

---

# **Quantitative Big Imaging - Bimodal experiments**

**Anders Kaestner**

**May 03, 2022**



# CONTENTS

<b>1 Bimodal experiments</b>	<b>3</b>
1.1 Literature / Useful References . . . . .	3
1.2 Previously on QBI . . . . .	4
1.3 Outline . . . . .	4
<b>2 Imaging modalities</b>	<b>5</b>
2.1 Some imaging experiments and their challenges . . . . .	5
2.2 Reasons to select an imaging modality? . . . . .	5
2.3 The aim of multimodal imaging . . . . .	7
2.4 Some considered modalities - Neutrons and X-rays . . . . .	8
2.5 Some considered modalities for medical imaging . . . . .	10
2.6 Some considered modalities - Grating interferometry . . . . .	10
2.7 Some considered modalities - Spectroscopic imaging . . . . .	11
2.8 Other modalities and dimensionality . . . . .	12
<b>3 Data and image fusion</b>	<b>13</b>
3.1 Definition . . . . .	13
3.2 Aim . . . . .	13
3.3 Fusion approaches - no golden recipe . . . . .	13
<b>4 Image fusion workflow</b>	<b>15</b>
4.1 Catastrophic fusion . . . . .	15
4.2 Image registration . . . . .	16
4.3 Registration considerations . . . . .	16
<b>5 Qualitative fusion: Registration and covisualization</b>	<b>19</b>
5.1 Let's load some test data . . . . .	20
5.2 Visualization techniques - Checker board . . . . .	20
5.3 Visualization techniques - Color channel mixing . . . . .	22
<b>6 Bimodal segmentation</b>	<b>23</b>
6.1 Histogram of single modality . . . . .	23
6.2 Individual histograms of two modalities . . . . .	23
6.3 Bivariate histogram . . . . .	24
6.4 Segmentation methods . . . . .	24
6.5 Previous segmentation methods . . . . .	26
6.6 Gaussian mixture model . . . . .	26
6.7 Classification distances . . . . .	31
6.8 Graphical presentation of different distances . . . . .	32
6.9 Segmentation by Euclidean distance . . . . .	33

<b>7 Bivariate estimation: Working with attenuation coefficients</b>	<b>35</b>
7.1 Beer-Lamberts law . . . . .	35
7.2 Equation system . . . . .	35
<b>8 Beyond multi modal experiments</b>	<b>37</b>
<b>9 Some software engineering</b>	<b>39</b>
9.1 Unit Testing . . . . .	39
9.2 Unit Testing - design . . . . .	39
9.3 Example . . . . .	40
9.4 Unit Testing: Example . . . . .	40
9.5 Unit Testing in Python . . . . .	41
9.6 Unit testing examples from Scikit Image . . . . .	41
9.7 Unit testing in python - DocTests . . . . .	42
9.8 Unit Testing with Jupyter . . . . .	42
9.9 Test Driven Programming . . . . .	45
9.10 Using repositories . . . . .	45
9.11 Repository workflows . . . . .	47
9.12 Continuous Integration . . . . .	47
<b>10 Summary</b>	<b>49</b>
10.1 Multiple modalities . . . . .	49
10.2 Software engineering . . . . .	49

This is the lecture notes for the 10th lecture of the Quantitative big imaging class given during the spring semester 2021 at ETH Zurich, Switzerland.



## BIMODAL EXPERIMENTS

```
%reload_ext autoreload
%autoreload 2
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import itertools
import numpy as np
import skimage.io as io
from scipy import linalg
import matplotlib as mpl
from sklearn import mixture
import pandas as pd

plt.rcParams["figure.figsize"] = (8, 8)
plt.rcParams["figure.dpi"] = 100
plt.rcParams["font.size"] = 14
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.style.use('default')
sns.set_style("whitegrid", {'axes.grid': False})
```

## 1.1 Literature / Useful References

### 1.1.1 Books

#### General:

- John C. Russ, “The Image Processing Handbook”,(Boca Raton, CRC Press)

#### Fusion specific:

- Mitchell, H.B., “Data Fusion: Concepts and Ideas”, Springer Verlag, 2012.
- Mitchel, H.B., “Image Fusion - Theories, Techniques and Applications”, Springer Verlag, 2010.
- T. Stathaki, “Image fusion”, Academic Press, 2008
- Goshtasby, A. Ardeshir, “Image Registration Principles, Tools and Methods”, Springer Verlag, 2012
- \*Xiao, G., Bavirisetti, D.P., Liu, G., Zhang, X., “Image Fusion”, Springer Verlag

### **Software engineering**

- Okken, B., [Python testing with pytest](#)
- Wuttke, J. et al.[Guidelines for collaborative software development](#), 2022.

## **1.2 Previously on QBI ...**

- Image Enhancement
- Highlighting the contrast of interest in images
- Minimizing Noise
- Understanding image histograms
- Automatic Methods
- Component Labeling
- Single Shape Analysis
- Complicated Shapes
- Dynamic Experiments
- Image registration
- Statistics
- Plotting

## **1.3 Outline**

- Motivation (Why and How?)
- Scientific Goals
- Image fusion
- Bivariate segmentation
- Software engineering for repeatability

## IMAGING MODALITIES

### 2.1 Some imaging experiments and their challenges

- Segmentation accuracy
- Estimate water content
- Segmentation accuracy
- Material classification
- Estimate water content
- Dimensional changes
- Penetration power
- Ambiguous readings

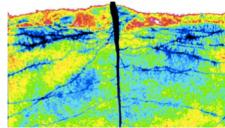


Fig. 2.1: In the soil the graylevels are often ambiguous.

### 2.2 Reasons to select an imaging modality?

Reasons to select or reject a specific imaging method

- Good transmission
- Good contrast
- Relevant features visible
- Materials can be identified
- Low transmission
- Low contrast
- Not all features visible
- Ambiguous response

Until now, we only collected image features from a single modality.



Fig. 2.2: Studies of the cultural heritage.

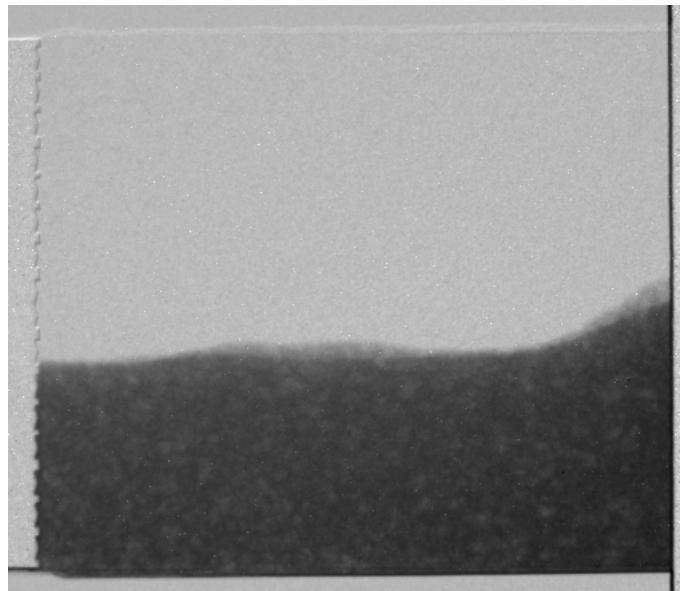


Fig. 2.3: Dimensional changes in porous media.

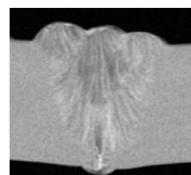


Fig. 2.4: Material science with material mixes.

## 2.3 The aim of multimodal imaging

### 2.3.1 Purpose of multi-modality

Match the advantages of each method against the disadvantages of the other methods to obtain more information than using each method individually.

1. Extend range of operation.
2. Extend spatial and temporal coverage.
3. Reduce uncertainty.
4. Increase reliability.
5. Robust system performance.



Fig. 2.5: The multispectral glasses from the movie 'National Treasure'.

## The players of an imaging experiment

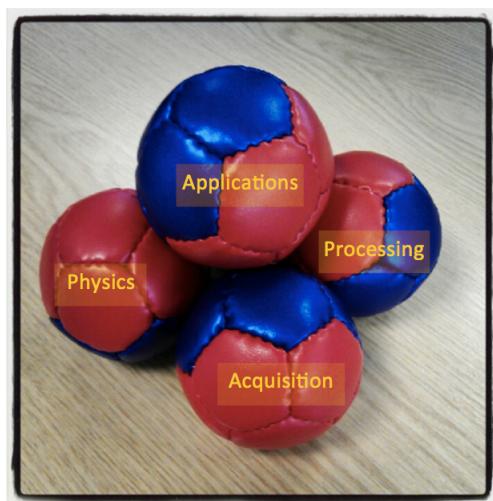


Fig. 2.6: An imaging experiment is only successful when all aspects are considered.

## 2.4 Some considered modalities - Neutrons and X-rays

In material science it often relevant to combine imaging with neutrons and X-rays. The reason is the complementarity between the two modalities. Simply put, neutrons are often sensitive to low-z materials while x-rays are more sensitive to high-z materials. Combining the two modalities is of particular interest when the sample is a mix of high and low-z materials.

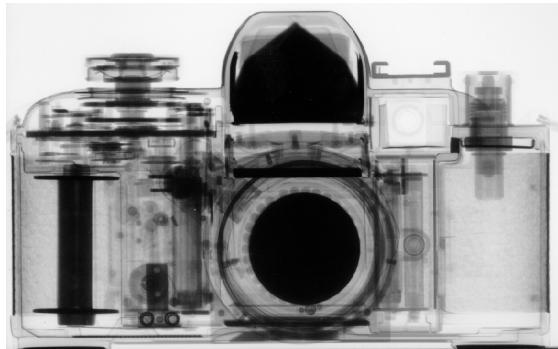


Fig. 2.7: Neutron radiography of a camera.

