Inverse Problems in Imaging
Lecture 2

Simon R. Arridge[1]

[1]Department of Computer Science, University College London, UK

UCL, Term 2

Outline

Outline

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.
- From a computational point of view, we are trying to fit a model to data, which is done by *optimisation*.

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.
- From a computational point of view, we are trying to fit a model to data, which is done by *optimisation*.
- Not all inverse problems are solved by optimisation (but many are).

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.
- From a computational point of view, we are trying to fit a model to data, which is done by *optimisation*.
- Not all inverse problems are solved by optimisation (but many are).
- Not all optimisation problems are about inverse problems. We will do some simple “functional” examples to introduce useful algorithms.

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.
- From a computational point of view, we are trying to fit a model to data, which is done by *optimisation*.
- Not all inverse problems are solved by optimisation (but many are).
- Not all optimisation problems are about inverse problems. We will do some simple "functional" examples to introduce useful algorithms.
- Not all inverse problems involve images. We will do some "toy" problems to get the concepts across.

- Inverse problems involves the inference of the parameters of a model from data. In this course we are considering (mainly) *Image Reconstruction* - i.e. the infered parameters form an image, but the data comes from many different types of experimental observations.
- From a computational point of view, we are trying to fit a model to data, which is done by *optimisation*.
- Not all inverse problems are solved by optimisation (but many are).
- Not all optimisation problems are about inverse problems. We will do some simple "functional" examples to introduce useful algorithms.
- Not all inverse problems involve images. We will do some "toy" problems to get the concepts across.
- Mostly, we concentrate on *Inverse Problems in Imaging* : i.e. model inversion whose solutions are images.

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$.

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$. We might also have time variation (sometimes called 4D).

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$. We might also have time variation (sometimes called 4D). Hyperspectral images add another dimension. 5D imaging !

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$. We might also have time variation (sometimes called 4D). Hyperspectral images add another dimension. 5D imaging !
- Images are *spatially organised* : the connectivity between pixels is due to physical proximity. We can make use of *structural coherence*, e.g. edges. In time-varying imaging we can exploit *temporal coherence*. In Hyperspectral imaging we can exploit *spectral coherence*.

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$. We might also have time variation (sometimes called 4D). Hyperspectral images add another dimension. 5D imaging !
- Images are *spatially organised* : the connectivity between pixels is due to physical proximity. We can make use of *structural coherence*, e.g. edges. In time-varying imaging we can exploit *temporal coherence*. In Hyperspectral imaging we can exploit *spectral coherence*.
- In other words, imaging comes with a lot of detailed *prior information* of a specific kind. This helps a lot !

What's special about images rather than generic inverse problems ?

- Images are very large : several million pixels in camera images.
- In tomography in the real world we deal with 3D data. Images might be $> 10^8$. We might also have time variation (sometimes called 4D). Hyperspectral images add another dimension. 5D imaging !
- Images are *spatially organised* : the connectivity between pixels is due to physical proximity. We can make use of *structural coherence*, e.g. edges. In time-varying imaging we can exploit *temporal coherence*. In Hyperspectral imaging we can exploit *spectral coherence*.
- In other words, imaging comes with a lot of detailed *prior information* of a specific kind. This helps a lot !
- Compare this to machine learning where the connectivity of a graph representation of data has to be defined in more abstract ways, e.g. using kernels.

Broadly speaking Optimisation requires us to consider four things

Broadly speaking Optimisation requires us to consider four things

- A *metric*. This addresses the question “Optimisation *of* what?”. A metric is a single real positive number that acts as a “figure of merit” of our success at optimisation.

Broadly speaking Optimisation requires us to consider four things

- A *metric*. This addresses the question “Optimisation *of* what?”. A metric is a single real positive number that acts as a “figure of merit” of our success at optimisation.
- A *model*. This addresses the question “what is the metric applied to?”. A model is a mathematical or computational process that predicts a result which maybe a function, vector, list or some other quantity.

Broadly speaking Optimisation requires us to consider four things

- A *metric*. This addresses the question “Optimisation *of* what?”. A metric is a single real positive number that acts as a “figure of merit” of our success at optimisation.
- A *model*. This addresses the question “what is the metric applied to?”. A model is a mathematical or computational process that predicts a result which maybe a function, vector, list or some other quantity.
- *Parameters*. This addresses the question “Optimisation *over* what ?”. These are the variable controls of the model. The aim of optimisation is to adjust the parameters until the figure of metric is in some way “as good as we can get”

Broadly speaking Optimisation requires us to consider four things

- A *metric*. This addresses the question “Optimisation *of* what?”. A metric is a single real positive number that acts as a “figure of merit” of our success at optimisation.
- A *model*. This addresses the question “what is the metric applied to?”. A model is a mathematical or computational process that predicts a result which maybe a function, vector, list or some other quantity.
- *Parameters*. This addresses the question “Optimisation *over* what ?”. These are the variable controls of the model. The aim of optimisation is to adjust the parameters until the figure of metric is in some way “as good as we can get”
- An *algorithm*. This addresses the question “How do we optimise?”

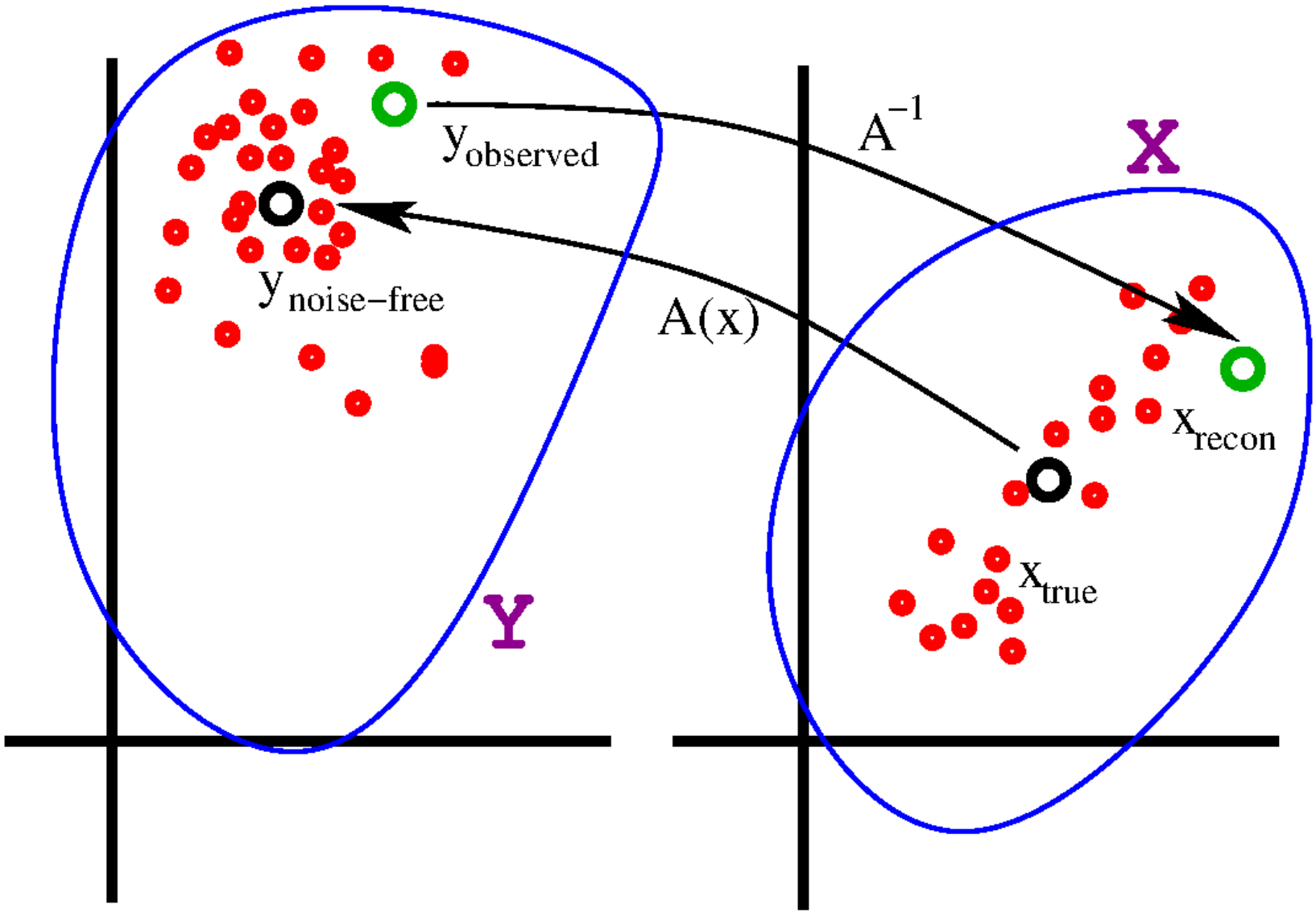
Broadly speaking Optimisation requires us to consider four things

- A *metric*. This addresses the question “Optimisation *of* what?”. A metric is a single real positive number that acts as a “figure of merit” of our success at optimisation.
- A *model*. This addresses the question “what is the metric applied to?”. A model is a mathematical or computational process that predicts a result which maybe a function, vector, list or some other quantity.
- *Parameters*. This addresses the question “Optimisation *over* what ?”. These are the variable controls of the model. The aim of optimisation is to adjust the parameters until the figure of metric is in some way “as good as we can get”
- An *algorithm*. This addresses the question “How do we optimise?”

And for inverse problems we need a fifth thing : Data !

Inverse Problems in Imaging

Introduction : overall picture



Key concepts :

- A space Y, ("Data Space"), where the data lives. Here the "distance" between points is based on the statistics of the measurement noise.
- A space X, ("Solution Space"), where the solution (i.e. the images) live. Here we have different concepts of the "distance" between images. Commonly used
  1. Peak Signal to Noise Ratio (PSNR)
  2. Structural Similarity Index Measure (SSIM)

  There are other choices.
- X and Y might be the same. E.g. Image Deblurring, Image Denoising, Inpainting etc. These are *Image Processing* problems.
- In other cases Y is quite different, e.g. in tomography, geophysical imaging etc.

At the heart of solving inverse problems is to understand the *Forward Problem*

$$y = A(x)$$

A may be linear or non-linear.
A takes an element of solution space X (i.e. an image) and predicts the corresponding element of data space Y.

- In the case of **denoising** A is just the identity operator
- In the case of **deblurring** A is a blurring operator (stationary or non-stationary)
- In the case of **inpainting** A is a subsampling operator (like a mask)
- In the case of **X-ray tomography** A represents integration along lines.

A major part of the field of inverse problems is to formulate the forward problem correctly, and then to develop a computational model for it.
For example in Geophysics imaging it involves solving a Partial Differential Equation in 3D.

The forward problem may often be the computational bottleneck in solving inverse problems.

We want to go the other way : find x from the data y

$$x_{\text{recon}} = A^{-1}(y_{\text{observed}})$$

That's the hard part !
A simple idea : discretise x and y into vectors, then A should be a matrix. Then invert this. We call this the "Brute Force" method!

- But matrix inversion could be very costly. In fact even holding a matrix representation of the operator in memory could be intractable. E.g. 10^6 pixels and 10^6 data makes A of size $10^6 \times 10^6$. About a terabyte.
- We need smarter ideas $\Rightarrow$ "Matrix-Free" methods.
- But sometimes we might do the Brute Force method on a scaled down problem as a kind of "Sanity Check".

If we had the ground truth x_{true} and apply the forward model we get *noise-free* data

$$y_{\text{noise}-\text{free}} = A(x_{\text{true}})$$

If we have "solved" the inverse problem, i.e. we have a method implementing A^{-1}, then we will get x_{true} back, obviously.
However this can be misleading. In fact it is sometimes called an *Inverse Crime*. We have neglected the effect of noise. In practice the observations are corrupted by noise (assume this is additive)

$$y_{\text{observed}} = A(x_{\text{true}}) + \eta$$

The noise η is assumed drawn from a distribution $\eta \sim P(\eta)$.
The reconstruction error $e = x_{recon} - x_{true}$ is then also from some distribution.
But what form is it ?

If we repeat solving the inverse problem with different instantiations of the measurement noise (e.g. just doing different physical experiments, or adding different samples of noise from the model $P(\eta)$), then we will get *samples* of the reconstructed image drawn from an (unknown) *posterior probability distribution* $P_{\text{post}}(x)$.

$$y^{(k)}_{\text{observed}} = A(x_{\text{true}}) + \eta^{(k)} \Leftrightarrow x^{(k)}_{\text{recon}} = A^{-1}(y^{(k)}_{\text{observed}})$$

We might expect that if the noise was well behaved, e.g. Gaussian white noise, then so would be the posterior $P_{\text{post}}(x)$.
However, it usually turns out that posterior distribution of the error is far from being white noise.
The good news is that the mean of $P_{\text{post}}(x)$ is actually the correct solution x_{true}
The bad news is that our confidence estimates of this solution are usually terrible!

The reason is because the forward operator **does not propagate information uniformly.**

From the Bayesian perspective, if we take an observation and invert it, we get the *Maximum Likelihood* solution : i.e. x_{recon} is "most likely" to match the data y_{observed}.
If we take a set of data and reconstructions $\{y^{(k)}_{\text{observed}}, x^{(k)}_{\text{recon}}\}$ then the *Conditional Mean*

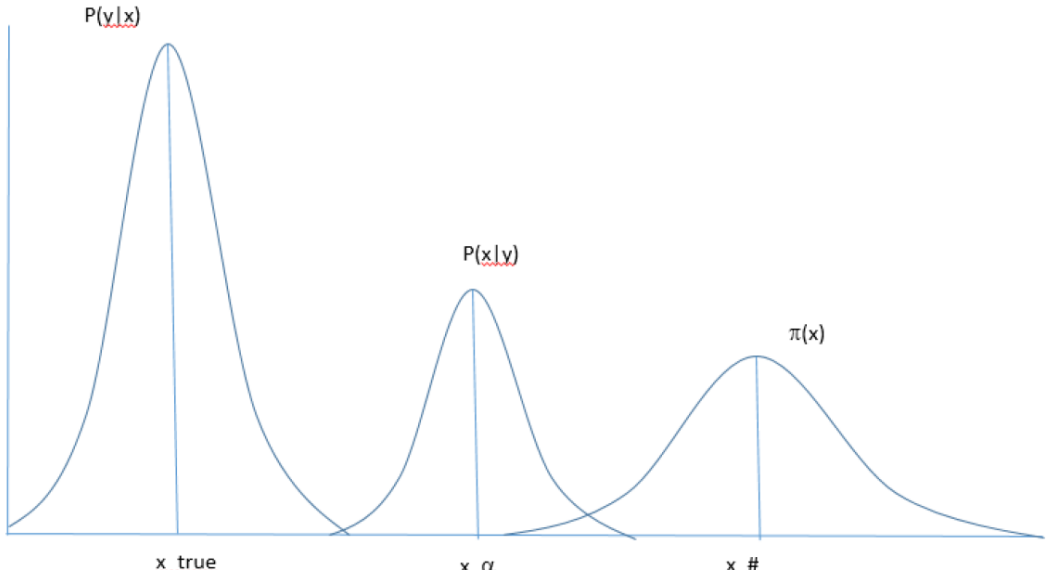
$$x_{\text{CM}} = \frac{1}{N}\sum_{k=1}^{N} x^{(k)}_{\text{recon}}$$

converges to x_{true} in the limit as $N \to \infty$.
However our estimate of covariance

$$C = \frac{1}{N}\sum_{k=1}^{N}\left(x^{(k)}_{\text{recon}} - x_{\text{CM}}\right)\left(x^{(k)}_{\text{recon}} - x_{\text{CM}}\right)^{\text{T}}$$

Can have eigenvalues tending to ∞

Including instead a *prior distribution* of the solutions $\pi(x)$ produces instead the *Maximum A-Priori* (MAP) estimate.



