Module 3M1: Mathematical Methods
Linear Algebra

Garth N. Wells

gnw20@cam.ac.uk

Lent Term 2019

4th February 2019

Contents

1 Introduction

These notes cover four lectures and are just a taste of linear algebra. A number of topics that are important in scientific computing and engineering are touched upon, but there is much more left untold!

Task: Review carefully the linear algebra from Part IB.

Books

There are many books on linear algebra. A gentle text is:

- Strang, G. (2006) *Linear Algebra and its Applications*.

An excellent textbook is

- Trefethen, L.N. and Bau, D (1997) *Numerical Linear Algebra*, SIAM.

Formal mathematical analysis of some of the topics covered can be found in:

- Süli, E. and Mayers, D. (2006) *An Introduction to Numerical Analysis*, Cambridge University Press.

Another book which also has a more formal emphasis, but which can be downloaded freely from the author's webpage is:

- Scott, L.R. (2014) *Numerical Analysis*, Princeton University Press. `http://people.cs.uchicago.edu/~ridg/newna/natwo.pdf`

A classic matrix monograph is

- Golub, G.H. and Van Loan, C.F. (2012) *Matrix Computations*, The John Hopkins University Press.

2 Definitions

Linear algebra involves a number of definitions and considerable jargon, and most problems and applications involve a synthesis of the basic concepts/definitions. Some key definitions and concepts that will be built upon in this course are presented in this section. Some concepts will be familiar, and others less so.

2.1 Vector spaces

2.1.1 What is a vector? [slide 2]

A vector $\mathbf{v}$ is an element of a *vector space*. We will define a vector space shortly, but a space is not very intuitive without a concrete example of a vector.

You will be most familiar with vectors from the Euclidean space $\mathbb{R}^n$, which is all vectors with n real components (an n-tuple). For example a vector $\mathbf{x} \in \mathbb{R}^n$ has the form

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^T$$

and a vector $\mathbf{x} \in \mathbb{R}^4$ has the form

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix}^T$$

For generality, we will consider complex problems. A vector $\mathbf{x} \in \mathbb{C}^n$ has the form

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & x_3 & \dots & x_n \end{bmatrix}^T$$

where $x_i \in \mathbb{C}$ (each component is a complex number).

2.1.2 What is a vector space? [slide 3]

A vector space, often denoted by V, is a 'family' of vectors that obey some basic rules. For vectors $\mathbf{u}$, $\mathbf{v}$ and $\mathbf{w}$ in a space V, and scalars r and s (complex), the rules are:

1. $\mathbf{u} + (\mathbf{v} + \mathbf{w}) = (\mathbf{u} + \mathbf{v}) + \mathbf{w}$
2. $\mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$
3. $\mathbf{u} + \mathbf{0} = \mathbf{u}$
4. $\mathbf{u} + (-\mathbf{u}) = \mathbf{0}$
5. $r(s\mathbf{u}) = (rs)\mathbf{u}$
6. $r(\mathbf{u} + \mathbf{v}) = r\mathbf{u} + r\mathbf{v}$
7. $1\mathbf{u} = \mathbf{u}$

Simply put, a vector in V that is multiplied by a scalar is also in V. If the scalar is zero, the resulting vector is zero. Hence, a vector space must always contain a zero vector. The sum of any two vectors in vector space V (possibly multiplied by a scalar) is also in the space.

2.1.3 Vector space examples [slide 4]

Example: real vectors in n dimensions
The vector space $\mathbb{R}^n$ is the space of all real vectors of length n.

Example: polynomials of degree p
All polynomials of degree p in the variable x on the interval $x \in (x_1, x_2)$ form a vector space. For example, if you add any two cubic polynomials, the results is also a cubic polynomial.

2.1.4 What is a subspace? [slide 5]

A subspace of a vector space is a subset that obeys the rules of a vector space.

Example: A subspace of $\mathbb{C}^3$

All vectors of the form:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ 0 \\ x_3 \end{bmatrix},$$

where x_1 and $x3$ are any complex numbers, come from a subspace of $\mathbb{C}^3$. This subspace consists of all vectors that lie in the x_1–x_3 'plane'.

Vectors of the form:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ 1 \\ x_3 \end{bmatrix}$$

do not form a subspace since

$$\mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 + y_1 \\ 2 \\ x_3 + y_3 \end{bmatrix}$$

Moreover, it does not contain the zero vector.

Example: Algebra subspace of polynomials of degree p
Polynomials of degree $p - 1$ form a subspace of the space of all polynomials of degree p. A polynomial of degree p contains all the lower order terms, i.e. a space of all quadratic polynomials contains all linear polynomials. The sum of two linear polynomials is also a linear polynomial.

2.2 Matrices

2.2.1 What is a matrix? [slide 6]

A matrix $\mathbf{A}$ is a linear operator that performs a linear transformation from one vector to another:

$$\mathbf{A}\mathbf{x} = \mathbf{b}.$$

The above maps the vector $\mathbf{x} \in \mathbb{C}^n$ to the vector $\mathbf{b} \in \mathbb{C}^m$. We could also express this as $\mathbf{A} : \mathbb{C}^n \to \mathbb{C}^m$. This says that $\mathbf{A}$ matrix maps a vector of length n to a vector of length m.

A matrix is a rectangular array of numbers. For example, a 2×3 matrix has the form:

$$\mathbf{A} = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}.$$

An $m \times n$ matrix comes from the *space* $\mathbb{C}^{m \times n}$, and we will often write $\mathbf{A} \in \mathbb{C}^{m \times n}$.

A number of operations are defined for matrices, such as addition and multiplication. You should be familiar already with these. One you should memorise is the index notation expression for the product of two matrices $\mathbf{C} = \mathbf{AB}$:

$$C_{ij} = \sum_k A_{ik} B_{kj}.$$

Treating a column vector as an $n \times 1$ matrix, we can use the above to compute $\mathbf{Ax} = \mathbf{b}$:

$$b_i = \sum_j A_{ij} x_j.$$

2.2.2 What is the transpose and conjugate transpose? [slide 7]

The transpose of a matrix will be familiar,

$$\left(\mathbf{A}^T\right)_{ij} = A_{ji},$$

i.e. rows and columns are exchanged. The transpose is not commonly used in the context of complex problems. Common is the *conjugate transpose*:

$$\left(\mathbf{A}^H\right)_{ij} = \bar{A}_{ji},$$

or expressed alternatively,

$$\mathbf{A}^H = \overline{\mathbf{A}^T} = \overline{\mathbf{A}}^T$$

where $\bar{x} \in \mathbb{C}$ is the complex conjugate of x. For example,

$$\begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \end{bmatrix}^H = \begin{bmatrix} \bar{A}_{11} & \bar{A}_{21} \\ \bar{A}_{12} & \bar{A}_{22} \\ \bar{A}_{13} & \bar{A}_{23} \end{bmatrix}$$

Clearly the transpose of the conjugate transpose coincide for real-valued matrices ($\mathbf{A} \in \mathbb{R}^{m \times n}$),

A matrix $\mathbf{M} \in \mathbb{C}^{n \times n}$ is *Hermitian* if

$$\mathbf{M}^H = \mathbf{M}.$$

If the matrix is real-valued, this is the same as the matrix being symmetric. Note that $(\mathbf{AB})^H = \mathbf{B}^H \mathbf{A}^H$. By extension, $\mathbf{A}^H \mathbf{A}$ must be Hermitian.

For real-valued matrices, the distinction between 'H' and 'T' vanishes.

2.2.3 What is the dot/scalar/inner product? [slide 8]

The dot (or scalar) product is an operation between two equal-length vectors that yields a scalar, and is defined by:

$$\mathbf{x}^H \mathbf{y} = \sum_{i=1}^{n} \bar{x}_i y_i.$$

The dot product yields a complex number, and $\mathbf{x}^H \mathbf{y} = \overline{\mathbf{y}^H \mathbf{x}}$.

The notation $\mathbf{x} \cdot \mathbf{y}$ is sometimes used. An ambiguity is that this is sometimes defined to be

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{y}^H \mathbf{x} = \sum_{i=1}^{n} x_i \bar{y}_i,$$

and other times

$$\mathbf{x} \cdot \mathbf{y} = \mathbf{x}^H \mathbf{y} = \overline{\mathbf{y}^H \mathbf{x}} = \sum_{i=1}^{n} \bar{x}_i y_i.$$

Since $\mathbf{x} \cdot \mathbf{y} \neq \mathbf{y} \cdot \mathbf{x}$ in the complex case and due to the ambiguity in the definition, we will not use the '$\cdot$' notation.

The dot/scalar product is sometimes called the *inner product* as it is a inner product, but there are others operations that satisfy the requirements of an inner product.

2.2.4 What is an eigenpair? [slide 9]

For an $n \times n$ matrix $\mathbf{A}$, $(\lambda, \mathbf{x})$ is an eigenpair of $\mathbf{A}$ if

$$\mathbf{A}\mathbf{x} = \lambda \mathbf{x},$$

where λ an eigenvalue of $\mathbf{A}$, and $\mathbf{x}$ is the corresponding eigenvector of $\mathbf{A}$. Recall that λ can be equal to zero, but $\mathbf{x}$ must be nonzero.

Recall that the eigenvalues of $\mathbf{A}^{-1}$ are the reciprocal of the eigenvalues of $\mathbf{A}$.

Note
Revise computation of eigenpairs for small matrices from Part I.

2.2.5 Eigenvalues of a Hermitian matrix [slide 10]

The eigenvalues of a Hermitian matrix are *real*.

The eigenvectors of a Hermitian matrix are *orthogonal*, i.e. $\mathbf{u}_i^H \mathbf{u}_j = \mathbf{u}_j^H \mathbf{u}_i = 0$ when $i \neq j$.

2.2.6 What is a unitary matrix? [slide 11]

A matrix $\mathbf{Q} \in \mathbb{C}^{n \times n}$ is a *unitary* matrix if $\mathbf{Q}^H = \mathbf{Q}^{-1}$, i.e. $\mathbf{Q}^H \mathbf{Q} = \mathbf{Q}\mathbf{Q}^H = \mathbf{I}$. If $\mathbf{Q}$ was real, we would call it an *orthogonal* matrix.

2.2.7 What is a Hermitian positive-definite matrix? [slide 12]

A Hermitian matrix $\mathbf{M} \in \mathbb{C}^{n \times n}$ is *positive definite* if

$$\mathbf{x}^H \mathbf{M} \mathbf{x} > 0 \quad \forall \mathbf{x} \in \mathbb{C}^n \backslash \mathbf{0}. \tag{1}$$

The eigenvalues of a Hermitian positive definite matrix are strictly positive (this is a sufficient condition). Positive-definite matrices are particularly important for quadratic problems, which we will see later.

A Hermitian matrix $\mathbf{M} \in \mathbb{C}^{n \times n}$ is *semi-positive definite* if

$$\mathbf{x}^H \mathbf{M} \mathbf{x} \geq 0 \quad \forall \mathbf{x} \in \mathbb{C}^n,$$

which implies that all eigenvalues are positive.

The matrix $\mathbf{A}^H \mathbf{A}$ is positive semi-definite (see Examples Paper).

2.2.8 What is the rank of a matrix? [slide 13]

The rank of a matrix $\mathbf{A} \in \mathbb{C}^{m \times n}$ is the number of linearly independent rows or columns (the number is equal). It satisfies:

$$\operatorname{rank} \mathbf{A} \leq \min(m, n).$$

A matrix is *full rank* if

$$\operatorname{rank} \mathbf{A} = \min(m, n)$$

and *rank deficient* if

$$\operatorname{rank} \mathbf{A} < \min(m, n).$$

2.2.9 What is a sparse matrix? [slide 14]

A matrix in which most entries are zero is a *sparse matrix*. Typically, the number of non-zeroes on each row of a sparse matrix will be roughly the same and independent of the matrix size. Sparse matrices are very common in science and engineering, and often arise in connection with methods for solving differential equations, such as finite difference and finite element methods.

The number of entries in a dense $n \times n$ matrix is n^2. For a sparse matrix, the number of entries is cn, where c is a small constant (say $c < 30$). For large n the difference in the required storage is substantial.

Exercise
Compare the algorithmic complexity of matrix–vector multiplication for dense and sparse matrices.

2.3 Norms

2.3.1 What is a norm? [slide 15]

A very important concept in linear algebra (and more generally vector spaces) is that of a norm. A norm of an object is a non-negative, real-valued number that is a measure of 'how big' something is and which allows ordering.