Group →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
↓ Period																		
1	H 3.44																He 0.02	
2	Li 3.30	Be 0.79											B 101.6	C 0.56	N 0.43	O 0.17	F 0.20	Ne 0.10
3	Na 0.09	Mg 0.15											Al 0.1	Si 0.11	P 0.12	S 0.06	Cl 1.33	Ar 0.03
4	K 0.06	Ca 0.08	Sc 2.00	Ti 0.60	V 0.72	Cr 0.54	Mn 1.21	Fe 1.19	Co 3.92	Ni 2.05	Cu 1.07	Zn 0.35	Ga 0.49	Ge 0.47	As 0.67	Se 0.73	Br 0.24	Kr 0.61
5	Rb 0.08	Sr 0.14	Y 0.27	Zr 0.29	Nb 0.40	Mo 0.52	Tc 1.76	Ru 0.58	Rh 10.88	Pd 0.78	Ag 4.04	Cd 115.1	In 7.58	Sn 0.21	Sb 0.30	Te 0.25	I 0.23	Xe 0.43
6	Cs 0.29	Ba 0.07		Hf 4.99	Ta 1.49	W 1.47	Re 6.85	Os 2.24	Ir 30.46	Pt 1.46	Au 6.23	Hg 16.21	Tl 0.47	Pb 0.38	Bi 0.27	Po -	At -	Rn -
7	Fr -	Ra 0.34		Rf -	Db -	Sg -	Bh -	Hs -	Mt -	Ds -	Rg -	Uub -	Uut -	Uuq -	Uup -	Uuh -	Uus -	Uuo -
Lanthanides		La 0.52	Ce 0.14	Pr 0.41	Nd 1.87	Pm 5.72	Sm 171.47	Eu 94.58	Gd 1479.0	Tb 0.93	Dy 32.42	Ho 2.25	Er 5.48	Tm 3.53	Yb 1.40	Lu 2.75		
Actinides		Ac -	Th 0.59	Pa 8.46	U 0.82	Np 9.80	Pu 50.20	Am 2.86	Cm -	Bk -	Cf -	Es -	Fm -	Md -	No -	Lr -		

Fig. 2.8: Attenuation coefficients for thermal neutrons.

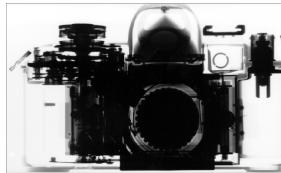


Fig. 2.9: X-ray radiography of a camera.

	Group →	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
↓ Period																			
1		H 0.02																He 0.02	
2		Li 0.06	Be 0.22																
3		Na 0.13	Mg 0.24																
4		K 0.14	Ca 0.26	Sc 0.48	Ti 0.73	V 1.04	Cr 1.29	Mn 1.32	Fe 1.57	Co 1.78	Ni 1.96	Cu 1.97	Zn 1.64	Ga 1.42	Ge 1.33	As 1.50	Se 1.23	Br 0.90	Kr 0.73
5		Rb 0.47	Sr 0.86	Y 1.61	Zr 2.47	Nb 3.43	Mo 4.29	Tc 5.06	Ru 5.71	Rh 6.08	Pd 6.13	Ag 5.67	Cd 4.84	In 4.31	Sn 3.98	Sb 4.28	Te 4.06	I 3.45	Xe 2.53
6		Cs 1.47	Ba 2.73		Hf 19.70	Ta 25.47	W 30.49	Re 34.47	Os 37.92	Ir 39.01	Pt 38.61	Au 35.94	Hg 25.88	Tl 23.23	Pb 22.81	Bi 20.28	Po 20.22	At -	Rn 9.77
7		Fr -	Ra 11.80		Rf -	Db -	Sg -	Bh -	Hs -	Mt -	Ds -	Rg -	Uub -	Uut -	Uuq -	Uup -	Uuh -	Uus -	Uuo -
Lanthanides			La 5.04	Ce 5.79	Pr 6.23	Nd 6.46	Pm 7.33	Sm 7.68	Eu 5.66	Gd 8.69	Tb 9.46	Dy 10.17	Ho 10.17	Er 11.70	Tm 12.49	Yb 9.32	Lu 14.07		
Actinides			Ac 24.47	Th 28.95	Pa 39.65	U 49.08	Np -	Pu -	Am -	Cm -	Bk -	Cf -	Es -	Fm -	Md -	No -	Lr -		

Fig. 2.10: Attenuation coefficients for 125keV X-rays.

## 2.5 Some considered modalities for medical imaging

Imaging is widely used in medical applications. There are also many different imaging modalities available, each revealing its own particular information.

The modalities also differ in the resolution that can be achieved. Therefore, it makes sense to combine the modalities to increase the understanding of provided information.

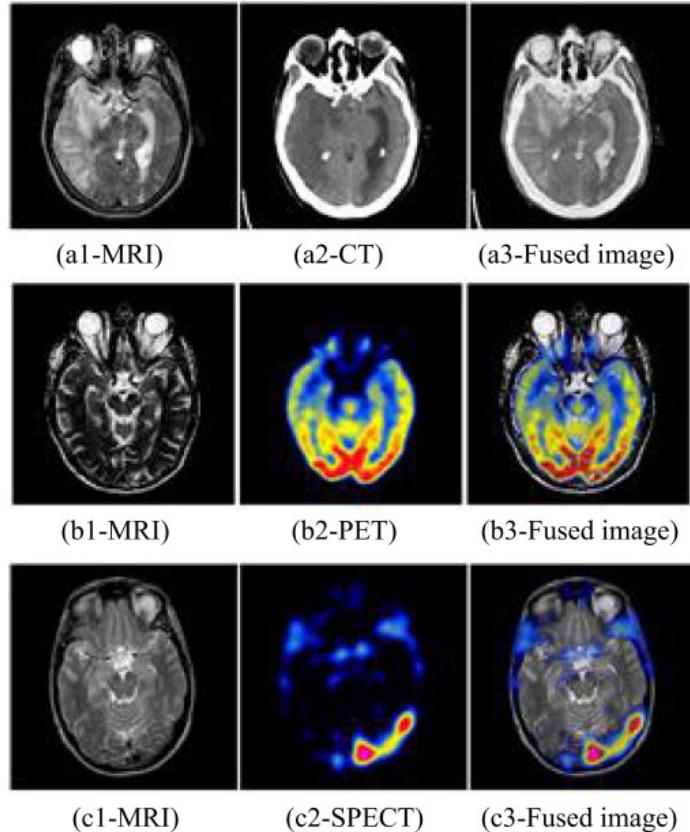


Fig. 2.11: Combining different medical imaging modalities.

Du et al. 2015

## 2.6 Some considered modalities - Grating interferometry

Grating interferometry is an imaging technique that exploits the wave property of the beam. This makes it possible to extract more information than the traditional transmission image. These are

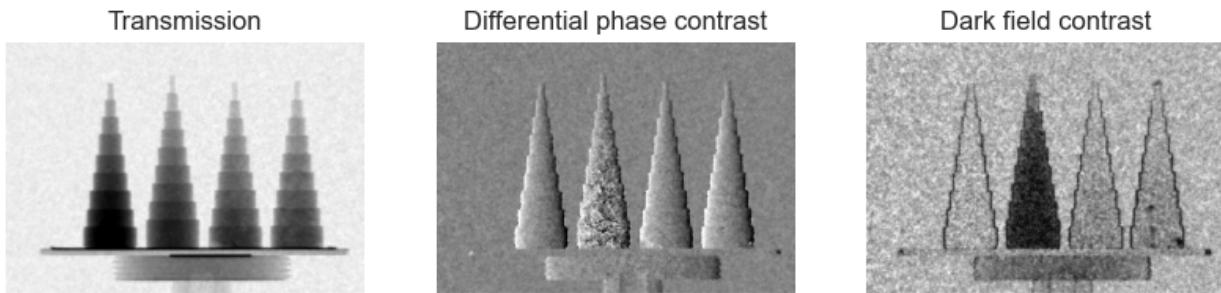
- The phase contrast - measures the phase shift of the beam to provide better contrast than the transmission in some cases.
- The dark field contrast - is related to the scattered beam and can probe clusters of sample features that are much smaller than the resolution of the imaging system.

```
fig,ax=plt.subplots(1,3,figsize=(10,5))
ax[0].imshow(io.imread("figures/nGI_TI.png")); ax[0].set_title('Transmission');
    ↵ ax[0].axis('off')
```

(continues on next page)

(continued from previous page)

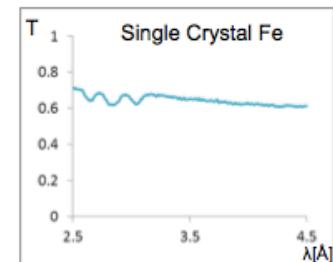
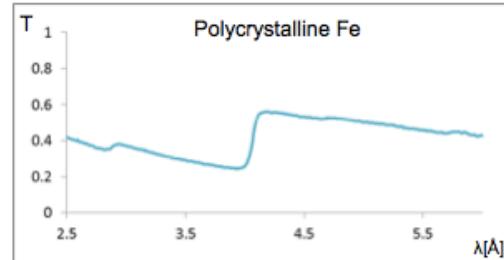
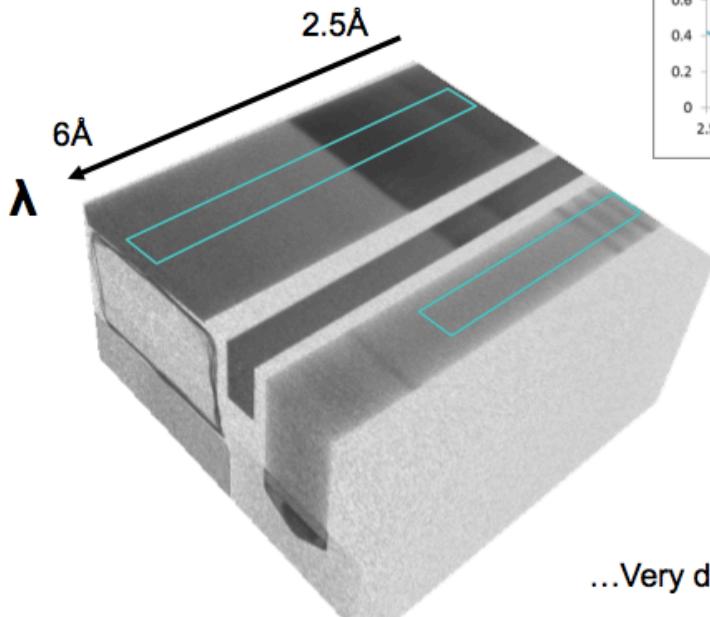
```
ax[1].imshow(io.imread("figures/nGI_DPC.png")); ax[1].set_title('Differential phase contrast')
ax[2].imshow(io.imread("figures/nGI_DFI.png")); ax[2].set_title('Dark field contrast')
ax[1].axis('off')
ax[2].axis('off');
```



- Data comparable on pixel level
- Non-linear relation between the variables.
- Improved estimation schemes using iterative process
- Physical interpretation/motivation to fuse?

