sinv      = diag(abs(ones(n,1)));
idx       = randsample(n^2, 0.001*n^2);
Sinv(idx) = ones(numel(idx), 1);
Sinv = Sinv + Sinv';   % make symmetric
if min(eig(Sinv)) < 0  % make positive definite
    Sinv = Sinv + 1.1*abs(min(eig(Sinv)))*eye(n);
end
S = inv(Sinv);

% generate Gaussian samples
D = mvnrnd(zeros(1,n), S, N);
[X] = covsel(D, 0.01, 1, 1);

function [Z] = covsel(D, lambda, rho, alpha)
% covsel  Sparse inverse covariance selection via ADMM
% Solves the following problem via ADMM:
%
%   minimize  trace(S*X) - log det X + lambda*||X||_1
%
% with variable X, where S is the empirical covariance of the data
% matrix D (training observations by features).
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter

    MAX_ITER = 1000;

    S = cov(D);
    n = size(S,1);

    X = zeros(n);
    Z = zeros(n);
    U = zeros(n);

    for k = 1:MAX_ITER
        % x-update
        [Q,L] = eig(rho*(Z - U) - S);
        es = diag(L);
        xi = (es + sqrt(es.^2 + 4*rho))./(2*rho);
        X = Q*diag(xi)*Q';

        % z-update with relaxation
        Zold = Z;
        X_hat = alpha*X + (1 - alpha)*Zold;
        Z = shrinkage(X_hat + U, lambda/rho);

