Quantitative Big Imaging - Image enhancement

Anders Kaestner

Aug 10, 2024

CONTENTS

This is the lecture notes for the 3rd lecture of the Quantitative big imaging class given during the spring semester 2022 at ETH Zurich, Switzerland.

0.1 Image Enhancement

Quantitative Big Imaging ETHZ: 227-0966-00L

0.1.1 Todays lecture

- Imperfect images and noise
- Filters
- The Fourier transform
- Advanced filters
- Structure tensor
- Evaluation workflows

Literature

Basic filters

- B. Jähne,Digital Image Processing, Springer Verlag, 2005.
- J.C. Russ,The Image Processing Handbook, Wiley, 2006.

Orientations

- Z. Püspöki et al., Transforms and Operators for Directional Bioimage Analysis: A Survey in Focus on Bio-Image Informatics, Springer Verlag, 2016.
- J. Bigün Vision with Direction, Springer Verlag, 2006.

Orientation demonstration

Metrics

Wang and Bovik, Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures, IEEE Signal Processing Magazine, 2009

… and some more detailed publications during the lecture.

We need some modules

These modules are needed to run the python cells in this lecture.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import matplotlib        as mpl
from matplotlib.colors import LogNorm
import skimage as ski
import skimage.filters as flt
import skimage.io as io
from skimage.morphology import disk
import scipy.ndimage as ndimage
from scipy.signal import convolve2d
from scipy.ndimage import gaussian_filter
from matplotlib.patches import ConnectionPatch
from scipy.signal import medfilt

mpl.rcParams['figure.dpi'] = 100
```

0.2 Measurements are rarely perfect

There is no perfect measurement. This is also true for neutron imaging. The ideal image is the sample is distorted for many reasons. The figure below shows how an image of a sample can look after passing though the acquisition system. These quailty degradations will have an impact on the analysis of the image data. In some cases, it is possible to correct for some of these artifacts using classing image processing techniques. There are however also cases that require extra effort to correct the artifacts.

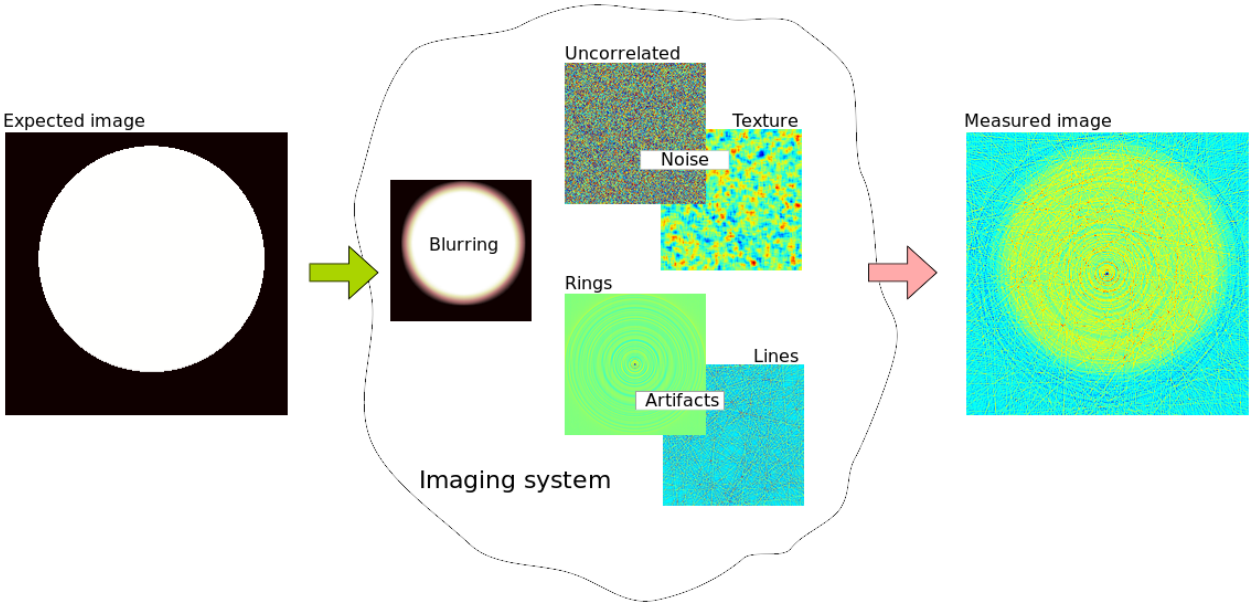


Fig. 1: Schematic showing the error contributions in an imperfect imaging system.

0.2.1 Factors affecting the image quality

No measurement system is perfect and therefore, you will also not get perfect images of the sample you measured. In the figure you can see the slice as it would be in a perfect world to the left and what you actually would measure with an imaging system. Some are of higher quality than others but there are still a collection of factors that have an impact on the image quality.

The list below provides some factors that affect the quality of the acquired images. Most of them can be handled by changing the imaging configuration in some sense. It can however be that the sample or process observed put limitiations on how much the acquisition can be tuned to obtain the perfect image.

- Resolution (Imaging system transfer functions)
- Noise
- Contrast
- Inhomogeneous contrast
- Artifacts

Resolution

The resolution is primarily determined optical transfer function of the imaging system. The actual resolution is given by the extents of the sample and how much the detector needs to capture in one image. This gives the field of view and given the number pixels in the used detector it is possible to calculate the pixel size. The pixel size limits the size of the smallest feature in the image that can be detected. The scintillator, which is used to convert neutrons into visible light, is chosen to

1. match the sampling rate given by the pixel size.
2. provide sufficient neutron capture to obtain sufficient light output for a given exposure time.

Noise

An imaging system has many noise sources, each with its own distribution e.g.

1. Neutron statistics - how many neutrons are collected in a pixel. This noise is Poisson distributed.
2. Photon statistics - how many photons are produced by each neutron. This noise is also Poisson distributed.
3. Thermal noise from the electronics which has a Gaussian distribution.
4. Digitation noise from converting the charges collected for the photons into digital numbers that can be transfered and stored by a computer, this noise has a binominal distribution.

The neutron statistics are mostly dominant in neutron imaging but in some cases it could also be that the photon statistics play a role.

Contrast

The contrast in the sample is a consequence of

1. how well the sample transmits the chosen radiation type. For neutrons you obtain good contrast from materials containing hydrogen or lithium while many metals are more transparent.
2. the amount of a specific element or material represented in a unit cell, e.g. a pixel (radiograph) or a voxel (tomography).

The objective of many experiments is to quantify the amount of a specific material. This could for example be the amount of water in a porous medium.

Good contrast between different image features is important if you want to segment them to make conclusions about the image content. Therefore, the radiation type should be chosen to provide the best contrast between the features.

Inhomogeneous contrast

The contrast in the raw radiograph depends much on the beam profile. These variations are easily corrected by normalizing the images by an open beam or flat field image.

- **Biases introduced by scattering** Scattering is the dominant interaction for many materials use in neutron imaging. This means that neutrons that are not passing straight though the sample are scattered and contribute to a background cloud of neutrons that build up a bias of neutron that are also detected and contribute to the
- **Biases from beam hardening** is a problem that is more present in x-ray imaging and is caused by that fact that the attenuation coefficient depends on the energy of the radiation. Higher energies have lower attenuation coefficient, thus will high energies penetrate the thicker samples than lower energies. This can be seen when a polychromatic beam is used.

Artifacts

Many images suffer from outliers caused by stray rays hitting the detector. Typical artefacts in tomography data are

- Lines, which are caused by outlier spots that only appear in single projections. These spot appear as lines in the reconstructed images.
- Rings are caused by stuck pixels which have the same value in a projections.

0.2.2 A typical processing chain

Traditionally, the processing of image data can be divided into a series of subtasks that provide the final result.

- **Acquisition** The data must obviously be acquired and stored. There are cases when simulated data is used. Then, the acquisition is replaced by the process to simulate the data.
- **Enhancement** The raw data is usually not ready to be processed in the form is comes from the acquisition. It usually has noise and artifacts as we saw on the previous slide. The enhancement step suppresses unwanted information in the data.
- **Segmentation** The segmenation identifies different regions based on different features such as intensity distribution and shape.
- **Post processing** After segmentation, there may be falsely identified regions. These are removed in a post processing step.
- **Evaluation** The last step of the process is to make conclusions based on the image data. It could be modelling material distrbutions, measuring shapes etc.

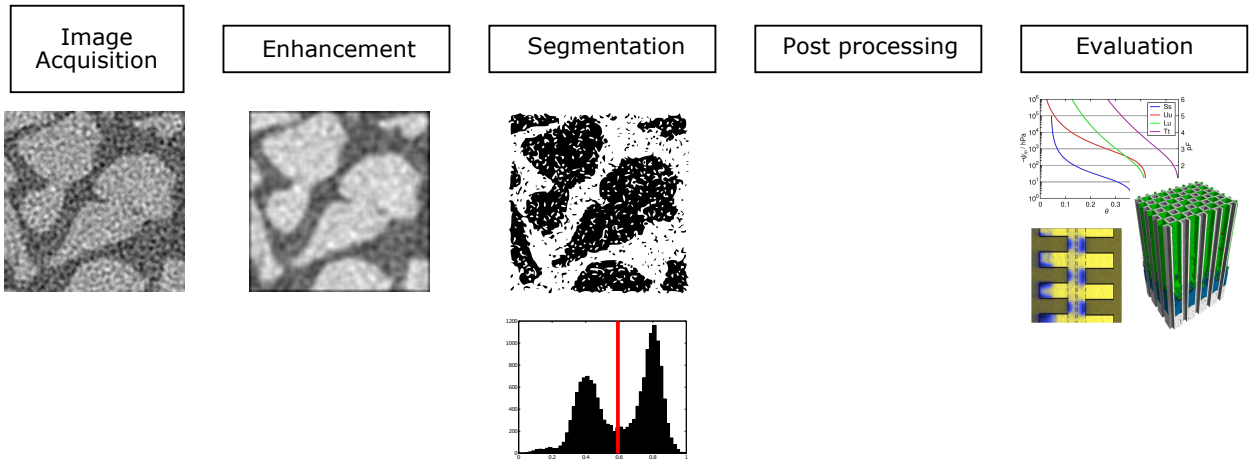


Fig. 2: Typical steps of an image processing work flow.

Today's lecture will focus on **enhancement**

0.3 Noise and artifacts

Noise is in very general terms for the unwanted information in a signal.

More specifically;
we are talking about **random contributions** that **obscure the image information** we are interested in.

0.3.1 Noise types

Noise can have many different characteristics. In general, it is driven by a random distribution.

- Spatially uncorrelated noise

With spatially uncorrelated noise each pixel has a random value which is independend of the pixel neighborhood. This is also the easiest noise type to simulate.

- Event noise

The even noise has a random activation function that triggers the event of each pixel with some probabilty. This noise type produces spots randomly distributed over the image. The spots may also have a randomw intensity.

- Stuctured noise

The structured noise depends on the values of the pixel neighborhood and is thus spatially correlated. It is mostly driven by an uncorrelated noise source which is blurred by a weighted combination of the neighborhood.

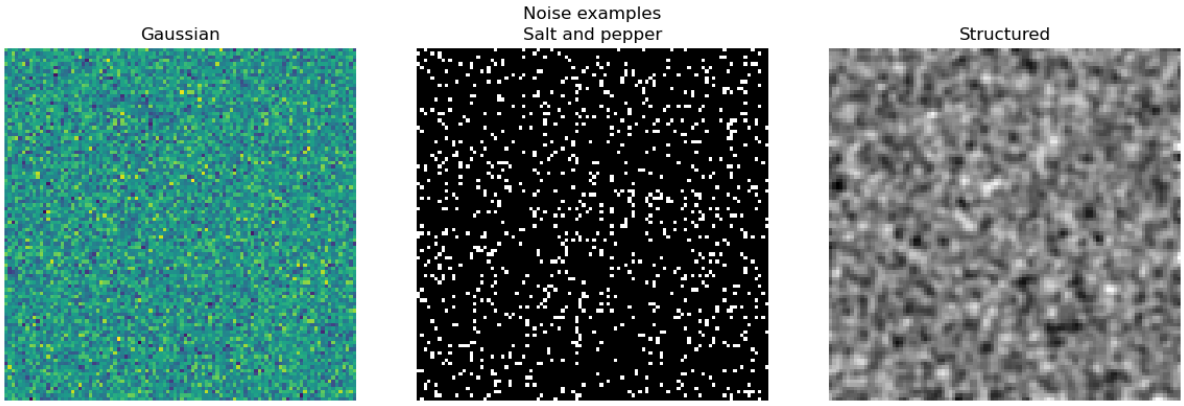
The figure below shows examples of the three noise types.

```
plt.figure(figsize=[12,4]);
plt.suptitle('Noise examples')
plt.subplot(1,3,1);plt.imshow(np.random.normal(0,1,[100,100])); plt.title('Gaussian');
 ↪plt.axis('off');
plt.subplot(1,3,2);plt.imshow(0.90<np.random.uniform(0,1,size=[100,100]),cmap='gray');
 ↪ plt.title("Salt and pepper"),plt.axis('off');
```

(continues on next page)

(continued from previous page)

```
plt.subplot(1,3,3);plt.imshow(ski.filters.gaussian(np.random.normal(0,1,size=[100,
 ↪100]),sigma=1),cmap='gray'); plt.title("Structured"),plt.axis('off');
plt.tight_layout()
```



Noise models - Gaussian noise

Gaussian noise is the most common random distribution used. All other distributions asymptotically converges towards the Gaussian distribution thanks to the central limit theorem. The Gaussian noise is an easy distribution to work with when you derive signal processing models. This is also the reason why it is so popular to use this model also for non-Gaussian noise.

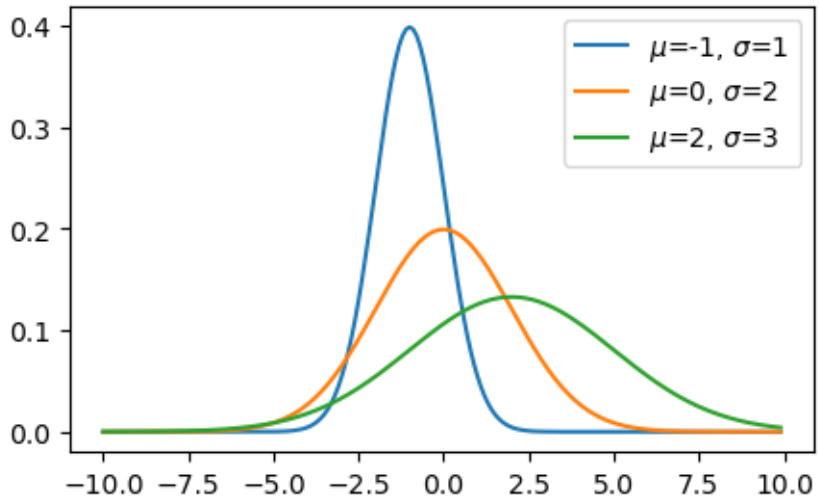
- Additive
- Easy to model
- Law of large numbers

Distribution function

$$n(x) = \frac{1}{\sqrt{2\pi\sigma}} \exp - \left(\frac{x-\mu}{2\sigma}\right)^2$$

Below you see plots of the Gaussian distribution with different parameters.

```
from scipy.stats import norm
rv = norm(loc = -1., scale = 1.0);rv1 = norm(loc = 0., scale = 2.0); rv2 = norm(loc =␣
 ↪2., scale = 3.0)
x = np.arange(-10, 10, .1)
plt.figure(figsize=(5,3))
#plot the pdfs of these normal distributions
plt.plot(x, rv.pdf(x),label='$\mu$=-1, $\sigma$=1')
plt.plot(x, rv1.pdf(x),label='$\mu$=0, $\sigma$=2')
plt.plot(x, rv2.pdf(x),label='$\mu$=2, $\sigma$=3')
plt.legend();
```



Noise models - Poisson noise

The Poisson noise is the central noise model for event counting processes. It is thus the type of noise you see in imaging as the detectors in some sense is counting the number of particles arriving at the detector, e.g. photons or neutrons. This noise distribution changes shape with increasing number of particles; the distribution is clearly asymmetric for few particles while it takes a Gaussian shape when many particles are counted. It is also multiplicative in contrast to the Gaussian noise. This is in other words the noise distribution you need to use if you want to model image noise correctly.

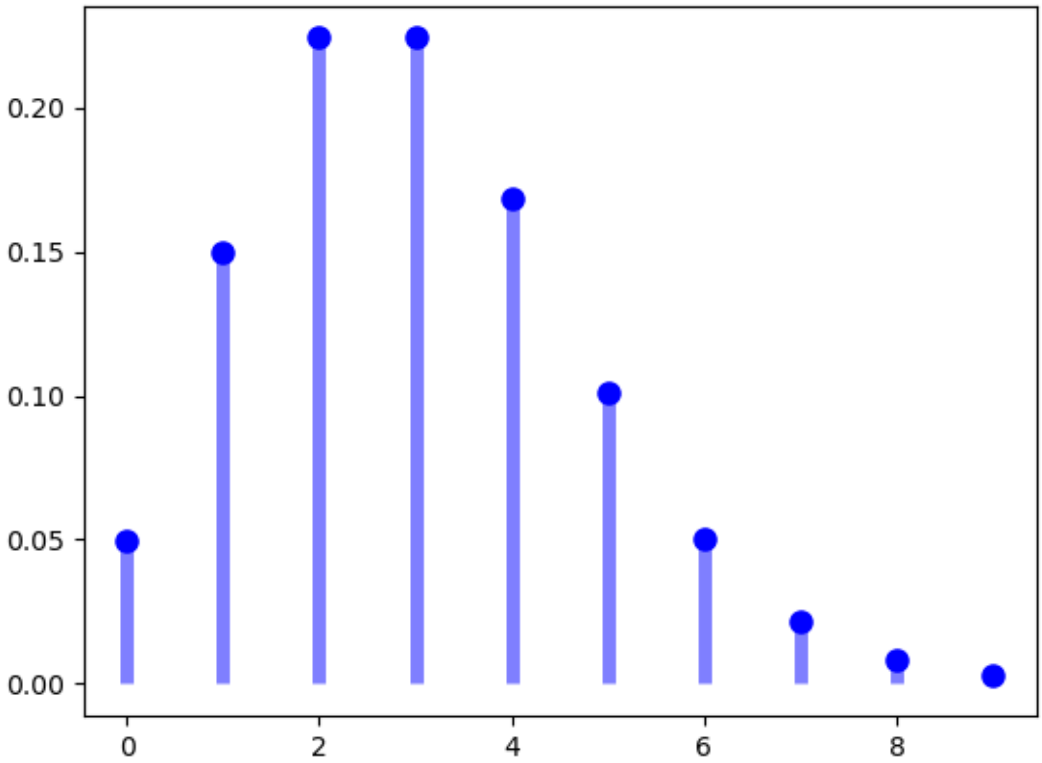
- Intensity dependent
- Physically correct for event counting

Distribition function

$$p(x) = \frac{\lambda^k}{k!} e^{-\lambda x}$$

The plot below show a poisson distribtion for λ=3

```
from scipy.stats import poisson
mu=3
fig, ax = plt.subplots(1, 1)
x = np.arange(poisson.ppf(0.01, mu),              poisson.ppf(0.999, mu))
ax.plot(x, poisson.pmf(x, mu), 'bo', ms=8, label='poisson pmf')
ax.vlines(x, 0, poisson.pmf(x, mu), colors='b', lw=5, alpha=0.5);
```



Compare Gaussian and Possion noise

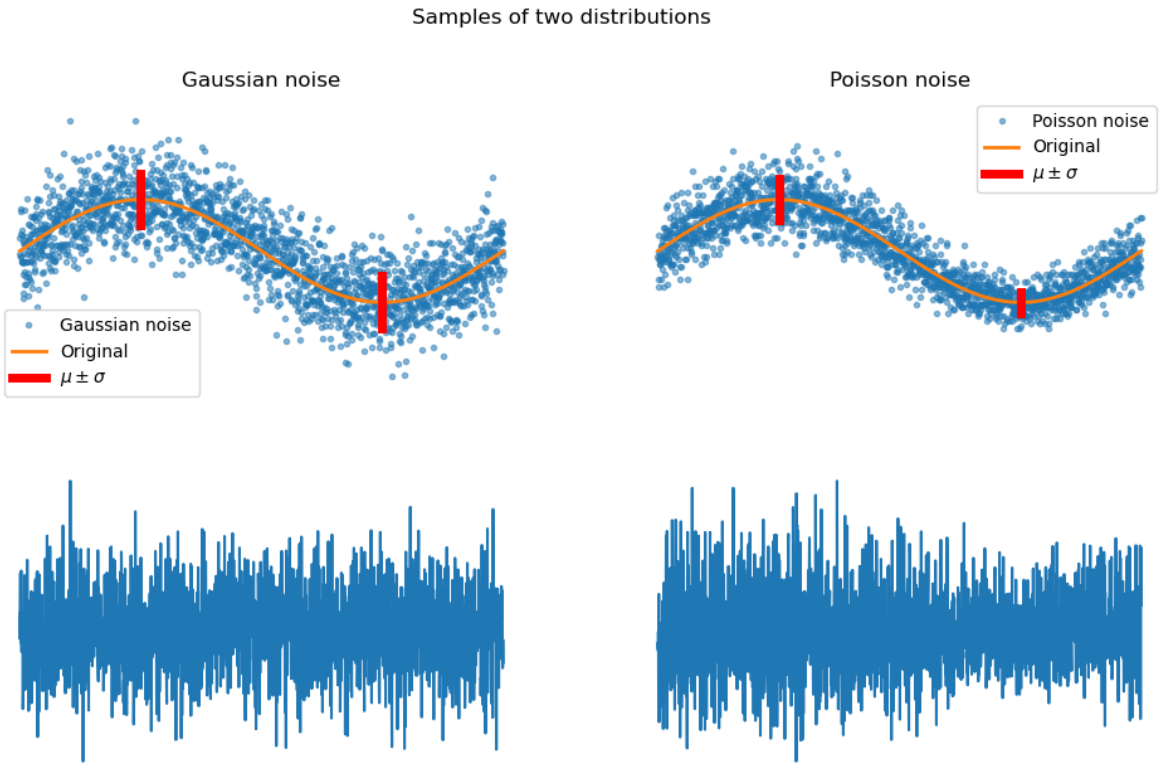
Now, let's compare relaizations of Gaussian and Poisson noise overlaid on a sine curve. The important thing to observe is that the noise amplitude is independent of the signal value and constant for the Gaussian noise. For Poisson noise it is very different. The noise amplitude changes with the signal values. Higher signal strength also produces greater noise amplitudes.

