Matrices

Objectives

06 Jan 2025

- review linear algebra concepts and terminology
- **set up and numerically solve linear systems in Julia**
- create and manipulate matrices in Julia
    1. **transposes**
    2. sub-blocks
    3. **sparse matrices**
- recognize an orthogonal matrix and describe its properties
- **compute coordinates using a basis, in particular an orthogonal basis**
- compute lengths of vectors and find angles of vectors in higher dimensions

Linear Algebra Review

Linear algebra provides a way of compactly representing and operating on sets of linear equations. Stanford has a 26-page online linear algebra for machine learning review guide that I recommend. **https://cs229.stanford.edu/section/cs229-linalg.pdf**
the following data and screenshots are taken from the stanford.edu pdf

Basic Concept

represent and operate on sets of linear equations
starting with:

$$\begin{aligned} 4x_1 \; - \; 5x_2 &= -13 \\ -2x_1 \; + \; 3x_2 &= 9. \end{aligned}$$

convert to the form:

$$Ax = b$$

to arrive at:

$$A = \begin{bmatrix} 4 & -5 \\ -2 & 3 \end{bmatrix}, \quad b = \begin{bmatrix} -13 \\ 9 \end{bmatrix}$$

Notation

$A \in \mathbb{R}^{m \times n}$: a matrix with m rows and m columns

$x \in \mathbb{R}^n$, : a vector with n entries

x^T : the transpose of x

$A_{ij}, A_{i,j}$: value 'A' at the i^{th} row and the j^{th} column

Matrix Multiplication

Definition

The product of two matrices $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times p}$ is the matrix

$$C = AB \in \mathbb{R}^{m\times p},$$

where

$$C_{ij} = \sum_{k=1}^{n} A_{ik}B_{kj}.$$

for the **matrix product** to exist, the number of **columns** in **A** must **equal** the number of **rows** in **B**

Dot Product (Inner Product)

Given two vectors $x, y \in \mathbb{R}^n$, the quantity $x^T y$, sometimes called the ***inner product*** or ***dot product*** of the vectors, is a real number given by

$$x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i.$$

dot products are special cases of matrix multiplication where

- **output** is a **scalar**
- $x^T y = y^T x.$

Outer Product

Given vectors $x \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ (not necessarily of the same size), $xy^T \in \mathbb{R}^{m\times n}$ is called the ***outer product*** of the vectors. It is a matrix whose entries are given by $(xy^T)_{ij} = x_i y_j$, i.e.,

$$xy^T \in \mathbb{R}^{m\times n} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_my_1 & x_my_2 & \cdots & x_my_n \end{bmatrix}.$$

- **output** is an $m\times n$ **matrix**

Matrix-Vector Multiplication

Given a matrix $A \in \mathbb{R}^{m\times n}$ and a vector $x \in \mathbb{R}^n$, their product is a vector $y = Ax \in \mathbb{R}^m$. There are a couple ways of looking at matrix-vector multiplication, and we will look at each of them in turn.

If we write A by rows, then we can express Ax as,

$$y = Ax = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} x = \begin{bmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_m^T x \end{bmatrix}.$$

- **'y'** is a **linear combination** of the **columns** of 'A', where the **coefficients** of the linear combination are given by the **entries of 'x'**

Matrix-Matrix Multiplication

(i,j)th C is equal to the inner product of the ith row of A and the jth column of B. Symbolically, this looks like the following,

$$C = AB = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} \begin{bmatrix} | & | & & | \\ b_1 & b_2 & \cdots & b_p \\ | & | & & | \end{bmatrix} = \begin{bmatrix} a_1^T b_1 & a_1^T b_2 & \cdots & a_1^T b_p \\ a_2^T b_1 & a_2^T b_2 & \cdots & a_2^T b_p \\ \vdots & \vdots & \ddots & \vdots \\ a_m^T b_1 & a_m^T b_2 & \cdots & a_m^T b_p \end{bmatrix}.$$

where, $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, $a_i \in \mathbb{R}^n$ and $b_j \in \mathbb{R}^n$

Properties of Matrix Multiplication

- Matrix multiplication is associative: $(AB)C = A(BC)$.
- Matrix multiplication is distributive: $A(B + C) = AB + AC$.
- Matrix multiplication is, in general, *not* commutative; that is, it can be the case that $AB \neq BA$. (For example, if $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times q}$, the matrix product BA does not even exist if m and q are not equal!)

Operations and Properties

Diagonal Matrix

A ***diagonal matrix*** is a matrix where all non-diagonal elements are 0. This is typically denoted $D = \text{diag}(d_1, d_2, \ldots, d_n)$, with

$$D_{ij} = \begin{cases} d_i & i = j \\ 0 & i \neq j \end{cases}$$

Identity Matrix

a **special case** of the **diagonal matrix** where all non-zero values = **1**

The ***identity matrix***, denoted $I \in \mathbb{R}^{n \times n}$, is a square matrix with ones on the diagonal and zeros everywhere else. That is,

$$I_{ij} = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases}$$

It has the property that for all $A \in \mathbb{R}^{m \times n}$,

$$AI = A = IA.$$

Transpose

The ***transpose*** of a matrix results from "flipping" the rows and columns. Given a matrix $A \in \mathbb{R}^{m \times n}$, its transpose, written $A^T \in \mathbb{R}^{n \times m}$, is the $n \times m$ matrix whose entries are given by

$$(A^T)_{ij} = A_{ji}.$$

properties of transpose:

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A + B)^T = A^T + B^T$

Symmetric Matrices

A square matrix $A \in \mathbb{R}^{n \times n}$ is ***symmetric*** if $A = A^T$. It is ***anti-symmetric*** if $A = -A^T$.

any **square matrix** can be represented as a **sum** of a **symmetric** matrix and **anti-symmetric** matrix where the **first** matrix is **symmetric** and the **second** is **antisymmetric**

$$A = \frac{1}{2}(A + A^T) + \frac{1}{2}(A - A^T)$$

the set of all **symmetric matrices** of size ***n*** is denoted as $\mathbb{S}^n$, so that $A \in \mathbb{S}^n$ means that A is a symmetric $n \times n$ matrix