## 2.7 Some considered modalities - Spectroscopic imaging

A closer look at the iron samples



...Very different wavelength dependence

Fig. 2.12: Neutron energy scan through a piece of iron.

- Material analysis
- Selector calibration

S. Peetermans

## 2.8 Other modalities and dimensionality

**The information can also be provided as few localized points**

- Single point measurements
- Surface information
- Single radiographs vs CT data

**to provide**

- Temperature
- Flowrate
- Pressure

## DATA AND IMAGE FUSION

### 3.1 Definition

The theory, techniques and tools which are used for

- combining sensor data, or data derived from sensory data,
- into a common representational format.

### 3.2 Aim

To improve the quality of the information, so that it is, in some sense, better than would be possible if the data sources were used individually.

Mitchell 2012

### 3.3 Fusion approaches - no golden recipe

#### 3.3.1 Fusion strategies

- **Multivariate fusion:** All data are combined using the same concept.
- **Augmented fusion:** Modalities have different functions in the fusion process.
- **Artifact reduction by fusion:** The second modality can be used to fill in the blanks.
- **Combination:** A single fusion method may not give the final result - combination

#### 3.3.2 Select strategy

The fusion strategy determined by:

- Sample composition
- Experiment objectives
- Condition of the data

## Levels of fusion

Input	Output	Description
Data	Data	Input data is smoothed/filtered/segmented
Data	Feature	The pixels are reduced to features using multiple sources.
Feature	Feature	Input features are reduced in number, or new features are generated by fusing input features.
Feature	Decision	Input features are fused together to give output decision.
Decision	Decision	Multiple input decisions are fused together to give a final output decision. e.g. Random forest

## IMAGE FUSION WORKFLOW

Image fusion is the process to combine images from different modalities with the aim to enhance the information compared to the images individually. This process has several steps and the fusion can be done on several levels of abstraction.

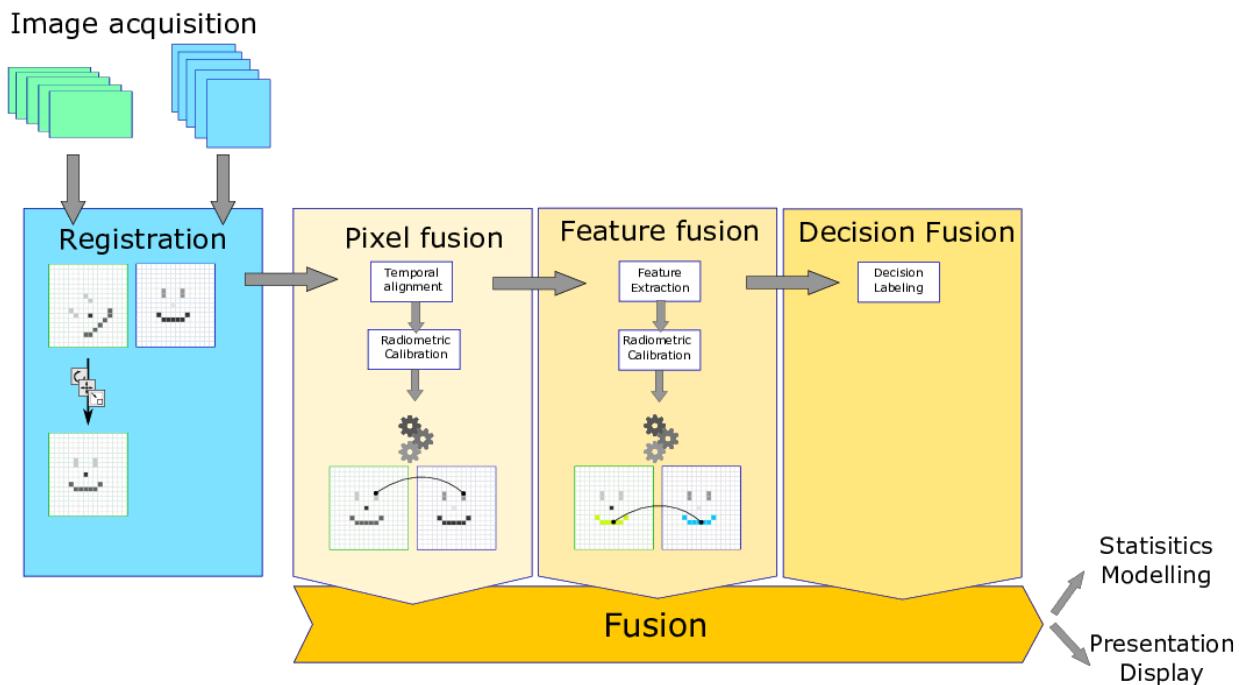


Fig. 4.1: Flow chart showing how image fusion can be done

Mitchel, 2010, Goshtasby, 2012

### 4.1 Catastrophic fusion

#### 4.1.1 Definition

The combination perform worse than the individual modalities... don't fuse because you can!

Catastrophic fusion can be caused by:

- Selection of the wrong variables.

- Too complex combination.
- Sensor information canceling each other.



Fig. 4.2: More chefs don't always mean better soup, the same applies to data fusion. Choose your source combination and fusion methods carefully.

## 4.2 Image registration

From last weeks lecture: A series of affine transformations to bring images on the same grid.

### 4.2.1 The process

## 4.3 Registration considerations

Registration is an optimization problem with many local minima.

### 4.3.1 Manual or guided registration

- Perform the full transformation manually
- Identify landmarks, points, lines, planes
- Provide a coarse preregistration

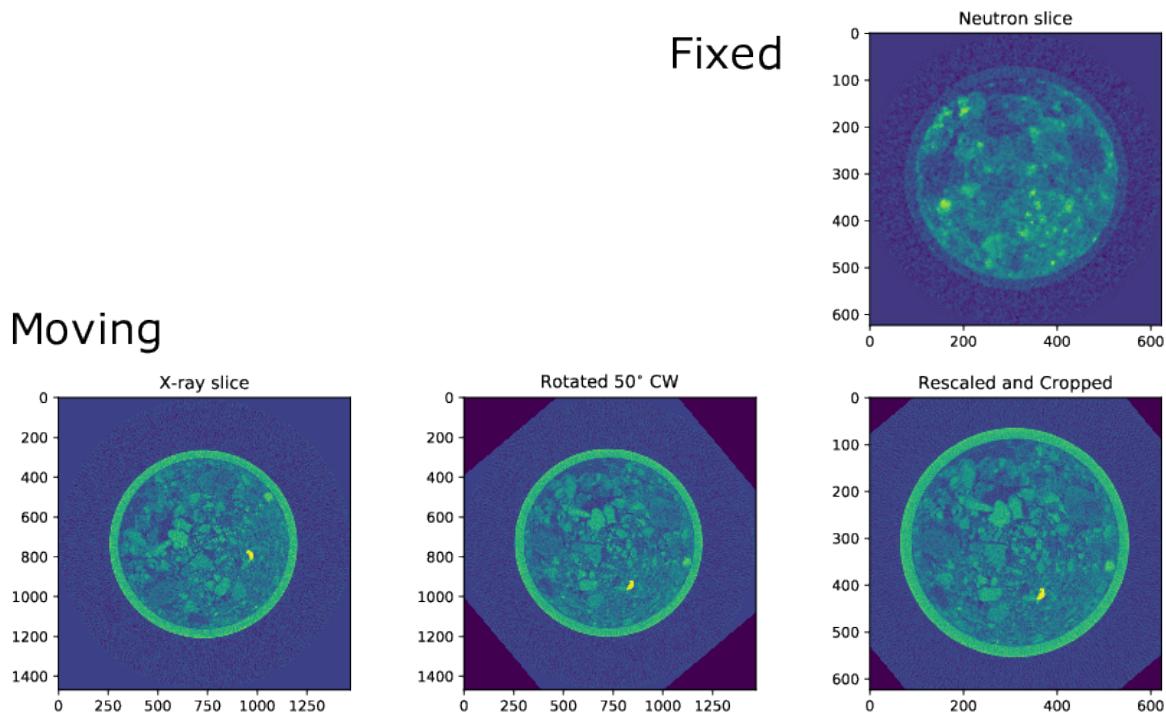


Fig. 4.3: Registration optimizes the scale, rotation, and position of an image compared to a fixed reference.

### 4.3.2 Automatic registration

- Iterative process
- Metrics
- Multi-modality loose common landmarks

Goshtasby, 2012



## QUALITATIVE FUSION: REGISTRATION AND COVISUALIZATION

Use e.g. VG Studio or 3DSlicer to

- Register data sets
- Interactive guided segmentation of the separate data sets.



Fig. 5.1: The sword from lake Zug as seen with neutrons.

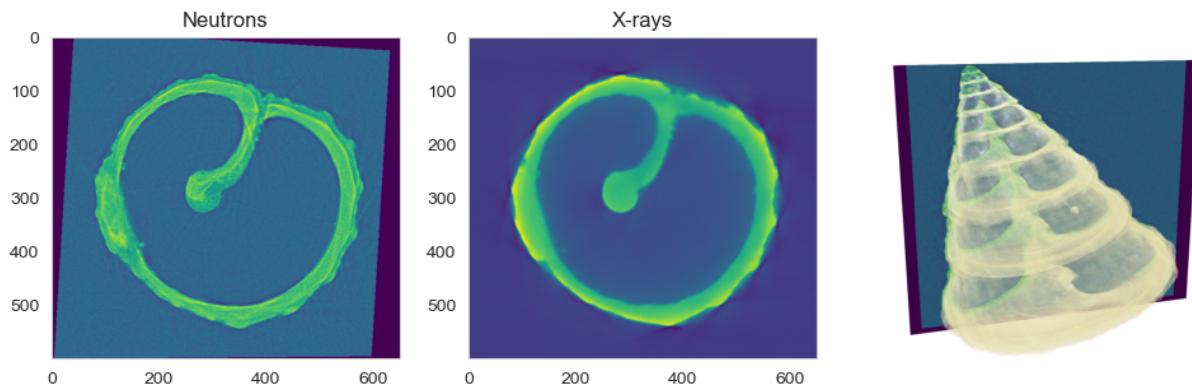


Fig. 5.2: The sword from lake Zug as seen with X-rays.

mannes2015\_NXCultHer

## 5.1 Let's load some test data

```
imgA=np.load('data/shellN.npy')
imgB=np.load('data/shellX.npy')
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(12, 5))
ax1.imshow(imgA, cmap='viridis'), ax1.set_title('Neutrons')
ax2.imshow(imgB, cmap='viridis'), ax2.set_title('X-rays');
ax3.imshow(plt.imread('figures/snailshellNeutron.png')); ax3.axis('off');
```



