# Regression example - sensor fusion

# Regression example: Sensor Fusion EDX + HAADF

# Data Science in Electron Microscopy

Philipp Pelz

2024

https://github.com/ECLIPSE-Lab/WS24_DataScienceForEM

# Load Python modules

```
from scipy.sparse import spdiags
import matplotlib.pyplot as plt
import fusion_utils as utils
from tqdm import tqdm
import numpy as np
import h5py
# import sys
# raise RuntimeError(sys.executable)
```

# Load Chemical Maps

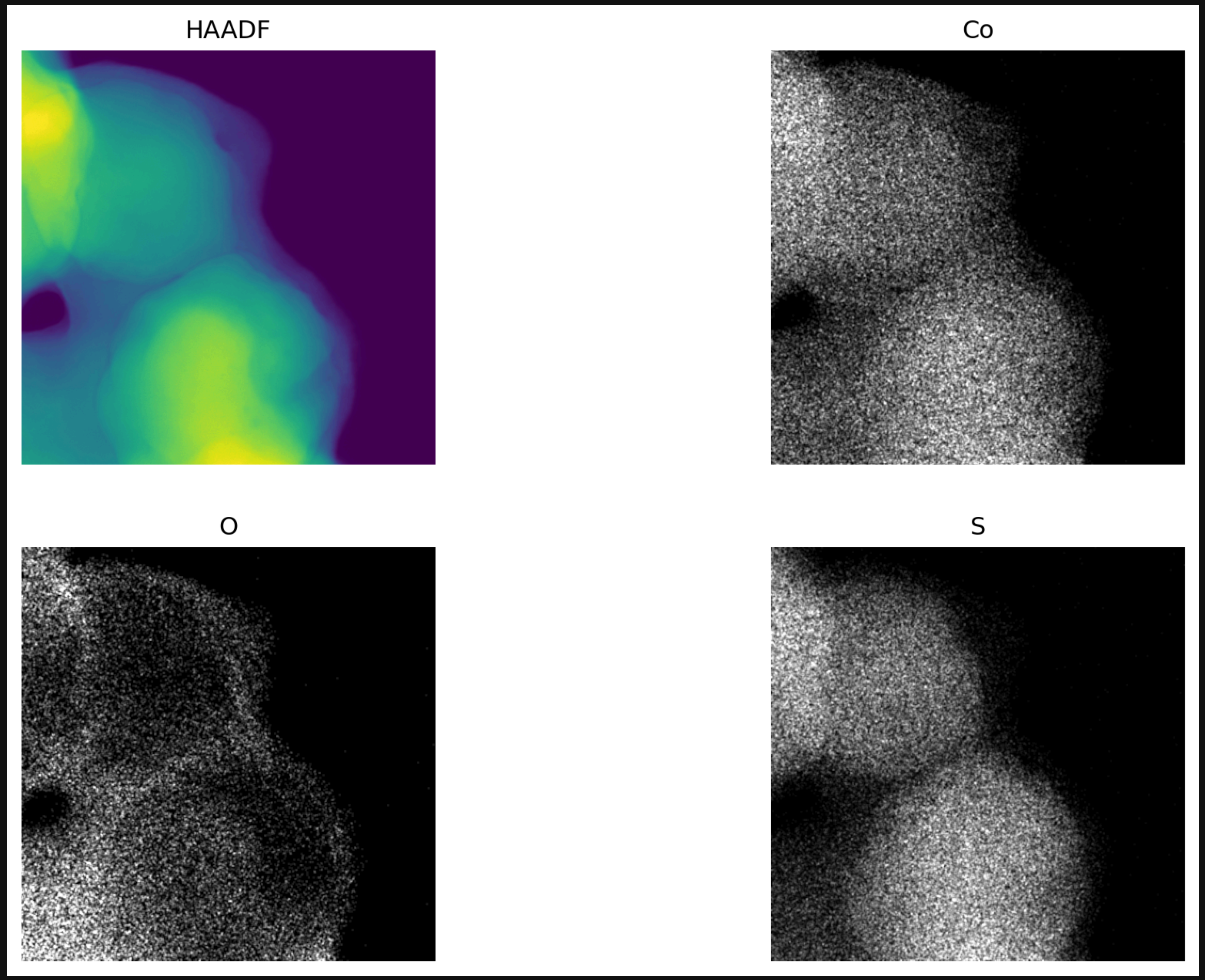
```
fname = 'CoSX_maps.h5'; mapNum = 'map7/'

# Parse Chemical Maps
elementList = ['Co', 'O', 'S']

# Load Raw Data and Reshape
file = h5py.File(fname, 'r')

print('Available EDX Maps: ', list(file))

xx = np.array([],dtype=np.float32)
for ee in elementList:

    # Read Chemical Map for Element "ee"
    edsMap = file[mapNum+ee][:,:]

    # Set Noise Floor to Zero and Normalize Chemical Maps
    edsMap -= np.min(edsMap); edsMap /= np.max(edsMap)
```

```
Available EDX Maps:  ['map4', 'map5', 'map6', 'map7', 'map8', 'map9']
```

