Dynamic Network Inference in Finance

Time-Varying Graphical Lasso for Portfolio Optimization

Authors: Shakil Rohimun & Chaker Meraihi
Degree Program: Master's in Quantitative Finance & ENSIIE Applied Mathematics
Instructor: Prof. VO Thi Phuong Thuy
Date: January 2025

Contents

Abstract

Graphical models provide a robust framework for modeling dependencies among variables in multivariate datasets. The Time-Varying Graphical Lasso (TVGL) extends this framework to dynamic data, enabling the discovery of evolving network structures. This report introduces the theoretical foundations of graph models and sparse precision matrix estimation using the Graphical Lasso and TVGL. It discusses the mathematical formulation, algorithms, and practical applications, highlighting their utility in financial data analysis.

Keywords: TVGL, Precision Matrices, Financial Networks, Covariance Estimation, Network Dynamics

1 Introduction

Networks are essential tools for understanding complex systems, providing a natural way to represent relationships among multiple entities. In a network representation, entities are modeled as nodes, while relationships or dependencies are depicted as edges. For multivariate datasets, such dependencies are often captured using **graphical models**, which encode conditional independence relationships among random variables.

1.1 Graphical Models: Overview

Graphical models are probabilistic frameworks that combine graph theory and probability theory to represent and analyze the dependencies among random variables. They provide a structured way of encoding and visualizing complex joint distributions, enabling efficient computation and reasoning in high-dimensional datasets.

A graphical model is represented as a graph $G = (V, E)$, where V is the set of vertices (nodes) representing random variables and E is the set of edges denoting relationships between these variables. The type of edges (directed or undirected) determines the nature of the model.

1.1.1 Types of Graphical Models

There are two primary types of graphical models, each suited to different kinds of dependency structures:

- **Directed Graphical Models (Bayesian Networks):** These models represent causal relationships between variables. An edge $X \to Y$ indicates that X directly influences Y. Directed models are particularly useful in situations where the directionality of influence is known or can be inferred.
- **Undirected Graphical Models (Markov Random Fields):** These models encode symmetric dependencies, focusing on conditional independencies. An edge $X - Y$ in an undirected graph indicates that X and Y are directly dependent, conditional on the remaining variables.

Each type of model has its advantages and applications. Bayesian networks excel in causal inference and sequential decision-making, while Markov random fields are well-suited for spatial and image data.

1.1.2 Gaussian Graphical Models (GGMs)

In the domain of continuous random variables, **Gaussian Graphical Models (GGMs)** are particularly popular due to their simplicity, interpretability, and computational efficiency. A GGM assumes that the data follows a multivariate Gaussian distribution:

$$\mathbf{X} \sim \mathcal{N}(\mu, \Sigma),$$

where μ is the mean vector and Σ is the covariance matrix. The structure of the GGM is determined by the **precision matrix**, defined as:

$$\Theta = \Sigma^{-1}.$$

The entries of the precision matrix encode conditional dependencies among the variables:

$$\Theta_{ij} = 0 \iff X_i \perp X_j \mid \text{rest}.$$

Key Properties of GGMs:

- **Diagonal Elements:** The diagonal elements of the precision matrix, Θ_{ii}, are inversely related to the conditional variances of the variables given all other variables.
- **Off-Diagonal Elements:** The off-diagonal elements, Θ_{ij}, indicate partial correlations. Specifically, they quantify the strength of the linear dependency between X_i and X_j, conditioned on all other variables.
- **Graph Structure:** The sparsity pattern of Θ (i.e., the location of zero entries) determines the structure of the GGM. A zero at position (i, j) in Θ implies that X_i and X_j are conditionally independent.

Advantages of GGMs:

- **Interpretability:** The sparse structure of the precision matrix provides a clear and interpretable representation of conditional independencies.
- **Scalability:** Computational techniques, such as sparse matrix factorization, make GGMs scalable for high-dimensional data.
- **Flexibility:** GGMs can be extended to incorporate regularization techniques, such as L_1-norm penalties, to enforce sparsity and improve generalization in high-dimensional settings.

Graphical Representation of GGMs: In GGMs, nodes represent variables, and edges represent non-zero entries in the precision matrix. For example, consider a GGM for three variables, X_1, X_2, X_3. If the precision matrix Θ has the structure:

$$\Theta = \begin{bmatrix} \Theta_{11} & \Theta_{12} & 0 \\ \Theta_{12} & \Theta_{22} & \Theta_{23} \\ 0 & \Theta_{23} & \Theta_{33} \end{bmatrix},$$

the corresponding graph G would have edges between X_1 and X_2, and between X_2 and X_3, but no edge between X_1 and X_3. This reflects the conditional independence $X_1 \perp X_3 \mid X_2$.

Learning GGMs: The goal of learning a GGM is to estimate the precision matrix Θ from data. The maximum likelihood estimate of Θ is given by:

$$\hat{\Theta} = S^{-1},$$

where S is the empirical covariance matrix. However, when the number of variables exceeds the number of observations ($p > n$), S is rank-deficient, and the estimate $\hat{\Theta}$ is ill-posed.

To address this issue, regularization techniques, such as the Graphical Lasso, are employed. These methods impose sparsity constraints on Θ, ensuring both numerical stability and interpretability.

Applications of GGMs: GGMs have found widespread use in various domains:

- **Finance:** Modeling dependencies among financial instruments, such as stocks or currencies.
- **Genomics:** Inferring gene regulatory networks from expression data.
- **Neuroscience:** Understanding functional connectivity in the brain by analyzing fMRI data.
- **Social Networks:** Identifying hidden relationships among individuals or groups.

