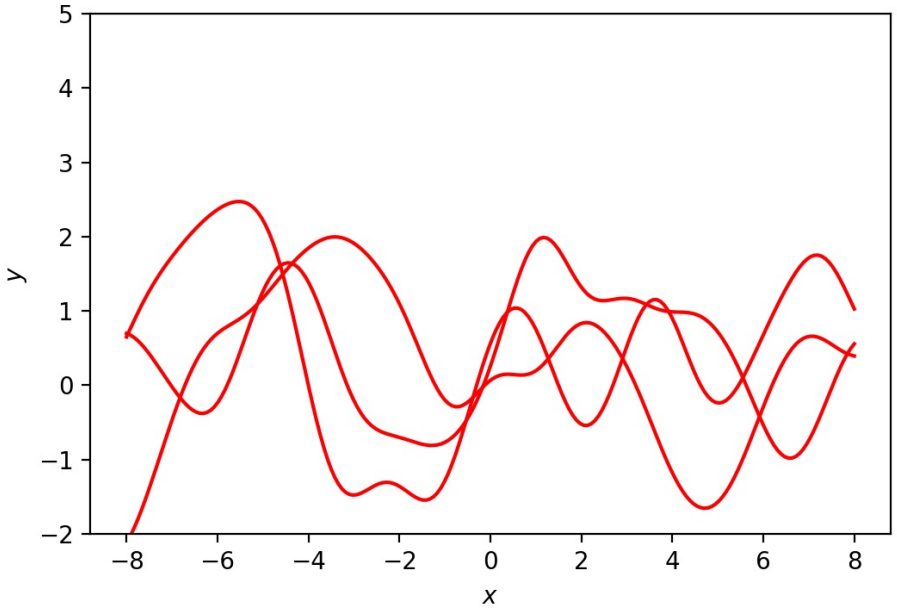
COMP 432 Machine Learning

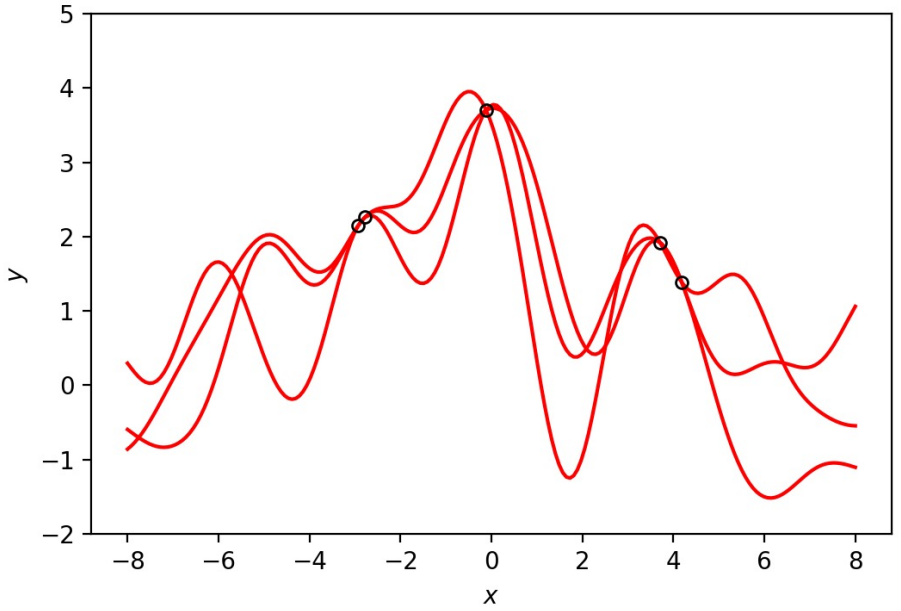
Gaussian Processes

Computer Science & Software Engineering
Concordia University, Fall 2021

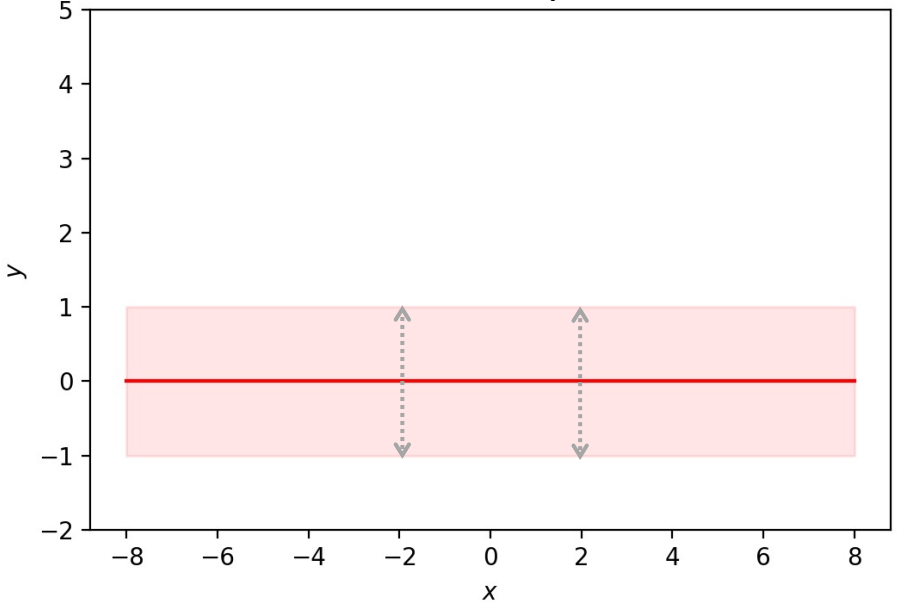
3 functions sampled from a GP



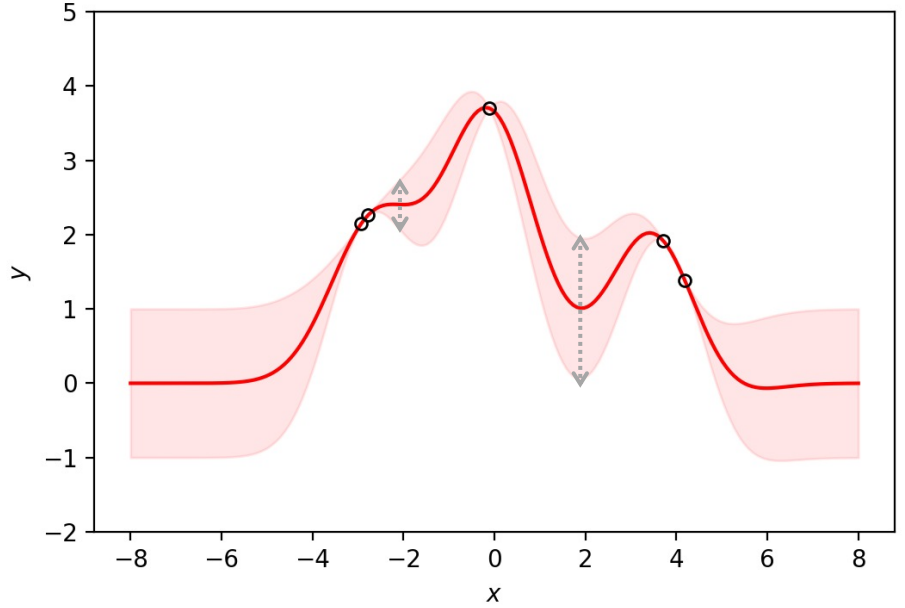
3 functions sampled from a GP
after conditioning on 5 training points



