Replicate or Explore?

FTC, Nov 2020

Robert B. Gramacy (rbg@vt.edu : http://bobby.gramacy.com)
Department of Statistics, Virginia Tech

Joint with Mickaël Binois (INRIA), Jiangeng Huang (UCSC),
Mike Ludkovski (UCSB) and Chris Franck (VT)

Surrogate modeling

A computer experiment involves a function $f(x) : x \in \mathbb{R}^p \to \mathbb{R}$ requiring expensive simulation to *approximate* a real-world (physical) relationship.

An **emulator** $\hat{f}_N$ is a *regression* or *response surface* fit to input-output pairs $(x_1, y_1), \ldots, (x_N, y_N)$, where $y_i \sim f(x_i)$.

- $\hat{f}_N(x')$ may serve as a cheap **surrogate** for $f(x')$ at new inputs x' for visualization, sensitivity analysis, optimization, etc.

Gaussian processes (GPs), i.e., MVN spatial modeling

$$Y_N \sim \mathcal{N}_N(0, \nu(C_N + \tau^2 \mathbb{I}_N)) \quad \text{where} \quad C_\theta(\cdot, \cdot) = \exp\left\{ - \sum_{k=1}^{m} \frac{(x_k - x'_k)^2}{\theta_k} \right\}$$

make popular surrogates because their predictions are

- accurate, have appropriate coverage, and can *interpolate*.

GP review

Gaussian process prior

To *generate* random functions following a GP

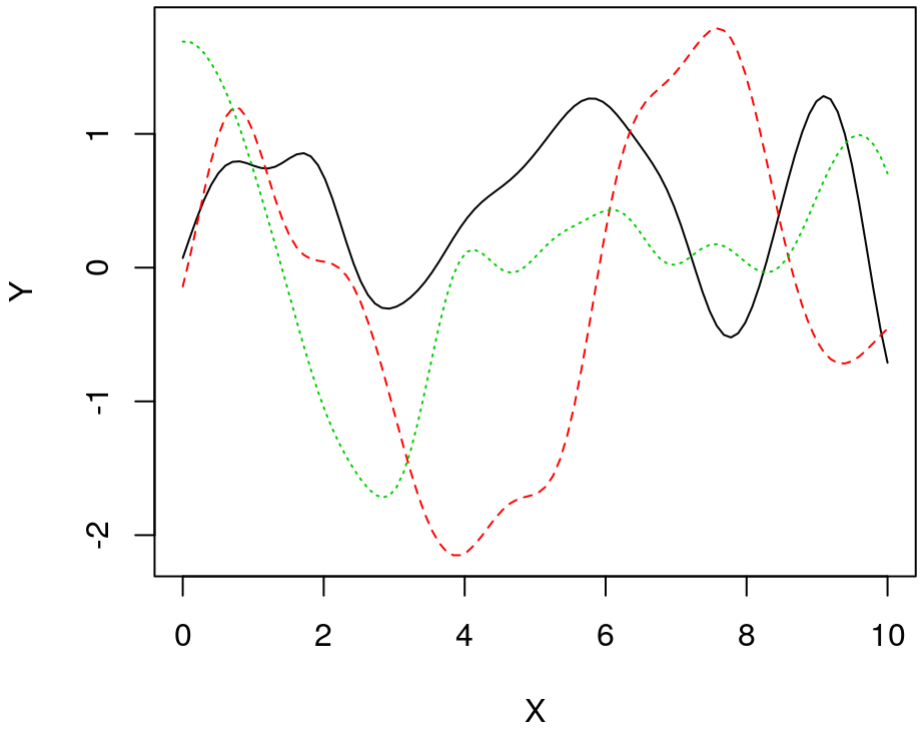
- take a bunch of x-values: $x_1, \ldots, x_N$
- build $\Sigma_N \equiv \nu(C_N + \tau^2 \mathbb{I}_n)$ with $(\nu, \theta, \tau^2) = (1, 1, 0)$ via $\Sigma_N^{ij} = e^{-||x_j - x_j||^2}$
- draw an N-variate realization $Y \sim \mathcal{N}_N(0, \Sigma_N)$

```
N <- 100
X <- matrix(seq(0, 10, length=N), ncol=1)
library(plgp)                           ## for easy Euclidean distance
D <- distance(X)
eps <- sqrt(.Machine$double.eps)        ## defining a small number
Sigma <- exp(-D) + diag(eps, N)         ## for numerical stability
library(mvtnorm)
Y <- rmvnorm(3, sigma=Sigma)
```

- and plot the result in the x-y plane.

Nice looking random functions

```
matplot(X, t(Y), type="l", ylab="Y")
```



Posterior

Of course, we're not in the business of generating random functions.

- I'm not sure what that would be useful for.

Instead, we want to ask …

- Given example pairs $(x_1, y_1), \ldots, (x_N, y_N)$ comprising data D_N,
- what random function realizations could explain those values?

I.e., we want to know about the conditional distribution of $Y \mid D_N$.

- If we call $Y \sim \mathcal{N}_N$ the prior, then $Y \mid D_N$ must be the posterior.

MVN partition & conditioning

From Wikipedia, if an N-dimensional random MVN vector x is partitioned as

$$x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \quad \text{with sizes} \quad \begin{pmatrix} q \times 1 \\ (N-q) \times 1 \end{pmatrix},$$

and accordingly μ and Σ are partitioned as,

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \quad \text{with sizes} \begin{pmatrix} q \times 1 \\ (N-q) \times 1 \end{pmatrix}, \quad \text{and}$$
$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix} \quad \text{with sizes} \begin{pmatrix} q \times q & q \times (N-q) \\ (N-q) \times q & (N-q) \times (N-q) \end{pmatrix},$$

… then, $x_1 \mid x_2 \sim \mathcal{N}_q(\bar{\mu}, \bar{\Sigma})$, where

$$\bar{\mu} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\text{and} \quad \bar{\Sigma} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}.$$

GP prediction or "kriging"

GP prediction, $Y(\mathcal{X}) \mid D_N$ at new locations $\mathcal{X}$, is just a special case (mean zero) of those conditional MVN equations:

$$Y(\mathcal{X}) \mid D_N \sim \mathcal{N}_{n'}(\mu(\mathcal{X}), \Sigma(\mathcal{X}))$$

with

$$\begin{aligned} \text{mean} \quad & \mu(\mathcal{X}) = \Sigma(\mathcal{X}, X_N)\Sigma_N^{-1} Y_N \\ \text{and variance} \quad & \Sigma(\mathcal{X}) = \Sigma(\mathcal{X}, \mathcal{X}) - \Sigma(\mathcal{X}, X_N)\Sigma_N^{-1}\Sigma(\mathcal{X}, X_N)^\top. \end{aligned}$$

- where $\Sigma(\mathcal{X}, X_N)$ is an $n' \times n$ matrix.

These are the so-called "kriging" equations in spatial statistics.

- They describe the best (minimizing MSPE) linear unbiased predictor (BLUP).

For example …

Consider some super simple data in 1d.

Here are the relevant data quantities.

```
n <- 8
X <- matrix(seq(0,2*pi,length=n), ncol=1)
y <- sin(X)
D <- distance(X)
Sigma <- exp(-D) + diag(eps, n)
```

Here are the relevant predictive quantities.

```
XX <- matrix(seq(-0.5, 2*pi+0.5, length=100), ncol=1)
DXX <- distance(XX)
SXX <- exp(-DXX) + diag(eps, ncol(DXX))
DX <- distance(XX, X)
SX <- exp(-DX)
```

Predictive equations in R

Following the MVN conditioning formulas.

```
Si <- solve(Sigma)
mup <- SX %*% Si %*% y
Sigmap <- SXX - SX %*% Si %*% t(SX)
```

Joint draws are obtained from the predictive distribution as follows.

```
YY <- rmvnorm(100, mup, Sigmap)
```

We can plot those $Y(\mathcal{X}) =$ YY samples as a function of the input $\mathcal{X} =$ XX locations,

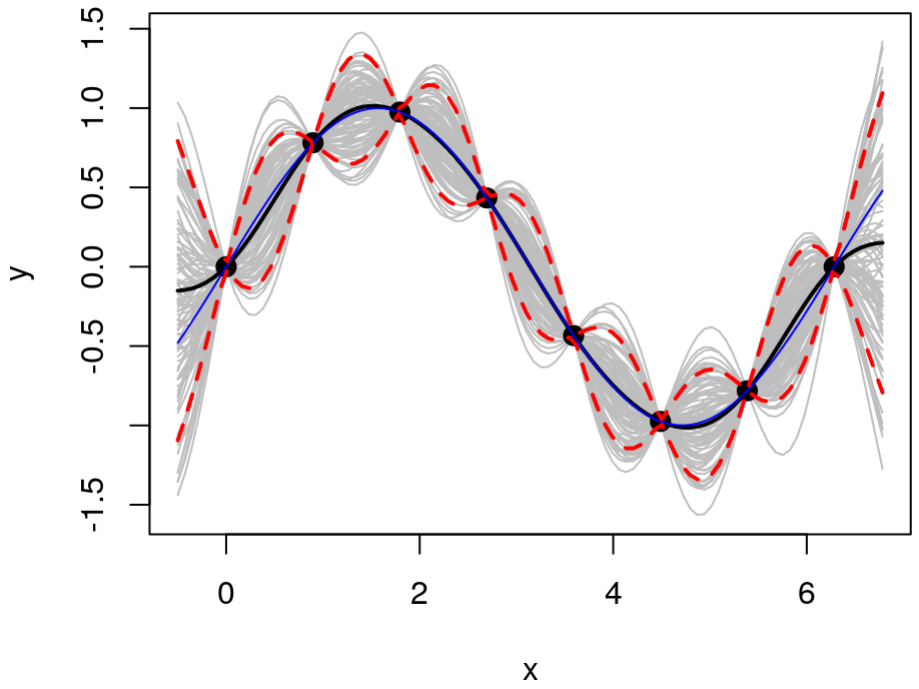
- maybe along with some pointwise quantile-based error bars.

```
q1 <- mup + qnorm(0.05, 0, sqrt(diag(Sigmap)))
q2 <- mup + qnorm(0.95, 0, sqrt(diag(Sigmap)))
```

- You ready?