In summary, Gaussian Graphical Models provide a powerful framework for analyzing multivariate dependencies, leveraging the sparsity of the precision matrix to extract interpretable insights from high-dimensional data.

1.2 Challenges in Estimating Precision Matrices

Estimating the precision matrix $\Theta = \Sigma^{-1}$ from data is a central problem in multivariate statistics, as it encodes the conditional dependency structure of variables. However, this estimation becomes particularly challenging in high-dimensional settings where the number of variables (p) exceeds the number of observations (n). In such cases:

- **Singularity of the Sample Covariance Matrix:** The sample covariance matrix S becomes singular when $p > n$, rendering its inverse S^{-1} undefined. This problem is common in high-dimensional applications like genomics and finance.
- **Overfitting in Dense Precision Matrices:** Standard estimators, such as the maximum likelihood estimator (MLE) for Θ, often yield dense precision matrices, making it difficult to interpret the underlying dependency structure. Dense matrices also lead to overfitting, particularly when the sample size is small relative to p.
- **Interpretability and Sparsity:** In many real-world applications, the true precision matrix is sparse, meaning that most variables are conditionally independent. Estimators that do not enforce sparsity fail to capture this structure, making the resulting precision matrix less interpretable.
- **Numerical Stability:** The inverse of the sample covariance matrix S^{-1} can be highly sensitive to noise in the data, leading to unstable estimates of Θ.

These challenges necessitate regularized approaches that enforce sparsity and robustness in the precision matrix estimation, especially in high-dimensional settings.

2 Problem Formulation

In this section, we introduce the problem formulated above and discuss the ADMM-based solution proposed by the authors of the article [1].

2.1 Time-Varying Graphical Lasso as a Convex Optimization Problem

Suppose we have a sequence of synchronous, time-stamped signals $\mathbf{x}_t$ evolving over time stamps $t = 1, \ldots, T$. The signals are sampled from a multivariate, zero-mean, Gaussian distribution in $\mathbb{R}^p$, such that:

$$\mathbf{x}_t \sim \mathcal{N}(0, \Sigma_t).$$

For each time stamp t, we observe only n_t signals from p variables. Let:

$$S_t = \frac{1}{n_t} \sum_{i=1}^{n_t} \mathbf{x}_t^{(i)} \mathbf{x}_t^{(i)T}$$

be the empirical covariance matrix of the observed data at time t. Finally, let $\Theta_t \in \mathcal{S}_{++}^p$ denote the symmetric positive definite matrix corresponding to the inverse covariance matrix of the observed data at time t. The time-varying graphical lasso (TVGL) problem is then defined as:

$$\min_{\Theta_1, \ldots, \Theta_T \in \mathcal{S}_{++}^p} \sum_{t=1}^{T} \Big(-\ell_t(\Theta_t) + \lambda \|\Theta_t\|_{\text{od},1} \Big) + \beta \sum_{t=2}^{T} \psi(\Theta_t - \Theta_{t-1}),$$

where:

- $-\ell_t(\Theta_t) = n_t \left(\log\det \Theta_t - \operatorname{tr}(S_t \Theta_t) \right)$ is the negative log-likelihood of Θ_t (up to a constant and a scale),
- $\|\Theta_t\|_{\text{od},1} = \sum_{i \neq j} |\Theta_{t,ij}|$ is a seminorm of Θ_t, promoting sparsity, and
- $\psi(\Theta_t - \Theta_{t-1})$ is a convex penalty that enforces temporal consistency.

Different penalty functions $\psi(\cdot)$ will be analyzed below. We remark that when S_t is full rank, $-\ell_t(\Theta_t)$ encourages Θ_t to be close (in the PSD ordering) to S_t^{-1}. However, when S_t is not full rank, the penalty term $\psi(\Theta_t - \Theta_{t-1})$ enforces structural similarity between Θ_t and Θ_{t-1}, allowing the estimate of Θ_t to partially rely on the information gained from S_t' for $t' \approx t$.

The parameters λ and β are positive regularization parameters that control different network behaviors. A larger λ results in sparser estimates of Θ_t, but may lead to Θ_t diverging from the true underlying inverse covariance matrix of the network. Conversely, a larger β reduces fluctuations in the value of Θ_t over time.

Remark: Convexity of the Objective.
We note that the above optimization problem is convex. The negative log-likelihood $-\ell_t(\Theta_t)$ is known to be convex in its argument (as described above), all seminorms are convex due to homogeneity and the triangle inequality, and $\psi(\cdot)$ is assumed to be convex. Since the positive sum of convex functions is again convex, we have a convex objective. Furthermore, the constraint set $\mathcal{S}_{++}^p$ is a convex set, and the Cartesian product of convex sets is also convex.

2.2 Network Evolution Penalty

The major contribution of the authors above is the time-varying nature of the graphical lasso problem. To ensure a solution that is not overly sensitive to minor changes, the authors introduced the term $\psi(\Theta_t - \Theta_{t-1})$ in the above optimization problem to penalize changes in the network structure. The choice of the function ψ depends on the application and should be informed by prior knowledge about how the network is expected to evolve.

We describe two commonly used penalties for ψ and their interpretations to highlight the versatility of this framework:

Group Lasso ℓ_2-Norm Penalty: The ℓ_2-norm penalty is defined as:

$$\psi(X) = \sum_j \|[X]_j\|_2,$$

where $[X]_j$ is the j^{th} column of X. This penalty allows the entire graph to change at select points in time, while encouraging the graph to remain identical at all other points. This is useful in applications where the objective is to detect significant events that would cause the network to restructure.

