Machine Learning for Optimal Stopping Problems with mlOSP

Mike Ludkovski

April 16, 2018

Contents

Simulation Based Optimal Stopping and Beyond

Numerical resolution of optimal stopping problems has been an active area of research for more than 2 decades. Originally investigated in the context of American Option pricing, it has since metamorphosed into a field unto itself, with numerous wide-ranging applications and dozens of proposed approaches.

A major strand, which is increasingly dominating the subject, are simulation-based methods that apply the Monte Carlo paradigm to Optimal Stopping. With its roots in 1990s, this framework remains without an agreed-upon name; we shall refer to it as Regression Monte Carlo (RMC). The main feature of RMC is its marriage of a probabilistic approach, namely direct reliance on the underlying stochastic state dynamics as part of the simulations, and statistical tools for approximating the quantities of interest, primarily the value and/or continuation functions. This combination of simulation and statistics brings scalability, flexibility in terms of underlying model assumptions and a vast arsenal of potential implementations. These benefits have translated into excellent performance which has made RMC popular both in the academic and practitioner (quantitative finance) communities. Indeed, in the opinion of the author, these developments can claim to be the most successful numerical strategy that emerged from Financial Mathematics. Dating back only about 20 years, they have now percolated down into standard Masters-level curriculum, and are increasingly utilized in cognate engineering and mathematical sciences domains.

Despite hundreds of journal publications addressing various variants of RMC, there remains a dearth of user-friendly benchmarks or unified overviews of the algorithms. In particular, to the author's knowledge, there is no R (or other free programming languages such as Python) package for RMC. One reason for this gap is the narrow focus of many research articles that tend to explore one small aspect of RMC and then illustrate their contributions on a small-scale, idiosyncratic example.

In the present article, I aim to offer an algorithmic template for RMC, coupled with its implementation within R. This endevour is meant to be an ongoing project, offering one way to centralize and standardize RMC approaches, as they are proposed. In particular, the associated **mlOSP** library is currently in its Version 1.0 incarnation, with further (hopefully regular) updates to be fully expected.

Contributions and Outline

This report offers a unified description of RMC via the underlying statistical concepts. Hence, we describe a *generic* RMC template that emphasizes the building blocks, rather than specific approaches. This perspective therefore aims to *nest* as many of existing works as possible, shedding light on the differences and the similarities between the numerous proposals. To do so, we utilize as much as possible the language of statistics (in contrast to the language of finance, or of probability). In particular, we attempt to place RMC in the context of modern machine/statistical learning, highlighting such aspects as Design of Experiments, Simulation Device, and Sequential Learning. Through this angle, we naturally connect RMC with numerous alternative tools, twists, and extensions. Thus, we believe already the template is an important contribution, giving a new take on some quite-old ideas.

Second, we propose several new variants based on the above templated RMC. These include: * Adaptive kernel regression using nnreg and XXX and svm * Varying the design size N across time-steps * hetGP/hetTP regression * implementation of swing option via ML methods * sfjlksdjfl

Third, we introduce a few benchmarks, i.e. fully specified problem instances, that would be useful for other researchers.

The article is structured as an extended vignette and consists of two parts. The first part lays out the RMC template which underlies the **mlOSP** library. The template emphasizes the three pieces of RMC framework: stochastic simulation, experimental design, and statistical approximation. These pieces are modularized and can be fully mixed-and-matched within the core backward dynamic programming loop driving the algorithms. Section XXX briefly summarizes the variants of each module that have been implemented, and includes references to the original articles where these were proposed. For example, we provide more than 10 different methods for the regression module, ranging from kernel regression to piecewise linear regression to Gaussian process regression.

The second part serves as a "User Guide" to mlOSP and illustrates how the template is implemented in the library. For ease of use, we provide about half-dozen top-level functions, whose usage is illustrated via code listings and the respective R output. The latter is organized as a RMarkdown document, enabling full reproduction by any reader who downlaods the **mlOSP** package. Simultaneously with illustrating *how* to use **mlOSP**, we also define several instances of OSP, including examples of Bermudan options in 1-, 2-, 3-, and 5-dimensions. By providing fully reproducible results on these instances, we hope to create a preliminary set of simple benchmarks that allow for a transparent, apples-to-apples comparison of different methods. As we discuss below, the numerous nuances that inevitably crop up when implementing RMC, frequently make such comparisons fraught, leading to a lack of consensus on what strategies are more efficient.

The **mlOSP** library is meant to be extensible in the sense of offering a simple interface to define new OSP instances. This is achieved by utilizing, where possible, function pointers. For example, R already offers a standardized interface for regression, consisting of the *fit* and *predict* methods. **mlOSP** can then piggy-back on that interface, allowing the user to easily "hook-up" a new regression method for the regression module. Similarly, **mlOSP** can easily handle user-defined system dynamics, incorporated through constructing a new instance of path-simulator. These features of **mlOSP** are illustrated at the end of the "User Guide" where we take examples from a couple of very recent papers and show how they can be embedded into the **mlOSP** template to facilitate comparison with existing ideas.

Optimal Stopping and RMC

An optimal stopping problem is described through two main ingredients: the state process and the reward function. We shall use X to denote the state process, assumed to be a stochastic process indexed generically by the time index t. The reward function is $h(t,x)$ where the notation emphasizes the common possibility of the reward depending on time, e.g. due to discounting.

We seek the rule τ, a *stopping time* to maximize expected reward:

$$\mathbb{E}[h(\tau, X_\tau)] \to \max!$$

To this end, we wish to evaluate the value function

$$V(t,x) = \sup_{\tau \in \mathcal{S}} \mathbb{E}[h(\tau, X_\tau)|X_t = x]$$

The state (X_t) is typically assumed to satisfy a Stochastic Differential Equation of Ito type,

$$dX_t = \mu(X_t)\, dt + \sigma(X_t)\, dW_t,$$

where (W_t) is a (multi-dimensional) Brownian motion.

To understand optimal stopping, it is most intuitive to think of it as a dynamic decision making. At each exercise step, the controller must decide whether to stop (0) or continue (1), which within a Markovian structure is encoded via the action map $A_t(x) \in \{0,1\}$. This action map gives rise to the stopping region

$$\mathcal{S}_t = \{x : A_t(x) = 0\}$$

where the decision is to stop and in parallel defines the corresponding $\tau = \min\{t : A(X_t) = 0\}$ Hence, solving an OSP is equivalent to classifying each x into $\mathcal{S}_t$ or its complement the continuation set. The respective objective function is the expected reward:

$$\widehat{V}(t,x)[A] := \mathbb{E}[h(\tau_A, X^x_{\tau_A})].$$

The RMC construction relies on recursively constructing A_t based on $(A_s : s > t)$. This is achieved by rephrasing

$$A_t(x) = 1 \quad \Leftrightarrow \quad \mathbb{E}[h(\tau_{A_{t+1}}, X^x_{\tau_{A_{t+1}}})] > h(t,x)$$

, i.e. one should contain if the expected reward-to-go dominates the immediate payoff. The Dynamic Programming Principle implies that the right-hand-side can be expressed as the conditional expectation of $V(t+1, X^x_{t+1})$, henceforth termed the continuation value

$$q(t,x) = \mathbb{E}[V(t+1, X^x_{t+1})|X_t = x].$$

This construction yields the following loop: 1. Set $V(T,x) = h(T,x)$ 2. For $t = T-1, ..., 1$ i) Learn the conditional expectation $\hat{q}(t,\cdot) = \hat{E}[\hat{V}(t+1,\cdot)|X_t = x]$ ii) Set $\hat{A}_t(x) = \{1 : \hat{q}(t,x) > h(t,x)\}$ iii) Set $\hat{V}(t,x) = \max(\hat{q}(t,x), h(t,x))$ 3. Run out-of-sample

$$\hat{V}(0,x_0) = \frac{1}{N'} \sum_{n'=1}^{N'} h(\tau_{\hat{A}(0)}, X_\tau)$$

The key step requiring numeric approximation is 2i. In the RMC paradigm it is handled by re-interpreting conditional expectation as the mean response within a stochastic input-output model. Thus, given an input x, there is a generative model which is not directly known but accessible through a pathwise reward simulator. The aim is then to predict the mean output of this simulator for an arbitrary x. Practically, this is done by running some simulations and then utilizing a statistical model to capture the observed input-output relationship. This statistical learning task can be broken further down into three sub-problems: 1. Defining the stochastic simulator 2. Defining the simulation design 3. Defining the regression step Recasting in the machine learning terminology, we need to define a simulator that accept a value x (the initial state at time t) and return Y which is a random realization of the pathwise reward starting at (t,x). We then need to decide which collection of x's should be applied as a training set. After selecting such experimental design of size N, $x^{1:N}$, we collect the $y^{1:N} = Y(x^{1:N})$ and reconstruct the model

$$Y(x) = f(x) + \epsilon(x), \qquad \mathbb{E}[\epsilon(x)] = 0, \mathbb{V}ar(\epsilon(x)) = \sigma^2(x)$$

where $f(x) \equiv \hat{q}(t, x)$.

