# ADMM-based Distributed Electric Vehicle Charging Optimisation Algorithm

**A PROJECT REPORT**

*Submitted by*


| | |
|---|---|
| **Advaith Krishna** | **- CB.SC.U4AIE24203** |
| **Amudala Jashwanth** | **- CB.SC.U4AIE24204** |
| **Bhuvan Rajasekar** | **- CB.SC.U4AIE24209** |
| **Sachcith G N** | **-CB.SC.U4AIE24251** |


Course Code

**22MAT220 - Mathematics for Computing - 3 (MFC - 3)**



**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**Ettimadai,COIMBATORE-641112**

**AMRITA SCHOOL OF ARTIFICIAL INTELLIGENCE**

**AMRITA VISHWA VIDYAPEETHAM**

**COIMBATORE-641112**

## DECLARATION

We, **Advaith Krishna - CB.SC.U4AIE24203, Amudala Jashwanth - CB.SC.U4AIE24204, Bhuvan Rajasekar - CB.SC.U4AIE24209 and Sachcith G N - CB.SC.U4AIE24251** hereby declare that the project work entitled "***ADMM-based Distributed Electric Vehicle Charging Optimisation Algorithm***" is the record of the original work done by us under the guidance of **Dr. Sunil Kumar S**, Assistant Professor, School of Artificial Intelligence, Amrita Vishwa Vidhyapeetham, Coimbatore.

**Advaith Krishna - CB.SC.U4AIE24203** **Amudala Jashwanth - CB.SC.U4AIE24204**

**Bhuvan Rajasekar - CB.SC.U4AIE24209** **Sachcith G N -CB.SC.U4AIE24251**

**Place:**Coimbatore – 641112

**Date: 20/08/2025**

## COUNTERSIGNED

**Dr. Sunil Kumar S,**

Assistant Professor,

## Introduction :

- The usage of electric vehicles is increasing rapidly.
- To keep up with this growth, the charging framework for electric vehicles needs to be optimized to reduce charging costs.
- Centralized charging strategies are currently being deployed
- Decentralizing the charging infrastructure can enhance scalability, improve computational efficiency, and reduce the risk of system failures, compared to that of Centralized framework

## Abstract :

The rapid adoption of electric vehicles (EVs) demands efficient charging frameworks that can minimise operational costs while ensuring reliable performance. Traditional centralised charging strategies, although effective, suffer from scalability issues and vulnerability to single-point failures. A decentralised approach distributes the computational load, enhances robustness, and better protects privacy. In this work, we recreate a distributed charging optimisation framework based on the Alternating Direction Method of Multipliers (ADMM) as given in the papet titled "ADMM-based Distributed Electric Vehicle Charging Optimisation Algorithm" . Here rather than making use of charging power, we are directly optimising charging currents thereby enabling the inclusion of important battery constraints such as state-of-charge and terminal voltage otherwise not possible. The proposed method achieves cost reduction without requiring a central coordinator.

## Problem Statement :

- To Decentralize the EV Charging points across the area, we can consider the points as Nodes in a graph G (undirected)
- $G = (V, E)$
- Where $V$ is non-empty set of nodes each representing an EV charging point and $E \subset V \times V$ is the set of edges representing connection between charging points.

- We model the count of EVs to be charged by assuming that a single charging point charges only one Battery of the EV at a given time.
- Optimizing the operational cost with respect to all Charging points of EV.

**Key Components:**

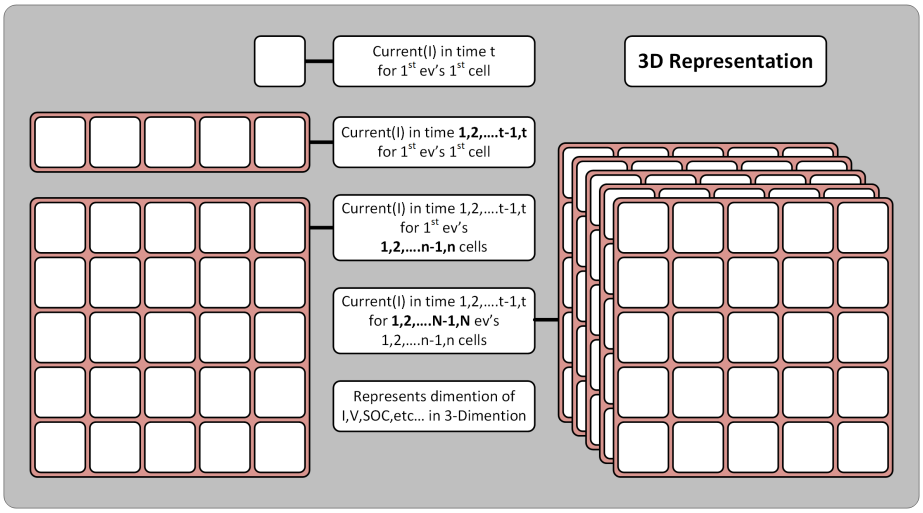
- Range vector transformations (`range2vector`, `constant2vector`)

