Ex1_Precipitation

March 11, 2020

1 Exercises precipitation

1.1 Exercise 1 - Interpolate missing values

The figure shows the location of 11 precipitation gages in a research watershed. Measurements are missing at gage F for a rain storm. Use the records from the other gages (shown in the following table) to fill the gap in the rainfall amount in gage F.

The file *RainfallData_Exercise_001.csv* contains the rainfall data.

Gage	X	Y	Average Annual Precip. (mm)	Measured Storm Precip. (mm)
C	385014	4778553	1404	11.6
D	389634	4779045	1433	14.8
E	380729	4775518	1665	13.3
F	387259	4776670	1137	-
G	389380	4776484	1235	12.3
H	382945	4772356	1114	11.5
I	386399	4771795	1101	11.6
J	388397	4772419	1086	11.2
K	389287	4771097	1010	9.7

Methods: Section 1.1.2 Section ?? Section 1.1.4

Missing data completion methods and interpolation methods are both based in the following general equation:

$$\hat{p_o} = \sum_{i=1}^{n} w_i \cdot p_i$$

Where $\hat{p_o}$ is the rainfall value to be filled/interpolated, n is the number of gages used to interpolate, w_i and p_i are the weighting factor and the rainfall value in each of those gages.

[1]:
```
import numpy as np

import pandas as pd

from matplotlib import pyplot as plt
%matplotlib inline
# plt.style.use('dark_background')
plt.style.use('seaborn-whitegrid')
```

1.1.1 Import data

Import data using *pandas* and save it in an object (*data frame* using pandas terminology).

[2]:
```
# Import 'RainfallData_Exercise_001.csv'
data1 = pd.read_csv('..\data\RainfallData_Exercise_001.csv', index_col=0)
data1
```

[2]:
```
            X        Y  Average Annual Precip. (mm)  \
Gage
A      387706  4781780                         1373
B      383422  4778885                         1452
C      385014  4778553                         1404
D      389634  4779045                         1433
E      380729  4775518                         1665
F      387259  4776670                         1137
G      389380  4776484                         1235
H      382945  4772356                         1114
I      386399  4771795                         1101
J      388397  4772419                         1086
K      389287  4771097                         1010

      Measured Storm Precip. (mm)
Gage
A                            14.4
B                            12.2
C                            11.6
D                            14.8
E                            13.3
F                             NaN
G                            12.3
H                            11.5
I                            11.6
J                            11.2
K                             9.7
```

Data frames have a series of attributes associated. For instance, we can ask for the dimensions of the table (`shape`), the number of elements (`size`), the name of the rows or columns (`index` or columns) or extract the data as a *numpy array* (values).

To call an attribute, type: DataFrame.attribute

[3]:
```
# dimension
data1.shape
```

[3]:
```
(11, 4)
```

[4]:
```
# n. of elements
data1.size
```

[4]:
```
44
```

[5]:
```
# row names
data1.index
```

[5]:
```
Index(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'J', 'K'], dtype='object',
name='Gage')
```

[6]:
```
# column names
data1.columns
```

[6]:
```
Index(['X', 'Y', 'Average Annual Precip. (mm)', 'Measured Storm Precip. (mm)'],
dtype='object')
```

Apart from attributes, we can apply any kind of function to a *data frame*. For instance, describe shows a statisical summary of the variables in the *data fram*.

To apply a function, type: DataFrame.function()

[7]:
```
# summary
data1.describe()
```

[7]:
```
                   X             Y  Average Annual Precip. (mm)  \
count      11.000000  1.100000e+01                    11.000000
mean   386379.272727  4.775873e+06                  1273.636364
std      2983.076804  3.550817e+03                   204.304808
min    380729.000000  4.771097e+06                  1010.000000
25%    384218.000000  4.772388e+06                  1107.500000
50%    387259.000000  4.776484e+06                  1235.000000
75%    388842.000000  4.778719e+06                  1418.500000
max    389634.000000  4.781780e+06                  1665.000000

       Measured Storm Precip. (mm)
count                    10.000000
mean                     12.260000
std                       1.536374
min                       9.700000
```

```
25%                        11.525000
50%                        11.900000
75%                        13.050000
max                        14.800000
```

[8]:
```python
# mean
round(data1.mean(), 0)
```

[8]:
```
X                                 386379.0
Y                                4775873.0
Average Annual Precip. (mm)         1274.0
Measured Storm Precip. (mm)           12.0
dtype: float64
```

