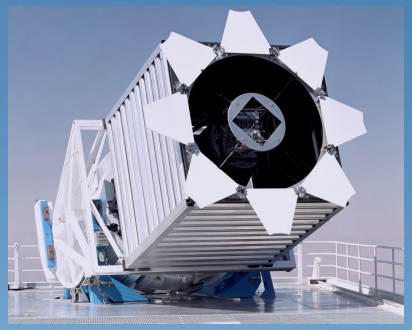
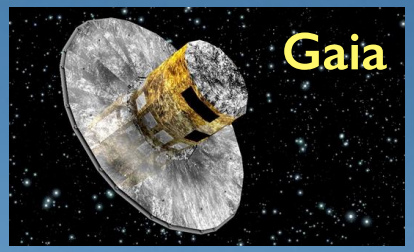
# Week 9: Density estimation and clustering. I

Željko Ivezić and Mario Jurić, Department of Astronomy, UW

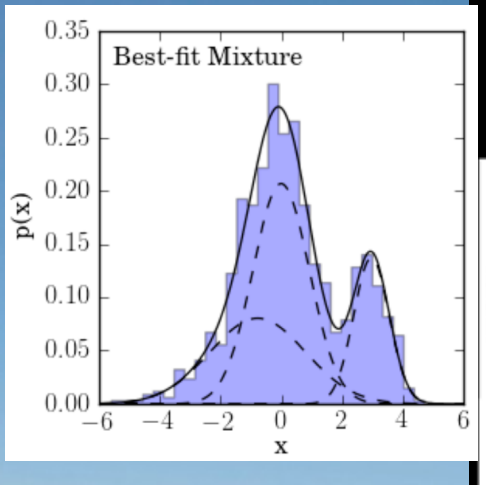
LSST

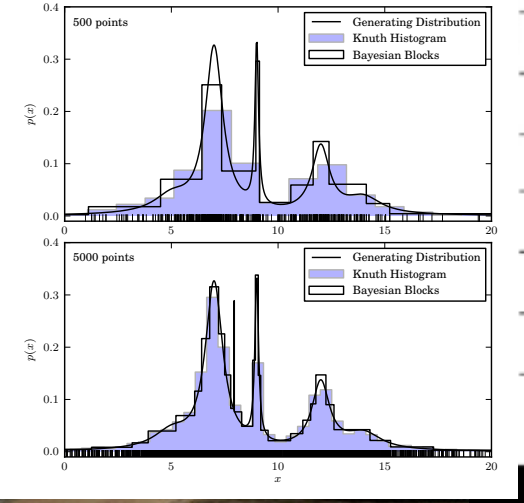


SDSS

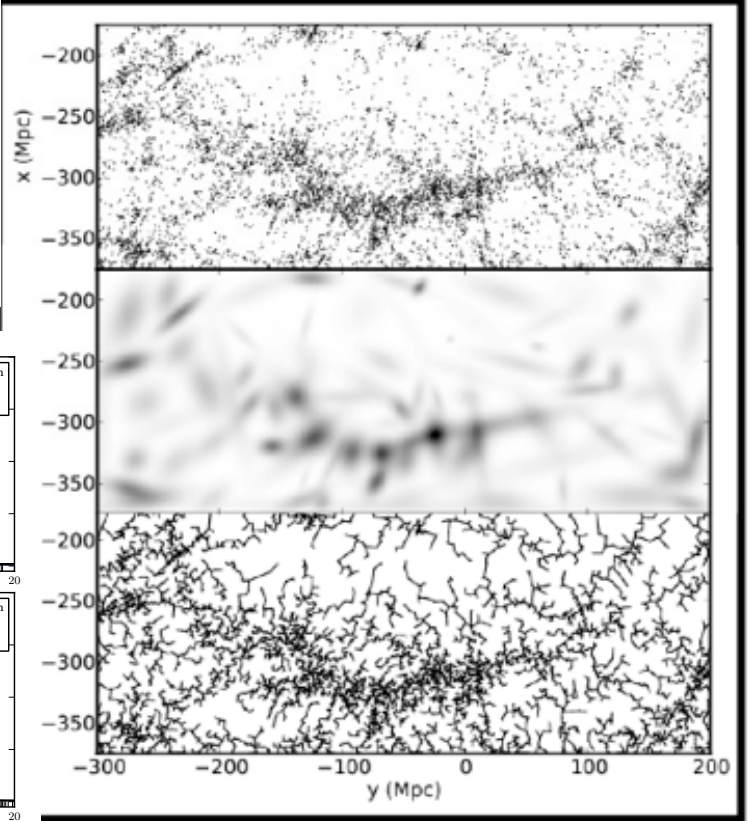


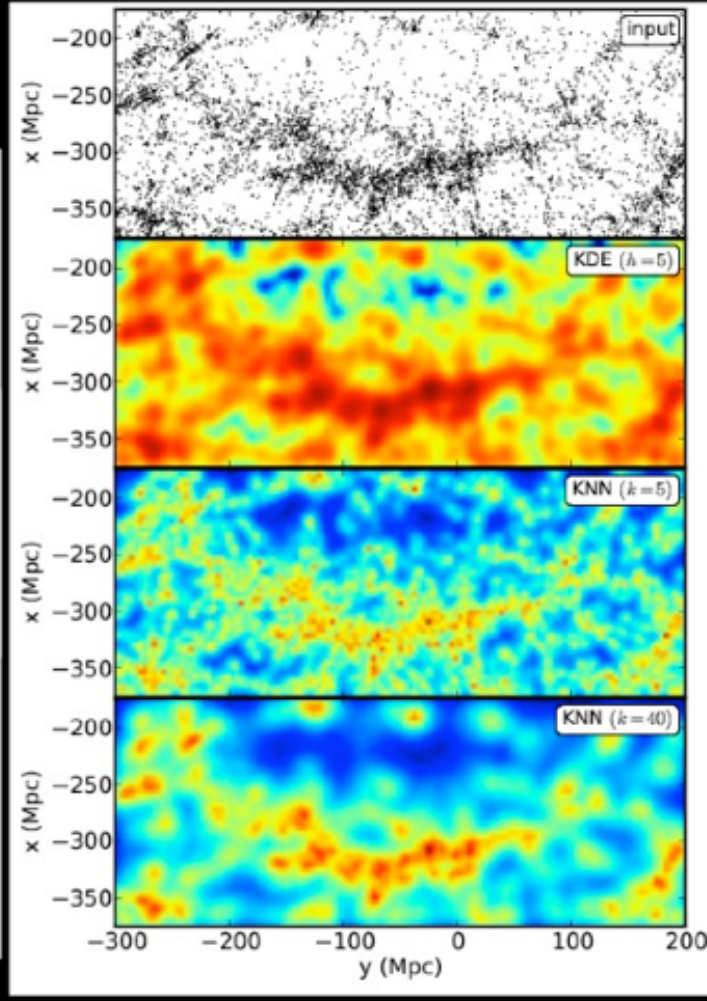

# Density Estimation




## Clustering and Density Estimation: SDSS Great Wall



# Outline

Density estimation is the act of estimating a continuous density field from a discretely sampled set of points drawn from that density field.

