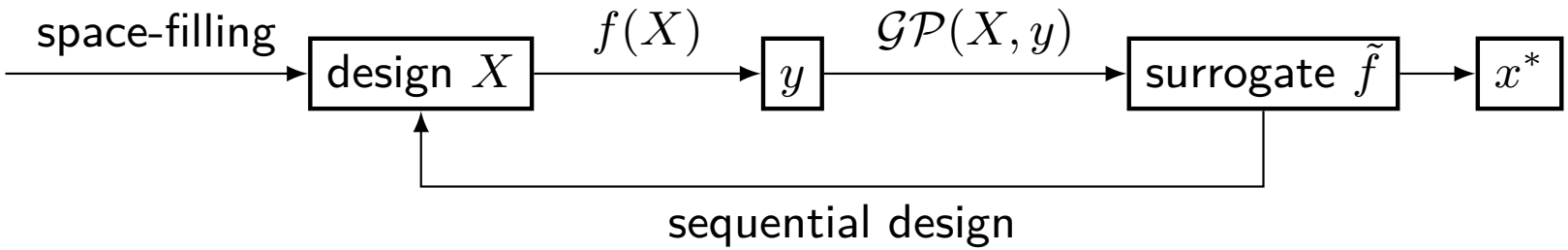
Surrogate Optimization: Gaussian Process Regression

BIOE 498/598 PJ

Spring 2021

Surrogate Optimization

- Assume we are trying to optimize a function f that is **expensive** to evaluate.
- Instead, we use evaluations of f to build a surrogate model $\tilde{f}$ that is **cheap** to evaluate.
- We optimize $\tilde{f}$ to find good candidates for evaluation by f.



Gaussian Process Regression

- For linear models, we decide *a priori* what shape the response surface will take.
- Linear regression estimates the parameters β_i using noisy data.
- **Gaussian Process Regression (GPR)** assumes the *covariance* between the data have a particular shape.
- The covariance function is called the *kernel*.

Our kernel of choice

- There are many kernels used for GPR.
- We will use the *inverse exponentiated squared Euclidean distance* kernel:

$$\Sigma(x, x') = \exp\{-\|x - x'\|^2\}.$$

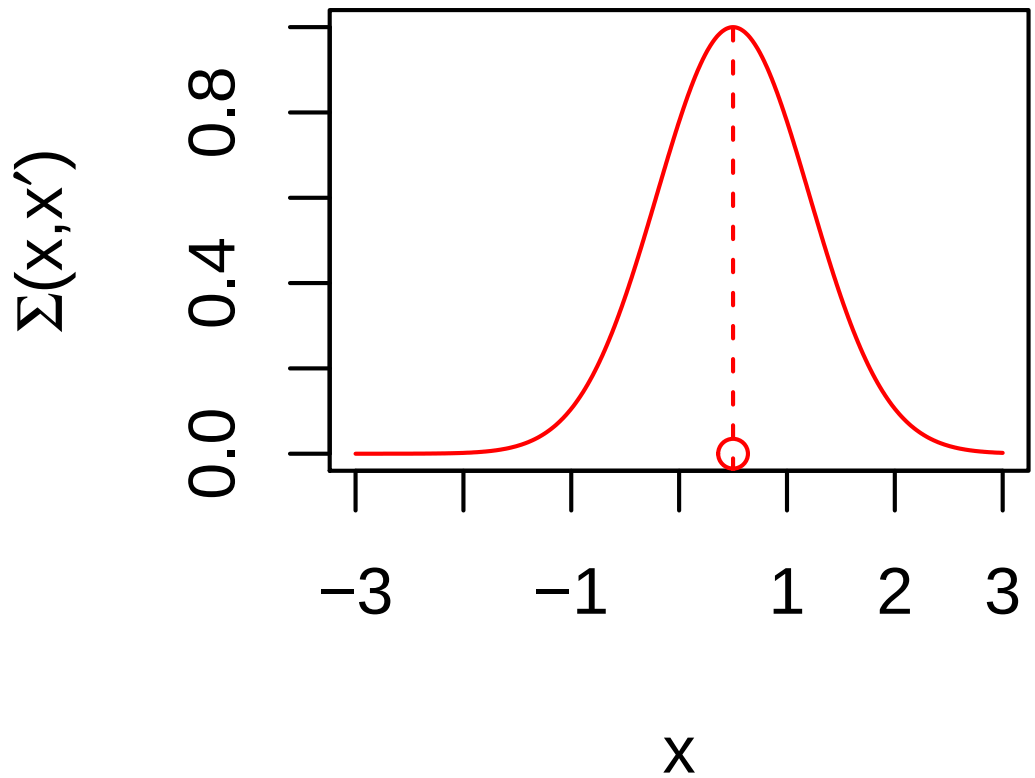
- Note that $\Sigma(x, x) = 1$ and $\Sigma(x, x') < 1$ if $x \neq x'$.

