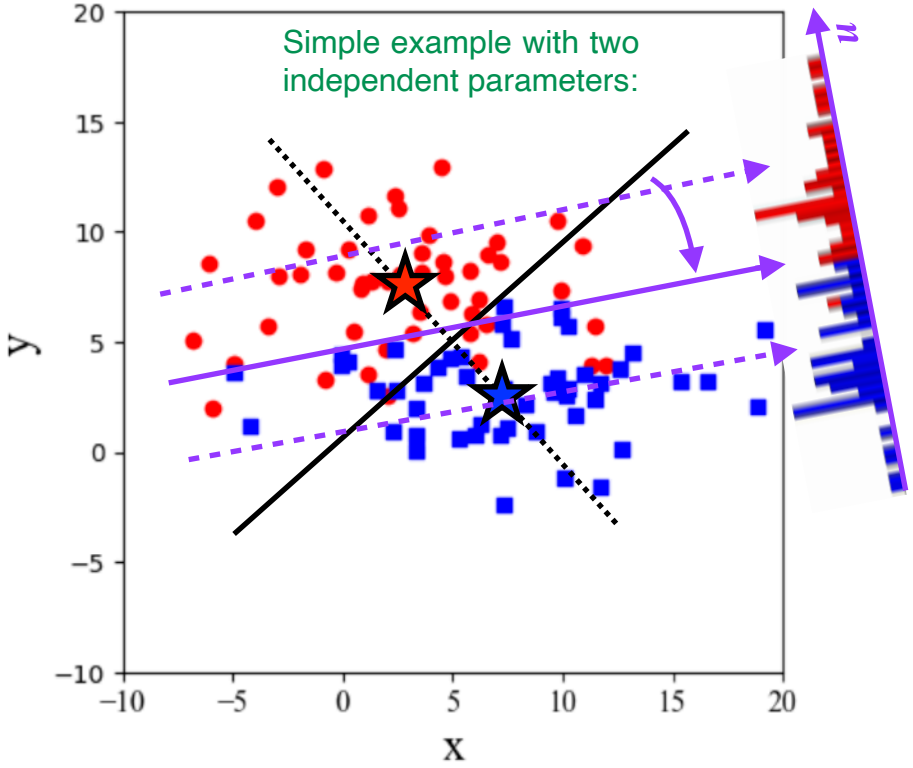
Lecture 12:

Data-Driven Approaches

- Fisher Discriminant
- Kernel Density Estimation
- Gaussian Processes
- Bayesian Optimisation

Fisher Linear Discriminant for Class Separation



Let's pick a new variable to act as a discriminant between the two classes that is some linear combination of x and y:

$$u \equiv w_x x + w_y y$$

(arbitrarily defining $u=0$ when $x=y=0$)

Want to choose values for w_x and w_y to maximise the mean distance (or variance) between the classes, while minimising the variances within each class so as to give the cleanest separation.

Fisher thus proposed maximising:

$$J = \frac{(\mu_1 - \mu_2)^2}{\sigma_1^2 + \sigma_2^2} \simeq \frac{(\overline{u_1} - \overline{u_2})^2}{s_1^2 + s_2^2}$$

$$J = \frac{[(w_x\overline{x_1} + w_y\overline{y_1}) - (w_x\overline{x_2} + w_y\overline{y_2})]^2}{[(w_x s_{x_1})^2 + (w_y s_{y_1})^2] + [(w_x s_{x_2})^2 + (w_y s_{y_2})^2]}$$

$$J = \frac{[(w_x(\overline{x_1} - \overline{x_2}) + w_y(\overline{y_1} - \overline{y_2})]^2}{[w_x^2(s_{x_1}^2 + s_{x_2}^2) + w_y^2(s_{y_1}^2 + s_{y_2}^2)]}$$

$$J = \frac{[(w_x(\overline{x_1} - \overline{x_2}) + w_y(\overline{y_1} - \overline{y_2})]^2}{[w_x^2(s_{x_1}^2 + s_{x_2}^2) + w_y^2(s_{y_1}^2 + s_{y_2}^2)]}$$

$$\frac{dJ}{dw_x} = \frac{2[\ldots](\overline{x_1} - \overline{x_2})}{[--]} - \frac{[\ldots]^2}{[--]^2} 2w_x(s_{x_1}^2 + s_{x_2}^2) = 0$$

$$\frac{dJ}{dw_y} = \frac{2[\ldots](\overline{y_1} - \overline{y_2})}{[--]} - \frac{[\ldots]^2}{[--]^2} 2w_y(s_{y_1}^2 + s_{y_2}^2) = 0$$

take ratios: $$\frac{w_x}{w_y} = \frac{(\overline{x_1} - \overline{x_2})(s_{y_1}^2 + s_{y_2}^2)}{(\overline{y_1} - \overline{y_2})(s_{x_1}^2 + s_{x_2}^2)}$$

so choose

$$w_x = \frac{\overline{x_1} - \overline{x_2}}{s_{x_1}^2 + s_{x_2}^2} \qquad w_y = \frac{\overline{y_1} - \overline{y_2}}{s_{y_1}^2 + s_{y_2}^2}$$

"train" on data sets or simulation

Data-driven and can also work for hypotheses that are not simple, or where it is not easy to capture all relevant correlations in a likelihood

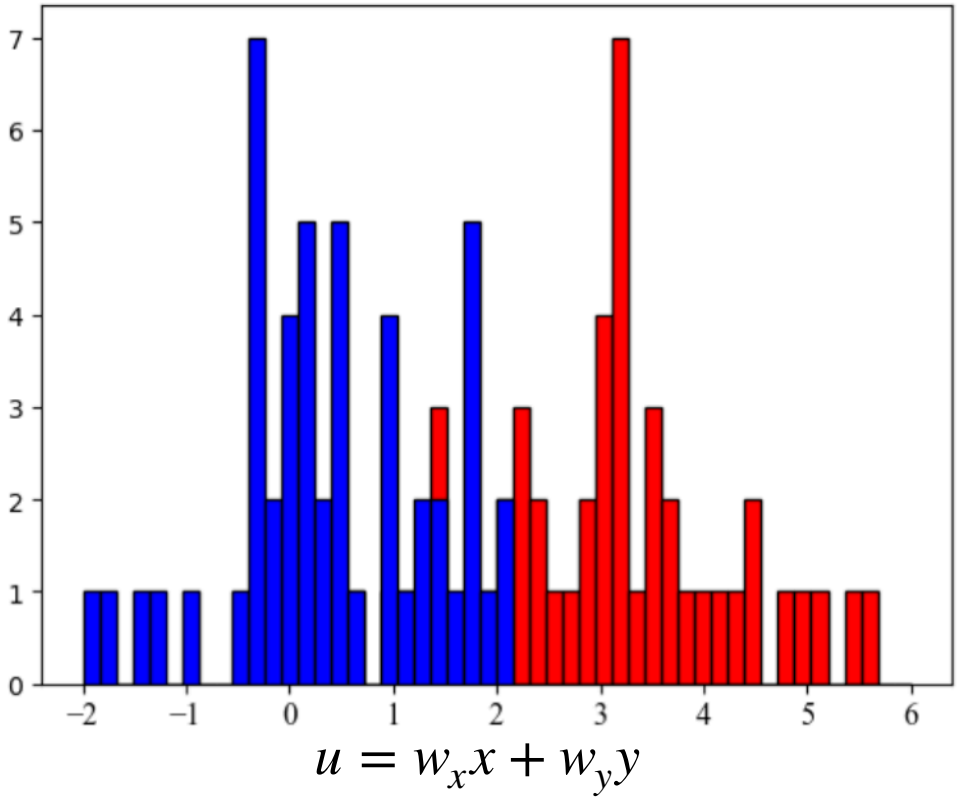
More Generally: $u = \vec{w}^T \vec{p}$

transpose vector of weights — vector of parameters

$$\vec{w} = (\Sigma_1 + \Sigma_2)^{-1}(\vec{\mu_1} - \vec{\mu_2})$$

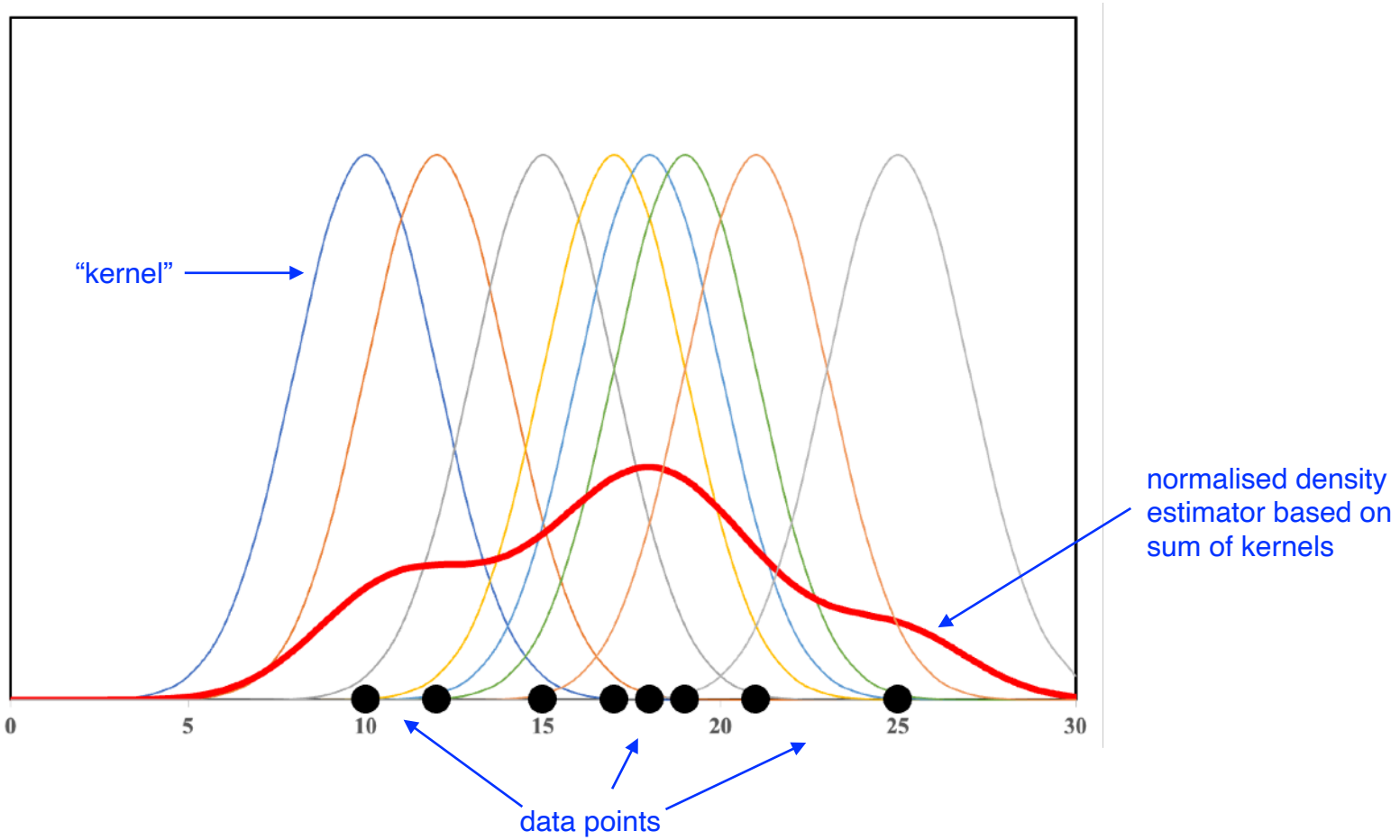
class covariance matrices — vectors of class means

