Quantitative Big Imaging - Image enhancement

Anders Kaestner

Mar 04, 2021

CONTENTS

Quantitative Big Imaging ETHZ: 227-0966-00L

CHAPTER

ONE

TODAYS LECTURE

- Imperfect images and noise
- Filters
- Advanced filters
- Evaluation workflows

1.1 We need some modules

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

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import skimage as ski
import skimage.io as io
from skimage.morphology import disk
import scipy.ndimage as ndimage
import matplotlib        as mpl
mpl.rcParams['figure.dpi'] = 300
```

```
---------------------------------------------------------------------------
ModuleNotFoundError                       Traceback (most recent call last)
<ipython-input-1-9de9c7643c12> in <module>
----> 1 get_ipython().run_line_magic('matplotlib', 'inline')
      2 import matplotlib.pyplot as plt
      3 import numpy as np
      4 import skimage as ski
      5 import skimage.io as io

~/miniconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py in run_line_
↪magic(self, magic_name, line, _stack_depth)
   2342                 kwargs['local_ns'] = self.get_local_scope(stack_depth)
   2343             with self.builtin_trap:
-> 2344                 result = fn(*args, **kwargs)
   2345             return result
   2346

<decorator-gen-101> in matplotlib(self, line)

~/miniconda3/lib/python3.8/site-packages/IPython/core/magic.py in <lambda>(f, *a, **k)
```

(continues on next page)

(continued from previous page)

```
    185     # but it's overkill for just that one bit of state.
    186     def magic_deco(arg):
--> 187         call = lambda f, *a, **k: f(*a, **k)
    188
    189         if callable(arg):

~/miniconda3/lib/python3.8/site-packages/IPython/core/magics/pylab.py in␣
↪matplotlib(self, line)
     97             print("Available matplotlib backends: %s" % backends_list)
     98         else:
---> 99             gui, backend = self.shell.enable_matplotlib(args.gui.lower() if␣
↪isinstance(args.gui, str) else args.gui)
    100             self._show_matplotlib_backend(args.gui, backend)
    101

~/miniconda3/lib/python3.8/site-packages/ipykernel/zmqshell.py in enable_
↪matplotlib(self, gui)
    599
    600     def enable_matplotlib(self, gui=None):
--> 601         gui, backend = super(ZMQInteractiveShell, self).enable_matplotlib(gui)
    602
    603         from ipykernel.pylab.backend_inline import configure_inline_support

~/miniconda3/lib/python3.8/site-packages/IPython/core/interactiveshell.py in enable_
↪matplotlib(self, gui)
   3511         """
   3512         from IPython.core import pylabtools as pt
-> 3513         gui, backend = pt.find_gui_and_backend(gui, self.pylab_gui_select)
   3514
   3515         if gui != 'inline':

~/miniconda3/lib/python3.8/site-packages/IPython/core/pylabtools.py in find_gui_and_
↪backend(gui, gui_select)
    278     """
    279
--> 280     import matplotlib
    281
    282     if gui and gui != 'auto':

ModuleNotFoundError: No module named 'matplotlib'
```

CHAPTER

TWO

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.

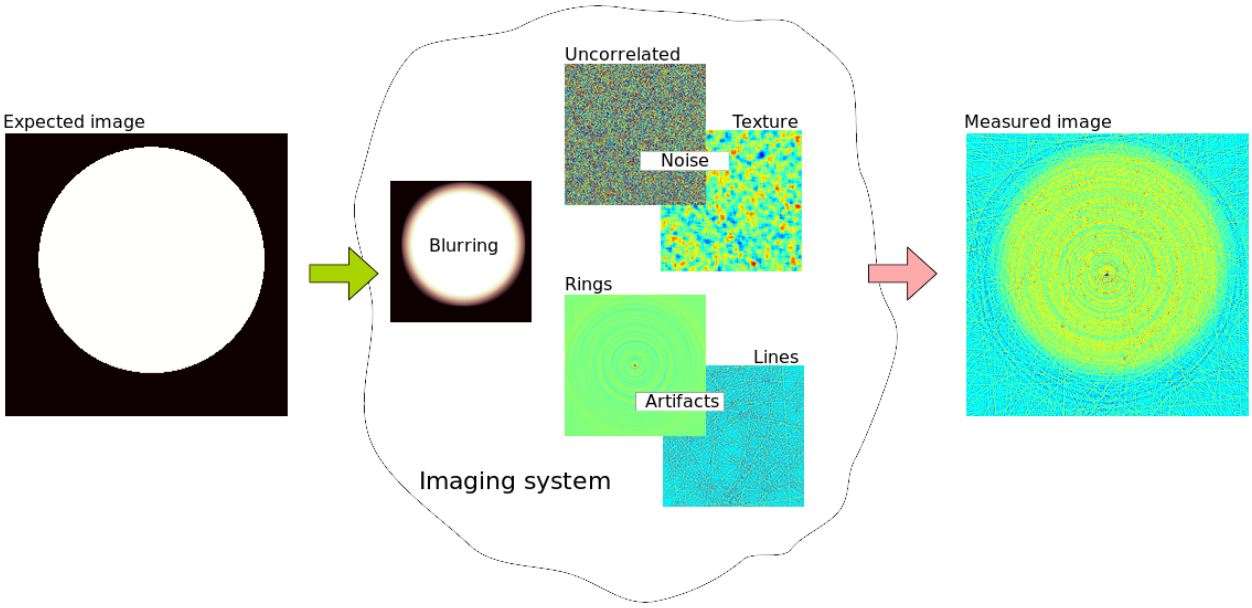


Fig. 2.1: Schematic showing the error contributions in an imperfect imaging system.

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

2.1.1 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.

2.1.2 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.

2.1.3 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.

2.1.4 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.

2.2 A typical processing chain

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

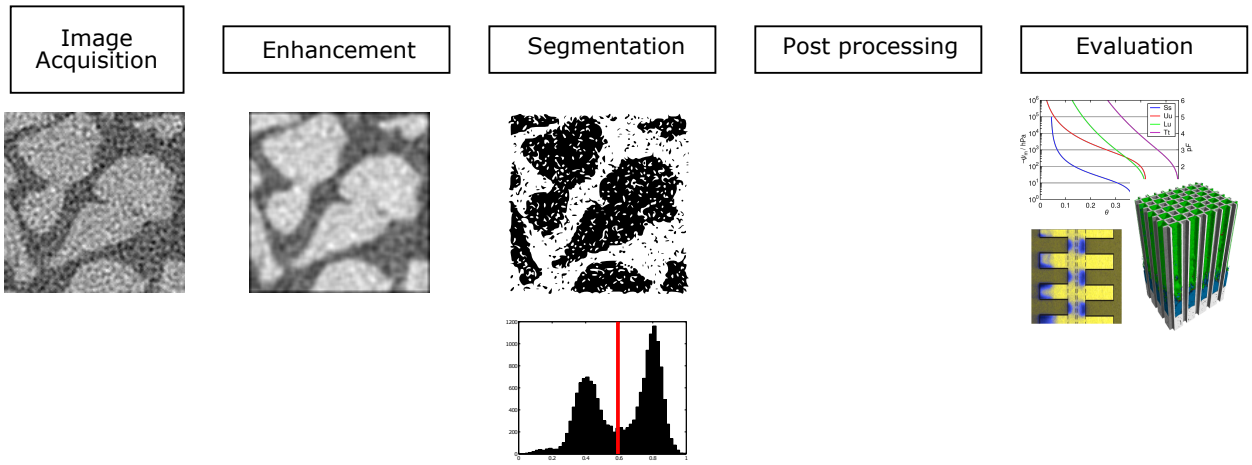


Fig. 2.2: Typical steps of an image processing work flow.

- **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.

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

CHAPTER

THREE

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.

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

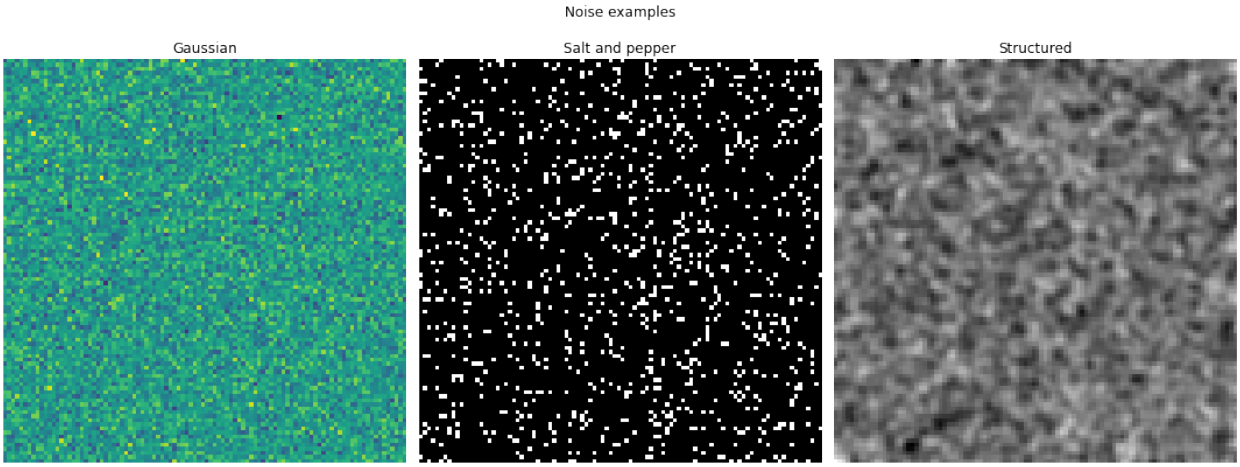
The even noise has a random activation function that triggers the event of each pixel with some probabilty. This noise type produces spots randomly distributed over the image. The spots may also have a randomw intensity.

- 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=[15,6]);
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');
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();
```



3.1.1 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.

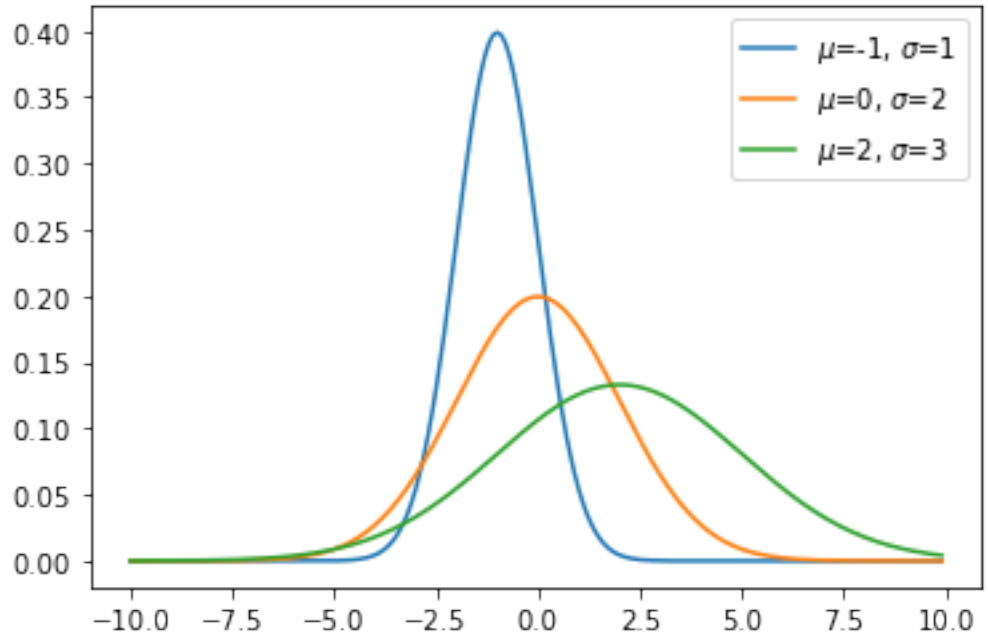
- Additive
- Easy to model
- Law of large numbers

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)