- Block diagonal & structured matrices (`directSum`, `HMatrix`)

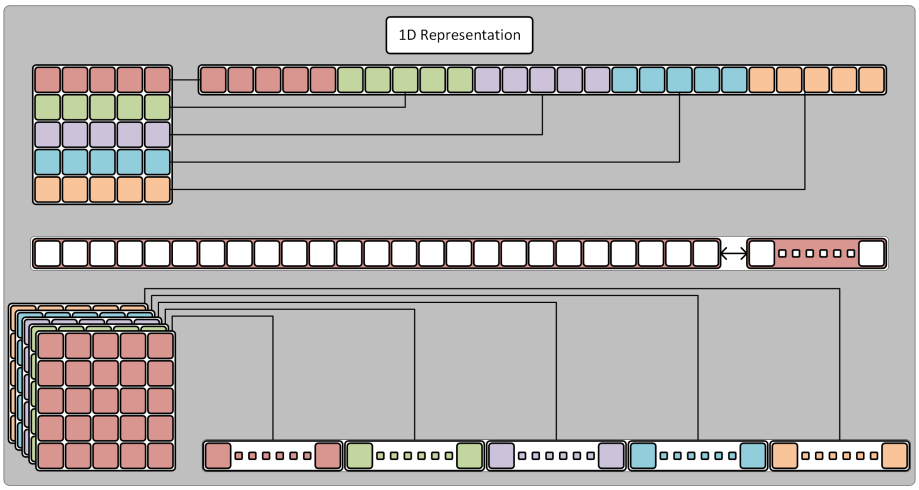
- Problem constraints: SOC, Voltage, Current

- Iterative ADMM update equations

**Data Description**




This is how the data is taken into a vector of size $R^{(Nnl\times 1)}$




```
clearvars; clear all; clc;
```

**Helper Functions :**

**range2vector()**

Converts a min/max range matrix into a long vector repeated across all

nodes and levels.

**Inputs:**

- `x` : Range matrix (N × 2)

- `is_max` : -1 → minimum, 1 → maximum

- `N, n, l` : Dimensions for replication

**Output:**

- Vector of size `(N*n*l, 1)`

```
function y=range2vector(x,is_max,N,n,l)
    if is_max==-1
        y = min(x');
```

```
    else
        y = max(x');
    end
    y = y.*ones(n*l,1);
    y = reshape(y,N*n*l,1);
end
```

**constant2vector()**

Replicates a constant parameter vector across all nodes and levels.

**Inputs:**

- `x` : Column vector of size (N×1) containing parameter values.

- `N, n, l` : Dimensions for replication

**Output:**

- `y` : Expanded column vector of size (N*n*l × 1).

```
function y=constant2vector(x,N,n,l)
    y = x';
    y = y.*ones(n*l,1);
    y = reshape(y,N*n*l,1);
end
```

**directSum()**

Constructs the block diagonal direct sum of two matrices:

**Inputs :**

Matrices $A$ *and* $B$

**Output :**

Matrix $Y = \begin{bmatrix} A & 0 \\ 0 & B \end{bmatrix}$

```
function y=directSum(A,B)
    [ax,ay] = size(A);
    [bx,by] = size(B);
    y = [A zeros(ax,by);zeros(bx,ay) B];
end
```

**lowertri()**

Multiplies a constant to the lower triangular ones matrix of input size l

**Inputs:**

constant x and size of matrix l

**Output:**

lowertriangular matrix multiplied with a constant

```
function y=lowertri(x,l)
```

```
    y = ones(l);
    y = tril(y);
    y = x.*y;
end
lowertri(1,4)
```

```
ans = 4×4
     1     0     0     0
     1     1     0     0
     1     1     1     0
     1     1     1     1
```

**HMatrix()**

Construct the block-diagonal constraint matrix **H** for the ADMM formulation.

Each row in `x` specifies an interval [start, end] for the index.