Norms that you will already be familiar with are the absolute value of a number, the modulus of a complex number and the Euclidean norm for vectors $\mathbf{x} \in \mathbb{R}^n$:

$$\|\mathbf{x}\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2} = \left(\sum_{i=1}^{n} x_i^2 \right)^{1/2}$$

Denoting a norm of a vector $\mathbf{x}$ by $\|\mathbf{x}\|$ (not necessarily the Euclidean norm), a norm is a scalar that satisfies:

$$\begin{aligned}
&\|\mathbf{x}\| > 0 \quad \text{when } \mathbf{x} \neq \mathbf{0}, \\
&\|\mathbf{x}\| = 0 \quad \text{when } \mathbf{x} = \mathbf{0}, \\
&\|k\mathbf{x}\| = |k|\|\mathbf{x}\|, \\
&\|\mathbf{x} + \mathbf{y}\| \leq \|\mathbf{x}\| + \|\mathbf{y}\| \quad \text{(triangle inequality)}.
\end{aligned}$$

2.3.2 Vector norms [slide 16]

A particular family of norms are known as l_p-norms:

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

(recalling that $|x| = \sqrt{[\mathrm{Re}(x)]^2 + [\mathrm{Im}(x)]^2}$). Commonly considered norms are the l_1 norm:

$$\|\mathbf{x}\|_1 = |x_1| + |x_2| + \ldots + |x_n| = \sum_{i=1}^{n} |x_i|,$$

the l_2 norm (which we have already seen):

$$\|\mathbf{x}\|_2 = \left(|x_1|^2 + |x_2|^2 + \ldots + |x_n|^2 \right)^{1/2} = \left(\sum_{i=1}^{n} |x_i|^2 \right)^{1/2},$$

and the l_∞ norm:

$$\|\mathbf{x}\|_\infty = \max_i |x_i|,$$

which is also known as the maximum norm.

The l_1 norm is sometimes called the 'taxicab norm'. Can you see why?

The Euclidean/l_2 norm is written as $\|\mathbf{x}\|_2$, but since it is so frequently used the subscript '2' is sometimes dropped.

We can define norms that involve a matrix $\mathbf{A}$, subject to some restrictions on the matrix:

$$\|\mathbf{x}\|_A^2 = \langle \mathbf{x}, \mathbf{A}\mathbf{x} \rangle,$$

where $\langle \cdot, \cdot \rangle$ is an inner product. Defining an inner product is not necessary at this stage, other than to say a common case is $\langle \mathbf{u}, \mathbf{v} \rangle = \mathbf{u} \cdot \bar{\mathbf{v}} = \mathbf{v}^H \mathbf{u}$. We will see a common example shortly.

Example: computing different norms for the same vector

Compute the l_1, l_2 and l_∞ norms for the vector

$$\mathbf{x} = \begin{bmatrix} 2 & -3 & 7 & 4 \end{bmatrix}.$$

- l_1 norm: $\|\mathbf{x}\|_1 = |2| + |-3| + |7| + |4| = 16$

- l_2 norm: $\|\mathbf{x}\|_2 = \left(2^2 + 3^2 + 7^2 + 4^2\right)^{1/2} = \sqrt{78} \approx 8.83$
- l_∞ norm: $\|\mathbf{x}\|_\infty = \max_i |x_i| = 7$

2.3.3 Example: different norms measure differently [slide 17]

Consider the two vectors

$$\mathbf{x} = \begin{bmatrix} 3 & 3 & 3 \end{bmatrix}$$

and

$$\mathbf{y} = \begin{bmatrix} 0 & 0 & 9 \end{bmatrix}$$

- Consider first the l_1 norm:
  - $\|\mathbf{x}\|_1 = |3| + |3| + |3| = 9$
  - $\|\mathbf{y}\|_1 = |9| = 9$

  In the l_1 norm $\mathbf{y}$ and $\mathbf{x}$ have the 'same magnitude'.
- Now the l_2 norm:
  - $\|\mathbf{x}\|_2 = \left(3^2 + 3^2 + 3^2\right)^{1/2} = \sqrt{27} \approx 5.20$
  - $\|\mathbf{y}\|_2 = \left(9^2\right)^{1/2} = \sqrt{81} = 9$

  In the l^2 norm $\mathbf{y}$ is 'bigger' than $\mathbf{x}$.
- Now the l_∞ norm:
  - $\|\mathbf{x}\|_\infty = 3$
  - $\|\mathbf{y}\|_\infty = 9$

  In the l_∞ norm $\mathbf{y}$ is three times 'bigger than' $\mathbf{x}$.

Note that $\|\mathbf{x}\|_2^2 = \mathbf{x}^H\mathbf{x}$.

2.3.4 Example: norm induced by a matrix [slide 18]

Linear systems $\mathbf{Ax} = \mathbf{b}$ where the components of $\mathbf{x}$ have units of length and the components of $\mathbf{b}$ have units of force arise often in engineering and physics. It then follows that $\mathbf{x}^H\mathbf{Ax}$ will have units of energy. We can use this to define a norm of $\mathbf{x}$:

$$\|\mathbf{x}\|_A^2 = \mathbf{x}^H\mathbf{Ax}$$

For the above to obey the rules at the start of this section to qualify as a norm, $\mathbf{A}$ must be positive definite. $\mathbf{A}$ being positive definite implies that: (a) the energy is non-negative; and (b) the energy is zero only if $\mathbf{x} = \mathbf{0}$. The above norm is often called the *energy norm*.

2.3.5 Matrix norms [slide 19]

Norms can be defined for matrices, but they are less intuitive (at first) than norms of vectors, and can be more expensive to compute.

2.3.6 Operator norms [slide 20]

A norm of a matrix $\mathbf{A}$ is defined as:

$$\|\mathbf{A}\| = \max_{\mathbf{x} \in \mathbb{C}^n \backslash \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|}{\|\mathbf{x}\|}. \tag{2}$$

This norm measures the 'maximum amount' by which the matrix $\mathbf{A}$ can re-scale a vector $\mathbf{x}$ (in relative terms). From eq. (2), we can write:

$$\|\mathbf{A}\mathbf{x}\| \le \|\mathbf{A}\| \|\mathbf{x}\| \quad \forall \mathbf{x}$$

To quantify the 'size' of the original vector $\mathbf{x}$ and the transformed vector $\mathbf{A}\mathbf{x}$, we need to choose a norm for the vectors.

Matrix norms obey the rules at the start of section, and it follows from the definition that

$$\|\mathbf{A}\mathbf{B}\| \le \|\mathbf{A}\| \|\mathbf{B}\| .$$

Starting with $\|\mathbf{A}\|_1$ (1-norm):

$$\begin{aligned}\|\mathbf{A}\|_1 &= \max_{\mathbf{x} \in \mathbb{C}^n \backslash \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_1}{\|\mathbf{x}\|_1} \\ &= \max_j \sum_{i=1}^n |a_{ij}|,\end{aligned}$$

which is column of $\mathbf{A}$ with the maximum l_1-norm. For $\|\mathbf{A}\|_\infty$ (∞-norm):

$$\begin{aligned}\|\mathbf{A}\|_\infty &= \max_{\mathbf{x} \in \mathbb{C}^n \backslash \mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_\infty}{\|\mathbf{x}\|_\infty} &(3) \\ &= \max_i \sum_j^n |a_{ij}|, &(4)\end{aligned}$$

which is row of $\mathbf{A}$ with the maximum l_1-norm. Proving the above two expressions is a question in the Examples Paper.

For $\|\mathbf{A}\|_2$ (2-norm), we will first square both sides of eq. (2):

$$\begin{aligned}\|\mathbf{A}\|_2^2 &= \max_{\mathbf{x}\in\mathbb{C}^n\setminus\mathbf{0}} \frac{\|\mathbf{A}\mathbf{x}\|_2^2}{\|\mathbf{x}\|_2^2} = \max \frac{\mathbf{x}^H\mathbf{A}^H\mathbf{A}\mathbf{x}}{\mathbf{x}^H\mathbf{x}} = \frac{\mathbf{x}^H\lambda_{\max}(\mathbf{A}^H\mathbf{A})\mathbf{x}}{\mathbf{x}^H\mathbf{x}} \\ &= \lambda_{\max}(\mathbf{A}^H\mathbf{A}),\end{aligned}$$

where $\lambda_{\max}(\mathbf{A}^H\mathbf{A})$ is the largest eigenvalue of $\mathbf{A}^H\mathbf{A}$ (recall that $\mathbf{A}^H\mathbf{A}$ is positive semi-definite, so all eigenvalues are positive). The norm $\|\mathbf{A}\|_2$ is therefore the square root of the largest eigenvalue of $\mathbf{A}^H\mathbf{A}$[1]

In the case that $\mathbf{A}$ is Hermitian, the eigenvalues of $\mathbf{A}^H\mathbf{A}$ are the square of the eigenvalues of $\mathbf{A}$. Hence, if $\mathbf{A}$ is Hermitian $\|\mathbf{A}\|_2 = |\lambda|_{\max}(\mathbf{A})$.

2.3.7 Invariance of the 2-norm under rotation [slide 21]

The vector and matrix 2-norms are invariant under 'rotation', i.e. for $\mathbf{x} \in \mathbb{C}^n$

$$\|\mathbf{Q}\mathbf{x}\|_2 = \|\mathbf{x}\|_2\,,$$

and for $\mathbf{A} \in \mathbb{C}^{n\times m}$

$$\|\mathbf{Q}\mathbf{A}\|_2 = \|\mathbf{A}\|_2\,,$$

where $\mathbf{Q}$ is a unitary matrix.

2.3.8 Frobenius norm [slide 22]

Some matrix norms treat the entries of a matrix like the entries of a vector. One such norm is the Frobenius norm. It is defined by:

$$\|\mathbf{A}\|_F = \sqrt{\sum_i \sum_j |a_{ij}|^2}.$$

The Frobenius norm is also invariant under rotation,

$$\|\mathbf{Q}\mathbf{A}\|_F = \|\mathbf{A}\|_F\,,$$

where $\mathbf{Q}$ is a unitary matrix.

Unless otherwise stated, when referring to matrix norms we mean *operator norms*.

[1]The square-roots of the eigenvalues of $\mathbf{A}^H\mathbf{A}$ are known at the *singular values* of $\mathbf{A}$.

2.3.9 Which norm to choose? [slide 23]

Choosing a norm can depend on what we want to measure (what we want to 'weight' as important), and what fits naturally with a particular algorithm. We will see that some norms are easier to work with than others, so the choice is often pragmatic.

A technical point is that on the vector spaces that we consider, all norm are *equivalent*, which means that there exist constants $c_1 > 0$ and $c_2 > 0$ (but which typically depend on the dimension n) such that:

$$c_1 \|\mathbf{x}\|_\alpha \leq \|\mathbf{x}\|_\beta \leq c_2 \|\mathbf{x}\|_\alpha \quad \forall \mathbf{x} \in V.$$

2.3.10 What is a condition number? [slide 24]

We will consider the significance of the condition number in the next section, but define it here for reference. The condition number of a matrix $\mathbf{A}$ is:

$$\kappa(\mathbf{A}) = \|\mathbf{A}\| \left\| \mathbf{A}^{-1} \right\|.$$

To be concrete, we need to select a norm.

For the 2-norm, we see that

$$\kappa_2(\mathbf{A}) = \frac{\sqrt{\lambda_{\text{max}}(\mathbf{A}^H\mathbf{A})}}{\sqrt{\lambda_{\text{min}}(\mathbf{A}^H\mathbf{A})}}$$

since the eigenvalues of $\mathbf{A}^{-1}$ are the reciprocal of the eigenvalues of $\mathbf{A}$. If $\mathbf{A}$ is Hermitian,

$$\kappa_2(\mathbf{A}) = \frac{|\lambda(\mathbf{A})|_{\text{max}}}{|\lambda(\mathbf{A})|_{\text{min}}}.$$

3 Stability and condition number

3.1 Stability of operations

3.1.1 Stability [slide 25]

Large linear systems of the form $\mathbf{A}\mathbf{x} = \mathbf{b}$ are solved by computers. When solving with a computer, round-off error cannot be avoided. The important question is whether or not round-off errors will have a significant impact on the accuracy of the computed solution. We can bound the error in terms of the condition number.

