Quantitative Big Imaging - Dynamic experiments

Anders Kaestner

Apr 16, 2025

CONTENTS

This is the lecture notes fot the 9th lecture of the Quantitative big imaging class given during the spring semester 2021 at ETH Zurich, Switzerland.

0.1 Dynamic Experiments

0.1.1 Load needed modules

```
import matplotlib.pyplot as plt
import seaborn as sns
from   skimage.morphology import label
from   skimage.measure import regionprops
import pandas as pd

from skimage.feature import corner_peaks, corner_harris, BRIEF
from skimage.transform import warp, AffineTransform
from skimage import data
from skimage.io import imread
from scipy.ndimage import distance_transform_edt
from skimage.filters import threshold_otsu

from matplotlib.patches import Rectangle
from matplotlib.patches import ConnectionPatch

import numpy as np
import pydot

%load_ext autoreload
%autoreload 2
plt.rcParams['font.family'] = ['sans-serif']
plt.rcParams['font.sans-serif'] = ['DejaVu Sans']
plt.style.use('default')
sns.set_style("whitegrid", {'axes.grid': False})

%config InlineBackend.figure_format = 'retina'
```

0.1.2 Literature / Useful References

Books

- John C. Russ, “The Image Processing Handbook”,(Boca Raton, CRC Press)
- A. Ardeshir Goshtasby, “Image Registration Principles, Tools and Methods (Springer Verlag)
- B. Jähne,”Spatio-Temporal Image Processing” ,(Springer Verlag)

Papers / Sites

Comparsion of Tracking Methods in Biology

- Chenouard, N., Smal, I., de Chaumont, F., Maška, M., Sbalzarini, I. F., Gong, Y., … Meijering, E. (2014). Objective comparison of particle tracking methods. Nature Methods, 11(3), 281–289. doi
- Maska, M., Ulman, V., Svoboda, D., Matula, P., Matula, P., Ederra, C., … Ortiz-de-Solorzano, C. (2014). A benchmark for comparison of cell tracking algorithms. Bioinformatics (Oxford, England), btu080–. doi

Keypoint and Corner Detection

- Lowe, D.G. Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision 60, 91–110 (2004). doi
- OpenCV SIFT tutorial

Registration

- ITK registration guide

Multiple Hypothesis Testing

- Coraluppi, S. & Carthel, C. Multi-stage multiple-hypothesis tracking. J. Adv. Inf. Fusion 6, 57–67 (2011).
- Chenouard, N., Bloch, I. & Olivo-Marin, J.-C. Multiple hypothesis tracking in microscopy images. in Proc. IEEE Int. Symp. Biomed. Imaging 1346–1349 (IEEE, 2009).

0.1.3 Previously on QBI ...

- Image Enhancment
  - Highlighting the contrast of interest in images
  - Minimizing Noise
- Understanding image histograms
- Automatic Methods
- Component Labeling
- Single Shape Analysis
- Complicated Shapes
- Distribution Analysis
- Uncertainty and Statistics

0.1.4 Quantitative "Big" Imaging

The course has covered imaging enough and there have been a few quantitative metrics, but "big" has not really entered.

What does **big** mean?

- Not just / even large
- Being ready for *big data*
- Scalable, fast, and easy to customize

The three V's

Volume

The production of volume images can already be an obstacle for efficient data management and image analysis.

Variety

Scientific experiments need great variation to allow statisical analysis and hypothesis testing as we saw in last weeks lecture.

Velocity

Today's lecture will approach the problems arising when processes are observed. This means that time series of data is captured and these do by nature increase the amount of data to analyse and manage.

0.1.5 Outline

- Motivation (Why and How?)
- Scientific Goals

Experiments

- Simulations
- Experiment Design

0.2 Imaging of dynamic experiments

0.2.1 Dynamic processes

- Relating to objects in motion
- Characterized by continuous change, activity, or progress

0.2.2 Real time

- The actual time during which a process or event occurs.
- (Computing) of or relating to a system in which
  - input data is processed within milliseconds
  - so that it is available virtually immediately as feedback.

0.2.3 Imaging

Imaging is the

- capture,
- storage,
- manipulation,
- and display

of images.

0.2.4 Experiments

1. What sort of dynamic experiments do we have?
2. How can we design good dynamic experiments?

Fig. 1: Contiuous processes evolve over time without guarantee that it returns to the same point.

Fig. 2: Repetetive processes always behaves the same with a given frequency.

0.2.5 What information are you looking for?

We can say that it looks like, but many pieces of quantitative information are difficult to extract

- How fast is it going?
- How many particles are present?
- Are their sizes constant?
- Are some moving faster?
- Are they rearranging?

0.2.6 Image analysis

How can we…

1. track objects between points?
2. track shape?
3. track distribution?
4. track topology?
5. track voxels?
6. assess deformation and strain?
7. assess more general cases?

→ **How does this help answering your questions?**

0.3 Motivation spatiotemporal analysis

- 3D images are already difficult to interpret on their own
  - Obscured structures
  - Screens only show 2D
- 3D movies (4D) are almost impossible

0.3.1 2D movies can also be challenging

They are 3D!

- x, y, t

Example: a water jet

Example: coffee making

0.3.2 Other image series - 3D Reconstruction

Tomography

One of the most commonly used scans in the hospital is called a computed tomography scan.

This scan works by creating 2D X-ray projections in a number of different directions in order to determine what the 3D volume looks like

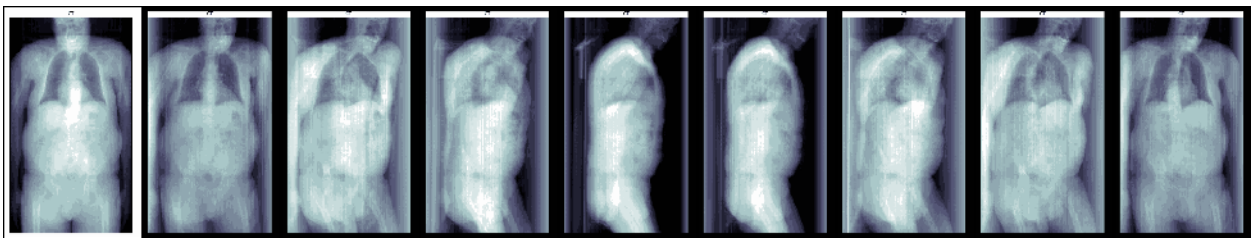


Fig. 1: Some views from a CT scan of a patient

Stage Tilting

Beyond just tracking we can take multiple frames of a still image and instead of looking for changes in the object, we can change the angle.

The pollen image below shows this for SEM

0.3.3 Scientific Goals of dynamic experiments

Rheology

Understanding the flow of liquids and mixtures is important for many processes

- blood movement in arteries, veins, and capillaries
- oil movement through porous rock
- air through dough when baking bread

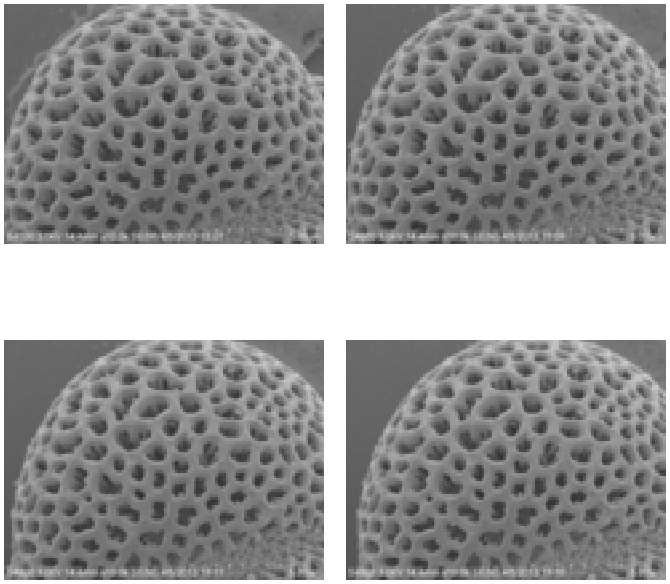


Fig. 2: A tilt series of a pollen grain.

Deformation

Deformation is similarly important since it plays a significant role in the following scenarios

- red blood cell lysis in artificial heart valves
- microfractures growing into stress fractures in bone
- toughening in certain wood types

0.4 Dynamic experiments

The first step of any of these analyses is proper experimental design.

Since there is always:

- A limited field of view
- A voxel size
- A maximum rate of measurements
- Dose limitations
  - Sample damage
  - Limited flux
- A non-zero cost for each measurement

There are always trade-offs to be made between

- getting the best possible high-resolution nanoscale dynamics

- and capturing the system level behavior.

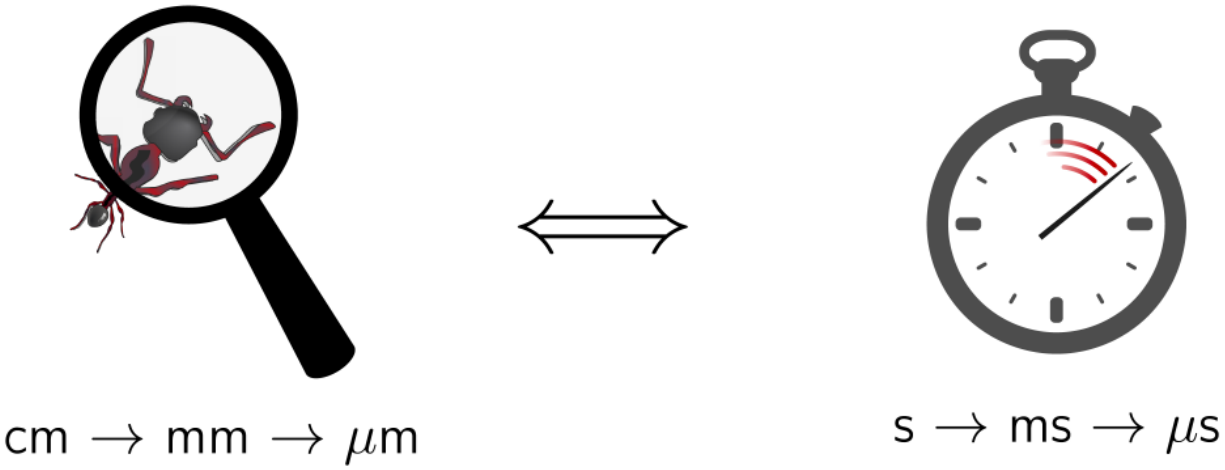


Fig. 1: The trade-off between time and resoultion.

0.4.1 Factors affecting the image quality

- Process speed
- Spatial resolution
- Intensity dynamics

Fig. 2: Parts moving too fast will introduce motion blur.

Example

We know from previous lectures that it is important to at least have a certain signal to noise ratio to be able to perform our analysis reliably. The SNR depends on the dose per pixel which can be controlled by several factors. In the following example we have the experiment conditions for an experiment. Now, if our preliminary assessment of the images tells us that the spatial *or* temporal resolution doesn't reach the reqiurements to allow quantitative analysis, then we have to change the frame rate or the pixel size. This can within some limitations be done without sacrificing the SNR.

You have a dose limited experiment

- 100μm pixels
- 1s Exposure time
- 800 neutrons per pixel

To maintain the SNR you have to

	Smaller pixels	Higher frame-rate
Modification	Longer exposure time	Larger pixels

The reality is however that we have to settle with a compromise that often means a lower SNR. This compromise may then require new thinking for preprocessing and analysis. You may need different filtering and segmentation methods.

0.4.2 Rebinning

Rebinning or using larger pixels can be used to improve the SNR.

Larger pixels can be obtained at experiment time by changing optical settings of the detector system or through binning or resampling of existing images after the experiment. The are pros and cons of each approach

- Setting pixel size at experiment time will optimzie the amount of produced data. Which has an impact on storage and processing if you produce long series. It is not possible to get higher resolution after experiment.
- Resampling has the advantage that it would be possible to go back higher resolution if needed. It will however produce more data the require additional processing.

…but you may miss small details

Fig. 3: Rebinning improve SNR at the cost of small features.

Frame rates

The sample rate needed to study a dynamic process must be chosen to allow several samples during the rise typically 5-20 times the fastest expected time constants in the system.

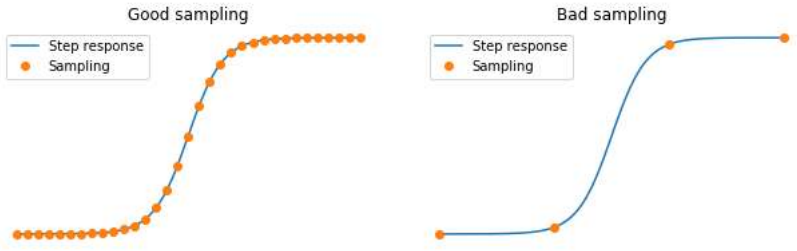


Fig. 4: Selecting the correct frame rate is essential to allow the study of a process.

Some processes can be controlled and even stopped at different levels. This allows measurements of steady intermediate steady state conditions.

Fig. 5: Some processes can be followed by piecewise constant measurements

0.4.3 Planning dynamic experiments

If we measure too fast

- sample damage
- miss out on long term changes
- have noisy data

Too slow

- miss small, rapid changes
- blurring and other motion artifacts

Too high resolution

- too few unique structures in field of view to track

Too low resolution

- not sensitive to small changes

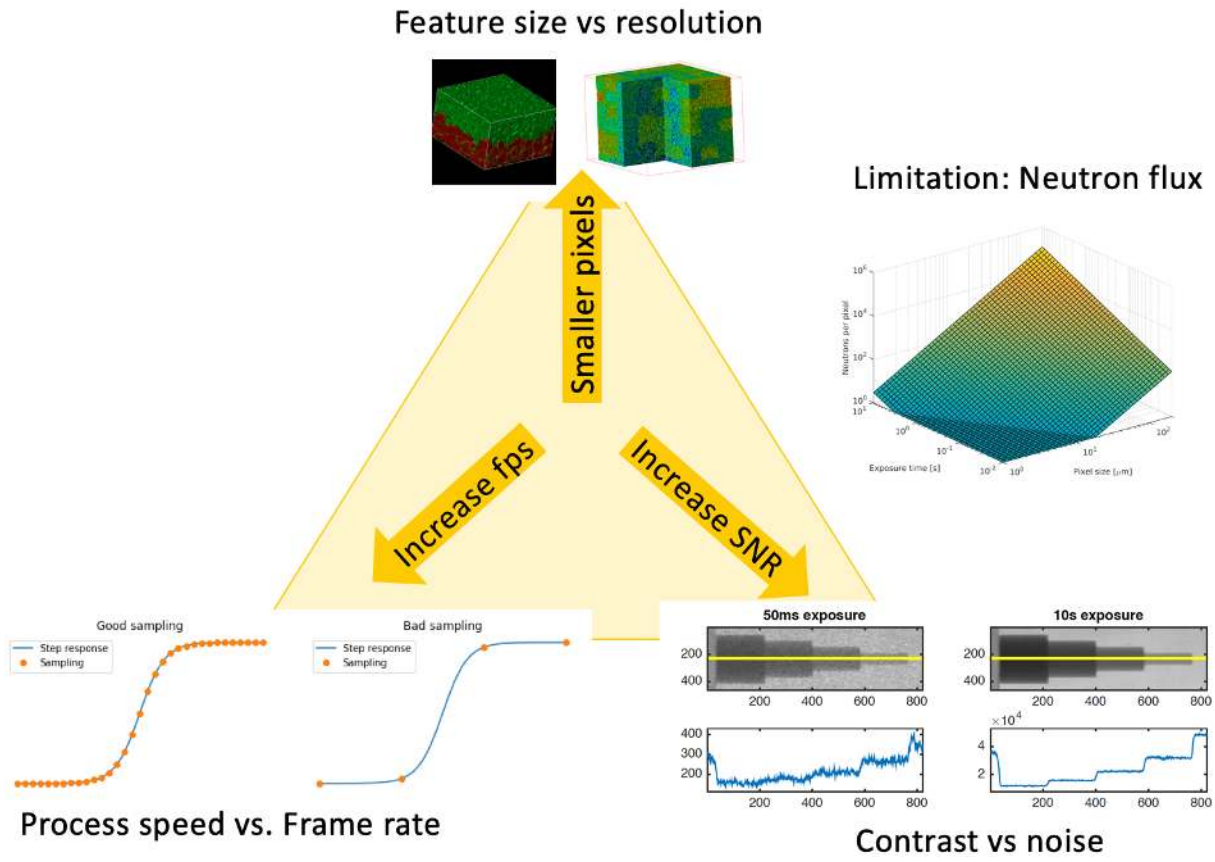


Fig. 6: Factors contributing to the deciding how to perform a dynamic experiment.

Analysing trivial processes

For relatively simple processes you can extract images along the time axes. This will make the analysis easier.

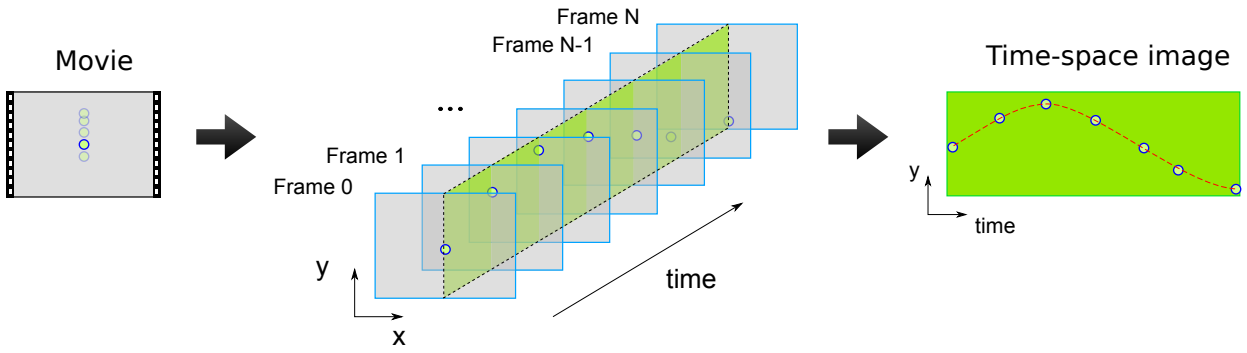


Fig. 7: Extracting a yt-image from a movie.

Example Capillary rise

We want to study capillary rise and Washbourne's equation

Analysis steps

- Extract y-t slices
- Segment the front
- Locate the front as function of time
- Plot as function of $\sqrt{t}$

For the interested: Neutron imaging tutorial - capillary rise

0.4.4 Snow flake tracking

The previous example was very simplified. Real applications are far more complicated with more particles and crossing trajectories.

Let's try to track snow flakes falling from a dark sky (Video by Niezlyziom)

Some initial qualitative observations:

- Direction: The snow flakes fall from the upper left corner to the lower right
- Sizes: There is great variation in size
- Velocity: Small flakes seem to fall slower than larger ones

Data preparation

The first step - extract the frames from the movie

Here we use the ffmpeg application to extract the frames from the mp4-movie clip we want to study. You may have to use a different approach depending on your data source. Sometimes you already have the frames straight from the detector system.

It takes a while to extract the frames, therefor we check if it is already done.

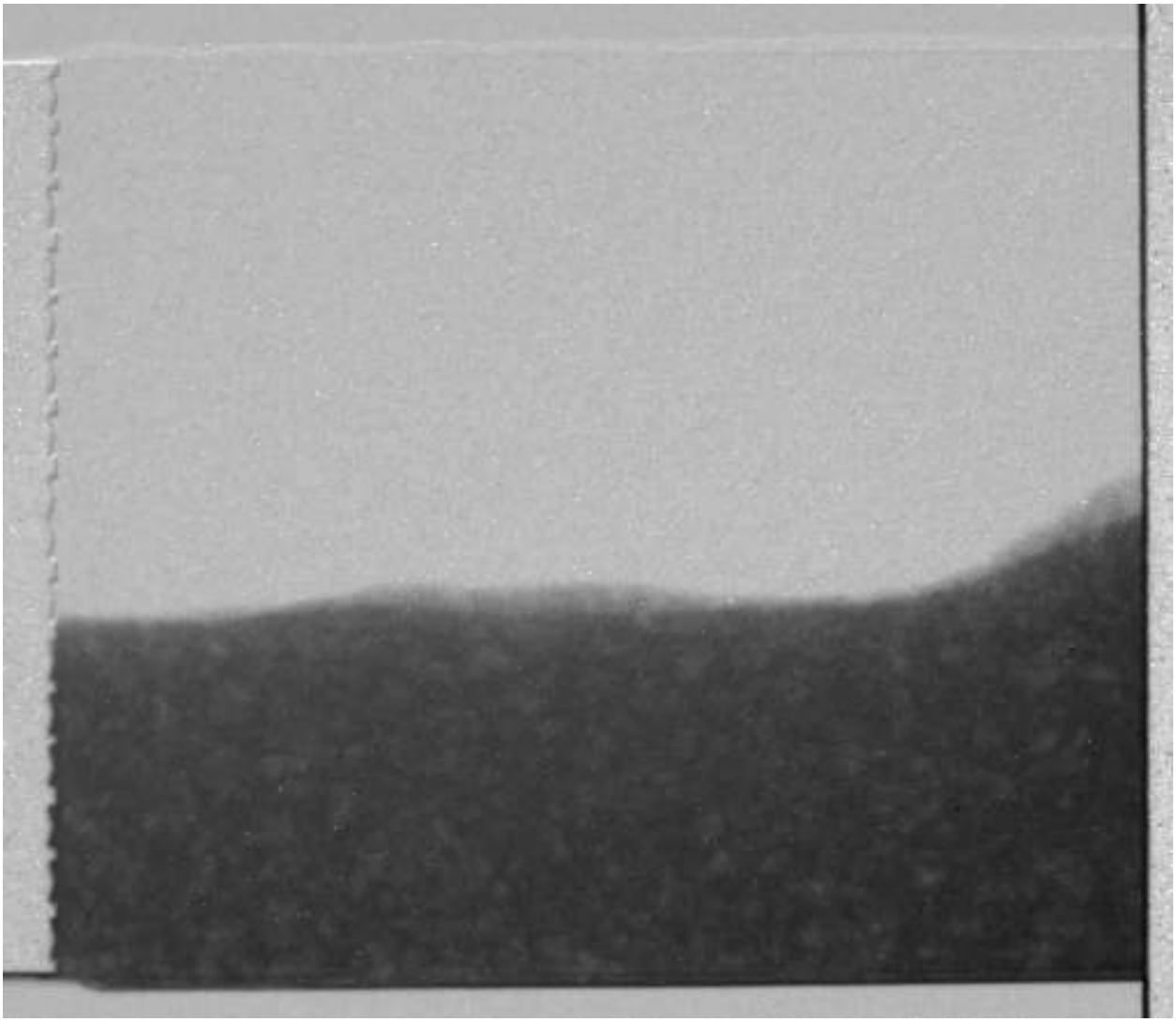


Fig. 8: A frame from the capillary rise movie.

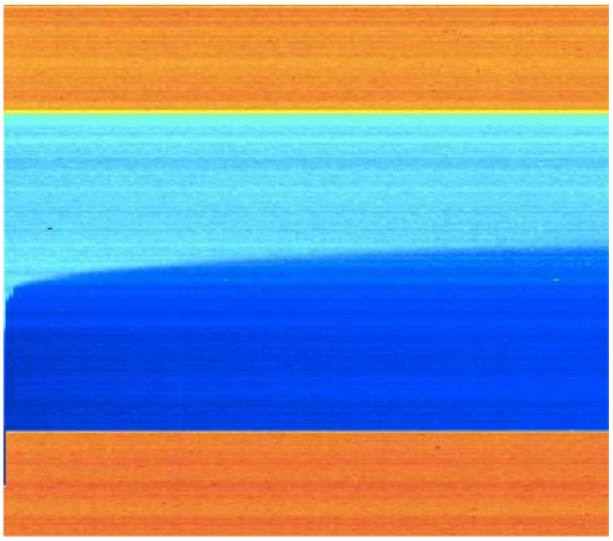


Fig. 9: An extracted yt-slice.

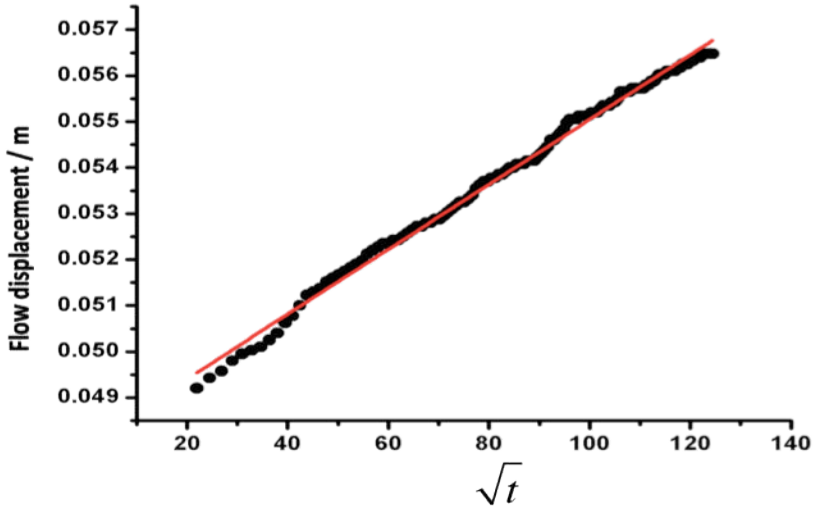
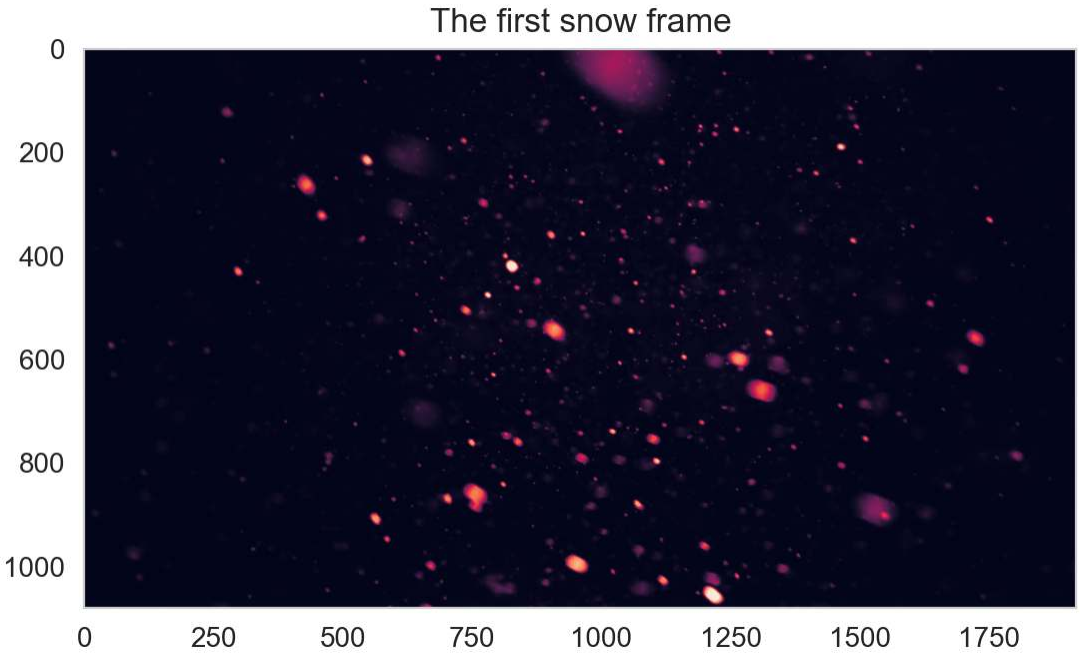


Fig. 10: Front positions follow Washbourne's equation.

