
Quantitative Big Imaging - Bimodal experiments

Anders Kaestner

May 01, 2025

CONTENTS

0.1	Bimodal experiments	1
0.2	Imaging modalities	2
0.3	Data and image fusion	9
0.4	Image fusion workflow	13
0.5	Qualitative fusion: Registration and covisualization	16
0.6	Bimodal segmentation	20
0.7	Bivariate estimation: Working with attenuation coefficients	29
0.8	Multi modal experiments	30
0.9	Some software engineering	30
0.10	Summary	45

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.

0.1 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})
```

0.1.1 Literature / Useful References

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. Ardeshtir, “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.

0.1.2 Previously on QBI ...

- Image Enhancement
 - Highlighting the contrast of interest in images
 - Minimizing Noise
- Automatic segmentation methods
- Component Labeling
- Single Shape Analysis
- Complicated Shapes
- Dynamic Experiments
- Image registration
- Statistics

0.1.3 Outline

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

0.2 Imaging modalities

0.2.1 Some imaging experiments and their challenges

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

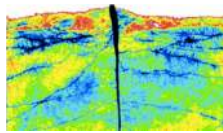


Fig. 1: In the soil the graylevels are often ambiguous.



Fig. 2: Studies of the cultural heritage.

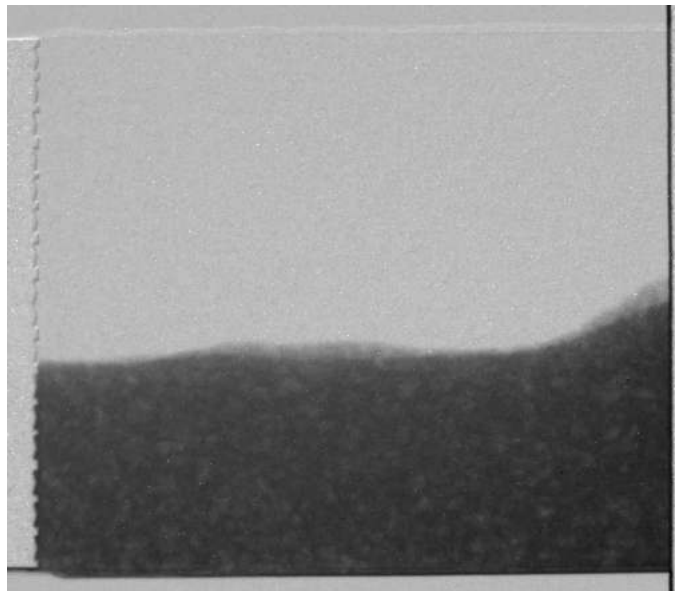


Fig. 3: Dimensional changes in porous media.

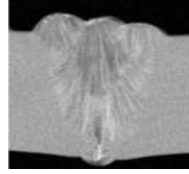


Fig. 4: Material science with material mixes.

0.2.2 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.

The players of an imaging experiment

An imaging experiment involves several components that have to be tuned in order to obtain reliable results. The application comes in the first place for the experimentalist. It has been identified that spatial information is needed to answer a question and support a hypothesis. The choice of imaging modality depends on the material composition for two reasons.

1. Does the emitted signal provide relevant responses from the samples?
2. Does the sample “survive” the irradiation?

Therefore, it is important to understand the physics of the chosen modality. The next step is to select an acquisition techniques that provides images in reasonable time, i.e. if the exposure times matches the time constants of the observed sample and available experiment time. Also, are the spatial resolution and field of view sufficient to represent object features in the images.

The final step of an imaging experiment is the processing and analysis which can be more or less complicated as we have seen in the previous lectures. The processing workflow has some parts in common between different modalities, while other parts are very modality specific.

In many cases, a single modality is sufficient to answer the questions posed in the experiment. There are however cases some aspects are weakly represented in the images. This would in a first step indicate that there is a better modality for the task. When this modality on the other hand has different weaknesses that makes it suboptimal too, we to decide how to proceed;

1. Select the fittest modality and deal with its weaknesses and handle the difficulties occurring in the analysis.
2. Use images from both modalities at the cost of additional experiment complexity and higher dimensional data analysis with the complication this brings.

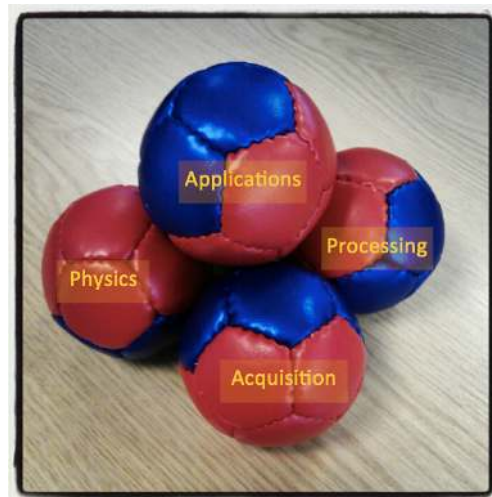


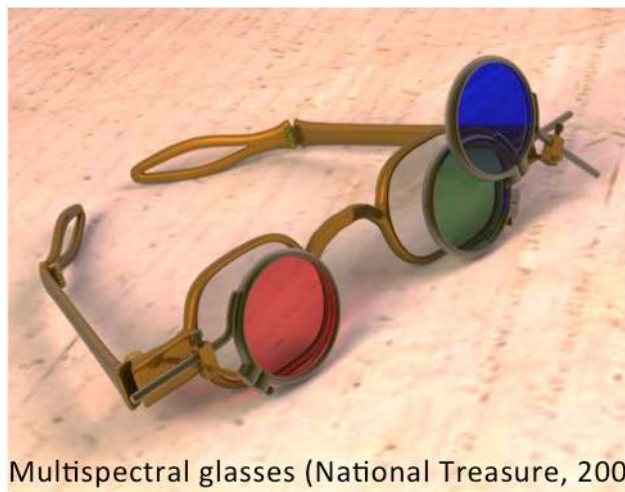
Fig. 5: An imaging experiment is only successful when all aspects are considered.

0.2.3 The aim of multimodal imaging

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.



Multispectral glasses (National Treasure, 2004)

Fig. 6: The multispectral glasses from the movie 'National Treasure'.

0.2.4 Some considered modalities for materials science

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.

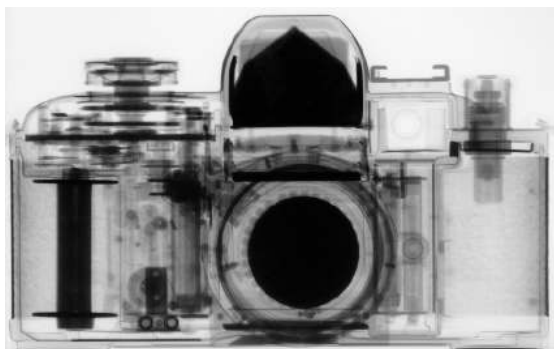


Fig. 7: Neutron radiography of a camera.

Group → ↓ Period	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
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. 8: Attenuation coefficients for thermal neutrons.

0.2.5 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.

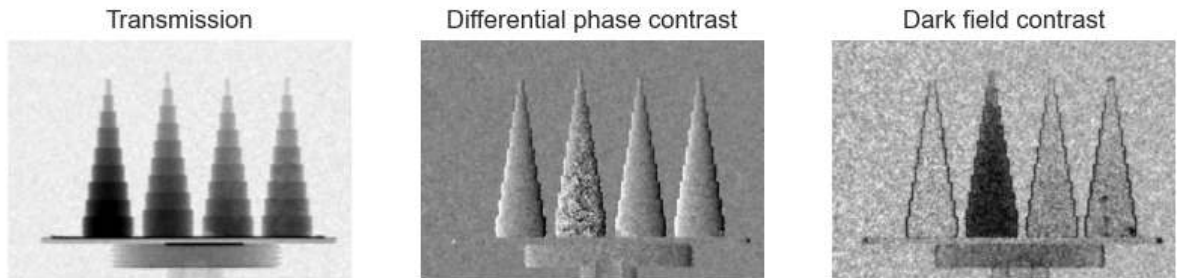


Fig. 9: X-ray radiography of a camera.

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											B 0.28	C 0.27	N 0.11	O 0.16	F 0.14	Ne 0.17
3	Na 0.13	Mg 0.24											Al 0.38	Si 0.33	P 0.25	S 0.30	Cl 0.23	Ar 0.20
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. 10: Attenuation coefficients for 125keV X-rays.

