Surrogates 7020

Chapter 6: Model-based Design for GPs

Dr. Alex Bledar Konomi

Department of Mathematical Sciences
University of Cincinnati

Model Based Design

- A model-based design is one where the model says what $\boldsymbol{X}_n$ it wants according to a criterion targeting some aspect of its fit.
- Example targets include the quality of estimates for parameters or hyperparameters, or accuracy of predictions at particular inputs out-of-sample, or over the entire input space.
- With a first-order linear model, you may recall from an elementary design course that maximizing spread in $\boldsymbol{X}_n$ maximizes leverage and therefore minimizes the standard error of $\hat{\beta}$-values.
- An optimal design for learning regression coefficients places inputs at the corners of the experimental region, for better or worse.
- Linear models are highly parametric. Parameter settings are intimately linked to the underlying predictor.
- With GPs, a nonparametric model whose hyperparameterization is only loosely connected to the predictive distribution, you might think a different strategy is required. But principles are largely the same.

Model Based Design

The general program is to specify a criterion $J(\boldsymbol{X}_n)$, and optimize that criterion with respect to $\boldsymbol{X}_n$
. Assuming maximization,

$$X_n^* = \underset{\boldsymbol{X}_n}{\operatorname{argmax}}\, J(\boldsymbol{X}_n),$$

which is typically solved numerically. Except when essential for clarity, I shall generally drop the $*$ superscript in X_n^*, indicating an optimally chosen design.

Maximum Entropy Design

The entropy of a density $p(x)$ is defined as

$$H(X) = -\int_X p(x) \log p(x) dx.$$

Entropy is larger when $p(x)$ is more uniform. You can think of it as a measure of surprise in random draws from the distribution corresponding to p. A uniform draw always yields a surprising, unpredictable, random value. Entropy is maximized for this choice of p. At the other end of the spectrum, a point-spike p, where all density is concentrated on a single point, offers no surprise. A draw from such a distribution yields the same value every time. Its entropy is zero.

AA

Information is the negative of entropy, $I = -H$. More information is less uniformity, less surprise. To see how information and entropy can relate to design, consider the following. Suppose $\mathcal{X}$ is a fixed, finite set of points in the input space. Place latent functions F at $\mathcal{X}$ under a GP prior $p(F|\mathcal{X})$; see 5.3.2. Let I_X denote the information of that prior. Now, denote F restricted to $X \subset \mathcal{X}$ as F_X with implied prior $p(F_X|X)$ and information I_X. For a particular choice $X_n \subset \mathcal{X}$, we have that

$$I_{\mathcal{X}} = I_{X_n} + E(F_{X_n}\{I_{X-X_n|D_n}\},$$

where $D_n = (X_n, Y_n)$ is the completion of X_n with a noise-augmented F_n.

Information partitioned

$$I_{\mathcal{X}} = I_{X_n} + E(F_{X_n}\{I_{X-X_n|D_n}\},$$

This equation in fact partitions information between the amount soaked up, and amount left over after selecting $\boldsymbol{X}_n$. Choosing $\boldsymbol{X}_n$ to maximize the expected information in prediction of latent function values, i.e., the amount soaked up by second term, is equivalent to minimizing the information I_{X_n} left over (maximizing entropy, H_{X_n}) in the distribution of F_{X_n}. A solution to that optimization problem is a so-called maximum entropy (or maxent) design. For more details see Shewry and Wynn (1987), who introduced the concept as a design criterion for spatial models. Maxent was appropriated for computer experiments by Currin et al. (1991) and TJ Mitchell and Scott (1987).

Maximizing the Entropy in Practice

That distribution – either for $F_{\boldsymbol{X}_n}$ or it's noisy analog for $D_n = (\boldsymbol{X}_n, \boldsymbol{Y}_n)$, depending on your preferred interpretation (5.3.2) – ultimately involves:

$$\boldsymbol{Y}_n \sim \mathcal{N}_n(0, \tau^2 \boldsymbol{K}_n)$$

with $\boldsymbol{K}_n^{ij} = C_\theta(x_i, x_j) + g\delta_{ij}$. MVN conditionals yield the posterior predictive $Y(x)|\boldsymbol{D}_n$. **One can show that the entropy of that distribution, for $\boldsymbol{Y}_n$ observed at $\boldsymbol{X}_n$, is maximized when $|\boldsymbol{K}_n|$ is maximized**. For a complete derivation, see Section 6.2.1 of Santner, Williams, and Notz (2018). Recall that $\boldsymbol{K}_n$ depends on design $\boldsymbol{X}_n$, usually through inverse exponentiation pairwise squared distances.