means $\mathbb{E}[\hat{y}(\mathbf{x})]$ & standard deviations $\mathrm{stdev}[\hat{y}(\mathbf{x})]$
of the functions sampled from a GP



means and standard deviations
after conditioning on 5 training points

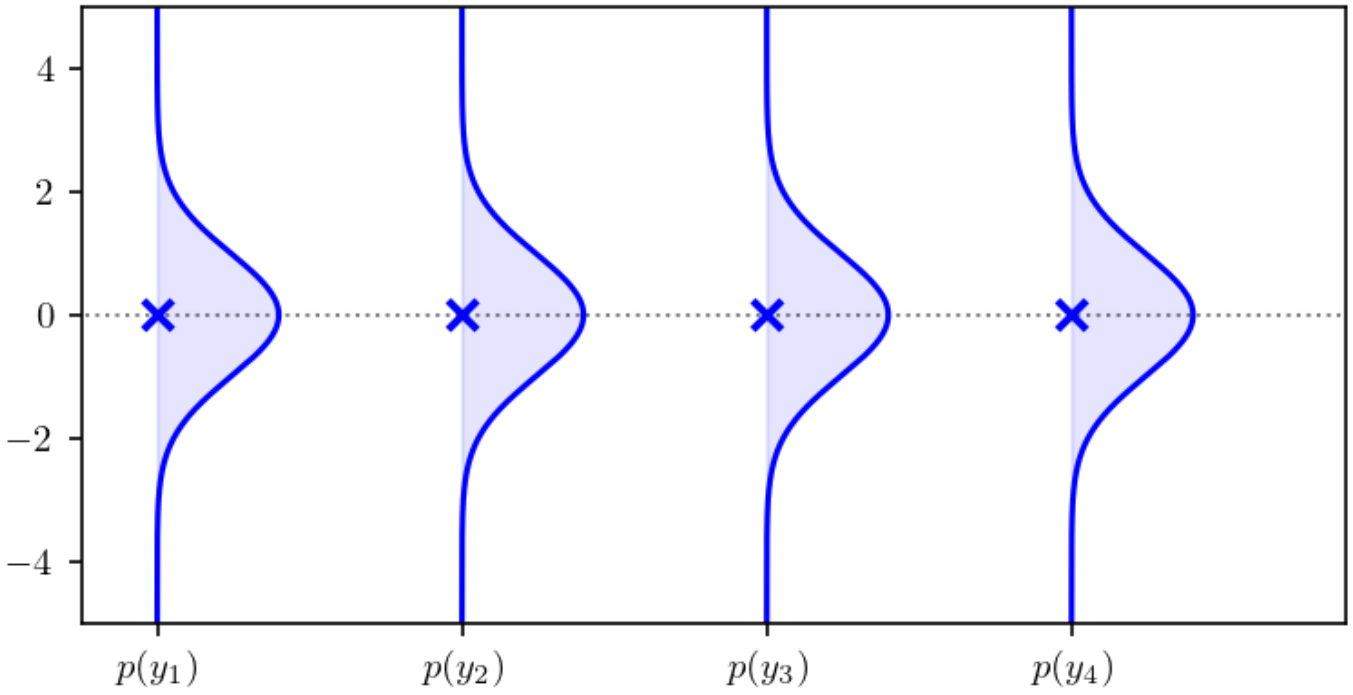


Gaussian Processes

- A *Gaussian process* (GP) is a probability distribution over *functions,* and has useful features:
  - Given a GP over domain $\mathbf{x} \in \mathbb{R}^D$, can sample $\hat{y} \sim \mathcal{GP}$ and evaluate $\hat{y}(\mathbf{x})$ over the entire domain
  - Or, can evaluate $\mathbb{E}[\hat{y}(\mathbf{x})]$ to get "average" at $\mathbf{x}$
  - Or, can evaluate $\mathrm{stddev}[\hat{y}(\mathbf{x})]$ to get "uncertainty" at $\mathbf{x}$
  - When we "fit" a GP, we are just *conditioning the GP distribution on the training data* $\{(\mathbf{x}_i, y_i)\}$*,* not learning any parameters (not "fitting" in traditional sense)
- *Non-parametric* model, so each training point $\mathbf{x}_i$ becomes part of the model
  - no parameters, only "hyperparameters" (kernel choice)
  - But in practice, kernel parameters *are* tuned to data

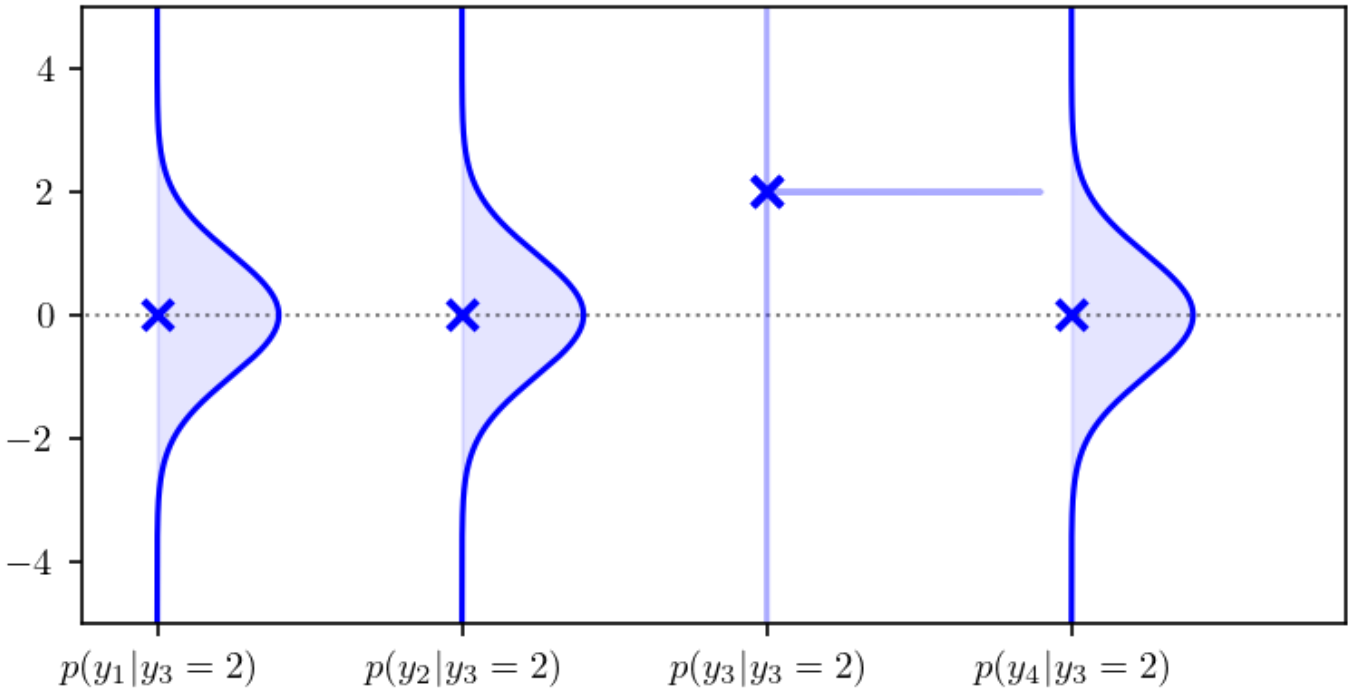
Preview of how GPs work:
Conditioning in high-dimensional Gaussians

