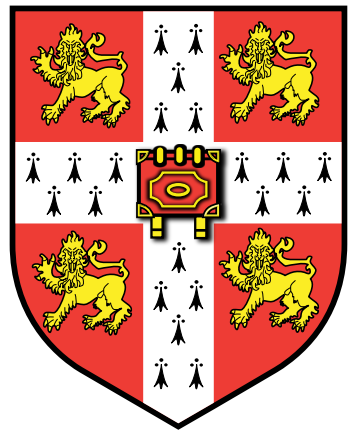
Additive Gaussian Processes

David Duvenaud, Hannes Nickisch, Carl Rasmussen



Cambridge University
Computational and Biological Learning Lab

January 13, 2012

Outline

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Spline Models - Nonparametric

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Spline Models - Nonparametric

Gaussian Process Regression

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Spline Models - Nonparametric

Gaussian Process Regression

- Non-parametric

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Spline Models - Nonparametric

Gaussian Process Regression

- Non-parametric
- Data-Efficient

Regression Methods

Given $\mathbf{X}, \mathbf{y}$, predict some new function value $\mathbf{y}*$ at location $\mathbf{x}*$.
Many methods with nice properties.

Linear Regression - Fast

Deep Belief Networks - Semi-supervised

Spline Models - Nonparametric

Gaussian Process Regression

- Non-parametric
- Data-Efficient
- Tractable Joint Posterior

Definition

Assume our data $(\mathbf{X}, \mathbf{y})$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \epsilon_\sigma$
$\mathbf{f}$ is a **latent function** which we need to do inference about.

Definition

Assume our data $(\mathbf{X}, \mathbf{y})$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \epsilon_\sigma$
$\mathbf{f}$ is a **latent function** which we need to do inference about.
A GP prior distribution over $\mathbf{f}$ means that, for any finite set of indices $\mathbf{X}$,

$$p(\mathbf{f_x}|\theta) = \mathcal{N}(\mu_\theta(\mathbf{X}), \mathbf{K}_\theta(\mathbf{X}, \mathbf{X}))$$

Definition

Assume our data $(\mathbf{X}, \mathbf{y})$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \epsilon_\sigma$
$\mathbf{f}$ is a **latent function** which we need to do inference about.
A GP prior distribution over $\mathbf{f}$ means that, for any finite set of indices $\mathbf{X}$,

$$p(\mathbf{f}_\mathbf{x}|\theta) = \mathcal{N}(\mu_\theta(\mathbf{X}), \mathbf{K}_\theta(\mathbf{X}, \mathbf{X}))$$

where

$$K_{ij} = k_\theta(\mathbf{x}, \mathbf{x}')$$

is the *covariance function* or *kernel*, which specifies the covariance between two function values $f(\mathbf{x}_1), f(\mathbf{x}_2)$ given their locations $\mathbf{x}_1, \mathbf{x}_2$.

Definition

Assume our data $(\mathbf{X}, \mathbf{y})$ is generated by $\mathbf{y} = \mathbf{f}(\mathbf{x}) + \epsilon_\sigma$
$\mathbf{f}$ is a **latent function** which we need to do inference about.
A GP prior distribution over $\mathbf{f}$ means that, for any finite set of indices $\mathbf{X}$,

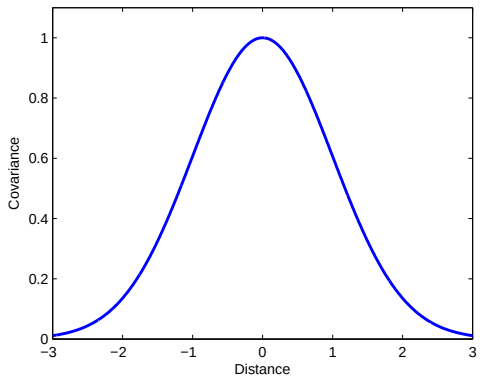
$$p(\mathbf{f}_\mathbf{x}|\theta) = \mathcal{N}(\mu_\theta(\mathbf{X}), \mathbf{K}_\theta(\mathbf{X}, \mathbf{X}))$$

where

$$K_{ij} = k_\theta(\mathbf{x}, \mathbf{x}')$$

is the *covariance function* or *kernel*, which specifies the covariance between two function values $f(\mathbf{x}_1), f(\mathbf{x}_2)$ given their locations $\mathbf{x}_1, \mathbf{x}_2$.
e.g.

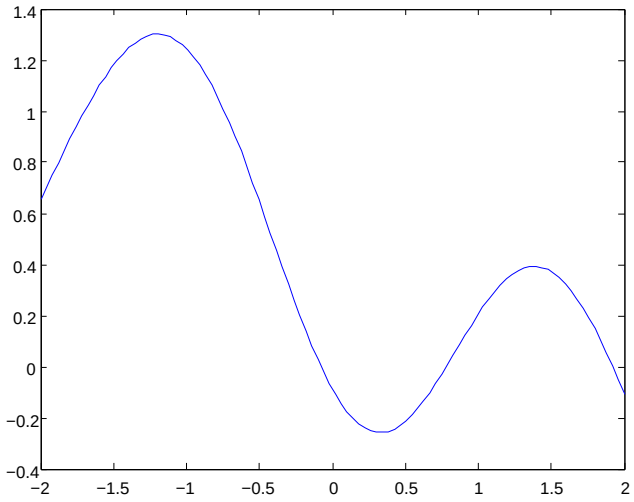
$$k_\theta(\mathbf{x}, \mathbf{x}') = \exp(-\frac{1}{2\theta}|\mathbf{x}-\mathbf{x}'|_2^2)$$



$\mathbf{x} - \mathbf{x}'$

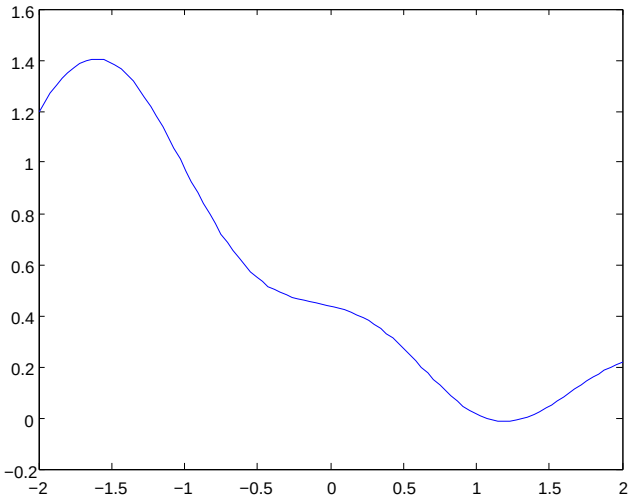
Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
   % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

   % Specify that the prior mean of f is zero.
   mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



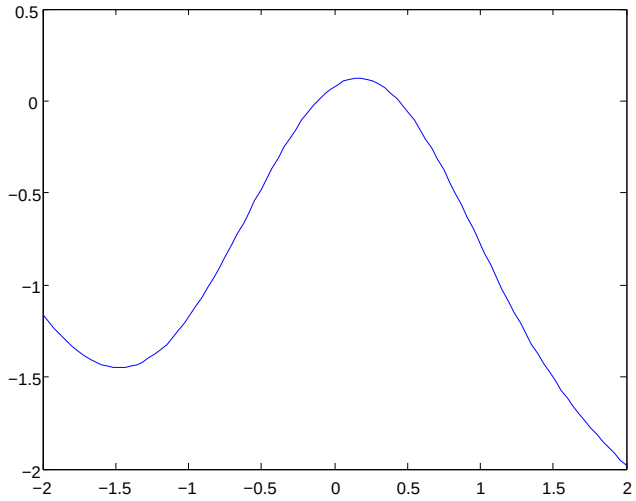
Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
   % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

   % Specify that the prior mean of f is zero.
   mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



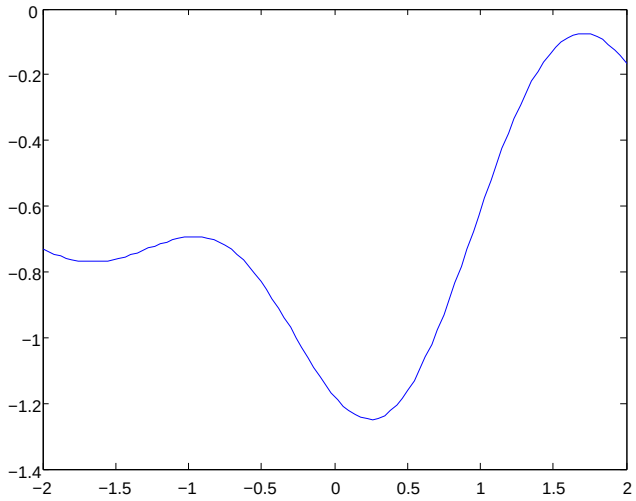
Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
   % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

   % Specify that the prior mean of f is zero.
   mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



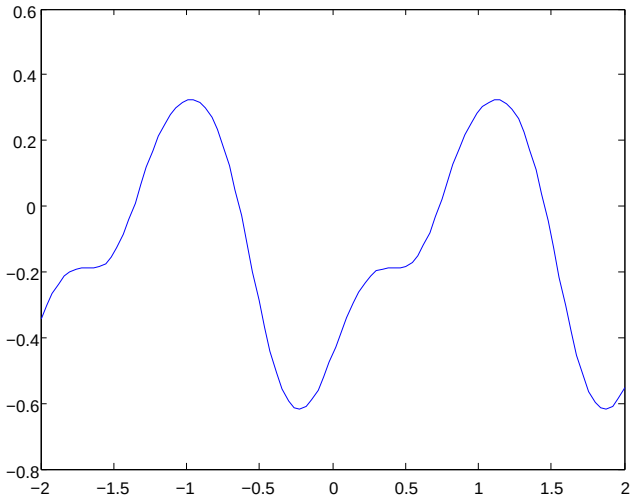
Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
   % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

   % Specify that the prior mean of f is zero.
   mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Squared-exp covariance function.
function k = covariance(x, y)
    k = exp( -0.5*( x - y )^2 );
end
```



Sampling from a GP

```
function simple_gp_sample

    % Choose a set of x locations.
    N = 100;
    x = linspace( -2, 2, N);

    % Specify the covariance between function
   % values, depending on their location.
    for j = 1:N
        for k = 1:N
            sigma(j,k) = covariance( x(j), x(k) );
        end
    end

   % Specify that the prior mean of f is zero.
   mu = zeros(N, 1);

    % Sample from a multivariate Gaussian.
    f = mvnrnd( mu, sigma );

    plot(x, f);
end

% Periodic covariance function.
function c = covariance(x, y)
    c = exp( -0.5*( sin(( x - y )*1.5).^2 ));
end
```



Conditional Posterior

After conditioning on some data $(\mathbf{X}, \mathbf{y})$,

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \frac{1}{Z} p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) p(\mathbf{f}|\theta)$$

Conditional Posterior

After conditioning on some data $(\mathbf{X}, \mathbf{y})$,

$$\begin{aligned} p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) &= \frac{1}{Z} p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) p(\mathbf{f}|\theta) \\ &= \mathcal{N}(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) \mathcal{N}(\mathbf{f}|\mu, K_\theta(\mathbf{X}, \mathbf{X})) \end{aligned}$$

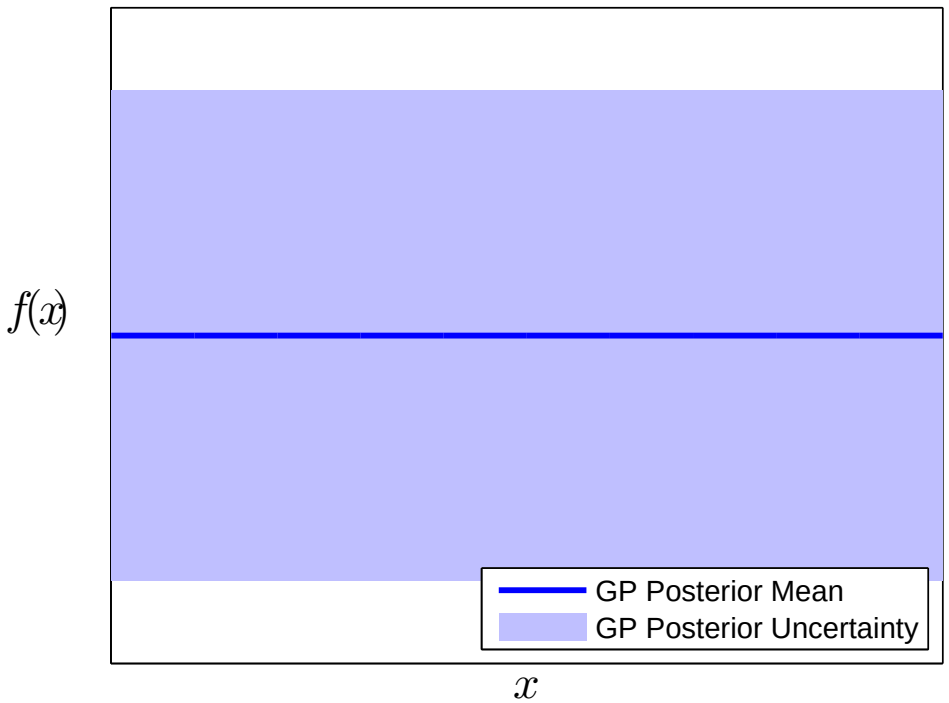
Conditional Posterior

After conditioning on some data $(\mathbf{X}, \mathbf{y})$,

$$
\begin{aligned}
p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) &= \frac{1}{Z} p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) p(\mathbf{f}|\theta) \\
&= \mathcal{N}(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) \mathcal{N}(\mathbf{f}|\mu, K_\theta(\mathbf{X}, \mathbf{X})) \\
&= \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X}) K^{-1} \mathbf{y}, \\
&\qquad \mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X}) K^{-1} k(\mathbf{X}, \mathbf{x}_*))
\end{aligned}
$$

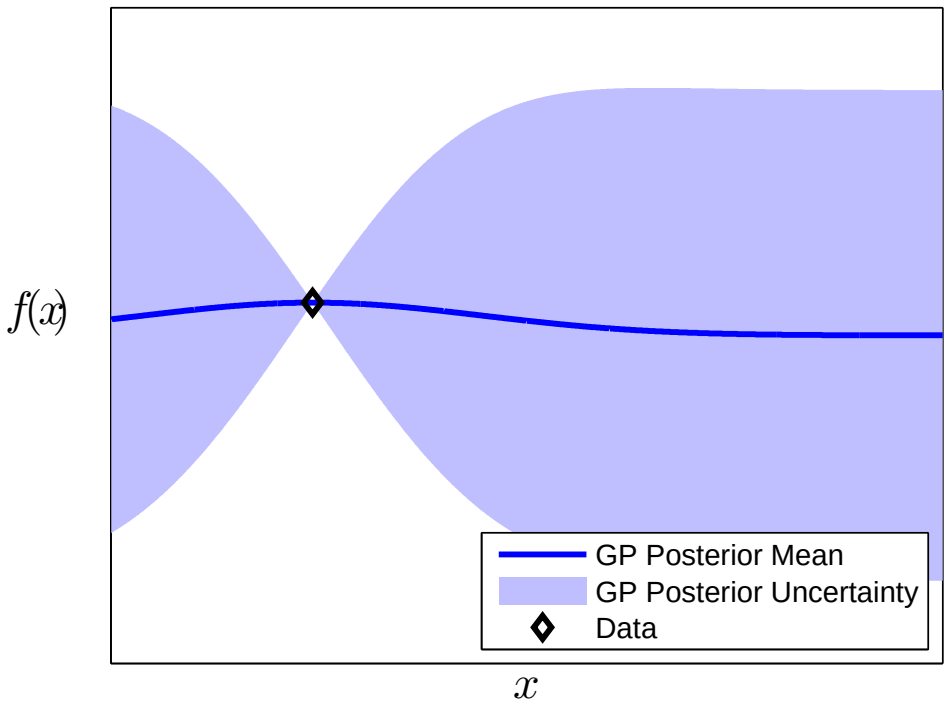
Conditional Posterior

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X})K^{-1}\mathbf{y},$$
$$\mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_*))$$



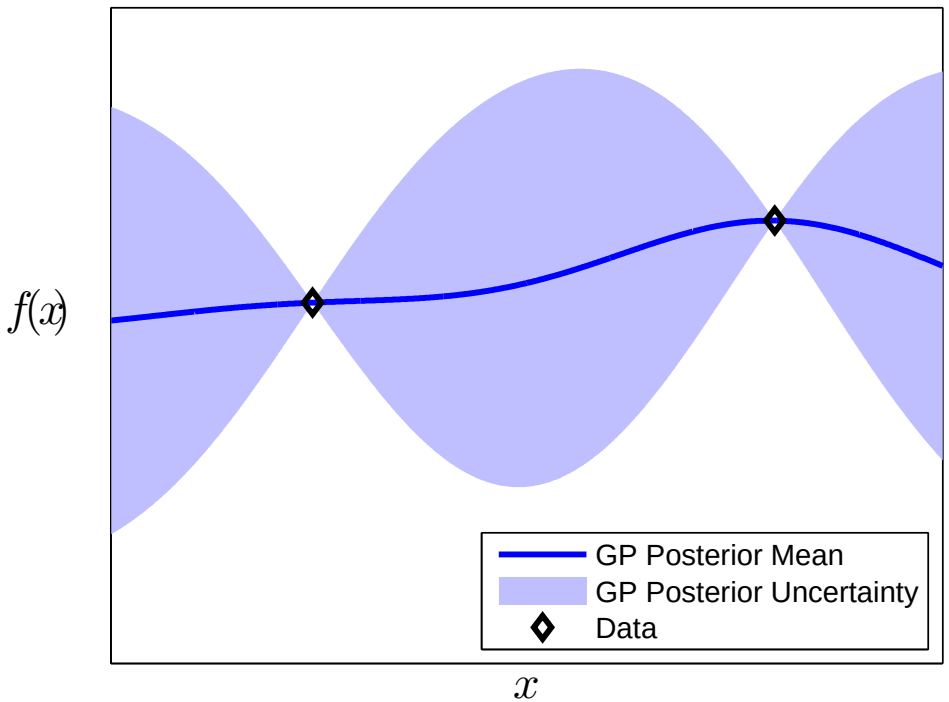
Conditional Posterior

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X})K^{-1}\mathbf{y},$$
$$\mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_*))$$



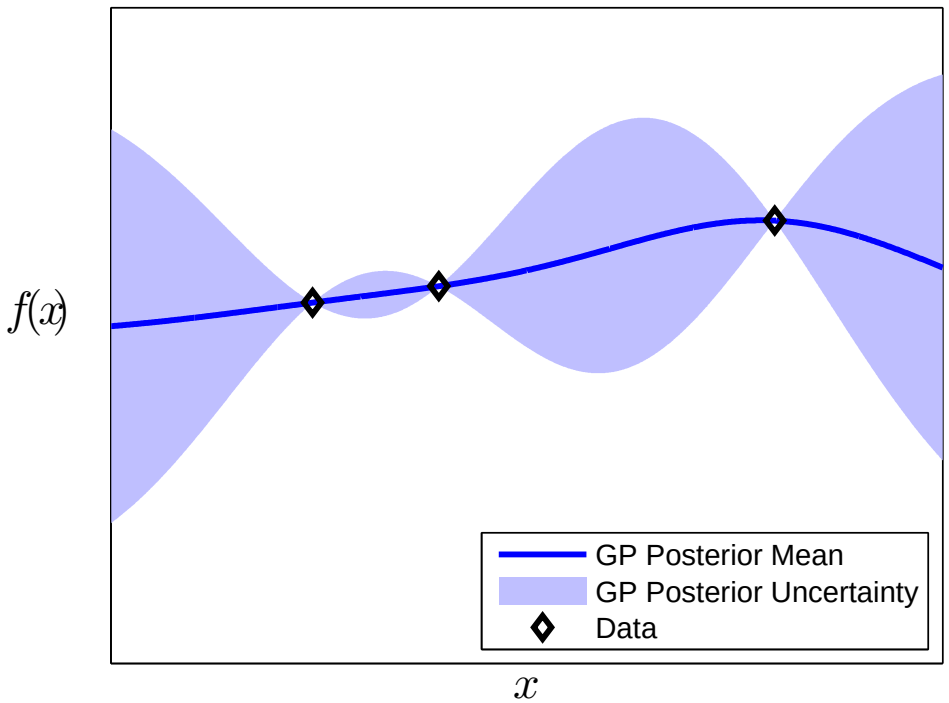
Conditional Posterior

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X})K^{-1}\mathbf{y},$$
$$\mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_*))$$



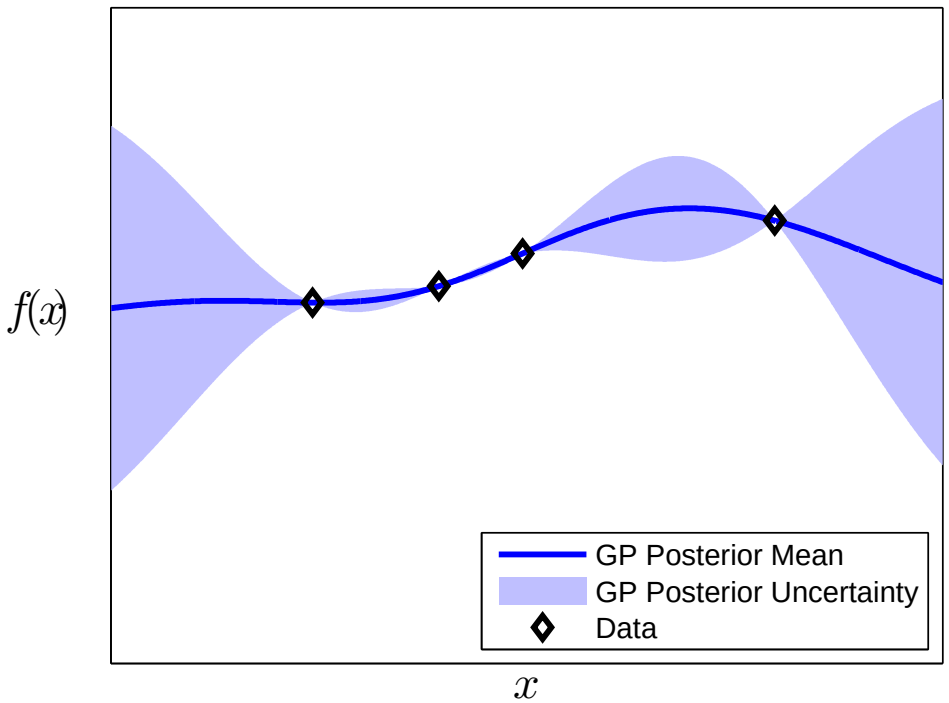
Conditional Posterior

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X})K^{-1}\mathbf{y},$$
$$\mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_*))$$



Conditional Posterior

$$p(\mathbf{f}(\mathbf{x}_*)|\mathbf{X}, \mathbf{y}, \theta) = \mathcal{N}(\mathbf{f}(\mathbf{x}_*)|\mu = k(\mathbf{x}_*, \mathbf{X})K^{-1}\mathbf{y},$$
$$\mathbf{\Sigma} = k(\mathbf{x}_*, \mathbf{x}_*) - k(\mathbf{x}_*, \mathbf{X})K^{-1}k(\mathbf{X}, \mathbf{x}_*))$$



Normalization Constant

Problem with Bayesian models:

$$p(\mathbf{f}|\mathbf{X}, \mathbf{y}, \theta) = \frac{1}{Z} p(\mathbf{y}|\mathbf{f}, \mathbf{X}, \theta) p(\mathbf{f}|\theta)$$

Normalization Constant

Problem with Bayesian models:

$$\begin{aligned} p(\mathbf{f}|\mathbf{X},\mathbf{y},\theta) &= \frac{1}{Z} p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta) \\ &= \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)}{\int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f}} \end{aligned}$$

Normalization Constant

Problem with Bayesian models:

$$\begin{aligned} p(\mathbf{f}|\mathbf{X},\mathbf{y},\theta) &= \frac{1}{Z}p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta) \\ &= \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)}{\int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f}} \end{aligned}$$

Can actually compute model evidence $p(\mathbf{y}|\mathbf{X},\theta)$, aka Z:

Normalization Constant

Problem with Bayesian models:

$$\begin{aligned} p(\mathbf{f}|\mathbf{X},\mathbf{y},\theta) &= \frac{1}{Z} p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta) \\ &= \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)}{\int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f}} \end{aligned}$$

Can actually compute model evidence $p(\mathbf{y}|\mathbf{X},\theta)$, aka Z:

$$\log p(\mathbf{y}|\mathbf{X},\theta) = \log \int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f}$$

Normalization Constant

Problem with Bayesian models:

$$\begin{aligned} p(\mathbf{f}|\mathbf{X},\mathbf{y},\theta) &= \frac{1}{Z}p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta) \\ &= \frac{p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)}{\int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f}} \end{aligned}$$

Can actually compute model evidence $p(\mathbf{y}|\mathbf{X},\theta)$, aka Z:

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X},\theta) &= \log \int p(\mathbf{y}|\mathbf{f},\mathbf{X},\theta)p(\mathbf{f}|\theta)d\mathbf{f} \\ &= \underbrace{-\frac{1}{2}\mathbf{y}^T(\mathbf{K}_\theta + \sigma_\epsilon^2\mathbb{I})^{-1}\mathbf{y}}_{\text{Data-fit}} \underbrace{-\frac{1}{2}\log|\mathbf{K}_\theta + \sigma_\epsilon^2\mathbb{I}|}_{\text{Bayesian Occam's Razor}} -\frac{N}{2}\log(2\pi) \end{aligned}$$

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

- Linear Regression

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

- Linear Regression
- Polynomial Regression

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

- Linear Regression
- Polynomial Regression
- Splines

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

- Linear Regression
- Polynomial Regression
- Splines
- Kalman Filters

Choice of Kernel Function

Depending on kernel function, GP Regression is equivalent to:

- Linear Regression
- Polynomial Regression
- Splines
- Kalman Filters
- Generalized Additive Models

Choice of Kernel Function

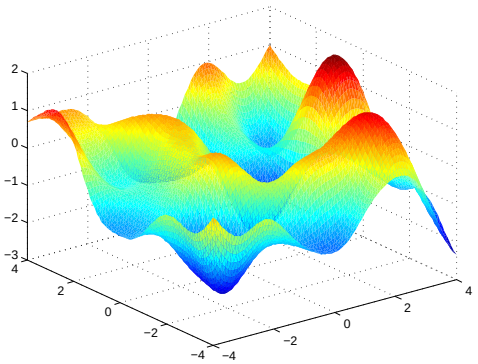
Depending on kernel function, GP Regression is equivalent to:

- Linear Regression
- Polynomial Regression
- Splines
- Kalman Filters
- Generalized Additive Models

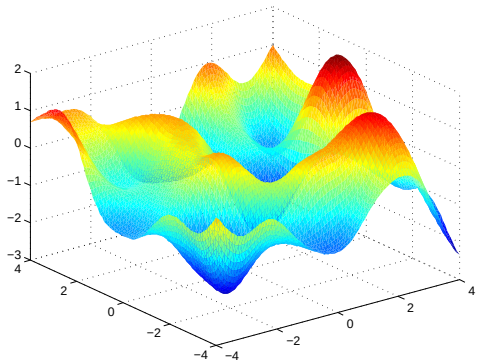
Can use gradients of model evidence to learn which model best explains the data; no need for cross-validation.

Not just for 1-D continuous functions

- D-dimensional input

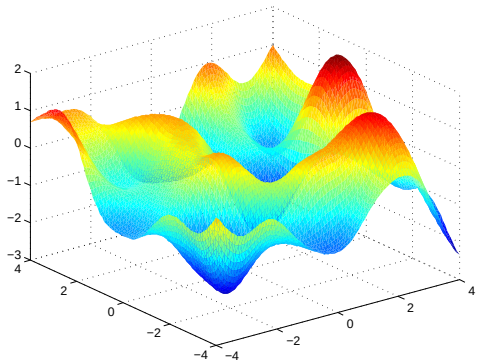


Not just for 1-D continuous functions



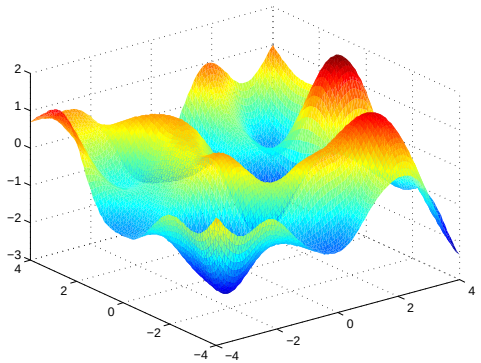
- D-dimensional input
- Functions over discrete domains

Not just for 1-D continuous functions



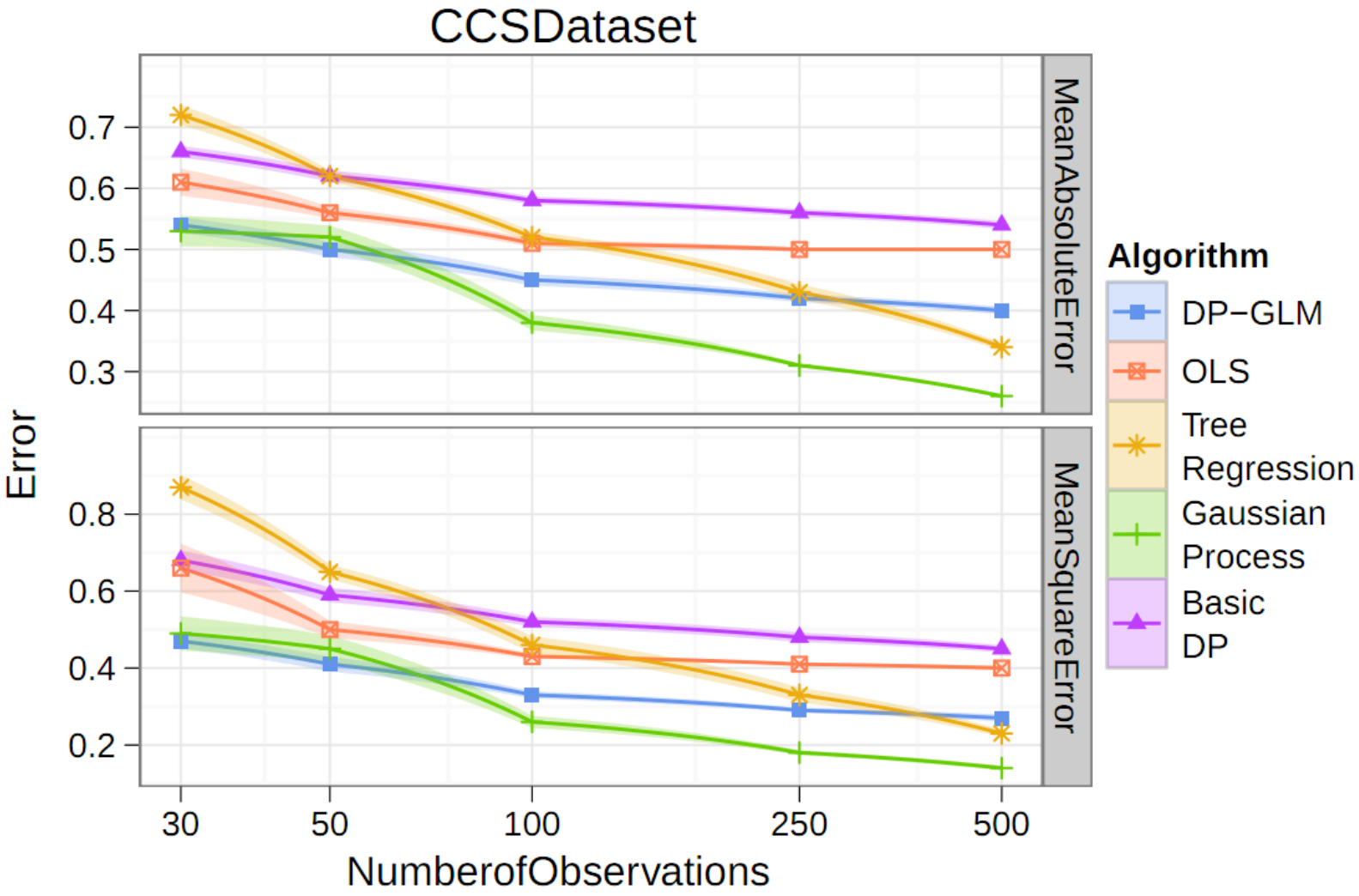
- D-dimensional input
- Functions over discrete domains
- Functions over strings, trees, trajectories

Not just for 1-D continuous functions



- D-dimensional input
- Functions over discrete domains
- Functions over strings, trees, trajectories
- Classification

Performance



[Blei et. al, 2011]

Limitations

- Slow: $O(N^3)$ means that $N < 3000$.

Limitations

- Slow: $O(N^3)$ means that $N < 3000$.
  - Recently some good $O(NM^2)$ approximations (FITC).

Limitations

- Slow: $O(N^3)$ means that $N < 3000$.
  - Recently some good $O(NM^2)$ approximations (FITC).
- Most commonly used kernels have fairly limited generalization abilities.

Limitations

- Slow: $O(N^3)$ means that $N < 3000$.
  - Recently some good $O(NM^2)$ approximations (FITC).
- Most commonly used kernels have fairly limited generalization abilities.
- Non-Gaussian noise requires approximate inference.

Limitations

- Slow: $O(N^3)$ means that $N < 3000$.
  - Recently some good $O(NM^2)$ approximations (FITC).
- Most commonly used kernels have fairly limited generalization abilities.
- Non-Gaussian noise requires approximate inference.

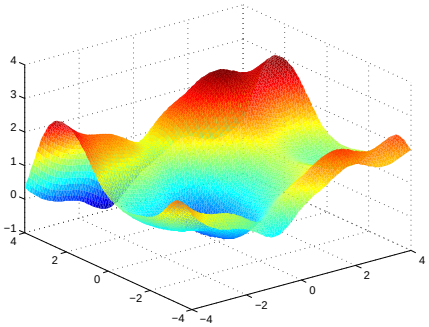
Best choice if:

- Data is small / expensive to gather.
- You want to do anything besides point prediction.

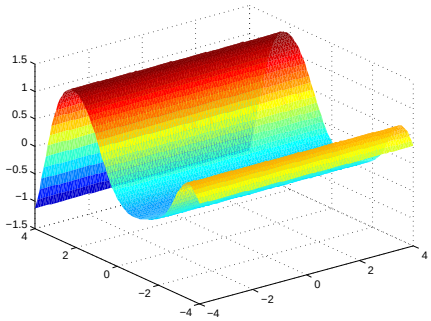
Outline

Central Dogma

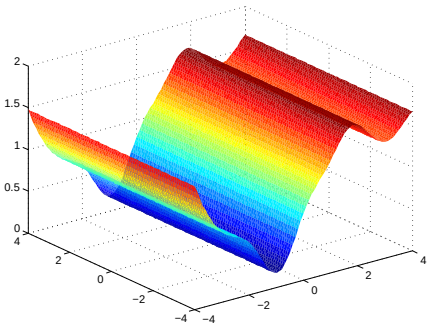
Central modeling assumption:



$f_1(x_1) + f_2(x_2)$ $=$



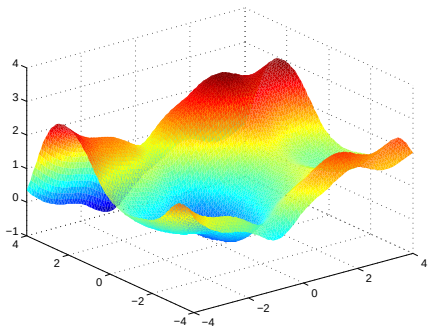
$f_1(x_1)$ $+$



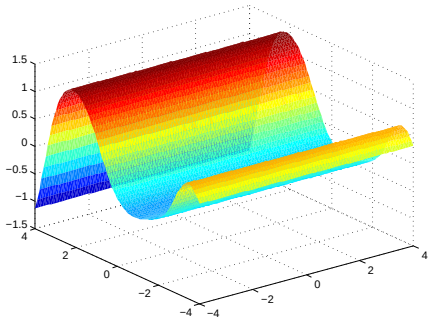
$f_2(x_2)$

Central Dogma

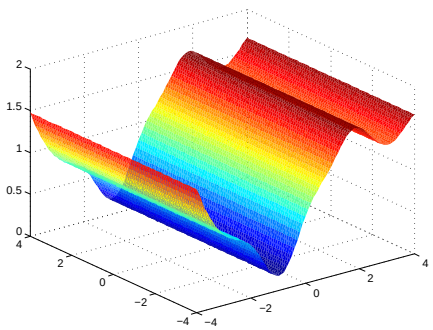
Central modeling assumption:



$f_1(x_1) + f_2(x_2)$ $=$



$f_1(x_1)$ $+$

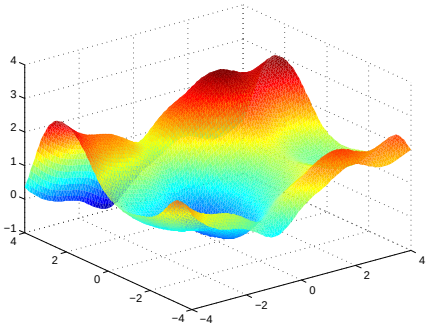


$f_2(x_2)$

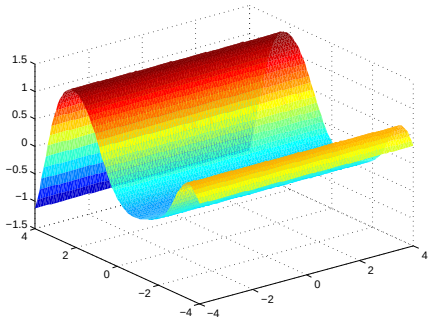
We hope our high-dimensional function can be written as a sum of orthogonal low-dimensional functions.

Central Dogma

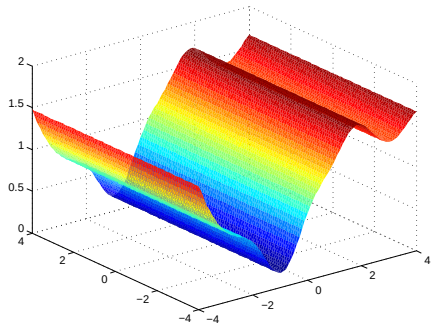
Central modeling assumption:



$f_1(x_1) + f_2(x_2)$ $=$



$f_1(x_1)$ $+$



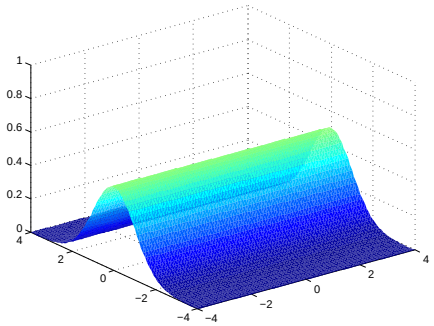
$f_2(x_2)$

We hope our high-dimensional function can be written as a sum of orthogonal low-dimensional functions.

it's far easier to learn ten 1-dimensional functions than one 10-dimensional function!

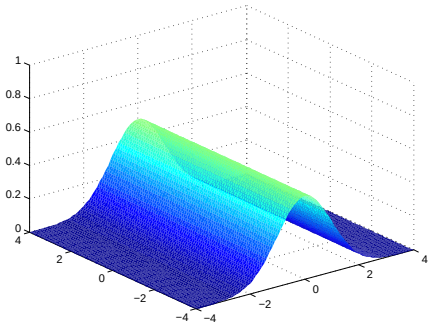
Additivity in GPs

Easy to express additive property in a GP:



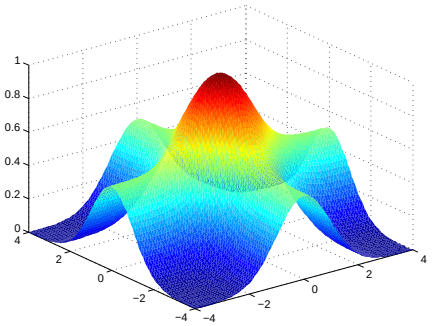
$k_1(x_1, x_2')$
1D kernel

+



$k_2(x_2, x_2')$
1D kernel

=

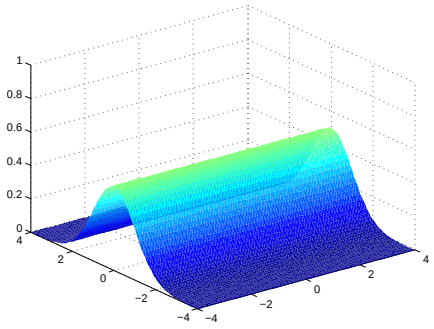


$k_1(x_1, x_1') + k_2(x_2, x_2')$

↓

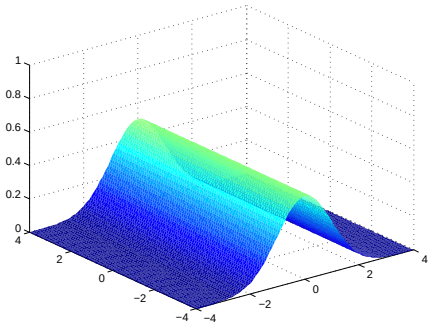
Additivity in GPs

Easy to express additive property in a GP:



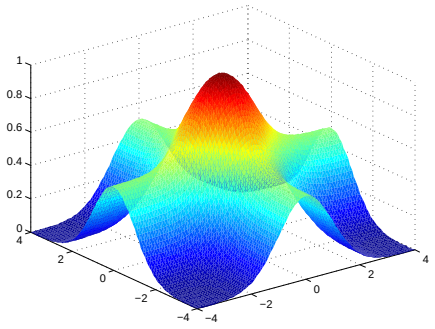
$k_1(x_1, x_2')$
1D kernel

+



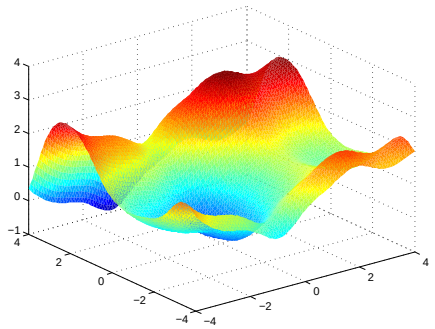
$k_2(x_2, x_2')$
1D kernel

=



$k_1(x_1, x_1') + k_2(x_2, x_2')$

↓



$f_1(x_1) + f_2(x_2)$
draw from
1st order GP

We can extend our prior to include more interaction terms:

We can extend our prior to include more interaction terms:

$$
\begin{aligned}
& f(x_1, x_2, x_3, x_4) = \\
& f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4) \\
+ \quad & f_{12}(x_1, x_2) + f_{13}(x_1, x_3) + f_{14}(x_1, x_4) + f_{23}(x_2, x_3) + f_{24}(x_2, x_4) + f_{34}(x_3, x_4) \\
+ \quad & f_{123}(x_1, x_2, x_3) + f_{124}(x_1, x_2, x_4) + f_{134}(x_1, x_3, x_4) + f_{234}(x_2, x_3, x_4) \\
+ \quad & f_{1234}(x_1, x_2, x_3, x_4)
\end{aligned}
$$

We can extend our prior to include more interaction terms:

$$
\begin{aligned}
& f(x_1, x_2, x_3, x_4) = \\
& f_1(x_1) + f_2(x_2) + f_3(x_3) + f_4(x_4) \\
+ \quad & f_{12}(x_1, x_2) + f_{13}(x_1, x_3) + f_{14}(x_1, x_4) + f_{23}(x_2, x_3) + f_{24}(x_2, x_4) + f_{34}(x_3, x_4) \\
+ \quad & f_{123}(x_1, x_2, x_3) + f_{124}(x_1, x_2, x_4) + f_{134}(x_1, x_3, x_4) + f_{234}(x_2, x_3, x_4) \\
+ \quad & f_{1234}(x_1, x_2, x_3, x_4)
\end{aligned}
$$

Corresponding GP model: assign each dimension $i \in \{1 \dots D\}$ a one-dimensional *base kernel* $k_i(x_i, x_i')$ Let $z_i = k_i(x_i, x_i')$

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= z_1 + z_2 + z_3 + z_4 \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4 + z_3 z_4 \\
k_{add_3}(\mathbf{x}, \mathbf{x}') &= z_1 z_2 z_3 + z_1 z_2 z_4 + z_1 z_3 z_4 + z_2 z_3 z_4 \\
k_{add_4}(\mathbf{x}, \mathbf{x}') &= z_1 z_2 z_3 z_4
\end{aligned}
$$

In D dimensions:

$$k_{add_1}(\mathbf{x}, \mathbf{x}') = \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i')$$

In D dimensions:

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j')
\end{aligned}
$$

In D dimensions:

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \\
k_{add_3}(\mathbf{x}, \mathbf{x}') &= \sigma_3^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} \sum_{k=j+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') k_k(x_k, x_k')
\end{aligned}
$$

In D dimensions:

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \\
k_{add_3}(\mathbf{x}, \mathbf{x}') &= \sigma_3^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} \sum_{k=j+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') k_k(x_k, x_k') \\
k_{add_n}(\mathbf{x}, \mathbf{x}') &= \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \prod_{d=1}^{N} k_{i_d}(x_{i_d}, x_{i_d}')
\end{aligned}
$$

In D dimensions:

$$
\begin{aligned}
k_{add_1}(\mathbf{x}, \mathbf{x}') &= \sigma_1^2 \sum_{i=1}^{D} k_i(x_i, x_i') \\
k_{add_2}(\mathbf{x}, \mathbf{x}') &= \sigma_2^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') \\
k_{add_3}(\mathbf{x}, \mathbf{x}') &= \sigma_3^2 \sum_{i=1}^{D} \sum_{j=i+1}^{D} \sum_{k=j+1}^{D} k_i(x_i, x_i') k_j(x_j, x_j') k_k(x_k, x_k') \\
k_{add_n}(\mathbf{x}, \mathbf{x}') &= \sigma_n^2 \sum_{1 \le i_1 < i_2 < \ldots < i_n \le D} \prod_{d=1}^{N} k_{i_d}(x_{i_d}, x_{i_d}') \\
k_{add_D}(\mathbf{x}, \mathbf{x}') &= \sigma_D^2 \prod_{d=1}^{D} k_d(x_d, x_d')
\end{aligned}
$$

Full additive kernel is a sum of the additive kernels of all orders, weighted by the order variances $\sigma_1 \ldots \sigma_D$

Efficient Evaluation

The nth order additive kernel corresponds to the nth *elementary symmetric polynomial*

$$e_1(z_1, z_2, z_3, z_4) = z_1 + z_2 + z_3 + z_4$$
$$e_2(z_1, z_2, z_3, z_4) = z_1z_2 + z_1z_3 + z_1z_4 + z_2z_3 + z_2z_4 + z_3z_4$$
$$e_3(z_1, z_2, z_3, z_4) = z_1z_2z_3 + z_1z_2z_4 + z_1z_3z_4 + z_2z_3z_4$$
$$e_4(z_1, z_2, z_3, z_4) = z_1z_2z_3z_4$$

Efficient Evaluation

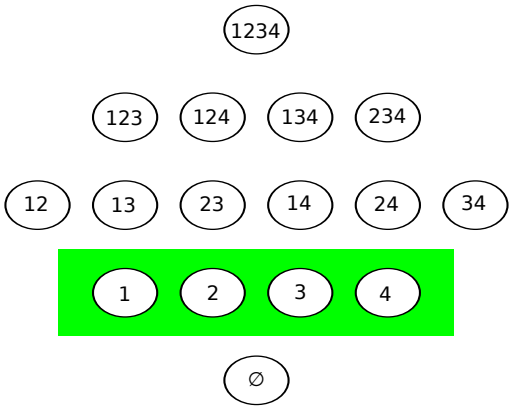
The nth order additive kernel corresponds to the nth *elementary symmetric polynomial*

$$
\begin{aligned}
e_1(z_1, z_2, z_3, z_4) &= z_1 + z_2 + z_3 + z_4 \\
e_2(z_1, z_2, z_3, z_4) &= z_1 z_2 + z_1 z_3 + z_1 z_4 + z_2 z_3 + z_2 z_4 + z_3 z_4 \\
e_3(z_1, z_2, z_3, z_4) &= z_1 z_2 z_3 + z_1 z_2 z_4 + z_1 z_3 z_4 + z_2 z_3 z_4 \\
e_4(z_1, z_2, z_3, z_4) &= z_1 z_2 z_3 z_4
\end{aligned}
$$

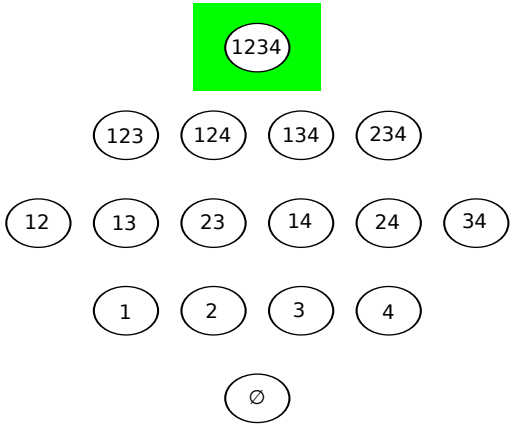
Newton-Girard formulae give efficient recursive form:

$$
e_n(z_1, \ldots, z_D) = \frac{1}{n} \sum_{k=1}^{n} (-1)^{(k-1)} e_{n-k}(z_1, \ldots, z_D) \sum_{i=1}^{D} z_i^k \qquad (2.1)
$$

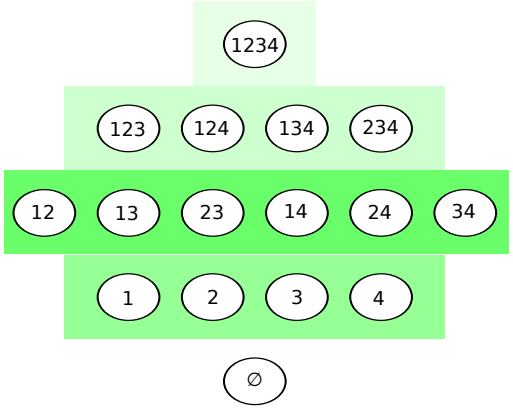
Interpretability



GP-GAM kernel

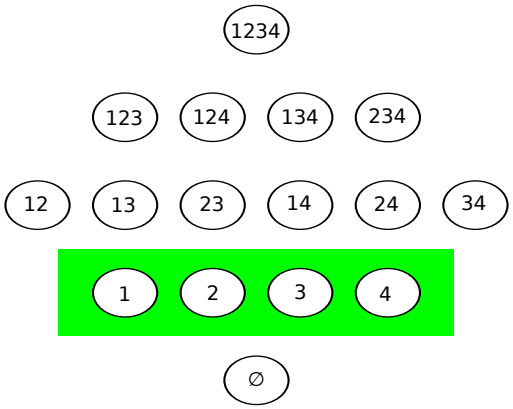


Squared-exp GP kernel

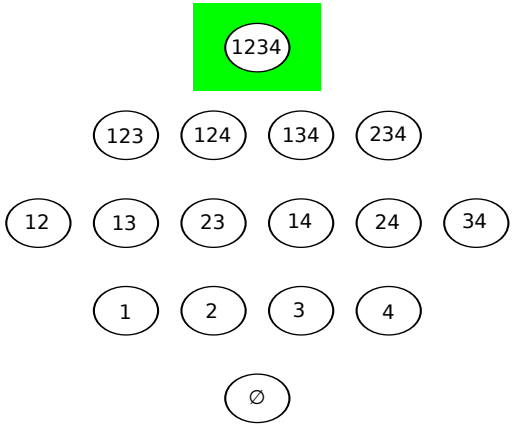


Additive GP kernel

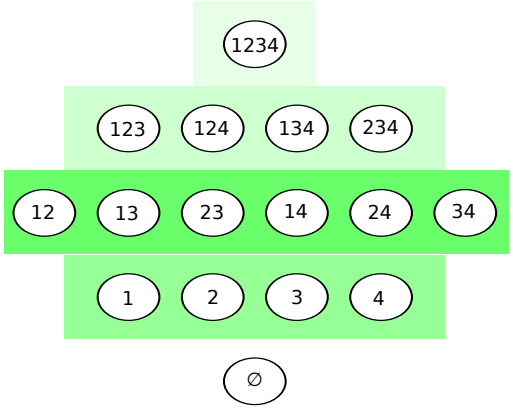
Interpretability



GP-GAM kernel



Squared-exp GP kernel



Additive GP kernel

Order	1st	2nd	3rd	4th	5th	6th	7th	8th	9th	10th
pima	0.1	0.1	0.1	0.3	1.5	**96.4**	1.4	0.0		
liver	0.0	0.2	**99.7**	0.1	0.0	0.0				
heart	**77.6**	0.0	0.0	0.0	0.1	0.1	0.1	0.1	0.1	22.0
concrete	**70.6**	13.3	13.8	2.3	0.0	0.0	0.0	0.0		
pumadyn	0.0	0.1	0.1	0.1	0.1	0.1	0.1	**99.5**		
servo	**58.7**	27.4	0.0	13.9						
housing	0.1	0.6	**80.6**	1.4	1.8	0.8	0.7	0.8	0.6	12.7

Interpretability

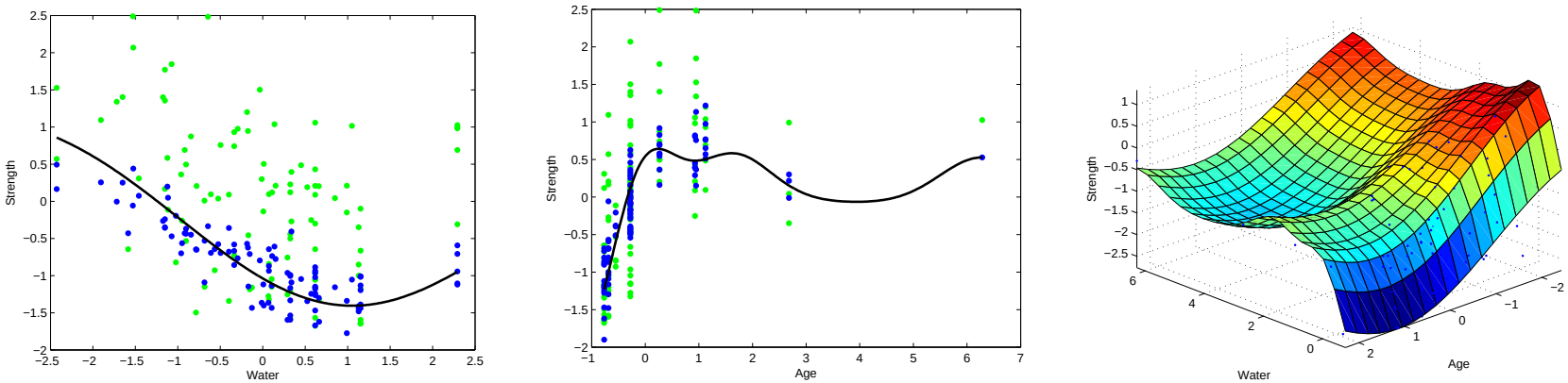


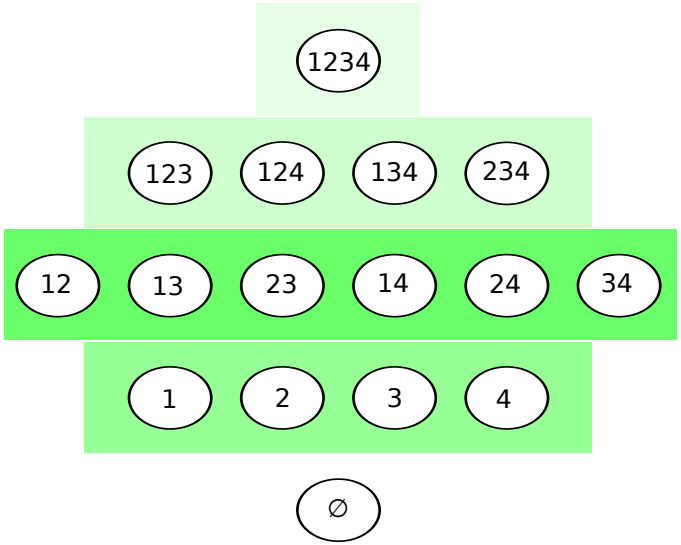
Figure: Green points indicate the original data, blue points are data after the mean contribution from the other first-order terms has been subtracted. The black line is the posterior mean of a GP with only one term in its kernel.

Related Work

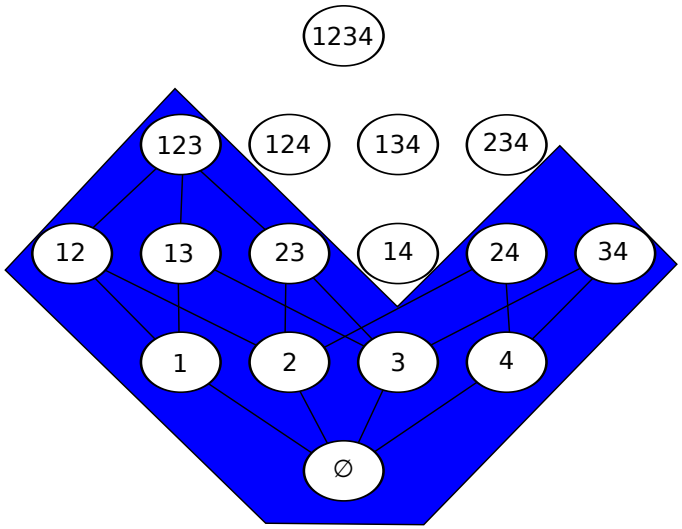
- Vapnik [1998] introduces additive kernel, calling it support vector ANOVA decomposition. Recommend choosing only one order, since must choose hypers by cross-validation.
- Plate [1999] constructs additive GP using only first-order and Dth order terms. Trades off interpretability with goodness of fit.
- Wahba [1990] introduces smoothing-splines ANOVA, a weighted sum of low-D splines, each with an individual weight. In practice, only 1-D and 2-D splines are used.

Additive GPs use all orders of interaction, learn base kernels, are probabilistic.

Hierarchical Kernel Learning (Bach [2009])



Additive GP kernel



HKL kernel

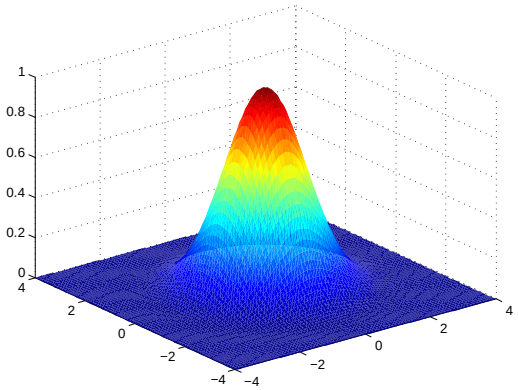
- HKL can selects a hull of interaction terms.
- Must use a pre-determined weighting over orders.
- Uses cross-validation to fit all hypers.

Neither class of kernels contains the other.

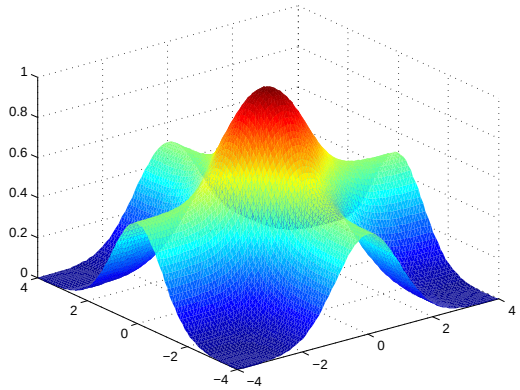
Local Kernels

A GP prior with squared-exp or Matérn kernels say that either the function doesn't change much at all, or that distant points can't tell you much about your current position.

- Nice for consistency
- bad for generalization.

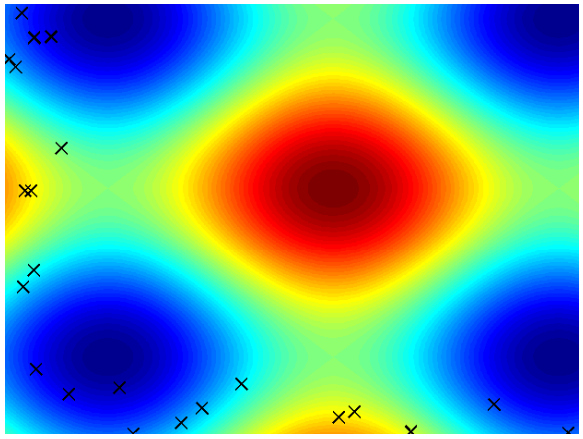


Squared-Exp kernel

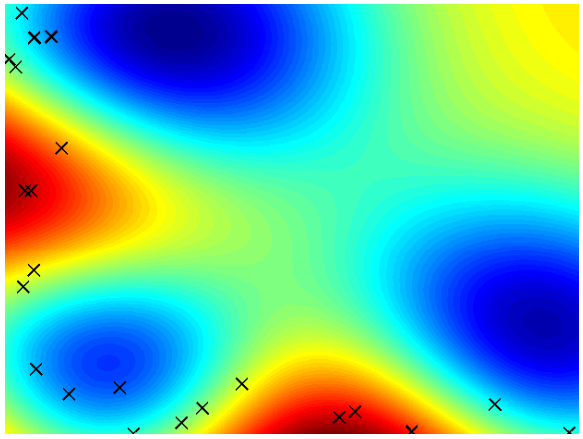


Additive kernel

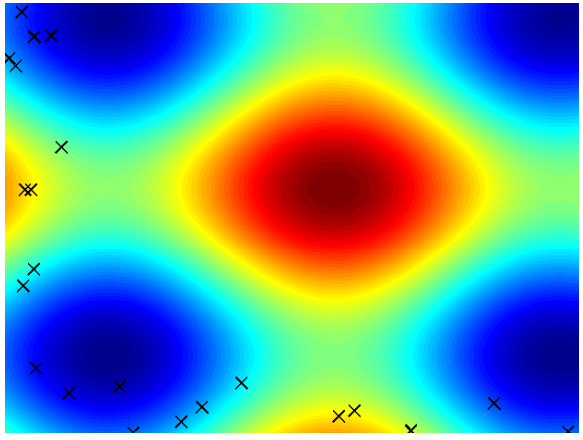
Non-Local Kernels



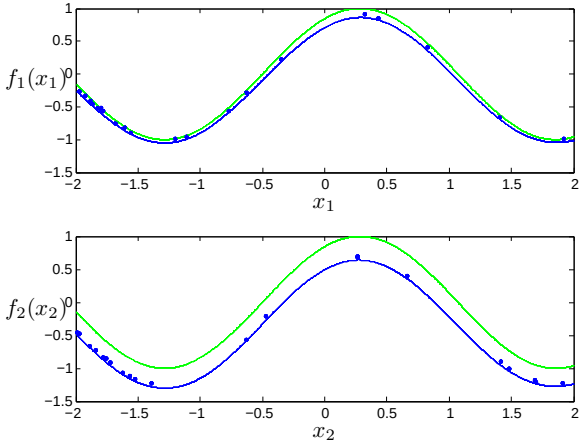
True Function
& data locations



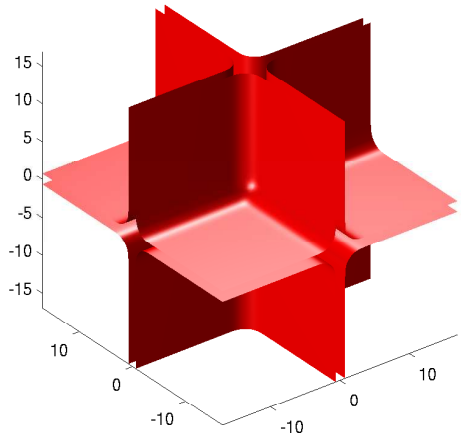
Squared-exp GP
posterior mean



Additive GP
posterior mean

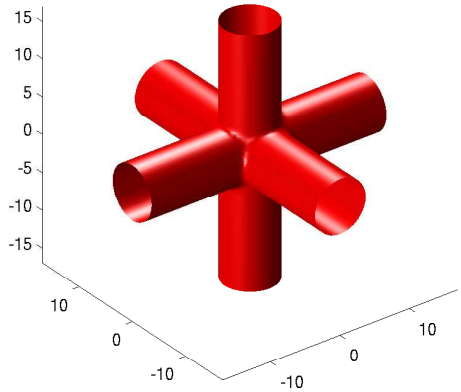


Additive GP
1st-order functions



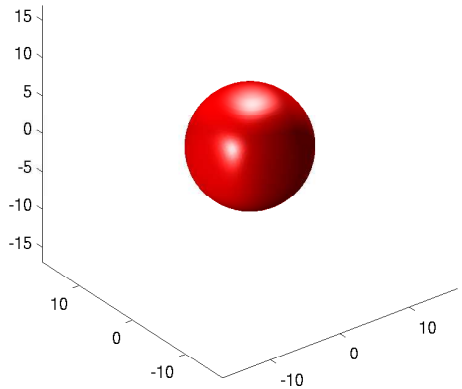
1st order interactions

$k_1 + k_2 + k_3$



2nd order interactions

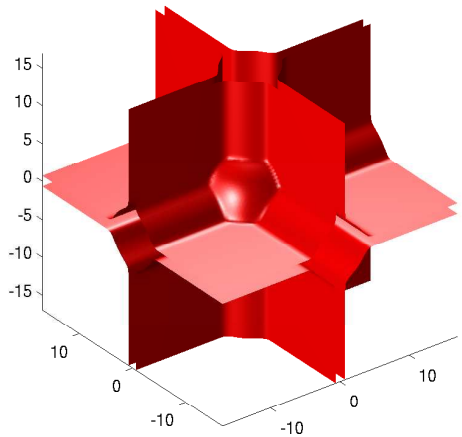
$k_1 k_2 + k_2 k_3 + k_1 k_3$



3rd order interactions

$k_1 k_2 k_3$

(Squared-exp kernel)



All interactions

$k_1 + k_2 + \cdots + k_1 k_2 k_3$

(Additive kernel)

Results

Table: Regression Mean Squared Error

Method	bach	concrete	pumadyn	servo	housing
Linear	1.031	0.404	0.641	0.523	0.289
GP GAM	1.259	0.149	0.598	0.281	0.161
HKL	**0.199**	0.147	0.346	0.199	0.151
GP sq-exp	**0.045**	0.157	**0.317**	**0.126**	**0.092**
GP Additive	**0.045**	**0.089**	**0.316**	**0.110**	**0.102**

Table: Regression Negative Log Likelihood

Method	bach	concrete	pumadyn	servo	housing
Linear	2.430	1.403	1.881	1.678	1.052
GP GAM	1.708	0.467	1.195	0.800	0.457
GP sq-exp	**−0.131**	0.398	**0.843**	0.429	**0.207**
GP Additive	**−0.131**	**0.114**	**0.841**	**0.309**	**0.194**

Summary

- Additive GPs generalize commonly used GPs and GAMs. Only penalty is time and R extra hyperparameters.

Summary

- Additive GPs generalize commonly used GPs and GAMs. Only penalty is time and R extra hyperparameters.
- Add a lot of tractable, intepretable structure to your model.

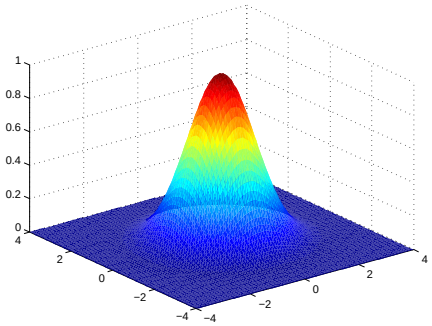
Summary

- Additive GPs generalize commonly used GPs and GAMs. Only penalty is time and R extra hyperparameters.
- Add a lot of tractable, intepretable structure to your model.
- Allows better generalization if the data supports it.

A puzzle

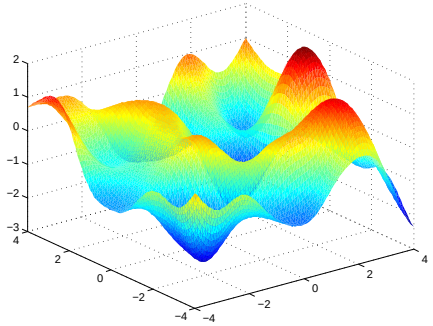
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$

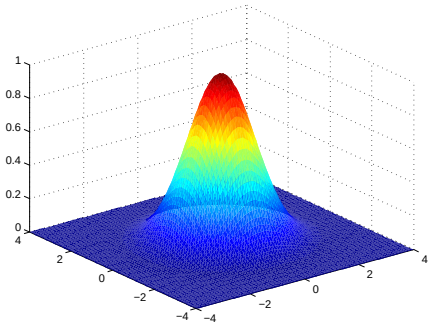


Draw from a GP with a 2D Sq-exp kernel

A puzzle

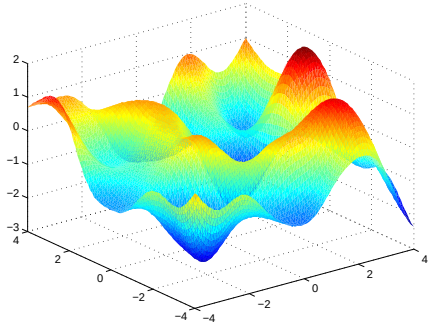
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$



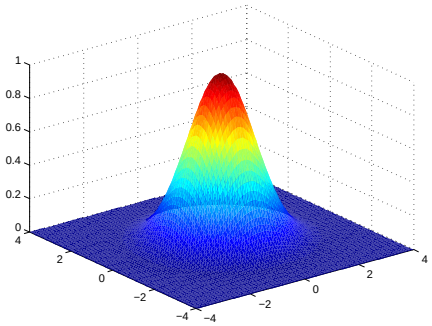
Draw from a GP with a 2D Sq-exp kernel

- Draw train and test points from a Gaussian centered at 0

A puzzle

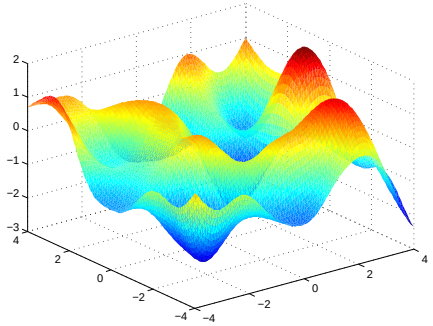
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$



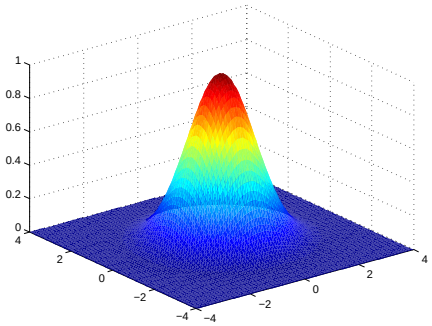
Draw from a GP with a 2D Sq-exp kernel

- Draw train and test points from a Gaussian centered at 0
- Predict test points from training points

A puzzle

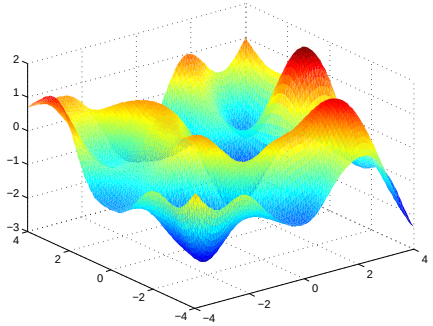
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$



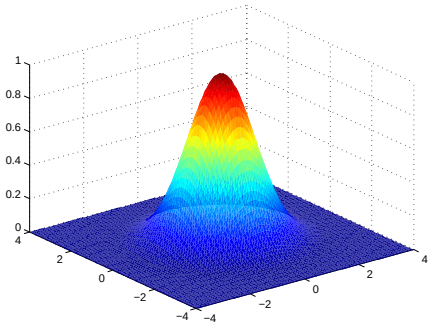
Draw from a GP with a 2D Sq-exp kernel

- Draw train and test points from a Gaussian centered at 0
- Predict test points from training points
- How many training points needed to learn?

A puzzle

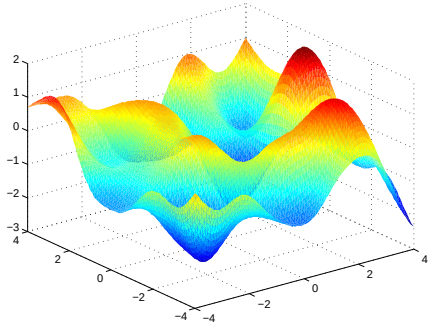
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$



Draw from a GP with a 2D Sq-exp kernel

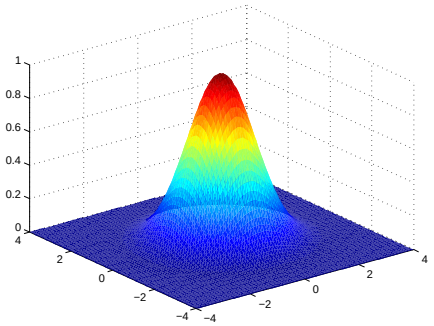
- Draw train and test points from a Gaussian centered at 0
- Predict test points from training points
- How many training points needed to learn?

Learning a high-dimensional function from this model class requires exponentially many training points.

A puzzle

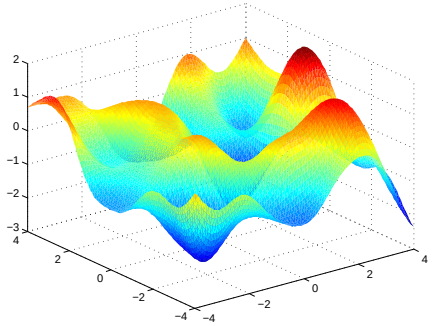
An experiment Carl did:

- Draw an 8-dimensional function from a GP with a sq-exp kernel



2D Sq-exp kernel

$\rightarrow$



Draw from a GP with a 2D Sq-exp kernel

- Draw train and test points from a Gaussian centered at 0
- Predict test points from training points
- How many training points needed to learn?

Learning a high-dimensional function from this model class requires exponentially many training points.
How is it that sq-exp GP regression actually works on high-dimensional functions?

The end!

The end!

Ideas and criticism welcome.

T.A. Plate. Accuracy versus interpretability in flexible modeling: Implementing a tradeoff using Gaussian process models. *Behaviormetrika*, 26:29–50, 1999. ISSN 0385-7417.

V.N. Vapnik. *Statistical learning theory*, volume 2. Wiley New York, 1998.

G. Wahba. *Spline models for observational data*. Society for Industrial Mathematics, 1990. ISBN 0898712440.