Our kernel of choice

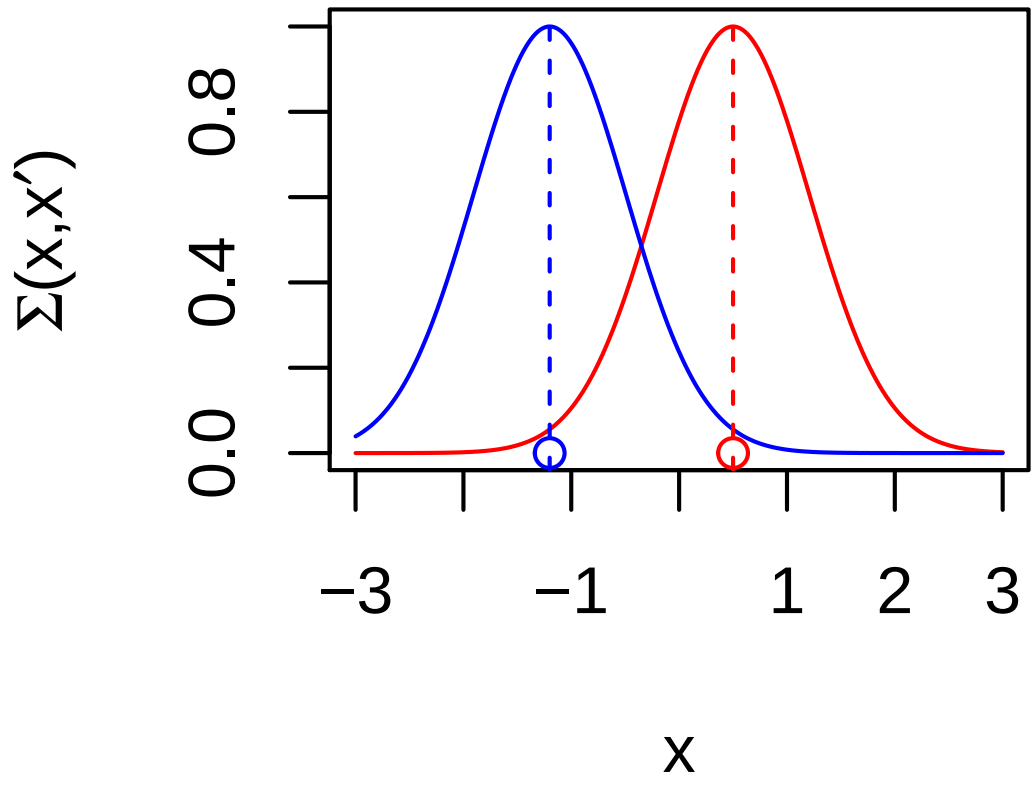
- There are many kernels used for GPR.
- We will use the *inverse exponentiated squared Euclidean distance* kernel:

$$\Sigma(x, x') = \exp\{-\|x - x'\|^2\}.$$

- Note that $\Sigma(x, x) = 1$ and $\Sigma(x, x') < 1$ if $x \neq x'$.