Fig. 11: A frame from the snow fall movie.

```
![ ! -d movies/snow ] && mkdir movies/snow && ffmpeg -i movies/snowfall.mp4 -r 25/1 -
 ↪vf format=gray movies/snow/snow_%04d.png

plt.imshow(plt.imread('movies/snow/snow_0001.png'))
plt.title('The first snow frame');
```



Load the frames

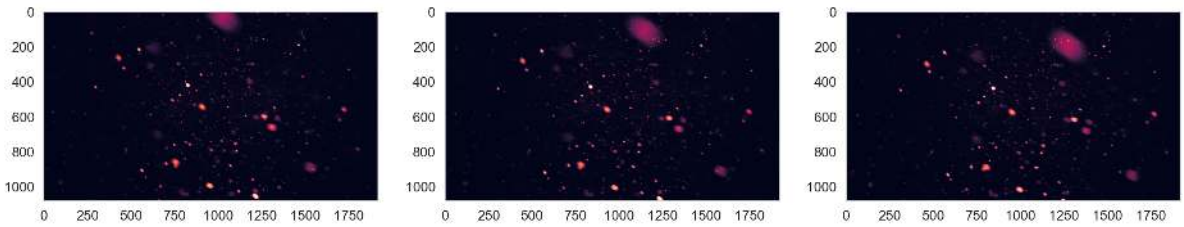
We now load the frames into a numpy array for the first analysis

```
def read_series(fmask,first,last) :
    tmp = plt.imread(fmask.format(first))

    img=np.zeros([last-first+1,tmp.shape[0],tmp.shape[1]])
    img[0]=tmp

    for idx in range(first+1,last+1) :
        img[idx-first]=plt.imread(fmask.format(idx))

    return img.astype(float)
```

```
snow = read_series('movies/snow/snow_{0:04d}.png',first=1, last=300)
```

Here, we see the first three frames from the movie.

```
fig,ax = plt.subplots(1,3,figsize=(15,3))

ax[0].imshow(snow[0])
ax[1].imshow(snow[1])
ax[2].imshow(snow[2]);
```



Observe the movie as a volume

First steps using the 3D data

Let's try projecting the time-frames on the XY-plane

```
plt.imshow(snow.max(axis=0));
```

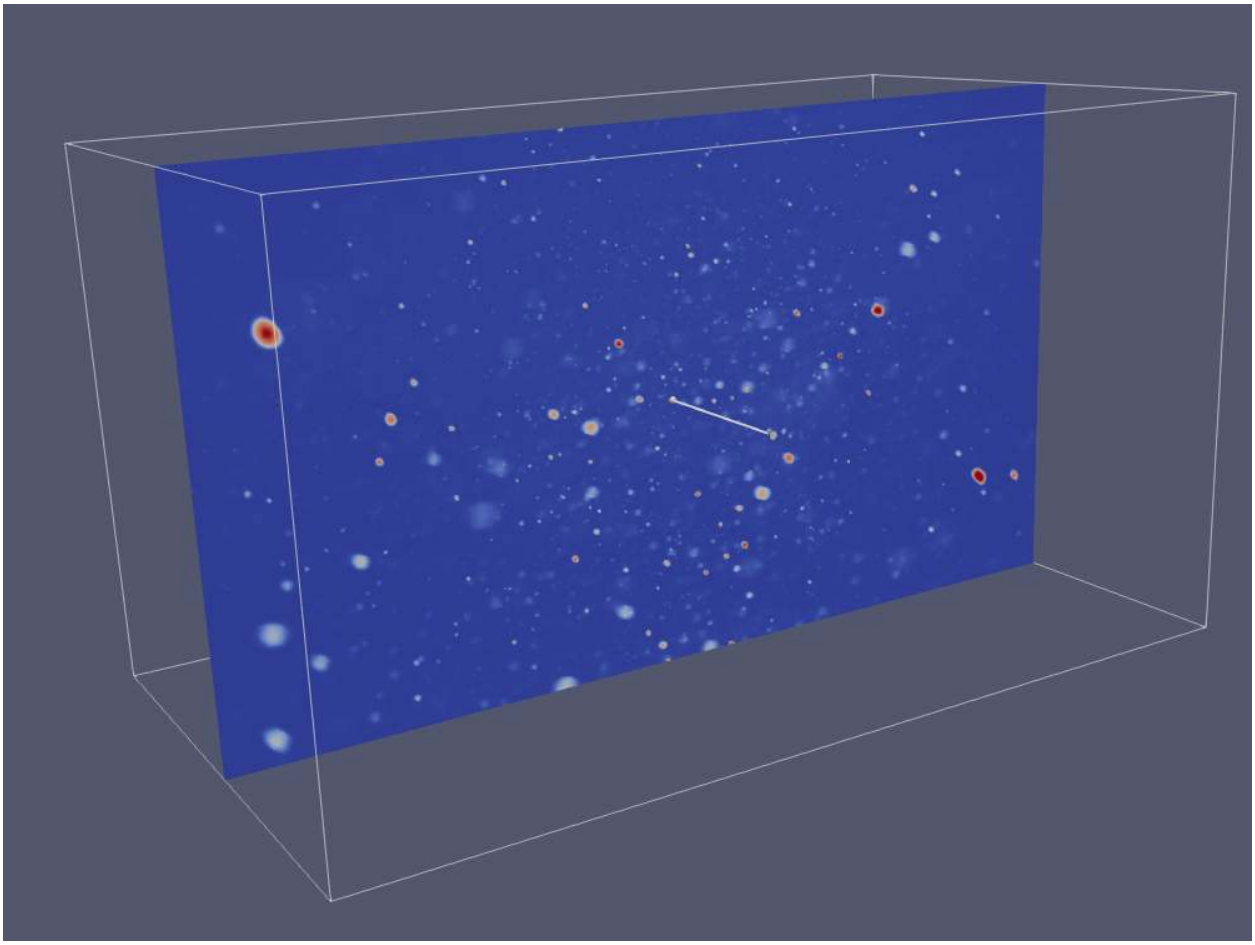
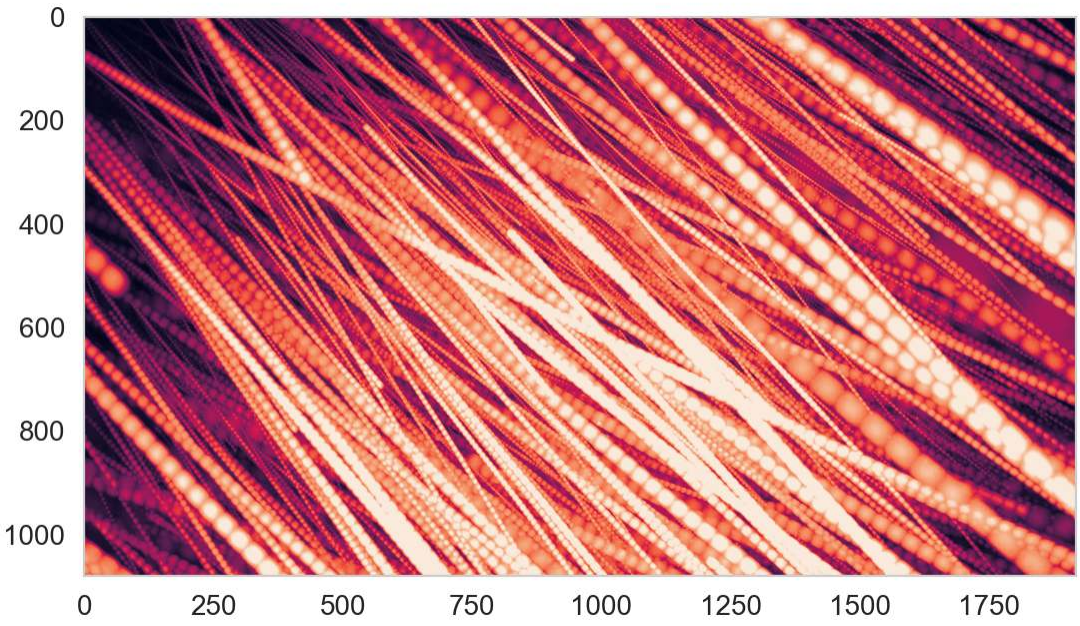


Fig. 12: A frame from the snow fall movie in the 3D frame.



Some more observations:

- Each snow flake draws a track
- There is an intensity gradient along the track
- The diagonal pattern is confirmed
- The frame rate is on the lower end.

It is however hard to track individual flakes in this image

Depending on the purpose of the analysis, we might already be satisfied with the track pattern produced by the snow flakes. These give an average directions of the snow fall.

The fact that the frame rate is on the lower end can be seen in the dotted track lines. This means that the time between two frames allowed the snow flakes to move further away than their extent and that there is no clear overlap.

A full 3D view of the snow fall

In 3D, we are able to better track the individual snow flakes.

Here, they appear as tubes in the 3D space, which allows to use volume segmentation approaches for the flake tracking. There is still the problem of crossing tracks which needs to be tackled.

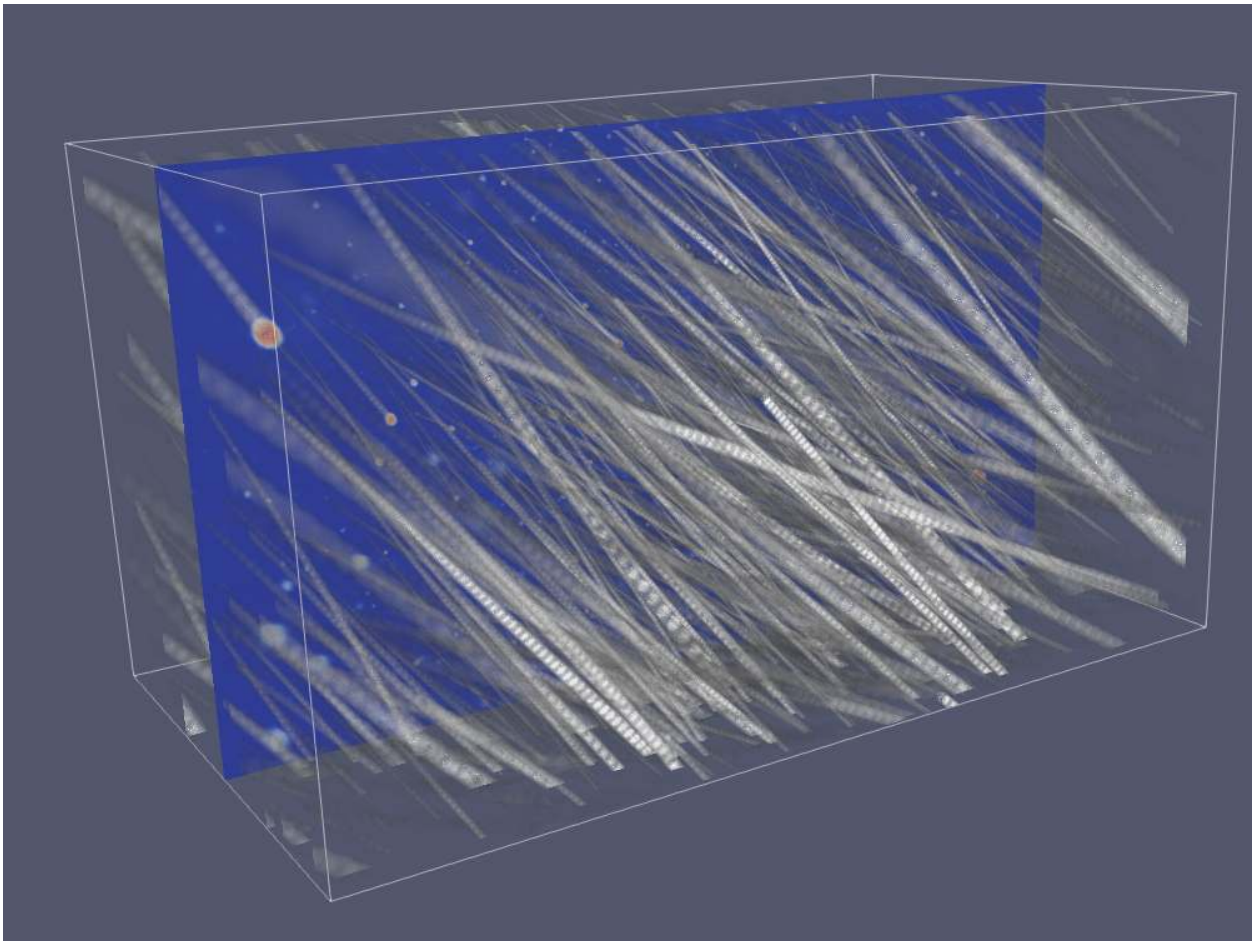


Fig. 13: A frame from the snow fall movie in the 3D frame with the track history added.

0.4.5 Tuning experiment by simulation

In many cases:

- experimental data or setup is inherited
- little can be done about the design,

When there is still the opportunity to tune the experiment

simulations provide a powerful tool for tuning and balancing a large number parameters

Validation

Simulations also provide the ability to pair post-processing to the experiments and determine the limits of tracking.

0.5 Tracking objects and changes

0.5.1 What do we start with?

Going back to our original cell image

1. We have been able to *get rid of the noise* in the image and *find all the cells* (**lecture 2-4**)
2. We have analyzed the shape of the cells using the shape tensor (**lecture 5**)
3. We even *separated cells* joined together using Watershed (**lecture 6**)
4. We have created even more *metrics characterizing the distribution* (**lecture 7**)

We have at least a few samples (or different regions), large number of metrics and an almost as large number of parameters to *tune*

How do we do something meaningful with it?

0.5.2 Pixel/Voxel-based Methods

- Cross Correlation
- DIC
- DIC + Physics
- Affine Transforms
- Non-rigid transform

Keypoint Detection

- Corner Detectors
- SIFT/SURF
- Tracking from Keypoints

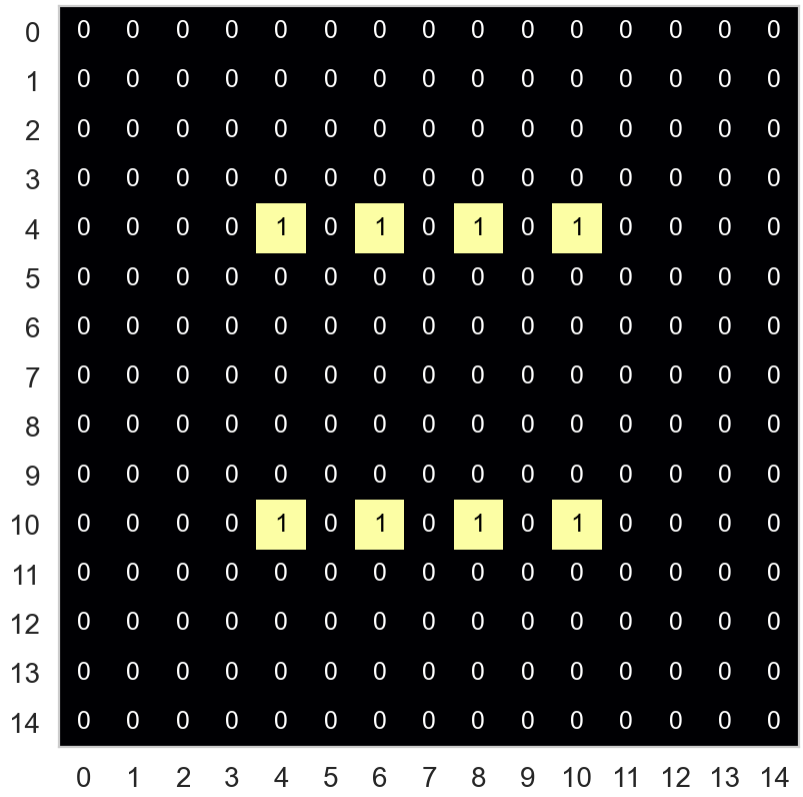
0.5.3 General Problems

- Thickness - Lung Tissue
- Curvature - Metal Systems
- Two Point Correlation - Volcanic Rock

0.5.4 A basic simulation

We start with an initial image with a number of points on a plane

```
import numpy as np
import matplotlib.pyplot as plt
from skimage.io import imread
import sys
sys.path.append('../common')
import plotsupport as ps

%matplotlib inline
xx, yy = np.meshgrid(np.linspace(-1.5, 1.5, 15),
                     np.linspace(-1.5, 1.5, 15))

N_DISK_ROW = 2
N_DISK_COL = 4
DISK_RAD = 0.15
disk_img = np.zeros(xx.shape, dtype=int)
for x_cent in 0.7*np.linspace(-1, 1, N_DISK_COL):
    for y_cent in 0.7*np.linspace(-1, 1, N_DISK_ROW):
        c_disk = np.sqrt(np.square(xx-x_cent)+np.square(yy-y_cent)) < DISK_RAD
        disk_img[c_disk] = 1
fig, ax1 = plt.subplots(1, 1)

# sns.heatmap(disk_img, annot=True, fmt='d', ax=ax1);
ps.heatmap(disk_img,bgbox=False,ax=ax1, precision=0, fontsize=9)
```



0.5.5 Create a series of moving “cells”

The cells will move with *dx=-1* and *dy=-1*

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(10, 10), dpi=200)

s_img = disk_img.copy()
img_list = [s_img]
for i in range(4):
    s_img = np.roll(s_img, -1, axis=1)
    s_img = np.roll(s_img, -1, axis=0)
    img_list += [s_img]

def update_frame(i):
    plt.cla()
#     sns.heatmap(img_list[i],annot=True,
#                 fmt="d",cmap='nipy_spectral',
#                 ax=c_ax,cbar=False,
#                 vmin=0,vmax=1)
    ps.heatmap(img_list[i],bgbox=False,ax=c_ax,precision=0,fontsize=9)
    c_ax.set_title('Iteration #{}'.format(i+1))

# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
```

(continues on next page)

(continued from previous page)

```
                          frames=len(img_list),
                          interval=1000, repeat_delay=2000)
#anim_code.save('movies/moving_cells.mp4', fps=2, extra_args=['-vcodec', 'libx264'])
anim_code.save('movies/moving_cells.mp4', fps=2)
plt.close('all')
```

0.5.6 Analysis

The analysis of the series requires the following steps:

1. Threshold
2. Component Label
3. Shape Analysis (label, position, area, and frame id)
4. Distribution Analysis

… and to put all in a data frame

The analysis code

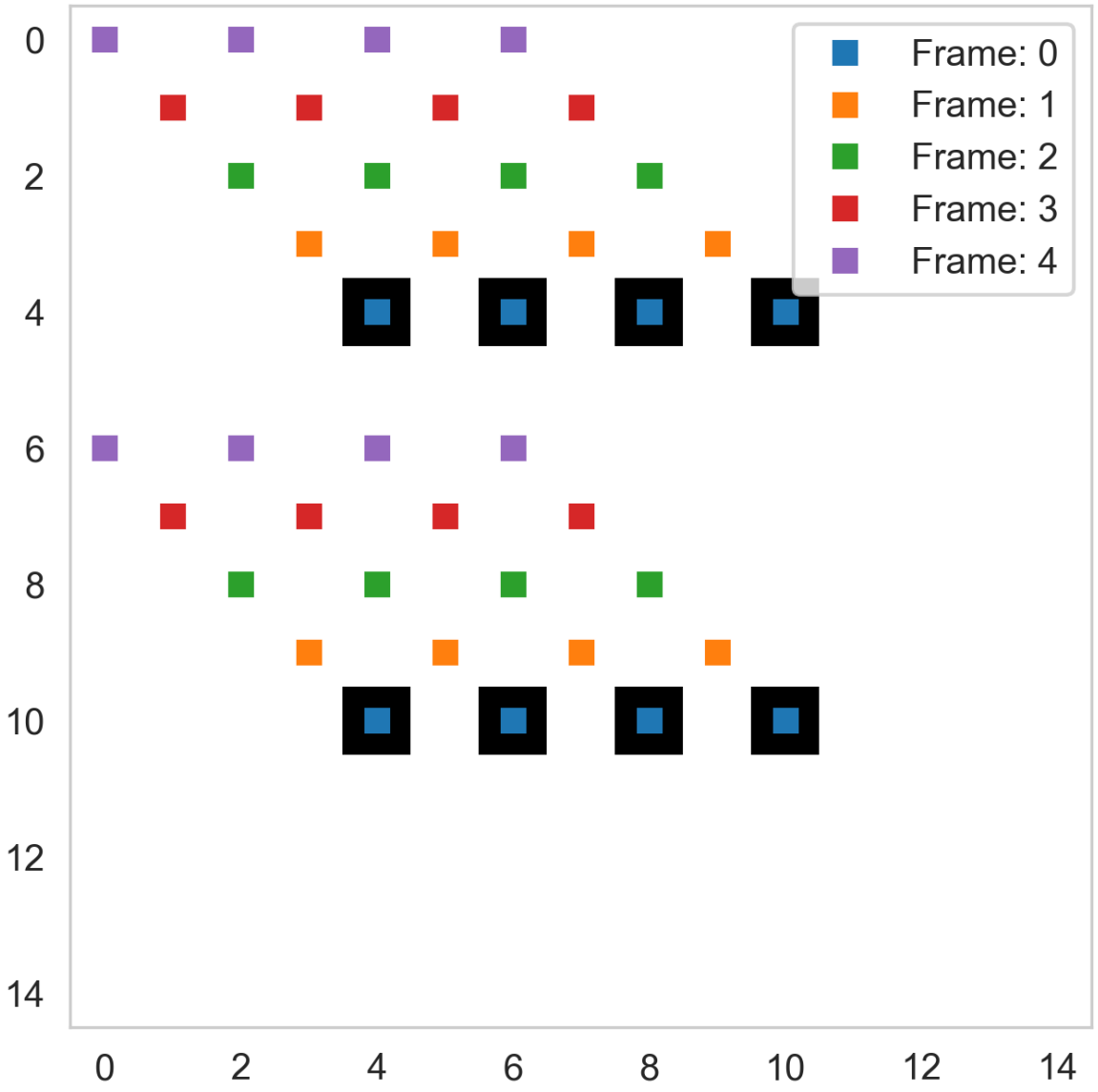
```
all_objs = []
for frame_idx, c_img in enumerate(img_list):        # For each time frame
    lab_img = label(c_img > 0)                      # Label the items
    for c_obj in regionprops(lab_img):              # Put region properties for each␣
 ↪object of the time frame
        all_objs += [dict(label     = int(c_obj.label),
                          y         = c_obj.centroid[0],
                          x         = c_obj.centroid[1],
                          area      = c_obj.area,
                          frame_idx = frame_idx)]

all_obj_df = pd.DataFrame(all_objs)                 # Create a Pandas data frame with␣
 ↪all the properties
all_obj_df.head(5)                                  # Look at the first five rows of␣
 ↪the data frame
```

	label	y	x	area	frame_idx
0	1	4.0	4.0	1.0	0
1	2	4.0	6.0	1.0	0
2	3	4.0	8.0	1.0	0
3	4	4.0	10.0	1.0	0
4	5	10.0	4.0	1.0	0

Looking at the positions of the items in all frames

Here, we look at the positions of the items found in each time frame. The information comes from the data frame we just created. We only use position and frame index here.

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.imshow(disk_img, cmap='bone_r')
for frame_idx, c_rows in all_obj_df.groupby('frame_idx'):
    c_ax.plot(c_rows['x'], c_rows['y'], 's', label='Frame: %d' % frame_idx)
c_ax.legend();
```

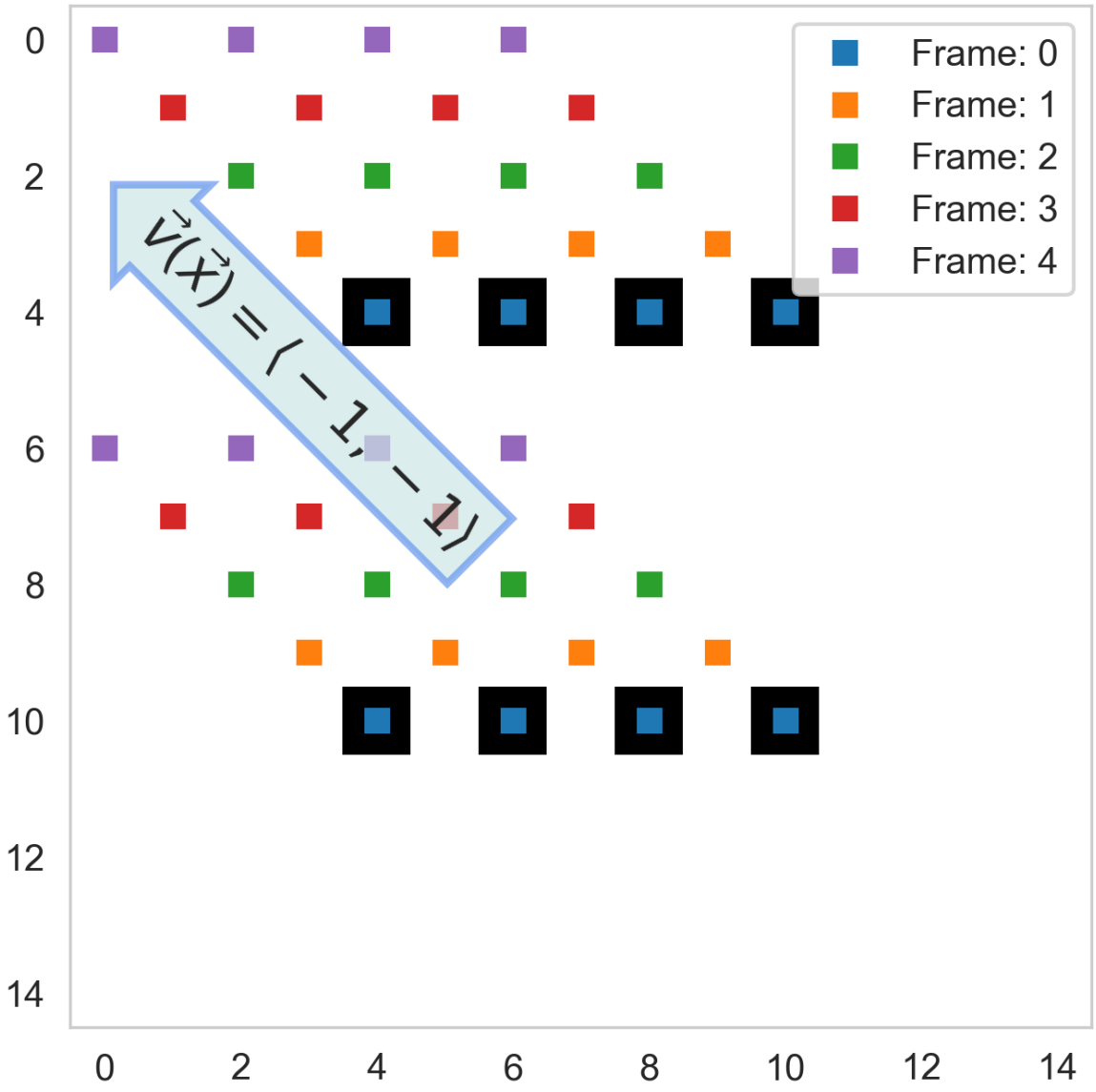


Describing Motion

We can describe the motion in the above example with a simple vector

$$\vec{v}(\vec{x}) = \langle -1, -1 \rangle$$

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.imshow(disk_img, cmap='bone_r')
for frame_idx, c_rows in all_obj_df.groupby('frame_idx'):
    c_ax.plot(c_rows['x'], c_rows['y'], 's', label='Frame: %d' % frame_idx)
c_ax.legend();

bbox_props = dict(boxstyle="larrow", fc=(0.8, 0.9, 0.9), ec='cornflowerblue', lw=2,
 ↪alpha=0.7)
t = c_ax.text(3,5, r"$ \vec{v}(\vec{x})=\langle -1,-1 \rangle $", ha="center", va=
 ↪"center", rotation=-45,
            size=15,
            bbox=bbox_props)
```



This is a very simple case of motion as it only follows a straight line towards the upper left corner. Motion in real experiments are much more complicated and it can also ambiguous which item is which when the trajectories are crossing.

0.5.7 Tracking methods

While there exist a number of different methods and complicated approaches for tracking.

For experimental design it is best to start with the (Occam's razor)

- simplest
- easiest understood method.

The limits of this can be found and components added as needed until it is possible to realize the experiment

---

If a dataset can only be analyzed with a multiple-hypothesis testing neural network model then it might not be so reliable

Scoring Tracking

In the following examples we will use simple metrics for scoring fits where the objects are matched and the number of misses is counted.

There are a number of more sensitive scoring metrics which can be used, by finding the best submatch for a given particle since the number of matches and particles does not always correspond.

See the papers at the beginning for more information

0.5.8 Tracking using Nearest Neighbor

We then return to *nearest neighbor* which means we track

- a point ($\vec{P}_0$) from an image (I_0) at t_0
- to a point ($\vec{P}_1$) in image (I_1) at t_1

by

$$\vec{P}_1 = \text{argmin}(\|\vec{P}_0 - \vec{y}\| \; \forall \vec{y} \in I_1)$$

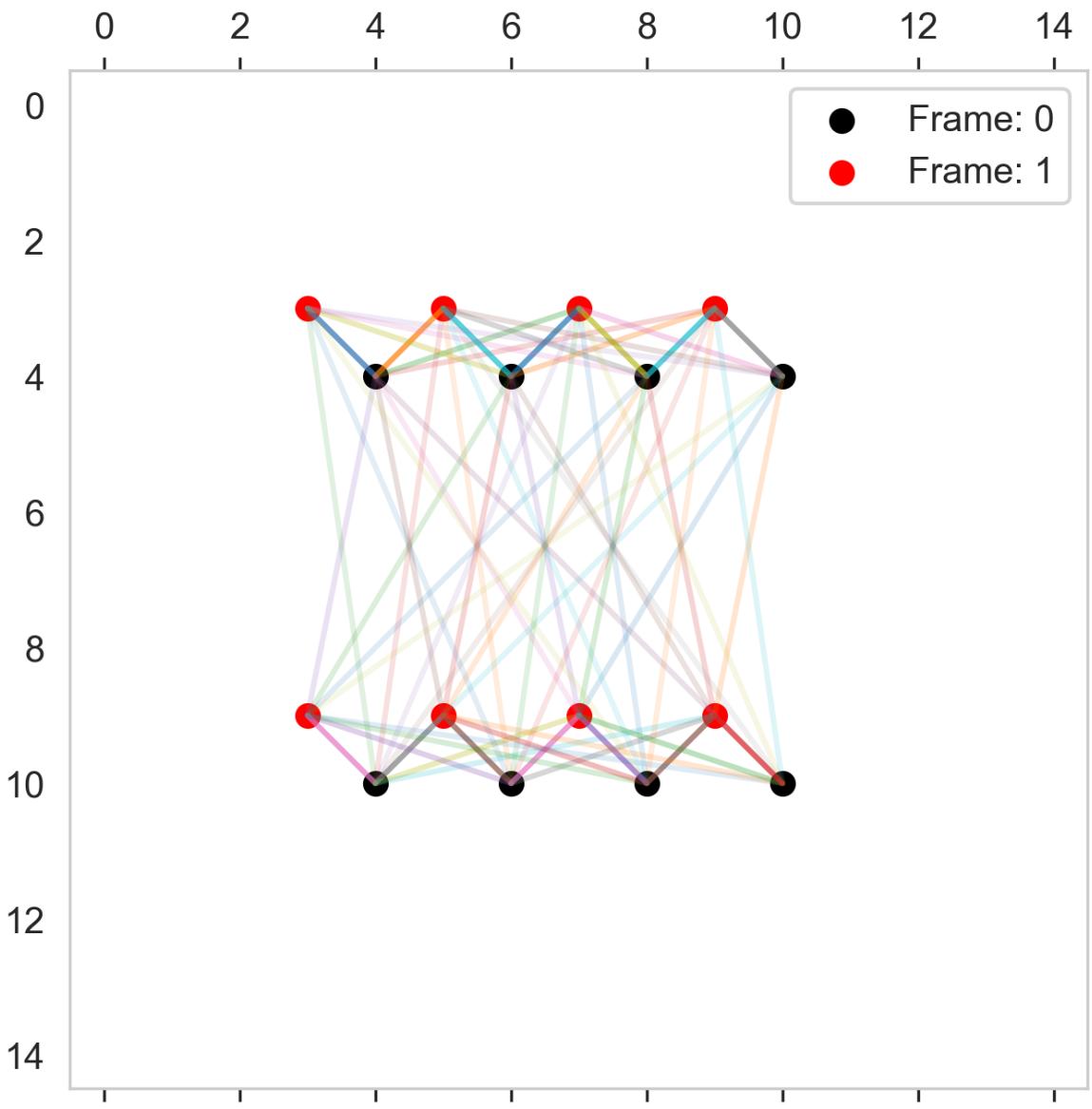
0.5.9 Distances between objects

The plot below shows the distances from each object to all other objects. Without additional information like

- we are looking for the nearest neighbor
- or the expected displacement is bounded within some limits.
- shape characteristics of the items

It is impossible to tell which is the matching object between frame 0 and 1.

