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

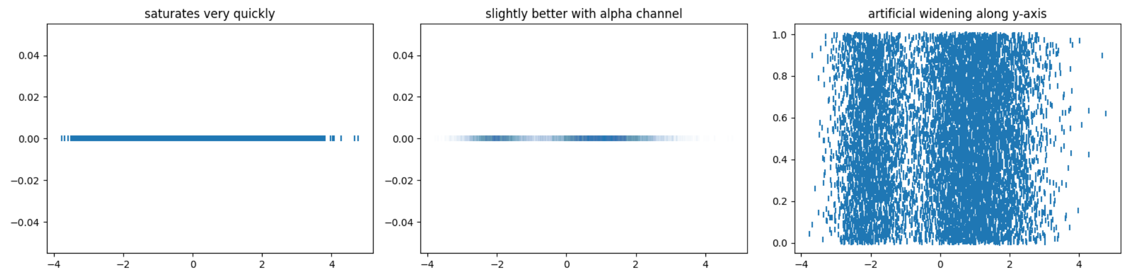
1.1 1d

[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?

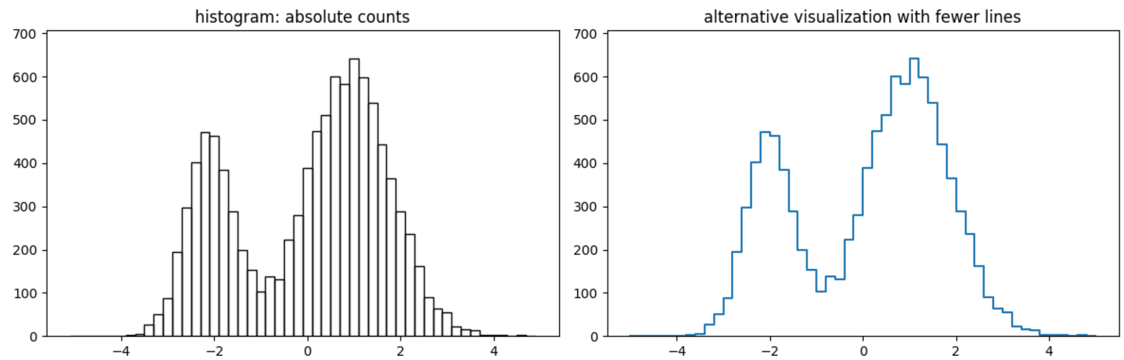
[6]:
```
fig=plt.figure(figsize=(16,4))

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
```

