cpda and gpda: Efficient Computation for Probability Density Approximation

Yi-Shin Lin
University of Tasmania

Andrew Heathcote
University of Tasmania

William R. Holmes
Vanderbilt University

Abstract

Probability density approximation (PDA) efficiently calculates likelihood even when their analytic functions are unavailable. It allows researchers to model compuationally complex biological processes, which in the past could only be approached by overly simplified models. It is however computationally demanding. We implement two R packges, **cpda** and **gpda**, using Armadillo C++ and CUDA C to provide a practical and efficient solution for Bayesian computation of cognitive models. Both packages harness the multi-thread nature of modern computational processing units, enabling parallel computations with a dozens of cores of central processing unit (CPU) and thoudands threads of graphics processing unit (GPU). **cpda** resolves the bottleneck when few fast CPU cores are efficient to find optimal solutions, such running multiple Markov chains in parallel, whereas **gpda** reduces the computation times when large numbers (>1e5) of Monte Carlo simulations is required to approximate probability densities. We conclude with three practical exmaples as a road map for the application in cognitive models.

Keywords: R, C++, GPU, kernel density estimate, Markov Chain Monte Carlo, Fourier-based PDA.

1. Introduction

Simulation-based algorithms recently find a new application in finding likelihood functions (Sisson and Fan 2010). This application is especially useful when specifying a likelihood function analytically is unlikely or when evaluating it is computationally prohibitive, a situation arises sometimes in cognitive and often in neuro-cognitive models. These algorithms are often referred to as *likelihood-free computation* or *approximate Bayesian computation* (ABC) (Sisson and Fan 2010). Probability density approximation (PDA) is one such method. Unlike other likelihood-free methods, it circumvents the difficulty of identifying sufficient summary statistics, a set of numbers supplying as much information for data as unknown model parameters (Turner and Sederberg 2014) by processing the data directly. This is crucial, because it is often unlikely to know whether a set of summary statistics is sufficient when its likelihood function is unavailable.

Because of its non-parameteric nature, PDA however suffers two computation bottlenecks. First is calculating kernel density estimates (KDE) for every data point and second is conducting Monte Carlo (MC) simulations. The computation problem is aggravated when applying PDA in Bayesian modeling, because KDE and MC simulations are conducted iteratively in

multiple Markov chains. More importantly, the bottlenecks not only incur computation burdens, but also closely relate to computation errors, which have to be minimized (Turner and Sederberg 2014). Hence an efficient computation method is critical not just for reducing time, but also for minimizing errors.

Holmes (2015) derived a fast Fourier-based algorithm, named resampled PDA (R-PDA), to reduce computation steps, and thereby to mitigate the first bottleneck. In short, R-PDA transfers KDE to spectrum domains, rendering convolution to multiplication operations. R-PDA thereby greatly decreases computation costs and potential errors in the first bottleneck. The second bottleneck however is largely unresolved. Here we present software for efficient PDA and R-PDA computations (Holmes, 2015; Turner & Sederberg, 2014), and resolve the second bottleneck by applying three recent computational techniques.

First, we code PDA in Armadillo C++, a high efficient C++ library for linear algebra (Sanderson & Curtin, 2016). Second, we implement Open MP to harness multiple-core CPU, as it has become commonplace with a regular personal computer (PC) equipping at least 4 cores in a central process unit (CPU). Third, we implement PDA also in Compute Unified Device Architecture, CUDA, a programming model recognizing the structure of graphics processing unit (GPU); hence, **gpda**[1] package allows a regular PC user to enjoy computation power of thousands GPU cores.

To ease the installation and usage of the packages, we take advantage of the infrastructure of the Comprehensive R Archive Network (CRAN). Specifically, **cpda** and **gpda** conform CRAN standard and available at http://CRAN.R-project.org/package=cpda and http://CRAN.R-project.org/package=gpda with numerous examples in their help pages. The user can easily install the packages using the install.packages function in R or GUI interface, such as the one provided by R GUI or *RStudio*.

This vignette corresponds to the paper published in the *Journal of Statistical Software*. It is currently still identical to the published paper. Over time, this vignette version may receive minor updates. This version corresponds to **cpda** version 0.0.2.1 and was typeset on March 29, 2017.

