Readme BB-Identity OSP.

Abel Guada Azze

August 10, 2018

The Optimal Stopping Problem

The modern formulation of the optimal stopping problem with finite horizon and non disconting factor goes as follows

$$V(t,x) = \sup_{0\le\tau\le T-t} \mathbb{E}_{t,x}[G(X_{t+\tau})] \tag{1}$$

where $T > 0$ is called the horizont, $t \in [0,T]$, and $x \in E$ (generally $E = \mathbb{R}$ or $E = \mathbb{R}_+$); $(X_s)_{s=0}^T$ is an stochastic process with space state E; the supreme above is taken among all the stopping times of $(X_s)_{s=t}^T$; the subscript in $\mathbb{E}_{t,x}$ indicates that $X_t = x$; V is called the value function; and G is called the payoff or gain function.

The goal here is to find the best strategy (the stopping time that maximazes the mean payoff) $\tau^*(t,x)$ and the value function $V(t,x) = \mathbb{E}_{t,x}[G(X_{t+\tau^*(t,x)})]$

Ussualy, $\tau^*(t,x)$ can be caracterized by means of the boundary between the so-called stopping set (i.e., the closed set $\bar{D} = \{(t,x) \in [0,T] \times \mathbb{R}_+ : V(t,x) = G(x)\}$) and its complement, the so-called continuation set (i.e., the open set $C = \{(t,x) \in [0,T] \times \mathbb{R}_+ : V(t,x) < G(x)\}$)

A common method to solve the optimal stopping problem (1) is by reformulating it into a free-boundary problem whose solution is both the value function and the boundary.

To know more about optimal stopping problems and, specifically, about the free-boundary technique, one can chek out the book (Peskir and Shiryaev 2006).

Our problem

We took G as the identity and $(X_s)_{s=0}^T$ as a Brownian bridge ending up in some point $y \in \mathbb{R}$ and endowed with an unknown volatility σ. Under these settings there exists a continuous non-increasing function $b : [0,T] \to \mathbb{R}$ such that $b(t) > y$ for all $t \in [0,T)$ and $b(T) = y$, playing the role of the boundary between the stopping set and the continuation set, and characterizing the optimal stopping time as follows [preprint bla bla]:

$$\tau^*(t,x) = \inf\{0 \le s \le T-t : X_{t+s} \ge b(t+s) \mid X_t = x\} \tag{2}$$

This is the repository companion of the preprint [bla bla], and it is intended to provide the users a set of tools for inferring and computing the boundary b based on a value of the volatility σ, which can be given in advance or estimated from the evolution of the stochastic process $(X_s)_{s=0}^T$.

In order to achieve that goal we broke down the code in four main block.

Simulation

This block is devoted to generate the data (Brownian bridges) the user might need to performe simulation studies to test (out) the results comming from the tools given at the computing and inferring blocks

`rBB`

Description

Simulates Brownian bridges.

Usage

```
rBB(n, a = 0, b = 0, sigma = 1, T = 1, N = 1e2, t = NULL)
```

Arguments

- `n` - number of Brownian bridges to be generated
- `a` - initial value X__0 = `a`
- `b` - final value X__T = `b`
- `sigma` - volatility of the process
- `T` - horizon
- `N` - number of equal spaced subintervals in which the interval $[0, \mathtt{T}]$ will be split
- `t` - discretization of the interval $[0, \mathtt{T}]$

Details

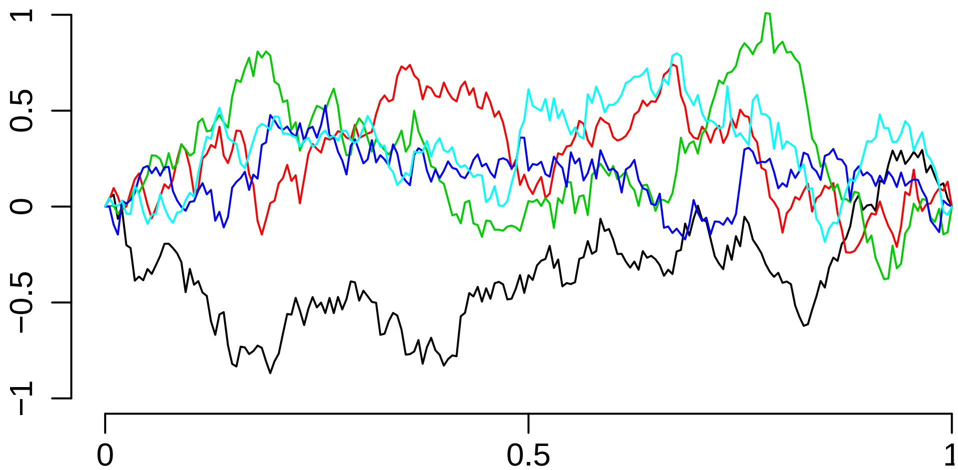
`rBB` simulates `n` Brownian bridges from $X_0 = \mathtt{a}$ to $X_{\mathtt{T}} = \mathtt{b}$ based on a Brownian motion with `sigma` volatility, and discretized at the points given by `t`. If `t` is not given it assumes the equally spaced discretization made of `N` subintervals.

