locus

May 9, 2021

1 Storm Centering

by: John Kucharksi | correspondance: johnkucharski@gmail.com | date: 06 May 2021

The notebook analyzes the spatial patterns of annaul daily maximum precipitation. It performs this analysis on the North Branch of the Potomac Watershed, using a dataset constructed from the Livneh data.[1] This dataset is constructed using by the imports.py contained in this module.

The analysis in this notebook: 1. generates a single datafile for analysis, 2. explores two different normalization routines, 3. projects the normalized data across its first n prinicpal components, 4. clusters the data (projected onto its first n principal components) around k-Means, 5…N subsquent steps will help us visualize, explore the results of the normalization, pca and clustering…

References: [1] Livneh B., E.A. Rosenberg, C. Lin, B. Nijssen, V. Mishra, K.M. Andreadis, E.P. Maurer, and D.P. Lettenmaier, 2013: A Long-Term Hydrologically Based Dataset of Land Surface Fluxes and States for the Conterminous United States: Update and Extensions, Journal of Climate, 26, 9384–9392.

[1]:
```
import statistics
import numpy as np
import pandas as pd
import geopandas as gpd
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

from kneed import KneeLocator

import dataimports.livneh as livneh
```

1.1 Data

The data being analyzed includes the annual maximum day of precipitation from 1915 through 2011 for the North Branch of the Potomac Watershed in Western Maryland, USA. The data for each of these 97 days (between 1915 and 2011) contains the preciptation depth for all 130 livneh grid cells located within or intersected by the North Branch of the Potomac 8 digit Hydrologic Unit Code (HUC08) boundary.

```
[2]: years = list(range(1915, 2012))
     importpath: str = '/Users/johnkucharski/Documents/source/locus/data/
      ↪livneh_1day_max/'
```

Grids Geometry This geometry of the 130 livneh grids clipped to the North Banch of the Potomac Watershed, is shown below. This is used for plotting.

```
[3]: points = pd.read_csv(importpath + 'prec.1915.csv')
     points = points[['id', 'lat', 'lon']]
     grids = livneh.points2grids(livneh.convert2geodataframe(points))
     grids = grids.sort_values(by = 'id')
     grids.head(2)
```

```
[3]:      id       lat       lon                                           geometry
     125   0  39.96875 -78.84375  POLYGON ((-78.87500 40.00000, -78.81250 40.000…
     126   1  39.96875 -78.78125  POLYGON ((-78.81250 40.00000, -78.75000 40.000…
```

Annual Maximum Gridded Precipitation Data Below is the view of dataset containing the gridded daily annual maximum precipitation data. It is a 97 x 130 (year x grid cell) matrix. The sum of a row's columns (i.e. event) gives the total precipitation depth for the watershed on the day (row) being summed, the sum of a columns row's gives the total precipitation recieved in that grid cell over the 97 days covered in the 97 year instrumental record.

```
[4]: df = livneh.aggregateprocessedfiles(importpath, years).sort_index()
     df.rename_axis(None, inplace = True) #can't seem to get rid of the 'id' above␣
      ↪the index
     df.head(2)
```

```
[4]: id         0       1       2       3       4      5      6       7      8   \
     1915  33.325  33.125  35.525  32.975  34.525  35.85  34.55  34.650  34.05
     1916  16.300  18.550  19.300  18.150  18.425  21.95  19.60  18.775  17.85

     id         9  …    120     121     122     123     124     125   126    127  \
     1915  33.675  …  33.05  32.300  34.700  35.475  41.950  24.550  30.3  30.95
     1916  17.450  …  40.30  35.775  34.625  34.200  35.275  24.825  31.6  29.70

     id       128     129
     1915  38.200  41.325
     1916  31.325  29.950

     [2 rows x 130 columns]
```

1.2 Methods

The primary goal of this study is to identify significant patterns in the spatial distribution of extreme precipitation events for the North Branch of the Potomac Watershed. A secondary goal is to prescribe these patterns to some hypothetical drivers: (1) orographics, (2) seasonal atmospheric flow patterns associated with extreme precipitation, and (3) storm types (i.e. midlatitude cyclone, tropical cyclone).

To achieve these goals the data is: (1) normalized, then (2) a Principal Component Analysis (PCA) is performed, finally (3) the 97 observations are clustered around the principal components (identified in step 2).

1.2.1 1. Normalization Routines

The data must be normalized, otherwise outliers will dominate the principal component analysis and clustering. The data can reasonably be expected to contain outliers for several reasons:

1. Event Magnitudes - the events being analyzed represent annaul maximum days of precipitation. Therefore, to one degree or another all of the events being analyzed are 'outliers' from the perspective of the underlying precipitation distribution. Maximum annual precipitation values like these are typically fit to an extreme values distribution (EVD), used to model observations sampled from the tail of some other distribution (such as a gamma distribution of daily rainfall). The EVDs model the asympotic behavior of the under distributions tail (or tails), therefore we should expect our 97 year sample to exhibit some of this asymptotic behanvior.
2. Spatial Outliers - one would expect precipitation totals to be higher at higher elevations, as adiabatic cooling forces more moisture to rain out of the air. This orographic effect is likely to lead to some grid cells (or columns) with substaintially larger means and variability (perhaps). Secondly, (I think) extreme preciptiation events over a "large" area, like the size of North Branch of the Potomac Watershed, to be dominated by advective synoptical (i.e. large) scale events. These synoptic scale events are driven by specific patterns of atmopheric flow (cite Schlef). We seek to test if this mode of spatial variability drives different clusterings in the data (see secondary goals above).

