Chapter 1

Introduction to Matrices

Contents (class version)

1.0 Introduction ......................................................... 1.3
1.1 Basics .............................................................. 1.4
1.2 Matrix structures ................................................... 1.13
   Notation .......................................................... 1.13
   Common matrix shapes and types ............................... 1.14
   Matrix transpose and symmetry ............................... 1.18
1.3 Multiplication ..................................................... 1.20
   Vector-vector multiplication ................................. 1.20
   Matrix-vector multiplication ............................... 1.23
   Matrix-matrix multiplication ............................... 1.24
   Matrix multiplication properties ........................... 1.24
   Using matrix-vector operations in high-level computing languages ............................ 1.30
1.4 Orthogonality ...................................................... 1.38
1.5 Matrix determinant ............................................... 1.41
1.6 Eigenvalues ....................................................... 1.47
This chapter reviews vectors and matrices and basic properties like shape, orthogonality, determinant, eigenvalues and trace. It also reviews some operations like multiplication and transpose.
1.1 Basics

Why vector?  
- Data organization
- To group into matrices (data)  
  or to be acted on by matrices (operations)

Example: Personal attributes - age, height, weight, eye color (?) ...

Example: Digital grayscale image (!)  
- A vector in the vector space of 2D arrays.
- I rarely think of a digital image as a “matrix”!

(Read the Appendix about general vector spaces.)
Why matrix? Two reasons:
• data array
• linear operation aka linear map or linear transformation

Example: data matrix:
return to personal attributes
person1, person2, ...

Example: linear operation:
DFT matrix in 1D
DFT matrix in 2D
unified as matrix-vector operation:

\[
\begin{align*}
X_{N \times 1 \text{ spectrum}} &= W_{N \times N \text{ DFT}} x_{N \times 1 \text{ signal}}
\end{align*}
\]
Example (classic): solve system of linear equations with $N$ equations in $N$ unknowns:

\[
\begin{align*}
ax_1 + bx_2 + cx_3 &= u \\
dx_1 + ex_2 + fx_3 &= v \\
gx_1 + hx_2 + ix_3 &= w
\end{align*}
\implies \mathbf{Ax} = \mathbf{b}, \quad \text{with } A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}, \quad b = \begin{bmatrix} u \\ v \\ w \end{bmatrix}.
\]

Certainly the matrix-vector notation $\mathbf{Ax} = \mathbf{b}$ is more concise (and general).
A traditional linear algebra course would focus extensively on solving $\mathbf{Ax} = \mathbf{b}$.
This topic will not be a major focus of this course!
Example: linear operation: 
**convolution** in DSP:

\[
y[n] = \sum_{k=0}^{K-1} h[k] x[n - k]
\]

\[
\begin{align*}
(x[0], \ldots, x[N-1]) & \rightarrow \text{LTI} \\
& \quad h[0], \ldots, h[K-1] \\
& \rightarrow (y[0], \ldots, y[M-1])
\end{align*}
\]

Matrix-vector representation of convolution:

where \(x\) is length-\(N\) vector, \(y\) is length-\(M\) vector and \(H\) is a \(M \times N\) matrix with elements

What is \(M\) in terms of \(N\) and \(K\)? (Choose best answer.)

A: \(\max(N, K) - 1\)  B: \(\max(N, K)\)  C: \(N + K\)  D: \(N + K + 1\)  E: None of these

Other matrix operators for other important linear operations: wavelet transform, DCT, 2D DFT, ...
Example: data matrix

A **term-document matrix** is used in information retrieval.

Document 1: ECE510 meets on Tuesdays and Thursdays
Document 2: Tuesday is the most exciting day of the week
Document 3: Let us meet next Thursday.

Keywords (terms): ECE510 meet Tuesday Thursday exciting day week next
(Note: use stem, ignore generic words "the" "and")

Term-document (binary) matrix:

<table>
<thead>
<tr>
<th>Term</th>
<th>Doc1</th>
<th>Doc2</th>
<th>Doc3</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECE510</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>meet</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Tuesday</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Thursday</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>exciting</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>day</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>week</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>next</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
The entries in a (mostly) numerical table like this are naturally represented by a $8 \times 3$ matrix $T$ (because each column is a vector in $\mathbb{R}^8$ and there are three columns) as follows:

$$T = \begin{bmatrix}
1 & 0 & 0 \\
1 & 0 & 1 \\
1 & 1 & 0 \\
1 & 0 & 1 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{bmatrix}.$$  

Why mostly? Column and row labels...

Query as a matrix vector operation: find all documents relevant to the query “exciting days of the week.” See example query vector to the right:

$$q = \begin{bmatrix}
0 \\
0 \\
0 \\
0 \\
1 \\
1 \\
1 \\
0 \\
\end{bmatrix}.$$  

In matrix terms, find columns of $T$ that are “close” (will be defined precisely later) to query vector $q$.

This is an information retrieval application expressed using matrix/vector operations.
Example: Networks, Graphs, and Adjacency Matrices

Consider a set of six web pages that are related via web links (outlinks and inlinks) according to the following directed graph [2, Ex. 1.3, p. 7]:

The $6 \times 6$ adjacency matrix $A$ for this graph has a nonzero value (unity) in element $A_{ij}$ if there is a link from node (web page) $j$ to $i$:

$$
A = \begin{bmatrix}
0 & 1 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 0 & 0 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 \\
\end{bmatrix}, \quad L = \begin{bmatrix}
0 & 1/3 & 0 & 0 & 0 & 0 \\
1/3 & 0 & 0 & 0 & 0 & 0 \\
0 & 1/3 & 0 & 0 & 1/3 & 1/2 \\
1/3 & 0 & 0 & 1/3 & 0 & 0 \\
1/3 & 1/3 & 0 & 0 & 0 & 1/2 \\
0 & 0 & 1 & 0 & 1/3 & 0 \\
\end{bmatrix}.
$$

The link graph matrix $L$ is found from the adjacency matrix by normalizing (each column) with respect to the number of outlinks so each column sums to unity.

We will see later that Google’s PageRank algorithm [3] for quantifying importance of web pages is related to an eigenvector of $L$. (There is a left eigenvector with $\lambda = 1$ so there must also be a right eigenvector with that eigenvalue and that is the one of interest.) So evidently representing web page relationships using a matrix and computing properties of that matrix is useful in many domains, even some that might seem quite unexpected at first.
Example: a simple linear operation

In the preceding two examples (viewing a 2D function and computing volume under a 2D function), the matrix was a 2D array of data. Recall that there were two “whys” for matrices: data and linear operations. Now we turn to an example where the matrix is associated with an linear operation that we apply to vectors.

Consider a polygon $P_1$ defined by $N$ vertex points $(x_1, y_1), \ldots, (x_N, y_N)$, connected in that order, and where the last point is also connected to the first point so there $N$ edges.

Define a new polygon $P_2$ where each vertex is the average of the two points along an edge of polygon $P_1$.

