Christian Vollmann *Winter Term 2021/2022*

Elements of Mathematics

Exercise Sheet 10

Submission due date: **18.01.2022, 10:15h**

THEORY

1 Orthogonal Projection

Prove the following statement: Let $V \subset \mathbb{R}^m$ be a linear subspace and $b \in \mathbb{R}^m$. Then

$$\widehat{z} = \arg\min_{z \in V} \|z-b\|_2^2 \quad \Leftrightarrow \quad \widehat{z} - b \in V^\perp := \{w \in \mathbb{R}^n : w^\top v = 0 \ \ \forall v \in V\}.$$

Hint: You can use: For all $x, y \in \mathbb{R}^m$: $\|x+y\|_2^2 = \|x\|_2^2 + \|y\|_2^2 \ \Leftrightarrow \ x^\top y = 0$. *(4 Points)*

Solution:

We use the hint with $x = \widehat{z} - b$ and $y := z - \widehat{z}$ for some $z \in V$ (note that $(z - \widehat{z}) \in V \ \forall z \in V$, since $\widehat{z} \in V$ and V subspace). More precisely, for a $\widehat{z} \in V$ we find

$$\begin{aligned}
\widehat{z} - b \in V^\perp \quad &\Leftrightarrow \forall z \in V : (\widehat{z} - b)^\top z = 0 \\
&\Leftrightarrow \forall z \in V : (\widehat{z} - b)^\top (z - \widehat{z}) = 0 \\
&\Leftrightarrow \forall z \in V : \|z - b\|_2^2 = \|\widehat{z} - b\|_2^2 + \|\widehat{z} - z\|_2^2 \\
&\Leftrightarrow \forall z \in V : \|\widehat{z} - b\|_2^2 \leq \|z - b\|_2^2 \\
&\Leftrightarrow \widehat{z} = \arg\min_{z \in V} \|z - b\|_2^2.
\end{aligned}$$

2 Linear Least Squares

Let $b \in \mathbb{R}^m$ and $A \in \mathbb{R}^{m \times n}$. Assume you are given the least squares problem

$$\min_{x \in \mathbb{R}^n} \|Ax - b\|_2^2.$$

1. Which equation does a solution $\hat{x}$ of the above least squares problem solve? Give formula and name of the equation.

2. Assume you are given the following data

z	-3	-1	0	1	3
y	-3	-1,5	0	2,5	4

.

   Solve the curve fitting problem

$$\min_{c_0, c_1 \in \mathbb{R}} \sum_{i=1}^{5} (c_0 + c_1 z_i - y_i)^2.$$

(6 Points)

Solution:

1. $\hat{x}$ solves the normal equation: $A^TA\hat{x} = A^Ty$
2. In this case: $A = \begin{pmatrix} 1 & -3 \\ 1 & -1 \\ 1 & 0 \\ 1 & 1 \\ 1 & 3 \end{pmatrix}$,

$$A^TA = \begin{pmatrix} 5 & 0 \\ 0 & 20 \end{pmatrix} \;,$$

$$A^Ty = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ -3 & -1 & 0 & 1 & 3 \end{pmatrix} \begin{pmatrix} -3 \\ -1,5 \\ 0 \\ 2,5 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 25 \end{pmatrix}$$

Normal equation:

$$\begin{pmatrix} 5 & 0 \\ 0 & 20 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \end{pmatrix} = \begin{pmatrix} 2 \\ 25 \end{pmatrix} \quad \Leftrightarrow \quad c_0 = \frac{2}{5} = 0,4, \; c_1 = \frac{25}{20} = 1,25$$

$$\Rightarrow \quad \hat{x} = \begin{pmatrix} 0,40 \\ 1,25 \end{pmatrix}$$

3 Linear Least Squares

We are given a sample of size m of measurements $(z_i, y_i) \in \mathbb{R}^2$ for $i = 1, \dots, m$. Determine the minimizer c_0 of the problem

$$\min_{c_0 \in \mathbb{R}} \sum_{i=1}^{m} (c_0 - y_i)^2.$$

Hint: This is the simple case where our assumed model is a constant function, i.e., $f(z_i) \equiv c_0$. Set up the design matrix A and solve the normal equation. *(4 Points)*

Solution:

Model: $f(x) = c_0$, with measurements (z_i, y_i), $i = 1, \dots, m$.

With the design matrix $A = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} \in \mathbb{R}^{m \times 1} = \mathbb{R}^m$ and $b = \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \in \mathbb{R}^m$, we find

$$\min_{c_0 \in \mathbb{R}} \sum_{i=1}^{m} (c_0 - y_i)^2 = \min_{c_0 \in \mathbb{R}} \|A \cdot c_0 - y\|_2^2$$

We have

$$A^TA = \sum_{i=1}^{m} 1 = m \quad \text{and} \quad A^Tb = \sum_{i=1}^{m} y_i.$$

Thus by solving the normal equation we find

$$A^TAc_0 = A^Tb \quad \Leftrightarrow \quad m \cdot c_0 = \sum_{i=1}^{m} y_i \quad \Leftrightarrow \quad c_0 = \frac{1}{m} \sum_{i=1}^{m} y_i.$$

With other words, the best constant fit in the least squares sense is the average of the data.