Two normalization schemes are explored below. For simplicity they are refered to as: (1) a "nieve" routine, and (2) a "hypothesis-based" routine. Both normalization routines normalize the data using the equation:

```
(x - $\mu$) / s
```

where x is the observed rainfall total for the cell in the dataset, μ is the mean of the data being normalized, and s is the standard deviation (of the data being normalized). The methods differ primarily in the data used to measure μ and s.

a. Nieve Routine The "nieve" normalization routine applies the normalization equation to all rows and columns simultaneously. Therefore, the mean: $/mu represents the average livneh grid cell total across all grid cells and events. For instance, a value of 2 after this normalization routine indicates that precipitaiton is two standard deviation above the mean - in that grid cell, relative to all grid cells and all events. This value might be product of: (a) an anomolously large event - in which case a dispropotionate share of the grid cells in that row would have postive values; on the other hand, (b) the value could be representative of a typically wet grid cells (due to orgographics or other factors) - in which case a disproportionate share of the cells in that column would have positive values; or (c) it could indicate some combination of the two (an anomolously large event and anomolously wet grid cell). This normalization scheme provide an emperical view of the data.

A programming note: The original data is 97 rows (years) x 130 columns (grid cells). I want the PCA to reduce the time or observational dimension (find a more parsimonious pattern that describes the 'types' annual max events). So, (I think) I have to transpose the dataset because the PCA will reduce the dimensionality of the columns (reduce the number of columns). After transposing I have 130 rows (grid cells) x 97 columns (years)

[5]:
```
nieve_std = StandardScaler().fit_transform(df.to_numpy()).T
# generates a normalized dataset a 130 (grid cells) x 97 (columns) numpy array
# this seems to drop the index values which is what we want.
print(nieve_std.shape)
```

```
(130, 97)
```

b. Hypothesis-based Routine The "hypothesis-based" normalization routine is a two step process.

1. Events (or rows) of data are normalized. In this step, */mu* represents an average grid cell's precipitation during that event. The resulting values express the livneh grid cell's precipitation in deviations from the mean grid cell precipitaiton, during that event. For example, a value of 2 after this normalization scheme would indicate that a livneh grid cell contains a precipitation total which was 2 standard deviations above the mean grid cell total for that particular row's event.

TODO: I think it could be valuable to also consider clustering pcs generated from this dataset, since this should capture the combined orographic + atmospheric flow patterns of precipitation.

[6]:
```
def standardize_events(df: pd.DataFrame) -> pd.DataFrame:
    '''
    Normalizes the row data using the formula: (x - u) / s,
    where x is a value in one of the row's columns, u is the row mean and s is␣
 ↪the row standard deviation.
    Assumes each row contains an list of the grid cell precipitaiton values for␣
 ↪a particular event or year.

    The resulting dataframe reports precipitation values for each grid cell in␣
 ↪terms of unit variance for that event's grid cell values.
    The idea is that this normalization capture both normal orographically␣
 ↪influenced spatial patterns as well as spatial characteristics of the storm.

    If these values are averaged across all events or years it should provide␣
 ↪information about the normal (orographically influenced) spatial patterns in␣
 ↪the rainfall.
    '''
    new_df = pd.DataFrame(columns = df.columns)
    for index, row in df.iterrows():
        data = list(row)
        u = statistics.mean(data)
        s = statistics.stdev(data)
```

```
        new_row: list = []
        for x in data:
            new_row.append((x - u) / s)
        new_df.loc[index] = new_row
    return new_df
```

2. Columns (or livneh grid cell) values are normalized. In the example above, I hypothesize that the value of 2 (or whatever value is found) may not be as anomolious as it would seem on face value. Perhaps, the grid cell is located in a zone of extreme orographic lift, and as a result it tends to recieve much more than an average grid cell amount of rain - across all 97 days in the analysis. In this case, the value of 2 may be an average value for that grid cell to help disentangle the orographic and storm centering patterns impact on rainfall totals we normalize this column of data. If in fact the value 2 in the first step was a local (i.e. grid cell average) we wil be left with a data set that describes the deviations from this localized average in standard deviation unit. For example, now a value of 2 would represent an anomolously high rainfall total for that grid cell based on its average across all event in the period of analysis.

[7]:
```
def standardize_grids(df: pd.DataFrame) -> pd.DataFrame:
    '''
    Normalizes the column data using the formula: (x - u) / s,
    where x is a value in a row of one of the columns, u is the column mean and␣
↪s is the column standard deviation.
    Assumes each column contains an list of a grid cell precipitaiton values␣
↪for all the events or years of interest.

    If the events have been standardized then this will report precipitation␣
↪values for each grid cell as deviations (of unit variance) of that specific␣
↪grid cell's normalized portion of the event total.
    The idea is that this process of standardizing by event and then␣
↪standardizing by grid cell should provide deviations from the normal␣
↪(oragraphically influenced) spatial characteristics of rainfall patterns in␣
↪the watershed.

    If the events have NOT been standarized first then the standarized results␣
↪will be heavily influenced by the size of the event, rather than the spatial␣
↪characteristics fo the storm.
    '''
    new_df = pd.DataFrame(index = df.index)
    for name, col in df.iteritems():
        data = list(col)
        u = statistics.mean(data)
        s = statistics.stdev(data)
        new_col: list = []
        for x in data:
            new_col.append((x - u) / s)
        new_df[name] = new_col
```