1.1. Four Simple Examples

We illustrate four examples, using **cpda** to construct simulated probability density functions (SPDFs). First example reconstructs the starnard Gaussian distribution. Because the Gaussian distribution has an analytic probability density function (PDF), we can easily verify whether the SPDF generated by **cpda** successfully approximates the analytic PDF.

```
require(cpda)
n      <- 1e5                        ## Number of simulations
x      <- seq(-3,3, length.out=100)  ## Support
xlabel <- "Observations"
ylabel <- "Density"

## Simulate Gaussian distribution -----------
sam  <- rnorm(n)                     ## Monte Carlo simulations
den1 <- cpda::logLik_pw(x, sam)      ## PDA
```

[1] gpda stands for GPGPU-based PDA. Similarly cpda stands for C++-based PDA

```
den2 <- logLik_fft2(x, sam)          ## R-PDA
den3 <- dnorm(x)                     ## Analytic Gaussian likelihood

## Verify the three methods converge to the same PDF
png(file="doc/gaussian.png", width=800, height=600)
par(mar=c(4,5.3,0.82,1))
plot(x, exp(den1), type="l", lty="dotted", xlab=xlabel, ylab=ylabel, cex.lab=3,
  cex.axis=1.5, lwd=3)
lines(x, exp(den2[,2]), col="blue", lty="dashed", lwd=2)
lines(x, den3, col="red", lwd=2)
dev.off()
```

`logLik_pw` returns point-wise log-likelihood. It uses PDA to calculate log-likelihood for each observation. It takes about 0.323 second to get SPDF with 100,000 simulations. Using R-PDA (Holmes, 2015), we can further reduce computation times. This is useful for the case where observations share the same parameters.

```
system.time(den2 <- logLik_fft2(x, sam)
##  user  system elapsed
## 0.124   0.000   0.122

head(den2)
##           [,1]      [,2]
## [1,] -3.000000 -5.420678
## [2,] -2.939394 -5.240477
## [3,] -2.878788 -5.048910
## [4,] -2.818182 -4.850126
## [5,] -2.757576 -4.671268
## [6,] -2.696970 -4.498645
```

`logLik_fft2` takes only 0.122 second return a SPDF. It is about 2.6 times faster than `logLik_pw`. It returns a matrix with the first column storing the data (ie `x`) and the second column storing log-likelihood.

The three density solutions, `logLik_fft2`, `logLik_pw`, and analytic Gaussian function, match one another, as shown by the overlapping density curves (Figure 1).

PDA is a general method, working also for other distributions. The following example demonstrates approximating the probabilities of an exponential modified Gaussian (ex-Gaussian) distribution, a standard distribution being used often to describe response time data (Dawson 1988). Again, becasue ex-Gaussian distribution has an analytic form, we can verfiy resampled PDA successfully recover likelihood.

```
## Simulate ex-Gaussian distribution -----------
## We used the rexGAUS function in gamlss.dist package, downloaded from
## https://cran.r-project.org/web/packages/gamlss.dist/index.html
## Simulate ex-Gaussian distribution -----------
n    <- 1e5
```

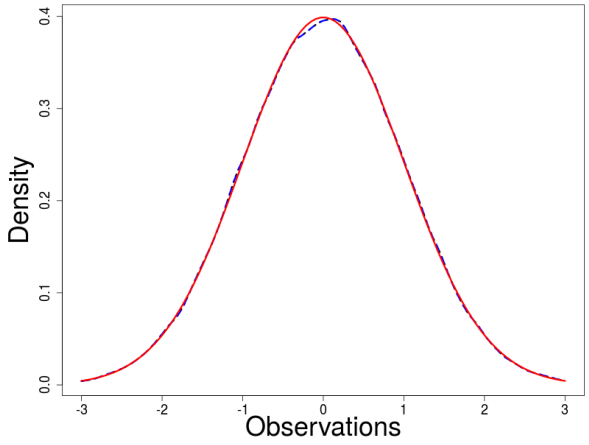

Figure 1: Three solutions for the Gaussian density function. The black, blue and red curves are on top of one another.

```
sam  <- gamlss.dist::rexGAUS(n, mu=0, sigma=1, nu=1)
x    <- seq(-4, 4, length.out=100) ## Support
den1 <- cpda::logLik_pw(x, sam)
den2 <- logLik_fft2(x, sam)
den3 <- gamlss.dist::dexGAUS(x, mu=0, sigma=1, nu=1)

xlabel <- "Observations"
ylabel <- "Density"


png(file="doc/exG.png", width=800, height=600)
par(mar=c(4,5.3,0.82,1))
plot(x, exp(den1), col="grey", type="l", lty="dotted", xlab=xlabel, ylab=ylabel,
  cex.lab=3, cex.axis=1.5, lwd=3)
lines(x, exp(den2[,2]), lty="dashed", lwd=3)
lines(x, den3, col="red", lwd=3)
dev.off()
```

Figure 2 shows PDA, R-PDA and analytic ex-Gaussian density function return almost identical likelihood.