Nice looking interpolations

```
matplot(XX, t(YY), type="l", col="gray", lty=1, xlab="x", ylab="y")
points(X, y, pch=20, cex=2)
lines(XX, mup, lwd=2); lines(XX, sin(XX), col="blue")
lines(XX, q1, lwd=2, lty=2, col=2); lines(XX, q2, lwd=2, lty=2, col=2)
```



Hyperparameters

If the covariance structure is (hyper-) parameterized,

$$Y_N \sim \mathcal{N}_N(0, \nu(C_N + \tau^2 \mathbb{I}_N)) \quad \text{where} \quad C_\theta(\cdot,\cdot) = \exp\left\{ -\sum_{k=1}^{m} \frac{(x_k - x'_k)^2}{\theta_k} \right\},$$

then use the MVN to define a (log) "likelihood" for the unknowns (ν, θ, τ^2).

$$\ell = \log L = -\frac{N}{2}\log 2\pi - \frac{N}{2}\log \nu - \frac{1}{2}\log |K_N| - \frac{1}{2\nu} Y_N^\top K_N^{-1} Y_N,$$

where $K_N = C_N + \tau^2 \mathbb{I}_N$. To maximize ℓ with respect to ν just differentiate and solve.

$$0 \stackrel{\text{set}}{=} \ell'(\nu) \equiv \frac{\partial \ell}{\partial \nu} = -\frac{N}{2\nu} + \frac{1}{2\nu^2} Y_N^\top K_N^{-1} Y_N$$

$$\hat{\nu}_N = \frac{Y_N^\top K_N^{-1} Y_N}{N}$$

Concentrated log likelihood

Now, plugging $\hat{\nu}$ into ℓ gives the so-called **concentrated log likelihood**,

$$\begin{aligned}\ell(\tau^2, \theta) &= -\frac{N}{2}\log 2\pi - \frac{N}{2}\log \hat{\nu} - \frac{1}{2\hat{\nu}} Y_N^\top K_N^{-1} Y_N \\ &= c - \frac{N}{2}\log Y_N K_N^{-1} Y_N - \frac{1}{2}\log |K_N|.\end{aligned}$$

Using the chain rule and that

$$\frac{\partial K_N^{-1}}{\partial \phi} = -K_N^{-1}\frac{\partial K_N}{\partial \phi}K_N^{-1} \quad \text{and} \quad \frac{\partial \log |K_N|}{\partial \phi} = \text{tr}\left\{K_N^{-1}\frac{\partial K_N}{\partial \phi}\right\}$$

yields closed-form expressions for the partial derivatives wrt $(\theta_1, \ldots, \theta_k)$ and τ^2.

- Unfortunately these cannot be set to zero and solved analytically.
- But numerical methods are pretty good …

```
nl <- function(par, X, Y)
  {
    theta <- par[1:ncol(X)]
    tau2 <- par[ncol(X)+1]
    K <- covar.sep(X, d=theta, g=tau2); Ki <- solve(K)
    ldetK <- determinant(K, logarithm=TRUE)$modulus
    ll <- - (length(Y)/2) * log(t(Y) %*% Ki %*% Y) - (1/2) * ldetK
    return(-ll)
  }

gnl <- function(par, X, Y)
 {
    n <- length(Y)
    theta <- par[1:ncol(X)]; tau2 <- par[ncol(X)+1]
    K <- covar.sep(X, d=theta, g=tau2); Ki <- solve(K)
    KiY <- Ki %*% Y
    dlltheta <- rep(NA, length(theta))
    for(k in 1:length(dlltheta)) {
      dotK <- K * distance(X[,k]) / (theta[k]^2)
      dlltheta[k] <- (n/2) * t(KiY) %*% dotK %*% KiY / (t(Y) %*% KiY) -
        (1/2) * sum(diag(Ki %*% dotK))
    }
    dlltau2 <- (n/2) * t(KiY) %*% KiY / (t(Y) %*% KiY) - (1/2) * sum(diag(Ki))
    return(-c(dlltheta, dlltau2))
 }
```

A 2d example

Consider a 2d input space and responses observed with noise.

```
library(lhs)
X <- randomLHS(40, 2)
X <- rbind(X,X)
X[,1] <- (X[,1] - 0.5)*6 + 1
X[,2] <- (X[,2] - 0.5)*6 + 1
y <- X[,1] * exp(-X[,1]^2 - X[,2]^2) + rnorm(nrow(X), sd=0.01)
```

Estimating hyperparameters.

```
out <- optim(c(rep(0.1, 2), 0.1*var(y)), nl, gnl, method="L-BFGS-B",
  lower=eps, upper=c(rep(10, 10), var(y)), X=X, Y=y)
out$par
```

```
## [1] 0.763963729 1.922457435 0.005637002
```

- The x_2 lengthscale (θ_2) is about $2\times$ longer than for x_1 (θ_1).

Predictive surface

Re-building the data quantities …

```
K <- covar.sep(X, d=out$par[1:2], g=out$par[3])
Ki <- solve(K)
nuhat <- drop(t(y) %*% Ki %*% y / nrow(X))
```

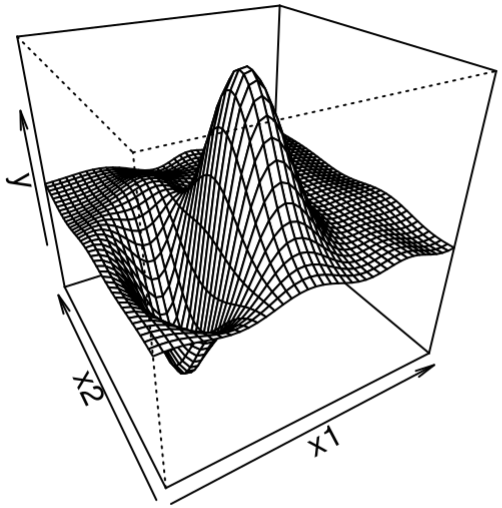
… and then the predictive quantities.

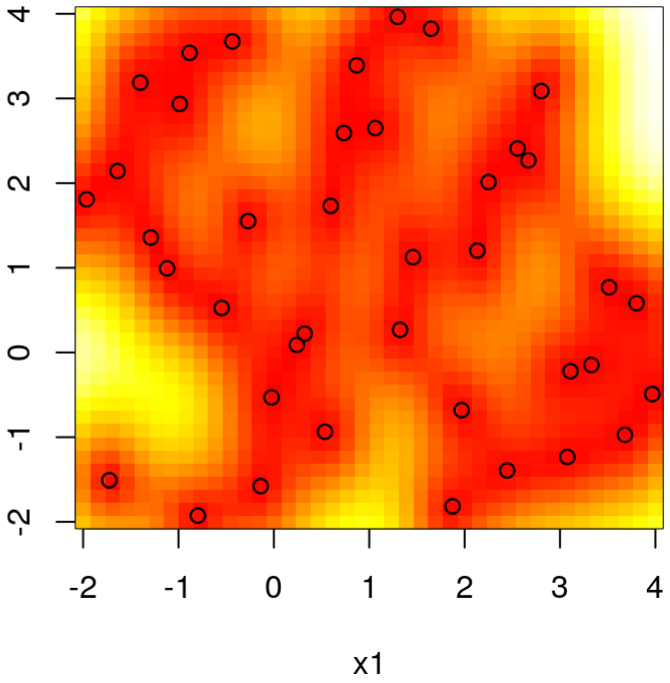
```
xx <- seq(-2,4, length=40)
XX <- as.matrix(expand.grid(xx, xx))
KXX <- covar.sep(XX, d=out$par[1:2], g=out$par[3])
KX <- covar.sep(XX, X, d=out$par[1:2], g=0)
mup <- KX %*% Ki %*% y
Sigmap <- nuhat * (KXX - KX %*% Ki %*% t(KX))
sdp <- sqrt(diag(Sigmap))
```

- (Cut-and-paste from above with estimated hyperparameters "plugged in".)

Fitted (predictive) mean and standard deviation surfaces:

```
par(mfrow=c(1,2)); cols <- heat.colors(128)
persp(xx,xx, matrix(mup, ncol=40), theta=-30,phi=30,xlab="x1",ylab="x2",zlab="y")
image(xx,xx, matrix(sdp, ncol=length(xx)), xlab="x1",ylab="x2", col=cols)
points(X[,1], X[,2])
```





That's great; so where from here?

Increasingly, data in geostatistics, machine learning, and computer simulation experiments involve signal-to-noise ratios which

- may be **low**
- and/or possibly **changing** over the input space.

Stochastic (computer) simulators from physics, business and epidemiology, even Bayesian MCMC, may exhibit both of those features simultaneously,

- but lets start with the first (walk before we run).

With noisy processes, **more samples** are needed to isolate the signal.

- But GPs buckle under the weight of even modestly big data.
- We must decompose an $N \times N$ matrix, to obtain K_N^{-1} and $|K_N|$, at $\mathcal{O}(N^3)$ cost.

What can be done?

Replication

Woodbury trick

Replication can be a powerful device for **separating signal from noise**, and can yield computational savings as well.

If N outputs are observed at n unique inputs, then the **Woodbury identity** provides $\mathcal{O}(n)$ sufficient statistics rather than $\mathcal{O}(N)$.

$$\mu_N(x) = c_n(x)^\top (C_n + \Lambda_n A_n^{-1}) \bar{Y}$$
$$\sigma_N^2(x) = \nu c_n(x)^\top (C_n + \Lambda_n A_n^{-1})^{-1} c_n(x), \quad \text{where}$$

- $\bar{Y} = (\bar{y}_1, \ldots, \bar{y}_n)^\top$ has averages of a_i replicates at $n \ll N$ unique $\bar{x}_i$
- $A_n = \mathrm{Diag}(a_1, \ldots, a_n)$, and
- $\Lambda_n = \tau^2 \mathbb{I}_n$ for now, with extensions momentarily.

Unlike stochastic kriging (SK: Ankenman, et al., 2010), no asymptopia required.

- These **unique-n predictive equations** establish identies to the full-N analog,
- leading to $\mathcal{O}(n^3)$ rather than $\mathcal{O}(N^3)$ computation.

Log likelihood

The same trick can be played with the (concentrated) log likelihood.

With $\Upsilon_n = C_n + A_n^{-1}\Lambda_n$,

$$\ell = c + \frac{N}{2}\log\hat{\nu}_N - \frac{1}{2}\sum_{i=1}^{n}[(a_i - 1)]\log\lambda_i + \log a_i] - \frac{1}{2}\log|\Upsilon_n|$$

$$\text{where } \hat{\nu}_N = N^{-1}(Y^\top\Lambda_N^{-1}Y - \bar{Y}^\top A_n\Lambda_n^{-1}\bar{Y} + \bar{Y}\Upsilon_n^{-1}\bar{Y}).$$

- Again requires just $\mathcal{O}(n^3)$ operations.

