combination_30d_10bin

April 28, 2020

1 Companion Code for

2 Soil moisture: variable in space but redundant in time

This is the companion code for the publication *Mälicke et al. (2019) Soil moisture: variable in space but redundant in time* (DOI: 10.5194/hess-2019-574). Please refer to the full text for details on the method.

A number of soil moisture time series recorded in the Attert Experimental Watershed will be loaded. The time series are analysed for spatial dependencies at different points in time. Are spatial patterns persistant nad can be hence compress the spatial information?

If you use parts of the code, please cite the publication.

Load packages.

[1]:
```
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from pprint import pprint
from datetime import datetime as dt
from scipy.spatial.distance import pdist
from collections import Counter
from skinfo import entropy
```

The `main.py` in the same folder is a collection of the actual method and some helpful plotting routines.

[2]:
```
from main import minmax, extract, dispersion, cluster_variograms, variograms,␣
 ↪clustered_series, compress_cluster
from main import plot_variogram, plot_overview, plot_cluster, heat_diagram,␣
 ↪plot_compressed
from main import variogram_entropy, entropy_report, cluster_entropy,␣
 ↪information_loss
```

In the paper you will find some basic hyperparameters, that are not changed for the whole analysis. These are listed below.

[3]:
```
# saving options
main_name = 'results/%s_dispersion_30days_%d.png'

#parameters
rank=False
estimator='cressie'
#maxlag='median'
maxlag=1200
window=30
bandwidth=30
#bandwidth=25
cl_threshold=10
```

Make the plots for publication friendly

[4]:
```
# some rc params
matplotlib.rc('font', **{'size': 20, 'family': 'Ubuntu', 'weight': 'normal'})
matplotlib.rc('axes', **{'labelsize': 20, 'spines.top': False, 'spines.right':␣
 ↪False, 'facecolor': 'white'})
```

2.1 Load data

The data is stored in a HDF5 file. The identifiers for the soil moisture observations are `m10`, `m30` and `m50`. Precipitation data is called `rain` and the air temperature is identified by the `temperature` key. All data is of daily resolution. The file is available upon request until a data publication is finished.

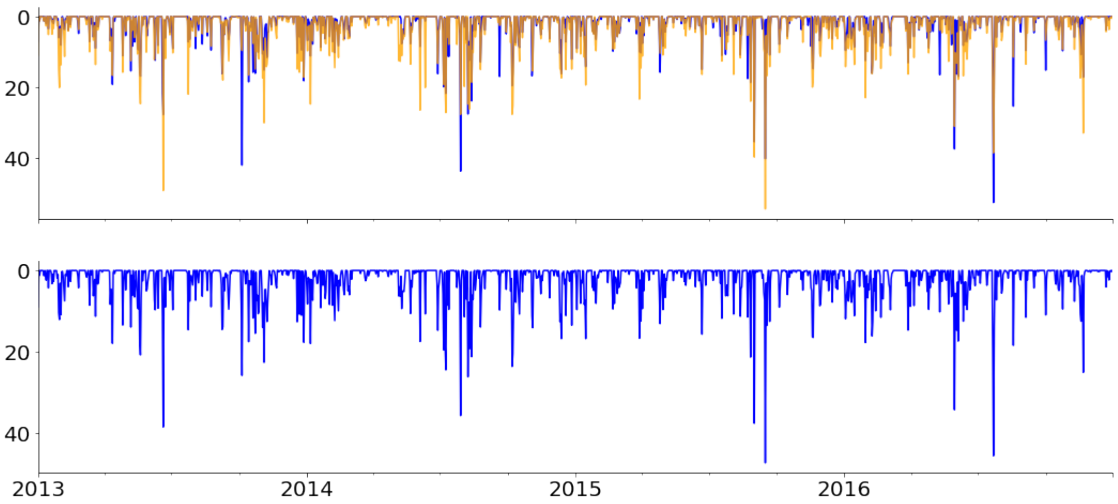
[5]:
```
store = pd.HDFStore('daily_agg.hd5')

# soil moisture
m10 = store.get('m10')
m30 = store.get('m30')
m50 = store.get('m50')

# rainfall
rain = store.get('rainfall')
temperature = store.get('temperature')
store.close()
```

2.1.1 Rainfall

[6]:
```
rr = rain.mean(axis=1)

def rainfall(r):
    fig, axes = plt.subplots(2, 1, figsize=(18,8), sharex=True)

    rain.Useldange.plot(ax=axes[0], color='b')
    rain.Roodt.plot(ax=axes[0], color='orange', alpha=0.8)
```

```
    rr.plot(ax=axes[1], legend=None, color='b')
    axes[0].invert_yaxis()
    axes[1].invert_yaxis()

    return fig
```

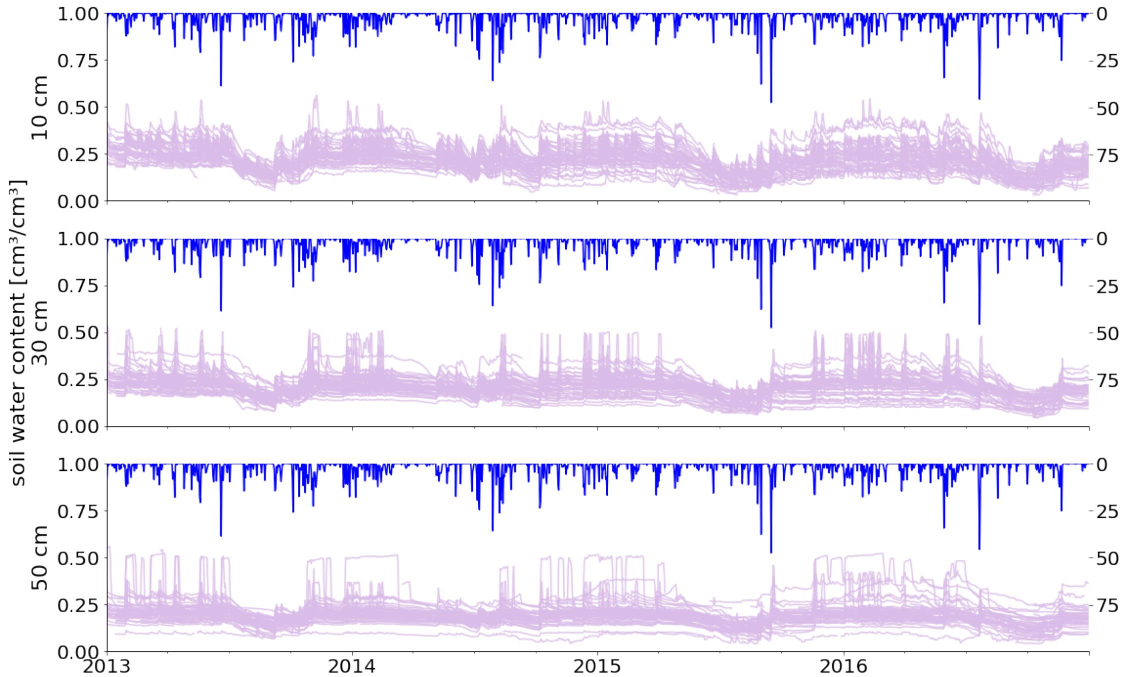
[7]:
```
fig = rainfall(rain)
```



2.1.2 Soil moisture overview

[8]:
```
fig, _a = plt.subplots(3,1, figsize=(18, 12), sharex=True, sharey=True)
axes = _a.flatten()

m10.drop(m10.columns[np.where(m10.mean() >= 0.35)].values, axis=1, inplace=True)
m30.drop(m30.columns[np.where(m30.mean() >= 0.35)].values, axis=1, inplace=True)
m50.drop(m50.columns[np.where(m50.mean() >= 0.35)].values, axis=1, inplace=True)

m10.plot(ax=axes[0], color='#DABCE8', legend=None, alpha=0.8)
m30.plot(ax=axes[1], color='#DABCE8', legend=None, alpha=0.8)
m50.plot(ax=axes[2], color='#DABCE8', legend=None, alpha=0.8)

for ax in axes:
    lim = ax.get_ylim()
    ax.set_ylim((0, 1))

ax2 = [ax.twinx() for ax in axes]
for ax in ax2:
    rr.plot(ax=ax, legend=None, color='b')
    ax.invert_yaxis()
    lim = ax.get_ylim()
```

```
    ax.set_ylim((lim[0] * 2, 0))

axes[0].set_ylabel('10 cm', fontsize=22)
axes[1].set_ylabel('soil water content [cm³/cm³]\n30 cm', fontsize=22)
axes[2].set_ylabel('50 cm', fontsize=22)

fig.savefig('results/all_data.pdf')
```



2.2 Spatial information

The spatial information is stored in the `positions.csv` file. Extract the x and y coordinate. The *d* column is a unique identifier, that can be found in the data series as column descriptors. This way, the locations can be mapped to the right data column. The coordinate system is a projected one (EPSG: 2169), using meter as an unit. Therefore no transformation needed.

[9]:
```
# get the positions
positions = pd.read_csv('positions.csv')
positions.head()
```

[9]:

	d	x	y	start	\
0	32	53693.837759	98752.205367	2012-03-01 18:50:00.000000	
1	33	53693.837759	98752.205367	2012-03-01 18:50:00.000000	
2	34	53693.837759	98752.205367	2012-03-01 18:50:00.000000	
3	35	53691.342952	98749.025185	2012-03-01 18:50:00.000000	
4	90	53665.550000	98779.541089	2012-05-12 18:05:00.000000	

```
                          stop  elevation  depth
0  2017-02-07 00:00:00.000000        470     10
1  2017-02-07 00:00:00.000000        470     30
2  2012-08-17 11:40:00.000000        470     50
3  2017-02-07 00:00:00.000000        470     10
4  2017-02-07 00:00:00.000000        473     10
```

[10]:

```
positions['pos'] = ['pos_%d' % i for i in positions.d.values]
positions['geom'] = [(r['x'], r['y'],) for i,r in positions.iterrows()]

pos10 = positions.where(positions.depth==10).dropna()[['pos', 'x', 'y']]
pos10.set_index('pos', inplace=True)
pos30 = positions.where(positions.depth==30).dropna()[['pos', 'x', 'y']]
pos30.set_index('pos', inplace=True)
pos50 = positions.where(positions.depth==50).dropna()[['pos', 'x', 'y']]
pos50.set_index('pos', inplace=True)
```

