COMPSCI 590N
Lecture 8: Numerical Linear Algebra 2

Roy J. Adams

College of Information and Computer Sciences
University of Massachusetts Amherst

Outline

Matrix Inversion

Review: An $n \times n$ square matrix A is said to be **invertible** if there exists an $n \times n$ matrix B such that:

$$AB = BA = I$$

where I is the identity matrix. If B exists it is called the **inverse** and is denoted A^{-1}. Matrix inversion is the process of finding A^{-1} for a given matrix A.

Matrix Inversion: Applications

- Matrix inverses appear in statistics frequently. Part of the reason for this is because of the appearance of a matrix inverse in the PDF of the multivariate normal distribution.

$$\mathcal{N}(x; \mu, \Sigma) \propto \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

Matrix Inversion: Applications

- Matrix inverses appear in statistics frequently. Part of the reason for this is because of the appearance of a matrix inverse in the PDF of the multivariate normal distribution.

$$\mathcal{N}(x; \mu, \Sigma) \propto \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

- The analytical solution for least-squares linear regression involves a matrix inverse.

Matrix Inversion: Applications

- Matrix inverses appear in statistics frequently. Part of the reason for this is because of the appearance of a matrix inverse in the PDF of the multivariate normal distribution.

$$\mathcal{N}(x;\mu,\Sigma) \propto \exp\left(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right)$$

- The analytical solution for least-squares linear regression involves a matrix inverse.
- Matrix inversion plays a fundamental role in many computer graphics routines.

Matrix Inversion: Applications

- Matrix inverses appear in statistics frequently. Part of the reason for this is because of the appearance of a matrix inverse in the PDF of the multivariate normal distribution.

$$\mathcal{N}(x; \mu, \Sigma) \propto \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

- The analytical solution for least-squares linear regression involves a matrix inverse.
- Matrix inversion plays a fundamental role in many computer graphics routines.
- Matrix inversion is a subroutine for many more complex linear algebra computations.

Gauss-Jordan Elimination

- Many algorithms exist for inverting a matrix.

Gauss-Jordan Elimination

- Many algorithms exist for inverting a matrix.
- We will analyze one of the fundamental algorithms called **Gauss-Jordan Elimination**.

Gauss-Jordan Elimination

- Many algorithms exist for inverting a matrix.
- We will analyze one of the fundamental algorithms called **Gauss-Jordan Elimination**.
- **Gaussian Elimination** is a method for solving equations of the form $Ax = b$ where A is a matrix, b is a vector, and we are solving for the vector b.

Gauss-Jordan Elimination

- Many algorithms exist for inverting a matrix.
- We will analyze one of the fundamental algorithms called **Gauss-Jordan Elimination**.
- **Gaussian Elimination** is a method for solving equations of the form $Ax = b$ where A is a matrix, b is a vector, and we are solving for the vector b.
- Gaussian elimination can be thought of as a systematic application of simple substitution rules.

Gauss-Jordan Elimination

- Many algorithms exist for inverting a matrix.
- We will analyze one of the fundamental algorithms called **Gauss-Jordan Elimination**.
- **Gaussian Elimination** is a method for solving equations of the form $Ax = b$ where A is a matrix, b is a vector, and we are solving for the vector b.
- Gaussian elimination can be thought of as a systematic application of simple substitution rules.
- Gauss-Jordan elimination is the application of this idea to the equation $AX = I$ where now X is a matrix rather than a vector.

Gauss-Jordan Elimination

We'll start with a simple example: Let A be the following 2×2 matrix.

Gauss-Jordan Elimination

We'll start with a simple example: Let A be the following 2×2 matrix.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix}$$

Gauss-Jordan Elimination

We'll start with a simple example: Let A be the following 2×2 matrix.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix}$$

Our goal is to find the inverse.

Gauss-Jordan Elimination

We'll start with a simple example: Let A be the following 2×2 matrix.

$$A = \begin{bmatrix} 1 & 3 \\ 2 & 5 \end{bmatrix}$$

Our goal is to find the inverse.

$$A^{-1} = \begin{bmatrix} -5 & 3 \\ 2 & -1 \end{bmatrix}$$

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

- Scaling a row by a non-zero constant.

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

- Scaling a row by a non-zero constant.
- Adding a scaled row to another row.

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

- Scaling a row by a non-zero constant.
- Adding a scaled row to another row.
- Swapping two rows (we will not use this).

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

- Scaling a row by a non-zero constant.
- Adding a scaled row to another row.
- Swapping two rows (we will not use this).

