# LAFF

# Linear Algebra
# Foundations and Frontiers

Daniel Homola

Notes for the MOOC course by
The University of Texas at Austin
September 2015

# Contents

# Part I

# Week 1-4

# 1 | Vectors

## 1.1 What is a vector

**Vectors have no location, only a direction and length.** We describe (column) vectors as a list of real numbers, and we call these numbers the components of the vector.

Mathematicians tend to **index** these starting with 1, while in programming 0 is more commonly used as the index of the first component.

The **length of a vector** is Euclidean in most cases, which is calculated as the 2nd norm: summing the squared components and taking the square root of the sum.

$$\|x\|_2 = \sqrt{\sum_{i=0}^{n-1} \chi_i^2} = \sqrt{x^T x}$$

It is also the square root of the dot product of the vector with itself. The 3rd norm (or nth root) is the same but instead of squaring we raise each component to the 3rd and then instead of taking the square root, we take the 3rd root.

**Size** of the vector is just the number of components in the vector.

**Unit basis vectors** are vectors with all their components equal to zero except one, which is set to 1. Using three of these can be used to describe axes of the coordinate system of the 3D space: (0,0,1), (0,1,0), (1,0,0). The notation of the in this course is e1, meaning the 2nd entry of vector e is a 1, all the other components are zero.

## 1.2 Simple vector operations

Two vectors are **equal** if all of their components are the same.

**Adding two vectors together** could be done by summing each component of the same index. The geometrical method, is the parallelogram method, where we lay the first vector's head to the toe of the second, and result is the vector that connects the toe of the first to the head of the second.

**Scaling a vector** by a scalar is achieved by multiplying all components by the scalar.

**Subtracting two vectors** could be done by subtracting each component of the first vector from same index components of the 2nd vector. Geometrically this is laying the -1 scaled version of 2nd vector's toe to the head of the first vector and result is the vector that connects the toe of the first to the head of the second (which now has been rotated 180 degrees).

## 1.3 Advanced vector operations

**AXPY** (alpha times x plus y) operation scales vector $x$ by $\alpha$ then adds $y$ to it.

**Linear combinations of vectors**, given two vectors $u$ and $v$ is $\alpha u + \beta v$, which could be calculated by chaining two AXPYs together:

1. w = AXPY$(a = \alpha, x = u, y = 0)$
2. AXPY$(a = \beta, x = v, y = w)$

This AXPY chaining method could be generalized for n vectors and n scaling components.

**Any vector could be rewritten as the linear combinations of its space's unit vectors.** Following the idea above in a 3D coordinate system, vector u(1,2,3) could be rewritten as: $1 * e_1 + 2 * e_2 + 3 * e_3$. If we denote the scaling vector's components with $\chi$ this operation takes the general form of:

$$\sum_{i=0}^{n-1} \chi_i e_i$$

**Dot = inner product**: multiply the components of the two vectors pair-wise and

add them together.

$$dot(x, y) = x^T y = \sum_{i=0}^{n-1} \chi_i \psi_i$$

So the length of a vector is just square root of the dot product with itself.

**Angle between vectors:** because of the law of cosine ($c^2 = a^2 + b^2 - 2 * ab * cos$) we can prove that for any two vectors regardless of their size=dimension, the angle between them could be calculated as:

$$cos\theta = \frac{x^T y}{\|x\|_2 \|y\|_2}$$

The proof is really simple, and can be watched here. Also from this we can also see that $x^T y = \|x\|_2 \|y\|_2 cos\theta$

Cosine in a right triangle is $b/c$, which in this scenario is the projection of vector y onto x. So the part of y that moves into the same direction. Thus the dot product of two vectors measures how collinear two vectors are and is maximised when they are parallel, and 0, when they are orthogonal since $cos(90°) = 0$.

The **cross product** is a binary operation on two vectors in 3D denoted by the symbol $\times$. The cross product $a \times b$ of two linearly independent vectors a and b is a vector that is perpendicular to both and therefore normal to the plane containing them: $a \times b = \|a\|_2 \|b\|_2 sin\theta n$, where n is the unit vector perpendicular to the plane containing a and b in the direction given by the right hand rule. If a and b are parallel the cross product is 0. It has many applications in physics and engineering and where they use the right hand rule.

The **length of the cross product of two vectors**: $\|x \times y\|_2 = \|x\|_2 \|y\|_2 sin\theta$. In a right triangle the sine is defined as $a/c$ which in this scenario measures the orthogonal component in $y$ with respect to $x$. So the length of the cross product is maximised when the two vectors are orthogonal and 0 when they are parallel, since $sin(0°) = 0$. All of this is nicely summarised by Sal, here.

**Vector functions** can take several scalar and / or vectors as input and yield a vector as output. So it effectively maps a vector to another.

## 1.4 Slicing and dicing

We can partition the dot product into equal sized subvector-operations. If $n = 5$, we can calculate the dot product of the first 3 components of $x$ and $y$: ${\chi_{123}}^T \psi_{123}$, then add it to dot product of the last two components in each vector: ${\chi_{45}}^T \psi_{45}$.

Although it does not make much sense to code vector operations this way, it makes matrix operation run a lot faster as we will see later in the course.

# 2 | Linear transformations

## 2.1 Linear transformations

A vector function $L$ is a linear transformation (LT) if:

- You can scale first and then transform or transform first and then scale:

$$L(\alpha x) = \alpha L(X)$$

**and**

- you can transform first and then sum or sum first and then transform:

$$L(x + y) = L(x) + L(y)$$

Another, more concise way of saying this is, $L$ is a $\mathbb{R}^n \to \mathbb{R}^m$ linear transformation if and only if (iff, $\iff$) for all $u, v \in \mathbb{R}$ and $\alpha, \beta \in \mathbb{R}$:

$$L(\alpha u + \beta v) = \alpha L(u) + \beta L(u)$$

## 2.2 Principle of mathematical induction

Many of the linear transformations could be proven by using the PMI. Which works as follows:

If one can show that:

- **(Base case)** a property holds for $k = k_b$; and
- **Inductive step** if it holds for $k = K$, where $K \geq k_b$, then it also holds for $k = K + 1$
- then one can conclude that the property holds for all integers $k \geq k_b$

## 2.3 Linear transformations as matrices

Since every vector could be rewritten as a linear combination of its space's unit basis vectors, we can think of the linear transformation $L$, such that simply transforms these unit basis vectors, and when we apply it to any arbitrary vector, the resulting vector will be given by the linear combination of these transformed unit basis vectors.

So if we know the transformed unit basis vectors, we don't even need to know the linear transformation function, yet we can calculate the result of such LT on a vector.

The linear transformation $L$ is completely described by the set of vectors $\{a_0, ..., a_{n-1}\}$, where $a_j = L(e_j)$. So for any vector x,

$$L(x) = \sum_{j=0}^{n-1} \chi_j a_j$$

Because of this, we could rewrite linear transformations in a matrix called $A$, where each column holds one of the transformed unit basis vector of the space. Then a LT on the vector $x$ is simply $L(x) = Ax$.

Let $\alpha \in \mathbb{R}^{m \times n}$; and $x, y \in \mathbb{R}^n$. Then

- $A(\alpha x) = \alpha Ax$
- $A(x + y) = Ax + Ay$

Put it simply, **matrix-vector multiplication is always a linear transformation.**

## 2.4 Unit basis vector indexing

If $A$ is a matrix and $e_j$ is a unit basis vector of appropriate length, then $Ae_j = a_j$ where $a_j$ is the jth column of the matrix A.

In other words if we have an $m \times n$ matrix, we can select any column of it, by multiplying it with a vector of length $n$ where each component is 0, except for the one which column we want from $A$.

Equivalently we can do the same with a vector, by multiplying it with the transposed unit basis vector of appropriate length.

**Combining these two rules, we can select any index from a matrix by applying two unit basis vector multiplication.** Let $A$ be a $m \times n$ matrix and $\alpha_{i,j}$ its $(i, j)$ element. Then

$$\alpha_{i,j} = e_i^T (A e_j)$$

## 2.5 Proving that a function is a LT

A function $f : \mathbb{R}^n \to \mathbb{R}^m$ is a linear transformation if and only if it can be written as a matrix vector multiplication. We can exploit this to check if any given function is a linear transformation, using unit basis vectors.

Is $f(\begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}) = \begin{pmatrix} \chi_0 + \chi_1 \\ \chi_1 \end{pmatrix}$ a linear transformation?

- Compute a possible matrix that represents $f$:
  $f(\begin{pmatrix} 1 \\ 0 \end{pmatrix}) = \begin{pmatrix} 1+0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$ and
  $f(\begin{pmatrix} 0 \\ 1 \end{pmatrix}) = \begin{pmatrix} 0+1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$
- if $f$ is a linear transformation, then $f(x) = Ax$ where
  $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$.
- Now,
  $Ax = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} \chi_0 + \chi_1 \\ \chi_1 \end{pmatrix}$

Thus we conclude that $f(x) = Ax$ and therefore $f(x)$ is a linear transformation.

A quick check before doing this: if $f(0) \neq 0$, then $f(x)$ is not a linear transformation.

## 2.6 Rotating vectors in 2D

If we take the first unit basis vector $\begin{pmatrix}1\\0\end{pmatrix}$ in 2D, and rotate it by $\theta = 37°$ then, since the rotated vector's length is still 0, its coordinates will be $\begin{pmatrix}cos\theta\\sin\theta\end{pmatrix}$.

Similarly the other unit basis vector $\begin{pmatrix}0\\1\end{pmatrix}$ will become $\begin{pmatrix}-sin\theta\\cos\theta\end{pmatrix}$.

Thus, the linear transformation that rotates a vector $x \in \mathbb{R}^2$ by an angle $\theta$ is represented by the $2 \times 2$ matrix:

$$\begin{pmatrix}cos(\theta) & -sin(\theta)\\sin(\theta) & cos(\theta)\end{pmatrix}$$

.

# 3 | Matrix vector operations

## 3.1 Special matrices

The **zero matrix** is an $n \times m$ matrix denoted by 0 or $0_{n\times m}$, with all its elements being zero. No matter what vector we multiply it by we get back a zero vector.

$$0_{n\times m}x = 0_m$$

The **identity matrix** is an $n \times n$ matrix denoted by $I$ or $I_n$, with all its elements being zero, except its diagonal which has ones. No matter what vector we multiply it by we get back the same vector.

$$I_{n\times n}x = x_n$$

The **diagonal matrix** is an $n \times n$ matrix, with all its elements being zero, except the diagonal ones which could be anything.

In the **upper triangular matrix** all the elements below the diagonal are zero. In the **strickly upper triangular matrix** all the elements below and on the diagonal are zero. In the **unit upper triangular matrix** all elements below the diagonal are zero, and the diagonal has ones. All three of these have their corresponding lower triangular version.

The **transpose of matrix A** has its rows as its columns:

$$A = \begin{pmatrix} 1_{\alpha_{0,0}} & 2_{\alpha_{0,1}} & 3_{\alpha_{0,2}} \\ 4_{\alpha_{1,0}} & 5_{\alpha_{1,1}} & 6_{\alpha_{1,2}} \end{pmatrix}, A^T = \begin{pmatrix} 1_{\alpha_{0,0}} & 4_{\alpha_{0,1}} \\ 2_{\alpha_{1,0}} & 5_{\alpha_{1,1}} \\ 3_{\alpha_{2,0}} & 6_{\alpha_{2,1}} \end{pmatrix}$$

More formally we could say, let $A \in \mathbb{R}^{m\times n}$ and $B \in \mathbb{R}^{n\times m}$.
Then $B$ is said to be the transpose of $A$ if for $0 \leq i < m$ and $0 \leq j < n, \beta_{j,i} = \alpha_{i,j}$.

$A \in \mathbb{R}^{n\times n}$ is **symmetric matrix** if $A^T = A$, so if $\alpha_{i,j} = \alpha_{j,i}$ for all $i$ and $j$.

## 3.2 Opereations with matrices

**Scaling a matrix** by a scalar simply means to multiply all of its components by it. This is commutative, so if we have a vector $x$ and a matrix $A \in \mathbb{R}^{m \times n}$, $(\beta A)x = \beta(Ax) = (\beta x)A$.

The same holds for matrix-matrix addition.

## 3.3 Matrix vector multiplication

$$
\begin{aligned}
Ax &= \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} \\
&= \left( \chi_0 \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \alpha_{2,0} \end{pmatrix} + \chi_1 \begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \alpha_{2,1} \end{pmatrix} + \chi_2 \begin{pmatrix} \alpha_{0,2} \\ \alpha_{1,2} \\ \alpha_{2,2} \end{pmatrix} \right)
\end{aligned}
$$

In this way we perform AXPY operations, then add these together. But we could also think of the matrix vector multiplication as forming dot products between the vector and the rows of the matrix from top to bottom. This way each dot product will be one of the components of the resulting vector.

Interestingly however, no matter how we partition the matrix-vector multiplication algorithm, the cost of $Ax$ in both cases, where $A \in \mathbb{R}^{m \times n}$ is $2n \times m$ flops.

When we work with triangular or symmetric matrices however, we can slice and dice the matrix and vector in a clever way, which makes the algorithm more complex, but reduces the flops overall since an AXPY or dot product with a zero vector is always zero, so we don't need to compute it. This reduces the cost to $n(n+1)$ flops. This is explained in Week 4's, 2$^{\text{nd}}$ video of both 4.2 and 4.3.

# 4 | Matrix-vector to matrix-matrix multiplication

## 4.1 Predicting the weather as a Markov process

In a Markov chain each event is only determined by its predecessor:

$$\chi_0 \to \chi_1 \to \chi_2 \to \chi_3$$

or to put it more rigorously:

$$P(\chi_3|\chi_0, \chi_1, \chi_2) = P(\chi_2)$$

.

If we incredibly oversimplify weather prediction, we can treat it as a discrete time Markov process, where each day's weather is solely dependent on yesterday's weather. In our hypothetical example we can only have sunny, cloudy and rainy weather, which can change into each other with distinct probabilities as shown in table 12.1.

What is the probability of having a sunny day the day after tomorrow, given today is sunny?

$$0.4 \times 0.4 + 0.4 \times 0.3 + 0.2 \times 0.1 = 0.3$$

This could be generalized by realizing that we can think of table 12.1 as a matrix and describe the fact that today is a sunny day, with the first unit basis vector. Then multiplying these two together will select the first column of the matrix. Then we can multiply this new column vector with the matrix again and we'll get not only the probability of having a sunny day the day after tomorrow, but also a cloudy (0.4) and a rainy (0.3) one.

We can further generalize this, by exponentiating the matrix to the number of days we are interested in. After a given time, if a discrete time Markov chain is both

| | Weather today | | |
|---|---|---|---|
| Weather tomorrow | Sunny | Cloudy | Rainy |
| Sunny | 0.4 | 0.3 | 0.1 |
| Cloudy | 0.4 | 0.3 | 0.6 |
| Rainy | 0.2 | 0.4 | 0.3 |

Table 4.1: Possibilities of changing weather

irreducible and aperiodic, it will converge to a stationary or equilibrium distribution, which will describe the system in the long run.

## 4.2 Matrix vector multiplication with special matrices

If we have a matrix $A \in \mathbb{R}^{m,n}$ and a vector $x$ and we want to compute $A^T x$, it's tempting to just make a copy of $A^T$ store it and then do the matrix vector multiplication. Reading in $A$, requires $2m \times n$ memory operations (memops) and writing/storing it requires $2m \times n$ memops again. Matrix vector multiplication requires $2m \times n$ flops, but flops are much quicker then memops.

Instead of inverting, storing and multiplying, we can modify the matrix-vector operation. In the original algorithm we take each row of the matrix, transpose it and take the dot product of it with the vector $x$. However, notice, that if we don't transpose the row, we will get exactly $A^T x$ at the end. This is a pretty neat trick to speed this kind of operation up a lot.

Implementations achieve better performance (finish faster) if one accesses data consecutively in memory. Now, most scientific computing codes store matrices in "column-major order" which means that the first column of a matrix is stored consecutively in memory, then the second column and so forth. Now, this means that an algorithm that accesses a matrix by columns tends to be faster than an algorithm that accesses a matrix by rows. That, in turn, means that when one is presented with more than one algorithm, one should pick the algorithm that accesses the matrix by columns.

Because of this for a matrix vector multiplication of $Ax + y$, algorithms that use AXPYs do better, while for a matrix vector multiplication of $A^T x + y$, algorithms with dot product are preferable.

Matrix vector multiplication with a symmetric matrix cost $2n^2$ flops.

## 4.3 Matrix-matrix multiplication

We compute a matrix-matrix multiplication of $A \in \mathbb{R}^{3\times 3}$ and $B \in \mathbb{R}^{3\times 3}$ as a chain of matrix-vector multiplications.

$$
\begin{aligned}
AB &= \begin{pmatrix} \alpha_{0,0} & \alpha_{0,1} & \alpha_{0,2} \\ \alpha_{1,0} & \alpha_{1,1} & \alpha_{1,2} \\ \alpha_{2,0} & \alpha_{2,1} & \alpha_{2,2} \end{pmatrix} \begin{pmatrix} \beta_{0,0} & \beta_{0,1} & \beta_{0,2} \\ \beta_{1,0} & \beta_{1,1} & \beta_{1,2} \\ \beta_{2,0} & \beta_{2,1} & \beta_{2,2} \end{pmatrix} \\
&= \begin{pmatrix} \beta_{0,0} \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \alpha_{2,0} \end{pmatrix} + \beta_{1,0} \begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \alpha_{2,1} \end{pmatrix} + \beta_{2,0} \begin{pmatrix} \alpha_{0,2} \\ \alpha_{1,2} \\ \alpha_{2,2} \end{pmatrix} \| \\ \beta_{0,1} \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \alpha_{2,0} \end{pmatrix} + \beta_{1,1} \begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \alpha_{2,1} \end{pmatrix} + \beta_{2,1} \begin{pmatrix} \alpha_{0,2} \\ \alpha_{1,2} \\ \alpha_{2,2} \end{pmatrix} \| \\ \beta_{0,2} \begin{pmatrix} \alpha_{0,0} \\ \alpha_{1,0} \\ \alpha_{2,0} \end{pmatrix} + \beta_{1,2} \begin{pmatrix} \alpha_{0,1} \\ \alpha_{1,1} \\ \alpha_{2,1} \end{pmatrix} + \beta_{2,2} \begin{pmatrix} \alpha_{0,2} \\ \alpha_{1,2} \\ \alpha_{2,2} \end{pmatrix} \end{pmatrix}
\end{aligned}
$$

The resulting $3 \times 3$ matrix consist of the column vectors of the individual matrix vector multiplications.

So the $\gamma_{i,j}$ element of $C$ is just the dot product of the $i^{\text{th}}$ row of $A$ and $j^{\text{th}}$ column of $B$: $\gamma_{i,j} = \alpha_{(i,\ldots)}\beta_{(\ldots,j)}$.

The more interesting question is, why can we do this? Remember, that every linear transformation (LT) can be represented in matrix form.

Let $L_A : \mathbb{R}^k \to \mathbb{R}^m$ and $L_B : \mathbb{R}^n \to \mathbb{R}^k$ be linear transformations and define $L_C(x) = L_A(L_B(x))$. Then:

- It could be easily proven that $L_C : \mathbb{R}^n \to \mathbb{R}^m$ is a linear transformations, always.
- There are $A \in \mathbb{R}^{m\times k}$ and $B \in \mathbb{R}^{k\times n}$, that represent $L_A$ and $L_B$ respectively.
- There is a matrix $C \in \mathbb{R}^{m\times n}$ that represents $L_C(x) = L_A(L_B(x)) = A(Bx)$.