Elementwise Laplacian Penalty: The Laplacian penalty is defined as:

$$\psi(X) = \sum_{i,j} (X_{ij})^2.$$

This penalty encourages the network structure to vary smoothly over time, avoiding abrupt changes in edges. It is particularly useful for high-frequency observations where gradual evolution of the network is expected.

The choice of ψ depends on the domain and objectives. For example, the Laplacian penalty is better suited for applications requiring smooth transitions, while the ℓ_2-norm penalty is ideal for identifying major structural changes.

3 Problem Solution

We now turn to the solution to TVGL proposed by the authors above. The problem posed earlier is a convex optimization problem in T variables, and it is possible to solve it using standard methods like the interior point method, Newton's method, or the subgradient method (note that the lasso penalty is not differentiable). However, due to the large number of decision variables and the high dimensionality of the graph, such naive methods are computationally expensive.

To address this, the authors propose a much faster algorithm leveraging the additive structure of the problem. Specifically, they employ a distributed convex optimization method known as the Alternating Direction Method of Multipliers (ADMM). By recasting the problem using consensus optimization, the authors reformulate the objective to have a separable structure. This reformulation enables efficient distribution and parallelization of the optimization process.

4 The ADMM Solution

4.1 Alternating Direction Method of Multipliers (ADMM)

ADMM is a robust, distributed method for solving equality-constrained, decomposable optimization problems. It is formally a method for solving convex problems of the form:

$$\min f(x) + g(z) \quad \text{subject to } Ax + Bz = c$$

ADMM introduces an augmented Lagrangian and solves the problem via dual ascent. The augmented Lagrangian is of the form:

$$\mathcal{L}_\rho(x, z, \lambda) = f(x) + g(z) + \lambda^\top (Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2$$

which can also be expressed as:

$$\mathcal{L}_\rho(x, z, \lambda) = f(x) + g(z) + (\rho/2)\|Ax + Bz - c + (1/\rho)\lambda\|_2^2 - (1/2\rho)\|\lambda\|_2^2.$$

Here, the parameter ρ acts as a learning rate for the dual ascent problem, and the term $\|Ax+Bz-c\|_2^2$ ensures convergence by driving the problem to a feasible solution at every step, even when performing updates separately for x and z.

The ADMM algorithm works by alternating updates for each variable while holding the others constant. The following steps are repeated until convergence:

$$x^{k+1} = \arg\min_x f(x) + (\rho/2)\|Ax + Bz^k - c + (1/\rho)\lambda^k\|_2^2 \quad \text{(x-update)}$$

$$z^{k+1} = \arg\min_z g(z) + (\rho/2)\|Ax^{k+1} + Bz - c + (1/\rho)\lambda^k\|_2^2 \quad \text{(z-update)}$$

$$\lambda^{k+1} = \lambda^k + \rho(Ax^{k+1} + Bz^{k+1} - c) \quad \text{(dual update)}.$$

The above updates are efficient due to their separability. ADMM is guaranteed to converge to a global optimum for convex problems due to strong duality.

4.2 Consensus Optimization

Consensus is a method for solving convex problems that decompose into smaller subproblems. It reformulates the optimization problem by introducing auxiliary variables to allow parallel computation. The equivalence between the two problems below demonstrates this:

$$\min \sum_{i=1}^N f_i(x_i) \iff \min \sum_{i=1}^N f_i(x_i) \quad \text{subject to } x_i - z = 0, \forall i = 1, \ldots, N.$$

By introducing the consensus constraint $x_i - z = 0$, the original problem is reformulated such that each subproblem can be solved independently. This allows for efficient distributed computation.

4.3 TVGL Solution via ADMM and Consensus Optimization

To compute the TVGL solution efficiently, we introduce a set of auxiliary consensus variables:

$$Z = \{Z_0, Z_1, Z_2\} = \{(Z_{1,0}, \ldots, Z_{T,0}), (Z_{1,1}, \ldots, Z_{T-1,1}), (Z_{2,2}, \ldots, Z_{T,2})\}.$$

Using these variables, the optimization problem can be rewritten as:

$$\min_{\Theta \in \mathcal{S}_{++}^p} \sum_{t=1}^{T} -\ell_t(\Theta_t) + \lambda \|Z_{t,0}\|_{\text{od},1} + \beta \sum_{t=2}^{T} \psi(Z_{t,1} - Z_{t-1,1}),$$

subject to the constraints:

$$\begin{aligned} Z_{t,0} = \Theta_t, \quad \Theta_t \in \mathcal{S}_{++}^p \quad \text{for } t = 1, \ldots, T, \\ (Z_{t-1,1}, Z_{t,2}) = (\Theta_{t-1}, \Theta_t), \quad \text{for } t = 2, \ldots, T. \end{aligned}$$

This reformulation introduces separability into the problem, enabling the use of ADMM to optimize Θ_t, Z, and dual variables in parallel. This approach dramatically improves computational efficiency.