```
fig,ax=plt.subplots(2,2,figsize=(12,7))
ax=ax.ravel()

x=np.linspace(0,2*np.pi,2000)
y=1*np.sin(x)+2

idxmax = np.argmax(y)
idxmin = np.argmin(y)


np.random.seed(42)
ng=np.random.normal(0,0.5,size=len(x))
gn=y+ng
ax[0].plot(x,gn,'.',label='Gaussian noise',alpha=0.5)
ax[0].plot(x,y,label='Original',lw=2)
ax[0].set_ylim([-1,5])
ax[0].set_title('Gaussian noise')
stdmin = gn[(idxmin-100):(idxmin+100)].std()
stdmax = gn[(idxmax-100):(idxmax+100)].std()
ax[0].plot([x[idxmin],x[idxmin]],[y[idxmin]+stdmin,y[idxmin]-stdmin],lw=5,c='red',
 ↪label=r"$\mu \pm \sigma$")
```

(continues on next page)

(continued from previous page)

```
ax[0].plot([x[idxmax],x[idxmax]],[y[idxmax]+stdmax,y[idxmax]-stdmax],lw=5,c='red')
ax[0].legend()
ax[0].axis('off')

pn=np.random.poisson(y*20)/20

ax[1].plot(x,pn,'.',label='Poisson noise',alpha=0.5)
ax[1].plot(x,y,label='Original',lw=2)
ax[1].set_ylim([-1,5])
ax[1].set_title('Poisson noise');
ax[1].axis('off')

stdmin = pn[(idxmin-100):(idxmin+100)].std()
stdmax = pn[(idxmax-100):(idxmax+100)].std()
ax[1].plot([x[idxmin],x[idxmin]],[y[idxmin]+stdmin,y[idxmin]-stdmin],lw=5,c='red',
 ↪label=r"$\mu \pm \sigma$")
ax[1].plot([x[idxmax],x[idxmax]],[y[idxmax]+stdmax,y[idxmax]-stdmax],lw=5,c='red')
ax[1].legend();

ax[2].plot(x,ng);ax[2].axis('off');
ax[3].plot(x,pn-y);ax[3].axis('off');
plt.suptitle('Samples of two distributions');
```



Noise models - Salt'n'pepper noise

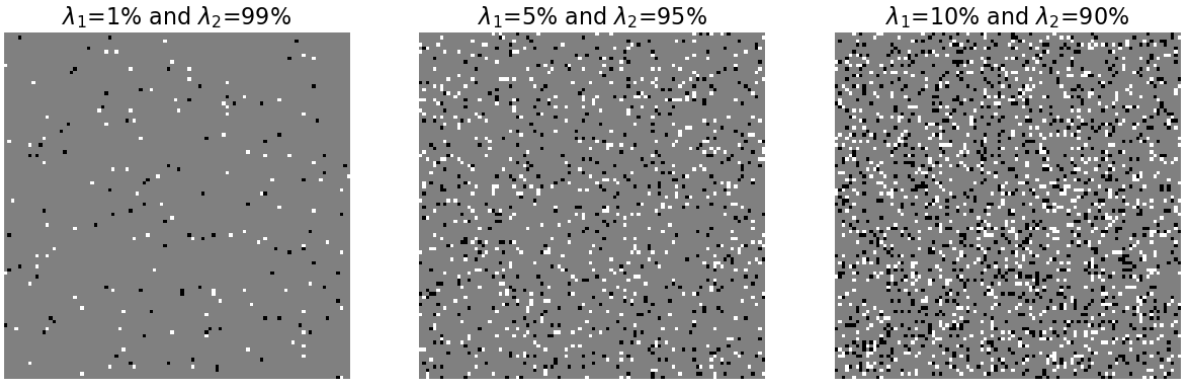
- A type of outlier noise
- Noise frequency described as probability of outlier
- Can be additive, multiplicative, and independent replacement

Example model $sp(x) = \begin{cases} -1 & x \leq \lambda_1 \\ 0 & \lambda_1 < x \leq \lambda_2 \\ 1 & \lambda_2 < x \end{cases}$ $x \in \mathcal{U}(0,1)$ $\lambda_1 < \lambda_2$ $\lambda_1 + \lambda_2 =$ noise fraction $

Salt'n'pepper examples

```
def snp(dims,Pblack,Pwhite) : # Noise model function
    uni=np.random.uniform(0,1,dims)
    img=(Pwhite<uni).astype(float)-(uni<Pblack).astype(float)
    return img
```

```
img10_90=snp([100,100],0.1,0.9); img5_95=snp([100,100],0.05,0.95);img1_99=snp([100,
 ↪100],0.01,0.99)
plt.figure(figsize=[15,5])
plt.subplot(1,3,1); plt.imshow(img1_99,cmap='gray'); plt.title('$\lambda_1$=1% and $\
 ↪lambda_2$=99%',fontsize=16); plt.axis('off');
plt.subplot(1,3,2); plt.imshow(img5_95,cmap='gray'); plt.title('$\lambda_1$=5% and $\
 ↪lambda_2$=95%',fontsize=16); plt.axis('off');
plt.subplot(1,3,3); plt.imshow(img10_90,cmap='gray'); plt.title('$\lambda_1$=10% and
 ↪$\lambda_2$=90%',fontsize=16); plt.axis('off');
```



0.3.2 Signal to noise ratio

It is important to know how strong the noise is compared to the signal in order to decide how to proceed with the analysis. Therefore, we need a metric to quantify the noise.

The Signal to noise ratio measures the noise strengh in a signal

Definition

$$SNR = \frac{mean(f)}{stddev(f)}$$

Sometimes the term contrast to noise ratio is also used. This means that you measure the intensity difference in between two relevant features and divide this by the noise.

Signal to noise ratio for Poisson noise

The SNR of poisson noise is particularly easy to compute because $E[x] = v[x]$. This means that the SNR is proportional to the square root of the number of particles.

- For a Poisson distribution the SNR is :

$$SNR = \frac{E[x]}{s[x]} \sim \frac{N}{\sqrt{N}} = \sqrt{N}$$

- N is the number of particles $\sim$ exposure time

where N is the number of captured particles. The figure below shows two neutron images acquired at 0.1s and 10s respectively. The plot shows the signal to noise ratio obtained for different exposure times.

The signal to noise ratio can be improved by increasing the number of neutrons per pixel. This can be achived through increasing

- Neutron flux - this is usually relatively hard as the neutron sources operate with the parameters it is designed for. There is a posibilty by changing the neutron aperture, but has an impact of the beam quality.
- Exposure time - the exposure time can be increased but in the end there is a limitation on how much this can be used. Beam time is limited which means the experiment must be finished in a given time. There is also an upper limit on the exposure time defined by the observed sample or process when it changes over time. Too long exposure times will result in motion artefacts.
- Pixel size - increasing the pixel size means that neutrons are collected over a greater area and thus more neutrons are captured during the exposure. The limit on how much you can increase the pixel size is defined by the smallest features you want to detect.
- Detector material and thickness - the number of captured neutrons depends on the scintillator material and how thick it is. The thickness does however have an impact on the resolution. Therefore scintillator thickness and pixel size often increase in parallel as there is no point in oversampling a smooth signal to much.

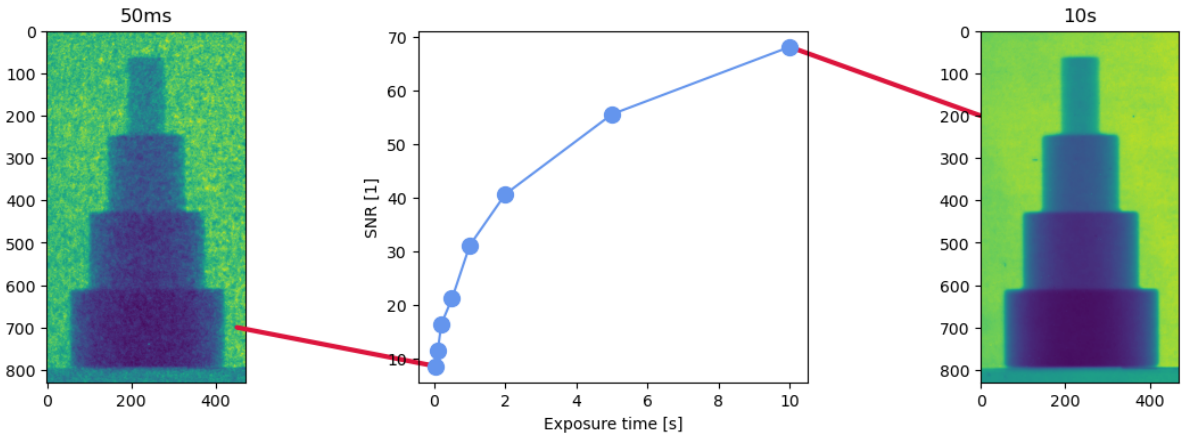
In the end, there are many parameters that combined results in the SNR you obtain. These parameters are tuned to match the experiment conditions. The filtering techniques presented in this lecture can help to increase the SNR and hopefuly make the way for a quantitaive analysis.

```
exptime=np.array([50,100,200,500,1000,2000,5000,10000])
snr = np.array([ 8.45949767, 11.40011621, 16.38118766, 21.12056507, 31.09116641,40.
↪65323123, 55.60833117, 68.21108979]);
marker_style = dict(color='cornflowerblue', linestyle='-', marker='o',markersize=10,␣
↪markerfacecoloralt='gray');
fig,ax = plt.subplots(1,3,figsize=(15,4))

ax[1].plot(exptime/1000,snr, **marker_style);
ax[1].set_xlabel('Exposure time [s]');
ax[1].set_ylabel('SNR [1]')
img50ms    = plt.imread('figures//tower_50ms.png');
img10000ms = plt.imread('figures/tower_10000ms.png');

ax[0].imshow(img50ms);
ax[0].set_title('50ms');

ax[2].imshow(img10000ms)
ax[2].set_title('10s');

con0 = ConnectionPatch(xyA=(450,700), xyB=(exptime[0]/1000,snr[0]),
                       coordsA="data", coordsB="data",
                       axesA=ax[0], axesB=ax[1],
```

(continues on next page)

(continued from previous page)

```
                        color="crimson", lw=3)
ax[1].add_artist(con0)

con2 = ConnectionPatch(xyA=(0,200), xyB=(exptime[-1]/1000,snr[-1]),
                        coordsA="data", coordsB="data",
                        axesA=ax[2], axesB=ax[1],
                        color="crimson", lw=3)
ax[1].add_artist(con2);
```



0.3.3 Useful python functions

Random number generators [numpy.random]

Generate an $m \times n$ random fields with different distributions:

- **Gauss** `np.random.normal(mu,sigma, size=[rows,cols])`
- **Uniform** `np.random.uniform(low,high,size=[rows,cols])`
- **Poisson** `np.random.poisson(lambda, size=[rows,cols])`

Statistics

- `np.mean(f)`, `np.var(f)`, `np.std(f)` Computes the mean, variance, and standard deviation of an image f.
- `np.min(f)`,`np.max(f)` Finds minimum and maximum values in f.
- `np.median(f)`, `np.rank()` Selects different values from the sorted data.

0.4 Basic filtering

0.4.1 What is a filter?

In general terms a filter is a component that separates mixed components form each other. In chemistry you use a filter to separate solid patricles from liquid. In signal processing the filter is used to separate frequencies in signals. In image processing we are not only talking about frequencies but also structures. This is something we will look into in a few weeks when we talk about morphological image processing.

General definition

A filter is a processing unit that

- Enhances the wanted information
- Suppresses the unwanted information

  Ideally without altering relevant features beyond recognition .

The filter should ideally perform the task it is meant to do, but at the same time maintain the information we want to keep. This is often a difficult problem, in particular with traditional filters. They apply the same characteristics to any pixel without concerning what this pixel actually represents. It only sees frequencies! As a consequence you may cancel all relevant information in the image in your mission to remove the noise.

0.4.2 Filter characteristics

Filters are characterized by the type of information they suppress or amplify.

In signal and image processing filters are applied to modify the amplitudes of different frequencies in the signal. Slow variations have low frequencies while rapid variations have high frequencies.

Low-pass filters

Low pass filters are designed to suppress frequencies in the upper part of the spectrum in order to better show slow changes in the images. The effect of a lowpass filter is that the images are blurred.

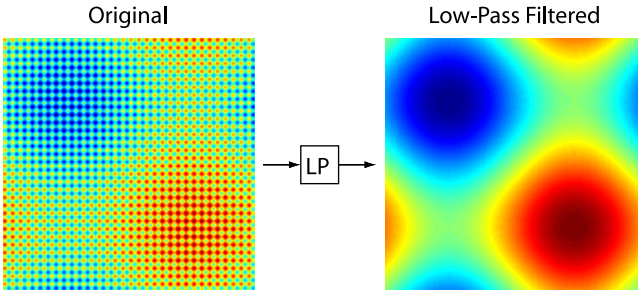


Fig. 1: The principle of a low-pass filter.

- Slow changes are enhanced
- Rapid changes are suppressed

High-pass filters

High pass filters are the opposite of the low pass filters as the name suggests. The suppress low frequency components of the spectrum leaving the rapid changes untouched. This is an important filter for edge detection as we will see later.

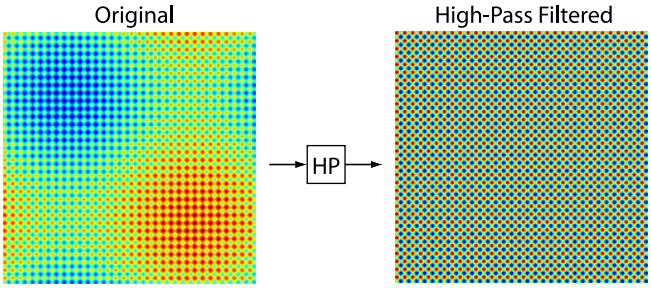


Fig. 2: The principle of a high-pass filter.

- Rapid changes are enhanced
- Slow changes are suppressed

0.5 Basic filters

0.5.1 Linear filters

Computed using the convolution operation

$$g(x) = h * f(x) = \int_{\Omega} f(x-\tau)h(\tau)d\tau$$

where

- f is the image
- h is the convolution kernel of the filter

0.5.2 Low-pass filter kernels

The most common linear filters used in image processing are the box filter and the Gauss filter. The box filter has a kernel where all filter weights have the same strength. This filter essentially computes the local average of the neighborhood it covers. A 5x5 box filter looks like this

$$B = \frac{1}{25} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

The scaling by the number of weights is sometimes omitted, but this would mean that the intensity of the resulting image is upscaled by this factor.

The Gauss filter kernel has its weights from the N-Dimensional Gauss function. Here, a 2D kernel:

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

The Gauss function is a continuous function that extends to infinity. Therefore, we have to define the size of the discrete kernel. A good choise is $N = 2 \cdot \lceil 2\sigma \rceil + 1$, multiple of σ can also be set to 2.5 or even 3 but that is almost too much because the boundary weights are very small compared to the central value.

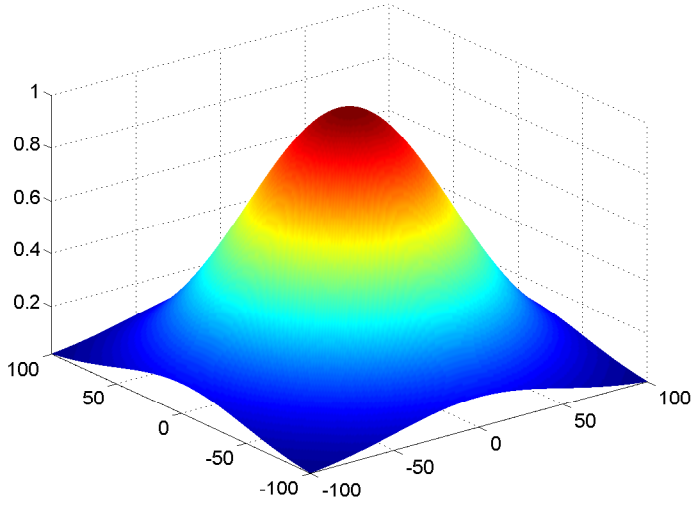


Fig. 1: The shape of a Gaussian filter kernel.

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Example: $B = \frac{1}{25} \cdot$

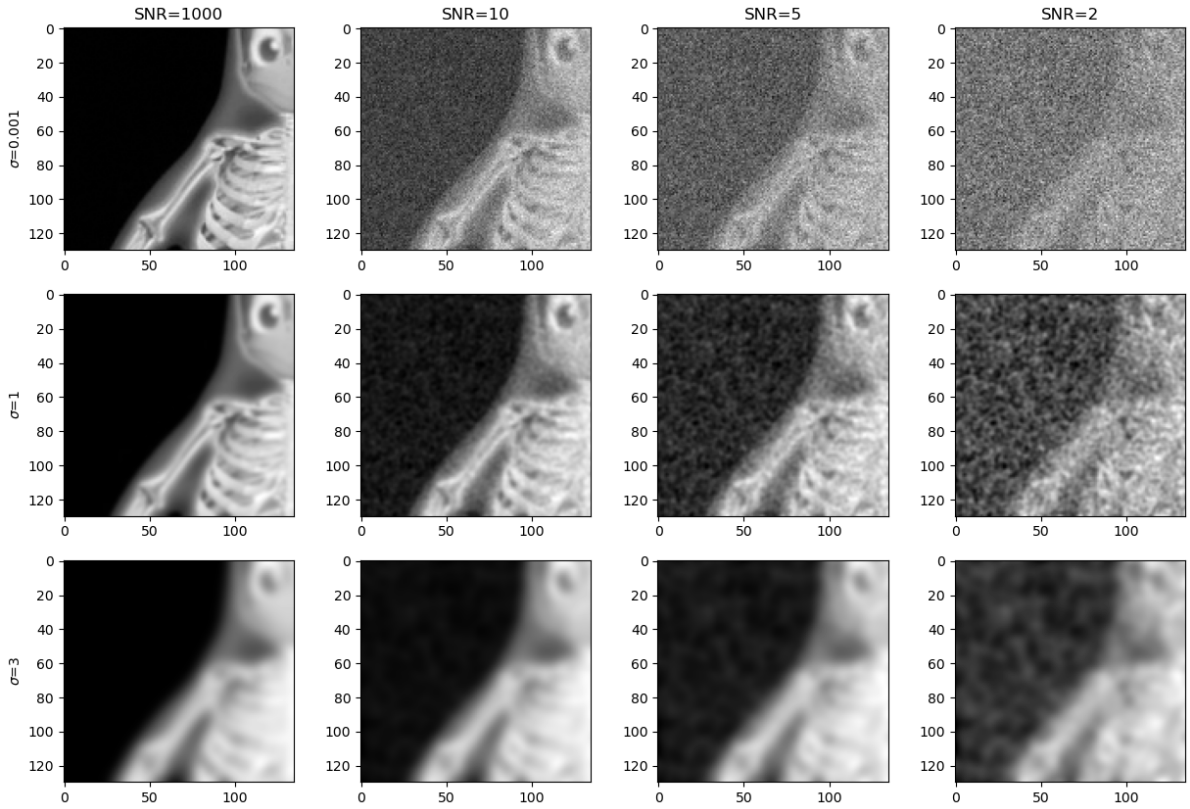
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Example:

0.5.3 Different SNR using a Gauss filter

The main purpose of lowpass filters is to reduce the noise in the images. In the following example you can see images with different SNR and what happens when you apply Gauss filters with different σ.

```
fig, ax = plt.subplots(3,4,figsize=(15,10)); ax=ax.ravel()
img   = plt.imread('figures/input_orig.png');
noise = np.random.normal(0,1,size=img.shape);
SNRs   = [1000,10, 5, 2]
sigmas = [0.001,1,3]
for r,sigma in enumerate(sigmas) :
    for c,SNR in enumerate(SNRs) :
        ax[r*(len(SNRs))+c].imshow(ski.filters.gaussian(img+noise/SNR,sigma=sigma),
 ↪cmap='gray');
    ax[r*(len(SNRs))].set_ylabel('$\sigma$={0}'.format(sigma))
for c,SNR in enumerate(SNRs) :
    ax[c].set_title('SNR={0}'.format(SNR))
```



What you see in the example is that you will need a wider filter kernel to reduce noise in images with low SNR. The cost of the SNR improvement is unfortunately that fine details in the image are also blurred by the operation. From this example we can conclude that linear filters can't be applied with out consequences for the image content. Therefore, we have to carefully select filter kernel balancing the improvement in SNR against loss of image features.

0.5.4 How is the convolution computed

Before, we saw that the convolution was comouted using an integral. This is however the definition for continuous variables. In image processing, we change the integral into a sum instead. The convolution is then a weighted sum of the neighborhood pixels. The example below shows how a pixel is updated using a box kernel with the size 3x3. This operation is repeated for all pixels or voxels in the image.

For a non-uniform kernel each term is weighted by its kernel weight.

0.5.5 Euclidean separability

The convolution involves quite many additions and multiplications to perform. This can be a bottleneck in a complicated processing workflow. Fortunately, some tricks can be used to recude the number of operations needed for the convolution.

The asociative and commutative laws apply to convoution

$$(a * b) * c = a * (b * c) \quad \text{and} \quad a * b = b * a$$

A convolution kernel is called *separable* if it can be split in two or more parts.

This is the case for the two filter kernels we have seen until now:

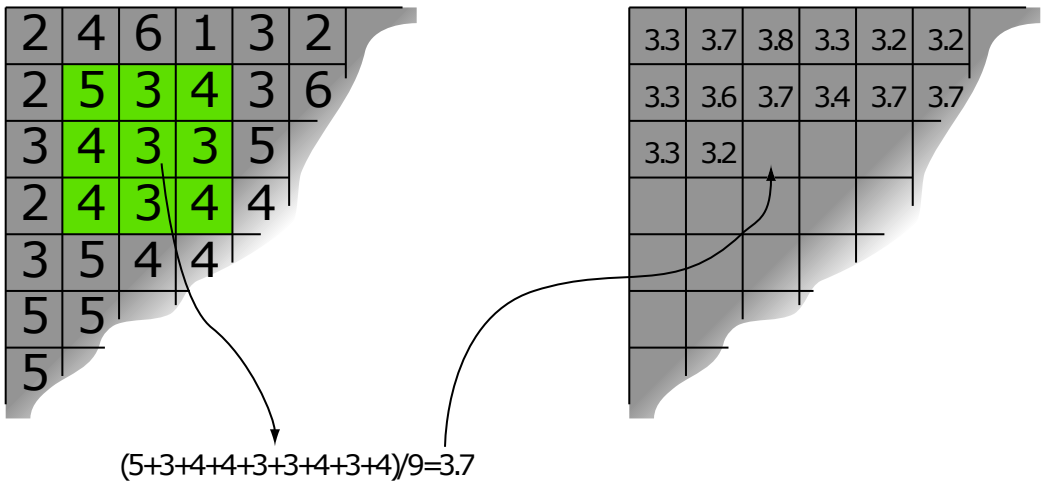


Fig. 2: Updating one pixel using a 3x3 box filter.

Examples of separable kernels

Box

$$\begin{bmatrix} \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & \cdot \end{bmatrix} = \begin{bmatrix} \cdot \\ \cdot \\ \cdot \end{bmatrix} * \begin{bmatrix} \cdot & \cdot & \cdot \end{bmatrix}$$

Gauss

$$\exp -\frac{x^2+y^2}{2\sigma^2} = \exp -\frac{x^2}{2\sigma^2} * \exp -\frac{y^2}{2\sigma^2}$$

Gain of using separable kernels

Let's see what it brings to split the kernels in to the principal directions of the image.

Separability reduces the number of computations $\rightarrow$ faster processing

- $3\times3 \rightarrow$ 9 mult and 8 add $\Leftrightarrow$ 6 mult and 4 add
- $3\times3\times3 \rightarrow$ 27 mult and 26 add $\Leftrightarrow$ 9 mult and 6 add

The gain is moderate in the 3x3 case, but if we increase the kernel size to 5 instead we see that the gain is increasing radically:

- $5\times5 \rightarrow$ 25 mult and 24 add $\Leftrightarrow$ 10 mult and 8 add
- $5\times5\times5 \rightarrow$ 125 mult and 124 add $\Leftrightarrow$ 15 mult and 12 add

This looks very promising and it would be great. There are however some things to consider:

- Overhead to call the filter function may consume some of the time gain from the separability.
- The result may deviate a little depending on the used numerical precision used.

0.5.6 The median filter

The median filter is a non-linear filter with low-pass characteristics. This filter computes the local median using the pixels in the neighborhood and uses this value in the filtered image.

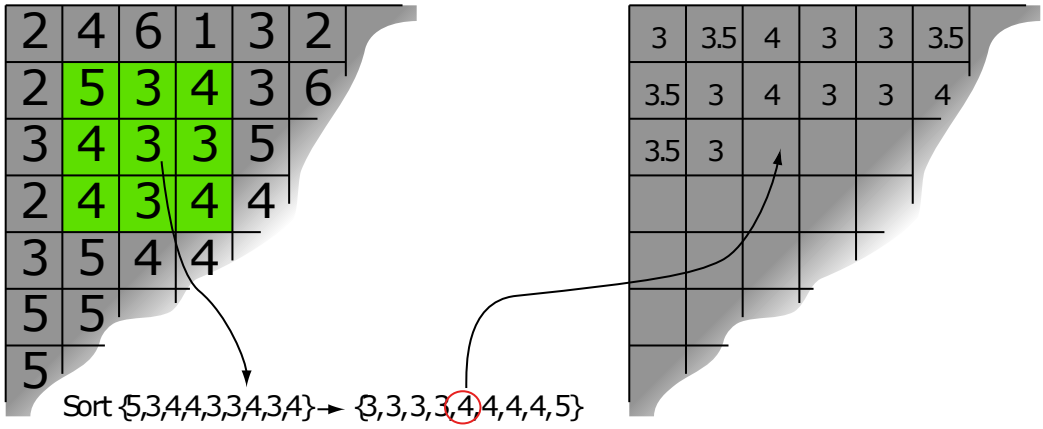
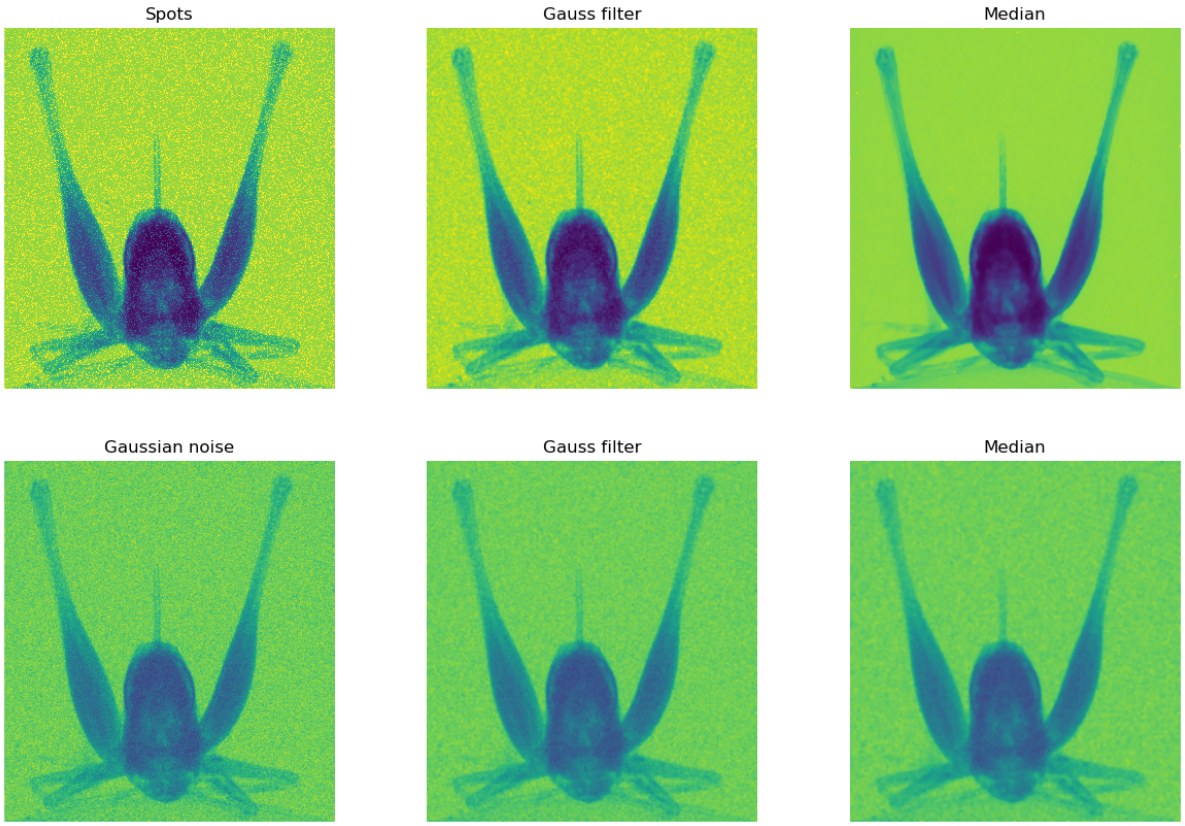


Fig. 3: Updating a pixel from its neighborhood using a 3x3 median filter.

0.5.7 Comparing filters for different noise types

Both linear low-pass filters and the median filter have SNR improving charactersitics. They do, however, differ in which noise types they are suited for. In the example below you see an image with add Gaussian noise and salt'n'pepper noise.