Value

A matrix with `n` rows and `lenght(t)` columns, such that each row is a Brownian bridge's path discretized at the points indicated in `t`

Example

```r
set.seed(89)
BBs <- rBB(n = 5, N = 200)
matplot(seq(0, 1, by = 1/200), t(BBs), type = "l",
        lty = 1, lwd = 1, bty="n", yaxt='n', ylab = "",
        xlab = "", xaxt='n', ylim = c(-1,1))
axis(1, at = c(0, 0.5, 1), labels = c("0", "0.5", "1"), padj = -0.75)
axis(2, at = c(-1, -0.5, 0, 0.5, 1), labels = c("-1","-0.5", "0", "0.5", "1"), padj = 0.3)
```



`rBBB`

Description

Simulates Brownian bridges forced to stop by a given point.

Usage

```
rBBB(n, a = 0, b = 1, c = 0, sigma = 1,
     T = 1, N = 1e2, t = NULL, N1 = 1e2, normals)
```

Arguments

- `n` - number of Brownian bridges to be generated
- `a` - initial value $X_0 = \mathtt{a}$
- `b` - ordinate of the point where the process has to stop by
- `c` - final value $X_{\mathtt{T}} = \mathtt{b}$
- `sigma` - volatility of the process
- `T` - horizon
- `N` - number of equal spaced subintervals in which the interval $[0, \mathtt{T}]$ will be split
- `t` - discretization of the interval $[0, \mathtt{T}]$
- `N1` - number of the element of `t` that will be the abscissa of the point where the procees has to stop by
- `normals` - matrix whose rows are multivariate vectors of independent normal random variables with size `lenght(t)`

Details

`rBBB` simulates `n` Brownian bridges from $X_0 = \mathtt{a}$ to $X_{\mathtt{T}} = \mathtt{y}$, such that $X_{\mathtt{t}_{\mathtt{N1}}} = \mathtt{c}$, based on a Brownian motion with `sigma` volatility, and discretized at the points given by `t`. If `t` is not given it assumes the equally spaced discretization made of `N` subintervals.

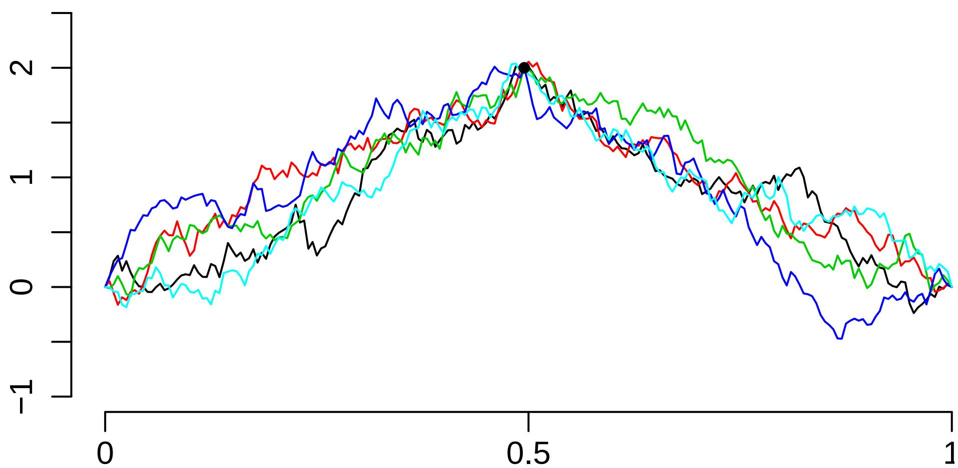
`normals` is a matrix with `n` rows and `lenght(t) - 2` columns, such that each row is a multivariate normal vector with the identity as the covariance matrix. It is used to generate the Brownian bridge's paths by adjusting the covariance matrix and the mean of each row. The reason it has only `lenght(t) - 2` columns is because it is known the paths will fit the points $(\mathtt{t}_{\mathtt{N1}}, \mathtt{c})$ and $(\mathtt{T}, \mathtt{b})$. `normals` can be useful when using a fixed seed, for example: $m + 1$ and m samples would have the first m samples in common; changing `N1` would output quite similar paths. If `normals` is not given, it is randomly generated.

