2023-04-17_Example-Anscombe

April 18, 2023

[12]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')

prop_cycle = plt.rcParams['axes.prop_cycle']
colors = prop_cycle.by_key()['color']

%matplotlib inline

import scipy.stats
```

[13]:
```
# raw input data
x1=np.array([10.,8.,13.,9.,11.,14.,6.,4.,12.,7.,5.])
y1=np.array([8.04,6.95,7.58,8.81,8.33,9.96,7.24,4.26,10.84,4.82,5.68])
y2=np.array([9.14,8.14,8.74,8.77,9.26,8.10,6.13,3.10,9.13,7.26,4.74])
y3=np.array([7.46,6.77,12.74,7.11,7.81,8.84,6.08,5.39,8.15,6.42,5.73])
x2=np.array([8.,8.,8.,8.,8.,8.,8.,19.,8.,8.,8.])
y4=np.array([6.58,5.76,7.71,8.84,8.47,7.04,5.25,12.50,5.56,7.91,6.89])
```

[14]:
```
# print output for latex script
for x in [x1,y1,y2,y3,x2,y4]:
    for i,s in enumerate(x):
        if i>0:
            print(" & ",end="")
        print("{:.2f}".format(s),end="")
    print(" \\\\")
```

```
10.00 & 8.00 & 13.00 & 9.00 & 11.00 & 14.00 & 6.00 & 4.00 & 12.00 & 7.00 & 5.00
\\
8.04 & 6.95 & 7.58 & 8.81 & 8.33 & 9.96 & 7.24 & 4.26 & 10.84 & 4.82 & 5.68 \\
```

```
9.14 & 8.14 & 8.74 & 8.77 & 9.26 & 8.10 & 6.13 & 3.10 & 9.13 & 7.26 & 4.74 \\
7.46 & 6.77 & 12.74 & 7.11 & 7.81 & 8.84 & 6.08 & 5.39 & 8.15 & 6.42 & 5.73 \\
8.00 & 8.00 & 8.00 & 8.00 & 8.00 & 8.00 & 8.00 & 19.00 & 8.00 & 8.00 & 8.00 \\
6.58 & 5.76 & 7.71 & 8.84 & 8.47 & 7.04 & 5.25 & 12.50 & 5.56 & 7.91 & 6.89 \\
```

[15]:
```
# arrange into (x,y) pairs
data=np.array([[x1,y1],[x1,y2],[x1,y3],[x2,y4]])
```

[16]:
```
data.shape
```

[16]:
```
(4, 2, 11)
```

[17]:
```
# mean, variance of x sequences
for x in [x1,x2]:
    print("{:.2f}\t{:.2f}".format(np.mean(x),np.var(x)))
```

```
9.00    10.00
9.00    10.00
```

[18]:
```
# mean, variance of y sequences
for x in [y1,y2,y3,y4]:
    print("{:.2f}\t{:.2f}".format(np.mean(x),np.var(x)))
```

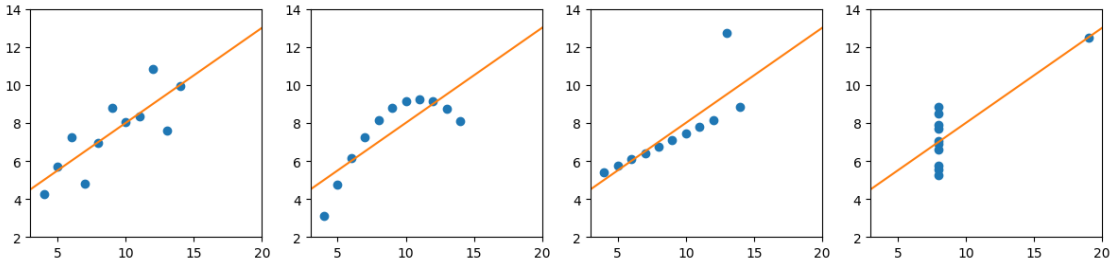
```
7.50    3.75
7.50    3.75
7.50    3.75
7.50    3.75
```

[19]:
```
# linear regression
regression=[scipy.stats.linregress(*dat) for dat in data]
print("slope\tintercept\tcorrelation\tstandard error slope")
for reg in regression:
    print("{: .2f} \t{: .2f}\t\t{: .2f}\t\t{: .2f}".format(reg.slope,reg.
 ↪intercept,reg.rvalue,reg.stderr))
```

```
slope   intercept       correlation     standard error slope
 0.50    3.00            0.82            0.12
 0.50    3.00            0.82            0.12
 0.50    3.00            0.82            0.12
 0.50    3.00            0.82            0.12
```

[20]:
```
# plots
xref=np.linspace(3,20,num=50)
fig=plt.figure(figsize=(12,3))
for i,(dat,reg) in enumerate(zip(data,regression)):
    fig.add_subplot(1,4,i+1)
    plt.scatter(dat[0],dat[1],c=colors[0])
    plt.plot(xref,reg.slope*xref+reg.intercept,c=colors[1])
```

```
    plt.xlim([3,20])
    plt.ylim([2,14])
plt.tight_layout()
plt.show()
```



[ ]:
```
fig.savefig("anscombe.pdf")
```

2023-04-17_WorldDemographics-Test-01

April 18, 2023

[4]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

%matplotlib inline
```

1 Naive first exploration of UN world population data

available at: https://population.un.org/wpp/Download/Standard/CSV/

available columns in dataset: * Total Population, as of 1 January (thousands) * Total Population, as of 1 July (thousands) * Male Population, as of 1 July (thousands) * Female Population, as of 1 July (thousands) * Population Density, as of 1 July (persons per square km) * Population Sex Ratio, as of 1 July (males per 100 females) * Median Age, as of 1 July (years) * Natural Change, Births minus Deaths (thousands) * Rate of Natural Change (per 1,000 population) * Population Change (thousands) * Population Growth Rate (percentage) * Population Annual Doubling Time (years) * Births (thousands) * Births by women aged 15 to 19 (thousands) * Crude Birth Rate (births per 1,000 population) * Total Fertility Rate (live births per woman) * Net Reproduction Rate (surviving daughters per woman) * Mean Age Childbearing (years) * Sex Ratio at Birth (males per 100 female births) * Total Deaths (thousands) * Male Deaths (thousands) * Female Deaths (thousands) * Crude Death Rate (deaths per 1,000 population) * Life Expectancy at Birth, both sexes (years) * Male Life Expectancy at Birth (years) * Female Life Expectancy at Birth (years) * Life Expectancy at Age 15, both sexes (years) * Male Life Expectancy at Age 15 (years) * Female Life Expectancy at Age 15 (years) * Life Expectancy at Age 65, both sexes (years) * Male Life Expectancy at Age 65 (years) * Female Life Expectancy at Age 65 (years) * Life Expectancy at Age 80, both sexes (years) * Male Life Expectancy at Age 80 (years) * Female Life Expectancy at Age 80 (years) * Infant Deaths, under age 1 (thousands) * Infant Mortality Rate (infant deaths per 1,000 live births) * Live births Surviving to Age 1 (thousands) * Deaths under age 5 (thousands) * Under-five Mortality Rate (deaths under age 5 per 1,000 live births) * Mortality before Age 40, both sexes (deaths under age 40 per 1,000 live births) * Male mortality before Age 40 (deaths under age 40 per 1,000 male live births) * Female mortality before Age 40 (deaths under age 40 per 1,000 female live births) * Mortality before Age 60, both sexes (deaths under age 60 per 1,000 live births) * Male mortality before Age 60 (deaths under age 60 per 1,000 male live births) * Female mortality before Age 60 (deaths under age 60 per 1,000 female live births) * Mortality between Age 15 and 50, both sexes (deaths under age 50 per 1,000 alive at age 15) * Male mortality between Age 15

and 50 (deaths under age 50 per 1,000 males alive at age 15) * Female mortality between Age 15 and 50 (deaths under age 50 per 1,000 females alive at age 15) * Mortality between Age 15 and 60, both sexes (deaths under age 60 per 1,000 alive at age 15) * Male mortality between Age 15 and 60 (deaths under age 60 per 1,000 males alive at age 15) * Female mortality between Age 15 and 60 (deaths under age 60 per 1,000 females alive at age 15) * Net Number of Migrants (thousands) * Net Migration Rate (per 1,000 population)

1.1 Import data

[5]:
```
# set up column data types
headerLine="SortOrder,LocID,Notes,ISO3_code,ISO2_code,SDMX_code,LocTypeID,LocTypeName,"+\
    "ParentID,Location,VarID,Variant,"+\
  ␣
 ↪"Time,TPopulation1Jan,TPopulation1July,TPopulationMale1July,TPopulationFemale1July,"+\
  ␣
 ↪"PopDensity,PopSexRatio,MedianAgePop,NatChange,NatChangeRT,PopChange,PopGrowthRate,"+\
  ␣
 ↪"DoublingTime,Births,Births1519,CBR,TFR,NRR,MAC,SRB,Deaths,DeathsMale,DeathsFemale,"+\
  ␣
 ↪"CDR,LEx,LExMale,LExFemale,LE15,LE15Male,LE15Female,LE65,LE65Male,LE65Female,"+\
    "LE80,LE80Male,LE80Female,InfantDeaths,IMR,LBsurvivingAge1,Under5Deaths,"+\
  ␣
 ↪"Q5,Q0040,Q0040Male,Q0040Female,Q0060,Q0060Male,Q0060Female,Q1550,Q1550Male,"+\
    "Q1550Female,Q1560,Q1560Male,Q1560Female,NetMigrations,CNMR"

dtypeDict={}
for x in headerLine.split(","):
    dtypeDict[x]=np.float64
for x in "SortOrder,LocID,LocTypeID,ParentID,VarID,Time".split(","):
    dtypeDict[x]=np.int32
for x in "Notes,ISO3_code,SDMX_code,ISO2_code,LocTypeName,Location,Variant".
 ↪split(","):
    dtypeDict[x]=str
```

[6]:
```
dataFull=pd.read_csv("data/WPP2022_Demographic_Indicators_Medium.
 ↪csv",sep=",",dtype=dtypeDict)
```

[7]:
```
dataFull.keys()
```

[7]:
```
Index(['SortOrder', 'LocID', 'Notes', 'ISO3_code', 'ISO2_code', 'SDMX_code',
       'LocTypeID', 'LocTypeName', 'ParentID', 'Location', 'VarID', 'Variant',
       'Time', 'TPopulation1Jan', 'TPopulation1July', 'TPopulationMale1July',
       'TPopulationFemale1July', 'PopDensity', 'PopSexRatio', 'MedianAgePop',
       'NatChange', 'NatChangeRT', 'PopChange', 'PopGrowthRate',
       'DoublingTime', 'Births', 'Births1519', 'CBR', 'TFR', 'NRR', 'MAC',
       'SRB', 'Deaths', 'DeathsMale', 'DeathsFemale', 'CDR', 'LEx', 'LExMale',
```

```
       'LExFemale', 'LE15', 'LE15Male', 'LE15Female', 'LE65', 'LE65Male',
       'LE65Female', 'LE80', 'LE80Male', 'LE80Female', 'InfantDeaths', 'IMR',
       'LBsurvivingAge1', 'Under5Deaths', 'Q5', 'Q0040', 'Q0040Male',
       'Q0040Female', 'Q0060', 'Q0060Male', 'Q0060Female', 'Q1550',
       'Q1550Male', 'Q1550Female', 'Q1560', 'Q1560Male', 'Q1560Female',
       'NetMigrations', 'CNMR'],
      dtype='object')
```

[8]:
```
dataFull.shape
```

[8]:
```
(43472, 67)
```

1.1.1 Filter for countries and non-projected data

[9]:
```
countryIndicator=(dataFull["LocTypeName"]=="Country/Area")
timeIndicator=(dataFull["Time"]<=2022)

data=dataFull[countryIndicator & timeIndicator]

data.shape
```

[9]:
```
(17301, 67)
```

[10]:
```
countryList=data["Location"].unique()
print(countryList)
```

```
['Burundi' 'Comoros' 'Djibouti' 'Eritrea' 'Ethiopia' 'Kenya' 'Madagascar'
 'Malawi' 'Mauritius' 'Mayotte' 'Mozambique' 'Réunion' 'Rwanda'
 'Seychelles' 'Somalia' 'South Sudan' 'Uganda'
 'United Republic of Tanzania' 'Zambia' 'Zimbabwe' 'Angola' 'Cameroon'
 'Central African Republic' 'Chad' 'Congo'
 'Democratic Republic of the Congo' 'Equatorial Guinea' 'Gabon'
 'Sao Tome and Principe' 'Algeria' 'Egypt' 'Libya' 'Morocco' 'Sudan'
 'Tunisia' 'Western Sahara' 'Botswana' 'Eswatini' 'Lesotho' 'Namibia'
 'South Africa' 'Benin' 'Burkina Faso' 'Cabo Verde' "Côte d'Ivoire"
 'Gambia' 'Ghana' 'Guinea' 'Guinea-Bissau' 'Liberia' 'Mali' 'Mauritania'
 'Niger' 'Nigeria' 'Saint Helena' 'Senegal' 'Sierra Leone' 'Togo'
 'Kazakhstan' 'Kyrgyzstan' 'Tajikistan' 'Turkmenistan' 'Uzbekistan'
 'China' 'China, Hong Kong SAR' 'China, Macao SAR'
 'China, Taiwan Province of China' "Dem. People's Republic of Korea"
 'Japan' 'Mongolia' 'Republic of Korea' 'Afghanistan' 'Bangladesh'
 'Bhutan' 'India' 'Iran (Islamic Republic of)' 'Maldives' 'Nepal'
 'Pakistan' 'Sri Lanka' 'Brunei Darussalam' 'Cambodia' 'Indonesia'
 "Lao People's Democratic Republic" 'Malaysia' 'Myanmar' 'Philippines'
 'Singapore' 'Thailand' 'Timor-Leste' 'Viet Nam' 'Armenia' 'Azerbaijan'
 'Bahrain' 'Cyprus' 'Georgia' 'Iraq' 'Israel' 'Jordan' 'Kuwait' 'Lebanon'
```

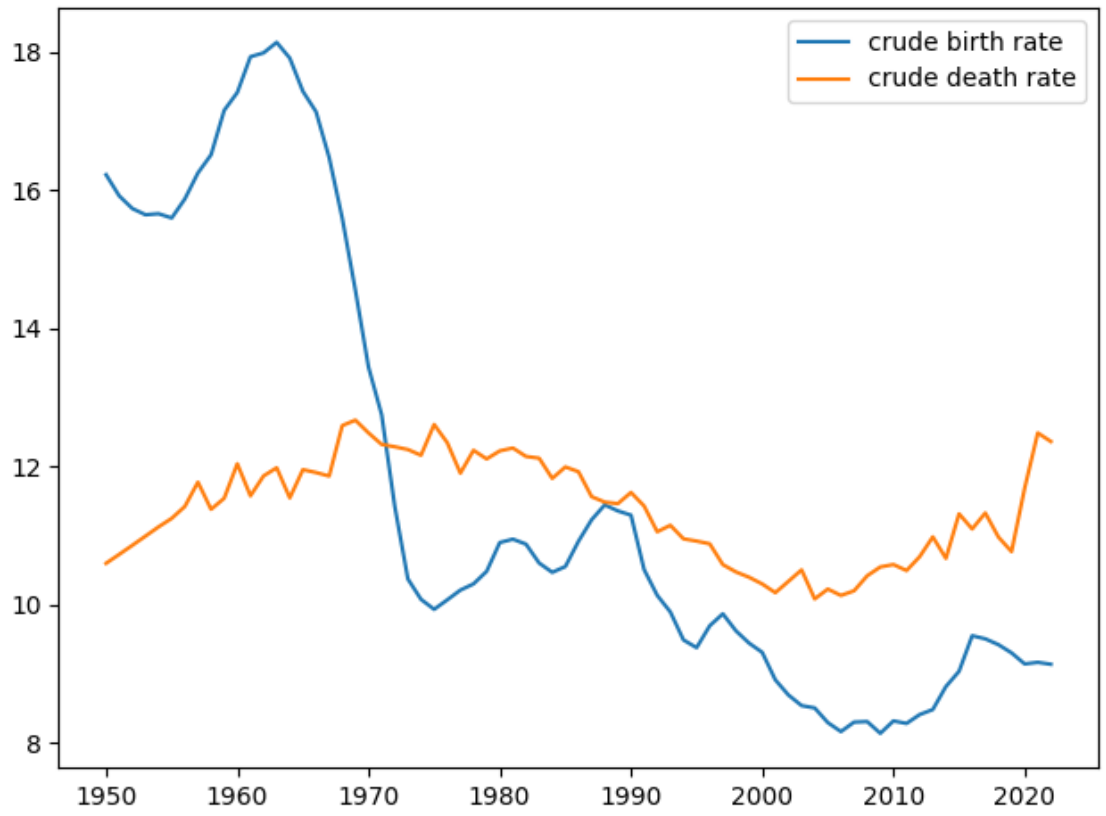
```
'Oman' 'Qatar' 'Saudi Arabia' 'State of Palestine' 'Syrian Arab Republic'
'Türkiye' 'United Arab Emirates' 'Yemen' 'Belarus' 'Bulgaria' 'Czechia'
'Hungary' 'Poland' 'Republic of Moldova' 'Romania' 'Russian Federation'
'Slovakia' 'Ukraine' 'Denmark' 'Estonia' 'Faroe Islands' 'Finland'
'Guernsey' 'Iceland' 'Ireland' 'Isle of Man' 'Jersey' 'Latvia'
'Lithuania' 'Norway' 'Sweden' 'United Kingdom' 'Albania' 'Andorra'
'Bosnia and Herzegovina' 'Croatia' 'Gibraltar' 'Greece' 'Holy See'
'Italy' 'Kosovo (under UNSC res. 1244)' 'Malta' 'Montenegro'
'North Macedonia' 'Portugal' 'San Marino' 'Serbia' 'Slovenia' 'Spain'
'Austria' 'Belgium' 'France' 'Germany' 'Liechtenstein' 'Luxembourg'
'Monaco' 'Netherlands' 'Switzerland' 'Anguilla' 'Antigua and Barbuda'
'Aruba' 'Bahamas' 'Barbados' 'Bonaire, Sint Eustatius and Saba'
'British Virgin Islands' 'Cayman Islands' 'Cuba' 'Curaçao' 'Dominica'
'Dominican Republic' 'Grenada' 'Guadeloupe' 'Haiti' 'Jamaica'
'Martinique' 'Montserrat' 'Puerto Rico' 'Saint Barthélemy'
'Saint Kitts and Nevis' 'Saint Lucia' 'Saint Martin (French part)'
'Saint Vincent and the Grenadines' 'Sint Maarten (Dutch part)'
'Trinidad and Tobago' 'Turks and Caicos Islands'
'United States Virgin Islands' 'Belize' 'Costa Rica' 'El Salvador'
'Guatemala' 'Honduras' 'Mexico' 'Nicaragua' 'Panama' 'Argentina'
'Bolivia (Plurinational State of)' 'Brazil' 'Chile' 'Colombia' 'Ecuador'
'Falkland Islands (Malvinas)' 'French Guiana' 'Guyana' 'Paraguay' 'Peru'
'Suriname' 'Uruguay' 'Venezuela (Bolivarian Republic of)' 'Bermuda'
'Canada' 'Greenland' 'Saint Pierre and Miquelon'
'United States of America' 'Australia' 'New Zealand' 'Fiji'
'New Caledonia' 'Papua New Guinea' 'Solomon Islands' 'Vanuatu' 'Guam'
'Kiribati' 'Marshall Islands' 'Micronesia (Fed. States of)' 'Nauru'
'Northern Mariana Islands' 'Palau' 'American Samoa' 'Cook Islands'
'French Polynesia' 'Niue' 'Samoa' 'Tokelau' 'Tonga' 'Tuvalu'
'Wallis and Futuna Islands']
```

1.2 Exploring dataset

1.2.1 First naive test: birth and death rates in Germany

[11]:
```
# CBR: "crude birth rate": births per 1000 persons in that year
# CDR: "crude death rate"
```

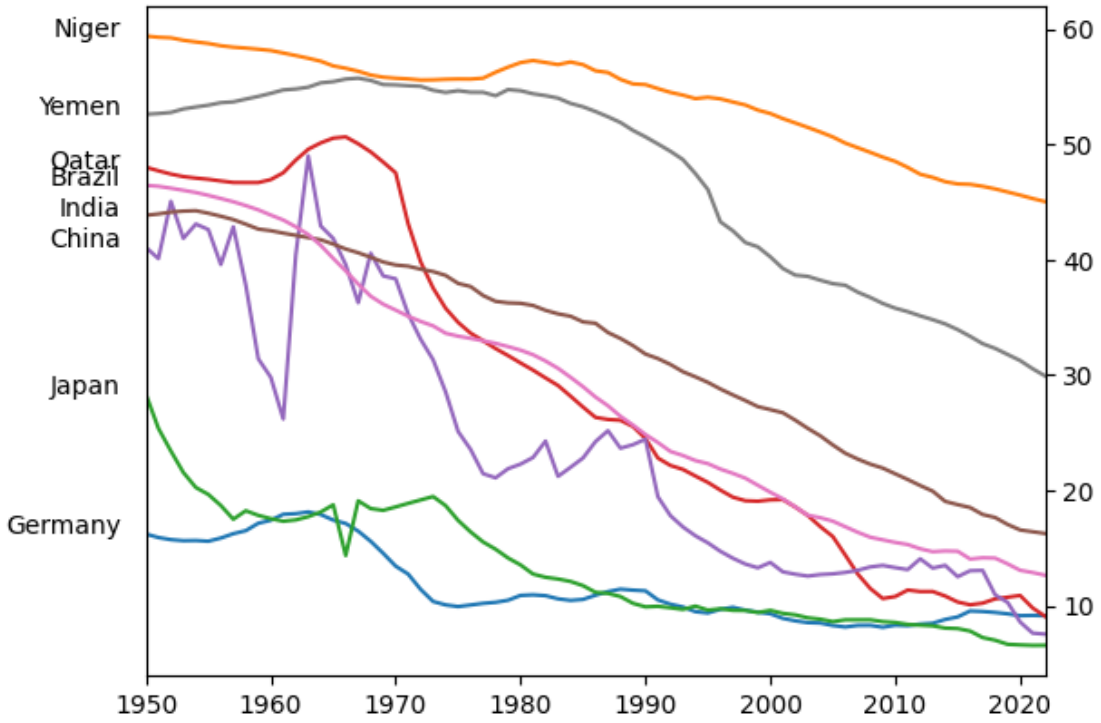
[12]:
```
indicatorCountry=(data["Location"]=="Germany")
dataCountry=data[indicatorCountry]
fig=plt.figure()
plt.plot(dataCountry["Time"],dataCountry["CBR"],label="crude birth rate")
plt.plot(dataCountry["Time"],dataCountry["CDR"],label="crude death rate")
plt.legend()
plt.tight_layout()
plt.show()
```



1.2.2 Birth rates some selected countries over years

[14]:
```
countrySelList=["Germany","Niger","Japan","Qatar",\
        "China","India","Brazil","Yemen"]

fig=plt.figure()
ax=fig.add_subplot()

for country in countrySelList:
    dataCountry=data[data["Location"]==country]
    ax.plot(dataCountry["Time"],dataCountry["CBR"],label=country)

minyear=data["Time"].min()
maxyear=data["Time"].max()
ax.set_xlim([minyear,maxyear])


# explicit legend
#plt.legend()
```

```
# directl labelling
if True:
    ax.yaxis.tick_right()
    for country in countrySelList:
        y=data.loc[(data["Location"]==country)&(data["Time"]==minyear),"CBR"].
 ↪iloc[0]
        ax.text(x=minyear-2,y=y,s=country,horizontalalignment="right")

plt.show()
```

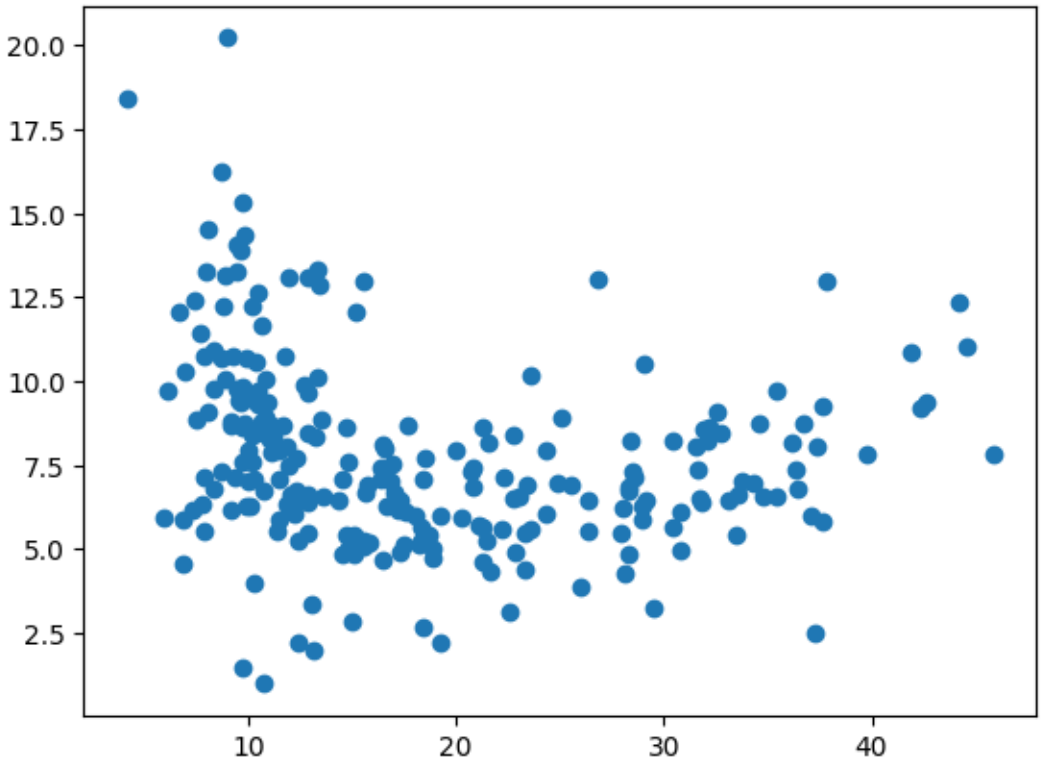


[15]:
```
# could now proceed to annotate / discuss peculiar artefacts in data:
# in japan 1966: "fire horse superstition"
# in china: great leap forward and one child policy
```

1.2.3 Birth and death rates in a given year for all countries

[17]:
```
year=2019
dataYear=data[(data["Time"]==year)]
```

[18]:
```
plt.scatter(dataYear["CBR"],dataYear["CDR"])
plt.show()
```



[19]:
```
# now: we don't know which country is which; too many for legend
# try hover technique
```

[20]:
```
import plotly.express as px
```

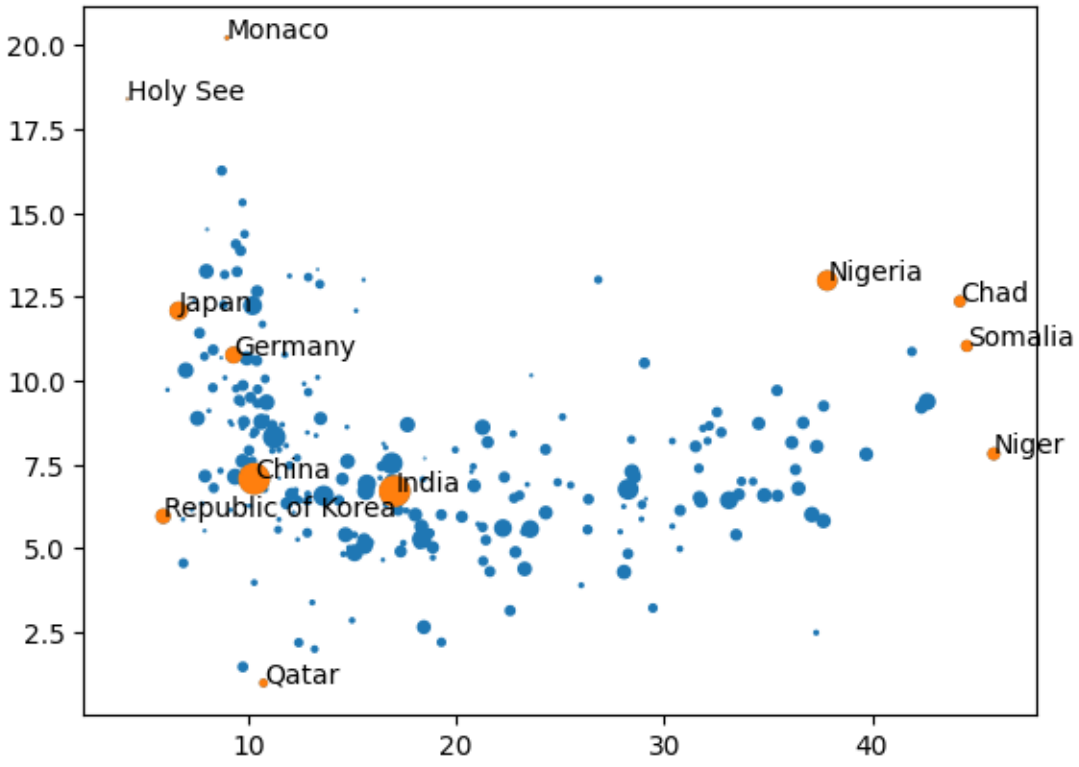
[26]:
```
fig = px.scatter(dataYear, x="CBR", y="CDR", hover_data=['Location'],\
        #size="TPopulation1Jan"\
        #color="Under5Deaths"\
        )
fig.show()
```

[36]:
```
# also experiment: map other columns to additional attributes

# but this is difficult for getting a "single glance impression"
# instead: static plot with some directly labelled "interesting" countries
```

[22]:
```
countrySelList=["Germany","Chad","Somalia","Niger","Monaco","Holy See","Japan",\
        "Qatar","Nigeria","Republic of Korea","China","India"]
```

[23]:
```
sList=1E-1*(np.array(dataYear["TPopulation1Jan"])**0.5)
sListSel=1.01E-1*(np.array(dataYear.loc[dataYear["Location"].
 ↪isin(countrySelList),"TPopulation1Jan"])**0.5)
```

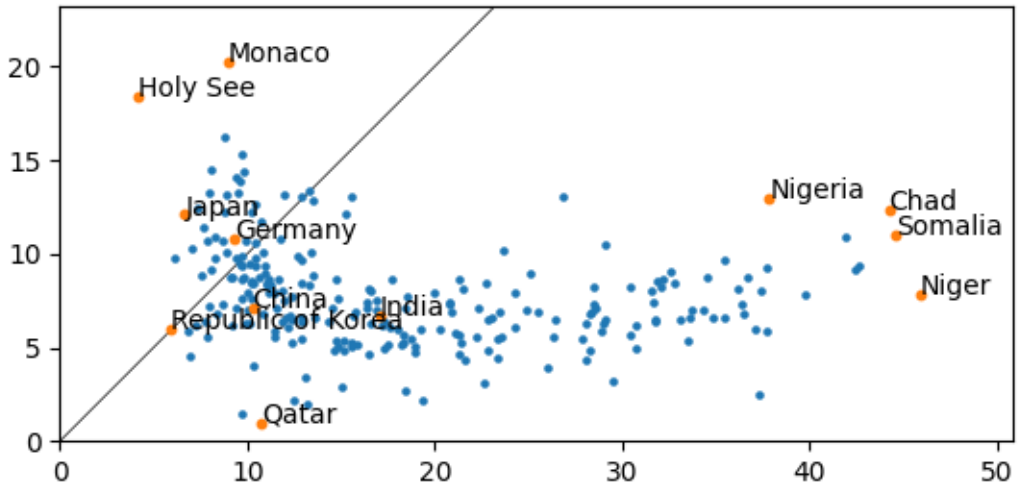
```
plt.scatter(dataYear["CBR"],dataYear["CDR"],\
        s=sList)
plt.scatter(dataYear.loc[dataYear["Location"].isin(countrySelList),"CBR"],\
        dataYear.loc[dataYear["Location"].
  ↪isin(countrySelList),"CDR"],s=sListSel)
ax=plt.gca()
for country in countrySelList:
    x,y=np.array(dataYear.loc[dataYear["Location"]==country,["CBR","CDR"]])[0]
    ax.annotate(country,(x,y))
plt.show()
```



[27]:
```
# fine tuning:
# * e.g. positioning of annotation
```

[28]:
```
plt.scatter(dataYear["CBR"],dataYear["CDR"],s=5)
plt.scatter(dataYear.loc[dataYear["Location"].isin(countrySelList),"CBR"],\
        dataYear.loc[dataYear["Location"].isin(countrySelList),"CDR"],\
        s=10)
ax=plt.gca()
for country in countrySelList:
    x,y=np.array(dataYear.loc[dataYear["Location"]==country,["CBR","CDR"]])[0]
```

```
    ax.annotate(country,(x,y))
# ax aspect
ax.set_aspect(1.)
# manual axes limits
CBRmax,CDRmax=dataYear[["CBR","CDR"]].max()
CBRmax+=5
CDRmax+=3
ax.set_xlim([0,CBRmax])
ax.set_ylim([0,CDRmax])
# "balance line"
ax.plot([0,CDRmax],[0,CDRmax],zorder=-1,lw=0.5,c="k")
plt.show()
```



[30]:
```
# movement over time
from matplotlib.animation import FuncAnimation
matplotlib.rc('animation',html='jshtml')
```

[31]:
```
fig=matplotlib.figure.Figure(figsize=(8,4))
ax=fig.add_subplot(aspect=1.)
pltobj_pts=ax.scatter([],[],s=5)

# manual axes limits
ax.set_xlim([0,60])
ax.set_ylim([0,30])


def update(year):
    dataYear=data[(data["Time"]==year)]
    # now update scatter-plot object with new vertex positions
```

```
    pltobj_pts.set_offsets(np.array(dataYear[["CBR","CDR"]]))

ani = FuncAnimation(fig, update, frames=np.arange(1955, 2022),
        blit=True,interval=2*1000/20)

ani
```

[31]:
```
<matplotlib.animation.FuncAnimation at 0x7f416164c550>
```

[ ]:
```

```

2023-04-17_WorldDemographics-Test-02

May 15, 2023

[1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

%matplotlib inline
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
```

1 Exploration of UN world population data

available at: https://population.un.org/wpp/Download/Standard/CSV/

Available columns in dataset: * Total Population, as of 1 January (thousands) * Total Population, as of 1 July (thousands) * Male Population, as of 1 July (thousands) * Female Population, as of 1 July (thousands) * Population Density, as of 1 July (persons per square km) * Population Sex Ratio, as of 1 July (males per 100 females) * Median Age, as of 1 July (years) * Natural Change, Births minus Deaths (thousands) * Rate of Natural Change (per 1,000 population) * Population Change (thousands) * Population Growth Rate (percentage) * Population Annual Doubling Time (years) * Births (thousands) * Births by women aged 15 to 19 (thousands) * Crude Birth Rate (births per 1,000 population) * Total Fertility Rate (live births per woman) * Net Reproduction Rate (surviving daughters per woman) * Mean Age Childbearing (years) * Sex Ratio at Birth (males per 100 female births) * Total Deaths (thousands) * Male Deaths (thousands) * Female Deaths (thousands) * Crude Death Rate (deaths per 1,000 population) * Life Expectancy at Birth, both sexes (years) * Male Life Expectancy at Birth (years) * Female Life Expectancy at Birth (years) * Life Expectancy at Age 15, both sexes (years) * Male Life Expectancy at Age 15 (years) * Female Life Expectancy at Age 15 (years) * Life Expectancy at Age 65, both sexes (years) * Male Life Expectancy at Age 65 (years) * Female Life Expectancy at Age 65 (years) * Life Expectancy at Age 80, both sexes (years) * Male Life Expectancy at Age 80 (years) * Female Life Expectancy at Age 80 (years) * Infant Deaths, under age 1 (thousands) * Infant Mortality Rate (infant deaths per 1,000 live births) * Live births Surviving to Age 1 (thousands) * Deaths under age 5 (thousands) * Under-five Mortality Rate (deaths under age 5 per 1,000 live births) * Mortality before Age 40, both sexes (deaths under age 40 per 1,000 live births) * Male mortality before Age 40 (deaths under age 40 per 1,000 male live births) * Female mortality before Age 40 (deaths under age 40 per 1,000 female live births) * Mortality before Age 60, both sexes (deaths under age 60 per 1,000 live births) * Male mortality before Age 60 (deaths under age 60 per 1,000 male live births) * Female mortality before Age 60 (deaths under age 60 per 1,000 female live births) * Mortality between Age 15 and

50, both sexes (deaths under age 50 per 1,000 alive at age 15) * Male mortality between Age 15 and 50 (deaths under age 50 per 1,000 males alive at age 15) * Female mortality between Age 15 and 50 (deaths under age 50 per 1,000 females alive at age 15) * Mortality between Age 15 and 60, both sexes (deaths under age 60 per 1,000 alive at age 15) * Male mortality between Age 15 and 60 (deaths under age 60 per 1,000 males alive at age 15) * Female mortality between Age 15 and 60 (deaths under age 60 per 1,000 females alive at age 15) * Net Number of Migrants (thousands) * Net Migration Rate (per 1,000 population)

Each row corresponds to one country or region, and one year, for the interval 1950 to 2100 where future years contain projected data. The dataset contains rows for "aggregated regions" (such as continents). We will later discard the projected data and the aggregated regions.

1.1 Import data

[2]:
```
# set up column data types
headerLine="SortOrder,LocID,Notes,ISO3_code,ISO2_code,SDMX_code,LocTypeID,LocTypeName,"+\
    "ParentID,Location,VarID,Variant,"+\
    ␣
  ↪"Time,TPopulation1Jan,TPopulation1July,TPopulationMale1July,TPopulationFemale1July,"+\
    ␣
  ↪"PopDensity,PopSexRatio,MedianAgePop,NatChange,NatChangeRT,PopChange,PopGrowthRate,"+\
    ␣
  ↪"DoublingTime,Births,Births1519,CBR,TFR,NRR,MAC,SRB,Deaths,DeathsMale,DeathsFemale,"+\
    ␣
  ↪"CDR,LEx,LExMale,LExFemale,LE15,LE15Male,LE15Female,LE65,LE65Male,LE65Female,"+\
    "LE80,LE80Male,LE80Female,InfantDeaths,IMR,LBsurvivingAge1,Under5Deaths,"+\
    ␣
  ↪"Q5,Q0040,Q0040Male,Q0040Female,Q0060,Q0060Male,Q0060Female,Q1550,Q1550Male,"+\
    "Q1550Female,Q1560,Q1560Male,Q1560Female,NetMigrations,CNMR"

dtypeDict={}
for x in headerLine.split(","):
    dtypeDict[x]=np.float64
for x in "SortOrder,LocID,LocTypeID,ParentID,VarID,Time".split(","):
    dtypeDict[x]=np.int32
for x in "Notes,ISO3_code,SDMX_code,ISO2_code,LocTypeName,Location,Variant".
  ↪split(","):
    dtypeDict[x]=str
```

[3]:
```
dataFull=pd.read_csv("data/WPP2022_Demographic_Indicators_Medium.
  ↪csv",sep=",",dtype=dtypeDict)
```

[4]:
```
# check list of columns; labels much shorter than above, but consistent
dataFull.keys()
```

[4]:
```
Index(['SortOrder', 'LocID', 'Notes', 'ISO3_code', 'ISO2_code', 'SDMX_code',
       'LocTypeID', 'LocTypeName', 'ParentID', 'Location', 'VarID', 'Variant',
       'Time', 'TPopulation1Jan', 'TPopulation1July', 'TPopulationMale1July',
       'TPopulationFemale1July', 'PopDensity', 'PopSexRatio', 'MedianAgePop',
       'NatChange', 'NatChangeRT', 'PopChange', 'PopGrowthRate',
       'DoublingTime', 'Births', 'Births1519', 'CBR', 'TFR', 'NRR', 'MAC',
       'SRB', 'Deaths', 'DeathsMale', 'DeathsFemale', 'CDR', 'LEx', 'LExMale',
       'LExFemale', 'LE15', 'LE15Male', 'LE15Female', 'LE65', 'LE65Male',
       'LE65Female', 'LE80', 'LE80Male', 'LE80Female', 'InfantDeaths', 'IMR',
       'LBsurvivingAge1', 'Under5Deaths', 'Q5', 'Q0040', 'Q0040Male',
       'Q0040Female', 'Q0060', 'Q0060Male', 'Q0060Female', 'Q1550',
       'Q1550Male', 'Q1550Female', 'Q1560', 'Q1560Male', 'Q1560Female',
       'NetMigrations', 'CNMR'],
      dtype='object')
```

[6]:
```
# confirm time range of data:
minyear,maxyear=[dataFull["Time"].min(),dataFull["Time"].max()]
print([minyear,maxyear])
```

```
[1950, 2101]
```

[7]:
```
# how many rows and columns does the dataset have?
dataFull.shape
```

[7]:
```
(43472, 67)
```

1.1.1 Filter for countries and non-projected data

Remove all rows that contain data which is: * not related to a single country (e.g. to whole regions) * contains projected data about the future

[8]:
```
countryIndicator=(dataFull["LocTypeName"]=="Country/Area")
timeIndicator=(dataFull["Time"]<=2022)

data=dataFull[countryIndicator & timeIndicator]

print(f"remaining columns: {data.shape[0]}")
```

```
remaining columns: 17301
```

[20]:
```
minyear,maxyear=[data["Time"].min(),data["Time"].max()]
print([minyear,maxyear])
```

```
[1950, 2022]
```

[9]:
```
# for first orientation, print a list of countries contained in the dataset:
countryList=data["Location"].unique()
```

```
print(countryList)
```

```
['Burundi' 'Comoros' 'Djibouti' 'Eritrea' 'Ethiopia' 'Kenya' 'Madagascar'
 'Malawi' 'Mauritius' 'Mayotte' 'Mozambique' 'Réunion' 'Rwanda'
 'Seychelles' 'Somalia' 'South Sudan' 'Uganda'
 'United Republic of Tanzania' 'Zambia' 'Zimbabwe' 'Angola' 'Cameroon'
 'Central African Republic' 'Chad' 'Congo'
 'Democratic Republic of the Congo' 'Equatorial Guinea' 'Gabon'
 'Sao Tome and Principe' 'Algeria' 'Egypt' 'Libya' 'Morocco' 'Sudan'
 'Tunisia' 'Western Sahara' 'Botswana' 'Eswatini' 'Lesotho' 'Namibia'
 'South Africa' 'Benin' 'Burkina Faso' 'Cabo Verde' "Côte d'Ivoire"
 'Gambia' 'Ghana' 'Guinea' 'Guinea-Bissau' 'Liberia' 'Mali' 'Mauritania'
 'Niger' 'Nigeria' 'Saint Helena' 'Senegal' 'Sierra Leone' 'Togo'
 'Kazakhstan' 'Kyrgyzstan' 'Tajikistan' 'Turkmenistan' 'Uzbekistan'
 'China' 'China, Hong Kong SAR' 'China, Macao SAR'
 'China, Taiwan Province of China' "Dem. People's Republic of Korea"
 'Japan' 'Mongolia' 'Republic of Korea' 'Afghanistan' 'Bangladesh'
 'Bhutan' 'India' 'Iran (Islamic Republic of)' 'Maldives' 'Nepal'
 'Pakistan' 'Sri Lanka' 'Brunei Darussalam' 'Cambodia' 'Indonesia'
 "Lao People's Democratic Republic" 'Malaysia' 'Myanmar' 'Philippines'
 'Singapore' 'Thailand' 'Timor-Leste' 'Viet Nam' 'Armenia' 'Azerbaijan'
 'Bahrain' 'Cyprus' 'Georgia' 'Iraq' 'Israel' 'Jordan' 'Kuwait' 'Lebanon'
 'Oman' 'Qatar' 'Saudi Arabia' 'State of Palestine' 'Syrian Arab Republic'
 'Türkiye' 'United Arab Emirates' 'Yemen' 'Belarus' 'Bulgaria' 'Czechia'
 'Hungary' 'Poland' 'Republic of Moldova' 'Romania' 'Russian Federation'
 'Slovakia' 'Ukraine' 'Denmark' 'Estonia' 'Faroe Islands' 'Finland'
 'Guernsey' 'Iceland' 'Ireland' 'Isle of Man' 'Jersey' 'Latvia'
 'Lithuania' 'Norway' 'Sweden' 'United Kingdom' 'Albania' 'Andorra'
 'Bosnia and Herzegovina' 'Croatia' 'Gibraltar' 'Greece' 'Holy See'
 'Italy' 'Kosovo (under UNSC res. 1244)' 'Malta' 'Montenegro'
 'North Macedonia' 'Portugal' 'San Marino' 'Serbia' 'Slovenia' 'Spain'
 'Austria' 'Belgium' 'France' 'Germany' 'Liechtenstein' 'Luxembourg'
 'Monaco' 'Netherlands' 'Switzerland' 'Anguilla' 'Antigua and Barbuda'
 'Aruba' 'Bahamas' 'Barbados' 'Bonaire, Sint Eustatius and Saba'
 'British Virgin Islands' 'Cayman Islands' 'Cuba' 'Curaçao' 'Dominica'
 'Dominican Republic' 'Grenada' 'Guadeloupe' 'Haiti' 'Jamaica'
 'Martinique' 'Montserrat' 'Puerto Rico' 'Saint Barthélemy'
 'Saint Kitts and Nevis' 'Saint Lucia' 'Saint Martin (French part)'
 'Saint Vincent and the Grenadines' 'Sint Maarten (Dutch part)'
 'Trinidad and Tobago' 'Turks and Caicos Islands'
 'United States Virgin Islands' 'Belize' 'Costa Rica' 'El Salvador'
 'Guatemala' 'Honduras' 'Mexico' 'Nicaragua' 'Panama' 'Argentina'
 'Bolivia (Plurinational State of)' 'Brazil' 'Chile' 'Colombia' 'Ecuador'
 'Falkland Islands (Malvinas)' 'French Guiana' 'Guyana' 'Paraguay' 'Peru'
 'Suriname' 'Uruguay' 'Venezuela (Bolivarian Republic of)' 'Bermuda'
 'Canada' 'Greenland' 'Saint Pierre and Miquelon'
 'United States of America' 'Australia' 'New Zealand' 'Fiji'
```

```
'New Caledonia' 'Papua New Guinea' 'Solomon Islands' 'Vanuatu' 'Guam'
'Kiribati' 'Marshall Islands' 'Micronesia (Fed. States of)' 'Nauru'
'Northern Mariana Islands' 'Palau' 'American Samoa' 'Cook Islands'
'French Polynesia' 'Niue' 'Samoa' 'Tokelau' 'Tonga' 'Tuvalu'
'Wallis and Futuna Islands']
```

1.1.2 Shorten some country names for more compact labelling

[10]:
```
substitutions={"United States of America":"USA"}
```

[11]:
```
for long,short in substitutions.items():
    data.loc[data["Location"]==long,"Location"]=short
```

1.1.3 Load and merge with continent data

Now load load a separate CSV file which contains for each country the associated continent. It has been extracted from the metadata provided with the original dataset at:

https://population.un.org/wpp/Download/Documentation/Documentation/

Specifically, from this file:

https://population.un.org/wpp/Download/Files/4_Metadata/WPP2022_F01_LOCATIONS.XLSX

Shortened some continent names.

[12]:
```
dataContinents=pd.read_csv("data/UN_population/continent_association.csv",\
        dtype={"SDMX_code":str,"Continent":str})
```

[13]:
```
dataContinents.dtypes
```

[13]:
```
SDMX_code    object
Continent    object
dtype: object
```

[14]:
```
Continents=list(dataContinents["Continent"].unique())
print(Continents)
```

```
['Africa', 'Asia', 'Europe', 'Central and South America', 'North America',
'Oceania']
```

[15]:
```
# merge with continent data
data=pd.merge(data,dataContinents,on="SDMX_code")
```

[17]:
```
# this will later allow us to assign colors according to continents
continentColors={a:colors[i] for i,a in enumerate(Continents)}
```

1.2 Visualizations

1.2.1 Key indicators of some selected countries over years

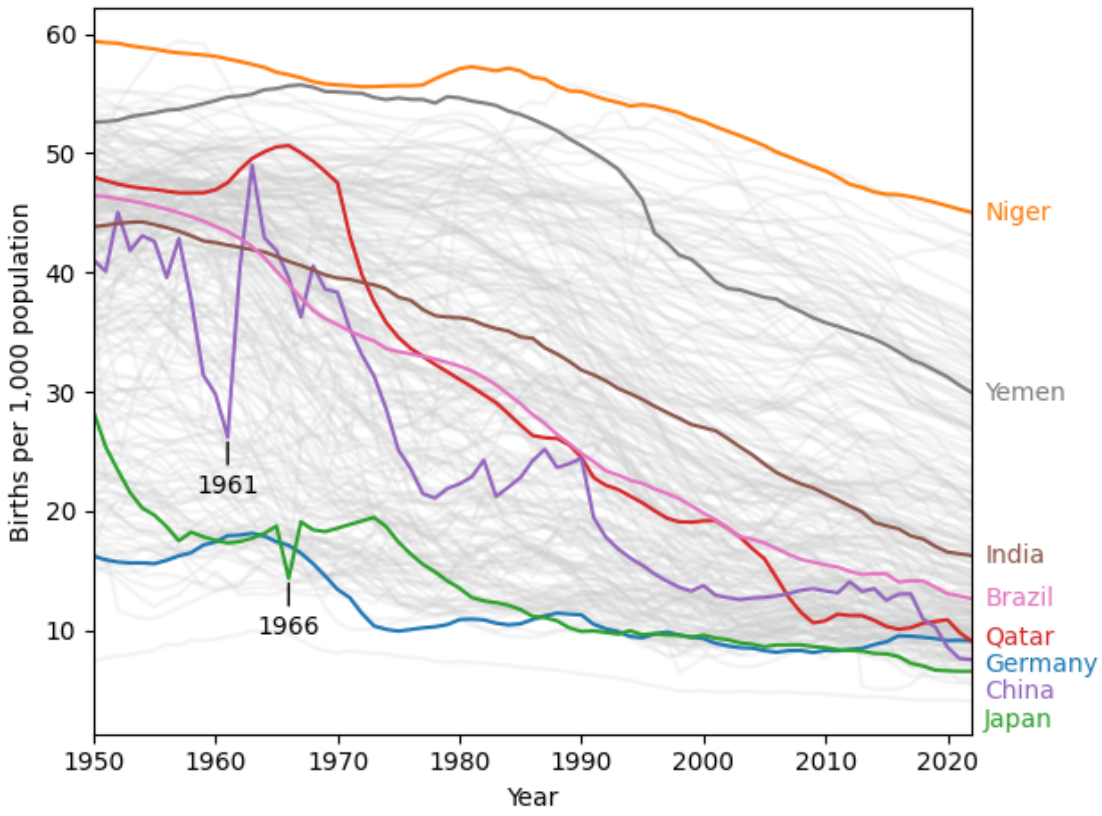
The following plots show: * birth rate * child mortality * median age

Each for a (relevant) selection of countries over time.

Birth rate

[21]:
```
countrySelList=["Germany","Niger","Japan","Qatar",\
        "China","India","Brazil","Yemen"]

fig=plt.figure()
ax=fig.add_subplot()

for country in countryList:
    if country not in countrySelList:
        dataCountry=data[data["Location"]==country]
        ax.plot(dataCountry["Time"],dataCountry["CBR"],c=cm.gray(0.8),alpha=0.
 ↪2,zorder=-1)


for country in countrySelList:
    dataCountry=data[data["Location"]==country]
    ax.plot(dataCountry["Time"],dataCountry["CBR"],label=country)


# directl labelling
offsets={\
        "Germany":-10,"Niger":0,"Japan":-20,"Qatar":2,\
        "China":-13,"India":0,"Brazil":0,"Yemen":0\
        }
for i,country in enumerate(countrySelList):
    y=data.loc[(data["Location"]==country)&(data["Time"]==maxyear),"CBR"].
 ↪iloc[0]
    ax.
 ↪annotate(text=country,xy=(maxyear,y),va="center",ha="left",textcoords="offset␣
 ↪points",\
            xytext=(5,offsets[country]),c=colors[i],\
            arrowprops=None)

# fire horse annotation
y=data.loc[(data["Location"]=="Japan")&(data["Time"]==1966),"CBR"].iloc[0]
ax.annotate(text="1966",xy=(1966,y),va="center",ha="center",textcoords="offset␣
 ↪points",\
            xytext=(0,-20),c="k",\
            arrowprops={"arrowstyle":"-"})
```

```
# great leap
y=data.loc[(data["Location"]=="China")&(data["Time"]==1961),"CBR"].iloc[0]
ax.annotate(text="1961",xy=(1961,y),va="center",ha="center",textcoords="offset␣
 ↪points",\
            xytext=(0,-20),c="k",\
            arrowprops={"arrowstyle":"-"})

plt.xlabel("Year")
plt.ylabel("Births per 1,000 population")
ax.set_xlim([minyear,maxyear])
plt.tight_layout()
plt.show()
```

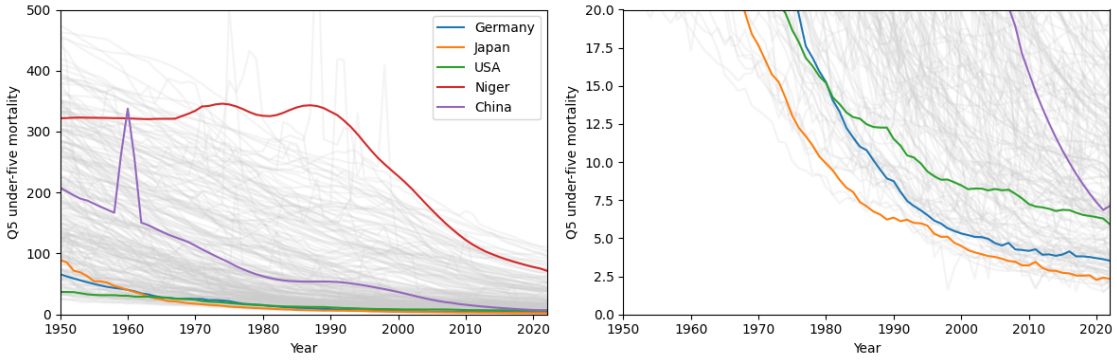


Annotations: * 1966, Japan: "Fire horse superstition" leads to a lowered birth rate in that year * 1961, China: Birth rate dropped significantly during the "Great Leap Forward" from 1958 to 1962.

Q5 Child mortality rate

- Under-five Mortality Rate (deaths under age 5 per 1,000 live births)

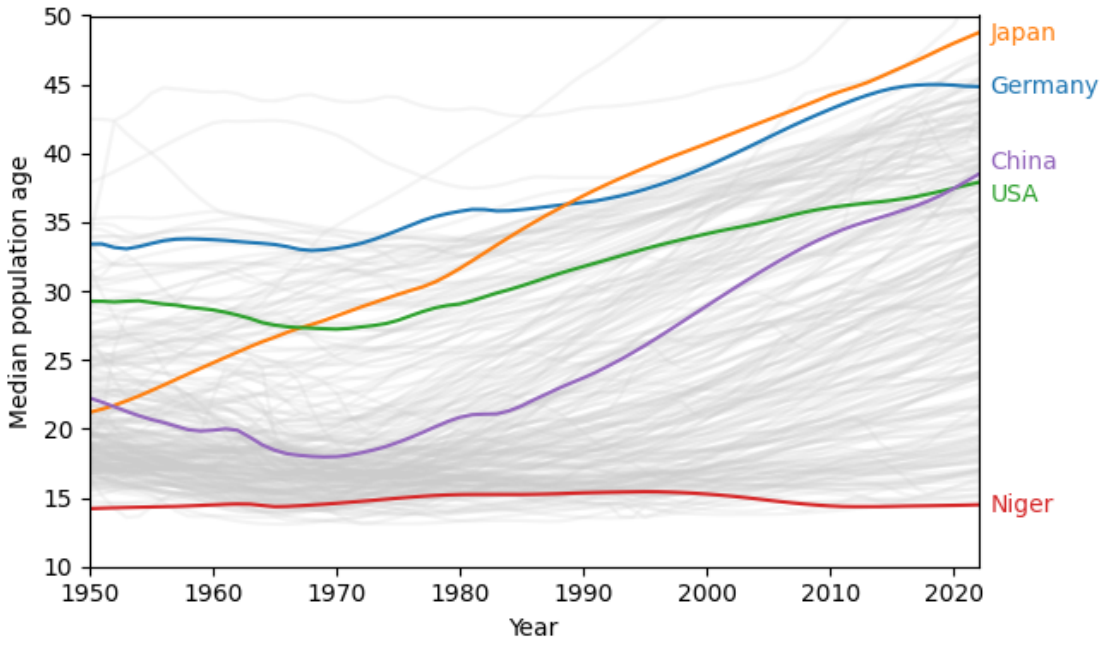
[22]:
```
countrySel=["Germany","Japan","USA","Niger","China"]

fig=plt.figure(figsize=(12,4))
ax=fig.add_subplot(1,2,1)
for country in countryList:
    if country not in countrySel:
        dataCountry=data[data["Location"]==country]
        plt.plot(dataCountry["Time"],dataCountry["Q5"],c=cm.gray(0.8),alpha=0.
 ↪2,zorder=-1)
for i,country in enumerate(countrySel):
    dataCountry=data[data["Location"]==country]
    plt.plot(dataCountry["Time"],dataCountry["Q5"],c=colors[i],label=country)
plt.xlim([1950,2022])
plt.ylim([0,500])
plt.legend()
plt.xlabel("Year")
plt.ylabel("Q5 under-five mortality")

ax=fig.add_subplot(1,2,2)
for country in countryList:
    if country not in countrySel:
        dataCountry=data[data["Location"]==country]
        plt.plot(dataCountry["Time"],dataCountry["Q5"],c=cm.gray(0.8),alpha=0.
 ↪2,zorder=-1)
for i,country in enumerate(countrySel):
    dataCountry=data[data["Location"]==country]
    plt.plot(dataCountry["Time"],dataCountry["Q5"],c=colors[i])
plt.xlim([1950,2022])
plt.ylim([0,20])

plt.xlabel("Year")
plt.ylabel("Q5 under-five mortality")

plt.tight_layout()
plt.show()
```



- In China, the "Great Leap Forward" lead to a strongly increased child-mortality around 1960. Visualization discussion:
- Did not go for direct labelling here for several reasons: values all really close at end; no space between sub-figures; fewer distinct countries, so also less confusing.

Median population age

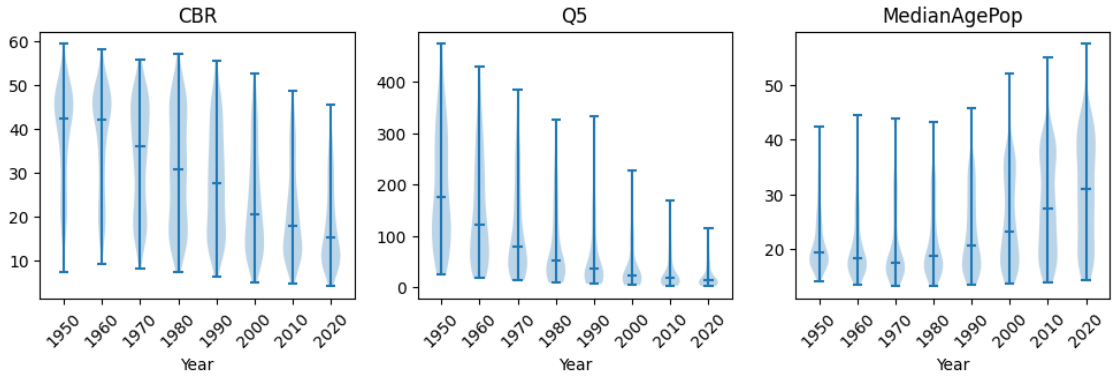
[23]:
```
countrySelList=["Germany","Japan","USA","Niger","China"]

fig=plt.figure(figsize=(12,4))
ax=fig.add_subplot(1,2,1)

for country in countryList:
    if country not in countrySelList:
        dataCountry=data[data["Location"]==country]
        ax.plot(dataCountry["Time"],dataCountry["MedianAgePop"],c=cm.gray(0.
 ↪8),alpha=0.2,zorder=-1)


for i,country in enumerate(countrySelList):
    dataCountry=data[data["Location"]==country]
    plt.plot(dataCountry["Time"],dataCountry["MedianAgePop"],c=colors[i])

# directl labelling
offsets={\
        "Germany":0,"Japan":0,"USA":-5,"Niger":0,"China":+5
        }
for i,country in enumerate(countrySelList):
    y=data.
 ↪loc[(data["Location"]==country)&(data["Time"]==maxyear),"MedianAgePop"].
 ↪iloc[0]
    ax.
 ↪annotate(text=country,xy=(maxyear,y),va="center",ha="left",textcoords="offset␣
 ↪points",\
            xytext=(5,offsets[country]),c=colors[i],\
            arrowprops=None)


plt.ylim([10,50])
plt.xlabel("Year")
plt.ylabel("Median population age")
ax.set_xlim([minyear,maxyear])
plt.tight_layout()

plt.show()
```



1.2.2 Evolution of distribution over countries

- Visualize same three quantities
- Instead of highlighting individual countries, show distribution (and its evolution over time) by violin plots.

[24]:
```
yearsSel=range(1950,2022,10)
nYearsSel=len(yearsSel)
dataG=data.groupby("Time")
keyList=["CBR","Q5","MedianAgePop"]
nKeys=len(keyList)
fig=plt.figure(figsize=(4*nKeys,3))
for i,key in enumerate(keyList):
    ax=fig.add_subplot(1,nKeys,i+1)
    ptdata=[]
    for year,dat in dataG:
        if year in yearsSel:
            ptdata.append(dat[key])
    #plt.boxplot(ptdata,labels=yearsSel)
    ax.violinplot(ptdata,showmedians=True)
    ax.set_xticks(range(1,nYearsSel+1),yearsSel,rotation=45)
    plt.title(key)
    plt.xlabel("Year")
plt.show()
```

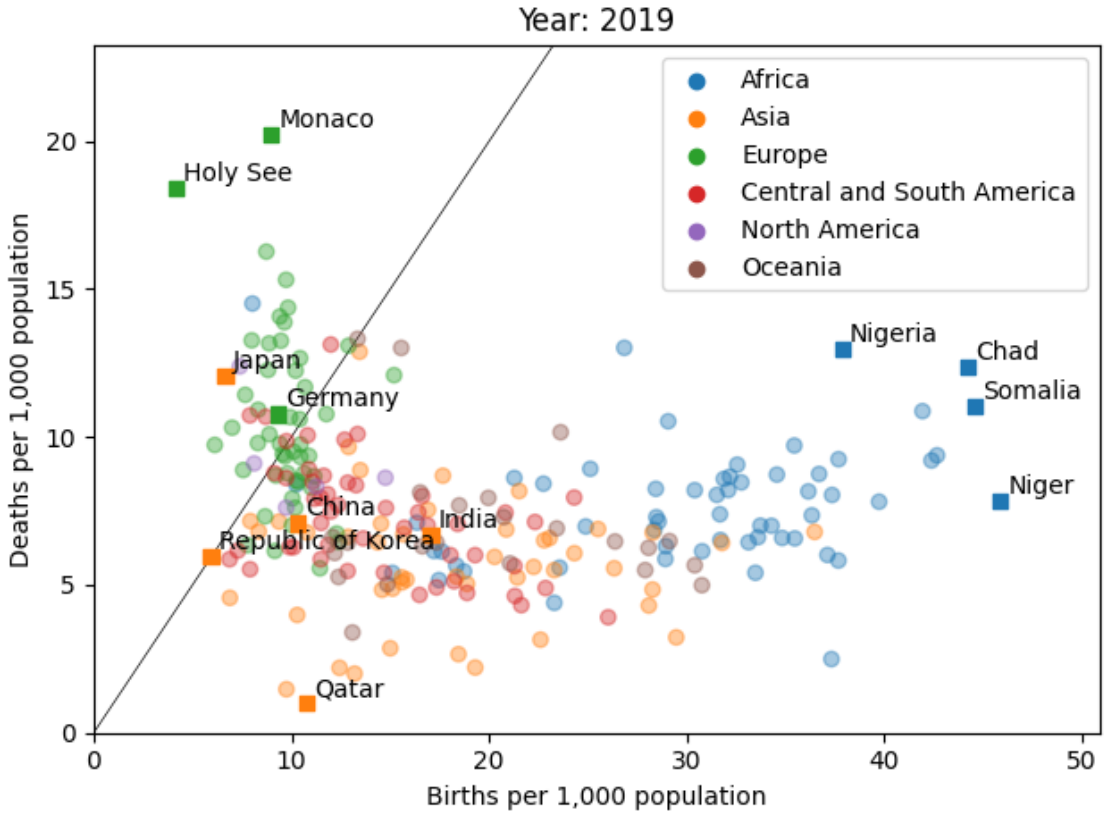


1.3 Birth and death rates in a given year for all countries

[25]:
```
# highlight these countries by direct labelling
countrySelList=["Germany","Chad","Somalia","Niger","Monaco","Holy See","Japan",\
        "Qatar","Nigeria","Republic of Korea","China","India"]
```

[26]:
```
year=2019
dataYear=data[data["Time"]==year]
sel=dataYear["Location"].isin(countrySelList)
for s in [False,True]:
    plt.scatter(dataYear.loc[sel==s,"CBR"],dataYear.loc[sel==s,"CDR"],\
            alpha={False:0.4,True:1.}[s],marker={False:"o",True:"s"}[s],\
            c=dataYear.loc[sel==s,"Continent"].map(continentColors))

ax=plt.gca()
for country in countrySelList:
    x,y=np.array(dataYear.loc[dataYear["Location"]==country,["CBR","CDR"]])[0]
    ax.annotate(country,(x,y),xycoords="data",xytext=(3,3),textcoords="offset␣
 ↪points",\
    arrowprops=None)

# add continent legend
for i,cont in enumerate(Continents):
    plt.scatter([],[],marker="o",color=continentColors[cont],label=cont)

# ax aspect
#ax.set_aspect(1.)
# manual axes limits
CBRmax,CDRmax=dataYear[["CBR","CDR"]].max()
CBRmax+=5
CDRmax+=3
ax.set_xlim([0,CBRmax])
```

```
ax.set_ylim([0,CDRmax])
# "balance line"
ax.plot([0,CDRmax],[0,CDRmax],zorder=-1,lw=0.5,c="k")
plt.legend(loc="upper right")
plt.title(f"Year: {year}")
plt.xlabel("Births per 1,000 population")
plt.ylabel("Deaths per 1,000 population")
plt.tight_layout()
plt.show()
```

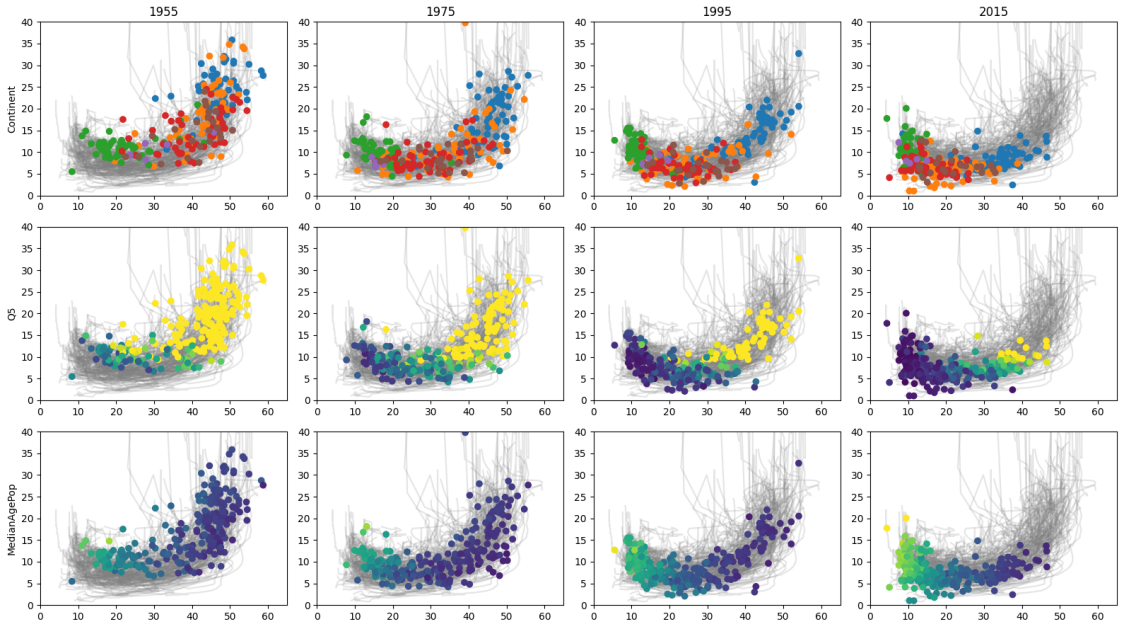


Visualize evolution over time

- Use small multiplies
- First, as above use color to encode continent
- Then use color to encode under-five mortality and median age, as above.

[27]:
```
yearsSel=list(range(2015,1950,-20))[::-1]
#print(yearsSel)
nYearsSel=len(yearsSel)
keyList=["Continent","Q5","MedianAgePop"]
```

```
nRows=len(keyList)

colorFunction={"Continent":lambda x : x.map(continentColors),\
               "Q5": lambda x : cm.viridis(np.clip(x,0,100)/100),\
               "MedianAgePop": lambda x : cm.viridis((np.clip(x,10,50)-10)/40)\
               }

# background: lines of each country over time
fig=plt.figure(figsize=(4*nYearsSel,3*nRows))
for j,key in enumerate(keyList):
    for i,year in enumerate(yearsSel):
        ax=fig.add_subplot(nRows,nYearsSel,j*nYearsSel+i+1)
        for country in countryList:
            dataSel=data[data["Location"]==country]
            ax.plot(dataSel["CBR"],dataSel["CDR"],c=cm.gray(0.5),alpha=0.
 ↪2,zorder=-1)
        dataYear=data[data["Time"]==year]
        ax.
 ↪scatter(dataYear["CBR"],dataYear["CDR"],c=colorFunction[key](dataYear[key]))
        ax.set_ylim([0,40])
        ax.set_xlim([0,65])
        if j==0:
            plt.title(year)
        if i==0:
            plt.ylabel(key)
plt.tight_layout()
plt.show()
```



Color scales:

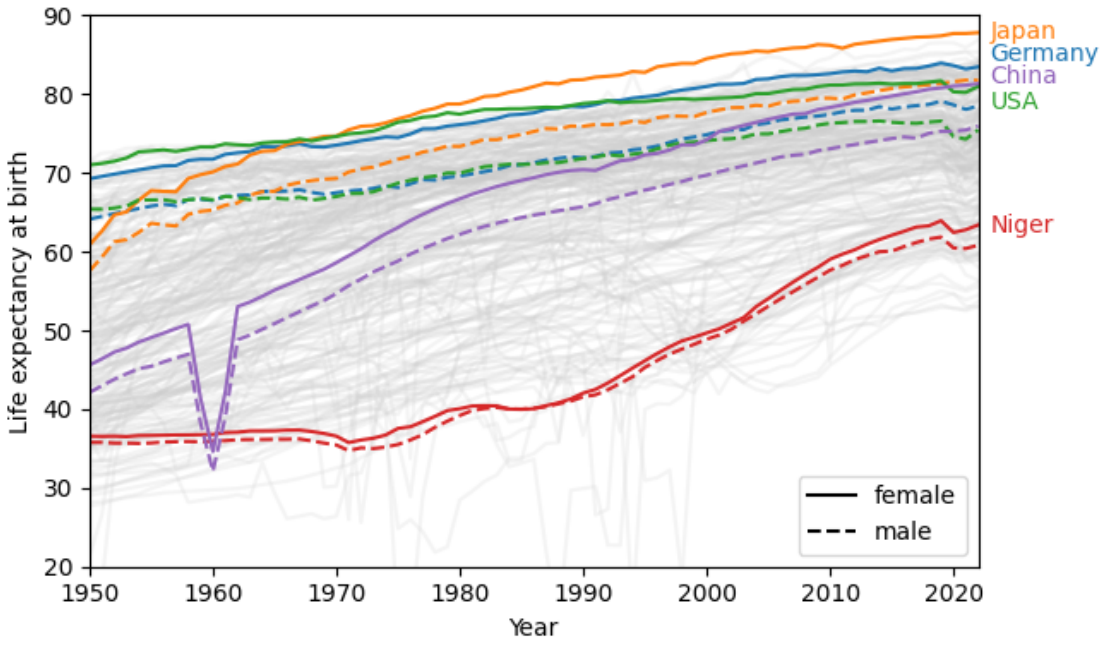
- Continents as above
- under-five mortality: 0 to 100+
- median age: 10- to 50+ #### Take aways:
- general trend towards lower birth rates, this is probably causally related to child mortality (but of course, we cannot know for sure based on this data)
- as birth rates lower, median age rises, and thus eventually so does the death rate

1.3.1 Life expectancy

Simple overview First, a simple overview. Evolution of life expectancy of women and men of some selected countries over time.

[28]:
```
fig=plt.figure(figsize=(12,4))
ax=fig.add_subplot(1,2,1)

countrySelList=["Germany","Japan","USA","Niger","China"]

for country in countryList:
    if country not in countrySelList:
        dataCountry=data[data["Location"]==country]
        ax.plot(dataCountry["Time"],dataCountry["LEx"],c=cm.gray(0.8),alpha=0.
 ↪2,zorder=-1)


for i,country in enumerate(countrySelList):
    dataCountry=data[data["Location"]==country]
    plt.plot(dataCountry["Time"],dataCountry["LExMale"],c=colors[i],ls="dashed")
    plt.
 ↪plot(dataCountry["Time"],dataCountry["LExFemale"],c=colors[i])#,ls="dotted")


# directl labelling
offsets={\
        "Germany":+5,"Japan":0,"USA":-7,"Niger":0,"China":+3
        }
for i,country in enumerate(countrySelList):
    y=data.loc[(data["Location"]==country)&(data["Time"]==maxyear),"LExFemale"].
 ↪iloc[0]
    ax.
 ↪annotate(text=country,xy=(maxyear,y),va="center",ha="left",textcoords="offset␣
 ↪points",\
            xytext=(5,offsets[country]),c=colors[i],\
            arrowprops=None)
```

```
plt.plot([],[],ls="solid",c="k",label="female")
plt.plot([],[],ls="dashed",c="k",label="male")
plt.legend(loc="lower right")
plt.ylim([20,90])
plt.xlabel("Year")
plt.ylabel("Life expectancy at birth")
ax.set_xlim([minyear,maxyear])
plt.tight_layout()

plt.show()
```

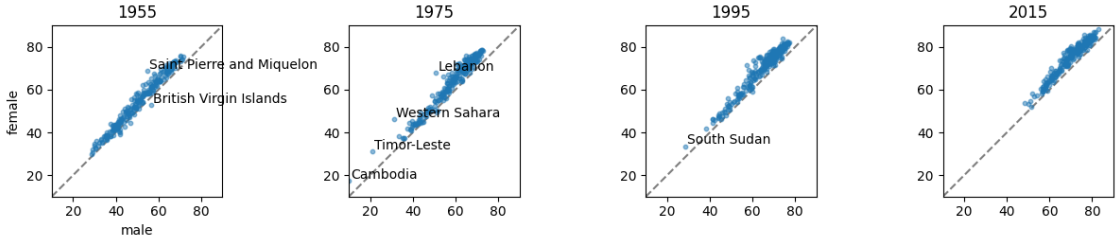


Background shows average life expectancy at birth of all countries for context.

Male vs female scatter plot

[29]:
```
yearsSel=list(range(2015,1950,-20))[::-1]
nYearsSel=len(yearsSel)

highlights={1955:["British Virgin Islands","Saint Pierre and Miquelon"],\
        1975:["Cambodia","Lebanon","Western Sahara","Timor-Leste"],\
        1995:["South Sudan"],2015:[]}

fig=plt.figure(figsize=(nYearsSel*3,3))
for i,year in enumerate(yearsSel):
    ax=fig.add_subplot(1,nYearsSel,i+1,aspect=1.)
```

```
    dataYear=data[data["Time"]==year]
    plt.
↪scatter(dataYear["LExMale"],dataYear["LExFemale"],s=10,marker="o",alpha=0.5)
    plt.xlim([10,90])
    plt.ylim([10,90])
    plt.plot([10,90],[10,90],c=cm.gray(0.5),ls="dashed",zorder=-1)

    # annotations
    for country in highlights[year]:
        x=dataYear.loc[(dataYear["Location"]==country),"LExMale"].iloc[0]
        y=dataYear.loc[(dataYear["Location"]==country),"LExFemale"].iloc[0]
        plt.annotate(country,xy=(x+1,y+1))

    plt.title(year)
    if i==0:
        plt.ylabel("female")
        plt.xlabel("male")
plt.tight_layout()
plt.show()
```



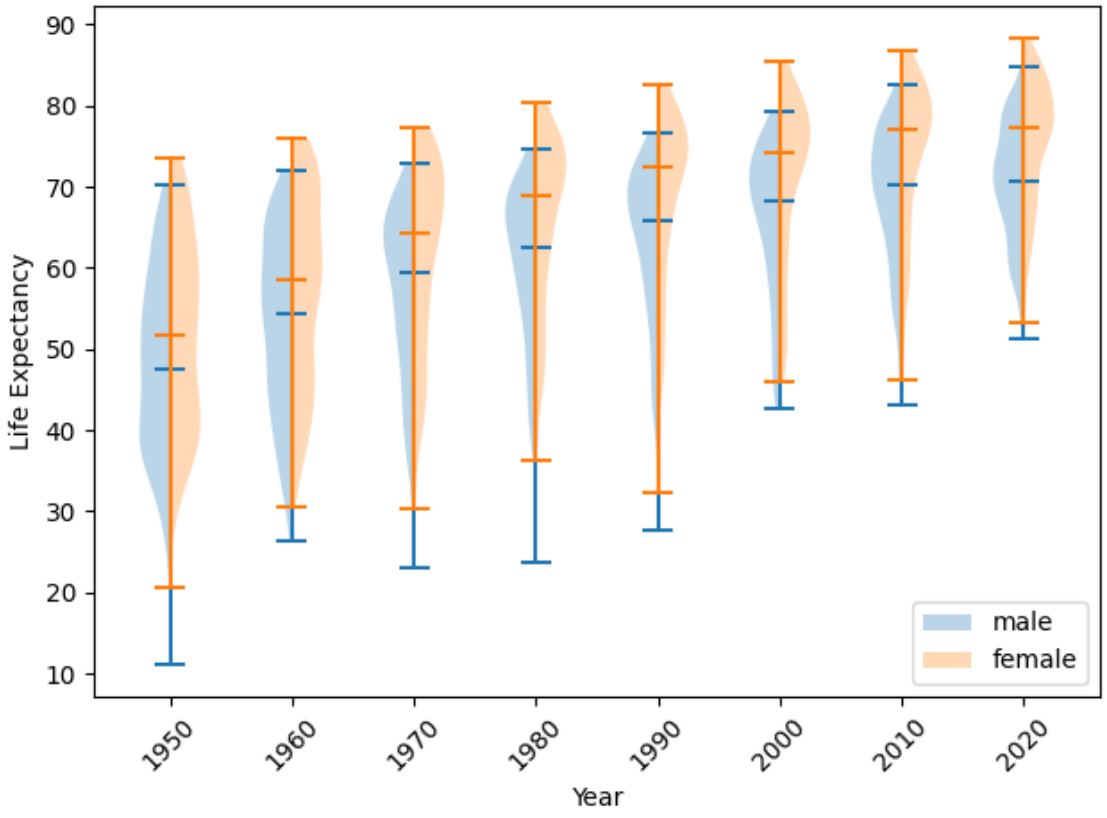
Half-violin plot

- show male and female life expectancy for various years as half-violin plot
- allows for decent statistical overview, and finer temporal resolution
- does not immediately allow to see correlation between the two, at level of individual countries

[30]:
```
# code based on:
# https://stackoverflow.com/questions/29776114/half-violin-plot-in-matplotlib
```

[31]:
```
yearsSel=range(1950,2022,10)
nYearsSel=len(yearsSel)
dataG=data.groupby("Time")
fig=plt.figure()
ax=fig.add_subplot()
ptdatam=[]
ptdataf=[]
for year,dat in dataG:
```