Let $\boldsymbol{y} \in \mathbb{R}^4$ and $p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$ $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$



Preview of how GPs work:
Conditioning in high-dimensional Gaussians

Let $\boldsymbol{y} \in \mathbb{R}^4$ and $p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$ $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

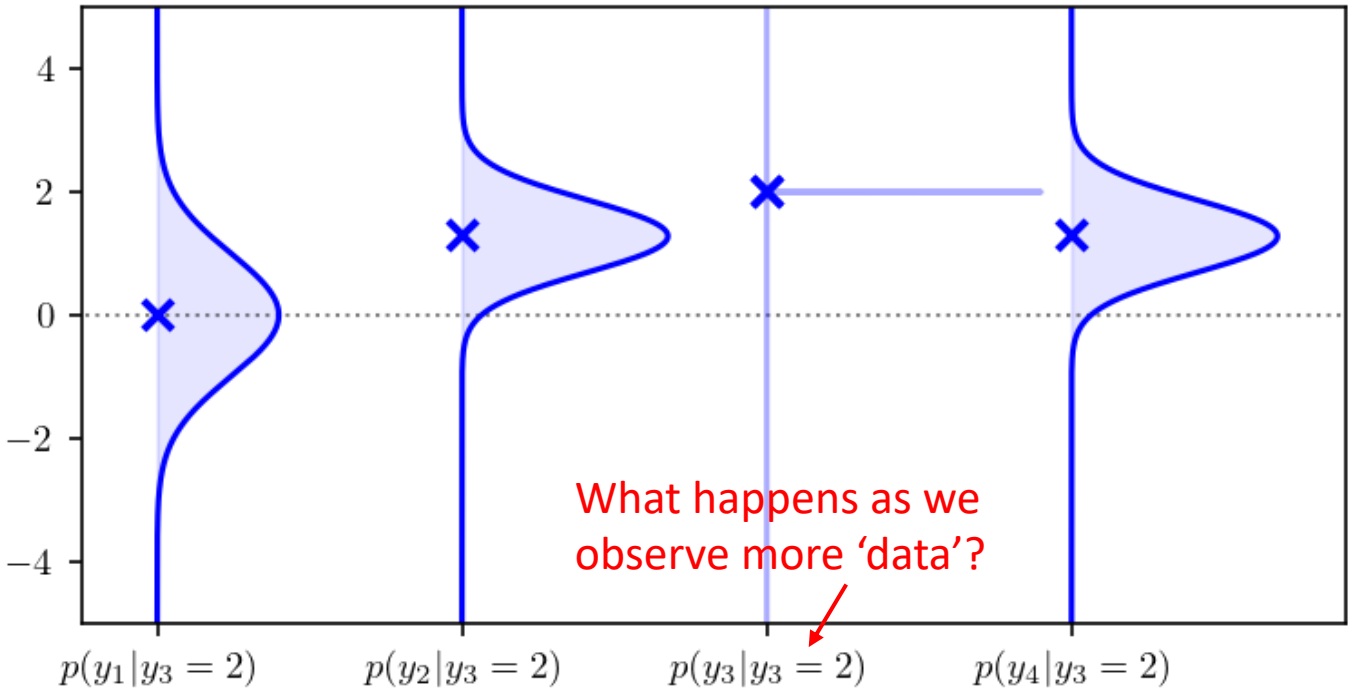


What happens as we add covariances farther off-diagonal?

Preview of how GPs work:
Conditioning in high-dimensional Gaussians

What happens as 4 approaches infinity?

Let $\boldsymbol{y} \in \mathbb{R}^4$ and $p(\boldsymbol{y}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$ where $\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$ $\boldsymbol{\mu} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ $\boldsymbol{\Sigma} = \begin{bmatrix} 1 & .8 & 0 & 0 \\ .8 & 1 & .8 & 0 \\ 0 & .8 & 1 & .8 \\ 0 & 0 & .8 & 1 \end{bmatrix}$



Gaussian Process key ideas

Idea 1. A function $f(\mathbf{x})$ is an infinite-dimensional vector, where $\mathbf{x}$ indexes the component

finite dimensional

$f_i \quad \mathbf{f} = [f_1, \ldots, f_N]$

$f : \{1, \ldots, N\} \to \mathbb{R}$

for each of these...

... look up one of these

infinite dimensional

$f(x)$

$f : \mathbb{R} \to \mathbb{R}$

for each of these...

... look up one of these

Our training data is finite-dimensional subset of the infinite-dimensional domain we want to predict on!

Gaussian Process key ideas

Idea 2. If we assume that two values $f(\mathbf{x})$ and $f(\mathbf{x}')$ are positively correlated whenever $\mathbf{x}$ and $\mathbf{x}'$ are "similar", then we are also assuming that $f(\mathbf{x})$ is "smooth" in a specific sense.

If we defined a prior over functions f by specifying only its mean $\mathbb{E}[f(\mathbf{x})]$ and covariance $\mathrm{Cov}[f(\mathbf{x}), f(\mathbf{x}')]$ at every point $\mathbf{x}$ and $\mathbf{x}'$, then the prior we have defined is a Gaussian process prior over functions.

Infinite-dimensional mean and covariance: $\mu_{\mathbf{x}}$ $\Sigma_{\mathbf{x},\mathbf{x}'}$

Finite-dimensional mean and covariance (familiar!): μ_i $\Sigma_{i,j}$

GPs from RLS prior (Bishop §6.4.1)

- Recall linear least squares regression with feature transformation:

$$\hat{y}(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x})$$

- Recall maximum a posteriori (MAP) learning and how we assumed a "small weight" prior:

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \tfrac{1}{\beta}\mathbf{I})$$

- Notice that defining $p(\mathbf{w})$ implicitly defines a probability distribution over functions $\hat{y}(\mathbf{x}, \mathbf{w})$
  - Functions $\hat{y}(\mathbf{x}, \mathbf{w})$ with small weights "more probable" under this prior on $p(\mathbf{w})$

Gaussian Process from RLS prior

- Suppose training set is $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ and we ignore targets for a moment, focusing on $\mathbf{x}_i$
- Let $\hat{\mathbf{y}}$ be vector of all predictions across training set

$$\hat{\mathbf{y}} = \mathbf{\Phi}\mathbf{w} = \begin{bmatrix} \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_1) \\ \vdots \\ \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}_N) \end{bmatrix}$$

- What can we say about probability distribution $p(\hat{\mathbf{y}})$ given our assumed prior $p(\mathbf{w})$?

