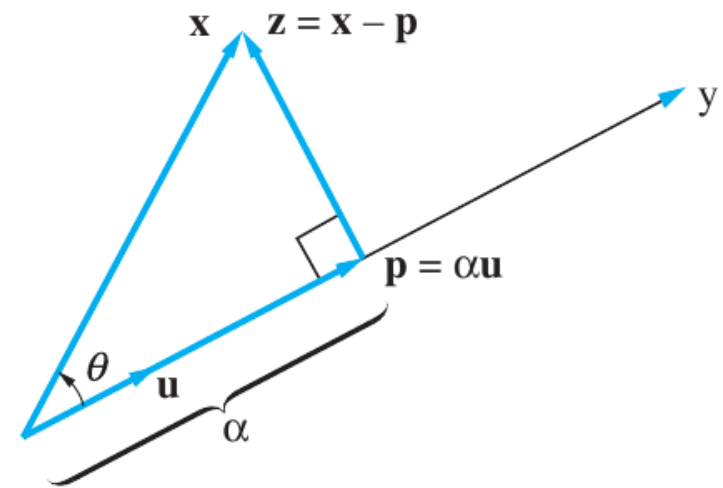


15.1 Orthogonal Projection

- Vectors are **orthogonal** when their dot product (and more generally their **inner product** $\langle \mathbf{x}, \mathbf{y} \rangle$) equals 0.

- Note that the dot product can be written as:

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^T \mathbf{y} = \|\mathbf{x}\| \|\mathbf{y}\| \cos \theta$$



Scalar projection of \mathbf{x} onto \mathbf{y} :

$$\alpha = \frac{\mathbf{x}^T \mathbf{y}}{\|\mathbf{y}\|}$$

Vector projection of \mathbf{x} onto \mathbf{y} :

$$\mathbf{p} = \alpha \mathbf{u} = \alpha \frac{1}{\|\mathbf{y}\|} \mathbf{y} = \frac{\mathbf{x}^T \mathbf{y}}{\mathbf{y}^T \mathbf{y}} \mathbf{y}$$

- ❑ Vectors are **orthonormal** if they are orthogonal and also unit vectors.
- ❑ An **orthogonal matrix** is composed of orthonormal column vectors.
- ❑ If **A** is an **orthogonal matrix** then: $\mathbf{A}^{-1} = \mathbf{A}^T$
- ❑ This means that $(\mathbf{AB})^T = \mathbf{B}^T \mathbf{A}^T$ for **orthogonal matrices**. This result can also be proved for non-orthogonal matrices.

15.2 Vector Spaces

- We define a set of rules that are characteristic of vectors and any mathematical objects that obey those rules we call a **vector space**.
- Examples include:
 - 3D space \mathbb{R}^3
 - Set of all real $m \times n$ matrices
 - Set of all polynomials
 - Set of all continuous functions
- For all vector spaces we can utilise the properties and methods of linear algebra.

- ❑ Taking all possible combinations of the rows of a matrix gives the **row space**. It represents all sets of equations that the system can define.
- ❑ Taking all possible combinations of the columns of a matrix gives the **column space**. It represents all possible solutions to the matrix equation $\mathbf{Ax} = \mathbf{b}$.
- ❑ All possible solution to the matrix equation $\mathbf{Ax} = \mathbf{0}$ is groups into a set known as the **null space**. The dimension of the null space is called the **nullity**.
- ❑ The number of linearly independent vectors in the row or column space is called the **rank** of the matrix (**dimension of column/row space**).
- ❑ The **rank + nullity = n** for a square matrix.

Definition of a vector space

Let V be a set on which the operations of addition and scalar multiplication are defined. By this we mean that, with each pair of elements \mathbf{x} and \mathbf{y} in V , we can associate a unique element $\mathbf{x} + \mathbf{y}$ that is also in V , and with each element \mathbf{x} in V and each scalar α , we can associate a unique element $\alpha\mathbf{x}$ in V . The set V together with the operations of addition and scalar multiplication is said to form a **vector space** if the following axioms are satisfied:

- A1. $\mathbf{x} + \mathbf{y} = \mathbf{y} + \mathbf{x}$ for any \mathbf{x} and \mathbf{y} in V .
- A2. $(\mathbf{x} + \mathbf{y}) + \mathbf{z} = \mathbf{x} + (\mathbf{y} + \mathbf{z})$ for any \mathbf{x} , \mathbf{y} , and \mathbf{z} in V .
- A3. There exists an element $\mathbf{0}$ in V such that $\mathbf{x} + \mathbf{0} = \mathbf{x}$ for each $\mathbf{x} \in V$.
- A4. For each $\mathbf{x} \in V$, there exists an element $-\mathbf{x}$ in V such that $\mathbf{x} + (-\mathbf{x}) = \mathbf{0}$.
- A5. $\alpha(\mathbf{x} + \mathbf{y}) = \alpha\mathbf{x} + \alpha\mathbf{y}$ for each scalar α and any \mathbf{x} and \mathbf{y} in V .
- A6. $(\alpha + \beta)\mathbf{x} = \alpha\mathbf{x} + \beta\mathbf{x}$ for any scalars α and β and any $\mathbf{x} \in V$.
- A7. $(\alpha\beta)\mathbf{x} = \alpha(\beta\mathbf{x})$ for any scalars α and β and any $\mathbf{x} \in V$.
- A8. $1\mathbf{x} = \mathbf{x}$ for all $\mathbf{x} \in V$.

15.3 Inner Products

An **inner product** on a vector space V is an operation on V that assigns, to each pair of vectors \mathbf{x} and \mathbf{y} in V , a real number $\langle \mathbf{x}, \mathbf{y} \rangle$ satisfying the following conditions:

- I. $\langle \mathbf{x}, \mathbf{x} \rangle \geq 0$ with equality if and only if $\mathbf{x} = \mathbf{0}$.
- II. $\langle \mathbf{x}, \mathbf{y} \rangle = \langle \mathbf{y}, \mathbf{x} \rangle$ for all \mathbf{x} and \mathbf{y} in V .
- III. $\langle \alpha \mathbf{x} + \beta \mathbf{y}, \mathbf{z} \rangle = \alpha \langle \mathbf{x}, \mathbf{z} \rangle + \beta \langle \mathbf{y}, \mathbf{z} \rangle$ for all $\mathbf{x}, \mathbf{y}, \mathbf{z}$ in V and all scalars α and β .

- ❑ Inner products are a generalisation of the dot product.
- ❑ You can think of the dot product as a type of inner product. In other words it obeys all those rule.
- ❑ Any vector space with a defined inner product is known as an **Inner Product Space**.

- ❑ Other inner products include the **weighted dot product** which has applications such as networks:

$$\langle \mathbf{x}, \mathbf{y} \rangle = \sum_{i=1}^n x_i y_i w_i$$

- ❑ The **sum of element-wise multiplication between matrices** (which can be used in data handling):

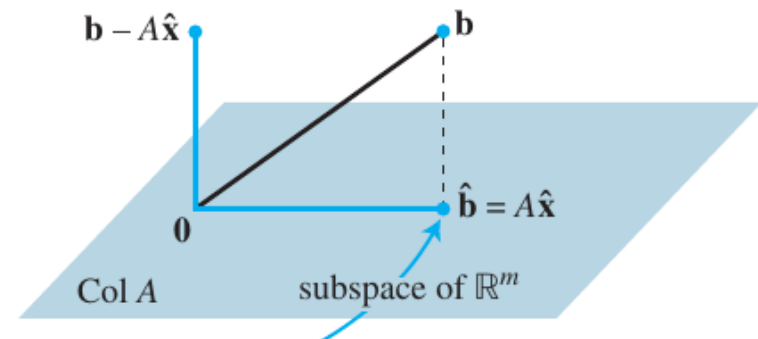
$$\langle A, B \rangle = \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ij}$$

- ❑ The **integral of the product of 2 continuous functions** (which means things like coefficients in Fourier series can be calculated using linear algebra methods):

$$\langle f, g \rangle = \int_a^b f(x) g(x) dx$$

15.4 Least Squares Revisited

- ❑ A fitting method that minimises the square of the error vector that we encountered last lecture.
- ❑ It finds the “best” solution to an inconsistent system of equations.
- ❑ The equation to solve for the “least squares” values of \mathbf{x} are the solutions to the **normal equations**:



$$\mathbf{A}^T \mathbf{A} \mathbf{x} = \mathbf{A}^T \mathbf{b}$$

- ❑ Notice that **if \mathbf{A} is orthogonal then $\mathbf{x} = \mathbf{A}^T \mathbf{b}$.**

