1. Calculate the Radon transform of an image and test the back-projection method

The file `SLphan.npy` contains the image. You just need to load this with `numpy.load('SLphan.npy')`. We require the Astra Toolbox (`Python 3.6` is required) to perform Radon transform. To install the toolbox with `conda` run one of the following:

```
conda install -c astra-toolbox astra-toolbox
conda install -c astra-toolbox/label/dev astra-toolbox
```

The following code finds the Radon transform of an 2D image $\boldsymbol{f}$ in 1 degree intervals from $0-180$,

```
import numpy as np
import astra

# Create volume geometries
v,h = f.shape
vol_geom = astra.create_vol_geom(v,h)

# Create projector geometries
angles = np.linspace(0,np.pi,180,endpoint=False)
proj_geom = astra.create_proj_geom('parallel',1.,det_count,angles)

# Create projector
projector_id = astra.create_projector('strip', proj_geom, vol_geom)

# Radon transform (generate sinogram)
sinogram_id, sinogram = astra.create_sino(SLphantom, projector_id)
```

where `det_count` is the number of detector pixels in a single projection or in other words the "number of projection samples". The following call implements (unfiltered) back-projection

```
# Create a data object for the reconstruction
rec_id = astra.data2d.create('-vol', vol_geom)

# Set up the parameters for a reconstruction via back-projection
cfg = astra.astra_dict('BP')
cfg['ReconstructionDataId'] = rec_id
cfg['ProjectionDataId'] = sinogram_id
cfg['ProjectorId'] = projector_id

# Create the algorithm object from the configuration structure
alg_id = astra.algorithm.create(cfg)

# Run back-projection and get the reconstruction
astra.algorithm.run(alg_id)
f_rec = astra.data2d.get(rec_id)
```

Simply changing the parameter

```
cfg = astra.astra_dict('FBP')
```

computes the filtered back-projection. You can add noise on the sinogram and obtain the ID as follow:

```
gNoisy =  astra.functions.add_noise_to_sino(g,theta)
gNoisy_id = astra.data2d.create('-sino',proj_geom,gNoisy)
```

where `g` is the measurement (sinogram) and `theta` is the noisy level.

2. Calculate an explicit matrix form of the Radon transform and investigate its SVD

Let's assume we have an image $\boldsymbol{f}$ of dimensions 64×64 and we want to compute the Radon transform for the angle vector with 45 angles:

```
angles = numpy.linspace(0,numpy.pi,45,endpoint=False)
```

Then the measurement g (sinogram) will be of dimensions 95×45. Where 45 is the "number of angles" and 95 is the "number of projection samples" or in other words the resolution of your measurement detector. In vectorized form $\boldsymbol{f}$ is then of dimension $64^2 = 4096$ and $\boldsymbol{g}$ will be of size $45 \cdot 95 = 4275$. Your matrix representation of the radon transform then needs to map the vectorized image to the vectorized measurement, that is $\mathsf{A} : \mathbb{R}^{4096} \to \mathbb{R}^{4275}$, in other words your matrix is of size 4275×4096 (rows $\times$ columns).

You can use '`numpy.linalg.svd`' or '`scipy.sparse.linalg.svds`' to only compute the singular values.

To construct the matrix A, you need to iterate through the image column by column and set the 'order' argument in `numpy.reshape` function as follow

```
numpy.reshape(array_like,newshape,order='F')
```

3. Implement a matrix-free regularised least-squares solver for the Radon Transform

To determine the regularisation parameter you could use for instance the discrepancy principle from coursework 2. We note that in this case the parameter is typically larger than for the deconvolution.

You can define the a forward and adjoint operator as the following

```
def A(x,otherParameter):
    % implementation of forward propagation
    return A_x

def AT(y,otherParameter):
    % implementation of the back-propagation
    return AT_y
```

Then the forward-backward operator with input $\boldsymbol{f}$ and output $\boldsymbol{y}$ is shown as the following

```
y = AT(A(f,otherParameter),otherParameter)
```

Take care as usual with reshaping images to 1D vectors and back, for the purpose of calling a builtin Krylov solver. Also, you need to define `linearOperator` as the input of the Krylov solver similarly as what you did in coursework 2.

4. Write a Haar wavelet denoiser Wavelets are a so-called multi-scale decomposition of your image. For an image of size 128×128 you will get 7 scales, each scale will be have three "images" of gradually decreasing resolution. These images are the wavelet coefficients and represent the horizontal, vertical, and diagonal components.

We require Pywavelets Toolbox to perform the wavelet transform. To install the toolbox with `conda` run one of the following:

```
conda install -c conda-forge pywavelets
conda install -c conda-forge/label/gcc7 pywavelets
conda install -c conda-forge/label/cf201901 pywavelets
```

You can compute the coefficients for your image $\boldsymbol{f}$ by calling

```
# The function pywt.wavedec2 performs the 2D wavelet transform and returns a
# list of wavelet coefficients in the form [cA7,(cH7,cV7,cD7),(cH6,cV6,cD6),
# ...,(cH2,cV2,cD2),(cH1,cV1,cD1)], where cA, cH, cV, cD is the
# approximation, horizontal detail, vertical detail and diagonal detail
# coefficients respectively.

import pywt
coeffs = pywt.wavedec2(f,'haar',level = 7)
```

You can reconsruct the image by calling

```
f_rec = pywt.waverec2(coeffs,'haar')
```

For denoising write a threshold function, for instance as the following

```
tRange = range(maxRange)
tVal = 1
def thresholdFunction(coeffs,tRange,tVal,otherParameter):
    % implementation of the thresholding on wavelet coefficients
    return coeffsT
```

In the thresholding function you can make use of Pywavelet's function `pywt.threshold`. Then a denoised image is reconstructed by

```
f_denoise = pywt.waverec2(coeffsT,'haar')
```

In your thresholding function the `tRange` denotes the scales you want to threshold. Setting

```
maxRange = 7;
tRange = range(maxRange);
```

would mean that you loop through all scales and threshold on each scale. It might be a good idea to limit the range to the higher finer coefficients that mostly consist of the noise components. For instance `tRange` = `range`(`4`) would only threshold the first 4 scales. Note that the indexing start with 0 in Python.

How to choose a thresholding parameter `tVal`: The parameter should be chose such that it thresholds the noise components of your coefficients. This can be done either by visual examination of the coefficients or a more stable option would be to determine a value to threshold the lowest few (x) percent of parameters. To do that, collect all coefficients for all scales and directions into one large vector, sort them by absolute value and then determine the value for which x percent of the coefficients are smaller. This is your threshold value.

The `otherParameter` is not essential and was only added in case you want further personalise your thresholding function.

5. Iterative soft-thresholding for X-ray tomography To clarify the choice of step length λ. During the iterations you can keep the step length constant. To find a suitable choice of λ start with a small value, like 10^{-4} and see if the iterations are stable, i.e. they are not heavily oscillating. To check this you can just plot the reconstruction in each iteration and monitor the progress. If you found a stable choice for λ increase it as long as you still have a stable algorithm, this will speed up convergence, since larger λ will lead to faster convergence.