4.4 The ADMM Solution

Using the matrix version of the augmented Lagrangian with scaled dual variable $U = \{U_0, U_1, U_2\}$ introduced in Section 3.1, the authors write the augmented Lagrangian as:

$$\begin{aligned} \mathcal{L}_\rho(\Theta, Z, U) = \sum_{t=1}^{T} \Big(- \ell_t(\Theta_t) + \lambda \|Z_{t,0}\|_{\text{od},1} + \beta \sum_{t=2}^{T} \psi(Z_{t,2} - Z_{t-1,1}) + \frac{\rho}{2} \Big(\|\Theta_t - Z_{t,0} + U_{t,0}\|_F^2 \\ + \|\Theta_{t-1} - Z_{t-1,1} + U_{t-1,1}\|_F^2 - \|\Theta_t - Z_{t,2} + U_{t,2}\|_F^2 \Big) \Big). \end{aligned}$$

The ADMM algorithm is used to sequentially minimize the variables Θ then Z, and subsequently update the scaled dual variable U.

4.4.1 Θ Update

With the introduction of the consensus variables, the update for each Θ_t is fully separable from all other $\Theta_{t'}$. If we allow $A = Z_{t,0} + Z_{t,1} + Z_{t,2} - U_{t,0} - U_{t,1} - U_{t,2}$ and $\eta = \frac{n_t}{3p}$, the authors write:

$$\Theta_t^{k+1} = \arg \min_{\Theta_t \in \mathcal{S}_{++}^p} - \log \det(\Theta_t) + \text{tr}(S_t \Theta_t) + \frac{1}{2\eta} \Big\| \Theta_t - \frac{A + A^\top}{2} \Big\|_F^2$$

$$\Theta_t^{k+1} = \text{prox}_{\eta(-\log\det(\cdot) + \text{tr}(S_t))} \Big(\frac{A + A^\top}{2} \Big) = \frac{\eta}{2} Q \Big(D + \sqrt{D^2 + 4\eta^{-1} I} \Big) Q^\top,$$

where $QDQ^\top$ is the eigendecomposition of $\frac{A+A^\top}{2\eta} - S_t$. This is the most computationally expensive part of the algorithm, requiring $O(p^3)$ runtime.

4.4.2 Z Update

The update for each $Z_{t,0}$ can be fully separated from the update for (Z_1, Z_2) as well as the other $Z_{t',0}$. The update for $Z_{t,0}$ is easily written in the form of a proximal operator:

$$Z_{t,0}^{k+1} = \arg\min_{Z_{t,0}} \mathcal{L}_\rho(\Theta, Z, U^k) = \arg\min_{Z_{t,0}} \lambda\|Z_{t,0}\|_{\text{od},1} + \frac{\rho}{2}\|\Theta_t^{k+1} + U_{t,0}^k - Z_{t,0}\|_F^2 = \text{prox}_{\lambda/\rho\|\cdot\|_{\text{od},1}}(\Theta_t^{k+1} + U_{t,0}^k).$$

The proximal operator is then defined element-wise, yielding a final solution of:

$$[Z_{t,0}^{k+1}]_{i,j} = \begin{cases} \text{sgn}\Big(([\Theta_t^{k+1} + U_{t,0}^k]_{i,j})\Big)\Big(\big|[\Theta_t^{k+1} + U_{t,0}^k]_{i,j}\big| - \frac{\lambda}{\rho}\Big)_+ & i \neq j, \\ [\Theta_t^{k+1} + U_{t,0}^k]_{i,i} & i = j. \end{cases}$$

The most complex part of the update is the joint update of $Z_{t,1}, Z_{t,2}$. Note that each of these can be separated from all other updates $Z_{t',1}, Z_{t',2}$. In order to simplify notation, the authors define $\psi\Big(\begin{bmatrix} Z_1 \\ Z_2 \end{bmatrix}\Big)$.

The Z_1, Z_2 update can then be written as:

$$\begin{bmatrix} Z_{t-1,1}^{k+1} \\ Z_{t,2}^{k+1} \end{bmatrix} = \arg\min_{Z_{t-1,1}, Z_{t,2}} \beta\psi(Z_{t-1,1} - Z_{t,2}) + \frac{\rho}{2}\|[\Theta_{t-1}^{k+1} + U_{t-1,1}^k] - Z_{t-1,1}\|_F^2 + \frac{\rho}{2}\|[\Theta_t^{k+1} + U_{t,2}^k] - Z_{t,2}\|_F^2.$$

$$= \arg\min_{Z_{t-1,1}, Z_{t,2}} \beta\tilde{\psi}\Big(\begin{bmatrix} Z_{t-1,1} \\ Z_{t,2} \end{bmatrix}\Big) + \frac{\rho}{2}\Big\|\begin{bmatrix} Z_{t-1,1} \\ Z_{t,2} \end{bmatrix} - \begin{bmatrix} \Theta_{t-1}^{k+1} + U_{t-1,1}^k \\ \Theta_t^{k+1} + U_{t,2}^k \end{bmatrix}\Big\|_F^2.$$

$$\begin{bmatrix} Z_{t-1,1}^{k+1} \\ Z_{t,2}^{k+1} \end{bmatrix} = \text{prox}_{\beta/\rho\tilde{\psi}}\Big(\begin{bmatrix} \Theta_{t-1}^{k+1} + U_{t-1,1}^k \\ \Theta_t^{k+1} + U_{t,2}^k \end{bmatrix}\Big).$$

Efficient solutions to TVGL thus rely on finding efficient, preferably closed-form, solutions to the proximal operator of the $\tilde{\psi}$ function. Provided that these operators can be evaluated in faster than $O(p^3)$ time, then the time complexity of the problem is not increased. The authors of [above] give closed-form solutions for each of the ψ penalties introduced in Section 2.2 in their paper.