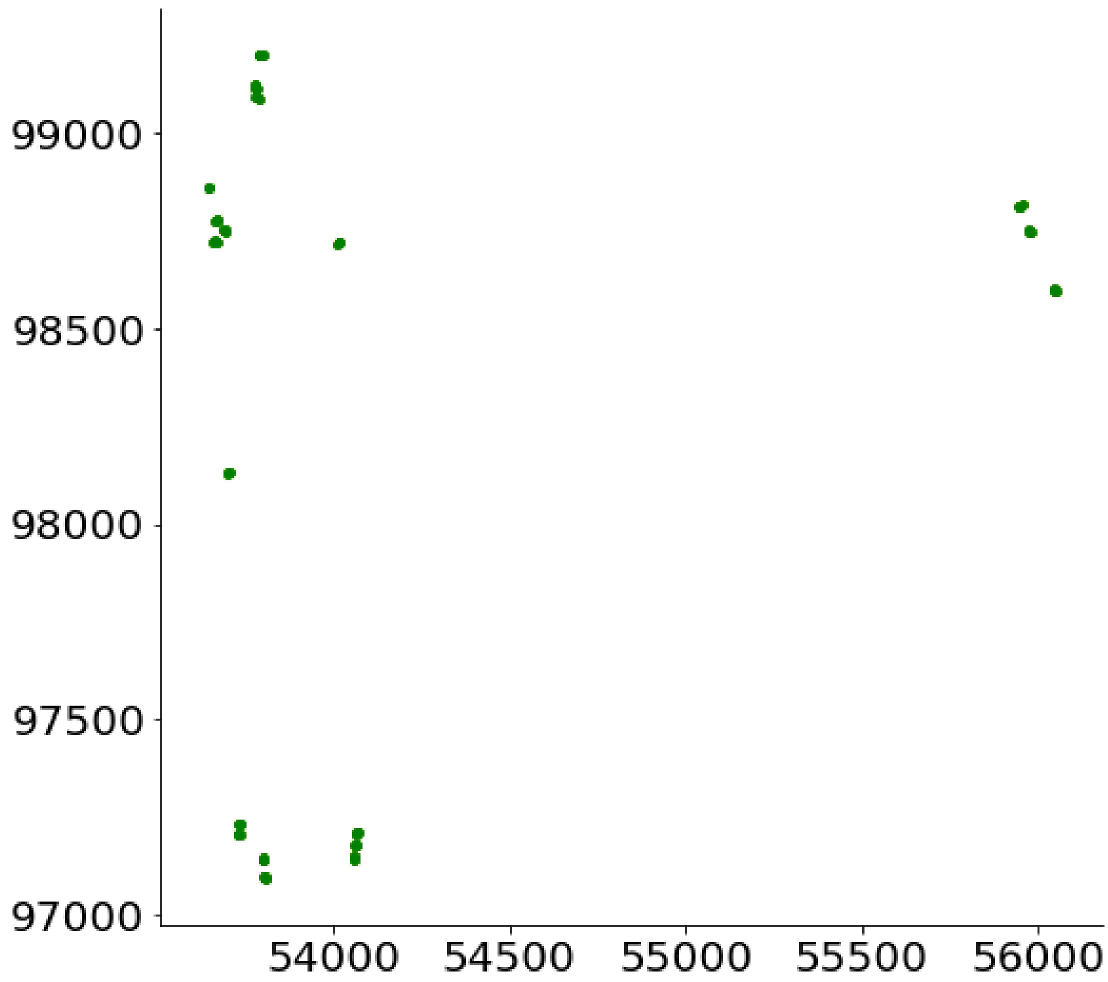
Make an overview plot:

[11]:

```
# uncomment these lines to further subset the data
#npos10 = pos10.where((pos10.x < 54500) & (pos10.y < 97500)).dropna()
#npos30 = pos30.where((pos30.x < 54500) & (pos30.y < 97500)).dropna()
#npos50 = pos50.where((pos50.x < 54500) & (pos50.y < 97500)).dropna()
npos10 = pos10
npos30 = pos30
npos50 = pos50
```

[12]:

```
fig, ax = plt.subplots(1, 1, figsize=(8, 8))
ax.scatter(pos10.x.values, pos10.y.values, 15, c='y')
ax.scatter(npos10.x.values, npos10.y.values, 15, c='g');
```



3 Analysis

The following section runs the same code for all years and depths to produce result output graphs.

[13]:
```
# Helper function to create the output graph
def create_segment_plot(obs, variograms, mean_shifts, compressed, rainfall,␣
 ↪temperature):
    # create figure
    fig, axes = plt.subplots(2, 3, figsize=(3*6, 2*6), sharey='row')
    largeax = plt.subplot2grid((2,3), (1,0), colspan=3)

    # extract the compressed info
    xbins = compressed[0]
    monos = compressed[1]

    # plot
```

```
    plot_variogram(variograms, ax=axes[0,0], norm=True)
    plot_cluster(variograms, mean_shifts, ax=axes[0,1], alpha=0.8, norm=True,␣
 ↪ylabel=False)
    plot_compressed(xbins, monos, ax=axes[0,2], ylabel=False)
    clustered_series(
        obs, mean_shifts, ax=largeax, rainfall=rainfall,␣
 ↪temperature=temperature,
        cumsum=True, cl_threshold=cl_threshold, bbox=(1.25, 1.7)
    )

    # vegetation period
    cum = temperature.cumsum()
    vp = cum.where((cum>=0.15* cum.max()) & (cum<=0.9* cum.max())).dropna()
    largeax.fill_between(vp.index.values, 0, 0.05, color='green', alpha=0.3)

    for ax, l in zip(axes.flatten(), ('a', 'b', 'c')):
        ax.annotate(l, xy=(0.03, 0.9), xycoords='axes fraction', fontsize=22)
    largeax.annotate('d', xy=(0.01, 0.9), xycoords='axes fraction', fontsize=22)

    return fig
```

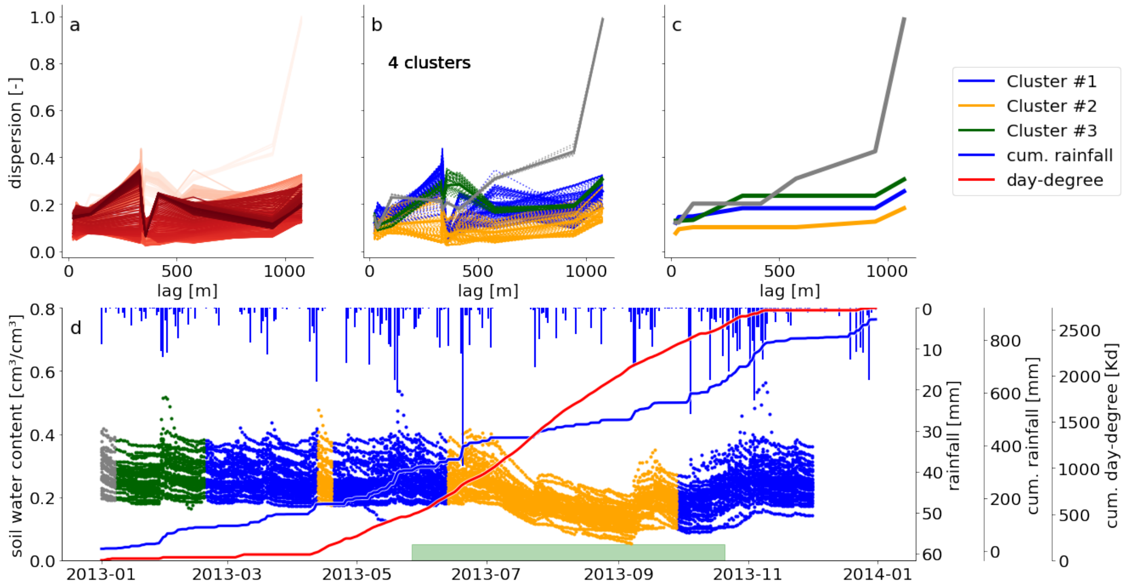
4 2013

Analysis:

[14]:
```
# get data
m2013 = list(extract((m10,m30,m50), '20130101', '20131231'))

# apply analysis
v2013 = variograms(m2013, (npos10, npos30, npos50),␣
 ↪binify='uniform',window=window, estimator=estimator, rank=rank,␣
 ↪maxlag=maxlag)
mean_shifts2013 = cluster_variograms(v2013, bandwidth=bandwidth)
compress2013 = [compress_cluster(ms, v) for ms, v in zip(mean_shifts2013,␣
 ↪v2013)]
```

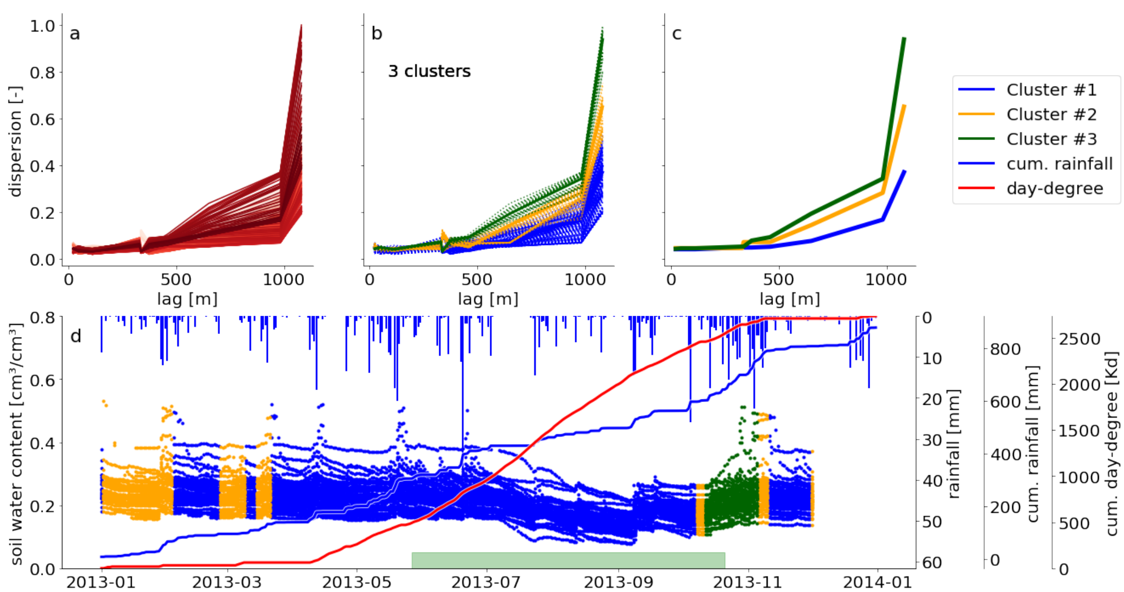
4.1 10 cm

[15]:
```
d = 0 # depth:= 10cm
fig = create_segment_plot(m2013[d], v2013[d], mean_shifts2013[d],␣
 ↪compress2013[d], rr['20130101':'20131231'], temperature['20130101':
 ↪'20131231'])
fig.savefig(main_name % ('10cm', 2013), bbox_inches='tight', dpi=70)
```



