1 Polynomial curve fitting

Consider a dataset given by n pairs of real numbers

$$(t_1, y_1), (t_2, y_2), \dots, (t_n, y_n),$$

which we interpret as n points in $\mathbb{R}^2$. We would like to find a polynomial

$$p(t) = q_0 + q_1 t + \cdots + q_{n-1} t^{n-1},$$

of degree $n-1$, which passes through these points. Thus we would like to solve the system of n equations

$$\begin{aligned} p(t_1) &= y_1, \\ p(t_2) &= y_2, \\ &\vdots \\ p(t_n) &= y_n, \end{aligned} \tag{1}$$

for the unknown polynomial coefficients $q = (q_0, q_1, \dots, q_{n-1}) \in \mathbb{R}^n$.

Question 1.

1.1. Write down the total matrix corresponding to the system of equations (1).

1.2. Use Gaussian elimination to find the quadratic polynomial passing through the points $(1, 4)$, $(2, 0)$, and $(3, 12)$. Plot the graph of the resulting polynomial and the points in the dataset.

1.3. Use Gaussian elimination to find the fourth degree polynomial passing through the points $(-2, 3)$, $(-1, 5)$, $(0, 1)$, $(1, 4)$, and $(2, 10)$. Plot the graph of the resulting polynomial and the points in the dataset.

2 Linear transformations and digital image processing

Digital images as vectors in $\mathbb{R}^n$

In this workshop we will apply some of the linear algebraic concepts we learned so far towards image processing. For simplicity, we will focus on grayscale images only. We will think of an $N \times M$ image P as a two-dimensional array containing $n = NM$ real numbers representing the grayscale levels of individual pixels.[1] We can now concatenate the pixels of the image, row-by-row, in order to obtain a one-dimensional array $v = \mathrm{vec}(P) \in \mathbb{R}^n$, see the figure below:

$$2 \times 3 \text{ image:} \quad P = \begin{array}{|c|c|c|} \hline P_{1,1} & P_{1,2} & P_{1,3} \\ \hline P_{2,1} & P_{2,2} & P_{2,3} \\ \hline \end{array} \quad \Longleftrightarrow \quad \text{vector } \mathrm{vec}(P) = \begin{bmatrix} P_{1,1} \\ P_{1,2} \\ P_{1,3} \\ P_{2,1} \\ P_{2,2} \\ P_{2,3} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ v_6 \end{bmatrix} = v \in \mathbb{R}^6$$

In this way the grayscale value of pixel $P_{y,x}$, $y = 1, \dots, N$, $x = 1, \dots, M$ corresponds to the value $v_{M(y-1)+x}$ component in the equivalent vector representation $v = \mathrm{vec}(P)$. We will use double index notation to refer to the pixels in the image, and single index notation to refer to the components in the vector.

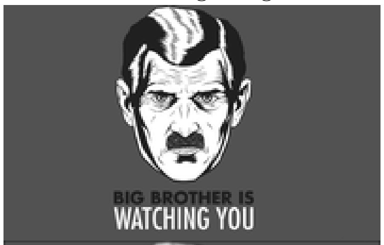
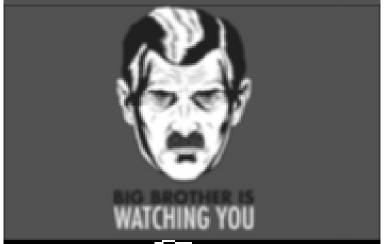
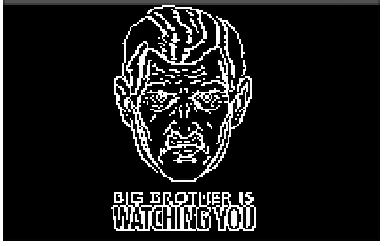
Image convolution with a kernel