        U = U + (X_hat - Z);
    end
end

function y = shrinkage(a, kappa)
    y = max(0, a-kappa) - max(0, -a-kappa);
end
```

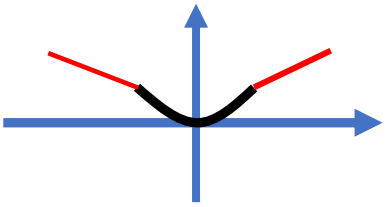
## 5. ADMM for Huber loss:

Huber loss is a loss function that lies between least square error and LAD. In brief, this function is such that it considers absolute error like LAD if absolute value lies after a particular threshold based on the problem(usually 1), and squared error like LSQR for absolute values below the same threshold.

Thus:

$$huber(z_i) = \frac{1}{2} z_i^2, \quad |z_i| \le 1$$

$$huber(z_i) = |z_i| - \frac{1}{2}, \quad |z_i| > 1$$



Huber Loss function plot

The Optimisation problem now is:

$$\min_{x,z} \; g(z) \; = \; \sum_{i=1}^{m} huber(z_i)$$

$$subject \text{ to } z = Ax - b$$

$$ADMM \text{ formulation is}$$

$$\min_{x,z} g(z)$$

$$subject \text{ to } z = Ax - b$$

$$\Rightarrow Ax - b - z = \mathbf{0} \text{ vector}$$

$$Augmented \text{ Lagrangian is}$$

$$L(x,z,u) = g(z) + \frac{1}{2\lambda} \| Ax - b - z + u \|_2^2$$

*Update* x:

$$\text{L(x,z,u)}=g(z)+\frac{1}{2\lambda}\left\|Ax-b-z+u\right\|_2^2$$

$$x^{k+1}=\arg\min_x\left\|Ax-b-z^k+u^k\right\|_2^2$$

*Solution*:

$$Let\ f(x)=\left\|Ax-b-z^k+u^k\right\|_2^2$$

$$=\left(Ax-b-z^k+u^k\right)^T\left(Ax-b-z^k+u^k\right)$$

$$\nabla f(x)=\mathbf{0}\Rightarrow A^T\left(Ax-b-z^k+u^k\right)=\mathbf{0}$$

$$A^TAx=A^T\left(b+z^k-u^k\right)$$

$$x^{k+1}=\left(A^TA\right)^{-1}A^T\left(b+z^k-u^k\right)$$

*Update* z:

$$\text{L(x,z,u)}=g(z)+\frac{1}{2\lambda}\left\|Ax-b-z+u\right\|_2^2$$

$$z^{k+1}=\arg\min_z g(z)+\frac{1}{2\lambda}\left\|Ax^{k+1}-b-z+u^k\right\|_2^2$$

*Solution*: g(z)=Huber(z) consists of two piecewise continous functions .

Let us consider first one.

$$Let\ g(z)=\frac{z^2}{2},\ \text{then}$$

$$z^{k+1}=\arg\min_z\frac{z^2}{2}+\frac{1}{2\lambda}\left\|Ax^{k+1}-b-z+u^k\right\|_2^2$$

$$z-\frac{1}{\lambda}\left(Ax^{k+1}-b-z+u^k\right)=0\Rightarrow(1+\lambda)z^{k+1}=\left(Ax^{k+1}-b+u^k\right)$$

$$z_i^{k+1}=\frac{1}{1+\lambda}\left(Ax^{k+1}-b+u^k\right)_i\text{ ----------(1)}$$

$\left(Ax^{k+1}-b+u^k\right)_i$ is the ith element of vector $\left(Ax^{k+1}-b+u^k\right)$

*Update* z: Consider second function

*Let* $g(z)=|z|-\frac{1}{2}$, then

$$z^{k+1}=\arg\min_z |z|-\frac{1}{2}+\frac{1}{2\lambda}\left\|Ax^{k+1}-b-z+u^k\right\|_2^2$$

$$z^{k+1}=\arg\min_z |z|+\frac{1}{2\lambda}\left\|Ax^{k+1}-b-z+u^k\right\|_2^2$$

$$z_i^{k+1}=S_\lambda\left(\left(Ax^{k+1}-b+u^k\right)_i\right)=\left(Ax^{k+1}-b+u^k\right)_i\pm\lambda \text{ ---------(2)}$$

$S_\lambda$ represent shrinkage by $\pm\lambda$, sign depending on the sign of $\left(Ax^{k+1}-b+u^k\right)_i$

$\left(Ax^{k+1}-b+u^k\right)_i$ is the ith element of vector $\left(Ax^{k+1}-b+u^k\right)$

Then we combine the two formula (1) and (2) for z update.