Using the covariance function for interpolation



How do we make predictions with GPR?

- Let's start with a space-filling design X_n and assume we measured the responses y_n at each point in the design.
- Using our kernel function, we can calculate the covariance among the points in the design set

$$\Sigma_n = \Sigma(X_n, X_n)$$

How do we make predictions with GPR?

- Let's start with a space-filling design X_n and assume we measured the responses y_n at each point in the design.
- Using our kernel function, we can calculate the covariance among the points in the design set

$$\Sigma_n = \Sigma(X_n, X_n)$$

- We want to find the response y at a new, unmeasured point x. Using some identities for multivariate normal distributions,

$$y(x) = \Sigma(x, X_n)\Sigma_n^{-1} y_n.$$

How do we make predictions with GPR?

- Let's start with a space-filling design X_n and assume we measured the responses y_n at each point in the design.
- Using our kernel function, we can calculate the covariance among the points in the design set

$$\Sigma_n = \Sigma(X_n, X_n)$$

- We want to find the response y at a new, unmeasured point x. Using some identities for multivariate normal distributions,

$$y(x) = \Sigma(x, X_n)\Sigma_n^{-1} y_n.$$

- GPR assumes that $y(x)$ is itself normally distributed with variance

$$\sigma^2(x) = \Sigma(x, x) - \Sigma(x, X_n)\Sigma_n^{-1}\Sigma(x, X_n)^\top.$$

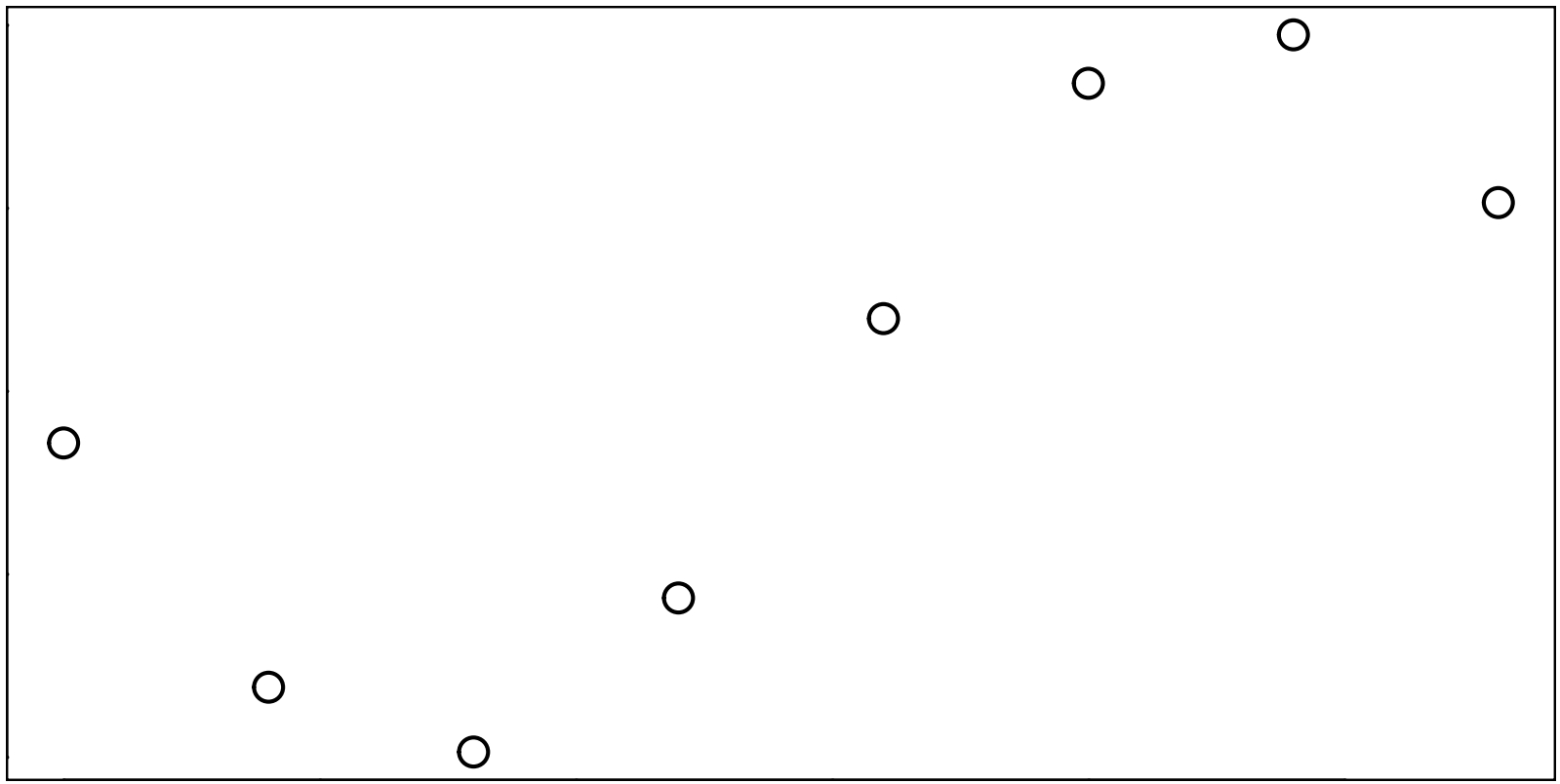
Let's try it!