GPs from RLS prior (Bishop §6.4.1)

- First, since $\hat{\mathbf{y}}$ is a linear transformation of a D-dimensional $\mathbf{w}$ and $p(\mathbf{w})$ is Gaussian, then $p(\hat{\mathbf{y}})$ must be an N-dimensional Gaussian.
  - But what are the $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ of this new Gaussian?

$$\mathbb{E}[\hat{\mathbf{y}}] = \mathbb{E}[\boldsymbol{\Phi}\mathbf{w}] = \boldsymbol{\Phi}\mathbb{E}[\mathbf{w}] = \boxed{\mathbf{0}}$$

Implicitly assume that mean of all predictions is 0

$$\begin{aligned}\mathrm{Cov}[\hat{\mathbf{y}}, \hat{\mathbf{y}}] &= \mathbb{E}[\hat{\mathbf{y}}\hat{\mathbf{y}}^T] - \mathbb{E}[\hat{\mathbf{y}}]\mathbb{E}[\hat{\mathbf{y}}]^T \\ &= \mathbb{E}[\boldsymbol{\Phi}\mathbf{w}\mathbf{w}^T\boldsymbol{\Phi}^T] - \mathbf{0} \\ &= \boldsymbol{\Phi}\mathbb{E}[\mathbf{w}\mathbf{w}^T]\boldsymbol{\Phi}^T = \boxed{\frac{1}{\beta}\boldsymbol{\Phi}\boldsymbol{\Phi}^T}\end{aligned}$$

inner product between every pair of feature vectors

This is a Gram matrix $\mathbf{K}$ where the kernel is:

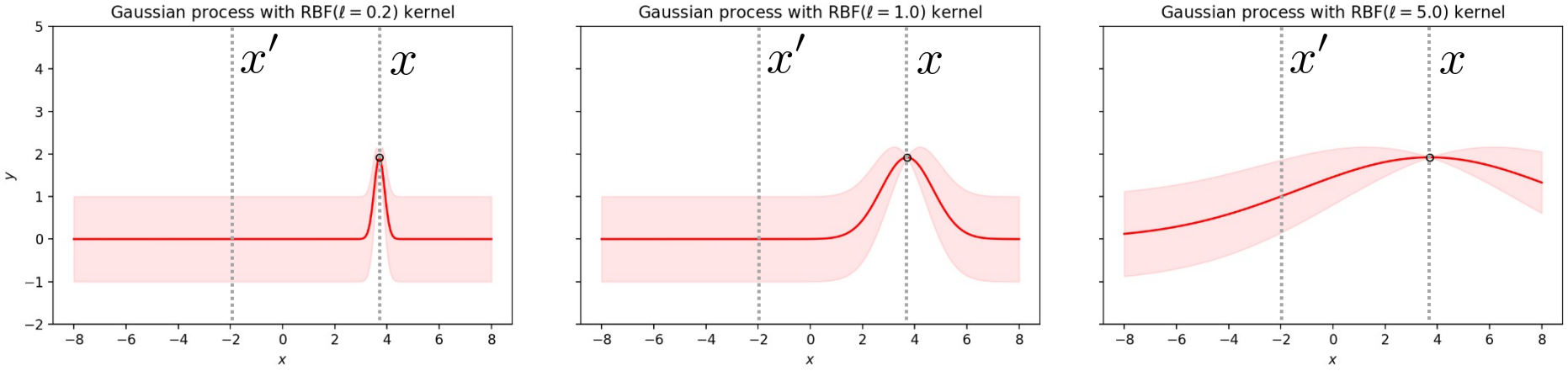
$k(\mathbf{x}, \mathbf{x}') = \frac{1}{\beta}\boldsymbol{\phi}(\mathbf{x})^T\boldsymbol{\phi}(\mathbf{x}')$

where $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$

$= \mathbb{E}[\hat{y}(\mathbf{x}_i)\hat{y}(\mathbf{x}_j)]$

Gaussian Processes

- By changing $\phi(\mathbf{x})$ we change the kernel $k(\mathbf{x}, \mathbf{x}')$ and thereby $p(\hat{\mathbf{y}})$. Can also choose $k(\mathbf{x}, \mathbf{x}')$ directly!
- Just like for SVMs, the kernel $k(\mathbf{x}, \mathbf{x}')$ effectively determines a notion of "similarity":
  - If $k(\mathbf{x}, \mathbf{x}')$ is large value, then $\mathbf{x}$ and $\mathbf{x}'$ are considered "closer" in ϕ-space, so $\hat{y}(\mathbf{x})$ and $\hat{y}(\mathbf{x}')$ more correlated



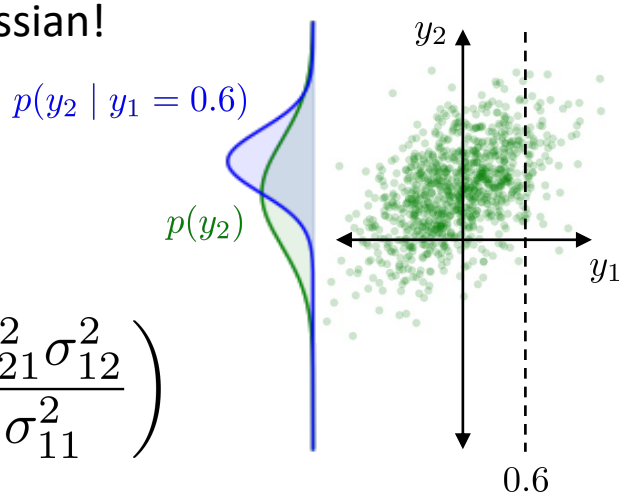
Fact about multivariate Gaussians:

Suppose two variables y_1 and y_2 are jointly Gaussian. If we observe y_1, then the resulting conditional distribution over y_2 is still Gaussian!

$$p\left(\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 \\ \sigma_{21}^2 & \sigma_{22}^2 \end{bmatrix}\right)$$

$$\Downarrow$$

$$p\left(y_2 \mid y_1\right) = \mathcal{N}\left(\mu_2 + \frac{\sigma_{21}^2}{\sigma_{11}^2}(y_1 - \mu_1), \sigma_{22}^2 - \frac{\sigma_{21}^2 \sigma_{12}^2}{\sigma_{11}^2}\right)$$



The same holds for any two vectors $\mathbf{y}_1 \in \mathbb{R}^{N_1}$ and $\mathbf{y}_2 \in \mathbb{R}^{N_2}$ that are jointly Gaussian: the resulting conditional distribution is multivariate Gaussian!

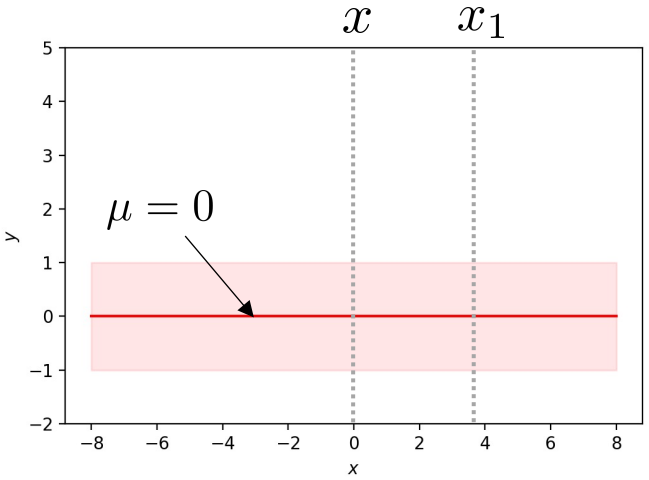
$$p\left(\begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{bmatrix}\right)$$

$$\Downarrow$$

$$p\left(\mathbf{y}_2 \mid \mathbf{y}_1\right) = \mathcal{N}\left(\boldsymbol{\mu}_2 + \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}(\mathbf{y}_1 - \boldsymbol{\mu}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21}\boldsymbol{\Sigma}_{11}^{-1}\boldsymbol{\Sigma}_{12}\right)$$

← Mean and covariance matrix breaks down into block structure corresponding to $\mathbf{y}_1$ and $\mathbf{y}_2$

How to predict?
"1 training point" case:



Let $\mathcal{D} = \{(x_1, y_1)\}$ and $\hat{y} = \hat{y}(x)$

and assume Gaussian process, so

$$p\left(\begin{bmatrix} y_1 \\ \hat{y} \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \mu(x_1) \\ \mu(x) \end{bmatrix}, \begin{bmatrix} k(x_1,x_1) & k(x_1,x) \\ k(x,x_1) & k(x,x) \end{bmatrix}\right)$$

