use for comparison is root mean-square error (RMSE)

$$\text{MSE} = \sum_{i=1}^{n} (\mu_i - \hat{z}_i)^2/n \qquad\qquad \text{RMSE} = \sqrt{\text{MSE}}$$

where $\hat{z}_i$ is the model-predicted response for input $\mathbf{x}_i$. The $\mathbf{x}$'s are randomly distributed on the unit interval.

Input data, responses, and predictive locations of size $N = 200$ and $N' = 1000$, respectively, can be obtained by a function included in the `tgp` package.

```
> f <- friedman.1.data(200)
> ff <- friedman.1.data(1000)
> X <- f[, 1:10]
> Z <- f$Y
> XX <- ff[, 1:10]
```

This example compares Bayesian linear CART with Bayesian GP LLM (not treed), following the RMSE experiments of Chipman et al. It helps to scale the responses so that they have a mean of zero and a range of one. First, fit the Bayesian linear CART model, and obtain the RMSE.

```
> fr.btlm <- btlm(X = X, Z = Z, XX = XX, tree = c(0.95,
+     2, 10), m0r1 = TRUE)
> fr.btlm.mse <- sqrt(mean((fr.btlm$ZZ.mean - ff$Ytrue)^2))
> fr.btlm.mse
```

Next, fit the GP LLM, and obtain its RMSE.

```
> fr.bgpllm <- bgpllm(X = X, Z = Z, XX = XX, m0r1 = TRUE)
> fr.bgpllm.mse <- sqrt(mean((fr.bgpllm$ZZ.mean - ff$Ytrue)^2))
> fr.bgpllm.mse
```

So, the GP LLM is 3.868 times better than Bayesian linear CART on this data, in terms of RMSE (in terms of MSE the GP LLM is 1.967 times better). Watching the evolution of the Markov chain for the GP LLM (via the progress statements written to `stdout`, not shown because the not would fit on the page), it is easy to see how the GP LLM quickly learns that $\mathbf{b} = (1, 1, 1, 0, 0, 0, 0, 0, 0, 0)$, and that $\beta_4 \approx 4$ and $\beta_5 \approx 10$—basically that only the first three inputs contribute nonlinearly, the fourth and fifth contribute linearly, and the remaining five not at all [10].

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()
> X <- exp2d.data$X
> Z <- exp2d.data$Z
> Xcand <- exp2d.data$XX
```

Start by fitting 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")

> tgp.trees(exp1)

NOTICE: skipped plotting tree of height 1, with lpost = 143.976
```

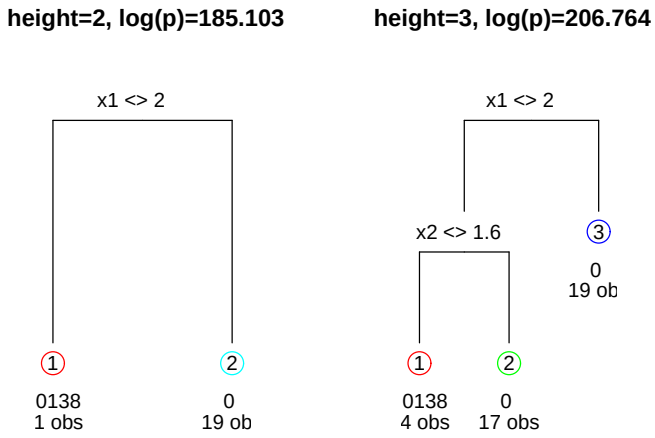

Figure 15: MAP trees of each height encountered in the Markov chain for the exponential data, showing $\hat{\sigma}^2$ and the number of observations n at the leaves. $\hat{\mathcal{T}}$ is the one with the maximum $\log(p)$ above.

The trees are shown in Figure 15. Then, use the `tgp.design` function to create D-optimal candidate designs in each region of $\hat{\mathcal{T}}$.

```
> XX <- tgp.design(10, Xcand, exp1)

sequential treed D-Optimal design in 3 partitions
dopt.gp (1) choosing 2 new inputs from 66 candidates
dopt.gp (2) choosing 3 new inputs from 104 candidates
dopt.gp (3) choosing 6 new inputs from 191 candidates
```

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 [2].

```
> plot(exp1$X, pch = 19, cex = 0.5)
> points(XX)
> tgp.plot.parts.2d(exp1$parts)
```

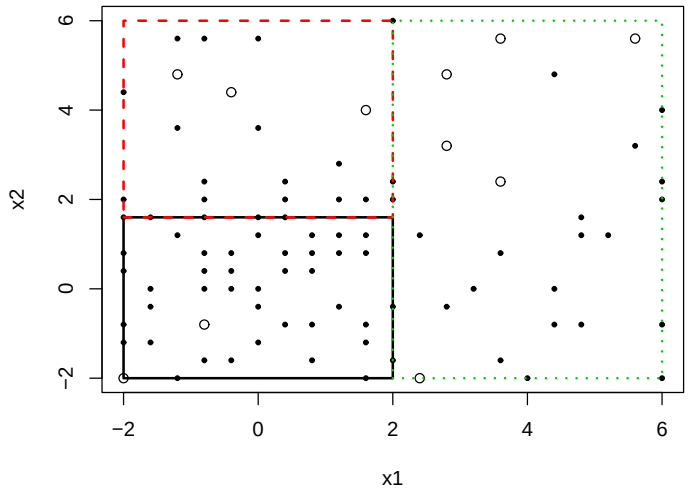

Figure 16: Treed D-optimal candidate locations `XX` (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$

Figure 16 uses the `tgp.plot.parts.2d` function. Unfortunately, this function is not well documented in the current version of the `tgp` package. This should change in future versions.