Trace

The ***trace*** of a square matrix $A \in \mathbb{R}^{n \times n}$, denoted $\mathrm{tr}(A)$ (or just $\mathrm{tr}A$ if the parentheses are obviously implied), is the sum of diagonal elements in the matrix:

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii}.$$

properties of **trace**:

- For $A \in \mathbb{R}^{n \times n}$, $\mathrm{tr}A = \mathrm{tr}A^T$.
- For $A, B \in \mathbb{R}^{n \times n}$, $\mathrm{tr}(A + B) = \mathrm{tr}A + \mathrm{tr}B$.
- For $A \in \mathbb{R}^{n \times n}$, $t \in \mathbb{R}$, $\mathrm{tr}(tA) = t\,\mathrm{tr}A$.
- For A, B such that AB is square, $\mathrm{tr}AB = \mathrm{tr}BA$.
- For A, B, C such that ABC is square, $\mathrm{tr}ABC = \mathrm{tr}BCA = \mathrm{tr}CAB$, and so on for the product of more matrices.

Norms

A ***norm*** of a vector $\|x\|$ is informally a measure of the "length" of the vector. For example, we have the commonly-used Euclidean or ℓ_2 norm,

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2}.$$

Note that $\|x\|_2^2 = x^T x$.

More formally, a norm is any function $f : \mathbb{R}^n \to \mathbb{R}$ that satisfies 4 properties:

1. For all $x \in \mathbb{R}^n$, $f(x) \geq 0$ (non-negativity).
2. $f(x) = 0$ if and only if $x = 0$ (definiteness).
3. For all $x \in \mathbb{R}^n$, $t \in \mathbb{R}$, $f(tx) = |t| f(x)$ (homogeneity).
4. For all $x, y \in \mathbb{R}^n$, $f(x + y) \leq f(x) + f(y)$ (triangle inequality).

Other examples of norms are the ℓ_1 norm,

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$

and the ℓ_∞ norm,

$$\|x\|_\infty = \max_i |x_i|.$$

Linear Dependance

A set of vectors $\{x_1, x_2, \ldots x_n\} \subset \mathbb{R}^m$ is said to be ***(linearly) independent*** if no vector can be represented as a linear combination of the remaining vectors. Conversely, if one vector belonging to the set *can* be represented as a linear combination of the remaining vectors, then the vectors are said to be ***(linearly) dependent***. That is, if

$$x_n = \sum_{i=1}^{n-1} \alpha_i x_i$$

for some scalar values $\alpha_1, \ldots, \alpha_{n-1} \in \mathbb{R}$, then we say that the vectors $x_1, \ldots, x_n$ are linearly dependent; otherwise, the vectors are linearly independent. For example, the vectors

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad x_2 = \begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix} \quad x_3 = \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

are linearly dependent because $x_3 = -2x_1 + x_2$.

Inverse

The ***inverse*** of a square matrix $A \in \mathbb{R}^{n \times n}$ is denoted A^{-1}, and is the unique matrix such that

$$A^{-1}A = I = AA^{-1}.$$

A^{-1} may not exist. In particular, we say that A is ***invertible*** or ***non-singular*** if A^{-1} exists and ***non-invertible*** or ***singular*** otherwise.[1]

The following are properties of the inverse; all assume that $A, B \in \mathbb{R}^{n \times n}$ are non-singular:

- $(A^{-1})^{-1} = A$
- $(AB)^{-1} = B^{-1}A^{-1}$
- $(A^{-1})^T = (A^T)^{-1}$. For this reason this matrix is often denoted A^{-T}.

Orthogonal Matrices

Two vectors $x, y \in \mathbb{R}^n$ are ***orthogonal*** if $x^T y = 0$. A vector $x \in \mathbb{R}^n$ is ***normalized*** if $\|x\|_2 = 1$. A square matrix $U \in \mathbb{R}^{n \times n}$ is ***orthogonal*** (note the different meanings when talking about vectors versus matrices) if all its columns are orthogonal to each other and are normalized (the columns are then referred to as being ***orthonormal***).

It follows immediately from the definition of orthogonality and normality that

$$U^T U = I = UU^T.$$

Span

The ***span*** of a set of vectors $\{x_1, x_2, \ldots x_n\}$ is the set of all vectors that can be expressed as a linear combination of $\{x_1, \ldots, x_n\}$. That is,

$$\text{span}(\{x_1, \ldots x_n\}) = \left\{ v : v = \sum_{i=1}^{n} \alpha_i x_i, \quad \alpha_i \in \mathbb{R} \right\}.$$

Range

The ***range*** (sometimes also called the columnspace) of a matrix $A \in \mathbb{R}^{m \times n}$, denoted $\mathcal{R}(A)$, is the the span of the columns of A. In other words,

$$\mathcal{R}(A) = \{v \in \mathbb{R}^m : v = Ax, x \in \mathbb{R}^n\}.$$

Nullspace

The ***nullspace*** of a matrix $A \in \mathbb{R}^{m \times n}$, denoted $\mathcal{N}(A)$ is the set of all vectors that equal 0 when multiplied by A, i.e.,

$$\mathcal{N}(A) = \{x \in \mathbb{R}^n : Ax = 0\}.$$

Determinant

The ***determinant*** of a square matrix $A \in \mathbb{R}^{n\times n}$, is a function $\det : \mathbb{R}^{n\times n} \to \mathbb{R}$, and is denoted $|A|$ or $\det A$ (like the trace operator, we usually omit parentheses). Algebraically, one could write down an explicit formula for the determinant of A, but this unfortunately gives little intuition about its meaning. Instead, we'll start out by providing a geometric interpretation of the determinant and then visit some of its specific algebraic properties afterwards.