There is also a form that can be used for more than 2 classes, but the optimisation of this can be trickier



Kernel Density Estimation (KDE) for Smoothing

A method for non-parametric smoothing of density distributions by associating an assumed density function or “kernel” with the values of measured data points:



The application of this is relatively simple (will show an example), but we'll dig into some of the messy details to understand why certain choices are typically made…

Assume some true but unknown density function $f(x)$,
and try to approximate it using a generic kernel density K :

estimated function value at point of interest

sum over measured data points

point of interest

data point

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right) \equiv \frac{1}{n}\sum_{i=1}^{n} K_h\left(x - X_i\right)$$

assumed kernel density about data points with scale length (bandwidth) h

How do you find the optimal bandwidth for smoothing?

You would expect that the bandwidth or scale might be comparable to the sampled rms of the distribution that you're trying to approximate.

You would also expect that the required scale would become smaller as more data points are added. If we think in terms of estimating moments, we know that the accuracy of the sampled mean and distribution rms value improves as $\sim n^{-1/2}$, so maybe a good guess is that the bandwidth scales as $\sigma n^{-1/2}$

In fact, a better estimate is a slightly less obvious $\sigma n^{-1/5}$

To get here takes a little work…

Following Wand and Jones' *, let's start by examining the bias and variance:

$$\left\langle \hat{f}(x) \right\rangle = \left\langle K_h(x) \right\rangle = \int K_h\left(x - y\right) f(y) dy$$

$$\left\langle \hat{f}^2(x) \right\rangle = (K_h^2 * f)(x)$$

bias: $$\left\langle \hat{f}(x) \right\rangle - f(x) = (K_h * f)(x) - f(x)$$

variance: $$var[\hat{f}(x)] = \frac{1}{n}[(K_h^2 * f)(x) - (K_h * f)^2(x)]$$

*Wand and Jones, "Kernel Smoothing," Chapman and Hall/CRC, 1994

Mean Squared Error

$$MSE = \left\langle (\hat{\theta} - \theta)^2 \right\rangle$$

$$= \hat{\theta}^2 + \left\langle \theta^2 \right\rangle - 2\hat{\theta}\,\langle \theta \rangle$$

$$= \left\langle \theta^2 \right\rangle - \langle \theta \rangle^2 + \left\langle \hat{\theta} - \theta \right\rangle^2$$

variance squared bias

$$\left\langle \hat{\theta} - \theta \right\rangle^2 = \hat{\theta}^2 + \langle \theta \rangle^2 - 2\hat{\theta}\,\langle \theta \rangle$$

$$-2\hat{\theta}\,\langle \theta \rangle = \left\langle \hat{\theta} - \theta \right\rangle^2 - \hat{\theta}^2 - \langle \theta \rangle^2$$

$$MSE(f) = \frac{1}{n}[(K_h^2 * f)(x) - (K_h * f)^2(x)] + [(K_h * f)(x) - f(x)]^2$$

$$MISE(f) = \frac{1}{n}\int [(K_h^2 * f)(x) - (K_h * f)^2(x)]\; dx$$

$$+ \int [(K_h * f)(x) - f(x)]^2\; dx$$

Mean Integrated Squared Error
(want to minimise!)

Let's try an approximation to make things more tractable:

$$\left\langle \hat{f}(x) \right\rangle = \int \frac{1}{h} K\left(\frac{x-y}{h}\right) f(y)\, dy$$

$$= \int K(z)\, f(x - hz)\, dz$$

$$z \equiv \frac{x-y}{h}$$

$$y = x - hz$$

$$dy = -h\, dz$$

Approximate f with a Taylor expansion around $hz=x$:

$$f(x - hz) \simeq f(x) - hzf'(x) + \frac{1}{2}h^2z^2f''(x)$$

$$\left\langle \hat{f}(x) \right\rangle \simeq f(x)\int K(z)\, dz - hf'(x)\int zK(z)\, dz + \frac{1}{2}h^2f''(x)\int z^2K(z)\, dz$$

$$= f(x) - \underset{\text{(for K even)}}{0} + \frac{1}{2}h^2f''(x)\int z^2K(z)\, dz$$

approximate bias: $$\left\langle \hat{f}(x) \right\rangle - f(x) \simeq \frac{1}{2}h^2f''(x)\int z^2K(z)\, dz$$

approximate bias:

$$\left\langle \hat{f}(x) \right\rangle - f(x) \simeq \frac{1}{2} h^2 f''(x) \underbrace{\int z^2 K(z)\, dz}_{\equiv \mu_2(K)}$$

following nomenclature of Wand and Jones*

$$var\left[\hat{f}(x)\right] = \frac{1}{nh} \int K^2(z)\, f(x - hz)\, dz - \frac{1}{n} \left\langle \hat{f}(x) \right\rangle^2$$

h → 0 as n →∞, while keeping nh > 0, so 1st term dominates for large n

& take 0th order approximation for f(x-hz) ~ f(x)

approximate variance:

$$var\left[\hat{f}(x)\right] \simeq \frac{1}{nh} f(x) \underbrace{\int K^2(z)\, dz}_{\equiv R(K)}$$

Approximate Mean Squared Error:

$$AMSE = \frac{f(x)}{nh} R(K) - \frac{h^4}{4} \left[f''(x)\right]^2 \mu_2^2(K)$$

*Wand and Jones, "Kernel Smoothing," Chapman and Hall/CRC, 1994

$$AMSE = \frac{f(x)}{nh} R(K) - \frac{h^4}{4} \left[f''(x) \right]^2 \mu_2^2(K)$$

Approximate Mean Integrated Squared Error:

$$AMISE = \frac{1}{nh} R(K) \int f(x)\ dx - \frac{h^4}{4} \mu_2^2(K) \int \left[f''(x) \right]^2\ dx$$

$$= \frac{1}{nh} R(K) - \frac{h^4}{4} \mu_2^2(K) R(f'')$$

Find value of h that minimises AMISE:

$$\frac{\partial(AMISE)}{\partial h} = -\frac{1}{nh^2} R(K) - h^3 \mu_2^2(K) R(f'') = 0$$

$$h_{min} = \left[\frac{R(K)}{\mu_2^2(K) R(f'') n} \right]^{1/5}$$

recall: $$R(K) \equiv \int K^2(z)dz \qquad \mu_2(K) \equiv \int z^2 K(z)dz$$

if we take a Gaussian Kernel density, then: $$K(z) = \frac{1}{\sqrt{2\pi}} e^{-z^2/2}$$

$$R(K) \longrightarrow \frac{1}{2\sqrt{\pi}} \qquad \mu_2(K) \longrightarrow 1$$

But we don't know the true underlying function that we're trying to approximate, so how do we evaluate:

$$R(f'') \equiv \int (f'')^2(z)dz \quad ??$$

Can try choosing some arbitrary Gaussian function to be indicative:

$$f(z) \sim \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-z_0)^2/2\sigma^2}$$

Where σ is an estimated equivalent standard deviation, either from the sampled rms or a robust percentile estimator, such as $\frac{F(>75\%) - F(>25\%)}{1.34}$

$$R(f'') \longrightarrow \sim \frac{3}{8\sqrt{\pi}} \frac{1}{\sigma^5}$$

$$R(K) \longrightarrow \frac{1}{2\sqrt{\pi}} \qquad \mu_2(K) \longrightarrow 1 \qquad R(f'') \longrightarrow \sim \frac{3}{8\sqrt{\pi}} \frac{1}{\sigma^5}$$

$$h_{min} \sim \left(\frac{4}{3}\right)^{\frac{1}{5}} n^{-1/5}\, \sigma \quad \sim n^{-1/5}\, \sigma$$

