Adaptive Array Signal Processing

[5SSC0]

Ruud van Sloun (responsible lecturer)
Iris Huijben (assistant)
Julian Merkofer (assistant)
Vincent van de Schaft (assistant)

Assignments Part 2

2022-2023

In this assignment we will explore some of the basic elements of Compressed sensing. Use the predefined answer sheet to deliver your report and **make sure to add snippets of relevant code to your report as well**.

1 Sparse Signals and Denoising

1.1 Sparse Signals

Before we start with compressed sensing (CS), we'll look at sparse signal de-noising. There's a strong connection between CS and denoising. Here we will attempt to denoise a sparse signal that is corrupted by random noise. Generate the sparse signal s, by creating a 1×128 vector with 5 non-zero ($[1:5]/5$) coefficients and permute them randomly.

```
>> x = [[1:5]/5 zeros(1,128-5)];
>> x = x(randperm(128));
```

Add random gaussian noise with a standard deviation $\sigma = 0.05$ to the signal, $y = x + n$.

```
>> y = x + 0.05*randn(1,128)
```

Many approaches for denoising and regularization use the Tychonov penalty to estimate the signal x from noisy data y. Specifically, they try to solve:

$$\arg\min \frac{1}{2}||\hat{x} - y||_2^2 + \lambda \frac{1}{2}||\hat{x}||_2^2. \tag{1}$$

Q1: Explain which tradeoff is being optimized here, and why the Tychonov penalty works for the problem of denoising.

The nice thing about this approach that it has a closed form solution, and finding the minimum is a linear problem.

Q2: Show that the solution for this problem is, and take into account in your derivation that both x and y are vectors:

$$\hat{x} = \frac{1}{1+\lambda} y. \tag{2}$$

Q3: How does the resulting estimate of x depend on λ? Include plots for $\lambda = \{0.1, 1, 3\}$ in your report.

1.2 Regularization

Instead of Tychonov regularization, which penalizes the ℓ_2 norm ($||x||_2 = \sqrt{\Sigma x_i^2}$), we will now employ the ℓ_1 ($||x_1|| = \Sigma|x_i|$) based penalty. That is, we will solve:

$$\arg\min \frac{1}{2}||\hat{x} - y||_2^2 + \lambda||\hat{x}||_1. \tag{3}$$

Q4: Show that the solution to this problem is:
(Remember that both x and y are vectors)

$$\hat{x} = \begin{cases} y + \lambda, & \text{if } y < -\lambda \\ 0, & \text{if } |y| < \lambda \\ y - \lambda, & \text{if } y > \lambda \end{cases} \tag{4}$$

Write a function `SoftThresh` with inputs y and λ that returns the ℓ_1 penalyzed estimate for x, i.e. $\hat{x}$. The effect of this function is often referred to as soft-thresholding or shrinkage. Check that your soft-thresholding implementation also works for complex numbers, i.e. the behaviour on the magnitude of the complex number should be equal as the behaviour on a real number:

$$\hat{x} = \texttt{SoftThresh}(y, \lambda) = \begin{cases} 0, & \text{if } |y| \leq \lambda \\ \frac{|y|-\lambda}{|y|} y, & \text{if } |y| > \lambda \end{cases} \tag{5}$$

Q5: Plot the output of the soft-thresholding function for a list of real numbers: $y \in [-10, 10]$ and complex numbers: $y \in [0, 10] * \exp(j)$ for $\lambda = 2$. Describe what this soft-thresholding functions does (for generic λ) and how this behaviour enforces sparsity. Don't forget to add your implementation of the soft-thresholding function to your report!

Q6: Apply shrinkage to the noisy signal with $\lambda = \{0.01, 0.05, 0.1, 0.2\}$, and include the different plots in your report.

Another well-known norm is the ℓ_0 norm, which counts the non-zero values in a vector. Figure 1 shows for a 1×2 vector $\mathbf{w} = [w_1, w_2]$, the contour plots of the three different norms ℓ_0, ℓ_1, and ℓ_2, respectively.

