A technical document for *R* package *sgdpd*

Version 1.0.0

Akifumi Okuno, Ph.D.

January 21, 2024

1 Overview

This document is a software manual for our *R* package *sgdpd*, which robustly estimates the parameter θ in general (non-negative) functions f_θ, using density power divergence (Basu et al., 1998). Typically, we can estimate parametric density functions. This package is an *Rcpp* implementation of our paper (Okuno, 2023). Please see `https://github.com/oknakfm/sgdpd` for our github repository.

1.1 Score function to be minimized

Throughout this document, $f_\theta(z)$ represent general (non-negative) parametric functions, defined for the observed data vector $z \in \mathbb{R}^q$ and the parameter vector $\theta \in \mathbb{R}^s$ to be estimated. Herein, assume that data vectors $\{z_i\}_{i=1}^n$ are observed. Our *sgdpd* package simply estimates the parameter θ by minimizing the empirical *density-power cross entropy (score)*:

$$L_{\beta,n}(\theta) := -\frac{1}{\beta n}\sum_{i=1}^{n} f_\theta(z_i)^\beta + \frac{1}{1+\beta}\int f_\theta(z)^{1+\beta}\mathrm{d}z, \tag{1}$$

which was originally proposed by Basu et al. (1998). While we mainly consider estimating the parameter for probability density functions (p.d.f.), the score (1) can be used to estimate parameters for general non-negative functions (a.k.a. *unnormalized models*). See Kanamori and Fujisawa (2015) for details of the estimation of the unnormalized models using the score (1).

If f_θ represents the p.d.f., the score (1) can be seen as a robust generalization of the (negative) log-likelihood. Therefore, the parameter estimation using the score (1) can be regarded as a robust generalization of the maximum likelihood estimation. Please also see Basu et al. (1998); Kanamori and Fujisawa (2015); Okuno (2023) for more details of the score (1).

1.2 Optimization procedure

While Basu et al. (1998), Kanamori and Fujisawa (2015) showed that the general (non-negative) parametric functions f_θ can be estimated using the score (1), the integral term in (1) is hard to be computed in practice. It is known that the integral can be analytically expanded only for specific distributions: normal (Basu et al., 1998), exponential (Jones et al., 2001), generalized Pareto (Juárez and Schucany, 2004), Weibull (Basu et al., 2016), and generalized exponential (Hazra, 2022). Numerical integration is also used to finitely approximate the integral term, when considering normal-mixutre: (Fujisawa and Eguchi, 2006), Poisson: (Kawashima and Fujisawa, 2019), skew-normal (Nandy et al., 2022). However, minimizing the score (1) for general models is in general difficult.

To address this computational problem, this *sgdpd* package follows Okuno (2023), which proposed leveraging the stochastic optimization technique in line

with Robbins and Monro (1951). More specifically, this *sgdpd* package iteratively updates the parameter $\theta^{(t)}$ (at iteration t) by the stochastic gradient descent (SGD; Robbins and Monro, 1951):

$$\theta^{(t+1)} \leftarrow \theta^{(t)} - l^{(t)} \frac{g(\theta^{(t)}; \zeta^{(t)})}{\|g(\theta^{(t)}; \zeta^{(t)})\|_2}. \tag{2}$$

Here, $l^{(t)} \geq 0$ denotes the learning rate, and

$$g_t(\theta^{(t)}) = -\frac{1}{N}\sum_{j=1}^{N} f_\theta(z_{i_j^{(t)}})^\beta \frac{\partial \log f_\theta(z_{i_j^{(t)}})}{\partial \theta} + \frac{1}{M}\sum_{k=1}^{M} f_\theta(\xi_k^{(t)})^\beta \frac{\partial \log f_\theta(\xi_k^{(t)})}{\partial \theta} \frac{f_\theta(\xi_k^{(t)})}{\tilde{p}(\xi_k^{(t)})}, \tag{3}$$

represents the stochastic gradient[1]. In (3), in each step t, the index $\{i_1^{(t)}, i_2^{(t)}, \dots, i_N^{(t)}\}$ is uniformly sampled from $\{1, 2, \dots, n\}$ (with repetition in our implementation, to allow the setting $N > n$), and $\xi_k^{(t)}$ is randomly generated from the density function[2] $\tilde{p}(\xi)$. As the stochastic gradient (3) is an unbiased estimator of the gradient of the score (1), Ghadimi and Lan (2013) provides the theoretical evidence of the convergence of SGD (2) as also described in Okuno (2023) Proposition 2.