We now make a few remarks: * The procedure is recursive, necessarily the simulator at step t is linked to the previous simulators at steps $s > t$. Therefore, errors will tend to back-propagate. * There is no "data" per se, the controller is fully in charge of selecting what simulations to run. Judicious choice of how to do so is our primary criterion of numerical efficiency. Deciding how to train $\hat{q}$ is a key step in RMC. * In classical ML tasks, there is a well-defined loss functions that quantifies the quality of the constructed approximation. In OSP, this loss function is highly implicit; ultimately we judge algorithm performance in terms of the final $\hat{V}(0, x_0)$ (higher is better). Thus, one must construct heuristics to translate this into the loss function for the local learning task. * RMC is a **sequence** of tasks, indexed by t. While the tasks are inter-related, since they are solved one-by-one, there is a large scope for modularization, adaptation, etc to be utilized. * The stochasticity in RMC comes from using the pathwise simulator and is ultimately based on the random shocks driving the evolution of (X_t). Thus, the stochasticity is deeply imbedded in the problem. Because it is so intrinsic, $\epsilon(x)$ must be understood not as observation noise but rather simulation noise. In particular, its statistical properties tend to be quite complex and non-Gaussian. * The basic loop makes it clear that like in standard regression, there ought to be a training set (used in the backward iteration) and a test set, used for estimating $\hat{V}(0, x_0)$. Because the latter is essentially a plain MC estimate of expected reward given the specific stopping rule $\tau(\hat{A})$, ie. of $\mathbb{E}[h(\tau_{\hat{A}(0)}, X_{\tau_{\hat{A}(0)}})]$, modulo MC error (captured by the LLN and CLT) it will yield a lower bound on the true maximum expected reward. In contrast, any attempt to use the training data to estimate expected rewards cannot come with any reasonably upper/lower bound guarantees.

Dynamic Emulation Template

Put the algorithm in here

Bringing it Back into Focus: Bermudan Option Pricing

To offer the most intuitive context for using **mlOSP**, the vignette below focuses on OSP instances coming from Bermudan option pricing. In this context, we take the point of view of a buyer of an American option, which is a financial contract that gives the holder the ability (but not the obligation) to obtain a certain payoff, contingent on the underlying asset price. For example, an American Put allows its owner the right to exchange the asset for a pre-specified strike K, equivalent to the payoff $(K - x)_+$. The contract has an expiration date T, and the exercise frequency is taken to be Δt (such as daily). The state X_t is then interpreted as the *share price* of the asset at date t, which is naturally must be non-negative. Taking into account the time-value of money, it follows that the reward function is

$$h_{Put}(t, x) = e^{-rt}(K - x)_+,$$

where r is the constant interest rate.

With this setup, the stopping set is known as the exercise region. Another term is the timing value, $T(t, x) = q(t, x) - h(t, x)$, so that exercising is optimal when the timing value is negative. The fact that the reward has a lower bound of zero, reflects the optionality and the fact that the holder can never lose money. It implies that $\epsilon(x)$ has a mixed distribution with a point mass of zero…

For Bermudan option pricing, the seminal breakthrough were the works of Longstaff and Schwartz and Tsitsiklis and van Roy, that popularized the RMC approach in this context. Especially the former, which contained a simple toy example with $N = 8$ paths has been adopted as pedagogical device to explain RMC. The LS approach proposed to use a single set of global simulations, and to adopt the full pathwise simulator.

Its other insight was that due to the special structure, learning $\hat{q}$ is only necessary in-the-money, i.e. in the region where $h(t,x) > 0$; otherwise it is clear that $T(t,x) > 0$ and hence $A_t(x) = 1$.

Numerous methods have since embellished and improved LS. One particularly large strand of literature has addressed different regression variants: * Piecewise regression with adaptive sub-grids by Bouchard and Warin * Regularized regression, such as LASSO, by Kohler]Kohler and KrzyÅ¼ak (2012)] * Kernel regressoin by Belomestny et al * Gaussian Process regression by Ludkovski * Neural nets by Kohler and XXX * dynamic trees by Gramacy and Ludkovski (Gramacy and Ludkovski 2015)

Getting Started with mlOSP

The following user guide highlights the key aspects of mlOSP. It is intended to be fully reproducible, so that the reader can simply cut-and-paste (or download the RMarkdown document online) the R code. Since all the algorithms are intrinsically based on generating random outputs based on the underlying stochastic simulator, where possible we fix the RNG seeds. Depending on the particular machine and R version, the seeds might nevertheless lead to different results.

Load the necessary libraries (there are quite a few since we try many different regression methods)!

```r
library(ks)
library(fields) # for plotting purposes, use quilt.plot in 2D
library(seqOSP)
library(DiceKriging)
library(tgp)  # use lhs from there
library(randtoolbox)  # use sobol and halton QMC sequences
library(randomForest)
library(earth)
library(hetGP)
library(kernlab)
```

The seqOSP library has several principal schemes for constructing the emulators that describe the estimated stopping rule. The emulators are indexed by the underlying type of simulation *design* which most affects the implementation. We have:

- osp.prob.design – this is the original Longstaff Schwartz scheme that generates forward trajectories that are then re-used as the design sites. In our notation, this is a pure probabilistic design, without any replication. It is married to a variety of regression methods, both parametric and nonparametric.
- osp.fixed.design – the generic RMC scheme with a pre-specified/fixed (in the sense of not sequential) design. In particular the design can be replicated using *km.batch* parameter. This approach nests the osp.prob.design when km.batch=1 and the input.domain is fully specified.
- osp.probDesign.piecewisebw – this is the Bouchard Warin implementation of RMC that utilizes a hierarchical (piecewise) linear model based on an equi-probable partition of the forward trajectory sites. Since we have a "quick-and-dirty" recursive construction of the sub-domains, the fits from this function *cannot* be fed into a forward simulator. However, the function directly accepts a collection of test trajectories that are evaluated in parallel with the backward DP iteration.
- mc.put.adaptree – sequential RMC with *dynaTree* emulators
- osp.seq.design — sequential RMC with GP (ie Stochastic Kriging) emulators that are used to construct sequential design acquisition functions via the posterior emulator variance

1D Toy Example

We start with a simple 1-D Bermudan Put. The underlying dynamics are given by Geometric Brownian Motion

$$dX_t = rX_t dt + \sigma X_t dW_t, \qquad X_0 = x_0$$

In the model specification below we have $r = 0.06, T = 1, \sigma = 0.2$ and the option payoff is a Put $(K - X_t)_+$ with $K = 40$. The exercise is possible 25 times before expiration, i.e $\Delta t = 0.04$.