```
    if year in yearsSel:
        ptdatam.append(dat["LExMale"])
        ptdataf.append(dat["LExFemale"])
v1=ax.violinplot(ptdatam,showmeans=False, showextrema=True, showmedians=True)
for b in v1['bodies']:
    m = np.mean(b.get_paths()[0].vertices[:, 0])
    b.get_paths()[0].vertices[:, 0] = np.clip(b.get_paths()[0].vertices[:, 0],␣
 ↪-np.inf, m)

v2=ax.violinplot(ptdataf,showmeans=False, showextrema=True, showmedians=True)
for b in v2['bodies']:
    m = np.mean(b.get_paths()[0].vertices[:, 0])
    b.get_paths()[0].vertices[:, 0] = np.clip(b.get_paths()[0].vertices[:, 0],␣
 ↪m, np.inf)


ax.set_xticks(range(1,nYearsSel+1),yearsSel,rotation=45)
plt.ylabel("Life Expectancy")
plt.xlabel("Year")
plt.legend([v1['bodies'][0],v2['bodies'][0]],['male', 'female'],loc="lower␣
 ↪right")
plt.tight_layout()
plt.show()
```



1.4 More to explore

- In principle, this dataset allows for even more exploration, e.g.~of migration data.

1.5 Some final comments

- Plot types: scatter and line plots, a few violin plots; everything static in time; no interaction
- Not a single map in this notebook
- Used position and color for encoding quantities, this got us quite far
- Not a prerequisite for a good project to use all sorts of fancy plot types, more important: use them in a good way.
- But of course: feel free to be creative and explore/experiment with other plot types (e.g. networks), animations, interactive plots. But do not create ducks :)

[ ]:

2023-05-08_Example-1-ColorSpaces

May 8, 2023

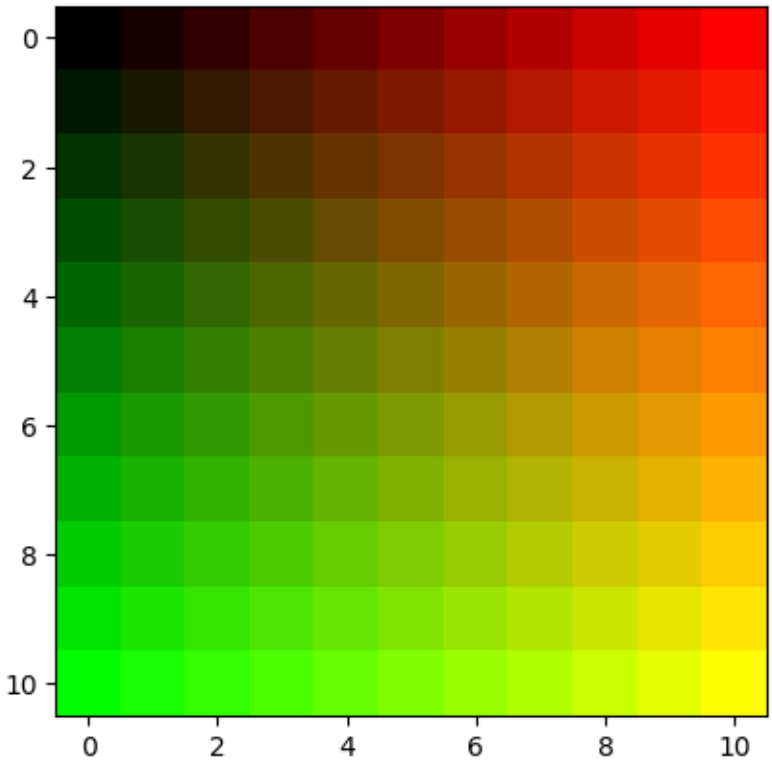
[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')

prop_cycle = plt.rcParams['axes.prop_cycle']
colors = prop_cycle.by_key()['color']

%matplotlib inline
```

1 RGB color space

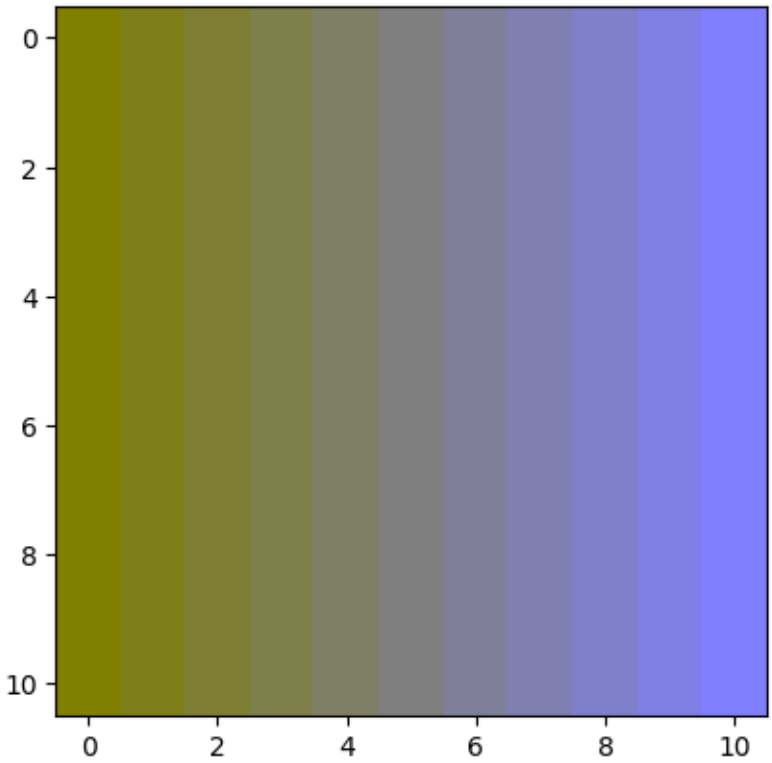
1.1 Example: simple color gradients

[2]:
```
n=11
img=np.zeros((n,n,3),dtype=np.double)
img[:,:,0]=(np.arange(n)/(n-1)).reshape((1,n))
img[:,:,1]=(np.arange(n)/(n-1)).reshape((n,1))

for i in range(n):
    for j in range(n):
        print("{:.1f}  ".format(img[i,j,1]),end="")
    print("")


plt.imshow(img)
plt.show()
```

```
0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1  0.1
0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2  0.2
0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3  0.3
0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4  0.4
0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5  0.5
0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6  0.6
0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7  0.7
0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8  0.8
0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9  0.9
1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0  1.0
```



[3]:
```
n=11
img=np.zeros((n,n,3),dtype=np.double)
img[:,:,:2]=0.5
img[:,:,2]=(np.arange(n)/(n-1)).reshape((1,n))

plt.imshow(img)
plt.show()
```



1.2 Example: load an image

[4]:
```
# image source: https://en.wikipedia.org/wiki/File:
  ↪BlueAndYellowMacaw_AraAraraunа.jpg
img=imageio.v2.imread("data/BlueAndYellowMacaw_AraAraraunа.jpg")
```

[5]:
```
img.shape
```

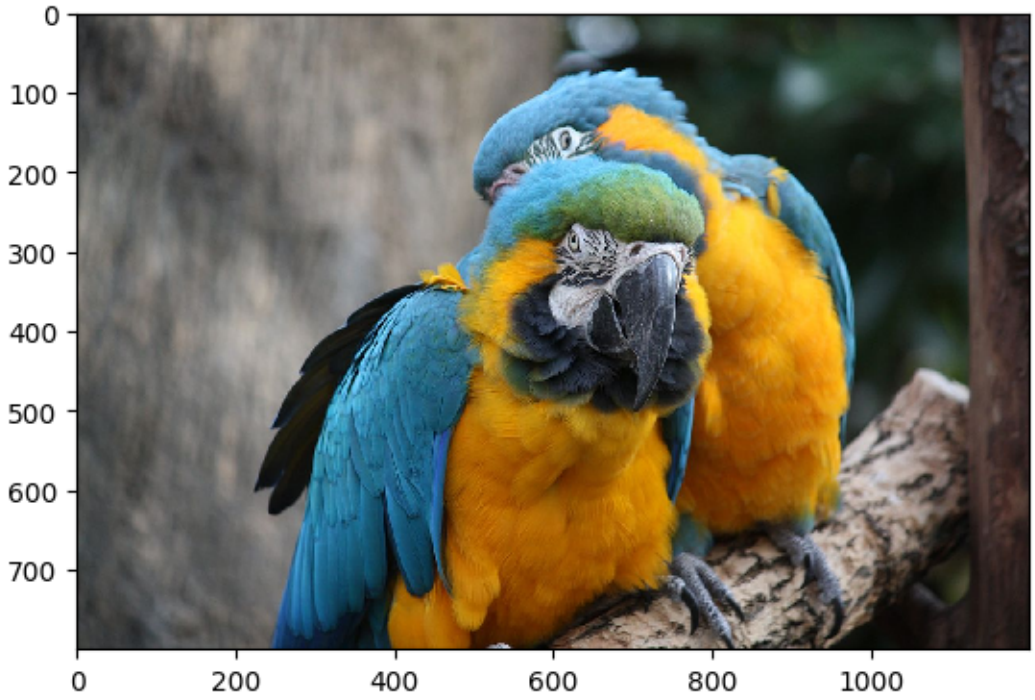
[5]: (800, 1200, 3)

[6]:
```
img.dtype
```

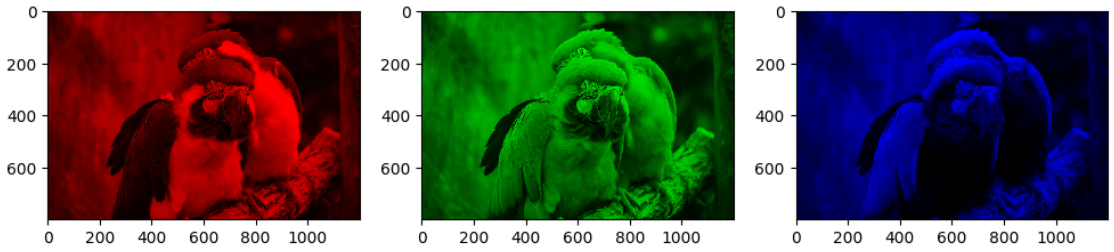
[6]: dtype('uint8')

[7]:
```
# transform to float in 0,1:
img=img/255.
```

[8]:
```
%matplotlib inline
plt.imshow(img)
plt.show()
```



[37]:
```
%matplotlib inline
# display individual RGB channels
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,3,1)
imgTest=img.copy()
imgTest[:,:,1:]=0
plt.imshow(imgTest)
fig.add_subplot(1,3,2)
imgTest=img.copy()
imgTest[:,:,0]=0
imgTest[:,:,2]=0
plt.imshow(imgTest)
fig.add_subplot(1,3,3)
imgTest=img.copy()
imgTest[:,:,:2]=0
plt.imshow(imgTest)
plt.show()
```



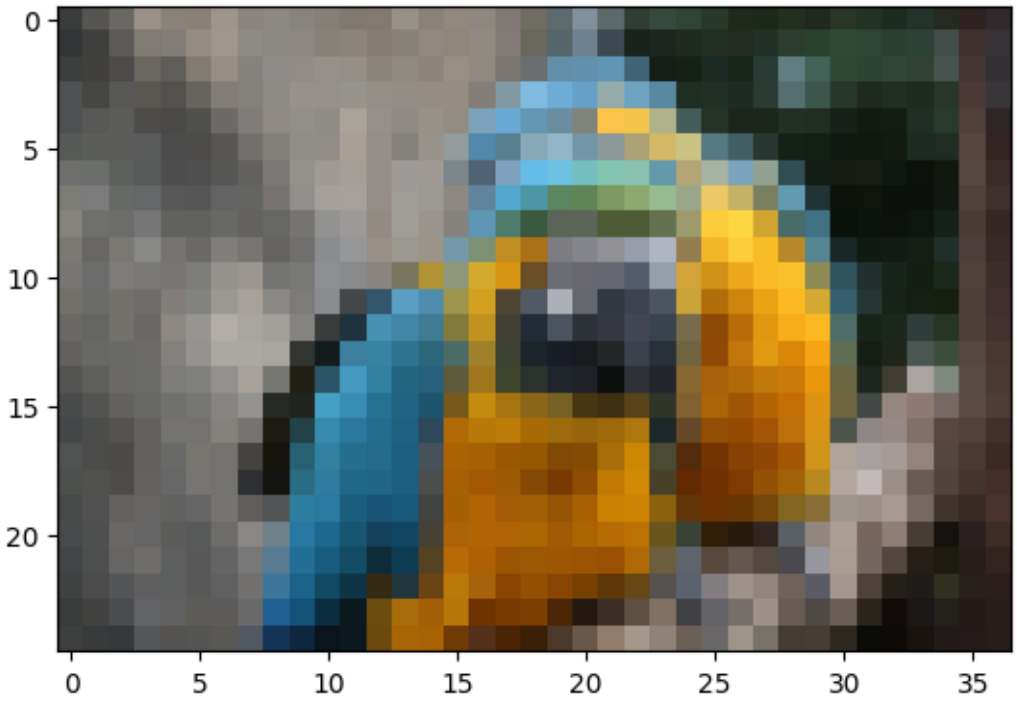
Extract color distribution

[9]:
```
# subsample image to reduce complexity a little
def subsample(dat,n):
    shape=dat.shape[:2]
    newShape=[x//n for x in shape]
    keepShape=[(x//n)*n for x in shape]
    newDat=dat[:keepShape[0],:keepShape[1]]
    interShape=(newShape[0],n,newShape[1],n)
    if dat.ndim==3:
        interShape=interShape+(dat.shape[2],)
    newDat=newDat.reshape(interShape)
    newDat=np.sum(newDat,axis=(1,3))/(n**2)
    return newDat

imgSmall=subsample(img,32)
```

[39]:
```
imgSmall.shape
```

[39]:
```
(25, 37, 3)
```

[40]:
```
plt.imshow(imgSmall)
plt.show()
```

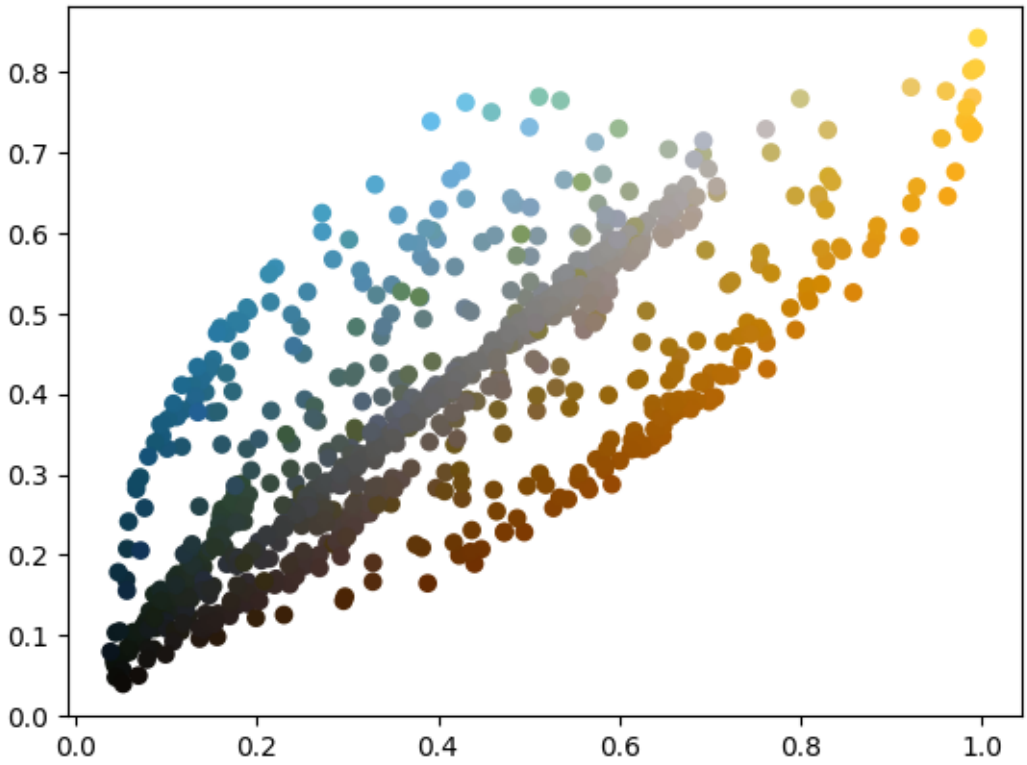


[43]:
```
pts=imgSmall.reshape((-1,3))
```

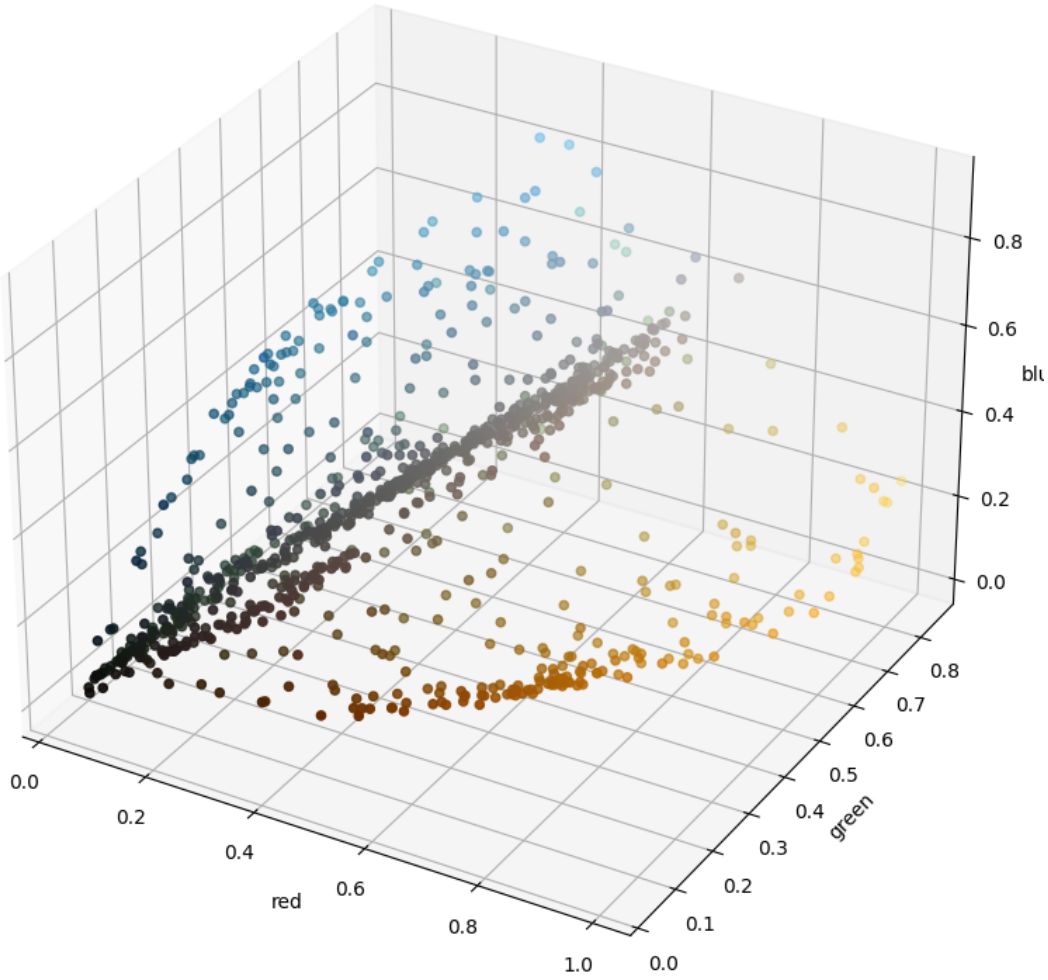
[44]:
```
pts.shape
```

[44]:
```
(925, 3)
```

[46]:
```
%matplotlib inline
plt.scatter(pts[:,0],pts[:,1],c=pts)
plt.show()
```



[47]:
```
%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(pts[:,0],pts[:,1],pts[:,2],c=pts)
ax.set_xlabel("red")
ax.set_ylabel("green")
ax.set_zlabel("blue")
plt.tight_layout()
plt.show()
```



[48]:
```
plt.close()
%matplotlib inline
```

2 HSV

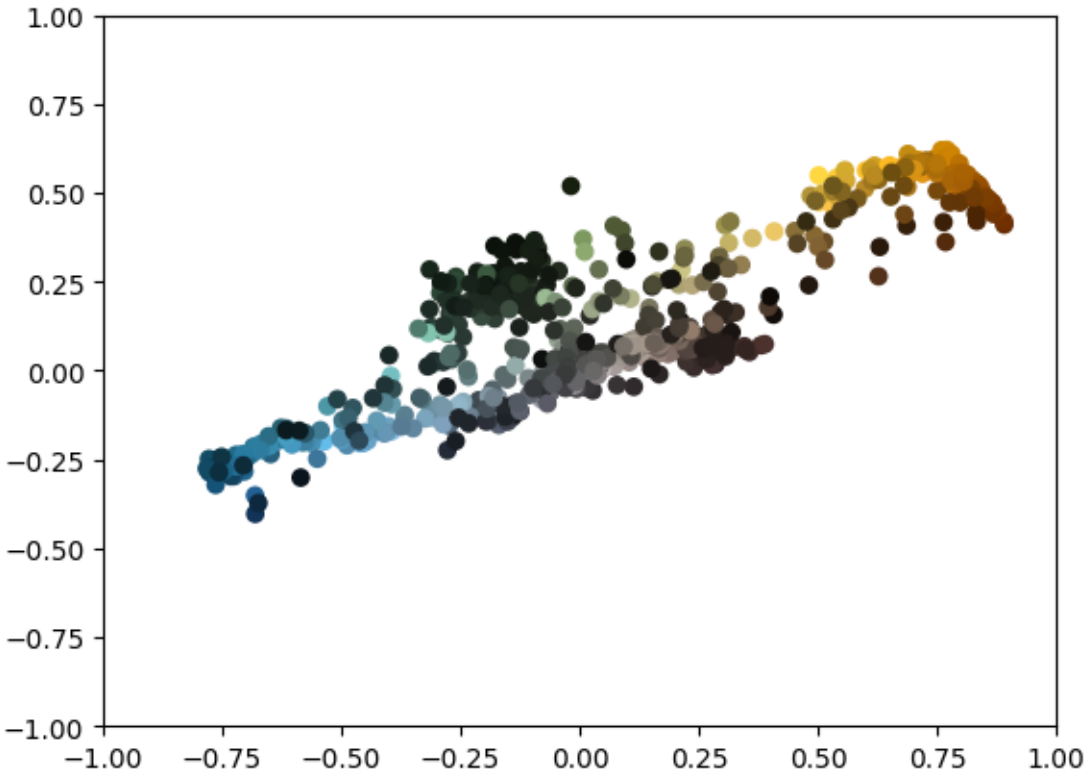
2.1 Load image and show HSV histogram

[10]:
```
# image source: https://en.wikipedia.org/wiki/File:
  ↪BlueAndYellowMacaw_AraAraraunа.jpg
img=imageio.v2.imread("data/BlueAndYellowMacaw_AraAraraunа.jpg")
img=img/255.
```

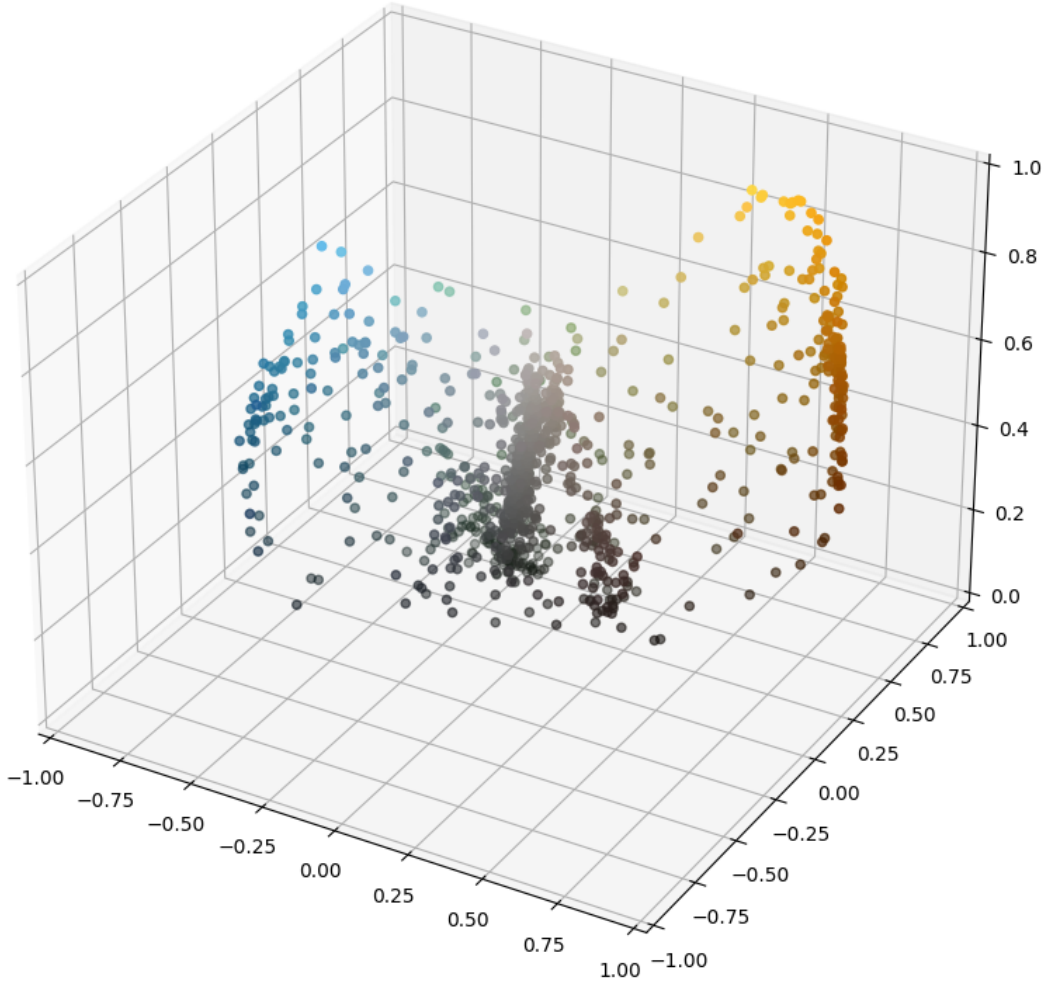
```
imgSmall=subsample(img,32)

imgHSV=matplotlib.colors.rgb_to_hsv(img)
imgSmallHSV=matplotlib.colors.rgb_to_hsv(imgSmall)
imgSmallFlat=imgSmall.reshape((-1,3))
```

[11]:
```
xres,yres=imgSmall.shape[:2]
hsvXYZ=np.zeros((xres,yres,3),dtype=np.double)
hsvXYZ[:,:,0]=imgSmallHSV[:,:,1]*np.cos(2*np.pi*imgSmallHSV[:,:,0])
hsvXYZ[:,:,1]=imgSmallHSV[:,:,1]*np.sin(2*np.pi*imgSmallHSV[:,:,0])
hsvXYZ[:,:,2]=imgSmallHSV[:,:,2]

hsvXYZ=hsvXYZ.reshape((-1,3))
```

[6]:
```
%matplotlib inline
plt.scatter(hsvXYZ[:,0],hsvXYZ[:,1],c=imgSmallFlat)
plt.xlim([-1,1])
plt.ylim([-1,1])
plt.show()
```



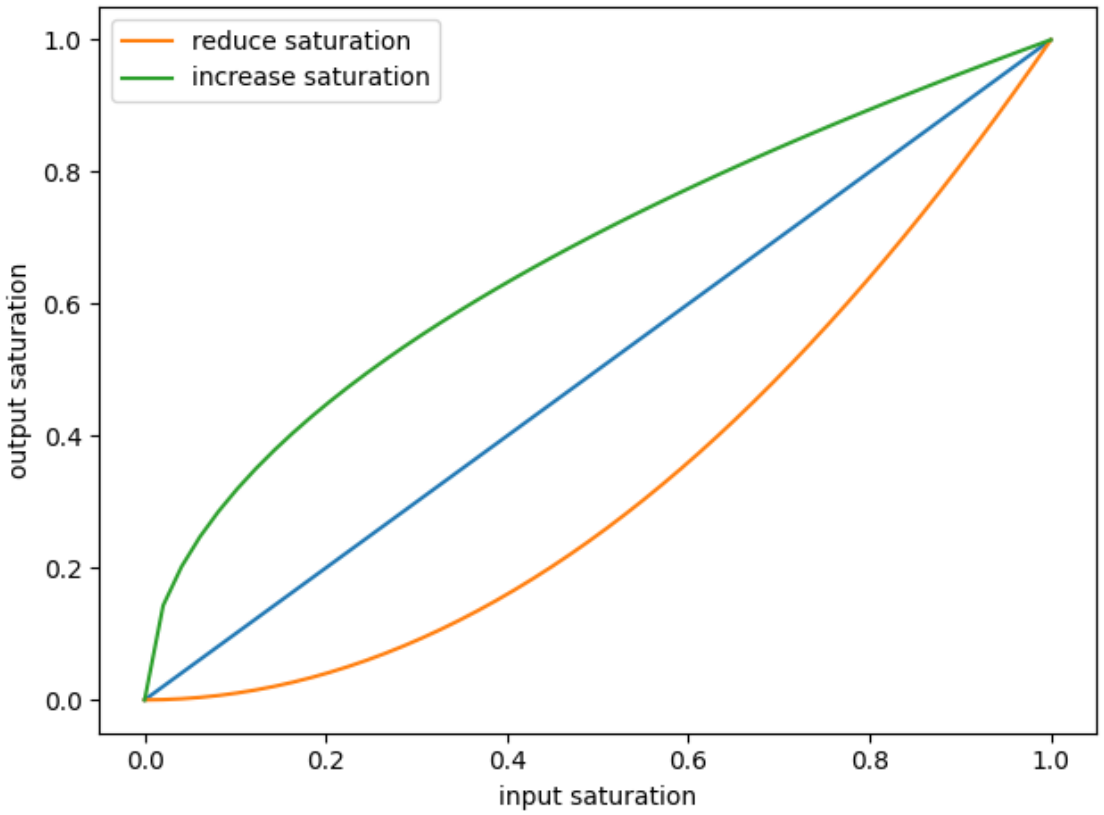
[7]:
```
%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(hsvXYZ[:,0],hsvXYZ[:,1],hsvXYZ[:,2],c=imgSmallFlat)
ax.set_xlim([-1,1])
ax.set_ylim([-1,1])
ax.set_zlim([0,1])

plt.tight_layout()
plt.show()
```



[53]:
```
plt.close()
%matplotlib inline
```

2.2 Modify saturation

[12]:
```
%matplotlib inline
xdat=np.linspace(0,1)
plt.plot(xdat,xdat)
plt.plot(xdat,xdat**2,label="reduce saturation")
plt.plot(xdat,xdat**0.5,label="increase saturation")
plt.xlabel("input saturation")
plt.ylabel("output saturation")
plt.legend()
plt.tight_layout()
plt.show()
```



[13]:
```
# image source: https://en.wikipedia.org/wiki/File:
 ↪BlueAndYellowMacaw_AraAraraunа.jpg
img=imageio.v2.imread("data/BlueAndYellowMacaw_AraAraraunа.jpg")
```

```
img=img/255.

imgHSV=matplotlib.colors.rgb_to_hsv(img)
imgHSV1=imgHSV.copy()
imgHSV2=imgHSV.copy()
imgHSV3=imgHSV.copy()

# reduce saturation
imgHSV1[:,:,1]=imgHSV1[:,:,1]**2
# increase saturation
imgHSV2[:,:,1]=imgHSV2[:,:,1]**.5

# set color to pure hue
imgHSV3[:,:,1]=1.
imgHSV3[:,:,2]=1.

img1=matplotlib.colors.hsv_to_rgb(imgHSV1)
img2=matplotlib.colors.hsv_to_rgb(imgHSV2)
img3=matplotlib.colors.hsv_to_rgb(imgHSV3)
```

[14]:
```
%matplotlib inline
# display individual RGB channels
fig=plt.figure(figsize=(16,4))
fig.add_subplot(1,4,1)
plt.imshow(img)
plt.axis("off")
fig.add_subplot(1,4,2)
plt.imshow(img1)
plt.axis("off")
fig.add_subplot(1,4,3)
plt.imshow(img2)
plt.axis("off")
fig.add_subplot(1,4,4)
plt.imshow(img3)
plt.axis("off")
plt.tight_layout()
plt.show()
```

[ ]:

2023-05-08_Example-2-ColorSchemes

May 8, 2023

[30]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')

prop_cycle = plt.rcParams['axes.prop_cycle']
colors = prop_cycle.by_key()['color']

%matplotlib inline
```

[31]:
```
def scramble(dat):
    """scramble a linear array to test visual uniformness of color schemes,
    assume dat has shape (n,) with n even
    here just mix up adjacent values"""
    result=np.zeros_like(dat)
    n=len(dat)
    result[::2]=dat[1::2]
    result[1::2]=dat[::2]
    return result



def createTestImg(dat,k,colfun):
    """create a small (k,n)-image where colfun is applied to dat along the n␣
 ↪axis"""
    n=len(dat)
    result=np.zeros((n,k),dtype=np.double)
    result[...]=dat.reshape((n,1))
    result=result.transpose()
    result=colfun(result)
```

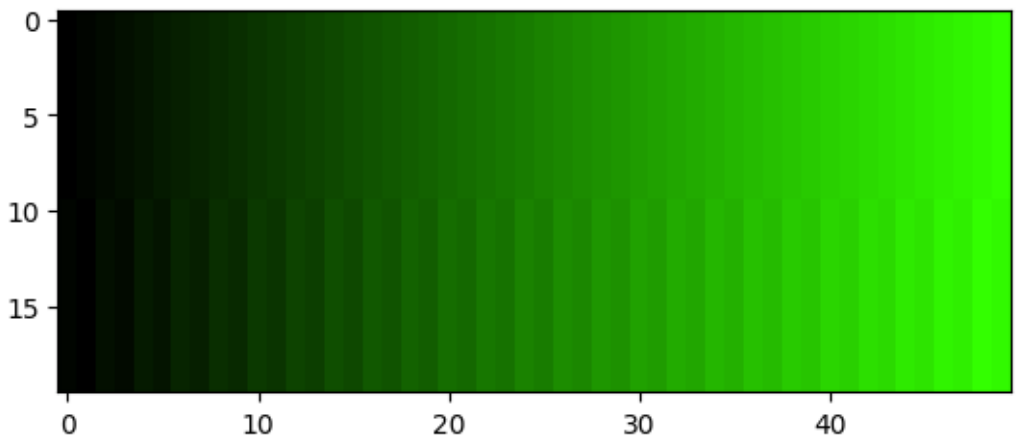
```
    return result

n=50
xrange1=np.arange(n)/(n-1)
xrange2=scramble(xrange1)

k=10

# create a pair of test images of a color scale, one with clean and one with␣
 ↪scambled sequence
# to shorten the rest of the code / signature of the function, use global␣
 ↪variables
def createTestImgPair(colfun):
    img1=createTestImg(xrange1,k,colfun)
    img2=createTestImg(xrange2,k,colfun)
    img=np.concatenate((img1,img2))
    return img
```

1 Create simple lightness variation maps

1.1 HSV: value variation for fixed hue

[32]:
```
def ex_colfun_1(dat,hue):
    """apply simple HSV-value-variation to a 2d array with values in [0,1]"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=1.
    imgHSV[:,:,2]=dat
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

[33]:
```
fun=lambda dat : ex_colfun_1(dat,.3)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```

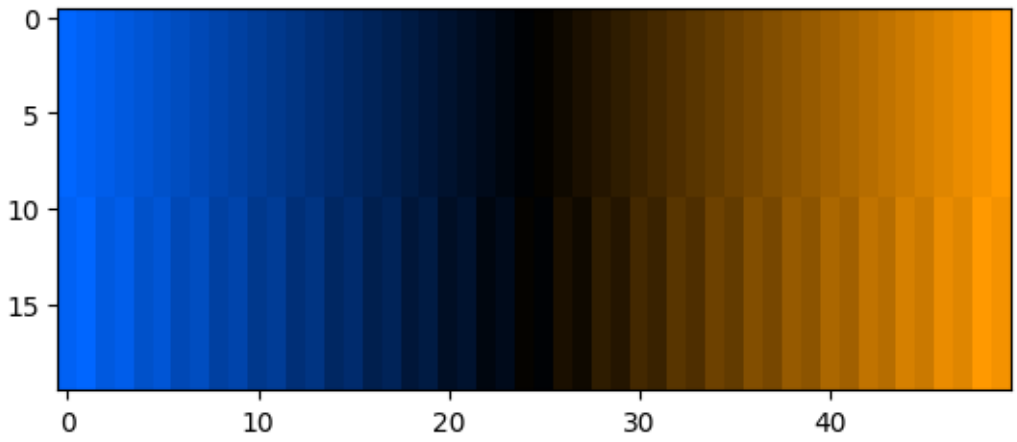


[34]:
```
# low end values are difficult to distinguish
```

1.1.1 Signed example

[35]:
```
def ex_colfun_2(dat,hue):
    """apply simple HSV-value-variation to a 2d array with values in [-1,1],␣
 ↪signed,
    negative values are mapped to opposite hue"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=1.
    imgHSV[:,:,2]=np.abs(dat)
    # now flip hue where dat is negative
    posNeg=np.where(dat<0)
    imgHSV[posNeg[0],posNeg[1],0]+=0.5
    imgHSV[:,:,0]=np.mod(imgHSV[:,:,0],1.)
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

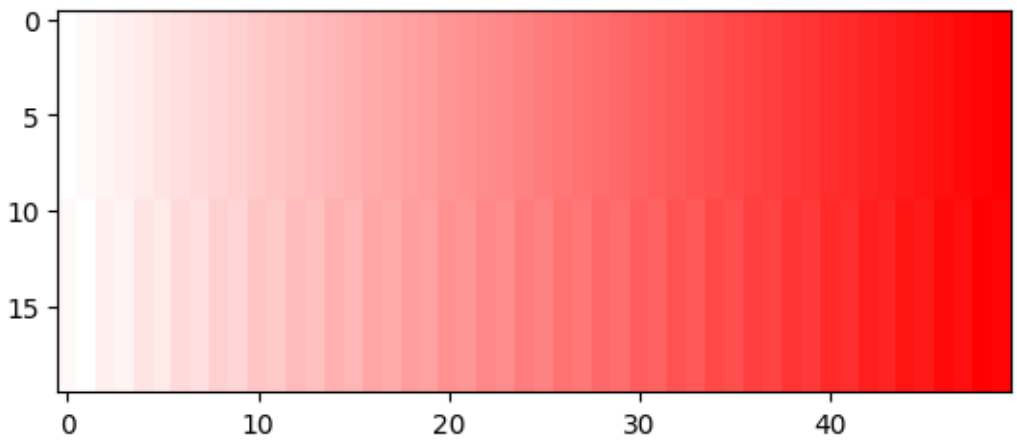
[36]:
```
fun=lambda dat : ex_colfun_2(2*dat-1,.1)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```



1.2 HSV: saturation variation for fixed hue

[37]:
```
def ex_colfun_3(dat,hue,s):
    """apply simple HSV-saturation-variation to a 2d array with values in␣
 ↪[0,1]"""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=dat
    imgHSV[:,:,2]=s+(1-s)*dat
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

[38]:
```
fun=lambda dat : ex_colfun_3(dat,.0,1.)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```

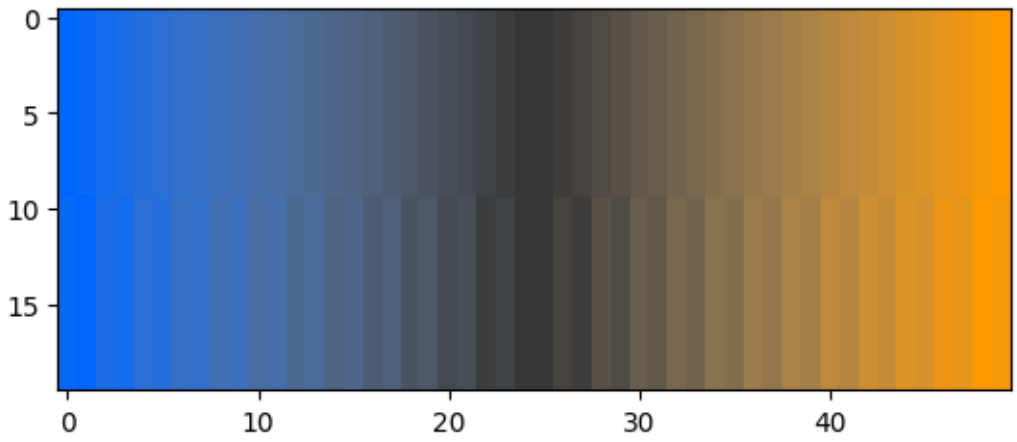


[39]:
```
# much weaker separation of steps at high end
```

1.2.1 Signed example

[40]:
```
def ex_colfun_4(dat,hue,s):
    """apply simple HSV-value/saturation-variation to a 2d array with values in␣
 ↪[-1,1], signed,
    negative values are mapped to opposite hue,
    dont go through black, but through some greyish value. regulate this with␣
 ↪param s."""
    res=dat.shape
    imgHSV=np.zeros(res+(3,),dtype=np.double)
    imgHSV[:,:,0]=hue
    imgHSV[:,:,1]=np.abs(dat)
    imgHSV[:,:,2]=s+(1-s)*np.abs(dat)
    # now flip hue where dat is negative
    posNeg=np.where(dat<0)
    imgHSV[posNeg[0],posNeg[1],0]+=0.5
    imgHSV[:,:,0]=np.mod(imgHSV[:,:,0],1.)
    img=matplotlib.colors.hsv_to_rgb(imgHSV)
    return img
```

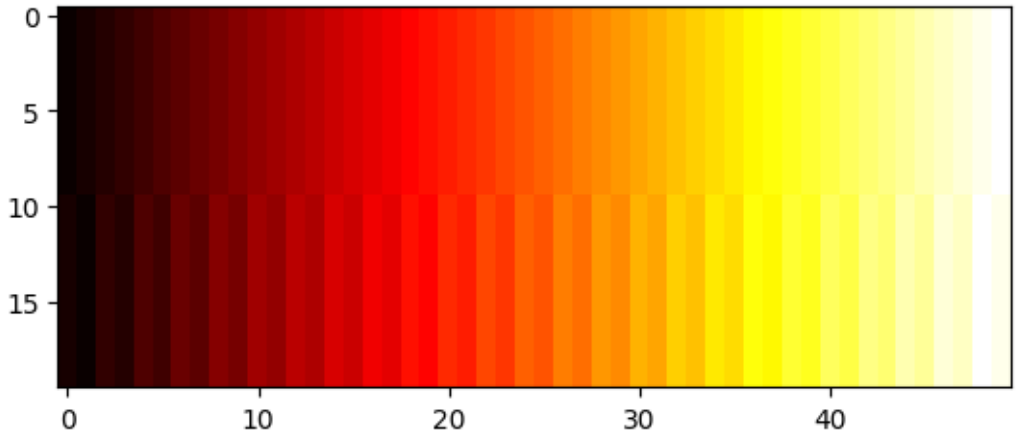
[41]:
```
fun=lambda dat : ex_colfun_4(2*dat-1,.1,0.2)
#img=createTestImg(xrange1,k,fun)
img=createTestImgPair(fun)
plt.imshow(img)
plt.show()
```



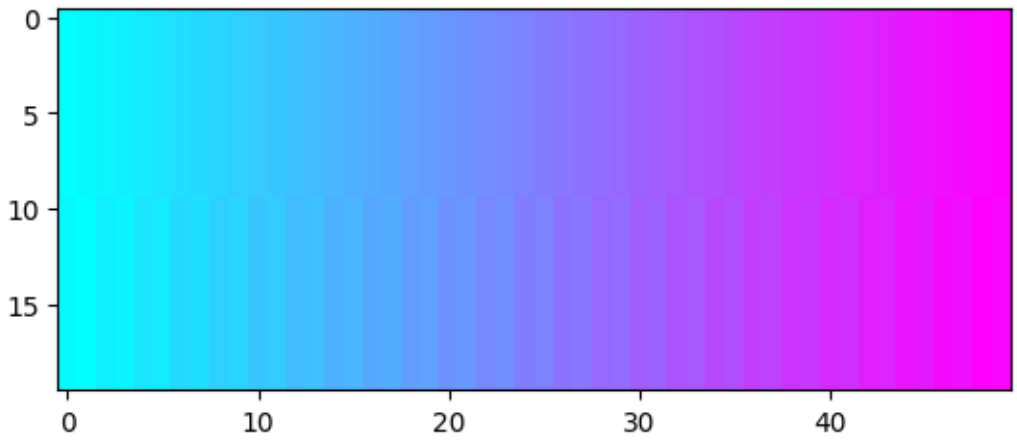
[42]:
```
# this one may look a bit less harsh at zero, but it has perceptually weaker␣
↪transitions
```

2 Perceptual uniformity

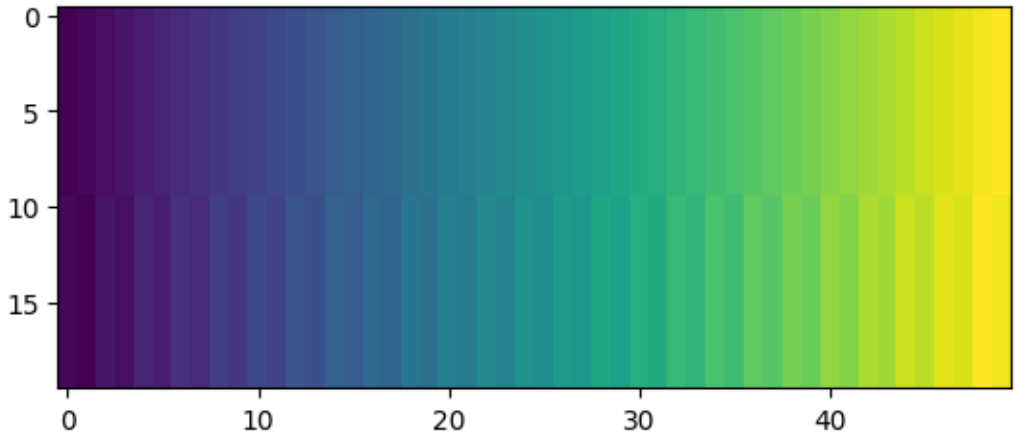
[43]:
```
img=createTestImgPair(cm.hot)
plt.imshow(img)
plt.show()
```



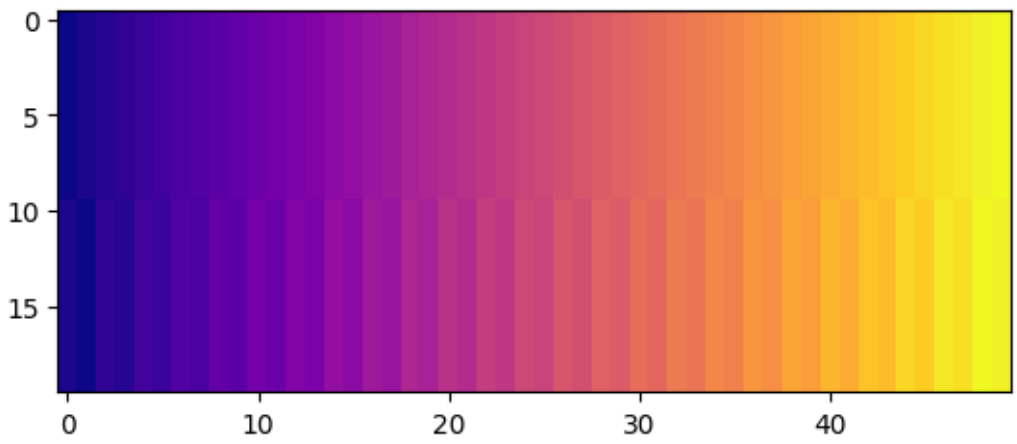
[44]:
```
img=createTestImgPair(cm.cool)
plt.imshow(img)
plt.show()
```



[45]:
```
img=createTestImgPair(cm.viridis)
plt.imshow(img)
plt.show()
```

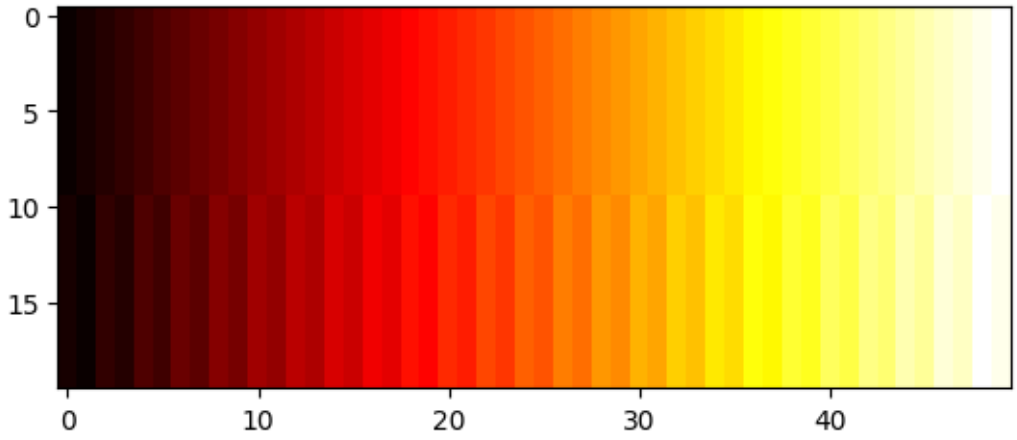


[46]:
```
img=createTestImgPair(cm.plasma)
plt.imshow(img)
plt.show()
```

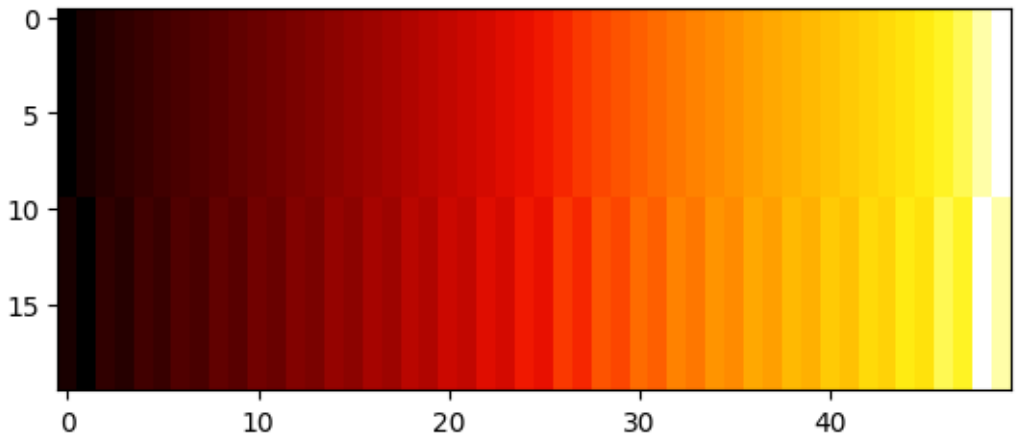


[47]:
```
import colorcet
```

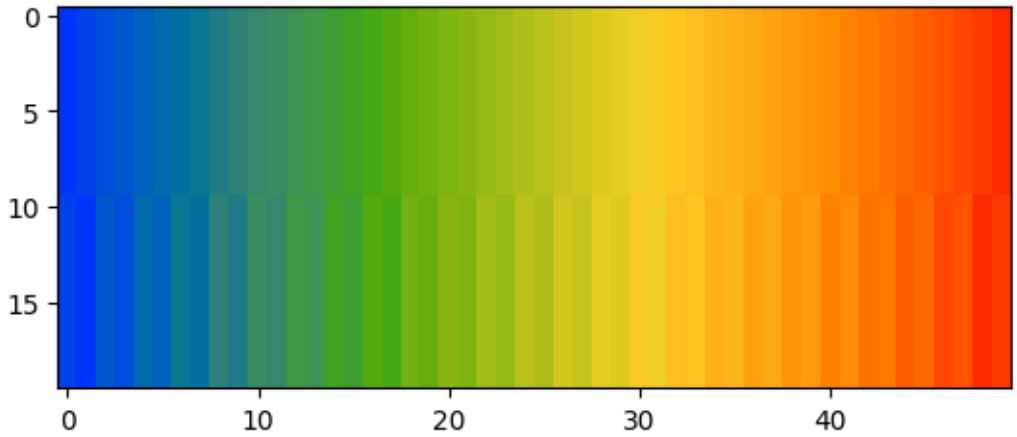
[48]:
```
img=createTestImgPair(cm.hot)
plt.imshow(img)
plt.show()
```



[49]:
```
img=createTestImgPair(colorcet.cm.fire)
plt.imshow(img)
plt.show()
```



[50]:
```
img=createTestImgPair(colorcet.cm.rainbow)
plt.imshow(img)
plt.show()
```

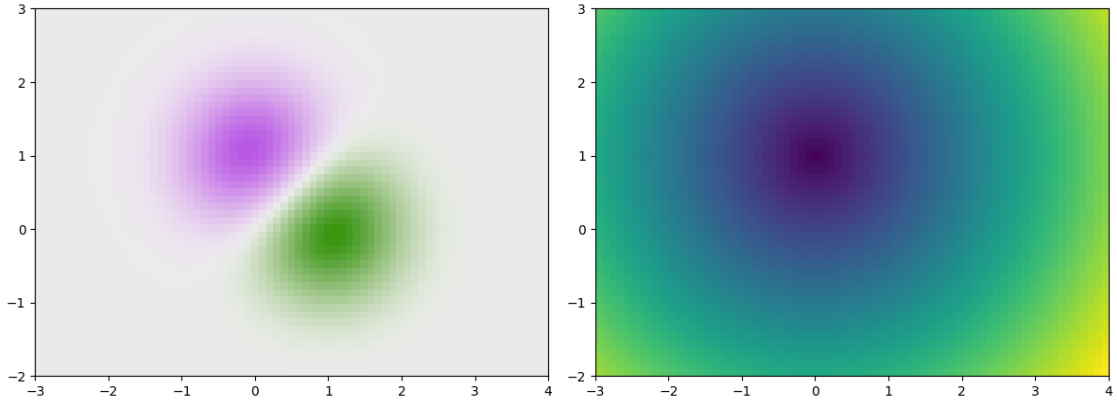


3 Improve perception

[51]:
```
# create test data
# taken from https://matplotlib.org/stable/gallery/images_contours_and_fields/
  ↪contour_image.html
extent = (-3, 4, -2, 3)
delta=0.1
x = np.arange(-3.0, 4.001, delta)
y = np.arange(-2.0, 3.001, delta)
```

```
X, Y = np.meshgrid(x, y)
Z1 = np.exp(-X**2 - Y**2)
Z2 = np.exp(-(X - 1)**2 - (Y - 1)**2)
Z = (Z1 - Z2) * 2

ZB=(X**2+Y**2)**0.5

# crude normalization to [-1,1] (but still in [0,1]):
vmax=np.max(np.abs(Z))
img=(Z+vmax)/(2*vmax)

# crude normalization to [0,1]:
vmax=np.max(ZB)
vmin=np.min(ZB)
imgB=(ZB-vmin)/(vmax-vmin)
```

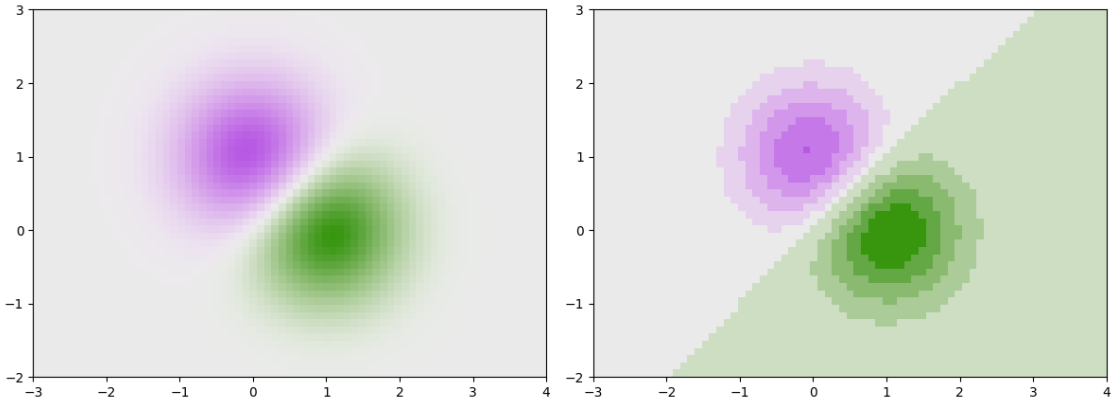
[52]:
```
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(colorcet.cm.gwv(img),extent=extent)
#plt.imshow(cm.viridis(img),extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(imgB,extent=extent)
plt.tight_layout()
plt.show()
```

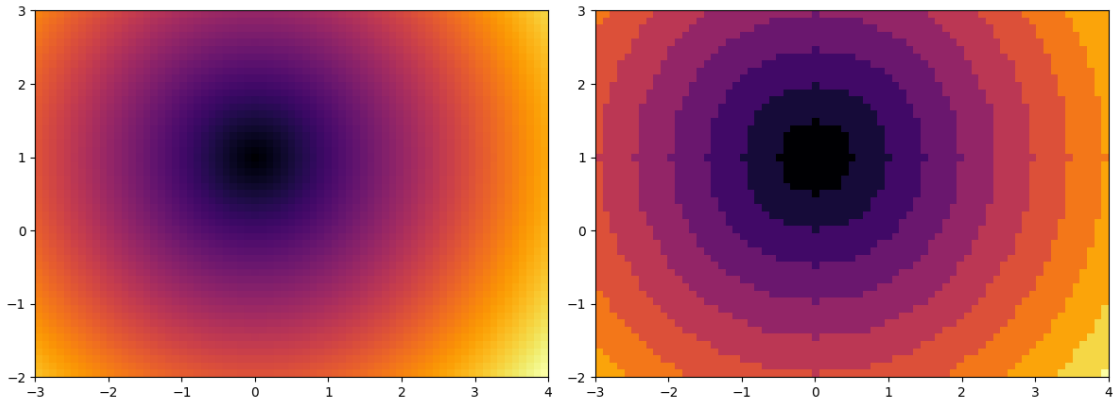


3.1 Discretize values into bins

[53]:
```
colfun=colorcet.cm.gwv
colfunB=cm.inferno
scale=10
```

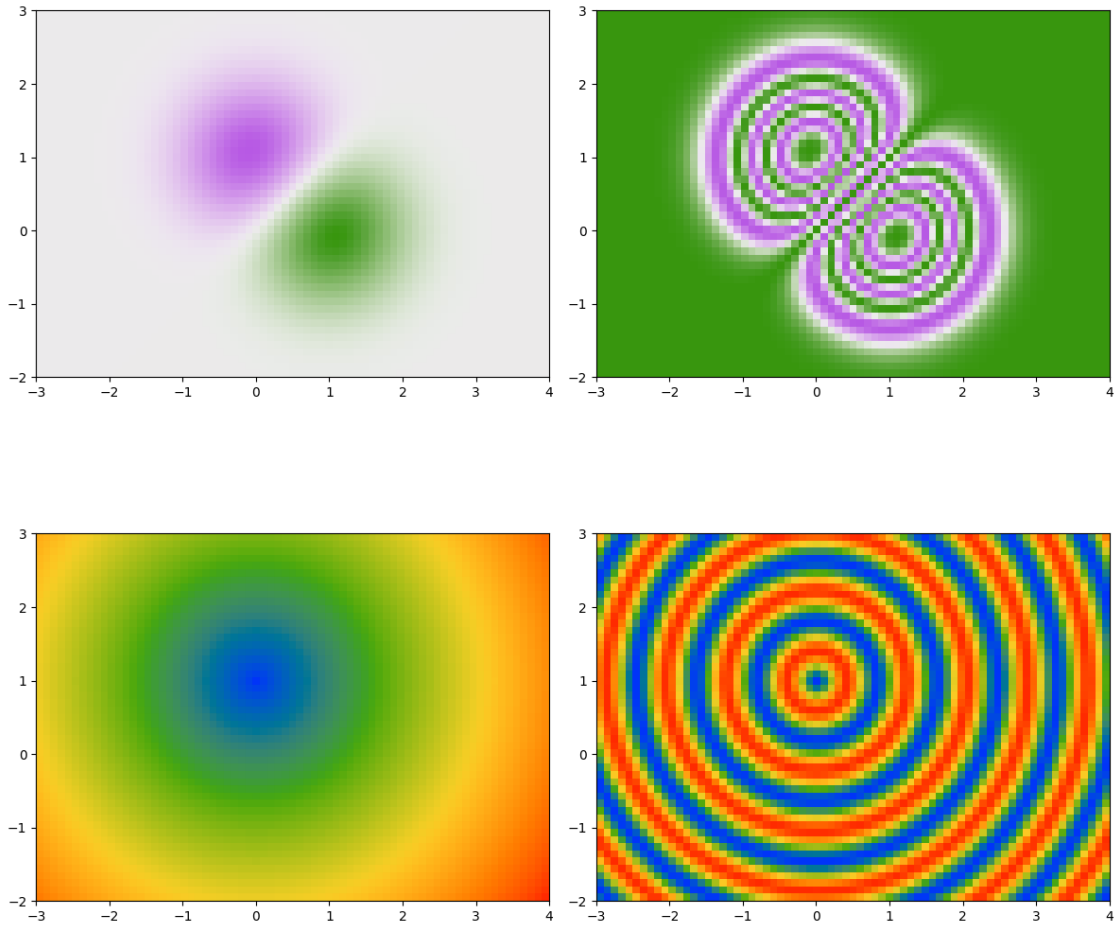
```
img1=colfun(img)
img2=colfun(np.trunc(img*scale)/scale)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()

img1=colfunB(imgB)
img2=colfunB(np.trunc(imgB*scale)/scale)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()
```





3.2 High-frequency periodic color scale

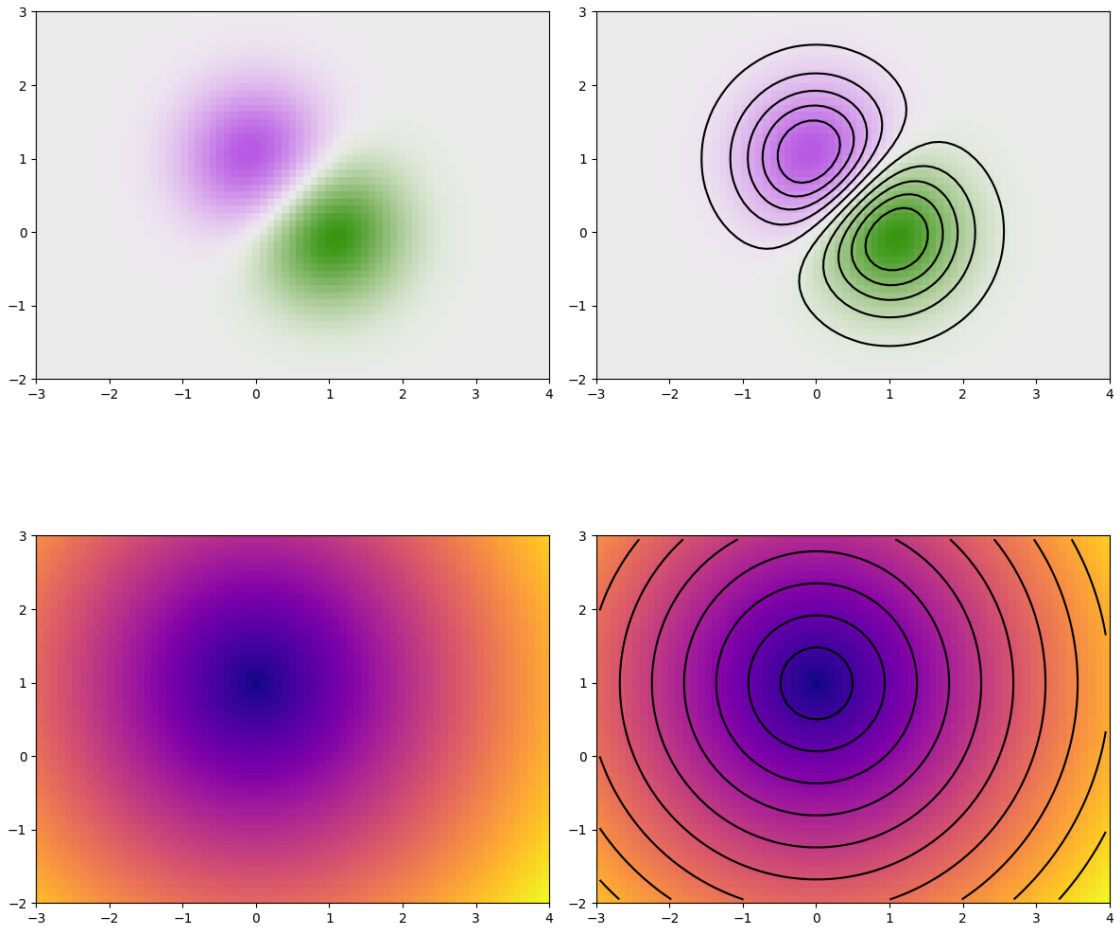
[55]:
```
colfun=colorcet.cm.gwv
colfunB=colorcet.cm.rainbow

img1=colfun(img)
img2=colfun(np.sin(img*2*np.pi*3)**2)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()

img1=colfunB(imgB)
img2=colfunB(np.sin(imgB*2*np.pi*3)**2)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img2,extent=extent)

plt.tight_layout()
plt.show()
```



3.3 Add contours

[56]:
```
colfun=colorcet.cm.gwv
colfunB=cm.plasma
levels=np.linspace(0.1,0.9,num=10)

img1=colfun(img)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img1,extent=extent)
ax=plt.gca()
ax.contour(img, levels, colors='k', origin='image', extent=extent)
plt.tight_layout()
plt.show()
```

```
img1=colfunB(imgB)
fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(img1,extent=extent)
fig.add_subplot(1,2,2)
plt.imshow(img1,extent=extent)
ax=plt.gca()
ax.contour(imgB, levels, colors='k', origin='image', extent=extent)

plt.tight_layout()
plt.show()
```

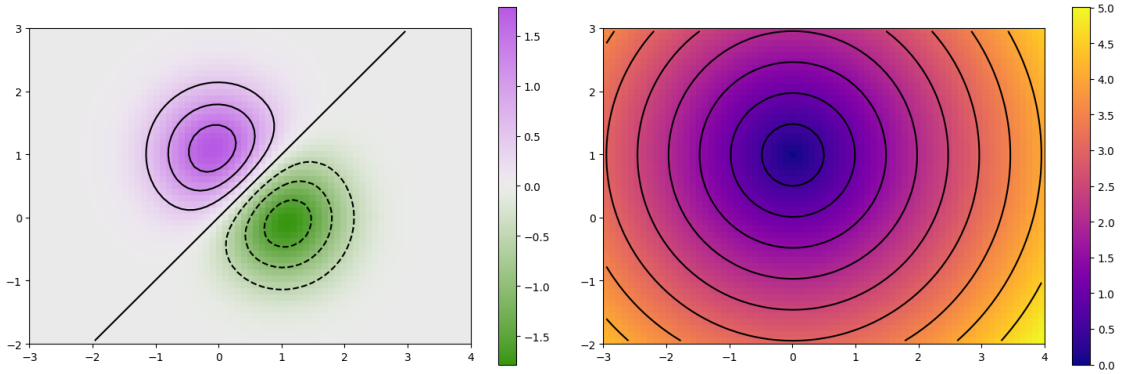


3.4 Plot with proper color bar legend

manual normalization: (what is done above) * unsigned data: scale into [0,1] from minimal to maximal value (or from zero to max) * signed data: scale first into [-1,1] from -max(abs(...)) to +max(abs(...)), then scale this into [0,1]

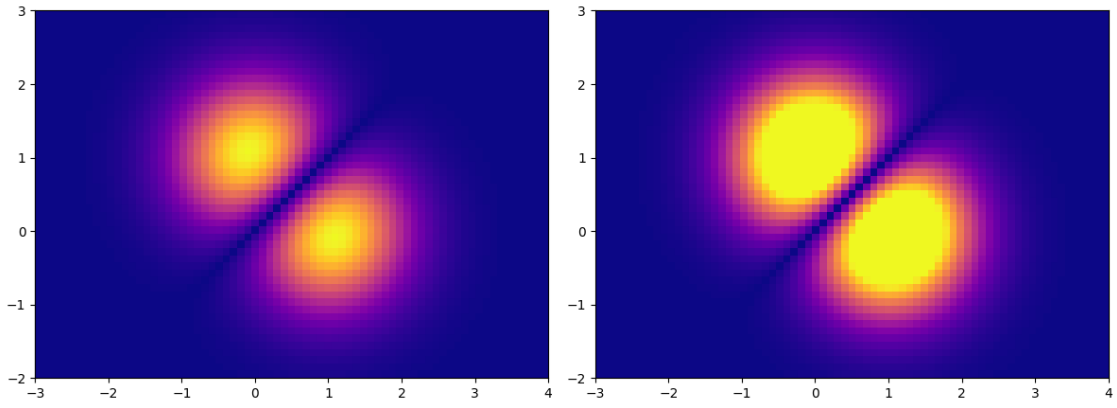
using the matplotlib normalization object:

[27]:
```
# normalization for signed data
norm = cm.colors.Normalize(vmax=abs(Z).max(), vmin=-abs(Z).max())
# normalization for unsigned data
normB = cm.colors.Normalize(vmax=abs(ZB).max(), vmin=0)

colfun=colorcet.cm.gwv
colfunB=cm.plasma
# now choose levels as absolute values, use them for contour plot and ticks in␣
 ↪colorbar
levelsA=np.linspace(-1.5,1.5,num=7)
levelsB=np.linspace(0,5,num=11)

fig=plt.figure(figsize=(15,5))
fig.add_subplot(1,2,1)
implt=plt.imshow(Z,extent=extent,cmap=colfun,norm=norm)
ax=plt.gca()
ax.contour(Z, levelsA, colors='k', origin='image', extent=extent)
fig.colorbar(implt, ax=ax,ticks=levelsA)

fig.add_subplot(1,2,2)
implt=plt.imshow(ZB,extent=extent,cmap=colfunB,norm=normB)
ax=plt.gca()
ax.contour(ZB, levelsB, colors='k', origin='image', extent=extent)
fig.colorbar(implt, ax=ax,ticks=levelsB)

plt.tight_layout()
plt.show()
```



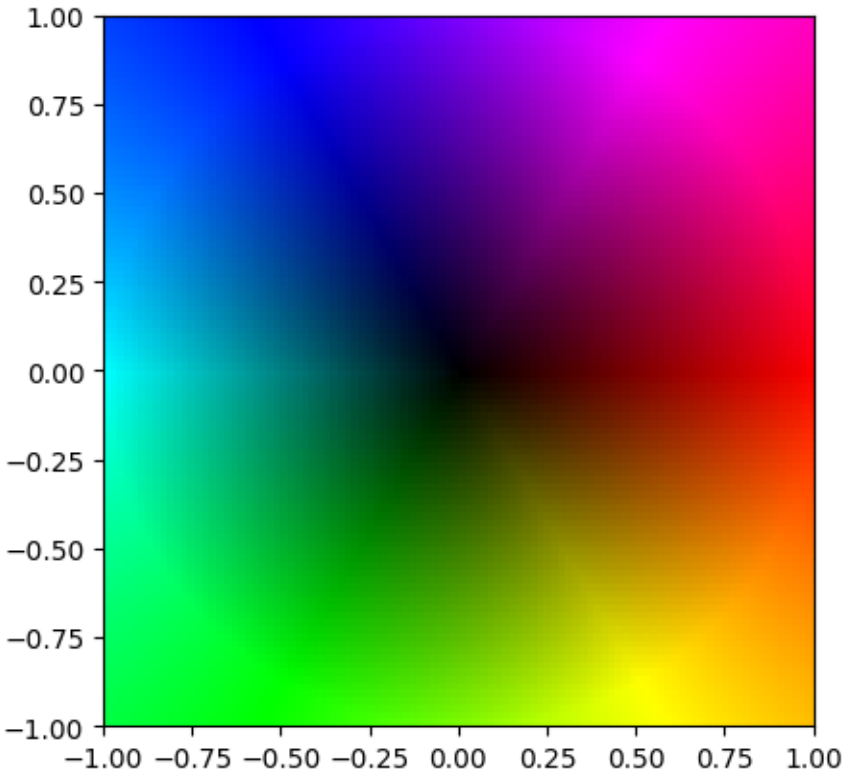
3.5 Choosing the value range

[28]:
```
colfun=cm.plasma

# for simplicity: work on unsigned data, use abs of first example
ZC=np.abs(Z)
vmax=np.max(ZC)

normC=cm.colors.Normalize(vmax=vmax, vmin=0)
if False:
    ## more focus on high end
    normC1=cm.colors.Normalize(vmax=vmax, vmin=0.8*vmax)
if True:
    ## more focus on small values
    normC1=cm.colors.Normalize(vmax=0.6*vmax,vmin=0)
if False:
    ## cut off small and/or large values
    s=0.2
    normC1=cm.colors.Normalize(vmax=(1-s)*vmax, vmin=s*vmax)



fig=plt.figure(figsize=(12,6))
fig.add_subplot(1,2,1)
plt.imshow(ZC,extent=extent,cmap=colfun,norm=normC)

fig.add_subplot(1,2,2)
plt.imshow(ZC,extent=extent,cmap=colfun,norm=normC1)

plt.tight_layout()
plt.show()
```



4 Using HSV for encoding a simple map

[29]:
```
extent = (-1, 1, -1, 1)
delta=0.02
x = np.arange(extent[0], extent[1], delta)
y = np.arange(extent[2], extent[3], delta)
X, Y = np.meshgrid(x, y)

# translate X and Y to polar coordinates
# radius
Rad=(X**2+Y**2)**0.5
# angle (arctan2 returns values in [-pi,pi]. transform this to [0,2*pi] via mod)
Phi=np.mod(np.arctan2(Y,X),2*np.pi)

# allocate free space for an HSV image
imgHSV=np.zeros(X.shape+(3,),dtype=np.double)
# set hue to Phi (re-scale by 2*pi)
imgHSV[:,:,0]=Phi/(2*np.pi)
# saturation to full value
imgHSV[:,:,1]=1.
# value given by radius
# scale radius=1 to maximal value=1, replace all larger radii by 1
maxRad=1.
imgHSV[:,:,2]=np.minimum(Rad/maxRad,1.)