## 5.2 Visualization techniques - Checker board

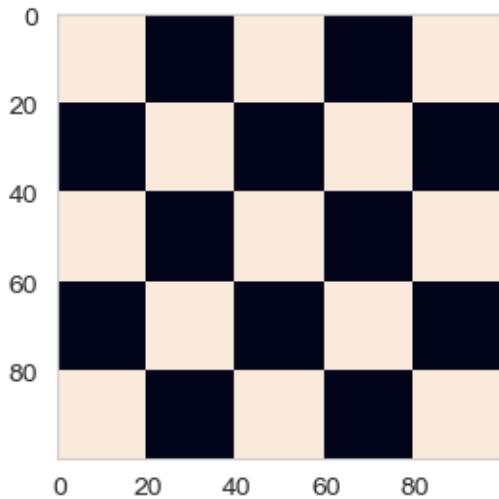
```
def checkerBoard(imgA,imgB,tiles=10) :
    if imgA.shape != imgB.shape :
        raise Exception('Image have different sizes')

    dims      = imgA.shape
    tileSize = (dims[0]//tiles,dims[1]//tiles)

    mix = np.zeros(dims)

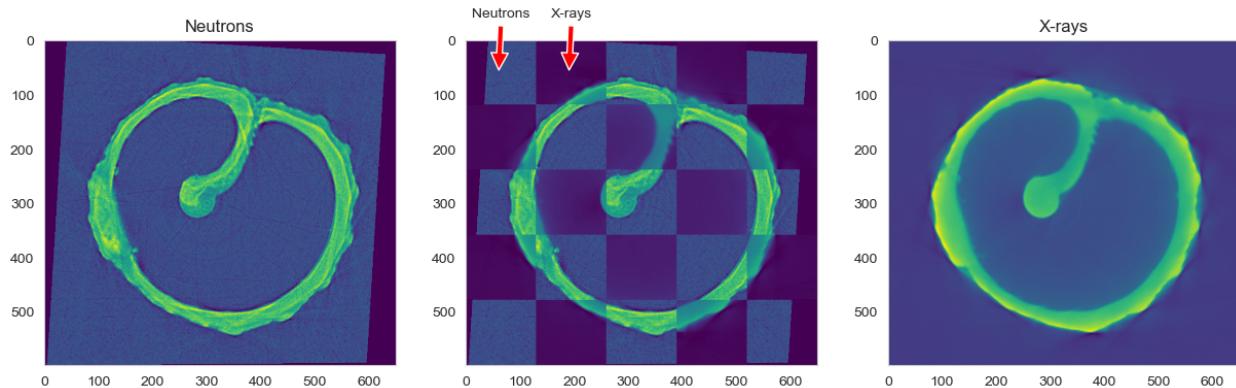
    for r in np.arange(0,tiles) :
        for c in np.arange(0,tiles) :
            if (c+r) % 2 :
                mix[(r*tileSize[0]):((r+1)*tileSize[0]),
←(c*tileSize[1]):((c+1)*tileSize[1])] = imgB[(r*tileSize[0]):((r+1)*tileSize[0]),
←(c*tileSize[1]):((c+1)*tileSize[1])]
            else :
                mix[(r*tileSize[0]):((r+1)*tileSize[0]),
←(c*tileSize[1]):((c+1)*tileSize[1])] = imgA[(r*tileSize[0]):((r+1)*tileSize[0]),
←(c*tileSize[1]):((c+1)*tileSize[1])]

    return mix
plt.figure(figsize=(3,3))
plt.imshow(checkerBoard(np.ones((100,100)),np.zeros((100,100)),tiles=5),interpolation=
←'none');
```



### 5.2.1 Try checker board with images

```
fig, (ax1, ax2, ax3)=plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(imgA,cmap='viridis',vmin=10000,vmax=60000), ax1.set_title('Neutrons')
ax2.imshow(checkerBoard(imgA,imgB,tiles=5),cmap='viridis',vmin=10000,vmax=60000);
ax2.annotate('Neutrons',
             xy=(60, 60), xycoords='data',
             xytext=(0.1, 1.1), textcoords='axes fraction',
             arrowprops=dict(facecolor='red', shrink=0.05),
             horizontalalignment='center', verticalalignment='top')
ax2.annotate('X-rays',
             xy=(190, 60), xycoords='data',
             xytext=(0.3, 1.1), textcoords='axes fraction',
             arrowprops=dict(facecolor='red', shrink=0.05),
             horizontalalignment='center', verticalalignment='top')
ax3.imshow(imgB,cmap='viridis'), ax3.set_title('X-rays');
```



## 5.3 Visualization techniques - Color channel mixing

With two or more modalities, we can visualize the mix using the RGB color channels:

$$\begin{cases} R & \text{modality}_A \\ G & \text{modality}_B \\ B & \frac{\text{modality}_A + \text{modality}_B}{2} \end{cases}$$

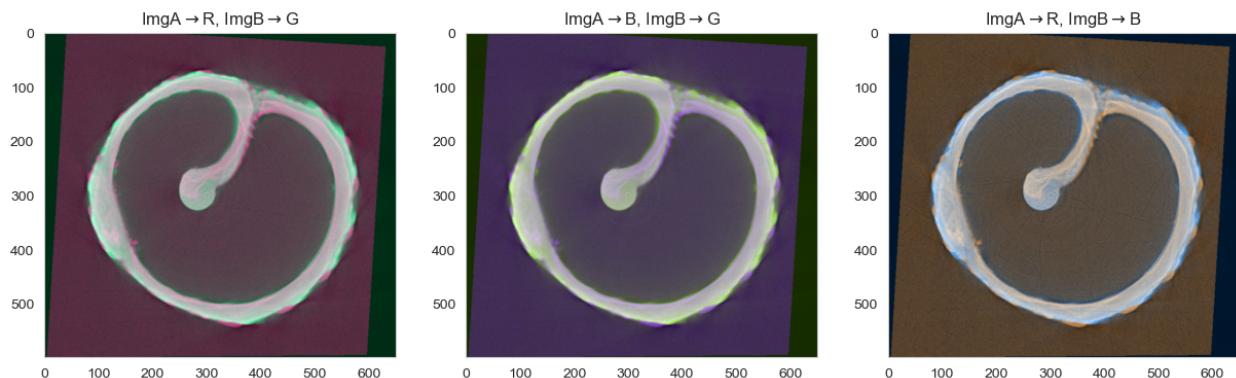
some intensity scaling may be needed for best result.

```
def channelMix(imgA,imgB, order=(0,1,2)) :
    imgAN=(imgA-imgA.min())/(imgA.max()-imgA.min())
    imgBN=(imgB-imgB.min())/(imgB.max()-imgB.min())

    rgb=np.zeros((imgA.shape[0],imgA.shape[1],3));
    rgb[:, :, order[0]]=imgAN
    rgb[:, :, order[1]]=imgBN
    rgb[:, :, order[2]]=0.5*(imgAN+imgBN)

    return rgb
```

```
fig, (ax1,ax2,ax3)=plt.subplots(1,3,figsize=(15,6))
ax1.imshow(channelMix(imgA,imgB,order=(0,1,2))), ax1.set_title(r'ImgA$\rightarrow$R, $\rightarrow$ImgB$\rightarrow$G');
ax2.imshow(channelMix(imgA,imgB,order=(2,1,0))), ax2.set_title(r'ImgA$\rightarrow$B, $\rightarrow$ImgB$\rightarrow$G');
ax3.imshow(channelMix(imgA,imgB,order=(0,2,1))), ax3.set_title(r'ImgA$\rightarrow$R, $\rightarrow$ImgB$\rightarrow$B');
```



## BIMODAL SEGMENTATION

### 6.1 Histogram of single modality

When you do experiments with a single modality, you only obtain a single histogram. The modes of the histogram may merge into a single mode if the SNR is too low to separate the feature classes. This leads to a large amount of miss-classifications. The blue region between the histogram peaks in [Figure 6.1](#) represents the area of ambiguous decisions.

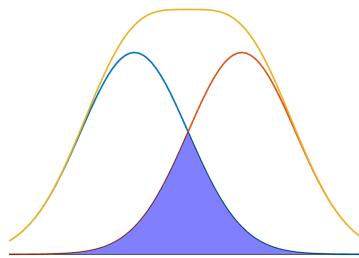


Fig. 6.1: Histogram of two classes using modality A.

### 6.2 Individual histograms of two modalities

Now we may conclude that the first modality we looked at doesn't provide sufficient information to make a reliable segmentation. Therefore, we go to a second modality. Unfortunately, this modality has the same low class separability as you can see in [Figure 6.2](#). This time the two classes have different responses and the histogram modes have swapped compared to [Figure 6.1](#).

So the conclusion is that we don't get much closer to our segmented image using these modalities individually.

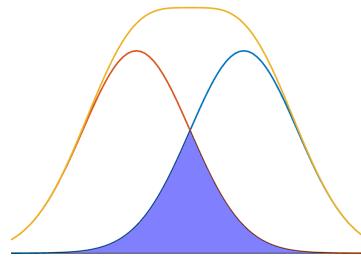


Fig. 6.2: Histogram of two classes using modality  $B$ .

### 6.3 Bivariate histogram

Now, if we start combining the two modalities, we start seeing the benefit of using more than one modality. The bivariate histogram, which we already have looked at in previous lectures is a great way to visualize how two variables depend on each other.

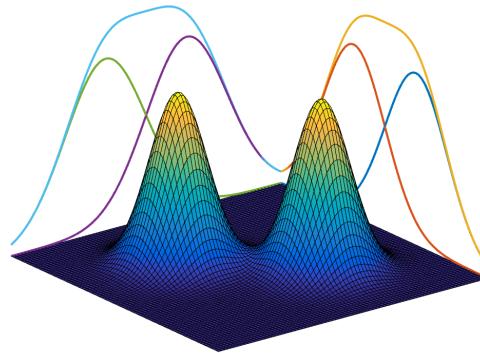


Fig. 6.3: A bivariate histogram of modalities  $A$  and  $B$ .

In the histogram show in Figure 6.3, we see that there is a clear separation between class  $A$  and  $B$  that could be easily thresholded.