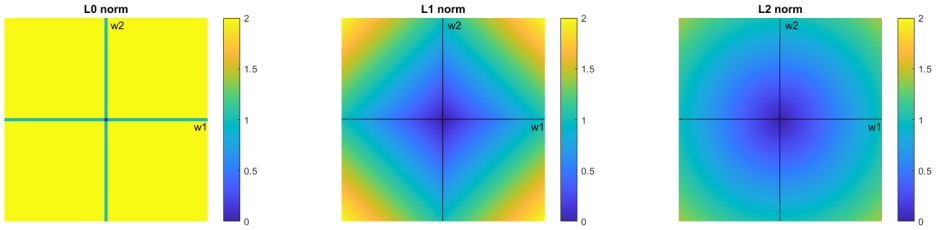

Figure 1: Contour plots of the three different norms.

Q7: What property of the ℓ_0 norm makes it challenging to optimize for? So, for sparse signal reconstruction, would you then use the ℓ_1 norm or ℓ_2 norm, and why?

1.3 Undersampling for compressive sensing

As we mentioned before, there is a strong connection between compressive sensing and denoising. We'll now explore this connection and the importance of incoherent undersampling. In the rest of this assignment we omit measurement noise vector n.

Undersampling data can have several benefits. In MR imaging it for example reduces the total time needed to create a scan as a lower number of (spatial) samples have to be acquired.

Q8: Can you mention two other reasons why one would be interested to reduce system sampling rates in general?

Undersampling can be achieved by designing a sampling matrix $A \in \{0, 1\}^{M \times 128}$, that takes M samples ($M < 128$): it contains exactly one 1 per row (the rest is zero), and all rows are unique. In this way, it exactly selects the M (different) samples at the indices where the

ones are positioned in A. When applying such undersampling, the optimization problem to reconstruct x changes to:

$$\arg\min \frac{1}{2}||A\hat{x} - y||_2^2 + \lambda \frac{1}{2}||\hat{x}||_1, \tag{6}$$

where $\hat{x}$ is the estimated signal, $A\hat{x}$ is the undersampled estimate, and y is the measurement vector that contains the samples that we have acquired.

Q9: Why can we not find a closed-form solution for this optimization problem in the way we did it for Eq. 3 in Section 1.2?

In the following we move one step further and we will sample in another domain than in which we want to reconstruct the signal, as this is common in practical applications. An example is MRI, where samples are acquired in k-space, a spatial Fourier representation, whereas the images should be reconstructed in time domain. We will simulate data that has been (under)sampled in the Fourier domain, which we aim to fully reconstruct in the time domain again. Our optimization problem now becomes:

$$\arg\min \frac{1}{2}||AF\hat{x} - y||_2^2 + \lambda \frac{1}{2}||\hat{x}||_1, \tag{7}$$

with $\hat{x}$ the estimated signal, $AF\hat{x}$ the undersampled Fourier transform of this estimate, and $y = AFx$ the measurement vector that contains the samples of the Fourier domain that we have acquired.

By undersampling, we measure an underdetermined set of data, for which no unique analytical solution for the signal reconstruction problem exists. However, compressed sensing makes the *important assumption* that the signal must be sparse in a transform domain. So if we reconstruct our signal in that specific domain, we know that our reconstructed signal should be sparse. Using this knowledge, there is hope that we will be able to reconstruct the signal even though the problem was underdetermined. The theory of compressed sensing suggests to apply random undersampling (opposed to uniform undersampling). To see why, we will compare both sampling strategies.

Start with creating the Fourier domain representation of x: $X = Fx$, where F is the Fourier transform operator (use the `fft()` function from Matlab). Undersample X by taking 32 equispaced samples using sampling matrix A. Then, compute the inverse Fourier transform on the zero-filled subsampled data, and multiply by 4 to correct for the fact that we have only 1/4 of the samples.

Hint: If you easily want to zero-fill the Fourier domain at the locations where you did not sample, you can use the transpose of the sampling matrix for this:
>> $A^T AFx$
resulting in a zero-filled partial Fourier measurement. Check this by yourself!

Q10: Plot the magnitude of the inverse Fourier transform results and answer the following questions (in max. 3 sentences in total): What do you notice in the reconstructed signal?, What is causing this?, Are we going to be able to reconstruct the original signal from the result?

Now, we will undersample X by taking 32 random samples (Hint: Use the `randperm()` function). Compute the zero-filled inverse Fourier transform and multiply by 4 again.