# transform to RGB
img=matplotlib.colors.hsv_to_rgb(imgHSV)

# show image
plt.imshow(img,extent=extent)
plt.show()
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

[ ]:

2023-05-12_BasicDataTransformations

May 15, 2023

[18]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

%matplotlib inline
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
```

1 Basic transformations on tabular data

As example data use UN World Population Example Dataset (see other files)

1.1 Import data

[19]:
```
# set up column data types
headerLine="SortOrder,LocID,Notes,ISO3_code,ISO2_code,SDMX_code,LocTypeID,LocTypeName,"+\
    "ParentID,Location,VarID,Variant,"+\
  ␣
 ↪"Time,TPopulation1Jan,TPopulation1July,TPopulationMale1July,TPopulationFemale1July,"+\
  ␣
 ↪"PopDensity,PopSexRatio,MedianAgePop,NatChange,NatChangeRT,PopChange,PopGrowthRate,"+\
  ␣
 ↪"DoublingTime,Births,Births1519,CBR,TFR,NRR,MAC,SRB,Deaths,DeathsMale,DeathsFemale,"+\
  ␣
 ↪"CDR,LEx,LExMale,LExFemale,LE15,LE15Male,LE15Female,LE65,LE65Male,LE65Female,"+\
    "LE80,LE80Male,LE80Female,InfantDeaths,IMR,LBsurvivingAge1,Under5Deaths,"+\
  ␣
 ↪"Q5,Q0040,Q0040Male,Q0040Female,Q0060,Q0060Male,Q0060Female,Q1550,Q1550Male,"+\
    "Q1550Female,Q1560,Q1560Male,Q1560Female,NetMigrations,CNMR"

dtypeDict={}
for x in headerLine.split(","):
    dtypeDict[x]=np.float64
```

```
for x in "SortOrder,LocID,LocTypeID,ParentID,VarID,Time".split(","):
    dtypeDict[x]=np.int32
for x in "Notes,ISO3_code,SDMX_code,ISO2_code,LocTypeName,Location,Variant".
 ↪split(","):
    dtypeDict[x]=str
```

[20]:
```
dataFull=pd.read_csv("data/WPP2022_Demographic_Indicators_Medium.
 ↪csv",sep=",",dtype=dtypeDict)
```

[21]:
```
# check list of columns
dataFull.keys()
```

[21]:
```
Index(['SortOrder', 'LocID', 'Notes', 'ISO3_code', 'ISO2_code', 'SDMX_code',
       'LocTypeID', 'LocTypeName', 'ParentID', 'Location', 'VarID', 'Variant',
       'Time', 'TPopulation1Jan', 'TPopulation1July', 'TPopulationMale1July',
       'TPopulationFemale1July', 'PopDensity', 'PopSexRatio', 'MedianAgePop',
       'NatChange', 'NatChangeRT', 'PopChange', 'PopGrowthRate',
       'DoublingTime', 'Births', 'Births1519', 'CBR', 'TFR', 'NRR', 'MAC',
       'SRB', 'Deaths', 'DeathsMale', 'DeathsFemale', 'CDR', 'LEx', 'LExMale',
       'LExFemale', 'LE15', 'LE15Male', 'LE15Female', 'LE65', 'LE65Male',
       'LE65Female', 'LE80', 'LE80Male', 'LE80Female', 'InfantDeaths', 'IMR',
       'LBsurvivingAge1', 'Under5Deaths', 'Q5', 'Q0040', 'Q0040Male',
       'Q0040Female', 'Q0060', 'Q0060Male', 'Q0060Female', 'Q1550',
       'Q1550Male', 'Q1550Female', 'Q1560', 'Q1560Male', 'Q1560Female',
       'NetMigrations', 'CNMR'],
      dtype='object')
```

1.2 Transformations

1.2.1 Filtering rows

[22]:
```
# keep only rows for locations that are countries
print(f"rows full data: {dataFull.shape[0]}")

# creating boolean indicators according which to filter
countryIndicator=(dataFull["LocTypeName"]=="Country/Area")
timeIndicator=(dataFull["Time"]<=2022)

# apply logical operation and apply filter (apply copy to avoid later issues)
data=dataFull[countryIndicator & timeIndicator]

# apply copy to avoid later issues (will be illustrated below)
# clear original full dataset from memory
data=data.copy()
del dataFull
```

```
print(f"remaining columns: {data.shape[0]}")
```

```
rows full data: 43472
remaining columns: 17301
```

1.2.2 Select colums

[23]:
```
dataReduced=data[["CBR","CDR"]]
dataReduced.columns
```

[23]:
```
Index(['CBR', 'CDR'], dtype='object')
```

1.2.3 Select rows and columns

(See pandas documentation for more details about accessors.)

[24]:
```
dataSel=data.loc[data["Location"]=="Germany",["Time","CBR"]]
dataSel
```

[24]:

	Time	CBR
29184	1950	16.222
29185	1951	15.916
29186	1952	15.733
29187	1953	15.643
29188	1954	15.658
…	…	…
29252	2018	9.423
29253	2019	9.304
29254	2020	9.144
29255	2021	9.167
29256	2022	9.139

[73 rows x 2 columns]

1.2.4 Careful when writing on selections of dataframe

[25]:
```
# try this first:
# select rows that are to be changed
dataSel=data[data["Location"]=="United States of America"]
# try to write on them
dataSel["Location"]="USA"
```

```
/tmp/ipykernel_6449/3824782913.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
```

```
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dataSel["Location"]="USA"
```

[26]:
```
# but in the original dataframe the entries have not been changed
data[data["Location"]=="United States of America"].shape
```

[26]:
```
(73, 67)
```

[27]:
```
# preferred way to handle this:
data.loc[data["Location"]=="United States of America","Location"]="USA"
```

[28]:
```
# check again
data[data["Location"]=="United States of America"].shape
```

[28]:
```
(0, 67)
```

1.2.5 Generate derived columns

[29]:
```
# compute "crude change rate": crude birth rate - crude death rate
data["CCR"]=data["CBR"]-data["CDR"]
```

1.2.6 Join tables

[30]:
```
# have a separate table available, that lists for each country the␣
 ↪corresponding continent
# now want to add this info as new column into the data frame
# i.e. for each row in main table, look up row with corresponding country and␣
 ↪copy the given value of continent
# to a new continent column
dataContinents=pd.read_csv("data/UN_population/continent_association.csv",\
        dtype={"SDMX_code":str,"Continent":str})
dataContinents.dtypes
```

[30]:
```
SDMX_code    object
Continent    object
dtype: object
```

[31]:
```
dataContinents["Continent"].unique()
```

[31]:
```
array(['Africa', 'Asia', 'Europe', 'Central and South America',
       'North America', 'Oceania'], dtype=object)
```

[32]:
```
# merge with continent data
data=pd.merge(data,dataContinents,on="SDMX_code")
```

[33]:
```
data.shape
```

[33]:
```
(17301, 69)
```

1.2.7 Summaries / reductions

Reduce big dataset to smaller one by computing basic aggregations, summaries, statistics. Typical examples are: * Maximum, minimum, mean, median, variance, ... * Count (distinguish: size, count, nunique in pandas)

[35]:
```
[data["Time"].min(),data["Time"].max()]
```

[35]:
```
[1950, 2022]
```

[37]:
```
continents=data["Continent"].unique()
print(continents)
```

```
['Africa' 'Asia' 'Europe' 'Central and South America' 'North America'
 'Oceania']
```

[38]:
```
data["Continent"].nunique()
```

[38]:
```
6
```

[40]:
```
data["TPopulation1Jan"].median()
```

[40]:
```
3121.132
```

[41]:
```
data["TPopulation1Jan"].sum()
```

[41]:
```
365821161.818
```

1.2.8 Grouping

Of course: applying such simplistic summaries to the whole dataset is usually too reductive. Often want to apply them only to subsets separately. This is most conveniently achieved by grouping a dataframe.

[42]:
```
# this splits a dataframe into many smaller, according to distinct values of␣
 ↪the given column
dataGrouped=data.groupby("Time")
print("year\t# rows")
for year,datasub in dataGrouped:
```

```
    print(f"{year}\t{datasub.shape[0]}")
```

```
year    # rows
1950    237
1951    237
1952    237
1953    237
1954    237
1955    237
1956    237
1957    237
1958    237
1959    237
1960    237
1961    237
1962    237
1963    237
1964    237
1965    237
1966    237
1967    237
1968    237
1969    237
1970    237
1971    237
1972    237
1973    237
1974    237
1975    237
1976    237
1977    237
1978    237
1979    237
1980    237
1981    237
1982    237
1983    237
1984    237
1985    237
1986    237
1987    237
1988    237
1989    237
1990    237
1991    237
1992    237
1993    237
```

```
1994    237
1995    237
1996    237
1997    237
1998    237
1999    237
2000    237
2001    237
2002    237
2003    237
2004    237
2005    237
2006    237
2007    237
2008    237
2009    237
2010    237
2011    237
2012    237
2013    237
2014    237
2015    237
2016    237
2017    237
2018    237
2019    237
2020    237
2021    237
2022    237
```

[43]:
```
# can group over several columns
# this splits a dataframe into many smaller, according to distinct values of␣
  ↪the given column
dataGrouped=data.groupby(["Time","Continent"])
print("year\tcont\t# rows")
for (year,continent),datasub in dataGrouped:
    print(f"{year}\t{continent[:4]}\t{datasub.shape[0]}")
```

```
year    cont    # rows
1950    Afri    58
1950    Asia    51
1950    Cent    50
1950    Euro    50
1950    Nort    5
1950    Ocea    23
1951    Afri    58
1951    Asia    51
1951    Cent    50
```

```
1951    Euro    50
1951    Nort    5
1951    Ocea    23
1952    Afri    58
1952    Asia    51
1952    Cent    50
1952    Euro    50
1952    Nort    5
1952    Ocea    23
1953    Afri    58
1953    Asia    51
1953    Cent    50
1953    Euro    50
1953    Nort    5
1953    Ocea    23
1954    Afri    58
1954    Asia    51
1954    Cent    50
1954    Euro    50
1954    Nort    5
1954    Ocea    23
1955    Afri    58
1955    Asia    51
1955    Cent    50
1955    Euro    50
1955    Nort    5
1955    Ocea    23
1956    Afri    58
1956    Asia    51
1956    Cent    50
1956    Euro    50
1956    Nort    5
1956    Ocea    23
1957    Afri    58
1957    Asia    51
1957    Cent    50
1957    Euro    50
1957    Nort    5
1957    Ocea    23
1958    Afri    58
1958    Asia    51
1958    Cent    50
1958    Euro    50
1958    Nort    5
1958    Ocea    23
1959    Afri    58
1959    Asia    51
1959    Cent    50
```

```
1959    Euro    50
1959    Nort    5
1959    Ocea    23
1960    Afri    58
1960    Asia    51
1960    Cent    50
1960    Euro    50
1960    Nort    5
1960    Ocea    23
1961    Afri    58
1961    Asia    51
1961    Cent    50
1961    Euro    50
1961    Nort    5
1961    Ocea    23
1962    Afri    58
1962    Asia    51
1962    Cent    50
1962    Euro    50
1962    Nort    5
1962    Ocea    23
1963    Afri    58
1963    Asia    51
1963    Cent    50
1963    Euro    50
1963    Nort    5
1963    Ocea    23
1964    Afri    58
1964    Asia    51
1964    Cent    50
1964    Euro    50
1964    Nort    5
1964    Ocea    23
1965    Afri    58
1965    Asia    51
1965    Cent    50
1965    Euro    50
1965    Nort    5
1965    Ocea    23
1966    Afri    58
1966    Asia    51
1966    Cent    50
1966    Euro    50
1966    Nort    5
1966    Ocea    23
1967    Afri    58
1967    Asia    51
1967    Cent    50
```

```
1967    Euro    50
1967    Nort    5
1967    Ocea    23
1968    Afri    58
1968    Asia    51
1968    Cent    50
1968    Euro    50
1968    Nort    5
1968    Ocea    23
1969    Afri    58
1969    Asia    51
1969    Cent    50
1969    Euro    50
1969    Nort    5
1969    Ocea    23
1970    Afri    58
1970    Asia    51
1970    Cent    50
1970    Euro    50
1970    Nort    5
1970    Ocea    23
1971    Afri    58
1971    Asia    51
1971    Cent    50
1971    Euro    50
1971    Nort    5
1971    Ocea    23
1972    Afri    58
1972    Asia    51
1972    Cent    50
1972    Euro    50
1972    Nort    5
1972    Ocea    23
1973    Afri    58
1973    Asia    51
1973    Cent    50
1973    Euro    50
1973    Nort    5
1973    Ocea    23
1974    Afri    58
1974    Asia    51
1974    Cent    50
1974    Euro    50
1974    Nort    5
1974    Ocea    23
1975    Afri    58
1975    Asia    51
1975    Cent    50
```

```
1975    Euro    50
1975    Nort    5
1975    Ocea    23
1976    Afri    58
1976    Asia    51
1976    Cent    50
1976    Euro    50
1976    Nort    5
1976    Ocea    23
1977    Afri    58
1977    Asia    51
1977    Cent    50
1977    Euro    50
1977    Nort    5
1977    Ocea    23
1978    Afri    58
1978    Asia    51
1978    Cent    50
1978    Euro    50
1978    Nort    5
1978    Ocea    23
1979    Afri    58
1979    Asia    51
1979    Cent    50
1979    Euro    50
1979    Nort    5
1979    Ocea    23
1980    Afri    58
1980    Asia    51
1980    Cent    50
1980    Euro    50
1980    Nort    5
1980    Ocea    23
1981    Afri    58
1981    Asia    51
1981    Cent    50
1981    Euro    50
1981    Nort    5
1981    Ocea    23
1982    Afri    58
1982    Asia    51
1982    Cent    50
1982    Euro    50
1982    Nort    5
1982    Ocea    23
1983    Afri    58
1983    Asia    51
1983    Cent    50
```

```
1983    Euro    50
1983    Nort    5
1983    Ocea    23
1984    Afri    58
1984    Asia    51
1984    Cent    50
1984    Euro    50
1984    Nort    5
1984    Ocea    23
1985    Afri    58
1985    Asia    51
1985    Cent    50
1985    Euro    50
1985    Nort    5
1985    Ocea    23
1986    Afri    58
1986    Asia    51
1986    Cent    50
1986    Euro    50
1986    Nort    5
1986    Ocea    23
1987    Afri    58
1987    Asia    51
1987    Cent    50
1987    Euro    50
1987    Nort    5
1987    Ocea    23
1988    Afri    58
1988    Asia    51
1988    Cent    50
1988    Euro    50
1988    Nort    5
1988    Ocea    23
1989    Afri    58
1989    Asia    51
1989    Cent    50
1989    Euro    50
1989    Nort    5
1989    Ocea    23
1990    Afri    58
1990    Asia    51
1990    Cent    50
1990    Euro    50
1990    Nort    5
1990    Ocea    23
1991    Afri    58
1991    Asia    51
1991    Cent    50
```

```
1991    Euro    50
1991    Nort    5
1991    Ocea    23
1992    Afri    58
1992    Asia    51
1992    Cent    50
1992    Euro    50
1992    Nort    5
1992    Ocea    23
1993    Afri    58
1993    Asia    51
1993    Cent    50
1993    Euro    50
1993    Nort    5
1993    Ocea    23
1994    Afri    58
1994    Asia    51
1994    Cent    50
1994    Euro    50
1994    Nort    5
1994    Ocea    23
1995    Afri    58
1995    Asia    51
1995    Cent    50
1995    Euro    50
1995    Nort    5
1995    Ocea    23
1996    Afri    58
1996    Asia    51
1996    Cent    50
1996    Euro    50
1996    Nort    5
1996    Ocea    23
1997    Afri    58
1997    Asia    51
1997    Cent    50
1997    Euro    50
1997    Nort    5
1997    Ocea    23
1998    Afri    58
1998    Asia    51
1998    Cent    50
1998    Euro    50
1998    Nort    5
1998    Ocea    23
1999    Afri    58
1999    Asia    51
1999    Cent    50
```

```
1999    Euro    50
1999    Nort    5
1999    Ocea    23
2000    Afri    58
2000    Asia    51
2000    Cent    50
2000    Euro    50
2000    Nort    5
2000    Ocea    23
2001    Afri    58
2001    Asia    51
2001    Cent    50
2001    Euro    50
2001    Nort    5
2001    Ocea    23
2002    Afri    58
2002    Asia    51
2002    Cent    50
2002    Euro    50
2002    Nort    5
2002    Ocea    23
2003    Afri    58
2003    Asia    51
2003    Cent    50
2003    Euro    50
2003    Nort    5
2003    Ocea    23
2004    Afri    58
2004    Asia    51
2004    Cent    50
2004    Euro    50
2004    Nort    5
2004    Ocea    23
2005    Afri    58
2005    Asia    51
2005    Cent    50
2005    Euro    50
2005    Nort    5
2005    Ocea    23
2006    Afri    58
2006    Asia    51
2006    Cent    50
2006    Euro    50
2006    Nort    5
2006    Ocea    23
2007    Afri    58
2007    Asia    51
2007    Cent    50
```

```
2007    Euro    50
2007    Nort    5
2007    Ocea    23
2008    Afri    58
2008    Asia    51
2008    Cent    50
2008    Euro    50
2008    Nort    5
2008    Ocea    23
2009    Afri    58
2009    Asia    51
2009    Cent    50
2009    Euro    50
2009    Nort    5
2009    Ocea    23
2010    Afri    58
2010    Asia    51
2010    Cent    50
2010    Euro    50
2010    Nort    5
2010    Ocea    23
2011    Afri    58
2011    Asia    51
2011    Cent    50
2011    Euro    50
2011    Nort    5
2011    Ocea    23
2012    Afri    58
2012    Asia    51
2012    Cent    50
2012    Euro    50
2012    Nort    5
2012    Ocea    23
2013    Afri    58
2013    Asia    51
2013    Cent    50
2013    Euro    50
2013    Nort    5
2013    Ocea    23
2014    Afri    58
2014    Asia    51
2014    Cent    50
2014    Euro    50
2014    Nort    5
2014    Ocea    23
2015    Afri    58
2015    Asia    51
2015    Cent    50
```

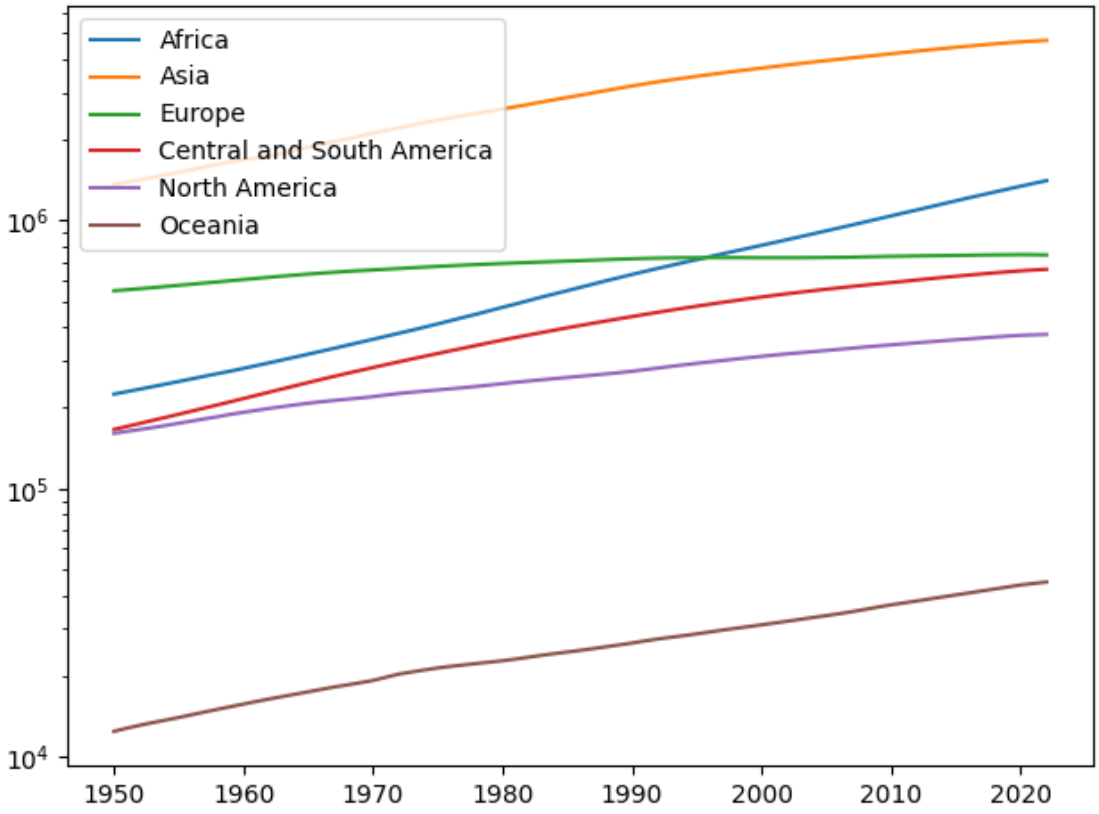
```
2015    Euro    50
2015    Nort    5
2015    Ocea    23
2016    Afri    58
2016    Asia    51
2016    Cent    50
2016    Euro    50
2016    Nort    5
2016    Ocea    23
2017    Afri    58
2017    Asia    51
2017    Cent    50
2017    Euro    50
2017    Nort    5
2017    Ocea    23
2018    Afri    58
2018    Asia    51
2018    Cent    50
2018    Euro    50
2018    Nort    5
2018    Ocea    23
2019    Afri    58
2019    Asia    51
2019    Cent    50
2019    Euro    50
2019    Nort    5
2019    Ocea    23
2020    Afri    58
2020    Asia    51
2020    Cent    50
2020    Euro    50
2020    Nort    5
2020    Ocea    23
2021    Afri    58
2021    Asia    51
2021    Cent    50
2021    Euro    50
2021    Nort    5
2021    Ocea    23
2022    Afri    58
2022    Asia    51
2022    Cent    50
2022    Euro    50
2022    Nort    5
2022    Ocea    23
```

[66]:
```
# now combine this with stats
```

[44]:
```
dataGrouped=data.groupby(["Continent","Time"])
dataSummed=dataGrouped["TPopulation1Jan"].sum()
print(dataSummed)
```

```
Continent  Time
Africa     1950    225120.311
           1951    229978.205
           1952    234989.784
           1953    240182.336
           1954    245492.559
                      …
Oceania    2018     42175.314
           2019     42904.041
           2020     43652.259
           2021     44214.592
           2022     44768.856
Name: TPopulation1Jan, Length: 438, dtype: float64
```

[45]:
```
for cont in continents:
    plt.plot(dataSummed[cont].index,dataSummed[cont].values,label=cont)
plt.yscale("log")
plt.legend()
plt.tight_layout()
plt.show()
```


Africa
Asia
Europe
Central and South America
North America
Oceania
10^6
10^5
10^4
1950
1960
1970
1980
1990
2000
2010
2020

[ ]:

[ ]:

2023-05-15_ChartTypes_001_Bar

May 15, 2023

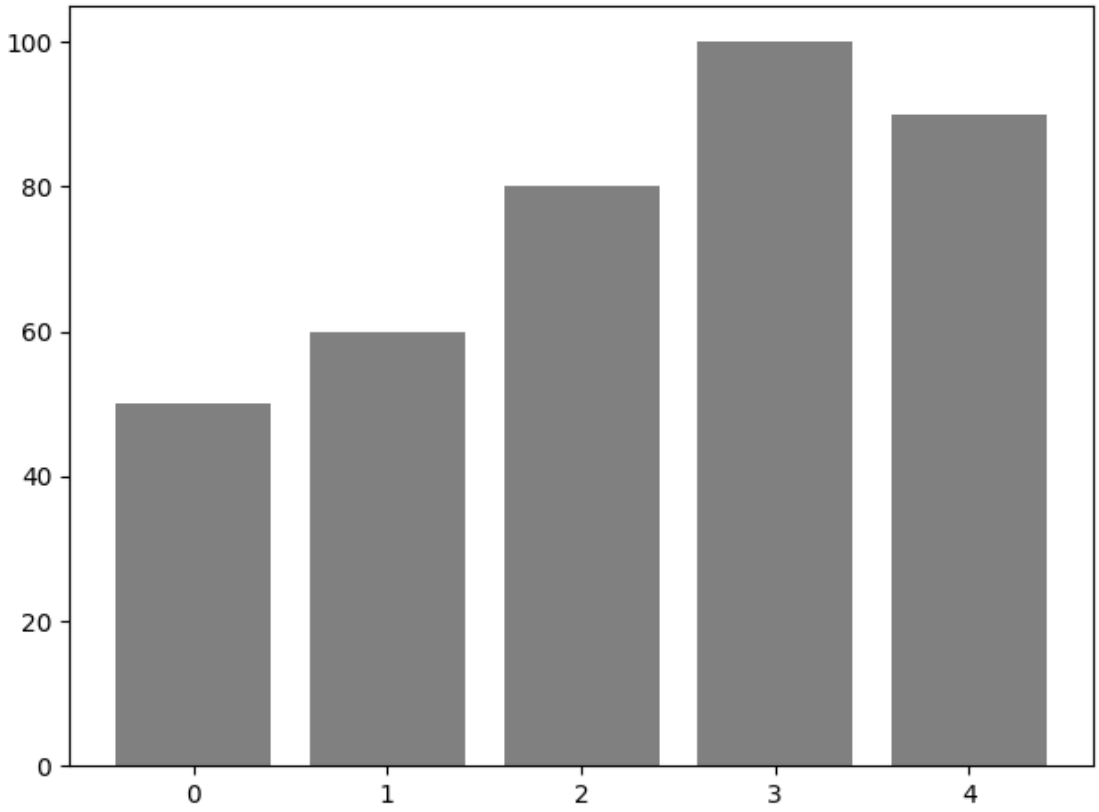
[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Bar charts

1.1 Basic version

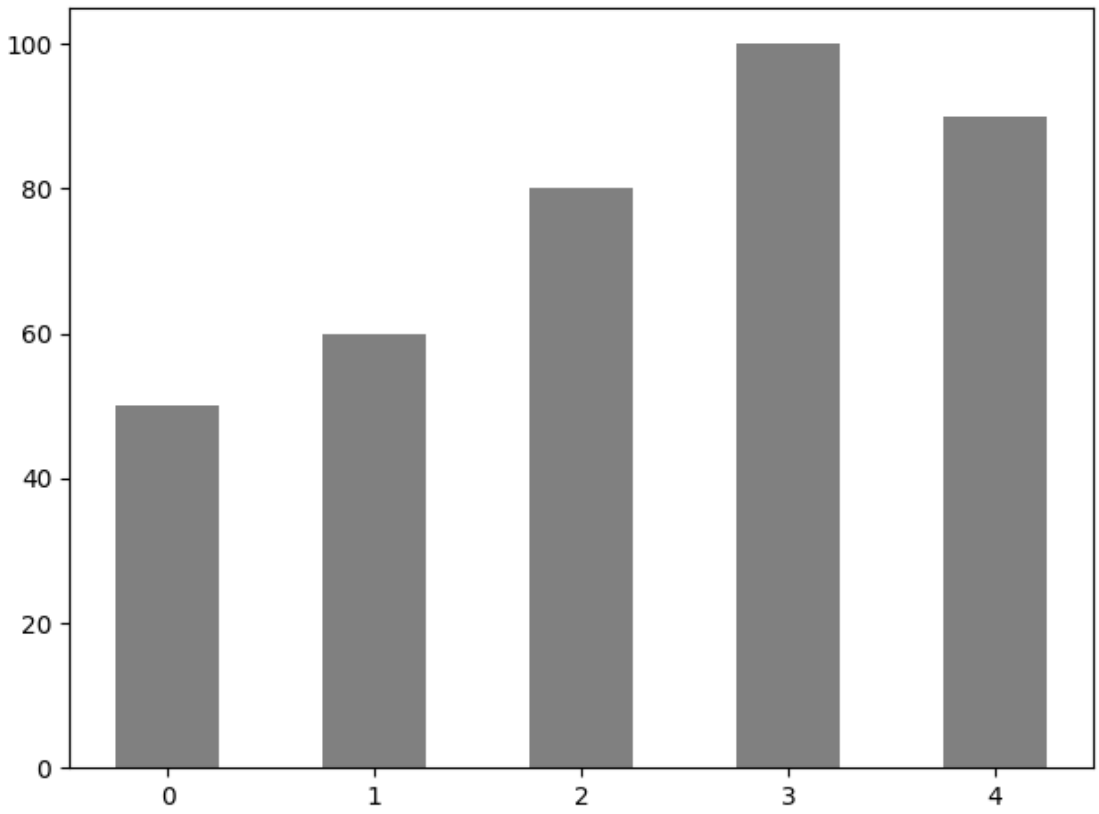
[2]:
```
# raw data: number per category
data=np.array([50,60,80,100,90])
data2=np.array([55,55,70,107,96])
n=len(data)
# horizontal position of each bar
x=np.arange(n)
```

[3]:
```
# basic bar chart
plt.bar(x,data,color="#808080")
plt.tight_layout()
plt.show()
```

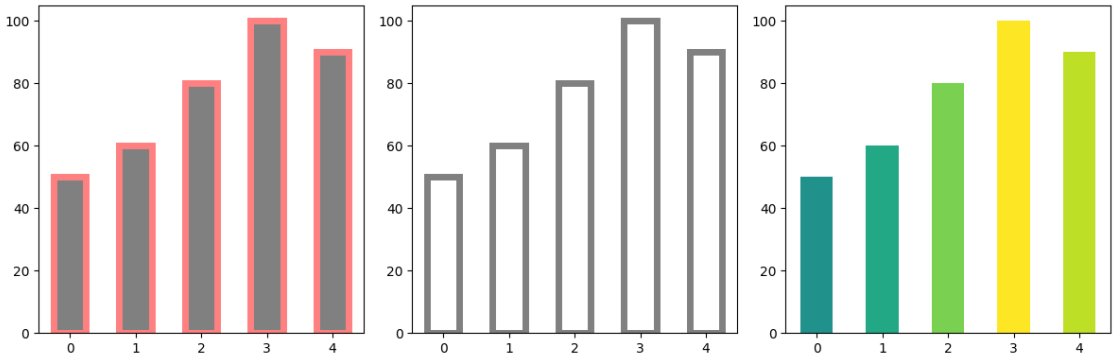


1.2 Adjusting appearance / additional degrees of freedom

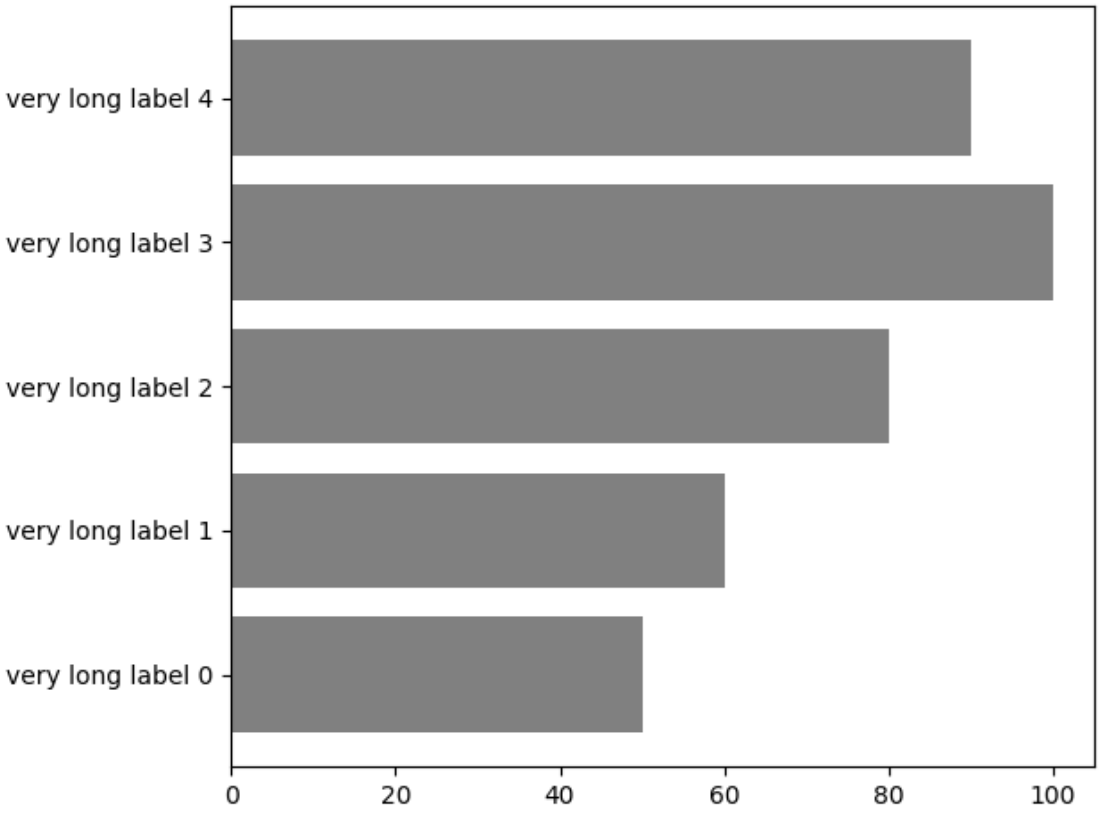
[4]:
```
# adjusting width of bars
# be careful with visual vibrations due to oscillating contrast beween bars!
width=.5
plt.bar(x,data,width=width,color="#808080")
plt.tight_layout()
plt.show()
```



[5]:
```
# draw fancy bars
# careful: this will almost always be visual ducks (in the sense of Tufte)
# use with caution and restraint

fig=plt.figure(figsize=(12,4))
# adding lines
fig.add_subplot(1,3,1)
plt.bar(x,data,width=width,color="#808080",lw=5,edgecolor="#ff8080")
# only lines
fig.add_subplot(1,3,2)
plt.bar(x,data,width=width,fill=False,lw=5,edgecolor="#808080")
# more colors
fig.add_subplot(1,3,3)
plt.bar(x,data,width=width,color=[cm.viridis(i/100) for i in data])

plt.tight_layout()
plt.show()
```

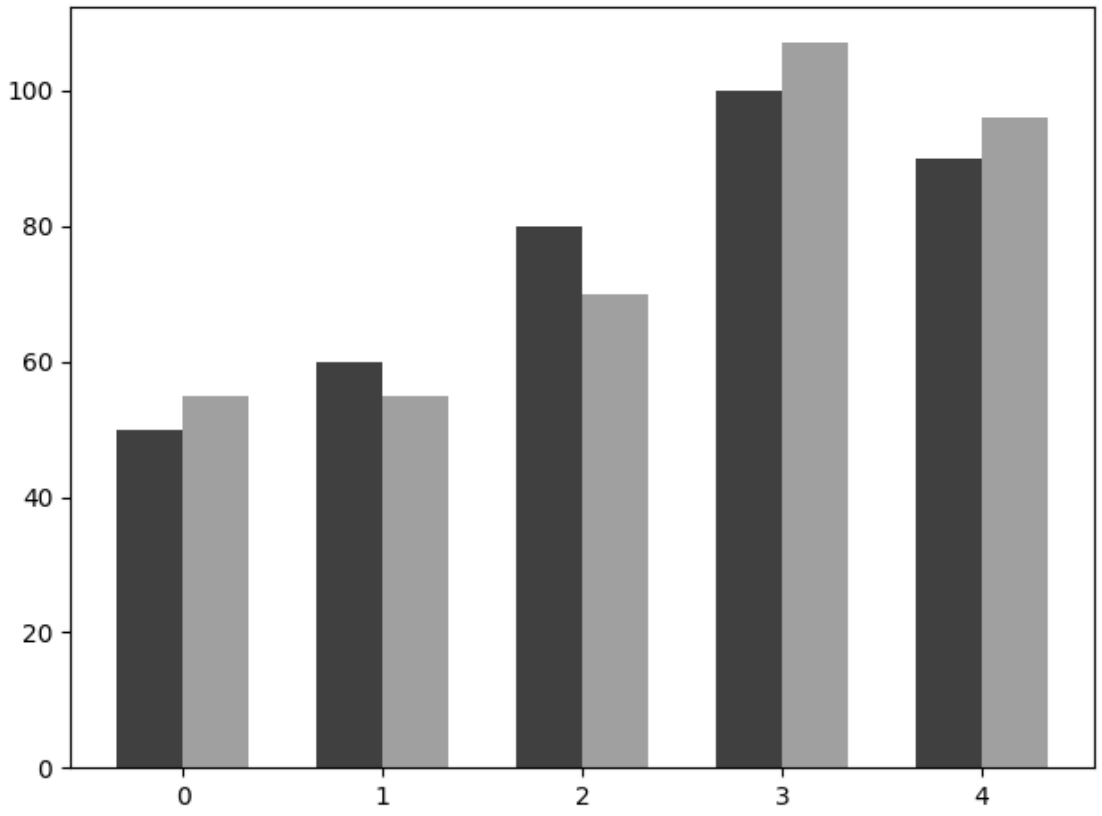


[6]:
```
# horizontal
# often useful for better labelling, may fit better on page
plt.barh(x,data,color="#808080")
labelList=["very long label "+str(i) for i in x]
plt.yticks(ticks=x,labels=labelList)
plt.tight_layout()
plt.show()
```



1.3 Grouped bar chart

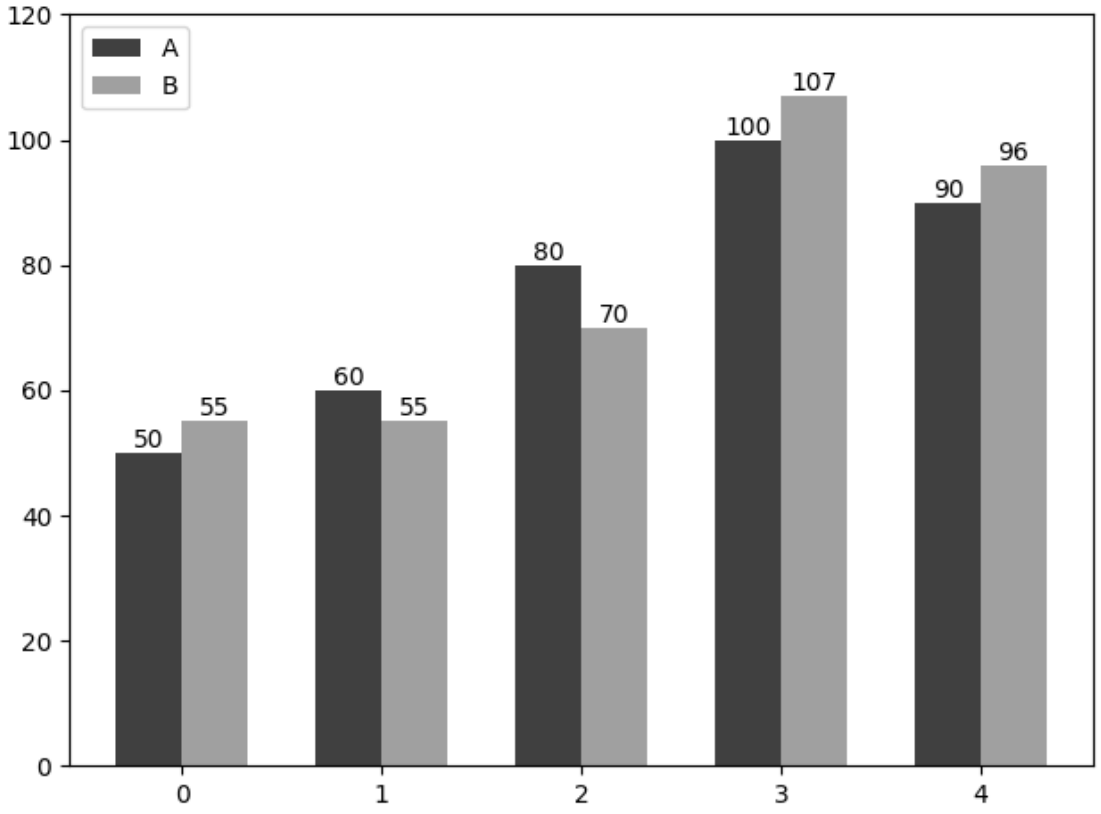
[7]:
```
# can easily be implemented with matplotlib tools
width=0.33
plt.bar(x-width/2,data,width=width,color="#404040")
plt.bar(x+width/2,data2,width=width,color="#A0A0A0")
plt.tight_layout()
plt.show()
```



1.4 Legends and annotations

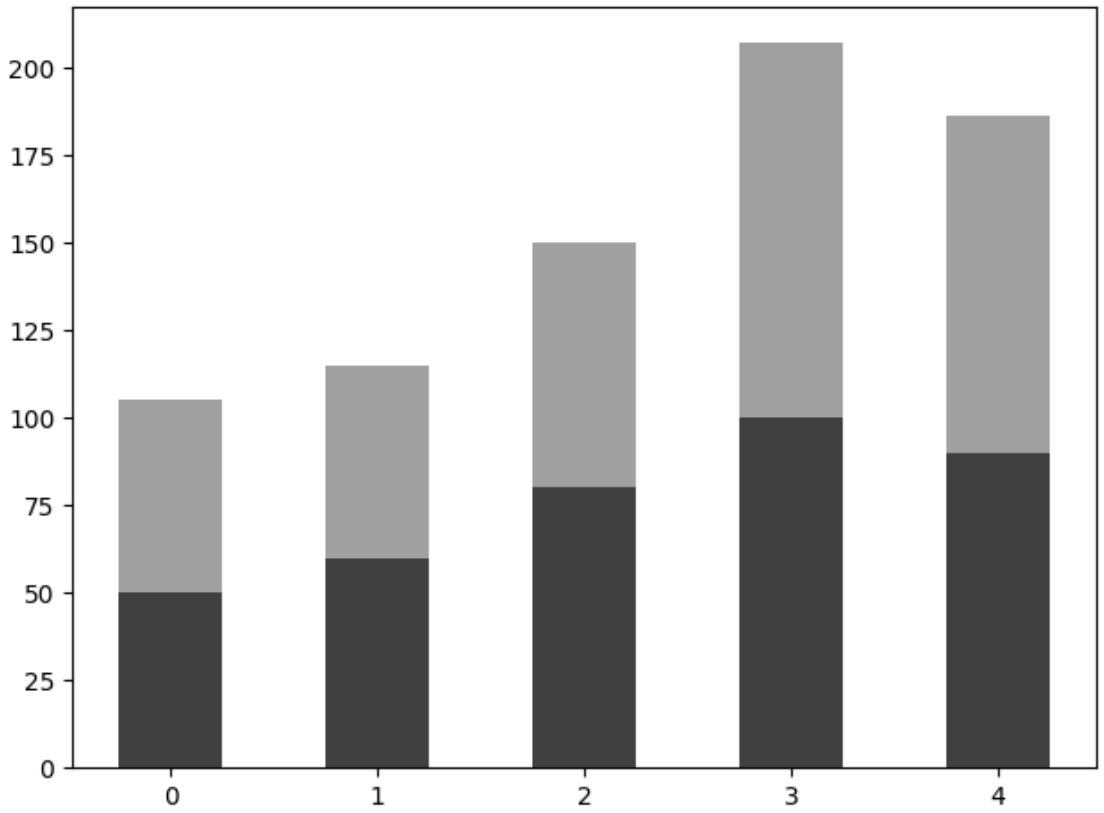
[8]:
```
# adding annotations
width=0.33
b1=plt.bar(x-width/2,data,width=width,label="A",color="#404040")
b2=plt.bar(x+width/2,data2,width=width,label="B",color="#A0A0A0")

# add data annotation (relatively new function)
plt.bar_label(b1)
plt.bar_label(b2)
```

```
# manual implementation
#ax=plt.gca()
#for a,b in zip(x,data):
#    ax.text(a-width/2,b+2,b,va="bottom",ha="center")
#for a,b in zip(x,data2):
#    ax.text(a+width/2,b+2,b,va="bottom",ha="center")


plt.ylim([0,120])

# adding a legend (note the "label" argument above)
plt.legend(loc=2)
plt.tight_layout()
plt.show()
```

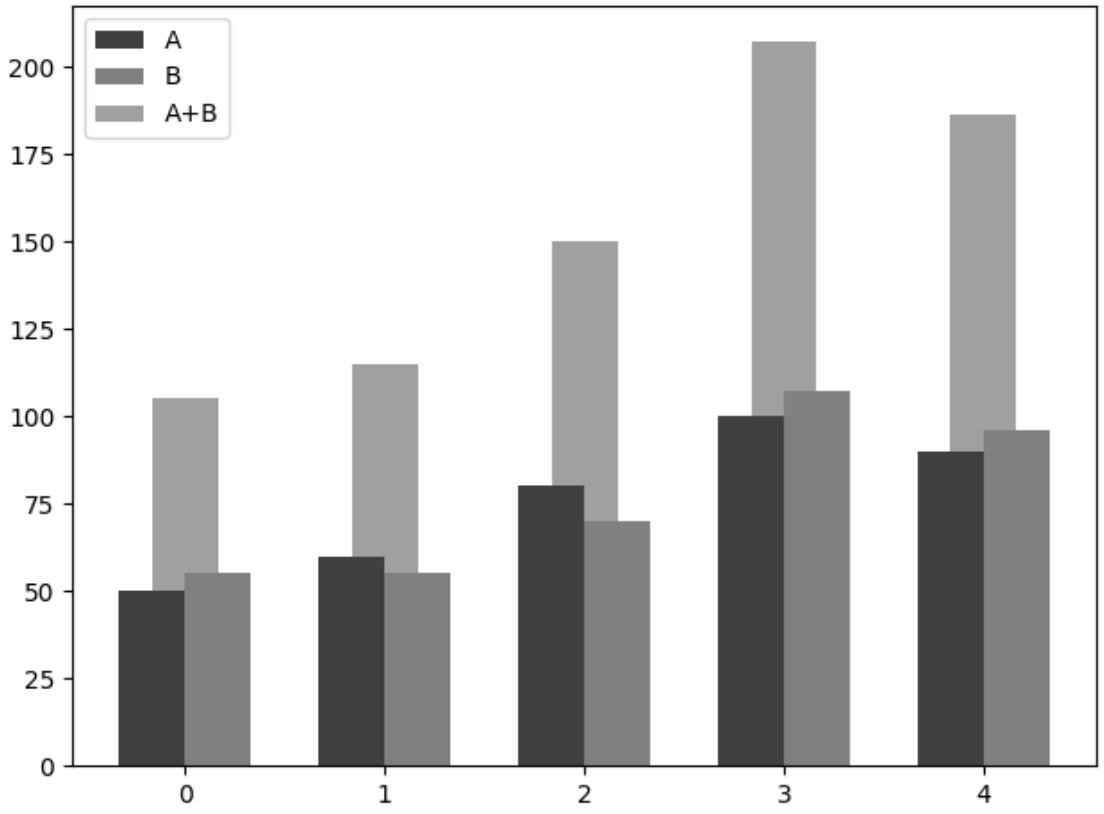


1.5 Stacked

[9]:
```
width=0.5
plt.bar(x,data,width=width,color="#404040")
plt.bar(x,data2,width=width,bottom=data,color="#A0A0A0")
plt.tight_layout()
plt.show()
```

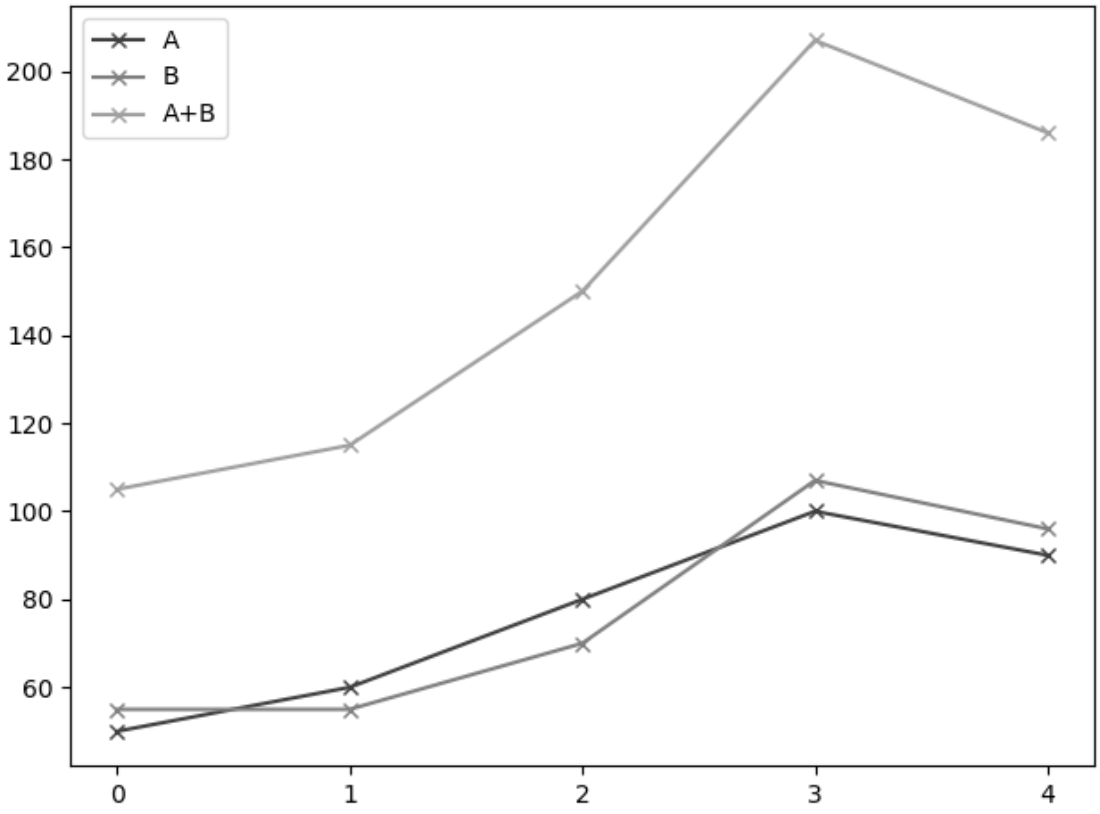


[10]:
```
# an alternative
width=0.33
plt.bar(x-width/2,data,width=width,color="#404040",label="A")
plt.bar(x+width/2,data2,width=width,color="#808080",label="B")
plt.bar(x,data+data2,width=width,color="#A0A0A0",label="A+B",zorder=-1)
plt.legend(loc=2)
plt.tight_layout()
plt.show()
```



1.6 Maybe a scatter /line plot after all?

[11]:
```
# if the x-axis is ordinal/interval data, a line graph often works better
# add lines to visualize trends, visually link points in each data series
n=len(data)
x=np.arange(n)
width=0.5
plt.plot(x,data,marker="x",color="#404040",label="A")
plt.plot(x,data2,marker="x",color="#808080",label="B")
plt.plot(x,data+data2,marker="x",color="#A0A0A0",label="A+B")
plt.xticks(x)
plt.legend(loc=2)
plt.tight_layout()
plt.show()
```


A
B
A+B
200
180
160
140
120
100
80
60
0
1
2
3
4

[ ]:

2023-05-15_ChartTypes_002_Scatter

May 15, 2023

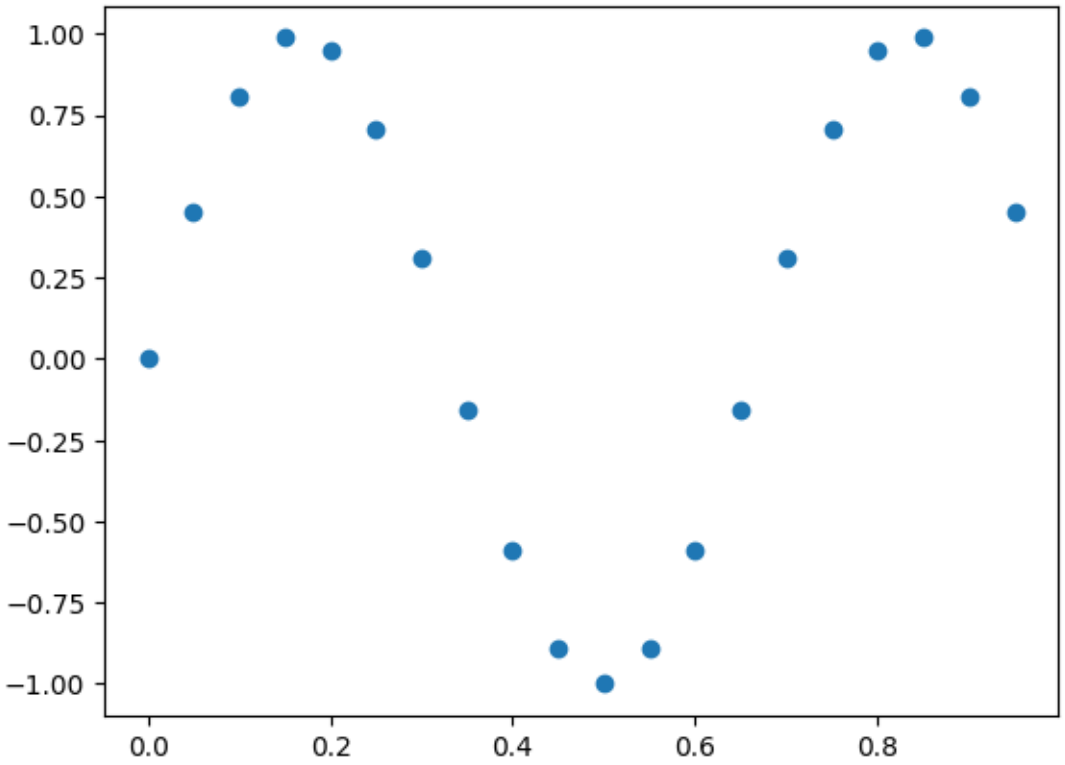
[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

[2]:
```
from sklearn.neighbors import KernelDensity
```

1 Scatter and line plots

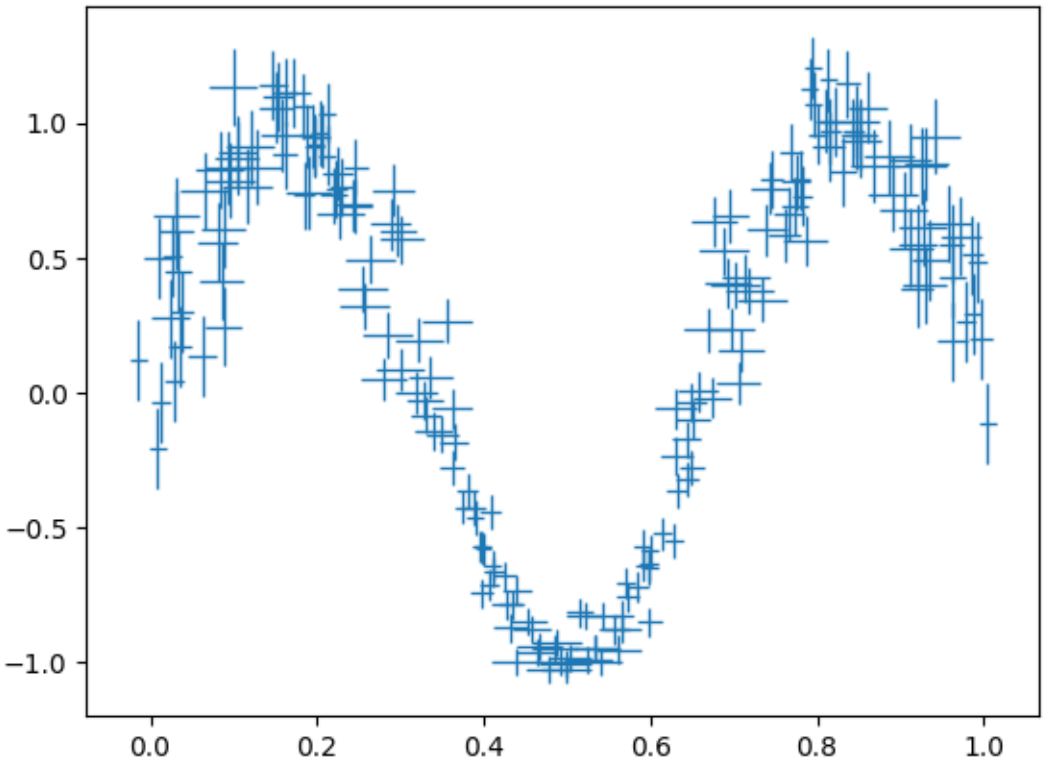
1.1 Standard simple scatter plot

[3]:
```
# generate toy data: samples from sinus curve
nX=20
x=np.arange(nX)/nX
y=np.sin(2*np.pi*x*1.5)

plt.scatter(x,y)
plt.show()
```



1.2 Error bars

[4]:
```
# error bars can represent level of uncertainty, works for both axes!

# use same data as before, now add Gaussian noise
nX=200
x=np.arange(nX)/nX
y=np.sin(2*np.pi*x*1.5)
# choose level of noise in x and y direction
# to make it interesting: the noise depends on x
stdX=0.02*np.sin(2*np.pi*x*2.5)**2+0.01
stdY=0.1*np.cos(2*np.pi*x*0.5)**2+0.05
# apply noise
xErr=x+stdX*np.random.normal(size=x.shape)
yErr=y+stdY*np.random.normal(size=y.shape)

plt.errorbar(xErr,yErr,yerr=stdY,xerr=stdX,lw=0,elinewidth=1)
plt.show()
```



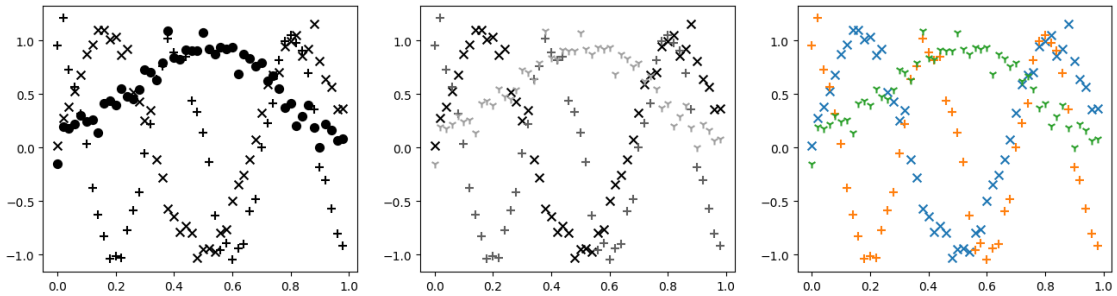
1.3 More graphical degrees of freedom

1.3.1 Markers

- ideal to encode nominal data axis, usually for separating multiple data series
- list of available markers at: https://matplotlib.org/stable/api/markers_api.html
- precise position easier to see with non-bulky marker
- if possible, use color in addition to marker for separating series (markers are merely "backup encoding")
- only a few series can be distinguished in this way

[5]:
```
# thee simple data series with some noise on top
nX=50
x=np.arange(nX)/nX
y1=np.sin(2*np.pi*x*1.5)
y2=np.cos(2*np.pi*x*2.5)
y3=np.exp(-(x-0.5)**2/.1)

stdY=0.1
for y in [y1,y2,y3]:
    y[...]+=stdY*np.random.normal(size=y.shape)
```

```
# plot with different markers
# note: markers alone quickly become messy, color usually works better, even␣
 ↪greyscale

fig=plt.figure(figsize=(16,4))
fig.add_subplot(1,3,1)
plt.scatter(x,y1,marker="x",c="#000000",s=50)
plt.scatter(x,y2,marker="+",c="#000000",s=50)
plt.scatter(x,y3,marker="o",c="#000000",s=50)


fig.add_subplot(1,3,2)
plt.scatter(x,y1,marker="x",c="#000000",s=50)
plt.scatter(x,y2,marker="+",c="#606060",s=50)
plt.scatter(x,y3,marker="1",c="#a0a0a0",s=50)

fig.add_subplot(1,3,3)
plt.scatter(x,y1,marker="x",c=colors[0],s=50)
plt.scatter(x,y2,marker="+",c=colors[1],s=50)
plt.scatter(x,y3,marker="1",c=colors[2],s=50)

plt.show()
```

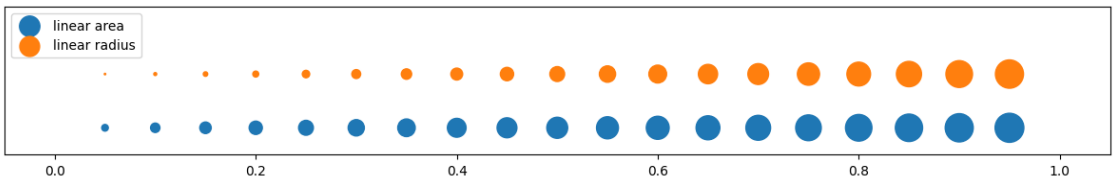


1.3.2 Markersize

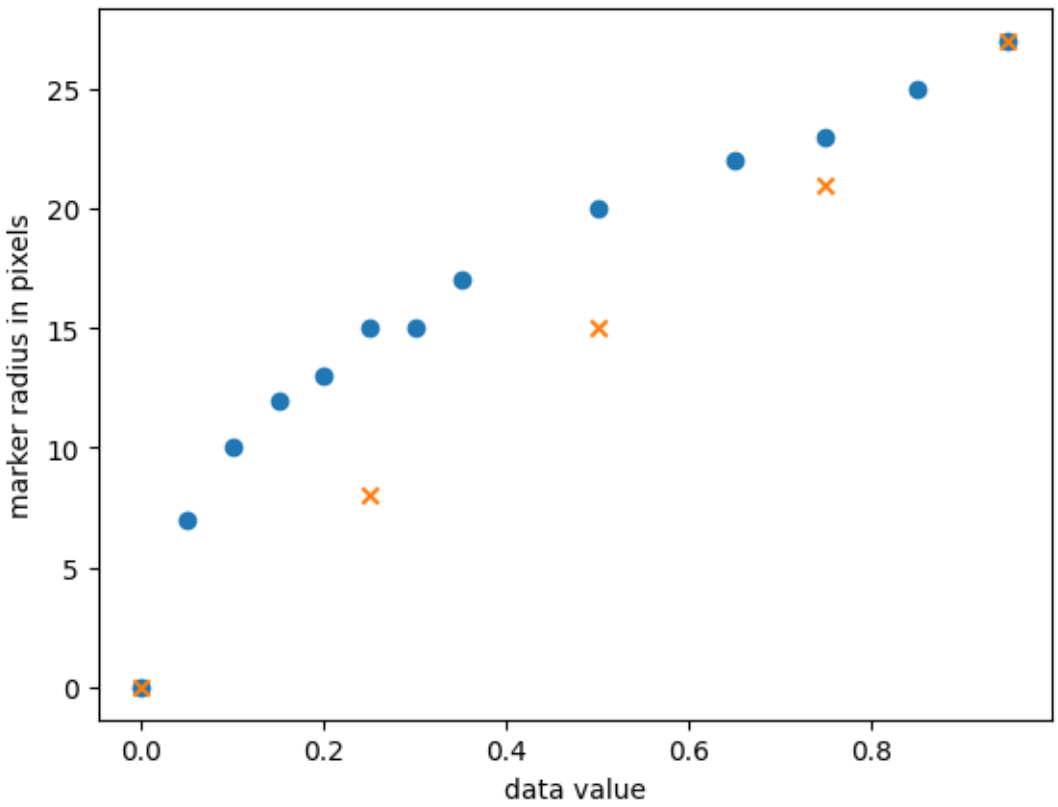
- can be used to encode third, continuous, positive dimension
- something that behaves like a mass, since points with zero marker size will be invisible
- careful: scale area (not radius!) of markers proportional to value (matplotlib does this automatically in newer versions)

Example: scale markersize area and radius

[6]:
```
fig=plt.figure(figsize=(15,2))
n=20
x=np.arange(n)/n

y=np.full(shape=n,fill_value=0.15)
plt.scatter(x,y,s=500*x,label="linear area")

y=np.full(shape=n,fill_value=0.55)
plt.scatter(x,y,s=500*(x**2),label="linear radius")

plt.xlim([-0.05,1.05])
plt.ylim([-0.05,1.05])
plt.yticks([])
plt.legend(loc=2)
plt.show()
```



[7]:
```
# for first sequence (blue): pairs of data point nr and measured radii in pixels
radii1=np.
  ↪array([[19,27],[17,25],[15,23],[13,22],[10,20],[7,17],[6,15],[5,15],[4,13],[3,12],[2,10],[1
radii2=np.array([[19,27],[15,21],[10,15],[5,8],[0,0]])
plt.scatter(radii1[:,0]/n,radii1[:,1])
plt.scatter(radii2[:,0]/n,radii2[:,1],marker="x")
plt.ylabel("marker radius in pixels")
plt.xlabel("data value")
plt.show()
```



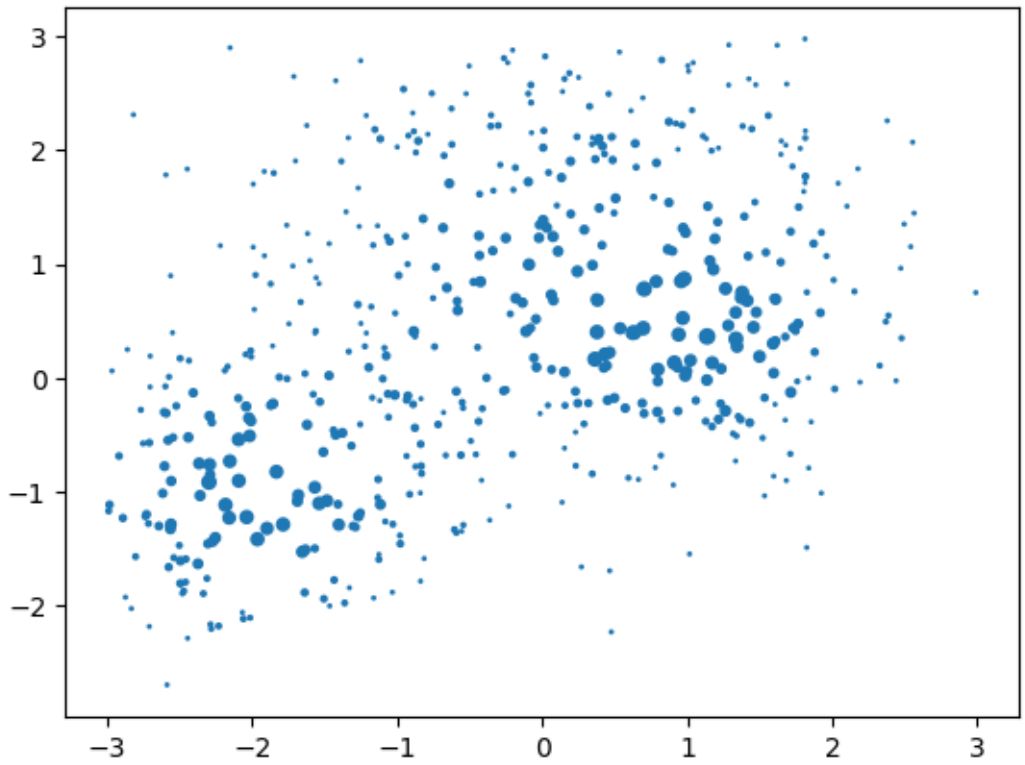
Example: visualize scalar function that is known at sampled points

[8]:
```
# define a 2d density function (based on a mixture of Gaussians)
dim=2
rng=[-3,3]
meanList=np.array([[-2,-1],[0,1],[1,0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])

def getDens(x):
    val=0
    for mean,std,weight in zip(meanList,stdList,weightList):
        val+=(1./np.sqrt(2*np.pi)**dim/std**dim)*np.exp(-0.5*np.
 ↪sum((x-mean)**2)/std**2)*weight
    return val
```

[9]:
```
# now, sample points uniformly from range, sample point mass from Poisson␣
 ↪distribution given by local density
nSamples=1000
lambdaPoisson=100
x=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
```

```
y=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
densxy=np.array([getDens([a,b]) for a,b in zip(x,y)])
m=np.random.poisson(lam=lambdaPoisson*densxy,size=densxy.shape)

# discard zero mass points
keep=(m>0)
x=x[keep]
y=y[keep]
m=m[keep]
```

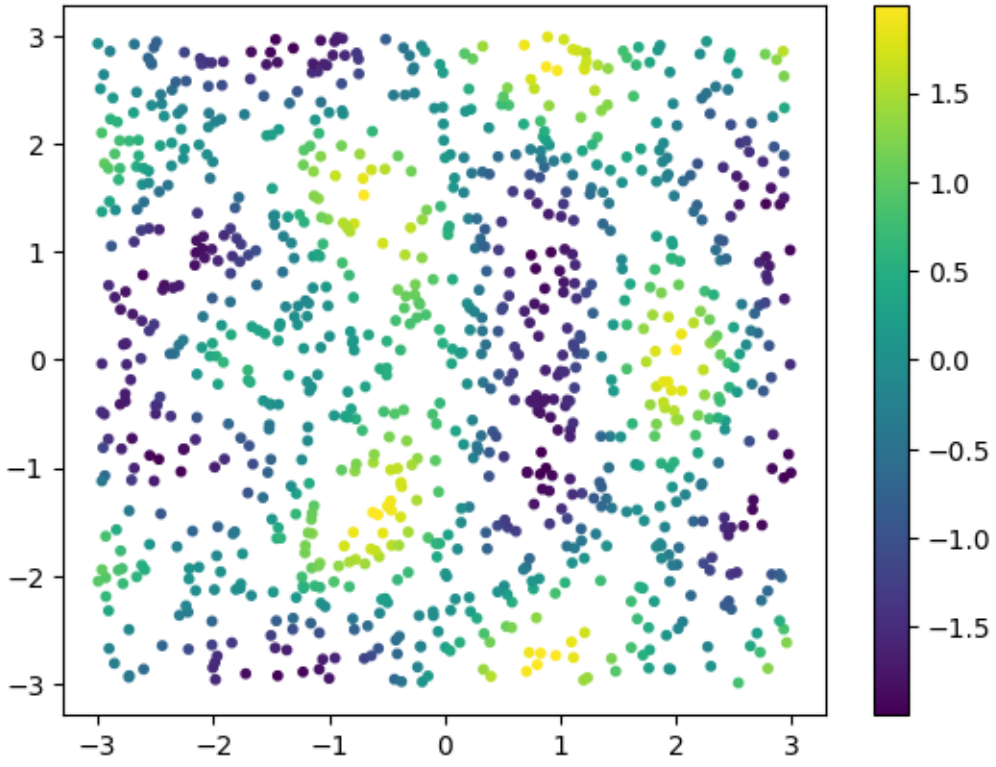
[10]:
```
plt.scatter(x,y,s=m)
plt.show()
```



1.3.3 Color

- separate different data series
- show an independent third dimension, such as height, temperature, density,...
- recall lesson on colors and color maps
- mostly a qualitative display

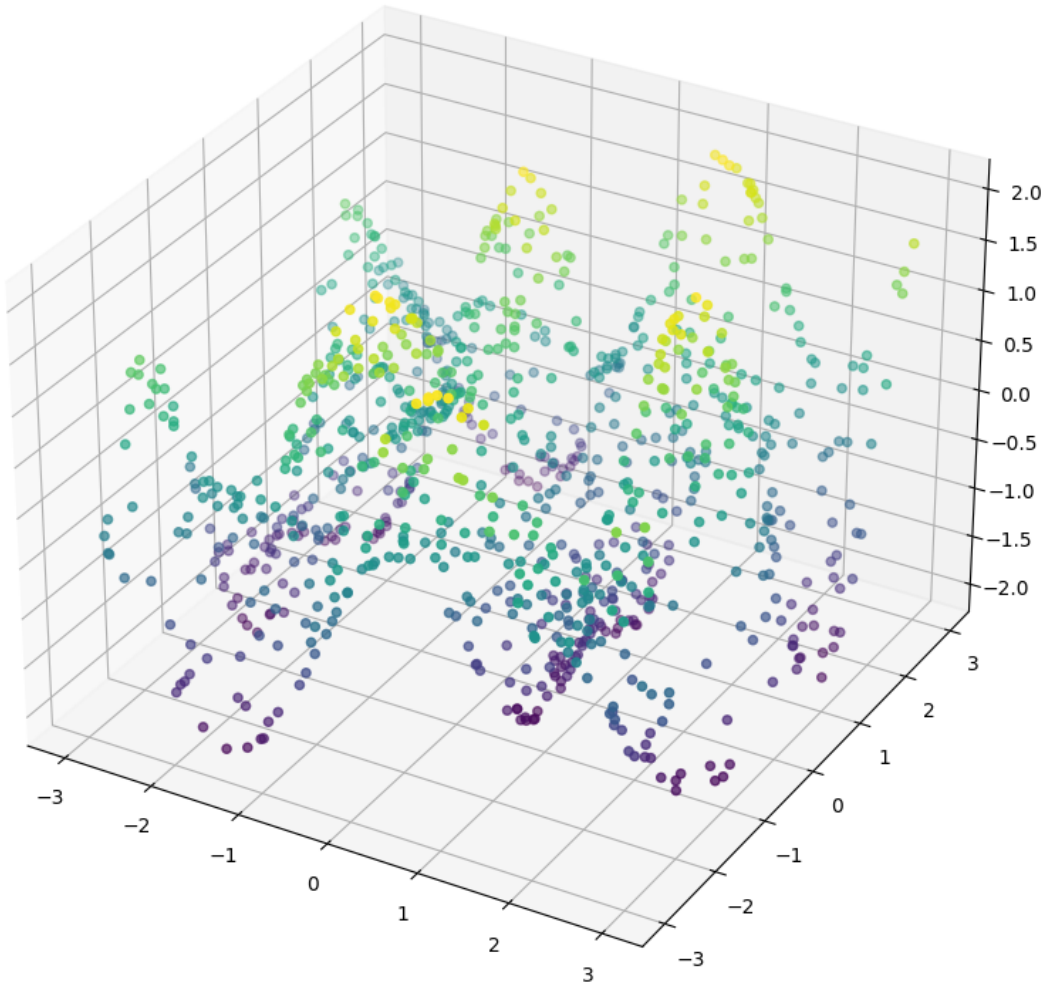
[11]:
```
# similar to marker size example: visualize scalar function on top of points
# toy data
nSamples=1000
rng=[-3,3]
# sample x and y coords uniformly from rectangle
x=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
y=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
# compute f as function of x and y
r1=((x-2)**2+y**2)**.5
r2=((x+2)**2+y**2)**.5
f=np.cos(2*np.pi*r1/3)+np.cos(2*np.pi*r2/2)

# visualize f as color in scatter plot of x and y
fig=plt.figure()
ax=fig.add_subplot()
pltobj=plt.scatter(x,y,c=f,s=10)
fig.colorbar(pltobj, ax=ax)

plt.show()
```



1.4 3d

- visualize additional dimension
- works best with animation or interactive rotation
- perspective is often problematic in 2d static renderings, use with caution; use auxiliary lines such as a mesh grid or color
- data has even more dimensions? will be addressed later

[12]:
```
#%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(x,y,f,c=f)
plt.tight_layout()
plt.show()
```



[13]:
```
plt.close()
%matplotlib inline
```

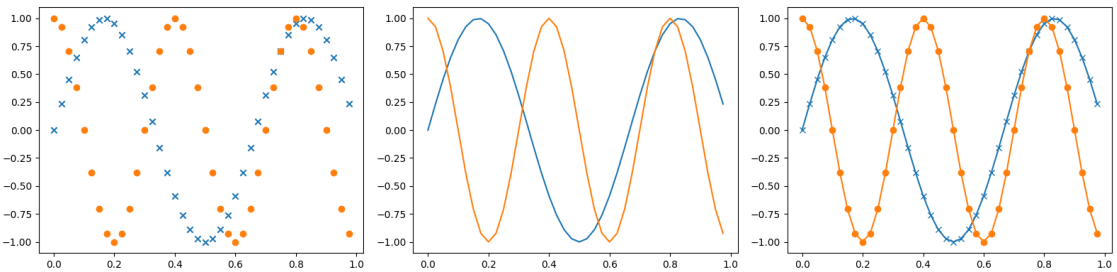
1.5 Lines

when is adding lines appropriate/helpful? * if x-axis is interval data, actually or practically continuous * to visually connect a data series for better distinction from other series * to emphasize trends (line gives explicit encoding of slope) → regression lines * if line is decent approximation of intermediate values that might also be measured

do not use lines if data is highly oscillating, or subject to strong stochastic fluctuations * on data with clear trend + overlap by noise, maybe a smoothed interpolation is appropriate (always be careful to mention how it is generated)

1.5.1 Better visual conectivity of series, emphasis of trends

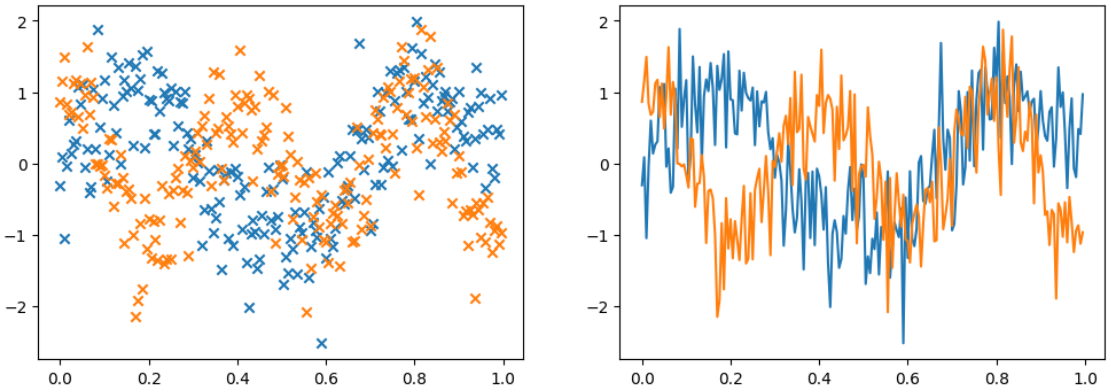
[14]:
```
# toy data
nX=40
x=np.arange(nX)/nX
y1=np.sin(2*np.pi*x*1.5)
y2=np.cos(2*np.pi*x*2.5)

fig=plt.figure(figsize=(16,4))
fig.add_subplot(1,3,1)
plt.scatter(x,y1,marker="x")
plt.scatter(x,y2,marker="o")
fig.add_subplot(1,3,2)
plt.plot(x,y1)
plt.plot(x,y2)
fig.add_subplot(1,3,3)
plt.plot(x,y1,marker="x")
plt.plot(x,y2,marker="o")

plt.tight_layout()
plt.show()
```



1.5.2 Be careful on noisy or oscillating data

- strictly speaking not meaningful, but possibly still better (subjective), be careful about misinterpretation
- below: regression curves

[15]:
```
# noisy toy data
nX=200
x=np.arange(nX)/nX
y1=np.sin(2*np.pi*x*1.5)+0.5*np.random.normal(size=nX)
y2=np.cos(2*np.pi*x*2.5)+0.5*np.random.normal(size=nX)

fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.scatter(x,y1,marker="x")
plt.scatter(x,y2,marker="x")
fig.add_subplot(1,2,2)
plt.plot(x,y1)
plt.plot(x,y2)

plt.show()
```

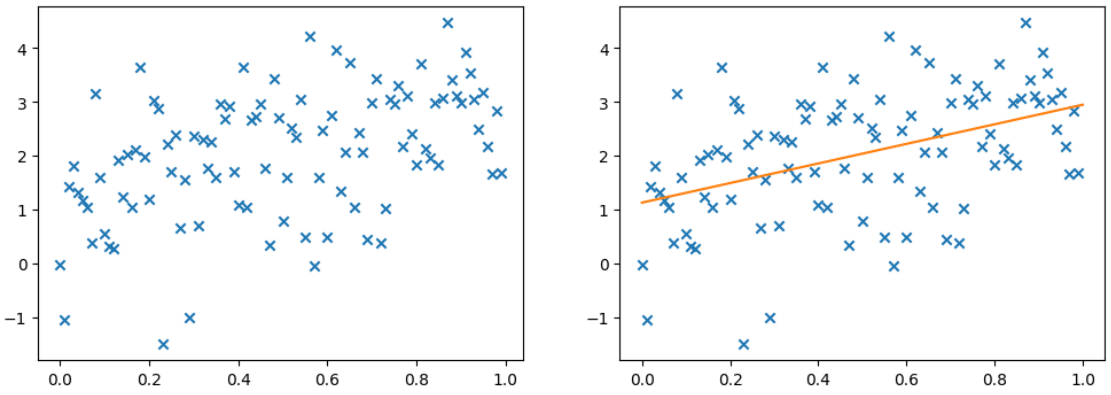


Linear regression

[16]:
```
import scipy.stats
```

[17]:
```
# toy data: y=2x+1 + noise
nX=100
x=np.arange(nX)/nX
```

```
y=2*x+1+np.random.normal(size=nX)

# linear regression
reg=scipy.stats.linregress(x,y)
xref=np.linspace(0,1,num=50)


fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.scatter(x,y,marker="x")

fig.add_subplot(1,2,2)
plt.scatter(x,y,marker="x")
plt.plot(xref,reg.slope*xref+reg.intercept,c=colors[1])

plt.show()
```



Kernel regression

- given: values y_i at locations x_i, interpolate to full space X via:

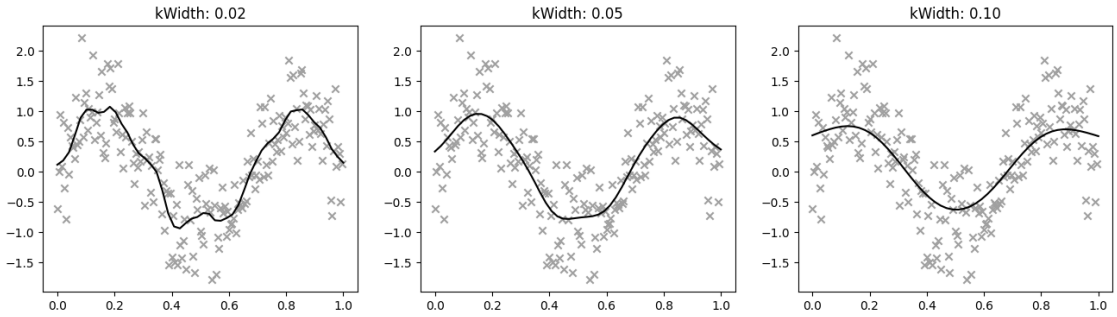
$$f(x) = \frac{\sum_i y_i \cdot k(x, x_i)}{\sum_i k(x, x_i)}$$

- here k is often a Gaussian kernel with a suitable kernel width
- if width is very small: becomes essentially nearest neighbour interpolation
- if width is very high: becomes low pass filter

[19]:
```
def simpleKernelInterpolator(x,y,z,kWidth):
    # works only for 1d arrays x,y,z; not very computationally efficient, only␣
 ↪works on small arrays
    # input empirical data pairs (x,y); interpolate y onto points given by z
    # via Gaussian kernels: at each z sum value of nearby y with Gaussian␣
 ↪weights; normalized to 1
```