3.2 Error bounds

3.2.1 Bounding the error when solving linear systems [slide 26]

Consider the problem

$$\mathbf{A}(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}+\delta\mathbf{b},$$

where $\delta\mathbf{b}$ is the error in the RHS and $\delta\mathbf{x}$ is the consequent error in the solution. Since $\mathbf{b} = \mathbf{A}\mathbf{x}$ and $\delta\mathbf{x} = \mathbf{A}^{-1}\delta\mathbf{b}$, we have

$$\|\mathbf{b}\| = \|\mathbf{A}\mathbf{x}\| \le \|\mathbf{A}\|\|\mathbf{x}\| \quad \rightarrow \quad \frac{1}{\|\mathbf{x}\|} \le \frac{\|\mathbf{A}\|}{\|\mathbf{b}\|}$$

and

$$\|\delta\mathbf{x}\| = \left\|\mathbf{A}^{-1}\delta\mathbf{b}\right\| \le \left\|\mathbf{A}^{-1}\right\|\|\delta\mathbf{b}\|.$$

Combining the above inequalities, we have

$$\begin{aligned}\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}\|} &\le \|\mathbf{A}\|\left\|\mathbf{A}^{-1}\right\|\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}\\ &= \kappa(\mathbf{A})\frac{\|\delta\mathbf{b}\|}{\|\mathbf{b}\|}\end{aligned}$$

This shows how an error in **b** can propagate through to the solution **x**. For large condition numbers we can expect *small* errors in **b** to cause *large* errors in **x**

An alternative, and sometimes more relevant scenario, is an error in the matrix **A**:

$$(\mathbf{A}+\delta\mathbf{A})(\mathbf{x}+\delta\mathbf{x}) = \mathbf{b}$$

For example, the term $\delta\mathbf{A}$ could represent the floating point errors introduced when performing an LU decomposition. In this case,

$$\frac{\|\delta\mathbf{x}\|}{\|\mathbf{x}+\delta\mathbf{x}\|} \le \left\|\mathbf{A}^{-1}\right\|\|\delta\mathbf{A}\| = \kappa(\mathbf{A})\frac{\|\delta\mathbf{A}\|}{\|\mathbf{A}\|}.$$

A matrix with a large condition number is said to be *ill-conditioned*. It is important to note the distinction between the *determinant* and the *condition number* of a matrix. A small determinant does not necessarily mean that a matrix is ill-conditioned, and a moderate determinant does not mean that a matrix well conditioned.

Some examples are presented below, and larger examples can be found in the online Jupyter notebooks, and in particular for the notoriously ill-conditioned *Hilbert matrix*.

3.2.2 Example: ill-conditioned system [slide 27]

Consider the problem

$$\begin{bmatrix} 1 & 2 \\ 2 & 4.0001 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}.$$

- When $\mathbf{b} = \begin{bmatrix} 2 & 4.0001 \end{bmatrix}^T$, the solution is $\mathbf{x} = \begin{bmatrix} 0 & 1 \end{bmatrix}^T$.
- When $\mathbf{b} = \begin{bmatrix} 2 & 4 \end{bmatrix}^T$, the solution is $\mathbf{x} = \begin{bmatrix} 2 & 0 \end{bmatrix}^T$.

We see here that a small change in the RHS leads to a very significant change in the solution.

3.2.3 Example: moderate determinant, large condition number [slide 28]

The $n \times n$ matrix

$$\mathbf{A} = \begin{bmatrix} 1 & -1 & \dots & -1 \\ 0 & 1 & \dots & -1 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{bmatrix}$$

has $\det(\mathbf{A}) = 1$ and $\kappa_\infty(\mathbf{A}) = n2^{n-1}$, hence the condition number becomes very large as n increases, despite the determinant remaining constant at one.

3.2.4 Example: small determinant, small condition number [slide 29]

The very simple $n \times n$diagonal matrix

$$\mathbf{D} = \mathrm{diag}(0.1, 0.1, \dots, 0.1)$$

has $\det(\mathbf{D}) = 10^{-n}$ and $\kappa_p(\mathbf{D}) = 1$.

Conditioning and determinant
Conditioning should not be confused with the determinant

4 Least-squares methods

4.1 Interpolation

Interpolation is fitting a function to a data set that passes through the data points.

4.1.1 Polynomial interpolation [slide 30]

If we have n data points in a two-dimensional space (x, y), we can usually fit a polynomial with n coefficients. Say we have 6 data points $\mathbf{f} = \left[f_1(x_1, y_1), \dots, f_6(x_6, y_6)\right]^T$. We can (hopefully) interpolate the data points with a polynomial of the form:

$$f(x, y) = c_0 + c_1 x + c_2 y + c_3 xy + c_4 x^2 + c_5 y^2.$$

We have six equations (one for each data point)

$$f_i = c_0 + c_1 x_i + c_2 y_i + c_3 x_i y_i + c_4 x_i^2 + c_5 y_i^2,$$

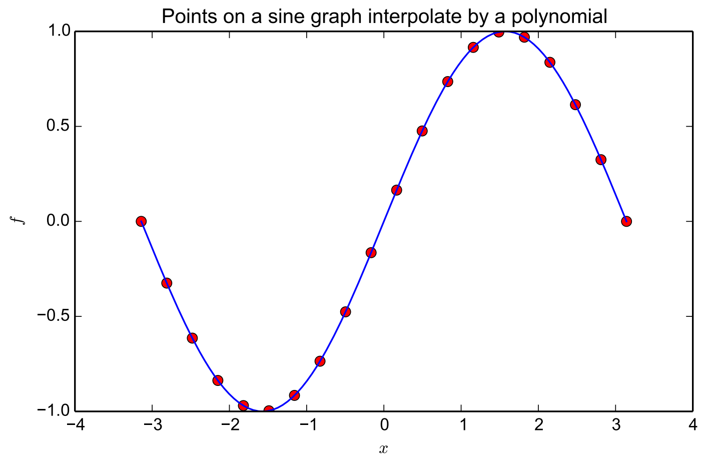
and we can solve

$$\underbrace{\begin{bmatrix} 1 & x_1 & y_1 & x_1 y_1 & x_1^2 & y_1^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_6 & y_6 & x_6 y_6 & x_6^2 & y_6^2 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} c_0 \\ \vdots \\ c_6 \end{bmatrix} = \begin{bmatrix} f_1 \\ \vdots \\ f_6 \end{bmatrix}$$

- As long as **A** can be solved, i.e. the points do not all lie on a line, we can find the interpolating polynomial.
- The matrix **A** is known as the *Vandermonde* matrix. It is a notoriously ill-conditioned matrix, and the condition number deteriorates with increasing polynomial degree.

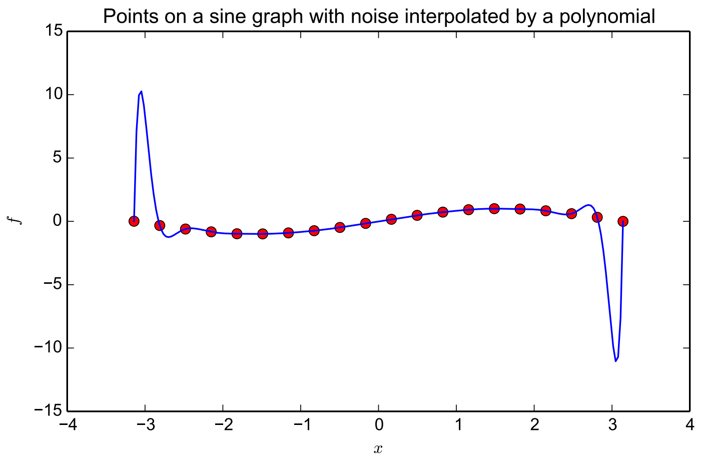
4.1.2 Interpolating the sine function [slide 31]

Polynomial interpolation can yield some unexpected results. We consider interpolating $\sin(x)$ at 20 equally spaced points on $x \in [-\pi, \pi]$. We can fit a polynomial of degree 19 to the points:



The fit appears to be very good.

We now try adding a small amount of white noise, $\max|\epsilon| = 10^{-2}$, at each point – we cannot perceive this visually in the points, but it as a very large effect on the interpolating polynomial:



We see large oscillations at the boundaries. This is common in polynomial interpolation, particularly with equally spaced points.

A message is that when we have n data points, it is not always wise to fit the highest possible degree polynomial. We will see next a way to fit lower degree polynomials

4.2 Data fitting

4.2.1 Over-determined systems [slide 32]

To find a solution to the problem $\mathbf{A}\mathbf{x} = \mathbf{b}$, the vector $\mathbf{b}$ must lie in the column space of $\mathbf{A}$ (look back over your IB notes if you need to review this concept). If $\mathbf{A}$ is an $m \times n$ matrix and $m > n$ (a *skinny* matrix), we have more equations than unknowns and in general there will be no solution.

For some vector $\hat{\mathbf{x}}$, we can define a 'residual' vector $\mathbf{r}$:

$$\mathbf{r} = \mathbf{A}\hat{\mathbf{x}} - \mathbf{b}$$

To find an approximate solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$, we can search for a vector $\hat{\mathbf{x}}$ that minimises the residual in some *norm*:

$$\min_{\hat{\mathbf{x}} \in \mathbb{C}^n} \left\| \mathbf{r}(\hat{\mathbf{x}}) \right\| = \min_{\hat{\mathbf{x}} \in \mathbb{C}^n} \left\| \mathbf{A}\hat{\mathbf{x}} - \mathbf{b} \right\| \tag{5}$$

This says: find $\hat{\mathbf{x}}$ that minimises the residual $\mathbf{r}$ in the chosen norm.

The residual is a vector, so we need a way to compare different residual vectors to decide if one is smaller than another. For this we need to pick a norm in which to minimise the residual. Different norms will give different results, and it turns out that one norm will be much easier to work with than others.

4.2.2 Minimising the error in the l_2 norm [slide 33]

If we use the l_2-norm for the problem in eq. (5), we seek:

$$\min_{\hat{\mathbf{x}} \in \mathbb{C}^n} \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2 \tag{6}$$

Squaring $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2$ and expanding:

$$\begin{aligned} r(\hat{\mathbf{x}}) = \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2^2 &= (\mathbf{A}\hat{\mathbf{x}} - \mathbf{b})^H (\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}) \\ &= \hat{\mathbf{x}}^H\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} - \hat{\mathbf{x}}^H\mathbf{A}^H\mathbf{b} - \mathbf{b}^H\mathbf{A}\hat{\mathbf{x}} + \mathbf{b}^H\mathbf{b}. \end{aligned} \tag{7}$$

Expanding with indices, we can see that

$$r = \sum_{i=1}^{m} \left| \sum_{j=1}^{n} \left(a_{ij}\hat{x}_j\right) - b_i \right|^2 ,$$

which is the least-squares error. Hence, minimisation in the l_2-norm is known as the *least-squares method*.

Equation (7) is clearly quadratic in $\hat{\mathbf{x}}$. To find $\hat{\mathbf{x}}$ that minimises r, we can differentiate r and set the derivative equation to zero. But how? Consider the Gâteaux derivative, and set it to zero:

$$\begin{aligned} \left.\frac{\mathrm{d}\, r(\hat{\mathbf{x}} + \epsilon\mathbf{v})}{\mathrm{d}\epsilon}\right|_{\epsilon=0} &= \left.\frac{\mathrm{d}\left((\hat{\mathbf{x}} + \epsilon\mathbf{v})^H \mathbf{A}^H\mathbf{A}(\hat{\mathbf{x}} + \epsilon\mathbf{v}) - (\hat{\mathbf{x}} + \epsilon\mathbf{v})^H \mathbf{A}^H\mathbf{b} - \mathbf{b}^H\mathbf{A}(\hat{\mathbf{x}} + \epsilon\mathbf{v}) + \mathbf{b}^H\mathbf{b}\right)}{\mathrm{d}\epsilon}\right|_{\epsilon=0} \\ &= \hat{\mathbf{x}}^H\mathbf{A}^H\mathbf{A}\mathbf{v} + \mathbf{v}^H\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} - \mathbf{v}^H\mathbf{A}^H\mathbf{b} - \mathbf{b}^H\mathbf{A}\hat{\mathbf{v}} \\ &= \mathbf{v}^H\left(\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} - \mathbf{A}^H\mathbf{b}\right) + \left(\mathbf{v}^H\left(\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} - \mathbf{A}^H\mathbf{b}\right)\right)^H \\ &= 0. \end{aligned}$$

We want the above to hold for all $\mathbf{v}$. This is the case if

$$\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^H\mathbf{b},$$

which is the classic least-squares problem. Note that a matrix of the form $\mathbf{A}^H\mathbf{A}$ is known as a *normal matrix*.

Re-arranging the least-squares problem:

$$
\begin{aligned}
\hat{\mathbf{x}} &= \left(\mathbf{A}^H\mathbf{A}\right)^{-1}\mathbf{A}^H\mathbf{b} \\
&= \mathbf{A}^{+}\mathbf{b}
\end{aligned}
$$

Pseudoinverse

The matrix $\mathbf{A}^{+} = \left(\mathbf{A}^H\mathbf{A}\right)^{-1}\mathbf{A}^H$ is known as the *pseudoinverse* or the *Moore–Penrose inverse*. The inverse $\left(\mathbf{A}^H\mathbf{A}\right)^{-1}$ can be computed when $\mathbf{A}$ is full rank, i.e. the columns of $\mathbf{A}$ are linearly independent (see Examples Paper). In this section, $\mathbf{A}$ has been a 'skinny matrix', i.e. $m > n$. If $\mathbf{A}$ is 'fat matrix' $(m < n)$ and full rank, then $\mathbf{A}^{+} = \mathbf{A}^H\left(\mathbf{A}\mathbf{A}^H\right)^{-1}$.