Mathematically, the new vertex points are linearly related to the old points as: $\hat{x}_1 = \frac{x_1 + x_2}{2}, \hat{y}_1 = \frac{y_1 + y_2}{2}$. 
In general:

\[
\begin{bmatrix}
\hat{x}_1 \\
\vdots \\
\hat{x}_{N-1} \\
\hat{x}_N
\end{bmatrix} = \frac{1}{2} \begin{bmatrix}
x_1 + x_2 \\
\vdots \\
x_{N-1} + x_N \\
x_N + x_1
\end{bmatrix},
\begin{bmatrix}
\hat{y}_1 \\
\vdots \\
\hat{y}_{N-1} \\
\hat{y}_N
\end{bmatrix} = \frac{1}{2} \begin{bmatrix}
y_1 + y_2 \\
\vdots \\
y_{N-1} + y_N \\
y_N + y_1
\end{bmatrix}.
\]  

(1.1)

This is a **linear operation** so we can write it concisely in matrix-vector form:

\[
\hat{x} = Ax, \quad \hat{y} = Ay, \quad A \triangleq \frac{1}{2} \begin{bmatrix}
1 & 1 & 0 & 0 & \ldots & 0 \\
0 & 1 & 1 & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0 & 1 & 1 \\
1 & 0 & 0 & 0 & \ldots & 1
\end{bmatrix}.
\]  

(1.2)

The matrix form might not seem more illuminating than the expressions in (1.1) at first. But now suppose we ask the following question: what happens to the polygon shape if we perform the process repeatedly (assuming we do some appropriate normalization)? The answer is not obvious from (1.1) but in matrix form the answer is related to the **power iteration** that we will discuss later for computing the principle eigenvector of \( A \). See [4] and [https://www.jasondavies.com/random-polygon-ellipse/](https://www.jasondavies.com/random-polygon-ellipse/)
1.2 Matrix structures

Notation

- \( \mathbb{R} \): real numbers
- \( \mathbb{C} \): complex numbers
- \( \mathbb{F} \): field (see Appendix on p. 1.56), which here will always be \( \mathbb{R} \) or \( \mathbb{C} \).
  We will use this symbol frequently to discuss properties that hold for both real and complex cases.
- \( \mathbb{R}^N \): set of \( N \)-tuples of real numbers
- \( \mathbb{C}^N \): set of \( N \)-tuples of complex numbers
- \( \mathbb{F}^N \): either \( \mathbb{R}^N \) or \( \mathbb{C}^N \)
- \( \mathbb{R}^{M \times N} \): set of real \( M \times N \) matrices
- \( \mathbb{C}^{M \times N} \): set of complex \( M \times N \) matrices
- \( \mathbb{F}^{M \times N} \): either \( \mathbb{R}^{M \times N} \) or \( \mathbb{C}^{M \times N} \)

Because \( \mathbb{R}^{M \times N} \subset \mathbb{C}^{M \times N} \), whenever you see the symbol \( \mathbb{F}^{M \times N} \) you can just think of it as \( \mathbb{C}^{M \times N} \), but it means the properties being discussed also hold for \( \mathbb{R}^{M \times N} \).
- vectors are typically column vectors here (but see Appendix for more detail)
- row vectors are written \( \mathbf{x}^\top \) or \( \mathbf{x}' \), where \( \mathbf{x} \in \mathbb{R}^n \) or \( \mathbf{x} \in \mathbb{C}^n \).
## Common matrix shapes and types

For each matrix shape, this table gives one of many examples of why that shape is useful in practice.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>diagonal</strong></td>
<td>Covariance matrix of a random vector with uncorrelated elements. Easiest to invert!</td>
</tr>
<tr>
<td><strong>upper triangular</strong></td>
<td>Arises in <strong>Gaussian elimination</strong> for solving systems of equations. Quite easy to invert.</td>
</tr>
<tr>
<td><strong>lower triangular</strong></td>
<td>Used in the <strong>Cholesky decomposition</strong>.</td>
</tr>
<tr>
<td><strong>tridiagonal</strong></td>
<td><strong>Finite difference</strong> approximation of a 2nd derivative. Arises in numerical solutions to differential equations.</td>
</tr>
</tbody>
</table>
The above are all square matrix shapes!
There is also a rectangular diagonal matrix shape that will be useful later for the SVD:

\[
\begin{pmatrix}
\sigma_1 \\
\vdots \\
\sigma_r \\
0
\end{pmatrix}
\quad \text{or} \quad
\begin{pmatrix}
\sigma_1 \\
\vdots \\
\sigma_r \\
0
\end{pmatrix}
\]

Often these are just called “diagonal matrices” too.
Important matrix classes

- A **full matrix** aka **dense matrix**: most entries are nonzero.
- A **sparse matrix**: many entries are zero or few entries are nonzero. Arises in many fields including medical imaging (tomography) [5] (Obviously terms like “most” “many” “few” are qualitative.)
- **Toeplitz**: constant along each diagonal Arises in any application that has time invariance or shift invariance
- **Circulant**: each row is a shifted (circularly to the right by one column) version of the previous row. Arises when considering periodic boundary conditions, e.g., with the DFT

Which statement is correct?
A: All Toeplitz matrices are circulant.
B: All circulant matrices are Toeplitz.
C: Both are true.
D: Neither statement is true.

Which statement is correct?
A: All Toeplitz matrices are full.
B: All sparse matrices are Toeplitz.
C: There are no sparse Toeplitz matrices.
D: None of these statements is true.

What kind of matrix is \( A \) in (1.2)? Choose most specific correct answer.
A: Diagonal  B: Toeplitz  C: Circulant  D: Upper Hessenberg  E: None
**Block matrix classes**

The above definitions of shapes and classes were defined in terms of the scalar entries of a matrix. We generalize these definitions by replacing scalars with matrices, leading to **block matrix** shapes.

**Example:** block diagonal matrix

\[
A = \begin{bmatrix}
A_1 & 0 \\
0 & A_2
\end{bmatrix}
\]

where \(A_1\) and \(A_2\) are arbitrary (possibly full) matrices and each \(0\) denotes an all-zero matrix of the appropriate size.

If \(A_1\) is \(6 \times 8\) and \(A_2\) is \(7 \times 9\) what is the size of the \(0\) matrix in the upper right?

A: \(6 \times 7\)  
B: \(6 \times 9\)  
C: \(7 \times 8\)  
D: \(8 \times 7\)  
E: \(8 \times 9\)

**Example:** block circulant matrix

\[
M = \begin{bmatrix}
A & B & C \\
C & A & B \\
B & C & A
\end{bmatrix}
\]

Combinations of block shapes and the shape of each block are also important. **Example:** a matrix form of the 2D DFT is **block circulant with circulant blocks** (BCCB).
Matrix transpose and symmetry

L§1.1

Define. The transpose of a $M \times N$ matrix $A$ is denoted $A^\top$ and is the $N \times M$ matrix whose $(i, j)$th entry is the $(j, i)$th entry of $A$.

If $A \in \mathbb{C}^{M \times N}$ then its Hermitian transpose is the $N \times M$ matrix whose $(i, j)$th entry is the complex conjugate of the $(j, i)$th entry of $A$.

Common notations for Hermitian transpose are: $A'$, $A^H$ and $A^*$. These notes will mostly use $A'$ because that notation matches MATLAB and JULIA.

If $A$ is real, then $A^H = A^\top$ so these notes will mostly use $A'$ regardless of whether $A$ is real or complex for simplicity of notation.

Define. A matrix $A$ is called symmetric iff $A = A^\top$.

Define. A matrix $A$ is called Hermitian symmetric or just Hermitian iff $A = A'$.

See [1, Example 1.2].

Properties of transpose

- $(A')' = A$
- $(A + B)' = A' + B'$
- $(AB)' = B'A'$ if $A \in \mathbb{F}^{M \times N}$ and $B \in \mathbb{F}^{N \times M}$.
- And more properties ...
Practical implementation

Caution: in MATLAB and JULIA: \( x' \) denotes the **Hermitian transpose** of a (possibly complex) vector (or matrix) \( x \), whereas \( x.\prime \) denotes the **transpose** (defined below).