```
    # use kWidth for Gaussian kernel width
    weights=np.exp(-0.5*(x.reshape(-1,1)-z.reshape(1,-1))**2/kWidth**2)
    signal=np.einsum(weights,[0,1],y,[0],[1])/np.sum(weights,axis=0)
    return signal
```

[20]:
```
nX=200
x=np.arange(nX)/nX
y1=np.sin(2*np.pi*x*1.5)+0.5*np.random.normal(size=nX)
z=np.linspace(0,1,num=50)

fig=plt.figure(figsize=(16,4))


for i,kWidth in enumerate([0.02,0.05,0.1]):
    fig.add_subplot(1,3,1+i)

    signal=simpleKernelInterpolator(x,y1,z,kWidth)
    plt.scatter(x,y1,marker="x",color="#A0A0A0")
    plt.plot(z,signal,color="#000000")
    plt.title("kWidth: {:.02f}".format(kWidth))
plt.show()
```

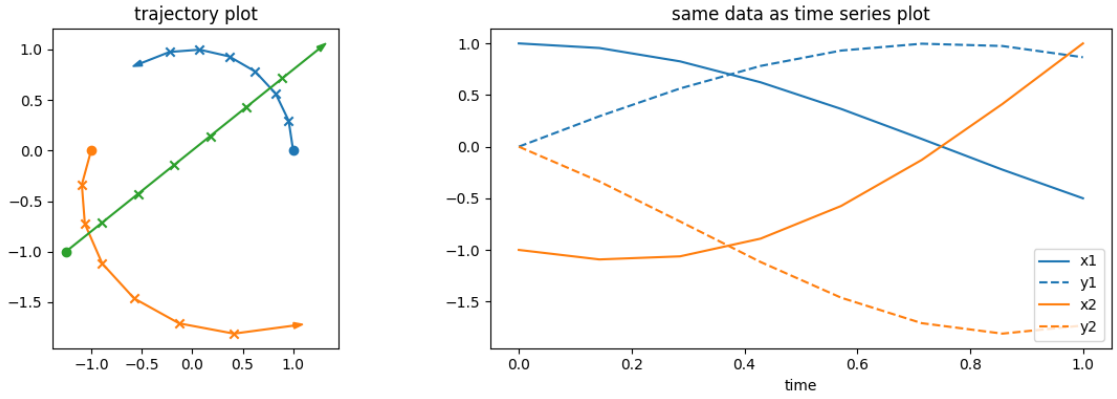


1.5.3 Trajectory plot / connected scatter plot

[21]:
```
def pltTrajectory(ax,x,y,**kwargs):
    ax.plot(x,y,**kwargs)
    ax.arrow(x[-1],y[-1],0.01*(x[-1]-x[-2]),0.01*(y[-1]-y[-2]),**kwargs,width=0.
 ↪02)
    ax.scatter(x[0],y[0],**kwargs,marker="o")
    ax.scatter(x[1:-1],y[1:-1],**kwargs,marker="x")
```

[22]:
```
nPts=8
t=np.linspace(0,1,nPts)
```

```
x1=np.cos(2*np.pi*t/3)
y1=np.sin(2*np.pi*t/3)

x2=(1+t)*np.cos(2*np.pi*(t/3+0.5))
y2=(1+t)*np.sin(2*np.pi*(t/3+0.5))

x3=2.5*(t-0.5)
y3=2*(t-0.5)


fig=plt.figure(figsize=(12,4))
# since now both axes represent length in same "physical space", aspect ratio␣
 ↪of axes becomes relevant!
ax=fig.add_subplot(1,2,1,aspect=1.)
plt.title("trajectory plot")
pltTrajectory(ax,x1,y1,color=colors[0])
pltTrajectory(ax,x2,y2,color=colors[1])
pltTrajectory(ax,x3,y3,color=colors[2])

fig.add_subplot(1,2,2)
plt.title("same data as time series plot")
plt.plot(t,x1,color=colors[0],label="x1")
plt.plot(t,y1,color=colors[0],ls="dashed",label="y1")
plt.plot(t,x2,color=colors[1],label="x2")
plt.plot(t,y2,color=colors[1],ls="dashed",label="y2")
plt.xlabel("time")
plt.legend(loc=4)

plt.tight_layout()
plt.show()
```



Example: interacting particles in potential Here we consider a simple model of n interacting particles in 2 dimensions. Each particle follows the following equation of motion:

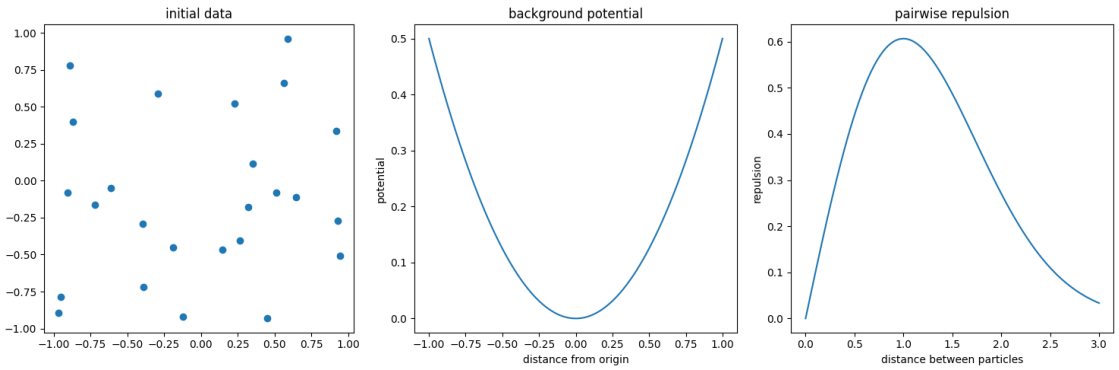
$$\partial_t x_i = -\nabla V(x_i) + \frac{1}{n}\sum_{j=1}^{n} F(x_j - x_i)$$

* for some initial positions $(x_1(0), \dots, x_n(0))$ * for some background potential V, here simply $V(x) = \frac{1}{2}\|x\|^2$ * for some interaction force, here $F(z) = -z \cdot \exp(-\|z\|^2/2)$

[22]:
```
# set up the simulation
dim=2

def gradV(x):
    """gradient of static background potential in which the particles want to␣
 ↪move
    (in this case: simple quadratic well around origin)"""
    return x

def g(s):
    """scalar function that modulates strength of pairwise interaction"""
    return np.exp(-0.5*s**2)

def pairwiseForce(x,y):
    """pairwise forces between all particles"""
    # relative position between all particles
    deltaij=x.reshape((-1,1,dim))-y.reshape((1,-1,dim))
    # relative distance
    dij=np.linalg.norm(deltaij,axis=2)
    fi=np.einsum(deltaij,[0,1,2],g(dij),[0,1],[0,2])/n
    return fi


# initial conditions
if False:
    # uniform on 2d grid
    n1d=5
    x0_1d=np.linspace(-1,1,num=n1d)
    x0=np.zeros((n1d,n1d,dim))
    x0[:,:,0]=x0_1d.reshape((1,-1))
    x0[:,:,1]=x0_1d.reshape((-1,1))
    x0=x0.reshape((-1,dim))
if True:
    # uniform random sampling from unit square
    n=25
    x0=2*(np.random.random(size=(n,dim))-0.5)
if False:
    # manual
    x0=np.array([[1,0],[-1,0]])
```

```
n=x0.shape[0]
```

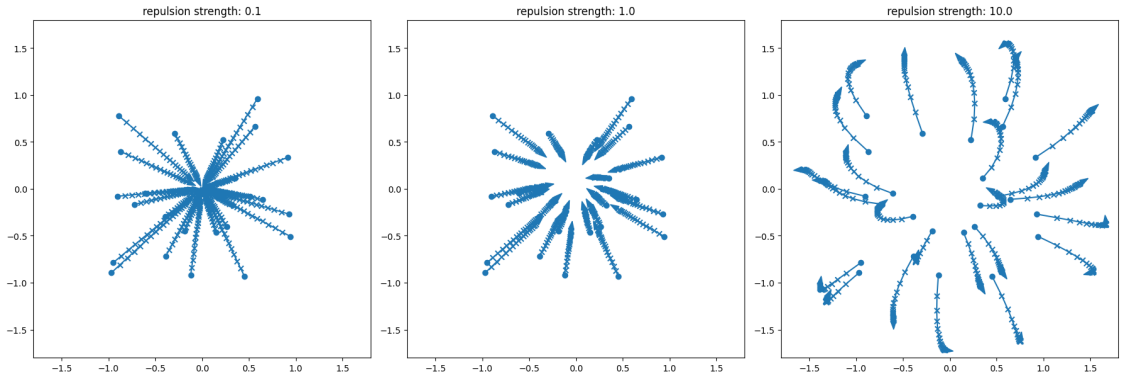
[23]:
```
# before we get to actual simulation:
# * visualize initial data
# * visualize potential
# * visualize pairwise force
fig=plt.figure(figsize=(15,5))

fig.add_subplot(1,3,1,aspect=1.)
plt.title("initial data")
plt.scatter(x0[:,0],x0[:,1])

fig.add_subplot(1,3,2)
plt.title("background potential")
z=np.linspace(-1,1,num=100)
plt.plot(z,0.5*z**2)
plt.xlabel("distance from origin")
plt.ylabel("potential")

fig.add_subplot(1,3,3)
plt.title("pairwise repulsion")
z=np.linspace(0,3,num=100)
plt.plot(z,z*np.exp(-z**2/2))
plt.xlabel("distance between particles")
plt.ylabel("repulsion")


plt.tight_layout()
plt.show()
```



[24]:
```
def getSimulation(x0,forceStrength):
    nT=1000
    tList=np.linspace(0,2,num=nT)
```

```
    def f(t,x):
        return (-gradV(x.reshape((-1,dim)))+\
                forceStrength*pairwiseForce(x.reshape((-1,dim)),x.
 ↪reshape((-1,dim)))).ravel()

    res=scipy.integrate.solve_ivp(f, tList[[0,-1]], x0.ravel(), method='RK45',␣
 ↪t_eval=tList)
    x=res["y"].transpose().reshape((nT,n,dim))
    return x
```

[25]:
```
fig=plt.figure(figsize=(18,6))

nSubT=50 # only show every nSubT-th point to avoid plot being too cluttered

# now do this simulation for various forces
# show them in same coordinate system (axis ranges)
for k,forceStrength in enumerate([0.1,1,10]):
    ax=fig.add_subplot(1,3,1+k,aspect=1.)
    plt.title("repulsion strength: {:.01f}".format(forceStrength))

    x=getSimulation(x0,forceStrength)
    for i in range(n):
        pltTrajectory(ax,x[::nSubT,i,0],x[::nSubT,i,1],color=colors[0])
    scale=1.8
    ax.set_xlim([-scale,scale])
    ax.set_ylim([-scale,scale])
plt.tight_layout()
plt.show()
```



1.6 Logarithmic axis scaling

1.6.1 Application 1: visualize exponential growth

consider following model:

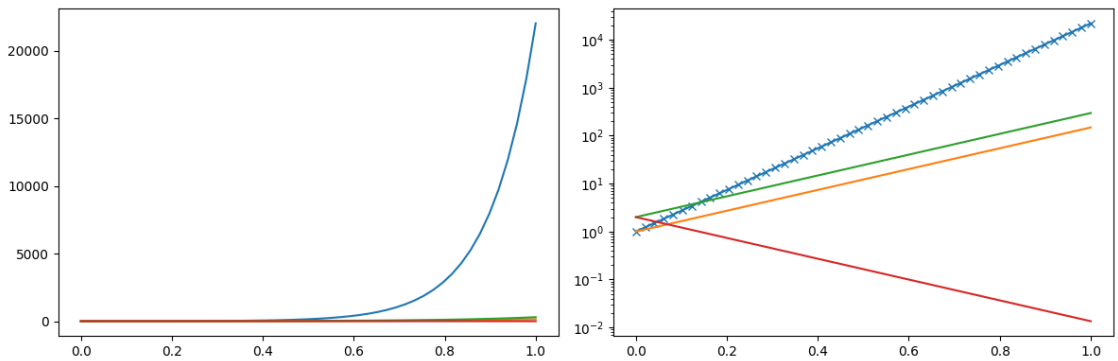
$$y(x) = a \cdot \exp(b \cdot x)$$

Now scale y-axis logarithmically, that is, plot $\log(y)$ instead of y. One finds:

$$\log(y(x)) = \log(a) + b \cdot x$$

* linear relation beween $\log(y)$ and x * prefactor a becomes intercept $\log(a)$ * time-scale b becomes slope

[26]:
```
nX=50
x=np.linspace(0,1,num=nX)
y1=np.exp(10*x)
y2=np.exp(5*x)
y3=2*np.exp(5*x)
y4=2*np.exp(-5*x)

fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.plot(x,y1,marker="")
plt.plot(x,y2,marker="")
plt.plot(x,y3,marker="")
plt.plot(x,y4,marker="")

fig.add_subplot(1,2,2)
plt.plot(x,y1,marker="x")
plt.plot(x,y2,marker="")
plt.plot(x,y3,marker="")
plt.plot(x,y4,marker="")
plt.yscale("log")

plt.tight_layout()
plt.show()
# note: markers of exponential data equally spaced in log plot
```



1.6.2 Application 2: visualize power laws

consider following model:

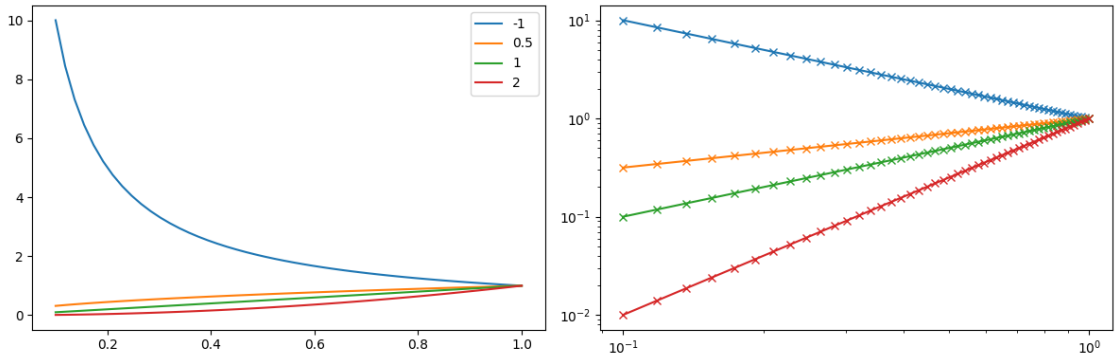
$$y(x) = ax^p$$

Now scale x and y-axis logarithmically. Now need to express $\log(y)$ as function of $\log(x)$:

$$\log(y) = \log(a) + p \cdot \log(x)$$

* linear relation beween $\log(y)$ and $\log(x)$ * prefactor a becomes intercept $\log(a)$ * exponent p becomes slope

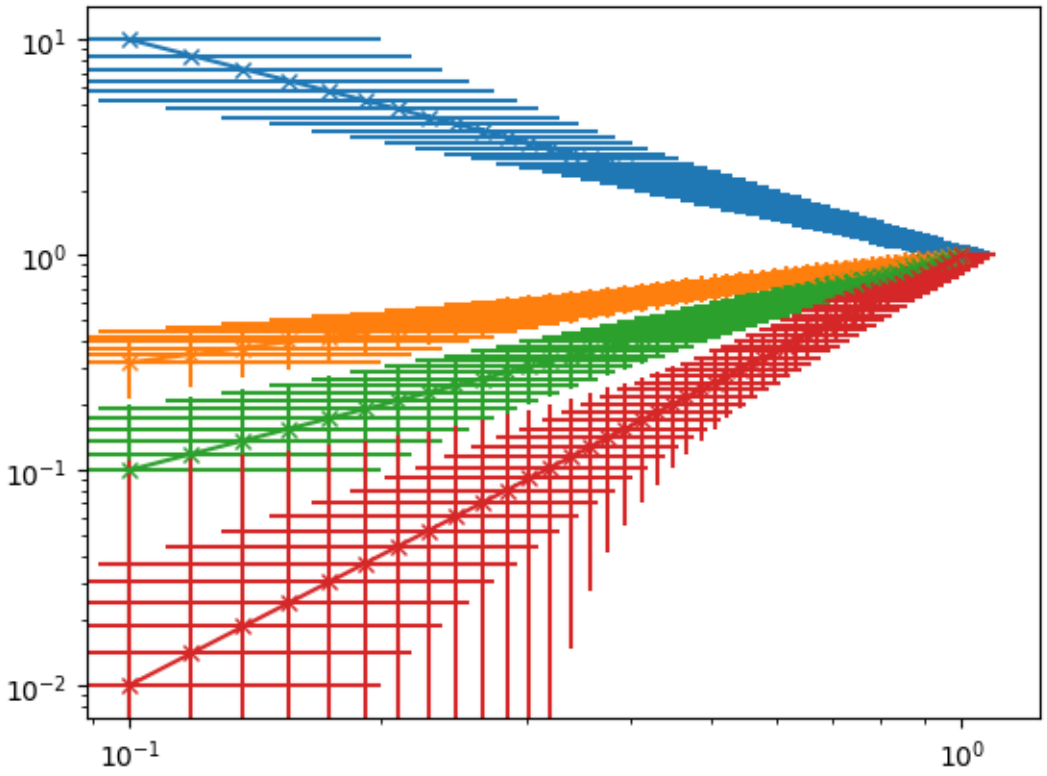
[27]:
```
nX=50
x=np.linspace(0.1,1,num=nX)
pList=[-1,0.5,1,2]
yList=[x**p for p in pList]

fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
for p,y in zip(pList,yList):
    plt.plot(x,y,marker="",label=p)
plt.legend()
#plt.ylim([0,2])

fig.add_subplot(1,2,2)
for y in yList:
    plt.plot(x,y,marker="x")
plt.xscale("log")
plt.yscale("log")

plt.tight_layout()
plt.show()
# note: markers no longer equally spaced in x
```



Errorbars in log plot

- note: if errors are constant in size, they will look vastly different in log plot
- therefore: be very careful with regression on such data! linear regression on log-data is usually not a good idea

[28]:
```
fig=plt.figure()
fig.add_subplot()
for y in yList:
    plt.errorbar(x,y,xerr=0.1,yerr=0.1,marker="x")
plt.xscale("log")
plt.yscale("log")

plt.show()
```



[ ]:
```

```

2023-05-15_ChartTypes_003_Histograms-Densities

May 25, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

[2]:
```
from sklearn.neighbors import KernelDensity
```

1 Distribution of points, histograms, estimated densities

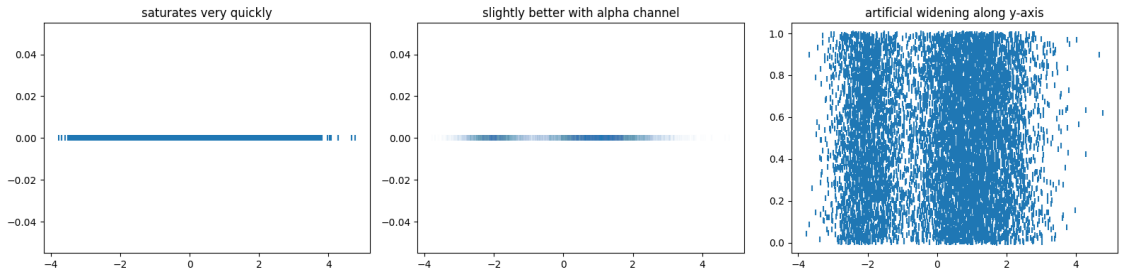
1.1 1d

[4]:
```
# number of sample points
nX=10000
# relevant region
rng=[-5,5]

# specify a simple Gaussian mixture model:
# lists of: mean, standar deviation, relative weight
meanList=[-2,1]
stdList=[0.5,0.9]
weightList=[0.3,0.7]

# sample points from Gaussian mixture model
x=np.zeros((0,),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    x=np.concatenate((x,mean+std*np.random.normal(size=int(nX*weight))))
```

```
# evaluate the true Gaussian density function for comparison
nPlot=1000
x_plot=np.linspace(rng[0],rng[1],num=nPlot)
real_dens=np.zeros_like(x_plot)
for mean,std,weight in zip(meanList,stdList,weightList):
    real_dens+=(1./np.sqrt(2*np.pi)/std)*np.exp(-0.5*(x_plot-mean)**2/
 ↪std**2)*weight
```

Experiment: can we get an impression of the distribution of points from a pure "1d"-scatter plot?

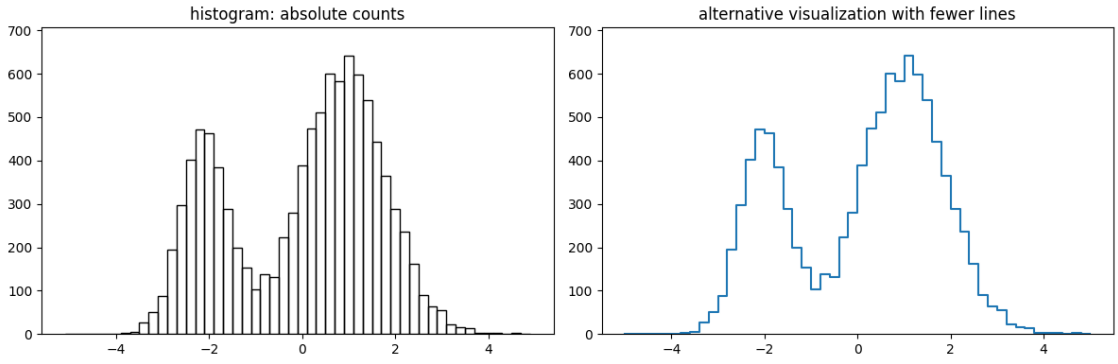
[6]:
```
fig=plt.figure(figsize=(16,4))

fig.add_subplot(1,3,1)
plt.title("saturates very quickly")
plt.scatter(x,np.full(x.shape,0.),marker="|")

fig.add_subplot(1,3,2)
plt.title("slightly better with alpha channel")
plt.scatter(x,np.full(x.shape,0.),marker="|",alpha=0.01)

# does it get better with some artificial widening along the y-axis?
# technique is also called "jitter", built-in option elsewhere, e.g. in ggplot2␣
 ↪in R
fig.add_subplot(1,3,3)
plt.title("artificial widening along y-axis")
plt.scatter(x,np.random.random(size=x.shape),marker="|")

plt.tight_layout()
plt.show()
```



Simple 1d histogram

[7]:
```
# number of bins, width of bins
nBins=50
kWidth=(rng[1]-rng[0])/nBins
```

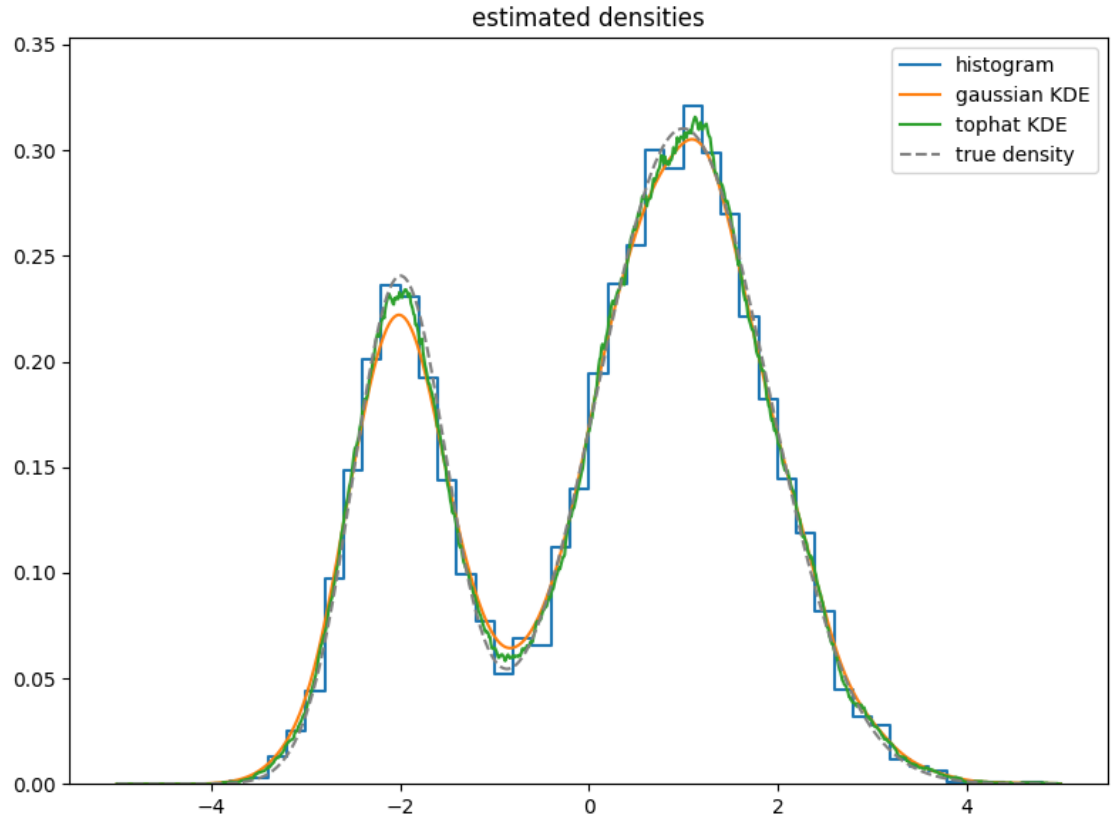
```
# standard histogram
hist,edges=np.histogram(x,range=rng,bins=nBins)

fig=plt.figure(figsize=(12,4))

# histogram of discrete counts per bin
fig.add_subplot(1,2,1)
plt.bar(edges[:-1],hist,width=edges[1]-edges[0],fill=False)
plt.ylim([0,np.max(hist)*1.1])
plt.title("histogram: absolute counts")

fig.add_subplot(1,2,2)
plt.step(edges,np.concatenate((np.array([0.]),hist)),c=colors[0])
plt.ylim([0,np.max(hist)*1.1])
plt.title("alternative visualization with fewer lines")


plt.tight_layout()
plt.show()
```



(Kernel)-density estimation

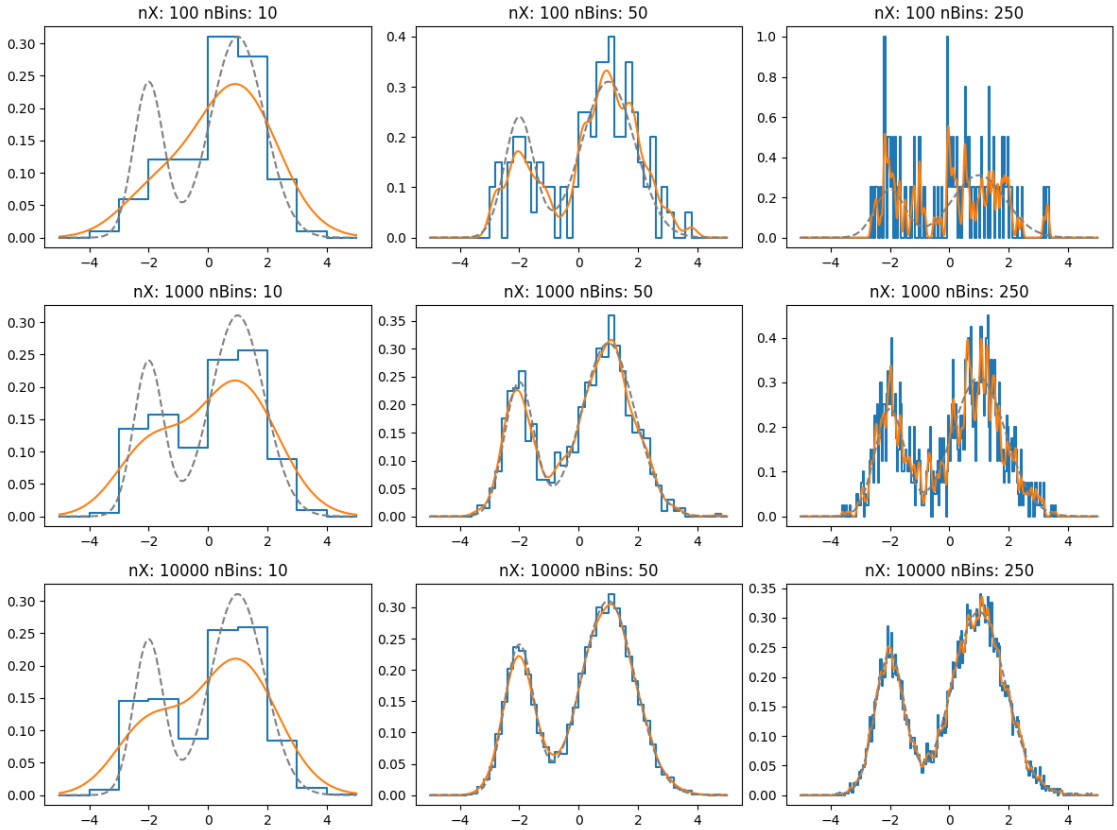
[9]:
```
# normalization of absolute counts as density function
#hist_dens=hist/np.sum(hist)/kWidth
# alternative: use proper keyword
hist_dens,edges=np.histogram(x,range=rng,bins=nBins,density=True)


# Gaussian kernel density estimation
kde = KernelDensity(kernel='gaussian', bandwidth=kWidth).fit(x.reshape((-1,1)))
log_dens = kde.score_samples(x_plot.reshape((-1,1)))
```

```
# tophat kernel density estimation
kde_tophat = KernelDensity(kernel='tophat', bandwidth=kWidth).fit(x.
  ↪reshape((-1,1)))
log_dens_tophat = kde_tophat.score_samples(x_plot.reshape((-1,1)))

fig=plt.figure(figsize=(8,6))

# comparison of different density estimations
#fig.add_subplot(1,1,1)
plt.step(edges,np.concatenate((np.array([0.
  ↪]),hist_dens)),c=colors[0],label="histogram")
plt.plot(x_plot,np.exp(log_dens),c=colors[1],label="gaussian KDE")
plt.plot(x_plot,np.exp(log_dens_tophat),c=colors[2],label="tophat KDE")
plt.plot(x_plot,real_dens,c="grey",ls="dashed",label="true density")

plt.ylim([0,np.max(hist_dens)*1.1])

plt.title("estimated densities")
plt.legend()


plt.tight_layout()
plt.show()
```



Number of samples, number of bins, kernel width

[10]:
```
# subsample full point cloud
nXList=[100,1000,10000]
nBinsList=[10,50,250]
fig=plt.figure(figsize=(4*len(nBinsList),3*len(nXList)))
for i,nXL in enumerate(nXList):
    for j,nBinsL in enumerate(nBinsList):

        # subsample points
        xL=x[np.random.choice(nX,size=nXL,replace=False)]
        # recompute histogram and gaussian kernel density
        kWidthL=(rng[1]-rng[0])/nBinsL
        hist_dens,edges=np.histogram(xL,range=rng,bins=nBinsL,density=True)

        kde = KernelDensity(kernel='gaussian', bandwidth=kWidthL).fit(xL.
 ↪reshape((-1,1)))
        log_dens = kde.score_samples(x_plot.reshape((-1,1)))

        fig.add_subplot(len(nXList),len(nBinsList),i*len(nBinsList)+j+1)
        plt.title("nX: {:d} nBins: {:d}".format(nXL,nBinsL))
        plt.step(edges,np.concatenate((np.array([0.
 ↪]),hist_dens)),c=colors[0],label="histogram")
        plt.plot(x_plot,np.exp(log_dens),c=colors[1],label="gaussian KDE")
        plt.plot(x_plot,real_dens,c="grey",ls="dashed",label="true density")

plt.tight_layout()
plt.show()
```

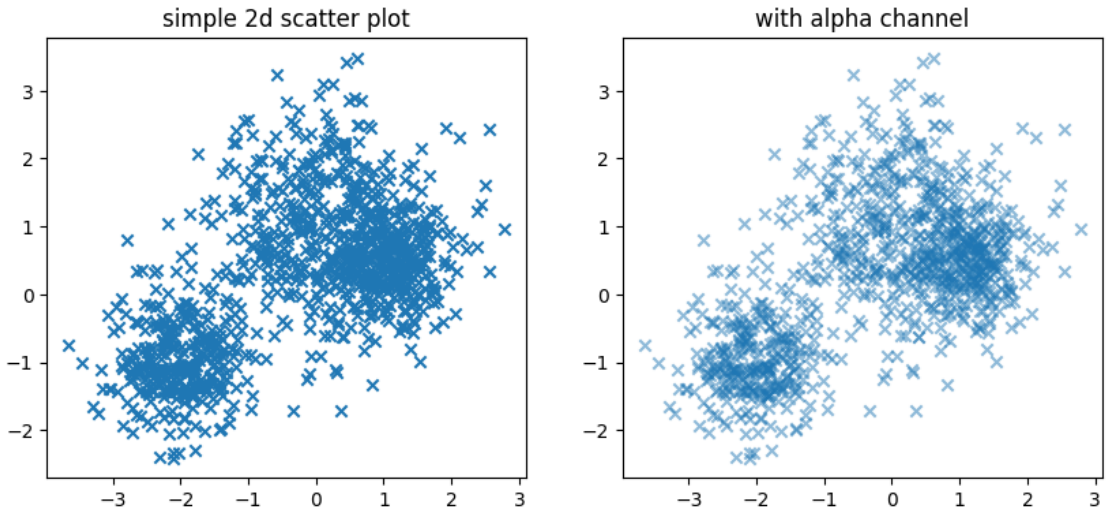


1.2 2d

[11]:
```
# 2d Gaussian mixture model
nX=1000
dim=2
rng=[-3,3]
meanList=np.array([[-2,-1],[0,1],[1,0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])


# sample points from Gaussian mixture model
pts=np.zeros((0,dim),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))
```

[12]:
```
# as above: try simple scatter plot first
fig=plt.figure(figsize=(10,5))
```

```
fig.add_subplot(1,2,1,aspect=1.)
plt.title("simple 2d scatter plot")
plt.scatter(pts[:,0],pts[:,1],marker="x",alpha=1.)
fig.add_subplot(1,2,2,aspect=1.)
plt.title("with alpha channel")
plt.scatter(pts[:,0],pts[:,1],marker="x",alpha=.5)

plt.show()
```

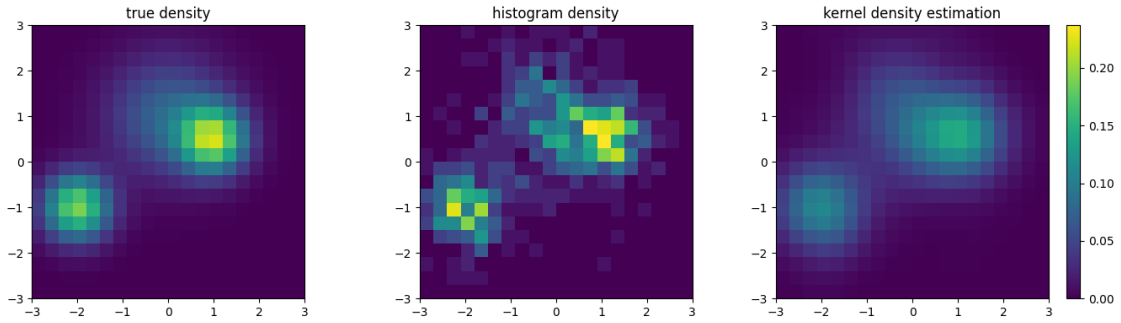


[13]:
```
# 2d Gaussian mixture model
# play with nX
nX=1000
dim=2
rng=[-3,3]
meanList=np.array([[-2,-1],[0,1],[1,0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])

# sample points from Gaussian mixture model
pts=np.zeros((0,dim),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))

nBins=20
binWidth=(rng[1]-rng[0])/nBins
kWidth=binWidth*np.sqrt(dim)
```

```
hist,edgesX,edgesY=np.histogram2d(pts[:,0],pts[:
 ↪,1],bins=nBins,range=[rng,rng],density=True)
# needed for plotting purposes
hist=hist.transpose()

xGrid=rng[0]+(rng[1]-rng[0])*(np.arange(nBins)+0.5)/nBins
XGrid,YGrid=np.meshgrid(xGrid,xGrid)

def getDens(x,y):
    val=np.zeros_like(x)
    for mean,std,weight in zip(meanList,stdList,weightList):
        val+=(1./np.sqrt(2*np.pi)**dim/std**dim)*np.exp(-0.
 ↪5*((x-mean[0])**2+(y-mean[1])**2)/std**2)*weight
    return val

dens=getDens(XGrid.ravel(),YGrid.ravel())
dens=dens.reshape((nBins,nBins))
```

[14]:
```
# Gaussian kernel density estimation
XYGrid=np.stack((XGrid.ravel(),YGrid.ravel())).transpose()
kde = KernelDensity(kernel='gaussian', bandwidth=kWidth).fit(pts)
log_dens = kde.score_samples(XYGrid)
dens_estm=np.exp(log_dens).reshape((nBins,nBins))
```

[15]:
```
dataList=[dens,hist,dens_estm]
labels=["true density","histogram density","kernel density estimation"]
vmax=max([np.max(dat) for dat in dataList])

fig=plt.figure(figsize=(14,4))
for i in range(len(dataList)):
    ax=fig.add_subplot(1,3,i+1)
    img=dataList[i]
    pltobj=plt.imshow(img,extent=rng+rng,origin="lower",vmin=0,vmax=vmax)
    plt.title(labels[i])
    if i==len(dataList)-1:
        fig.colorbar(pltobj, ax=ax)

plt.tight_layout()
plt.show()
```

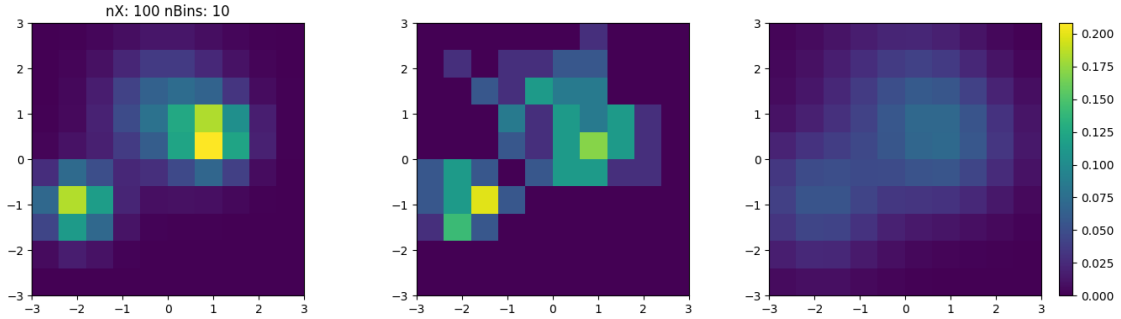


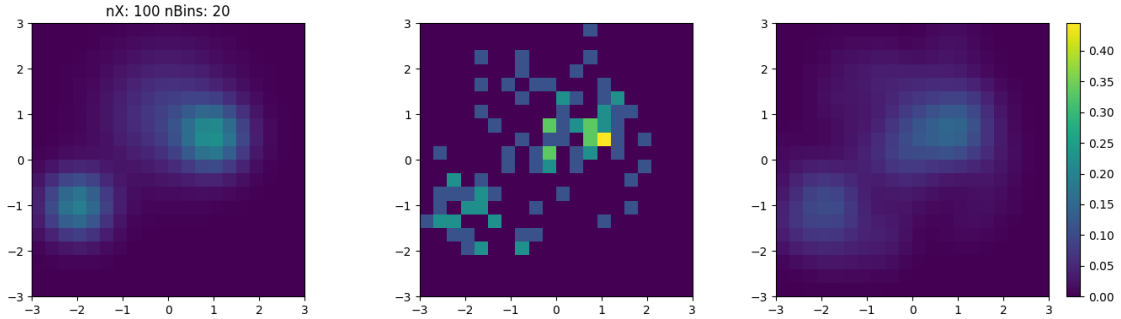
- optional: combine multiple histograms via separate color channels

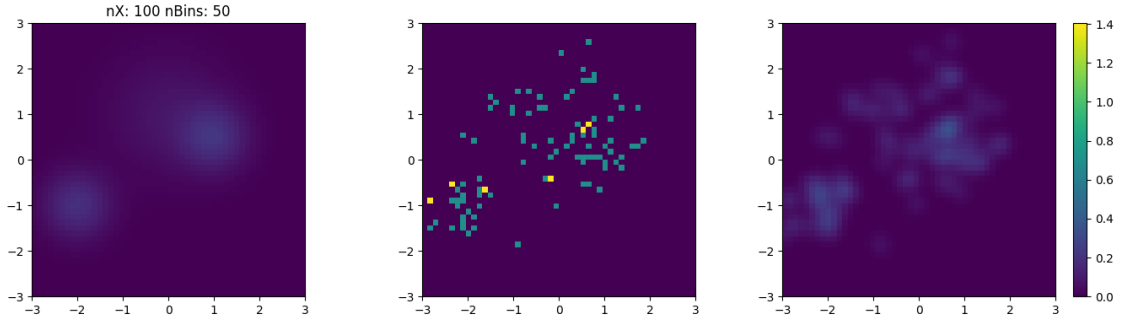
Number of samples, number of bins, kernel width

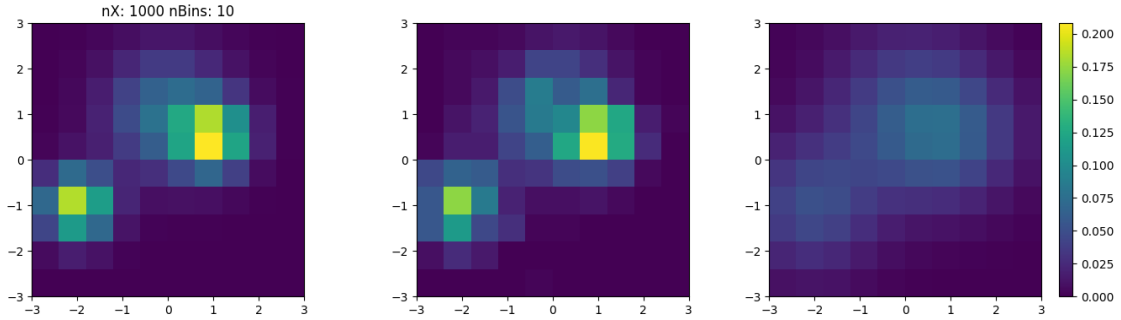
[16]:
```
dim=2
rng=[-3,3]
meanList=np.array([[-2,-1],[0,1],[1,0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])


nXList=[100,1000,10000]
nBinsList=[10,20,50]
for i,nX in enumerate(nXList):
    for i,nBins in enumerate(nBinsList):
        # sample points from Gaussian mixture model
        pts=np.zeros((0,dim),dtype=np.double)
        for mean,std,weight in zip(meanList,stdList,weightList):
            pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))

        binWidth=(rng[1]-rng[0])/nBins
        kWidth=binWidth*np.sqrt(dim)

        hist,edgesX,edgesY=np.histogram2d(pts[:,0],pts[:
 ↪,1],bins=nBins,range=[rng,rng],density=True)
        # needed for plotting purposes
        hist=hist.transpose()

        xGrid=rng[0]+(rng[1]-rng[0])*(np.arange(nBins)+0.5)/nBins
        XGrid,YGrid=np.meshgrid(xGrid,xGrid)

        def getDens(x,y):
            val=np.zeros_like(x)
            for mean,std,weight in zip(meanList,stdList,weightList):
```

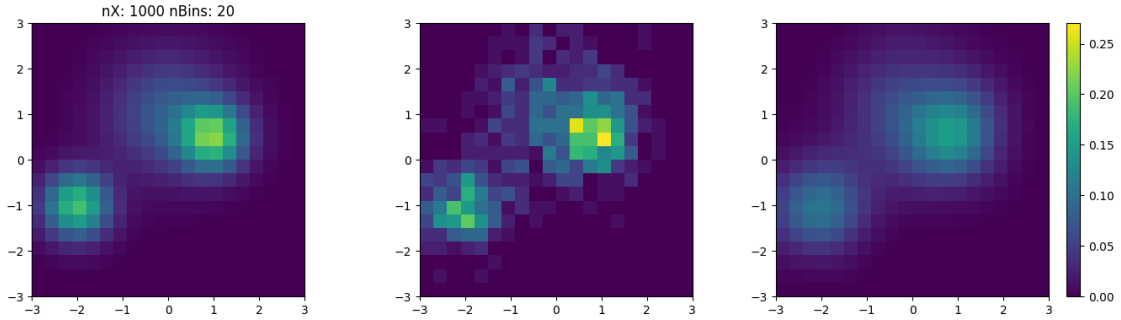
```
                val+=(1./np.sqrt(2*np.pi)**dim/std**dim)*np.exp(-0.
↪5*((x-mean[0])**2+(y-mean[1])**2)/std**2)*weight
            return val

        dens=getDens(XGrid.ravel(),YGrid.ravel())
        dens=dens.reshape((nBins,nBins))

        # Gaussian kernel density estimation
        XYGrid=np.stack((XGrid.ravel(),YGrid.ravel())).transpose()
        kde = KernelDensity(kernel='gaussian', bandwidth=kWidth).fit(pts)
        log_dens = kde.score_samples(XYGrid)
        dens_estm=np.exp(log_dens).reshape((nBins,nBins))


        dataList=[dens,hist,dens_estm]
        labels=["true density","histogram density","kernel density estimation"]
        vmax=max([np.max(dat) for dat in dataList])

        fig=plt.figure(figsize=(14,4))
        for k in range(len(dataList)):
            ax=fig.add_subplot(1,3,k+1)
            img=dataList[k]
            pltobj=plt.
↪imshow(img,extent=rng+rng,origin="lower",vmin=0,vmax=vmax)
            if k==0: plt.title("nX: {:d} nBins: {:d}".format(nX,nBins))
            if k==len(dataList)-1:
                fig.colorbar(pltobj, ax=ax)

        plt.tight_layout()
        plt.show()
```

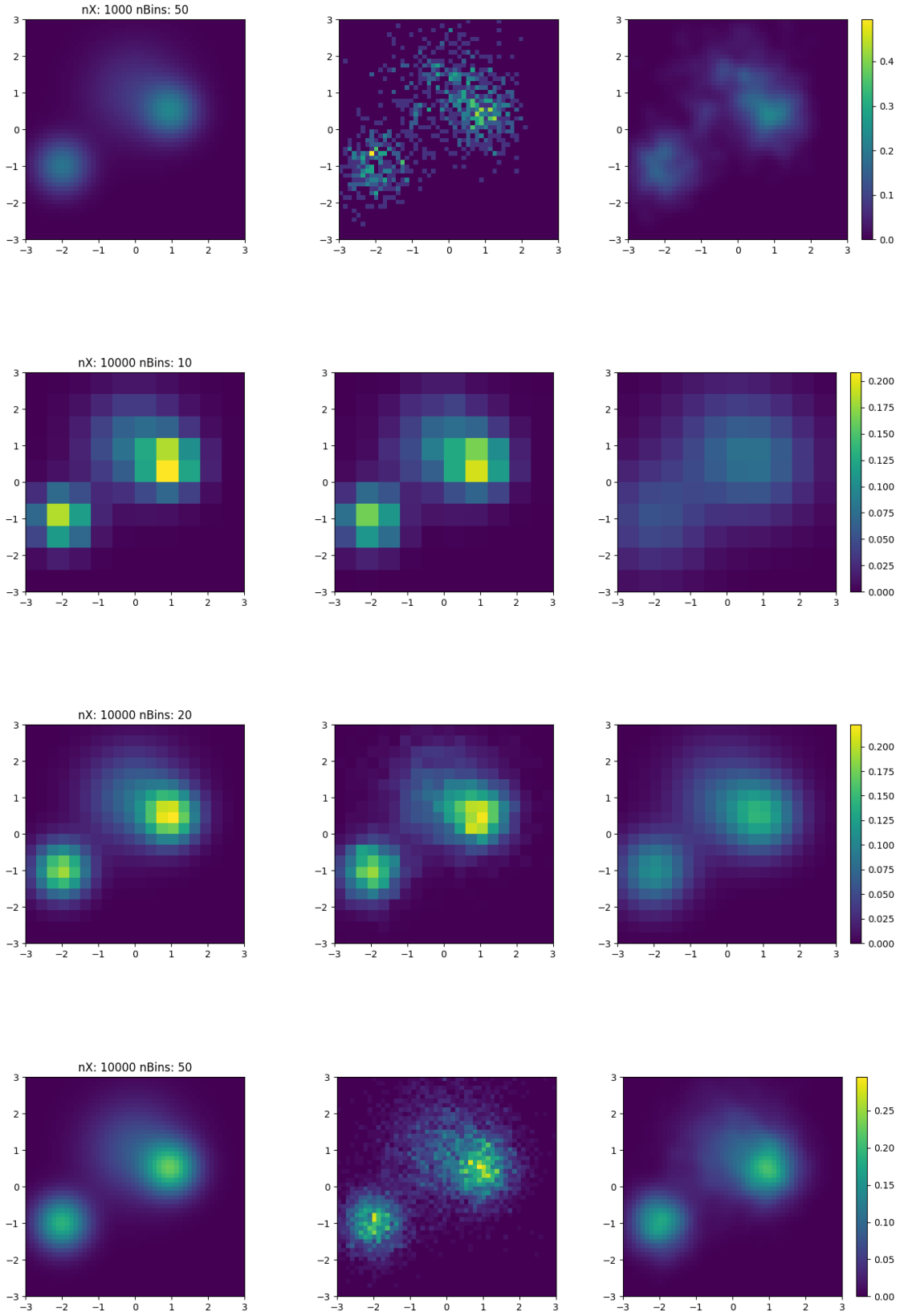



nX: 100 nBins: 20


nX: 100 nBins: 50


nX: 1000 nBins: 10


nX: 1000 nBins: 20

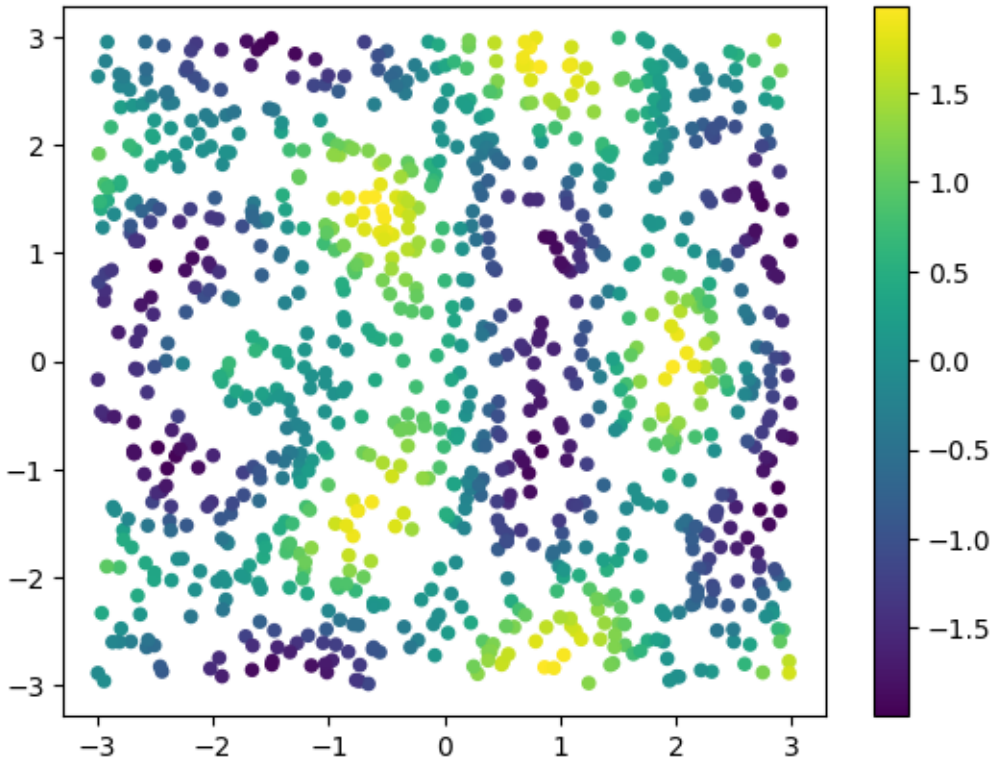

nX: 1000 nBins: 50
nX: 10000 nBins: 10
nX: 10000 nBins: 20
nX: 10000 nBins: 50

1.3 2d: interpolate data from unstructured point cloud

Recall earlier example: function evaluated on unstructured point cloud

- visualized as colored point cloud

[17]:
```
# toy data
nSamples=1000
rng=[-3,3]
# sample x and y coords uniformly from rectangle
x=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
y=rng[0]+(rng[1]-rng[0])*np.random.random(size=nSamples)
# compute f as function of x and y
r1=((x-2)**2+y**2)**.5
r2=((x+2)**2+y**2)**.5
f=np.cos(2*np.pi*r1/3)+np.cos(2*np.pi*r2/2)

# visualize f as color in scatter plot of x and y
fig=plt.figure()
ax=fig.add_subplot()
pltobj=plt.scatter(x,y,c=f,s=20)
fig.colorbar(pltobj, ax=ax)

plt.show()
```

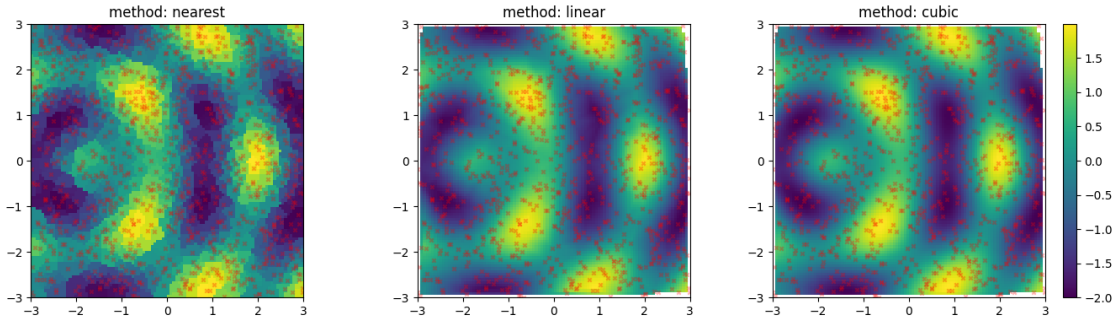


For better visualization: interpolate to a regular grid

- can use built-in scipy function
- show original point locations for reference
- mostly intended for qualitative impression
- or only ”local” interpolation when many samples available

[18]:
```
nPts=100
xGrid=np.linspace(rng[0],rng[1],num=nPts)
yGrid=np.linspace(rng[0],rng[1],num=nPts)
XGrid,YGrid=np.meshgrid(xGrid,yGrid)

fig=plt.figure(figsize=(14,4))
for i,method in enumerate(["nearest","linear","cubic"]):
    fig.add_subplot(1,3,i+1)
    plt.title("method: "+method)
    dat=scipy.interpolate.griddata((x,y),f,(XGrid.ravel(),YGrid.
 ↪ravel()),method=method)
    pltobj=plt.imshow(dat.
 ↪reshape((nPts,nPts)),extent=(rng[0],rng[1],rng[0],rng[1]),origin="lower")
    plt.scatter(x,y,color=(1,0,0,0.3),marker="x",s=10)
    ax=plt.gca()
```

```
fig.colorbar(pltobj, ax=ax)
plt.tight_layout()
plt.show()
```

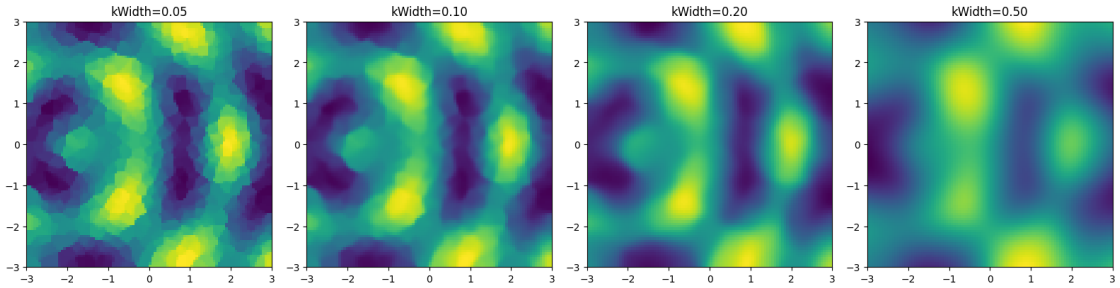


Simple kernel interpolation

[19]:
```
def simpleKernelInterpolatorNd(x,y,z,kWidth):
    # works for x and z being higher-dimensional y: 1d
    # not very computationally efficient, only works on small arrays
    # input empirical data pairs (x,y); interpolate y onto points given by z
    # via Gaussian kernels: at each z sum value of nearby y with Gaussian␣
 ↪weights; normalized to 1
    # use kWidth for Gaussian kernel width
    dim=x.shape[1]
    weights=np.exp(-0.5*np.sum((x.reshape(-1,1,dim)-z.
 ↪reshape(1,-1,dim))**2,axis=2)/kWidth**2)
    signal=np.einsum(weights,[0,1],y,[0],[1])/np.sum(weights,axis=0)
    return signal
```

[20]:
```
# create point cloud of evaluation positions
z=np.zeros((nPts,nPts,2),dtype=np.double)
z[:,:,0]=xGrid.reshape((-1,1))
z[:,:,1]=xGrid.reshape((1,-1))
z=z.reshape((-1,2))
X=np.zeros((x.shape[0],2),dtype=np.double)
X[:,0]=x
X[:,1]=y

kWidthList=[0.05,0.1,0.2,0.5]
fig=plt.figure(figsize=(4*len(kWidthList),4))

for i,kWidth in enumerate(kWidthList):
    img=simpleKernelInterpolatorNd(X,f,z,kWidth=kWidth)
    img=img.reshape((nPts,nPts))
```

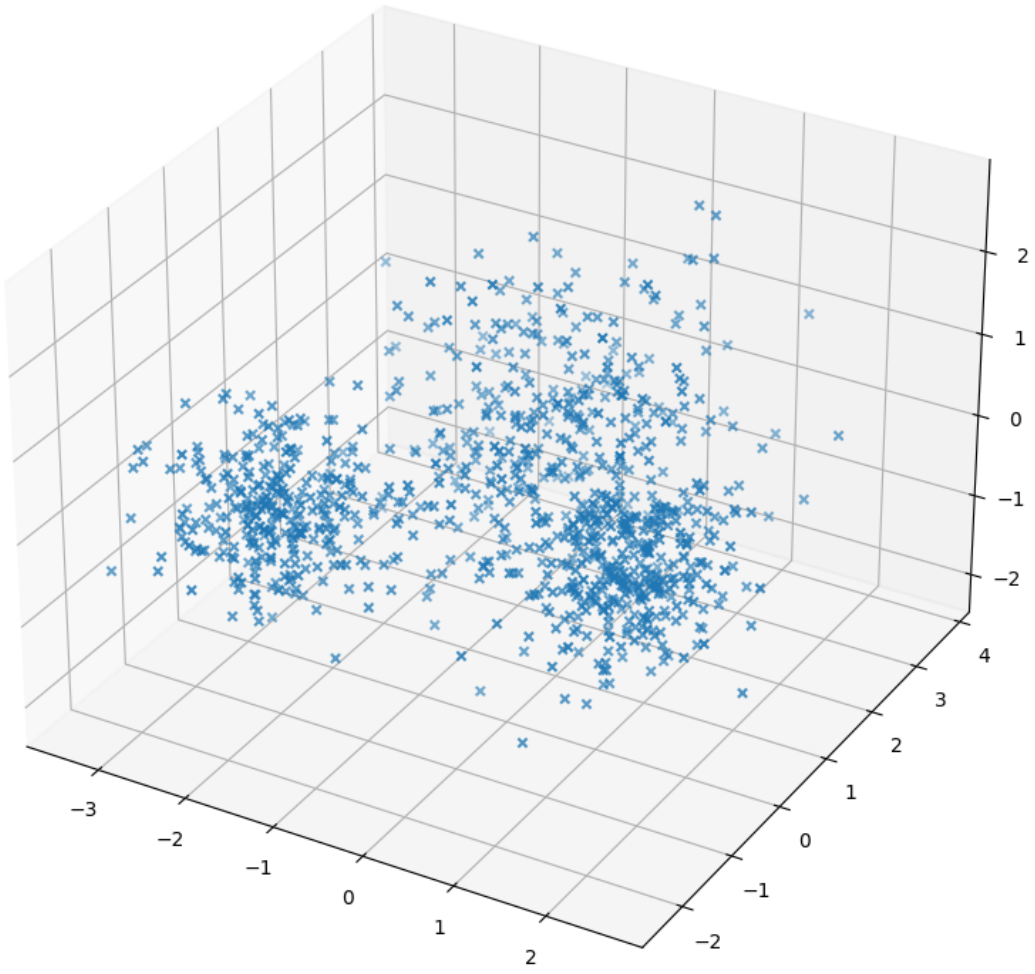
```
    fig.add_subplot(1,len(kWidthList),1+i,aspect=1.)
    plt.imshow(img.
 ↪transpose(),extent=(rng[0],rng[1],rng[0],rng[1]),origin="lower")
    plt.title("kWidth={:.02f}".format(kWidth))
plt.tight_layout()
plt.show()
```



1.4 Proof of concept: 3d histograms

[21]:
```
# 3d Gaussian mixture model
nX=1000
dim=3
rng=[-3,3]
meanList=np.array([[-2,-1,0],[0,1,0.5],[1,0.5,-0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])


# sample points from Gaussian mixture model
pts=np.zeros((0,dim),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))
```

[22]:
```
#%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

ax.scatter(pts[:,0],pts[:,1],pts[:,2],marker="x")
plt.tight_layout()
plt.show()
```



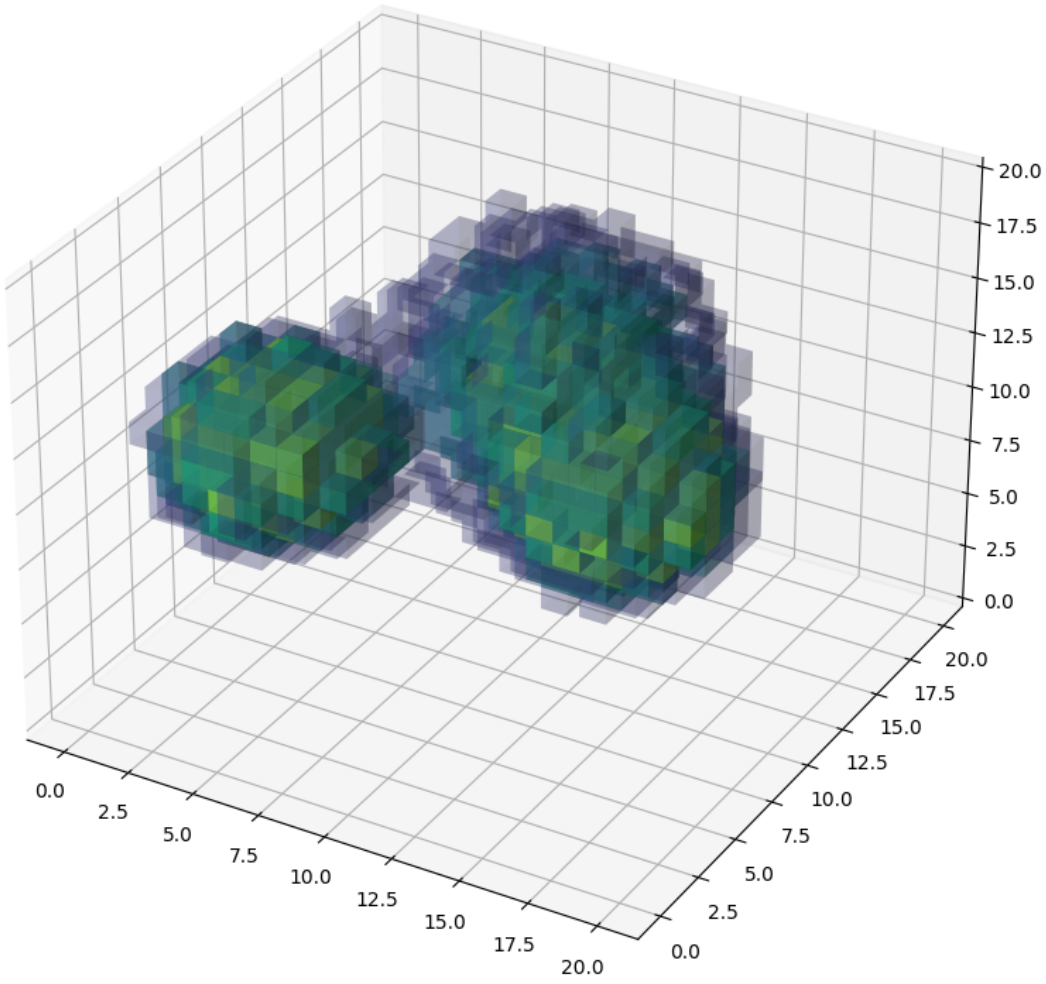
[23]:
```
plt.close()
%matplotlib inline
```

[24]:
```
# 3d Gaussian mixture model
nX=100000
dim=3
rng=[-3,3]
meanList=np.array([[-2,-1,0],[0,1,0.5],[1,0.5,-0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])


# sample points from Gaussian mixture model
```

```
pts=np.zeros((0,dim),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))

nBins=20
binWidth=(rng[1]-rng[0])/nBins
kWidth=binWidth*np.sqrt(dim)

hist,edges=np.histogramdd(pts,bins=nBins,range=[rng,rng,rng])
```

[25]:
```
vmax=np.max(hist)
img=np.minimum(5*hist/vmax,1.)
```

[26]:
```
#%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')

for q in [0.2,0.4,0.6,0.8]:
    ind=img>q
    col=cm.viridis(np.array([q]))[0]
    col[3]=q
    ax.voxels(ind,facecolors=col)
plt.tight_layout()
plt.show()
```



[ ]:
```
plt.close()
%matplotlib inline
```

a few comments

- only works when data is relatively sparse (a few concentrated modes)
- need a color function that involves color and alpha
- as in previous 3d plots: perspective can be problematic on static image
- to be honest: matplotlib far from ideal for this, much better software available!

[ ]:

2023-05-15_ChartTypes_004_StochasticFunctionalRelation

May 25, 2023

[43]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Stochastic functional relation

- In previous scatter plot examples data was often of form $y = f(x)$, show y as function of x
- Maybe with 'a little bit of noise': $y = f(x) + \varepsilon$, in most examples ε was normal distributed. In these cases means and error bars were enough to visualize distribution of y for each x.
- Then went to histograms: visualize distribution of some random y distributed according to some distribution μ
- Now combine the two: Let y be distributed like μ_x where μ_x depends on x (and probably is more complicated than a simple normal distribution), will call this 'stochastic functional relation'. So we need to visualize distribution μ_x of y for each x. How can we do this?
- reminder: usually full μ_x is not available, only sampled data

1.1 Data examples

1.1.1 Data example 1: unimodal relation

$y = \sin(4\pi x) + \sigma_x \cdot z$ where σ_x is a standard deviation, depending on x, and z is normal distributed.

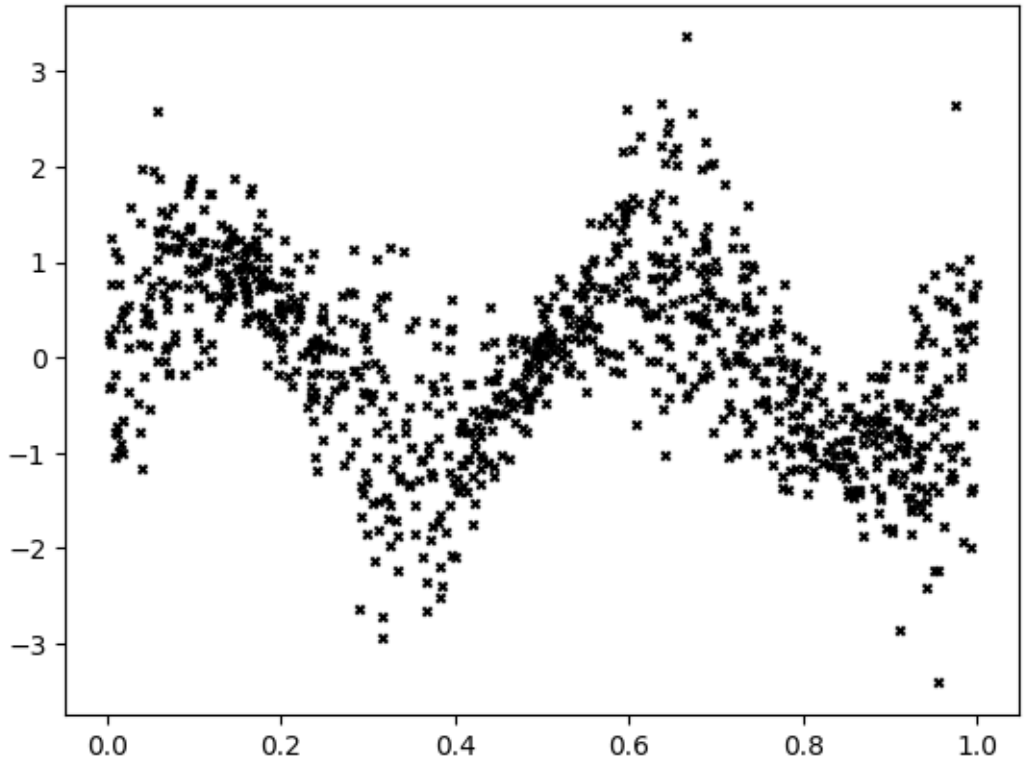
[44]:
```
# nr of points
nP=1000
```

```
# X: uniform sampling from 0 to 1
dataX=np.random.random(size=(nP,))
# Y
dataMean=np.sin(2*np.pi*dataX*2)
dataStd=np.sin(2*np.pi*(dataX+0.5)*1.5)**2*0.7+0.3
dataY=dataMean+dataStd*np.random.normal(size=nP)
```

1.2 Visualization of data

1.2.1 Preview: simple scatter plot

[45]:
```
plt.scatter(dataX,dataY,marker="x",c="k",s=10)
plt.show()
```



1.2.2 Kernel density estimation

- as in histogram chapter, could try and estimate 2d density of (x, y) distribution via kernel density estimation and visualize level lines of this
- something similar will be done on problem sheet, here we proceed with other strategies instead

1.2.3 Binning the data

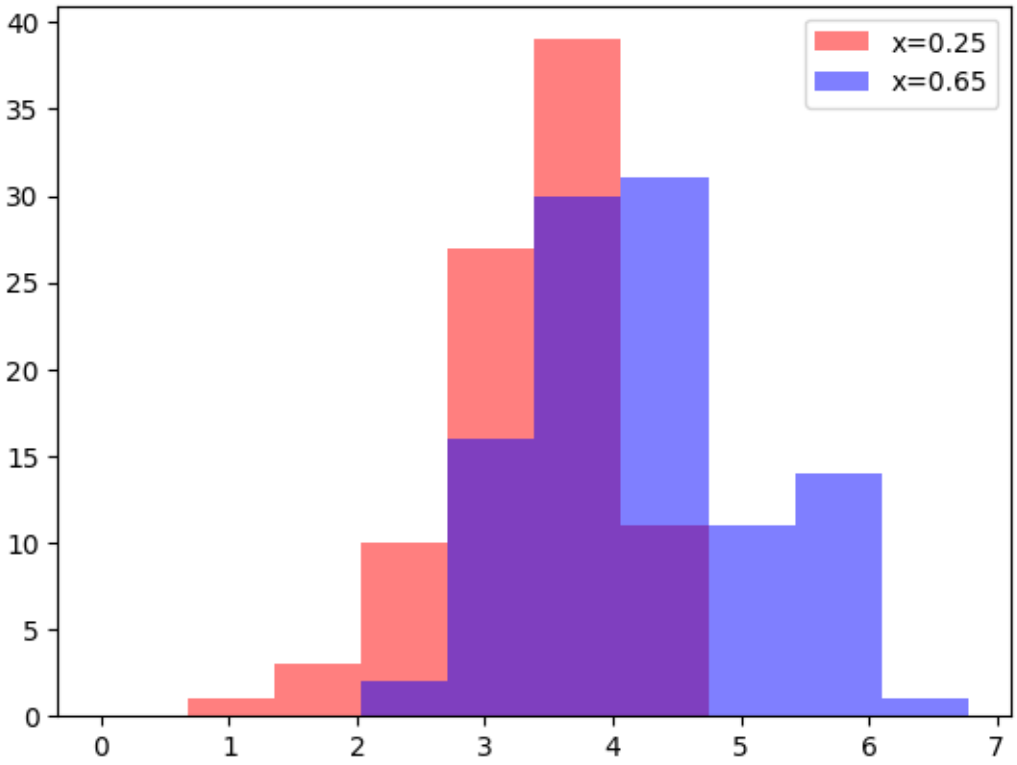
- one strategy is to partition data into bins along x-axis, and try to visualize distribution along y for each bin
- for this, need to bin data first, use simple custom function (will not scale to very large examples)

[46]:
```
def binData(dataX,dataY,nBins):
    """Divide range of dataX values into nBins even sized bins.
    Generate sublists of dataY for the points that lie in each of the bins.
    Returns (midpoints,binnedYLists).
    midpoints: list of midpoints of the X-bins, binnedYLists: list of Y-values␣
 ↪within each bin"""
    vMax=np.max(dataX)
    vMin=np.min(dataX)
    # compute bin nr for each data point, based on x coordinate
    binNr=np.clip(((dataX-vMin)/(vMax-vMin+1E-10)*nBins).astype(np.
 ↪int32),0,nBins-1)
    # generate list of bin midpoints
    midpoints=((np.arange(nBins)+0.5)/nBins)*(vMax-vMin)+vMin
    # create binned YLists, based on binNr array:
    binnedYLists=[[] for i in range(nBins)]
    for b,y in zip(binNr,dataY):
        binnedYLists[b].append(y)
    return midpoints,binnedYLists
```

[47]:
```
# try a low number of bins now, can also experiment with higher numbers if␣
 ↪desired
nBinsX=10
midpointsX,binnedYLists=binData(dataX,dataY,nBinsX)

# compute histograms over y for each x-bin

nBinsY=10

# range of y-data
rng=[np.min(dataY),np.max(dataY)]
histList=[np.histogram(binnedYList,range=rng,bins=nBinsY) for binnedYList in␣
 ↪binnedYLists]
# each entry of this list is of form hist, bin_edges where hist contains the␣
 ↪counts, bin_edges contains
# information about bin boundaries
```

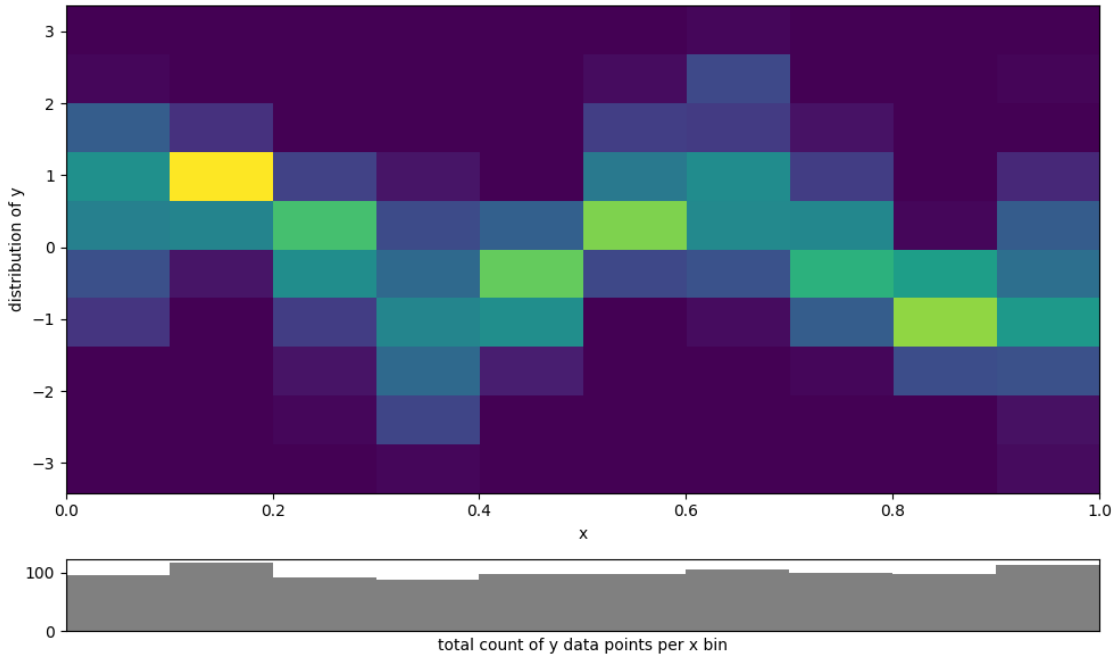
1.2.4 For a first, local impression, plot a few individual histograms

[48]:
```
# use transparency to display multiple histograms as overlapping bar charts

# width and midpoints of bins along Y-axis
deltaX=(rng[1]-rng[0])/nBinsY
midpointsY=deltaX*(np.arange(nBinsY)+0.5)

# colorList
colorList=[(1,0,0,0.5),(0,0,1,0.5)]

for i,iX in enumerate([2,6]):
    plt.
 ↪bar(midpointsY,histList[iX][0],width=deltaX,color=colorList[i],label="x={:.
 ↪02f}".format(midpointsX[iX]))
plt.legend()
plt.show()
```



1.2.5 Try to visualize all histograms for better global impression

[49]:
```
# show all histograms as color coded image
# * we will normalize each xBin to unit mass
# * show absolute count of data points per xBin below in separate plot
# * during generation of the image be careful about axis order and alignments!␣
 ↪(more details later)

xcounts=[len(binnedYList) for binnedYList  in binnedYLists]

img=np.zeros((nBinsY,nBinsX),dtype=np.double)
for iX in range(nBinsX):
    # write each y-histogram into corresponding column of image
    img[:,iX]=histList[iX][0]
    # normalize this colum
    img[:,iX]=img[:,iX]/np.sum(img[:,iX])


fig=plt.figure(figsize=(10,6))

gs = fig.add_gridspec(5, 1)

fig.add_subplot(gs[:-1,:])
plt.imshow(img,extent=(0,1,rng[0],rng[1]),aspect="auto",origin="lower")
plt.xlabel("x")
plt.ylabel("distribution of y")

fig.add_subplot(gs[-1,:])
plt.bar((np.arange(nBinsX)+0.5)/nBinsX,xcounts,width=1/nBinsX,color="#808080")
plt.xlim([0,1])
plt.xticks([])
plt.xlabel("total count of y data points per x bin")
plt.tight_layout()
plt.show()
```



1.2.6 Interlude

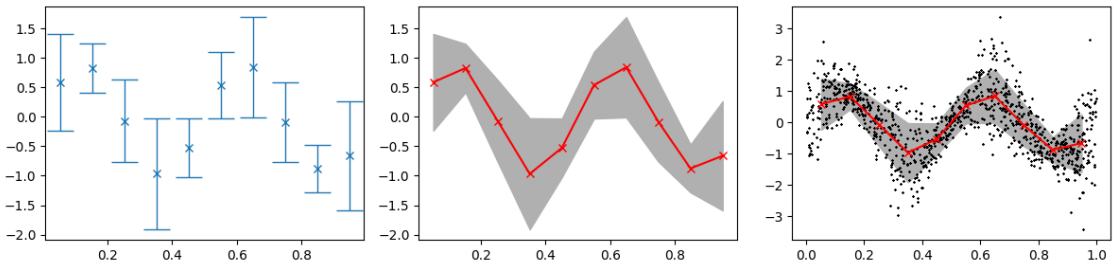
- this encodes most of the information contained in the data
- but it is complicated and color coding is not very quantitative
- in many cases a simpler, but more quantitative representation of the histograms can be shown instead
- now go through a few examples for this

1.2.7 Mean and standard deviation

- simplest version: for each bin show mean and standard deviation
- represent in figure by markers with error bars

[50]:
```
# compute mean and standard variation of y in each bin
ymeans=np.array([np.mean(y) for y in binnedYLists])
ystd=np.array([np.std(y) for y in binnedYLists])

fig=plt.figure(figsize=(12,3))
fig.add_subplot(1,3,1)
# display this as scatter plot with error bars
plt.
  ↪errorbar(midpointsX,ymeans,yerr=ystd,capsize=10,c=colors[0],marker="x",lw=0,elinewidth=1)