[9]:
```python
# first rows
data1.head()
```

[9]:
```
           X        Y  Average Annual Precip. (mm)  \
Gage
A     387706  4781780                         1373
B     383422  4778885                         1452
C     385014  4778553                         1404
D     389634  4779045                         1433
E     380729  4775518                         1665

      Measured Storm Precip. (mm)
Gage
A                            14.4
B                            12.2
C                            11.6
D                            14.8
E                            13.3
```

To extract a datum or a subset of data from a *data frame*, we must indicate row and column by either the name or the position. * Function `.loc` extracts subsets of a *data frame* calling the row and column names. * Function `.iloc` does the same, but using the row and column position. WARNING: Python positioning starts from 0, so the first row is row number 0 instead of 1.

[10]:
```python
# Extract using .loc
data1.loc['A', 'Measured Storm Precip. (mm)']
data1.loc['A', :]
data1.loc[['A', 'C'], 'Average Annual Precip. (mm)']
```

[10]:
```
Gage
A    1373
C    1404
Name: Average Annual Precip. (mm), dtype: int64
```

[11]:
```python
# Extract using .iloc
data1.iloc[0, 0]
data1.iloc[0, :]
data1.iloc[[0, 2], 1]
```

[11]:
```
Gage
A    4781780
C    4778553
Name: Y, dtype: int64
```

[12]:
```python
# Extract a column by its name
data1['Average Annual Precip. (mm)']
```

[12]:
```
Gage
A    1373
B    1452
C    1404
D    1433
E    1665
F    1137
G    1235
H    1114
I    1101
J    1086
K    1010
Name: Average Annual Precip. (mm), dtype: int64
```

[13]:
```python
# Simplify column names
# d: 'Distance from gage F (km)'
# P: Average Annual Precip. (mm)
# p: Measured Storm Precip. (mm)
data1.columns = ['X', 'Y', 'Pan', 'p']

data1.head(2)
```

[13]:

	X	Y	Pan	p
Gage				
A	387706	4781780	1373	14.4
B	383422	4778885	1452	12.2

1.1.2 The station-average method

In this method, we assume that rainfall in the target point is the average rainfall in the surrounding gages.

Following the general equation, we give the same weight w_i to every gage.

$$w_i = \frac{1}{n}$$

$$\hat{p_o} = \frac{1}{n} \sum_{i=1}^{n} p_i$$

where n is the number of gages.

[14]:
```
po_mm = data1['p'].mean()
```

[15]:
```
print('Rainfall in F is:')
print('pf =', round(po_mm, 1), 'mm')
```

```
Rainfall in F is:
pf = 12.3 mm
```

When averaging rainfall throughout all the available gages, we obtain a smoothed value of rainfall in F. That is, we may be using data from stations so far away from the study point that there is no connection between rainfall values in those two stations.

To avoid this problem, the station-average method is usually applied using only the closest gage to the study point in each quadrant.

[16]:
```
closest = ['C', 'D', 'G', 'I']
po_mmc = data1.loc[closest, 'p'].mean()

print('Rainfall in F is:')
print('pf =', round(po_mmc, 1), 'mm')
```

```
Rainfall in F is:
pf = 12.6 mm
```

1.1.3 The normal-ratio method

The normal-ratio is the quotient between the annual precipitation in two gages.

$$NR_i = \frac{P_o}{P_i}$$

We can use this value as a measure of the connection of the rainfall amounts between those gages. The normal-ratio method applies the normal-ratio as a correction to the weights derived by the station-average method.

$$w_i = \frac{1}{n}\frac{P_o}{P_i} = \frac{1}{n} NR_i$$

$$\hat{p_o} = \frac{1}{n}\sum_{i=1}^{n}\frac{P_o}{P_i}p_i$$

Where P_o and P_i are annual precipitation in the point of interest and the available gages, respectively, and NR is the normal-ratio.