```
frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([0])]
frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([1])]
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(1 < disk_img , cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')
dist_df_list = []
for _, row_0 in frame_0.iterrows():
    for _, row_1 in frame_1.iterrows():
        seg_dist = np.sqrt(np.square(row_0['x']-row_1['x']) +
                           np.square(row_0['y']-row_1['y']))
        c_ax.plot([row_0['x'], row_1['x']],
                  [row_0['y'], row_1['y']], '-', alpha=1/seg_dist)
        dist_df_list += [dict(x0=row_0['x'],
                              y0=row_0['y'],
                              lab0=int(row_0['label']),
                              x1=row_1['x'],
                              y1=row_1['y'],
                              lab1=int(row_1['label']),
                              dist=seg_dist)]
c_ax.legend();
```



0.5.10 Put the distances in a data frame

```
dist_df = pd.DataFrame(dist_df_list)
dist_df.head(5)
```

	x0	y0	lab0	x1	y1	lab1	dist
0	4.0	4.0	1	3.0	3.0	1	1.414214
1	4.0	4.0	1	5.0	3.0	2	1.414214
2	4.0	4.0	1	7.0	3.0	3	3.162278
3	4.0	4.0	1	9.0	3.0	4	5.099020
4	4.0	4.0	1	3.0	9.0	5	5.099020

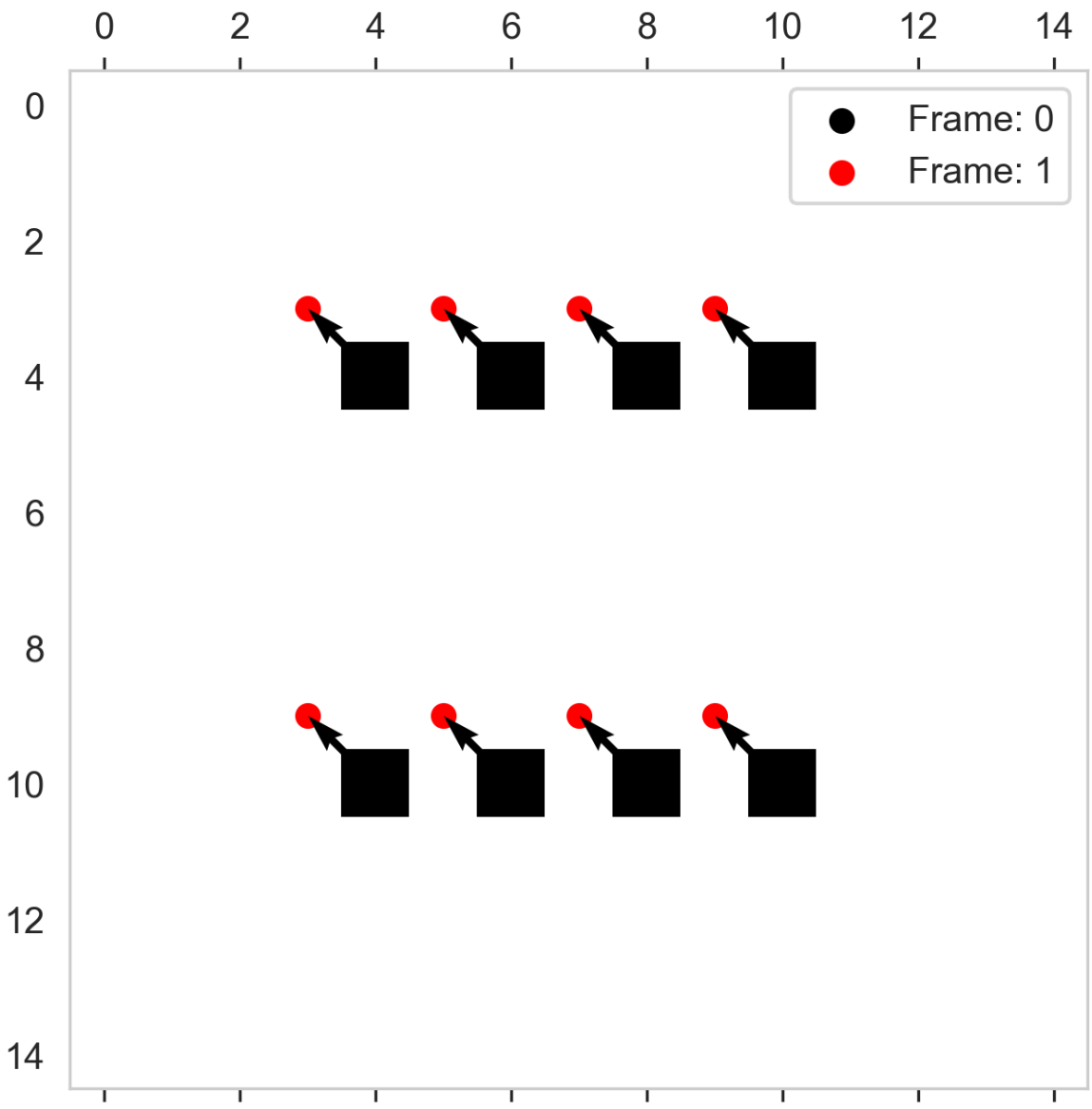
Plotting the distances

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img > 1, cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')

def update_frame(i):
    # plt.cla()
    c_rows = dist_df.query('lab0==%d' % i)
    for _, c_row in c_rows.iterrows():
        c_ax.quiver(c_row['x0'], c_row['y0'],
                    c_row['x1']-c_row['x0'],
                    c_row['y1']-c_row['y0'], scale=1.0, scale_units='xy', angles='xy',
                    alpha=1/c_row['dist'])
    c_ax.set_title('Point #{}'.format(i+1))



# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=np.unique(dist_df['lab0']),
                          interval=1000,
                          repeat_delay=2000)

#anim_code.save('movies/neighbor_distances.mp4', fps=2, extra_args=['-vcodec',
 ↪'libx264'])
anim_code.save('movies/neighbor_distances.mp4', fps=2)
plt.close('all')
```

With the nearest neighbor criterion we still have the problem because most items have two possible next position.

Looking at the nearest points found

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img >= 1,cmap='gist_yarg')
c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: 0')
c_ax.scatter(frame_1['x'], frame_1['y'], c='red', label='Frame: 1')
for _, c_rows in dist_df.groupby('lab0'):
    _, c_row = next(c_rows.sort_values('dist').iterrows())
    c_ax.quiver(c_row['x0'], c_row['y0'],
                c_row['x1']-c_row['x0'],
                c_row['y1']-c_row['y0'],
                scale=1.0, scale_units='xy', angles='xy')

c_ax.legend();
```



Tracking in all frames

This tracking seems to be success full. The reason is that we were lucky that the direction is -1,-1. On the other hand, it is relatively rare that you have perfect positioning of the items and therefore this ambiguity doesn't happen very often.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)
c_ax.matshow(disk_img >= 1,cmap='gist_yarg')


def draw_timestep(i):
    # plt.cla()
```

(continues on next page)

(continued from previous page)

```
    frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([i])]
    frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([i+1])]
    c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: %d' % i)
    c_ax.scatter(frame_1['x'], frame_1['y'],
                 c='red', label='Frame: %d' % (i+1))
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        c_ax.quiver(best_row['x0'], best_row['y0'],
                    best_row['x1']-best_row['x0'],
                    best_row['y1']-best_row['y0'],
                    scale=1.0, scale_units='xy', angles='xy')
    c_ax.set_title('Frame #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_timestep,
                          frames=all_obj_df['frame_idx'].max(),
                          interval=1000,
                          repeat_delay=2000)

#anim_code.save('movies/tracking_all_frames.mp4', fps=2, extra_args=['-vcodec',
 ↪'libx264'])
anim_code.save('movies/tracking_all_frames.mp4', fps=2)
plt.close('all')
```

0.6 Key Points (or feature points)

Tracking and registration using the full data set is time demaning.

- We can detect feature points in an image and use them to make a registration.

The key points located at characteristic positions around the obejct and their positions relative to each other are invariant to translation and rotation. The points will still be at the same features when the object is skewed, but then they obtain new relative positions compared to the original.

0.6.1 Identifying key points

We first focus on the detection of points.

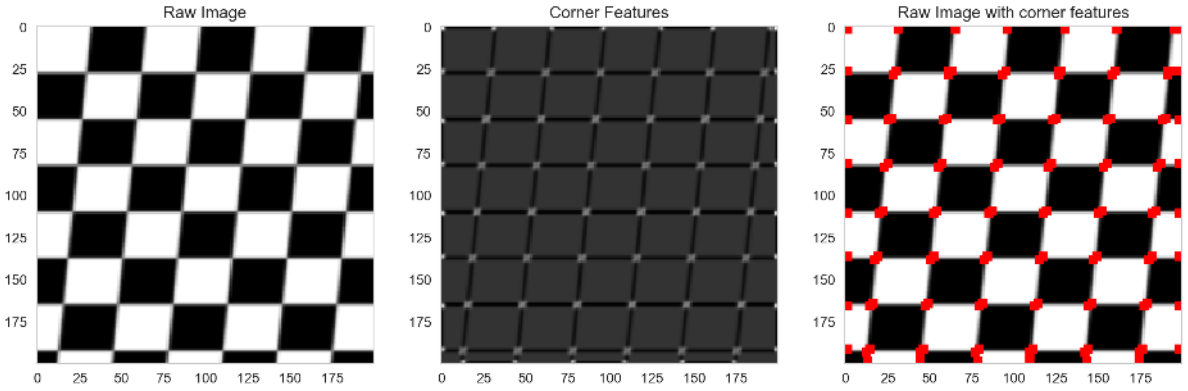
- Corners are of most interest.

Many methods have been proposed to detect corners. A Harris corner detector helps us here:

The Harris corner detector uses the structure tensor to identify corner points in the image. In the checkerboard example below we identify the corners using the Harris corner detector. In the corner feature image in the middle you see that the corners/crossings provide a strong response. The feature image is thresholded to provide the corner points, which are plotted in the panel to the right.

```
tform       = AffineTransform(scale=(1.3, 1.1), rotation=0, shear=0.1,translation=(0,␣
 ↪0))
image       = warp(data.checkerboard(), tform.inverse, output_shape=(200, 200))

corners     = corner_harris(image)
peak_coords = corner_peaks(corners,threshold_rel=0)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(image,interpolation='none',cmap='gray'); ax1.set_title('Raw Image')

ax2.imshow(corners,interpolation='none',cmap='gray'); ax2.set_title('Corner Features')

ax3.imshow(image,interpolation='none',cmap='gray'); ax3.set_title('Raw Image with␣
 ↪corner features')
ax3.plot(peak_coords[:, 1], peak_coords[:, 0], 's',color='red');
```



0.6.2 Let's try the corner detection on real data

We return to the bone image we have use many times before and test the Harris detector to identify key points in a natural image.

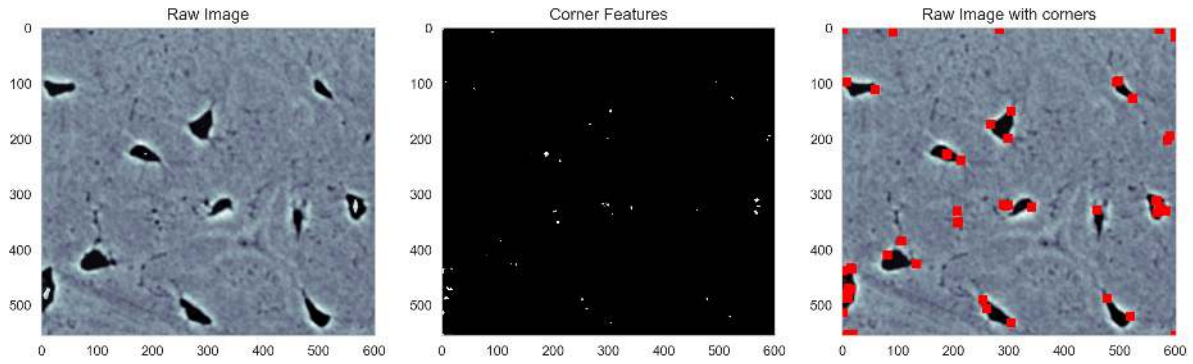
```
full_img    = imread("figures/bonegfiltslice.png").mean(axis=2)
corners     = corner_harris(full_img)
peak_coords = corner_peaks(corners,threshold_rel=0.002)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))
ax1.imshow(full_img,cmap='bone')
```

(continues on next page)

(continued from previous page)

```
ax1.set_title('Raw Image')
ax2.imshow(1.5e7<corners, cmap='gray', interpolation='none'),
ax2.set_title('Corner Features')


ax3.imshow(full_img,cmap='bone'),
ax3.set_title('Raw Image with corners')
ax3.plot(peak_coords[:, 1], peak_coords[:, 0], 'rs');
```



0.7 Tracking with Points

Goal: To reducing the tracking efforts

We can use the corner points to track features between multiple frames.

In this sample, we see that they are

- quite stable
- and fixed

on the features.

0.7.1 We need data - a series transformed images

In this example we apply different affine transformations like

- Translation in x and y
- Rotation
- Shear

to see how well the Harris detector copes with these.

The animation clearly shows that the corners are found.

It can however happen that there are various number of pixels allocated to each corner point. This can be pruned using morphological algorihms if needed.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=100)


def update_frame(i):
    c_ax.cla()
    tform = AffineTransform(scale=(1.3+i/20, 1.1-i/20), rotation=-i/10, shear=i/20,
                            translation=(0, 0))
    image = warp(data.checkerboard(), tform.inverse, output_shape=(200, 200))
    c_ax.imshow(image,interpolation='none')
    peak_coords = corner_peaks(corner_harris(image),threshold_rel=0.1)
    c_ax.plot(peak_coords[:, 1], peak_coords[:, 0], 's',color='lightgreen')


# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=np.linspace(0, 5, 10),
                          interval=1000,
                          repeat_delay=2000)


#anim_code.save('movies/affine_transformation.mp4', fps=2, extra_args=['-vcodec',
 ↪'libx264'])
anim_code.save('movies/affine_transformation.mp4', fps=2)
plt.close('all')
```

0.8 Features and Descriptors

We can move beyond just key points to key points and feature vectors (called descriptors) at those points.

A descriptor is a vector that describes a given keypoint uniquely.

A common descriptor is computed using the ORB algorithm based on

- FAST keypoint detector (**F** eatures from **a** ccelerated **s** egment **t** est)
- BRIEF descriptor (**B** inary **R** obust **I** ndependent **E** lementary **F** eatures) doi

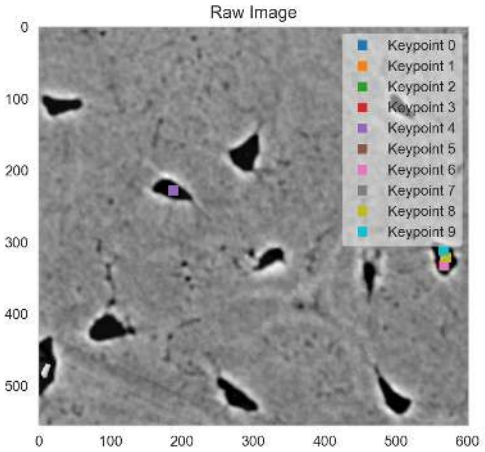
This will be demonstrated using two methods in the following notebook cells…

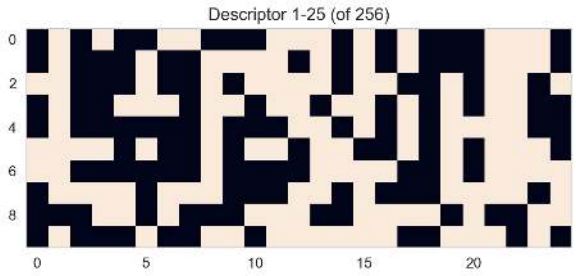
```
from skimage.feature import ORB
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
orb_det  = ORB(n_keypoints=10)
det_obj  = orb_det.detect_and_extract(full_img)

fig, (ax3, ax5) = plt.subplots(1, 2, figsize=(15, 5))
ax3.imshow(full_img, cmap='gray')
ax3.set_title('Raw Image')
for i in range(orb_det.keypoints.shape[0]):
    ax3.plot(orb_det.keypoints[i, 1], orb_det.keypoints[i,
                                                        0], 's', label='Keypoint {}'.
 ↪format(i))
```

(continues on next page)

(continued from previous page)

```
ax5.imshow(np.stack([x[:25] for x in orb_det.descriptors], 0))
ax5.set_title('Descriptor 1-25 (of 256)')
ax3.legend(facecolor='white', framealpha=0.5);
```





0.8.1 Defining a supporting function to show the matches

This is a support function used to display the matched feature points between two images.

```
from skimage.feature import match_descriptors, plot_matches
import matplotlib.pyplot as plt


import numpy as np
import matplotlib.pyplot as plt

def plot_matched_features(ax, img1, img2,
                          keypoints1, keypoints2, matches,
                          keypoints_color='r', line_color='y'):
    """
    Plot matched keypoints between two images, using plain Matplotlib calls.
    Roughly mimics the old skimage 'plot_matched_features' function.
    """
    # Step 1: Concatenate the images side by side
    #         (assuming they have the same height or you handle mismatched sizes)
    if img1.shape[0] != img2.shape[0]:
        # Optionally resize or pad one image so they match in height
        pass
    combined = np.concatenate([img1, img2], axis=1)

    # Step 2: Display the combined image
    ax.imshow(combined, cmap='gray')

    # Step 3: For each match, plot the keypoints and a line connecting them
    offset_x = img1.shape[1]  # offset for the second image's x-coordinates

    for (i1, i2) in matches:
        y1, x1 = keypoints1[i1]
        y2, x2 = keypoints2[i2]
```

(continues on next page)

(continued from previous page)

```
        x2_shifted = x2 + offset_x

        # Plot the matched keypoints
        ax.plot(x1, y1, marker='o', color=keypoints_color, ms=5)
        ax.plot(x2_shifted, y2, marker='o', color=keypoints_color, ms=5)

        # Draw a line between them
        ax.plot([x1, x2_shifted], [y1, y2], color=line_color, linewidth=1)

    ax.axis('off')  # Hide the axis if desired


def show_matches(img1, img2, feat1, feat2):
    matches12 = match_descriptors(feat1['descriptors'],
                                  feat2['descriptors'],
                                  cross_check=True)

    fig, (ax3, ax2) = plt.subplots(1, 2, figsize=(15, 5))

    plot_matched_features(ax3,
                  img1, img2,
                  feat1['keypoints'], feat2['keypoints'],
                  matches12,
                  keypoints_color='r')

    ax3.axvline(x=img1.shape[1]-1,color='white',linewidth=2)

    ax2.plot(feat1['keypoints'][:, 1],
             feat1['keypoints'][:, 0],
             '.',
             label='Before')

    ax2.plot(feat2['keypoints'][:, 1],
             feat2['keypoints'][:, 0],
             '.', label='After')

    for i, (c_idx, n_idx) in enumerate(matches12):
        x_vec = [feat1['keypoints'][c_idx, 0], feat2['keypoints'][n_idx, 0]]
        y_vec = [feat1['keypoints'][c_idx, 1], feat2['keypoints'][n_idx, 1]]
        dist = np.sqrt(np.square(np.diff(x_vec))+np.square(np.diff(y_vec)))
        alpha = np.clip(50/dist, 0, 1)[0]

        ax2.plot(
            y_vec,
            x_vec,
            'k-',
            alpha=alpha,
            label='Match' if i == 0 else ''
        )

    ax2.legend()

    ax3.set_title(r'{} $\rightarrow$ {}'.format('Before', 'After'));
```

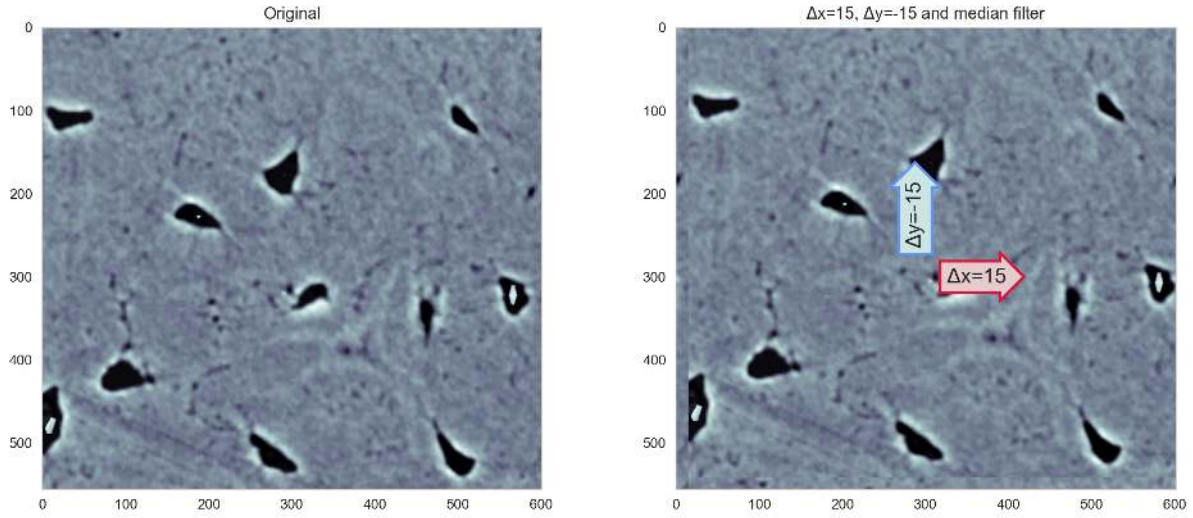
0.8.2 Let's create some data

We make the following modifications to the bone image:

- affine transformation (linear displacement)
- blurring (median filter)

For this example we create a pair of images, where one translated and a smoothing median filter is applied. We will use this image pair to dest different descriptors in the next sections.

```
from skimage.filters import median
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
full_shift_img = median(
    np.roll(np.roll(full_img, -15, axis=0), 15, axis=1), np.ones((1, 3)))

bw_img    = full_img
shift_img = full_shift_img

fig,ax = plt.subplots(1,2,figsize=(15,6))
ax[0].imshow(bw_img,cmap='bone')
ax[0].set_title('Original')
ax[1].imshow(shift_img,cmap='bone')
ax[1].set_title('$\Delta{}$x=15, $\Delta{}$y=-15 and median filter');

bbox_props = dict(boxstyle="rarrow", fc=(0.9, 0.8, 0.8), ec="crimson", lw=2)
ax[1].text(325, 300, r"$\Delta{}$x=15", ha="left", va="center", rotation=0,
           size=15,
           bbox=bbox_props)

bbox_props = dict(boxstyle="rarrow", fc=(0.8, 0.9, 0.9), ec="cornflowerblue", lw=2)
ax[1].text(275, 225, r"$\Delta{}$y=-15", ha="left", va="center", rotation=90,
           size=15,
           bbox=bbox_props);
```



0.8.3 Features found by the BRIEF descriptor

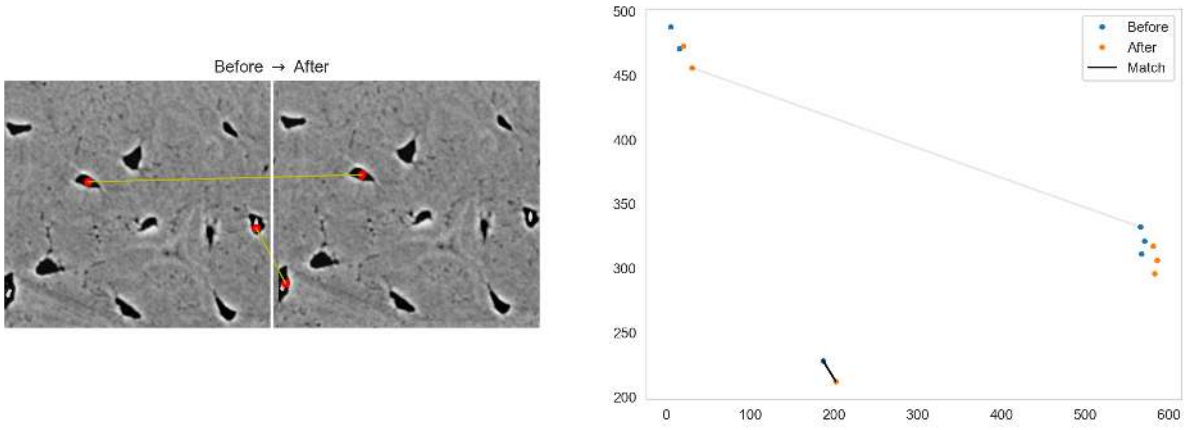
Scan the neighborhood of pixel x, $\mathcal{N}(x)$ to create at local fingerprint

$$\forall_{y\in\mathcal{N}(x)} \begin{cases} 1 & p(x) < p(y) \\ 0 & p(x) \geq p(y) \end{cases}$$

Produces a binary number where each bit is represented by a neighbor pixel.

In this first example we use a combination of the Harris corner detector and BRIEF to create descriptors.

```
from skimage.feature import corner_peaks, corner_harris, BRIEF

def calc_corners(*imgs):
    b = BRIEF()
    for c_img in imgs:
        corner_img = corner_harris(c_img)
        coords = corner_peaks(corner_img, min_distance=5,threshold_rel=0.1)
        b.extract(c_img, coords)
        yield {'keypoints'  : coords,
               'descriptors': b.descriptors}


feat1, feat2 = calc_corners(bw_img, shift_img)
show_matches(bw_img, shift_img, feat1, feat2)
```



You see two matches in the right panel. One short and the other obviously far away which is less likely to be realistic.

0.8.4 Features found by the ORB descriptor

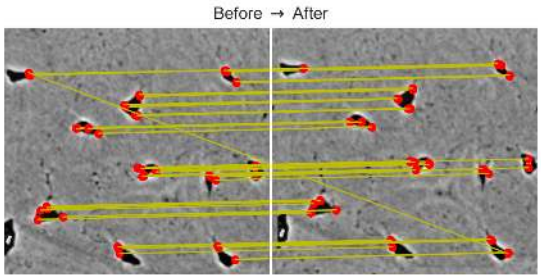
The ORB descriptor which has a more robust algorithm is able to find many more good pairs between the two time steps.

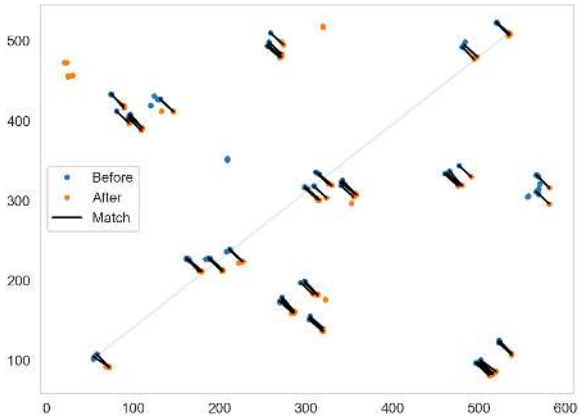
```
from skimage.feature import ORB, BRIEF, CENSURE


def calc_orb(*imgs):
    descriptor_extractor = ORB(n_keypoints=100)
    for c_img in imgs:
        descriptor_extractor.detect_and_extract(c_img)
        yield {'keypoints': descriptor_extractor.keypoints,
```

(continues on next page)

(continued from previous page)

```
                'descriptors': descriptor_extractor.descriptors}


feat1, feat2 = calc_orb(bw_img, shift_img)
show_matches(bw_img, shift_img, feat1, feat2)
```





0.9 Computing Average Flow

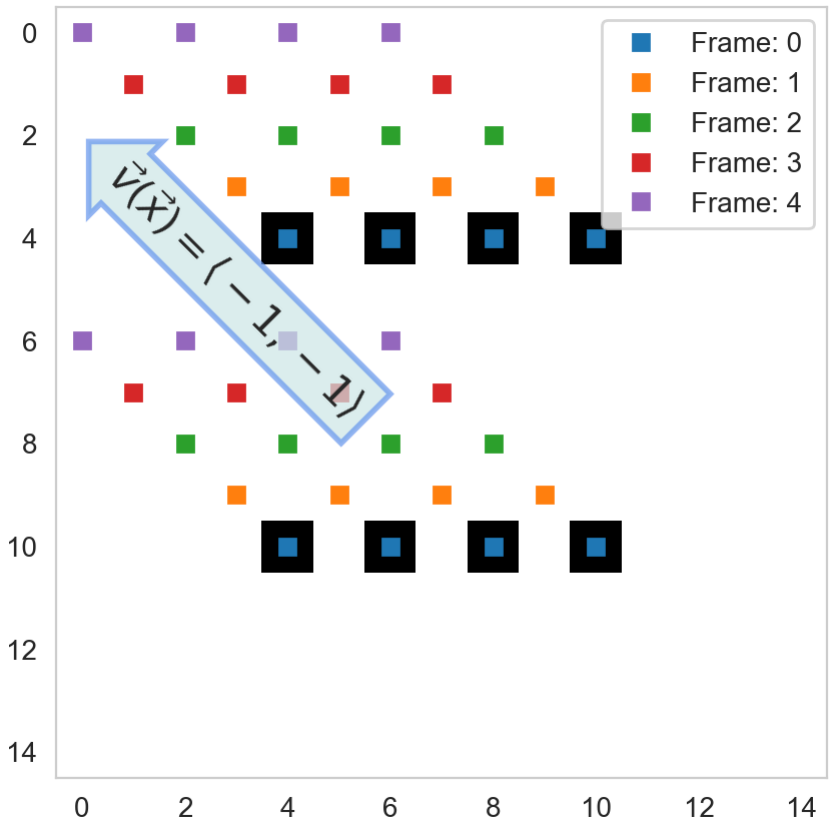
From each of the time steps in a time series we can now proceed to compute the average flow.

We can perform this :

- spatially (averaging over regions),
- temporally (averaging over time),
- or spatial-temporally (averaging over regions for every time step)

Let's return to the $\langle -1, -1 \rangle$ flow data set:

```
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5))
c_ax.imshow(disk_img, cmap='bone_r')
for frame_idx, c_rows in all_obj_df.groupby('frame_idx'):
    c_ax.plot(c_rows['x'], c_rows['y'], 's', label='Frame: %d' % frame_idx)
c_ax.legend();

bbox_props = dict(boxstyle="larrow", fc=(0.8, 0.9, 0.9), ec='cornflowerblue', lw=2,
 ↪alpha=0.7)
t = c_ax.text(3,5, r"$ \vec{v}(\vec{x})=\langle -1,-1 \rangle $", ha="center", va=
 ↪"center", rotation=-45,
            size=15,
            bbox=bbox_props)
```



0.9.1 Compute the average velocity vector

Here we want the average of all displacements

```
average_field = []
for i in range(all_obj_df['frame_idx'].max()):
    frame_0 = all_obj_df[all_obj_df['frame_idx'].isin([i])]
    frame_1 = all_obj_df[all_obj_df['frame_idx'].isin([i+1])]
    dist_df_list = []

    # Compute distances to all points between two frames
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df = pd.DataFrame(dist_df_list)
    # Get shortest distance for each label
    for _, c_rows in dist_df.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
```

(continues on next page)

(continued from previous page)

```
        average_field += [dict(frame_idx=i,
                               x=best_row['x0'],
                               y=best_row['y0'],
                               dx=best_row['x1']-best_row['x0'],
                               dy=best_row['y1']-best_row['y0'])]
average_field_df = pd.DataFrame(average_field)
print('Average Flow:\n{}'.format(average_field_df[['dx', 'dy']].mean()))

average_field_df.sample(5)
```

```
Average Flow:
dx   -1.0
dy   -1.0
dtype: float64
```

	frame_idx	x	y	dx	dy
10	1	7.0	3.0	-1.0	-1.0
15	1	9.0	9.0	-1.0	-1.0
29	3	3.0	7.0	-1.0	-1.0
12	1	3.0	9.0	-1.0	-1.0
7	0	10.0	10.0	-1.0	-1.0

Here, create an average field using the dx=-1, dy=-1 moving cells in the all_obj_df. The result is put in a new data frame.

The loop compares items in the current frame with items in the next frame and computes the displacements between all items. A second loop sorts the displacements to find the best combinations.

At the end, the average flow is computed.

0.9.2 Spatially Averaging

To spatially average we first create a grid of values and then interpolate our results onto this grid

0.9.3 Averaging vector fields

Vector fields can be very jittery in noisy studies. Spatial averaging can reduce the jitter.