4.4.3 U Update

The update for the scaled dual variable is the matrix extension of the update introduced in 3.1.

$$U^{k+1} = \begin{bmatrix} U_0^{k+1} \\ U_1^{k+1} \\ U_2^{k+1} \end{bmatrix} = \begin{bmatrix} U_0^k \\ U_1^k \\ U_2^k \end{bmatrix} + \begin{bmatrix} \Theta_1^{k+1}, \ldots, \Theta_{T-1}^{k+1} \\ \Theta_2^{k+1}, \ldots, \Theta_T^{k+1} \end{bmatrix} - \begin{bmatrix} Z_1^{k+1}, \ldots, Z_{T-1}^{k+1} \\ Z_2^{k+1}, \ldots, Z_T^{k+1} \end{bmatrix}.$$

5 Implementation

In this section, we present the MATLAB implementation of the Time-Varying Graphical Lasso (TVGL) algorithm. The code is divided into key steps, focusing on initialization, ADMM updates, and convergence checks.

5.1 MATLAB Code

5.1.1 Initialization

The initialization step computes empirical covariance matrices and sets initial values for variables Θ, Z, and U.

```
% Dimensions
[numSamples, numFeatures] = size(data);
numTimeSlices = floor(numSamples / numFeatures);

% Precompute empirical covariance matrices
empCovSet = cell(1, numTimeSlices);
for t = 1:numTimeSlices
    empCovSet{t} = cov(data((t-1)*numFeatures+1:t*numFeatures, :));
end

% Initialize variables
thetaSet = cell(1, numTimeSlices); % Precision matrices
Z = cell(3, numTimeSlices); % Auxiliary variables Z_0, Z_1, Z_2
U = cell(3, numTimeSlices); % Dual variables U_0, U_1, U_2
for t = 1:numTimeSlices
    thetaSet{t} = eye(numFeatures); % Start with identity
    Z{1, t} = zeros(numFeatures);
    Z{2, t} = zeros(numFeatures);
    Z{3, t} = zeros(numFeatures);
    U{1, t} = zeros(numFeatures);
    U{2, t} = zeros(numFeatures);
    U{3, t} = zeros(numFeatures);
end
```

Listing 1: Initialization of TVGL

5.1.2 Θ-Update

The Θ-update step computes the precision matrices by solving a convex optimization problem.

```
% ----------------- (1) Theta-Update --------------- %
for t = 1:numTimeSlices
    A = (Z{1, t} + Z{2, t} + Z{3, t} - U{1, t} - U{2, t} - U{3, t}) /
        3;
    A = 0.5 * (A + A')'; % Symmetrize
    eta = numFeatures / (3 * rho);
    S = empCovSet{t};

    % Eigendecomposition and analytical solution
    [Q, D] = eig(eta^(-1) * A - S); % Eigendecomposition
    D = diag(D);
    D_new = (D + sqrt(D.^2 + 4 * eta^(-1))) / (2 * eta^(-1));
    thetaSet{t} = Q * diag(D_new) * Q'; % Reconstruct Theta
end
```

Listing 2: Theta-Update Step

5.1.3 Z_0-Update

The Z_0-update step involves element-wise soft-thresholding for sparsity.

```
% ----------------- (2-1) Z0-Update ----------------- %
for t = 1:numTimeSlices
    A = thetaSet{t} + U{1, t};
    lambda_rho = lambda / rho; % Soft-threshold parameter

    % Element-wise soft-thresholding
    for i = 1:numFeatures
        for j = 1:numFeatures
            if i ~= j
                Z{1, t}(i, j) = sign(A(i, j)) * max(abs(A(i, j)) -
                    lambda_rho, 0);
            else
                Z{1, t}(i, j) = A(i, j); % Keep diagonal elements
                    unchanged
            end
        end
    end
end
```

Listing 3: Z0-Update Step

5.1.4 (Z_1, Z_2)-Update

This step handles the update of temporal consistency variables based on the penalty type.

```
% -------------- (2-2) (Z_1, Z_2)-Update ------------- %
for t = 2:numTimeSlices
    A = (thetaSet{t-1} - thetaSet{t} + U{2, t-1} - U{3, t});
    eta = 2 * beta / rho;

    % Compute E based on penalty type
    switch penaltyType
        case 'L1'
            E = sign(A) .* max(abs(A) - eta, 0);
        case 'L2'
            normA = sqrt(sum(A.^2, 2));
            scaling = max(0, 1 - eta ./ normA);
            E = bsxfun(@times, A, scaling);
        case 'Laplacian'
            E = (1 / (1 + 2 * eta)) * A;
        case 'Linf'
            normA = sum(abs(A), 2);
            E = zeros(size(A));
            for j = 1:size(A, 1)
                if normA(j) <= eta
                    E(j, :) = 0;
                else
                    sigma = max(abs(A(j, :))) / eta / 2;
                    E(j, :) = A(j, :) - eta * sigma * sign(A(j, :));
                end
            end
        otherwise
            error('Unsupported penalty type.');
    end


```

```
    % Update Z_1 and Z_2
    Z{2, t-1} = 0.5 * (thetaSet{t-1} + thetaSet{t} + U{2, t-1} + U{3,
        t}) - 0.5 * E;
    Z{3, t} = 0.5 * (thetaSet{t-1} + thetaSet{t} + U{2, t-1} + U{3,
        t}) + 0.5 * E;
end
```