#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();
```



3.1.2 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.

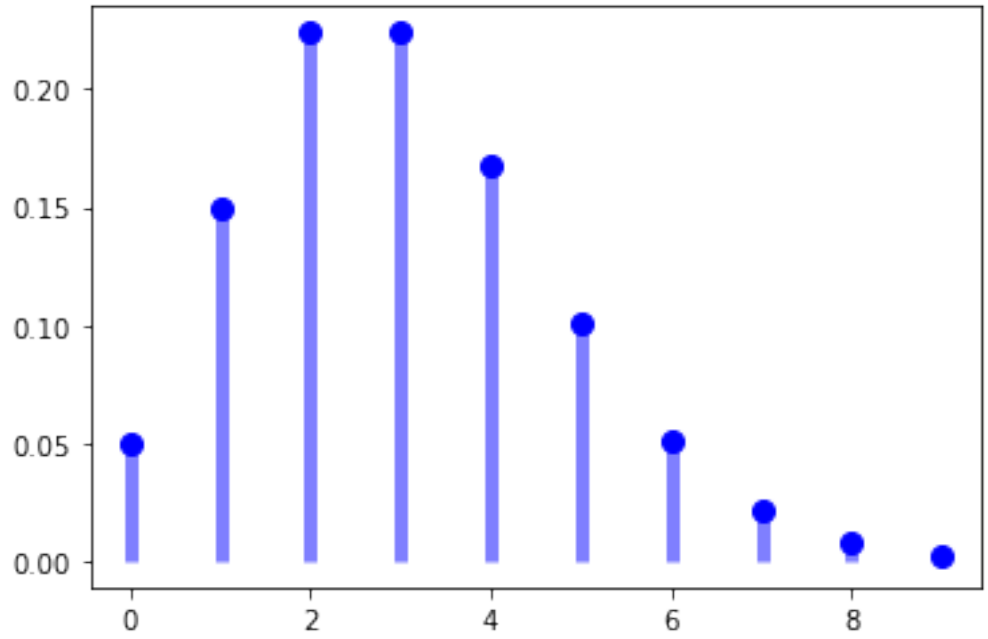
- Multiplicative
- Physically correct for event counting

Distribition function

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

The plot below show a poisson distribtion for λ=3

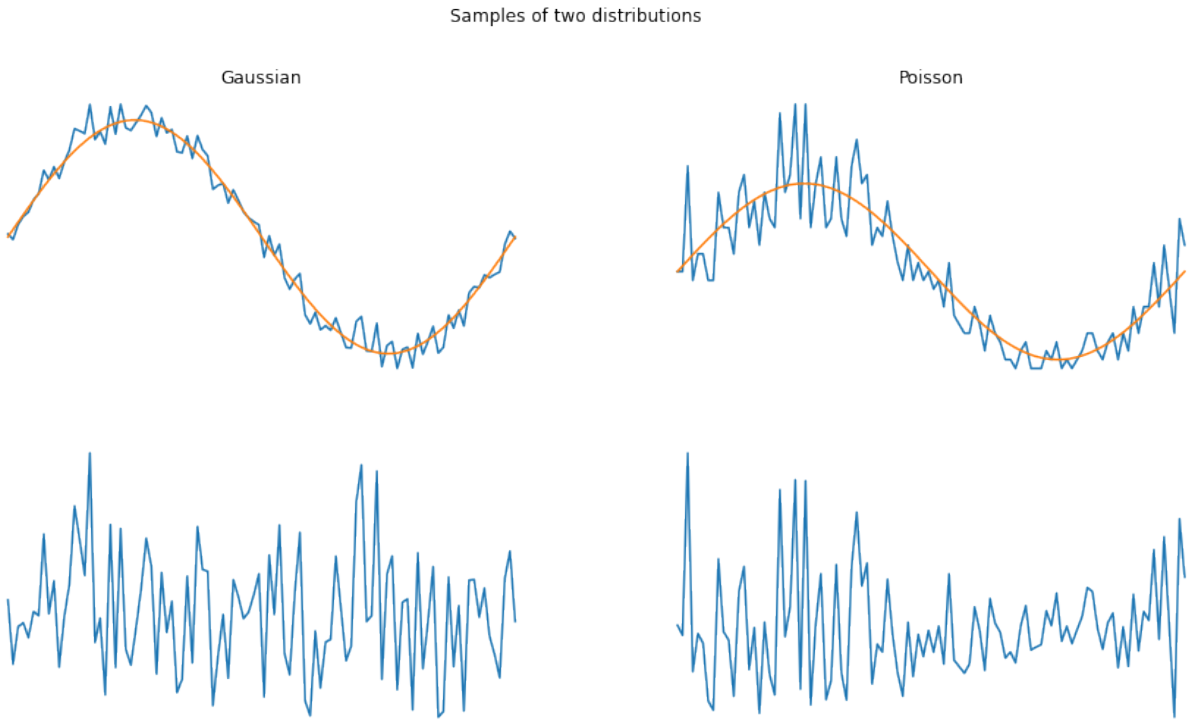
```
from scipy.stats import poisson
mu=3
fig, ax = plt.subplots(1, 1)
x = np.arange(poisson.ppf(0.01, mu),              poisson.ppf(0.999, mu))
ax.plot(x, poisson.pmf(x, mu), 'bo', ms=8, label='poisson pmf')
ax.vlines(x, 0, poisson.pmf(x, mu), colors='b', lw=5, alpha=0.5);
```



3.1.3 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.

```
plt.figure(figsize=[15,8])
x=np.linspace(0,2*np.pi,100);
y=10*np.sin(x)+11; ng=np.random.normal(0,1,size=len(x)); npoi = np.random.poisson(y);
plt.subplot(2,2,1); plt.plot(x,y+ng);plt.plot(x,y); plt.axis('off');plt.title(
↪'Gaussian'); plt.subplot(2,2,3);plt.plot(x,ng);plt.axis('off');
plt.subplot(2,2,2); plt.plot(x,npoi);plt.plot(x,y); plt.axis('off');plt.title('Poisson
↪'); plt.subplot(2,2,4);plt.plot(x,npoi-y);plt.axis('off');
plt.suptitle('Samples of two distributions');
```



3.1.4 Noise models - Salt'n'pepper noise

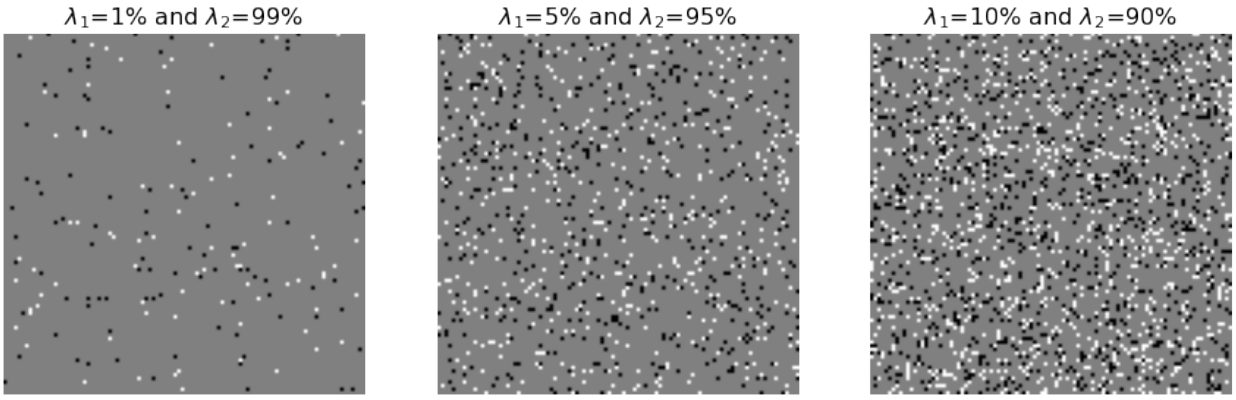
- A type of outlier noise
- Noise frequency described as probability of outlier
- Can be additive, multiplicative, and independent replacement

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

3.1.5 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');
```



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.

3.2.1 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.

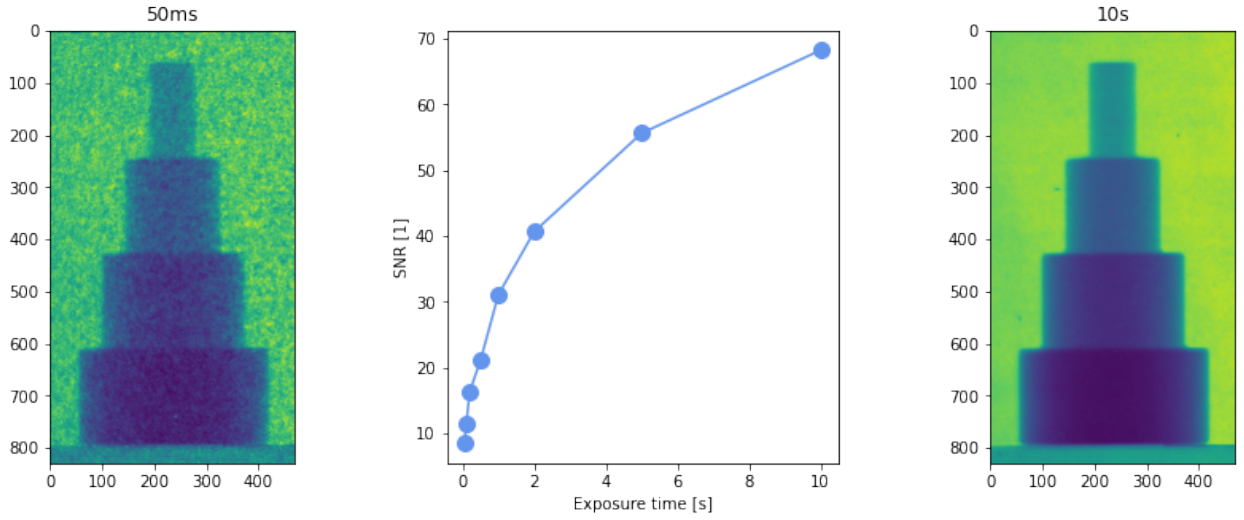
The signal to noise ratio can be improved by increasing the number of neutrons per pixel. This can be achived through increasing