When performing mathematical operations with complex data, we usually need the Hermitian transpose. However, when simply rearranging how data is stored, sometimes we need only the transpose. Many hours of software debugging time are spent on missing or extra periods for a transpose!

Practical considerations:

- The transpose of a vector takes negligible time in a modern language like MATLAB because the array elements stay in the same order in memory; one just needs to modify the variable type from Vector to RowVector.
- However, the transpose of a (large) array requires considerable shuffling of values in memory and should be avoided when possible to save time.

Example. To compute \( x' A' y \) it may be faster (for large data) to use \((A \ast x)' \ast y\) instead of \( x' \ast A' \ast y\) because the latter may require a transpose operation (unless the compiler is smart enough to avoid it).

For complex data, \( \text{conj}(y' \ast A \ast x) \) is another alternative.
1.3 Multiplication

The defining two operations in linear algebra are addition and multiplication of vectors and matrices. Addition is trivial so these notes focus on multiplication.

Vector-vector multiplication

If \( x \in \mathbb{C}^N \) and \( y \in \mathbb{C}^N \) are two vectors (of the same length) then their dot product or inner product is:

\[
\langle y, x \rangle = x' y = \begin{bmatrix} x_1^* & \cdots & x_N^* \end{bmatrix} \begin{bmatrix} y_1 \\ \vdots \\ y_N \end{bmatrix} = \sum_{i=1}^{N} x_i^* y_i \quad \text{(scalar)}
\]

Most books use \( \langle x, y \rangle = y' x \) whereas [1] uses \( \langle x, y \rangle = x' y \). These notes will use the common convention.

The dot product is central to linear discriminant analysis (LDA), a topic in pattern recognition and machine learning, for two-class classification. In SP terms, it is at the heart of the matched filter for signal detection. The dot product is central to convolution and to neural networks [6], e.g., the perceptron [7].
In contrast, the **outer product** of vector \( x \in \mathbb{C}^M \) with vector of possibly different length \( y \in \mathbb{C}^N \) is the following \( M \times N \) matrix:

\[
x y' = \begin{bmatrix} x_1 \\ \vdots \\ x_M \end{bmatrix} [ y_1^* \cdots y_N^* ]_{1 \times N} = \begin{bmatrix} x_1 y_1^* & x_1 y_2^* & \cdots & x_1 y_N^* \\ \vdots & \vdots & \ddots & \vdots \\ x_M y_1^* & x_M y_2^* & \cdots & x_M y_N^* \end{bmatrix}_{M \times N}
\]

Later we will see that this outer product is a **rank 1 matrix** (unless either \( x \) or \( y \) are 0).

Outer products are central to matrix decompositions like the SVD discussed soon.
Colon notation

Next we consider multiplication with a matrix. First we need some notation because matrix multiplication acts on rows and/or columns of a matrix.

For $A \in \mathbb{F}^{M \times N}$:
- $A_{:,1}$ denotes the first column of $A$ (a vector of length $M$)
- $A_{1,:}$ denotes the first row of $A$ (a row vector of length $N$)

In MATLAB: $A[:,1]$ and $A[1,:]$ essentially do the same.

Using this colon notation we have the following two ways to think about a matrix.
- Column partition of a $M \times N$ matrix: 
  $$A = [A_{:,1} \ldots A_{:,N}]$$
- Row partition of a $M \times N$ matrix: 
  $$A = [A_{1,:} \ldots A_{M,:}]$$

Of course we also have the element-wise way of writing out a $M \times N$ matrix:

$$A = \begin{bmatrix} a_{1,1} & \cdots & a_{1,N} \\ \vdots & \ddots & \vdots \\ a_{M,1} & \cdots & a_{M,N} \end{bmatrix},$$

but for multiplication operations we often think about the column or row partitions above instead.
Matrix-vector multiplication

If $A$ is a $M \times N$ matrix and $x$ is vector of length $N$, then the matrix-vector product is

$$Ax = \begin{bmatrix} A_{1,:} \cdot x \\ \vdots \\ A_{M,:} \cdot x \end{bmatrix} = \sum_{n=1}^{N} A_{:,n}x_n = A_{:,1}x_1 + \cdots + A_{:,N}x_N.$$  

In MATLAB, typically we simply write $y = A \times x$ but we could instead implement matrix-vector multiplication using one of the following two loops, corresponding to the two forms above:

```matlab
y = zeros(M,1); % preallocate
for m = 1:M
    y(m) = A(m,:)\.' * x;
end
```

```matlab
y = zeros(M,1); % preallocate
for n = 1:N
    y = y + A(:,n) * x(n);
end
```

For efficiency reasons and for better readability, it is better to preallocate the result vector $y = \text{zeros}(M,1)$ so that memory does not grow with iteration.

Which of these two for loops could easily use parallel computing (concurrent computation)?

A: Left ($m$)  
B: Right ($n$)  
C: Neither

??
Matrix-matrix multiplication

(Includes matrix-vector and vector-vector multiplication as special cases.)

Let \( A \in \mathbb{F}^{M \times K} \) and \( B \in \mathbb{F}^{K \times N} \) then the result of the matrix-matrix product is \( C = AB \in \mathbb{F}^{M \times N} \).

Define. The standard definition for matrix multiplication: is

\[
C_{ij} = \sum_{k=1}^{K} A_{ik} B_{kj}, \quad \text{for} \ i = 1, \ldots, M, \ j = 1, \ldots, N. \tag{1.3}
\]

Matrix multiplication properties

- The **distributive property** of matrix multiplication is: \( A(B + C) = AB + AC \) if the sizes match appropriately.
- There is also **associative property**: \( A(BC) = (AB)C \) if the sizes match.
- There is no general **commutative property**! In general \( AB \neq BA \) even if the sizes match.
- Multiplication by an (appropriately sized) identity matrix has no effect: \( IA = AI = A \).
- See [wiki] for more matrix multiplication properties.
Four views of matrix multiplication

Besides formula (1.3), there are four (!) ways of viewing (and implementing) matrix-matrix products. Why would you need four different versions? For big data and distributed computing using parallel processing some versions are preferable to others [8].

Version 1 (dot or inner product form)

Using row partition of $A$ and column partition of $B$, one can verify:

\[ C = AB = \begin{bmatrix} A_{1,:} \\ \vdots \\ A_{M,:} \end{bmatrix} \begin{bmatrix} B_{:,1} & \cdots & B_{:,N} \end{bmatrix} = \begin{bmatrix} A_{1,:}B_{:,1} & \cdots & A_{1,:}B_{:,N} \\ \vdots \\ A_{M,:}B_{:,1} & \cdots & A_{M,:}B_{:,N} \end{bmatrix}. \]

Specifically, the elements of $C$ are given by the following (conjugate-less) inner products:

\[ C_{ij} = \]
In MATLAB code:
Simple but opaque: \[ C = A \times B \]

Double loop of vector inner products:

\begin{verbatim}
C = zeros(M,N);
for m=1:M
    for n=1:N
        C(m,n) = A(m,:).' * B(:,n);
    end
end
\end{verbatim}

The transpose .’ (not Hermitian transpose ’) is crucial for complex matrices!