*Split* $z_i^{k+1}$*in* (2) int*o* two terms, one term containing (1) and express the residual in terms of a Shrinkage function

$$z_i^{k+1}=\left(Ax^{k+1}-b+u^k\right)_i\pm\lambda----(2)$$

$$z_i^{k+1}=\left(Ax^{k+1}-b+u^k\right)_i\pm\lambda=\frac{1}{1+\lambda}\left(Ax^{k+1}-b+u^k\right)_i+\frac{\lambda}{1+\lambda}\left\{\left(Ax^{k+1}-b+u^k\right)_i\pm(1+\lambda)\right\}$$

$$=\frac{1}{1+\lambda}\left(Ax^{k+1}-b+u^k\right)_i+\frac{\lambda}{1+\lambda}S_{1+\lambda}\left(Ax^{k+1}-b+u^k\right)_i$$

*or*

$$z^{k+1}=\frac{1}{1+\lambda}\left(Ax^{k+1}-b+u^k\right)+\frac{\lambda}{1+\lambda}S_{1+\lambda}\left(Ax^{k+1}-b+u^k\right)$$

## 6. ADMM for non-convex Regression Selector:

As an example, consider the least squares regressor selection or feature selection problem,

$$x^*=\underset{x}{\operatorname{argmin}}\ \frac{1}{2}\|Ax-b\|_2^2$$

$$\textit{subject to } \text{card(x)} \le \text{k}$$

which is to find the best fit to b as a linear combination of no more than c columns of A. For this problem, ADMM takes the form above, where the x-update involves a regularized least squares problem, and the z-update involves keeping the c largest magnitude elements of $x_{k+1} + u_k$.

This is just like ADMM for the lasso, except that soft thresholding is replaced with 'hard' thresholding. This close connection is hardly surprising, since lasso can be thought of as a heuristic for solving the regressor selection problem. From this viewpoint, the lasso controls the trade-off between least squares error and sparsity through the parameter λ, whereas in ADMM for regressor selection, the same trade-off is controlled by the parameter c, the exact cardinality desired.

**Problem formulation is as follows:**

$$\min_{x,z} f(x) + g(z) = \frac{1}{2}\|Ax - b\|_2^2 + g(z)$$

$$g(z) = \begin{cases} 0 \text{ if } \mathrm{card}(z) \le k \\ \infty, \text{ if } \mathrm{card}(z) > k \end{cases}$$

*ADMM* formulation :

*Augmented* Lagrangian is

$$L(x,z,u) = f(x) + g(z) + \frac{1}{2\lambda}\|x - z + u\|_2^2$$

$$= \frac{1}{2}\|Ax - b\|_2^2 + g(z) + \frac{1}{2\lambda}\|x - z + u\|_2^2$$

x-update is as follows:

$$x^{k+1} = \arg\min_x \frac{1}{2}\|Ax - b\|_2^2 + \frac{1}{2\lambda}\|x - z^k + u^k\|_2^2$$

*Solution*: Let $\rho = \frac{1}{\lambda}$

$$A^T(Ax - b) + \rho(x - z^k + u^k) = 0 \text{ vector}$$

$$(A^T A + \rho I)x = A^T b + \rho(z^k - u^k)$$

$$x^{k+1} = (A^T A + \rho I)^{-1}(A^T b + \rho(z^k - u^k))$$

z-update is as follows:

$$L(z) = g(z) + \frac{\rho}{2}\|x^{k+1} - z + u^k\|_2^2$$

From the latter term, we get updated z and project it to get:

$z^{k+1} = \max(0, x^{k+1} + u^k)$

{in accordance with g(z)->Indicator Func$^n$}

*The* original lagrangian multiplier term is $u^T(x - z)$

$\therefore$ The gradient w.r.t.u is $x - z$

$\Rightarrow u^{k+1} = u^k + \left(x^{k+1} - z^{k+1}\right)$

## MATLAB Code to execute is as follows:

```
clc;clear all;close all;
randn('seed', 0);
rand('seed',0);

m = 1500;       % number of examples
n = 5000;       % number of features
p = 100/n;      % sparsity density

% generate sparse solution vector
x = sprandn(n,1,p);

% generate random data matrix
A = randn(m,n);

% normalize columns of A
A = A*spdiags(1./sqrt(sum(A.^2))', 0, n, n);

% generate measurement b with noise
b = A*x + sqrt(0.001)*randn(m,1);

xtrue = x;   % save solution
x  = regressor_sel(A, b, p*n, 1.0);
function z = regressor_sel(A, b, K, rho)
% regressor_sel  Solve lasso problem via ADMM
%
% Attempts to solve the following problem via ADMM:
%
%   minimize || Ax - b ||_2^2
%   subject to card(x) <= K
%
% where card() is the number of nonzero entries.
%
% The solution is returned in the vector x.
%
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter
    MAX_ITER = 1000;
    [m, n] = size(A);

    % save a matrix-vector multiply
    Atb = A'*b;
    x = zeros(n,1);
    z = zeros(n,1);
```

```
    u = zeros(n,1);

    % cache the factorization
    [L U] = factor(A, rho);
    for k = 1:MAX_ITER

        % x-update
        q = Atb + rho*(z - u);    % temporary value
        if( m >= n )    % if skinny
           x = U \ (L \ q);
        else            % if fat
           x = q/rho - (A'*(U \ ( L \ (A*q) )))/rho^2;
        end

        % z-update with relaxation
        zold = z;
        z = keep_largest(x + u, K);

        % u-update
        u = u + (x - z);
    end
end

function z = keep_largest(z, K)
    [val,pos] = sort(abs(z), 'descend');
    z(pos(K+1:end)) = 0;
end

function [L U] = factor(A, rho)
    [m, n] = size(A);
    if ( m >= n )    % if skinny
       L = chol( A'*A + rho*speye(n), 'lower' );
    else            % if fat
       L = chol( speye(m) + 1/rho*(A*A'), 'lower' );
    end