There are cases where $\mathbf{A}$ is not full rank, in which case the problem is under-determined and there are multiple solutions to the least-squares problem. We will look at strategies to handle this case once we have covered the singular value decomposition.

4.2.3 Example: fitting a linear polynomial to a dataset in two dimensions [slide 34]

At the start of this section, we considered polynomial interpolation of six data points in a two-dimensional space. We now consider a least-squares fit with the linear polynomial

$$
f(x, y) = c_0 + c_1 x + c_2 y.
$$

We have six equations of the form

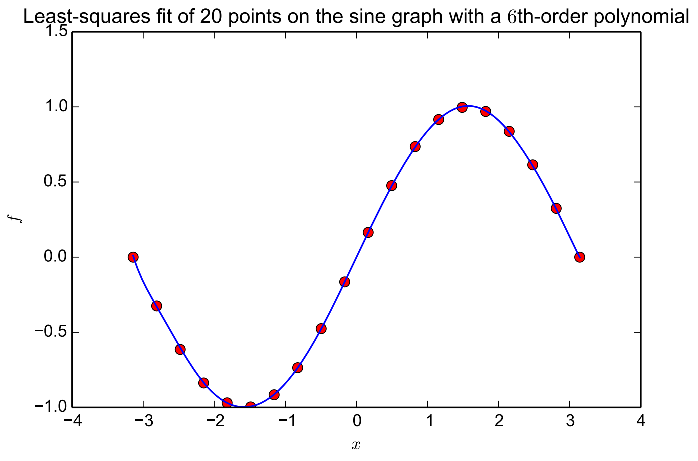
$$
f_i(x_i, y_i) = c_0 + c_1 x_i + c_2 y_i
$$

and we can solve

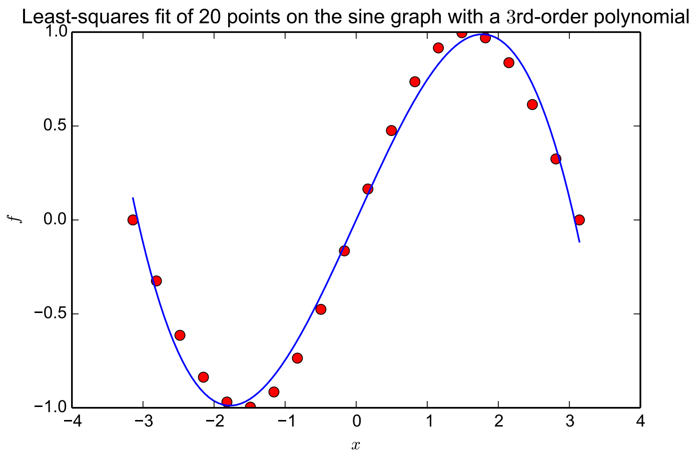
$$
\underbrace{\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \end{bmatrix}}_{\mathbf{A}^T}
\underbrace{\begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \\ 1 & x_4 & y_4 \\ 1 & x_5 & y_6 \\ 1 & x_5 & y_6 \end{bmatrix}}_{\mathbf{A}}
\begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}
=
\begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 \\ x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \\ y_1 & y_2 & y_3 & y_4 & y_5 & y_6 \end{bmatrix}
\begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \\ f_5 \\ f_6 \end{bmatrix}
$$

4.2.4 Least-squares fit of points on the sine graph [slide 35]

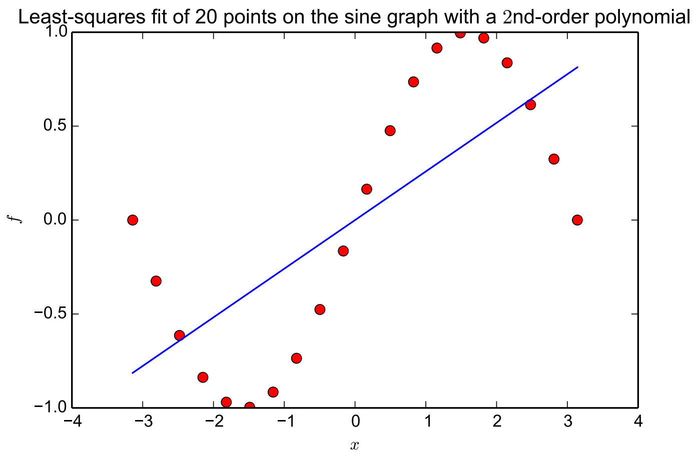
We consider the problem we looked at earlier for polynomial interpolation. If we fit a 6th-order polynomial:



The fit to the data looks very good. Fitting a 3rd-order polynomial:



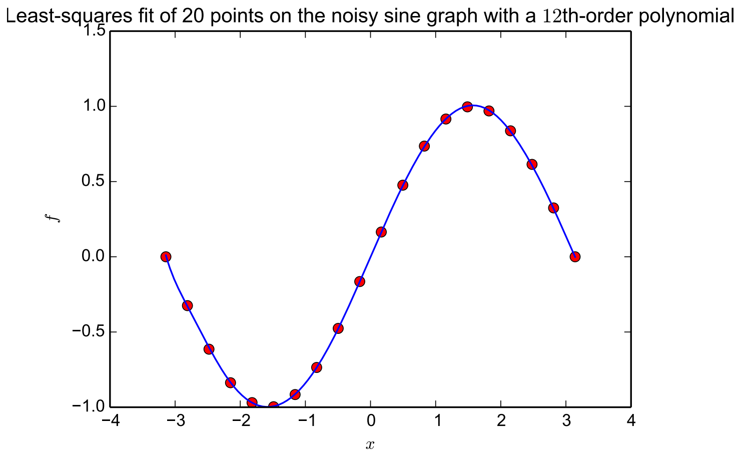
we can see some difference between the fit (solid line) and the data points. Fitting a quadratic function:



The fit is clearly very poor – it looks nothing like a sine function.

4.2.5 Least-squares fit of points on a noisy sine graph [slide 36]

Recall that when a small amount of noise was introduced to the sine data, the interpolating function had large spikes at edges of the domain. We now fit the same noisy data with a 12th-order polynomial in a least-squares sense:



The fit looks very good, and no oscillations are observed. As the degree of the fitting function is increased, oscillations begin to appear.

Experiment
Use the Jupyter notebook to test what happens as you increase the polynomial degree for this problem.

4.2.6 Solving least squares problems in practice [slide 37]

To solve the least squares problem

$$\mathbf{A}^H\mathbf{A}\hat{\mathbf{x}} = \mathbf{A}^H\mathbf{b},$$

if $\mathbf{A}$ is full rank we could just apply LU factorisation to $\mathbf{A}^H\mathbf{A}$ (or more specially, Cholesky factorisation since $\mathbf{A}^H\mathbf{A}$ is Hermitian). The problem is that $\mathbf{A}^H\mathbf{A}$ is notoriously ill-conditioned, and the conditioning deteriorates as the matrix becomes larger.

- For the Vandermonde matrix used for the sine interpolation problem with degree 12, $\kappa_2(\mathbf{A}) = 9.3 \times 10^{11}$.
- To fit a polynomial of degree 15 to the sine problem, $\kappa_2(\mathbf{A}) = 1.3 \times 10^{16}$!

Least-squares methods are usually solved using specialised methods that can cope with the ill-conditioning of $\mathbf{A}^H\mathbf{A}$. QR factorisation (recall from Part IB) is better than LU, but can also suffer from accuracy/stability problems.

Note
In practice, do not write your own least-squares solver. Use a specialised function or library.

4.2.7 Minimising in other norms

Least-squares/2-norm minimisation is popular because the residual is quadratic, so when we take the derivative the minimisation problem becomes linear and is straightforward to solve. Also, being quadratic it will have a unique minimum (if $\mathbf{A}$ is full rank).

We could pick another norm in which to minimise the error. The use of the 1-norm has become popular in recent years for some applications (e.g., compressed sensing). It is known as *least absolute deviations*. An issue with least-squares fitting is that it is sensitive to data outliers; the square of the 'error' amplifies the effect on the fitted function. Minimisation in the 1-norm is less sensitive to outliers.

The difficulty with norms other than the 2-norm is that the minimisation problem is non-linear, and therefore more challenging to solve.

5 Iterative methods for linear systems

5.1 Direct and iterative methods

We often want to solve a system of the form $\mathbf{Ax} = \mathbf{b}$. Solution methods can be categorised as *direct* or *iterative*.

5.1.1 Direct methods [slide 38]

A direct method computes a solution to $\mathbf{Ax} = \mathbf{b}$ in a known/predictable number of operations. In the absence of round-off errors, the solution is exact. Solving a system via LU decomposition ($\mathbf{Ax} = \mathbf{LUx} = \mathbf{b}$) is an example of a direct method.

- For an dense $n \times n$ matrix, LU decomposition has complexity $O(n^3)$, which makes it very expensive for large matrices.
- For a sparse $n \times n$, the LU complexity is typically in the range $O(n^{3/2})$ and $O(n^2)$. This is clearly better better than $O(n^3)$, but still expensive for large n.
- A more specialised point, direct methods cannot generally handle singular equations, i.e. matrices with a non-trivial nullspace.
- For reasonably conditioned systems, direct solvers can be robust, i.e. will reliably compute a solution.
- Direct solvers do not scale well on parallel computers since the substitution steps are inherently serial.

5.1.2 Iterative methods [slide 39]

An iterative method seeks an approximate solution via a series of steps (iterations), and is usually terminated when the error/residual falls below a prescribed threshold (in the chosen norm!).

There are many iterative methods for solving problems in linear algebra. In general, these methods can sometimes be very fast, but can be slow or fail abjectly.

For useful iterative solvers:

- In some cases $O(n)$ schemes are possible (this is known as an optimal solver), and iterative solvers can be orders of magnitude faster than direct solvers.
- Use less memory than direct solvers.
- Some methods can solve singular problems.
- Iterations can be terminated early if high accuracy is not required.
- Iterative are often (but not always) suited to large, parallel computers.

5.2 An iterative method for the largest eigenvalue and eigenvector

5.2.1 Power iteration [slide 40]

A classic iterative method for finding the eigenvector associated with the largest absolute eigenvalue of a matrix is known as *power iteration*. Recall that a vector $\mathbf{x} \in \mathbb{C}^n$ can be expressed in terms of the n eigenvectors of an $n \times n$ matrix:

$$\mathbf{x} = \sum_{i=1}^{n} \alpha_i \mathbf{u}_i \tag{8}$$

where $\alpha_i \in \mathbb{R}$. If we multiple $\mathbf{x}$ repeatedly by $\mathbf{A}$:

$$\underbrace{\mathbf{A}\mathbf{A}\ldots\mathbf{A}}_{k\text{times}}\mathbf{x} = \mathbf{A}^k\mathbf{x} = \sum_{i=1}^{n} \alpha_i \lambda_i^k \mathbf{u}_i,$$

---

we see that if the largest eigenvalue is distinct the resulting vector will be aligned with the eigenvector of the largest eigenvector (if $\mathbf{x}$ has a component in the direction of the eigenvector.

The power iteration method is demonstrated in the Jupyter notebooks, where it is seen shown that convergence can be slow.

If we have an approximation of the eigenvector associated with λ_{max}, a question is how can we find an approximation of λ_{max}? In Part IA, you used the scaling from $\mathbf{A}^k\mathbf{x}$ to $\mathbf{A}^{k+1}\mathbf{x}$. However this can be very unreliable. There is a better way.

5.2.2 Rayleigh quotient [slide 41]

Say we have an estimate of an eigenvector $\mathbf{x}$ of the matrix $\mathbf{A}$, in which case:

$$\mathbf{A}\mathbf{x} \approx \lambda \mathbf{x}.$$

To find an estimate of the corresponding eigenvalue $\lambda^{\star}$, we could pose a minimisation problem in the 2-norm:

$$\min_{\lambda^{\star} \in \mathbb{C}} \|\mathbf{A}\mathbf{x} - \lambda^{\star}\mathbf{x}\|_2 .$$

This minimised when

$$\lambda^{\star} = R(\mathbf{A}, \mathbf{x}) = \frac{\mathbf{x}^H \mathbf{A}\mathbf{x}}{\mathbf{x}^H \mathbf{x}},$$

where R is known as the *Rayleigh quotient*. To prove this, for any $\mu \in \mathbb{C}$ we have

$$\begin{aligned}
\|\mathbf{Ax} - \mu\mathbf{x}\|_2^2 &= \mathbf{x}^H\mathbf{A}^H\mathbf{Ax} - \bar{\mu}\mathbf{x}^H\mathbf{A}^H\mathbf{x} - \mu\mathbf{x}^H\mathbf{Ax} + |\mu|^2\mathbf{x}^H\mathbf{x} \\
&= \|\mathbf{Ax}\|_2^2 + \|\mathbf{x}\|_2^2\left(|\mu|^2 - \mu\overline{\lambda^\star} - \overline{\mu}\lambda^\star\right) \\
&= \|\mathbf{Ax}\|_2^2 + \|\mathbf{x}\|_2^2\left(\underbrace{(\mu - \lambda^\star)\left(\overline{\mu} - \overline{\lambda^\star}\right)}_{=|\mu-\lambda^\star|^2 \geq 0} - |\lambda^\star|^2\right) \\
&\geq \|\mathbf{Ax}\|_2^2 - |\lambda^\star|^2\|\mathbf{x}\|_2^2 \\
&= \|\mathbf{Ax} - \lambda^\star\mathbf{x}\|_2^2
\end{aligned}$$

The Rayleigh quotient is often defined as being for Hermitian matrices only, in which case it must be real-valued. For Hermitian matrices is has a number of special properties, including second-order accuracy for estimating eigenvalues. That is, if the error in the eigenvector is $O(\epsilon)$, then the error in the eigenvalue estimated via the Rayleigh quotient is $O(\epsilon^2)$.