Given a matrix

$$\begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_n^T & — \end{bmatrix},$$

consider the set of points $S \subset \mathbb{R}^n$ formed by taking all possible linear combinations of the row vectors $a_1, \ldots, a_n \in \mathbb{R}^n$ of A, where the coefficients of the linear combination are all between 0 and 1; that is, the set S is the restriction of $\text{span}(\{a_1, \ldots, a_n\})$ to only those linear combinations whose coefficients $\alpha_1, \ldots, \alpha_n$ satisfy $0 \le \alpha_i \le 1$, $i = 1, \ldots, n$. Formally,

$$S = \{v \in \mathbb{R}^n : v = \sum_{i=1}^{n} \alpha_i a_i \text{ where } 0 \le \alpha_i \le 1, i = 1, \ldots, n\}.$$

The absolute value of the determinant of A, it turns out, is a measure of the "volume" of the set S.[2]

Algebraically, the determinant satisfies the following three properties (from which all other properties follow, including the general formula):

1. The determinant of the identity is 1, $|I| = 1$. (Geometrically, the volume of a unit hypercube is 1).
2. Given a matrix $A \in \mathbb{R}^{n\times n}$, if we multiply a single row in A by a scalar $t \in \mathbb{R}$, then the determinant of the new matrix is $t|A|$,

$$\left|\begin{bmatrix} — & t\,a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix}\right| = t|A|.$$

   (Geometrically, multiplying one of the sides of the set S by a factor t causes the volume to increase by a factor t.)
3. If we exchange any two rows a_i^T and a_j^T of A, then the determinant of the new matrix is $-|A|$, for example

$$\left|\begin{bmatrix} — & a_2^T & — \\ — & a_1^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix}\right| = -|A|.$$

these **three properties** lend to these **properties**:

- For $A \in \mathbb{R}^{n\times n}$, $|A| = |A^T|$.
- For $A, B \in \mathbb{R}^{n\times n}$, $|AB| = |A||B|$.
- For $A \in \mathbb{R}^{n\times n}$, $|A| = 0$ if and only if A is singular (i.e., non-invertible). (If A is singular then it does not have full rank, and hence its columns are linearly dependent. In this case, the set S corresponds to a "flat sheet" within the n-dimensional space and hence has zero volume.)
- For $A \in \mathbb{R}^{n\times n}$ and A non-singular, $|A^{-1}| = 1/|A|$.

Quadratic Form

Given a square matrix $A \in \mathbb{R}^{n \times n}$ and a vector $x \in \mathbb{R}^n$, the scalar value $x^T Ax$ is called a ***quadratic form***. Written explicitly, we see that

$$x^T Ax = \sum_{i=1}^{n} x_i (Ax)_i = \sum_{i=1}^{n} x_i \left(\sum_{j=1}^{n} A_{ij} x_j \right) = \sum_{i=1}^{n} \sum_{j=1}^{n} A_{ij} x_i x_j \ .$$

Note that,

$$x^T Ax = (x^T Ax)^T = x^T A^T x = x^T \left(\frac{1}{2} A + \frac{1}{2} A^T \right) x,$$

implies:

- A symmetric matrix $A \in \mathbb{S}^n$ is ***positive definite*** (PD) if for all non-zero vectors $x \in \mathbb{R}^n$, $x^T Ax > 0$. This is usually denoted $A \succ 0$ (or just $A > 0$), and often times the set of all positive definite matrices is denoted $\mathbb{S}^n_{++}$.
- A symmetric matrix $A \in \mathbb{S}^n$ is ***positive semidefinite*** (PSD) if for all vectors $x^T Ax \geq 0$. This is written $A \succeq 0$ (or just $A \geq 0$), and the set of all positive semidefinite matrices is often denoted $\mathbb{S}^n_{+}$.
- Likewise, a symmetric matrix $A \in \mathbb{S}^n$ is ***negative definite*** (ND), denoted $A \prec 0$ (or just $A < 0$) if for all non-zero $x \in \mathbb{R}^n$, $x^T Ax < 0$.
- Similarly, a symmetric matrix $A \in \mathbb{S}^n$ is ***negative semidefinite*** (NSD), denoted $A \preceq 0$ (or just $A \leq 0$) if for all $x \in \mathbb{R}^n$, $x^T Ax \leq 0$.
- Finally, a symmetric matrix $A \in \mathbb{S}^n$ is ***indefinite***, if it is neither positive semidefinite nor negative semidefinite — i.e., if there exists $x_1, x_2 \in \mathbb{R}^n$ such that $x_1^T Ax_1 > 0$ and $x_2^T Ax_2 < 0$.

Eigenvalues and Eigenvectors

Given a square matrix $A \in \mathbb{R}^{n\times n}$, we say that $\lambda \in \mathbb{C}$ is an ***eigenvalue*** of A and $x \in \mathbb{C}^n$ is the corresponding ***eigenvector***[3] if

$$Ax = \lambda x, \quad x \neq 0.$$

The following are properties of eigenvalues and eigenvectors (in all cases assume $A \in \mathbb{R}^{n\times n}$ has eigenvalues $\lambda_i, \ldots, \lambda_n$ and associated eigenvectors $x_1, \ldots x_n$):

- The trace of a A is equal to the sum of its eigenvalues,

$$\mathrm{tr}A = \sum_{i=1}^{n} \lambda_i.$$

- The determinant of A is equal to the product of its eigenvalues,

$$|A| = \prod_{i=1}^{n} \lambda_i.$$

- The rank of A is equal to the number of non-zero eigenvalues of A.
- If A is non-singular then $1/\lambda_i$ is an eigenvalue of A^{-1} with associated eigenvector x_i, i.e., $A^{-1}x_i = (1/\lambda_i)x_i$. (To prove this, take the eigenvector equation, $Ax_i = \lambda_i x_i$ and left-multiply each side by A^{-1}.)
- The eigenvalues of a diagonal matrix $D = \mathrm{diag}(d_1, \ldots d_n)$ are just the diagonal entries $d_1, \ldots d_n$.