The derivative is available in $\mathcal{O}(n^3)$ time too, e.g.,

$$\frac{\partial\ell}{\partial\cdot} = \frac{N}{2}\frac{\partial(Y^\top\Lambda_N Y - \bar{Y}A_n\Lambda_n^{-1}\bar{Y} + n\hat{\nu}_n)}{\partial\cdot} \times (N\hat{\nu}_N)^{-1} - \frac{1}{2}\sum_{i=1}^{n}\left[(a_i - 1)\frac{\partial\log\lambda_i}{\partial\cdot}\right] - \frac{1}{2}\mathrm{tr}\left(\Upsilon_n^{-1}\frac{\partial\Upsilon_n}{\partial\cdot}\right).$$

For example

Revisit our 2d problem from earlier with

- $n = 100$ unique input locations, expanding to $N \approx 2500$ when
- each has a random number of replicates $a_i \sim \text{Unif}\{1, 2, \ldots, 50\}$.

```
Xbar <- randomLHS(100, 2)
Xbar[,1] <- (Xbar[,1] - 0.5)*6 + 1
Xbar[,2] <- (Xbar[,2] - 0.5)*6 + 1
ytrue <- Xbar[,1] * exp(-Xbar[,1]^2 - Xbar[,2]^2)
a <- sample(1:50, 100, replace=TRUE)
N <- sum(a)
X <- matrix(NA, ncol=2, nrow=N)
y <- rep(NA, N)
nf <- 0
for(i in 1:100) {
  X[(nf+1):(nf+a[i]),] <- matrix(rep(Xbar[i,], a[i]), ncol=2, byrow=TRUE)
  y[(nf+1):(nf+a[i])] <- ytrue[i] + rnorm(a[i], sd=0.01)
  nf <- nf + a[i]
}
```

Comparing full-N to unique-n

```
library(hetGP)
```

- Let's use our new `hetGP` package to fit "full-N" and "unique-n" GPs.

```
Lwr <- rep(sqrt(.Machine$double.eps),2); Upr <- rep(10,2)
fN <- mleHomGP(list(X0=X, Z0=y, mult=rep(1,N)), y, Lwr, Upr)
un <- mleHomGP(X, y, Lwr, Upr)
```

Essentially no difference on lengthscale hyperparameter estimates $\hat{\theta}$ …

```
cbind(rbind(fN=fN$theta, un=un$theta), time=c(fN$time, un$time))
```

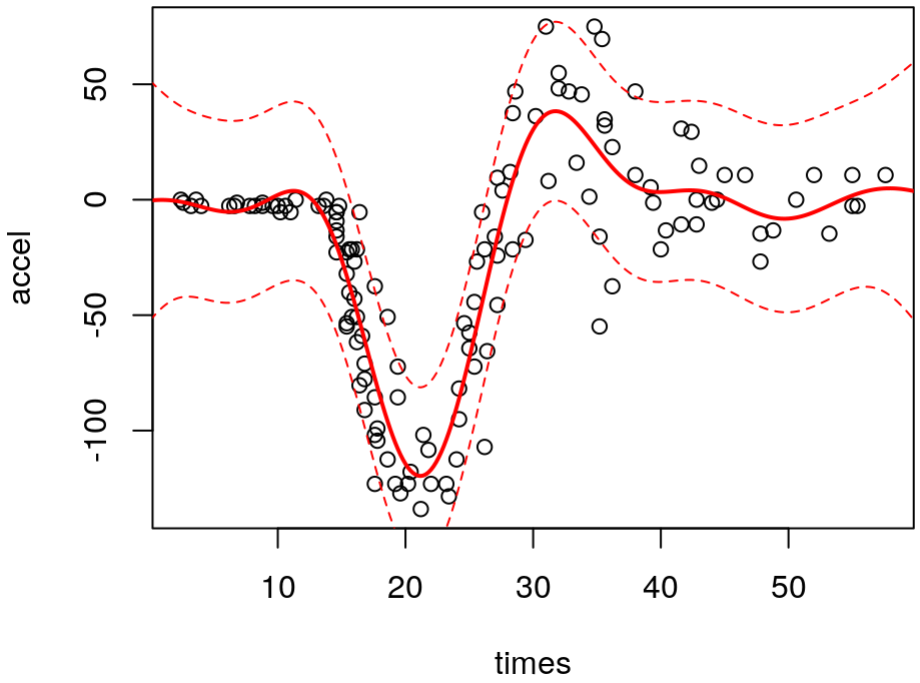
```
##                          time
## fN 1.169473 1.935809 198.297
## un 1.168974 1.936552   0.033
```

… but a big difference in time!

Heteroskedasticity

GPs disappoint on the motorcycle data.

```
library(MASS); hom <- mleHomGP(mcycle$times, mcycle$accel)
Xgrid <- matrix(seq(0, 60, length=301), ncol=1)
p <- predict(x=Xgrid, object=hom)
plot(mcycle); lines(Xgrid, p$mean, col=2, lwd=2)
lines(Xgrid, qnorm(0.05, p$mean, sqrt(p$sd2 + p$nugs)), col=2, lty=2)
lines(Xgrid, qnorm(0.95, p$mean, sqrt(p$sd2 + p$nugs)), col=2, lty=2)
```



Latent GP noise

Although there are many appealing methods for **heteroskedastic GP modeling**, predominantly from the machine learning literature (e.g., Goldberg, et al., 1998),

- most are impractical (MCMC), except on the smallest data sets.

The key ingredient, of **latent noise variables** $\delta_1, \delta_2, \ldots, \delta_n$ stored diagonally in Δ_n, **kept smooth** under a (log) GP prior, has merit.

$$\Delta_n \sim \mathcal{N}_n(0, \nu^2_{(g)}(C_{(g)} + gA_n^{-1}))$$

Then derive predictive (smoothed) λ_i values for plugging into the "mean" GP:

$$\Lambda_n = C_{(g)}(C_{(g)} + gA_n^{-1})^{-1}\Delta_n =: C_{(g)}\Upsilon^{-1}_{(g)}\Delta_n.$$

Smoothly varying variance Λ_n, or $\log \Lambda_n$

- generalize $\Lambda_n = \tau^2 \mathbb{I}_n$ from our earlier homoskedastic setup,
- bridging the gap between constant variance and independent variance (SK).

Latent learning

Rather than Goldberg's MCMC, we can **stay within a (Woodbury) MLE framework**, by defining a joint log likelihood over both (mean and variance) GPs

$$\tilde{\ell} = c - \frac{N}{2}\log \hat{\nu}^2 - \frac{1}{2}\sum_{i=1}^{n}\left[(a_i - 1)\log \lambda_i + \log a_i\right] - \frac{1}{2}\log|\Upsilon_n| - \frac{n}{2}\log \hat{\nu}_{(g)} - \frac{1}{2}\log|\Upsilon_{(g)}|.$$

Maximization is facilitated via **closed form derivatives** with respect to the latent Δ_n values, all in $\mathcal{O}(n^3)$ time.

$$\frac{\partial \ell}{\partial \Delta_n} = \frac{\partial \Lambda_n}{\partial \Delta_n}\frac{\partial \log L}{\partial \Lambda_n} = C_{(g)}\Upsilon_{(g)}^{-1}\frac{\partial \ell}{\partial \Lambda_n}$$

$$\text{where} \quad \frac{\partial \ell}{\partial \lambda_i} = \frac{N}{2} \times \frac{\frac{a_i s_i^2}{\lambda_i^2} + \frac{(\Upsilon_n^{-1}\bar{Y}_n)_i^2}{a_i}}{\hat{\nu}_N} - \frac{a_i - 1}{2\lambda_i} - \frac{1}{2a_i}(\Upsilon_n)_{i,i}^{-1}$$

Full mean and noise inference

Joint likelihood based inference via derivatives has been automated in `hetGP`.

```
het2 <- mleHetGP(mcycle$times, mcycle$accel,
  settings=list(initStrategy='smoothed'), covtype="Matern5_2")
het2$time
```

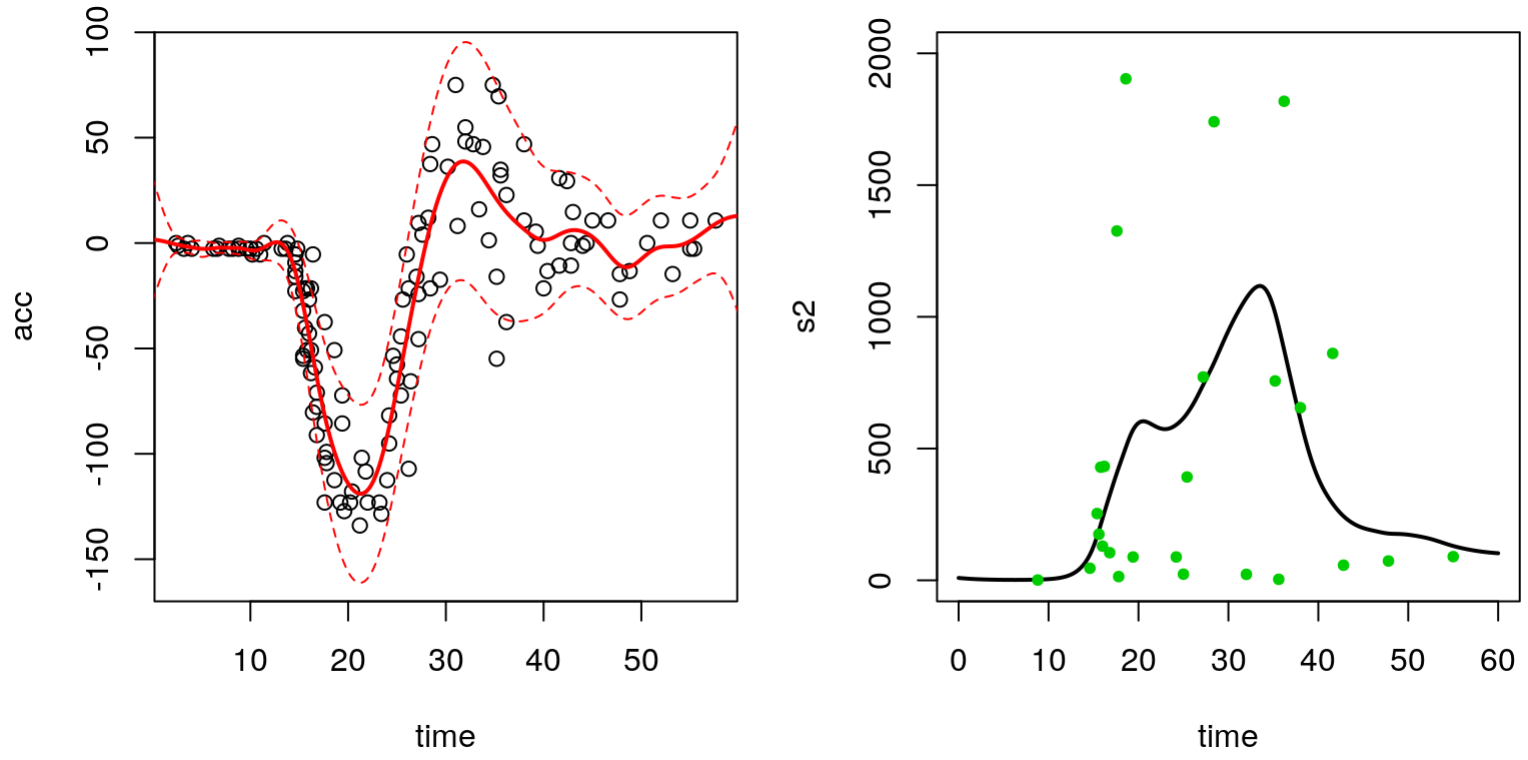
```
## elapsed
##   0.294
```

- Not too bad, time wise.