5.3 Stationary methods for $Ax = b$

5.3.1 Family of stationary methods [slide 42]

We start with a family of simple methods for finding approximate solutions to $\mathbf{Ax} = \mathbf{b}$. We decompose the matrix operator such that $\mathbf{A} = \mathbf{N} - \mathbf{P}$. Rather than solving the exact problem

$$\mathbf{Nx} = \mathbf{b} + \mathbf{Px},$$

we compute an approximate solution $\mathbf{x}_{k+1}$:

$$\mathbf{Nx}_{k+1} = \mathbf{b} + \mathbf{Px}_k, \tag{9}$$

where $\mathbf{x}_k$ is the most recent known estimate of the solution, and the above equation is solved to compute the for the new estimate $\mathbf{x}_{k+1}$. The process is then repeated to hopefully converge to the exact solution.

The key is to split $\mathbf{A}$ such that equations of the form $\mathbf{Nx} = \mathbf{f}$ are easy (inexpensive) to solve. Classic examples are:

- Richardson iteration: $\mathbf{N} = \mathbf{I}$.

- Jacobi method: $\mathbf{N} = \mathrm{diag}(\mathbf{A})$.
- Gauss–Seidel: $\mathbf{N} = L(\mathbf{A})$ is the lower triangular part of $\mathbf{A}$ (including the diagonal).

To understand whether or not we can expect these iterative methods to work, we need to examine what happens to the error as we iterate.

5.3.2 Convergence [slide 43]

Defining the error at the kth iterate $\mathbf{e}_k = \mathbf{x}_{\text{exact}} - \mathbf{x}_k$, from eq. (9) we have:

$$\mathbf{N}\mathbf{e}_{k+1} = \mathbf{P}\mathbf{e}_k,$$

and therefore

$$\mathbf{e}_{k+1} = \mathbf{N}^{-1}\mathbf{P}\mathbf{e}_k.$$

To converge, we need the error to approach zero as the number of iterations k increases.

We express the error vector $\mathbf{e}_0$ as a linear combination of the eigenvectors of $\mathbf{N}^{-1}\mathbf{P}$:

$$\mathbf{e}_0 = c_1\mathbf{u}_1 + \ldots + c_n\mathbf{u}_n$$

and the error $\mathbf{e}_k$ as

$$\mathbf{e}_k = \left(\mathbf{N}^{-1}\mathbf{P}\right)^k \mathbf{e}_0 = c_1\lambda_1^k\mathbf{u}_1 + \ldots + c_n\lambda_n^k\mathbf{u}_n.$$

The method will converge only if all eigenvalues are less than one.

The largest absolute eigenvalue of a matrix $\mathbf{A}$ is often denoted by $\rho(\mathbf{A})$ and is known as the *spectral radius*.

The stationary methods based on splitting will converge if

$$\rho(\mathbf{N}^{-1}\mathbf{P}) < 1.$$

The smaller $\rho < 1$, the faster the convergence. Examples of the Jacobi and Gauss–Seidel methods for a 50×50 matrix are presented in the Jupyter notebooks.

5.4 Conjugate gradient method

The conjugate gradient (CG) method is remarkably simple but very powerful method. It is a *Krylov subspace method*, which is a more powerful family of the methods than the stationary methods in the previous section. The CG method is for Hermitian positive definite matrices only. There are other Krylov methods for other matrix types.

The CG method is sometimes presented as a direct method, but is applied in practice usually as an iterative method. The reasons will be clear when we consider some of its properties.

5.4.1 Conjugate gradient as a direct method [slide 44]

We wish to solve $\mathbf{A}\mathbf{x} = \mathbf{b}$, where $\mathbf{A}$ is a $n \times n$ Hermitian positive-definite matrix. Consider that we have a set P of n non-zero vectors that are A-conjugate:

$$P = \{\mathbf{p}_0, \mathbf{p}_1, \dots, \mathbf{p}_{n-1}\}.$$

A-conjugate implies that

$$\mathbf{p}_i^H \mathbf{A} \mathbf{p}_j = 0 \quad \text{if } i \neq j.$$

Since $\mathbf{A}$ is positive definite:

$$\mathbf{p}_i^H \mathbf{A} \mathbf{p}_i > 0.$$

Using P as a basis for the solution:

$$\mathbf{x} = \sum_{i=0}^{n-1} \alpha_i \mathbf{p}_i,$$

and

$$\mathbf{A}\mathbf{x} = \sum_{i=0}^{n-1} \alpha_i \mathbf{A}\mathbf{p}_i = \mathbf{b}.$$

Multiplying by $\mathbf{p}_j^H$:

$$\mathbf{p}_j^H \mathbf{A}\mathbf{x} = \sum_{i=0}^{n-1} \alpha_i \mathbf{p}_j^H \mathbf{A}\mathbf{p}_i = \alpha_j \mathbf{p}_j^H \mathbf{A}\mathbf{p}_j = \mathbf{p}_j^H \mathbf{b}$$

Since the vectors in P are A-conjugate:

$$\alpha_j = \frac{\mathbf{p}_j^H \mathbf{b}}{\mathbf{p}_j^H \mathbf{A} \mathbf{p}_j}$$

A question is how to generate the A-conjugate set P. A simple approach is to pick n linearly-independent vectors and apply the Gram–Schmidt process to build P. However, this approach is not practical for large problems. Building and storing P is too expensive.

5.4.2 Conjugate gradient as an iterative method [slide 45]

There is a remarkable algorithm for the CG method in which elements of P are computed with a very short recurrence relation. Moreover, in many cases we do not need all of P; sometimes we can get a very accurate solution with few elements of P.

Below is the CG algorithm. We will not prove it and you are not expect to memorise it, but it shows just how simple it is.

Algorithm 1 Conjugate gradient method

```
1: $\mathbf{x}_0 = \mathbf{0}$, $\mathbf{r}_0 = \mathbf{b}$, $\mathbf{p}_0 = \mathbf{r}_0$
2: for $k = 0, 1, \ldots$ do
3:     $\alpha_k = \mathbf{r}_k^H \mathbf{r}_k / \mathbf{r}_k^H \mathbf{A} \mathbf{r}_k$
4:     $\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$
5:     $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{A} \mathbf{p}_k$
6:     $\beta_k = \mathbf{r}_{k+1}^H \mathbf{r}_{k+1} / \mathbf{r}_k^H \mathbf{r}_k$
7:     $\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k \mathbf{p}_k$
8: end for
```

- The only 'serious' operations you need to perform are matrix-vector products and dot products. For very large sparse problems, these two operations are amongst the easiest when using a parallel computer.
- Note also that only a few work vectors need to be stored. Contrast this with LU where the factorisations must be stored.
- We don't have time to derive the method, but we do want to know some of its key properties.

5.4.3 Convergence [slide 46]

There is a lot of very rich analysis for the conjugate gradient method. We present here some key results.

Solving $\mathbf{A}\mathbf{x} = \mathbf{b}$ where $\mathbf{A}$ is an $n \times n$ matrix:

- For the error $\mathbf{e}_k = \mathbf{x}_{\text{exact}} - \mathbf{x}_k$, the CG method is monotone in the $\mathbf{A}$-norm $\|\mathbf{y}\|_A^2 = \mathbf{y}^H\mathbf{A}\mathbf{y}$:

  $$\|\mathbf{e}_{k+1}\|_A \leq \|\mathbf{e}_k\|_A$$

  and $\|\mathbf{e}_k\|_A = 0$ for some $k \leq n$.

- From the preceding point, the CG method will solve the problem exactly (in the absence of round-off error) in at most n iterations. This is why it is sometimes considered a direct method.

- The error norm $\|\mathbf{e}_k\|_A$ is minimised in the Krylov space.

- The number of iterations required to solve exactly equal to the number of distinct eigenvalues of $\mathbf{A}$.

- The rate of convergence is affected by the condition number $\kappa_2(\mathbf{A})$:

  $$\frac{\|\mathbf{e}_k\|_A}{\|\mathbf{e}_0\|_A} \leq 2\left(\frac{\sqrt{\kappa_2}-1}{\sqrt{\kappa_2}+1}\right)^k$$

Example
The monotonic convergence is demonstrated by the example in the Jupyter notebooks.

5.4.4 Optimisation [slide 47]

The conjugate gradient method can be viewed as a clever optimisation method. The update to $\mathbf{x}_k$ in the algorithm is

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_{k+1}\mathbf{p}_k$$

This is typical in optimisation, where we move from the current state $\mathbf{x}_k$ to a new position a distance α_{k+1} in the direct of $\mathbf{p}_k$.

If the CG method is an optimisation method, what is it optimising? From the analysis it appears to be minimising $\|\mathbf{e}_k\|_A$, but an optimisation method needs to compute its 'objective function', and we cannot compute $\|\mathbf{e}_k\|_A$ without know the exact solution.

Consider

$$\psi(\mathbf{x}) = \frac{1}{2}\mathbf{x}^H\mathbf{A}\mathbf{x} - \mathbf{x}^H\mathbf{b} \tag{10}$$

It is clear that the minimiser to this expression is the solution to $\mathbf{Ax} = \mathbf{b}$. If we expand $\|\mathbf{e}_k\|_A$:

$$\begin{aligned}
\|\mathbf{e}_k\|_A^2 &= \mathbf{e}_k^H \mathbf{A} \mathbf{e}_k \\
&= \left(\mathbf{x}_{\text{exact}} - \mathbf{x}_k\right)^H \mathbf{A} \left(\mathbf{x}_{\text{exact}} - \mathbf{x}_k\right) \\
&= \mathbf{x}_k^H \mathbf{A} \mathbf{x}_k - 2\mathbf{x}_k^H \mathbf{A} \mathbf{x}_{\text{exact}} + \mathbf{x}_{\text{exact}}^H \mathbf{A} \mathbf{x}_{\text{exact}} \\
&= \mathbf{x}_k^H \mathbf{A} \mathbf{x}_k - 2\mathbf{x}_k^H \mathbf{b} + \mathbf{x}_{\text{exact}}^H \mathbf{b} \\
&= 2\psi(\mathbf{x}_k) + \text{constant}
\end{aligned}$$

This shows that the CG method is an iterative optimisation method for minimising eq. (10).

5.4.5 Preconditioning [slide 48]

If the condition number of a matrix large, the CG method may be too slow to converge for practical use. In this case, preconditioning can be attempted, where the transformed system:

$$\mathbf{P}^{-1}\mathbf{A}\mathbf{x} = \mathbf{P}^{-1}\mathbf{b}$$

This is known as ‘left preconditioning’.

- The idea is that $\mathbf{P}^{-1} \approx \mathbf{A}^{-1}$, the condition number of the $\mathbf{P}^{-1}\mathbf{A}$ will be better than $\mathbf{A}$, and the CG method will therefore converge faster.
- In practice it can be a balancing act – $\mathbf{P}^{-1}$ must be cheap to apply for efficiency, but it must be close enough to $\mathbf{A}^{-1}$ to be effective.
- When it all works, preconditioned CG solvers can be orders of magnitude faster the LU factorisations, and they work much better on large, parallel computers.

6 Singular value decomposition (SVD)

Singular value decomposition (SVD) is one of the most beautiful concepts in mathematics. The definition is wonderfully simple, but the insights and applications are rich.

6.1 Definition and properties

6.1.1 Matrix diagonalisation – review [slide 49]

The diagonalisation of real, symmetric matrices was covered in Part IA[2]. Extending this to Hermitian matrices, for a Hermitian matrix $\mathbf{M} \in \mathbb{C}^{n \times n}$:

$$\mathbf{Q}^H \mathbf{M} \mathbf{Q} = \mathbf{\Lambda}$$

where the columns of $\mathbf{Q}$ are the normalised (in l_2) eigenvectors of $\mathbf{M}$ and $\mathbf{\Lambda}$ is a diagonal matrix of the eigenvalues of $\mathbf{M}$ (which are real).