Gauss-Jordan Elimination

We begin by writing A in the following augmented form:

$$A = \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

We then apply elementary row operations until the left side is equal to the identity matrix. Elementary row operations include:

- Scaling a row by a non-zero constant.
- Adding a scaled row to another row.
- Swapping two rows (we will not use this).

If we are able to do this without getting a row of all zeros on the left, then the right side will be A^{-1}. If at any point we get a row with all zeros, then the matrix has no inverse.

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow R_2 - 2R_1$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 2 & 5 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow R_2 - 2R_1$$

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{array}\right]$$

$$R_2 \leftarrow -R_2$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & -1 & -2 & 1 \end{array}\right]$$

$$R_2 \leftarrow -R_2$$

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

$$R_1 \leftarrow R_1 - 3R_2$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

$$R_1 \leftarrow R_1 - 3R_2$$

$$\left[\begin{array}{cc|cc} 1 & 0 & -5 & 3 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

$$R_1 \leftarrow R_1 - 3R_2$$

$$\left[\begin{array}{cc|cc} 1 & 0 & -5 & 3 \\ 0 & 1 & 2 & -1 \end{array}\right]$$

Done!

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

- For each row i from top to bottom:

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

- For each row i from top to bottom:
  - Scale the row so that the diagonal entry equals 1.

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

- For each row i from top to bottom:
  - Scale the row so that the diagonal entry equals 1.
  - Subtract a scaled version of row i from each row below i so that the ith column in each of these rows is 0.

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

- For each row i from top to bottom:
  - Scale the row so that the diagonal entry equals 1.
  - Subtract a scaled version of row i from each row below i so that the ith column in each of these rows is 0.
  - This eliminates all entries below the diagonal and sets the diagonal to ones.

Gauss-Jordan Elimination

With such a small matrix it is hard to get a sense for what the steps are:

- For each row i from top to bottom:
  - Scale the row so that the diagonal entry equals 1.
  - Subtract a scaled version of row i from each row below i so that the ith column in each of these rows is 0.
  - This eliminates all entries below the diagonal and sets the diagonal to ones.
- Repeat this process from the bottom up, this time eliminating entries above the diagonal.

Gauss-Jordan Elimination

$$A = \begin{bmatrix} 2 & 3 & 0 \\ 1 & -2 & -1 \\ 2 & 0 & -1 \end{bmatrix}$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 2 & 3 & 0 & 1 & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 2 & 3 & 0 & 1 & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

$$R_1 \leftarrow \frac{1}{2} R_1$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 2 & 3 & 0 & 1 & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

$$R_1 \leftarrow \frac{1}{2}R_1$$

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow R_2 - R_1$$
$$R_3 \leftarrow R_3 - 2R_1$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & -2 & -1 & 0 & 1 & 0 \\ 2 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow R_2 - R_1$$
$$R_3 \leftarrow R_3 - 2R_1$$

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & -\frac{7}{2} & -1 & -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & -\frac{7}{2} & -1 & -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & -\frac{7}{2} & -1 & -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow -\frac{7}{2}R_2$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & -\frac{7}{2} & -1 & -\frac{1}{2} & 1 & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

$$R_2 \leftarrow -\frac{7}{2} R_2$$

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

$$R_3 \leftarrow R_3 + \frac{6}{2}R_2$$
$$R_3 \leftarrow -7R_3$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & -\frac{6}{2} & -1 & -1 & 0 & 1 \end{array}\right]$$

$$R_3 \leftarrow R_3 + \frac{6}{2} R_2$$
$$R_3 \leftarrow -7 R_3$$

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

$$R_2 \leftarrow R_2 - \frac{2}{7} R_3$$

$$R_1 \leftarrow R_1 + 0 R_3$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & \frac{2}{7} & \frac{1}{7} & -\frac{2}{7} & 0 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

$$R_2 \leftarrow R_2 - \frac{2}{7}R_3$$
$$R_1 \leftarrow R_1 + 0R_3$$

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

$$R_1 \leftarrow R_1 - \frac{3}{2} R_2$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

$$R_1 \leftarrow R_1 - \frac{3}{2}R_2$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 2 & 3 & -3 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Gauss-Jordan Elimination

$$\left[\begin{array}{ccc|ccc} 1 & \frac{3}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

$$R_1 \leftarrow R_1 - \frac{3}{2}R_2$$

$$\left[\begin{array}{ccc|ccc} 1 & 0 & 0 & 2 & 3 & -3 \\ 0 & 1 & 0 & -1 & -2 & 2 \\ 0 & 0 & 1 & -4 & -6 & -7 \end{array}\right]$$

Done!

