Computational Linear Algebra: Ax=b (Part III: Cholesky Factorization)

David Shuman

February 17, 2022

Review of Eigenvalues and Diagonalizability

Recall an eigenvalue of an $n \times n$ matrix A is a scalar $\lambda \in \mathbb{R}$ such that there exists a nonzero vector $\mathrm{v} \in \mathbb{R}^n$ with the property that $A\mathrm{v} = \lambda\mathrm{v}$. The vector v is an eigenvector with eigenvalue λ

Review of Eigenvalues and Diagonalizability

Recall an eigenvalue of an $n \times n$ matrix A is a scalar $\lambda \in \mathbb{R}$ such that there exists a nonzero vector $\mathrm{v} \in \mathbb{R}^n$ with the property that $A\mathrm{v} = \lambda\mathrm{v}$. The vector v is an eigenvector with eigenvalue λ

A is diagonalizable if $A = PDP^{-1}$ for some diagonal matrix D (whose diagonal elements are the eigenvalues of A) and invertible matrix P (the columns of which are eigenvectors of A)

Review of Eigenvalues and Diagonalizability

Recall an eigenvalue of an $n \times n$ matrix A is a scalar $\lambda \in \mathbb{R}$ such that there exists a nonzero vector $\mathrm{v} \in \mathbb{R}^n$ with the property that $A\mathrm{v} = \lambda \mathrm{v}$. The vector v is an eigenvector with eigenvalue λ

A is diagonalizable if $A = PDP^{-1}$ for some diagonal matrix D (whose diagonal elements are the eigenvalues of A) and invertible matrix P (the columns of which are eigenvectors of A)

Q: When is A diagonalizable?

Review of Eigenvalues and Diagonalizability

Recall an eigenvalue of an $n \times n$ matrix A is a scalar $\lambda \in \mathbb{R}$ such that there exists a nonzero vector $\mathrm{v} \in \mathbb{R}^n$ with the property that $A\mathrm{v} = \lambda\mathrm{v}$. The vector v is an eigenvector with eigenvalue λ

A is diagonalizable if $A = PDP^{-1}$ for some diagonal matrix D (whose diagonal elements are the eigenvalues of A) and invertible matrix P (the columns of which are eigenvectors of A)

Q: When is A diagonalizable?

A: When it has n linearly independent eigenvectors (i.e., columns of P form an eigenbasis)

Eigenspaces of Symmetric Matrices ($A^\top = A$)

Definition

An $n \times n$ matrix A is orthogonally diagonalizable if there is a diagonal matrix D and an orthogonal matrix P (i.e., $P^\top = P^{-1}$) such that

$$A = PDP^\top = PDP^{-1}$$

Eigenspaces of Symmetric Matrices ($A^\top = A$)

Definition

An $n \times n$ matrix A is orthogonally diagonalizable if there is a diagonal matrix D and an orthogonal matrix P (i.e., $P^\top = P^{-1}$) such that

$$A = PDP^\top = PDP^{-1}$$

Theorem (Spectral Theorem for Symmetric Matrices)

- *A matrix A is orthogonally diagonalizable if and only if it is symmetric*
- *A symmetric $n \times n$ matrix A has n real eigenvalues (counting multiplicities)*
- *The eigenspaces of a symmetric $n \times n$ matrix A are orthogonal*
- *The dimension of each eigenspace is equal to the multiplicity of its eigenvalue as a root of the characteristic equation*
- *Spectral decomposition:* $A = \lambda_1 \mathrm{v}_1 \mathrm{v}_1^\top + \lambda_2 \mathrm{v}_2 \mathrm{v}_2^\top + \ldots + \lambda_n \mathrm{v}_n \mathrm{v}_n^\top$
  - *Can view mapping A as a linear combination of orthogonal projections onto the different (orthogonal) eigenspaces, with the weights determined by the eigenvalues*

Symmetric Positive Definite Matrices

Note: If A is an $n \times n$ matrix and $\mathrm{x} \in \mathbb{R}^n$, then $\mathrm{x}^T A \mathrm{x}$ is a scalar

Definition

An $n \times n$ real matrix A is positive definite if $\mathrm{x}^T A \mathrm{x} > 0$ for all vectors $\mathrm{x} \in \mathbb{R}^n$ with $\mathrm{x} \neq 0$ (and positive semidefinite if $\mathrm{x}^T A \mathrm{x} \geq 0$)

Theorem

- *A symmetric matrix A is positive definite if and only if all of its eigenvalues are (strictly) positive*
- *Any principal submatrix of a symmetric positive-definite matrix is symmetric positive-definite*