Let ω be a small $(2k+1) \times (2k+1)$ matrix; often $k = 1$ or $k = 2$ is used, which results in 3×3 or 5×5 matrices. With the help of this matrix, we will now define a function (transformation) T, which maps $N \times M$ images to $N \times M$ images. Formally, we say that $\mathrm{vec}(P) = T_\omega(\mathrm{vec}(O))$, when

$$P_{y,x} = \sum_{\zeta=\max\{y-k,1\}}^{\min\{y+k,N\}} \sum_{\xi=\max\{x-k,1\}}^{\min\{x+k,M\}} \omega_{y-\zeta+k+1,x-\xi+k+1} O_{\zeta,\xi} \tag{2}$$

In other words, pixels of the image $\mathrm{vec}(P) = T_\omega(\mathrm{vec}(O))$ are weighted averages of the pixels of the image O over a small $(2k+1) \times (2k+1)$ neighbourhood, with weights specified in the kernel matrix ω. Below we show some examples of the application of various kernels to a small 125×200 pixel image:

[1] That is, we will ignore the fact that digital images allow only a discrete number of grayscale levels.

Operation	Kernel matrix ω	Resulting image
Original image	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Blur	$\frac{1}{256}\begin{bmatrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{bmatrix}$	
Ridge detection	$\frac{1}{64}\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Question 2.

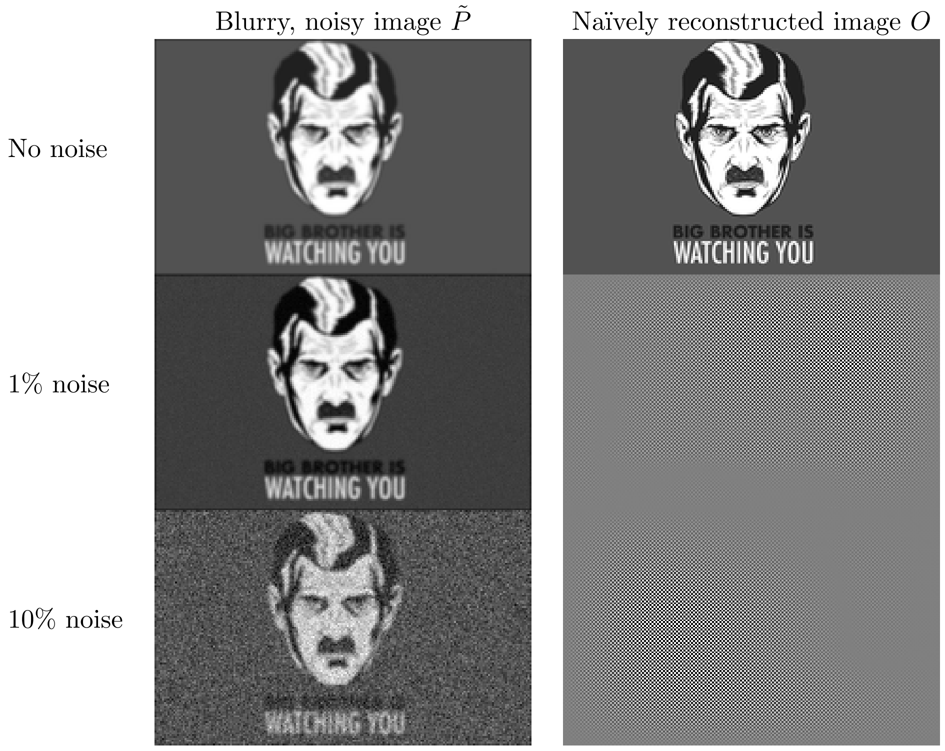
2.1. Verify that given a kernel matrix ω, the resulting transformation $T_\omega : \mathbb{R}^{MN} \to \mathbb{R}^{MN}$ given by (2) is linear. That is, check that it satisfies the definition given in section 1.8, p. 82 in the textbook.

2.2. Consider two indices $i = M(y-1)+x$ and $j = M(\zeta-1)+\xi$, where x, ξ are integers between 1 and M, and y, ζ are integers between 1 and N. Determine the expression for the element A_{ij} of the standard matrix correspoinding to the linear transformation T. Hint: Consider two cases. (i): $|x-\xi| \le k$ and $|y-\zeta| \le k$; (ii): the opposite of (i).

