COMP0114 - Coursework 1

Due: Wednesday January 25th, 2023 at 16:00

Week 1:

(a) See the code.

(b) See the code. We can note that we have had to compute the first solution for $p = 1.0001$ instead of 1 for numerical reasons. We also could have used an absolute value instead.

The results can be seen in (c).

(c) As expected, different norms yield different solutions along the constrained axis.

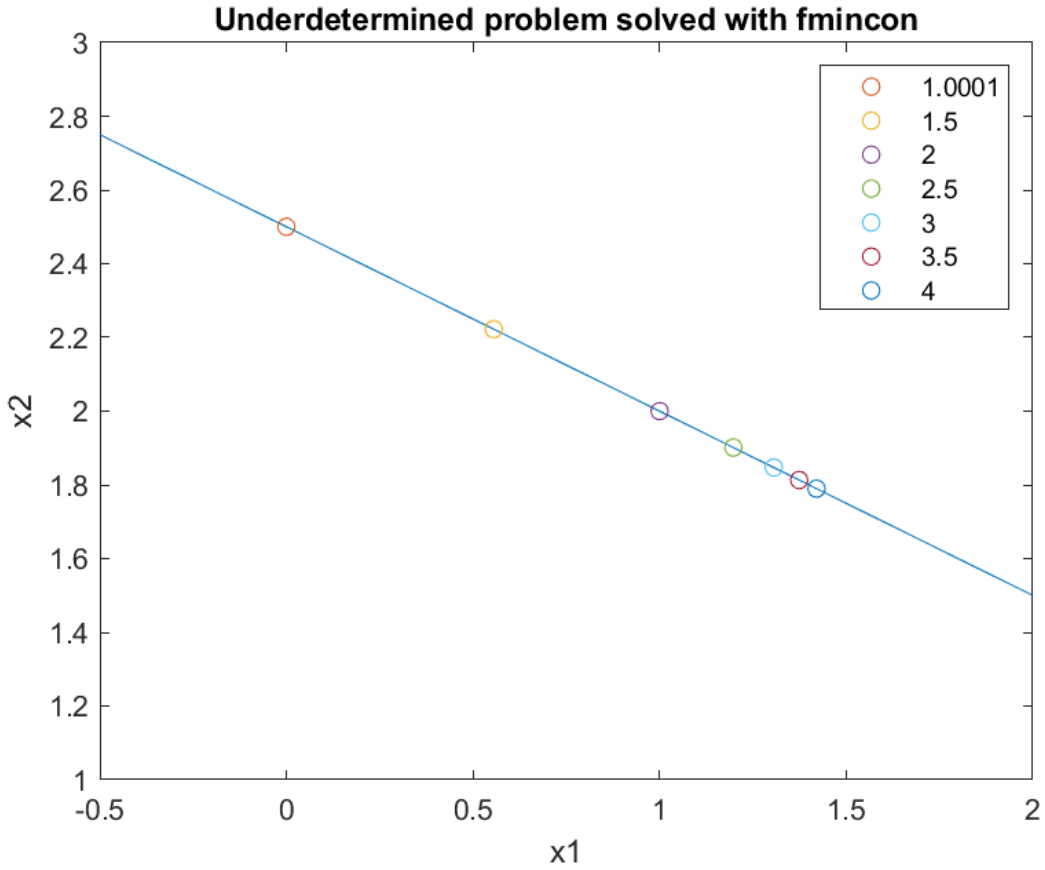

Figure 1: Different solutions for the underdetermined optimisation problem

(d) We can observe that the solution computed with the Moore-Penrose inverse corresponds to the solution computed for the $p = 2$ norm.

It comes to the fact that we are solving this optimization problem:

$$x^* = arg\min_x \|Ax - b\|^2$$

.

Taking the derivative and solving for x gives the pseudoinverse, and we can then show that taking this x solution and adding to it some value from the range space or the null space of the A matrix will always lead to a greater x value, thus showing that the x obtained with the pseudoinverse is the minimum. The full demonstration can be found in section 2.3.2 of the lecture notes. It relies on the fact that the norm 2 verifies the triangle inequality, thus that $\|x^{\dagger} + x^{'}\|^2 \leq \|x^{\dagger}\|^2 + \|x^{'}\|^2$, so that the quantities are always positive, so always add up to the final result.