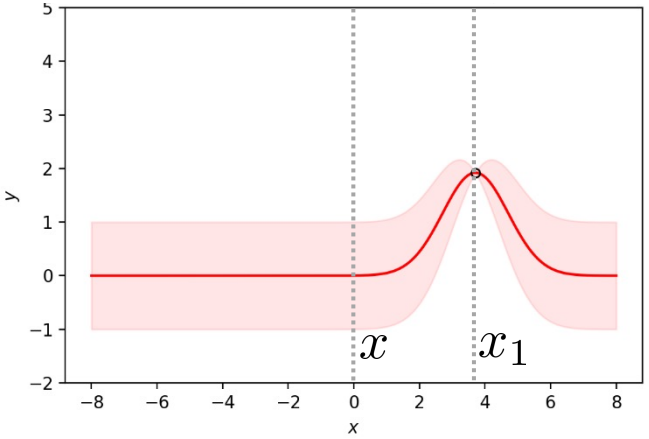
Conditioning on any component *also* results in a Gaussian!

$$p\left(\hat{y} \mid y_1\right) = \mathcal{N}\left(\mu(x) + \frac{k(x,x_1)}{k(x_1,x_1)}(y_1 - \mu(x_1)), k(x,x) - \frac{k(x,x_1)k(x_1,x)}{k(x_1,x_1)}\right)$$

Assuming mean zero gives

$$\mathbb{E}[\hat{y}(x)] = \frac{k(x,x_1)}{k(x_1,x_1)} y_1$$

$$\mathrm{Var}[\hat{y}(x)] = k(x,x) - \frac{k(x,x_1)k(x_1,x)}{k(x_1,x_1)}$$



How to predict, general case

The joint probability of both the observed targets $\mathbf{y}$ at $\mathbf{X}$ and the predictions $\hat{\mathbf{y}}$ we want to make at $\hat{\mathbf{X}}$:

$$p\left(\begin{bmatrix}\mathbf{y}\\ \hat{\mathbf{y}}\end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}\\ \hat{\boldsymbol{\mu}}\end{bmatrix}, \begin{bmatrix}K(\mathbf{X},\mathbf{X}) & K(\mathbf{X},\hat{\mathbf{X}})\\ K(\hat{\mathbf{X}},\mathbf{X}) & K(\hat{\mathbf{X}},\hat{\mathbf{X}})\end{bmatrix}\right)$$

As before, conditioning on observed points gives a new Gaussian distribution over the query points:

$$p\left(\hat{\mathbf{y}} \mid \mathbf{y}\right) = \mathcal{N}\big(\hat{\boldsymbol{\mu}} + K(\hat{\mathbf{X}},\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}(\mathbf{y}-\boldsymbol{\mu}),\\ K(\hat{\mathbf{X}},\hat{\mathbf{X}}) - K(\hat{\mathbf{X}},\mathbf{X})K(\mathbf{X},\mathbf{X})^{-1}K(\mathbf{X},\hat{\mathbf{X}})\big)$$

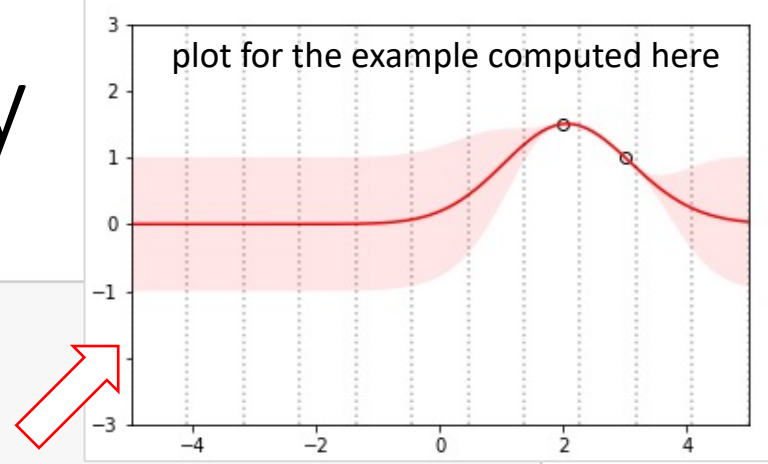
Gram matrix of points we want to predict on

Gram matrix between training points and points we want to predict on

Inverse of Gram matrix of training points

GP prediction in Numpy

For simplicity we've assumed $\mu = \hat{\mu} = 0$




```
def K(X, Y, length_scale=1.0):
    """Returns the (N,M) RBF kernel from
    (N,D) matrix X0 and (M,D) matrix X1"""
    n, d = X.shape
    m, _ = Y.shape
    squared_norms = np.sum((X.reshape(n,1,d) - Y.reshape(1,m,d))**2, axis=2)
    return np.exp(-0.5*squared_norms/length_scale**2)

X1 = np.array([[2.0], [3.0]])                # X values for training (2)
y1 = np.array([1.5, 1.0])                    # y values for training (2)
X2 = np.linspace(-5, 5, 12).reshape(-1, 1)   # X values for predicting (12)

K11_inv = np.linalg.inv(K(X1, X1))  # Training-training covariances (inverted)
K12 = K(X1, X2)                     # Training-predicting covariances
K21 = K12.T                         # Prediction-training covariances
K22 = K(X2, X2)                     # Prediction-prediction covariances

y2_mean = K21 @ K11_inv @ y1                 # Mean at each prediction X
y2_std = np.diag(K22 - K21 @ K11_inv @ K12)  # Stddev at each prediction X

print(y2_mean)  # Print the mean at the 12 prediction points
print(y2_std)   # Print the standard deviations at the 12 prediction points
```

```
[0.    0.    0.    0.    0.005 0.07  0.434 1.192 1.471 0.843 0.238 0.035]
[1.    1.    1.    1.    1.    0.996 0.875 0.247 0.019 0.017 0.618 0.974]
```

Gram matrices HUGE, so real implementations do all this in more efficient way

GPs in scikit-learn

class `sklearn.gaussian_process.` **`GaussianProcessRegressor`**(*kernel=None, *, alpha=1e-10, optimizer='fmin_l_bfgs_b', n_restarts_optimizer=0, normalize_y=False, copy_X_train=True, random_state=None*) [source]

Gaussian process regression (GPR).

The implementation is based on Algorithm 2.1 of Gaussian Processes for Machine Learning (GPML) by Rasmussen and Williams.

Kernel is the main hyperparameter
Kernels can be composed

Parameters:

kernel : ***kernel instance,***

The kernel specifying the covariance function of the GP. If None is passed, the kernel "1.0 * RBF(1.0)" is used as default. Note that the kernel's hyperparameters are optimized during fitting.

alpha : ***float or array-like of shape (n_samples), default=1e-10***

Value added to the diagonal of the kernel matrix during fitting. Larger values correspond to increased noise level in

Measurements may have uncertainty

GPs in scikit-learn

predict(*X*, *return_std=False*, *return_cov=False*) [source]

Predict using the Gaussian process regression model.

We can also predict based on an unfitted model by using the GP prior. In addition to the mean of the predictive distribution, optionally also returns its standard deviation (`return_std=True`) or covariance (`return_cov=True`). Note that at most one of the two can be requested.