Predictions and summary stats to visualize on the next slide.

```
p2 <- predict(x=Xgrid, object=het2)
ql <- qnorm(0.05, p2$mean, sqrt(p2$sd2 + p2$nugs))
qu <- qnorm(0.95, p2$mean, sqrt(p2$sd2 + p2$nugs))
```

```
par(mfrow=c(1,2))
plot(mcycle$times, mcycle$accel, ylim=c(-160,90), ylab="acc", xlab="time")
lines(Xgrid, p2$mean, col=2, lwd=2)
lines(Xgrid, ql, col=2, lty=2); lines(Xgrid, qu, col=2, lty=2)
plot(Xgrid, p2$nugs, type="l", lwd=2, ylab="s2", xlab="time", ylim=c(0, 2e3))
points(het2$X0, sapply(find_reps(mcycle[,1],mcycle[,2])$Zlist,var), col=3, pch=20)
```



29/61

Real examples

Example: Epidemics management

One method of studying disease outbreak dynamics is based on stochastic compartmental modeling,

- for example, so-called Susceptible, Infected & Recovered (SIR) models.

Here we consider the total number of newly infected individuals

$$f(x) := \mathbb{E}\left\{S_0 - \lim_{T\to\infty} S_T \mid (S_0, I_0, R_0) = x\right\} = \gamma\mathbb{E}\left\{\int_0^\infty I_t\, dt \mid x\right\}$$

under continuous time Markov dynamics with transitions $S + I \to 2I$ and $I \to R$, solved with Monte Carlo (Hu et al., 2015).

- The inputs are in 2d: $x = (S_0, I_0)$;
- the resulting surface is heteroskedastic
- and has some very-high noise regions.

Consider a space-filling design of size $n = 200$ unique runs, with a random number of replicates $a_i \in \{1, \ldots, 100\}$, for $i = 1, \ldots, n$.

```
Xbar <- randomLHS(200, 2)
a <- sample(1:100, nrow(Xbar), replace=TRUE)
X <- matrix(NA, ncol=2, nrow=sum(a))
nf <- 0
for(i in 1:nrow(Xbar)) {
  X[(nf+1):(nf+a[i]),] <- matrix(rep(Xbar[i,], a[i]), ncol=2, byrow=TRUE)
  nf <- nf + a[i]
}
nf
```

```
## [1] 9565
```

The `hetGP` package provides `sirEval`, returning the expected number of infecteds at the end of the simulation.

```
Y <- apply(X, 1, sirEval)
```

SIR hetGP fit

```
fit <- mleHetGP(X, Y, lower=rep(0.05, 2), upper=rep(10, 2),
  settings = list(linkThetas="none"), covtype="Matern5_2", maxit=1e4)
fit$time
```

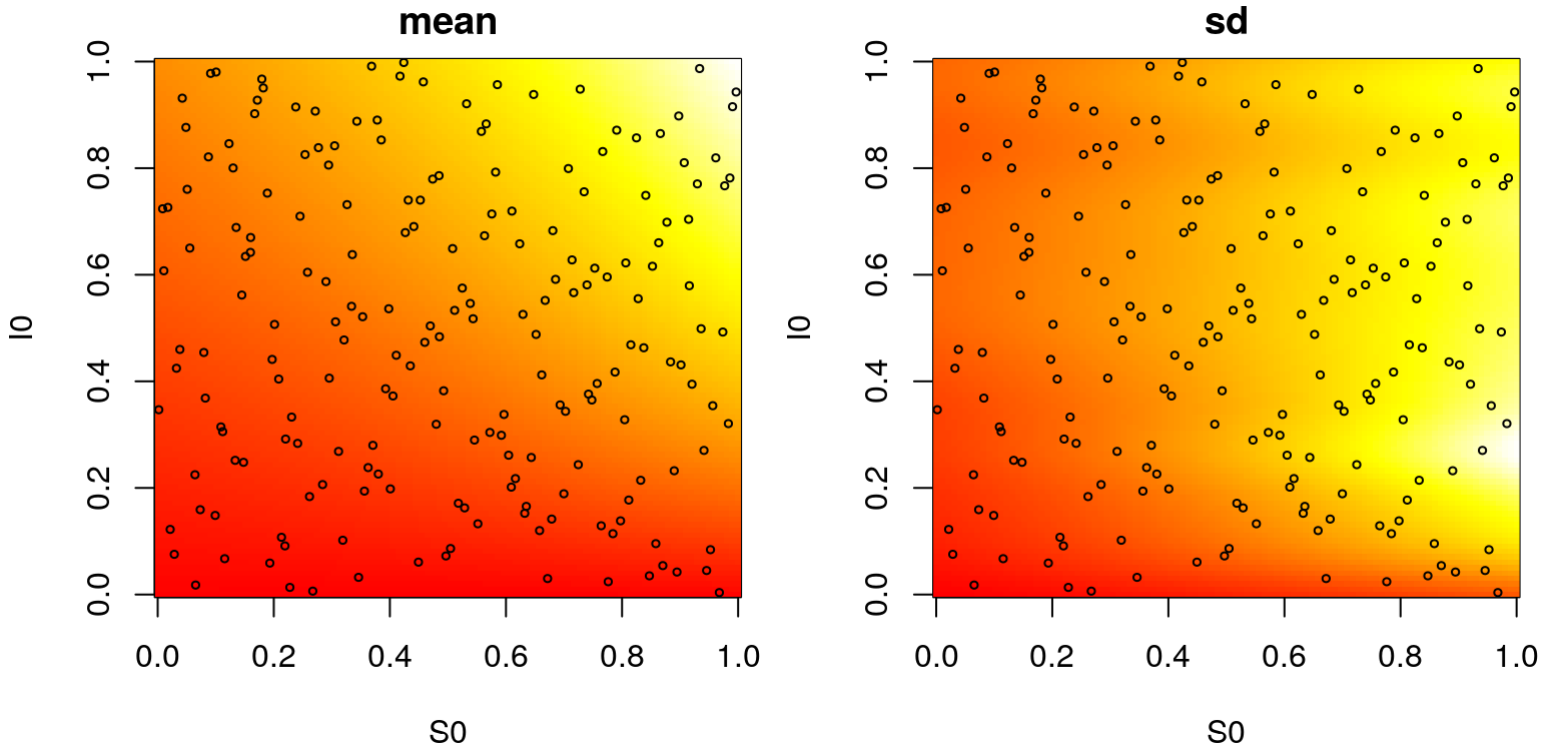
```
## elapsed
##   2.593
```

- Around 2.6 seconds to train the model; not bad for 10K runs.

To help with the visuals on the next slide, the code below creates a dense grid in 2D and calls the `predict` method on the `"hetGP"`-class fit object.

```
xx <- seq(0, 1, length = 100)
XX <- as.matrix(expand.grid(xx, xx))
p <- predict(fit, XX)
psd <- sqrt(p$sd2 + p$nugs)
```

```
par(mfrow=c(1,2), mar=c(4,4,2,1))
image(xx, xx, matrix(p$mean, 100), xlab="S0", ylab="I0", main="mean", col=cols)
points(Xbar, cex=0.5)
image(xx, xx, matrix(psd, 100), xlab="S0", ylab="I0", main="sd", col=cols)
points(Xbar, cex=0.5)
```



Example: Ocean oxygen

Herbei & Berliner (2014) describe a Feynman-Kac simulator modeling the concentration of a tracer within a given spatial domain.

- One application is modeling the oxygen concentration in a thin water layer deep in the ocean (McKeague, et al., 2005).

Source adapted from this GitHub page.

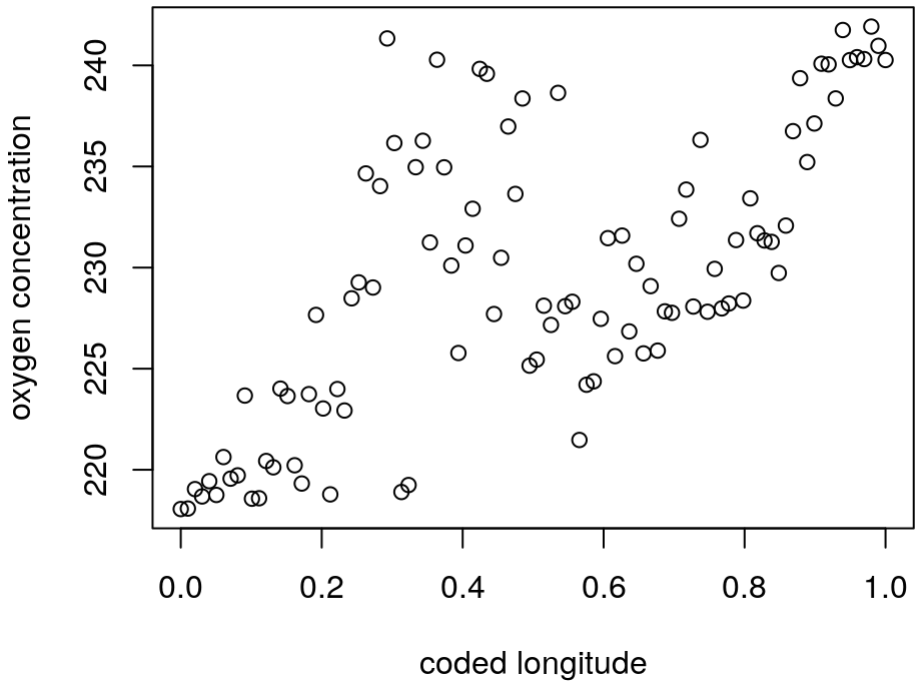
```
source("fksim.R")
```

There are four real-valued inputs: two spatial coordinates (longitude and latitude) and two diffusion coefficients.