EXAMPLE 1 Find the solution to the least squares which fits a straight line to 3 data points.

$$(1, 0), (2, 1), \text{ and } (3, 3) \longrightarrow y = c_0 + c_1x$$

- Write as a linear system of equations:

$$\begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix}$$

$$\begin{aligned} c_0 + c_1 &= 0 \\ c_0 + 2c_1 &= 1 \\ c_0 + 3c_1 &= 3 \end{aligned}$$

- Construct the least squares equation:

$$A^T A = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \end{bmatrix} = \begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix}$$

$$A^T \mathbf{b} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \end{bmatrix}$$

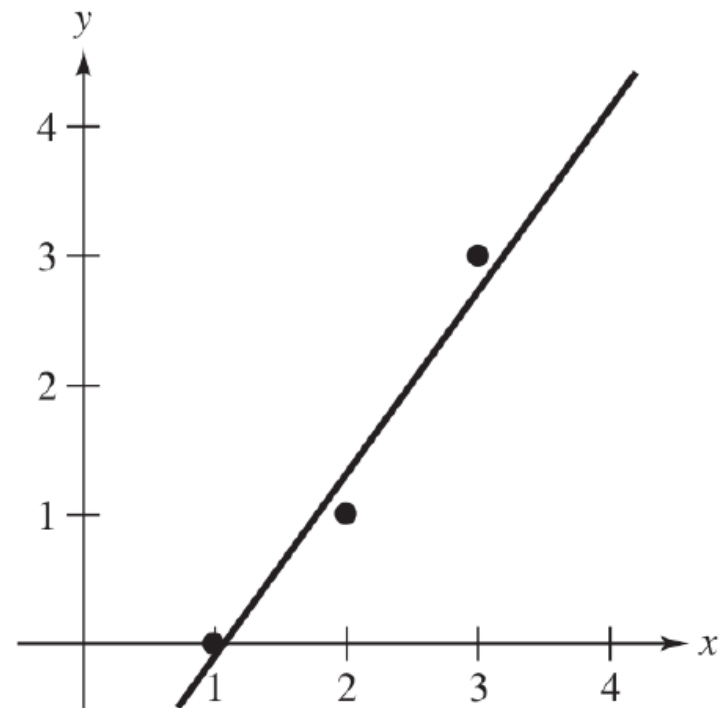
- Solve the resulting system:

$$A^T A \mathbf{x} = A^T \mathbf{b}$$

$$\begin{bmatrix} 3 & 6 \\ 6 & 14 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix} = \begin{bmatrix} 4 \\ 11 \end{bmatrix}$$

$$\mathbf{x} = \begin{bmatrix} -\frac{5}{3} \\ \frac{3}{2} \end{bmatrix}$$

$$y = \frac{3}{2}x - \frac{5}{3}$$



15.5 Norms

- ❑ Can be thought of as some way to quantify various sizes in a set of vectors. One norm is the **absolute value norm**: $\|\mathbf{x}\| = |\mathbf{x}|$
- ❑ Different norms give us different information and are each useful in various applications (depend on **inner product space**).

- ❑ Given a vector \mathbf{x}_i we denote a general **p-norm** as:

$$\|\mathbf{x}\|_p = \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$

- ❑ When $p = 1$ we have the **Taxicab value norm**: $\|\mathbf{x}\|_1 = \sum_{i=1}^n |x_i|$

- ❑ When $p = 2$ we have the **Euclidean norm** (classic vector magnitude):

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^n |x_i|^2 \right)^{1/2}$$

Note that squaring the 2-norm gives the dot (inner) product of a vector with itself

- Another common norm is the **maximum (infinity) norm**:

$$\|\mathbf{x}\|_{\infty} = \max_{1 \leq i \leq n} |x_i|$$

- Each of the standard vector norms above have a compatible **matrix norm**. We define the relationship between a vector norm and the matrix norm it induces as:

$$\|A\| = \max_{\mathbf{x} \neq \mathbf{0}} \frac{\|A\mathbf{x}\|}{\|\mathbf{x}\|}$$

- From the above definition, for matrices the **1-norm** and **∞ -norms** are calculated as maximums of **row-sums** and **column-sums** respectively. The 2-norm for a matrix is more involved so we won't get into that here.

$$\|A\|_1 = \max_{1 \leq j \leq n} \left(\sum_{i=1}^m |a_{ij}| \right)$$

$$\|A\|_{\infty} = \max_{1 \leq i \leq m} \left(\sum_{j=1}^n |a_{ij}| \right)$$

- ❑ Other matrix norms that can be defined that relate to whatever inner product we define for a given vector space such as the **Frobenius norm**:

$$\|A\|_F = (\langle A, A \rangle)^{1/2} = \left(\sum_{i=1}^m \sum_{j=1}^n a_{ij}^2 \right)^{1/2}$$

- ❑ The above comes from the inner product:

$$\langle A, B \rangle = \sum_{i=1}^m \sum_{j=1}^n a_{ij} b_{ij}$$

- ❑ Matlab has a **norm()** function that calculates these norms.

- Norms are defined to conform to the following 3 conditions:

- (i) $\|A\| \geq 0$ and $\|A\| = 0$ if and only if $A = O$
- (ii) $\|\alpha A\| = |\alpha| \|A\|$
- (iii) $\|A + B\| \leq \|A\| + \|B\|$ Triangle inequality

- For matrix norms there is a family of norms that also satisfy:

$$(iv) \|AB\| \leq \|A\| \|B\|$$

- This last property allows us to define the condition number of a matrix.
- Note that all the norms defined in these notes obey all 4 properties.
- The norm of a difference gives an indication of the “distance” between the objects.
 $\|u - v\|$

EXAMPLE 2 Calculate the 1-norm and the ∞ -norm of the following matrix.

$$A = \begin{pmatrix} -3 & 2 & 4 & -3 \\ 5 & -2 & -3 & 5 \\ 2 & 1 & -6 & 4 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

- The 1-norm is the sum of the absolute values of the elements of the column vector \mathbf{x} . For a matrix we take the largest of these:

$$\|A\|_1 = |4| + |-3| + |-6| + |1| = 14$$

- The ∞ -norm is the maximum absolute value of the vector \mathbf{x} . For a matrix we create the vector \mathbf{x} from the sums of the rows.

$$\|A\|_\infty = |5| + |-2| + |-3| + |5| = 15$$

15.6 Condition Number

- ❑ The **condition number** of a matrix can tell us how numerically stable our method is.
- ❑ Stability here can be thought of as **whether our solution is sensitive to small changes or not**. In other words, if I change the system by a small amount (possibly induced by measurement or round-off errors etc) then does the computed solution drastically change or not?
- ❑ Matrices with a large condition number are called **ill-conditioned**.

EXAMPLE 3 Observe how the solution to the linear system changes by a large amount when the data in the matrix changes only by a small amount.

- ❑ Create an ill-conditioned matrix:

```
>> format long
>> A = [1 1-1e-8; 1 1+1e-8]
A =
           1           0.999999999
           1           1.000000001
```

- ❑ Let's pick a vector $\mathbf{x} = [1; 3]$ to obtain \mathbf{b} in the matrix equation $\mathbf{Ax} = \mathbf{b}$.

```
>> x = [1; 3]
x =
     1
     3
>> b = A*x
b =
     3.999999997
     4.000000003
```

- Now let's use the backslash operator to solve the linear system for \mathbf{x} using the \mathbf{b} we just calculated:

```
>> x_check_method = A \ b
x_check_method =
    0.999999994448885
    3.00000000555112
```

- So solving the system didn't quite bring us back to our original values of \mathbf{x} but still pretty close.

- Now look what happens if the measured data, \mathbf{b} , has a small error of $1e-7 = 0.0000001$:

```
>> x2 = A \ (b + [1e-7;-1e-7])
x2 =
```

The solution for \mathbf{x} has become totally corrupted from a small in change in \mathbf{b}

← $\begin{cases} 11.0000001165734 \\ -7.00000011657342 \end{cases}$

- Similarly if we change **A** slightly it corrupts the solution:

```
>> A = [1 1-1e-7; 1 1+1e-7]
```

```
A =
```

```
1.0000000000000000 0.9999999800000000  
1.0000000000000000 1.0000000200000000
```

```
>> A \ b
```

```
ans =
```

```
3.699999999611422  
0.300000000388578
```

- We can check the **condition number** of a matrix in Matlab to see how well/badly conditioned it is.

```
>> format shortg
```

```
>> cond(A)
```

```
ans =
```

```
2e+08
```

- We will discuss the meaning of this number after we look more closely at how the condition number is calculated.