- The operation that computes $C$ from $A$ and $B$ is called a matrix matrix multiplication.
- Notation: $C = AB$ and $Cx = (AB)x = A(Bx)$.
- The number of rows in $A$ must be same as the number of rows in $C$.
- The number of columns in $B$ must be same as the number of columns in $C$.
- The number of columns in $A$ must be same as the number of rows in $B$.
- The cost of $C = AB$ is $2m \times n \times k$.

All scalar-scalar, scalar-vector, vector-scalar, vector-vector (dot-product), outer-product (see below), matrix-vector, vector-matrix multiplication are just special cases of the above described matrix matrix multiplication, see Figure 4.1.

**Outer product**: let $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. Then the outer product of $x$ and $y$ is $xy^T$, which yields a $m \times n$ matrix, with each element being the corresponding product of the components of $x$ and $y$, see Figure 4.1.

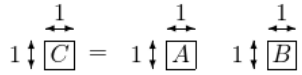
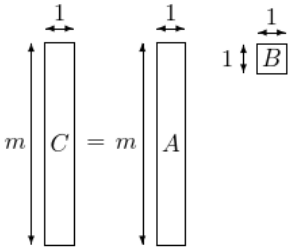
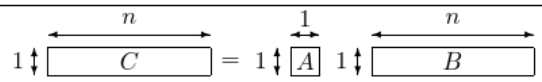
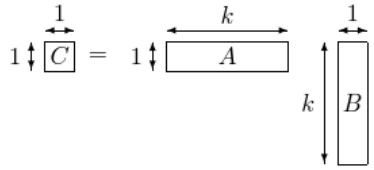
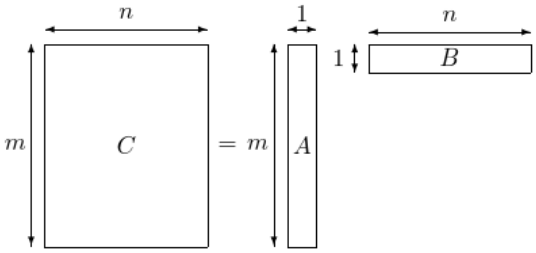
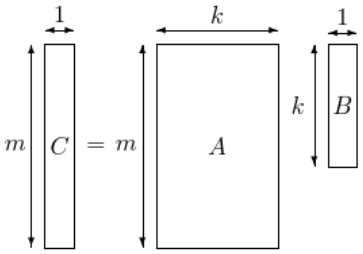
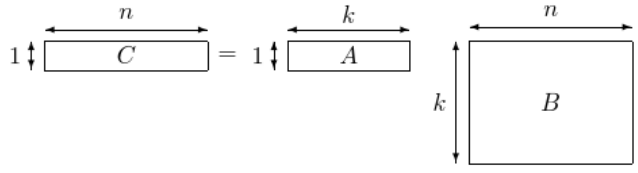
| $m$ | $n$ | $k$ | Shape | Comment |
|---|---|---|---|---|
| 1 | 1 | 1 |  | Scalar multiplication |
| $m$ | 1 | 1 |  | Vector times scalar = scalar times vector |
| 1 | $n$ | 1 |  | Scalar times row vector |
| 1 | 1 | $k$ |  | Dot product (with row and column) |
| $m$ | $n$ | 1 |  | Outer product |
| $m$ | 1 | $k$ |  | Matrix-vector multiplication |
| 1 | $n$ | $k$ |  | Row vector times matrix multiply |

Figure 4.1: Types of matrix-matrix multiplication

## 4.4 Special properties of matrix multiplication

Matrix matrix multiplication **does not always commute**. But if $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k}$ and $C \in \mathbb{R}^{k \times l}$, then

- $(AB)C = A(BC)$
- $A(B + C) = AB + AC$
- $(A + B)C = AC + BC$
- $(AB)^T = B^T A^T$
- $(ABC)^T = C^T B^T A^T$

**always**.

If $A \in \mathbb{R}^{m \times n}$ and $I$ be the appropriate sized identity matrix, then $AI = IA = A$ **always**.

If $A \in \mathbb{R}^{m \times n}$ and $D$ is the diagonal matrix with diagonal elements $\delta_0, \delta_1 ... \delta_{m-1}$, where we partition $A$ by its columns as
$A = (a_0|a_1|...|a_{n-1})$
and by its rows like:

$$A = \begin{pmatrix} \tilde{a}_0^T \\ \tilde{a}_1^T \\ \vdots \\ \tilde{a}_{m-1}^T \end{pmatrix}$$

then

- $AD = (\delta_0 a_0|\delta_1 a_1|...|\delta n - 1 a_{n-1})$
- $$DA = \begin{pmatrix} \delta_0 \tilde{a}_0^T \\ \delta_1 \tilde{a}_1^T \\ \vdots \\ \delta_{m-1} \tilde{a}_{m-1}^T \end{pmatrix}$$

**always**.

Let $A, B \in \mathbb{R}^{n \times n}$ are both upper triangular matrices, then $AB$ is upper triangular as well. The same holds for lower triangular matrices.

Let $A \in \mathbb{R}^{m \times n}$, then $A^T A$ and $AA^T$ are both a symmetric matrix. Furthermore let $A, B \in \mathbb{R}^{n \times n}$, be symmetric matrices. $AB$ then is not necessarily symmetric.

Let $x \in \mathbb{R}^m$. Then $xx^T$ is symmetric.

# Part II

# Week 5-8

# 5 | Matrix-matrix multiplication

## 5.1 Multiple rotations in one matrix

Remember that we can rotate any vector $x \in \mathbb{R}^2$ by angle $\theta$, if we multiply it with the following LT matrix:

$$\begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}$$

If we would then further rotate $x$ by $\rho$ we would end up applying another linear transformation to the vector, which is (as we've seen in week 4) is just a matrix matrix multiplication:

$$\begin{pmatrix} cos(\rho) & -sin(\rho) \\ sin(\rho) & cos(\rho) \end{pmatrix} \left( \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} \right)$$

Notice that we could have just rotate by $\theta + \rho$:

$$\begin{pmatrix} cos(\theta + \rho) & -sin(\theta + \rho) \\ sin(\theta + \rho) & cos(\theta + \rho) \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix}$$

If we instead do the matrix matrix multiplication, we derive the law of sines and cosines:

$$cos(\theta + \rho) = cos(\rho)cos(\theta) - sin(\rho)sin(\theta)$$
$$sin(\theta + \rho) = cos(\rho)sin(\theta) + sin(\rho)cos(\theta)$$

So if we ever forget the law of cosines and sines, and we don't have access to the internet and if remember the matrix of the LT we need to rotate a 2D vector, then we can *simply* derive it. How useful is that.

## 5.2 Computational aspepcts of matrix-matrix multiplication

Computationally, a matrix-matrix multiplication consists of three nested for loops:

- looping over the rows of A
  - looping over the columns of B
    - looping over the row of A and column of B to calculate their dot-product

This three for loops could be ordered in $3! = 6$ ways all together, leading to different implementations of the matrix-matrix multiplication and different intuitions about what this operation represents. Having the inner-most loop as the top one, makes the algorithm perform rank-1 updates on the resulting $C = AB$ matrix, where each cell gets iteratively computed from sums of multiplications between the appropriate cells of $A$ and $B$.

To understand how matrix-matrix multiplication could be optimized, we need to learn a bit about CPU architecture and L1, L2 caches. This is all nicely explained in the enrichment chapter of week 5's lecture notes. The bottom line is, that if we slice and dice cleverly and make sure to utilize as much of the L1 and L2 cache as possible, we achieve a matrix-matrix multiplication routine, that is around 90% efficient, compared to the naive implementation, which achieves 2-3 %.

# 6 | Gaussian elimination

## 6.1 Gaussian elimination

Solving systems of linear equations could be automated or greatly sped up using linear algebra. It relies on three facts@

- Equations could be reordered.
- Equations of a system could be modified by subtracting a multiple of another equation of the system from it.
- Both sides of the equations in the system could be scaled by a non-zero scalar.

We do these manipulations to eventually reduce the system to an upper-triangular system which is easier to solve. For example to following system of linear equations:

$$\begin{aligned} 2\chi_0 + 4\chi_1 - 2\chi_2 &= -10 \\ 4\chi_0 - 2\chi_1 + 6\chi_2 &= 20 \\ 6\chi_0 - 4\chi_1 + 2\chi_2 &= 18 \end{aligned}$$

could be reduced to:

$$\begin{aligned} 2\chi_0 + 4\chi_1 - 2\chi_2 &= -10 \\ -10\chi_1 + 10\chi_2 &= 40 \\ -8\chi_2 &= -16 \end{aligned}$$

by

- row 2 - 2× row 1
- row 3 - 3× row 1
- row 3 - 1.6× row 2

Then using **back-substitution**, we solve for $\chi_2$ in 3rd equation. Then use that in 2nd equation to solve for $\chi_1$. Finally we solve for $\chi_0$ in the 1st equation.

It is a lot easier to represent these systems however, with appended matrices instead of many equations (GE is Gaussian Elimination):

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 4 & -2 & 6 & 20 \\ 6 & -4 & 2 & 18 \end{array}\right) \xrightarrow{GE} \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & & -8 & -16 \end{array}\right)$$

**Gauss transform:** we can cast the problem of finding the appropriate scalars for the deduction of one equation from the other, as a matrix-matrix multiplication. Notice that the matrix we use, is the identity matrix, extended with the scalars we computed before. Following on with the example above, we could have arrived at the same upper triangular matrix by:

$$\begin{pmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ -3 & 0 & 1 \end{pmatrix} \times \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ 4 & -2 & 6 & 20 \\ 6 & -4 & 2 & 18 \end{array}\right)$$

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -1.6 & 1 \end{pmatrix} \times \left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & -16 & 8 & 48 \end{array}\right)$$

$$\left(\begin{array}{ccc|c} 2 & 4 & -2 & -10 \\ & -10 & 10 & 40 \\ & & -8 & 16 \end{array}\right)$$

The great thing about this, is that we can only work with the left-hand side of the appended matrix, and apply our Gauss transforms onto those. Then if we store these Gauss matrices, we can separately apply them to the right-hand side of the equations, *a posteriori*, which is called the **forward substitution**. This is really handy when a new set of right-hand side values come along, because we don't have to recompute all the Gauss matrices, but simply perform a few matrix-vector multiplications.

All of these steps could be combined into a neat algorithm that solves a system of linear equations. Homework 6.2.5.1 summarises that algorithm.

**Solve it with Python:** as an example, solving the following system of linear equations:

$$\left(\begin{array}{ccc|c} -1 & 2 & -3 & 2 \\ -2 & 2 & -8 & 10 \\ 2 & -6 & 6 & -2 \end{array}\right)$$

in Python, requires only four lines of code:

```
import numpy as np
a = np.array([[-1,2,-3], [-2,2,-8],[2,-6,6]])
b = np.array([2,10,-2])
x = np.linalg.solve(a,b)
```

## 6.2 LU Factorisation

If certain conditions are satisfied, then the $Ax = b$ problem could be solved with the LU factorisation, so there exist:

- unit lower triangular matrix $L \in \mathbb{R}^{n \times n}$ , and
- upper triangular matrix $U \in \mathbb{R}^{n \times n}$,

such that

$$A = LU$$

We can work this out just by using slicing and dicing:

$$A \to \left(\begin{array}{c|c} \alpha_{11} & a_{12}^T \\ \hline a_{21} & A_{22} \end{array}\right), L \to \left(\begin{array}{c|c} 1 & 0 \\ \hline l_{21} & L_{22} \end{array}\right), U \to \left(\begin{array}{c|c} \upsilon_{11} & u_{12}^T \\ \hline 0 & U_{22} \end{array}\right)$$

So following the rules of matrix-matrix multiplication (and getting rid off some 0s and 1s), $A = LU$, becomes:

$$\left(\begin{array}{c|c} \alpha_{11} = \upsilon_{11} & a_{12}^T = u_{12}^T \\ \hline a_{21} = l_{21}\upsilon_{11} & A_{22} = l_{21}u_{12}^T + L_{22}U_{22} \end{array}\right).$$

We are getting there. So we can start to see an iterative algorithm emerging, where we move along the diagonal, then section $A$ as follows:

$$A \rightarrow \left(\begin{array}{c|c} A_{TL} & A_{TR} \\ \hline A_{BL} & A_{BR} \end{array}\right),$$

then compute the values of $LU$ as:

$$\left(\begin{array}{c|c} \upsilon_{11} \leftarrow \alpha_{11} & u_{12}^T \leftarrow a_{12}^T \\ \hline l_{21} \leftarrow a_{21}/\upsilon_{11} & L_{22}U_{22} \leftarrow A_{22} - l_{21}u_{12}^T \end{array}\right).$$

So we effectively

- leave the row on the right from the diagonal ($u_{12}^T$) element the same,
- we divide the column below the diagonal element ($a_{21}$), by the diagonal element ($\upsilon_{11}$),
- and finally we do a rank-1 update on the sub-matrix ($A_{BR}$) by subtracting the outer product of $l_{21}$ and $u_{12}^T$ from it.

This is effectively the same algorithm we end up with for the Gaussian elimination. As an example, the $LU$ factorisation of the following matrix:

$$A = \begin{bmatrix} 1 & -2 & 2 \\ 5 & -15 & 8 \\ -2 & -11 & -11 \end{bmatrix}$$

is firstly:

$$\left(\begin{array}{c|cc} 1 & -2 & 2 \\ \hline 5 & -15 & 8 \\ -2 & -11 & -11 \end{array}\right) \rightarrow \left(\begin{array}{c|cc} 1 & -2 & 2 \\ \hline 5 & -5 & -2 \\ -2 & -15 & -7 \end{array}\right)$$

then:

$$\left(\begin{array}{cc|c} 1 & -2 & 2 \\ 5 & -5 & -2 \\ \hline -2 & -15 & -7 \end{array}\right) \rightarrow \left(\begin{array}{cc|c} 1 & -2 & 2 \\ 5 & -5 & -2 \\ \hline -2 & 3 & 1 \end{array}\right)$$

So:

$$L \leftarrow \begin{pmatrix} 1 & 0 & 0 \\ 5 & 1 & 0 \\ -2 & 3 & 1 \end{pmatrix}, U \leftarrow \begin{pmatrix} 1 & -2 & 2 \\ 0 & -5 & -2 \\ 0 & 0 & 1 \end{pmatrix}$$

So we have $A = LU$, but we want to solve $Ax = b$. So

- we just use the above algorithm te get $L$ and $U$,
- then $(LU)x = b$, so $L(Ux) = b$, where we will call $Ux = z$
- so $Lz = b$, which we can solve easily for $z$, since $L$ is a lower triangular matrix, and we can just use the forward substitution algorithm for this.
- Finally, we substitute back to get $Ux = z$, using the back substitution algorithm we have seen before in the Gaussian elimination.

# 7 | More on Gaussian elimination and matrix inversion

## 7.1 When does Gaussian elimination work?

Gaussian elimination and the algorithm we derived for it might not work always, as we might have to divide by zero. This can happen if one of the diagonal elements of the original matrix is zero, or they become zero as the algorithm iterates over the diagonal. But even if the Gaussian eliminates finishes, how can we know for sure that we have a unique solution?

- Let $L \in \mathbb{R}^{n \times n}$ be a unit lower triangular matrix, $Lx = b$ **always** has a unique solution (where $x$ is unknown and $b$ is given).
- Let $U \in \mathbb{R}^{n \times n}$ be an upper triangular matrix with no zeros on its diagonal, $Ux = b$ **always** has a unique solution (where $x$ is unknown and $b$ is given).

And consequently, if $A \in \mathbb{R}^{n \times n}$ and the Gaussian elimination / the LU factorisation finishes and produces an upper triangular matrix which doesn't have any zeros on its diagonal, then we could be sure that there is a unique solution to $Ax = b$.

## 7.2 When Gaussian elimination does *not* work?

Sometimes, the LU factorisation cannot run fully as it will encounter a zero on the diagonal of the matrix $A$. This however does not mean that there isn't a solution to the system of equations, but instead it just shows that the LU factorisation might break down sometimes if the equations are not in the "right" order for example.

A permutation matrix is matrix with all of its elements set to zero, except for one in each row and column. which are set to one.

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, A = \begin{pmatrix} 2 & 1 & 3 \\ 3 & 2 & 4 \\ 1 & 0 & 2 \end{pmatrix}, PA = \begin{pmatrix} 3 & 2 & 4 \\ 1 & 0 & 2 \\ 2 & 1 & 3 \end{pmatrix}$$

Notice, how the permutation matrix $P$, reordered the rows of $A$. This is because we multiplied with it from the left. If instead we multiply from the right:

$$AP = \begin{pmatrix} 3 & 2 & 1 \\ 4 & 3 & 2 \\ 2 & 1 & 0 \end{pmatrix}$$

we reorder by column.

Additionally, a pivot matrix is an identity matrix, with its first and $\pi^{\text{th}}$ column swapped:

$$\tilde{P}(\pi) = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix},$$

where $\pi = 3$.

Using pivoting we can help the LU factorisation to reach to the end of the diagonal and don't terminate with a "cannot divide by zero" error. So as an example, let's say:

$$A = \begin{pmatrix} 2 & 4 & -2 \\ 4 & 8 & 6 \\ 6 & -4 & 2 \end{pmatrix}$$

As we procede with the LU factorisation, we compute the dividing factors, as $l_{21} \leftarrow a_{21}/\upsilon_{11}$ which is $(4, 6)^T/2$, then we negate them $(-2, -3)$ and store them below the diagonal:

$$\left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline -2 & 8 & 6 \\ -3 & -4 & 2 \end{array}\right)$$

Now, when we perform the rank 1 update on the remainder sub-matrix $L_{22}U_{22} \leftarrow$

$A_{22} - l_{21}u_{12}^T$, we get:

$$\left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline -2 & 0 & 10 \\ -3 & -16 & 8 \end{array}\right)$$

If we were to continue with the original LU factorisation algorithm, we would have to divide by zero. To avoid that, we use $\tilde{P}(1) \in \mathbb{R}^{2\times 2}$, to swap the 3rd and 2nd rows. We need to embed this $2 \times 2$ matrix into the $3 \times 3$ identity matrix however, so we get:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix} \times \left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline -2 & 0 & 10 \\ -3 & -16 & 8 \end{array}\right) = \left(\begin{array}{c|cc} 2 & 4 & -2 \\ \hline -3 & -16 & 8 \\ -2 & 0 & 10 \end{array}\right)$$

Notice how the multipliers $(-2, -3)$, swapped with the rows! Now we are effectively done, because the Gauss transform is $0/-16 = 0$, and so $10 - 0 \times 8 = 0$. So the LU factorisation finished, with:

$$\left(\begin{array}{cc|c} 2 & 4 & -2 \\ -3 & -16 & 8 \\ \hline -2 & 0 & 10 \end{array}\right), L = \begin{pmatrix} 0 & 0 & 0 \\ -3 & 0 & 0 \\ -2 & 0 & 0 \end{pmatrix}, U = \begin{pmatrix} 2 & 4 & -2 \\ 0 & -16 & 8 \\ 0 & 0 & 10 \end{pmatrix}.$$

So with partial pivoting we can modify the LU factorisation to eventually get $PA = LU$, where $P$ is the matrix product of all the pivot matrices we had to apply to $A$ while performing the LU factorisation. Then, we can update $b \leftarrow Pb$, and consequently can use the regular forward substitution ($Lz = Pb$) and backward substitution ($Ux = z$) routines to get $x$.