First, let's make a helper function for computing the covariance between two sets of design points.

```
Sigma <- function(X1,X2) {
  X1 <- as.matrix(X1)
  X2 <- as.matrix(X2)
  D <- plgp::distance(X1,X2)
  exp(-D)
}
```

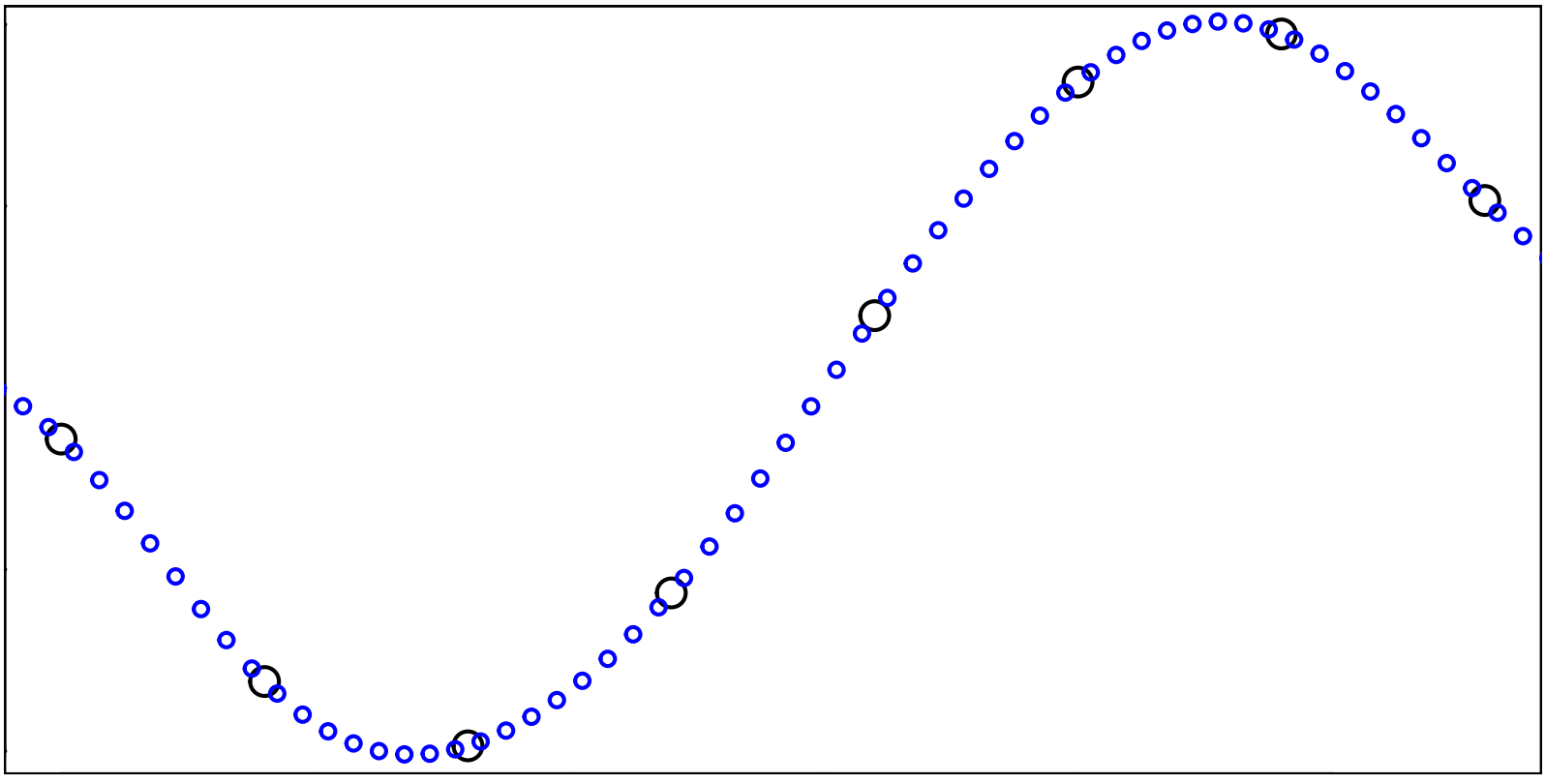
Let's make some training data

```
par(mar=rep(0,4))
Xn <- matrix(seq(-3,3,0.8), ncol=1)
yn <- sin(Xn[ ,1])
plot(Xn,yn)
```