[17]:
```
# Extract stations with data during the storm
data1_ = data1.drop('F').copy()
data1_
```

[17]:

Gage	X	Y	Pan	p
A	387706	4781780	1373	14.4
B	383422	4778885	1452	12.2
C	385014	4778553	1404	11.6
D	389634	4779045	1433	14.8
E	380729	4775518	1665	13.3
G	389380	4776484	1235	12.3
H	382945	4772356	1114	11.5
I	386399	4771795	1101	11.6
J	388397	4772419	1086	11.2
K	389287	4771097	1010	9.7

[18]:
```
# Calculate the normal ration between station F and the rest of the stations
data1_['RN'] = data1.loc['F', 'Pan'] / data1_['Pan']

data1_
```

[18]:

Gage	X	Y	Pan	p	RN
A	387706	4781780	1373	14.4	0.828114
B	383422	4778885	1452	12.2	0.783058
C	385014	4778553	1404	11.6	0.809829
D	389634	4779045	1433	14.8	0.793440
E	380729	4775518	1665	13.3	0.682883
G	389380	4776484	1235	12.3	0.920648
H	382945	4772356	1114	11.5	1.020646
I	386399	4771795	1101	11.6	1.032698
J	388397	4772419	1086	11.2	1.046961
K	389287	4771097	1010	9.7	1.125743

Seguidamente multiplicamos, para cada estación, la **razón normal** por la precipitación medida en la tormenta.

[19]:
```
# Multiply the storm rainfall by the normal ratio
data1_['NR*p'] = data1_['RN'] * data1_['p']
data1_['NR*p']
```

[19]:
```
Gage
A    11.924836
B     9.553306
C     9.394017
D    11.742917
E     9.082342
G    11.323968
H    11.737433
I    11.979292
J    11.725967
K    10.919703
Name: NR*p, dtype: float64
```

[20]:
```python
# the mean of that product is the rainfall interpolated by the normal-ratio␣
↪method
po_rn = data1_['NR*p'].mean()

print('Rainfall in F is:')
print('pf =', round(po_rn, 1), 'mm')
```

```
Rainfall in F is:
pf = 10.9 mm
```

[21]:
```python
# All at once
po_rn = np.mean(data1_.loc[:, 'RN'] * data1_.loc[:, 'p'])

print('Rainfall in F is:')
print('pf =', round(po_rn, 1), 'mm')
```

```
Rainfall in F is:
pf = 10.9 mm
```

We can also apply the normal-ratio method to the closest gage in each quadrant

[22]:
```python
po_rnc = np.mean(data1_.loc[closest, 'RN'] * data1_.loc[closest, 'p'])

print('Rainfall in F is:')
print('pf =', round(po_rnc, 1), 'mm')
```

```
Rainfall in F is:
pf = 11.1 mm
```

1.1.4 The inverse distance method

The inverse distance method is based on the assumption that gages closer to the point of interest are more representative of its rainfall. In this methdos, gage weights are estimated as the inverse of the distance to a power. Since the sum of weights must be 1, we normalize the weighted inverse distances dividing them by their sum.

$$w_i = \frac{d_i^{-b}}{\sum_{i=1}^{n} d_i^{-b}}$$

$$\hat{p_o} = \sum_{i=1}^{n} \frac{d_i^{-b}}{\sum_{i=1}^{n} d_i^{-b}} \cdot p_i = \frac{1}{\sum_{i=1}^{n} d_i^{-b}} \sum_{i=1}^{n} d_i^{-b} \cdot p_i$$

Where d_i is the distance between gage i and target point, and b is the power to be chosen by the modeller.

A squared power ($b = -2$) is usually applied, what is named the squared inverse distance method. The larger the exponent, the higher weight is given to the closer gages.

[23]:
```
# Extract stations with data during the storm
data1_ = data1.drop('F').copy()
data1_
```

[23]:

	X	Y	Pan	p
Gage				
A	387706	4781780	1373	14.4
B	383422	4778885	1452	12.2
C	385014	4778553	1404	11.6
D	389634	4779045	1433	14.8
E	380729	4775518	1665	13.3
G	389380	4776484	1235	12.3
H	382945	4772356	1114	11.5
I	386399	4771795	1101	11.6
J	388397	4772419	1086	11.2
K	389287	4771097	1010	9.7

[24]:
```
# calculate distance to F
distX = data1.loc['F', 'X'] - data1_.loc[:, 'X'] # distance in the X axis
distY = data1.loc['F', 'Y'] - data1_.loc[:, 'Y'] # distance in the Y axis
data1_['d'] = np.sqrt(distX**2 + distY**2)       # total distance

data1_.head()
```

[24]:

	X	Y	Pan	p	d
Gage					
A	387706	4781780	1373	14.4	5129.513525
B	383422	4778885	1452	12.2	4430.439482
C	385014	4778553	1404	11.6	2930.138905
D	389634	4779045	1433	14.8	3358.757211
E	380729	4775518	1665	13.3	6630.837353