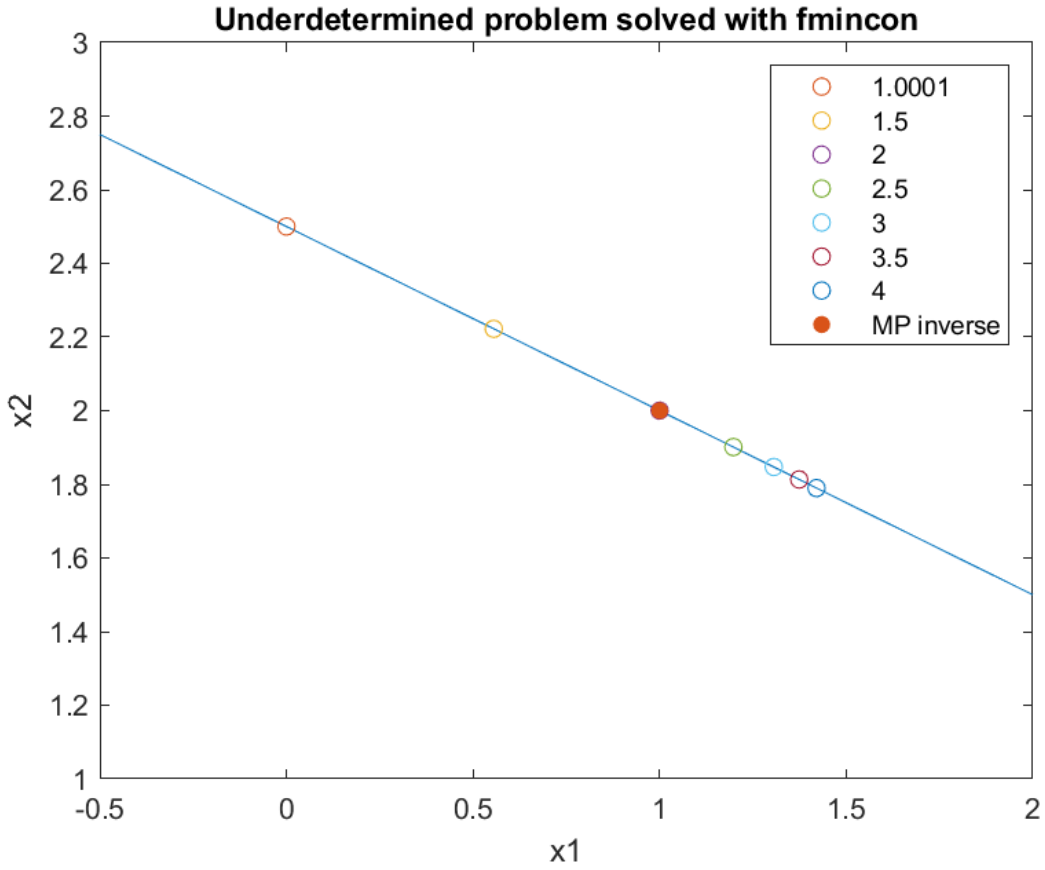

Figure 2: Different solutions for the underdetermined optimisation problem, including the Moore-Penrose inverse

Week 2:

(a) See the code.

(b) See the code.

(c) See the code.

(d) We can clearly see a Gaussian appear, with the highest values on the diagonal, and decreasing when getting further from it.

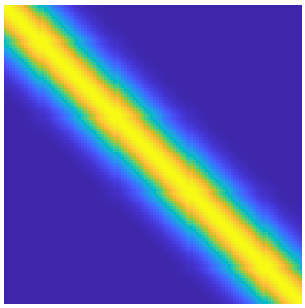

Figure 3: Matrix A (Gaussian kernel)

(e) We have computed the norm difference between the two matrices:

```
Norm difference between the original and reconstructed matrices: 1.8479
e-15
```

We can therefore say that the SVD reconstruction is accurate, for our $n = 100$ case.

(f) To construct the pseudoinverse method, we have stored W as a sparse matrix (because it is a square diagonal matrix in our case), and have computed the inverse of each term.