```
img = plt.imread('figures/grasshopper.png'); noise=img+np.random.normal(0,0.1,
 ↪size=img.shape); spots=img+0.2*snp(img.shape,0,0.8); noise=(noise-noise.min())/
 ↪(noise.max()-noise.min());spots=(spots-spots.min())/(spots.max()-spots.min());
plt.figure(figsize=[15,10]); vmin=0.0; vmax=0.9;
cmap='viridis'
plt.subplot(2,3,1); plt.imshow(spots,vmin=vmin,vmax=vmax,cmap=cmap,interpolation='none
 ↪'); plt.title('Spots'); plt.axis('off');
plt.subplot(2,3,2); plt.imshow(ski.filters.gaussian(spots,sigma=1),vmin=vmin,
 ↪vmax=vmax,cmap=cmap,interpolation='none'); plt.title('Gauss filter'); plt.axis('off
 ↪');
plt.subplot(2,3,3); plt.imshow(ski.filters.median(spots,disk(3)),vmin=vmin,vmax=vmax,
 ↪cmap=cmap); plt.title('Median'); plt.axis('off');

plt.subplot(2,3,4); plt.imshow(noise,vmin=vmin,vmax=vmax,cmap=cmap); plt.title(
 ↪'Gaussian noise'); plt.axis('off');
plt.subplot(2,3,5); plt.imshow(ski.filters.gaussian(noise,sigma=1),vmin=vmin,
 ↪vmax=vmax,cmap=cmap); plt.title('Gauss filter'); plt.axis('off');
plt.subplot(2,3,6); plt.imshow(ski.filters.median(noise,disk(3)),vmin=vmin,vmax=vmax,
 ↪cmap=cmap); plt.title('Median'); plt.axis('off');
```



In this comparison, you can clearly see that the median filter is supperior when it comes to remove outliers in an image. In this case, the Gauss filter is only smearing out the outliers, but they still appear as noisy.

In the case of Gaussian noise, it is harder to tell which filter to use. Many are using the median filter as their main goto choise to filter images. Because it is usually gentler to edges and is also good at removing spots. Still, you should be aware of the additional processing time and also that the statistical distribution of the data is not following the original model anymore. The distribution of the data may not be of any concern in many cases, but if you are trying to process the image further based on its distribution, this may result in less good results.

0.5.8 Filter example: Spot cleaning

- Many neutron images are corrupted by spots that confuse following processing steps.
- The amount, size, and intensity varies with many factors.
- Low pass filter
- Median filter
- Detect spots and replace by estimate

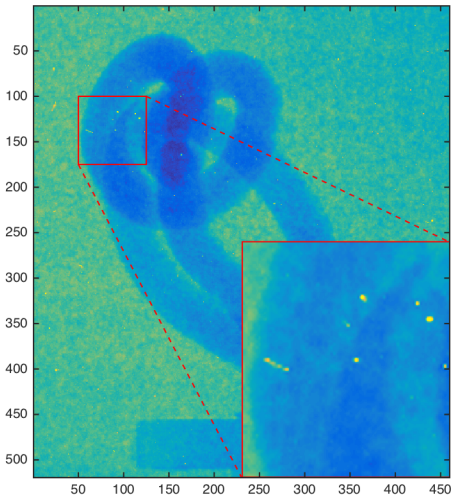


Fig. 4: A neutron image with outliers that we want to remove.

0.5.9 Spot cleaning algorithm

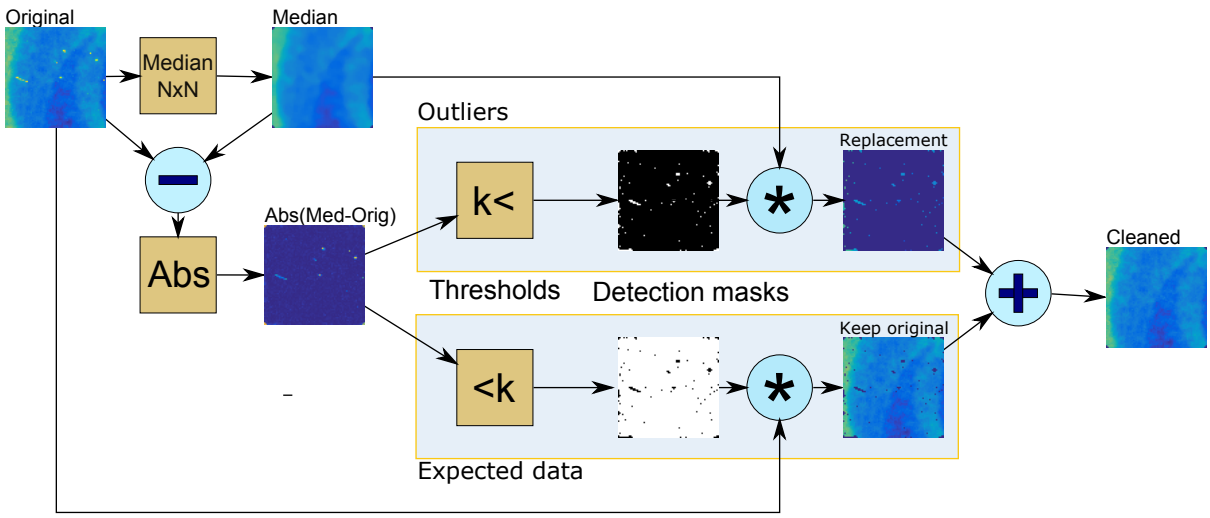


Fig. 5: Process workflow to selectively remove spots from an image.

Parameters

- N Width of median filter.
- k Threshold level for outlier detection.

0.5.10 Spot cleaning - Compare performance

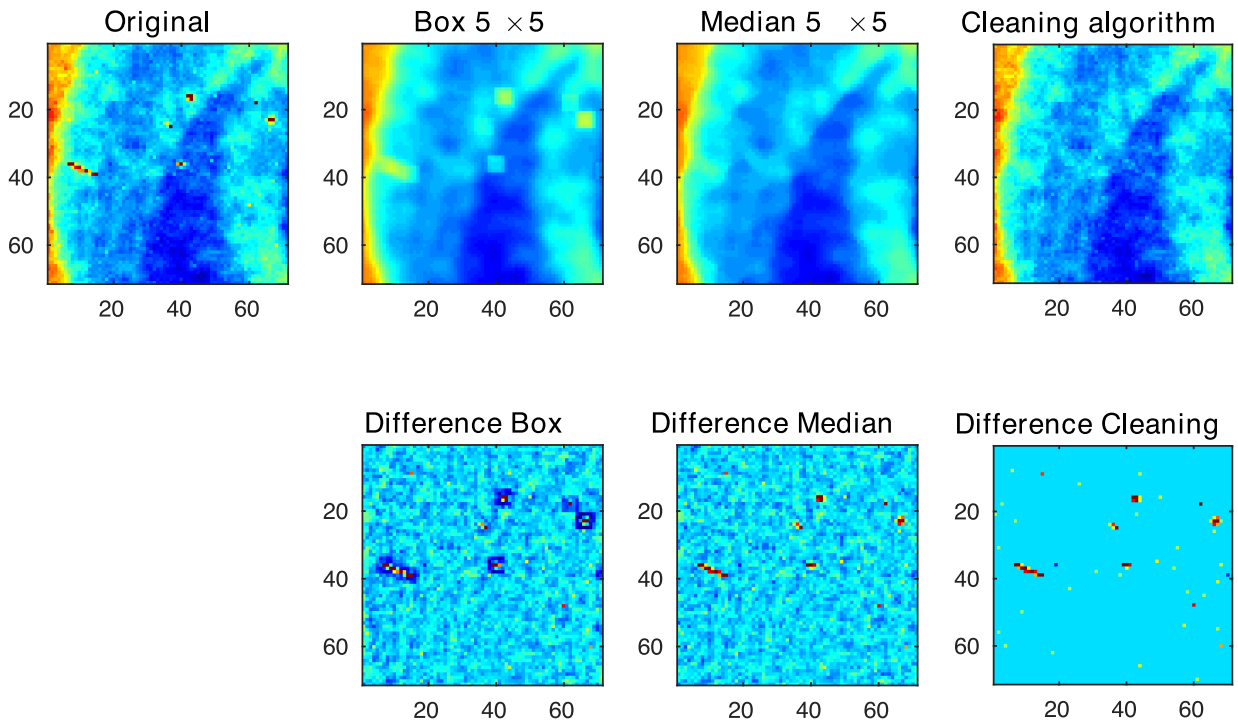


Fig. 6: Comparing different filters to remove spots from an image.

The ImageJ ways for outlier removal

ImageJ is a popular application for interactive image analysis. It offers two ways to remove outliers in the noise menu:

- **Despeckle** Median … please avoid this one!!!
- **Remove outliers** Similar to cleaning described algorithm

0.5.11 High-pass filters

High-pass filters enhance rapid changes → ideal for edge detection

Typical high-pass filters:

Gradients $\frac{\partial}{\partial x} = \frac{1}{2} \cdot \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array}$

$$\frac{\partial}{\partial x} = \frac{1}{32} \cdot \begin{array}{|c|c|c|}\hline -3 & 0 & 3 \\ \hline -10 & 0 & 10 \\ \hline -3 & 0 & 3 \\ \hline \end{array}$$

Laplacian

$\triangle = \frac{1}{2} \cdot$

1	2	1
2	−12	2
1	2	1

Sobel - Magnitude of grandient

$$G = |\nabla f| = \sqrt{\left(\frac{\partial}{\partial x} f\right)^2 + \left(\frac{\partial}{\partial y} f\right)^2}$$

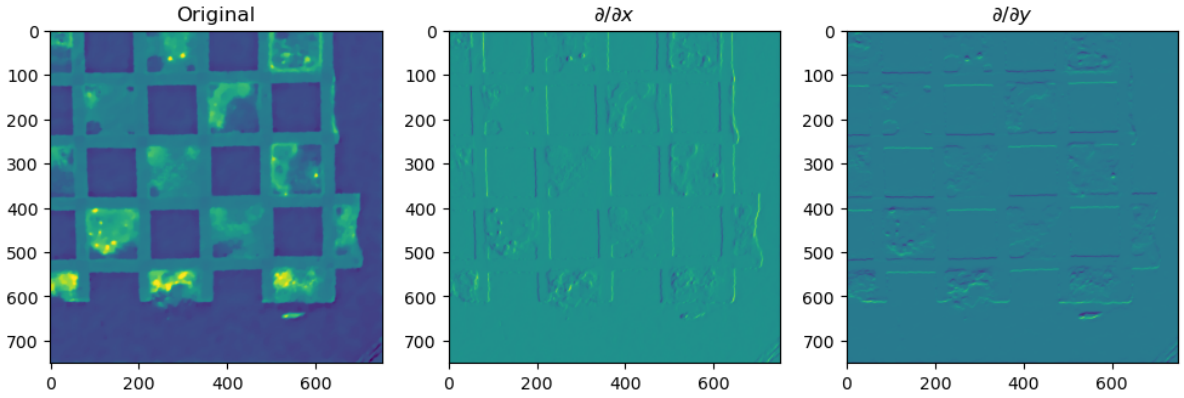
0.5.12 Gradient example - structure orientation

Vertical edges $\$\frac{\partial}{\partial x} = \frac{1}{32} \cdot \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}\$$

Horizontal edges $\$\frac{\partial}{\partial y} = \frac{1}{32} \cdot \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ 3 & 10 & 3 \end{bmatrix}\$$

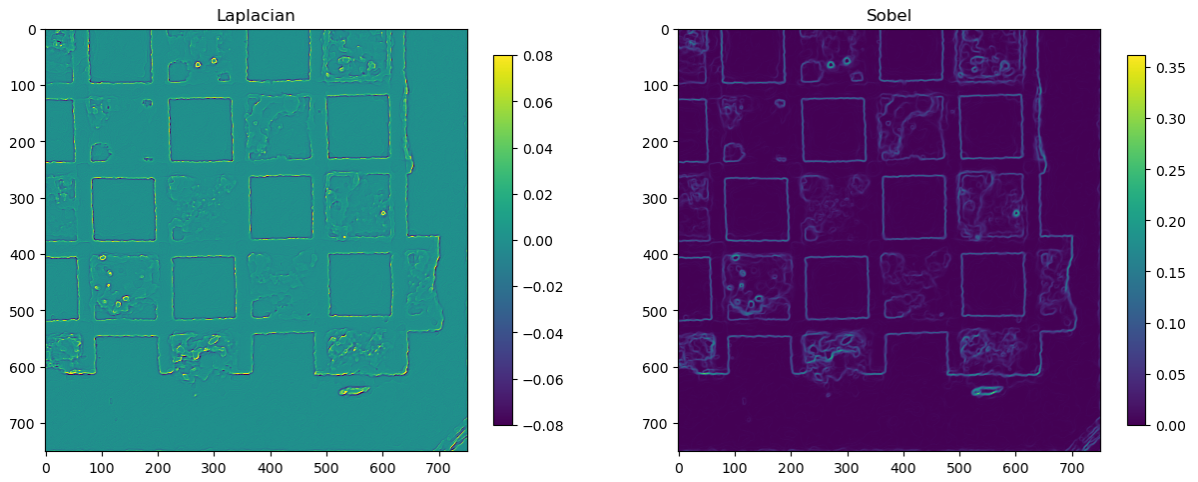
Jaehne, 2005

```
img=plt.imread('figures/orig.png')
k = np.array([[-3,-10,-3],[0,0,0],[3,10,3]]);
plt.figure(figsize=[12,5])
plt.subplot(1,3,1); plt.imshow(img);plt.title('Original');
plt.subplot(1,3,2); plt.imshow(ndimage.convolve(img,np.transpose(k)));plt.title('$\
↪partial / \partial x$');
plt.subplot(1,3,3); plt.imshow(ndimage.convolve(img,k));plt.title('$\partial / \
↪partial y$');
```



0.5.13 Edge detection examples

```
img=plt.imread('figures/orig.png');
plt.figure(figsize=[15,6])
plt.subplot(1,2,1);plt.imshow(ski.filters.laplace(img),clim=[-0.08,0.08]); plt.title(
↪'Laplacian'); plt.colorbar(shrink=0.8);
plt.subplot(1,2,2);plt.imshow(ski.filters.sobel(img)); plt.title('Sobel');  plt.
↪colorbar(shrink=0.8);
```



0.5.14 Relevance of filters to machine learning

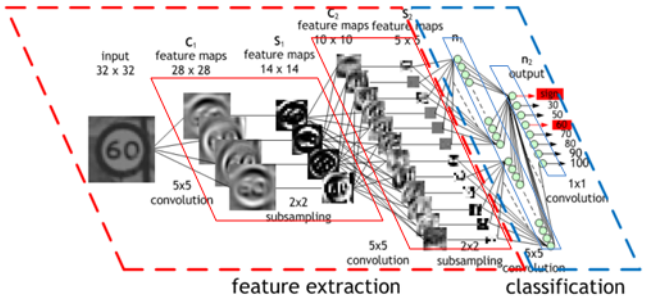


Fig. 7: An example of a convolutional neural network for image classification.

NVIDIA Developer zone

0.6 Frequency space filters

We already mentioned that the there are frequencies to filter in images. There is a whole collection of filters that operate directly in frequency space. This is mostly done using the Fourier transform and its inverse.

0.6.1 Applications of the Fourier transform

0.6.2 The Fourier transform

You may only know the Fourier transform for the 1D case. It is however very easy to increase the number of dimensions. Below you see the 2D Fourier transform and its inverse. These operations are lossless one-to-one, meaning you can go from one domain to the other without loosing any information.

Transform

$$G(\xi_1, \xi_2) = \mathcal{F}\{g\} = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} g(x, y) \exp -i(\xi_1 x + \xi_2 y)\, dx\, dy$$

Inverse

$$g(x,y) = \mathcal{F}^{-1}\{G\} = \frac{1}{(2\,\pi)^2}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} G(\xi_1,\xi_2)\exp i(\xi_1 x + \xi_2 y)d\xi_1 d\xi_2$$

FFT (Fast Fourier Transform)

The previous equations are meant for continuous signals. Images are discrete signals and the integrals turn into sums. This may introduce some numerical losses when the transforms are computed. Computing the DFT is an $O(N^2)$ operation. There is a way to speed this up by using the Fast Fourier Transform algorithm. This algorithm requires that the data has the length $N = 2^k$. In this case the complexity reduces to $O(N \cdot \log(N))$, which is a radical speed-up.

In practice - you never see the transform equations.
The Fast Fourier Transform algorithm is available in numerical libraries and tools. Jaehne, 2005

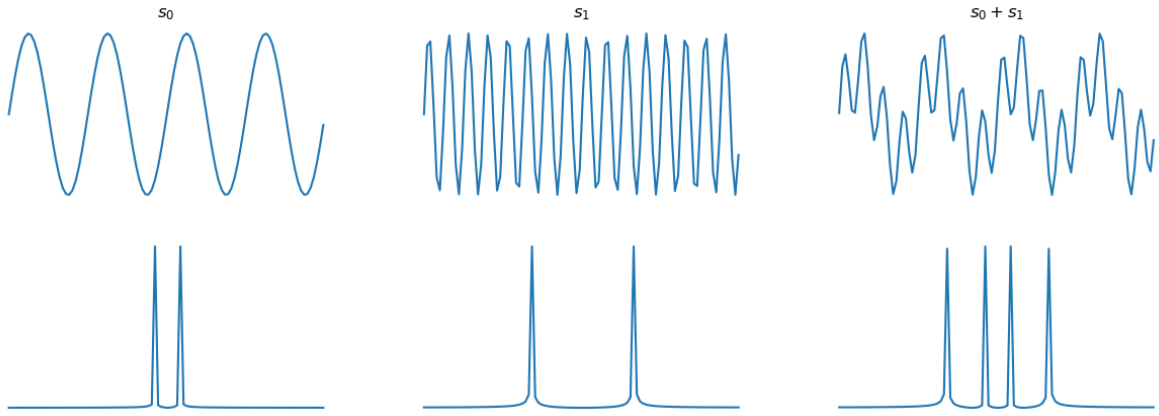
0.6.3 Some mathematical features of the FT

Addition

Addition of Fourier spectra works the same way as for real space signal. The reason is that the transform is based on summarions. You probably proved this as an exercise in your math classes. The plots below shows the spectra of the sum of two signals.

$\mathcal{F}\{a+b\} = \mathcal{F}\{a\} + \mathcal{F}\{b\}$

```
x = np.linspace(0,50,100); s0=np.sin(0.5*x); s1=np.sin(2*x);
plt.figure(figsize=[15,5])
plt.subplot(2,3,1);plt.plot(x,s0); plt.axis('off');plt.title('$s_0$');plt.subplot(2,3,
 ↪4);plt.plot(np.abs(np.fft.fftshift(np.fft.fft(s0))));plt.axis('off');
plt.subplot(2,3,2);plt.plot(x,s1); plt.axis('off');plt.title('$s_1$');plt.subplot(2,3,
 ↪5);plt.plot(np.abs(np.fft.fftshift(np.fft.fft(s1))));plt.axis('off');
plt.subplot(2,3,3);plt.plot(x,s0+s1); plt.axis('off');plt.title('$s_0+s_1$');plt.
 ↪subplot(2,3,6);plt.plot(np.abs(np.fft.fftshift(np.fft.fft(s0+s1))));plt.axis('off');
```



Convolution

Convolution, which is a relatively intense task to perform in real space reduces to a frequency-wise multiplication in the Fourier space. This is a very useful property of the transform. It allows to design some filters much easier than in real space. In particular, band stop filters to remove a specific feature in the image. Also filters with wide kernels can be faster to compute using the Fourier transform.

$\mathcal{F}\{a * b\} = \mathcal{F}\{a\} \cdot \mathcal{F}\{b\}$

alternatively

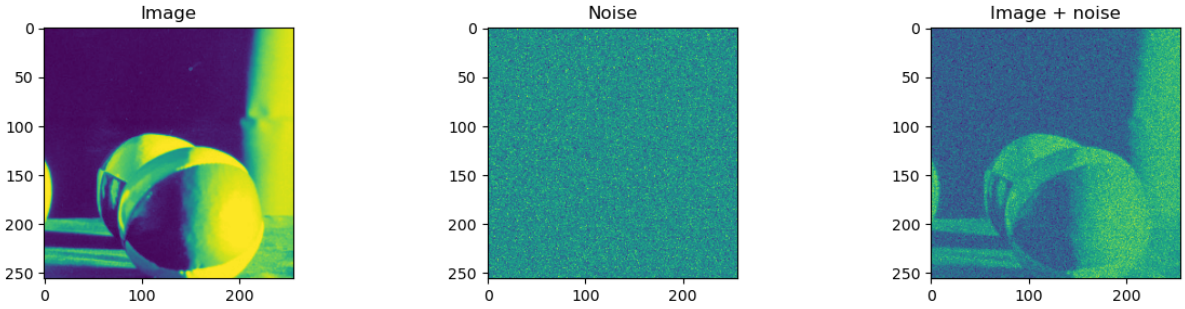
$\mathcal{F}\{a \cdot b\} = \mathcal{F}\{a\} * \mathcal{F}\{b\}$

0.6.4 Additive noise in Fourier space

Real space

```
img   = plt.imread('figures/bp_ex_original.png');
noise = np.random.normal(0,0.2,size=img.shape);
nimg  = img+noise;
```

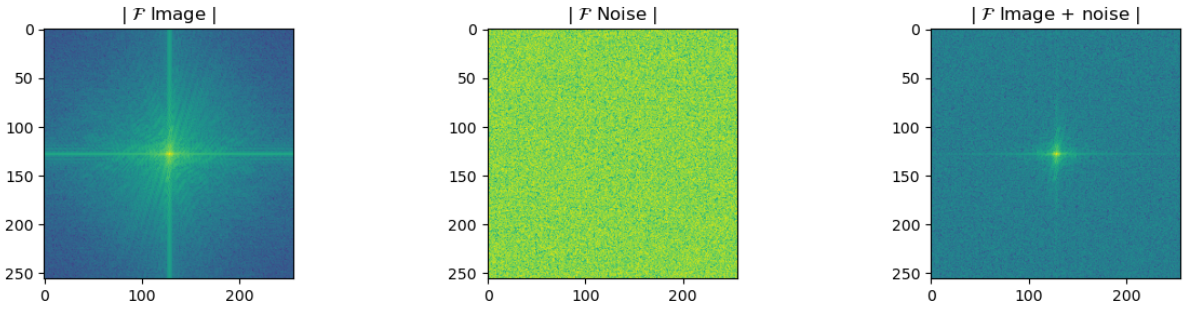
```
#Visualization
fig,ax= plt.subplots(1,3,figsize=[15,3])
ax[0].imshow(img);  ax[0].set_title('Image');
ax[1].imshow(noise);ax[1].set_title('Noise');
ax[2].imshow(nimg); ax[2].set_title('Image + noise');
```



Fourier space

```
fimg   = np.fft.fftshift(np.fft.fft2(img))
fnoise = np.fft.fftshift(np.fft.fft2(noise))
fnimg  = np.fft.fftshift(np.fft.fft2(nimg))
```

```
#Visualization
fig,ax =plt.subplots(1,3,figsize=[15,3])

ax[0].imshow(np.abs(fimg),norm=LogNorm());
ax[0].set_title(r'| $\mathcal{F}$ Image |');
ax[1].imshow(np.abs(fnoise),norm=LogNorm());
ax[1].set_title(r'| $\mathcal{F}$ Noise |');
ax[2].imshow(np.abs(fnimg),norm=LogNorm())
ax[2].set_title(r'| $\mathcal{F}$ Image + noise |');
```



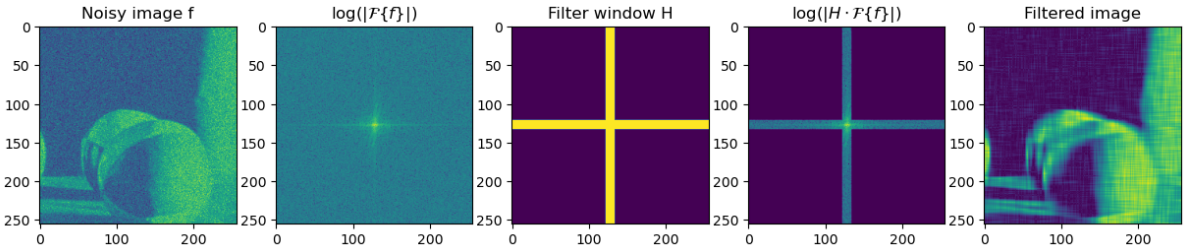
The FFT algorithm result in a cyclic representation of the Fourier transform and the quadrants of the transformed images are permuted such that $\xi_0, \xi_1 = (0, 0)$ is located in the top-left corner and $\xi_0, \xi_1 = (-\epsilon, -\epsilon)$ is located in the lower right corner. The function `fftshift` permutes the quadrants to place the (0,0) in the center of the image.

Convolution in Fourier space

How can we suppress noise without destroying relevant image features?