It could still be the case however that there are no rows in $A$ **below the diagonal** with which we could swap the given row/diagonal element, and consequently even with partial pivoting we cannot avoid to divide by zero, so we cannot factorise $A$ into $L$ and $U$. In these cases we will need to use the inverse of $A$ to compute $Ax = b$.

The $U$ matrix we get with the Gaussian elimination is said to be in **row echelon form**. This means that all non-zero rows are above any rows of all zeroes, and the leading coefficient of a non-zero row is always strictly to the right of the leading coefficient of the row above it.

**The cost of LU factorisation** is:

- $\frac{2}{3}n^3$ flops for $A = LU$,

- $n^2$ flops for $Lz = Pb$,
- $n^2$ flops for $Ux = z$.

## 7.3 Inverse of matrices

As a quick recap:

- if $f : \mathbb{R} \to \mathbb{R}$
- and $f$ is a bijection meaning there is a one-to-one relationship between the two domains
- then $f(x) = y$ has a unique solution for all $y \in \mathbb{R}$
- the function that maps $y$ to $x$ as $f(x) = y$ is called the inverse of $f$, denoted as $f^{-1} : \mathbb{R} \to \mathbb{R}$
- just as $\frac{1}{x} \times x = x$, $f(f^{-1}(x)) = x$ and $f^{-1}(f(x)) = x$.

Let $L : \mathbb{R}^{\times} \to \mathbb{R}^{\times}$ be a linear transformation, and let $A$ be the matrix tat represents $L$. If there exists a matrix $B$ such that $AB = I$, then $L$ has an inverse, $L^{-1}$, and $B$ equals the matrix that represents that linear transformation.

In other words, $L$ has an inverse of and on if there exists a matrix $B$ such that $AB = I$. This matrix $B$ is the inverse of $A$, denoted by $A^{-1}$.

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

A matrix is said to *invertible* or *nonsingular* if its inverse $A^{-1}$ exists.

The following statements are all equivalent about $A \in \mathbb{R}^{n \times n}$:

- $A$ is nonsingular.
- $A$ is invertible.
- $A^{-1}$ exists.
- $AA^{-1} = A^{-1}A = I$.
- $A$ represent a linear transformation that is a bijection.

- $Ax = b$ has a unique solution for all $B \in \mathbb{R}^n$.
- $Ax = e_j$ has a solution for all $j \in 0, ...n-1$.
- The determinant of $A$ is nonzero: $det(A)neq0$.

Only square matrices could be inverted.

The **inverse of a diagonal matrix** is just all the elements inverted on the diagonal:

$$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1/3 \end{pmatrix}^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ 0 & 1/2 & 0 \\ 0 & 0 & 3 \end{pmatrix}$$

The **inverse of a Gauss transform** could be derived by negating the values below the diagonal:

$$\begin{pmatrix} -1 & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix}^{-1} = \begin{pmatrix} -1 & 0 & 0 \\ -1 & 1 & 0 \\ 2 & 0 & 1 \end{pmatrix}$$

Using this observation, we can derive how the Gaussian elimination in fact results in the same result as the LU factorisation. For an explanation see p. 316 in the LAFF notes.

The **inverse of a permutation matrix** is the transpose of it:

$$\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{-1} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}^{T} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

The **inverse of the rotating linear transformation** $R(\theta)$ is:

$$R(\theta) = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix}^{-1} = \begin{pmatrix} cos(-\theta) & -sin(-\theta) \\ sin(-\theta) & cos(-\theta) \end{pmatrix}$$

And also from trigonometry we know that $cos(-\theta) = cos(\theta)$, while $sin(-\theta) = -sin(\theta)$, so:

$$R(\theta)^{-1} = \begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix}$$

In a more general case when we want to solve $AB = I$:

$$AB = I = A(b_o|b_1 \dots |b_{n-1}) = (e_o|e_1 \dots |e_{n-1}),$$

so for each column of $B$ we want to solve $AB_j = e_j$, which is essentially $Ax = b$.

The **inverse of** $2 \times 2$ **matrix** $A$:

$$\left(\begin{array}{c|c} \alpha_{0,0} & \alpha_{0,1} \\ \hline \alpha_{1,0} & \alpha_{1,1} \end{array}\right)^{-1} = \frac{1}{\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}} \left(\begin{array}{c|c} \alpha_{1,1} & -\alpha_{0,1} \\ \hline -\alpha_{1,0} & \alpha_{0,0} \end{array}\right),$$

where $\alpha_{0,0}\alpha_{1,1} - \alpha_{1,0}\alpha_{0,1}$ is the **determinant** of the matrix. This method only works if the determinant is not zero.

Finally some properties of matrix inverses, given $A, B$ and $C$ are invertible and are of appropriate size:

- $(AB)^{-1} = B^{-1}A^{-1}$
- $(ABC)^{-1} = C^{-1}B^{-1}A^{-1}$
- $(A^T)^{-1} = (A^{-1})^T$

# 8 | More on matrix inversion

Let $BA = C$ and $B$ be nonsingular. Then it could be proven that $A$ is nonsingular if and only if $C$ is nonsingular.

This also means, that if we try to perform an LU factorisation with row pivoting on matrix $A$, and it fails at some element of the diagonal, (because there isn't any rows below that element which are non-zero), then $A$ is singular and cannot be inverted.

## 8.1 Gauss-Jordan elimination

To solve $Ax = b$ with Gauess-Jordan elimination, we do everything as we would with the Gaussian elimination, but we not only reduce $A$ to an upper triangular matrix, but go further in each step. Using the rules introduced in Chapter 6.1, we reduce all column elements above *and* below the diagonal to 0. For example:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \times \left(\begin{array}{rrr|r} -2 & 2 & -5 & -7 \\ 2 & -3 & 7 & 11 \\ -4 & 3 & -7 & -9 \end{array}\right) \xrightarrow{r2-(-1)r1,r3-(2)r1} \left(\begin{array}{rrr|r} -2 & 2 & -5 & -7 \\ & -1 & 2 & 4 \\ & -1 & 3 & 5 \end{array}\right)$$

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \times \left(\begin{array}{rrr|r} -2 & 2 & -5 & -7 \\ & -1 & 2 & 4 \\ & -1 & 3 & 5 \end{array}\right) \xrightarrow{r1-(-2)r2,r3-(1)r2} \left(\begin{array}{rrr|r} -2 & & -1 & 1 \\ & -1 & 2 & 4 \\ & & 1 & 1 \end{array}\right)$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} \times \left(\begin{array}{rrr|r} -2 & & -1 & 1 \\ & -1 & 2 & 4 \\ & & 1 & 1 \end{array}\right) \xrightarrow{r1-(-1)r3,r2-(2)r3} \left(\begin{array}{rrr|r} -2 & & & 2 \\ & -1 & & 2 \\ & & 1 & 1 \end{array}\right)$$

$$\begin{pmatrix} -\frac{1}{2} & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{array}{ccc|c} & -2 & & 2 \\ & & -1 & 2 \\ & & 1 & 1 \end{array} \xrightarrow{r1-(-1)r3,r2-(2)r3} \begin{array}{ccc|c} \chi_0 & & & -1 \\ & \chi_1 & & -2 \\ & & \chi_2 & 1 \end{array}$$

We can do the exact same thing if we have multiple right sides:

$$\begin{array}{ccc|cc} -2 & 2 & -5 & -7 & 8 \\ 2 & -3 & 7 & 11 & -13 \\ -4 & 3 & -5 & -9 & 9 \end{array} \xrightarrow{Gauss-Jordan} \begin{array}{ccc|cc} \chi_0 & & & -1 & 2 \\ & \chi_1 & & -2 & 1 \\ & & \chi_2 & 1 & -2 \end{array}$$

## 8.2 Gauss-Jordan inverse of a matrix

Using this method above we can compute the inverse of a matrix by reiterating what we've established in Chapter 7:

$$AX = I$$

$$A(x_o|x_1 \ldots |x_{n-1}) = (e_o|e_1 \ldots |e_{n-1})$$

$$Ax_j = e_j$$

So we can solve $AX$ with $I$:

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{pmatrix} \times \begin{array}{ccc||ccc} -2 & 2 & -5 & 1 & 0 & 0 \\ 2 & -3 & 7 & 0 & 1 & 0 \\ -4 & 3 & -7 & 1 & 0 & 1 \end{array} = \begin{array}{ccc||ccc} -2 & 2 & -5 & 1 & 0 & 0 \\ 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & -1 & 3 & -2 & 0 & 1 \end{array}$$

$$\begin{pmatrix} 1 & 2 & 0 \\ 0 & 1 & 0 \\ 0 & -1 & 1 \end{pmatrix} \times \begin{array}{ccc||ccc} -2 & 2 & -5 & 1 & 0 & 0 \\ 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & -1 & 3 & -2 & 0 & 1 \end{array} = \begin{array}{ccc||ccc} -2 & 0 & -1 & 3 & 2 & 0 \\ 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array}$$

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{pmatrix} \times \begin{array}{ccc||ccc} -2 & 0 & -1 & 3 & 2 & 0 \\ 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array} = \begin{array}{ccc||ccc} -2 & 0 & 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 7 & 3 & -2 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array}$$

$$\left(\begin{array}{ccc} -\frac{1}{2} & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{array}\right) \times \left(\begin{array}{ccc|ccc} -2 & 0 & 0 & 0 & 1 & 1 \\ 0 & -1 & 0 & 7 & 3 & -2 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array}\right) = \left(\begin{array}{ccc|ccc} 1 & 0 & 0 & 0 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & 1 & 0 & -7 & -3 & 2 \\ 0 & 0 & 1 & -3 & -1 & 1 \end{array}\right)$$

To check our results, $AX = I$:

$$\begin{pmatrix} -2 & 2 & -5 \\ 2 & -3 & 7 \\ -4 & 3 & -7 \end{pmatrix} \times \begin{pmatrix} 0 & -\frac{1}{2} & -\frac{1}{2} \\ -7 & -3 & 2 \\ -3 & -1 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

The last scaling step where we make the left hand side the identity matrix, could be incorporated into the algorithm so it's not needed any more:

$$\left(\begin{array}{ccc} -\frac{1}{2} & 0 & 0 \\ 1 & 1 & 0 \\ -2 & 0 & 1 \end{array}\right) \times \left(\begin{array}{c|cc|ccc} -2 & 2 & -5 & 1 & 0 & 0 \\ \hline 2 & -3 & 7 & 0 & 1 & 0 \\ -4 & 3 & -7 & 1 & 0 & 1 \end{array}\right) = \left(\begin{array}{c|cc|ccc} \mathbf{1} & -1 & -\frac{5}{2} & \frac{1}{2} & 0 & 0 \\ \hline 0 & -1 & 2 & 1 & 1 & 0 \\ 0 & -1 & 3 & -2 & 0 & 1 \end{array}\right)$$

This made the top left item of the resulting matrix a 1 already. If we continue like this we don't have to do the scaling, and we end up with a matrix which is in **reduced row echelon form**. This means the matrix is in row echelon form and every leading coefficient (the first you encounter when you read a row from left to right) is one, and is the only non-zero entry in its column.

**The cost of matrix inversion is** $2n^3$. Since solving $Ax = b$ with matrix inversion, costs $2n^3 + 2n^2$ flops, and doing it by LU factorisation costs $\frac{2}{3}n^3 + 2n^2$, we should almost **never** invert a matrix for solving such a system as the difference is exactly 3-fold.

Matrix is inversion is incredibly useful in other parts of statistics (like when we want to compute the precision matrix, which is the inverse of the covariance matrix), but not for solving $Ax = b$. The precision matrix $AB^{-1}$ holds all the partial correlations between each variable in $A$ and $B$.

## 8.3 Cholesky factorization

The covariance matrix is symmetric positive definite (SPD) and consequently invertible. Let $A \in \mathbb{R}^{n \times n}$. Matrix $A$ is said to be SPD if:

- $A$ is symmetric; and
- $x^T Ax > 0$ for all nonzero vectors $x \in \mathbb{R}^n$.

SPD matrices are useful in an overdetermined system $Bx = y$ where $B \in \mathbb{R}^{m \times n}$ and $m > n$. In other words, when there are more equations than there are unknowns in our linear system of equations.

When $B$ has linearly independent columns, the best solution to $Bx = y$ satisfies $B^T Bx = B^T y$. If we set $A = B^T B$ and $b = B^T y$, then we need to solve $Ax = b$, and now A is square and nonsingular (which we will prove later in the course).

Now, we could solve Ax = b via any of the methods we have discussed so far. However, these methods ignore the fact that A is symmetric.

By taking advantage of symmetry, we can factor A akin to how we computed the LU factorization, but at roughly half the computational cost. This new factorization is known as the Cholesky factorization.

Let $A \in \mathbb{R}^{n \times n}$ be a SPD matrix. Then there exists a lower triangular matrix $L \in \mathbb{R}^{n \times n}$ such that $A = LL^T$. If the diagonal elements of $L$ are chosen to be positive, this factorization is unique.

# Part III

# Week 9-12

# 9 | Vector spaces

We can use the previously learnt LU factorisation for instance, to fit a polynomial. In Figure 9.1 we have a simple problem, with three points to fit: $P = \{(-2,-1),(0,2),(2,3)\}$. We can fit a polynomial of the form $p(\chi) = \gamma_0 + \gamma_1\chi + \gamma_2\chi^2$ to these points by simply solving the following system of equations:

$$\left(\begin{array}{ccc|c} 1 & -2 & -2^2 & -1 \\ 1 & 0 & 0^2 & 2 \\ 1 & 2 & 2^2 & 3 \end{array}\right) \xrightarrow{Gauss-Jordan} \begin{pmatrix} \gamma_0 \\ \gamma_1 \\ \gamma_2 \end{pmatrix} = \begin{pmatrix} 2 \\ 1 \\ -\frac{1}{4} \end{pmatrix}$$

We can see this resulting polynomial plotted on the right in purple. Alternatively, we can think of this problem as finding the linear combination of the blue ($p_0(x) = \gamma_0$), green ($p_1(x) = \gamma_1\chi$) and red ($p_2(x) = \gamma_2\chi^2$) parent functions which will all intersect three points in $P$.

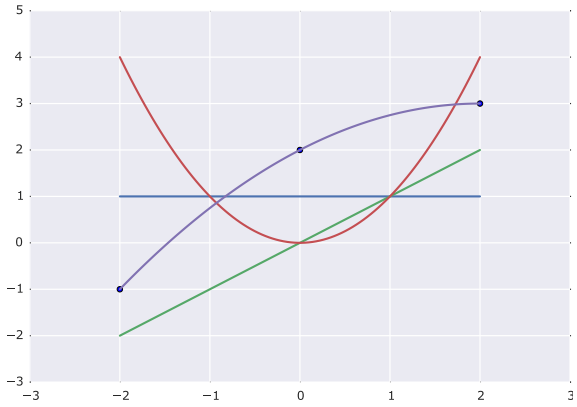



Figure 9.1: Polynomial fitting.

Since computers are not capable of thinking about continuous functions, we discretize these parent functions at the three $x$ points we have in $P$, $Px = \{-2, 0, 2\}$, so we end up with three vectors with their values at these points:

$$p_0(Px) = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, p_1(Px) = \begin{pmatrix} -2 \\ 0 \\ 2 \end{pmatrix}, p_0(Px) = \begin{pmatrix} 4 \\ 0 \\ 4 \end{pmatrix}.$$

Eventually we get the exact same system to solve as we had before. If we had a fourth point we might not get a solution with a 3 degree polynomial e.g. $(1, 4)$. So in general sometimes $Ax = b$ has a unique solution, sometimes it doesn't have a solution at all, and sometimes it has infinite solutions.

## 9.1 When linear systems have no solutions

If the Gauss-Jordan algorithm introduces an inconsistent equation along the way, e.g. $1 = 0$, then we can be sure that the system has no solution.

Conversely if we end up with an under-determined system, because one of the equations end up cancelling all variables in it (e.g. $0 = 0$), then we have an infinite amount of solutions:

$$\left(\begin{array}{ccc|c} -2 & 2 & -2 & 0 \\ 2 & -3 & 4 & 3 \\ 4 & 3 & -2 & 3 \end{array}\right) \xrightarrow{\text{Gauss-Jordan}} \left(\begin{array}{ccc|c} \chi_0 & & \chi_2 & 3 \\ & \chi_1 & -2\chi_2 & -3 \\ & & 0 & 0 \end{array}\right)$$

Since we have more unknowns than equations, we cannot calculate a unique solution obviously. But we can find the **general solution** (also sometimes called the complete solution) to this system:

$$\left(\begin{array}{ccc|c} \chi_0 & & \chi_2 & 3 \\ & \chi_1 & -2\chi_2 & -3 \\ & & 0 & 0 \end{array}\right) \xrightarrow[\text{variable to } \beta]{\text{Set the } \chi_2 \text{ free}} \left(\begin{array}{ccc|c} \chi_0 & & \chi_2 & 3 \\ & \chi_1 & -2\chi_2 & -3 \\ & & \chi_2 & \beta \end{array}\right)$$

$$\left(\begin{array}{ccc|c} \chi_0 & & \chi_2 & 3 \\ & \chi_1 & -2\chi_2 & -3 \\ & & \chi_2 & \beta \end{array}\right) \xrightarrow{\text{Substitute in } \beta} \left(\begin{array}{ccc|c} \chi_0 & & \beta & 3 \\ & \chi_1 & -2\beta & -3 \\ & & \chi_2 & \beta \end{array}\right)$$

$$\left(\begin{array}{ccc|c} \chi_0 & & \beta & 3 \\ & \chi_1 & -2\beta & -3 \\ & & \chi_2 & \beta \end{array}\right) \xrightarrow{\text{Reorder}} \left(\begin{array}{ccc|c} \chi_0 & & & 3-\beta \\ & \chi_1 & & -3+2\beta \\ & & \chi_2 & 0+\beta \end{array}\right)$$

Finally we have the general solution in the form of two vectors:

$$\begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix} = \begin{pmatrix} 3 \\ -3 \\ 0 \end{pmatrix} + \beta \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix}.$$

What is really important that if we plug in the first vector, we get back the original system:

$$\begin{pmatrix} -2 & 2 & -2 \\ 2 & -3 & 4 \\ 4 & 3 & -2 \end{pmatrix} \times \begin{pmatrix} 3 \\ -3 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 3 \\ 3 \end{pmatrix}$$

whereas if we plug in the second we get a zero vector:

$$\begin{pmatrix} -2 & 2 & -2 \\ 2 & -3 & 4 \\ 4 & 3 & -2 \end{pmatrix} \times \beta \begin{pmatrix} -1 \\ 2 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$$

And of course no matter what $\beta$ we choose the result still be a zero vector. So any vector of the form above is a solution to the system, thus we have infinite amount of solutions.

## 9.2 Vector spaces

A vector space is a subset, $S$ of $\mathbb{R}^n$ with the following properties:

- $0 \in S$, meaning the zero vector is in $S$,
- S is closed under addition and scalar multiplication:
  - If $v, w \in S$, then $v + w \in S$,
  - If $\alpha \in S$ and $v \in S$, then $\alpha v \in S$.