Parameters:

X : *array-like of shape (n_samples, n_features) or list of object*
Query points where the GP is evaluated.

return_std : *bool, default=False*
If True, the standard-deviation of the predictive distribution at the query points is returned along with the mean.

return_cov : *bool, default=False*
If True, the covariance of the joint predictive distribution at the query points is returned along with the mean.

Unlike most estimators, can ask for uncertainty in the prediction, cool!

Returns:

y_mean : *ndarray of shape (n_samples,) or (n_samples, n_targets)*
Mean of predictive distribution a query points.

y_std : *ndarray of shape (n_samples,) or (n_samples, n_targets), optional*
Standard deviation of predictive distribution at query points. Only returned when `return_std` is True.

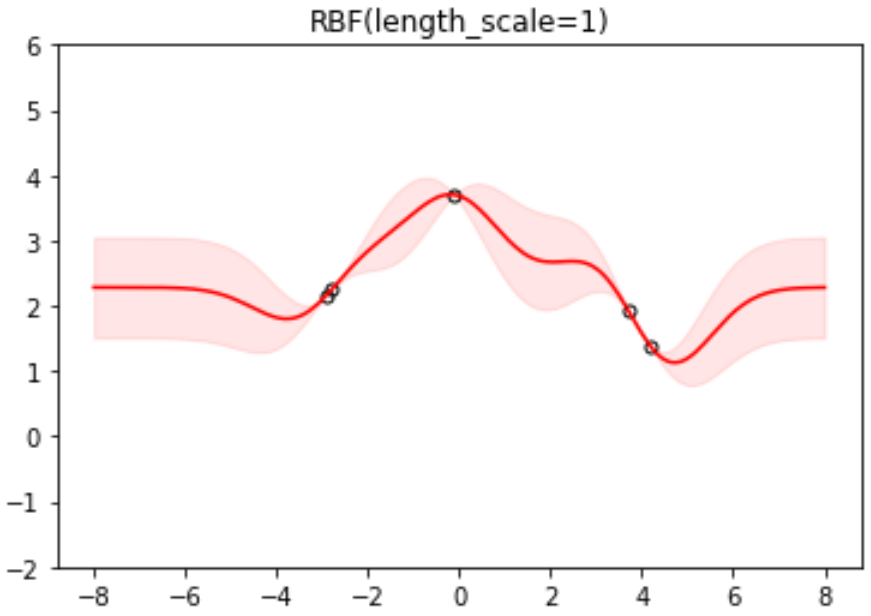
y_cov : *ndarray of shape (n_samples, n_samples) or (n_samples, n_samples, n_targets), optional*
Covariance of joint predictive distribution a query points. Only returned when `return_cov` is True.

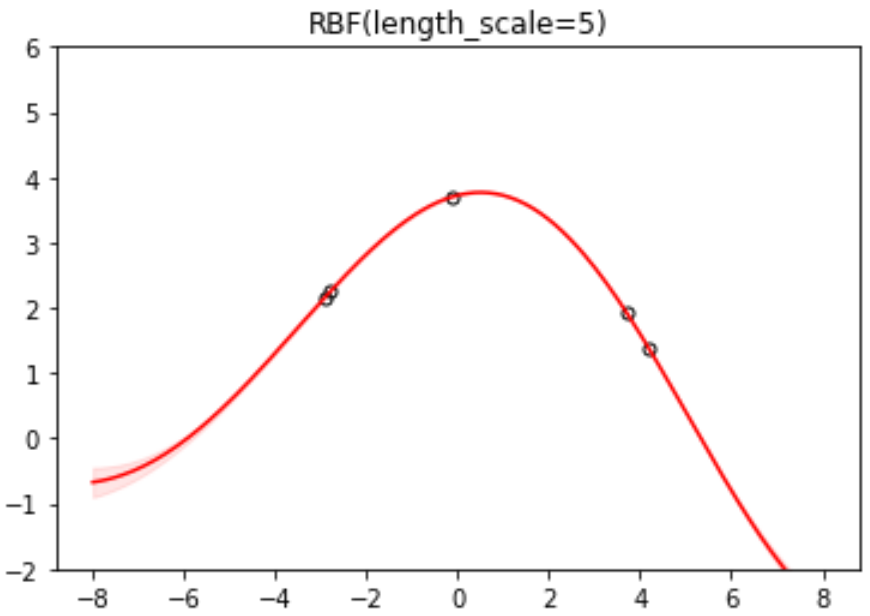
GPs in scikit-learn

```
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process import GaussianProcessRegressor

kernel = RBF(length_scale=1.0)
gp = GaussianProcessRegressor(kernel,
                              normalize_y=True,   # Return mean(y) when far from data.
                              optimizer=None)     # Optionally prevent from tuning length_scale

gp.fit(X, y)
```