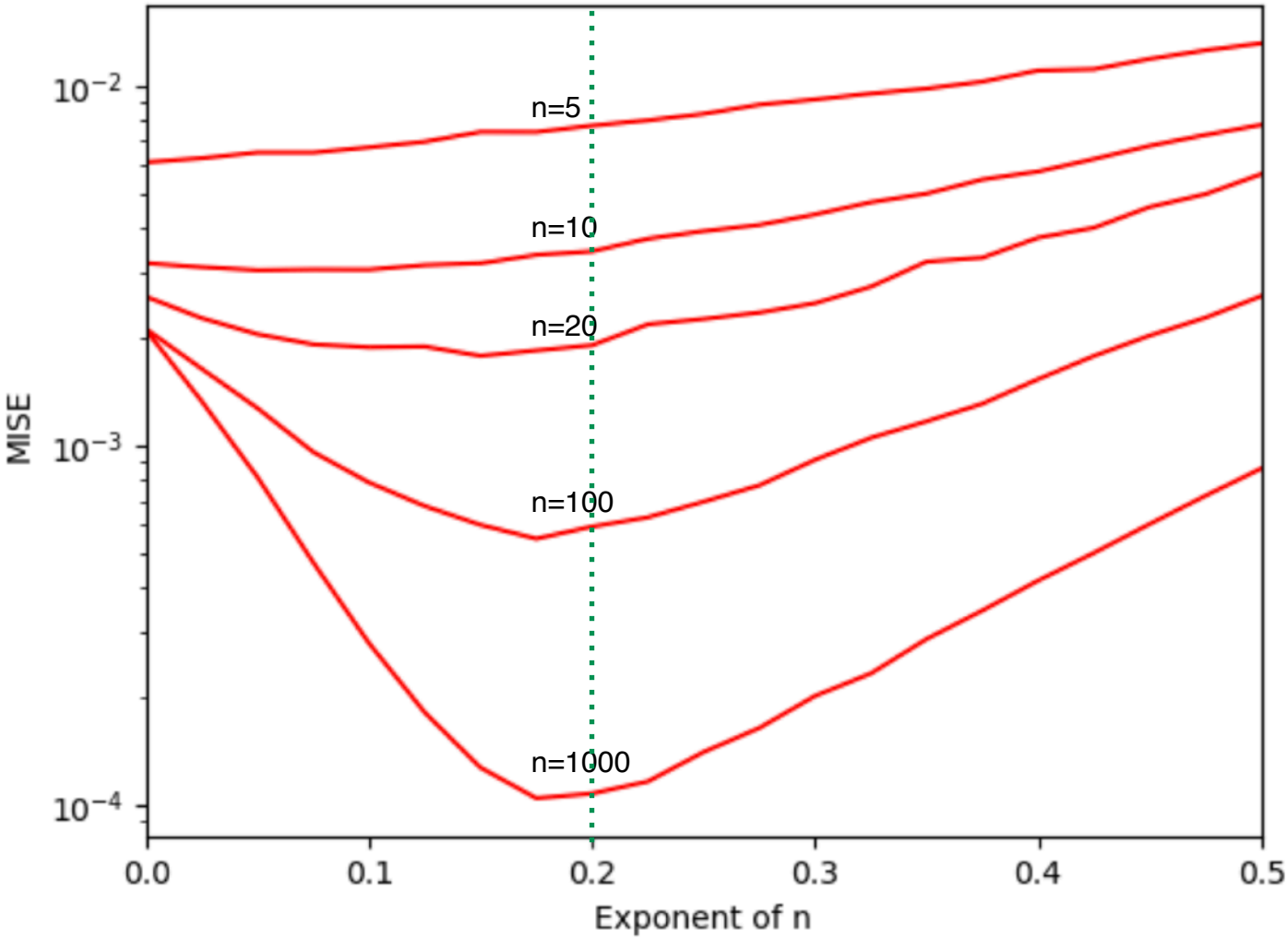
"Silverman's Rule of Thumb"

For a d-dimensional kernel, this becomes:

$$h_{min}^{(d)} \sim n^{-1/(4+d)}\, \sigma$$

(not so surprising, since the 1st term of the AMISE will now have a factor of $1/h^d$)

“Brute Force” Direct MC calculation of MISE vs α (assuming $h = \sigma n^{-\alpha}$) for Gaussian kernels approximating a 1D Gaussian



Cross-Validation

(Another approximation approach for bandwidth selection)

$$ISE = \int \left(\hat{f}(x \mid h) - f(x) \right)^2 \, dx$$

Assume the measured data representatively samples the distribution, so MISE ~ ISE

$$= \underbrace{\int \hat{f}^2(x \mid h) \, dx - 2 \int \hat{f}(x \mid h) f(x) \, dx}_{\text{want to minimise this quantity with respect to h}} + \underbrace{\int f^2(x) \, dx}_{\text{doesn't depend on h}}$$

Least Square Cross-Validation:

$$LSCV \equiv \int \hat{f}^2(x \mid h) \, dx - \frac{2}{n} \sum_{i=1}^{n} \hat{f}_{-i}(X_i \mid h)$$

Again, assuming the measured data representatively samples $f(x)$

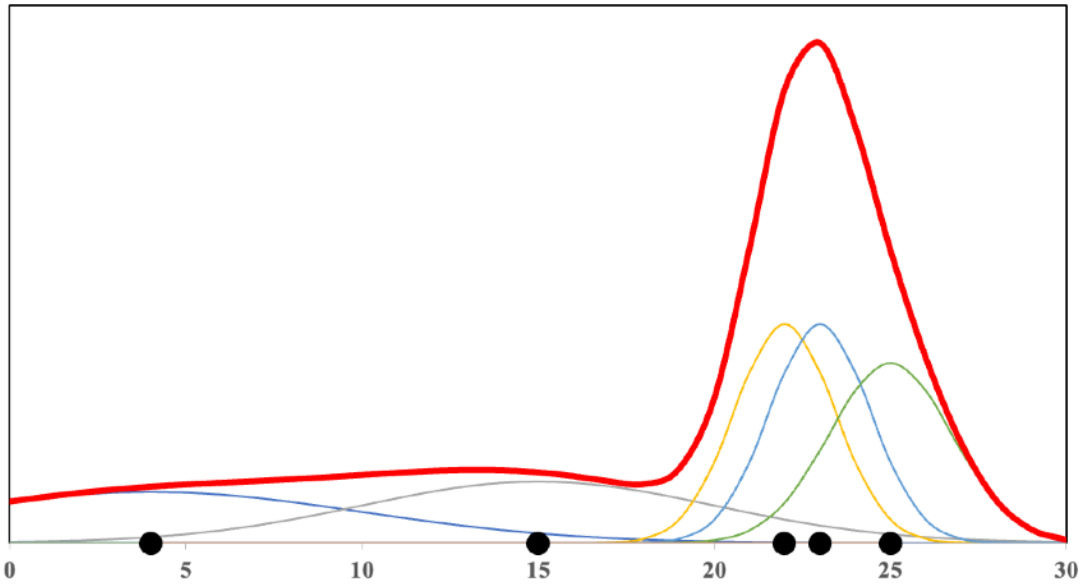
where $$\hat{f}_{-i}(X_i \mid h) \equiv \frac{1}{n-1} \sum_{j \neq i}^{n} K_h(x - X_j)$$

i^{th} contribution is removed to keep the average unbiased

Then choose the value of h that numerically minimises the LSCV

Adaptive KDE

Rather than just use a fixed bandwidth everywhere, we could aim to improve our estimate by increasing the bandwidth in regions of lower density and decrease it in regions of higher density (i.e. make the bandwidth inversely proportional to the density).



This can be done as an additional iteration on our best estimate for the density from our fixed-bandwidth KDE.

Silverman's prescription*:

1) Find a *pilot estimate* for the density, $\tilde{f}(\mathbf{x})$

(for example, using a fixed-bandwidth method)

2) Define *local bandwidth* factors: $\lambda_i \equiv \left(\frac{g}{\tilde{f}(\mathbf{x_i})} \right)^{\alpha}$

"sensitivity" parameter

relative to the geometric mean density

$$g \equiv \left(\prod_i^n \tilde{f}(x_i) \right)^{1/n}$$

Abramson** showed that a choice of α=1/2 causes the second derivative term of the AMISE to vanish, thus insuring a better estimate than the fixed-bandwidth case

3) Define the *adaptive kernel estimate*:

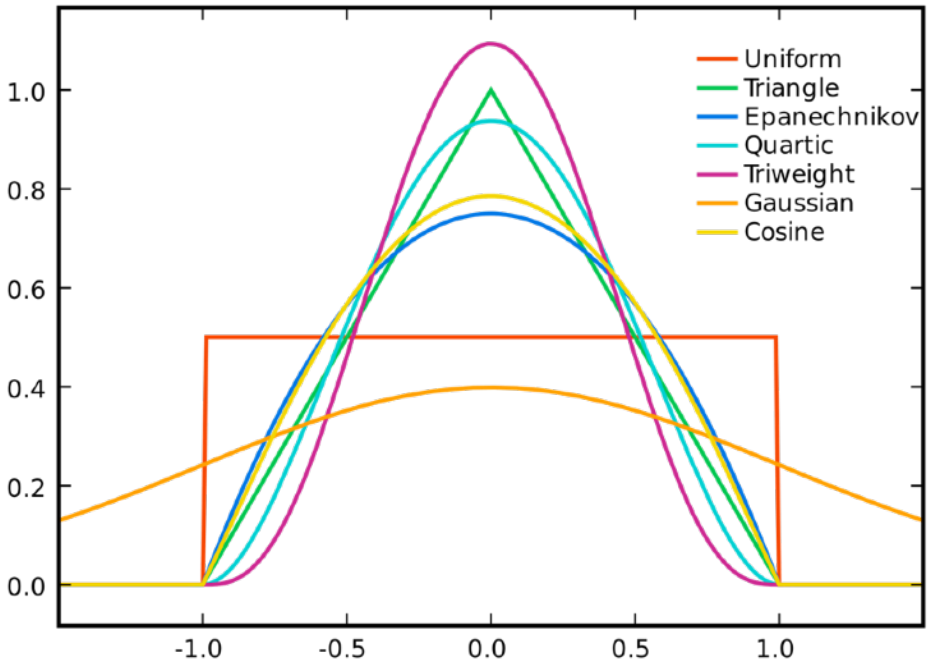
$$\hat{f}(x) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{(h\lambda_i)^d} K\left(\frac{x - X_i}{h\lambda_i} \right)$$