As an example the plane of vectors $x = \begin{pmatrix} \chi_0 \\ \chi_1 \\ \chi_2 \end{pmatrix}$ where $\chi_0 = 0$ is a subspace of $\mathbb{R}^3$:

- The null vector is in this subspace: $\chi_1 = 0, \chi_2 = 0 \rightarrow \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$
- If we choose $v = \begin{pmatrix} 0 \\ v_1 \\ v_2 \end{pmatrix}, w = \begin{pmatrix} 0 \\ w_1 \\ w_2 \end{pmatrix}$, then $v + w = \begin{pmatrix} 0 \\ v_1 + w_1 \\ v_2 + w_2 \end{pmatrix}$, which is still in $R^3$.
- If we have $v = \begin{pmatrix} 0 \\ v_1 \\ v_2 \end{pmatrix}$ and $\alpha \in \mathbb{R}$, then $\alpha v = \begin{pmatrix} 0 \\ \alpha v_1 \\ \alpha v_2 \end{pmatrix}$ which is still in $R^3$

So we conclude that $x = \begin{pmatrix} 0 \\ \chi_1 \\ \chi_2 \end{pmatrix}$ is a subspace of $R^3$.

As another example $x = \begin{pmatrix} 1 \\ \chi_1 \\ \chi_2 \end{pmatrix}$ is *not* a subspace of $R^3$, because we simply don't have the null vector in it, no matter what we set $\chi_1$ and $\chi_2$.

Also $\{x|||x||_2 < 0\}$ is not a subspace of $\mathbb{R}^n$, because we can easily find an $\alpha$ with which $\alpha x$ will be longer than 1, and therefore be outside of the defined boundary $< 1$.

## 9.3 Column space and row space

The set $S \subset \mathbb{R}^m$ described by $\{Ax|x \in \mathbb{R}^n\}$, where $A \in \mathbb{R}^{m \times n}$ is a subspace. In other words, the set $S$ holds all vectors that could be acquired by multiplying $A$ by a vector $x$. So $S$ is a subspace because:

- We know the zero vector is in the set ($0 \in S$), because if we pick $x = 0$, then we get the zero vector.
- Also, if we pick two random vectors in this set $v, w \in S$, then we can show that $(v + w) \in S$:
  - For some, $x, y \in \mathbb{R}^n$, $v = Ax$ and $w = Ay$.
  - But then $v + w = Ax + Ay = A(x + y)$, which also must be in $S$.
- Finally if we pick a scalar $\alpha \in \mathbb{R}$ and a random vector $v \in S$, then $\alpha v \in S$:
  - For some $x \in \mathbb{R}^{\kappa}, v = Ax$.
  - But then $\alpha v = \alpha(Ax) = A(\alpha x)$, which also must be in $S$.

The **column space** of $A \in \mathbb{R}^{m \times n}$, equals the set $\{Ax|x \in \mathbb{R}^n\}$. Or in other words, the set of all vectors that could be achieved by multiplying $A$ by $x$. It is denoted by $\mathcal{C}(A)$, and it is essentially all the **linear combinations** we can get by varying $x$. Because of this $\mathcal{A} = Span(a_1, a_2, \ldots, a_n)$, where $a_1, a_2, \ldots, a_n$ are the columns of $A$. Also, the column space is often called the **range of the matrix**.

As a consequence, the problem $Ax = b$ only has a solution when $b \in \mathcal{C}(A)$, i.e. when it is in the column space of the independent variables of $A$. So as another definition we can say, the column space of $A, \mathcal{C}(A)$, describes the subspace of all vectors $b$ for which $Ax = b$ has a solution.

The **row space** is the subspace of all vectors that can be created by taking linear combinations of the rows of a matrix. In other words, the row space of $A$ equals $\mathcal{C}(A^T)$.

## 9.4 Null space

As we have seen in section 9.1, when we were trying to identify *all* possible solutions to $Ax = b$, if $Ax_s = b$ and $Ax_n = 0$, then $x_s + x_n$ is also a solution, because $A(x_s + x_n) = b$. Hence identifying the set pf all vectors such that $Ax = 0$ is really important.

The set of all vectors $x \in \mathbb{R}^n$ that have the property that $Ax = 0$ is called the null space of $A$. It is denoted as $\mathcal{N}(A) = \{x|Ax = 0\}$. We will skip the proof of why this is a proper subspace, as it is pretty intuitive.

$A$'s columns are linearly independent if and only if $\mathcal{N}(A) = \{0\}$, i.e. when the only vector in its null space is the zero vector. This because $Ax = a_1\chi_1 + a_2\chi_2, \ldots, +a_n\chi_n$, where $\{a_1, a_2, \ldots, a_n\}$ are the columns of $A$ and $\{\chi_1, \chi_2, \ldots, \chi_n\}$ are the components of $x$. So basically we just take a linear combination of $A$'s columns and we require it to be zero. As we will see below, this is only possible if those vectors are all linearly independent, meaning, none of them could be rewritten as a combination of the rest. In this case all of $x$'s components have to be zero in order to $Ax$ produce a zero vector.

We can find the null space of a matrix by reducing it to its reduced row echelon form and solving for the free variables. Reduced row echelon form could be achieved by reducing a matrix to its row echelon form, with the added constraints that all leading coefficients have to be one, and these leading entries have to be the only non zero entries in their column.

The null space is often called the **kernel of the matrix**.

## 9.5 Span

The span of vectors $\{v_0, v_1, ..., v_{n-1}\}$ is the set of all vectors that are a linear combination of this set. So for example the span of $\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right\} = \mathbb{R}^2$. This is because we can create every single vector in $\mathbb{R}^2$ from te linear combination of those two unit basis vectors.

So if we look at these vectors as a column of a matrix:

$$V = \{v_0|v_1|...|v_{n-1}\}$$

then $Vx$ is a linear combination of these vectors. The set of all possible vectors arising from $Vx$ is the column space of $V$, $\mathcal{C}(V)$.

So the span of a set of vectors is the column space of the matrix that has those vectors as its columns.

Usually a subspace has infinite number of vectors within it. A spanning set of a subspace is really useful, because using it we can describe, or create all vectors of the subspace, using only a finite set of vectors.

## 9.6 Linear independence

Let $\{v_0, v_1, ..., v_{n-1}\} \in \mathbb{R}^n$. This set of vectors is said to be *linearly independent* if and only if

$$\chi_0 v_0 + \chi_1 v_1, ..., \chi_{n-1} v_{n-1} = 0,$$

could be only achieved by:

$$\chi_0, \chi_1, ..., \chi_{n-1} = 0.$$

A more formal definition: let $\{v_0, v_1, ..., v_{n-1}\} \subset \mathbb{R}^n$, and let $V = \{v_0|v_1|...|v_{n-1}\}$ . Then the vectors $\{v_0, v_1, ..., v_{n-1}\}$ are linearly independent if and only if $\mathcal{N}(V) = \{0\}$. So matrix $V$ can only have the zero vector in its null space.

As an example let's look at the following set of vectors:

$$\left\{ \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix} \right\}.$$

These are linearly dependent, because we can rewrite the 3rd vector as the sum of the first two. Or to use the 2nd definition:

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

So we can find a vector that maps to the null space, which is not the zero vector, so these vectors have to be linearly dependent.

**Columns of the identity matrix and a lower or upper triangular matrix with non-zero diagonal values, are linearly independent.**

As generalisation of the above rule is, let $\{v_0, v_1, ..., v_{n-1}\} \in \mathbb{R}^m$ and $n > m$. Then these vectors must be *linearly dependent*. This is because in such a matrix, we have more columns than rows, which essentially means we have more variables than equations, which inherently means we have a vector in the null space which is not the zero vector. As a consequence, if we have a lower triangular matrix with non-zero diagonal elements, and we add a new row to it, its columns will remain linearly independent.

So we can now add the following statements to the list of section 7.3 about $A \in \mathbb{R}^{n \times n}$:

- $A$ is nonsingular.
- $A$ is invertible.
- LU factorisation with partial pivoting does not break down.
- $\mathcal{C}(A) = \mathbb{R}^n$
- $A$ has linearly independent columns.
- $\mathcal{N}(A) = \{0\}$.

## 9.7 Bases for subspaces

Let $S$ be a subspace of $\mathbb{R}^m$. Then the set $\{v_0, v_1, ..., v_{n-1}\} \subset \mathbb{R}^m$ is said to be a *basis* for $S$ if $\{v_0, v_1, ..., v_{n-1}\}$ are linearly independent, and the span of $\{v_0, v_1, ..., v_{n-1}\}$ is $S$.

In other words, if we have the smallest possible set of linearly independent vectors, from which we can create all vectors of $S$ as a linear combination, then this set is the basis of $S$.

So if we have a set of vectors $\{v_0, v_1, ..., v_{n-1}\} \subset \mathbb{R}^n$, and let $V = \{v_0|v_1|...|v_{n-1}\}$ be invertible, then $\{v_0, v_1, ..., v_{n-1}\}$ form the basis for $\mathbb{R}^n$.

As an example:

$$\left\{ \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \right\}$$

is the basis of $\mathbb{R}^3$. This means, that the matrix:

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

is invertible, so $Ax = b$, always has a solution, as a linear combination of the columns of $A$, which are the unit basis vectors in this case.

Furthermore if we have a set of vectors $S \subset \mathbb{R}^m$, then $S$ can only contain $m$ linearly independent vectors.

## 9.8 Dimension of a subspace

Every non-trivial (meaning more than just a zero vector) subspace $\mathbb{R}^m$ has a basis. This is not unique, as we can multiply the vectors within this basis and get a new set of vectors. However, **the number of vectors within a basis of a subspace is always the same. This is the dimension of the subspace.**

## 9.9 Rank of a matrix

Let $A \in \mathbb{R}^{m \times n}$. The **rank** of $A$, is the number of vectors in a basis for the columns space of $A$. In other words the rank of $A$ is the number of linearly independent columns in $A$. If $rank(A) = n$, then $A$ is said to be **full rank, which also inherently means it is invertible.**

# 10 | Orthogonality and linear least squares

Taking the first function, from our 2 degree polynomial from Chapter 9 again:

$$\left.\begin{array}{ccc|c} 1 & -2 & -2^2 & -1 \\ 1 & 0 & 0^2 & 2 \\ 1 & 2 & 2^2 & 3 \end{array}\right. \quad \rightarrow \quad \chi_0 - 2\chi_1 + 4\chi_2 = -1,$$

we can solve it by calculating first a specific solution by setting $\chi_1, \chi_2$ free variables to zero:

$$X_s = \begin{pmatrix} \chi_0 \\ 0 \\ 0 \end{pmatrix}.$$

We do this arbitrarily because it is easy to compute with zero. The solution then is:

$$X_s = \begin{pmatrix} -1 \\ 0 \\ 0 \end{pmatrix}.$$

What we do next to find a solution that maps to zero so we set the two right hand side of the equation to zero, and pick the free variables $\chi_1, \chi_2$:

$$X_{n_0} = \begin{pmatrix} \chi_0 \\ 1 \\ 0 \end{pmatrix}, X_{n_1} = \begin{pmatrix} \chi_0 \\ 0 \\ 1 \end{pmatrix}.$$

If we do that we end up with two linearly independent vectors that are in the null space, which is a plane in this case:

$$X_{n_0} = \begin{pmatrix} 2 \\ 1 \\ 0 \end{pmatrix}, X_{n_1} = \begin{pmatrix} -4 \\ 0 \\ 1 \end{pmatrix}.$$

The general solution then is:

$$X_{general} = X_s + \alpha X_{n_0} + \beta X_{n_1}.$$

The plane of the null space is shown in Figure 10.1, and the strong black arrow, represents $X_s$. The red vector is $X_{n_1}$ which is about twice as long as the blue one $X_{n_0}$, but they both lie in the red plane of the null space. So if we offset this red plane by the $X_s$ vector, we get the visual representation of the general solution, which is the plane in which all vectors of the free variables $\chi_1, \chi_2$ must lie.

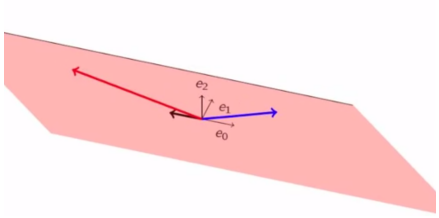



Figure 10.1: Plane of solutions.

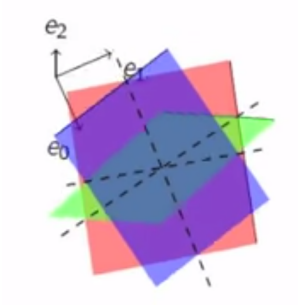



Figure 10.2

If we would orthogonalise $X_{n_0}$ and $X_{n_1}$, the figure would look even nicer, but the main point is, no matter what we plug into the equation to get $X_s$ we will have the same general solution.

Finally, if we would visualise the plane of each equation, we would get Figure 10.2. Here each plane represents the solutions of a given equation. The solutions of just two out of the three equations are represented as dashed lines. As we have three equations we have three of those, and where they intersect is the final solution to the system:

$$X = \begin{pmatrix} 2 \\ 1 \\ -0.25 \end{pmatrix}.$$

## 10.1 Important attributes of a linear system

Let's say we have a linear system that we reduced to row echelon form:

$$A = \left(\begin{array}{cccc|c} 1 & 3 & 1 & 2 & 1 \\ 2 & 6 & 4 & 8 & 3 \\ 0 & 0 & 2 & 4 & 1 \end{array}\right) \xrightarrow[elimination]{Gaussian} A_r = \left(\begin{array}{cccc|c} 1 & 3 & 1 & 2 & 1 \\ 0 & 0 & 2 & 4 & 1. \\ 0 & 0 & 0 & 0 & 0 \end{array}\right)$$

Then, this system has the following properties:

- The **pivots** are the leading coefficients in $\chi_0, \chi_2$: 1 and 2. Consequently $\chi_0$ and $\chi_2$ are the so called **dependent or basic variables**, while the others $\chi_1$ an d$\chi_3$ are the **free variables**.
- A **specific solution** could be found by setting the free variables to 0, and solving for the dependent variables. In this case that works out to be $\chi_0 = \frac{1}{2}, \chi_2 = \frac{1}{2}$.
- The **basis for the null space** could be found by setting the right hand side of the appended system to 0. Then to find the first vector of the basis we set the first free variable to 1, and the second to 0, then vice versa. This way for every variable we can find a vector in the null space.
- A **general solution** could be described by adding a linear combination of the null basis vectors to the specific solution we found earlier: $X_{general} = X_s + \beta_0 X_{n_0} + \beta_1 X_{n_1}$.
- A dimension of the **basis for the column space** equals the number of linearly independent variables in the system. To find the vectors of the basis for $\mathcal{C}(A)$, we simply take the columns of $A$ which have pivots in $A_r$:

$$\begin{pmatrix} 1 \\ 2 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 \\ 4 \\ 2 \end{pmatrix}.$$

- The **basis for the row space** could be found by taking the *transposed* rows of $A_r$ where we have pivots (i.e. the non zero rows for the basis of the row space):

$$\begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 2 \\ 4 \end{pmatrix}.$$

- **The dimension of the column space = the dimension of the row space = the rank of the matrix = $k$ = the number of pivots we have**.
- **The dimension of the null-space = $n - k$**

## 10.2 Orthogonal vectors and spaces

Vectors $x, y \in \mathbb{R}^n$ are orthogonal (perpendicular) if and only if their dot/inner product is zero: $x^T y = 0$.

We can extend this to subspaces. Let $\mathbf{V}$,$\mathbf{W} \subset \mathbb{R}^n$ be subspaces. $\mathbf{V}$ and$\mathbf{W}$ are said to be orthogonal if $v \in \mathbf{V}$ and $w \in \mathbf{W}$ implies that $v^T w = 0$.

In other words if all vectors from $\mathbf{V}$ are orthogonal to all vectors from $\mathbf{W}$, then $\mathbf{V}^{\perp}\mathbf{W}$.

This means that if $\mathbf{V}^{\perp}\mathbf{W}$, then $\mathbf{V} \cap \mathbf{W} = \{0\}$, the zero vector. Also if $\mathbf{V} \in \mathbb{R}^m$ is a subspace, then $\mathbf{V}^{\perp}$ is a subspace.

## 10.3 Fundamental spaces

If we have a matrix $A \in \mathbb{R}^{m \times n}$ that maps vectors from $\mathbb{R}^n$ to $\mathbb{R}^m$, with $r$ pivot points, then there are four fundamental subspaces we need to know about:

- Column space: $\mathcal{C}(A) \subset \mathbb{R}^m$. Its dimension is: $r$.
- Null space: $\mathcal{N}(A) \subset \mathbb{R}^n$. Its dimension is: $n - r$.
- Row space: $\mathcal{R}(A) \subset \mathbb{R}^n$ or $\mathcal{C}(A^T) \subset \mathbb{R}^n$. Its dimension is: $r$.
- Left null space: $\mathcal{N}(A^T) \subset \mathbb{R}^m$. Its dimension is: $m - r$.

The columns space and the left null space are in $\mathbb{R}^m$ so they are on the right hand side in Figure 10.3, whereas the row space and the null space are in $\mathbb{R}^n$ and on the left hand side of the figure.

**Two very important theorems:**

1. Let $A \in \mathbb{R}^{m \times n}$. Then $\mathcal{R}(A) \perp \mathcal{N}(A)$.
    - Taking any two vectors $y \in \mathcal{R}(A)$ and $z \in \mathcal{N}(A)$, $y^T z$ should be zero.
    - We can use the fact if $y \in \mathcal{R}(A)$ it also means $y \in \mathcal{C}(A^T)$. So $y = A^T x$ for some $x \in \mathbb{R}^m$, meaning $y$ is just some linear combination of the columns of $A^T$.

- So $y^T z \to (A^T x)^T z \to x^T A z$
- Since $z$ is in the null space of $A$, then $Az$ must be a zero vector, therefore:
- $x^T A z \to x^T 0 \to 0$.

2. Let $A \in \mathbb{R}^{m \times n}$. Every $x \in \mathbb{R}^n$ can be written as $x = x_r + x_n$, where $x_r \in \mathcal{R}(A)$ and $x_n \in \mathcal{N}(A)$. This is because $dim(\mathcal{R}(A)) = r$ and $dim(\mathcal{N}(A)) = n - r$, and it could be proven that all $r + (n - r) = n$ vectors within their bases are linearly independent. So there is $n$ linearly independent vector in these bases so they must form the basis for $\mathbb{R}^n$.

