Matrix Inversion

Group 23

UM-SJTU Joint Institute

October 17, 2018

Matrix Inversion

Definition

The **inverse** of a $n \times n$ matrix $\mathbf{A}$, denoted as $\mathbf{A}^{-1}$, is a matrix that satisfies the expression

$$\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I_n}$$

Where $\mathbf{I_n}$ is a $n \times n$ matrix with 1 on its diagonal and 0 otherwise.

Naive Matrix Inversion Method

The naive method to solve matrix inversion is to treat the expression $\mathbf{AA^{-1}} = \mathbf{I_n}$ as a linear equation system whose unkonwns are $1 \times n$ vectors (rows of $\mathbf{A^{-1}}$)

Solving this equation is of $\mathcal{O}(n^4)$ complexity (Gaussian Elimination)

Two Matrix Inversion Algorithms

- Cholesky Decomposition
  - Complexity: $\mathcal{O}(n^3)$
  - Application: Positive Definite Hermitian matrices
- Levinson-Durbin Recursion
  - Complexity: $\mathcal{O}(n^3)$
  - Application: Symmetric Toeplitz matrices

Cholesky Decomposition

A positive definite symmetric matrix $\mathbf{A}$ can be represented as product of two transpose triangular matrix as the following

$$\mathbf{A} = \mathbf{U}^{\mathbf{T}}\mathbf{U}$$

where $\mathbf{U}$ is an $n \times n$ upper triangular matrix.
Note: The implementation of finding $\mathbf{U}$ for a certain $\mathbf{A}$ can be in-place, so that no extra space is needed.

Based on the Cholesky decomposition, the euqation can be transformed as

$$\mathbf{U}^{\mathbf{T}}\mathbf{U}\mathbf{A}^{-1} = \mathbf{I_n} \Rightarrow \begin{cases} \mathbf{U}^{\mathbf{T}}\mathbf{B} & = \mathbf{I_n} \\ \mathbf{U}\mathbf{A}^{-1} & = \mathbf{B} \end{cases}$$

and solving these two equations is simple because back/forward substitution can be directly applied.
What's more, the symmetric property of **A** will also cause a reduction in the calculation.

Levinson-Durbin Recursion

The first step of Levinson Durbin is to partition the matrix into blocks

$$\mathbf{A} = \begin{bmatrix} \mathbf{B}_{(\mathbf{n}-\mathbf{1})\times(\mathbf{n}-\mathbf{1})} & \mathbf{C}_{(\mathbf{n}-\mathbf{1})\times\mathbf{1}} \\ \mathbf{D}_{\mathbf{1}\times(\mathbf{n}-\mathbf{1})} & \mathbf{E}_{\mathbf{1}\times\mathbf{1}} \end{bmatrix}$$

The algorithm will first find a inversion of $\mathbf{B}$, denoted as $\mathbf{B}^{-1}$ using recursive call and also find a **backward vector** of $\mathbf{A}$, denoted as $\mathbf{u}$ which satisfies

$$\mathbf{Au} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}$$

Append a row of zeros to the bottom of $\mathbf{B}^{-1}$, and it can be seen that

$$\begin{bmatrix} \mathbf{B}_{(n-1)\times(n-1)} & \mathbf{C}_{(n-1)\times 1} \\ \mathbf{D}_{1\times(n-1)} & \mathbf{E}_{1\times 1} \end{bmatrix} \begin{bmatrix} \mathbf{B}^{-1}_{(n-1)\times(n-1)} \\ \mathbf{0}_{1\times(n-1)} \end{bmatrix} = \begin{bmatrix} \mathbf{I}_{(n-1)} \\ \mathbf{error}_{1\times(n-1)} \end{bmatrix}$$

Then what should we do is add scaled $\mathbf{u}$ on the columns of the new matrix to eliminate the errors. At last append $\mathbf{u}$ to the right, then $\mathbf{A}^{-1}$ is calculated.

Python specifics included to improve coding efficiency

The basic object used is **numpy.mat** to store the matrices. There are several reasons

- It is more convenient for storing and generating test matrices using numpy commands
- Many loops and case statement in the implementation can be substituted by matrices operations (row operation, matrix partition, matrix multiplicaiton, etc.), and numpy.mat has adequette implementations of these operations.

References

- Aravindh Krishnamoorthy, Deepak Menon (2013), "Matrix Inversion Using Cholesky Decomposition"
- Trench, W. F. (1964). "An algorithm for the inversion of finite Toeplitz matrices." J. Soc. Indust. Appl. Math., v. 12, pp. 515522.
- Musicus, B. R. (1988). "Levinson and Fast Choleski Algorithms for Toeplitz and Almost Toeplitz Matrices." RLE TR No. 538, MIT