

Pseudo-inverse

Gemini AI Pro 2.5

Q

Pseudo-inverse. What is it, and how is this different from the inverse of a matrix? Please explain how to compute it, both manually, by matrix operations and with Julia functions. What are the use cases for the Pseudo-inverse? Please also explain the symbol for it used in the image, and the background for this symbol.

A

Here is a detailed explanation of the pseudo-inverse, its computation, and its uses.

What is the Pseudo-inverse?

The **pseudo-inverse**, most commonly the **Moore-Penrose pseudo-inverse**, is a generalization of the standard matrix inverse. While a regular inverse (A^{-1}) only exists for square, non-singular (invertible) matrices, a pseudo-inverse (A^\dagger) exists for *any* $m \times n$ matrix, including rectangular or singular (non-invertible) matrices.

If a matrix A *is* invertible, its pseudo-inverse is identical to its regular inverse ($A^\dagger = A^{-1}$).

Pseudo-inverse vs. Regular Inverse

The key differences can be summarized in this table:

Feature	Regular Inverse (A^{-1})	Moore-Penrose Pseudo-inverse (A^\dagger)
Existence	Only exists for square ($n \times n$) matrices that are non-singular (i.e., determinant is non-zero).	Exists for any $m \times n$ matrix (square, rectangular, singular, or non-singular).

Feature	Regular Inverse (A^{-1})	Moore-Penrose Pseudo-inverse (A^\dagger)
Property	Satisfies $AA^{-1} = A^{-1}A = I$, where I is the identity matrix.	Satisfies a set of four criteria (the Moore-Penrose conditions). It acts as a “best fit” inverse.
Uniqueness	If it exists, it is unique.	If it exists (which it always does), it is unique.

How to Compute the Pseudo-inverse

There are several ways to compute the pseudo-inverse, depending on the properties of the matrix and the tools you are using.

1. By Matrix Operations (General Case via SVD) The most general and numerically stable method uses **Singular Value Decomposition (SVD)**. Any $m \times n$ matrix A can be decomposed as:

$$A = U\Sigma V^T$$

Where:

- U is an $m \times m$ orthogonal matrix.
- Σ is an $m \times n$ diagonal matrix of singular values.
- V^T is an $n \times n$ orthogonal matrix (transpose of V).

To find the pseudo-inverse A^\dagger , you compute:

$$A^\dagger = V\Sigma^\dagger U^T$$

Here, Σ^\dagger is the pseudo-inverse of Σ . It is found by taking the **reciprocal of all non-zero** singular values on the diagonal and then transposing the resulting matrix.

2. By Matrix Operations (Special Full-Rank Cases) Simpler formulas exist if the matrix A has **full rank** (all its columns or rows are linearly independent).

- **Full Column Rank ($m > n$, “tall” matrix):** This is the case for overdetermined systems, like typical least-squares problems. The pseudo-inverse is a **left inverse** ($A^\dagger A = I$). The formula is: $A^\dagger = (A^T A)^{-1} A^T$
- **Full Row Rank ($m < n$, “wide” matrix):** This is the case for underdetermined systems. The pseudo-inverse is a **right inverse** ($AA^\dagger = I$). The formula is: $A^\dagger = A^T (AA^T)^{-1}$

Connection to your image: The formula in your image, $A^\dagger = R^{-1}Q^T$, is a computationally efficient way to solve the **full column rank** case. It comes from the QR factorization ($A = QR$). If $A = QR$ and A has full column rank, then: $A^\dagger = (A^T A)^{-1} A^T = ((QR)^T (QR))^{-1} (QR)^T = (R^T Q^T Q R)^{-1} (R^T Q^T)$. Since Q is orthogonal, $Q^T Q = I$, so this simplifies to: $A^\dagger = (R^T R)^{-1} R^T Q^T = R^{-1} (R^T)^{-1} R^T Q^T = \mathbf{R}^{-1} \mathbf{Q}^T$. This is exactly the formula mentioned in your text, which is often used to solve least-squares problems as taught in courses like “VMLS” (likely “Vectors, Matrices, and Least Squares” by Boyd and Vandenberghe).

3. With Julia Functions As your image correctly states, Julia’s `LinearAlgebra` standard library provides a direct function:

```
using LinearAlgebra

# Example of a non-square matrix (3x2)
A = [1 2;
     3 4;
     5 6]

# Compute the pseudo-inverse
A_pinv = pinv(A)
```

The `pinv(A)` function handles all cases (full rank or not) and typically uses the robust SVD method internally.

Use Cases for the Pseudo-inverse

The primary use of the pseudo-inverse is to solve the linear system of equations $Ax = b$ when a unique solution does not exist.

1. Overdetermined Systems (Least Squares):

- **Problem:** You have more equations than unknowns (e.g., fitting a line to 100 data points). There is no exact solution x that satisfies all equations perfectly.
- **Solution:** $x = A^\dagger b$ provides the **least-squares solution**. This is the vector x that minimizes the error, specifically minimizing the squared Euclidean norm of the residual: $\|Ax - b\|^2$. This is the foundation of linear regression.