```
ff=np.fft.fftshift(np.fft.fft2(nimg))
x,y = np.meshgrid(np.linspace(-1,1,ff.shape[1]),np.linspace(-1,1,ff.shape[1]))
R=np.sqrt(x**2+y**2)
H=R<0.5
H=np.exp(-R**2/0.1)
H=np.maximum(np.abs(x)<0.05,np.abs(y)<0.05)
hff=np.fft.ifft2(np.fft.fftshift(H*ff)) # the inverse FFT back to the spatial domain
```

```
fig,ax = plt.subplots(1,5,figsize=(15,5))
ax[0].imshow(nimg)
ax[0].set_title('Noisy image f')
ax[1].imshow(np.log(np.abs(ff)))
ax[1].set_title('log($|\mathcal{F}\{f\}|$)')
ax[2].imshow(H)
ax[2].set_title('Filter window H')
ax[3].imshow(H*np.log(np.abs(ff)))
ax[3].set_title('log($|H\cdot{}\mathcal{F}\{f\}|$)')
ax[4].imshow(np.abs(hff));
ax[4].set_title('Filtered image');
```



Wiener filter in Fourier space

The Wiener filter is a filter that is optimised for the noise model and pointspread function.

$$G(u,v) = \frac{H^*(u,v)P_s(u,v)}{|H(u,v)|^2 P_s(u,v) + P_n(u,v)}$$

- $H(u,v)$ Fourier transform of the point-spread function
- $P_s(u,v)$ Power spectrum of the signal process. Obtained by taking FFT of the signal autocorrelation.
- $P_n(u,v)$ Power spectrum of the noise process. Obtained by taking FFT of the signal autocorrelation.

Special case of the Wiener filter

In the case of additive white Gaussian noise and excluding the PSF, the Wiener filter reduces to

$$G(u,v) = \frac{P_s(u,v)}{P_s(u,v) + \sigma_n^2}$$

Here for

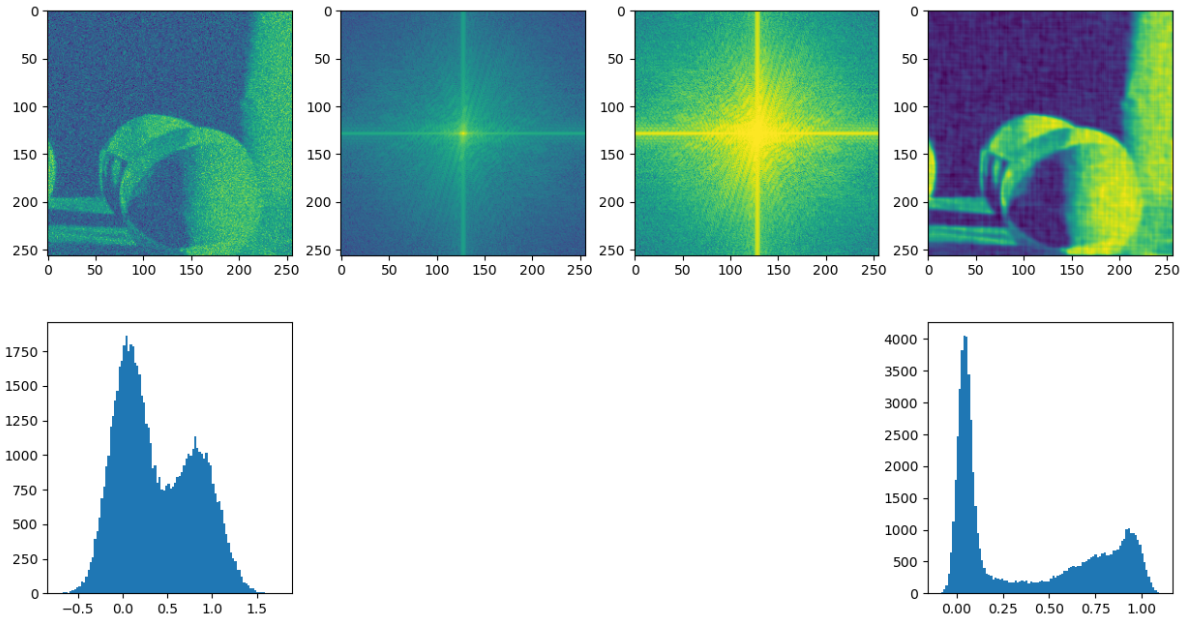
- $P_s(u,v) >> \sigma^2 \Rightarrow G(u,v) = 1$
- $P_s(u,v) << \sigma^2 \Rightarrow G(u,v) = 0$

Problem: We need to know or measure $P_s(u,v)$

Demonstration of the Wiener filter in AWGN

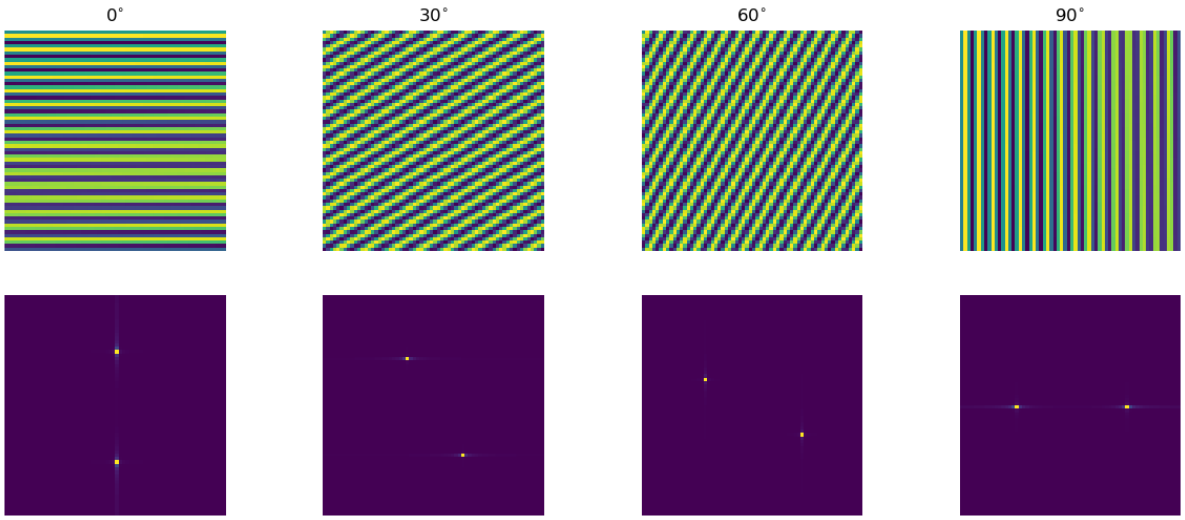
```
fimg=np.fft.fftshift(np.fft.fft2(img))
Ps=fimg*np.conj(fimg)

var=0.1*np.prod(nimg.shape)*noise.var()
G=Ps/(Ps+10*var)
fnimg=np.fft.fftshift(np.fft.fft2(nimg))
filtered=np.real(np.fft.ifft2(np.fft.fftshift(G*fnimg)))
```

```
fig,ax=plt.subplots(2,4,figsize=(15,8))
ax=ax.ravel()
ax[1].imshow(np.abs(Ps),norm=LogNorm())
ax[2].imshow(np.abs(G),norm=LogNorm())
ax[3].imshow(filtered)
ax[0].imshow(nimg)
ax[6].axis('off')
ax[5].axis('off')
ax[7].hist(filtered.ravel(),bins=100);
ax[4].hist(nimg.ravel(),bins=100);
```



0.6.5 Spatial frequencies and orientation

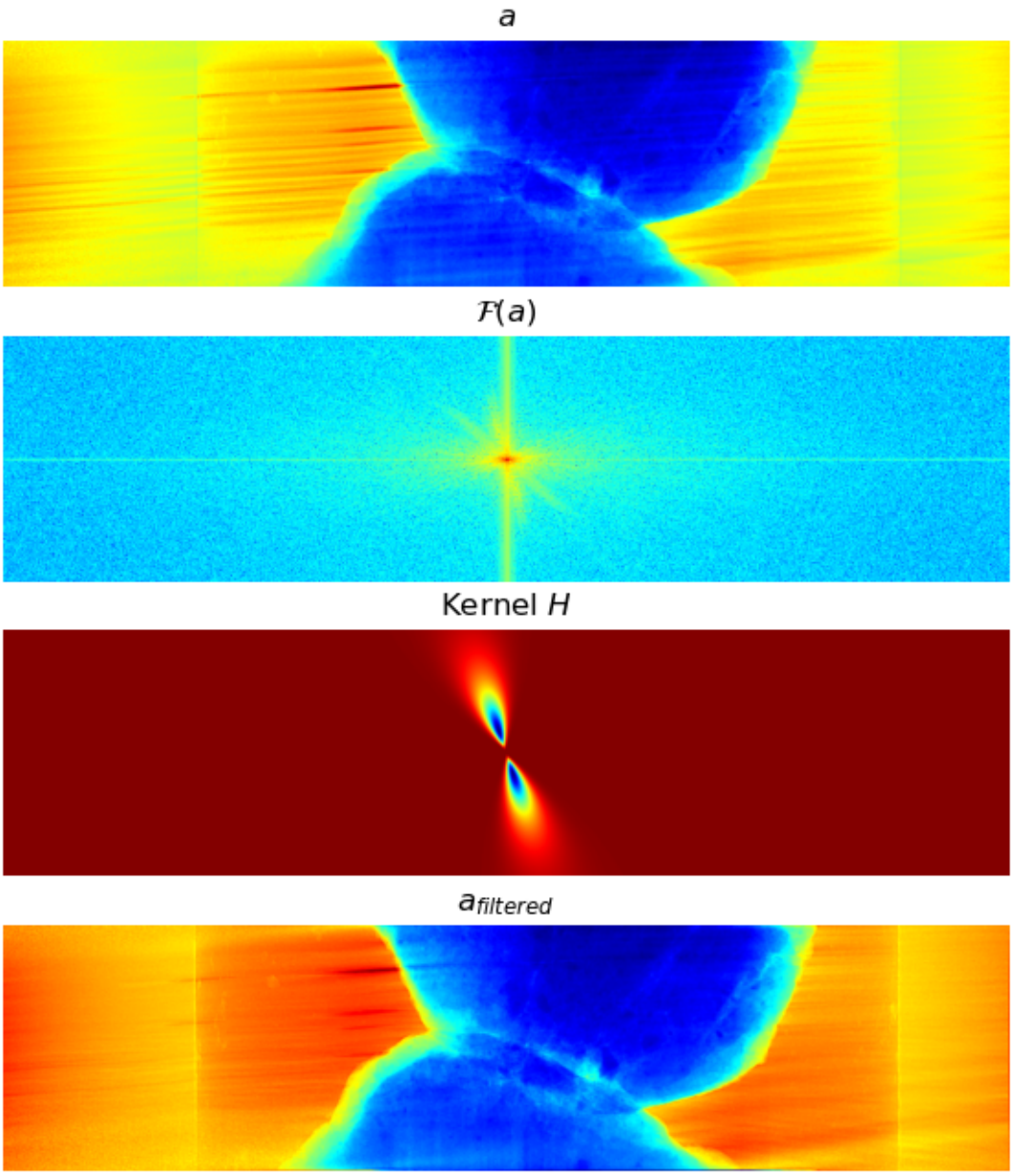
```
def ripple(size=128,angle=0,w0=0.1) :
    w=w0*np.linspace(0,1,size);
    [x,y]=np.meshgrid(w,w);
    img=np.sin((np.sin(angle)*x)+(np.cos(angle)*y));
    return img
N=64;
fig, ax = plt.subplots(2,4, figsize=(15,6)); ax=ax.ravel()

for idx,angle in enumerate([0,30,60,90]) :
    d=ripple(N,angle=(angle+0.1)/180*np.pi,w0=100);
    ax[idx].imshow(d); ax[idx].set_title(r'{0}'.format(angle)+r'$^{\circ}$'); ax[idx].
↪axis('off');
    ax[idx+4].imshow((np.abs(np.fft.fftshift(np.fft.fft2(d)))));   ax[idx+4].axis('off
↪');
```



0.6.6 Example - Stripe removal in Fourier space

- Transform the image to Fourier space $\mathcal{F}_{2\mathcal{D}}\{a\} \Rightarrow A$
- Multiply spectrum image by band pass filter $A_{filtered} = A \cdot H$
- Compute the inverse transform to obtain the filtered image in real space $\mathcal{F}_{2\mathcal{D}}{}^{-1}\{A_{filtered}\} \Rightarrow a_{filtered}$