Triple loop completely in terms of scalars per (1.3):

\begin{verbatim}
C = zeros(M,N);
for m=1:M
    for n=1:N
        inprod = 0; % accumulator
        for k=1:K
            inprod = inprod + ... A(m,k) * B(k,n);
        end
        C(m,n) = inprod;
    end
end
\end{verbatim}
Version 2 (column-wise accumulation)

Using column partition of $A$ and elements of $B$:

$$C = AB = \begin{bmatrix} A_{:,1} & \ldots & A_{:,K} \end{bmatrix} \begin{bmatrix} B_{1,1} & \ldots & B_{1,N} \\ \vdots \\ B_{K,1} & \ldots & B_{K,N} \end{bmatrix}$$

$$\implies C_{:,n} = \begin{bmatrix} A_{:,1} & \ldots & A_{:,K} \end{bmatrix} \begin{bmatrix} B_{1,n} \\ \vdots \\ B_{K,n} \end{bmatrix} = \sum_{k=1}^{K} A_{:,k} B(k,n)$$

In MATLAB code we would loop over columns of $A$, performing vector-scalar multiplication:

```matlab
C = zeros(M, N);
for n=1:N
    for k=1:K
        C(:,n) = C(:,n) + A(:,k) * B(k,n);
    end
end
```
version 3 (matrix-vector products)

Writing just $B$ using column partition, one can verify:

$$C = AB = \begin{bmatrix} B_{:,1} & \cdots & B_{:,N} \end{bmatrix} = \begin{bmatrix} AB_{:,1} & \cdots & AB_{:,N} \end{bmatrix}$$

$$\Rightarrow C_{:,j} = \ldots$$

MATLAB code using a single loop over columns of $B$ of matrix-vector products:

```matlab
C = zeros(M,N);
for n=1:N
    C(:,n) = A * B(:,n);
end
```

This class will mostly use the above matrix-vector operation view.
Version 4 (sum of outer products)  

Now using column partition of $A$ and row partition of $B$:

$$C = AB = \begin{bmatrix} A_{:,1} & \cdots & A_{:,K} \end{bmatrix} \begin{bmatrix} B_{1,:} \\ \vdots \\ B_{K,:} \end{bmatrix} = \sum_{k=1}^{K} A_{:,k} \otimes B_{k,:}$$

For proof, see [1, Theorem 1.3].

MATLAB code using a single loop (over inner dimension) of outer products:

```matlab
C = zeros(M,N);
for k=1:K
    C += A(:,k) * B(k,:).'; % column times row!
end
```
Using matrix-vector operations in high-level computing languages

Particularly in high-level, array-oriented languages like MATLAB, matrix and vector operations abound. One should be on the lookout for opportunities to “parallelize” (vectorize) using such operations.

Example. Suppose we want to visualize the 2D Gaussian bump function \( f(x, y) = e^{-(x^2 + 3y^2)} \).

Elementary implementation with double loop:

```matlab
x = linspace(-2, 2, 101)'; % set up as column vector
y = linspace(-1.1, 1.1, 103)';
M = length(x);
N = length(y);
F = zeros(M, N);
for m=1:M
    for n=1:N
        F(m,n) = exp(-(x(m)^2 + 3 * y(n)^2));
    end
end
imagesc(x, y, F.')
colormap gray
```

How do we leverage matrix-vector concepts to write this more concisely? (Concise code is often easier to read and maintain, and often looks more like the mathematical expressions.)
Idea: Define a matrix $A$ such that $A_{ij} = x_i^2 + 3y_j^2$.
Then use element-wise exponential operation: $F = \exp(-A)$

In MATLAB, $\exp(A)$ applies exponentiation element-wise to an array. In MATLAB, $\expm$ and $\exp$ differ greatly!

How do we define $A$ where $A_{ij} = x_i^2 + 3y_j^2$ without writing a double loop?

One way is to use outer products. If $u \in \mathbb{R}^M$ has elements $u_i = x_i^2$ and $v \in \mathbb{R}^N$ has elements $v_j = y_j^2$, then the following sum of outer products yields a $M \times N$ matrix:

$$A = u1_N' + 31_Mv' = \begin{bmatrix} x_1^2 \\ \vdots \\ x_M^2 \end{bmatrix} \begin{bmatrix} 1 & \ldots & 1 \end{bmatrix}_{1 \times N} + 3 \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}_{M \times 1} \begin{bmatrix} y_1^2 & \ldots & y_N^2 \end{bmatrix}_{M \times 1}$$

$$= \begin{bmatrix} x_1^2 & \ldots & x_1^2 \\ \vdots & \vdots & \vdots \\ x_M^2 & \ldots & x_M^2 \end{bmatrix} + 3 \begin{bmatrix} y_1^2 & \ldots & y_N^2 \\ \vdots & \vdots & \vdots \\ y_1^2 & \ldots & y_N^2 \end{bmatrix},$$

where $1_N$ denotes the vector of all ones, i.e., $\text{ones}(N)$ in JULIA or $\text{ones}(N,1)$ in MATLAB.
Here we need $\text{ones}(1,N)$ and $\text{ones}(M,1)$.
The key line of the following MATLAB code looks reasonably close to the above outer-product form.

```matlab
x = linspace(-2, 2, 101)';
y = linspace(-1.1, 1.1, 103)';
M = length(x);
N = length(y);
A = (x.^2) * ones(1,N) + 3 * ones(M,1) * (y.^2).';
F = exp(-A);
imagesc(x, y, F.')
colormap gray
```

This version avoids you, the user, from writing a double loop. Of course MATLAB itself must do double loops internally when evaluating `exp(-A)` and similar expressions.
The outer product approach has the benefit of avoiding writing double loops. However, this type of operation arises so frequently that MATLAB (recent versions only) provides software functions that avoid the unnecessary operations of multiplying by the `ones` vector.

In MATLAB: 

```matlab
[xgrid, ygrid] = ndgrid(x, y); A = xgrid.^2 + 3 * ygrid.^2;
```

This broadcast feature leads to the most concise code that looks the most like the math:

```matlab
x = linspace(-2, 2, 101)'; % set up as column vector
y = linspace(-1.1, 1.1, 103)';
[xgrid, ygrid] = ndgrid(x, y);
A = xgrid.^2 + 3*ygrid.^2; % a lot is happening here!
F = exp(-A);
imagesc(x, y, F.');
colormap gray
```
Finally, if your goal is merely to make a picture of the function \( f(x,y) \), without illustrating any matrix properties, the following is a “JULIA way” to do it. This version is the shortest of all so I added a few labeling commands.

```julia
using Plots; plotly()
x = linspace(-2, 2, 101)
y = linspace(-1.1, 1.1, 103)
f(x,y) = exp(- (x^2 + 3y^2)) # look, no "∗" !
heatmap(x, y, f, color=:grays, aspect_ratio=1)
xlabel!("x"); ylabel!("y"); title!("f(x,y)")
```

![](image.png)
Example. One can perform 1D numerical integration using a vector operation (a dot product).

To compute the area under a curve:

\[
\text{Area} = \int_{a}^{b} f(x) \, dx \approx \sum_{m=1}^{M} (x_m - x_{m-1}) f(x_m) = w' f, \quad w = \begin{bmatrix} x_1 - x_0 \\ \vdots \\ x_M - x_{M-1} \end{bmatrix}, \quad f = \begin{bmatrix} f(x_1) \\ \vdots \\ f(x_M) \end{bmatrix},
\]

(1.5)

where (possibly nonuniformly spaced) sampled points satisfy: \(a = x_0 < x_1 < \ldots < x_M = b\).