Application to image deblurring

Often blurring occurs as a result of optical and other imperfections. As a result, given a blurry image P that we take and having a blurring model $\text{vec}(P) = T(\text{vec}(O))$, we may be interested in computing the original image O. In this case th answer can be obtained by solving the system of linear algebraic equations $A\,\text{vec}(O) = \text{vec}(P)$, where A is the standard matrix corresponding to the linear transformation T.

Unfortunately, in reality things are somewhat more complicated. Namely, instead of observing the "ideal" blurred image $\text{vec}(P) = A\,\text{vec}(O)$, we are typically presented with a "noisy version" $\text{vec}(\tilde{P}) = A\,\text{vec}(O) + b$, where b is some unknown random measurement noise (for example, "cross-talk" between electronic components in the digital camera). As shown in the table below, naïvely computing the solution by solving the system of linear algebraic equations $A\,\text{vec}(O) = \text{vec}(\tilde{P})$ leads to disasterous results.


Blurry, noisy image $\tilde{P}$
Naïvely reconstructed image O
No noise
1% noise
10% noise
BIG BROTHER IS WATCHING YOU

In order to better understand such a behaviour we will discuss it on a tiny 2×2 system of linear algebraic equations.

Question 3. Consider a 2×2 matrix A and a vector $b \in \mathbb{R}^2$ given by

$$A = \begin{bmatrix} 1 & 1 \\ 1 & 1+\epsilon \end{bmatrix} \qquad \text{and} \qquad b = \begin{bmatrix} 2 \\ 2+\delta \end{bmatrix}, \tag{3}$$

where ϵ and δ are small numbers (close to 0). Note that this setup corresponds to the dataset $(t_1, y_1) = (1, 2)$ and $(t_2, y_2) = (1 + \epsilon, 2 + \delta)$ in question 1.

3.1. Sketch the two vectors corresponding to the two columns of the matrix A in (3) in $\mathbb{R}^2$ for $\epsilon = 0.5$, $\epsilon = 0.01$ and $\epsilon = 0$. Discuss, in what way the columns of A become "almost linearly dependent" for small $\epsilon \approx 0$.

3.2. Use Gaussian elimination to solve the system $Ax = b$ with A, b given in (3), assuming that $\epsilon \neq 0$. Write down the expression for the solution x in terms of the parameters $\epsilon \neq 0$ and δ.

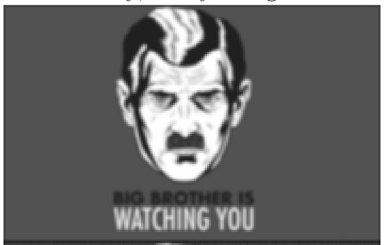
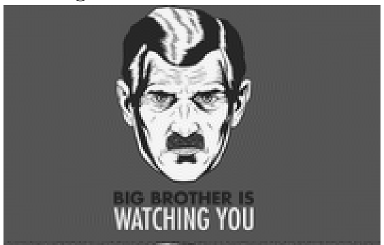
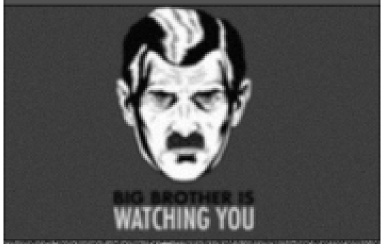
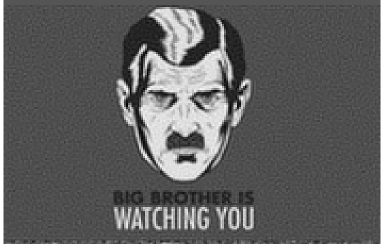
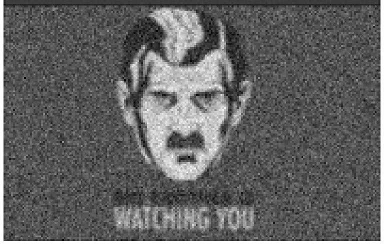
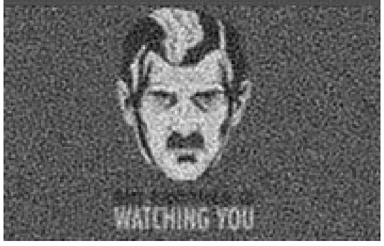
3.3. We will now interpret parameter δ as a random measurement noise. Explain how even very small changes to δ may lead to very large changes in the solution x, provided that $\epsilon \neq 0$ is close to zero.
Hint: you can compare pairs of solutions corresponding to $\delta = 0$ and $\delta = 0.01$ for various values of $\epsilon \neq 0$: $\epsilon = 0.1$, $\epsilon = 0.01$, $\epsilon = 0.0001$.