Listing 4: (Z1, Z2)-Update Step

5.1.5 Dual Variable (U)-Update

The dual variable U-update ensures convergence of the ADMM algorithm.

```
% -------------- Update the U variables ------------- %
for t = 1:numTimeSlices
    U{1, t} = U{1, t} + (thetaSet{t} - Z{1, t});
    if t < numTimeSlices
        U{2, t} = U{2, t} + (thetaSet{t} - Z{2, t});
        U{3, t+1} = U{3, t+1} + (thetaSet{t+1} - Z{3, t+1});
    end
end
```

Listing 5: Dual Variable U-Update

5.1.6 Convergence Check

Finally, we check for convergence based on primal and dual residuals.

```
% -------------- Convergence check ------------------ %
[primalRes, dualRes] = ComputeResiduals(thetaSet, Z, U);
if primalRes < tol && dualRes < tol
    fprintf('Outer ADMM converged at iteration %d\n', iter);
    break;
end
```

Listing 6: Convergence Check

—

6 Results: Application to Portfolio Management

The Time-Varying Graphical Lasso (TVGL) algorithm was applied to real-world financial data to estimate dynamic precision matrices (Θ_t) across multiple time slices. These matrices represent evolving dependencies among assets and were used to construct minimum variance portfolios.

6.1 Data Description

To demonstrate the varying correlation structures, we collected **daily closing prices** for a range of assets across multiple sectors using the `yfinance` package.
The selection includes :

- **Big Tech:** AAPL, MSFT, GOOGL, AMZN, META
- **Energy:** XOM, CVX, BP, SLB, HAL

- **Healthcare:** JNJ, PFE, MRK, UNH, LLY
- **Consumer Goods:** KO, PEP, PG, UL, CL
- **Aerospace & Defense:** BA, RTX, LMT, NOC, GD
- **Automotive:** TSLA, GM, F, NIO, RIVN
- **Cryptocurrencies:** BTC-USD, ETH-USD, DOGE-USD
- **Precious Metals & ETFs:** GOLD, SLV, PPLT, PALL, GLD
- **Indices:** SPY, QQQ, DIA, IWM, VTI
- **Chinese Tech:** BABA, TCEHY, NIO, JD, PDD

By combining stocks from distinctly different industries (and adding cryptocurrencies plus metals), we obtain a dataset with a wide range of correlation patterns.

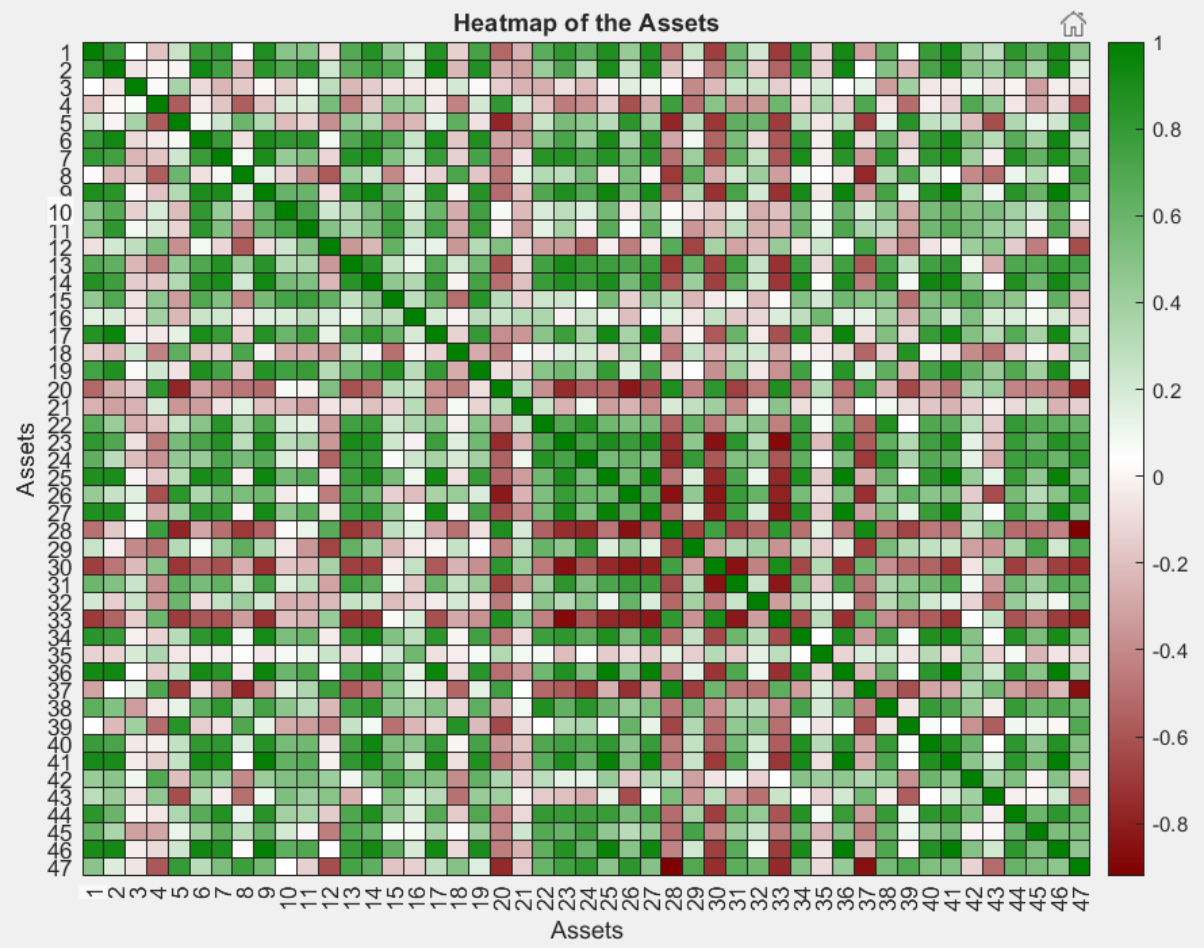

Figure 1: Heatmap of Pairwise Correlations among Selected Assets

In Figure 1, the correlation heatmap illustrates how assets within similar sectors (e.g., Big Tech or Automotive) tend to show stronger positive correlations (Green colors), while distinct sectors (or asset classes like crypto or precious metals) often exhibit weaker or even negative correlations (Red colors).