However the MAP estimate is not equal to x_{true} . It introduces *bias*.

$$x_{\text{bias}} := x_{\text{MAP}} - x_{\text{true}} \neq 0$$

Part of the effort in solving inverse problems is to design (or learn) appropriate priors and determine a tradeoff between bias and variance.

- Introduced Imaging, Inverse Problems, and Optimisation.
- Inverse Problems involve Forward Problems, which can be the computational bottleneck.
- Inverting a Forward Problem produces the Maximum Likelihood solution. It fits the data but may be unstable with respect to noise.
- Adding prior information can stabilise an inverse problem but introduces bias.

Next we need to analyse these concepts more carefully, and make quantitative statements.

Outline

- In this section we introduce some key concepts :
  - Non-Uniqueness
  - Non-Existence
  - Instability of solutions
- We'll look at "toy" examples in two dimensions. This will allow us to analyse in a lot of detail and also visualise the ideas as 2D graphics.
- Some of the ideas are "obvious", but keep in mind how these examples will generalise to large number of dimensions.

Consider the following problem : Solve

$$x_1 + 2x_2 = 5 \tag{2.1}$$

Consider the following problem : Solve

$$x_1 + 2x_2 = 5 \tag{2.1}$$

Obviously, there is no unique solution because the problem has one equation in two unknowns . The set of possible solutions forms a line on the x_1, x_2 plane.

Consider the following problem : Solve

$$x_1 + 2x_2 = 5 \tag{2.1}$$

Obviously, there is no unique solution because the problem has one equation in two unknowns . The set of possible solutions forms a line on the x_1, x_2 plane. Eq.(2.1) corresponds to the matrix equation

$$y = A\boldsymbol{x}$$

where $A = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $y = 5$. A non-square matrix doesn't have an exact inverse.

Consider the following problem : Solve

$$x_1 + 2x_2 = 5 \tag{2.1}$$

Obviously, there is no unique solution because the problem has one equation in two unknowns . The set of possible solutions forms a line on the x_1, x_2 plane. Eq.(2.1) corresponds to the matrix equation

$$y = A\boldsymbol{x}$$

where $A = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $y = 5$. A non-square matrix doesn't have an exact inverse.
For other values of y, the solutions always lie on lines that are parallel to this case.

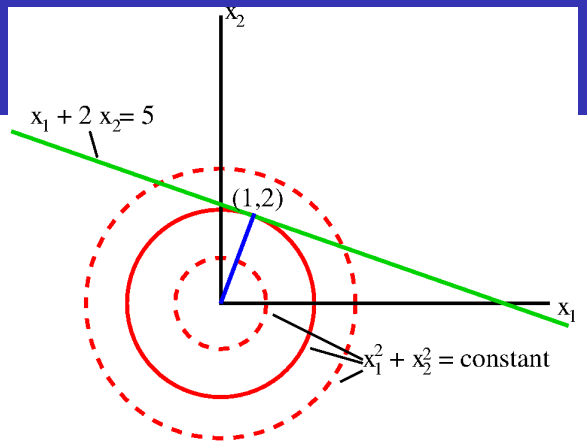
Consider the following problem : Solve

$$x_1 + 2x_2 = 5 \tag{2.1}$$

Obviously, there is no unique solution because the problem has one equation in two unknowns . The set of possible solutions forms a line on the x_1, x_2 plane. Eq.(2.1) corresponds to the matrix equation

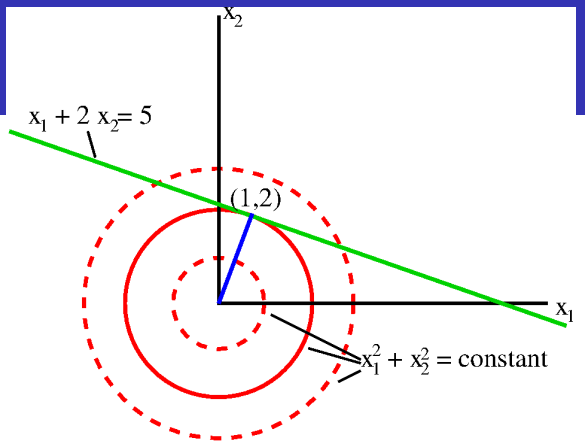
$$y = A\boldsymbol{x}$$

where $A = \begin{pmatrix} 1 & 2 \end{pmatrix}$, $\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$, $y = 5$. A non-square matrix doesn't have an exact inverse.
For other values of y, the solutions always lie on lines that are parallel to this case.
Even if A is square it is not always invertible. So how can we get a useful solution ?



Consider instead the following problem :

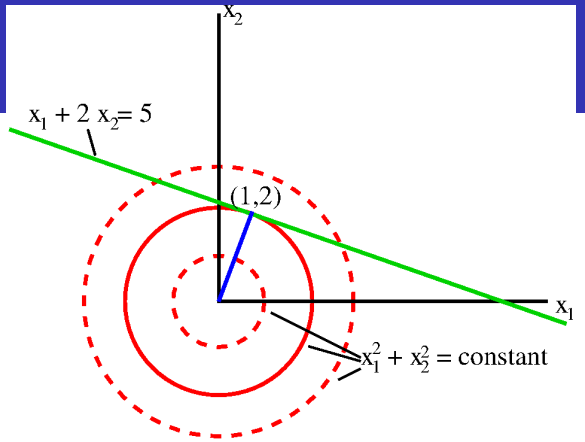
$$\begin{aligned} \text{minimise} \quad & \Phi = x_1^2 + x_2^2 \\ \text{subject to} \quad & x_1 + 2x_2 = 5 \quad (2.2) \end{aligned}$$



Consider instead the following problem :

$$\begin{aligned} \text{minimise} \quad & \Phi = x_1^2 + x_2^2 \\ \text{subject to} \quad & x_1 + 2x_2 = 5 \quad (2.2) \end{aligned}$$

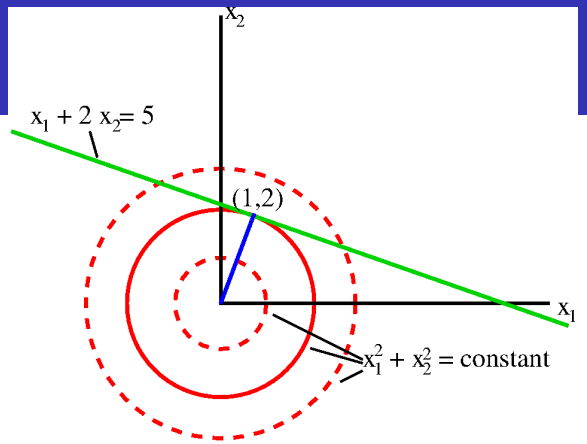
- The level sets of Φ are circles with value equal to the square of their radius. We can find the solution by starting at a large value $\Phi \sim \infty$ and "shrinking" the circles towards the origin. Obviously the minimum of Φ itself is zero, but when including the constraint we get the solution $(x_1, x_2) = (1, 2)$.



Consider instead the following problem :

$$\begin{aligned} \text{minimise} \quad & \Phi = x_1^2 + x_2^2 \\ \text{subject to} \quad & x_1 + 2x_2 = 5 \quad (2.2) \end{aligned}$$

- The level sets of Φ are circles with value equal to the square of their radius. We can find the solution by starting at a large value $\Phi \sim \infty$ and "shrinking" the circles towards the origin. Obviously the minimum of Φ itself is zero, but when including the constraint we get the solution $(x_1, x_2) = (1, 2)$.
- This is an example of *constrained optimisation*. The unique solution is the one where the surface $\Phi =$ constant is **tangent to the constraint** $x_1 + 2x_2 = 5$. The expression $\Phi = x_1^2 + x_2^2$ is a *norm* (i.e. a metric, or distance measure) on the Euclidean space $\mathbb{R}^2$ with basis vectors $\hat{\boldsymbol{x}}_1 = (1, 0)$, $\hat{\boldsymbol{x}}_2 = (0, 1)$.



Consider instead the following problem :

$$\begin{aligned} \text{minimise} \quad & \Phi = x_1^2 + x_2^2 \\ \text{subject to} \quad & x_1 + 2x_2 = 5 \quad (2.2) \end{aligned}$$

- The level sets of Φ are circles with value equal to the square of their radius. We can find the solution by starting at a large value $\Phi \sim \infty$ and "shrinking" the circles towards the origin. Obviously the minimum of Φ itself is zero, but when including the constraint we get the solution $(x_1, x_2) = (1, 2)$.
- This is an example of *constrained optimisation*. The unique solution is the one where the surface $\Phi =$ constant is **tangent to the constraint** $x_1 + 2x_2 = 5$. The expression $\Phi = x_1^2 + x_2^2$ is a *norm* (i.e. a metric, or distance measure) on the Euclidean space $\mathbb{R}^2$ with basis vectors $\hat{\boldsymbol{x}}_1 = (1, 0)$, $\hat{\boldsymbol{x}}_2 = (0, 1)$.
- The solution that we get is the *Minimum Norm* solution. It gives a unique point out of the otherwise infinite possible set of solutions.

Comment : We could also have got the answer using the *Moore-Penrose Inverse* (MPI) of the matrix A.

Comment : We could also have got the answer using the *Moore-Penrose Inverse* (MPI) of the matrix A. The MPI is a way of getting an inverse even if a matrix is non-square. It is defined

$$A^{\dagger} = \begin{cases} A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1} & \text{if } A \text{ has more columns than rows} \\ (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} & \text{if } A \text{ has more rows than columns} \end{cases}.$$

Comment : We could also have got the answer using the *Moore-Penrose Inverse* (MPI) of the matrix A. The MPI is a way of getting an inverse even if a matrix is non-square. It is defined

$$A^{\dagger} = \begin{cases} A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1} & \text{if } A \text{ has more columns than rows} \\ (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} & \text{if } A \text{ has more rows than columns} \end{cases}.$$

In this case we take the first expression and get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, (AA^{\mathrm{T}}) = (5), (AA^{\mathrm{T}})^{-1} = \frac{1}{5} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} 1/5 \\ 2/5 \end{pmatrix} \tag{2.3}$$

Applying $A^{\dagger}$ to the "data" $y = 5$ gives the solution $\boldsymbol{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

Comment : We could also have got the answer using the *Moore-Penrose Inverse* (MPI) of the matrix A. The MPI is a way of getting an inverse even if a matrix is non-square. It is defined

$$A^{\dagger} = \begin{cases} A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1} & \text{if } A \text{ has more columns than rows} \\ (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} & \text{if } A \text{ has more rows than columns} \end{cases}.$$

In this case we take the first expression and get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, (AA^{\mathrm{T}}) = (5), (AA^{\mathrm{T}})^{-1} = \frac{1}{5} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} 1/5 \\ 2/5 \end{pmatrix} \tag{2.3}$$

Applying $A^{\dagger}$ to the "data" $y = 5$ gives the solution $\boldsymbol{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

Note that for any data y the solution always lies along the same line, i.e. $\boldsymbol{x} = \lambda \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ with $\lambda = \frac{y}{5}$.

Comment : We could also have got the answer using the *Moore-Penrose Inverse* (MPI) of the matrix A. The MPI is a way of getting an inverse even if a matrix is non-square. It is defined

$$A^{\dagger} = \begin{cases} A^{\mathrm{T}}(AA^{\mathrm{T}})^{-1} & \text{if } A \text{ has more columns than rows} \\ (A^{\mathrm{T}}A)^{-1}A^{\mathrm{T}} & \text{if } A \text{ has more rows than columns} \end{cases}.$$

In this case we take the first expression and get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, (AA^{\mathrm{T}}) = (5), (AA^{\mathrm{T}})^{-1} = \frac{1}{5} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} 1/5 \\ 2/5 \end{pmatrix} \tag{2.3}$$

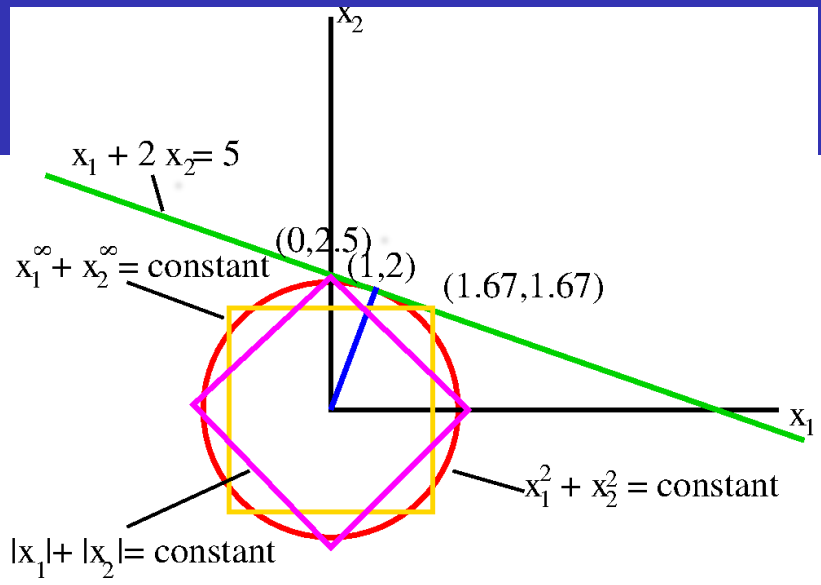
Applying $A^{\dagger}$ to the "data" $y = 5$ gives the solution $\boldsymbol{x} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$.

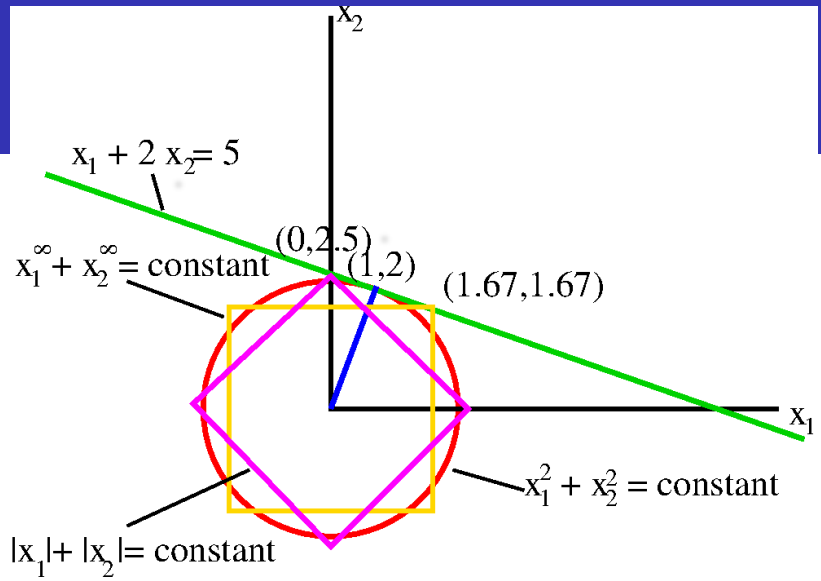
Note that for any data y the solution always lies along the same line, i.e. $\boldsymbol{x} = \lambda \begin{pmatrix} 1 \\ 2 \end{pmatrix}$ with $\lambda = \frac{y}{5}$.

This idea is somewhat obvious. What's interesting is that we get the solution by minimising the particular *Euclidean norm* with the constraint that the value got by applying the forward model to the solution **agrees with the data**.