We hand-build a few designs that cover the in-the-money region $[25, 40]$ using QMC sequences. As a start we use a Gaussian Process emulator with a replicated design. The replications are treated using the SK method (Ankenman et al 2012), pre-averaging the replicated outputs before training the GP. The GP has a constant prior mean, learned via the Universal Kriging equations (Ginsbourger et al 2013)

```
# a few prespecified designs
grid1 <- c(30, 33:39)
grid2 <- c(28, 30, seq(32.5, 39, by=0.5))
grid3 <- sort(15*sobol(15)+25)
grid4 <- sort(15*sobol(200)+25)

# specify the simulation model, the payoff, and the emulator setup
model1 <- c(final.runs=0, K=40,x0=40,sigma=0.2,r=0.06,div=0,T=1,dt=0.04,dim=1,sim.func=sim.gbm,
 lhs.rect=c(25,40),init.size=15,km.cov=4,km.var=1,cand.len=1000,look.ahead=1)

put1d.model <- model1
put1d.model$km.batch <- 200
put1d.model$pilot.nsims = 0
put1d.model$covfamily="matern5_2"
put1d.model$N <- 500
option.payoff <- put.payoff

# use the DiceKriging Gaussian Process emulator with fixed hyperparameters specified in
# km.cov, km.var. The default kernel is Matern-5/2
km.fit <- osp.fixed.design(put1d.model,input.domain=grid3, method="km")
```

Now we visualize the results from one time-step. To do so, we first predict the timing value based on a fitted emulator (at $t = 10\Delta t$ step) over a collection of points. We then plot the point estimate of $T(t, x)$ and the corresponding credible interval (which is organically provided by the GP emulator).

```
check.x <- seq(24, 40, len=500)   # predictive sites
myCol <- "grey"
km.pred <- predict(km.fit$fit[[10]],data.frame(x=check.x), type="UK")  # syntax specific to km
plot(check.x, km.pred$mean, lwd=2, type="l", xlim=c(27,40), ylim=c(-0.4,1.1), xlab='X', ylab='Timing Va
lines(check.x, km.pred$lower95, lty=2,col=myCol)  # 95% CI band, see documentation of predict.km
lines(check.x, km.pred$upper95, lty=2,col=myCol)
abline(h=0,lty=2)
rug(grid3, quiet=TRUE)
```

2D Average Put example

Next we try a 2D example where things get a bit more interesting. Here the payoff is the basket average Put,

$$(K - (X_1 + X_2)/2)_+$$

. We continue to use $K = 40$. Thus, the option is in-the-money when $X_1 + X_2 > 80$ in the example below. For the initial condition we primarily use $(40, 40)$ which is At-the-money.

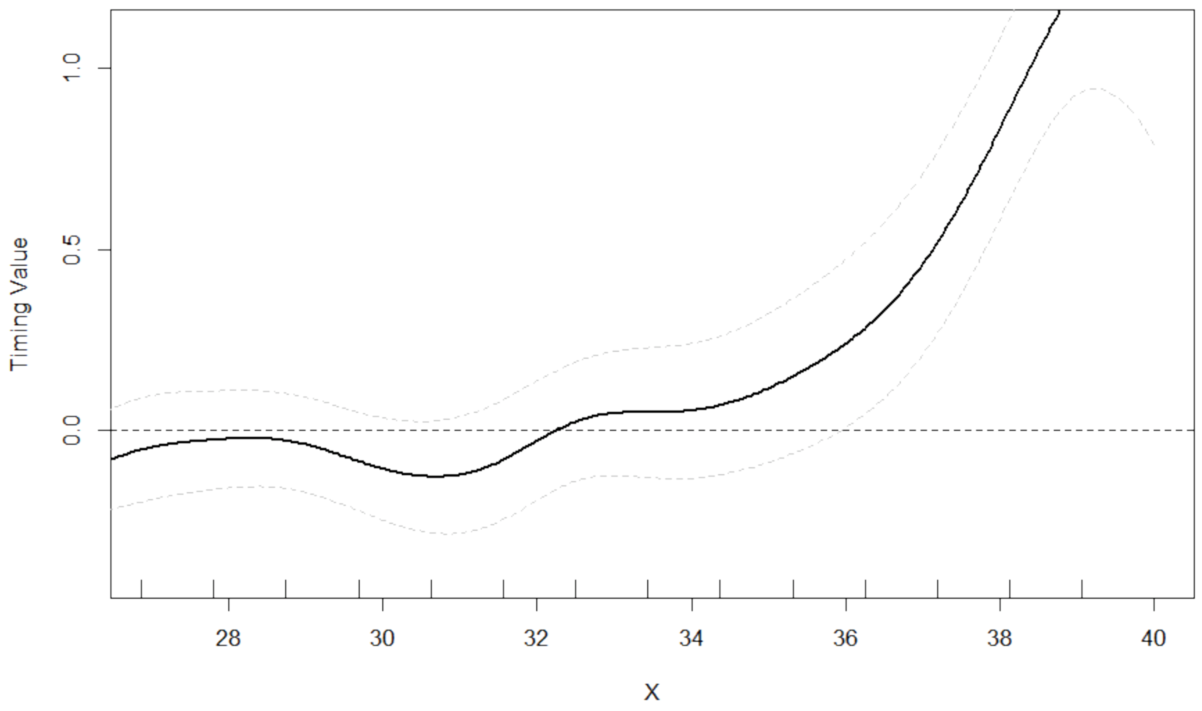


Figure 1: Timing Value of a Bermudan Put based on GP emulator. We also display the underlying simulation design (the rug plot) and the uncertainty quantification regarding the fit of T(t,x) (the shaded 95 CI)

The two assets are assumed to be uncorrelated and with identical dynamics, thus the whole metamodel should be symmetric in X_1, X_2.

```
al.params.km <- list(adaptive.grid.loop=100,look.ahead=1,init.size=100,final.runs=0,al.heuristic='sur',
km.batch=100,km.var=4,km.cov=c(6,6),km.upper=c(10,10))
lsmc.params <- list(mars.pen=1.5, mars.len=6,mars.thresh=1e-8,nChildren=5,rf.ntree=200,rf.nodesize=40,r

model2d <- c(al.params.km,
             list(K=40,x0=rep(40,2),sigma=rep(0.2,2),r=0.06,div=0,T=1,dt=0.04,dim=2,sim.func=sim.gbm))
option.payoff <- put.payoff
```

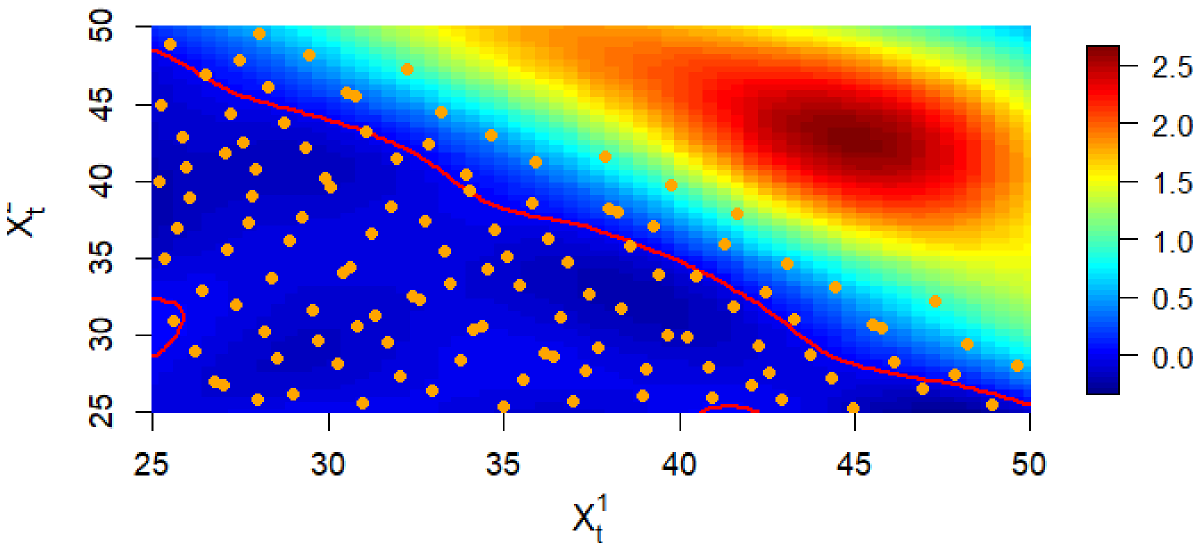
Test with a regular OLS regression against polynomial bases: a total of N=15000 training paths. To do so, we first manually define the basis functions that are passed to the **bases** model parameter, utilized when the method is set to "lm". Below we use a total of 5 bases $x_1, x_1^2, x_2, x_2^2, x_1 x_2$ (be default **lm** also includes the constant term, so there are a total of 6 regression coefficients $\vec{\beta}$).

```
bas22 <- function(x) return(cbind(x[,1],x[,1]^2,x[,2],x[,2]^2,x[,1]*x[,2]))
model2d$bases <- bas22

# subset= controls how the total of 30K simulations are split between training and testing
# Here we use a 50/50 split
prob.lm <- osp.prob.design(30000,model2d,method="lm",subset=1:15000)
```

```
## [1] "in-sample v_0 1.457890; and out-of-sample: 1.449412"
```