- Interest is in learning, i.e., **inverting**, the diffusion coefficients from field data.
- For now lets focus on the spatial coordinates.

Using defaults for the diffusion coefficients, consider simulations along a longitudinal slice of the input space.

```
x2 <- seq(0, 1, length=100)
y <- sapply(x2, function(x2) { fksim(c(0.8, x2)) })
plot(x2, y, xlab="coded longitude", ylab="oxygen concentration")
```



Separating signal from noise in 2d (and 4d) is harder.

Consider a 2d design of $n = 25$ unique sites with twenty replicates ($N = 500$).

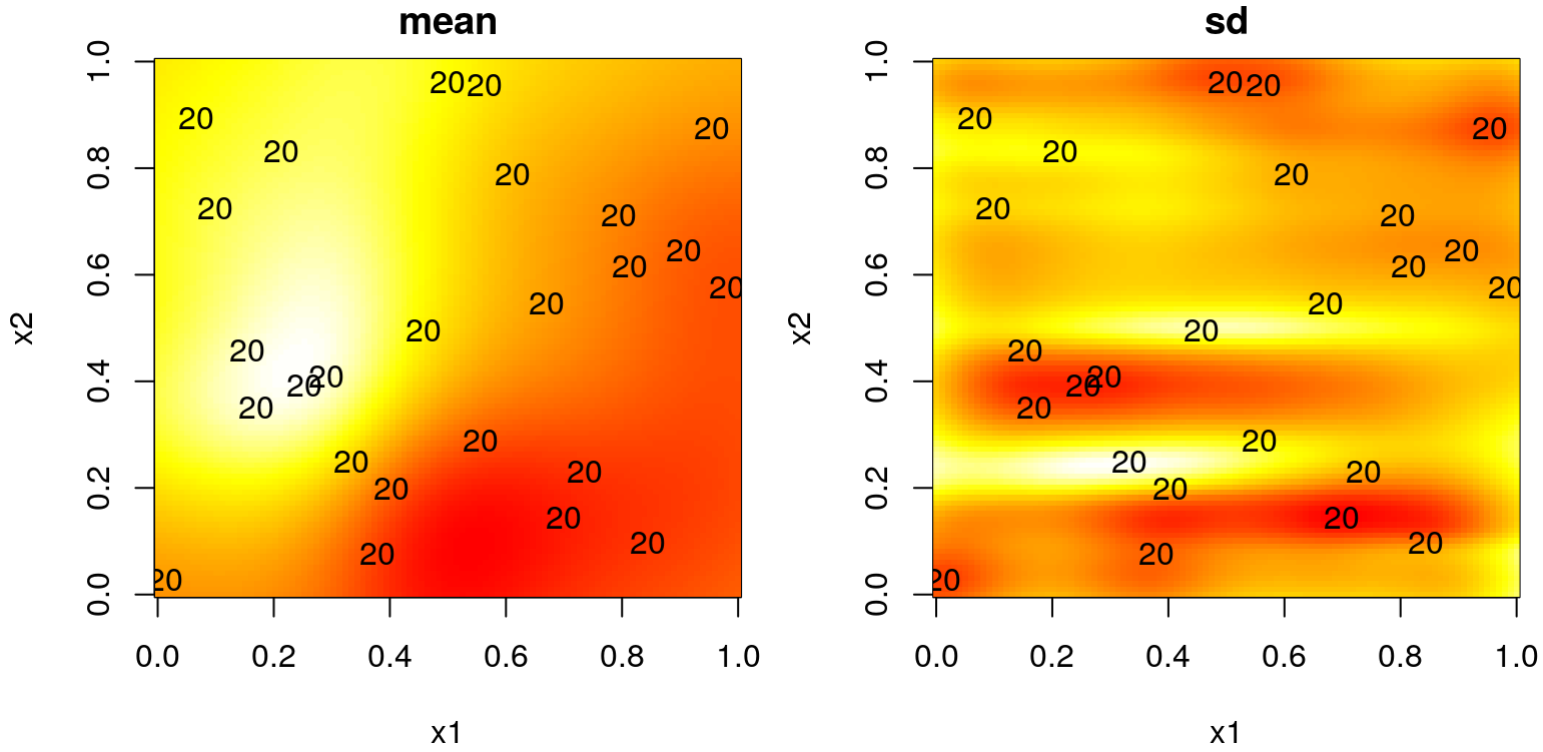
```
n <- 25
Xn <- randomLHS(n, 2)
XN <- matrix(rep(t(Xn), 20), ncol=2, byrow=TRUE)
YN <- apply(XN, 1, fksim)
```

Fit `hetGP`.

```
lower <- rep(0.01, 2); upper <- rep(30, 2); covtype <- "Matern5_2"
noiseControl <- list(g_min=1e-6, g_bounds=c(1e-6, 1), lowerDelta=log(1e-6))
settings <- list(linkThetas="none", initStrategy="smoothed", return.hom=TRUE)
oxhet <- mleHetGP(XN, YN, lower=lower, upper=upper, covtype=covtype,
  noiseControl=noiseControl, settings=settings, maxit=1000)
pox <- predict(oxhet, XX); psd <- sqrt(pox$sd2 + pox$nugs)
oxhet$time
```

```
## elapsed
##   0.069
```

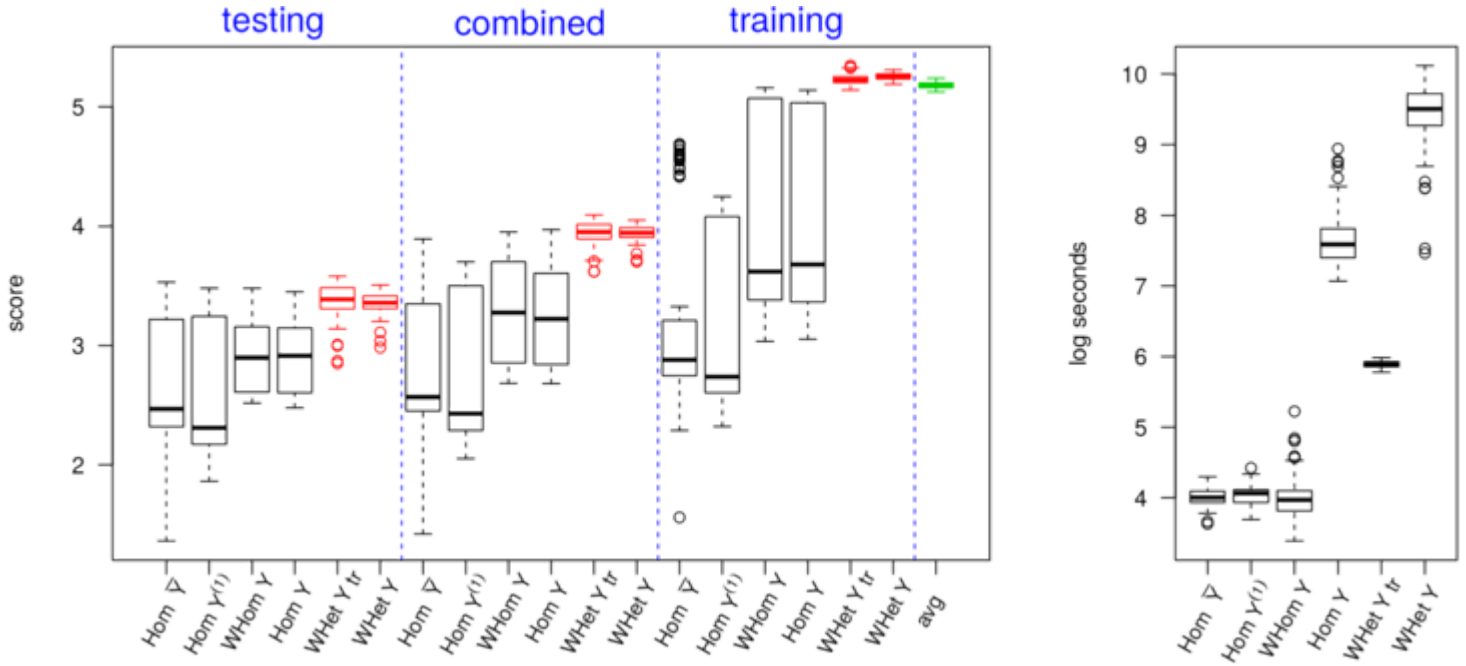
```
par(mfrow=c(1,2), mar=c(4,4,2,1))
image(xx, xx, matrix(pox$mean, ncol=100), col=cols, main="mean",
  xlab="x1", ylab="x2"); text(oxhet$X0[,1], oxhet$X0[,2], oxhet$mult)
image(xx, xx, matrix(psd, ncol=100), col=cols, main="sd",
  xlab="x1", ylab="x2"); text(oxhet$X0[,1], oxhet$X0[,2], oxhet$mult)
```



38/61

Example: assemble to order

Inventory management simulator (Hong & Nelson, 2006); $d = 8$; $N \approx 5000$.



Proper scores (Gneiting & Raftery, 2007) measure mean accuracy (MSE) relative to predicted variance (larger is better).

The `hetGP` package has a saved fit on space-filling training/testing data from this experiment.

```
data("ato")
c(n=nrow(Xtrain), N=length(unlist(Ztrain)))
```

```
##    n    N
## 1000 5594
```

Reproducing the comparison on proper scores:

```
phet <- predict(out, Xtest)
phets2 <- phet$sd2 + phet$nugs
mhet <- as.numeric(t(matrix(rep(phet$mean, 10), ncol=10)))
s2het <- as.numeric(t(matrix(rep(phets2, 10), ncol=10)))
sehet <- (unlist(t(Ztest)) - mhet)^2
sc <- -sehet/s2het - log(s2het)
mean(sc)
```

```
## [1] 3.396427
```

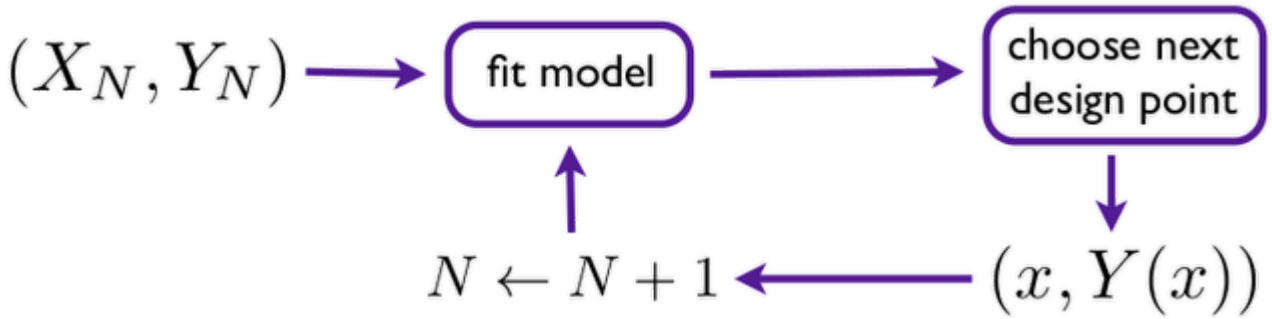
Sequential design

One step at a time

Model-based **one-shot** design is almost never appropriate in this setting.