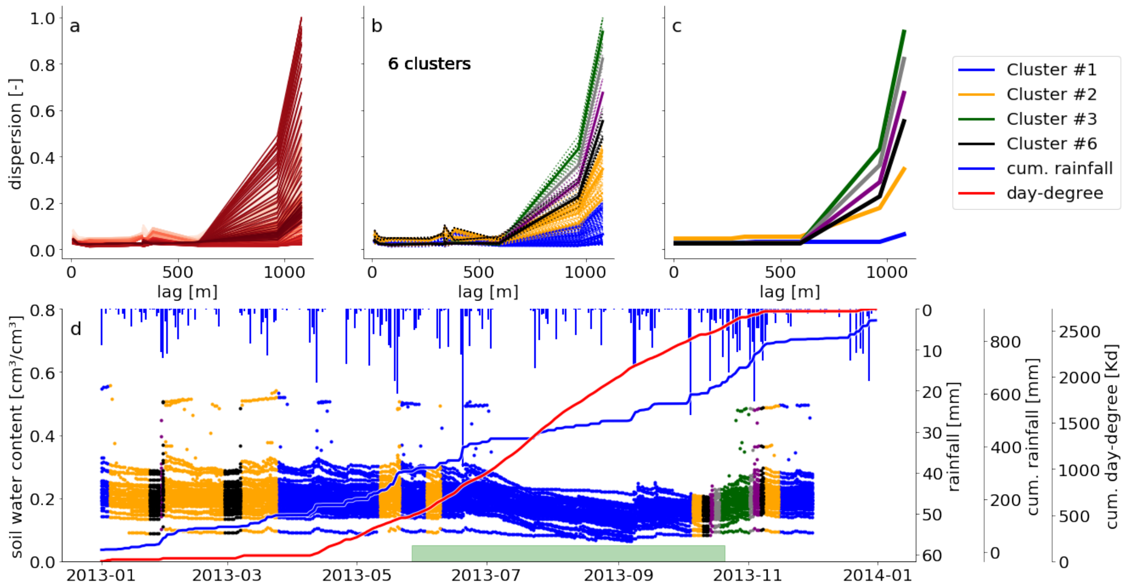
4.2 30 cm

[16]:
```
d = 1 # depth:= 30cm
fig = create_segment_plot(m2013[d], v2013[d], mean_shifts2013[d],␣
↪compress2013[d], rr['20130101':'20131231'], temperature['20130101':
↪'20131231'])
fig.savefig(main_name % ('30cm', 2013), bbox_inches='tight', dpi=70)
```



4.3 50 cm

[17]:
```
d = 2 # depth:= 50cm
fig = create_segment_plot(m2013[d], v2013[d], mean_shifts2013[d],␣
↪compress2013[d], rr['20130101':'20131231'], temperature['20130101':
↪'20131231'])
fig.savefig(main_name % ('50cm', 2013), bbox_inches='tight', dpi=70)
```



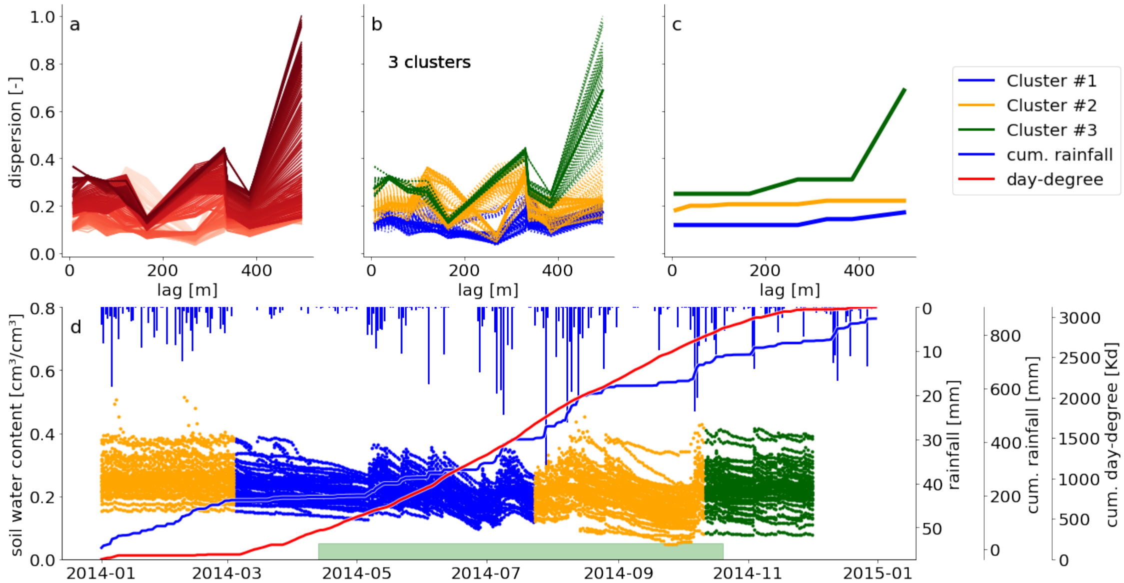
5 2014

[18]:
```
# get data
m2014 = list(extract((m10,m30,m50), '20140101', '20141231'))

# apply analysis
v2014 = variograms(m2014, (npos10, npos30, npos50),␣
↪binify='uniform',window=window, estimator=estimator, rank=rank,␣
↪maxlag=maxlag)
mean_shifts2014 = cluster_variograms(v2014, bandwidth=bandwidth)
compress2014 = [compress_cluster(ms, v) for ms, v in zip(mean_shifts2014,␣
↪v2014)]
```

5.1 10 cm

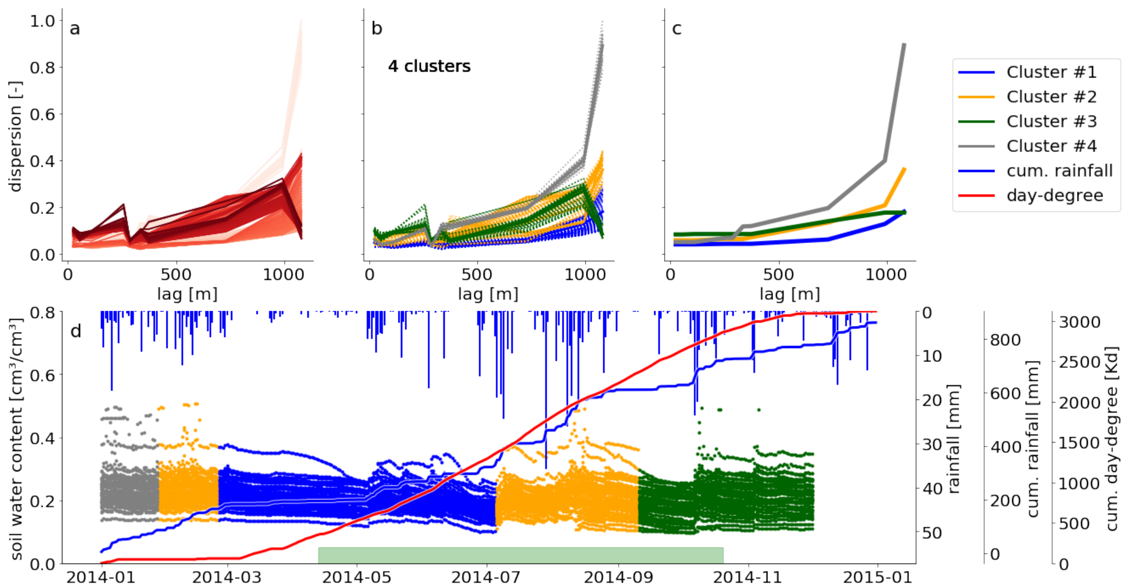
[19]:
```
d = 0 # depth:= 10cm
fig = create_segment_plot(m2014[d], v2014[d], mean_shifts2014[d],␣
↪compress2014[d], rr['20140101':'20141231'], temperature['20140101':
↪'20141231'])
```

```
fig.savefig(main_name % ('10cm', 2014), bbox_inches='tight', dpi=70)
```



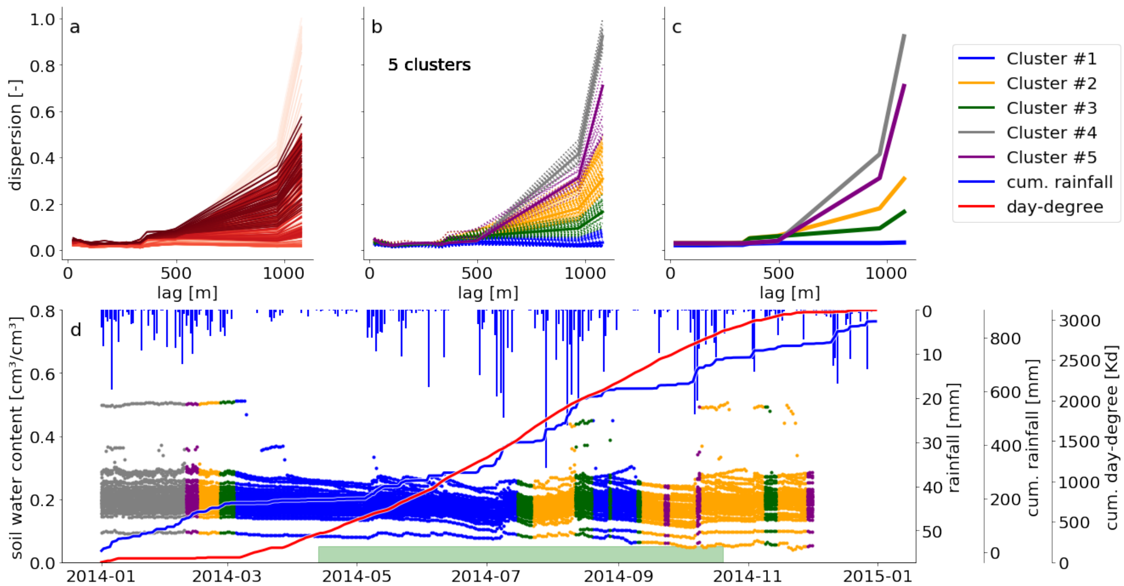
5.2 30 cm

[20]:
```
d = 1 # depth:= 30cm
fig = create_segment_plot(m2014[d], v2014[d], mean_shifts2014[d],␣
↪compress2014[d], rr['20140101':'20141231'], temperature['20140101':
↪'20141231'])
fig.savefig(main_name % ('30cm', 2014), bbox_inches='tight', dpi=70)
```



5.3 50 cm

[21]:
```
d = 2 # depth:= 50cm
fig = create_segment_plot(m2014[d], v2014[d], mean_shifts2014[d],␣
↪compress2014[d], rr['20140101':'20141231'], temperature['20140101':
↪'20141231'])
fig.savefig(main_name % ('50cm', 2014), bbox_inches='tight', dpi=70)
```