$b = -1$ **step by step**

[25]:
```python
# set the exponent
b = -1
```

[26]:
```python
# compute the weighted inverse of the distance
data1_['di'] = data1_['d']**b

data1_.head()
```

[26]:

Gage	X	Y	Pan	p	d	di
A	387706	4781780	1373	14.4	5129.513525	0.000195
B	383422	4778885	1452	12.2	4430.439482	0.000226
C	385014	4778553	1404	11.6	2930.138905	0.000341
D	389634	4779045	1433	14.8	3358.757211	0.000298
E	380729	4775518	1665	13.3	6630.837353	0.000151

[27]:
```python
# sum of all the weighted inverse distances
Sd = data1_['di'].sum()
Sd
```

[27]: 0.0024419303169758745

[28]:
```python
# compute the weights
data1_['w'] = data1_['di'] / Sd

data1_.head()
```

[28]:

Gage	X	Y	Pan	p	d	di	w
A	387706	4781780	1373	14.4	5129.513525	0.000195	0.079834
B	383422	4778885	1452	12.2	4430.439482	0.000226	0.092431
C	385014	4778553	1404	11.6	2930.138905	0.000341	0.139759
D	389634	4779045	1433	14.8	3358.757211	0.000298	0.121924
E	380729	4775518	1665	13.3	6630.837353	0.000151	0.061759

[29]:
```python
# compute rainfall in F
po_di1 = np.sum(data1_['w'] * data1_['p'])

print('Rainfall in F is:')
print('pf =', round(po_di1, 1), 'mm')
```

```
Rainfall in F is:
pf = 12.3 mm
```

$b = -2$ **shortened**

[30]:
```python
b = -2
```

[31]:
```
# Calcula el inverso de la distancia al cuadrado
data1_['di2'] = data1_['d']**b
data1_
```

[31]:
```
           X        Y   Pan     p            d        di         w  \
Gage
A     387706  4781780  1373  14.4  5129.513525  0.000195  0.079834
B     383422  4778885  1452  12.2  4430.439482  0.000226  0.092431
C     385014  4778553  1404  11.6  2930.138905  0.000341  0.139759
D     389634  4779045  1433  14.8  3358.757211  0.000298  0.121924
E     380729  4775518  1665  13.3  6630.837353  0.000151  0.061759
G     389380  4776484  1235  12.3  2129.139967  0.000470  0.192337
H     382945  4772356  1114  11.5  6100.917308  0.000164  0.067123
I     386399  4771795  1101  11.6  4950.275245  0.000202  0.082725
J     388397  4772419  1086  11.2  4400.686878  0.000227  0.093056
K     389287  4771097  1010   9.7  5930.523839  0.000169  0.069052

               di2
Gage
A     3.800560e-08
B     5.094556e-08
C     1.164725e-07
D     8.864266e-08
E     2.274381e-08
G     2.205929e-07
H     2.686642e-08
I     4.080762e-08
J     5.163677e-08
K     2.843242e-08
```

[32]:
```
# compute rainfall in F
po_di2 = np.sum(data1_['di2'] / np.sum(data1_['di2']) * data1_['p'])

print('Rainfall in F:')
print('pf =', round(po_di2, 1), 'mm')
```

```
Rainfall in F:
pf = 12.4 mm
```

Again, we can apply the inverse distance method only to the closest station per quadrant.

[33]:
```
# compute rainfall in F
po_di2c = np.sum(data1_.loc[closest, 'di2'] * data1_.loc[closest, 'p']) / \
          np.sum(data1_.loc[closest, 'di2'])

print('La precipitación en F es:')
print('pf =', round(po_di2c, 1), 'mm')
```