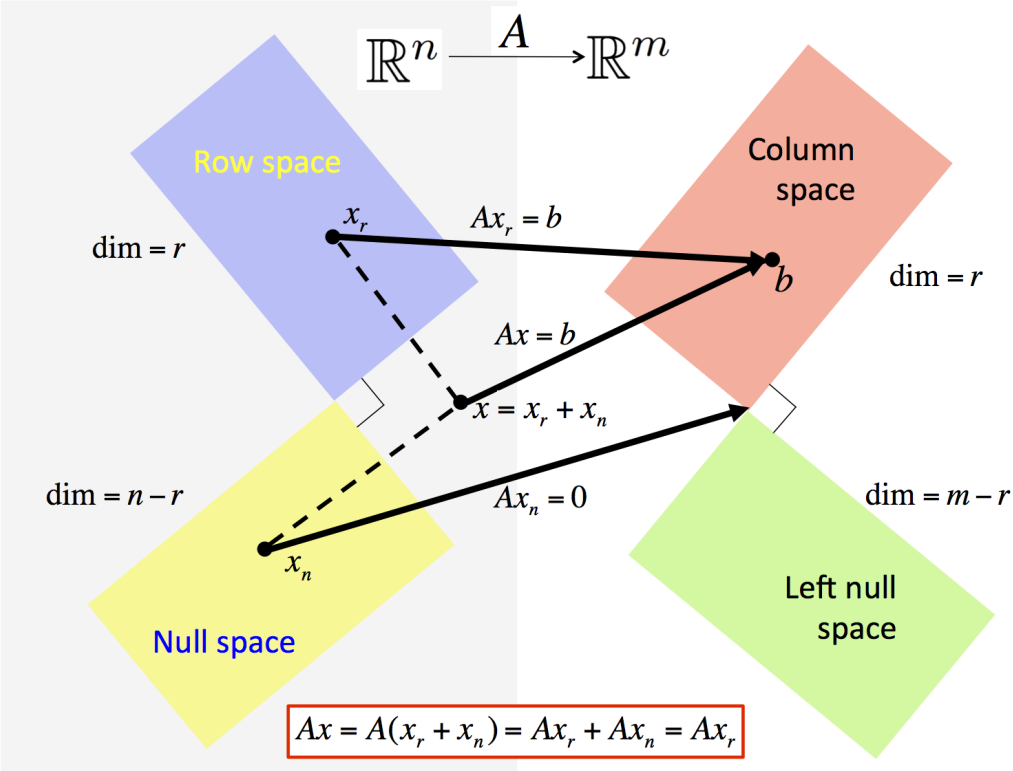



Figure 10.3: Fundamental spaces.

About this figure:

- Matrix $A \in \mathbb{R}^{m \times n}$ that maps vectors from $\mathbb{R}^n$ to $\mathbb{R}^m$.
- To illustrate that, we have a vector in $\mathbb{R}^n$, called $x$.
- $A$ takes $x$ to $b$, which is now in $\mathbb{R}^m$.

- So $b$ must be in the column space of $A$, because we just got from $x$ to $b$ by taking the linear combinations of the columns of $A$.
- There is also a row space and the as the zero vector is part of every subspace it is in both the row space and column space of $A$. It is denoted as little 0 in the bottom right corner of these spaces.
- Furthermore the first theorem proved us that the null space of $A$ has to perpendicular to the row space of $A$, hence the right-angle sign between the two rectangles.
- This orthogonality is also present between the column space of $A$ and the left null space of $A$.
- The second theorem told us that any vector $x \in \mathbb{R}^n$ can be written as a vector in the row space of $A$ plus the vector in the null space of $A$. This denoted as $x = x_r + x_n$.
- We know that the vector in the null space of $A$ is mapped by A to the zero vector. Denoted by the vector $Ax_n = 0$.
- Focusing on the bottom box, and combining the above points, we can see how $Ax_r$ must take us to $b$ as well.
- So **A is a one-to-one mapping between the row space and the column space of A**.

This figure tells us everything about what happens when we perform $Ax = b$, and helps us to come up with an approximation to the real solution, when $Ax = b$ does not have a solution. This is known as the linear least squares.

## 10.4 Linear least squares approximation

As we can see in Figure 10.4, in the real word, we often cannot easily fit a polynomial through our data-points, and more often than not, we would like to fit a simple linear line to our data, to get a sense of the trend. So since $b \notin \mathcal{C}(A)$, we cannot write $Ax = b$, but instead we aim for $Ax \approx b$.

$$\begin{pmatrix} \psi_0 \\ \psi_1 \\ \psi_2 \\ \psi_3 \end{pmatrix} = \begin{pmatrix} 1 & \chi_0 \\ 1 & \chi_1 \\ 1 & \chi_2 \\ 1 & \chi_3 \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \end{pmatrix} \text{ or, } \begin{pmatrix} 1.97 \\ 6.97 \\ 8.89 \\ 10.01 \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} \gamma_0 \\ \gamma_1 \end{pmatrix}$$

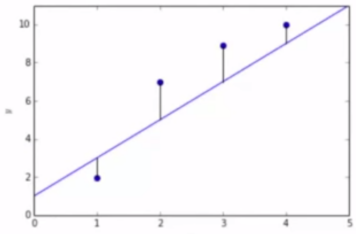


Figure 10.4: Example of a LLS problem.

If we take an even simpler example with just three data-points, and we visualise the column space of $A$ in 3D (which is just a thin purple line in this figure due to the chosen viewing angle), as shown in Figure 10.5, then we clearly see, that $b \notin \mathcal{C}(A)$. But we also see that there is a vector $z$ which is in the column space of $A$. The point $z$ points to within $\mathcal{C}(A)$, is the closest possible solution ($Ax \approx b$) to $b$. We can also see, that $b = z + w$ and that $z \perp w$.

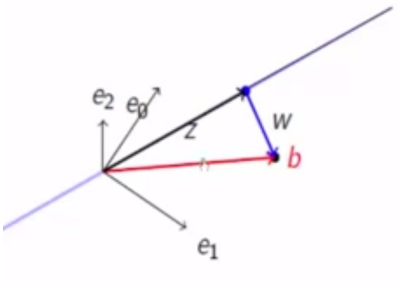



Figure 10.5: $b \notin \mathcal{C}(A)$.

Putting all this together, and using Figure 10.3 as well:

- We want to solve $Ax \approx b$.
- Let $\hat{x}$ be the best solution.
- $b = z + w$, where $z \in \mathcal{C}(A)$ and $w \in \mathcal{C}(A)^{\perp}$.
- We know that we can solve $Ax = b$ for $b \in \mathcal{C}(A)$.
- We also know that the left null space is orthogonal to the column space of $A$, and that the left null space of $A$ is the same as the null space of $A^T$. So we can move around the fundamental spaces as $w \in \mathcal{C}(A)^{\perp} \Rightarrow w \in \mathcal{N}_{left}(A) \Rightarrow w \in \mathcal{N}(A^T)$. This means that $A^T w = 0$.
- Since $w = b - z \rightarrow A^T(b - z) = 0$.

- But notice that $z = A\hat{x}$, i.e. the point in the column space for which we want to compute our best approximate solution. So $A^T(b - A\hat{x}) = 0 \rightarrow A^Tb - A^TA\hat{x} = 0 \rightarrow A^TA\hat{x} = A^Tb$
- With that we derived the **normal equation**: $A^TA\hat{x} = A^Tb$.
- We can solve this as long as $A^TA$ is nonsingular, which is true if $A$ has linearly independent columns.
- So we can solve for $\hat{x}$ by computing $(A^TA)^{-1}A^Tb$. But we should almost never invert a matrix, so it is better to calculate $A^TA$ and do an LU or Cholesky factorisation to solve it.
- Remember that $\hat{x}$ is just our vector of $\begin{pmatrix} \gamma_0 \\ \gamma_1 \end{pmatrix}$. So we can plot the line of least squares regression using that two coefficients, but if we want our original data projected onto that line we need to compute $\hat{b} = A\hat{x}$. This is known as the **orthogonal projection** of $b$ onto the column space of $A$, which in our toy example, and in Figure 10.5 is $z \in \mathcal{C}(A)$.
- Then we can easily calculate residuals as $b - \hat{b}$.

**Linear least squares essentially minimises the squared residuals:** $||b - A\hat{x}||_2 = \min x||b - Ax||_2$ as the name suggests. So to stick to the previous notation, it tries to find $z$ that has the shortest possible corresponding perpendicular $w$, between $z$ and $b$.

# 11 | Orthogonal projection and low rank approximation

## 11.1 Projecting a vector onto a subspace

Looking at Figure 11.1, we see two vectors $a, b \in \mathbb{R}^m$. The span of $a$ is the column space of a matrix, which has only one column, which is the $a$ vector in this case.

Then we also have two components, $z$ that points in the direction of $a$, and $w$ that is orthogonal to $a$, so $b = z+w$, and $z$ is just some scalar times $a$, so $z = \chi a$.

So putting it all together, we know that $a^T w = 0$ since they are perpendicular, so: $0 = a^T(b - z) = a^T(b - \chi a) = a^T b - \chi a^T a \rightarrow a^T a \chi = a^T b$.

This is exactly the same formula we ended up with in the previous least squares fitting chapter.

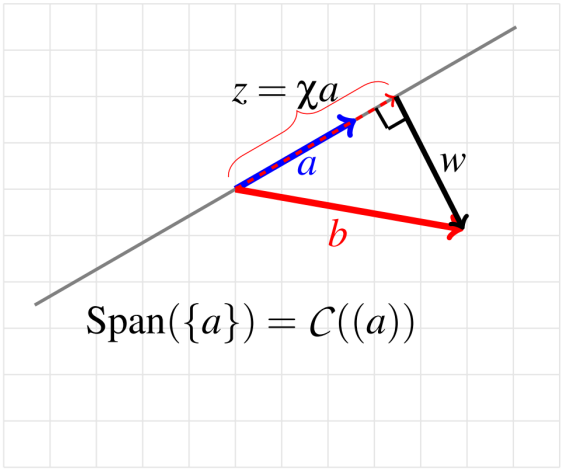



Figure 11.1: Orthogonal projection

Then we can solve for $\chi$ by:

$$\chi = (a^T a)^{-1} a^T b.$$

If we want to know $z$, i.e. the projection of $b$ into the column space of $a$, we calculate:

$$z = \chi a = a(a^T a)^{-1} a^T b,$$

where $a(a^T a)^{-1} a^T$ is the matrix that projects any vector onto the space spanned by the vector $a$. Finally if we want to know the the orthogonal component to $z$, we

calculate:

$$w = b - z = b - a(a^T a)^{-1}a^T b \xrightarrow[\text{outside}]{\text{bring } b} (I - a(a^T a)^{-1}a^T)b,$$

where $(I - a(a^T a)^{-1}a^T)$ is a matrix that projects any vector onto the orthogonal space of the span of $a$.

The really neat thing is that we end up with the same formulas if we want to project a vector into the column space of a matrix $A$, given it has linearly independent columns:

- $z = Ax$
- $A^T Ax = A^T b$
- $x = (A^T A)^{-1}A^T b$
- $z = A(A^T A)^{-1}A^T b$
- $w = (I - A(A^T A)^{-1}A^T)b$

Given $a, x \in \mathbb{R}^m$, let $P_a(x)$ and $P_a^\perp(x)$ be the projection of vector $x$ onto $Span(\{a\})$ and $Span(\{a\})^\perp$. Then the following are true:

- $P_a(a) = a$
- $P_a(\chi a) = \chi a$
- $P_a^\perp(\chi a) = 0$, the zero vector.
- $P_a(P_a(x)) = P_a(x)$
- $P_a(P_a^\perp(x)) = 0$, the zero vector.

## 11.2 Vector projection and the dot product

In Figure 11.2, we project $\vec{v}$ into the column space of $\vec{s}$. The resulting vector $\vec{p} \in \{c\vec{s} | c \in \mathbb{R}\}$, meaning $\vec{p}$ will basically be $\vec{s}$ times a constant $c$.

We can also see that $(\vec{v} - \vec{p})$ is perpendicular to $\vec{s}$.

Because of this their inner product has to be 0: $(\vec{v} - \vec{p}) \cdot \vec{s} = 0 \rightarrow (\vec{v} - c\vec{s}) \cdot \vec{s} = 0$.

By distributing $s$ and reorganising we get:

$$c = \frac{\vec{v} \cdot \vec{s}}{\vec{s} \cdot \vec{s}}.$$

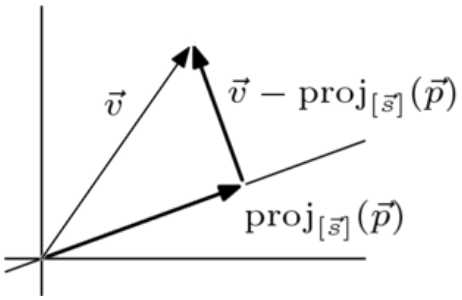



Figure 11.2

Since the resulting vector $\vec{p} \in \{c\vec{s} | c \in \mathbb{R}\}$, the projection of $\vec{v}$ onto $\vec{s}$ is:

$$\left(\frac{\vec{v} \cdot \vec{s}}{\vec{s} \cdot \vec{s}}\right) \vec{v}$$

If $\vec{s}$ is a unit vector, then this simplifies to $(\vec{v} \cdot \vec{s})\vec{s}$. Notice also, that this expression is the same as the one we derived in the previous section: $\chi = (a^T a)^{-1} a^T b$, because $(a^T a)^{-1} = \frac{1}{(a^T a)^{-1}}$.

The inner product of two vectors is $v^T s = ||v||_2 ||s||_2 \cos(\theta)$, where $||v||_2 \cos(\theta) = \vec{v} \cdot \vec{s_u}$ is called the scalar projection of $\vec{v}$ onto $\vec{s}$ and is equal to the length of the orthogonal projection of $\vec{v}$ on $\vec{s}$, and where $\vec{s_u}$ represents the unit vector in the direction of $\vec{s}$. So the dot product of two vectors express the amount of shared directionality in a sense.

**Importantly**, if $\vec{s}$ is not a unit vector, then $||v||_2 \cos(\theta) = \frac{\vec{v} \cdot \vec{s}}{|\vec{s}|}$, or in other words, we need to normalise $\vec{s}$ for this to hold.

Finally if we have a unit vector in say $\mathbb{R}^2$, then we can calculate a matrix $A$ that will project anything onto it. $\vec{s}$ could be described as a linear combination of its unit basis vectors, which are $\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}$ in $\mathbb{R}^2$. Since we have a unit vector, the projection formula is $(\vec{v} \cdot \vec{s})\vec{s}$, which now could be simply rewritten as $A\vec{x}$, where

$A = I \cdot \vec{s} \cdot \vec{s}$. So:

$$A = \begin{pmatrix} s_1^2 & s_1 s_2 \\ s_1 s_2 & s_2^2 \end{pmatrix},$$

so no matter what $\vec{s}$ we pick, we can calculate this matrix, and then project any other vector in $\mathbb{R}^2$ onto its column space, i.e. line.

## 11.3 Rank-1 approximation

If we have a black and white picture (to make things easier), where every pixel is characterised by a real value, corresponding to the given pixel's intensity, then we can think of such a picture as a matrix of numbers.

If now we partition this matrix $B$ into its columns: $B = (b_0|b_1|...|b_{n-1})$ and pick $a = b_0$, we can project any other column onto the $Span(\{a\})$ with the previously introduced $a(a^Ta)^{-1}a^Tx$, where $x \in B$.

If we would take the first few columns of the matrix and plot their values against their $i$ position, we would see a few plotted lines that are really similar. This is because, usually high-definition images don't change rapidly from pixel to pixel as we move from left to right. So if we project $b_1$ onto the $Span(\{b_0\})$: $a(a^Ta)^{-1}a^Tb_1 \approx b_0$, meaning we get a vector that is very similar or approximately equal to $b_0$.

We could do this for every one of the columns of $B$: $a(a^Ta)^{-1}a^TB$, where $(a^Ta)^{-1}a^TB = ((a^Ta)^{-1}B^Ta)^T = y$, and since $(a^Ta)^{-1}$ is just a scalar, $B^Ta$ is a row vector times a matrix which is a row vector, we get $a(a^Ta)^{-1}a^TB = ay^T$. So this projection is really just the outer product of a column vector $(a)$ and a row vector $(y^T)$.

This is known as the rank-1 approximation of $B$, where every column is just a scalar multiple of $y^T$, where $y = (a^Ta)^{-1}B^Ta$. This rank-1 approximated matrix has a rank of 1 (hence the name), because if we have two vectors $u \in \mathbb{R}^m$, and $v \in \mathbb{R}^n$, then their $m \times n$ outer product matrix formed by $uv^T$ has a rank of at most one. It can also have zero rank if either of those vectors are zero vector, but not more than one.

## 11.4 Rank-k approximation

Rank-k approximation follows logically from the above. Instead of projecting onto the column space of a single vector, i.e. $Span(\{a\})$, we can take $k$ columns of the picture/matrix $B$, form a matrix from them $A$ and project the whole picture $B$ onto $\mathcal{C}(A)$, with: $AY^T$, where $Y = ((A^TA)^{-1}A^TB)^T = B^TA(A^TA)^{-1}$.

This approximated matrix will have a rank of $k$, because let $U \in \mathbb{R}^{m\times k}$ and $V \in \mathbb{R}^{n\times k}$, then the $m \times n$ matrix $UV^T$ has a rank of at most $k$. The code below produces the various $k$ rank approximations of an image.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy import misc

# load the picture of lena
lena = misc.lena()
cols = lena.shape[1]
# approximate with the following ranks
ranks = [8, 32, 64, 256]
ranks_cols = [range(0,cols, cols/x) for x in ranks]