# alternative visualization
```

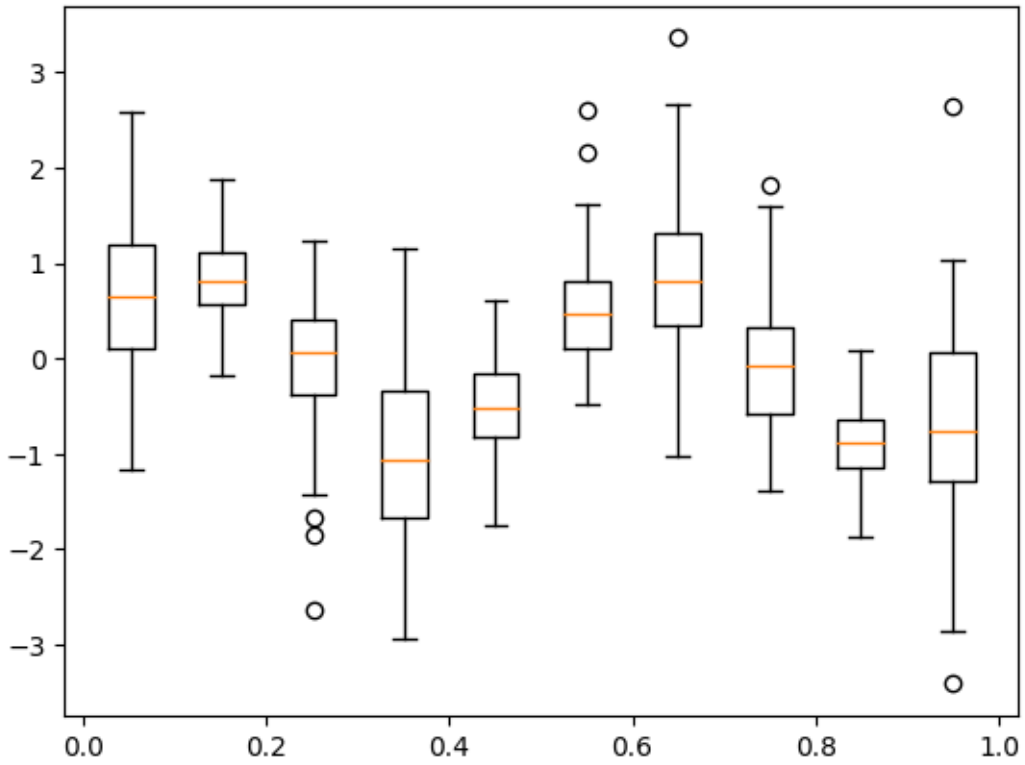
```
fig.add_subplot(1,3,2)
plt.fill_between(midpointsX,ymeans-ystd,ymeans+ystd,color="#b0b0b0",zorder=-1)
plt.plot(midpointsX,ymeans,c="r",marker="x")

# another alternative: add points
fig.add_subplot(1,3,3)
plt.fill_between(midpointsX,ymeans-ystd,ymeans+ystd,color="#b0b0b0",zorder=-1)
plt.plot(midpointsX,ymeans,marker="x",c="r")
plt.scatter(dataX,dataY,marker="x",s=1,c="k")

plt.tight_layout()
plt.show()
```



- mean and standard deviation are really only good representations if the distribution is Gaussian
- this is often true, but definitely not always

1.3 Box plot

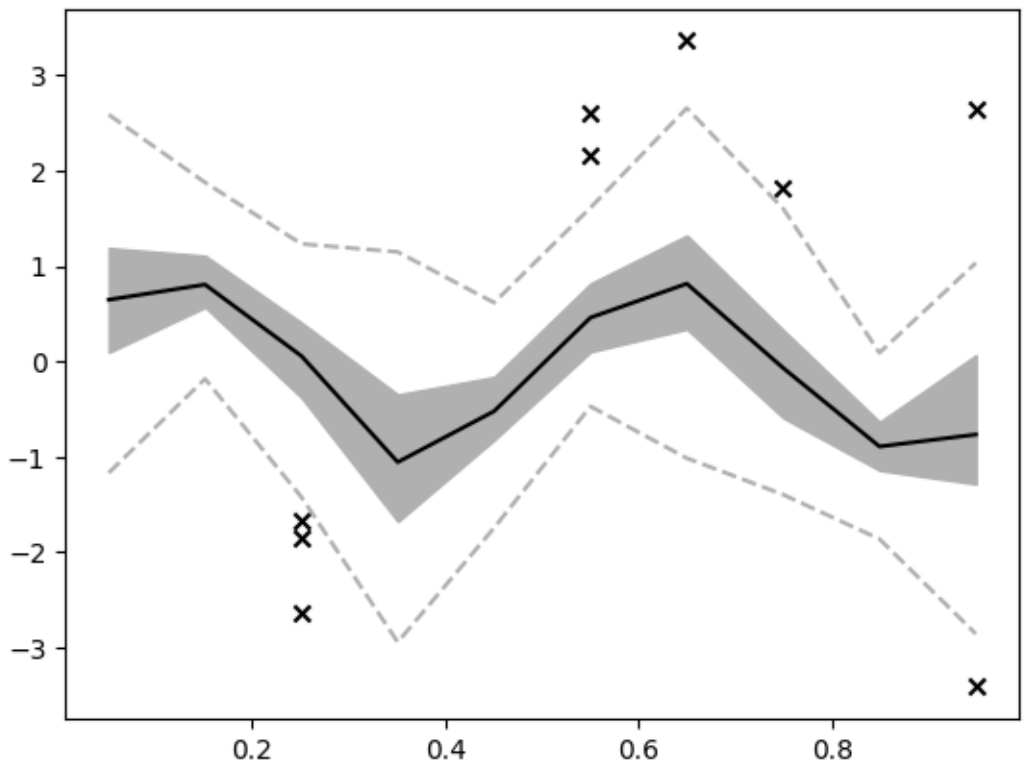
[51]:
```
plt.boxplot(binnedYLists,positions=midpointsX,manage_ticks=False,widths=0.5/
↪nBinsX)
plt.show()
```



[52]:
```
# manual computation of boxplot data

# compute 25,50 and 75 percentiles of each y-distribution
q=np.array([25,50,75])
qDat=np.array([np.percentile(y,q) for y in binnedYLists])
# interquartile range
IQR=qDat[:,2]-qDat[:,0]

# whisker positions
# preliminary position, based on IQR:
wlo=qDat[:,0]-1.5*IQR
whi=qDat[:,2]+1.5*IQR
# now find closest actual data points (from within the whiskers)
for i in range(len(binnedYLists)):
    y=np.array(binnedYLists[i])
    wlo[i]=np.min(y[y>wlo[i]])
    whi[i]=np.max(y[y<whi[i]])


# determine outliers
outliers=[]
for i in range(len(binnedYLists)):
```

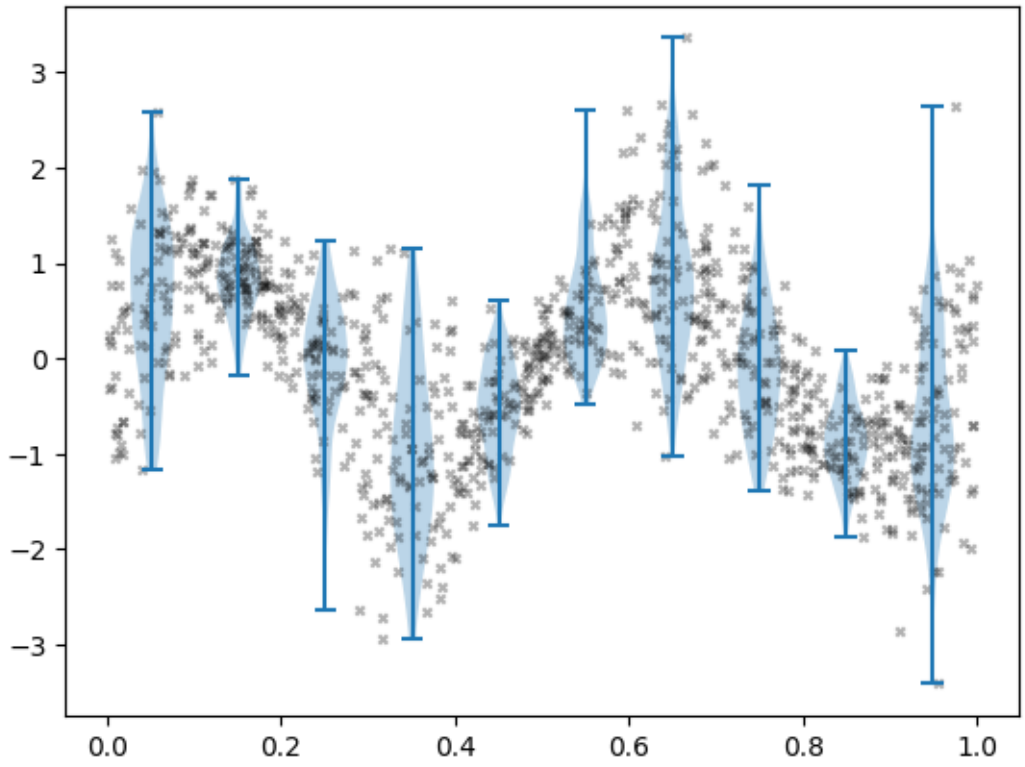
```
    y=np.array(binnedYLists[i])
    dist=np.maximum(y-whi[i],wlo[i]-y)
    y=y[dist>0]
    for z in y:
        outliers.append([midpointsX[i],z])
outliers=np.array(outliers)
```

[53]:
```
# this allows alternative visualization
# which might work better in case of many bins

# median line
plt.plot(midpointsX,qDat[:,1],c="k")
# IQR as filled region
plt.fill_between(midpointsX,qDat[:,2],qDat[:,0],color="#b0b0b0")
# whiskers
plt.plot(midpointsX,wlo,c="#b0b0b0",ls="dashed")
plt.plot(midpointsX,whi,c="#b0b0b0",ls="dashed")
# outliers
if len(outliers)>0:
    plt.scatter(outliers[:,0],outliers[:,1],c="k",marker="x")

plt.show()
```



1.4 Violin plot

[54]:
```
plt.violinplot(binnedYLists,positions=midpointsX,widths=0.5/
  ↪nBinsX,showmedians=False)
plt.scatter(dataX,dataY,marker="x",c="k",s=10,alpha=0.3)
plt.show()
```

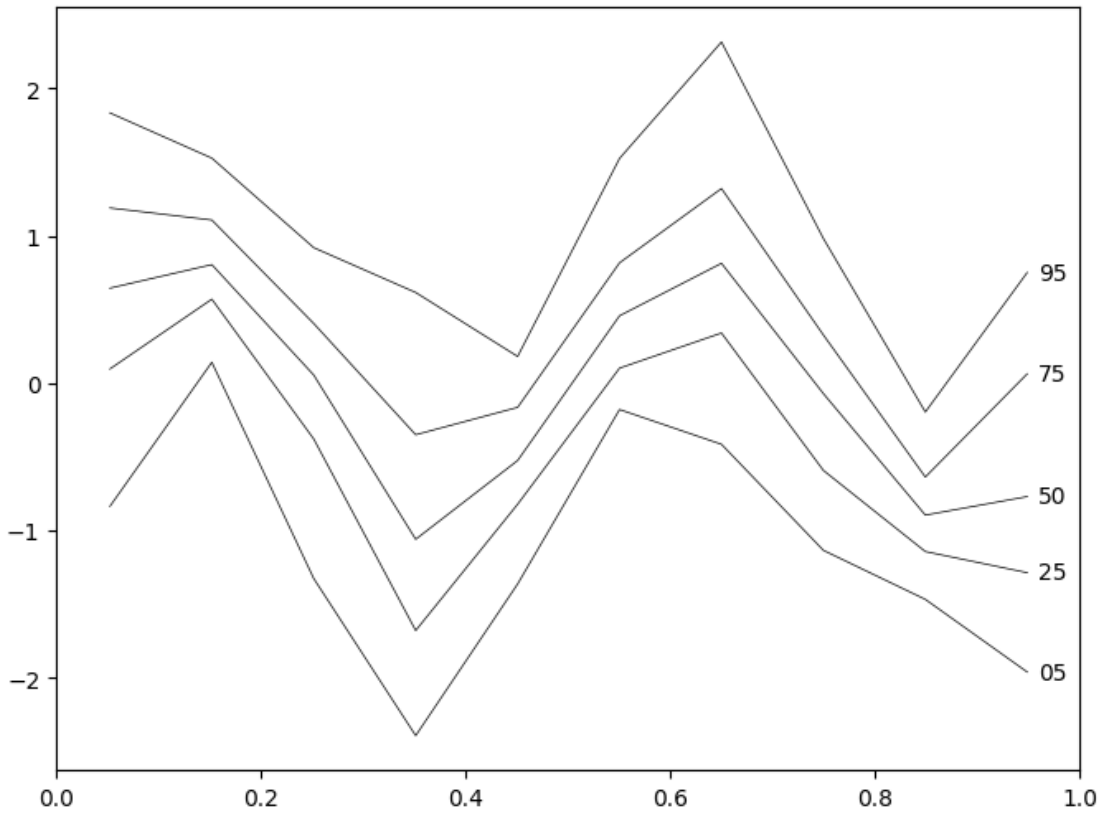


[55]:
```
# hint: check WorldPopulation Project example for 2-sided violin plot
```

1.5 Experiment: plot percentiles

[56]:
```
q=[5,25,50,75,95]
nQ=len(q)
qList=[np.percentile(y,q) for y in binnedYLists]
qList=np.array(qList)

fig=plt.figure(figsize=(8,6))
ax=fig.add_subplot()

for i in range(nQ):
```

```
    ax.plot(midpointsX,qList[:,i],c="k",lw=0.5)
    ax.text(midpointsX[-1]+0.01,qList[-1,i],"{:02d}".
 ↪format(int(q[i])),va="center",ha="left")
plt.xlim([0,1])
plt.show()
```



1.6 More example data

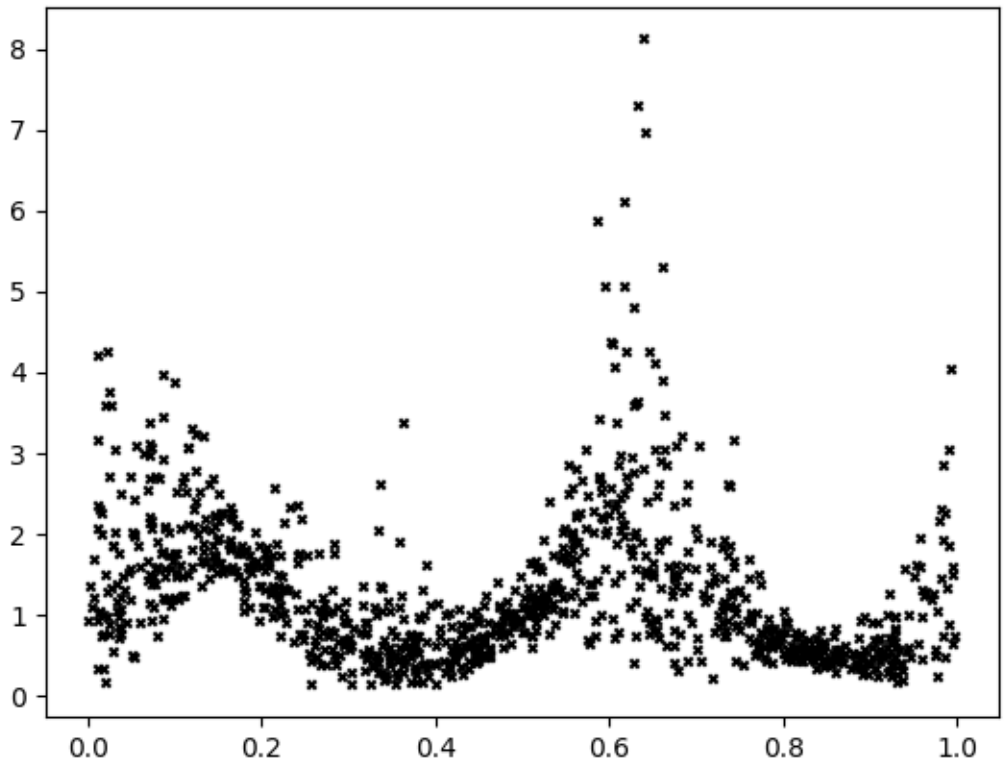
- the two examples below provide more challenging example data for which some simpler visualization methods may fail / be in appropriate
- run the corresponding cells below and then run through the cells in the Visualization section above again to see the effects

1.6.1 Data example 2: Non-linearly distorted Gaussian

[57]:
```
# first generate same data as above

# nr of points
nP=1000
```

```
# X: uniform sampling from 0 to 1
dataX=np.random.random(size=(nP,))
# Y
dataMean=np.sin(2*np.pi*dataX*2)
dataStd=np.sin(2*np.pi*(dataX+0.5)*1.5)**2*0.7+0.3
dataY=dataMean+dataStd*np.random.normal(size=nP)

# now apply non-linear map, this will make distribution asymmetric
dataY=np.exp(0.7*dataY)
```

[58]:
```
plt.scatter(dataX,dataY,marker="x",c="k",s=10)
plt.show()
```

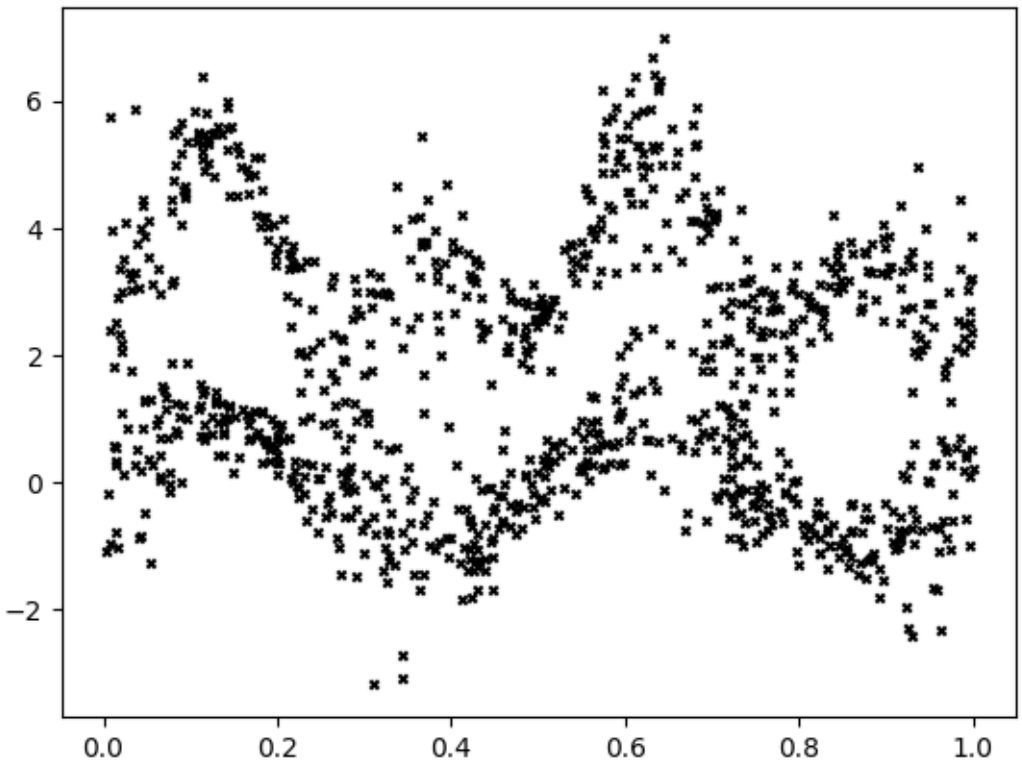


1.6.2 Data example 3: Bimodal distribution

[31]:
```
# first generate same data as above

# nr of points
nP=1000
```

```
# X: uniform sampling from 0 to 1
dataX=np.random.random(size=(nP,))
# Y
dataMean=np.sin(2*np.pi*dataX*2)
dataStd=np.sin(2*np.pi*(dataX+0.5)*1.5)**2*0.7+0.3
dataY=dataMean+dataStd*np.random.normal(size=nP)

# now add random discrete noise, i.e. the data will become bimodal
dataFlips=np.random.randint(0,2,size=(nP,))
dataFlipsSize=np.sin(2*np.pi*(dataX+0.25)*2.)**2*2.+2.5

dataY+=dataFlips*dataFlipsSize
```

[32]:
```
plt.scatter(dataX,dataY,marker="x",c="k",s=10)
plt.show()
```



[ ]:

2023-05-15_ChartTypes_011_VectorFields_Quiver

May 25, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Vector fields

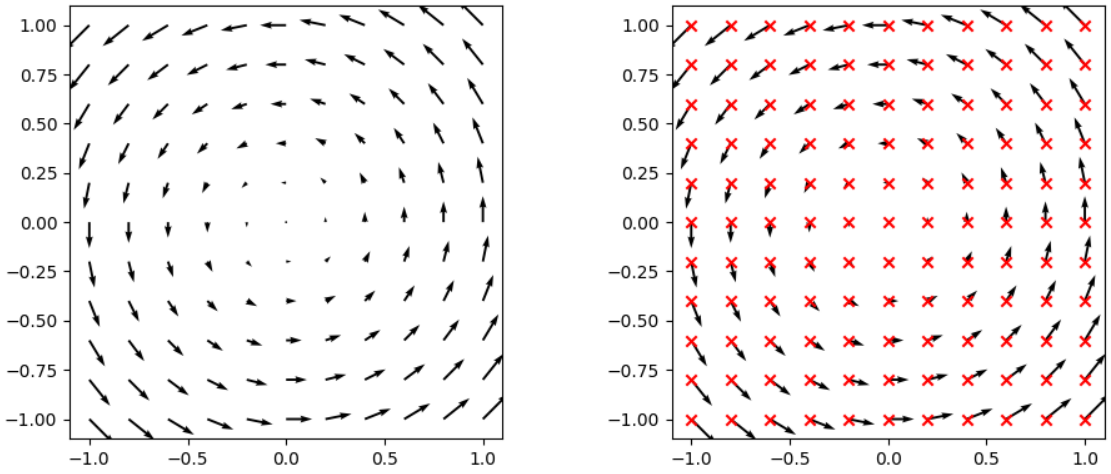
- vector field is function from $\mathbb{R}^m$ to $\mathbb{R}^n$, here focus on $m = n = 2$
- can write it as pair of functions $u(x, y), v(x, y)$ where x, y are horizontal and vertical coordinate in $\mathbb{R}^2$, u and v are horizontal and vertical coordinates of vector field
- think of a small arrow attached to each point in space
- applications:
- velocity fields of air, fluid
- force fields (e.g. from electric charges)
- gradients of functions during optimization

1.1 Quiver plots

[2]:
```
# generate a simple vector field
nPts1d=11
x = np.linspace(-1,1,num=nPts1d)
y = x
nPts=nPts1d**2
X, Y = np.meshgrid(x, y)
u = -Y
v = X
```

1.1.1 Basic example

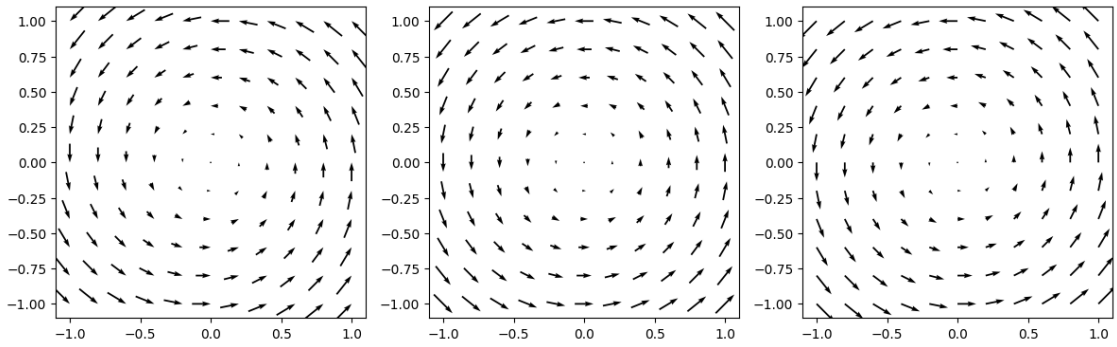
[3]:
```
fig=plt.figure(figsize=(10,4))
# for quiver plot: accurate aspect ratio is important! (we return to this in a␣
 ↪moment)
ax=fig.add_subplot(1,2,1,aspect=1.)
ax.quiver(X,Y,u,v)

# show data points for comparison
ax=fig.add_subplot(1,2,2,aspect=1.)
ax.quiver(X,Y,u,v)
ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")

plt.tight_layout()
plt.show()
```



1.1.2 Pivot of arrows

[4]:
```
# how do we attach arrows to data points? tail, mid, tip?

fig=plt.figure(figsize=(12,4))
ax=fig.add_subplot(1,3,1,aspect=1.)
ax.quiver(X,Y,u,v,pivot="tip")

ax=fig.add_subplot(1,3,2,aspect=1.)
ax.quiver(X,Y,u,v,pivot="mid")
#ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")

ax=fig.add_subplot(1,3,3,aspect=1.)
```

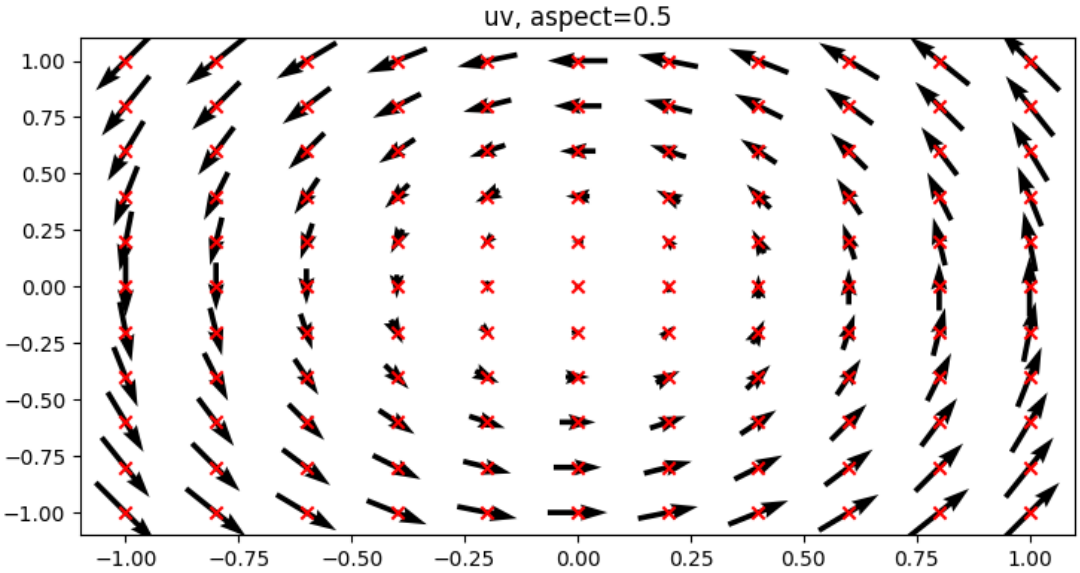
```
ax.quiver(X,Y,u,v,pivot="tail")


plt.tight_layout()
plt.show()
```

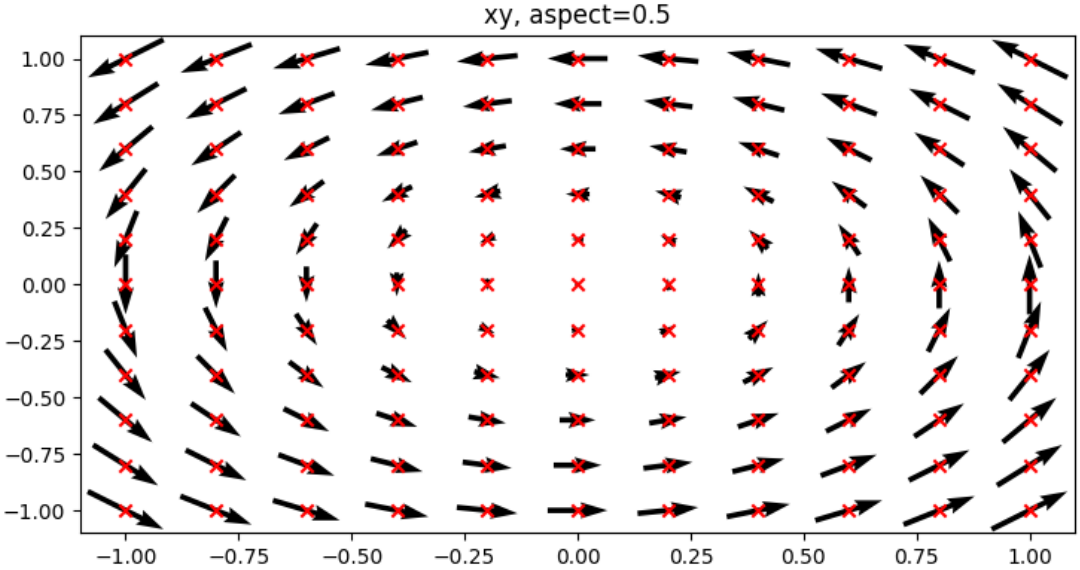


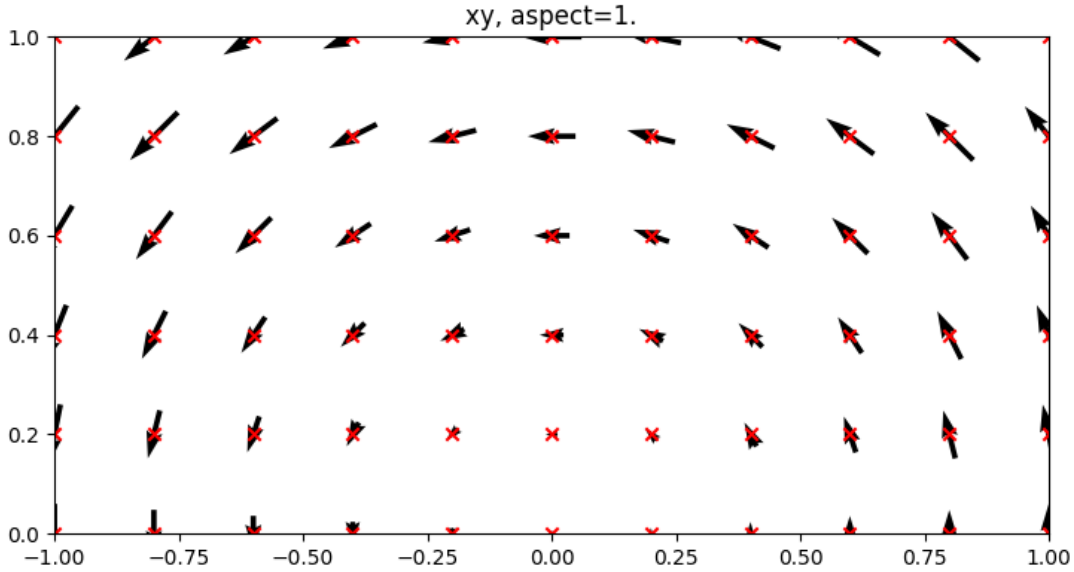
1.1.3 Aspect ratio and angles

[6]:
```
# should arrows be added to data points in "same axis scalings"
#   or in an "independent axis scaling"?
# for aspect ratios != 1, this can have different effects
# my personal opinion: avoid aspect ratio != 1 for quiver plots, since it will
# almost certainly collide with intuitive perception

# angles, orientation

fig=plt.figure(figsize=(8,12))

# uv: arrow orientation is based on aspect ratio 1
# independent of plot coordinate system, i.e. u=v=1 will result in 45° line
# this can be confusing when u,v "live in same coordinate system" as x,y
# e.g. as in a gradient field

ax=fig.add_subplot(3,1,1,aspect=.5)
plt.title("uv, aspect=0.5")
ax.quiver(X,Y,u,v,pivot="mid",angles="uv")
# show data points for comparison
ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")

# xy: arrow orientations are consistent with coordinate axis
# i.e. at each point arrow will point from (x,y) in direction (x+u,y+v)
# so arrow orientation is changed with axis aspect ratio
# can still be confusing if aspect ratio != 1
```

```
ax=fig.add_subplot(3,1,2,aspect=.5)
plt.title("xy, aspect=0.5")
ax.quiver(X,Y,u,v,pivot="mid",angles="xy")
# show data points for comparison
ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")

ax=fig.add_subplot(3,1,3,aspect=1.)
plt.title("xy, aspect=1.")
ax.quiver(X,Y,u,v,pivot="mid",angles="xy")
# show data points for comparison
ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")
ax.set_ylim([0.,1.])
ax.set_xlim([-1.,1.])


plt.tight_layout()
plt.show()
```

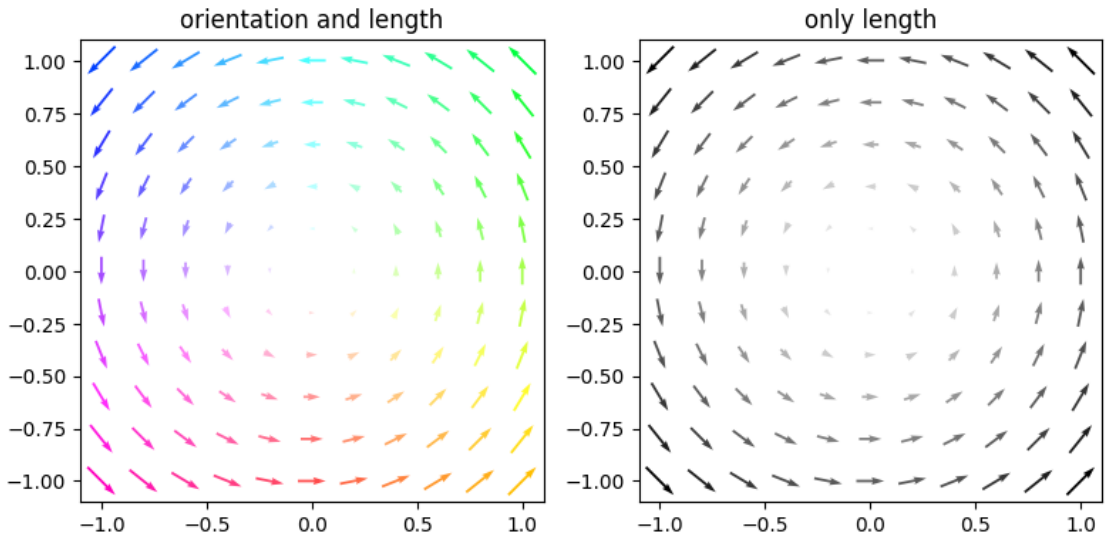

uv, aspect=0.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


xy, aspect=0.5
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


xy, aspect=1.
1.0
0.8
0.6
0.4
0.2
0.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

1.1.4 Color

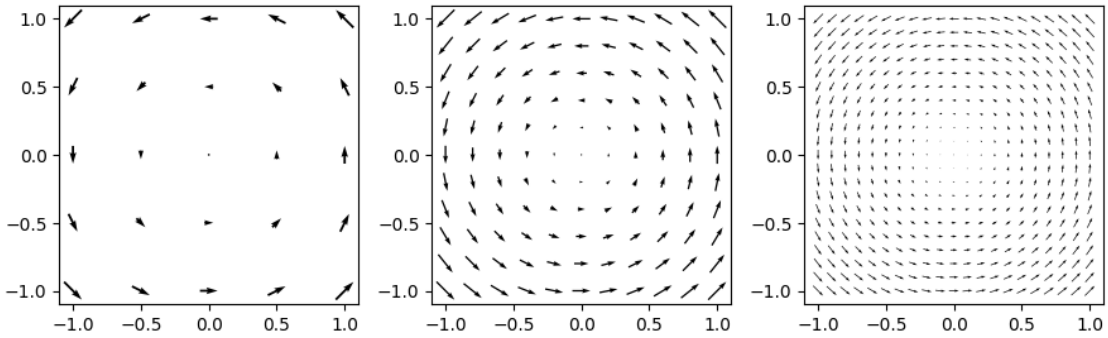
[7]:
```
# additional degree of freedom: color; use with care
# experiment: how about orientation as hue, length as saturation

# creat empty HSV array
colorsHSV=np.zeros((nPts,3))
# set Saturation to 1
colorsHSV[:,2]=1.
# set Value to (rescaled) magnitude
magnitude=((u**2+v**2)**0.5).ravel()
vmax=np.max(magnitude)
colorsHSV[:,1]=np.clip(magnitude/vmax,0.,1.)
# set Hue to orientation
phi=np.mod(np.arctan2(v,u).ravel()/2/np.pi,1.)
colorsHSV[:,0]=phi
# conver to RGB
colorsRGB=matplotlib.colors.hsv_to_rgb(colorsHSV)

# just visualize magnitude
colorsRGB2=cm.Greys(magnitude/vmax)
#colorsRGB2=cm.Reds(magnitude/vmax)
# use a colormap with 0=white

# elegant side effect of this coloring scheme: zero length arrows become␣
 ↪invisible,
# since color merges into background color


fig=plt.figure(figsize=(8,5))

ax=fig.add_subplot(1,2,1,aspect=1.)
ax.quiver(X,Y,u,v,color=colorsRGB,pivot="mid")
plt.title("orientation and length")

ax=fig.add_subplot(1,2,2,aspect=1.)
ax.quiver(X,Y,u,v,color=colorsRGB2,pivot="mid")
plt.title("only length")

plt.tight_layout()
plt.show()
```



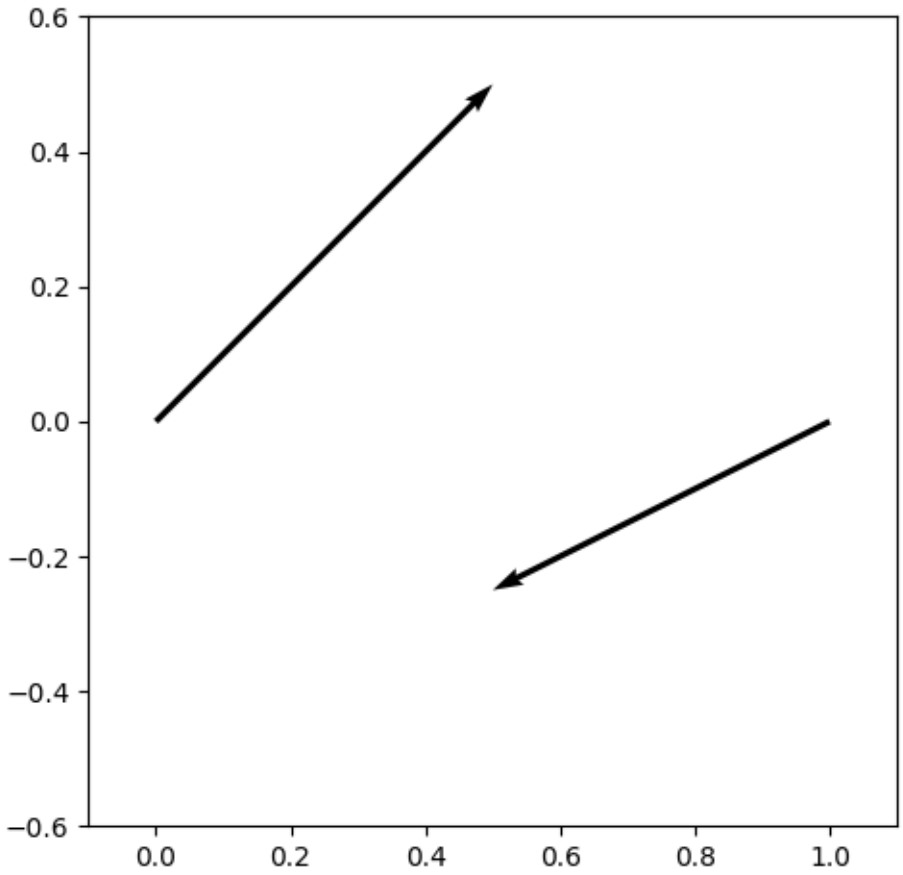
1.1.5 Number and length of arrows

[8]:
```
# generate a simple vector field at different resolutions
nPts1dList=[5,11,21]
nPlots=len(nPts1dList)

fig=plt.figure(figsize=(3*nPlots,3))


for i, nPts1d in enumerate(nPts1dList):
    x = np.linspace(-1,1,num=nPts1d)
    y = x
    nPts=nPts1d**2
    X, Y = np.meshgrid(x, y)
    u = -Y
    v = X

    ax=fig.add_subplot(1,nPlots,i+1,aspect=1.)
    ax.quiver(X,Y,u,v,pivot="mid")
    # show data points for comparison
    #ax.scatter(X.ravel(),Y.ravel(),marker="x",c="r")

plt.tight_layout()
plt.show()

# obvious trade-off:
# * more arrows: better spatial resolution
```

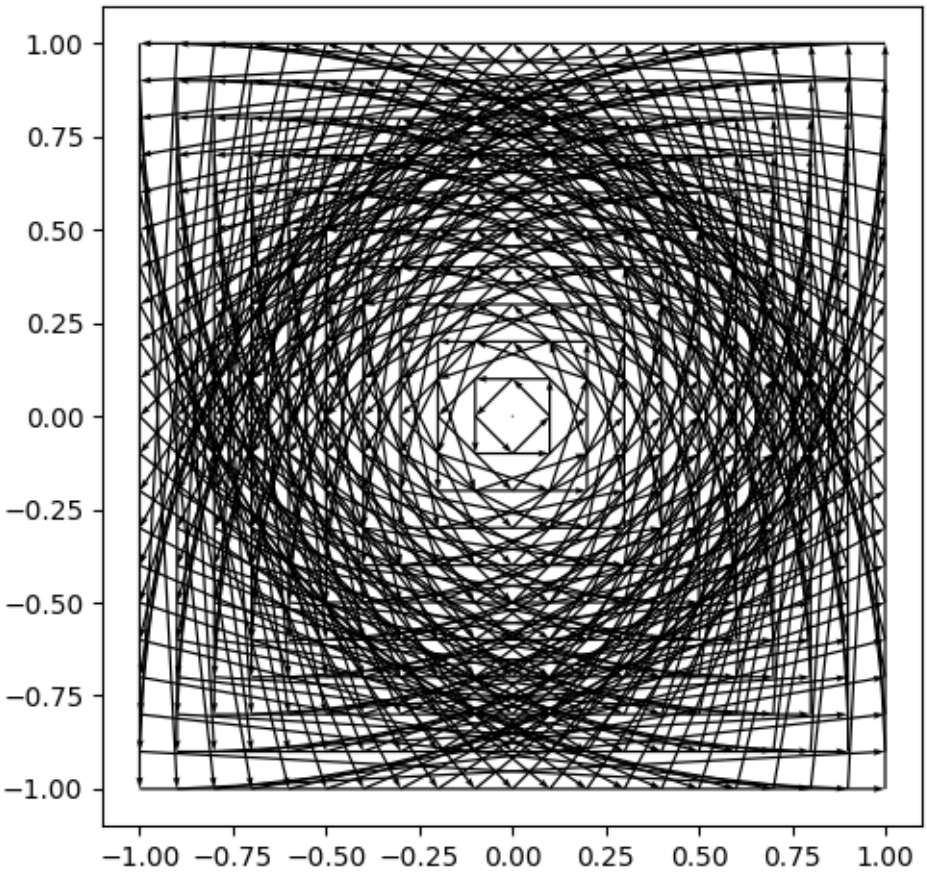
```
# * with too many arrows the plot becomes hard to read
```



[9]:
```
# length of the arrows:
# without specific instructions uses a simple auto-scale algorithm to produce
# a nice quiver plot with arrows not overlapping each other
# so the arrow lengths that we give to quiver are not exactly
# the lengths of the drawn arrows

# in principle vector field might encode function \R^2 \to \R^2
# then f(x,y)=(x+u,y+v)
# plotting arrows that literally go from (x,y) to f(x,y) will almost
# always look chaotic for large deformations
# still: sometimes we may want to plot arrows of exact length
# this can be done as follows


fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.quiver([0,1],[0,0],[0.5,-0.5],[0.5,-0.25],\
        pivot="tail",angles="xy",scale_units="xy",scale=1.)
plt.xlim([-0.1,1.1])
plt.ylim([-0.6,0.6])
plt.tight_layout()
plt.show()
```



[10]:
```
# using this to visualize maps from R^2 to R^2 as explicit arrows is almost
# always not working (if one has large displacements)
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.quiver(X,Y,u-X,v-Y,pivot="tail",angles="xy",scale_units="xy",scale=1.)
plt.tight_layout()
plt.show()
# we will return to this challenge with different ideas a bit later
```


1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−1.00 −0.75 −0.50 −0.25 0.00 0.25 0.50 0.75 1.00

[ ]:

[ ]:

[ ]:

2023-05-15_ChartTypes_012_VectorFields_DeformationImages

May 25, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Vector fields

1.1 Deformation plots

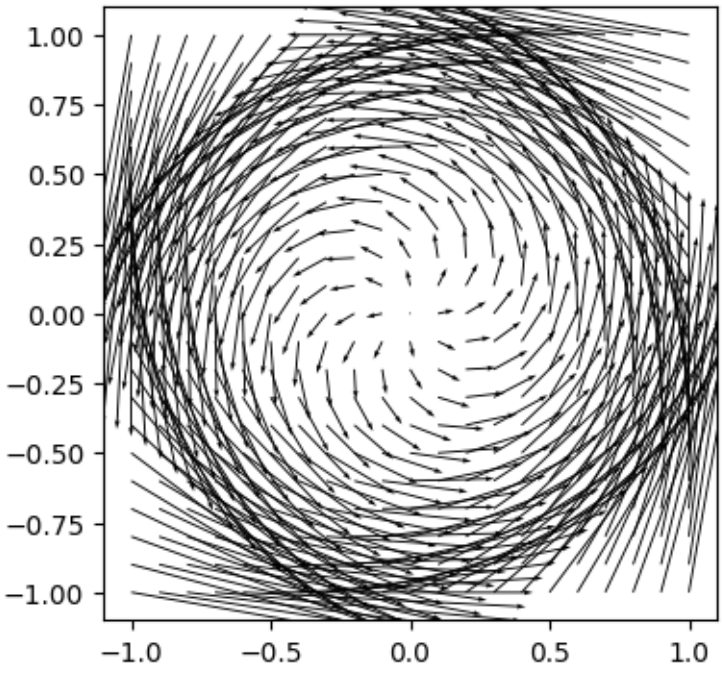
[2]:
```
# build some non-trivial examples of deformations for us to visualize
def map1(x,y):
    """merely rescale radius a bit"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map1inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn
    u=np.cos(phiOut)*rOut
```

```
    v=np.sin(phiOut)*rOut
    return u,v


def map2(x,y):
    """rescale radius + fixed rotation"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn+0.25*np.pi
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map2inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn-0.25*np.pi
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v


def map3(x,y):
    """rescale radius + adaptive rotation"""
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**0.8
    phiOut=phiIn+0.25*np.pi*rIn
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v

def map3inv(x,y):
    rIn=np.sqrt(x**2+y**2)
    phiIn=np.arctan2(y,x)
    rOut=rIn**(1/0.8)
    phiOut=phiIn-0.25*np.pi*rOut
    u=np.cos(phiOut)*rOut
    v=np.sin(phiOut)*rOut
    return u,v


usemap=map3
usemapinv=map3inv
```

```
def getUniformData(nPts1d,rng,usemap):
    x = np.linspace(*rng,num=nPts1d)
    X, Y = np.meshgrid(x, x)
    U, V = usemap(X,Y)
    return X,Y,U,V
```

1.1.1 Visualization as quiver plot

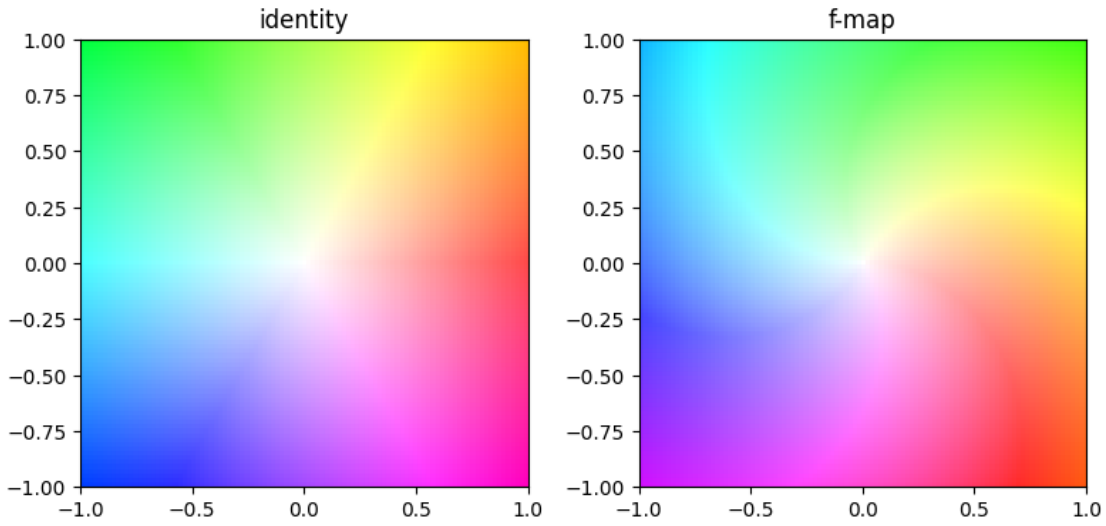
[3]:
```
X,Y,U,V=getUniformData(21,[-1,1],usemap)

fig=plt.figure(figsize=(4,4))
ax=fig.add_subplot(aspect=1.)
ax.quiver(X,Y,U-X,V-Y,pivot="tail",angles="xy",scale_units="xy",scale=1.)
plt.tight_layout()
plt.show()

# shown earlier: this tends not to work well on actual maps with long vectors
```



1.1.2 Visualization as HSV image

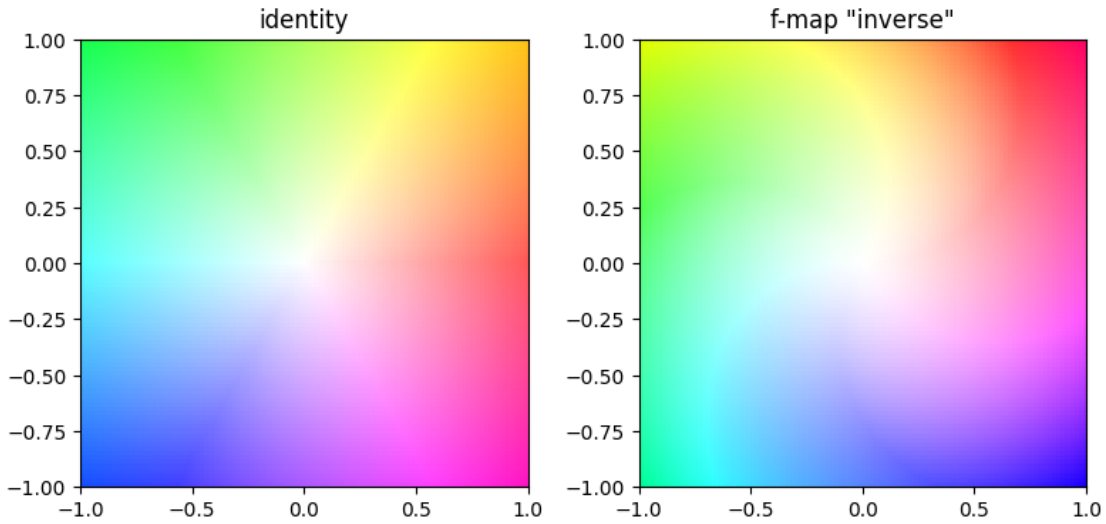
[4]:
```
def HSVfromUV(U,V,magmax=None):
    colorsHSV=np.zeros(U.shape+(3,))
    # set Saturation to 1
    colorsHSV[:,:,2]=1.
    # set Value to (rescaled) magnitude
    magnitude=(U**2+V**2)**0.5
    if magmax is None:
        vmax=np.max(magnitude)
    else:
        vmax=magmax
    colorsHSV[:,:,1]=np.clip(magnitude/vmax,0.,1.)
    # set Hue to orientation
    phi=np.mod(np.arctan2(V,U)/(2*np.pi),1.)
    colorsHSV[:,:,0]=phi
    # conver to RGB
    colorsRGB=matplotlib.colors.hsv_to_rgb(colorsHSV)
    return colorsRGB
```

[5]:
```
nPts1d=101
rng=[-1,1]
X,Y,U,V=getUniformData(nPts1d,rng,usemap)

magmax=max(np.max((U**2+V**2)**0.5),np.max((X**2+Y**2)**0.5))

img1=HSVfromUV(X,Y,magmax)
# try: color each pixel by color corresponding to target position
img2=HSVfromUV(U,V,magmax)

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
plt.imshow(img2,extent=rng+rng,origin="lower")

plt.tight_layout()
plt.show()
```



[6]:
```
# comments:
# * see how perceptually non-uniform HSV is!
# * in this way, displayed side-by-side like this, it seems like map is
#   a rotation in the opposite direction!
# -> we should color each pixel by the color corresponding to its preimage
```

[7]:
```
nPts1d=101
rng=[-1,1]
X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)

magmax=max(np.max((U**2+V**2)**0.5),np.max((X**2+Y**2)**0.5))

img1=HSVfromUV(X,Y,magmax)
# try: color each pixel by color corresponding to target position
img3=HSVfromUV(U,V,magmax)

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map \"inverse\"")
plt.imshow(img3,extent=rng+rng,origin="lower")

plt.tight_layout()

plt.show()
```

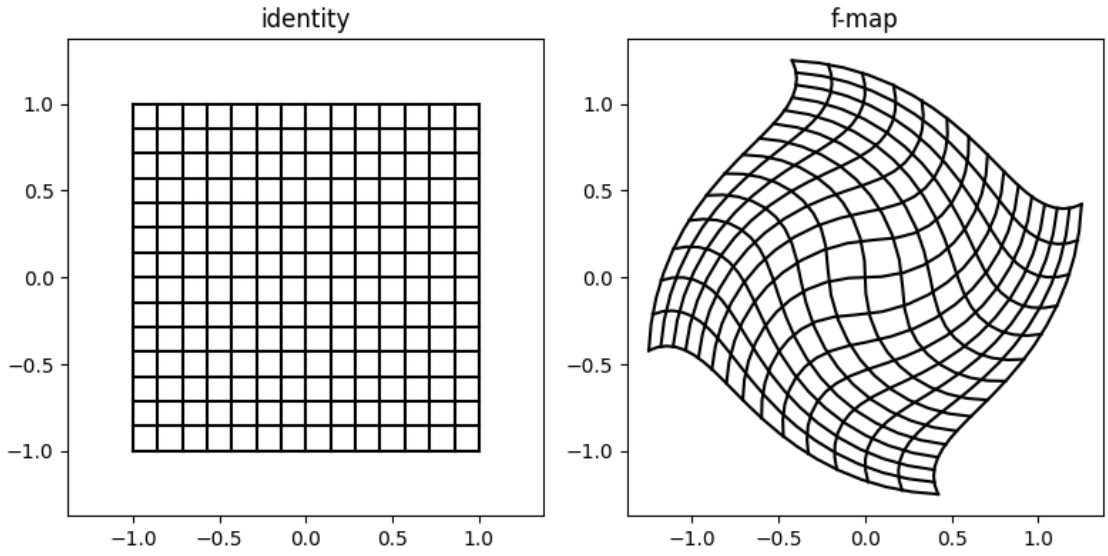


1.1.3 Visualization as warped grid

[8]:
```
# generate data for evaluating the map on a nice regular grid
# make spacings in X and Y direction very different, so that we can get
# good visualization as warped grid (will become clear in a moment)
def getLines(rng,nLines,nPtsPerLine):
    x=np.linspace(*rng,num=nLines)
    y=np.linspace(*rng,num=nPtsPerLine)
    X, Y = np.meshgrid(x, y)
    X=X.transpose()
    Y=Y.transpose()
    return np.concatenate((X,Y)),np.concatenate((Y,X))
```

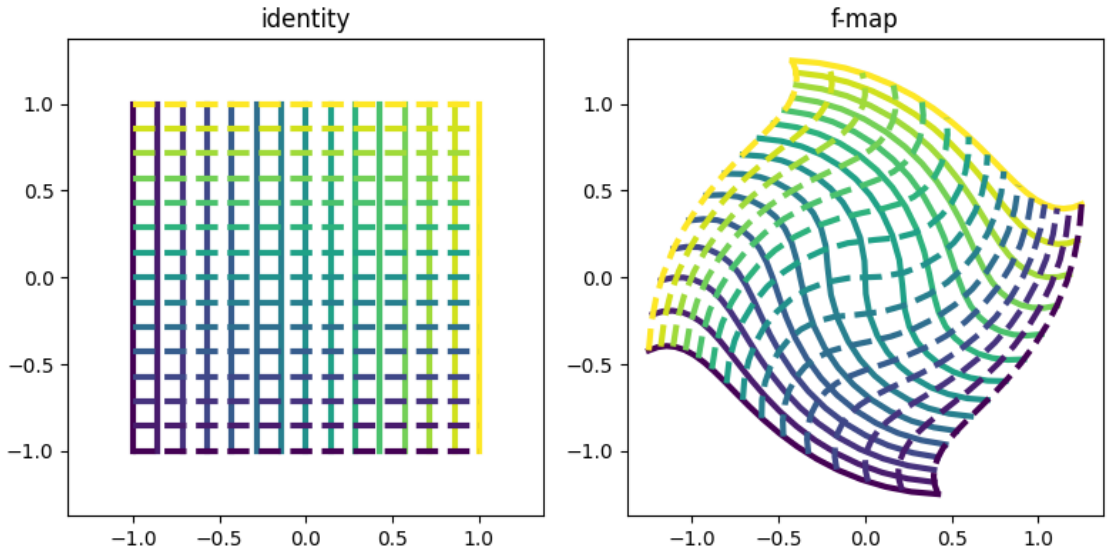
[9]:
```
rng=[-1,1]
nLines=15
nPtsPerLine=20
X,Y=getLines(rng,nLines,nPtsPerLine)
U,V=usemap(X,Y)

# get consistent plot range in both images
magmax=max([np.max(np.abs(Z)) for Z in [X,Y,U,V]])*1.1


fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
for x,y in zip(X,Y):
```

```
    plt.plot(x,y,c="k")
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
for x,y in zip(U,V):
    plt.plot(x,y,c="k")
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

plt.tight_layout()
plt.show()
```



[10]:
```
# experiment: add color to lines to make identification a bit more obvious in
#   ambiguous cases
# try also: line style (solid vs dashed)
# make lines a bit thicker, so that these features (color, dash pattern)
#   can be seen better

colors=np.linspace(0,1,num=nLines)
colors=np.concatenate((colors,colors))
linestyle=["solid" for _ in range(nLines)]+["dashed" for _ in range(nLines)]

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
```

```
for x,y,c,ls in zip(X,Y,colors,linestyle):
    plt.plot(x,y,c=cm.viridis(c),ls=ls,lw=3)
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map")
for x,y,c,ls in zip(U,V,colors,linestyle):
    plt.plot(x,y,c=cm.viridis(c),ls=ls,lw=3)
plt.xlim([-magmax,magmax])
plt.ylim([-magmax,magmax])

plt.tight_layout()
plt.show()
```

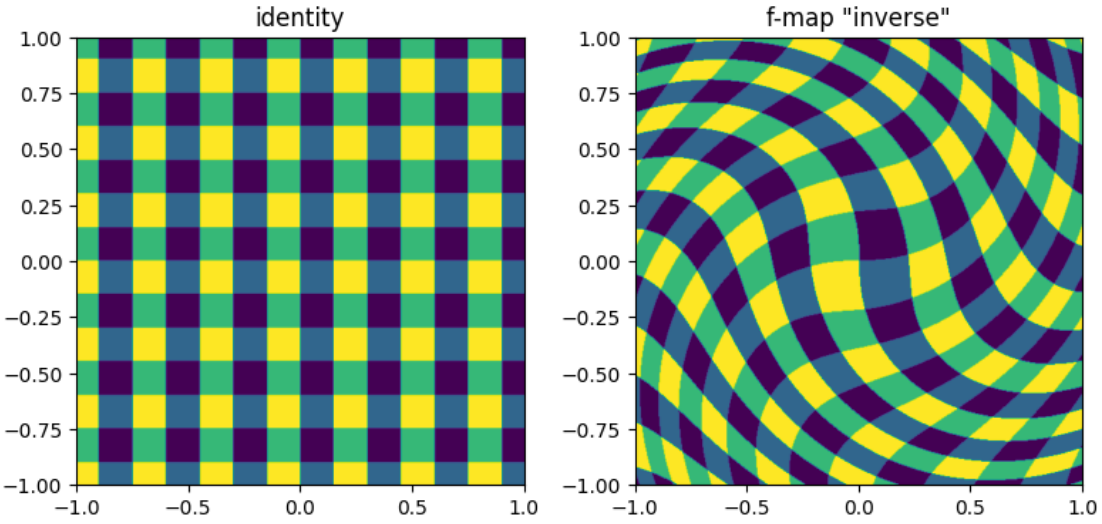


1.1.4 Visualization as warped grid, variant 2

This is a fusion of the two previous approaches. Visualize deformation as colored image, but instead of encoding full position as color, only encode position in a grid.

[11]:
```
def colorFromPos(X,Y,res):
    # divide coords into square grid of edge length res
    # determine color of "which square" pixel is in
    indexX=np.clip(np.floor(np.mod((X/res),2)),0,1)
    indexY=np.clip(np.floor(np.mod((Y/res),2)),0,1)
    return indexX*0.67+indexY*0.33
```

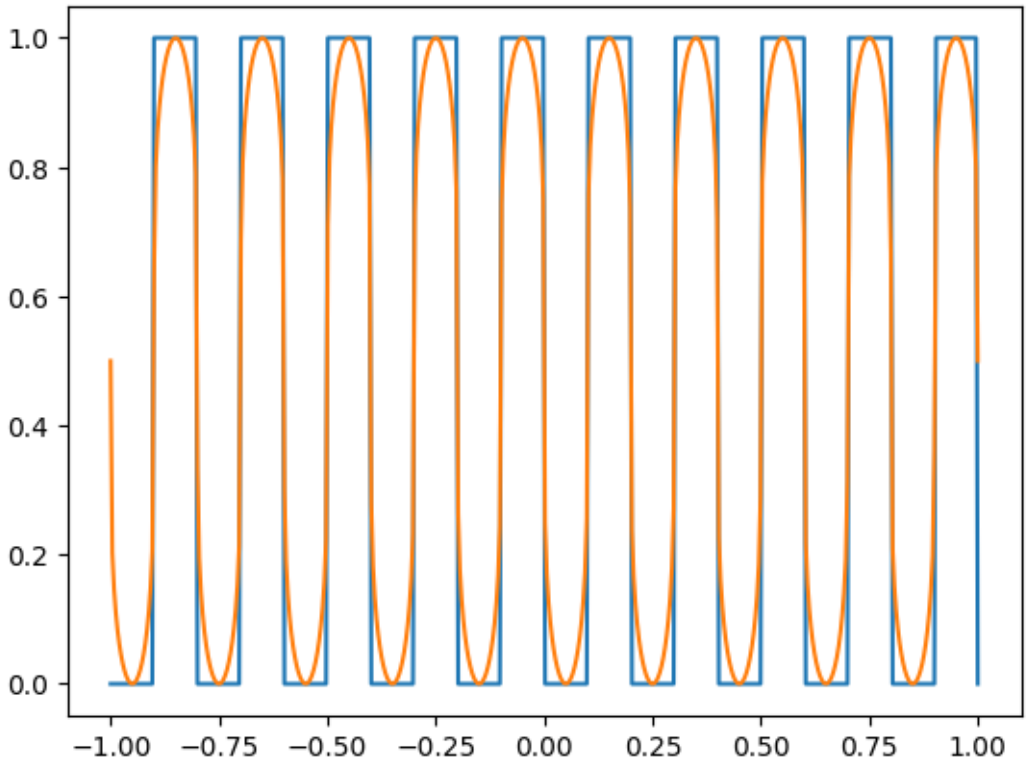
[12]:
```
nPts1d=500
rng=[-1,1]
res=0.15

X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)

img1=colorFromPos(X,Y,res)
img3=colorFromPos(U,V,res)

fig=plt.figure(figsize=(8,4))

# here an interpolation mode that is not "nearest" is more appropriate
# to dampen discretization artefacts
fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower",interpolation="bilinear")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map \"inverse\"")
plt.imshow(img3,extent=rng+rng,origin="lower",interpolation="bilinear")

plt.tight_layout()

plt.show()
```



[13]:
```
# another alternative: use a smoother pattern function
```

[14]:
```
# visualize discrete grid vs smoth oscillation
def comb(x,res):
```

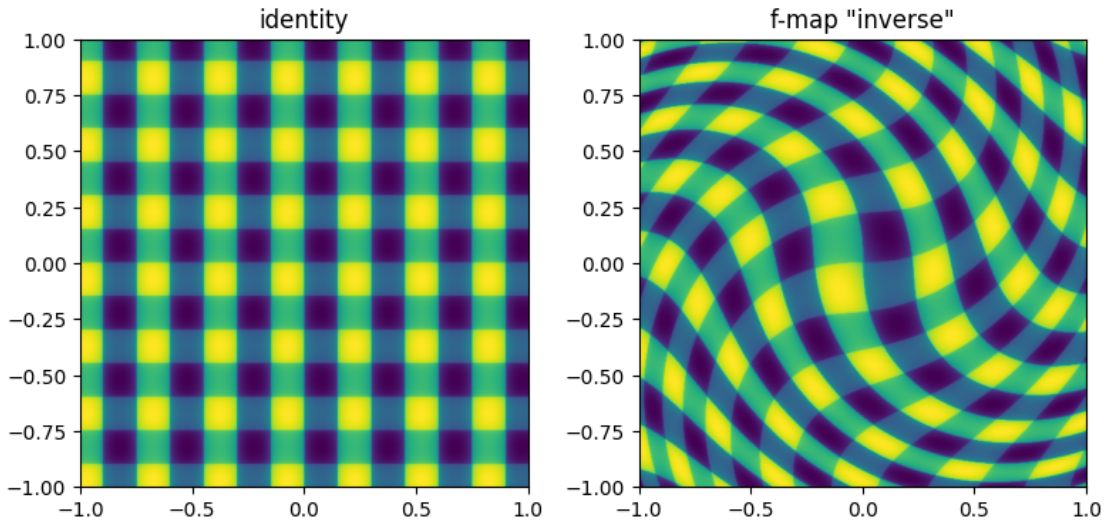
```
    return np.floor(np.mod(x/res,2))
def smooth_comb(x,res):
    cosdat=-np.sin(x/res*np.pi)
    return 0.5*(1+np.sign(cosdat)*np.abs(cosdat)**0.25)
x=np.linspace(-1,1,num=500)
plt.plot(x,comb(x,0.1))
plt.plot(x,smooth_comb(x,0.1))
plt.show()
```



[15]:
```
def colorFromPos2(X,Y,res):
    # divide coords into square grid of edge length res
    # determine color of "which square" pixel is in
    indexX=smooth_comb(X,res)
    indexY=smooth_comb(Y,res)
    return indexX*0.67+indexY*0.33
```

[16]:
```
nPts1d=500
rng=[-1,1]
res=0.15

X,Y,U,V=getUniformData(nPts1d,rng,usemapinv)
```

```
img1=colorFromPos2(X,Y,res)
img3=colorFromPos2(U,V,res)

fig=plt.figure(figsize=(8,4))

fig.add_subplot(1,2,1,aspect=1.)
plt.title("identity")
plt.imshow(img1,extent=rng+rng,origin="lower",interpolation="bilinear")

fig.add_subplot(1,2,2,aspect=1.)
plt.title("f-map \"inverse\"")
plt.imshow(img3,extent=rng+rng,origin="lower",interpolation="bilinear")

plt.tight_layout()

plt.show()
```



[ ]:

2023-05-15_ChartTypes_013_VectorFields-StreamPlots

May 25, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Vector fields

1.1 Stream plots

- given vector field $v : \mathbb{R}^2 \to \mathbb{R}^2$, quiver plot shows small arrow $v(x)$ at positions x
- in some cases v is velocity field, e.g. wind in the air; a leaf would be taken along by the wind, its path is described by the equation

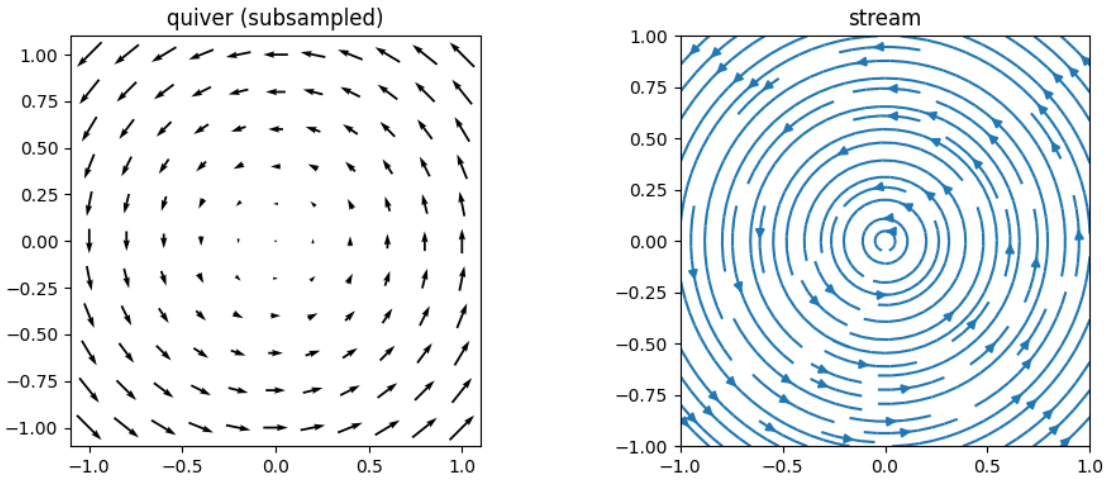
$$\partial_t y(t) = v(y(t)) \qquad \text{with some starting point } y(0)$$

- so instead of showing small arrows v at many positions, an alternative is to show paths of leafs for various starting points
- this is the idea of the stream plot

1.1.1 Simple circular example

[3]:
```
nPts1d=21
x = np.linspace(-1,1,num=nPts1d)
y = x
nPts=nPts1d**2
X, Y = np.meshgrid(x, y)
u = -Y
```

```
v = X

fig=plt.figure(figsize=(10,4))
ax=fig.add_subplot(1,2,1,aspect=1.)
ax.quiver(X[::2,::2],Y[::2,::2],u[::2,::2],v[::2,::2],pivot="mid")
plt.title("quiver (subsampled)")

ax=fig.add_subplot(1,2,2,aspect=1.)
ax.streamplot(X,Y,u,v)
plt.title("stream")

plt.tight_layout()
plt.show()
```

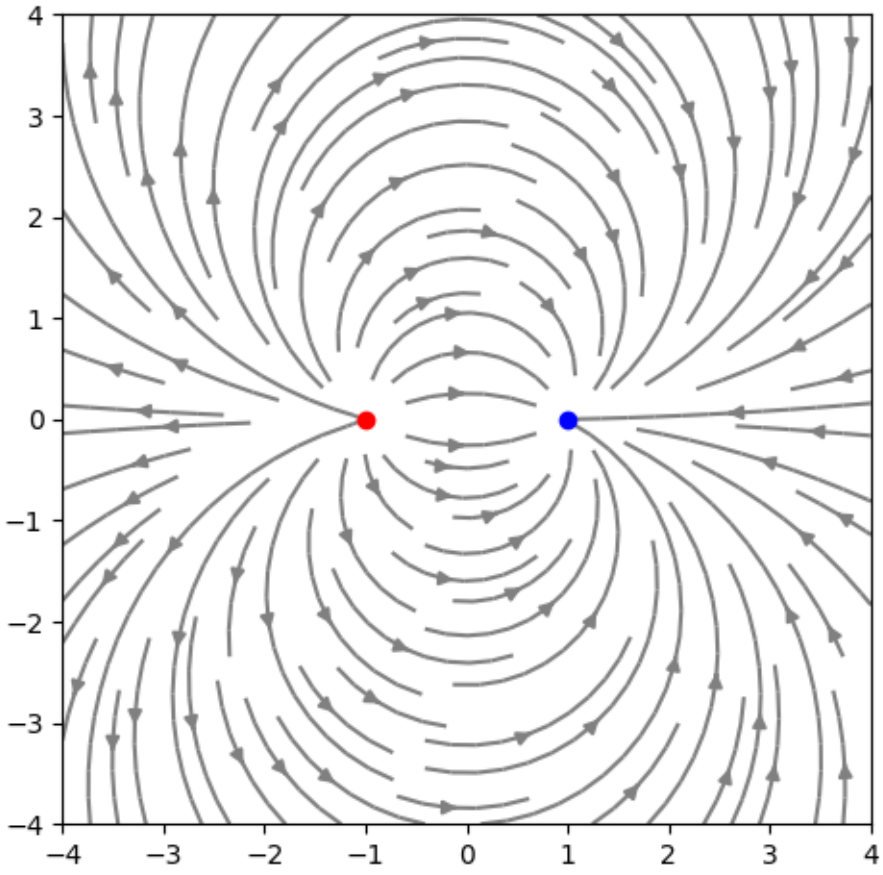


[4]:
```
# nice illustration of curl of a div free vector field:
# lines do not "collide", converge or diverge
```

1.1.2 Electric field of point charges

[5]:
```
# this example is inspired by:
# https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.
  ↪15-Quiver-and-Stream-Plots/
# Coulomb force field of two point charges wit opposite sign

nPts1d=200
x = np.linspace(-4,4,num=nPts1d)
y = x
nPts=nPts1d**2
```

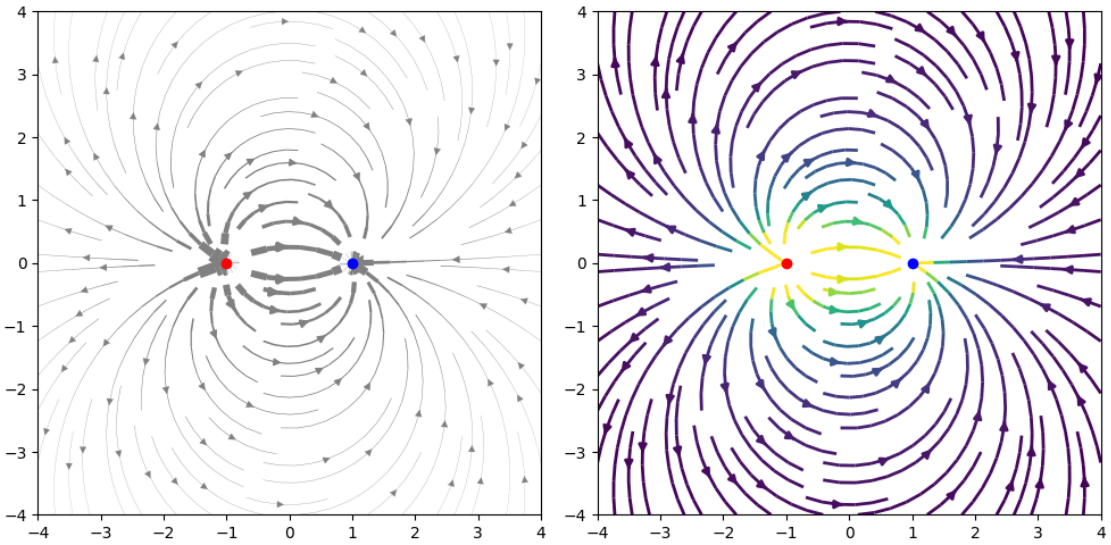
```
X,Y = np.meshgrid(x,y)
Ex = (X + 1)/((X+1)**2 + Y**2) - (X - 1)/((X-1)**2 + Y**2)
Ey = Y/((X+1)**2 + Y**2) - Y/((X-1)**2 + Y**2)

fig=plt.figure()
ax=fig.add_subplot(aspect=1.)

ax.streamplot(X,Y,Ex,Ey,color="#808080")

ax.scatter([-1],[0],c="r",zorder=2)
ax.scatter([1],[0],c="b",zorder=2)

plt.tight_layout()

plt.show()
```



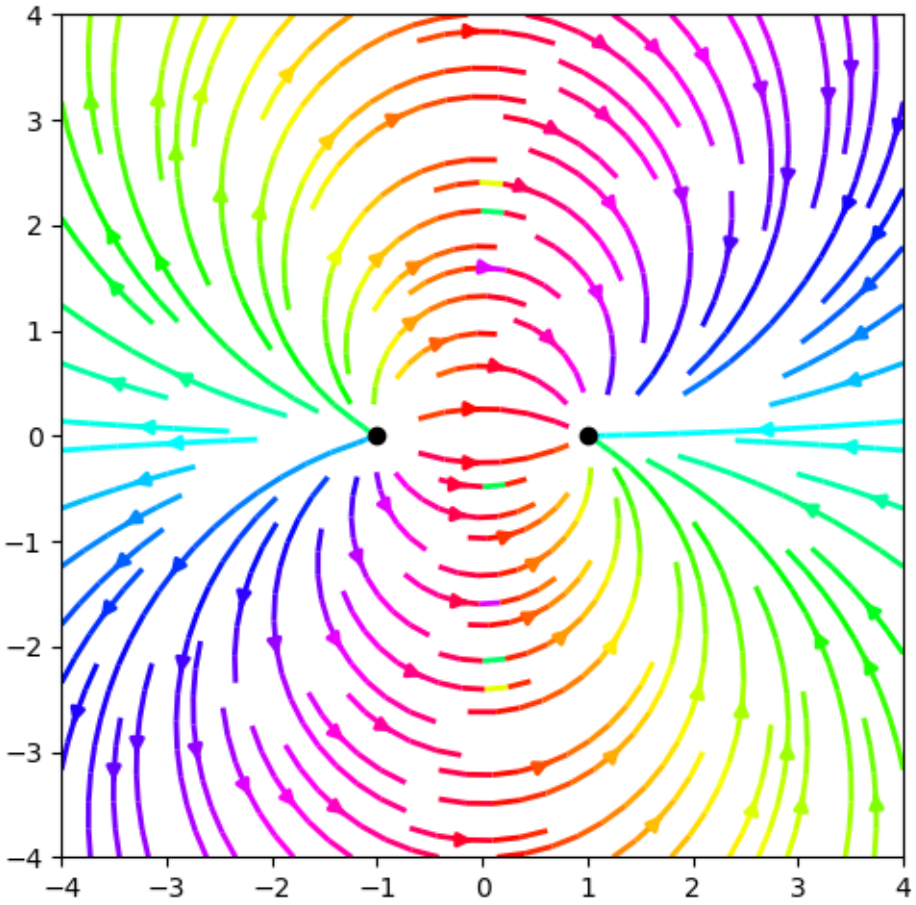
[6]:
```
# we see the divergence structure of the field relatively nicely
# there are clear sources and sinks of lines
```

1.1.3 Speed / field strength as color or thickness

[7]:
```
# this example is inspired by:
# https://problemsolvingwithpython.com/06-Plotting-with-Matplotlib/06.
 ↪15-Quiver-and-Stream-Plots/

nPts1d=100
x = np.linspace(-4,4,num=nPts1d)
y = x
nPts=nPts1d**2

X,Y = np.meshgrid(x,y)
Ex = (X + 1)/((X+1)**2 + Y**2) - (X - 1)/((X-1)**2 + Y**2)
Ey = Y/((X+1)**2 + Y**2) - Y/((X-1)**2 + Y**2)