    % force matlab to recognize the upper / lower triangular structure
    L = sparse(L);
    U = sparse(L');
end
```

## 7. ADMM for total variable minimisation: (One of the powerful tools used in Signal and Image processing)

Problem formulation is as follows:

$$\text{minimize } \frac{1}{2}\|x-b\|_2^2 + \lambda\sum_{i=1}^{n-1}|x_{i+1}-x_i|, \quad x \in R^n$$

$$\Rightarrow \text{minimize} \frac{1}{2}\|x-b\|_2^2 + \lambda\|z\|_1$$

*Subject* to $z = Dx$

*Augmented* Lagrangian is

$$L(x,z,y) = \frac{1}{2}\|x-b\|_2^2 + \lambda\|z\|_1 + y^T(Dx-z) + \frac{\rho}{2}\|Dx-z\|_2^2$$

$$\underbrace{\begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{pmatrix}}_{z} = \underbrace{\begin{pmatrix} -1 & 1 & . & . & . & . & . & . \\ . & -1 & 1 & . & . & . & . & . \\ . & . & -1 & 1 & . & . & . & . \\ . & . & . & -1 & 1 & . & . & . \\ & & & & -1 & 1 & & \\ . & . & . & . & . & . & . & . \\ . & . & . & . & . & . & -1 & 1 \end{pmatrix}}_{D} \begin{pmatrix} x_1 \\ x_2 \\ x_2 \\ x_2 \\ \vdots \\ x_{n-1} \\ x_n \end{pmatrix}_{x}$$

Representation of constraint in matrix format.

x-update is as follows:

$$L(x, z^k, y^k) = \frac{1}{2}\left\|x-b\right\|_2^2 + \left(y^k\right)^T Dx + \frac{\rho}{2}\left\|Dx - z^k\right\|_2^2$$

$$x - b + D^T y^k + \rho D^T\left(Dx - z^k\right) = 0$$

$$\left(I + \rho D^T D\right)x = b - D^T y^k + \rho D^T z^k = b + \rho D^T\left(z^k - \frac{1}{\rho}y^k\right)$$

$$x^{k+1} = \left(I + \rho D^T D\right)^{-1}\left(b + \rho D^T\left(z^k - \frac{1}{\rho}y^k\right)\right) - - - (1)$$

z-update is as follows:

*Augmented* Lagrangian is

$$L(x, z, y) = \frac{1}{2}\left\|x-b\right\|_2^2 + \lambda\left\|z\right\|_1 + y^T\left(Dx - z\right) + \frac{\rho}{2}\left\|Dx - z\right\|_2^2$$

$$L(x^{k+1}, z, y^k) = \lambda\left\|z\right\|_1 - \left(y^k\right)^T z + \frac{\rho}{2}\left\|Dx^{k+1} - z\right\|_2^2$$

$$L(x^{k+1}, z, y^k) = \left\|z\right\|_1 + \frac{\rho}{2\lambda}\left\|Dx^{k+1} - z + \frac{y^k}{\rho}\right\|_2^2$$

$$z^{k+1} = S_{\lambda/\rho}\left(Dx^{k+1} + \frac{y^k}{\rho}\right)$$

Similarly, $y$(scaled u update) is:

$$y^{k+1} = y^k + Dx^{k+1} - z^k$$

## Code to execute is as follows:

```
clc;clear all;close all;
rand('seed', 0);
randn('seed', 0);
n = 100;
x0 = ones(n,1);

for j = 1:3
    idx = randsample(n,1);
    k = randsample(1:10,1);
    x0(ceil(idx/2):idx) = k*x0(ceil(idx/2):idx);
end

b = x0 + randn(n,1);
lambda = 5;
e = ones(n,1);
D = spdiags([e -e], 0:1, n,n);

x = total_variation(b, lambda, 1.0, 1.0);

function x = total_variation(b, lambda, rho, alpha)
% total_variation  Solve total variation minimization via ADMM
% Solves the following problem via ADMM:
%
%   minimize  (1/2)||x - b||_2^2 + lambda * sum_i |x_{i+1} - x_i|
% where b in R^n.
%
% The solution is returned in the vector x.
%
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter
    MAX_ITER = 1000;
    n = length(b);

    % difference matrix
    e = ones(n,1);
    D = spdiags([e -e], 0:1, n,n);
    x = zeros(n,1);
    z = zeros(n,1);
    u = zeros(n,1);
    I = speye(n);
    DtD = D'*D;

    for k = 1:MAX_ITER
         % x-update
        x = (I + rho*DtD) \ (b + rho*D'*(z-u));