Based on this, we create diagonal "selector" vectors and build structured

block matrices.

**Inputs:**

- `x`  : (N × 2) matrix, each row = [start_index, end_index]

- `N`  : number of EVs

- `n`  : number of cells in an EV

- `l`  : No.pf Time frames

- `b_ij`  : weight matrix (N × N), diagonal entries used for scaling

**Output:**

- `y`  : Block-diagonal matrix of size `(N*n*l) × (N*n*l)`

```
function y=HMatrix(x,N,n,l,b_ij)
    temp = 1:l;
    y = [];
    for i=1:N
        index = zeros(l,1);
        index(temp>=x(i,1))=1;
        index(temp>x(i,2))=0;
        BD=lowertri(b_ij(i,i),l)*diag(index);
        for j=1:n
            y = directSum(y,BD);
        end
    end
end
% HMatrix(aidi,N,n,l,b_ij);
```

Assume $\Delta\eta = 0.25$

### Constraints Initialization

$N$, (No.of EV charged in the decentralised charging system)

$n$, (No.of Batteries present in an EV)

$l$, (Total no.of timeframes the charging states have been measured)

```
% clearvars;
N = 5
```

```
N = 5
```

```
n = 5
```

```
n = 5
```

```
l = 5
```

```
l = 5
```

$SOC_{range}$ is the Range of minimum and maximum State Of Charge for each EV

$I_{range}$ is the Range of minimum and maximum Current allowed to the cells of each EV

$V_{\text{range}}$ is the Range of minimum and maximum terminal voltage of cells in each EV

```
SOCrange = [0.1 1;0.1 1;0.1 1;0.1 1;0.1 1]
```

```
SOCrange = 5×2
    0.1000    1.0000
    0.1000    1.0000
    0.1000    1.0000
    0.1000    1.0000
    0.1000    1.0000
```

```
Irange = [0 12;0 15;0 15;0 15;0 20]
```

```
Irange = 5×2
     0    12
     0    15
     0    15
     0    15
     0    20
```

```
Vrange = [3.2 5;3.2 5;3.2 5;3.2 5;3.2 5]
```

```
Vrange = 5×2
    3.2000    5.0000
    3.2000    5.0000
    3.2000    5.0000
    3.2000    5.0000
    3.2000    5.0000
```

$f$is the open circuit voltage for each EV

$y$ is the internal resistance for each EV

```
f = [3.8; 4; 4.2; 4.1;3.6];
y = [0.04; 0.05; 0.08; 0.06; 0.03];
```

Constant $\omega_i, \nu_i$ used in the Optimization problem

```
w = [-22.6; -20; -13.8; -18.8; -27]
```

```
w = 5×1
  -22.6000
  -20.0000
  -13.8000
  -18.8000
  -27.0000
```

```
% temp = -0.25*f.*f./y
% temp./w
v = [50; 40; 25; 33.3; 66.7]
```

```
v = 5×1
   50.0000
   40.0000
   25.0000
   33.3000
   66.7000
```

```
% temp = 2./y
```

Translating Minimum and Maximum values for each field (for an EV - N total Values) to dimension of $(Nnl \times 1)$

Now each vector belongs to $\mathbb{R}^{\mathrm{Nnl}}$

They transcend for Each $EV_N$, Each of their $\mathrm{cell}_n$ and Each Timeframe $t_l$ as shown above

```
SOCmin = range2vector(SOCrange,-1,N,n,l);
SOCmax = range2vector(SOCrange,1,N,n,l);

Imin = range2vector(Irange,-1,N,n,l);
Imax = range2vector(Irange,1,N,n,l);

Vmin = range2vector(Vrange,-1,N,n,l);
Vmax = range2vector(Vrange,1,N,n,l);
```

Translating Constant values of $f$ and $y$ for each EV to dimension of $(Nnl \times 1)$

Now each vector belongs to $\mathbb{R}^{\mathrm{Nnl}}$

They transcend for Each $EV_N$, Each of their $\mathrm{Battery}_n$ and Each Timeframe $t_l$ as shown above

```
fi = constant2vector(f,N,n,l);
yi = constant2vector(y,N,n,l);
```

Constant $\kappa_i$ used in the Optimization problem

```
ki = -0.5*fi./yi;
```

To multiply using while loop easier we take the vector as a diagonal matrix

```
yi = diag(yi);
```

Constant $\omega_i, \nu_i$ used in the Optimization problem

```
wi = constant2vector(w,N,n,l);
vi = constant2vector(v,N,n,l);
vi = diag(vi);
```