- 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');
plt.figure(figsize=(15,5))
plt.subplot(1,3,2);plt.plot(exptime/1000,snr, **marker_style);plt.xlabel('Exposure␣
↪time [s]');plt.ylabel('SNR [1]')
img50ms=plt.imread('figures//tower_50ms.png'); img10000ms=plt.imread('figures/tower_
↪10000ms.png');
plt.subplot(1,3,1);plt.imshow(img50ms); plt.title('50ms'); plt.subplot(1,3,3); plt.
↪imshow(img10000ms),plt.title('10s');
```



3.3 Useful python functions

3.3.1 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])`

3.3.2 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.

CHAPTER

FOUR

BASIC FILTERING

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.

4.1.1 General definition

A filter is a processing unit that

- Enhances the wanted information
- Suppresses the unwanted information

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.

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.

4.2.1 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.

- Slow changes are enhanced
- Rapid changes are suppressed

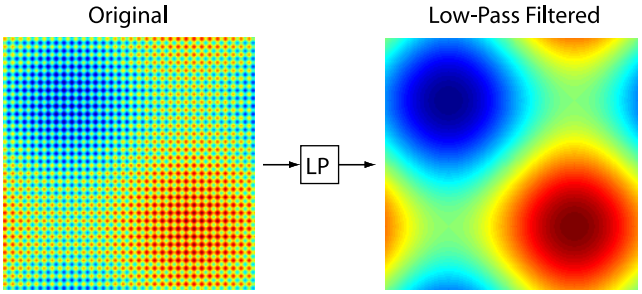


Fig. 4.1: The principle of a low-pass filter.

4.2.2 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.

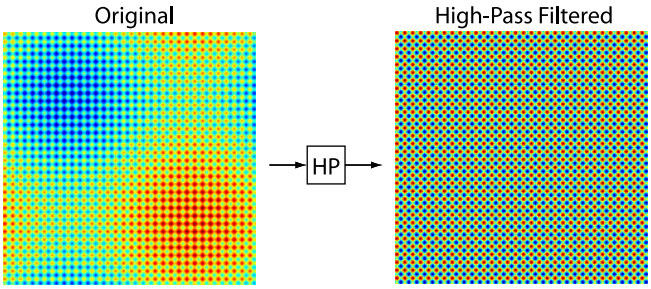


Fig. 4.2: The principle of a high-pass filter.

- Rapid changes are enhanced
- Slow changes are suppressed

CHAPTER

FIVE

BASIC FILTERS

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

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

$$B = \frac{1}{25} \cdot \begin{array}{|c|c|c|c|c|} \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 \\ \hline \end{array}$$

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:

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

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.

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

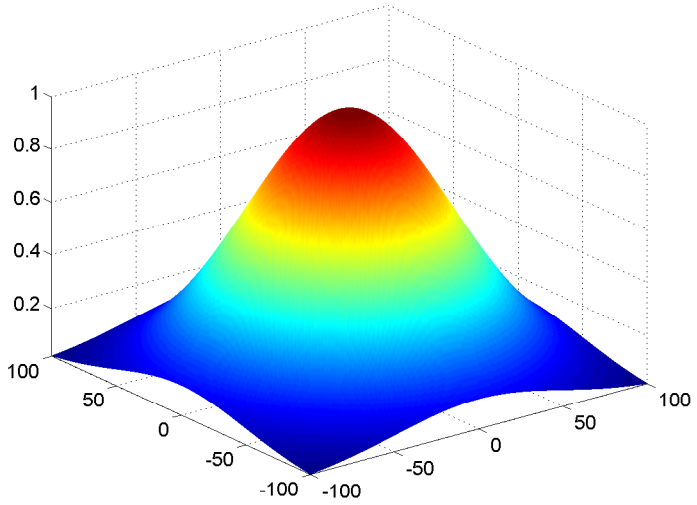


Fig. 5.1: The shape of a Gaussian filter kernel.

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

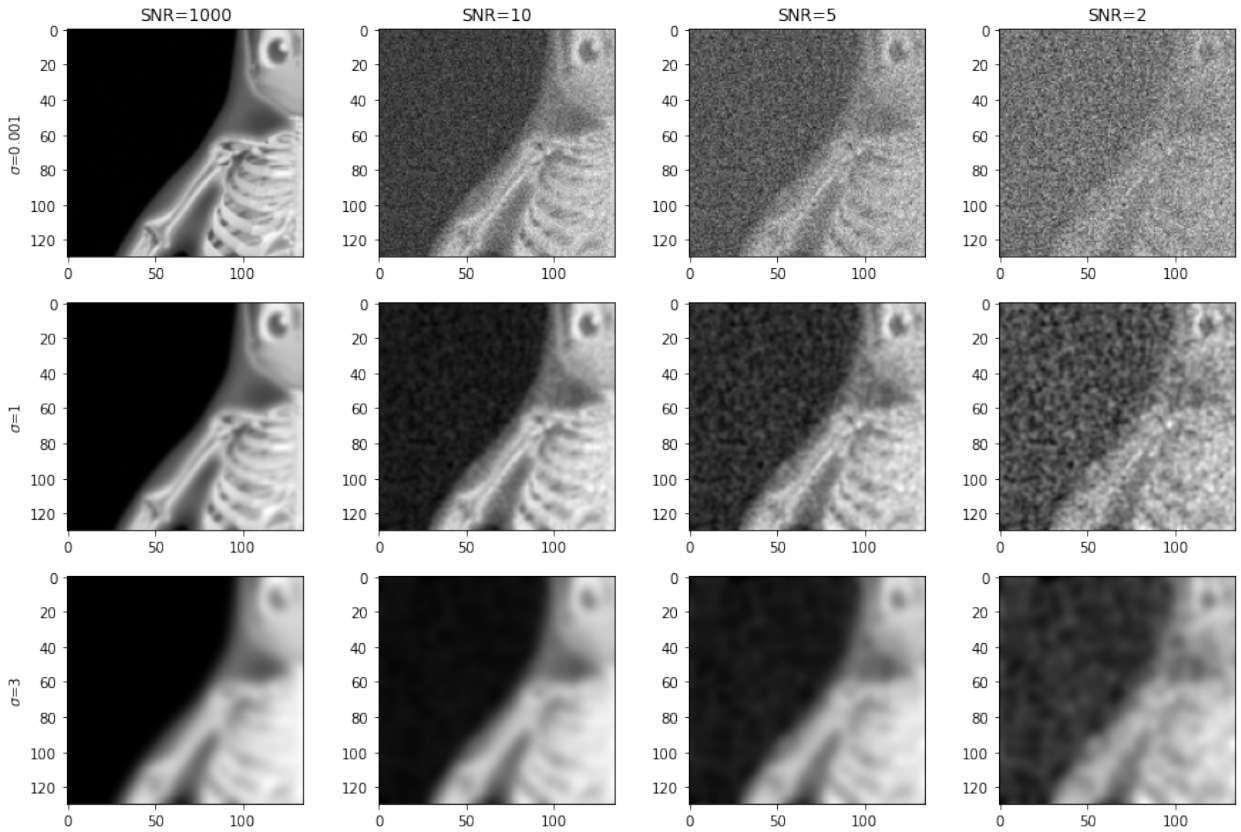
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1
1	1	1	1	1

Example:

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.

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.

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:

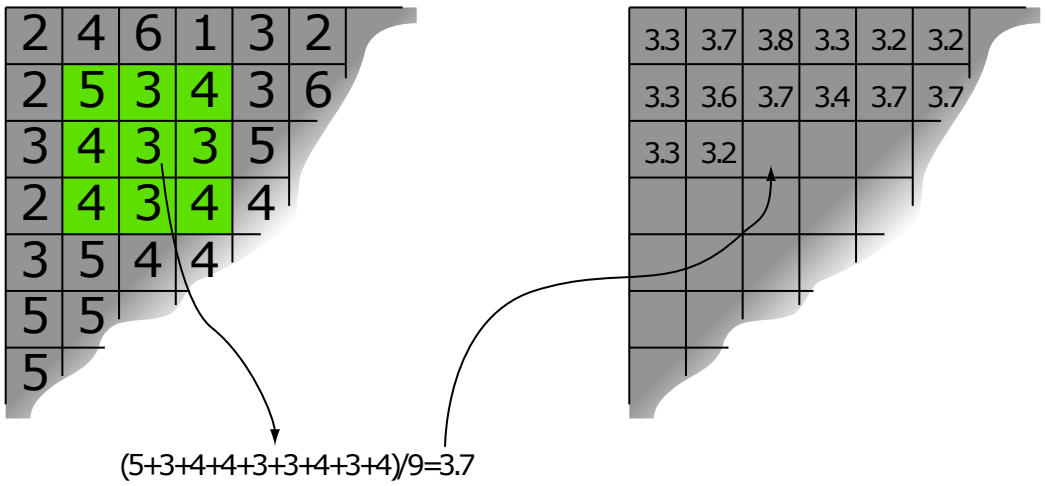


Fig. 5.2: Updating one pixel using a 3x3 box filter.

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

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

5.5.1 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 three 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.

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.

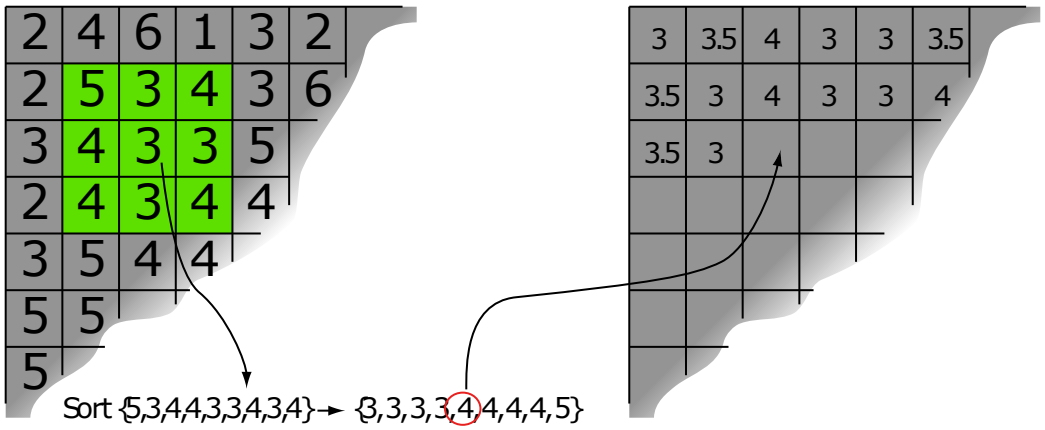
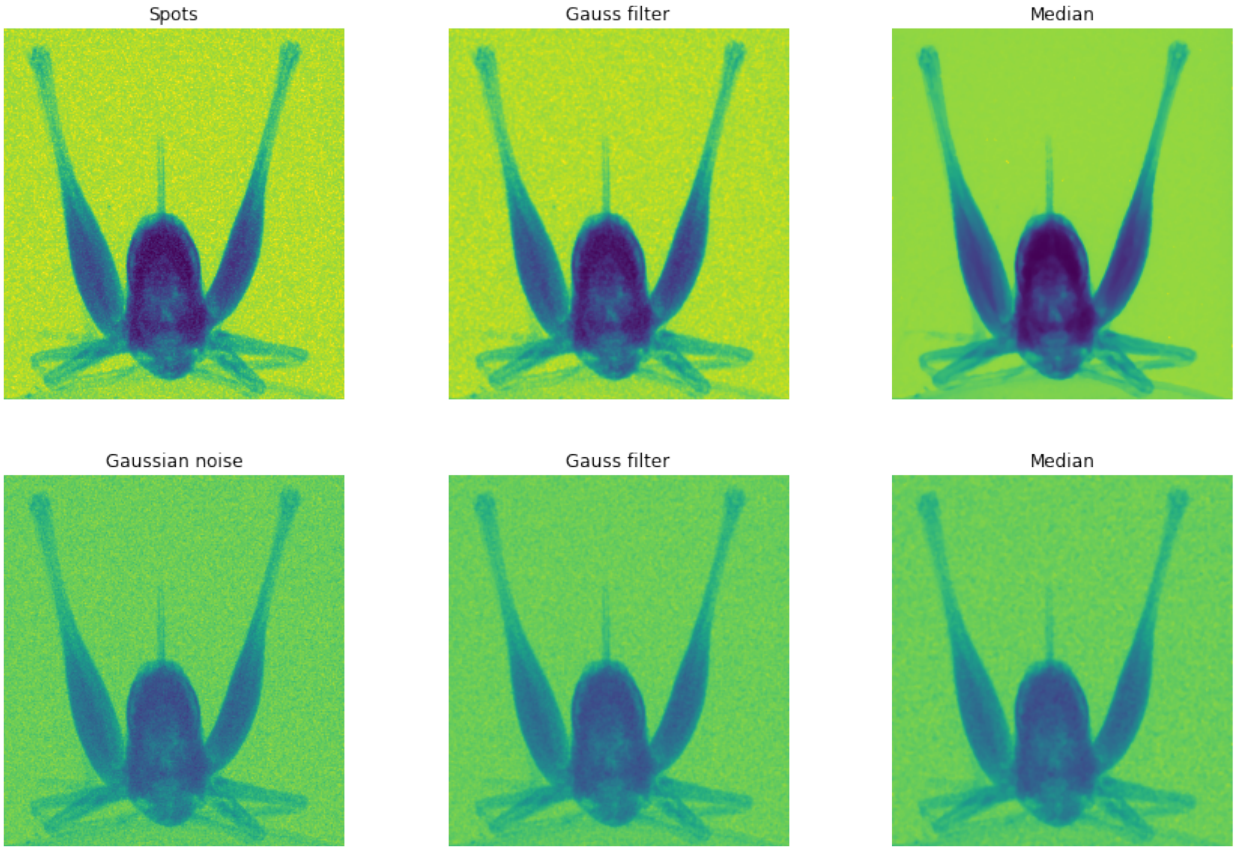


Fig. 5.3: Updating a pixel from its neighborhood using a 3x3 median filter.

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;
plt.subplot(2,3,1); plt.imshow(spots,vmin=vmin,vmax=vmax); plt.title('Spots'); plt.
↪axis('off');
plt.subplot(2,3,2); plt.imshow(ski.filters.gaussian(spots,sigma=1),vmin=vmin,
↪vmax=vmax); plt.title('Gauss filter'); plt.axis('off');
plt.subplot(2,3,3); plt.imshow(ski.filters.median(spots,disk(3)),vmin=vmin,vmax=vmax);
↪ plt.title('Median'); plt.axis('off');

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

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

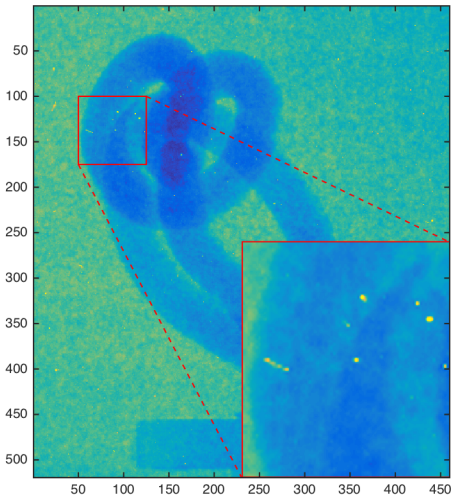


Fig. 5.4: A neutron image with outliers that we want to remove.

5.9 Spot cleaning algorithm

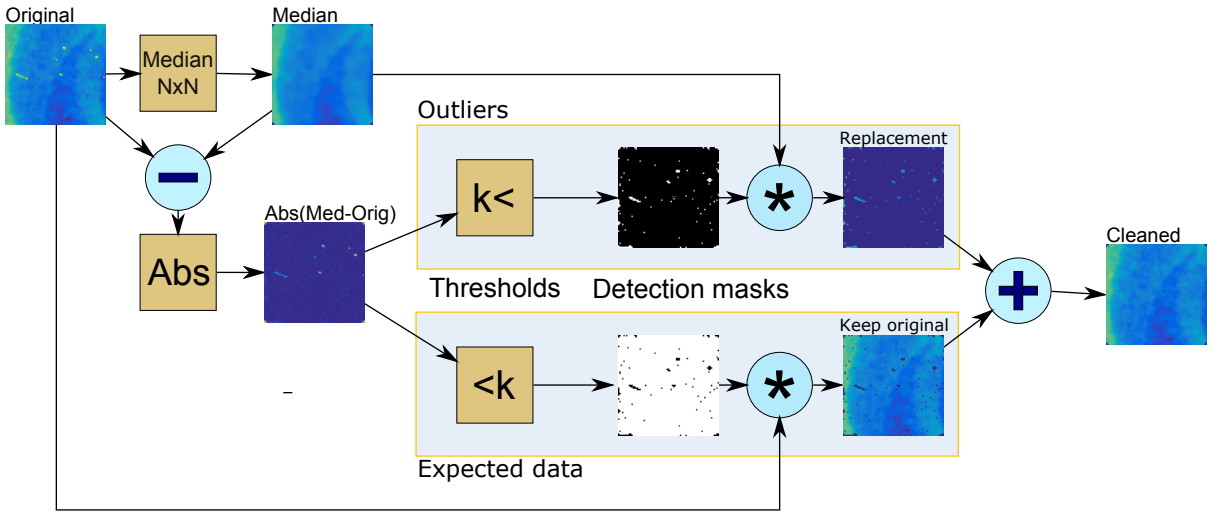


Fig. 5.5: Process workflow to selectively remove spots from an image.

Parameters

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

5.10 Spot cleaning - Compare performance

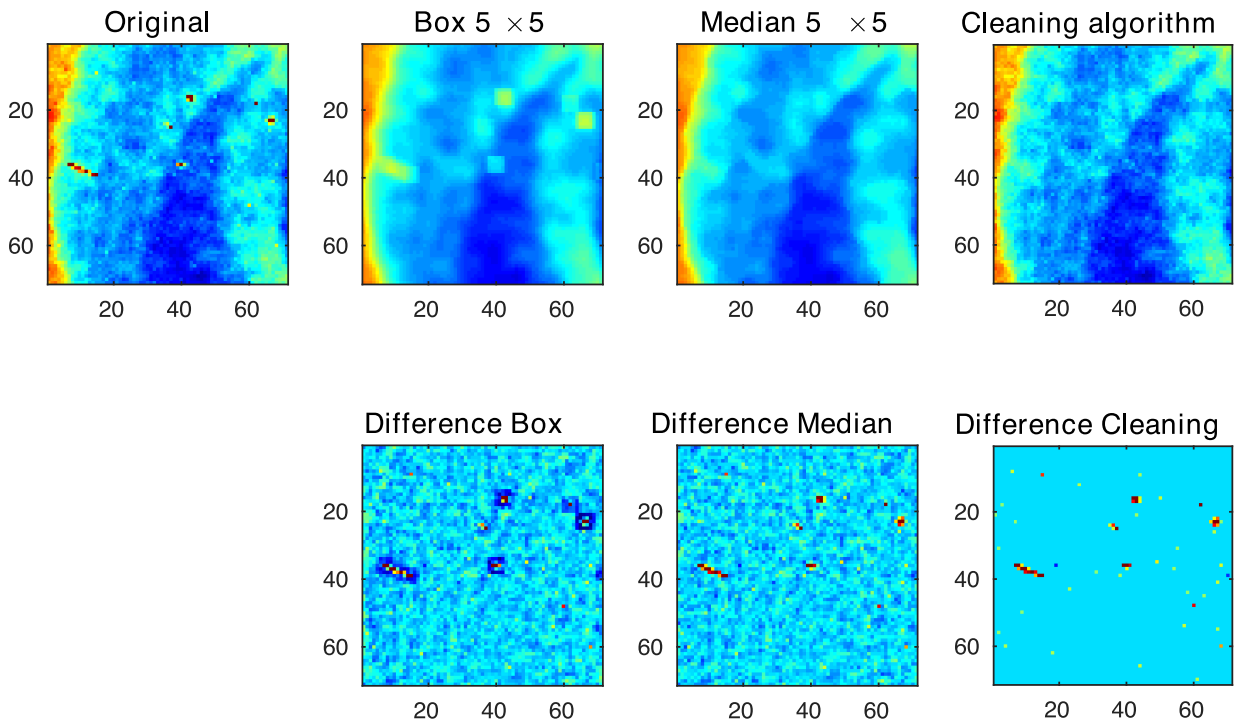


Fig. 5.6: Comparing different filters to remove spots from an image.

5.10.1 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

5.11 High-pass filters

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

5.11.1 Typical high-pass filters:

Gradients

$$\frac{\partial}{\partial x} = \frac{1}{2} \cdot \begin{array}{|c|c|}\hline -1 & 1 \\ \hline \end{array} \qquad \frac{\partial}{\partial x} = \frac{1}{32} \cdot \begin{array}{|c|c|c|}\hline -3 & 0 & 3 \\ \hline -10 & 0 & 10 \\ \hline -3 & 0 & 3 \\ \hline \end{array}$$

Laplacian

$$\triangle = \frac{1}{2} \cdot \begin{array}{|c|c|c|}\hline 1 & 2 & 1 \\ \hline 2 & -12 & 2 \\ \hline 1 & 2 & 1 \\ \hline \end{array}$$

Sobel

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

5.12 Gradient example

5.12.1 Vertical edges

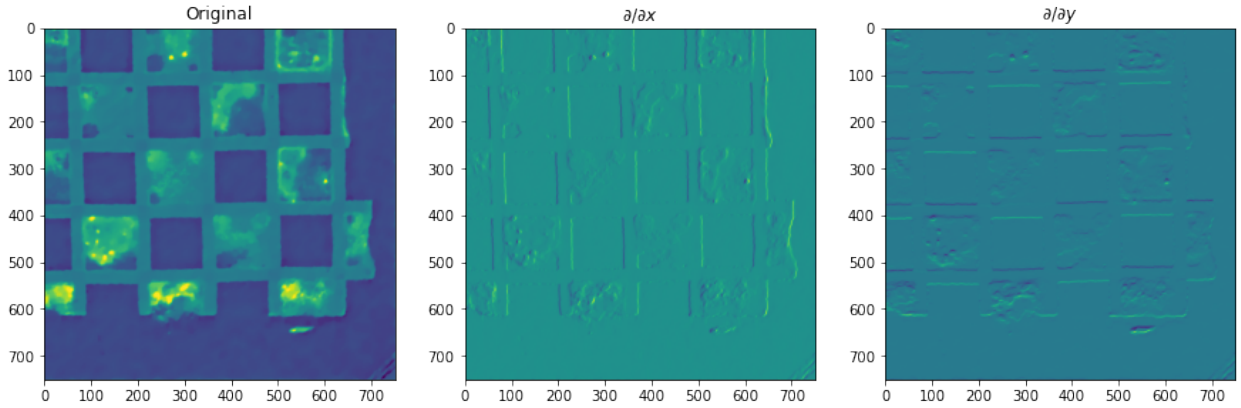
$$\frac{\partial}{\partial x} = \frac{1}{32} \cdot \begin{array}{|c|c|c|}\hline -3 & 0 & 3 \\ \hline -10 & 0 & 10 \\ \hline -3 & 0 & 3 \\ \hline \end{array}$$

5.12.2 Horizontal egdges

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

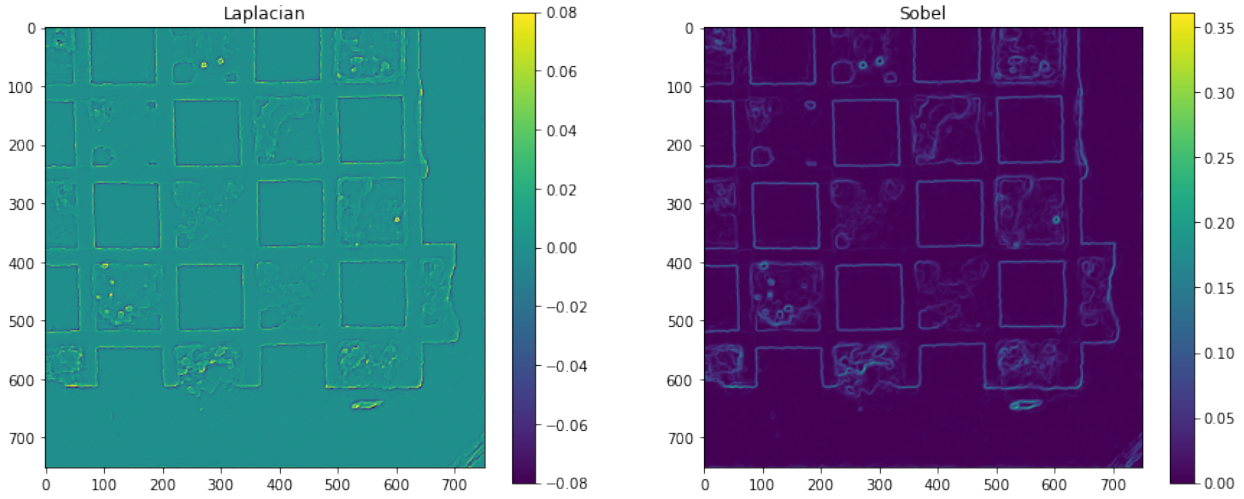
Jaehne, 2005