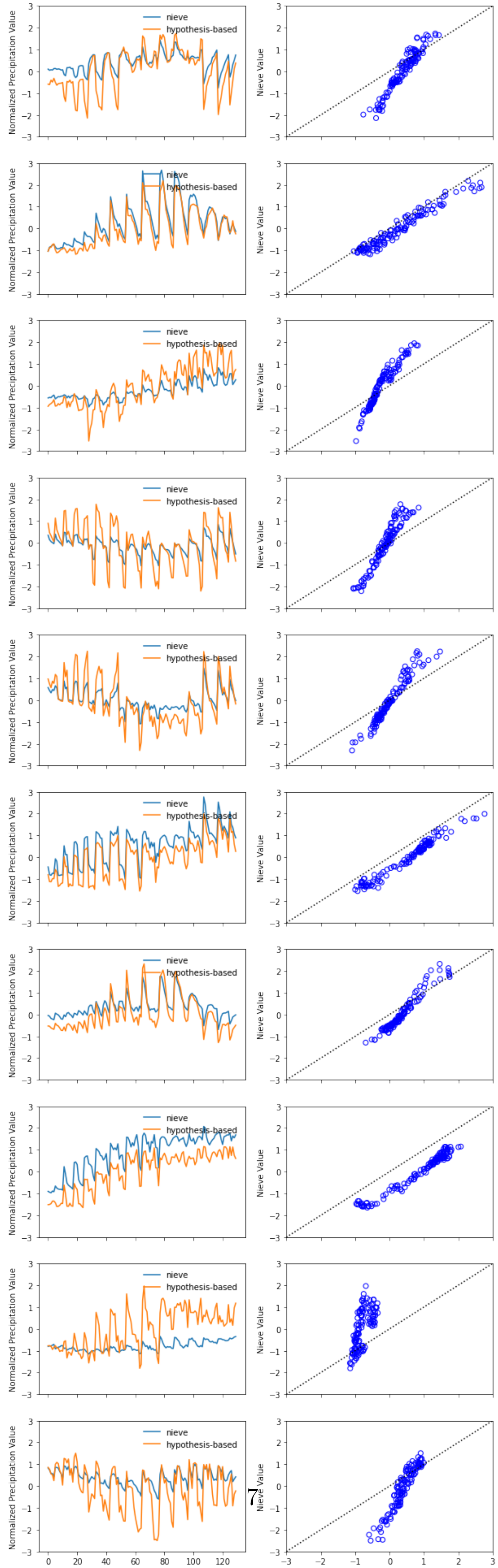
```
    return new_df
```

[8]:
```
hypothesis_std = standardize_grids(standardize_events(df)).to_numpy().T
print(hypothesis_std.shape)
```

```
(130, 97)
```

c. Comparison of Normalization Routines The plots below explore the coorelation between the nieve and hypothesis-based routines results. This comparision is carried forward in subsequent steps of the analysis.

[9]:
```
fig, axs = plt.subplots(nrows=10, ncols=2, figsize=(10, 35), sharex='col')
row, col = 0, 0
# plots the fist 10 years of data
for i in range(0, 10):
    col = 0
    axs[row, col].set_ylim([-3, 3])
    axs[row, col].plot(nieve_std[:,i], label = 'nieve')
    axs[row, col].plot(hypothesis_std[:,i], label = 'hypothesis-based')
    axs[row, col].set_ylabel('Normalized Precipitation Value')
    leg = axs[row, col].legend(loc='upper right', frameon=False)
    col += 1
    axs[row, col].set_xlim([-3, 3])
    axs[row, col].set_ylim([-3, 3])
    axs[row, col].scatter(nieve_std[:,i], hypothesis_std[:,i],␣
 ↪facecolors='none', edgecolors='blue')
    axs[row, col].plot([-3, 3], [-3, 3], 'k:')
    axs[row, col].set_ylabel('Nieve Value')
    row += 1
axs[9, 0].set_xlabel('Grid Cell Id')
axs[9, 1].set_xlabel('Hypothesis Value')
# lines, labels = fig.axes[-1].get_legend_handles_labels()
# fig.legend(lines, labels, loc = 'upper center')
plt.show()
```



nieve
hypothesis-based
Normalized Precipitation Value
Nieve Value
Grid Cell Id
Hypothesis Value

1.2.2 2. PCA Analysis

Two principal component analyses are performed.

1. The PCA is performed on the time (or observational) dimension of both normalized data sets. This should identify some event patterns, that describe variability in the spatial distribution of precipitation across the 97 events. Actual patterns of events may incorporate more than one of these patterns, these are identfied with the k-Means clustering algorithm.
2. To prepare the data for the k-Means clustering the PCA analysis is performed on the spatial (i.e. grid cell) dimension of both normalized data sets. This reduces the dimensionality of the events (by summarizing variabiltiy acorss the 130 livneh grid cells), improving the clustering algorithm.

a. Performed on "Nieve"ly normalized data

[10]:
```
nieve_pca = PCA(n_components=20)
nieve_pcs = nieve_pca.fit_transform(StandardScaler().fit_transform(nieve_std))
```

[11]:
```
df_nieve_pcs = pd.DataFrame(nieve_pcs)
df_nieve_pcs.head(2)
```

[11]:
```
          0         1         2         3         4         5         6  \
0 -6.586659 -6.769666 -4.119799 -0.181315 -1.989221 -0.409279 -0.282502
1 -6.952877 -6.836459 -3.523584 -0.129390 -2.005440 -0.811883 -1.049697

          7         8         9        10        11        12        13  \
0 -2.626674 -0.073455 -0.334871  0.788295  0.712648 -1.014148  1.009108
1 -2.875417 -1.292171  0.100133 -0.907063  0.604568 -0.153377 -0.072425

         14        15       16        17        18        19
0 -0.708411  1.579860 -0.15129  0.400870 -0.724544  0.488351
1 -1.156030  1.869879 -0.23928  0.257174 -1.060651  0.189715
```

[12]:
```
# now if I take each column and attach the grid cell geometries to it I should␣
↪be able to plot it again
nieve_grids = grids.copy(deep = True)
for i in range(0, nieve_pcs.shape[1]):
    nieve_grids['pc_' + str(i + 1)] = nieve_pcs[:,i]
nieve_grids.head(1)
```

[12]:
```
     id       lat       lon  \
125   0  39.96875 -78.84375

                                              geometry      pc_1      pc_2  \
125  POLYGON ((-78.87500 40.00000, -78.81250 40.000… -6.586659 -6.769666
```

```
        pc_3      pc_4      pc_5      pc_6  …     pc_11     pc_12  \
125 -4.119799 -0.181315 -1.989221 -0.409279  …  0.788295  0.712648

       pc_13     pc_14     pc_15     pc_16     pc_17     pc_18     pc_19  \
125 -1.014148  1.009108 -0.708411  1.57986 -0.15129  0.40087 -0.724544

       pc_20
125  0.488351

[1 rows x 24 columns]
```

b. Performed on "Hypothesis-based" normalized data

[13]:
```
hypothesis_pca = PCA(n_components=20)
hypothesis_pcs = hypothesis_pca.fit_transform(StandardScaler().
 ↪fit_transform(hypothesis_std))
```

