Understanding high-dimensional networks for continuous variables using ECL

Kshitij Khare and Syed Rahman

Department of Statistics
University of Florida

Motivation

- Availability of high-dimensional data from various applications
- Number of variables (p) much larger than (or sometimes comparable to) the sample size (n)
- Examples:
  - Biology: gene expression data
  - Environmental science: climate data on spatial grid
  - Finance: returns on thousands of stocks

Goal: Understanding relationships between variables

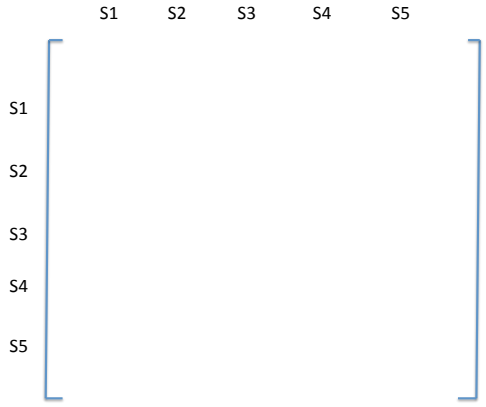
- Common goal in many applications: Understand complex network of relationships between variables
- Covariance matrix: a fundamental quantity to help understand multivariate relationships
- Even if estimating the covariance matrix is not the end goal, it is a crucial first step before further analysis

Quick recap: What is a covariance matrix?

- The covariance of two variables/features (say two stock prices) is a measure of linear dependence between these variables

- Positive covariance indicates similar behavior, Negative covariance indicates opposite behavior, zero covariance indicates lack of linear dependence

Lets say we have five stock prices S1, S2, S3, S4, S5. The covariance matrix of these five stocks looks like



Challenges in high-dimensional estimation

- If $p = 1000$, we need to estimate roughly 1 million covariance parameters
- If sample size n is much smaller (or even same order) than p, this is not viable
- The sample covariance matrix (classical estimator) can perform very poorly in high-dimensional situations (not even invertible when $n < p$)

Is there a way out?

- **Reliably estimate small number of parameters** in the covariance matrix or an appropriate function of the covariance matrix
- Set insignificant parameters to zero
- Sparsity pattern (pattern of 0s) can be represented by graphs/networks

Directed acyclic graph models: Sparsity in the Cholesky Parameter

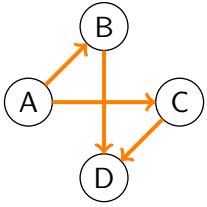
- Set entries of L to be zero: corresponds to assuming certain conditional independences
- Sparsity pattern in L can be represented by an directed graph
- Build a graph from sparse L

- Consider Cholesky decompostion of $\Sigma^{-1} = L^t L$
- $$\underbrace{\begin{pmatrix} 4.29 & 0.65 & 0.76 & 0.80 \\ 0.65 & 4.25 & 0.76 & 0.80 \\ 0.76 & 0.76 & 4.16 & 0.8 \\ 0.80 & 0.80 & 0.80 & 4.0 \end{pmatrix}}_{\Sigma^{-1}} = \underbrace{\begin{pmatrix} 2.0 & 0.2 & 0.3 & 0.4 \\ 0 & 2.0 & 0.3 & 0.4 \\ 0 & 0 & 2.0 & 0.4 \\ 0 & 0 & 0 & 2.0 \end{pmatrix}}_{L^t} \underbrace{\begin{pmatrix} 2.0 & 0 & 0 & 0 \\ 0.2 & 2.0 & 0 & 0 \\ 0.3 & 0.3 & 2.0 & 0 \\ 0.4 & 0.4 & 0.4 & 2 \end{pmatrix}}_{L}$$
- Entires of L have a **concrete** and **direct** interpretation in terms of appropriate conditional covariances

Directed acyclic graph models: Sparsity in the Cholesky Parameter

- Consider Cholesky decompostion of $\Sigma^{-1} = L^t L$
- Set entries of L to be zero
- Sparsity pattern in L can be represented by a directed graph

$$L = \begin{array}{c} \begin{array}{cccc} \text{A} & \text{B} & \text{C} & \text{D} \end{array} \\ \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0.2 & 2 & 0 & 0 \\ 0.3 & 0 & 2 & 0 \\ 0 & 0.4 & 0.4 & 2 \end{pmatrix} \end{array} \begin{array}{c} \text{A} \\ \text{B} \\ \text{C} \\ \text{D} \end{array}$$



STATISTICAL CHALLENGE

How do we estimate a covariance matrix with a sparse Cholesky factor based on data?