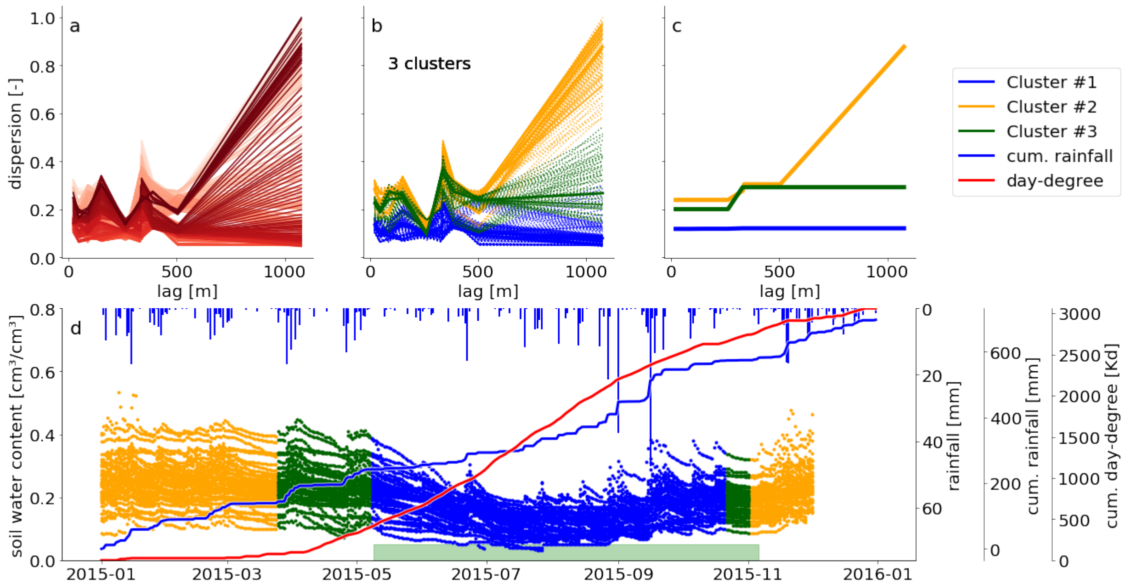


6 2015

[22]:
```
# get data
m2015 = list(extract((m10,m30,m50), '20150101', '20151231'))

# apply analysis
v2015 = variograms(m2015, (npos10, npos30, npos50),␣
↪binify='uniform',window=window, estimator=estimator, rank=rank,␣
↪maxlag=maxlag)
mean_shifts2015 = cluster_variograms(v2015, bandwidth=bandwidth)
compress2015 = [compress_cluster(ms, v) for ms, v in zip(mean_shifts2015,␣
↪v2015)]
```

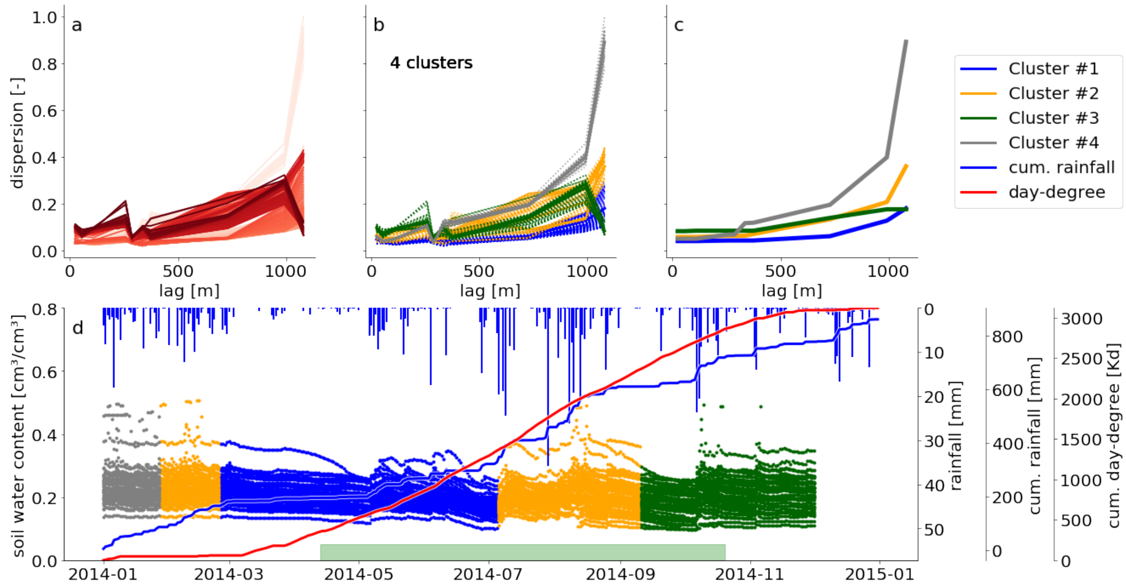
6.1 10 cm

[23]:
```
d = 0 # depth:= 10cm
fig = create_segment_plot(m2015[d], v2015[d], mean_shifts2015[d],␣
↪compress2015[d], rr['20150101':'20151231'], temperature['20150101':
↪'20151231'])
fig.savefig(main_name % ('10cm', 2015), bbox_inches='tight', dpi=70)
```



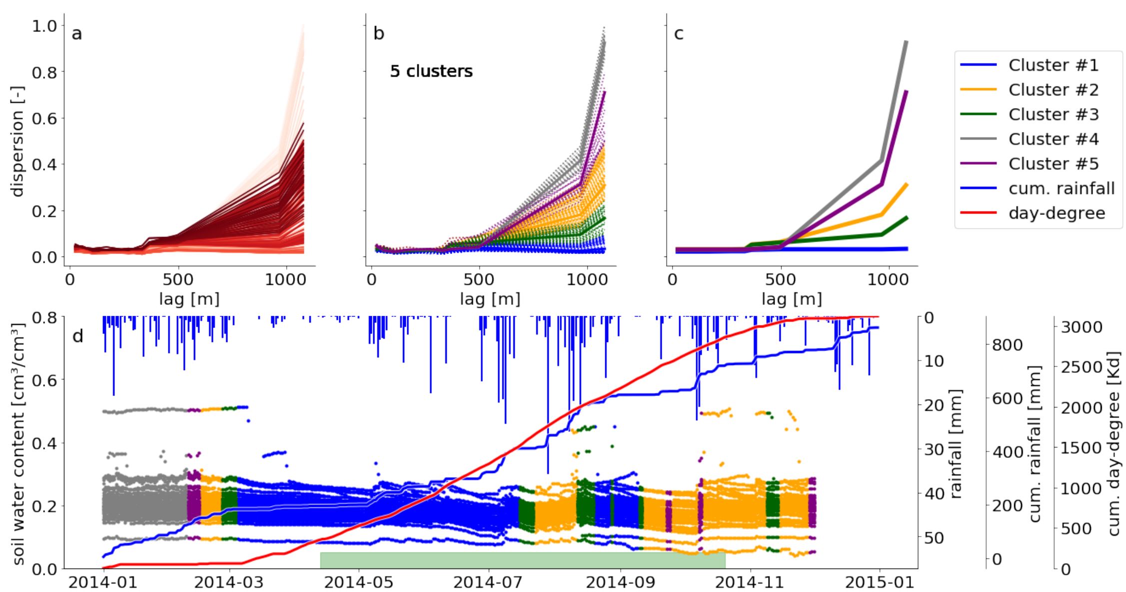
6.2 30 cm

[24]:
```
d = 1 # depth:= 30cm
fig = create_segment_plot(m2014[d], v2014[d], mean_shifts2014[d],␣
↪compress2014[d], rr['20140101':'20141231'], temperature['20140101':
↪'20141231'])
fig.savefig(main_name % ('30cm', 2015), bbox_inches='tight', dpi=70)
```



6.3 50 cm

[25]:
```
d = 2 # depth:= 50cm
fig = create_segment_plot(m2014[d], v2014[d], mean_shifts2014[d],␣
↪compress2014[d], rr['20140101':'20141231'], temperature['20140101':
↪'20141231'])
fig.savefig(main_name % ('50cm', 2015), bbox_inches='tight', dpi=70)
```

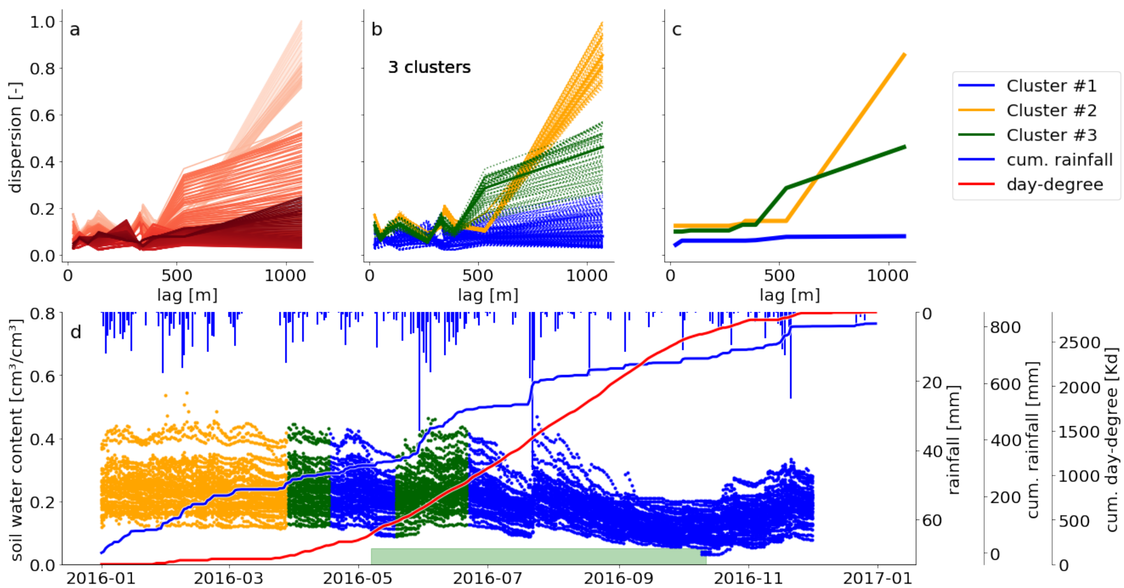


7 2016

[26]:
```
# get data
m2016 = list(extract((m10,m30,m50), '20160101', '20161231'))

# apply analysis
v2016 = variograms(m2016, (npos10, npos30, npos50),␣
↪binify='uniform',window=window, estimator=estimator, rank=rank,␣
↪maxlag=maxlag)
mean_shifts2016 = cluster_variograms(v2016, bandwidth=bandwidth)
compress2016 = [compress_cluster(ms, v) for ms, v in zip(mean_shifts2016,␣
↪v2016)]
```