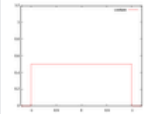
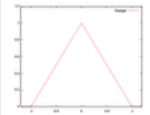
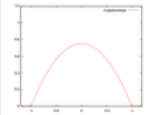
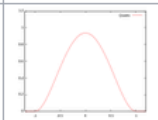
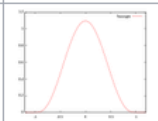
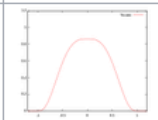
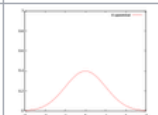
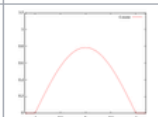
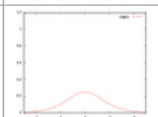
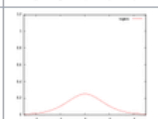
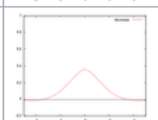
* Silverman, "Density Estimation for Statistics and Data Analysis," 1986

** Abramson, The Annals of Statistics, vol 10, no 4, 1217-1223, 1982

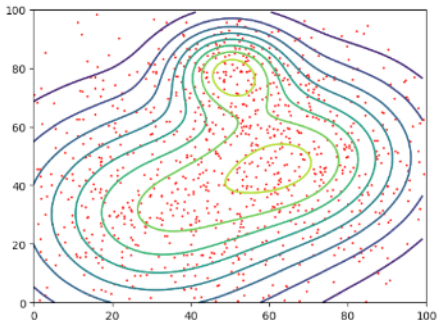
Some other common kernels:

(thanks Wikipedia!)



Kernel Functions, $K(u)$			$\int u^2 K(u)du$	$\int K(u)^2 du$
Uniform ("rectangular window")	$K(u) = \frac{1}{2}$ Support: $\lvert u\rvert \le 1$	 "Boxcar function"	$\frac{1}{3}$	$\frac{1}{2}$
Triangular	$K(u) = (1 - \lvert u\rvert)$ Support: $\lvert u\rvert \le 1$		$\frac{1}{6}$	$\frac{2}{3}$
Epanechnikov (parabolic)	$K(u) = \frac{3}{4}(1 - u^2)$ Support: $\lvert u\rvert \le 1$		$\frac{1}{5}$	$\frac{3}{5}$
Quartic (biweight)	$K(u) = \frac{15}{16}(1 - u^2)^2$ Support: $\lvert u\rvert \le 1$		$\frac{1}{7}$	$\frac{5}{7}$
Triweight	$K(u) = \frac{35}{32}(1 - u^2)^3$ Support: $\lvert u\rvert \le 1$		$\frac{1}{9}$	$\frac{350}{429}$
Tricube	$K(u) = \frac{70}{81}(1 - \lvert u\rvert^3)^3$ Support: $\lvert u\rvert \le 1$		$\frac{35}{243}$	$\frac{175}{247}$
Gaussian	$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$		1	$\frac{1}{2\sqrt{\pi}}$
Cosine	$K(u) = \frac{\pi}{4}\cos\left(\frac{\pi}{2}u\right)$ Support: $\lvert u\rvert \le 1$		$1 - \frac{8}{\pi^2}$	$\frac{\pi^2}{16}$
Logistic	$K(u) = \frac{1}{e^u + 2 + e^{-u}}$		$\frac{\pi^2}{3}$	$\frac{1}{6}$
Sigmoid function	$K(u) = \frac{2}{\pi}\frac{1}{e^u + e^{-u}}$		$\frac{\pi^2}{4}$	$\frac{2}{\pi^2}$
Silverman kernel[6]	$K(u) = \frac{1}{2}e^{-\frac{\lvert u\rvert}{\sqrt{2}}} \cdot \sin\left(\frac{\lvert u\rvert}{\sqrt{2}} + \frac{\pi}{4}\right)$		0	$\frac{3\sqrt{2}}{16}$

A Simple 2D Example



Fixed Bandwidth

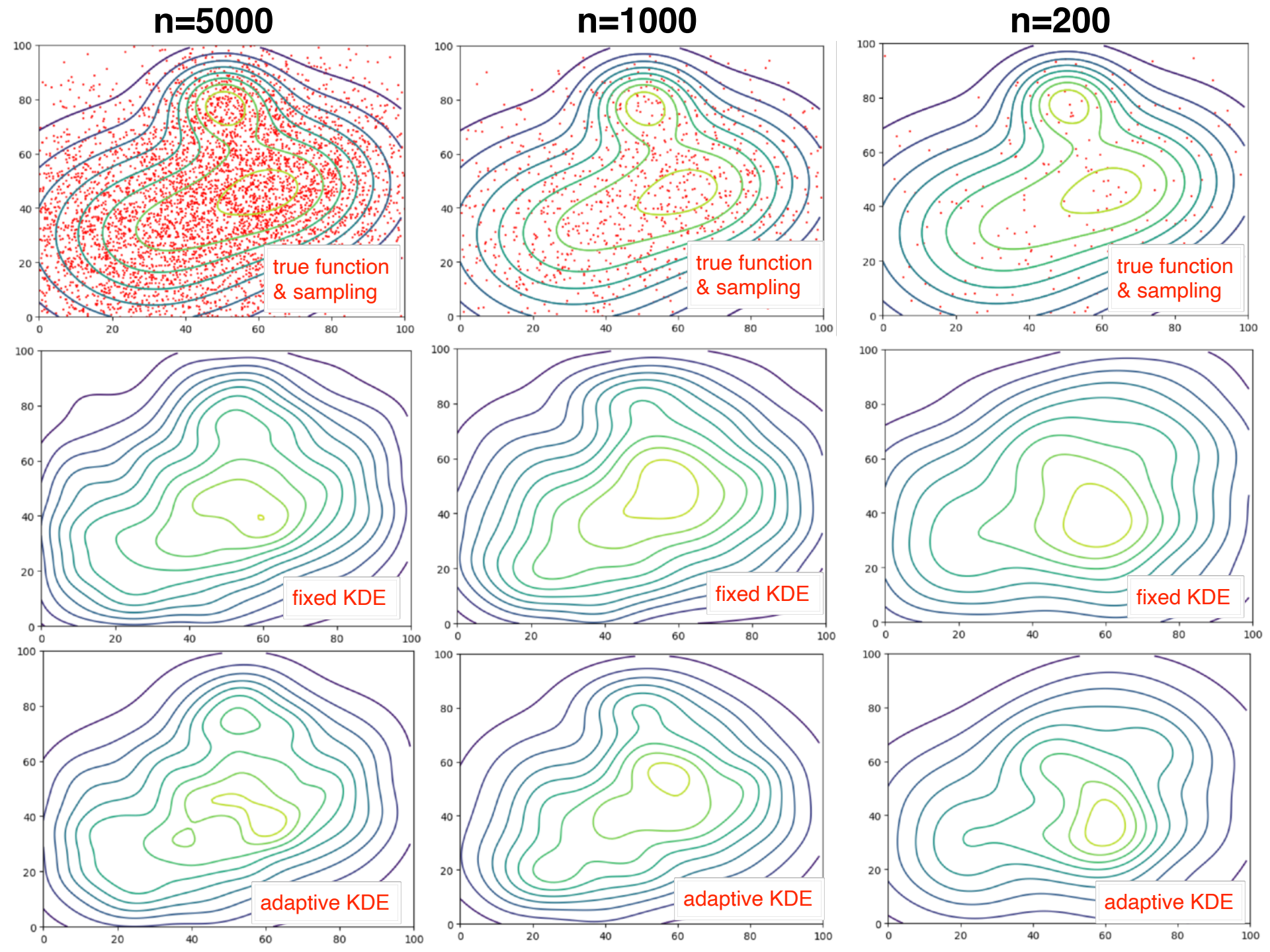
```
hx=np.sqrt(np.var(x))*(ndata**(-1/6)) # Silverman 2D bandwidth
hy=np.sqrt(np.var(y))*(ndata**(-1/6))

f=F*0
for i in range(100):       # Loop over grid points
    if((i+1)/10 == int((i+1)/10)): print('i=',i+1) # Report progress
    for j in range(100):
        for k in range(ndata):  # Sum KDE contributions from each data point
            f[i][j]=f[i][j]+np.exp(-((Xg[i,0]-x[k])**2)/(2*hx*hx)
                                   -((Yg[0,j]-y[k])**2)/(2*hy*hy))
        f[i][j]=(1/ndata)*(1/(2*math.pi*hx*hy))*f[i][j]
plt.xlim(0,100)
plt.ylim(0,100)
plt.contour(Xg,Yg,f,10)
plt.show()
```

$$\frac{1}{n_{data}}\frac{1}{2\pi h_x h_y}\sum_{k}^{n_{data}}\exp\left[-\frac{1}{2}\left(\frac{x_g(i)-x(k)}{h_x}\right)^2-\frac{1}{2}\left(\frac{y_g(j)-y(k)}{h_y}\right)^2\right]$$