One obstacle

- $\boldsymbol{K}_n$ and thus $|\boldsymbol{K}_n|$ also depend on hyperparameters, exemplified by $\boldsymbol{\theta}$ and g, so a maximum entropy design is hyperparameter dependent.
- This creates a chicken-or-egg problem because we hope to use data, which we've not yet observed since we're still designing the experiment, to learn hyperparameter settings.
- We'll see shortly that the choice of lengthscale $\boldsymbol{\theta}$ can have a substantial impact on maximum entropy designs. The role of the nugget g is more nuanced.
- For now assume $\boldsymbol{\theta}$ and g are known.

How to Code it?

Our stochastic exchange *mymaximin* implementation (4.2.1) may easily be altered to optimize $|\boldsymbol{K}_n|$. Note that the code uses $\log |\boldsymbol{K}_n|$ for greater numerical stability, assumes a separable Gaussian family kernel, but allows the user to tweak default settings of its hyperparameterization.

```
library(plgp)
maxent <- function(n, m, theta=0.1, g=0.01, T=100000) {
  if(length(theta) == 1) theta <- rep(theta, m)
  X <- matrix(runif(n*m), ncol=m)
  K <- covar.sep(X, d=theta, g=g)
  ldetK <- determinant(K, logarithm=TRUE)$modulus

  for(t in 1:T) {
    row <- sample(1:n, 1)
    xold <- X[row,]
    X[row,] <- runif(m)
    Kprime <- covar.sep(X, d=theta, g=g)
    ldetKprime <- determinant(Kprime, logarithm=TRUE)$modulus
    if(ldetKprime > ldetK) { ldetK <- ldetKprime
    } else { X[row,] <- xold }
  }
  return(X)
}
```

Some Observations

- As with *mymaximin* this is a simple, yet inefficient implementation from a computational perspective. Many proposed swaps will be rejected either because the outgoing point (xold) isn't that bad, or because the incoming runif(m) location is chosen too clumsily.
- Rules of thumb about how to make improvements here are harder to come by, however. One reason is that the effect of hyperparameterization is more difficult to intuit.
- Heuristics which favor swapping out points with small Euclidean distances can indeed reduce rejections.
- Derivatives can be helpful for local refinement. A homework exercise (6.4) asks the curious reader to entertain such enhancements.
- In spite of its inefficiencies, *maxent* as coded above works well in the illustrative examples below.

Simulate 25 locations in 2-d

```
X <- maxent(25, 2)
plot(X, xlab="x1", ylab="x2")
```

Similarities and differences to *maximin*

- One reason for the high degree of similarity between *maxent* and *maximin* designs is that the default *maxent* hyperparameterization is isotropic (i.e., radially symmetric), using the same θ_k-values in each input direction, $k \in 1, 2$. So both are using the same Euclidean distances under the hood.
- On the rare occasion where we prefer a Franken-kernel (5.3.3), rather than the friendly Gaussian, we could end up with a very interesting looking maximum entropy design.
- In a more common separable (Gaussian) setup, one might speculate desire for more spread in some directions rather than others through θ_k. Whether or not that's a good idea depends upon confidence in whatever evidence led to preferring differing lengthscales before collecting any data.

Simulate 25 locations in 2-d: anisotropic case

```
X <- maxent(25, 2, theta=c(0.1, 0.5))
plot(X, xlab="x1", ylab="x2")
```

Computational Complexity

- In general, model-based optimality comes at potentially substantial computational cost. Each iteration of *maxent* requires $O(n^3)$ cost. Maximin, by contrast, is only $O(n^2)$ via pairwise distances.
- Maximin can be further improved to $O(n)$. A similar trick can be performed with the log determinant to get an $O(n^2)$ implementation when proposing a change in just one coordinate, leveraging the following decomposition.

$$\log|\boldsymbol{K}_n| = log|\boldsymbol{K}_{n-1}| + \log(1 + g - C_\theta(\boldsymbol{x}_n, \boldsymbol{X}_{n-1})\boldsymbol{K}_{n-1}^{-1}C_\theta(\boldsymbol{X}_{n-1}, \boldsymbol{x}_n)) \quad (1)$$

- The origin of this result - see Eq. (6.10) – will be discussed in more detail when we get to sequential updating of GP models later in 6.3.