- Designs are hyperparameter sensitive for homoskedastic processes,
- which is exacerbated when additional variance processes are in play.

It makes sense to slow down and take things one step at a time.



Choose the next point (x_{N+1}) by exploring its impact on the predictive equations.

IMSPE

A common criteria is **integrated mean-square prediction error**

$$I_{n+1} \equiv \text{IMSPE}(x_1, \ldots, x_n, x_{n+1}) = \int_{x \in \mathcal{X}} \sigma^2_{n+1}(x)\, dx.$$

IMSPE has a closed form as long as $\mathcal{X}$ is an easily integrable domain, such as a hyperrectangle.

$$\begin{aligned} I_{n+1} &= \mathbb{E}\{\sigma^2_{n+1}(X)\} = \mathbb{E}\{K_\theta(X,X) - k_{n+1}(X)^\top K_{n+1}^{-1} k_{n+1}(X)\} \\ &= \mathbb{E}\{K_\theta(X,X)\} - \text{tr}(K_{n+1}^{-1} W) \end{aligned}$$

- where $W_{ij} = \int_{x \in \mathcal{X}} k(x_i, x) k(x_j, x)\, dx$. E.g., in the Gaussian case,

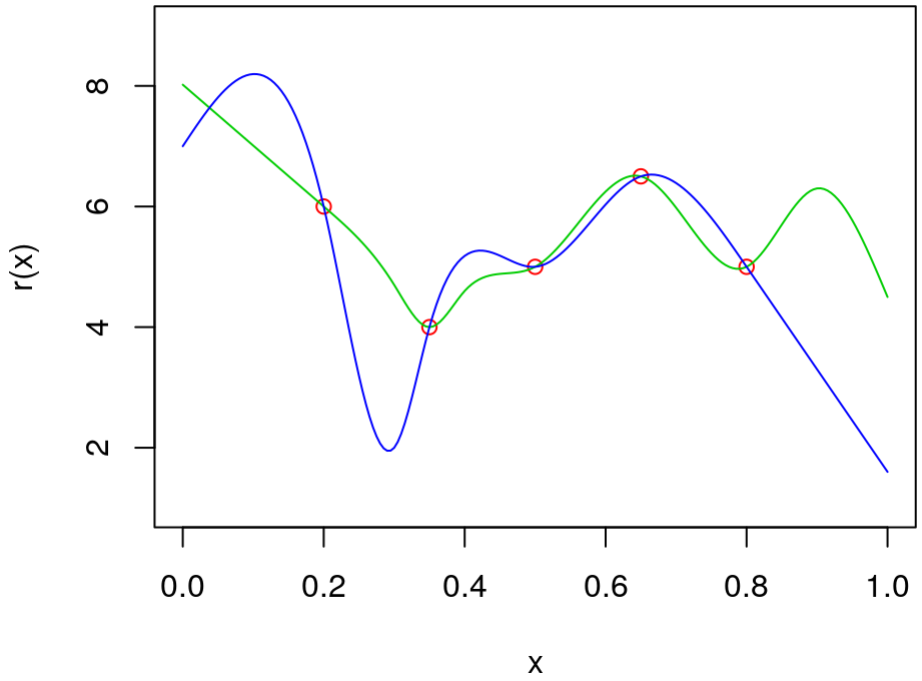
$$W_{ij} = \prod_{k=1}^{d} \frac{\sqrt{2\pi\theta_k}}{4} \exp\left\{ -\frac{(x_{i,k} - x_{j,k})^2}{2\theta_k} \right\} \left[\text{erf}\left\{ \frac{2 - (x_{i,k} + x_{j,k})}{\sqrt{2\theta_k}} \right\} + \text{erf}\left\{ \frac{x_{i,k} + x_{j,k}}{\sqrt{2\theta_k}} \right\} \right].$$

- Gradient of $I_{n+1} \mid x_1, \ldots, x_n$ facilitates optimization for **sequential** design.

Replication in sequential design

Consider evaluations of the *true* noise $r(x)$ and two potential interpolations.

```
plot(X0, rn, xlab="x", ylab="r(x)", xlim=c(0,1), ylim=c(1,9), lty=2, col=2)
lines(XXg, r1(XXg), col=3); lines(XXg, r2(XXg), col=4)
```



From $r(x)$ to IMSPE(x)

Now, consider IMSPE calculated using those noise surfaces …

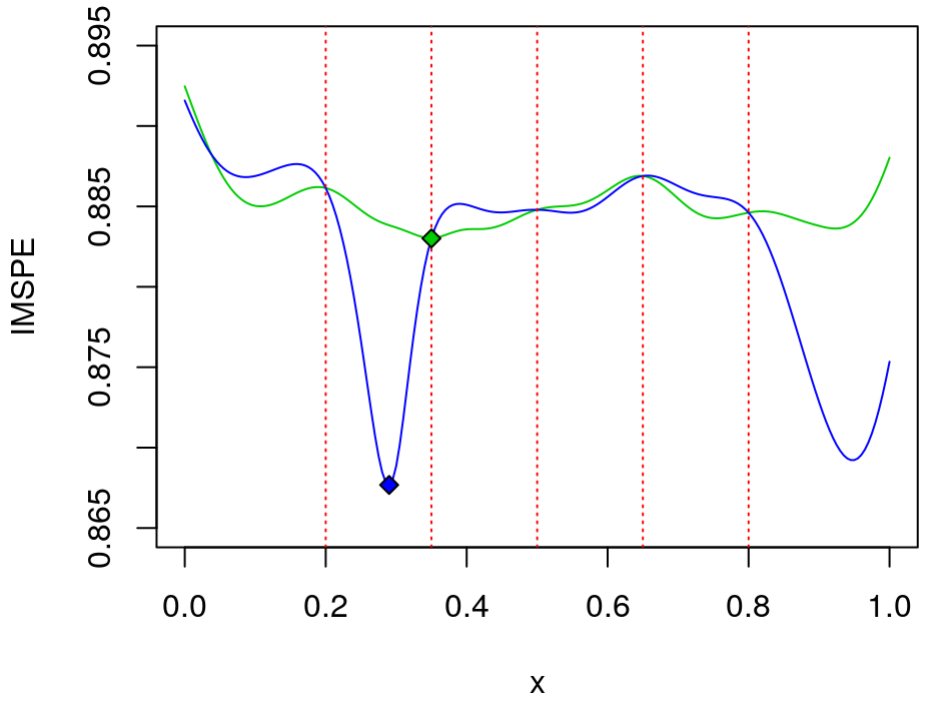
```
IMSPE.r <- function(x, X0, theta, r)
 {
  x <- matrix(x, nrow=1)
  Wijs <- Wij(mu1=rbind(X0, x), theta=theta, type="Gaussian")

  K <- cov_gen(X1 = rbind(X0, x), theta=theta)
  K <- K + diag(apply(rbind(X0, x), 1, r))

  return(1 - sum(solve(K) * Wijs))
 }
```

… and evaluated on a predictive grid.

```
imspe1 <- apply(XXg, 1, IMSPE.r, X0=X0, theta=0.01, r=r1)
imspe2 <- apply(XXg, 1, IMSPE.r, X0=X0, theta=0.01, r=r2)
xstar1 <- which.min(imspe1)
xstar2 <- which.min(imspe2)
```

The **green** noise interpolation **prefers replication**.

```
plot(XXg, imspe1, type="l", col=3, ylab="IMSPE", xlab="x", ylim=c(0.865, 0.895))
lines(XXg, imspe2, col=4)
abline(v=X0, lty=3, col = 'red')
points(XXg[xstar1], imspe1[xstar1], pch=23, bg=3)
points(XXg[xstar2], imspe2[xstar2], pch=23, bg=4)
```



A replicating relation

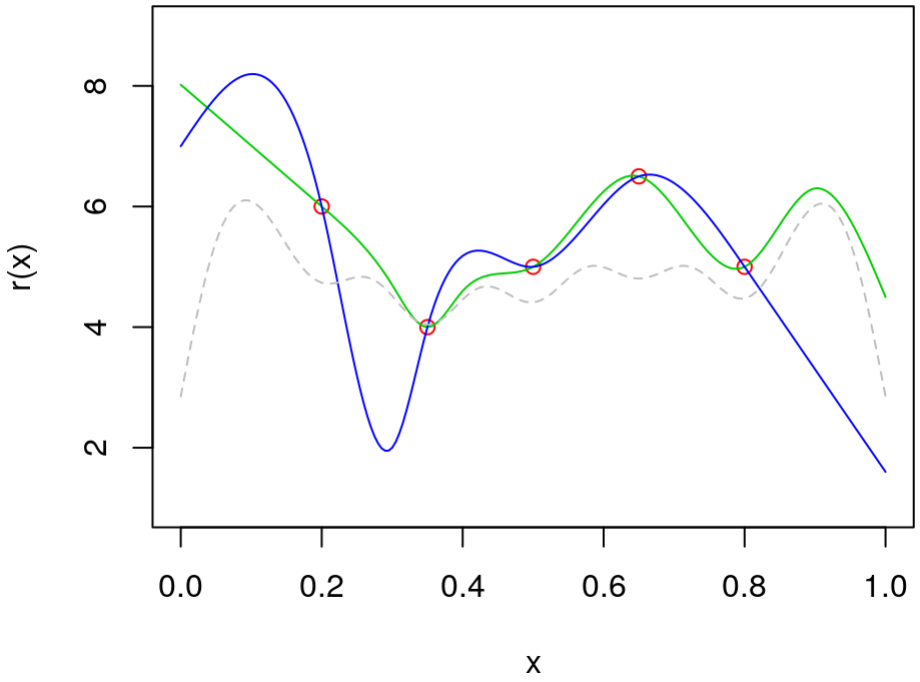
You can show that the next point, x_{N+1}, will be a replicate when

$$r(x_{N+1}) \geq \frac{k_n(x_{N+1})^\top K_n^{-1} W_n K_n^{-1} k_n(x_{N+1}) - 2w_{n+1}^\top K_n^{-1} k_n(x_{N+1}) + w_{n+1,n+1}}{\mathrm{tr}(B_{k^*} W_n)} - \sigma_n^2(x_{N+1}),$$

where $k^* = \mathrm{argmin}_{1 \leq k \leq n} \mathrm{IMSPE}(x_k)$ and $B_k = \frac{(\Upsilon_n^{-1})_{\cdot,k}(\Upsilon_n^{-1})_{k,\cdot}}{\frac{\nu\lambda_k}{a_k(a_k+1)} - (\Upsilon_n)^{-1}_{k,k}}$.