        % z-update with relaxation
        zold = z;
        Ax_hat = alpha*D*x +(1-alpha)*zold;
        z = shrinkage(Ax_hat + u, lambda/rho);

        % y-update
        u = u + Ax_hat - z;
    end
end

function y = shrinkage(a, kappa)
    y = max(0, a-kappa) - max(0, -a-kappa);
end
```

## 8. Lasso using ADMM:

An important special case of regularized linear regression, also called the *lasso*. Lambda is a scalar regularization parameter that is usually chosen by cross-validation. In typical applications, there are many more features than training examples, and the goal is to find a parsimonious model for the data. The lasso has been widely applied, particularly in the analysis of biological data, where only a small fraction of a huge number of possible factors are actually predictive of some outcome of interest. It is prevalent across machine learning, model selection in statistics.
Problem formulation and solution is as follows:

min f(x) + g(x)

where f(x) = $\frac{1}{2}\|Ax-b\|^2$ ; g(x) = $\tau\|z\|_1$

*subj* $x - z = 0$

$$L(x,y,z) = (Ax-b)^T(Ax-b) + \tau\|z\|_1 + y^T(x-z) + \frac{\beta}{2}(x-z)^T(x-z)$$

$$L(x,y,z) = (Ax-b)^T(Ax-b) + \tau\|z\|_1 + \frac{\beta}{2}\|x-z+u\|^2$$

$$\Delta L_x(x,y,z) = A^T(Ax-b) + \beta(x-z+u) = 0$$

$$\Rightarrow x_{k+1} = \left(A^TA + \beta I\right)^{-1}\left(A^Tb + \beta\left(z^k - u^k\right)\right)$$

$$\Delta L_z(x,y,z) = \tau + \beta(x-z+u) = 0$$

$$\Rightarrow z_{k+1} = S_\lambda\left(x^{k+1} + u^k\right)$$

where $S_\lambda(c) = \begin{cases} c - \lambda(sign(c)) & \lambda \le c \\ 0 & \lambda > c \end{cases}$

$$\Delta L_u(x,y,z) = x^{k+1} - z^{k+1} = 0$$

$$u_{k+1} = u_k + x^{k+1} - z^{k+1}$$

**Code to execute is as follows:**

```
function [x,z,u] = lasso(A, b, lambda, rho, alpha)
% lasso  Solve lasso problem via ADMM
% Solves the following problem via ADMM:
%
%   minimize 1/2*|| Ax - b ||_2^2 + \lambda || x ||_1
%
% rho is the augmented Lagrangian parameter and alpha is the over-
relaxation parameter

    MAX_ITER = 1000;
    [m, n] = size(A);

    x = zeros(n,1);
    z = zeros(n,1);
    u = zeros(n,1);

    % cache the factorization
    [L U] = factor(A, rho);

    for k = 1:MAX_ITER

        % x-update
        q = Atb + rho*(z - u);    % temporary value
        if( m >= n )    % if skinny
           x = U \ (L \ q);
        else            % if fat
           x = q/rho - (A'*(U \ ( L \ (A*q) )))/rho^2;
        end

        % z-update with relaxation
        zold = z;
        x_hat = alpha*x + (1 - alpha)*zold;
        z = shrinkage(x_hat + u, lambda/rho);

        % u-update
        u = u + (x_hat - z);
    end
end

function z = shrinkage(x, kappa)
    z = max( 0, x - kappa ) - max( 0, -x - kappa );
end

function [L U] = factor(A, rho)
    [m, n] = size(A);
    if ( m >= n )    % if skinny
       L = chol( A'*A + rho*speye(n), 'lower' );
    else            % if fat
       L = chol( speye(m) + 1/rho*(A*A'), 'lower' );
    end