```
plt.figure(figsize=[10,8])
plt.subplot(4,1,1);plt.imshow(plt.imread('figures/raw_img.png')); plt.title('$a$');␣
↪plt.axis('off');
plt.subplot(4,1,2);plt.imshow(plt.imread('figures/raw_spec.png')); plt.title('$\
↪mathcal{F}(a)$'); plt.axis('off');
plt.subplot(4,1,3);plt.imshow(plt.imread('figures/filt_spec.png')); plt.title('Kernel
↪$H$'); plt.axis('off');
plt.subplot(4,1,4);plt.imshow(plt.imread('figures/filt_img.png')); plt.title('$a_
↪{filtered}$'); plt.axis('off');
```



0.6.7 The effect of the stripe filter

Intensity variations are suppressed using the stripe filter on all projections.

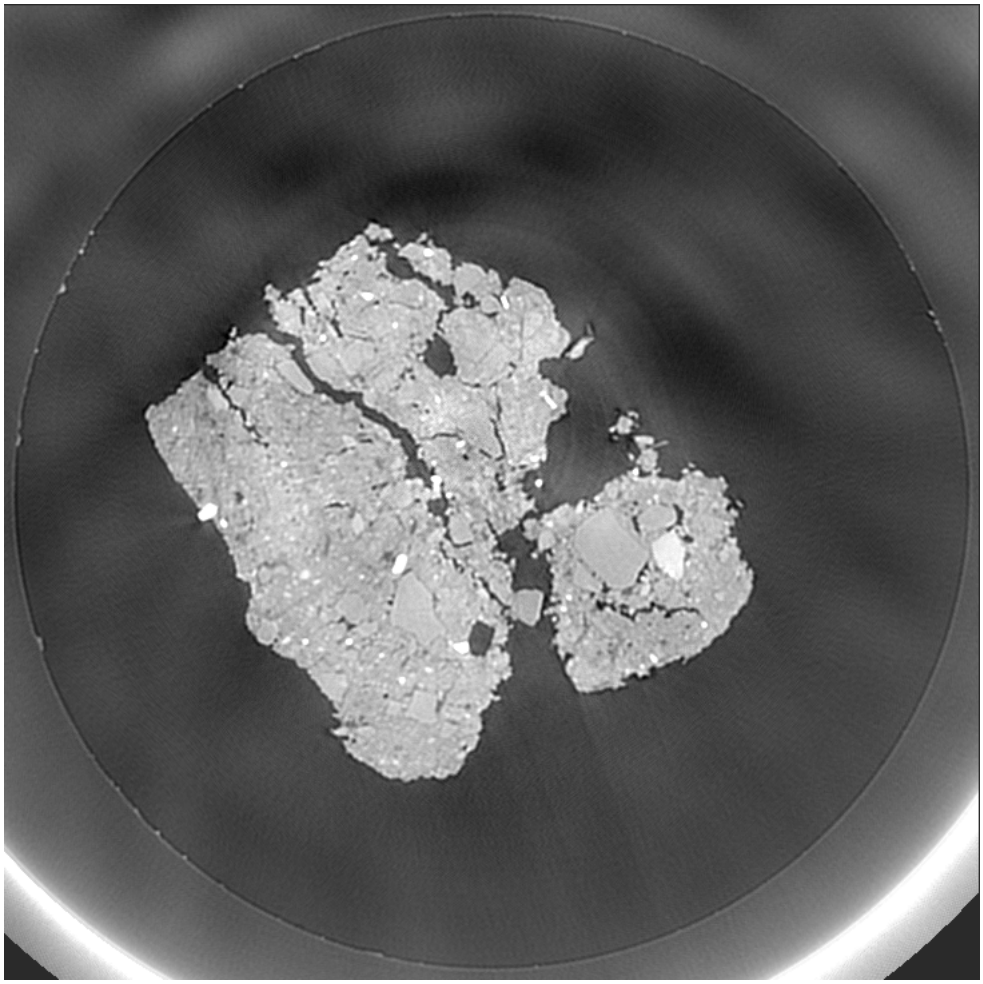


Fig. 1: CT slice before stripe removal filter.

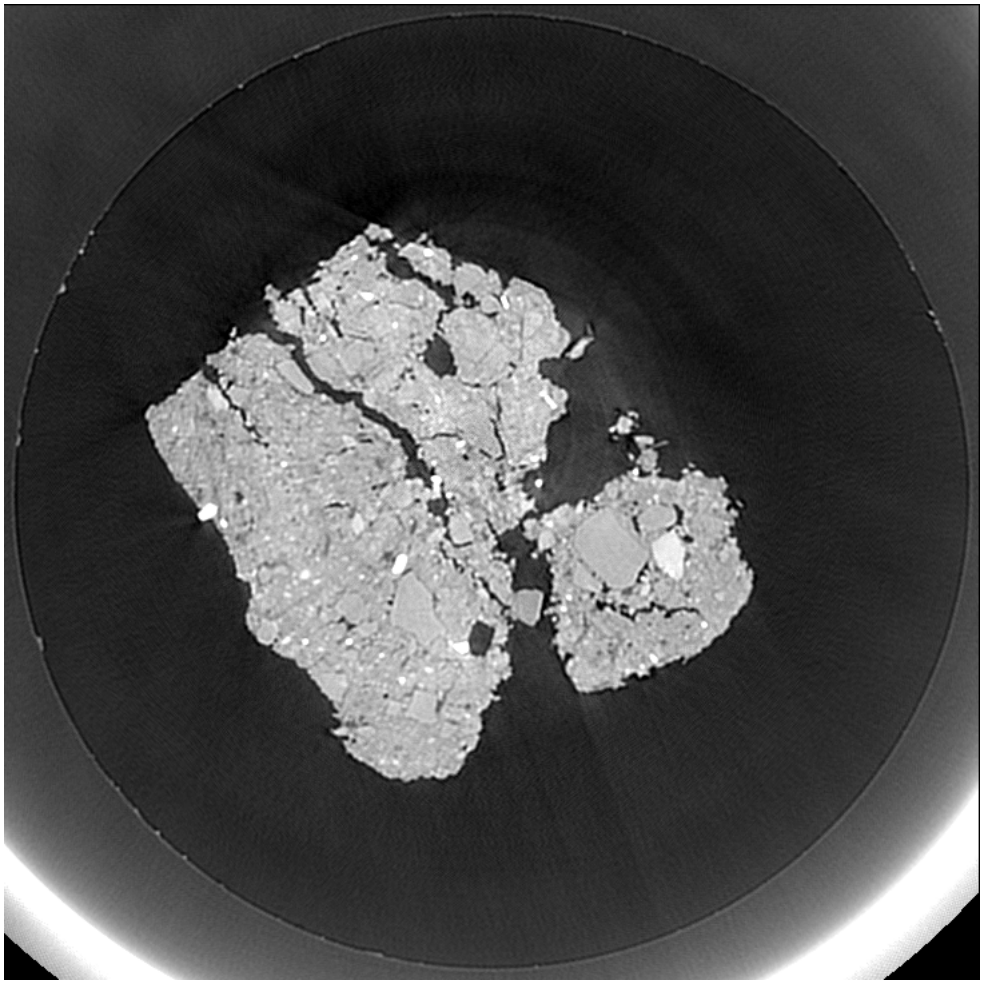


Fig. 2: CT slice after applying stripe removal filter.

0.6.8 Technical details on Fourier space filters

When should you use convolution in Fourier space?

- Simplicity
- Kernel size
- Speed at repeated convolutions

Zero padding

The FFT is only working with data of size in 2^N. If your data has a different length, you have to pad (fill with constant value) up the next 2^N.

0.6.9 Python functions

Filters in the spatial domain

e.g. from `scipy import ndimage`

- `ndimage.filters.convolve(f,h)` Linear filter using kernel h on image f.
- `ndimage.filters.median_filter(f,\[n,m\])` Median filter using an $n \times m$ filter neighborhood

Fourier transform

- `np.fft.fft2(f)` Computes the 2D Fast Fourier Transform of image f
- `np.fft.ifft2(F)` Computes the inverse Fast Fourier Transform F.
- `np.fft.fftshift()` Rearranges the data to center the ω=0. Works for 1D and 2D.

Complex numbers

- `np.abs(f), np.angle(f)` Computes amplitude and argument of a complex number.
- `np.real(f), np.imag(f)` Gives the real and imaginary parts of a complex number.

0.7 Scale spaces

0.7.1 Why scale spaces?

Motivation

Basic filters have problems to handle low SNR and textured noise.

The solution

Filtering on different scales can take noise suppression one step further.

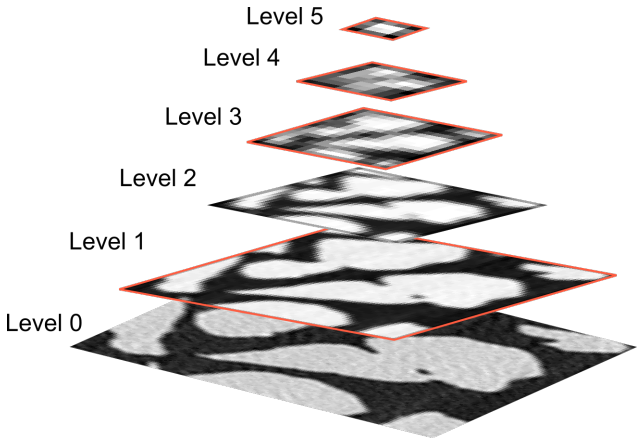


Fig. 1: A scale pyramid of an image can be useful for filtering and segmentation.

0.7.2 Wavelets - the basic idea

- The wavelet transform produces scales by decomposing a signal into two signals at a coarser scale containing \emph{trend} and \emph{details}.\
- The next scale is computed using the trend of the previous transform

$$WT\{s\} \rightarrow \{a_1, d_1\}, WT\{a_1\} \rightarrow \{a_2, d_2\}, \dots, WT\{a_{N-1}\} \rightarrow \{a_N, d_N\}$$

- The inverse transform brings s back using $\{a_N, d_1, \dots, d_N\}$.
- Many wavelet bases exists, the choice depends on the application.

Applications of wavelets

- Noise reduction
- Analysis
- Segmentation
- Compression

Walker 2008 Mallat 2009

0.7.3 Wavelet transform of a 1D signal

Using **symlet-4**

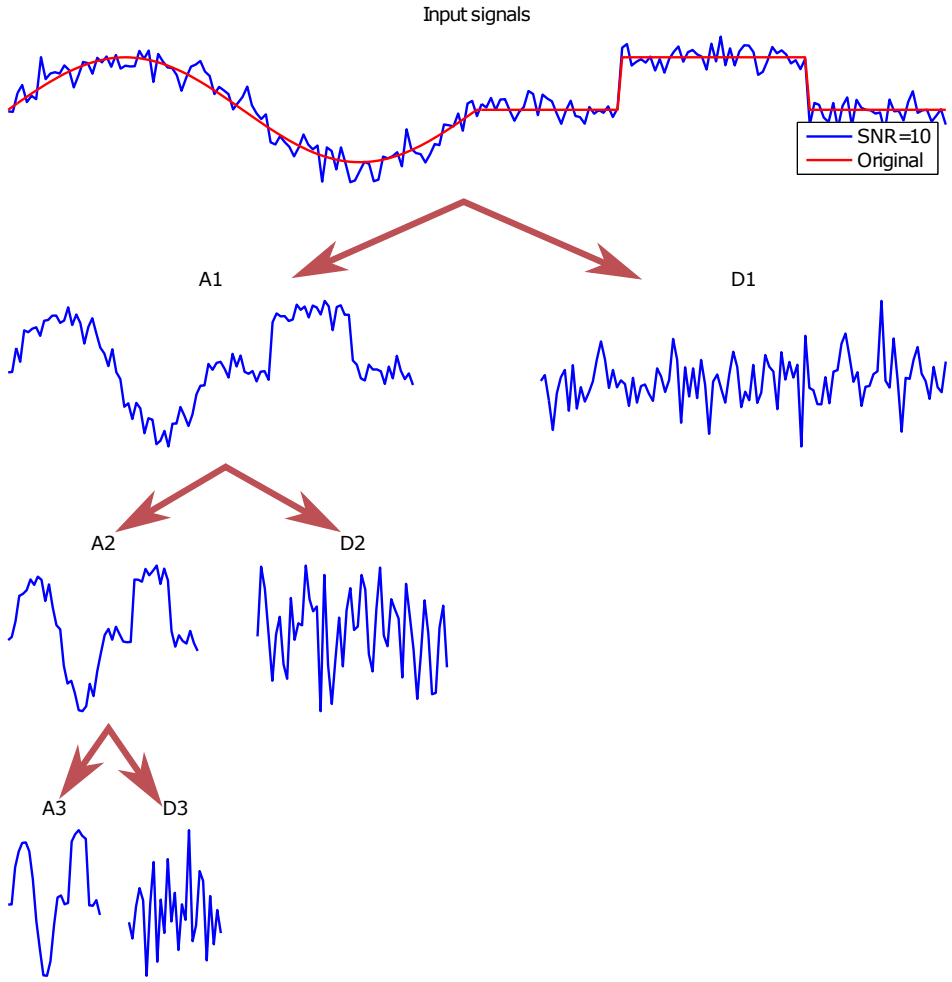


Fig. 2: A noisy test signal decomposed using the *symlet-4* wavelet base function.

0.7.4 Wavelet transform of an image

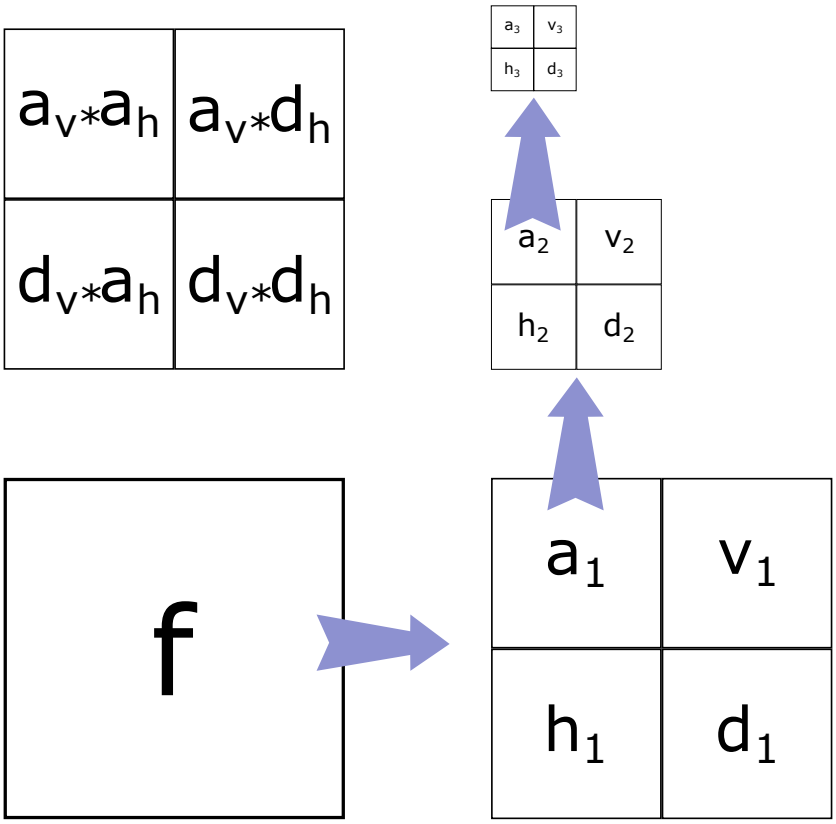


Fig. 3: Transform workflow for a 2D wavelet decomposition.

0.7.5 Wavelet transform of an image - example

0.7.6 Using wavelets for noise reduction

The noise is found in the detail part of the WT

- Make a WT of the signal to a level that corresponds to the scale of the unwanted information.
- Threshold the detail part $d_\gamma = |d| < \gamma \quad ? \quad 0 : d$.
- Inverse WT back to normal scale $\rightarrow$ image is filtered.

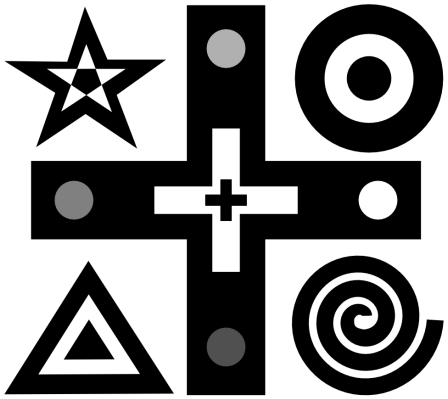


Fig. 4: Transform workflow for a 2D wavelet decomposition.

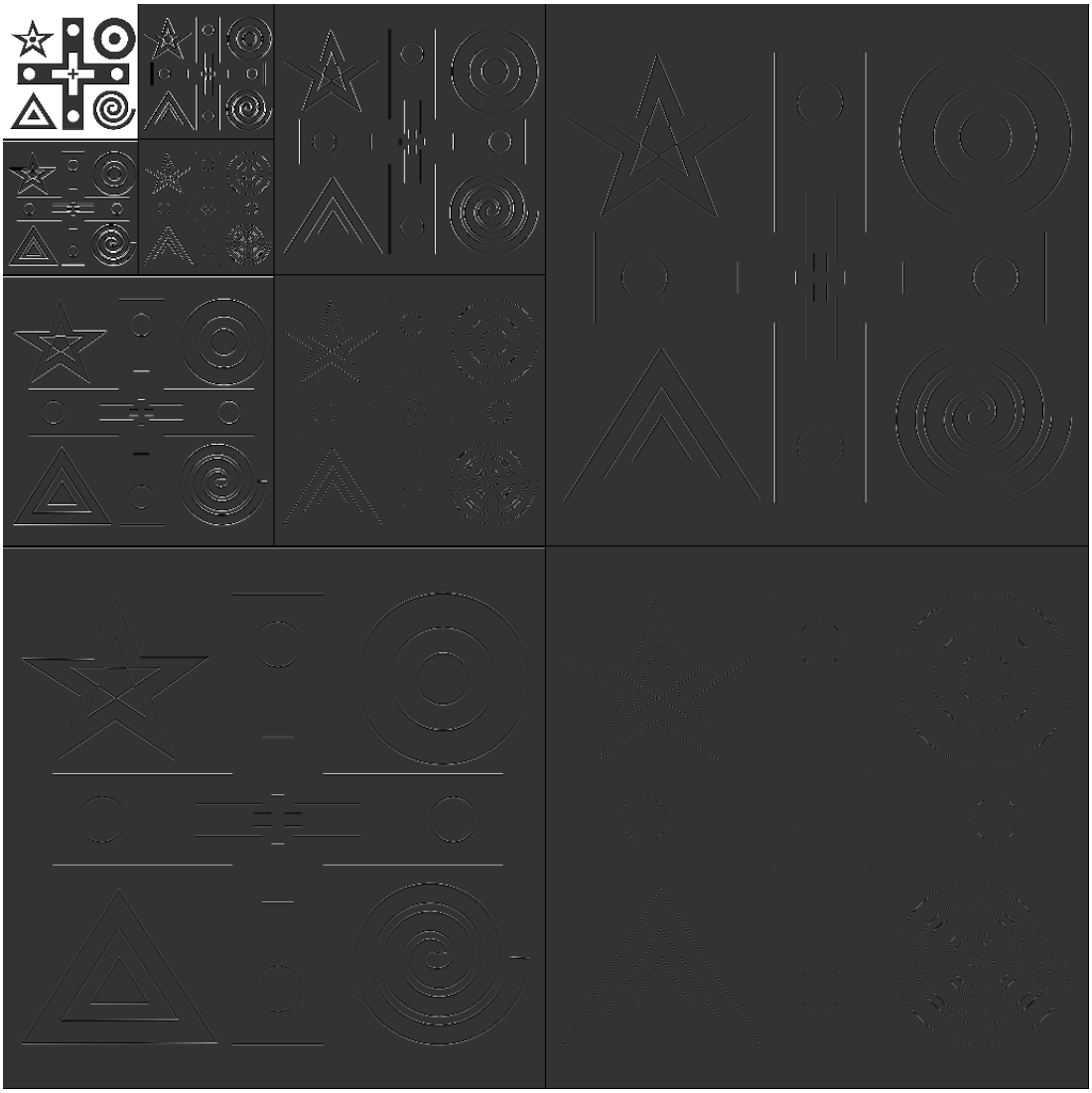


Fig. 5: Transform workflow for a 2D wavelet decomposition.

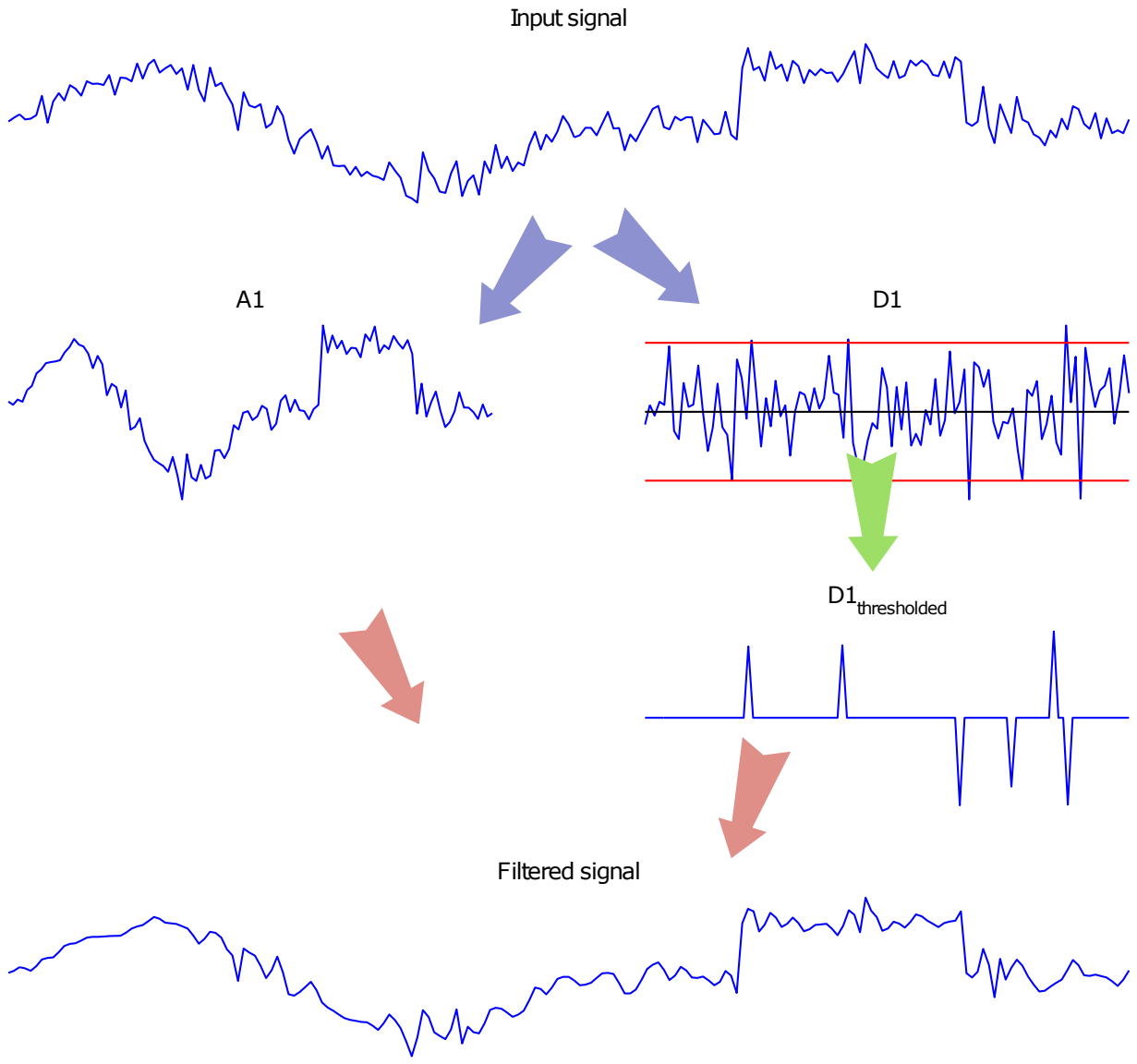


Fig. 6: The principle of a basic noise reduction filter using wavelets.

0.7.7 Wavelet noise reduction - Image example

Example Filtered using two levels of the Symlet-2 wavelet

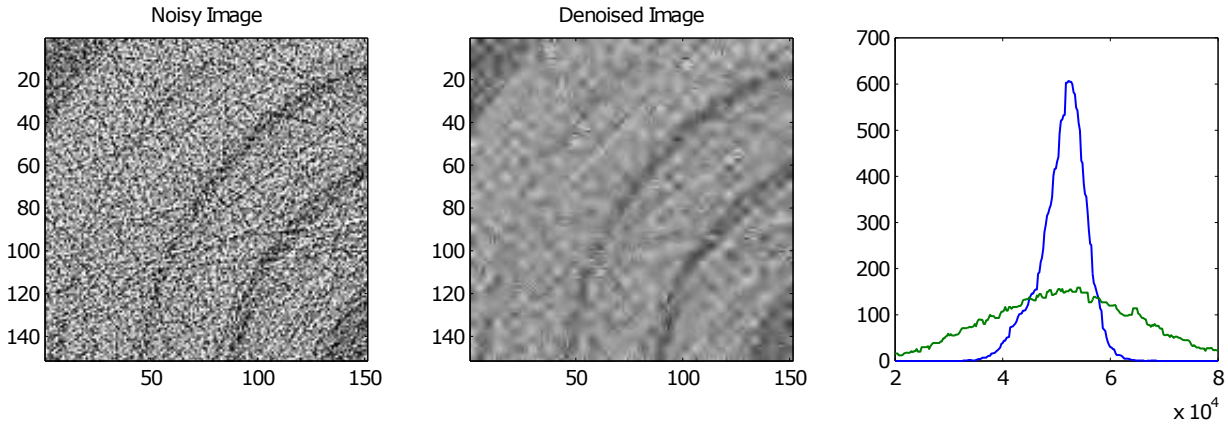


Fig. 7: An example of noise reduction using a wavelet filter.

Data: Neutron CT of a lead scroll

0.7.8 Python functions for wavelets

- **dwt2/idtw2** Makes one level of the wavelet transform or its inverse using wavelet base specified by 'wn'.
- **wavedec2** Performs N levels of wavelet decomposition using a specified wavelet base.
- **wbmpen** Estimating threshold parameters for wavelet denoising.
- **wdencmp** Wavelet denoising and compression using information from *wavedec2* and *wbmpen.

0.8 Parameterized scale spaces

0.8.1 PDE based scale space filters

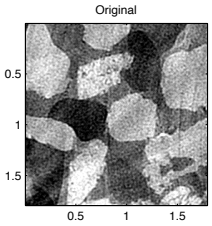


Fig. 1: Noisy slice to be filtered.

These filters may work for applications where Linear and Rank filters fail.

Kaestner et al. 2008 Aubert 2002.

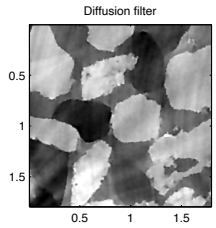


Fig. 2: Slice after diffusion filter.

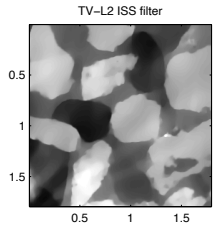


Fig. 3: Slice after ISS filter.

0.8.2 The starting point

The heat transport equation $\$\frac{\partial T}{\partial t} = \kappa\nabla^2 T\$$

- T Image to filter (intensity $\equiv$ temperature)
- κ Thermal conduction capacity

0.8.3 Controlling the diffusivity

We want to control the diffusion process…

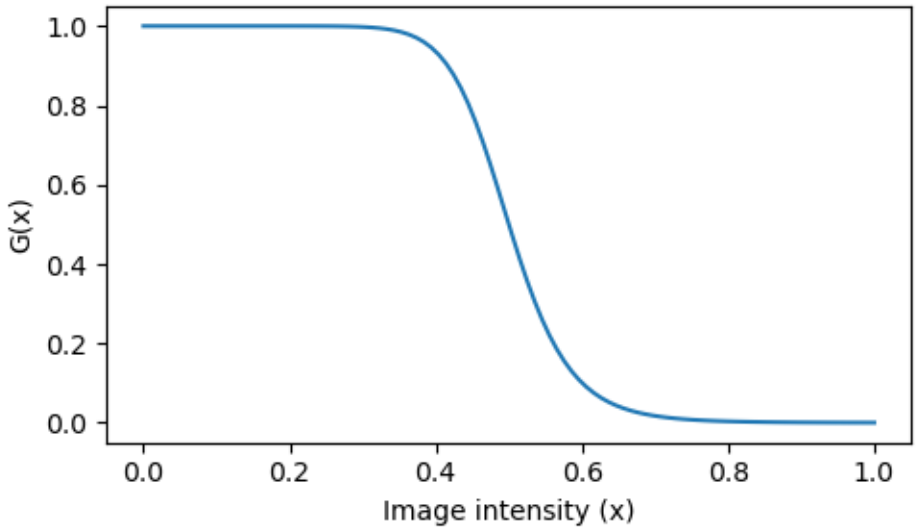
- *Near edges* The Diffusivity $\rightarrow 0$
- *Flat regions* The Diffusivity $\rightarrow 1$

The contrast function G is our control function $\$G(x) = \frac{1}{1+(\frac{x}{\lambda})^n}\$$

- λ Threshold level
- n Steepness of the threshold function

```
def g(x,lambd,n) :
    g=1/(1+(x/lambd)**n)
    return g
```

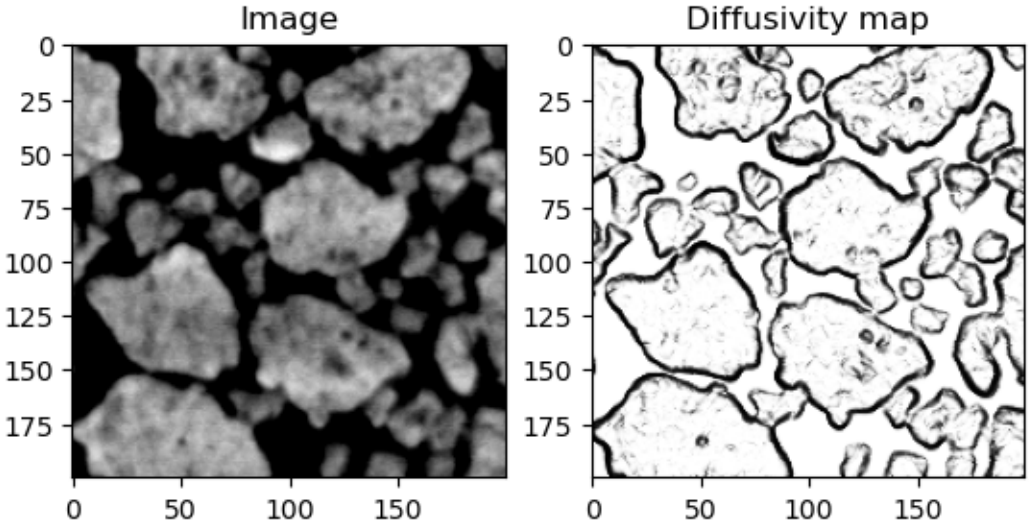
```
x=np.linspace(0,1,100);

plt.figure(figsize=(5,3))
plt.plot(x,g(x,lambd=0.5,n=12));
plt.xlabel('Image intensity (x)'); plt.ylabel('G(x)');plt.tight_layout()
```



0.8.4 Gradient controlled diffusivity

$$\frac{\partial u}{\partial t} = G(|\nabla u|)\nabla^2 u$$