```
fig, ax=plt.subplots(1,3,figsize=(10,5))
ax[0].imshow(io.imread("figures/nGI_TTI.png")); ax[0].set_title('Transmission');
ax[0].axis('off')
ax[1].imshow(io.imread("figures/nGI_DPC.png")); ax[1].set_title('Differential phase_
contrast'); ax[1].axis('off')
ax[2].imshow(io.imread("figures/nGI_DFI.png")); ax[2].set_title('Dark field contrast
');
```



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

0.2.6 Some considered modalities - Spectroscopic imaging

- Material analysis
- Selector calibration

S. Peetermans

0.2.7 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.

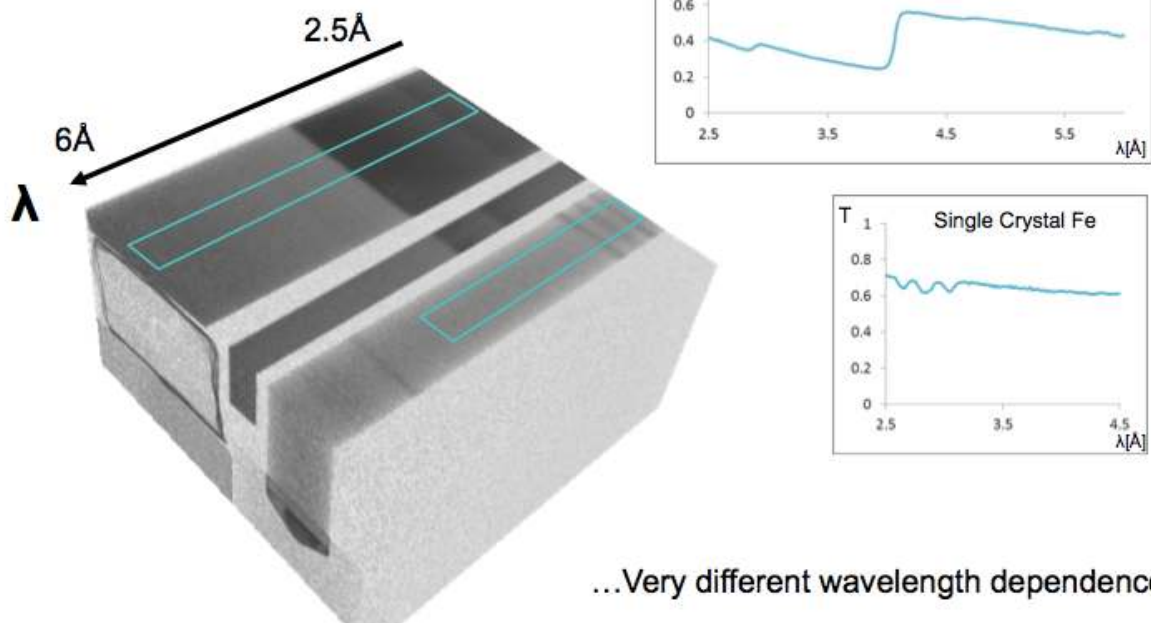
The modalities also differ in the resolution that can be achieved. Therefore, it makes sense to combine the modalities to increase the understanding of provided information.

Du et al. 2015

0.2.8 Considered modalities in biology and microscopy

Loussert-Fonta et al. 2020

A closer look at the iron samples



...Very different wavelength dependence

Fig. 11: Neutron energy scan through a piece of iron.

0.2.9 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

0.3 Data and image fusion

0.3.1 Definition

The theory, techniques and tools which are used for

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

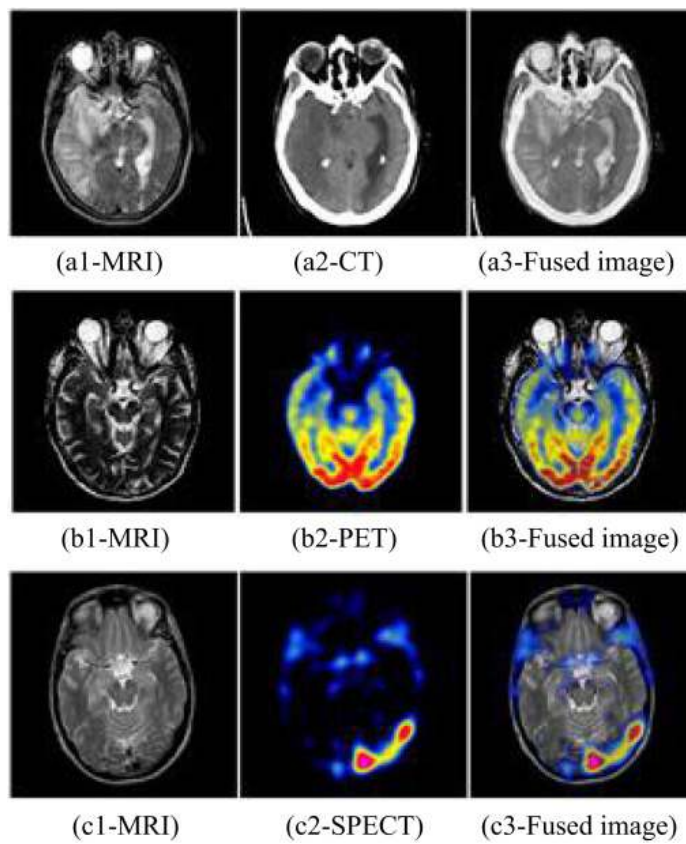


Fig. 12: Combining different medical imaging modalities.

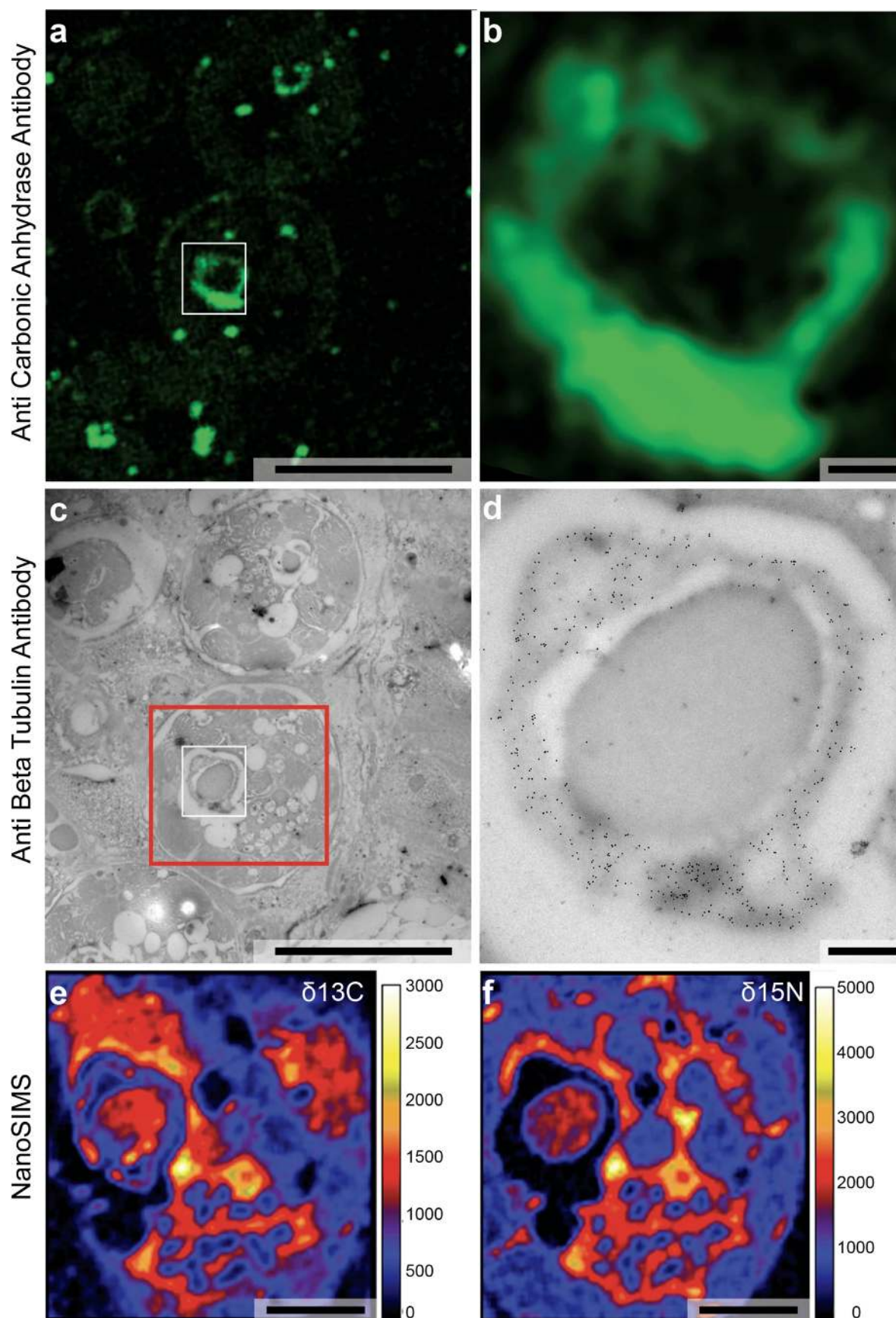


Fig. 13: The cryo section (ca. 100 nm thick) was imaged after simultaneous immunolocalization of carbonic anhydrase with anti-Alexa-associated secondary antibodies (a and b) and β -tubulin observed in TEM by 10 nm gold particles associated to the secondary antibody (c and d). The fluorescence microscope images (a and b; zoom in a) and the TEM micrographs (c and d; zoom in c) exhibit identical areas of the same thin section. e and f are NanoSIMS images showing the $\delta^{13}\text{C}$ and $\delta^{15}\text{N}$ distributions in the interior of a dinoflagellate symbiont. The area

0.3.2 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

0.3.3 Fusion approaches - no golden recipe

You know by now that there is no guaranteed recipe in image analysis. The choice of methods leading to the quantitative information must be tailored to the type of images and what information you need to extract. The same applies to image fusion. There are as always some general concepts to rest upon, but afterwards you need to get creative.