Q11: Plot the magnitude and describe the result by answering the following questions (in max. 4 sentences in total): In what way is the result different than the result after uniform subsampling?, What happened with the aliasing that we saw before?, How does this resemble the denoising problem?, Are we going to be able to reconstruct the original signal from the result?

1.4 Signal reconstruction

We now learned how to undersample a signal in Fourier domain, and that the inverse Fourier transformations do not results in good estimates of x yet. We also learned what kind of penalty to use for sparse signal reconstruction. Let's take a better look at how to exactly reconstruct the sparse signal from the undersampled data.

Since there is no close-form solution for Eq. 7, an iterative algorithm can help to find the solution. In this assignment, you will implement two such iterative algorithms.

The first one is a Projection Over Convex Sets (POCS) type of algorithm. The idea of such algorithms, is that steps are consecutively made to adhere to the measured data (data consistency), and move towards the regularizer (ℓ_1 penalty in this case).

Consider y, which is your undersampled Fourier measurement: $y = AFx$. A first (naive) guess of $\hat{X}$, is the zero-filled Fourier measurement: $A^T AFx$
So initialize: $\hat{X}_0 = A^T AFx$. Then:

$\hat{x}_0 = 0$;
$\hat{x}_1 = 0$;
i = 0;
while $||\hat{x}_{i+1} - \hat{x}_i||_2 > \epsilon$ or $i = 0$ **do**
  Calculate the inverse Fourier transform $\hat{x}_{i+1} = F^{-1}\hat{X}_i$;
  Apply soft thresholding: $\hat{x}_{i+1} = \texttt{SoftThresh}(\hat{x}_{i+1}, \lambda)$;
  Compute its Fourier transform: $\hat{X}_{i+1} = F\hat{x}_{i+1}$;
  Enforce data consistency: $\hat{X}_{i+1}[j] = \begin{cases} \hat{X}_{i+1}[j], & \text{if } \hat{X}_0[j] = 0 \\ \hat{X}_0[j], & \text{otherwise} \end{cases}$,
end

where the subscript i denotes the iteration-index. Use the `fft()` and `ifft()` functions from Matlab to implemented F and F^{-1}, respectively.

Q12: Implement the POCS algorithm in Matlab, apply it to the randomly undersampled signal for $\lambda = \{0.001, 0.01, 0.1\}$, and visualize the results. Add your code to the report.

We will now implement a second type of algorithm that can iteratively reconstruct sparse signals, a proximal gradient algorithm: Iterative Soft-Thresholding Algorithm (ISTA). The idea of such proximal gradient algorithms is that steps are consecutively made in the negative gradient direction and towards (the proximity of) the regularizer (ℓ_1 penalty in this case). For the gradient update step, it uses the update rule you know from the steepest gradient descent algorithm:

$$\hat{x}_{i+1} = \hat{x}_i - \alpha \nabla_x J, \tag{8}$$

where $J = ||AF\hat{x} - y||_2^2$ and α is the step size. So we can write (verify this for yourself):

$$\hat{x}_{i+1} = \hat{x}_i - \beta F^H A^T (AF\hat{x}_i - y), \tag{9}$$

with β the step size.
To make the update rule of ISTA te, we sparsify the predicted $\hat{x}$ by applying the soft-thresholding function. Therefore, the complete update rule can be written as:

$$\hat{x}_{i+1} = \texttt{SoftThresh}\{\hat{x}_i - \beta F^H A^T (AF\hat{x}_i - y), \lambda\}. \tag{10}$$

Initialize $\hat{x}_0 = 4 * F^{-1} A^T AFX$, where $A^T AFX$ is your zero-filled undersampled Fourier measurement of X. Again, use the `fft()` and `ifft()` functions from Matlab to implemented F and F^{-1}, respectively.

Q13: Implement the ISTA algorithm in Matlab, apply it to the randomly undersampled signal for $\lambda = \{1e{-}5, 1e{-}4, 1e{-}3\}$, and visualize the results. Add your implementation to your report.

Q14: The influence of using a high or low value for λ is equivalent for both algorithms, what can you say about this influence?

Q15: What is the biggest conceptual algorithmic difference between POCS and ISTA?