3d application

Before moving on, consider application in 3d.

```
X <- maxent(25, 3)

Is <- as.list(as.data.frame(combn(ncol(X),2)))
par(mfrow=c(1,length(Is)))
for(i in Is) {
  plot(X[,i], xlim=c(0,1), ylim=c(0,1), type="n",
    xlab=paste0("x", i[1]), ylab=paste0("x", i[2]))
  text(X[,i], labels=1:nrow(X))
}
```

Criteria based on predictive uncertainty

As the basis of another criterion, consider predictive uncertainty. At a particular location $\boldsymbol{x} \in \mathcal{X}$, the predictive variance is:

$$\sigma_n^2(\boldsymbol{x}) = \tau^2[1 + g - \boldsymbol{k}_n^T(\boldsymbol{x})\boldsymbol{K}_n^{-1}\boldsymbol{k}_n(\boldsymbol{x})]$$

where $k_n(\boldsymbol{x}) = k(\boldsymbol{X}_n, \boldsymbol{x})$. This is the same as what some call mean-squared prediction error (MSPE):

$$MSPE[\hat{y}(\boldsymbol{x})] = \mathbb{E}\{(\hat{y} - \boldsymbol{Y}(\boldsymbol{x}))^2\} = \sigma_n^2(\boldsymbol{x}).$$

An integrated MSPE (IMSPE) criterion is defined as MSPE (divided by τ^2), averaged over the entire input region $\mathcal{X}$, which is expressed below as a function of design $\boldsymbol{X}_n$.

$$J(\boldsymbol{X}_n) = \int_{\mathcal{X}} \frac{\sigma_n^2(\boldsymbol{x})}{\tau^2} w(\boldsymbol{x}) d\boldsymbol{x}$$

How to compute the integral?

- That's an m-dimensional integral for m-dimensional $\mathcal{X}$.
- Fortunately if $\mathcal{X}$ is rectangular, and where covariance kernels take on familiar forms such as isotropic or separable Gaussian and Matern $\nu = \{3/2, 5/2, \ldots\}$, closed forms are analytically tractable, or at least nearly so (e.g., depending on fast/accurate numerical evaluations of an error function/standard Gaussian distribution function, say.)
- Details are left to Eq. (10.9) and 10.3.1, on heteroskedastic (i.e., inputp-dependent noise) GPs, where the substance of such developments is of greater value to the narrative. Here it represents too much of a digression.

Implementation

We will, however, borrow the implementation in the supporting *hetGP* package (Binois and Gramacy 2019) on CRAN. The relevant function is called IMSPE. Since heteroskedastic GPs involve a few more bells and whistles, the function below strips down IMSPE's interface somewhat so that the setup better matches our ordinary GP setting.

```
library(hetGP)
imspe.criteria <- function(X, theta, g, ...)
 {
  IMSPE(X, theta=theta, Lambda=diag(g, nrow(X)), covtype="Gaussian",
    mult=rep(1, nrow(X)), nu=1)
 }
```

Implementation

Now we're ready to use IMSPE in a design search. R code below is ported from *maxent* (6.1.1) with modifications to minimize rather than maximize.

```
imspe <- function(n, m, theta=0.1, g=0.01, T=100000, ...)
 {
  if(length(theta) == 1) theta <- rep(theta, m)
  X <- matrix(runif(n*m), ncol=m)
  I <- imspe.criteria(X, theta, g, ...)

  for(t in 1:T) {
    row <- sample(1:n, 1)
    xold <- X[row,]
    X[row,] <- runif(m)
    Iprime <- imspe.criteria(X, theta, g, ...)
    if(Iprime < I) { I <- Iprime
    } else { X[row,] <- xold }
  }
  return(X)
}
```

2d illustration

Let's illustrate in 2d. The code below provides the search

```
X <- imspe(25, 2)

plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
```

Differences to maxent or maximin

- You can see that the resulting design is quite similar to maxent or maximin analogs, but there's one important distinction: **chosen sites avoid the boundary of the input space**.
- There's an intuitive explanation for this. IMSPE considers variance integrated over the entire input space (6.4).
- Design sites at the boundary don't "cover" that space as efficiently as interior ones.
- Points on a boundary in 2d cover half as much of the input space as (deep) interior ones do.