Repeat with GP: design of 150 sites replicated with 100 each. Uses the Gaussian squared-exponential kernel.

```
lhs.rect <- matrix(0, nrow=2, ncol=2)
lhs.rect[1,] <- c(25,55)  # atm is x1+x2 < 80
lhs.rect[2,] <- c(25,55)
model2d$lhs.rect <- lhs.rect
model2d$pilot.nsims <- 0
model2d$N <- 150
model2d$covfamily <- "gauss"

sob150 <- sobol(276, d=2)
sob150 <- sob150[ which( sob150[,1] + sob150[,2] <= 1) ,]  # a lot are on the diagonal
sob150 <- 25+30*sob150

sob.km <- osp.fixed.design(model2d,input.domain=sob150, method="km")
require(fields)
plt.2d.surf( sob.km$fit[[15]], x=seq(25,50, len=101), y=seq(25,50,len=101), ub=10)
```



The above plot visualizes the stopping boundary (red contour) and the fitted timing value (the stopping region is the level set where the timing value is negative). We also display the underlying Sobol QMC design of size 150.

To do a proper comparison between the fits we create a single out-of-sample database and run both stopping rules on it.

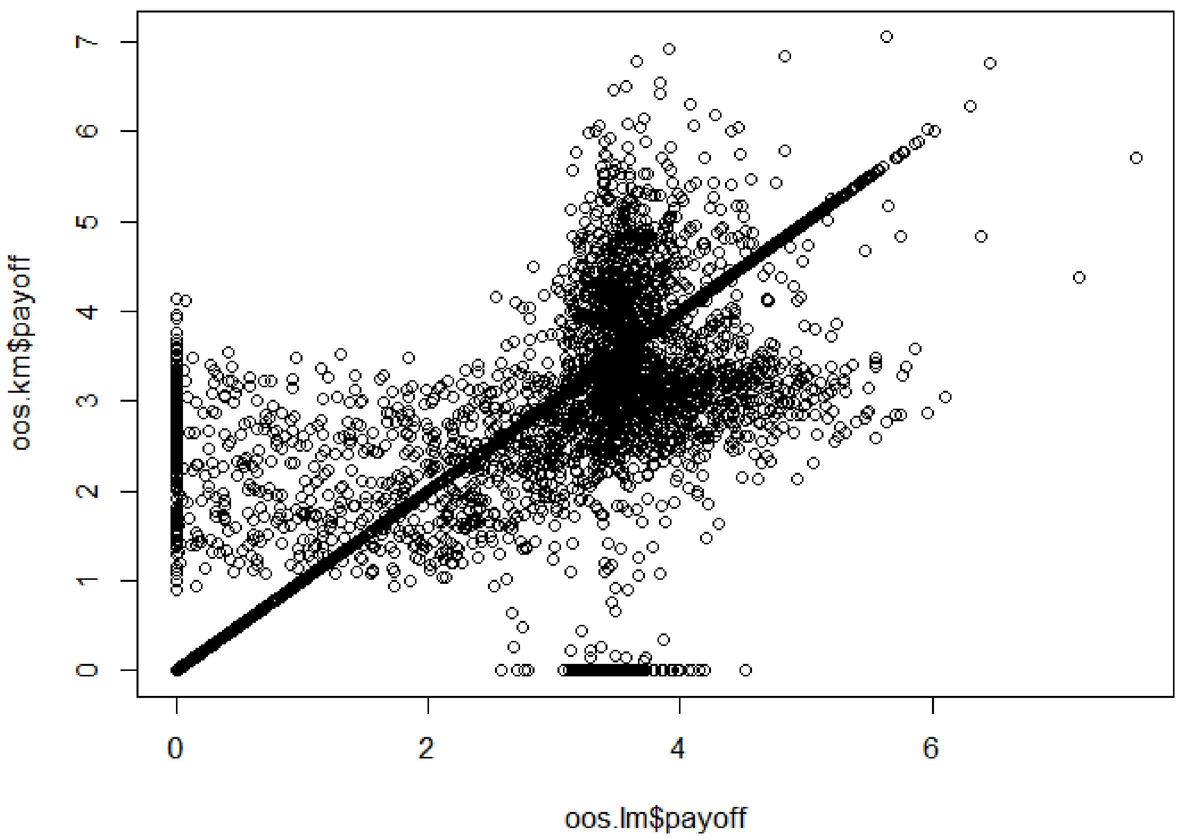
```
NN <- 16000
MM <- 25
set.seed(101)
mygr <- list()
mygr[[1]] <- model2d$sim.func( matrix(rep(model2d$x0, NN), nrow=NN, byrow=T), model2d, model2d$dt)
for (i in 2:(MM+1))
   mygr[[i]] <- model2d$sim.func( mygr[[i-1]], model2d, model2d$dt)
# sanity check: European option value
print(mean( exp(-model2d$r*model2d$T)*option.payoff(K=40,mygr[[MM]])))
```

```
## [1] 1.224074
```

```
oos.lm <- forward.sim.policy( mygr, MM, prob.lm$fit, model2d)
oos.km <- forward.sim.policy( mygr, MM, sob.km$fit,  model2d)
print( c(mean(oos.lm$payoff), mean(oos.km$payoff)) )
```

```
## [1] 1.426594 1.431723
```

```
plot(oos.lm$payoff, oos.km$payoff)
```



Different types of regression emulators

The first function is **osp.prob.design**. This is classical RMC that builds a design using a forward simulation of state trajectories. Its main input is the *method* which can take a large range of regression methods. Specifics of each regression are controlled through specifying *required* respective model parameters.

First we consider a 1D example with **smooth.spline**. Note the syntax: we use a total of 30000 trajectories, of which the first 10000 are reserved for testing, and the last 20000 is the training set. We then also run a **randomForest**.

```
put1d.model$nk=20  # number of knots for the smoothing spline
spl.eml.1dput <- osp.prob.design(30000,put1d.model,subset=1:10000,method="spline")
```

```
## [1] "in-sample v_0 2.307815; and out-of-sample: 2.326382"
```

```
put1d.model$rf.ntree = 200  # random forest parameters
put1d.model$rf.maxnode=100
rf.eml.1dput <- osp.prob.design(30000,put1d.model,subset=1:10000,method="randomforest")
```

```
## [1] "in-sample v_0 2.442872; and out-of-sample: 2.270105"
```

As a a last example, We try **MARS** (multivariate adaptive regression splines) from the **earth** package. We set the degree to be 2, so that bases consist of linear/quadratic hinge functions.

```
put1d.model$earth.deg = 2;  # earth parameters
put1d.model$earth.nk = 100;
put1d.model$earth.thresh = 1e-8
mars.eml.1dput <- osp.prob.design(30000,put1d.model,subset=1:10000,method="earth")
```

```
## [1] "in-sample v_0 2.301745; and out-of-sample: 2.229798"
```