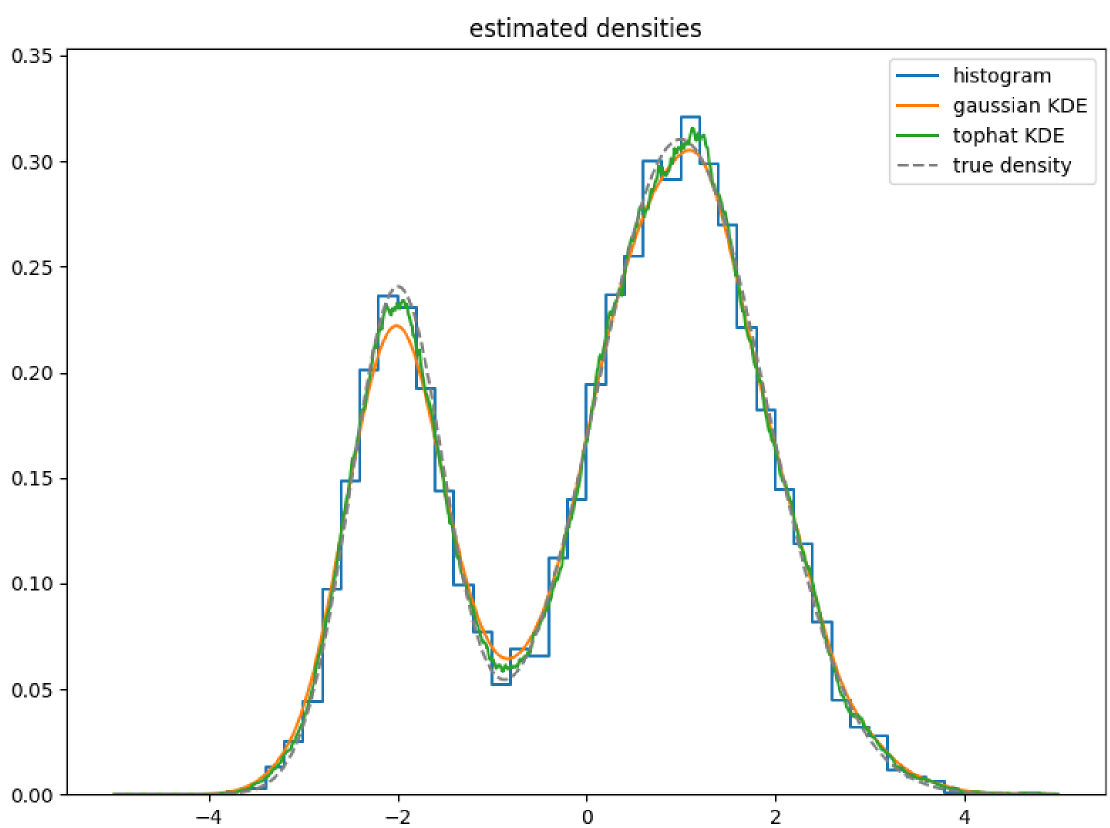
```
# standard histogram
hist,edges=np.histogram(x,range=rng,bins=nBins)

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

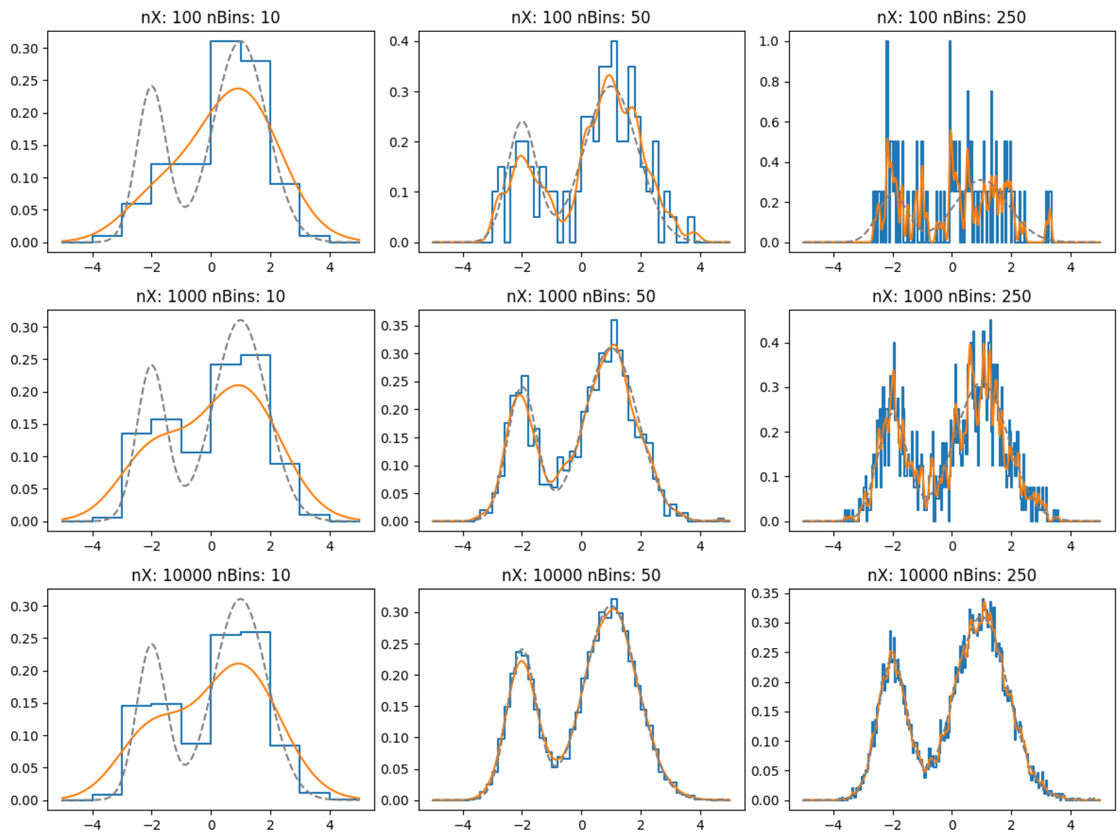
[9]:
```
# normalization of absolute counts as density function
#hist_dens=hist/np.sum(hist)/kWidth
# alternative: use proper keyword
hist_dens,edges=np.histogram(x,range=rng,bins=nBins,density=True)