```
from scipy.interpolate import RegularGridInterpolator, LinearNDInterpolator

def img_intp(f):
    def new_f(x, y):
        return np.stack([f(ix, iy) for ix, iy in zip(np.ravel(x), np.ravel(y))], 0).
 ↪reshape(np.shape(x))
    return new_f


dx_func = img_intp(
    LinearNDInterpolator((average_field_df['x'], average_field_df['y']), average_
 ↪field_df['dx']))
dy_func = img_intp(
    LinearNDInterpolator((average_field_df['x'], average_field_df['y']), average_
 ↪field_df['dy']))
dx_func(8, 8), dy_func(8, 8)
```

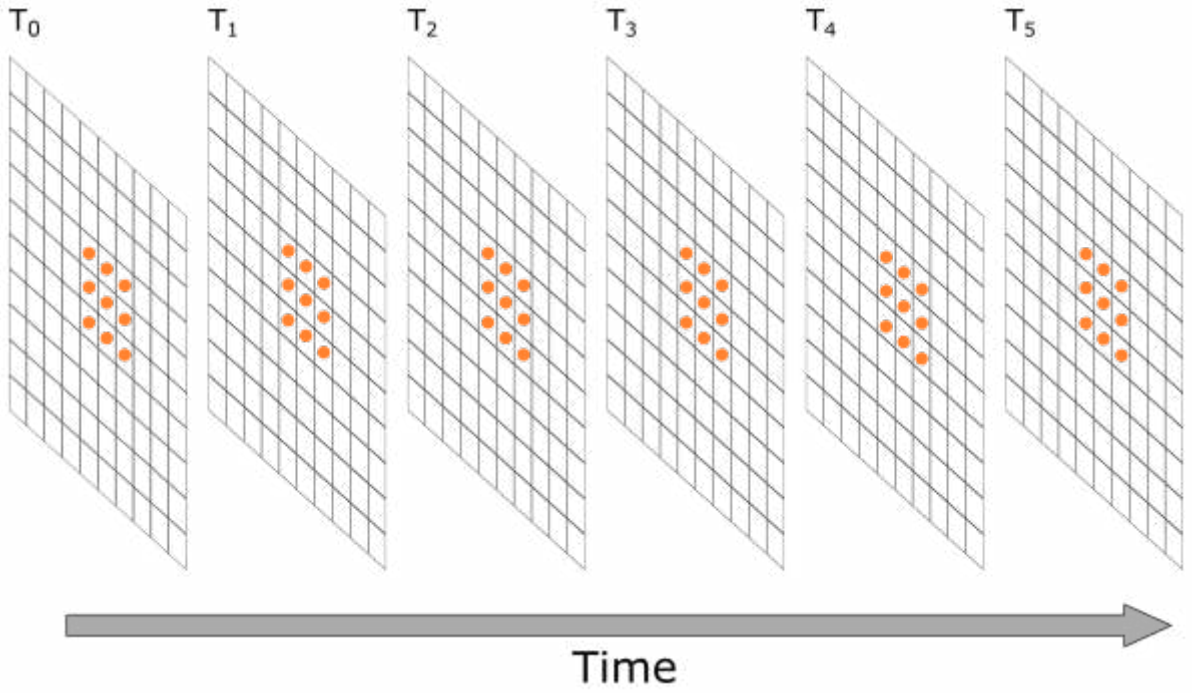
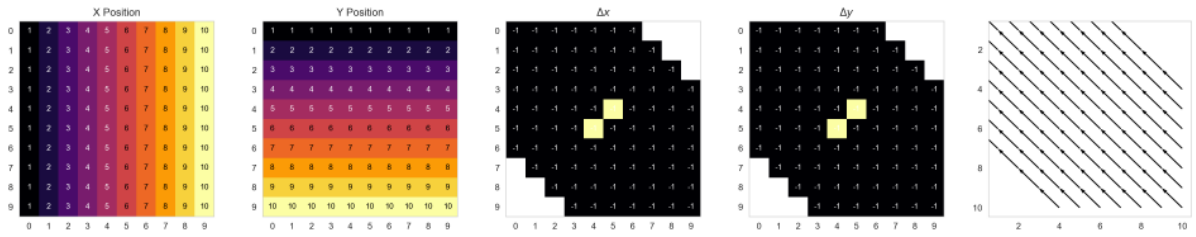


Fig. 1: Spatial averaging.

```
(array(-1.), array(-1.))
```

```
g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 10),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 10))
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(24, 4))
ps.heatmap(g_x, bgbox=False,ax=ax1, precision=0, fontsize=9)
ax1.set_title('X Position')
ps.heatmap(g_y, bgbox=False,ax=ax2, precision=0, fontsize=9)
ax2.set_title('Y Position')

ps.heatmap(dx_func(g_x, g_y), bgbox=False,ax=ax3, precision=0, fontsize=9)
ax3.set_title('$\Delta x$')
ps.heatmap(dy_func(g_x, g_y), bgbox=False,ax=ax4, precision=0, fontsize=9)

ax4.set_title('$\Delta y$')
ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
           scale=1.0, scale_units='xy', angles='xy');
ax5.invert_yaxis()
```



0.9.4 Temporarly Averaging

Here we take the average at each time point

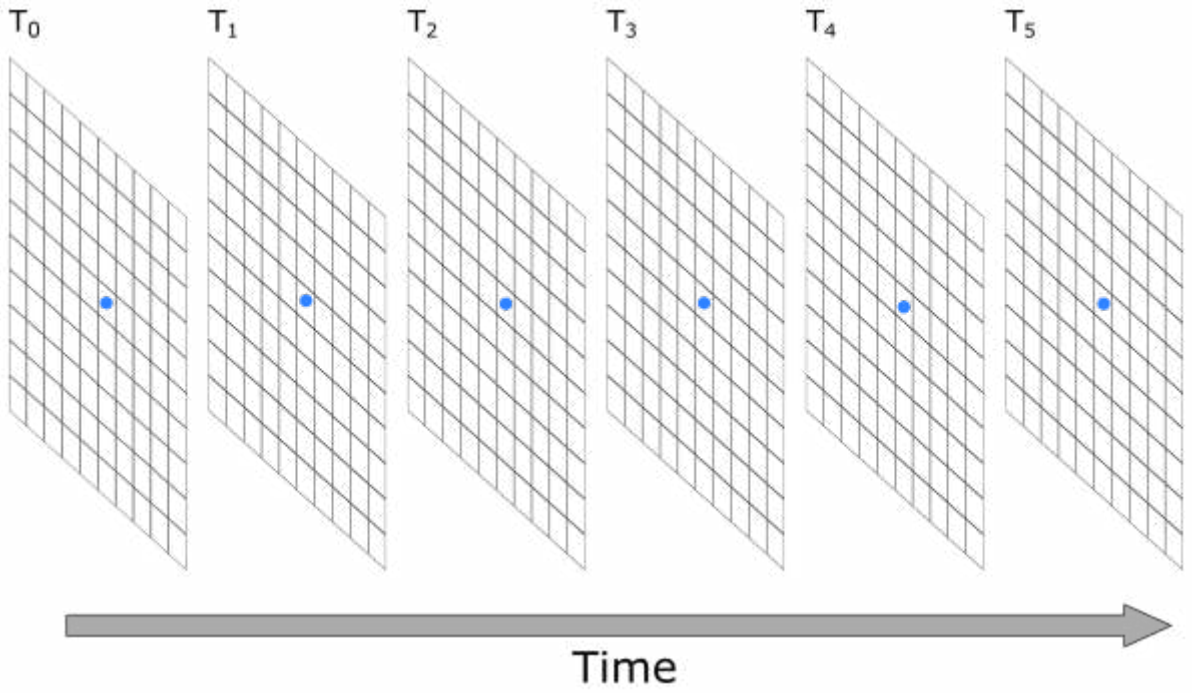


Fig. 2: Temporarly averaging.

Temporarly averaging on the cell movement

Compute the average displacement $\langle\overline{\Delta x_i}, \overline{\Delta y_i}\rangle$ for each time frame i.

```
temp_avg_field = average_field_df[['frame_idx', 'dx', 'dy']].groupby(
    'frame_idx').agg('mean').reset_index()
temp_avg_field
```

```
   frame_idx   dx   dy
0          0 -1.0 -1.0
1          1 -1.0 -1.0
2          2 -1.0 -1.0
3          3 -1.0 -1.0
```

The time frame average is computed using some pandas craft.

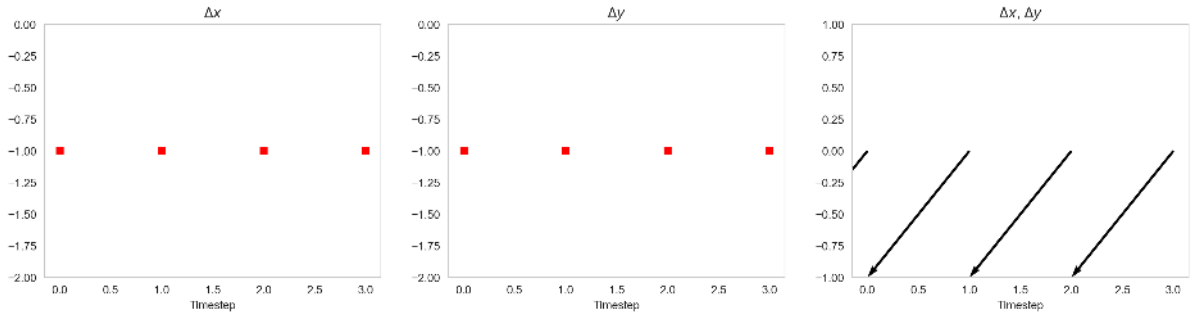
1. Extract the dx and dy from each frame
2. Group by the frame_idx column
3. Compute the average of dx and dy for each frame.

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 4))
ax1.plot(temp_avg_field['frame_idx'], temp_avg_field['dx'], 'rs')
ax1.set_title('$\Delta x$')
```

(continues on next page)

(continued from previous page)

```
ax1.set_xlabel('Timestep')
ax1.set_ylim([-2,0])
ax2.plot(temp_avg_field['frame_idx'], temp_avg_field['dy'], 'rs')
ax2.set_ylim([-2,0])
ax2.set_title('$\Delta y$')
ax2.set_xlabel('Timestep')
ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],
           scale=1, scale_units='xy', angles='xy')
ax3.set_ylim([-1,1])
# ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],scale=10)
ax3.set_title('$\Delta x$, $\Delta y$')
ax3.set_xlabel('Timestep');
```



0.9.5 Spatio-temporal Relationship

We can also divide the images into space and time steps

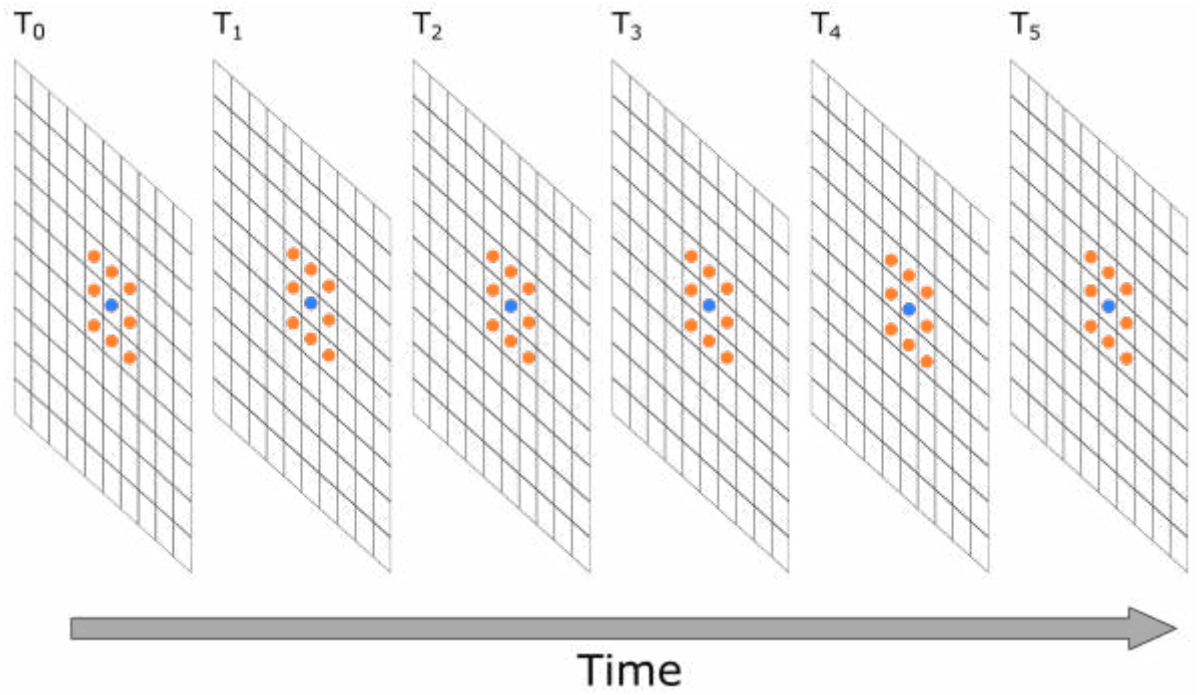


Fig. 3: Spatio-temporal averaging

Spatio-temporal averaging on moving cells

Find all items for a time frame and compute spatial average field.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 4),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 4))

frames = len(sorted(np.unique(average_field_df['frame_idx'])))
fig, m_axs = plt.subplots(2, 3, figsize=(12, 8))
for c_ax in m_axs.flatten():
    c_ax.axis('off')
[(ax1, ax2, _), (ax3, ax4, ax5)] = m_axs


def draw_frame_idx(idx):
    plt.cla()
    c_df = average_field_df[average_field_df['frame_idx'].isin([idx])]
    dx_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dx']))
    dy_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dy']))
    ps.heatmap(g_x,ax=ax1, bgbox=False,precision=0, fontsize=9)

    ax1.set_title('X Position')
    ps.heatmap(g_y,ax=ax2, bgbox=False,precision=0, fontsize=9)
    ax2.set_title('Y Position')

    ps.heatmap(dx_func(g_x, g_y).transpose(),ax=ax3, bgbox=False,precision=0,␣
↪fontsize=9)
    ax3.set_title('$\Delta x$')

    ps.heatmap(dy_func(g_x, g_y).transpose(),ax=ax4, bgbox=False,precision=0,␣
↪fontsize=9)
    ax4.set_title('$\Delta y$')

    ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
               scale=1.0, scale_units='xy', angles='xy')
    ax5.invert_yaxis()
    plt.suptitle('Frame {0}'.format(idx),fontsize=18)

# write animation frames
anim_code = FuncAnimation(fig,
                          draw_frame_idx,
                          frames=frames,
                          interval=1000,
                          repeat_delay=2000)

anim_code.save('movies/spatiotemporal_avg.mp4', fps=2)
plt.close('all')
```

Here, we look at the down sampled vector field for each frame. The NaNs you see in Δx and Δy are points outside the interpolation support area.

0.9.6 Longer Series

We see that this approach becomes problematic when we want to work with longer series

```
import pandas as pd
from skimage.morphology import label
from skimage.measure import regionprops
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, c_ax = plt.subplots(1, 1, figsize=(5, 5), dpi=150)

s_img = disk_img.copy()
s_img = np.roll(np.roll(s_img,4,axis=0),4,axis=1)
img_list = [s_img]
for i in range(8):
    if i % 2 == 0:
        s_img = np.roll(s_img, -2, axis=0)
    else:
        s_img = np.roll(s_img, -1, axis=1)
    img_list += [s_img]

all_objs = []
for frame_idx, c_img in enumerate(img_list):
    lab_img = label(c_img > 0)
    for c_obj in regionprops(lab_img):
        all_objs += [dict(label=int(c_obj.label),
                          y=c_obj.centroid[0],
                          x=c_obj.centroid[1],
                          area=c_obj.area,
                          frame_idx=frame_idx)]
all_obj_df2 = pd.DataFrame(all_objs)
all_obj_df2.head(5)


def update_frame(i):
    plt.cla()
    ps.heatmap(img_list[i],bgbox=False,precision=0,ax=c_ax,fontsize=9)
    c_ax.set_title('Iteration #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(img_list),
                          interval=1000,
                          repeat_delay=2000)

anim_code.save('movies/longseries.mp4', fps=2)
plt.close('all')
```

The resulting tracking

Tracking this longer series soon get complicated with the amount of direction changes in the scene.

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

fig, c_ax = plt.subplots(1, 1, figsize=(8, 8), dpi=200)
c_ax.matshow(disk_img > 1,cmap='gist_yarg')


def draw_timestep(i):
    # plt.cla()
    frame_0 = all_obj_df2[all_obj_df2['frame_idx'].isin([i])]
    frame_1 = all_obj_df2[all_obj_df2['frame_idx'].isin([i+1])]
    c_ax.scatter(frame_0['x'], frame_0['y'], c='black', label='Frame: %d' % i)
    c_ax.scatter(frame_1['x'], frame_1['y'],
                 c='red', label='Frame: %d' % (i+1))
    dist_df_list = []
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df2 = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df2.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        c_ax.quiver(best_row['x0'], best_row['y0'],
                    best_row['x1']-best_row['x0'],
                    best_row['y1']-best_row['y0'],
                    scale=1.0, scale_units='xy', angles='xy', alpha=0.25)
    c_ax.set_title('Frame #{}'.format(i+1))


# write animation frames
anim_code = FuncAnimation(fig,
                          draw_timestep,
                          frames=all_obj_df['frame_idx'].max(),
                          interval=1000,
                          repeat_delay=2000)
anim_code.save('movies/tracking_result2.mp4', fps=2)
plt.close('all')
```

```
from scipy.interpolate import interp2d
average_field = []
for i in range(all_obj_df2['frame_idx'].max()):
    frame_0 = all_obj_df2[all_obj_df2['frame_idx'].isin([i])]
    frame_1 = all_obj_df2[all_obj_df2['frame_idx'].isin([i+1])]
    dist_df_list = []
```

(continues on next page)

(continued from previous page)

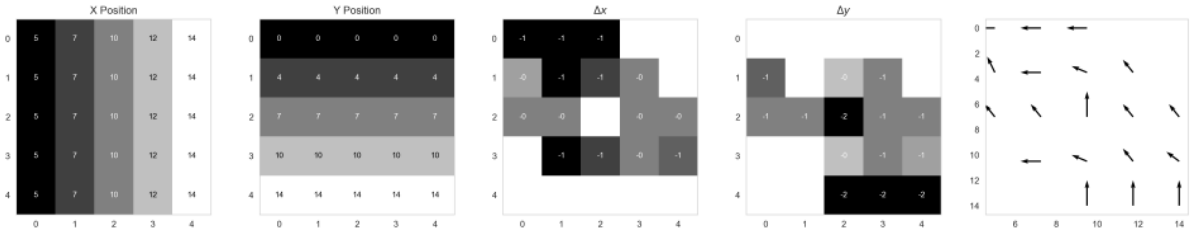
```
    for _, row_0 in frame_0.iterrows():
        for _, row_1 in frame_1.iterrows():
            dist_df_list += [dict(x0=row_0['x'],
                                  y0=row_0['y'],
                                  lab0=int(row_0['label']),
                                  x1=row_1['x'],
                                  y1=row_1['y'],
                                  lab1=int(row_1['label']),
                                  dist=np.sqrt(
                                      np.square(row_0['x']-row_1['x']) +
                                      np.square(row_0['y']-row_1['y'])))]
    dist_df = pd.DataFrame(dist_df_list)
    for _, c_rows in dist_df.groupby('lab0'):
        _, best_row = next(c_rows.sort_values('dist').iterrows())
        average_field += [dict(frame_idx=i,
                               x=best_row['x0'],
                               y=best_row['y0'],
                               dx=best_row['x1']-best_row['x0'],
                               dy=best_row['y1']-best_row['y0'])]
average_field_df2 = pd.DataFrame(average_field)
print('Average Flow:')
print(average_field_df2[['dx', 'dy']].mean())


def img_intp(f):
    def new_f(x, y):
        return np.stack([f(ix, iy) for ix, iy in zip(np.ravel(x), np.ravel(y))], 0).
 ↪reshape(np.shape(x))
    return new_f


dx_func = img_intp(
    LinearNDInterpolator((average_field_df2['x'], average_field_df2['y']), average_
 ↪field_df2['dx']))
dy_func = img_intp(
    LinearNDInterpolator((average_field_df2['x'], average_field_df2['y']), average_
 ↪field_df2['dy']))

g_x, g_y = np.meshgrid(np.linspace(average_field_df2['x'].min(),
                                   average_field_df2['x'].max(), 5),
                       np.linspace(average_field_df2['y'].min(),
                                   average_field_df2['y'].max(), 5))
fig, (ax1, ax2, ax3, ax4, ax5) = plt.subplots(1, 5, figsize=(24, 4))

ps.heatmap(g_x,ax=ax1, bgbox=False, precision=0, fontsize=9,cmap='gray')
ax1.set_title('X Position')

ps.heatmap(g_y,ax=ax2, bgbox=False, precision=0, fontsize=9,cmap='gray')
ax2.set_title('Y Position')


ps.heatmap(dx_func(g_x, g_y), ax=ax3, bgbox=False, precision=0, fontsize=9,cmap='gray
 ↪')
ax3.set_title('$\Delta x$')

ps.heatmap(dy_func(g_x, g_y), ax=ax4, bgbox=False, precision=0, fontsize=9,cmap='gray
```

(continues on next page)

(continued from previous page)

```
↪')
ax4.set_title('$\Delta y$')
ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
           scale=1.0, scale_units='xy', angles='xy');
ax5.invert_yaxis()
```

```
Average Flow:
dx   -0.5
dy   -1.0
dtype: float64
```

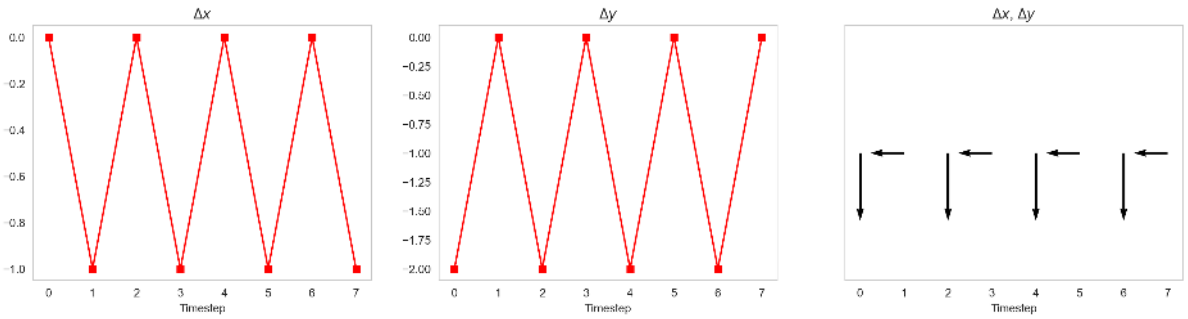


Temporal averaging of the long series

```
temp_avg_field = average_field_df2[['frame_idx', 'dx', 'dy']].groupby(
    'frame_idx').agg('mean').reset_index()
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(18, 4))
ax1.plot(temp_avg_field['frame_idx'], temp_avg_field['dx'], 'rs-')
ax1.set_title('$\Delta x$')
ax1.set_xlabel('Timestep')
ax2.plot(temp_avg_field['frame_idx'], temp_avg_field['dy'], 'rs-')
ax2.set_title('$\Delta y$')
ax2.set_xlabel('Timestep')

ax3.quiver(temp_avg_field['dx'], temp_avg_field['dy'],
           scale=10)
ax3.set_ylim([-1,1])
ax3.set_yticks([])

ax3.set_title('$\Delta x$, $\Delta y$')
ax3.set_xlabel('Timestep')
temp_avg_field
```

	frame_idx	dx	dy
0	0	0.0	-2.0
1	1	-1.0	0.0
2	2	0.0	-2.0
3	3	-1.0	0.0
4	4	0.0	-2.0
5	5	-1.0	0.0
6	6	0.0	-2.0
7	7	-1.0	0.0



Spatio-temporal averaging of the longer series

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML

g_x, g_y = np.meshgrid(np.linspace(average_field_df['x'].min(),
                                   average_field_df['x'].max(), 4),
                       np.linspace(average_field_df['y'].min(),
                                   average_field_df['y'].max(), 4))

frames = len(sorted(np.unique(average_field_df['frame_idx'])))
fig, m_axs = plt.subplots(2, 3, figsize=(14, 10))
for c_ax in m_axs.flatten():
    c_ax.axis('off')
#     c_ax.set(xticks=[],yticks=[])
[(ax1, ax2, _), (ax3, ax4, ax5)] = m_axs


def draw_frame_idx(idx):
    plt.cla()
    c_df = average_field_df2[average_field_df2['frame_idx'].isin([idx])]
    dx_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dx']))
    dy_func = img_intp(LinearNDInterpolator((c_df['x'], c_df['y']), c_df['dy']))
#     ps.heatmap(g_x, ax=ax1, annot=False, cbar=False)
    ps.heatmap(g_x,ax=ax1, bgbox=False, precision=0, fontsize=9,cmap='gray')
    ax1.set_title('Frame %d\nX Position' % idx)
#     ps.heatmap(g_y, ax=ax2, annot=False, cbar=False)
    ps.heatmap(g_y,ax=ax2, bgbox=False, precision=0, fontsize=9,cmap='gray')
    ax2.set_title('Y Position')

    ps.heatmap(dx_func(g_x, g_y).transpose(), ax=ax3, bgbox=False, precision=1,␣
 ↪fontsize=9,cmap='gray')
    ax3.set_title('$\Delta x$')
#     ps.heatmap(dy_func(g_x, g_y), ax=ax4, annot=False, cbar=False, fmt='2.1f')
    ps.heatmap(dy_func(g_x, g_y).transpose(), ax=ax4, bgbox=False, precision=1,␣
 ↪fontsize=9,cmap='gray')
    ax4.set_title('$\Delta y$')
    ax5.quiver(g_x, g_y, dx_func(g_x, g_y), dy_func(g_x, g_y),
               scale=1.0, scale_units='xy', angles='xy')
    ax5.invert_yaxis