[14]:
```
df_hypothesis_pcs = pd.DataFrame(hypothesis_pcs)
df_hypothesis_pcs.head(2)
```

[14]:
```
          0         1         2         3         4         5         6   \
0 -6.489586 -5.574379  3.060878 -1.008519  2.222517  0.236691 -0.450607
1 -6.807569 -5.413299  2.537163 -0.938396  2.476195  0.191514 -0.871971

          7         8         9         10        11        12        13  \
0 -2.417624  0.519472 -0.256399  0.355994 -1.155114  1.134794  0.732806
1 -1.773113  0.847755 -1.911194  1.217832 -0.938905  1.114320 -0.367808

          14        15        16        17        18        19
0 -0.538917  0.887238  1.065609  0.681716  0.182249  0.793632
1 -0.538994  0.910948  0.895752  0.696188  0.363692  1.059839
```

[15]:
```
hypothesis_grids = grids.copy(deep = True)
for i in range(0, hypothesis_pcs.shape[1]):
    hypothesis_grids['pc_' + str(i + 1)] = hypothesis_pcs[:,i]
hypothesis_grids.head(1)
```

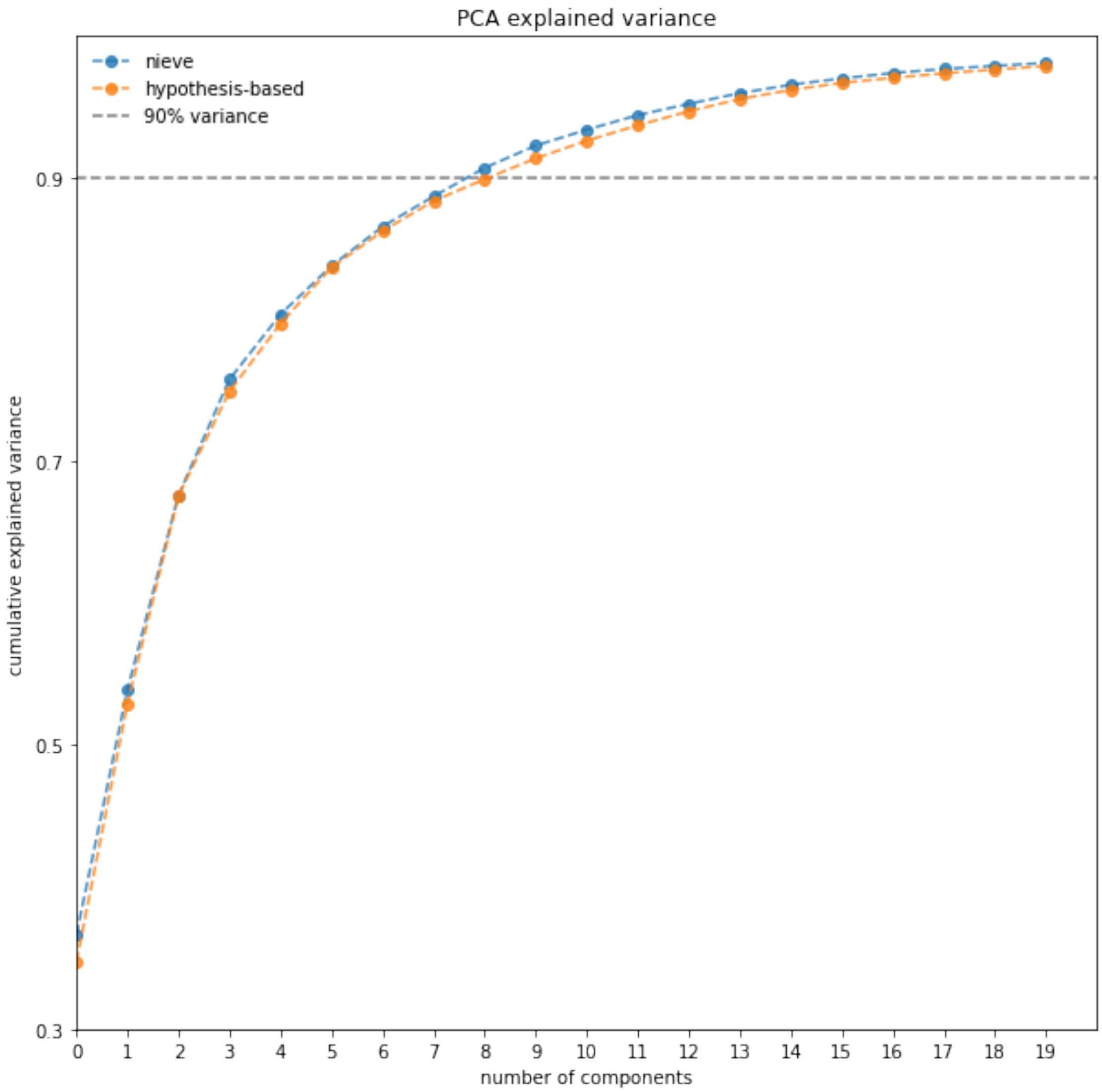
[15]:
```
     id       lat       lon  \
125   0  39.96875 -78.84375

                                              geometry      pc_1      pc_2  \
125  POLYGON ((-78.87500 40.00000, -78.81250 40.000… -6.489586 -5.574379

        pc_3      pc_4      pc_5      pc_6  …     pc_11     pc_12  \
125  3.060878 -1.008519  2.222517  0.236691  …  0.355994 -1.155114
```

```
        pc_13     pc_14     pc_15     pc_16     pc_17     pc_18     pc_19  \
125  1.134794  0.732806 -0.538917  0.887238  1.065609  0.681716  0.182249

        pc_20
125  0.793632

[1 rows x 24 columns]
```

c. Comparision of "Nieve" and "Hypothesis-based" PCA The figures below compare the principal component analysis under the 'nieve' and 'hypothesis-based' normalization routines. The first 2 principal components explain more than half the variation in the data. 3 principal components under either normalization routine explain almost 70% of the variation in the data, 9 principal components are required to explain 90% of the variation in the data.