Drawing the Undirected graph of the 5 EVs considered

```
D = diag([2 3 2 2 1]); % No.of Nodes connected to i_th index
A = [0 1 0 0 1;1 0 1 1 0;0 1 0 1 0;0 1 1 0 0;1 0 0 0 0]; % Adjacency Matrix
L = D-A; % Laplacian Matrix
```

State of Charge Initial = $SOC_{min}$

Current Initial = $0$

```
SOCin = SOCmin;
Iin = zeros(N*n*l,1);
```

$\mu = 0.95$= Coulombic Efficiency - Assumed to be same for all EVs

$\Delta = 4$ = Timeframe resolution assumption

$a_i$ is the arrival time index

$d_i$ is the intended departure time index

$q_i$ is the coulombic efficiency of each cell in $i^{th}$ ev's $j^{th}$ cell

```
aidi = [1 3;2 4;2 5;1 4;4 5]
```

```
aidi = 5×2
     1     3
     2     4
     2     5
     1     4
     4     5
```

```
qi = [1;2;3;4;5];
qij = ones(1,n).*qi;
muDelta = 0.95*4;
```

$b_{i,j}$ = $\mu\Delta/Q_{i,j}$ is a constant defined for each cell Signifying the relationship between current in the cell with the State of Charge of the cell

```
b_ij = muDelta./qij;
```

Block Diagonal Constraint = the cell charges in the timeframe between arrival and departure time index

The H Matrix selects the cell instances where the constraints are satisfied and take their $b_{i,j}$ into consideration for further computations

```
H = HMatrix(aidi,N,n,l,b_ij);
```

```
rank(H)
```

```
ans = 80
```

```
size(H)
```

```
ans = 1×2
   125   125
```

# ADMM

**Objective Function**

**The Power** utilised by the Electric Vehicle at time t will be

$$P_i(t) = \sum_{j=1}^{n} I_{B_{i,j}}(t) V_{B_{i,j}}(t)$$

where,

$I$ is the current in $i^{th}$ EV's $j^{th}$ battery at time t

$V$ is the voltage in $i^{th}$ EV's $j^{th}$ battery at time t

and the product of $I$ and $V$ is added for $n$ batteries per EV

**The Operational Costs** for operating $i^{th}$ EVs for total time,

$$F(I_{B_i}) = \sum_{t=1}^{l} \Delta \eta P_i(t)$$

where,

$I$ is the Current for each EV

$\Delta$ is the time frame resolution

$\eta$ is the cost of electricity for unit energy

$P$ is the power utilised by the EV at time t

and the product of $\Delta$, $\eta$, $P$ are added for $l$ timeframe resolutions considered

**The Cost of Operating** all the EVs for total time,

$$F(I_B) = \sum_{i=1}^{N} F(I_{B_i})$$

$F(I_{B_i})$ is the operational cost for $i^{th}$ EV for total time

and we sum up for all the EVs to get the cost function

**Problem Formulation**

$\min_{I_B} F(I_B)$ s.t. $I_B^{min} \leq I_B \leq I_B^{max}$

$SOC^{min} \leq SOC^0 + HI_B \leq SOC^{max}$

$V_B^{min} \leq f + yI_B \leq V_B^{max}$

The Constraints can restrict the domain into a small space

Let $A$ be

$$\begin{bmatrix} I^{\text{Nnl}} \\ -I^{\text{Nnl}} \\ H^{\text{Nnl}} \\ -H^{\text{Nnl}} \\ y^{\text{Nnl}} \\ -y^{\text{Nnl}} \end{bmatrix}$$ belonging to $R^{6Nnl \times Nnl}$

and $C$ be

$$\begin{bmatrix} I_B^{\min} \\ -I_B^{\max} \\ \text{SOC}^{\min} - \text{SOC}^0 \\ \text{SOC}^0 - \text{SOC}^{\max} \\ V_B^{\min} - f \\ f - V_B^{\max} \end{bmatrix}$$ belonging to $R^{6Nnl \times Nnl}$

And then we can write the conditions as

$AI_B \geq C$

and define the set as $\Omega = \{I_B : AI_B \geq C\}$

**So the Final Objective function can be written as,**

$min_{I_B} F(I_B) \ s.t. \ I_B \ \epsilon \ \Omega$

(or)

$min_{I_B} F(I_B) \ s.t. \ AI_B \geq C$

ADMM solution

① We use an indicator function.