Differences to maxent or maximin

Points in the corner of a 2d space, i.e., at the intersection of two boundaries, cover one quarter of the space compared to ones in the interior. Thus boundary locations are far less likely to be chosen by IMSPE compared to *maxent*.
This effect is even more pronounced in higher input dimension. Consider 3d.

```
X <- imspe(25, 3)
Is <- as.list(as.data.frame(combn(ncol(X),2)))
par(mfrow=c(1,length(Is)))
for(i in Is) {
  plot(X[,i], xlim=c(0,1), ylim=c(0,1), type="n",
    xlab=paste0("x", i[1]), ylab=paste0("x", i[2]))
  text(X[,i], labels=1:nrow(X))
}
```

Differences to maxent or maximin

- Compared to the 2d version, selected sites are even more "off the boundary", but otherwise positioning behavior is quite similar to maxent or maximin.
- Many practitioners find these designs to be more aesthetically pleasing than the maxent analog.
- Points off of the boundary make sense, and the criteria itself is easier to intuit. Desire for a design which predicts equally well everywhere is easy to justify to non-experts.
- One downside to IMSPE, however, is that when good predictions are desired in non-rectangular regions of the input space, or where that space is not weighted equally.

Approximation of the Integral

- A simple approximation offers far greater flexibility, but implies a sometimes limiting accuracy-versus-computation trade-off.
- The idea is to replace the integral in Eq. above with a (possibly weighted) sum over a reference grid in $\mathcal{X}$, implementing a poor-man's quadrature.
- Reference grids need not be regular; or they could follow a space-filling construction.
- Many variations supported by this approximation amount to changes in the form or nature of reference grids. Grid density is intimately linked to approximation accuracy.

Concretely, the idea is:

$$J(\boldsymbol{X}_n) = \int_{\mathcal{X}} \frac{\sigma_n^2(\boldsymbol{x})}{\tau^2} w(\boldsymbol{x}) d\boldsymbol{x} \approx \frac{1}{T} \sum_{t=1}^{T} \frac{\sigma_n^2(\boldsymbol{x}_t)}{\tau^2} w(\boldsymbol{x}_t)$$

where $\boldsymbol{x}_t \sim Unif(\boldsymbol{X})$, or

$$\frac{1}{T} \sum_{t=1}^{T} \frac{\sigma_n^2(\boldsymbol{x}_t)}{\tau^2}$$

where $\boldsymbol{x}_t \sim \mathcal{W}(\boldsymbol{X})$ and $\mathcal{W}(\boldsymbol{X})$ is the measure of $w(\boldsymbol{x})$ applied in the input domain $\mathcal{X}$.

IMSPE approximation

To implement and illustrate this scheme, the subroutine below calculates GP predictive variance at reference locations *Xref* for design $X = \boldsymbol{X}_n$ and then approximates the integral (6.4) by a mean over *Xref*. The function below re-defines our *imspe.criteria* from above, and ellipses (...) allows re-use of the *imspe* searching function above under the new approximate criterion.

```
imspe.criteria <- function(X, theta, g, Xref)
 {
  K <- covar.sep(X, d=theta, g=g)
  Ki <- solve(K)
  KXref <- covar.sep(X, Xref, d=theta, g=0)
  return(mean(1 + g - diag(t(KXref) %*% Ki %*% KXref)))
 }
```

IMSPE approximation in 2D

```
g <- expand.grid(seq(0,1,length=10), seq(0,1, length=10))
X <- imspe(25, 2, Xref=g)

###################################


plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
points(g, pch=20, cex=0.25, col="gray")
```

IMSPE in a non-rectangular shape

```
Xref <- rmvnorm(100, mean=c(0.25, 0.25),
  sigma=0.005*rbind(c(2, 0.35), c(0.35, 0.1)))
Xref <- rbind(Xref,
  rmvnorm(100, mean=c(0.25, 0.25),
    sigma=0.005*rbind(c(0.1, -0.35), c(-0.35, 2))))
X <- imspe(25, 2, Xref=Xref)

plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
points(Xref, pch=20, cex=0.25, col="gray")
```

Sequential design/active learning

Figure 29 summarizes a sequential design setup. Although there are variations, here I shall emphasize the simple setup of augmenting an initial design by one, repeated until a desired size is reached.