Gauss-Jordan Complexity

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.
  - n times. Once per row.

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.
  - n times. Once per row.
- When performing a row reduction (adding one scaled row to another), how many multiplications and additions do we perform? How many many rows to we add row i to?

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.
  - n times. Once per row.
- When performing a row reduction (adding one scaled row to another), how many multiplications and additions do we perform? How many many rows to we add row i to?
  - n multiplications. (technically $n - i$)

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.
  - n times. Once per row.
- When performing a row reduction (adding one scaled row to another), how many multiplications and additions do we perform? How many many rows to we add row i to?
  - n multiplications. (technically $n - i$)
  - n additions. (technically $n - i$)

Gauss-Jordan Complexity

- When you scale a row so that its diagonal is one, how many multiplications do we perform? How many times do we do this?
  - n multiplications. One per item in the row.
  - n times. Once per row.
- When performing a row reduction (adding one scaled row to another), how many multiplications and additions do we perform? How many many rows to we add row i to?
  - n multiplications. (technically $n - i$)
  - n additions. (technically $n - i$)
  - We add row i to all rows below row i, so $n - i$ times.

Gauss-Jordan Complexity

$$\text{No. Operations } = 2\sum_{i=1}^{n}(n + 2n(n-i))$$

Gauss-Jordan Complexity

$$\begin{aligned}\text{No. Operations} &= 2\sum_{i=1}^{n}\left(n + 2n(n-i)\right)\\ &= 2\sum_{i=1}^{n}\left(n + 2n^2 - 2ni\right)\end{aligned}$$

Gauss-Jordan Complexity

$$
\begin{aligned}
\text{No. Operations} &= 2\sum_{i=1}^{n}(n + 2n(n-i)) \\
&= 2\sum_{i=1}^{n}\left(n + 2n^2 - 2ni\right) \\
&= 2\left[\sum_{i=1}^{n} n + 2\sum_{i=1}^{n} n^2 - 2\sum_{i=1}^{n} ni\right]
\end{aligned}
$$

Gauss-Jordan Complexity

$$
\begin{aligned}
\text{No. Operations} &= 2\sum_{i=1}^{n}\left(n + 2n(n-i)\right) \\
&= 2\sum_{i=1}^{n}\left(n + 2n^2 - 2ni\right) \\
&= 2\left[\sum_{i=1}^{n} n + 2\sum_{i=1}^{n} n^2 - 2\sum_{i=1}^{n} ni\right] \\
&= 2\left[n^2 + 2n^3 - 2n\frac{n(n+1)}{2}\right]
\end{aligned}
$$

Gauss-Jordan Complexity

$$
\begin{aligned}
\text{No. Operations} &= 2\sum_{i=1}^{n}\left(n + 2n(n-i)\right) \\
&= 2\sum_{i=1}^{n}\left(n + 2n^2 - 2ni\right) \\
&= 2\left[\sum_{i=1}^{n} n + 2\sum_{i=1}^{n} n^2 - 2\sum_{i=1}^{n} ni\right] \\
&= 2\left[n^2 + 2n^3 - 2n\frac{n(n+1)}{2}\right] \\
&= 2\left[n^2 + 2n^3 - n^3 + n^2\right]
\end{aligned}
$$

Gauss-Jordan Complexity

$$
\begin{aligned}
\text{No. Operations} &= 2\sum_{i=1}^{n}(n+2n(n-i)) \\
&= 2\sum_{i=1}^{n}\left(n+2n^2-2ni\right) \\
&= 2\left[\sum_{i=1}^{n}n+2\sum_{i=1}^{n}n^2-2\sum_{i=1}^{n}ni\right] \\
&= 2\left[n^2+2n^3-2n\frac{n(n+1)}{2}\right] \\
&= 2\left[n^2+2n^3-n^3+n^2\right] \\
&= 4n^2+2n^3
\end{aligned}
$$

Gauss-Jordan Complexity

$$\begin{aligned}
\text{No. Operations} &= 2\sum_{i=1}^{n}(n+2n(n-i)) \\
&= 2\sum_{i=1}^{n}\left(n+2n^2-2ni\right) \\
&= 2\left[\sum_{i=1}^{n} n + 2\sum_{i=1}^{n} n^2 - 2\sum_{i=1}^{n} ni\right] \\
&= 2\left[n^2+2n^3-2n\frac{n(n+1)}{2}\right] \\
&= 2\left[n^2+2n^3-n^3+n^2\right] \\
&= 4n^2+2n^3 = \mathcal{O}(n^3)
\end{aligned}$$

Advanced Matrix Inverse Algorithms

As with matrix multiplication, more sophisticated algorithms exist the have complexity between n^2 and n^3.