We can write all the eigenvector equations simultaneously as

$$AX = X\Lambda$$

where the columns of $X \in \mathbb{R}^{n\times n}$ are the eigenvectors of A and Λ is a diagonal matrix whose entries are the eigenvalues of A, i.e.,

$$X \in \mathbb{R}^{n\times n} = \begin{bmatrix} | & | & & | \\ x_1 & x_2 & \cdots & x_n \\ | & | & & | \end{bmatrix}, \quad \Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n).$$

If the eigenvectors of A are linearly independent, then the matrix X will be invertible, so $A = X\Lambda X^{-1}$. A matrix that can be written in this form is called ***diagonalizable***.

An application where eigenvalues and eigenvectors come up frequently is in maximizing some function of a matrix. In particular, for a matrix $A \in \mathbb{S}^n$, consider the following maximization problem,

$$\max_{x\in\mathbb{R}^n} \; x^T Ax \quad \text{subject to } \|x\|_2^2 = 1$$

The Gradient

partial derivatives, defined as:

$$\nabla_A f(A) \in \mathbb{R}^{m\times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

i.e., an $m \times n$ matrix with

$$(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}.$$

Note that the size of $\nabla_A f(A)$ is always the same as the size of A. So if, in particular, A is just a vector $x \in \mathbb{R}^n$,

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}.$$

Note that the size of $\nabla_A f(A)$ is always the same as the size of A. So if, in particular, A is just a vector $x \in \mathbb{R}^n$,

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}.$$

It follows directly from the equivalent properties of partial derivatives that:

- $\nabla_x(f(x) + g(x)) = \nabla_x f(x) + \nabla_x g(x)$.
- For $t \in \mathbb{R}$, $\nabla_x(t\, f(x)) = t\nabla_x f(x)$.

The Hessian

Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is a function that takes a vector in $\mathbb{R}^n$ and returns a real number. Then the ***Hessian*** matrix with respect to x, written $\nabla_x^2 f(x)$ or simply as H is the $n \times n$ matrix of partial derivatives,

$$\nabla_x^2 f(x) \in \mathbb{R}^{n\times n} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}.$$

In other words, $\nabla_x^2 f(x) \in \mathbb{R}^{n\times n}$, with

$$(\nabla_x^2 f(x))_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}.$$

Note that the Hessian is always symmetric, since

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}.$$

Similar to the gradient, the Hessian is defined only when $f(x)$ is real-valued.

Gradients of the Determinant

Now let's consider a situation where we find the gradient of a function with respect to a matrix, namely for $A \in \mathbb{R}^{n\times n}$, we want to find $\nabla_A|A|$. Recall from our discussion of determinants that

$$|A| = \sum_{i=1}^{n}(-1)^{i+j}A_{ij}|A_{\setminus i,\setminus j}| \quad \text{(for any } j \in 1,\dots,n)$$

so

$$\frac{\partial}{\partial A_{k\ell}}|A| = \frac{\partial}{\partial A_{k\ell}}\sum_{i=1}^{n}(-1)^{i+j}A_{ij}|A_{\setminus i,\setminus j}| = (-1)^{k+\ell}|A_{\setminus k,\setminus \ell}| = (\text{adj}(A))_{\ell k}.$$

The Lagrangian

Finally, we use matrix calculus to solve an optimization problem in a way that leads directly to eigenvalue/eigenvector analysis. Consider the following, equality constrained optimization problem:

$$\max_{x\in\mathbb{R}^n} \; x^TAx \quad \text{subject to } \|x\|_2^2 = 1$$

for a symmetric matrix $A \in \mathbb{S}^n$. A standard way of solving optimization problems with equality constraints is by forming the ***Lagrangian***, an objective function that includes the equality constraints.[5] The Lagrangian in this case can be given by

$$\mathcal{L}(x,\lambda) = x^TAx - \lambda x^Tx$$

where λ is called the Lagrange multiplier associated with the equality constraint. It can be established that for x^* to be a optimal point to the problem, the gradient of the Lagrangian has to be zero at x^* (this is not the only condition, but it is required). That is,

$$\nabla_x\mathcal{L}(x,\lambda) = \nabla_x(x^TAx - \lambda x^Tx) = 2A^Tx - 2\lambda x = 0.$$

Notice that this is just the linear equation $Ax = \lambda x$. This shows that the only points which can possibly maximize (or minimize) x^TAx assuming $x^Tx = 1$ are the eigenvectors of A.

Async Materials

Matrices

Linear Transformation

For matrices

$$A\begin{bmatrix}x_1\\x_2\end{bmatrix} = A\left(x_1\begin{bmatrix}1\\0\end{bmatrix} + x_2\begin{bmatrix}0\\1\end{bmatrix}\right) = x_1A\begin{bmatrix}1\\0\end{bmatrix} + x_2A\begin{bmatrix}0\\1\end{bmatrix} = x_1Ae_1 + x_2Ae_2 = x_1u_1 + x_2u_2$$

A linear combination of the columns of A

Given a transformation, you can compute the matrix for it by

$$A = \begin{bmatrix} T(e_1) & T(e_2) & \cdots & T(e_n)\end{bmatrix}$$

This gives you the column form of the matrix product

$$Ax = \begin{bmatrix} u_1 & u_2 & \cdots & u_n\end{bmatrix}x = x_1u_1 + x_2u_2 + \cdots x_nu_n$$

Gaussian Elimination

Convert a system to a series of equivalent system of equations, where eventually the solution is obvious.

$$\begin{bmatrix} 1 & 2 \\ 2 & -3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -2 \\ 4 \\ 4 \end{bmatrix} \qquad \begin{bmatrix} 1 & 2 \\ 2 & -3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 1 \\ 8 \end{bmatrix}$$

$$\left[\begin{array}{cc|c} 1 & 2 & -2 \\ 2 & -3 & 4 \\ 3 & 2 & 4 \end{array}\right] \quad \left[\begin{array}{cc|c} 1 & 2 & -2 \\ 0 & -7 & 8 \\ 0 & -4 & 10 \end{array}\right] \qquad \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 2 & -3 & 1 \\ 3 & 2 & 8 \end{array}\right] \quad \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & -7 & -7 \\ 0 & -4 & -4 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 1 & 2 & -2 \\ 0 & 1 & -8/7 \\ 0 & 1 & -5/2 \end{array}\right] \quad \left[\begin{array}{cc|c} 1 & 0 & 2/7 \\ 0 & 1 & -8/7 \\ 0 & 0 & -19/14 \end{array}\right] \qquad \left[\begin{array}{cc|c} 1 & 2 & 4 \\ 0 & 1 & 1 \\ 0 & 1 & 1 \end{array}\right] \quad \left[\begin{array}{cc|c} 1 & 0 & 2 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{array}\right]$$

x = 3, y = -8/7 solves equations 1 and 2 but not equation 3

x = 2, y = 1 is the only solution to this problem.