```
img=plt.imread('figures/orig.png')
k = np.array([[-3,-10,-3],[0,0,0],[3,10,3]]);
plt.figure(figsize=[15,8])
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('$\partial / \
↪partial y$');
```



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();
plt.subplot(1,2,2);plt.imshow(ski.filters.sobel(img)); plt.title('Sobel');  plt.
↪colorbar();
```



5.14 Relevance of filters to machine learning

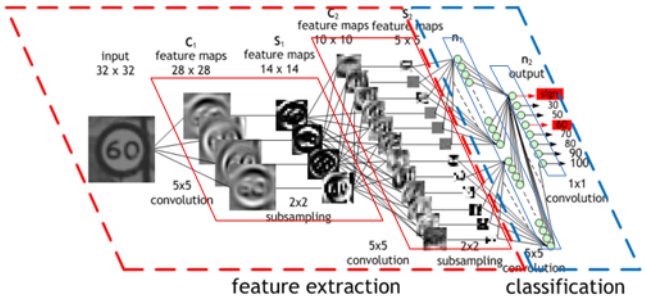


Fig. 5.7: An example of a convolutional neural network for image classification.

NVIDIA Developer zone

CHAPTER

SIX

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.

6.1 Applications of the Fourier transform

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

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

6.2.1 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

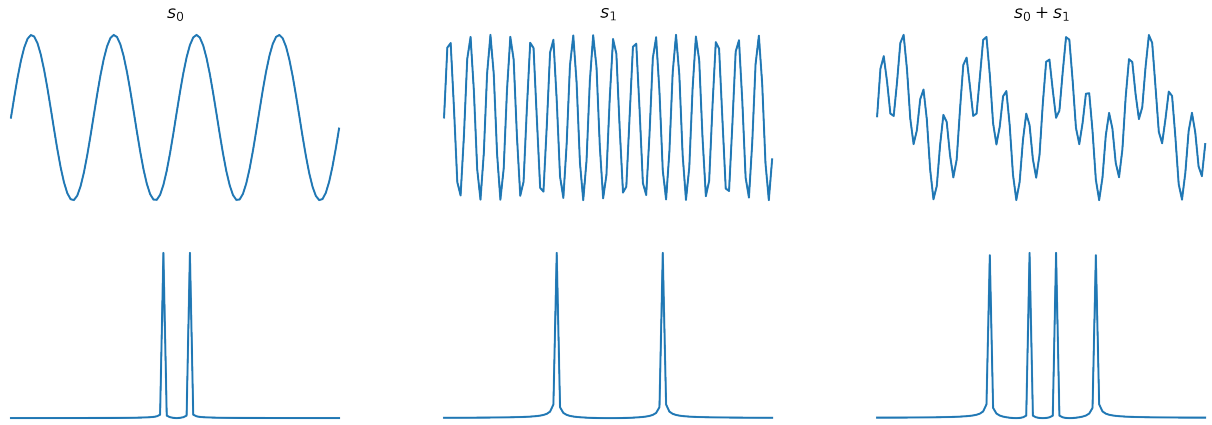
6.3 Some mathematical features of the FT

6.3.1 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');
```



6.3.2 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.

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

6.4 Additive noise in Fourier space

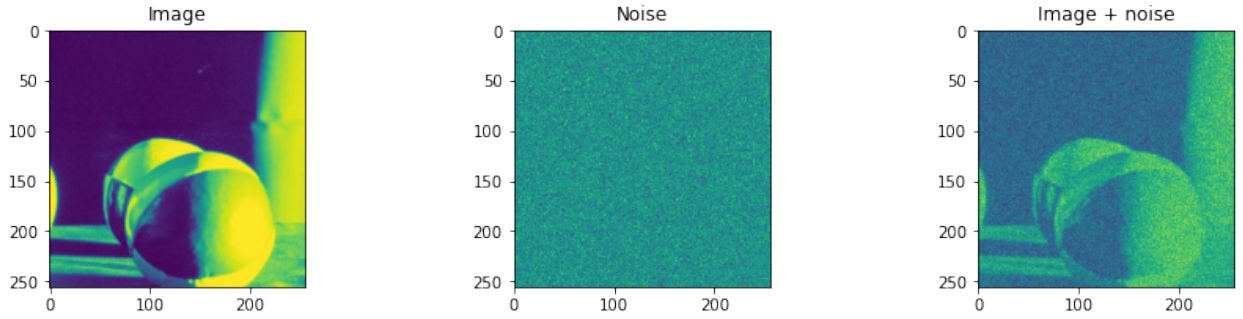
6.4.1 Real space

```
img=plt.imread('figures/bp_ex_original.png'); noise=np.random.normal(0,0.2,size=img.
↪shape); nimg=img+noise;
plt.figure(figsize=[15,3])
plt.subplot(1,3,1); plt.imshow(img);plt.title('Image');
```

(continues on next page)

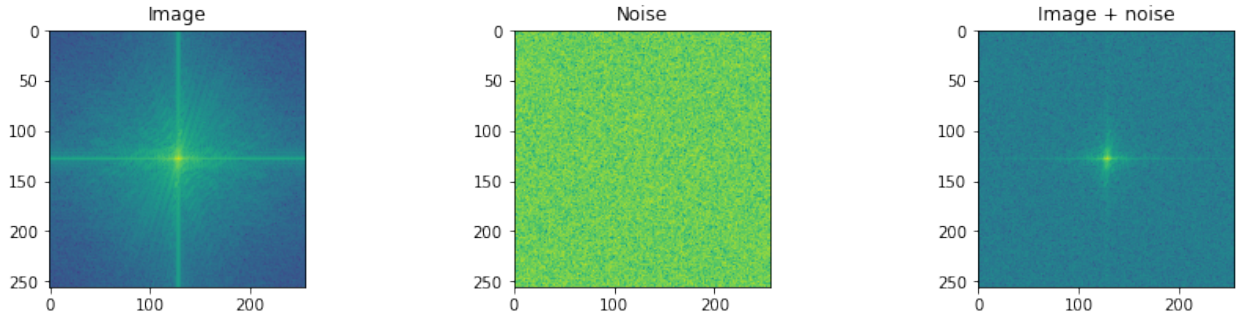
(continued from previous page)

```
plt.subplot(1,3,2); plt.imshow(noise);plt.title('Noise');
plt.subplot(1,3,3); plt.imshow(nimg);plt.title('Image + noise');
```



6.4.2 Fourier space

```
plt.figure(figsize=[15,3])
plt.subplot(1,3,1); plt.imshow(np.log(np.abs(np.fft.fftshift(np.fft.fft2(img)))));plt.
↪title('Image');
plt.subplot(1,3,2); plt.imshow(np.log(np.abs(np.fft.fftshift(np.fft.fft2(noise)))));
↪plt.title('Noise');
plt.subplot(1,3,3); plt.imshow(np.log(np.abs(np.fft.fftshift(np.fft.fft2(nimg)))));
↪plt.title('Image + noise');
```



6.4.3 Problem

How can we suppress noise without destroying relevant image features?

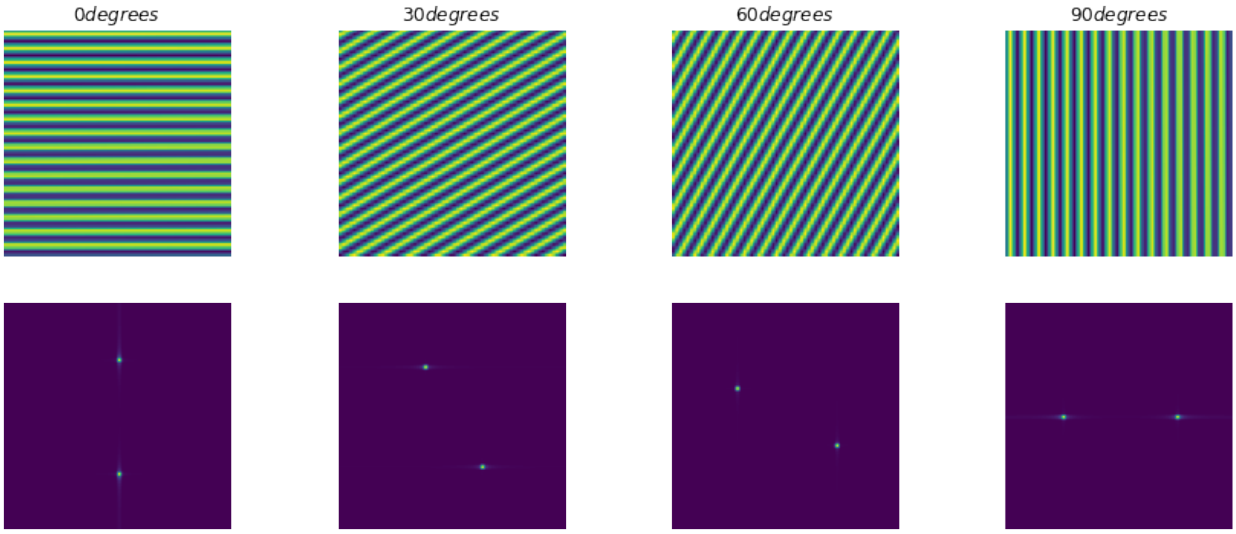
6.5 Spatial frequencies and orientation