# 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)))
```

```python
# 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()
```

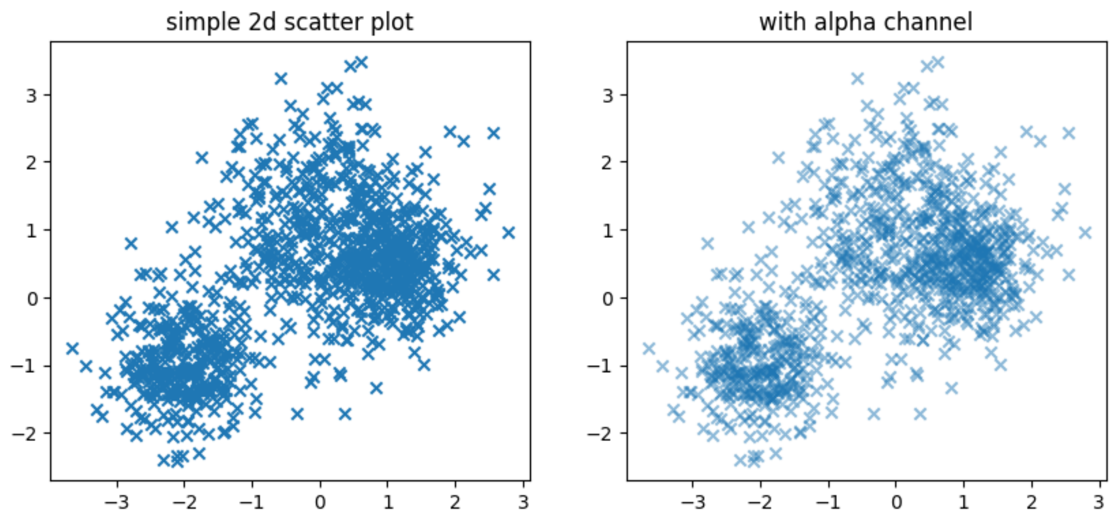


1.2 2d

[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()
```

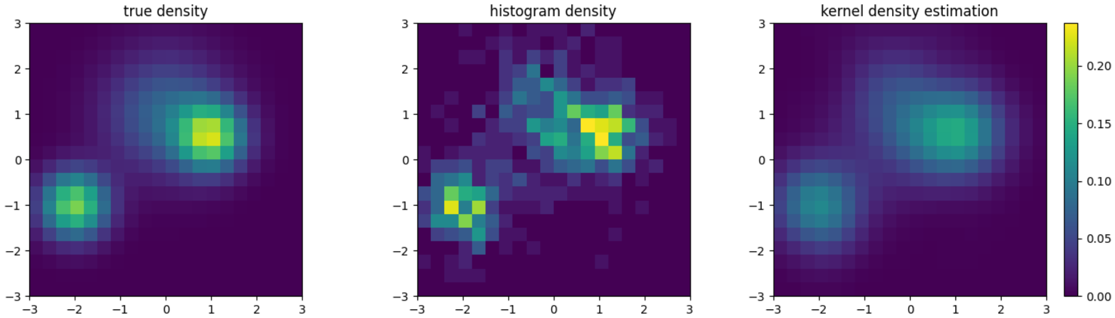


[13]:
```
# 2d Gaussian mixture model
# play with nX
nX=1000
dim=2
rng=[-3,3]
meanList=np.array([[-2,-1],[0,1],[1,0.5]])
stdList=np.array([0.5,0.9,0.5])
weightList=np.array([0.3,0.4,0.3])

# 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)
```

```python
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]:
```python
# 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]:
```python
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()
```

