Computational MR imaging
Laboratory 8: Compressed Sensing

Code submission is due by 12:00 before the next Thursday lab section. Please upload your code to StudOn in a described format. Late submissions will not be accepted.

Learning objectives

- Refresh your linear algebra skills
- Apply compression transforms (e.g., wavelets) to obtain sparse representations of MR images
- Reconstruct randomly undersampled k-space data using compressed sensing approach

1. **Sparsity/compressibility of brain images using the wavelet transform:**
   Medical images are generally not sparse, but they usually have a sparse representation after applying an appropriate transform. An example is the wavelet transform, which is the core transform used in the JPEG2000 standard. Wavelet coefficients are sub-band filters that hold both spatial (pixels) and frequency (k-space) information and thus they are able to represent an image with fewer non-zero coefficients.

   1.1. The data, file data_lab8.mat, is loaded on these valiables
   - kdata_fs: the fully-sampled kdata
   - kdata_us: the undersampled kdata

   1.2. Implement a method, dwt2.
   1.2.1. You should split the complex-valued data into real and imaginary data.
   1.2.2. Use pywt.wavedec2 for 2D discrete wavelet transform to the real and imagniary data.
   1.2.3. Use pywt.coeffs_to_array to re-arrange the wavelet coefficient list from 1.2.2. into a single array.
   1.2.4. Merge the real and imaginary data into a compelx-valued coefficient arrays. Since coefficient slices are the same for real and imaginary data, you can still use it for the compelx data as well.

   1.3. Implement a method, idwt2.
   1.3.1. You should split the complex-valued coefficient array into real and imaginary coefficient arrays.
   1.3.2. Use pywt.array_to_coeffs to convert a combined array of coefficients back to a list. Since you used pywt.wavedec2 for 2D discrete wavelet transform, the output_format of array_to_coeffs should be *wavedec2*.
   1.3.3. Use pywt.waverec2 for 2D inverse discrete wavelet transform.
   1.3.4. Merge the real and imaginary data into a complex-valued reconstructed data.

   1.4. Plot the reconstruction of kdata_fs and its wavelet representation.

1.5. Implement a method, is_wt_sparser, to determine if wavelet coefficients are sparser than the ground truth image based on the L1-norm.

1.6. Implement a method, compress.

1.6.1. Set the threshold to the $\frac{N}{Compression\ factor}$-th highest absolute value. (hint: sort the wavelet coefficients in a descending order using the np.sort and np.flip)

1.6.2. The coefficients in coeff_arr that are smaller in absolute value than the threshold are set to zero.

1.7. Compress the brain image by factors 5, 10 and 20 using the wavelet transform.

1.8. Plot the compressed images with NMSE and their error images.

1.9. Which compression ratio would you choose?

2. Compressed sensing reconstruction using iterative soft thresholding

2.1. Plot the undersampling pattern of kdata_us and compute the acceleration factor.

2.1.1. Implement calc_acc_rate.

2.1.2. Implement get_sampling_mask

2.2. Implement a method, soft_threshold.

2.3. Implement a method cs_ista.

2.3.1. Implement a method, calc_cost. Look at the lecture note p.22.

2.3.2. Implement methods, cs_ista_x, corresponding to each step x below.

$$\mathbf{m}_{n+1} = \underbrace{\mathbf{T}^{-1}\left[\underbrace{S\left(\underbrace{\mathbf{T}\left[\underbrace{\mathbf{m}_n - \underbrace{\mathbf{E}^H\left(\underbrace{\underbrace{\mathbf{E}\mathbf{m}_n}_{1} - \mathbf{d}}_{2}\right)}_{3}}_{4}\right]}_{5}, \lambda\right)}_{6}\right]}_{7}$$

2.3.3. Set the threshold to lamda_percent of the maximum absolute value of the initial solution

2.4. Reconstruct kdata_us using your iterative soft-thresholding algorithm. Try your reconstructions with λ=5%, 1% and 0.5%. Plot the initial solution, final reconstruction and corresponding error images with respect to the fully-sampled one. Compute the value of the cost function for each iteration.