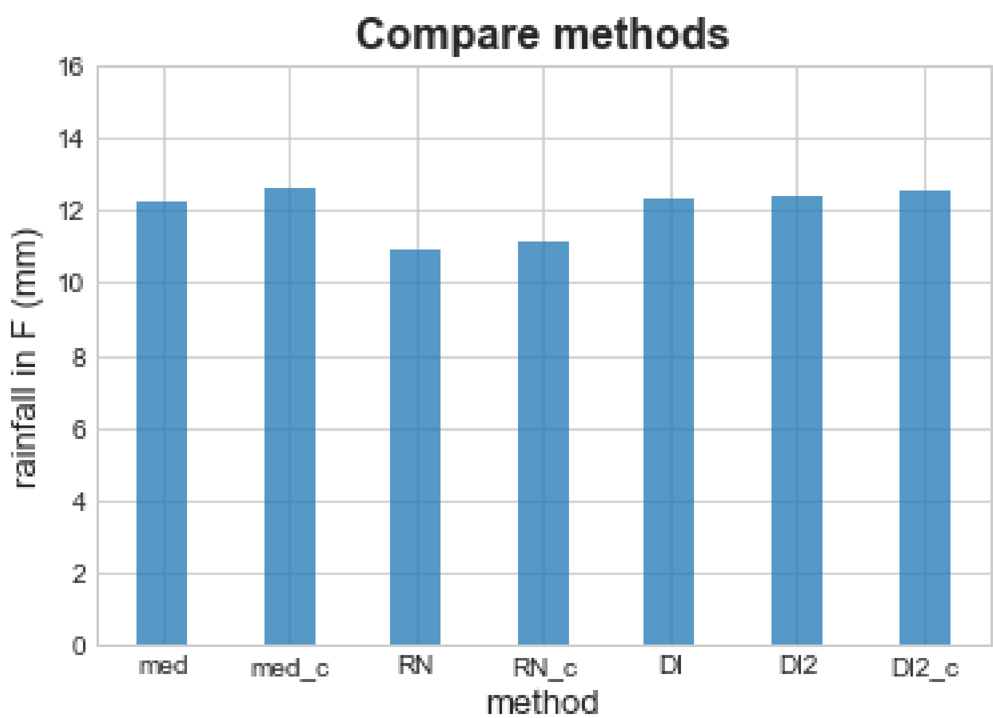
```
La precipitación en F es:
pf = 12.5 mm
```

Comparativa de métodos

[34]:
```python
# create an array with all the results
results = np.array([po_mm, po_mmc, po_rn, po_rnc, po_di1, po_di2, po_di2c])
```

[35]:
```python
plt.bar(range(len(results)), results, width=0.4, alpha=.75)
plt.title('Compare methods', fontsize=16, weight='bold')
plt.xlabel('method', fontsize=13)
plt.xticks(range(len(results)), ['med', 'med_c', 'RN', 'RN_c', 'DI',
                                  'DI2', 'DI2_c'])
plt.ylim((0, 16))
plt.ylabel('rainfall in F (mm)', fontsize=13);

plt.savefig('../output/Ex1_compare methods.png', dpi=300)
```



[36]:
```python
# convert the array of results in a data frame
results = pd.DataFrame(np.transpose(results),
                       index=['po_mm', 'po_mmc', 'po_rn', 'po_rnc',␣
 ↪'po_di1', 'po_di2', 'po_di2c'])
results = results.transpose()
results
```

[36]:

	po_mm	po_mmc	po_rn	po_rnc	po_di1	po_di2	po_di2c
0	12.26	12.575	10.938378	11.110048	12.333644	12.382834	12.539028

[37]:
```
# export results as a csv
results.to_csv('../output/Ex1_compare methods.csv', index=False,␣
↪float_format='%.1f')
```

1.1.5 Interpolation

The methods aboved explained may also be used in spatial interpolation, i.e., to generate maps of precipitation.

We are going to create a precipitation map using the inverse distance weighted method. To do that, we are going to create a *Python* functions that performs IDW.

[38]:
```
def IDW(x, y, stnX, stnY, stnP, b=-2):
    """Interpolate by the inverse distance weighted method

    Parameters:
    ----------
    x:       float. Coordenate X of the target point
    y:       float. Coordenate Y of the target point
    stnX:    Series. Coordenates X of the gages
    stnY:    Series. Coordenates Y of the gages
    stnP:    Series. Observed precipitation in the gages
    b:       int. Exponent in the inverse distance

    Returns:
    --------
    p:       float. Precipitation interpolated for point (x, y)
    """

    # distance to the target point
    distX = x - stnX                      # distancia en el eje X
    distY = y - stnY                      # distancia en el eje X
    dist = np.sqrt(distX**2 + distY**2) # distancia total
    # inverse of the distance
    idw = dist**b
    # interpolate
    p = np.sum(idw / np.sum(idw) * stnP)

    return round(p, 1)
```

[39]:
```
# check the function in point F
IDW(data1.loc['F', 'X'], data1.loc['F', 'Y'], data1_['X'], data1_['Y'],
    data1_['p'], b=-2)
```

[39]: 12.4

Now we can apply the function to a grid of cells representing an area.