PROGRAMMING

4 More General Regularization Terms and Image Inpainting

Background:
Based on the idea of Tikhonov regularization we can use more general regularization terms by transforming the vector x with some matrix $G \in \mathbb{R}^{m\times n}$. Specifically, let us consider the more general regularized problem

$$\min_{x\in\mathbb{R}^n} \|Ax-b\|_2^2 + \frac{\delta}{2}\|Gx\|_2^2, \quad \delta > 0 \text{ small.} \tag{1}$$

The corresponding "regularized" normal equation than reads as

$$(A^TA + \delta G^\top G)x = A^Tb. \tag{2}$$

Observe that for $G = I$ we obtain the standard Tikhonov (or L^2-) regularization. We easily see that if $G^\top G$ is positive definite, then so is $A^TA + \delta G^\top G$ for positive δ, so that (**??**) is uniquely solvable.

We will apply this framework to the problem of image inpainting. Therefore assume you are given the deteriorated image b, which is obtained from the unknown original image x through the following masking operation

$$b_i = \begin{cases} x_i & i \in \texttt{indices}, \\ 0 & \text{else}, \end{cases} \tag{3}$$

where `indices` is a list of random pixels. In words, the pixels in `indices` survived, the rest is set to zero and therefore lost. We want to recover those lost pixels. Note that the images are considered being flattened and thus vectors, so that the masking operation (**??**) can be written as a matrix-vector product

$$b = Ax$$

for some quadratic matrix A. You will see below that A is not of full rank, so that we cannot simply solve this equation. Instead, we will seek for solutions of the regularized least squares problem (**??**) by solving the linear equation (**??**).
For this purpose will stick to a particular G which is related to what is called Sobolev (or H^1-) regularization. Specifically we consider the 1-d finite difference quotient

$$G = \begin{pmatrix} 1 & & & \\ -1 & 1 & & \\ & \ddots & \ddots & \\ & & -1 & 1 \end{pmatrix} \in \mathbb{R}^{(n+1)\times n}, \tag{4}$$

which has 1 on the main diagonal, -1 on the first lower off-diagonal, 0 everywhere else and is of dimension $(n+1)\times n$. Then, given the measured image b, the masking operator A (we assume that we know which pixels are original) and the regularization G we reconstruct the unknown image x by solving equation (**??**). See Figure **??** for an example experiment.

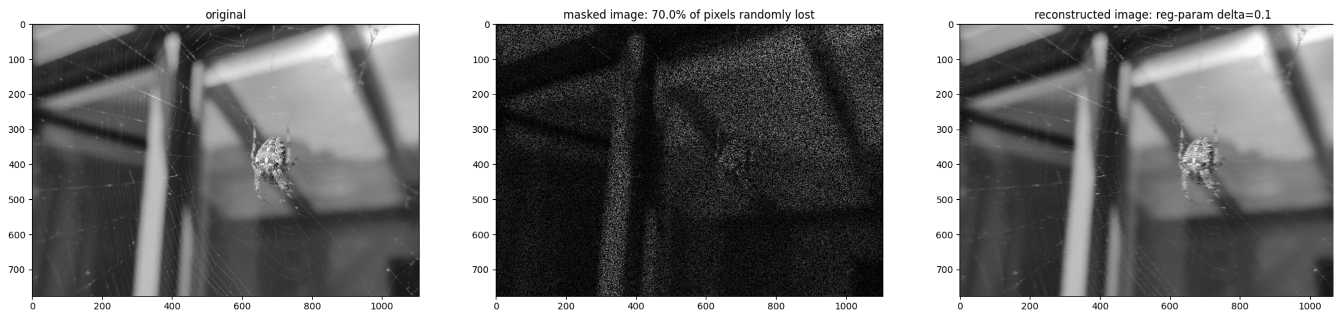
original
masked image: 70.0% of pixels randomly lost
reconstructed image: reg-param delta=0.1

Figure 1: Example experiment.

Step by Step

First note that due to the high dimensional image data you need to work with sparse matrices (`scipy.sparse`); specifically you should work with the CSR format.

1. **Choose original image:** Choose an image and load it in gray-scale as $H \times W$ `numpy.ndarray` using the code snippet:

```python
def load_image_as_gray(path_to_image):
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img
```

2. **Masking:** Write a function

   `b, indices = masking(img, percentage),`

   which takes as input an image `img` as $H \times W$ `numpy.ndarray` and a number `percentage` $\in (0,1)$ which indicates the percentage of pixels that are randomly kept. It shall return the masked image b as an $n := (H \cdot W)$-dimensional vector and the list `indices` $\subset \{0, \dots, n-1\}$ indicating which pixels are original.

   - Ultimately, the image needs to become a vector. For this purpose you can use for example the method `.ravel()`. No matter which method/function you choose, be aware of how the vector is flattened (standard is often: row-major/C-style order).
   - To generate a random set of indices based on the parameter `percentage` have a look at the function `numpy.random.choice`.