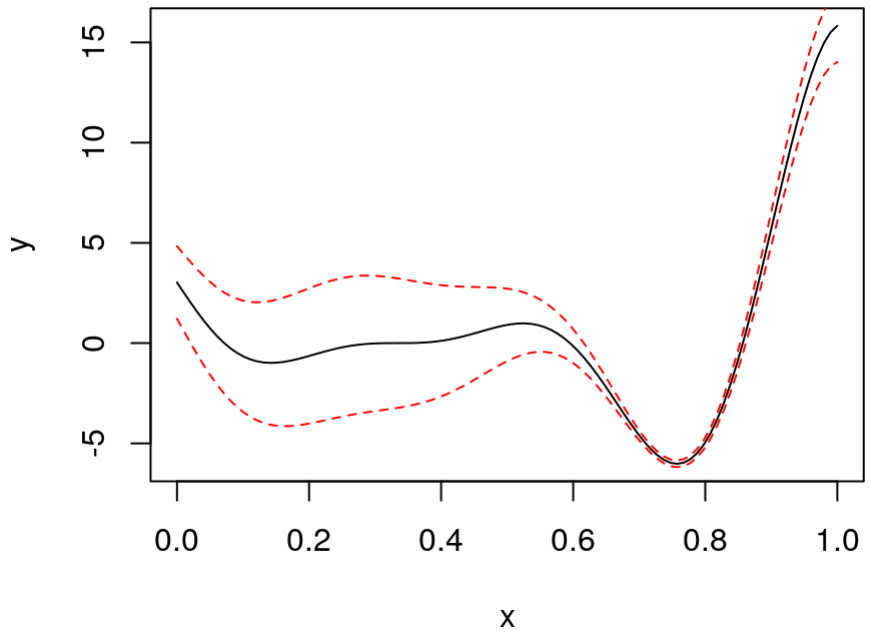
```
rx <- function(x, X0, rn, theta, Ki, kstar, Wijs)
 {
   x <- matrix(x, nrow=1); kn1 <- cov_gen(x, X0, theta=theta)
   wn <- Wij(mu1=x, mu2=X0, theta=theta, type="Gaussian")
   a <- kn1 %*% Ki %*% Wijs %*% Ki %*% t(kn1) - 2*wn %*% Ki %*% t(kn1)
   a <- a + Wij(mu1=x, theta=theta, type="Gaussian")
   Bk <- tcrossprod(Ki[,kstar], Ki[kstar,]) / (2/rn - Ki[kstar, kstar])
   b <- sum(Bk * Wijs); sn <- 1 - kn1 %*% Ki %*% t(kn1)
   return(a/b - sn)
 }
```

```
bestk <- which.min(apply(X0, 1, IMSPE.r, X0=X0, theta=0.01, r=r1))
Wijs <- Wij(X0, theta=0.01, type="Gaussian")
Ki <- solve(cov_gen(X0, theta=0.01, type="Gaussian") + diag(rn))
rx.thresh <- apply(XXg, 1, rx, X0=X0,rn=rn,theta=0.01,Ki=Ki,kstar=bestk,Wijs=Wijs)
plot(X0, rn, xlab="x", ylab="r(x)", xlim=c(0,1), ylim=c(1,9), lty=2, col=2)
lines(XXg, r1(XXg), col=3); lines(XXg, r2(XXg), col=4)
lines(XXg, rx.thresh, lty=2, col="grey")
```



Simple (more realistic) example

```
fm <- function(x) { ((x*6-2)^2)*sin((x*6-2)*2) }
fn <- function(x) { (1.1 + sin(x*2*pi)) }
xgrid <- seq(0,1, length=100); plot(xgrid, fm(xgrid), type="l",xlab="x",ylab="y")
lines(xgrid, qnorm(0.05, fm(xgrid), fn(xgrid)), col=2, lty=2)
lines(xgrid, qnorm(0.95, fm(xgrid), fn(xgrid)), col=2, lty=2)
```



Starting up

Here is a fit to a small space-filling design.

```
fr <- function(x) { fm(x) + rnorm(length(x), sd=fn(x)) }
X <- seq(0, 1, length=10); Y <- fr(X)
mod <- mleHetGP(X=X, Z=Y, lower=0.0001, upper=10)
```

Followed by a search via IMSPE with $h = 3$ look-ahead over replication.

```
opt <- IMSPE_optim(mod, h=3)
c(X, opt$par)
```

```
##  [1] 0.00000000 0.11111111 0.22222222 0.33333333 0.44444444 0.55555556
##  [7] 0.66666667 0.77777778 0.88888889 1.00000000 0.05749379
```

Update the fit to incorporate the new data.

```
X <- c(X, opt$par); Ynew <- fr(opt$par); Y <- c(Y, Ynew)
mod <- update(mod, Xnew=opt$par, Znew=Ynew, ginit=mod$g*1.01)
```

Repeat a bunch

Let's continue and gather a total of $N = 500$ samples in this way.

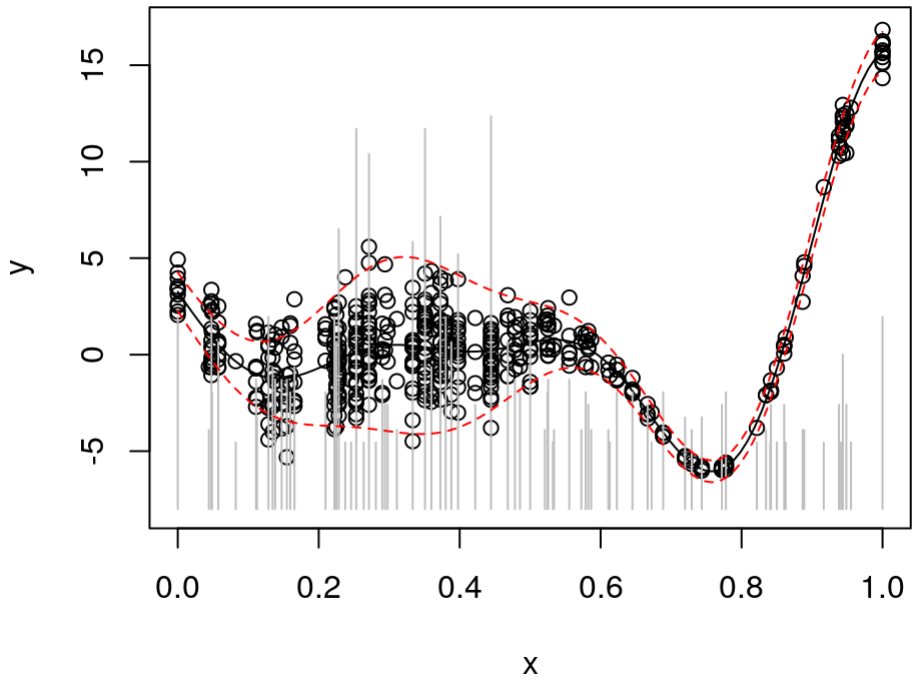
```
for(i in 1:489) {

  ## find the next point and update
  opt <- IMSPE_optim(mod, h=3)
  X <- c(X, opt$par); Ynew <- fr(opt$par); Y <- c(Y, Ynew)
  mod <- update(mod, Xnew=opt$par, Znew=Ynew, ginit=mod$g*1.01)

  ## periodically restart model fit
  if(i %% 25 == 0){
    mod2 <- mleHetGP(X=list(X0=mod$X0, Z0=mod$Z0, mult=mod$mult), Z=mod$Z,
      lower=0.0001, upper=1)
    if(mod2$ll > mod$ll) mod <- mod2
  }
}
```

Once that's done, gather a final prediction throughout the input space.

```
p <- predict(mod, matrix(xgrid, ncol=1))
pvar <- p$sd2 + p$nugs
```

Visualizing

```
plot(xgrid, p$mean, type="l", ylim=c(-8,17), xlab="x", ylab="y"); points(X, Y)
segments(mod$X0, rep(0, nrow(mod$X0))-8, mod$X0, (mod$mult-8)*0.65, col="gray")
lines(xgrid, qnorm(0.05, p$mean, sqrt(pvar)), col=2, lty=2)
lines(xgrid, qnorm(0.95, p$mean, sqrt(pvar)), col=2, lty=2)
```