- Let's define the **residual** to be the difference in the **b**-vectors. You can think of it as a small change in the original system due to measurement errors for example:

$$\mathbf{r} = \mathbf{b} - A\mathbf{x}' = A\mathbf{x} - A\mathbf{x}' = A\mathbf{e}$$

where **e** represents the **solution error**.

- We will scale these by the original vectors and call them **relative residual** and **relative error**:

$$\frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

$$\frac{\|\mathbf{e}\|}{\|\mathbf{x}\|}$$

Note we can use any norm that is useful to measure the difference in vectors

- Now from the top equation along with the 4th property of matrix norms we have:

$$\|\mathbf{e}\| \leq \|A^{-1}\| \|\mathbf{r}\|$$

$$\|\mathbf{r}\| = \|A\mathbf{e}\| \leq \|A\| \|\mathbf{e}\|$$

- Combining those inequalities we have:

$$\frac{\|\mathbf{r}\|}{\|A\|} \leq \|\mathbf{e}\| \leq \|A^{-1}\| \|\mathbf{r}\|$$

- Similarly since $\mathbf{Ax} = \mathbf{b}$ and $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ we have:

$$\frac{\|\mathbf{b}\|}{\|A\|} \leq \|\mathbf{x}\| \leq \|A^{-1}\| \|\mathbf{b}\|$$

- These 2 double inequalities imply:

$$\frac{1}{\|A\| \|A^{-1}\|} \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \leq \frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq \|A\| \|A^{-1}\| \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

- That inequality relates the relative error to the relative residual with a scaling factor. We define that scaling factor to be the condition number:

$$\text{cond}(\mathbf{A}) = \|\mathbf{A}\| \|\mathbf{A}^{-1}\|$$

- This gives:

$$\frac{1}{\text{cond}(A)} \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|} \leq \frac{\|\mathbf{e}\|}{\|\mathbf{x}\|} \leq \text{cond}(A) \frac{\|\mathbf{r}\|}{\|\mathbf{b}\|}$$

- When $\text{cond}(\mathbf{A})$ is close to 1 the relative error and relative residual are close so the matrix is well-conditioned.
- In our previous example $\text{cond}(\mathbf{A})$ was $2e8$ so the relative error in \mathbf{x} could be up to 200 million times larger than the relative change in the matrix system (\mathbf{A} or \mathbf{b}). Hence ill-conditioned.

EXAMPLE 4 Use the ∞ -norm to calculate the condition number of the following matrix.

$$A = \begin{pmatrix} 3 & 3 \\ 4 & 5 \end{pmatrix}$$

□ Find the inverse:

$$A^{-1} = \frac{1}{3} \begin{pmatrix} 5 & -3 \\ -4 & 3 \end{pmatrix}$$

□ We have:

$$\|A\|_{\infty} = 9 \quad \|A^{-1}\|_{\infty} = 8/3 \quad \longrightarrow \quad \text{cond}(A) = 24$$

15.7 Eigenvalues/Eigenvectors

- ❑ Certain vectors only get stretched or compressed when multiplied by a matrix. These are called **eigenvectors**.

Solutions to:

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

- ❑ The amount they are stretched/compressed by are called the **eigenvalues, λ** .
- ❑ It's important to note that **eigenvectors do not change direction** upon multiplication by the matrix.
- ❑ One of their primary uses involves solving systems of ODEs (stability of equilibrium points).
- ❑ They often represent key values in physical systems such as resonant frequencies of structures.

- ❑ Additionally they can be **used for efficient data compression** as well as more efficient machine learning algorithms.
- ❑ They do this by storing matrix entries as special sums involving the eigenvalues, then identifying and throwing away less important data (**principal component analysis**), followed by retrieving the approximated data back using the special sum.
- ❑ Eigenvalues can be computed in an algebraic way but it is costly. Instead it is more common to use a **matrix factorisation to approximate them for large data sets**.
- ❑ To understand why the methods work and also the relationship between eigenvalues/eigenvectors and the systems they represent we will look at a simple system of differential equations in the next example.

EXAMPLE 5 Take a population of 10,000 mobile phone users split between Apple and Android.

For this population, each year 30% of Apple users convert to Android and 20% of Android users convert to Apple.

A simple system of linear ordinary differential equations models the situation:

$$\begin{aligned}\dot{x} &= -0.3x + 0.2y \\ \dot{y} &= 0.3x - 0.2y\end{aligned}$$

where x is No. of Apple users and y is No. of Android users.

- ❑ Let's start off with 8000 Apple users and 2000 Android users.
- ❑ Will the market continuously change year-by-year or will it eventually settle down (approach a steady state)?

- Now it should be easy to see that there is a **steady state whenever $x = 2y/3$** . For our example the market would settle down to 4000 Apple users and 6000 Android users.
- But let's see how this relates to eigenvalues/eigenvectors by using a matrix that represents the transformation of the market after each year:

$$\mathbf{x}_{n+1} = \mathbf{A}\mathbf{x}_n$$

$$\begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix} = \begin{pmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{pmatrix} \begin{pmatrix} x_n \\ y_n \end{pmatrix}$$

- Now if we start with the vector $\mathbf{x}_0 = (8000, 2000)^\top$ we have:

$$\mathbf{x}_1 = \begin{pmatrix} 6000 \\ 4000 \end{pmatrix} \quad \mathbf{x}_2 = \begin{pmatrix} 5000 \\ 5000 \end{pmatrix} \quad \dots \quad \mathbf{x}_5 = \begin{pmatrix} 4125 \\ 5875 \end{pmatrix} \quad \dots \quad \mathbf{x}_{17} = \begin{pmatrix} 4000 \\ 6000 \end{pmatrix} \\ \dots \quad \mathbf{x}_{50} = \begin{pmatrix} 4000 \\ 6000 \end{pmatrix}$$

- ❑ So after applying the matrix multiplication a certain number of times the vector converged to the steady-state vector.
- ❑ Now if we look at the eigenvalues and eigenvectors in Matlab we see:


```
>> [V,L] = eig(A)
```

```
V =
```

```
   -0.70711   -0.5547  
    0.70711   -0.83205
```

```
L =
```

```
    0.5    0  
    0    1
```



```
>> rats(V)
```

```
ans =
```

```
2x28 char array
```

```
' -408/577   -360/649 '
```

```
'  408/577   -540/649 '
```

```
>> V(:,1) = V(:,1)/(-408/577)
```

```
V =
```

```
    1   -0.5547  
   -1   -0.83205
```

```
>> V(:,2) = V(:,2)/(-180/649)
```