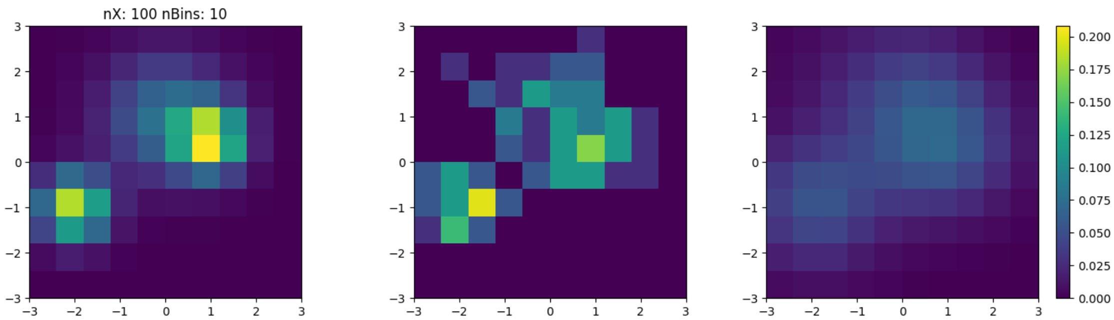


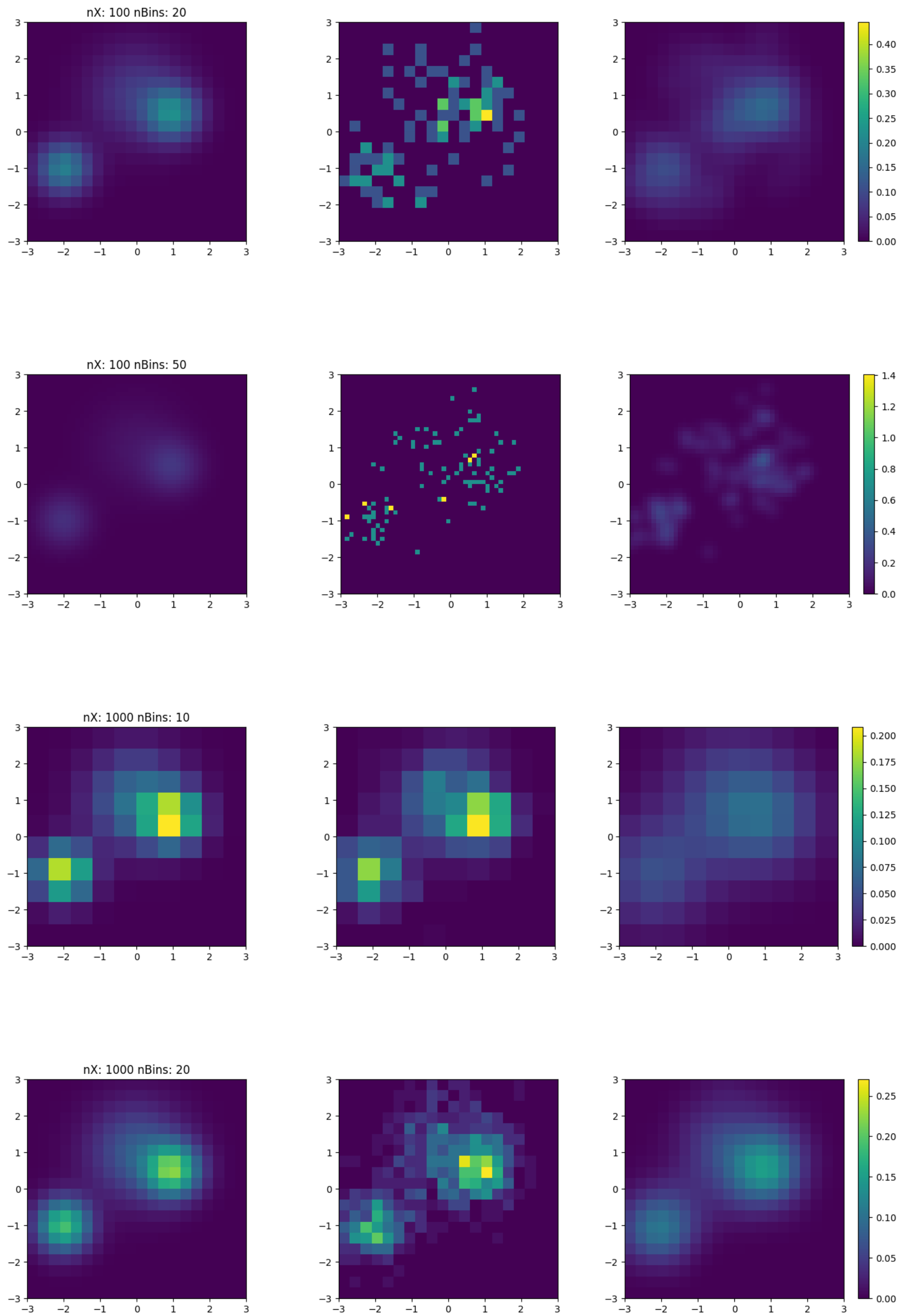
- optional: combine multiple histograms via separate color channels