# Parse Meta Data, Prepare Reconstruction and Display Raw Chemical Maps

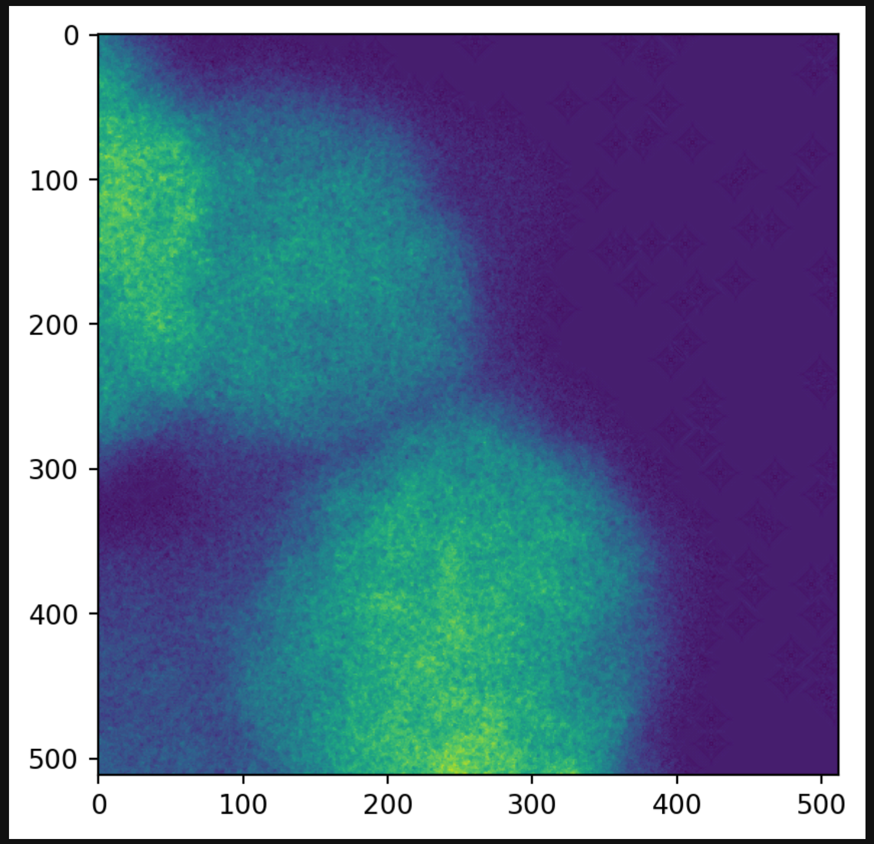
```python
# Image Dimensions
(nx, ny) = edsMap.shape; nPix = nx * ny
nz = len(elementList); lambdaHAADF = 1/nz

import torch
import kornia
import cv2
import numpy as np

import matplotlib.pyplot as plt
class TVDenoise(torch.nn.Module):
    def __init__(self, noisy_image, lambdaTV):
        super(TVDenoise, self).__init__()
        self.lambdaTV = lambdaTV
        self.l2_term = torch.nn.MSELoss(reduction='mean')
        self.regularization_term = kornia.losses.TotalVariation()
        # create the variable which will be optimized to produce the noise free image
        self.clean_image = torch.nn.Parameter(data=noisy_image.clone(), requires_grad=True)
        self.noisy_image = noisy_image
```

```python
# Show Raw Data
utils.plot_elemental_images(xx, b, elementList, nx, ny, 2,2)
```


HAADF
Co
O
S

# Plot Regularization

```python
regularize = True; ng = 15; lambdaTV = 0.1;
r, cost = reg(edsMap, lambdaTV, ng)

import matplotlib.pyplot as plt
fig, ax = plt.subplots()
ax.imshow(r)
plt.show()
```