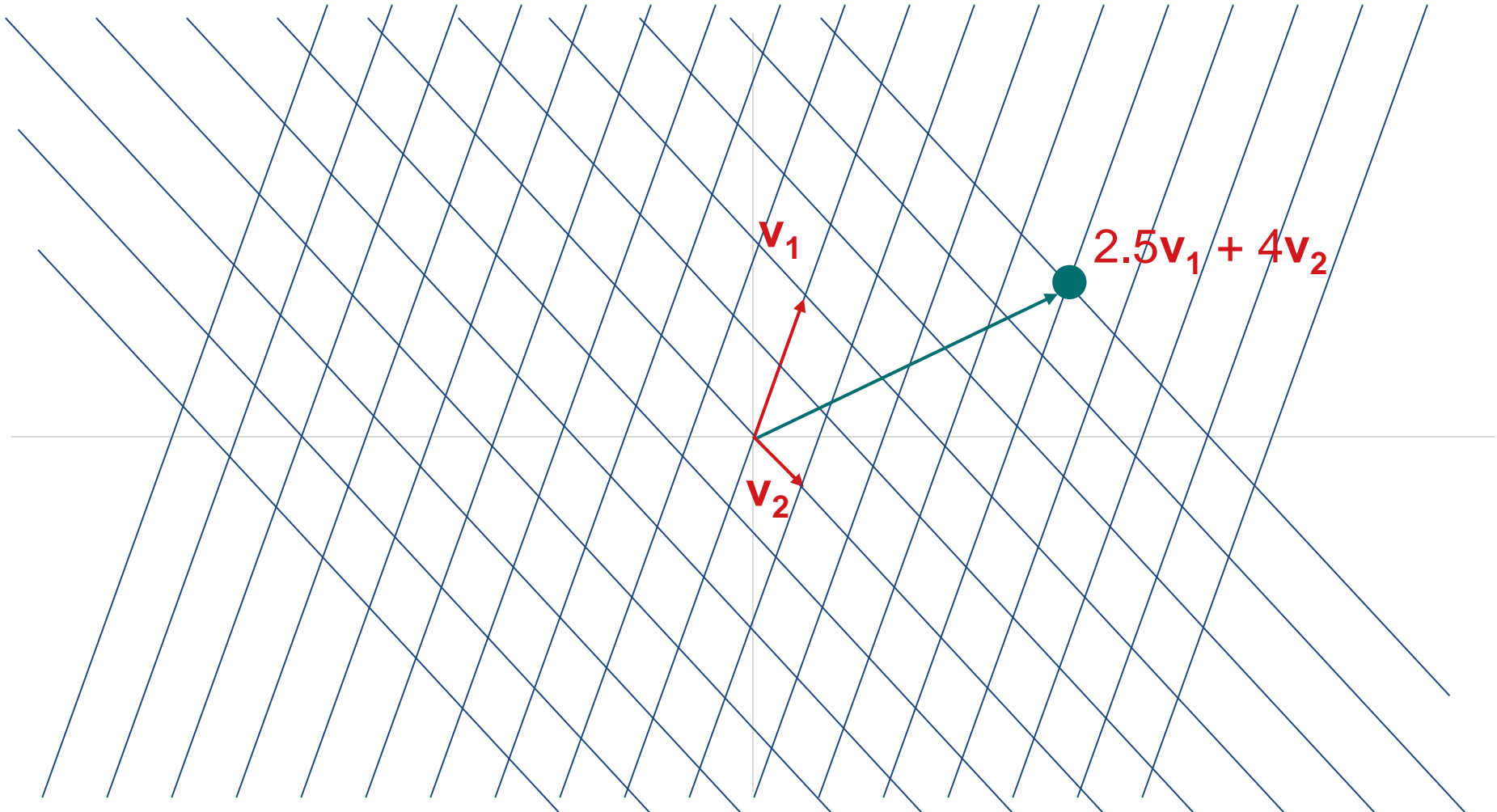
```
V =
```

```
    1    2  
   -1    3
```



- ❑ The eigenvector $(2, 3)^T$ has associated eigenvalue 1.
- ❑ This means that any multiple of $(2, 3)^T$ will stay the same (steady-state vector).
- ❑ Since $(4000, 6000)^T$ is a multiple it is a steady-state vector of the system.
- ❑ Notice that we found the steady-state vector by taking a high enough n and calculating $A^n \mathbf{x}$.
- ❑ How does this relate to the other eigenvector $(1, -1)^T$ with eigenvalue $\frac{1}{2}$?

- Since the eigenvectors don't change direction (only stretch or compress) we can **use them as a basis** for all 2D vectors (they will be like our coordinate axes).



- This allows us to express any vector as a **linear combination** of the eigenvectors.
- Now let's take a **random starting vector**: $\mathbf{z}_0 = (z_1, z_2) = \alpha \mathbf{v}_1 + \beta \mathbf{v}_2$
- Applying the **matrix transformation n times** gives:

$$\mathbf{z}_n = \mathbf{A}^n \mathbf{z}_0 = \alpha \mathbf{A}^n \mathbf{v}_1 + \beta \mathbf{A}^n \mathbf{v}_2$$
- The **second term tends to 0 as $n \rightarrow \infty$** and the **first term tends to $\alpha \mathbf{v}_1$** which is an eigenvector representing the steady-state vector of the system.
- This process indicates the relevance of raising matrices to high powers and hints at a process to retrieve eigenvalues from a matrix.

□ Note that **Example 5** is an example of a **Markov chain**.

15.8 Similar Matrices

- ❑ **Similar matrices** obey the following relationship: $\mathbf{B} = \mathbf{M}^{-1}\mathbf{A}\mathbf{M}$
- ❑ You can think about \mathbf{A} and \mathbf{B} representing linear transformations with respect to different coordinate vectors (**bases**).
- ❑ The matrix \mathbf{M} is known as the **transition matrix** which converts coordinates in the space between the bases.
- ❑ Intuitively the matrices are similar because they represent the same transformation but in different coordinate systems.
- ❑ An important result is that **similar matrices have the same eigenvalues**.

- Eigenvalues are traditionally solved just using the definition which we can use to prove the previous result:

λ comes from solving: $|A - \lambda I| = 0$

For similar matrices $B = S^{-1}AS$:

$$\begin{aligned} |B - \lambda I| &= |S^{-1}AS - \lambda I| \\ &= |S^{-1}(A - \lambda I)S| \\ &= |S^{-1}| |A - \lambda I| |S| \\ &= |A - \lambda I| \end{aligned}$$

15.9 Diagonalisation

- If a **matrix is similar to a diagonal matrix** then we say it is diagonalisable. This allows us to easily raise matrices to high powers which **significantly lowers no. of flops**:

$$D^k = \begin{bmatrix} 5^k & 0 \\ 0 & 3^k \end{bmatrix} \quad \text{for } k \geq 1$$

$$\longrightarrow A^2 = (PDP^{-1})(PDP^{-1}) = PD(P^{-1}P)DP^{-1} = PDDP^{-1}$$

$$= PD^2P^{-1} = \begin{bmatrix} 1 & 1 \\ -1 & -2 \end{bmatrix} \begin{bmatrix} 5^2 & 0 \\ 0 & 3^2 \end{bmatrix} \begin{bmatrix} 2 & 1 \\ -1 & -1 \end{bmatrix}$$

$$\longrightarrow A^3 = (PDP^{-1})A^2 = (PDP^{-1})PD^2P^{-1} = PDD^2P^{-1} = PD^3P^{-1} \quad \text{etc.}$$

- ❑ Note that a matrix is **diagonalisable** if and only if it is square ($n \times n$) and has **exactly n linearly independent eigenvectors** (they are not linear combinations of each other).
- ❑ We use the eigenvectors as a basis and construct the diagonalisation from the **eigenvectors as columns of the \mathbf{P} -matrix** and the **eigenvalues** are used for the leading diagonal of **\mathbf{D}** .
- ❑ Note that **calculating eigenvalues is generally a costly process** so this method is only worth it if you require their use multiple times. This is generally the case in numerical methods and therefore efficient ways of calculating eigenvalues and eigenvectors are needed.

EXAMPLE 6 Raise the following matrix to the power 20 using diagonalisation.

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

```
A = [ 1 3 3; -3 -5 -3; 3 3 1 ];
```

```
[P,L] = eig(A);
```

```
A20 = P*L^20*inv(P);
```

```
disp('P = '); disp(P)
```

```
disp('L = '); disp(L)
```

```
disp('A^20 = '); disp(A20)
```

```
>> P =  
-0.5774 -0.7876 0.4206  
0.5774 0.2074 -0.8164  
-0.5774 0.5802 0.3957
```

```
>> L =  
1.0000 0 0  
0 -2.0000 0  
0 0 -2.0000
```

```
>> A^20 =  
1.0e+06 *  
  
0.0000 -1.0486 -1.0486  
1.0486 2.0972 1.0486  
-1.0486 -1.0486 0.0000
```

- ❑ Later we will see that it is beneficial to diagonalise matrices while **making \mathbf{P} orthogonal** (it makes the method more **numerically stable**).
- ❑ We then call this the **eigenvalue decomposition** of **\mathbf{A}** .
- ❑ If we want that to be the case we can only **diagonalise as above** **if \mathbf{A} is symmetric**.
- ❑ To achieve this we apply the **Gram-Schmidt process** to the eigenvectors of the symmetric matrix.