Convex Sparse Cholesky Selection(CSCS)

- **Obtain a sparse estimate for L by minimizing the objective function:**

$$Q_{CSCS}(L) = \underbrace{tr(L^t LS) - 2log|L|}_{\text{log-likelihood}} + \underbrace{\lambda \sum_{1 \leqslant j < i \leqslant p} |L_{ij}|}_{\substack{\text{penalty term to} \\ \text{induce sparsity/zeros}}}.$$

- λ (chosen by the user) controls the level of sparsity in the estimator
- Larger the λ, sparser the estimator

CSCS method: Comparison with other methods

	METHOD		
Property	Sparse Cholesky	Sparse DAG	CSCS
No constraints on sparsity pattern	+	+	+
No constraints on D	+		+
Convergence guarantee to acceptable global minimum		+	+
Asymptotic consistency ($n, p \to \infty$)		+	+

CSCS outperfroms and improves on existing methods!

Breaking up the objective function row-wise

- $Q_{CSCS}(L)$ breaks up as a sum of independent functions of the rows of L
- If $F(x, y, z) = F_1(x) + F_2(y) + F_3(z)$, then to minimize $F(x, y, z)$, we can minimize $F_1(x)$ with respect to x, $F_2(y)$ with respect to y and $F_3(z)$ with respect to z
-

$$Q_{CSCS}(L) = \underbrace{Q_1(L_{1.})}_{\substack{\text{Function of entries of} \\ 1^{st} \text{ row of L}}} + \ldots + \underbrace{Q_p(L_{p.})}_{\substack{\text{Function of entries of} \\ p^{th} \text{ row of L}}}$$

Call center data

- The data come from one call centre in a major U.S. northeastern financial organisation.
- For each day, a 17-hour period was divided into p = 102 10-minute intervals, and the number of calls arriving at the service queue during each interval was counted
- $n = 239$ days using the singular value decomposition to screen out outliers that include holidays and days when the recording equipment was faulty
- Hence $\binom{102}{2} = 10506$ parameters need to be estimated

GOAL

Build a predictor to forecast the calls coming in during the second half of the day based on the number of calls during the first half of the day

Call Center data

- The best mean squared error forecast of $y_i^{(2)}$, calls coming in during the second half of day t, using $y_i^{(1)}$, calls coming in during the first half of the day is

$$\hat{y}_{it}^{(2)} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(y_{it}^{(1)} - \mu_1)$$

  where $\mu_2, \mu_1, \Sigma_{21}$ and Σ_{11} must be estimated.

- To evaluate the performance of the sample covariance versus CSCS, we use Average Absolute Forecast Error, defined as,

$$AE_t = \frac{1}{34}\sum_{i=206}^{239}\left|\hat{y}_{it}^{(2)} - y_{it}^{(2)}\right|$$

- For CSCS, λ is chosen using 5- fold cross-validation

Call center data

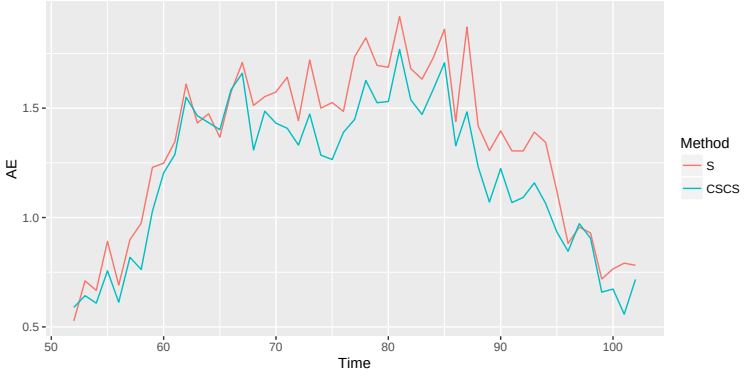


Figure : Average Absolute Forecast error using cross-validation

CSCS outperforms the sample covariance matrix 46 out of 51 times!