[16]:

```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10, 10), sharex = True,␣
 ↪sharey = True)
ax.plot(np.cumsum(nieve_pca.explained_variance_ratio_), marker='o',␣
 ↪linestyle='dashed', alpha=0.8, label = 'nieve')
ax.plot(np.cumsum(hypothesis_pca.explained_variance_ratio_), marker='o',␣
 ↪linestyle='dashed', alpha=0.8, label = 'hypothesis-based')
ax.hlines(y=0.9, xmin=0, xmax=20, linestyle = 'dashed', color = 'black',␣
 ↪alpha=0.5, label = '90% variance')
ax.set_xlabel('number of components')
ax.set_ylabel('cumulative explained variance')
ax.set_title('PCA explained variance')
ax.set_xlim([0, 20])
ax.set_ylim([0.3, 1.0])
plt.yticks(np.arange(0.3, 1.0, 0.2))
plt.xticks(np.arange(0, 20, 1.0))
plt.legend(frameon=False)
plt.show()
```



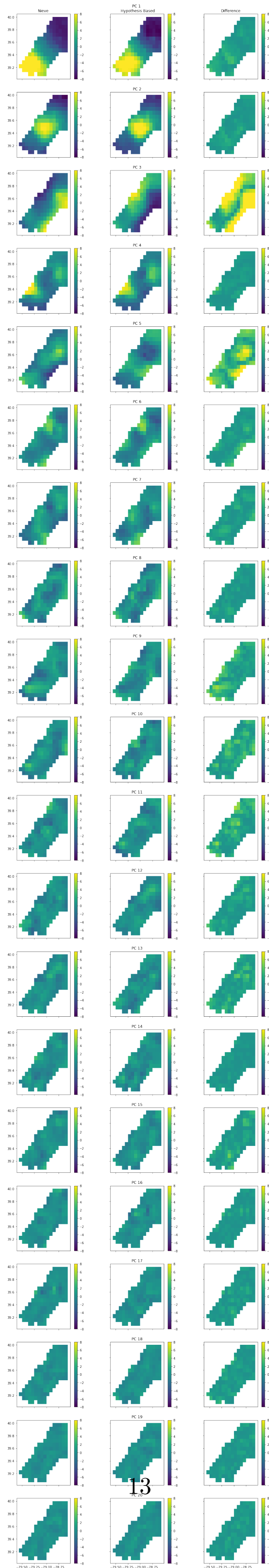

```
[17]: difference_grids = grids.copy(deep = True)
      difference_pcs = np.absolute(nieve_pcs - hypothesis_pcs)
      for i in range(0, difference_pcs.shape[1]):
          difference_grids['pc_' + str(i + 1)] = difference_pcs[:,i]
      difference_grids.head(1)
```

```
[17]:      id       lat       lon  \
      125   0  39.96875 -78.84375

                                                  geometry      pc_1      pc_2  \
      125  POLYGON ((-78.87500 40.00000, -78.81250 40.000…  0.097074  1.195287

               pc_3      pc_4      pc_5      pc_6  …      pc_11      pc_12      pc_13  \
```

```
125  7.180677  0.827204  4.211738  0.64597  …  0.432301  1.867762  2.148942

         pc_14     pc_15     pc_16     pc_17     pc_18     pc_19     pc_20
125  0.276302  0.169494  0.692622  1.216899  0.280847  0.906794  0.305281

[1 rows x 24 columns]
```

The first 2 principal components display obvious spatial patterns (see below): * PC1 show anamolously high rainfall in the southern region, * PC2 shows anomolously high precipitation in the central region of the watershed. * PC3 shows opposite patterns depending on the normolizaiton routine. Under the nieve approach, precipitation is anolomously high along the eastern edge of the watershed, under the hypothesis based approch it is anomolously high along the western watershed boundary.

[18]:
```
fig, axs = plt.subplots(nrows = 20, ncols = 3, figsize = (15, 90), sharex =␣
↪True, sharey = True)
for i in range(0, 20):
    col_name = 'pc_' + str(i + 1)
    nieve_grids.plot(ax = axs[i, 0], column = col_name, vmin=-8, vmax=8, legend␣
↪= True)
    hypothesis_grids.plot(ax = axs[i, 1], column = col_name, vmin=-8, vmax=8,␣
↪legend = True)
    difference_grids.plot(ax = axs[i, 2], column = col_name, vmin=-8, vmax=8,␣
↪legend = True)
    if i == 0:
        axs[i, 0].set_title('Nieve')
        axs[i, 1].set_title('PC 1 \n Hypothesis Based')
        axs[i, 2].set_title('Difference')
    else:
        axs[i, 1].set_title('PC ' + str(i + 1))
```