7.1 10 cm

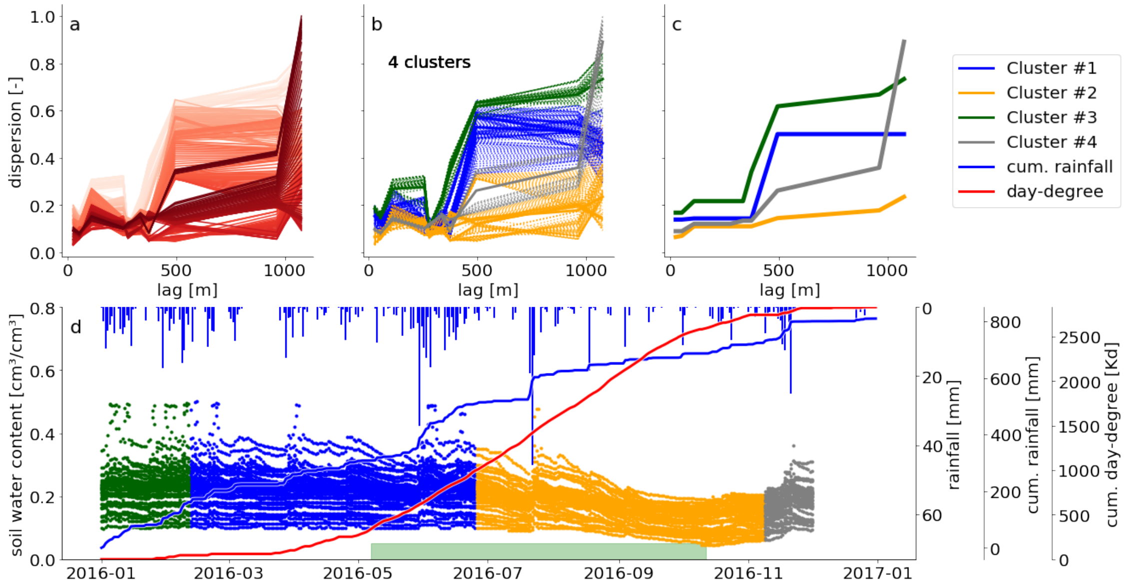
[27]:
```
d = 0 # depth:= 10cm
fig = create_segment_plot(m2016[d], v2016[d], mean_shifts2016[d],␣
↪compress2016[d], rr['20160101':'20161231'], temperature['20160101':
↪'20161231'])
fig.savefig(main_name % ('10cm', 2016), bbox_inches='tight', dpi=70)
```



7.2 30 cm

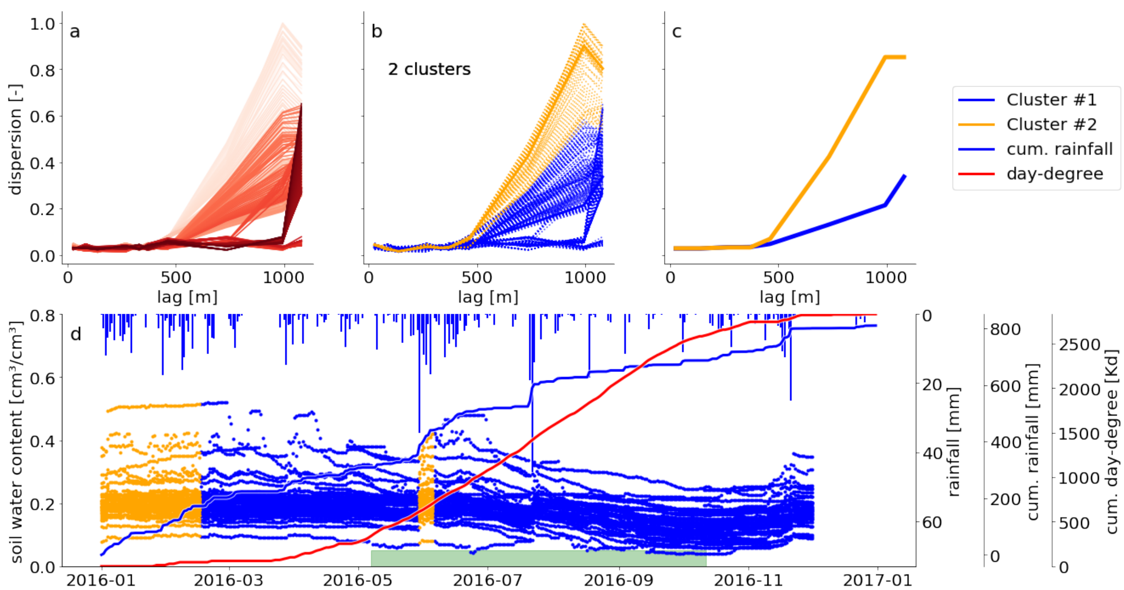
[28]:
```
d = 1 # depth:= 30cm
fig = create_segment_plot(m2016[d], v2016[d], mean_shifts2016[d],␣
↪compress2016[d], rr['20160101':'20161231'], temperature['20160101':
↪'20161231'])
```

```
fig.savefig(main_name % ('30cm', 2016), bbox_inches='tight', dpi=70)
```



7.3 50 cm

[29]:
```
d = 2 # depth:= 50cm
fig = create_segment_plot(m2016[d], v2016[d], mean_shifts2016[d],␣
↪compress2016[d], rr['20160101':'20161231'], temperature['20160101':
↪'20161231'])
fig.savefig(main_name % ('50cm', 2016), bbox_inches='tight', dpi=70)
```



8 Rainfall and Cluster statistics

8.1 rainfall sum and frequency

This section uses the results from 2016 to calculate statistics about the rainfall observations based on the identified periods. The goal was to explain cluster transitions with rainfall statistics.

This part was added due to the fruitfull reviews from the interactive discussion.

[30]:
```
colorcode = {0: 'blue', 1: 'orange', 2: 'darkgreen', 3: 'grey', 4: 'purple', 5:␣
↪'black', 6: 'pink'}
```

Calculate rainfall sums and rainfall frequency within the moving window, to align it with the clusters. This is necessary as the rainfall statistics would otherwise be based on other rainfall observations that are used in the moving window.

[31]:
```
labels = mean_shifts2016[1].labels_
rr2016 = rr['20160101':'20161231']

rr_sums = []
rr_freq = []
res2016 = dict(sums=dict(), freq=dict())

# sums within the cluster
for i in range(0, len(rr2016.index) - window, 1):
    win = rr2016.iloc[i:i+window]
    rr_sums.append(win.dropna().sum())
    rr_freq.append((win > 0).astype(int).sum())

for i in np.unique(labels):
    res2016['sums'][colorcode[i]] = np.mean(np.array(rr_sums)[np.
↪where(labels==i)])
    res2016['freq'][colorcode[i]] = np.mean(np.array(rr_freq)[np.
↪where(labels==i)])

df = pd.DataFrame(data={'moving sums': rr_sums,'moving freq': rr_freq,␣
↪'cluster': labels}, index=rr2016.iloc[:336].index)
```

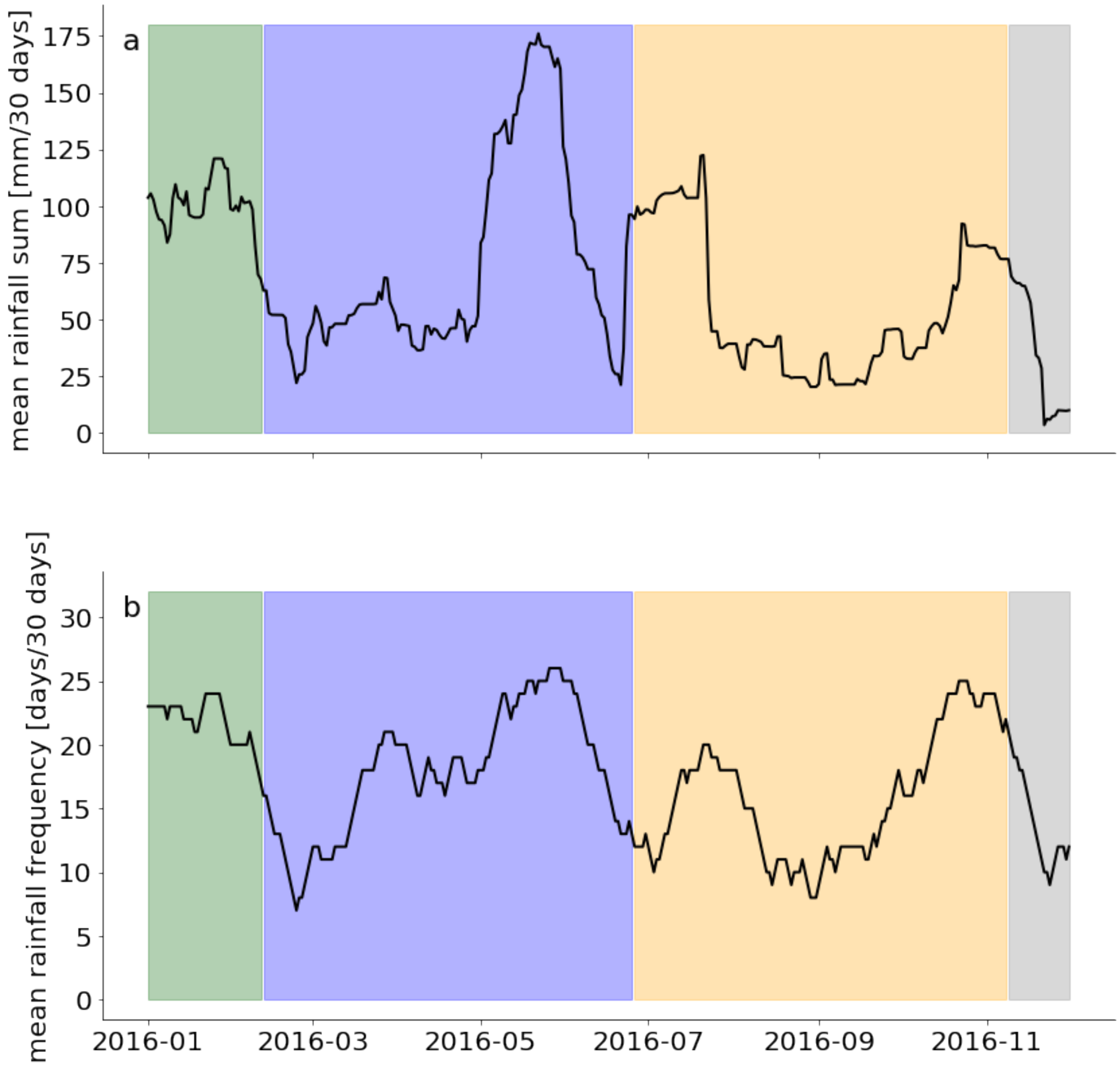
[32]:
```
pprint(res2016)
```

```
{'freq': {'blue': 17.844444444444445,
          'darkgreen': 21.857142857142858,
          'grey': 14.0,
          'orange': 15.772058823529411},
 'sums': {'blue': 73.32000000000001,
          'darkgreen': 100.61904761904762,
```

```
'grey': 35.35869565217392,
'orange': 54.870588235294115}}
```

Show these numbers over time with the active clusters marked in the background.

```
[33]: fig, axes = plt.subplots(2, 1, figsize=(12,12), sharex=True)

# first subplot
axes[0].fill_between(df['moving sums'].where(df.cluster==0).dropna().index,␣
 ↪180, alpha=0.3, color=colorcode[0], edgecolor=None)
axes[0].fill_between(df['moving sums'].where(df.cluster==1).dropna().index,␣
 ↪180, alpha=0.3, color=colorcode[1], edgecolor=None)
axes[0].fill_between(df['moving sums'].where(df.cluster==2).dropna().index,␣
 ↪180, alpha=0.3, color=colorcode[2], edgecolor=None)
axes[0].fill_between(df['moving sums'].where(df.cluster==3).dropna().index,␣
 ↪180, alpha=0.3, color=colorcode[3], edgecolor=None)
art = axes[0].plot(df.index, df['moving sums'], lw=2, color='k', label="mean")
axes[0].set_ylabel('mean rainfall sum [mm/30 days]')

# second subplot
axes[1].fill_between(df['moving freq'].where(df.cluster==0).dropna().index, 32,␣
 ↪alpha=0.3, color=colorcode[0], edgecolor=None)
axes[1].fill_between(df['moving freq'].where(df.cluster==1).dropna().index, 32,␣
 ↪alpha=0.3, color=colorcode[1], edgecolor=None)
axes[1].fill_between(df['moving freq'].where(df.cluster==2).dropna().index, 32,␣
 ↪alpha=0.3, color=colorcode[2], edgecolor=None)
axes[1].fill_between(df['moving freq'].where(df.cluster==3).dropna().index, 32,␣
 ↪alpha=0.3, color=colorcode[3], edgecolor=None)
art = axes[1].plot(df.index, df['moving freq'], lw=2, color='k', label="mean")
axes[1].set_ylabel('mean rainfall frequency [days/30 days]')

# marker
axes[0].annotate('a', xy=(0.02, 0.9), xycoords='axes fraction', fontsize=22)
axes[1].annotate('b', xy=(0.02, 0.9), xycoords='axes fraction', fontsize=22)

plt.tight_layout()

fig.savefig(main_name % ('rolling_compare', 2016), bbox_inches='tight', dpi=70)
```



8.2 cluster durations

The follwoing code creates an overview table about all cluster durations (for all years as reference).