Since the eigenvectors of a Hermitian matrix are orthogonal, $\mathbf{Q}$ is a unitary matrix, i.e. $\mathbf{Q}^{-1} = \mathbf{Q}^H$, and

$$\mathbf{M} = \mathbf{Q} \mathbf{\Lambda} \mathbf{Q}^H. \tag{11}$$

Matrix diagonalisation can be very helpful because we see that in a particular basis ('co-ordinate system') the action of a matrix on a vector is easy to interpret. Moreover, some operations, such as matrix–matrix multiplication, become trivial.

Whilst matrix diagonalisation is a very powerful concept it has some major limitations:

- It is only valid for square matrices;
- Not all matrices can be diagonalised, e.g. defective matrices; and
- The matrix $\mathbf{Q}$ of normalised eigenevctors is guaranteed unitary only for Hermitian matrices.

The singular value decomposition overcomes these shortcomings. It effectively 'diagonalises' any matrix, even rectangular matrices.

Expanding the diagonalisation

[2]If this is unfamiliar, look back over your Part IA notes.

If we expand eq. (11), we get

$$\mathbf{M} = \sum_i \lambda_i \mathbf{u}_i \mathbf{u}_i^H \tag{12}$$

where $(\lambda_i, \mathbf{u}_i)$ is the ith eigenpair of $\mathbf{M}$. The term $\mathbf{u}_i \mathbf{u}_i^H$ is a rank-1 matrix.

Hermitian matrices
All Hermitian matrices can be diagonalised

Sign ambiguity
Recall that eigenvectors of a matrix are not unique; it is direction that is important. Even when normalised, there persists a sign ambiguity:

$$\mathbf{M} = \mathbf{Q\Lambda Q}^H = (-\mathbf{Q})\mathbf{\Lambda}(-\mathbf{Q}^H).$$

6.1.2 Definition [slide 50]

The singular value decomposition of an $m \times n$ matrix $\mathbf{A}$ is:

$$\mathbf{A} = \mathbf{U\Sigma V}^H \tag{13}$$

---

where

- $\mathbf{U} \in \mathbb{C}^{m\times m}$ is a unitary matrix;
- $\mathbf{\Sigma} \in \mathbb{R}^{m\times n}$ is a diagonal matrix, with diagonal entries σ_i (the 'singular values') sorted such that $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_p \geq 0$, where $p = \min(m, n)$; and
- $\mathbf{V} \in \mathbb{C}^{n\times n}$ is a unitary matrix.

If $\mathbf{A}$ was Hermitian, then we would have $\mathbf{U} = \mathbf{V} = \mathbf{Q}$, and $\mathbf{\Sigma} = \mathbf{\Lambda}$.

6.1.3 Question: what should U, Σ and V be? [slide 51]

1. To answer part of this question, premultiply both sides of eq. (13) by $\mathbf{A}^H$:

$$\begin{aligned}\mathbf{A}^H\mathbf{A} &= \left(\mathbf{U\Sigma V}^H\right)^H \mathbf{U\Sigma V}^H \\ &= \mathbf{V\Sigma}^H\mathbf{U}^H\mathbf{U\Sigma V}^H \\ &= \mathbf{V\Sigma}^H\mathbf{\Sigma V}^H.\end{aligned}$$

Noting that $\mathbf{A}^H\mathbf{A}$ is Hermitian and that $\mathbf{\Sigma}^H\mathbf{\Sigma}$ is diagonal with entries $\sigma_1^2, \sigma_2^2, \dots, \sigma_p^2$, comparing this to the diagonalisation of a Hermition matrix in eq. (11), we see that this is just diagonalisation of the square matrix $\mathbf{A}^H\mathbf{A}$. Therefore:

- The columns of $\mathbf{V}$ are the (normalised) eigenvectors of $\mathbf{A}^H\mathbf{A}$; and
- The diagonal entries of $\mathbf{\Sigma}^H\mathbf{\Sigma}$ are the eigenvalues of $\mathbf{A}^H\mathbf{A}$.

Now post-multiplying both sides of eq. (13) by $\mathbf{A}^H$:

$$\begin{aligned}\mathbf{A}\mathbf{A}^H &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\left(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\right)^H \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H\mathbf{V}\mathbf{\Sigma}^H\mathbf{U}^H \\ &= \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}^H\mathbf{U}^H\end{aligned}$$

We now see that:

- The columns of $\mathbf{U}$ are the (normalised) eigenvectors of $\mathbf{A}\mathbf{A}^H$; and
- The diagonal entries of $\mathbf{\Sigma}\mathbf{\Sigma}^H$ are the eigenvalues of $\mathbf{A}\mathbf{A}^H$, which are the same as the eigenvalues of $\mathbf{A}^H\mathbf{A}$.

What is missing is the signs of the eigenvectors. We consider this next.

2. Post-multiplying both sides of $\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H$ by $\mathbf{V}$,

$$\mathbf{A}\mathbf{V} = \mathbf{U}\mathbf{\Sigma}$$

which can also be expressed at

$$\mathbf{A}\mathbf{v}_i = \sigma_i\mathbf{u}_i$$

where $\mathbf{v}_i$ is the ith column of $\mathbf{V}$ and where $\mathbf{u}_i$ is the ith column of $\mathbf{U}$.

6.1.4 SVD definition Summary [slide 52]

- The columns of $\mathbf{U}$ are the (normalised) eigenvectors of $\mathbf{A}\mathbf{A}^H$;
- The diagonal entries of $\mathbf{\Sigma}$ are the square roots of the eigenvalues of $\mathbf{A}^H\mathbf{A}$ (or, equivalently $\mathbf{A}\mathbf{A}^H$); and

- The columns of $\mathbf{V}$ are the (normalised) eigenvectors of $\mathbf{A}^H\mathbf{A}$;
- Use $\mathbf{A}\mathbf{v}_i = \sigma_i \mathbf{u}_i$ to deduce the sign for $\mathbf{u}_i$ given $\mathbf{v}_i$ (or *vice versa*); and
- Every matrix has a SVD, which is unique up to the signs of the eigenvectors in $\mathbf{U}$ and $\mathbf{V}$.

The SVD of a Hermitian matrix is the usual eigen diagonalisation.

6.1.5 The reduced SVD [slide 53]

Visualising the shape of the SVD on an $m \times n$ matrix with $m > n$:

$$\underbrace{\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}}_{\mathbf{A}\ (m\times n)} = \underbrace{\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}}_{\mathbf{U}\ (m\times m)} \underbrace{\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \sigma_n \\ \\ \\ \end{bmatrix}}_{\mathbf{\Sigma}\ (m\times n)} \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{\mathbf{V}^H\ (n\times n)}$$

the terms below row n in $\mathbf{\Sigma}$ are always zero. This means that the last $m - n$ columns of $\mathbf{U}$ make no contribution. In practice, the 'reduced' SVD, in which redundant entries are removed, is typically used:

$$\underbrace{\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}}_{\mathbf{A}\ (m\times n)} = \underbrace{\begin{bmatrix} \\ \\ \\ \\ \\ \\ \\ \end{bmatrix}}_{\mathbf{U}\ (m\times n)} \underbrace{\begin{bmatrix} \sigma_1 & & & \\ & \ddots & & \\ & & \ddots & \\ & & & \sigma_n \end{bmatrix}}_{\mathbf{\Sigma}\ (n\times n)} \underbrace{\begin{bmatrix} \\ \\ \end{bmatrix}}_{\mathbf{V}^H\ (n\times n)}$$

6.1.6 Computing the SVD: example [slide 54]

How to compute an SVD robustly could be an entire lecture course on its own. We will now compute an example 'by hand' by computing eigenpairs of $\mathbf{A}\mathbf{A}^H$ and $\mathbf{A}^H\mathbf{A}$. As is often the case in linear algebra, the method by which we compute problems by hand is not the way real problems should be computed. As touched upon with least-squares methods, the normal matrix $\mathbf{A}\mathbf{A}^H$ is notoriously ill-conditioned. Use built-in library functions to compute an SVD.

Compute the SVD of the matrix

$$\mathbf{A} = \begin{bmatrix} -1 & 1 & 0 \\ 0 & -1 & 1 \end{bmatrix}.$$

1. Compute eigenvalues/vectors of $\mathbf{A}\mathbf{A}^T$:

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}$$

For this matrix, $\lambda_1 = 3$ and $\mathbf{u}_1 = (1/\sqrt{2})[-1\ 1]$, and $\lambda_2 = 1$ and $\mathbf{u}_2 = (1/\sqrt{2})[1\ 1]$. Therefore

$$\mathbf{U} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix}$$

and

$$\mathbf{\Sigma} = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}.$$

2. Compute eigenvalues/vectors of $\mathbf{A}^T\mathbf{A}$:

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}.$$

For this matrix, $\lambda_1 = 3$ and $\mathbf{u}_1 = (1/\sqrt{6}) \begin{bmatrix} 1 & -2 & 1 \end{bmatrix}^T$, $\lambda_2 = 1$ and $\mathbf{u}_2 = (1/\sqrt{2}) \begin{bmatrix} -1 & 0 & 1 \end{bmatrix}^T$, and $\lambda_3 = 0$ and $\mathbf{u}_3 = (1/\sqrt{3}) \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}^T$. Therefore,

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1/\sqrt{6} & -2/\sqrt{6} & 1/\sqrt{6} \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \\ 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}.$$

3. The last column of $\mathbf{\Sigma}$ is zero and therefore make no contribution. The *reduced SVD* is therefore:

$$\mathbf{A} = \frac{1}{\sqrt{2}} \begin{bmatrix} -1 & 1 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} 1/\sqrt{6} & -2/\sqrt{6} & 1/\sqrt{6} \\ -1/\sqrt{2} & 0 & 1/\sqrt{2} \end{bmatrix}.$$

6.1.7 Low rank approximations [slide 55]

If we expand the SVD in eq. (13), we get

$$\mathbf{A} = \sum_{i=1}^{r} \sigma_i \mathbf{u}_i \mathbf{v}_i^H$$

where r is the number of non-zero singular values, $\mathbf{u}_i$ is the ith column of $\mathbf{U}$ and $\mathbf{v}_i$ is the ith column of $\mathbf{V}$. The above is the expression of a matrix as a sum of rank-1 matrices.

A 'low rank' approximation of $\mathbf{A}$ is

$$\mathbf{A}_k = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^H$$

where $k < r$. The rank of $\mathbf{A}_k$ is k, which is less than the rank of $\mathbf{A}$ ($\mathbf{A}_k$ has fewer linearly independent rows/columns than $\mathbf{A}$).

How good is $\mathbf{A}_k$? It is possible to show that

$$\|\mathbf{A} - \mathbf{A}_k\|_F = \sqrt{\sigma_{k+1}^2 + \ldots + \sigma_r^2} \leq \|\mathbf{A} - \mathbf{B}\|_F$$

for all matrices $\mathbf{B}$ of rank k or less. This says that $\mathbf{A}_k$ is a better approximation of $\mathbf{A}$ than *any* other matrix of rank k, measured in the Frobenius norm.

Similarly in the 2-norm:

$$\|\mathbf{A} - \mathbf{A}_k\|_2 = \sigma_{k+1} \leq \|\mathbf{A} - \mathbf{B}\|_2$$

for all matrices $\mathbf{B}$ of rank k or less.

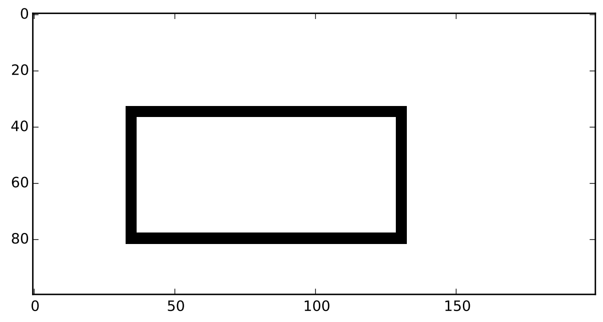
The message is that we can use the SVD to construct optimal low-rank approximations of a matrix $\mathbf{A}$. We'll see the use of low-rank approximation for some applications.

6.2 Applications of the SVD

There are many applications of the SVD. Here we present just a few. All the presented examples can be found in the Jupyter notebooks.