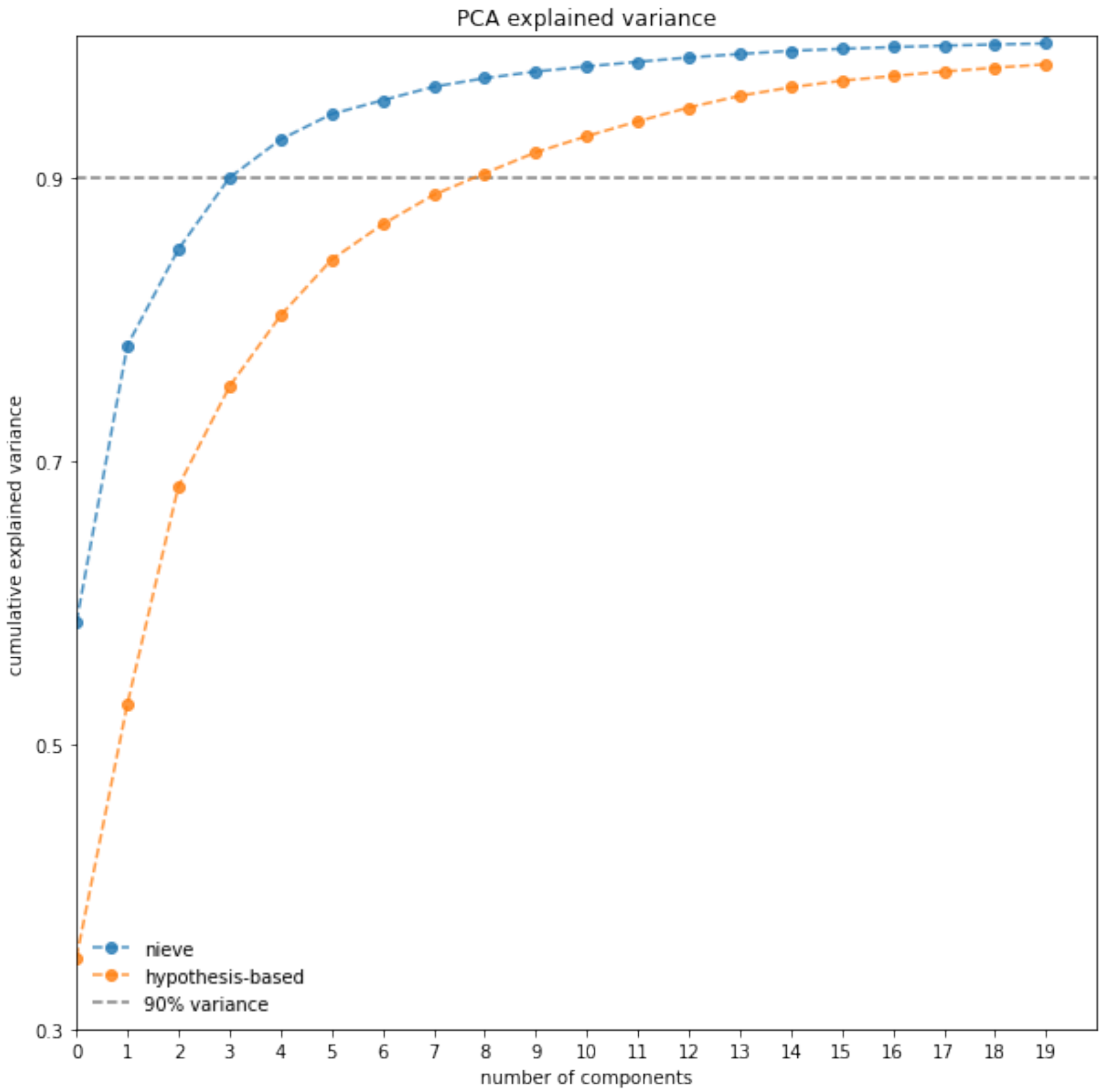

PC 1
Naive
Hypothesis Based
Difference
PC 2
PC 3
PC 4
PC 5
PC 6
PC 7
PC 8
PC 9
PC 10
PC 11
PC 12
PC 13
PC 14
PC 15
PC 16
PC 17
PC 18
PC 19

d. Spatial (grid cell) dimensionality reduction This subsection performs a PCA to reduce the dimenionality of the spatial (grid cell) data. This is not as easly to visualize since it reduces the 130 grid cells to a set of 20 PCAs (rathan reducing the number of events).

Here the 'nieve' normalization routine lead to a dramatically more effecient PCA. Only 4 PCs are required to the explain 90% of the variation, whereas 9 are required when using the 'hypothesis-based' approach. (It'd be great to understand why).

[19]:
```
nieve_spatial_pca = PCA(n_components=20)
nieve_spatial_pcs = nieve_spatial_pca.fit_transform(StandardScaler().
 ↪fit_transform(nieve_std.T))

hypothesis_spatial_pca = PCA(n_components=20)
hypothesis_spatial_pcs = hypothesis_spatial_pca.fit_transform(StandardScaler().
 ↪fit_transform(hypothesis_std.T))
```

[20]:
```
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10, 10), sharex = True,␣
 ↪sharey = True)
ax.plot(np.cumsum(nieve_spatial_pca.explained_variance_ratio_), marker='o',␣
 ↪linestyle='dashed', alpha=0.8, label = 'nieve')
ax.plot(np.cumsum(hypothesis_spatial_pca.explained_variance_ratio_),␣
 ↪marker='o', linestyle='dashed', alpha=0.8, label = 'hypothesis-based')
ax.hlines(y=0.9, xmin=0, xmax=20, linestyle = 'dashed', color = 'black',␣
 ↪alpha=0.5, label = '90% variance')
ax.set_xlabel('number of components')
ax.set_ylabel('cumulative explained variance')
ax.set_title('PCA explained variance')
ax.set_xlim([0, 20])
ax.set_ylim([0.3, 1.0])
plt.yticks(np.arange(0.3, 1.0, 0.2))
plt.xticks(np.arange(0, 20, 1.0))
plt.legend(frameon=False)
plt.show()
```