[34]:
```
durations = {2013: dict(), 2014: dict(), 2015: dict(), 2016: dict()}

for meanshifts, year in␣
↪zip((mean_shifts2013,mean_shifts2014,mean_shifts2015,mean_shifts2016),␣
↪(2013, 2014, 2015, 2016)):
    for ms, depth in zip(meanshifts, ('10cm', '30cm', '50cm')):
        durations[year][depth] = {colorcode[k]:v for k,v in Counter(ms.labels_).
↪items()}
```

[35]:
```
pprint(durations)
```

```
{2013: {'10cm': {'blue': 171, 'darkgreen': 42, 'grey': 7, 'orange': 115},
        '30cm': {'blue': 248, 'darkgreen': 25, 'orange': 62},
        '50cm': {'black': 19,
                 'blue': 198,
                 'darkgreen': 14,
                 'grey': 5,
                 'orange': 93,
                 'purple': 6}},
 2014: {'10cm': {'blue': 141, 'darkgreen': 51, 'orange': 143},
        '30cm': {'blue': 131, 'darkgreen': 82, 'grey': 27, 'orange': 95},
        '50cm': {'blue': 153,
                 'darkgreen': 31,
                 'grey': 40,
                 'orange': 98,
                 'purple': 13}},
 2015: {'10cm': {'blue': 167, 'darkgreen': 55, 'orange': 113},
        '30cm': {'blue': 127, 'darkgreen': 83, 'orange': 125},
        '50cm': {'blue': 241, 'darkgreen': 32, 'orange': 62}},
 2016: {'10cm': {'blue': 194, 'darkgreen': 54, 'orange': 88},
        '30cm': {'blue': 135, 'darkgreen': 42, 'grey': 23, 'orange': 136},
        '50cm': {'blue': 282, 'orange': 54}}}
```

8.3 cluster variability

How do the cluster members vary? Use cressie-hawkins and the Shannon entropy to calculate dispersion function variability per distance lag class across all cluster members for each cluster.

[36]:
```
from skgstat.estimators import cressie
from skinfo import entropy

var = {'10cm': dict(), '30cm': dict(), '50cm': dict()}
info = {'10cm': dict(), '30cm': dict(), '50cm': dict()}
for i, v, ms, d in zip((0, 1, 2), v2016, mean_shifts2016, ('10cm', '30cm',␣
 ↪'50cm')):
    _v = [_.experimental for _ in v]
    bins = np.linspace(np.min(_v), np.max(_v), 25)
    for c in np.unique(ms.labels_):
        _var = np.apply_along_axis(cressie, 0, np.array(_v)[np.where(ms.
 ↪labels_==c)])
        _inf = np.apply_along_axis(lambda x: entropy(x, bins), 0, np.
 ↪array(_v)[np.where(ms.labels_==c)])

        var[d][colorcode[c]] = np.mean(_var)
        info[d][colorcode[c]] = np.mean(_inf)
```

Using Cressie-Hawkins:

[37]:
```
pprint(var)
```

```
{'10cm': {'blue': 5.455663731878889e-06,
          'darkgreen': 4.362666733553778e-05,
          'orange': 0.00011219350574327655},
 '30cm': {'blue': 2.161149090814767e-05,
          'darkgreen': 4.39520274944583e-05,
          'grey': 2.4053851281802403e-05,
          'orange': 4.207678253281902e-06},
 '50cm': {'blue': 3.509880150737812e-05, 'orange': 0.000363864649867988}}
```

Using Shannon Entropy:

[38]:
```
pprint(info)
```

```
{'10cm': {'blue': 1.3432649900516511,
          'darkgreen': 1.2480315643549917,
          'orange': 0.6933201479479684},
 '30cm': {'blue': 1.8264678997928763,
          'darkgreen': 1.3097442982424856,
          'grey': 0.9566838469706622,
          'orange': 1.6301938122220883},
 '50cm': {'blue': 0.9879358660635822, 'orange': 0.899269434023096}}
```

Does not show anything. This is not too surprising as Mean shift will minimize these variabilities. Maybe can be used to assess the compression rate?

9 Combined Plots

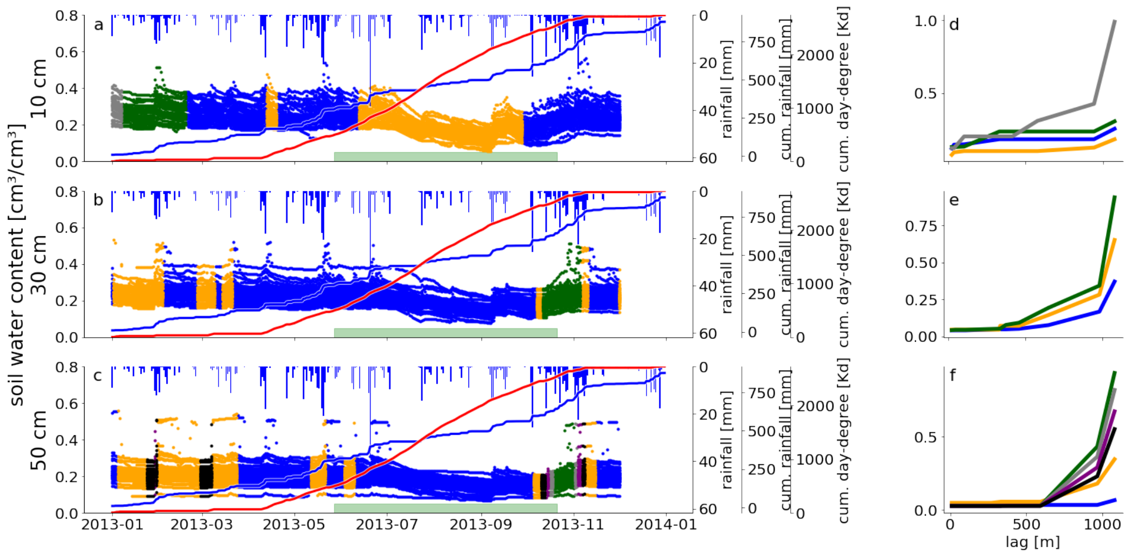
9.1 Depth

[39]:
```
# Helper function for depth compare plots
def depth_compare_plot(obs, mean_shifts, compressed, rainfall, temperature):
    # create figure
    fig = plt.figure(figsize=(4*6.1, 2*6))
    axes = []
    clus = []

    axes.append(plt.subplot2grid((3, 5), (0, 0), colspan=3))
    axes.append(plt.subplot2grid((3, 5), (1, 0), colspan=3))
    axes.append(plt.subplot2grid((3, 5), (2, 0), colspan=3))
    clus.append(plt.subplot2grid((3, 5), (0, 4), colspan=1))
    clus.append(plt.subplot2grid((3, 5), (1, 4), colspan=1))
    clus.append(plt.subplot2grid((3, 5), (2, 4), colspan=1))

    # plot
    for d, ax in zip((0, 1, 2, ), axes):
        clustered_series(
            obs[d], mean_shifts[d], ax=ax, rainfall=rainfall,␣
 ↪temperature=temperature,
```

```
            cumsum=True, cl_threshold=cl_threshold, legend=None
        )

    # vegetation period
    cum = temperature.cumsum()
    vp = cum.where((cum>=0.15* cum.max()) & (cum<=0.9* cum.max())).dropna()

    for ax, l in zip(axes, ('a', 'b', 'c')):
        ax.fill_between(vp.index.values, 0, 0.05, color='green', alpha=0.3)
        ax.annotate(l, xy=(0.015, 0.9), xycoords='axes fraction', fontsize=22)

    for ax, l in zip(clus, ('d', 'e', 'f')):
        ax.annotate(l, xy=(0.03, 0.9), xycoords='axes fraction', fontsize=22)


    plot_compressed(compressed[0][0], compressed[0][1], ax=clus[0],␣
↪ylabel=False, xlabel=False)
    plot_compressed(compressed[1][0], compressed[1][1], ax=clus[1],␣
↪ylabel=False, xlabel=False)
    plot_compressed(compressed[2][0], compressed[2][1], ax=clus[2],␣
↪ylabel=False)

    axes[0].set_ylabel('10 cm', fontsize=26)
    axes[0].set_xticklabels([])
    clus[0].set_xticklabels([])
    axes[1].set_ylabel('soil water content [cm³/cm³]\n30 cm', fontsize=26)
    axes[1].set_xticklabels([])
    clus[1].set_xticklabels([])
    axes[2].set_ylabel('50 cm', fontsize=26)

    return fig
```

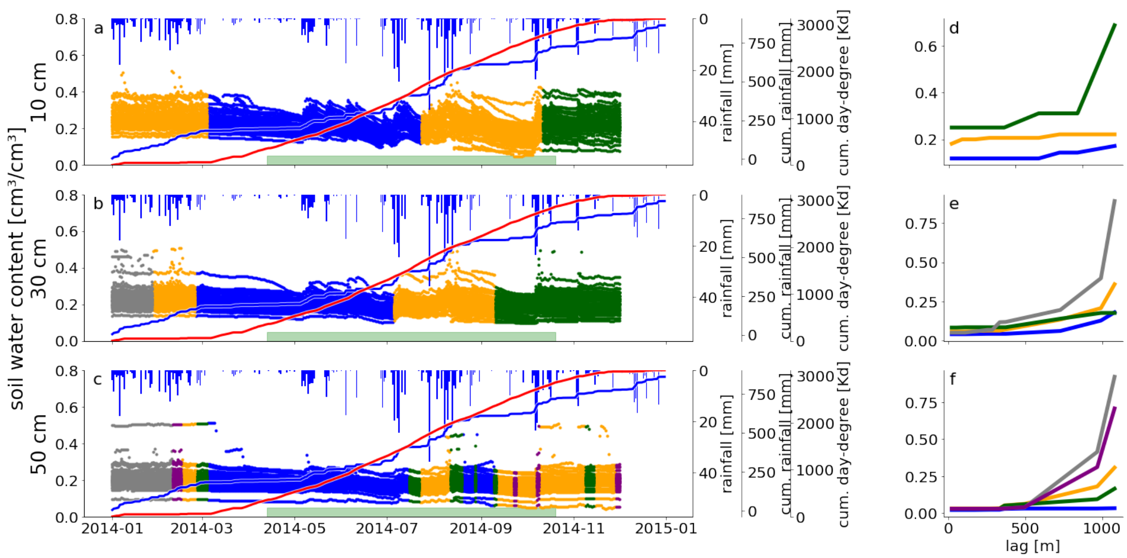
9.1.1 2013

[40]:
```
fig = depth_compare_plot(m2013, mean_shifts2013, compress2013, rr['20130101':
↪'20131231'], temperature['20130101':'20131231'])
fig.savefig(main_name % ('depth_compare', 2013), bbox_inches='tight', dpi=70)
```