- One-dimensional introduction
  - Knuth's histograms
  - Scargle's Bayesian Blocks algorithm
  - Gaussian Mixture models
  - kernel density estimates (KDE)
  - the Wiener filter and connection to KDE
- Density estimation in high-D
  - high-D KDE
  - Bayesian nearest neighbor method
  - Extreme Deconvolution in high-D

# What is a histogram?

- ---> Data modeled by a step function

Assuming that we have selected a bin size, $\Delta_b$, the $N$ values of $x_i$ are sorted into $M$ bins, with the count in each bin $n_k$, $k = 1, \ldots, M$. If we want to express the results as a properly normalized $f(x)$, with the values $f_k$ in each bin, then it is customary to adopt

$$f_k = \frac{n_k}{\Delta_b N}. \tag{4.80}$$

The unit for $f_k$ is the inverse of the unit for $x_i$.

Each estimate of $f_k$ comes with some uncertainty. It is customary to assign "error bars" for each $n_k$ equal to $\sqrt{n_k}$ and thus the uncertainty of $f_k$ is

$$\sigma_k = \frac{\sqrt{n_k}}{\Delta_b N}. \tag{4.81}$$

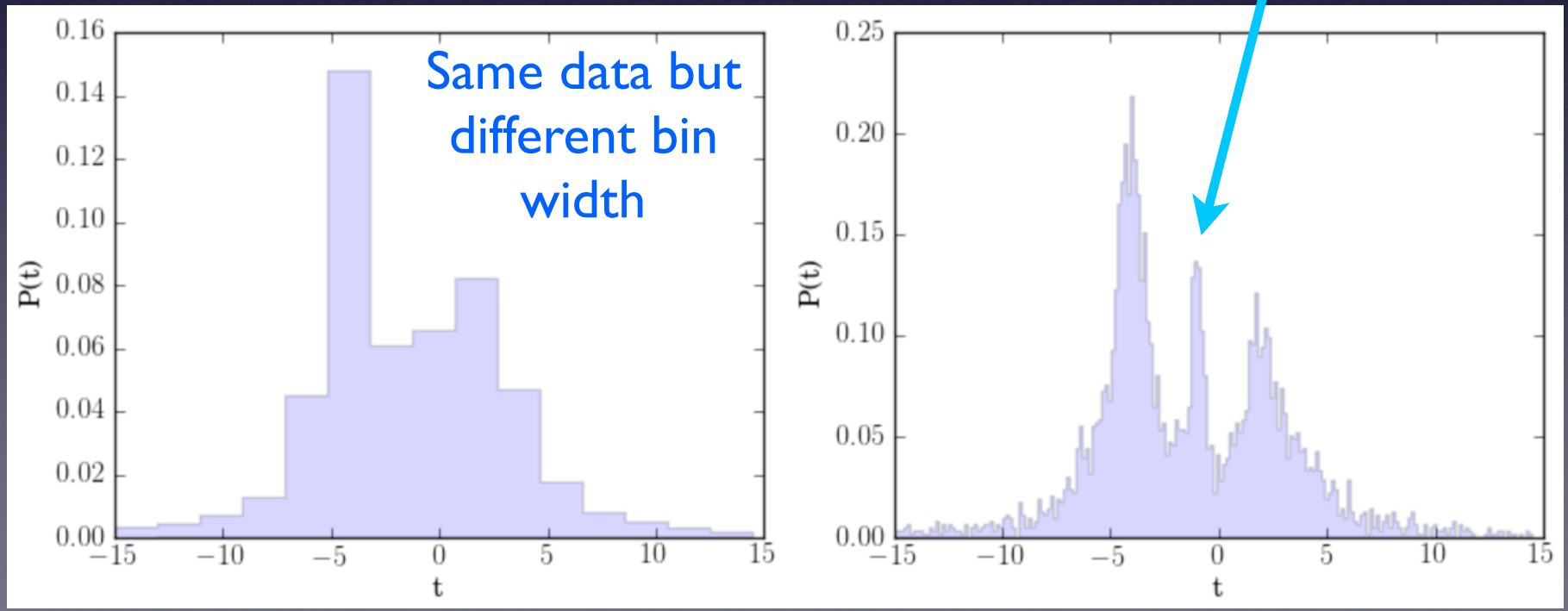
This practice assumes that $n_k$ are scattered around the true values in each bin ($\mu$) according to a Gaussian distribution, and that error bars enclose the 68% confidence range for the true value. However, when counts are low this assumption of Gaussianity breaks down and the Poisson distribution should be used instead. For example, according to the Gaussian distribution, negative values of $\mu$ have nonvanishing probability for small $n_k$ (if $n_k = 1$, this probability is 16%). This is clearly wrong since in counting experiments, $\mu \geq 0$. Indeed, if $n_k \geq 1$, then even $\mu = 0$ is clearly ruled out. Note also that $n_k = 0$ does not necessarily imply that $\mu = 0$: even if $\mu = 1$, counts will be zero in $1/e \approx 37\%$ of cases. Another problem is that the range $n_k \pm \sigma_k$ does not correspond to the 68% confidence interval for true $\mu$ when $n_k$ is small. These issues are important when fitting

# What is a histogram?

- ---> Data modeled by a step function
- How do we determine/estimate/guess the bin width?
- Do all the bins have to have the same width?
- Do we really have to bin data to estimate model parameters?

# What is a histogram?

- ---> Data modeled by a step function
- How do we determine/estimate/guess the bin width?
- Do all the bins have to have the same width?
- Do we really have to bin data to estimate model parameters?



# Simple rules for estimating bin width

Various proposed methods for choosing optimal bin width typically suggest a value proportional to some estimate of the distribution's scale, and decreasing with the sample size. The most popular choice is "Scott's rule" which prescribes a bin width

$$\Delta_b = \frac{3.5\sigma}{N^{1/3}}, \tag{4.78}$$

where $\sigma$ is the sample standard deviation, and $N$ is the sample size. This rule asymptotically minimizes the mean integrated square error (see eq. 4.14) and assumes that the underlying distribution is Gaussian; see [22]. An attempt to generalize this rule to non-Gaussian distributions is the Freedman–Diaconis rule,

$$\Delta_b = \frac{2(q_{75} - q_{25})}{N^{1/3}} = \frac{2.7\sigma_G}{N^{1/3}}, \tag{4.79}$$

which estimates the scale ("spread") of the distribution from its interquartile range (see [12]). In the case of a Gaussian distribution, Scott's bin width is 30% larger than the Freedman–Diaconis bin width. Some rules use the extremes of observed values to estimate the scale of the distribution, which is clearly inferior to using the interquartile range when outliers are present.

Although the Freedman–Diaconis rule attempts to account for non-Gaussian distributions, it is too simple to distinguish, for example, multimodal and unimodal distributions that have the same $\sigma_G$

# Knuth's rule for estimating bin width

Knuth shows that the best piecewise constant model has the number of bins, $M$, which maximizes the following function (up to an additive constant, this is the logarithm of the posterior probability):

data

$$F(M|\{x_i\},I)) = N\,\log M + \log\left[\Gamma\left(\frac{M}{2}\right)\right] - M\log\left[\Gamma\left(\frac{1}{2}\right)\right] - \log\left[\Gamma\left(N+\frac{M}{2}\right)\right] + \sum_{k=1}^{M}\log\left[\Gamma\left(n_k+\frac{1}{2}\right)\right], \quad (5.107)$$

This is Bayesian model selection of M models

where $\Gamma$ is the gamma function, and $n_k$ is the number of measurements $x_i$, $i = 1,\ldots,N$, which are found in bin $k$, $k = 1,\ldots,M$. Although this expression is more involved than the "rules of thumb" listed in §4.8.1, it can be easily evaluated for an *arbitrary* data set.