```
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()
```

(continues on next page)

(continued from previous page)

```
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('${0} degrees$'.format(angle)); ax[idx].axis(
↪'off');
    ax[idx+4].imshow((np.abs(np.fft.fftshift(np.fft.fft2(d)))));   ax[idx+4].axis('off
↪');
```

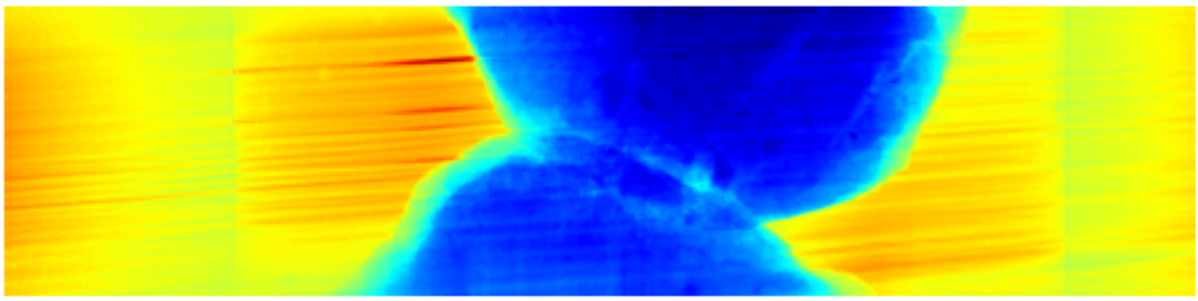


6.6 Example - Stripe removal in Fourier space

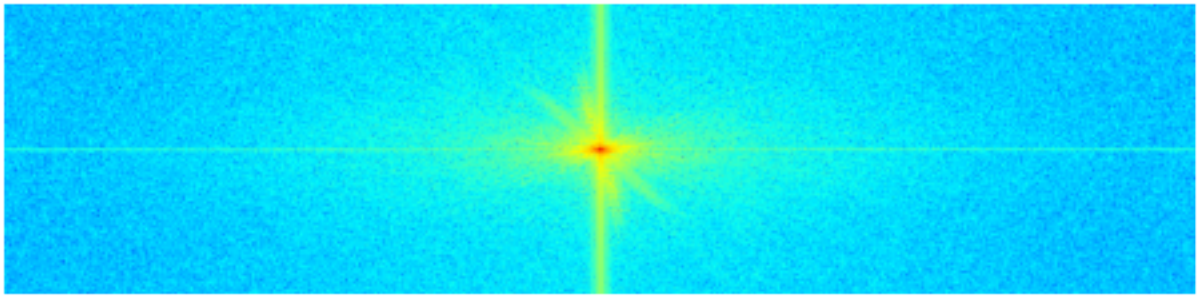
- Transform the image to Fourier space $\mathcal{F}_{\in\mathcal{D}}\{a\} \Rightarrow A$
- Multiply spectrum image by band pass filter $A_{filtered} = A \cdot H$
- Compute the inverse transform to obtain the filtered image in real space $\mathcal{F}_{\in\mathcal{D}}{}^{-\infty}\{A_{filtered}\} \Rightarrow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