Value

A matrix with `n` rows and `lenght(t)` columns, such that each row is a Brownian bridge's path discretized at the points indicated in `t` and forced to pass through (t_{N1}, b)

Example

```
set.seed(91)
BBs <- rBBB(n = 5, b = 2, N1 = 100, N = 200)
matplot(seq(0, 1, by = 1/200), t(BBs), type = "l",
        lty = 1, lwd = 1, bty="n", yaxt='n', ylab = "",
        xlab = "", xaxt='n', ylim = c(-1, 2.5))
points((100 - 1)/200, 2, pch = 20)
axis(1, at = c(0, 0.5, 1), labels = c("0", "0.5", "1"), padj = -0.75)
axis(2, at = c(-1, -0.5, 0, 0.5, 1, 1.5, 2, 2.5),
     labels = c("-1", "", "0", "", "1", "", "2", ""), padj = 0.3)
```



Computing the boundary

This entire project relies on having a fairly good numerical computation of the optimal stopping boundary when there is not even uncertainity, i.e., the value of σ is known. This is what this section stands for, which only function implements Algorithm 1 from the preprint [bla bla]. It was developed using the package Rcpp in R, which allows the integration of R and `C++`. This was done mainly for the sake of speed in simulation studies, which could take too much time if the boundary computation had a bad timing

`bBBCpp`

Description

Computes the optimal stopping boundary for the optimal stopping problem with a Brownian bridge as the underlying process and the identity as the gain function

Usage

bBBCpp(arma::vec sigma, arma::vec t = 0, double tol = 1e-3, double y = 0, double T = 1, int N = 2e2)

Arguments

- `sigma` - vector of volatilities
- `t` - discretization of the interval $[0, \texttt{T}]$
- `tol` - tolerance used in the point fixed algorithm
- `y` - final value $X_{\texttt{T}} = \texttt{y}$
- `T` - horizon
- `N` - number of equal spaced subintervals in which the interval $[0, \texttt{T}]$

Details

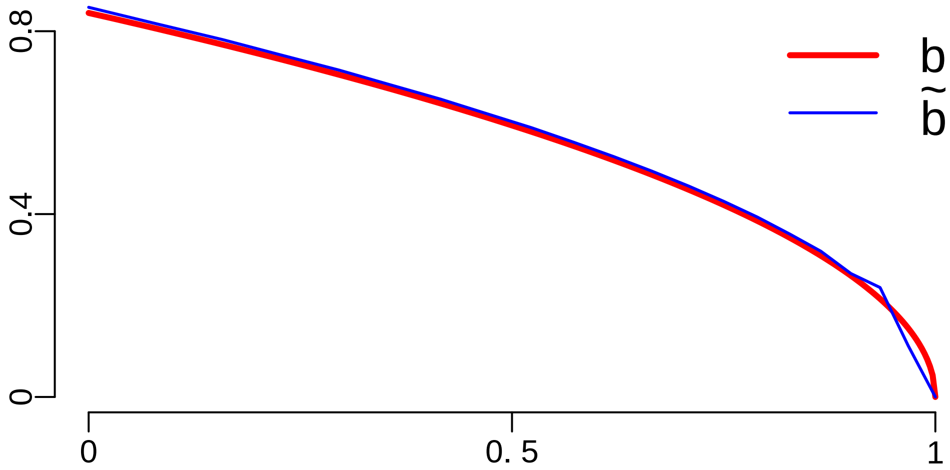
`bBBCpp` as many optimal stoping boundaries for the problem 1 as volatilities are given in `sigma`. The boundaries are computed via Algorithm 1 from the preprint [bla bla]. If $\texttt{t} = 0$, `bBBCpp` will use the equally spaced discretization made of `N` subintervals.

Value

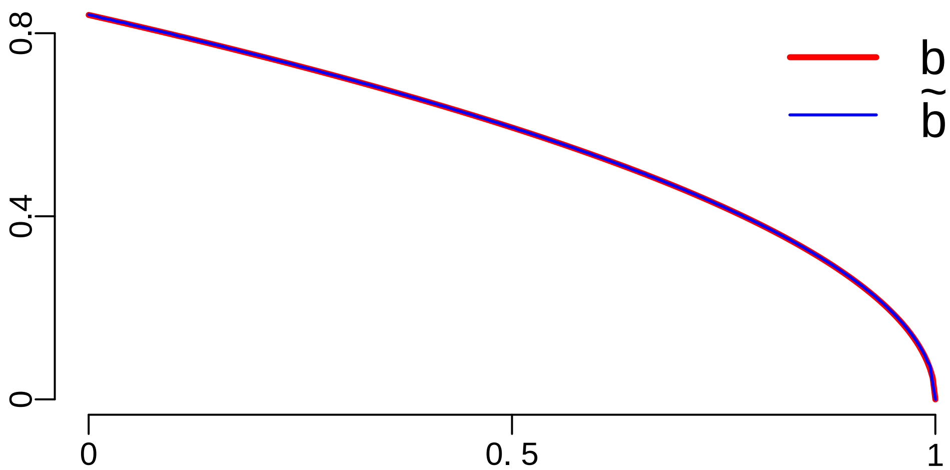
A matrix whose i-th column is a the optimal stoping boundary for problem 1, with a Brownian bridge having volatility given by `sigma[i]` as the underlying process, and the identity as the payoff function