1.3 Installation and preparation

To install our *sgdpd* package, please enter and execute the following command in *R* commandline:

```
install.packages(
    "https://okuno.net/R-packages/sgdpd_1.0.0.zip",
    repos=NULL, type="win.binary")
```

After the installation, please call our *sgdpd* package by

```
library("sgdpd")
```

Then, it is ready to use our *sgdpd* package!

2 Technical details

In this section, details of three main functions *lr_scheduler()*, *sgdpd()* and *apply_f()* provided in our *sgdpd* package are described. Therein, arguments highlighted by *blue* are needed to be input by users, while those of *red* have default values.

[1] In our implementation, the gradient is computed with numerical differentiation.

[2] In our implementation, we employ the normal density whose (element-wise) mean and variance are compatible with the sample mean and variance of $\{z_i\}_{i=1}^n$.

2.1 *lr_scheduler()*

The function *lr_scheduler()* provides the learning rates $\{l^{(t)}\}_{t=1}^{T}$ used to compute the SGD (2). This scheduler adopts a cyclic learning rate schedule.

arguments: this function takes the following arguments

- *n_itr*: number of iterations (3000 by default).
- *lr0*: initial learning rate (0.1 by default).
- *gamma*: decay rate ($\in (0, 1)$, 0.1 by default).
- *decay_period*: period of decay ($\in \mathbb{N}$, 50 by default).
- *cyclic_period*: period of cycle ($\in \mathbb{N}$, 200 by default).

In our scheduler, the learning rate is started from *lr0* (i.e., $l^{(1)}$ =*lr0*). For each *decay_period* iterations, the learning rate is multiplied by *gamma*. To escape from the local optimum in the score (1), this scheduler employs a cyclic "reignition": for each *cyclic_period* iterations, the learning rate is reset (but the rate is multiplied by *gamma* compared to the the previous reignition). Figure (1) shows the scheduled learning rates with our default setting.

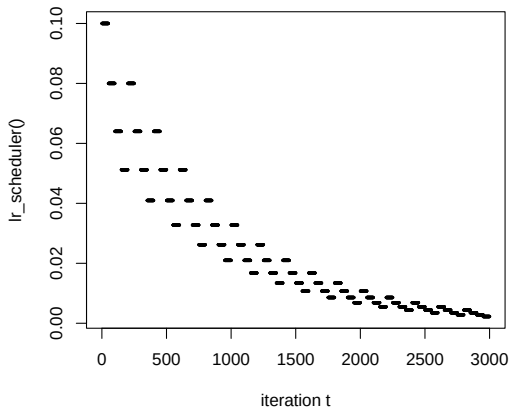

Figure 1: cyclic learning rate

2.2 *sgdpd()*

The function *sgdpd()* computes the minimizer $\hat{\theta}$ of the score (1) using SGD (2) equipped with the stochastic gradient (3).

arguments: this function takes the following arguments. Therein, *f*, *Z*, *theta0* are *needed to be input* while the remaining arguments have default values.

- *f*: non-negative parametric model $f = f(z, \theta)$ to be estimated (none by default; *needed to be input*).
- *Z*: $n \times q$ design matrix $Z = (z_1^\top, z_2^\top, \ldots, z_n^\top)^\top$. Even if $q = 1$, *Z* should be in the form of $n \times 1$ matrix. Vector of length n is not allowed. (none by default; *needed to be input*).
- *theta0*: initial parameter (none by default; *needed to be input*).
- *lr*: a vector representing learning rate schedule[3]. you can input the vector returned from *lr_scheduler()*. If you input an real value (but not the scheduled rates as a vector), *lr_scheduler()* is called internally with *lr0=lr*. (0.1 by default).
- *positives*: parameter index whose estimated value should be positive, e.g., standard deviation of normal distribution. 0 indicates no positive parameter (0 by default).
- *conditions*: outcome index in the design matrix *Z*, if you consider regression problem. 0 indicates no conditioned outcome (0 by default).
- *exponent*: exponent parameter β in the score (1). For stable computation, we strongly recommend to use values in $(0.1, 1]$. (0.1 by default)
- *N*, *M*: subsample sizes to compute the 1st/2nd terms in stochastic gradient (3). (20 and 2 for *N* and *M*, respectively, by default).
- *showProgress*: if *TRUE*, the optimization progress is shown, and not otherwise (default: *TRUE*).
- *h*: bandwidth for numerical differentiation (10^{-10} by default).
- *log_interval*: interval to trace the parameter update. If 0, the update is not going to be traced (10 by default).