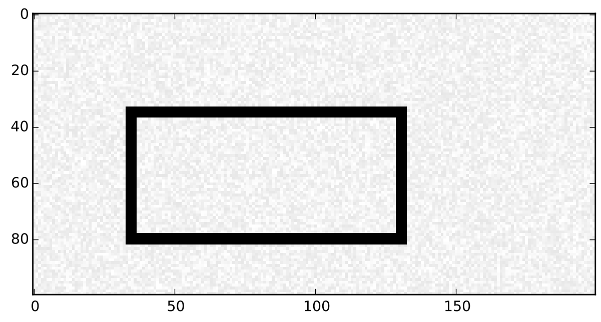
6.2.1 Finding patterns in data [slide 56]

We can use SVD to interpret large data sets. As an example, consider a 100×200 matrix with entries that are equal to 0 or 1. The entries have been arranged in a pattern which is shown graphically below:



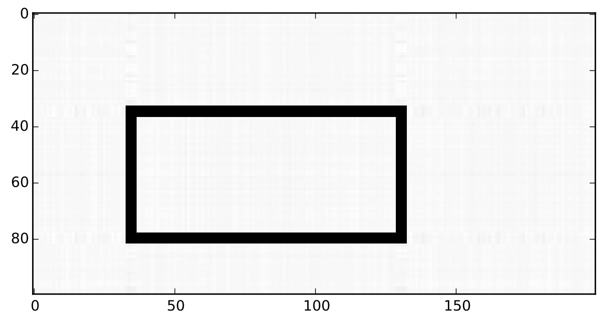
Performing a SVD of the matrix, we find $\{\sigma_i\} = \{1.35 \times 10^2, 1.96 \times 10^1, 1.32 \times 10^1, 0, \dots, 0\}$. This means that the matrix is rank 3, and can be represented as the sum of just three rank-1 matrices.

We now take the same problem, and add noise (between 0 and 0.1) to the 'white' background image:



If you look carefully you can see squares for each matrix entry (depending on print quality, you might want to look at this example on a screen).

If we perform an SVD of the matrix with noise, the largest singular values is roughly 1.29×10^2, and the smallest is roughly 1.2×10^{-1}. Reconstructing the image using only singular values that are larger than 1 (there are three of these):



Visually, much of the noise has been removed.

6.2.2 Image compression [slide 57]

Images can be represented as matrices, with one matrix entry for each pixel. The matrix entry is a number that represents an intensity. For RGB colour images, an image is represented by three such matrices - one for red, one for green and one for blue. In 24-bit colour, each colour at each pixel is represented with 8 bits, which means an integer between 0 and 255 for the colour intensity.

An 8 megapixel digital image has 2448×3264 pixels. Storing this as an RGB matrix would require roughly 24MB. To compress the image, we could try a low-rank approximation. We can construct an approximation by computing the SVD of the image and discarding the singular values that fall below a threshold.

Below is a photograph taken in the first lecture:

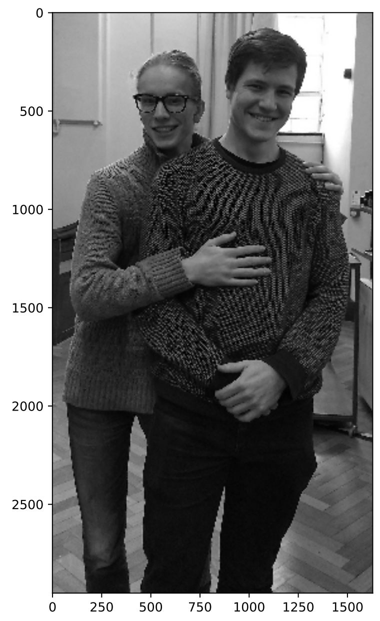


Figure 1: Original image.

We can represent the grey scale the image as a matrix with dimensions equal to the number of pixels and with float values between 0 and 1 corresponding the intensity (0=black, 1=white), or integers between 0 and 255 (0=black, 255=white).

Singular values Performing a SVD of the image, we plot the singular values:

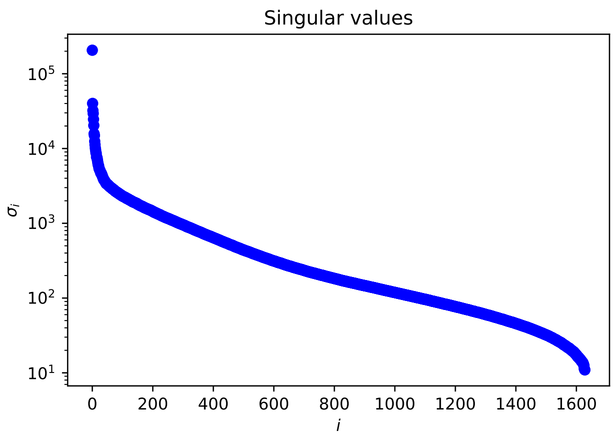


Figure 2: Singular values of the image.

The ratio between the largest and smallest singular values is several orders of magnitude.

We now create a low-rank approximation of the original image by retaining only the larger singular values. If we retain the largest 10% of the singular values:

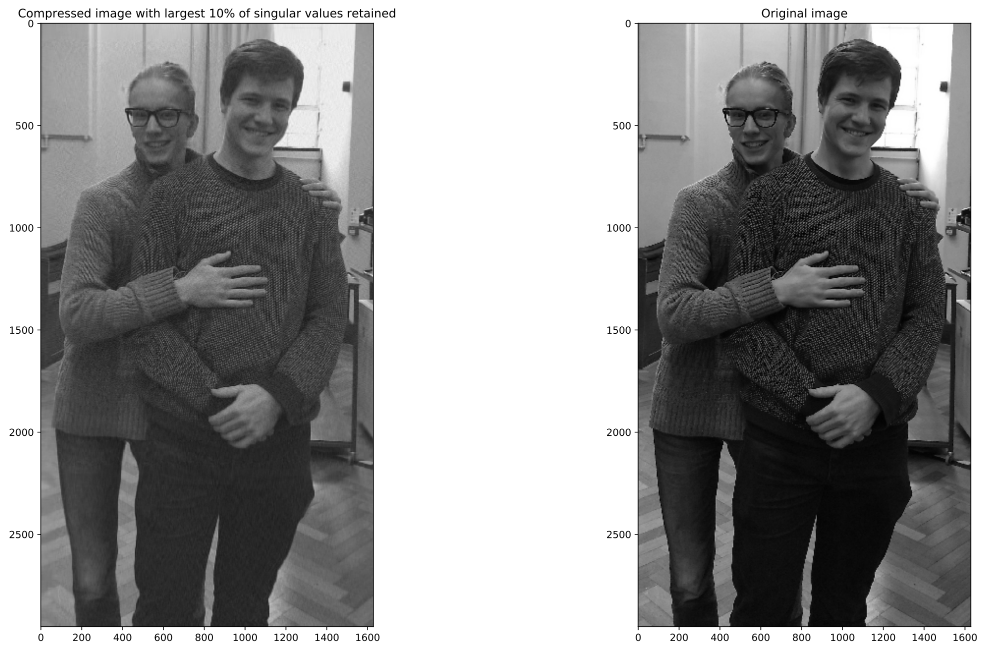


Figure 3: Compressed (left) with largest 10% of the singular values and original (right) images.

If we retain only the largest 2% of the singular values:

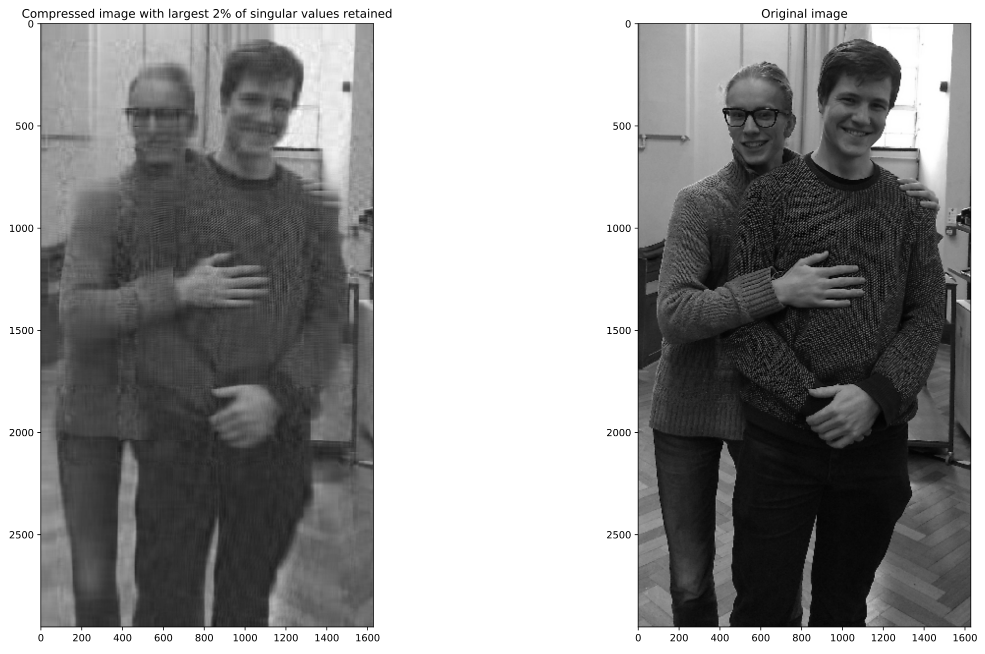


Figure 4: Compressed (left) with largest 2% of the singular values and original (right) images.

If we retain only the largest 0.5% of the singular values:

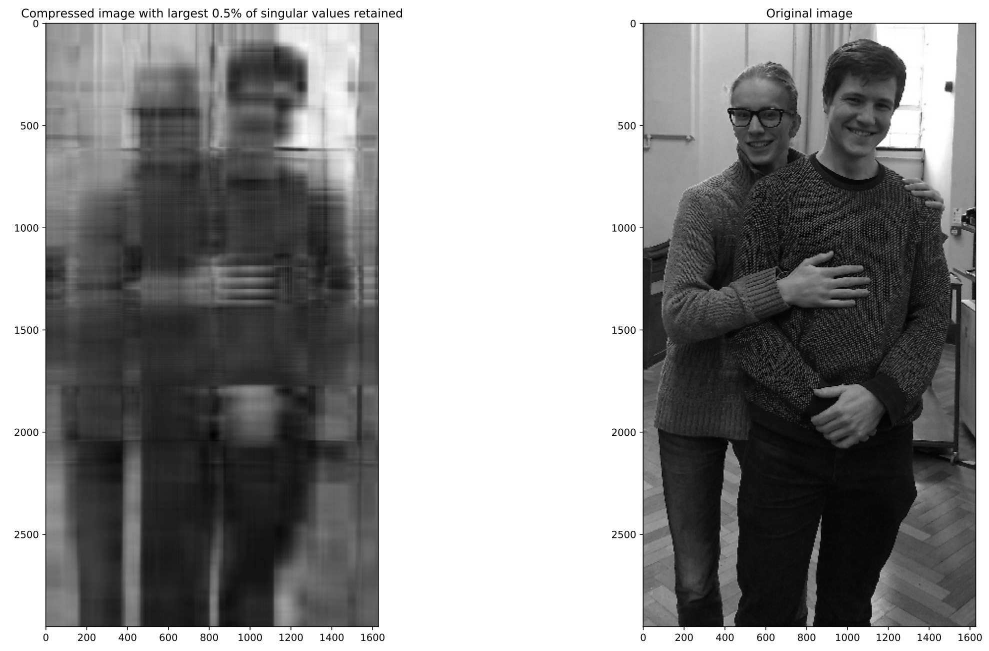


Figure 5: Compressed (left) with largest 0.5% of the singular values and original (right) images.

6.2.3 Effective rank [slide 58]

When taking measurements, noise is often unavoidable and this can make it hard to detect (near) linear dependencies. Consider the matrix

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

By inspection, this matrix is clearly rank 2. If we add noise to the matrix entries in the range $(0, 10^{-6})$:

$$\mathbf{A} = \begin{bmatrix} 1.00000221e+00 & 1.00000060e+00 & 1.00000142e+00 \\ 2.00000178e+00 & 2.00000152e+00 & 2.00000119e+00 \\ 1.00000082e+00 & 2.19239352e-06 & 1.00000121e+00 \end{bmatrix}$$

it becomes full rank (rank 3). In practice we would work with large data sets, so would have little chance of detecting any near linear dependencies by inspection.

Performing an SVD on the above matrix the singular values are: 4.048, 0.7811 and 3.90×10^{-7}. The effective rank is the number of singular values that are *greater* than the noise level, i.e. in this case the effective rank is 2.

6.2.4 Least-squares solutions [slide 59]

We'll now do some linear algebra gymnastics to use the SVD to better understand least squares problems, including the rank deficient case.

Recall that if the $m \times n$ matrix $\mathbf{A}$, with $m > n$, is full rank, then the least squares solution to $\mathbf{A}\mathbf{x} = \mathbf{b}$ is:

$$\hat{\mathbf{x}} = \left(\mathbf{A}^H\mathbf{A}\right)^{-1}\mathbf{A}^H\mathbf{b},$$

where $\mathbf{A}^+ = \left(\mathbf{A}^H\mathbf{A}\right)^{-1}\mathbf{A}^H$ is the psuedoinverse.