Example

```
# libraries
library(Rcpp)
library(RcppArmadillo)
library(latex2exp)

Sys.setenv("PKG_CXXFLAGS" = "-std=c++11") # Needs C++11

# logarithmic  discretizations
t20 <- log(seq(exp(0), exp(1), l = 21))
t50 <- log(seq(exp(0), exp(1), l = 51))
t100 <- log(seq(exp(0), exp(1), l = 101))
t200 <- log(seq(exp(0), exp(1), l = 201))

# boundary computations
boundary20 <- bBBCpp(tol = 1e-3, y = 0, sigma = 1, t = t20)
boundary200 <- bBBCpp(tol = 1e-3, y = 0, sigma = 1, t = t200)

# true boundaary
boundary.true <- 0.8399*sqrt(1 - t200)

# plot 1
plot(t200, boundary.true, type = "l", lwd = 3, col = "red", bty="n",
     yaxt='n', xaxt='n', ylab = "", xlab = "")
lines(t20, boundary20, lty = 1, lwd = 1.5, col = "blue")
axis(1, at = c(0, 0.5, 1), labels = TeX(c("0", "0.5", "1")), padj = -0.8)
axis(2, at = c(0, 0.4, 0.8), labels = c("0", "0.4", "0.8"), padj = 1)
legend("topright", legend = c(TeX("$b$"), TeX("$\\tilde{b}$")), lty = c(1, 1),
       lwd = c(3, 1.5), bty = "n", col = c("red", "blue"), cex = 1.5)
```




```r
# plot 2
plot(t200, boundary.true, type = "l", lwd = 3, col = "red", bty="n",
     yaxt='n', xaxt='n', ylab = "", xlab = "")
lines(t200, boundary200, lty = 1, lwd = 1.5, col = "blue")
axis(1, at = c(0, 0.5, 1), labels = TeX(c("0", "0.5", "1")), padj = -0.8)
axis(2, at = c(0, 0.4, 0.8), labels = c("0", "0.4", "0.8"), padj = 1)
legend("topright", legend = c(TeX("$b$"), TeX("$\\tilde{b}$")), lty = c(1, 1),
       lwd = c(3, 1.5), bty = "n", col = c("red", "blue"), cex = 1.5)
```



Inference

This section holds two functions that tackle, respectively, the two task that involve inference, which are, roughly: to estimate the volatility of a Brownian bridge, and make that estimation extensible to the boundary.

`SigML`

Description

Estimates the volatility of a Brownian bridge using the maximum likelihood method

Usage

```
SigML(samp, T = 1, t = NULL, N1 = 2)
```

Arguments

- `samp` - a matrix whose rows are Brownian bridge's paths, not necessarily with the same volatility
- `N1` - an index between 1 and `ncol(samp)` - 1
- `T` - horizon
- `t` - discretization of the interval $[0, \mathtt{T}]$ where the Brownian bridges were sampled

Details

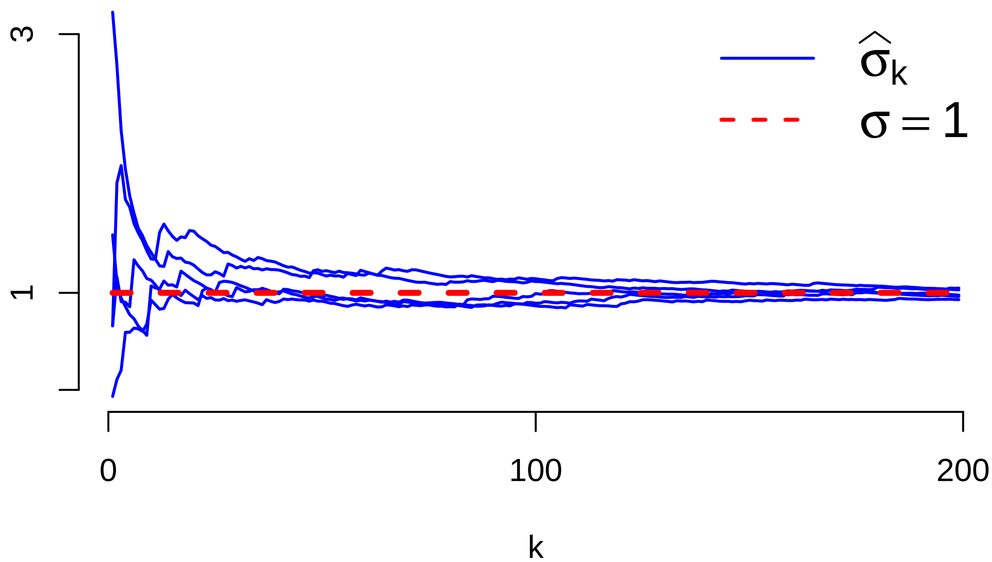
`SigML` takes the first `N1` values of each one of the discretized Brownian bridges given at `samp` and computes the maximum likelihood estimator for their volatilities. If `t` is not given, `SigML` will use the equally spaced discretization made of `N` subintervals.