```
plt.subplot(1,2,1); plt.imshow(io.imread("figures/aggregates.png"),cmap='gray'); plt.
↪title('Image');
plt.subplot(1,2,2); plt.imshow(io.imread("figures/diffusivity.png"),cmap='gray'); plt.
↪title('Diffusivity map');
```



- u Image to be filtered
- $G(\cdot)$ Non-linear function to control the diffusivity
- τ Time increment
- N Number of iterations

0.8.5 The non-linear diffusion filter

A more robust filter is obtained with

$$\frac{\partial u}{\partial t} = G(|\nabla_\sigma u|)\nabla^2 u$$

- u Image to be filtered
- $G(\cdot)$ Non-linear function to control the contrast
- τ Time increment per numerical iteration
- N Number of iterations
- ∇_σ Gradient smoothed by a Gaussian filter, width σ

0.8.6 Diffusion filter example

Neutron CT slice from a real-time experiment observing the coalescence of cold mixed bitumen.

0.8.7 Filtering as a regularization problem

The continued development

- **90's** During the late 90's the diffusion filter was described in terms of a regularization problem.
- **00's** Work toward regularization of total variation minimization.

TV-L1

$$u = \underset{u \in BV(\Omega)}{\operatorname{argmin}} \left\{ \underbrace{|u|_{BV}}_{noise} + \underbrace{\tfrac{\lambda}{2}\|f-u\|_1}_{fidelity} \right\}$$

TV-L2 Rudin-Osher-Fatemi model (ROF)

$$u = \underset{u \in BV(\Omega)}{\operatorname{argmin}} \left\{ \underbrace{|u|_{BV}}_{noise} + \underbrace{\tfrac{\lambda}{2}\|f-u\|_2^2}_{fidelity} \right\}$$

with $|u|_{BV} = \int_\Omega |\nabla u|^2$

0.8.8 The inverse scale space filter

The idea

We want smooth regions with sharp edges\ldots

- Turn the processing order of scale space filter upside down
- Start with an empty image
- Add large structures successively until an image with relevant features appears

The ISS filter - Some properties

- is an edge preserving filter for noise reduction.
- is defined by a partial differential equation.
- has a well defined termination point.

Burger et al. 2006

The ROF filter equation

The image f is filtered by solving

$\frac{\partial u}{\partial t} = \text{div}\left(\frac{\nabla u}{|\nabla u|}\right) + \lambda(f - u + v)$
$\frac{\partial v}{\partial t} = \alpha(f - u)$

Variables

- f Input image
- u Filtered image
- v Regularization term (feedback of previous iteration)

Filter parameters

- λ Related to the scale of the features to suppress.
- α Quality refinement
- N Number of iterations
- τ Time increment

Filter iterations

Neutron CT of dried lung filtered using 3D ISS filter

How to choose lambda and alpha

The requirements varies between different data sets.

Initial conditions:

- Signal to noise ratio
- Image features (fine grained or wide spread)

Experiment:

- Scan λ and α
- Stop at $T = N\tau = \sigma$ use different τ
- When does different effects occur, related to σ?

Solutions at different times

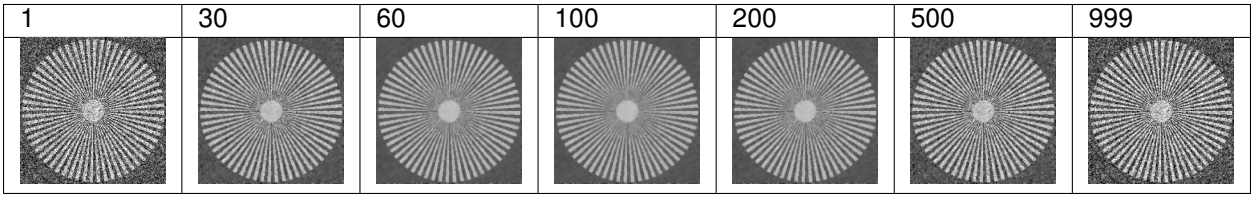


Fig. 4: Error plot for different solution times of the ISS filter.

Solution time

The solution time ($T = N\tau$) is essential to the result

- τ *large* The solution is reached fast
- τ *small* The numerical accuracy is better

The choice of initial image

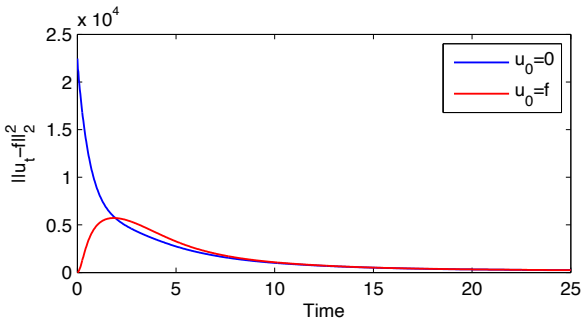


Fig. 5: Error plots depend on the choice of the initial image.

At some T the solution with $u_0 = f$ and $u_0 = 0$ converge.

0.9 Non-local means

0.9.1 Non-local smoothing

The idea

Smoothing normally consider information from the neighborhood like

- Local averages (convolution)
- Gradients and Curvatures (PDE filters)

Non-local smoothing average similiar intensities in a global sense.

- Every filtered pixel is a weighted average of all pixels.
- Weights computed using difference between pixel intensities.

Buades et al. 2005

0.9.2 Filter definition

The non-local means filter is defined as $u(p) = \frac{1}{C(p)} \sum_{q \in \Omega} v(q) f(p,q)$ where

- v *and* u input and result images.
- $C(p)$ is the sum of all pixel weights as $C(p) = \sum_{q \in \Omega} f(p,q)$
- $f(p,q)$ is the weighting function $f(p,q) = e^{-\frac{|B(q)-B(p)|^2}{h^2}}$
- *B(x)* is a neighborhood operator e.g. local average around x

0.9.3 Non-local means 2D - Example

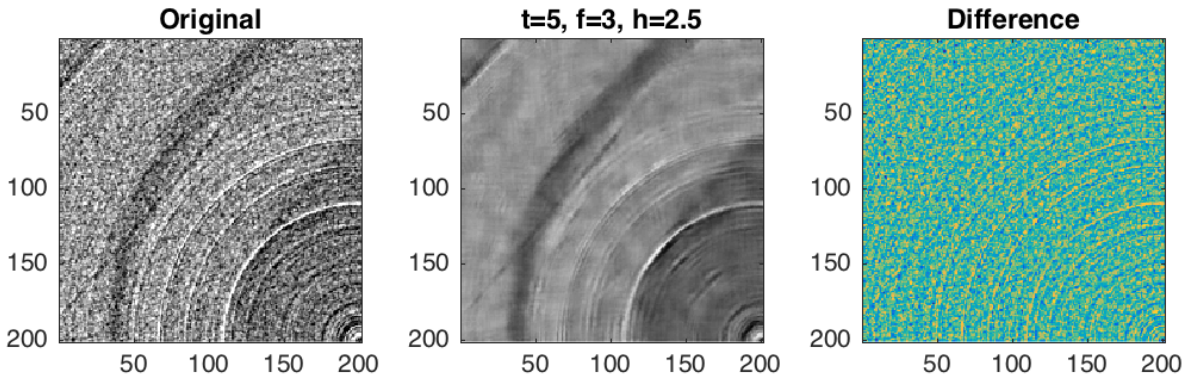


Fig. 1: Demonstration on the non-local means filter.

Observations

- Good smoothing effect.
- Strong thin lines are preserved.
- Some patchiness related to filter parameter t, *i.e.* the size of Ω_i.

0.9.4 Performance complications

Problem

The orignal filter compares all pixels with all pixels…

- Complexity $\mathcal{O}(N^2)$
- Not feasible for large images, and particular 3D images!

Solution

It has been shown that not all pixels have to be compared to achieve a good filter effect. i.e. Ω in the filter equations can be replaced by $\Omega_i << \Omega$

0.9.5 Explore the Non-local means in more detail

A notebook is prepared to study the non-local means.

0.10 The BM3D filter

- The BM3D has evolved from the ideas in the non-local means filter.
- Applies filters on the matching blocks instead of the average.
- It is currently one of the most efficient denoising filters.

References

- K. Dabov et al. 2007 Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering, IEEE Trans on Image Processing, 2007
- M. LeBrun An Analysis and Implementation of the BM3D Image Denoising Method, Image Processing Online, 2012
- M. Elad et al. Image Denoising: The Deep Learning Revolution and Beyond — A Survey Paper, SIAM J. on Imaging Sciences, 2023

0.10.1 Filter algorithm outline

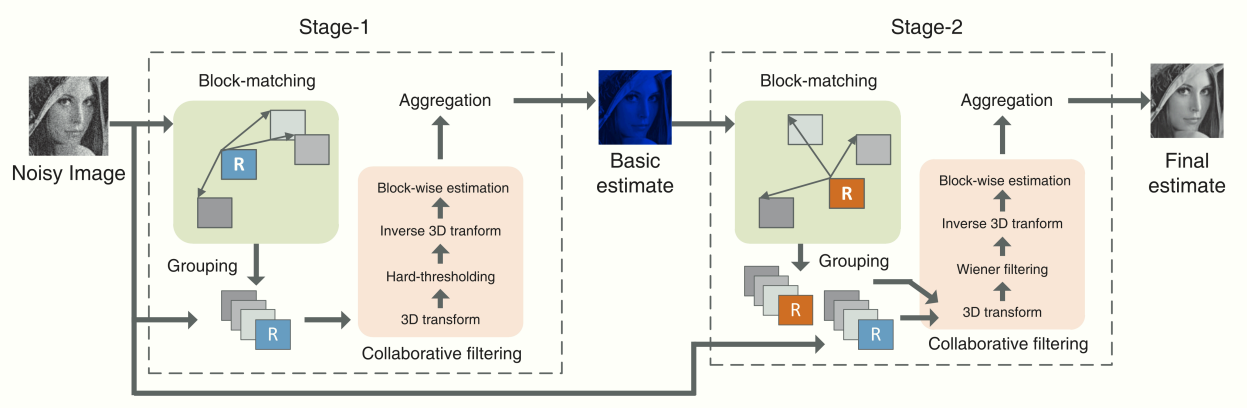
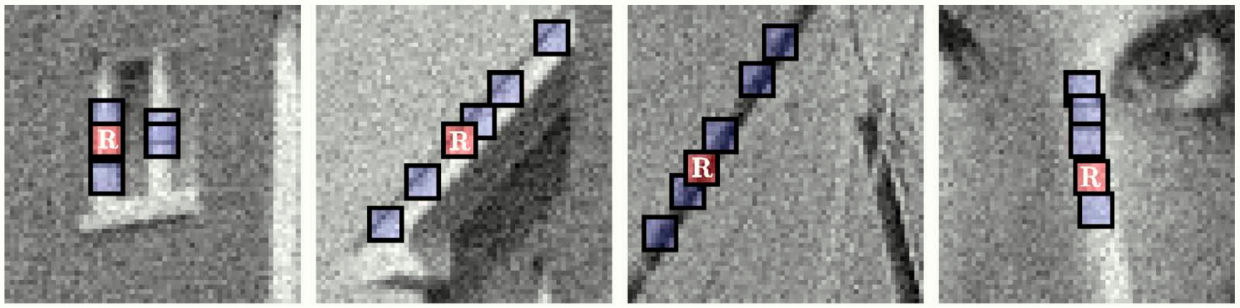


Figure from D. Wang et al., 2020

What does block matching mean?



- Similar blocks found by a least squared difference between thresholded spectra of the images.
- The threshold is given by the image noise variance.

Figure from K. Dabov et al. 2007

0.10.2 BM3D in real images

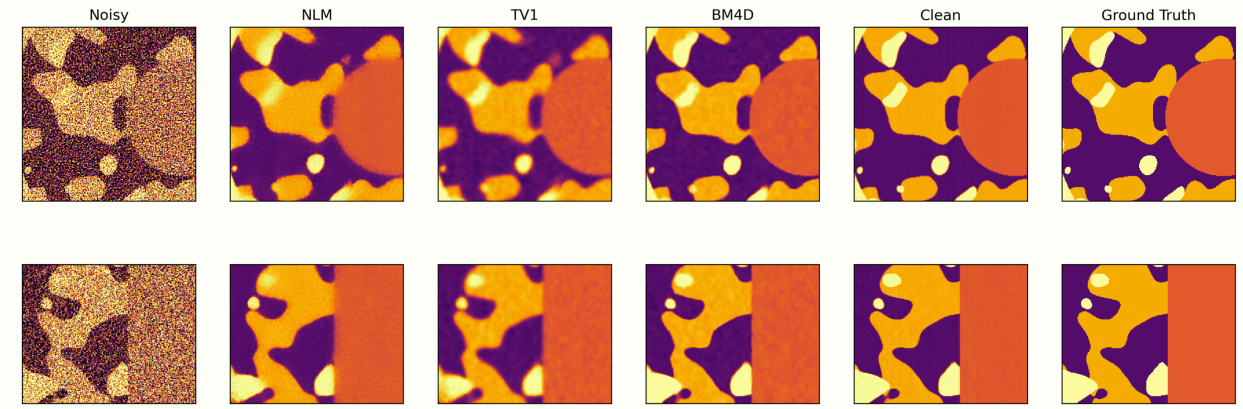


Figure from Q. Zhan et al. 2024

What about 3D images?

The BM3D filter was originally defined for 2D images.

The upscaling to 3D images is quite straight forward

- All processing steps add one dimension
- The filtering colaborative filtering step is now done on a 4D image

The filter is now called **BM4D** instead.

Issues with BM4D

The additional dimension add some complexity

- Implementation: It may not be so easy to wrap your head around the 4th dimension
- Computational
  - Memory: 3D images requires much more memory.
  - Processing time: The increased amount of data takes longer time to process. Implementations with GPUs exist.

0.10.3 Can denoising go even further?

- BM3D was long the king among denoisers
- On global scale, it is hard to get better.
- Looking into the details there is still potential to improve

Deep learning for denoising

0.11 How to choose denoiser

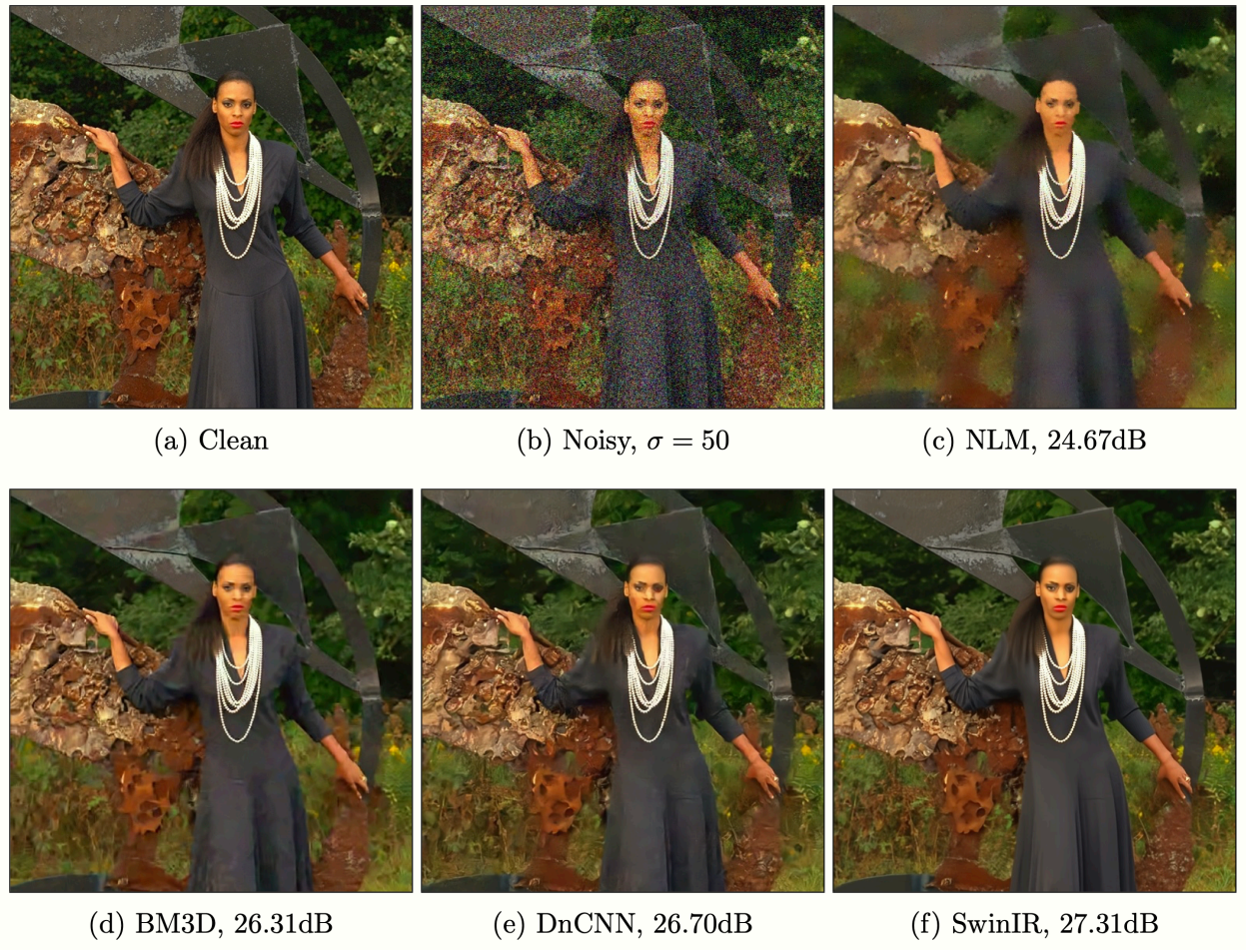
We have seen a collection of filters to improve the SNR.

> You may say take the best! But what is best for your images?

There are some criteria that determines the filter type you choose.

- Do you have 2D or 3D images?
- General image composition
- Which level of detail needs to be preserved?
- How noisy are your images?
- Which filters do you already have access to?
- Do you have the hardware to use the filter?
- How much time do you have
  - To implement - Many advanced filters are not easily available.
  - To use - You can spend days on a single image, but what if you have 1000 or even more?
- etc.

Often you have to test and compare the performance of different filters to decide.



(a) Clean (b) Noisy, $\sigma = 50$ (c) NLM, 24.67dB

(d) BM3D, 26.31dB (e) DnCNN, 26.70dB (f) SwinIR, 27.31dB

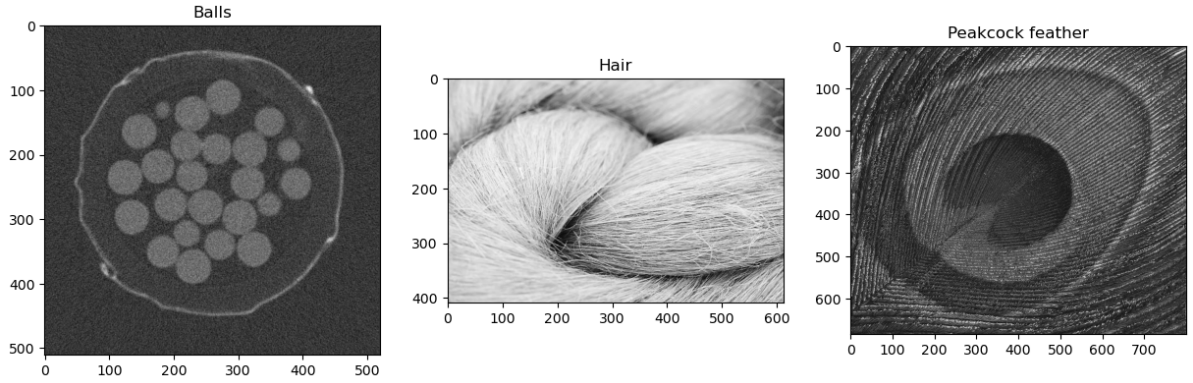
Fig. 1: Figure from Elad et al., https://doi.org/10.48550/arXiv.2301.03362, 2023.

0.12 Enhancing and analyzing orientation

Many images have structures with dominant orientations locally and globally.

```
balls = io.imread('figures/recon5s_0130.tif').astype(float)
hair  = io.imread('figures/hair.jpeg').astype(float).mean(axis=2)
pea   = io.imread('figures/peacock.jpeg').astype(float).mean(axis=2)
```

```
fig,ax = plt.subplots(1,3,figsize=(15,5))
ax[0].imshow(balls,cmap='gray')
ax[0].set_title('Balls')
ax[1].imshow(hair,cmap='gray')
ax[1].set_title('Hair')
ax[2].imshow(pea,cmap='gray')
ax[2].set_title('Peakcock feather');
```



The question is how we can measure them…

0.12.1 Gradients

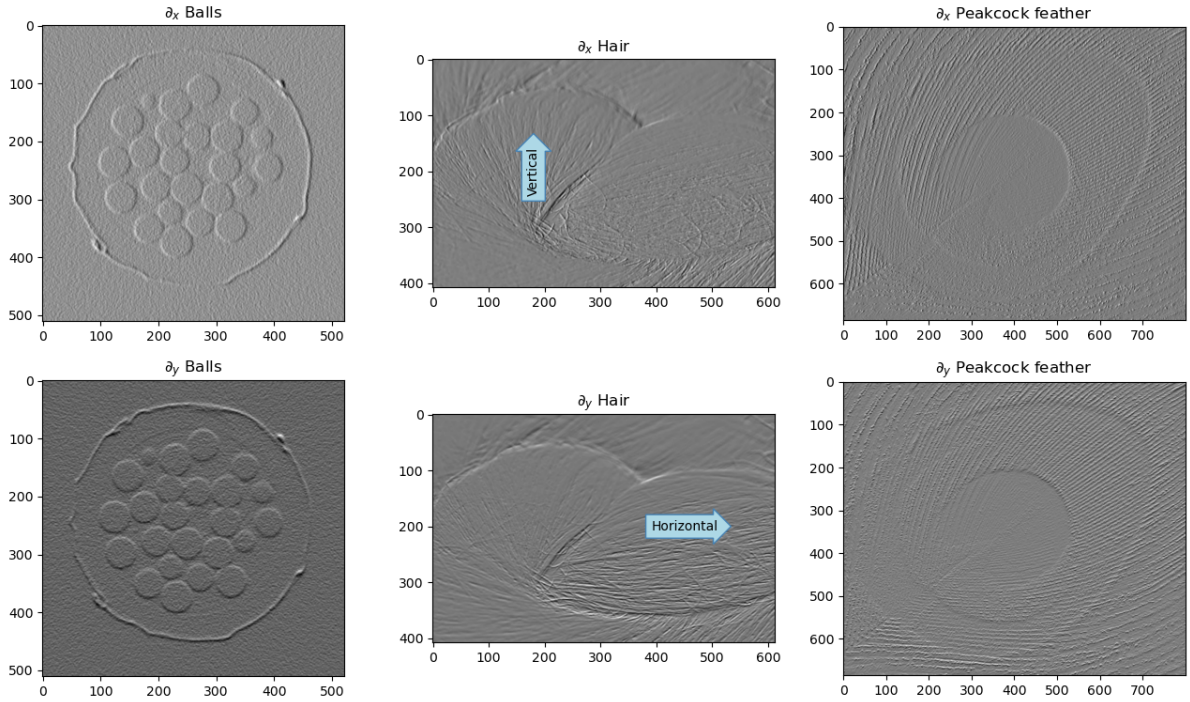
We just learned that the gradient can isolate edge information in the image. It is also computed along the principal axes of the image. Below, we see the gradient of the three test images.

```
fig,ax = plt.subplots(2,3,figsize=(16,9))
ax=ax.ravel()
sigma=1
ax[0].imshow(gaussian_filter(balls,sigma,order=[0,1]),cmap='gray')
ax[0].set_title(r'$\partial_x$ Balls')
ax[1].imshow(gaussian_filter(hair,sigma,order=[0,1]),cmap='gray')
ax[1].set_title(r'$\partial_x$ Hair')
t = ax[1].text(180,200,  "Vertical",
            ha="center", va="center", rotation=90, size=10,
            bbox=dict(boxstyle="rarrow,pad=0.4",
                      fc="lightblue", ec="steelblue", lw=1))
ax[2].imshow(gaussian_filter(pea,sigma,order=[0,1]),cmap='gray')
ax[2].set_title(r'$\partial_x$ Peakcock feather');

ax[3].imshow(gaussian_filter(balls,sigma,order=[1,0]),cmap='gray')
ax[3].set_title(r'$\partial_y$ Balls')
ax[4].imshow(gaussian_filter(hair,sigma,order=[1,0]),cmap='gray')
```

(continues on next page)

(continued from previous page)

```
ax[4].set_title(r'$\partial_y$ Hair')
t = ax[4].text(450,200,  "Horizontal",
            ha="center", va="center", rotation=0, size=10,
            bbox=dict(boxstyle="rarrow,pad=0.4",
                      fc="lightblue", ec="steelblue", lw=1))
ax[5].imshow(gaussian_filter(pea,sigma,order=[1,0]),cmap='gray')
ax[5].set_title(r'$\partial_y$ Peakcock feather');
```



In these images, we clearly see that structures in different directions are enhanced. We can for example see that the vertially oriented hairs are enhanced by ∂_x and the horizontally oriented hairs by ∂_y.

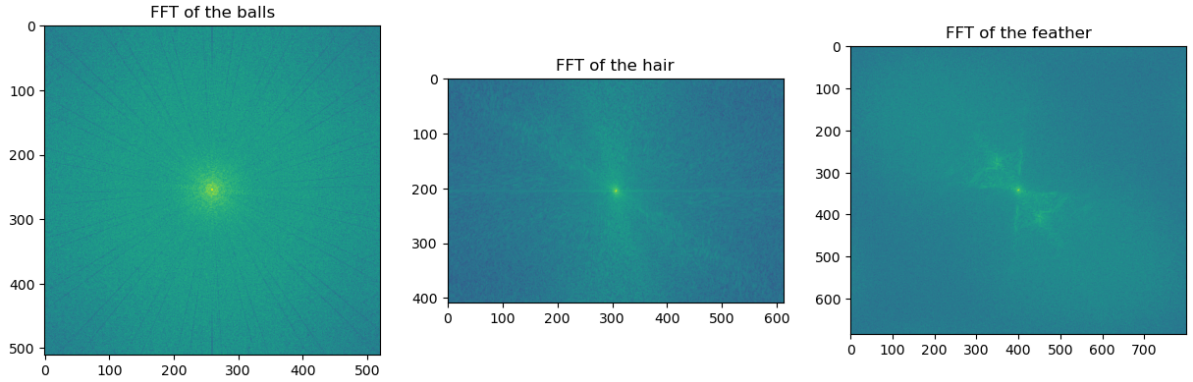
… but we still only have image intensities. Now showing the local slope.

0.12.2 How about the Fourier transform?

We saw before that the Fourier transform can identify orientations.

