3.6 Adaptive Sampling

In this section, sequential design of experiments, a.k.a. *adaptive sampling*, is demonstrated on the exponential data of Section 3.3. Gathering, again, the data:

```
> exp2d.data <- exp2d.rand(lh = 0, dopt = 10)
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> Xcand <- lhs(1000, rbind(c(-2, 6), c(-2, 6)))
```

In contrast with the data from Section 3.3, which was based on a grid, the above code generates a randomly subsampled D–optimal design $\mathbf{X}$ from LH candidates, and random responses $\mathbf{Z}$. As before, design configurations are more densely packed in the interesting region. Candidates $\tilde{\mathbf{X}}$ are from a large LH–sample.

Given some data $\{\mathbf{X}, \mathbf{Z}\}$, the first step in sequential design using `tgp` is to fit a treed GP LLM model to the data, without prediction, in order to infer the MAP tree $\hat{\mathcal{T}}$.

```
> exp1 <- btgpllm(X = X, Z = Z, pred.n = FALSE, corr = "exp",
+     verb = 0)
```

The trees are shown in Figure 15. Then, use the `tgp.design` function to create D–optimal candidate designs in each region of $\hat{\mathcal{T}}$. For the purposes of illustrating the `improv` statistic, I have manually added the known (from calculus) global minimum to `XX`.

```
> XX <- tgp.design(200, Xcand, exp1)

sequential treed D-Optimal design in 3 partitions
dopt.gp (1) choosing 55 new inputs from 272 candidates
dopt.gp (2) choosing 53 new inputs from 263 candidates
dopt.gp (3) choosing 93 new inputs from 465 candidates

> XX <- rbind(XX, c(-sqrt(1/2), 0))
```

Figure 16 shows the sampled `XX` locations (circles) amongst the input locations `X` (dots) and MAP partition ($\hat{\mathcal{T}}$). Notice how the candidates `XX` are spaced out relative to themselves, and relative to the inputs `X`, unless they are near partition boundaries. The placing of configurations near region boundaries is a symptom particular to D–optimal designs. This is desirable for experiments with `tgp` models, as model uncertainty is usually high there [3].

Now, the idea is to fit the treed GP LLM model, again, in order to assess uncertainty in the predictive surface at those new candidate design points. The following code gathers all three adaptive sampling statistics: ALM, ALC, & EI.

```
> exp.as <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     improv = TRUE, Ds2x = TRUE, verb = 0)
```

```
> tgp.trees(exp1)

NOTICE: skipped plotting tree of height 1, with lpost = 120.941
```

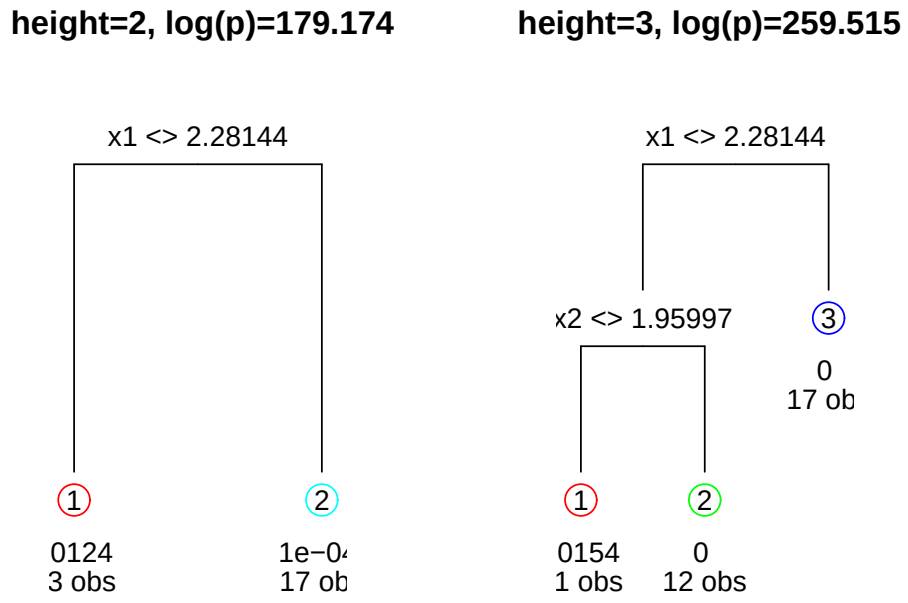

Figure 15: MAP trees of each height encountered in the Markov chain for the exponential data, showing $\hat{\sigma}^2$ and the number of observations n at the leaves. $\hat{\mathcal{T}}$ is the one with the maximum $\log(p)$ above.

Figure 17 shows the posterior predictive estimates of the adaptive sampling statistics. The error surface, on the *left*, summarizes posterior predictive uncertainty by a norm of quantiles.

In accordance with the ALM algorithm, candidate locations `XX` with largest predictive error would be sampled (added into the design) next. These are most likely to be in the interesting region, i.e., the first quadrant. However, these results depend heavily on the clumping of the original design in the uninteresting areas, and on the estimate of $\hat{\mathcal{T}}$. Adaptive sampling via the ALC, or EI (or both) algorithms proceeds similarly, following the surfaces shown in *center* and *right* panels of Figure 17.

A Implementation notes

The treed GP model is coded in a mixture of `C` and `C++`: `C++` for the tree data structure ($\mathcal{T}$) and `C` for the GP at each leaf of $\mathcal{T}$. The code has been tested on Unix (`Solaris, Linux, FreeBSD, OSX`) and Windows (2000, XP) platforms.