Fusion strategies

The purpose of the additional modality helps you to select the fusion strategy.

- **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 or identify artifacts.
- **Combination:** A single fusion method may not give the final result - combination

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

0.4 Image fusion workflow

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

Image acquisition

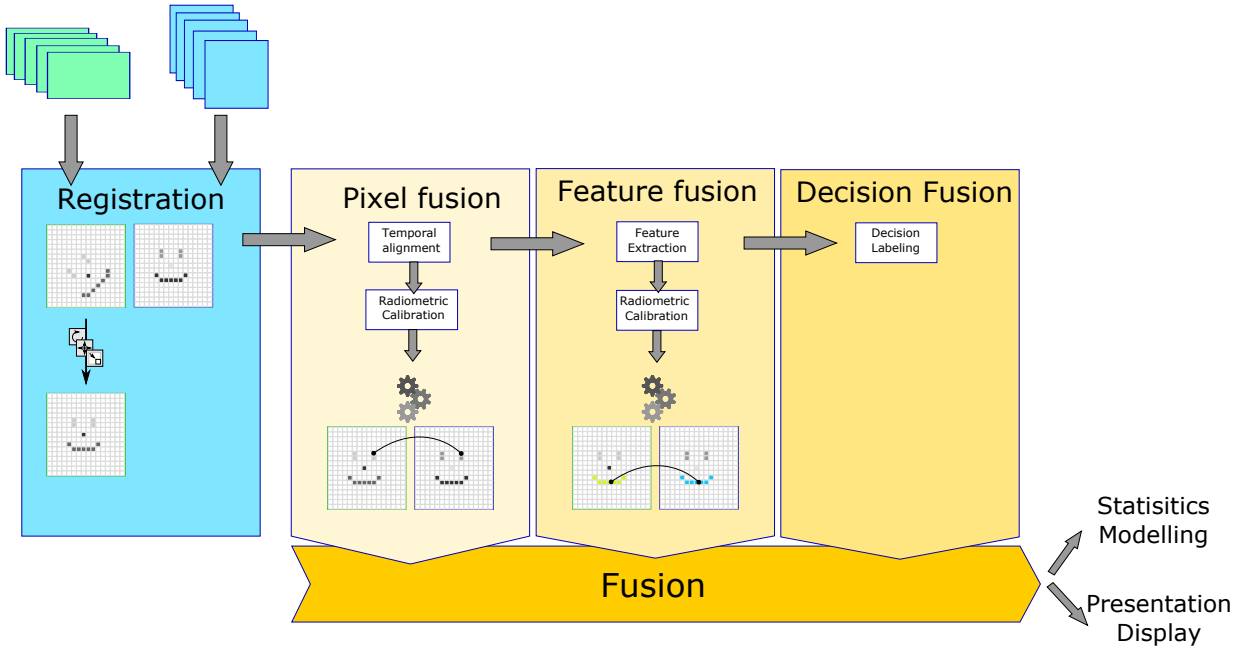


Fig. 1: Flow chart showing how image fusion can be done

Mitchel, 2010, Goshtasby, 2012

0.4.1 Catastrophic fusion

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. 2: More chefs don't always mean better soup, the same applies to data fusion. Chose your source combination and fusion metods carefully.

0.4.2 Image registration

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

We saw last week that image registration can be a quite complicated matter that involves the choice of optimizer, transform types, and cost function. Then we only considered images of the same modality. The introduction of a new modality adds to the complexity:

- The images show different features due their contrast mechanisms
- Different image sharpness.
- Different scales.
- Distortion

This leads to different choices of cost functions and maybe even changing the optimizer.

The process

The registration process still remains the same in the greater context. You have a fixed image and a moving image. The example in [Figure 3](#) show an example how an X-ray image is registered to a neutron image.

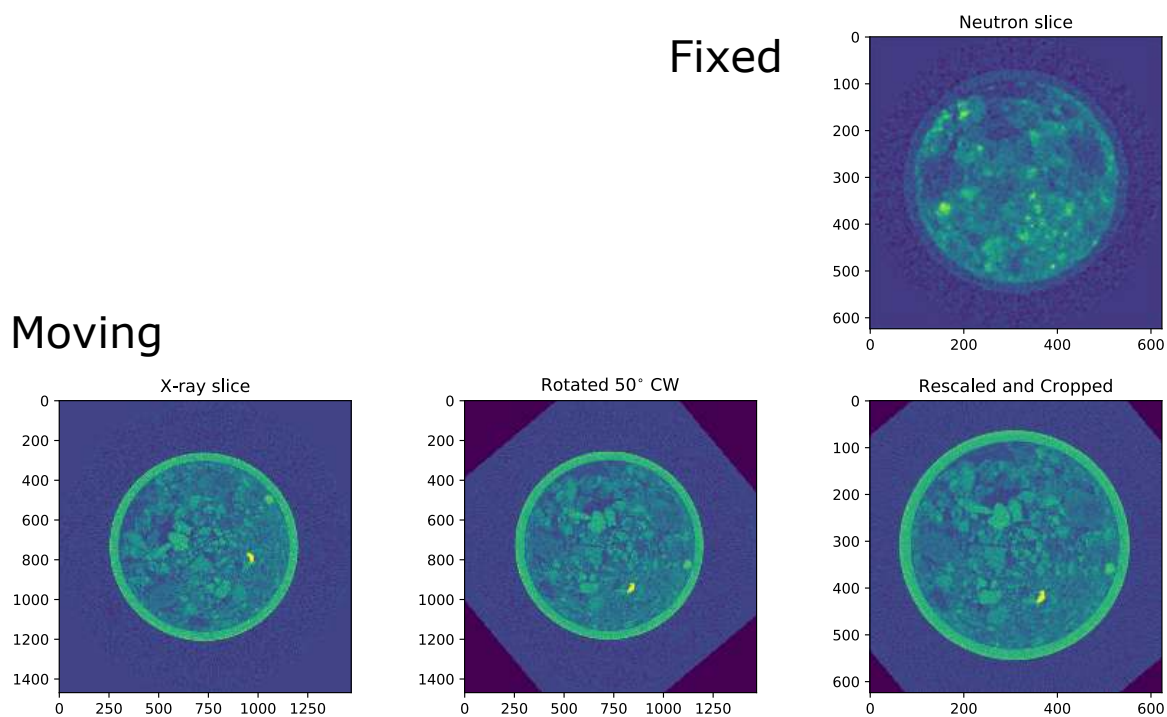


Fig. 3: Registration optimizes the scale, rotation, and position of an image compared to a fixed reference.

0.4.3 Registration considerations

Registration is an optimization problem with many local minima.

Manual or guided registration

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

Automatic registration

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

Goshtasby, 2012

0.5 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. 1: The sword from lake Zug as seen with neutrons.

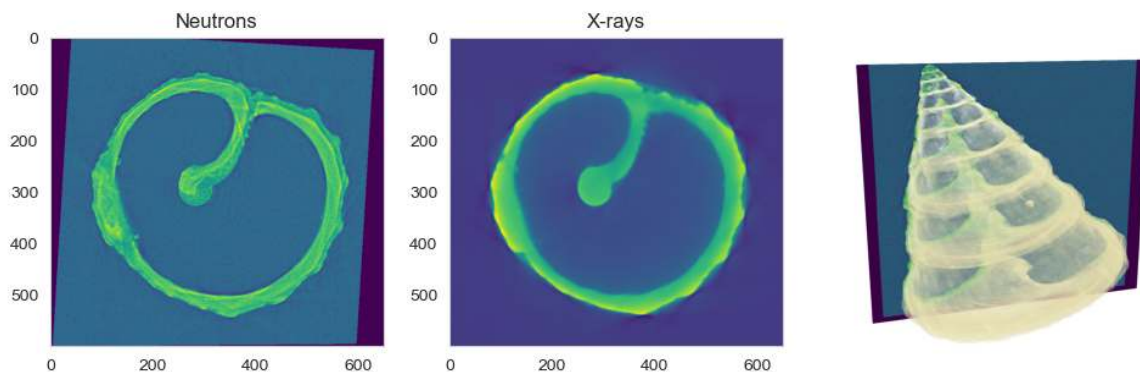


Fig. 2: The sword from lake Zug as seen with X-rays.

Mannes et al., 2015

0.5.1 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');
```