# setup image
fig, axes = plt.subplots(2,2)
fig.set_figheight(8.27)
fig.set_figwidth(8.27)

for i, ax in enumerate(axes.flat):
    l = l=lena[:,ranks_cols[i]]
    ATA = np.dot(l.T, l)
    ATAinv = np.linalg.pinv(ATA)
    AATAinv = np.dot(l, ATAinv)
    ATB = np.dot(l.T, lena)
    image = np.dot(AATAinv, ATB)
    ax.imshow(image, cmap=cm.gray)
    ax.set_title('Rank '+ str(l.shape[1]))
fig.savefig('rankkapprox.pdf', dpi=300,
            bbox_inches='tight', pad_inches=0)
```

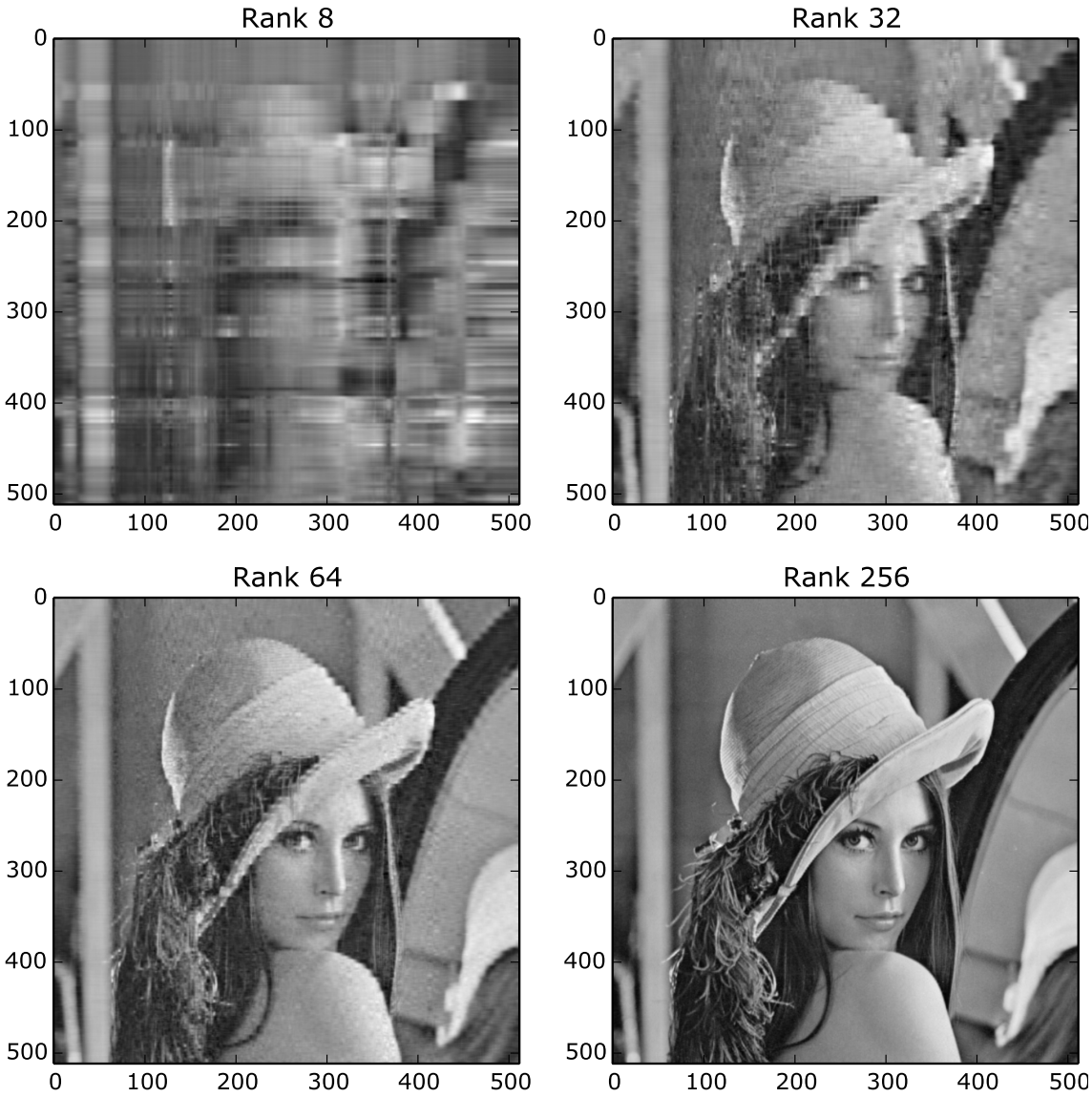



Figure 11.3: Rank-k approximations of the original $512 \times 512$ image.

## 11.5 Orthonormal vectors

Let $a_0, a_1, ..., a_{n-1} \in \mathbb{R}^m$. Then these vectors are mutually orthonormal if for all $0 \leq i, j < k$:

$$q_i^T q_j = \begin{cases} 1 = \text{if } i = j \\ 0 = \text{otherwise} \end{cases}$$

The first equation basically says that if they are parallel, then their dot product is 1, which can only happen if their length is one, so orthonormal vectors are mutually

perpendicular and have a length of one.

Orthogonal vectors like:

$$\begin{pmatrix} -1 \\ 1 \end{pmatrix}, \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

could be turned into orthonormal, by normalising their length to 1, which in this case just means dividing each component by $\sqrt{2}$:

$$\begin{pmatrix} \frac{-1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}, \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}.$$

The following vectors are mutually orthonormal:

$$\begin{pmatrix} -\sin(\theta) \\ \cos(\theta) \end{pmatrix} \perp \begin{pmatrix} \cos(\theta) \\ \sin(\theta) \end{pmatrix} \text{ and } \begin{pmatrix} \sin(\theta) \\ \cos(\theta) \end{pmatrix} \perp \begin{pmatrix} \cos(\theta) \\ -\sin(\theta) \end{pmatrix}$$

Here's an interesting two way relationship between orthonormal vectors and the identity matrix.

- Let $Q \in \mathbb{R}^{m \times n}$ (with $n \leq m$), and $Q^T Q = I$, then the columns of $Q = (q_0|q_1|...|1_{n-1})$ are mutually orthonormal.
- This is true in the other direction as well: if we have a set of mutually orthonormal vectors $q_0, q_1, ..., q_{n-1}$ then the matrix $Q$ we form from these have the property: $Q^T Q = I$.

Projecting a vector $x$ onto the $Span(\{q\})$ of a unit vector

- Remember for a non-unit vector (length $neq1$) the matrix that projects onto the column space of the vector $a$ is $a(a^T a)^{-1} a^T$.
- In this case, this is $q(q^T q)^{-1} q^T$, but since $||q||_2 = 1$, $(a^T a)^{-1} = 1$, so the expression simplifies to $qq^T$.
- So projecting $x$ onto the $Span(\{q\})$ is simply $q^T qx$ that we can reorder as $q^T xq$.

Following the previous point, let $Q \in \mathbb{R}^{m \times n}$ have orthonormal columns, then:

- The matrix that projects onto $\mathcal{C}(C)$ is simply $Q^TQ$.
- This is because in the equation $Q(Q^TQ)^{-1}Q^T$, the term $(Q^TQ)^{-1} = I$, because $Q$ is an orthonormal matrix, so $Q^TQ = I$ as we have shown before.
- Furthermore, the matrix that projects onto the orthogonal column space of $Q$, $\mathcal{C}(C)^\perp$ is simply $I - QQ^T$.
- This is because in the expression $I - Q(Q^TQ)^{-1}Q^T$, the term $(Q^TQ)^{-1} = I$ so it could be simply left out.

## 11.6 Orthonormal bases

We have three linearly independent vectors as shows in 11.4:

$$a_0 = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}, a_1 = \begin{pmatrix} 1 \\ 2 \\ 3 \end{pmatrix}, a_2 = \begin{pmatrix} -1 \\ -2 \\ 2 \end{pmatrix}.$$

We would like to find three orthonormal vectors $q_0, q_1$ and $q_2$, such that

$$Span(\{q_0, q_1, q_2\}) = Span(\{a_0, a_1, a_2\}).$$

This could be achieved with the **Gram-Schmidt orthogonalisation**. As we can see in Figure 11.4, in this example, $a_0$ and $a_1$ span the blue plane, while $a_2$ sticks out of it.

**Step 1 - Compute $\mathbf{q_0}$:**

- Normalise $a_0$: calculate its length as $\rho_{0,0} = ||a_0||_2$, and divide it by this: $q_0 = a_0/\rho_{0,0}$. In this example $\rho_{0,0} = \sqrt{(3)}$.
- The resulting vector could be seen in 11.5 as a thick vector pointing in the same direction as $a_0$.

**Step 2 - Compute $\mathbf{q_1}$:**

- Remember, we want $q_1$ to be perpendicular to $q_0$, look at Figure 11.6.

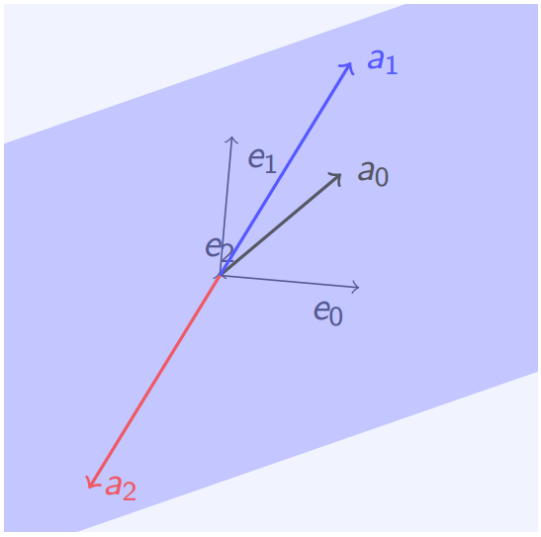



Figure 11.4

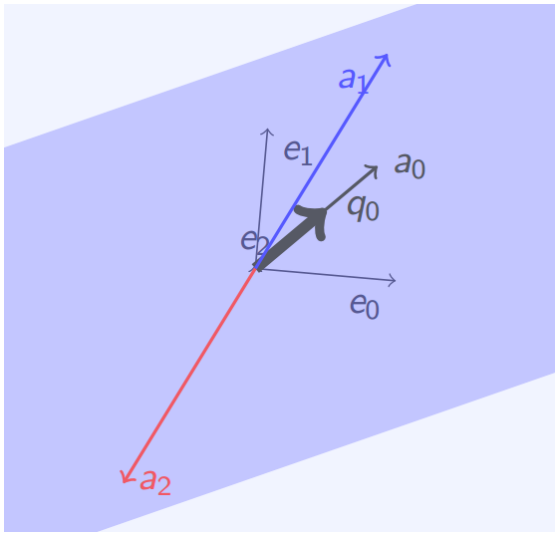



Figure 11.5

- As we see in Figure 11.6, the component of $a_1$ in the direction of $q_0$ is given by $q_0^T a_1 q_0$, because $q_0$ is already a unit vector, so no need to divide by $q_0^T q_0$.
- $\rho_{0,1} := q_0^T a_1$ represents the scalar by which we need to scale $q_0$ to get $a_1$'s orthogonal projection onto the line of $q_0$.
- $a_1^\perp = a_1 - q_0^T a_1 q_0$, which is perpendicular to $q_0$, so we want that to be $q_1$.
- All we need to do is normalise it: $q_1 = a_1^\perp / \rho_{1,1}$, where $\rho_{1,1} = ||a_1^\perp||_2$.
- Now we have $q_0$ and $q_1$ which span the same space as $a_0$ and $a_1$ but they are mutually orthonormal.

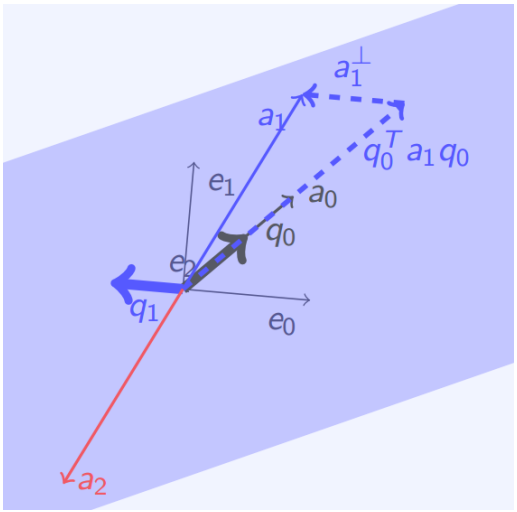



Figure 11.6

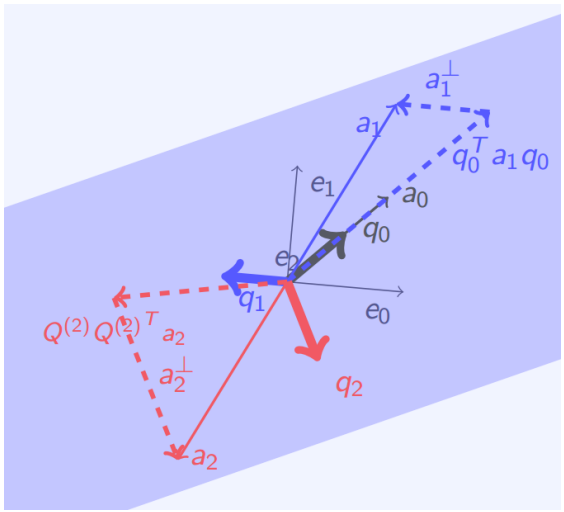



Figure 11.7

**Step 3 - Compute $q_2$:**

- As we can see in Figure 11.7 $a_2$ is not in the blue plane spanned by $q_0$ and $q_1$.
- But we can compute the projection of $a_2$ onto that plane.
- To do that, we need to project $a_2$ onto $\mathcal{C}(Q^{(2)})$, which is a matrix with two columns: $q_0$ and $q_1$. Then we can also compute $a_2^\perp$ by subtracting the projection of $a_2$ from $a_1$, just as we did before with $a_1^\perp$.
- $a_2^\perp = a_1 - (q_0 q_1)(q_0 q_1)^T a_2$, where $(q_0 q_1)(q_0 q_1)^T = Q^{(2)} {Q^{(2)}}^T$.
- Alternatively we can do this in two steps:
  - Compute $\rho_{0,2} := q_0^T a_2$ and $\rho_{1,2} := q_1^T a_2$, which are the scalers of the projections of $a_2$ onto $q_0$ and $q_1$, and which is just ${Q^{(2)}}^T a_1$.
  - Then take the linear combinations of those with $q_0$ and $q_1$ and subtract it from $a_2$: $a_2^\perp = a_2 - \rho_{0,2} q_0 - \rho_{1,2} q1$.
  - Essentially take vector $a_2$ and subtract out the component in the direction of $q_0$ ($\rho_{0,2} q_0$) and then from $q_1$ ($\rho_{1,2} q_1$).
- Finally we need to do is normalise it: $q_2 = a_2^\perp / \rho_{2,2}$, where $\rho_{2,2} = ||a_2^\perp||_2$.

Naturally the Gram-Schmidt process could be continued to infinite number of directions orthogonalising all linearly independent vectors in the original set.

If we use the second method, where we subtract each previously calculated orthogonal vector's component from $a_k$, then the calculation of the $\rho$ factors could be posed as a matrix-vector multiplication:

$$\left.\begin{array}{r} \rho_{0,k} := q_0^T a_k \\ \rho_{1,k} := q_1^T a_k \\ \vdots \\ \rho_{k-1,k} := q_{k-1}^T a_k \end{array}\right\} \begin{pmatrix} \rho_{0,k} \\ \rho_{1,k} \\ \vdots \\ \rho_{k-1,k} \end{pmatrix} := \begin{pmatrix} q_0^T \\ q_1^T \\ \vdots \\ q_{k-1}^T \end{pmatrix} a_k = (q_0 | \ldots | q_{k-1})^T a_k$$

Then the calculation of the $k^{\text{th}}$ orthonormal vector becomes:

$$a_k^\perp := ak - \rho_{0,k} q_0 - \rho_{1,k} q_1 - \ldots - \rho_{k-1,k} q_{k-1}.$$

This, again, could be written in a matrix-vector multiplication:

$$a_k^{\perp} := a_k - (q_0|\dots|q_{k-1}) \begin{pmatrix} \rho_{0,k} \\ \rho_{1,k} \\ \vdots \\ \rho_{k-1,k} \end{pmatrix}$$

This will be very important in the QR factorisation.

## 11.7 QR factorisation

Besides the LU and Cholesky factorisation the QR factorisation is the 3rd really important matrix decomposition in this course. The problem is still the same, we have $n$ linearly independent vectors $(a_0|a_1|\dots|a_{n-1})$, for which we would like to derive an orthonormal basis.

As we have seen earlier, with the Gram-Schmidt orthogonalisation, this could be written as:

$$(\rho_{0,0}q_0|\rho_{0,1}q_0 + \rho_{1,1}q_1|\dots|\rho_{0,n-1}q_0 + \rho_{1,n-1}q_1) + \dots + \rho_{n-1,n-1}q_{n-1},$$

or in matrix form:

$$QR = (q_0|q_1|\dots|q_{n-1}) \left(\begin{array}{c|c|c|c} \rho_{0,0} & \rho_{0,1} & \dots & \rho_{0,n-1} \\ \hline 0 & \rho_{1,1} & \dots & \rho_{1,n-1} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline 0 & 0 & \dots & \rho_{n-1,n-1} \end{array}\right)$$

Notice that the $R$ matrix is upper triangular. We calculated the exact same $q$ vectors and $\rho$ scalar coefficients with the Gram-Schmidt, but arranging everything in matrix form makes it really easy to get back $A = QR$.

We can use this in the linear least squares problem $Ax \approx b$. First we perform the $QR$ factorisation of $A$, and we get $A = QR$, where $Q^TQ = I$,since $Q$ has orthonormal basis.

- Remember that $x = (A^TA)^{-1}A^Tb$
- Substituting in $A = QR$, we get $x = ((QR)^T(QR))^{-1}(QR)^Tb$

- Remember that $(QR)^T = R^T Q^T$ with any matrix, so we get
- $x = (R^T Q^T QR)^{-1} R^T Q^T b$, where $Q^T Q = I$ and falls out.
- $x = (R^T R)^{-1} R^T Q^T b = R^{-1} R^{-T} R^T Q^T b$
- Here $R^{-T} R^T = I$ so we are left with:
- $x = R^{-1} Q^T b$

Instead of actually inverting, we can multiply by $R$, so for $Ax \approx b$ we get $Rx = Q^T b$. This is a nice alternative to the normal equation for LLS problems.

## 11.8 Change of basis

If $Q \in \mathbb{R}^{n \times n}$ has mutually orthonormal columns then which of the following are true:

- $Q^T Q = I$
- $QQ^T = I$
- $QQ^{-1} = I$
- $Q^{-1} = Q^T$

Let's say we have a vector $b$, which is the red one in Figure 11.8. We have some orthonormal basis for the subspace $b$ is in, and we want to express $b$ as a linear component of those unit basis vectors $q_0$ and $q_1$. Notice these have been normalised to make them orthonormal, hence the $\sqrt{2}$ term before their coordinates.

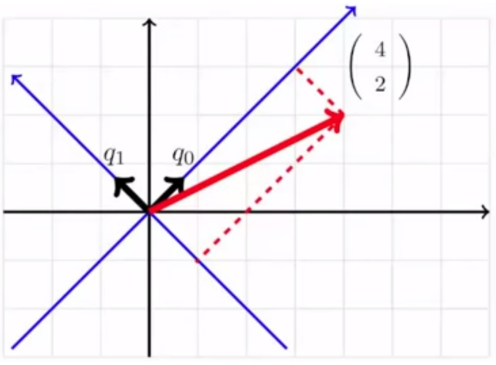



Figure 11.8

All we need to do then to calculate the new components of $b$, is shown below the graph. We are simply projecting $b$ onto the 1st then 2nd unit basis vector. Then, by subtracting one from the other we get $3\sqrt{2}q_0 - \sqrt{2}q_1$.

If we have an arbitrary basis for $\mathbb{R}^n$, we can rewrite it as a linear combination of its unit basis vectors:

$$b = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_{n-1} \end{pmatrix} = \beta_0 e_0 + \beta_1 e_1 + \ldots + \beta_{n-1} e_{n-1}.$$

Then we want to represent this space in a different space, for which we have the basis vectors $(a_0 \ldots a_{n-1})$. Then all we need to do is find the coefficients $(\chi_0 \ldots \chi_{n-1})$ that makes this true:

$$\begin{aligned} b &= \beta_0 e_0 + \beta_1 e_1 + \ldots + \beta_{n-1} e_{n-1} \\ &= \chi_0 a_0 + \chi_1 a_1 + \ldots + \chi_{n-1} a_{n-1} \end{aligned}$$

If we represent the linearly independent vectors $(a_0 \ldots a_{n-1})$ of the new basis, as a matrix $A$, this will be invertible, since its columns are linearly independent vectors. So we can rewrite the equation: $b = AA^{-1}b = Ax$, because $AA^{-1} = I$.

So we can simply solve $Ax = b$ or $x = A^{-1}b$ to get $x$ and perform the basis change.

If the original basis we start with is orthonormal and we just rotate the basis to get another orthonormal basis, then the calculation becomes even easier: $b = QQ^{-1}b = QQ^Tb$.

## 11.9 Singular Value Decomposition

This is one of the most important topics in linear algebra. Unfortunately this introductory course does not cover it deeply, so make sure to read up on it and check other sources.

Going back to our example rank-k approximation example, let $B \in \mathbb{R}^{m \times n}$. Then the singular value decomposition states, that:

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

- $U \in \mathbb{R}^{m \times r}$ and $U^T U = I$.
- $\Sigma \in \mathbb{R}^{r \times r}$ is a diagonal matrix whit positive diagonal elements that are ordered so that $\sigma_0 \geq \sigma_1 \geq \ldots \geq \sigma_{r-1} \geq 0$.
- $V \in \mathbb{R}^{n \times r}$ and $V^T V = I$.
- $r$ equals the rank of matrix $B$.

Writing this out in matrix form:

$$\begin{aligned} B &= U\Sigma V^T \\ &= (u_0|u_1|\ldots|u_{r-1}) \begin{pmatrix} \sigma_0 & 0 & \ldots & 0 \\ 0 & \sigma_1 & \ldots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & \sigma_{r-1} \end{pmatrix} (v_0|v_1|\ldots|v_{r-1}) \\ &= \sigma_0 u_0 v_0^T + \sigma_1 u_1 v_1^T + \ldots + \sigma_{r-1} u_{r-1} v_{r-1}^T \end{aligned}$$

So as we can see in the last equation, all we do is take a scalar ($sigma_0$) times a column vector ($u_0$) times a row vector ($v_0^T$), and then another scalar ($sigma_1$) times a column vector ($u_1$) times a row vector ($v_1^T$), etc. $u_0 v_0^T$ is just an outer product, so with every added term we add/reconstruct more and more detail of the original $B$ matrix.

Since the diagonal values in $\Sigma$ are ordered from highest to lowest, we start with the most important rank updates, and continue towards the least important ones. So unlike in the rank-k approximation, where we randomly selected some columns

to reconstruct the original matrix, here we have a clear order in which we have to add the column. This is highlighted below in Figure 11.9, where we have the same image in each row, reconstructed with the same number of columns, with rank-k approximation on the left and SVD on the right. Generated by the following code:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from scipy import linalg as la
from scipy import misc

# load the picture of lena, define ranks
lena = misc.lena()
cols = lena.shape[1]
ranks = [8, 32, 64, 128]
ranks_cols = [range(0,cols, cols/x) for x in ranks]

# perform SVD on lena
u, sv, v = la.svd(lena)
# create diagonal matrix from the returned singular values
s = np.zeros((lena.shape))
s[np.diag_indices(lena.shape[0])] = sv

# setup image
fig, axes = plt.subplots(4,2)
rank_axes = [0,2,4,6]
fig.set_figheight(11.69)
fig.set_figwidth(8.27)
for ind, ax in enumerate(axes.flat):
    i = ind/2
    if ind in rank_axes:
        l = l=lena[:,ranks_cols[i]]
        ATA = np.dot(l.T, l)
        ATAinv = la.pinv(ATA)
        AATAinv = np.dot(l, ATAinv)
        ATB = np.dot(l.T, lena)
        image = np.dot(AATAinv, ATB)
        ax.imshow(image, cmap=cm.gray)
        ax.set_title('Rank '+ str(l.shape[1]))
    else:
        r = ranks[i]
        sub_u = u[:,:r]
        sub_s = s[:r,:r]
        sub_v = v[:r,:]
        image = np.dot(np.dot(sub_u, sub_s),sub_v)
        ax.imshow(image,cmap=cm.gray)
        ax.set_title('Rank '+ str(r))
fig.savefig('rank_k_vs_svc.pdf', dpi=300,
            bbox_inches='tight', pad_inches=0)
```