2. Underdetermined Systems (Minimum Norm):

- **Problem:** You have fewer equations than unknowns (e.g., $2x + 3y = 5$). There are infinitely many solutions.
- **Solution:** $x = A^\dagger b$ provides the **minimum-norm solution**. Of all the infinite possible solutions, this is the unique solution x that has

the smallest magnitude (smallest Euclidean norm $\|x\|^2$).

It is also used in signal processing, control systems, and image restoration.

The Symbol A^\dagger

- **What it is:** The symbol \dagger is called the “**dagger**” (or sometimes “obelisk”).
 - **Background:** The symbol is used to denote the **Moore-Penrose pseudo-inverse**, named after its originators, **E. H. Moore** (who introduced it around 1920) and **Roger Penrose** (who independently rediscovered and popularized it in 1955). Penrose defined it as the unique matrix that satisfies four specific algebraic properties.
 - **Potential Confusion:** In physics and some areas of mathematics, the \dagger symbol is also used to represent the **conjugate transpose** (or Hermitian conjugate) of a complex matrix. This is a *different operation* (though related). However, in the context of linear algebra and solving systems, A^\dagger almost universally refers to the Moore-Penrose pseudo-inverse.
-

This video discusses the pseudo-inverse in the context of numerical linear algebra and its relation to the SVD. The Pseudoinverse - Numerical Linear Algebra http://googleusercontent.com/youtube_content/0

Q

Short clarification: what is meant by singular value?

A

A **singular value** is a non-negative number that measures how much a matrix “stretches” or “squashes” space in a particular direction.

Here is a breakdown, from the most intuitive to the most formal.

1. The Geometric Meaning (The “Stretch” Factor)

Imagine you have a unit circle (in 2D) or a unit sphere (in 3D). If you apply a matrix A to every point on that circle/sphere, it will be transformed into an ellipse (or an ellipsoid).

The **singular values** (σ_1, σ_2 , etc.) are the **lengths of the semi-axes** (the half-lengths of the longest and shortest diameters) of that new ellipse.

- The **first singular value** (σ_1) is the largest one, representing the direction of *maximum* stretch.
- The **second singular value** (σ_2) represents the next-most stretch, in a direction perpendicular to the first.

- If a singular value is 0, it means the matrix “flattens” the shape entirely in that direction (i.e., it reduces the dimension).

2. The Formal Mathematical Definition

The singular values of any $m \times n$ matrix A are the **non-negative square roots of the eigenvalues** of the related square matrix $A^T A$ (where A^T is the transpose of A).

These values are the diagonal entries of the Σ matrix in the **Singular Value Decomposition (SVD)**, which you saw in the previous explanation: $A = U\Sigma V^T$

The number of non-zero singular values tells you the **rank** of the matrix, which is a measure of its “true” dimension.

3. Singular Values vs. Eigenvalues

This is a common point of confusion, but the distinction is very important.

Feature	Eigenvalues (λ)	Singular Values (σ)
Matrix Type	Only defined for square matrices.	Defined for any $m \times n$ matrix.
Geometric Meaning	“Scaling factors” for directions that do not change (invariant directions). $Av = \lambda v$.	“Stretching factors” for directions of maximum stretch . $Av = \sigma u$.
Value	Can be negative, zero, or complex.	Are always real and non-negative (0 or positive).

In short: * **Eigenvalues** tell you how a matrix scales its *eigenvectors* (vectors that don’t change direction). * **Singular values** tell you the magnitude of a matrix’s “action” (how much it stretches or squashes) along its principal directions, which *do* change.

This video provides a helpful visual explanation of the geometric interpretation of SVD and singular values. Geometric Interpretation of the SVD http://googleusercontent.com/youtube_content/2

Q

Matrix notation. Does a star after a matrix symbol have a special meaning?

A

Yes, the star symbol (A^*) has a very important meaning, but it is also one of the most ambiguous symbols in matrix algebra. Its definition depends heavily on the subject.

It almost always means one of two things:

1. **Conjugate Transpose** (Most common in modern math and physics)
 2. **Adjugate Matrix** (Common in introductory linear algebra)
-

1. The Conjugate Transpose (or Hermitian Transpose)

In physics, functional analysis, and modern linear algebra involving complex numbers, A^* almost universally means the **conjugate transpose** of A .

This is a two-step operation: 1. **Transpose** the matrix (swap rows and columns, A^T). 2. Take the **complex conjugate** of every element (change the sign of the imaginary part, \overline{A}).