Estimating the complexity of NumPy Matrix Inverse

The algorithm used by NumPy is not well documented. How could we estimate its complexity?

Estimating the complexity of NumPy Matrix Inverse

The algorithm used by NumPy is not well documented. How could we estimate its complexity? (Hint: We know the complexity is approximately a monomial (e.g. $\mathcal{O}(n^3)$)).

Estimating the complexity of NumPy Matrix Inverse

The algorithm used by NumPy is not well documented. How could we estimate its complexity? (Hint: We know the complexity is approximately a monomial (e.g. $\mathcal{O}(n^3)$)).

$$\begin{aligned} \text{Run time } &\approx Cn^b \\ \log(\text{Run time}) &\approx \log(C) + b\log(n) \end{aligned}$$

Estimating the complexity of NumPy Matrix Inverse

The algorithm used by NumPy is not well documented. How could we estimate its complexity? (Hint: We know the complexity is approximately a monomial (e.g. $\mathcal{O}(n^3)$)).

$$\begin{aligned} \text{Run time } &\approx Cn^b \\ \log(\text{Run time}) &\approx \log(C) + b\log(n) \end{aligned}$$

Solution:

1. Run a bunch of tests for different n and record the run times.
2. Fit a line to the log run times. The slope will be the degree of the polynomial and the intercept will be the logged constant.

Estimating the complexity of NumPy Matrix Inverse

Demo

Eigenreview

Let A be a square $n \times n$ matrix, then the $\mathbf{v}$ is an **eigenvector** of A if

$$A\mathbf{v} = \lambda\mathbf{v}$$

for some constant λ known as an **eigenvalue**. **Eigen decomposition** is the process of finding the eigenvalue/eigenvector pairs of a matrix.

Eigen Decomposition: Applications

- Eigen decompositions are used in many practical applications:

Eigen Decomposition: Applications

- Eigen decompositions are used in many practical applications:
  - Google's PageRank computes the largest eigenvalue/vector pair.

Eigen Decomposition: Applications

- Eigen decompositions are used in many practical applications:
  - Google's PageRank computes the largest eigenvalue/vector pair.
  - Principal Components Analysis (PCA) is used in many data analysis settings to reduce the dimensionality of a dataset and reduce collinearity.

Eigen Decomposition: Applications

- Eigen decompositions are used in many practical applications:
  - Google's PageRank computes the largest eigenvalue/vector pair.
  - Principal Components Analysis (PCA) is used in many data analysis settings to reduce the dimensionality of a dataset and reduce collinearity.
  - Spectral clustering is used in machine learning and computer vision for clustering data points and parts of images. Spectral clustering requires calculating the Eigen decomposition of a similarity matrix.

Eigen Decomposition: Eigenfaces

- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.

Eigen Decomposition: Eigenfaces

- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.
- The basic idea was to calculate the number of images by number of images image covariance matrix and computing the eigenvectors of this matrix.

Eigen Decomposition: Eigenfaces

- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.
- The basic idea was to calculate the number of images by number of images image covariance matrix and computing the eigenvectors of this matrix.
- The result is one eigenvector for each image.

Eigen Decomposition: Eigenfaces

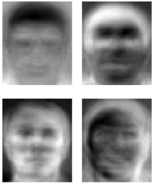
- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.
- The basic idea was to calculate the number of images by number of images image covariance matrix and computing the eigenvectors of this matrix.
- The result is one eigenvector for each image.
- The most well known application was facial recognition where each image was of a face, hence eigenfaces.

Eigen Decomposition: Eigenfaces

- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.
- The basic idea was to calculate the number of images by number of images image covariance matrix and computing the eigenvectors of this matrix.
- The result is one eigenvector for each image.
- The most well known application was facial recognition where each image was of a face, hence eigenfaces.

Eigen Decomposition: Eigenfaces

- One of the seminal early pieces of work in computer vision worked by compressing images into eigenvectors.
- The basic idea was to calculate the number of images by number of images image covariance matrix and computing the eigenvectors of this matrix.
- The result is one eigenvector for each image.
- The most well known application was facial recognition where each image was of a face, hence eigenfaces.