Knuth derived eq. 5.107 using Bayesian model selection and treating the histogram as a piecewise constant model of the underlying density function. By assumption, the bin width is constant and the number of bins is the result of model selection. Given the number of bins, $M$, the model for the underlying pdf is

$$h(x) = \sum_{k=1}^{M} h_k \Pi(x|x_{k-1},x_k), \quad (5.108)$$

where the boxcar function $\Pi = 1$ if $x_{k-1} < x \le x_k$, and 0 otherwise. The $M$ model parameters, $h_k$, $k = 1,\ldots,M$, are subject to normalization constraints, so that there are only $M-1$ free parameters. The uninformative prior distribution for $\{h_k\}$ is given by

$$p(\{h_k\}|M,I) = \frac{\Gamma(\frac{M}{2})}{\Gamma(\frac{1}{2})^M}\left[h_1\,h_2\ldots h_{M-1}\left(1-\sum_{k=1}^{M-1}h_k\right)\right]^{-1/2}, \quad (5.109)$$

which is known as the Jeffreys prior for the multinomial likelihood. The joint data likelihood is a multinomial distribution (see §3.3.3)

$$p(\{x_i\}|\{h_k\},M,I) \propto h_1^{n_1}\,h_2^{n_2}\ldots h_M^{n_M}. \quad (5.110)$$

# Scargle's Bayesian Blocks algorithm

- Knuth's method assumes constant bin width!
- We can use the same Bayesian machinery to generalize Knuth's model to varying bin width (these are additional model parameters, just like the other ones)

In the Bayesian blocks formalism, the data are segmented into *blocks*, with the borders between two blocks being set by *changepoints*. Using a Bayesian analysis based on Poissonian statistics within each block, an objective function, called the log-likelihood fitness function, can be defined for each block:

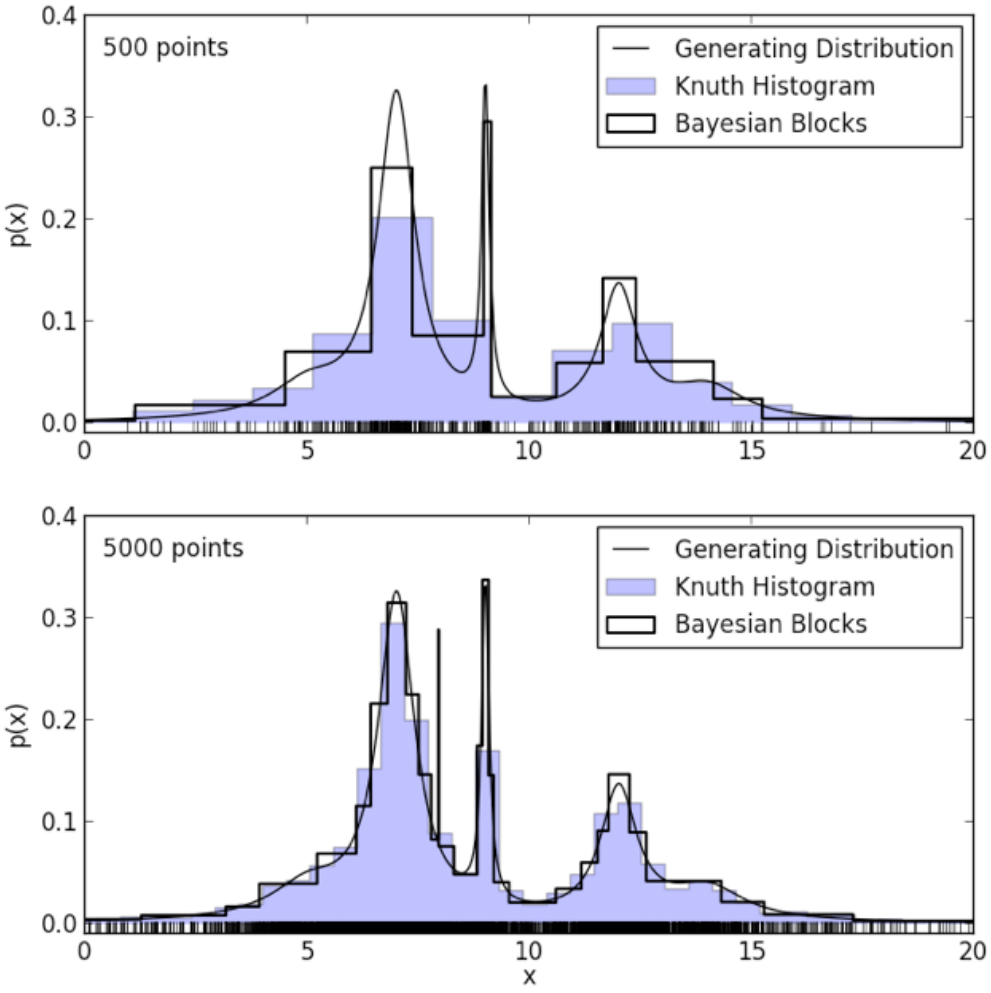
$$F(N_i, T_i) = N_i(\log N_i - \log T_i), \tag{5.113}$$

where $N_i$ is the number of points in block $i$, and $T_i$ is the width of block $i$ (or the duration, in time-series analysis). Because of the additive nature of log-likelihoods, the fitness function for any set of blocks is simply the sum of the fitness functions for each individual block. This feature allows for the configuration space to be explored quickly using dynamic programming concepts: for more information see [31] or the Bayesian blocks implementation in AstroML.

[31] Scargle, J. D., J. P. Norris, B. Jackson, and J. Chiang (2012).
Studies in astronomical time series analysis. VI.
Bayesian block representations. ArXiv:astro-ph/1207.5578.

# Comparing Knuth's rule and Bayesian Blocks

- make this plot by running
  %run fig_bayes_blocks.py




Note that Knuth's method does not find the narrow peak in the middle for the smaller dataset!

**Bayesian Blocks method gives you the best step function that describes your data. It is excellent for low-count data and for time-series analysis!**

**But remember that data errors are not included!!!**

# Gaussian Mixture Models

The likelihood of a datum $x_i$ for a Gaussian mixture model is given by

$$p(x_i|\boldsymbol{\theta}) = \sum_{j=1}^{M} \alpha_j \mathcal{N}(\mu_j, \sigma_j), \tag{4.18}$$

where dependence on $x_i$ comes via a Gaussian $\mathcal{N}(\mu_j, \sigma_j)$. The vector of parameters $\boldsymbol{\theta}$ that need to be estimated for a given data set $\{x_i\}$ includes normalization factors for each Gaussian, $\alpha_j$, and its parameters $\mu_j$ and $\sigma_j$. It is assumed that the data have negligible uncertainties (e.g., compared

Usually solved using **Expectation Maximization algorithm** (for a good tutorial see ArXiv:statistics/1105.1476)