Symmetric positive definite matrices show up *a lot* in applications

Symmetric Positive Definite Matrices

Example

- If $A_1 = \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix}$ then

$$\begin{pmatrix} x_1 & x_2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = 3x_1^2 + 4x_1x_2 + 6x_2^2 = x_1^2 + 2(x_1 + x_2)^2 + 4x_2^2$$

  so A_1 is positive definite

- Find the eigenvalues:
$\begin{pmatrix} 8 & 4 & 2 \\ 4 & 6 & 0 \\ 2 & 0 & 3 \end{pmatrix}$, $\begin{pmatrix} 3 & 2 & 0 \\ 2 & 3 & 3 \\ 0 & 2 & 3 \end{pmatrix}$, $\begin{pmatrix} 3 & -1 & 0 & 0 & 0 & .5 \\ -1 & 3 & -1 & 0 & .5 & 0 \\ 0 & -1 & 3 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & 0 \\ 0 & .5 & 0 & -1 & 3 & -1 \\ .5 & 0 & 0 & 0 & -1 & 3 \end{pmatrix}$

- Symmetric but not positive definite:
$\begin{pmatrix} 1 & -1 \end{pmatrix} \begin{pmatrix} 1 & 2 \\ 2 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \end{pmatrix} = -2 < 0$

Cholesky Factorization

- Assume that A is symmetric positive definite
- LU factorization does not take full advantage of the structure of A
- LU factorization also does not always return L and U matrices that are symmetric (or even symmetric on the off-diagonal elements)
  - Example:

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 4 & -2 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = LU$$

Cholesky Factorization

- Assume that A is symmetric positive definite
- LU factorization does not take full advantage of the structure of A
- LU factorization also does not always return L and U matrices that are symmetric (or even symmetric on the off-diagonal elements)
  - Example:

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 4 & -2 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = LU$$

- Enter the Cholesky factorization: $A = R^T R$, where R is upper-triangular

Cholesky Factorization

- Assume that A is symmetric positive definite
- LU factorization does not take full advantage of the structure of A
- LU factorization also does not always return L and U matrices that are symmetric (or even symmetric on the off-diagonal elements)
  - Example:

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 4 & -2 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = LU$$

- Enter the Cholesky factorization: $A = R^T R$, where R is upper-triangular
- Leverages the structure of A to require half as many operations as LU

Cholesky Factorization

- Assume that A is symmetric positive definite
- LU factorization does not take full advantage of the structure of A
- LU factorization also does not always return L and U matrices that are symmetric (or even symmetric on the off-diagonal elements)
  - Example:

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 4 & -2 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = LU$$

- Enter the Cholesky factorization: $A = R^T R$, where R is upper-triangular
- Leverages the structure of A to require half as many operations as LU
- Every symmetric positive definite A has a unique $R^T R$ factorization

Cholesky Factorization

- Assume that A is symmetric positive definite
- LU factorization does not take full advantage of the structure of A
- LU factorization also does not always return L and U matrices that are symmetric (or even symmetric on the off-diagonal elements)
  - Example:

$$A = \begin{pmatrix} 4 & -2 & 0 \\ -2 & 2 & -3 \\ 0 & -3 & 10 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ -0.5 & 1 & 0 \\ 0 & -3 & 1 \end{pmatrix} \begin{pmatrix} 4 & -2 & 0 \\ 0 & 1 & -3 \\ 0 & 0 & 1 \end{pmatrix} = LU$$

- Enter the Cholesky factorization: $A = R^T R$, where R is upper-triangular
- Leverages the structure of A to require half as many operations as LU
- Every symmetric positive definite A has a unique $R^T R$ factorization
- Cholesky factorization provides a way to tell if a symmetric matrix is positive definite
- Example and general method on board

Cholesky Factorization: Pseudocode and Complexity

Pseudocode:

$$R = A$$

for $k = 1$ to n

for $j = k + 1$ to n

$$R_{j,j:n} = R_{j,j:n} - R_{k,j:n} * \frac{R_{k,j}}{R_{k,k}}$$

$$R_{k,k:n} = \frac{1}{\sqrt{R_{k,k}}} * R_{k,k:n}$$

Cholesky Factorization: Pseudocode and Complexity

Pseudocode:

$$
\begin{aligned}
&R = A \\
&\text{for } k = 1 \text{ to } n \\
&\quad \text{for } j = k+1 \text{ to } n \\
&\qquad R_{j,j:n} = R_{j,j:n} - R_{k,j:n} * \frac{R_{k,j}}{R_{k,k}} \\
&\quad R_{k,k:n} = \frac{1}{\sqrt{R_{k,k}}} * R_{k,k:n}
\end{aligned}
$$

Complexity:

$O(\frac{n^3}{3})$ additions and multiplications (half as many as LU)