Eigen Decomposition: The Power Method

The Power method is a method for calculating the eigenvalue corresponding to the greatest eigenvalue. Many other methods for calculating the full set of eigenvalues are generalizations of this method.

Eigen Decomposition: The Power Method

The Power method is a method for calculating the eigenvalue corresponding to the greatest eigenvalue. Many other methods for calculating the full set of eigenvalues are generalizations of this method.

The Power Method

1. Given an $n \times n$ matrix A, choose a random initial vector b_0.

Eigen Decomposition: The Power Method

The Power method is a method for calculating the eigenvalue corresponding to the greatest eigenvalue. Many other methods for calculating the full set of eigenvalues are generalizations of this method.

The Power Method

1. Given an $n \times n$ matrix A, choose a random initial vector b_0.
2. Then, under some mild assumptions, the following sequence will converge to the dominant eigenvector:

$$\frac{Ab_0}{\|Ab_0\|}, \frac{A^2b_0}{\|A^2b_0\|}, \frac{A^3b_0}{\|A^3b_0\|}, \dots$$

Eigen Decomposition: Complexity

If we were to calculate matrix matrix powers A^k directly using matrix multiplication, what is the complexity of calculating A^k?

Eigen Decomposition: Complexity

If we were to calculate matrix matrix powers A^k directly using matrix multiplication, what is the complexity of calculating A^k?

- Answer: We would need to perform k matrix multiplications, thus using Strassen's this would take $\approx \mathcal{O}(kn^{2.807})$.

Eigen Decomposition: Complexity

If we were to calculate matrix matrix powers A^k directly using matrix multiplication, what is the complexity of calculating A^k?

- Answer: We would need to perform k matrix multiplications, thus using Strassen's this would take $\approx \mathcal{O}(kn^{2.807})$.

Fortunately, there is a better way. We can use the following interative algorithm:

$$b_k = \frac{A^k b_0}{\|A^k b_0\|} = \frac{A(A^{k-1}b_0)}{\|A(A^{k-1}b_0)\|} = \frac{Ab_{k-1}}{\|Ab_{k-1}\|}$$

Eigen Decomposition: Complexity

If we were to calculate matrix matrix powers A^k directly using matrix multiplication, what is the complexity of calculating A^k?

- Answer: We would need to perform k matrix multiplications, thus using Strassen's this would take $\approx \mathcal{O}(kn^{2.807})$.

Fortunately, there is a better way. We can use the following interative algorithm:

$$b_k = \frac{A^k b_0}{\|A^k b_0\|} = \frac{A(A^{k-1} b_0)}{\|A(A^{k-1} b_0)\|} = \frac{Ab_{k-1}}{\|Ab_{k-1}\|}$$

What is the complexity of computing Ab_{k-1}?

Eigen Decomposition: Complexity

If we were to calculate matrix matrix powers A^k directly using matrix multiplication, what is the complexity of calculating A^k?

- Answer: We would need to perform k matrix multiplications, thus using Strassen's this would take $\approx \mathcal{O}(kn^{2.807})$.

Fortunately, there is a better way. We can use the following interative algorithm:

$$b_k = \frac{A^k b_0}{\|A^k b_0\|} = \frac{A(A^{k-1} b_0)}{\|A(A^{k-1} b_0)\|} = \frac{A b_{k-1}}{\|A b_{k-1}\|}$$

What is the complexity of computing Ab_{k-1}?

- The power method has complexity $\mathcal{O}(n^2)$ **per iteration**.

Inversion and Decomposition in Action: Linear Regression

Let $X \in \mathbb{R}^{n\times m}$ be a $n \times m$ matrix of data cases (i.e. design matrix) and let $y \in \mathbb{R}^n$ be a length n vector of real values. Then in ordinary least squares linear regression, we have the following model:

$$y = \beta X + \epsilon$$

where ϵ is a normally distributed vector of noise.

Inversion and Decomposition in Action: Linear Regression

Let $X \in \mathbb{R}^{n \times m}$ be a $n \times m$ matrix of data cases (i.e. design matrix) and let $y \in \mathbb{R}^n$ be a length n vector of real values. Then in ordinary least squares linear regression, we have the following model:

$$y = \beta X + \epsilon$$

where ϵ is a normally distributed vector of noise. Then using Maximum Likelihood Estimation, we estimate $\hat{\beta}$ as

$$\hat{\beta} = \arg\min_{\beta} (\beta X - y)^T (\beta X - y)$$

Inversion and Decomposition in Action: Linear Regression

The solution to this minimization problem can be found by taking the gradient with respect to β, setting it to zero, and solving. The results is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

Inversion and Decomposition in Action: Linear Regression

The solution to this minimization problem can be found by taking the gradient with respect to β, setting it to zero, and solving. The results is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- We can solve this using just matrix inversion and matrix multiplication.

