Numerical stability of high-order kernels

In this short note, we investigate the numerical behavior of high-order kernels, depending on how they are built and whether their support is symmetric or not.

1 High-order kernels

1.1 Definition

In kernel density estimation or regression with the Nadaraya-Watson estimator, the kernel k is typically chosen as a *symmetric second-order* kernel (Epanechnikov, Gaussian, ...) with the following properties:

$$\int k(u)du = 1$$
$$\int uk(u)du = 0$$
$$\int u^2k(u) > 0.$$

The terminology *second-order* refers to the fact that the first non-zero moment of k is the second one (except for the zero-th order one which ensures the kernel is normalized). More generally, a *high-order* kernel of order r satisfies

$$\int k(u)du = 1$$
$$\int u^jk(u)du = 0,\ \forall j = 1, \dots, r-1$$
$$\int u^rk(u) > 0.$$

Here, we will focus on high-order kernels with **compact support**, which are used together with mirror-type transformations to avoid boundary effects appearing when the domain is compact [1, 5]. In particular, we will study symmetric kernels on $[-1, 1]$ and non-symmetric ones on $[0, 1]$.

1.2 Construction

In order to build a kernel of order r with compact support $[-1, 1]$, there are at least two approaches, which are described below.

Legendre orthonormal polynomials. The first construction relies on the (normalized) Legendre orthonormal polynomials on $[-1,1]$ denoted by $\{P_m(\cdot)\}_{m\in\mathbb{N}}$. Then we define the kernel k as

$$k(u) = \sum_{m=0}^{r+1} P_m(0)P_m(u)\mathbb{1}_{u\in[-1,1]}, \tag{1}$$

see [2, Proposition 2.8].

High-order Epanechnikov kernel. [3] proposes a high-order generalization of smooth and second-order kernels on $[-1,1]$ including the uniform, biweight, and Epanechnikov ones. Focusing on the latter, the kernel

$$k(u) = B_r(u)k_e(u) \tag{2}$$

where $k_e(u) = \frac{3}{4}(1-u^2)\mathbb{1}_{u\in[-1,1]}$ and

$$B_r(u) = \frac{\left(\frac{3}{2}\right)_{r/2-1}\left(\frac{5}{2}\right)_{r/2-1}}{(2)_{r/2-1}} \sum_{k=0}^{r/2-1} \frac{(-1)^k\left(\frac{r+3}{2}\right)_k u^{2k}}{k!(r/2-1-k)!\left(\frac{3}{2}\right)_k}$$

is of order r for odd r where $(x)_a$ is the Pochhammer's symbol.

As for kernels with compact support $[0,1]$, the two following methods can be envisioned.

Shifted Legendre orthonormal polynomials. Similarly to the first construction above, we can also consider the shifted Legendre orthonormal polynomials on $[0,1]$, denoted by $\{Q_m(\cdot)\}_{m\in\mathbb{N}}$, leading to

$$k(u) = 2\sum_{m=0}^{r+1} Q_m(0)Q_m(u)\mathbb{1}_{u\in[0,1]}. \tag{3}$$

Dilatation. Another approach, due to [4], relies on dilatations of an integrable function $g:\mathbb{R}\to\mathbb{R}$:

$$k(u) = \sum_{k=1}^{r}\binom{r}{k}(-1)^{k+1}\frac{1}{k}g\left(\frac{u}{k}\right). \tag{4}$$

If g has support $[a,b]$, then k has support $[a,rb]$ and is of order r. To obtain a kernel with support $[0,1]$, one can for example take a shifted Epanechnikov kernel k_{shift} on $[0,1/r]$:

$$k_{\text{shift}}(u) = 6u(1-ru)r^2\mathbb{1}_{u\in[0,1/r]}.$$

2 Numerical stability

In what follows, we investigate numerically the high-order kernels introduced above. Since kernels in (1) and (3) are identical up to a shift, we only focus on kernels as defined in (2) for $[-1, 1]$ and (3) and (4) for $[0, 1]$. They are coded below, note that they all take as input a parameter h which corresponds to the kernel bandwidth.

```
> library(orthopolynom)
> # Kernel (2)
> kepach_norm <- function(x,h,order){
+   z_squared <- (x/h)^2
+   k <- 3/4*(1-z_squared)
+   r <- order/2
+   B <- 0
+   for (i in 0:(r-1)){
+     B <- B + (-1)^i*pochhammer(3/2+r,i)*z_squared^i/factorial(i)/factorial(r-1-i)/pochhamm
+   }
+   B <- B*pochhammer(3/2,r-1)*pochhammer(5/2,r-1)/pochhammer(2,r-1)
+   k <- B*k*(z_squared<1)
+   return(k/h)
+ }

> # Kernel (3)
> klegendre <- function(x,h,order){
+   order <- order-1
+   x <- x/h
+   id_notdom <- (x<0) | (x>1)
+   list.poly=legendre.polynomials(order, normalized=FALSE)
+   K0 <- (1+2*(0:order))*unlist(polynomial.values(list.poly, -1))
+   Ktemp <- polynomial.values(list.poly, 2*x-1)
+   Kprod <- lapply(1:(order+1),function(i) K0[i]*Ktemp[[i]])
+   k <- Reduce("+", Kprod)
+   k[id_notdom] <- 0
+   return(k/h)
+ }

> # Kernel (4)
> kbase <- function(x,order){
+   return(x*(1-order*x)*(x>0)*(order*x<1)*6*order^2)
+ }
> korder <- function(x,h,order){
+   order <- order-1
+   x <- x/h
+   k <- 0
+   for (i in 1:order){
+     k <- k + choose(order,i)*(-1)^(i+1)*kbase(x/i,order)/i
```

```
+   }
+   return(k/h)
+ }
```

2.1 Kernel values versus order

As a simple illustration, we first display the kernel values when the order increases.

```
> order_to_test <- c(2,4,6,8,10)
> norder <- length(order_to_test)
> kernels_to_test <- c("klegendre","korder","kepach_norm")
> nkernels <- length(kernels_to_test)
> nx <- 1000
> x_test <- seq(-1,1,length.out=nx)
> data_plot <- NULL
> for (o in 1:norder){
+   for (k in 1:nkernels){
+     y_test <- do.call(kernels_to_test[k],list(x=x_test,h=1,order=order_to_test[o]))
+     data_plot <- rbind(data_plot,data.frame(x=x_test,y=y_test,
+                                             kernel=rep(kernels_to_test[k],nx),
+                                             order=rep(order_to_test[o],nx)))
+   }
+ }
> library(ggplot2)
> ggplot(data_plot, aes(x = x, y = y, color = kernel)) +
+   geom_line() +
+   facet_wrap(~order, scales = 'free') +
+   theme_minimal()
```

It appears clearly that non-symmetric kernels with support $[0, 1]$ exhibit large variations which increase with the order, as opposed to the symmetric kernel on $[-1, 1]$. This implies that numerical instabilities when computing estimators are to be expected, as illustrated below on a simple regression case.

2.2 Regression with mirror transformations

Now we consider a standard regression setting: we have access to a n-sample (X_i, Y_i) for $i = 1, \ldots, n$ with

$$Y_i = m(X_i) + \epsilon_i$$

where X_i are i.i.d. random variables with domain $[0, 1]$ and ϵ_i is a centered noise. The goal is to build a nonparametric estimate $\hat{m}$ of the regression function m.

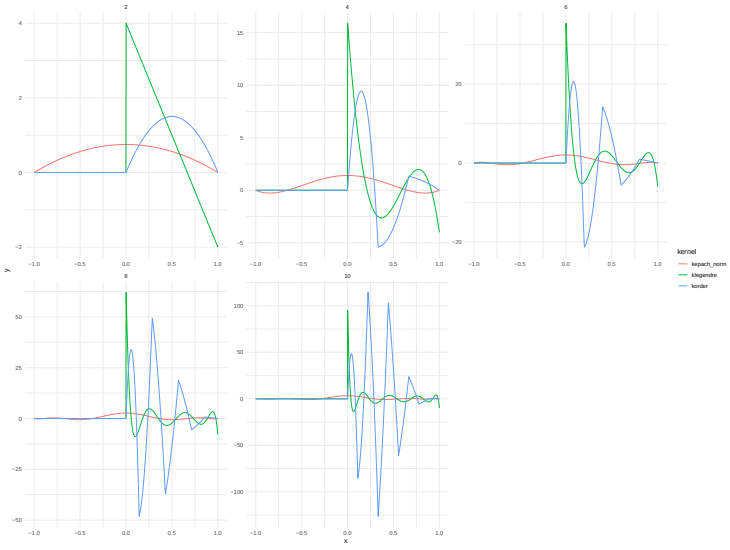

Figure 1: Kernel values versus order for symmetric kernel (2) (kepach_norm) and non-symmetric kernels (3) (klegendre) and (4) (korder).

2.2.1 Definition of kernel estimators

To do so, we consider two mirror-type kernel estimators:

$$\widehat{m}^1(x) = \frac{\sum_{j\neq i} Y_j k_h^1 \circ A_x(X_j - x)}{\sum_{j\neq i} k_h^1 \circ A_x(X_j - x)} \tag{5}$$

$$\widehat{m}^2(x) = \frac{\sum_{j\neq i} Y_j \sum_{a\in\{-1,0,1\}^d} k_h^2\big(M^a(X_j) - x\big)}{\sum_{j\neq i} \sum_{a\in\{-1,0,1\}^d} k_h^2\big(M^a(X_j) - x\big)} \tag{6}$$

where

- for $\widehat{m}^1$, kernel k_h^1 writes as $k_h^1(u) = k(u/h)/h$ for a kernel k with support $[0,1]$ and A_x is a mirror transformation defined as $A_x(u) = (1 - 2\mathbb{1}_{(1/2,1)(x)})u$, see [1].
- for $\widehat{m}^2$, kernel k_h^2 writes as $k_h^2(u) = k(u/h)/h$ for a kernel k with support $[-1,1]$ and M^a is a mirror transformation defined for $a \in \{-1,0,1\}$ as $M^{-1}(x) = -x$, $M^0(x) = x$ and $M^1(x) = 2 - x$, see [5].

Both are defined below.

```
> # First mirror-type estimator
> # dXsign stores A_xi(xj-xi) which will be pre-computed
> # ------------------------------------------------------------------------------
```

```
> kernel.estimator.mirror1 <- function(dXsign,Y,id.index,h,order,kernel){
+   n <- nrow(Y)
+   K <- 1
+   n.index <- length(id.index)
+   for (i in 1:n.index){
+     K <- K * do.call(kernel,list(dXsign[[id.index[i]]],h=h[i],order))
+   }
+   diag(K) <- 0
+   fhat <- matrix(pmax(rowSums(K),1e-5),n,1)
+   mhat <- K%*%Y / fhat
+   mhat[is.na(mhat)] <- mean(Y)
+   return(list(mhat=mhat,fhat=fhat/(n-1)))
+ }

> # Second mirror-type estimator
> # mX stores all observations, with each column containing all mirror transformations
> # except the one for the corresponding row
> # mX is precomputed with function mirrorX.
> # ------------------------------------------------------------------------------
> remove_diag <- function(x){
+   n <- nrow(x)
+   return(t(matrix(t(x)[-seq(1,n^2,n+1)], n-1, n)))
+ }
> mirror_base <- function(x,a){
+   return(2*(a>0)+(1-2*a^2)*x)
+ }
> mirrorX <- function(X){
+   d <- ncol(X)
+   mX <- vector('list',d)
+   ll <- lapply(1:d,function(i) c(-1,0,1))
+   a_combi <- expand.grid(ll)
+   n_combi <- nrow(a_combi)
+   for(i in 1:d){
+     mXtemp <- NULL
+     for (j in 1:n_combi){
+       mXtemp <- cbind(mXtemp,remove_diag(as.matrix(outer(X[,i],
+                              mirror_base(X[,i],a_combi[j,i]), "-"))))
+     }
+     mX[[i]] <- mXtemp
+   }
+   return(mX)
+ }
> kernel.estimator.mirror2 <- function(mX,Y,id.index,h,order){
+   n <- nrow(Y)
+   K <- 1
+   n.index <- length(id.index)
```

```
+   for (i in 1:n.index){
+     K <- K * kepach_norm(mX[[id.index[i]]],h=h[i],order)
+   }
+   fhat <- matrix(rowSums(K),n,1)
+   mhat <- sapply(1:n,function(row) sum(K[row,]*Y[-row])) / fhat
+   mhat[is.na(mhat)] <- mean(Y)
+   return(list(mhat=mhat,fhat=fhat/(n-1)))
+ }
```

2.2.2 Test case

Now we evaluate these estimators on the Bratley function defined by:

$$g_{\text{Bratley}}(X_1, \ldots, X_p) = \sum_{i=1}^{p} (-1)^i \prod_{j=1}^{i} V_j, \tag{7}$$

with $X_i \sim \mathcal{U}[0,1]$ i.i.d. with no noise.

```
> d <- 1; n <- 500
> lb <- 0; ub <- 1
> bratley.fun.vec <- function(X,lb,ub){
+   d <- ncol(X)
+   X <- (X-lb)/(ub-lb)
+   Y <- 0
+   for (i in 1:d){
+     Y <- Y + (-1)^i * apply(X[,1:i,drop=FALSE],1,prod)
+   }
+   return(matrix(Y,ncol=1))
+ }
> X <- lb+(ub-lb)*matrix(runif(n*d),n,d)
> Y <- bratley.fun.vec(X,lb,ub)
> # We precompute matrices dXsign and mX
> dX <- vector('list',d)
> for(i in 1:d){
+   dX[[i]] <- as.matrix(outer(X[,i], X[,i], "-"))
+ }
> dXsign <- vector('list',d)
> for (i in 1:d){
+   Xsign <- sign(0.5*(lb+ub)-X[,i])
+   Xsign <- matrix(Xsign,n,n,byrow=TRUE)
+   dXsign[[i]] <- Xsign*dX[[i]]
+ }
> mX <- mirrorX(X)
```

The only parameter which needs to be tuned is the bandwidth h. Here, we will consider a grid of evenly-spaced values on a logarithmic scale, and compute the leave-one-out mean square error for each of them.

We clearly see a very high numerical instability for the first estimator with kernels supported on $[0, 1]$, even on a simple regression example in dimension 1.

References

[1] Karine Bertin, Nicolas Klutchnikoff, Jose R. Léon, and Clémentine Prieur. Adaptive density estimation on bounded domains under mixing conditions. *Electronic Journal of Statistics*, 14(1):2198 – 2237, 2020.

[2] Fabienne Comte. *Nonparametric estimation. Master and Research.* Paris: Spartacus IDH, 2017.

[3] Bruce E Hansen. Exact mean integrated squared error of higher order kernel estimators. *Econometric Theory*, 21(6):1031–1057, 2005.

[4] Gérard Kerkyacharian, Oleg Lepski, and Dominique Picard. Nonlinear estimation in anisotropic multi-index denoising. *Probability theory and related fields*, 121(2):137–170, 2001.

[5] Louis Pujol. Nonparametric estimation of a multivariate density under kullback-leibler loss with ISDE. *arXiv preprint arXiv:2205.03199*, 2022.

```
> # Leave-one-out function
> fnloo <- function(exph,order,kernel){
+   if (kernel=="kepach_norm"){
+     mm <- kernel.estimator.mirror2(mX,Y,1:d,10^exph,order)
+   }else{
+     mm <- kernel.estimator.mirror1(dXsign,Y,1:d,10^exph,order,kernel)
+   }
+   return(sqrt(sum((Y-mm$mhat)^2)))
+ }
> # Grid of h values
> ngrid <- 200
> gridexph <- seq(-3,1,length.out=ngrid)
> data_rmse <- NULL
> for (o in 1:norder){
+   for (k in 1:nkernels){
+     rmse <- sapply(gridexph, function(exph) fnloo(exph,order=order_to_test[o],kernels_to_t
+     data_rmse <- rbind(data_rmse,data.frame(h=10^gridexph,rmse=rmse,kernel=rep(kernels_to_
+   }
+ }
> ggplot(data_rmse, aes(x = h, y = rmse, color = kernel)) +
+   geom_line() +
+   facet_wrap(~order, scales = 'free') +
+   scale_x_continuous(trans='log10') +
+   scale_y_continuous(trans='log10') +
+   theme_minimal()
```

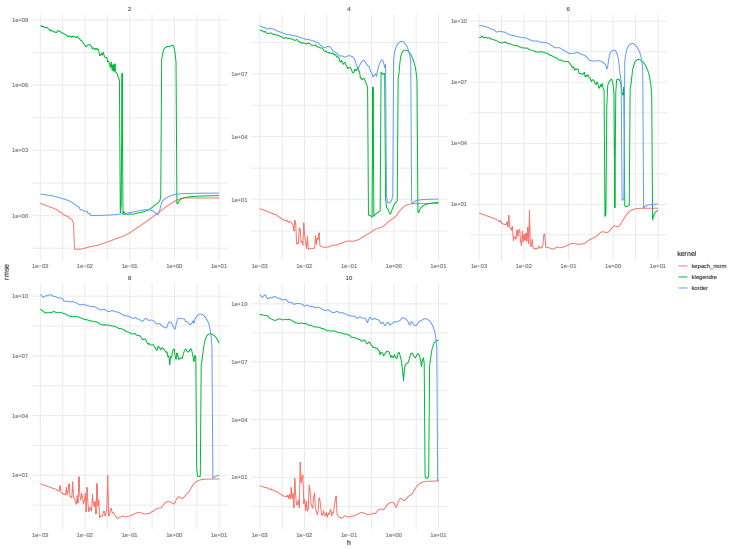

Figure 2: Leave-one-out error versus h for $\widehat{m}^1$ with symmetric kernel (2) (kepach_norm) and $\widehat{m}^2$ with non-symmetric kernels (3) (klegendre) and (4) (korder).