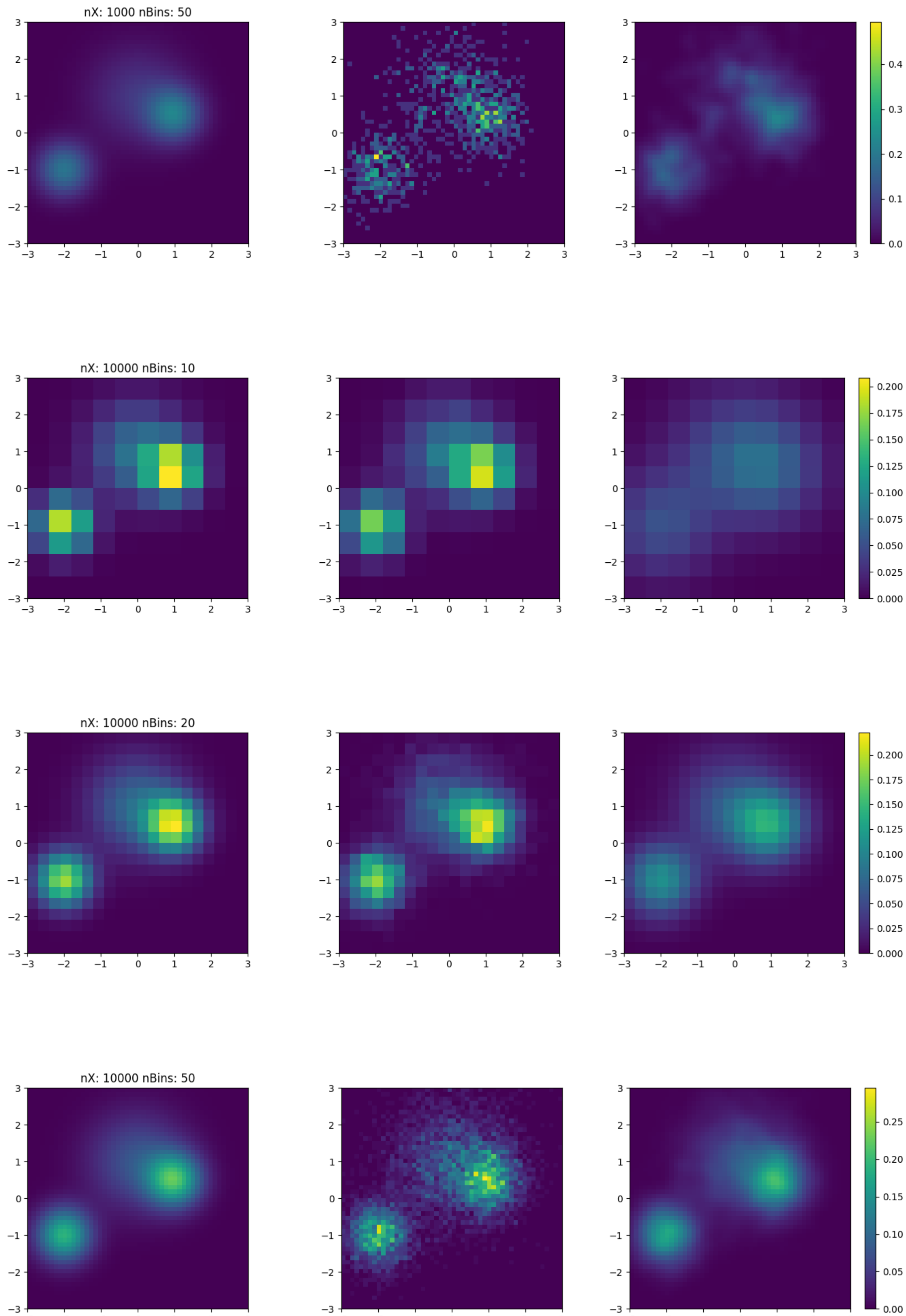
Number of samples, number of bins, kernel width