$$g_1(S) = \begin{cases} 0, & S \in \Omega \\ \infty, & \text{otherwise.} \end{cases}$$

Then the set is dragged into the problem.

$$\min_{I_B, S \in R^{N \times L}} \quad F(I_B) + g_1(S)$$

$$\text{s.t.} \quad I_B - S = 0.$$

The Lagrangian.

$$L_{(I_B, S, \lambda)} = F(I_B) + g_1(S) + \lambda^T (I_B - S)$$

Augmented Lagrangian

$$L_{\rho(I_B, S, \lambda)} = F(I_B) + g_1(S) + \lambda^T (I_B - S) + \frac{\rho}{2} \| I_B - S \|_2^2$$

Second form ( Paper writes $u$ as $\lambda^t/\rho$ )

$$L_{\rho(I_B, S, \lambda)} = F(I_B) + g_1(S) + \frac{\rho}{2} \| I_B - S + \lambda^t/\rho \|_2^2$$

$$I_B^{k+1} = \arg\min_{I_B} L_\rho (I_B, S, \lambda)$$

$$= \arg\min_{I_B} \underbrace{\left( F(I_B) + \frac{\rho}{2} \| I_B - S + \lambda^t/\rho \|_2^2 \right)}_{\Phi}$$

$$I_B^{k+1} \Rightarrow \nabla_{IB} F(I_B) + \frac{\rho}{\cancel{4}} \left( I_B - S + \frac{\lambda^{th}}{\rho} \right) = 0$$

$\beta_i \Rightarrow$ Gradient of $\Phi$

Combining Eq. 1 and Eq. 2

$$F(I_B) = \Delta\eta \, I_B * (f + y I_B)$$

$$= \Delta\eta \left( I_B f + y I_B^2 \right)$$

$$= y \, \Delta\eta \left( \frac{2 I_B f}{2y} + I_B^2 \right)$$

$$= y \, \Delta\eta \left( \frac{2 I_B f}{2y} + I_B^2 + \frac{f^2}{4y^2} - \frac{f^2}{4y^2} \right)$$

$$= y \, \Delta\eta \left( \left( I_B + \left( \frac{f}{2y} \right) \right)^2 - \frac{f^2}{4y^2} \right)$$

$$= y \, \Delta\eta \left( \left( I_B - k \right)^2 - k^2 \right)$$

where $k = \frac{-f}{2y}$

$$= \cancel{y \Delta\eta} \left( -y \cdot \Delta\eta \cdot k^2 + y \Delta\eta \left( I_B - k \right)^2 \right)$$

$\omega = -y\Delta\eta k^2$

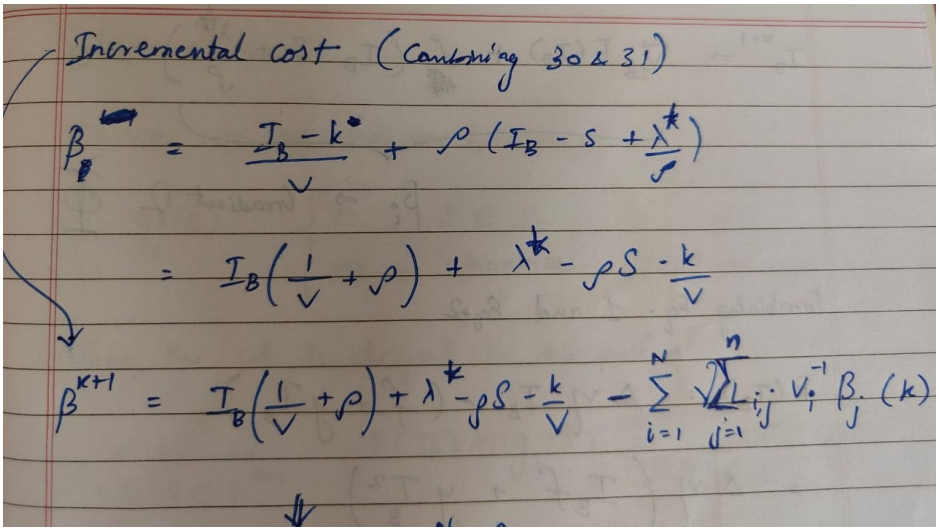
$$= \omega_k + y \Delta\eta \left( I_B - k \right)^2$$