Space-filling Batched designs

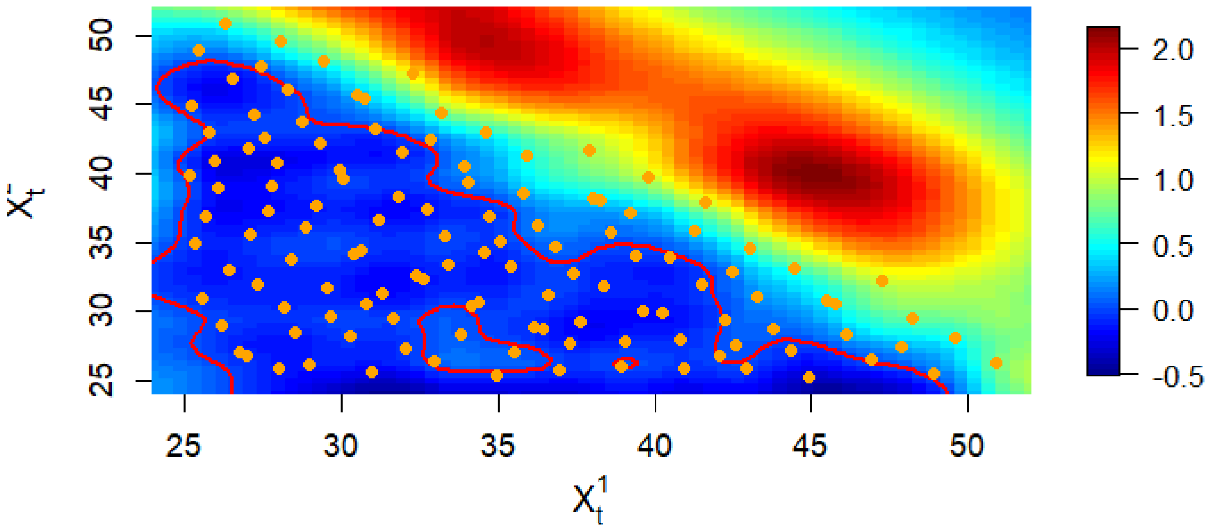
The second function, which is the workhorse of seqOSP, is **osp.fixed.design**. It has three key parameters: input.domain which controls the construction of the design, km.batch that controls the replication amount and type which controls the regression method. Because the design is now user-specified, its size can vary step-by-step. The package also allows to vary the replication amounts.

We proceed to an overview of different ways to build a space-filling design. The main issue is how to generate the bounding hyper-rectangle that defines the effective input space. The package provides several ways to do so.

The first example utilizes a user-defined simulation design that is used as-is. The constructed "sob150" macro-design places 150 design sites in a trianglular input domain, using a Sobol QMC sequence. For the latter we employ the *sobol* function in the **randtoolbox** package. Based on model parameters, the input domain is the lower-left triangle of $[25, 55]^2$. Every site is then batched with 100 replications/site for a total simulation budget of $N = 15000$.

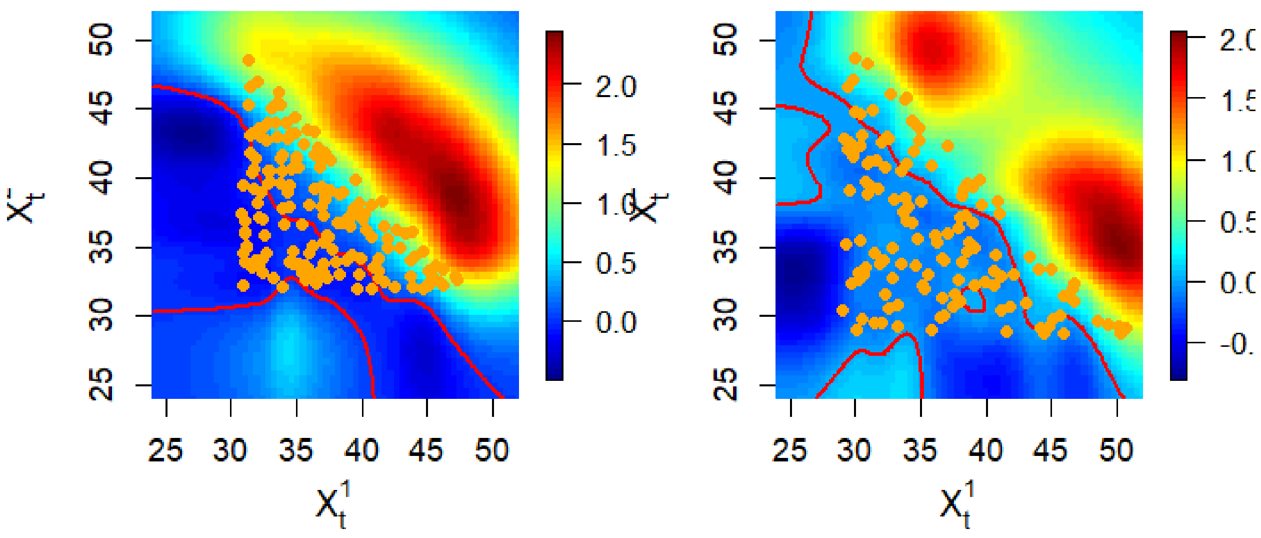
The **plt.2d.surf** function provides a way to visualize the emulator of $q(t, \cdot)$ at a single time-step, showing both the timing value $T(t, \cdot)$ and the stopping boundary (which is the zero-contour of the latter). It also shows the respective design $\mathcal{D}_t$ (the 150 points).

```
sob150 <- sobol(276, d=2)
sob150 <- sob150[ which( sob150[,1] + sob150[,2] <= 1) ,]  # a lot are on the diagonal
sob150 <- 25+30*sob150

model2d$km.batch <- 100
model2d$pilot.nsims = 0
model2d$covfamily="matern5_2"
model2d$N = 500   # size of unique design sites; effective size is smaller as only in-the-money sites a

put2d.sobol.km <- osp.fixed.design(model2d,input.dom=sob150, method="km")
plt.2d.surf(put2d.sobol.km$fit[[12]], x=seq(24,52,len=101),y=seq(24,52,len=101),ub=10)
```