The next example demonstrates an interesting application, using a slightly complicated example. We use PDA to recover probabilities from a regression model, $y = ax + b + \epsilon$. This is the case where the parameters for each data point differ.

Because each data point is sampled from a Gaussian distribution with a different mean, we need to conduct simulations separately for each of them. We can use R's sapply function or even better C++ iterator to speed up the compuation. To make code more readable, we wrote it in a simple for loop here.

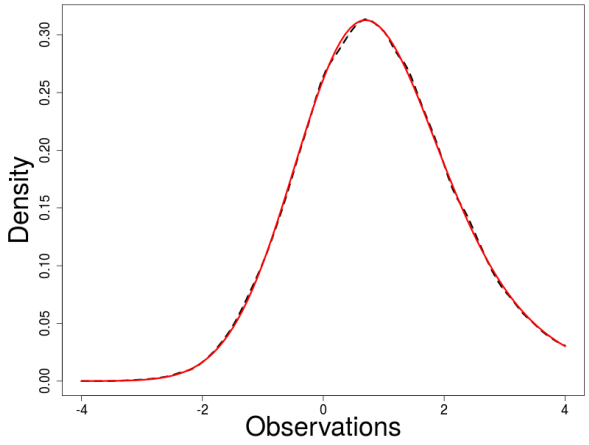

Figure 2: Approximating exponential modified Gaussian distribution.

```
n      <- 1e5
x      <- seq(-3, 3, length.out=100) ## Support
xlabel <- "x"
ylabel <- "Density"
theta  <- c(a=7.5, b=3.5, s=5)       ## slope, intercept and sigma
y      <- rnorm(length(x), mean=theta[2]+theta[1]*x, sd=theta[3])
dat    <- cbind(x, y)
den1   <- numeric(length(x))         ## a container to store likelihood

## Even we simulate for each data point, it takes cpda only 2 seconds with
## 100 x 1e5 simulations
##   user  system elapsed
##  2.084   0.004   2.087
for(i in 1:length(x)) {
  sam     <- rnorm(n, theta[2]+theta[1]*x[i], theta[3])
  den1[i] <- cpda::logLik_pw(x[i], sam)
}

## Get analytic likelihood
den2 <- dnorm(x, theta[2]+theta[1]*x, theta[3])


png(file="doc/regression.png", width=800, height=600)
par(mfrow=c(1,2))
par(mar=c(4,5.3,0.82,1))
plot(x,y, cex.lab=3, cex.axis=1.5, lwd=3)
plot(x, exp(den1),  col="grey", type="l", lty="dotted", xlab=xlabel,
  ylab=ylabel, cex.lab=3, cex.axis=1.5, lwd=3)
lines(x, den2, lwd=3, lty="dashed")
dev.off()
```

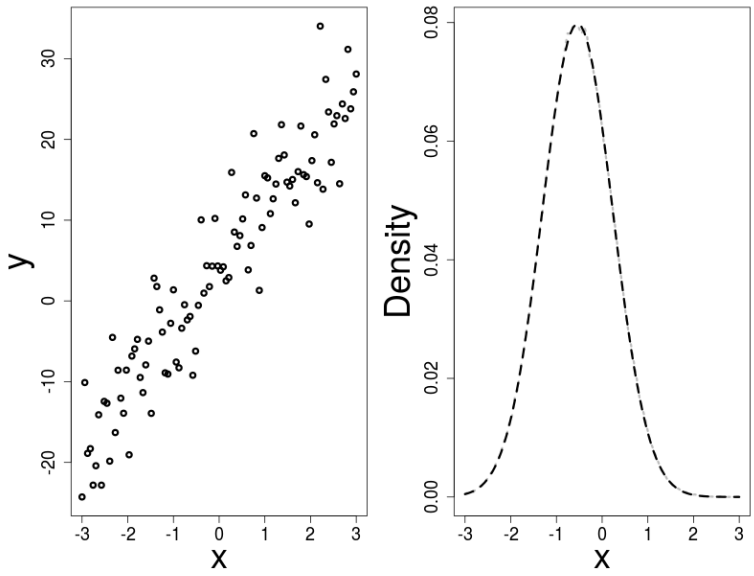

Figure 3: Regression data and their likelihood approximation. The approximated solution is almost identical to the analytic solution, so the grey and drak lines are hardly distinguishable.