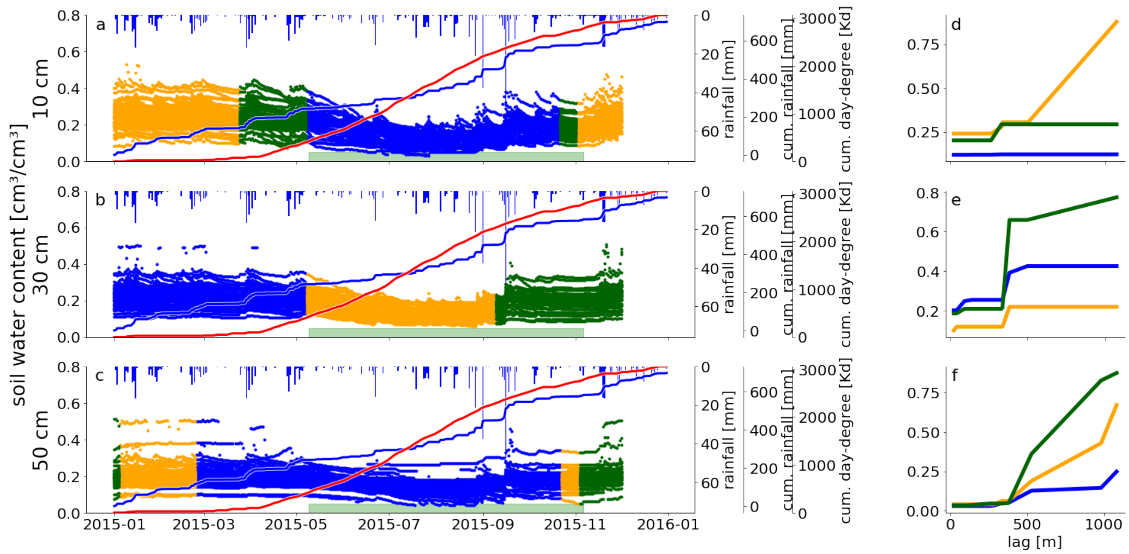
9.1.2 2014

[41]:
```
fig = depth_compare_plot(m2014, mean_shifts2014, compress2014, rr['20140101':
↪'20141231'], temperature['20140101':'20141231'])
fig.savefig(main_name % ('depth_compare', 2014), bbox_inches='tight', dpi=70)
```



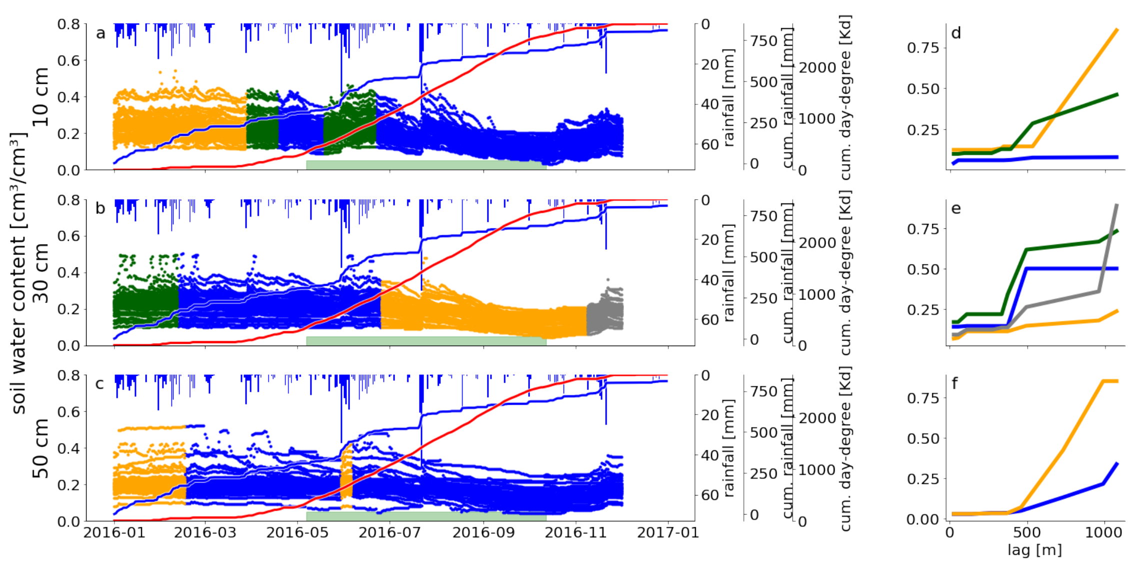
9.1.3 2015

[42]:
```
fig = depth_compare_plot(m2015, mean_shifts2015, compress2015, rr['20150101':
↪'20151231'], temperature['20150101':'20151231'])
fig.savefig(main_name % ('depth_compare', 2015), bbox_inches='tight', dpi=70)
```



9.1.4 2016

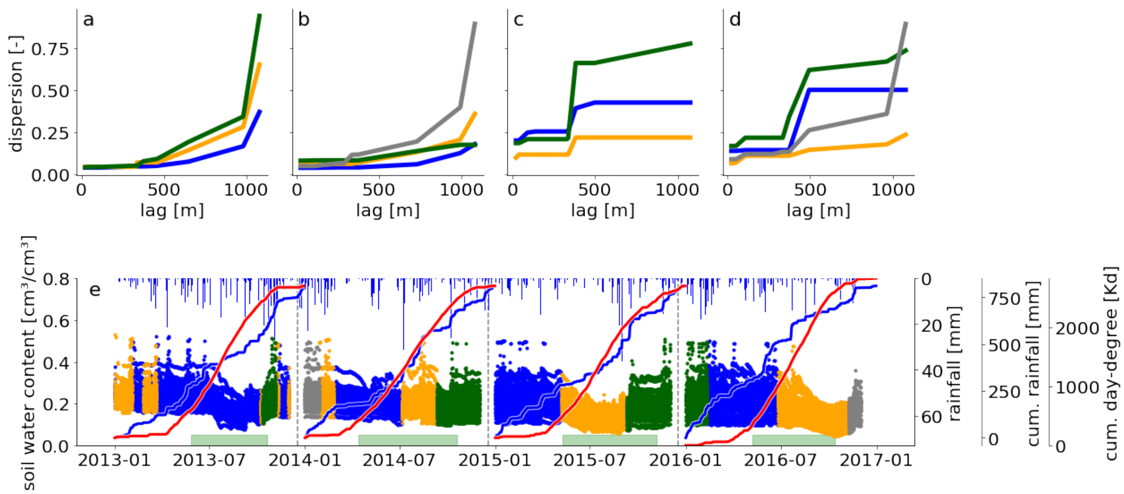
[43]:
```
fig = depth_compare_plot(m2016, mean_shifts2016, compress2016, rr['20160101':
↪'20161231'], temperature['20160101':'20161231'])
fig.savefig(main_name % ('depth_compare', 2016), bbox_inches='tight', dpi=70)
```



9.2 Inter-annual

[44]:
```
# prepare data - 30cm of all years
moistures = (m2013[1], m2014[1], m2015[1], m2016[1],)
clusters = (mean_shifts2013[1], mean_shifts2014[1], mean_shifts2015[1],␣
↪mean_shifts2016[1])

# create figure
axes = []
fig = plt.figure(figsize=(18, 2*4))
axes.append(plt.subplot2grid((2,4), (0, 0), colspan=1))
axes.append(plt.subplot2grid((2,4), (0, 1), colspan=1, sharey=axes[0]))
axes.append(plt.subplot2grid((2,4), (0, 2), colspan=1, sharey=axes[0]))
axes.append(plt.subplot2grid((2,4), (0, 3), colspan=1, sharey=axes[0]))
ax = plt.subplot2grid((2, 4), (1, 0), colspan=4)
#fig, ax = plt.subplots(1, 1, figsize=(18, 1*4), sharex=True, sharey=True)


years = (2013, 2014, 2015, 2016)
# plot
for _moisture, _ms, y in zip(moistures, clusters, years):
    show = y == 2016
    clustered_series(_moisture, _ms, ax=ax, rainfall=rr['%d0101' % y:'%d1231' %␣
↪y], temperature=temperature['%d0101' % y:'%d1231' % y],
                     cumsum=True, cl_threshold=cl_threshold, legend=None,␣
↪show_spines=show)
    # vegetation period
    cum = temperature['%d0101' % y:'%d1231' % y].cumsum()
    vp = cum.where((cum>=0.15* cum.max()) & (cum<=0.9* cum.max())).dropna()
    ax.fill_between(vp.index.values, 0, 0.05, color='green', alpha=0.3)

ax.vlines([dt(2013, 12, 18), dt(2014, 12, 18), dt(2015, 12, 18)], 0, 0.8,␣
↪linestyle='--', color='grey')

# plot centroids
plot_compressed(compress2013[1][0], compress2013[1][1], ax=axes[0],␣
↪ylabel=True, xlabel=True)
plot_compressed(compress2014[1][0], compress2014[1][1], ax=axes[1],␣
↪ylabel=False, xlabel=True)
plot_compressed(compress2015[1][0], compress2015[1][1], ax=axes[2],␣
↪ylabel=False, xlabel=True)
plot_compressed(compress2016[1][0], compress2016[1][1], ax=axes[3],␣
↪ylabel=False, xlabel=True)

# hide labels
```

```python
for a in axes[1:]:
    plt.setp(a.get_yticklabels(), visible=False)

# annotate
for a, l in zip(axes, ('a', 'b', 'c', 'd')):
    a.annotate(l, xy=(0.03, 0.9), xycoords='axes fraction', fontsize=22)
ax.annotate('e', xy=(0.015, 0.9), xycoords='axes fraction', fontsize=22)

plt.tight_layout()
fig.savefig(main_name % ('interannual', 1316), bbox_inches='tight', dpi=70)
```



9.3 Uncertainty propagation

9.3.1 At one example

Only for 30cm depth in 2016:

This part was develeped very early while we were still developing the method. It was kept for reference and to foster a better understanding of the 'For all data' part, that actually was used in the paper

Define the functions with propagated uncertainties, then calculate the *uncertainty distance matrix*. That means, how big is the uncetainty in dispersion function distance for all possbile combinations of all dispersion functions ever calculated?