Call center data

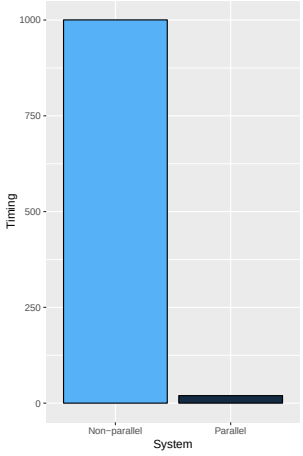


Figure : Timing Comparison between Parallel and Non-parallel Versions of CSCS

Algorithm1

Algorithm 1 Cyclic coordinatewise algorithm for $h_{k,A,\lambda}$

Input: Fix k, A, λ
Input: Fix maximum number of iterations: r_{max}
Input: Fix initial estimate: $\hat{x}^{(0)}$
Input: Fix convergence threshold: ϵ
Set $r \leftarrow 1$
converged = FALSE
Set $\hat{x}^{current} \leftarrow \hat{x}^{(0)}$
repeat
  $\hat{x}^{old} \leftarrow \hat{x}^{current}$)
  for $j \leftarrow 1, 2, ..., k-1$ **do** $\hat{x}_j^{current} \leftarrow T_j(j, \lambda, A, \hat{x}^{old})$
  end for
  $\hat{x}_k^{current} \leftarrow T_k(\lambda, A, \hat{x}^{old})$
  $\hat{x}^r \leftarrow \hat{x}^{current}$
  ▷ Convergence Checking
  if $||\hat{x}^{current} - \hat{x}^{old}|| < \epsilon$ **then**
    converged = TRUE
  else
    $r \leftarrow r + 1$
  end if
until converged = TRUE or $r > r_{max}$
Return final estimate: $\hat{x}^r$

Algorithm1 in ECL

```
CSCS_h1(DATASET(xElement) xx0, DATASET(DistElem) A, UNSIGNED k,
      REAL lambda, UNSIGNED maxIter = 100, REAL tol = 0.00001):=
      FUNCTION
      out := LOOP(xx0,(COUNTER<=maxIter AND
                MaxAbsDff(ROWS(LEFT))>tol),
                OuterBody(ROWS(LEFT), A, COUNTER, k, lambda));
      RETURN SORT(PROJECT(out(typ=xType.x), DistElem),x,y);
END;
```

CSCS algorithm

Algorithm 2 CSCS Algorithm

Input: Data $Y_1, Y_2, ..., Y_n$ and λ
Input: Fix maximum number of iterations: r_{max}
Input: Fix initial estimate: $\hat{L}^{(0)}$
Input: Fix convergence threshold: ϵ

▷ Can be done in parallel

for $i \leftarrow 1, 2, \ldots, p$ **do**
$(\hat{\eta}^i)^{(0)} \leftarrow i^{th}$ row of $\hat{L}^{(0)}$
Set $\hat{\eta}^i$ to be minimizer of objective function $Q_{CSCS,i}$ obtained by using Algorithm 1 with $k = 1, A = S_i, \lambda, r_{max}, \hat{x}^{(0)} = (\hat{\eta}^i)^{(0)}, \epsilon$
end for
Construct $\hat{L}$ by setting its i^{th} row (up to the diagonal) as $\hat{\eta}^i$
Return final estimate: $\hat{L}$

CSCS in ECL

```
CSCS2(DATASET(Elem) YY, REAL lambda3, REAL tol2=0.00001,
        UNSIGNED maxIter2=100) := FUNCTION
        nobs := MAX(YY,x);
        pvars := MAX(YY,y);
        S:= Mat.Scale(Mat.Mul(Mat.Trans(YY),YY),(1/nobs));
        S1 := DISTRIBUTE(NORMALIZE(S, (pvars DIV 2),
                TRANSFORM(DistElem, SELF.nid:=COUNTER, SELF:=LEFT)), nid);
        L:= Identity(pvars);
        LL:= DISTRIBUTE(L,nid);
        L11 := PROJECT(CHOOSEN(S1(x=1 AND y=1 AND nid=1),1),
                  TRANSFORM(DistElem, SELF.x := 1, SELF.y := 1,
                  SELF.value:=1/LEFT.value, SELF.nid := 1, SELF:=[]));
        newL := LL(x <> 1) + L11;
        newLL := LOOP(newL,COUNTER<pvars,OuterOuterBody(ROWS(LEFT), S1,
                       COUNTER, lambda3, maxIter2, tol2));
       RETURN newLL;
END;
```