# write animation frames
anim_code = FuncAnimation(fig,
```

(continues on next page)

(continued from previous page)

```
                            draw_frame_idx,
                            frames=frames,
                            interval=1000,
                            repeat_delay=2000)
anim_code.save('movies/spacecorrelation_animation.mp4', fps=2)
plt.close('all')
```

Tracking using Trackpy

Someone else did the job for you… trackpy

Tutorial

0.10 Problems with tracking

0.10.1 Random Appearance / Disappearance

Under perfect imaging and experimental conditions objects should not appear and reappear but due to

- Noise
- Limited fields of view / depth of field
- Discrete segmentation approaches
- Motion artifacts
  - Blurred objects often have lower intensity values than still objects
- Crossed paths

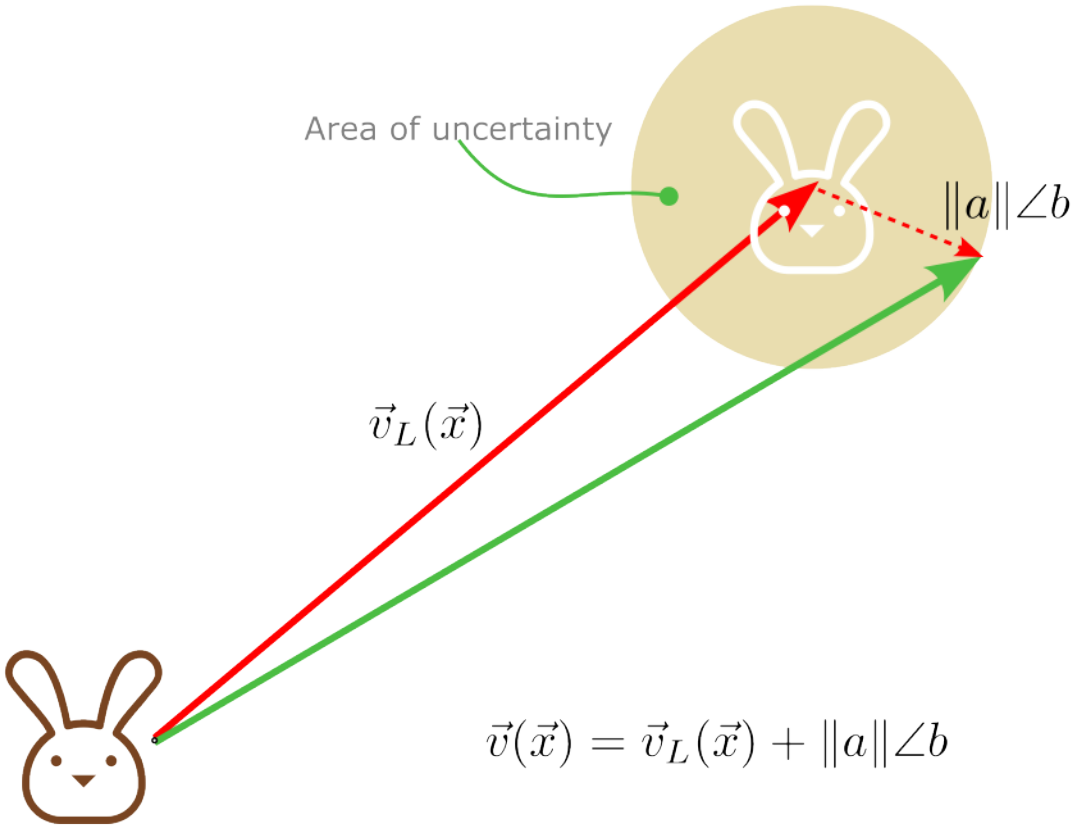
It is common for objects to appear and vanish regularly in an experiment.

0.10.2 Jitter / Motion Noise

Even perfect spherical objects do not move in a straight line:

- The jitter can be seen as a stochastic variable with a random
  - magnitude (a) and
  - angle (b).
- This is then sampled at every point in the field

$$\vec{v}(\vec{x}) = \underbrace{\vec{v}_L(\vec{x})}_{\text{deterministic}} + \underbrace{\|a\|\angle b}_{\text{random}}$$



There are several factor in the real experiment situation that can't be accounted for. The consequence is that the intended motion must be considered a random variable that has an expected velocity which is affected by the random effects. This random effect results in a region of uncertainty where the item is observed in the next time frame. Here, we model this as the random distance a in the direction b. I.e. the uncertainty region has a radius of $E[a]$.

0.10.3 Limitations of Tracking

We see that visually tracking samples can be difficult and there are a number of parameters which affect the ability for us to clearly see the tracking.

- flow rate
- flow type
- density
- appearance and disappearance rate
- jitter
- particle uniqueness

These experiment factors limit the accuracy and precision of the tracking. The sampling rate of the detector also plays a role here. In particular, it must match the flow rate and type of the process. The particle density and uniqueness makes it harder to identify the same particle between two frames.

0.11 How to improve tracking

We have to try to quantify the limits of these parameters for different tracking methods in order to design experiments better.

0.11.1 Acquisition-based Parameters

- Acquisition rate
  - flow rate vs. sampling rate
  - jitter (per frame)
- Resolution
  - density,
  - appearance rate

0.11.2 Experimental Parameters

- Experimental setup (pressure, etc) $\rightarrow$ flow rate/type
- Polydispersity $\rightarrow$ particle uniqueness
- Vibration/temperature $\rightarrow$ jitter
- Mixture $\rightarrow$ density contrast

0.11.3 Basic Simulations

Input flow from simulation

$$\vec{v}(\vec{x}) = \langle 0, 0, 0.05 \rangle + ||0.01||\measuredangle b$$

0.11.4 Designing Experiments

How do simulations help us to design experiments?

- density can be changed by adjusting the concentration of the substances being examined or the field of view
- flow per frame (image velocity) can usually be adjusted by changing pressure or acquisition time
- jitter can be estimated from images

How much is enough?

Difficult to create one number for every experiment

- 5% error in bubble position →
  - <5% in flow field
  - >20% error in topology
- 5% error in shape or volume →
  - 5% in distribution or changes
  - > 5% in individual bubble changes
  - > 15% for single bubble strain tensor calculations

0.11.5 Extending Nearest Neighbor

We have two time frames I_0 and I_1

Bijective Requirement

We define $\vec{P}_f$ as the result of performing the nearest neigbhor tracking on $\vec{P}_0$ $\vec{P}_f = \text{argmin}(||\vec{P}_0 - \vec{y}||\forall \vec{y} \in I_1)$$

We define $\vec{P}_i$ as the result of performing the nearest neigbhor tracking on $\vec{P}_f$ $\vec{P}_i = \text{argmin}(||\vec{P}_f - \vec{y}||\forall \vec{y} \in I_0)$$

We say the tracking is bijective if these two points are the same $\vec{P}_i \stackrel{?}{=} \vec{P}_0$$

Maximum Displacement

$$\vec{P}_1 = \begin{cases} ||\vec{P}_0 - \vec{y}|| < \text{MAXD}, & \text{argmin}(||\vec{P}_0 - \vec{y}||\forall \vec{y} \in I_1) \\ \text{Otherwise}, & \emptyset \end{cases}$$

0.11.6 Extending Nearest Neighbor (Continued)

Models of movement behavior to support tracking

Prior / Expected Movement

$$\vec{P}_1 = \text{argmin}(||\vec{P}_0 + \vec{v}_{offset} - \vec{y}||\forall \vec{y} \in I_1)$$

Adaptive Movement

Can then be calculated in an iterative fashion where the offset is the average from all of the $\vec{P}_1 - \vec{P}_0$ vectors. It can also be performed

$$\vec{P}_1 = \text{argmin}(||\vec{P}_0 + \vec{v}_{offset} - \vec{y}||\forall \vec{y} \in I_1)$$

More advanced models

- Use expected physical model
- Use track derivative to find v_{offset}
- Kalman filters can be used for particle tracking

0.11.7 Beyond Nearest Neighbor

While nearest neighbor

- provides a useful starting tool
- it is not sufficient for truly complicated flows and datasets.

Better Approaches

Multiple Hypothesis Testing

- Nearest neighbor just compares the points between two frames and there is much more information available in most time-resolved datasets.
- This approach allows for multiple possible paths to be explored at the same time and the best chosen only after all frames have been examined

Shortcomings

Merging and Splitting Particles

- The simplicity of the nearest neighbor model does really allow for particles to merge and split (relaxing the bijective requirement allows such behavior, but the method is still not suited for such tracking).
- For such systems a more specific, physically-based is required to encapsulate this behavior.

0.11.8 Voxel-based Approaches

For voxel-based approachs the most common analyses are digital image correlation (or for 3D images digital volume correlation), where the correlation is calculated between two images or volumes.

Standard Image Correlation

Given images $I_0(\vec{x})$ and $I_1(\vec{x})$ at time t_0 and t_1 respectively. The correlation between these two images can be calculated for each $\vec{r}$

$$C_{I_0,I_1}(\vec{r}) = \langle I_0(\vec{x}) I_1(\vec{x}+\vec{r}) \rangle$$

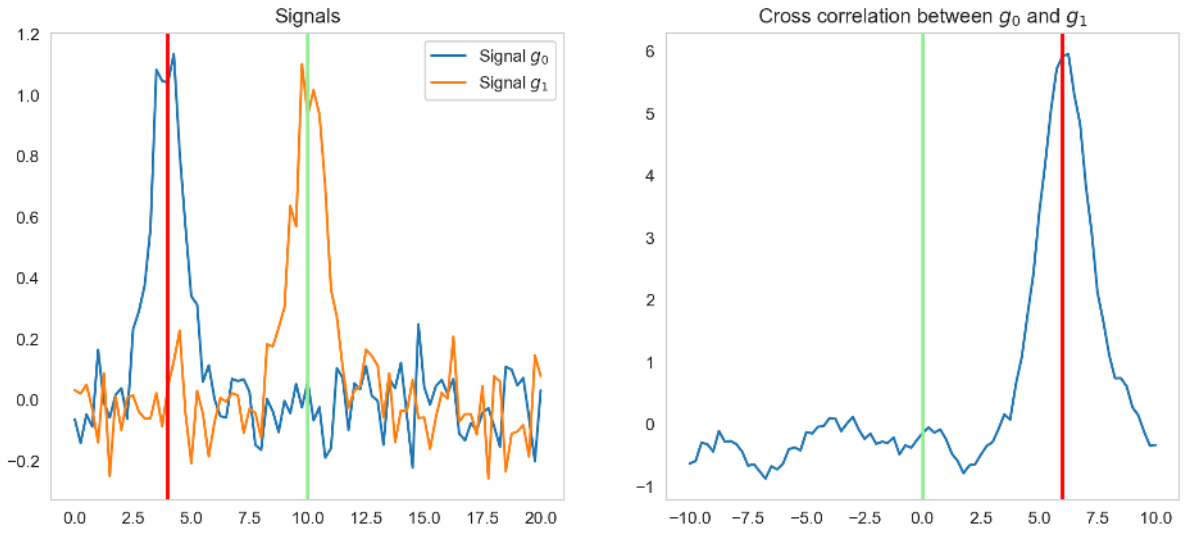
This can also be done in the Fourier space

$$C_{I_0,I_1}(\vec{r}) = \mathcal{F}^{-1}\{\mathcal{F}\{I_0\} \cdot \mathcal{F}\{I_1\}^*\}$$

Correlation in 1D

```
N=81
x=np.linspace(0,20,N)
g0=np.exp(-(x-4)**2)+np.random.normal(0,0.1,size=N)
g1=np.exp(-(x-10)**2)+np.random.normal(0,0.1,size=N)

fg0=np.fft.fft(g0)
fg1=np.fft.fft(g1)
fg0g1=fg1*np.conj(fg0)
cg0g1=np.real(np.fft.ifft(fg0g1))
```

```
fig,ax = plt.subplots(1,2,figsize=(12,5))
ax[0].plot(x,g0,label='Signal $g_0$')
ax[0].plot(x,g1,label='Signal $g_1$')
ax[0].legend()
ax[0].axvline(4,color='red',lw=2)
ax[0].axvline(10,color='lightgreen',lw=2)
ax[0].set_title('Signals')

ax[1].plot(x-10,np.fft.fftshift(cg0g1))
ax[1].axvline(0,color='lightgreen',lw=2)
ax[1].axvline(6,color='red',lw=2);
ax[1].set_title('Cross correlation between $g_0$ and $g_1$');
```

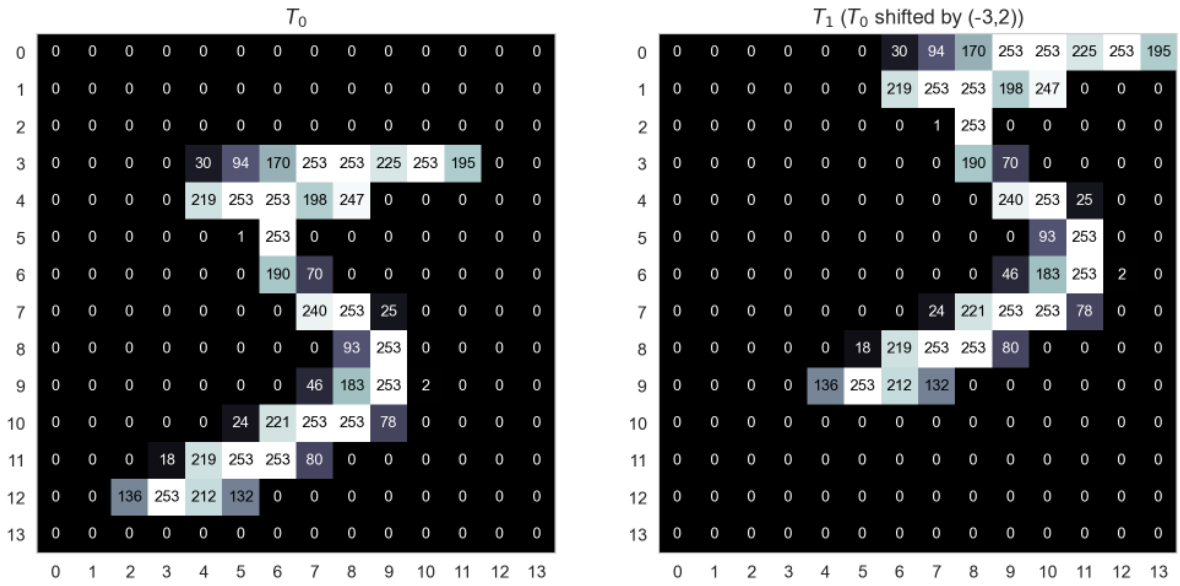


Let's make some test data

We use a 'five' from the MNIST data set of handwritten numbers

```
bw_img = np.load('data/five.npy') # A 'five' from the mnist data set

shift_img = np.roll(np.roll(bw_img, -3, axis=0), 2, axis=1) # Shifting image by (dx,
↪dy)=(2,-3) using rolling
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))
# sns.heatmap(bw_img, ax=ax1, cbar=False, annot=True, fmt='d', cmap='bone')
ps.heatmap(bw_img, ax=ax1, bgbox=False, precision=0,cmap='bone', fontsize=9)
ax1.set_title('$T_0$')
# sns.heatmap(shift_img, ax=ax2, cbar=False, annot=True, fmt='d', cmap='bone') ,
ps.heatmap(shift_img, ax=ax2, bgbox=False, precision=0,cmap='bone', fontsize=9)
ax2.set_title('$T_1$ ($T_0$ shifted by (-3,2))');
```



Demonstration of the correlation in space

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5),dpi=200)

mx, my = np.meshgrid(np.arange(-4, 6, 2),
                     np.arange(-4, 6, 2))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)


def update_frame(i):
    a_img = bw_img
```

(continues on next page)

(continued from previous page)

```
    b_img = np.roll(np.roll(shift_img, nx[i], axis=1), ny[i], axis=0)
    ax1.cla()
#     sns.heatmap(a_img, ax=ax1, cbar=False, annot=True, fmt='0.0f', cmap='bone')
    ps.heatmap(a_img,ax=ax1,bgbox=False, precision=0,fontsize=9, cmap='bone')
    ax2.cla()
#     sns.heatmap(b_img, ax=ax2, cbar=False, annot=True, fmt='0.0f', cmap='bone')
    ps.heatmap(b_img,ax=ax2,bgbox=False, precision=0,fontsize=9, cmap='bone')

    out_score[i] = np.mean(a_img*b_img)
    ax3.cla()
#     sns.heatmap(out_score.reshape(mx.shape), ax=ax3,
#                 cbar=False, annot=True, fmt='2.1f', cmap='viridis')
    ps.heatmap(out_score.reshape(mx.shape),ax=ax3,bgbox=False, precision=1,fontsize=9,
 ↪cmap='viridis',vmin=0, vmax=15)
    ax3.set_xticklabels(mx[0, :])
    ax3.set_yticklabels(my[:, 0])
    ax1.set_title('Iteration #{}'.format(i+1))
    ax2.set_title('X-Offset: {0}\nY-Offset: {1}'.format(nx[i], ny[i]))
    ax3.set_title(r'$\langle I_0(\vec{x}) I_1(\vec{x}+\vec{r}) \rangle$')


# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(nx),
                          interval=300,
                          repeat_delay=4000)

anim_code.save('movies/spacecorrelation_animation_five.mp4', fps=5)
plt.close('all')
```

0.11.9 Other metrics

Mean Squared Error

We can also use

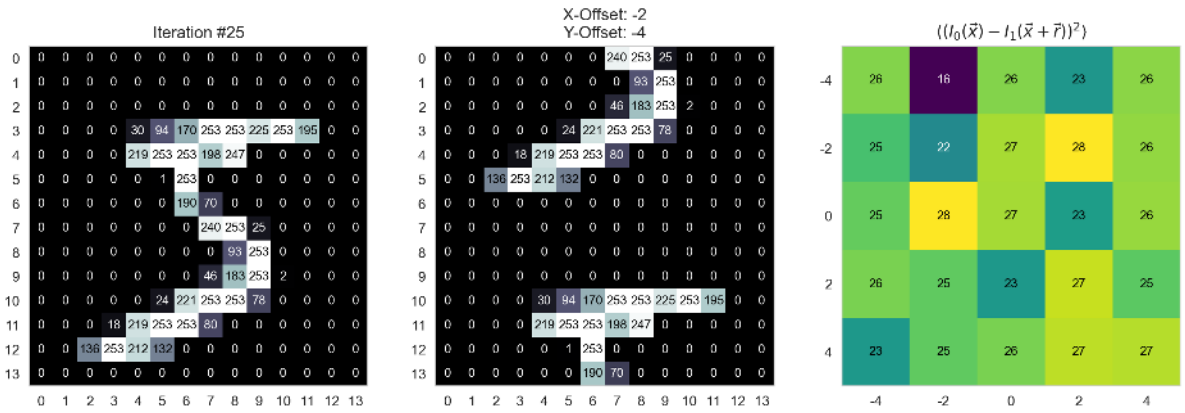
- MSE
- or RMSE

and look for minima

Testing MSE

```
out_score = np.zeros(nx.shape, dtype=np.float32)

for i in range(len(nx)):
    a_img = bw_img
    b_img = np.roll(np.roll(shift_img, nx[i], axis=1), ny[i], axis=0)
    out_score[i] = np.mean(np.square(a_img-b_img))


# get the minimum
i_min = np.argmin(out_score)
b_img = np.roll(np.roll(shift_img, nx[i_min], axis=1), ny[i_min], axis=0)
```

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 6))

ps.heatmap(a_img, ax=ax1, bgbox=False, precision=0, fontsize=9, cmap='bone')
ps.heatmap(b_img, ax=ax2, bgbox=False, precision=0, fontsize=9, cmap='bone')
ps.heatmap(out_score.reshape(mx.shape), ax=ax3, bgbox=False,
           precision=0, fontsize=9, cmap='viridis')

ax3.set_xticklabels(mx[0, :])
ax3.set_yticklabels(my[:, 0])
ax1.set_title('Iteration #{}'.format(i+1))
ax2.set_title('X-Offset: %d\nY-Offset: %d' % (nx[i_min], ny[i_min]))
ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');
```



0.11.10 Correlation using the Fourier transform

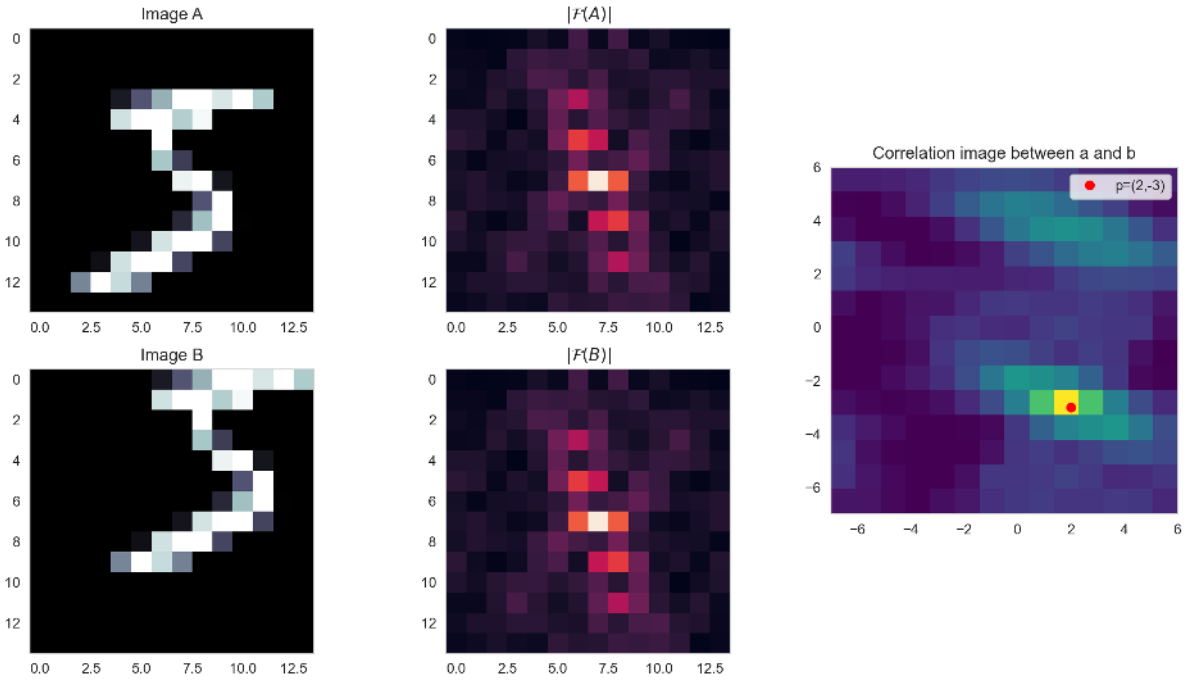
$$C_{I_0,I_1}(\vec{r}) = \mathcal{F}^{-1}\{\mathcal{F}\{I_0\} \cdot \mathcal{F}\{I_1\}^*\}$$

```
plt.figure(figsize=(15,8))
plt.subplot(2,3,1); plt.imshow(bw_img,cmap='bone'); plt.title('Image A')
plt.subplot(2,3,4); plt.imshow(shift_img,cmap='bone'); plt.title('Image B')

fa=(np.fft.fft2(bw_img));
fb=(np.fft.fft2(shift_img));

plt.subplot(2,3,2); plt.imshow(np.abs(np.fft.fftshift(fa))); plt.title('$|\mathcal{F}
```

(continues on next page)

(continued from previous page)

```
 ↪(A)|$')
plt.subplot(2,3,5); plt.imshow(np.abs(np.fft.fftshift(fb))); plt.title('$|\mathcal{F}
 ↪(B)|$')

f=fb*np.conjugate(fa);
co=np.abs(np.fft.fftshift(np.fft.ifft2(f)));
plt.subplot(1,3,3)
plt.imshow(np.abs(co), extent = [-7 , 6, -7 , 6], cmap='viridis',origin='lower');
plt.title('Correlation image between a and b');
plt.plot(2,-3,'ro',label='p=(2,-3)')
plt.legend();
```



0.11.11 Real Examples

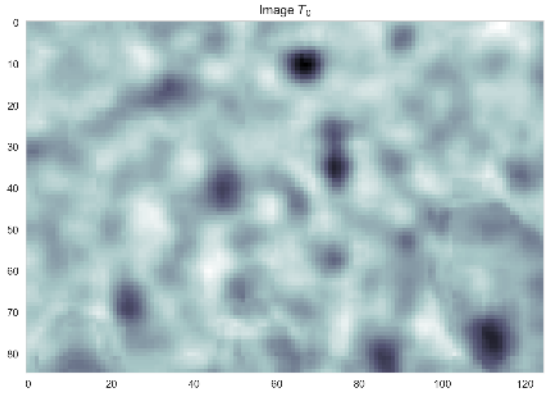
Bone Slice Registration

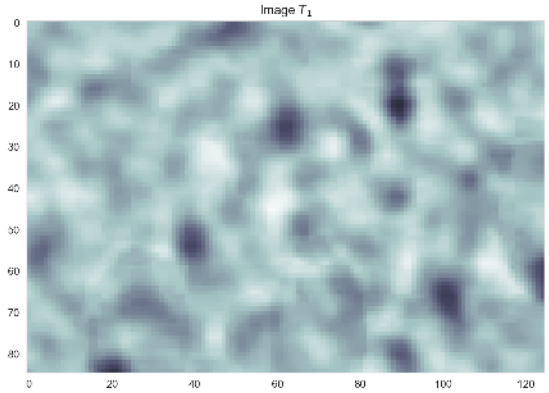
```
import numpy as np
from skimage.filters import median
import seaborn as sns
import matplotlib.pyplot as plt
from skimage.io import imread
%matplotlib inline
full_img = imread("figures/bonegfiltslice.png").mean(axis=2)
full_shift_img = median(
    np.roll(np.roll(full_img, -15, axis=0), 15, axis=1), np.ones((1, 3)))

def g_roi(x): return x[5:90, 150:275]

bw_img = g_roi(full_img)
```

(continues on next page)

(continued from previous page)

```
shift_img = g_roi(full_shift_img)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(20, 6), dpi=100)
ax1.imshow(bw_img, cmap='bone'),ax1.set_title('Image $T_0$')
ax2.imshow(shift_img, cmap='bone',vmin=bw_img.min(),vmax=bw_img.max()),ax2.set_title(
↪'Image $T_1$');
```





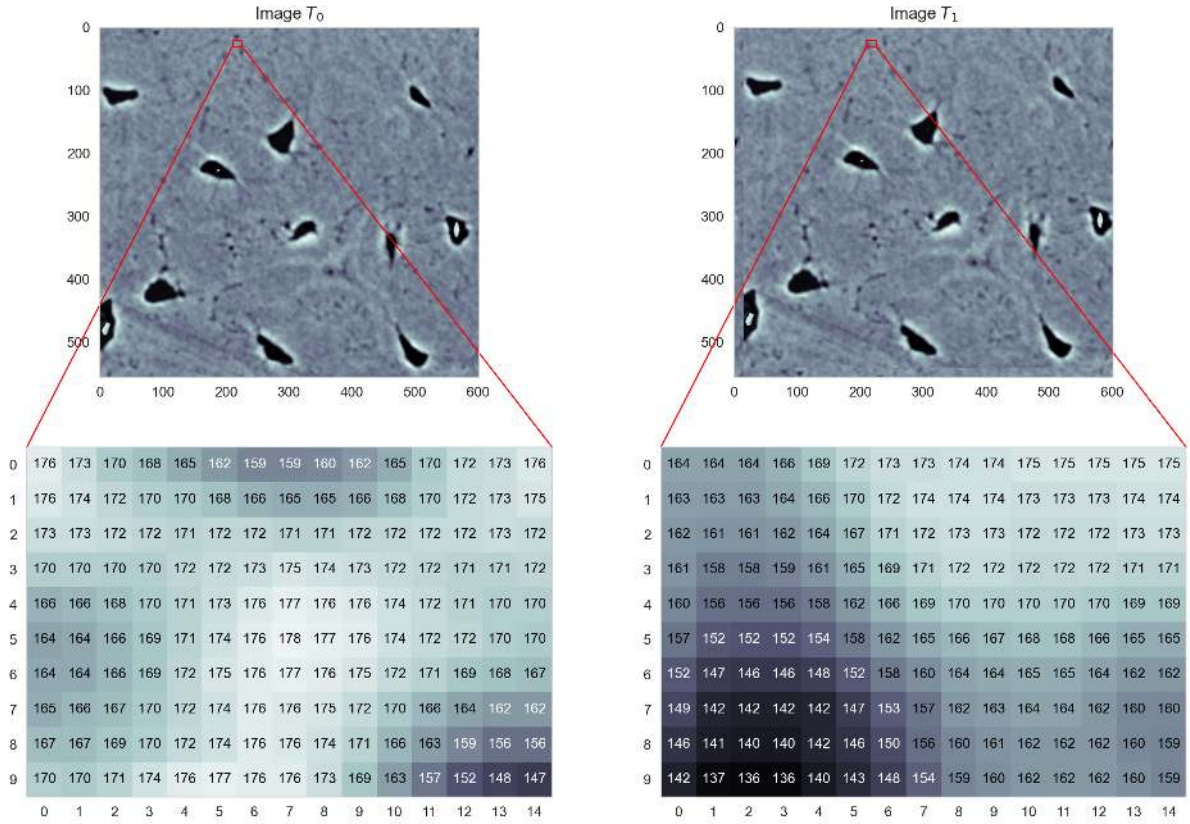
Let's look at a smaller region

```
fig, ((ax1, ax2),(ax3,ax4)) = plt.subplots(2, 2, figsize=(15, 10))

def g_roi(x): return x[20:30, 210:225]

ax1.imshow(full_img,cmap='bone')
ps.heatmap(g_roi(full_img), ax=ax3, precision=0, bgbox=False, cmap='bone',fontsize=10,
↪vmin=135,vmax=180),
ax1.set_title('Image $T_0$')

rect=Rectangle((210,20),height=10,width=15,color='r',fc='none')
ax1.add_patch(rect)
con1 = ConnectionPatch(xyA=(210,30), xyB=(-0.5,-0.5), coordsA="data", coordsB="data",
                       axesA=ax1, axesB=ax3, color="red", lw=1)
ax1.add_artist(con1);
con2 = ConnectionPatch(xyA=(225,30), xyB=(14.5,-0.5), coordsA="data", coordsB="data",
                       axesA=ax1, axesB=ax3, color="red", lw=1)

ax1.add_artist(con2)




ax2.imshow(full_shift_img,cmap='bone')
ps.heatmap(g_roi(full_shift_img), ax=ax4, precision=0, bgbox=False, cmap='bone',
↪fontsize=10,vmin=135,vmax=180);
ax2.set_title('Image $T_1$');
```

(continues on next page)

(continued from previous page)

```
rect=Rectangle((210,20),height=10,width=15,color='r',fc='none')
ax2.add_patch(rect)
con3 = ConnectionPatch(xyA=(210,30), xyB=(-0.5,-0.5), coordsA="data", coordsB="data",
                       axesA=ax2, axesB=ax4, color="red", lw=1)
ax2.add_artist(con3);
con4 = ConnectionPatch(xyA=(225,30), xyB=(14.5,-0.5), coordsA="data", coordsB="data",
                       axesA=ax2, axesB=ax4, color="red", lw=1)

ax2.add_artist(con4);
```



Computing the correlation sum for each pixel

```
from matplotlib.animation import FuncAnimation
from IPython.display import HTML
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,5),dpi=200)

def g_roi(x): return x[20:30:2, 210:225:2]

mx, my = np.meshgrid(np.arange(-10, 12, 4),
                     np.arange(-10, 12, 4))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)
```

(continues on next page)

(continued from previous page)