Here is our MATLAB output for the different norm differences computed:

```
n = 10
Norm difference between Id and WdW: 1.1102e-16
Norm difference between Id and WWd: 1.1102e-16
Norm difference between Id and AdA: 3.9138e-15
Norm difference between Id and AAd: 6.2148e-15
Norm difference between Ad computed by MATLAB's pinv and by SVD: 0
```

Our method generates good results and saves storage space. We even achieve a score of 0 between MATLAB's pinv and our SVD inverse renconstruction, that is because MATLAB also uses the SVD decomposition to compute the pseudoinverse of a matrix.

(g)
```
n = 20
Norm difference between Id and WdW: 1.1102e-16
```

```
Norm difference between Id and WWd: 1.1102e-16
Norm difference between Id and AdA: 4.3531e-10
Norm difference between Id and AAd: 3.5053e-10
Norm difference between Ad computed by MATLAB's pinv and by SVD: 0
```

The reconstructions are still accurate for the SVD pseudoinverse and MATLAB's pseudoinverse. But, even if our $W^\dagger$ is correct, our SVD pseudoinverse achieves a worse score than for $n = 10$. We have a first indication that the SVD operation might depend on the dimension of the matrix. Let's increase once more the dimension of A:

```
n = 100
Norm difference between Id and WdW: 1.1102e-16
Norm difference between Id and WWd: 1.1102e-16
Norm difference between Id and AdA: 7.3714
Norm difference between Id and AAd: 5.8686
Norm difference between Ad computed by MATLAB's pinv and by SVD: 6.0817338387023
e+16
```

We can clearly see that for higher dimensions, taking the inverse of W makes the values explode. Inverting W is still something we can do, but once we multiply it with U and V, we do not achieve reconstruction at all.

We can recall the formula to compute the pseudoinverse from the SVD:

$$A^\dagger = \sum_{i=1}^{100} \frac{\boldsymbol{v_i u_i^T}}{w_i}$$

We clearly see in Figure 12 that the singular values take infinitesimal values (and decrease almost linearly on a log scale, i.e exponentially on a regular scale). Therefore, the inverse of A will be very unstable: the singular values in the denominator will make any small change in the UV product have a disproportionate impact. We already know that due to numerical imprecisions, there will be a small noise due to the smallest values in A (the smallest values in A are around $1e-24$), and as a result, this residual noise will be greatly amplified.

Moreover, we know that the inverse of A is built from the columns of matrix V, the row space. We can see in Figures 4 to 6 that the first columns give vectors varying nicely: small fluctuations, no sudden change of values, this is the low-frequency part, corresponding to the highest singular values. But, as we have chosen a high dimension, comparing Figures 7 to 9, we have decided to add higher and higher frequencies. To the point where in Figure 9, the frequencies are so high due to the very low singular values, that we cannot distinguish clear frequencies, it looks more like noise.

We, therefore, obtain a noisy and inaccurate estimate.

What can we say about the difference between our SVD reconstruction and the pinv MATLAB function then? MATLAB uses the SVD decomposition too, so we should have obtained similar results. But MATLAB discards the singular values inferior to a parameter `tol` we can change. By default, it is set to $1e-4$. Thus, by looking at our

plotted singular values (Figures 10 to 12), we understand that to prevent numerical instabilities from happening, it has discarded the majority of singular values. That is a process of regularisation.

As a conclusion for this part, we can say that the SVD is useful to detect possible ill-conditioned matrices. The SVD will not perform well in higher dimensions for these, we have to introduce regularisation to it.