6.2 Minimum Variance Portfolio

For each time slice t, the portfolio weights were computed using the precision matrix Θ_t as follows:

$$w_t = \frac{\Theta_t \mathbf{1}}{\mathbf{1}^\top \Theta_t \mathbf{1}}$$

where:

- w_t: Portfolio weights at time slice t,
- Θ_t: Precision matrix (inverse covariance matrix) at time slice t,
- $\mathbf{1}$: Vector of ones ensuring portfolio weights sum to one.

This ensures a minimum variance portfolio by leveraging the structure of the estimated precision matrix.

6.3 Graphical Representation

The graph below represents the dependencies between assets for a specific time slice, as captured by the precision matrix Θ_t. Nodes represent assets, and edges indicate significant dependencies.

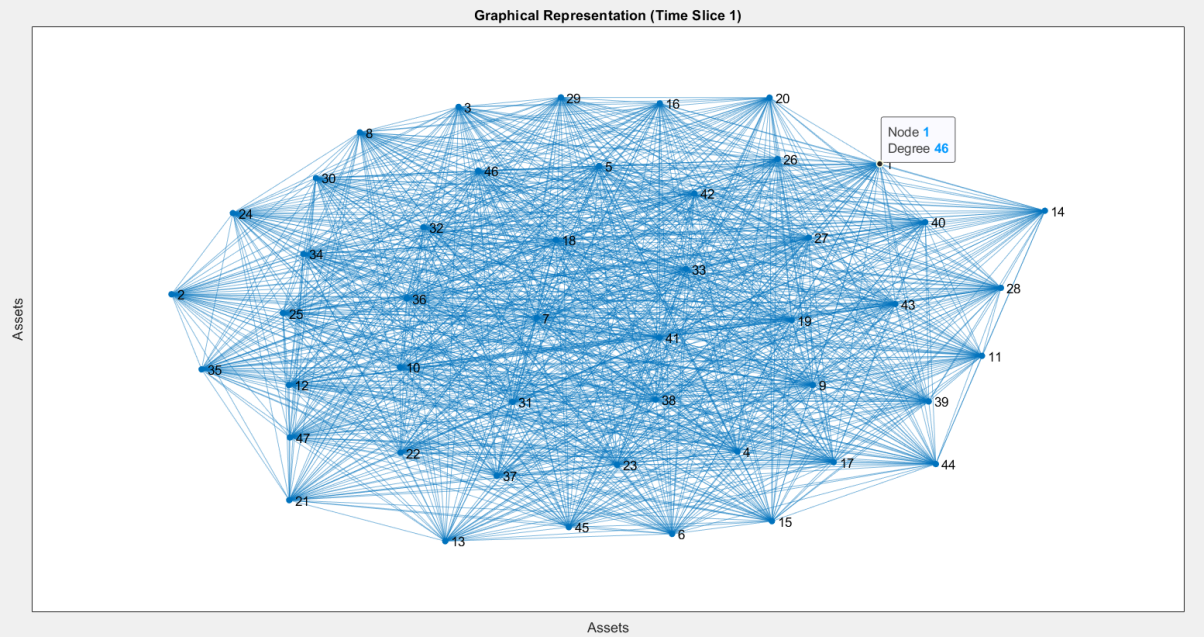

Figure 2: Graphical Representation of Asset Dependencies at the begining

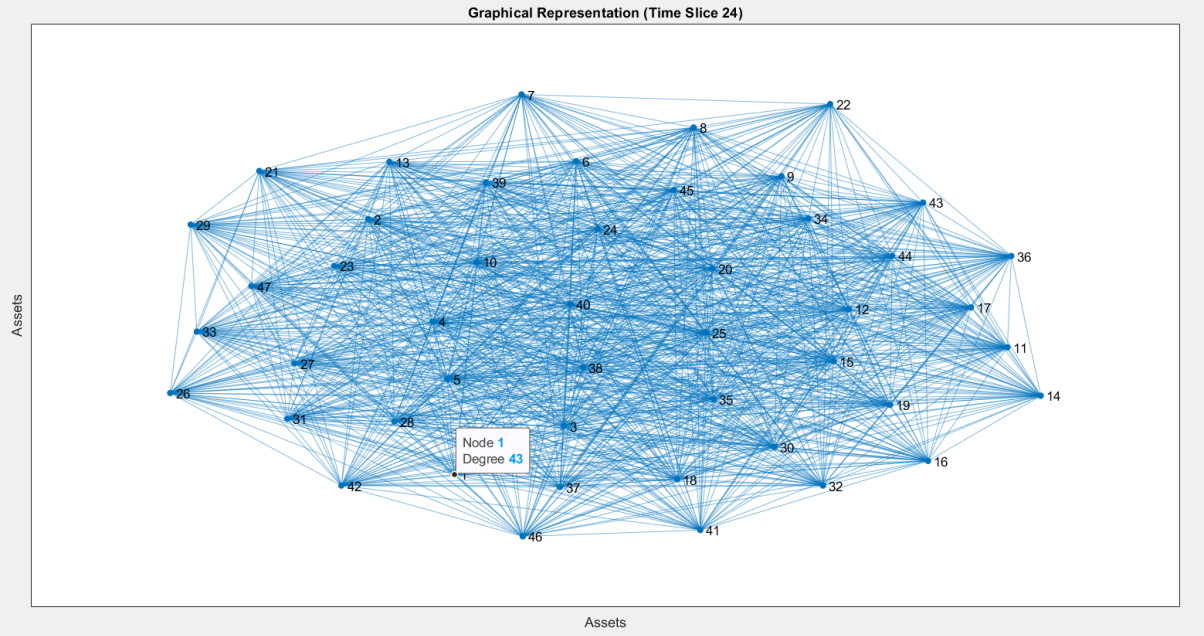

Figure 3: Graphical Representation of Asset Dependencies at the end

6.4 Portfolio Weight Evolution

The evolution of portfolio weights over time reflects the dynamic changes in asset dependencies captured by the TVGL algorithm. The plot below illustrates how the weights of individual assets in the portfolio change across different time slices.

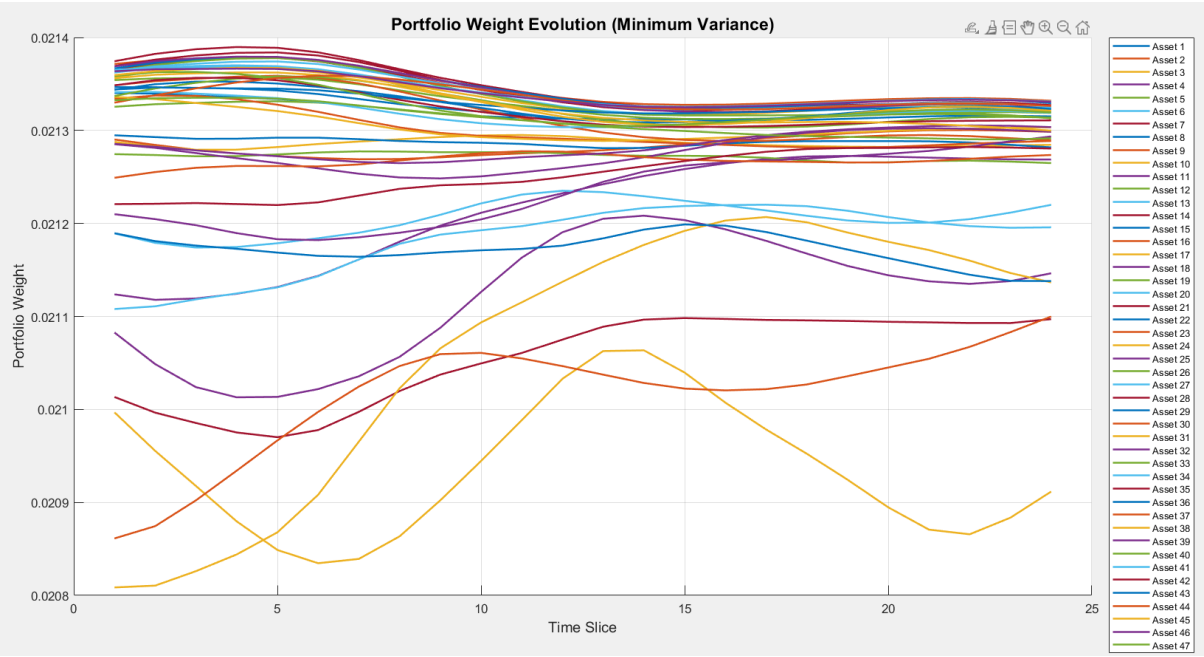

Figure 4: Portfolio Weight Evolution Over Time (Minimum Variance Portfolio)

6.5 Summary of Results

The TVGL algorithm successfully captured evolving dependencies among financial assets, enabling the construction of dynamic minimum variance portfolios. The graphical representation and portfolio weight evolution demonstrate the practical application of the algorithm in portfolio optimization.