Solution with Inverse

Solve the problem T(u) = v

$$\begin{aligned} 5x + 6y + 7z &= -1 \\ 2x + 3y + 5z &= 0 \\ 3x + 2y + 9z &= 9 \end{aligned} \qquad \begin{bmatrix} 5 & 6 & 7 \\ 2 & 3 & 5 \\ 3 & 2 & 9 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -1 \\ 0 \\ 9 \end{bmatrix}$$

Has a unique single solution for every right hand side. This is a mapping, the inverse of T and also a linear transformation. and therefore a matrix, the inverse matrix

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 5 & 6 & 7 \\ 2 & 3 & 5 \\ 3 & 2 & 9 \end{bmatrix}^{-1} \begin{bmatrix} -1 \\ 0 \\ 9 \end{bmatrix} = \begin{bmatrix} 2 \\ -3 \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} 5 & 6 & 7 \\ 2 & 3 & 5 \\ 3 & 2 & 9 \end{bmatrix}^{-1} = \frac{1}{32} \begin{bmatrix} 17 & -40 & 9 \\ -3 & 24 & -11 \\ -5 & 8 & 3 \end{bmatrix}$$

Numerics

Numerical Software

A huge selection of numerical linear algebra methods available.

Low level - BLAS and LAPack

Python - General purpose with packages that implement them. Main ones numpy and scipy

MATLAB/Julia - Primarily numerical languages

```
david — Python — 80×19
>>> import numpy as np
>>> A = np.array([[5, 6, 7],[2,3,5],[3,2,9]])
>>> A
array([[5, 6, 7],
       [2, 3, 5],
       [3, 2, 9]])
>>> x = np.array([2,-3,1])
>>> x
array([ 2, -3,  1])
>>> b = np.matmul(A,x)
>>> b
array([-1,  0,  9])
>>>
```

```
bin — julia — 80×19
julia> A = [5 6 7;2 3 5;3 2 9]
3×3 Matrix{Int64}:
 5  6  7
 2  3  5
 3  2  9

julia> x = [2,-3,1]
3-element Vector{Int64}:
  2
 -3
  1

julia> b = A*x
3-element Vector{Int64}:
 -1
  0
  9

julia>
```

Leaving aside for now how the numerical methods work, solving linear systems is handled differently.

Julia adds a new operator to the language - \ - just for this purpose.

```
>>> import numpy as np
>>> A = np.array([[5,6,7],[2,3,5],[3,2,9]])
>>> b = np.array([-1,0,9])
>>> x = np.linalg.solve(A,b)
>>> x
array([ 2., -3.,  1.])
>>> A
array([[5, 6, 7],
       [2, 3, 5],
       [3, 2, 9]])
>>> A*x
array([[ 10., -18.,   7.],
       [  4.,  -9.,   5.],
       [  6.,  -6.,   9.]])
>>> np.dot(A,x)
array([-1.00000000e+00, -5.55111512e-16,  9.00000000e+00])
>>> np.linalg.inv(A)
array([[ 0.53125, -1.25   ,  0.28125],
       [-0.09375,  0.75   , -0.34375],
       [-0.15625,  0.25   ,  0.09375]])
>>>
```

```
julia> A = [5 6 7;2 3 5;3 2 9]
3×3 Matrix{Int64}:
 5  6  7
 2  3  5
 3  2  9

julia> b = [-1,0,9];

julia> x = A\b
3-element Vector{Float64}:
  2.0
 -3.0
  0.9999999999999999

julia> A*x
3-element Vector{Float64}:
 -1.0000000000000009
 -8.881784197001252e-16
  8.999999999999998

julia> inv(A)
3×3 Matrix{Float64}:
  0.53125  -1.25   0.28125
 -0.09375   0.75  -0.34375
 -0.15625   0.25   0.09375
```

Matrix Operations

Transpose

The transpose takes a m x n matrix and creates a n x m matrix according to

$$(A^T)_{i,j} = A_{j,i}$$

Rows become columns and columns rows

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}^T = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \qquad \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \qquad \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix} \qquad \begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}^T = \begin{bmatrix} a & d \\ b & e \\ c & f \end{bmatrix}$$

Mirror Diagonal

Transpose and Product

Rule for the transpose of a matrix product is as follows

$$(AB)^T = B^T A^T \qquad (ABC)^T = C^T B^T A^T$$

Follows from the definitions

$$(AB)_{i,j} = \sum_k A_{i,k} B_{k,j}$$

For an inner product, if u and v are column vectors

$$(u^T v)^T = v^T (u^T)^T = v^T u$$

And as expected

$$u \cdot v = u^T v = v^T u = v \cdot u$$

Row and Column Vectors

A is m by n matrix. Break it down in terms of column vectors

$$A = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}, u_i \in \mathbb{R}^m$$

The transpose makes the vectors into row vectors.

$$A^T = \begin{bmatrix} u_1 & u_2 & \cdots & u_n \end{bmatrix}^T = \begin{bmatrix} u_1^T \\ u_2^T \\ \vdots \\ u_n^T \end{bmatrix}$$

Everything in terms of columns. If B is a n by k matrix broken down in terms of rows, write

$$B = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix} = \begin{bmatrix} v_1 & v_2 & \cdots & v_n \end{bmatrix}^T, v_i \in \mathbb{R}^k$$