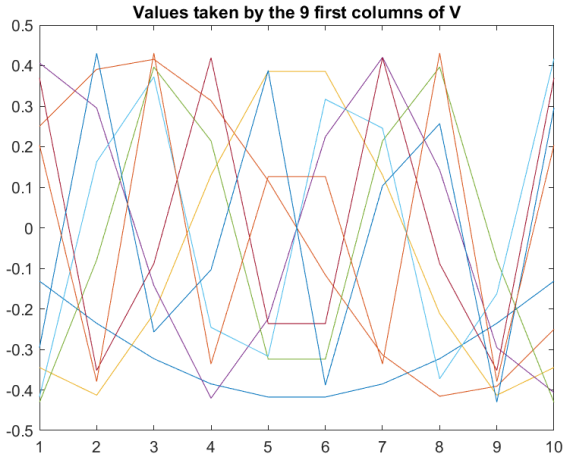

Figure 4: Values taken by the 9 first columns of V, case n = 10

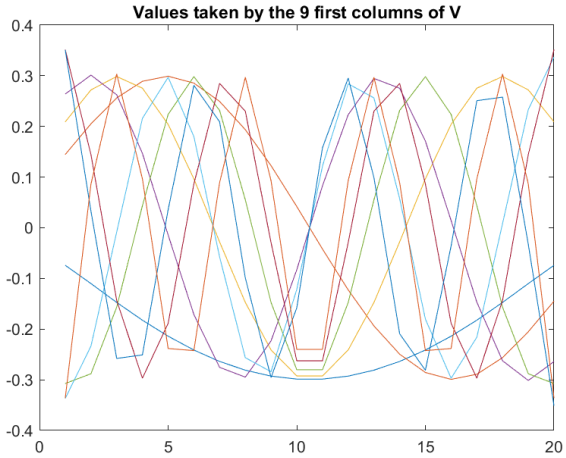

Figure 5: Values taken by the 9 first columns of V, case n = 20

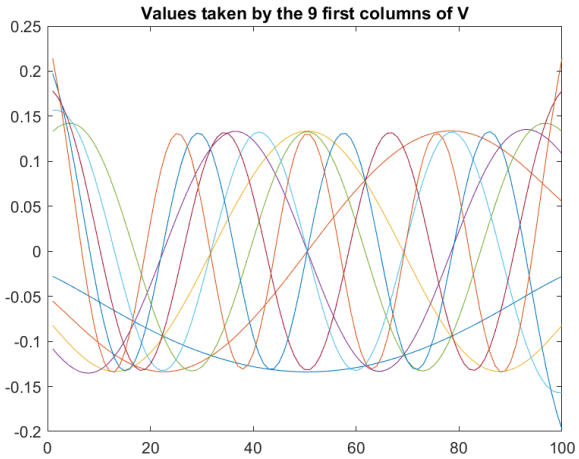

Figure 6: Values taken by the 9 first columns of V, case n = 100

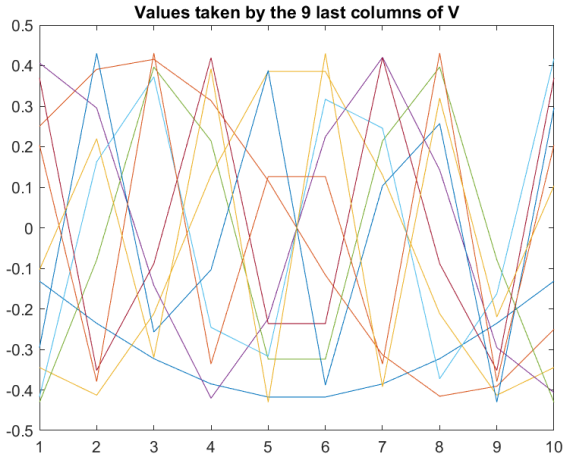

Figure 7: Values taken by the 9 last columns of V, case n = 10

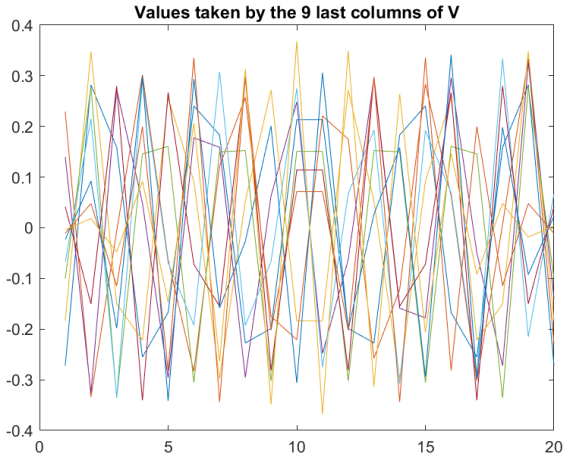

Figure 8: Values taken by the 9 last columns of V, case n = 20

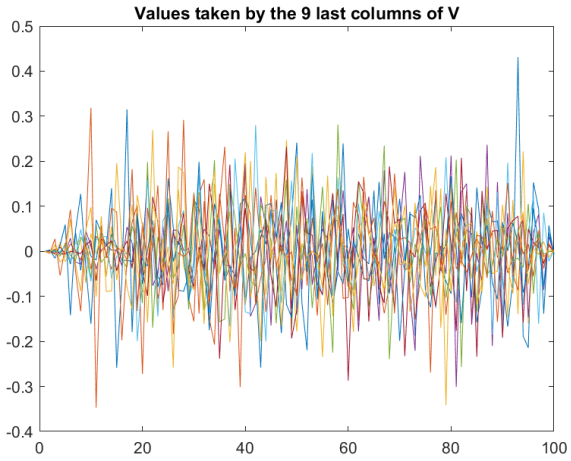