Real examples

Example: sequential ocean oxygen

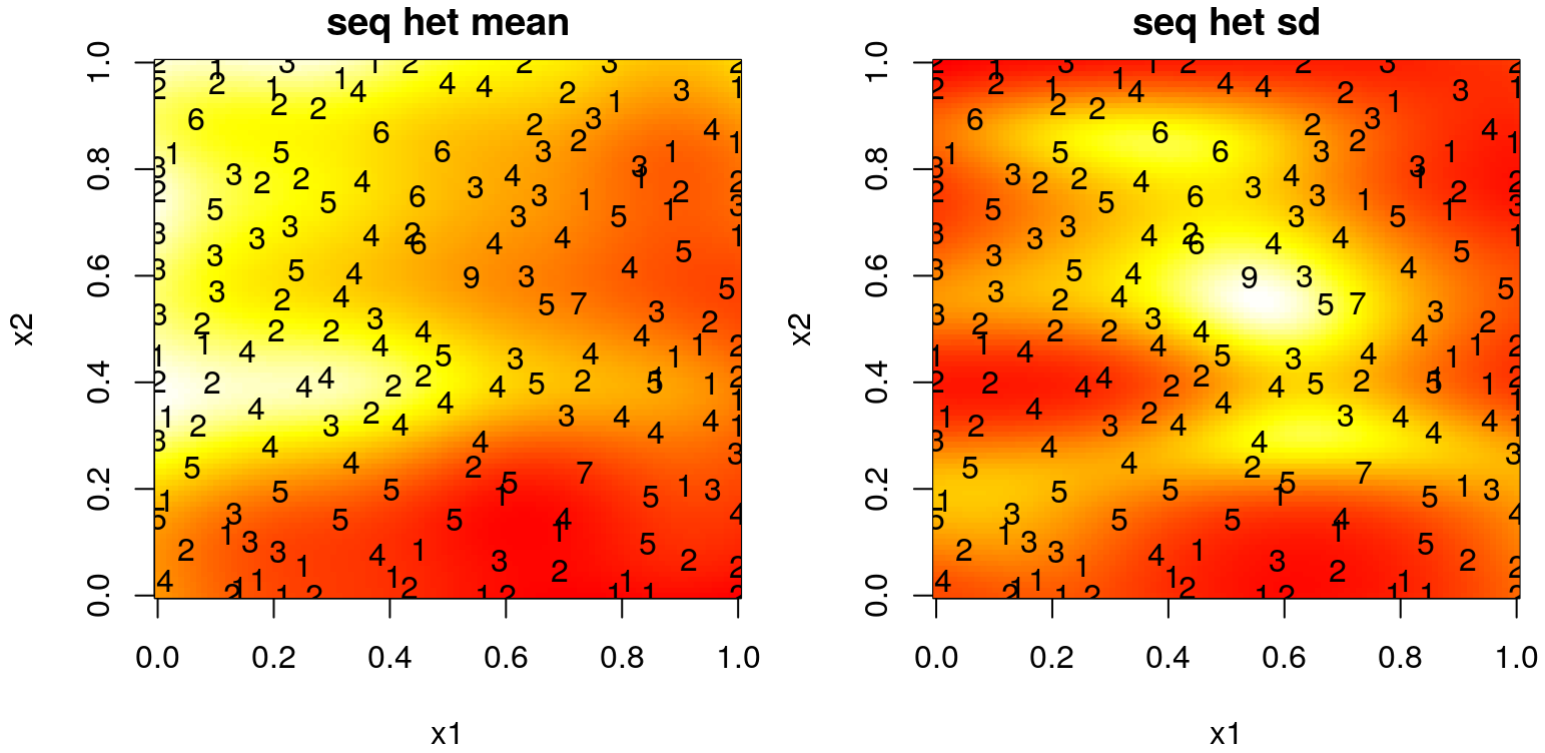
Still 2d, initialize `hetGP` fit with the first four replicates from earlier.

```
n <- nrow(oxhet$X0)
X <- XN[1:(4*n),]; Y <- YN[1:(4*n)]
oxseq <- mleHetGP(X, Y, lower=lower, upper=upper, covtype=covtype,
  noiseControl=noiseControl, settings=settings, maxit=1000)
```

Then loop over active learning acquisitions.

```
control <- list(tol_dist=1e-4, tol_diff=1e-4, multi.start=30)
for(i in 1:(length(oxhet$Z)-length(oxseq$Z))) {
  opt <- IMSPE_optim(oxseq, horizon(oxseq), control=control)
  ynew <- fksim(opt$par)
  oxseq <- update(oxseq, Xnew=opt$par, Znew=ynew, ginit=oxseq$g*1.01)
  if(i %% 25 == 0){
    oxseq2 <- mleHetGP(list(X0=oxseq$X0, Z0=oxseq$Z0, mult=oxseq$mult), Z=oxseq$Z,
      lower=lower, upper=upper, covtype=covtype, noiseControl=noiseControl,
      settings=settings, maxit=1000)
    if(oxseq2$ll > oxseq$ll) oxseq <- oxseq2
  }
}
```

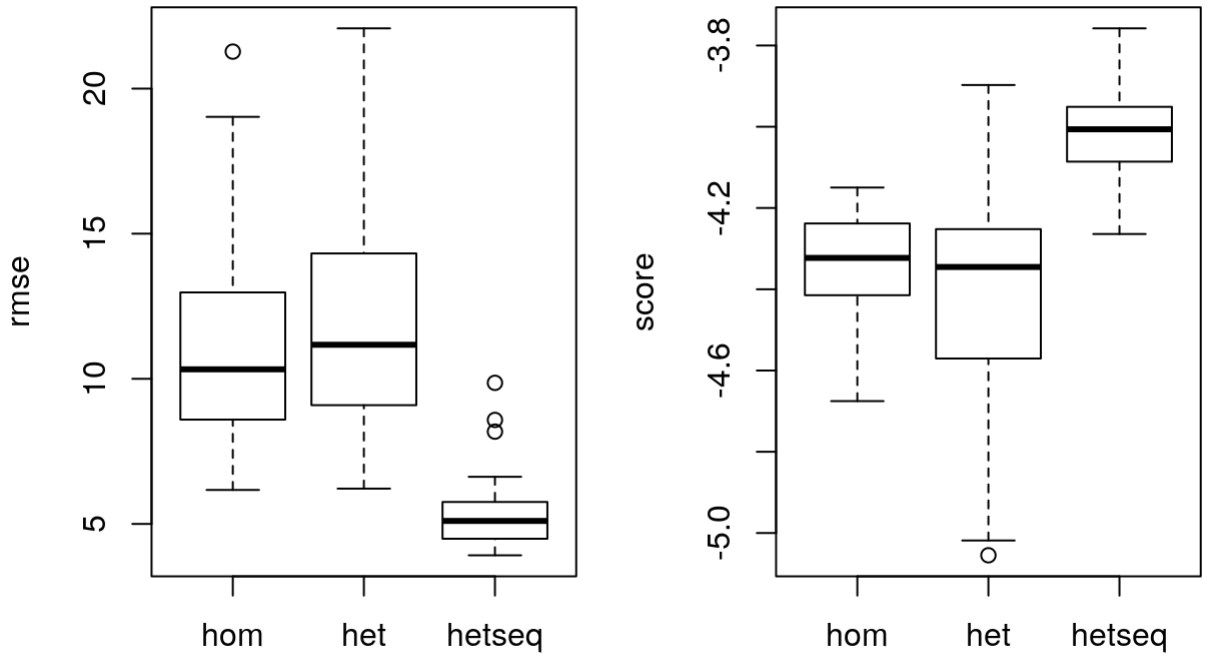
```
par(mfrow=c(1,2), mar=c(4,4,2,1))
pseq <- predict(oxseq, XX); seqsd <- sqrt(pseq$sd2 + pseq$nugs)
image(xx, xx, matrix(pseq$mean, ncol=100), col=cols, main="seq het mean",
  xlab="x1", ylab="x2"); text(oxseq$X0[,1], oxseq$X0[,2], oxseq$mult)
image(xx, xx, matrix(seqsd, ncol=100), col=cols, main="seq het sd",
  xlab="x1", ylab="x2"); text(oxseq$X0[,1], oxseq$X0[,2], oxseq$mult)
```



A potent combination

Now imagine repeating that a bunch of times; see ocean_seqdes_sol.R.

```
load("ocean_seqdes_sol.RData")
par(mfrow=c(1,2)); boxplot(rmses, ylab="rmse"); boxplot(scores, ylab="score")
```



Example: sequential ATO

The `ato` data object contains a second saved `"hetGP"`-class model

- trained with an adaptive horizon IMSPE-based sequential design scheme.

Design size and training time are quoted below.

```
data("ato")
rbind(batch=c(n=nrow(out$X0), N=length(out$Z)),
  adapt=c(n=nrow(out.a$X0), N=length(out.a$Z)))
```

```
##          n    N
## batch 1000 5594
## adapt 1194 2000
```

The adaptive design has a higher proportion of unique locations

- but still a nontrivial degree of replication,
- resulting in **many fewer overall** runs of the expensive ATO simulator.

Out-of-sample comparison

With the same out-of-sample testing set from the previous score-based comparison,

- the code below calculates predictions and scores under **IMSPE**-based sequential design.

```
phet.a <- predict(out.a, Xtest)
phets2.a <- phet.a$sd2 + phet.a$nugs
mhet.a <- as.numeric(t(matrix(rep(phet.a$mean, 10), ncol=10)))
s2het.a <- as.numeric(t(matrix(rep(phets2.a, 10), ncol=10)))
sehet.a <- (unlist(t(Ztest)) - mhet.a)^2
sc.a <- -sehet.a/s2het.a - log(s2het.a)
c(batch=mean(sc), adapt=mean(sc.a))
```

```
##    batch    adapt
## 3.396427 3.615494
```

- The adaptive score is better despite the substantially smaller training set.

Summarizing

Thanks

When the mean and the variance are changing non-linearly in the input space

- it is still possible to get very accurate fits via coupled GPs,
- and it can be advantageous to have replication in the design.

The more heteroskedasticity the more replication.

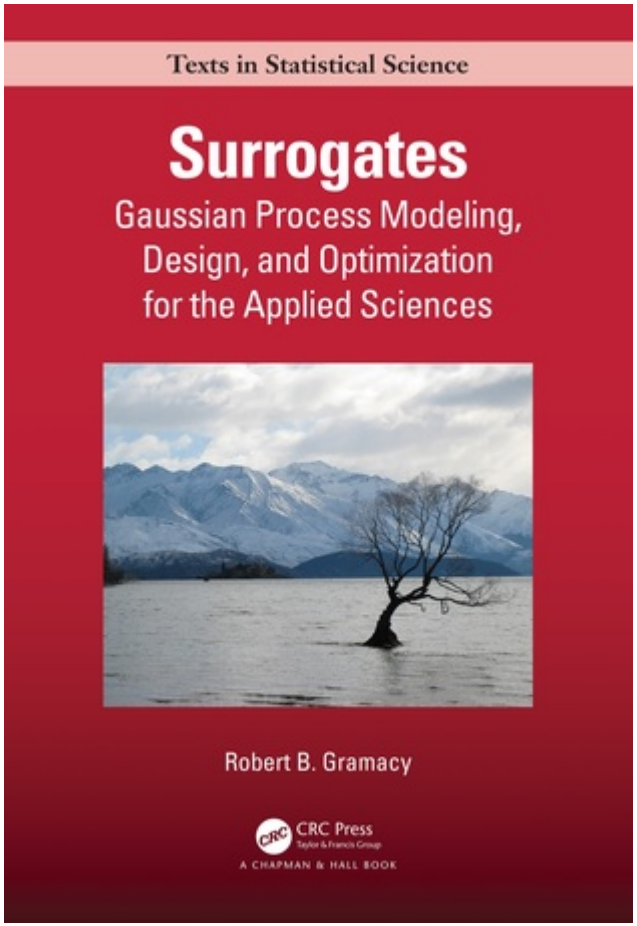
- Intuitively, that must be true: both signal and noise are changing and replication is the only reliable tool for separating the two.

Replication has the added benefit of yielding faster fitting of the GPs.

Our `hetGP` package, facilitating everything in this reproducible `Rmarkdown` talk, is available on CRAN.

- A reproducible vignette can be found with package source.

Fully reproducible book



A graduate textbook at the interface between machine learning, spatial statistics, computer simulation, meta-modeling, design, and optimization.

- GPs for flexible nonparametric and nonlinear modeling
- UQ, sensitivity analysis, calibration, sequential design/active learning and Bayesian optimization
- Treed partitioning, local GP approximation, **heteroskedastic modeling**

bobby.gramacy.com/surrogates