Now consider that we take a different norm and ask for the solution to

$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$





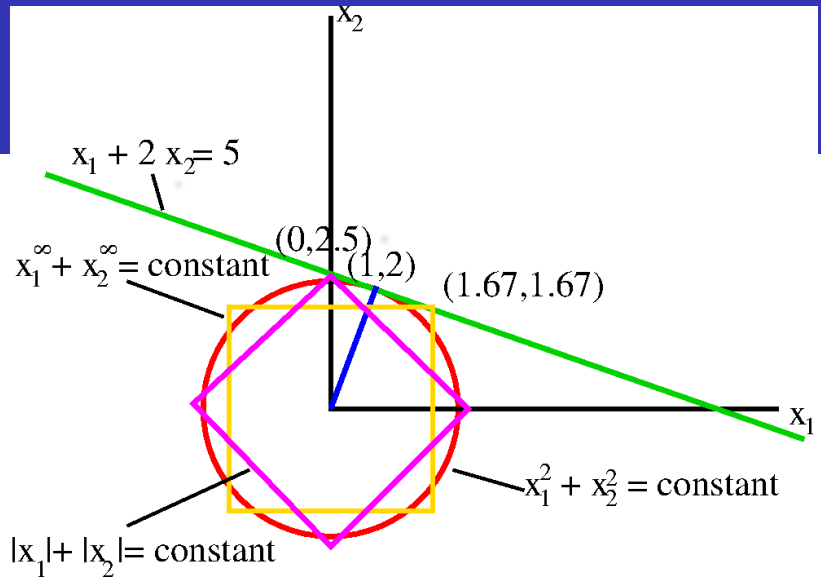
Now consider that we take a different norm and ask for the solution to

$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$

When $1 < p < \infty$ the surfaces $\Phi =$ constant are *convex* and solutions can be found relatively easily (at least "by hand").

Basic Concepts

Generalising the norm



Now consider that we take a different norm and ask for the solution to

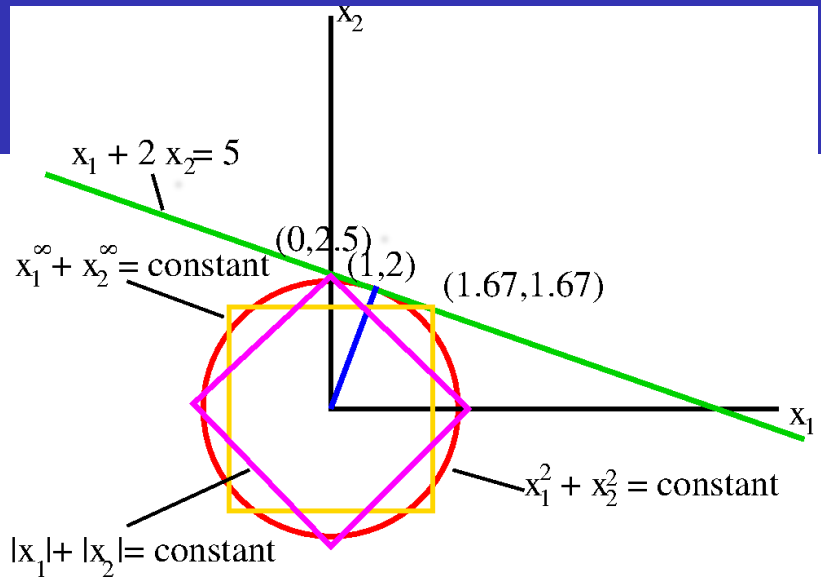
$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$

When $1 < p < \infty$ the surfaces $\Phi =$ constant are *convex* and solutions can be found relatively easily (at least "by hand").

We see that we can obtain any solution between $(0, 2.5)$ at $p = 1$ and $(1\frac{2}{3}, 1\frac{2}{3})$ for $p = \infty$.

Basic Concepts

Generalising the norm



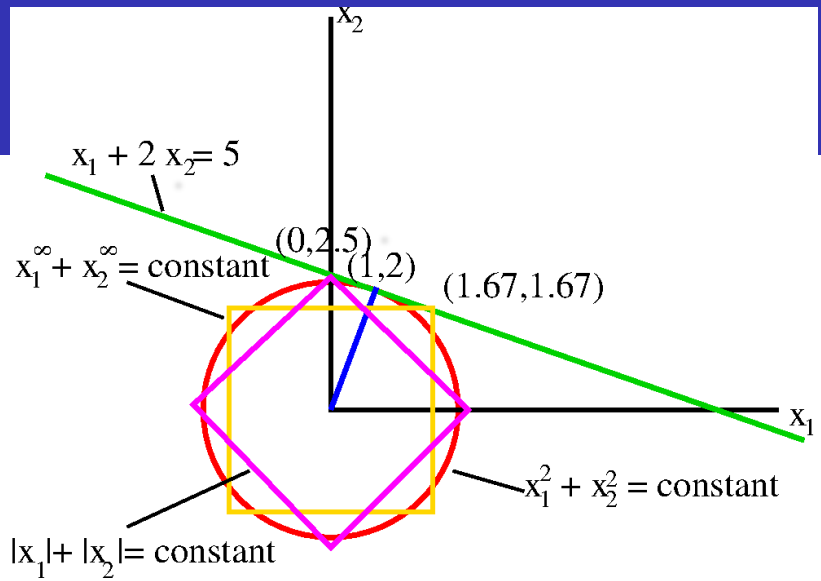
Now consider that we take a different norm and ask for the solution to

$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$

When $1 < p < \infty$ the surfaces $\Phi =$ constant are *convex* and solutions can be found relatively easily (at least "by hand").

We see that we can obtain any solution between $(0, 2.5)$ at $p = 1$ and $(1\frac{2}{3}, 1\frac{2}{3})$ for $p = \infty$.

However, actually calculating the solution is not as simple as solving a linear problem like eq. 2.3. The Moore-Penrose Inverse doesn't work in this case.



Now consider that we take a different norm and ask for the solution to

$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$

When $1 < p < \infty$ the surfaces $\Phi =$ constant are *convex* and solutions can be found relatively easily (at least "by hand").

We see that we can obtain any solution between $(0, 2.5)$ at $p = 1$ and $(1\frac{2}{3}, 1\frac{2}{3})$ for $p = \infty$.

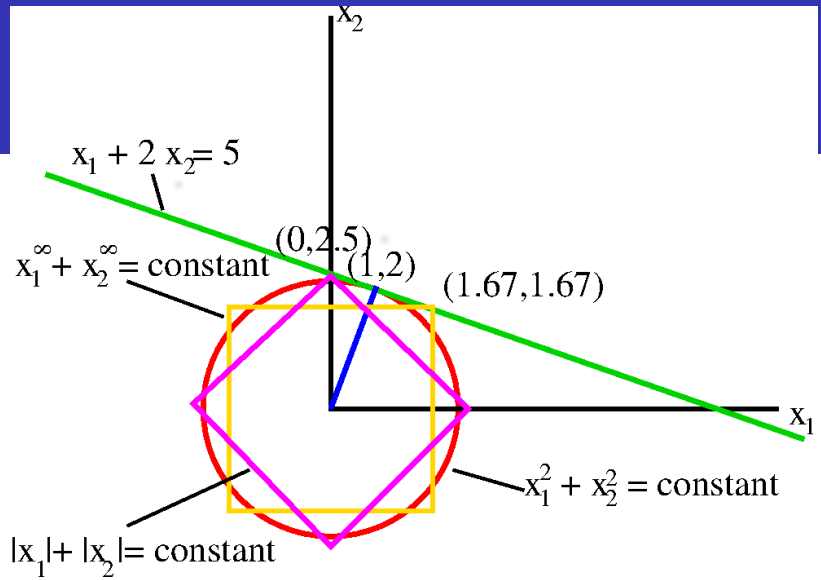
However, actually calculating the solution is not as simple as solving a linear problem like eq. 2.3. The Moore-Penrose Inverse doesn't work in this case.

Notice that for the case $p = 1$ the solution for any equation $ax_1 + bx_2 = c$ will (almost always) lie on the coordinate axes : $\boldsymbol{x} = \begin{pmatrix} 0 \\ c/b \end{pmatrix}$ or $\boldsymbol{x} = \begin{pmatrix} c/a \\ 0 \end{pmatrix}$.

I.e. the solution will (almost always) have only one non-zero component.

Basic Concepts

Generalising the norm



Now consider that we take a different norm and ask for the solution to

$$\begin{aligned} \text{minimise} \quad & \Phi = |x_1|^p + |x_2|^p \\ \text{subject to} \quad & x_1 + 2x_2 = 5\,. \end{aligned} \tag{2.4}$$

When $1 < p < \infty$ the surfaces $\Phi =$ constant are *convex* and solutions can be found relatively easily (at least "by hand").

We see that we can obtain any solution between $(0, 2.5)$ at $p = 1$ and $(1\frac{2}{3}, 1\frac{2}{3})$ for $p = \infty$.

However, actually calculating the solution is not as simple as solving a linear problem like eq. 2.3. The Moore-Penrose Inverse doesn't work in this case.

Notice that for the case $p = 1$ the solution for any equation $ax_1 + bx_2 = c$ will (almost always) lie on the coordinate axes : $\boldsymbol{x} = \begin{pmatrix} 0 \\ c/b \end{pmatrix}$ or $\boldsymbol{x} = \begin{pmatrix} c/a \\ 0 \end{pmatrix}$.

I.e. the solution will (almost always) have only one non-zero component.

By contrast, as $p \to \infty$ the solution components tend to be equal $|x_1| \simeq |x_2|$.

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!
The value of any number to the zeroth power is defined

$$x^0 = \begin{cases} 0 & \text{if } |x| = 0 \\ 1 & \text{otherwise} \end{cases}$$

So for $p = 0$ the functional Φ just counts the number of non zeros in $\boldsymbol{x}$.

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!
The value of any number to the zeroth power is defined

$$x^0 = \begin{cases} 0 & \text{if } |x| = 0 \\ 1 & \text{otherwise} \end{cases}$$

So for $p = 0$ the functional Φ just counts the number of non zeros in $\boldsymbol{x}$.
This means minimising Φ minimises the number of non-zeros in the solution, subject to being consistent with the data.

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!
The value of any number to the zeroth power is defined

$$x^0 = \begin{cases} 0 & \text{if } |x| = 0 \\ 1 & \text{otherwise} \end{cases}$$

So for $p = 0$ the functional Φ just counts the number of non zeros in $\boldsymbol{x}$.
This means minimising Φ minimises the number of non-zeros in the solution, subject to being consistent with the data. The solution is called *sparse* when it has few non-zeros

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!
The value of any number to the zeroth power is defined

$$x^0 = \begin{cases} 0 & \text{if } |x| = 0 \\ 1 & \text{otherwise} \end{cases}$$

So for $p = 0$ the functional Φ just counts the number of non zeros in $\boldsymbol{x}$.
This means minimising Φ minimises the number of non-zeros in the solution, subject to being consistent with the data. The solution is called *sparse* when it has few non-zeros
But as we see, at least in simple cases, we get the same result using $p = 1$ (or any $p \in [0, 1]$).

Comment about *Sparsity*:
For values $0 < p < 1$ the level sets of Φ are *concave*.
Optimisation of non-convex functions is much harder than convex functions.
Nevertheless, we can see that starting with a large value of Φ and shrinking to the origin, the same solutions are obtained as for solving the $p = 1$ case!
The value of any number to the zeroth power is defined

$$x^0 = \begin{cases} 0 & \text{if } |x| = 0 \\ 1 & \text{otherwise} \end{cases}$$

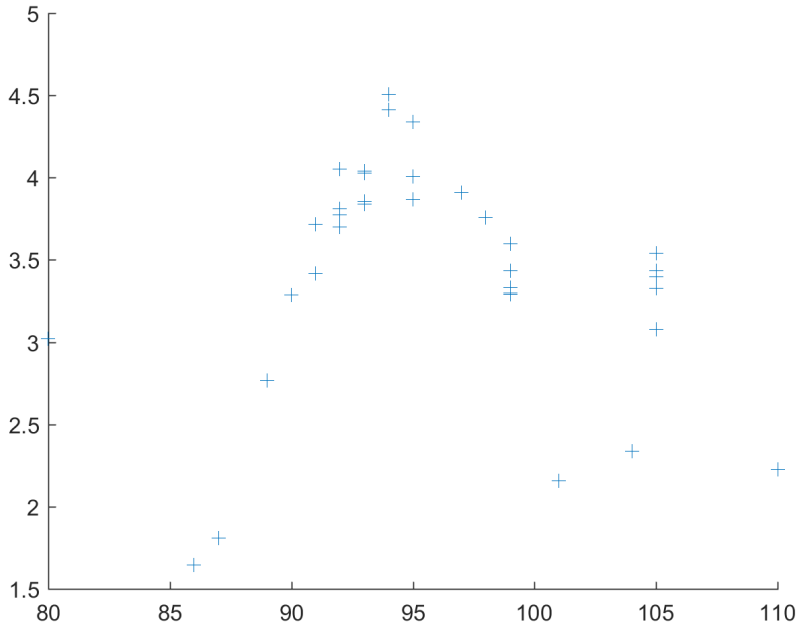
So for $p = 0$ the functional Φ just counts the number of non zeros in $\boldsymbol{x}$.
This means minimising Φ minimises the number of non-zeros in the solution, subject to being consistent with the data. The solution is called *sparse* when it has few non-zeros
But as we see, at least in simple cases, we get the same result using $p = 1$ (or any $p \in [0, 1]$). This motivates the use of the $p = 1$ norm for priors when we want to enforce sparsity.

Model Fitting Example

Generate data, $\{x^{(k)}, y^{(k)}; k = 1 \ldots N\}$ with the model

$$A_{\text{true}}(x) = 1 + \frac{1}{50}x - \frac{1}{3}\sin\left(\frac{x^2}{30}\right) + e^{\left(-\frac{0.2x^3}{40}\right)}$$

Add 20% multiplicative noise
$y = (1 + 0.2r)A_{\text{true}}(x)$, $r \sim \mathcal{N}(0, 1)$.



Fit data with model $A_{\text{model}}(x; \xi) = \sum_{m=0}^{M} \xi_m x^m$ in order to minimise the *Least Squares Error*

$$\Phi = \frac{1}{2}\sum_{i=k}^{N} |y^{(k)} - A_{\text{model}}(x^{(k)}, \xi)|^2 \tag{2.5}$$

The model $A_{\text{model}}(x;\boldsymbol{\xi})$ is non-linear in x but linear in $\boldsymbol{\xi}$).
E.g. for a cubic fit ($M = 4$), Φ is minimised by solving

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$$

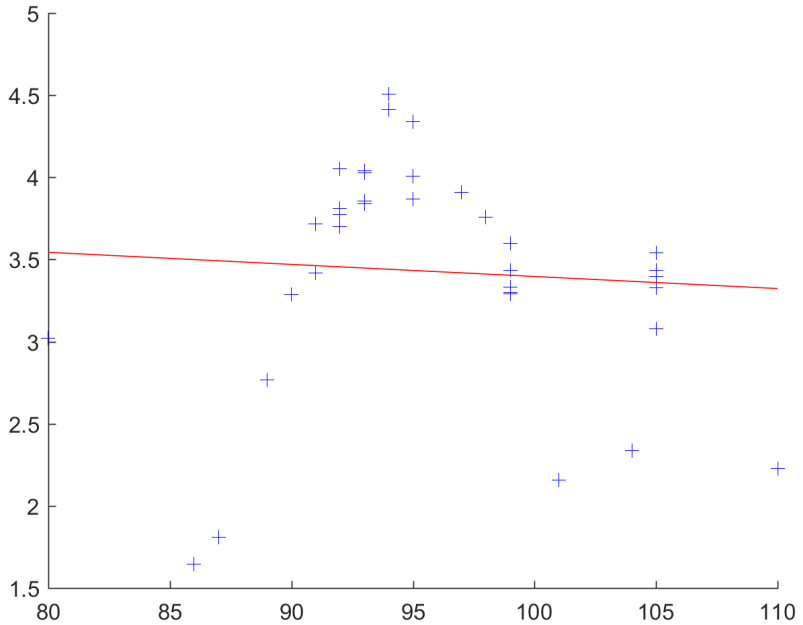
The model $A_{\text{model}}(x;\boldsymbol{\xi})$ is non-linear in x but linear in $\boldsymbol{\xi}$).
E.g. for a cubic fit ($M = 4$), Φ is minimised by solving

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$$

The model $A_{\text{model}}(x;\boldsymbol{\xi})$ is non-linear in x but linear in $\boldsymbol{\xi}$).
E.g. for a cubic fit ($M = 4$), Φ is minimised by solving