- ❑ The Gram-Schmidt process finds an orthonormal basis for a matrix:

The Gram–Schmidt Process

To convert a basis $\{\mathbf{u}_1, \mathbf{u}_2, \dots, \mathbf{u}_r\}$ into an orthogonal basis $\{\mathbf{v}_1, \mathbf{v}_2, \dots, \mathbf{v}_r\}$, perform the following computations:

Step 1. $\mathbf{v}_1 = \mathbf{u}_1$

Step 2. $\mathbf{v}_2 = \mathbf{u}_2 - \frac{\langle \mathbf{u}_2, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1$

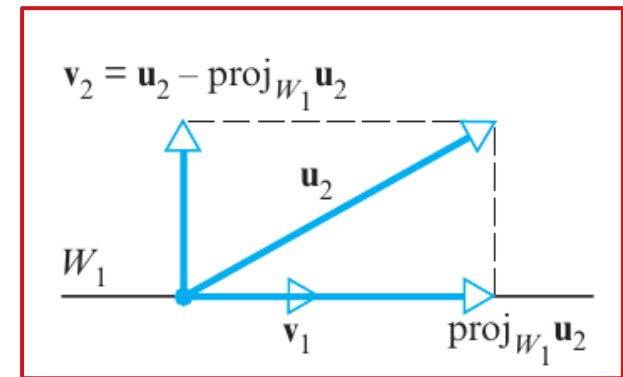
Step 3. $\mathbf{v}_3 = \mathbf{u}_3 - \frac{\langle \mathbf{u}_3, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1 - \frac{\langle \mathbf{u}_3, \mathbf{v}_2 \rangle}{\|\mathbf{v}_2\|^2} \mathbf{v}_2$

Step 4. $\mathbf{v}_4 = \mathbf{u}_4 - \frac{\langle \mathbf{u}_4, \mathbf{v}_1 \rangle}{\|\mathbf{v}_1\|^2} \mathbf{v}_1 - \frac{\langle \mathbf{u}_4, \mathbf{v}_2 \rangle}{\|\mathbf{v}_2\|^2} \mathbf{v}_2 - \frac{\langle \mathbf{u}_4, \mathbf{v}_3 \rangle}{\|\mathbf{v}_3\|^2} \mathbf{v}_3$

\vdots

(continue for r steps)

Optional Step. To convert the orthogonal basis into an orthonormal basis $\{\mathbf{q}_1, \mathbf{q}_2, \dots, \mathbf{q}_r\}$, normalize the orthogonal basis vectors.



EXAMPLE 7 Find the eigenvalue decomposition of the following symmetric matrix.

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

```
A = [ 0 2 -1; 2 3 -2; -1 -2 0];
```

```
>> L = eig(A)
```

```
L =
```

```
-1.0000
```

```
-1.0000
```

```
5.0000
```

$\lambda = 1$:

```
>> rref(A+eye(3))
```

```
ans =
```

```
1    2   -1
```

```
0    0    0
```

```
0    0    0
```

→ $\mathbf{x}_1 = (1, 0, 1)^T$

→ $\mathbf{x}_2 = (-2, 1, 0)^T$

$\lambda = 5$:

```
>> rref(A-5*eye(3))
```

```
ans =
```

```
1    0    1
0    1    2
0    0    0
```

→ $\mathbf{x}_3 = (-1, -2, 1)^T$

(orthogonal to \mathbf{u}_1 and \mathbf{u}_2)

Gram-Schmidt

$$\mathbf{u}_1 = \frac{1}{\|\mathbf{x}_1\|} \mathbf{x}_1 = \frac{1}{\sqrt{2}}(1, 0, 1)^T$$

$$\mathbf{p} = (\mathbf{x}_2^T \mathbf{u}_1) \mathbf{u}_1 = -\sqrt{2} \mathbf{u}_1 = (-1, 0, -1)^T$$

$$\mathbf{x}_2 - \mathbf{p} = (-1, 1, 1)^T$$

$$\mathbf{u}_2 = \frac{1}{\|\mathbf{x}_2 - \mathbf{p}\|} (\mathbf{x}_2 - \mathbf{p}) = \frac{1}{\sqrt{3}}(-1, 1, 1)^T$$

→
$$\mathbf{u}_3 = \frac{1}{\|\mathbf{x}_3\|} \mathbf{x}_3 = \frac{1}{\sqrt{6}}(-1, -2, 1)^T$$

$$\mathbf{A} = \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{6}} \\ 0 & \frac{1}{\sqrt{3}} & -\frac{2}{\sqrt{6}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{6}} \end{pmatrix} \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 5 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & 0 & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} & \frac{1}{\sqrt{3}} \\ -\frac{1}{\sqrt{6}} & -\frac{2}{\sqrt{6}} & \frac{1}{\sqrt{6}} \end{pmatrix}$$

15.10 QR Algorithm

- ❑ The QR matrix factorisation is used in the QR algorithm for solving linear systems and estimating eigenvalues of a matrix.
- ❑ If \mathbf{A} is an $m \times n$ matrix with linearly independent columns then it can be factored as,

$$\mathbf{A} = \mathbf{QR}$$

where \mathbf{Q} is a matrix ($m \times n$) whose columns form an orthonormal basis for $\text{Col } \mathbf{A}$ and \mathbf{R} is an upper triangular matrix ($n \times n$) with positive diagonal entries.

- ❑ This relationship is known as a **QR**-factorisation and can be seen through the Gram-Schmidt process. Let's use the process to make an orthonormal basis from the columns of \mathbf{A} .

- Writing the orthonormal vectors as \mathbf{q}_i , a common linear algebra result is that any vector in a given inner product space can be written as:

$$\mathbf{u} = \langle \mathbf{u}, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}, \mathbf{q}_2 \rangle \mathbf{q}_2 + \dots$$

so we can write,

$$A = [\mathbf{u}_1 \mid \mathbf{u}_2 \mid \cdots \mid \mathbf{u}_n] \quad \text{and} \quad Q = [\mathbf{q}_1 \mid \mathbf{q}_2 \mid \cdots \mid \mathbf{q}_n]$$

with,

$$\begin{aligned} \mathbf{u}_1 &= \langle \mathbf{u}_1, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_1, \mathbf{q}_2 \rangle \mathbf{q}_2 + \cdots + \langle \mathbf{u}_1, \mathbf{q}_n \rangle \mathbf{q}_n \\ \mathbf{u}_2 &= \langle \mathbf{u}_2, \mathbf{q}_1 \rangle \mathbf{q}_1 + \langle \mathbf{u}_2, \mathbf{q}_2 \rangle \mathbf{q}_2 + \cdots + \langle \mathbf{u}_2, \mathbf{q}_n \rangle \mathbf{q}_n \\ \vdots & \quad \quad \quad \vdots \quad \quad \quad \vdots \quad \quad \quad \vdots \end{aligned}$$

- The last equation can be written as:

$$[\mathbf{u}_1 \mid \mathbf{u}_2 \mid \cdots \mid \mathbf{u}_n] = [\mathbf{q}_1 \mid \mathbf{q}_2 \mid \cdots \mid \mathbf{q}_n] \begin{bmatrix} \langle \mathbf{u}_1, \mathbf{q}_1 \rangle & \langle \mathbf{u}_2, \mathbf{q}_1 \rangle & \cdots & \langle \mathbf{u}_n, \mathbf{q}_1 \rangle \\ \langle \mathbf{u}_1, \mathbf{q}_2 \rangle & \langle \mathbf{u}_2, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{u}_n, \mathbf{q}_2 \rangle \\ \vdots & \vdots & & \vdots \\ \langle \mathbf{u}_1, \mathbf{q}_n \rangle & \langle \mathbf{u}_2, \mathbf{q}_n \rangle & \cdots & \langle \mathbf{u}_n, \mathbf{q}_n \rangle \end{bmatrix}$$

- Since each \mathbf{q}_k is orthogonal to each \mathbf{u}_m where $k > m$ it follows that the matrix on the right is an upper triangular matrix (\mathbf{R}):

$$\mathbf{R} = \begin{bmatrix} \langle \mathbf{u}_1, \mathbf{q}_1 \rangle & \langle \mathbf{u}_2, \mathbf{q}_1 \rangle & \cdots & \langle \mathbf{u}_n, \mathbf{q}_1 \rangle \\ 0 & \langle \mathbf{u}_2, \mathbf{q}_2 \rangle & \cdots & \langle \mathbf{u}_n, \mathbf{q}_2 \rangle \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \langle \mathbf{u}_n, \mathbf{q}_n \rangle \end{bmatrix}$$

- ❑ To approximate the eigenvalues of a matrix using QR factorisation we write $A = Q_1 R_1$ then $A_1 = R_1 Q_1$.

- ❑ Since,

$$A = R_1^{-1} (R_1 Q_1) R_1$$

it means that A is similar to A_1 .