Value

A vector whose i-th entry is the volatility estimated for the Brownian bridge at the i-th row of `samp`

Example

```r
# libraries
library(mvtnorm)
library(latex2exp)

set.seed(43)

# generating the Brownian briges samples
samp <- rBB(n = 5, N = 200)

# computing the volatilities
Sigma <- sapply(2:200, function(x) SigMl(samp = samp, N1 = x))

# plot
matplot(1:199, t(Sigma), type = "l", lty = 1, lwd = 1.5, col = "blue",
        xlab = TeX("$k$"), ylab = "", bty="n", yaxt = "n", xaxt = "n")
lines(1:199, rep(1, 199), lty = 2, lwd = 3, col = "red")
axis(1, at = c(0, 100, 200), labels = c("0", "100", "200"), padj = 0)
axis(2, at = c(0.25, 1, 3), labels = c("", "1", "3"), padj = 0)
legend("topright", legend = c(TeX("$\\widehat{\\sigma}_{k}$"), TeX("$\\sigma = 1$")),
       lty = c(1, 2), lwd = c(1.5, 2), col = c("blue", "red"), bty = "n", cex = 1.6)
```



`bBBConf`

Description

Computes confidence curves for the optimal stopping boundary of a BB-identity optimal stopping probrem, which was computed via bBBCpp and whose variance was estimated using the maximum likelihood method.

Usage

```
bBBConf(bnd, tol = 1e-3, y = 0, sigma = 1, T = 1, N = 1e2, t = NULL, nsamp, alpha = 0.05,
eps = 1e-2)
```

Arguments

- `bnd` - a matrix whose i-th row represents an optimal stopping boundary. This input is meant to be the output of `bBBCpp`
- `tol` - tolerance used in the point fixed algorithm. Despite it has not to be the same tolerance used for computing `bnd`, it is adviced to be the same to avoid the arising of numerical approximation errors
- `y` - final value $X_{\mathtt{T}} = \mathtt{y}$
- `sigma` - vector of estimated volatilities
- `T` - horizon
- `N` - number of equal spaced subintervals in which the interval $[0, \mathtt{T}]$
- `t` - discretization of the interval $[0, \mathtt{T}]$
- `nsamp` - integer indicating how many observations the estimation of volatilities in `sigma` was based on
- `alpha` - vector of confidence levels
- `eps` - increment in the incremental ratio

For a better understanding of the inputs see the section "Learning the volatility" of the preprint [bla bla]

Details

`bBBConf` takes the i-th optimal stopping boundary in `bnd`, and the j-th each element of `alpha`, and compute pointwise confidence curves at level `alpha[i]` for the boundary. Both `tol` and eps are used to approximate the derivatirve of the boundary with respect to the volatility by a incremental ratio, as described in the section “Learning the volatility” from the preprint [bla bla]. If `t` is not given, `bBBConf` will use the equally spaced discretization made of `N` subintervals.

Value

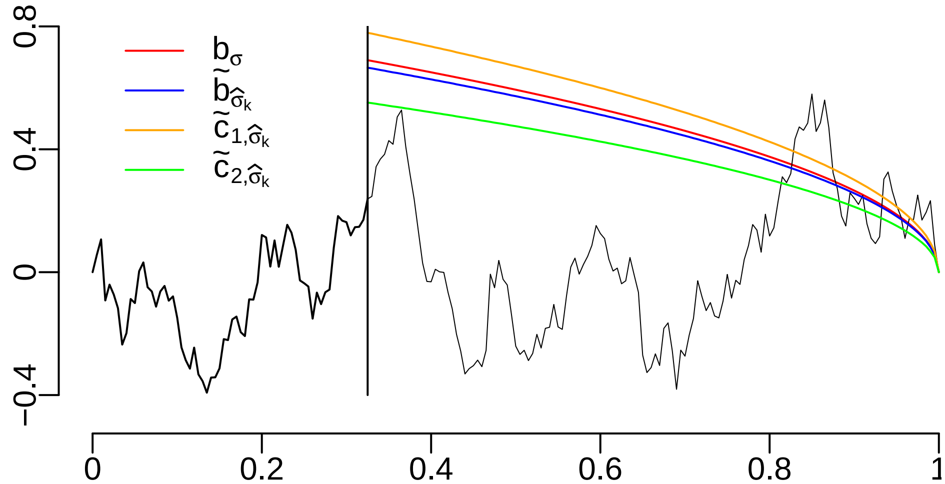
A list made of one matrix and tow hypermatrixes:

- `bBB` - the input `bnd` without any modification
- `bBB.low` - an hypermatrix (if `alpha` = it will be just a matrix 1) whose `i`-th row of the `j`-th matrix is the lower confidence curve at level `alpha[i]`
- `bBB.up` - an hypermatrix (if `alpha` = 1 it will be just a matrix) whose `i`-th row of the `j`-th matrix is the upper confidence curve at level `alpha[i]`

Example

```
set.seed(88)

# generating the Brownian bridge
N <- 200
samp <- rBB(n = 1, N = N)

# estimating the volatility using one third of the path
N1 <- floor(N / 3)
Sigma <- SigMl(samp = samp, N1 = N1)

# defining the logarithmic discretization
tlog <- log(seq(exp(0), exp(1), l = N + 1))

# computing the boundary with the estimated volatility
boundary <- bBBCpp(sigma = drop(Sigma), t = tlog)

# computing the confidence curves
alpha <- 0.05
boundary <- bBBConf(tol = 1e-3, bnd = boundary, sigma = drop(Sigma),
                    t = tlog, nsamp = N1, alpha = alpha, eps = 1e-2)

# creating the equally spaced partition
t <- seq(0, 1, by = 1/200)
boundary.true <- 0.8399*sqrt(1 - t)

# obtaining the values of the boundary over t by assuming linearity between points
boundary$bBB <- splinefun(tlog, boundary$bBB)(t)
boundary$bBB.low <- splinefun(tlog, boundary$bBB.low)(t)
boundary$bBB.up <- splinefun(tlog, boundary$bBB.up)(t)

# plot
plot(t, samp, type = "n", xlab = "", ylab = "", ylim = c(-0.5,0.75),
     xlim = c(0,1), bty = "n", yaxt = "n", xaxt = "n")
lines(t[1:N1], samp[1:N1], lty = 1, lwd = 1)
lines(t[N1:(N + 1)], boundary.true[N1:(N + 1)], lty = 1, lwd = 1, col = "red")
lines(t[N1:(N + 1)], samp[N1:(N + 1)], lty = 1, lwd = 0.5)
lines(t[N1:(N + 1)], boundary$bBB[N1:(N + 1)], lty = 1, lwd = 1, col = "blue")
```

```r
lines(t[N1:(N + 1)], boundary$bBB.up[N1:(N + 1)], lty = 1, lwd = 1, col = "orange")
lines(t[N1:(N + 1)], boundary$bBB.low[N1:(N + 1)], lty = 1, lwd = 1, col = "green")
lines(c(t[N1],t[N1]), c(-0.4, 0.85), lty = 1)
axis(1, at = c(0, 0.2, 0.4, 0.6, 0.8, 1),
     labels = c("0", "0.2", "0.4", "0.6", "0.8", "1"),
     padj = -1,  line = -0.2)
axis(2, at = c(-0.4, 0, 0.4, 0.8),
     labels = c("-0.4", "0", "0.4", "0.8"), padj = 1)
legend(x = 0.005, y = 0.85,
       legend = c(TeX("$b_{\\sigma}$"), TeX("$\\tilde{b}_{\\widehat{\\sigma}_{k}}$"),
                  TeX("$\\tilde{c}_{1,\\widehat{\\sigma}_{k}}$"),
                  TeX("$\\tilde{c}_{2,\\widehat{\\sigma}_{k}}$")),
       lty = c(1, 1, 1, 1), lwd = c(1, 1 , 1, 1),
       col = c("red", "blue", "orange", "green"), bty = "n")
```



Simulation study

This section is intended for the reproducibility of the simulation study performed in the preprint [bla bla], but one can perform similar simulation studies changing the settings at will

`VS`

Description

Generates the data needed for a comparison of the payoff associated to different stopping strategies: to stop at the true optimal stopping boundary, at its numerical computation via `bBBCpp`, or at one of the confidence curves.

Usage

```r
VS(q = c(0.2, 0.4, 0.6, 0.8), tol = 1e-3, n = 1e3, a = 0, y = 0, sigma = 1, T = 1, N =
2e2, t = NULL, t.boundary = NULL, alpha = 0.05, eps = 1e-2)
```

Arguments

- `q` - vector that determines which percentiles of the Brownian bridge are going to be taken for the initial conditions needed in order to stablish a comparition of the payoffs
- `tol` - tolerance used in the point fixed algorithm. Despite it has not to be the same tolerance used for computing `bnd`, it is adviced to be the same to avoid the arising of numerical approximation errors
- `a` - initial value $X_0 = \mathtt{a}$
- `y` - final value $X_{\mathtt{T}} = \mathtt{y}$
- `sigma` - volatility of the process
- `T` - horizon
- `N` - number of equal spaced subintervals in which the interval $[0, \mathtt{T}]$
- `t` - discretization of the interval $[0, \mathtt{T}]$ where the Brownian bridges are sampled
- `t.boundaruy` - discretization of the interval $[0, \mathtt{T}]$ where the boundaries are going to be computed
- `alpha` - confidence levels
- `eps` - increment in the incremental ratio