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$$

Code example `modelfit.m`
Results $M = 2$ (linear)



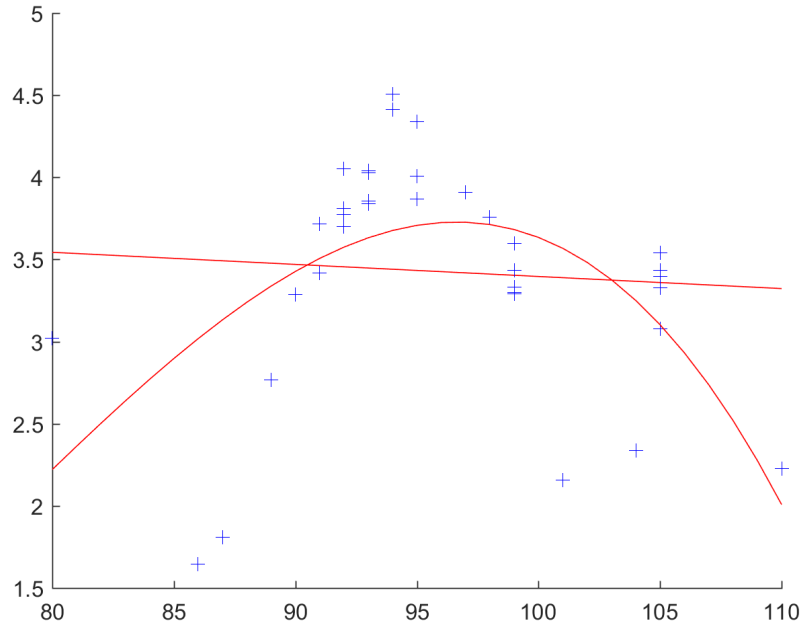
Basic Concepts

Model Fitting Example Results

The model $A_{\text{model}}(x;\boldsymbol{\xi})$ is non-linear in x but linear in $\boldsymbol{\xi}$).
E.g. for a cubic fit ($M = 4$), Φ is minimised by solving

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$$

Code example `modelfit.m`
Results $M = 4$ (cubic)



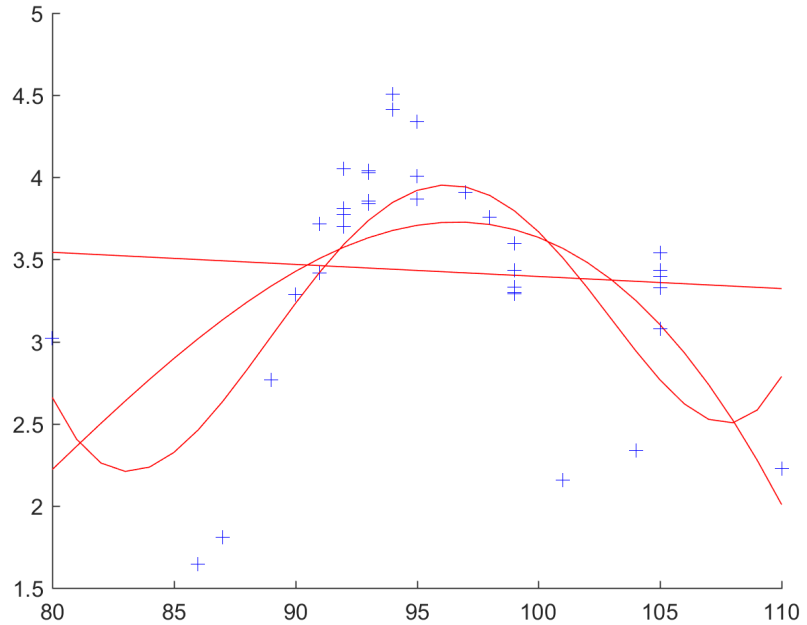
Basic Concepts

Model Fitting Example Results

The model $A_{\text{model}}(x;\boldsymbol{\xi})$ is non-linear in x but linear in $\boldsymbol{\xi}$.
E.g. for a cubic fit ($M = 4$), Φ is minimised by solving

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & x_N^3 \end{pmatrix} \begin{pmatrix} \xi_0 \\ \xi_1 \\ \xi_2 \\ \xi_3 \end{pmatrix}$$

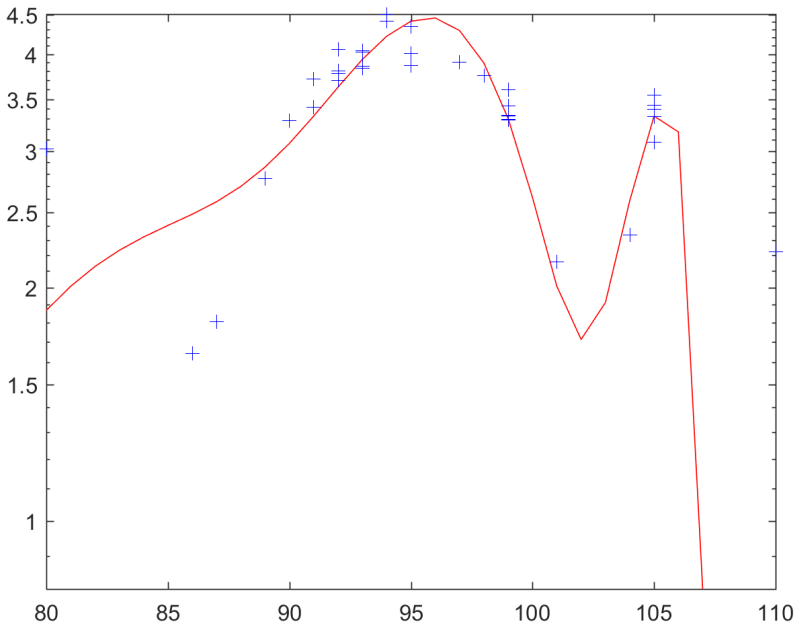
Code example `modelfit.m`
Results $M = 6$ (5^{th} order)



Increasing the number of fitting parameters leads to *Over-fitting*

Code example `modelfit.m`

Results $M = 36$

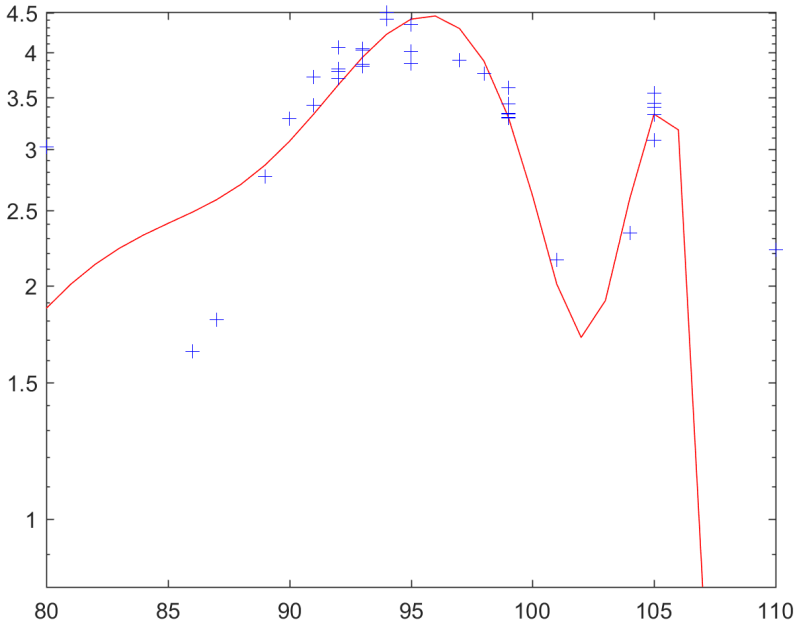


- Closer fitting to the data leads to undesirable features in the solution (noise propagation).

Increasing the number of fitting parameters leads to *Over-fitting*

Code example `modelfit.m`

Results $M = 36$

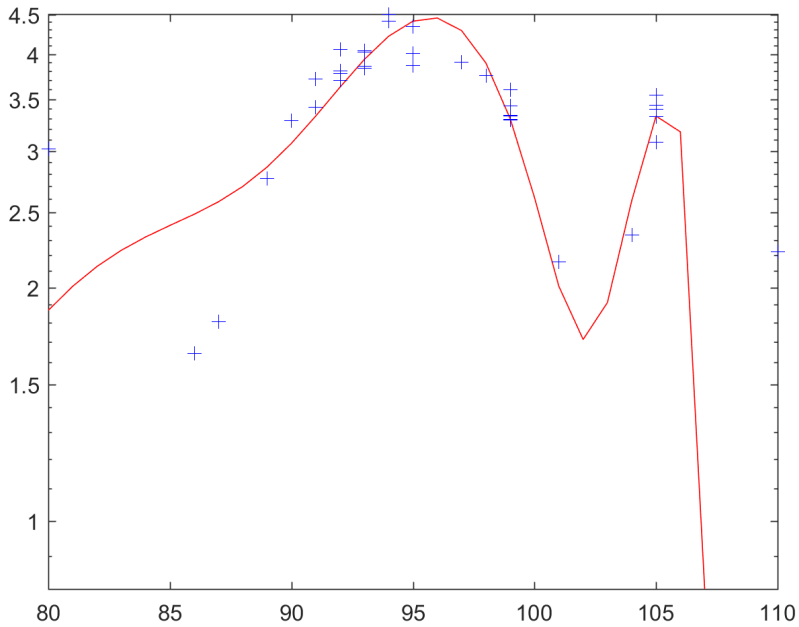


- Closer fitting to the data leads to undesirable features in the solution (noise propagation).
- We should choose the complexity of the model appropriately (number of parameters).

Increasing the number of fitting parameters leads to *Over-fitting*

Code example `modelfit.m`

Results $M = 36$



- Closer fitting to the data leads to undesirable features in the solution (noise propagation).
- We should choose the complexity of the model appropriately (number of parameters).
- Alternatively : use *Regularisation*. See later !

- Least-square fitting corresponds to a belief that the noise is i.i.d. Gaussian white noise.

- Least-square fitting corresponds to a belief that the noise is i.i.d. Gaussian white noise.
- More generally we could define a *data-fitting* functional

$$\Phi = \mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi}))\,. \tag{2.6}$$

- Least-square fitting corresponds to a belief that the noise is i.i.d. Gaussian white noise.
- More generally we could define a *data-fitting* functional

$$\Phi = \mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi}))\,. \tag{2.6}$$

Noise model	data fitting functional
Gaussian white	$\Phi = \frac{1}{2}\sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert^2$
Poissonian	$\Phi = \sum_{i=k}^{N} y^{(k)} \log(A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})) - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})$
Salt and Pepper	$\Phi = \sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert$

- Least-square fitting corresponds to a belief that the noise is i.i.d. Gaussian white noise.
- More generally we could define a *data-fitting* functional

$$\Phi = \mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi}))\,. \tag{2.6}$$

Noise model	data fitting functional
Gaussian white	$\Phi = \frac{1}{2}\sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert^2$
Poissonian	$\Phi = \sum_{i=k}^{N} y^{(k)} \log(A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})) - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})$
Salt and Pepper	$\Phi = \sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert$

- In general, we should analyse (or *learn*) the statistical model of the data and take the negative log of the *likelihood*

$$\mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi})) = -\log P(\boldsymbol{y}|A(\boldsymbol{x}, \boldsymbol{\xi}))$$

- Least-square fitting corresponds to a belief that the noise is i.i.d. Gaussian white noise.
- More generally we could define a *data-fitting* functional

$$\Phi = \mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi}))\,. \tag{2.6}$$

Noise model	data fitting functional
Gaussian white	$\Phi = \frac{1}{2}\sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert^2$
Poissonian	$\Phi = \sum_{i=k}^{N} y^{(k)} \log(A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})) - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})$
Salt and Pepper	$\Phi = \sum_{i=k}^{N} \lvert y^{(k)} - A_{\text{model}}(x^{(k)}, \boldsymbol{\xi})\rvert$

- In general, we should analyse (or *learn*) the statistical model of the data and take the negative log of the *likelihood*

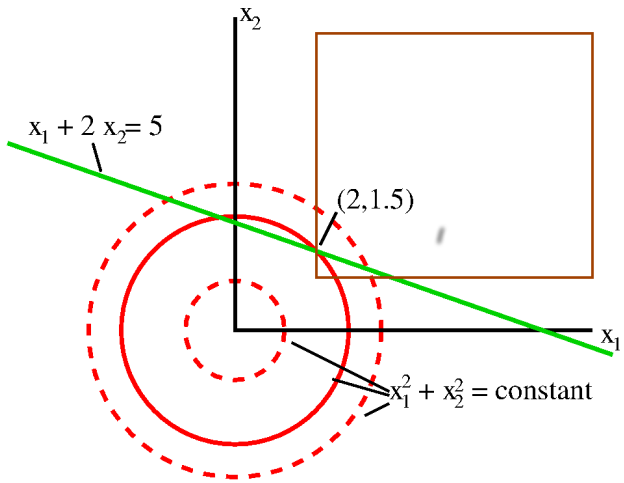
$$\mathcal{D}(\boldsymbol{y}, A(\boldsymbol{x}, \boldsymbol{\xi})) = -\log P(\boldsymbol{y}|A(\boldsymbol{x}, \boldsymbol{\xi}))$$

- There is no reason to assume that the number of model parameters of the model is the same as the number of data. E.g when reconstructing 3D tomographic images, the number of pixels (voxels) is in some sense "arbitrary". How many should we choose ?

Consider the same problem as before but introduce bound on the solution

$$\begin{aligned} \text{minimise} \quad & \Phi = x_1^2 + x_2^2 \\ \text{subject to} \quad & x_1 + 2x_2 = 5 \\ & 2 \leq x_1 \leq 4 \\ & 1 \leq x_2 \leq 5 \end{aligned}$$

which has solution $(x_1, x_2) = (2, 1.5)$,



Unlike eq. 2.2 this cannot be solved by a simple linear method like eq. 2.3. Instead we need *Iterative methods*.

Now we define explicitly the three aspects we mentioned before

Problem 1 : Non-Uniqueness

Solve

$$x_1 + x_2 = 2 \tag{2.7}$$

This problem has an infinite number of solutions. The set of equations are *underdetermined*. To find a single solution we need to add constraints and use a constrained optimisation method.

Now we define explicitly the three aspects we mentioned before

Problem 1 : Non-Uniqueness

Solve

$$x_1 + x_2 = 2 \tag{2.7}$$

This problem has an infinite number of solutions. The set of equations are *underdetermined*. To find a single solution we need to add constraints and use a constrained optimisation method.

Problem 2 : Non-Existance

Solve

$$\begin{aligned} x_1 &= 1 \\ x_1 &= 3 \end{aligned} \tag{2.8}$$

This problem has no solution. The set of equations are *overdetermined*. We can find a possible solution using unconstrained optimisation.

Non-Existance is the mirror-image of non-uniqueness.

Non-Existance is the mirror-image of non-uniqueness.
Problem 2 in matrix form is

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_1$$

Let's use the Moore-Penrose inverse in its overdetermined form :

Non-Existance is the mirror-image of non-uniqueness.
Problem 2 in matrix form is

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_1$$

Let's use the Moore-Penrose inverse in its overdetermined form : In this case we get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 1 \end{pmatrix}, (A^{\mathrm{T}}A) = (2), (A^{\mathrm{T}}A)^{-1} = \frac{1}{2} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix} \qquad (2.9)$$

Non-Existance is the mirror-image of non-uniqueness.
Problem 2 in matrix form is

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_1$$

Let's use the Moore-Penrose inverse in its overdetermined form : In this case we get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 1 \end{pmatrix}, (A^{\mathrm{T}}A) = (2), (A^{\mathrm{T}}A)^{-1} = \frac{1}{2} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{2.9}$$

Applying $A^{\dagger}$ to the "data" $y = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$ gives the solution $x_1 = 2$.