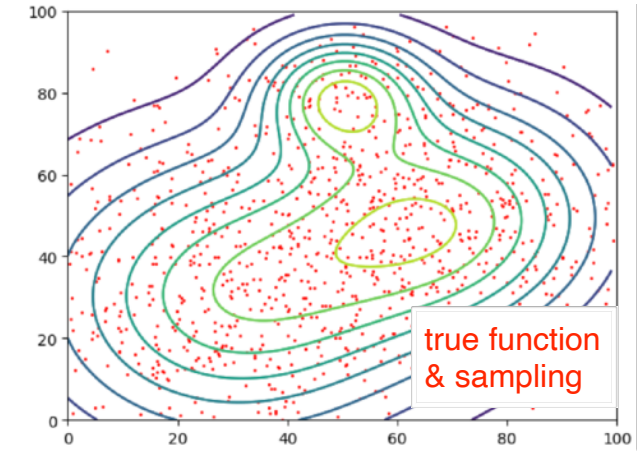
Adaptive Bandwidth

```
g=1
for k in range(ndata):
    i=int(x[k]/delta)
    j=int(y[k]/delta)
    g=g*(f[i][j]**(1/ndata))    # Compute geometric means

lam=[0]*ndata
for k in range(ndata):
    i=int(x[k]/delta)
    j=int(y[k]/delta)
    lam[k]=np.sqrt(g/f[i][j])   # Compute local bandwidth factors

fa=F*0
for i in range(100):       # Loop over grid points
    if((i+1)/10 == int((i+1)/10)): print('i=',i+1) # Report progress
    for j in range(100):
        for k in range(ndata):  # Sum KDE contributions from each data point
            fa[i][j]=fa[i][j]+(1/(lam[k]**2))*np.exp(-((Xg[i,0]-x[k])**2)/(2*(hx*lam[k])**2)
                                                     -((Yg[0,j]-y[k])**2)/(2*(hy*lam[k])**2))
        fa[i][j]=(1/ndata)*(1/(2*math.pi*hx*hy))*fa[i][j]

plt.xlim(0,100)
plt.ylim(0,100)
plt.contour(Xg,Yg,fa,10)
plt.show()
```

$$\frac{1}{n_{data}}\frac{1}{2\pi h_x h_y}\sum_{k}^{n_{data}}\frac{1}{\lambda(k)^2}\exp\left[-\frac{1}{2}\left(\frac{x_g(i)-x(k)}{\lambda(k)*h_x}\right)^2-\frac{1}{2}\left(\frac{y_g(j)-y(k)}{\lambda(k)*h_y}\right)^2\right]$$

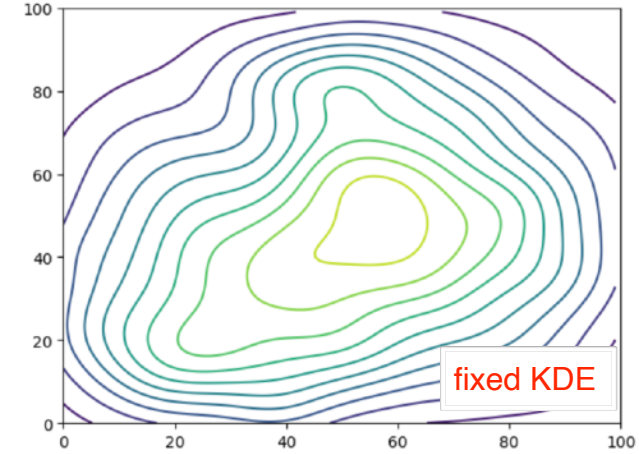

n=5000
n=1000
n=200
true function & sampling
true function & sampling
true function & sampling
fixed KDE
fixed KDE
fixed KDE
adaptive KDE
adaptive KDE
adaptive KDE

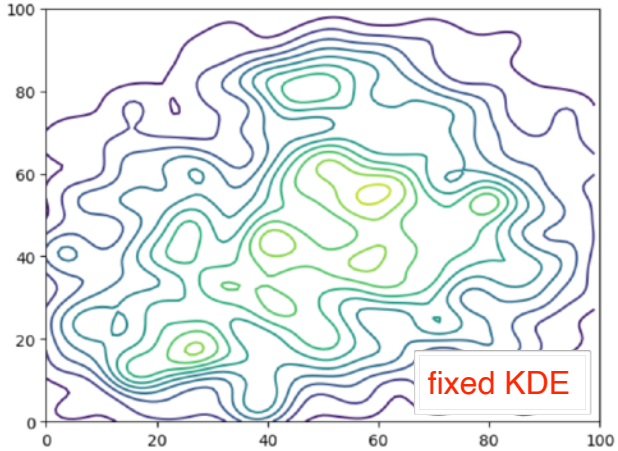
n=1000

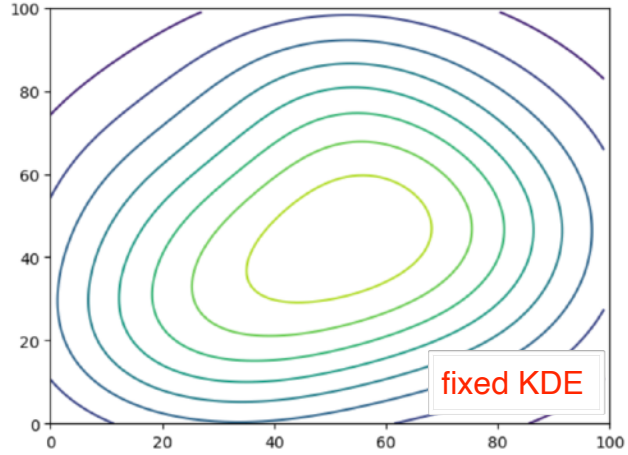


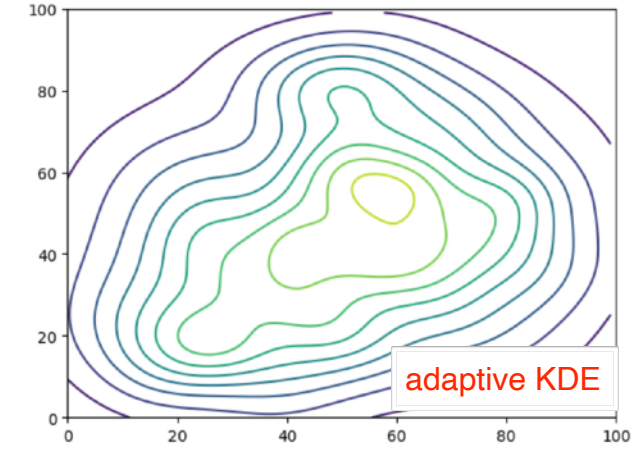
using half the nominal choice of scale

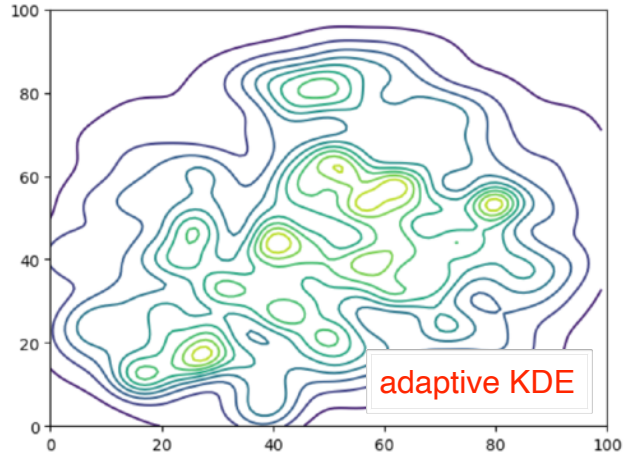
using twice the nominal choice of scale

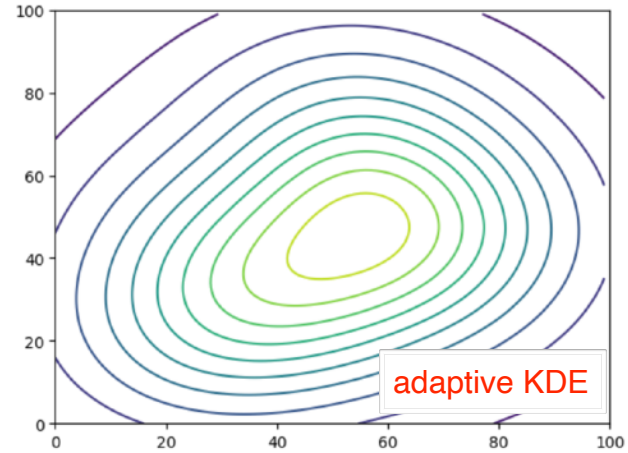












ANY smoothing process is necessarily a fabrication!

One way or another, you are guessing at the form in order to make inferences about data you do not actually have.

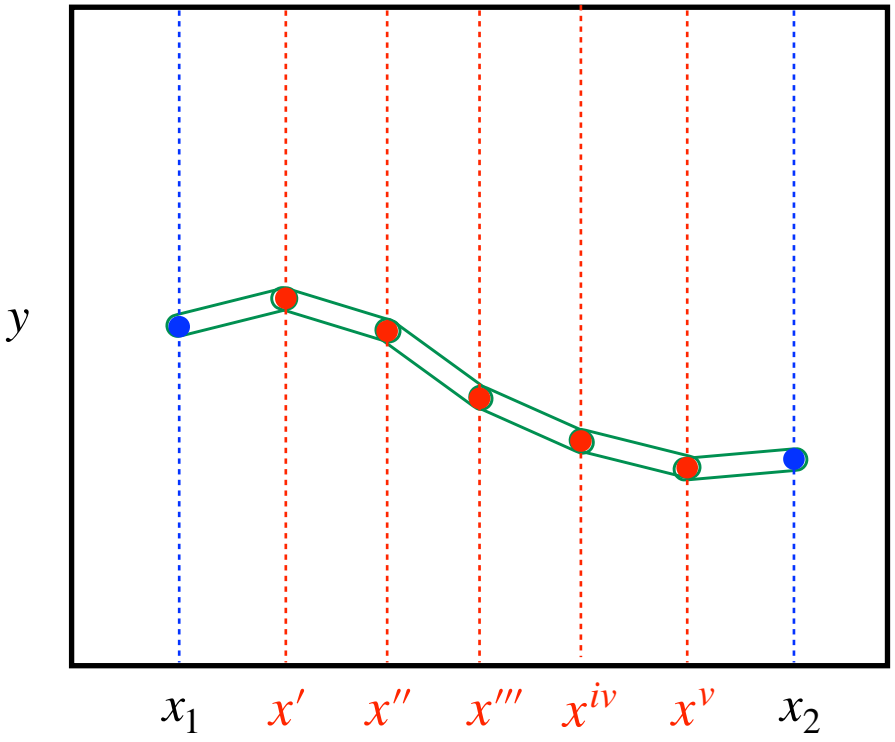
While this method is non-parametric, you are still making assumptions concerning the nature of continuity between data points. Although this generally works well for functions that are continuous and well-behaved, you can run into problems near boundaries or for functions that are discontinuous, discreet or rapidly changing.