***How to choose the number of classes?***

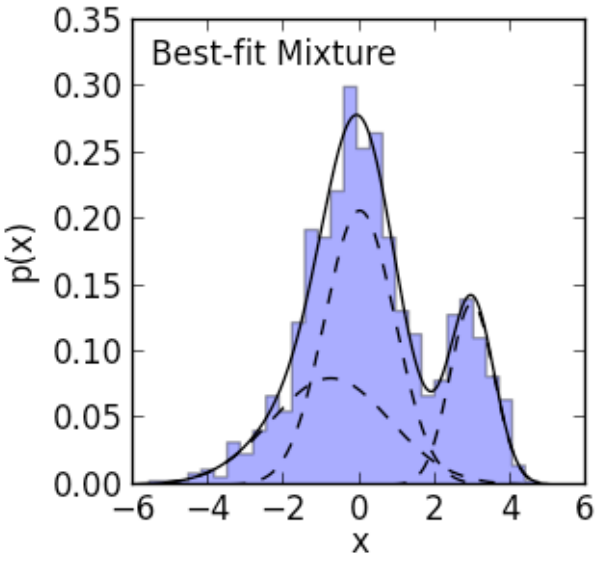
We have assumed in the above discussion of the EM algorithm that the number of classes in a mixture, $M$, is known. As $M$ is increased, the description of the data set $\{x_i\}$ using a mixture model will steadily improve. On the other hand, a very large $M$ is undesired—after all, $M = N$ will assign a mixture component to each point in a data set. How do we choose $M$ in practice?

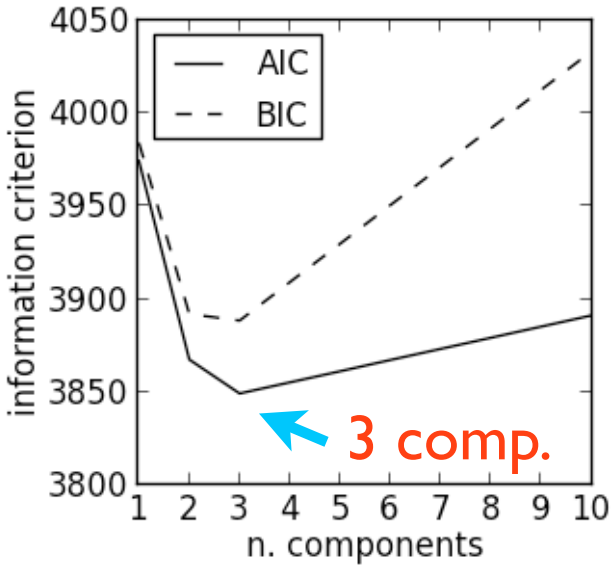
Usually determined using Bayesian Information Criterion (**BIC**), or **cross-validation**

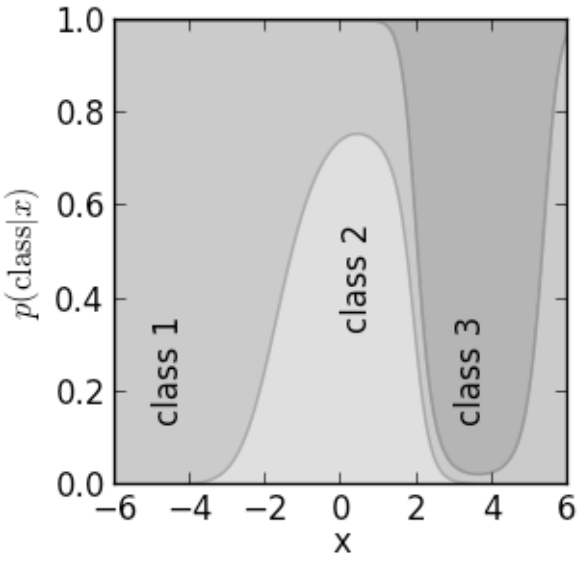
# Gaussian Mixture Models

- make this plot by running
  %run fig_GMM_1D.py

**This best GMM fit is also density estimation! The only difference compared to histograms is in the chosen fitting model function!**




Uses Expectation Maximization method implemented in scikit-learn

Bayesian Information Criterion: it tells you how many components data support!

**Example of classification: details in Week 10**

# Gaussian Mixture Models with Errors

The previous example assumed that uncertainty in data values (x) was negligible. What do we do when this is not the case?

For example, in case of homescedastic Gaussian measurement errors, the three GMM components from the last example would be broadened (the measurement error would be added in quadrature to their intrinsic widths).

A recently introduced application of GMM with errors in astronomical context was dubbed "**Extreme Deconvolution**" (see Bovy et al. 2009, ArXiv:0905.2979)

**Extreme Deconvolution works in multi-dimensional spaces too, and naturally treats noisy, heterogeneous, and incomplete data.**

More details and a high-D example later...

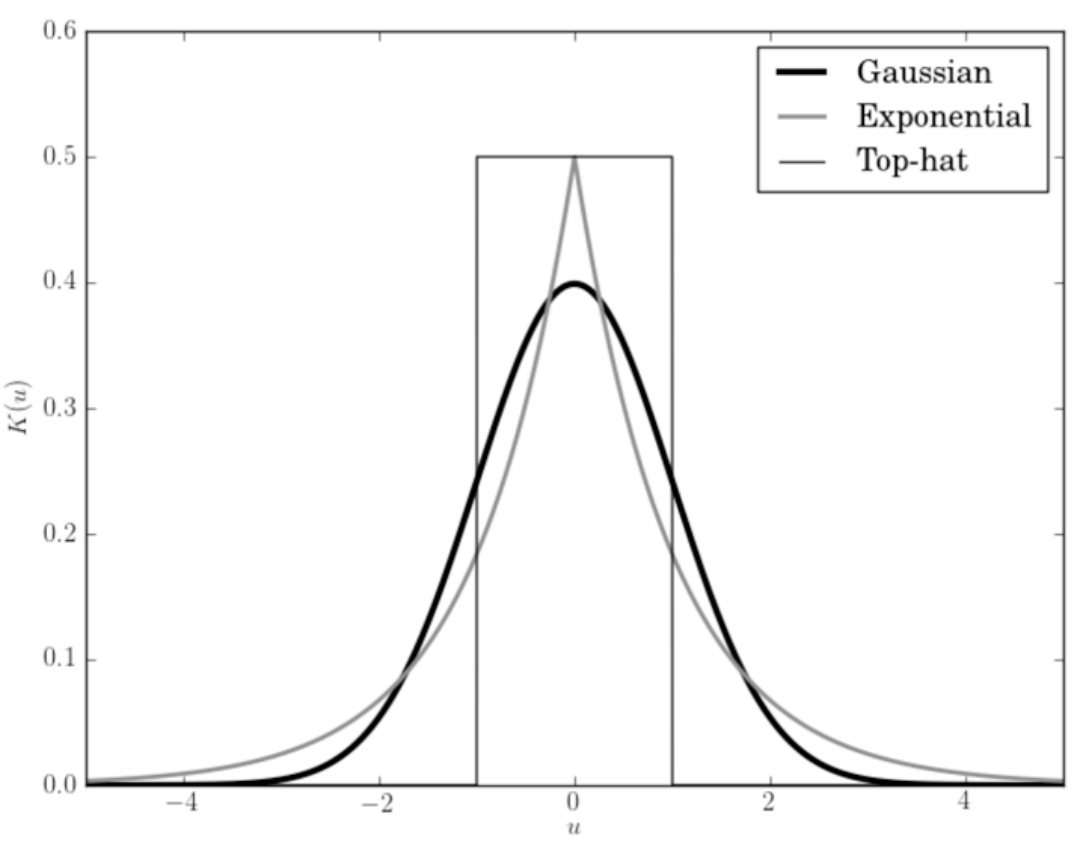
# Kernel Density Estimates (KDE)