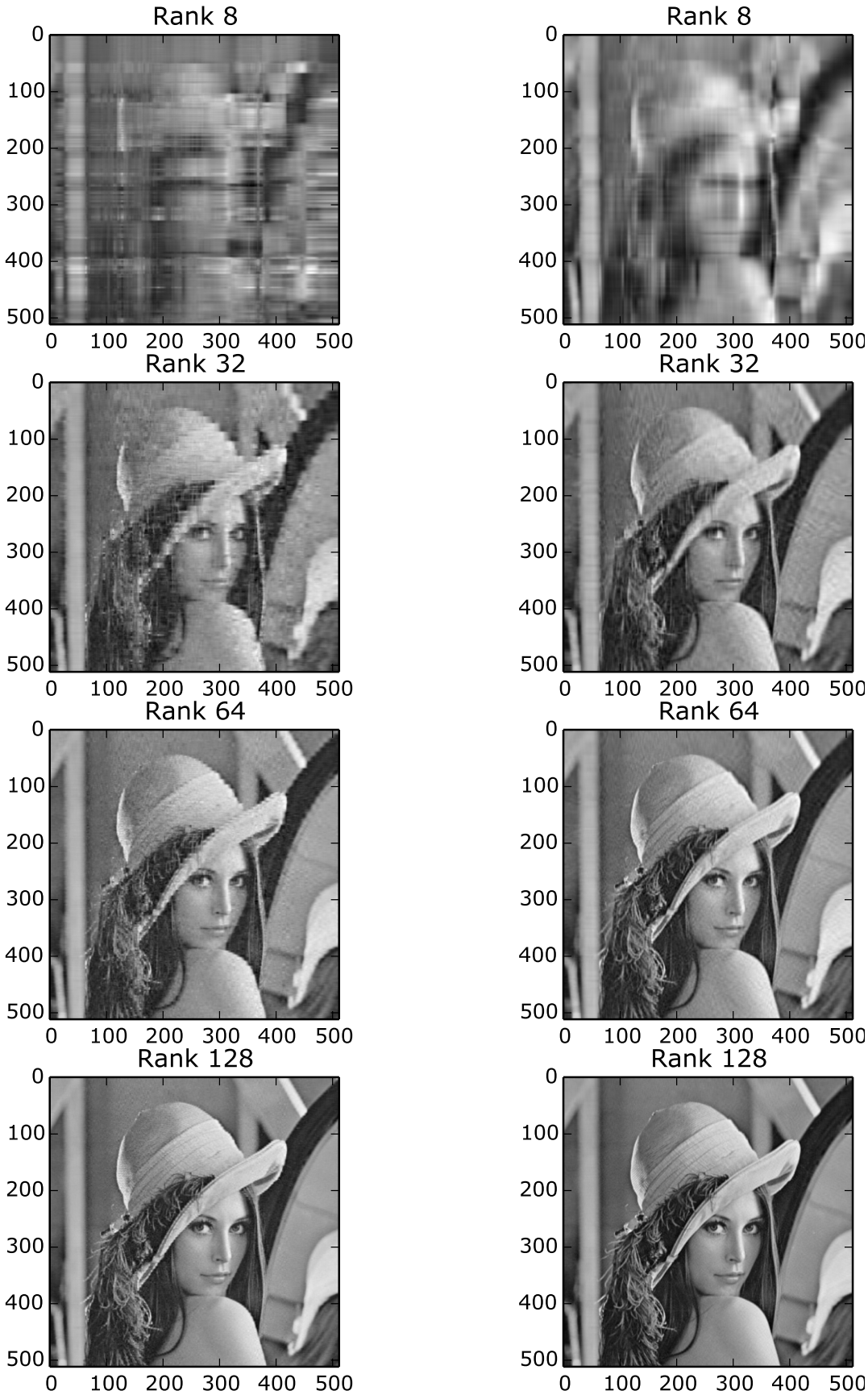



Figure 11.9: Rank-k approximations (left column) vs SVD. (right column)

We can clearly see in Figure 11.9, that with fewer ranks SVD does a much better job at reconstructing the details of the image. This is because SVD with rank $k$, is mathematically the *best possible approximation* of $B$, while rank-k approximation is suboptimal.

We can think of SVD of a 2D shearing matrix $M$ as a three step transformation: rotation $\rightarrow$ scaling $\rightarrow$ rotation. Have a look at Figure 11.10.

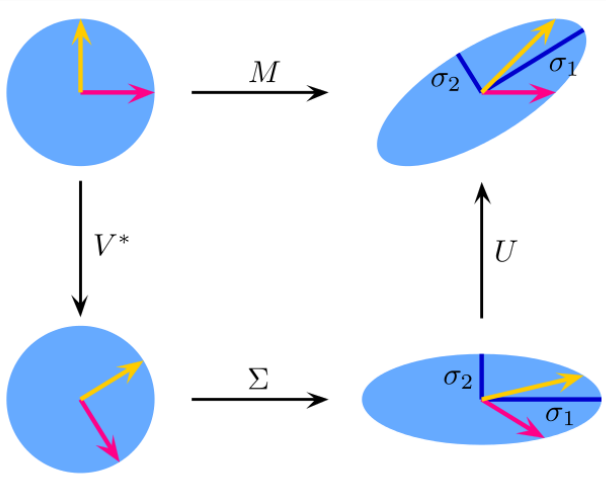



Figure 11.10: SVD as three transformations.

A really important result of linear algebra is the eigendecomposition or sometimes spectral decomposition is the factorization of a matrix into a canonical form, whereby the matrix is represented in terms of its eigenvalues and eigenvectors.

This topic and its importance is covered in the next chapter, however, he we have to point out how the eigendecomposition of a diagonalizable matrix $M$ relates to the SVD of $M^TM$ and $MM^T$:

- The left-singular vectors of $M$, which are the columns of $U$ are eigenvectors of $MM^T$.
- The right-singular vectors of $M$, which are the columns of $V$ are eigenvectors of $M^TM$.
- The non-zero singular values of M, which are the diagonal entries of $\Sigma$, are the square roots of the non-zero eigenvalues of both $M^TM$ and $MM^T$.

Finally we can use SVD to solve the LLS problem, because $A = U\Sigma V^T$ and we want to solve $Ax \approx b$ so we can substitute in, and simplify, and we get $x = V^T\Sigma^{-1}U^Tb$, where notice that the matrix we need to invert is a diagonal one, so this is a trivial and computationally cheap step.

# 12 | Eigenvalues and eigenvectors

In section 4.1 we saw this table for weather prediction. We can get from today's weather to tomorrow's with the probabilities shown.

| | Weather today | | |
|---|---|---|---|
| Weather tomorrow | Sunny | Cloudy | Rainy |
| Sunny | 0.4 | 0.3 | 0.1 |
| Cloudy | 0.4 | 0.3 | 0.6 |
| Rainy | 0.2 | 0.4 | 0.3 |

Table 12.1: Possibilities of changing weather

Predicting the weather a week from now, consists of taking a day, i.e. cloudy to start of with:

$$\begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix}$$

Then we multiply the transition matrix by it:

$$\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix} \begin{pmatrix} 0.3 \\ 0.3 \\ 0.4 \end{pmatrix} = \begin{pmatrix} 0.25 \\ 0.45 \\ 0.3 \end{pmatrix}.$$

If we do this for a weak, we will get:

$$d_5 = \begin{pmatrix} 0.2632 \\ 0.4207 \\ 0.3160 \end{pmatrix}, d_6 = \begin{pmatrix} 0.2621 \\ 0.4211 \\ 0.3157 \end{pmatrix}, d_7 = \begin{pmatrix} 0.2632 \\ 0.4210 \\ 0.3158 \end{pmatrix},$$

for day 5, 6 and 7. And we can see that the vector is converging to a stationary probability vector. The starting vector doesn't matter, we will settle on the same vector if we perform enough iterations.

Eventually, the state vector for tomorrow will be the state vector of today: $x^{k+1} \approx Px^{(k)}$. Then the question becomes, is there such vector that $Px = x$, because that

would describe the typical weather, i.e. the stationary distribution of the Markov chain.

This is an example of the algebraic eigenvalue problem:

$$Ax = \lambda x,$$

where $x$ is vector, $\lambda$ is a scalar, and $A \in \mathbb{R}^{n \times n}$ is a square matrix. In fact it has to be a square matrix, because $Ax$ has to have the same dimensions as $x$. So we want to find the eigenvalues $\lambda$, of which there are a maximum of $n$, and the non-zero eigenvectors for our $A$ matrix. These eigenvectors are not unique so we will want to give a general solution to describe them.

## 12.1 Intuition behind eigenvectors and PCA

This material is from this and this site.

If we imagine $A$ to be a 2D linear transformation matrix (which it is), that acts on a subspace by transforming (rotating, shearing, skewing, scaling) it as a gust of wind shown in Figure 12.1, then the eigenvector will show us the direction the matrix is blowing in.

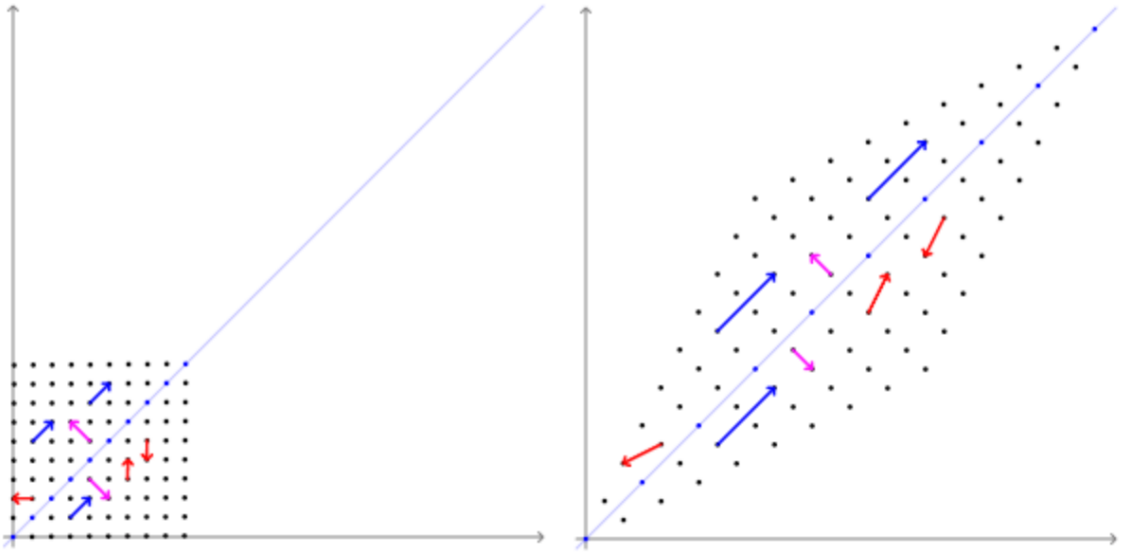


Figure 12.1: Eigenvectors of a subspace

So out of all the vectors affected by a matrix blowing through one space, which one is the eigenvector? It's the one that doesn't change direction; that is, the eigenvector is

already pointing in the same direction that the matrix is pushing all vectors toward. An eigenvector is a like a weathervane. The definition of an eigenvector, therefore, is a vector that responds to a matrix as though that matrix were a scalar coefficient.

You might also say that eigenvectors are axes along which linear transformation acts, stretching or compressing input vectors. They are the lines of change that represent the action of the larger matrix, the very "line" in linear transformation.

Because eigenvectors distil the axes of principal force that a matrix moves input along, they are useful in matrix decomposition; i.e. the diagonalisation of a matrix along its eigenvectors. Because those eigenvectors are representative of the matrix, they perform the same task as the autoencoders employed by deep neural networks.

To quote Yoshua Bengio:

> Many mathematical objects can be understood better by breaking them into constituent parts, or finding some properties of them that are universal, not caused by the way we choose to represent them.
>
> For example, integers can be decomposed into prime factors. The way we represent the number 12 will change depending on whether we write it in base ten or in binary, but it will always be true that $12 = 2 \times 2 \times 3$.
>
> From this representation we can conclude useful properties, such as that 12 is not divisible by 5, or that any integer multiple of 12 will be divisible by 3.
>
> Much as we can discover something about the true nature of an integer by decomposing it into prime factors, we can also decompose matrices in ways that show us information about their functional properties that is not obvious from the representation of the matrix as an array of elements.
>
> One of the most widely used kinds of matrix decomposition is called eigen-decomposition, in which we decompose a matrix into a set of eigenvectors and eigenvalues.

So when we do PCA on dataset, we calculate the covariance matrix first, and then the eigenvalues and eigenvectors of that. The great thing about calculating covariance is that, in a high-dimensional space where you can't eyeball intervariable relationships, you can know how two variables move together by the positive, negative or non-existent character of their covariance. The covariance matrix defines the shape of the data. In Figure 12.2, we see various relationships between two random variables: $X_1$

and$X_2$. Top-left: positive correlation, top-right: negative correlation, bottom: no correlation. The diagonal spread along eigenvectors is expressed by the covariance, while x-and-y-axis-aligned spread is expressed by the variance.

Finding the eigenvectors and eigenvalues of the covariance matrix is the equivalent of fitting those straight, principal-component lines to the variance of the data. Why? Because eigenvectors trace the principal lines of force, and the axes of greatest variance and covariance illustrate where the data is most susceptible to change.

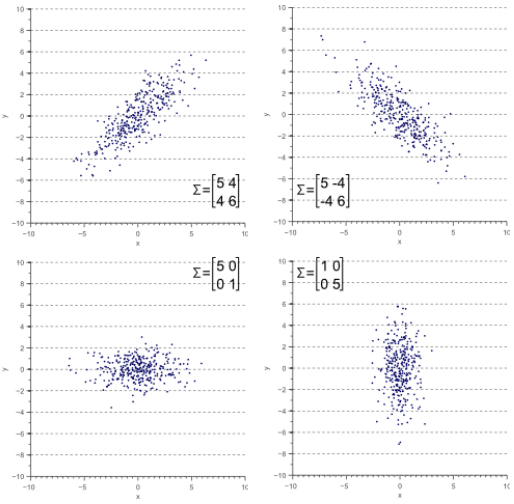



Figure 12.2

If a variable changes, it is being acted upon by a force known or unknown. If two variables change together, in all likelihood that is either because one is acting upon the other, or they are both subject to the same hidden and unnamed force.

When a matrix performs a linear transformation, eigenvectors trace the lines of force it applies to input; when a matrix is populated with the variance and covariance of the data, eigenvectors reflect the forces that have been applied to the given. One applies force and the other reflects it.

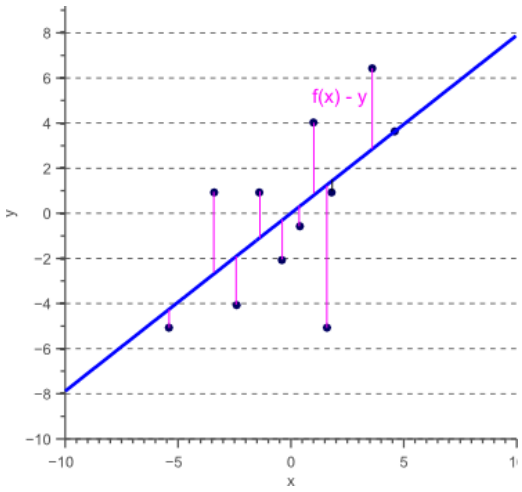



Figure 12.3

Eigenvalues are simply the coefficients attached to eigenvectors, which give the axes magnitude. In this case, they are the measure of the data's covariance. By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

When we do least squares fitting to a given dataset, we want to minimise the residual errors: $\sum_{i=0}^{N}(f(x_i)-y_i)^2$. With only one predictive variable, this means minimising the vertical distance between our regression line and the data points. data-points, as show in Figure 12.3

However, in the context of feature extraction, one might wonder why we would define

feature $x$ as the independent variable and feature $y$ as the dependent variable. In fact, we could easily define $y$ as the independent variable and find a linear function $f(y)$ that predicts the dependent variable $x$, such that $\sum_{i=0}^{N}(f(y_i)-x_i)^2$ is minimized.

Clearly, the choice of independent and dependent variables changes the resulting model, making ordinary least squares regression an asymmetric regressor. The reason for this is that least squares regression assumes the independent variable to be noise-free, whereas the dependent variable is assumed to be noisy. However, in the case of classification, all features are usually noisy observations such that neither $x$ or $y$ should be treated as independent. In fact, we would like to obtain a model $f(x, y)$ that minimizes both the horizontal and the vertical projection error simultaneously. This corresponds to finding a model such that the orthogonal projection error is minimized as shown by Figure 12.4.