Details

For each `i`, `j` such that $1 \leq \mathtt{i} \leq \mathtt{lenght[t]}$ and $1 \leq \mathtt{j} \leq \mathtt{lenght[q]}$,`VS` generates `n` values of the payoff derived as follows:

- simulate `n` Brownian bridges via `rBBB` and force then to stop by $(\mathtt{t[i]}, X_{\mathtt{t[i]}}^{\mathtt{q[j]}})$, where $X_{\mathtt{t[i]}}^{\mathtt{q[j]}})$ is the `q[j]` percentile of the marginal distribution of the process at time `t`[i],
- use the paths of the Brownian bridges, from $X_{\mathtt{t[1]}}$ until $X_{\mathtt{t[i]}}$ to estimate a vector of `n` volatilities of the process
- compute `n` boundaries associated to the estimated volatilities using the function `bBBCpp`
- compute the `n` pairs of confidence curves associated to the boundaries
- look at the paths of the Brownian bridges, from $X_{\mathtt{t[j]}}$ until $X_{\mathtt{T}}$ and pick the first value that lies above the numerical computed boundary, the two confidence functions, and the true optimal stopping boundary

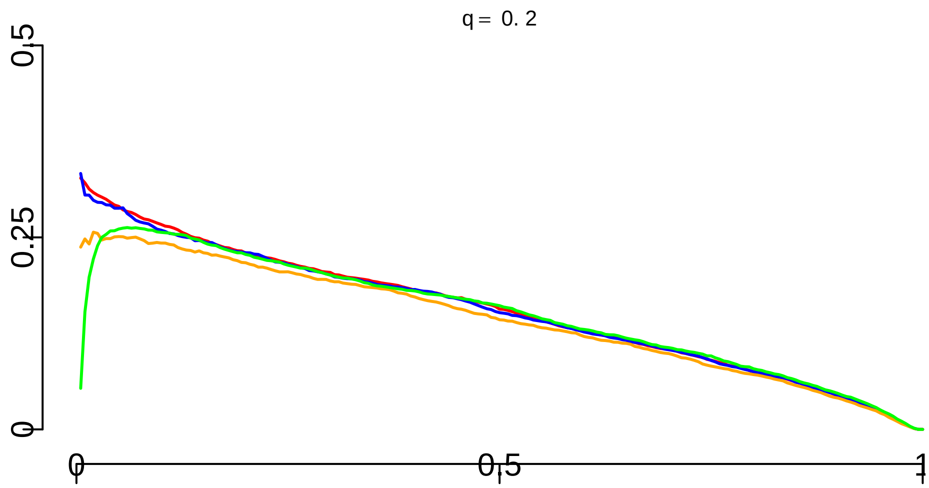
Value

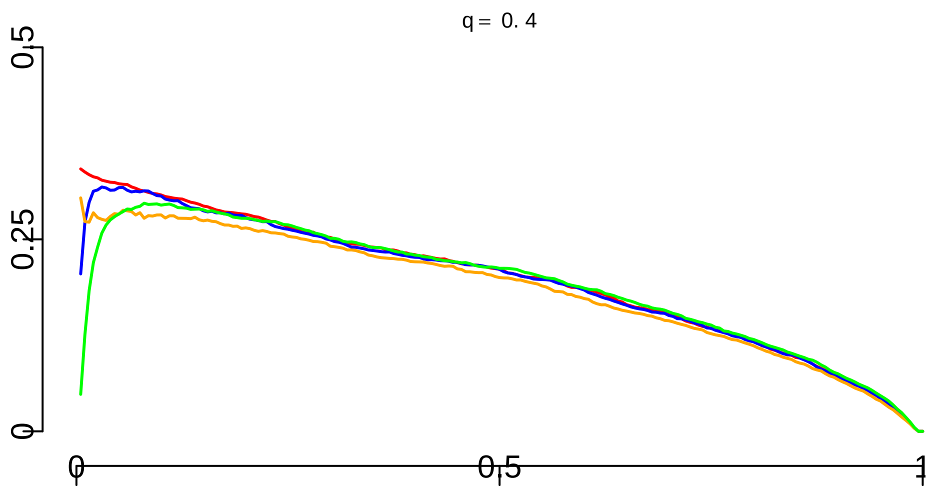
A hypermatrix whose entry (`i, j, k`) is the payoff associated to the `i`-th Brownian bridge with initial conditions $X_{\mathtt{t[j]}} = X_{\mathtt{t[j]}}^{\mathtt{q[k]}})$