```
plt.figure(figsize=[8,10])
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');
```

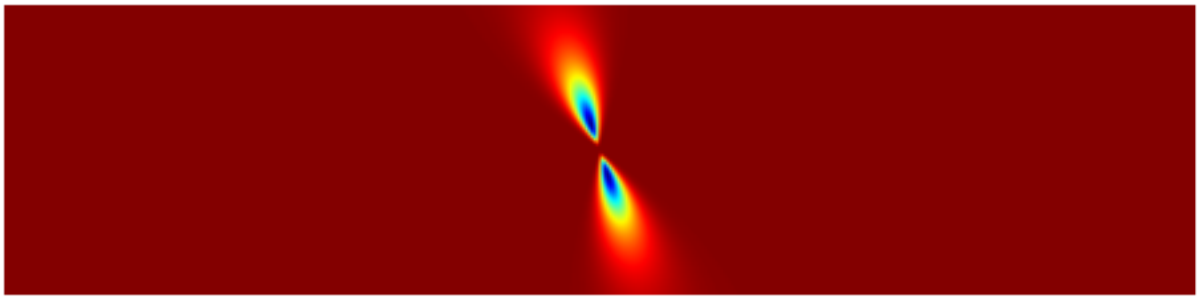
a



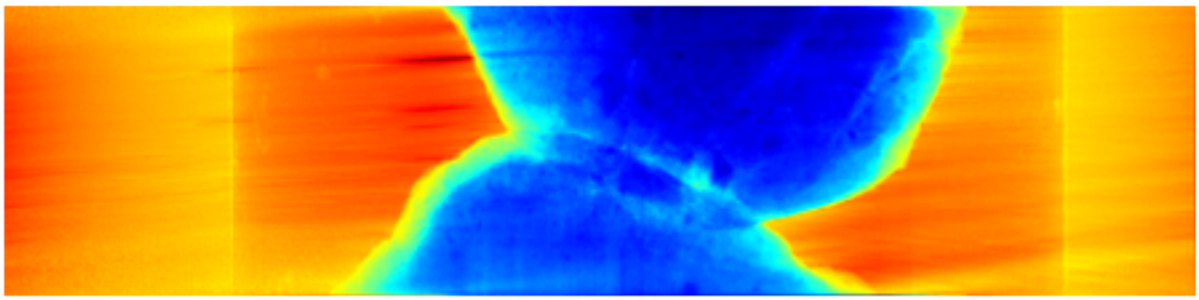
$\mathcal{F}(a)$



Kernel H



$a_{filtered}$



6.7 The effect of the stripe filter

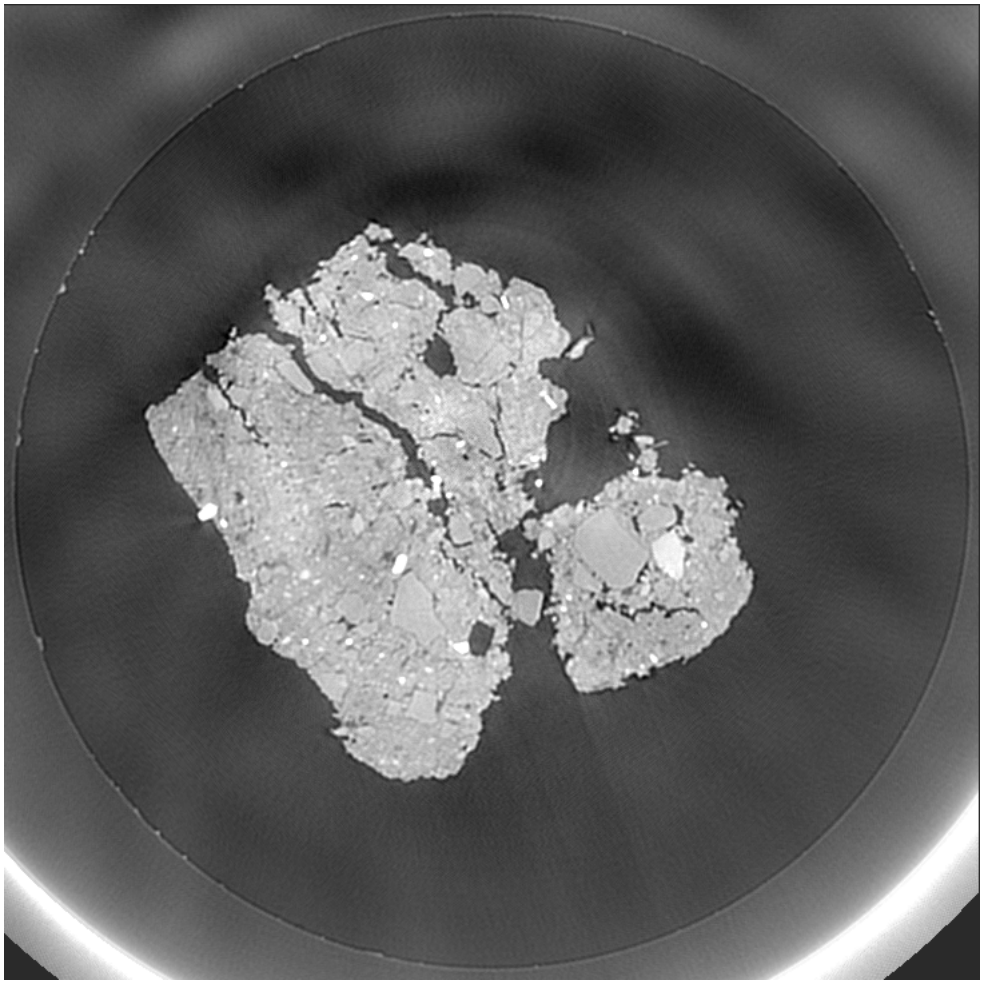


Fig. 6.1: CT slice before stripe removal filter.

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

6.8 Technical details on Fourier space filters

6.8.1 When should you use convolution in Fourier space?

- Simplicity
- Kernel size
- Speed at repeated convolutions

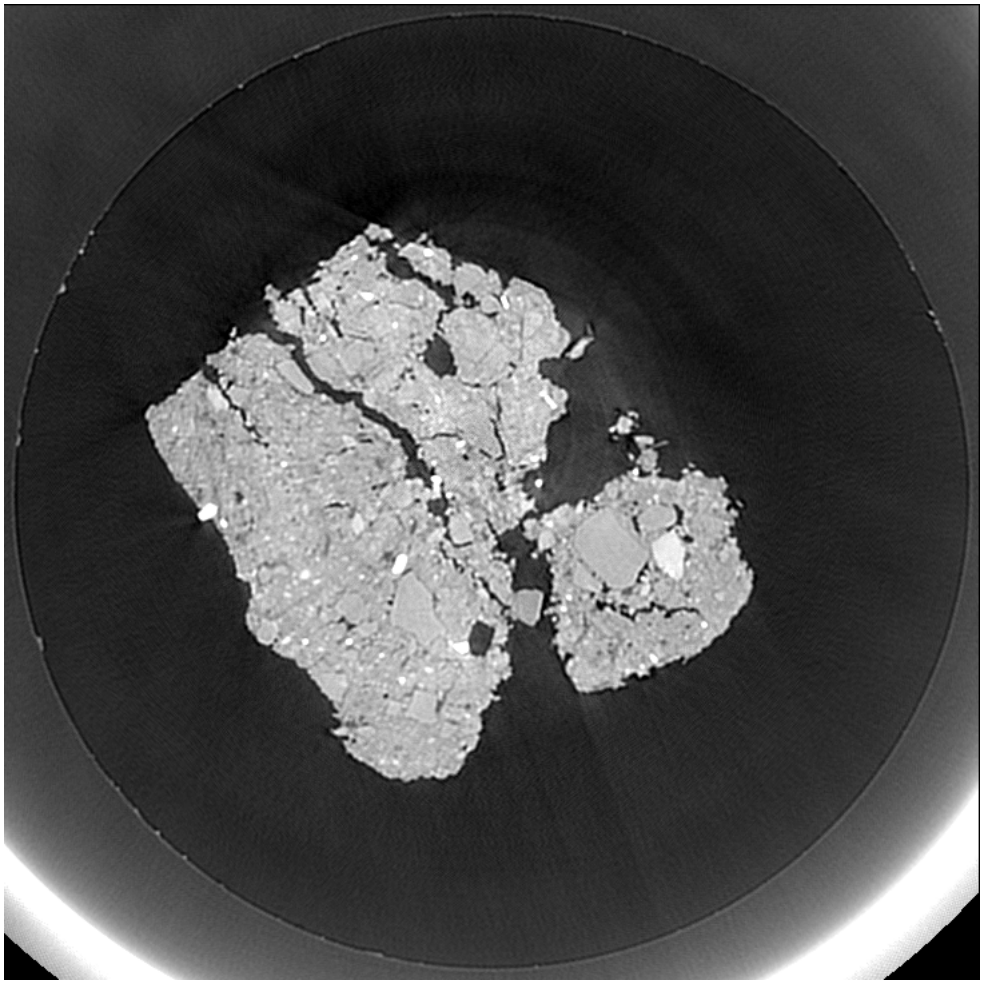


Fig. 6.2: CT slice after applying stripe removal filter.

6.8.2 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.

6.9 Python functions

6.9.1 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

6.9.2 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.

6.9.3 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.

CHAPTER

SEVEN

SCALE SPACES

7.1 Why scale spaces?

7.1.1 Motivation

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

Something new is required. . .

7.1.2 The solution

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

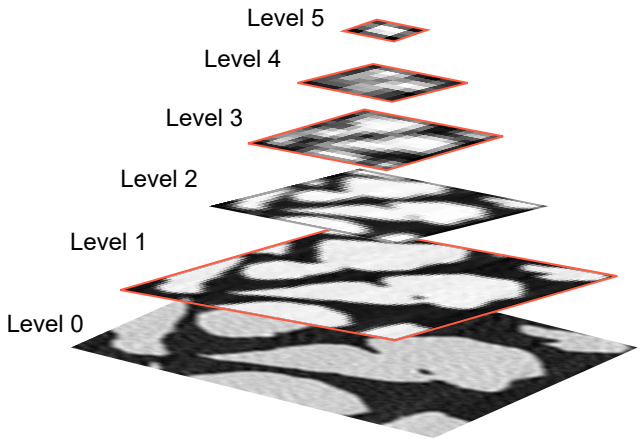


Fig. 7.1: A scale pyramid of an image can be useful for filtering and segmentation.

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

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

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

7.2.1 Applications of wavelets

- Noise reduction
- Analysis
- Segmentation
- Compression

Walker 2008 Mallat 2009

7.3 Wavelet transform of a 1D signal

Using **symlet-4**

7.4 Wavelet transform of an image

7.5 Wavelet transform of an image - example

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.

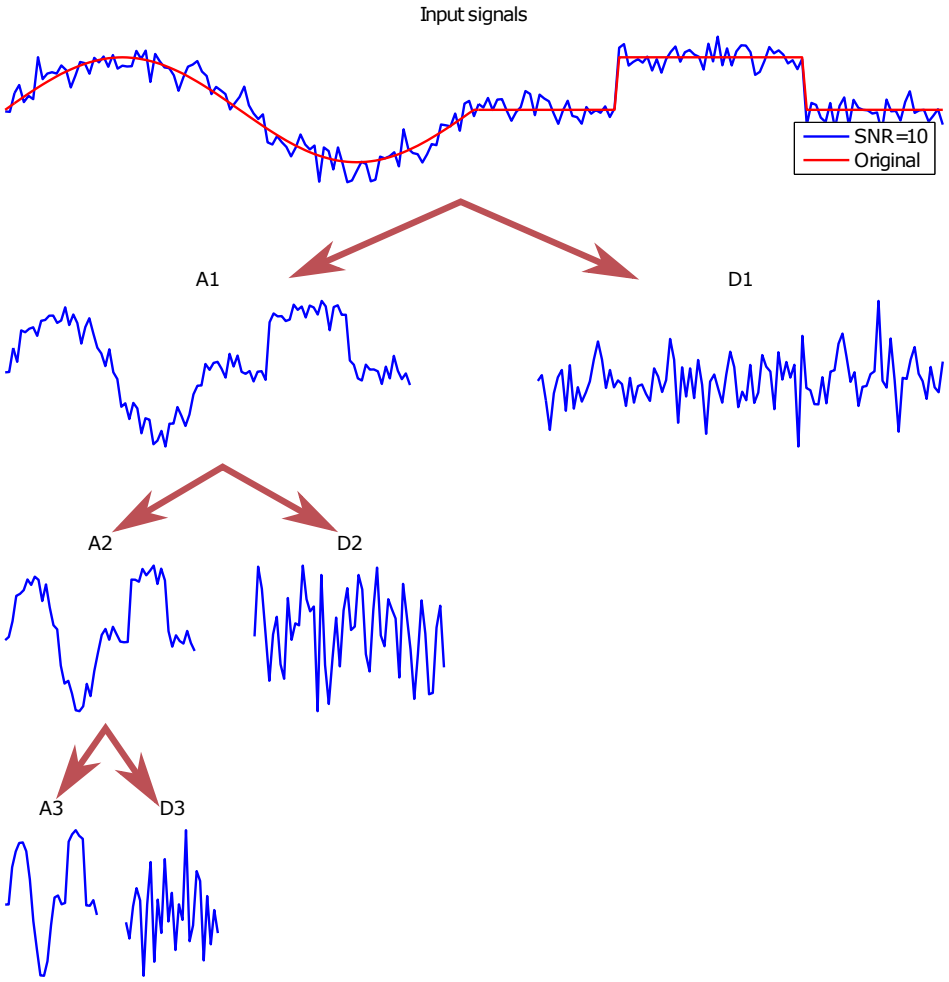


Fig. 7.2: A noisy test signal decomposed using the *symlet-4* wavelet base function.

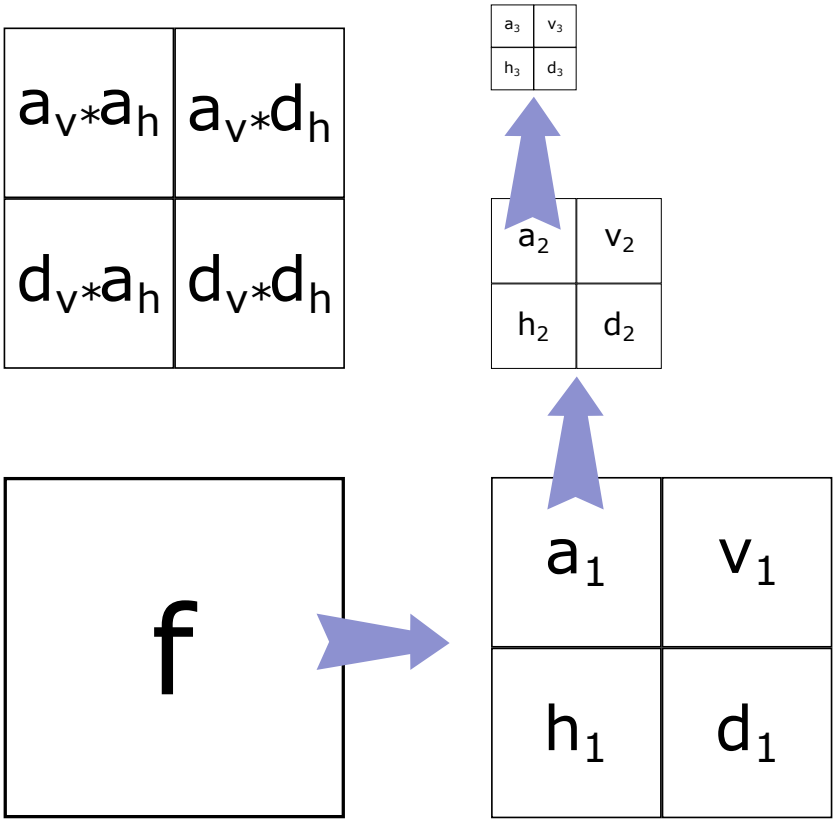


Fig. 7.3: Transform workflow for a 2D wavelet decomposition.

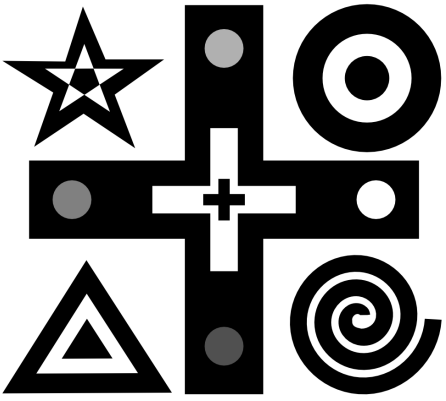


Fig. 7.4: Transform workflow for a 2D wavelet decomposition.

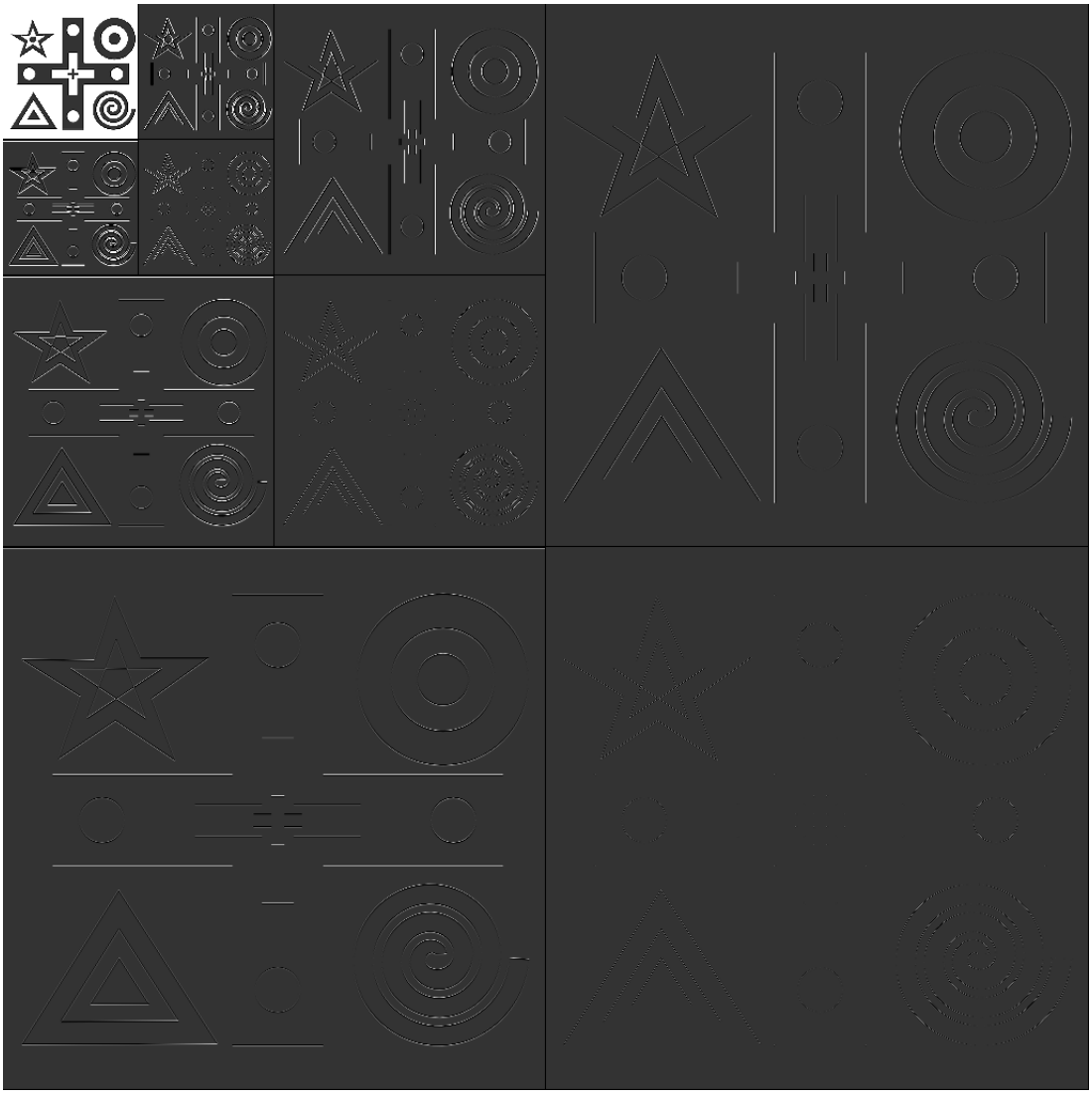


Fig. 7.5: Transform workflow for a 2D wavelet decomposition.

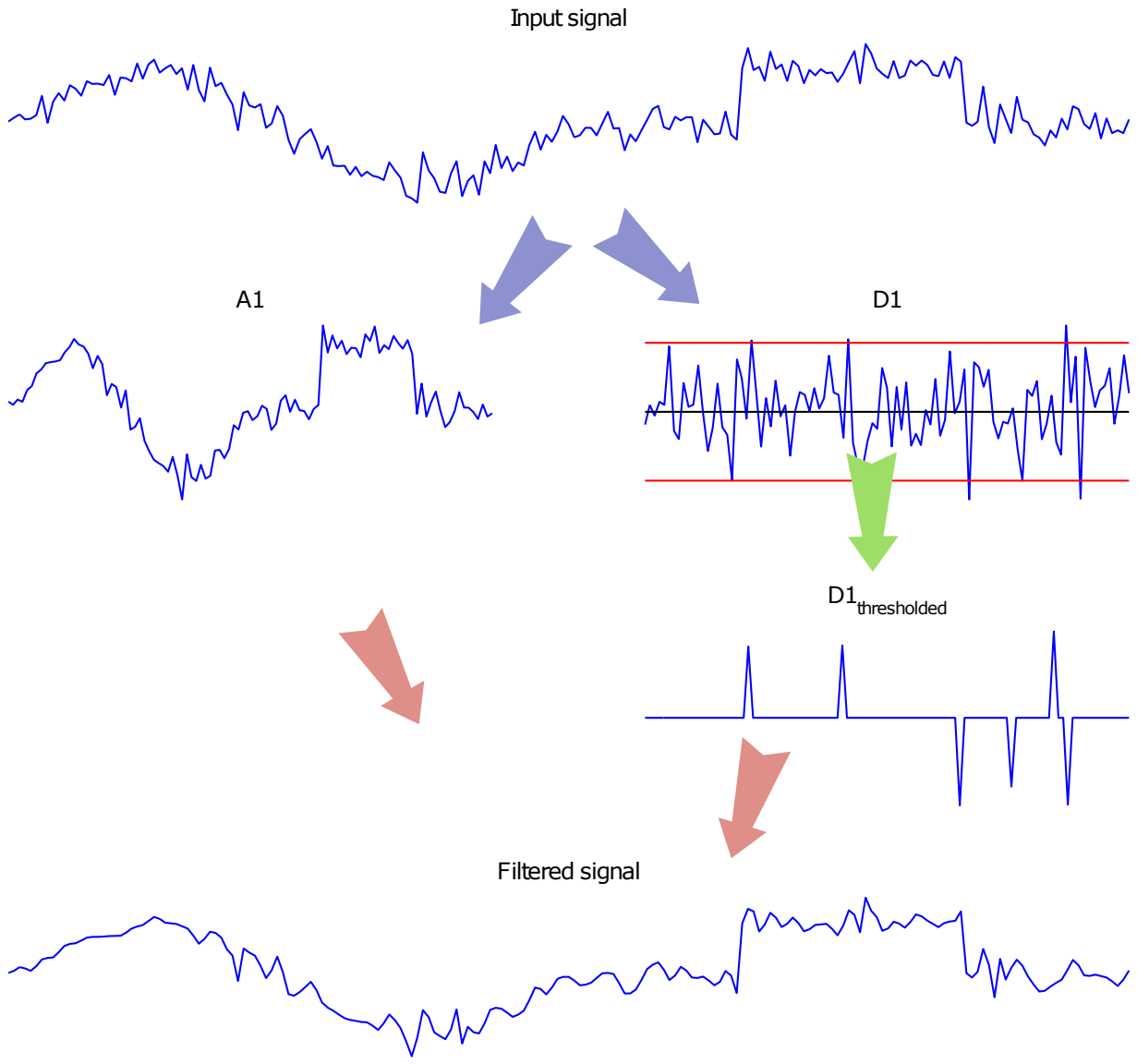


Fig. 7.6: The principle of a basic noise reduction filter using wavelets.

7.7 Wavelet noise reduction - Image example

Example Filtered using two levels of the Symlet-2 wavelet

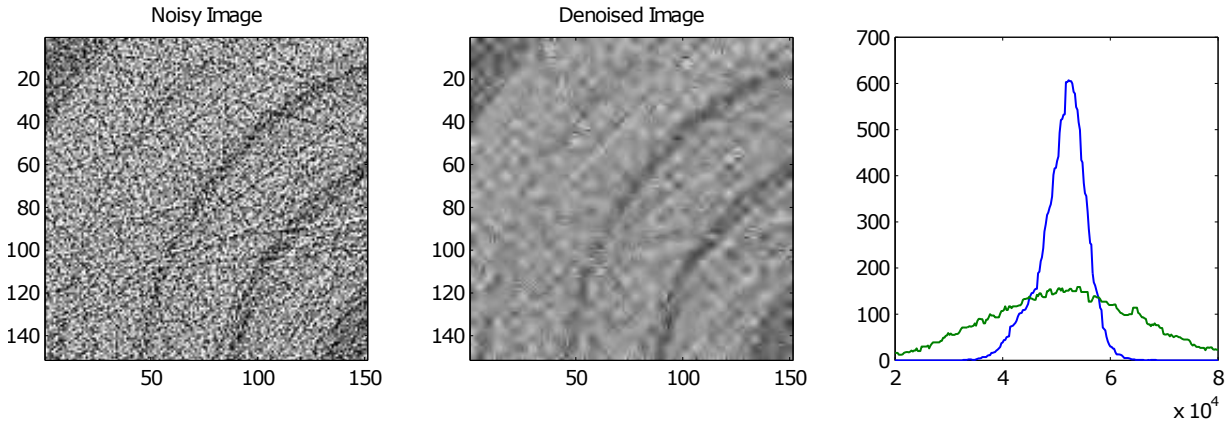


Fig. 7.7: An example of noise reduction using a wavelet filter.

Data: Neutron CT of a lead scroll

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.

CHAPTER

EIGHT

PARAMETERIZED SCALE SPACES

8.1 PDE based scale space filters

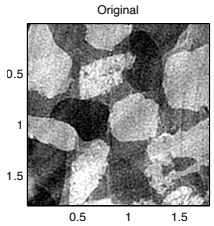


Fig. 8.1: Noisy slice to be filtered.

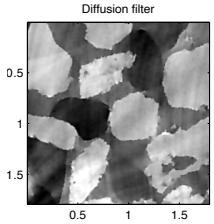


Fig. 8.2: Slice after diffusion filter.

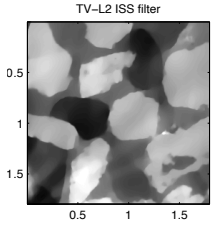


Fig. 8.3: Slice after ISS filter.

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

Aubert 2002.

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

8.3 Controlling the diffusivity

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

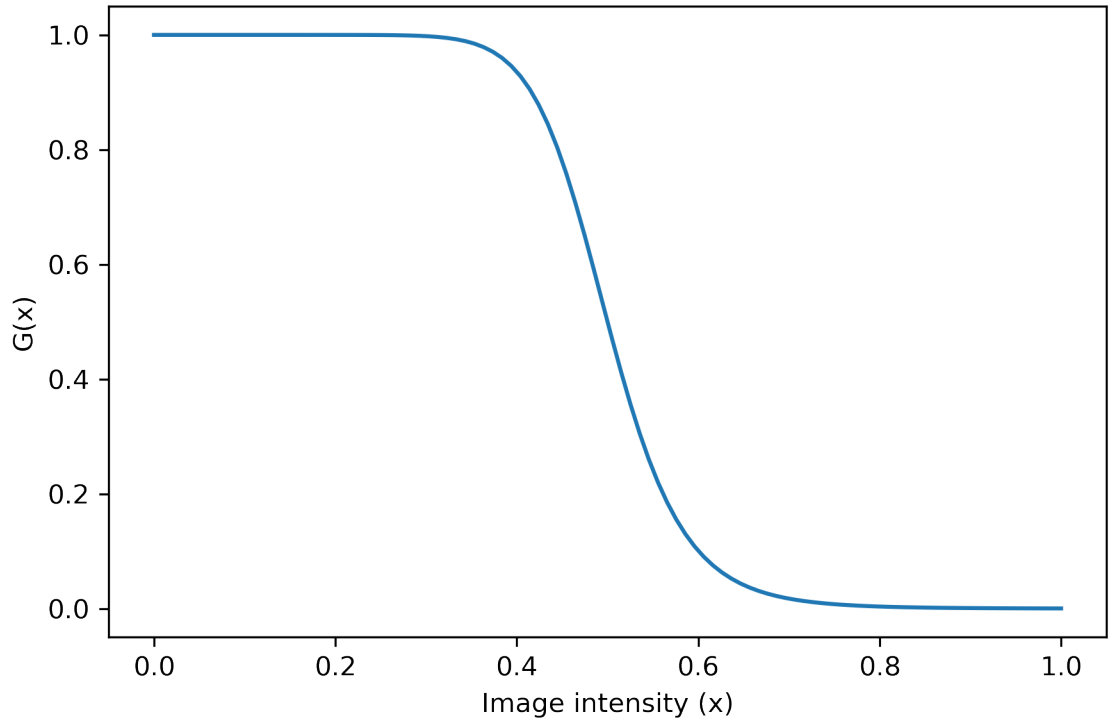
We want to control the diffusion process...

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

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

- λ Threshold level
- n Steepness of the threshold function

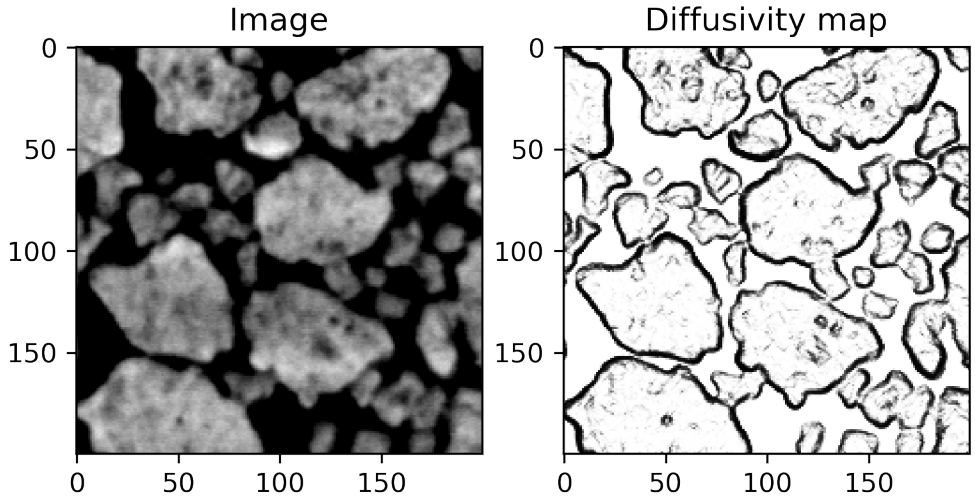
```
x=np.linspace(0,1,100);