[45]:
```python
variograms = (v2013, v2014, v2015, v2016)
dz = 0.02

def cressie_err(x):
    N = len(x)
    return 0.5 * (0.457+(1/N)+(0.045 / N**2))**-1 * 2 * ((1 / N) * np.sum(np.
↪sqrt(np.abs(x))))**3 * (1 / N) * np.sqrt(np.sum(np.abs(x)**1))
```

```
def d_err(u, v, du, dv):
    return (0.5 * (  np.sum(np.abs(u-v)**-0.5) * np.sum((2*(u-v)*du)**2 +␣
 ↪(2*(u-v) * dv)**2) ) )**0.5

def vario_err(v, dz):
    return np.fromiter((cressie_err(c) * np.sqrt(2) * dz for c in v.
 ↪lag_classes()), dtype=float)

def varios_err(variograms, dz):
    return np.asarray([vario_err(v, dz=dz) for v in variograms])
```

[46]:
```
variogram_errors = list()

for v_year in variograms:
    variogram_errors.append([varios_err(v_depth, dz=dz) for v_depth in v_year])
```

[47]:
```
# errors of one depth in one year
_v2016_10 = variograms[3][1]
_v2016_10_err = variogram_errors[3][1]



dm_v = pdist([_.experimental for _ in _v2016_10])
dm_err = np.ones(dm_v.shape) * -1

k = 0
for i in range(len(_v2016_10_err)):
    for j in range(len(_v2016_10_err)):
        if i > j:
            dm_err[k] = d_err(_v2016_10[i].experimental, _v2016_10[j].
 ↪experimental, _v2016_10_err[i], _v2016_10_err[j])
            k += 1
```

[48]:
```
print(np.mean(dm_v), np.min(dm_v), np.max(dm_v))
print(np.mean(_v2016_10_err), np.min(_v2016_10_err), np.max(_v2016_10_err))
print(np.mean(dm_err), np.min(dm_err), np.max(dm_err))
```

```
0.007256814650269572 2.0498315605701888e-05 0.017369805407659145
3.0382558527904338e-05 5.080227017303131e-06 0.00015026920248325287
1.7624505670933738e-05 7.146598451708756e-08 0.00026782699941538
```

9.3.2 For all data

First, extract all experimental variograms and use the error propagation functions developed to propagate the uncertainty into the distance used for clustering

```
[49]: all_vs = list()
all_errs = list()

for v, err in zip(variograms, variogram_errors):
    all_vs.extend([_.experimental for _ in v[0]])
    all_vs.extend([_.experimental for _ in v[1]])
    all_vs.extend([_.experimental for _ in v[2]])
    all_errs.extend(err[0])
    all_errs.extend(err[1])
    all_errs.extend(err[2])
assert len(all_vs) == len(all_errs)

dm = pdist(all_vs)
dme = np.ones(dm.shape) * -1

k = 0
for i in range(len(all_errs)):
    for j in range(len(all_errs)):
        if i > j:
            dme[k] = d_err(all_vs[i], all_vs[j], all_errs[i], all_errs[j])
            k += 1
```

```
[50]: print('Maximum dispersion function distance: ', np.max(dm))
print('Maximum uncertainty of distance:      ', np.max(dme))
bins = np.arange(0, np.max(dm), np.max(dme))
#bins = np.arange(0, np.max(dm), np.percentile(dme, 95))
n = len(bins)
print('Information Entropy number of bins:   ', n)
print(bins)
print('Maximum Entropy of binning at uniform distribution: %.2f' %␣
 ↪entropy(bins, bins))
```

```
Maximum dispersion function distance:  0.06902437182593772
Maximum uncertainty of distance:       0.005726514471981985
Information Entropy number of bins:    13
[0.         0.00572651 0.01145303 0.01717954 0.02290606 0.02863257
 0.03435909 0.0400856  0.04581212 0.05153863 0.05726514 0.06299166
 0.06871817]
Maximum Entropy of binning at uniform distribution: 3.55
```

9.4 The information loss due to compression

Extract the necessary information from the result objects above. Then substitute all cluster members by their centroid function and calculate the kullback-leibler divergence between compressed and uncompressed clusters.

[51]:
```
variograms = (v2013, v2014, v2015, v2016)
shifts = (mean_shifts2013, mean_shifts2014, mean_shifts2015, mean_shifts2016)

def extract_all_vectors(d, variograms):
    result = list()
    for variogram in variograms:
        result.extend([_.experimental for _ in variogram[d]])
    return result

import skinfo
from scipy.spatial.distance import pdist
def information_loss(variograms, mean_shift, bins=None, cl_threshold=10):
    _d = np.asarray([v.experimental for v in variograms])
    data = pdist(_d)
    if bins is None:
        bins = 15
    if isinstance(bins, int):
        bins = np.linspace(0, np.max(data), bins)
    intr = []
    for cl in np.unique(mean_shift.labels_):
        mem = data[np.where(mean_shift.labels_ == cl), ][0]
        if len(mem) > cl_threshold:
            intr.extend(len(mem) * [mean_shift.cluster_centers_[cl]])
    compressed = pdist(np.asarray(intr))

    return skinfo.kullback_leibler(compressed, data, bins), skinfo.
 ↪entropy(data, bins)

all10cm = extract_all_vectors(1, variograms)


res = dict()
res['KL'] = dict()
res['N'] = dict()
res['H'] = dict()
res['H2'] = dict()
res['rel'] = dict()

for year, tup in zip((2013, 2014, 2015, 2016), zip(variograms, shifts)):
    v, ms = tup
    kld10, e10 = information_loss(v[0], ms[0], bins=bins)
    kld30, e30 = information_loss(v[1], ms[1], bins=bins)
    kld50, e50 = information_loss(v[2], ms[2], bins=bins)
    res['KL'][year] = {
        '10cm': kld10,
        '30cm': kld30,
        '50cm': kld50
```

```
    }
    res['N'][year] = {
        '10cm': len(np.unique(ms[0].labels_)),
        '30cm': len(np.unique(ms[1].labels_)),
        '50cm': len(np.unique(ms[2].labels_))
    }
    res['H'][year] = {
        '10cm': e10,
        '30cm': e30,
        '50cm': e50
    }
    res['H2'][year] = {
        '10cm': 2**e10,
        '30cm': 2**e30,
        '50cm': 2**e50
    }
    res['rel'][year] = {
        '10cm': kld10 / (e10 + kld10),
        '30cm': kld30 / (e30 + kld30),
        '50cm': kld50 / (e50 + kld50)
    }
```

[52]:
```
pprint(res)
```

```
{'H': {2013: {'10cm': 0.9661027934368046,
              '30cm': 1.4904779231660545,
              '50cm': 1.995448506253},
       2014: {'10cm': 1.352079889231375,
              '30cm': 1.5700497874754031,
              '50cm': 2.439567526212624},
       2015: {'10cm': 1.8749813753202447,
              '30cm': 1.1787810943484325,
              '50cm': 2.389513733502411},
       2016: {'10cm': 2.4903082942419292,
              '30cm': 1.4404685000048163,
              '50cm': 3.212315259499693}},
 'H2': {2013: {'10cm': 1.953556245764517,
               '30cm': 2.809820409601468,
               '50cm': 3.9874004650713513},
        2014: {'10cm': 2.5527988949713216,
               '30cm': 2.9691496049860637,
               '50cm': 5.4247908873873065},
        2015: {'10cm': 3.667968820340402,
               '30cm': 2.2638542748680504,
               '50cm': 5.239807218462751},
        2016: {'10cm': 5.61898011298258,
               '30cm': 2.7140898838789975,
```

```
                 '50cm': 9.26836756339506}},
 'KL': {2013: {'10cm': 0.43654060776174186,
               '30cm': 0.06097410024469685,
               '50cm': 0.1348528365806574},
        2014: {'10cm': 0.21714232128892708,
               '30cm': 0.29953681207818406,
               '50cm': 0.28141571589748704},
        2015: {'10cm': 0.1750225808234085,
               '30cm': 0.08704300076566018,
               '50cm': 0.9030205791483923},
        2016: {'10cm': 0.7630951916639441,
               '30cm': 0.022220351491858414,
               '50cm': 2.5047954420446326}},
 'N': {2013: {'10cm': 4, '30cm': 3, '50cm': 6},
       2014: {'10cm': 3, '30cm': 4, '50cm': 5},
       2015: {'10cm': 3, '30cm': 3, '50cm': 3},
       2016: {'10cm': 3, '30cm': 4, '50cm': 2}},
 'rel': {2013: {'10cm': 0.3112270783783831,
                '30cm': 0.039301312141544566,
                '50cm': 0.06330223516701187},
         2014: {'10cm': 0.1383757633770235,
                '30cm': 0.16021553221963952,
                '50cm': 0.10342427382215347},
         2015: {'10cm': 0.08537670393214784,
                '30cm': 0.06876389942459954,
                '50cm': 0.2742630731830931},
         2016: {'10cm': 0.23455289052518763,
                '30cm': 0.01519144107041068,
                '50cm': 0.4381226064711373}}}
```

The last cell will transform the dictionary above into a latex table.

[53]:
```
for year in (2013, 2014, 2015, 2016):
    for d in ('10cm', '30cm', '50cm'):
        print(year, '&', d[:2] + ' \\unit{cm}', '&',
              res['N'][year][d], '&',
              round(res['H'][year][d], 2), '&',
              round(res['H2'][year][d], 2), '&',
              round(res['KL'][year][d], 2), '&',
              round(res['rel'][year][d], 2),
              '\\\\')
```

```
2013 & 10 \unit{cm} & 4 & 0.97 & 1.95 & 0.44 & 0.31 \\
2013 & 30 \unit{cm} & 3 & 1.49 & 2.81 & 0.06 & 0.04 \\
2013 & 50 \unit{cm} & 6 & 2.0 & 3.99 & 0.13 & 0.06 \\
2014 & 10 \unit{cm} & 3 & 1.35 & 2.55 & 0.22 & 0.14 \\
2014 & 30 \unit{cm} & 4 & 1.57 & 2.97 & 0.3 & 0.16 \\
2014 & 50 \unit{cm} & 5 & 2.44 & 5.42 & 0.28 & 0.1 \\
```

```
2015 & 10 \unit{cm} & 3 & 1.87 & 3.67 & 0.18 & 0.09 \\
2015 & 30 \unit{cm} & 3 & 1.18 & 2.26 & 0.09 & 0.07 \\
2015 & 50 \unit{cm} & 3 & 2.39 & 5.24 & 0.9 & 0.27 \\
2016 & 10 \unit{cm} & 3 & 2.49 & 5.62 & 0.76 & 0.23 \\
2016 & 30 \unit{cm} & 4 & 1.44 & 2.71 & 0.02 & 0.02 \\
2016 & 50 \unit{cm} & 2 & 3.21 & 9.27 & 2.5 & 0.44 \\
```