```
def update_frame(i):
    a_img = g_roi(full_img)
    b_img = g_roi(np.roll(np.roll(full_shift_img, nx[i], axis=1), ny[i], axis=0))
    ax1.cla(),
#     sns.heatmap(a_img, ax=ax1, cbar=False, annot=True, fmt='0.0f', cmap='bone')
    ps.heatmap(a_img,ax=ax1,bgbox=False,cmap='bone',fontsize=10,precision=0)
    ax2.cla(),
#     sns.heatmap(b_img, ax=ax2, cbar=False, annot=True, fmt='0.0f', cmap='bone')
    ps.heatmap(b_img,ax=ax2,bgbox=False,cmap='bone',fontsize=10,precision=0)
    out_score[i] = np.mean(np.square(a_img-b_img))
    ax3.cla(),
#     sns.heatmap(out_score.reshape(mx.shape), ax=ax3, cbar=False, annot=True, fmt='2.
 ↪1f', cmap='RdBu')
    ps.heatmap(out_score.reshape(mx.shape),ax=ax3,bgbox=False,cmap='RdBu',fontsize=10,
 ↪precision=1,vmin=0.0,vmax=420.0)
    ax1.set_title('Iteration #{}'.format(i+1))
    ax2.set_title('X-Offset: %d\nY-Offset: %d' % (2*nx[i], 2*ny[i]))
    ax3.set_xticklabels(mx[0, :])
    ax3.set_yticklabels(my[:, 0])
    ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');

# write animation frames
anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(nx),
                          interval=300,
                          repeat_delay=2000)
anim_code.save('movies/spacecorrelation_bone.mp4', fps=5)
plt.close('all')
```

Checking the results

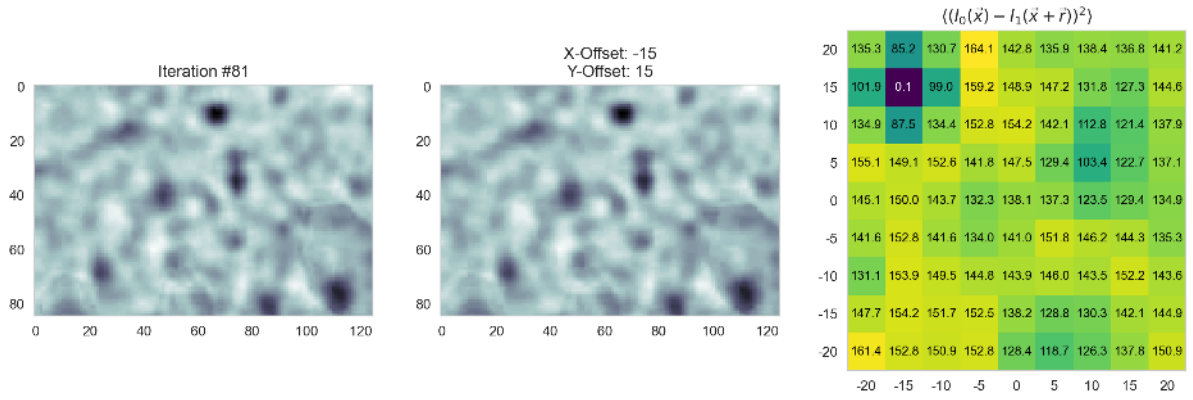
The analysis resulted in a minimum at displacement $\Delta x = -15$, $\Delta y = 15$

```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15, 5))


mx, my = np.meshgrid(np.arange(-20, 25, 5),
                     np.arange(-20, 25, 5))

nx = mx.ravel()
ny = my.ravel()
out_score = np.zeros(nx.shape, dtype=np.float32)

out_score = np.zeros(nx.shape, dtype=np.float32)


def g_roi(x): return x[5:90, 150:275]


for i in range(len(nx)):
    a_img = g_roi(full_img)
```

(continues on next page)

(continued from previous page)

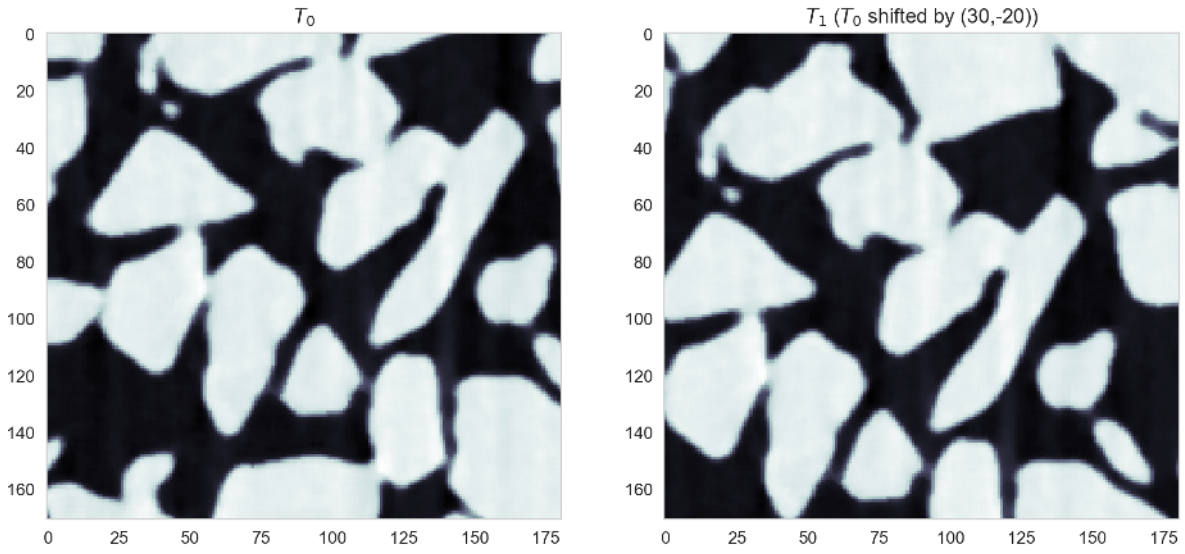
```
    b_img = g_roi(
        np.roll(np.roll(full_shift_img, nx[i], axis=1), ny[i], axis=0))
    out_score[i] = np.mean(np.square(a_img-b_img))

# get the minimum
i_min = np.argmin(out_score)
b_img = g_roi(np.roll(np.roll(full_shift_img, nx[i_min], axis=1), ny[i_min], axis=0))
ax1.imshow(a_img, cmap='bone'), ax1.set_title('$T_0$')
ax2.imshow(b_img, cmap='bone'), ax2.set_title('$T_1$ Registered')
# sns.heatmap(out_score.reshape(mx.shape), ax=ax3, cbar=False,annot=True, fmt='2.1f',␣
 ↪cmap='viridis')
ps.heatmap(out_score.reshape(mx.shape), ax=ax3, precision=1, fontsize=9, bgbox=False,␣
 ↪cmap='viridis')
ax3.invert_yaxis()
ax3.set_xticklabels(mx[0, :]), ax3.set_yticklabels(my[:, 0])
ax1.set_title('Iteration #{}'.format(i+1))
ax2.set_title('X-Offset: %d\nY-Offset: %d' % (nx[i_min], ny[i_min]))
ax3.set_title(r'$\langle (I_0(\vec{x})-I_1(\vec{x}+\vec{r}))^2 \rangle$');
```



Sand grains

```
bw_sand = plt.imread('data/sand.png').astype(float) # Some sand grains

shift_sand = bw_sand[:-30,20:]
bw_sand = bw_sand[30:,:shift_sand.shape[1]]
```

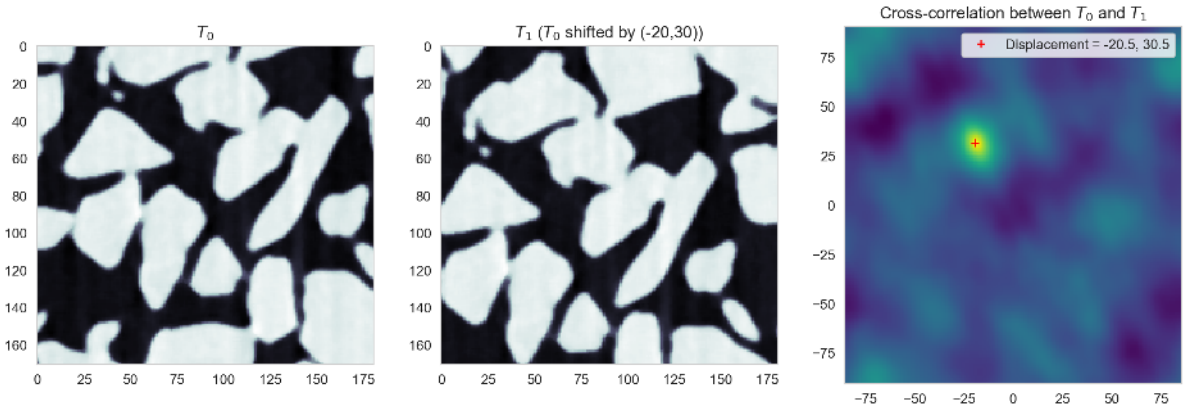
```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6), dpi=100)
ax1.imshow(bw_sand,cmap='bone'), ax1.set_title('$T_0$')
ax2.imshow(shift_sand, cmap='bone') , ax2.set_title('$T_1$ ($T_0$ shifted by (30,-20))
 ↪');
```



Correlation in Fourier space

```
fg0=np.fft.fft2(bw_sand)
fg1=np.fft.fft2(shift_sand)
fg0g1=fg1*np.conj(fg0)
cg0g1=np.real(np.fft.fftshift(np.fft.ifft2(fg0g1)))

pos=np.unravel_index(np.argmax(cg0g1[::-1]),cg0g1.shape)
x=pos[1]-cg0g1.shape[1]/2
y=-(pos[0]-cg0g1.shape[0]/2)
```

```
fig,ax=plt.subplots(1,3,figsize=(15,5))
ax[0].imshow(bw_sand,cmap='bone'), ax[0].set_title('$T_0$')
ax[1].imshow(shift_sand, cmap='bone') , ax[1].set_title('$T_1$ ($T_0$ shifted by (-20,
↪30))');

ax[2].imshow(cg0g1[::-1],cmap='viridis',extent=[-cg0g1.shape[0]/2,cg0g1.shape[0]/2,-
↪cg0g1.shape[1]/2,cg0g1.shape[1]/2])
ax[2].plot(x+1,y+1,'r+',label='Displacement = {0}, {1}'.format(x,y));
ax[2].legend();
ax[2].set_title('Cross-correlation between $T_0$ and $T_1$');
```



0.12 Registration

Before any meaningful tracking tasks can be performed, the first step is to register the measurements so they are all on the same coordinate system.

Often the registration can be done along with the tracking by separating the movement into actual sample movement and other (camera, setup, etc) if the motion of either the sample or the other components can be well modeled.

In medicine this is frequently needed because different scanners produce different kinds of outputs with different scales, positioning and resolutions. This is also useful for 'follow-up' scans with patients to identify how a disease has progressed. With scans like chest X-rays it isn't uncommon to have multiple (some patients have hundreds) all taken under different conditions

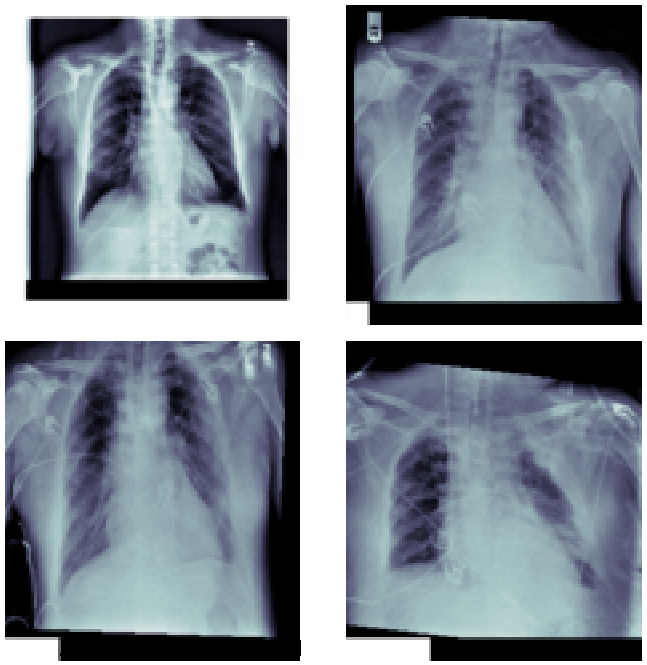


Fig. 1: A series of chest X-ray images taken at different times.

0.12.1 The Process

We informally followed a process before when trying to match the two images together, but we want to make this more generic for a larger spectrum of problems.

We thus follow the model set forward by tools like ITK with the components divided into the input data:

- *Moving Image*
- and *Fixed Image* sometimes called *Reference Image*).

The algorithmic components

- The *Transform* operation to transform the moving image.
- The *interpolator* to handle bringing all of the points onto a pixel grid.
- The *Metric* which is the measure of how well the transformed moving image and fixed image match
- and finally the *Optimizer* that tries to find the best solution

```
from IPython.display import SVG
from subprocess import check_output
import pydot
import os


def show_graph(graph):
    try:
        return SVG(graph.create_svg())
    except AttributeError as e:
        output = check_output('dot -Tsvg', shell=True,
                              input=g.to_string().encode())
        return SVG(output.decode())


g = pydot.Graph(graph_type='digraph')
fixed_img = pydot.Node('Fixed Image\nReference Image',
                       shape='folder', style="filled", fillcolor="lightgreen")
moving_img = pydot.Node('Moving Image', shape='folder',
                        style="filled", fillcolor="lightgreen")
trans_obj = pydot.Node('Transform', shape='box',
                       style='filled', fillcolor='yellow')
g.add_node(fixed_img)
g.add_node(moving_img)
g.add_node(trans_obj)
g.add_edge(pydot.Edge(fixed_img, 'Metric'))
g.add_edge(pydot.Edge(moving_img, 'Interpolator'))
g.add_edge(pydot.Edge(trans_obj, 'Interpolator', label='Transform Parameters'))
g.add_edge(pydot.Edge('Interpolator', 'Metric'))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

An automatic registration algorithm

```
g.add_edge(pydot.Edge('Metric', 'Optimizer'))
g.add_edge(pydot.Edge('Optimizer', trans_obj))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

0.12.2 Images

Fixed Image

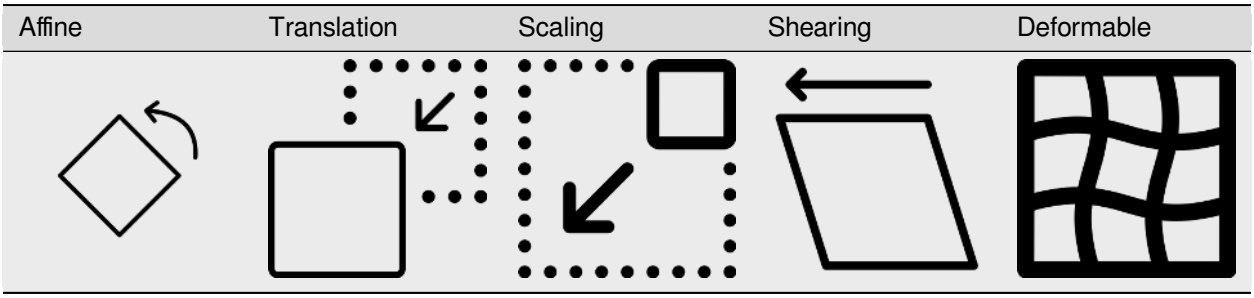
The fixed image (or reference image) is the image that will be left untouched and used for comparison

Moving Image

The moving image will be transformed (translated, scaled, rotated, deformed, …) to try and match as closely as possible the fixed image.

0.12.3 Transform

The transform specifies the transformations which can take place on the moving image, a number of different types are possible, but the most frequent types are listed below.



0.12.4 Interpolator

The interpolator is the component applies the transform to the moving image. The common ways of interpolating are

- Nearest Neighbor
- Bilinear
- Bicubic
- Bspline
- …

0.12.5 Metric

The metric is how the success of the matching of the two images is measured. The goal is to measure similarity between images.

- Mean Squared Error - the simplest metric to use just recording the raw difference, but often this can lead to unusual matches since noise and uneven illumination can lead to high MSE for images that match well.
- SSIM similarity metric
- Correlation Factor

0.12.6 Optimizer

The optimizer component is responsible for updating the parameters based on the metric. A standard approach with this is gradient descent where the gradient is calculated and a small step (determined by the learning rate) is taken in the direction of maximum descent.

- Gradient Descent
- Adam
- Stochastic Gradient Descent
- AdaGrad
- AdaDelta

0.12.7 Our tracker

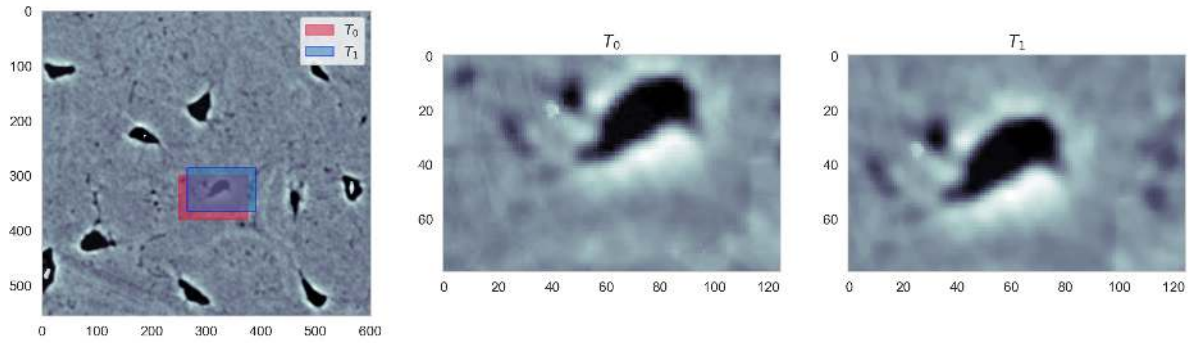
```
from IPython.display import Image,SVG

g = pydot.Dot(graph_type='digraph')
fixed_img = pydot.Node('Fixed Image\nReference Image',
                       shape='folder', style="filled", fillcolor="lightgreen")
moving_img = pydot.Node('Moving Image', shape='folder',
                        style="filled", fillcolor="lightgreen")
trans_obj = pydot.Node('Transform', shape='box',
                       style='filled', fillcolor='yellow')
g.add_node(fixed_img)
g.add_node(moving_img)
g.add_node(trans_obj)
g.add_edge(pydot.Edge(fixed_img, 'Metric\nMean Squared Error'))
g.add_edge(pydot.Edge(moving_img, 'Interpolator\nNearest Neighbor'))
g.add_edge(pydot.Edge(trans_obj, 'Interpolator\nNearest Neighbor',
                      label='Transform Parameters'))
g.add_edge(pydot.Edge('Interpolator\nNearest Neighbor',
                      'Metric\nMean Squared Error'))
#g.add_edge(pydot.Edge('Metric\nMean Squared Error', 'Optimizer\nGrid Search', style␣
↪= ''))
g.add_edge(pydot.Edge('Optimizer\nGrid Search', trans_obj))
show_graph(g)
```

```
<IPython.core.display.SVG object>
```

0.12.8 Registration of the bone image

```
import tensorflow as tf
import numpy as np
from skimage.filters import median
import matplotlib.pyplot as plt
from matplotlib.patches import Rectangle
from skimage.io import imread
%matplotlib inline

full_img = imread("figures/bonegfiltslice.png")[:,:,0:3].mean(axis=2)
dx,dy = -15,15
full_shift_img = median(
    np.roll(np.roll(full_img, dy, axis=0), dx, axis=1), footprint=np.ones((3, 3)))

roi = [300,250,380,375]
def g_roi(x): return x[roi[0]:roi[2],roi[1]:roi[3]]
```

```
a = g_roi(full_img)
b = g_roi(full_shift_img)
fig, (ax0,ax1, ax2) = plt.subplots(1, 3, figsize=(15, 4))
ax0.imshow(full_img,cmap='bone')
rect=Rectangle((roi[1],roi[0]),roi[3]-roi[1],roi[2]-roi[0],ec='crimson',fc='crimson',
 ↪alpha=0.5,label=r"$T_0$")
ax0.add_patch(rect)
rect=Rectangle((roi[1]+dy,roi[0]+dx),roi[3]-roi[1],roi[2]-roi[0],ec='blue',alpha=0.5,
 ↪label=r"$T_1$")
ax0.add_patch(rect)
ax0.legend()
ax1.imshow(a, cmap='bone')
ax1.set_title('$T_0$')
ax2.imshow(b, cmap='bone')
ax2.set_title('$T_1$');
```



Convert images to TensorFlow

```
# Convert NumPy arrays to tf.Tensor
fixed_image = tf.convert_to_tensor(g_roi(full_img), dtype=tf.float32)
moving_image = tf.convert_to_tensor(g_roi(full_shift_img), dtype=tf.float32)

# If they are 2D (grayscale), add channel dimension → [H, W, 1]
if fixed_image.ndim == 2:
    fixed_image = tf.expand_dims(fixed_image, axis=-1)

if moving_image.ndim == 2:
    moving_image = tf.expand_dims(moving_image, axis=-1)
```

```
2025-04-16 10:49:53.260289: I metal_plugin/src/device/metal_device.cc:1154] Metal␣
↪device set to: Apple M1 Max
2025-04-16 10:49:53.260327: I metal_plugin/src/device/metal_device.cc:296]␣
↪systemMemory: 32.00 GB
2025-04-16 10:49:53.260330: I metal_plugin/src/device/metal_device.cc:313]␣
↪maxCacheSize: 10.67 GB
WARNING: All log messages before absl::InitializeLog() is called are written to␣
↪STDERR
I0000 00:00:1744793393.260949 33884209 pluggable_device_factory.cc:305] Could not␣
↪identify NUMA node of platform GPU ID 0, defaulting to 0. Your kernel may not␣
↪have been built with NUMA support.
I0000 00:00:1744793393.261166 33884209 pluggable_device_factory.cc:271] Created␣
↪TensorFlow device (/job:localhost/replica:0/task:0/device:GPU:0 with 0 MB␣
↪memory) -> physical PluggableDevice (device: 0, name: METAL, pci bus id:
↪<undefined>)
```

Transformation and cost functions

```
import regtools as rt
import importlib

importlib.reload(rt)

def affine_transform(fixed_img, moving_img, transform_params):
    # Define affine transform matrix
    theta = transform_params[0]
    tx    = transform_params[1]
    ty    = transform_params[2]

    transformed=rt.affine_transform_tf(moving_img,theta=theta,tx=tx,ty=ty)
    return transformed

def similarity_loss(fixed_img, moving_img, transform_params):
    # Apply affine transform to moving image
    transformed_img = affine_transform(fixed_img, moving_img, transform_params)

    # Compute mean squared error between fixed and transformed moving images
    mse = tf.reduce_mean(tf.square(fixed_img - transformed_img))

    return mse
```

The registration iterations

```
# Define the transform parameters as trainable variables
theta = tf.Variable(0.0, dtype=tf.float32, trainable=True)
tx    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
ty    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
transform_params = [theta, tx, ty]

# Define the optimizer
optimizer = tf.optimizers.Adam(learning_rate=0.2)

# Define the training loop
N = 150
losses = []
frames = []
pars   = []
for i in range(N):
    with tf.GradientTape() as tape:
        tape.watch(transform_params)
        loss = similarity_loss(fixed_image, moving_image, transform_params)

    gradients = tape.gradient(loss, transform_params)

    optimizer.apply_gradients(zip(gradients, transform_params))

    losses.append(loss.numpy())
    pars.append(np.array([p.numpy() for p in transform_params], dtype=np.float32))
    frames.append(affine_transform(fixed_image, moving_image, transform_params))

# Apply the learned transform to the moving image
registered_image = affine_transform(fixed_image, moving_image, transform_params)
```

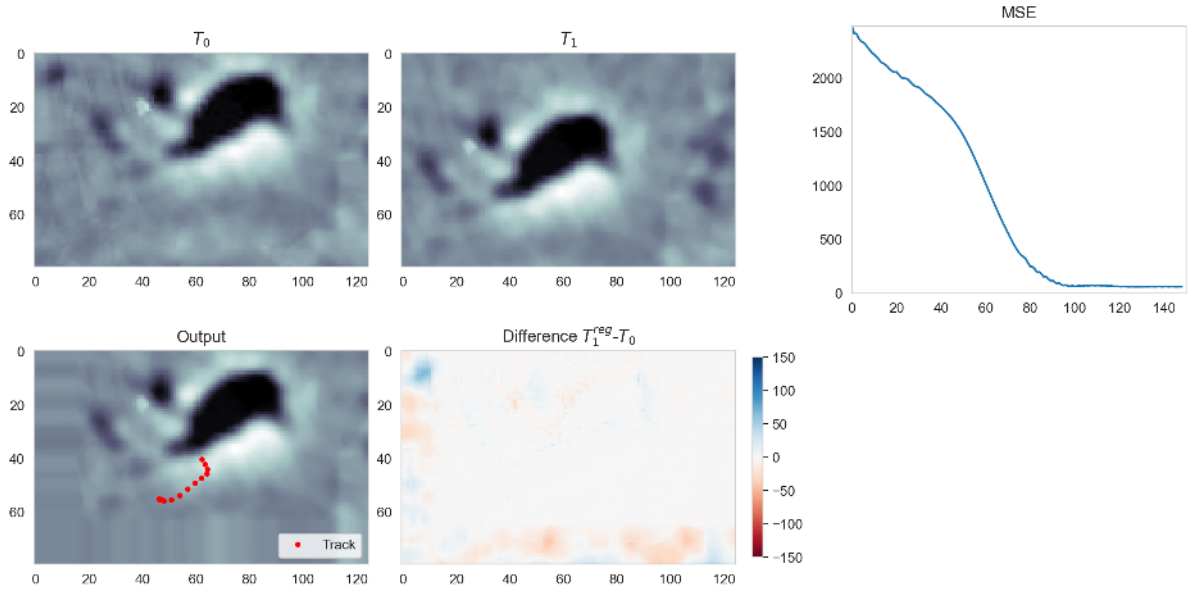
Showing the registration result

```
def show_registration(fixed_image,moving_image,frames,pars,losses) :
    pars1 = np.stack(pars)
    fig, m_axs = plt.subplots(2, 3, figsize=(12, 6), layout='constrained', gridspec_
 ↪kw={"height_ratios": [1,1]})
    (ax1, ax2, ax5), (ax3, ax4, ax6) = m_axs

    a4=ax4.imshow(frames[i]-fixed_image,cmap='RdBu',vmin=-150, vmax=150)
    ax6.cla()
    ax6.axis('off')
    cbar=fig.colorbar(a4,ax=ax4,shrink=0.75)

    ax1.imshow(fixed_image, cmap='bone')
    ax1.set_title('$T_0$')
    ax2.imshow(moving_image, cmap='bone')
    ax2.set_title('$T_1$')
    ax3.imshow(frames[i], cmap='bone')
    ax3.set_title('Output')

    ax5.plot(losses[:i])
    ax5.set_xlim([0,N])
    ax5.set_ylim([0,np.max(losses)])
```

(continues on next page)

(continued from previous page)

```
    ax5.set_title("MSE")

    im=ax4.imshow(frames[i]-fixed_image,cmap='RdBu',vmin=-150, vmax=150)
    ax4.set_title('Difference $T_1^{reg}$-$T_0$')
    # Update the colorbar mappable
    a4.set_data(frames[i] - fixed_image)

    ax3.plot(pars1[::10,1]+fixed_image.shape[1]//2,pars1[::10,2]+fixed_image.shape[0]/
↪/2,'r.', label="Track");
    ax3.legend(loc="lower right");

show_registration(fixed_image,moving_image,frames,pars,losses)
```



Animating the registration process

```
from matplotlib.animation import FuncAnimation
from matplotlib import patches

pars1 = np.stack(pars)

fig, m_axs = plt.subplots(2, 3, figsize=(15, 6), layout='constrained')
(ax1, ax2, ax5), (ax3, ax4, ax6) = m_axs

a4=ax4.imshow(frames[i]-fixed_image,cmap='RdBu',vmin=-150, vmax=150)
ax6.cla()
ax6.axis('off')
cbar=fig.colorbar(a4,ax=ax4,shrink=0.75)


def update_frame(i):

    for c_ax in m_axs.flatten():
        c_ax.cla()
```

(continues on next page)

(continued from previous page)

```
    ax1.imshow(fixed_image, cmap='bone')
    ax1.set_title('$T_0$')
    ax2.imshow(moving_image, cmap='bone')
    ax2.set_title('$T_1$')
    ax3.imshow(frames[i], cmap='bone')
    ax3.set_title('Output')

    ax5.plot(losses[:i])
    ax5.set_xlim([0,N])
    ax5.set_ylim([0,np.max(losses)])
    im=ax4.imshow(frames[i]-fixed_image,cmap='RdBu',vmin=-150, vmax=150)
    # Update the colorbar mappable
    a4.set_data(frames[i] - fixed_image)
    ax4.set_title('Difference $T_1^{reg}$-$T_0$')

    ax3.plot(fixed_image.shape[1]//2-pars1[:i:10,1],fixed_image.shape[0]//2-
 ↪pars1[:i:10,2],'r.', label="Track");
    ax3.legend(loc="lower right");
    ax6.axis('off')

anim_code = FuncAnimation(fig,
                          update_frame,
                          frames=len(frames),
                          interval=1000,
                          repeat_delay=2000)#.to_html5_video()

anim_code.save('movies/mse_reg.mp4', fps=10)
plt.close('all')
```

0.12.9 Smoother Gradient

We can use a distance map of the segmentation to give us a smoother gradient

Prepare images

The image preparation this time requires a bit more work. To obtainthe distance map, we have to

1. Threshold the image. Here, I used a threshold at intensity 50.
2. Compute the distance map.

Afterwards we use the same steps as with the previous image.