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



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

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

8.6 Diffusion filter example

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

8.7 Filtering as a regularization problem

8.7.1 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

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

Rudin-Osher-Fatemi model (ROF)

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

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

8.8 The inverse scale space filter

8.8.1 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

8.8.2 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

8.9 The ROF filter equation

The image f is filtered by solving

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

$\frac{\partial v}{\partial t} = \alpha(f - u)$

8.9.1 Variables:

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

8.9.2 Filter parameters

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

8.10 Filter iterations

Neutron CT of dried lung filtered using 3D ISS filter

8.11 How to choose λ and α

The requirements varies between different data sets.

8.11.1 Initial conditions:

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

8.11.2 Experiment:

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

8.12 Solutions at different times

nini

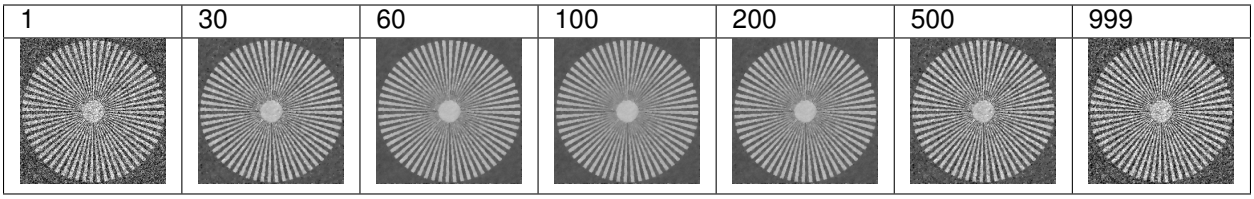


Fig. 8.4: Error plot for different solution times of the ISS filter.

8.13 Solution time

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

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

8.14 The choice of initial image

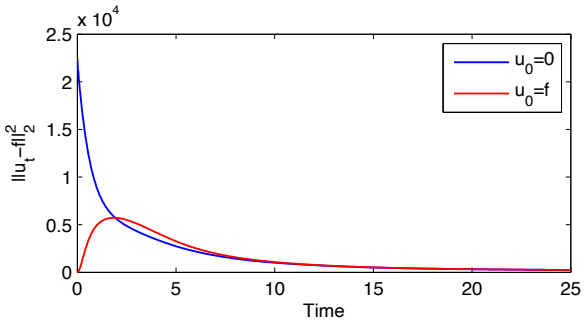


Fig. 8.5: Error plots depend on the choice of the initial image.

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

CHAPTER

NINE

NON-LOCAL MEANS

9.1 Non-local smoothing

9.1.1 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

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

9.3 Non-local means 2D - Example

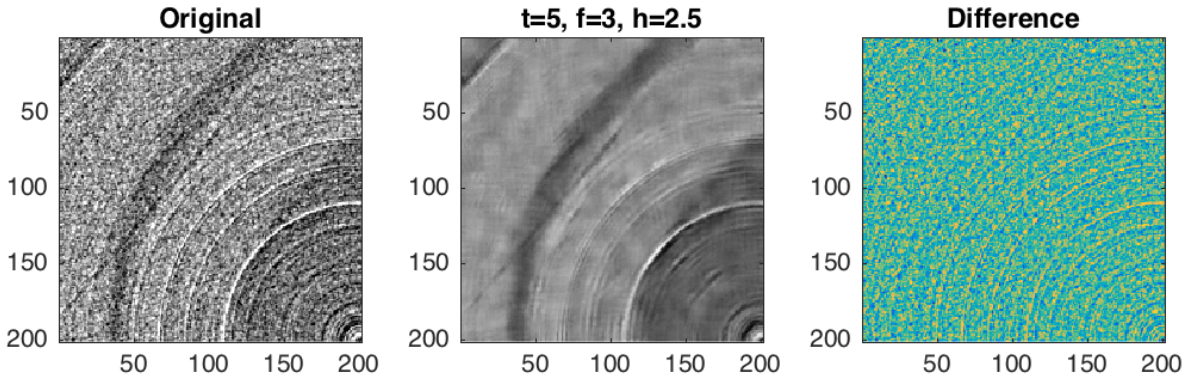


Fig. 9.1: Demonstration on the non-local means filter.

9.3.1 Observations

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

9.4 Performance complications

9.4.1 Problem

The orignal filter compares all pixels with all pixels\ldots\

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

9.4.2 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 $\Omega_i << \Omega$

CHAPTER

TEN

VERIFICATION

10.1 How good is my filter?

10.2 Verify the correctness of the method

10.2.1 “Data massage”

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

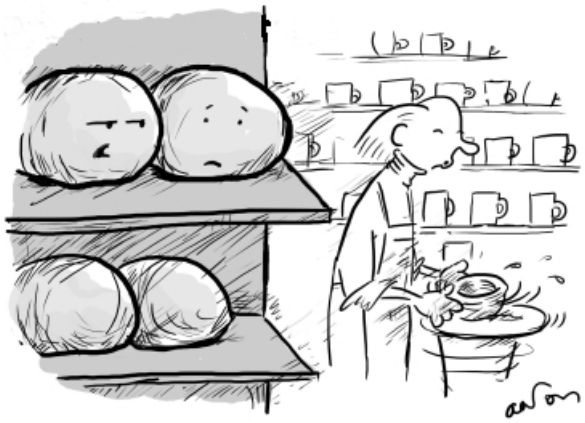


Fig. 10.1: Be careful when you apply different filters. You may make to great modifications.

10.2.2 Verify the validity your method

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

10.3 Verification using difference images

Compute pixel-wise difference between image f and g

Difference images provide first diagnosis about processing performance

10.4 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.

10.4.1 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.

10.5 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. 10.2: The shepp logan phantom.