returned values in the form of *List* with the following elements:

- *$theta*: estimated parameter.
- *$log*: traced log of the update parameters (if *log_interval* $\neq 0$).
- *$setup*: arguments input (this element contains further descendant elements $positives, $conditions, $exponent, $N, $M, $h, $lr, $log_interval).

Note that this *sgdpd* function does not provide any convergence guarantee. Please carefully use *sgdpd* and check the convergence manually using the design matrix *Z*.

[3] Number of iterations is automatically determined by the length of the *lr* vector.

2.3 *apply_f()*

This function applies the user-specified function $f(z, \theta)$ for all the entries in the design matrix Z and (one) parameter vector θ. arguments *f*, *Z* are the same as those in the function *sgdpd()*.

3 Execution examples

3.1 Normal density estimation

In this section, we start by estimating the normal density. In this experiment, we generate n = 1000 samples from the normal distribution $\mathcal{N}(3, 3^2)$ mean 3 and the standard deviation 3. Started from the initial parameter θ = (0, 1), we estimate the parameter θ = (μ, σ) using our *sgdpd()* function. See the code in Listing 1 and the result is shown in Figure 2.

```
## dataset generation (normal)
Z = as.matrix(rnorm(n=1000, mean=3, sd=3), 1000, 1)

## model specification (normal density)
f_n <- function(z, theta) dnorm(x=z, mean=theta[1], sd=theta[2])

## parameter estimation
par_n = sgdpd(f=f_n, Z=Z, lr=0.1, theta0=c(0,2),
    positives=c(2), exponent=0.2)
```

Listing 1: Normal density estimation

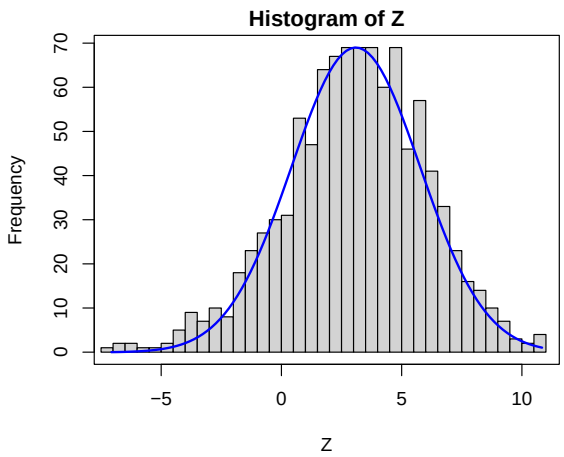

Figure 2: Estimated density with the histogram of Z.

3.2 Skew-normal mixture density estimation

To demonstrate our optimizer *sgdpd* we estimate the mixture of the skew-normal densities in this section. We first generate $n = 700$ samples from a skew-normal distribution $\mathcal{S}(-2, 1^2, 4)$ and $n = 300$ samples from $\mathcal{S}(3, 0.5^2, 1/3)$. Using the combined $n = 1000$ samples, we estimate the mixture of the skew-normal density. Technically, the mixture rate ($\in [0,1]$) is specified by the sigmoid function $\rho(z) = 1/\{1+\exp(-z)\}$ (provided in our *sgdpd* package as a function *sigmoid()*). See the code in Listing 2 and the result is shown in Figure 3.

```
## package for skew-normal density/random number generation
library("fGarch")

## dataset generation (skew-normals mixtured with the rate 0.7)
Z = as.matrix(append(
    rsnorm(n=700, mean=-2, sd=1, xi=4),
    rsnorm(n=300, mean=3, sd=0.5, xi=1/3)),
    1000, 1)

## model specification (skew normal mixture density)
f_snm <- function(z, theta){
  alpha = sigmoid(theta[7])
  p1 = dsnorm(x=z, mean=theta[1], sd=theta[2], xi=theta[3])
  p2 = dsnorm(x=z, mean=theta[4], sd=theta[5], xi=theta[6])
  alpha * p1 + (1-alpha) * p2
}

## parameter estimation
par_snm = sgdpd(f=f_snm, Z=Z, lr=0.1,
    theta0=c(-1,1,1,1,1,1,0),
    positives=c(2,3,5,6), exponent=0.2)
```