You don't get error bars on the model, so takes some work to insure result is robust to smoothing method

It's always important to specifically test the sensitivity of your conclusions to the particular smoothing technique!

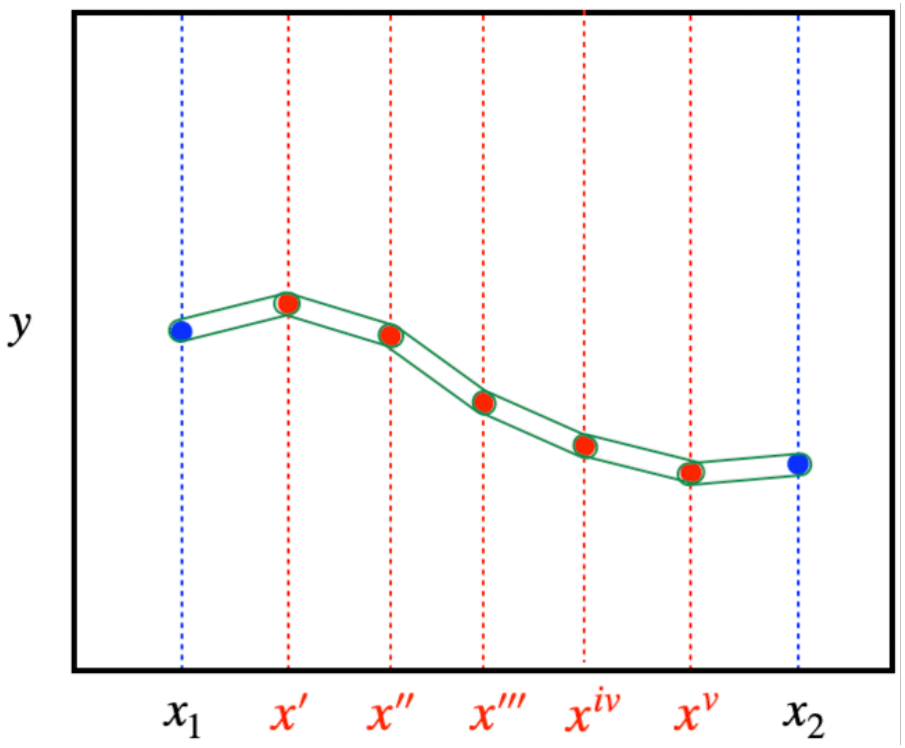
Gaussian Processes for Non-Parametric Regression

Given measurements of some function $y(x)$ at positions of x_1 and x_2, we would like to infer the likely function values at other positions, and the uncertainties of such projections:



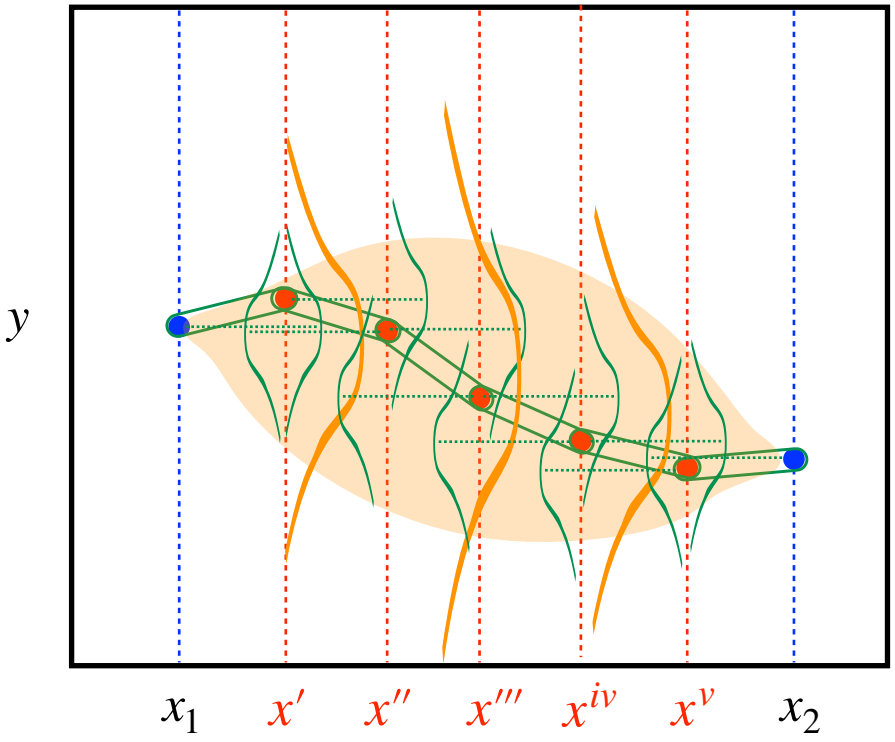
Assume $y(x)$ is continuous, which means that nearby values of x are likely to have nearby values of $y(x)$. Imagine imposing continuity by "attaching elastic bands" between the points, with the amount of elasticity specifying the assumed degree of correlation

The elasticity then constrains the range of possible solutions, with the largest freedom of movement seen at positions that are furthest from measured values



The elasticity then constrains the range of possible solutions, with the largest freedom of movement seen at positions that are furthest from measured values

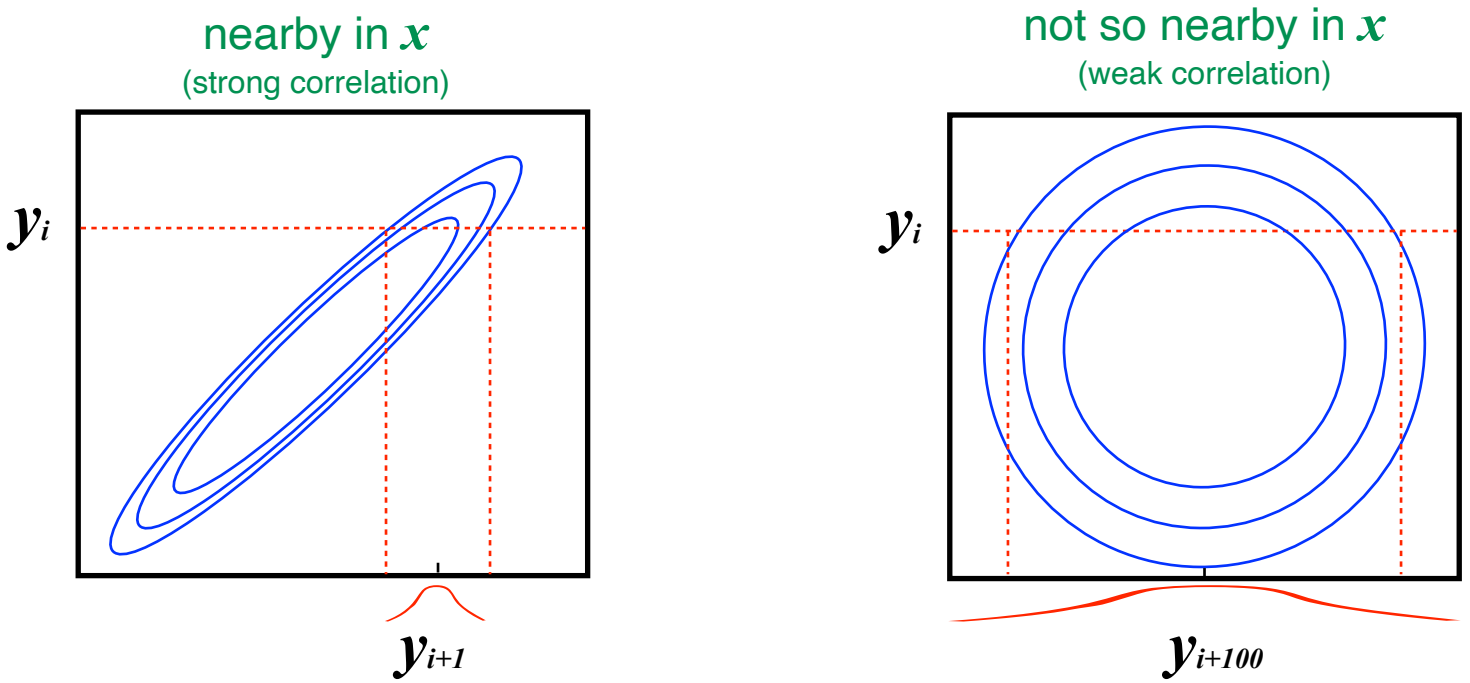
Let's characterise the elasticity by a Gaussian sampling of $y(x)$ about the mean of the values at nearby x positions:



A nice property of Gaussian processes is that the cumulative effect of multiple samplings is also a Gaussian, with a variance that can be analytically computed. So, you can calculate error bands for the inferred y values!

For simplicity, this is just illustrating nearest neighbour constraints. But, in general, we want the y values at each position to influence the likely values at any other position. This sounds hideously complicated, but is trivially handled by a covariance matrix (which is what it's there for!)

Another view: plotting Gaussian covariances between two different y values:



$P(y_{i+1} \mid y_i)$ $P(y_{i+100} \mid y_i)$

1) Conditional probabilities are also Gaussian!

2) Projected y values largely depend on the covariances, as opposed to the distribution means

(in the limit of perfect covariance, the correlation is a line, and the "average position of a line" is now meaningless!)

So, we want to specify a covariance matrix that allows us to tune the elasticity, with a correlation that decreases as the separation between x values increases.