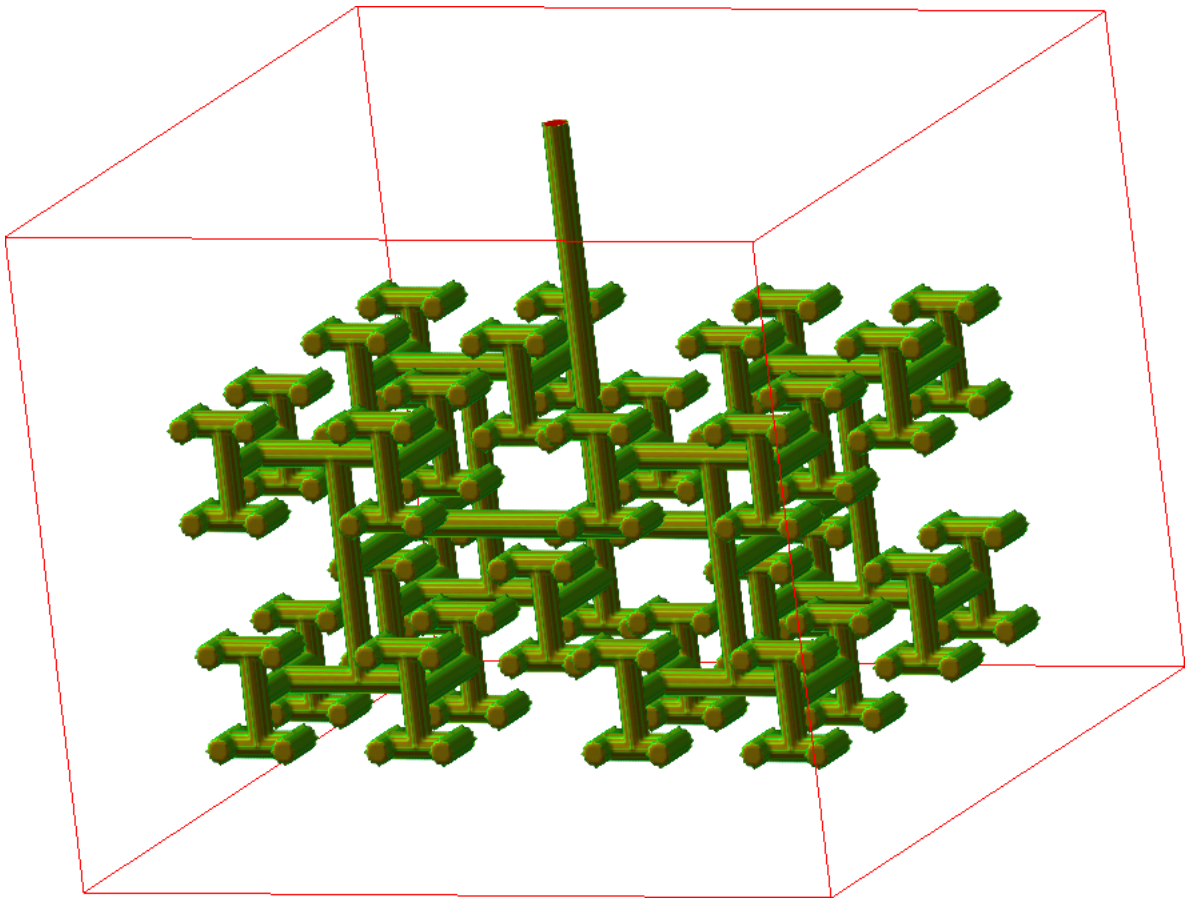


Fig. 10.3: A simulated root network.

10.6 Data for evaluation - Labelled data

Often ‘real’ data

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

Fig. 10.4: Images of hand written numbers from the MNIST data base.

10.7 Evaluation metrics for images

An evaluation procedure need a metric to compare the performance

10.7.1 Mean squared error

10.7.2 Structural similarity index

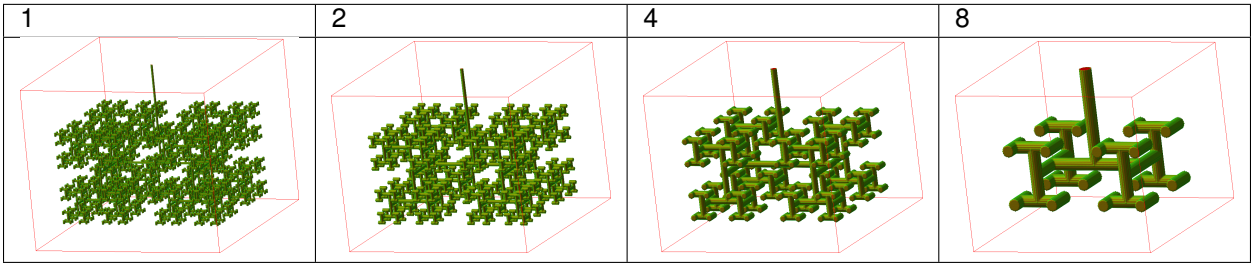
- μ_f, μ_g Local mean of f and g.
- σ_{fg} Local correlation between f and g.
- σ_f, σ_g Local standard deviation of f and g.
- C_1, C_2 Constants based on the image dynamics (small numbers).

Wang 2009

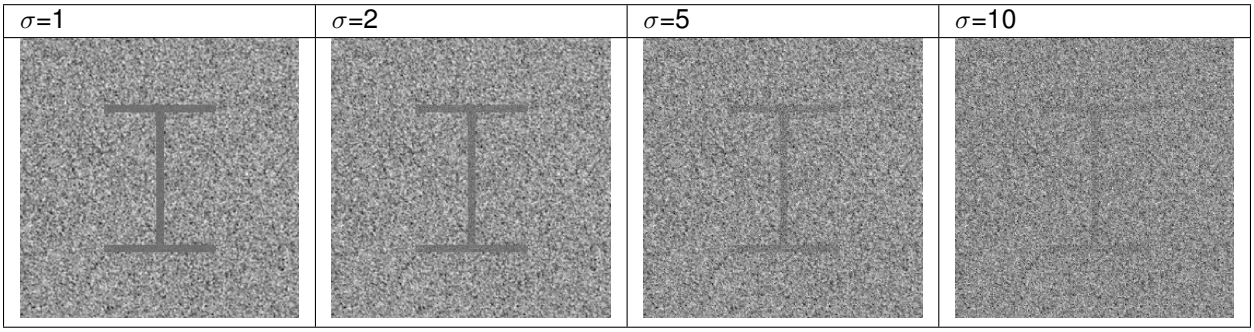
10.8 Test run example

Runnig tests with different structure sizes and SNR.

10.8.1 Phantom structure sizes

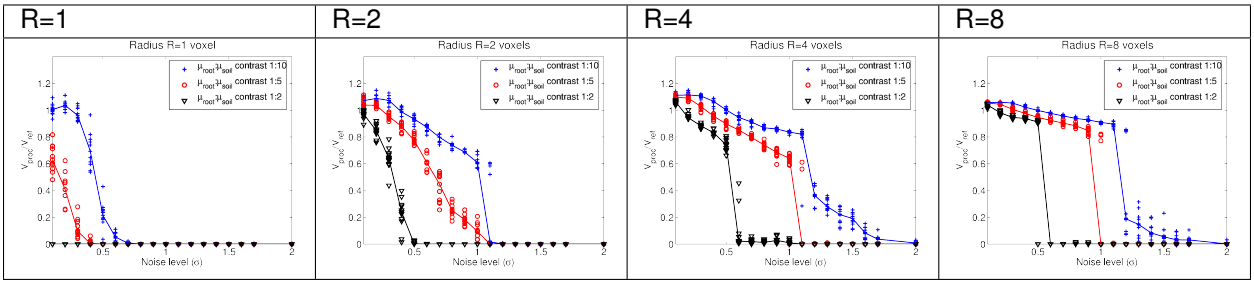


10.8.2 Change SNR and contrast



Process

Plot results



Kaestner et al. 2006

CHAPTER

ELEVEN

OVERVIEW

11.1 Many filters

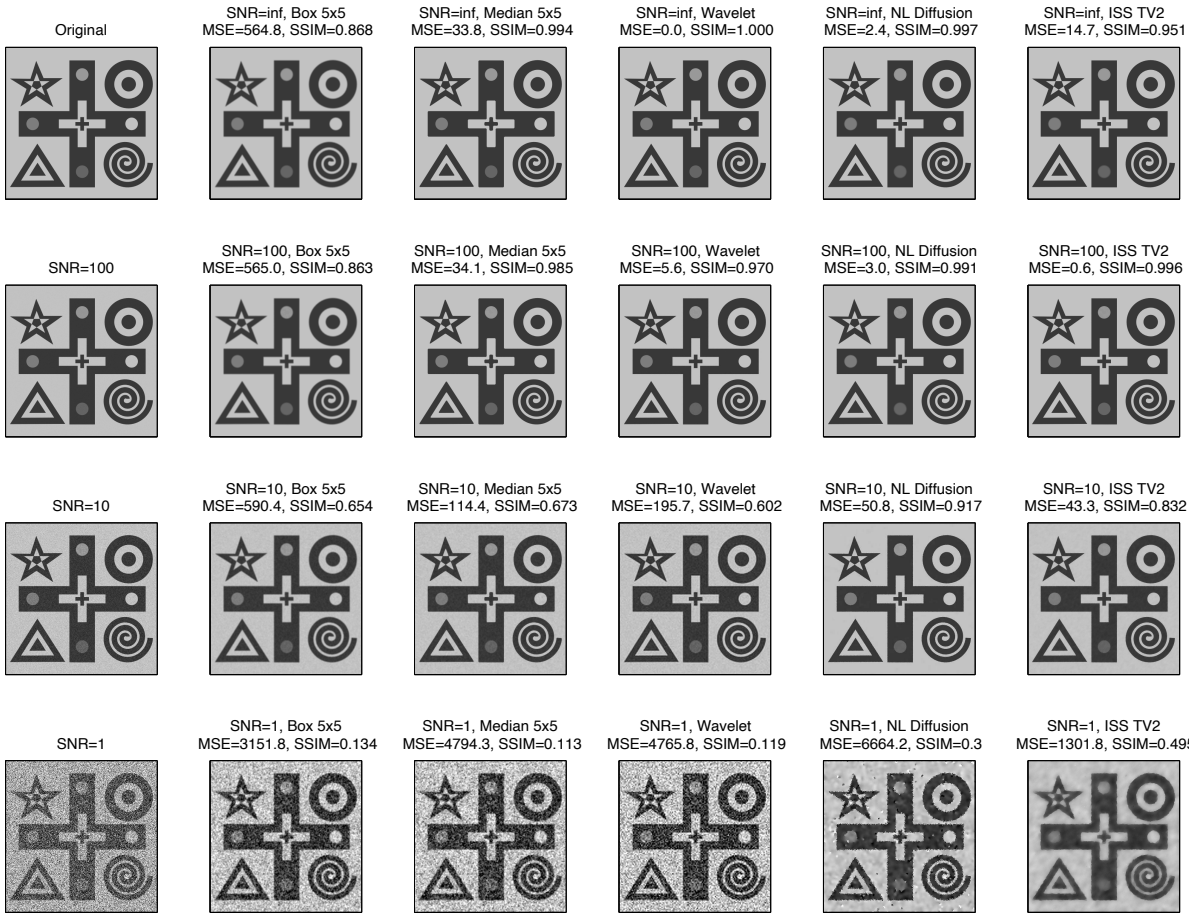


Fig. 11.1: All filters form the lecture for different SNR.

11.2 Details of filter performance

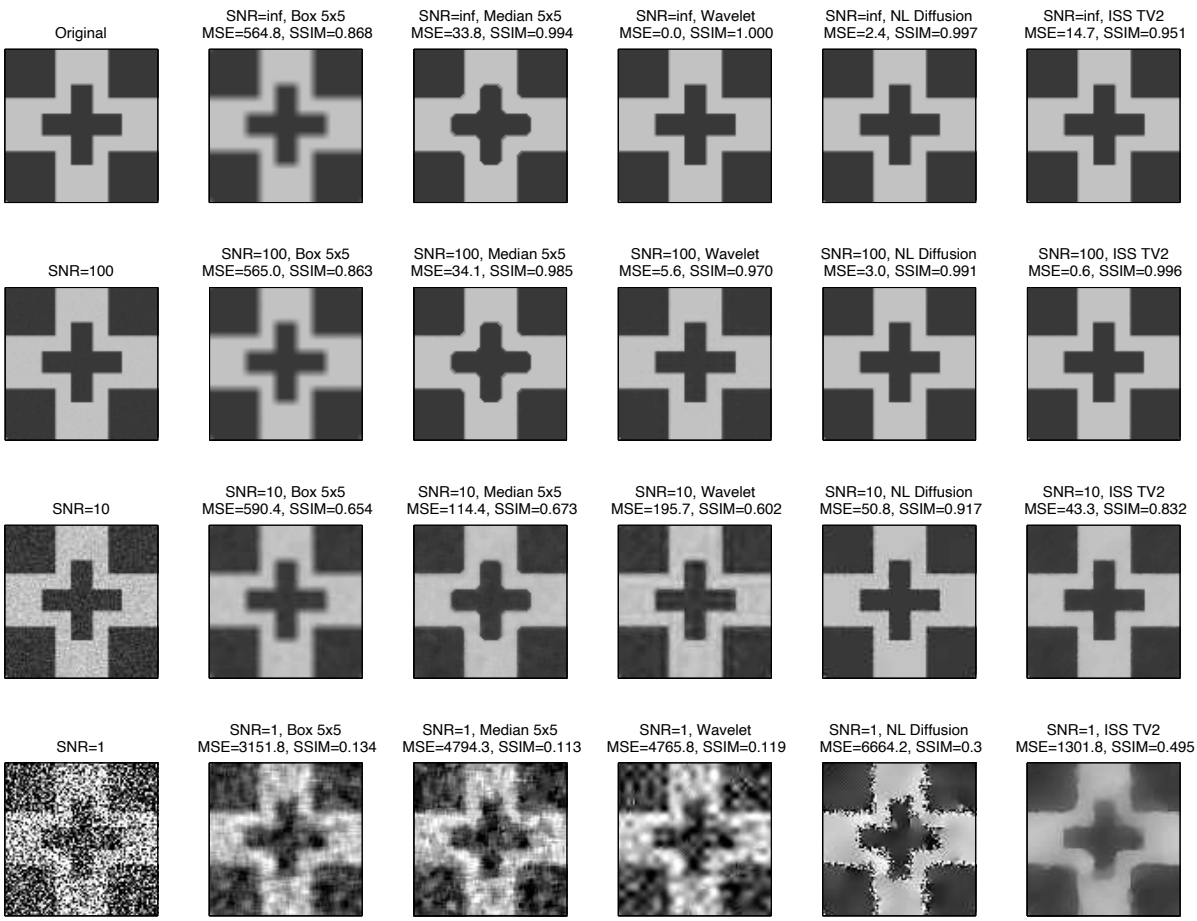


Fig. 11.2: A close-up of all filters form the lecture for different SNR.

11.3 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
- Available time