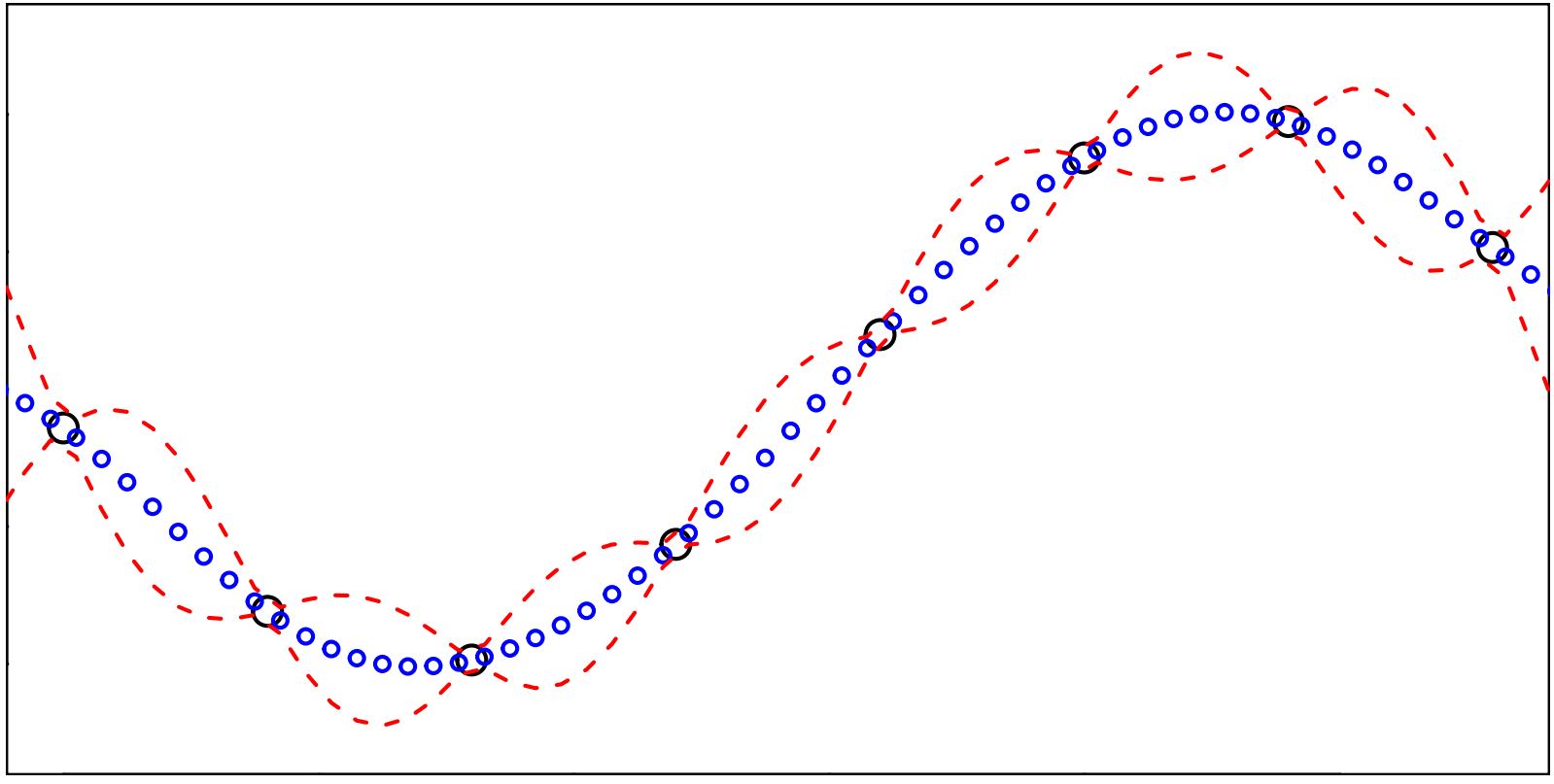
And then interpolate!

```
X <- seq(-3.25,3.15,0.1)
y = Sigma(X,Xn) %*% solve(Sigma(Xn,Xn)) %*% yn
par(mar=rep(0,4))
plot(Xn,yn)
points(X,y, col="blue", cex=0.5)
```