Cholesky Factorization: Stability and Solving $Ax = b$

- Stability
  - Unlike Gaussian elimination, this algorithm is always stable in the sense that $R^T R$ is close to A
  - Intuitive reason: Norms of factors cannot grow too large (recall the examples we showed where L or U had very large matrix norms in comparison to the norm of A)
  - Example in 2-norm: $||R||_2 = ||R^T||_2 = \sqrt{||A||_2}$
  - No need for pivoting thanks to positive definite structure

Cholesky Factorization: Stability and Solving $Ax = b$

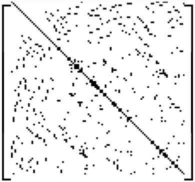
- Stability
  - Unlike Gaussian elimination, this algorithm is always stable in the sense that $R^T R$ is close to A
  - Intuitive reason: Norms of factors cannot grow too large (recall the examples we showed where L or U had very large matrix norms in comparison to the norm of A)
  - Example in 2-norm: $||R||_2 = ||R^T||_2 = \sqrt{||A||_2}$
  - No need for pivoting thanks to positive definite structure

- Solve $Ax = R^T Rx = b$ in two steps:
  - $R^T y = b$
  - $Rx = y$
  - Each of these two triangular systems is $O(n^2)$ [same as LU]

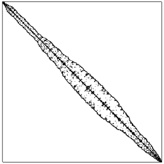
Sparse Matrices

- In *many* applications, the matrices we need are sparse: a large number of the entries are 0
  - Non-sparse matrices are called full or dense matrices
- Typically a sparse $n \times n$ matrix A has $\leq O(n)$ nonzero entries
- Here are a few examples (nonzero entries shown in black)

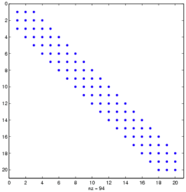
$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 3 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 4 & 5 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 5 & 8 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6 & 13 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 7 & 21 \end{bmatrix},$$



,

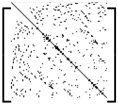
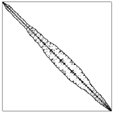
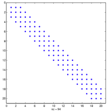


- Sometimes the nonzero entries have patterns, like in banded matrices:



$$\begin{pmatrix} A & I & O & \cdots & O & O \\ I & A & I & \cdots & O & O \\ O & I & A & \cdots & O & O \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ O & O & O & \cdots & I & A \end{pmatrix}$$

Sparse Matrices, cont.

 ,  ,  , $\begin{pmatrix} A & I & O & \cdots & O & O \\ I & A & I & \cdots & O & O \\ O & I & A & \cdots & O & O \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ O & O & O & \cdots & I & A \end{pmatrix}$

- The idea is to record the list of positions of nonzero entries (i_p, j_p) and the matrix entry a_p in that position (a triplet of information)

$$\begin{array}{rl} \textit{positions} & \begin{bmatrix} i_1 & i_2 & i_3 & i_4 & \cdots & i_n \\ j_1 & j_2 & j_3 & j_4 & \cdots & j_n \end{bmatrix} \\ \textit{entries} & \begin{bmatrix} a_1 & a_2 & a_3 & a_4 & \cdots & a_n \end{bmatrix} \end{array}$$

- i.e., do not store the 0's
- Then you need to get your matrix algorithms to work correctly with these new data structures
- Can be even more efficient if there is a strong pattern in the nonzero entries as in banded matrices
  - Just store the bands and the positions are understood

Sparse Matrices in R

- Need to include the Matrix package
- Try

```
A = Matrix(0,nrow=10,ncol=10,sparse=TRUE)
A[1,3] = A[5,6] = A[10,1] = 1
A[10,10] = A[2,10] = A[5,10] =  A[7,2] = 2
A
image(A)
```

- In triplet form:

```
A = spMatrix(10,20, i = c(1,3:8), j = c(2,9,6:10),  x = 7 * (1:7))
A
image(A)
```

- Try computing

```
3 * A
A %*% rep(0,20)
t(A)
A %*% t(A)
A + 1
```

Sparse Matrices in R

- Saving Space

```
m1 =  matrix(0, nrow = 1000, ncol = 1000)
m2 = Matrix(0, nrow = 1000, ncol = 1000, sparse = TRUE)
m3 = spMatrix(nrow = 1000, ncol = 1000)
object.size(m1)
object.size(m2)
object.size(m3)
```

- ```
  m1[500, 500] = 1
  m2[500, 500] = 1
  m3[500, 500] = 1
  object.size(m1)
  object.size(m2)
  object.size(m3)
  ```
  ```

Sparse Matrices in R

Saving Time

- Matrix multiplication for sparse matrices is optimized to take advantage of the fact that the matrix is sparse
- For example, the matrix-vector product Ax takes $O(n^2)$ multiplications if the matrix A is $n \times n$ and dense
- If A has only $O(n)$ nonzero entries then Ax can be optimized to take only $O(n)$ multiplications