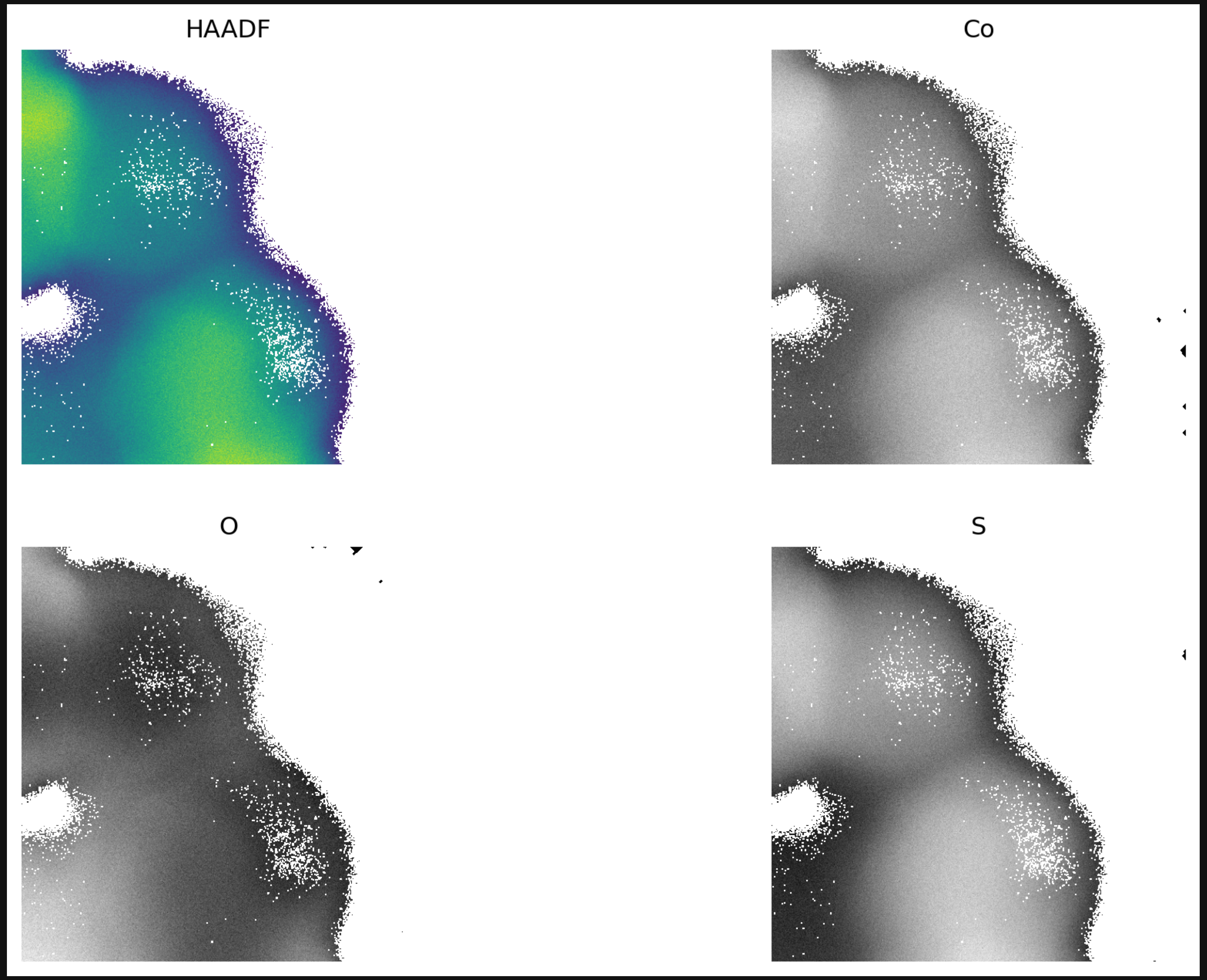


# Perform Multi-Modal Data Fusion

```
# Convergence Parameters
gamma = 1.6; lambdaEDS = 5e-6; nIter = 30; bkg = 1e-1

# TV Min Parameters
regularize = True; ng = 15; lambdaTV = 0.1;

# Auxiliary Functions
lsqFun = lambda inData : 0.5 * np.linalg.norm(A.dot(inData**gamma) - b) **2
poissonFun = lambda inData : np.sum(xx0 * np.log(inData + 1e-8) - inData)

# Main Loop
costHAADF = np.zeros(nIter,dtype=np.float32); costEDS = np.zeros(nIter, dtype=np.float32); costTV = np.zeros(nIter, dtype=np.float32);
for kk in tqdm(range(nIter)):

    # HAADF Update
    xx -=  gamma * spdiags(xx**(gamma - 1), [0], nz*nx*ny, nz*nx*ny) * lambdaHAADF * A.transpose() * (A.dot(xx**gamma) - b) \
            + lambdaEDS * (1 - xx0 / (xx + bkg))
    xx[xx<0] = 0
```

# Show Reconstructed Signal

```
utils.plot_elemental_images(xx,A.dot(xx**gamma),elementList,nx,ny,2,2)
```



# Display Cost Functions and Descent Parameters

```
utils.plot_convergence(costHAADF, lambdaHAADF, costEDS, lambdaEDS, costTV, lambdaTV)
```