The phenomenon we have observed and discussed is known as the *ill conditioning*: small changes in problem's data lead to large changes in the solution. One standard approach to dealing with ill-conditioned linear systems is known as the Tikhonov regularization. In this approach, we multiply both sides of the the ill-conditioned system $A\,\mathrm{vec}(O) = \mathrm{vec}(\tilde{P})$ with A^T, and then "improve" the matrix $A^T A$ in the left hand side of the system by adding the identity matrix multiplied by a small regularization parameter $\alpha > 0$. Thus we end up with the following system of linear algebraic equations for computing the deblurred image O:

$$[\alpha I + A^T A]\,\mathrm{vec}(O) = A^T\,\mathrm{vec}(\tilde{P}),$$

where $\alpha > 0$ is a regularization parameter, and I is the identity matrix.[2] The results of deblurring based on Tikhonov-regularization are shown below.

[2]In block 3 of this course we will return to linear algebraic systems of this type.

	α	Blurry, noisy image $\tilde{P}$	Tikhonov regularized reconstructed image O
No noise	10^{-6}		
1% noise	10^{-2}		
10% noise	0.3		

Two comments are in order.

- Significanly better image deblurring algorithms exist. However, those go far beyond linear algebraic ideas, which we focus on in this course.
- Note that even for the small 125×200 image we consider in this workshop, the size of the algebraic system that we have to solve to reconstruct the image is $n \times n = 25000 \times 25000$. For a modestly sized image with 768×1024 pixels the algebraic system becomes 786432×786432 in size, and at 4K UHDTV resolution of 2160×3840 we need to solve the system of the size 8294400×8294400. Note that even to store all elements of the matrix (as 8-byte floating point numbers) for such a system would require $8294400^2 \cdot 8\text{B} \approx 500GB$ of memory.

  Luckily, most elements of the matrix defining the system are zeros, and as such do not have to be stored. For solving such systems Gaussian elimination based algorithms quickly become inefficient or even infeasible. In the last part of this workshop we will discuss some simple alternative algorithms that may be used in place of Gaussian elimination for approximately solving systems of linear algebraic equations.

3 Iterative algorithms for solving linear algebraic systems

Whenever Gaussian elimination for solving linear algebraic systems is prohibitevely expensive, or simply inefficient, one can look at alternative approaches. Unlike Gaussian elimination, which arrives at a solution to a linear system after performing a finite number of algorithmic steps, iterative algorithms construct an infinite sequence of vectors $x^{(1)}, x^{(2)}, x^{(3)}, \ldots$, which progressively become better and better approximations to solution to the system of equations.