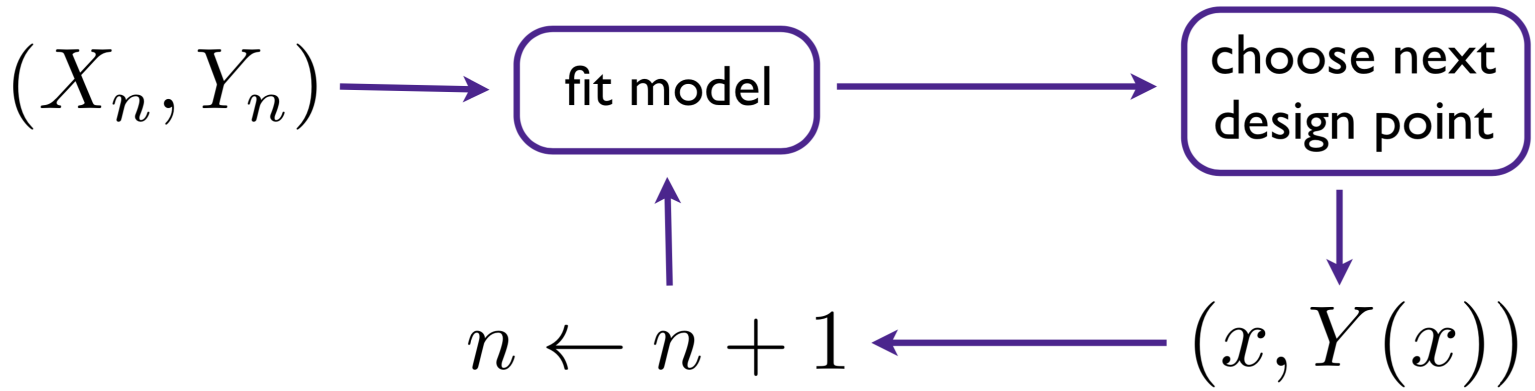

Figure: Diagram of sequential design/active learning/design augmentation.

Sequential design/active learning

To supplement that cartoon, consider Algorithm 6.1. The development here is a little backwards, saying what the algorithm is before delving into its key components, in particular Step 3 where a criterion is optimized to choose the next design element. Figure 6.8 is suggestive of indefinite updating, augmenting a design by one in each pass through the circuit. Algorithm 6.1 assumes a fixed total design size N, iterating in $n = n_0, \ldots, N$, starting from a small seed design of size n_0.

Sequential design/active learning

Assume a flexible surrogate, e.g., a GP model, but with potentially unknown hyperparameterization.
Require a function $f(\cdot)$ providing outputs $y = f(\boldsymbol{x})$ for inputs $\boldsymbol{x}$, either deterministic or observed with noise; a choice of initial design size n_0 and final size N; and criterion $J(\boldsymbol{x})$ to search for design augmentation.
Then: Run a small seed, or bootstrapping experiment.

(a) Create an initial seed design X_{n_0} with n_0 runs. Typically X_{n_0} is a model-free choice, e.g., derived from a static LHS or maximin design.

(b) Evaluate $y_i = f(\boldsymbol{x}_i)$ under each x_i^T in the i^{th} row of X_{n_0}, for $i = 1, \ldots, n_0$, obtaining $D_{n_0} = (X_{n_0}, Y_{n_0})$.

(c) Set $n \longleftarrow n_0$, indexing iterations of sequential design.