Dot Product

The dot product of two vectors in 3D is

$$(a,b,c)\cdot(x,y,z) = ax + by + cz$$

That is multiply each component and sum up the result. View this in terms of column vectors.

$$(a,b,c)\cdot(x,y,z) = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \end{bmatrix}^T \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Dot product can be viewed as a matrix product of a row and column vector.

$$Bx = \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_n^T \end{bmatrix} x = \begin{bmatrix} v_1^T x \\ v_2^T x \\ \vdots \\ v_n^T x \end{bmatrix} = \begin{bmatrix} v_1 \cdot x \\ v_2 \cdot x \\ \vdots \\ v_n \cdot x \end{bmatrix}$$

Example of block form

Sub-Blocks

Matrices can be broken up

- Single number
- Column Vector
- Row Vector
- Any horizontal and vertical division

$$\left[\begin{array}{c|c|c|c} 1 & 2 & 3 & 4 \\ \hline 5 & 6 & 7 & 8 \\ \hline 9 & 10 & 11 & 12 \\ \hline 13 & 14 & 15 & 16 \end{array}\right]$$

$$\left[\begin{array}{c|c|c} a_{11} & a_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \\ \hline a_{31} & a_{32} & A_{33} \end{array}\right]$$

```
julia> A = [1 2 3 4;5 6 7 8;9 10 11 12;13 14 15 16]
4×4 Matrix{Int64}:
  1   2   3   4
  5   6   7   8
  9  10  11  12
 13  14  15  16

julia> A[:,1]
4-element Vector{Int64}:
  1
  5
  9
 13

julia> A[1,:]
4-element Vector{Int64}:
 1
 2
 3
 4

julia> A[2:3,3:4]
2×2 Matrix{Int64}:
  7   8
 11  12

julia>
```

Products of Block Matrices

Products of block matrices work, as long as dimensions match

$$\left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \\ \hline A_{31} & A_{32} \end{array}\right] \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \end{array}\right] = \left[\begin{array}{c|c} A_{11}B_{11}+A_{12}B_{21} & A_{11}B_{12}+A_{12}B_{22} \\ \hline A_{21}B_{11}+A_{22}B_{21} & A_{21}B_{12}+A_{22}B_{22} \\ \hline A_{31}B_{11}+A_{32}B_{21} & A_{31}B_{12}+A_{32}B_{22} \end{array}\right]$$

Gives you a way to block a matrix product, for example

$$\begin{bmatrix} A & B \\ 0 & C \end{bmatrix} \begin{bmatrix} A^{-1} & D \\ 0 & C^{-1} \end{bmatrix} = \begin{bmatrix} I & AD+BC^{-1} \\ 0 & I \end{bmatrix}$$

So the inverse is

$$\begin{bmatrix} A & B \\ 0 & C \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & -A^{-1}DC^{-1} \\ 0 & C^{-1} \end{bmatrix}$$

Example

This can really be powerful when we look at matrix matrix product in terms of sub-blocks. For example

$$\begin{bmatrix} 1 & 0 & 5 & 1 \\ 0 & 1 & 2 & 0 \\ 3 & 2 & 1 & 0 \\ 1 & 1 & 0 & 1 \end{bmatrix}^2 = \begin{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} & \begin{bmatrix} 5 & 1 \\ 2 & 0 \end{bmatrix} \\ \begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} & \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \end{bmatrix}^2 = \begin{bmatrix} I & A \\ B & I \end{bmatrix}^2 = \begin{bmatrix} I & A \\ B & I \end{bmatrix} \begin{bmatrix} I & A \\ B & I \end{bmatrix}$$

$$= \begin{bmatrix} I^2+AB & A+A \\ B+B & BA+I^2 \end{bmatrix}$$

$$= \begin{bmatrix} I+AB & 2A \\ 2B & I+BA \end{bmatrix}$$

The only thing left is to compute AB and BA.

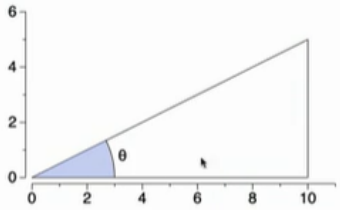
Angles and Orthogonal Matrices

Length and Angles

Compute the angle with respect to the x axis

$$u = (10, 5) = (10, 0) + (0, 5)$$

$$||u|| = \sqrt{u \cdot u} = \sqrt{10^2 + 5^2} = \sqrt{125}$$



Using the sin/cos formulas

$$u = (10, 5) = (||u|| \cos(\theta), ||u|| \sin(\theta))$$

The unit vector in the x direction

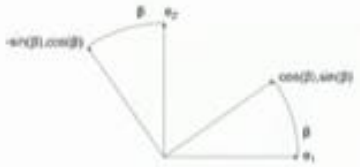
$$e_1 = (1, 0)$$

$$e_1 \cdot u = ||u|| \cos(\theta) = ||e_1|| \, ||u|| \cos(\theta)$$

Rotation in 2-D

This is done with the rotation matrix in 2D

$$R_\beta = \begin{bmatrix} \cos(\beta) & -\sin(\beta) \\ \sin(\beta) & \cos(\beta) \end{bmatrix}$$



The first column vector is what (1,0) maps into, and the second what (0,1) maps into

$$R_\beta \begin{bmatrix} x \\ y \end{bmatrix} = R_\beta \left(x \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right) = x R_\beta \begin{bmatrix} 1 \\ 0 \end{bmatrix} + y R_\beta \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

The first column vector is what (1,0) maps into, and the second what (0,1) maps into

$$R_\beta^T R_\beta = I$$

Because the matrix product is the dot product of all combinations of columns

Angles and Orthogonal Matrices

Even for non-square Q, orthogonal matrices preserve length and angles.

$$(Qu) \cdot (Qv) = (Qu)^T (Qv) = u^T Q^T Q v = u^T I v = u^T v = u \cdot v$$

$$||Qu||^2 = (Qu) \cdot (Qu) = (Qu)^T (Qu) = u^T Q^T Q u = u^T I u = u^T u = ||u||^2$$

Cauchy-Schwartz inequality holds for any vectors in any dimensions

$$|u \cdot v| \leq ||u||\,||v||$$

Which makes the definition of angles well defined

$$\cos(\theta) = \frac{u \cdot v}{||u||\,||v||}$$

And unchanged, i.e. the angle between u and v and Qu and Qv are the same.