[40]:
```python
# Coordenates X and Y of the grid
xo, xf = 382200, 390200
X = np.arange(xo, xf, 100)
yo, yf = 4771400, 4779400
Y = np.arange(yo, yf, 100)
```

[41]:
```python
# create empty map (zeros) with the dimensions of 'X' and 'Y'
pcp = np.zeros((len(X), len(Y)))
```

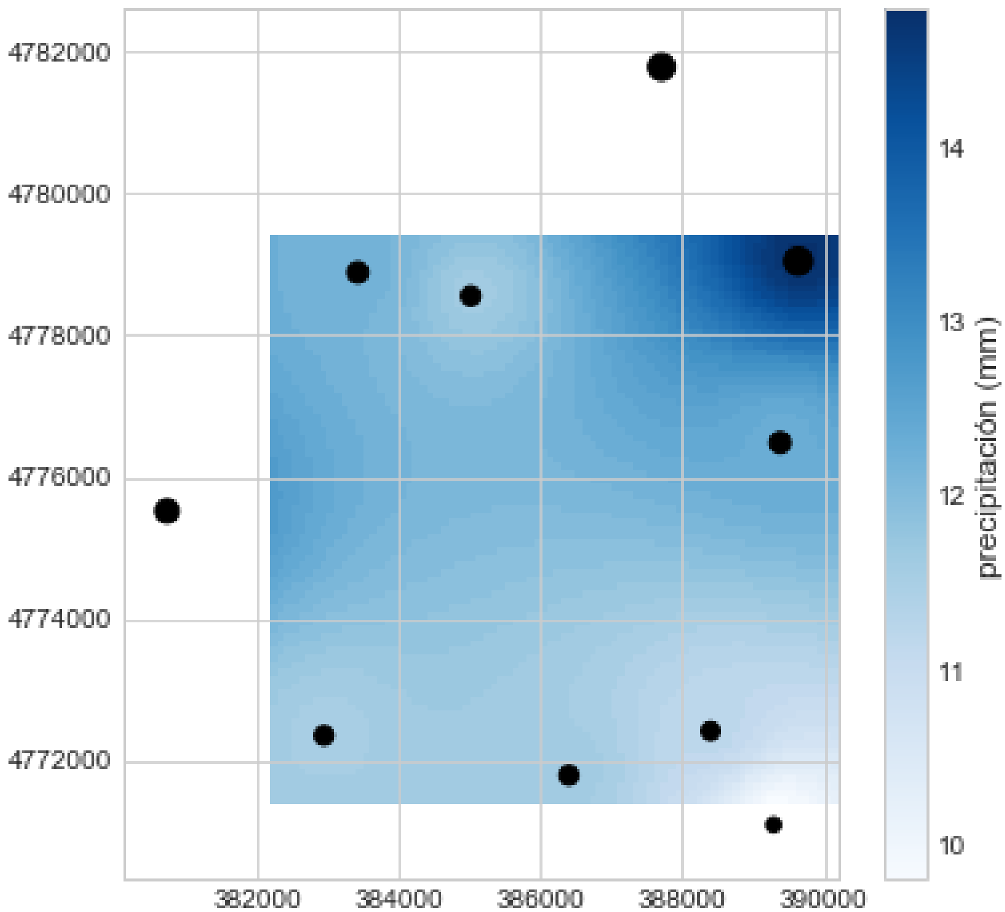
[42]:
```python
# interpolate rainfall in each cell of the gridr la precipitación en cada una␣
↪de las celdas del mapa
for i, y in enumerate(Y[::-1]): # important to invert the position of 'Y'
    for j, x in enumerate(X):
        pcp[i, j] = IDW(x, y, data1_.X, data1_.Y, data1_.p, b=-2)
```

[43]:
```python
# gráfico con las estaciones y el mapa de precipitación interpolada
# ----------------------------------------------------------------
# configuración
plt.figure(figsize=(6, 6))
plt.axis('equal')

# mapa interpolado
pmap = plt.imshow(pcp, extent=[xo, xf, yo, yf], cmap='Blues')
cb = plt.colorbar(pmap)
cb.set_label('precipitación (mm)', rotation=90, fontsize=12)

# puntos con las estaciones
plt.scatter(data1_.X, data1_.Y, c='k', s=data1_.p**3/30);

plt.savefig('../output/Ex1_precipitation map.png', dpi=300)
```


4782000
4780000
4778000
4776000
4774000
4772000
382000
384000
386000
388000
390000
14
13
12
11
10
precipitación (mm)

[ ]: