Estimating local extrema using derivative Gaussian processes

In this vignette, we provide worked examples on simulated data to demonstrates how to use the package `dgp` to compute the posterior density of local extrama using derivative Gaussian processes, which is discussed in Li et al. (2023). Semiparametric Bayesian inference for local extrema of functions in the presence of noise.

```
library(dgp)
```

Simulated Data

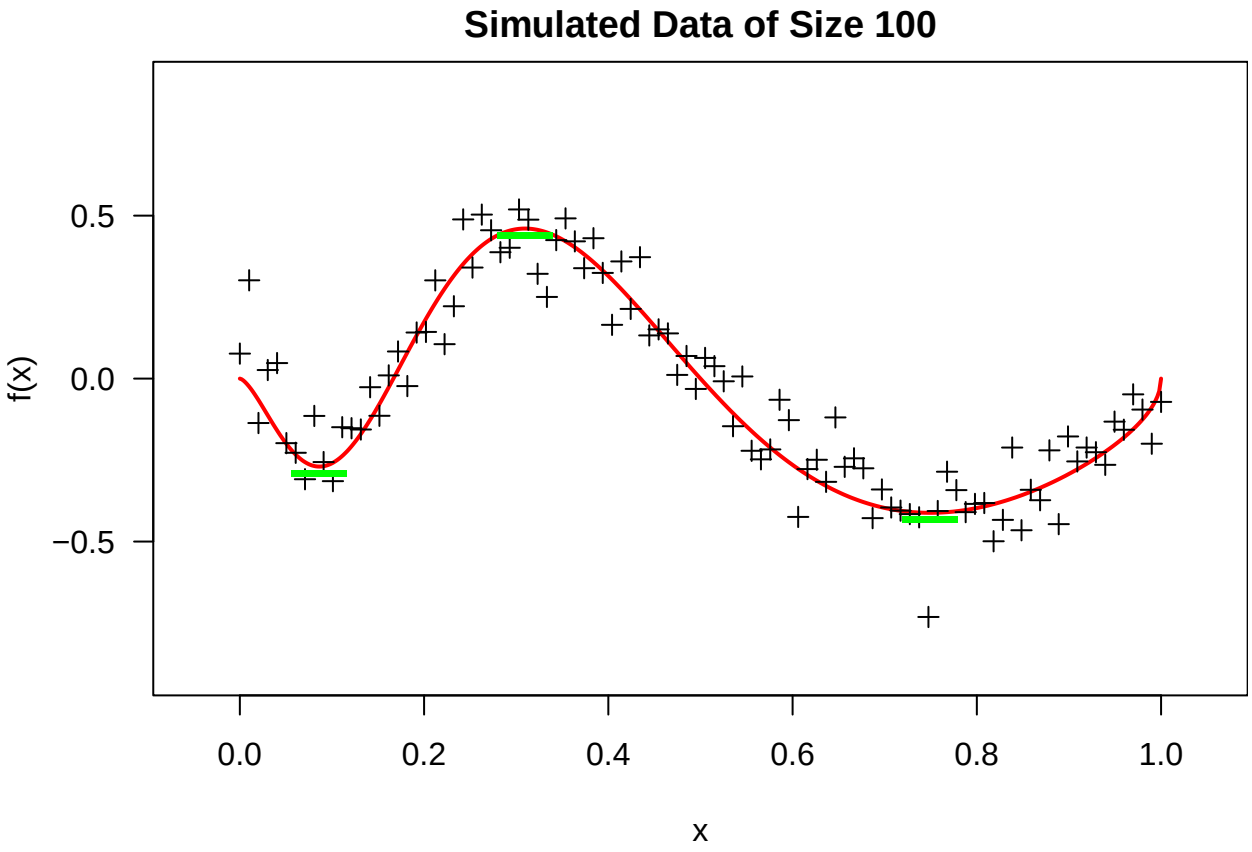
All 100 simulated data sets for analysis in the paper are stored in `./data/sim_data_n100.RData`. The simulated data are generated and plotted using the script `./data-raw/gen_sim_data.R` as shown below.

```
load("../data/sim_data_n100.RData", verbose = TRUE)
```

```
## Loading objects:
##   YY
##   sig
##   x_a
##   x_b
##   cri_pts
##   no_data
```

The object `YY` save the 100 simulated data sets. `x_a` and `x_b` are the end points of the domain of `x`. `cri_pts` stores the true stationary points, and `no_data` is the number of replicated data.

```
source("../data-raw/seed.R")
source("../data-raw/gen_sim_data.R")
```



Simulation study

The simulation study in the reference paper can be reproduced by running the script `dgp_theory.R`, including comparing the methods DGP Beta(1, 1), DGP Beta(2, 3), STS (Kovac, 2007), and NKS (Song et. al, 2006).

The script `effect_n.R` analyzes the effect of sample size on the posterior distribution of the local extrema, and reproduces Figure 2 in the paper. The script `estimate_number_point.R` reproduces Figure 3.

```
## required packages
# library(KernSmooth) ## for NKS
# library(ftnonpar) ## for STS 'GNU Fortran (GCC) 8.2.0'
library(matrixcalc)
library(emulator)
```

```
## Loading required package: mvtnorm
```

First create objects needed. `H0_diff` is the distance matrix in the powered exponential kernel function.

```
H0_diff <- lapply(YY, function(d) {
    outer(as.vector(d$x), as.vector(d$x),
          FUN = function(x1, x2) (x1 - x2))
})

idx <- seq(x_a, x_b, length.out = 500)
x_test <- sort(seq(x_a, x_b, length.out = 100))
n_test <- length(x_test)
n <- length(YY[[1]]$x)
grid_t <- seq(x_a, x_b, length.out = 400)
type1err <- 0.05
```

Then estimate σ, and the hyperpameters τ and lengthscale h in the kernel by maximizing the marginal log likelihood.

```
library(parallel)
library(doParallel)
```

```
## Loading required package: foreach
```

```
## Loading required package: iterators
```

```
library(Rsolnp)
cl <- makeCluster(6, type = "FORK")
registerDoParallel(cl)
EB_gp <- foreach(k = 1:no_data) %dopar% {
    res <- Rsolnp::solnp(pars = c(1, 1, 1), fun = log_mar_lik_gp,
                         LB = c(0.0001, 0.0001, 0.0001),
                         UB = c(1 / 0.0001, 1 / 0.0001, 1 / 0.0001),
                         control = list(TOL = 1e-5, trace = 0),
                         y = YY[[k]]$y, H0 = H0_diff[[k]])
    res$par
}
stopCluster(cl)
EB_gp[[1]]
```

```
## [1] 0.1155130 0.2896112 0.1300747
```

Given the hyperparameters, we apply the function `log_post_t_theory()` to compute the log posterior density of local extrema. The function requires the following arguments:

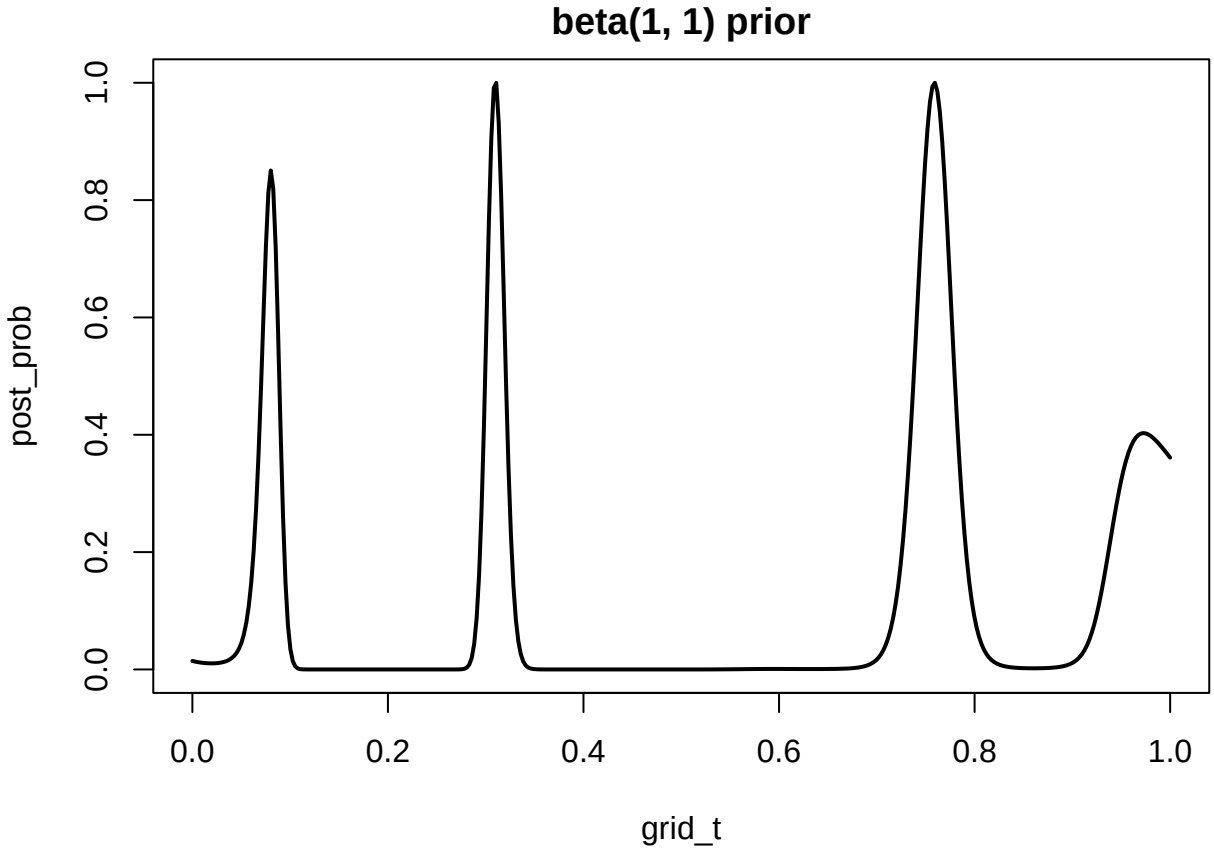
- `t`: The grid of t for evaluating the density value.
- `y`: the response value vector
- `x`: the input value vector
- `Kff`: The covariance matrix formed by the kernel function
- `A`: The matrix $K_{ff} + n\lambda I$ needed for computing the density
- `lambda`: the parameter $\lambda = \sigma^2/(n\tau^2)$
- `h`: the lengthscale parameter
- `sig`: the noise scale parameter
- `shape1`, `shape2`: the shape parameters of the beta prior on the stationary point.
- `a`, `b`: input domain.

The following is an example of the first simulated data set.

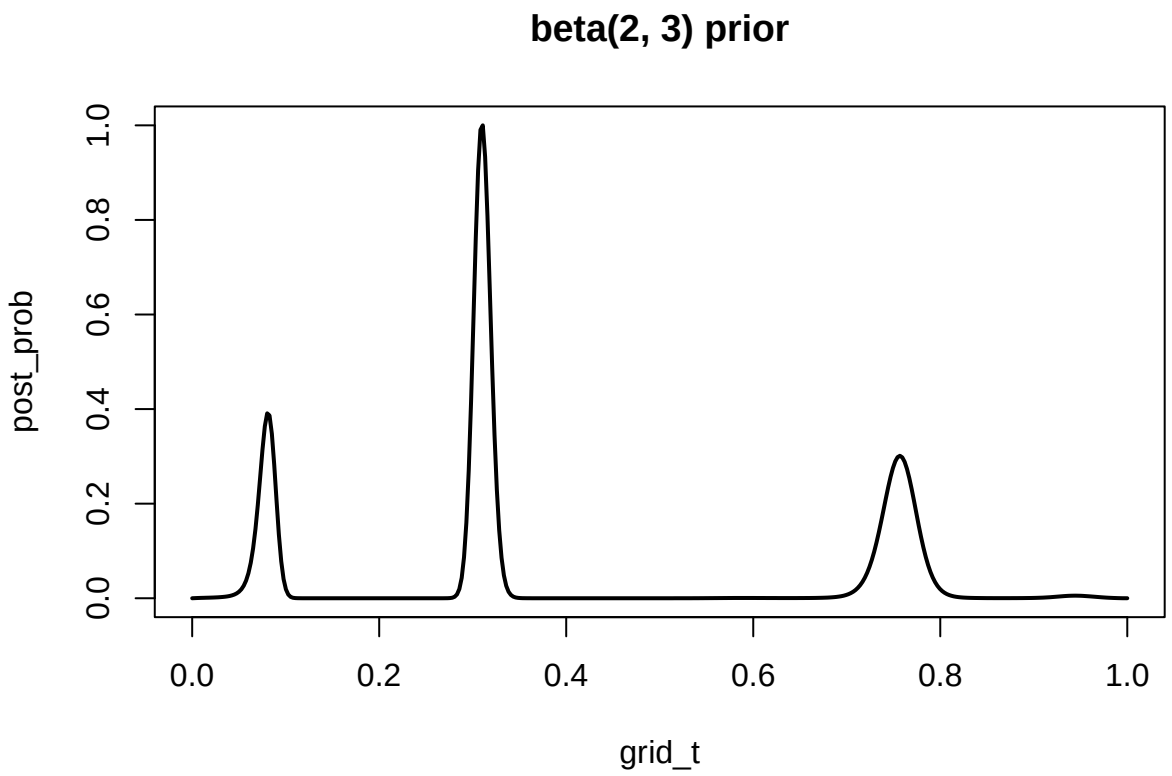
```
## ============================================================================
## DGP beta(1, 1) by Li et al. (2023)
## ============================================================================
log_post <- rep(0, length(grid_t))
sig_gp <- EB_gp[[1]][1]
tau_gp <- EB_gp[[1]][2]
h_gp <- EB_gp[[1]][3]
lambda_gp <- sig_gp ^ 2 / (n * tau_gp ^ 2)
Kff_gp <- se_ker(H0 = H0_diff[[1]], tau = 1, h = h_gp)
A_gp <- Kff_gp + diag((n * lambda_gp), n)
for (i in 1:length(grid_t)) {
    log_post[i] <-
        log_post_t_theory(t = grid_t[i], y = YY[[1]]$y, x = YY[[1]]$x,
                          Kff = Kff_gp, A = A_gp, lambda = lambda_gp,
                          h = h_gp, sig2 = sig_gp ^ 2, shape1 = 1, shape2 = 1,
                          a = x_a, b = x_b)
```

```
}
post_prob <- exp(log_post - max(log_post))
```

```
par(mar = c(4, 4, 2, 1))
plot(grid_t, post_prob, type = "l", lwd = 2, main = "beta(1, 1) prior")
```



The density with $beta(2, 3)$ prior is shown below.



The function `get_hpd_interval_from_den()` obtains the $(1-\alpha)100\%$ highest posterior density (HPD) intervals from the density. The following calculate the HPD intervals for the first data set when $beta(2,3)$ prior is used.

```
hpdi <- get_hpd_interval_from_den(post_prob, grid_t = grid_t, target_prob = 0.95)
hpdi
```

```
## $no_cluster
## [1] 3
##
## $ci_lower
## [1] 0.06015038 0.28822055 0.71929825
##
## $ci_upper
## [1] 0.0952381 0.3333333 0.7919799
##
## $prob_value
## [1] 0.9585813
##
## $den_value
## [1] 0.0412678
```

```
cri_pts
```

```
## [1] 0.08632681 0.30955769 0.74905641
```

There are three separated HPD intervals, each capturing one of the true critical points.

Another useful function is `get_map()` that obtains the maximum a posteriori (MAP) estimate from the posterior density.

```r
get_map(post_den = post_prob, grid_t = grid_t, hpdi = hpdi)
```

```
## [1] 0.0802005 0.3107769 0.7568922
```