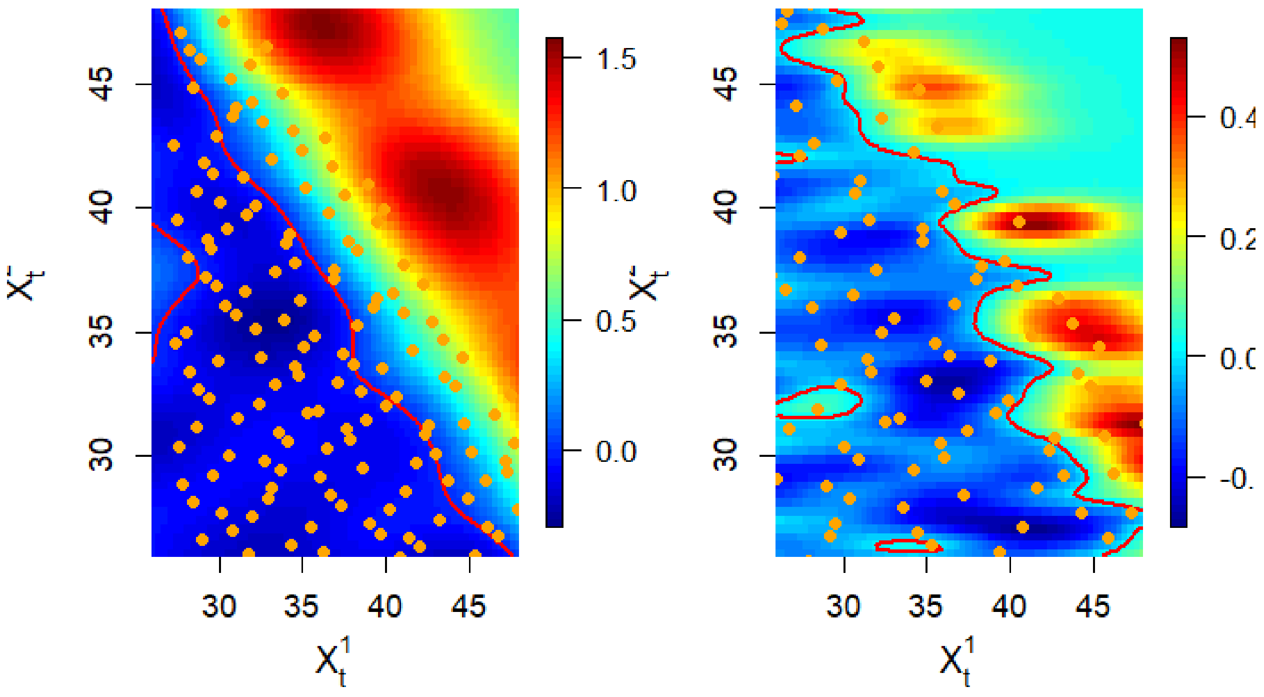
Secondly, we use an adaptive LHS design using some pilot simulations of $X_{0:T}$. Here *input.dom=0.02* makes the LHS space-fill a box between the 2nd and 98th percentiles of the pilot.nsim=1000 pilot trajectories at each time step. The pilot trajectories act as "scaffolding," organically expanding the input domain as t grows.

```
model2d$pilot.nsims <- 1000
model2d$N = 500   # size of unique design sites; effective size is smaller as only in-the-money sites a
model2d$qmc.method <- NULL
put2d.lhsAdaptive.km <- osp.fixed.design(model2d,input.dom=0.02, method="km")
par(mfrow=c(1,2))
plt.2d.surf(put2d.lhsAdaptive.km$fit[[10]],x=seq(24,52,len=101),y=seq(24,52,len=101))
plt.2d.surf(put2d.lhsAdaptive.km$fit[[20]],x=seq(24,52,len=101),y=seq(24,52,len=101))
```



If we set *input.dom=-1* then the full range of the pilot scenarios is used. Here we space-fill using a Halton QMC sequence. In addition, we also train the **DiceKriging** model hyperparameters, which is the Stochastic Kriging metamodel. Here for variety sake we pick a Gaussian kernel.

```
model2d$pilot.nsims <- 1000
model2d$km.upper <- 20
model2d$N = 500   # size of unique design sites; effective size is smaller as only in-the-money sites a
model2d$qmc.method <- randtoolbox::halton
model2d$covfamily <- "gauss"
put2d.haltonRange.trainkm <- osp.fixed.design(model2d,input.dom=-1, method="trainkm")
par(mfrow=c(1,2))
plt.2d.surf(put2d.haltonRange.trainkm$fit[[10]],x=seq(26,48,len=101),y=seq(26,48,len=101))
plt.2d.surf(put2d.haltonRange.trainkm$fit[[20]],x=seq(26,48,len=101),y=seq(26,48,len=101))
```



```
coef(put2d.haltonRange.trainkm$fit[[15]])
```

```
## $trend
## [1] 0.1625656
##
## $range
## [1] 3.605977 1.163375
##
## $shape
## numeric(0)
##
## $sd2
## [1] 0.073349
##
## $nugget
## numeric(0)
```

Heteroskedastic GP emulator

The classical GP emulator assumes homoskedastic Gaussian simulation noise. To tackle the strong heteroskedasticity encountered in our context, we have utilized above the Stochastic Kriging (SK) approach. SK utilizes a replicated design to locally estimate $\sigma^2(x) = \mathbb{V}\eth\diagdown(\epsilon(x))$. This estimation is done empirically via the classical MC variance estimator based on the batch of r pathwise rewards starting at the same x:

$$\sigma^2\hat{(x)} = \frac{1}{r-1}\sum_{i=1}^{r}(y^i - \bar{y})^2.$$

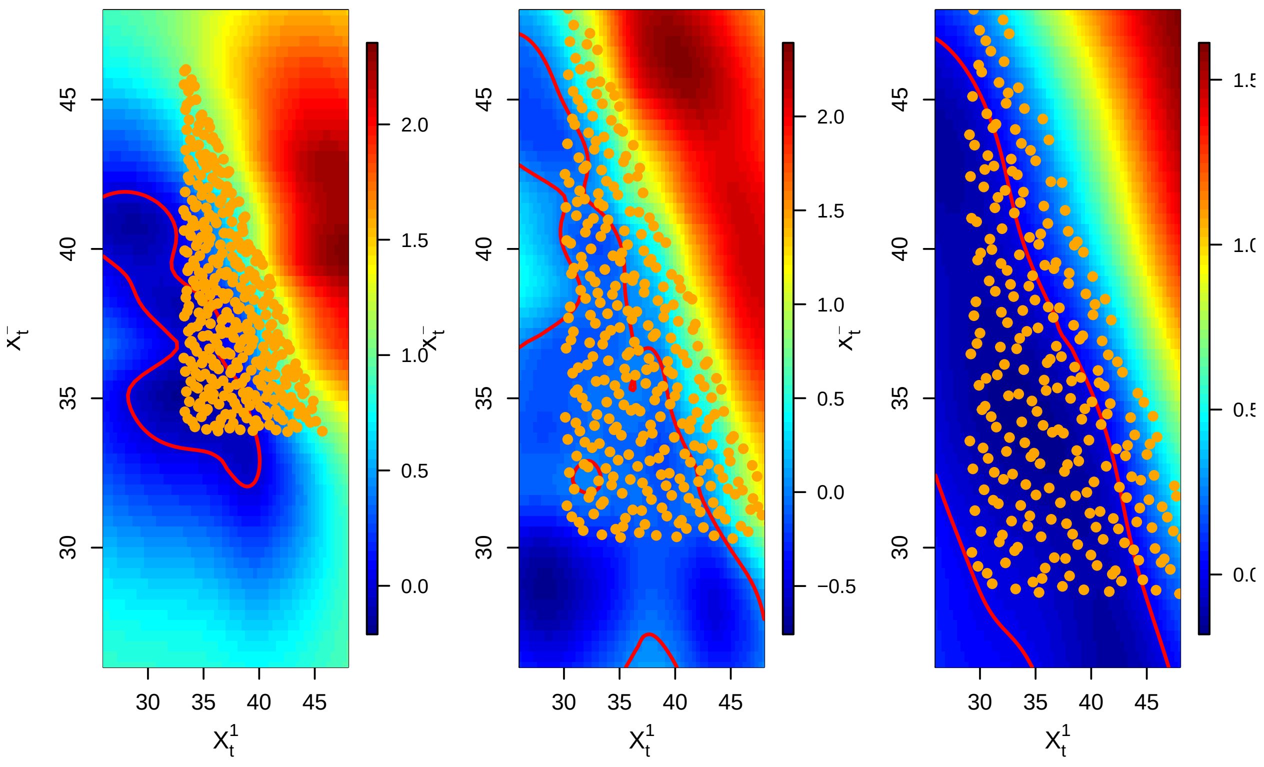
To be reliable, this strategy necessitates using large batch sizes r (called **n.reps** in mlOSP); in practice we find that $r \gg 20$ is necessary. For example, in the previous example we used $n.reps = 100$.