```
# Convert NumPy arrays to tf.Tensor
th=50
dfixed_image = tf.convert_to_tensor(distance(g_roi(full_img)<th), dtype=tf.float32)
dmoving_image = tf.convert_to_tensor(distance(g_roi(full_shift_img)<th), dtype=tf.
 ↪float32)

# If they are 2D (grayscale), add channel dimension → [H, W, 1]
if dfixed_image.ndim == 2:
```

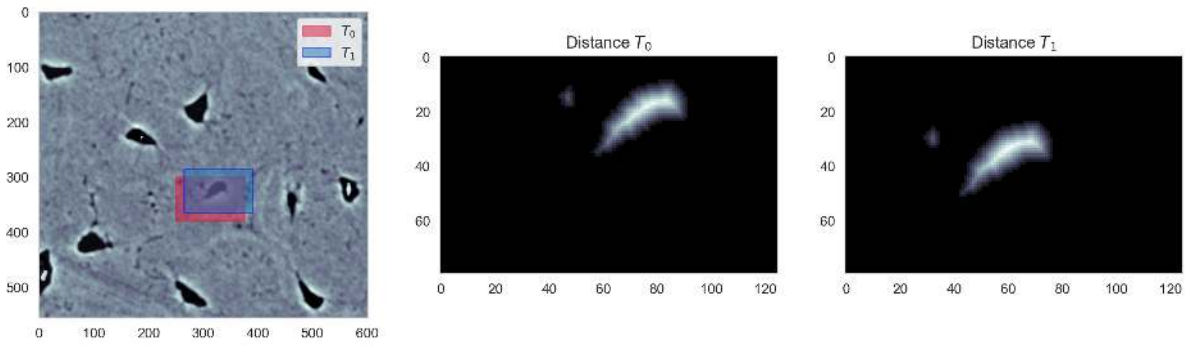
(continues on next page)

(continued from previous page)

```
    dfixed_image = tf.expand_dims(dfixed_image, axis=-1)

if dmoving_image.ndim == 2:
    dmoving_image = tf.expand_dims(dmoving_image, axis=-1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[66], line 3
      1 # Convert NumPy arrays to tf.Tensor
      2 th=50
----> 3 dfixed_image = tf.convert_to_tensor(distance(g_roi(full_img)<th), dtype=tf.
 ↪float32)
      4 dmoving_image = tf.convert_to_tensor(distance(g_roi(full_shift_img)<th),␣
 ↪dtype=tf.float32)
      6 # If they are 2D (grayscale), add channel dimension → [H, W, 1]

NameError: name 'distance' is not defined
```

```
a = distance(g_roi(full_img)<th)
b = distance(g_roi(full_shift_img)<th)
fig, (ax0,ax1, ax2) = plt.subplots(1, 3, figsize=(15, 4))
ax0.imshow(full_img,cmap='bone')
rect=Rectangle((roi[1],roi[0]),roi[3]-roi[1],roi[2]-roi[0],ec='crimson',fc='crimson',
 ↪alpha=0.5,label=r"$T_0$")
ax0.add_patch(rect)
rect=Rectangle((roi[1]+dy,roi[0]+dx),roi[3]-roi[1],roi[2]-roi[0],ec='blue',alpha=0.5,
 ↪label=r"$T_1$")
ax0.add_patch(rect)
ax0.legend()
ax1.imshow(a, cmap='bone')
ax1.set_title('Distance $T_0$')
ax2.imshow(b, cmap='bone')
ax2.set_title('Distance $T_1$');
```



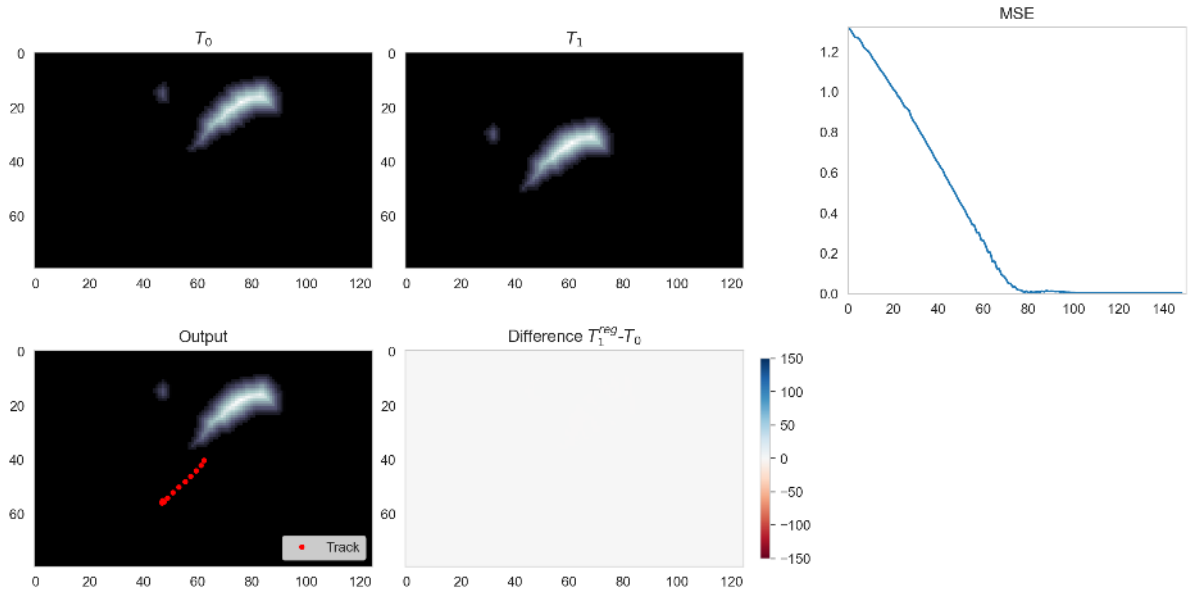
Run the registration loop

The registration loop looks much like the previous one. The only things changed are variable names to be able to compare the results later.

```
# Define the transform parameters as trainable variables
theta = tf.Variable(0.0, dtype=tf.float32, trainable=True)
tx    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
ty    = tf.Variable(0.0, dtype=tf.float32, trainable=True)
dtransform_params = [theta, tx, ty]

# Define the optimizer
optimizer = tf.optimizers.Adam(learning_rate=0.2)

# Define the training loop
N = 150
dlosses = []
dframes = []
dpars   = []
for i in range(N):
    with tf.GradientTape() as tape:
        tape.watch(dtransform_params)
        loss = similarity_loss(dfixed_image, dmoving_image, dtransform_params)

    gradients = tape.gradient(loss, dtransform_params)

    optimizer.apply_gradients(zip(gradients, dtransform_params))

    dlosses.append(loss.numpy())
    dpars.append(np.array([p.numpy() for p in dtransform_params], dtype=np.float32))
    dframes.append(affine_transform(dfixed_image, dmoving_image, dtransform_params))

# Apply the learned transform to the moving image
registered_image = affine_transform(dfixed_image, dmoving_image, dtransform_params)
```

The resulting registration using a distance map

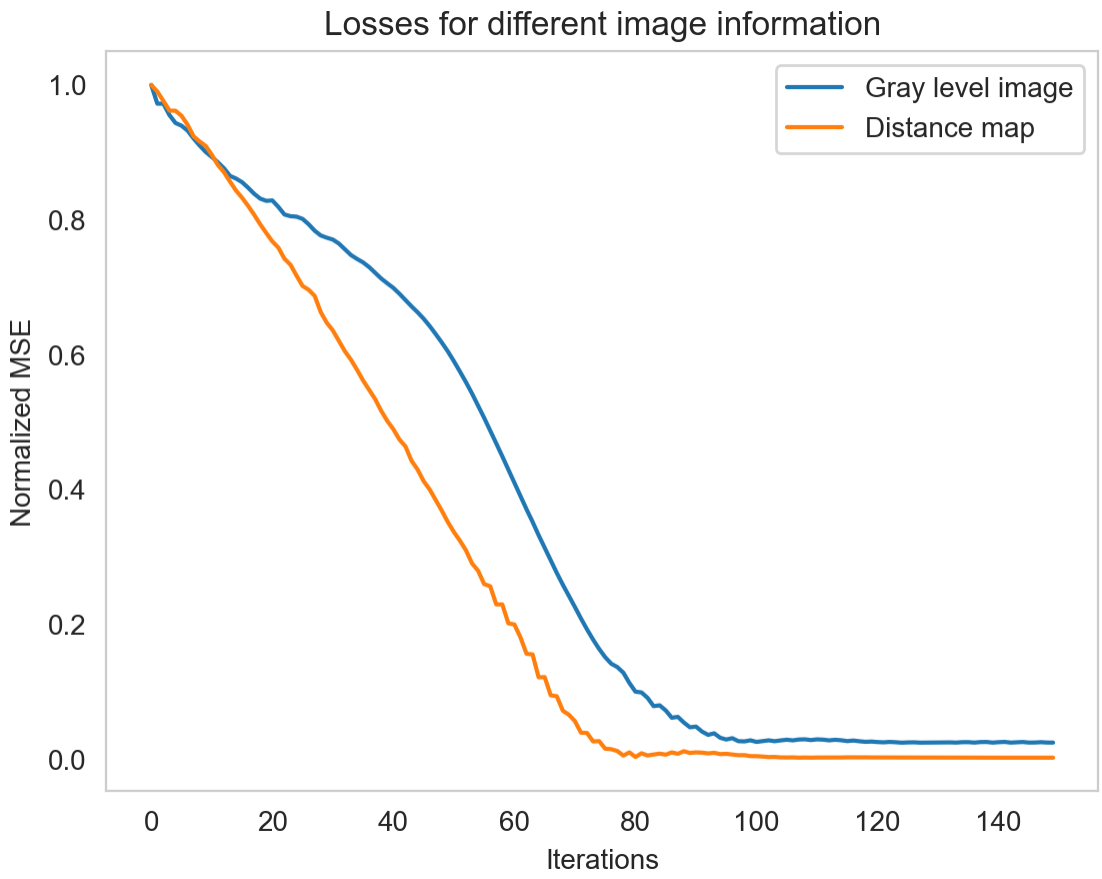
```
show_registration(dfixed_image,dmoving_image,dframes,dpars,dlosses)
```



We see here that registration using a distance map produces an almost straight track, while the registration with the gray level image had to search a bit before finding the optimal registration.

Compare losses

```
plt.plot(losses/np.max(losses),label="Gray level image")
plt.plot(dlosses/np.max(dlosses),label = "Distance map")
plt.legend()
plt.title("Losses for different image information")
plt.xlabel("Iterations")
plt.ylabel("Normalized MSE");
```



The straighter registration track also results in faster convergence as we can see when the two losses are plotted together. Here, we look at the normalized losses because the MSE is much greater in the gray level images due to a different gray level distribution ocmpared to the distance map.

Try structural similarity index metric

We have used MSE as loss function, how about trying SSIM from lecture 3?

0.12.10 Registration using ITK and Simple ITK

For medical imaging the standard tools used are ITK and SimpleITK and they have been optimized over decades to deliver high-performance registration tasks.

They are a bit clumsy to use from python, but they offer by far the best established tools for these problems.

ITK book

Registration script

```
import SimpleITK as sitk

def register_img(fixed_arr,
                 moving_arr):

    fixed_image = sitk.GetImageFromArray(fixed_arr)
    moving_image = sitk.GetImageFromArray(moving_arr)

    transform = sitk.AffineTransform(2)
    initial_transform = sitk.CenteredTransformInitializer(sitk.Cast(fixed_image,␣
↪moving_image.GetPixelID()),
                                                           moving_image,
                                                           transform,
                                                           sitk.
↪CenteredTransformInitializerFilter.GEOMETRY)
    ff_img = sitk.Cast(fixed_image, sitk.sitkFloat32)
    mv_img = sitk.Cast(moving_image, sitk.sitkFloat32)
    registration_method = sitk.ImageRegistrationMethod()

    registration_method.SetMetricAsMeanSquares()
    registration_method.SetMetricSamplingStrategy(registration_method.RANDOM)
    registration_method.SetMetricSamplingPercentage(0.25)
    registration_method.SetInterpolator(sitk.sitkLinear)
    registration_method.SetOptimizerAsGradientDescent(learningRate=1.0,
                                                      numberOfIterations=200,
                                                      convergenceMinimumValue=1e-6,
                                                      convergenceWindowSize=10)
    # Scale the step size differently for each parameter, this is critical!!!
    registration_method.SetOptimizerScalesFromPhysicalShift()

    registration_method.SetInitialTransform(initial_transform, inPlace=False)
    final_transform_v1 = registration_method.Execute(ff_img, mv_img)

    resample = sitk.ResampleImageFilter()
    resample.SetReferenceImage(fixed_image)
    # SimpleITK supports several interpolation options, we go with the simplest that␣
↪gives reasonable results.
    resample.SetInterpolator(sitk.sitkBSpline)
    resample.SetTransform(final_transform_v1)

    return sitk.GetArrayFromImage(resample.Execute(moving_image))
```

```
# A script with more flexibility
import SimpleITK as sitk

def register_img2(fixed_arr,
                  moving_arr,
                  use_affine=True,
                  use_mse=True,
                  brute_force=True):
    fixed_image = sitk.GetImageFromArray(fixed_arr)
    moving_image = sitk.GetImageFromArray(moving_arr)
    transform = sitk.AffineTransform(
        2) if use_affine else sitk.ScaleTransform(2)
```

(continues on next page)

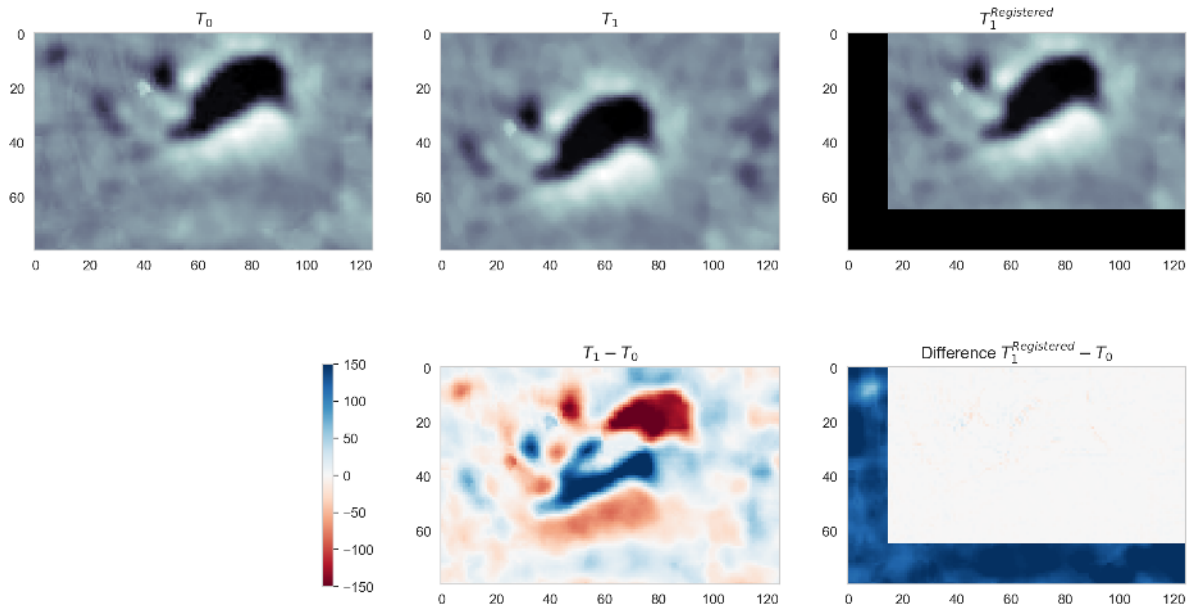
(continued from previous page)

```
    initial_transform = sitk.CenteredTransformInitializer(sitk.Cast(fixed_image,␣
↪moving_image.GetPixelID()),
                                                           moving_image,
                                                           transform,
                                                           sitk.
↪CenteredTransformInitializerFilter.GEOMETRY)
    ff_img = sitk.Cast(fixed_image, sitk.sitkFloat32)
    mv_img = sitk.Cast(moving_image, sitk.sitkFloat32)
    registration_method = sitk.ImageRegistrationMethod()
    if use_mse:
        registration_method.SetMetricAsMeanSquares()
    else:
        registration_method.SetMetricAsMattesMutualInformation(
            numberOfHistogramBins=50)

    if brute_force:
        sample_per_axis = 12
        registration_method.SetOptimizerAsExhaustive(
            [sample_per_axis//2, 0, 0])
        # Utilize the scale to set the step size for each dimension
        registration_method.SetOptimizerScales(
            [2.0*3.14/sample_per_axis, 1.0, 1.0])
    else:
        registration_method.SetMetricSamplingStrategy(
            registration_method.RANDOM)
        registration_method.SetMetricSamplingPercentage(0.25)

    registration_method.SetInterpolator(sitk.sitkLinear)

    registration_method.SetOptimizerAsGradientDescent(learningRate=1.0,
                                                      numberOfIterations=200,
                                                      convergenceMinimumValue=1e-6,
                                                      convergenceWindowSize=10)
    # Scale the step size differently for each parameter, this is critical!!!
    registration_method.SetOptimizerScalesFromPhysicalShift()

    registration_method.SetInitialTransform(initial_transform, inPlace=False)
    final_transform_v1 = registration_method.Execute(ff_img,
                                                     mv_img)
    print('Optimizer\'s stopping condition, {0}'.format(
        registration_method.GetOptimizerStopConditionDescription()))
    print('Final metric value: {0}'.format(
        registration_method.GetMetricValue()))
    resample = sitk.ResampleImageFilter()
    resample.SetReferenceImage(fixed_image)

    # SimpleITK supports several interpolation options, we go with the simplest that␣
↪gives reasonable results.
    resample.SetInterpolator(sitk.sitkBSpline)
    resample.SetTransform(final_transform_v1)
    return sitk.GetArrayFromImage(resample.Execute(moving_image))
```

Registration result

```
%matplotlib inline
reg_img = register_img(bw_img, shift_img)
```

```
fig, ((ax1, ax2, ax3),(ax10,ax2d, ax4)) = plt.subplots(2,3, figsize=(15, 8))
a0=ax10.axis('off')
ax1.imshow(bw_img, cmap='bone')
ax1.set_title('$T_0$')
ax2.imshow(shift_img, cmap='bone')
ax2.set_title('$T_1$')
ax2d.imshow(1.0*bw_img-shift_img, cmap='RdBu', vmin=-150, vmax=150)
ax2d.set_title('$T_1-T_0$')
ax3.imshow(reg_img, cmap='bone')
ax3.set_title('$T_1^{Registered}$')
a4=ax4.imshow(1.0*bw_img-reg_img, cmap='RdBu',vmin=-150, vmax=150)
ax4.set_title(r'Difference $T_1^{Registered}-T_0$');#
fig.colorbar(a4,ax=ax10,shrink=0.8);
```



0.12.11 Local changes between images

We can approach the problem by subdividing the data into smaller blocks and then apply the digital volume correlation independently to each block.

- information on changes in different regions
- less statistics than a larger box

0.13 Introducing Physics

DIC or DVC by themselves include no *sanity check* for realistic offsets in the correlation itself.

The method can, however be integrated with physical models to find a more optimal solutions.

- information from surrounding points
- smoothness criteria
- maximum deformation / force
- material properties

$$C_{\text{cost}} = \underbrace{C_{I_0,I_1}(\vec{r})}_{\text{Correlation Term}} + \underbrace{\lambda||\vec{r}||}_{\text{deformation term}}$$

0.13.1 Distribution Metrics

As we covered before distribution metrics like the distribution tensor can be used for tracking changes inside a sample.

Of these the most relevant is the texture tensor from cellular materials and liquid foam. The texture tensor is the same as the distribution tensor except that the edges (or faces) represent physically connected / touching objects rather than touching Voronoi faces (or conversely Delaunay triangles).

These metrics can also be used for tracking the behavior of a system without tracking the single points since most deformations of a system also deform the distribution tensor and can thus be extracted by comparing the distribution tensor at different time steps.

0.13.2 Quantifying Deformation: Strain

We can take any of these approaches and quantify the deformation using a tool called the strain tensor.

Strain is defined in mechanics for the simple 1D case as the change in the length against the change in the original length.

$$e = \frac{\Delta L}{L}$$

While this defines the 1D case well, it is difficult to apply such metrics to voxel, shape, and tensor data.

0.13.3 Strain Tensor

There are a number of different ways to calculate strain and the strain tensor, but the most applicable for general image based applications is called the infinitesimal strain tensor, because the element matches well to square pixels and cubic voxels.

0.13.4 Types of Strain

We catagorize the types of strain into two main catagories:

$$\underbrace{\underset{\sim}{\mathbf{E}}}_{\text{Total Strain}} = \underbrace{\varepsilon_M \mathbf{I_3}}_{\text{Volumetric}} + \underbrace{\underset{\sim}{\mathbf{E}'}}_{\text{Deviatoric}}$$

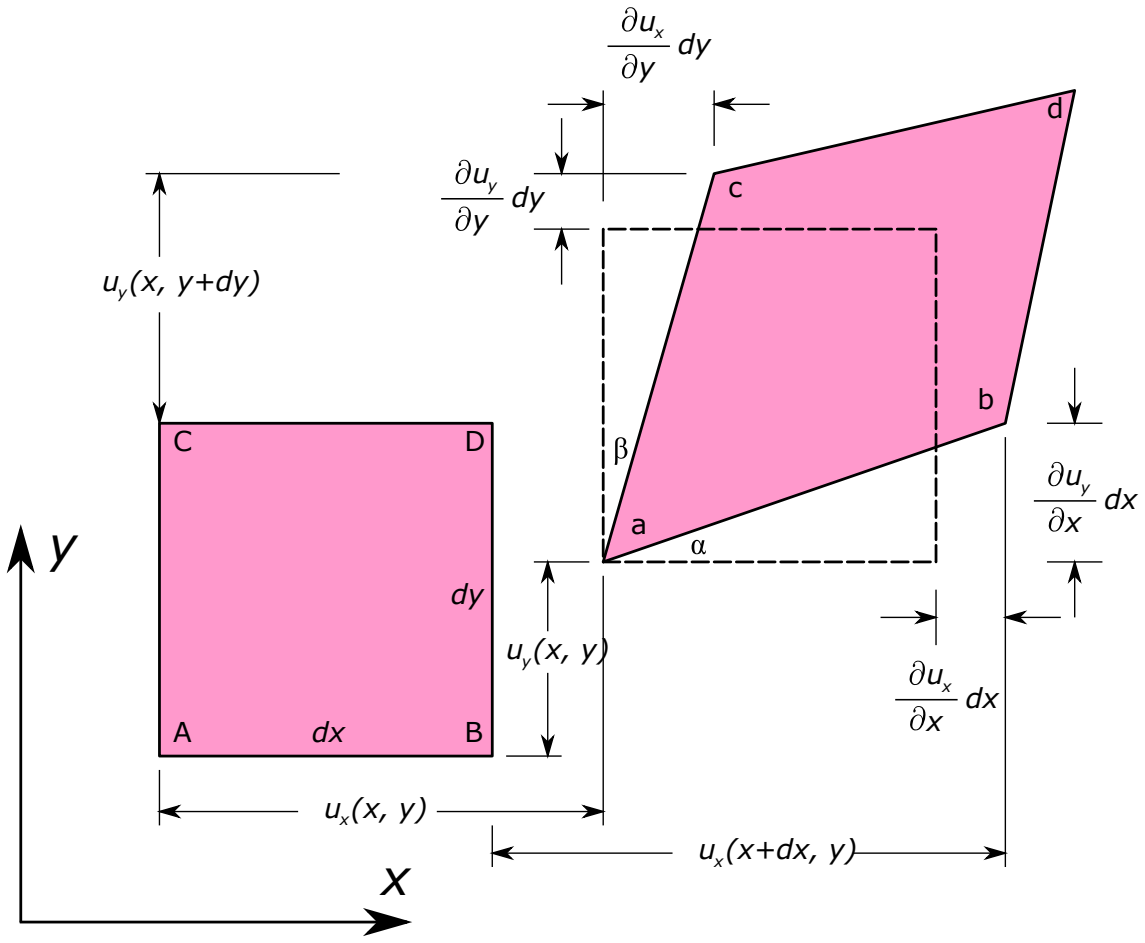


Fig. 1: Deformation by strain.

Volumetric / Dilational

The isotropic change in size or scale of the object.

Deviatoric

The change in the proportions of the object (similar to anisotropy) independent of the final scale

Shear band

S. Hall 2013

0.14 Two Point Correlation - Volcanic Rock

Data provided by Mattia Pistone and Julie Fife

The air phase changes from small very anisotropic bubbles to one large connected pore network.

- The same tools cannot be used to quantify those systems.
- Furthermore there are motion artifacts which are difficult to correct.

We can utilize the two point correlation function of the material to characterize the shape generically for each time step and then compare.

0.15 Presenting the results - bringing out the message

In the end you will want to present your results

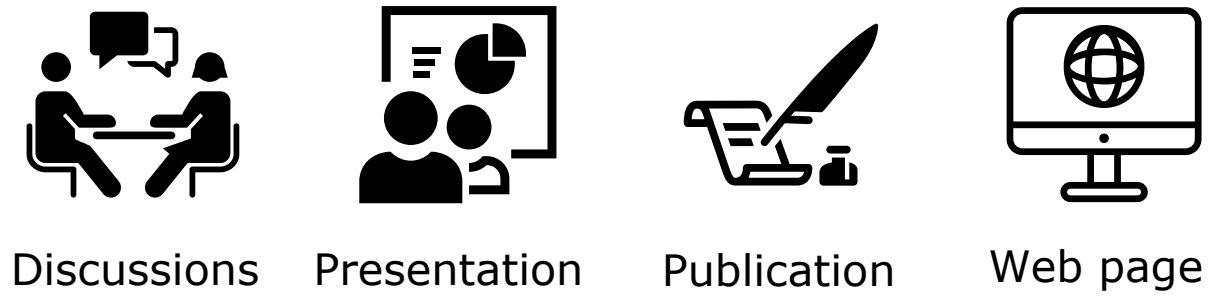


Fig. 1: Different ways to present your data.

0.15.1 Visualization

One of the biggest problems with *big* sciences is trying to visualize a lot of heterogeneous data.

- Tables are difficult to interpret
- 3D Visualizations are very difficult to compare visually
- Contradictory necessity of simple single value results and all of the data to look for trends and find problems

Purpose of the visualization

You visualize your data for different reasons:

1. **Understanding and exploration**
    - Small and known audience (you and colleagues)
    - High degree of understanding of specific topic.
2. **Presenting your results**
    - Wider and sometimes unknown audience (reader of paper, person listening to presentation)
    - At best general understanding of the topic.

from Knaflic 2015

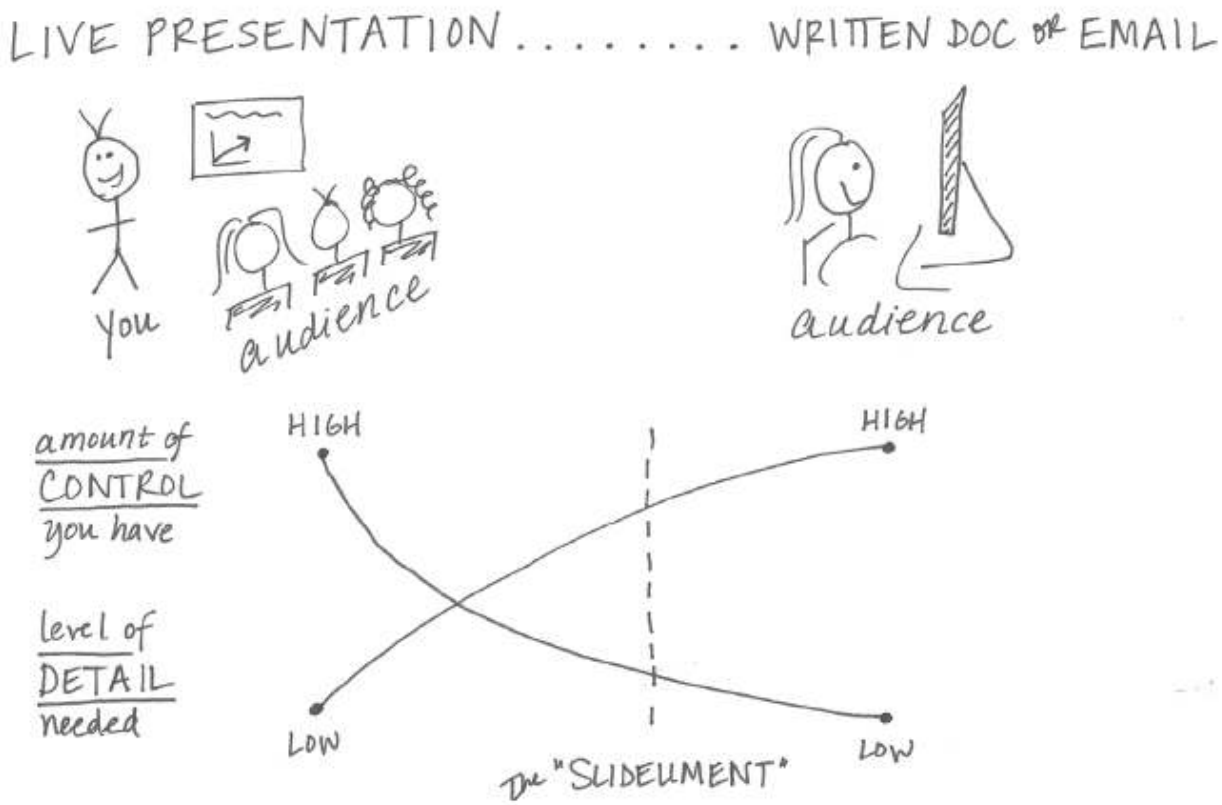


Fig. 2: The level of detail in a presentation depends on the medium it is presented Knaflic 2015.

0.15.2 Bad Graphs

There are too many graphs which say:

- *my data is very complicated*
- *I know how to use __ toolbox in Python/Matlab/R/Mathematica*
- Most programs by default make poor plots
- Good visualizations takes time to produce

xkcd

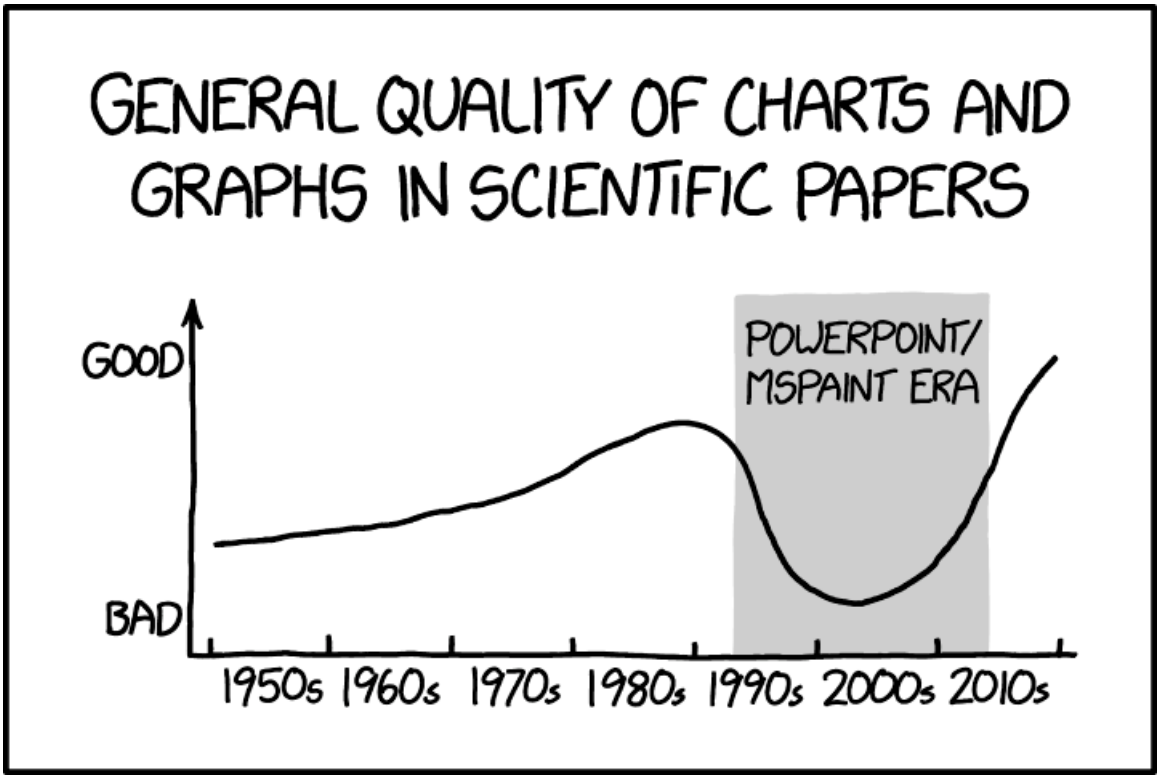


Fig. 3: This cartoon from XKCD highlights a problem with the access to software making it too easy to produce a graph or illustration. Unfortunately, without any thoughts about information content and artistic or illustration rules.

Some bad examples

There are plenty examples on how you shouldn't present your data. The problem is in general that there is way too much information that needs to be predigestend before it is ready to any audience.

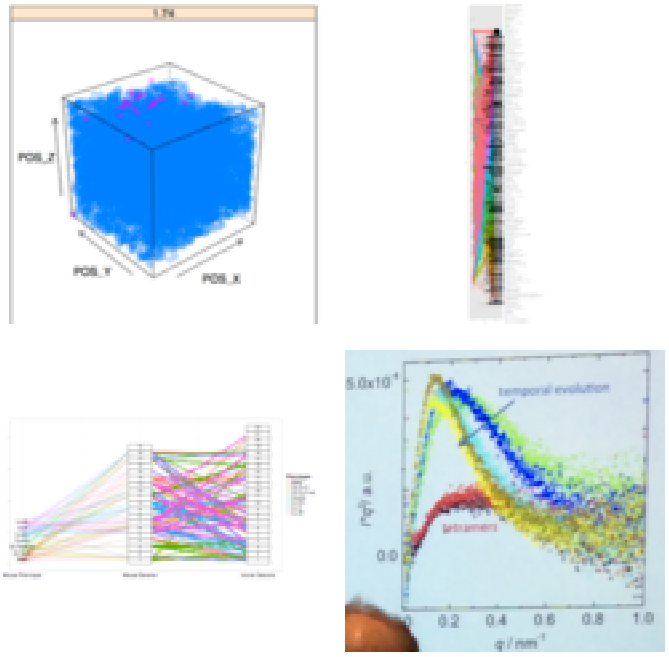


Fig. 4: Four examples of how *not* to present your data.

0.15.3 How to improve - Key Ideas

There is a need to consciously prepare your figures to bring your message to the audience in an understandable way. The first step is to ask yourself the following questions.

What is my message?

Is it clearly communicated?

Is it really necessary?

- Does every line / color serve a purpose?
- Pretend ink is very expensive

Keep this in mind every time you create a figure and you will notice that after while you will have a tool set that makes it easier and faster to produce well thought figures that clearly brings out you message to your audience.

Personally, I always write scripts to produce each plot of a publication. This makes it easier to revise the manuscript in a reproducible and efficient manner. The first implementation may take longer, but the revision is done in no time.

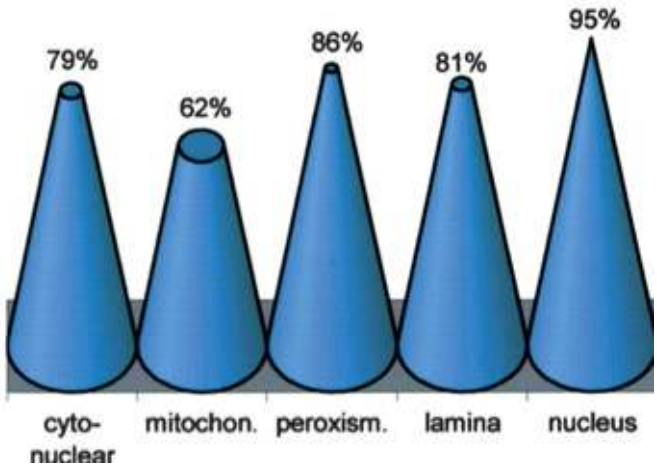
Some literature

These are some recommended books about data visualization:

- Knaflic, Storytelling with Data: A Data Visualization Guide for Business Professionals, 2015
- Few, Should data visualization always be beautiful?, 2012

Simple rules

1. Never use 3D graphics when it can be avoided (unless you want to be deliberately misleading)



2. Pie charts can also be hard to interpret
3. Background color should almost always be white (not light gray)
4. Use color palettes adapted to human visual sensitivity
5. Use colors and transparency smart

0.15.4 Grammar of Graphics

What is a grammar?

- Set of rules for constructing and validating a sentence
- Specifies the relationship and order between the words constituting the sentence

How does grammar apply to graphics?

If we develop a consistent way of

- expressing graphics (sentences)
- in terms of elements (words) we can compose and decompose graphics easily

The most important modern work in graphical grammars is
"The Grammar of Graphics" by Wilkinson, Anand, and Grossman (2005).

This work built on earlier work by Bertin (1983) and proposed a grammar that can be used to describe and construct a wide range of statistical graphics.

Grammar Explained

Normally we think of plots in terms of some sort of data which is fed into a plot command that produces a picture

- In Excel you select a range and plot-type and click "Make"
- In Matlab you run `plot(xdata,ydata,color/shape)`

1. These produces entire graphics (sentences) or at least phrases in one go and thus abstract away from the idea of grammar.
2. If you spoke by finding entire sentences in a book it would be very ineffective, it is much better to build up word by word

Grammar

Separate the graph into its component parts

$$\begin{cases} var1 \rightarrow x \\ var2 \rightarrow y \end{cases}$$

Construct graphics by focusing on each portion independently.

Figure decorations

Besides the data you also need to provide annotating items to the visualization.

It may seem unnescessary to list these annotations, but it happens too often that they are missing. This leaves the observers wondering about what they see in the figure. It is true that it takes a little more time to add annotation to your figure. Sometimes, you may think that the plot is only for your own understanding and you don't need to waste the time on making it complete. Still, in the next moment it finds its way to the presentation and then all of a sudden it is offical… Annotations are fundamental features of figures and available in any plotting library. In some cases you have to look a little longer to find them or write a little more code to use them, but they are there.

Plots

- Curve legend - telling what each curve represents.
- Axis labels - tellning what information you see on each axis.
- Figure title - if you use multiples plots in the same figure.

Images

- Color bar - to tell how the colors are mapped to the values.
- Scale bar - to tell the size of the object in the image.

Color maps revisited

Choosing the right color is a science.

Crameri, F., et al. (2020)

The choice depends on the type of data you want to present and how humans percieve different colors. Some combinations make it easier to highlight relevant features in the images. Still, you have to be cautious not put too much a priori information into the color map.

Visualization toolboxes provide a great collection of colormaps as we have seen several times already in this course. There are however cases when you have to define your own color map. An example is the colormap we created last week to be able to identify each item in a watershed segmented image.

0.15.5 What is my message?

Plots to "show the results" or "get a feeling" are usually not good