Q16: Summarize the two most important assumptions for successfully using compressed sensing (max 2 sentences).

2 Compressed Sensing of Medical Images

We wil now explore 2D compressed sensing from undersampled medical image data. As mentioned before, MRI measurements are inherently taken from the k-space. Traditionally, the image is then recovered by performing a 2D inverse Fourier transform on the k-space data.

2.1 Phantom data

Let's first experiment with 2D data by creating a phantom image in Matlab, after which we sample its k-space:

```
>> im = phantom('Modified Shepp-Logan',64);
```

Where (:) transforms the 2D data into its vectorized form, and A is an $M \times 64^2$ matrix that describes (like before) the sampling procedure, and M is the number of samples we take. We want to sample the k-space randomly.

Q17: Create a random sampling matrix A for $M = 1024$. Make an efficient implementation without using a for- or while-loop and show your code.

Subsample the k-space by vectorizing the 2D k-space to 1D. Subsequently, reconstruct the undersampled k-space by using the inverse Fourier transform. (use ifft2()) on the (zero-filled) undersampled Fourier image and make sure to reshape the zero-filled image again to 2D before applying the 2D ifft)
Q18: Show your code and results.

You've seen that directly reconstructing in the Fourier domain did not work well. Have a good look at the phantom image. Is it sparse? Does it resemble the "spiky" type of data we used before? The obvious answer is: No. But that does not mean that we can not use compressed sensing for this problem. The key here lies in transforming the data into a domain in which it is actually sparse, using a sparsifying transformation Ψ^{-1}. The sparsity enforcing ℓ_1 penalty can then work on this sparse domain, rather than the measurement domain, and our optimization problem now becomes:

$$\arg\min_{\hat{x}_s} ||AF\Psi\hat{x}_s F - y||_2 + \lambda||\Psi^{-1}\hat{x}||_1, \tag{11}$$

where $\hat{x}_s = \Psi^{-1}\hat{x}$ is the sparse representation of estimate $\hat{x}$, and F is the DFT matrix. $F\hat{x}F$ is a matrix-wise notation of taking the 2D Fourier transform of $\hat{x}$.

Looking back at our phantom image, we can see that it is for example sparse in variations; the number of variations in the image are rare, i.e. transforming the image to a so-called total-variation (TV) domain makes it sparse. We can thus rewrite the sparsity penalty $||\Psi^{-1}\hat{x}||_{\ell_1}$ to $||D_x\hat{x}||_{\ell_1} + ||D_y\hat{x}||_{\ell_1}$ when we use the total variation domain as the sparsifying domain. D_x and D_y then represent the difference operators in the x and y direction of the image, respectively.

Instead of using our POCS or ISTA implementation, we will now use a convex optimization toolbox of Matlab called CVX. You can download it at `http://cvxr.com/cvx/download/`. After installing CVX in your working directory (and making sure that all subfolders are also part of this working directory), we will have to set it up:

```
>> run('path_to_cvx_setup.m_file');
```

Q19: Implement the optimization problem given in equation 11 in Matlab using CVX (note: this solver is rather slow and can take up to 10-45min, dependent on your laptop). Add your code and reconstruction plots, and explain the differences in reconstruction quality between inverse Fourier reconstruction and TV-based reconstruction.
Hint: Create the DFT matrix F using dftmtx().

2.2 MRI brain data

We will now apply these strategies to a T_2 weighted MRI image of the brain. You probably first want to downsample the image data to 64×64, as reconstruction for large images is slow with CVX:

```
>> im = imresize(abs(data.im),[64,64]);
```

Q20: Apply the same strategies as before (Fourier- and TV-domain based reconstructions) to the brain MR image and compare both results to the reconstructions of the phantom image. What do you see and why?

Q21: How can the use of another domain transformation (than TV) contribute to a better reconstruction from the undersampled data? Do some literature search to such a domain transformation for MRI and report it (only report an example with a reference, not a full report of your literature search).

Q22: Suppose you want to implement MR image reconstruction via this sparsifying domain by adapting your earlier-written POCS algorithm, then you'll have to adjust it to deal with 2D data. Apart from that, you'll have to implement two other major changes, describe where in the algorithm these changes take place and what these changes are.