One of the simplest iterative algorithms for solving linear algebraic systems is the Gauss–Seidel method. Consider the matrix equation $Ax = b$, where

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}, \qquad x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}, \quad \text{and} \quad b = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}.$$

We now split the matrix A into its lower triangle (including the diagonal) L and the upper triangle U:

$$L = \begin{bmatrix} a_{1,1} & 0 & \ldots & 0 \\ a_{2,1} & a_{2,2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,n} \end{bmatrix}, \qquad \text{and} \qquad U = A - L = \begin{bmatrix} 0 & a_{1,2} & \cdots & a_{1,n} \\ 0 & 0 & \cdots & a_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{bmatrix},$$

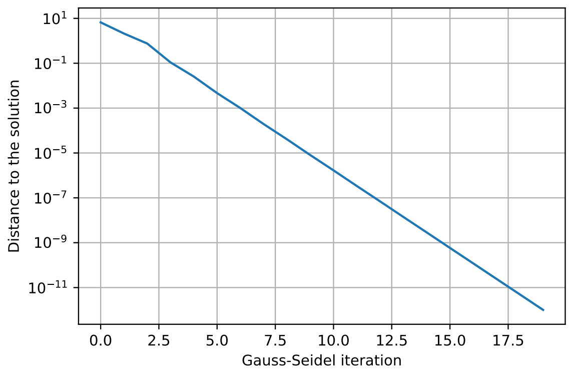
and assume that all diagonal elements are non-zero: $a_{i,i} \neq 0$, $i = 1, \dots, n$. The algorithm is defined as follows.

```
1: Select some initial guess $x^{(0)} \in \mathbb{R}^n$
2: for $k = 0, 1, 2 \dots$ do
3:     Solve the system $Lx^{(k+1)} = b - Ux^{(k)}$ to determine $x^{(k+1)}$
4:     if $x^{(k+1)} \approx x^{(k)}$ (up to some desired tolerance) then
5:         Stop, $x^{(k+1)}$ is an approximate solution to the system $Ax = b$
6:     end if
7: end for
```

We will now discuss a few basic details of this algorithm. As a specific example, we will use a small 4×4 system defined by

$$A = \begin{bmatrix} -12 & 4 & 0 & -6 \\ 6 & 14 & 3 & -3 \\ -5 & -8 & 24 & 8 \\ 1 & -4 & 10 & 16 \end{bmatrix} \quad \text{and} \quad b = \begin{bmatrix} -8 \\ 47 \\ -93 \\ -13 \end{bmatrix} \tag{4}$$

The behaviour of Gauss–Seidel algorithm for this (very small) system, starting from the initial guess $x^{(0)} = [1, 2, 3, 4]^T$, is illustrated in the figure below.



Question 4.

4.1. Write down the matrices L and U corresponding to the system (4).

4.2. Let $x^{(0)} = [1, 2, 3, 4]^T$. Compute the next iterate $x^{(1)}$ in the Gauss–Seidel algorithm applied to the system (4).

Note: in order to solve the system of linear algebraic equations on line 3 of the algorithm, one does not need to carry out Gaussian elimination. Note that one can compute $x_1^{(1)}$ from the first equation in the system; with this knowledge, $x_2^{(1)}$ can be computed using the second equation in the system, etc.

4.3. Let now $x^{(0)} = [1, 4, -3, 2]^T$. Show that the next iterate $x^{(1)}$ in the Gauss–Seidel algorithm applied to the system (4) equals $x^{(0)}$. Does $x^{(0)}$ solve the system (4)?

We will now generalize the observations in points 4.3 and 4.2.

4.4. Assume that at some iteration k of the Gauss–Seidel algorithm we have that $x^{(k+1)} = x^{(k)}$. Show that $x^{(k)}$ solves the system $Ax = b$. This explains the stopping criterion on line 4 of the Gauss–Seidel algorithm.

4.5. Explain why the following *forward substitution* algorithm solves the system of linear algebraic equations $Lx^{(k+1)} = b - Ux^{(k)}$ found on line 3 of the Gauss–Seidel algorithm:

```
1: for $i = 1, 2, \dots, n$ do
2:     $$x_i^{(k+1)} = a_{i,i}^{-1} \left[ b - \sum_{j=1}^{i-1} a_{i,j} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{i,j} x_j^{(k)} \right]$$
3: end for
```