7 Potential Extensions and Improvements

While the current implementation of the Time-Varying Graphical Lasso (TVGL) provides a robust framework for modeling dynamic dependencies, there are several avenues for potential extensions and improvements:

- **Incorporation of Non-Gaussian Data:** Extend TVGL to handle non-Gaussian distributions using copula-based methods or semiparametric approaches, enabling broader applicability to financial and biological datasets.
- **Adaptive Regularization:** Implement adaptive tuning of the parameters λ and β to dynamically adjust sparsity and temporal smoothness, improving flexibility in varying market conditions.
- **Enhanced Penalty Functions:** Explore alternative penalty functions, such as elastic-net-based penalties, to balance sparsity and temporal consistency more effectively.
- **Portfolio Optimization with Constraints:** Integrate additional practical constraints, such as sector diversification or transaction costs, into the portfolio optimization framework.

References

[1] D. Hallac, Y. Park, S. Boyd, and J. Leskovec. Network inference via the time-varying graphical lasso. Stanford University. Technical Paper.

[2] D. Hallac, Y. Park, S. Boyd, and J. Leskovec. Network inference via the time-varying graphical lasso. Stanford University, ESE605, Convex Optimization, Spring 2019, Project Report, 2019.

[3] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction.* Springer, 2 edition, 2017.

[4] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective.* MIT Press, 2012.