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. For illustrative purpose, the following code gathers all three adaptive sampling statistics: ALM, ALC, & EGO.

```
> exp1.btgpllm <- btgpllm(X = X, Z = Z, XX = XX, corr = "exp",
+     ego = TRUE, ds2x = TRUE)
```

Figure 17 shows the posterior predictive surface. The error surface, on the *right*, summarizes posterior predictive uncertainty by a norm of quantiles. Since the combined data and predictive locations are not densely packed in the input space, the `loess` smoother may have trouble with the interpolation. One option is increase the `tgp`-default kernel span supplied to `loess`, e.g., `span = 0.5`. Or, the `akima` method can be used instead.

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, due to the random nature of this `Sweave` document, this is not always the case. Results

```
> par(mfrow = c(1, 2), bty = "n")
> plot(exp1.btgpllm, main = "treed GP LLM,", method = "akima",
+     layout = "surf")
> plot(exp1.btgpllm, main = "treed GP LLM,", method = "akima",
+     layout = "as", as = "alm")
```

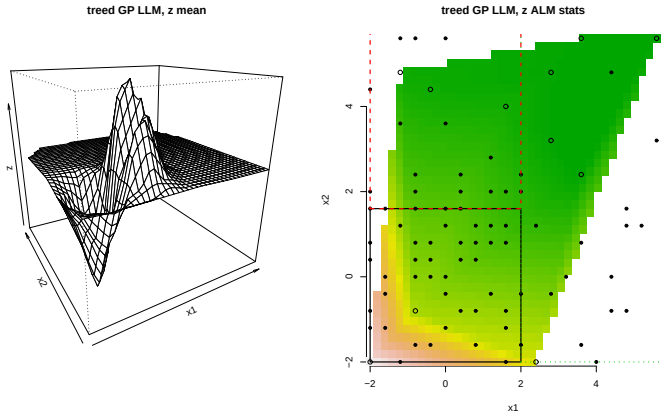

Figure 17: *Left*: Posterior mean surface; *right* ALM adaptive sampling image for (only) candidate locations `XX` (circles), MAP tree $\mathcal{T}$ and input locations `X` (dots). (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$

depend heavily on the clumping of the original design in the un-interesting areas, and on the estimate of $\hat{\mathcal{T}}$.

Adaptive sampling via the ALC, or EGO (or both) algorithms proceeds similarly, following the surfaces shown in Figure 18.

A Linking to ATLAS

`ATLAS` [23] is supported as an alternative to standard BLAS and LAPACK for fast, automatically tuned, linear algebra routines. Compared standard `BLAS/Lapack`, those automatically tuned by `ATLAS` are signigantly faster. If you know that `R` has already been linked to tuned linear algebra libraries (e.g., on `OSX`), then compiling with ATLAS as described below, is unncessary—just install as usual. As an alternative to linking `tgp` to ATLAS directly, one could re-compile all of `R` linking it to `ATLAS` following the documentation on the `R` website. While this is arguably best solution since all of `R` benefits, the task can prove challenging to accomplish and may require administrator (root) priviledges. Linking `tgp` with `ATLAS` directly is described here.

Three easy steps (assuming, of course, you have already compiled and installed `ATLAS` – `http://math-atlas.sourceforge.net`) need to be performed before you install the `tgp` package from source.

1. Edit src/Makevars. Comment out the existing `PKG_LIBS` line, and replace

```
> par(mfrow = c(1, 2), bty = "n")
> plot(exp1.btgpllm, main = "treed GP LLM,", method = "akima",
+     layout = "as", as = "alc")
> plot(exp1.btgpllm, main = "treed GP LLM,", method = "akima",
+     layout = "as", as = "ego")
```

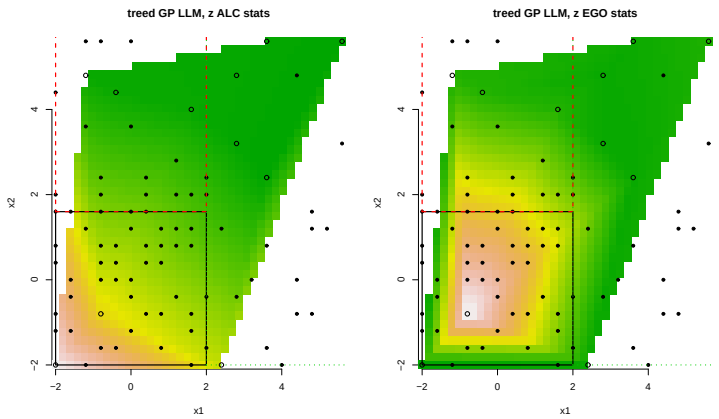

Figure 18: Adaptive sampling images for (only) candidate locations `XX` (circles), MAP tree $\mathcal{T}$ and input locations `X` (dots). (circles), input locations `X` (dots), and MAP tree $\hat{\mathcal{T}}$. *Left*: ALC; *right*: EGO.

it with:

```
PKG_LIBS = -L/path/to/ATLAS/lib -llapack -lcblas -latlas
```

You may need replace `"-llapack -lcblas -latlas"` with whatever ATLAS recommends for your OS. (See `ATLAS` README.) For example, if your ATLAS compilation included F77 support, you might need to add `"-lF77blas"`, if you compiled with pthreads, you would might use `"-llapack -lptcblas -lptf77blas -latlas"`.

2. Continue editing src/Makevars. Add:

```
PKG_CFLAGS = -I/path/to/ATLAS/include
```

3. Edit src/linalg.h and commend out lines 40 & 41:

```
/*#define FORTPACK
#define FORTBLAS*/
```

Now simply install the `tgp` package as usual. Reverse the above instructions to disable ATLAS. Don't forget to re-install the package when you're done.