2-D Geometric View

Orthogonal matrices preserve angles, but not orientation

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

Orthogonal, but mirrors along the x axis, but cos(angle) = cos(-angle)

Restrictions

The first column has to be on the unit disk (circle with radius 1).

$$(\cos(\theta), \sin(\theta))$$

Second column is orthogonal to the first

$$\pm(-\sin(\theta), \cos(\theta))$$

Rotation and an optional reflection

Coordinates

Definition

What is a coordinate?

(1,2,3)

Everything is with respect to a basis.

$$\begin{bmatrix} 192 \\ 85 \end{bmatrix}$$

192cm and 85kg

$$\begin{bmatrix} 75.6 \\ 187.4 \end{bmatrix}$$

In inches and pounds, a different coordinate system. A different basis.

Coordinate Transform

How are the coordinates related

$$\begin{bmatrix} 75.6 \\ 187.4 \end{bmatrix} \approx \begin{bmatrix} 0.3937 & 0 \\ 0 & 2.205 \end{bmatrix} \begin{bmatrix} 192 \\ 85 \end{bmatrix} = S \begin{bmatrix} 192 \\ 85 \end{bmatrix}$$

This matrix is invertible, and the inverse maps from imperial to metric

The matrix

$$\begin{bmatrix} 0.3937 & 0 \\ 0 & 2.205 \end{bmatrix}$$

Is the coordinate of the metric basis in terms of the imperial basis

Subspaces

Subspaces

$$D \subseteq \mathbb{R}^n$$

Subset is called a Subspace if

$$0 \in D$$

$$x, y \in D \Rightarrow x + y \in D$$

$$x \in D \Rightarrow cy \in D \; \forall c \in \mathbb{R}$$

If it has a vector it has the whole line
If it has two vectors it has a plane

...

Dimension of a subspace is the minimum number of vectors that spans it

Subspaces and Coordinates

$$D = \text{im}(A) = \{Ax : x \in \mathbb{R}^n\} = \text{span}(u_1, \ldots, u_n)$$

If the columns are linearly independent they form a basis

$$b \in \text{im}(A) \Rightarrow b = Ax$$

x is a coordinate only if it is unique

$$b = Ay = Ax \Rightarrow A(y - x) = 0$$

If there are different solutions, there are infinitely many solutions to Ax=b.

$$A(x + c(y - x)) = Ax + cA(y - x) = Ax + 0 = b$$

Check that by computing the row reduced echelon form and see if there is a free variable

Finding the Coordinate

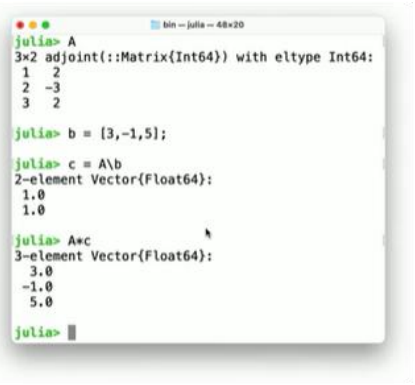
Issue: Given b, if b is in the subspace, find the coordinate

$$\begin{bmatrix} 1 & 2 \\ 2 & -3 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = b = \begin{bmatrix} 3 \\ -1 \\ 5 \end{bmatrix}$$

Julia makes this very simple with the same syntax as solving a linear system

Can also set this up as an augmented system and then row reduce it.

Note that this works because the vector is in the subspace.



Live Session Notes

06 Jan 2026

Instructor	Joseph Slagel (Tanner)
Email	slagel@unc.edu
Website	https://shemesh.larc.nasa.gov/people/jts/ << NASA!!
Office Hours	Friday at 12:00 pm

What to expect

- **"a really fun class"**
- **Julia programming language**
- **assignments due on Sunday**

Julia Programming Language

What is Julia?

- an open-source, multi-platform, high-level, high-performance programming language for **technical computing**.
- an **LLVM-based JIT compiler** that allows it to match the **performance** of languages such as **C** and **FORTRAN without** the hassle of **low-level code**
- **dynamically typed**, provides multiple dispatches, and is designed for **parallelism** and **distributed** computation.
- many built-in **mathematical functions**, including special functions (e.g. Gamma) and **supports complex numbers** right out of the box.
- **generates code** automagically thanks to **Lisp-inspired macros**.

Julia Commands

Accessing Help

```
# Access help mode with an empty ?
?

# Get help on a function with ?functionname
?first

# Search for help on a topic with ?topic
?function
```

Comments

```
# This is a single-line comment
#= This is a
multi-line comment =#
```

Information About Objects

```
# Get the type of an object with typeof() – Example returns Int64
typeof(20)
```

Using Packages

```
# Enter package mode with ] to install and work with packages
]

# Install a new package with add
add CSV

# Exit package mode with DELETE
<DEL>

# Load a package with using
using CSV

# Load a package with import without an alias
import CSV

# Load a package with import with an alias
import DataFrames as df
```

The Working Directory

```
# Get current working director with pwd()
pwd()
"/home/programming_languages/julia"

# Set the current directory with cd()
cd("/home/programming_languages/julia/cheatsheets")
```

Arithmetic Operators

```
# Add two numbers with +
37 + 102

# Subtract two numbers with -
102 - 37

# Multiply two numbers with *
4 * 6

# Divide a number by another with /
21/7

# Integer divide a number with ÷
22 ÷ 7 # This returns 3

# Inverse divide a number with \
5 \ 0 # This is equivalent to 0/5

# Raise to the power using ^
3 ^ 3

# Get the remainder after division with %
22 % 7
```

Assignment Operators

```
# Assign a value to an object with =
a = 5

# Add two objects; store in left-hand object with +=
a += 3 # This is the same as a = a + 3

# Subtract an object from another; store in left-hand object with -=
a -= 3 # This is the same as a = a - 3
```