This summation is a **dot product** between two vectors in \(\mathbb{R}^M\) and is easy in MATLAB.

\[
f = @x x.^2; \quad \text{% parabola}
\]

\[
x = \text{linspace}(0,3,2000)'; \quad \text{% sample points}
\]

\[
w = \text{diff}(x); \quad \text{% "widths" of rectangles}
\]

\[
\text{Area} = w' * f(x(2:end));
\]

\[
\text{Area, } 3^3/3
\]

What should the exact value be for the area in this example? \(\int_0^3 x^2 \, dx = x^3/3 |_{x=3} = 9\)

**Why** \(x(2:end)\) instead of \(x(1:end)\) or simply \(x\)?

**Because** \(w = \text{diff}(x)\) returns a vector without the first element “\(x_1 - x_0\)” in (1.5).

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_area.html

https://web.eecs.umich.edu/~fessler/course/551/julia/demo/01_area.ipynb
Example. Consider the problem of computing numerically the volume under a 2D function $f(x, y)$:

$$V = \int_{a}^{b} \int_{c}^{d} f(x, y) \, dy \, dx.$$

We can also use matrix-vector products for this operation:

$$V = \int_{a}^{b} \int_{c}^{d} f(x, y) \, dy \, dx \approx S \triangleq \sum_{m=1}^{M} \sum_{n=1}^{N} (x_m - x_{m-1})(y_n - y_{n-1}) f(x_m, y_n),$$

where here $c = y_0 < y_1 < \ldots < y_N = d$.

Define vector $w \in \mathbb{R}^M$ as in 1D example above and $u \in \mathbb{R}^N$ and $F \in \mathbb{R}^{M \times N}$ by

$$w = \begin{bmatrix} x_1 - x_0 \\ \vdots \\ x_M - x_{M-1} \end{bmatrix}, \quad u = \begin{bmatrix} y_1 - y_0 \\ \vdots \\ y_N - y_{N-1} \end{bmatrix}, \quad F = \begin{bmatrix} f(x_1, y_1) & \ldots & f(x_1, y_N) \\ \vdots & \ddots & \vdots \\ f(x_M, y_1) & \ldots & f(x_M, y_N) \end{bmatrix}.$$

Then the above double sum is the following product in matrix-vector form:

$$V \approx S = w^T F u.$$
Proof.

\[
\begin{aligned}
    \mathbf{w}' \mathbf{F} \mathbf{u} &= \begin{bmatrix} w_1 & \ldots & w_M \end{bmatrix} \begin{bmatrix} f(x_1, y_1) & \ldots & f(x_1, y_N) \\ \vdots & & \vdots \\ f(x_M, y_1) & \ldots & f(x_M, y_N) \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_N \end{bmatrix} \\
    &= \begin{bmatrix} w_1 & \ldots & w_M \end{bmatrix} \begin{bmatrix} \sum_{n=1}^{N} u_n f(x_1, y_n) \\ \vdots \\ \sum_{n=1}^{N} u_n f(x_M, y_n) \end{bmatrix} = \sum_{m=1}^{M} w_m \sum_{n=1}^{N} u_n f(x_M, y_n) = S.
\end{aligned}
\]

Again this computation is easy to code in a high-level language like MATLAB.

```matlab
f = @(x,y) exp(-(x.^2 + 3*y.^2)); % gaussian bump function
x = linspace(0,3,2000)'; % sample points
y = linspace(0,2,1000)'; % sample points
[xgrid,ygrid] = ndgrid(x,y);
w = diff(x); % "widths" of rectangles in x
u = diff(y); % "widths" of rectangles in y
F = f(xgrid(2:end,2:end), ygrid(2:end,2:end));
S = w' * F * u;
```

(It would also be fine to write it as a double loop, albeit with more typing and possibly slower in some languages.)
1.4 Orthogonality

For ordinary scalars: $0 \cdot x = 0$.

For vectors and matrices, there are more interesting ways to multiply and end up with zero!

**Vectors**

Define. We say two vectors $x$ and $y$ (of same length) are orthogonal (or perpendicular) if their inner product is zero: $x' y = 0$. In such cases we write $x \perp y$.

If, in addition, $x' x = y' y = 1$ (both unit norm), then we call them orthonormal.

Define. We say a collection of vectors is orthogonal if they are pairwise orthogonal.

Likewise for a set of orthonormal vectors.

**Matrices**

Define. We say a (square) matrix $Q \in \mathbb{R}^{N \times N}$ is an orthogonal matrix iff $Q^\top Q = QQ^\top = I$.

(Personally I think the term “orthonormal” would be more appropriate, but alas.)

Define. We say a complex square matrix $Q \in \mathbb{C}^{N \times N}$ is a unitary matrix iff $Q' Q = QQ' = I$.

The columns of a unitary matrix are orthonormal.

The columns of an orthogonal matrix are orthonormal.

A interesting generalization is a tight frame where only one of the two conditions holds [9, 10].
A key property of orthogonal and unitary matrices is that their inverse is simply their transpose: $Q^{-1} = Q'$. 

<table>
<thead>
<tr>
<th>If a (possibly complex) matrix $A$ has orthonormal columns, is it unitary?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Always</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>If a square, real matrix $A$ has orthogonal columns, is it always an orthogonal matrix?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Yes</td>
</tr>
</tbody>
</table>

??
Define. The (Euclidean) norm of a vector \( x \in \mathbb{F}^N \) is defined by

\[ \|x\| = \sqrt{\langle x, x \rangle} = \sqrt{x'x}. \]  \hspace{1cm} (1.6)

Later when needed we will write \( \|x\|_2 \) but for now we focus on the Euclidean norm, aka 2 norm.

Key properties of the Euclidean norm are:

- Triangle inequality: \( \|u + v\| \leq \|u\| + \|v\| \)
- \( \|u + v\|_2^2 = \|u\|_2^2 + 2 \text{ real } \{u'v\} + \|v\|_2^2 \)  \hspace{1cm} (1.7)
- \( x \perp y \implies \|u + v\|_2^2 = \|u\|_2^2 + \|v\|_2^2 \)

**Norm invariance to rotations**

Orthogonal matrices act like rotation matrices. An important property of \( N \times N \) orthogonal or unitary matrices is that they do not change the Euclidean norm of a vector:

\[ Q \text{ orthogonal or unitary } \implies \forall x \in \mathbb{C}^N. \]  \hspace{1cm} (1.8)

Proof: \( \|Qx\|_2 = \sqrt{(Qx)'(Qx)} = \sqrt{x'Q'Qx} = \sqrt{x'Ix} = \sqrt{x'x} = \|x\|_2 \), using the orthogonality of \( Q \).

This fact is related to Parseval’s theorem.
Now we begin discussing important matrix properties. We start with the matrix determinant [1, Sect. 1.4], a property that is defined only for square matrices. Data matrices are very rarely square, so we essentially never examine the determinant of a data matrix directly! But if $X$ is a $M \times N$ data matrix, often we will work with the $N \times N$ Gram matrix $X'X$ (e.g., when solving least-squares problems) and a Gram matrix is always square. Many operator matrices (like DFT) are also square.

There a many ways to introduce the determinant of a matrix because it has many properties\(^1\). Here we consider the following four “axioms” expressed in terms of matrices.