0.5.2 Visualization techniques - Checker board

It is, in many cases, relevant to visualize the resulting registration no matter how it was obtained. The checker board visualization shows the two modalities with alternating sub image blocks. This shows how well aligned the two modalities are but also the differences in contrast.

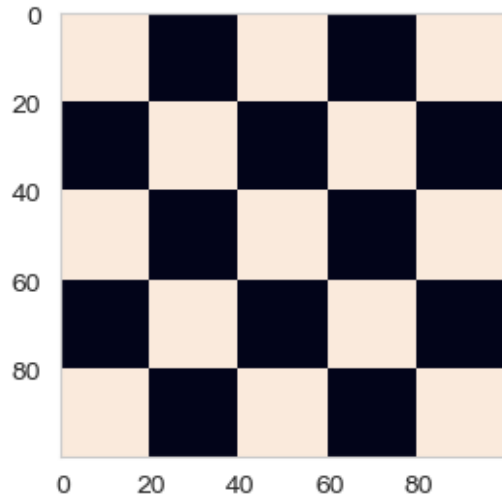
```
def checkerBoard(imgA, imgB, tiles=10) :
    if imgA.shape != imgB.shape :
        raise Exception('Image have different sizes')

    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])]

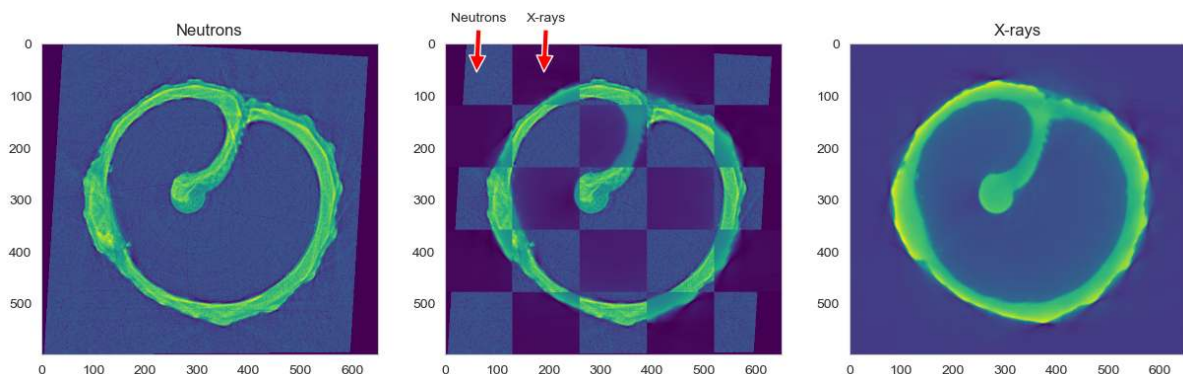
    return mix
plt.figure(figsize=(3,3))
plt.imshow(checkerBoard(np.ones((100,100)), np.zeros((100,100)), tiles=5), interpolation=
    ↪ 'none');
```



Try checker board with images

In this example, we show the same sample using neutron and X-ray images. This was done without any bias or contrast adjustment. We could have aimed for having the void outside the snail shell on the same level and thus get less checkered appearance.

```
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');
```



0.5.3 Visualization techniques - Color channel mixing

Another covisualization method is to use color coding. The images from the single modalities are mostly only describing a scalar value like intensity, attenuation coefficients etc. We saw in the first lecture that this scalar value can be mapped to a color map and thus show colorful images that may be easier to interpret than a gray scale.

Color images like pictures and cartoons use different models to represent colors as they appear in the real world. These models use multiple channels to represent the color. Some examples are RGB *red, green, and blue* and HSV *hue, saturation, and value*.

We can (mis-)use these color triplets to co-visualize the information from our modalities.

With two or three 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 normalization and scaling may be needed for best result.

Implementation of channel mixing

The function below implements the co-visualization of two modalities. It does a min-max normalization, this could also have been a whitening normalization $\$norm(img) = \frac{img - E[img]}{s[img]} \$$

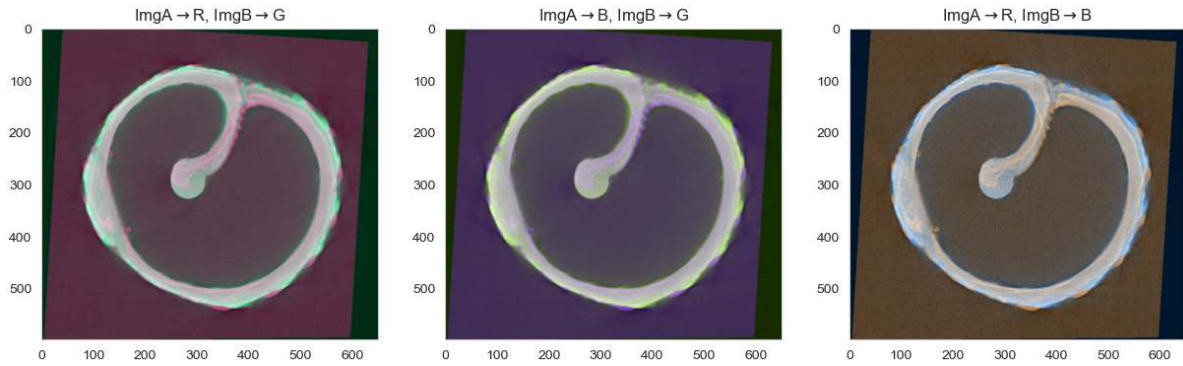
The order argument tells which color channel each piece of information should use.

```
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');
```



The choice of which color permutation you use depends on how well the two modalities appear and also on the application field. The permutation to the right in the figure is for example well suited for visualizing water in soil assuming the blue channel modality is more sensitive to water.

0.6 Bimodal segmentation

0.6.1 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 1](#) represents the area of ambiguous decisions.

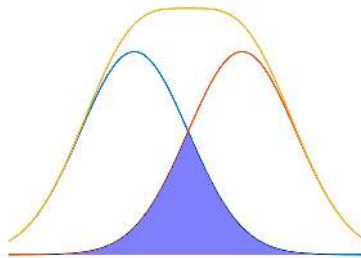


Fig. 1: Histogram of two classes using modality A.

0.6.2 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 2](#). This time the two classes have different responses and the histogram modes have swapped compared to [Figure 1](#).

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

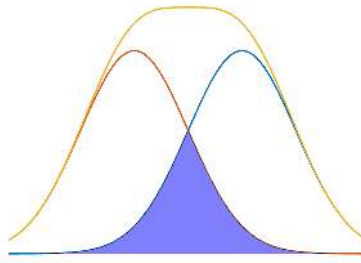


Fig. 2: Histogram of two classes using modality B .

0.6.3 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.

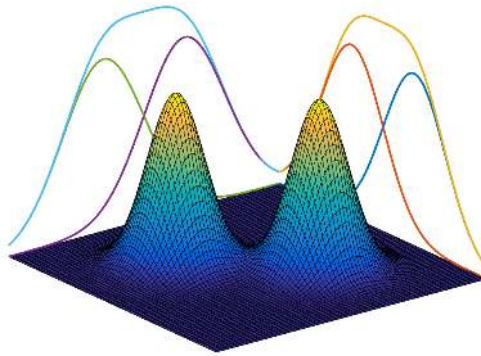


Fig. 3: A bivariate histogram of modalities A and B .

In the histogram shown in [Figure 3](#), we see that there is a clear separation between class A and B that could be easily thresholded.