Non-Existance is the mirror-image of non-uniqueness.
Problem 2 in matrix form is

$$\begin{pmatrix} 1 \\ 3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} x_1$$

Let's use the Moore-Penrose inverse in its overdetermined form : In this case we get

$$A^{\mathrm{T}} = \begin{pmatrix} 1 & 1 \end{pmatrix}, (A^{\mathrm{T}}A) = (2), (A^{\mathrm{T}}A)^{-1} = \frac{1}{2} \quad \Rightarrow \quad A^{\dagger} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \end{pmatrix} \tag{2.9}$$

Applying $A^{\dagger}$ to the "data" $y = \begin{pmatrix} 1 \\ 3 \end{pmatrix}$ gives the solution $x_1 = 2$.

This is the "direct" solution. But it can also be solved by optimisation.

Consider the following problem :

$$\Phi = (x_1 - 1)^2 + (x_1 - 3)^2 \rightarrow \min \tag{2.10}$$

Consider the following problem :

$$\Phi = (x_1 - 1)^2 + (x_1 - 3)^2 \rightarrow \text{ min} \tag{2.10}$$

It is easy to see we cannot get a solution with $\Phi = 0$ because we can solve a quadratic equation exactly

$$\begin{aligned} (x_1 - 1)^2 + (x_2 - 3)^2 &= 0 \\ \Rightarrow 2x_1^2 - 8x_1 + 10 &= 0 \\ \Rightarrow x_1 &= \frac{4 \pm \sqrt{16 - 20}}{2} \\ &= 2 \pm 2i \end{aligned}$$

i.e. the solution has no real roots.

Consider the following problem :

$$\Phi = (x_1 - 1)^2 + (x_1 - 3)^2 \rightarrow \text{ min} \tag{2.10}$$

It is easy to see we cannot get a solution with $\Phi = 0$ because we can solve a quadratic equation exactly

$$\begin{aligned} (x_1 - 1)^2 + (x_2 - 3)^2 &= 0 \\ \Rightarrow 2x_1^2 - 8x_1 + 10 &= 0 \\ \Rightarrow x_1 &= \frac{4 \pm \sqrt{16 - 20}}{2} \\ &= 2 \pm 2i \end{aligned}$$

i.e. the solution has no real roots.
If we want a real valued solution we can find the minimum :

$$x_1 = 2 \Leftrightarrow \Phi(x_1) = 2$$

This is the *least squares solution*.

Comments

- Non-uniqueness **always occurs** in real life problems because the number of unknowns is “infinite” even if the measurements are finite. We make a problem finite by discretisation, but this implicitly means that we have chosen a particular *basis* for the solution space X.

Comments

- Non-uniqueness **always occurs** in real life problems because the number of unknowns is “infinite” even if the measurements are finite. We make a problem finite by discretisation, but this implicitly means that we have chosen a particular *basis* for the solution space X.
- Non-Existence means that the model cannot match the data exactly. There is some component of the data outside the range of the model.

Comments

- Non-uniqueness **always occurs** in real life problems because the number of unknowns is “infinite” even if the measurements are finite. We make a problem finite by discretisation, but this implicitly means that we have chosen a particular *basis* for the solution space X.
- Non-Existence means that the model cannot match the data exactly. There is some component of the data outside the range of the model.
- Physics tells us that a solution must exist, but our model may be incomplete. This is not measurement noise but *modelling error* (also known as *systematic error*).

Comments

- Non-uniqueness **always occurs** in real life problems because the number of unknowns is “infinite” even if the measurements are finite. We make a problem finite by discretisation, but this implicitly means that we have chosen a particular *basis* for the solution space X.
- Non-Existence means that the model cannot match the data exactly. There is some component of the data outside the range of the model.
- Physics tells us that a solution must exist, but our model may be incomplete. This is not measurement noise but *modelling error* (also known as *systematic error*).
- In practice we can “fix” non-existence by using least-square. I.e. we find the solution that matches the data “as close as possible”. But this may still give problems when modelling error is present.

Problem 3 : Discontinuous dependence on data

Consider this problem :

$$\mathrm{A}\boldsymbol{x} = \begin{pmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{2.11}$$

Problem 3 : Discontinuous dependence on data

Consider this problem :

$$\mathrm{A}\boldsymbol{x} = \begin{pmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{2.11}$$

The matrix A has determinant ϵ^2 so it has a proper inverse

$$\mathrm{A}^{-1} = \frac{1}{\epsilon^2} \begin{pmatrix} 1 & -(1+\epsilon) \\ -(1-\epsilon) & 1 \end{pmatrix} \tag{2.12}$$

Thus the inverse grows as an inverse square in the parameter ϵ. This problem has a solution that tends towards instability.

Problem 3 : Discontinuous dependence on data

Consider this problem :

$$\mathsf{A}\boldsymbol{x} = \begin{pmatrix} 1 & 1+\epsilon \\ 1-\epsilon & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{2.11}$$

The matrix A has determinant ϵ^2 so it has a proper inverse

$$\mathsf{A}^{-1} = \frac{1}{\epsilon^2} \begin{pmatrix} 1 & -(1+\epsilon) \\ -(1-\epsilon) & 1 \end{pmatrix} \tag{2.12}$$

Thus the inverse grows as an inverse square in the parameter ϵ. This problem has a solution that tends towards instability.
We can do an experiment to see this

Algorithm 1 Statistical trial of ill-posed matrix inversion

for all trials $i = 1 \ldots N$ **do**

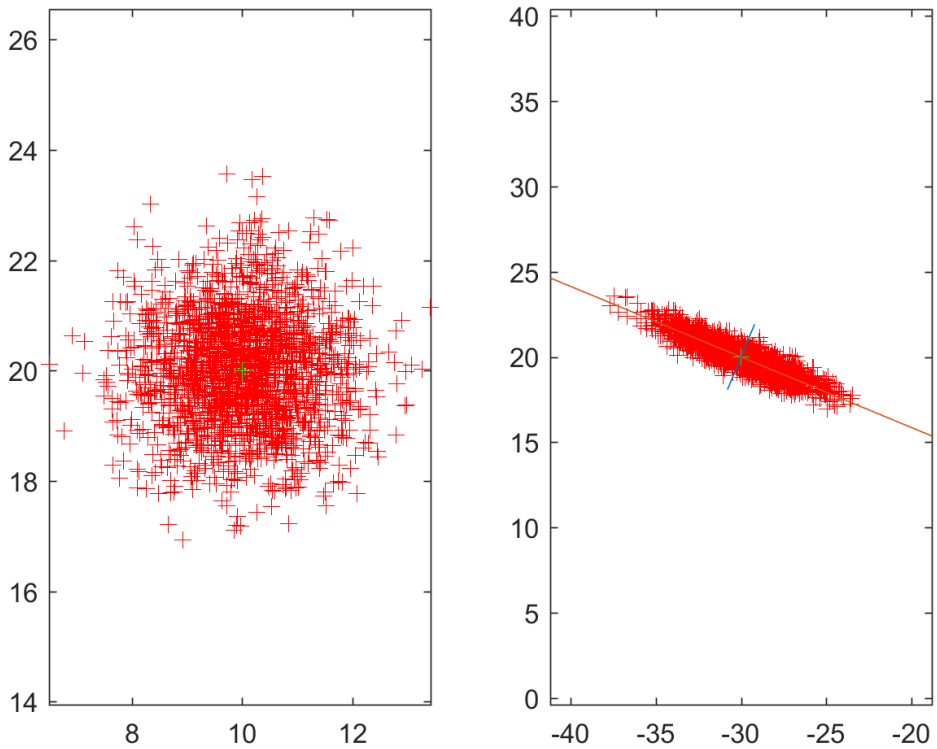
draw a random vector of length 2 with Gaussian probability $\eta_i \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\right)$

generate data $b_i = (10, 20) + \eta_i$

solve $\mathsf{A}\boldsymbol{x}_i = b_i$

end for

Code example `iptoy.m`

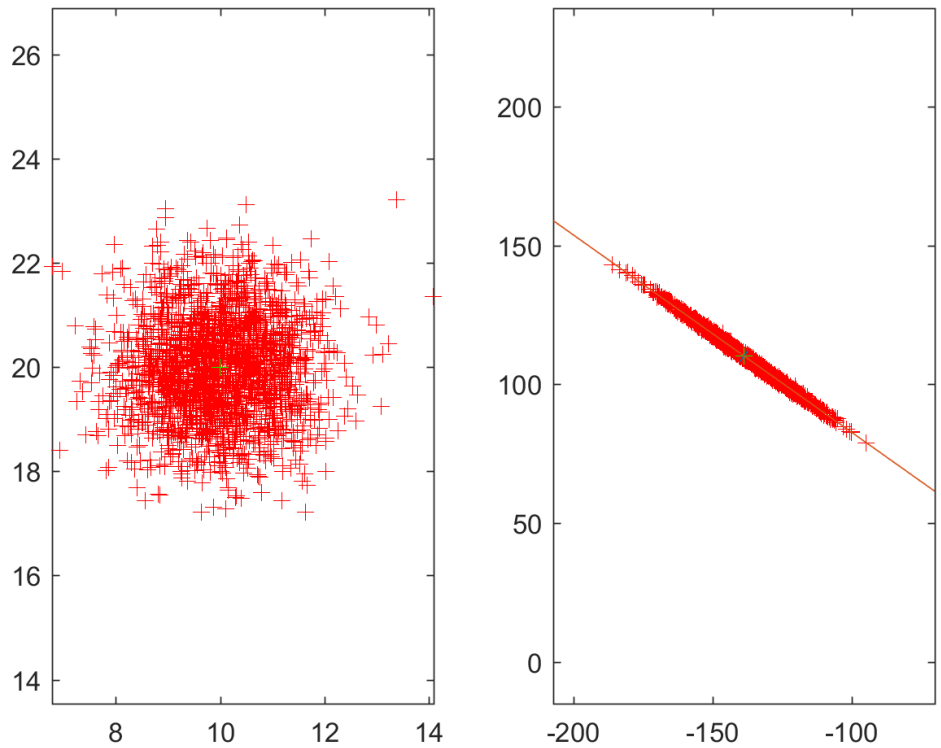


Results for $\epsilon = 1$

Sample Covariance eigenvalues : $[0.1696, 5.7801]$,

eigenvectors : $\begin{pmatrix} 0.3840 \\ 0.9233 \end{pmatrix}, \begin{pmatrix} -0.9233 \\ 0.3840 \end{pmatrix}$

Code example `iptoy.m`

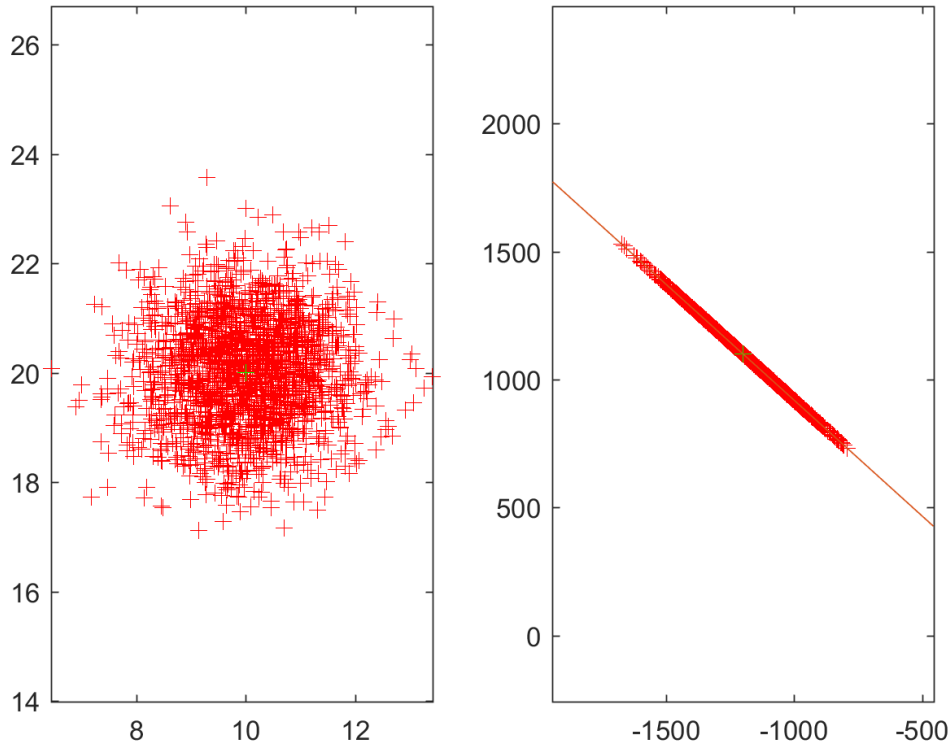


Results for $\epsilon = 0.35$

Sample Covariance eigenvalues : $[0.2296, 261.0188]$,

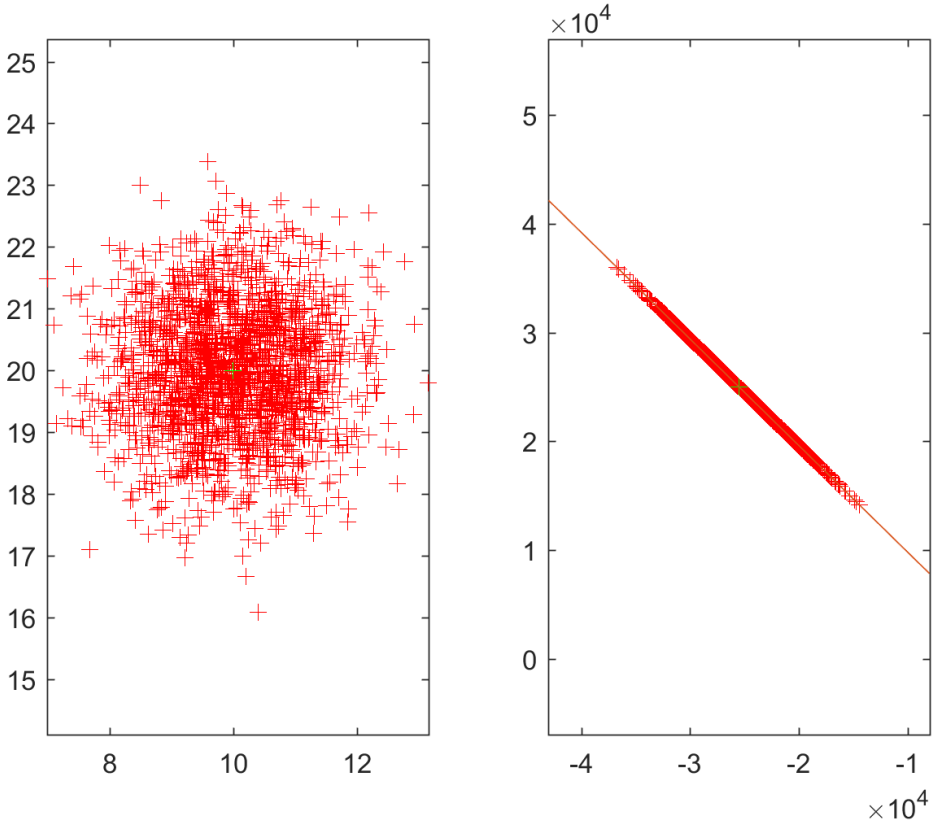
eigenvectors : $\begin{pmatrix} 0.5785 \\ 0.8157 \end{pmatrix}, \begin{pmatrix} -0.8157 \\ 0.5785 \end{pmatrix}$

Code example `iptoy.m`



Results for $\epsilon = 0.1$
Sample Covariance eigenvalues : [0.2507, , 38084],
eigenvectors : $\begin{pmatrix} 0.6710 \\ 0.7415 \end{pmatrix}, \begin{pmatrix} -0.7415 \\ 0.6710 \end{pmatrix}$

Code example `iptoy.m`



Results for $\epsilon = 0.0202$

Sample Covariance eigenvalues : $[0.2556, 2.3806 \times 10^7]$,

eigenvectors : $\begin{pmatrix} 0.6999 \\ 0.7142 \end{pmatrix}, \begin{pmatrix} -0.7142 \\ 0.6999 \end{pmatrix}$

Comments

- As $\epsilon \rightarrow 0$ the distribution of solutions becomes more and more elongated and also rotates clockwise.