6.2.5 Full rank case [slide 60]

If $\mathbf{A}$ is an $m \times n$ matrix with $m > n$, we can partition the SVD as:

$$\mathbf{A} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_1 \\ \mathbf{0}\end{bmatrix}\mathbf{V}^H.$$

Since matrix is full rank, $\mathbf{\Sigma}_1$ will be $n \times n$, and $\mathbf{U}_1$ is $m \times n$. Since the columns of $\mathbf{U}$ are orthonormal, $\mathbf{U}_1^H\mathbf{U}_1 = \mathbf{I}_{n\times n}$ and $\mathbf{U}_2^H\mathbf{U}_1 = \mathbf{0}_{(m-n)\times n}$. Recall that $\|\mathbf{A}\mathbf{x}\|_2 = \|\mathbf{U}\mathbf{A}\mathbf{x}\|_2$ when $\mathbf{U}$ is unitary. We will use this property below.

A least squares solution minimises:

$$\begin{aligned} r = \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2^2 &= \left\|\begin{bmatrix}\mathbf{U}_1^H \\ \mathbf{U}_2^H\end{bmatrix}(\mathbf{A}\hat{\mathbf{x}} - \mathbf{b})\right\|_2^2 \\ &= \left\|\begin{bmatrix}\mathbf{U}_1^H \\ \mathbf{U}_2^H\end{bmatrix}\left(\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}^H\hat{\mathbf{x}} - \mathbf{b}\right)\right\|_2^2 \\ &= \left\|\begin{bmatrix}\mathbf{\Sigma}_1\mathbf{V}^H\hat{\mathbf{x}} - \mathbf{U}_1^H\mathbf{b} \\ -\mathbf{U}_2^H\mathbf{b}\end{bmatrix}\right\|_2^2 \\ &= \left\|\mathbf{\Sigma}_1\mathbf{V}^H\hat{\mathbf{x}} - \mathbf{U}_1^H\mathbf{b}\right\|_2^2 + \left\|\mathbf{U}_2^H\mathbf{b}\right\|_2^2. \end{aligned}$$

The residual is obviously minimised when $\mathbf{\Sigma}_1\mathbf{V}^H\mathbf{x} = \mathbf{U}_1^H\mathbf{b}$, i.e.

$$\hat{\mathbf{x}} = \mathbf{V}\mathbf{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b}.$$

Both $\mathbf{\Sigma}_1$ and $\mathbf{V}$ are full rank, therefore the least-squares solution is unique.

If we wish, we can add the 'zero padding' back to $\mathbf{U}$:

$$\hat{\mathbf{x}} = \mathbf{V}\mathbf{\Sigma}^+\mathbf{U}^H\mathbf{b}$$

where $\mathbf{\Sigma}^+$ is the pseudo inverse of $\mathbf{\Sigma}$:

$$\mathbf{\Sigma}^+ = \begin{bmatrix} 1/\sigma_1 & & \\ & \ddots & \\ & & 1/\sigma_n \\ & & \\ & & \end{bmatrix}$$

The SVD can be use to compute least-squares solutions. Compared to Cholesky factorisation of the normal matrix $\mathbf{A}^H\mathbf{A}$ or QR factorisation, it is the most stable.

6.2.6 Relationship to the normal equations [slide 61]

If $\mathbf{A}$ is full rank and we have a SVD of $\mathbf{A}$:

$$\begin{aligned}\mathbf{A}^+ = \left(\mathbf{A}^H\mathbf{A}\right)^{-1}\mathbf{A}^H &= \left(\mathbf{V}\boldsymbol{\Sigma}_1^H\mathbf{U}_1^H\mathbf{U}_1\boldsymbol{\Sigma}_1\mathbf{V}^H\right)^{-1}\mathbf{V}\boldsymbol{\Sigma}_1^H\mathbf{U}_1^H \\ &= \left(\mathbf{V}\boldsymbol{\Sigma}_1^H\boldsymbol{\Sigma}_1\mathbf{V}^H\right)^{-1}\mathbf{V}\boldsymbol{\Sigma}_1^H\mathbf{U}_1^H \\ &= \left(\mathbf{V}\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_1^{-H}\mathbf{V}^H\right)\mathbf{V}\boldsymbol{\Sigma}_1^H\mathbf{U}_1^H \\ &= \mathbf{V}\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\end{aligned}$$

This gives the least squares solution $\hat{\mathbf{x}} = \mathbf{V}\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b} = \mathbf{V}\boldsymbol{\Sigma}^+\mathbf{U}^H\mathbf{b}$.

6.2.7 Stability [slide 62]

If $\hat{\mathbf{x}} = \mathbf{A}^+\mathbf{b}$ ($\hat{\mathbf{x}} \in \mathbb{C}^n$) then:

$$\begin{aligned}\|\hat{\mathbf{x}}\|_2^2 &= \left(\mathbf{V}\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b}\right)^H\mathbf{V}\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b} \\ &= \mathbf{b}^H\mathbf{U}_1\boldsymbol{\Sigma}_1^{-H}\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b} \\ &= \left\|\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b}\right\|_2^2 \\ &\geq \left|\left(\boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b}\right)_n\right|^2 \quad \text{(absolute value of the the last entry in the } \boldsymbol{\Sigma}_1^{-1}\mathbf{U}_1^H\mathbf{b}\text{, squared)} \\ &= \left|\mathbf{u}_n^H\mathbf{b}\right|^2/\sigma_{\min}^2 \quad (\mathbf{u}_n \text{ is the last column of } \mathbf{U}_1).\end{aligned}$$

This says that if the smallest singular value $\sigma_{\min}$ is small, then the least squares solution will be large and very sensitive to changes in $\mathbf{b}$.

6.2.8 Rank deficient case [slide 63]

The preceding shows that if $\sigma_{\min} = 0$, we cannot compute a unique least squares solution. In the case that $\sigma_{\min}$ is small, while we can compute a solution, $\|\mathbf{x}\|_2$ can become very large with small changes in $\mathbf{b}$, which is not very satisfactory either.

For a problem with zero singular values, consider a partitioning of the SVD:

$$\mathbf{A} = \begin{bmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{bmatrix}\begin{bmatrix}\boldsymbol{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{0}\end{bmatrix}\begin{bmatrix}\mathbf{V}_1 & \mathbf{V}_2\end{bmatrix}^H$$

where $\boldsymbol{\Sigma}_1$ has size $r \times r$. It follows that

$$\mathbf{A} = \mathbf{U}_1\boldsymbol{\Sigma}_1\mathbf{V}_1^H$$

Examining the least-squares residual (recall that we can apply $[\mathbf{U}_1\ \mathbf{U}_2]^H$ to $\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}$ and it will not change the l_2 norm):

$$
\begin{aligned}
\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2^2 &= \left\| \begin{bmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{bmatrix} (\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}) \right\|_2^2 \\
&= \left\| \begin{bmatrix} \mathbf{U}_1^H \\ \mathbf{U}_2^H \end{bmatrix} \left(\mathbf{U}_1 \mathbf{\Sigma}_1 \mathbf{V}_1^H \hat{\mathbf{x}} - \mathbf{b} \right) \right\|_2^2 \\
&= \left\| \begin{bmatrix} \mathbf{\Sigma}_1 \mathbf{V}_1^H \hat{\mathbf{x}} - \mathbf{U}_1^H \mathbf{b} \\ -\mathbf{U}_2^H \mathbf{b} \end{bmatrix} \right\|_2^2 \\
&= \left\| \mathbf{\Sigma}_1 \mathbf{V}_1^H \hat{\mathbf{x}} - \mathbf{U}_1^H \mathbf{b} \right\|_2^2 + \left\| \mathbf{U}_2^H \mathbf{b} \right\|_2^2
\end{aligned}
\tag{14}
$$

It is clear that $\|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\|_2$ is minimised when $\mathbf{\Sigma}_1 \mathbf{V}_1^H \hat{\mathbf{x}} = \mathbf{U}_1^H \mathbf{b}$.

For the full rank case, we worked with $\mathbf{V}$, which has only the trivial nullspace. However, we now work with $\mathbf{V}_1^H$, which is only part of $\mathbf{V}^H$, and the matrix $\mathbf{V}_1^H$ consequently has a non-trivial nullspace. Specifically, we have $\mathbf{V}_1^H(\mathbf{V}_2\mathbf{z}) = \mathbf{0}$ for all $\mathbf{z} \in \mathbb{C}^{n-r}$ since the columns of $\mathbf{V}_1$ and $\mathbf{V}_2$ are orthogonal. The result is that there are an infinite number of solutions $\hat{\mathbf{x}}$ that satisfy $\mathbf{\Sigma}_1 \mathbf{V}_1^H \hat{\mathbf{x}} = \mathbf{U}_1^H \mathbf{b}$.

Since vectors $\mathbf{V}_2\mathbf{z}$ lie in the nullspace of $\mathbf{V}_1$, we have that

$$\hat{\mathbf{x}} = \mathbf{V}_1 \mathbf{\Sigma}_1^{-1} \mathbf{U}_1^H \mathbf{b} + \mathbf{V}_2 \mathbf{z}$$

for all $\mathbf{z} \in \mathbb{C}^{n-r}$ is a solution to the least squares problem. Taking the norm of the above expression and exploiting that $\mathbf{V}_1$ and $\mathbf{V}_2$ are orthogonal with respect to each other:

$$\|\hat{\mathbf{x}}\|_2^2 = \left\| \mathbf{V}_1 \mathbf{\Sigma}_1^{-1} \mathbf{U}_1^H \mathbf{b} \right\|_2^2 + \|\mathbf{V}_2 \mathbf{z}\|_2^2 .$$

Therefore

$$\hat{\mathbf{x}} = \mathbf{V}_1 \mathbf{\Sigma}_1^{-1} \mathbf{U}_1^H \mathbf{b}$$

is the minimiser to the least-squares problem with *minimal l_2 norm*. In other words, of the all the solutions to the least squares problem, $\hat{\mathbf{x}}$ is the smallest in the l_2 norm.

6.2.9 Fitting points and singular systems [slide 64]

Say we are given four data points that depend on x and y, and we are asked to fit a polynomial of the form

$$f(x, y) = c_{00} + c_{10}x + c_{01}y + c_{11}xy$$

to the data points. Normally, we would expect to be able to fit the above polynomial to four data points by interpolation, i.e. solving $\mathbf{Ac} = \mathbf{f}$ where $\mathbf{A}$ a square Vandermonde matrix. However, if the points happen to lie on a line, then $\mathbf{A}$ will be singular. If the points happen to lie almost on a line, then $\mathbf{A}$ will be almost singular.

A possibility is to exclude zero or small singular values from the process, thereby finding a least-squares fit with minimal $\|\mathbf{c}\|_2$. We test this for the data set $f_1(1, 0) = 3$, $f_2(2, 0) = 5$, $f_3(3, 0) = 7$, $f_4(4, 0) = 9$. The data lies on the line $y = 0$, and is in fact is linear in x.

To find the polynomial coefficients we want to solve

$$\underbrace{\begin{bmatrix} 1 & x_1 & y_1 & x_1y_1 \\ 1 & x_2 & y_2 & x_2y_2 \\ 1 & x_3 & y_3 & x_3y_3 \\ 1 & x_4 & y_4 & x_4y_4 \end{bmatrix}}_{\mathbf{A}} \begin{bmatrix} c_0 \\ c_{10} \\ c_{01} \\ c_{11} \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix}$$

For the points in the data set we have:

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

which is clearly singular. We could try a least-squares fit by solving $\mathbf{A}^T\mathbf{Ac} = \mathbf{A}^T\mathbf{f}$, but

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 4 & 10 & 0 & 0 \\ 10 & 30 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

which is singular.

If we perform an SVD of $\mathbf{A}$ and compute $\mathbf{c} = \mathbf{V\Sigma}^+\mathbf{U}^T\mathbf{f}$, we get

$$f(x, y) = 1 + 2x,$$

which in fact interpolates the points.

See the Jupyter notebook for the steps.

6.2.10 Example: fitting with nearly singular systems [slide 65]

If we take the previous example and add a small amount of noise $\epsilon \in (-5 \times 10^{-4}, -5 \times 10^{-4})$ to x_i, y_i and f_i, the Vandermonde matrix is no longer singular and can be used to compute an interpolating polynomial. For some noise, we find that

$$f(x, y) = 1.00365037 + 1.99853161x - 5.16321091y + 2.40001974xy$$

The coefficients of the y and xy terms are suddenly significant, and this is due only to the noise.

If we compute and SVD and ignore singular values less than 10^{-3}, we get:

$$f(x, y) = 0.999257206 + 2.00013498x + 6.49882602 \times 10^{-4}y + 1.13588861 \times 10^{-3}xy,$$

which is a more reasonable fit and very close to the noise-free case. See the Jupyter notebook for the steps.