the underlying pdf is

$$h(x) = \sum_{k=1}^{M} h_k \Pi(x|x_{k-1}, x_k),$$

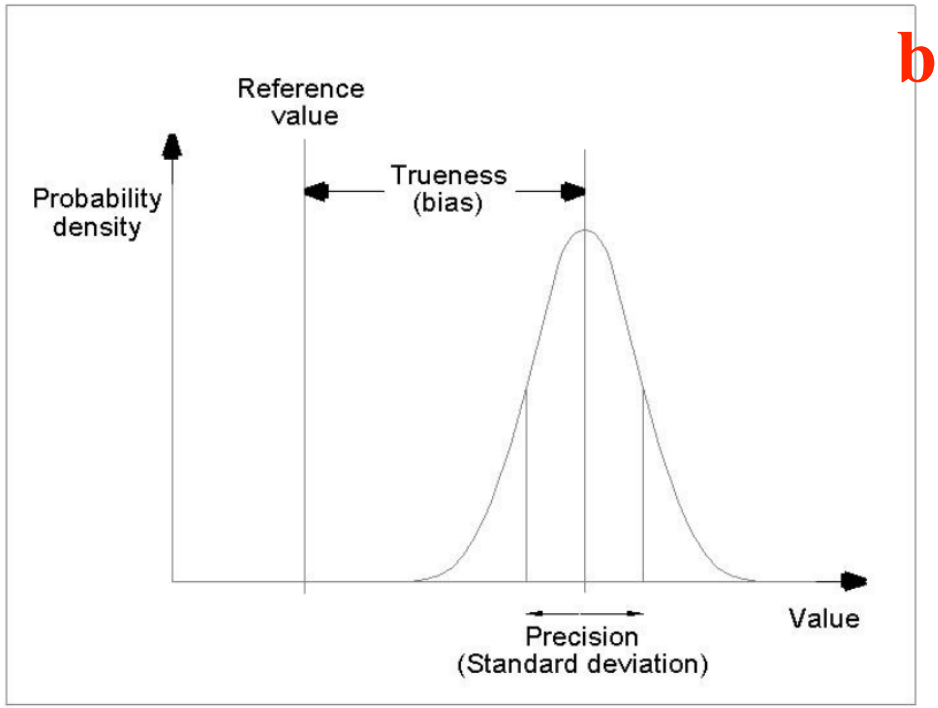
where the boxcar function $\Pi = 1$ if $x_{k-1} < x \leq x_k$, and 0 otherwise.

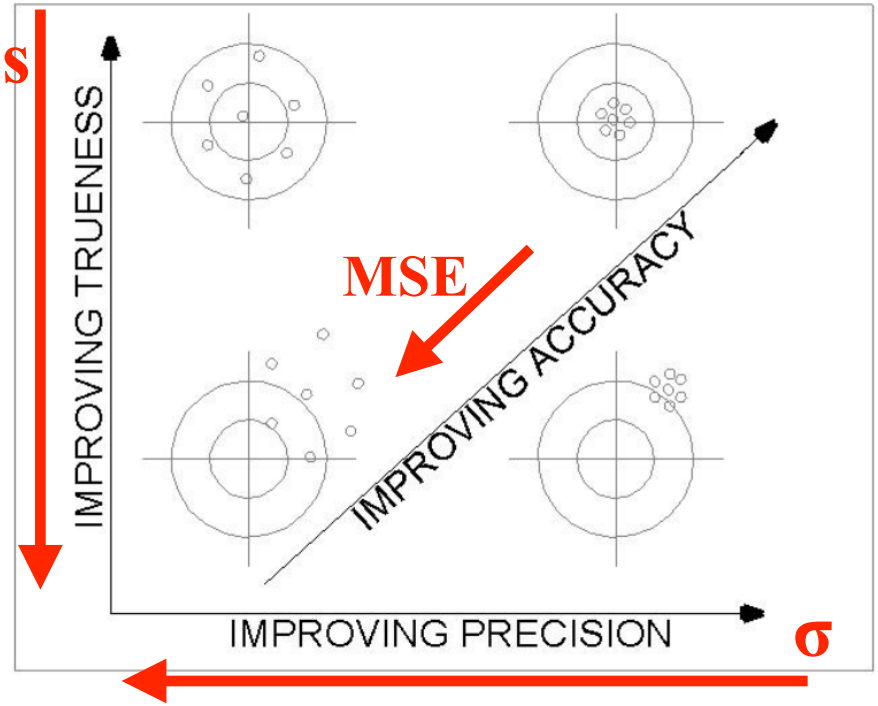
When computing histogram, we replace each object by the boxcar (top-hat) function. This function is called **kernel**. But of course we can use any other function (well, non-negative, normalized to 1, zero-mean, finite variance). This is the main idea of the KDE; it also works in high-D spaces.



# • Measurements with known errors

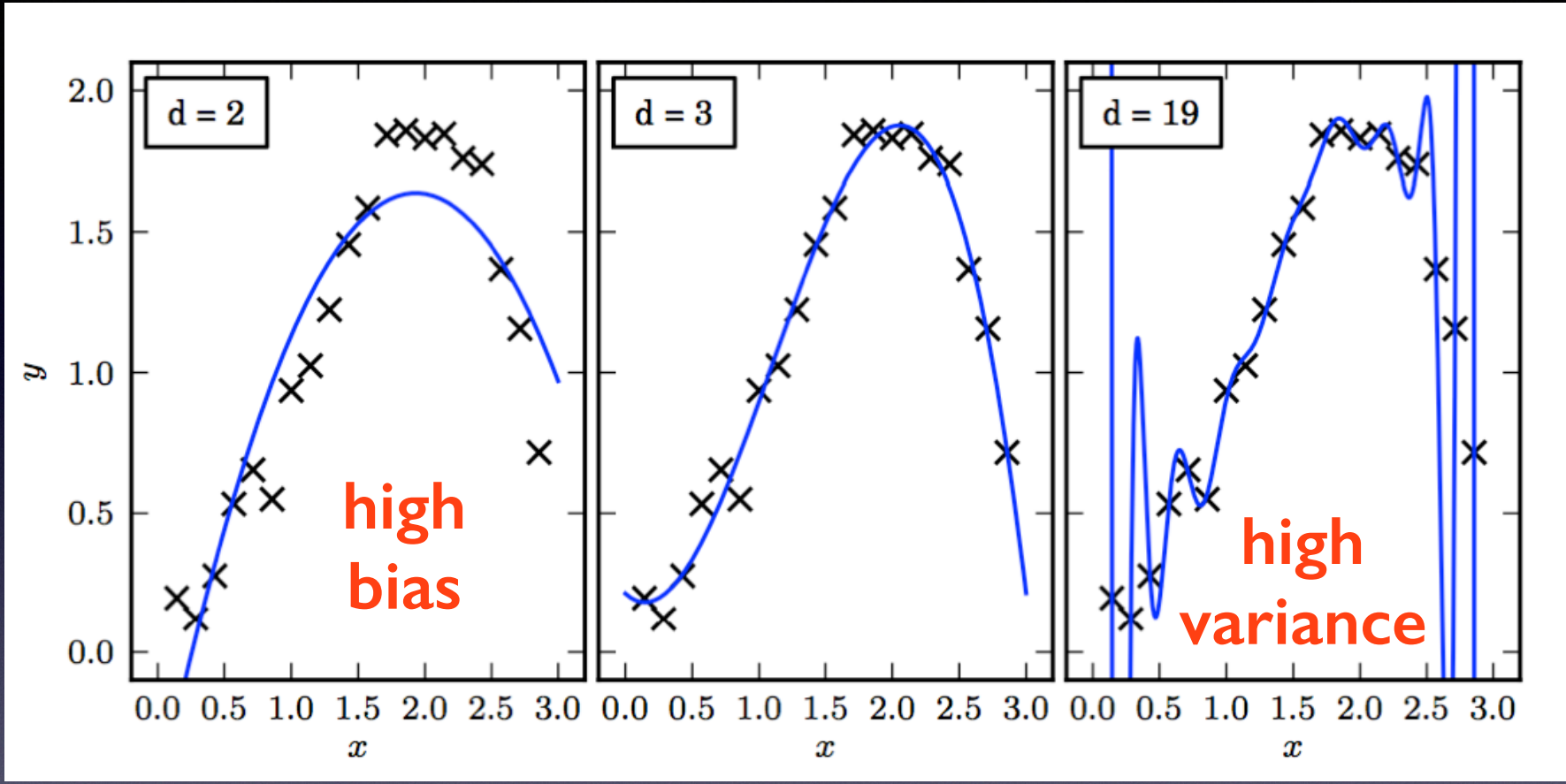
Let's assume that we have N measurements $x_i$, and that for each measurement we know the corresponding error distribution, that is, the expected distribution of $x_i$ around the true value μ (which we want to estimate)