It is useful to first translate and re-scale the input data ($\mathbf{X}$) so that it lies in

```
> plot(exp1$X, pch = 19, cex = 0.5)
> points(XX)
> mapT(exp1, add = TRUE)
```

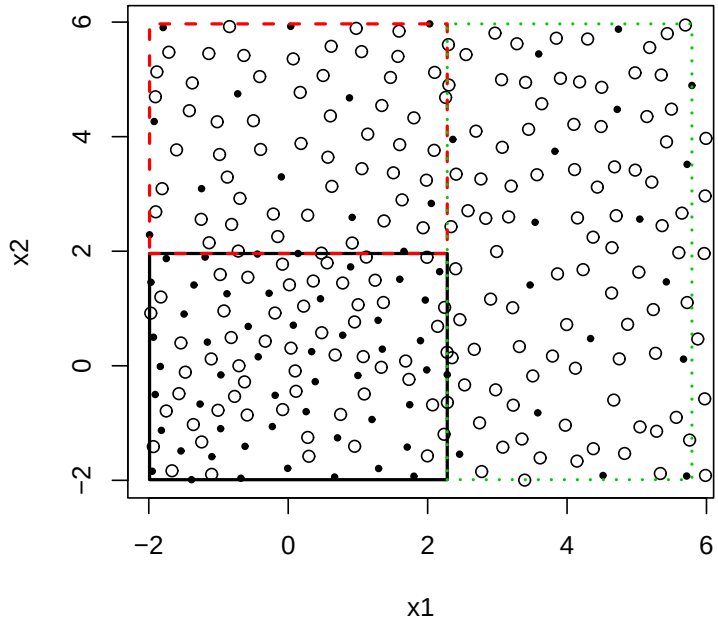

Figure 16: Treed D–optimal candidate locations `XX` (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$

```
> par(mfrow = c(1, 3), bty = "n")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "alm")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "alc")
> plot(exp.as, main = "tgpllm,", layout = "as", as = "improv")
```

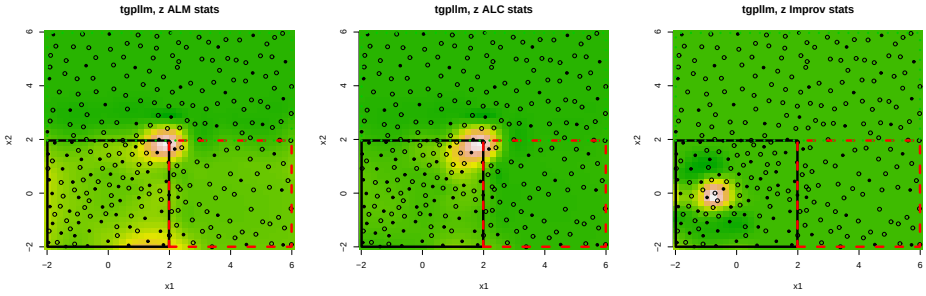

Figure 17: *Left*: Image plots of adaptive sampling statistics and MAP trees $\hat{\mathcal{T}}$; *Left*; ALM adaptive sampling image for (only) candidate locations `XX` (circles); *center*: ALC; and *right:* EI.

an $\Re^{m_X}$ dimensional unit cube. This makes it easier to construct prior distributions for the width parameters to the correlation function $K(\cdot,\cdot)$. Proposals for all parameters which require MH sampling are taken from a uniform "sliding window" centered around the location of the last accepted setting. For example, a proposed a new nugget parameter g_ν to the correlation function $K(\cdot,\cdot)$ in region r_ν would go as

$$g_\nu^* \sim \text{Unif}\left(\frac{3}{4}g_\nu, \frac{4}{3}g_\nu\right).$$

Calculating the corresponding forward and backwards proposal probabilities for the MH acceptance ratio is straightforward.

B Interfaces and features

The following subsections describe some of the ancillary features of the `tgp` package such as the gathering and summarizing of MCMC parameter traces, the progress meter, and an example of how to use the `predict.tgp` function in a collaborative setting.

B.1 Parameter traces

Traces of (almost) all parameters to the `tgp` model can be collected by supplying `trace=TRUE` to the `b*` functions. In the current version traces for the linear prior correlation matrix ($\mathbf{W}$) are not provided. I shall illustrate the gathering and analyzing of traces through example. But first, a few notes and cautions.

Models which involve treed partitioning may have more than one base model (GP or LM). The process governing a particular input `x` depends on the coordinates of `x`. As such, `tgp` records region–specific traces of parameters to GP (and linear) models at the locations enumerated in the `XX` argument. Even traces of single–parameter Markov chains can require hefty amounts of storage, so recording traces at each of the `XX` locations can be an enormous memory hog. A related warning will be given if the product of |`XX`|, `(BTE[2]-BTE[1])/BTE[3]` and `R` is beyond a threshold. The easiest way to keep the storage requirements for traces down is to control the size of `XX` and the thinning level `BTE[3]`. Finally, traces for most of the parameters are stored in output files. The contents of the trace files are read into `R` and stored as `data.frame` objects, and the files are removed. The existence of partially written trace files in the current working directory (CWD)—while the `C` code is executing—means that not more than one `tgp` run (with `trace = TRUE`) should be active in the CWD at one time.

Consider again the exponential data. For illustrative purposes I chose `XX` locations (where traces are gathered) to be (1) in the interior of the interesting region, (2) on/near the plausible intersection of partition boundaries, and (3) in the interior of the flat region. The hierarchical prior `bprior = "b0"` is used to leverage a (prior) belief the most of the input domain is uninteresting.

```
> exp2d.data <- exp2d.rand(n2 = 150, lh = 0, dopt = 10)
> X <- exp2d.data$X
```