EMag=(Ex**2+Ey**2)**0.5

fig=plt.figure(figsize=(10,5))

# line width
ax=fig.add_subplot(1,2,1,aspect=1.)
# signal: normalize, truncate at some maximal strength
vmax=10.
signal=np.clip(EMag/vmax,0.,1.)
ax.streamplot(X,Y,Ex,Ey,linewidth=15*signal,color="#808080")
ax.scatter([-1],[0],c="r",zorder=2)
ax.scatter([1],[0],c="b",zorder=2)


# line width
ax=fig.add_subplot(1,2,2,aspect=1.)
# signal: normalize, truncate at some maximal strength

vmax=2.
#signal=1-np.clip(EMag/vmax,0.,1.)
#ax.streamplot(X,Y,Ex,Ey,linewidth=2,color=1-signal,cmap=cm.gray)
signal=np.clip(EMag/vmax,0.,1.)
ax.streamplot(X,Y,Ex,Ey,linewidth=2,color=signal,cmap=cm.viridis)

ax.scatter([-1],[0],c="r",zorder=2)
ax.scatter([1],[0],c="b",zorder=2)

plt.tight_layout()
plt.show()
```



[8]:
```
# yet another one: color for orientation
fig=plt.figure(figsize=(5,5))

ax=fig.add_subplot(aspect=1.)

signal=np.mod(np.arctan2(Ey,Ex)/2/np.pi,1.)
ax.streamplot(X,Y,Ex,Ey,linewidth=2,color=signal,cmap=cm.hsv)

ax.scatter([-1,1],[0,0],c="k",zorder=2)

plt.tight_layout()
plt.show()
```



[ ]:
```
# can see some artifacts from linear interpolation around cut-off
# for "production" level figures probably need to implement this more carefully


# to be honest: most of these additional gimmicks seem to be "ducks" in these␣
 ↪examples
```

2023-05-15_ChartTypes_021_Chernoff

June 5, 2023

[23]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 Chernoff faces

1.1 Function for drawing

[26]:
```
def cface(ax, posX,posY,s,␣
 ↪x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17,x18):
    # based on:
    # https://healthyalgorithms.com/2012/11/12/
 ↪dataviz-in-python-chernoff-faces-with-matplotlib/
    # https://gist.github.com/aflaxman/4043086

    # posX, posY: position of center of face
    # s: scale of face
    tf=matplotlib.transforms.Affine2D(matrix=np.array([[s,0.,posX],[0.
 ↪,s,posY],[0.,0.,1.]]))+ax.transData

    # x1 = height  of upper face
    # x2 = overlap of lower face
    # x3 = half of vertical size of face
    # x4 = width of upper face
    # x5 = width of lower face
```

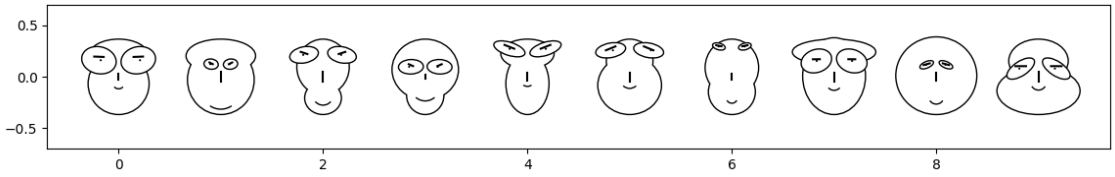
```
    # x6 = length of nose
    # x7 = vertical position of mouth
    # x8 = curvature of mouth
    # x9 = width of mouth
    # x10 = vertical position of eyes
    # x11 = separation of eyes
    # x12 = slant of eyes
    # x13 = eccentricity of eyes
    # x14 = size of eyes
    # x15 = position of pupils
    # x16 = vertical position of eyebrows
    # x17 = slant of eyebrows
    # x18 = size of eyebrows

    # transform some values so that input between 0,1 yields variety of output
    x3 = 1.9*(x3-.5)
    x4 = (x4+.25)
    x5 = (x5+.2)
    x6 = .3*(x6+.01)
    x8 = 5*(x8+.001)
    x11 /= 5
    x12 = 2*(x12-.5)
    x13 += .05
    x14 += .1
    x15 = .5*(x15-.5)
    x16 = .25*x16
    x17 = .5*(x17-.5)
    x18 = .5*(x18+.1)

    # top of face, in box with l=-x4, r=x4, t=x1, b=x3
    e = matplotlib.patches.Ellipse( (0,(x1+x3)/2), 2*x4, (x1-x3), fc='white',
            linewidth=2,ec="k",transform=tf)
    ax.add_artist(e)

    # bottom of face, in box with l=-x5, r=x5, b=-x1, t=x2+x3
    e = matplotlib.patches.Ellipse( (0,(-x1+x2+x3)/2), 2*x5, (x1+x2+x3),␣
↪fc='white',
            linewidth=2,ec="k",transform=tf)
    ax.add_artist(e)

    # cover overlaps
    e = matplotlib.patches.Ellipse( (0,(x1+x3)/2), 2*x4, (x1-x3), fc='white',␣
↪ec='none',transform=tf)
    ax.add_artist(e)
    e = matplotlib.patches.Ellipse( (0,(-x1+x2+x3)/2), 2*x5, (x1+x2+x3),␣
↪fc='white', ec='none',transform=tf)
    ax.add_artist(e)
```

```
    # draw nose
    ax.plot([0,0], [-x6/2, x6/2], 'k',transform=tf)

    # draw mouth
    p = matplotlib.patches.Arc( (0,-x7+.5/x8), 1/x8, 1/x8, theta1=270-180/np.
↪pi*np.arctan(x8*x9),
              theta2=270+180/np.pi*np.arctan(x8*x9),color="k",transform=tf)
    ax.add_artist(p)

    # draw eyes
    p = matplotlib.patches.Ellipse( (-x11-x14/2,x10), x14, x13*x14, angle=-180/
↪np.pi*x12,
              facecolor='white',ec="k",transform=tf)
    ax.add_artist(p)

    p = matplotlib.patches.Ellipse( (x11+x14/2,x10), x14, x13*x14, angle=180/np.
↪pi*x12,
              facecolor='white',ec="k",transform=tf)
    ax.add_artist(p)

    # draw pupils
    p = matplotlib.patches.Ellipse( (-x11-x14/2-x15*x14/2, x10), .05, .05,␣
↪facecolor='black',transform=tf)
    ax.add_artist(p)
    p = matplotlib.patches.Ellipse( (x11+x14/2-x15*x14/2, x10), .05, .05,␣
↪facecolor='black',transform=tf)
    ax.add_artist(p)

    # draw eyebrows
    ax.plot([-x11-x14/2-x14*x18/2,-x11-x14/2+x14*x18/
↪2],[x10+x13*x14*(x16+x17),x10+x13*x14*(x16-x17)],
              'k',transform=tf)
    ax.plot([x11+x14/2+x14*x18/2,x11+x14/2-x14*x18/
↪2],[x10+x13*x14*(x16+x17),x10+x13*x14*(x16-x17)],
              'k',transform=tf)
```

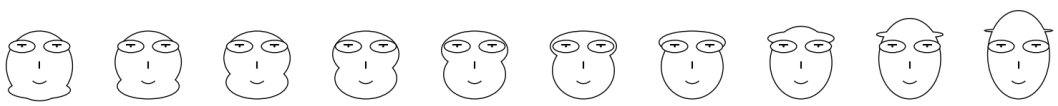
1.2 A few single faces to get acquainted

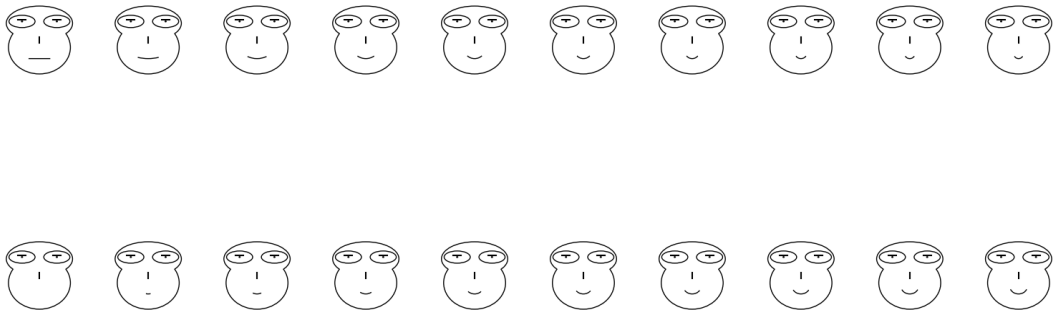
[25]:
```
# draw a random sequence
iMax=10
buffer=0.7
fig=plt.figure(figsize=(2*iMax,2))
ax=fig.add_subplot(aspect=1.)
for i in range(iMax):
```

```
    x=0.2+np.random.random(17)*0.6
    cface(ax,i,0.,0.4, .9, *x)
ax.axis([-buffer,iMax-1+buffer,-buffer,buffer])
plt.tight_layout()
plt.show()
```



[27]:
```
# vary one parameter at a time
for parNr in [0,1,2,6,7]:
    iMax=10
    buffer=0.6
    zList=np.linspace(0.,1.,num=iMax)
    fig=plt.figure(figsize=(2*iMax,2))
    ax=fig.add_subplot()
    for i in range(iMax):
        x=np.full(17,0.5)
        x[parNr]=zList[i]
        cface(ax,i,0.,0.4, .9, *x)
    ax.axis([-buffer,iMax-1+buffer,-buffer,buffer])
    plt.axis("off")
    plt.show()
```



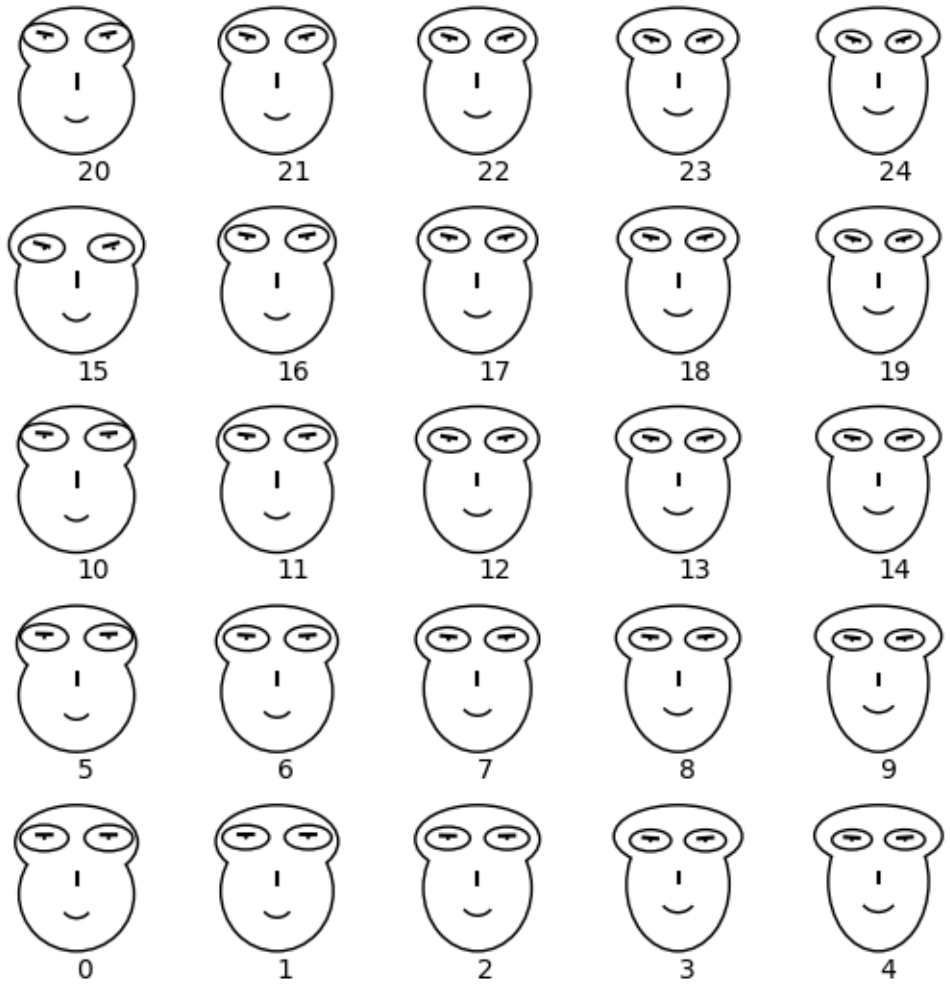


1.3 Try to spot outliers

[28]:
```
# create nPts * nPts grid of faces with random parameters
nPts=5

# create prep data
dataPre=np.zeros((nPts,nPts,3))
# positions
dataPre[:,:,0]=np.arange(nPts).reshape((1,-1))/nPts
dataPre[:,:,1]=np.arange(nPts).reshape((-1,1))/nPts
# a bit of randomness
dataPre[:,:,2]=0.2*np.random.random(size=(nPts,nPts))
dataPre=dataPre.reshape((-1,3))

# create full data
data=np.zeros((nPts*nPts,3+18))
# copy positions
data[:,:2]=dataPre[:,:2]
sizeparam=.4
sizeparam2=0.6
# size and x1
data[:,2]=sizeparam/nPts
data[:,3]=0.9
# initialize all others with 0.5 ("average")
data[:,4:]=0.5
for i in range(4,21):
    # now vary each parameter smoothly based on position and with the above␣
 ↪randomness
    B=np.random.random(3)-0.5
    data[:,i]+=.3*np.einsum(B,[0],dataPre,[1,0],[1])
```

```
# select one sample and 10 attributes at random and perturb the corresponding␣
 ↪values
i=np.random.randint(nPts*nPts)
j=np.random.choice(np.arange(4,21),replace=False,size=10)
nAttr=j.shape[0]
data[i,j]+=0.6*(np.random.random(size=nAttr)-0.5)
```

[29]:
```
# now plot all faces
fig=plt.figure(figsize=(7,7))
ax=fig.add_subplot()
for j,x in enumerate(data):
    cface(ax,*x)
    ax.text(x[0],x[1]-0.5/nPts,j)
plt.xlim([-sizeparam2/nPts,(nPts-1+sizeparam2)/nPts])
plt.ylim([-sizeparam2/nPts,(nPts-1+sizeparam2)/nPts])
plt.axis("off")
plt.show()
```



[30]:
```
i
```

[30]: 15

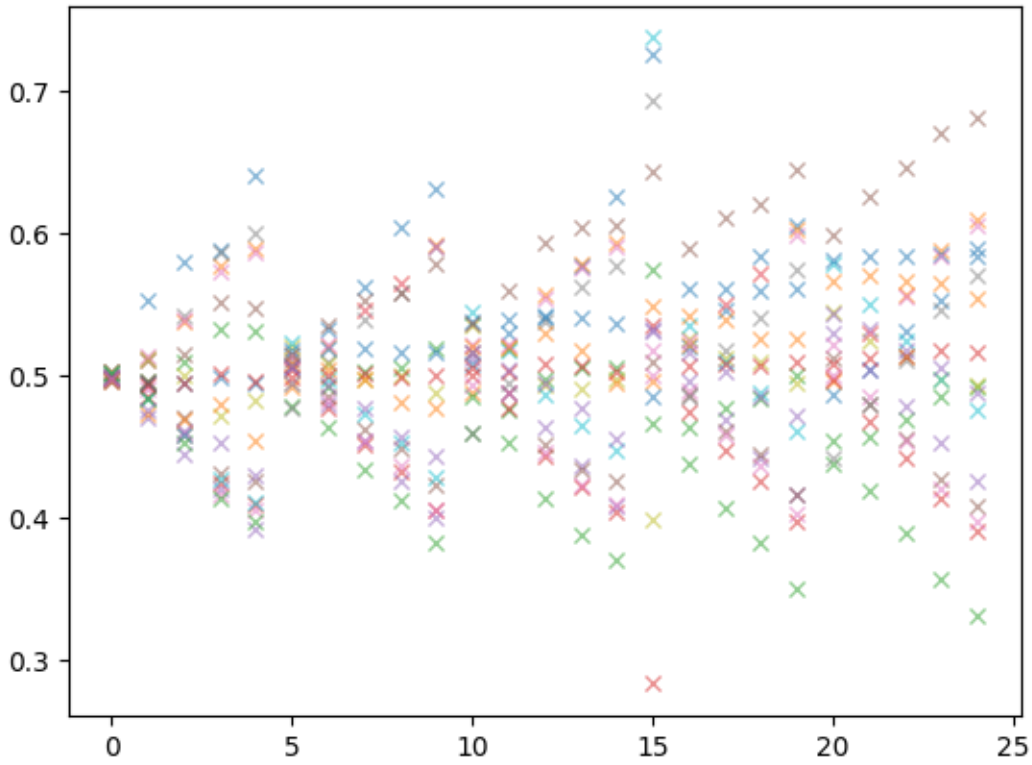
1.4 Alternative Visualizations

[31]:
```
data.shape
```

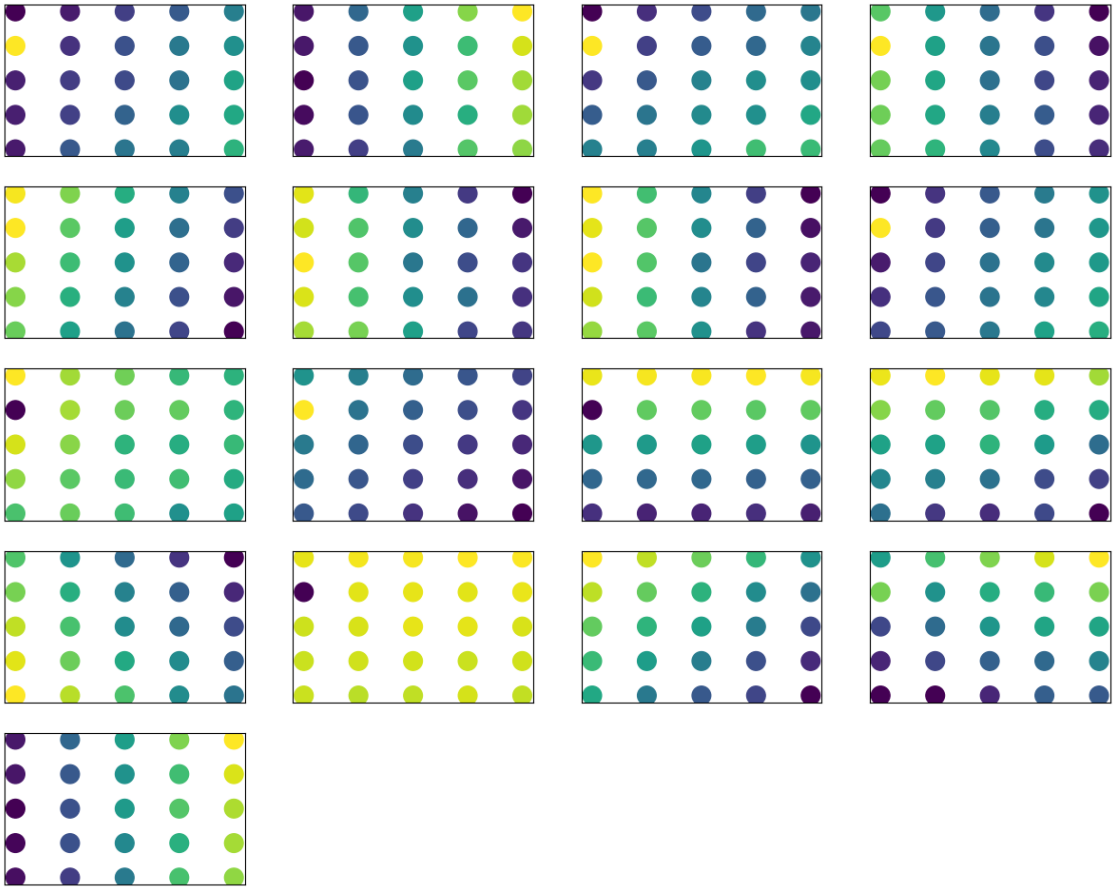
[31]: (25, 21)

[32]:
```
# primitive alternative visualization
# each x-value represents one sample, all values are simply shown as markers at␣
 ↪y-positions, with different colors
```

```
for i in range(4,21):
    plt.plot(data[:,i],alpha=0.5,marker="x",lw=0)
plt.show()
```



[33]:
```
# alternative visualization as many small scatter plots
# one attribute per scatter plot, one dot per sample (always at same position␣
  ↪as face), value encoded as color
nCols=5
jList=np.arange(4,21)
nRows=(len(jList)-1)//nCols+1

fig=plt.figure(figsize=(3*nCols,3*nRows))
for k,j in enumerate(jList):
    ax=fig.add_subplot(nCols,nRows,k+1)
    plt.scatter(data[:,0],data[:,1],c=data[:,j],s=200)
    #for i2,x in enumerate(data[:,:2]):
    #    ax.text(x[0],x[1]-0.5/nPts,i2)
    plt.xticks([])
    plt.yticks([])
plt.show()
```



1.5 Revisit same problem with PCA

[34]:
```
# do simple PCA on data matrix
# dataMat is assumed to be centered, matrix of shape (nSamples,dimSample)
def PCA(dataMat,keep=None):
    nSamples,dim=dataMat.shape
    if dim<nSamples:
        if keep is None:
            keep=dim
        A=dataMat.transpose().dot(dataMat)/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]
    else:
        if keep is None:
            keep=nSamples
        A=dataMat.dot(dataMat.transpose())/nSamples
        eigData=np.linalg.eigh(A)
```

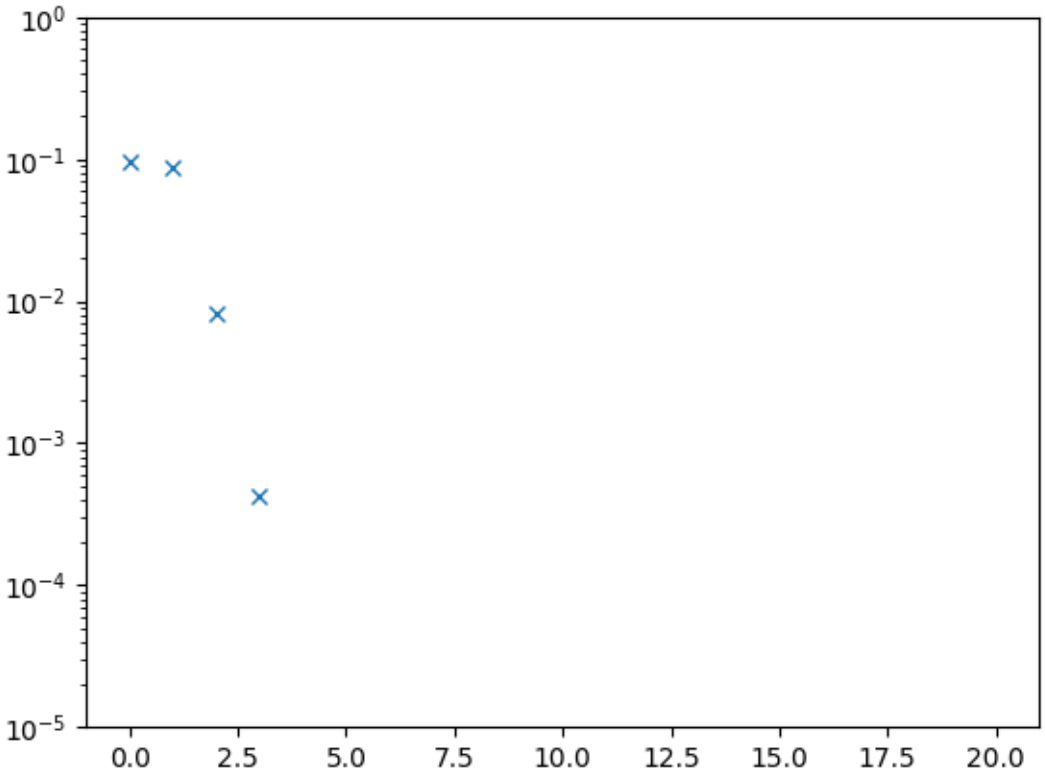
```
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]

        eigvec=np.einsum(eigvec,[0,1],dataMat,[1,2],[0,2])
        # renormalize
        normList=np.linalg.norm(eigvec,axis=1)
        eigvec=np.einsum(eigvec,[0,1],1/normList,[0],[0,1])
    return eigval,eigvec
```

[35]:
```
data.shape
```

[35]: (25, 21)

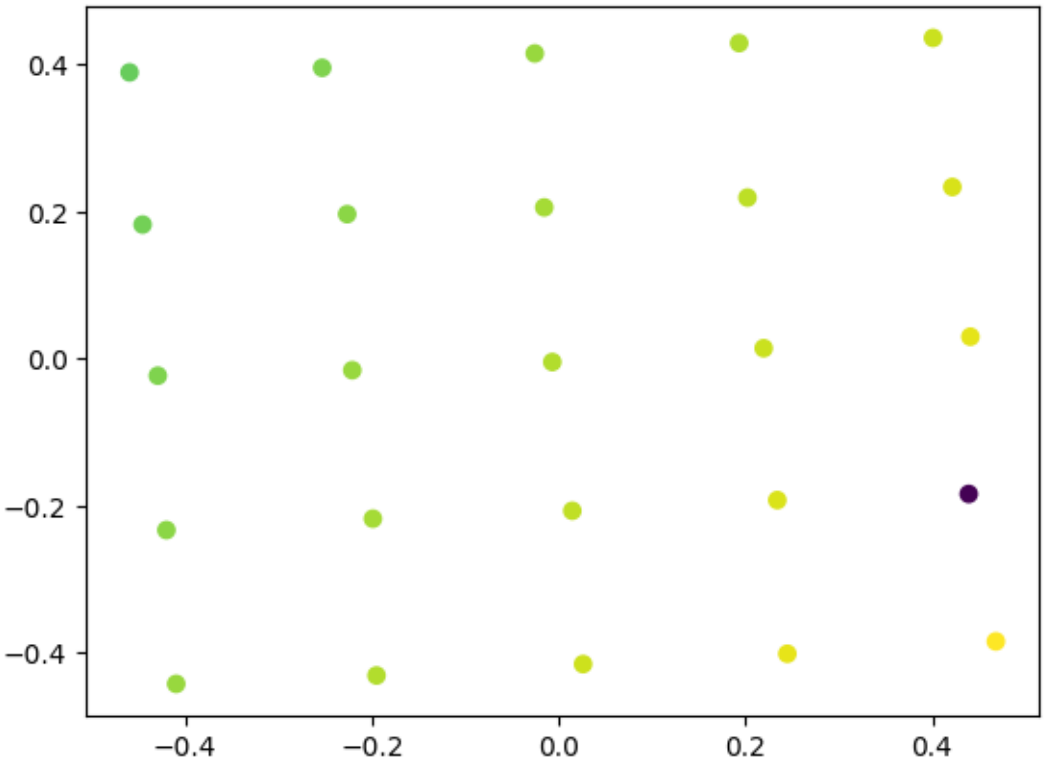
[36]:
```
dataNew=data.copy()
dataMean=np.mean(dataNew,axis=0)
dataNew-=dataMean
eigval,eigvec=PCA(dataNew)
```

[39]:
```
plt.plot(eigval,marker="x",lw=0)
plt.yscale("log")
plt.ylim([1E-5,1E0])
plt.show()
```



[40]:
```
coef=np.einsum(eigvec,[0,1],dataNew,[2,1],[2,0])
```

[41]:
```
plt.scatter(coef[:,0],coef[:,1],c=coef[:,2])
plt.show()
```



[ ]:

[ ]:

2023-05-15_ChartTypes_022_PCA

June 5, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

1 PCA

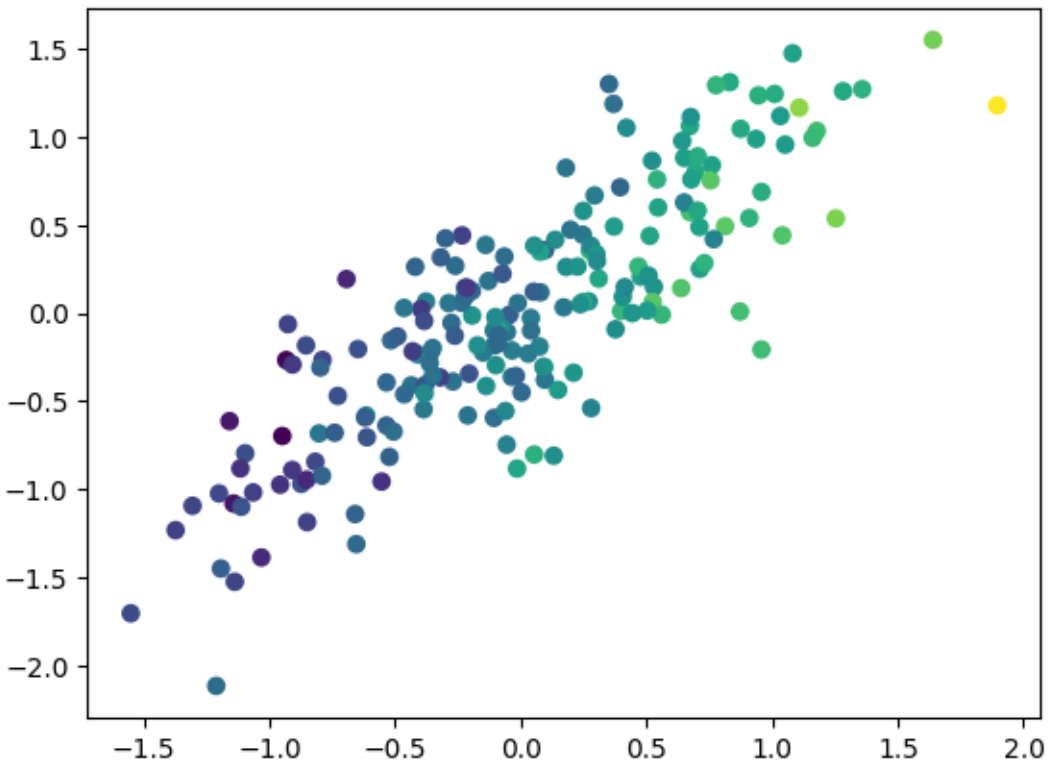
[2]:
```
# do simple PCA on data matrix
# dataMat is assumed to be centered, matrix of shape (nSamples,dimSample)
def PCA(dataMat,keep=None):
    nSamples,dim=dataMat.shape
    if dim<nSamples:
        if keep is None:
            keep=dim
        A=dataMat.transpose().dot(dataMat)/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]
    else:
        if keep is None:
            keep=nSamples
        A=dataMat.dot(dataMat.transpose())/nSamples
        eigData=np.linalg.eigh(A)
        eigval=(eigData[0][-keep::])[::-1]
        eigvec=((eigData[1][:,-keep::]).transpose())[::-1]

        eigvec=np.einsum(eigvec,[0,1],dataMat,[1,2],[0,2])
```

```
        # renormalize
        normList=np.linalg.norm(eigvec,axis=1)
        eigvec=np.einsum(eigvec,[0,1],1/normList,[0],[0,1])
    return eigval,eigvec
```

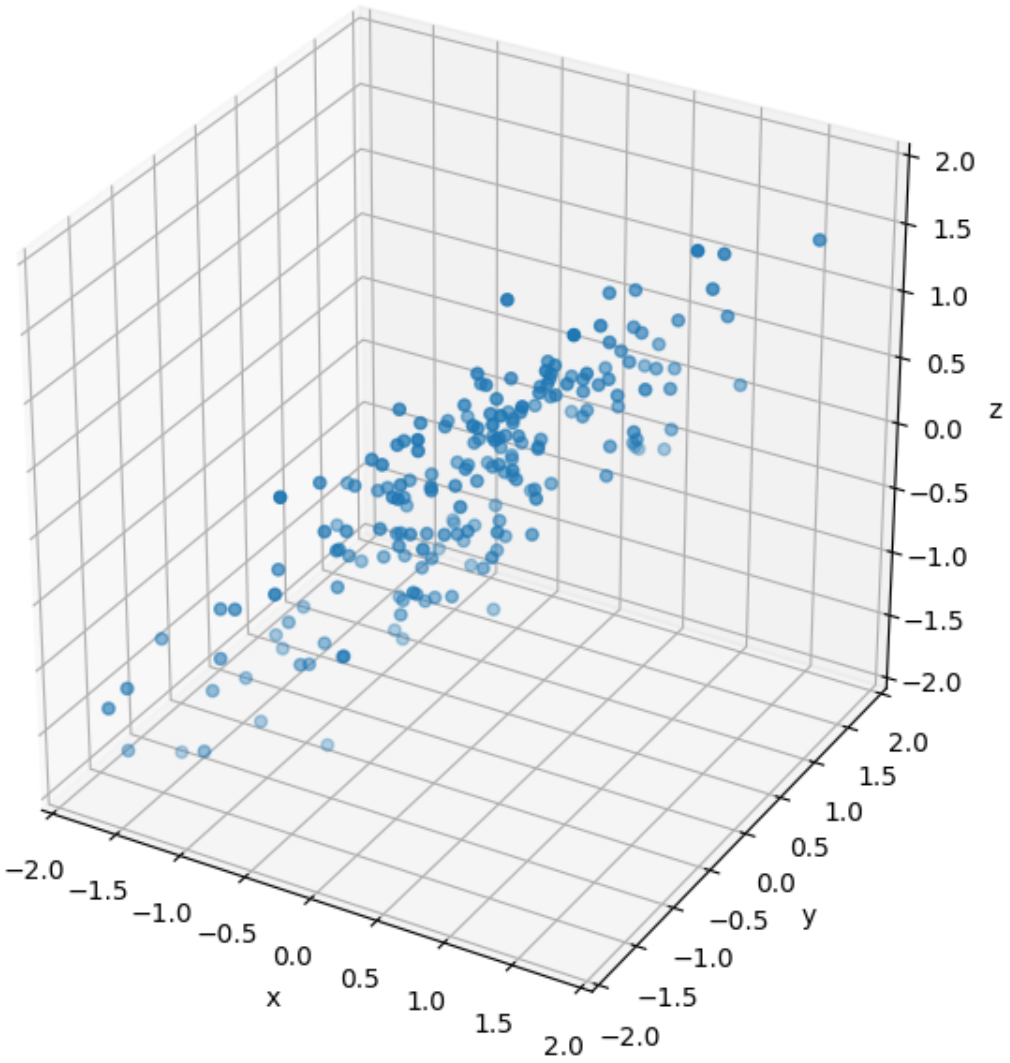
1.1 A toy example

[3]:
```
nPts=200
dim=3
data=np.random.normal(size=(nPts,dim))
# scale axes differently
scales=np.array([1.,0.5,0.2])
data=np.einsum(data,[0,1],scales,[1],[0,1])
# move directions a little
# each row of A contains a direction into which the corresponding dimension of␣
 ↪data should be pointed
A=np.array([[1.,1.,1.],[0.,1.,-1.],[1.,-0.5,-0.5]])
# normalize rows of A to unit length
norms=np.linalg.norm(A,axis=1)
A=np.einsum(A,[0,1],1/norms,[0],[0,1])
data=np.einsum(A,[0,1],data,[2,0],[2,1])
# center the data
dataMean=np.mean(data,axis=0)
data-=dataMean
```

[4]:
```
# just visualize the first 2 dimensions, plug third dimension into color
# we see: color is definitely strongly correlated with position
# so the positions do not tell the full story of the data
%matplotlib inline
plt.scatter(data[:,0],data[:,1],c=data[:,2])
plt.show()
```



[7]:
```
# visualize as 3d point cloud
%matplotlib widget
fig = plt.figure(figsize=(6,6))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
ax.set_xlim([-2,2])
ax.set_ylim([-2,2])
ax.set_zlim([-2,2])
ax.set_box_aspect((1.,1.,1.))

ax.scatter(data[:,0],data[:,1],data[:,2])
ax.set_xlabel("x")
ax.set_ylabel("y")
ax.set_zlabel("z")
plt.tight_layout()
plt.show()
```



[8]:
```
plt.close()
%matplotlib inline
```

[4]:
```
eigval,eigvec=PCA(data)
```

[5]:
```
# eigenvalues: correspond to variance along the principal directions
eigval
```

[5]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[6]:
```
# compare with predictions: scale factors in original Gaussians, squared
(scales)**2
```

[6]:
```
array([1.  , 0.25, 0.04])
```

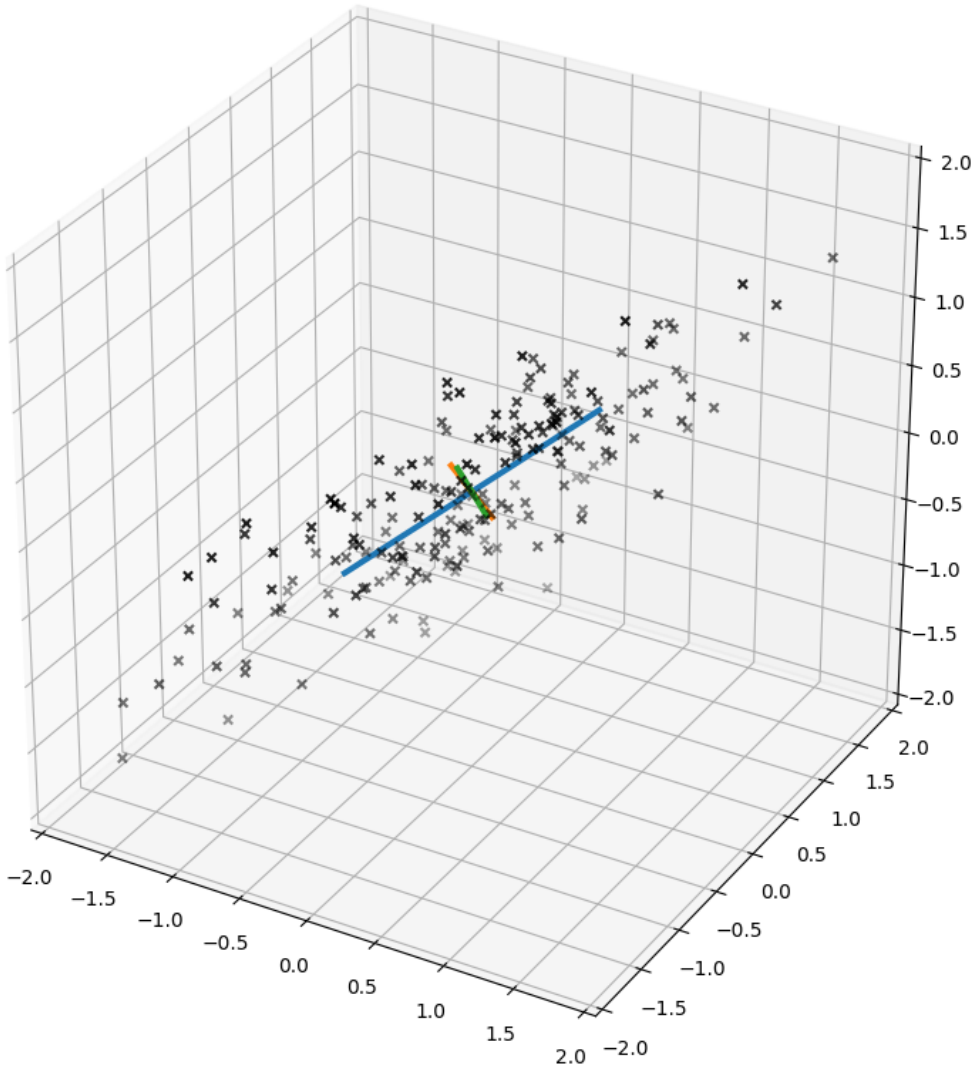
[7]:
```
# same for eigenvectors
eigvec
```

[7]:
```
array([[-0.61400857, -0.6193394 , -0.48929764],
       [ 0.08315523, -0.66722254,  0.74020219],
       [-0.7849068 ,  0.41380283,  0.46118167]])
```

[8]:
```
# compare with rows of A
A
```

[8]:
```
array([[ 0.57735027,  0.57735027,  0.57735027],
       [ 0.        ,  0.70710678, -0.70710678],
       [ 0.81649658, -0.40824829, -0.40824829]])
```

[9]:
```
# add eigenvectors with scale to plot
%matplotlib inline
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
ax.set_xlim([-2,2])
ax.set_ylim([-2,2])
ax.set_zlim([-2,2])
ax.set_box_aspect((1.,1.,1.))
# this step is required such that eigenvectors actually appear perpendicular to␣
 ↪each other

ax.scatter(data[:,0],data[:,1],data[:,2],marker="x",color="k")
for i,(val,vec) in enumerate(zip(eigval,eigvec)):
    # let x be the (unit) eigenvector, scaled by sqrt of eigenvalue (=std␣
 ↪deviation)
    x=vec*(val**0.5)
    # add line from -x to x into plot
    ax.plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=3)
plt.tight_layout()
plt.show()
```



[10]:
```
# projecting the points onto the eigenbasis
# (position of each point along the three colored lines)
coef=np.einsum(eigvec,[0,1],data,[2,1],[2,0])
print(coef.shape)
```

```
(200, 3)
```

[11]:
```
# compare covariance matrix of naive data, and of PCA coefficients
# by construction: the latter is diagonal
covAfter=np.cov(coef.transpose())
covBefore=np.cov(data.transpose())
print(covBefore)
print()
```

```
print(covAfter)
```

```
[[0.38012569 0.33179565 0.28455095]
 [0.33179565 0.47931376 0.16873304]
 [0.28455095 0.16873304 0.37090379]]

[[ 9.41557963e-01  1.45268061e-16 -7.75646453e-17]
 [ 1.45268061e-16  2.50773650e-01 -1.80758074e-18]
 [-7.75646453e-17 -1.80758074e-18  3.80116235e-02]]
```

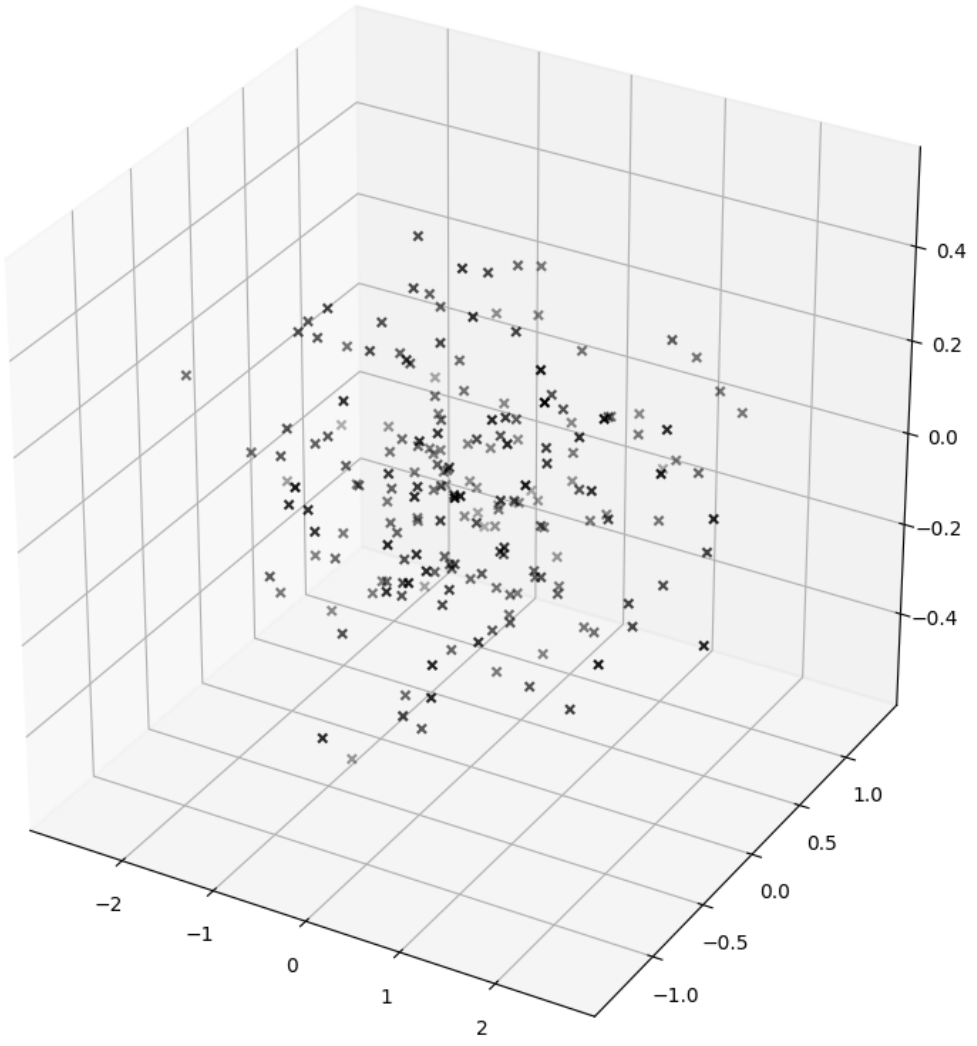
[12]:
```
# compare empirical variance of each coefficient with eigenvalues
np.var(coef,axis=0)
```

[12]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[13]:
```
eigval
```

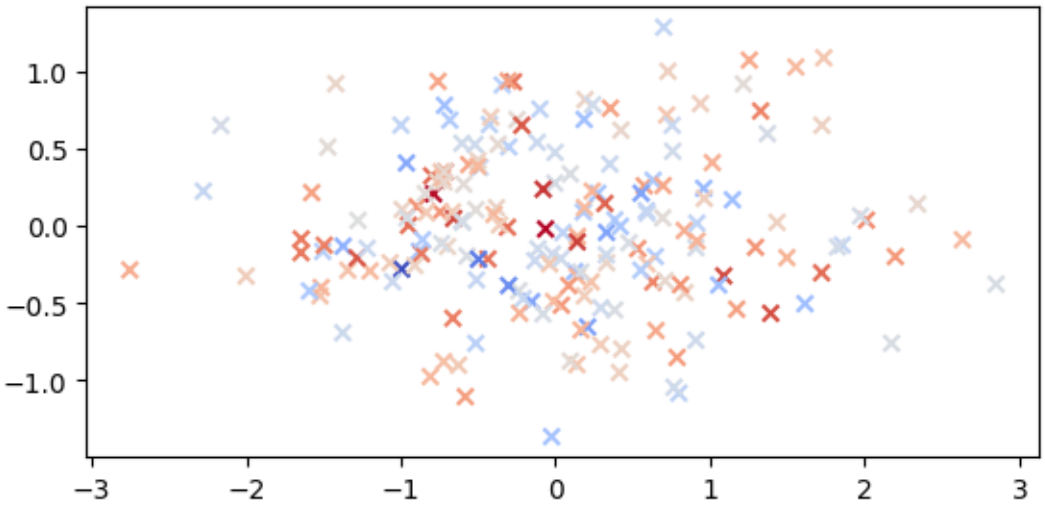
[13]:
```
array([0.93685017, 0.24951978, 0.03782157])
```

[14]:
```
# plot transformed coordinates
%matplotlib inline
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# set lim of each axis to -3 to +3 sigma of the given axis
ax.set_xlim([-3*eigval[0]**0.5,3*eigval[0]**0.5])
ax.set_ylim([-3*eigval[1]**0.5,3*eigval[1]**0.5])
ax.set_zlim([-3*eigval[2]**0.5,3*eigval[2]**0.5])
# set uniform aspect ratio = plot will look like cube
ax.set_box_aspect((1.,1.,1.))

ax.scatter(coef[:,0],coef[:,1],coef[:,2],marker="x",color="k")
plt.tight_layout()
plt.show()
```



[15]:
```
plt.close()
%matplotlib inline
```

[16]:
```
%matplotlib inline
fig=plt.figure()
fig.add_subplot(aspect=1.)
plt.scatter(coef[:,0],coef[:,1],marker="x",c=coef[:,2],cmap="coolwarm")
plt.show()
# this is the "best" 2d representation of the original point cloud
# in the sense that it is the 2d projection with the least loss of variance
# note: now color is uncorrelated with position
```

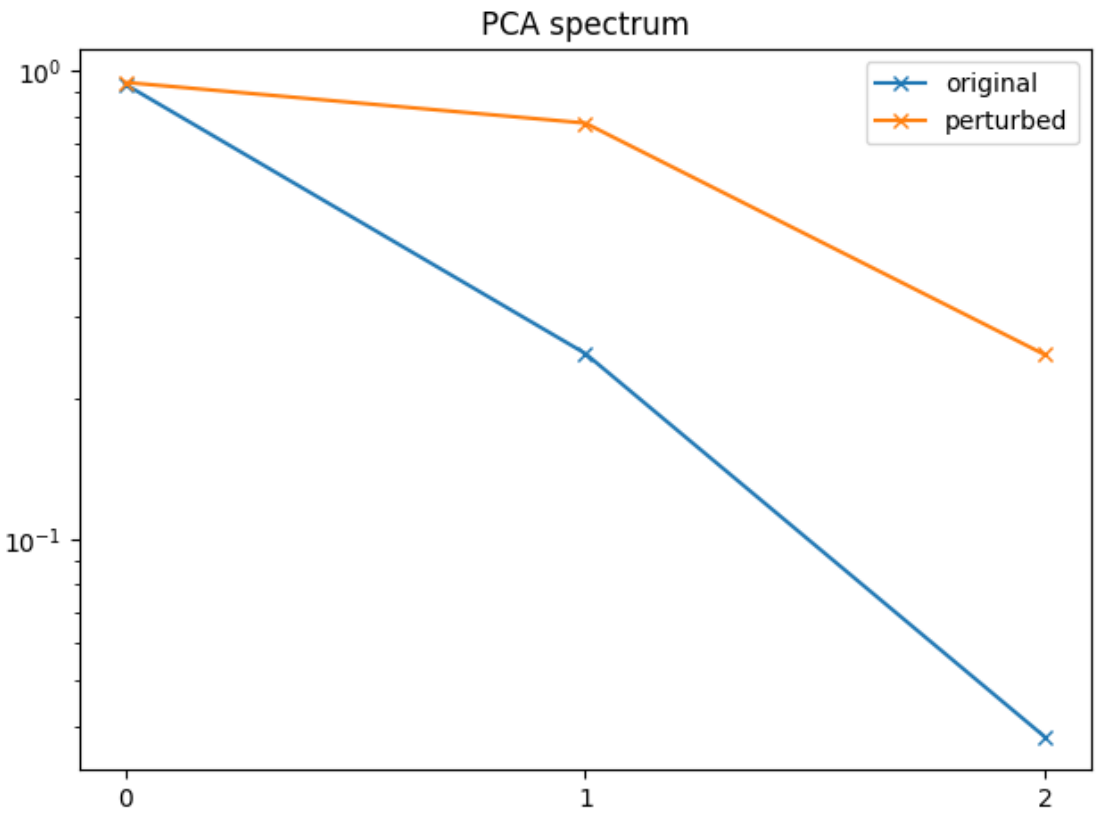


1.2 Careful: PCA is relatively susceptible to outliers

[17]:
```
# disrupt one of the points
dataNew=data.copy()
dataNew[0]=np.array([10.,-5.,-5.])
dataMean=np.mean(dataNew,axis=0)
dataNew-=dataMean
```

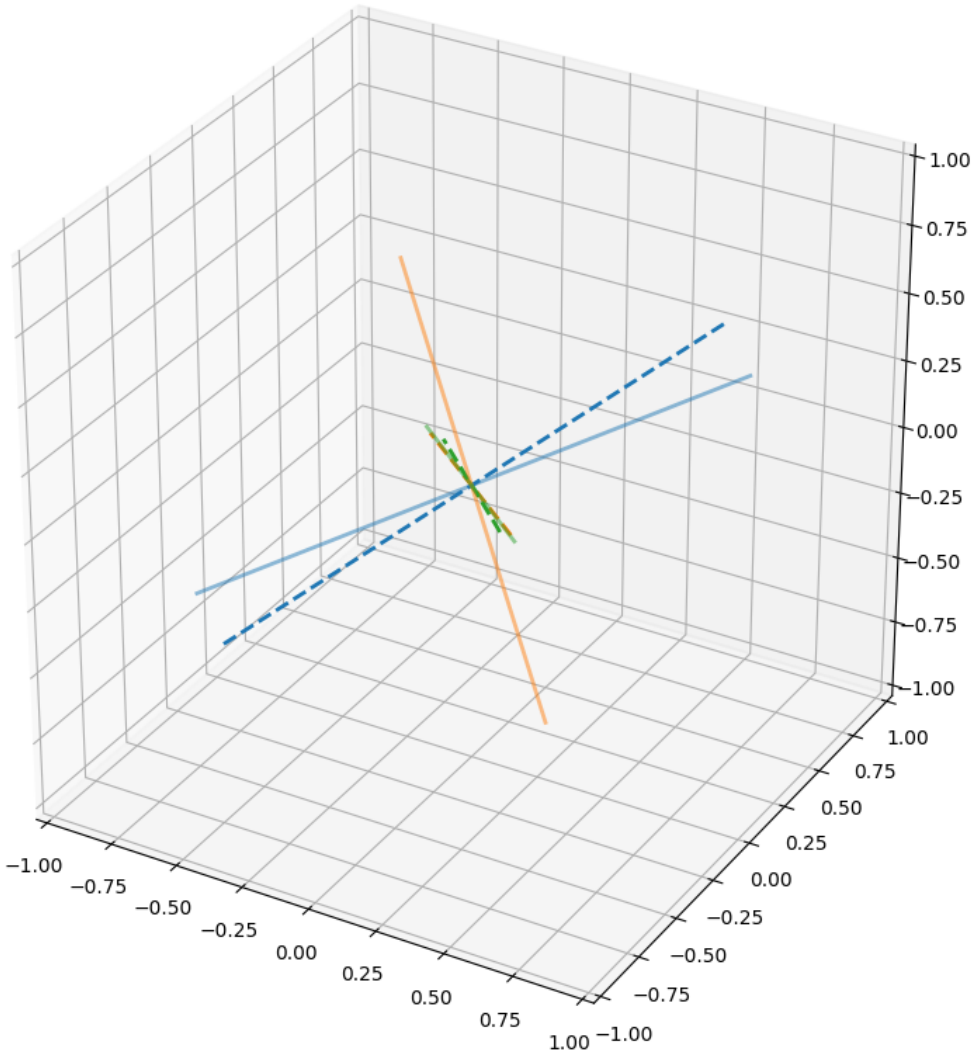
[18]:
```
eigvalNew,eigvecNew=PCA(dataNew)
```

[19]:
```
plt.title("PCA spectrum")
plt.plot(eigval,marker="x",label="original")
plt.plot(eigvalNew,marker="x",label="perturbed")
plt.yscale("log")
plt.xticks([0,1,2])
plt.legend()
plt.tight_layout()
plt.show()
```



[20]:
```
# add eigenvectors with scale to plot
%matplotlib widget
fig = plt.figure(figsize=(8,8))
ax = fig.add_subplot(111, projection='3d')
# aspect ratio
scale=1.
ax.set_xlim([-scale,scale])
ax.set_ylim([-scale,scale])
ax.set_zlim([-scale,scale])
ax.set_box_aspect((1.,1.,1.))

#ax.scatter(data[:,0],data[:,1],data[:,2],marker="x",color="k")
for i,(val,vec) in enumerate(zip(eigval,eigvec)):
    x=vec*(val**0.5)
    ax.
 ↪plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=2,ls="dashed")
for i,(val,vec) in enumerate(zip(eigvalNew,eigvecNew)):
    x=vec*(val**0.5)
    ax.plot([-x[0],x[0]],[-x[1],x[1]],[-x[2],x[2]],color=colors[i],lw=2,alpha=0.
 ↪5)
plt.tight_layout()
```

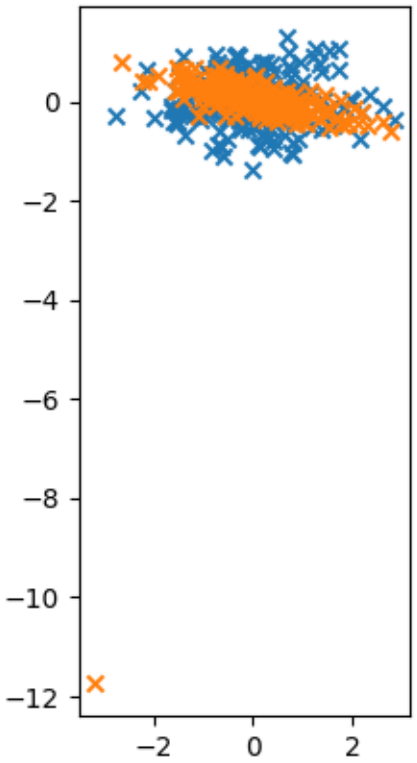
```
plt.show()
```



[21]:
```
plt.close()
%matplotlib inline
```

[22]:
```
coefNew=np.einsum(eigvecNew,[0,1],dataNew,[2,1],[2,0])
```

[23]:
```
%matplotlib inline
fig=plt.figure()
fig.add_subplot(aspect=1.)
plt.scatter(coef[:,0],coef[:,1],marker="x")
plt.scatter(coefNew[:,0],coefNew[:,1],marker="x")
plt.show()
```



1.3 Higher-dimensional example: discretized functions

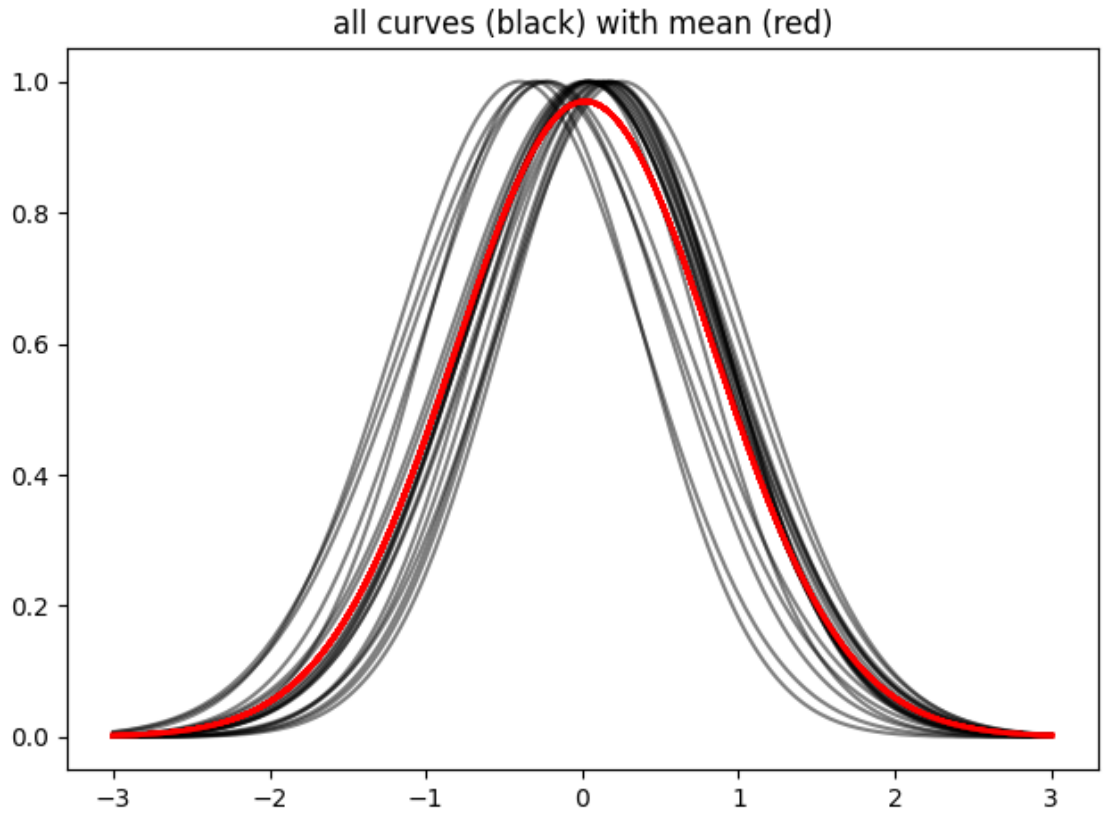
- each sample is a discrete function on nGrid gridpoints
- each sample generated as Gaussian bell curve, with random mean and variance
- we can interpret them as abstract vectors in $\mathbb{R}^{\text{nGrid}}$, but also visualize them easily as functions

1.3.1 data creation

[26]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=.2*np.random.normal(size=nPts)
listVar=0.3*np.random.random(size=nPts)+0.5
listY=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in␣
 ↪zip(listMean,listVar)])
mean=np.mean(listY,axis=0)
data=listY-np.reshape(mean,(1,-1))
```

1.3.2 processing

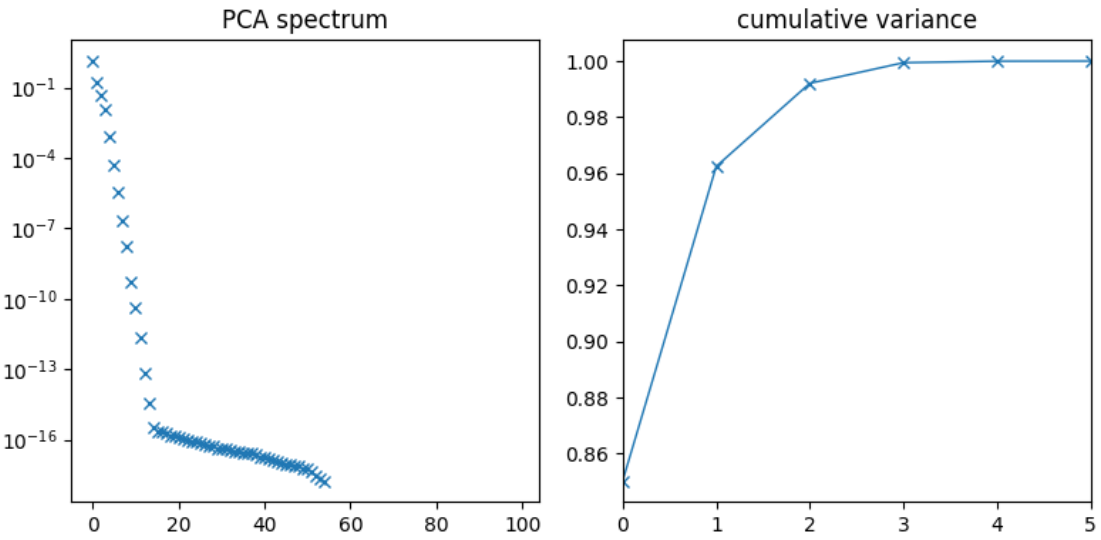
[27]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



[28]:
```
# perform PCA, interpret functions as abstract vectors
eigval,eigvec=PCA(data)
```

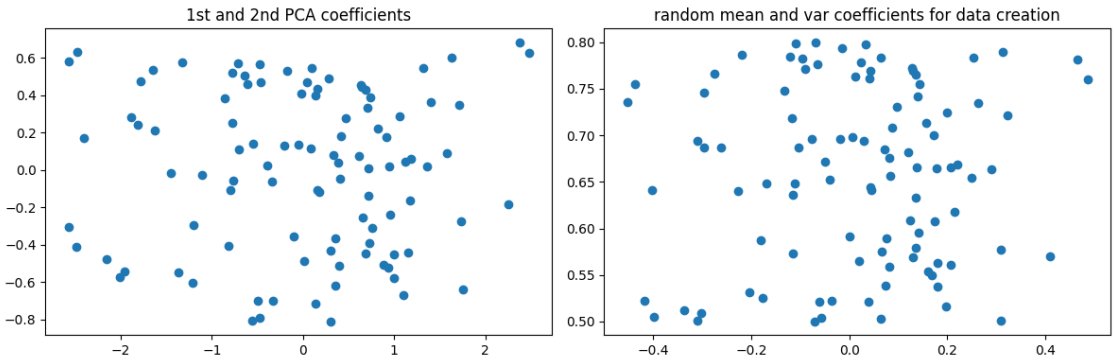
[29]:
```
# plot PCA spectrum, note: decays very quickly
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.yscale("log")
```

```
fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```



[30]:
```
# projection of samples onto eigenbasis
coef=np.einsum(eigvec,[0,1],data,[2,1],[2,0])
```

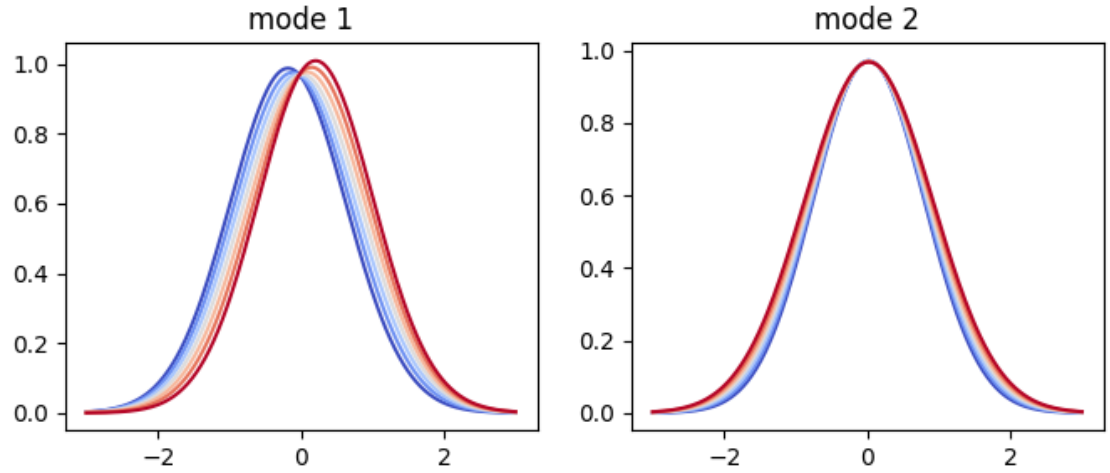
[31]:
```
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.title("1st and 2nd PCA coefficients")
plt.scatter(coef[:,0],coef[:,1])

fig.add_subplot(1,2,2)
plt.title("random mean and var coefficients for data creation")
plt.scatter(listMean,listVar)

plt.tight_layout()
plt.show()
```



visualize eigenmodes by projecting back to original sample space

- each eigenvector v_i can again be interpreted as function
- add this function to mean m, with different scales t, visualize collection of curves $m + t \cdot v_i$ where t should vary approximately according to the standard deviation $\sqrt{\lambda_i}$ along v_i
- in this way we can try and interpret 'meaning' of eigenvectors

[33]:
```
fig=plt.figure(figsize=(2*4,3))
for i in range(2):
    fig.add_subplot(1,2,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval[i]**0.5
    vList=np.linspace(-1,1,num=7)
    dataRe=np.einsum(eigvec[i],[0],vList*scale,[1],[1,0])
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



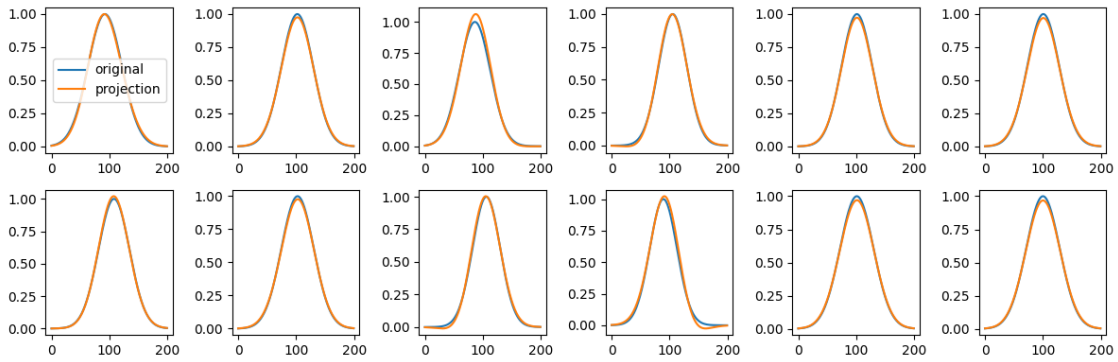
Visualize projections of samples onto eigenbasis

- the eigenvectors $(v_i)_{i=1}^n$ obtained by the PCA decomposition provide a full orthonormal basis of the sample space, each sample x can be written as

$$x = \sum_{i=1}^{n} v_i (v_i^\top x)$$

- sorting this sum from largest to smallest eigenvalue, and truncating after a few entries, we can project samples to the lower-dimensional reduced space. this gives a good qualitative feeling of how well a given basis can represent the samples
- careful: in the above formula we ignored the 'centering' of the data. x represents the centered sample. so for good visualization we need to add the mean again afterwards

[34]:
```
# how many basis vectors to use
keep=2
# the values v_i^\top x are already stored in the coef array
# now combine the first eigenvectors and the coef array
# np.einsum is really powerful, indices can sometimes be a bit tricky
# dont forget to add mean in the end
dataRed=np.einsum(eigvec[:keep,:],[0,1],coef[:,:keep],[2,0],[2,1])+mean
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data[i,:]+mean,label="original")
    plt.plot(dataRed[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```



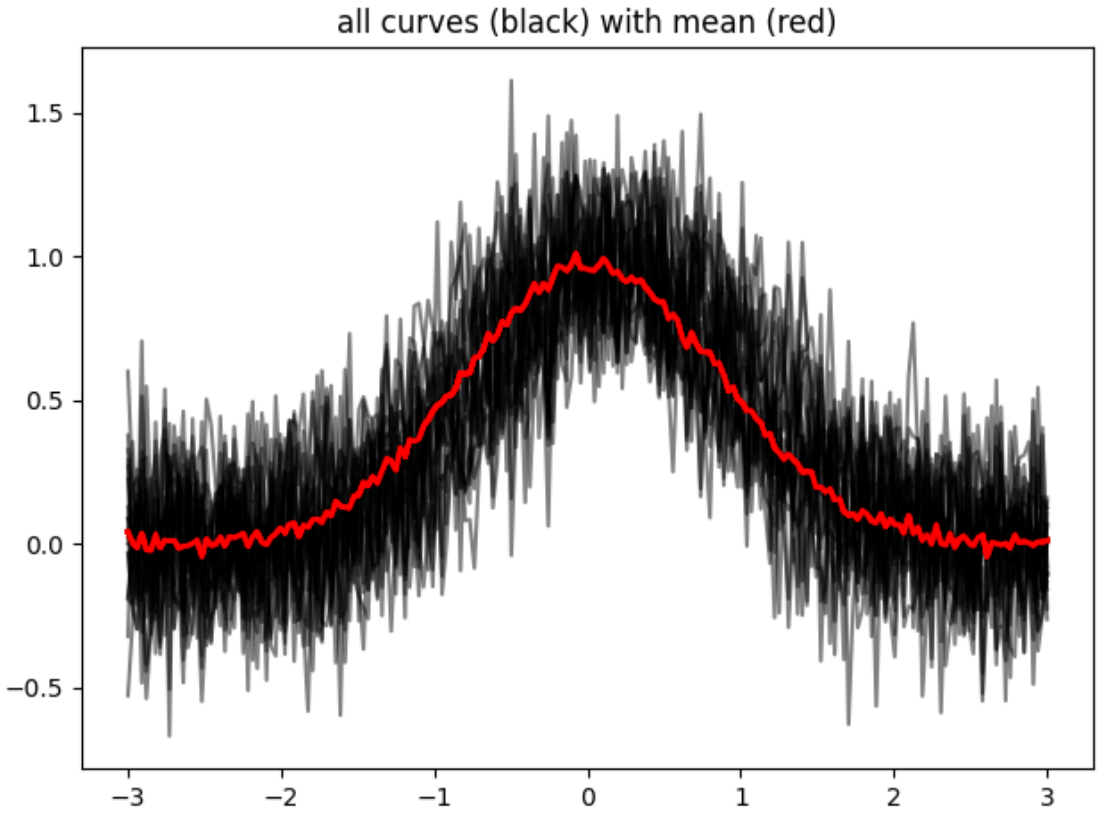
1.3.3 Retry with other data

Add random noise

- this still works, if noise has variance below data; otherwise we are in trouble

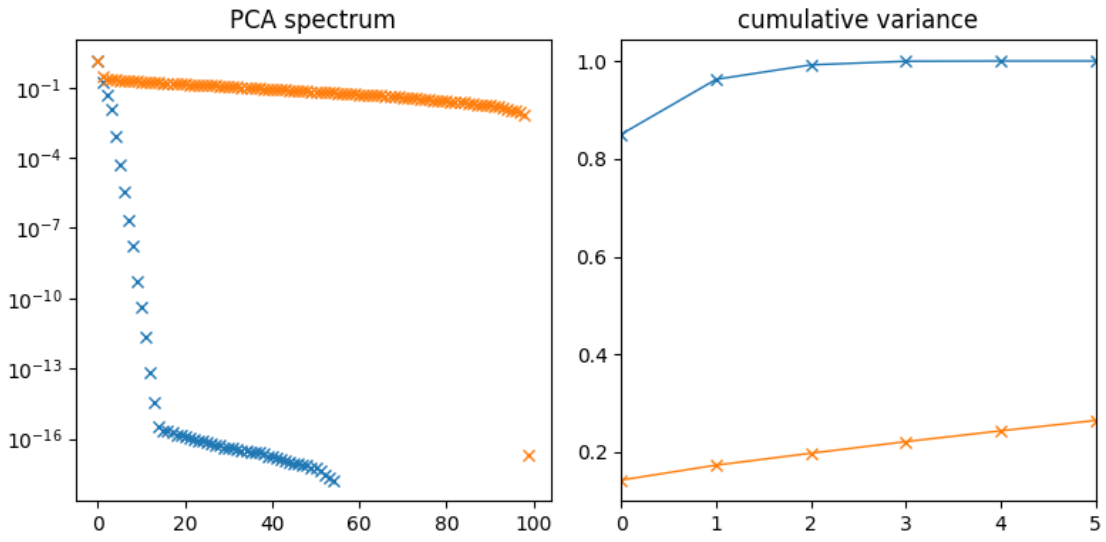
[42]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=.2*np.random.normal(size=nPts)
listVar=0.3*np.random.random(size=nPts)+0.5
listY=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in␣
 ↪zip(listMean,listVar)])
listY2=listY+0.2*np.random.normal(size=listY.shape)
mean2=np.mean(listY2,axis=0)
data2=listY2-np.reshape(mean2,(1,-1))
```

[43]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY2[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean2,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



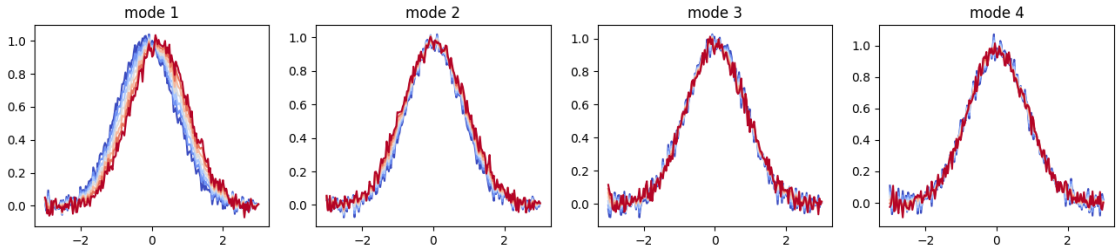
[44]:
```
# PCA and projected coefficients
eigval2,eigvec2=PCA(data2)
coef2=np.einsum(eigvec2,[0,1],data2,[2,1],[2,0])
```

[45]:
```
# plot spectrum
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.plot(eigval2,lw=0,marker="x")
plt.yscale("log")

fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.plot(np.cumsum(eigval2)/np.sum(eigval2),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```

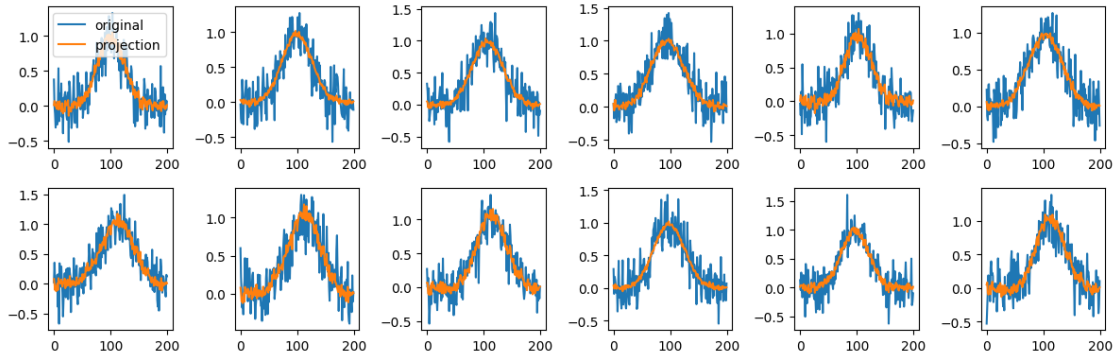


[46]:
```
# shooting along modes
nModes=4
fig=plt.figure(figsize=(nModes*4,3))
for i in range(nModes):
    fig.add_subplot(1,nModes,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval2[i]**0.5
    vList=np.linspace(-1,1,num=7)
```

```
    dataRe=np.einsum(eigvec2[i],[0],vList*scale,[1],[1,0])
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean2,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



[47]:
```
# projection onto basis
keep=2
dataRed2=np.einsum(eigvec2[:keep,:],[0,1],coef2[:,:keep],[2,0],[2,1])+mean2
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data2[i,:]+mean2,label="original")
    plt.plot(dataRed2[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```

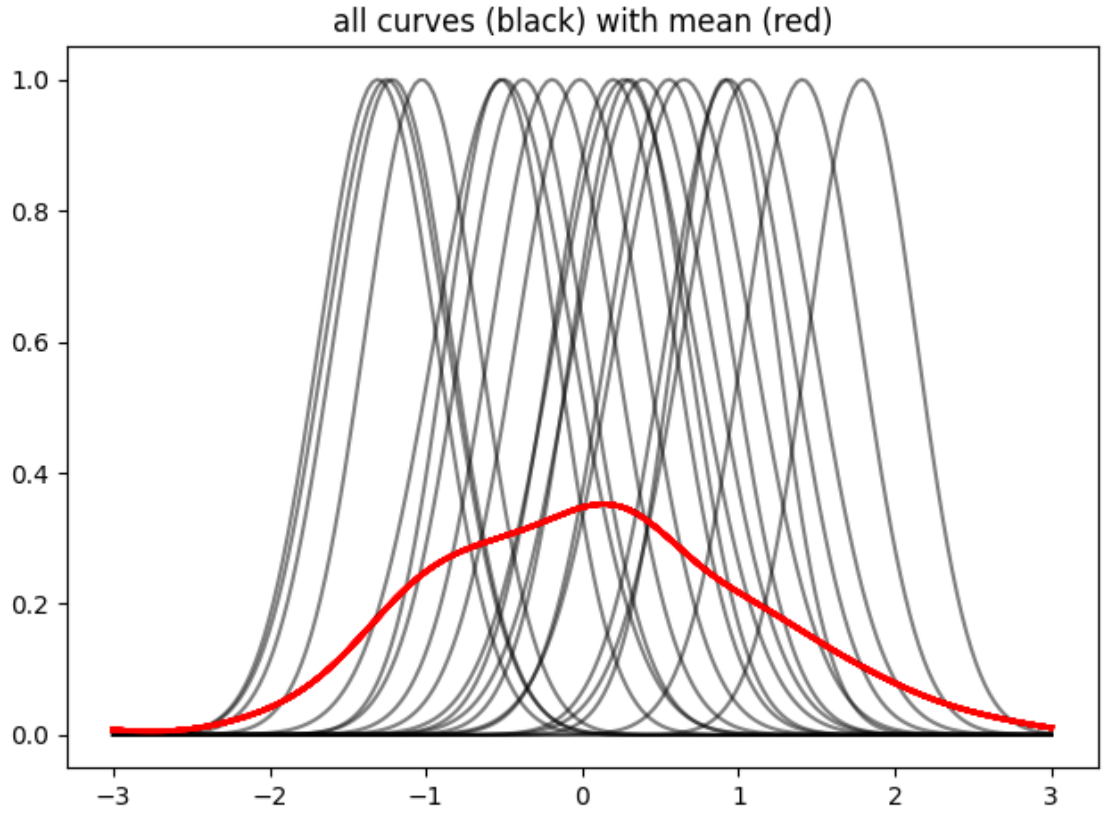


Much less overlap (lower variance, higher fluctuation in mean)

- in this case the standard Euclidean metric on the data samples is no longer informative
- good metrics are subject of ongoing research

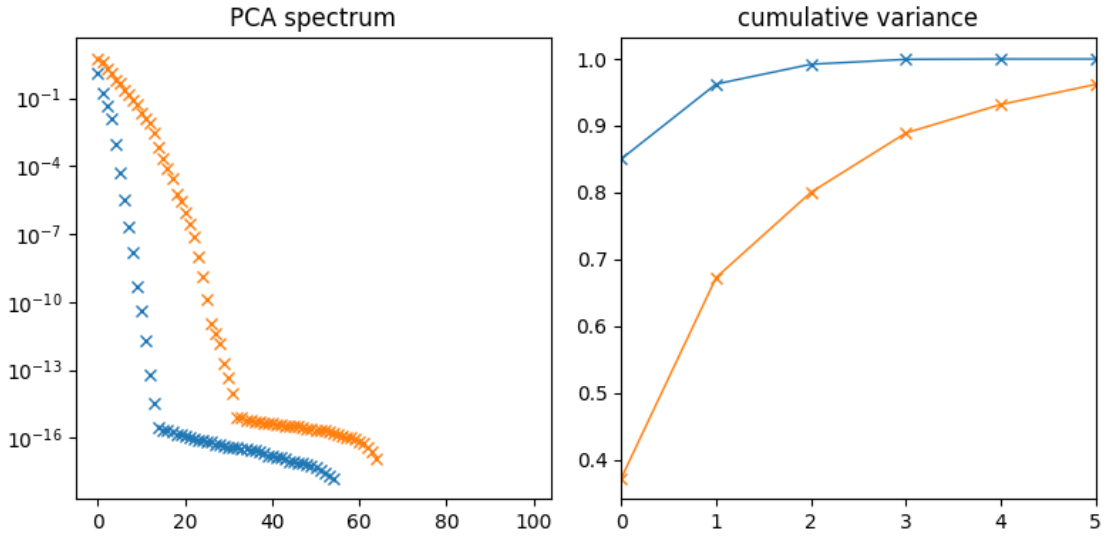
[49]:
```
nPts=100
nGrid=200
x=np.linspace(-3,3,num=nGrid)
listMean=1.*np.random.normal(size=nPts)
listVar=0.1*np.random.random(size=nPts)+0.1
listY2=np.array([np.exp(-0.5*(x-mean)**2/var) for mean,var in␣
 ↪zip(listMean,listVar)])
mean2=np.mean(listY2,axis=0)
data2=listY2-np.reshape(mean2,(1,-1))
```

[50]:
```
# visualize all functions/vectors and the mean, as simple line plots
%matplotlib inline
plt.title("all curves (black) with mean (red)")
for y in listY2[:20]:
    plt.plot(x,y,c="k",alpha=0.5)
    plt.plot(x,mean2,c="r",lw=2,zorder=2)
plt.tight_layout()
plt.show()
```



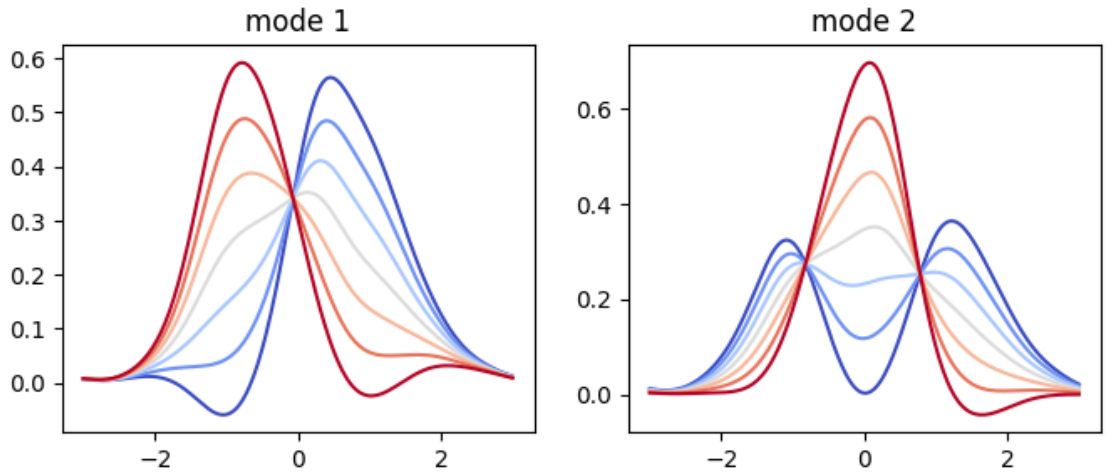
[51]:
```
# PCA and projected coefficients
eigval2,eigvec2=PCA(data2)
coef2=np.einsum(eigvec2,[0,1],data2,[2,1],[2,0])
```

[52]:
```
# plot spectrum
fig=plt.figure(figsize=(8,4))
fig.add_subplot(1,2,1)
plt.title("PCA spectrum")
plt.plot(eigval,lw=0,marker="x")
plt.plot(eigval2,lw=0,marker="x")
plt.yscale("log")

fig.add_subplot(1,2,2)
plt.title("cumulative variance")
plt.plot(np.cumsum(eigval)/np.sum(eigval),lw=1,marker="x")
plt.plot(np.cumsum(eigval2)/np.sum(eigval2),lw=1,marker="x")
plt.xlim([0,5])

plt.tight_layout()
plt.show()
```



[53]:
```
# shooting along modes
fig=plt.figure(figsize=(2*4,3))
for i in range(2):
    fig.add_subplot(1,2,i+1)
    plt.title("mode {:d}".format(i+1))
    scale=eigval2[i]**0.5
    vList=np.linspace(-1,1,num=7)
    dataRe=np.einsum(eigvec2[i],[0],vList*scale,[1],[1,0])
```

```
    for v,y in zip(vList,dataRe):
        plt.plot(x,y+mean2,c=cm.coolwarm(0.5*v+0.5))
plt.show()
```



[ ]:
```
# projection onto basis
keep=2
dataRed2=np.einsum(eigvec2[:keep,:],[0,1],coef2[:,:keep],[2,0],[2,1])+mean2
# for a few of them plot original sample and projection for comparison
fig=plt.figure(figsize=(12,4))
for i in range(12):
    fig.add_subplot(2,6,i+1)
    plt.plot(data2[i,:]+mean2,label="original")
    plt.plot(dataRed2[i,:],label="projection")
    if i==0:
        plt.legend()
plt.tight_layout()
plt.show()
```

[ ]:
```
```

2023-06-05_Charleston-Example

June 9, 2023

[2]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

[3]:
```
# original data
# number of victims
categoriesVictims=["total", "white", "black", "hisp", "other"]
total=np.array([6484507,4091971,955800,995996,440741])
# distribution of offenders: rows are victim categories, columns are␣
 ↪percentages of offenders
categoriesOffenders=["white", "black", "hisp", "other", "unknown"]
ratios=np.array([\
        [42.9,22.4,14.8,12.1,7.8],
        [56.0,13.7,11.9,10.6,7.8],
        [10.4,62.2,4.7,15.0,7.7],
        [21.7,21.2,38.6,11.6,6.9],
        [40.3,19.3,10.6,20.3,9.5]
        ])/100.