Sequential design/active learning

1. Fit the surrogate (and hyperparameters) using $\boldsymbol{D}_n$, e.g., via MLE.
2. Solve criterion $J(\boldsymbol{x})$ based on the fitted model from Step 2, resulting in a choice of $\boldsymbol{x}_{n+1}|\boldsymbol{D}_n$ via $\boldsymbol{x}_{n+1} = argmax_{\boldsymbol{x}\in X} J(\boldsymbol{x})|\boldsymbol{D}_n$.
3. Observe the response at the chosen location by running a new simulation, $y_{n+1} = f(\boldsymbol{x}_{n+1})$.
4. Update $\boldsymbol{D}_{n+1} = \boldsymbol{D}_n(\boldsymbol{x}_{n+1}, y_{n+1})$; set $n \longleftarrow n+1$ and repeat from Step 1 unless $n = N$.

Return the chosen design and function evaluations $\boldsymbol{D}_N$, along with surrogate fit (i.e., after a final application of Step 2).

Some observations/ Efficient computations

- Numerical optimization of the likelihood to infer hyperparameters can require initial values.
- Random initialization is common when n is small, e.g., $n = n_0$. Initializing with hyperparameters found in earlier iterations of sequential design – a warm start – offers computationally favorable results (faster convergence) and negligible deleterious effects when n is large.
- Earlier on, periodic random reinitialization can confer a certain robustness in the face of multimodal likelihoods (5.3.4), which are not an uncommon occurrence.
- Skipping expensive $\mathcal{O}(n^3)$ calculations in Step 1, e.g., using MLE calculations from earlier n, is risky but could represent a huge computational savings: taking flops in $\mathcal{O}(n^3)$ down to $\mathcal{O}(n^2)$.

Active Learning MacKay (ALM)

- The simplest sequential design scheme for GPs, but it's of course more widely applicable, involves choosing the next point to maximize predictive variance.
- Given data $\boldsymbol{D}_n = (\boldsymbol{X}_n, \boldsymbol{Y}_n)$, infer unknown hyperparameters (τ^2, θ, g) by maximizing the likelihood, say, and choose $\boldsymbol{x}_{n+1}$ as

$$\boldsymbol{x}_{n+1} = \arg\max_{\boldsymbol{x} \in \mathcal{X}} \sigma_n^2(\boldsymbol{x}).$$

- Obtain $\boldsymbol{y}_{n+1} = Y(\boldsymbol{x}_{n+1}) = f(\boldsymbol{x}_{n+1}) + \epsilon$; combine to form the new dataset: $\boldsymbol{D}_{n+1} = ([\boldsymbol{X}_n; \boldsymbol{x}_{n+1}], [\boldsymbol{Y}_n; y_{n+1}])$; repeat.
- In other words, the criterion provided to Algorithm 6.1 is $J(x)|\boldsymbol{D}_n \equiv \sigma_n^2(\boldsymbol{x})$, where $\sigma_n^2(\boldsymbol{x})$ comes from kriging equations (5.3 book) applied using $(\hat{\tau}_n^2, \hat{\theta}_n^2, \hat{g}_n)$ trained on data compiled from earlier sequential design iterations: $\boldsymbol{D}_n = (\boldsymbol{X}_n, \boldsymbol{Y}_n)$.
- The n in the subscript is included to remind readers that

ALM in practice

To demonstrate ALM, let's revisit our favorite 2d dataset from 5.1.2, seeding with a small LHS of size $n_0 = 12$. The code chunk below implements Step 1 of Algorithm 6.1 with this choice of $f(\cdot)$.

```
library(lhs)
ninit <- 12
X <- randomLHS(ninit, 2)
f <- function(X, sd=0.01)
 {
  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=sd)
 }
y <- f(X)
```

ALM in practice

Step 2 involves fitting a model. Below, an isotropic GP is fit using laGP library routines.

```
library(laGP)
g <- garg(list(mle=TRUE, max=1), y)
d <- darg(list(mle=TRUE, max=0.25), X)
gpi <- newGP(X, y, d=d$start, g=g$start, dK=TRUE)
mle <- jmleGP(gpi, c(d$min, d$max), c(g$min, g$max), d$ab, g$ab)
```

```
x1 <- x2 <- seq(0, 1, length=100)
XX <- expand.grid(x1, x2)
yytrue <- f(XX, sd=0)

rmse <- sqrt(mean((yytrue - predGP(gpi, XX, lite=TRUE)$mean)^2))

obj.alm <- function(x, gpi)
  - sqrt(predGP(gpi, matrix(x, nrow=1), lite=TRUE)$s2)
```

Problem with Local Minima

- Predictive variance $\sigma_n^2(\boldsymbol{x})$ produces football/sausage-shaped error-bars, so it must have many local maxima. In fact, it's easy to see how the number of local maxima could grow linearly in n.
- Optimizing globally over that surface presents challenges. Global optimization is a hard sequential design problem in its own right, hence Chapter 7.
- Here we shall stick to our favorite library-based local solver, *optim* with method="L-BFGS-B" (Byrd et al. 1995). Code below establishes the ALM objective as minus predictive variance, with the intention of passing to *optim* whose default is to search for *minima*.