```
fig,ax=plt.subplots(1,3,figsize=(15,5))
fballs = np.fft.fftshift(np.fft.fft2(balls));
ax[0].imshow(np.abs(fballs),norm=LogNorm())
ax[0].set_title('FFT of the balls')

fhair = np.fft.fftshift(np.fft.fft2(hair));
ax[1].imshow(np.abs(fhair),norm=LogNorm())
ax[1].set_title('FFT of the hair')

fpea = np.fft.fftshift(np.fft.fft2(pea));
ax[2].imshow(np.abs(fpea),norm=LogNorm());
ax[2].set_title('FFT of the feather')
```

```
Text(0.5, 1.0, 'FFT of the feather')
```



… but mainly in the global sense. Complicated images are hard to analyze.

0.12.3 The structure tensor

Let's try combinations of the gradients using the outer product of the gradients

$$J(f) = \nabla f \cdot \nabla^T f = \begin{bmatrix} (\partial_x f)^2 & \partial_x f \cdot \partial_y f \\ \partial_y f \cdot \partial_x f & (\partial_y f)^2 \end{bmatrix}$$

Gradients are quite noisy!
Therefore, it is common to add smoothing in the computation, e.g. using a Gaussian filter.

Using a structure tensor, also known as the second moment matrix or the inertia tensor, instead of directly using gradients to compute an orientation field, offers several advantages, particularly in the context of image analysis and computer vision. This approach is especially relevant for applications like texture analysis, edge detection, motion detection, and the analysis of local orientations in images.

Robustness to Noise

Gradients directly computed from an image are highly sensitive to noise since differentiation amplifies high-frequency components, which include noise. The structure tensor, on the other hand, incorporates a form of averaging (usually through Gaussian smoothing) which makes the orientation field computation more robust to noise.

Capturing Coherent Structures

The structure tensor effectively captures the dominant directions of gradient flows within a local neighborhood by aggregating the information from gradients. This aggregation helps in identifying coherent structures in regions where gradients are not consistent due to texture or noise, providing a more reliable estimate of local orientations.

Handling Ambiguities in Orientation

Direct gradients provide a local direction of change but can be ambiguous (e.g., the gradient direction is perpendicular to an edge, not along it). The structure tensor, by considering the outer product of gradients, offers a way to resolve these ambiguities by identifying principal directions of variance in gradient orientations, which are more informative for understanding the underlying structure of the image.

Anisotropy and Coherence Measurement

Beyond just orientation, the structure tensor can be used to measure the coherence or anisotropy of local patterns. By analyzing the eigenvalues of the structure tensor, one can distinguish between isotropic regions (where intensity changes uniformly in all directions), coherent structures (with a clear orientation), and corners (where two dominant orientations intersect).

Scale Adaptability

The structure tensor formulation typically involves a Gaussian smoothing step, where the scale of the Gaussian kernel can be adjusted. This allows for the analysis of structures at different scales, making the method adaptable to various applications and capable of capturing features of different sizes within an image.

Mathematical and Computational Framework

The structure tensor integrates well into the mathematical and computational frameworks used in image analysis and computer vision. It allows for a unified approach to feature extraction, and its computation can be efficiently implemented and parallelized, which is particularly beneficial for processing large images or real-time applications.

In summary, the structure tensor approach provides a more robust, informative, and versatile method for computing orientation fields in images, especially in the presence of noise, varying textures, and complex structural patterns. It enhances the reliability and accuracy of orientation estimation, which is crucial for advanced image analysis tasks.

Implementing the structure tensor

The equation to implement the structure tensor includes a Gaussian filter, G_σ, with the filter with set by σ.

$$J_\sigma(f) = G_\sigma * \nabla f \cdot \nabla^T f = \begin{bmatrix} G_\sigma * (\partial_x f)^2 & G_\sigma * (\partial_x f \cdot \partial_y f) \\ G_\sigma * (\partial_y f \cdot \partial_x f) & G_\sigma * (\partial_y f)^2 \end{bmatrix}$$

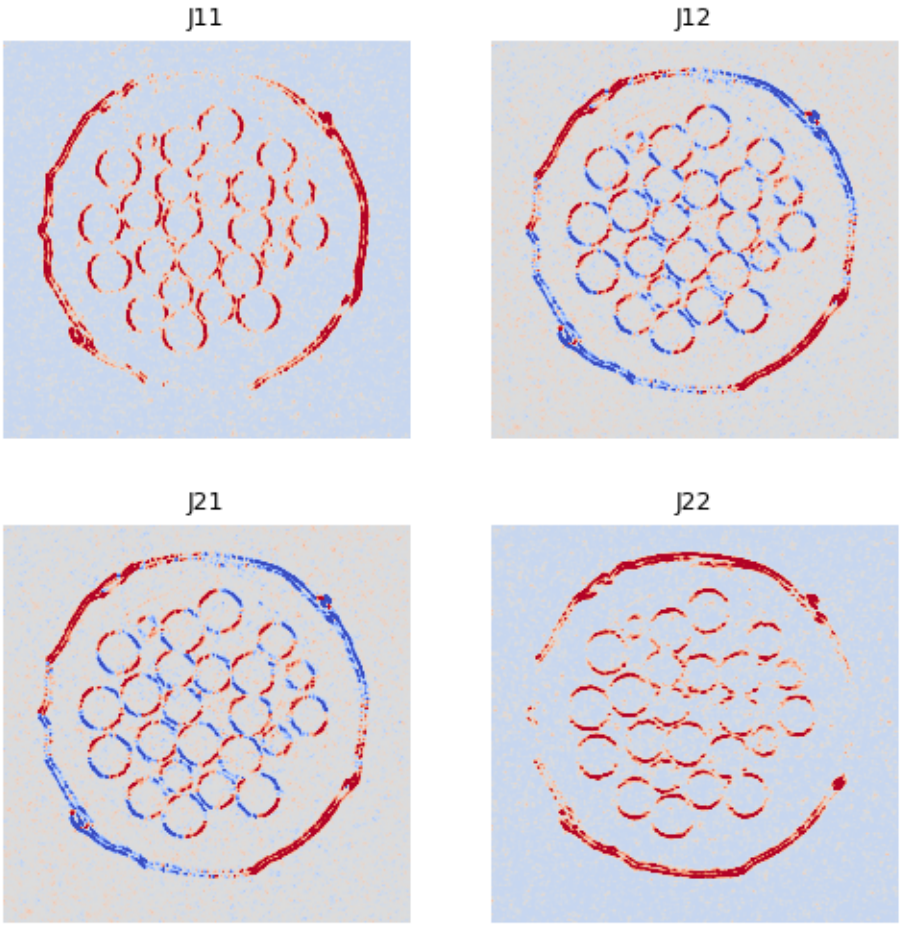
```
def structure_tensor(img,sigma) :
    Ix = gaussian_filter(img,sigma,order=[0,1]) # Technical note: The Gaussian filter␣
 ↪function can compute
    Iy = gaussian_filter(img,sigma,order=[1,0]) # derivatives.
    J = {'J11' : Ix**2,
         'J12' : Ix*Iy,
         'J21' : Ix*Iy,
         'J22' : Iy**2} # A dict of the tensor elements

    return J
```

This function computes the structure tensor of an image and stores it in a dict having the labels J11, J12, J21, and J22. The gradients that normally would be computed using a gradient kernel is here computed together with the Gaussian smoothing of the image. This is a special feature of the Gaussian filter function provided by scipy.ndimage.

Looking at the components of the structure tensor

```
J = structure_tensor(balls,1.5)
```

```
# Visualization
fig,axes =plt.subplots(2,2,figsize=(6,6))
axes=axes.ravel()
for ax,j in zip(axes,J) :
    m = J[j].mean()
    s = J[j].std()
    clims=[m-2*s,m+2*s]
    cmap = 'coolwarm'

    ax.imshow(J[j],clim=clims,cmap=cmap)
    ax.axis('off')
    ax.set_title(j,fontsize=9)
```



0.12.4 Local eigen-value analysis of the structure tensor

$$\mathrm{J}\mathrm{v} = \lambda\mathrm{v}$$

where λ is a diagonal matrix $\$\lambda = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}\$$

The coherency - C

$$0 \leq C = \frac{\lambda_{max} - \lambda_{min}}{\lambda_{max} + \lambda_{min}} = \frac{\sqrt{(J_{22} - J_{11})^2 + 4J_{12}^2}}{J_{22} + J_{11}} \leq 1$$

Measures the level of confidence of the pixel/region

$$\begin{cases} C = 0 & \lambda_{min} \approx \lambda_{max}, \text{ the region is homogeneous, i.e. no dominant rotation.} \\ C = 1 & \lambda_{min} \ll \lambda_{max}, \text{ the rotation is well aligned with one of the axes.} \\ 0 < C < 1 & \text{the local orientation lies betwenn the axes.} \end{cases}$$

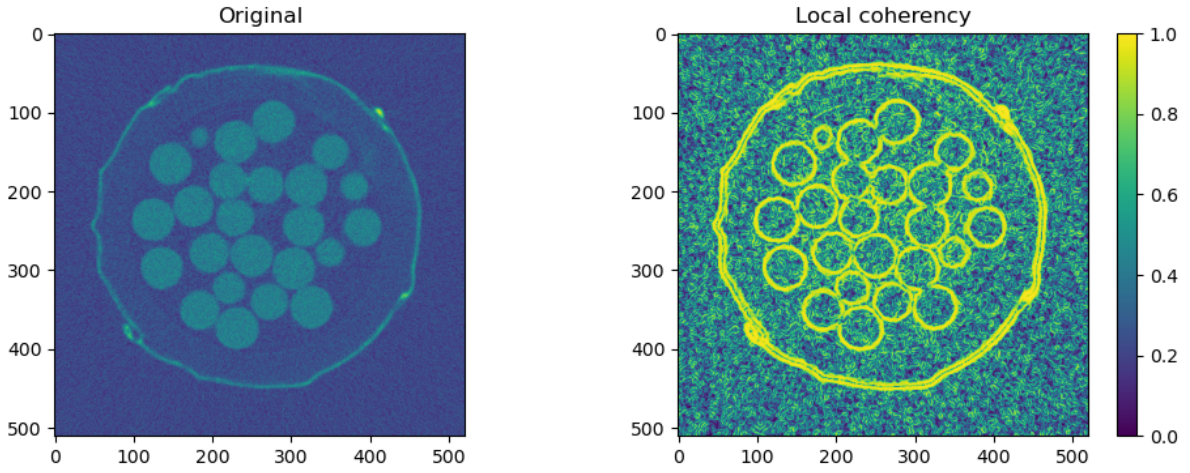
```
def coherency(J,eps=0.1) :
    return np.sqrt(((J['J22']-J['J11'])**2+4*J['J12']**2))/(J['J22']+J['J11']+eps)
```

The epsilon in the function is a protection against division by zero in the case when J_{11} and J_{22} are close to zero. It should be set to a small value relative to the varaiations in the image.

Looking at the coherency

```
C=coherency(J,eps=(J['J22']+J['J11']).std()/10)
```

```
# Visualization
fig,ax = plt.subplots(1,2,figsize=(12,4))
ax[0].imshow(balls)
ax[0].set_title('Original')
a=ax[1].imshow(C,clim=[0,1])
plt.colorbar(a,ax=ax[1])
ax[1].set_title('Local coherency');
```



Here, eps was set to 10% of the standard deviation of J_{11} and J_{22}.

Note: The coherency is high at the edges.

0.12.5 Local orientation angle

The eigen vectors represent a rotation matrix. This allows us to compute the local orientation

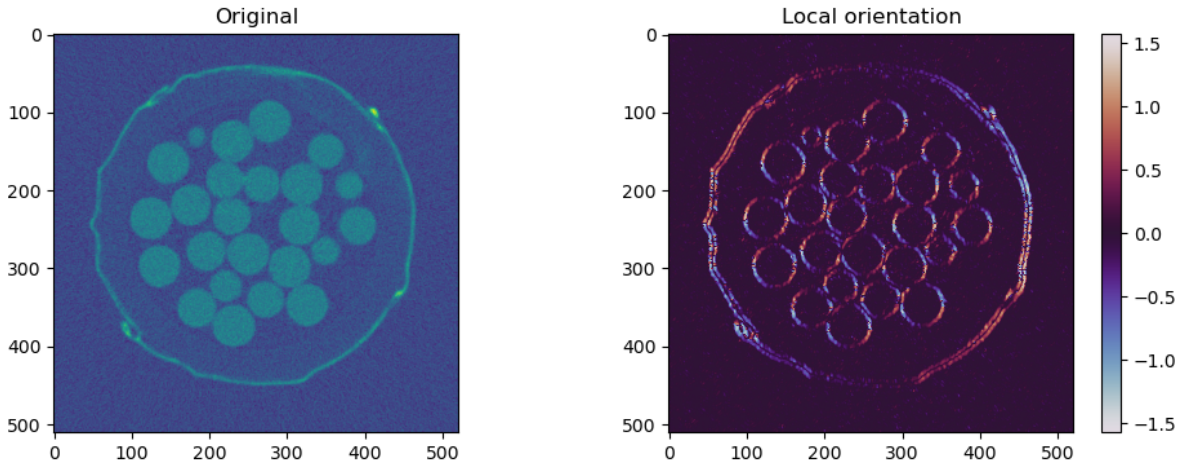
$$\theta = \frac{1}{2}\arctan\left(\frac{2J_{12}}{J_{22}-J_{11}}\right)$$

```
def orientation_angle(J,eps=0.1) :
    return 0.5*np.arctan2(2*J["J12"],J["J22"]-J["J11"]+eps)
```

Local orientations of the balls

```
ang = orientation_angle(J,eps=0.01)
```

```
# Visualization
fig,ax = plt.subplots(1,2,figsize=(12,4))
ax[0].imshow(balls)
ax[0].set_title('Original')
a=ax[1].imshow(ang,cmap='twilight',clim=[-np.pi/2,np.pi/2])
plt.colorbar(a,ax=ax[1])
ax[1].set_title('Local orientation');
```



0.12.6 Images with line textures

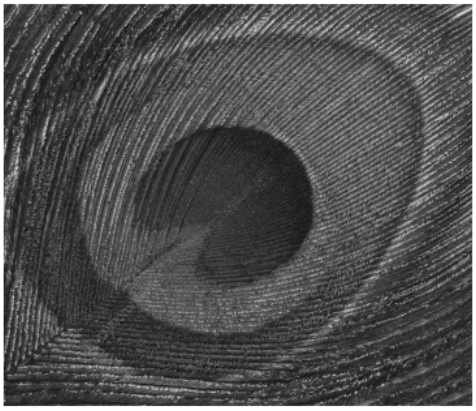
The ball image has a relatively simple structure with clear edges to compute the direction of. Now, we will turn our focus to textured images with collective orientations.

```
fig,ax=plt.subplots(1,2,figsize=(12,4))
ax[0].imshow(hair,cmap='gray')
ax[0].axis('off')
```

(continues on next page)

(continued from previous page)

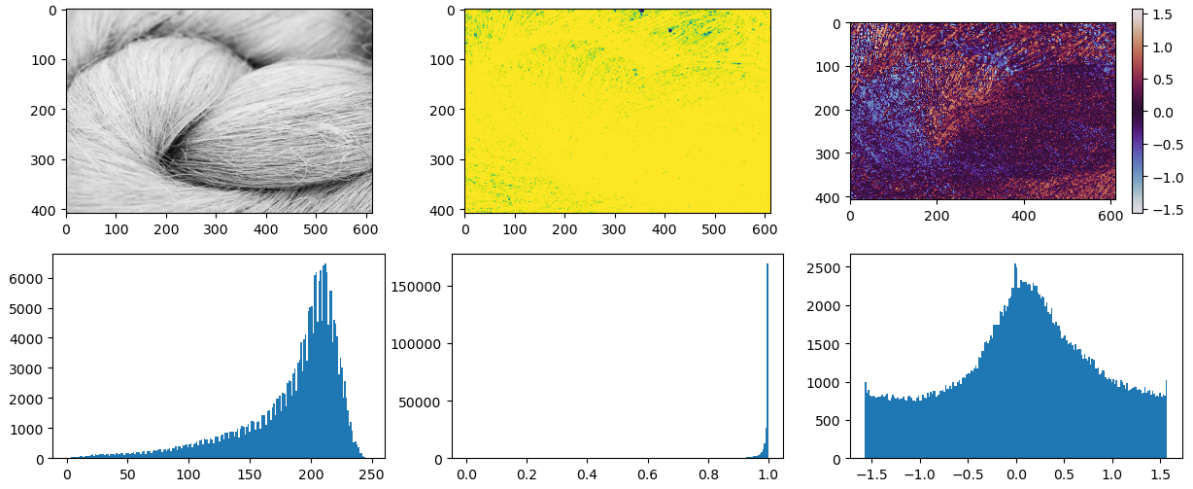
```
ax[1].imshow(pea,cmap='gray')
ax[1].axis('off');
```



Struture tensor the hair image

```
Jh = structure_tensor(hair,sigma=0.75)
Ch = coherency(Jh,eps=0.05)
Ah = orientation_angle(Jh,eps=0.1)
```

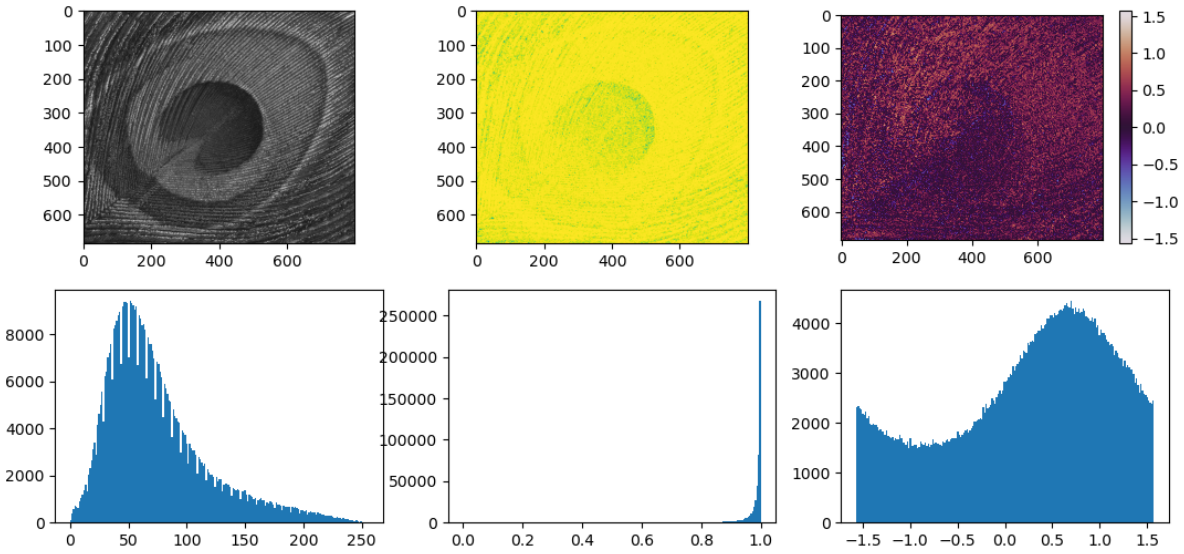
```
fig,axes = plt.subplots(2,3,figsize=(15,6))
axes = axes.ravel()

axes[0].imshow(hair,cmap='gray')
axes[1].imshow(Ch)
a2=axes[2].imshow(Ah,cmap='twilight',clim=[-np.pi/2,np.pi/2])
fig.colorbar(a2,ax=axes[2])
axes[3].hist(hair.ravel(),bins=200);
axes[4].hist(Ch.ravel(),bins=200);
axes[5].hist(Ah.ravel(),bins=200);
```



Structure tensor of the peacock feather

```
Jp = structure_tensor(pea,sigma=0.75)
Cp = coherency(Jp,eps=1)
Ap = orientation_angle(Jp,eps=0.1)
```

```
fig,axes = plt.subplots(2,3,figsize=(13,6))
axes = axes.ravel()

axes[0].imshow(pea,cmap='gray')
axes[1].imshow(Cp)
a2=axes[2].imshow(Ap,cmap='twilight',clim=[-np.pi/2,np.pi/2])
fig.colorbar(a2,ax=axes[2])
axes[3].hist(pea.ravel(),bins=200);
axes[4].hist(Cp.ravel(),bins=200);
axes[5].hist(Ap.ravel(),bins=200);
```



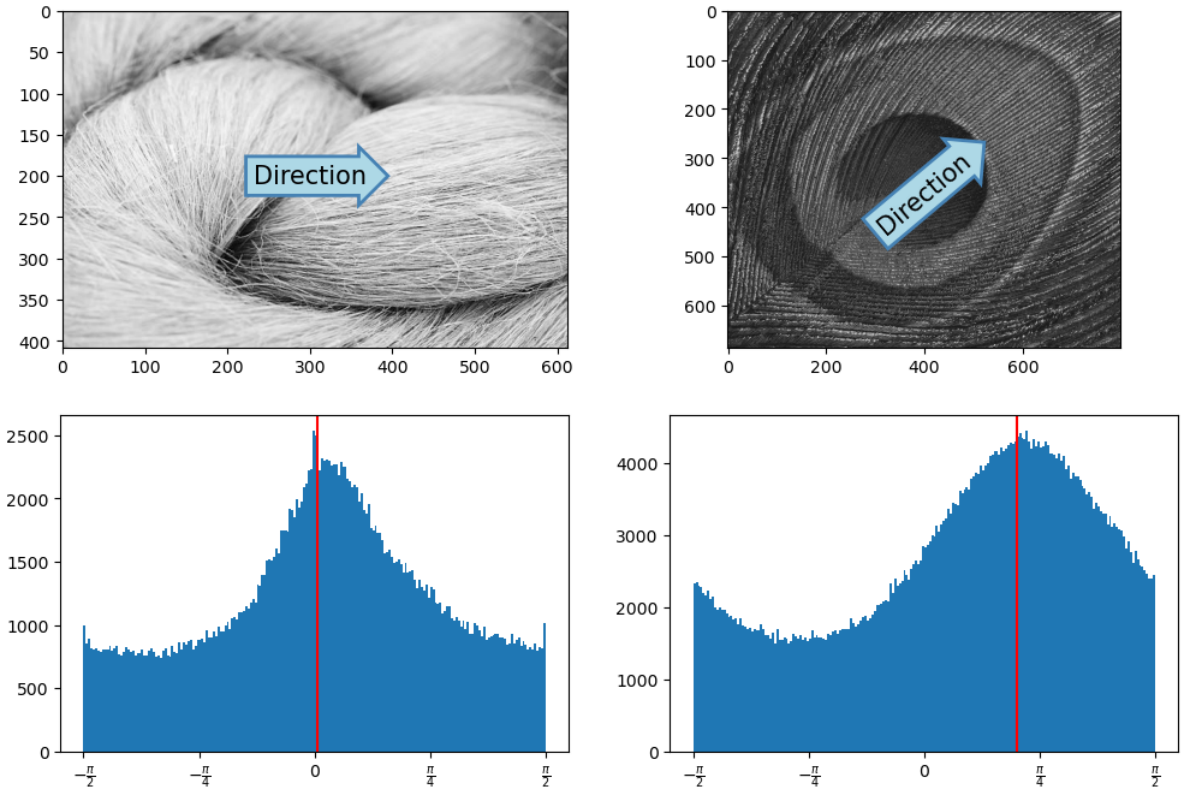
0.12.7 Compare the orientations globally

```
fig,ax = plt.subplots(2,2,figsize=(12,8))
ax=ax.ravel()
ax[0].imshow(hair,cmap='gray')
t = ax[0].text(300,200,  "Direction",
            ha="center", va="center", rotation=0, size=15,
            bbox=dict(boxstyle="rarrow,pad=0.3",
                      fc="lightblue", ec="steelblue", lw=2))
ax[1].imshow(pea,cmap='gray');
t = ax[1].text(400,370,  "Direction",
            ha="center", va="center", rotation=40, size=15,
            bbox=dict(boxstyle="rarrow,pad=0.3",
                      fc="lightblue", ec="steelblue", lw=2))

hh=ax[2].hist(Ah.ravel(),bins=200);
hhm=medfilt(hh[0],9)
```

(continues on next page)

(continued from previous page)

```
ap=hh[1][np.argmax(hhm)]
ax[2].axvline(x=ap, color='r')
ax[2].set_xticks([-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2])
ax[2].set_xticklabels([r'$-\frac{\pi}{2}$',r'$-\frac{\pi}{4}$',0,r'$\frac{\pi}{4}$',r'
↪$\frac{\pi}{2}$']);

hp=ax[3].hist(Ap.ravel(),bins=200);
hpm=medfilt(hp[0],9)
ap=hp[1][np.argmax(hpm)]
ax[3].axvline(x=ap, color='r')
ax[3].set_xticks([-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2])
ax[3].set_xticklabels([r'$-\frac{\pi}{2}$',r'$-\frac{\pi}{4}$',0,r'$\frac{\pi}{4}$',r'
↪$\frac{\pi}{2}$']);
```



0.12.8 Compare orientations locally

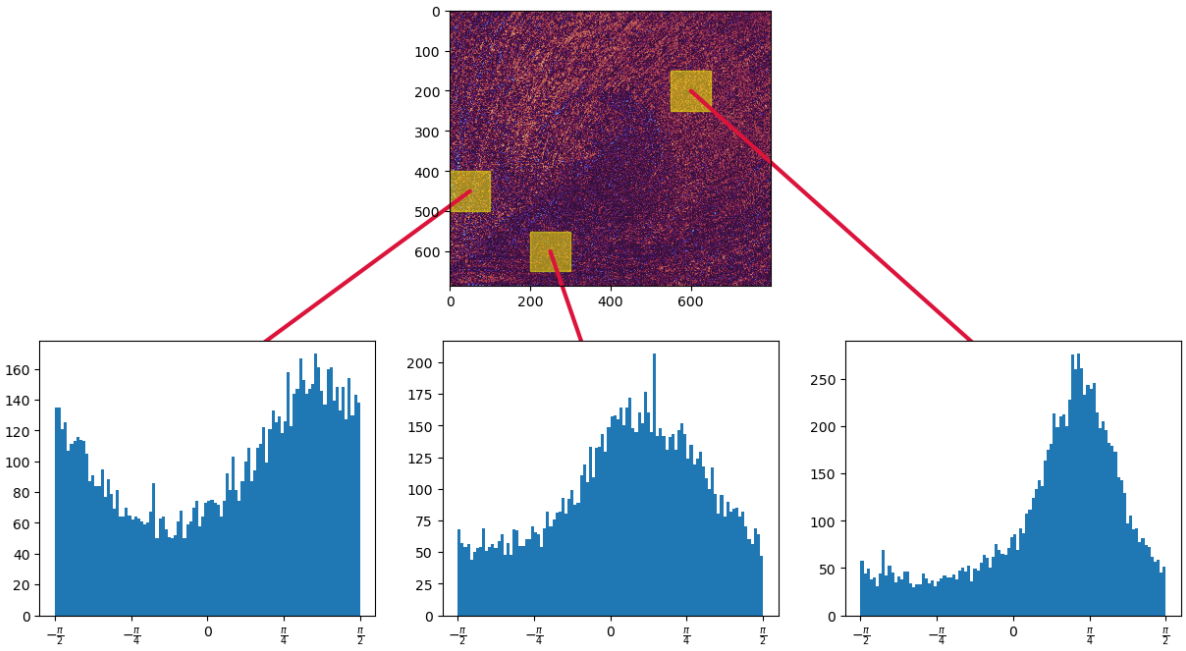
In both cases, we see that the global orientation is unable to capture that there are local variations. Now, let's take a look at the local histograms in some regions of the peacock feather.