Example: Roots in soil

Kaestner et al., 2016

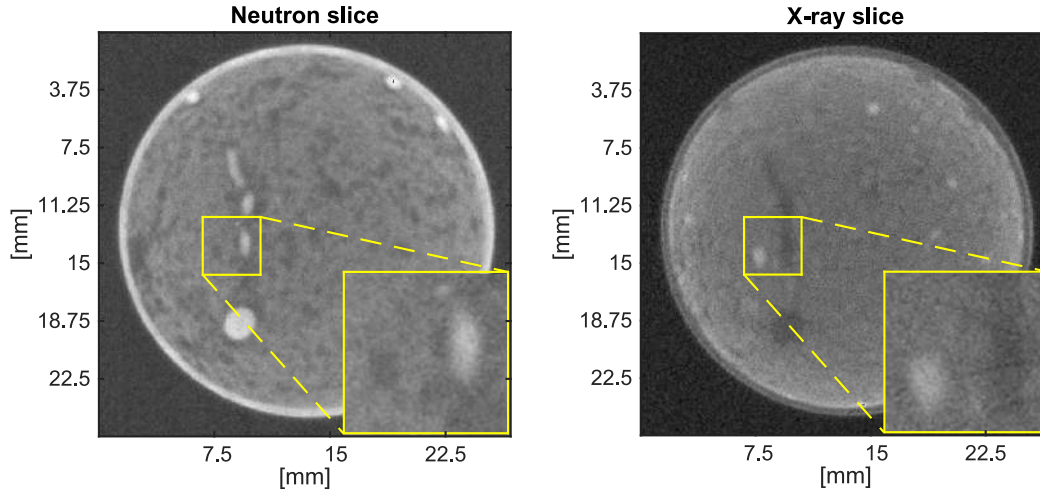


Fig. 4: Tomography slices of a soil sample with roots.

Bivariate histogram of roots

0.6.4 Segmentation methods

Data

- Images from M modalities f_1, \dots, f_M
- Registered
- Artifact corrected

Classes

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

Duda, Hart, and Stork, 2001

0.6.5 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

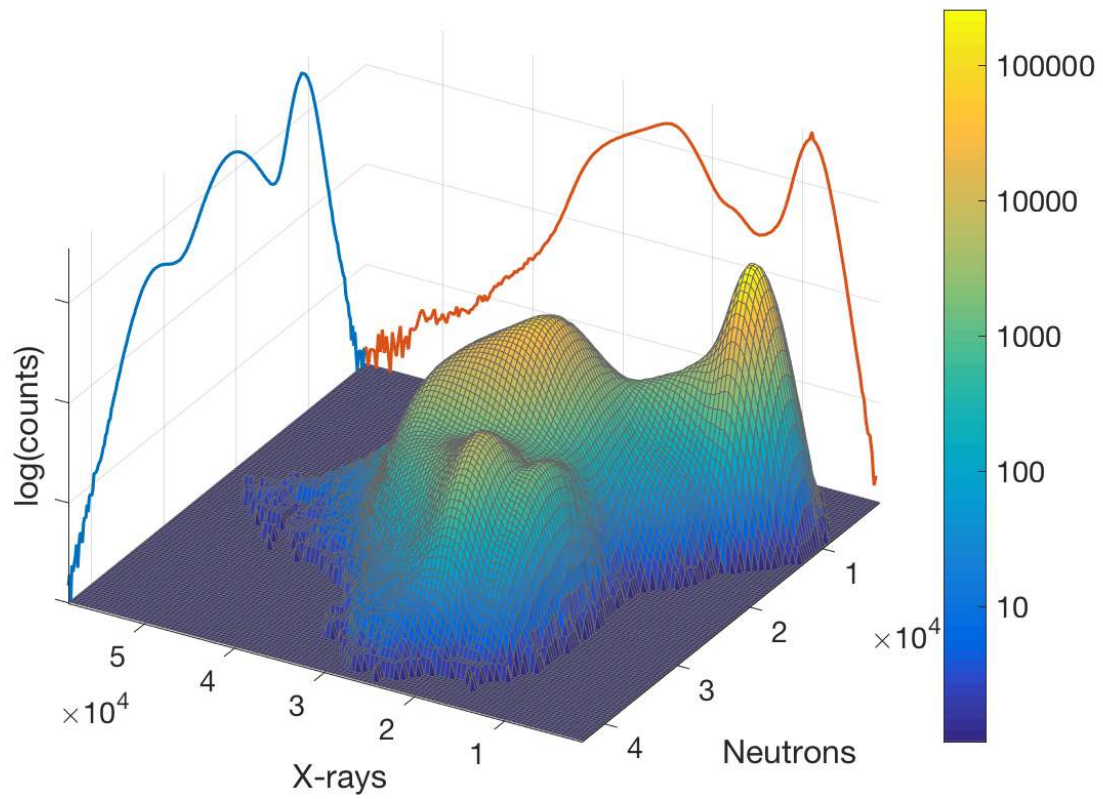


Fig. 5: Bivariate histogram of the root images in [Figure 4](#)

0.6.6 Gaussian mixture model

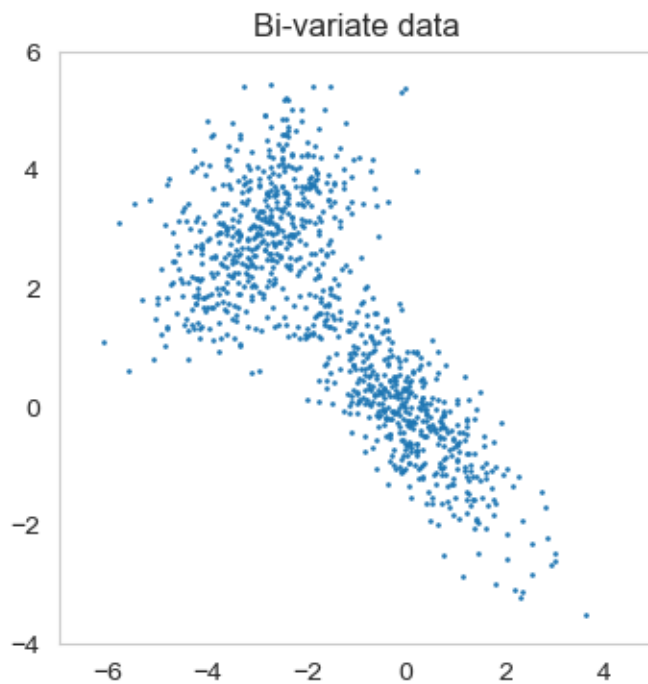
With Gaussian distribution we can describe the bivariate histogram using: $p(\theta) = \sum_1^N \phi_i \mathcal{N}(\mu_i, \Sigma_i)$

- μ_i - vector with averages for each class.
- Σ_i - covariance matrix for each class.
- ϕ_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)):
```

(continues on next page)

(continued from previous page)

```

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], 2, color=color, alpha=0.2)

# 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], angle=180. + angle,
    color=color)
    ell.set_clip_box(ax.bbox)
    ell.set_alpha(0.3)
    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)

```

Gaussian mixture model fitting

In this example we explore how the Gaussian mixture model handles an increasing number of classes. The data set is created with two classes, but what happens when we increase the number of classes.

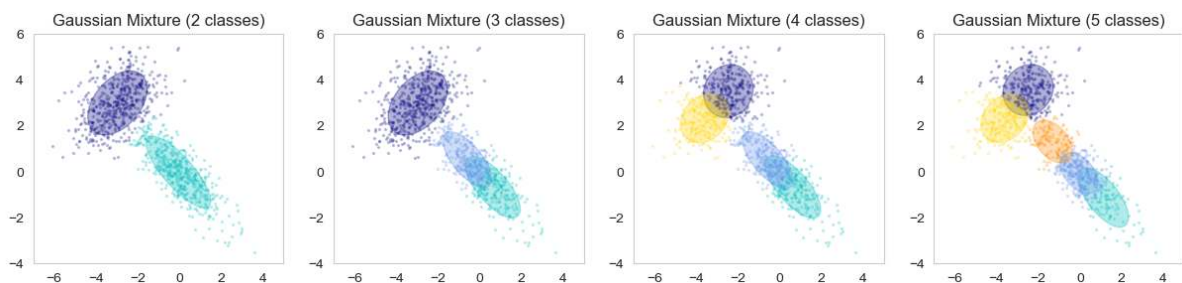
```

fig, axes = plt.subplots(1,4,figsize=(15,3))
# Fit a Gaussian mixture with EM using five components
for i,ax in zip(np.arange(0,len(axes.ravel())) ,axes.ravel()) :
    classes = i+2

    gmm = mixture.GaussianMixture(n_components=classes,random_state=0, covariance_
    type='full').fit(X)