    % force matlab to recognize the upper / lower triangular structure
    L = sparse(L);
    U = sparse(L');
end
```

## Essential Methods that help in speeding up the process of solving problems using ADMM:

### a) Block Splitting:

The concepts of Computer software were written conventionally for serial computing. This meant that to solve a problem, an algorithm divides the problem into smaller instructions. These discrete instructions are then executed on Central Processing Unit of a computer one by one. Only after one instruction is finished, next one starts.

Parallel Computing is the use of multiple processing elements simultaneously for solving any problem. Problems are broken down into instructions and are solved concurrently as each resource which has been applied to work is working at the same time.

### Primal LPP Block Splitting:

Since the size of data needed to be optimised in ADMM is very huge, we need to divide into B blocks of equal sizes as follows:

$$x_1 = \begin{bmatrix} x_{1,1} \\ . \\ . \\ . \\ x_{1,B} \end{bmatrix}; \; A = \begin{bmatrix} A_1 & . & . & . & A_B \end{bmatrix}; \; \mathrm{c} = \begin{bmatrix} c_1 \\ . \\ . \\ . \\ c_B \end{bmatrix}$$

For the sake of derivation, we will do it using 2 blocks. So values become:

$$x_1 = \begin{bmatrix} x_{1,1} \\ x_{1,2} \end{bmatrix}; \; \mathrm{A} = \begin{bmatrix} A_1 & A_2 \end{bmatrix}; \; \mathrm{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

$$L_P(x_1, x_2, y) = c_1{}^T x_{1,1} + c_2{}^T x_{1,2} - y^T \left( A_1 x_{1,1} + A_2 x_{1,2} - b \right) - s_1{}^T \left( x_{1,1} - x_{2,1} \right) ....$$

$$.... - s_2{}^T \left( x_{1,2} - x_{2,2} \right) + \frac{\beta}{2} \left( \left\| A_1 x_{1,1} + A_2 x_{1,2} - b \right\|^2 + \left\| x_{1,1} - x_{2,1} \right\|^2 + \left\| x_{1,2} - x_{2,2} \right\|^2 \right)$$

$$\Delta L_{P, x_{1,1}}(x_1, x_2, y) = c_1 - A_1{}^T y - s_1 + \beta \left( A_1{}^T \left( A_1 x_{1,1} + A_2 x_{1,2} - b \right) + \left( x_{1,1} - x_{2,1} \right) \right)$$

$$\Delta L_{P, x_{1,2}}(x_1, x_2, y) = c_2 - A_2{}^T y - s_2 + \beta \left( A_2{}^T \left( A_2 x_{2,1} + A_2 x_{2,2} - b \right) + \left( x_{2,1} - x_{2,2} \right) \right)$$

Our Update statements will be:

$$\Rightarrow x_{1,1}^{k+1} = \left(A_1^T A_1 + I\right)^{-1}\left(\frac{1}{\beta}\left(-c_1 + A_1^T y + s_1\right) + \left(A_1^T b + x_{2,1}^{k} - A_1^T A_2 x_{1,2}^{k}\right)\right)$$

$$\Rightarrow x_{1,2}^{k+1} = \left(A_2^T A_2 + I\right)^{-1}\left(\frac{1}{\beta}\left(-c_2 + A_2^T y + s_2\right) + \left(A_2^T b + x_{2,1}^{k} - A_1^T A_2 x_{1,1}^{k}\right)\right)$$

The general statement for n blocks will be:

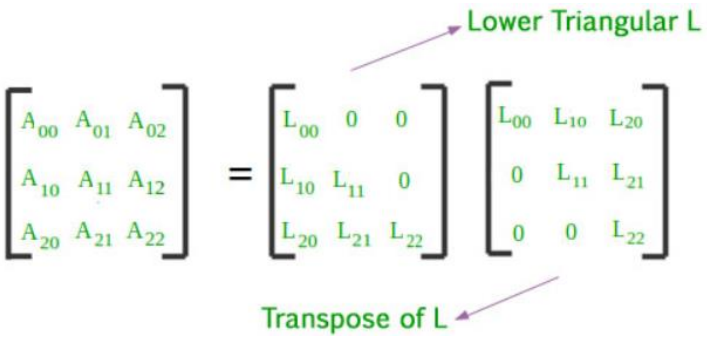
$$x_{1,i}^{k+1} = \left(A_i^T A_i + I\right)^{-1}\left(\frac{1}{\beta}\left(-c_i + A_i^T y + s_i\right) + \left(A_i^T b + x_{2,i}^{k} - \sum_{j=1;\, j\notin U} A_i^T A_j x_{1,j}^{k}\right)\right)$$

**b) Cholesky Decomposition:**

In linear algebra, a matrix decomposition or matrix factorization is a factorization of a matrix into a product of matrices. There are many different matrix decompositions. One of them is Cholesky Decomposition.

The Cholesky decomposition or Cholesky factorization is a decomposition of a Hermitian, positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. The Cholesky decomposition is roughly twice as efficient as the LU decomposition for solving systems of linear equations.

The Cholesky decomposition of a Hermitian positive-definite matrix A is a decomposition of the form $A = LL^T$, where L is a lower triangular matrix with real and positive diagonal entries, and $L^T$ denotes the conjugate transpose of L. Every Hermitian positive-definite matrix (and thus also every real-valued symmetric positive-definite matrix) has a unique Cholesky decomposition.



Every symmetric, positive definite matrix A can be decomposed into a product of a unique lower triangular matrix L and its transpose: $A = LL^T$. The following formulas are obtained by solving above lower triangular matrix and its transpose.

$$L_{i,j} = \sqrt{A_{j,j} - \sum_{k=0}^{j-1} L_{j,k}{}^2}$$

$$L_{i,j} = \frac{1}{L_{j,j}} \left( A_{j,j} - \sum_{k=0}^{j-1} L_{i,k} L_{j,k} \right)$$

Example of Cholesky Decomposition is as follows:

$$\begin{bmatrix} 4 & 12 & -16 \\ 12 & 37 & -43 \\ -16 & -43 & 98 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 6 & 1 & 0 \\ -8 & 5 & 3 \end{bmatrix} \begin{bmatrix} 2 & 6 & -8 \\ 0 & 1 & 5 \\ 0 & 0 & 3 \end{bmatrix}$$

**c) LU Decomposition:**

L U decomposition of a matrix is the factorization of a given square matrix into two triangular matrices, one upper triangular matrix and one lower triangular matrix, such that the product of these two matrices gives the original matrix. Basically, the L U decomposition method comes handy whenever it is possible to model the problem to be solved into matrix form. Conversion to the matrix form and solving with triangular matrices makes it easy to do calculations in the process of finding the solution.

Representation is as follows:

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & l_{33} \end{bmatrix} \begin{bmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{bmatrix}$$

$$A \qquad = \qquad \mathrm{L} \qquad \mathrm{U}$$

**Steps for LU Decomposition:**

Given a set of linear equations, first convert them into matrix form A X = C where A is the coefficient matrix, X is the variable matrix and C is the matrix of numbers on the right-hand side of the equations.

Now, reduce the coefficient matrix A, i.e., the matrix obtained from the coefficients of variables in all the given equations such that for 'n' variables we have an nXn matrix, to row echelon form using Gauss Elimination Method. The matrix so obtained is U.

To find L, we have two methods. The first one is to assume the remaining elements as some artificial variables, make equations using A = L U and solve them to find those artificial variables. The other method is that the remaining elements are the multiplier coefficients because of which the respective positions became zero in the U matrix.

Now, we have A (the nxn coefficient matrix), L (the nxn lower triangular matrix), U (the nxn upper triangular matrix), X (the nX1 matrix of variables) and C (the nx1 matrix of numbers on the right-hand side of the equations).
The given system of equations is A X = C. We substitute A = L U. Thus, we have LUX=C. We put Z = U X, where Z is a matrix or artificial variables and solve for L Z = C first and then solve for U X = Z to find X or the values of the variables, which was required.

**Example of LU Decomposition is as follows:**

$$\begin{bmatrix} 1 & 8 & 9 \\ 4 & 6 & 8 \\ -5 & -7 & -2 \end{bmatrix} = \begin{bmatrix} -0.2 & 1 & 0 \\ -0.8 & 0.06 & 1 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} -5 & -7 & -2 \\ 0 & 6.6 & 8.6 \\ 0 & 0 & 5.9 \end{bmatrix}$$

$$A \quad = \quad L \quad U$$