3. **Solving:** Write a function

   `reconImg = inpainting(b, indices, delta, G),`

   which expects a deteriorated image b as vector of length n, a list `indices` $\subset \{0, \dots, n-1\}$ of length $\leq n$ indicating which pixels are original, a regularization parameter `delta` > 0 and a matrix `G` $\in \mathbb{R}^{m \times n}$. It then solves (**??**) and returns the solution as an n-dimensional vector `reconImg`.

   - The sparse $(n \times n)$ masking matrix A is zero everywhere except for $a_{ii} = 1$ for $i \in$`indices`. For example, you can easily implement this matrix with `sparse.coo_matrix` and then transform it to CSR format via its method `.tocsr()`.
   - You can implement the sparse matrix G from (**??**) with the help of the function `scipy.sparse.eye`.
   - You can then solve the system (**??**) with `scipy.sparse.linalg.spsolve`.

4. **Analysis:**

   - Play around with different choices for the parameters `delta` and `percentage`.
   - You can plot your images (original, masked, reconstructed) with

     `matplotlib.pyplot.imshow((...).reshape(H,W), cmap='gray').`

   - Bonus: Try to recover the image with standard Tikhonov regularization, i.e., $G = I$. Do you have an idea why this does not work here?
   - Bonus: With this choice of G, you get flattening artifacts in one dimension (horizontal or vertical depending on your flattening approach) and at the boundary of the reconstructed image. Do you have an idea why?

(12 Points)

Solution:

```python
import numpy as np
import matplotlib.pyplot as plt
import scipy.sparse as sparse
import scipy.sparse.linalg


def load_image_as_gray(path_to_image):
    import matplotlib
    img = matplotlib.image.imread(path_to_image)
    # ITU-R 601-2 luma transform (rgb to gray)
    img = np.dot(img, [0.2989, 0.5870, 0.1140])
    return img


def masking(img, percentage):
    """Randomly sets (1-percentage)*100 % of the pixels to zero

    Parameters
    ----------
    img : (H, W) ndarray
          original image
    percentage : float
                 number in (0,1)

    Returns
    -------
    b : ndarray
        masked image of shape (H*W, 1)
    indices : list
              of length <=H*W, subset of {0,...,(H*W-1)}
              contains indices of pixels that were kept
    """
    # flatten image in C order, i.e., append row by row
    img = img.ravel()
    n = len(img)
    # masking operator
    indices = np.random.choice(np.arange(n), replace=False,
                               size=int(n * percentage))
    b = np.zeros(n)
    b[indices] = img[indices]
    return b, indices


def inpainting(b, indices, delta, G):
    """
    inpainting based on trivial 1-d Sobolev Regularization

    Parameters
    ----------
    b : ndarray
        of shape (n, 1)
    indices : list or array-like
              of length <= n, subset of {0,...,(n-1)}
              contains indices of pixels that were kept
    delta : float
            positive number, regularization parameter
    G : numpy.ndarray
        of dimension (m,n), determining regularization

    Returns
    -------
    reconImg : ndarray
               of shape (n, 1)
               reconstructed image
    """
    n = len(b)
    # masking operator with A = A.T@A (n,n)
    A = sparse.coo_matrix((np.ones(len(indices)),
                          (indices, indices)), shape=(n, n)).tocsr()
    # solve with scipy sparse (note that: A.T@A = A)
    reconImg = scipy.sparse.linalg.spsolve(A + delta * G.T@G, A.dot(b))
    return reconImg


if __name__ == "__main__":

    # INPUT PARAMETERS
    # --------------
    path_to_image = 'spider.jpg'  # 'happy_dog.jpg' # the image
    percentage = 0.1  # we randomly keep 100*percentage % of the data
    delta = 0.5  # regularization parameter

    # ORIGINAL IMAGE
    # --------------
    img = load_image_as_gray(path_to_image)
    H, W = np.shape(img)
    # plot original image
    plt.figure("Image Inpainting")
    plt.subplot(1, 3, 1)
    plt.imshow(img, cmap='gray')
    plt.title("original")

    # MASKED IMAGE
    # -----------
    b, indices = masking(img, percentage)
    n = len(b)
    # plot noisy image
    plt.subplot(1, 3, 2)
    plt.imshow(b.reshape((H, W)), cmap='gray')
    plt.title("masked image: {}% of pixels randomly lost".format(
              (1-percentage)*100))

    # RECONSTRUCTED IMAGE
    # -------------------
    # difference quotient (k+1,k)
    G = sparse.eye(n,n) # sparse.eye(n+1, n, k=0) - sparse.eye(n+1, n, k=-1)
    reconImg = inpainting(b, indices, delta, G)
    # plot denoised image
    plt.subplot(1, 3, 3)
    plt.imshow(reconImg.reshape((H, W)), cmap='gray')
    plt.title("reconstructed image: reg-param delta={}".format(delta))
#    plt.savefig("Image_Inpainting")
```

Total Number of Points = 26 (T:14, P:12)