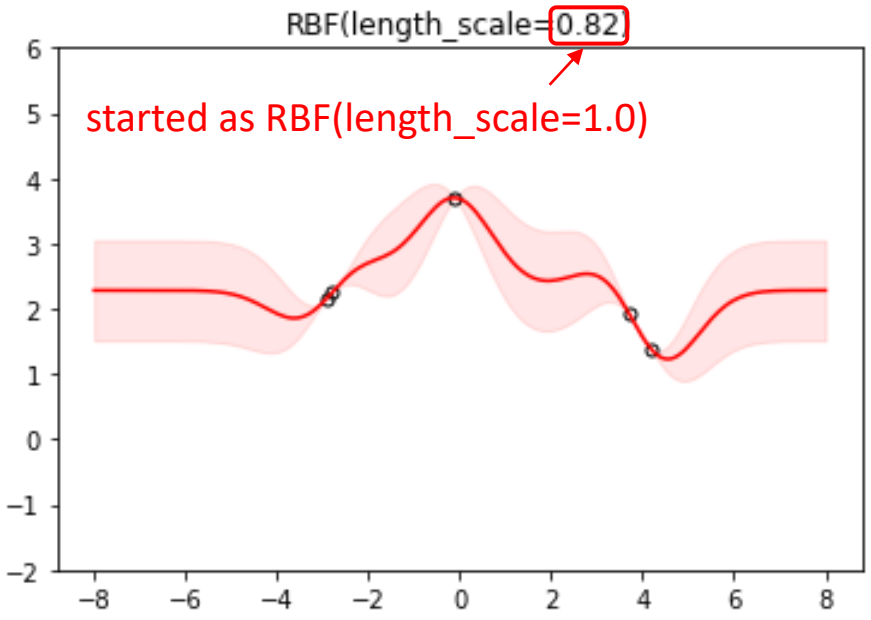
GPs in scikit-learn

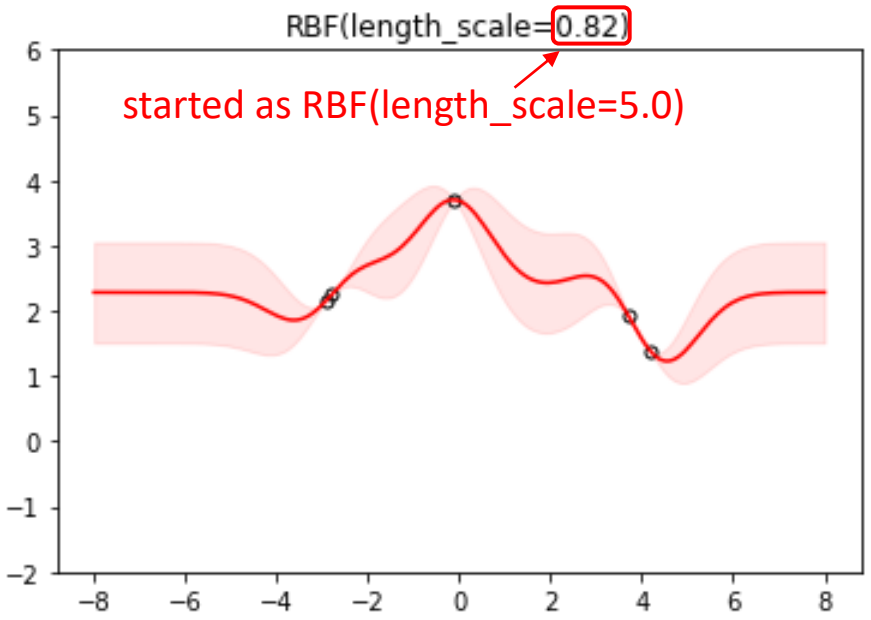
```
from sklearn.gaussian_process.kernels import RBF
from sklearn.gaussian_process import GaussianProcessRegressor

kernel = RBF(length_scale=1.0)
gp = GaussianProcessRegressor(kernel,
                              normalize_y=True)  # Return mean(y) when far from data.

gp.fit(X, y)
```

By default, GP will try to "fit" all kernel parameters, no matter what you specified in your kernel!

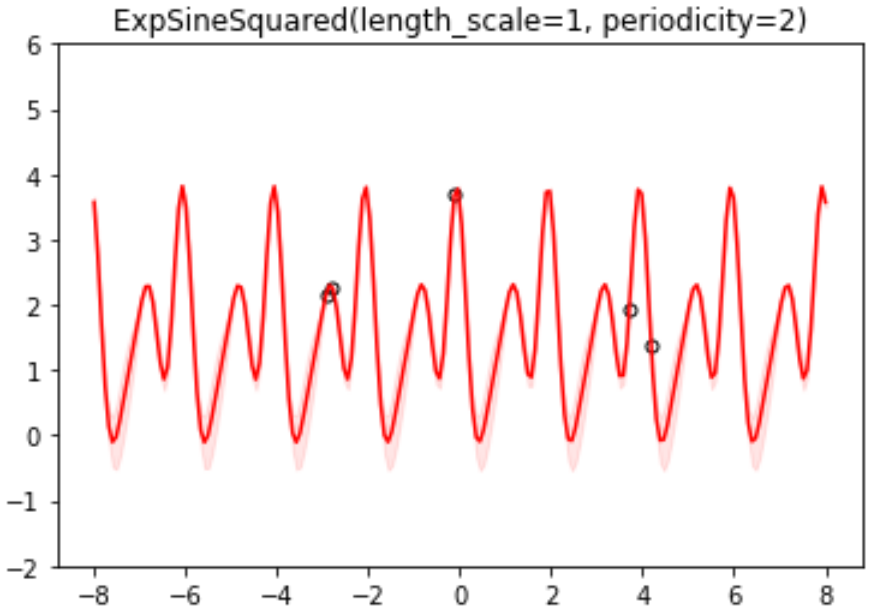


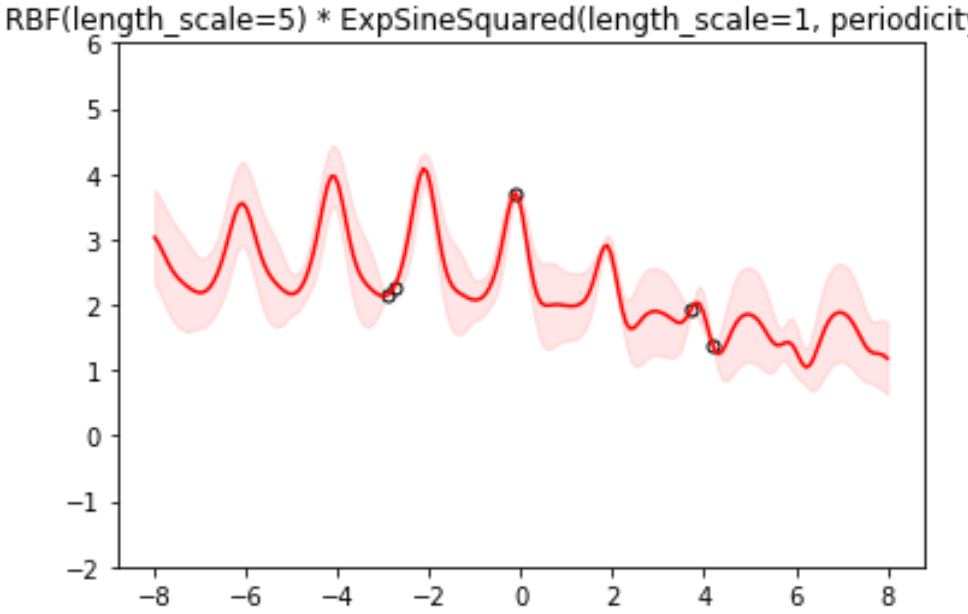


GPs in scikit-learn

```
from sklearn.gaussian_process.kernels import RBF, ExpSineSquared
from sklearn.gaussian_process import GaussianProcessRegressor

kernel = RBF(length_scale=5.0) * ExpSineSquared(length_scale=1.0, periodicity=2.0)
gp = GaussianProcessRegressor(kernel,
                              normalize_y=True,   # Return mean(y) when far from data.
                              optimizer=None)     # Optionally prevent from tuning length_scale
gp.fit(X, y)
```





Techniques for Constructing New Kernels.

from Bishop, 6.2

Given valid kernels $k_1(\mathbf{x}, \mathbf{x}')$ and $k_2(\mathbf{x}, \mathbf{x}')$, the following new kernels will also be valid:

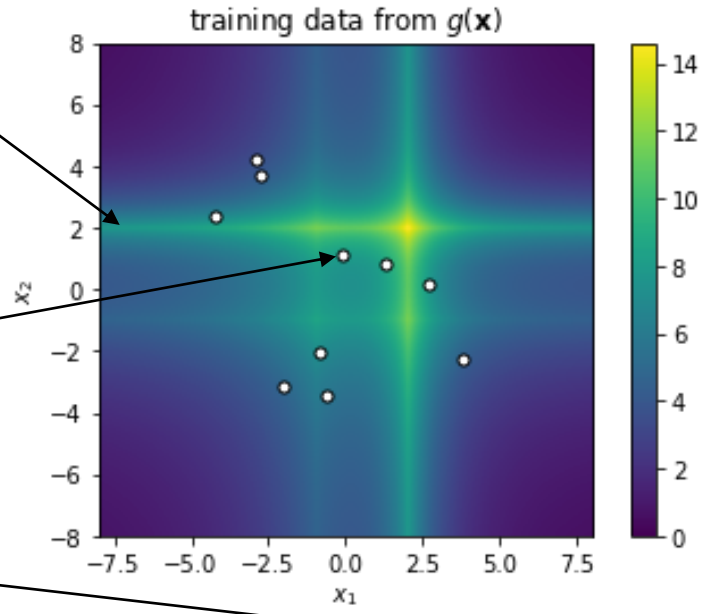
$$
\begin{aligned}
k(\mathbf{x}, \mathbf{x}') &= ck_1(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= f(\mathbf{x})k_1(\mathbf{x}, \mathbf{x}')f(\mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= q\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \exp\left(k_1(\mathbf{x}, \mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}') + k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_1(\mathbf{x}, \mathbf{x}')k_2(\mathbf{x}, \mathbf{x}') \\
k(\mathbf{x}, \mathbf{x}') &= k_3\left(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}')\right) \\
k(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}' \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a) + k_b(\mathbf{x}_b, \mathbf{x}'_b) \\
k(\mathbf{x}, \mathbf{x}') &= k_a(\mathbf{x}_a, \mathbf{x}'_a)k_b(\mathbf{x}_b, \mathbf{x}'_b)
\end{aligned}
$$

where $c > 0$ is a constant, $f(\cdot)$ is any function, $q(\cdot)$ is a polynomial with nonnegative coefficients, $\boldsymbol{\phi}(\mathbf{x})$ is a function from $\mathbf{x}$ to $\mathbb{R}^M$, $k_3(\cdot, \cdot)$ is a valid kernel in $\mathbb{R}^M$, $\mathbf{A}$ is a symmetric positive semidefinite matrix, $\mathbf{x}_a$ and $\mathbf{x}_b$ are variables (not necessarily disjoint) with $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, and k_a and k_b are valid kernel functions over their respective spaces.