ALM search

```
xnp1.search <- function(X, gpi, obj=obj.alm, ...)
 {
  start <- mymaximin(nrow(X), 2, T=100*nrow(X), Xorig=X)
  xnew <- matrix(NA, nrow=nrow(start), ncol=ncol(X) + 1)
  for(i in 1:nrow(start)) {
    out <- optim(start[i,], obj, method="L-BFGS-B", lower=0,
      upper=1, gpi=gpi, ...)
    xnew[i,] <- c(out$par, -out$value)
  }
  solns <- data.frame(cbind(start, xnew))
  names(solns) <- c("s1", "s2", "x1", "x2", "val")
  return(solns)
 }
```

Talking about the output

- On output, a data.frame is returned combining starting and ending locations with a final column recording the value of the objective found by *optim*.
- Such comprehensive output is probably overkill for most situations, because all that matters is $c(x_1, x_2)$ coordinates in the row having the smallest val.
- Other rows are furnished primarily to aid in illustration. For example, Figure below draws arrows connecting starting $c(s_1, s_2)$ coordinates to locally optimal variance-maximizing solutions, with a red dot at the terminus of the best val arrow. (Any arrows with near-zero length are omitted, and occasionally such an arrow "terminates" at the red dot.)

ALM search 2D

```
solns <- xnp1.search(X, gpi)
plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
arrows(solns$s1, solns$s2, solns$x1, solns$x2, length=0.1)
m <- which.max(solns$val)
prog <- solns$val[m]
points(solns$x1[m], solns$x2[m], col=2, pch=20)
```

A different approach: ALC

- You could say that ALM has somewhat of a scale problem.
- Just because predictive variance is high doesn't mean that there's value in adding more data.
- One could instead work with epistemic variance (5.16), i.e., the variance of the latent field (5.3.2), or variance of the mean.
- Operationally speaking, this amounts to using a zero nugget when calculating out-of-sample covariances among $\mathcal{X} = XX$.
- A better metric might: how helpful is a potential new design site at reducing predictive uncertainty?
- If a lot, relative to previous reductions and relative to other new design sites, then it could be quite helpful to perform a new run at that location. If not, perhaps another location would be preferred or perhaps we have enough data already.

ALC idea

- But this begs the question where; where should the reduction be measured? One option is everywhere, integrating over the entire input space like IMSPE does for global design. Such an integral is tractable analytically for rectangular input spaces, but I shall defer that discussion to Chapter 10.
- At the other extreme is measuring reduction in variance at a particular reference location, or at a collection of locations, thereby approximating the integral with a sum like we did with IMSPE (6.5).
- The first person to suggest such an acquisition heuristic in a nonparametric regression context was Cohn (1994), for neural networks. As with ALM, Seo et al. (2000) adapted Cohn's idea to GPs and called it active learning Cohn (ALC).
- The result is essentially a sequential analog of IMSPE design, and in fact the sequential version can be shown to approximate a full A-optimal design.

Criteria using IMSPE

Recall that predictive variance follows

$$\sigma_n^2(\boldsymbol{x}) = \hat{\tau}_n^2[1 + \hat{g}_n - \boldsymbol{k}_n^T(\boldsymbol{x})\boldsymbol{K}_n^{-1}\boldsymbol{k}_n(\boldsymbol{x})]$$