Comments

- As $\epsilon \rightarrow 0$ the distribution of solutions becomes more and more elongated and also rotates clockwise.
- Can we determine this behaviour from the matrix A itself ? If so what is the limiting behaviour ?

Comments

- As $\epsilon \rightarrow 0$ the distribution of solutions becomes more and more elongated and also rotates clockwise.
- Can we determine this behaviour from the matrix A itself ? If so what is the limiting behaviour ?
- It turns out we can predict this behaviour accurately using the *Singular Value Decomposition*.

Since we have a lot of samples of the data, can we use **learning** to improve the results ?

Basic Concepts

Instability Results : Comparison to Learning

Since we have a lot of samples of the data, can we use **learning** to improve the results ? Results using a simple network and single true data with added noise

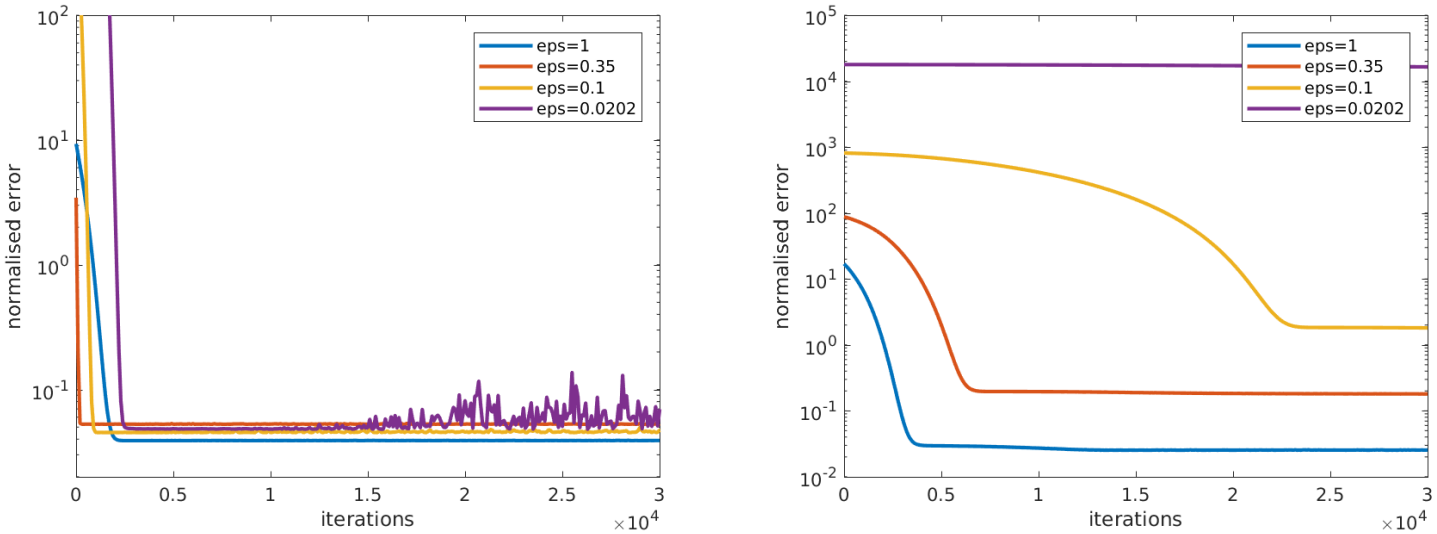


Figure: Convergence of training procedure for forward and inverse network for a test set of 128 samples. Left: forward problem, right: inverse problem

Since we have a lot of samples of the data, can we use **learning** to improve the results ? Results using a simple network and single true data with added noise

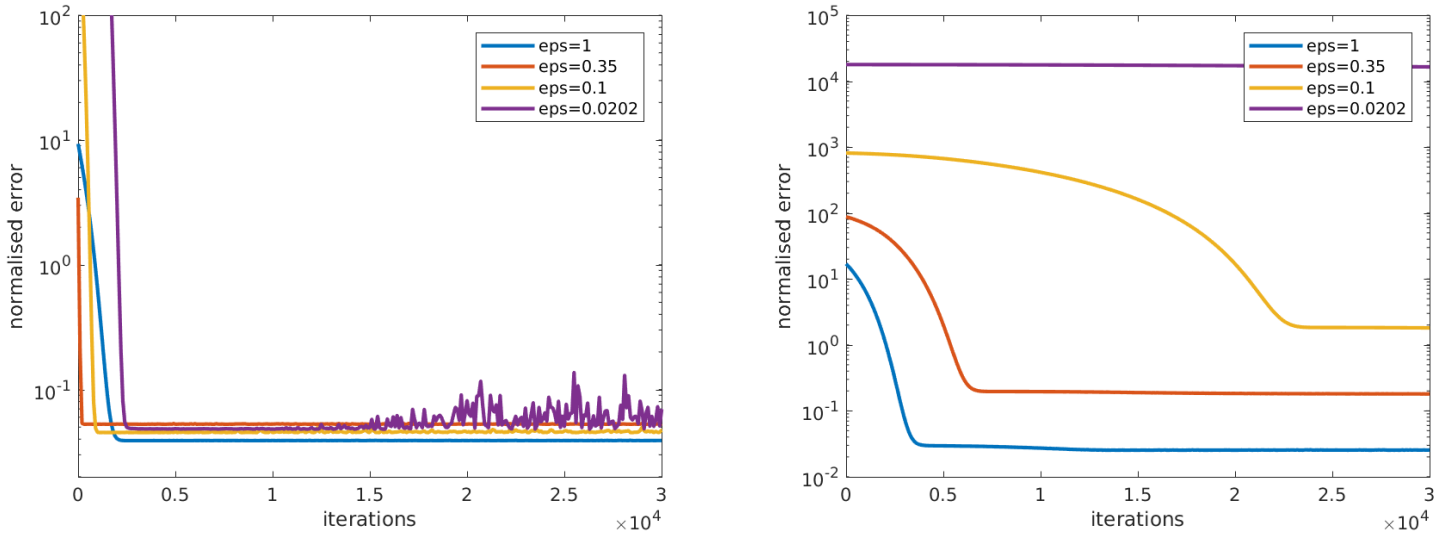


Figure: Convergence of training procedure for forward and inverse network for a test set of 128 samples. Left: forward problem, right: inverse problem

Results show there is a marked difference between the problems!

As a more general idea, we also draw the true data from a distribution

As a more general idea, we also draw the true data from a distribution

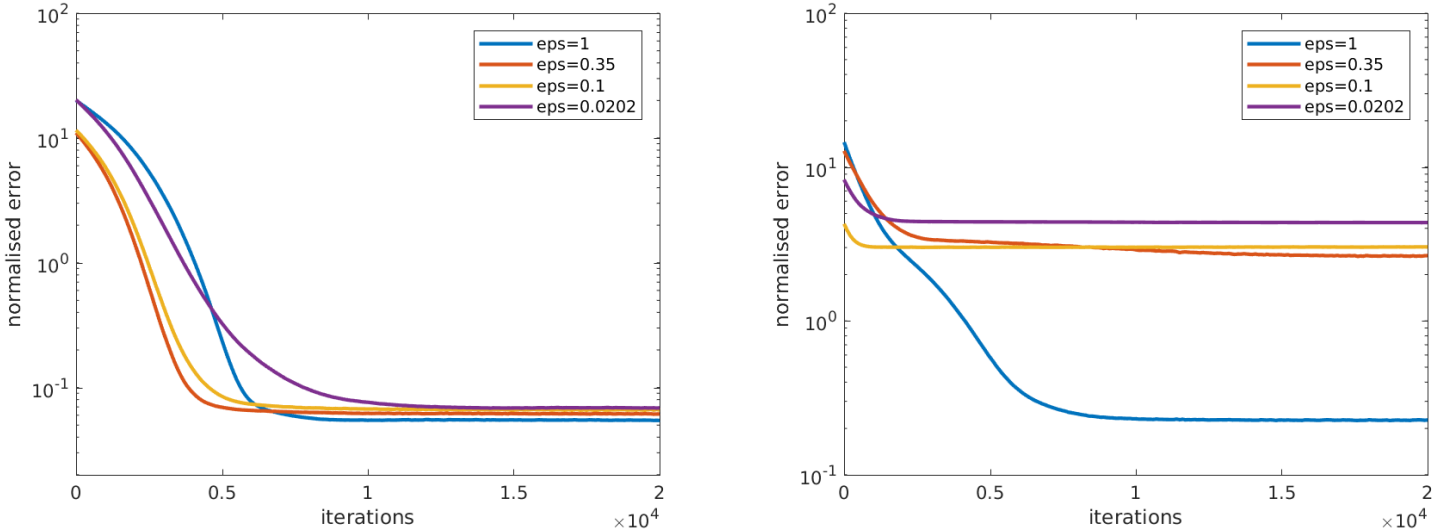


Figure: Convergence of training procedure for forward and inverse network for a test set of 128 samples. Left: forward problem, right: inverse problem

As a more general idea, we also draw the true data from a distribution

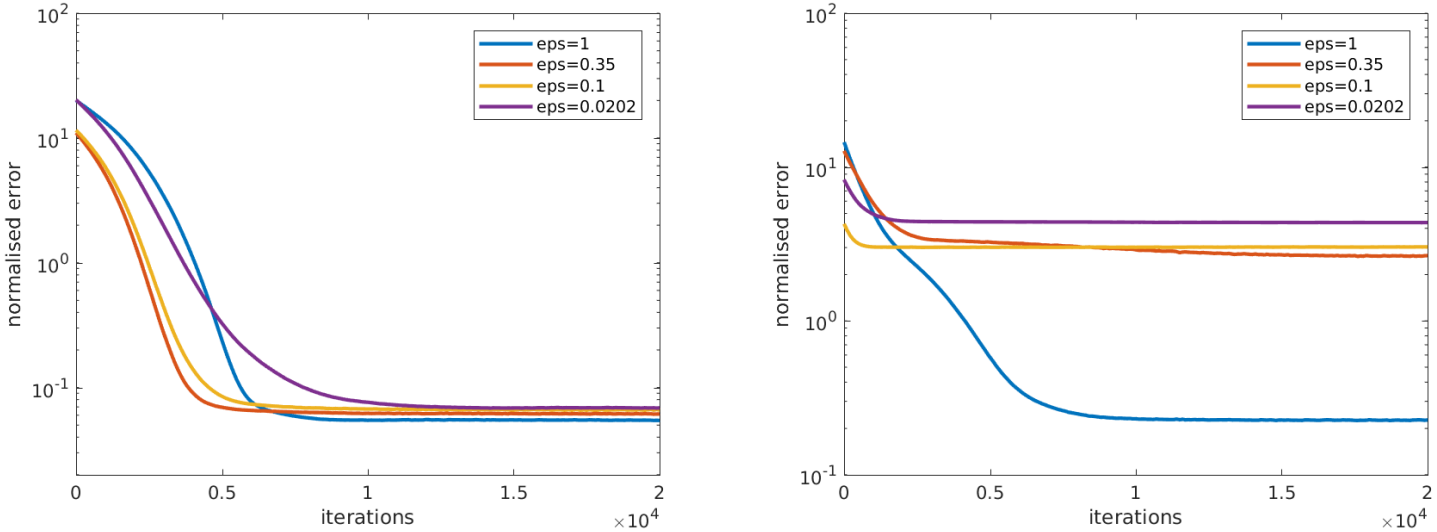


Figure: Convergence of training procedure for forward and inverse network for a test set of 128 samples. Left: forward problem, right: inverse problem

Forward problem is slower to converge, but the inverse problem still is unstable when ϵ is too small

Basic Concepts

Instability Results : Comparison to Learning

As a more general idea, we also draw the true data from a distribution

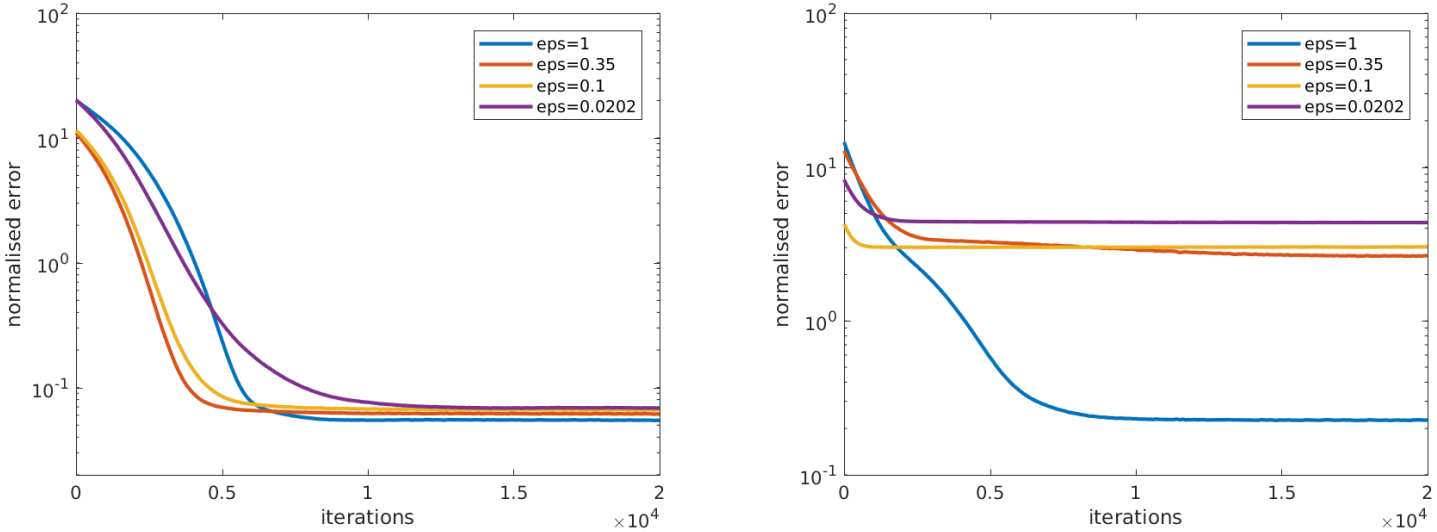


Figure: Convergence of training procedure for forward and inverse network for a test set of 128 samples. Left: forward problem, right: inverse problem

Forward problem is slower to converge, but the inverse problem still is unstable when ϵ is too small

Could different network architectures produce better results ?