Mean Squared Error: $\mathrm{MSE} = V + \mathrm{bias}^2$ (V=variance=$\sigma^2$)

# Bias vs. variance tradeoff



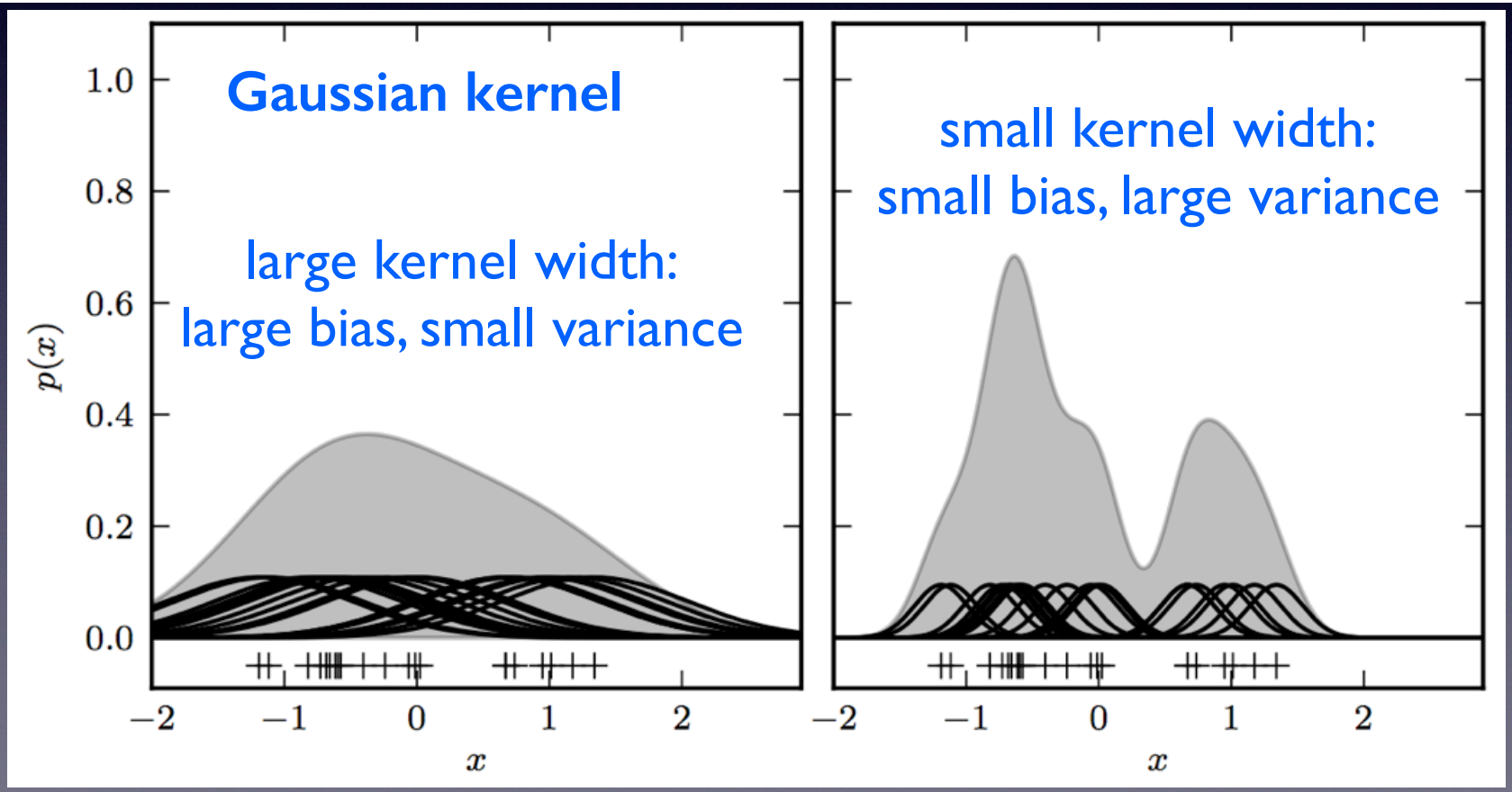

**high bias:** "can't fit" data points
**high variance:** "oscillations" between data points (how would we know? c.f. cross-validation method: "leave one out")

# Kernel Density Estimates (KDE)

the underlying pdf is

$$h(x) = \sum_{k=1}^{M} h_k \Pi(x|x_{k-1}, x_k),$$

where the boxcar function $\Pi = 1$ if $x_{k-1} < x \leq x_k$, and 0 otherwise.



# Kernel Density Estimates (KDE)

## How do we choose the kernel width?

That is, the kernel width is a free parameter to be determined from data (despite KDE being called a non-parametric method)

Ideally we would select a kernel that has $h$ as small as possible. If $h$ becomes too small we increase the variance of the density estimation. If $h$ is too large then the variance decreases but at the expense of the bias in the derived density. The optimal kernel function, in terms of minimum variance, turns out to be

$$K(x) = \frac{3}{4}\left(1 - x^2\right) \tag{6.9}$$

for $|x| \leq 1$ and 0 otherwise; see [37]. This function is called the *Epanechnikov kernel.*

## We choose the kernel width by **cross-validation:**

Cross-validation can be used for any cost function (see §8.11); we just have to be able to evaluate the cost on *out-of-sample* data (i.e., points not in the training set). If we consider the likelihood cost for KDE, for which we have the leave-one-out *likelihood cross-validation*, then the cost is simply the sum over all points in the data set (i.e., $i = 1, \ldots, N$) of the log of the likelihood of the density, where the density, $\widehat{f}_{h,-i}(x_i)$, is estimated leaving out the $i$th data point. This can be written as

$$\mathrm{CV}_l(h) = \frac{1}{N}\sum_{i=1}^{N} \log \widehat{f}_{h,-i}(x_i), \tag{6.5}$$

# Kernel Density Estimates (KDE)

The optimal KDE bandwidth decreases at the rate $\mathcal{O}(N^{-1/5})$ (in a one-dimensional problem), and the error of the KDE using the optimal bandwidth converges at the rate $\mathcal{O}(N^{-4/5})$; it can be shown that histograms converge at a rate $\mathcal{O}(N^{-2/3})$; see [35]. KDE is, therefore, theoretically superior to the histogram as an estimator of the density. It can also be shown that there does not exist a density estimator that converges faster than $\mathcal{O}(N^{-4/5})$ (see Wass10).