What about the variance?

```
s2 <- Sigma(X,X) - Sigma(X,Xn) %*%
        solve(Sigma(Xn,Xn)) %*% t(Sigma(X,Xn))
par(mar=rep(0,4))
plot(Xn,yn, ylim=c(-1.3,1.3))
points(X,y, col="blue", cex=0.5)
lines(X, y + qnorm(0.05, 0, sqrt(diag(s2))), lty=2, col=2)
lines(X, y + qnorm(0.95, 0, sqrt(diag(s2))), lty=2, col=2)
```



Why not use GPR for everything?

- **Data intensive.** Since GPR does not use a parametric model, the entire shape of the response surface must come from data. GPR generally requires more data than a linear model.
- **Computationally intensive.** Training a GPR requires inverting Σ_n, an $n \times n$ dense matrix. Practically, this limits GPR to 1,000's or a few 10,000's of points.
- **Interpolation only.** GPR has no idea what the response should look like beyond the training data. GPR requires a space-filling design that covers the entire search region.

Why not use GPR for everything?

- **Data intensive.** Since GPR does not use a parametric model, the entire shape of the response surface must come from data. GPR generally requires more data than a linear model.
- **Computationally intensive.** Training a GPR requires inverting Σ_n, an $n \times n$ dense matrix. Practically, this limits GPR to 1,000's or a few 10,000's of points.
- **Interpolation only.** GPR has no idea what the response should look like beyond the training data. GPR requires a space-filling design that covers the entire search region.

Still, for global search with (relatively) expensive experiments, GPR remains a flexible and powerful method.