- ❑ If 2 matrices are similar then they have the same eigenvalues.

- ❑ Now we write $A_1 = R_1 Q_1 = Q_2 R_2$ as a new QR factorisation.

- ❑ Defining $A_2 = R_2 Q_2$ means that A_1 is also similar to A_2 . We have,

$$A = R_1^{-1} (R_2^{-1} (R_2 Q_2) R_2) R_1$$

- ❑ Continuing we have,

$$A = R_1^{-1} R_2^{-1} R_3^{-1} (R_3 Q_3) R_3 R_2 R_1$$

- ❑ Rearranging gives,

$$R_3 R_2 R_1 A R_1^{-1} R_2^{-1} R_3^{-1} = R_3 Q_3 = A_3$$

- ❑ We could **continue this for n -steps** with A having the same eigenvalues as A_n .
- ❑ Also recall that **the product of 2 triangular matrices is a triangular matrix.**
- ❑ Since all of the **R 's and R^{-1} 's are upper triangular**, under general conditions, **the product on the left will approach upper triangular as n becomes large.**
- ❑ The **eigenvalues of an upper triangular matrix are just the diagonal entries** and so we can approximate the eigenvalues of A by reading off the diagonal entries of A_n .

EXAMPLE 8 Use a basic QR algorithm in Matlab to approximate the eigenvalues of,

$$A = \begin{pmatrix} 9 & 10 & 8 \\ 6 & 3 & 4 \\ 6 & 8 & 6 \end{pmatrix}$$

- The eigenvalues can be calculated simply in Matlab using the eig function:

```
>> eig(A)
```

```
ans =
```

```
20.031793877299595  
0.138039991486861  
-2.169833868786456
```

- We can compare the result above with the QR algorithm.

□ A basic script for doing this is shown below:

```
n = 4;  
A(:,:,1) = [9 10 8; 6 3 4; 6 8 6];
```

Notice we stored each calculation
in a multidimensional array

```
[Q(:,:,1),R(:,:,1)] = qr(A(:,:,1));
```

```
for k = 2:n
```

```
    [Q(:,:,k),R(:,:,k)] = qr(A(:,:,k-1));
```

```
    A(:,:,k) = R(:,:,k)*Q(:,:,k);
```

```
    disp(A(:,:,end))
```

```
end
```

- The results are shown below and the **matrix is tending towards an upper triangular matrix** with **diagonal entries** that are **tending towards the previously calculated eigenvalues** after just 4 iterations.

```
20.294117647058826  2.422513263298084 -2.867297488414613
-2.514217235410313 -2.386944651278234  2.005779852621150
-0.063017527217904  0.065494852330486  0.092827004219409
```

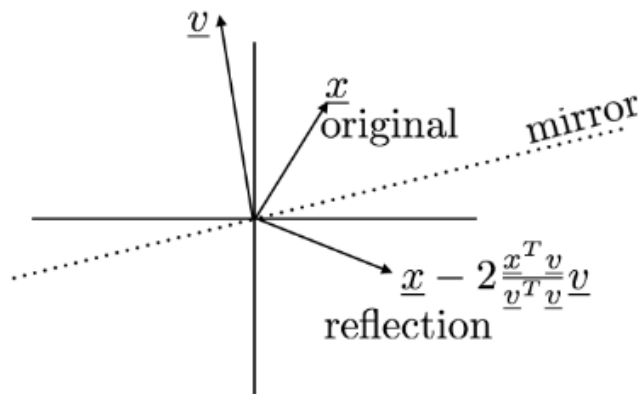
```
19.972007314671544  5.295810763656401  2.844079028771228
0.249751065337936 -2.113502052849184 -1.560517143178407
0.000436314395690  0.005042703644795  0.141494738177640
```

```
20.037952617142839  5.012409279886520 -2.835248644194296
-0.027288278771855 -2.175776765434255  1.601195951008108
-0.000003010719525  0.000311596935312  0.137824148291422
```

```
20.031122973214373  5.043059859944603  2.836697358198402
0.002953496035521 -2.169176724435577 -1.597001113141804
0.000000020742675  0.000019881133105  0.138053751221212
```

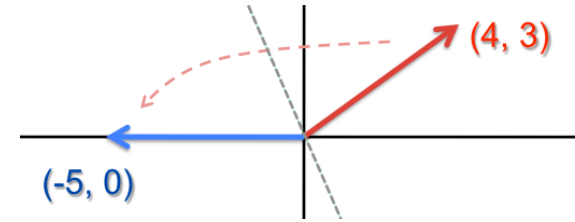
- The number of iterations required for convergence (if it converges at all) will vary depending on the matrix in question.

- ❑ Now in reality the Gram-Schmidt process is quite computationally expensive and also numerically unstable. We can calculate **Q** and **R** using what are known as **Householder transformations**.
- ❑ This transformation operates on the columns of **A** to produce 0's underneath the leading diagonal entries by performing an appropriate reflection (**orthogonal transformation**).
- ❑ Let the first column of **A** be called **x**.



$$\underline{x} - 2 \frac{\underline{v}^T \underline{x}}{\underline{v}^T \underline{v}} \underline{v} = \underbrace{\left(I - 2 \frac{\underline{v} \underline{v}^T}{\underline{v}^T \underline{v}} \right)}_{\text{Householder reflection}} \underline{x}$$

- ❑ Consider the reflection shown here:



- ❑ To ensure the previous formula creates 0's in the column vector we are looking for a reflection that gives a vector along the direction of the first unit vector since all other components will be 0 (notice the length is preserved).
- ❑ We accomplish this by setting: $\mathbf{v} = \mathbf{x} - \|\mathbf{x}\|_2 \mathbf{e}_1$
where $\mathbf{e}_1 = (1, 0, 0, \dots, 0)^T$.
- ❑ Now since we could equally reflect onto $-\mathbf{e}_1$, we could have also had: $\mathbf{v} = \mathbf{x} + \|\mathbf{x}\|_2 \mathbf{e}_1$.
- ❑ During the factoring process we can mitigate numerical round-off errors by setting:

$$\mathbf{v} = \mathbf{x} - \text{sign}(x_1) \|\mathbf{x}\|_2 \mathbf{e}_1$$

- ❑ We calculate the **Householder matrix** (Slide 46) then **multiply it by A** to obtain the transformation. The result will be a **column of 0's under the first entry** of the first column.
- ❑ We then **take a sub-matrix from A_{22} and repeat the process**. At the end of this we will have the **R** matrix.
- ❑ Multiplying the Householder matrices together will give the **Q** matrix however this is not typically needed as the main purpose is to obtain the **eigenvalues** and these **are retrieved from R** .
- ❑ An **alternative** transformation can be used (**Givens rotation**) which performs a similar task however it creates 0-entries one-at-a-time rather than zeroing all entries beneath the top entry in the column in one go. This can be advantageous when **parallelising** the code.

EXAMPLE 9 Use Householder transformations to find the QR factorisation of:

$$\mathbf{A} = \begin{pmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{pmatrix}$$

□ Calculate the reflector: $\mathbf{v}_1 = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - 2 \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$

□ Get Householder matrix (**notice it is symmetric**):

$$\mathbf{H}_1 = \begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{pmatrix} - \frac{2}{4} \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1/2 & 1/2 & 1/2 & 1/2 \\ 1/2 & 1/2 & -1/2 & -1/2 \\ 1/2 & -1/2 & 1/2 & -1/2 \\ 1/2 & -1/2 & -1/2 & 1/2 \end{pmatrix}$$

- Compute transformed matrix:

$$\begin{aligned}\mathbf{A}^{(1)} = \mathbf{H}_1 \mathbf{A}^{(0)} = \mathbf{A}^{(0)} - \frac{1}{2} \mathbf{v} \mathbf{v}^T \mathbf{A}^{(0)} &= \begin{pmatrix} 1 & -1 & 4 \\ 1 & 4 & -2 \\ 1 & 4 & 2 \\ 1 & -1 & 0 \end{pmatrix} - \frac{1}{2} \begin{pmatrix} -1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} 2 & 8 & -4 \end{pmatrix} \\ &= \begin{pmatrix} 2 & 3 & 2 \\ 0 & 0 & 0 \\ 0 & 0 & 4 \\ 0 & -5 & 2 \end{pmatrix}\end{aligned}$$