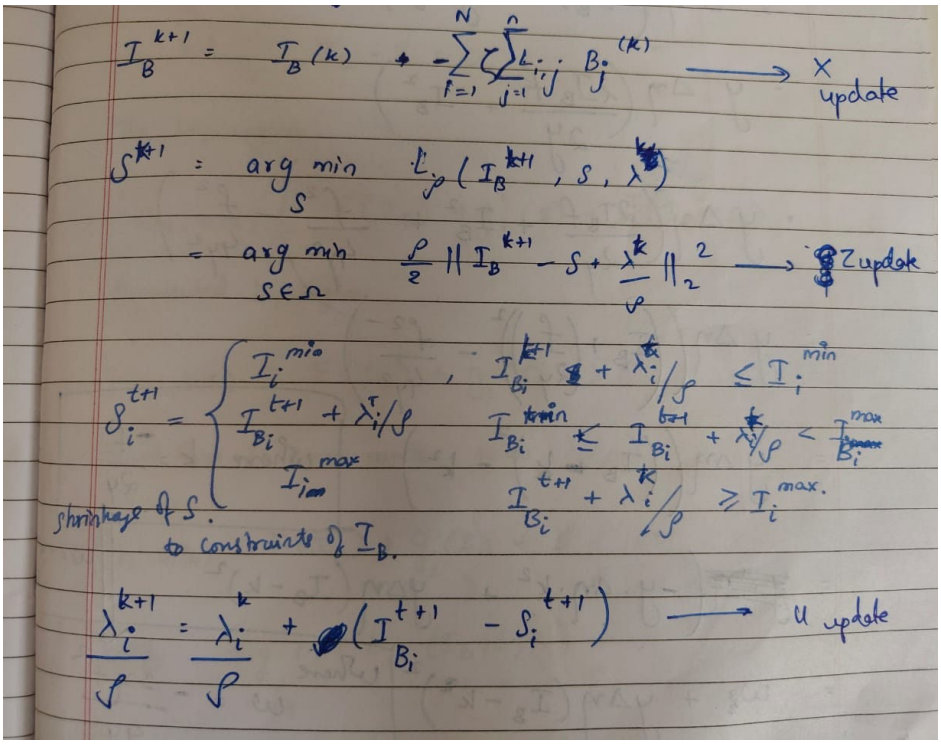
where $\omega = -\frac{\Delta\eta f^2}{4y}$

$$= \omega + \frac{\left( I_B - k \right)^2}{2V} \; //$$

where $V = \frac{1}{2\eta\Delta y}$

$$\nabla_{I_B} F(I_B) = \frac{I_B - k}{V}$$




### Initialization Parameters

```
rng(242)
rho=1.6;
I0 = Iin;
lambda0 = randi([0,10],size(Imin));
S_ij0 = randi([0,10],size(Imin));
beta0 = zeros(N*n*l,1);
design = 2;
tol = 1e-5;
max_iter=10;
converged = false;
```

### Matrix / Vector Reshaping

```
viVector = diag(vi);
viVector = reshape(viVector,25,5);
```

```
sig_min = SOCmin-SOCin;
sig_max = SOCmax-SOCin;

del_min = Vmin - fi;
del_max = Vmax - fi;

IminActual = max(Imin,max(pinv(H)*sig_min,yi\del_min));
ImaxActual = min(Imax,min(pinv(H)*sig_max,yi\del_max));
```

**Iteration**

```
for i=1:max_iter
    betaTemp = reshape(beta0,25,5);
    graphtemp = design*betaTemp*L./viVector;
    graphtemp1 = reshape(graphtemp,N*n*l,1);

    beta1 = vi\((rho*vi+ones(size(vi)))*I0 + vi*(lambda0-rho.*S_ij0)-ki)-
graphtemp1;
    I1 = I0 - reshape(graphtemp.*viVector,[],1);

    S_ij1 = I1 + lambda0/rho;
    S_ij1(S_ij1<IminActual)=IminActual(S_ij1<IminActual);
    S_ij1(S_ij1>ImaxActual)=ImaxActual(S_ij1>ImaxActual);

    lambda1 = lambda0 + rho*(I1-S_ij1);

    if norm(S_ij1-S_ij0)<tol
        converged = true;
        break;
    end

    beta0 = beta1;
    I0 = I1;
    S_ij0 = S_ij1;
    lambda0 = lambda1;
end

I0
```

```
I0 = 125×1
10^7 ×
    0.2129
    0.0273
   -0.1379
    0.2082
   -0.1971
    0.0089
    0.6247
   -0.9523
   -0.1267
   -0.1722
     ⋮
     ⋮
```