where $k_n(\boldsymbol{x}) = C_{\hat{\theta}}(\boldsymbol{X}_n, \boldsymbol{x})$. Written here with an n subscript on all estimated quantities in order to emphasize their dependence on data $\boldsymbol{D}_n = (\boldsymbol{X}_n, \boldsymbol{Y}_n)$ via $\hat{\tau}_n^2$, $\hat{g}_n$, and $\hat{\theta}_n$ hidden inside $\boldsymbol{K}_n$. Let $\tilde{\sigma}_{n+1}^2(\boldsymbol{x})$ denote the deduced variance based on design $\boldsymbol{X}_{n+1}$ combining $\boldsymbol{X}_n$ and a new input location $\boldsymbol{x}_{n+1}$ residing in its $n+1$st row. Otherwise, $\tilde{\sigma}_{n+1}^2(\boldsymbol{x})$ conditions on the same estimated quantities as $\sigma_n^2(\boldsymbol{x})$, i.e., $\hat{\tau}_n^2$, $\hat{g}_n$, and $\hat{\theta}_n$, all estimated from $\boldsymbol{Y}_n$. Since $\boldsymbol{Y}_n$ doesn't directly appear in $\sigma_n^2(\boldsymbol{x})$, nor would it directly appear in $\tilde{\sigma}_{n+1}^2(\boldsymbol{x})$. Therefore, quite simply

$$\tilde{\sigma}_{n+1}^2(\boldsymbol{x}) = \hat{\tau}_{n+1}^2[1 + \hat{g}_{n+1} - \boldsymbol{k}_{n+1}^T(\boldsymbol{x})\boldsymbol{K}_{n+1}^{-1}\boldsymbol{k}_{n+1}(\boldsymbol{x})]$$

where $k_{n+1}(\boldsymbol{x}) = C_{\hat{\theta}}(\boldsymbol{X}_n, \boldsymbol{x})$.

ALC

Using that definition, the ALC criteria is the average (integrated over X or any other subset of the input space) reduction in variance from n→n+1 measured through a choice of $\boldsymbol{x}_{n+1}$, augmenting the design:

$$\Delta\sigma^2_{n+1}(\boldsymbol{x}_{n+1}) = \int_{\boldsymbol{x}} \sigma^2_n(\boldsymbol{x}) - \tilde{\sigma}^2_{n+1}(\boldsymbol{x})d\boldsymbol{x} = c - \int_{\boldsymbol{x}} \tilde{\sigma}^2_{n+1}(\boldsymbol{x})d\boldsymbol{x}.$$

Wishing predictive uncertainty to be reduced as much as possible, that translates into finding an $\boldsymbol{x}_{n+1}$ maximizing $\Delta\sigma^2_{n+1}(\boldsymbol{x}_{n+1})$. But that's the same as minimizing the integrated deduced variance. Therefore the criterion that must be solved in each iteration of sequential design, occupying Step 3 of Algorithm 6.1, is

$$\boldsymbol{x}_{n+1} = \arg\min_{\boldsymbol{x}\in\mathcal{X}} \int_{\boldsymbol{x}\in\mathcal{X}} \tilde{\sigma}^2_{n+1}(\boldsymbol{x})d\boldsymbol{x}$$

A closed form for rectangular $\mathcal{X}$, complete with derivatives for maximizing, is provided later in Chapter 10.

ALC search 2D

```
obj.alc <- function(x, gpi, Xref)
  - sqrt(alcGP(gpi, matrix(x, nrow=1), Xref))

####
deleteGP(gpi)
X <- X[1:ninit,]
y <- y[1:ninit]
g <- garg(list(mle=TRUE, max=1), y)
d <- darg(list(mle=TRUE, max=0.25), X)
gpi <- newGP(X, y, d=d$start, g=g$start, dK=TRUE)
mle <- jmleGP(gpi, c(d$min, d$max), c(g$min, g$max), d$ab, g$ab)
p <- predGP(gpi, XX, lite=TRUE)
rmse.alc <- sqrt(mean((yytrue - p$mean)^2))
```

ALC search 2D Cont.

```
 Xref <- randomLHS(100, 2)
solns <- xnp1.search(X, gpi, obj=obj.alc, Xref=Xref)
m <- which.max(solns$val)
xnew <- as.matrix(solns[m, 3:4])
prog.alc <- solns$val[m]

plot(X, xlab="x1", ylab="x2", xlim=c(0,1), ylim=c(0,1))
arrows(solns$s1, solns$s2, solns$x1, solns$x2, length=0.1)
points(solns$x1[m], solns$x2[m], col=2, pch=20)
points(Xref, cex=0.25, pch=20, col="gray")
```