Probabilistic Numerics

I – Introduction to Gaussian Regression

Philipp Hennig

Dobbiaco Summer School
19 June 2017

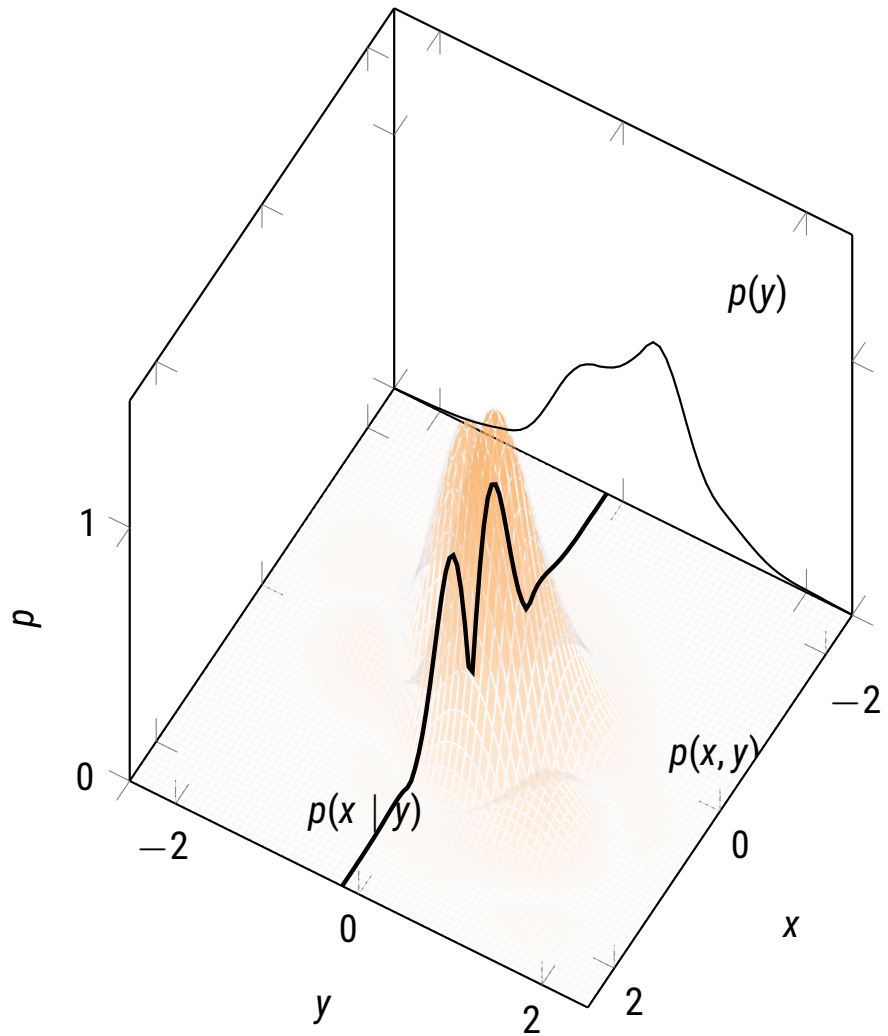
MAX-PLANCK-GESELLSCHAFT

Research Group for Probabilistic Numerics
Max Planck Institute for Intelligent Systems
Tübingen, Germany

Some of the presented work was supported by
the Emmy Noether Programme of the DFG

Probabilistic Inference

An explicit framework for dealing with uncertainty



- probability for $X = x$ AND $Y = y$:

$$p(x,y) \text{ s.t. } \int p(x,y)\,dx\,dy = 1$$

- sum rule: $\underbrace{p(y)}_{\text{marginal}} = \int \underbrace{p(x,y)}_{\text{joint}}\,dx$
- product rule: $p(x,y) = p(y)\,\underbrace{p(x \mid y)}_{\text{conditional}}$
- Bayes' Theorem:

$$\underbrace{p(x \mid y)}_{\text{posterior}} = \frac{\overbrace{p(y \mid x)}^{\text{likelihood}} \cdot \overbrace{p(x)}^{\text{prior}}}{\underbrace{p(y)}_{\text{evidence}}}$$

Grand Plan:

- today:
  - **Gaussian distributions**
    as a computationally efficient (linear) algebra of uncertainty
  - **computing** under uncertainty:
    integration and **ordinary differential equations**
- tomorrow:
  - **linear algebra** as inference: the computational limits of tracking uncertainty
  - **beyond classic methods**: new probabilistic functionality for contemporary computational challenges:
    - local and global optimization under uncertainty
    - propagating uncertainty through computational pipelines
    - detecting inappropriate models

Computation is inference on unknown quantities, using silicon as a data source.
Uncertainty deserves a central role.

Gaussian Distributions

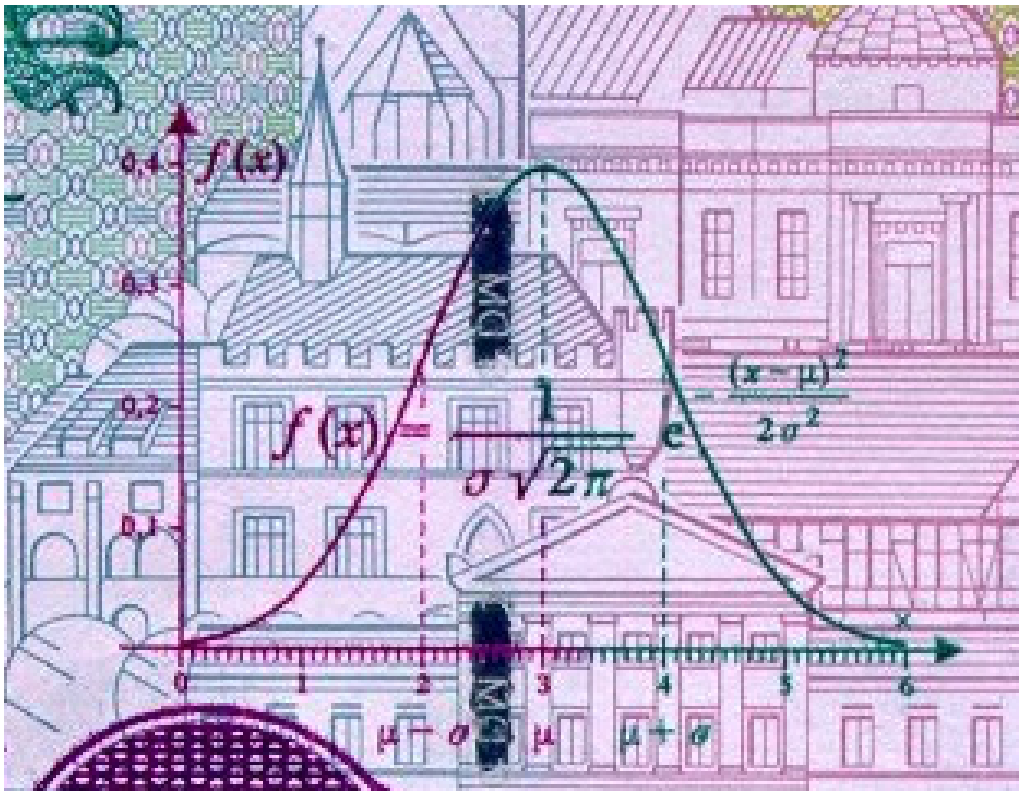
a central concept

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^{\intercal}\Sigma^{-1}(x-\mu)\right]$$

Gaussian Distributions

a central concept

$$\mathcal{N}(x; \mu, \Sigma) = \frac{1}{|2\pi\Sigma|^{1/2}} \exp\left[-\frac{1}{2}(x-\mu)^{\intercal}\Sigma^{-1}(x-\mu)\right]$$



Coming up:

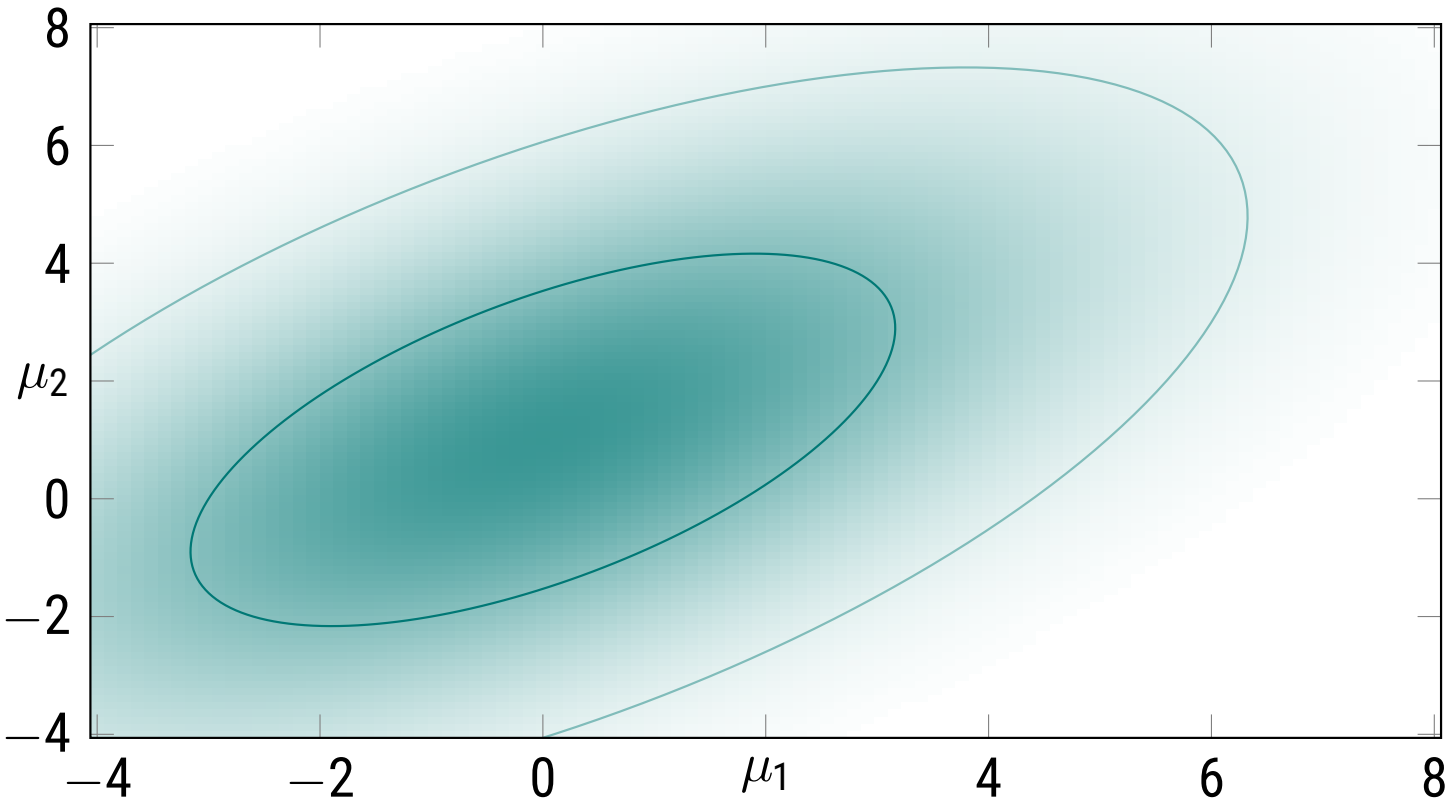
- **Gaussians** are to **inference** as **linear maps** are to **algebra**
- **Numerical computation** is about mapping intractable problems onto **linear** (or discrete) ones. Gaussians provide the algebra to achieve this in the context of **uncertainty**

Closure under Multiplication

multiple Gaussian factors form a Gaussian

$$\mathcal{N}(x; a, A)\mathcal{N}(x; b, B) = \mathcal{N}(x; c, C)Z$$

$$C := (A^{-1} + B^{-1})^{-1} \qquad c := C(A^{-1}a + B^{-1}b) \qquad Z := \mathcal{N}(a; b, A + B)$$

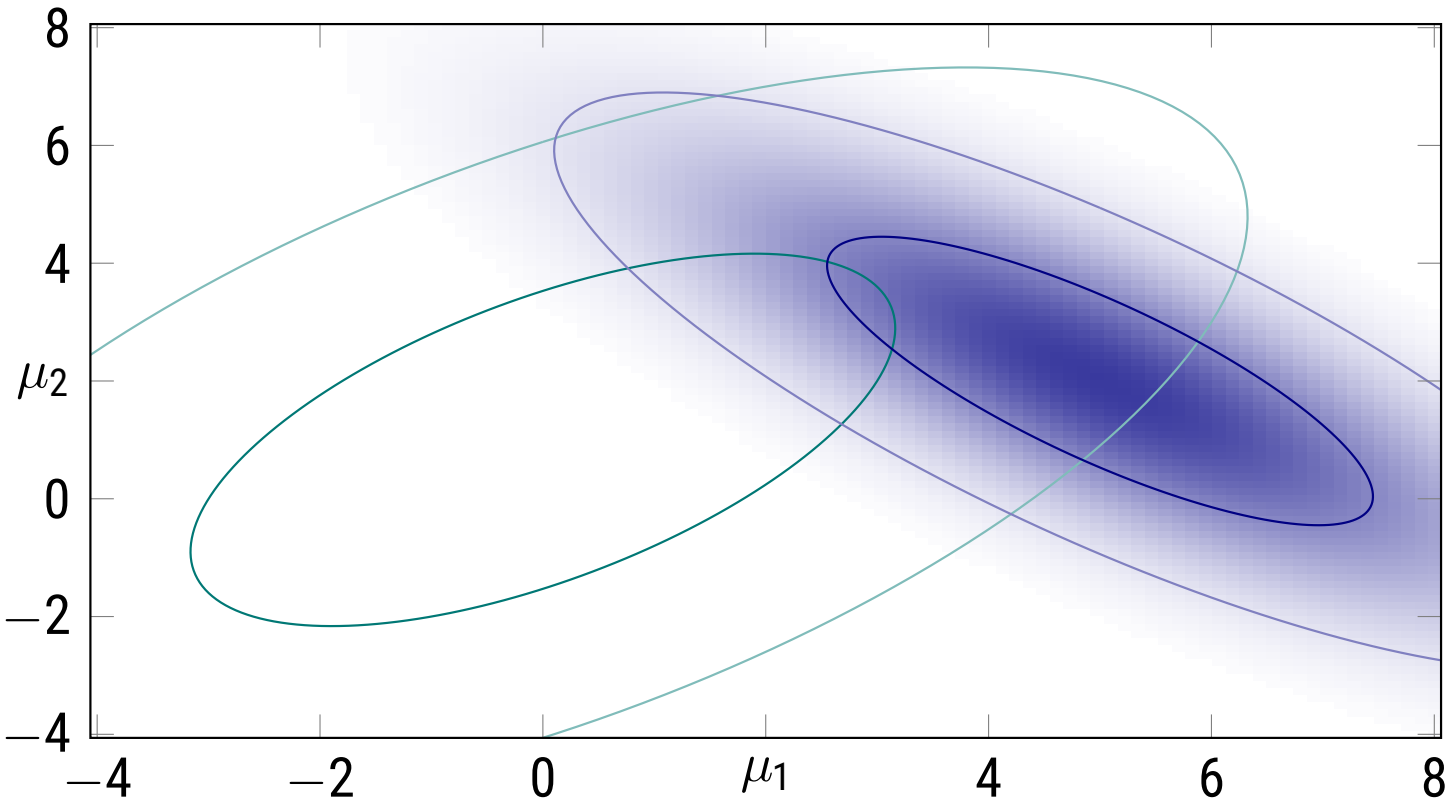


Closure under Multiplication

multiple Gaussian factors form a Gaussian

$$\mathcal{N}(x; a, A)\mathcal{N}(x; b, B) = \mathcal{N}(x; c, C)Z$$

$$C := (A^{-1} + B^{-1})^{-1} \qquad c := C(A^{-1}a + B^{-1}b) \qquad Z := \mathcal{N}(a; b, A + B)$$

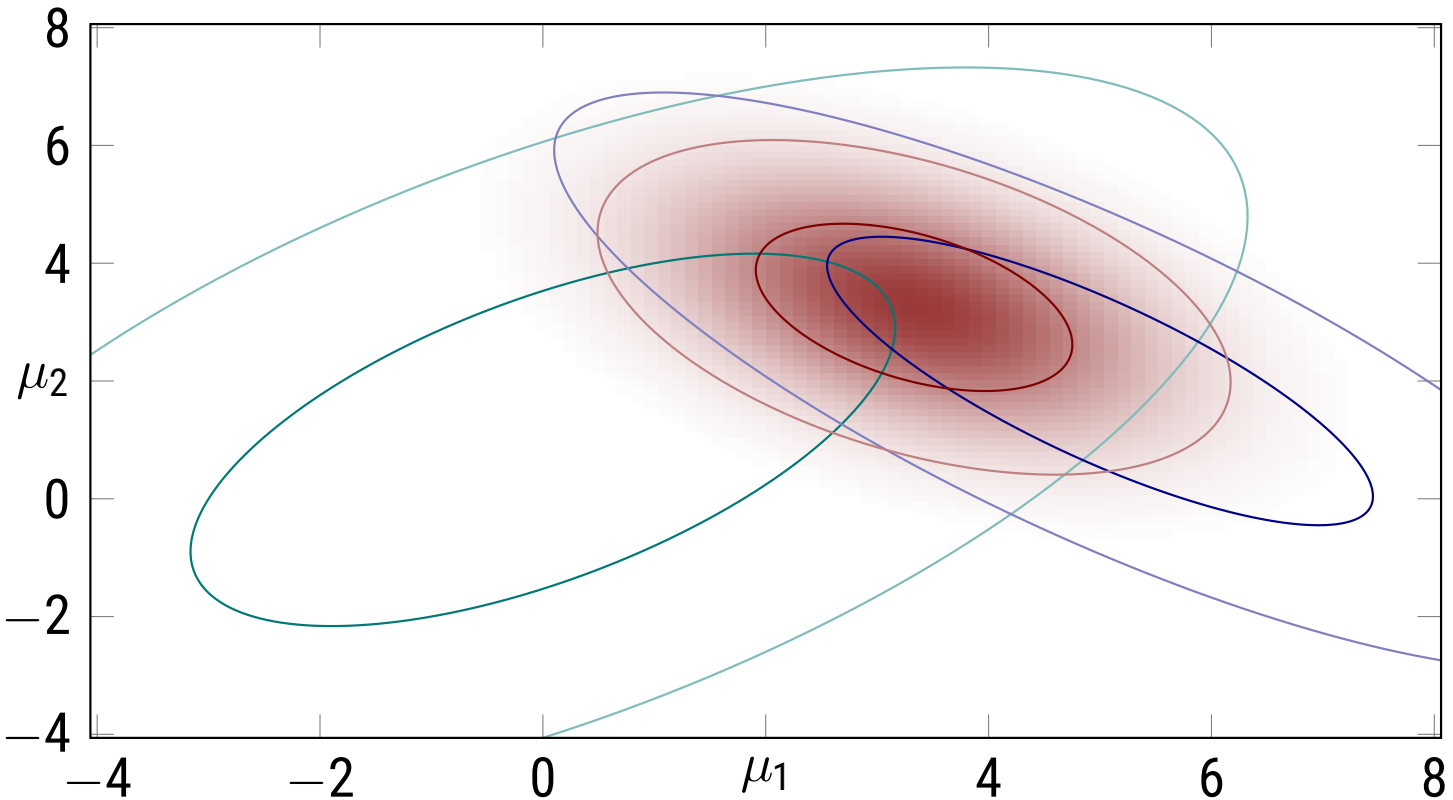


Closure under Multiplication

multiple Gaussian factors form a Gaussian

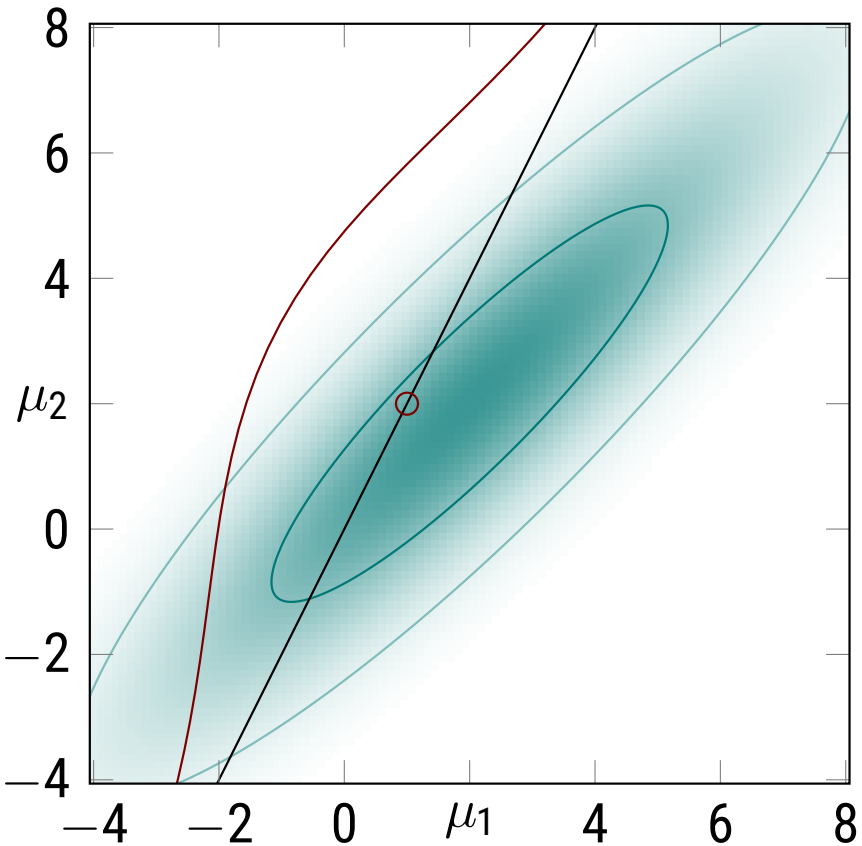
$$\mathcal{N}(x; a, A)\mathcal{N}(x; b, B) = \mathcal{N}(x; c, C)Z$$

$$C := (A^{-1} + B^{-1})^{-1} \qquad c := C(A^{-1}a + B^{-1}b) \qquad Z := \mathcal{N}(a; b, A + B)$$



Closure under Linear Maps

linear maps of Gaussians are Gaussians



$$p(z) = \mathcal{N}(z; \mu, \Sigma)$$

$$\Rightarrow \quad p(Az) = \mathcal{N}(Az, A\mu, A\Sigma A^\intercal)$$

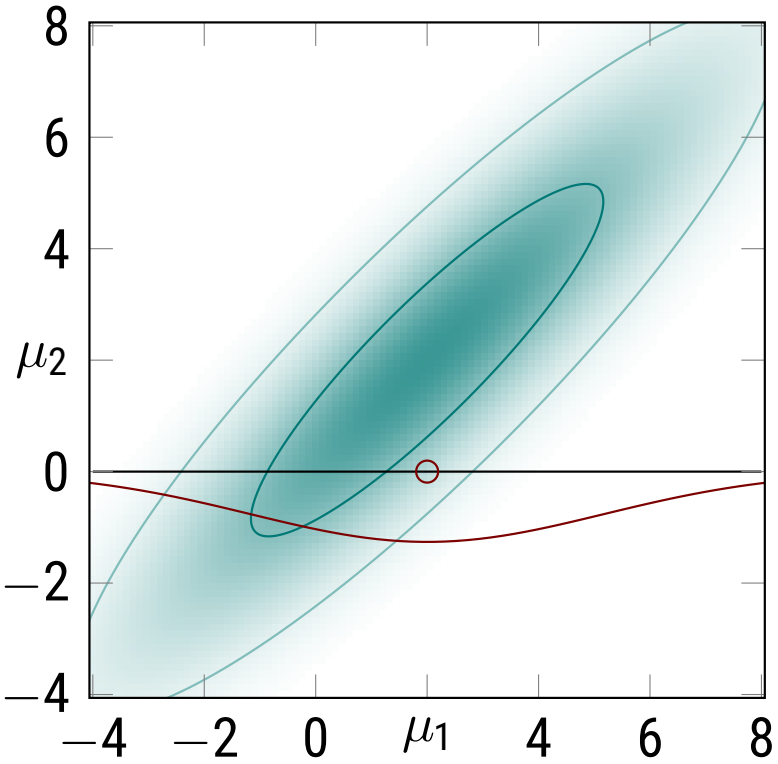
Here: $A = [1, -0.5]$

Closure under Marginalization

projections of Gaussians are Gaussian

- projection with $A = \begin{pmatrix} 1 & 0 \end{pmatrix}$

$$\int \mathcal{N}\left[\begin{pmatrix} x \\ y \end{pmatrix}; \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix}\right] dy = \mathcal{N}(x; \mu_x, \Sigma_{xx})$$



- this is the sum rule

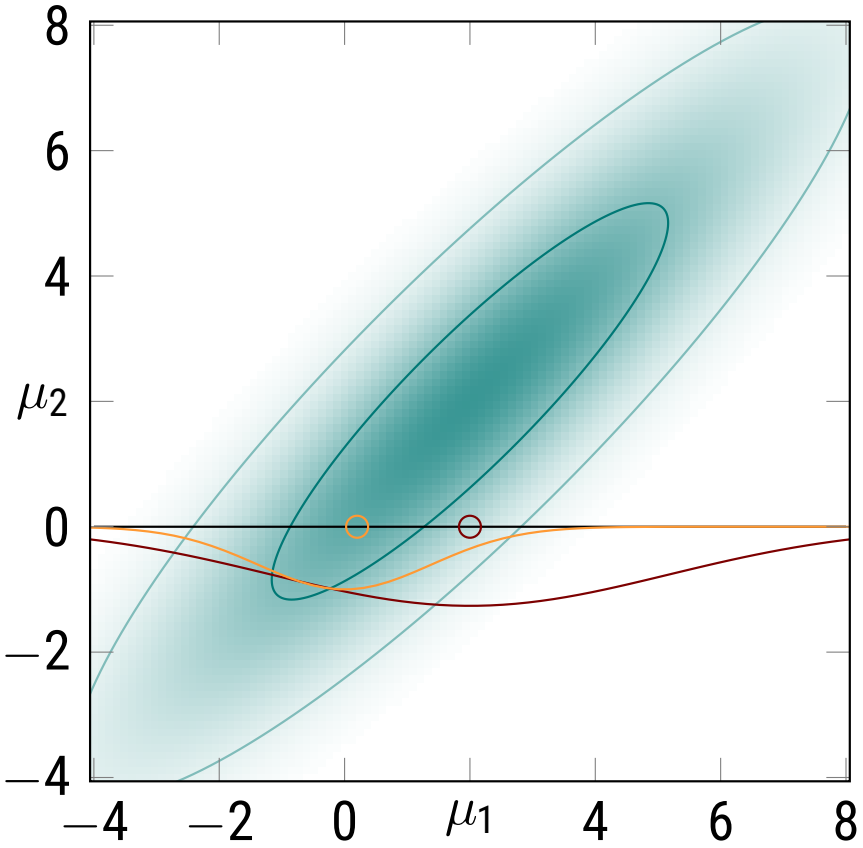
$$\int p(x, y)\, dy = \int p(y \mid x) p(x)\, dy = p(x)$$

- so every finite-dim Gaussian is a marginal of infinitely many more

Closure under Conditioning

cuts through Gaussians are Gaussians

$$p(x \mid y) = \frac{p(x,y)}{p(y)} = \mathcal{N}\left(x; \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y), \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}\right)$$



- this is the product rule
- so Gaussians are closed under the rules of probability

Gaussians provide the linear algebra of inference

if all joints are Gaussian and all observations are linear, all posteriors are Gaussian

- products of Gaussians are Gaussians $\quad C := (A^{-1} + B^{-1})^{-1} \quad c := C(A^{-1}a + B^{-1}b)$

$$\mathcal{N}(x; a, A)\mathcal{N}(x; b, B) = \mathcal{N}(x; c, C)\mathcal{N}(a; b, A + B)$$

- marginals of Gaussians are Gaussians

$$\int \mathcal{N}\left[\begin{pmatrix} x \\ y \end{pmatrix}; \begin{pmatrix} \mu_x \\ \mu_y \end{pmatrix}, \begin{pmatrix} \Sigma_{xx} & \Sigma_{xy} \\ \Sigma_{yx} & \Sigma_{yy} \end{pmatrix}\right] dy = \mathcal{N}(x; \mu_x, \Sigma_{xx})$$

- (linear) conditionals of Gaussians are Gaussians

$$p(x \mid y) = \frac{p(x, y)}{p(y)} = \mathcal{N}\left(x; \mu_x + \Sigma_{xy}\Sigma_{yy}^{-1}(y - \mu_y), \Sigma_{xx} - \Sigma_{xy}\Sigma_{yy}^{-1}\Sigma_{yx}\right)$$

- linear projections of Gaussians are Gaussians

$$p(z) = \mathcal{N}(z; \mu, \Sigma) \quad \Rightarrow \quad p(Az) = \mathcal{N}(Az, A\mu, A\Sigma A^{\mathsf{T}})$$

Bayesian inference becomes linear algebra

$$p(x) = \mathcal{N}(x; \mu, \Sigma) \qquad p(y \mid x) = \mathcal{N}(y; A^{\mathsf{T}}x + b, \Lambda)$$

$$p(B^{\mathsf{T}}x + c \mid y) = \mathcal{N}[B^{\mathsf{T}}x + c; B^{\mathsf{T}}\mu + c + B^{\mathsf{T}}\Sigma A(A^{\mathsf{T}}\Sigma A + \Lambda)^{-1}(y - A^{\mathsf{T}}\mu - b),$$
$$B^{\mathsf{T}}\Sigma B - B^{\mathsf{T}}\Sigma A(A^{\mathsf{T}}\Sigma A + \Lambda)^{-1}A^{\mathsf{T}}\Sigma B]$$

- **Gaussians** are to **inference** as **linear maps** are to **algebra**
- Gaussian measures turn probabilistic inference into **linear algebra**

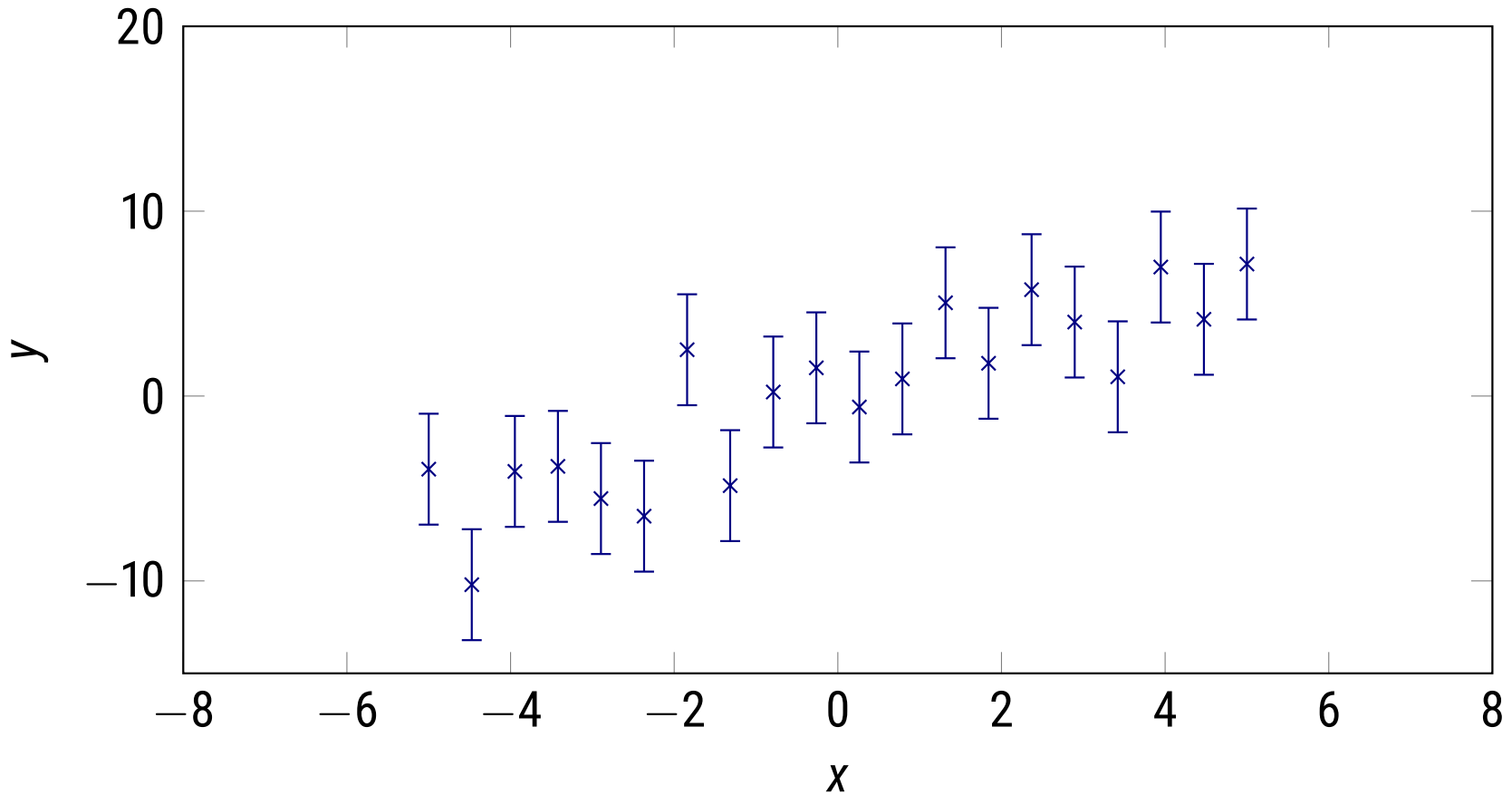
Next up:

- Gaussian measures can be used to infer **functions** and **curves**

A dataset

linear regression

given $y \in \mathbb{R}^N$, $p(y \mid f)$, what is f?

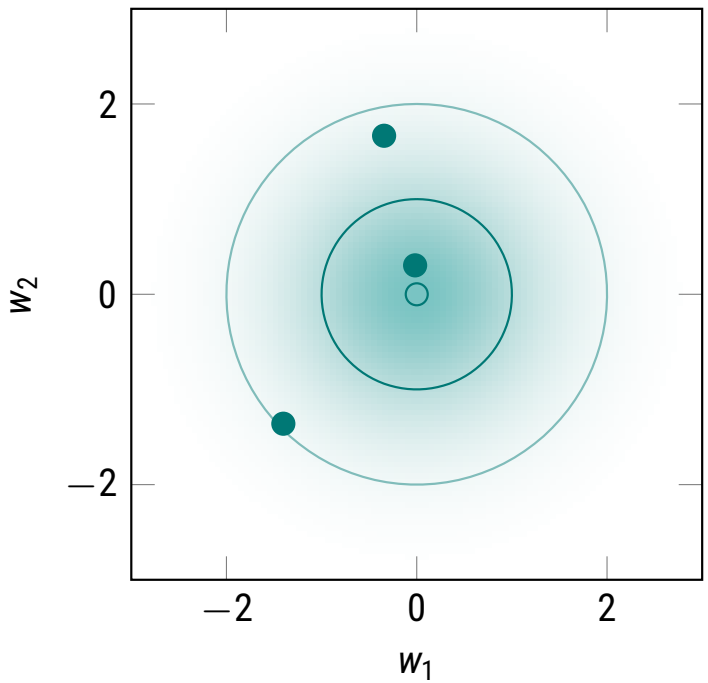


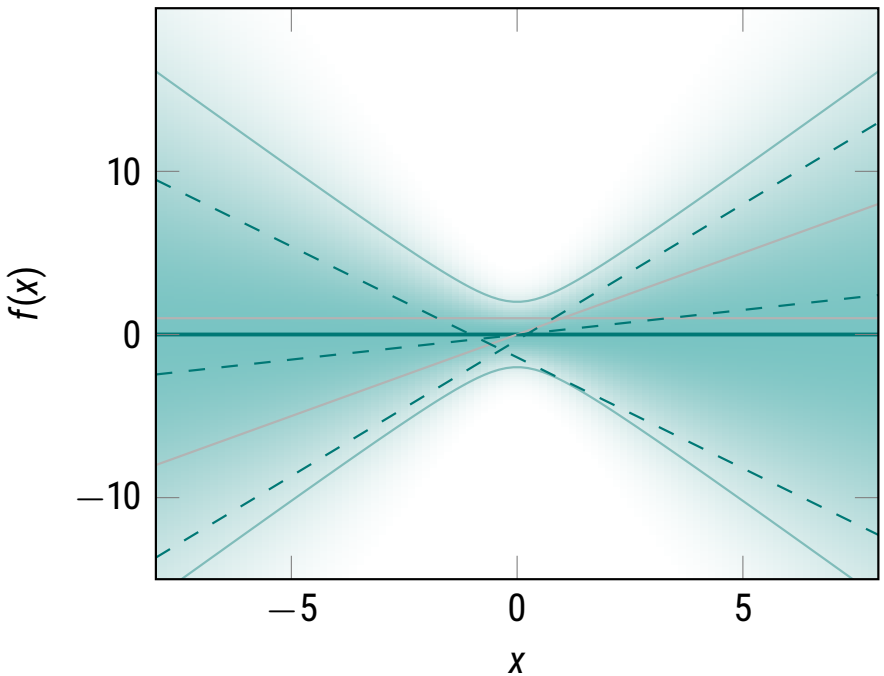
A prior

over linear functions

$$f(x) = w_1 + w_2 x = \phi_x^\intercal w \qquad \phi_x = \begin{pmatrix} 1 \\ x \end{pmatrix}$$

$$p(w) = \mathcal{N}(w; \mu, \Sigma) \qquad p(f) = \mathcal{N}(f; \phi_x^\intercal \mu, \phi_x^\intercal \Sigma \phi_x)$$



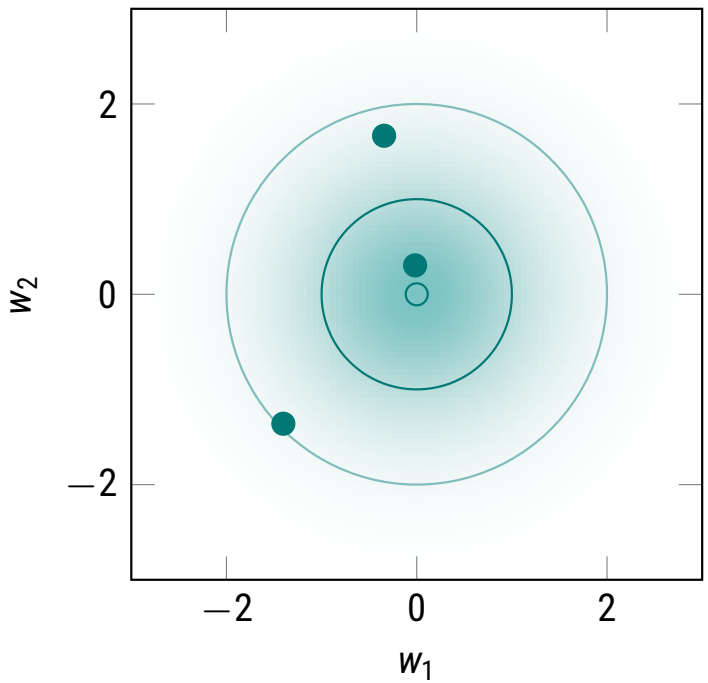


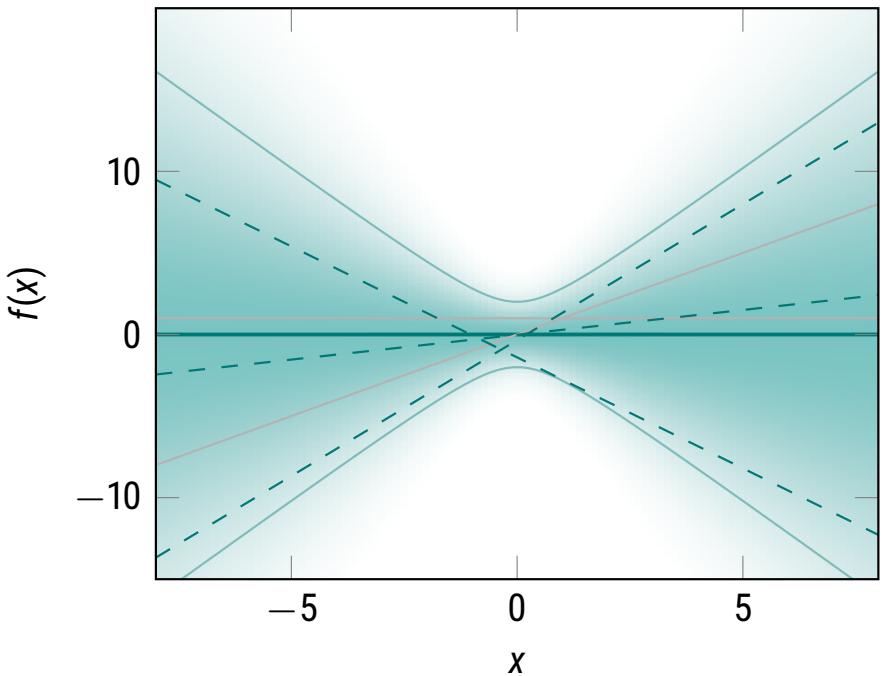
A prior

over linear functions

$$f(x) = w_1 + w_2 x = \phi_x^\mathsf{T} w \qquad \phi_x = \begin{pmatrix} 1 \\ x \end{pmatrix}$$

$$p(w) = \mathcal{N}(w; \mu, \Sigma) \qquad p(f) = \mathcal{N}(f; \phi_x^\mathsf{T} \mu, \phi_x^\mathsf{T} \Sigma \phi_x)$$



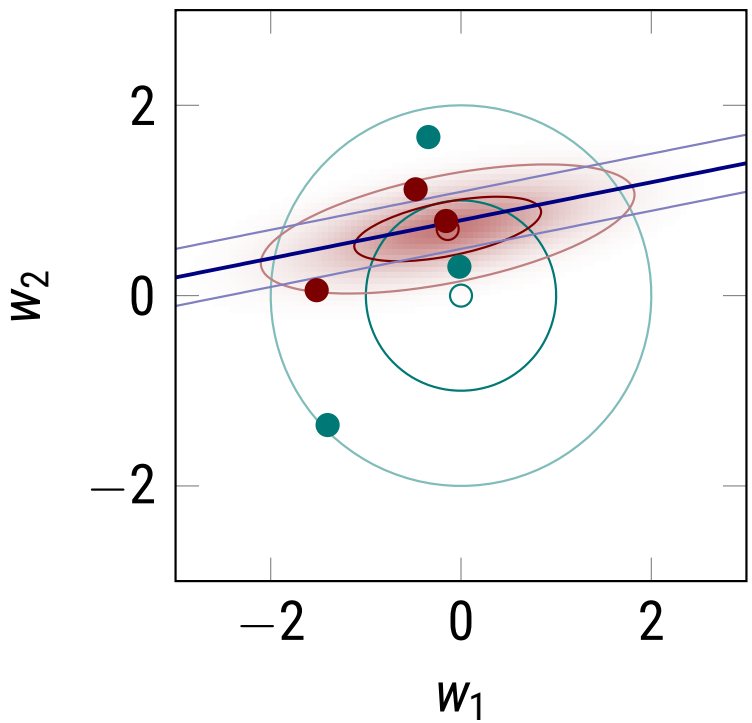


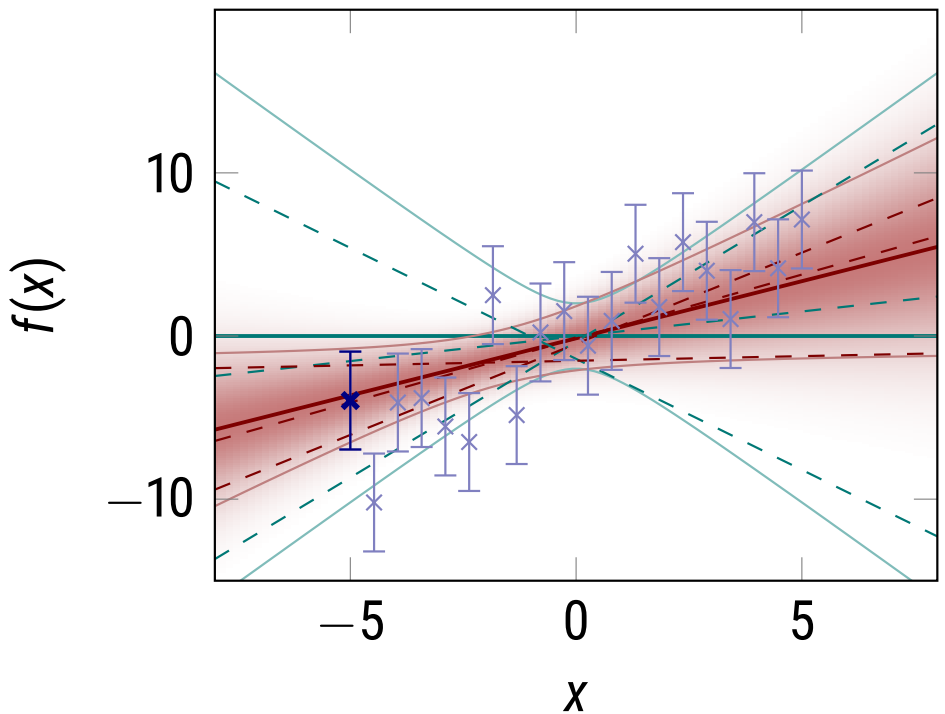
The posterior

over weights and functions

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),$$
$$\phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_x)$$



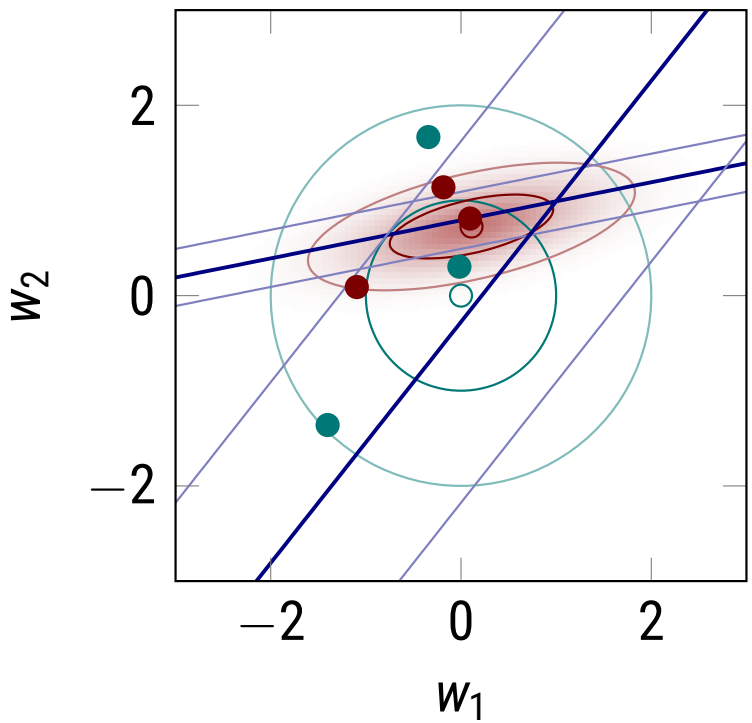


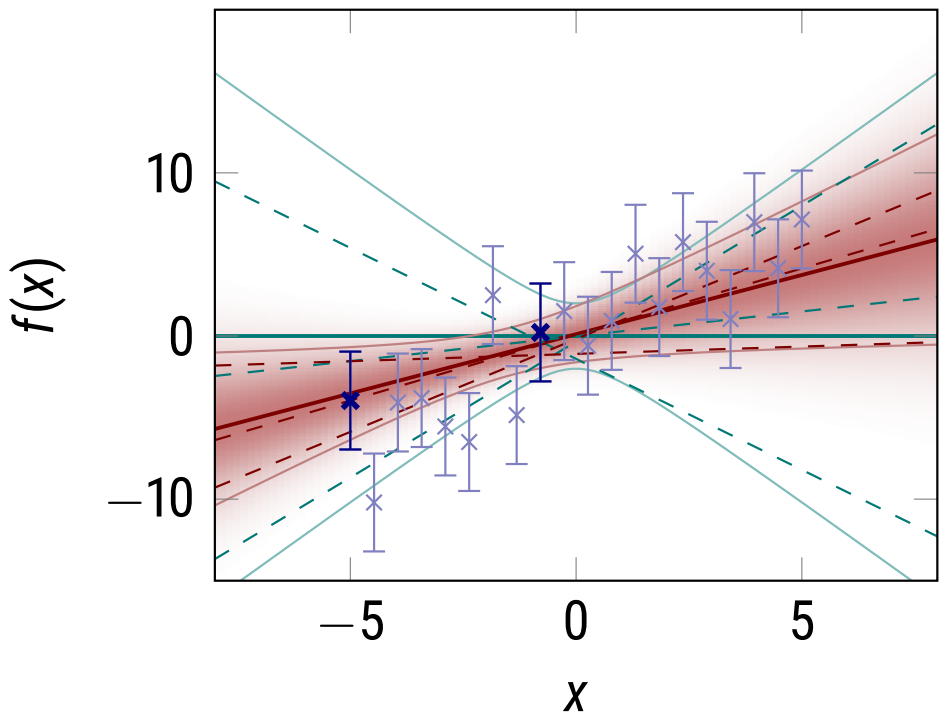
The posterior

over weights and functions

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),$$
$$\phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_x)$$



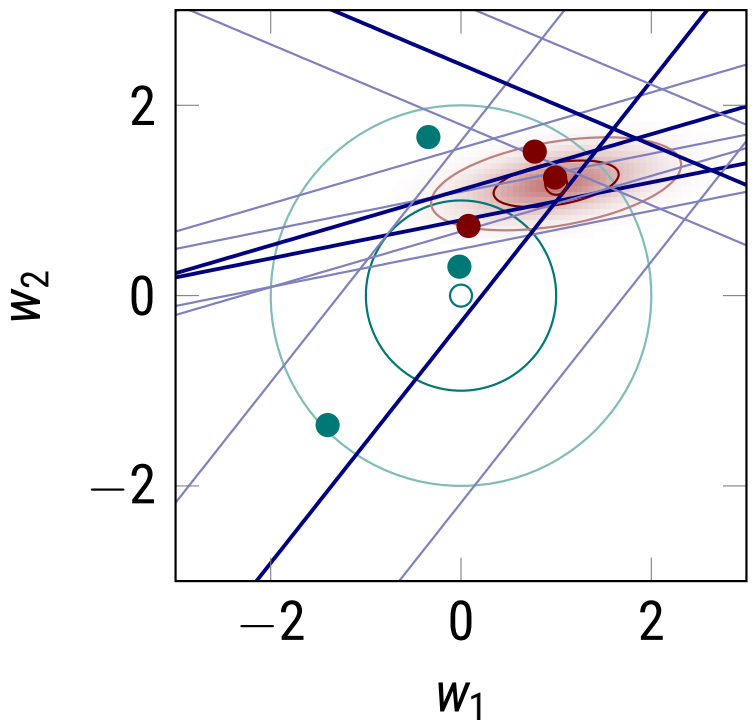


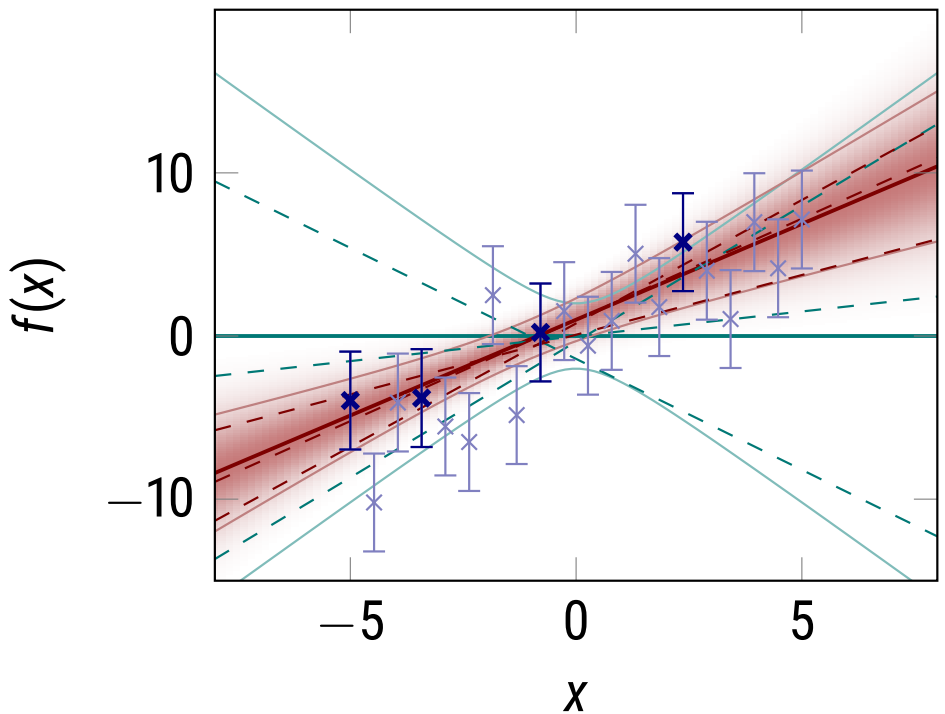
The posterior

over weights and functions

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),$$
$$\phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_x)$$



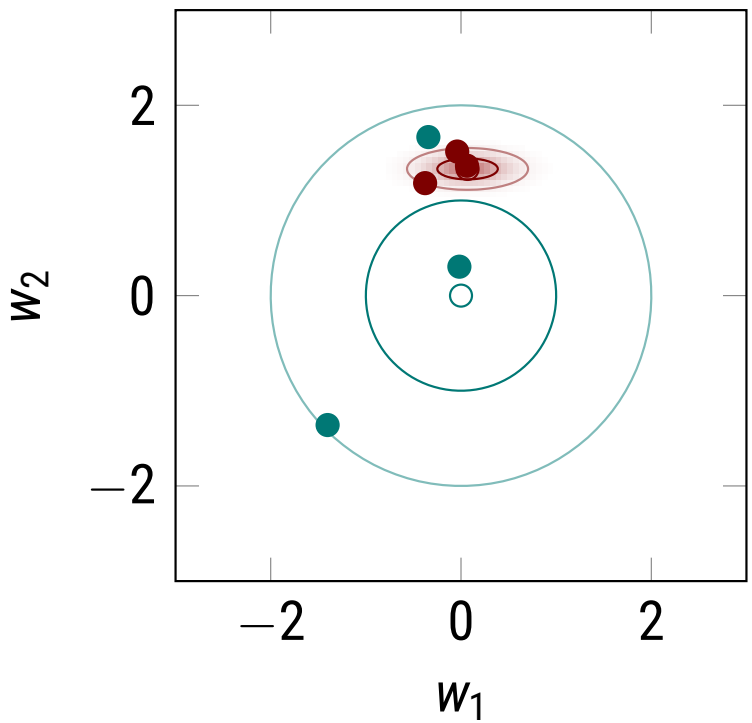


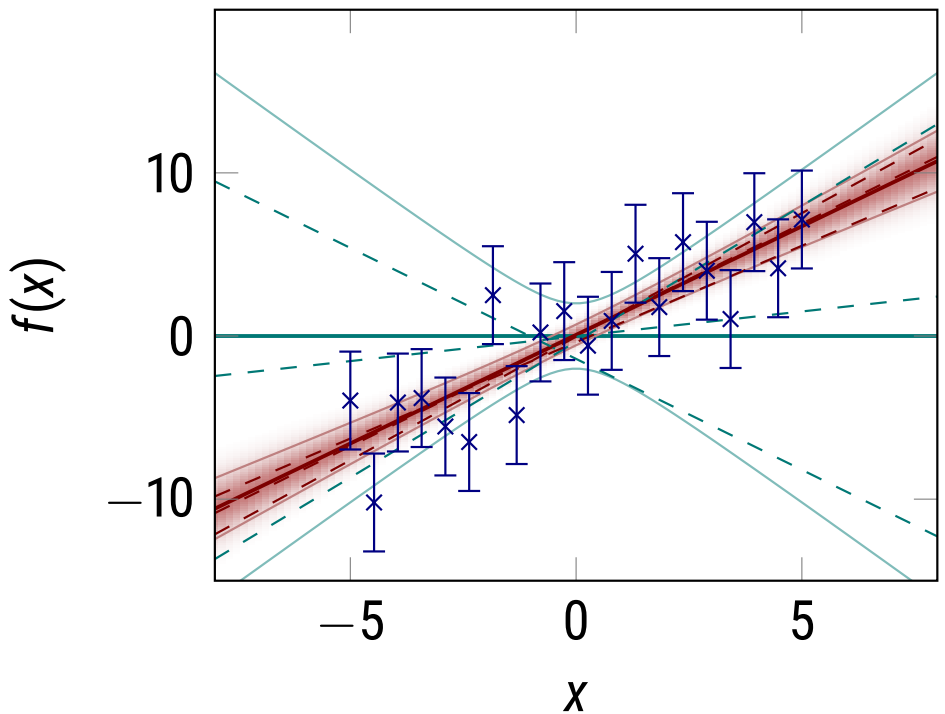
The posterior

over weights and functions

$$p(y \mid w, \phi_X) = \mathcal{N}(y; \phi_X^\intercal w, \sigma^2 I)$$

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),$$
$$\phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_x)$$





Code

python

```
#                              ==== PARAMETRIC GAUSSIAN REGRESSION IN PYTHON ====
# prior on w
F     = 2                                                  # number of features
def phi(a) : return power(a,range(F))                          # phi(a) = [1,a]
mu    = zeros((F,1))
Sigma = eye(F)                                             # p(w)=N(mu,Sigma)

# implied prior on f_x
n     = 100
x     = reshape(linspace(-8,8,n),(n,1))     # reshape is needed for phi to work
m     = dot(phi(x),mu)
kxx   = dot(dot(phi(x),Sigma),phi(x).transpose())            # p(f_x)=N(m,k_xx)
s     = dot(cholesky(kxx + 1e-6 * eye(n)),randn(n,3)) + m       # prior samples
stdpi = reshape(sqrt(diag(kxx)),[n,1])          # marginal stddev, for plotting

# loading data from disk
import scipy.io; data = scipy.io.loadmat('data.mat')
X = data['X']; Y = data['Y']; sigma = data['sigma']; N  = len(X)

# prior on Y = f_X + eps
M     = dot(phi(X),mu)
kXX   = dot(dot(phi(X),Sigma),phi(X).transpose())          # p(f_X) = N(M,k_XX)

# inference
G     = cho_factor(kXX + sigma**2 * eye(N))       # Most expensive step, O(N^3)
kxX   = dot(dot(phi(x),Sigma),phi(X).transpose())         # Cov(f_x,f_X) = k_xX
A     = cho_solve(G,kxX.transpose()).transpose()       # pre-compute for re-use

# posterior p(f_x|Y)
mpost = m + dot(A,(Y-M))                                                # mean
vpost = kxx - dot(A,kxX.transpose())                                # covariance
spost = dot(cholesky(vpost + 1e-8 * eye(n)),randn(n,3)) + mpost       # samples
stdpo = reshape(sqrt(diag(vpost)),[n,1])        # marginal stddev, for plotting
```

Code

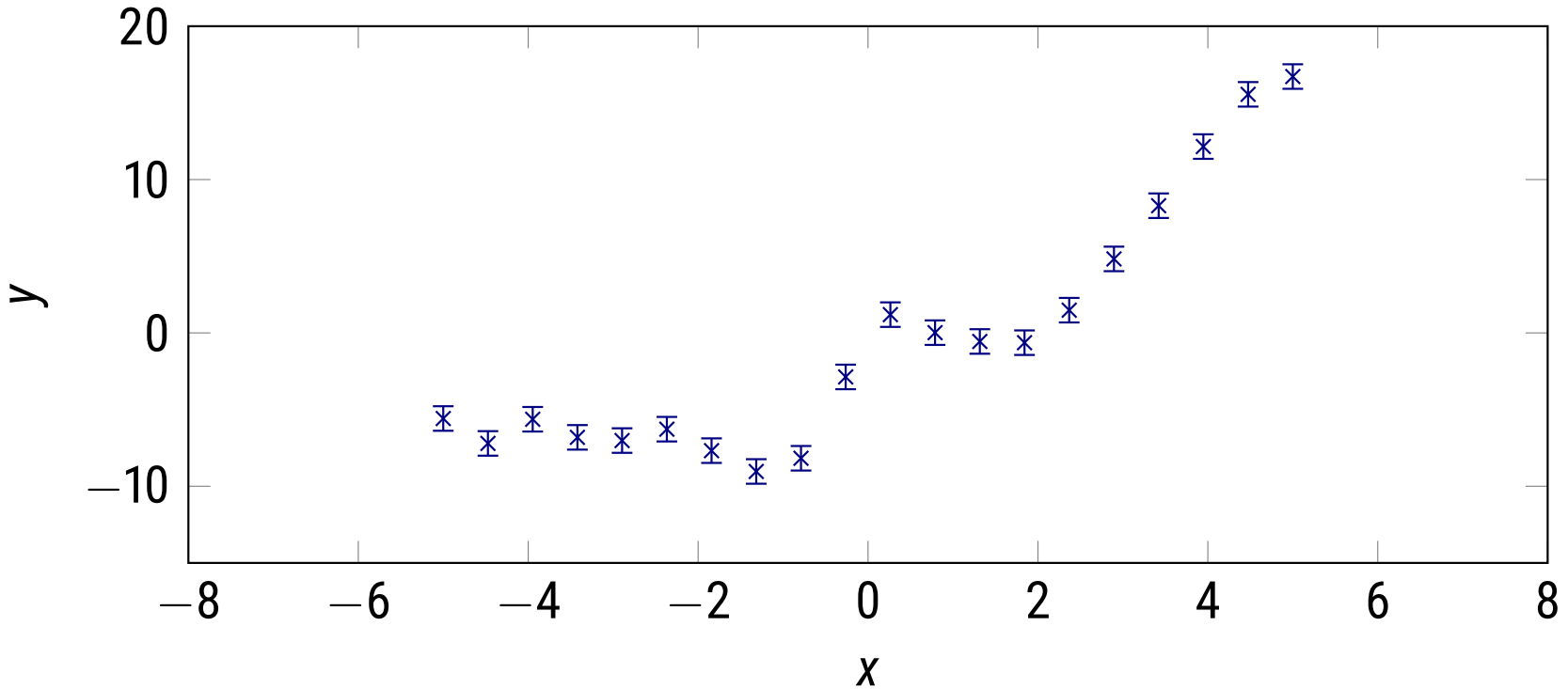
matlab

```
%                              ==== PARAMETRIC GAUSSIAN REGRESSION IN MATLAB ====
% prior on w
F     = 2;                                                  % number of features
phi   = @(a)(bsxfun(@power,a,0:F-1));                            % phi(a)=[1;a]
mu    = zeros(F,1);
Sigma = eye(F);                                             % p(w) = N(mu,Sigma)

% implied prior on f_x
n     = 100;
x     = linspace(-8,8,n)';                                      % `test' points
m     = phi(x) * mu;
kxx   = phi(x) * Sigma * phi(x)';                             % p(f_x)=N(m,k_xx)
s     = bsxfun(@plus,m,chol(kxx + 1.0e-8 * eye(n))' * randn(n,3));    % samples
stdpi = sqrt(diag(kxx));                              % marginal stddev, for plotting

% loading data from disk
load('data.mat'); N = length(Y);                               % gives Y,X,sigma

% prior on Y=f_X + epsilon
M     = phi(X) * mu;
kXX   = phi(X) * Sigma * phi(X)';                             % p(f_X) = N(M,k_XX)

% inference
G     = chol(kXX + sigma^ 2 * eye(N);)            % most expensive step: O(N^3)
kxX   = phi(x) * Sigma * phi(X)';                             % Cov(f_x,f_X) = k_xX
A     = kxX / G;                                    % pre-compute for re-use

% posterioe p (f_x|Y)
mpost = m + A * (G' \ (Y-M));                                              % mean
vpost = kxx - A * A';                                                % covariance
spost = bsxfun(@plus,mpost,chol(vpost + 1.0e-8 * eye(n))' * randn(n,3)) % smpls
stdpo = sqrt(diag(vpost));                            % marginal stddev, for plotting
```

A More Realistic Dataset

General Linear Regression

$$f(x) = \phi_x^\mathsf{T} w \qquad ?$$



$$f(x) = w_1 + w_2 x = \phi_x^\top w$$

$$\phi_x := \begin{pmatrix} 1 \\ x \end{pmatrix}$$

No big deal!

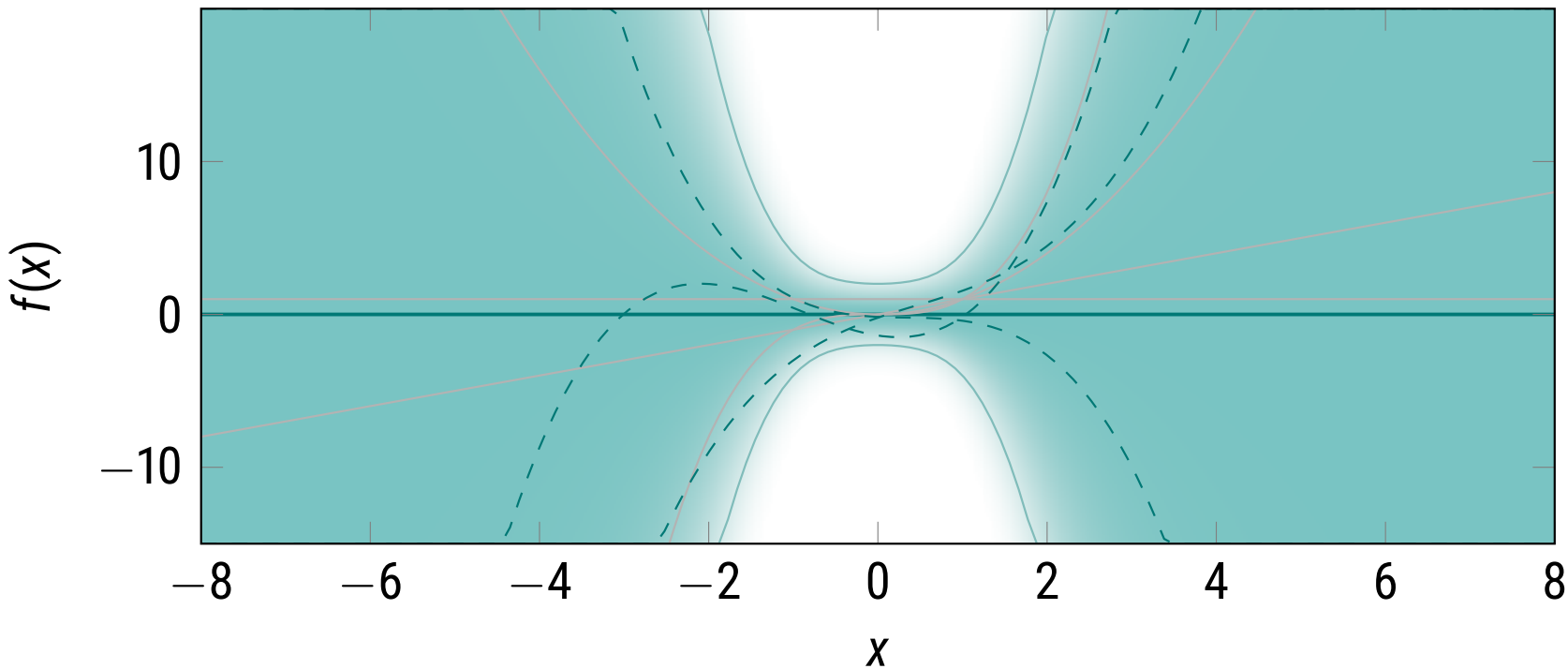
a one-line change

```
#                              ==== PARAMETRIC GAUSSIAN REGRESSION IN PYTHON ====
# prior on w
F     = 4                                                    # number of features
def phi(a) : return power(a,range(F))                            # phi(a) = [1,a]
mu    = zeros((F,1))
Sigma = eye(F)                                                 # p(w)=N(mu,Sigma)

# implied prior on f_x
n     = 100
x     = reshape(linspace(-8,8,n),(n,1))     # reshape is needed for phi to work
m     = dot(phi(x),mu)
kxx   = dot(dot(phi(x),Sigma),phi(x).transpose())              # p(f_x)=N(m,k_xx)
s     = dot(cholesky(kxx + 1e-6 * eye(n)),randn(n,3)) + m       # prior samples
stdpi = reshape(sqrt(diag(kxx)),[n,1])           # marginal stddev, for plotting

# loading data from disk
import scipy.io; data = scipy.io.loadmat('data.mat')
X = data['X']; Y = data['Y']; sigma = data['sigma']; N  = len(X)

# prior on Y = f_X + eps
M     = dot(phi(X),mu)
kXX   = dot(dot(phi(X),Sigma),phi(X).transpose())            # p(f_X) = N(M,k_XX)

# inference
G     = cho_factor(kXX + sigma**2 * eye(N))       # Most expensive step, O(N^3)
kxX   = dot(dot(phi(x),Sigma),phi(X).transpose())          # Cov(f_x,f_X) = k_xX
A     = cho_solve(G,kxX.transpose()).transpose()       # pre-compute for re-use

# posterior p(f_x|Y)
mpost = m + dot(A,(Y-M))                                                 # mean
vpost = kxx - dot(A,kxX.transpose())                                # covariance
spost = dot(cholesky(vpost + 1e-8 * eye(n)),randn(n,3)) + mpost       # samples
stdpo = reshape(sqrt(diag(vpost)),[n,1])         # marginal stddev, for plotting
```

Cubic Regression

```
phi = @(a)(bsxfun(@power,a,[0:3]));
```

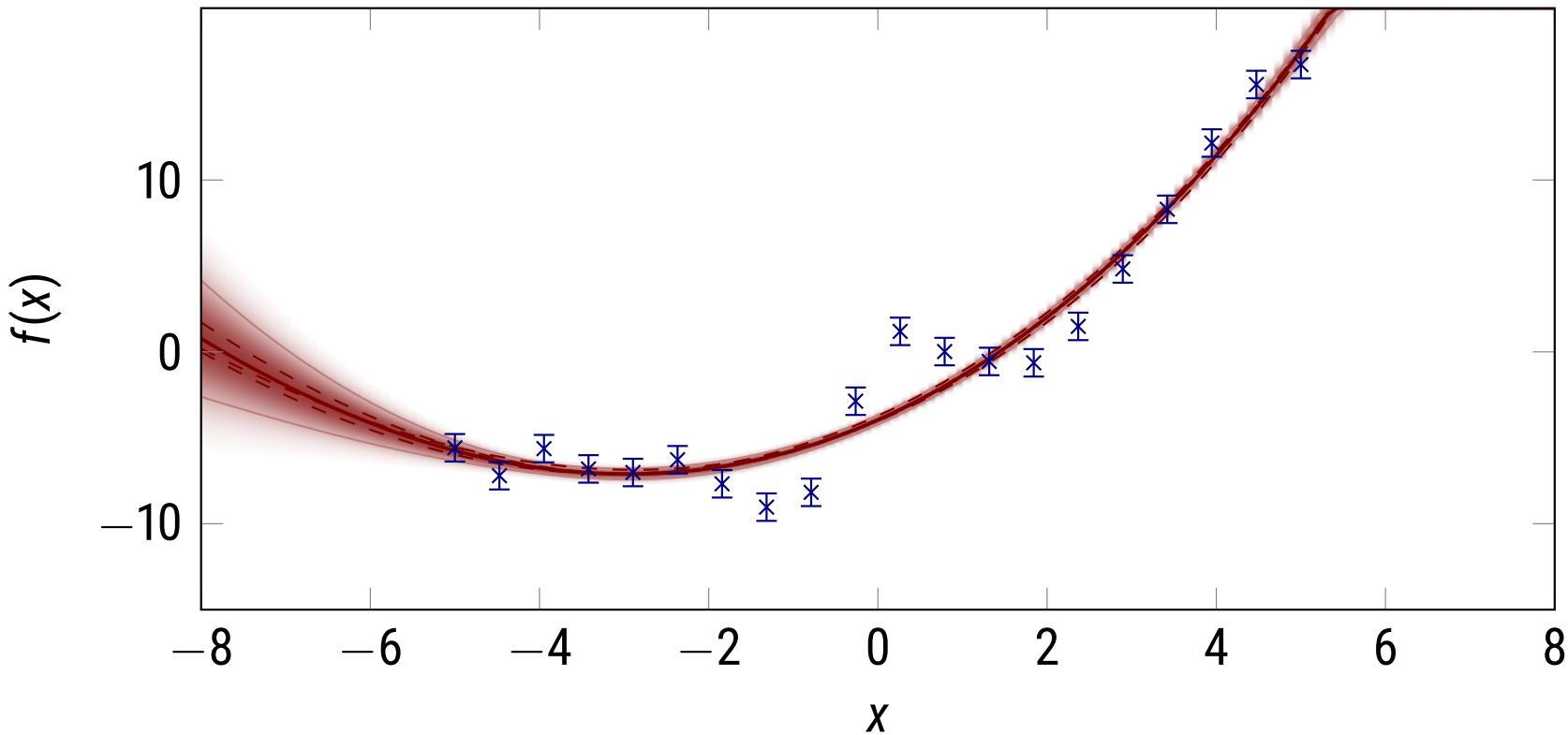
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix} 1 & x & x.^2 & x.^3 \end{pmatrix}^\intercal$$



Cubic Regression

```
phi = @(a)(bsxfun(@power,a,[0:3]));
```

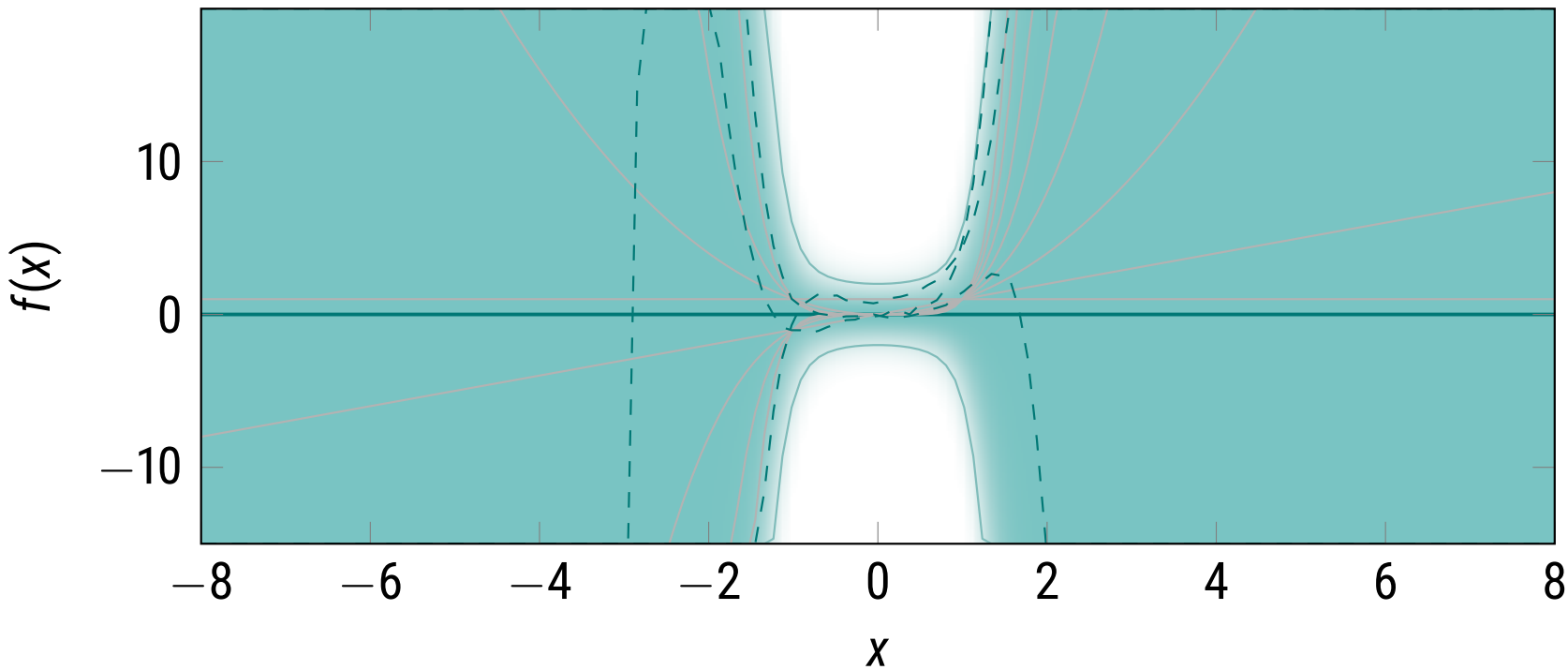
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix} 1 & x & x.^2 & x.^3 \end{pmatrix}^\intercal$$



Septic Regression?

```
phi = @(a)(bsxfun(@power,a,[0:7]));
```

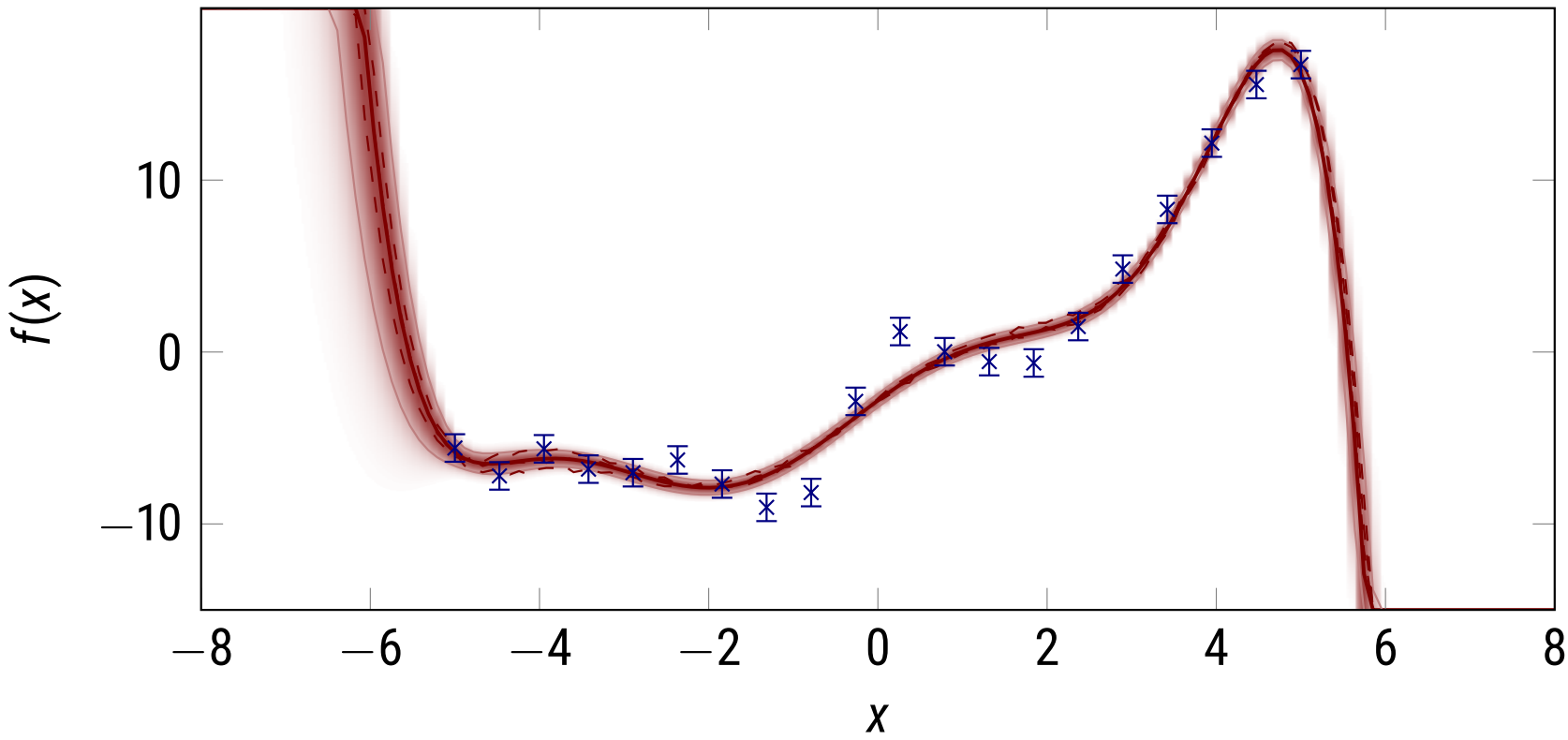
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix} 1 & x & x.^2 & \cdots & x.^7 \end{pmatrix}^\intercal$$



Septic Regression?

```
phi = @(a)(bsxfun(@power,a,[0:7]));
```

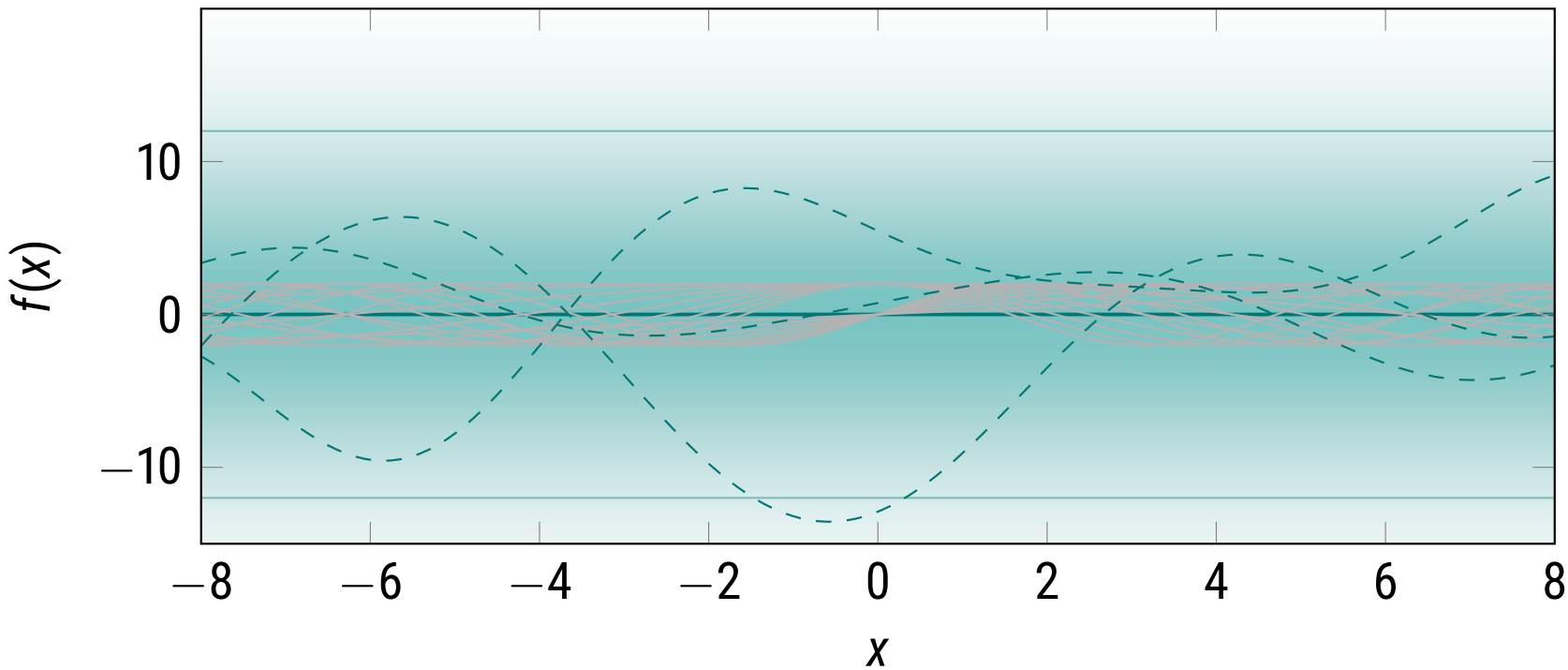
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix} 1 & x & x.^2 & \cdots & x.^7 \end{pmatrix}^\intercal$$



Fourier Regression

```
phi = @(a)(2 * [cos(bsxfun(@times,a/8,[0:8])), sin(bsxfun(@times,a/8,[1:8]))]);
```

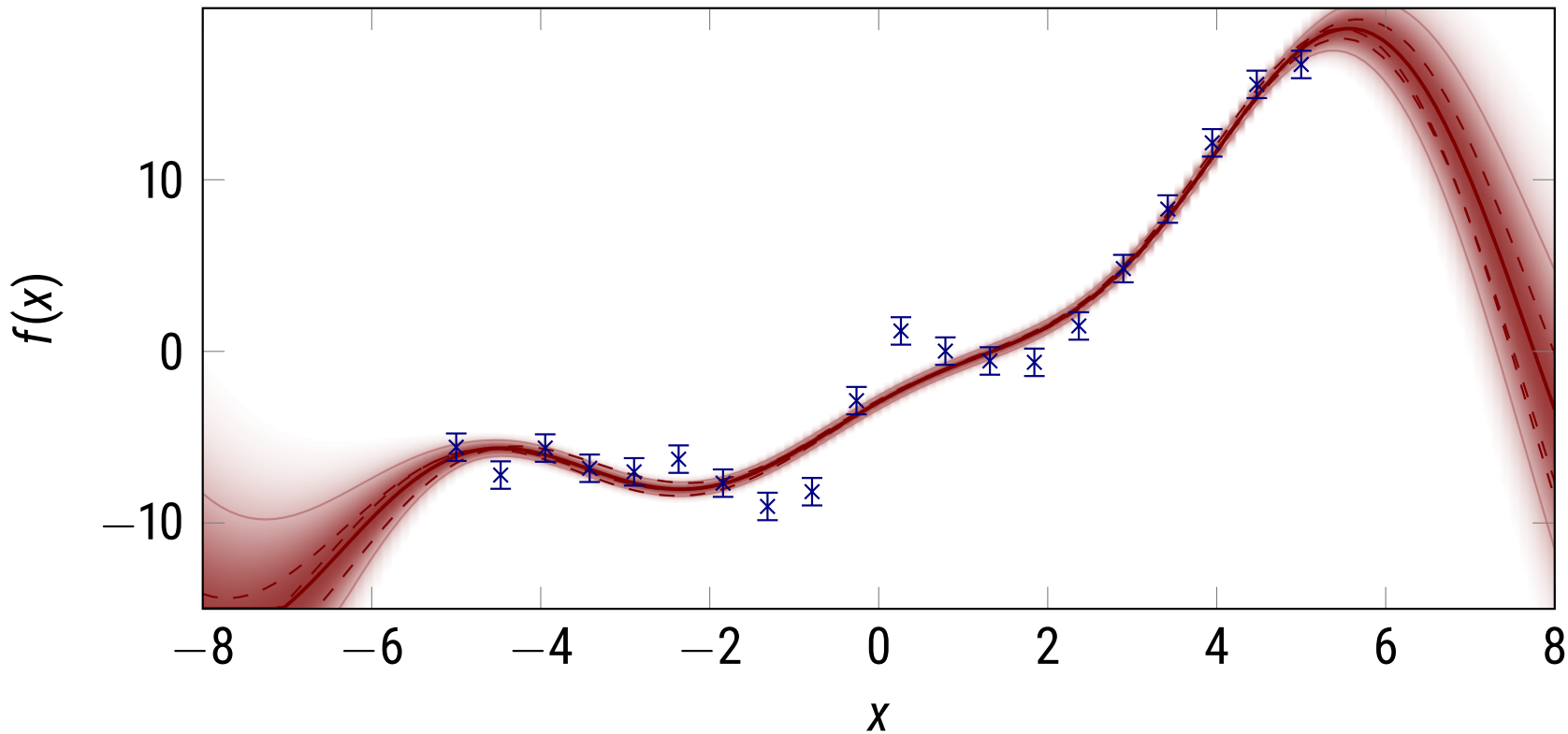
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix}\cos(x) & \cos(2x) & \cos(3x) & \dots & \sin(x) & \sin(2x) & \dots\end{pmatrix}^\intercal$$



Fourier Regression

```
phi = @(a)(2 * [cos(bsxfun(@times,a/8,[0:8])), sin(bsxfun(@times,a/8,[1:8]))]);
```

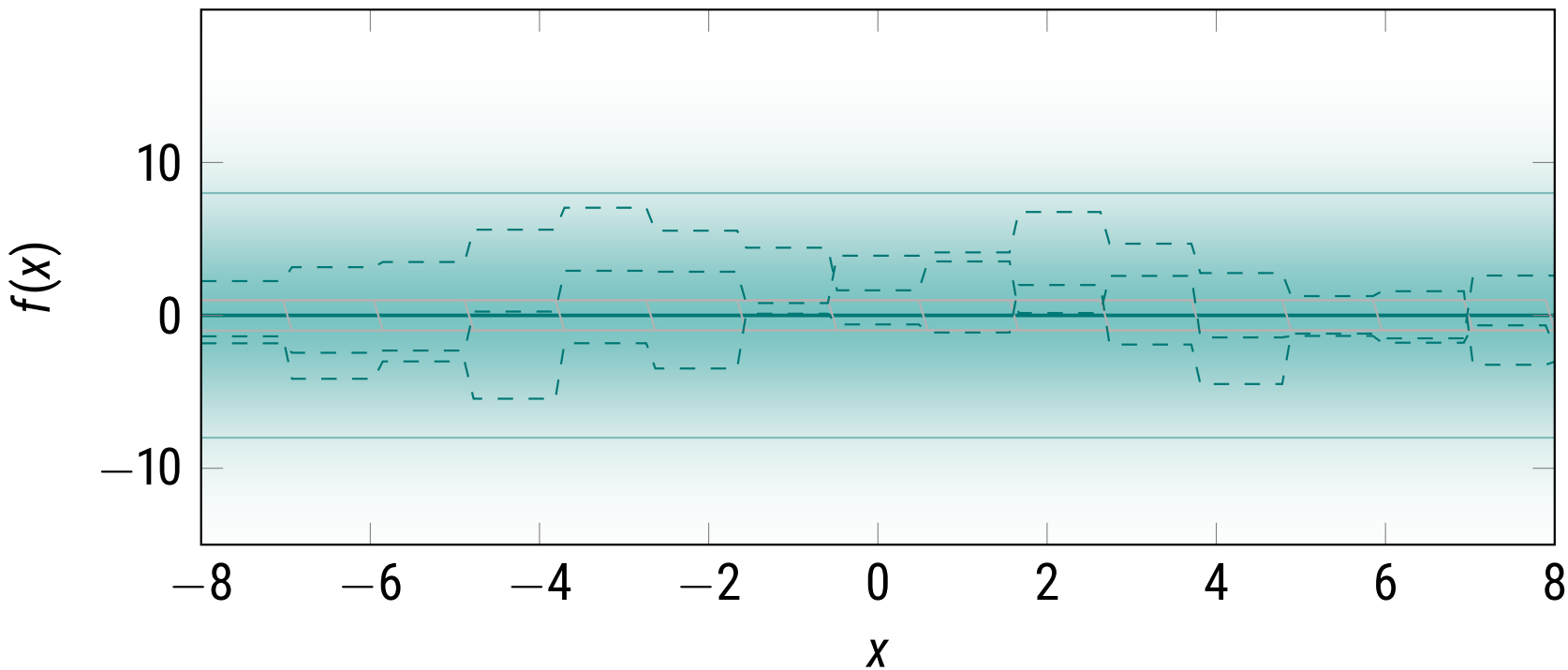
$$f(x) = \phi(x)^\intercal w \qquad \phi(x) = \begin{pmatrix}\cos(x) & \cos(2x) & \cos(3x) & \dots & \sin(x) & \sin(2x) & \dots\end{pmatrix}^\intercal$$



Step Regression

```
phi = @(a)(-1 + 2 * bsxfun(@lt,a,linspace(-8,8,16)));
```

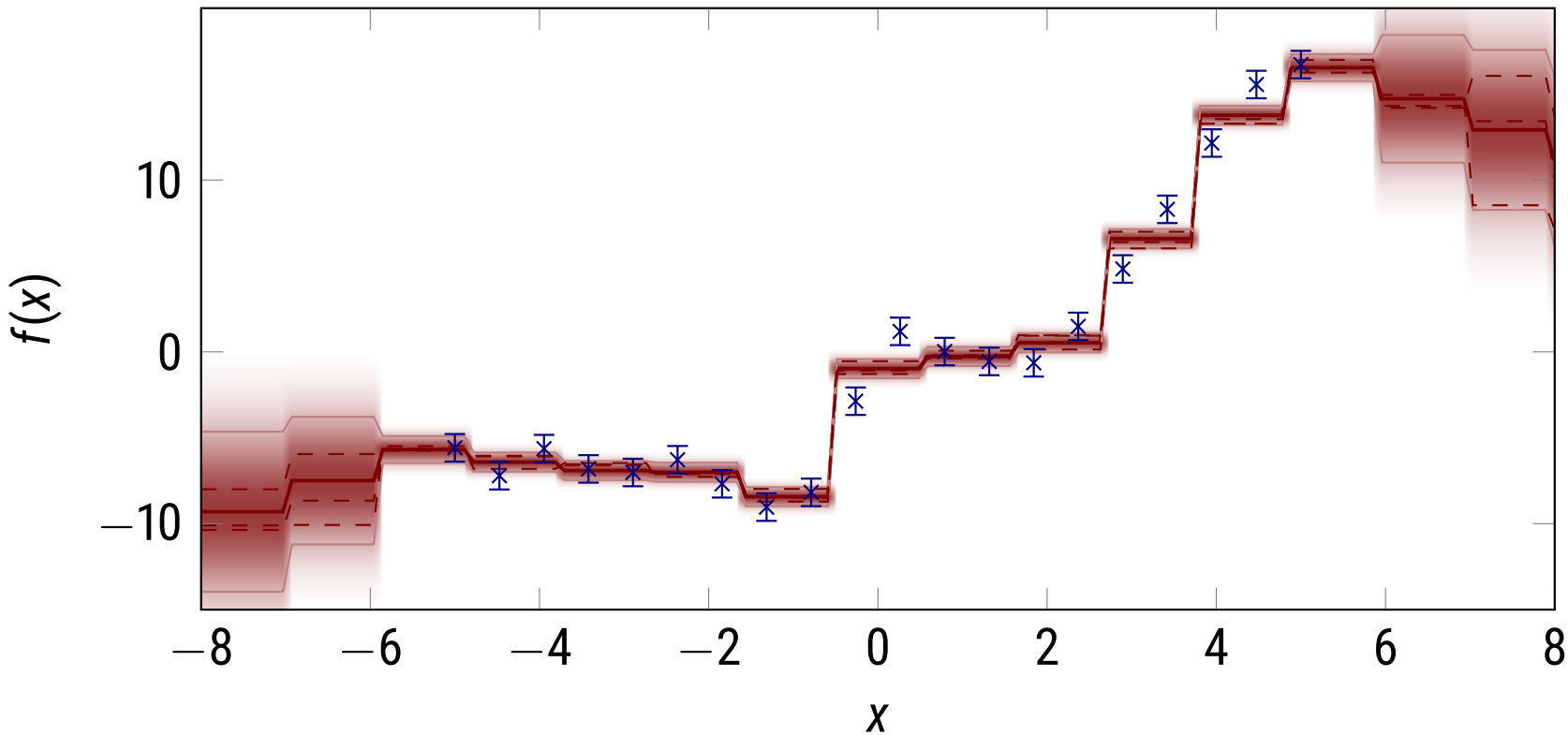
$$\phi(x) = -1 + 2\left(\theta(x-8) \quad \theta(8-x) \quad \theta(x-7) \quad \theta(7-x) \quad \ldots\right)^{\mathsf{T}}$$



Step Regression

```
phi = @(a)(-1 + 2 * bsxfun(@lt,a,linspace(-8,8,16)));
```

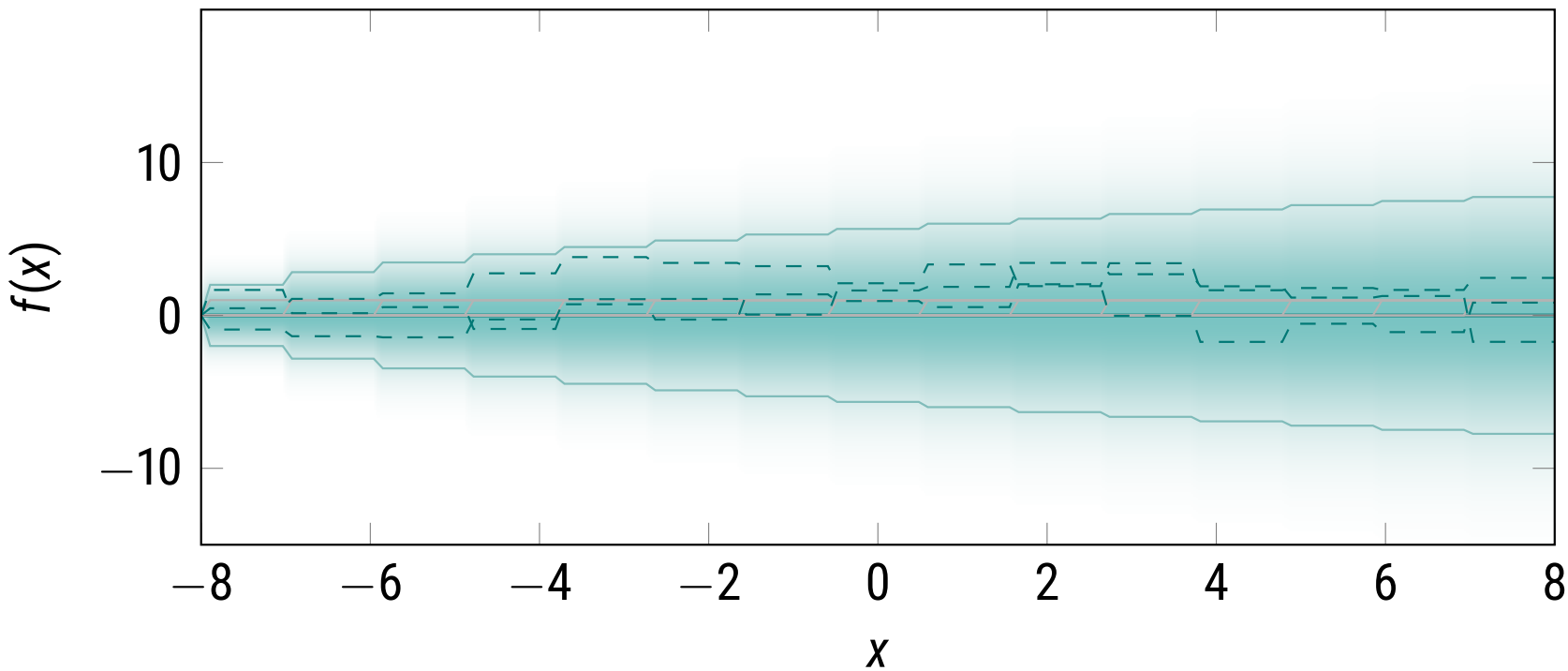
$$\phi(x) = -1 + 2\begin{pmatrix}\theta(x-8) & \theta(8-x) & \theta(x-7) & \theta(7-x) & \ldots\end{pmatrix}^{\mathsf{T}}$$



Another Kind of Step Regression

```
phi = @(a)(bsxfun(@gt,a,linspace(-8,8,16)));
```

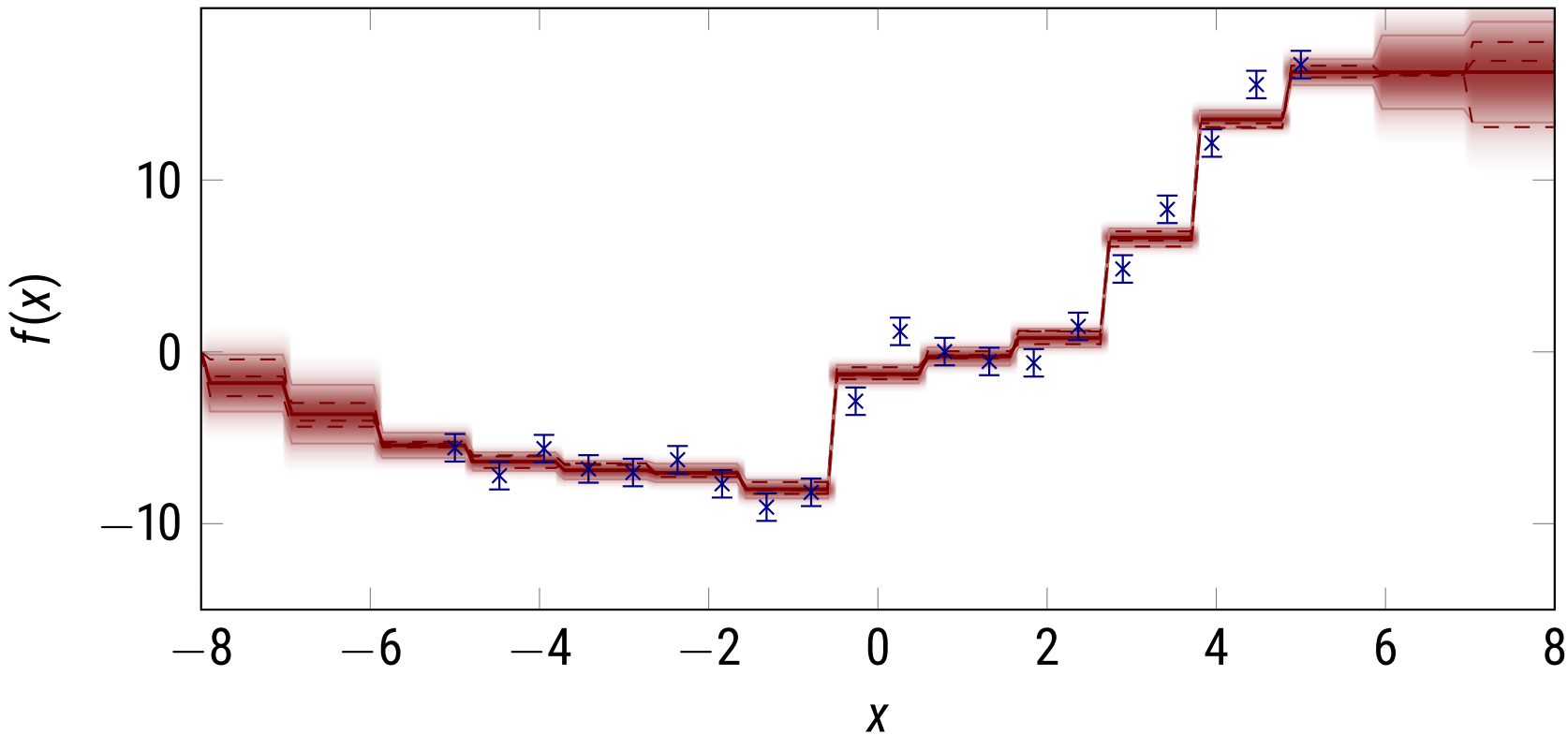
$$\phi(x) = \begin{pmatrix} \theta(x-8) & \theta(8-x) & \theta(x-7) & \theta(7-x) & \dots \end{pmatrix}^{\mathsf{T}}$$



Another Kind of Step Regression

```
phi = @(a)(bsxfun(@gt,a,linspace(-8,8,16)));
```

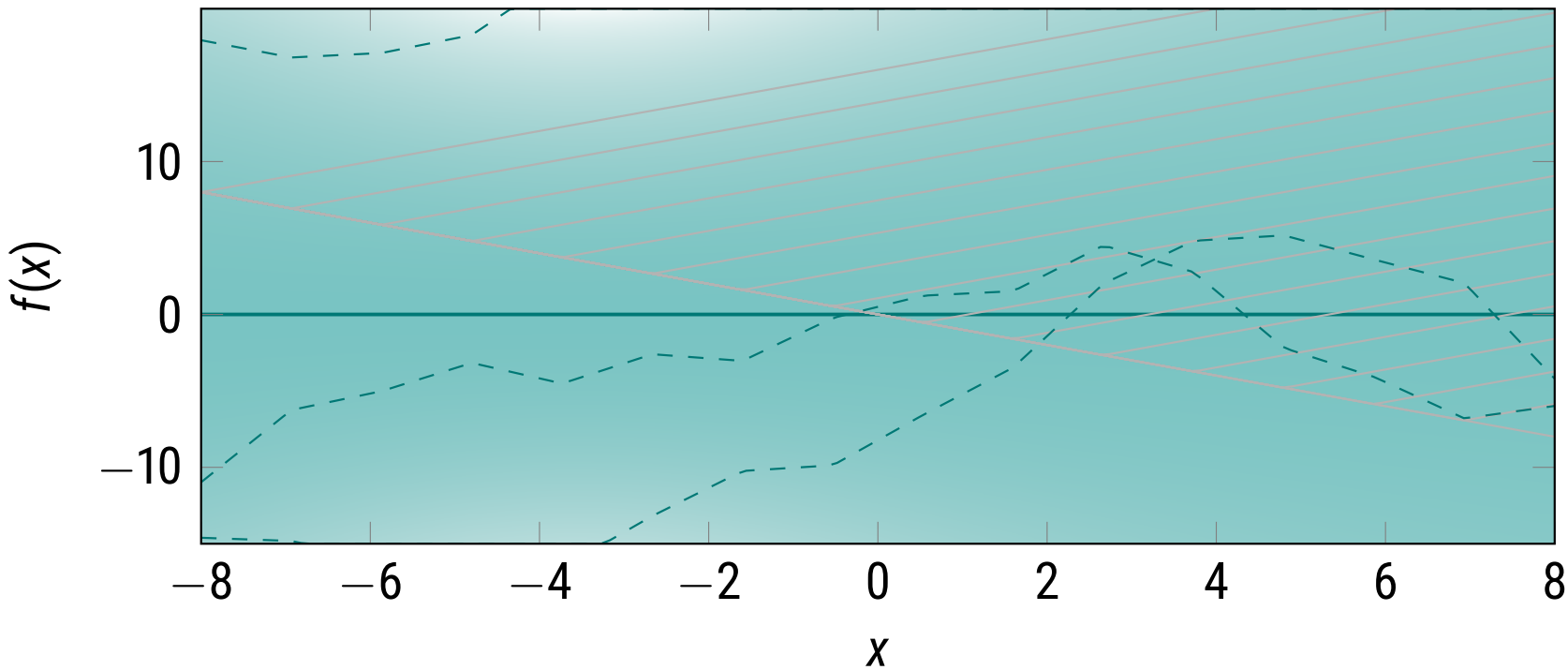
$$\phi(x) = \begin{pmatrix} \theta(x-8) & \theta(8-x) & \theta(x-7) & \theta(7-x) & \dots \end{pmatrix}^{\mathsf{T}}$$



V Regression

```
phi = @(a)(bsxfun(@minus,abs(bsxfun(@minus,a,linspace(-8,8,16))),linspace(-8,8,16)));
```

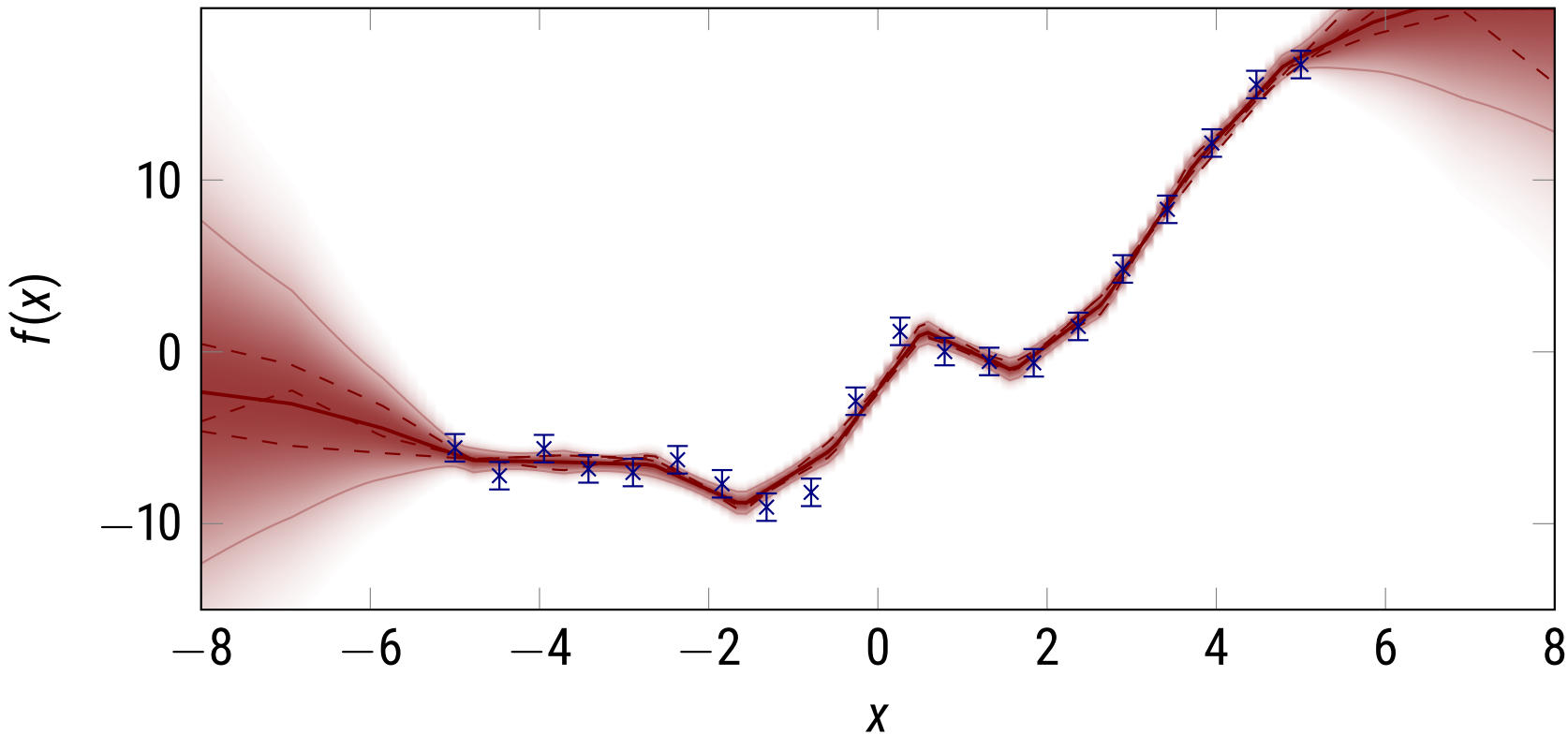
$$\phi(x) = \begin{pmatrix} |x-8|-8 & |x-7|-7 & |x-6|-6 & \dots \end{pmatrix}^{\mathsf{T}}$$



V Regression

```
phi = @(a)(bsxfun(@minus,abs(bsxfun(@minus,a,linspace(-8,8,16))),linspace(-8,8,16)));
```

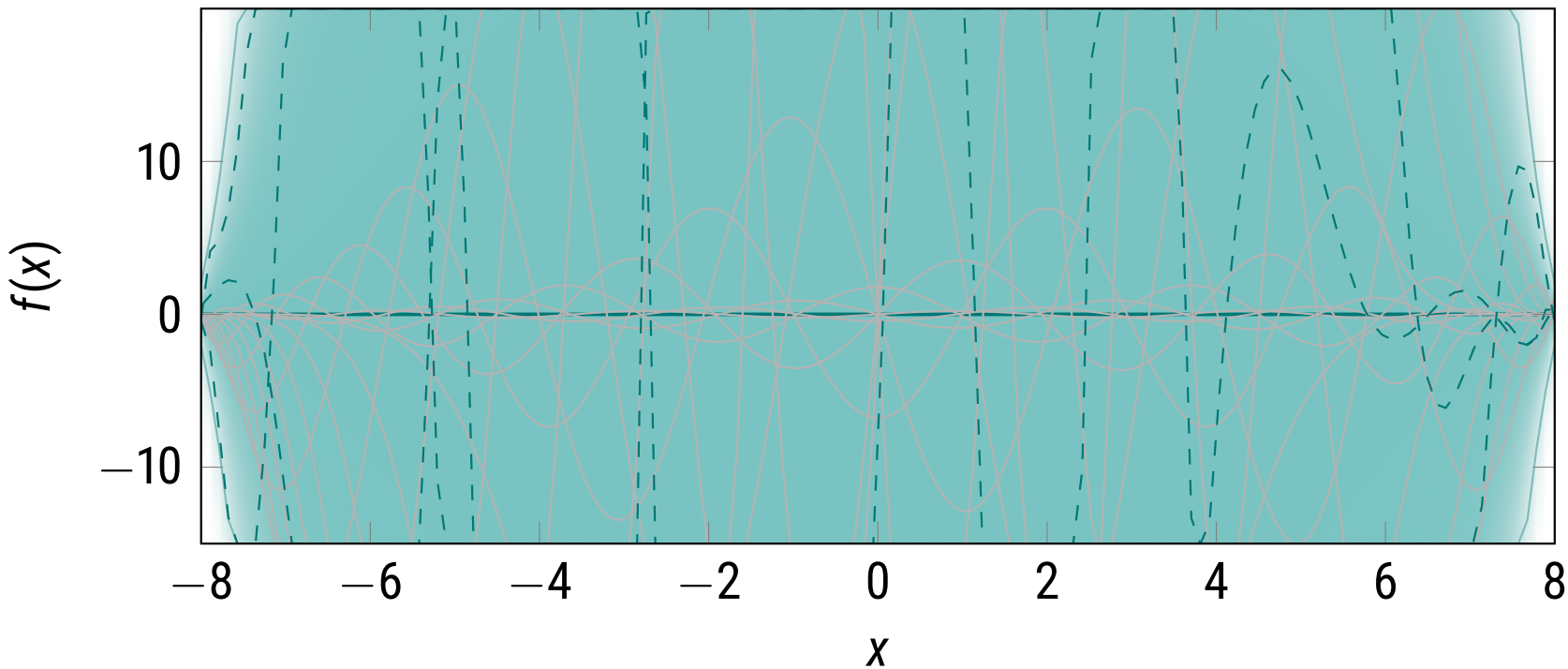
$$\phi(x) = \begin{pmatrix} |x-8|-8 & |x-7|-7 & |x-6|-6 & \dots \end{pmatrix}^{\mathsf{T}}$$



Legendre Regression

```
phi = @(a)(bsxfun(@times,legendre(13,a/8)',0.15.^[0:13]));
```

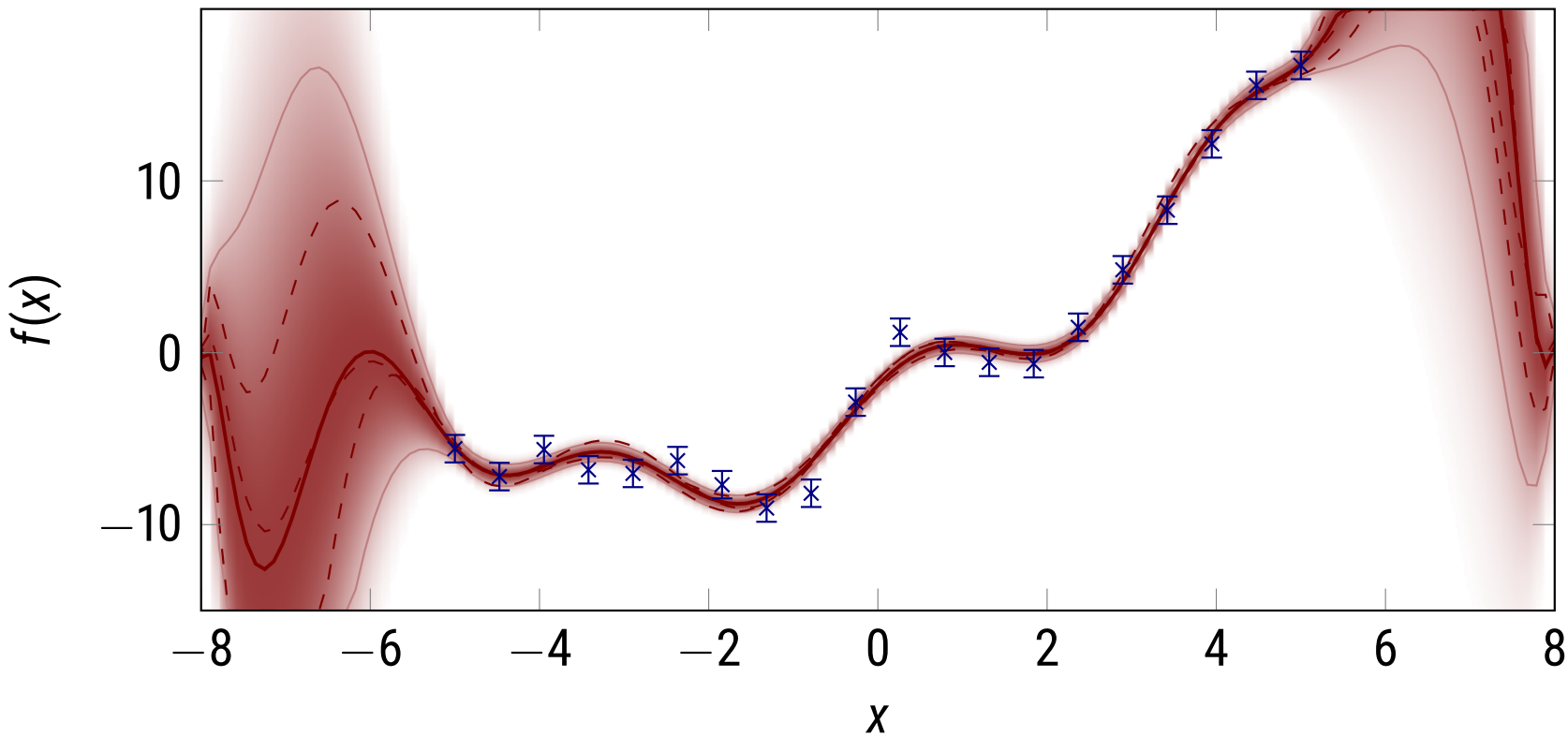
$$\phi(x) = \left(b^0 P_0(x), b^1 P_1(x), \ldots, b^{13} P_{13}(x)\right)^\mathsf{T} \qquad P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n$$



Legendre Regression

```
phi = @(a)(bsxfun(@times,legendre(13,a/8)',0.15.^[0:13]));
```

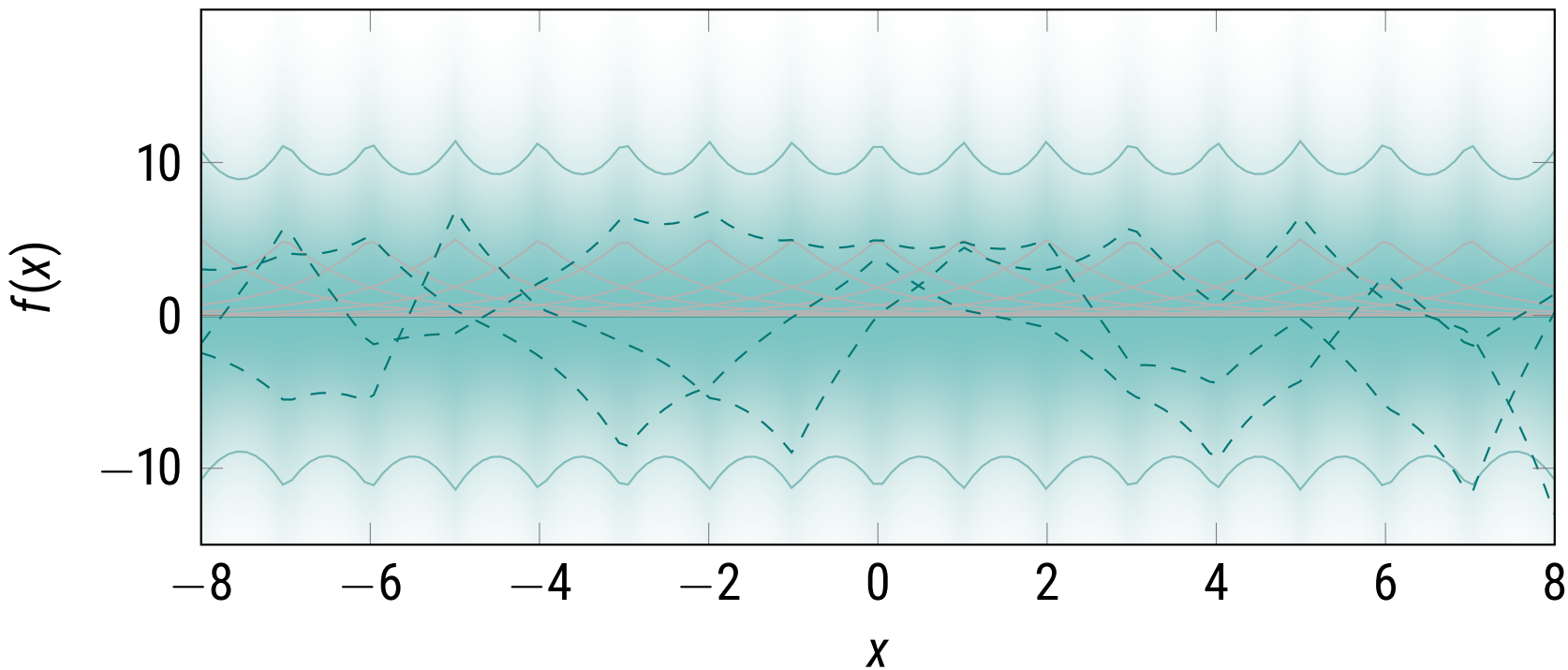
$$\phi(x) = \left(b^0 P_0(x), b^1 P_1(x), \ldots, b^{13} P_{13}(x)\right)^\mathsf{T} \qquad P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n}(x^2 - 1)^n$$



Eiffel Tower Regression

```
phi = @(a)(exp(-abs(bsxfun(@minus,a,[-8:1:8]))));
```

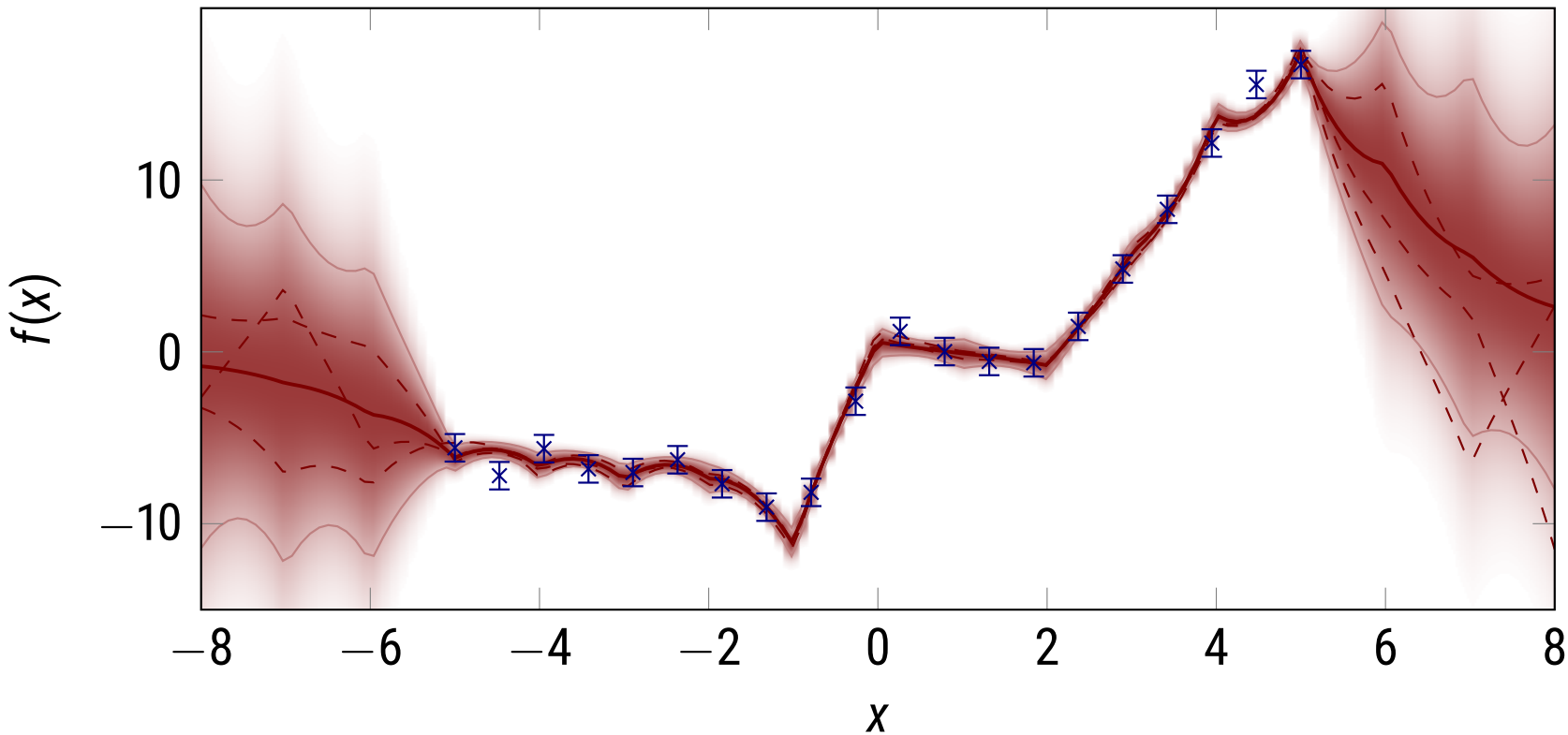
$$\phi(x) = \begin{pmatrix} e^{-|x-8|} & e^{-|x-7|} & e^{-|x-6|} & \dots \end{pmatrix}^{\mathsf{T}}$$



Eiffel Tower Regression

```
phi = @(a)(exp(-abs(bsxfun(@minus,a,[-8:1:8]))));
```

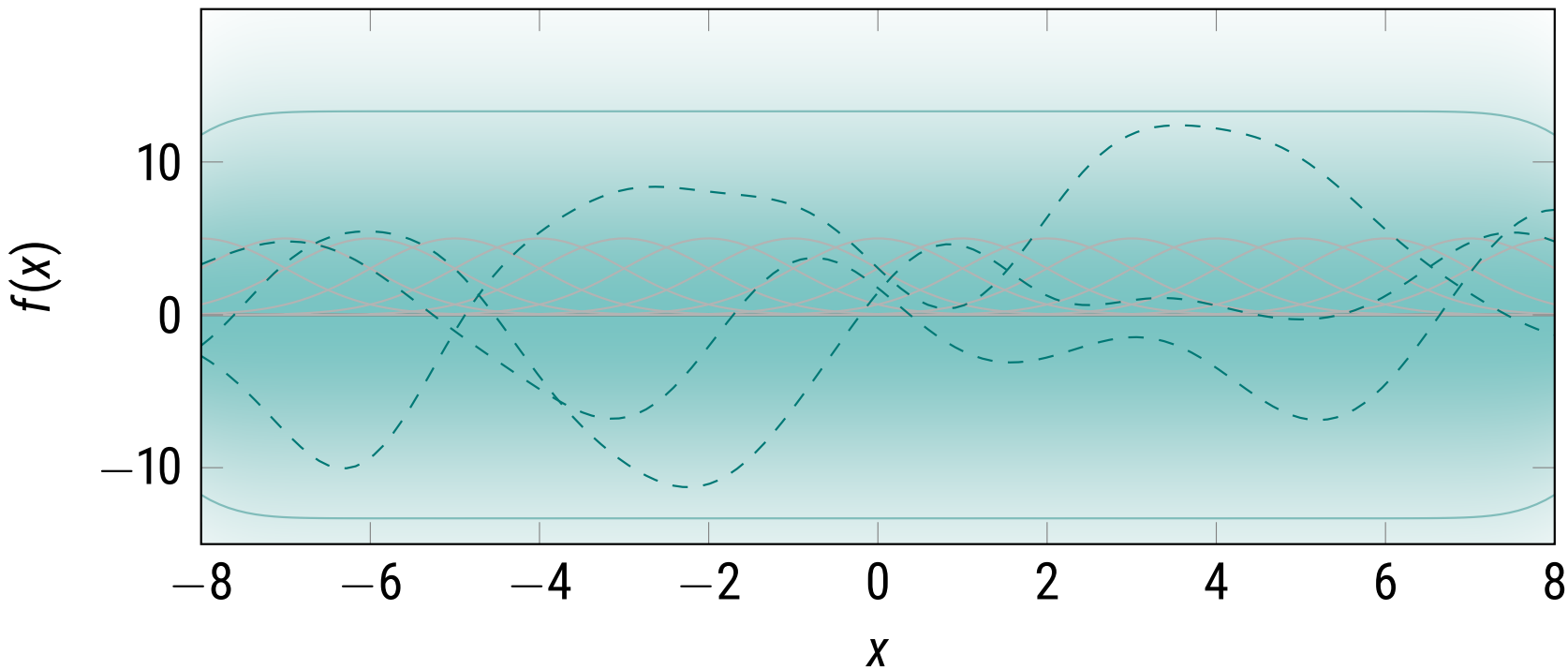
$$\phi(x) = \begin{pmatrix} e^{-|x-8|} & e^{-|x-7|} & e^{-|x-6|} & \dots \end{pmatrix}^{\mathsf{T}}$$



Bell Curve Regression

```
phi = @(a)(exp(-0.5 * bsxfun(@minus,a,[-8:1:8]).^2));
```

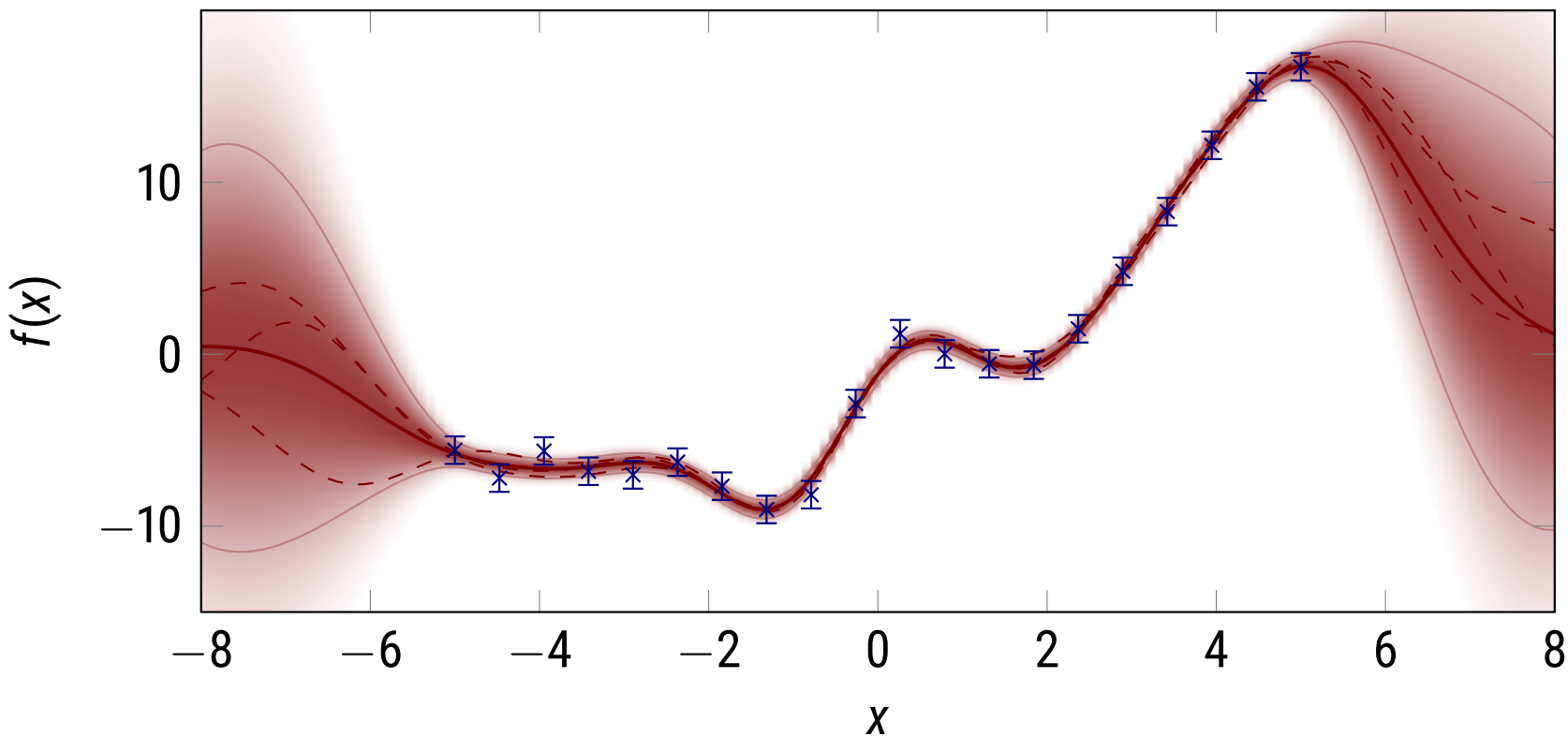
$$\phi(x) = \begin{pmatrix} e^{-\frac{1}{2}(x-8)^2} & e^{-\frac{1}{2}(x-7)^2} & e^{-\frac{1}{2}(x-6)^2} & \dots \end{pmatrix}^T$$



Bell Curve Regression

```
phi = @(a)(exp(-0.5 * bsxfun(@minus,a,[-8:1:8]).^2));
```

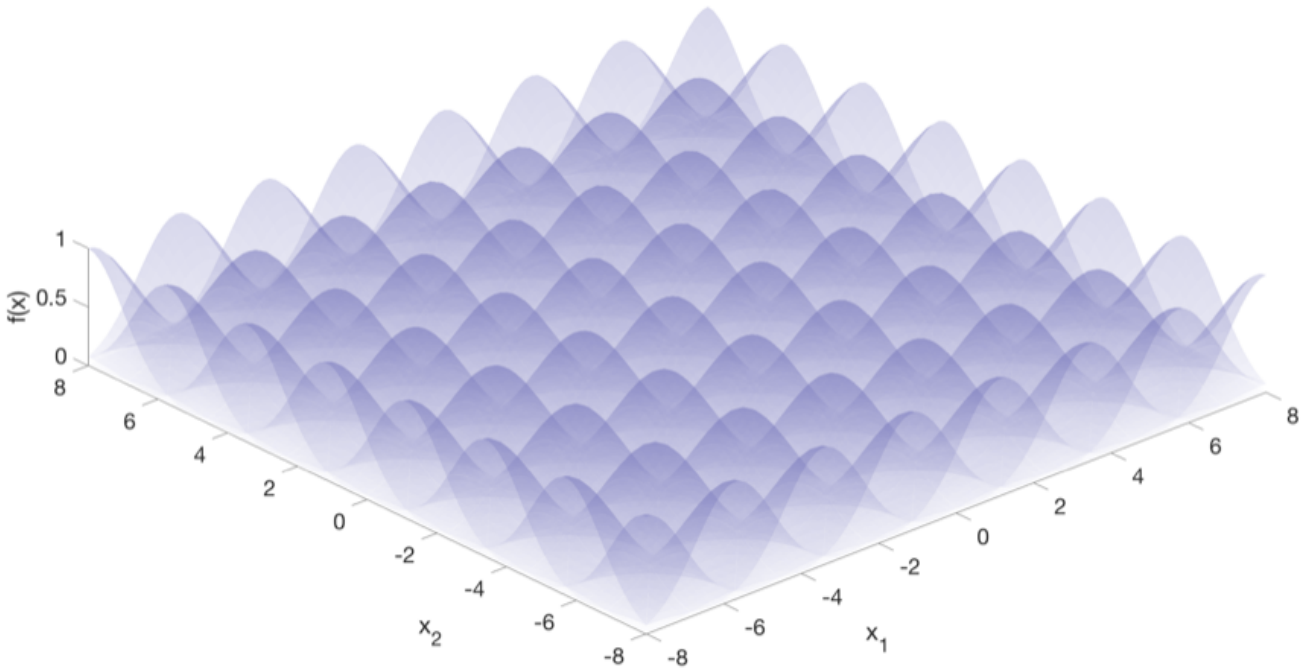
$$\phi(x) = \left(e^{-\frac{1}{2}(x-8)^2} \quad e^{-\frac{1}{2}(x-7)^2} \quad e^{-\frac{1}{2}(x-6)^2} \quad \dots \right)^T$$



Multiple Inputs

all this works for in multiple dimensions, too

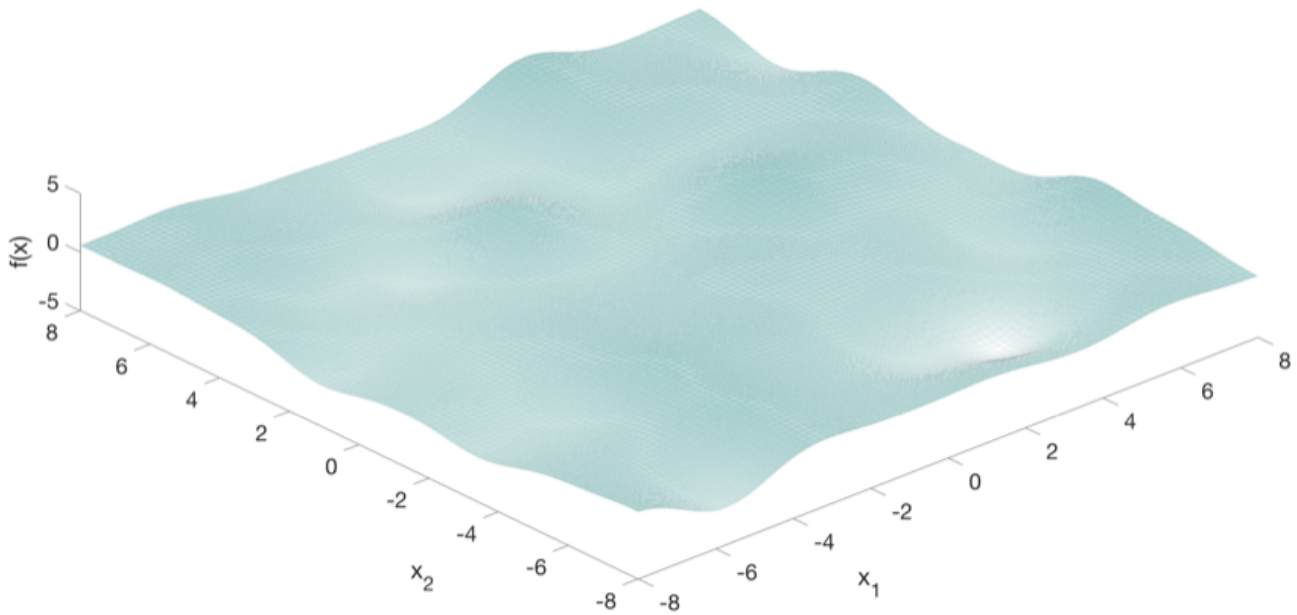
$$\phi : \mathbb{R}^N \to \mathbb{R} \qquad f : \mathbb{R}^N \to \mathbb{R}$$



Multiple Inputs

all this works for in multiple dimensions, too

$$\phi : \mathbb{R}^N \to \mathbb{R} \qquad f : \mathbb{R}^N \to \mathbb{R}$$

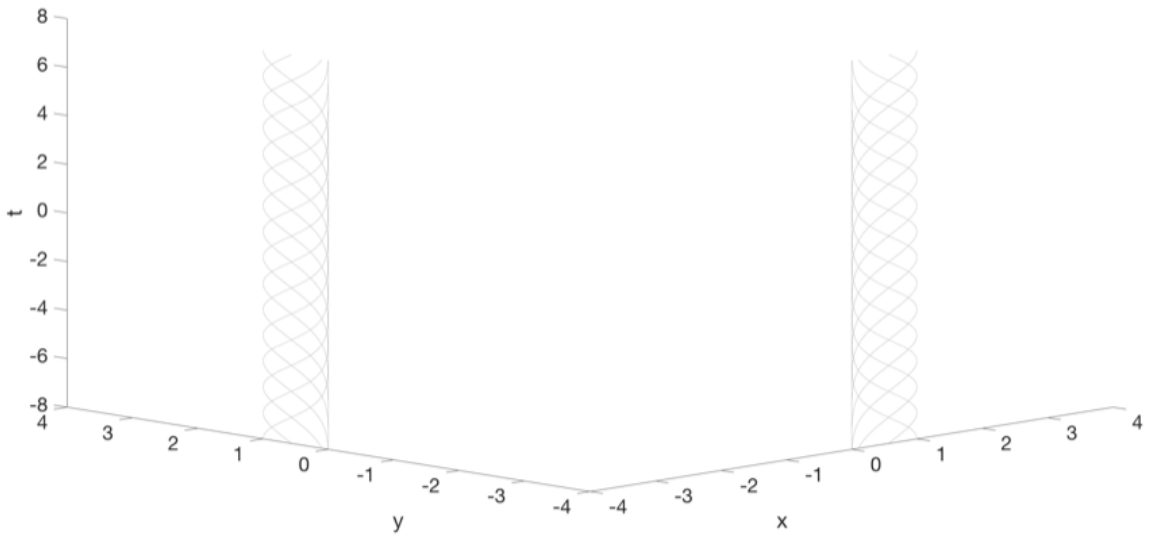


Multiple Outputs

slightly more confusing, but no algebraic problem

$$\phi : \mathbb{R} \to \mathbb{R}^M \qquad f : \mathbb{R} \to \mathbb{R}^M \qquad \text{cov}(f_i(t), f_j(t)) = \sum_{\ell} \phi_{\ell,i}(t)\phi_{\ell,j}(t')$$

- $[f_1(t_1), \ldots, f_1(t_N), f_2(t_1), \ldots, f_2(t_N), \ldots, f_M(t_1), \ldots f_M(t_N)]$ are just some co-varying Gaussian variables
- requires careful matrix algebra

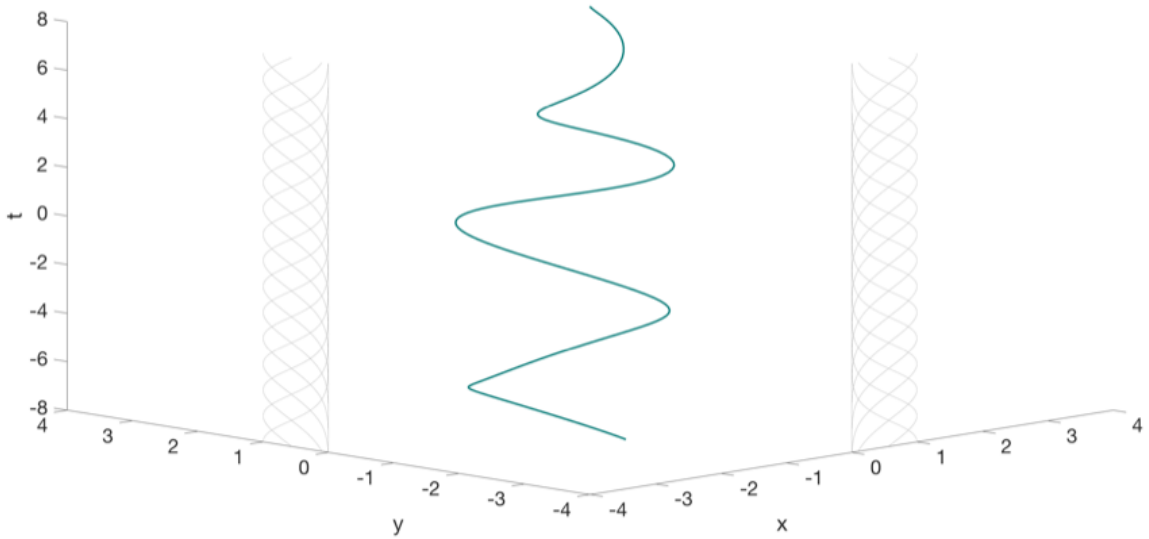


Multiple Outputs

slightly more confusing, but no algebraic problem

$$\phi : \mathbb{R} \to \mathbb{R}^M \qquad f : \mathbb{R} \to \mathbb{R}^M \qquad \text{cov}(f_i(t), f_j(t)) = \sum_{\ell} \phi_{\ell,i}(t)\phi_{\ell,j}(t')$$

- $[f_1(t_1), \ldots, f_1(t_N), f_2(t_1), \ldots, f_2(t_N), \ldots, f_M(t_1), \ldots f_M(t_N)]$ are just some co-varying Gaussian variables
- requires careful matrix algebra

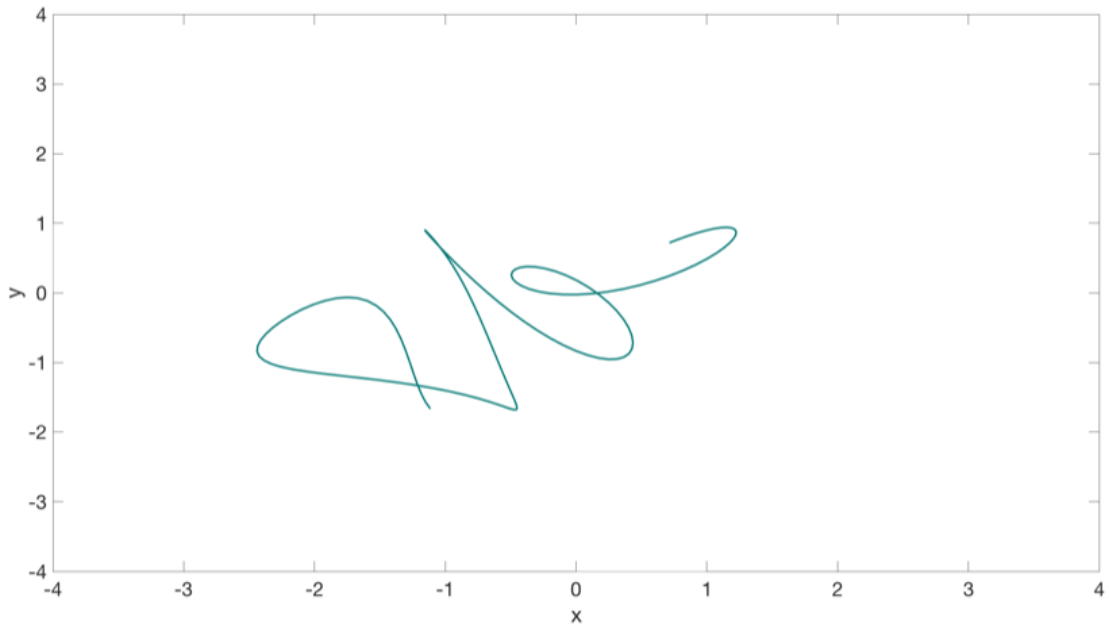


Multiple Outputs

slightly more confusing, but no algebraic problem

$$\phi : \mathbb{R} \to \mathbb{R}^M \qquad f : \mathbb{R} \to \mathbb{R}^M \qquad \text{cov}(f_i(t), f_j(t)) = \sum_{\ell} \phi_{\ell,i}(t)\phi_{\ell,j}(t')$$

- $[f_1(t_1), \ldots, f_1(t_N), f_2(t_1), \ldots, f_2(t_N), \ldots, f_M(t_1), \ldots f_M(t_N)]$ are just some co-varying Gaussian variables
- requires careful matrix algebra



- Gaussian measures turn probabilistic inference into **linear algebra**
- Gaussian measures can be used to infer **functions** and **curves**

Next up:

- In fact, the number of inferred features / weights can be **infinite**!

How many features should we use?

let's look at that algebra again

$$p(f_x \mid y, \phi_X) = \mathcal{N}(f_x; \phi_x^\intercal \mu + \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1}(y - \phi_X^\intercal \mu),$$
$$\phi_x^\intercal \Sigma \phi_x - \phi_x^\intercal \Sigma \phi_X (\phi_X^\intercal \Sigma \phi_X + \sigma^2 I)^{-1} \phi_X^\intercal \Sigma \phi_x)$$

- no lonely ϕ. Instead:
  - $\phi^\intercal \mu$ – the mean function
  - $\phi^\intercal \Sigma \phi$ – the kernel
- given these, cost independent of features
- remember the code:

```
mx     = dot(phi(x),mu)
kxx    = dot(dot(phi(x),Sigma),phi(x).transpose())
mX     = dot(phi(X),mu)
kXX    = dot(dot(phi(X),Sigma),phi(X).transpose())
kxX    = dot(dot(phi(x),Sigma),phi(X).transpose())
```

Code using Kernels and mean functions

python

```
#                          ==== NONPARAMETRIC GAUSSIAN REGRESSION IN PYTHON ====
# assume external definition of kernel k and mean function m
# prior on f_x
n      = 100
x      = reshape(linspace(-8,8,n),(n,1))     # reshape is needed for phi to work
mx     = m(x)
kxx    = k(x,x)                                                  # p(f_x)=N(m,k_xx)
s      = dot(cholesky(kxx + 1e-6 * eye(n)),randn(n,3)) + mx      # prior samples
stdpi  = reshape(sqrt(diag(kxx)),[n,1])          # marginal stddev, for plotting

# loading data from disk
import scipy.io; data = scipy.io.loadmat('data.mat')
X = data['X']; Y = data['Y']; sigma = data['sigma']; N  = len(X)

# prior on Y = f_X + eps
mX     = m(X)
kXX    = k(X,X)                                             # p(f_X) = N(mX,k_XX)

# inference
G      = cho_factor(kXX + sigma**2 * eye(N))       # Most expensive step, O(N^3)
kxX    = k(x,X)                                             # Cov(f_x,f_X) = k_xX
A      = cho_solve(G,kxX.transpose()).transpose()       # pre-compute for re-use

# posterior p(f_x|Y)
mpost  = mx + dot(A,(Y-mX))                                          # mean
vpost  = kxx - dot(A,kxX.transpose())                              # covariance
spost  = dot(cholesky(vpost + 1e-8 * eye(n)),randn(n,3)) + mpost       # samples
stdpo  = reshape(sqrt(diag(vpost)),[n,1])        # marginal stddev, for
```

Code using Kernels and mean functions

matlab

```
%                          ==== NONPARAMETRIC GAUSSIAN REGRESSION IN MATLAB ====
% assume external definition of kernel k and mean function m
% prior on f_x
n      = 100;
x      = linspace(-8,8,n)';                                        % `test' points
mx     = m(x);
kxx    = k(x,x);                                                % p(f_x)=N(mx,k_xx)
s      = bsxfun(@plus,mx,chol(kxx + 1.0e-8 * eye(n))' * randn(n,3));   % samples
stdpi  = sqrt(diag(kxx));                              % marginal stddev, for plotting

% loading data from disk
load('data.mat'); N = length(Y);                                  % gives Y,X,sigma

% prior on Y=f_X + epsilon
mX     = m(X);
kXX    = k(X,X);                                              % p(f_X) = N(mX,k_XX)

% inference
G      = chol(kXX + sigma^ 2 * eye(N);)              % most expensive step: O(N^3)
kxX    = k(x,X);                                             % Cov(f_x,f_X) = k_xX
A      = kxX / G;                                          % pre-compute for re-use

% posterioe p (f_x|Y)
mpost  = mx + A * (G' \ (Y-mX));                                          % mean
vpost  = kxx - A * A';                                                % covariance
spost  = bsxfun(@plus,mpost,chol(vpost + 1.0e-8 * eye(n))' * randn(n,3)) % smpls
stdpo  = sqrt(diag(vpost));                            % marginal stddev, for plotting
```

Features are cheap, so let's use a lot

an example

[DJC MacKay, 1998]

- For simplicity, let's fix $\Sigma = \frac{\sigma^2(c_{\max} - c_{\min})}{F} I$

$$\text{thus:} \quad \phi(x_i)^\intercal \Sigma \phi(x_j) = \frac{\sigma^2(c_{\max} - c_{\min})}{F} \sum_{\ell=1}^{F} \phi_\ell(x_i)\phi_\ell(x_j)$$

- especially, for $\phi_\ell(x) = \exp\left(-\frac{(x - c_\ell)^2}{2\lambda^2}\right)$

$$\phi(x_i)^\intercal \Sigma \phi(x_j)$$

$$= \frac{\sigma^2(c_{\max} - c_{\min})}{F} \sum_{\ell=1}^{F} \exp\left(-\frac{(x_i - c_\ell)^2}{2\lambda^2}\right) \exp\left(-\frac{(x_j - c_\ell)^2}{2\lambda^2}\right)$$

$$= \frac{\sigma^2(c_{\max} - c_{\min})}{F} \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right) \sum_{\ell}^{F} \exp\left(-\frac{(c_\ell - \frac{1}{2}(x_i + x_j))^2}{\lambda^2}\right)$$

Features are cheap, so let's use a lot

an example

[DJC MacKay, 1998]

$$\phi(x_i)^\intercal \Sigma \phi(x_j) =$$

$$\frac{\sigma^2(c_{\max} - c_{\min})}{F} \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right) \sum_{\ell}^{F} \exp\left(-\frac{(c_\ell - \frac{1}{2}(x_i + x_j))^2}{\lambda^2}\right)$$

- now increase F so # of features in δc approaches $\frac{F \cdot \delta c}{(c_{\max} - c_{\min})}$

$$\phi(x_i)^\intercal \Sigma \phi(x_j) \to$$

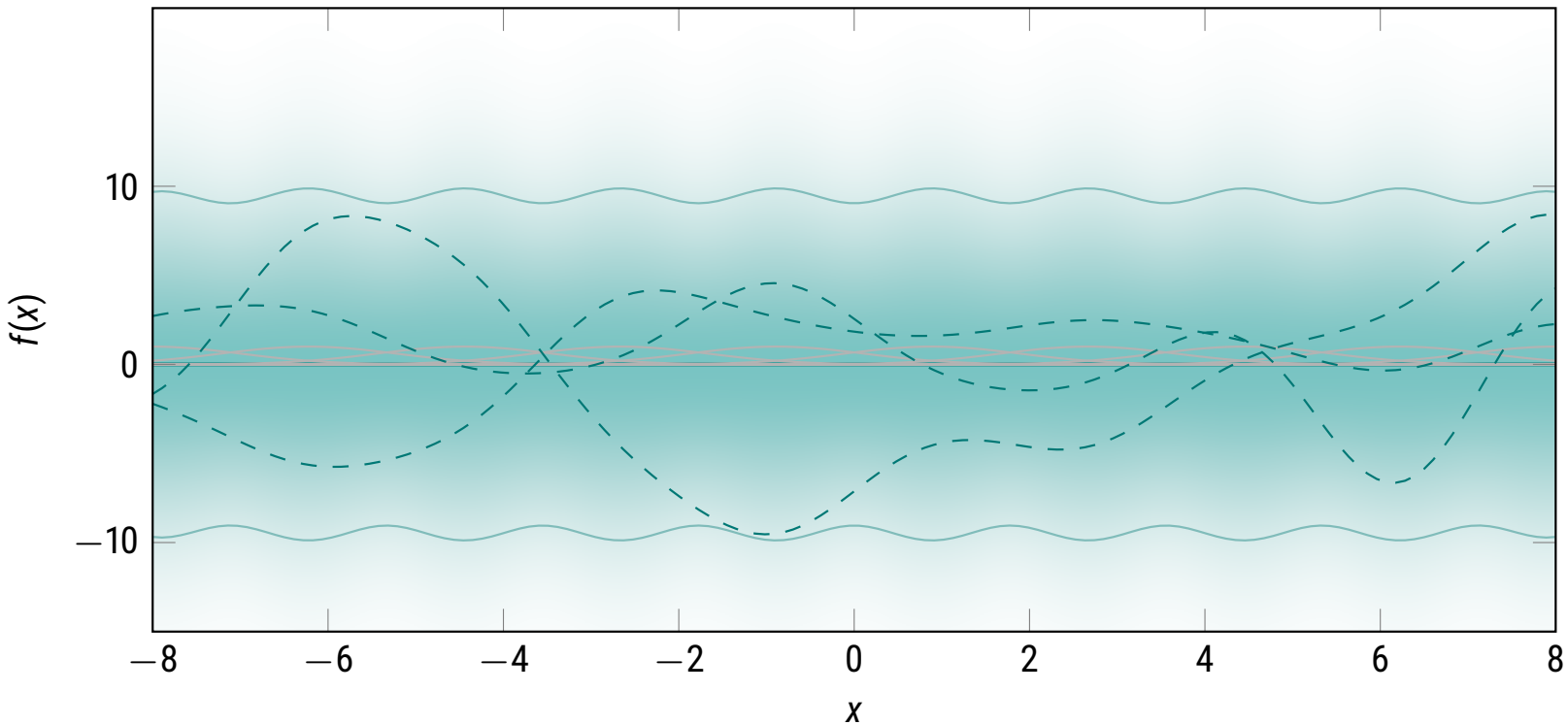
$$\sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right) \int_{c_{\min}}^{c_{\max}} \exp\left(-\frac{(c - \frac{1}{2}(x_i + x_j))^2}{\lambda^2}\right) dc$$

- let $c_{\min} \to -\infty$, $c_{\max} \to \infty$

$$k(x_i, x_j) := \phi(x_i)^\intercal \Sigma \phi(x_j) \to \sqrt{2\pi}\lambda\sigma^2 \exp\left(-\frac{(x_i - x_j)^2}{4\lambda^2}\right)$$

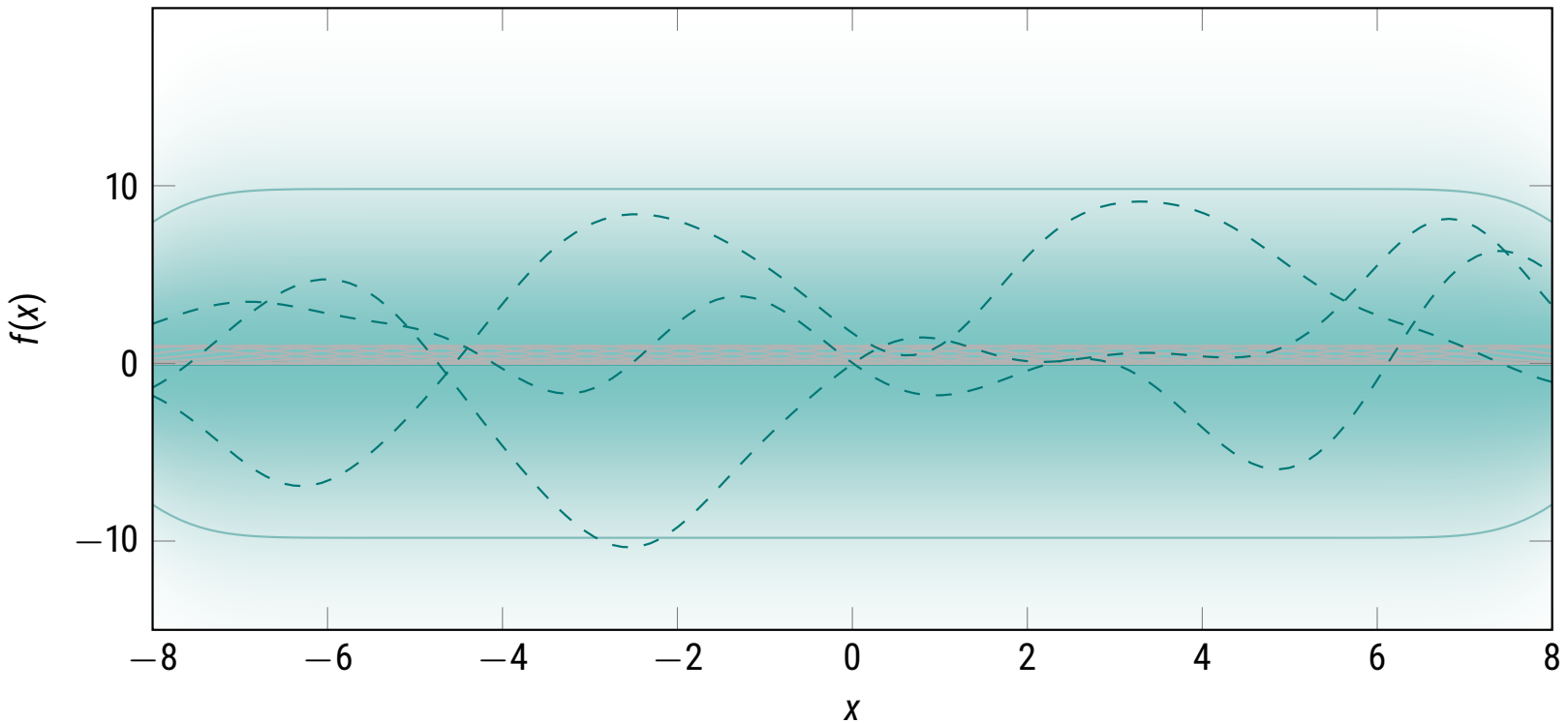
Exponentiated Squares

```
phi = @(a)(exp(-0.5 * bsxfun(@minus,a,linspace(-8,8,10))).^2 ./ell.^2));
```



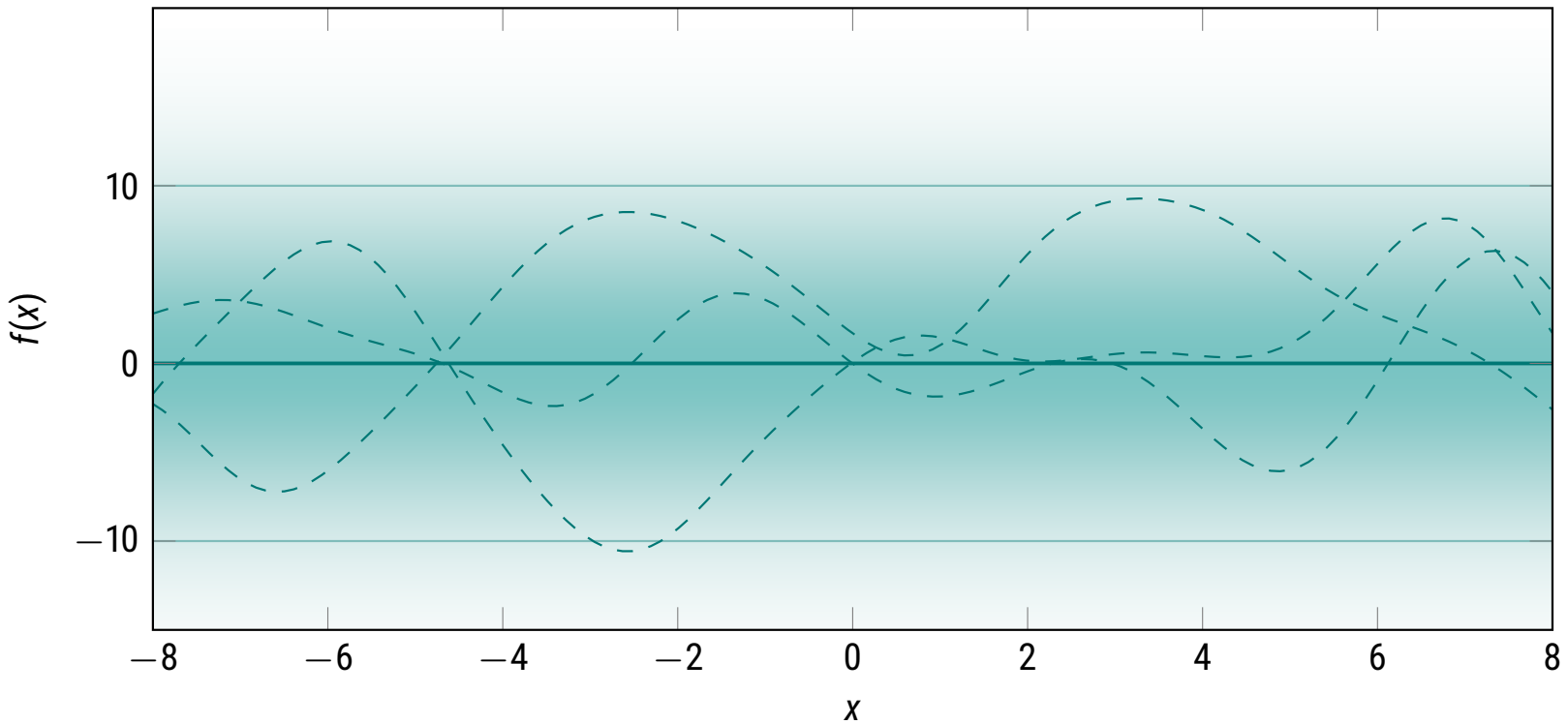
Exponentiated Squares

```
phi = @(a)(exp(-0.5 * bsxfun(@minus,a,linspace(-8,8,30))).^2 ./ell.^2));
```



Exponentiated Squares

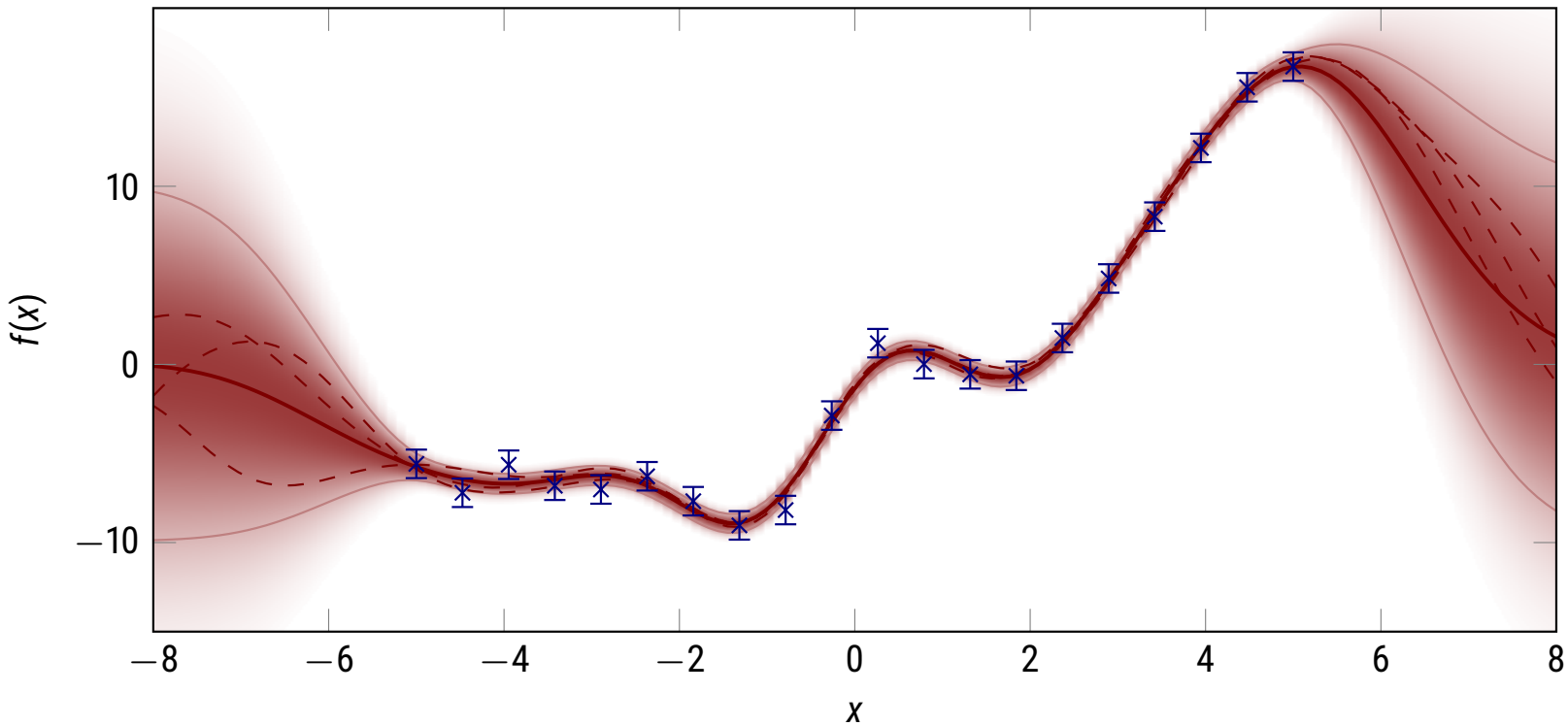
```
k = @(a,b)(5*exp(-0.25*bsxfun(@minus,a,b').^2));
```



- aka. radial basis function, square(d)-exponential kernel

Exponentiated Squares

```
k = @(a,b)(5*exp(-0.25*bsxfun(@minus,a,b').^2));
```



- aka. radial basis function, square(d)-exponential kernel

What just happened?

kernelization and Gaussian processes

$k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ is a **Mercer kernel** if, for any finite collection $X = [x_1, \dots, x_N]$, the matrix $k_{XX} \in \mathbb{R}^{N \times N}$ with $[k_{XX}]_{ij} = k(x_i, x_j)$ is **positive semidefinite**.

Lemma:

k is a Mercer kernel if it can be written as (assuming $\mathcal{L}$ is positive set for ν)

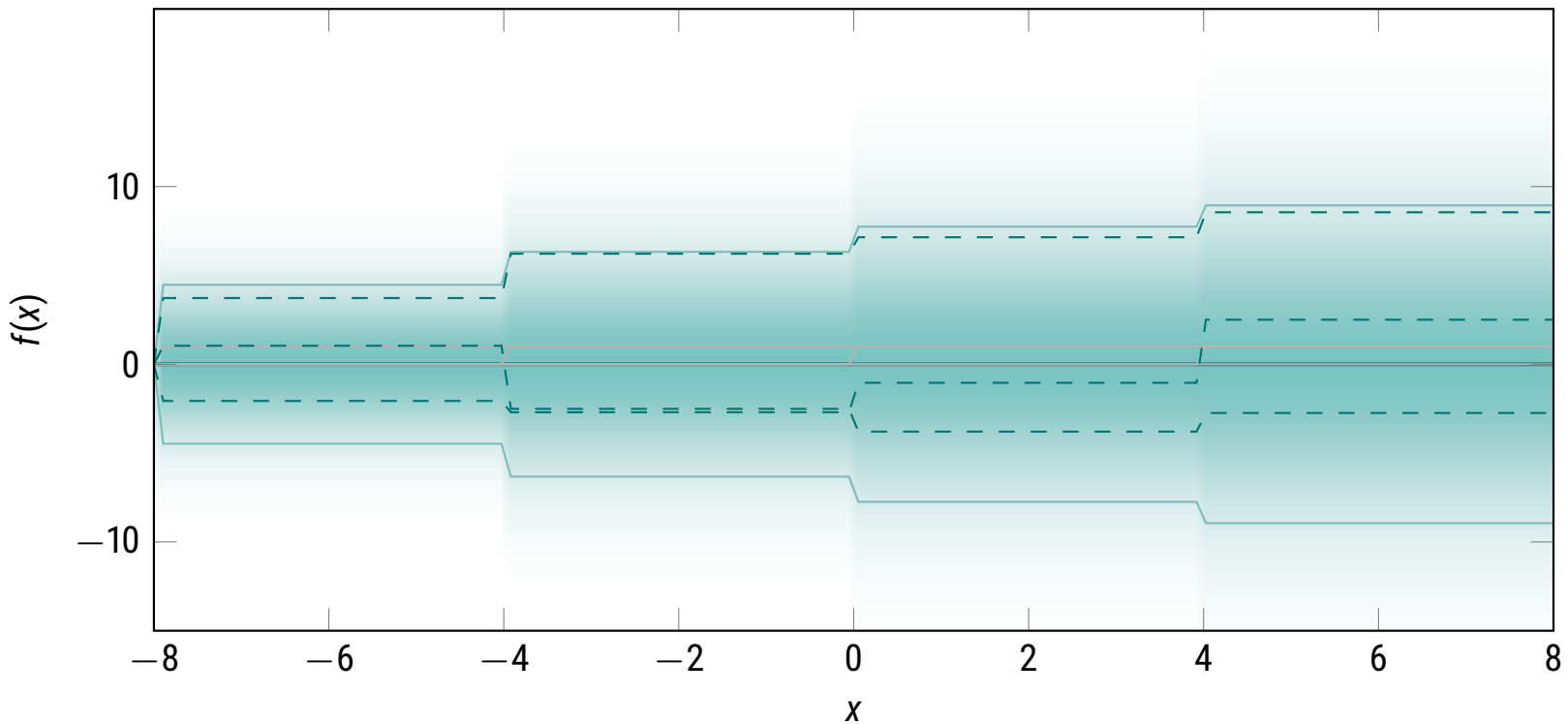
$$k(x, x') = \sum_{\ell \in \mathcal{L}} \phi_\ell(x)\phi_\ell(x') \quad \text{or} \quad = \int_{\mathcal{L}} \phi_\ell(x)\phi_\ell(x')\, d\nu(\ell)$$

Proof: $\forall X \in \mathbb{X}^N, v \in \mathbb{R}^N : v^\intercal k_{XX} v = \int \sum_i^N v_i \phi_\ell(x_i) \sum_j^N v_j \phi_\ell(x_j)\, d\nu(\ell) = \int \left[\sum_i v_i \phi_\ell(x_i)\right]^2 d\nu(\ell) \geq 0$ □

Let $\mu : \mathbb{X} \to \mathbb{R}$ be any function, $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ be a Mercer kernel.
A **Gaussian process** $p(f) = \mathcal{GP}(f; \mu, k)$ is a probability distribution over the function $f : \mathbb{X} \to \mathbb{R}$, such that every finite restriction to function values $f_X := [f_{x_1}, \dots, f_{x_N}]$ is a Gaussian distribution $p(f_X) = \mathcal{N}(f_X; \mu_X, k_{XX})$.

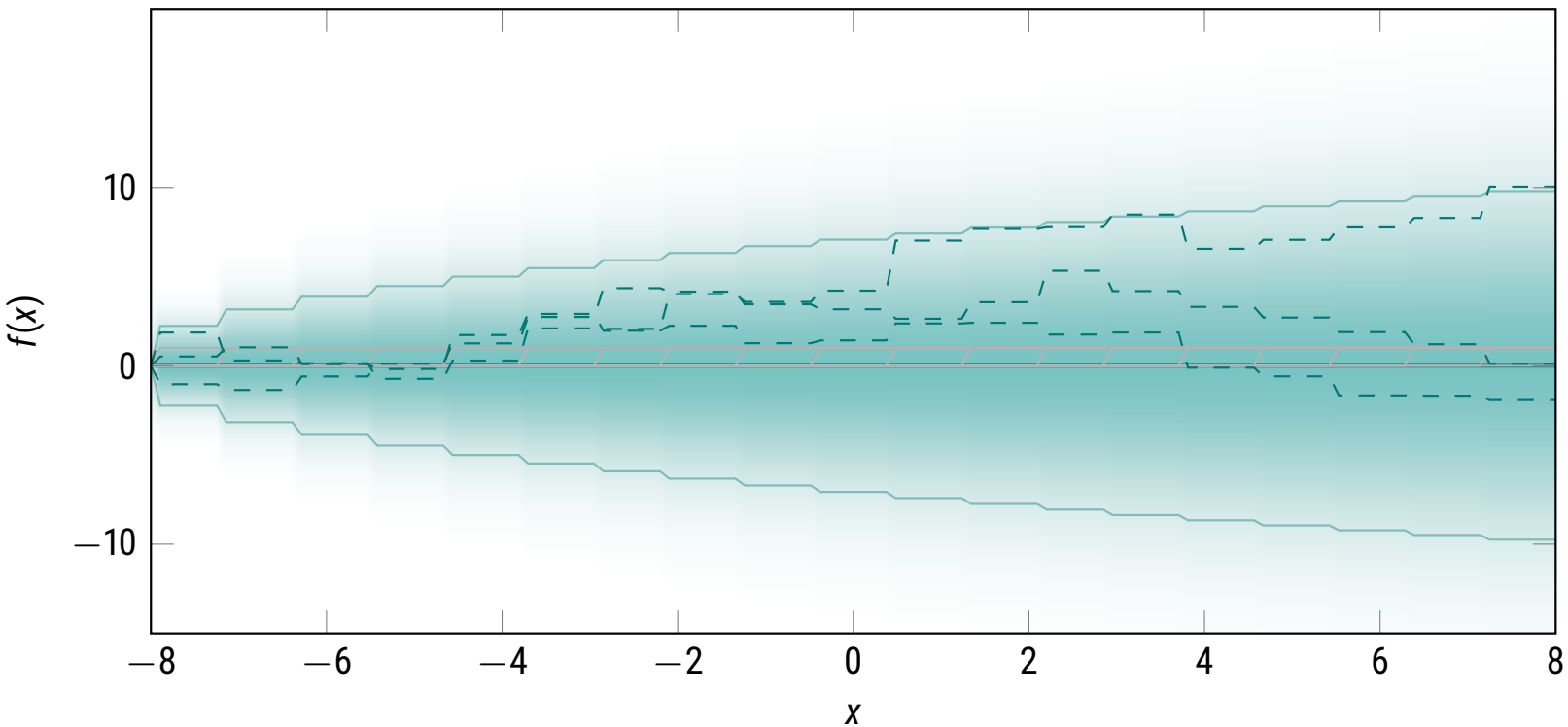
Those Step Functions

```
phi = @(a)(bsxfun(@gt,a,linspace(-8,8,5))./sqrt(5));
```



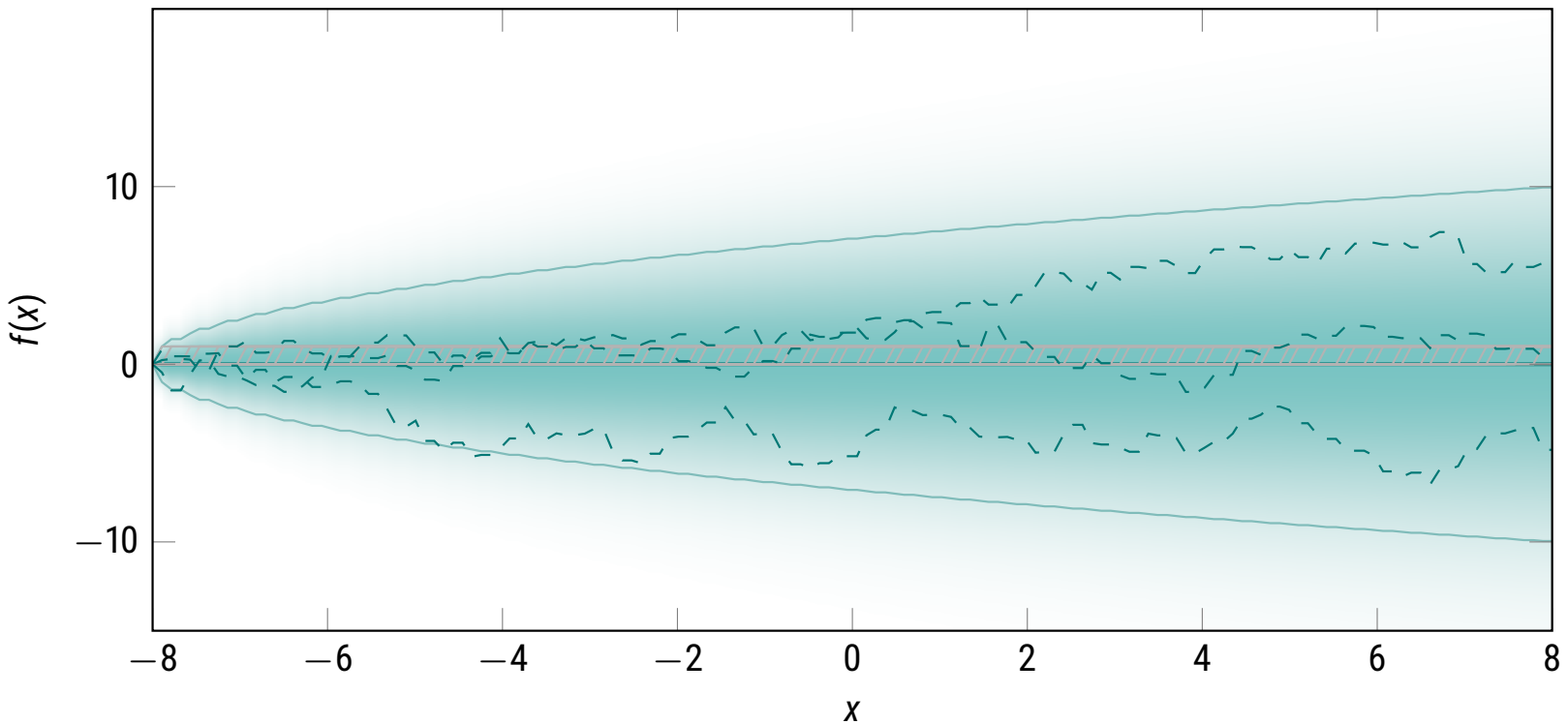
Those Step Functions

```
phi = @(a)(bsxfun(@gt,a,linspace(-8,8,20))./sqrt(20));
```



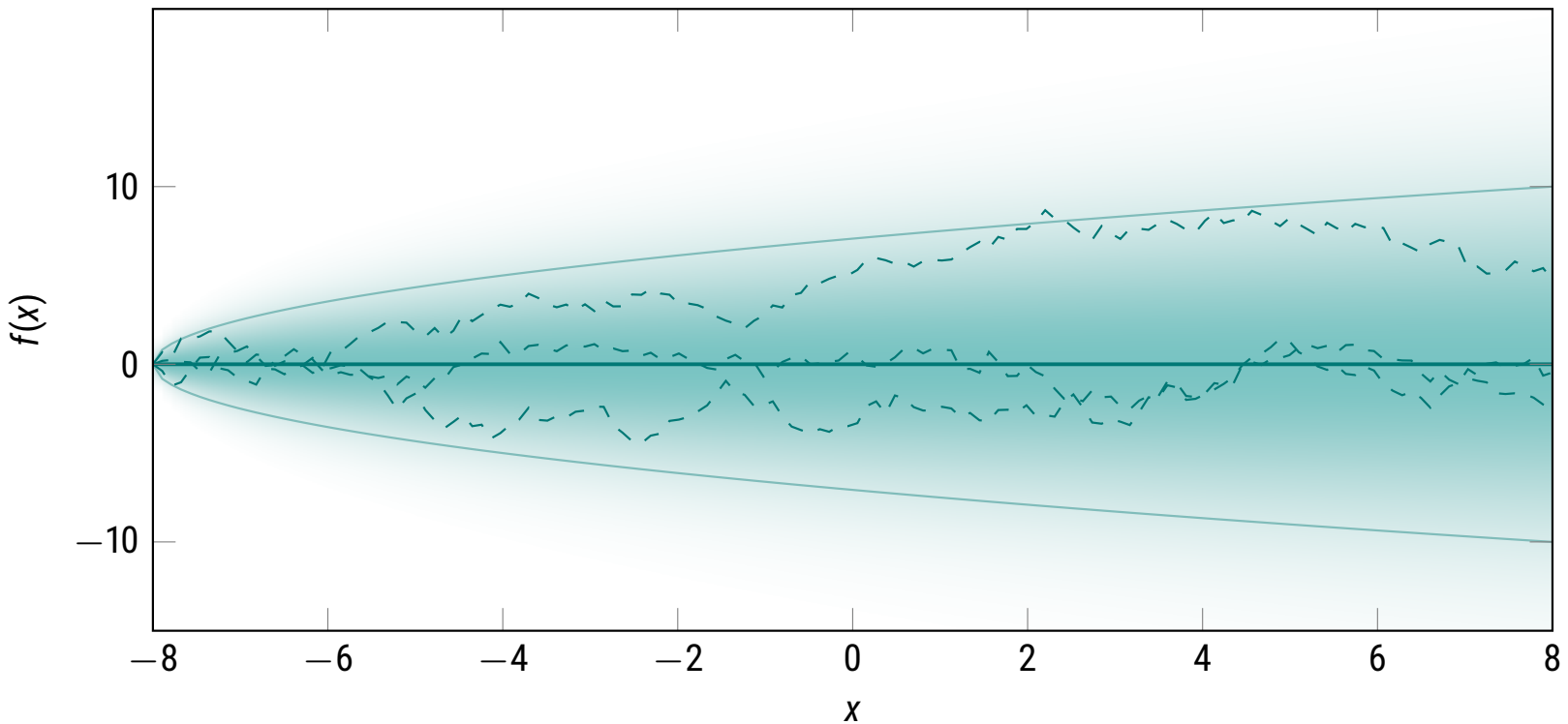
Those Step Functions

```
phi = @(a)(bsxfun(@gt,a,linspace(-8,8,100))./sqrt(100));
```



Those Step Functions

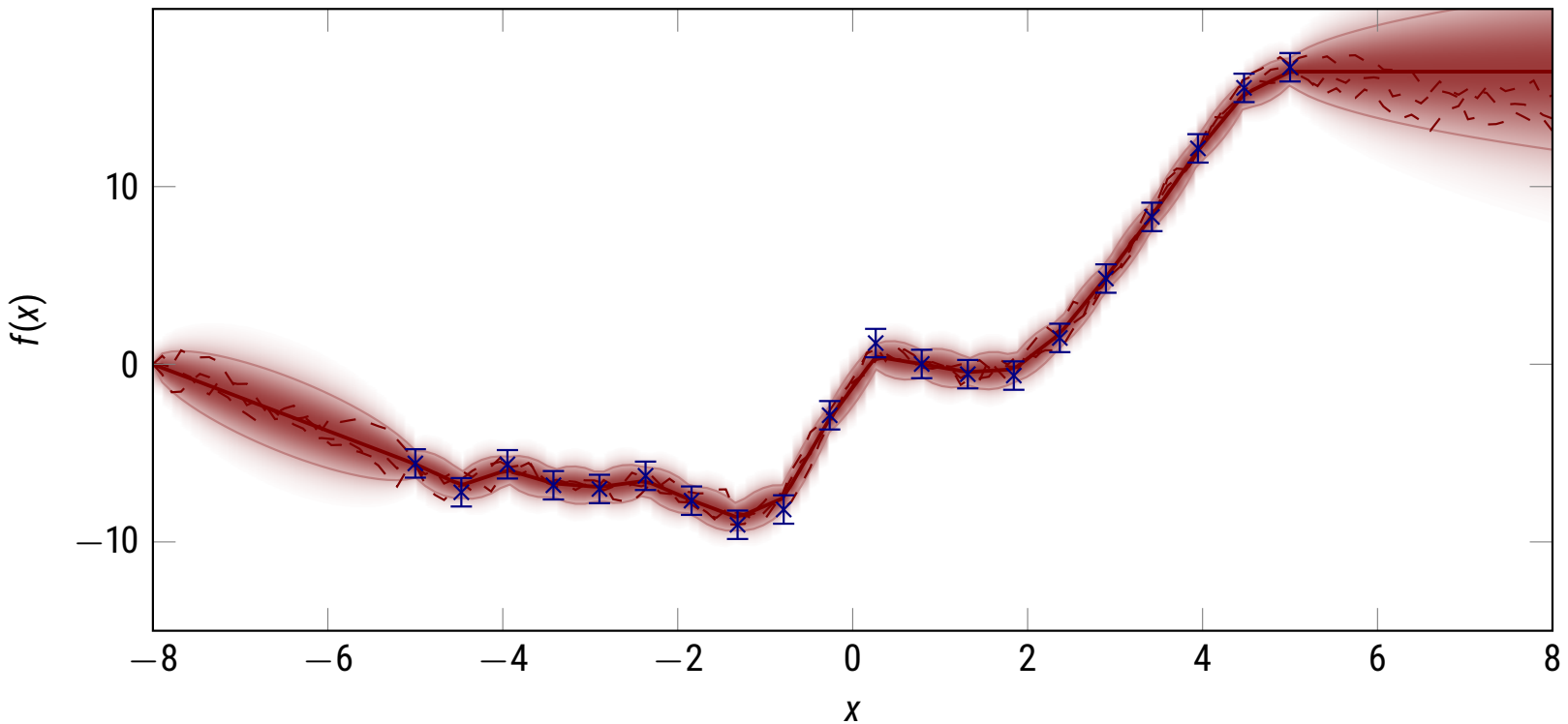
```
k = @(a,b)(theta.^2 * bsxfun(@min,a+8,b'+8)/16);
```



The **Wiener process**.

Those Step Functions

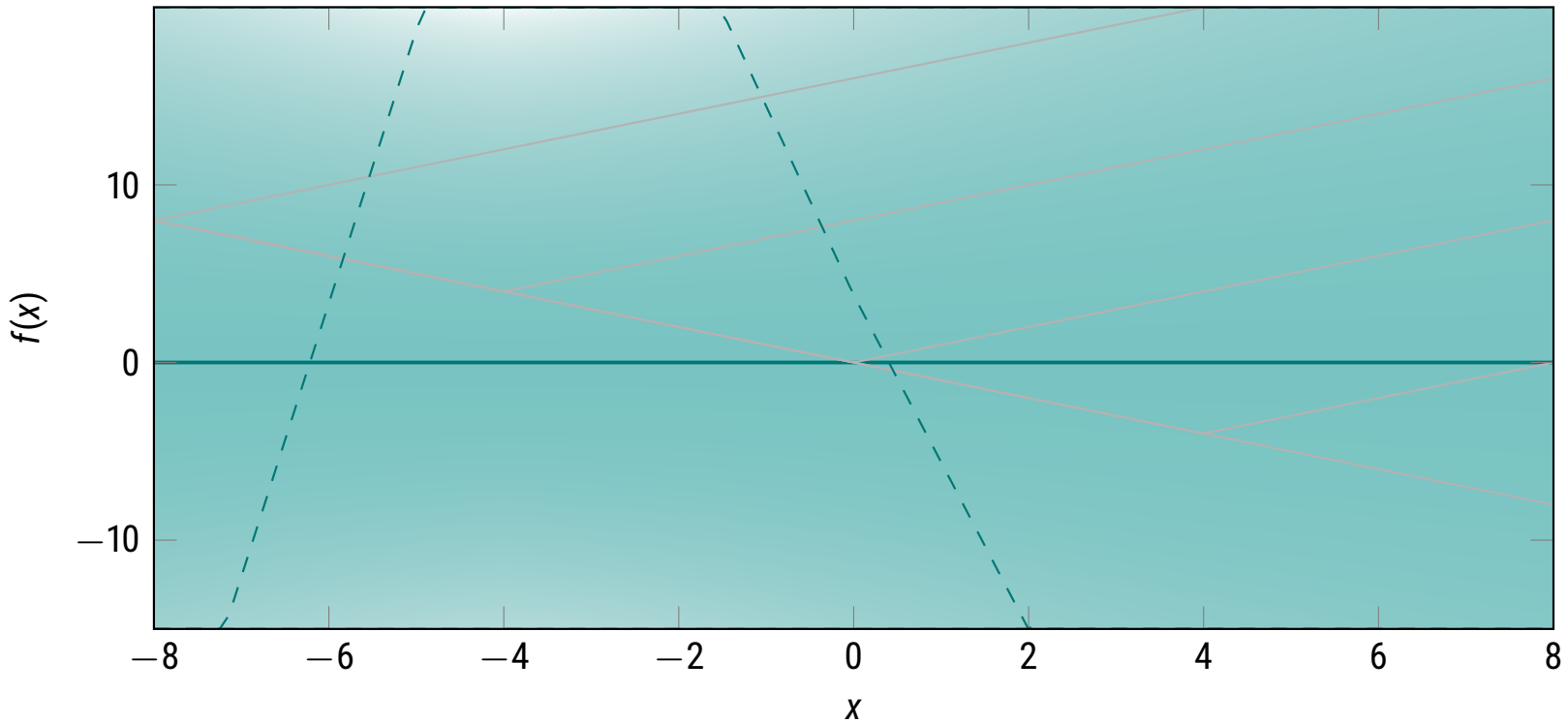
```
k = @(a,b)(theta.^2 * bsxfun(@min,a+8,b'+8)/16);
```



The **Wiener process**.

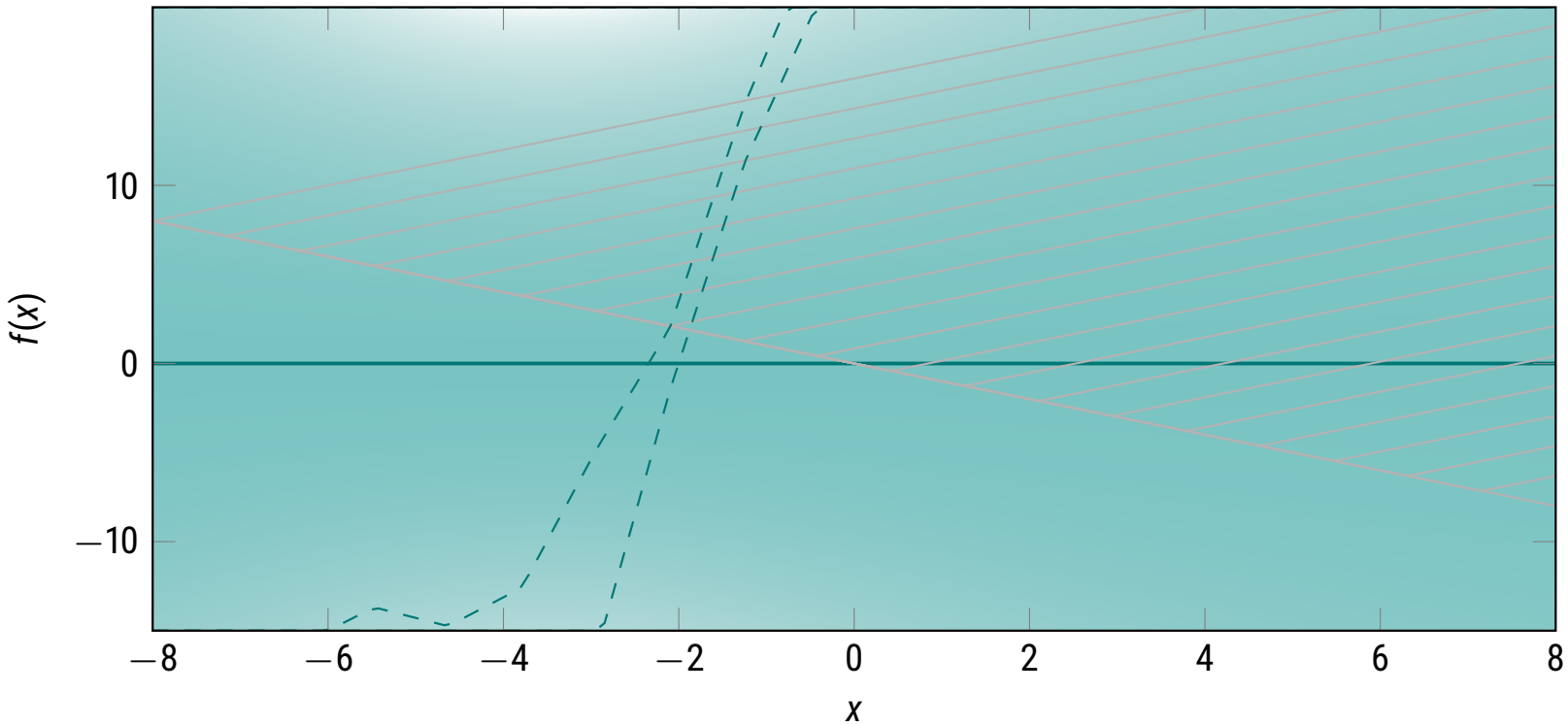
Those linear features

```
phi = @(a)(bsxfun(@minus,abs(bsxfun(@minus,a,linspace(-8,8,5))),linspace(-8,8,5)));
```



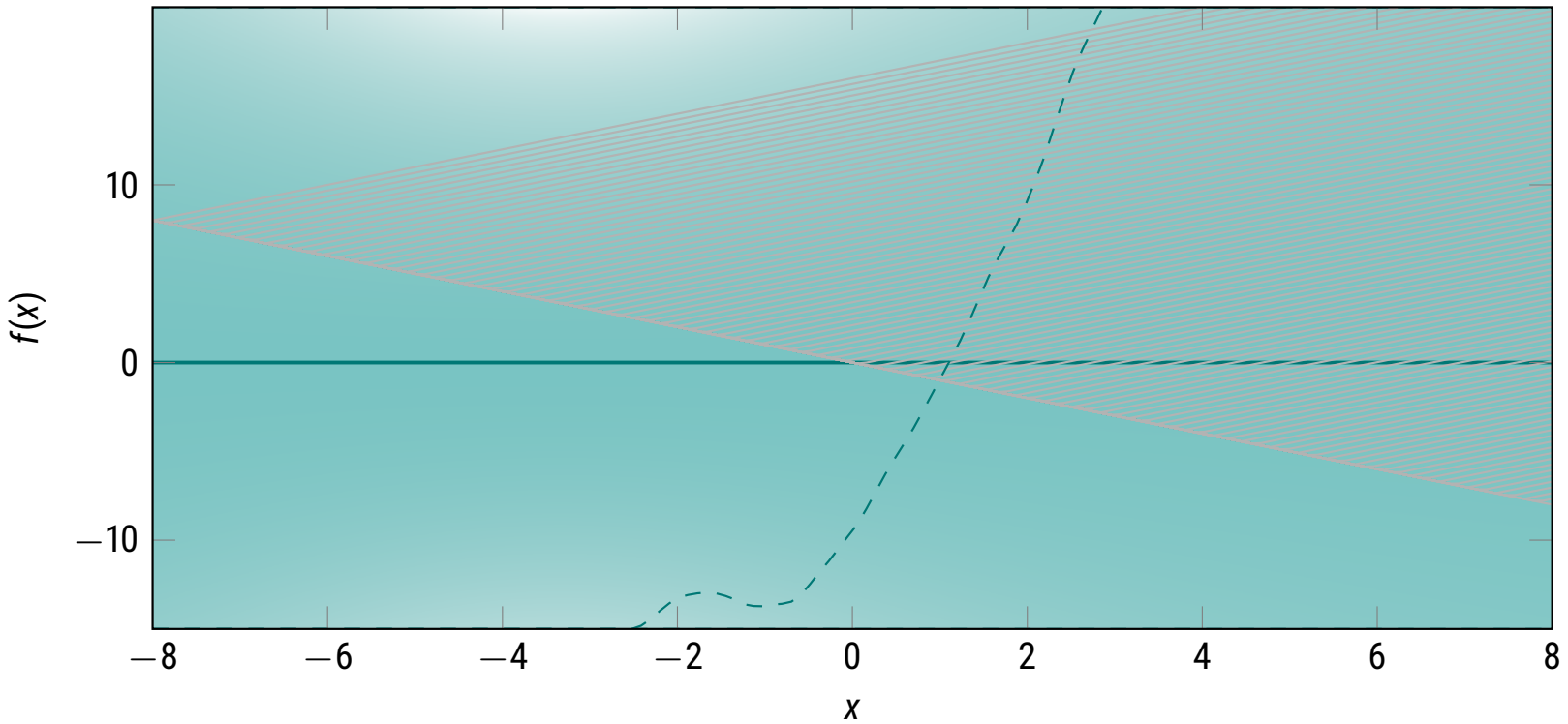
Those linear features

```
phi = @(a)(bsxfun(@minus,abs(bsxfun(@minus,a,linspace(-8,8,20))),linspace(-8,8,20)));
```



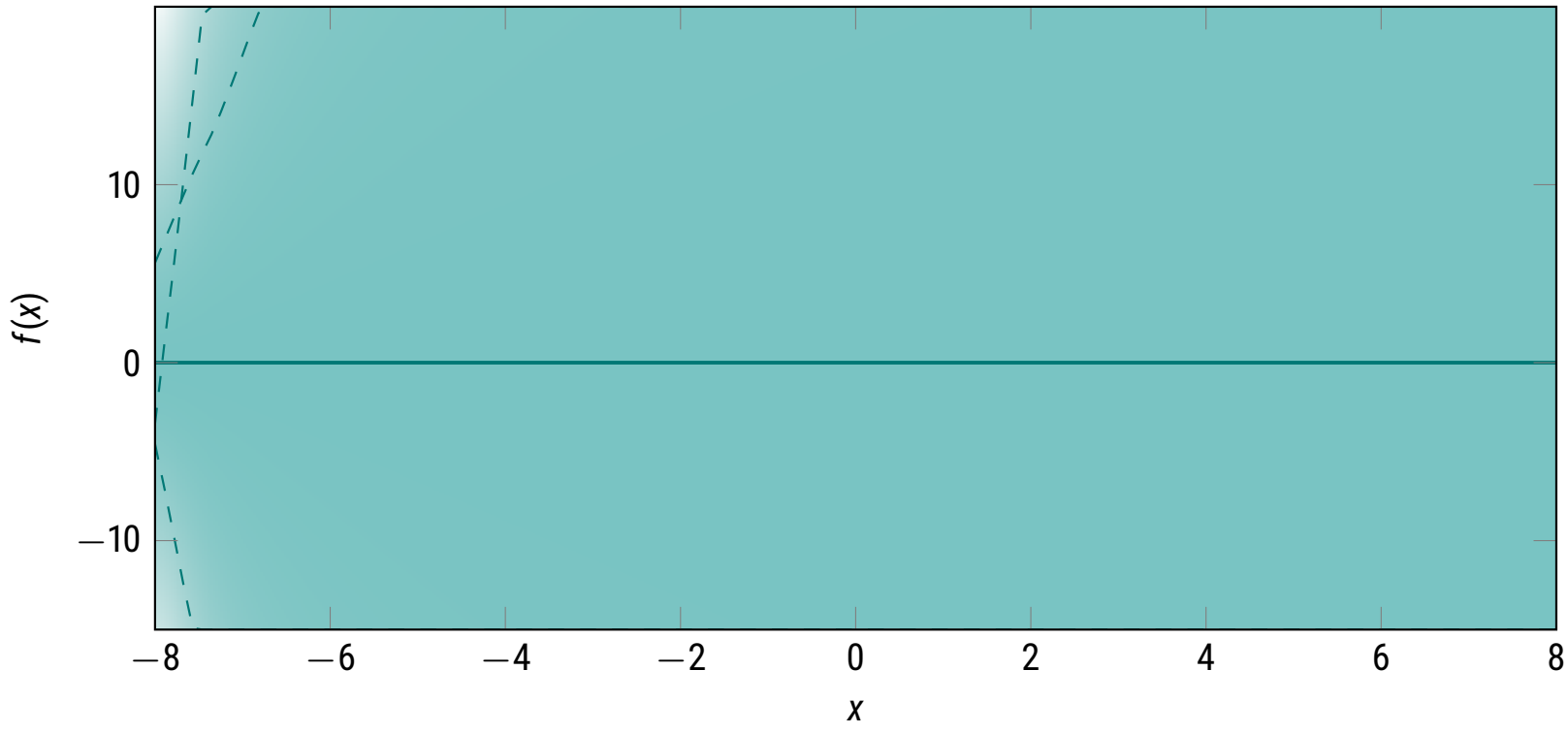
Those linear features

```
phi=@(a)(bsxfun(@minus,abs(bsxfun(@minus,a,linspace(-8,8,100))),linspace(-8,8,100)));
```



Those linear features

```
k = @(a,b)(theta.^2 * (1 + (1+c) * bsxfun(@times,a+8,b'+8)./16 + c ./ 3 *
(abs(bsxfun(@minus,a,b')/16).^3 - bsxfun(@plus,((a+8)./16).^3,((b'+8)./16).^3))));
```



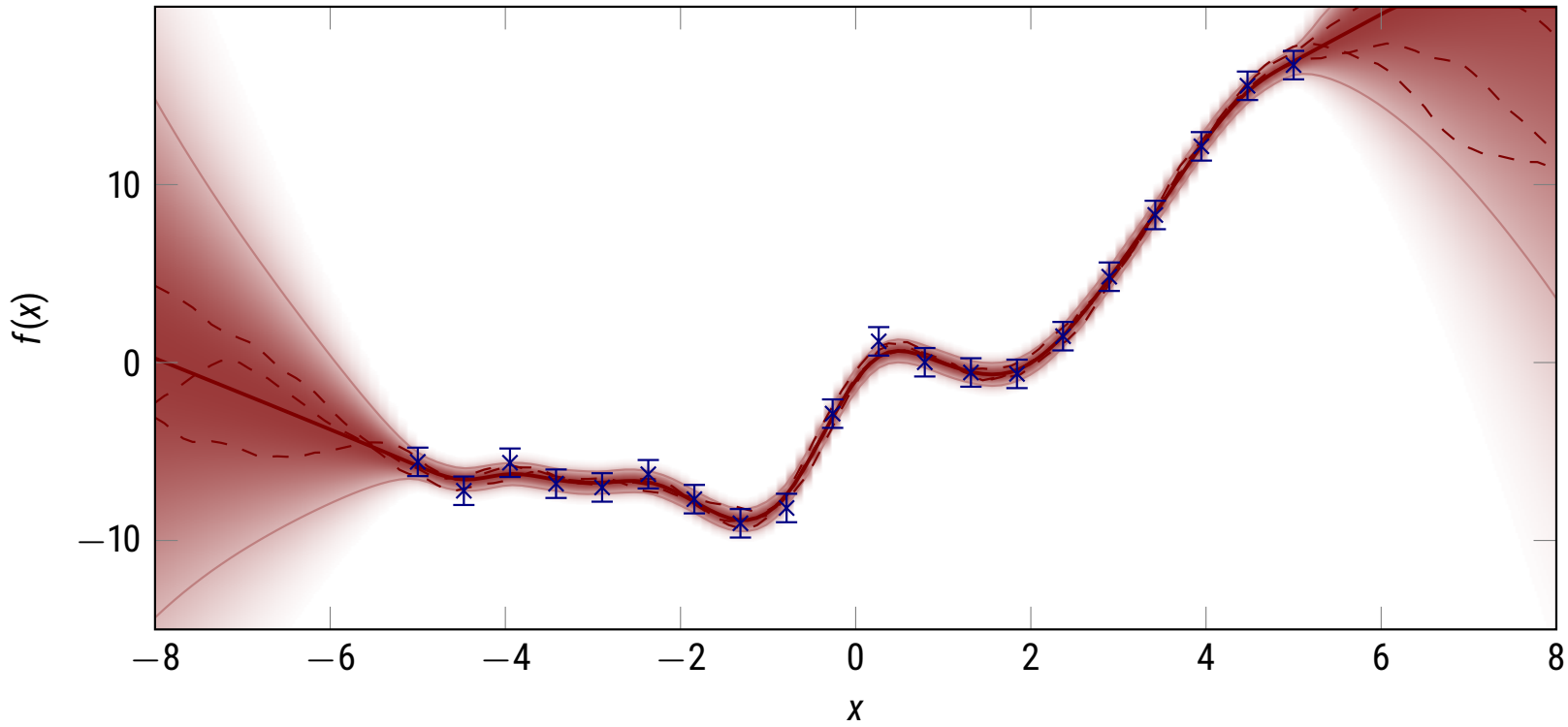
$$\operatorname{cov}(f_{x_i}, f_{x_j}) = 1 + x_i x_j + b \int_0^1 (|x_i - c| - c)(|x_j - c| - c)\,dc$$

$$= 1 + (1 + b)x_i x_j + \frac{b}{3}(|x_i - x_j|^3 - x^3 - y^3)$$

Cubic Splines.

Those linear features

```
k = @(a,b)(theta.^2 * (1 + (1+c) * bsxfun(@times,a+8,b'+8)./16 + c ./ 3 *
(abs(bsxfun(@minus,a,b')/16).^3 - bsxfun(@plus,((a+8)./16).^3,((b'+8)./16).^3))));
```



$$\mathrm{cov}(f_{x_i}, f_{x_j}) = 1 + x_i x_j + b \int_0^1 (|x_i - c| - c)(|x_j - c| - c)c$$

$$= 1 + (1 + b)x_i x_j + \frac{b}{3}(|x_i - x_j|^3 - x^3 - y^3)$$ **Cubic Splines**.

- Gaussian measures turn probabilistic inference into **linear algebra**
- Gaussian measures can be used to infer **functions** and **curves**
- In fact, the number of inferred features / weights can be **infinite**

Next up:

- There is a **large space** of kernels, and thus GP priors
- The choice of kernel / GP prior affects **model capacity** and **convergence**.

GPs are a flexible model class

because the space of kernels is large

Theorem:

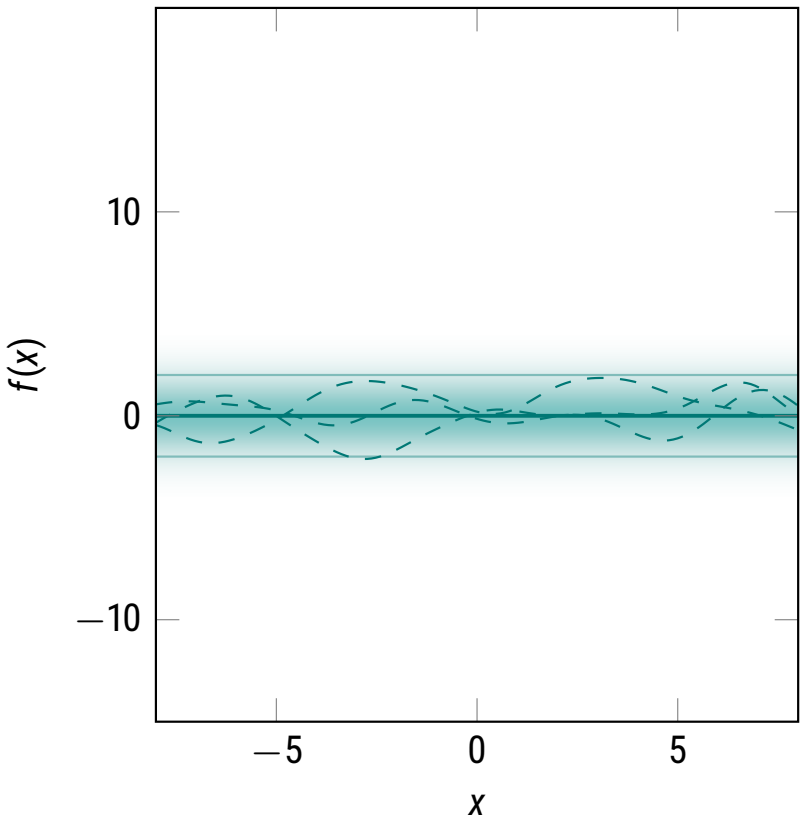
Let $\mathbb{X}, \mathbb{Y}$ be index sets and $\phi : \mathbb{Y} \to \mathbb{X}$. If $k_1, k_2 : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ are Mercer kernels, then the following functions are also Mercer kernels (up to minor regularity assumptions)

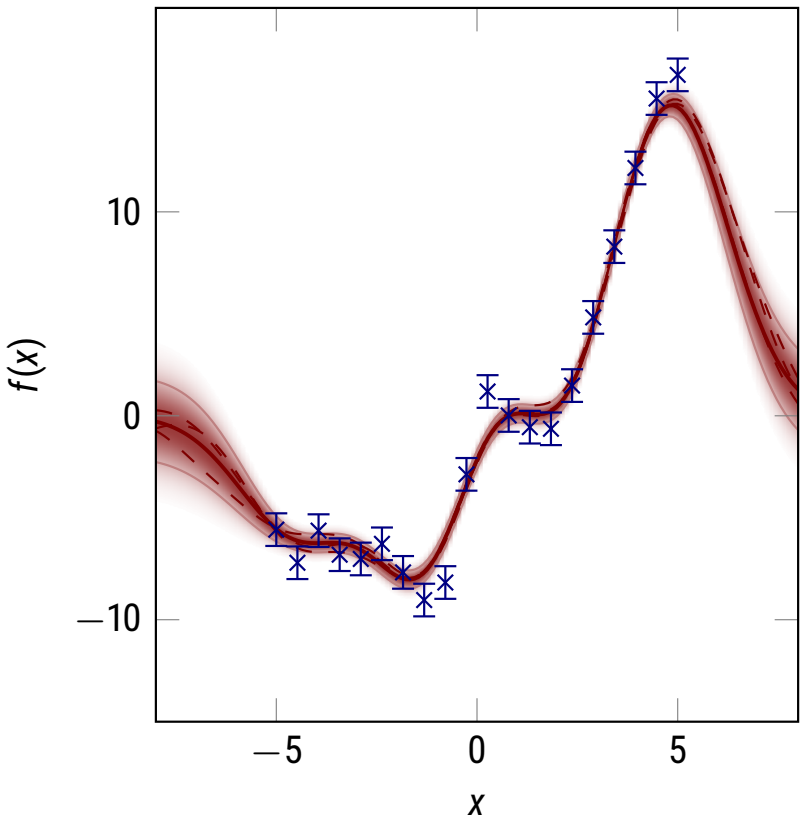
- $\alpha \cdot k_1(a, b)$ for $\alpha \in \mathbb{R}_+$ (proof: trivial)
- $k_1(\phi(c), \phi(d))$ for $c, d \in \mathbb{Y}$ (proof: by Mercer's theorem, coming up)
- $k_1(a, b) + k_2(a, b)$ (proof: trivial)
- $k_1(a, b) \cdot k_2(a, b)$ **Schur product theorem** (proof e.g. in Bapat, 1997. Million, 2007)

Scaled Kernels are Kernels

(this affects the effect of noise)

$$k(a, b) = 1^2 \cdot \exp\left(-\frac{(a-b)^2}{2}\right)$$

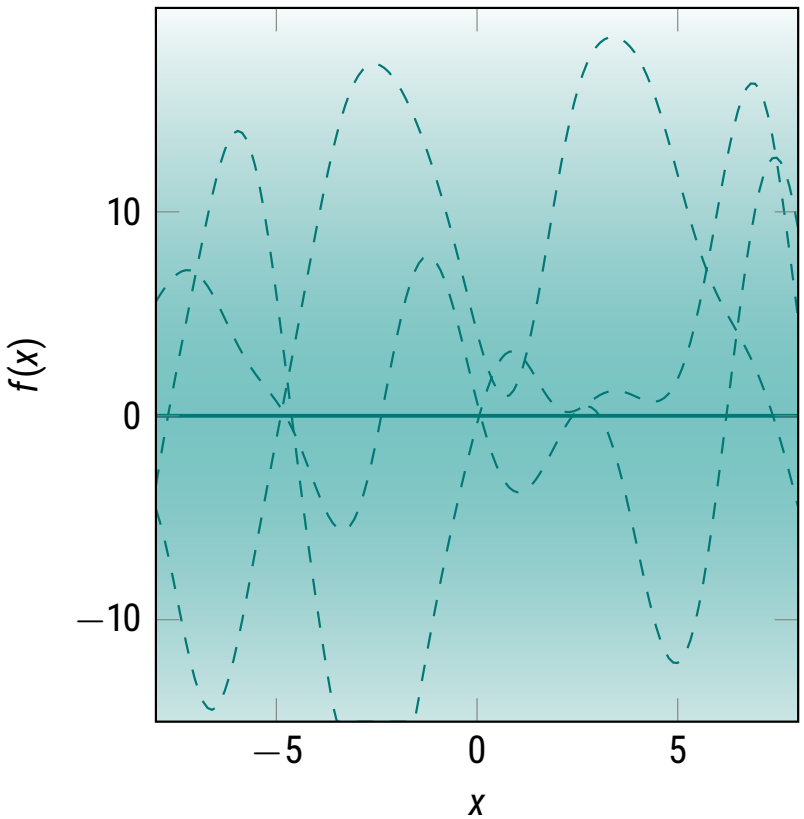


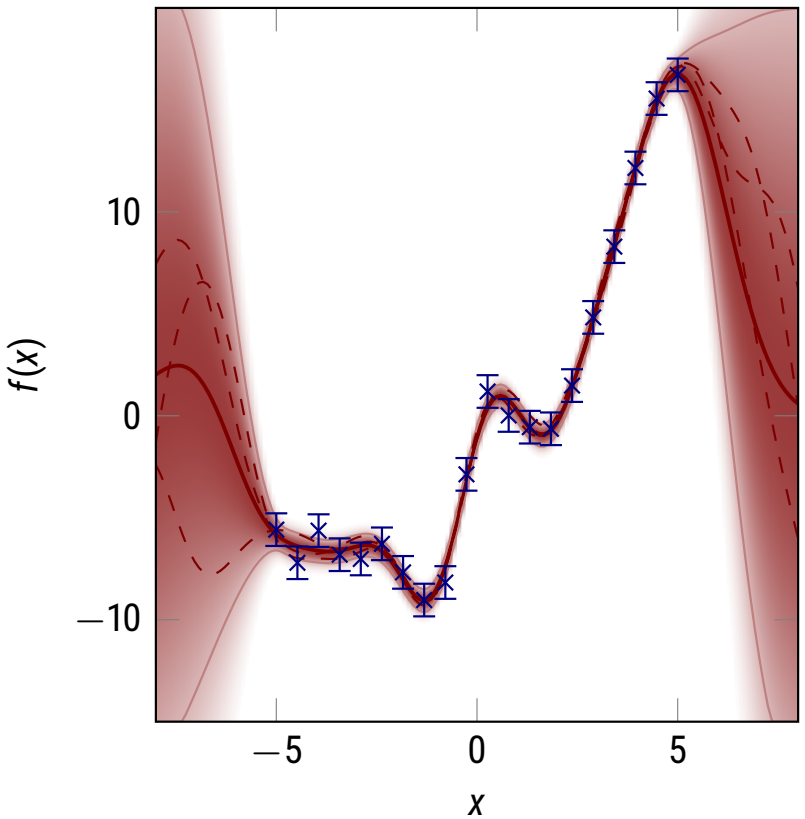


Scaled Kernels are Kernels

(this affects the effect of noise)

$$k(a, b) = 10^2 \cdot \exp\left(-\frac{(a-b)^2}{2}\right)$$

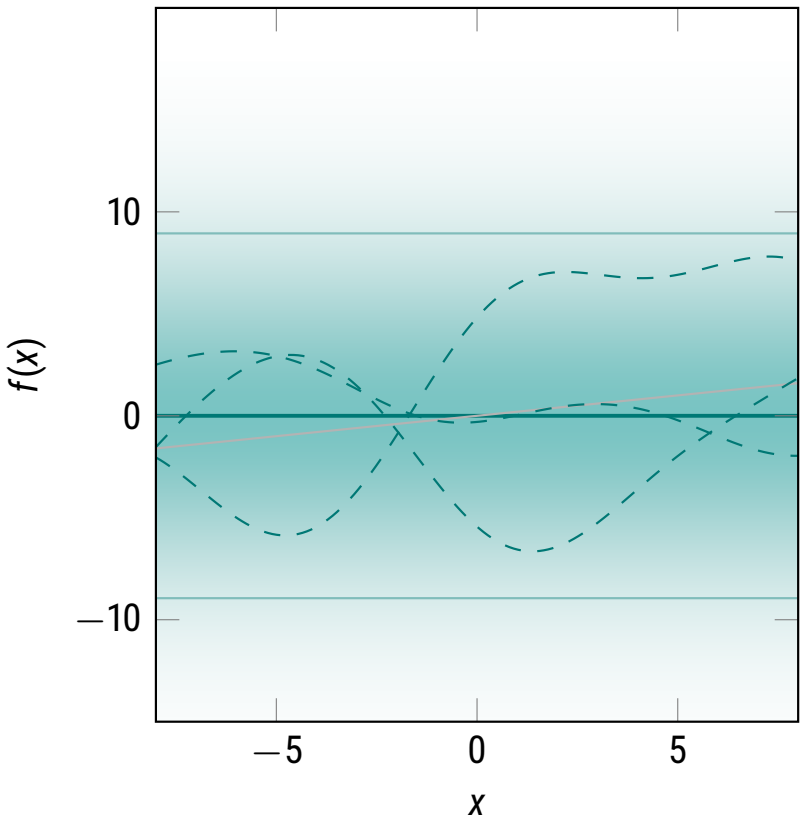


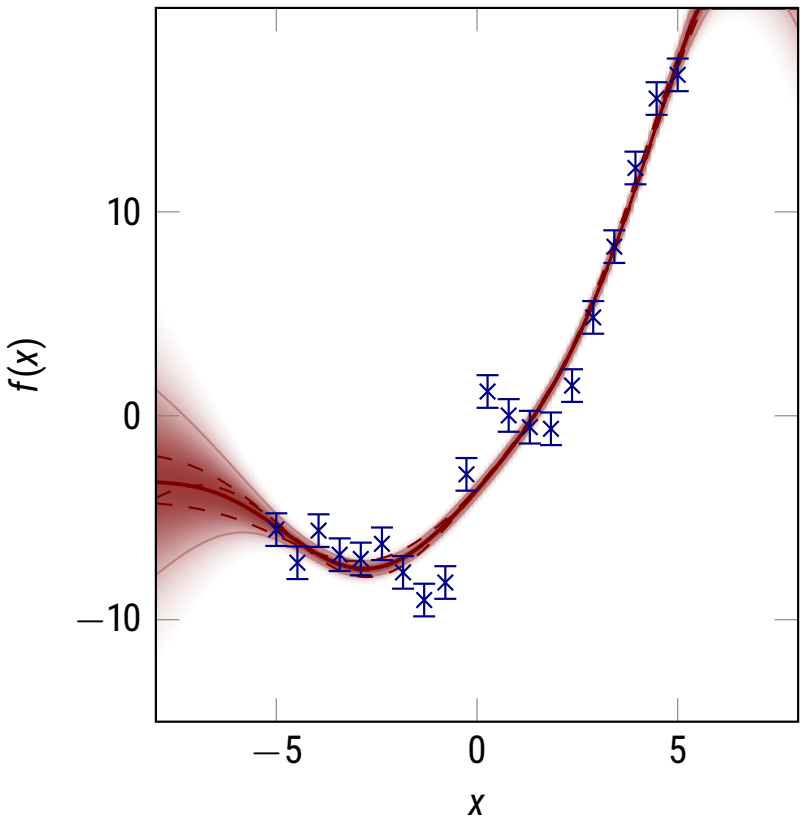


Kernels over Transformed Inputs are Kernels

this can be used to define length-scales

$$k(a, b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2 \cdot 5^2}\right)$$

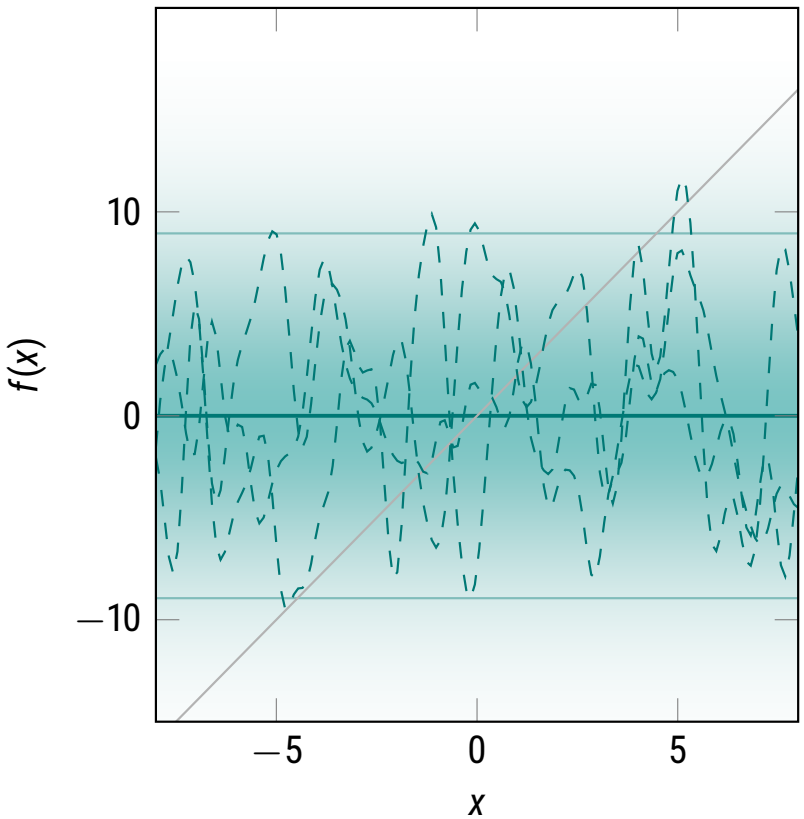


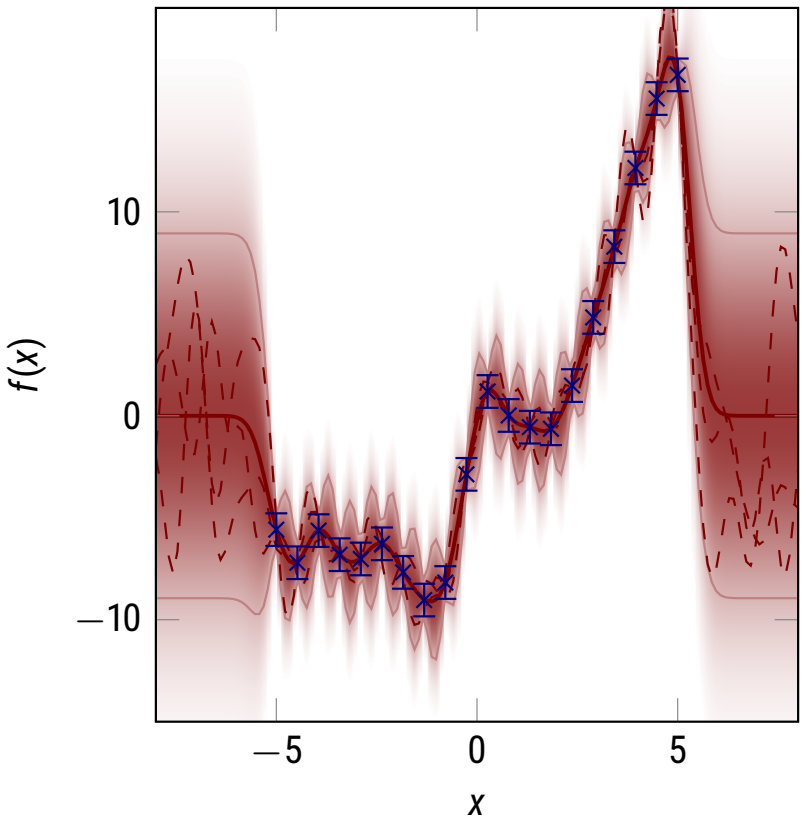


Kernels over Transformed Inputs are Kernels

this can be used to define length-scales

$$k(a, b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2 \cdot 0.5^2}\right)$$

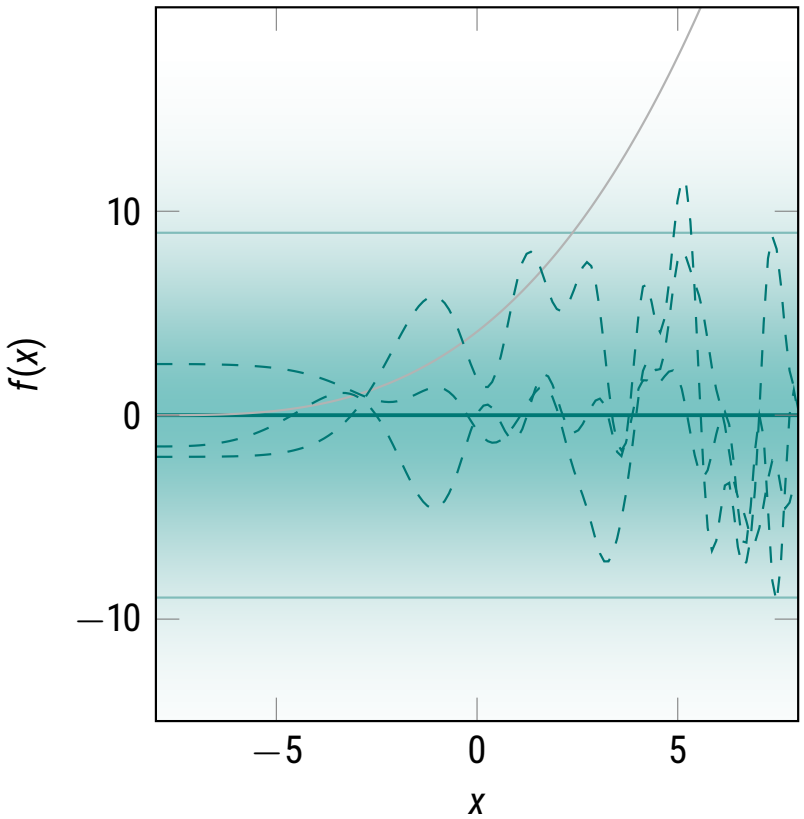


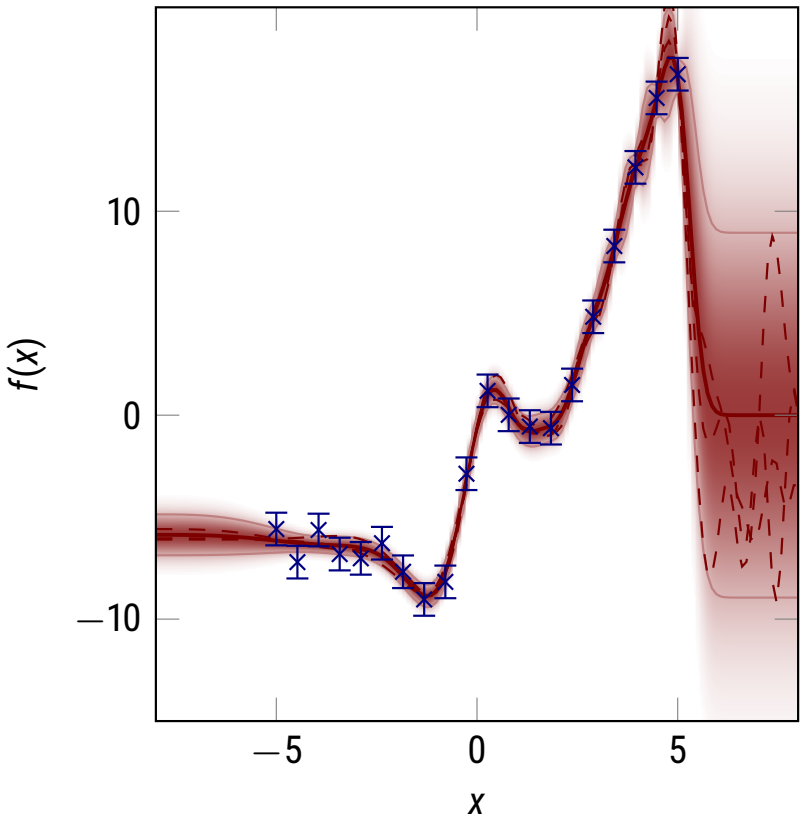


Kernels over Transformed Inputs are Kernels

nonlinear transformations (also at the heart of deep learning)

$$k(a,b) = 20 \cdot \exp\left(-\frac{(\phi(a)-\phi(b))^2}{2}\right) \qquad \phi(a) = \left(\frac{a+8}{5}\right)^3$$

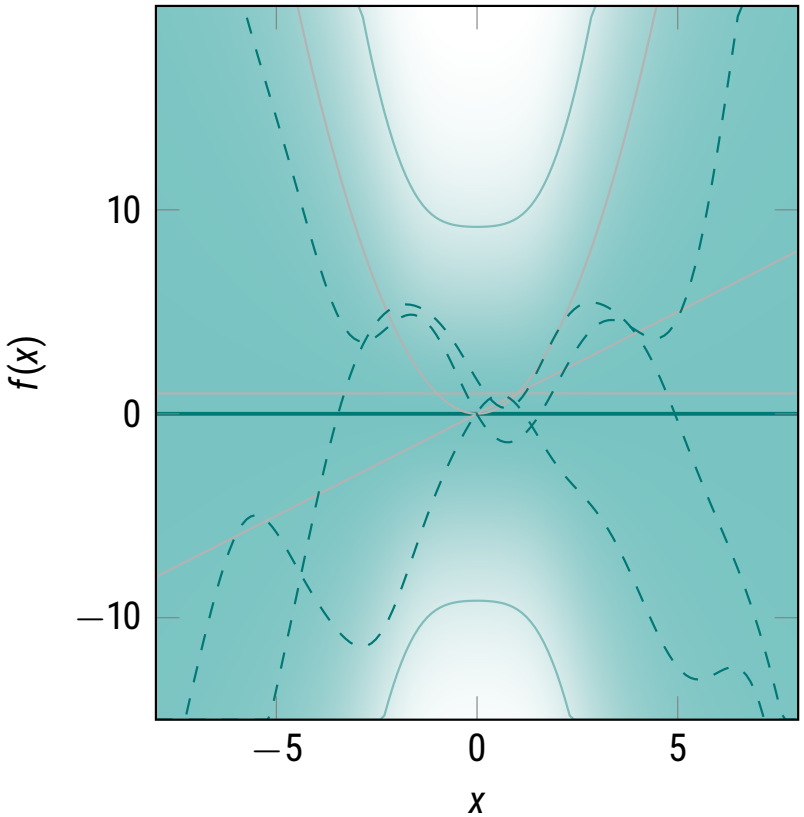


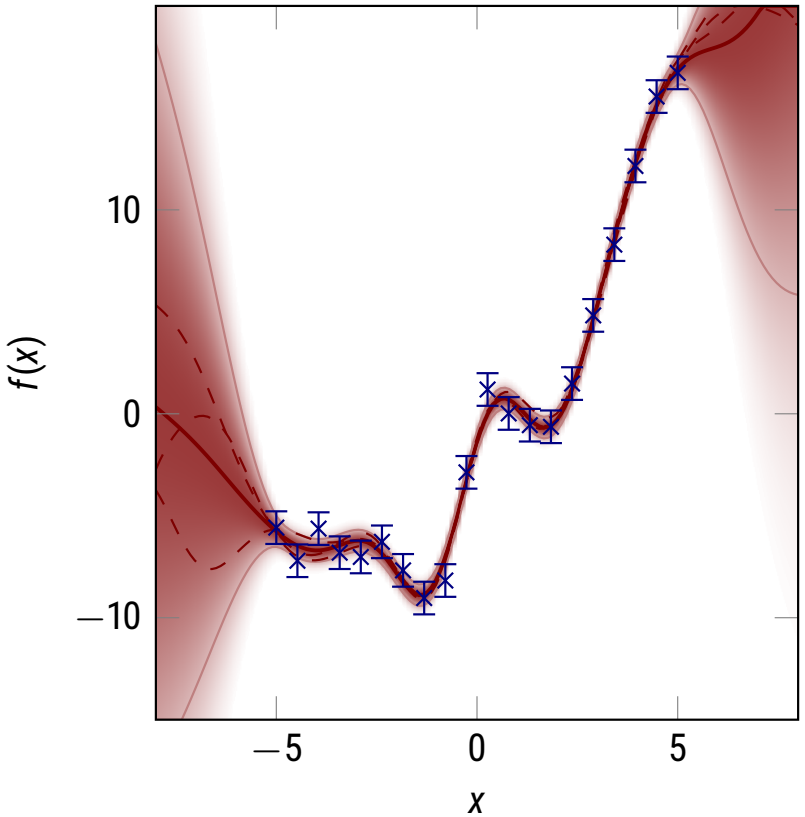


Sums of Kernels are Kernels

$f \sim \mathcal{GP}(0, k_1)$ and $g \sim \mathcal{GP}(0, k_2) \quad \Leftrightarrow \quad h = f + g \sim \mathcal{GP}(0, k_1 + k_2)$

$$k(a, b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2}\right) + \phi(a)^\intercal \phi(b) \qquad \phi(x) := [1, x, x^2]$$

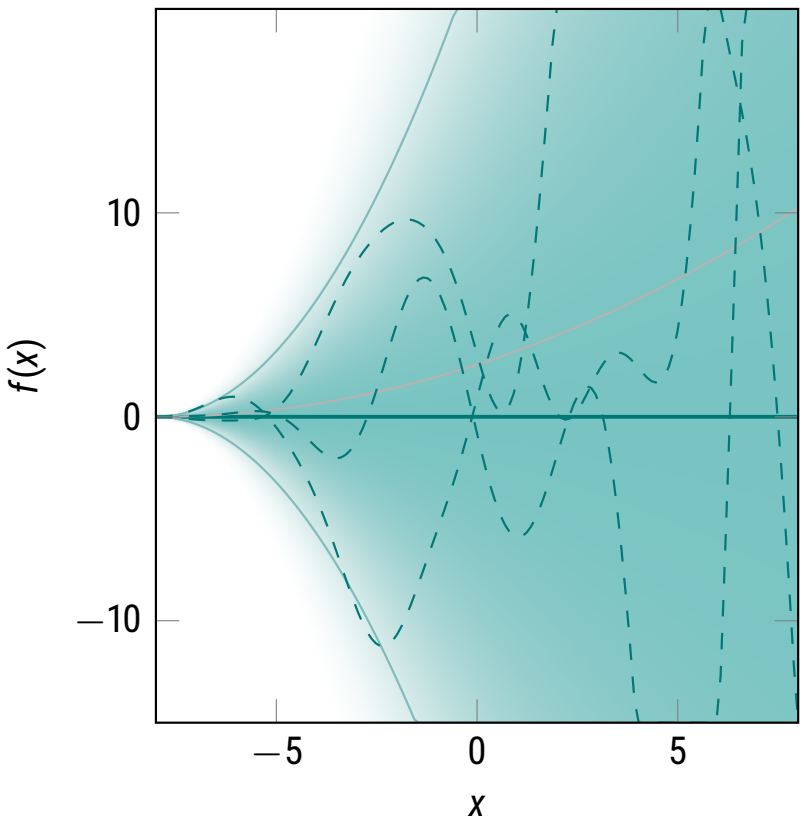


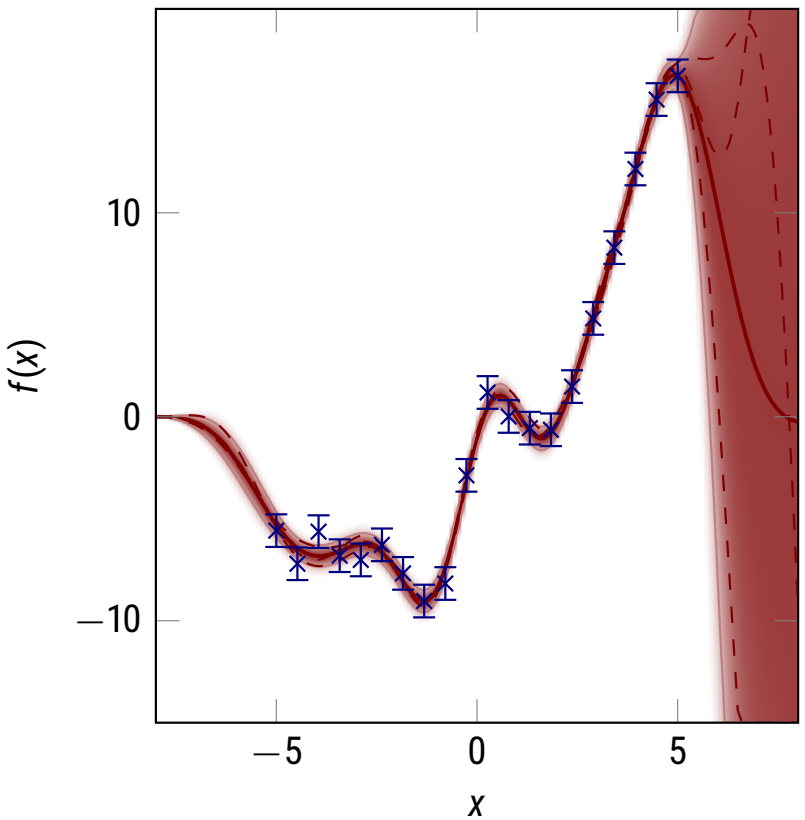


Products of Kernels are Kernels

Schur product theorem

$$k(a,b) = 20 \cdot \exp\left(-\frac{(a-b)^2}{2}\right) \cdot \phi(a)^\intercal \phi(b) \quad \phi(x) := \left(\frac{x+8}{5}\right)^2$$

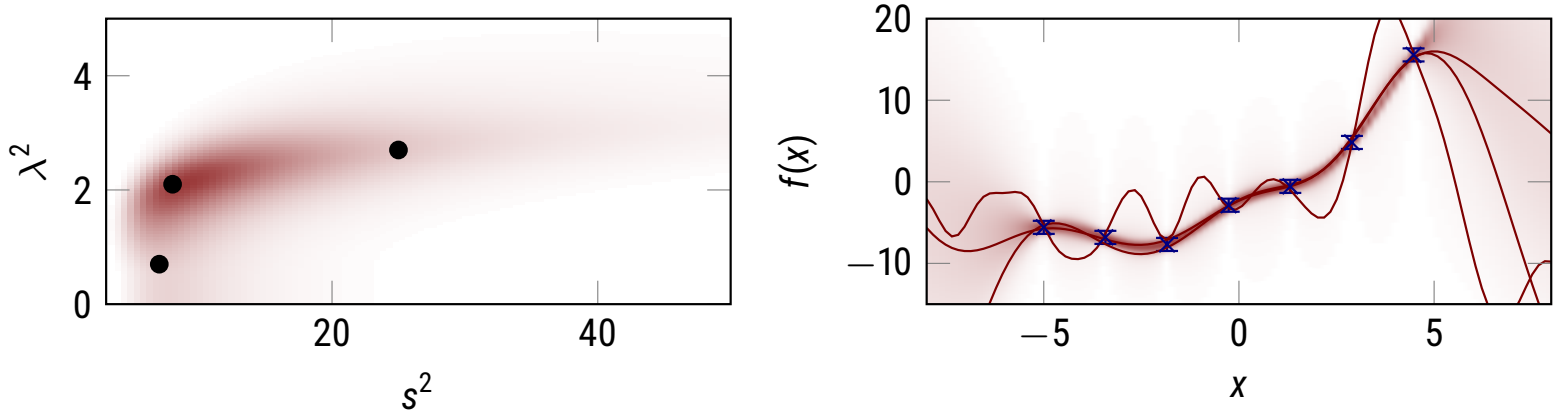




If you think your model has no parameters, you just haven't found them yet!

Can one **learn** the kernel?

hierarchical Bayesian inference



- So there are a lot of choices to be made. Can we **learn** the right kernel?
- In principle, Bayesian inference remains the right framework:
  Parametrize $k = k^\theta, \mu = \mu^\theta, \Lambda = \Lambda^\theta$

$$p(y \mid \theta) = \int p(y \mid f, \theta) p(f \mid \theta)\, df = \int \mathcal{N}(y; f_X, \Lambda^\theta) \mathcal{GP}(f; \mu^\theta, k^\theta)$$

$$= \mathcal{N}(y, \mu_X^\theta, \Lambda^\theta + k_{XX}^\theta)$$

$$p(f \mid y) = \int p(f \mid y, \theta) p(\theta \mid y)\, d\theta$$

- in general, none of this is of closed form (but see later for a special case)

What does the GP prior mean?

Relation to Least-Squares Fitting, and other model classes

GP regression is also known as

- Kriging
- Wiener-Kolmogorov prediction

and is closely related to

- Kernel Ridge Regression
- Kernel Least Squares

In particular, the posterior mean function

$$\mu(x) = k_{xX}(k_{XX} + \sigma^2 I)^{-1} Y$$

is **equal** to the kernel ridge estimate.

What does the GP prior mean?

GP **means** are elements of RKHSs.

Let $\mathcal{H} = (\mathbb{X}, \langle\cdot,\cdot\rangle)$ be a Hilbert space of functions $f : \mathbb{X} \to \mathbb{R}$. Then $\mathcal{H}$ is called a **reproducing kernel Hilbert space** if there exists a **kernel** $k : \mathbb{X} \times \mathbb{X} \to \mathbb{R}$ s.t.

$$\forall x \in \mathbb{X}: \quad k(\cdot, x) \in \mathcal{H} \qquad \text{and} \qquad \forall f \in \mathcal{H}: \quad \langle f(\cdot), k(\cdot, x)\rangle_{\mathcal{H}} = f(x)$$

An **eigenfunction** ϕ_i of k relative to measure ν is a function in $\mathcal{H}$ such that

$$\int \phi_i(x) k(x, \tilde{x})\, d\nu(x) = \lambda_i \phi_i(\tilde{x}).$$

- Aronszajn, 1950: For every pos.def. k on $\mathbb{X}$, there exists a unique RKHS.
- Mercer, 1909: Every k can be written as

$$k(x, \tilde{x}) = \sum_i \lambda_i \phi_i(x) \phi^\intercal(\tilde{x}) \quad \text{with} \quad \sum_i \lambda_i < \infty \tag{1}$$

- The RKHS is the space of functions

$$f(x) = \sum_i \alpha_i \phi_i(x) \text{ with } \sum_i \frac{\alpha_i^2}{\lambda_i} < \infty, \text{ where } \langle f, g\rangle_{\mathcal{H}} = \sum_i \frac{\alpha_i \beta_i}{\lambda_i}.$$

Proof: $\langle f(\cdot), k(\cdot, x)\rangle = \sum_i \alpha_i \lambda_i / \lambda_i \phi_i(x) = f(x)$. And $\|k(x,\cdot)\| = \langle k(x,\cdot), k(x,\cdot)\rangle = k(x,x) < \infty$. □

What does the GP prior mean?

GP **samples** are not in the RKHS. But **almost**.

Karhunen-Loève Expansion:

$$f(x) = \sum_i \alpha_i \phi_i \qquad \alpha_i \overset{iid.}{\sim} \mathcal{N}(0, \lambda_i) \quad \Leftrightarrow \quad f(x) \sim \mathcal{GP}(0, k)$$

Thus: $f \sim \mathcal{GP}(0, k) \quad \Rightarrow \quad \mathsf{E}(\|f\|_{\mathcal{H}}^2) = \sum_i \frac{\mathsf{E}(\alpha_i^2)}{\lambda_i} = \sum_i 1$

Samples from a GP are not in the RKHS (with probability 1).

Theorem: [Driscoll, 1973; Steinwart, 2014]

Let $f \sim \mathcal{GP}(0, k)$. The following two statements are equivalent:

- $\bar{\mathcal{H}}$ is RKHS such that $\mathcal{H}_k \subset \bar{\mathcal{H}}$ (with Hilbert-Schmidt compact embedding)
- $f \in \bar{\mathcal{H}}$ almost surely.

(Misleading) example: Draws from GP with $k_\ell(x, x') = \exp\left(-\frac{(x-x')^2}{2\ell^2}\right)$ are not in RKHS of k. But they are in the RKHS of $k_{\tilde{\ell}}$ for all $\tilde{\ell} < \ell$.

What does the posterior variance mean?

Frequentist interpretations of Bayesian error estimates

How far could the ridge regressor / posterior mean be from the truth?

$$\sup_{f \in \mathcal{H}, \|f\| \leq 1} \left(\sum_i f(x_i) \underbrace{[K_{XX}^{-1} k(X, x)]_i}_{w_i} - f(x) \right)^2 = \sup \left\langle \sum_i w_i k(\cdot, x_i) - k(\cdot, x), f(\cdot) \right\rangle_{\mathcal{H}}^2$$

$$= \left\| \sum_i w_i k(\cdot, x_i) - k(\cdot, x) \right\|_{\mathcal{H}}^2 = \sum_{ij} w_i w_j k(x_i, x_j) - 2 \sum_i w_i k(x, x_i) + k(x, x)$$

$$= k_{xx} - k_{xX} K_{XX}^{-1} k_{Xx} = \mathrm{E}_{|y}[(f_x - \mu_x)^2]$$

What does the GP posterior mean?

Frequentist interpretations of Bayesian error estimates

- the GP posterior mean

$$\mu(x) = \mathrm{E}[f(x)] = k_{xX}(k_{XX} + \sigma^2)^{-1}Y$$

  is the **least-squares** estimate in the RKHS of k.

- **for noise-free observations**, the GP posterior variance

$$\mathbb{V}(x) = \mathrm{E}[f(x)^2 - \mu(x)^2] = k_{xx} - k_{xX}k_{XX}^{-1}k_{Xx}$$

  is the scale of the **worst-case** error for functions of bounded norm in the RKHS.

In later lectures, we will see that for advanced computational problems with **non-Gaussian likelihoods**, additional care is required.

Universal Kernels

a slightly different statement

Definition (Universal Kernel)

*A kernel **k** acting on **X** is said to be universal if its RKHS lies dense in the space of all continuous functions.*

Journal of Machine Learning Research 7 (2006) 2651-2667 Submitted 7/06; Revised 10/06; Published 12/06

Universal Kernels

Charles A. Micchelli CAM@MATH.ALBANY.EDU
Department of Mathematics and Statistics
State University of New York
The University at Albany
Albany, New York 12222, USA

Yuesheng Xu YXU06@SYR.EDU
Haizhang Zhang HZHANG12@SYR.EDU
Department of Mathematics
Syracuse University
Syracuse, NY 13244, USA

Editor: Gabor Lugosi

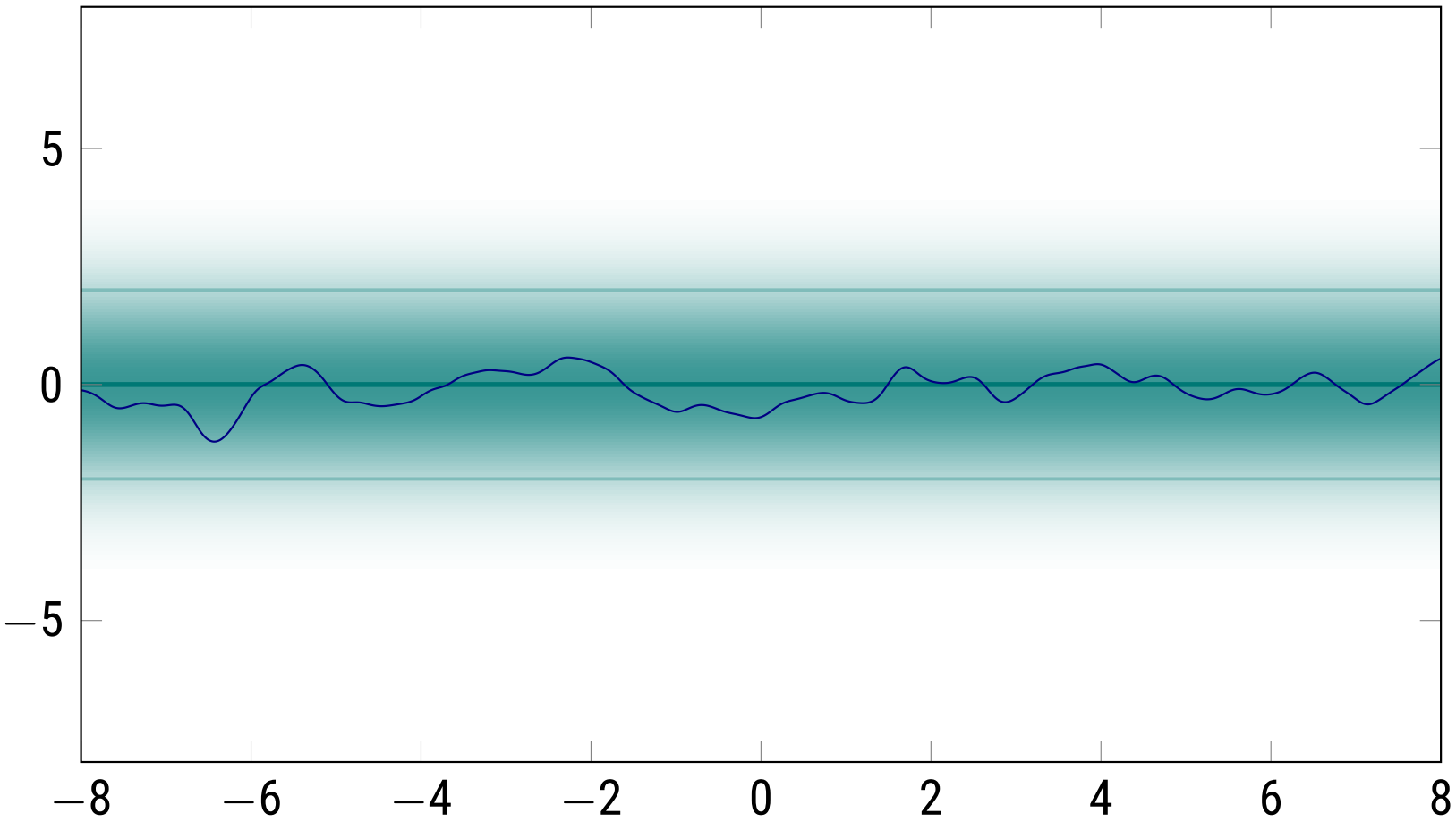
Abstract

In this paper we investigate conditions on the features of a continuous kernel so that it may approximate an arbitrary continuous target function uniformly on any compact subset of the input space. A number of concrete examples are given of kernels with this universal approximating property.

Keywords: density, translation invariant kernels, radial kernels

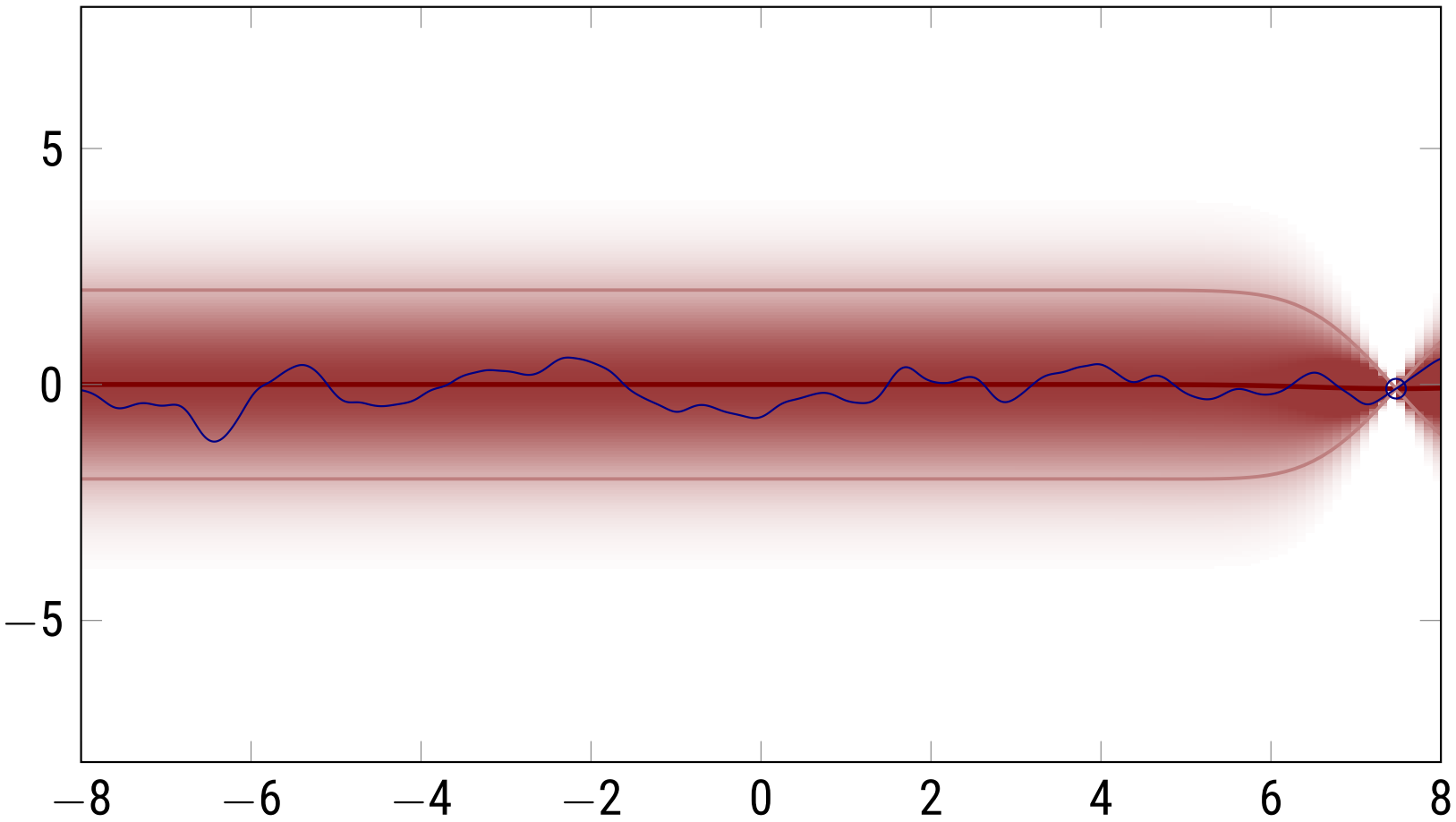
Universal RKHSs

an experiment – prior



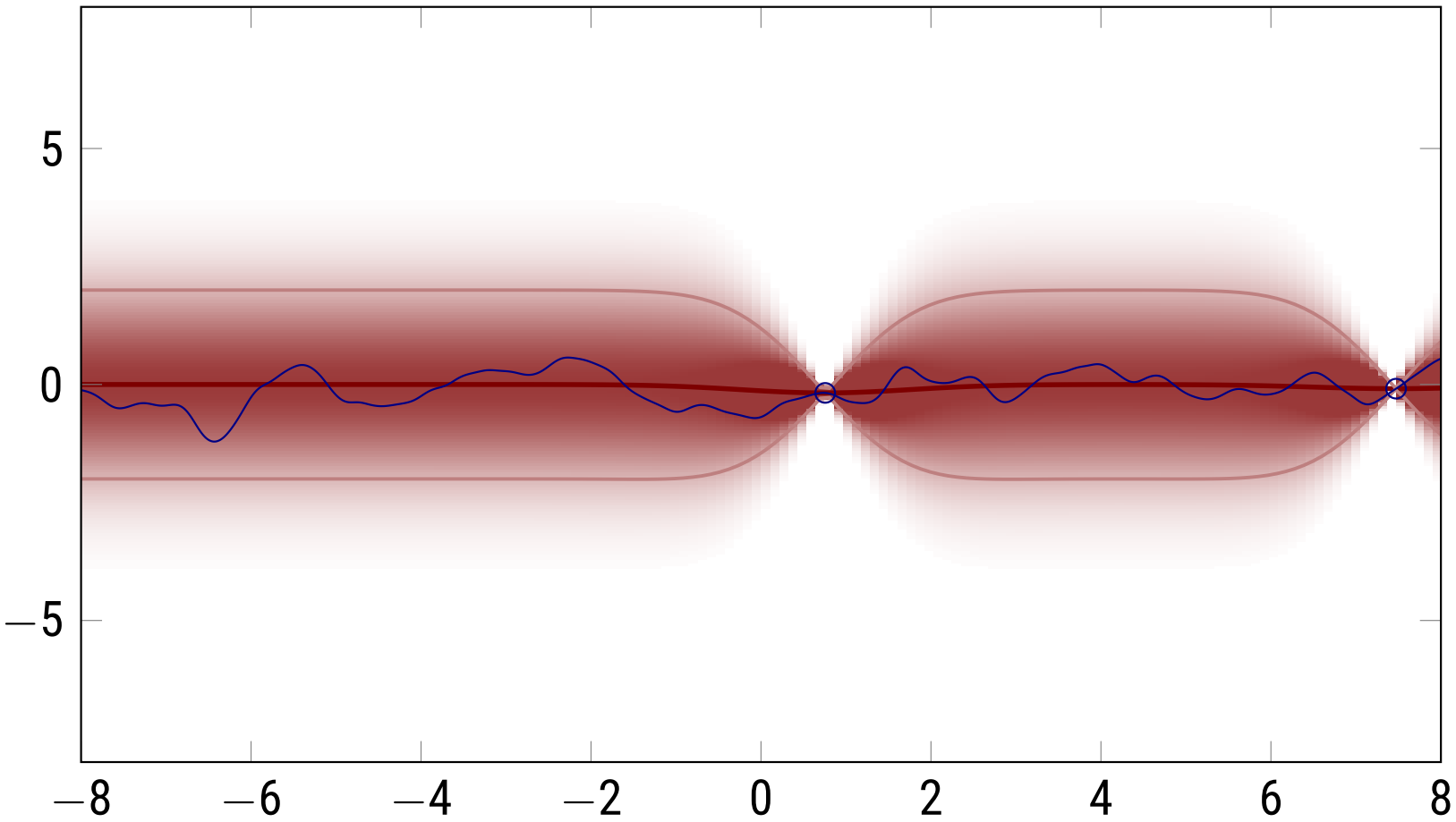
Universal RKHSs

an experiment – 1 evaluation



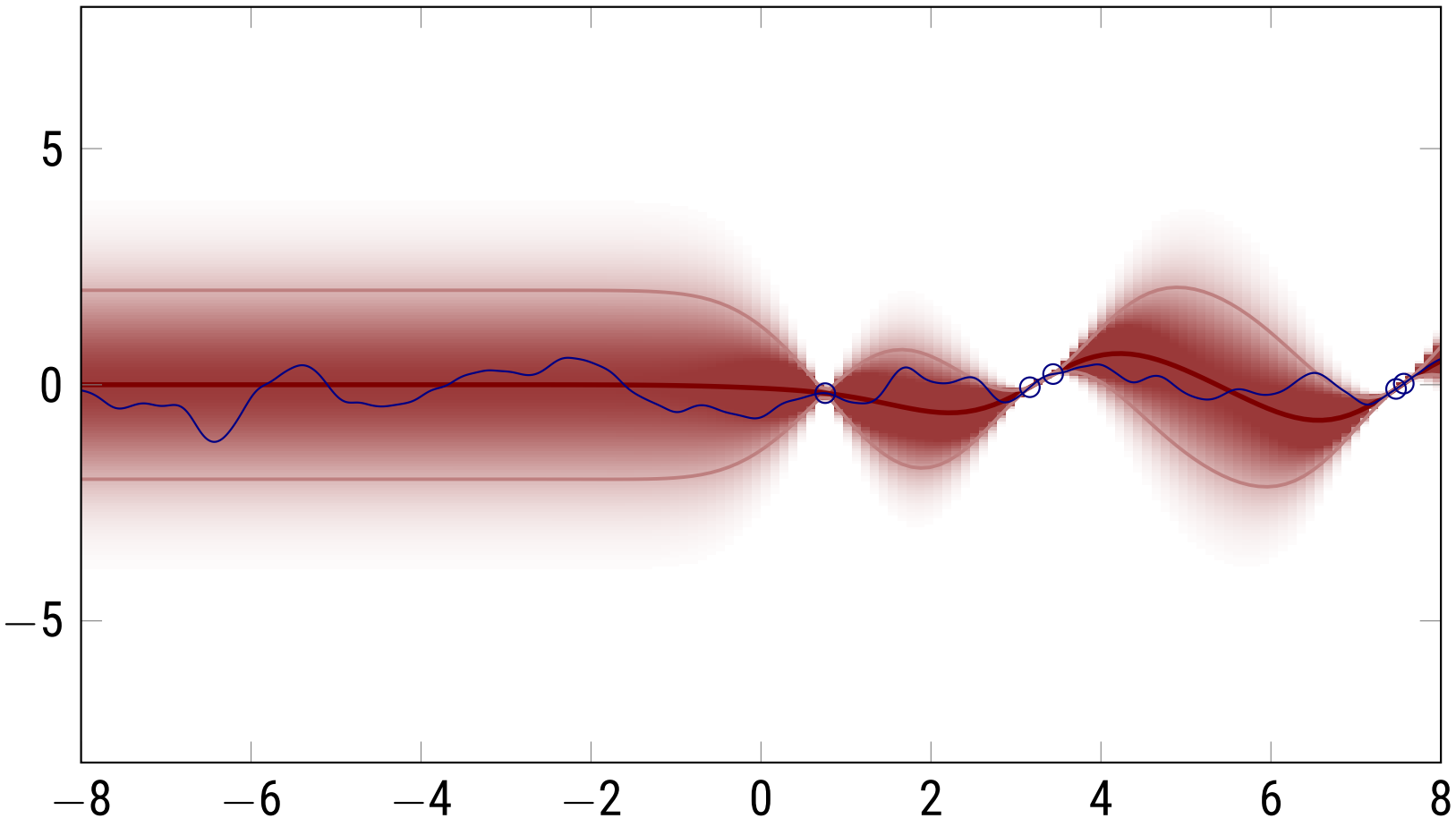
Universal RKHSs

an experiment – 2 evaluations



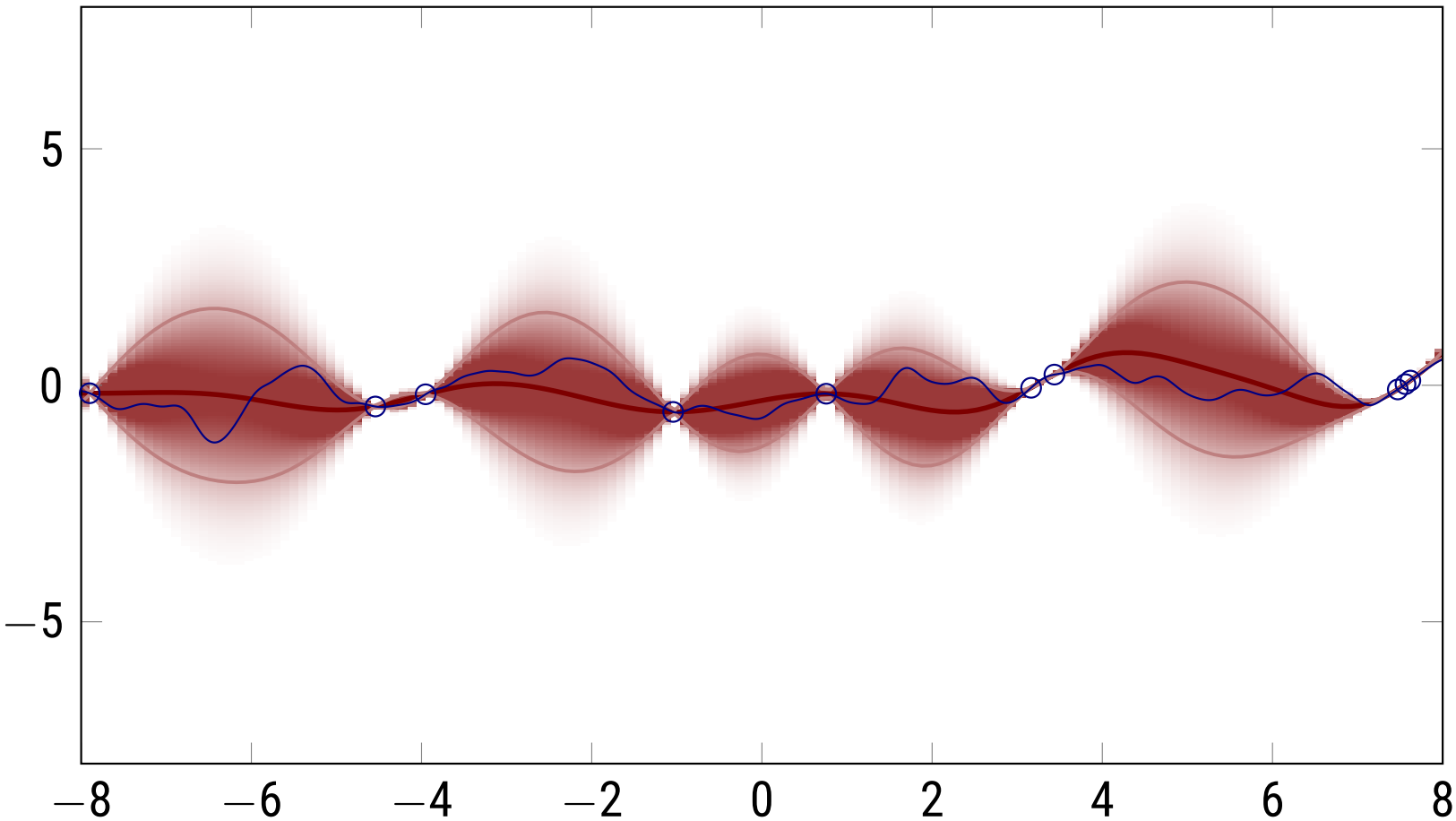
Universal RKHSs

an experiment – 5 evaluations



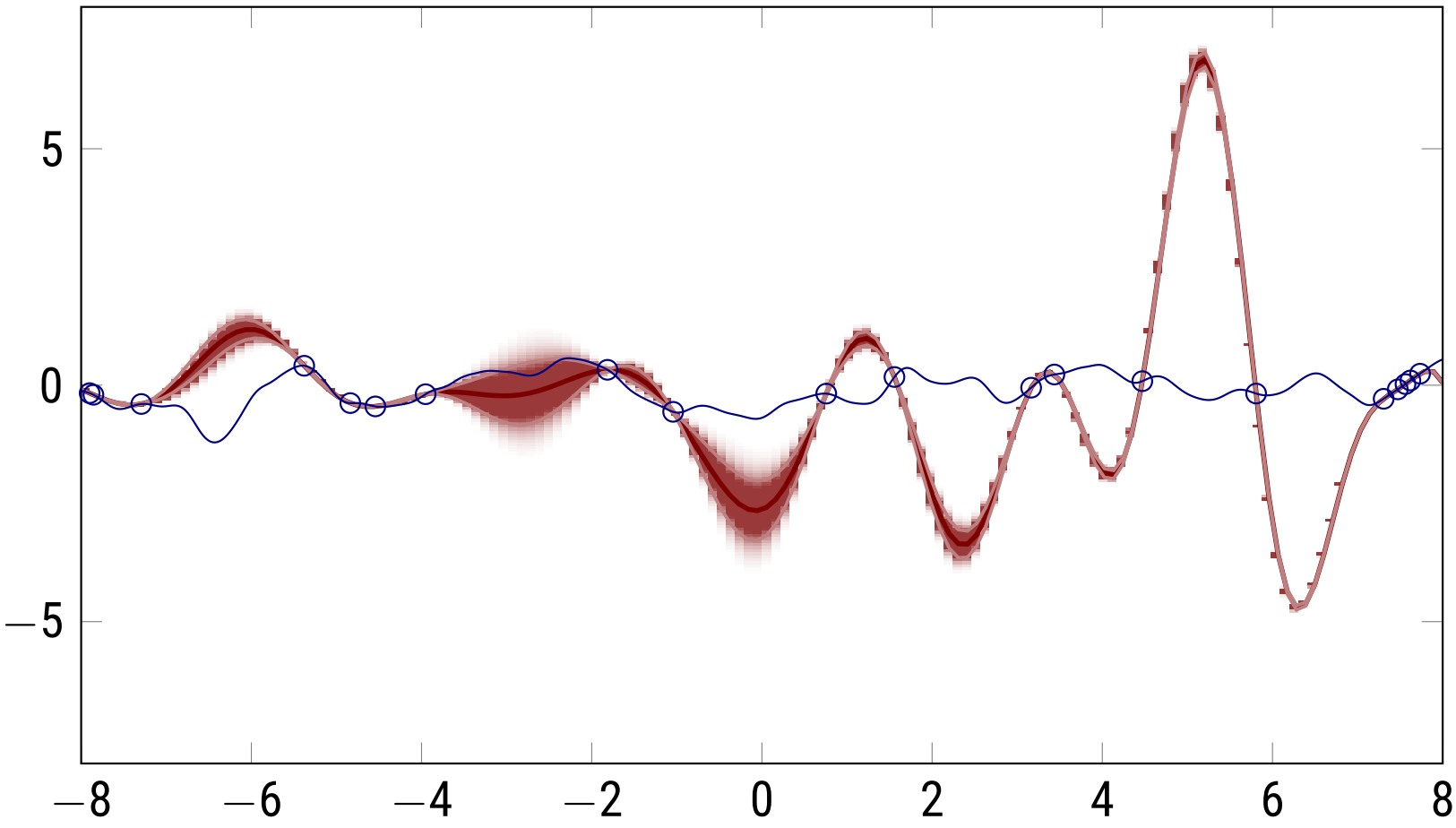
Universal RKHSs

an experiment – 10 evaluations



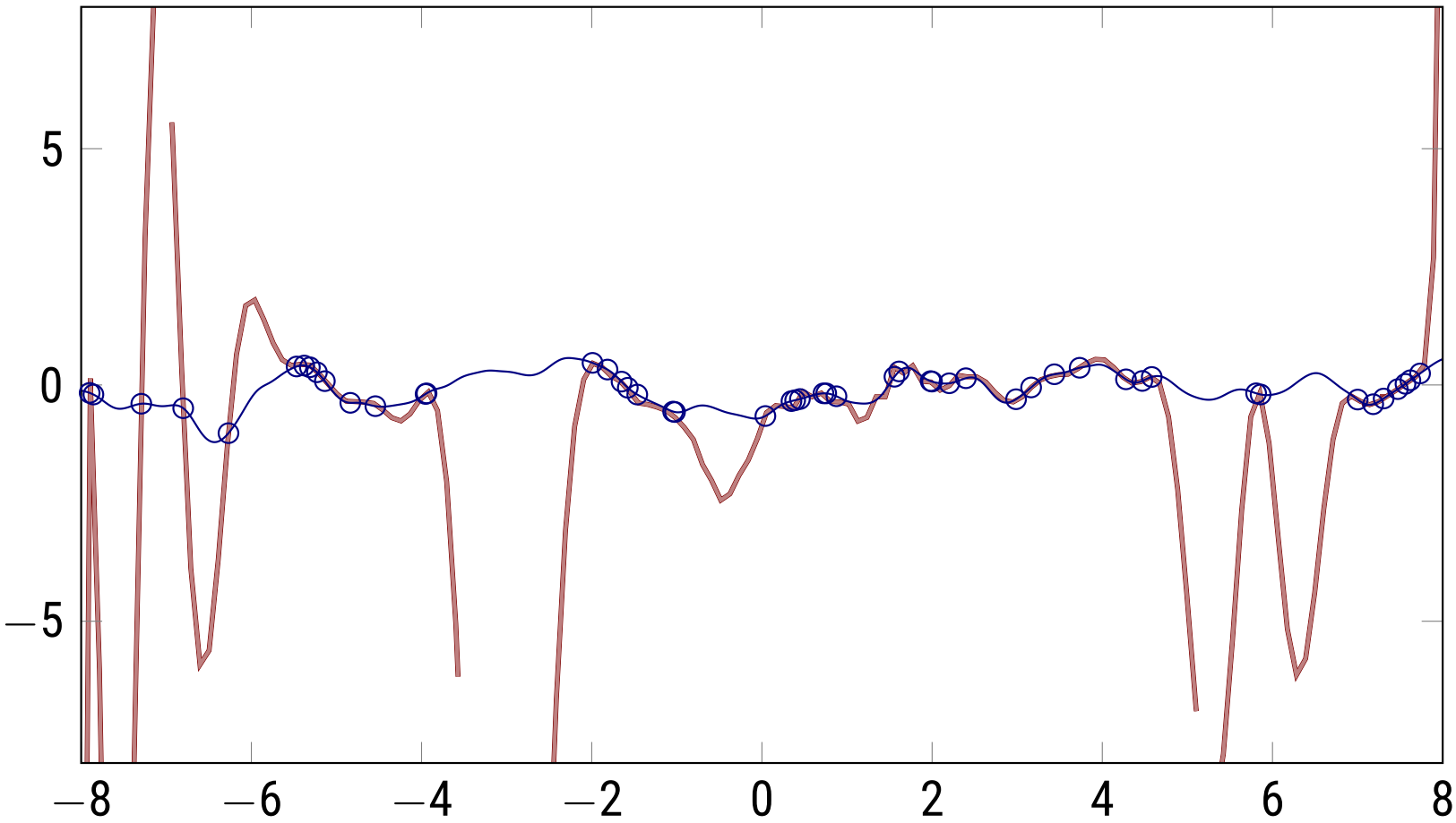
Universal RKHSs

an experiment – 20 evaluations



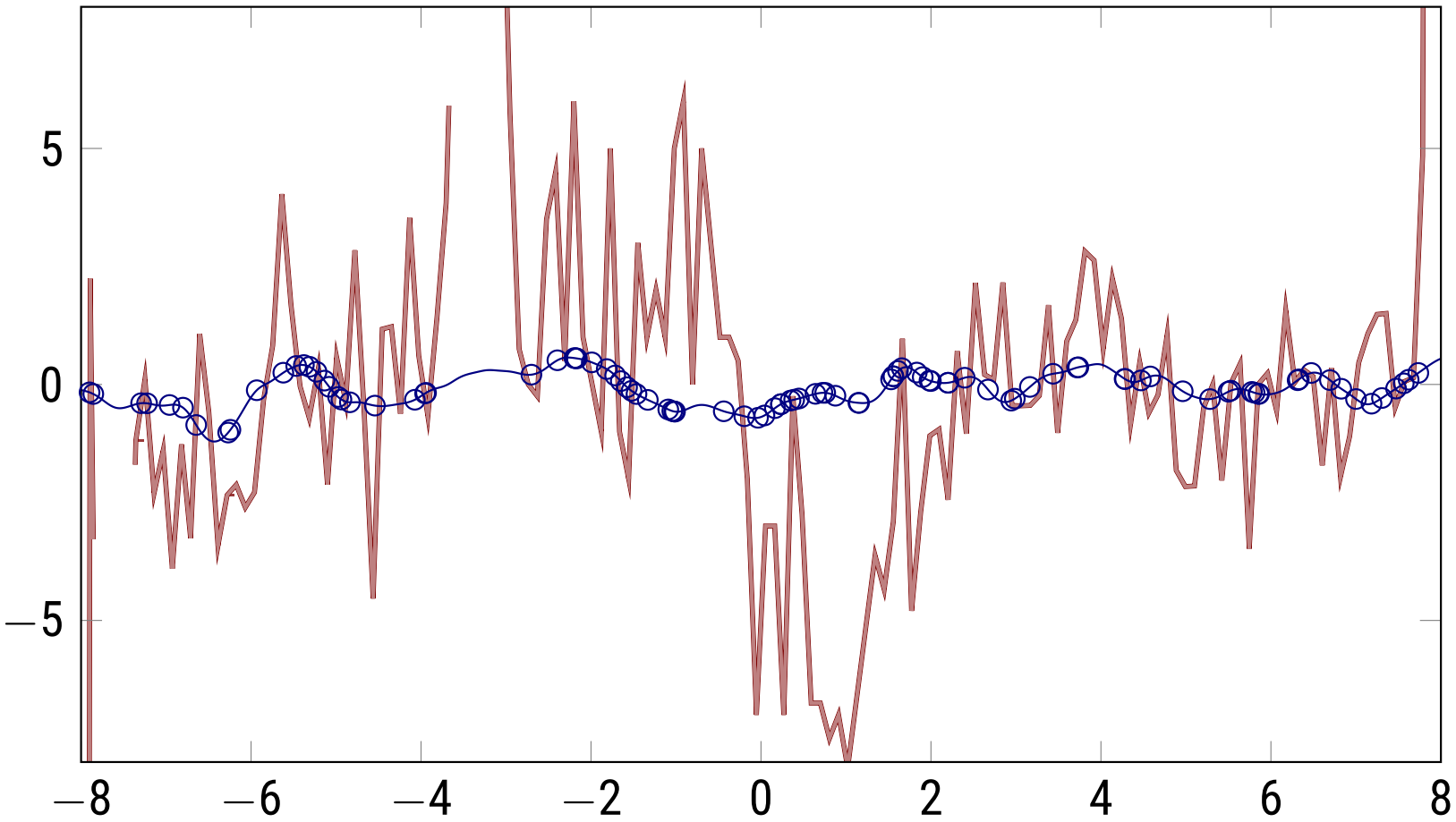
Universal RKHSs

an experiment – 50 evaluations



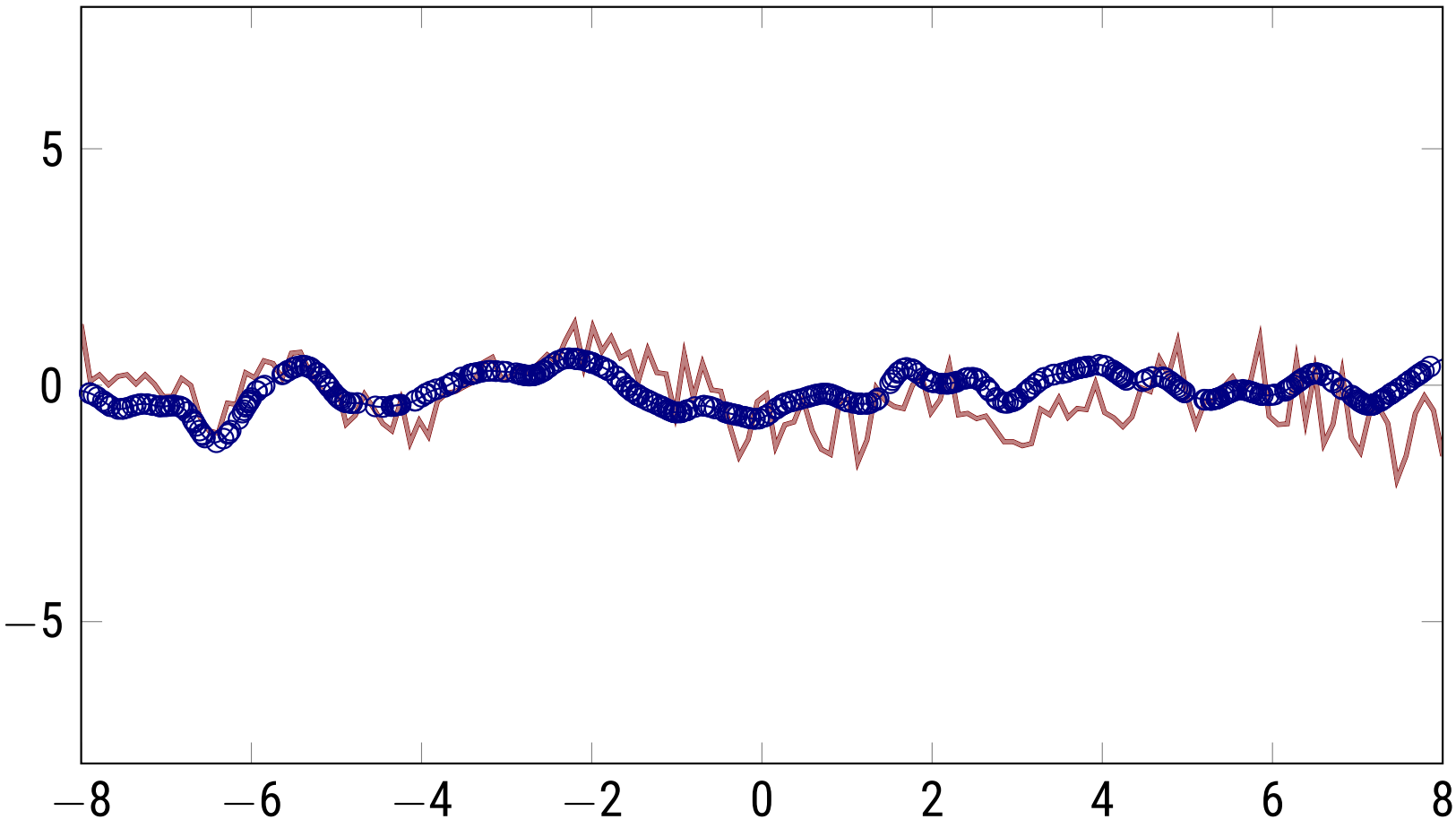
Universal RKHSs

an experiment – 100 evaluations



Universal RKHSs

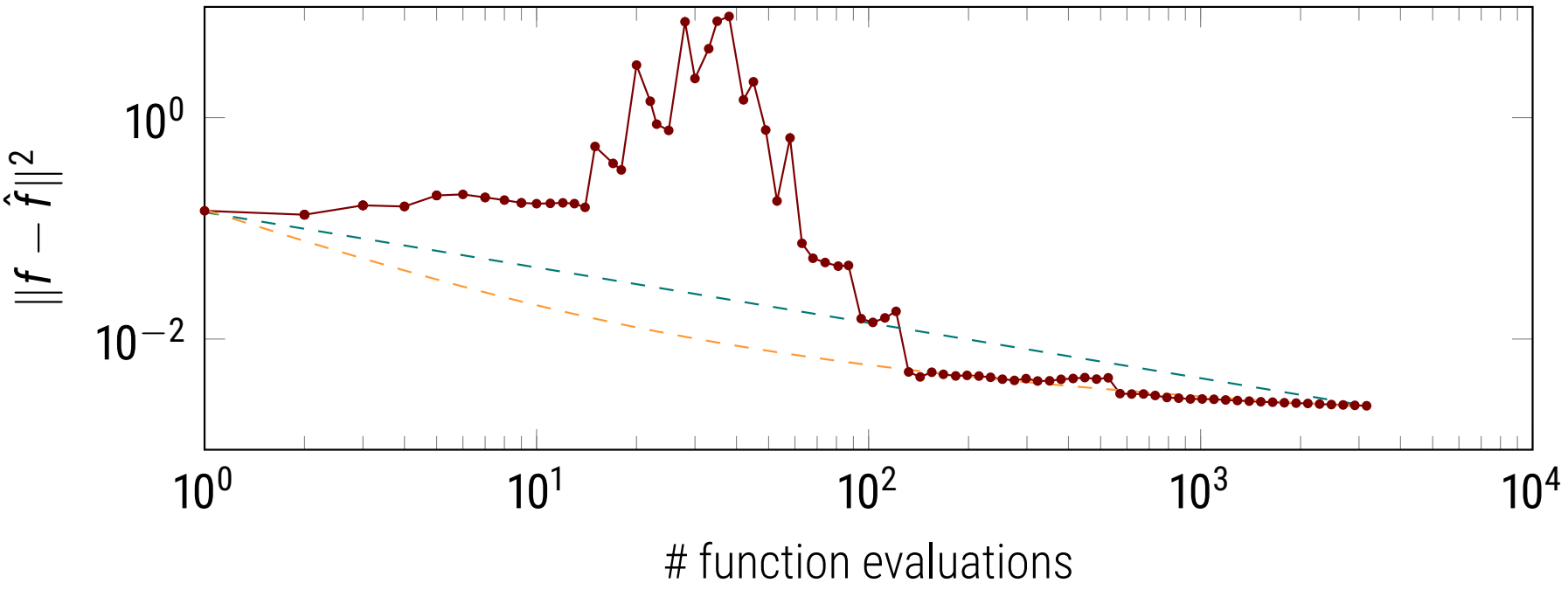
an experiment – 500 evaluations



Convergence Rates are Important

non-obvious aspects of f can ruin convergence

v.d.Vaart & v.Zanten, 2011



If f is "not well represented" by the kernel (has low prior density), the number of datapoints required to achieve ϵ error can be exponential in ϵ. Outside of the observation range, there are no guarantees at all.

An Analogy

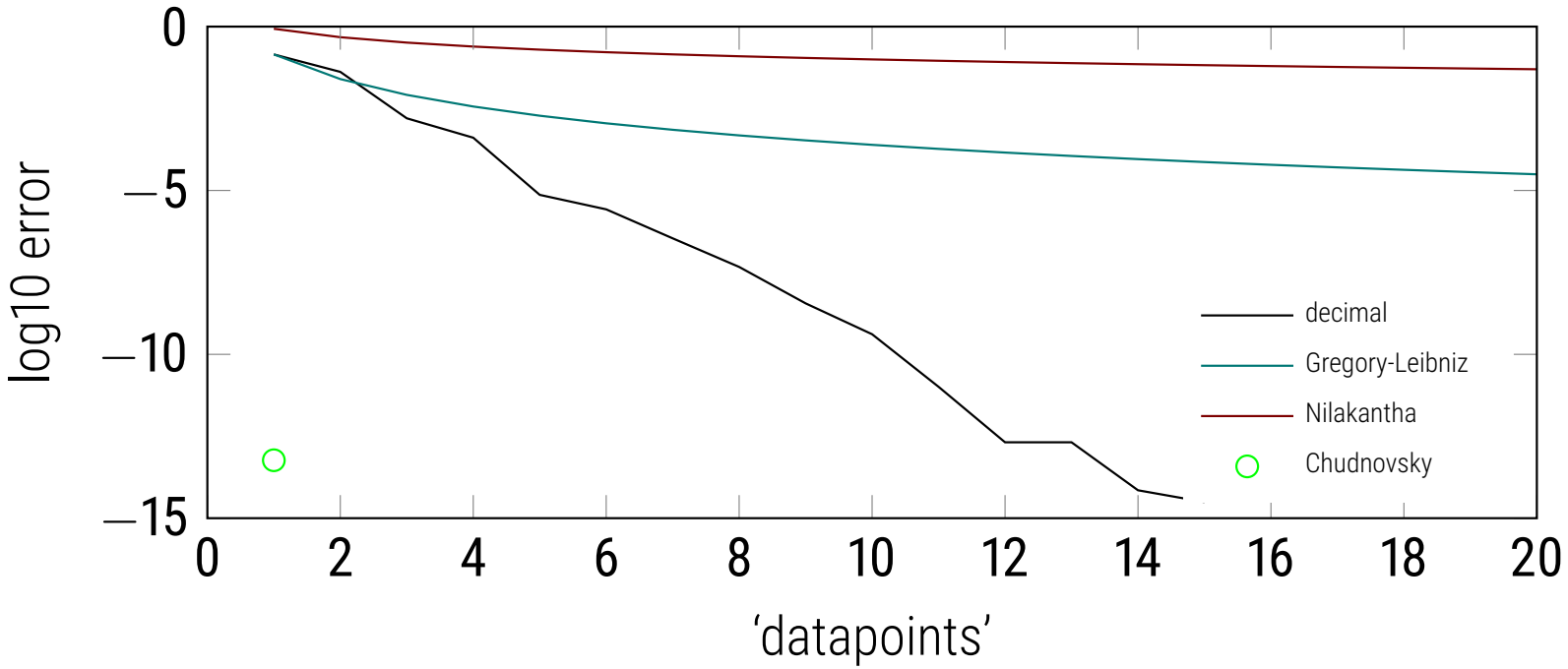
representing π in $\mathbb{Q}$

- $\mathbb{Q}$ is dense in $\mathbb{R}$

$$\pi = 3 \cdot \frac{1}{1} + 1 \cdot \frac{1}{10} + 4 \cdot \frac{1}{100} + 1 \cdot \frac{1}{1000} + \ldots \qquad \text{decimal}$$

$$= 4 \cdot \frac{1}{1} - 4 \cdot \frac{1}{3} + 4 \cdot \frac{1}{5} - 4 \cdot \frac{1}{7} + \ldots \qquad \text{Gregory-Leibniz}$$

$$= 3 \cdot \frac{1}{1} + 4 \cdot \frac{1}{2 \cdot 3 \cdot 4} - 4 \cdot \frac{1}{4 \cdot 5 \cdot 6} + 4 \cdot \frac{1}{6 \cdot 7 \cdot 8} \qquad \text{Nilakantha}$$



- Gaussian measures turn probabilistic inference into **linear algebra**
- Gaussian measures can be used to infer **functions** and **curves**
- In fact, the number of inferred features / weights can be **infinite**
- The choice of kernel / GP prior affects **model capacity** and **convergence**

next up:

- **Gauss-Markov processes**: fast probabilistic inference
- **Solving Ordinary Differential Equations** with GPs

These slides can be found at
`http://tinyurl.com/Dobbiaco-Hennig-1`

What does the GP **prior** mean?

Effect of kernel on samples

Theorem [Adler, 1981]

A stochastic process (a random field) over $\mathbb{X}$ is continuous in mean square at $x \in \mathbb{X}$ if, and only if, its covariance function $k(a, b)$ is continuous at $a = b = x$.

Continuous in mean square at x_*:

$$\mathrm{E}[|f(x_k) - f(x_*)|^2] \to 0$$

for any sequence x_k that converges to x_*.

$\Rightarrow$ "a GP is as often differentiable as the kernel at $x = a = b$."

but: continuity of individual samples is more tricky.

much more detail in Adler, *The geometry of random fields*. Wiley, 1981.