Define. If $A \in F^{N \times N}$ then the determinant of $A$ is defined so that

- D1: If $A$ is upper triangular and $N \times N$ then \( \det\{A\} = a_{11} \cdot a_{22} \cdots \cdot a_{NN} \)
- D2: If $A, B \in F^{N \times N}$ then \( \det\{AB\} = \det\{A\} \det\{B\} \)
- D3: \( \det\{A'\} = \det\{A\}^* \) where $z^*$ denotes the complex conjugate of $z$.
- D4: If $P_{i,j} \in R^{N \times N}$ denotes the matrix that swaps the $i$th and $j$th rows, $i \neq j$, then \( \det\{P_{i,j}\} = -1. \)

\(^1\) [wiki] lists about 13 properties and claims that a certain set of 3 of them completely characterize the determinant. For our purposes there is no need to make more work for ourselves by using a minimal set so we start with 4 axioms instead.
Fact. A matrix $A$ is invertible iff its determinant is nonzero. So a linear system of $N$ equations with $N$ unknowns has a unique solution iff the determinant corresponding to the coefficients is nonzero. This is the historical reason for the term determinant because it “determines” uniqueness of the solution to such a set of equations.

From D2 it follows immediately that

$$A, B \in \mathbb{F}^{N \times N} \implies \det\{AB\} = \det\{BA\}.$$  \hspace{1cm} (1.9)

Mathematically, a concise definition of the row-swapping matrix $P_{i,j}$ (a special permutation matrix) used in D4 is

$$P_{i,j} = I - e_j e_j' - e_i e_i' + e_j e_i' + e_i e_j',$$

for $i, j \in \{1, \ldots, N\}$, where $e_i$ denotes the $i$th unit vector.

Example. For $N = 5$: $P_{1,4} = P_{4,1} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$, so $P_{1,4} \begin{bmatrix} A_{1,:} \\ A_{2,:} \\ A_{3,:} \\ A_{4,:} \\ A_{5,:} \end{bmatrix} = \begin{bmatrix} A_{4,:} \\ A_{2,:} \\ A_{3,:} \\ A_{1,:} \\ A_{5,:} \end{bmatrix}$.

In words, left multiplying a matrix by $P_{i,j}$ swaps the $i$th and $j$th rows of the matrix.
Row (or column) swap property

A consequence of D2 and D4 is that swapping any two rows of a matrix negates the sign of the determinant:

\[ i \neq j \implies \det \{ P_{i,j} A \} = -\det \{ A \}. \]

By D3, the same property holds for swapping two columns.

Column (or row) scaling property

Use the above four axioms to prove that multiplying a column of \( A \) by a scalar gives a new matrix whose determinant scales by that scalar factor:

\[ B = [a_1, \ldots, a_{n-1}, ba_n, a_{n+1}, \ldots, a_N] \implies \det \{ B \} = b \det \{ A \}. \]

Proof. Simply write \( B = AD \) where \( D = \text{diag} \{1, \ldots, 1, b, 1, \ldots, 1\} \), so \( \det \{ B \} = \det \{ A \} \det \{ D \} = b \det \{ A \} \).

Note the strategy of writing (linear) operations in terms of matrix products so we can apply D2.

Matrix inversion property

Exercise. Use two of the above “axioms” to prove \( \det \{ A^{-1} \} = 1 / \det \{ A \} \).
Row (or column) combination property

Multiplying a row of a matrix by a scalar and adding it to a different row does not change the determinant [1, property 8, p. 5], i.e.:

\[ \text{if } A = \begin{bmatrix} A_{1,:} \\ \vdots \\ A_{M,:) \end{bmatrix} \text{ and } B = \begin{bmatrix} A_{1,:} \\ \vdots \\ bA_{k,:) + A_{m,:} \\ A_{m+1,:} \\ \vdots \\ A_{M,:) \end{bmatrix}, \text{ with } k \neq m, \text{ then } \det\{B\} = \det\{A\}. \]

Proof: \( B = (I + be_m e'_k)A \implies \det\{B\} = \det\{I + be_m e'_k\} \det\{A\} = \det\{A\}. \)

Determinant formula

A general rule, called Laplace’s formula, for a \( N \times N \) matrix \( A \) is:

\[ \det\{A\} = \sum_{n=1}^{N} (-1)^{m+n} a_{m,n} \det\{A_{m,n}\}, \]

for any \( m \in \{1, \ldots, N\} \), where here \( A_{m,n} \) denotes the submatrix of \( A \) formed by deleting the \( m \)th row and \( n \)th column and \( \det\{A_{m,n}\} \) is called a minor of \( A \). (I have rarely needed to use this.)
Sylvester’s determinant identity or matrix determinant lemma:

- \( A, B \in \mathbb{F}^{M \times N} \implies \)

\[
\det\{I_M + AB'\} = \det\{I_N + B'A\}. 
\]

- More generally:

\[
A, B \in \mathbb{F}^{M \times N}, X \in \mathbb{F}^{M \times M} \text{ invertible} \implies \det\{X + AB'\} = \det\{I_N + B'X^{-1}A\} \det\{X\}.
\]
2 by 2 matrix

Use the above properties to show that the determinant of a $2 \times 2$ matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$ is $ad - bc$.

Proof. If $a$ is nonzero, then multiplying the first row by $-c/a$ and adding to the second row yields:

$$B = \begin{bmatrix} a & b \\ 0 & d - bc/a \end{bmatrix},$$

which is upper triangular so has determinant $a(d - bc/a) = ad - bc$.

If $a$ is zero, then swapping the first and second rows yields $C = \begin{bmatrix} c & d \\ 0 & b \end{bmatrix}$ which again is upper triangular and the determinant is $cb$, so the determinant of $A$ in this case is $-cb = 0d - cb = ad - cb$. \qed

Example. $\det\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = 0 \cdot 0 - 1 \cdot 1 = -1$, consistent with D4.

Determinant of $3 \times 3$ matrix

If $A = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$ then $\det\{A\} = a \det\begin{bmatrix} e & f \\ h & i \end{bmatrix} - b \det\begin{bmatrix} d & f \\ g & i \end{bmatrix} + c \det\begin{bmatrix} d & e \\ g & h \end{bmatrix}$.

You should verify this yourself from the properties.
Define. An important property of any square matrix $A \in \mathbb{F}^{N \times N}$ is its eigenvalues, defined as the set of solutions of the characteristic equation

$$\det\{A - zI\} = 0,$$

(1.10)

where $I$ denotes the identity matrix of the same size as $A$.

Viewed as a function of $z$, we call $\det\{A - zI\}$ the characteristic polynomial.

When $A \in \mathbb{F}^{N \times N}$, the characteristic polynomial has degree $N$ and thus $N$ (possibly complex) roots by the fundamental theorem of algebra. Thus by the definition (1.10), $A$ has $N$ eigenvalues (but they are not necessarily distinct).

The literature can be inconsistent about how many eigenvalues a $N \times N$ matrix has. For example, the characteristic polynomial of the $2 \times 2$ matrix $A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$ is $\det\{A - zI\} = z^2 = (z - 0)(z - 0)$ which has a repeated root at zero and sometimes is said to have “only one eigenvalue” [wiki]. We will not use that terminology; we will say that matrix has two eigenvalues, both of which are zero.
As mentioned earlier, data matrices are nearly never square so basically we will never need to examine the eigenvalues of a data matrix $X \in \mathbb{F}^{M \times N}$ directly. But often we will consider the eigenvalues of the square $M \times M$ Gram matrix $X'X$ or of the square $N \times N$ outer-product matrix $XX'$ that arises when estimating a covariance matrix or a scatter matrix.

**Eigenvectors**

If $z$ is an eigenvalue of $A$, then (1.10) implies $A - zI$ is a singular matrix (not invertible), so there exists a nonzero vector $v$ such that $(A - zI)v = 0$ or equivalently

$$Av = zv.$$  \hspace{1cm} (1.11)

Conversely, if (1.11) holds for a nonzero vector $v$, then $z$ is an eigenvalue of $A$.