Figure 9: Values taken by the 9 last columns of V, case n = 100

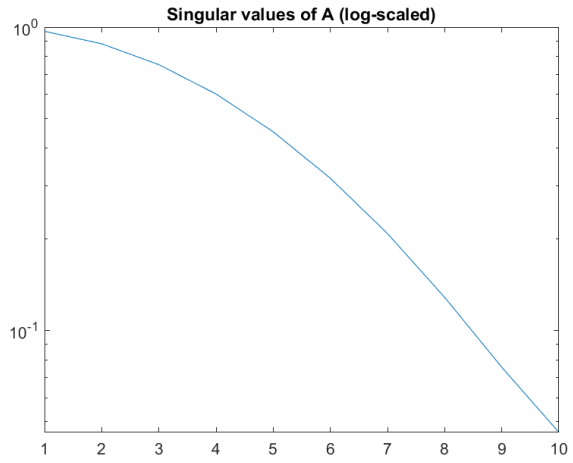

Figure 10: Singular values of the A matrix, case n = 10

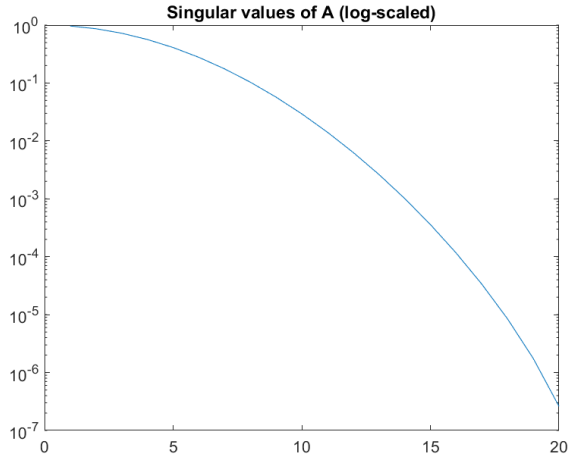

Figure 11: Singular values of the A matrix, case n = 20

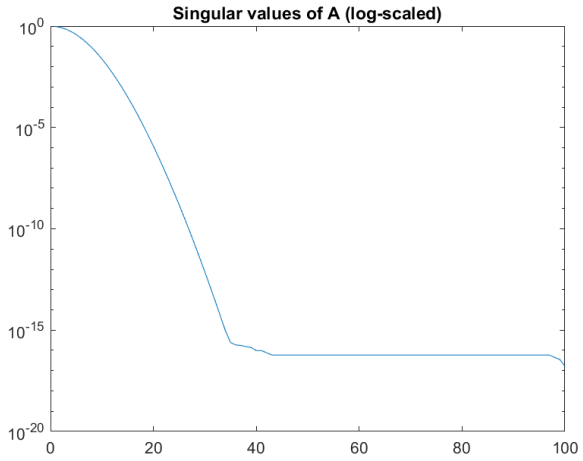

Figure 12: Singular values of the A matrix, case n = 100

Week 3:

(a) We have chosen $n = 200$ points, here is the plot we obtain:

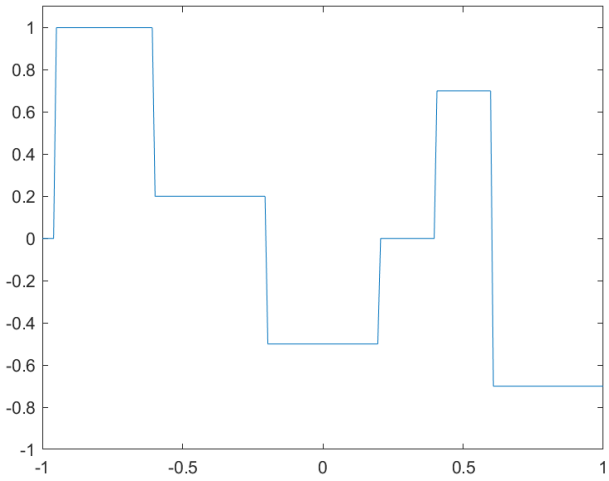

Figure 13: Plot of the f function with n=200 points

(b) Here are the plotted singular values of the Gaussian kernel (matrix A) for the different values of σ:

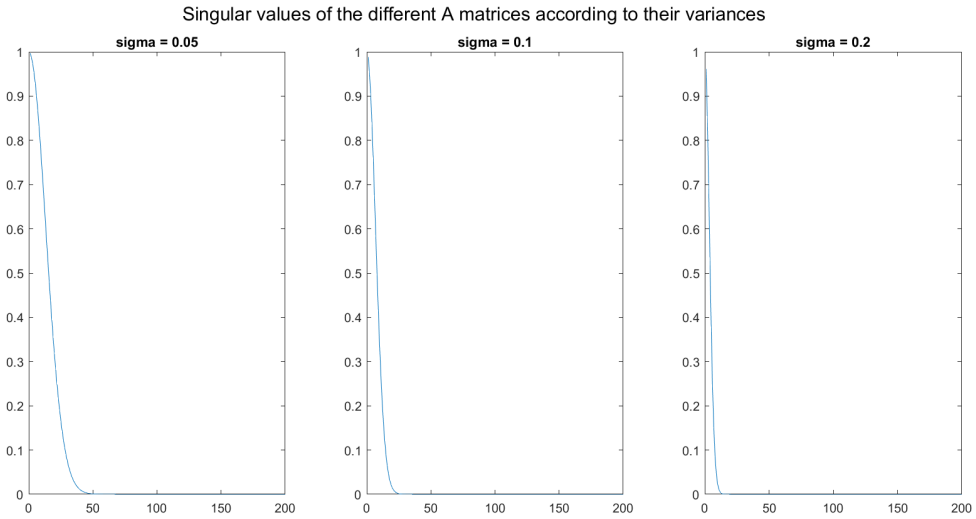

Figure 14: Singular values of the Gaussian kernel for the different values of sigma

(c) It is clear from Figure 14 that the plotted values look like half a Gaussian. We have put the singular values twice in a new vector, so we could simulate an entire Gaussian, and we have estimated its variance. The results are as follows:

$$\sigma_{0,05} = 0,3146 \qquad \sigma_{0,1} = 0,2278 \qquad \sigma_{0,2} = 0,1611$$

$$\sigma^2_{0,05} = 0,0990 \qquad \sigma^2_{0,1} = 0,0519 \qquad \sigma^2_{0,2} = 0,0260$$

We can see that the variance of the singular values is going in the opposite direction to the data variance: the more spread the Gaussian is, the less their singular values are spread. It can be explained by the fact that the more a Gaussian is spread, the more smooth the variation is between the higher and lower values, therefore the less high-frequency terms there are, so the less need there is to go further in singular values.

(d, e, f) Let's first explain our method.

The convolution is equivalent to the matrix multiplication of the kernel A with our data. Each column of A represents a movement of the Gaussian (it starts at the top, and progresses all the way to the bottom).

To perform the term-by-term multiplication in the Fourier domain, we have to use a 1D kernel instead of a matrix. We have chosen a vector representing a matrix centered in 0, of variance σ (depending on the kernel).

The results can be found in the first two rows of Figure 16 below.

All figures seem alike, except for two parts: the left and right boundaries. We can clearly see a sharper slope in the second method. As if there were missing data for the first method. To understand what is missing, let's look at the kernel used:

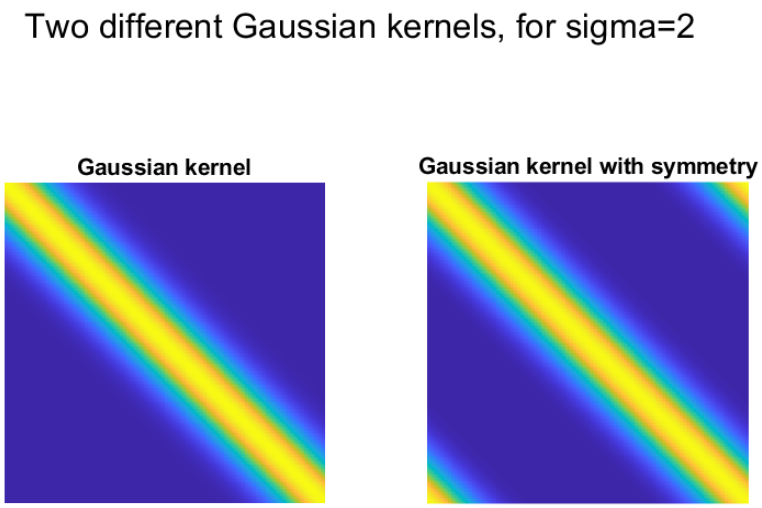

Figure 15: Plot of two Gaussian kernels for sigma=2

We have used the kernel on the left of Figure 15. What is missing is that the Gaussians at the start and the end are incomplete. Taking the sum of each column, we do not find 1 for the first and last columns. To complete them, we add the missing part of the Gaussian below and above, the result is the right kernel in the Figure.

Now, if we run again the matrix multiplication of the data but with our new kernel, we can see on the third row of Figure 16 that we achieve the same filtering as the one achieved in the Fourier domain. This emphasizes the importance of padding and border effects in convolutions.

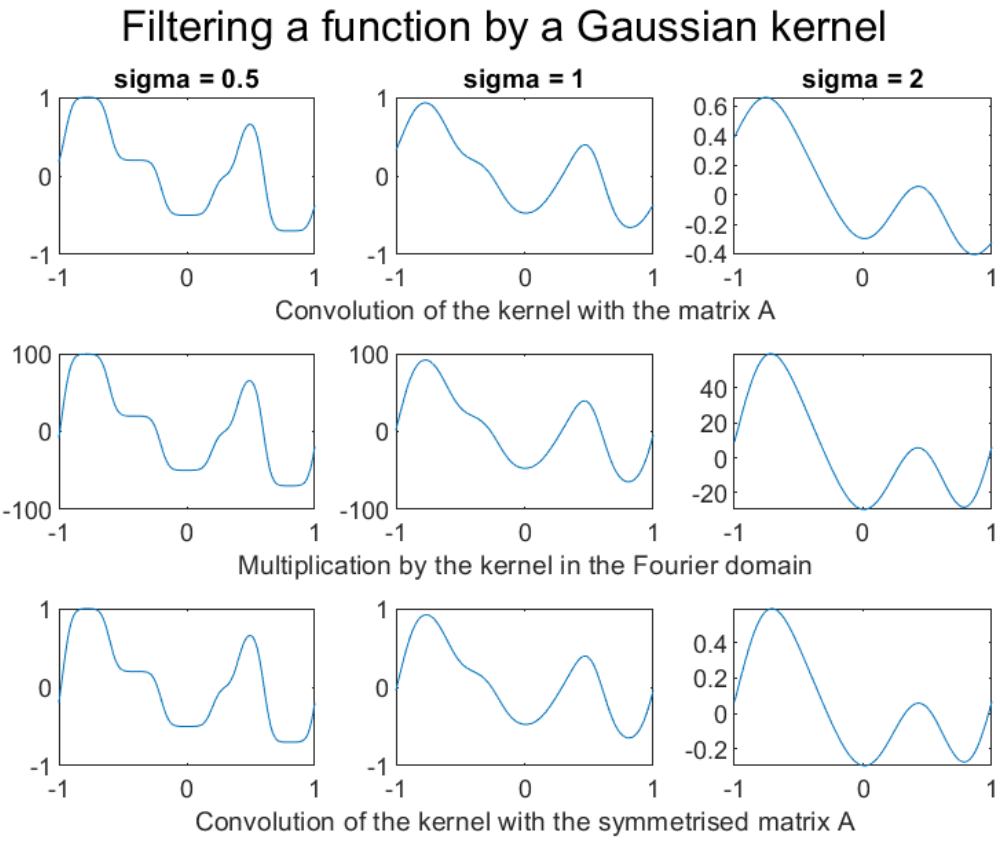

Figure 16: Plot of the different filtered signals for different filtering methods