- Repeat for the sub-matrix:

$$\begin{pmatrix} 0 & 0 \\ 0 & 4 \\ -5 & 2 \end{pmatrix}$$

- The reflector is: $\mathbf{v}_2 = \begin{pmatrix} 0 \\ 0 \\ -5 \end{pmatrix} - \begin{pmatrix} 5 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} -5 \\ 0 \\ -5 \end{pmatrix}$

□ We have:

$$\mathbf{I} - 2 \frac{\mathbf{v}_2 \mathbf{v}_2^T}{\mathbf{v}_2^T \mathbf{v}_2} = \begin{pmatrix} 1 & & \\ & 1 & \\ & & 1 \end{pmatrix} - \frac{1}{25} \begin{pmatrix} -5 \\ 0 \\ -5 \end{pmatrix} \begin{pmatrix} -5 & 0 & -5 \end{pmatrix} = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ -1 & 0 & 0 \end{pmatrix}$$

□ Now we **construct the next Householder matrix** in such a way that it **matches the original dimensions of \mathbf{A}** but **leaves the top entry of the 2nd column unaffected** and transforms the **terms beneath it into 0's**:

$$\mathbf{H}_2 = \begin{pmatrix} 1 & \\ & \mathbf{I} - 2 \frac{\mathbf{v}_2 \mathbf{v}_2^T}{\mathbf{v}_2^T \mathbf{v}_2} \end{pmatrix}$$

□ **Practically we don't need to calculate the above**, just **use** the result of the **lower block matrix** to calculate the next iteration of \mathbf{A} :

$$\left(\mathbf{I} - 2 \frac{\mathbf{v}_2 \mathbf{v}_2^T}{\mathbf{v}_2^T \mathbf{v}_2} \right) \begin{pmatrix} 0 & 0 \\ 0 & 4 \\ -5 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 4 \\ -5 & 2 \end{pmatrix} - \frac{1}{25} \begin{pmatrix} -5 \\ 0 \\ -5 \end{pmatrix} \begin{pmatrix} 25 & -10 \end{pmatrix} = \begin{pmatrix} 5 & -2 \\ 0 & 4 \\ 0 & 0 \end{pmatrix}$$

- ❑ Technically the matrix below comes from the product with \mathbf{H}_2 but it's probably easier to think about just slotting in a lower dimensional Householder transformation into the sub-matrix position we extracted it from.

$$\mathbf{A}^{(2)} = \mathbf{H}_2 \mathbf{A}^{(1)} = \begin{pmatrix} 2 & 3 & 2 \\ 0 & 5 & -2 \\ 0 & 0 & 4 \\ 0 & 0 & 0 \end{pmatrix}$$

- ❑ At this point we have arrived at \mathbf{R} and could calculate $\mathbf{Q} = \mathbf{H}_2 \mathbf{H}_1$ if we wanted. That would allow us to solve the system of equations that the matrix represent in a similar way as we did with \mathbf{LU} -factorisation.
- ❑ QR factorisation with Householder transformations takes approximately twice the number of flops as Gaussian elimination for square matrices but is more numerically stable and has the added bonus of taking a convenient form to solve the least squares formula:

$$\mathbf{x} = \mathbf{R}^{-1} \mathbf{Q}^T \mathbf{b}$$

- ❑ To see why methods which involve orthogonal transformations are more numerically stable than Gaussian-based methods consider values of a vector \mathbf{x} which have a round-off error present.

$$\mathbf{x} = \hat{\mathbf{x}} + \mathbf{e}$$

where the hat represents the true value.

- ❑ Applying an orthogonal transformation via matrix multiplication, \mathbf{P} , to the vector \mathbf{x} gives:

$$\|P\mathbf{x} - P\hat{\mathbf{x}}\| = \|\mathbf{x} - \hat{\mathbf{x}}\| = \|\mathbf{e}\|$$

- ❑ This means that the error when transforming the vector with round-off error is the same magnitude as the error in the vector itself.

15.11 Singular Value Decomposition

- ❑ We've seen that orthogonal methods are more numerically stable.
- ❑ So far we have only seen how to orthogonally diagonalise a symmetric matrix (**eigenvalue decomposition**) and how to deal with general matrices orthogonally (**QR decomposition**).
- ❑ Other decompositions exist for specific situations such as **Cholesky** (low flops but only very specific matrices such as the covariance matrix), **Hessenberg** and **Schur** decompositions (used in combination methods).
- ❑ The **SVD** is the most popular for **sparse matrices** (such as in web page ranking problems) due to its highly stable nature however it comes at significant computational cost (**$12n^3$**).

- The natural extension of the previously mentioned factorisations would be to consider one in which the **left and right matrices** in the product are **different** to each other but **still orthogonal**.

$$\text{SVD:} \quad \mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where \mathbf{U} and \mathbf{V} are orthogonal matrices and $\mathbf{\Sigma}$ takes the form:

$$\mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{pmatrix} \quad \mathbf{\Sigma}_1 = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{pmatrix}$$

- The leading diagonal are the “**singular values**” in descending order.

- The singular values are the **square roots of the eigenvalues of the matrix $\mathbf{A}^T\mathbf{A}$** .

$$\sigma_j = \sqrt{\lambda_j}$$

- Now **$\mathbf{A}^T\mathbf{A}$ is symmetric** and has the same rank as **\mathbf{A}** :

r = no. of non-zero eigenvalues

So we have:

$$\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r > 0 \quad \text{and} \quad \sigma_{r+1} = \sigma_{r+2} = \cdots = \sigma_n = 0$$

- Construct matrices with the eigenvectors: $\mathbf{V} = [\mathbf{V}_1 \ \mathbf{V}_2]$

$$\mathbf{V}_1 = (\mathbf{v}_1, \dots, \mathbf{v}_r), \quad \mathbf{V}_2 = (\mathbf{v}_{r+1}, \dots, \mathbf{v}_n)$$

- The columns of \mathbf{V}_2 form an orthonormal basis for the null spaces of both **$\mathbf{A}^T\mathbf{A}$** and **\mathbf{A}** so: $\mathbf{A}\mathbf{V}_2 = \mathbf{0}$

- This leads to:

$$\mathbf{A} = \mathbf{A}\mathbf{V}_1\mathbf{V}_1^T$$

- Now we find the \mathbf{U} matrix by comparing both sides of $\mathbf{A}\mathbf{V}_1 = \mathbf{U}_1\mathbf{\Sigma}$ (using $\mathbf{U} = [\mathbf{U}_1 \ \mathbf{U}_2]$):

$$\mathbf{A}\mathbf{v}_j = \sigma_j\mathbf{u}_j \quad j = 1, \dots, r \quad \longrightarrow \quad \mathbf{u}_j = \frac{1}{\sigma_j}\mathbf{A}\mathbf{v}_j$$

$$\mathbf{U}_1 = (\mathbf{u}_1, \dots, \mathbf{u}_r)$$

- The columns of \mathbf{U}_1 form an orthonormal basis for the column space of \mathbf{A} since each \mathbf{u}_j is in the column space.
- We construct \mathbf{U}_2 (which has dimension $m - r$) by using columns that form an orthonormal basis for the null space since this leads to:

$$\begin{aligned} \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T &= \begin{bmatrix} \mathbf{U}_1 & \mathbf{U}_2 \end{bmatrix} \begin{bmatrix} \mathbf{\Sigma}_1 & \mathbf{O} \\ \mathbf{O} & \mathbf{O} \end{bmatrix} \begin{bmatrix} \mathbf{V}_1^T \\ \mathbf{V}_2^T \end{bmatrix} \\ &= \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T \\ &= \mathbf{A}\mathbf{V}_1\mathbf{V}_1^T \\ &= \mathbf{A} \end{aligned}$$

EXAMPLE 10 Find the SVD of **A**.