#### 6.3.1 Example: Roots in soil

#### 6.3.2 Bivariate histogram of roots

### 6.4 Segmentation methods

#### 6.4.1 Data

- Images from  $M$  modalities  $f_1, \dots, f_M$

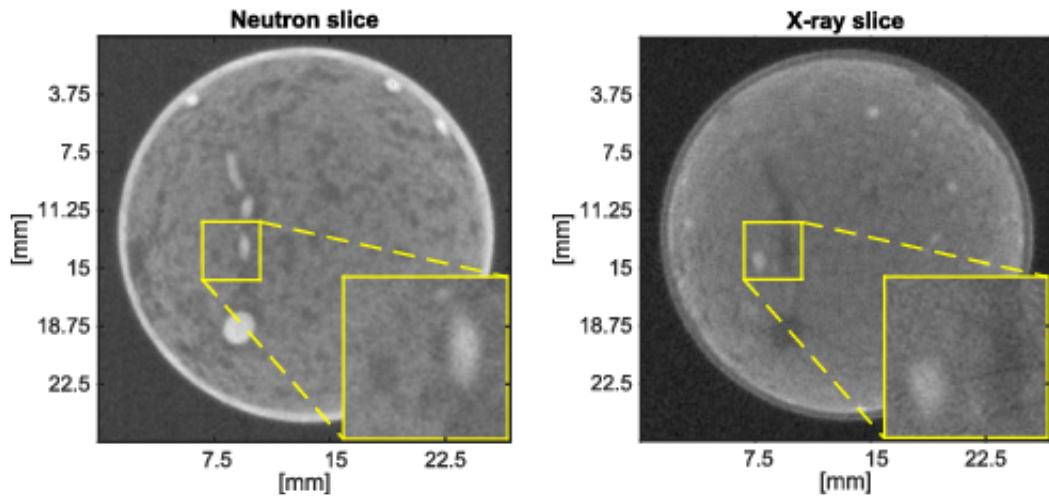


Fig. 6.4: Tomography slices of a soil sample with roots.

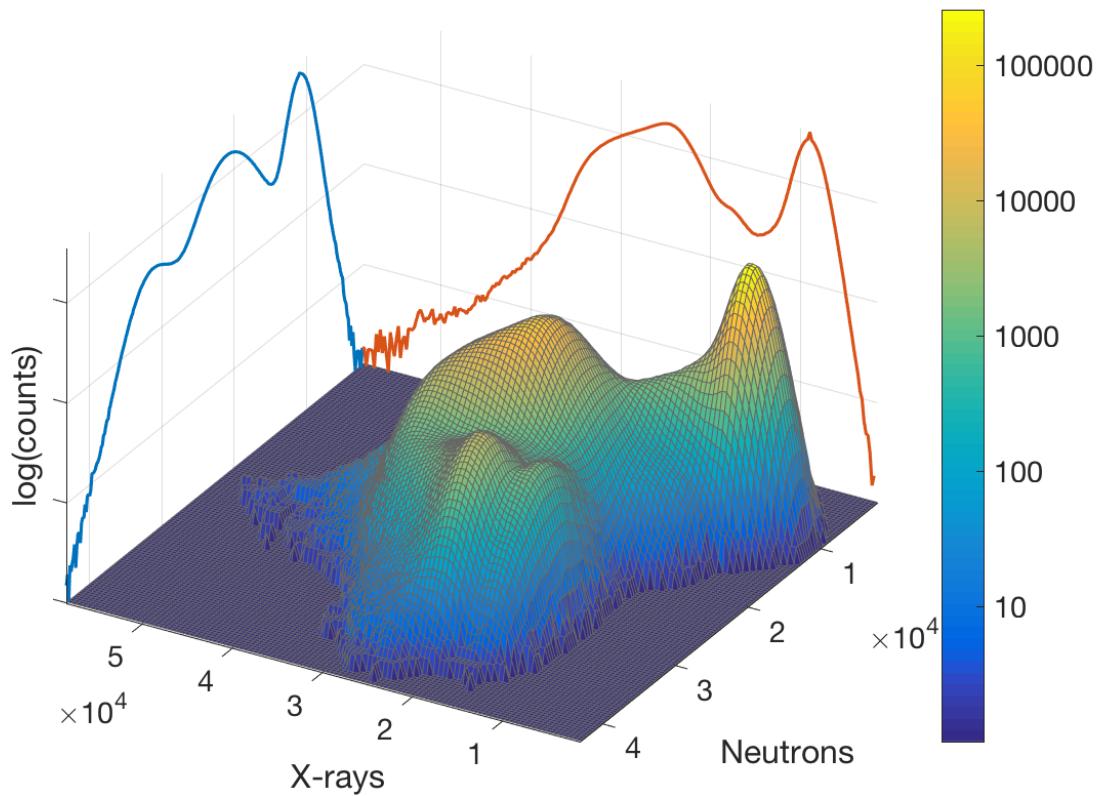


Fig. 6.5: Bivariate histogram of the root images in Figure 6.4

- Registered
- Artifact corrected

### 6.4.2 Classes

$$\text{The } N \text{ classes are described by: } \begin{cases} \mathcal{H}_1 : p(\mathbf{x}_1, \Sigma_1) \\ \mathcal{H}_2 : p(\mathbf{x}_2, \Sigma_2) \\ \vdots \\ \mathcal{H}_N : p(\mathbf{x}_N, \Sigma_N) \end{cases}$$

Duda, Hart, and Stork, 2001

## 6.5 Previous segmentation methods

In this class we have already looked into many different ways to perform the segmentation on images. These are methods that are well suited for segmenting bi- or multivariate data:

- k-means
- k-NN
- Regression
- Neural networks

## 6.6 Gaussian mixture model

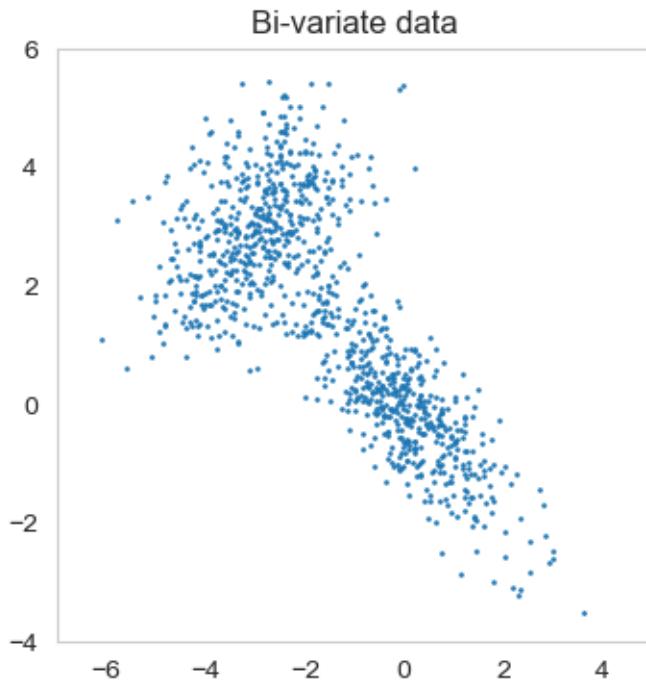
With Gaussian distribution we can describe the bivariate histogram using:  $p(\theta) = \sum_1^N \phi_i \mathcal{N}(\mathbf{x}_i, \Sigma_i)$

- $\mu_i$  - vector with averages for each class.
- $\Sigma_i$  - covariance matrix for each class.
- $\phi_i$  - mixing coefficient.

```
# Number of samples per component
n_samples = 500

# Generate random sample, two components
np.random.seed(0)
C1 = np.array([[1, -0.5], [-0.5, 1]])
C2 = np.array([[1, 0.25], [0.25, 1]])
X = np.r_[np.dot(np.random.randn(n_samples, 2), C1), np.dot(np.random.randn(n_samples,
    ↵ 2), C2) + np.array([-3, 3])]

plt.figure(figsize=[4, 4])
plt.scatter(X[:, 0], X[:, 1], 0.8)
plt.xlim(-7., 5.), plt.ylim(-4., 6.)
plt.title('Bi-variate data');
```



```

def plot_results(X, Y_, means, covariances, title, ax, showShape=True,
                 showCenter=False):
    color_iter = itertools.cycle(['navy', 'c', 'cornflowerblue', 'gold',
                                  'darkorange'])

    for i, (mean, covar, color) in enumerate(zip(
            means, covariances, color_iter)):
        v, w = linalg.eigh(covar)
        v = 2. * np.sqrt(2.) * np.sqrt(v)
        u = w[0] / linalg.norm(w[0])
        # as the DP will not use every component it has access to
        # unless it needs it, we shouldn't plot the redundant
        # components.
        if not np.any(Y_ == i):
            continue
        ax.scatter(X[Y_ == i, 0], X[Y_ == i, 1], .8, color=color)

        # Plot an ellipse to show the Gaussian component
        if showShape:
            angle = np.arctan(u[1] / u[0])
            angle = 180. * angle / np.pi # convert to degrees
            ell = mpl.patches.Ellipse(mean, v[0], v[1], 180. + angle, color=color)
            ell.set_clip_box(ax.bbox)
            ell.set_alpha(0.5)
            ax.add_artist(ell)

        if showCenter:
            ax.plot(mean[0], mean[1], 'ro')

    ax.set_xlim(-7., 5.)
    ax.set_ylim(-4., 6.)
    ax.set_title(title)

```

### 6.6.1 Gaussian mixture model fitting

```
fig, axes = plt.subplots(2,2, figsize=(8,8))
# Fit a Gaussian mixture with EM using five components
for i,ax in zip(np.arange(0, len(axes.ravel())), axes.ravel()):
    gmm = mixture.GaussianMixture(n_components=i+1, covariance_type='full').fit(X)

    plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
                 title='Gaussian Mixture ({} classes)'.format(i+1), ax=ax)
```

```
-----
AttributeError                                     Traceback (most recent call last)
Input In [10], in <cell line: 3>()
      2 # Fit a Gaussian mixture with EM using five components
      3 for i,ax in zip(np.arange(0, len(axes.ravel())), axes.ravel()):
--> 4     gmm = mixture.GaussianMixture(n_components=i+1, covariance_type='full').
      5     fit(X)
      6     plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
      7                 title='Gaussian Mixture ({} classes)'.format(i+1), ax=ax)

File ~/miniforge3/lib/python3.9/site-packages/sklearn/mixture/_base.py:198, in
    BaseMixture.fit(self, X, y)
   172 def fit(self, X, y=None):
   173     """Estimate model parameters with the EM algorithm.
   174
   175     The method fits the model ``n_init`` times and sets the parameters with
(...).
   196     The fitted mixture.
   197     """
--> 198     self._fit_predict(X, y)
   199     return self

File ~/miniforge3/lib/python3.9/site-packages/sklearn/mixture/_base.py:251, in
    BaseMixture._fit_predict(self, X, y)
   248 self._print_verbose_msg_init_beg(init)
   250 if do_init:
--> 251     self._initialize_parameters(X, random_state)
   253 lower_bound = -np.inf if do_init else self.lower_bound_
   255 for n_iter in range(1, self.max_iter + 1):

File ~/miniforge3/lib/python3.9/site-packages/sklearn/mixture/_base.py:143, in
    BaseMixture._initialize_parameters(self, X, random_state)
   140 if self.init_params == "kmeans":
   141     resp = np.zeros((n_samples, self.n_components))
   142     label = (
--> 143         cluster.KMeans(
   144             n_clusters=self.n_components, n_init=1, random_state=random_state
   145         )
   146         .fit(X)
   147         .labels_
   148     )
   149     resp[np.arange(n_samples), label] = 1
150 elif self.init_params == "random":
```

```
File ~/miniforge3/lib/python3.9/site-packages/sklearn/cluster/_kmeans.py:1171, in
    KMeans.fit(self, X, y, sample_weight)
1169 if self._algorithm == "full":
```