```
from plotnine import *
from plotnine.data import *
import pandas as pd
import numpy as np
# Some data
xd = np.random.rand(80)
yd = xd + np.random.rand(80)
zd = np.random.rand(80)

df = pd.DataFrame(dict(x=xd,y=yd,z=zd))
ggplot(df,aes(x='x',y='y')) + geom_point()
```

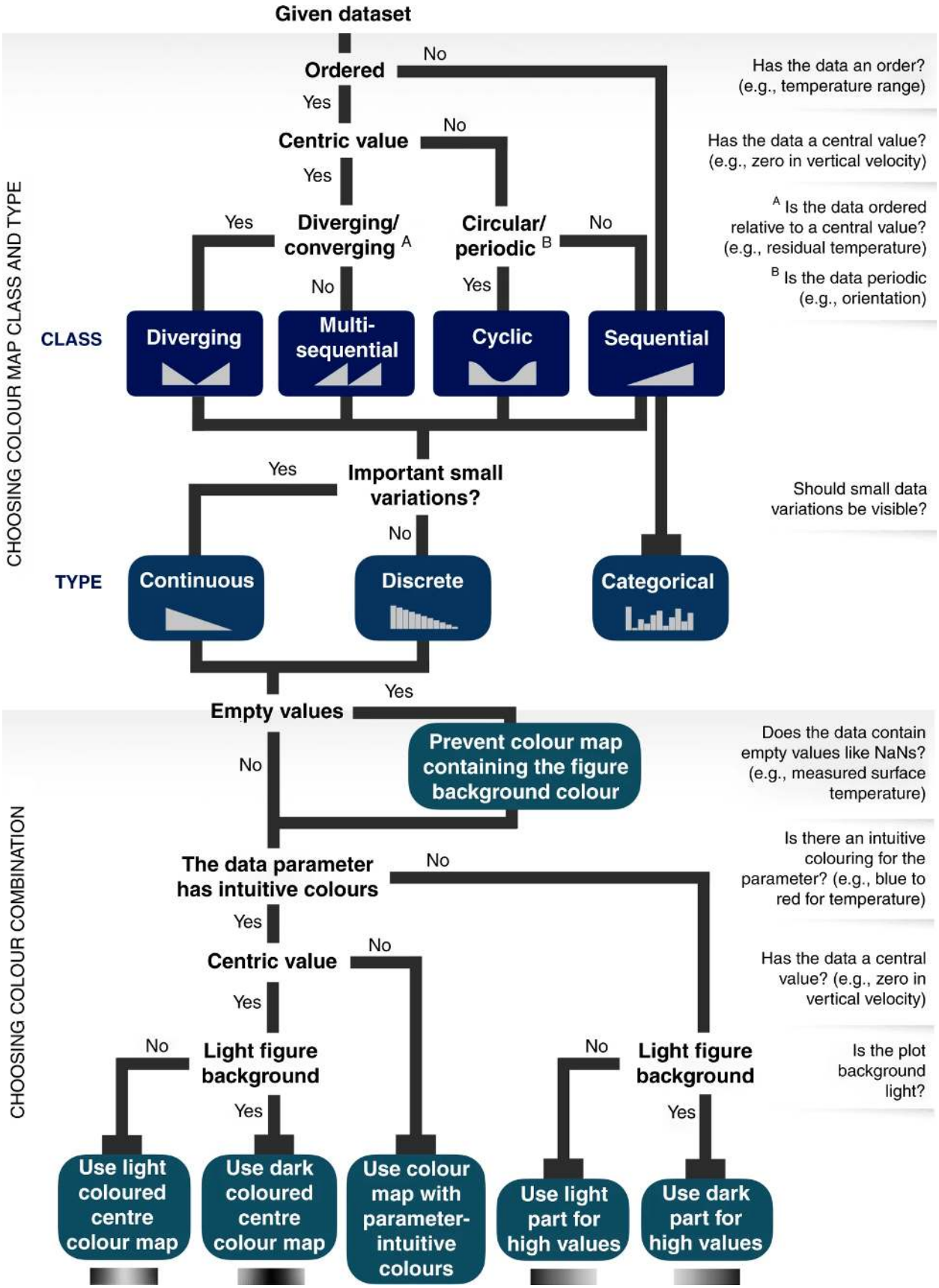
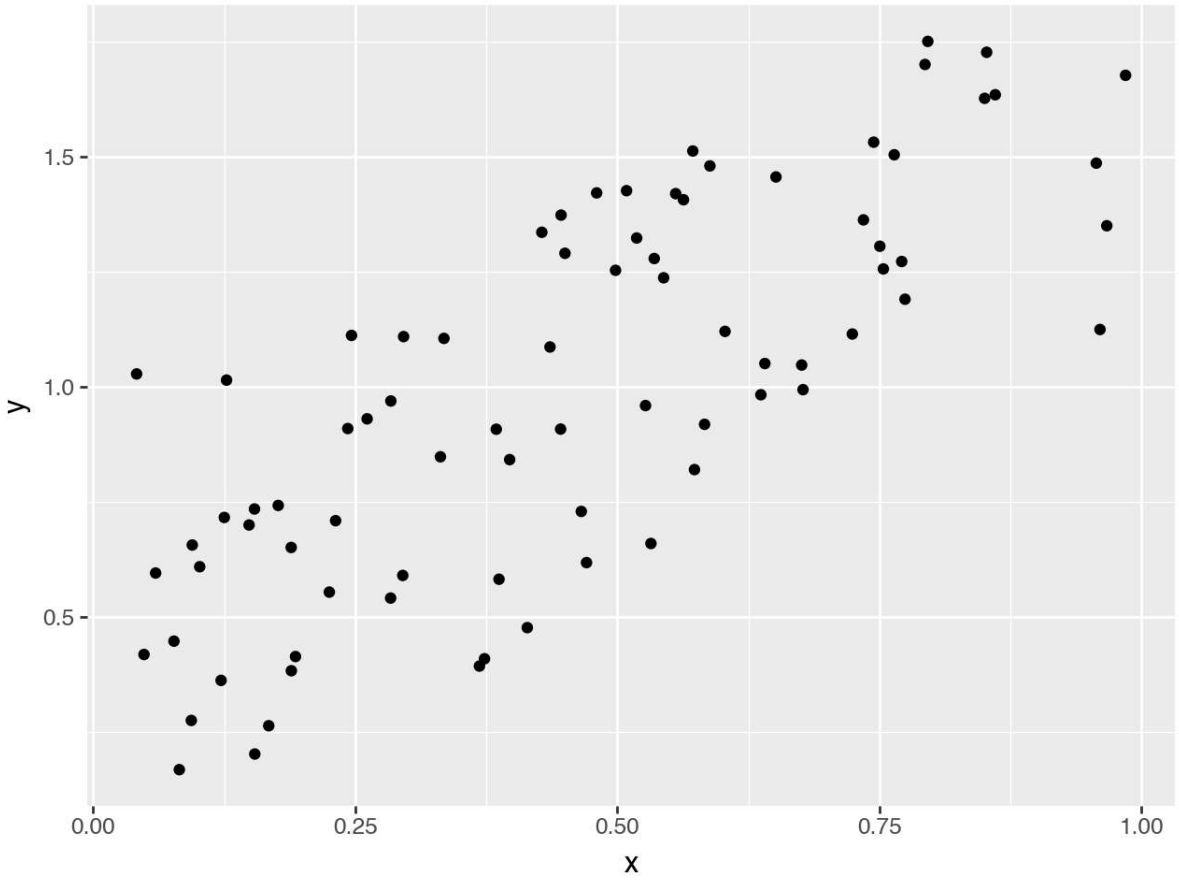


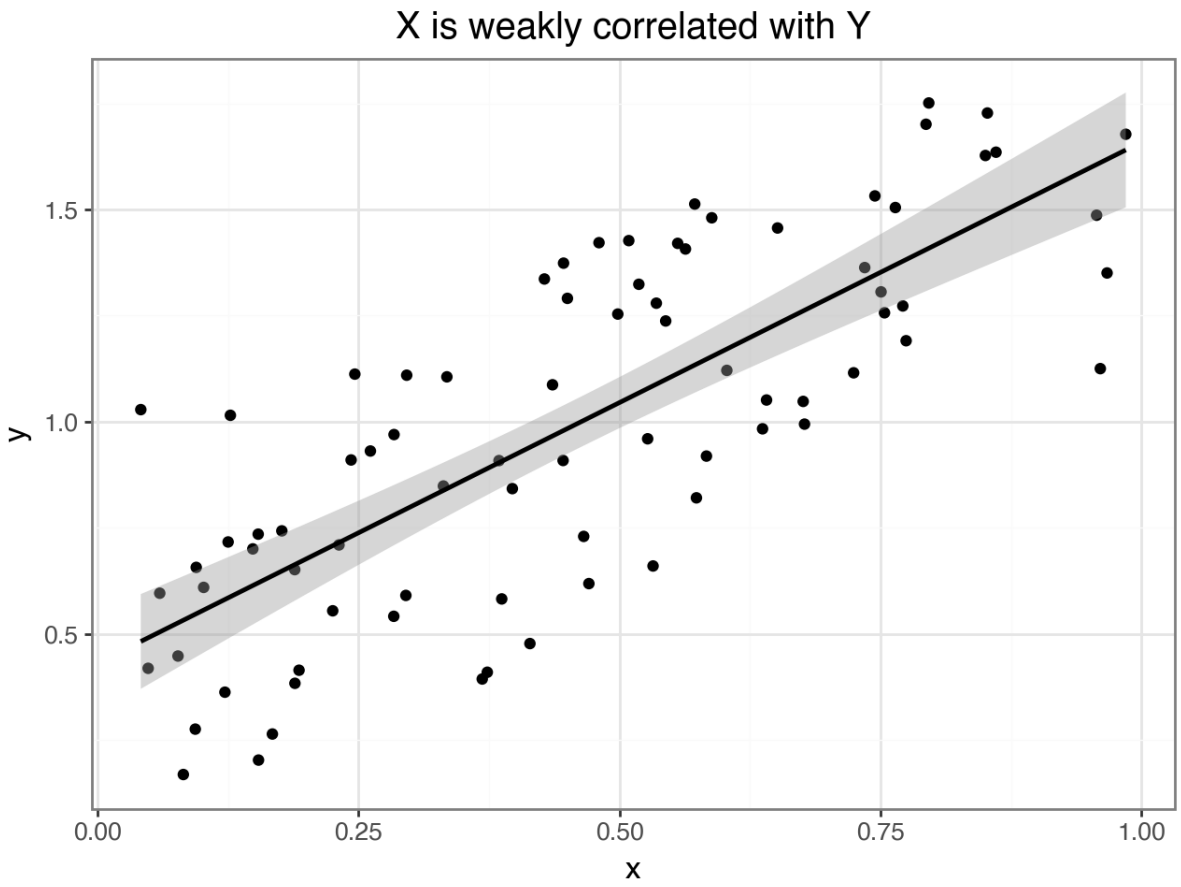
Fig. 5: Crameri et al. developed this decision flow chart to help you decide which type of color map is best suited for your data.



Focus on a single, simple message

“X is a little bit correlated with Y”

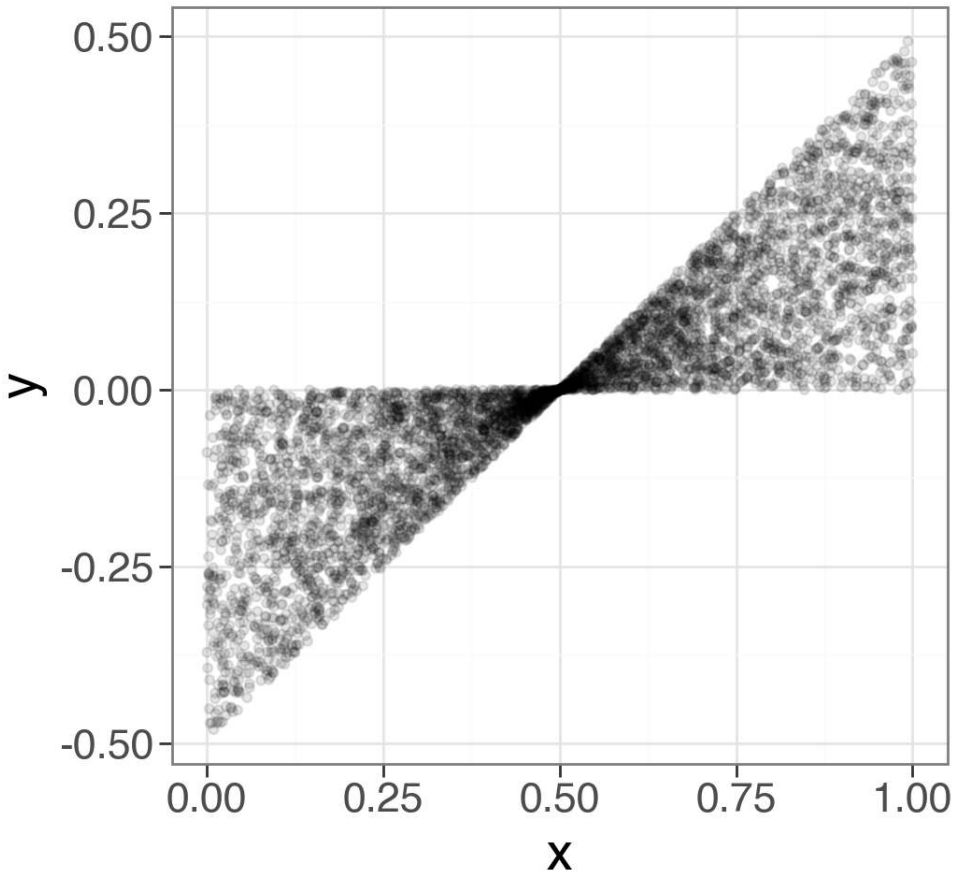
```
(ggplot(df,aes(x='x',y='y'))
 + geom_point()
 + geom_smooth(method="lm")
# + coord_equal()
 + labs(title="X is weakly correlated with Y")
 + theme_bw(12) )
```



Does my graphic communicate it clearly?

Too much data makes it very difficult to derive a clear message

```
xd = np.random.rand(5000)
yd = (xd-0.5)*np.random.rand(5000)

df = pd.DataFrame(dict(x=xd,y=yd))
(ggplot(df,aes(x='x',y='y'))
# + geom_point()
 + geom_point( alpha = 0.1 )
 + coord_equal()
 + theme_bw(20))
```



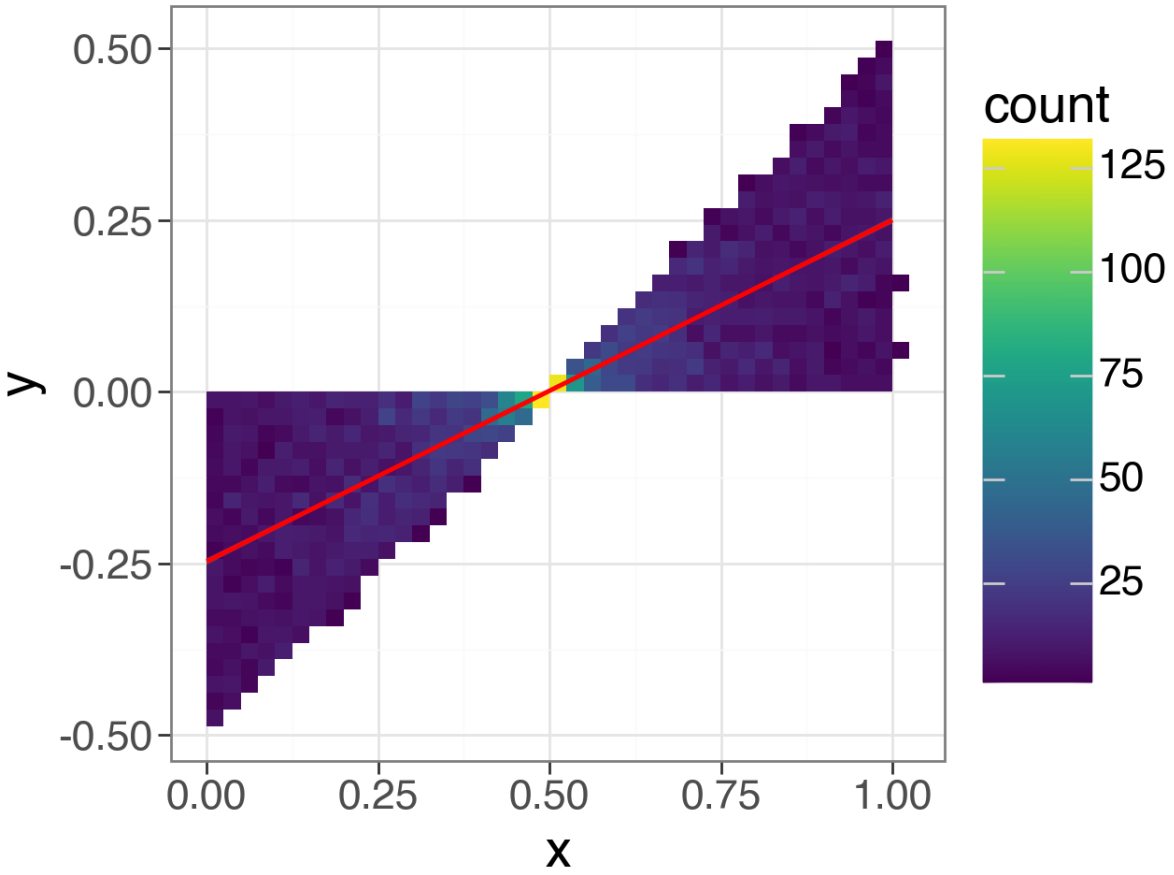
We have earlier used transparency to better visualize dense scatter plots. You can see the effect by setting the alpha parameter to geom_point. Using transparency is a qualitative way of showing higher density in the data.

Reduce the data

Filter and reduce information until it is extremely simple

In this plot we create a density count view of the data by downsampling the grid and count the amount of points in each bin. It is related to a histogram but it count in space instead of in the intensity levels.

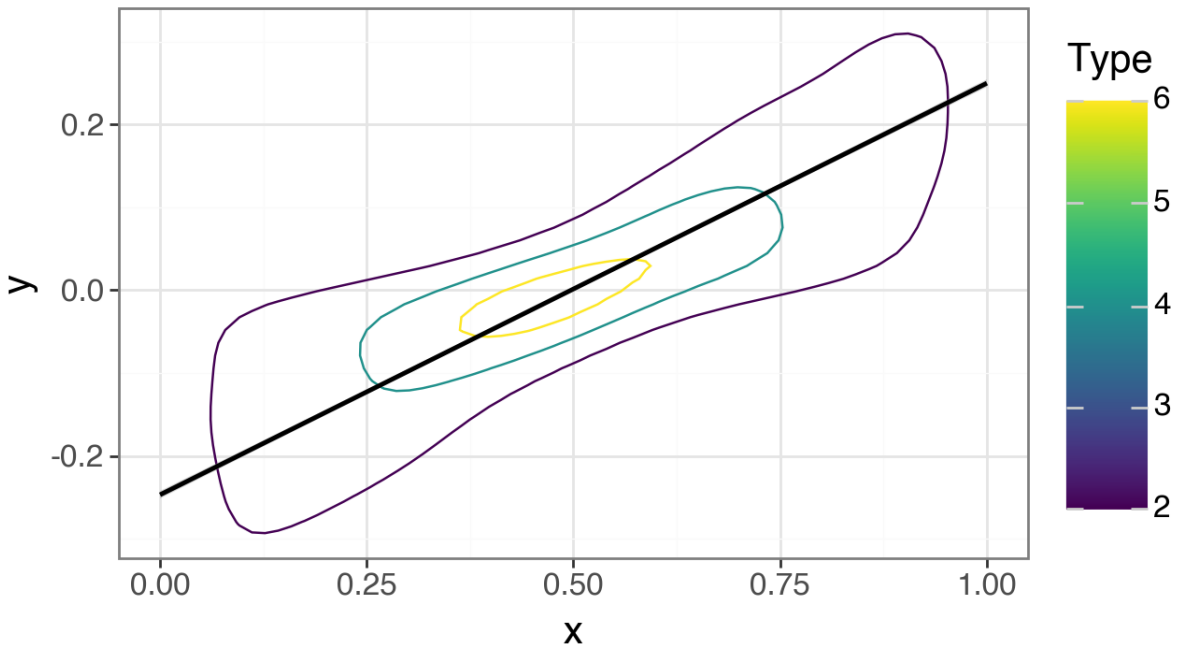
```
(ggplot(df,aes(x='x',y='y'))
 + stat_bin_2d(bins=40)
 + geom_smooth(method="lm",color='red')
 + coord_equal()
 + theme_bw(20)
 + guides(color='F')
)
```



Using this kind of plot allows us to measure how many points there are in each bin and thus we are now going towards a quantitative plot. The colorbar on the side helps us to interpret the colors.

Reduce even further

```
(ggplot(df,aes(x='x',y='y'))
  + geom_density_2d(aes(x='x', y='y', color='..level..'))
  + geom_smooth(method="lm")
  + coord_equal()
  + labs(color="Type")
  + theme_bw(15)
)
```



0.15.6 Common visualization packages for python

- Matplotlib Matplotlib 3.0 Cookbook or ETHZ lib, code examples
- Plotly
- Seaborn
- ggplot R using the ggplot2 library, which is ported to python.

A short summary of these packages can be found here.

0.16 Summary

0.16.1 Time series analysis

- Dynamic experiments
- Object tracking
- Registration
- Digital volume correlation (DIC)

0.16.2 Visualizing results

- Define your message
- Plan your visualization
- Use the right method