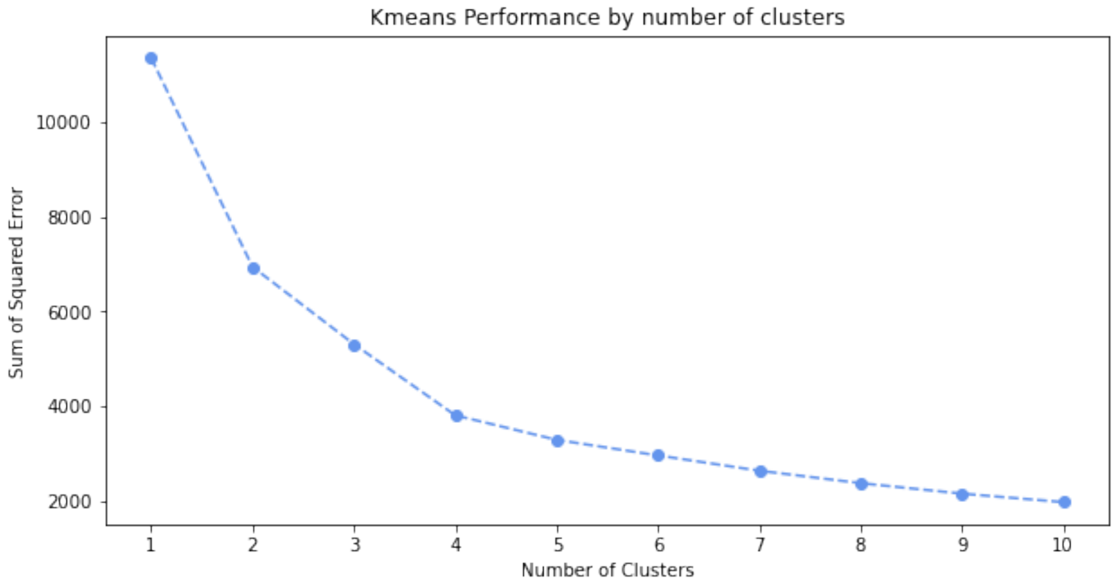


1.3 3. Clustering

Somehow the sum of squared error increases (foreach value of k) for larger number of PCs. I don't understand it, but for this reason I use only the first 4 PCs.

[21]:
```
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters = k)
    kmeans.fit(nieve_spatial_pcs[:,:4])
    sse.append(kmeans.inertia_)
fig, ax = plt.subplots(nrows = 1, ncols = 1, figsize = (10, 5))
ax.plot(range(1, 11), sse, marker = 'o', linestyle = 'dashed', color =␣
 ↪'cornflowerblue')
```

```
ax.set_xticks(list(range(1, 11)))
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('Sum of Squared Error')
ax.set_title('Kmeans Performance by number of clusters')
plt.show()
```



A needle algorithm from kneed is used to identify an 'elbow' in the function. This is a point of diminishing returns, in this case for the the 'k', sse relationship. This can also be checked visually, so in this case the kneed algorithm confirms what is the figure demonstrates.

[22]:
```
kl = KneeLocator(range(1, 11), sse, curve="convex", direction="decreasing")
print(kl.elbow)
```

4

[23]:
```
kmeans = KMeans(n_clusters = 4)
clusters = kmeans.fit(nieve_spatial_pcs[:,:4])
cluster_labels = clusters.labels_
df['cluster'] = cluster_labels
df.head(2)
```

[23]:

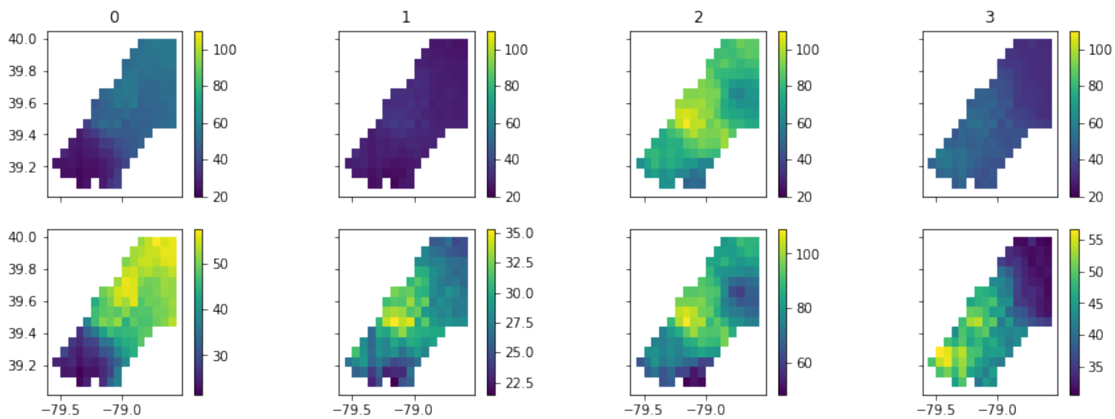
id	0	1	2	3	4	5	6	7	8	\
1915	33.325	33.125	35.525	32.975	34.525	35.850	34.550	34.650	34.050	
1916	16.300	18.550	19.300	18.150	18.425	21.950	19.600	18.775	17.850	
1917	23.425	23.775	24.275	23.350	24.150	25.300	25.175	25.325	24.300	
1918	37.000	34.775	35.325	32.175	31.475	39.750	35.500	34.475	32.800	
1919	40.350	39.425	40.200	40.625	40.400	43.425	41.350	33.800	32.825	

```
id          9  …     121     122     123     124     125     126     127  \
1915   33.675  …  32.300  34.700  35.475  41.950  24.550  30.300  30.950
1916   17.450  …  35.775  34.625  34.200  35.275  24.825  31.600  29.700
1917   24.275  …  30.050  29.075  29.150  36.200  36.100  39.375  27.625
1918   31.900  …  29.525  27.300  26.275  26.700  37.725  35.875  27.400
1919   32.425  …  33.575  31.700  29.550  30.825  41.825  39.125  32.275

id        128     129  labels
1915   38.200  41.325       3
1916   31.325  29.950       3
1917   33.300  35.225       1
1918   27.325  25.150       1
1919   33.225  31.425       1

[5 rows x 131 columns]
```

[33]:
```
cluster_means = grids.copy(deep = True)
for i in range(0, 4):
    _df = df[df.labels == i]
    _cluster_mean = np.zeros(130)
    for j in range(0, len(_cluster_mean)):
        _cluster_mean[j] = _df[j].mean()
        cluster_means[i] = _cluster_mean
cluster_means.head(2)
```