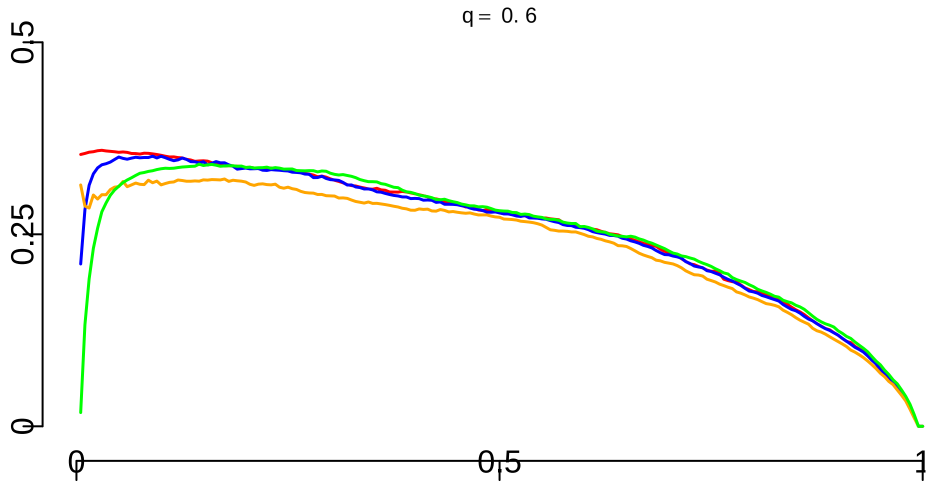
Example

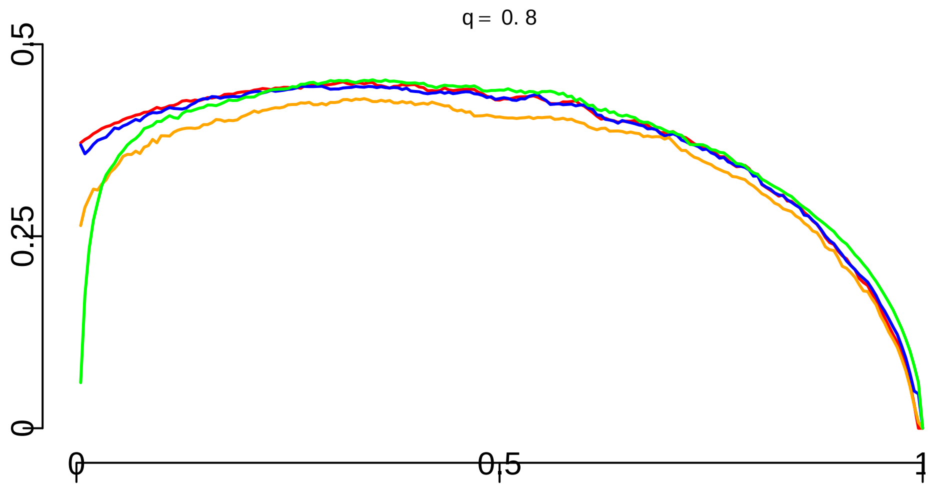
```
## Generating data (it could take a long time until done!)
# N <- 200
# tlog <- log(seq(exp(0), exp(1), l = N + 1))
# system.time(payoff_alpha05_sigma1 <- VS(sigma = 1, t.boundary = tlog))

# save(payoff_alpha05_sigma1, file = "payoff_alpha05_sigma1.RData")

# Loading dara
load(file = "payoff_alpha05_sigma1.RData")

x.axis <- seq(0, 1, by = 1/200)[-1]
percentiles <- c(0.2, 0.4, 0.6, 0.8)
y.lim <- c(.5, .5, .5, .5)

for (k in 1:4) {
```

```r
  plot(x.axis, colMeans(payoff_alpha05_sigma1$true[, , k]), type = "l",
       ylab = "", xlab = "", col = "red", lwd = 1.5, ylim = c(0, y.lim[k]),
       xlim = c(0,1), bty = "n", yaxt = "n", xaxt = "n")
  lines(x.axis, colMeans(payoff_alpha05_sigma1$est[, , k]),
        lty = 1, lwd = 1.5, col = "blue")
  lines(x.axis, colMeans(payoff_alpha05_sigma1$up[, , k]),
        lty = 1, lwd = 1.5,  col = "orange")
  lines(x.axis, colMeans(payoff_alpha05_sigma1$low[, , k]),
        lty = 1, lwd = 1.5, col = "green")
  axis(1, at = c(0, 0.5, 1), labels = c("0", "0.5", "1"), padj = -2.5,  line = 0.5)
  axis(2, at = c(0, 0.25, 0.5), labels = c("0", "0.25", "0.5"), padj = 1.6, line = 0)
  title(main = TeX(paste("$q =$", as.character(percentiles[k]))),
        cex.main = 0.7, line = 0.25)

}
```









References

Peskir, Goran, and Albert Shiryaev. 2006. *Optimal Stopping and Free-Boundary Problems.* Lectures in Mathematics. Eth Zürich. Birkhäuser.