Numeric Comparison Operators

```
# Test for equality with ==
3 == 3 # This returns true

# Test for not-equality with ≠
3 ≠ 3 # This returns false

# Test greater than with >
3 > 1

# Test greater or equal with ⩾
3 ⩾ 3

# Test less than with <
3 < 4

# Test less or equal than with ⩽
3 ⩽ 4
```

Logical Operators

```
# Logical not with ~
~(2 == 2) # Returns false

# Elementwise and with &
(1 ≠ 1) & (1 < 1) # Returns false

# Elementwise or with |
(1 ⩾ 1) | (1 < 1) # Returns true

# Elementwise xor (exclusive or) with ⊻
(1 ≠ 1) ⊻ (1 < 1) # Returns false
```

Other Operators

```
# Determine if a value is in an array with x in arr
x = [11, 13, 19]
13 in x # This returns true

# Pipe values to a function with value ▷ fn
x ▷ (y → length(y) + sum(y)) # This returns 43
```

Creating Vectors

```
# Create vectors with square brackets, [x1, x2, x3]
x = [1, 2, 3]

# Create vectors, specifying element types using Vector{type}()
Vector{Float64}([1, 2, 3])

# Create sequence of numbers from a to b with a:b
37:100

# Create sequence of numbers from a to b in steps with a:step:b
1:2:101

# Create vector that repeats m times and each element repeats n times
repeat(vector, inner=n, outer=m)
```

Vector Functions

```
# Sorting vectors with sort(x)
x = [9, 1, 4]
sort(x)

# Reversing vectors with reverse(x)
reverse(x)

# Reversing in-place with reverse!(x)
reverse!(x)

# Get vector's unique elements with unique()
unique(x)
```

Selecting Vector Elements

```
# Selecting the 6th element of a vector with x[6]
x = [9, 1, 4, 6, 7, 11, 5]
x[6]

# Selecting the first element of a vector with x[begin]
x[begin] # This is the same as x[1]

# Selecting the last element of a vector with x[end]
x[end] # This is the same as x[7]

# Slicing elements two to six from a vector with x[2:6]
x[2:6]

# Selecting the 2nd and 6th element of a vector with x[[2, 6]]
x[[2,6]]

# Selecting elements equal to 5 with x[x .== 5]
x[x .== 5]

# Selecting elements less than 5 with x[x .< 5]
x[x .< 5]

# Selecting elements in the vector 2, 5, 8 with x[in([2, 5, 8]).(x)]
x[in([2, 5, 8]).(x)]
```

Characters and Strings

```
# Create a character variable with single quotes
char = 'a'

# Create a string variable with double quotes
string = "Hello World!"

# Create a string variable with triple double quotes
string = """Hello
            World!"""

# Extract a single character from a string
string = "Hello World!"

string[1] # This extracts the first character
string[begin] # This extracts the first character
string[end] # This extracts the last character

# Extract a string from a string
string[1:3] # Extract first three characters as a string
string[begin:4] # Extract first four characters as a string
string[end-2: end] # Extract last three characters as a string
```

Combining and Splitting Strings

```
# Combine strings with *
"Listen" * " to " * "DataFramed!" # This returns "Listen to DataFramed!"

# Repeat strings with ^
"Echo! " ^ 3 # Returns "Echo! Echo! Echo! "

# Interpolate strings with "$value"
language = "Julia"
"I'm learning $language" # Returns "I'm learning Julia"

# Split strings on a delimiter with split()
split("lions and tigers and bears", " and ") # Returns 3-element vector
```

Finding and Mutating Strings

```
# Detect the presence of a pattern in a string with occursin()
occursin("Julia", "Julia for data science is cool") # This returns true

# Find the position of the first match in a string with findfirst()
findfirst("Julia", "Julia for data science is cool") # This returns 1:5

# Convert a string to upper case with uppercase()
uppercase("Julia") # Returns "JULIA"

# Convert a string to lower case with lowercase()
lowercase("Julia") # Returns "julia"

# Convert a string to title case case with titlecase()
titlecase("Julia programming") # Returns "Julia Programming"

# Replace matches of a pattern with a new string with replace()
replace("Learn Python on DataCamp.", "Python" ⇒ "Julia")
```

Defining DataFrames

```
# Install the DataFrames and CSV packages
]
add DataFrames
add CSV
using DataFrames
using CSV

# Create a DataFrame with DataFrame()
df = DataFrame(
    numeric_column = 1:4, # Vector of integers
    string_column= ['M', 'F', 'F', 'M'], # Vector of characters
    a_number = 0, # Fill whole column with one integer
    a_string = "data frames" # Fill whole column with one string
)

# Select a row from a data frame with [ and column number
df[3, :] # Return the third row and all columns

# Select a column from a DataFrame using . and column name
df.string_column

# Select a column from a DataFrame using [ and column number
df[:, 2] # Return the second column and all rows

# Select an element from a DataFrame using [ and row and column numbers
df[1, 2] # Return the first row of the second column
```

Manipulating DataFrames

```
# Concatenate two data frames horizontally with hcat()
df1 = DataFrame(column_A = 1:3, column_B = 1:3)
df2 = DataFrame(column_C = 4:6, column_D = 4:6)

df3 = hcat(df1, df2) # Returns 4-column DataFrame with columns A, B, C, D

# Filter for rows of a df3 with filter() where column_A > 2
df_filter = filter(row → row.column_A > 2, df3)

# Select columns of a data frame with select()
select(df3, 2) # Return the second column

# Drop columns of a data frame with select(Not())
select(df3, Not(2)) # Return everything except second column

# Rename columns of a data frame with rename(old → new)
rename(df3, ["column_A" → "first_column"])

# Get rows of a df3 with distinct values in column_A with unique(df, :col)
unique(df3, :column_A)

# Order the rows of a data frame with sort()
sort(df3, :numeric_column)

# Get data frame summary statistics with describe()
describe(df3)
```