[33]:
```
     id       lat       lon  \
125   0  39.96875 -78.84375
126   1  39.96875 -78.78125

                                              geometry          0          1  \
125  POLYGON ((-78.87500 40.00000, -78.81250 40.000…  51.770000  25.286441
126  POLYGON ((-78.81250 40.00000, -78.75000 40.000…  54.318333  25.508475

           2          3
125  80.2500  31.291667
126  83.0375  31.248810
```

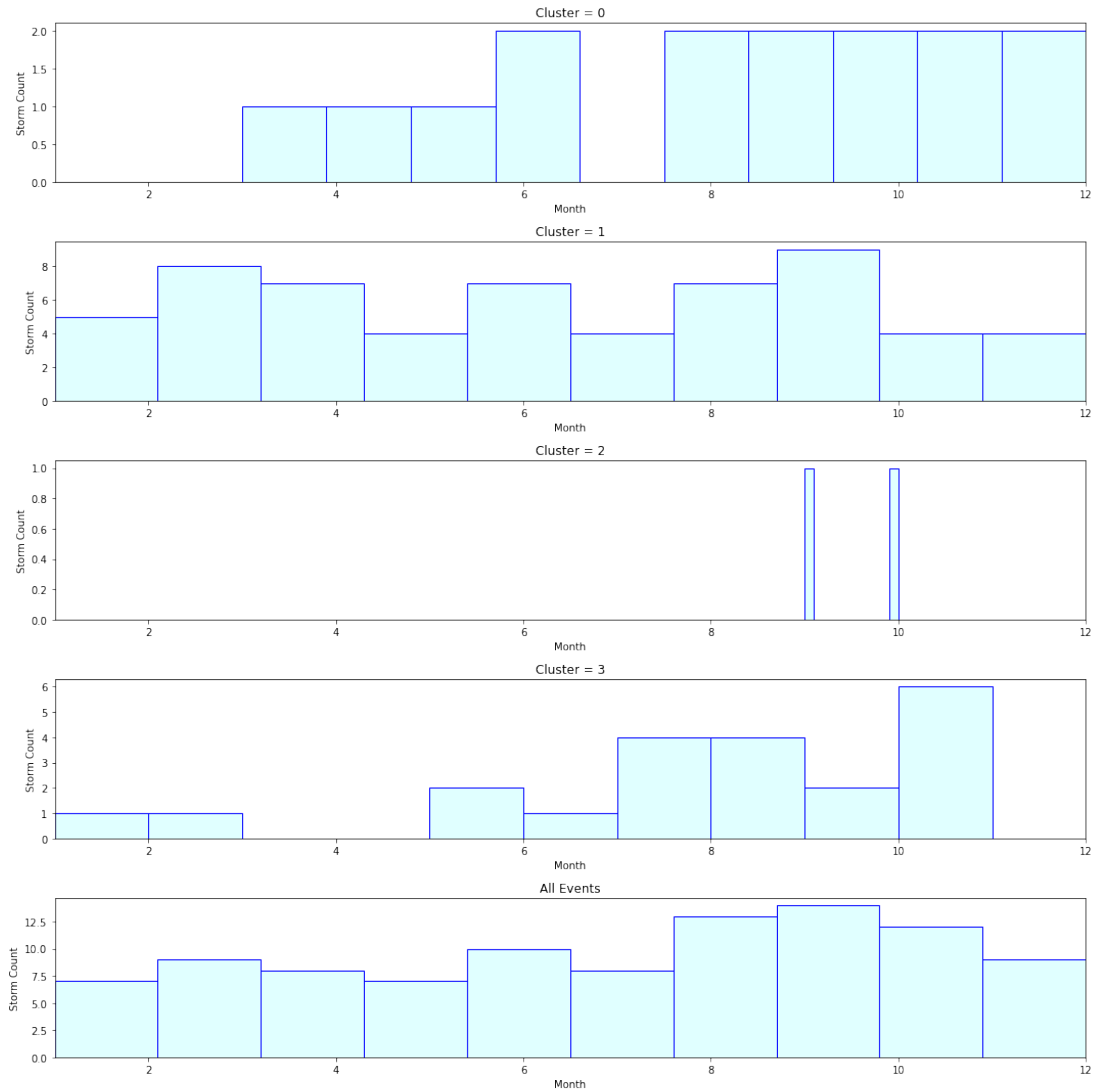
[40]:
```
fig, axs = plt.subplots(nrows = 2, ncols = 4, figsize = (15, 5), sharex = True,␣
 ↪sharey = True)
for i in range(0, 4):
    cluster_means.plot(ax = axs[0, i], column = i, vmin = 20, vmax = 110,␣
 ↪legend = True)
    cluster_means.plot(ax = axs[1, i], column = i, legend = True)
    axs[0, i].set_title(i)
```




```
[41]: events = livneh.eventdates(importpath, years)
      events['cluster'] = cluster_labels
      events.head(2)
```

```
[41]:          date  year  month  day  cluster
      0  1915-10-01  1915     10    1        3
      1  1916-08-04  1916      8    4        3
      2  1917-01-22  1917      1   22        1
```

```
[51]: fig, axs = plt.subplots(nrows = 5, ncols = 1, figsize=(15, 15))
      for i in range(0, 5):
          if i == 4:
              axs[i].hist(events.month, facecolor = 'lightcyan', edgecolor = 'blue')
              axs[i].set_title('All Events')
          else:
              axs[i].hist(events[events.cluster == i].month, facecolor = 'lightcyan',␣
       ↪edgecolor = 'blue')
              axs[i].set_title('Cluster = ' + str(i))
          axs[i].set_xlabel('Month')
          axs[i].set_ylabel('Storm Count')
          axs[i].set_xlim([1, 12])
          #axs[i].set_ylim([0,14])
      fig.tight_layout()
```


Cluster = 0
Storm Count
Month
Cluster = 1
Storm Count
Month
Cluster = 2
Storm Count
Month
Cluster = 3
Storm Count
Month
All Events
Storm Count
Month