# total ethnic census (careful: data from 2019, only for coarse illustrative␣
 ↪purposes)
population=np.array([0.603,0.134,0.185,0])
population[-1]=1-np.sum(population)
```

[7]:
```
# compute absolute numbers
# columns: victim category, rows: offender category
numbers=total.reshape((-1,1))*ratios
```

```
# now: normalize by colum (drop first row: total)
sums=np.sum(numbers[1:],axis=0)
ratios2=numbers[1:]/(sums.reshape((1,-1)))
```

[9]:
```
def print_table(offenders,victims,numbers,formstr):
    print("\t\toffender")
    print("\t",end="")
    for of in offenders:
        print("\t"+of,end="")
    print("")
    print("victim\t|")
    for i,vic in enumerate(victims):
        print(vic+"\t|",end="")
        for j in range(numbers.shape[1]):
            print("\t"+formstr.format(numbers[i,j]),end="")
        print("")
```

[10]:
```
print("absolute numbers (in 1E6)\n")
print_table(categoriesOffenders,categoriesVictims[1:],numbers[1:]/1E6,"{:5.2f}")

print("")
print("="*60)
print("")

print("relative numbers (per victim category: rows sum to 1)\n")
print_table(categoriesOffenders,categoriesVictims[1:],ratios[1:],"{:5.2f}")

print("")
print("="*60)
print("")

print("relative numbers (per offender category: cols sum to 1)\n")
print_table(categoriesOffenders,categoriesVictims[1:],ratios2,"{:5.2f}")
```

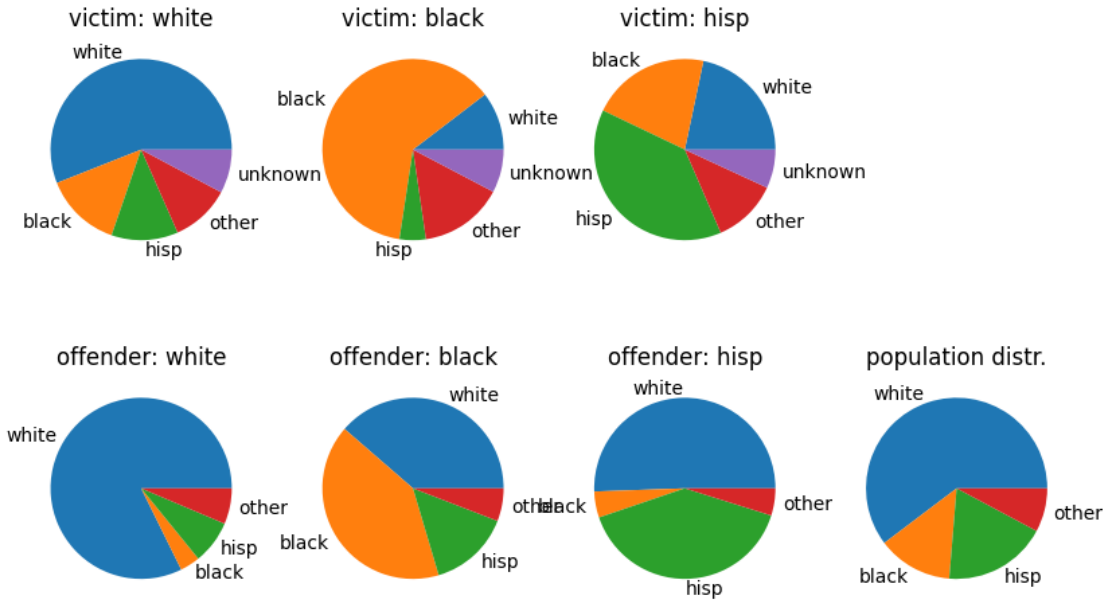
```
absolute numbers (in 1E6)

                offender
                white   black   hisp    other   unknown
victim  |
white   |        2.29    0.56    0.49    0.43    0.32
black   |        0.10    0.59    0.04    0.14    0.07
hisp    |        0.22    0.21    0.38    0.12    0.07
other   |        0.18    0.09    0.05    0.09    0.04

============================================================
```

```
relative numbers (per victim category: rows sum to 1)

                offender
                white   black   hisp    other   unknown
victim  |
white   |        0.56    0.14    0.12    0.11    0.08
black   |        0.10    0.62    0.05    0.15    0.08
hisp    |        0.22    0.21    0.39    0.12    0.07
other   |        0.40    0.19    0.11    0.20    0.10

============================================================

relative numbers (per offender category: cols sum to 1)

                offender
                white   black   hisp    other   unknown
victim  |
white   |        0.82    0.39    0.51    0.55    0.63
black   |        0.04    0.41    0.05    0.18    0.15
hisp    |        0.08    0.15    0.40    0.15    0.14
other   |        0.06    0.06    0.05    0.11    0.08
```

[7]:
```
# if we show the ratios normalized by victim category, we find that
# the highest value is "on the diagonal"
# Bureau of Justice Statistics (according to Cairo):
# "intraracial" crime is usualy higher than "interracial" for all crimes except␣
 ↪robbery
# usually you do not choose victims. mostly a matter of proximity (e.g.␣
 ↪domestic violence, or neighbours)
```

[15]:
```
fig=plt.figure(figsize=(10,6))

fig.add_subplot(2,4,1)
plt.title("victim: white")
plt.pie(ratios[1],labels=categoriesOffenders)

fig.add_subplot(2,4,2)
plt.title("victim: black")
plt.pie(ratios[2],labels=categoriesOffenders)

fig.add_subplot(2,4,3)
plt.title("victim: hisp")
plt.pie(ratios[3],labels=categoriesOffenders)


fig.add_subplot(2,4,5)
plt.title("offender: white")
```

```
plt.pie(ratios2[:,0],labels=categoriesVictims[1:])

fig.add_subplot(2,4,6)
plt.title("offender: black")
plt.pie(ratios2[:,1],labels=categoriesVictims[1:])

fig.add_subplot(2,4,7)
plt.title("offender: hisp")
plt.pie(ratios2[:,2],labels=categoriesVictims[1:])

fig.add_subplot(2,4,8)
plt.title("population distr.")
plt.pie(population,labels=categoriesVictims[1:])


plt.show()
```



[ ]:

2023-06-19_001_Graphs_Basic

June 19, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline
```

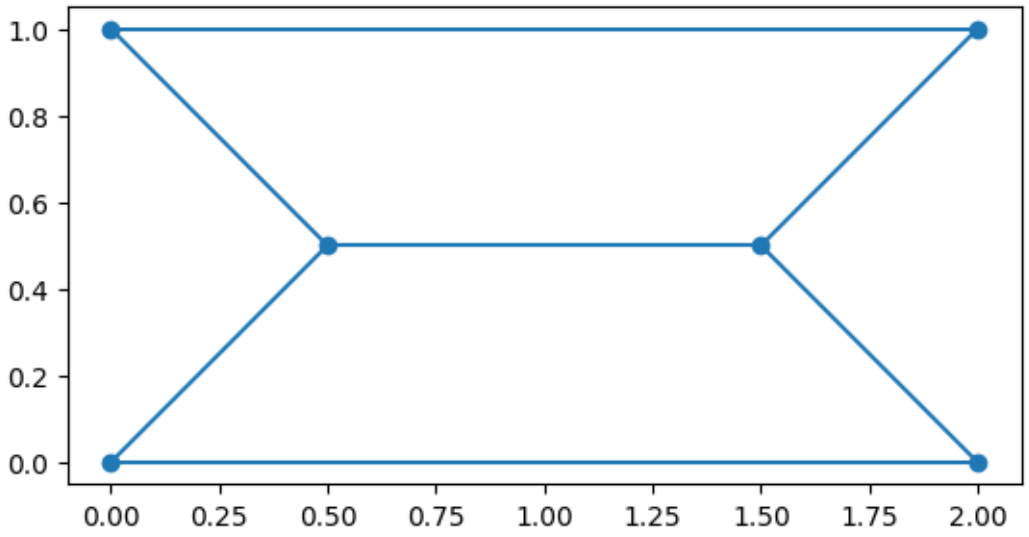
1 Graphs: getting started

1.1 Representing a graph and basic visualization

[2]:
```
# specify a simple (embedded) graph:
# * list of vertices with positions
# * list of (directed) edges, given as list of indices of involved vertices
#   convention: edge goes from first to second index
pointData=np.array([[0.,0.],[2.,0.],[0.5,0.5],[1.5,0.5],[0.,1.],[2.,1.
 ↪]],dtype=np.double)
edgeData=np.array([[0,1],[0,2],[1,3],[2,3],[2,4],[3,5],[4,5]],dtype=np.int32)

nPoints=pointData.shape[0]
nEdges=edgeData.shape[0]
```

[6]:
```
# basic display of graph: vertices and edges
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1])
lineCollection=matplotlib.collections.
 ↪LineCollection(pointData[edgeData],zorder=-1)
```

```
ax.add_collection(lineCollection)
# usually introducing some convenience functions helps:
#vertices(ax,pointData)
#edges(ax,pointData,edgeData)
plt.tight_layout
plt.show()
```



[7]:
```
pointData[edgeData].shape
# 0: nr of edges
# 1: two vertices per edge
# 2: two coordinates per vertex
```

[7]: (7, 2, 2)

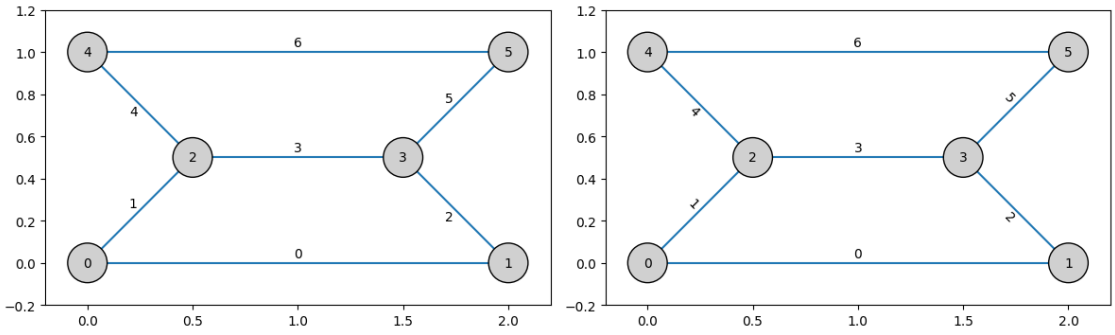
1.2 Simple graph and mesh plotting methods

[9]:
```
from graphplot import *
```

1.3 Annotations

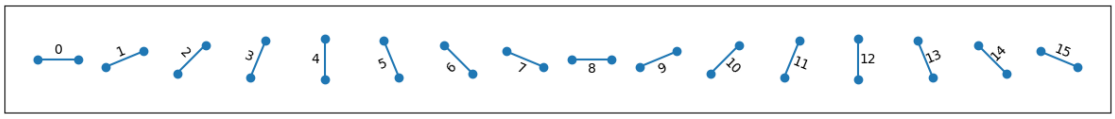
[10]:
```
# now with annotation
fig=plt.figure(figsize=(12,4))

for i,rotate in enumerate([False,True]):
    ax=fig.add_subplot(1,2,i+1,aspect=1.)
    edges(ax,pointData,edgeData)
```

```
    vertices(ax,pointData,annotate=True)
    annotateEdges(ax,pointData,edgeData,rotate=rotate)
    setBoxLimits(ax,pointData,buffer=0.2)

plt.tight_layout()
plt.show()
```



[11]:
```
# show annotation rotation for all edge rotations

# create point and edge data for individual edges at equally spaced different␣
  ↪orientations
nAngles=16
rad=0.3
angles=np.arange(nAngles)/nAngles*360
pointDataTmp=np.zeros((nAngles,2,2),dtype=np.double)
edgeDataTmp=np.zeros((nAngles,2),dtype=int)
for i,angle in enumerate(angles):
    pointDataTmp[i,0,0]=i-rad*np.cos(angle/180*np.pi)
    pointDataTmp[i,0,1]=-rad*np.sin(angle/180*np.pi)
    pointDataTmp[i,1,0]=i+rad*np.cos(angle/180*np.pi)
    pointDataTmp[i,1,1]=+rad*np.sin(angle/180*np.pi)
    edgeDataTmp[i,0]=2*i
    edgeDataTmp[i,1]=2*i+1
pointDataTmp=pointDataTmp.reshape((-1,2))

# visualize, to see how annotation orientation is adjusted
fig=plt.figure(figsize=(12,2))
ax=fig.add_subplot(aspect=1.)
edges(ax,pointDataTmp,edgeDataTmp)
vertices(ax,pointDataTmp,annotate=False)
annotateEdges(ax,pointDataTmp,edgeDataTmp,rotate=True,rotMode=45)
setBoxLimits(ax,pointDataTmp,buffer=0.5)
plt.xticks([])
plt.yticks([])
```

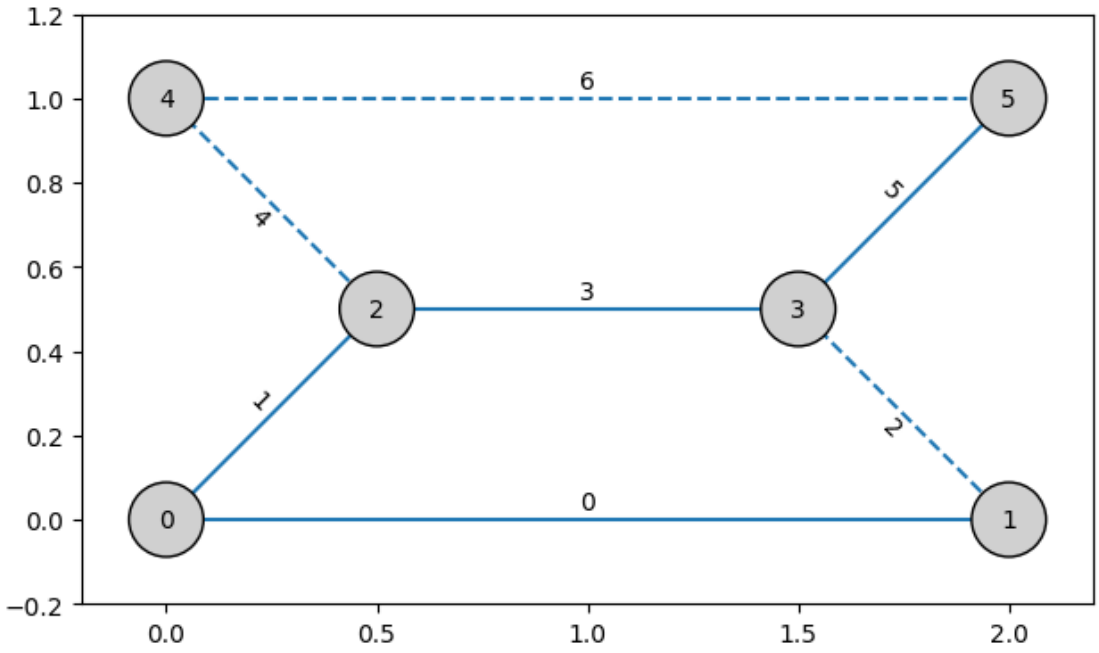
```
plt.tight_layout()
plt.show()
```



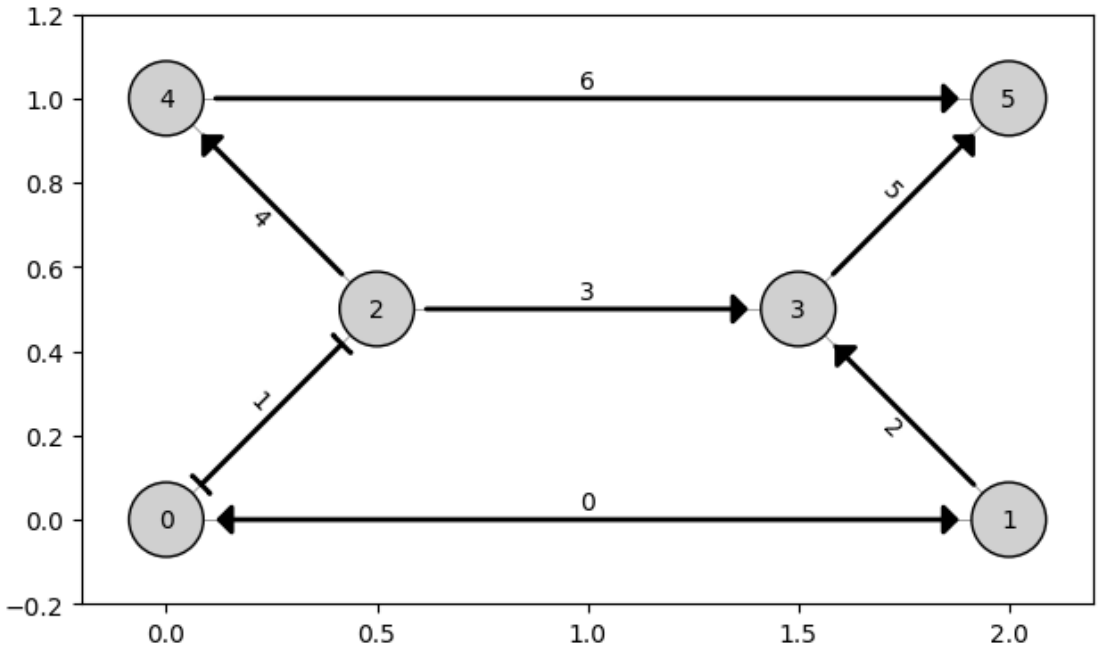
1.4 Edge-types and orientations

[12]:
```
# each edge now has a nominal "type", visualize these as different line styles
edgeTypeList=[0,0,1,0,1,0,1]
```

[13]:
```
# convert types to line styles
typeToLs=["solid","dashed"]
lsList=[typeToLs[i] for i in edgeTypeList]

fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
edges(ax,pointData,edgeData,ls=lsList)
vertices(ax,pointData,annotate=True)
annotateEdges(ax,pointData,edgeData,rotate=True)

setBoxLimits(ax,pointData,buffer=0.2)
plt.tight_layout()
plt.show()
```



[14]:
```
# specify arrow types for each edge, start with default "forward arrows"
orientationCodes=["-|>,head_length={0:s},head_width={0:s}" for i in␣
 ↪range(len(edgeData))]
# try a few different options
orientationCodes[0]="<|-|>,head_length={0:s},head_width={0:s}"
orientationCodes[1]="]-[,widthA={0:s},widthB={0:s}"

fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
vertices(ax,pointData,annotate=True,color="k")
edgesArrows(ax,pointData,edgeData,orientationCodes,shrink=20,size=5,color="k",lw=2)
annotateEdges(ax,pointData,edgeData,rotate=True)
# show full edge lines underneath, in faint style
edges(ax,pointData,edgeData,color="#808080",lw=0.5,zorder=-3)
setBoxLimits(ax,pointData,buffer=0.2)
plt.tight_layout()
plt.show()
```


1.2
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.5
1.0
1.5
2.0
4
5
2
3
0
1
6
4
5
3
1
2
0

[ ]:

[ ]:

[ ]:

2023-06-19_002_Graphs_Flows

June 19, 2023

[3]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline

import colorcet as ccm
from graphplot import *
```

1 Example: Flows on graphs

1.1 Background and simple example

- a common problem is to visualize a flow on a graph (could be a material, current, information, ...)
- flows are a signed (oriented) signal on the edges
- for each vertex, we can sum the total incoming and outgoing fluxes. This can be positive (mass increasing at the vertex), negative (mass decreasing) or zero (mass preserved). This is called the 'divergence' of the flow
- we will think a little bit about how to compute the divergence of a flow, and how to visualize the flow and its divergence

- let (V, E) be a directed graph
- the following matrix is called **incidence matrix** of the graph:

$$A : \mathbb{R}^{|V|\times|E|}, \qquad A_{i,j} = \begin{cases} 1 & \text{if edge } j \text{ leaves vertex } i, \\ -1 & \text{if edge } j \text{ enters vertex } i, \\ 0 & \text{else.} \end{cases}$$

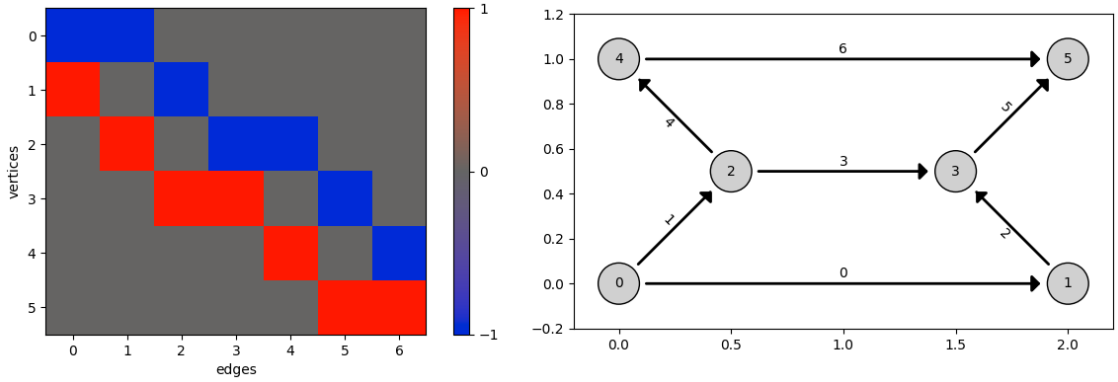
- now let $f \in \mathbb{R}^E$ be a flow on the graph where f_j is the mass flowing on edge j,
- we adopt the convention that $f_j > 0$ if the mass is flowing **along** the orientation of edge j,
- and negative if it flows **against** the orientation.
- now assume we sit at vertex i, how much mass is flowing into and out of this vertex?
- we denote this by m_i (positive if incoming, negative otherwise)
- if edge j is incoming, then f_j will contribute to incoming mass if it is positive, and to outcoming mass if it is negative. note that in this case we have $A_{i,j} = 1$.
- if edge j is outgoing, then the signs are reversed; and in tis case $A_{i,j} = -1$.
- if edge j is not connected to vertex i, then $A_{i,j} = 0$.
- so the total net amount of incoming mass is given by:

$$m_i = \sum_j A_{i,j} f_j = (Af)_i \quad \text{and therefore} \quad m = Af$$

[4]:
```
# specify a simple (embedded) graph:
pointData=np.array([[0.,0.],[2.,0.],[0.5,0.5],[1.5,0.5],[0.,1.],[2.,1.
 ↪]],dtype=np.double)
edgeData=np.array([[0,1],[0,2],[1,3],[2,3],[2,4],[3,5],[4,5]],dtype=np.int32)

nPoints=pointData.shape[0]
nEdges=edgeData.shape[0]
```

[5]:
```
# build the adjacency matrix
# note: only depends on edgeData, NOT on pointData
# in many cases this matrix is large, but sparse.
# then a more efficient implementation should be used
A=np.zeros((nPoints,nEdges))
A[edgeData[:,0],np.arange(nEdges)]=-1
A[edgeData[:,1],np.arange(nEdges)]=1
```

[6]:
```
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
obj=plt.imshow(A,cmap=ccm.cm.CET_D8)
plt.xlabel("edges")
plt.ylabel("vertices")
plt.colorbar(obj,ticks=[-1,0,1])

ax=fig.add_subplot(1,2,2,aspect=1.)
orientationCodes=["-|>,head_length={0:s},head_width={0:s}" for i in␣
 ↪range(len(edgeData))]
edgesArrows(ax,pointData,edgeData,orientationCodes,shrink=20,size=5,color="k",lw=2)
vertices(ax,pointData,annotate=True)
annotateEdges(ax,pointData,edgeData,rotate=True)
setBoxLimits(ax,pointData,buffer=0.2)


plt.tight_layout()
plt.show()
```



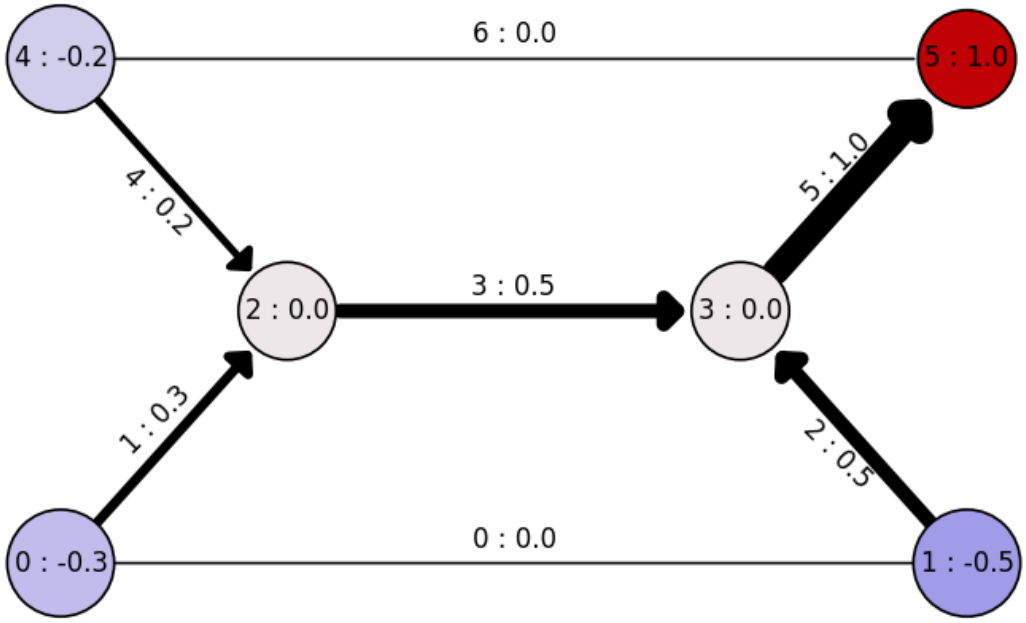
[7]:
```
# now define a flow vector f
f=np.zeros(nEdges,dtype=np.double)
f[1]=0.3
f[4]=-0.2
f[3]=0.5
f[2]=0.5
f[5]=1
```

[8]:
```
# use graph divergence operator to compute sources and sinks of flow
m=A.dot(f)
print(m)
```

```
[-0.3 -0.5  0.   0.  -0.2  1. ]
```

[9]:
```
# arrows and annotations
# now with annotation
fig=plt.figure(figsize=(6,4),dpi=120)
ax=fig.add_subplot()

flowThresh=1E-5
vmax=np.max(np.abs(f))
shrink=20
size=5
lwMax=10
lwMin=1


def getArrowCode(flow):
    if flow>flowThresh:
        return "-|>,head_length={0:s},head_width={0:s}"
    elif flow<-flowThresh:
        return "<|-,head_length={0:s},head_width={0:s}"
    else:
```

```
        return "-"

orientationCodes=[getArrowCode(flow) for flow in f]
#edgeColors=[ccm.cm.kr(flow/vmax) for flow in np.abs(f)]
#edgeColors=[ccm.cm.gray(0.5-0.5*flow/vmax) for flow in np.abs(f)]
edgeColors=[ccm.cm.gray(0) for flow in np.abs(f)]

for (x,y),o,col,flow in zip(pointData[edgeData],orientationCodes,edgeColors,f):
    a=matplotlib.patches.FancyArrowPatch(x,y,
            arrowstyle=o.format(str(size)),
            shrinkA=shrink, shrinkB=shrink,
            color=col,lw=(lwMax-lwMin)*np.abs(flow)/vmax+lwMin)
    ax.add_artist(a)


vmax=np.max(np.abs(m))
# annotate vertices
for i,(x,s) in enumerate(zip(pointData,m)):
    props = dict(boxstyle="circle,pad=0.25",fc=ccm.cm.CET_D1(0.5+0.5*s/vmax))
    ax.text(x[0],x[1],"{:d} : {:.1f}".
 ↪format(i,m[i]),ha="center",va="center",bbox=props,)

# annotate edges
msgList=["{:d} : {:.1f}".format(i,np.abs(f[i])) for i in range(nEdges)]
annotateEdges(ax,pointData,edgeData,textList=msgList,rotate=True,rotMode=90,dist=15)

setBoxLimits(ax,pointData,buffer=0.2)
plt.axis("off")
plt.tight_layout()
plt.show()
```


4 : -0.2
6 : 0.0
5 : 1.0
4 : 0.2
5 : 1.0
2 : 0.0
3 : 0.5
3 : 0.0
1 : 0.3
2 : 0.5
0 : -0.3
0 : 0.0
1 : -0.5

[ ]:

2023-06-19_003_Graphs_Stress

June 19, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline

from matplotlib.animation import FuncAnimation
matplotlib.rc('animation',html='html5')
import colorcet as ccm
from graphplot import *
```

1 Example: Stress in a bridge

1.1 Building basic grid graph (representing a ”bridge”)

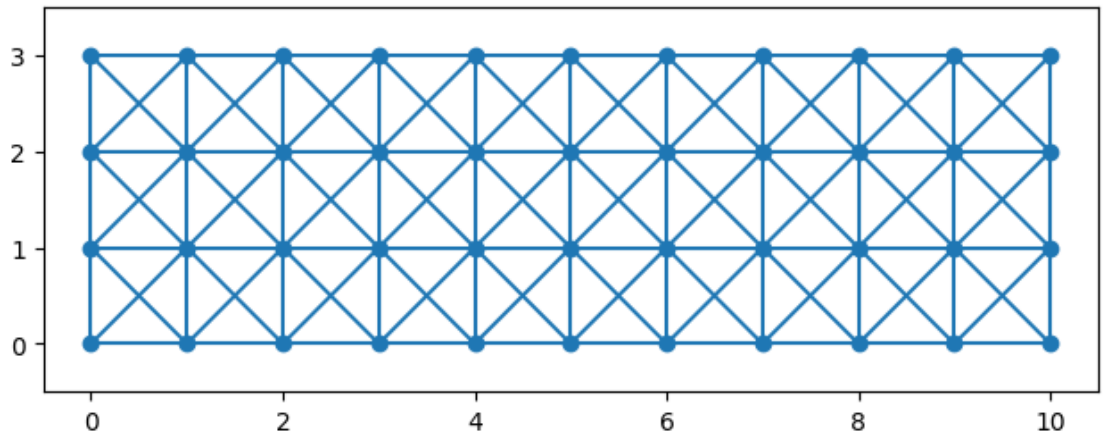
[2]:
```
def getPoslistNCube(shape,dtype=np.double):
    """Create list of positions in an n-dimensional cuboid of size shape."""
    ndim=len(shape)

    axGrids=[np.arange(i,dtype=dtype) for i in shape]
    prePos=np.array(np.meshgrid(*axGrids,indexing='ij'),dtype=dtype )
    # the first dimension of prepos is the dimension of the  posvector, the␣
 ↪successive dimensions are in the cube
    # so need to move first axis to end, and then flatten
    pos=np.rollaxis(prePos,0,ndim+1)
    # flattening
    newshape=(-1,ndim)
```

```
    return (pos.reshape(newshape)).copy()
```

[3]:
```
# build "rectangular" graphs
# how many rows and cols?
nRows=4
nCols=11
# spatial dimension
dim=2

# first create the vertex set
nList=[nCols,nRows]
posList=getPoslistNCube((nCols,nRows),dtype=np.int32)
nPoints=posList.shape[0]


# now add edges: we want to add edges according to an eight-neighbourhood
# the following gives the relative positions of all desired neighbours
# note: we only need "half" of them, otherwise each edge would be added twice
if False:
    # four neighbourhood
    relEdgeList=np.array([[1,0],[0,1]],dtype=np.int32)
if True:
    # eight neighbourhood
    relEdgeList=np.array([[1,0],[0,1],[1,1],[-1,1]],dtype=np.int32)

nRelEdges=relEdgeList.shape[0]

# add rel edges to all points, to get list of potential targets of edges
edgeTargets=posList.reshape((nPoints,1,dim))+relEdgeList.
 ↪reshape((1,nRelEdges,dim))
edgeInits=np.zeros((nPoints,nRelEdges,dim),dtype=np.int32)
for i in range(nRelEdges):
    edgeInits[:,i,:]=posList


# now identify invalid edges
keep=np.full((nPoints,nRelEdges),True)
for i in range(dim):
    keep*=(edgeTargets[:,:,i]<nList[i])
    keep*=(edgeTargets[:,:,i]>=0)

# now flatten edgeTarget and Init lists, then restrict to those we want to keep
edgeTargets=edgeTargets.reshape((-1,dim))
edgeInits=edgeInits.reshape((-1,dim))
keep=keep.ravel()
edgeTargets=edgeTargets[keep]
edgeInits=edgeInits[keep]
```

```
# now convert coordinates to edge indices
idxStrides=np.array([nRows,1],dtype=np.int32)
edgeTargets=np.einsum(edgeTargets,[0,1],idxStrides,[1],[0])
edgeInits=np.einsum(edgeInits,[0,1],idxStrides,[1],[0])

nEdges=edgeTargets.shape[0]
edgeData=np.zeros((nEdges,2),dtype=np.int32)
edgeData[:,0]=edgeInits
edgeData[:,1]=edgeTargets

edgeLengths=np.linalg.norm(posList[edgeData[:,0]]-posList[edgeData[:,1]],axis=1)

# add reversed edges
#edgeTargets,edgeInits=np.concatenate((edgeTargets,edgeInits)),np.
 ↪concatenate((edgeInits,edgeTargets))
```

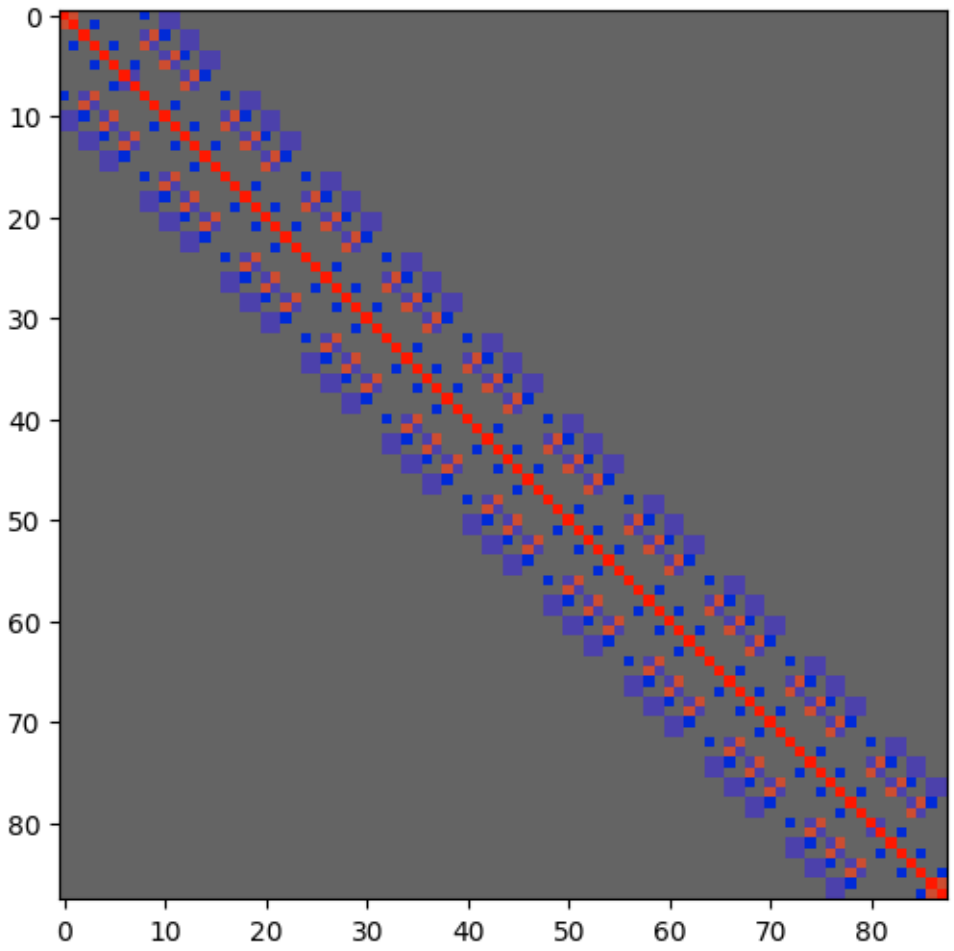
[4]:
```
# basic display of graph: vertices and edges
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
vertices(ax,posList)
edges(ax,posList,edgeData)
setBoxLimits(ax,posList,buffer=0.5)
plt.tight_layout()
plt.show()
```



1.2 Graph Laplacian

[5]:
```
# build a particular version of the graph Laplacian
# that encodes first order deformation forces
# see simulation-forces-in-graph.pdf for some details

L=np.zeros((nPoints,dim,nPoints,dim),dtype=np.double)
for i in range(dim):
    for j in range(dim):
        L[edgeData[:,0],i,edgeData[:,1],j]=-(posList[edgeData[:,0],i]-\
                posList[edgeData[:,1],i])*\
                (posList[edgeData[:,0],j]-posList[edgeData[:,1],j])/
 ↪edgeLengths**2
# add reversed contributions
L+=L.transpose((2,1,0,3))

# diagonal entries
rng=np.arange(nPoints)
L[rng,:,rng,:]=-np.sum(L,axis=2)

# flatten
L=L.reshape((nPoints*dim,nPoints*dim))
```

[6]:
```
fig=plt.figure(figsize=(6,6))
vmax=0.2*np.max(np.abs(L))
plt.imshow(L,cmap=ccm.cm.CET_D8,vmin=-vmax,vmax=vmax)
plt.show()
```

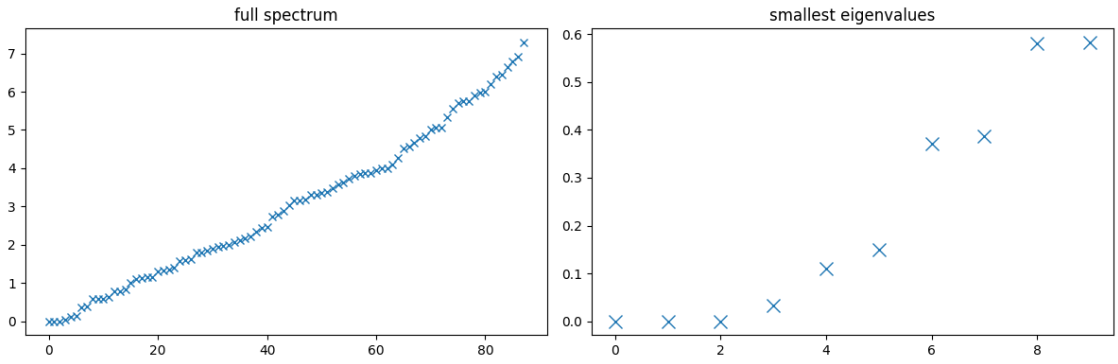


1.2.1 Eigenbasis of graph Laplacian

[7]:
```
eigData=np.linalg.eigh(L)
# some reordering:
# eigval: eigenvalues from small to large (more appropriate here)
# eigvec: eigenvectors for eigenvalues are in rows
eigval=eigData[0][::]
eigvec=(eigData[1].transpose())[::]
```

[8]:
```
# spectrum of Laplacian
fig=plt.figure(figsize=(12,4))
fig.add_subplot(1,2,1)
plt.title("full spectrum")
plt.plot(eigval,lw=0,marker="x")

fig.add_subplot(1,2,2)
```

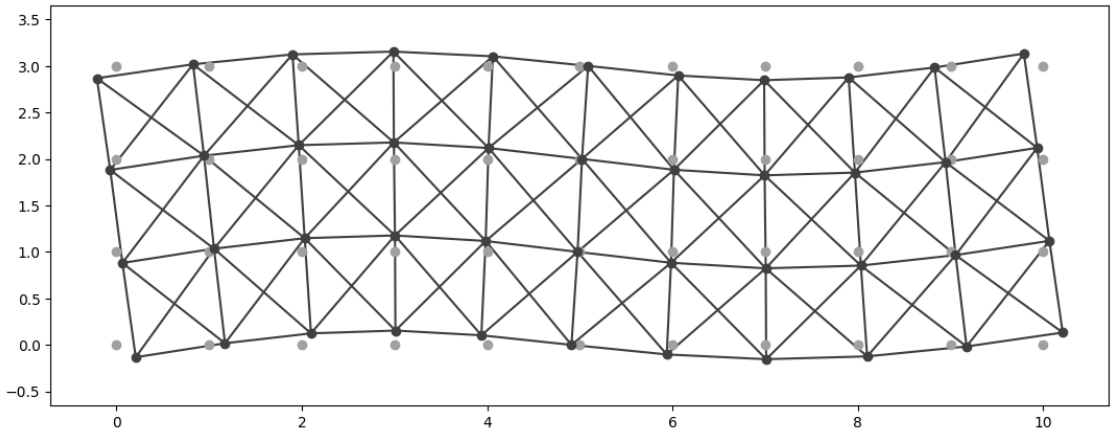
```
plt.title("smallest eigenvalues")
plt.plot(eigval[:10],lw=0,marker="x",markersize=10)

plt.tight_layout()
plt.show()
```



[9]:
```
# note: first three eigenvalues are zero.
# these correspond to "deformations"
# that do not change any pairwise lenghts
```

1.2.2 Visualize eigenmodes

[11]:
```
# static plot of "deformed graph"
i=5
mode=1.0*eigvec[i].reshape((-1,dim))
# visualize eigenmodes
fig=plt.figure(figsize=(12,12*(nRows/nCols)))
ax=fig.add_subplot(aspect=1.)
vertices(ax,posList,color="#a0a0a0")
vertices(ax,posList+mode,color="#404040")
edges(ax,posList+mode,edgeData,color="#404040")
setBoxLimits(ax,posList+mode,buffer=0.5)
plt.tight_layout()
plt.show()
```



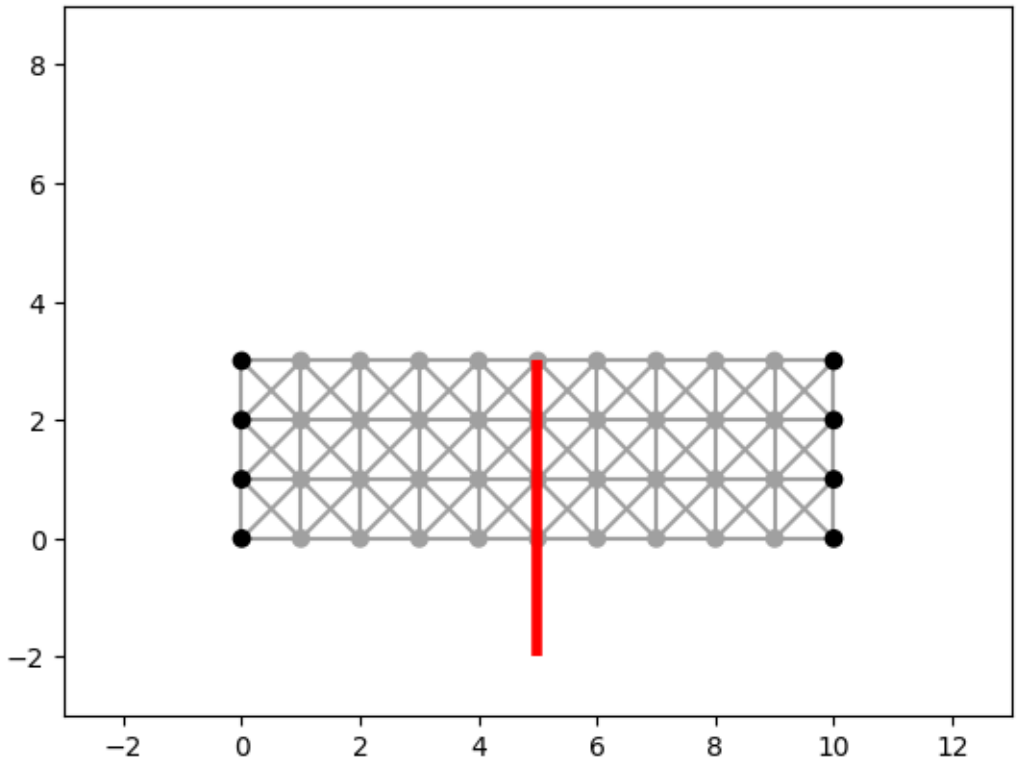
[12]:
```
# animation as oscillation
# will explain matplotlib animation later in more detail
i=3
mode=0.6*eigvec[i].reshape((-1,dim))

fig=matplotlib.figure.Figure(figsize=(12,4))
ax=fig.add_subplot(aspect=1.)

pltobj_pts = ax.scatter([], [],color="k")
pltobj_lineCollection=matplotlib.collections.
 ↪LineCollection([],zorder=-1,color="k")
ax.add_collection(pltobj_lineCollection)
ax.set_xlim([-0.5,nCols-0.5])
ax.set_ylim([-0.5,nRows-0.5])


def update(frame):
    data=posList+np.sin(frame)*mode
    pltobj_pts.set_offsets(data)
    pltobj_lineCollection.set_paths(data[edgeData])
    return pltobj_pts,pltobj_lineCollection,

ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*np.pi, 20),
        blit=True,interval=1000/20)

matplotlib.rc('animation',html='jshtml')
ani
```

[12]:
```
<matplotlib.animation.FuncAnimation at 0x7f17229618a0>
```

1.2.3 Application of external force, while keeping some vertices fixed

[13]:
```
V=np.zeros((nPoints,dim),dtype=np.double)
# apply a downward force on the upper middle point
V[nCols//2*nRows+nRows-1,1]=-1.
# keep vertices in first and last column fixed ("end points of bridge")
freeVertices=np.ones((nCols,nRows),dtype=bool)
freeVertices[0,:]=False
freeVertices[-1,:]=False
freeVertices=freeVertices.ravel()
nPointsFree=np.sum(freeVertices)
```

[14]:
```
mode=5.*V
# visualize forces
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
colList=["k" for _ in range(nPoints)]
for i in range(nPoints):
    if freeVertices[i]==True:
        colList[i]="#a0a0a0"
vertices(ax,posList,color=colList)

lineData=np.stack((posList,posList+mode),axis=1)
# simple plot as line collection
lc=matplotlib.collections.LineCollection(lineData,zorder=2,color="r",lw=4)
ax.add_collection(lc)


edges(ax,posList,edgeData,color="#a0a0a0")
plt.xlim([-3,nCols-1+3])
plt.ylim([-3,nRows-1+6])

plt.show()
```

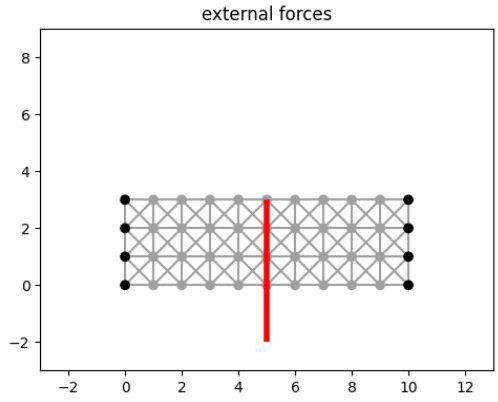


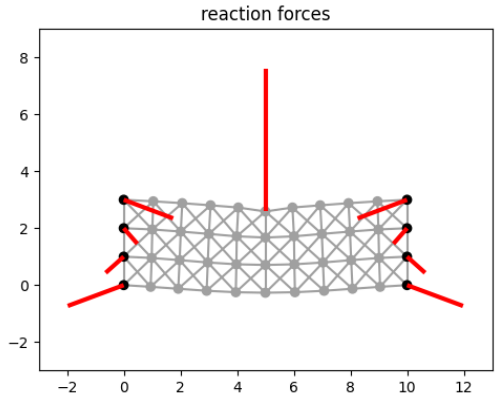
[15]:
```
VHat=V[freeVertices]
LHat=L.reshape((nPoints,dim,nPoints,dim))
LHat=LHat[freeVertices]
LHat=LHat[:,:,freeVertices]
LHat=LHat.reshape((nPointsFree*dim,nPointsFree*dim))

deformationData=np.linalg.lstsq(LHat,VHat.ravel(),rcond=None)

deformationHat=deformationData[0]
deformation=np.zeros((nPoints,dim),dtype=np.double)
deformation[freeVertices,:]=deformationHat.reshape((nPointsFree,dim))
```

[16]:
```
# what are the resulting forces of the bridge deformation
force=-L.dot(deformation.ravel()).reshape((-1,dim))
```

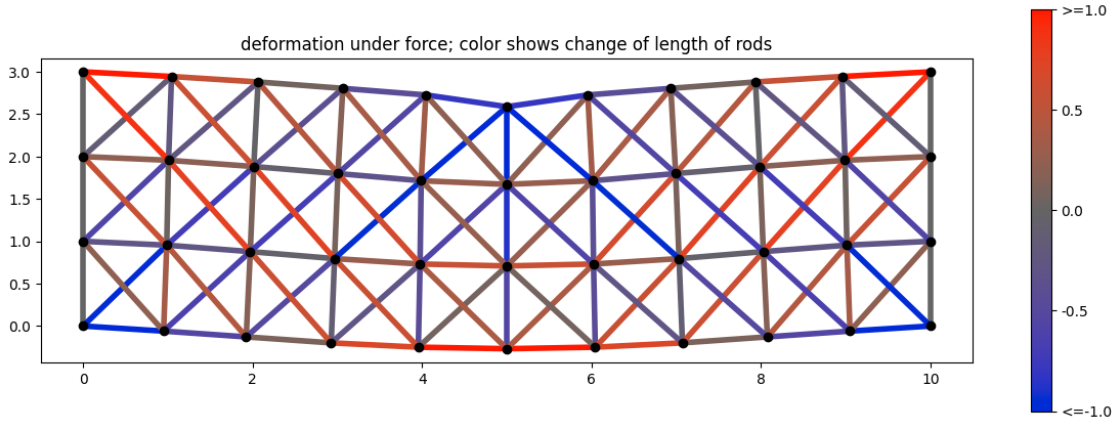
[17]:
```
# visualize external and reaction forces
fig=plt.figure(figsize=(12,4))


ax=fig.add_subplot(1,2,1,aspect=1.)
mode=5.*V
plt.title("external forces")
```

```
colList=["k" for _ in range(nPoints)]
for i in range(nPoints):
    if freeVertices[i]==True:
        colList[i]="#a0a0a0"
vertices(ax,posList,color=colList)

lineData=np.stack((posList,posList+mode),axis=1)
# simple plot as line collection
lc=matplotlib.collections.LineCollection(lineData,zorder=2,color="r",lw=4)
ax.add_collection(lc)


edges(ax,posList,edgeData,color="#a0a0a0")
plt.xlim([-3,nCols-1+3])
plt.ylim([-3,nRows-1+6])


ax=fig.add_subplot(1,2,2,aspect=1.)
modeForce=5.*force
mode=0.2*deformation
plt.title("reaction forces")
colList=["k" for _ in range(nPoints)]
for i in range(nPoints):
    if freeVertices[i]==True:
        colList[i]="#a0a0a0"
vertices(ax,posList+mode,color=colList)

lineData=np.stack((posList+mode,posList+mode+modeForce),axis=1)
lc=matplotlib.collections.LineCollection(lineData,zorder=1,color="r",lw=3)
ax.add_collection(lc)

edges(ax,posList+mode,edgeData,color="#a0a0a0")
plt.xlim([-3,nCols-1+3])
plt.ylim([-3,nRows-1+6])


plt.tight_layout()
plt.show()
```





[18]:
```
mode=0.2*deformation.reshape((-1,dim))
posDef=posList+mode
edgeLengthsNew=np.linalg.norm(posDef[edgeData[:,0]]-posDef[edgeData[:
 ↪,1]],axis=1)
edgeLengthsDelta=edgeLengthsNew-edgeLengths

vmax=0.5*np.max(np.abs(edgeLengthsDelta))
colfun=ccm.cm.CET_D8
edgeColors=colfun(np.clip((1+edgeLengthsDelta/vmax)/2,0,1))

fig=plt.figure(figsize=(12,12*nRows/nCols))
ax=fig.add_subplot(aspect=1.)
vertices(ax,posList+mode,c="k")
edges(ax,posList+mode,edgeData,colors=edgeColors,lw=4)
ticks=np.linspace(-1,1,5)
cbar=fig.colorbar(
        cm.ScalarMappable(norm=matplotlib.colors.Normalize(vmin=-1, vmax=1,␣
 ↪clip=False), cmap=colfun),
        ax=ax, ticks=ticks)
# manually change labels of colorbar ticks
ticksLabels=[str(i) for i in ticks]
ticksLabels[0]="<="+ticksLabels[0]
ticksLabels[-1]=">="+ticksLabels[-1]
cbar.ax.set_yticklabels(ticksLabels)


plt.title("deformation under force; color shows change of length of rods")
plt.tight_layout()
plt.show()
```



[19]:
```
mode=0.1*deformation.reshape((-1,dim))

# try to estimate maximal deformation
data=posList+mode
edgeLengthsNew=np.linalg.norm(data[edgeData[:,0]]-data[edgeData[:,1]],axis=1)
edgeLengthsDelta=edgeLengthsNew-edgeLengths
vmax=0.5*np.max(np.abs(edgeLengthsDelta))
colfun=ccm.cm.CET_D8


fig=matplotlib.figure.Figure(figsize=(12,4))
ax=fig.add_subplot(aspect=1.)

pltobj_pts = ax.scatter([], [],c="k")
#lineCollection=matplotlib.collections.
 ↪LineCollection(posList[edgeData],zorder=-1)
pltobj_lineCollection=matplotlib.collections.
 ↪LineCollection([[[0,0],[1,1]]],zorder=-1,lw=4)
ax.add_collection(pltobj_lineCollection)
ax.set_xlim([-0.5,nCols-0.5])
ax.set_ylim([-0.5,nRows-0.5])

#def init():
#    ax.set_xlim([-0.5,nCols-0.5])
#    ax.set_ylim([-0.5,nRows-0.5])
#    return ln,

def update(frame):
    data=posList+np.sin(frame)**2*mode

    edgeLengthsNew=np.linalg.norm(data[edgeData[:,0]]-data[edgeData[:
 ↪,1]]],axis=1)
```

```
    edgeLengthsDelta=edgeLengthsNew-edgeLengths

    edgeColors=colfun(np.clip((1+edgeLengthsDelta/vmax)/2,0,1))


    pltobj_pts.set_offsets(data)
    pltobj_lineCollection.set_paths(data[edgeData])
    pltobj_lineCollection.set_color(edgeColors)
    return pltobj_pts,pltobj_lineCollection,
    #ln2, = plt.plot([], [], 'ro')
    #return ln2,
#ani = FuncAnimation(fig, update, frames=np.linspace(0, 2*np.pi, 20),
#                    init_func=init, blit=True,interval=1000/20)
ani = FuncAnimation(fig, update, frames=np.linspace(0, np.pi, 20),
        blit=True,interval=1000/20*2)

matplotlib.rc('animation',html='jshtml')
ani
```

[19]:
```
<matplotlib.animation.FuncAnimation at 0x7f1720cdbb50>
```

[ ]:

2023-06-19_010_Meshes_Part-01

June 19, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline

from matplotlib.animation import FuncAnimation
matplotlib.rc('animation',html='html5')
import colorcet as ccm
from graphplot import *
```

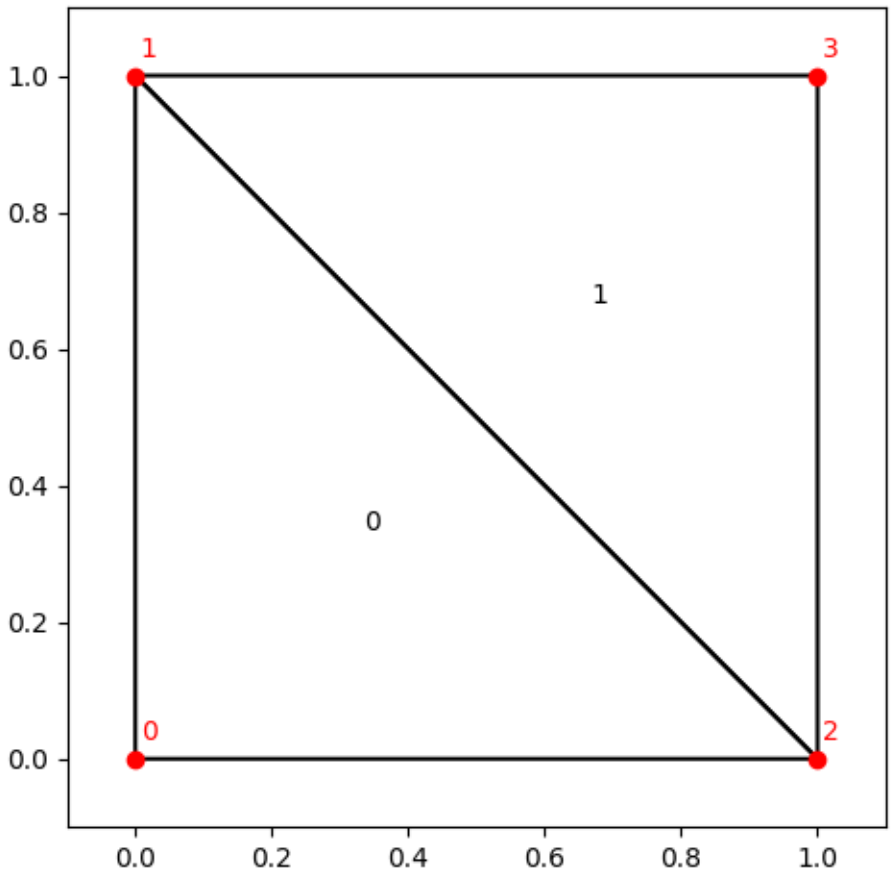
[2]:
```
import FEM
```

0.1 Points and triangles

[3]:
```
# specify domain by giving its extremal points
points=np.array([[0,0],[0,1],[1,0],[1,1]],dtype=np.double)
dim=points.shape[1]

# manually create simple triangulation
pointData=points.copy()
triangleData=np.array([[1,0,2],[2,3,1]],dtype=np.int32)
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[4]:
```
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="r")
```

```
# annotate points
for i,x in enumerate(pointData):
    ax.annotate(i,x,(5,5),textcoords="offset␣
 ↪points",va="bottom",ha="center",color="r")

for i,t in enumerate(triangleData):
    cycleData=np.concatenate((t,t[[0]]))
    plt.plot(pointData[cycleData,0],pointData[cycleData,1],c="k",zorder=-1)
    center=np.mean(pointData[t],axis=0)
    ax.annotate(i,center,color="k")

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.tight_layout()
plt.show()
```



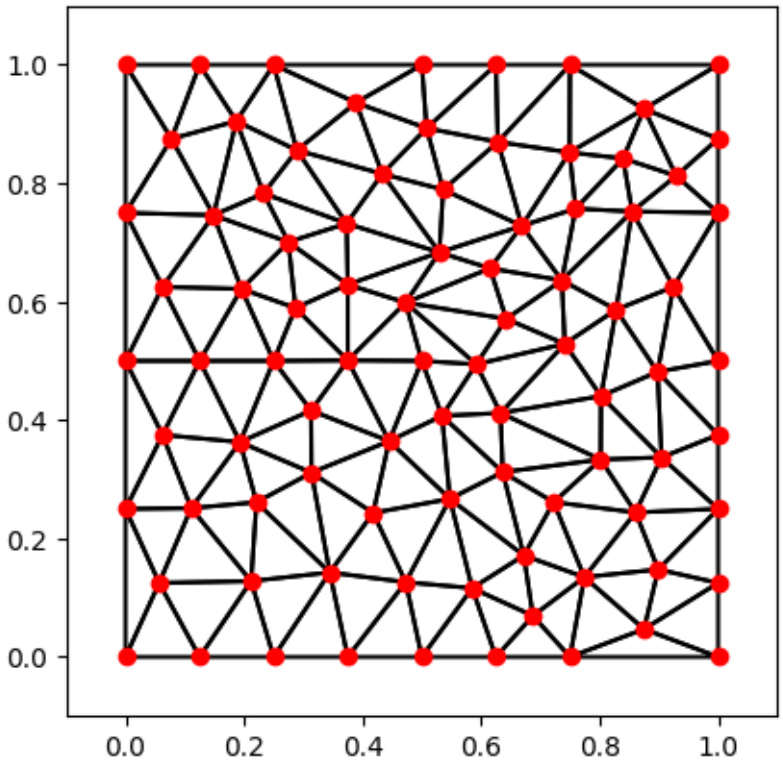
[5]:
```
triangleData
```

[5]:
```
array([[1, 0, 2],
       [2, 3, 1]], dtype=int32)
```

0.1.1 Load a finer triangulation

[5]:
```
# create/load a finer example mesh
if False:
    import lib.Triangle as Triangle

    pointData,triangleData=Triangle.createMesh(points,maxArea=0.01)
    np.savez("triangulation.npz",pointData=pointData,triangleData=triangleData)
else:
    data=np.load("triangulation.npz")
    pointData=data["pointData"]
    triangleData=data["triangleData"]
    data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[7]:
```
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="r")

for i,t in enumerate(triangleData):
    cycleData=np.concatenate((t,t[[0]]))
    plt.plot(pointData[cycleData,0],pointData[cycleData,1],c="k",zorder=-1)

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.show()
```



[8]:
```
nTriangles
```

[8]:
```
150
```

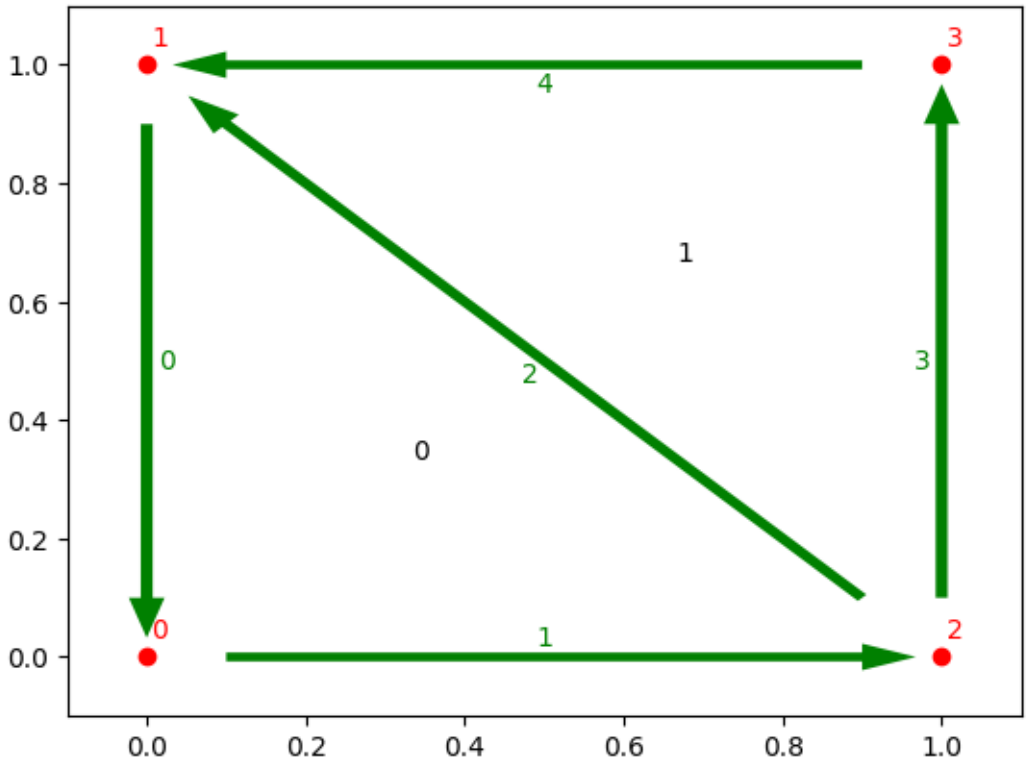
0.2 Edges

[9]:
```
# note: currently we draw every edge twice, as part of the two adjacent␣
 ↪triangles
# it is almost always useful to explicitly extract a list of edges
# and the relation between edges and triangles
```

[10]:
```
# demonstrate this on simple triangulation
pointData=points.copy()
triangleData=np.array([[1,0,2],[2,3,1]],dtype=np.int32)
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

[11]:
```
# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[12]:
```
# draw mesh with annotated edges
fig=plt.figure()
ax=fig.add_subplot()
ax.scatter(pointData[:,0],pointData[:,1],c="r")
# annotate points
for i,x in enumerate(pointData):
    ax.annotate(i,x,(5,5),textcoords="offset␣
 ↪points",va="bottom",ha="center",color="r")

# annotate triangle centers
for i,t in enumerate(triangleData):
    center=np.mean(pointData[t],axis=0)
    ax.annotate(i,center,color="k")

# draw and annotate edges
for i,e in enumerate(edgeData):
    x=pointData[e[0]]
    y=pointData[e[1]]
    # implement some manual shrinking
    shrink=0.1
    a=(1-shrink)*x+shrink*y
    delta=(1-2*shrink)*(y-x)

    ax.arrow(a[0],a[1],delta[0],delta[1],width=0.015,lw=0,color="g")

    orientation=delta
    flip=[-orientation[1],orientation[0]]
    flip=flip/np.linalg.norm(flip)
    midpoint=0.5*x+0.5*y
    ax.annotate(i,midpoint,10*flip,textcoords="offset␣
 ↪pixels",ha="center",va="center",c="g")
buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.show()
```



[13]:
```
# edge data: list of all edges in the triangulation
# as for graphs: each edge is given by indices of two connected points
edgeData
```

[13]:
```
array([[1, 0],
       [0, 2],
       [2, 1],
       [2, 3],
       [3, 1]], dtype=int32)
```

[14]:
```
# triangle-edge adjacency data:
# for each triangle: what are the indices of the edges that represent its sides
etAdjacencyData
```

[14]:
```
array([[0, 1, 2],
       [3, 4, 2]], dtype=int32)
```

[15]:
```
# orientation data:
# each triangle has an orientation (counter-clockwise)
# is the orientation of a boundary edge aligned (+1) or opposite (-1) ?
etAdjacencyDataOrientation
```

[15]:
```
array([[ 1,  1,  1],
       [ 1,  1, -1]], dtype=int32)
```

0.3 Plotting functions

[16]:
```
# again: create/load a finer example mesh
data=np.load("triangulation.npz")
pointData=data["pointData"]
triangleData=data["triangleData"]
data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]
```

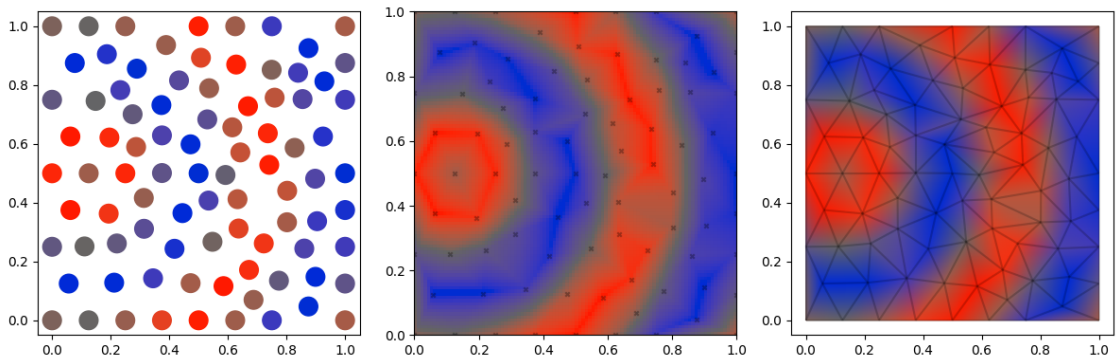
[17]:
```
# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[18]:
```
# compute some function on the vertices
x=pointData[:,0]
y=pointData[:,1]

f=np.sin(((x-0.1)**2+(y-0.5)**2)**0.5*2*np.pi*2)
vmin=-1
vmax=1
f=(f-vmin)/(vmax-vmin)
colfun=ccm.cm.CET_D8
```

0.3.1 Plotting in 2d, with color

[19]:
```
fig=plt.figure(figsize=(12,4))

ax=fig.add_subplot(1,3,1,aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c=colfun(f),s=200)

ax=fig.add_subplot(1,3,2,aspect=1.)
nPts=100
xGrid=yGrid=np.linspace(0,1,num=nPts)
XGrid,YGrid=np.meshgrid(xGrid,yGrid)
dat=scipy.interpolate.griddata((pointData[:,0],pointData[:,1]),f,(XGrid.
  ↪ravel(),YGrid.ravel()),method="linear")
ax.imshow(dat.reshape((nPts,nPts)),extent=(0,1,0,1),origin="lower",cmap=colfun)
ax.scatter(pointData[:,0],pointData[:,1],color=(0,0,0,0.3),marker="x",s=10)
```

```
ax=fig.add_subplot(1,3,3,aspect=1.)
tri=matplotlib.tri.Triangulation(pointData[:,0],pointData[:,1],triangleData)
ax.tripcolor(tri,f,shading='gouraud',cmap=colfun)
lineCollection=matplotlib.collections.
 ↪LineCollection(pointData[edgeData],zorder=2,color="k",alpha=0.2)
ax.add_collection(lineCollection)


plt.tight_layout()
plt.show()
```