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

$$A^T A = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix} \quad \lambda_1 = 4, \lambda_2 = 0 \quad \longrightarrow \quad V = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Only 1 non-zero singular value:

$$\mathbf{u}_1 = \frac{1}{\sigma_1} A \mathbf{v}_1 = \frac{1}{2} \begin{pmatrix} 1 & 1 \\ 1 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ 1 \\ \frac{1}{\sqrt{2}} \\ 0 \end{pmatrix}$$

Other 2 orthogonal vectors come from finding an orthonormal basis for the null space of **A**

$$\mathbf{x}_2 = (1, -1, 0)^T \quad \text{and} \quad \mathbf{x}_3 = (0, 0, 1)^T$$

Normalising:

$$\mathbf{u}_2 = \frac{1}{\|\mathbf{x}_2\|} \mathbf{x}_2 = \left(\frac{1}{\sqrt{2}}, -\frac{1}{\sqrt{2}}, 0 \right)^T$$

$$\mathbf{u}_3 = \mathbf{x}_3 = (0, 0, 1)^T$$

Giving:

$$A = U\Sigma V^T = \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{bmatrix}$$

Necessity of SVD in Numerical Methods

- ❑ A matrix with **no inverse** is called a **singular matrix** and it has a **determinant equal to 0**.
- ❑ If a nonsingular matrix is close to being singular then the matrix behaves computationally like a singular matrix and the solution is not guaranteed to be accurate.
- ❑ We thus define **numerical rank** as:

The **numerical rank** of an $m \times n$ matrix is the number of singular values of the matrix that are greater than a certain tolerance based on the machine epsilon.

- ❑ This is what Matlab calculates with its **rank()** function.

- ❑ Tolerance can be defined as:

$$\sigma_1 \cdot \max(m, n) \cdot \varepsilon$$

where σ_1 is the largest singular value of **A** and ε is the machine epsilon.

or as: `max(size(A))*eps(norm(A))` (Matlab)
etc.

- ❑ Matrices with singular values close to this tolerance are counted as 0 and this **effectively reduces the rank of the matrix**.
- ❑ We call matrices with $\text{rank} < n$ rank-deficient and they have no inverse.
- ❑ In this case we can use a pseudo-inverse to solve rank-deficient problems.

- We will look at the matrix generated in Matlab by `magic(8)`:

```
>> A = magic(8)
```

```
A =
```

```
64  2  3 61 60  6  7 57
 9 55 54 12 13 51 50 16
17 47 46 20 21 43 42 24
40 26 27 37 36 30 31 33
32 34 35 29 28 38 39 25
41 23 22 44 45 19 18 48
49 15 14 52 53 11 10 56
 8 58 59  5  4 62 63  1
```

- The numerical rank is 3 so it is rank-deficient.

- Generating a **b**:

```
>> b = sum(A(1,:))*ones(8,1)
```

```
b =
```

```
260
260
260
260
260
260
260
260
260
```

❑ Solving with the backslash operator gives:

```
>> A\b
```

Warning: Matrix is close to singular or badly scaled. Results may be inaccurate.

RCOND =

2.238603e-34.

```
ans =
```

```
1.0e+15 *
```

```
-2.765996593679859
```

```
-0.0000000000000001
```

```
0.0000000000000004
```

```
2.765505559312085
```

```
2.766651306170230
```

```
-0.0000000000000000
```

```
0.0000000000000001
```

```
-2.766160271802452
```

❑ Checking the solution illustrates the numerical error:

```
>> A(1,:)*ans
```

```
ans =
```

```
256
```

Solving Using Pseudo-Inverse

- A pseudo-inverse captures a certain number of required characteristics of an inverse matrix for a given application but not necessarily all of them.
- We define the **Moore-Penrose pseudo-inverse** to be:

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

where,

$$\Sigma^+ = \left(\begin{array}{c|c} \Sigma_1^{-1} & O \\ \hline O & O \end{array} \right) = \left(\begin{array}{ccc|c} \frac{1}{\sigma_1} & & & \\ & \ddots & & \\ & & \frac{1}{\sigma_r} & \\ \hline & O & & O \end{array} \right)$$

- This allows us to solve least-square problems in the case where \mathbf{A} is singular or close to singular (numerically rank-deficient):

$$\begin{aligned}\mathbf{x} &= (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b} \\ &= \mathbf{V}(\boldsymbol{\Sigma}^T \boldsymbol{\Sigma})^{-1} \mathbf{V}^T \mathbf{V} \boldsymbol{\Sigma}^T \mathbf{U}^T \mathbf{b} \\ &= \mathbf{V}(\boldsymbol{\Sigma}^T \boldsymbol{\Sigma})^{-1} \boldsymbol{\Sigma}^T \mathbf{U}^T \mathbf{b} \\ &= \mathbf{V} \boldsymbol{\Sigma}^+ \mathbf{U}^T \mathbf{b} \\ &= \mathbf{A}^+ \mathbf{b}\end{aligned}$$

- We can also solve non-square systems using the pseudo-inverse.

- ❑ Solving the previous system that gave us a numerical error with the pseudo-inverse gives us an accurate solution:

```
>> x = pinv(A)*b
```

```
x =
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
1.0000
```

```
>> A(1,:)*x
```

```
ans =
```

```
260
```

- Taking the first 6 columns of the previous magic matrix with **b**:

```
>> b = sum(A(1,:))*ones(8,1)
b =
    196
    196
    196
    196
    196
    196
    196
    196
```

gives a consistent system however there are 8 equations with 6 variables meaning it is also rank-deficient.

- Solving again:

```
>> x1 = A\b
Warning: Rank deficient, rank = 3, tol = 1.882938e-13.
```

```
x1 =
    2.2615
    3.0154
         0
         0
    0.7538
         0
```

- ❑ This time the solution is correct:

```
>> A(1,:)*x1
```

```
ans =
```

```
196
```

- ❑ However if we solve using the pseudo-inverse we get another solution which minimises the 2-norm:

```
>> x2 = pinv(A)*b
```

```
x2 =
```

```
0.8698
```

```
1.1018
```

```
1.0438
```

```
1.0438
```

```
1.1018
```

```
0.8698
```

```
>> A(1,:)*x2
```

```
ans =
```

```
196
```

```
>> norm(x1)
```

```
ans =
```

```
3.8439
```

```
>> norm(x2)
```

```
ans =
```

```
2.4739
```

Data Compression with SVD

- We take the **reduced SVD**: $\mathbf{A} = \mathbf{U}_1 \mathbf{\Sigma} \mathbf{V}_1^T$

and reduce the matrix rank by discarding the singular values lower than a specified amount.

- Multiplying out gives:

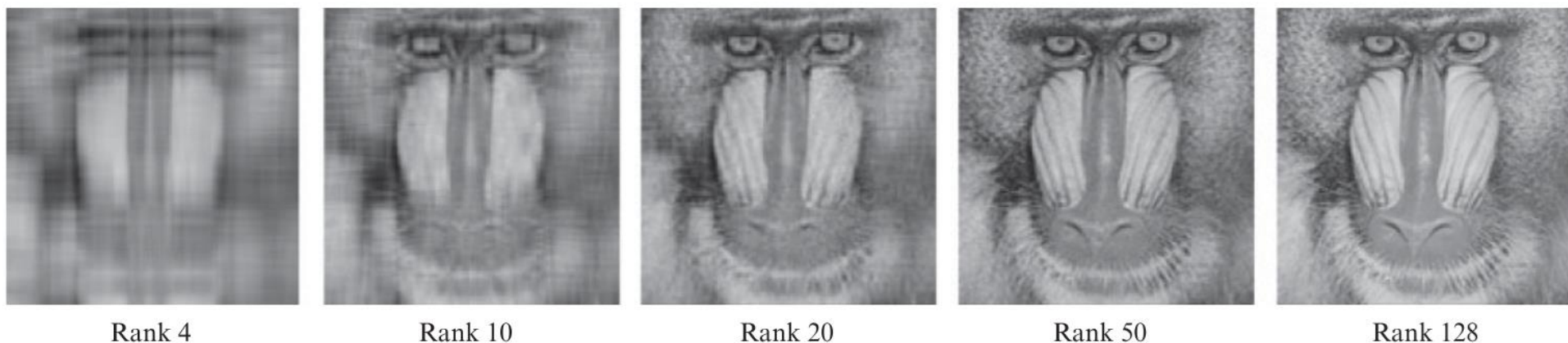
$$\mathbf{A}_k = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \cdots + \sigma_k \mathbf{u}_k \mathbf{v}_k^T$$

where k is the chosen rank.

- It can be shown that the storage required for the above reduced form is equal to:

$$k(m + n + 1)$$

- ❑ For example storing image data as a matrix of 1000×1000 pixels this would usually take $1e6$ numbers to be stored for this data.
- ❑ Performing SVD then choosing a rank of say, 100, we would only need 200,100 numbers ($\sim 80\%$ compression).
- ❑ Below is a grayscale image stored with various ranks of matrix and the consequently recovered data.



Source: *Elementary Linear Algebra* - Anton & Rorres