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

```



The two classes are, as expected, well identified in the two-class model. Models with more classes split the original classes

into smaller classes, but the new classes cluster in the regions of the original two classes.

0.6.7 Classification distances

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

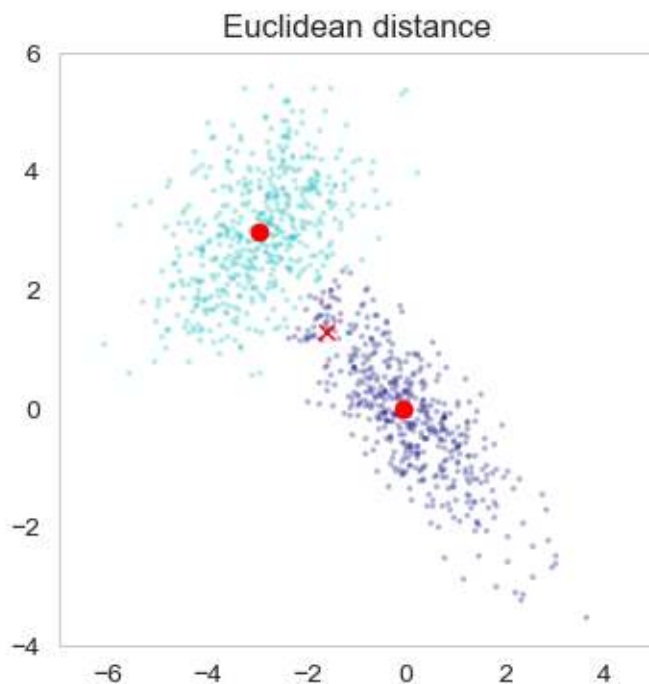
We can find the nearest neighbor class using the following distances

Euclidean

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

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

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



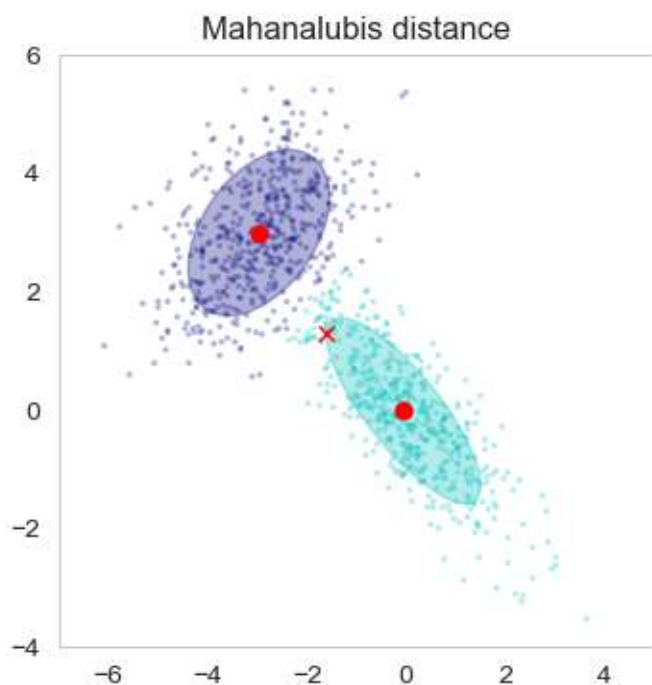
The Euclidean distance is the least complicated distance metric between two classes. It only requires the centroid, i.e. the mean, of the two classes. This does of course have the disadvantage that it is not sensitive to the distribution shape.

Mahanalubis

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

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

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



The Mahanalubis distance adds the covariance of the distribution in the calculation. This adds the orientation and shape of the distribution which makes it easier to handle ambiguous cases that were hard to catch with the Euclidean distance.

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.

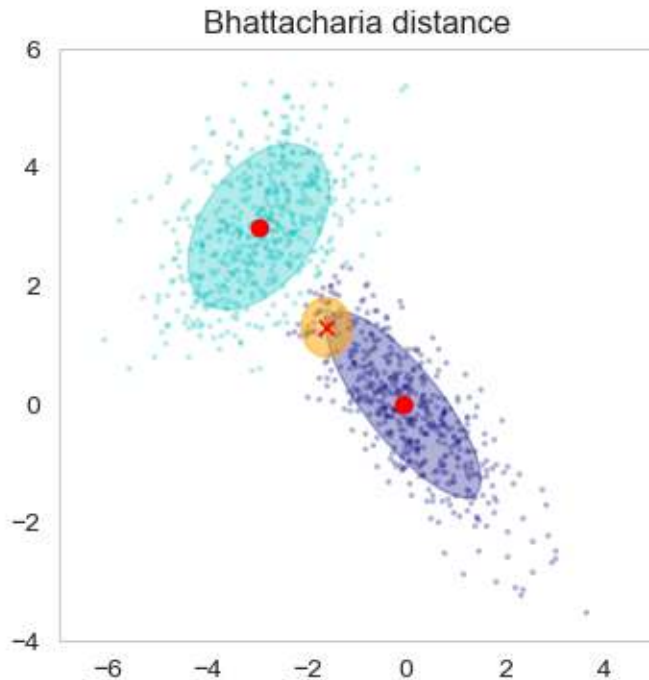
```
gmm = mixture.GaussianMixture(n_components=2, covariance_type='full').fit(X)
m=[-1.6,1.3]
fig,ax3=plt.subplots(1,figsize=(4,4))

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, angle=0, color='orange')
ell.set_clip_box(ax3.bbox)
```

(continues on next page)

(continued from previous page)

```
ell.set_alpha(0.5)
ax3.add_artist(ell)
ax3.plot(m[0],m[1], 'rx');
```



The Bhattacharia distance add even further complexity to the distance computation as it also requires the covariance matrix of the probed point. Obtaining the sample covariance is expensive in terms of resolution because it requires a neighborhood to be computed.

0.6.8 Graphical presentation of the different distances

Here is a side by side comparison of the three distance metrics.

```
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_,
             'Mahalanubis distance'.format(2), ax2, showCenter=True)
ax2.plot(-1.6,1.3, 'rx')

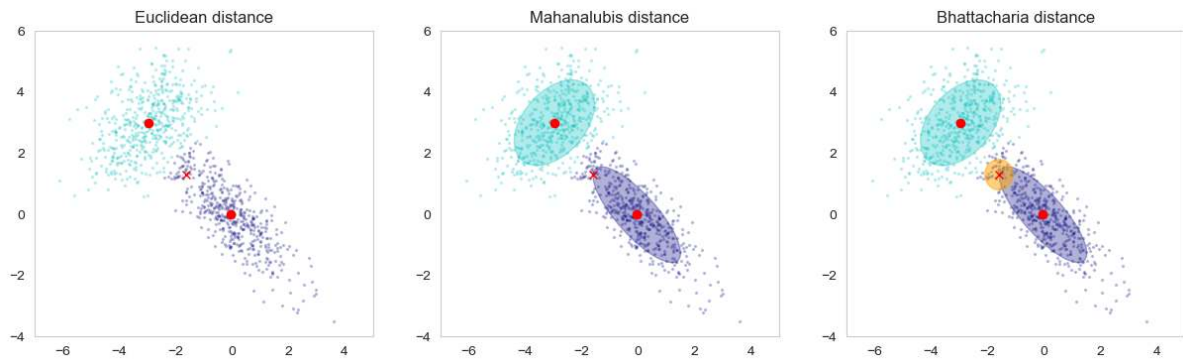
plot_results(X, gmm.predict(X), gmm.means_, gmm.covariances_,
             'Bhattacharia distance'.format(2), ax3, showCenter=True)

v=1
ell = mpl.patches.Ellipse(m, v, v, angle=0, color='orange')
ell.set_clip_box(ax3.bbox)
ell.set_alpha(0.5)
```

(continues on next page)

(continued from previous page)