[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

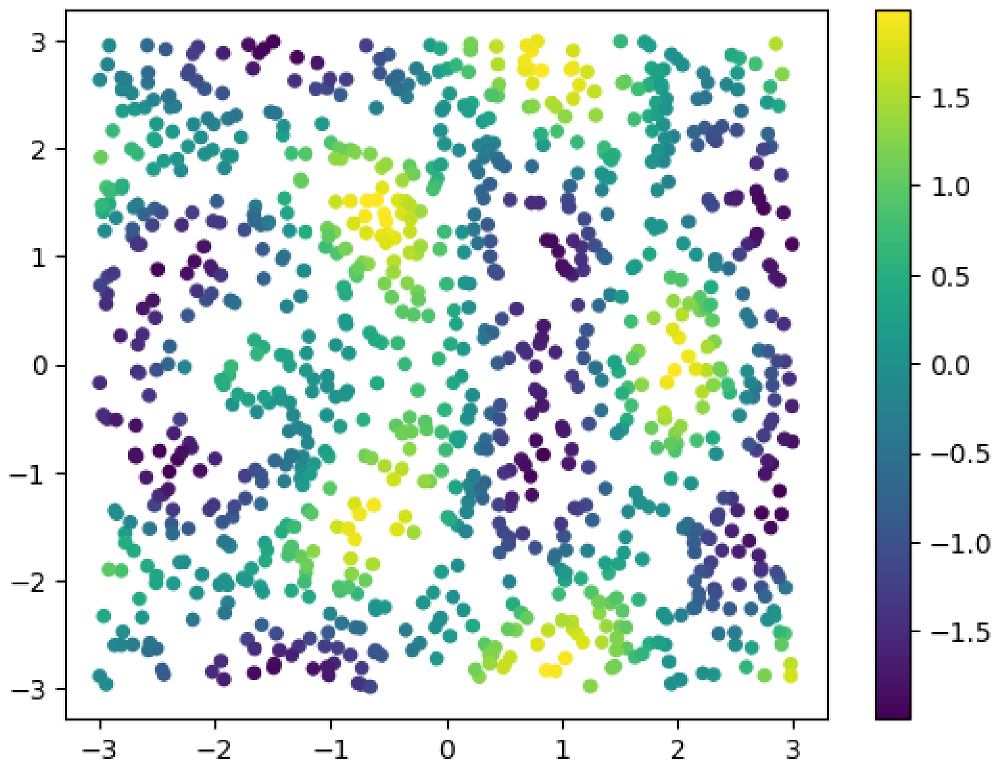

nX: 1000 nBins: 50
nX: 10000 nBins: 10
nX: 10000 nBins: 20
nX: 10000 nBins: 50

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()
```

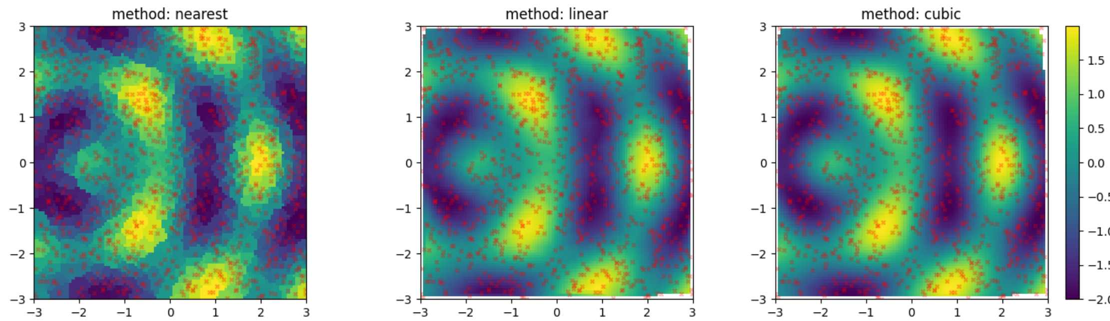


For better visualization: interpolate to a regular grid

- 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()
```