An alternative framework directly aims to learn $\sigma^2(\cdot)$ via a second spatial model that is jointly inferred with the standard mean response. This has been recently implemented (Binois et al 2018) in the **hetGP** package that we now illustrate. The main advantage of using **hetGP** is the ability to lower the replication counts

```
model2d$pilot.nsims <- 1000
model2d$km.upper <- 20
model2d$km.batch <- 25
model2d$covfamily <- "Matern5_2"  # note that hetGP has a slightly different naming convention for kern
model2d$N = 1000   # size of unique design sites; effective size is smaller as only in-the-money sites
model2d$qmc.method <- randtoolbox::halton
put2d.haltonAdaptive.hetgp <- osp.fixed.design(model2d,input.dom=0.02, method="hetgp")
```

```
## Homoskedastic model has higher log-likelihood:  -8930.404  compared to  -8942.43
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -9597.036  compared to  -9611.793
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -10319.91  compared to  -10331.29
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -11183.22  compared to  -11210.02
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -17780.54  compared to  -17784.91
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -18557.02  compared to  -18564.43
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -21844.39  compared to  -21850.73
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -21265.29  compared to  -21271.86
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -22178.58  compared to  -22190.17
## Return homoskedastic model
```

```
par(mfrow=c(1,3))
plt.2d.surf(put2d.haltonAdaptive.hetgp$fit[[6]],x=seq(26,48,len=101),y=seq(26,48,len=101))
plt.2d.surf(put2d.haltonAdaptive.hetgp$fit[[14]],x=seq(26,48,len=101),y=seq(26,48,len=101))
plt.2d.surf(put2d.haltonAdaptive.hetgp$fit[[22]],x=seq(26,48,len=101),y=seq(26,48,len=101))
```

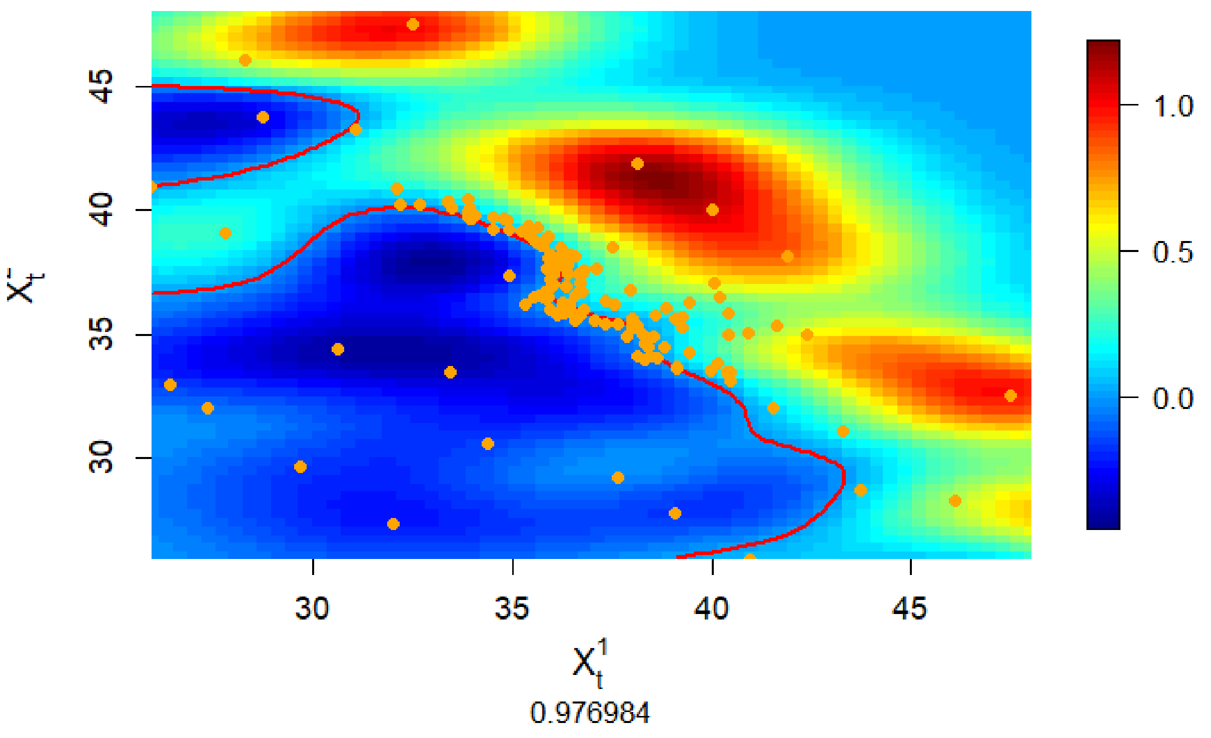


```
summary(put2d.haltonAdaptive.hetgp$fit[[14]])
```

```
## N =  7625  n =  305  d =  2
## Matern5_2  covariance lengthscale values of the main process:  8.281551 4.587073
## Variance/scale hyperparameter:  4.005986
## Summary of Lambda values:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  0.2720
```

```
plt.2d.surf(put2d.mcu.tp$fit[[10]],x=seq(26,48,len=101),y=seq(26,48,len=101),
            sub=sprintf("%4f",mean(oos.mcu.tp$payoff)))
```

```
## Warning in (object$nu + object$psi - 2)/(object$nu + n1 - 2) * as.vector(object$sigma2 - : Recycling
##   Use c() or as.vector() instead.
```



1D Put Example

We revisit the 1D Put example. Start with a grid of size 16 and go up to 100 total, or $N = 1000$ simulations.

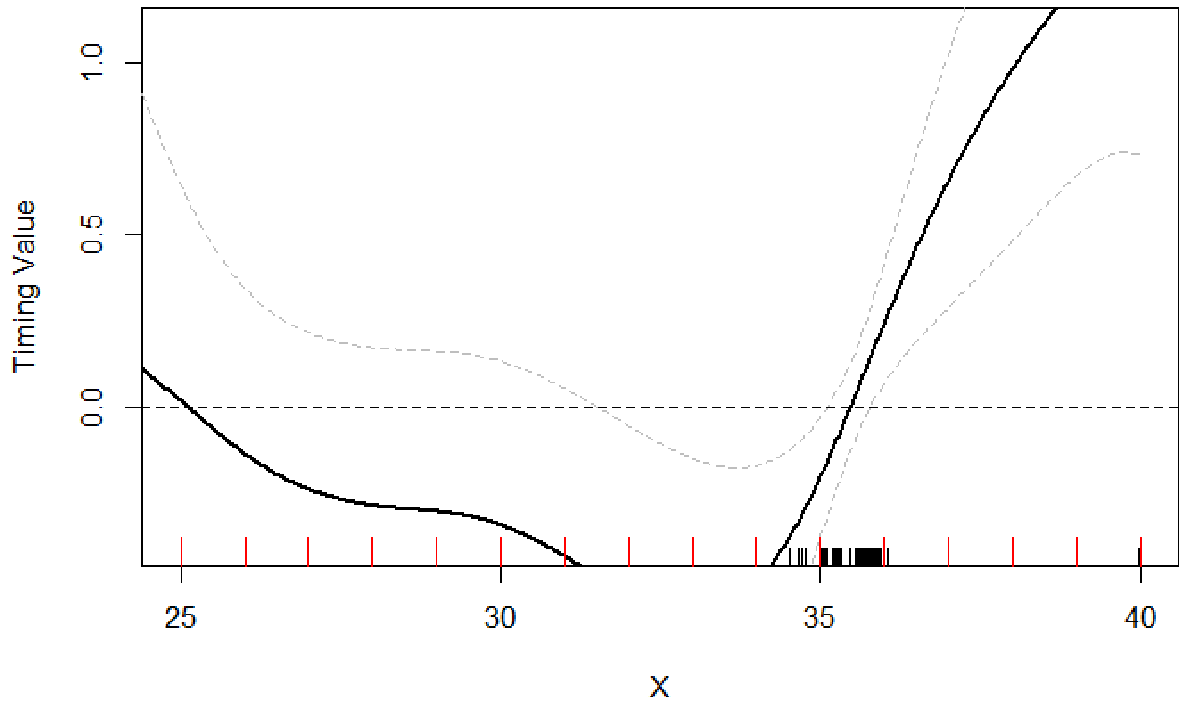
```
require(laGP)  # needed for distance function
model.seq1d <- put1d.model

option.payoff <- put.payoff
model.seq1d$adaptive.grid.loop <- 100  # final design size
model.seq1d$km.batch <- 10
model.seq1d$init.size <- 16  # initial design size
model.seq1d$init.grid <- as.matrix(seq(25,40,length=16))
model.seq1d$lhs.rect <- matrix(c(25,40),ncol=2)
model.seq1d$al.heuristic <- "csur"
model.seq1d$covfamily <- "Matern5_2"
model.seq1d$km.upper <- 8
put1d.sur.hetgp <- osp.seq.design(model.seq1d, method="hetgp")
```

```
# out-of-sample evaluation
NN <- 40000
MM <- 25
set.seed(102)
test.1d <- list()
test.1d[[1]] <- model1$sim.func( matrix(rep(c(40), NN), nrow=NN, byrow=T), model1, model1$dt)
for (i in 2:(MM+1))
```

```
   test.1d[[i]] <- model1$sim.func( test.1d[[i-1]], model1, model1$dt)
oos.hetgp.1d <- forward.sim.policy( test.1d, MM, put1d.sur.hetgp$fit, model.seq1d)
print(mean(oos.hetgp.1d$payoff))
```

```
## [1] 2.26985
```

```
check.x <- matrix(seq(24, 40, len=500))   # predictive sites
sur.pred <- predict(put1d.sur.hetgp$fit[[15]],x=check.x)  # at t=0.6
plot(check.x, sur.pred$mean, lwd=2, type="l", xlim=c(25,40), ylim=c(-0.4,1.1), xlab='X', ylab='Timing Va
lines(check.x, sur.pred$mean+2*sqrt(sur.pred$sd2), lty=2,col="grey")  # 95% CI band
lines(check.x, sur.pred$mean-2*sqrt(sur.pred$sd2), lty=2,col="grey")
abline(h=0,lty=2)
rug(put1d.sur.hetgp$fit[[15]]$X0, quiet=TRUE)
rug(model.seq1d$init.grid, quiet=TRUE, col="red",ticksize=0.05)
```