The eigenvalues of A can be found by solving

$$\lambda^2 - 2\lambda + \epsilon^2 = 0 \quad \Rightarrow \quad \lambda = 1 \pm \sqrt{1-\epsilon^2}.$$

with corresponding eigenvectors (unnormalised)

$$\boldsymbol{e}_1 = \begin{pmatrix} \sqrt{1-\epsilon^2} \\ 1-\epsilon \end{pmatrix}, \quad \boldsymbol{e}_2 = \begin{pmatrix} -\sqrt{1-\epsilon^2} \\ 1-\epsilon \end{pmatrix}$$

Note: Since A is not symmetric, the eigenvectors are not orthogonal
Taking the limit of these solutions as $\epsilon \to 0$ we find the limiting values for the eigenvalues :

$$\lambda_1 \to 2 - \frac{1}{2}\epsilon, \quad \lambda_2 \to \frac{1}{2}\epsilon$$

with corresponding eigenvectors

$$\boldsymbol{e}_1 \to \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad \boldsymbol{e}_2 \to \begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

Note: In the limit A is symmetric, so the limiting eigenvectors *are* orthogonal

We may also take the Singular Value Decomposition (SVD) of A;
In Matlab : `[U,W,V] = svd(A);`

We may also take the Singular Value Decomposition (SVD) of A;
In Matlab : `[U,W,V] = svd(A);`
It turns out that, we can write the SVD explicitly

$$\boldsymbol{u}_1 = \frac{1}{L_+}\begin{pmatrix}\epsilon + \sqrt{1+\epsilon^2}\\ 1\end{pmatrix} \quad , \quad \boldsymbol{u}_2 = \frac{1}{L_-}\begin{pmatrix}\epsilon - \sqrt{1+\epsilon^2}\\ 1\end{pmatrix}$$

$$w_1 = \left(2+\epsilon^2+2\sqrt{1+\epsilon^2}\right)^{1/2} \quad , \quad w_2 = \left(2+\epsilon^2-2\sqrt{1+\epsilon^2}\right)^{1/2}$$

$$\boldsymbol{v}_1 = \frac{1}{L_-}\begin{pmatrix}\sqrt{1+\epsilon^2}-\epsilon\\ 1\end{pmatrix} \quad , \quad \boldsymbol{v}_2 = \frac{1}{L_+}\begin{pmatrix}-\sqrt{1+\epsilon^2}-\epsilon\\ 1\end{pmatrix}$$

where $L_+ = \left(1+(\epsilon+\sqrt{1+\epsilon^2})^2\right)^{1/2}$, and $L_- = \left(1+(\epsilon-\sqrt{1+\epsilon^2})^2\right)^{1/2}$ are normalisations ;

We may also take the Singular Value Decomposition (SVD) of A;
In Matlab : `[U,W,V] = svd(A);`
It turns out that, we can write the SVD explicitly

$$\boldsymbol{u}_1 = \frac{1}{L_+}\begin{pmatrix}\epsilon+\sqrt{1+\epsilon^2}\\ 1\end{pmatrix} \quad , \quad \boldsymbol{u}_2 = \frac{1}{L_-}\begin{pmatrix}\epsilon-\sqrt{1+\epsilon^2}\\ 1\end{pmatrix}$$

$$w_1 = \left(2+\epsilon^2+2\sqrt{1+\epsilon^2}\right)^{1/2} \quad , \quad w_2 = \left(2+\epsilon^2-2\sqrt{1+\epsilon^2}\right)^{1/2}$$

$$\boldsymbol{v}_1 = \frac{1}{L_-}\begin{pmatrix}\sqrt{1+\epsilon^2}-\epsilon\\ 1\end{pmatrix} \quad , \quad \boldsymbol{v}_2 = \frac{1}{L_+}\begin{pmatrix}-\sqrt{1+\epsilon^2}-\epsilon\\ 1\end{pmatrix}$$

where $L_+ = \left(1+(\epsilon+\sqrt{1+\epsilon^2})^2\right)^{1/2}$, and $L_- = \left(1+(\epsilon-\sqrt{1+\epsilon^2})^2\right)^{1/2}$ are normalisations ;
The SVD provides an alternative way to express the matrix inverse, and works also for non square matrices.

The SVD of any $N \times M$ matrix A can be written as a product

$$\underbrace{\mathrm{A}}_{N\times M} = \underbrace{\mathrm{U}}_{N\times N}\underbrace{\mathrm{W}}_{N\times M}\underbrace{\mathrm{V}^{\mathrm{T}}}_{M\times M} \tag{2.13}$$

The columns of U are orthonormal, as are the columns of V. I.e.

$$\mathrm{U} = [\boldsymbol{u}_1\ \boldsymbol{u}_1\ \ldots\ \boldsymbol{u}_N] = \begin{bmatrix} \begin{pmatrix} u_{1,1} \\ u_{1,2} \\ \vdots \\ u_{1,N} \end{pmatrix} & \begin{pmatrix} u_{2,1} \\ u_{2,2} \\ \vdots \\ u_{2,N} \end{pmatrix} & \cdots & \begin{pmatrix} u_{N,1} \\ u_{N,2} \\ \vdots \\ u_{N,N} \end{pmatrix} \end{bmatrix}$$

and

$$\mathrm{V} = [\boldsymbol{v}_1\ \boldsymbol{v}_1\ \ldots\ \boldsymbol{v}_M] = \begin{bmatrix} \begin{pmatrix} v_{1,1} \\ v_{1,2} \\ \vdots \\ v_{1,M} \end{pmatrix} & \begin{pmatrix} v_{2,1} \\ v_{2,2} \\ \vdots \\ v_{2,M} \end{pmatrix} & \cdots & \begin{pmatrix} v_{M,1} \\ v_{M,2} \\ \vdots \\ v_{M,M} \end{pmatrix} \end{bmatrix}$$

Then we have

$$\boldsymbol{u}_i \cdot \boldsymbol{u}_j = \delta_{ij} \quad i,j \in [1 \ldots N], \qquad \boldsymbol{v}_i \cdot \boldsymbol{v}_j = \delta_{ij} \quad i,j \in [1 \ldots M].$$

The matrix W has a structure depending on the relative values of N, M.

- If $N = M$, W is square, with diagonal elements $W_{i,i} = w_i, i = 1 \ldots N$
- If $N > M$, W has a $M \times M$ diagonal matrix above $N - M$ rows of zero :

$$\mathrm{W} = \begin{pmatrix} \mathrm{diag}\,(\boldsymbol{w}) \\ 0 \end{pmatrix} = \begin{pmatrix} w_1 & 0 & 0 & \ldots & 0 \\ 0 & w_2 & 0 & \ldots & 0 \\ 0 & 0 & w_3 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & w_M \\ 0 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \ldots & 0 \end{pmatrix}$$

- If $N < M$, W has a $N \times N$ diagonal matrix to the left of $M - N$ columns of zero :

$$\mathrm{W} = \begin{pmatrix} \mathrm{diag}\,(\boldsymbol{w}) & 0 \end{pmatrix} = \begin{pmatrix} w_1 & 0 & 0 & \dots & 0 & 0 \dots & 0 \\ 0 & w_2 & 0 & \dots & 0 & 0 \dots & 0 \\ 0 & 0 & w_3 & \dots & 0 & 0 \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \\ 0 & 0 & 0 & \dots & w_N & 0 \dots & 0 \end{pmatrix}$$

The number of non-zeros in the diagonal part of W is equal to the *rank* of the matrix A i.e. the number of linearly independent columns (or rows) of the matrix

$$R = \mathrm{rank}(\mathrm{A}) \leq min(N, M)$$

For convenience, assume that the linearly independent columns of U, V are arranged as the first R components, and further that the indexing of columns is such that $|w_i| > |w_{i+1}|$. The elements of the diagonal part of W are termed the *spectrum* of A.
The R column-vectors of V corresponding to the non-zero elements of the diagonal of W form an orthonormal basis spanning a R-dimensional vector space called the *row-space*, and the R column-vectors of U corresponding to these elements form an orthonormal basis spanning a R-dimensional vector space called the *range* range(A) or *column-space*. These basis functions are related through

$$\mathrm{A}\boldsymbol{v}_i = w_i \boldsymbol{u}_i\,, \quad \mathrm{A}^{\mathrm{T}}\boldsymbol{u}_i = w_i \boldsymbol{v}_i \qquad i = 1 \ldots R \tag{2.14}$$

The remaining $M - R$ column vectors of V (if greater than zero) form an orthonormal basis spanning a $M - R$-dimensional vector space called the *null-space* Null(A), or *kernel* kern(A). For any vector $\boldsymbol{x}_\perp \in$ Null(A) we have

$$\mathrm{A}\boldsymbol{x}_\perp = \boldsymbol{0} \tag{2.15}$$

Similarly, the $N - R$ column vectors of U (if greater than zero) form an orthonormal basis spanning a $N - R$-dimensional vector space called the *range complement* range$_\perp$(A) or *co-kernel*. For any vector $\boldsymbol{b}_\perp \in$ range$_\perp$(A) we have that the linear equation

$$\mathrm{A}\boldsymbol{x} = \boldsymbol{b}_\perp \tag{2.16}$$

has no solutions.

For the simple case $N = M = R$ the matrix A is square and full-rank and its inverse can be expressed straightforwardly

$$\mathrm{A}^{-1} = \mathrm{V}\mathrm{W}^{-1}\mathrm{U}^{\mathrm{T}} \tag{2.17}$$

where W^{-1} is diagonal with components $1/w_i, i = 1 \ldots R$. This representation allows to express the solution of a particular problem

$$\mathrm{A}\,\boldsymbol{x} = \boldsymbol{b} \rightarrow \boldsymbol{x} = \sum_{i=1}^{R} \frac{\boldsymbol{u}_i \cdot \boldsymbol{b}}{w_i} \boldsymbol{v}_i \tag{2.18}$$

Now we can see a problem that is related to the third definition of an ill-posed problem given above : even if A is full rank, if the relative magnitudes of the w_i decrease rapidly, then the correponding components of the inverse increase at the same rate. In particular, if the decay of the spectrum is bounded by a geometric ratio then the inverse problem is termed *exponentially ill-posed*.

The inverse matrix built from the row-space of A is called the *Moore-Penrose pseudo-inverse* of A.

$$\mathrm{A}^{\dagger} = \mathrm{V}\mathrm{W}^{\dagger}\mathrm{U}^{\mathrm{T}} = \sum_{i=1}^{R} \frac{\boldsymbol{v}_i \boldsymbol{u}_i^{\mathrm{T}}}{w_i} \tag{2.19}$$

Here $\mathrm{W}^{\dagger}$ is a $M \times N$ matrix with the complementary structure to W. To be specific :

- If $N = M = R$, $\mathrm{W}^{\dagger}$ is square, with diagonal elements $W^{\dagger}_{i,i} = 1/w_i, i = 1 \ldots N$
- If $N > M$, $\mathrm{W}^{\dagger}$ has a $M \times M$ diagonal matrix to the left of $N - M$ columns of zero :

$$\mathrm{W}^{\dagger} = \begin{pmatrix} \mathrm{diag}\,(\mathbf{1}/\boldsymbol{w}) & 0 \end{pmatrix} = \begin{pmatrix} 1/w_1 & 0 & 0 & \ldots & 0 & 0 \ldots & 0 \\ 0 & 1/w_2 & 0 & \ldots & 0 & 0 \ldots & 0 \\ 0 & 0 & 1/w_3 & \ldots & 0 & 0 \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & & \\ 0 & 0 & 0 & \ldots & 1/w_M & 0 \ldots & 0 \end{pmatrix}$$

- If $N < M$, $\mathrm{W}^{\dagger}$ has a $N \times N$ diagonal matrix above $M - N$ rows of zero :

$$\mathrm{W}^{\dagger} = \begin{pmatrix} \mathrm{diag}\,(\mathbf{1}/\boldsymbol{w}) \\ 0 \end{pmatrix} = \begin{pmatrix} 1/w_1 & 0 & 0 & \dots & 0 \\ 0 & 1/w_2 & 0 & \dots & 0 \\ 0 & 0 & 1/w_3 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1/w_N \\ 0 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 0 \end{pmatrix}$$

Exercise

Write down the stuctures of the matrices $\mathrm{W}\mathrm{W}^{\dagger}$ *and* $\mathrm{W}^{\dagger}\mathrm{W}$ *for the cases* $N > M$ *and* $M < N$ *and* $M = N$. *Assume for the sake of illustration that the rank* $R < min(M, N)$, *i.e. that there is a non-trivial null space and range complement space even in the case where* $M = N$.

The pseudo-inverse can always be constructed regardless if A is non-square and/or rank-deficient. Thus any linear matrix problem has a pseudo-solution

$$\mathrm{A}\,\boldsymbol{x} = \boldsymbol{b} \rightarrow \boldsymbol{x}^{\dagger} = \mathrm{A}^{\dagger}\boldsymbol{b} \tag{2.20}$$

This solution has the property that it is the *least-squares, minimum norm* solution in the sense that

$$|\boldsymbol{b} - \mathrm{A}\,\boldsymbol{x}^{\dagger}|^2 \rightarrow min\,, |\boldsymbol{x}^{\dagger}|^2 \rightarrow min$$

first we note :

$$\begin{aligned} \boldsymbol{b} - \mathrm{A}\boldsymbol{x}^{\dagger} &= \left[\mathrm{I} - \mathrm{A}\mathrm{A}^{\dagger}\right]\boldsymbol{b} \\ &= \mathrm{U}\left[\mathrm{I} - \mathrm{W}\mathrm{W}^{\dagger}\right]\mathrm{U}^{\mathrm{T}}\boldsymbol{b} \\ &= \mathrm{P}_{\perp}\boldsymbol{b} \end{aligned}$$

where

$$\mathrm{P}_{\perp} = \sum_{i=R+1}^{N} \boldsymbol{u}_i \boldsymbol{u}_i^{\mathrm{T}} \tag{2.21}$$

is a matrix that projects data space vectors into the orthogonal complement of the range. Now consider another solution space vector $\tilde{\boldsymbol{x}} = \boldsymbol{x}^{\dagger} + \boldsymbol{x}'$, with $\boldsymbol{b}' = \mathrm{A}\boldsymbol{x}'$ the corresponding data space vector, which by definition lies in the range of A. We have

$$\begin{aligned} \left|\boldsymbol{b} - \mathrm{A}\tilde{\boldsymbol{x}}\right|^2 &= \left|\boldsymbol{b} - \mathrm{A}\boldsymbol{x}^{\dagger} - \mathrm{A}\boldsymbol{x}'\right|^2 \\ &= \left|\mathrm{P}_{\perp}\boldsymbol{b} - \mathrm{U}\mathrm{U}^{\mathrm{T}}\boldsymbol{b}'\right|^2 \\ &= \sum_{i=R+1}^{N} \left(\boldsymbol{u}_i \cdot \boldsymbol{b}\right)^2 + \sum_{i=1}^{R} \left(\boldsymbol{u}_i \cdot \boldsymbol{b}'\right)^2 \end{aligned}$$

where the second sum on the right holds due to the fact that $\boldsymbol{b}' \in$ Range(A). Thus any vector $\boldsymbol{x}'$ with a non-zero component in the row space of A will increase the data space norm.
We now need to see if any vector $\boldsymbol{x}' \in$ Null(A) could give a smaller norm $|\tilde{x}|$. We write

$$\begin{aligned}
|\tilde{\boldsymbol{x}}|^2 &= \left|\boldsymbol{x}^\dagger + \boldsymbol{x}'\right|^2 \\
&= \left|\mathrm{V}\mathrm{W}^\dagger\mathrm{U}^{\mathrm{T}}\boldsymbol{b} + \mathrm{V}\mathrm{V}^{\mathrm{T}}\boldsymbol{x}'\right|^2 \\
&= \left|\mathrm{V}\left[\mathrm{W}^\dagger\mathrm{U}^{\mathrm{T}}\boldsymbol{b} + \mathrm{V}^{\mathrm{T}}\boldsymbol{x}'\right]\right|^2 \\
&= \left|\left[\mathrm{W}^\dagger\mathrm{U}^{\mathrm{T}}\boldsymbol{b} + \mathrm{V}^{\mathrm{T}}\boldsymbol{x}'\right]\right|^2 \\
&= \sum_{i=1}^{R}\left(\frac{\boldsymbol{u}_i\cdot\boldsymbol{b}}{w_i}\right)^2 + \sum_{i=R+1}^{M}\left(\boldsymbol{v}_i\cdot\boldsymbol{x}'\right)^2
\end{aligned}$$

where the second sum on the right holds due to the fact that $\boldsymbol{x}' \in$ Null(A). Thus any vector $\boldsymbol{x}'$ with a non-zero component in the null-space of A will increase the solution space norm.