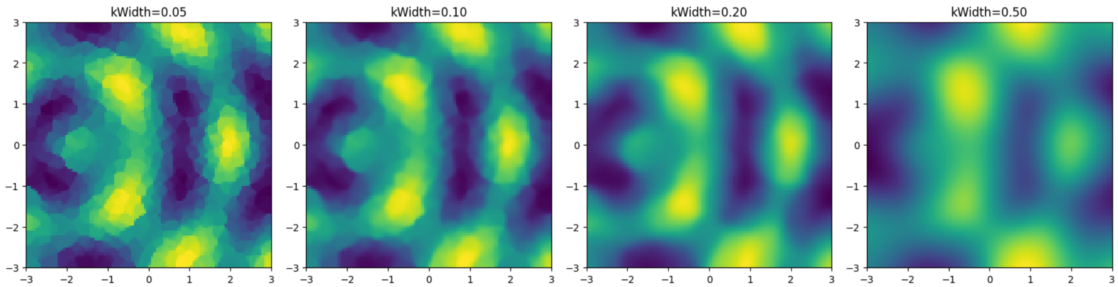


Simple kernel interpolation

[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))
```

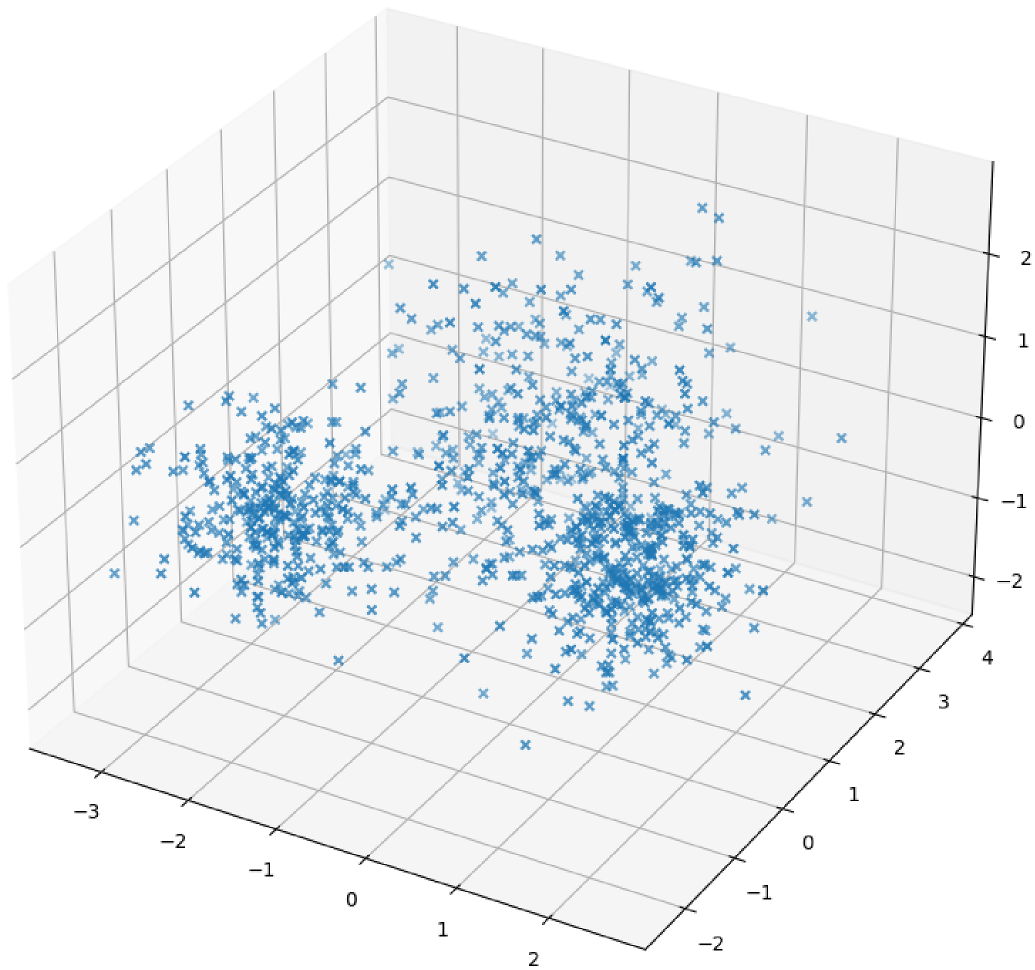
```
    fig.add_subplot(1,len(kWidthList),1+i,aspect=1.)
    plt.imshow(img.
  ↪transpose(),extent=(rng[0],rng[1],rng[0],rng[1]),origin="lower")
    plt.title("kWidth={:.02f}".format(kWidth))
plt.tight_layout()
plt.show()
```



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
```