Listing 2: Skew-normal mixture density estimation

3.3 Polynomial regression

Our optimizer *sgdpd* also can be used to conduct a regression analysis. We first generate $n = 1000$ samples from a $d = 3$ dimensional polynomial regression (with a normal error) and 5% outliers. Using the contaminated dataset, we first compute the maximum likelihood estimator (MLE). Starting from this MLE, we minimize the score (2) with the probabilistic model $f_\theta(z) := \phi(\{y - r_\theta(x)\}^2/2\sigma^2)$, where $z = (x, y)$ represents the pair of the covariate x and the outcome y, $r_\theta(x) = \theta_1 x + \cdots + \theta_d x^d + \theta_{d+1}$ is the polynomial regression model and ϕ represents the standard normal density. As the covariate x is stored in the 1st column in the design matrix Z, we use the

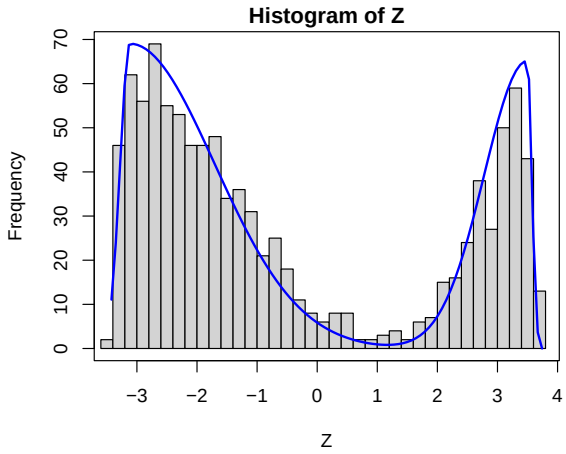

Figure 3: Estimated density with the histogram of Z.

option *conditions=c(1)*. See the code in Listing 3 and the result is shown in Figure 4. The *sgdpd* estimator is shown as the blue line, while the MLE is shown as the red line.

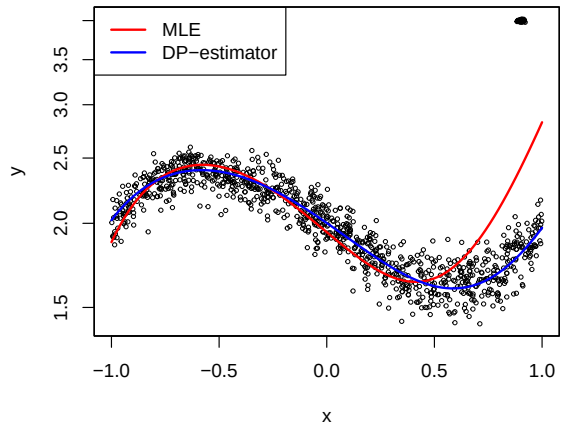

Figure 4: Polynomial regression.

4 Some notes on stochastic optimization

(a) The convergence of Stochastic Gradient Descent (SGD) does not necessarily imply convergence to even a local minimum. This behavior can be influenced by the learning rate settings:

- With a *too small* learning rate: When the learning rate is excessively small, SGD may fail to reach (even a) local minima. This is because the step size that can move at each iteration is too limited, hindering its ability to progress adequately toward the minimum.

```r
## underlying regression model
d = 3
theta = seq(-1, 2, length.out = (d+1))
regressor <- function(x, theta){
    sum(x^(1:d) * theta[1:d]) + theta[d+1]
}

## dataset generation (polynomial regression)
x = runif(n=1000, min=-1, max=1)
mu = sapply(x, function(z) regressor(z, theta))
y = rnorm(n=1000, mean=mu, sd=0.1)
x = append(x, rnorm(n=50, mean=0.9, sd=0.01)) #outliers
y = append(y, rnorm(n=50, mean=4, sd=0.01)) #outliers
Z = cbind(x,y)
X = NULL; for(j in 1:d) X = cbind(X,x^j); X=cbind(X, 1)

## maximum likelihood estimator
theta0 = as.vector(solve(t(X) %*% X) %*% t(X) %*% y)

## model specification (normal conditional density)
f_reg <- function(z, theta){
    dnorm(x=z[2], mean=regressor(z[1], theta), sd=0.1)
}

## parameter estimation
par_reg = sgdpd(f=f_reg, Z=Z, lr=0.1, theta0=theta0,
    conditions=c(1), exponent=0.2)
```

Listing 3: Polynomial regression

- With a *too large* learning rate: Conversely, if the learning rate is set too high, there is a risk that SGD parameters may diverge significantly. In such cases, instead of approaching a minimum, the parameters can fluctuate or move far away from the desired convergence point.

Therefore, it is essential to establish an appropriate learning rate schedule to ensure effective convergence.

(b) Employing an excessively small $N \in \mathbb{N}$ in SGD can lead to a significant influence from outliers. Consequently, using such a small $\mathbb{N}$, particularly when $N \leq 5$, fails to leverage the full robustness offered by density power score (1). To achieve more robust estimation, it is preferable to select $\mathbb{N} \geq 20$.

(c) The computational cost of the *sgdpd()* function primarily depends on the subsample sizes $N, M \in \mathbb{N}$, rather than on the overall sample size $n \in \mathbb{N}$. While

$\mathbb{M}$ can be set to a small value, employing a small $\mathbb{N}$ in *sgdpd()* leads to the issues previously discussed in (b).

(d) The convergence of the algorithm is significantly influenced by the choice of the proposal distribution $\tilde{p}$, which is utilized in computing the stochastic gradient as defined in (3). In our implementation, we have employed a normal distribution, with its element-wise mean and variance aligned with the empirical average and variance of the design matrix Z. However, it is important to note that in the context of regression analysis, where outcomes are fixed and the focus is on conditional density, employing a non-degenerate normal distribution may result in markedly inefficient estimation. To address this, we suggest utilizing the *conditions* option in the *sgdpd()* function. This feature allows users to specify which columns of the design matrix are fixed, thereby enhancing the efficiency and accuracy of the estimation process.

References

Basu, A., Harris, I. R., Hjort, N. L., and Jones, M. C. (1998). Robust and efficient estimation by minimising a density power divergence. *Biometrika*, 85(3):549–559.

Basu, A., Mandal, A., Martin, N., and Pardo, L. (2016). Generalized wald-type tests based on minimum density power divergence estimators. *Statistics*, 50(1):1–26.

Fujisawa, H. and Eguchi, S. (2006). Robust estimation in the normal mixture model. *Journal of Statistical Planning and Inference*, 136(11):3989–4011.

Ghadimi, S. and Lan, G. (2013). Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM Journal on Optimization*, 23(4):2341–2368.

Hazra, A. (2022). Minimum density power divergence estimation for the generalized exponential distribution. *arXiv preprint arXiv:2206.08216*.

Jones, M. C., Hjort, N. L., Harris, I. R., and Basu, A. (2001). A comparison of related density-based minimum divergence estimators. *Biometrika*, 88(3):865–873.

Juárez, S. F. and Schucany, W. R. (2004). Robust and efficient estimation for the generalized pareto distribution. *Extremes*, 7:237–251.

Kanamori, T. and Fujisawa, H. (2015). Robust estimation under heavy contamination using unnormalized models. *Biometrika*, 102(3):559–572.

Kawashima, T. and Fujisawa, H. (2019). Robust and sparse regression in generalized linear model by stochastic optimization. *Japanese Journal of Statistics and Data Science*, 2(2):465–489.

Nandy, A., Basu, A., and Ghosh, A. (2022). Robust inference for skewed data in health sciences. *Journal of Applied Statistics*, 49(8):2093–2123.

Okuno, A. (2023). Minimizing robust density power-based divergences for general parametric density models. *arXiv preprint arXiv:2307.05251*.

Robbins, H. and Monro, S. (1951). A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400 – 407.