(continues on next page)

(continued from previous page)

```

1170     kmeans_single = _kmeans_single_lloyd
-> 1171     self._check_mkl_vcomp(X, X.shape[0])
1172 else:
1173     kmeans_single = _kmeans_single_elkan

File ~/miniforge3/lib/python3.9/site-packages/sklearn/cluster/_kmeans.py:1028, in __
-> KMeans._check_mkl_vcomp(self, X, n_samples)
1026     active_threads = int(np.ceil(n_samples / CHUNK_SIZE))
1027 if active_threads < self._n_threads:
-> 1028     modules = threadpool_info()
1029     has_vcomp = "vcomp" in [module["prefix"] for module in modules]
1030     has_mkl = ("mkl", "intel") in [
1031         (module["internal_api"], module.get("threading_layer", None))
1032         for module in modules
1033     ]

File ~/miniforge3/lib/python3.9/site-packages/sklearn/utils/fixedes.py:325, in __
-> threadpool_info()
323     return controller.info()
324 else:
--> 325     return threadpoolctl.threadpool_info()

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:124, in threadpool_
-> info()
107 @_format_docstring(USER_APIS=list(_ALL_USER_APIS),
108                      INTERNAL_APIS=_ALL_INTERNAL_APIS)
109 def threadpool_info():
110     """Return the maximal number of threads for each detected library.
111
112     Return a list with all the supported modules that have been found. Each
113     (...).
114     In addition, each module may contain internal_api specific entries.
115     """
116     return _ThreadpoolInfo(user_api=_ALL_USER_APIS).todict()

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:340, in __
-> ThreadpoolInfo.__init__(self, user_api, prefixes, modules)
337     self.user_api = [] if user_api is None else user_api
339     self.modules = []
--> 340     self._load_modules()
341     self._warn_if_incompatible_openmp()
342 else:

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:371, in __
-> ThreadpoolInfo._load_modules(self)
369     """Loop through loaded libraries and store supported ones"""
370 if sys.platform == "darwin":
--> 371     self._find_modules_with_dyld()
372 elif sys.platform == "win32":
373     self._find_modules_with_enum_process_module_ex()

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:428, in __
-> ThreadpoolInfo._find_modules_with_dyld(self)
425     filepath = filepath.decode("utf-8")
426     # Store the module if it is supported and selected
--> 428     self._make_module_from_path(filepath)

```

(continues on next page)

(continued from previous page)

```

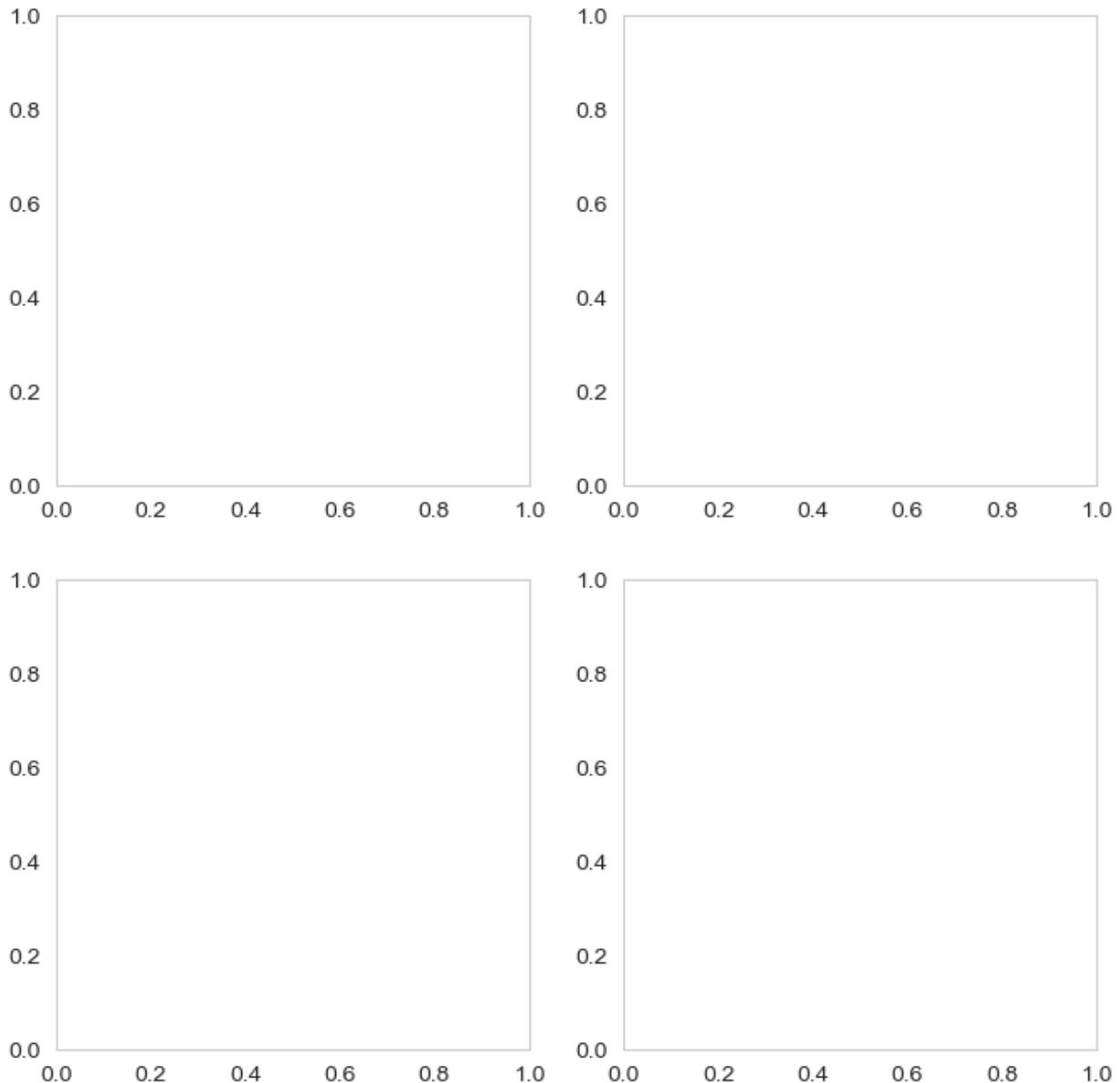
File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:515, in __
  ↪_ThreadpoolInfo._make_module_from_path(self, filepath)
  513 if prefix in self.prefixes or user_api in self.user_api:
  514     module_class = globals()[module_class]
--> 515     module = module_class(filepath, prefix, user_api, internal_api)
  516     self.modules.append(module)

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:606, in _Module.__init__
  ↪_(self, filepath, prefix, user_api, internal_api)
  604 self.internal_api = internal_api
  605 self._dynlib = ctypes.CDLL(filepath, mode=_RTLD_NOLOAD)
--> 606 self.version = self.get_version()
  607 self.num_threads = self.get_num_threads()
  608 self._get_extra_info()

File ~/miniforge3/lib/python3.9/site-packages/threadpoolctl.py:646, in __
  ↪_OpenBLASModule.get_version(self)
  643 get_config = getattr(self._dynlib, "openblas_get_config",
  644                         lambda: None)
  645 get_config.restype = ctypes.c_char_p
--> 646 config = get_config().split()
  647 if config[0] == b"OpenBLAS":
  648     return config[1].decode("utf-8")

AttributeError: 'NoneType' object has no attribute 'split'

```



## 6.7 Classification distances

For a set of multivariate normal distributions  $p_i = \mathcal{N}(\mu_i, \Sigma_i)$

We can find the nearest neighbor class using the following distances

### 6.7.1 Euclidean

Distance between two points  $D_E = \sqrt{(x - \mu_1)^T \cdot (x - \mu_1)}$

### 6.7.2 Mahanalubis

Distance from class  $i$  to point  $x$   $D_M = \sqrt{(x - \mu_i)^T \Sigma_i^{-1} (x - \mu_i)}$

### 6.7.3 Bhattacharia

Distance between two classes  $D_B = \frac{1}{8} (\mu_1 - \mu_2)^T \Sigma (\mu_1 - \mu_2) + \frac{1}{2} \ln \left( \frac{|\Sigma|}{\sqrt{|\Sigma_1| \cdot |\Sigma_2|}} \right)$   $\Sigma = \frac{\Sigma_1 + \Sigma_2}{2}$

Assign the point to the class with shortest distance.