The last example, linear ballistic accumulation (LBA) model, is a simplified evidence accumulation model (Brown and Heathcote 2008). The model accounts for the data of choice response time (RT), often collected in a psychological experiment where a participant make a speedy choice for multiple alternatives. In the case of two-choice task, a data point consists of a response time and a choice (e.g., choice 1 made by .800 second).

```
## Simulate linear ballistic accumulator distribution -----------
## We use the "rLBA" function in rtdists to generate "empirical" data, which
## have 1,000 choice RT responses.
## rtdists can be downloaded at:
## https://cran.r-project.org/web/packages/rtdists/index.html
y <- rtdists::rLBA(1e3, A=.5, b=1, t0=.25, mean_v=c(2.4, 1.2), sd_v=c(1, 0.6))
y1 <- sort(y[y$response==1, "rt"]) ## rt1
y2 <- sort(y[y$response==2, "rt"]) ## rt2
```

The empirical RT distributions for choice 1 and choice 2 skew towards positive side, where fast responses are truncated and slow responses are infrequent. Because the analytic LBA PDF is known, we can calculate it directly using the equation provided in (Brown and Heathcote 2008).

```
den0 <- rtdists::dLBA(y$rt, y$response, A=.5, b=1, t0=.25, mean_v=c(2.4, 1.2),
  sd_v=c(1, .6))
df0  <- cbind(y, den0)
```

```
df1  <- df0[df0[,2]==1,]
df2  <- df0[df0[,2]==2,]
den1 <- df1[order(df1[,1]),3]
den2 <- df2[order(df2[,1]),3]
```

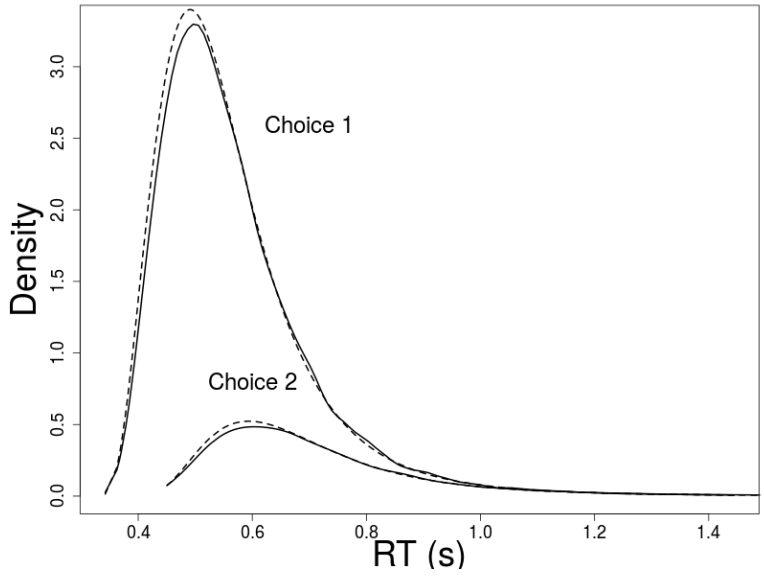

Figure 4: The solid and dashed lines show empirical and simulated probability density function, respectively. The empirical data were generated with the following parameters: upper bound for the starting evidence, $A = .5$, response threshold, $b = 1$, non-decisoin time $t0 = .25$, mean drift rates for accumulator *1* and accumulator *2*, $(2.4, 1.2)$, and their standard deviations, $(1, 0.6)$.