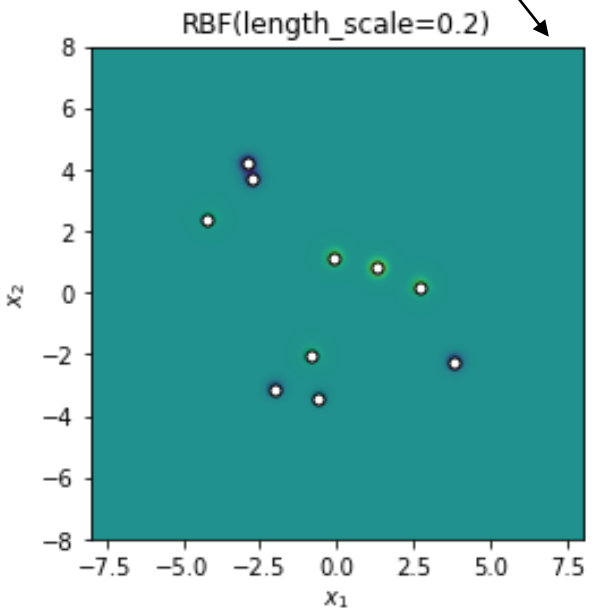
Higher dimensional GPs

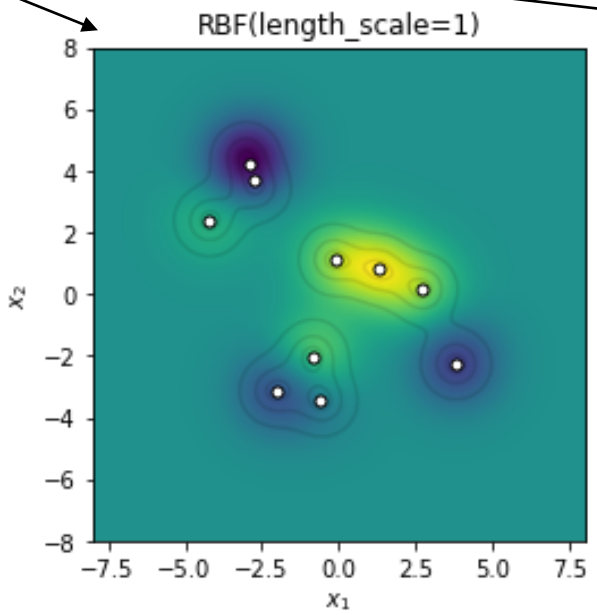
A two dimensional function $g(x_1, x_2)$ shown as the heatmap

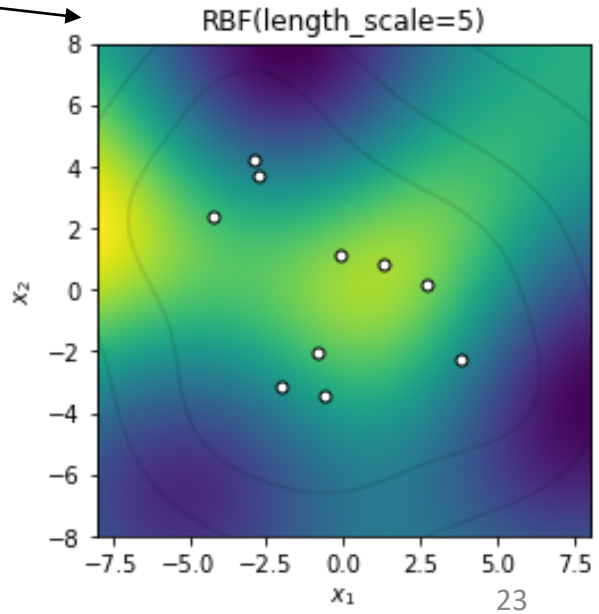
Ten samples of $g(x_1, x_2)$ to make a training set $\{(\mathbf{x}_i, g_i)\}_{i=1}^{10}$



2D Gaussian process regression on the ten samples, i.e., a GP estimate of the original function

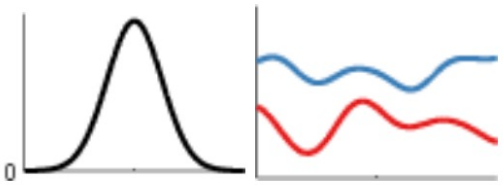






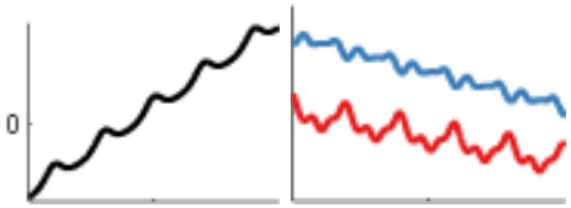
Different kernels correspond to different off-diagonal patterns in infinite-dimensional covariance matrix

Squared Exponential Kernel

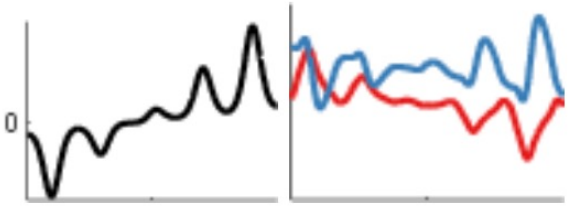


A.K.A. the Radial Basis Function kernel,

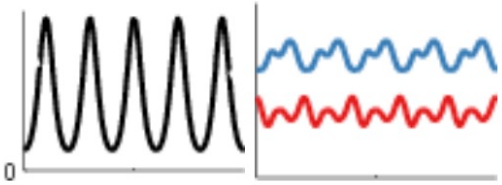
Linear plus Periodic



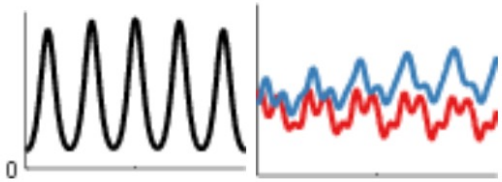
Linear times Periodic



Periodic Kernel



Locally Periodic Kernel



Examples from David Duvenaud's "Kernel Cookbook"
https://www.cs.toronto.edu/~duvenaud/cookbook/

PRML Readings

§6.4.0 Gaussian Processes

§6.4.1 Linear regression revisited

§6.4.2 Gaussian processes for regression

§6.4.5 Gaussian processes for classification