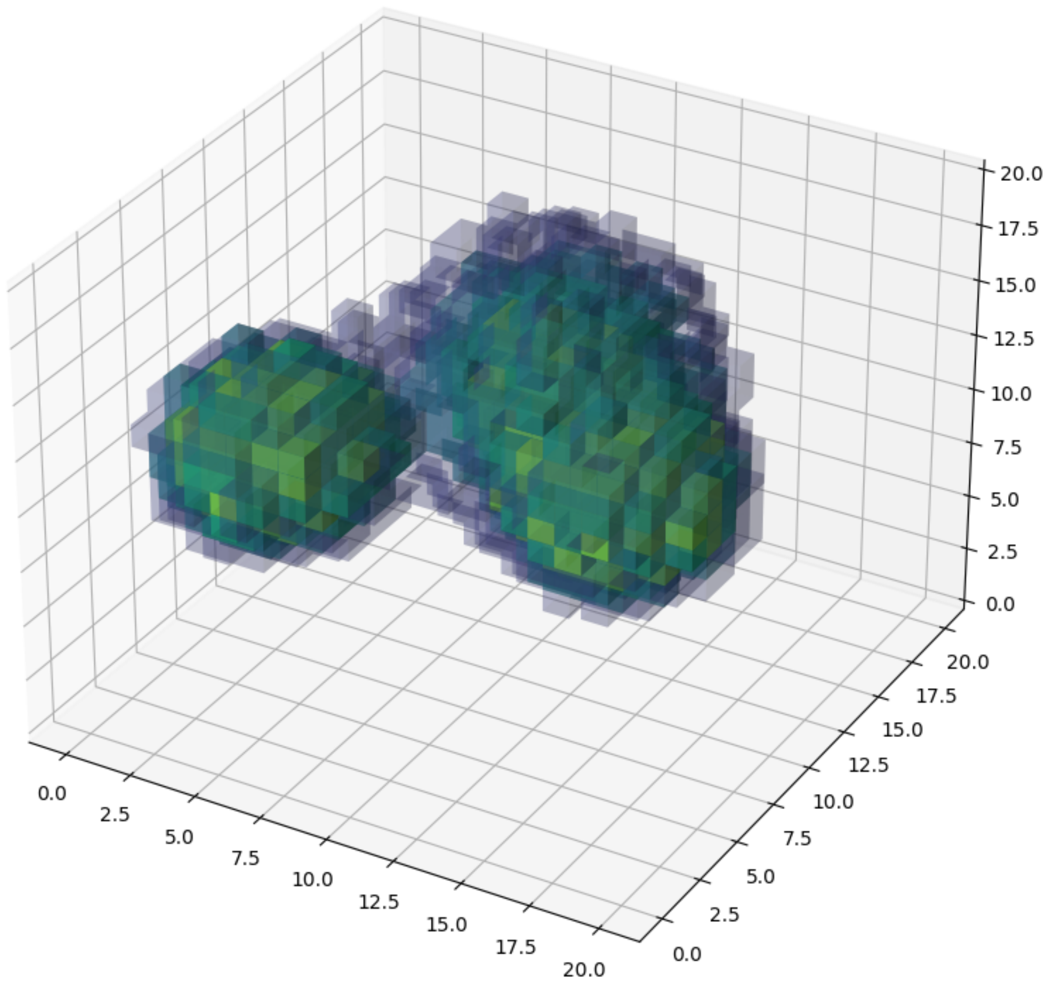
```
pts=np.zeros((0,dim),dtype=np.double)
for mean,std,weight in zip(meanList,stdList,weightList):
    pts=np.concatenate((pts,mean+std*np.random.
 ↪normal(size=(int(nX*weight),dim))))

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!

[ ]:
```

```