Define. Any nonzero vector $v$ that satisfies (1.11) is called an eigenvector of $A$.

The usual notation for an eigenvalue is $\lambda$ and when the eigenvalues are all real, by convention we often choose to order them such that $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N$. But the `eig` command does not return the eigenvalues in decreasing order.
Example. Find the eigenvalues of \( A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \).

The characteristic polynomial is \( \det(A - zI) = \det\left\{ \begin{bmatrix} 1 - z & 2 \\ 3 & 4 - z \end{bmatrix} \right\} = (1 - z)(4 - z) - 6 = z^2 - 5z - 2 \), which has roots \( z = \frac{5 \pm \sqrt{33}}{2} \), so the two eigenvalues of \( A \) are \( \lambda_1 = \frac{5 + \sqrt{33}}{2} \), \( \lambda_2 = \frac{5 - \sqrt{33}}{2} \).
Properties of eigenvalues

The defining properties of determinant on p. 1.41 lead to useful eigenvalue properties.

- **D1**  \( \Rightarrow \) If \( A \) is upper triangular, then the eigenvalues of \( A \) are its diagonal elements.

- **D3**  \( \Rightarrow \) The eigenvalues of \( A' \) are the complex conjugates of the eigenvalues of \( A \).

- **D2**  \( \Rightarrow \) The eigenvalues of \( \alpha A \) are \( \alpha \) times the eigenvalues of \( A \) for \( \alpha \in \mathbb{F} \).

- **D2**  \( \Rightarrow \) The eigenvalues of \( A \) are invariant to similarity transforms [1, Theorem 9.19]. If \( A \) is \( N \times N \) and \( T \) is an \( N \times N \) invertible matrix then \( TAT^{-1} \) has the same eigenvalues as \( A \).

As a special case, if \( Q \) is \( N \times N \) is orthogonal (or unitary in complex case) matrix, then \( QAQ' \) has the same eigenvalues as \( A \).

Proof. \( \det \{TAT^{-1} - zI\} = \det \{TAT^{-1} - zTT^{-1}\} = \det \{T(A - zI)T^{-1}\} = \det \{(A - zI)T^{-1}T\} = \det \{(A - zI)I\} = \det \{A - zI\} \). So \( TAT^{-1} \) and \( A \) have the same characteristic equation. Here we used the **distributive property** of matrix multiplication.

Another important property is that the determinant of any square matrix is the product of its eigenvalues [1, Theorem 9.25]:

\[
\det \{A\} = \prod_{i} \lambda_i(A).
\]
Define. Let $S - \{x\}$ denote the set $S$ with the vector $x$ removed.

Suppose $A \in \mathbb{F}^{M \times N}$ and $B \in \mathbb{F}^{N \times M}$. If $z$ is a nonzero eigenvalue of $AB$, then $z$ is also a nonzero eigenvalue of $BA$. We summarize this concisely as the following **commutative property** for eigenvalues:

\[
\text{eig}\{AB\} - \{0\} = \text{eig}\{BA\} - \{0\},
\]

**i.e.**, the nonzero elements of each set of eigenvalues are the same.

Proof. $ABv = zv$ for some nonzero $v$, then clearly $u \triangleq Bv$ is also nonzero. Multiplying by $B$ yields $BABv = zBv \implies BAu = zu$ where $u$ is nonzero, so $z$ is a nonzero eigenvalue of $BA$.

What are the nonzero eigenvalues of the outer product matrix $A = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}$?

A: $\{1, 2, 3\}$  
B: $\{1, 2\}$  
C: $\{1\}$  
D: $\{2\}$  
E: $\{6\}$
Exercise. What are the eigenvalues of \( AB = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & 2 & -2 & 2 \end{bmatrix} \)?

\[
BA = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & 2 & -2 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 4 & 2 \\ 0 & 4 \end{bmatrix}.
\]

So the only nonzero eigenvalue of \( BA \) is 4 (repeated) and thus the only nonzero eigenvalue of \( AB \) is also 4. But can we be sure that \( \text{eig}\{AB\} = (4, 4, 0, 0) \)? The proof on the preceding page does not seem general enough to make that conclusion.

One can verify this conjecture for this specific example using:

\[
A = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & -1 \\ 1 & 1 \end{bmatrix}; \\
B = \begin{bmatrix} 1 & 1 & 1 & 1 \\ -2 & 2 & -2 & 2 \end{bmatrix}; \\
[V,D] = \text{eig}(A*B); \\
\text{diag}(D)
\]

Challenge: resolve this question either by finding a more general proof, or finding counter-example where some nonzero eigenvalue has different multiplicity for \( AB \) than for \( BA \).

In practice, I usually use (1.12) to look for the smallest and or largest eigenvalue, for which multiplicity is unimportant (and hence I have not thought about it).
Trace

Define. The **trace** of a square matrix, denoted $\text{trace}\{A\}$, is the sum of its diagonal elements [1, p. 6].

**Properties of matrix trace** (proved in HW)

- trace{·} is a linear function
- A **cyclic commutative property**:
  \[
  A \in \mathbb{F}^{M \times N}, \quad B \in \mathbb{F}^{N \times M} \implies \text{trace}\{AB\} = \text{trace}\{BA\}.
  \] (1.13)

- Why cyclic? Because (let $A = X$ and $B = YZ$):
  \[
  \text{trace}\{XYZ\} = \text{trace}\{YXZ\} = \text{trace}\{ZXY\} \neq \text{trace}\{XZY\}.
  \]

- trace{A} is the sum of the eigenvalues of $A$ [1, Theorem 9.25].
Invertibility

If $A \in \mathbb{F}^{M \times N}$ and $B \in \mathbb{F}^{N \times M}$ and $BA = I_{N \times N}$ then we call $B$ a left inverse of $A$.

If and $C \in \mathbb{F}^{N \times M}$ and $AC = I_{M \times M}$ then we call $C$ a right inverse of $A$.

For non-square matrices, at most only one of a left or right inverse can exist, and is not unique.

A square matrix $A$ is called invertible iff there exists a matrix $X$ of the same size such that $AX = XA = I$.

For square matrices, a left inverse exists iff only a right inverse exists and the two are identical [wiki].

The easiest matrices to invert are orthogonal (unitary) matrices. Because $Q$ is orthogonal iff

\[ Q'Q = QQ' = I, \tag{1.14} \]

by definition of matrix inverse we have

\[ Q^{-1} = Q'. \tag{1.15} \]

Because of the equivalence of the left and right inverses for invertible square matrices, we really only need one of the two conditions in (1.14) to conclude the other and (1.15). So often people just write $Q'Q = I$ when mentioning that a (square) matrix $Q$ is orthonormal.
In the academic literature (in the software community) there are two different meanings of the term vector.

- In the numerical methods community and in MATLAB, a vector is simply a column of a 2D matrix. In other words, a (column) vector is a $N \times 1$ array of numbers.
- In general mathematics, e.g., linear algebra and functional analysis, a vector belongs to a vector space.

For a nice overview of how this distinction affected the design of the Julia language, see this video.

Although this course will mostly use the numerical methods perspective, students who want a thorough understanding should also be familiar with the more general notion of a vector space.

This Appendix reviews vector spaces and linear operators defined on vector spaces. Although the definitions in this section are quite general and thus might appear somewhat abstract, the ideas are important even for the topics of a sophomore-level signals and systems course! For example, analog systems like passive RLC networks are linear systems that are represented mathematically by linear transformations from one (infinite dimensional) vector space to another.