## astroML implementation is very easy to use:

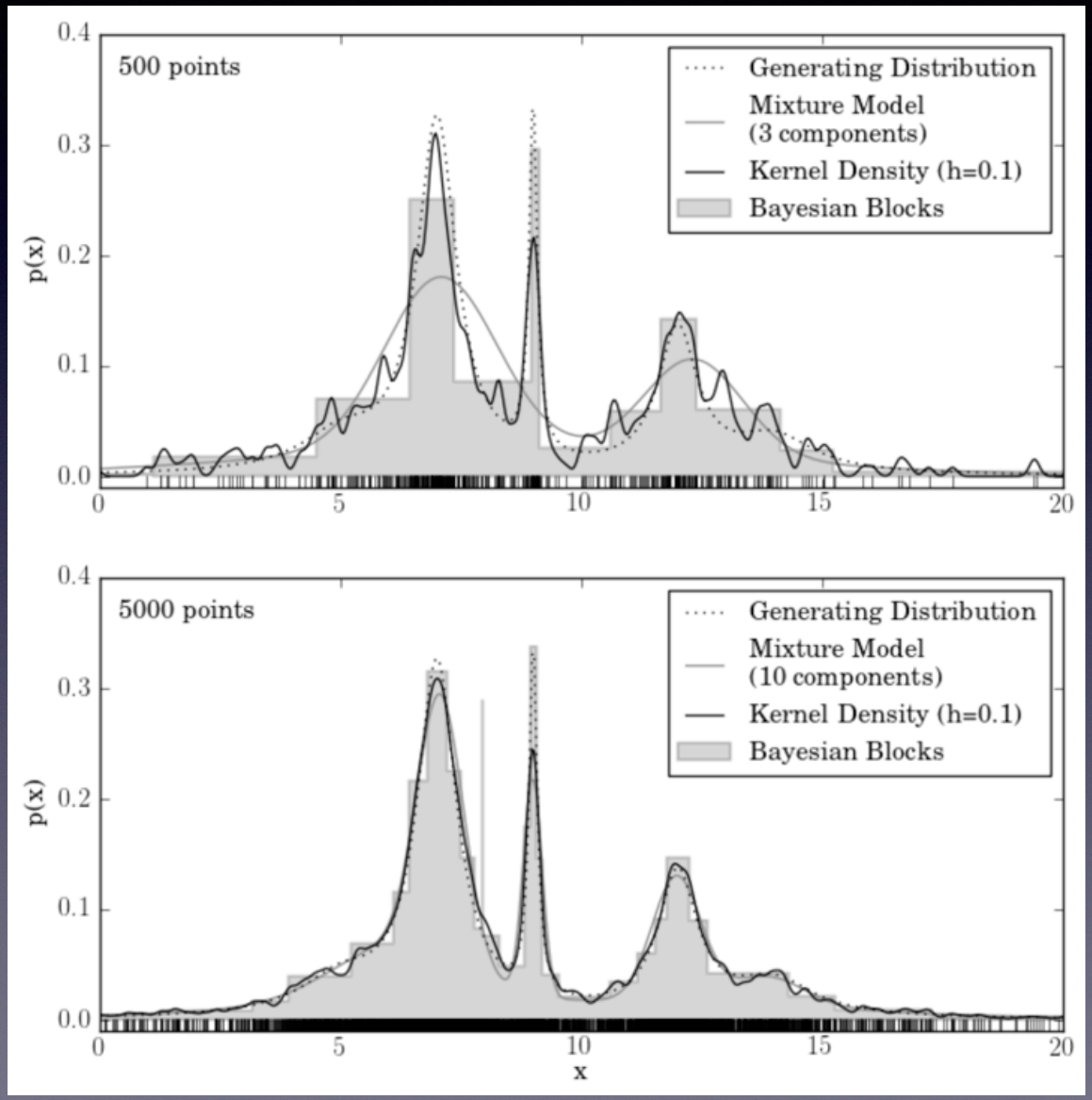
AstroML contains an implementation of kernel density estimation in $D$ dimensions using the above kernels:

```
import numpy as np
from astroML.density_estimation import KDE

X = np.random.normal(size=(1000, 2))  # 1000 points in 2 dims
kde = KDE('gaussian', h=0.1)  # select the gaussian kernel
kde.fit(X)  # fit the model to the data
dens = kde.eval(X)  # evaluate the model at the data
```

# Comparing different methods:

- you **could** make this plot by running
  %run fig_GMM_density_estimation.py




**Key point:** for large datasets, all methods result in similar estimates

puzzle: note the spike at x~8 for the BB algorithm in the bottom panel

# Data smoothing

- directly related to density estimation
- High-pass filtering: estimating the baseline
- Low-pass filtering: data smoothing

# Data smoothing

The power spectrum for common Gaussian noise is flat and will extend to frequencies as high as the Nyquist limit, $f_N = 1/(2\Delta t)$. If the data are band limited to a lower frequency, $f_c < f_N$, then they can be smoothed without much impact by suppressing frequencies $|f| > f_c$. Given a filter in frequency space, $\Phi(f)$, we can obtain a smoothed version of data by taking the inverse Fourier transform of

$$\hat{Y}(f) = Y(f)\,\Phi(f), \tag{10.19}$$

where $Y(f)$ is the discrete Fourier transform of data. At least in principle, we could simply set $\Phi(f)$ to zero for $|f| > f_c$, but this approach would result in ringing (i.e., unwanted oscillations) in the signal. Instead, the optimal filter for this purpose is constructed by minimizing the MISE between $\hat{Y}(f)$ and $Y(f)$ (for detailed derivation see NumRec) and is called the Wiener filter:

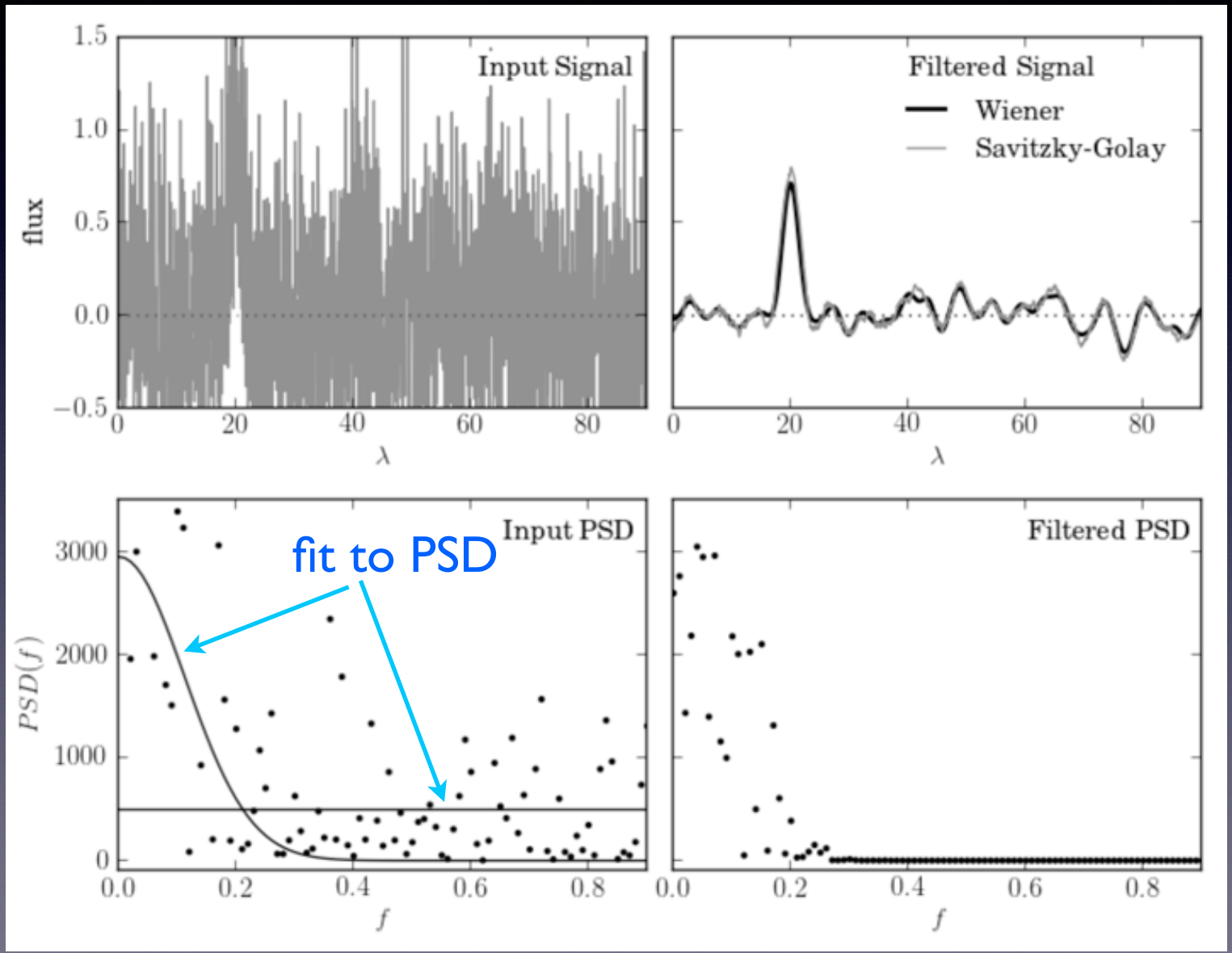
$$\Phi(f) = \frac{P_S(f)}{P_S(f) + P_N(f)}. \tag{10.20}$$

Here $P_S(f)$ and $P_N(f)$ represent components of a two-component (signal and noise) fit to the PSD of input data, $\mathrm{PSD}_Y(f) = P_S(f) + P_N(f)$, which holds as long as the signal and noise are uncorrelated. Given some assumed form of signal and noise, these terms can be determined from a fit to the observed PSD, as illustrated by the example shown in figure 10.10. Even when the fidelity of the PSD fit is not high, the resulting filter performs well in practice (the key features are that $\Phi(f) \sim 1$ at small frequencies and that it drops to zero at high frequencies for a band-limited signal).

# Data smoothing

- make this plot by running
  %run fig_wiener_filter.py

**Savitzky-Golay filter:**
local polynomial regression (here 4)
works for unevenly sampled data!




if error, add this to call in line 40:
use_fft=False

# KDE - Wiener filter connection

- make this plot by running
  %run fig_wiener_KDE.py

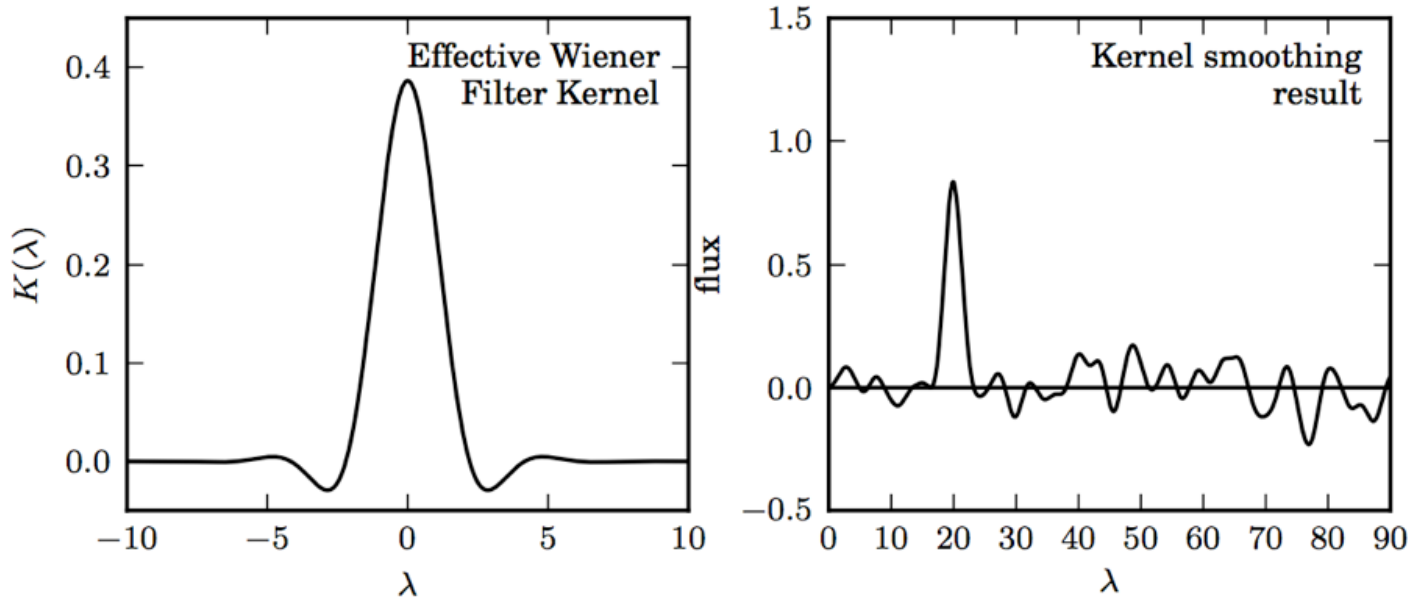



Figure 10.11.: The left panel shows the inverse Fourier transform of the Wiener filter $\Phi(f)$ applied in figure 10.10. By the convolution theorem, the Wiener-filtered result is equivalent to the convolution of the unfiltered signal with the kernel shown above, and thus Wiener filtering and kernel density estimation (KDE) are directly related. The right panel shows the data smoothed by this kernel, which is equivalent to the Wiener filter smoothing in figure 10.10.

Unlike FFT needed for Wiener filtering, KDE of course also works with unevenly sampled data!