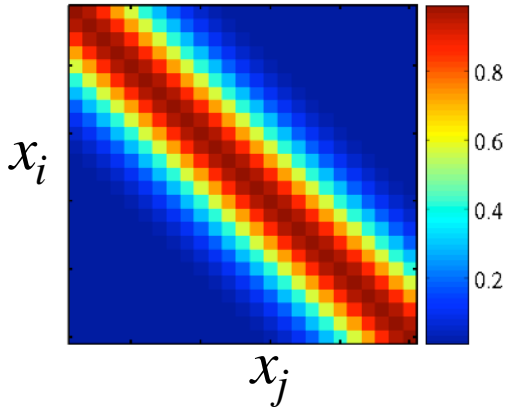
We can do this in a continuous way by defining a "kernel" (weighting function), such as the following:

$$k_y(x_i, x_j) = \sigma_y^2 \exp\left(-\frac{1}{2l^2}(x_i - x_j)^2\right) + \sigma_n^2 \delta_{ij}$$

maximum inherent variance scale

correlation length scale

uncorrelated measurement noise



This also happens to have a Gaussian form, but doesn't need to be… it is just a parameterisation defining the scale of the (Gaussian) variance. *Other kernels are available at your local retailer to suit your individual needs!*

We then just need to find the best values for the "hyperparameters" σ_y and l

…which we can do using maximum likelihood!

("training set")

Define $\boldsymbol{y_m}$ as a k-length vector of measured y values, and $\boldsymbol{y_p}$ as the vector of y values to be predicted. Then,

$$P(\mathbf{y_p} \,|\, \mathbf{y_m}) = \frac{P(\mathbf{y_p}, \mathbf{y_m})}{P(\mathbf{y_m})}$$

But the ratio of Gaussians is also a Gaussian!

$$P(\mathbf{y_m}) = \mathcal{N}\left(\mu_\mathbf{m}\ ,\ \boldsymbol{\Sigma}_\mathbf{m}\right) \equiv \frac{1}{\sqrt{(2\pi)^k\ \det(\boldsymbol{\Sigma}_\mathbf{m})}} \exp\left[-\frac{1}{2}(\mathbf{y_m} - \mu_\mathbf{m})\boldsymbol{\Sigma}_\mathbf{m}^{-1}(\mathbf{y_m} - \mu_\mathbf{m})\right]$$

(multi-variate Gaussian)

$$P(\mathbf{y_p}, \mathbf{y_m}) = \mathcal{N}\left(\begin{bmatrix}\mu_\mathbf{p} \\ \mu_\mathbf{m}\end{bmatrix}\ ,\ \begin{bmatrix}\boldsymbol{\Sigma}_\mathbf{p} & \boldsymbol{\Sigma}_\mathbf{pm} \\ \boldsymbol{\Sigma}_\mathbf{pm}^\mathbf{T} & \boldsymbol{\Sigma}_\mathbf{m}\end{bmatrix}\right)$$

all specified by kernel

linear relationship with measurement — uncertainty reduced by measurement

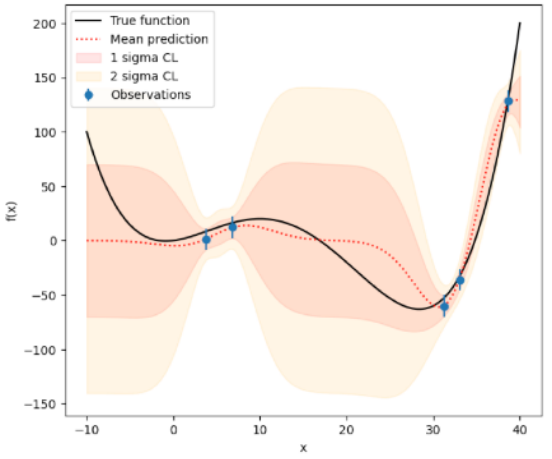
$$P(\mathbf{y_p} \,|\, \mathbf{y_m}) = \mathcal{N}\left(\mu_\mathbf{p} + \boldsymbol{\Sigma}_\mathbf{pm}\boldsymbol{\Sigma}_\mathbf{mm}^{-1}(\mathbf{y_m} - \mu_\mathbf{m})\ ,\ \boldsymbol{\Sigma}_\mathbf{p} - \boldsymbol{\Sigma}_\mathbf{pm}\boldsymbol{\Sigma}_\mathbf{mm}^{-1}\boldsymbol{\Sigma}_\mathbf{pm}^\mathbf{T}\right)$$

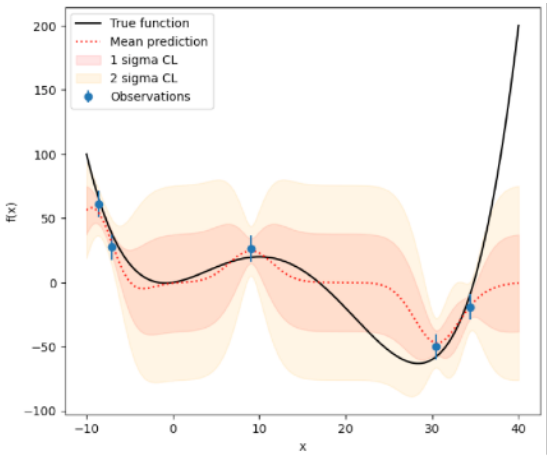
$$\sim \mathcal{N}\left(\boldsymbol{\Sigma}_\mathbf{pm}\boldsymbol{\Sigma}_\mathbf{mm}^{-1}\mathbf{y_m}\ ,\ \boldsymbol{\Sigma}_\mathbf{p} - \boldsymbol{\Sigma}_\mathbf{pm}\boldsymbol{\Sigma}_\mathbf{mm}^{-1}\boldsymbol{\Sigma}_\mathbf{pm}^\mathbf{T}\right)$$

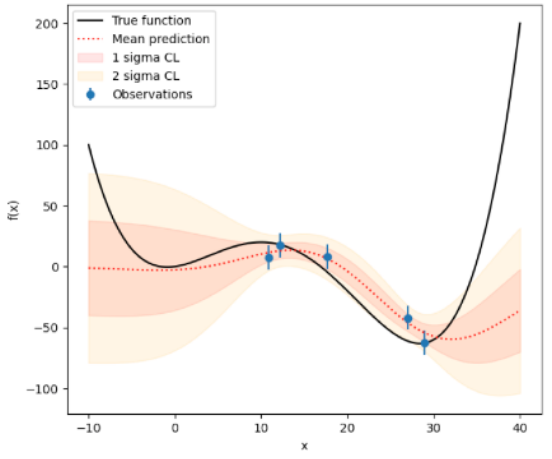
predictive mean — predictive variance

Simple Implementation with sklearn

```python
import numpy as np
import sklearn
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

seed=np.random.seed(1234562)

nsamp=5   # Nunber of samples
yerr=10.0 # Size of rms error

# Define true function to be sampled
# Note: X needs to be 2D for GP processor, so reshape as many rows as needed in 1 column
X = np.linspace(start=-10, stop=40, num=1000).reshape(-1, 1)
y = np.squeeze(0.001*X**4 - 0.05*X**3 + 0.5*X**2 + X) # Define function from X and then make 1D

# Randomly sample measurement points
X_train=[0]*nsamp
y_train=[0]*nsamp
yerr_train=[yerr]*nsamp
ybest=10000.0
for i in range(nsamp):
    isamp=np.random.randint(0,1000)
    X_train[i]=X[isamp]
    y_train[i]=np.random.normal(loc=y[isamp], scale=yerr) # Add Gaussian fluctuation to measurements
    if(y_train[i]<ybest):  # Keep track of the sampled point with the 'best guess' minimum value
        ybest=y_train[i]
        ibest=i

# Invoke Gaussian process regression
var=np.array([yerr_train[i]**2 for i in range(nsamp)])  # Array of sample variances (added to kernel as 'alpha')
kernel = 1.0*RBF(length_scale=1.0, length_scale_bounds=(0.001, 1000)) # Use simple Gaussian kernel
gaussian_process = GaussianProcessRegressor(kernel=kernel, alpha=var, n_restarts_optimizer=100)
gaussian_process.fit(X_train, y_train)

# Extract predictions
mean, rms = gaussian_process.predict(X, return_std=True)
```

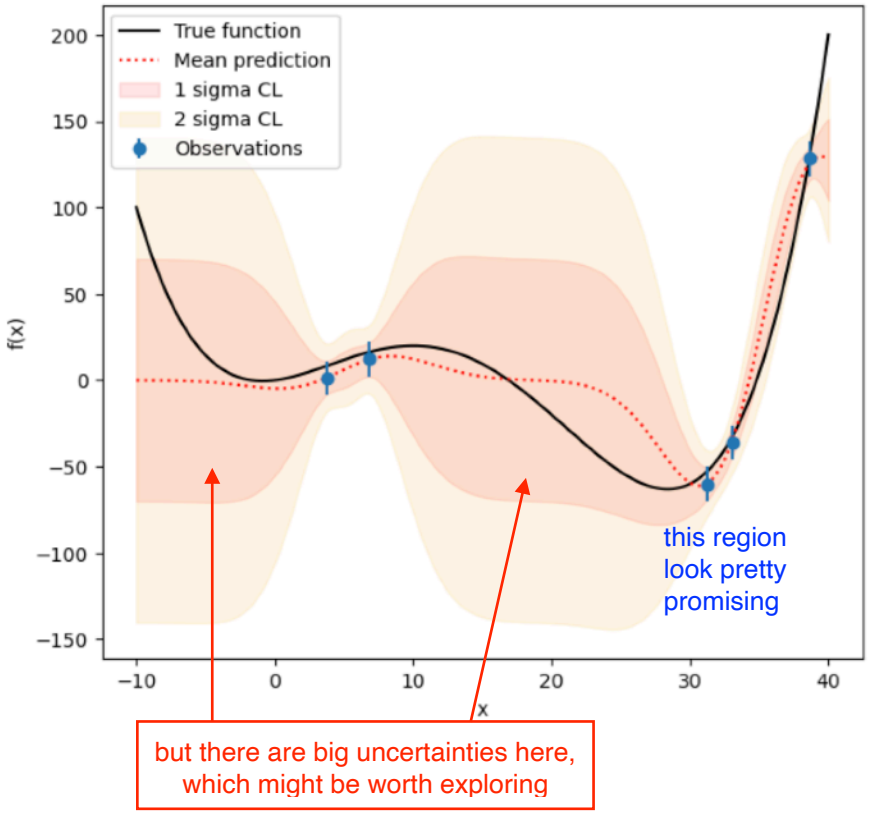






Bayesian Search Using Gaussian Processes

Say you're interested in finding the minimum of a function using information from your GP model. In particular, you want to know what value of x to test next:



"Exploration vs Exploitation"

We want to balance these by defining an "acquisition function" to tell us where to search next. One useful acquisition function is based on the expected improvement to our current best value $y(x^*)$

We can define the improvement from ***y(x*)*** to a new value, ***y(x)*** as follows:

$$I(x) \equiv \max[y(x^*) - y(x)\ ,\ 0]$$

But recall that we now have a Gaussian probability distribution associated with each possible value of $\boldsymbol{x}$, so what we really want is the expectation value of the improvement at each value of $\boldsymbol{x}$

from evaluating the predictive mean and variance at the specific location $\boldsymbol{x}$

$$\langle I(x)\rangle = \int_{-\infty}^{\infty} I(x)\ \mathcal{N}\left(\mu(x), \sigma^2(x)\right) dy(x)$$

$$= \int_{-\infty}^{y(x^*)} \left(y(x^*) - y(x)\right)\ \frac{1}{\sqrt{2\pi}\sigma(x)} \exp\left(-\frac{1}{2}\left(\frac{y(x) - \mu(x)}{\sigma(x)}\right)^2\right) dy(x)$$

$$z \equiv \frac{y(x) - \mu(x)}{\sigma(x)}$$

$$dz = \frac{dy(x)}{\sigma(x)}$$

$$= \int_{-\infty}^{z_0} \left(y(x^*) - \mu(x) - z\sigma(x)\right)\ \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}z^2\right) dz$$

$$z_0 \equiv \frac{y(x^*) - \mu(x)}{\sigma(x)}$$

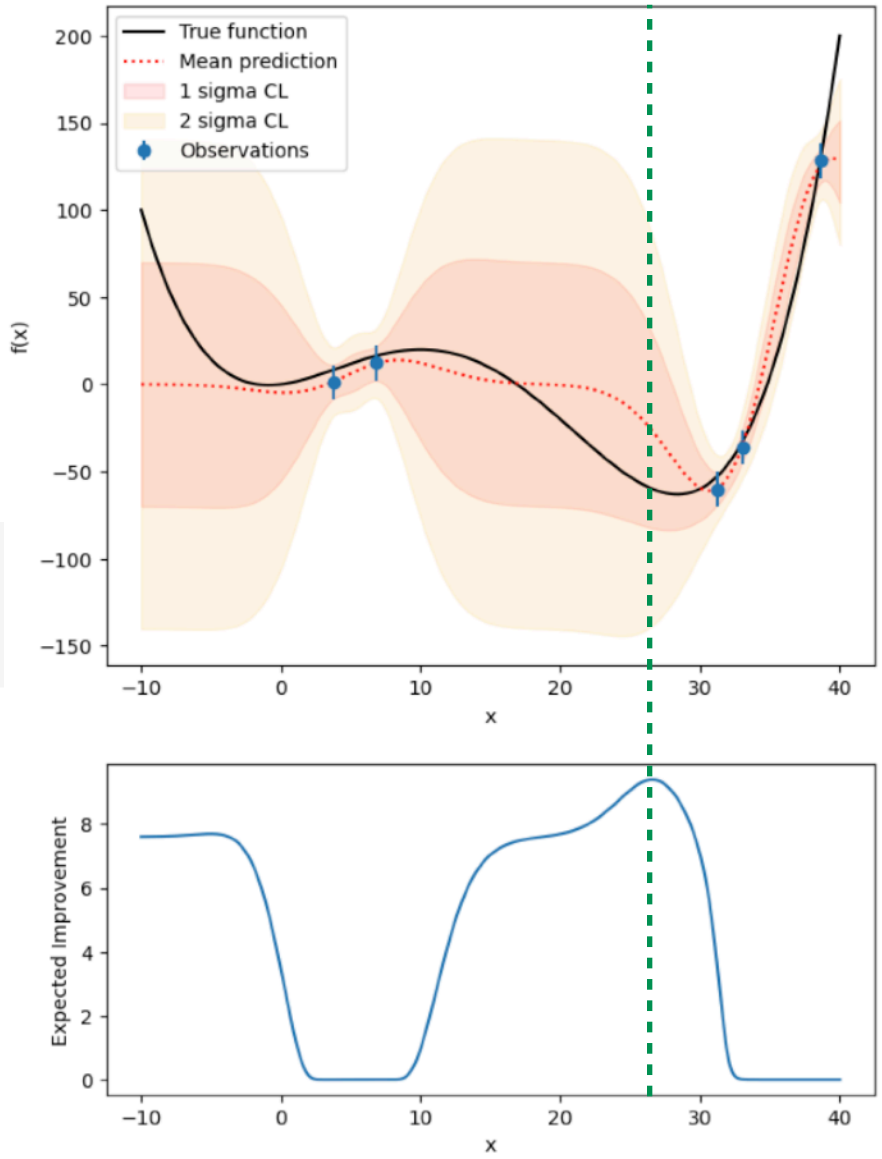
$$= \left(y(x^*) - \mu(x)\right)\ \mathcal{N}_C\left(< z_0 | 0,1\right)\ +\ \sigma(x)\ \mathcal{N}\left(z_0 | 0,1\right)$$

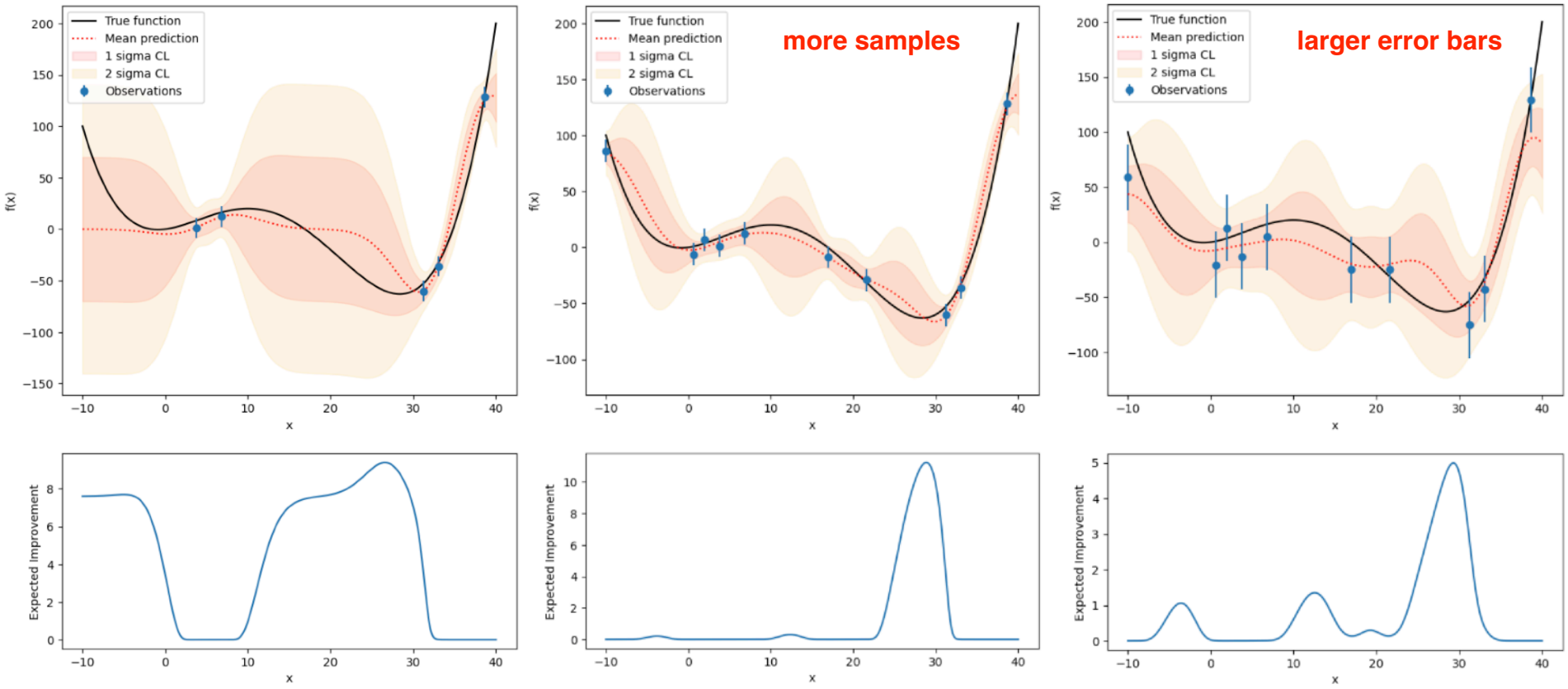
Cumulative (or integral) of Normal distribution with μ=0 and σ=1 integrated for values less than z_0

Normal distribution with μ=0 and σ=1 evaluated at z_0

Choose values of x where this is maximum

```
# Calculate expected improvement for different possible positions to sample from next
IM=[0]*1000
from scipy.stats import norm
for i in range(1000):
    z=(y_train[ibest]-mean[i])/rms[i]
    IM[i]=(y_train[ibest]-mean[i])*norm.cdf(z) + rms[i]*norm.pdf(z)
```




True function
Mean prediction
1 sigma CL
2 sigma CL
Observations
f(x)
x
Expected Improvement
more samples
larger error bars