Building a New Model

As an example of how the user may easily work with **mlOSP**, we proceed to show the step-by-step process of implementing a new example based on the recent article by Cheridito et al (arxiv.org/1804.05394 (Becker, Cheridito, and Jentzen 2018)).

Specifically we consider a multivariate GBM model with asymmetric volatilities and constant correlation. The payoff functional is of the max-Call type already described above. We have

$$S_t^i = s_0^i exp([r - \delta_i - \sigma_i^2/2]t + \sigma_i W_t^i), \qquad i = 1, \dots, d$$

where the instantaneous correlation between W^i and W^j is ρ_{ij}. In the asymmetric example of Becker et al, $d = 5, s_0^i = s_0, \delta_i = \delta, \rho_{ij} = \rho$ and $\sigma_i = 0.08i$ with other parameter values $\delta = 10\%, r = 5\%, \rho = 0$ and contract specification $s_0 = 90, T = 3, K = 100, M = 9$.

We first define a new simulation function using the *rmvnorm* function in the **mvtnorm** library. To avoid passing too many parameters, we introduce new *model* fields *rho* (taken to be a constant) and *sigma* (a vector of length d).

```
require(mvtnorm)
```

```
## Loading required package: mvtnorm

## Warning: package 'mvtnorm' was built under R version 3.4.3
```

```
sim.corGBM <- function( x0, model, dt)
{
    sigm <- kronecker(model$sigma, t(model$sigma))  # matrix of sigma_i*sigma_j
    sigm <- model$rho*sigm + (1-model$rho)*diag(model$sigma^2)  # correct the diagonal to be sigma_i^2

    # implement the correlated GBM, specify the covariance matrix and the mean, both are linear in 'dt'
    newX <- x0*exp( rmvnorm(nrow(x0), sig=sigm*dt, mean= (model$r- model$div- model$sigma^2/2)*dt) )

    return (newX)
}
```

Next, we construct the problem instance, i.e. the model in mlOSP parlance.

```
modelBecker <- list(dim=5,sigma=0.08*(1:5), r= 0.05, div=0.1, rho=0, x0 = rep(90,5), T=3, K=100, dt=1/3
                    sim.func=sim.corGBM, km.upper=rep(100,5))
#model1 <- c(final.runs=0, K=40,x0=40,sigma=0.2,r=0.06,div=0,T=1,dt=0.04,dim=1,sim.func=sim.gbm,
# lhs.rect=c(25,40),init.size=15,km.cov=4,km.var=1,cand.len=1000,look.ahead=1)
```

For the design we consider a union of a space-filling design on $[50, 200]^d$ of size 400 and a probabilistic design of size 200. We are now ready to test with a *hetGP* metamodel which requires a few more parameter specification

```
sf400 <- 50 + 150*sobol(400,d=5, scrambl=1)
pd200 <- sim.corGBM( matrix(rep(modelBecker$x0,200), nrow=200,byrow=T), modelBecker, modelBecker$T)
option.payoff <- maxCall
modelBecker$km.batch <= 50
```

```
## logical(0)
```

```
modelBecker$covtype <- "Matern5_2"
modelBecker$pilot.nsims <- 100
modelBecker$N <- 600
modelBecker$look.ahead <- 1
#  pilot.nsims=1000, look.ahead=1, N=100,km.batch=50,km.upper=c(20,20), cand.len=1000, covtype="Matern5

beckerFit <- osp.fixed.design(modelBecker,input.dom=rbind(sf400,pd200), method="hetgp")
```

```
## Homoskedastic model has higher log-likelihood:  -2466.147  compared to  -2466.195
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -2606.585  compared to  -2606.607
## Return homoskedastic model
## Homoskedastic model has higher log-likelihood:  -2886.634  compared to  -2886.709
## Return homoskedastic model
```

We finally test on an out-of-sample set of scenarios.

```r
NN <- 100000
MM <- 9
set.seed(102)
test.Becker <- list()
test.Becker[[1]] <- modelBecker$sim.func( matrix(rep(modelBecker$x0, NN), nrow=NN, byrow=T),
                                          modelBecker, modelBecker$dt)
for (i in 2:MM)
   test.Becker[[i]] <- modelBecker$sim.func( test.Becker[[i-1]], modelBecker, modelBecker$dt)
# sanity check: European option value
print(mean( exp(-modelBecker$r*modelBecker$T)*option.payoff(K=100,test.Becker[[MM]])))
```

```
## [1] 24.5945
```

```r
oos <- forward.sim.policy( test.Becker, MM, beckerFit$fit, modelBecker)$payoff
print( c(mean(oos) - 1.96*sd(oos)/sqrt(NN), mean(oos)+1.96*sd(oos)/sqrt(NN)) )
```

```
## [1] 15.57871 15.88224
```

Voila. This can be compared against the reported interval of [27.63, 27.69]. We note the very high standard deviation of realized payoffs which leads to a wide credible interval of the final answer. Thus, obtaining tight bounds requires a very large out-of-sample test set.

Andersen, L., and M. Broadie. 2004. "A Primal-Dual Simulation Algorithm for Pricing Multi-Dimensional American Options." *Management Science* 50 (9): 1222–34.

Becker, Sebastian, Patrick Cheridito, and Arnulf Jentzen. 2018. "Deep Optimal Stopping." arXiv preprint arXiv:1804.05394.

Bouchard, B., and X. Warin. 2011. "Monte-Carlo Valorisation of American Options: Facts and New Algorithms to Improve Existing Methods." In *Numerical Methods in Finance*, edited by R. Carmona, P. Del Moral, P. Hu, and N. Oudjane. Vol. 12. Springer Proceedings in Mathematics. Springer.

Gramacy, R.B., and M. Ludkovski. 2015. "Sequential Design for Optimal Stopping Problems." *SIAM Journal on Financial Mathematics* 6 (1): 748–75. http://arXiv.org/abs/1309.3832.

Kohler, Michael, and Adam KrzyÅ¼ak. 2012. "Pricing of American Options in Discrete Time Using Least Squares Estimates with Complexity Penalties." *Journal of Statistical Planning and Inference* 142 (8). Elsevier: 2289–2307.

Longstaff, F.A., and E.S. Schwartz. 2001. "Valuing American Options by Simulations: A Simple Least Squares Approach." *The Review of Financial Studies* 14: 113–48.