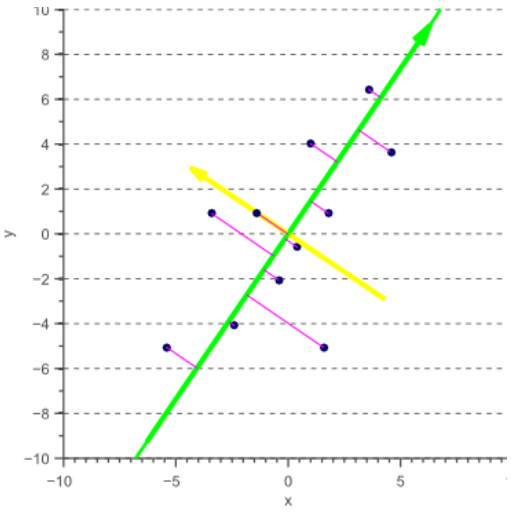



Figure 12.4

The resulting regression is called Total Least Squares regression or orthogonal regression, and assumes that both variables are imperfect observations. An interesting observation is now that the obtained vector, representing the projection direction that minimizes the orthogonal projection error, corresponds the the largest principal component of the data.

In other words, if we want to reduce the dimensionality by projecting the original data onto a vector such that the squared projection error is minimized in all directions, we can simply project the data onto the largest eigenvectors. This is exactly what we called PCA does.

## 12.2 Finding eigenvalues

- $Ax = \lambda x, \quad xneq0$, i.e. $x$ has to be non-trivial.
- $Ax - \lambda x = 0$
- $Ax - \lambda Ix = 0$

- $(A - \lambda I)x = 0$
- Notice that by definition, $x \in \mathcal{N}(A - \lambda I)$, and it cannot be the zero vector, which means, that there is more than just the zero vector in the null space of $A - \lambda I$. But by definition that also means that its columns are linearly dependent, which means its determinant has to zero, and it cannot be inverted so it is singular.
- Eigenvalues: find $\lambda$ so that $A - \lambda I$ is singular.
- Eigenvectors: find $x \in \mathcal{N}(A - \lambda I)$.

**Eigenvalues of a diagonal matrix**, are its diagonal elements $\delta_0, \delta_1, \ldots, \delta_{n-1}$, with the corresponding unit basis vectors $e_0, e_1, \ldots, e_{n-1}$ as eigenvectors.

**Eigenvalues of a triangular matrix:** are its diagonal elements. This is because an upper triangular matrix is only singular if and only if there is a zero on its diagonal. So in $A - \lambda I$ we get $v_{i,j} - \lambda = 0$, where $i = j$ for every diagonal element, which means every diagonal element will be an eigenvalue.

**Eigenvalues of a $2 \times 2$ matrix:** the inverse of $A \in \mathbb{R}^{2\times 2}$ is given by:

$$A^{-1} = \begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{det(A)} adj(A) = \frac{1}{ab - dc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}.$$

This means if $det(A) = 0$, then $A^{-1}$ does not exist. So what we want to do, is find where $det(A) = 0$:

$$A - \lambda I = \begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix},$$

so where $(a - \lambda)(d - \lambda) - cd = 0$ (which is called the characteristic polynomial of the matrix). Once we have concrete values in $A$, this is easy to solve using the $x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ formula, and results in two $\lambda$ eigenvalues.

Once we have eigenvalues, we can substitute them back and solve for the eigenvectors:

$$\begin{pmatrix} a - \lambda & b \\ c & d - \lambda \end{pmatrix} \begin{pmatrix} \chi_0 \\ \chi_1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}.$$

It could happen that $b^2 - 4ac < 0$ and consequently we end up with complex

eigenvalues for our $2 \times 2$ matrix. So for example the matrix:

$$\begin{pmatrix} 3-\lambda & -1 \\ 2 & 1-\lambda \end{pmatrix},$$

has no inverse if and only if $(3-\lambda)(1-\lambda)-(-1)(2) = 0$. This leads to $\lambda^2-4\lambda+5 = 0$, which we can solve with:

$$\lambda = \frac{-(-4) \pm \sqrt{(-4)^2 - 4(5)}}{2} = \frac{4 \pm \sqrt{-4}}{2} = 2 \pm i.$$

Then we could substitute in these $\lambda$ values as before and find complex valued eigenvectors.

**Eigenvalues of a $n \times n$ matrix:** The steps are exactly the same as for the $2 \times 2$ matrix.

- $A \in \mathbb{R}^{n \times n}$ is nonsingular if and only if $det(A) \neq 0$.
- So we want to find $\lambda$ such that, $det(A) = 0$, which is known as the characteristic polynomial of $A$.
- Since $A \in \mathbb{R}^{n \times n}$, the characteristic polynomial of $A$ is $n^{\text{th}}$ degree:
  - $det(A - \lambda I) = p_n(\lambda) = \gamma_0 + \gamma_1\lambda + \ldots + \gamma_{n-1}\lambda^{n-1} + \lambda^n$
  - $p_n(\lambda)$ has $n$ roots/eigenvalues, counting multiplicity, and $k$ distinct roots/eigenvalues, where $k \leq n$.
  - We can factor such a polynomial as: $p_n = (\lambda) = (\lambda-\lambda_0)^{n_0}(\lambda-\lambda_1)^{n_1} \ldots (\lambda-\lambda_{k-1})^{n_{k-1}}$, where $n_0 + n_1 + \ldots + n_{k-1} = n$, and where $n_j$ is the algebraic multiplicity of root/eigenvalue $\lambda_j$.
  - Even if all entries of $A$ coefficients of $p_n(\lambda)$ are real valued, some or all of the roots/eigenvalues may be complex values, in which case they come in conjugate pairs. So if $\lambda = \lambda_{Real} + i\lambda_{Imaginary}$ is a root, so is $\hat{\lambda} = \lambda_{Real} - i\lambda_{Imaginary}$
- The spectrum of $A$, $\Lambda(A)$ is the set of all eigenvalues.
- $A$ has $k = |\Lambda(A)|$ distinct eigenvalues.

The big problem however, is that **according to the Galois theory, if we have an arbitrary polynomial with degree $n \geq 5$, we cannot compute its roots in**

**a finite number of computations**. This means, this classical $det(A)$ dependent calculation of eigenvalues is limited to only small matrices.

## 12.3 Eigendecomposition

Eigendecomposition or spectral diagonalisation could be defined as: given $A \in \mathbb{R}^{n \times n}$, compute a nonsingular matrix $X$, such that

$$X^{-1}AX = \Lambda \iff AX = X\Lambda \iff A = X\Lambda X^{-1},$$

where $\Lambda$ is a diagonal matrix. It could be shown that $A$ could be diagonalised if and only if it has $n$ linearly independent eigenvectors.

Luckily the eigendecomposition of a matrix, corresponds to exactly what we have done before with our $2 \times 2$ matrix, when we were trying to find its eigenvalues and eigenvectors. For example, given:

$$A = \begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix},$$

Let's assume that we have found its eigenpairs $(\lambda, x)$ that satisfies $Ax = \lambda x$:

$$\begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} -1 \\ 1 \end{pmatrix} = 2 \begin{pmatrix} -1 \\ 1 \end{pmatrix},$$

and

$$\begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix} \begin{pmatrix} -1 \\ 2 \end{pmatrix} = 3 \begin{pmatrix} -1 \\ 2 \end{pmatrix}.$$

If we then combine the eigenvectors into a matrix and the eigenvalues into a diagonal matrix, we get:

$$\begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix} \left(\begin{array}{c|c} -1 & -1 \\ 1 & 2 \end{array}\right) = \left(\begin{array}{c|c} -1 & -1 \\ 1 & 2 \end{array}\right) \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}.$$

If we then multiply by the inverse of the eigenvector matrix we get:

$$\left(\begin{array}{c|c} -1 & -1 \\ 1 & 2 \end{array}\right)^{-1} \begin{pmatrix} 1 & -1 \\ 2 & 4 \end{pmatrix} \left(\begin{array}{c|c} -1 & -1 \\ 1 & 2 \end{array}\right) = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix},$$

which is $X^{-1}AX = \Lambda$.

There are a number of really important applications of the eigendecomposition of a matrix, but one particularly relevant is the calculation of power series. In the introduction of this chapter and in section 4.1, we talked about how we could calculate the weather 30 days from now, by exponentiating the transition matrix $P^{30} = P \times P \times \ldots \times P$. This however could become really expensive, especially if it's a large matrix. What we can do instead is, perform an eigendecomposition of $P = X\Lambda X^{-1}$, then exponentiate this 30 times as : $(X\Lambda X^{-1}) \times (X\Lambda X^{-1}) \ldots (X\Lambda X^{-1})$, then realize, that this is just $X\Lambda(X^{-1}X)\Lambda(X^{-1}X)\ldots\Lambda X^{-1} = X\Lambda^{30}X^{-1}$. Since $\Lambda$ is just a diagonal matrix with $P$'s eigenvalues as its entries, exponentiating it is really quick, as it just means raising each diagonal element to $k$: $\lambda_0^k, \lambda_1^k, \ldots, \lambda_{n-1}^k$.

A matrix that cannot be diagonalised is called a **deficient matrix**, meaning it doesn't have $n$ linearly independent eigenvalues associated with it. For example the matrix:

$$\begin{pmatrix} 2 & 1 \\ 0 & 2 \end{pmatrix},$$

is a deficient matrix, and had a null space with one dimension.

The algebraic multiplicity of an eigenvalue $\lambda$ is the power $m$ of the term $(x-\lambda)^m$ in the characteristic polynomial. The geometric multiplicity is the number of linearly independent eigenvectors you can find for an eigenvalue. Loosely speaking, the matrix is deficient in some sense when the two multiplicities do not match.

## 12.4 Properties of eigenvalues

- If we have a block upper triangular matrix:

$$A \in \mathbb{R}^{n\times n} = \begin{pmatrix} A_{0,0} & A_{0,1} & \ldots & A_{0,N-1} \\ 0 & A_{1,1} & \ldots & A_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \ldots & A_{N-1,N-1} \end{pmatrix},$$

  where all the matrices on the diagonal $(A_{0,0}, A_{1,1}, \ldots, A_{N-1,N-1})$ are square matrices, then:

$$\Lambda(A) = \Lambda(A_{0,0}) \cup \Lambda(A_{1,1}) \cup \ldots \cup \Lambda(A_{N-1,N-1}),$$

meaning that the eigenvalues of $A$ are simply the union of all the eigenvalues of the individual blocks on the diagonal.

- If we have a square matrix $A$, and choose two of its eigenvalues which are not the same, then the corresponding eigenvectors will be orthogonal to each other.
- If $Ax = \lambda x$, then $AAx = A^2x = \lambda^2 x$.
- If $Ax = \lambda x$, then $A^k x = \lambda^k x$.
- $A \in \mathbb{R}^{n \times n}$ is nonsingular if and only if $0 \notin \Lambda(A)$.
- If $\lambda \in \Lambda(A)$, then $\lambda \in \Lambda(A^T)$ and vice versa. So the eigenvalues of $A$ are the eigenvalues of $A^T$ as well and vice versa.

## 12.5 Why can we predict the weather as $Px^{(k)}$?

If P is a transition matrix as we have seen in section 4.1, and in the intro of this chapter, then all of its columns must sum to one to form a proper probability distribution. As a consequence of this, and the above, 1 must be an eigenvalue of such matrix, because:

$$\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix}^T \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = 1 \times \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}.$$

So if we use $x^{(k+1)} = Px^{(k)}$ to predict the weather, why does $x^{(k)}$ eventually has the property that $Px^{(k)} \approx x^{(k)}$? We know that one of the eigenvalues of $P$ is 1, in fact that is the largest:

$$\Lambda(\begin{pmatrix} 0.4 & 0.3 & 0.1 \\ 0.4 & 0.3 & 0.6 \\ 0.2 & 0.4 & 0.3 \end{pmatrix}) = \{1, \lambda_1, \lambda_2\} \text{ with } 1 > \lambda_1 \geq \lambda_2.$$

Let $(1, v_0), (\lambda_1, v_1)$ and $(\lambda_2, v_2)$ be eigenpairs. We will assume (it could be proven) that these eigenvectors are linearly independent. Then a starting vector $x^{(0)} \in \mathbb{R}^3$

could be written as a linear combination of them:

$$x^{(0)} = \gamma_0 v_0 + \gamma_1 v_1 + \gamma_2 v_2$$

The following day as we've seen is:

$$x^{(1)} = Px^{(0)} = P(\gamma_0 v_0 + \gamma_1 v_1 + \gamma_2 v_2) = \gamma_0 P v_0 + \gamma_1 P v_1 + \gamma_2 P v_2 = \gamma_0 v_0 + \gamma_1 \lambda_1 v_1 + \gamma_2 \lambda_2 v_2,$$

because $Pv_0 = v_0$, since $Pv_0 = (1)v_0$ by the definition of eigenvectors, and for the same reasons $Pv_1 = \lambda_1 v_1$ and $Pv_2 = \lambda_2 v_2$. If we then continue with the next day:

$$\begin{aligned} x^{(2)} = Px^{(1)} &= P(\gamma_0 v_0 + \gamma_1 \lambda_1 v_1 + \gamma_2 \lambda_2 v_2) \\ &= \gamma_0 P v_0 + \gamma_1 P \lambda_1 v_1 + \gamma_2 P \lambda_2 v_2 \\ &= \gamma_0 v_0 + \gamma_1 \lambda_1^2 v_1 + \gamma_2 \lambda 2^2 v_2 \end{aligned}$$

So we look at the limit of this:

$$\lim_{k \to \infty} x^{(k)} = \lim_{k \to \infty} \left( \gamma_0 v_0 + \gamma_1 \lambda_1^k v_1 + \gamma_2 \lambda_2^k v_2 \right),$$

we see that as $k \to \infty$ the 2nd and 3rd term will become 0, because $1 > \lambda_1 \geq \lambda_2$, so after many days we end up with a vector that points into the direction of the 1st eigenvector $\gamma_0 v_0$.

## 12.6 The power method

To introduce the power method or power iteration, which is a widely used algorithm to compute the eigenvectors and eigenvalues of a $A \in \mathbb{R}^{n \times n}$ matrix, where $n \geq 5$, we with the following assumptions:

- Let $\lambda_0, \lambda_1, \ldots, \lambda_{n-1}$ be eigenvalues of $A \in \mathbb{R}^{m \times n}$.
- $|\lambda_0| > |\lambda_1| \geq |\lambda_2| \geq \ldots \geq |\lambda_{n-1}|$.
- Let $(\lambda_j, v_j)$ be eigenpairs of $A$.
- Assume that $v_0, v_1, \ldots, v_{n-1}$ are linearly independent.
- Recall that then $A = V \Lambda V^{-1}$.

Then following the exact same procedure as we did in the previous section, we can pick an arbitrary vector $x^{(0)} \in \mathbb{R}^n$ which could be written as:

$$x^{(0)} = \gamma_0 v_0 + \gamma_1 v_1 + \ldots + \gamma_{n-1} v_{n-1}.$$

And the following iterations:

$$\begin{aligned} x^{(1)} = Ax^{(0)} &= A(\gamma_0 v_0 + \gamma_1 v_1 + \ldots + \gamma_{n-1} v_{n-1}) \\ &= \gamma_0 \lambda_0 v_0 + \gamma_1 \lambda_1 v_1 + \ldots + \gamma_{n-1} \lambda_{n-1} v_{n-1}, \end{aligned}$$

because $Av_0 = \lambda_0 v_0, Av_1 = \lambda_1 v_1, Av_{n-1} = \lambda_{n-1} v_{n-1}$, by the definition of eigenvectors: $Ax = \lambda x$. So consequently the following terms are:

$$\begin{aligned} x^{(2)} = Ax^{(1)} &= A(\gamma_0 \lambda_0 v_0 + \gamma_1 \lambda_1 v_1 + \ldots + \gamma_{n-1} \lambda_{n-1} v_{n-1}) \\ &= \gamma_0 \lambda_0^2 v_0 + \gamma_1 \lambda_1^2 v_1 + \ldots + \gamma_{n-1} \lambda_{n-1}^2 v_{n-1}, \end{aligned}$$

and in general

$$\begin{aligned} x^{(k+1)} = Ax^{(k)} &= A(\gamma_0 \lambda_0^k v_0 + \gamma_1 \lambda_1^k v_1 + \ldots + \gamma_{n-1} \lambda_{n-1}^k v_{n-1}) \\ &= \gamma_0 \lambda_0^{k+1} v_0 + \gamma_1 \lambda_1^{k+1} v_1 + \ldots + \gamma_{n-1} \lambda_{n-1}^{k+1} v_{n-1}. \end{aligned}$$

So as $k \to \infty$:

$$\lim_{k\to\infty} x^{(k)} = \lim_{k\to\infty} \left(\gamma_0 \lambda_0^k v_0 + \gamma_1 \lambda_1^k v_1 + \ldots + \gamma_{n-1} \lambda_{n-1}^k v_{n-1}\right),$$

which means, that the first term $\gamma_0 \lambda^k v_0$ is doing to dominate all other terms, as $\lambda_0$ is the largest eigenvalue. So eventually the resulting vector will point into the direction of the first eigenvector $v_0$.

The problem is however, that if $\lambda_0 > 1$, then as $k \to \infty$ so will $\gamma_0 \lambda^k v_0 \to \infty$. Conversely if $\lambda_0 < 1$ then as $k \to \infty$ so will $\gamma_0 \lambda^k v_0 \to 0$. This is obviously not good, we need to keep the length of this vector under control. So in the power method, we divide each term by $\lambda_0$. Then if we work out the algebra we get:

$$x^{(k+1)} = \gamma_0 v_0 + \gamma_1 \frac{\lambda_1}{\lambda_0}^{k+1} v_1 + \ldots + \gamma_{n-1} \frac{\lambda_{n-1}}{\lambda_0}^{k+1} v_{n-1}.$$

So in the limit:

$$\lim_{k\to\infty} x^{(k)} = \lim_{k\to\infty} \left( \gamma_0 v_0 + \gamma_1 \frac{\lambda_1}{\lambda_0}^k v_1 + \ldots + \gamma_{n-1} \frac{\lambda_{n-1}}{\lambda_0}^k v_{n-1} \right),$$

all terms will go to 0, except the first, and eventually the vector will point in the direction of the first eigenvector.

In it's current form of the algorithm, we still need to know the eigenvalues, which is problematic for $A \in \mathbb{R}^{n\times n}, n \geq 5$, as we have mentioned before, because of the Galois theory. Since we are only interested in the direction of the eigenvectors, we can simply divide in each step by the length of the vector to prevent it scaling to infinity or 0.

So we start the algorithm with $x^{(0)} \in \mathbb{R}^n$ and do the following step iteratively:

$$x^{(k+1)} := \frac{Ax^{(k)}}{||Ax^{(k)}||_2}.$$

Although the power iteration method approximates only one eigenvalue of a matrix, it remains useful for certain computational problems. For instance, Google uses it to calculate the PageRank of documents in their search engine, and Twitter uses it to show users recommendations of who to follow. For matrices that are well-conditioned and as sparse as the Web matrix, the power iteration method can be more efficient than other methods of finding the dominant eigenvector.

Some of the more advanced eigenvalue algorithms can be understood as variations of the power iteration. For instance, the inverse iteration method applies power iteration to the matrix $A^{-1}$.