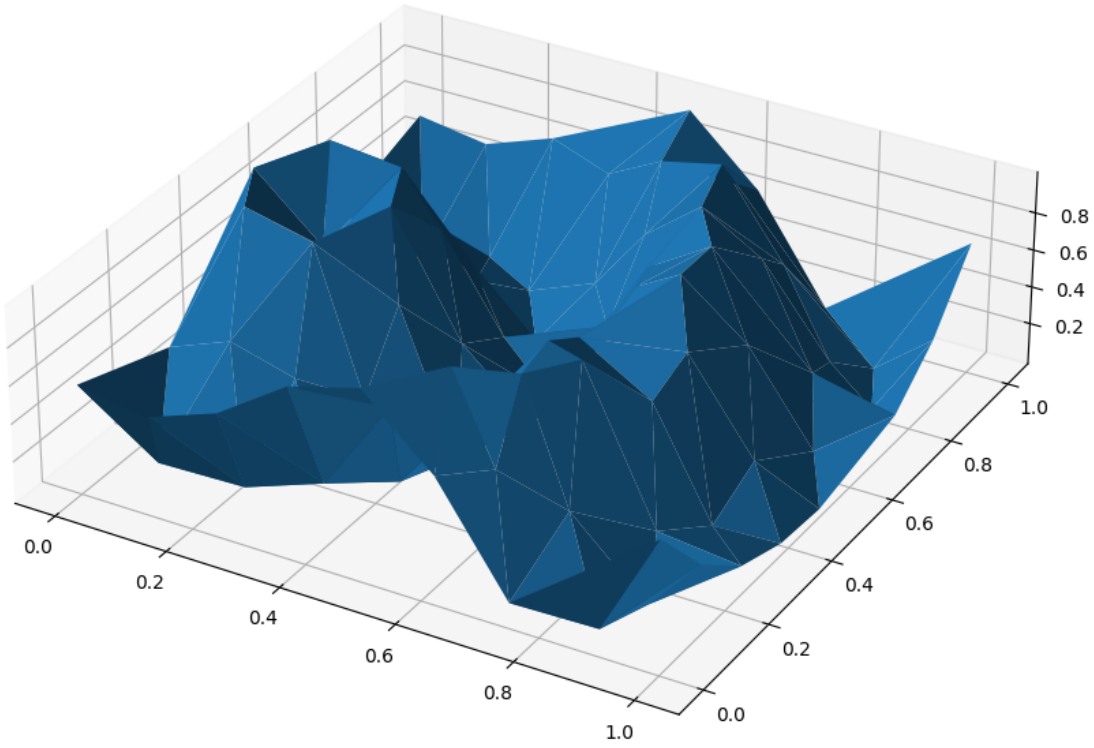


[20]:
```
# recall the interpolation example from the scatter plot session:
# this is one of the ways to handle this, if mesh information is available
```

0.3.2 Plotting in 3d

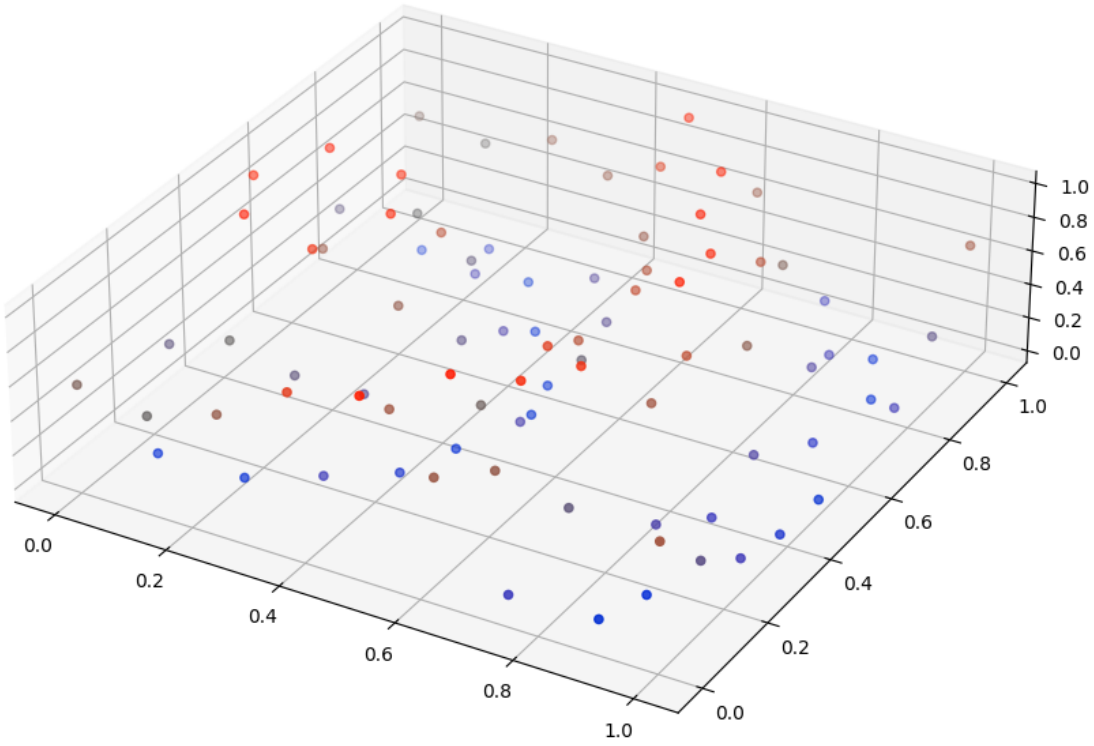
[21]:
```
%matplotlib inline
fig = plt.figure(figsize=(8,8))

ax = fig.add_subplot(projection='3d')

# uniform color with shading
ax.plot_trisurf(pointData[:,0], pointData[:,1], f, triangles=triangleData)

# color by height
#   only works approximately, since matplotlib can only assign one color to␣
 ↪each triangle
#   smooth color interpolation within each triangle is not available in 3d, but␣
 ↪is available in many
#   other plotting libraries / environments
#   also, matplotlib cannot do light-shading with variable colors, so we lose a␣
 ↪lot of the 3d impression
```

```
#   other plotting libraries are more powerful in this respect
#ax.plot_trisurf(pointData[:,0], pointData[:,1], f, triangles=triangleData,␣
 ↪cmap=colfun)

ax.set_box_aspect((1,1,0.3))

plt.tight_layout()
plt.show()
```



[30]:
```
# for comparison: still much better than only the points
# adding triangles is the 2d/3d equivalent of adding lines in a scatter plot
%matplotlib inline
fig = plt.figure(figsize=(8,8))
```

```
ax = fig.add_subplot(projection='3d')

# uniform color with shading
ax.scatter(pointData[:,0], pointData[:,1], f, c=f,cmap=colfun)


ax.set_box_aspect((1,1,0.3))

plt.tight_layout()
plt.show()
```



[ ]:

2023-06-19_010_Meshes_Part-02

June 19, 2023

[1]:
```
import numpy as np
import scipy
import imageio

import matplotlib
import matplotlib.pyplot as plt
import matplotlib.cm as cm

matplotlib.rc('image', interpolation='nearest')
matplotlib.rc('figure',facecolor='white')
matplotlib.rc('image',cmap='viridis')
colors=plt.rcParams['axes.prop_cycle'].by_key()['color']
%matplotlib inline

from matplotlib.animation import FuncAnimation
matplotlib.rc('animation',html='html5')
import colorcet as ccm
from graphplot import *
```

[2]:
```
import FEM as FEM
```

1 Meshes: Regions and boundaries

- intution: every surface has a boundary, which is a contour
- every path has a boundary, which consists of start and endpoint
- closed surfaces and cycles have an ”empty” boundary
- this is the beginning of a whole branch of pure and abstract mathematics
- but now and then these questions arise in our ”everyday research”, then it is helpful to have heard about these ideas

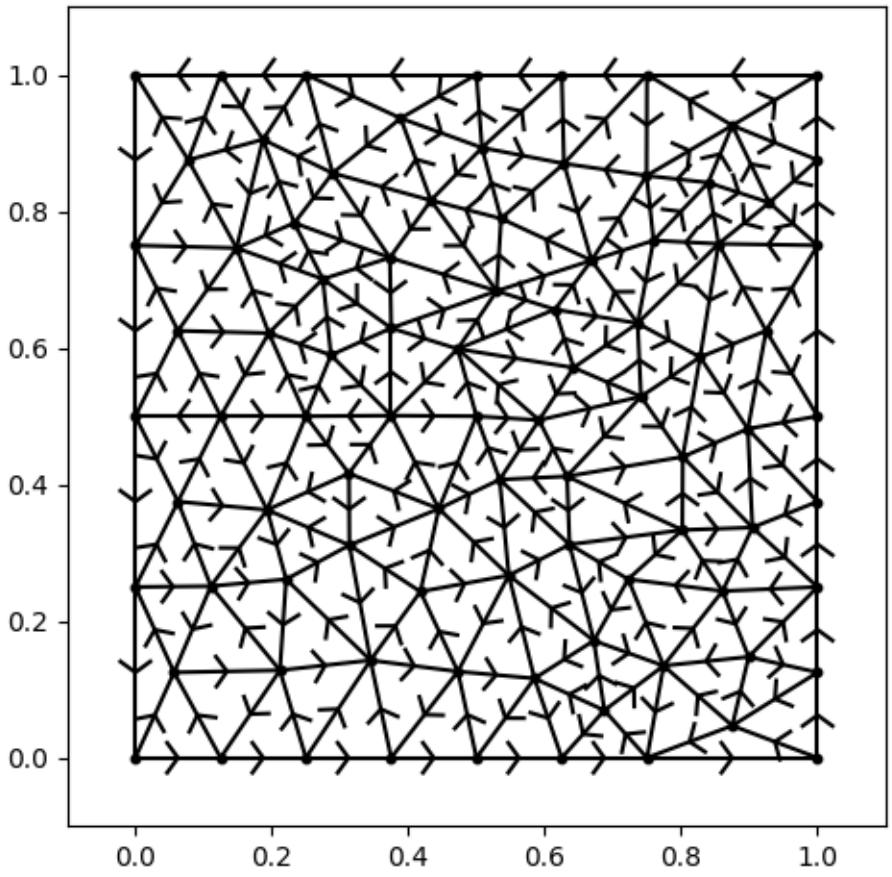
1.1 Setup of a simple example mesh

[3]:
```
# again: create/load a finer example mesh
data=np.load("triangulation.npz")
pointData=data["pointData"]
triangleData=data["triangleData"]
data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]

# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[4]:
```
def␣
 ↪drawEdges(ax,pointData,edgeData,colors="k",drawOrientations=False,arrowScale=0.
 ↪03,**kwargs):
    lineCollection=matplotlib.collections.
 ↪LineCollection(pointData[edgeData],color=colors,**kwargs)
    ax.add_collection(lineCollection)

    # draw small (manual) on edges
    if drawOrientations:
        edgeCoords=pointData[edgeData]
        midPoints=np.mean(edgeCoords,axis=1)
        orientations=edgeCoords[:,1,:]-edgeCoords[:,0,:]
        norms=np.linalg.norm(orientations,axis=1)
        orientations=np.einsum(orientations,[0,1],1./norms,[0],[0,1])
        for phi in [0.7*np.pi,-0.7*np.pi]:
            R=np.array([[np.cos(phi),-np.sin(phi)],[np.sin(phi),np.
 ↪cos(phi)]],dtype=np.double)
            arr=np.einsum(R,[0,1],orientations,[2,1],[2,0])
            lines=np.zeros((nEdges,2,2))
            lines[:,0,:]=midPoints
            lines[:,1,:]=midPoints+arrowScale*arr
            lineCollection=matplotlib.collections.
 ↪LineCollection(lines,color=colors,**kwargs)
            ax.add_collection(lineCollection)
```

[5]:
```
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="k",s=10)

drawEdges(ax,pointData,edgeData,colors="k",drawOrientations=True,arrowScale=0.
 ↪03)
#ax.scatter(midPoints[:,0],midPoints[:,1])
```

```
buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.tight_layout()
plt.show()
```



1.2 Representing regions

- now we can represent 2d regions that are given as unions of triangles
- we simply use an indicator vector: 1 if triangle is part of region, 0 otherwise

1.2.1 Example: single triangle

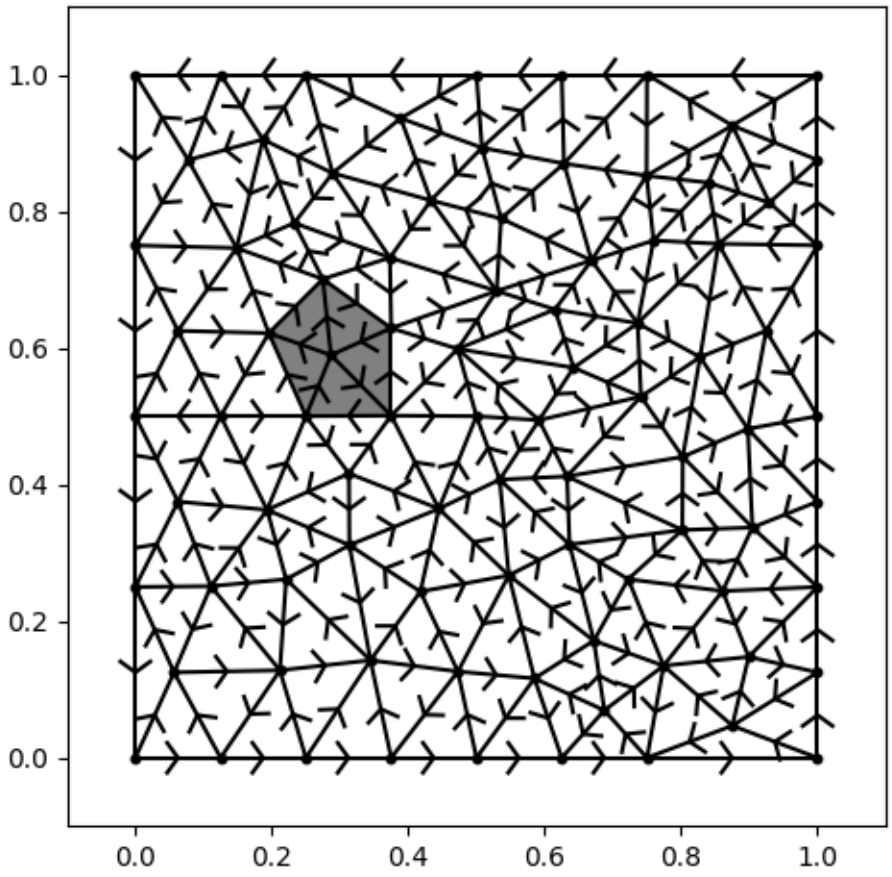
[6]:
```
# allocate an empty region, then set some index to True
region=np.zeros((nTriangles,),dtype=bool)
region[100]=True
```

1.2.2 Example: a few more triangles

[9]:
```
# allocate an empty region, then set some index to True
# find all triangles that touch a given vertex
region=(np.sum(triangleData==20,axis=1)>0)
```

1.3 Visualization of region

[10]:
```
# now visualize this as filled triangles
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="k",s=10)

drawEdges(ax,pointData,edgeData,colors="k",drawOrientations=True)

polyCollection=matplotlib.collections.
  ↪LineCollection(pointData[triangleData[region]],lw=0,fc="#808080",zorder=-2)
ax.add_collection(polyCollection)

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.tight_layout()
plt.show()
```



1.4 Compute boundary of region

[12]:
```
def getB2(nEdges,etAdjacencyData,etAdjacencyDataOrientation):
    nTriangles=etAdjacencyData.shape[0]
    data=etAdjacencyDataOrientation.copy().ravel()
    indices=etAdjacencyData.copy().ravel()
    indptr=3*np.arange(0,nTriangles+1)
    result=scipy.sparse.
 ↪csc_matrix((data,indices,indptr),shape=(nEdges,nTriangles)).tocsr()
    return result
```

[13]:
```
# boundary operator: take indicator vector of triangles to indicator vector of␣
 ↪bounding edges
# take orientation into account: this way, contributions from adjacent␣
 ↪triangles cancel

# B2 stands for: boundary of 2-dimensional objects
```

```
B2=getB2(nEdges,etAdjacencyData,etAdjacencyDataOrientation)
```

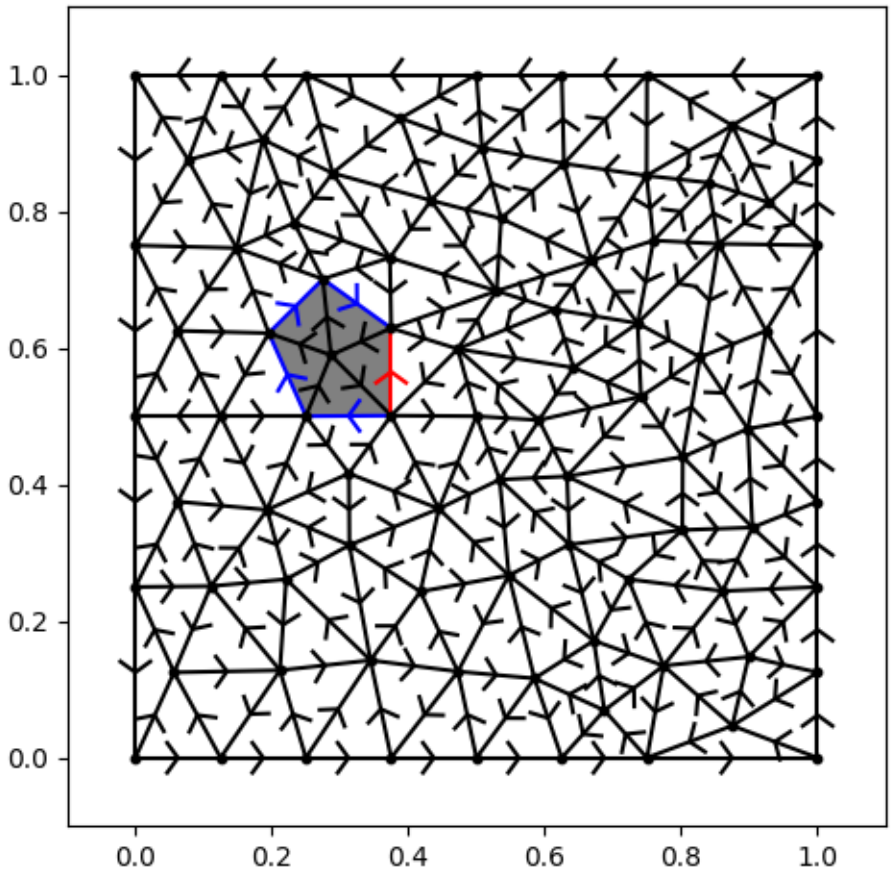
[14]:
```
B2.shape
```

[14]:
```
(237, 150)
```

[15]:
```
# now apply to region indicator
boundary=B2.dot(region)
```

[16]:
```
def signToColor(bdry):
    colors=np.zeros((bdry.shape[0],3))
    colors[bdry>0.1]=np.array([1.,0.,0.])
    colors[bdry<-0.1]=np.array([0.,0.,1.])
    return colors
```

[17]:
```
# highlight boundary in plot
fig=plt.figure()
ax=fig.add_subplot(aspect=1.)
ax.scatter(pointData[:,0],pointData[:,1],c="k",s=10)

lineColors=signToColor(boundary)
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
 ↪03,zorder=-1)

polyCollection=matplotlib.collections.
 ↪LineCollection(pointData[triangleData[region]],lw=0,fc="#808080",zorder=-2)
ax.add_collection(polyCollection)

buffer=0.1
plt.xlim([-buffer,1+buffer])
plt.ylim([-buffer,1+buffer])
plt.tight_layout()
plt.show()
```



1.5 Shortest paths

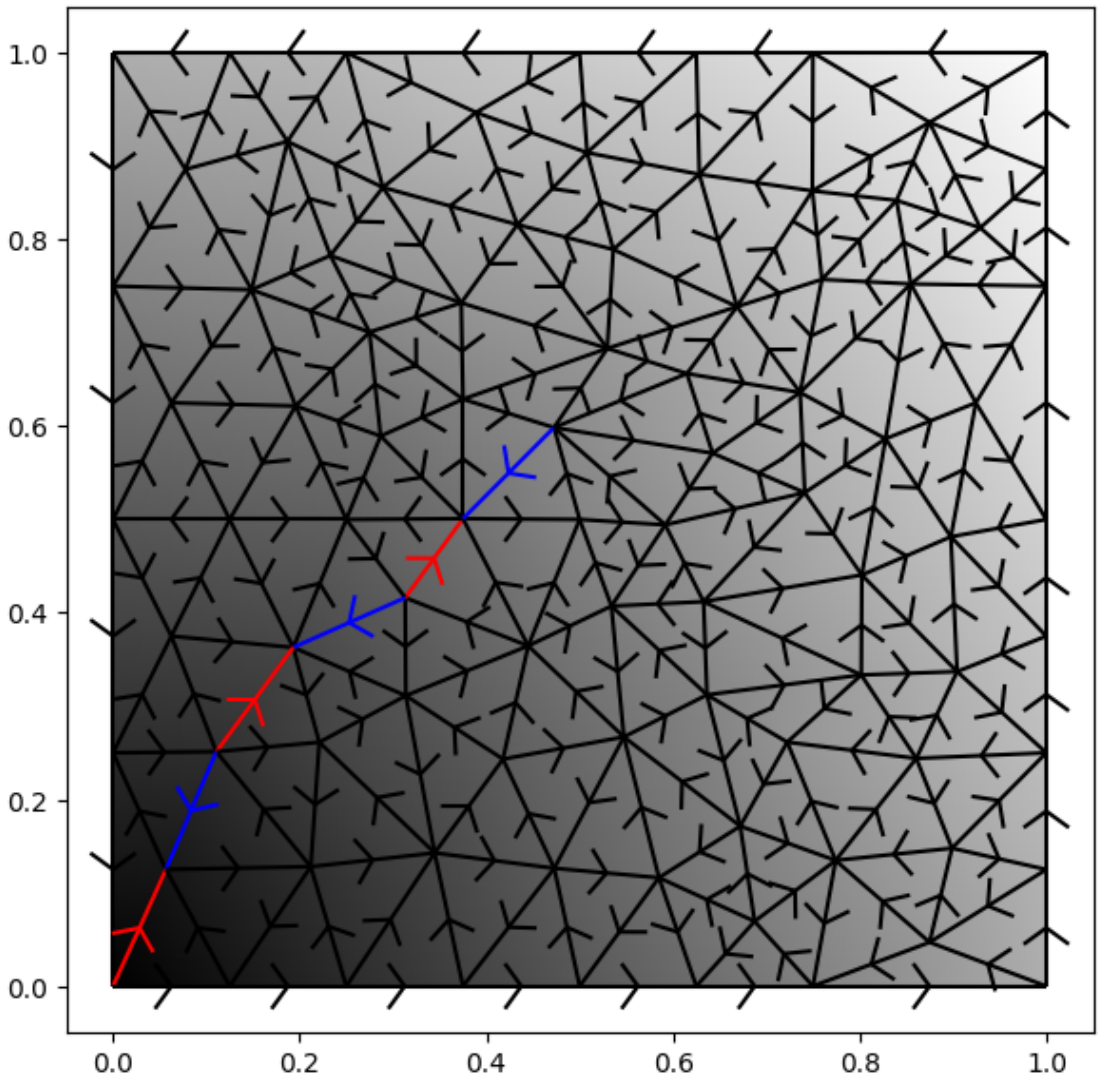
We can use Dijkstra's algorithm (or others) to find shortest paths on edges through the mesh

[18]:
```
def getShortestPaths(pointData,edgeData):
    # compute edge lengths
    edgeLengths=np.linalg.norm(pointData[edgeData[:,0]]-pointData[edgeData[:
↪,1]],axis=1)
    # build weighted adjacency matrix of graph, with weights given by lengths

    if False:
        # note: edges in the graph that we want to perform the search on, are␣
↪un-oriented, but the computational
        # representation has oriented edges. so we need to add both directions
        # the construction is most efficiently done in scipy.sparse.coo_matrix:
        graph=scipy.sparse.coo_matrix((\
                np.concatenate((edgeLengths,edgeLengths)),\
                (np.concatenate((edgeData[:,0],edgeData[:,1])),\
```

```
                np.concatenate((edgeData[:,1],edgeData[:,0])))\
                ))
        # non-existing entries correspond to infinite edge lengths (not zero!)
        # convert to csr, which is (probably) faster in graph algorithm
        graph=graph.tocsr()
    else:
        # use "directed=False" option of graph search instead
        graph=scipy.sparse.coo_matrix((edgeLengths,(edgeData[:,0],edgeData[:
↪,1])))


    dist,pred=scipy.sparse.csgraph.
↪shortest_path(graph,directed=False,return_predecessors=True)
    return dist,pred
```

[19]:
```
def extractPath(pred,i0,i1):
    path=[i1]
    j=i1
    while j!=i0:
        j=pred[i0,j]
        path.append(j)
    return np.array(path,dtype=np.int32)


def translatePath(path,edgeData):
    nEdges=edgeData.shape[0]
    # these matrices will help identifying edge indices from the vertex indices:
    auxmatInd=scipy.sparse.coo_matrix((\
            np.concatenate((np.arange(nEdges),np.arange(nEdges))),\
            (np.concatenate((edgeData[:,0],edgeData[:,1])),\
            np.concatenate((edgeData[:,1],edgeData[:,0])))\
            )).tocsr()
    auxmatOrientation=scipy.sparse.coo_matrix((\
            np.concatenate((np.full((nEdges,),fill_value=1.),np.
↪full((nEdges,),fill_value=-1.))),\
            (np.concatenate((edgeData[:,0],edgeData[:,1])),\
            np.concatenate((edgeData[:,1],edgeData[:,0])))\
            )).tocsr()

    result=np.zeros((nEdges,),dtype=np.double)
    for j in range(len(path)-1,0,-1):
        ␣
↪result[auxmatInd[path[j],path[j-1]]]=auxmatOrientation[path[j],path[j-1]]
    return result
```

[20]:
```
# plot length from a given vertex as color coded mesh
# which vertex?
i0=0
# find shortest path to a given target vertex
i1=30

dist,pred=getShortestPaths(pointData,edgeData)
pathList=extractPath(pred,i0,i1)
path=translatePath(pathList,edgeData)


fig=plt.figure(figsize=(6,6))
ax=fig.add_subplot(aspect=1.)

tri=matplotlib.tri.Triangulation(pointData[:,0],pointData[:,1],triangleData)
pltobj=ax.tripcolor(tri,dist[i0,:]/np.max(dist[i0,:]),shading='gouraud',cmap=cm.
 ↪gray)

lineColors=signToColor(path)
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
 ↪03,zorder=1)

#plt.colorbar(pltobj)

plt.tight_layout()
plt.show()
```



1.6 Boundaries of paths

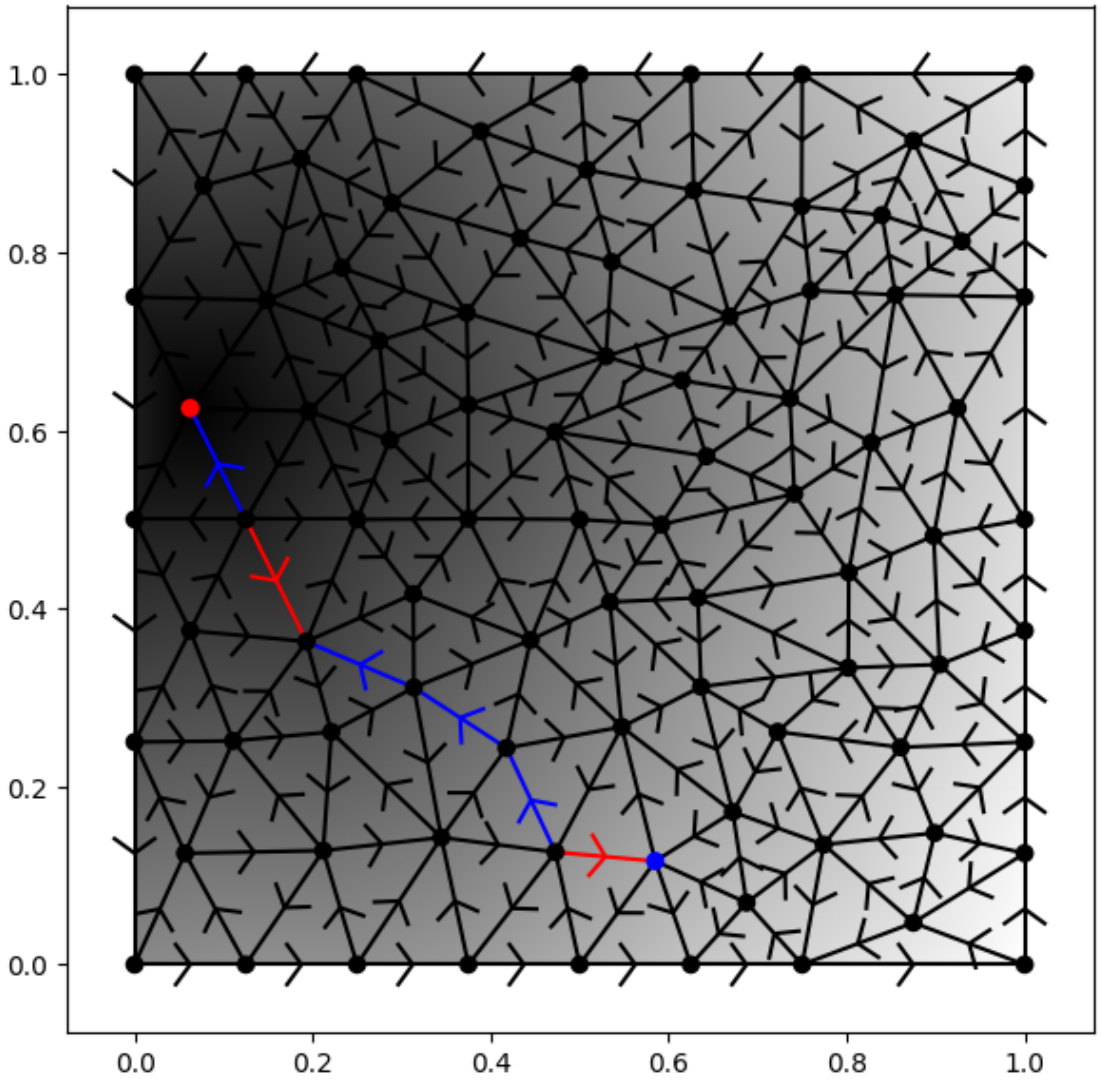
- We can interpret starting and end points of lines (paths) as their boundaries, weighted with opposite signs.
- A path from point A to point B will then have A as positive boundary and B as negative boundary.

[20]:
```
def getB1(nPoints,edgeData):
    nEdges=edgeData.shape[0]
    data=np.zeros((nEdges*2),dtype=np.double)
    data[0::2]=1.
    data[1::2]=-1.
```

```
    indices=edgeData.copy().ravel()
    indptr=2*np.arange(nEdges+1)
    result=scipy.sparse.
 ↪csc_matrix((data,indices,indptr),shape=(nPoints,nEdges)).tocsr()
    return result
```

[21]:
```
B1=getB1(nPoints,edgeData)
```

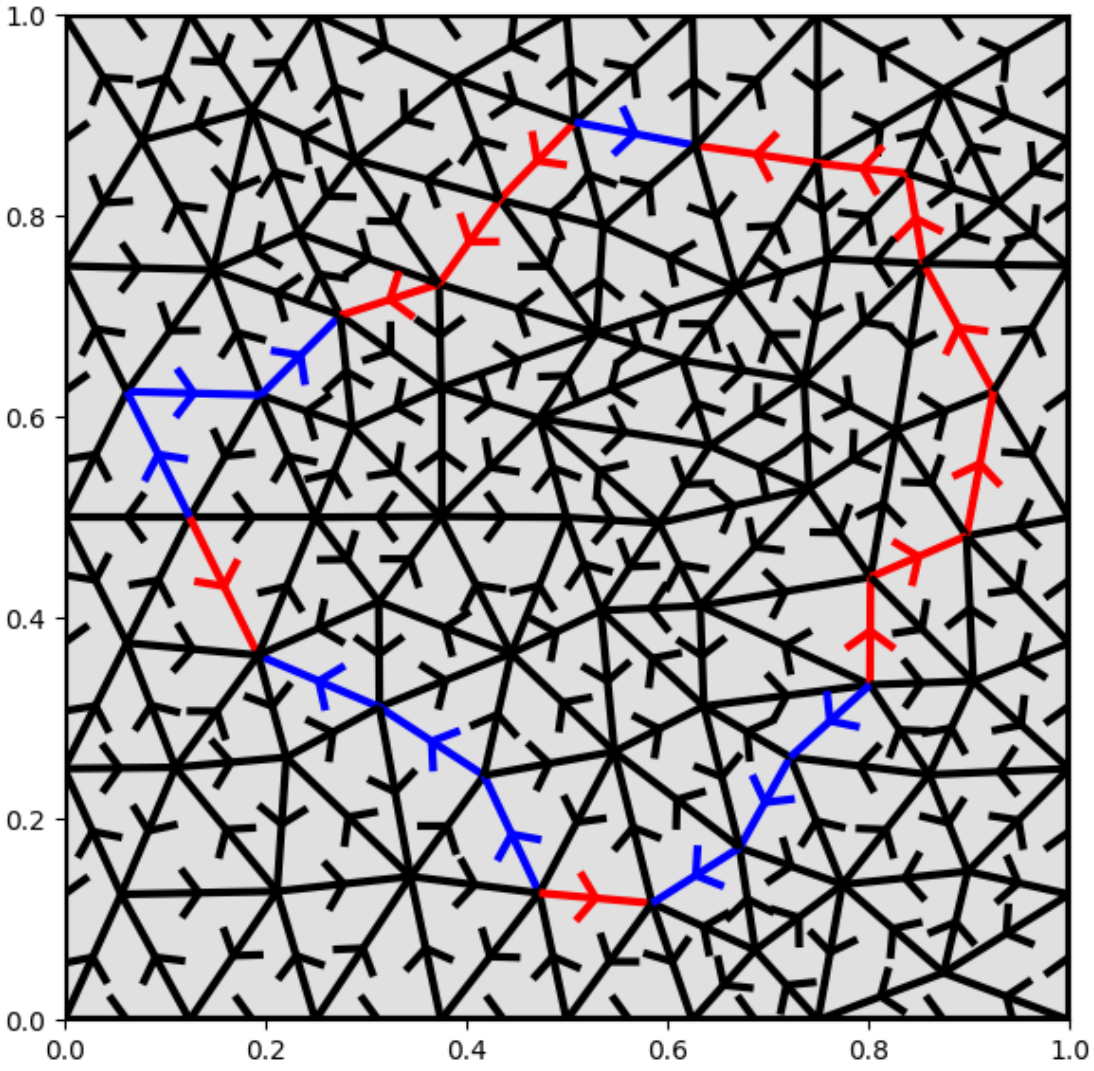
[22]:
```
# starting vertex
i0=10
# end vertex
i1=53

pathList=extractPath(pred,i0,i1)
path=translatePath(pathList,edgeData)


fig=plt.figure(figsize=(6,6))
ax=fig.add_subplot(aspect=1.)

tri=matplotlib.tri.Triangulation(pointData[:,0],pointData[:,1],triangleData)
ax.tripcolor(tri,dist[i0,:]/np.max(dist[i0,:]),shading='gouraud',cmap=cm.gray)

lineColors=signToColor(path)
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
 ↪03,zorder=1)

# add colored points to indicate boundary of shortest path
pathBoundary=B1.dot(path)
pointColors=signToColor(pathBoundary)
plt.scatter(pointData[:,0],pointData[:,1],c=pointColors,zorder=2)

#for i,x in enumerate(pointData):
#    ax.text(x[0],x[1],i,c="r")

plt.tight_layout()
plt.show()
```



1.7 Region from boundary

[23]:
```
# first build a cycle (which we conjecture to be a boundary)
# fix some vertices, connect shortest paths between them
iList=[10,53,31,56,10]
path=np.zeros(nEdges)
for j in range(len(iList)-1):
    pathList=extractPath(pred,iList[j],iList[j+1])
    path+=translatePath(pathList,edgeData)

fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.,facecolor="#e0e0e0")
```

```
lineColors=signToColor(path)
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
  ↪03,zorder=1,lw=3)

plt.tight_layout()
plt.show()
```



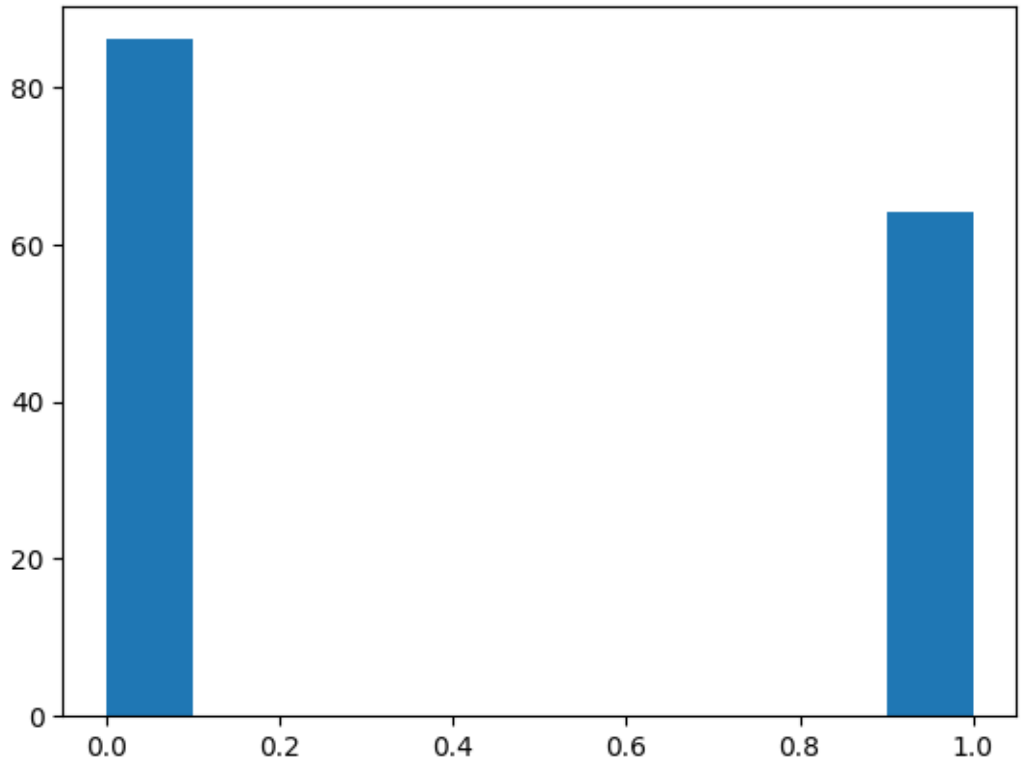
[24]:
```
import scipy.sparse.linalg
```

[25]:
```
# next: search for region indicator, such that it has the specified path as␣
  ↪boundary.
# solve: B2.dot(region)=path
```

```
# since we do not know for sure if solution exists, use leastsquares ansatz
res=scipy.sparse.linalg.lsqr(B2,path)
region=res[0]
# test solution quality:
print("norm of discrepancy between B2(region) and path:",res[3])
print("if solution was found, this should be 1: ",res[1])
```

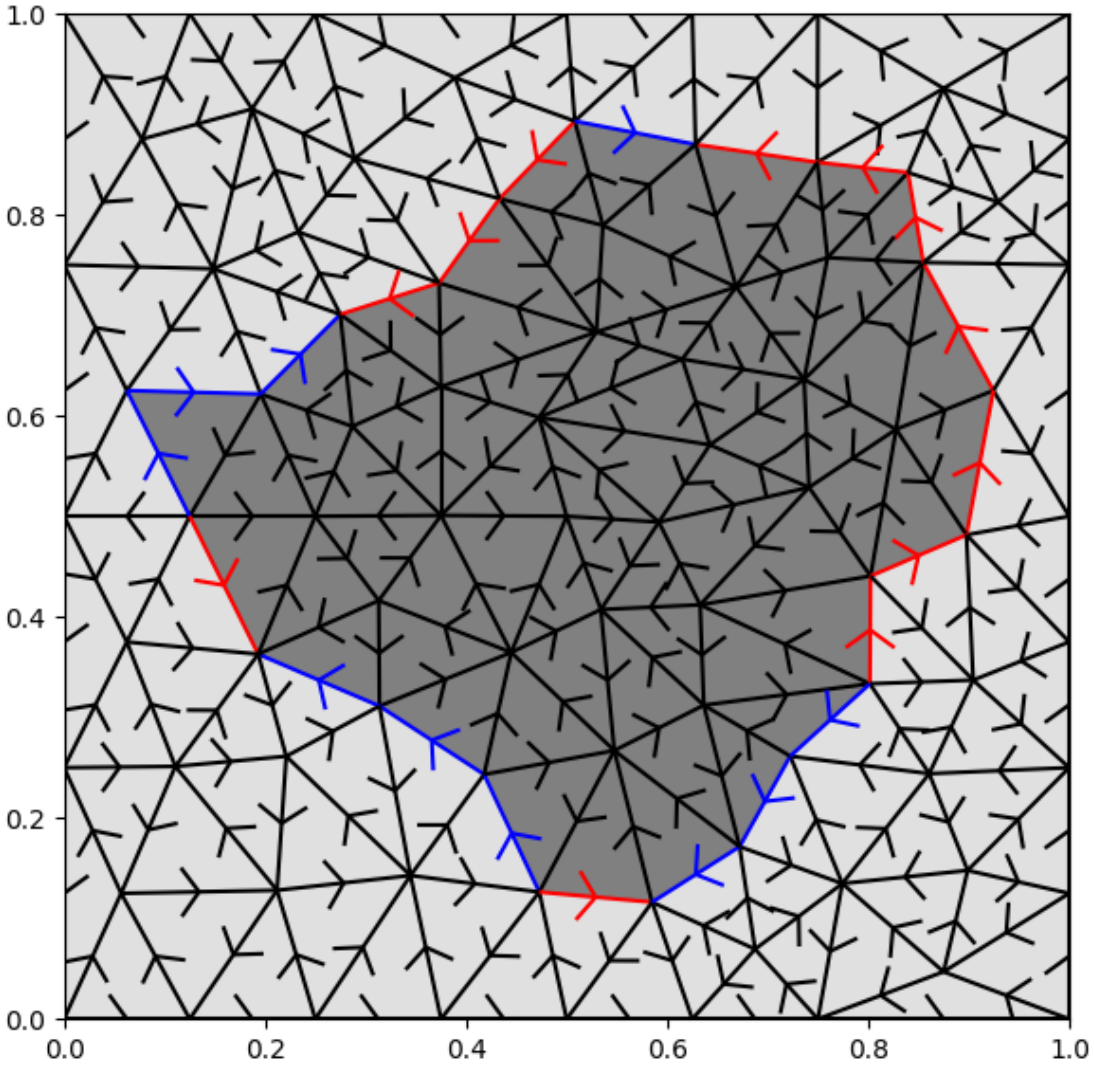
```
norm of discrepancy between B2(region) and path: 8.613445229696281e-05
if solution was found, this should be 1:  1
```

[26]:
```
# if a good solution was found, then region should be binary, with only 0 and 1␣
  ↪entries
plt.hist(region)
plt.show()
```



[27]:
```
# first build a cycle (which we conjecture to be a boundary)
# fix three vertices, connect shortest paths between them

fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.,facecolor="#e0e0e0")

lineColors=signToColor(path)
```

```
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
  ↪03,zorder=1)

regionIndicator=(region>0.1)
polyCollection=matplotlib.collections.
  ↪LineCollection(pointData[triangleData[regionIndicator]],\
        lw=0,fc="#808080",zorder=-2)
ax.add_collection(polyCollection)


plt.tight_layout()
plt.show()
```



1.8 Is every cycle path a boundary of a region?

[28]:
```
# first observation: each boundary path of a region has zero boundary itself
# (boundaries of regions are cycle paths; these have no beginning or end point)
# confirm by showing that B1.dot(B2)= zero matrix
B1.dot(B2)
```

[28]:
```
<88x150 sparse matrix of type '<class 'numpy.float64'>'
        with 0 stored elements in Compressed Sparse Row format>
```

[29]:
```
# so: image of B2 lives in kernel of B1
# compare dimensions:
#   dim(ker B1)=nEdges-rank(B1)
#   dim(img B2)=rank(B2)
```

[30]:
```
# since we use small matrices: determine rank on dense matrices
B1Dense=B1.todense()
B2Dense=B2.todense()

print(nEdges-np.linalg.matrix_rank(B1Dense))
print(np.linalg.matrix_rank(B2Dense))
```

```
150
150
```

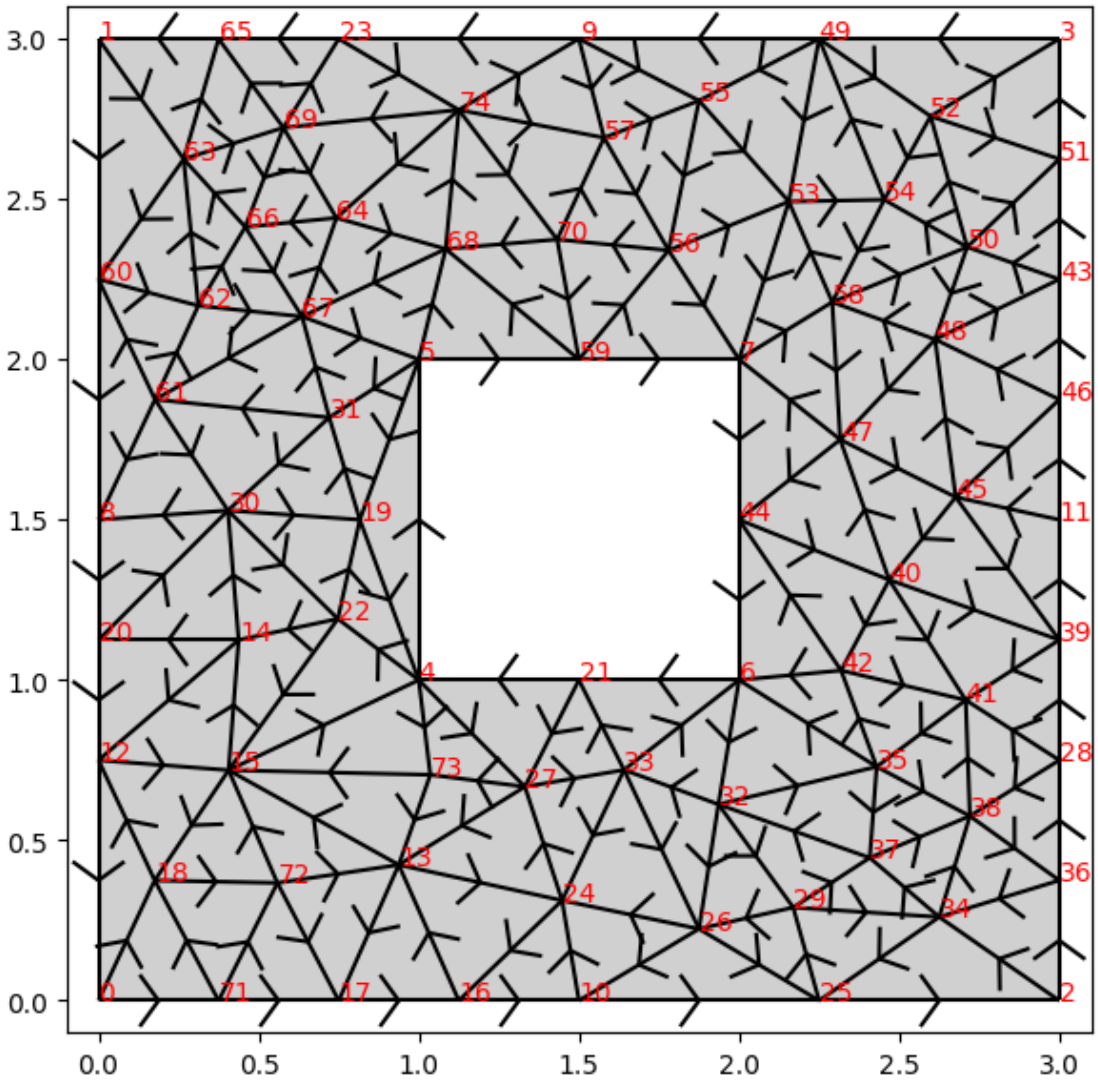
[31]:
```
# both numbers are equal: hence every cycle must be a boundary on our current mesh
```

1.9 A mesh with a hole

[33]:
```
# create/load a finer example mesh
if False:
    import lib.Triangle as Triangle

    points=np.array([[0,0],[0,3],[3,0],[3,3],[1,1],[1,2],[2,1],[2,2]],dtype=np.double)
    segments=np.array([[4,5],[5,7],[7,6],[6,4]],dtype=np.int32)
    segmentMarkers=np.zeros(segments.shape[0],dtype=np.int32)
    holes=np.array([[1.5,1.5]],dtype=np.double)
    maxArea=0.1
    pointData,triangleData=Triangle.createMeshAdvanced(points,segments,segmentMarkers,holes,\
              flagStr="zqpc",maxArea=maxArea)
    np.savez("triangulation_hole.npz",pointData=pointData,triangleData=triangleData)
else:
```

```
    data=np.load("triangulation_hole.npz")
    pointData=data["pointData"]
    triangleData=data["triangleData"]
    data.close()
nPoints=pointData.shape[0]
nTriangles=triangleData.shape[0]

# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[34]:
```
fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.)

polyCollection=matplotlib.collections.
 ↪LineCollection(pointData[triangleData],lw=0,fc="#d0d0d0",zorder=-2)
ax.add_collection(polyCollection)
drawEdges(ax,pointData,edgeData,colors="k",drawOrientations=True,arrowScale=0.1)

for i,x in enumerate(pointData):
    ax.text(x[0],x[1],i,c="r")

# plot range
buffer=0.1
lim1,lim2,lim3,lim4=np.min(pointData[:,0]),np.max(pointData[:,0]),np.
 ↪min(pointData[:,1]),np.max(pointData[:,1])
plt.xlim([lim1-buffer,lim2+buffer])
plt.ylim([lim3-buffer,lim4+buffer])


plt.tight_layout()
plt.show()
```



[35]:
```
B1=getB1(nPoints,edgeData)
B2=getB2(nEdges,etAdjacencyData,etAdjacencyDataOrientation)
```

[36]:
```
# do the rank test again:

# recall:
# -image of B2 lives in kernel of B1
# -compare dimensions:
#   dim(ker B1)=nEdges-rank(B1)
#   dim(img B2)=rank(B2)

# since we use small matrices: determine rank on dense matrices
B1Dense=B1.todense()
```

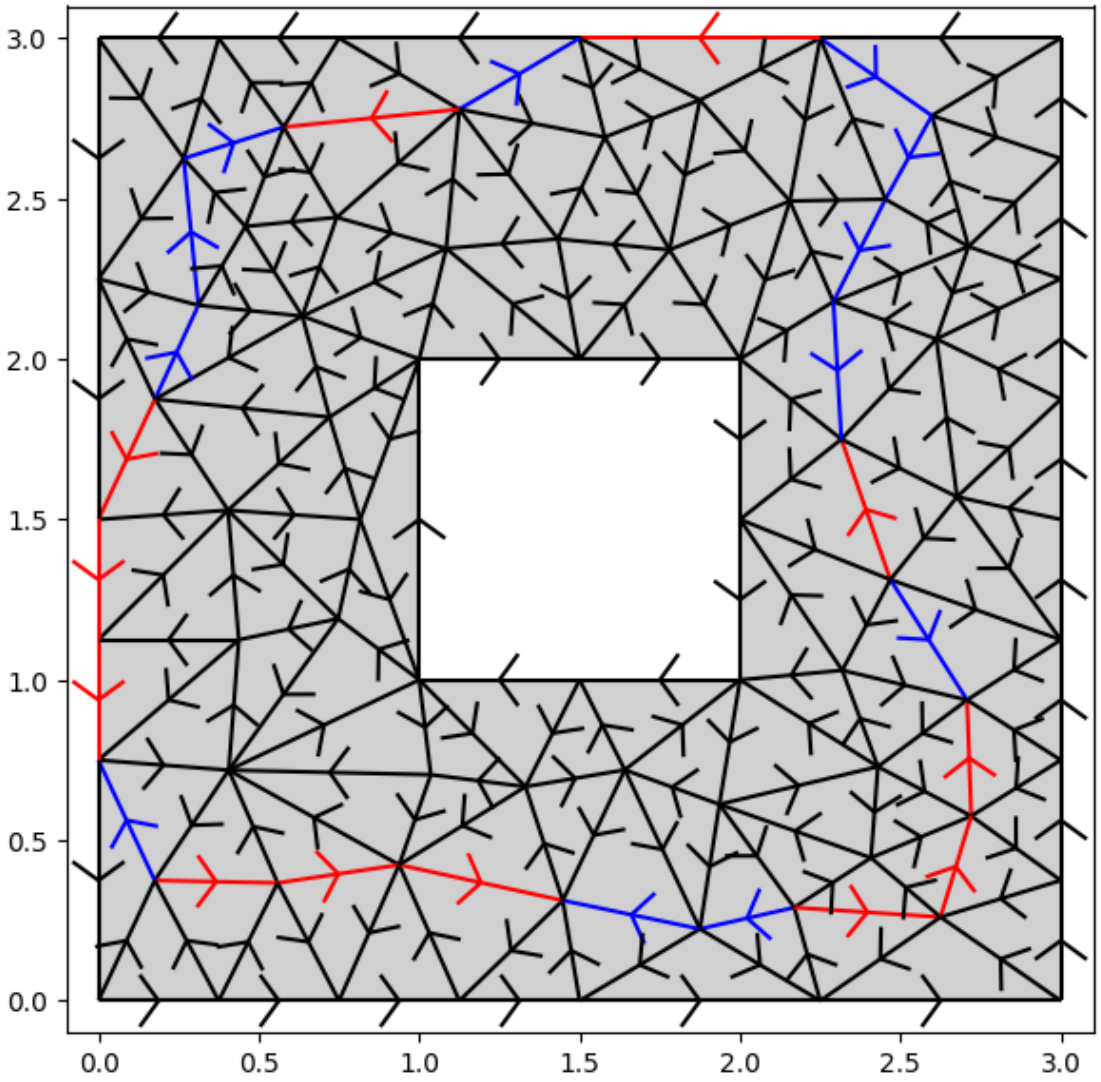
```
B2Dense=B2.todense()

print(nEdges-np.linalg.matrix_rank(B1Dense))
print(np.linalg.matrix_rank(B2Dense))
```

```
120
119
```

[37]:
```
# there is one equivalence class of cycles that is NOT a boundary!
# intuitively clear: these are the paths that go around the hole
```

[38]:
```
# construct such a path and examine it in more detail
dist,pred=getShortestPaths(pointData,edgeData)
```

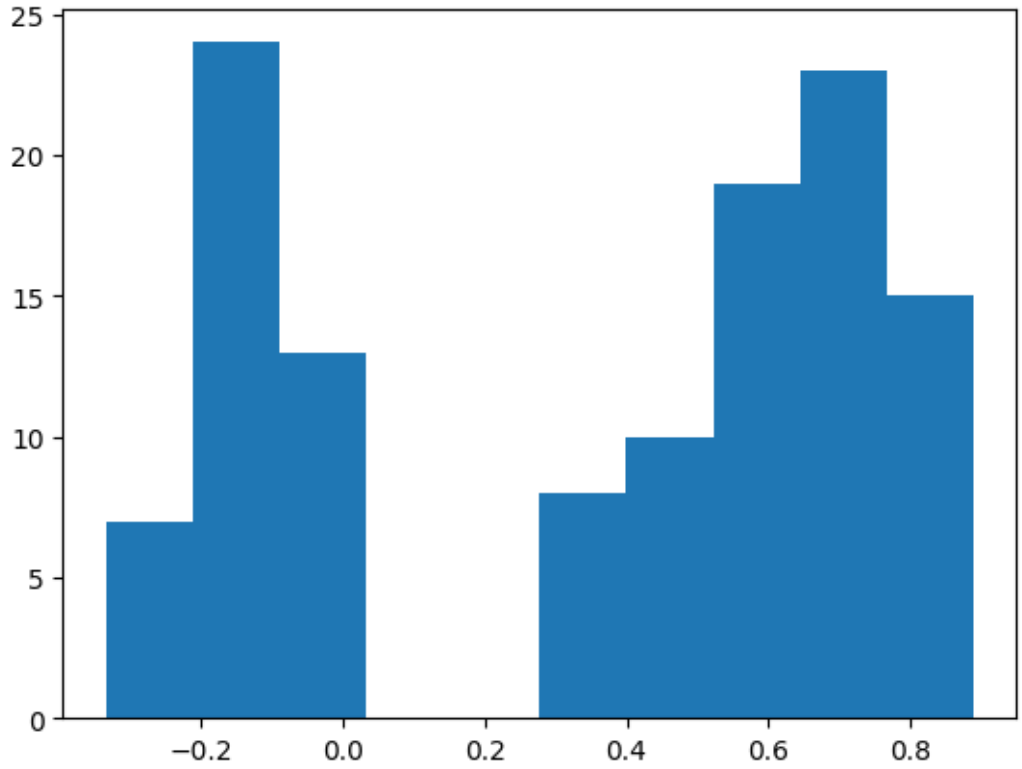
[39]:
```
# first build a cycle (which we conjecture to be a boundary)
# fix some vertices, connect shortest paths between them
iList=[18,34,52,63,18]
path=np.zeros(nEdges)
for j in range(len(iList)-1):
    pathList=extractPath(pred,iList[j],iList[j+1])
    path+=translatePath(pathList,edgeData)


fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.)

polyCollection=matplotlib.collections.
 ↪LineCollection(pointData[triangleData],lw=0,fc="#d0d0d0",zorder=-2)
ax.add_collection(polyCollection)
lineColors=signToColor(path)
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
 ↪1,zorder=1)

#for i,x in enumerate(pointData):
#    ax.text(x[0],x[1],i,c="r")

# plot range
buffer=0.1
lim1,lim2,lim3,lim4=np.min(pointData[:,0]),np.max(pointData[:,0]),np.
 ↪min(pointData[:,1]),np.max(pointData[:,1])
plt.xlim([lim1-buffer,lim2+buffer])
plt.ylim([lim3-buffer,lim4+buffer])


plt.tight_layout()
plt.show()
```



[40]:
```
# as above: try to find region which has given path as boundary
# by trying to solve solve: B2.dot(region)=path
# since we do not know for sure if solution exists, use leastsquares ansatz
res=scipy.sparse.linalg.lsqr(B2,path)
region=res[0]
# test solution quality: this time we have a non-zero residual!
print("norm of discrepancy between B2(region) and path:",res[3])
print("if solution was found, this should be 1: ",res[1])
```

```
norm of discrepancy between B2(region) and path: 1.4968407965017811
if solution was found, this should be 1:  2
```

[41]:
```
# if a good solution was found, then region should be binary, with only 0 and 1␣
  ↪entries
plt.hist(region)
plt.show()
```



[42]:
```
# now add the boundary of the hole as additional column to B2:

# first: extract bounday of hole
iList=[4,5,7,6,4]
holeBoundary=np.zeros(nEdges)
for j in range(len(iList)-1):
    pathList=extractPath(pred,iList[j],iList[j+1])
    holeBoundary+=translatePath(pathList,edgeData)

# now add to B2:
B2Ext=scipy.sparse.hstack((B2,scipy.sparse.csr_matrix(holeBoundary.
  ↪reshape((-1,1)))))
```

[43]:
```
# do the rank test again:

# recall:
# -image of B2 lives in kernel of B1
```

```
# -compare dimensions:
#   dim(ker B1)=nEdges-rank(B1)
#   dim(img B2)=rank(B2)

# since we use small matrices: determine rank on dense matrices
B1Dense=B1.todense()
B2Dense=B2Ext.todense()

print(nEdges-np.linalg.matrix_rank(B1Dense))
print(np.linalg.matrix_rank(B2Dense))
```

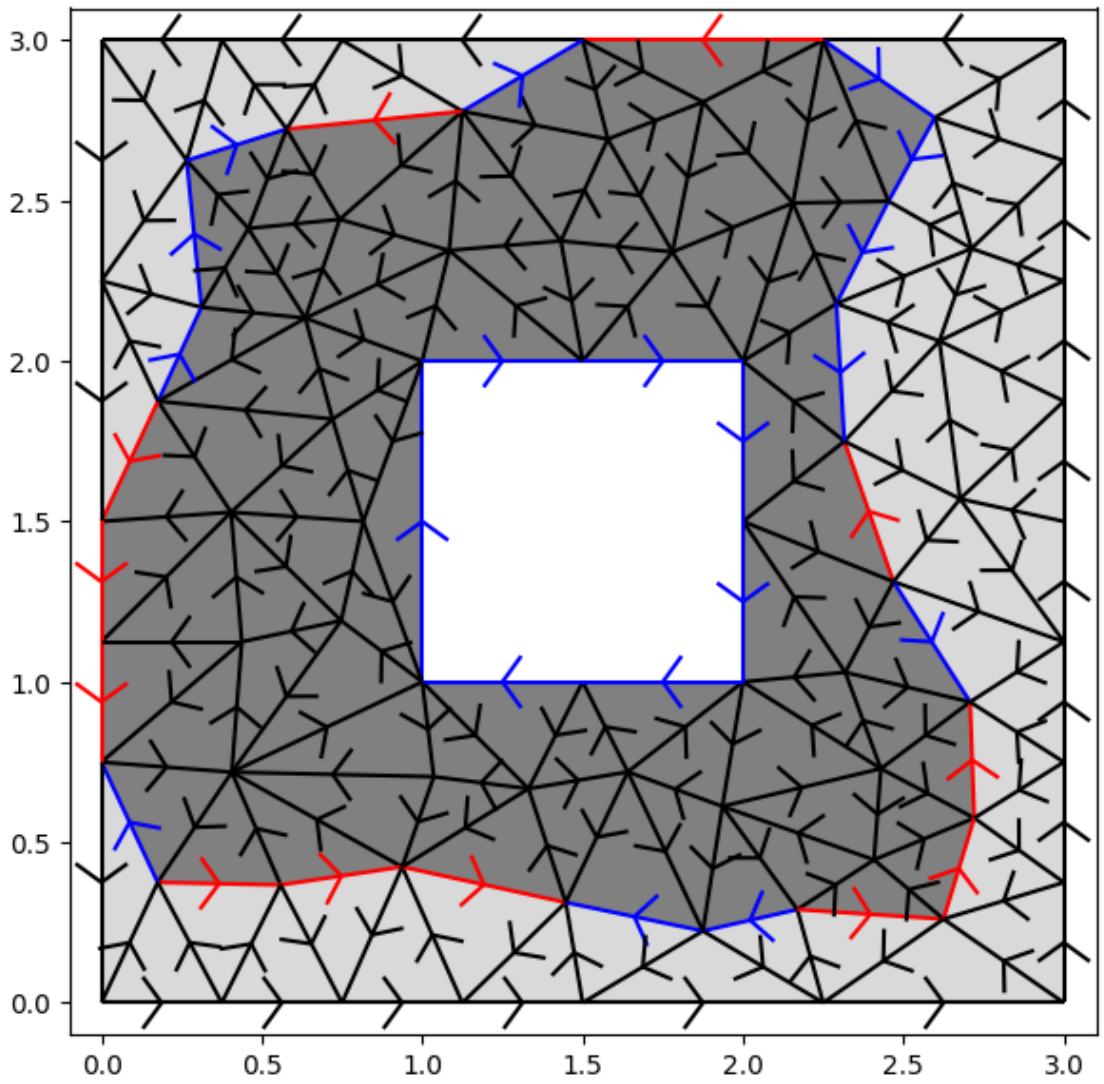
```
120
120
```

[44]:
```
# do least squares solving again:
res=scipy.sparse.linalg.lsqr(B2Ext,path)
region=res[0]
# test solution quality:
print("norm of discrepancy between B2(region) and path:",res[3])
print("if solution was found, this should be 1: ",res[1])
```

```
norm of discrepancy between B2(region) and path: 6.776446353336523e-05
if solution was found, this should be 1:  1
```

[45]:
```
iList=[18,34,52,63,18]
path=np.zeros(nEdges)
for j in range(len(iList)-1):
    pathList=extractPath(pred,iList[j],iList[j+1])
    path+=translatePath(pathList,edgeData)

pathExt=path.copy()
pathExt+=holeBoundary*region[-1]

fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.)


regionColors=np.zeros((nTriangles,3))
regionColors[...]=np.array([0.85,0.85,0.85])

#
regionColors[(region[:-1]>0.1),:]=np.array([0.5,0.5,0.5])
polyCollection=matplotlib.collections.
 ↪LineCollection(pointData[triangleData],lw=0,fc=regionColors,zorder=-2)
ax.add_collection(polyCollection)
lineColors=signToColor(pathExt)
```

```
drawEdges(ax,pointData,edgeData,colors=lineColors,drawOrientations=True,arrowScale=0.
  ↪1,zorder=1)

#for i,x in enumerate(pointData):
#    ax.text(x[0],x[1],i,c="r")

# plot range
buffer=0.1
lim1,lim2,lim3,lim4=np.min(pointData[:,0]),np.max(pointData[:,0]),np.
  ↪min(pointData[:,1]),np.max(pointData[:,1])
plt.xlim([lim1-buffer,lim2+buffer])
plt.ylim([lim3-buffer,lim4+buffer])


plt.tight_layout()
plt.show()
```



1.10 Now repeat this on the 2-torus

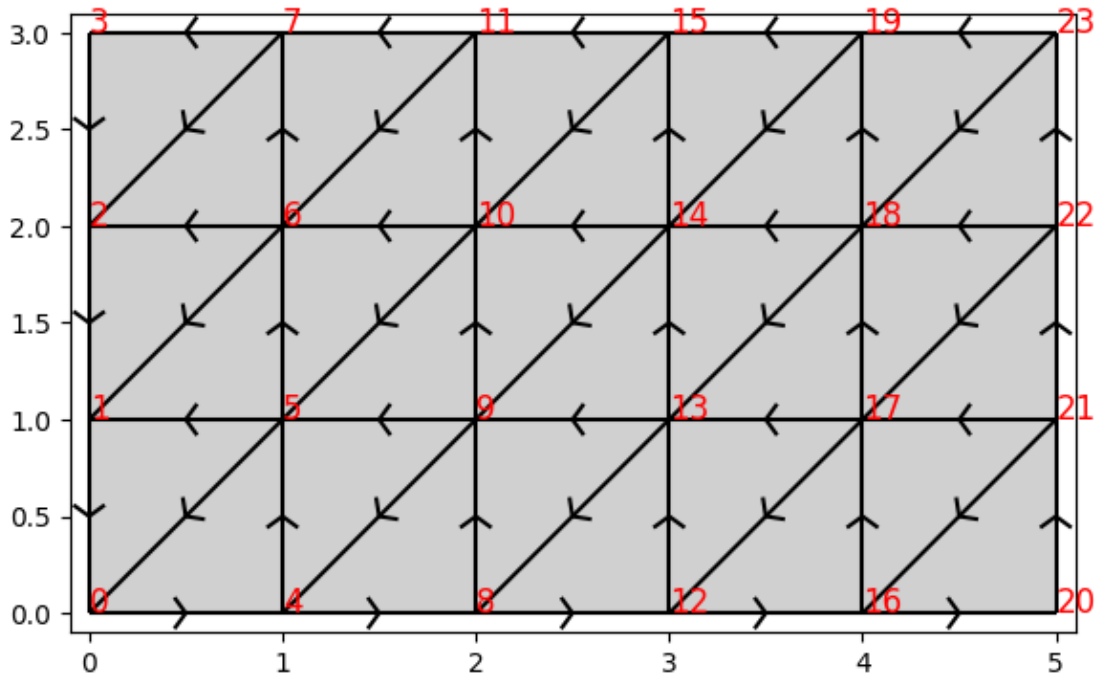
[46]:
```
def getPoslistNCube(shape,dtype=np.double):
    """Create list of positions in an n-dimensional cuboid of size shape."""
    ndim=len(shape)

    axGrids=[np.arange(i,dtype=dtype) for i in shape]
    prePos=np.array(np.meshgrid(*axGrids,indexing='ij'),dtype=dtype )
    # the first dimension of prepos is the dimension of the  posvector, the␣
 ↪successive dimensions are in the cube
    # so need to move first axis to end, and then flatten
    pos=np.rollaxis(prePos,0,ndim+1)
```

```
    # flattening
    newshape=(-1,ndim)
    return (pos.reshape(newshape)).copy()
```

[47]:
```
# first build regular triangulation of rectangle
nRows=4
nCols=6

pointData=getPoslistNCube((nCols,nRows))

nTriangles=(nRows-1)*(nCols-1)*2
triangleData=np.zeros((nTriangles,3),dtype=np.int32)
k=0
for i in range(nCols-1):
    for j in range(nRows-1):
        ind0=i*nRows+j
        ind1=(i+1)*nRows+j
        ind2=(i+1)*nRows+(j+1)
        ind3=i*nRows+(j+1)
        triangleData[k]=np.array([ind0,ind1,ind2])
        k+=1
        triangleData[k]=np.array([ind0,ind2,ind3])
        k+=1

nPoints=pointData.shape[0]

# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[48]:
```
%matplotlib inline
fig=plt.figure(figsize=(6,6),facecolor="w")
ax=fig.add_subplot(aspect=1.)

polyCollection=matplotlib.collections.
 ↪LineCollection(pointData[triangleData],lw=0,fc="#d0d0d0",zorder=-2)
ax.add_collection(polyCollection)
drawEdges(ax,pointData,edgeData,colors="k",drawOrientations=True,arrowScale=0.1)

for i,x in enumerate(pointData):
    ax.text(x[0],x[1],i,c="r",size=12)

# plot range
buffer=0.1
lim1,lim2,lim3,lim4=np.min(pointData[:,0]),np.max(pointData[:,0]),np.
 ↪min(pointData[:,1]),np.max(pointData[:,1])
plt.xlim([lim1-buffer,lim2+buffer])
```

```
plt.ylim([lim3-buffer,lim4+buffer])


plt.tight_layout()
plt.show()
```



[49]:
```
# Euler characteristic is that of planar mesh (not counting the "rest of␣
↪space"-face)
nTriangles-nEdges+nPoints
```

[49]: 1

[50]:
```
# now build mesh for torus
nRows=10
nCols=20

nTriangles=nRows*nCols*2
triangleData=np.zeros((nTriangles,3),dtype=np.int32)
k=0
for i in range(nCols):
    for j in range(nRows):
        ind0=i*nRows+j
        ind1=((i+1)%nCols)*nRows+j
        ind2=((i+1)%nCols)*nRows+((j+1)%nRows)
```

```
        ind3=i*nRows+((j+1)%nRows)
        triangleData[k]=np.array([ind0,ind1,ind2])
        k+=1
        triangleData[k]=np.array([ind0,ind2,ind3])
        k+=1

nPoints=nRows*nCols

# extract edge data
edgeData,etAdjacencyData,etAdjacencyDataOrientation=FEM.getEdges(triangleData)
nEdges=edgeData.shape[0]
```

[51]:
```
# Euler characteristic:
nTriangles-nEdges+nPoints
```

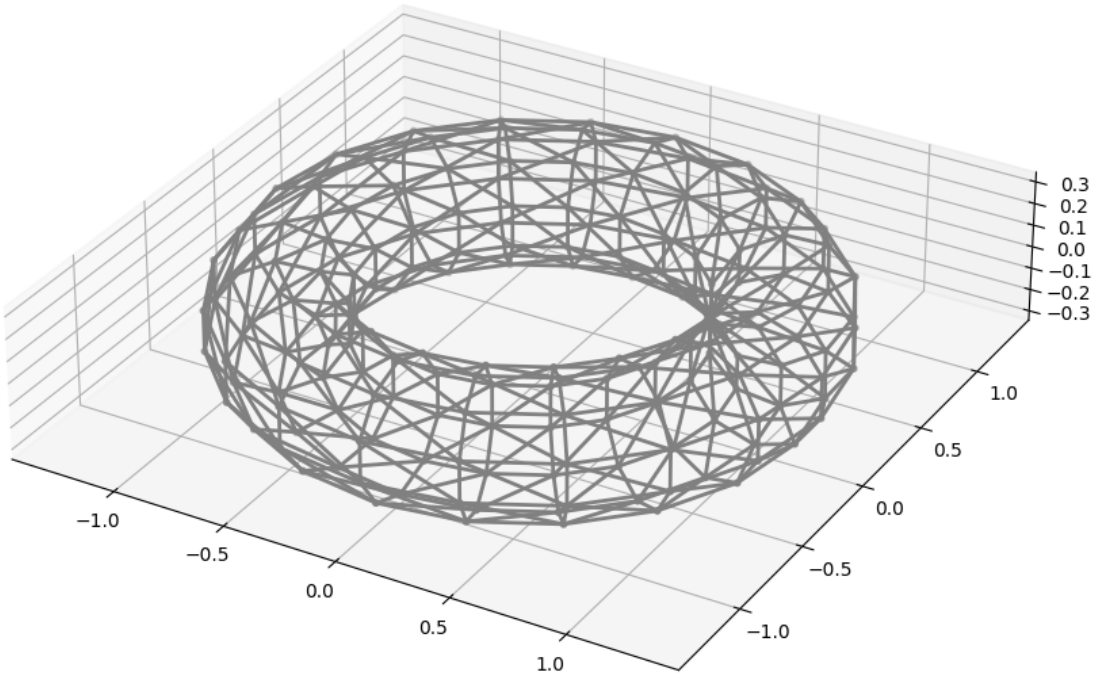
[51]: 0

[52]:
```
# at this point the mesh is only abstract, we have no coordinates
# now build the well-known "donut embedding"
pointData=np.zeros((nPoints,3),dtype=np.double)

# big and small radii:
R=1.
r=0.3
k=0
for i in range(nCols):
    for j in range(nRows):
        # "large angle"
        phi=i/nCols*2*np.pi
        theta=j/nRows*2*np.pi

        n1=np.array([np.cos(phi),np.sin(phi),0.])
        n2=np.array([0.,0.,1.])
        pointData[k,:]=R*n1+r*np.cos(theta)*n2+r*np.sin(theta)*n1
        k+=1
```

[53]:
```
import mpl_toolkits
```

[56]:
```
%matplotlib inline
fig = plt.figure(figsize=(8,8))

ax = fig.add_subplot(projection='3d')

ax.scatter(pointData[:,0],pointData[:,1],pointData[:,2],s=10,c="#808080")
lineCollection=mpl_toolkits.mplot3d.art3d.
 ↪Line3DCollection(pointData[edgeData],color="#808080",lw=2)
ax.add_collection(lineCollection)
```

```
ax.set_box_aspect((R+r,R+r,r))

plt.tight_layout()
plt.show()
```



[57]:
```
B1=getB1(nPoints,edgeData)
B2=getB2(nEdges,etAdjacencyData,etAdjacencyDataOrientation)
```

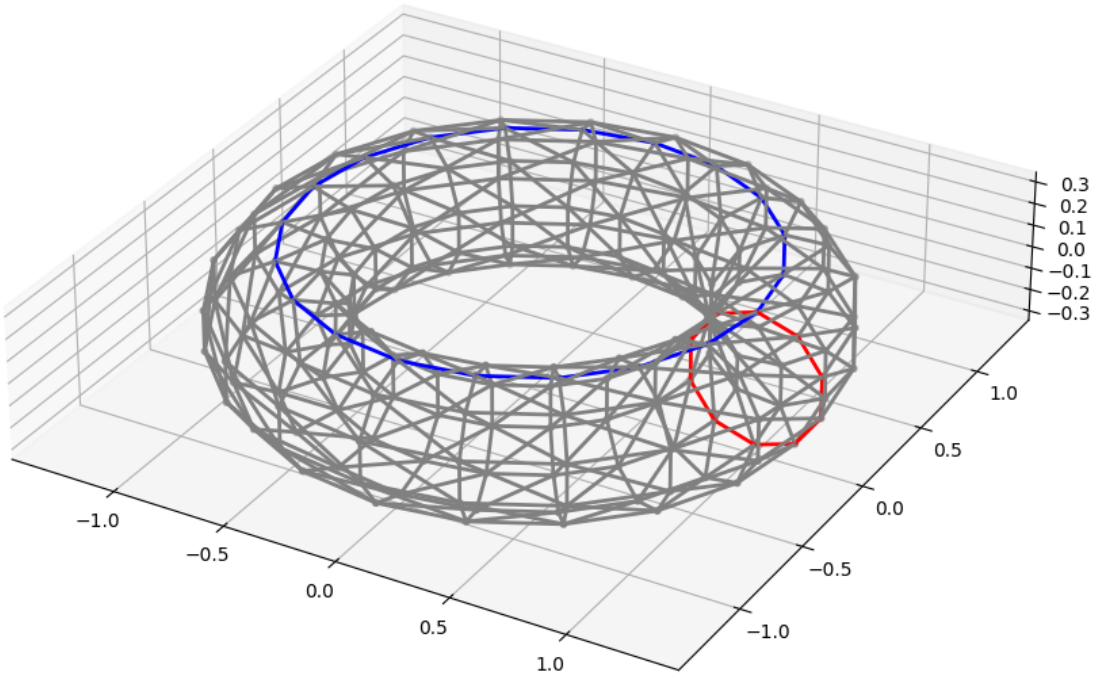
[58]:
```
# do the rank test again:

# recall:
# -image of B2 lives in kernel of B1
```

```
# -compare dimensions:
#   dim(ker B1)=nEdges-rank(B1)
#   dim(img B2)=rank(B2)

# since we use small matrices: determine rank on dense matrices
B1Dense=B1.todense()
B2Dense=B2.todense()

print(nEdges-np.linalg.matrix_rank(B1Dense))
print(np.linalg.matrix_rank(B2Dense))
```

```
401
399
```

[59]:
```
path1Pre=np.arange(nRows+1)
path1Pre[-1]=0
path1=translatePath(path1Pre,edgeData)

path2Pre=np.arange(nCols+1)*nRows
path2Pre[-1]=0
path2=translatePath(path2Pre,edgeData)
```

[62]:
```
%matplotlib inline
fig = plt.figure(figsize=(8,8))

ax = fig.add_subplot(projection='3d')

ax.scatter(pointData[:,0],pointData[:,1],pointData[:,2],s=10,c="#808080")

lineColors=np.zeros((nEdges,3))
lineColors[...]=np.array([0.5,0.5,0.5])
lineColors[(np.abs(path1[:])>0.1),:]=np.array([1.,0.,0.])
lineColors[(np.abs(path2[:])>0.1),:]=np.array([0.,0.,1.])

lineCollection=mpl_toolkits.mplot3d.art3d.
 ↪Line3DCollection(pointData[edgeData],color=lineColors,lw=2)
ax.add_collection(lineCollection)


ax.set_box_aspect((R+r,R+r,r))

plt.tight_layout()
plt.show()
```



[63]:
```
# add these as extra columns to B2:
B2Ext=scipy.sparse.hstack((B2,\
        scipy.sparse.csr_matrix(path1.reshape((-1,1))),\
        scipy.sparse.csr_matrix(path2.reshape((-1,1)))\
        ))
```

[64]:
```
# do the rank test again:

# recall:
# -image of B2 lives in kernel of B1
# -compare dimensions:
#   dim(ker B1)=nEdges-rank(B1)
#   dim(img B2)=rank(B2)
```

```
# since we use small matrices: determine rank on dense matrices
B1Dense=B1.todense()
B2Dense=B2Ext.todense()

print(nEdges-np.linalg.matrix_rank(B1Dense))
print(np.linalg.matrix_rank(B2Dense))
```

```
401
401
```

[65]:
```
B1.dot(B2Ext)
```

[65]:
```
<200x402 sparse matrix of type '<class 'numpy.float64'>'
        with 0 stored elements in Compressed Sparse Row format>
```

[ ]:
```

```