The definition of a vector space uses the concept of a field of scalars, so we first review that.
Field of scalars

A field or field of scalars $\mathbb{F}$ is a collection of elements $\alpha, \beta, \gamma, \ldots$ along with an “addition” and a “multiplication” operator [11]. For every pair of scalars $\alpha, \beta$ in $\mathbb{F}$, there must correspond a scalar $\alpha + \beta$ in $\mathbb{F}$, called the sum of $\alpha$ and $\beta$, such that

- Addition is commutative: $\alpha + \beta = \beta + \alpha$
- Addition is associative: $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
- There exists a unique element $0 \in \mathbb{F}$, called zero, for which $\alpha + 0 = \alpha, \forall \alpha \in \mathbb{F}$
- For every $\alpha \in \mathbb{F}$, there corresponds a unique scalar $(-\alpha) \in \mathbb{F}$ for which $\alpha + (-\alpha) = 0$.

For every pair of scalars $\alpha, \beta$ in $\mathbb{F}$, there must correspond a scalar $\alpha \beta$ in $\mathbb{F}$, called the product of $\alpha$ and $\beta$, such that

- Multiplication is commutative: $\alpha \beta = \beta \alpha$
- Multiplication is associative: $\alpha(\beta \gamma) = (\alpha \beta) \gamma$
- Multiplication distributes over addition: $\alpha(\beta + \gamma) = \alpha \beta + \alpha \gamma$
- There exists a unique element $1 \in \mathbb{F}$, called one, or unity, or the identity element, for which $1 \alpha = \alpha, \forall \alpha \in \mathbb{F}$
- For every nonzero $\alpha \in \mathbb{F}$, there corresponds a unique scalar $\alpha^{-1} \in \mathbb{F}$, called the inverse of $\alpha$ for which $\alpha \alpha^{-1} = 1$.

Example. The set $\mathbb{Q}$ of rational numbers is a field (with the usual definitions of addition and multiplication). The only fields that we will need are the field of real numbers $\mathbb{R}$ and the field of complex numbers $\mathbb{C}$. 
Vector spaces

A vector space or linear space consists of

- A field $\mathbb{F}$ of scalars.
- A set $\mathcal{V}$ of entities called vectors
- An operation called vector addition that associates a sum $x + y \in \mathcal{V}$ with each pair of vectors $x, y \in \mathcal{V}$ such that
  - Addition is commutative: $x + y = y + x$
  - Addition is associative: $x + (y + z) = (x + y) + z$
  - There exists a unique element $0 \in \mathcal{V}$, called the zero vector, for which $x + 0 = x, \forall x \in \mathcal{V}$
  - For every $x \in \mathcal{V}$, there corresponds a unique vector $(-x) \in \mathcal{V}$ for which $x + (-x) = 0$.
- An operation called multiplication by a scalar that associates with each scalar $\alpha \in \mathbb{F}$ and vector $x \in \mathcal{V}$ a vector $\alpha x \in \mathcal{V}$, called the product of $\alpha$ and $x$, such that:
  - Associative: $\alpha(\beta x) = (\alpha \beta)x$
  - Distributive $\alpha(x + y) = \alpha x + \alpha y$
  - Distributive $(\alpha + \beta)x = \alpha x + \beta x$
  - If 1 is the identity element of $\mathbb{F}$, then $1x = x, \forall x \in \mathcal{V}$.
- No operations are presumed to be defined for multiplying two vectors or adding a vector and a scalar.
Examples of important vector spaces

- **Euclidean n-dimensional space** or **n-tuple space**: $\mathcal{V} = \mathbb{R}^n$.
  
  If $x \in \mathcal{V}$, then $x = (x_1, x_2, \ldots, x_n)$ where $x_i \in \mathbb{R}$.
  
  The field of scalars is $\mathbb{F} = \mathbb{R}$. Of course $x + y = (x_1 + y_1, x_2 + y_2, \ldots, x_n + y_n)$, and $\alpha x = (\alpha x_1, \ldots, \alpha x_n)$.
  
  This space is very closely related to the definition of a vector as a column of a matrix. They are so close that most of the literature does not distinguish them (nor does MATLAB). However, to be rigorous, $\mathbb{R}^n$ is not the same as a column of a matrix because strictly speaking there is no inherent definition of the transpose of a vector in $\mathbb{R}^n$, whereas transpose is well defined for any matrix including a $n \times 1$ matrix.

- **Complex Euclidean n-dimensional space**: $\mathcal{V} = \mathbb{C}^n$. If $x \in \mathcal{V}$, then $x = (x_1, x_2, \ldots, x_n)$ where $x_i \in \mathbb{C}$.
  
  The field of scalars is $\mathbb{F} = \mathbb{C}$ and $x + y = (x_1 + y_1, \ldots, x_n + y_n)$ and $\alpha x = (\alpha x_1, \ldots, \alpha x_n)$.

- $\mathcal{V} = L_2(\mathbb{R}^3)$. The set of functions $f : \mathbb{R}^3 \to \mathbb{C}$ that are **square integrable**: 
  
  $\int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} |f(x, y, z)|^2 \, dx \, dy \, dz < \infty$. The field is $\mathbb{F} = \mathbb{C}$.
  
  Addition and scalar multiplication are defined in the natural way.
  
  To show that $f, g \in L_2(\mathbb{R}^3)$ implies $f + g \in L_2(\mathbb{R}^3)$, one can apply the triangle inequality:
  
  $\|f + g\| \leq \|f\| + \|g\|$ where $\|f\| = \langle f, f \rangle$, and $\langle f, g \rangle = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} f(x, y, z)g^*(x, y, z) \, dx \, dy \, dz$.

- The set of functions on the plane $\mathbb{R}^2$ that are zero outside of the unit square.

- The set of solutions to a homogeneous linear system of equations $A x = 0$. 

Linear transformations and linear operators

Define. Let $\mathcal{U}$ and $\mathcal{V}$ be two vector spaces over a common field $\mathbb{F}$.
A function $A : \mathcal{U} \to \mathcal{V}$ is called a **linear transformation** or **linear mapping** from $\mathcal{U}$ into $\mathcal{V}$ iff $\forall u_1, u_2 \in \mathcal{U}$ and all scalars $\alpha, \beta \in \mathbb{F}$:

$$A(\alpha u_1 + \beta u_2) = \alpha A(u_1) + \beta A(u_2).$$

Example:

- Let $\mathbb{F} = \mathbb{R}$ and let $\mathcal{V}$ be the space of continuous functions on $\mathbb{R}$. Define the linear transformation $A$ by: if $F = A(f)$ then $F(x) = \int_0^x f(t) \, dt$. Thus integration (with suitable limits) is linear.

If $A$ is a linear transformation from $\mathcal{V}$ into $\mathcal{V}$, then we say $A$ is a **linear operator**. However, the terminology distinguishing linear transformations from linear operators is not universal, and the two terms are often used interchangeably.

Simple fact for linear transformations:

- $A[0] = 0$. Proof: $A[0] = A[00] = 0A[0] = 0$. This is called the “zero in, zero out” property.

Caution! (From [12]) By induction it follows that $A \left( \sum_{i=1}^n \alpha_i u_i \right) = \sum_{i=1}^n \alpha_i A(u_i)$ for any finite $n$, but the above does not imply in general that linearity holds for infinite summations or integrals. Further assumptions about “smoothness” or “regularity” or “continuity” of $A$ are needed for that.
Bibliography