## 6.8 Graphical presentation of different distances

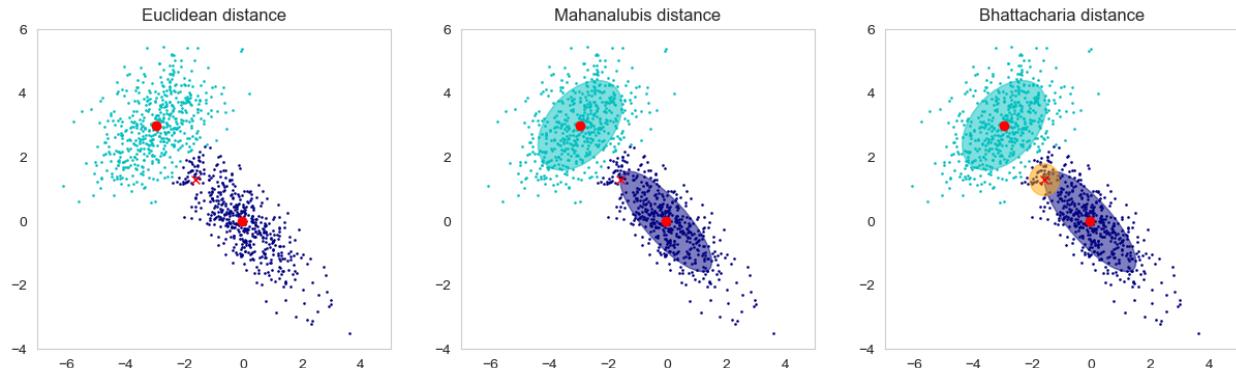
```

gmm = mixture.GaussianMixture(n_components=2, covariance_type='full').fit(X)
m=[-1.6,1.3]
fig, (ax1,ax2,ax3)=plt.subplots(1,3,figsize=(15,4))

plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
             'Euclidean distance',ax1, showShape=False,showCenter=True)
ax1.plot(-1.6,1.3,'rx')
plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
             'Mahanalubis distance'.format(2),ax2, showCenter=True)
ax2.plot(-1.6,1.3,'rx')

plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
             'Bhattacharia distance'.format(2),ax3, showCenter=True)
v=1
ell = mpl.patches.Ellipse(m, v, v, 0, color='orange')
ell.set_clip_box(ax3.bbox)
ell.set_alpha(0.5)
ax3.add_artist(ell)
ax3.plot(m[0],m[1],'rx');

```



## 6.9 Segmentation by Euclidean distance

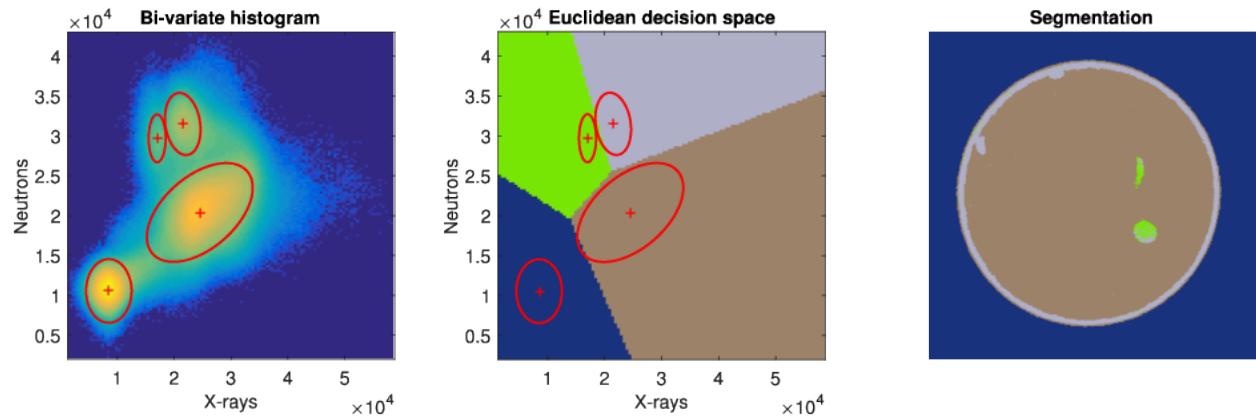


Fig. 6.6: Segmenting the root image in

kaestner2016\_itmnrnx



## BIVARIATE ESTIMATION: WORKING WITH ATTENUATION COEFFICIENTS

### 7.1 Beer-Lambers law

$$I = I_0 e^{-\frac{\rho}{A} N_A \sigma x}$$

- $\rho$  Material density
- $A$  Atomic weight
- $\sigma$  microscopic cross section
  - Probability of interaction
  - modality dependent
- $x$  propagation length

### 7.2 Equation system

$$\begin{aligned}\sum_{i=1}^N \Sigma_i x_i &= q_N \\ \sum_{i=1}^N \mu_i x_i &= q_X\end{aligned}$$

- attn coeff known  $\rightarrow$  estimate lengths.
- More pixels  $\rightarrow$  more materials.



---

**CHAPTER  
EIGHT**

---

## **BEYOND MULTI MODAL EXPERIMENTS**

Many bimodal experiments are done separately.

There many reasons for this, two are:

- Limited resources
- Scanners at different locations

This is often the case in medical imaging where the hospitals have different dedicated machines for each modality. It is also not always that the patient is scan using all relevant modalities at the same time, but different modalities are used at different stages of the therapy.

This is also a common approach in materials science and ex situ imaging. The home laboratory may own their own X-ray CT scanner but they need to go to a large scale facility to obtain more information with further modalities.

Next steps:

- Dynamic experiments

Last week we looked into the topic of dynamic experiments. The use of bimodal imaging is also very relevant in dynamic experiments. The observed samples and processes often change shape when you introduce a liquid, apply a pressure, etc. These shape changes are often more visible in one modality than the other. Ideally, you will have a system where one modality is sensitive to dimensional changes while the other is sensitive the changes in mixing ratios and other process related parameters.

- Combined setups

Combined setups allow simultaneous acquisition using two modalities. This has the advantage that you can perform dynamic experiments.

[Figure 6.3](#) show a setup for bimodal neutron and X-ray imaging. The system has two difference beam geometries neutrons uses parallel beam and X-rays a cone beam. The beams are also at oblique angles and mostly also resulting in different resolutions, there it is a first requirement that the resulting images are registered before any analysis can be performed.

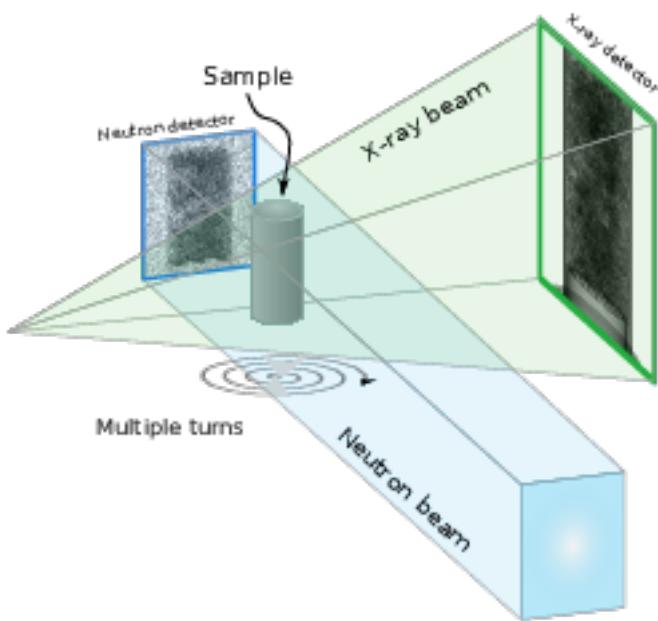


Fig. 8.1: The outline of a bimodal imaging system for neutrons and X-rays.

## SOME SOFTWARE ENGINEERING

Repeatable workflows require some software engineering skills

- Unit testing
- Working with repositories
- Continuous integration

### 9.1 Unit Testing

In computer programming, unit testing is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine if they are fit for use.

- Intuitively, one can view a unit as the smallest testable part of an application
- Unit testing is possible with every language
- Most (Java, C++, Matlab, R, Python) have built in support for automated testing and reporting

Computational science: ... Error

### 9.2 Unit Testing - design

The first requirement for unit testing to work well is to have your code divided up into small independent parts (functions)

#### 9.2.1 What to test?

- Each part can then be tested independently (unit testing)
  - If the tests are well done, units can be changed and tested independently
  - Makes upgrading or expanding tools *easy*
- The entire path can be tested (integration testing)
  - Catches mistakes in integration or *glue*

### 9.2.2 How to test

- The *happy path* - check what it is supposed to do
- To **provoke** your code - provide data that will fail execution

### 9.2.3 Test data

Ideally with realistic but simulated test data

## 9.3 Example

Given the following function `function vxCnt=countVoxs(inImage)`

We can write the following tests:

### 9.3.1 `testEmpty2d`

```
assert countVoxs(zeros(3,3)) == 0
```

### 9.3.2 `testEmpty3d`

```
assert countVoxs(zeros(3,3,3)) == 0
```

### 9.3.3 `testDiag2d`

```
assert countVoxs(eye(3)) == 3
```

## 9.4 Unit Testing: Example

Given the following function `function shapeTable=shapeAnalysis(inImage)`

We should decompose the function into sub-components with single tasks:

```
from graphviz import Digraph
dot = Digraph()
dot.edge('shapeAnalysis(inImage)', 'componentLabel(inImage)'), dot.edge(
    'shapeAnalysis(inImage)', 'analyzeObject(inObject)')
dot.edge('analyzeObject(inObject)', 'countVoxs(inObject)'), dot.edge(
    'analyzeObject(inObject)', 'calculateCOV(inObject)')
dot.edge('analyzeObject(inObject)', 'calcShapeT(covMat)'), dot.edge(
    'analyzeObject(inObject)', 'calcOrientation(shapeT)')
dot.edge('analyzeObject(inObject)', 'calcAnisotropy(shapeT)')
dot
```

```
<graphviz.dot.Digraph at 0x7f8c70f043d0>
```

## 9.5 Unit Testing in Python

### 9.5.1 PyTest

Packages like PyTest are

- well suited for larger projects
  - you make a set of specific tests for each module
  - run each time the project is updated.

## 9.6 Unit testing examples from Scikit Image

<https://github.com/scikit-image/scikit-image/tree/master/skimage>

- Test Watershed
  - Test Connected Components

(continues on next page)

(continued from previous page)

```
[-1, -1, -1, -1, -1, -1, -1]])
error = diff(expected, out)
assert error < eps
```

## 9.7 Unit testing in python - DocTests

Keep the tests in the code itself:

```
def apply_hysteresis_threshold(image, low, high):
    """Apply hysteresis thresholding to `image`.
    This algorithm finds regions where `image` is greater than `high`
    OR `image` is greater than `low` *and* that region is connected to
    a region greater than `high`.

    Parameters
    -----
    image : array, shape (M, [ N, ..., P])
        Grayscale input image.
    low : float, or array of same shape as `image`
        Lower threshold.
    high : float, or array of same shape as `image`
        Higher threshold.

    Returns
    -----
    thresholded : array of bool, same shape as `image`
        Array in which `True` indicates the locations where `image`
        was above the hysteresis threshold.

    Examples
    -----
    >>> image = np.array([1, 2, 3, 2, 1, 2, 1, 3, 2])
    >>> apply_hysteresis_threshold(image, 1.5, 2.5).astype(int)
    array([0, 1, 1, 1, 0, 0, 0, 1, 1])

    References
    -----
    .. [1] J. Canny. A computational approach to edge detection.
        IEEE Transactions on Pattern Analysis and Machine Intelligence.
        1986; vol. 8, pp.679-698.
        DOI: 10.1109/TPAMI.1986.4767851
    """
    low = np.clip(low, a_min=None, a_max=high) # ensure low always below high
    mask_low = image > low
    mask_high = image > high
```

## 9.8 Unit Testing with Jupyter

Working primarily in notebooks makes regular testing more difficult but not impossible. If we employ a few simple tricks we can use doctesting seamlessly inside of Jupyter. We can make what in python is called an annotation to setup this code.

```
import doctest
import copy
import functools
```

(continues on next page)

(continued from previous page)

```
def autotest(func):
    globs = copy.copy(globals())
    globs.update({func.__name__: func})
    doctest.run_docstring_examples(
        func, globs, verbose=True, name=func.__name__)
    return func
```

### 9.8.1 A very simple test

This test will be used in a jupyter notebook.