```
fig,ax = plt.subplots(2,3,figsize=(15,8))
ax=ax.ravel()
ax[0].axis('off')
ax[2].axis('off')
ax[1].imshow(Ap,cmap='twilight')

ax[3].hist(Ap[400:500,0:100].ravel(),bins=100);
```

(continues on next page)

(continued from previous page)

```
ax[3].set_xticks([-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2])
ax[3].set_xticklabels([r'$-\frac{\pi}{2}$',r'$-\frac{\pi}{4}$',0,r'$\frac{\pi}{4}$',r'
↪$\frac{\pi}{2}$'])

ax[4].hist(Ap[550:650,200:300].ravel(),bins=100);
ax[4].set_xticks([-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2])
ax[4].set_xticklabels([r'$-\frac{\pi}{2}$',r'$-\frac{\pi}{4}$',0,r'$\frac{\pi}{4}$',r'
↪$\frac{\pi}{2}$'])

ax[5].hist(Ap[150:250,550:650].ravel(),bins=100);
ax[5].set_xticks([-np.pi/2,-np.pi/4,0,np.pi/4,np.pi/2])
ax[5].set_xticklabels([r'$-\frac{\pi}{2}$',r'$-\frac{\pi}{4}$',0,r'$\frac{\pi}{4}$',r'
↪$\frac{\pi}{2}$'])



rect3 = plt.Rectangle((0,400), 100, 100, facecolor="yellow", alpha=0.5,ec="yellow")
ax[1].add_patch(rect3)



rect4 = plt.Rectangle((200,550), 100, 100,
                      facecolor="yellow", alpha=0.5,ec="yellow")
ax[1].add_patch(rect4)

rect5 = plt.Rectangle((550,150), 100, 100, facecolor="yellow", alpha=0.5,ec="yellow")
ax[1].add_patch(rect5)



con3 = ConnectionPatch(xyA=(0,150), xyB=(50,450),
                       coordsA="data", coordsB="data",
                       axesA=ax[3], axesB=ax[1],
                       color="crimson", lw=3)
ax[1].add_artist(con3)

con4 = ConnectionPatch(xyA=(0,150), xyB=(250,600),
                       coordsA="data", coordsB="data",
                       axesA=ax[4], axesB=ax[1],
                       color="crimson", lw=3)
ax[1].add_artist(con4)


con5 = ConnectionPatch(xyA=(0,250), xyB=(600,200),
                       coordsA="data", coordsB="data",
                       axesA=ax[5], axesB=ax[1],
                       color="crimson", lw=3)
ax[1].add_artist(con5);
```



Quantifying local orientations

It is mostly not relevant to observe the orientation per pixel, but rather in a region.

How can we get the region information

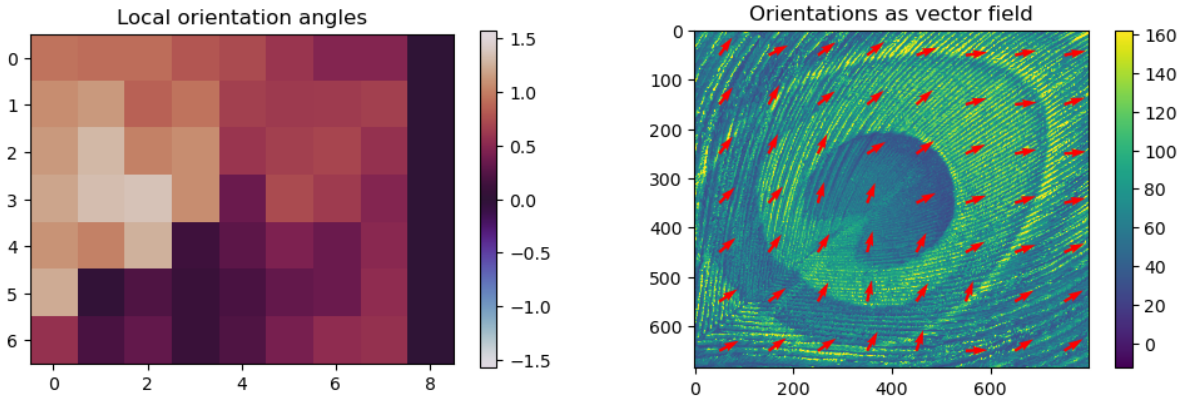
- We saw that there is a uniformly distributed bias in the angles
- The avagerage will be misleading, use max of histogram.

```
def local_angle(img, w) :
    res=np.zeros([1+img.shape[0]//w,1+img.shape[1]//w])

    for rr,r in enumerate(np.arange(0,img.shape[0],w)) :
        for rc,c in enumerate(np.arange(0,img.shape[1],w)) :
            h,a = np.histogram(img[r:r+w,c:c+w].ravel(),bins=w)
            h=medfilt(h)
            res[rr,rc]= a[np.argmax(h)]
    return res
```

Compute the local orientation on the peacock feather

```
# Compute the average angle in a window w
w=100
Ap_avg = local_angle(Ap,w)

# Compute positions
r,c = np.meshgrid(np.arange(0,Ap_avg.shape[0])*w+w/2,np.arange(0,Ap_avg.shape[1]-
↪1)*w+w/2)
# Compute the vectors
u = np.cos(Ap_avg[:,:-1])
v = np.sin(Ap_avg[:,:-1])
```

```
fig,ax = plt.subplots(1,2,figsize=(12,3.5))

a = ax[0].imshow(Ap_avg,clim=[-np.pi/2, np.pi/2],cmap='twilight')
ax[0].set_title('Local orientation angles')
fig.colorbar(a,ax=ax[0])
m,s=pea.mean(), pea.std()
a1=ax[1].imshow(pea,clim=[m-2*s,m+2*s])
fig.colorbar(a1,ax=ax[1])
ax[1].quiver( c,r, u,v, color='red');
ax[1].set_title('Orientations as vector field');
```



0.13 Verification

0.13.1 How good is my filter?

0.13.2 Verify the correctness of the method

"Data massage"

Filtering manipulates the data, avoid too strong modifications otherwise you may invent new image features!!!

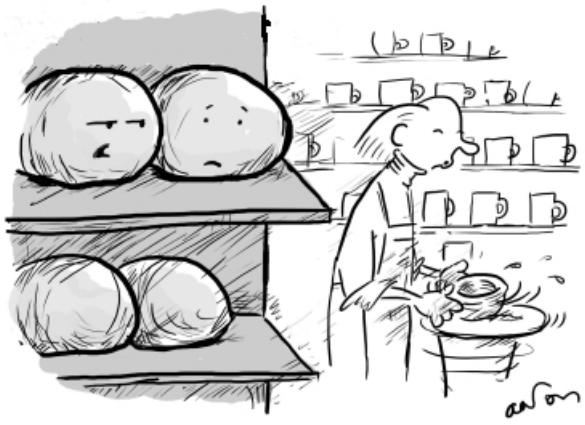


Fig. 1: Be careful when you apply different filters. You may make to great modifications.

Verify the validity your method

- Visual inspection
- Difference images
- Use degraded phantom images in a “smoke test”

0.13.3 Verification using difference images

Compute pixel-wise difference between image f and g

Difference images provide first diagnosis about processing performance

0.13.4 Performance testing - “The smoke test”

- Testing term from electronic hardware testing - drive the system until something fails due to overheating…
- In general: scan the parameter space for different SNR until the method fails to identify strength and weakness of the system.

Test strategy

1. Create a phantom image with relevant features.
2. Add noise for different SNR to the phantom.
3. Apply the processing method with different parameters.
4. Measure the difference between processed and phantom.
5. Repeat steps 2-4 N times for better test statistics.
6. Plot the results and identify the range of SNR and parameters that produce acceptable results.

0.13.5 Data for evaluation - Phantom data

General purpose can be controlled

- Data with known features.
- Parameters can be changed.
  - Shape
  - Sharpness
  - Contrast
  - Noise (distribution and strength)

Fig. 2: The shepp logan phantom.

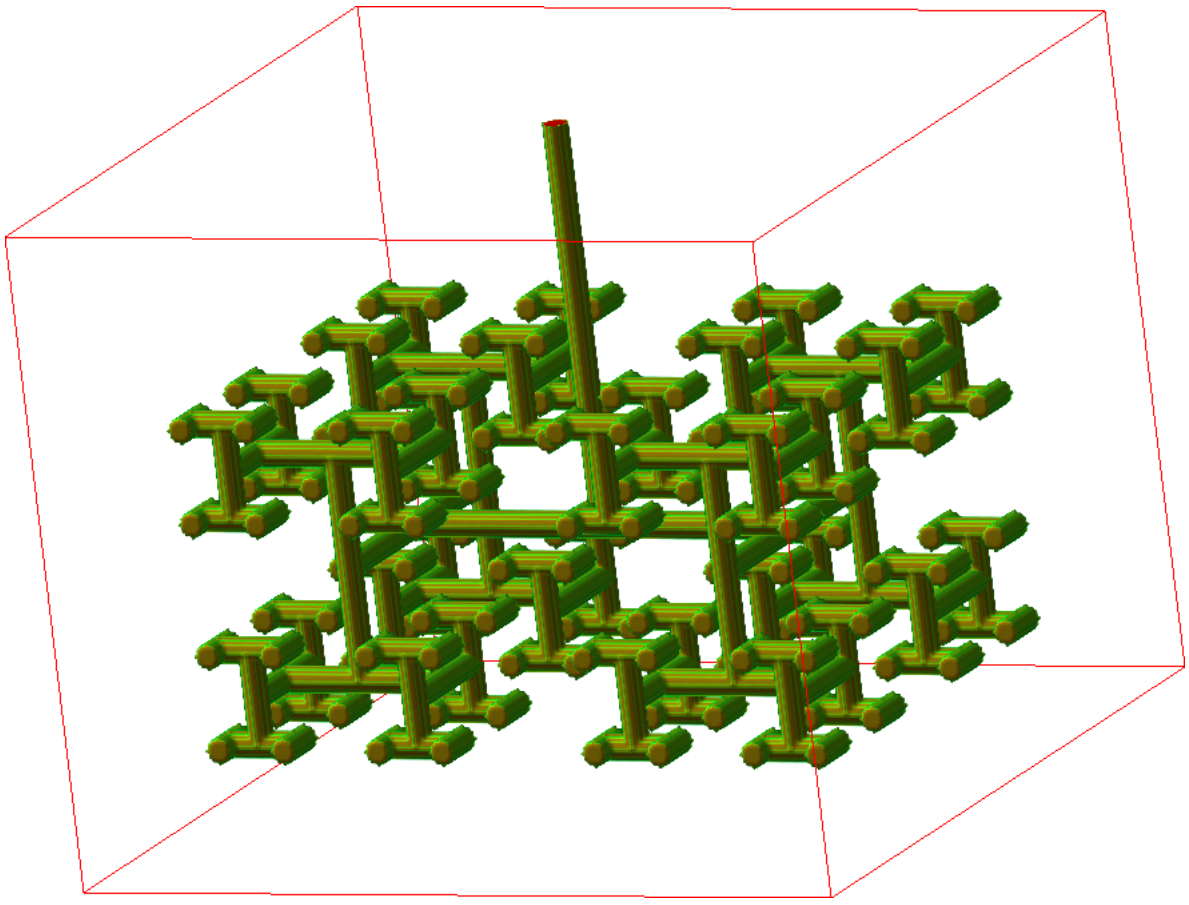


Fig. 3: A simulated root network.

0.13.6 Data for evaluation - Labelled data

Often ‘real’ data

- Labeled by experts
- Used for training and validation
  - Training of model
  - Validation
  - Test

Fig. 4: Images of hand written numbers from the MNIST data base.

0.13.7 Evaluation metrics for gray level images

An evaluation procedure needs a metric to compare the performance

Mean squared error

$$MSE(f,g) = \sum_{p \in \Omega} (f(p) - g(p))^2$$

Structural similarity index

$$SSIM(f,g) = \frac{(2\mu_f \mu_g + C_1)(2\sigma_{fg} + C_2)}{(\mu_f^2 + \mu_g^2 + C_1)(\sigma_f^2 + \sigma_g^2 + C_2)}$$

- μ_f, μ_g Local mean of f and g.
- σ_{fg} Local correlation between f and g.
- σ_f, σ_g Local standard deviation of f and g.
- C_1, C_2 Constants based on the image dynamics (small numbers).

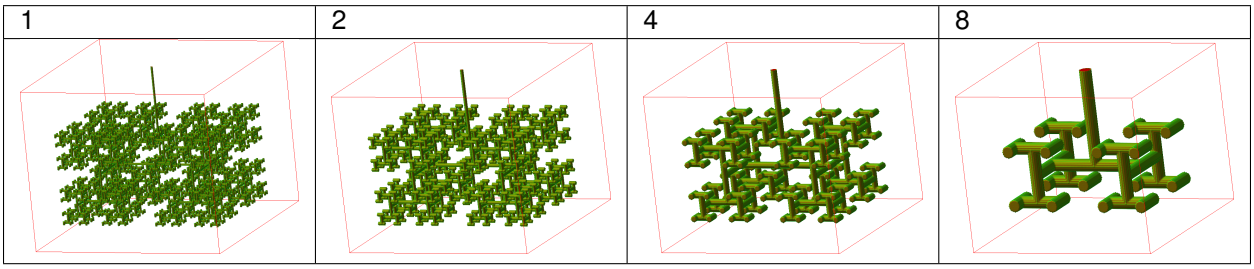
$$MSSIM(f,g) = E[SSIM(f,g)]$$

Wang 2009

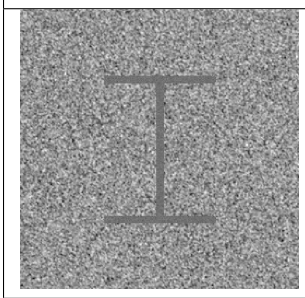
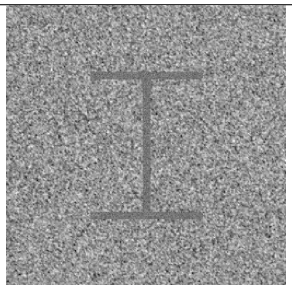
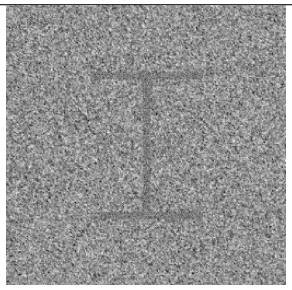
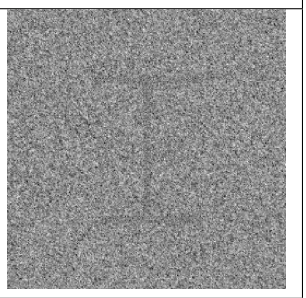
0.13.8 Test run example

Runnig tests with different structure sizes and SNR.

Phantom structure sizes

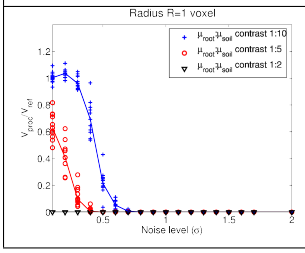
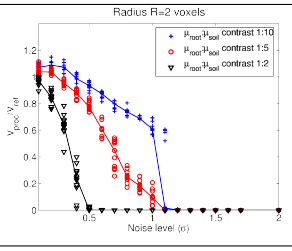
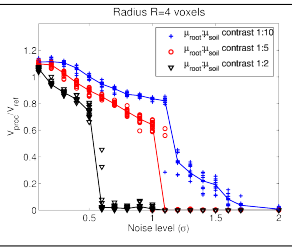
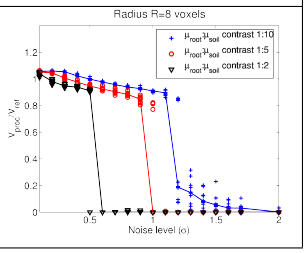


Change SNR and contrast

σ=1	σ=2	σ=5	σ=10
			

Process

Plot results

R=1	R=2	R=4	R=8
			

Kaestner et al. 2006

0.14 Overview

0.14.1 Many filters

0.14.2 Details of filter performance

0.14.3 Take-home message

We have looked at different ways to suppress noise and artifacts:

- Convolution
- Median filters
- Wavelet denoising
- PDE filters

Which one you select depends on

- Purpose of the data
- Quality requirements

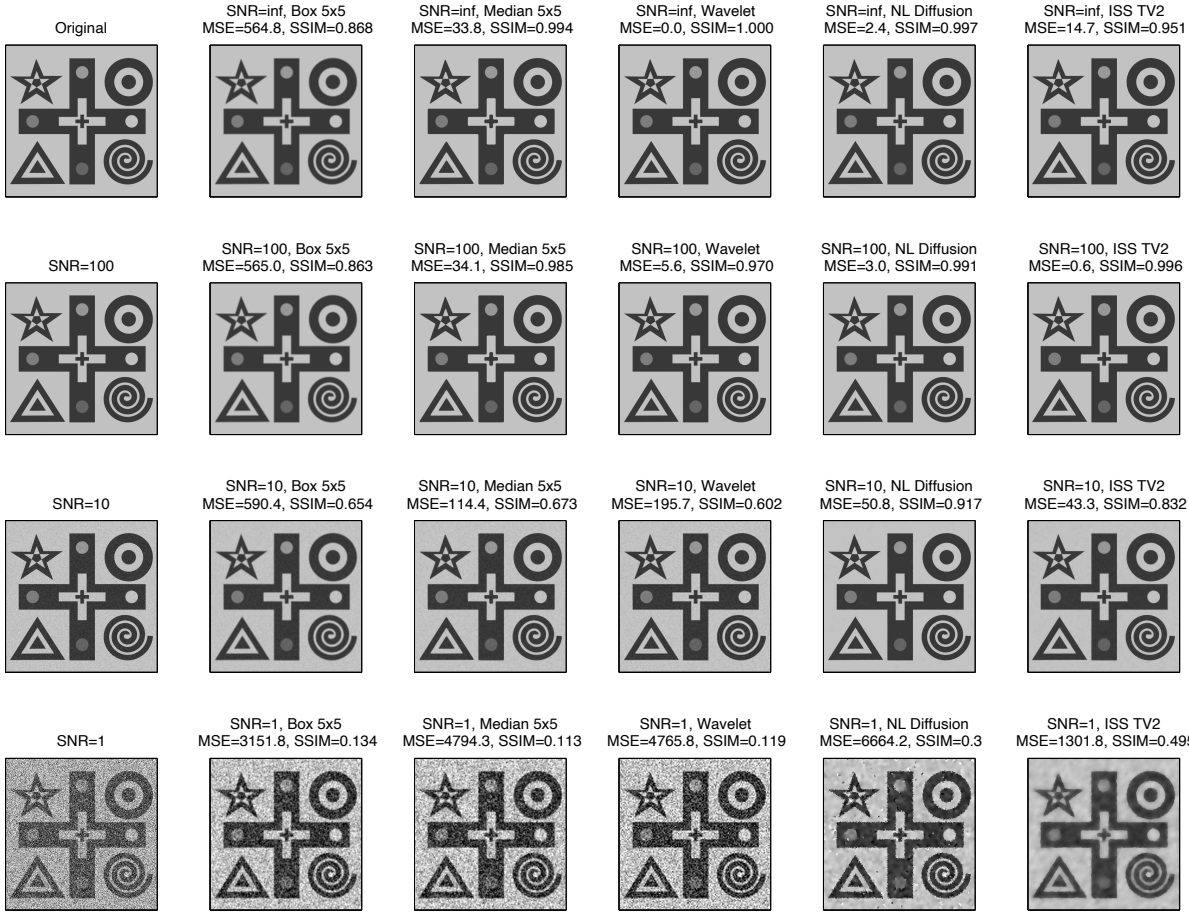


Fig. 1: All filters form the lecture for different SNR.

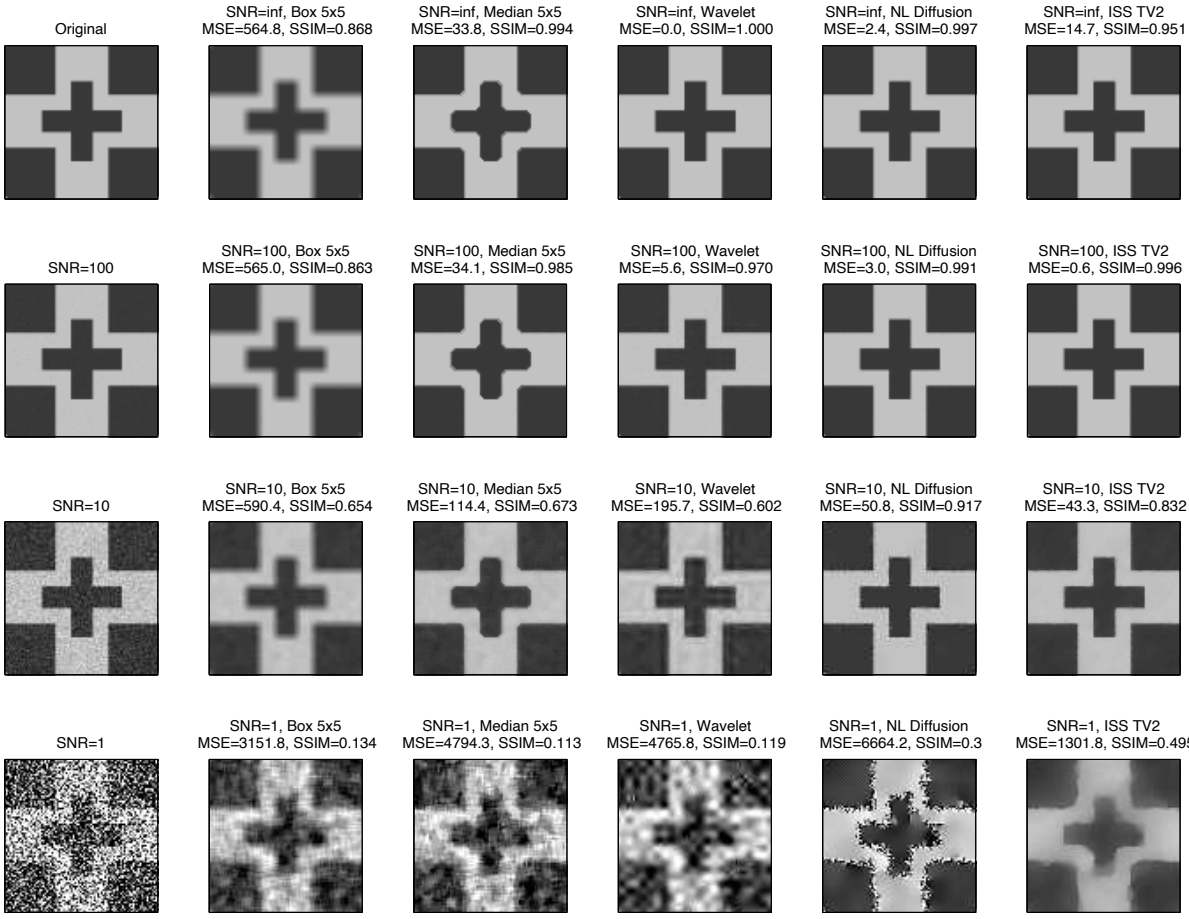


Fig. 2: A close-up of all filters form the lecture for different SNR.

- Available time

Filters can also be used to provide quantitative information