```
ax3.add_artist(e11)
ax3.plot(m[0],m[1], 'rx');
```



0.6.9 Segmentation by Euclidean distance

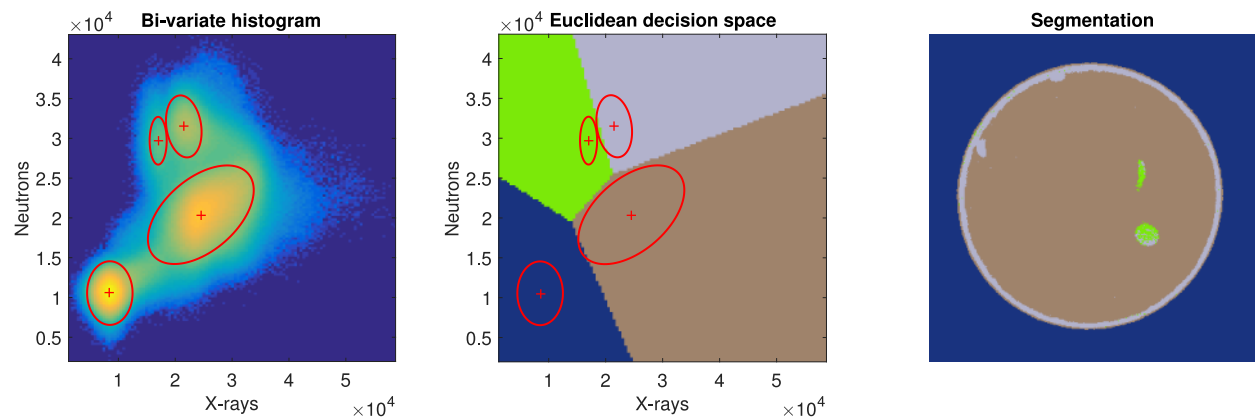


Fig. 6: Segmenting the root image in

Kaestner et al., 2016

0.7 Bivariate estimation: Working with attenuation coefficients

0.7.1 Beer-Lamberts law

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

- ρ Material density
- A Atomic weight
- σ microscopic cross section
 - Probability of interaction
 - modality dependent
- x propagation length

0.7.2 Equation system

$$\Sigma_0 x_0 + \Sigma_1 x_1 = q_N + noise$$

$$\mu_0 x_0 + \mu_1 x_1 = q_X + noise$$

- attn coeff known \rightarrow estimate lengths.
- More pixels \rightarrow use least squares estimate.

0.8 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 got 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 sensity the changes in mixing ratios and other process related parameters.

- Combined setups

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

Figure 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 iamges are registered before any analysis can be performed.

0.9 Some software engineering

Repeatable workflows require some software engineering skills

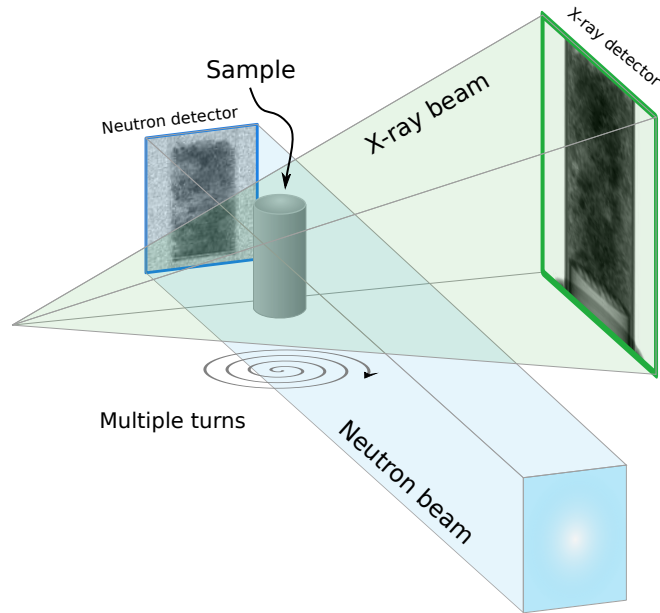


Fig. 1: The outline of a bimodal imaging system for neutrons and X-rays.

0.9.1 Repeatable workflows

From lecture 8

Repeatability requires that your code:

- is testable
- has a tracked history
- is reviewed
- is documented

... and keep track of bugs and new ideas in an issue manager.



Fig. 1: The outline of a bimodal imaging system for neutrons and X-rays.

Tools to support reproducible workflows

Use a repository

History management

Use automated tests

Guarantees unit functionality

Use a build server

Combines repository and tests

Each of these measures will be described in more detail in the following sections, but first a little introduction to what each step brings.

Repositories

Correctly used a source code repository helps you to keep track of your changes and also makes it possible to reproduce a result even though your code has evolved since you produced an earlier result. The repo also makes it possible to work on parallel ideas without ruining the main code. Hence, it gives you more freedom to be creative with less risk.

Automated tests

Automated tests are small programs written to verify that your main code actually does what it is supposed to do. This is a sanity check that is very helpful when you start working on different branches in your repository. The tests must always run successfully before you are allowed to merge a new piece of code into the main branch of your repository.

Build server

A build server is set up to build and test the code you submit to the repository. This is particularly important when your project grows. For smaller projects it is less relevant.

0.9.2 How to organize the code

In this course, we have mainly implemented the analysis using python and in particular using jupyter notebooks.

Notebooks are good for prototyping ... but are not good for

- structuring/maintenance
- repositories
- portability/sharing
- testing

To achieve a cleaner workflow with your code it is a good idea to separate common functions that are used often in many contexts into library files. This has the advantage that the notebooks are cleaner and that a change in a function is propagated to all notebooks using that function without having to copy the function between notebooks. Library files are also easier to access in testing frameworks.

Save functions in .py files which are imported

Your functions are saved in a file `'mylib.py'`

```
import mylib

mylib.myfunction(1234)
```

Reload during development

```
import importlib
importlib.reload(mylib)
```

Reloading a library is essential, otherwise the notebook doesn't know that you have updated your library functions. This is of course most important during the development phase. Reloading is naturally less relevant in end user notebooks or scripts.

More advanced use of libraries is beyond the scope of this lecture, but it is well described in many books and online resources.

0.9.3 Documentation

Documentation is a tedious task that many dread and leave to the end, if at all writing it with the motivation that "It's anyway only I who will use the code". This kind of reasoning is not very sustainable. Have you tried to take a look at your own code some months later? Did you instantaneously understand what it does and how it should be used?

There are different levels of documentation:

- In-code comments
- API (application programmer interface) describes which arguments and return values
- User documentation describes how the code is intended to be used.

API documentation is a minimum!

API Documentation

Doc string in python

```
def fancy_function(value) :
    """ Description: Makes fantastic calculations

    Parameters:
    value (int): Description of argument

    Returns:
    int: Returning value

    """
    return 42+value

help(fancy_function)
```

Help on function fancy_function in module __main__:

```
fancy_function(value)
  Description: Makes fantastic calculations
```

(continues on next page)

(continued from previous page)

```
Parameters:
value (int): Description of argument

Returns:
int: Returning value
```

Doxygen for many languages

<https://www.doxygen.nl>

0.9.4 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 support for automated testing and reporting
- [Computational science: ... Error](#)
- [Unit testing guidelines](#)

0.9.5 Unit Testing - design

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

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*

How to test

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

Test data

Ideally with realistic but simulated test data

An unit test example

Given the following function that counts all nonzero pixels in an image.

```
def vxCnt=countVoxs(inImage)
```

We can write the following tests:

```
def testEmpty2d() :
    assert countVoxs(zeros(3,3)) == 0
```

```
def testEmpty3d() :
    assert countVoxs(zeros(3,3,3)) == 0
```

```
def testDiag2d() :
    assert countVoxs(eye(3)) == 3
```

Unit tests can, as you see in this example, be quite trivial and that is the strength of the tests. They are fast to run and don't cost much waiting time and at the same time they also guarantee that the functionality is provided.

0.9.6 Unit Testing: Example

Given the following 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.graphs.Digraph at 0x13b38b4f0>
```

0.9.7 Unit Testing in Python

unittest

The built-in Python module, ideal for beginners due to its simplicity.

PyTest

A popular framework known for its flexibility, advanced features, and readable style.

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

0.9.8 Unit testing examples from Scikit Image

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

- Test Watershed
- Test Connected Components

```
class TestWatershed(unittest.TestCase):
    eight = np.ones((3, 3), bool)

    def test_watershed01(self):
        "watershed 1"
        data = np.array([[0, 0, 0, 0, 0, 0, 0],
                          [0, 0, 0, 0, 0, 0, 0],
                          [0, 0, 0, 0, 0, 0, 0],
                          [0, 1, 1, 1, 1, 1, 0],
                          [0, 1, 0, 0, 0, 1, 0],
                          [0, 1, 0, 0, 0, 1, 0],
                          [0, 1, 0, 0, 0, 1, 0],
                          [0, 1, 1, 1, 1, 1, 0],
                          [0, 0, 0, 0, 0, 0, 0],
                          [0, 0, 0, 0, 0, 0, 0]], np.uint8)
        markers = np.array([[ -1, 0, 0, 0, 0, 0, 0],
                             [0, 0, 0, 0, 0, 0, 0],
                             [0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 1, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0],
                             [ 0, 0, 0, 0, 0, 0, 0]],
                             np.int8)
        out = watershed(data, markers, self.eight)
        expected = np.array([[ -1, -1, -1, -1, -1, -1, -1],
                              [-1, -1, -1, -1, -1, -1, -1],
                              [-1, -1, -1, -1, -1, -1, -1],
                              [-1, 1, 1, 1, 1, 1, -1],
                              [-1, 1, 1, 1, 1, 1, -1],
```

(continues on next page)

(continued from previous page)

```

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

```

0.9.9 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

```

0.9.10 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

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
```

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 of 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.