Inversion and Decomposition in Action: Linear Regression

The solution to this minimization problem can be found by taking the gradient with respect to β, setting it to zero, and solving. The results is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- We can solve this using just matrix inversion and matrix multiplication.
- The advanced linear algebraist may have noticed that $(X^T X)^{-1} X^T$ is called the **Moore-Penrose pseudoinverse**.

Inversion and Decomposition in Action: Linear Regression

The solution to this minimization problem can be found by taking the gradient with respect to β, setting it to zero, and solving. The results is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- We can solve this using just matrix inversion and matrix multiplication.
- The advanced linear algebraist may have noticed that $(X^T X)^{-1} X^T$ is called the **Moore-Penrose pseudoinverse**.
- There are specialized algorithms for computing the Moore-Penrose pseudoinverse.

Inversion and Decomposition in Action: Linear Regression

The solution to this minimization problem can be found by taking the gradient with respect to β, setting it to zero, and solving. The results is:

$$\hat{\beta} = (X^T X)^{-1} X^T y$$

- We can solve this using just matrix inversion and matrix multiplication.
- The advanced linear algebraist may have noticed that $(X^T X)^{-1} X^T$ is called the **Moore-Penrose pseudoinverse**.
- There are specialized algorithms for computing the Moore-Penrose pseudoinverse.
- Part of Assignment 4 will be implementing and comparing linear regression using straight inversion vs. pseudoinversion.

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

- Products: inner, outer, matrix, etc.

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

- Products: inner, outer, matrix, etc.
- Decompositions: Cholesky, QR, SVD, Eigen

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

- Products: inner, outer, matrix, etc.
- Decompositions: Cholesky, QR, SVD, Eigen
- Special numbers: rank, norm, determinant, etc.

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

- Products: inner, outer, matrix, etc.
- Decompositions: Cholesky, QR, SVD, Eigen
- Special numbers: rank, norm, determinant, etc.
- Solvers: Inversion, solve $Ax = b$, least squares, pseudoinversion, etc.

Linear Algebra in NumPy

NumPy has a sub-module called **numpy.linalg** which implements the following groups of methods:

- Products: inner, outer, matrix, etc.
- Decompositions: Cholesky, QR, SVD, Eigen
- Special numbers: rank, norm, determinant, etc.
- Solvers: Inversion, solve $Ax = b$, least squares, pseudoinversion, etc.
- Plus a few more...

Numerical Linear Algebra: Major Takeaways

- If you continue to do numerical computing, you will likely find yourself using some of these linear algebra computations.

Numerical Linear Algebra: Major Takeaways

- If you continue to do numerical computing, you will likely find yourself using some of these linear algebra computations.
- Keep the approximate complexities for the major methods in mind so that you know what is feasible in your programs.

Numerical Linear Algebra: Major Takeaways

- If you continue to do numerical computing, you will likely find yourself using some of these linear algebra computations.
- Keep the approximate complexities for the major methods in mind so that you know what is feasible in your programs.
  - For example: Directly solving linear regression with 1,000 instances is feasible, but 1,000,000 might not be. In this case you should consider a different method.

Numerical Linear Algebra: Major Takeaways

- If you continue to do numerical computing, you will likely find yourself using some of these linear algebra computations.
- Keep the approximate complexities for the major methods in mind so that you know what is feasible in your programs.
  - For example: Directly solving linear regression with 1,000 instances is feasible, but 1,000,000 might not be. In this case you should consider a different method.
- As a rule of thumb, assume $\mathcal{O}(n^3)$ runtime.

Numerical Linear Algebra: Major Takeaways

- If you continue to do numerical computing, you will likely find yourself using some of these linear algebra computations.
- Keep the approximate complexities for the major methods in mind so that you know what is feasible in your programs.
  - For example: Directly solving linear regression with 1,000 instances is feasible, but 1,000,000 might not be. In this case you should consider a different method.
- As a rule of thumb, assume $\mathcal{O}(n^3)$ runtime.
- Approximations for many of these computations exist that are good enough in many cases.