The order doesn't matter: $\mathbf{A}^* = \overline{\mathbf{A}^T} = (\overline{\mathbf{A}})^T$

Example: If $A = \begin{bmatrix} 1 & 2+3i \\ 4i & 5 \end{bmatrix}$

1. **Transpose:** $A^T = \begin{bmatrix} 1 & 4i \\ 2+3i & 5 \end{bmatrix}$
2. **Conjugate:** $A^* = \overline{A^T} = \begin{bmatrix} 1 & -4i \\ 2-3i & 5 \end{bmatrix}$

Special Case (Real Matrices): If a matrix A contains only real numbers, its complex conjugate is just itself ($\overline{A} = A$). Therefore, for real matrices, A^* simplifies to just the **transpose**, A^T .

2. The Adjugate Matrix (or Classical Adjoint)

In some older or more introductory linear algebra textbooks, A^* (or more commonly $\text{adj}(A)$) is used to denote the **adjugate matrix**.

This is the **transpose of the cofactor matrix**. This matrix is mainly used to find the inverse of a matrix using the formula:

$$A^{-1} = \frac{1}{\det(A)} \text{adj}(A)$$

This is a completely different operation from the conjugate transpose.

Summary: The Source of Confusion

The confusion arises because both operations are sometimes called the “adjoint” of a matrix. * “**Hermitian Adjoint**” = Conjugate Transpose (A^*) * “**Classical Adjoint**” = Adjugate Matrix ($\text{adj}(A)$)

This is further complicated by the **dagger** (\dagger) **symbol** you asked about previously.

Here is a quick guide to the notation jungle:

Symbol	Name	Definition	Where It’s Used
A^* (star)	Conjugate Transpose	$\overline{A^T}$	Physics, Modern Linear Algebra
A^H	Conjugate Transpose	$\overline{A^T}$	Unambiguous (H is for Hermitian)
A^\dagger (dagger)	Conjugate Transpose	$\overline{A^T}$	Quantum Mechanics
A^\dagger (dagger)	Pseudo-inverse	Moore-Penrose Inverse	Numerical Linear Algebra, Statistics
$\text{adj}(A)$	Adjugate Matrix	Transpose of the cofactors	Classical Linear Algebra (Inverses)
A^* (star)	Adjugate Matrix	Transpose of the cofactors	<i>Less common, older texts</i>

Conclusion: When you see A^* , it most likely means the **conjugate transpose**. However, you should always check the context of the book or paper to be certain.

This video discusses the relationships between these different “adjoint” and “transpose” concepts. Adjoints, Transposes, and Hermitian Conjugates http://googleusercontent.com/youtube_content/4

Q

Is SVD computation expensive?

A

Yes, **SVD is computationally expensive**.

It is generally considered the most computationally demanding of the common matrix decompositions (like LU and QR).

The “cost” of a decomposition is measured in the number of floating-point operations (flops) it requires, which scales with the size of the matrix.

Cost Comparison (General Idea)

For a square $n \times n$ matrix, here is a simple comparison:

Decomposition	Computational Complexity	Relative Cost
LU Decomposition	$O(n^3)$ (specifically $\sim \frac{2}{3}n^3$)	Fastest
QR Decomposition	$O(n^3)$ (specifically $\sim \frac{4}{3}n^3$)	Medium (roughly 2x LU)
SVD	$O(n^3)$ (specifically $O(10n^3)$ to $O(20n^3)$)	Slowest (much larger constant)

For a rectangular $m \times n$ matrix (where $m > n$): * **LU** isn’t directly applicable (it’s for square matrices). * **QR** scales at roughly $O(mn^2)$. * **SVD** scales at roughly $O(mn^2)$ but with a significantly larger constant, making it much slower in practice.

Why Is SVD Used If It’s So Expensive?

You’re paying a high computational price for **power and robustness**.

SVD is the “Swiss Army knife” of linear algebra. It will *always* work and will *always* give you a useful, stable, and informative answer, even in situations where other methods fail.

The extra cost of SVD is justified when: 1. **The Matrix is Singular or “Ill-Conditioned”**: This is SVD’s greatest strength. If a matrix is non-invertible or “close” to non-invertible, SVD is the most reliable tool to get a meaningful answer (like computing the pseudo-inverse). 2. **You Need to Know the Rank**: SVD is the most reliable way to determine the “effective” rank of a matrix, as it can handle the ambiguity of very small (but non-zero) singular values. 3. **You Need the “Best” Approximation**: As you saw with the pseudo-inverse, SVD is used to find the best possible least-squares or minimum-norm solutions.

In summary: You don’t use SVD to solve a simple, well-behaved $n \times n$ system of equations (you’d use LU for that). You use SVD to analyze, diagnose, and solve complex, ill-conditioned, or rectangular problems where robustness is more important than raw speed.

Cheaper Alternatives to SVD

Because the full SVD is so expensive, several “approximate” methods are used in practice, especially in data science and machine learning:

- **Truncated SVD:** Computes only the top k largest singular values and their corresponding vectors (where k is much smaller than n or m). This is the basis for **Principal Component Analysis (PCA)** and is dramatically faster.
- **Randomized SVD (RSVD):** Uses random sampling to build a smaller matrix that approximates the original, then computes the SVD of that. It is extremely fast and gives a very good approximation of the most important singular values.