Even though there is an analytic solution for its probability density function (Brown and Heathcote 2008), we assume for now the density function is not available. Therefore, we used PDA to find SPDF as an alternative solution for the density function. A cannoical LBA model samples a starting value for sensory evidence stochastically from a uniform distribution with bounds, 0 and A and a rate of evidence accumulation (i.e., drift rate) from a Gaussian distribution. To eliminate the possibility of negative drift rate, we sampled drift rates from a truncated normal distribution. This LBA theory enables us to conduct Monte Carlo simulations of LBA model without knowing its PDF.

```
n    <- 1e5          ## the size of simulated sample
pVec <- c(A1=1, A2=1, b1=.5, b2=.5, v1=2.4, v2=1.2, sv1=1, sv2=.6,
          t01=.25, t02=.25)

## Monte Carlo simulations for choice 1 and 2
samp  <- cpda::rlba(n, pVec)
samp1 <- samp[samp[,2]==1, 1]
samp2 <- samp[samp[,2]==2, 1]
```

```
## Use R-PDA to estimate densities. Because each choice is defective,
## the user needs to indicate the total number of simulated samples.
system.time(fft1 <- cpda::logLik_fft2(y1, samp1, n=n))
system.time(fft2 <- cpda::logLik_fft2(y2, samp2, n=n))

## Use PDA to estimate densities. Again considering defective.
system.time(fft3 <- cpda::logLik_pw(y1, samp1, n=n))
system.time(fft4 <- cpda::logLik_pw(y2, samp2, n=n))

## Use Open MP to speed up piecewise computation
system.time(fft5 <- cpda::logLik_pw(y1, samp1, n=n, parallel=T))
system.time(fft6 <- cpda::logLik_pw(y2, samp2, n=n, parallel=T))

## ny1=828; nsamp1=81164
## 0.024   0.004   0.026   ## logLik_fft
## 1.568   0.004   1.568   ## logLik_pw
## 10.220  0.000   0.905   ## logLik_pw with parallel=T 12 cores


xlabel <- "RT (s)"
ylabel <- "Density"

par(mar=c(4,5.3,0.82,1))
plot(y1, exp(fft1[,2]), type="l", xlab=xlabel, ylab=ylabel, cex.lab=3,
     cex.axis=1.5, lwd=2)
lines(y2, exp(fft2[,2]), lwd=2)
lines(y1, exp(fft3), lty="longdash", lwd=2)
lines(y2, exp(fft4), lty="longdash", lwd=2)
lines(y1, exp(fft5), lty="dotted", lwd=2)
lines(y2, exp(fft6), lty="dotted", lwd=2)

lines(y1, den1, lty="dashed", lwd=2)
lines(y2, den2, lty="dashed", lwd=2)

text(0.7, 2.6, "Choice 1", cex=2)
text(.85,  0.8, "Choice 2", cex=2)
dev.off()
```

SPDFs for choice 1 and choice 2 are almost indistinguishable from the analytic PDF (Figure 4).

1.2. Probability Density Approximation

In this section, we gave an overview of PDA and its enhancement, resampled PDA. Interested readers can find an exposition in Turner and Sederberg (2014) for PDA and further elaboration about how resampled PDA improves PDA in Holmes (2015). Good reviews of ABC can be found in Sisson and Fan (2010) and Beaumont (2010).

Bayesian inference derives the postioer distribution defined by a set of model parameters via multiplying a *prior* distribution, $\pi(\theta)$, by a model likelihood function, $\pi(y|\theta)$. This is described by the well-known Bayes' theorem:

$$\pi(\theta|y) = \frac{\pi(y|\theta)\pi(\theta)}{\int \pi(y|\theta)\pi(\theta)d\theta} \tag{1}$$

More often, Bayes's theorem is processed proportionally, because the denominator in the equation 1 is a constant.

$$\pi(\theta|y) \propto \pi(y|\theta)\pi(\theta) \tag{2}$$

On the left hand side of the equation 2, $\pi(\theta|y)$ stands for the posterior probability densitiy function for a set of model parameters, θ, conditional on a given data set y. Both θ and y can be a scalar or a vector. The first term on the right hand side is the likelihood function conditional on the θ. The second term is the prior probability density function for the θ, which often is chosen arbitrarily.

In a standard Bayesian inference, one first proposes a set of parameters, often denoted θ^*. There are a number of ways to propose parameters. A traditional method is to sample from a jumping distribution (Gelman 2014), but other elaborated methods are also available, such as No-U-turn sampler (Hoffman and Gelman 2014) and DE-MCMC sampler (Braak 2006; Turner *et al.* 2013). Together with an empirical data set, one can derive a posterior probability density based on proposal parameters simply by calculating the right hand side of the equation 2. This proposal density is then compared to a reference density, for example in a Markov chain, the density in a previous iteration. This comparison is usually a ratio of the proposal density to the reference density, so the constant term in equation 1 can be safely ignored. The ratio, $\frac{\pi(\theta^*|y)}{\pi(\theta^{i-1}|y)}$, is then subjected to an accept-reject, such as Metropolis, algorithm to decide whether the proposed or the reference paraemters is more probable in light of given data, and thereby accepted (or rejected, if less probable).