It is implemented as a DocTest. The function is supposed to return five added to the input value. The test is implemented for the specific case with the input ‘5’ and we expect the function to return 10.

```
@autotest
def add_5(x):
    """
    Function adds 5
    >>> add_5(5)
    10
    """
    return x+5
```

```
Finding tests in add_5
Trying:
add_5(5)
Expecting:
10
ok
```

The test returned 10 and DocTest concludes that the test passed with an ‘ok’. This is a very simple function that already assumes you enter a number, but what would happen if we call the function with a string or a complicated object of some kind? The function should be extended with checks if the correct data type is provided, which in turn would require further tests to verify that the functionality under these conditions.

### 9.8.2 Testing an image processing algorithm

Numerical algorithms are often hard to check with unit tests. In particular, when noise is added. You can however always test the basic functionality. Below we have an example that tests whether the label function does what it is supposed to do:

1. A single object - labels 0 and 1
2. Break the object in two - labels 0-2

```
from skimage.measure import label
import numpy as np
@autotest
def simple_label(x):
    """
    Label an image
    >>> test_img = np.eye(3)
    >>> test_img
    array([[1.,  0.,  0.],
```

(continues on next page)

(continued from previous page)

```
[0., 1., 0.],  
 [0., 0., 1.]])  
>>> simple_label(test_img)  
array([[1, 0, 0],  
       [0, 1, 0],  
       [0, 0, 1]])  
>>> test_img[1,1] = 0  
>>> simple_label(test_img)  
array([[1, 0, 0],  
       [0, 0, 0],  
       [0, 0, 2]])  
"""  
return label(x)
```

Finding tests in simple\_label

Trying:

```
    test_img = np.eye(3)
```

Expecting nothing

ok

Trying:

```
    test_img
```

Expecting:

```
    array([[1., 0., 0.],  
          [0., 1., 0.],  
          [0., 0., 1.]])
```

ok

Trying:

```
    simple_label(test_img)
```

Expecting:

```
    array([[1, 0, 0],  
          [0, 1, 0],  
          [0, 0, 1]])
```

ok

Trying:

```
    test_img[1,1] = 0
```

Expecting nothing

ok

Trying:

```
    simple_label(test_img)
```

Expecting:

```
    array([[1, 0, 0],  
          [0, 0, 0],  
          [0, 0, 2]])
```

ok

### 9.8.3 Unit Testing Matlab

<https://www.mathworks.com/help/matlab/matlab-unit-test-framework.html>

## 9.9 Test Driven Programming

Test Driven programming is a style or approach to programming where

1. *the tests are written before the functional code.*
2. The tests are like very concrete specifications.
3. It is easy to estimate project progress since you can automatically see how many of the tests have been passed.

You and your collaborators are clear on the utility of the system.

1. shapeAnalysis must give an anisotropy of 0 when we input a sphere
2. shapeAnalysis must give the center of volume within 0.5 pixels
3. shapeAnalysis must run on a 1000x1000 image in 30 seconds

## 9.10 Using repositories

A repository is (simply put) a backup system tailored to the needs of software development.

- Synchronizes multiple versions
- Manages development branches
- Located on a centralized server

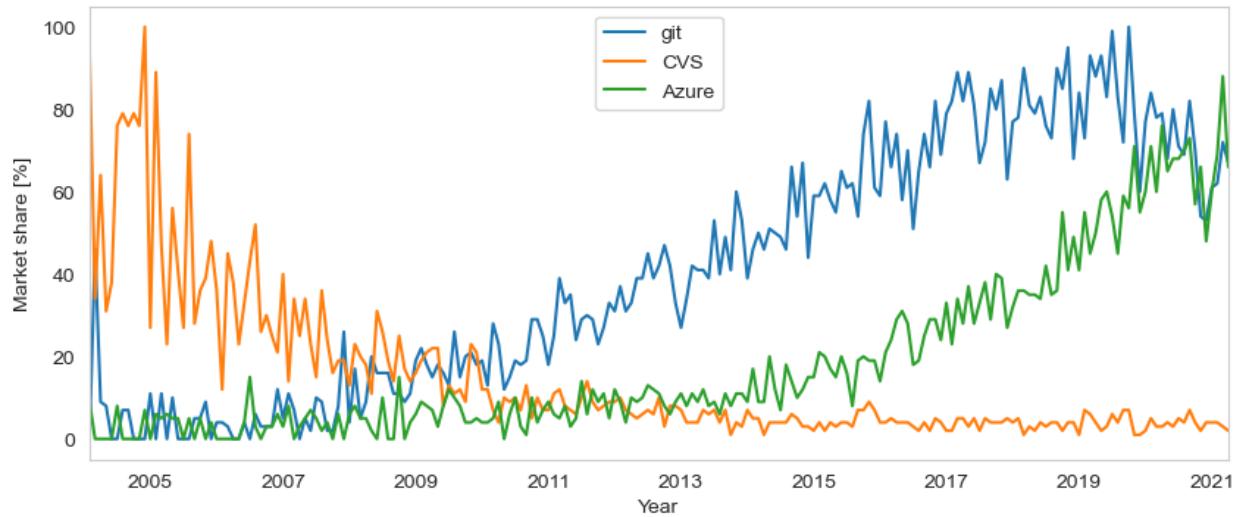
### 9.10.1 Why should I use a repository?

- It makes it easy to get back to earlier versions
- It is good for reproducibility
- Makes bug tracking easier
- Easier for a team to work on the same code (without disturbing each other)

### 9.10.2 Different repository frameworks

- Git
- Azure
- Subversion
- CVS

```
fig,ax=plt.subplots(1,1,figsize=(10,4))
repodf = pd.read_csv('data/repository_popularity.csv',parse_dates=['Date'],index_col=[  
    'Date'])
repodf.plot(ax=ax);
ax.set_xlabel('Year');
ax.set_ylabel('Market share [%]');
```



### 9.10.3 Git servers

There are many servers available. These services include

- Repository
- Issue tracking
- Project management (Kanban tables etc.)

#### Public

- GitHub
- GitLab
- BitBucket

#### Local

- ETH GitLab

## 9.11 Repository workflows

- Single branch (like a backup server with comments)
- Multiple branches (Recommended)

The recommended workflow when you work with a repository is to generously work with branches for each new sub task of the development. This make it easier to maintain a stable main branch that other people in the team can rely on to perform the tasks correctly and without crashing. The work in the branches are merged into the main after synchronization and review with a team mate.

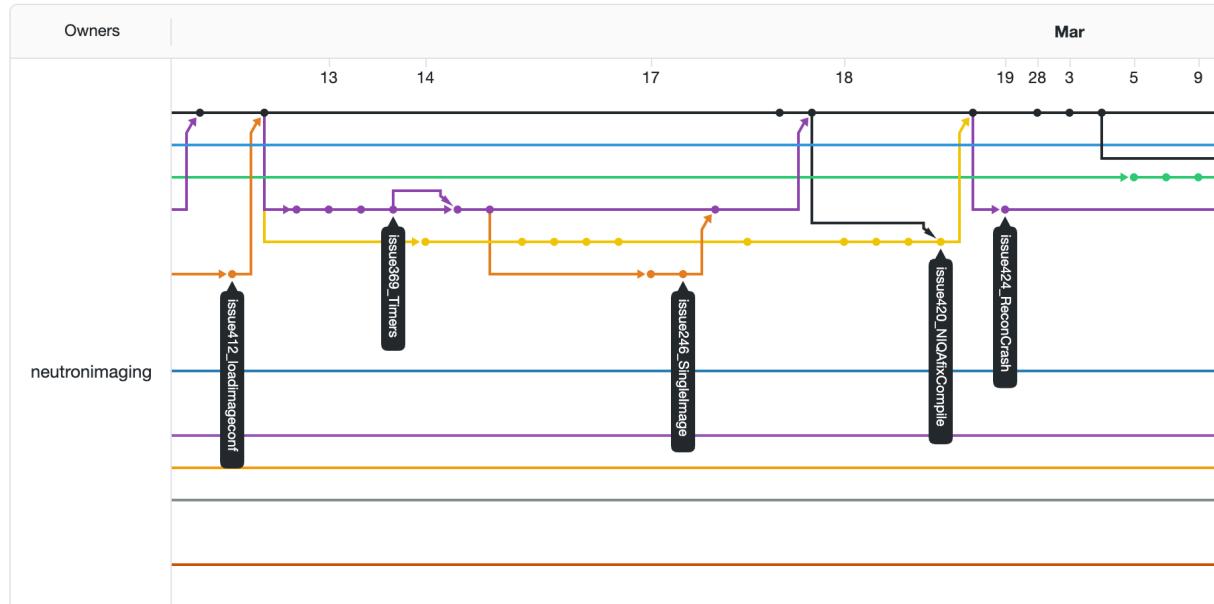


Fig. 9.1: A snapshot of the branch network from a git repository.

[GitHub tutorials](#)

## 9.12 Continuous Integration

Continuous integration is the process of running tests automatically everytime changes are made.

This is possible to setup inside of many IDEs and is offered as a commercial service from companies like CircleCI and Travis.

We use them for the QBI course to make sure all of the code in the slides are correct.

Projects like scikit-image use them to ensure changes that are made do not break existing code without requiring manual checks



**SUMMARY**

## 10.1 Multiple modalities

- Add more information to improve the conclusions
- Add component in the analysis and visualization
- Data fusion can be done on different levels of abstraction.

## 10.2 Software engineering

- Unit testing
- Working with repositories
- Continuous integration