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.],
           [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)
```

(continues on next page)

(continued from previous page)

```
Expecting:
    array([[1, 0, 0],
           [0, 0, 0],
           [0, 0, 2]])
ok
```

Unit Testing Matlab

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

Unit Testing in C++

- Google test
- Boost.test
- QTest

Test management by CTest from CMake

0.9.11 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

0.9.12 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

Why should I use a repository?

- It makes it easy to go 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)

Basic repository operations

Add and Commit

The first thing you do with a repository is to register which files to include this is the *add* operation. Once you have made some changes, you want to store these changes with a comment that tells what you have changed. You commit as often as you like, but need to find a balance in the granularity. Too often gets fragmented and too rarely include too many changes that are harder to track.

Push and Pull

Pushing and pulling are the operations that communicate with the repository server. Push sends all committed changes to the server. Pull fetches the latest available changes from the repository server. It often happens that you have to pull from the server before you can push anything because differences must be merged and resolved locally before you can upload your changes. Sometimes it even happens that there are unresolvable differences, in particular, when you have changes that overlap the same piece of code that was already changed in a different commit. The conflicts must be resolved and the unit tests should confirm that the merge was successful.

Create, switch, merge branch

Branching is a very important feature of a repository. Branches allow you to create an independent copy of the repository that you can change as much as you like without affecting the main code. This is the place where you can test new ideas work on bugs etc. It is also easy to jump between branches if need to work on a different task.

Now, it is not so relevant to work on a copy of the code. At some point you'll want to bring the changes back into the main code, e.g. when you have a satisfying solution for the task you address in the branch. This is then done with the merging operation. The workflow includes

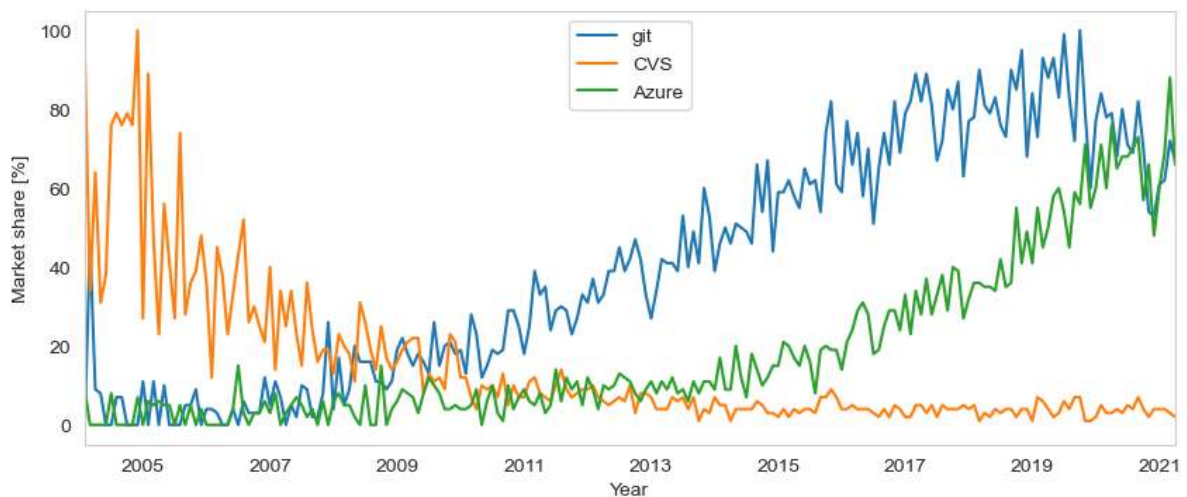
1. Merge main branch into the current and fix potential conflicts. They can have introduced while you were working on this task.
2. Run some tests to confirm the quality.
3. Commit and push the current branch to the server.
4. Make a so called "pull request" on the server to indicate that you want to bring new code into the main.
5. Confirm the merge. Now your branch is merged with the main and everybody can safely enjoy the new addition.

Bringing code that crashes the main build is considered bad behaviour and the one who did it is expected to fix asap and bring a cake to the team to apologize.

Different repository frameworks

- Git
- Azure
- Subversion
- CVS

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



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
- PSI Gitea

0.9.13 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.

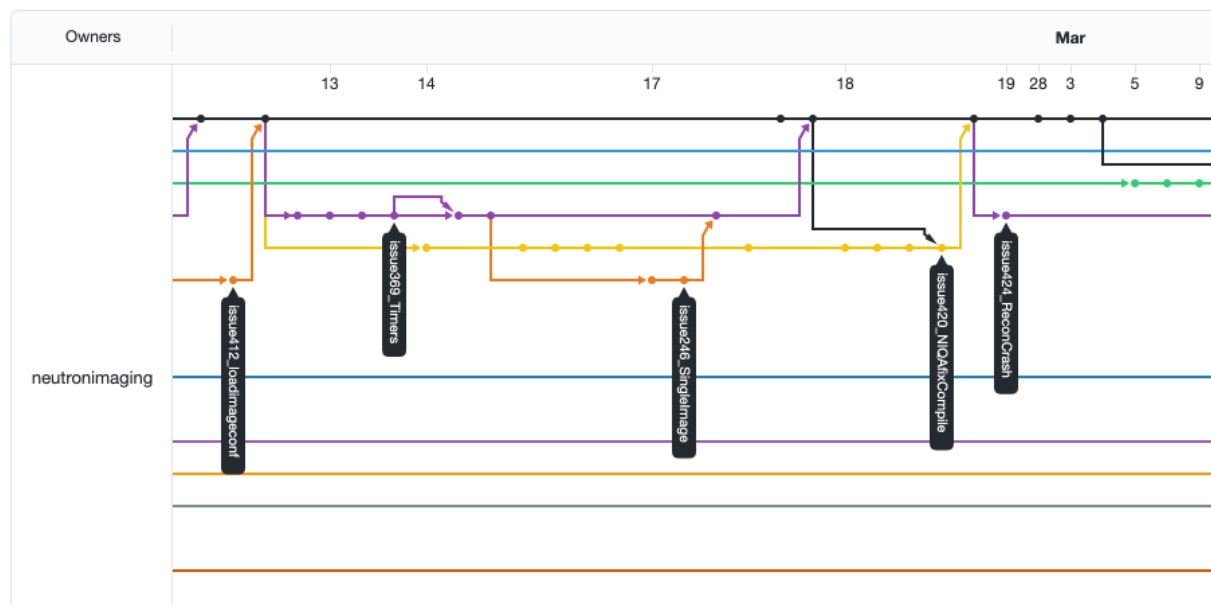


Fig. 2: A snapshot of the branch network from a git repository.

[GitHub tutorials](#)

0.9.14 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.

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

The screenshot displays the Travis CI web interface for the repository `neutronimaging/ToFImaging`. The interface is divided into a left sidebar and a main content area.

Left Sidebar (My Repositories):

- `neutronimaging/python_notebook` # 97: Duration: 1 min 1 sec, Finished: 15 hours ago.
- `neutronimaging/imagingsuite` # 75: Duration: 48 sec, Finished: a day ago.
- `ImagingLectures/Quantitative` # 232: Duration: 12 min 42 sec, Finished: 2 days ago.
- `neutronimaging/ToFImaging` # 86: Duration: 3 min 57 sec, Finished: 5 days ago.

Main Content Area (neutronimaging / ToFImaging):

The main area shows the current build status as **build passing**. The build details for the `66_reduction_tools_unit_tests` job are as follows:

- Status: **66_reduction_tools_unit_tests** advanced direct fitting is | **#86 passed**
- Commit: `5e06360`
- Compare: `3c6f1b8...5e06360`
- Branch: `66_reduction_tools_unit_tests`
- Author: Jean Bilheux
- Duration: Ran for 2 min 29 sec
- Total time: 3 min 57 sec
- Finished: 5 days ago
- Buttons: Restart build

Build jobs table:

Build jobs	View config
✓ # 86.1	AMD64 Focal Python: 3.7 no environment variables set 1 min 49 sec
✓ # 86.2	AMD64 Focal Python: 3.8 no environment variables set 2 min 8 sec

Fig. 3: Test reporting in Travis.

0.9.15 A git demonstration

Using GitHub we will

- Create a repository
- Add some files and commit
- Push and pull
- Create and switch to new branch
- Make changes
- Merge branch into main

0.10 Summary

0.10.1 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.

0.10.2 Software engineering

- Documentation
- Unit testing
- Working with repositories
- Continuous integration