Problem 1 revisited

$$A = \begin{pmatrix} 1 & 1 \end{pmatrix} \quad \Rightarrow \quad A^{\dagger} = \frac{1}{2}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

$$\text{SVD:} \qquad (1)\begin{pmatrix} \sqrt{2} & 0 \end{pmatrix}\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$\Rightarrow \quad \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = A^{\dagger}(2) \quad = \quad \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The row-space is spanned by vector $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and the null space is spanned by vector $\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

Problem 2 revisited

$$A = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \quad \Rightarrow \quad A^{\dagger} = \frac{1}{2}\begin{pmatrix} 1 & 1 \end{pmatrix}$$

$$\text{SVD:} \quad \begin{pmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} \sqrt{2} \\ 0 \end{pmatrix} (1)$$

$$\Rightarrow \quad x_1 = A^{\dagger}\begin{pmatrix} 1 \\ 3 \end{pmatrix} = 2$$

The range is spanned by vector $\frac{1}{\sqrt{2}}\begin{pmatrix} 1 \\ 1 \end{pmatrix}$, and the complement of the range is spanned by vector $\frac{1}{\sqrt{2}}\begin{pmatrix} -1 \\ 1 \end{pmatrix}$.

Problem 3 revisited

The matrix inverse was given in eq. 2.12 so we do not need to write it again. However, by considering it as

$$\mathrm{A}^{\dagger}(\boldsymbol{b}_0 + \boldsymbol{\eta}) \quad = \quad \mathrm{A}^{\dagger}\boldsymbol{b}_0 + \frac{\boldsymbol{\eta}\cdot\boldsymbol{u}_1}{w_1}\boldsymbol{v}_1 + \frac{\boldsymbol{\eta}\cdot\boldsymbol{u}_2}{w_2}\boldsymbol{v}_2$$

We see that the solution can be considered as a random vector with mean the noise free vector $\mathrm{A}^{\dagger}\boldsymbol{b}_0$ and covariance given by

$$\mathrm{C} = \frac{\boldsymbol{v}_1\boldsymbol{v}_1^{\mathrm{T}}}{w_1^2} + \frac{\boldsymbol{v}_2\boldsymbol{v}_2^{\mathrm{T}}}{w_2^2} = \mathrm{V}\mathrm{W}^{-2}\mathrm{V}^{\mathrm{T}}$$

Most of what we have discussed for matrices applies also to Operators. Let us restrict the discussion to linear integral operators $A : X \mapsto Y$

$$b = Af \quad \equiv \quad b(y) = \int_{-\infty}^{\infty} K(y,x) f(x) \mathrm{d}x \tag{2.22}$$

We define the *adjoint* operator $A^* : Y \mapsto X$ (the equivalent of matrix transpose) by integration of the (complex conjugate of the) kernel function $K(x,y)$ with the other variable :

$$h = A^* g \quad \equiv \quad h(x) = \int_{-\infty}^{\infty} \overline{K}(y,x) g(y) \mathrm{d}y \tag{2.23}$$

and we also need to attach a meaning to the inner product :

$$\langle h, f \rangle_X := \int_{-\infty}^{\infty} \overline{h}(x) f(x) \mathrm{d}x \ , \quad \langle g, b \rangle_Y := \int_{-\infty}^{\infty} \overline{g}(y) b(y) \mathrm{d}y \tag{2.24}$$

Note that the inner product is technically different for the functions in the data space and the solution space, because they might in general have different variables.

The SVD of operator A is then defined as the set of *singular functions* $\{v_k(x), u_k(y)\}$ and singular values $\{w_k\}$ such that

$$Av_k = w_k u_k \,; \quad A^* u_k = w_k v_k$$

and with

$$\langle u_j, u_k \rangle_Y = \delta_{jk} \,, \quad \langle v_j, v_k \rangle_X = \delta_{jk} \,.$$

One major difference is that the set of such functions is (usually) *infinite*. There is often no equivalence to the kernel (Null space) of the operator, although sometimes there is.

Note also the mappings

$A^*A : X \mapsto X$ given by

$$h = A^*Af \equiv h(x) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \overline{K}(y,x)K(y,x')f(x')\mathrm{d}x'\mathrm{d}y$$

$$\equiv h(x) = \int_{-\infty}^{\infty} B(x,x')f(x')\mathrm{d}x'$$

$$\text{where} \quad B(x,x') = \int_{-\infty}^{\infty} \overline{K}(y,x)K(y,x')\mathrm{d}y$$

$AA^* : Y \mapsto Y$ given by

$$b = AA^*g \equiv b(y) = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \overline{K}(y',x)K(y,x)g(y')\mathrm{d}x\mathrm{d}y$$

$$\equiv b(y) = \int_{-\infty}^{\infty} C(y,y')g(y')\mathrm{d}y'$$

$$\text{where} \quad C(y,y') = \int_{-\infty}^{\infty} \overline{K}(y',x)K(y,x)\mathrm{d}x$$

Example: convolution as a linear operator with stationary kernel. By *stationary* we mean that the kernel depends only on the difference $(x - y)$, in other words $K(x, y)$ is a function of only one variable $K(y - x) \equiv K(t)$

$$b = Af := K * f \quad \equiv \quad b(y) = \int_{-\infty}^{\infty} K(y - x) f(x) \mathrm{d}x \tag{2.25}$$

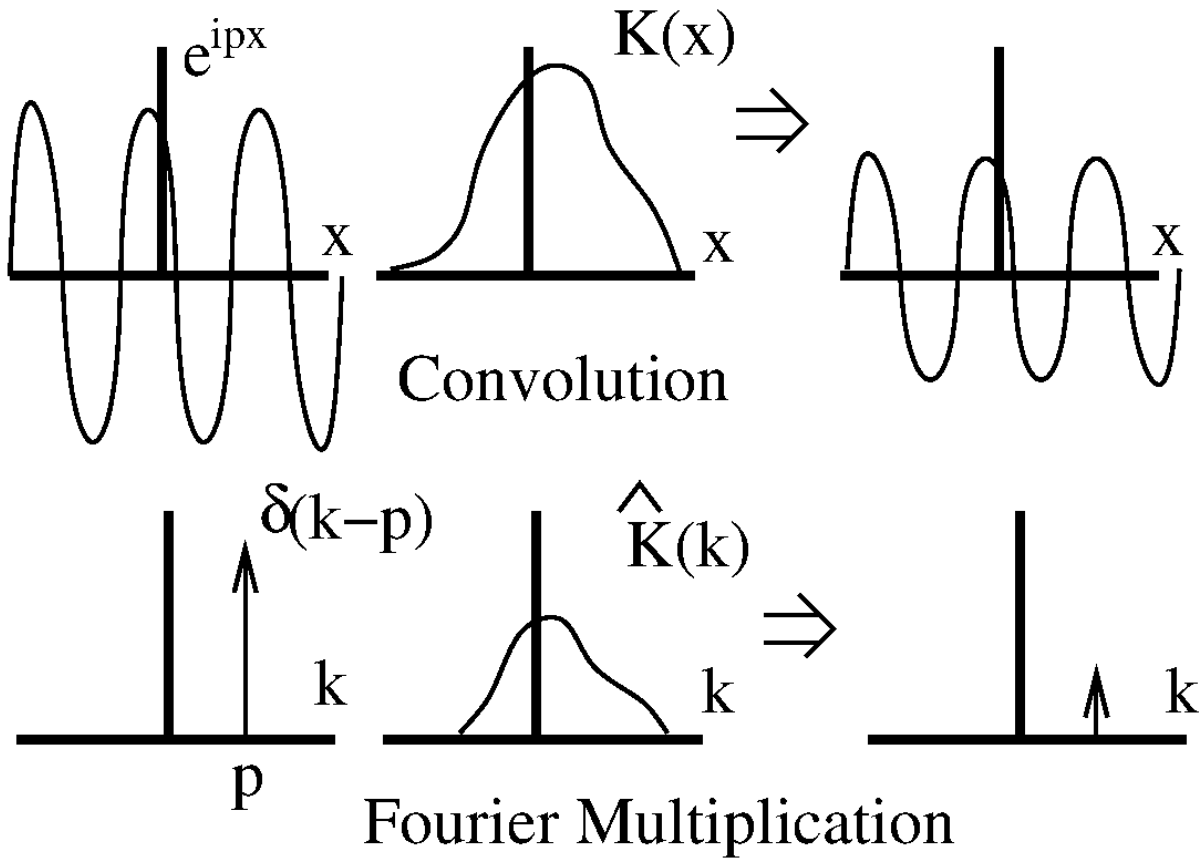
An important property of convolution is the *Convolution Theorem* which relates the functions under convolution, and the result, to their Fourier Transforms. Assuming that

$$\hat{b}(k) = \mathcal{F}_{x \to k} b(x), \quad \hat{K}(k) = \mathcal{F}_{x \to k} K(x), \quad \hat{f}(k) = \mathcal{F}_{x \to k} f(x)$$

this convolution theorm states that

$$b = K * f \quad \Rightarrow \quad \hat{b} = \hat{K}\hat{f} \tag{2.26}$$

Convolution of a trigonometric function of frequency p with any function K results in the same frequency trigonometric function, scaled by amplitude $\hat{K}(p)$



It follows from the convolution theorm in Fourier theory that the singular vectors of A are the functions $\{\cos px, \sin px\}$ with singular values $\hat{K}(p)$ where $\hat{K}$ is the Fourier Transform of K

$$A \begin{pmatrix} \cos px \\ \sin px \end{pmatrix} = \hat{K}(p) \begin{pmatrix} \cos py \\ \sin py \end{pmatrix} \quad (2.27)$$

$$A^* \begin{pmatrix} \cos py \\ \sin py \end{pmatrix} = \hat{K}(p) \begin{pmatrix} \cos px \\ \sin px \end{pmatrix} \quad (2.28)$$

Exercise

Verify that $\sin px$ *and* $\cos px$ *are eigenvectors (and therefore also singular vectors) of the linear convolution operator with a Gaussian* $K(y-x) = \exp\left[-\frac{(y-x)^2}{2\sigma^2}\right]$ *(for any non zero value of* σ*) and that singular values are* $\lambda_p = \exp\left[-\frac{1}{2}\sigma^2 p^2\right]$.

We also see exactly how to do the inversion - applying the Moore-Penrose inverse in the continuous case is given by

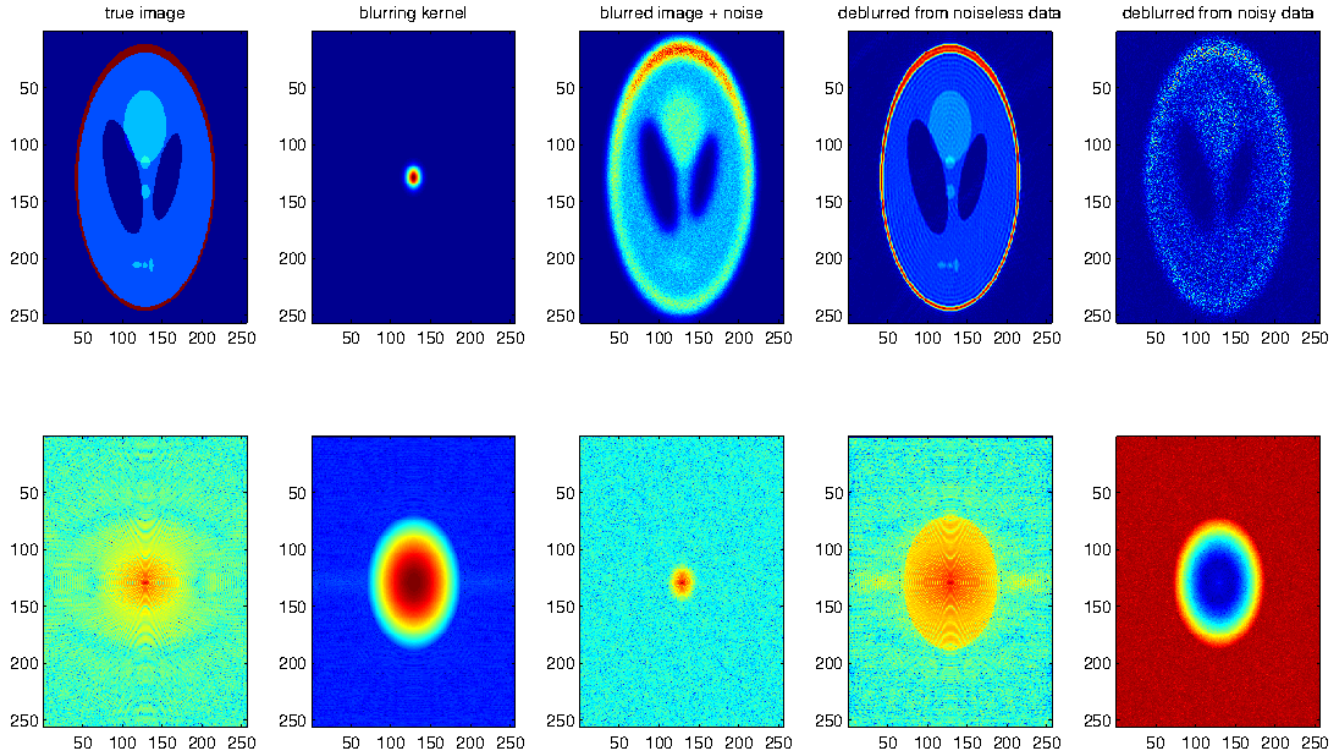
$$f^{\text{recon}}(x) = \sum_{i=1}^{\infty} v_i(x) \frac{\langle b(x), u_i(x) \rangle}{w_i} \quad \equiv \quad \mathcal{F}_{k\to x}^{-1} \left[\frac{\mathcal{F}_{x\to k} b(x)}{\mathcal{F}_{x\to k} K(x)} \right]$$

This also makes clear how the noise is propagated - the covariance of noise in the reconstruction is diagonalised by its Fourier components, and the higher spatial frequencies are amplified by the reciprocal of the square of the corresponding frequency of the convolution kernel. An example is shown in the next slide. The convolution kernel is a Gaussian so the noise is propogated with the exponential of the square of the frequency number.

Basic Concepts

Example Singular Value Decomposition for Operators

Convolution and Deconvolution in the Fourier Domain. Top row is spatial domain and bottom row is the Fourier Transform of the top row, on a logarithmic scale.



Lastly, we may consider the case where the data are a finite set of samples, but the reconstruction space consists of functions. The forward mapping is

$$\boldsymbol{b} = Af \quad \equiv \quad \boldsymbol{b} = \int_{-\infty}^{\infty} \boldsymbol{K}(x) f(x) \mathrm{d}x$$

where $\boldsymbol{K}(x)$ is a vector valued function. The Adjoint becomes a sum of functions

$$h = A^* \boldsymbol{g} \quad \equiv \quad h(x) = \boldsymbol{g}^{\mathrm{T}} \boldsymbol{K}(x) = \sum_{i=1}^{N} g_i \overline{K}_i(x)$$

We also see that $A^*A : X \mapsto X$ is an operator

$$h = A^*Af \quad \equiv h(x) = \int_{-\infty}^{\infty} \underbrace{\boldsymbol{K}^{\mathrm{T}}(x)\boldsymbol{K}(x')}_{B(x,x')} f(x')\mathrm{d}x' = \int_{-\infty}^{\infty} \underbrace{\left(\sum_{i=1}^{N} \overline{K}_i(x)K_i(x')\right)}_{B(x,x')} f(x')\mathrm{d}x'$$

and that $AA^* : Y \mapsto Y$ is a $N \times N$ matrix

$$\boldsymbol{b} = AA^*\boldsymbol{g} \quad \equiv \boldsymbol{b} = \underbrace{\left(\int_{-\infty}^{\infty} \boldsymbol{K}(x)\boldsymbol{K}^{\mathrm{T}}(x)\mathrm{d}x\right)}_{\mathrm{C}} \boldsymbol{g} = \sum_{j=1}^{N} \underbrace{\left(\int_{-\infty}^{\infty} K_i(x)K_j^{\mathrm{T}}(x)\mathrm{d}x\right)}_{C_{ij}} g_j$$

In this case the number of left singular vectors $\{\boldsymbol{u}_i; i = 1 \ldots N\}$ is finite, and the number of right singular functions $\{v_i(x)\}$ is (usually) infinite, but those greater than $i = N$ will be in the null-space of the forward operator. This is the *usual* case for real measurements : there exists an infinite dimensional space Null-Space, "invisible" to the measurements!