The problem PDA, and generally ABC, tackle is the situation when the analytic form of $\pi(y|\theta)$, is difficult to derive. When the analytic likelihood is unavailable, a standard Bayesian inference becomes impossible to process.

PDA replaces $\pi(y|\theta)$ with a weighting function, $\pi(y|x,\theta)$, multiplying a SPDF, $\pi(x|\theta)$. x represents simulated data. This renders the posterior probability density function:

$$\hat{\pi}(\theta|y) \propto \pi(y|x,\theta)\pi(x|\theta)\pi(\theta) \tag{3}$$

So when the analytic likelihood function, $\pi(y|\theta)$, is unavailable, one can still approximate probability densities, namely SPDF, via Monte Carlo simulations as long as a theoretical model can be prescribed, such as the LBA example in previous section. In PDA, the weighting function, $\pi(y|x,\theta)$, could be a standard Gaussian, a Epanechnikov (Holmes 2015; Turner and Sederberg 2014), or any other kernel function:

$$K_h(z) = \frac{1}{h}K(\frac{z}{h}) \tag{4}$$

z is the discrepancy between an empirical datum and a simulate datum ($y - x_j$ in the following equation). K is the kernel funciton. Replacing the smoothing kernel into equation 3, we

can then derive the equation of the weighting function mulitplying SPDF (namely, weighted SPDF):

$$\hat{f}(x) = \frac{1}{N_s} \sum_{j=1}^{N_s} K_h(y - x_j) \tag{5}$$

N_s is the number of MC simulations, used to construct SPDF. h is the bandwidth of the kernel. Bandwidth determines the degree to which one wishes to smooth the kernel (Silverman 1986). The larger the bandwidth, the more smoothing is appled on simulated data.

R-PDA harnesses the fact that the right hand side of equation 5 equals to convolution operations of SPDF and the kernel function, and convolutions are computationally intensive. It is a common practice in signal processing to transform signals in the time domain to the frequency domain.

$$\hat{f}(x) = \pi(x|\theta) \star K_h(z) \tag{6}$$

R-PDA applies exactly the same technique, transforming both SPDF and kernel function to the frequency domain, conducting mulitplication operations and transforming the result back. Hence, an efficent way to derive (weighted) approximated probability densities is:

$$\hat{f}(x) = \mathcal{F}^{-1}(\mathcal{F}[\tilde{d}] \cdot \mathcal{F}[K_h]) \tag{7}$$

$\mathcal{F}$ and $\mathcal{F}^{-1}$ stand for fast-Fourier transformation (FFT) and inverse FFT operations. $\tilde{d}$ is SPDF. Here, we use the canonical Gaussian kernel, utilizing its nature that a Fourier transformed Gaussian is another Gaussian (Holmes 2015).

2. Implementation

Both **cpda** and **gpda** were written in C++. The latter conducts FFT operations at the GPU by writing `logLik_fft` function in CUDA C. Both packages implement three main functions `logLik_pw`, `logLik_fft`, and `logLik_fft2` to calculate model likelihood. The first function implements equation 5 and the latter two implement equation 7. Although equation 7 is faster than equation 5, there are occassions, such as the regression example in Section 1, when the likelihood for each observation is based on MC simulations with their own parameter set.

The difference between `logLik_fft` and `logLik_fft2` is that the former returns a summed log-likelihood value, and the latter returns a matrix with first column orderd observations and second column storing their log-likelihood values.

Specifically, `logLik_pw` takes five arguments:

1. `y`: a vector storing empirical observations
2. `yhat`: a vector storing simulations
3. `h`: a scalar, indicating the kernel bandwidth.
4. `m`: a scalar, a multiplier to adjust bandwidth proportionally

5. `parallel`: a logical switch to run parallel computing with multiple CPU cores. When the switch is on, `logLik_pw` distributes simulated data to available CPU cores to calculate equation 4 in parallel.

`logLik_pw` returns logged likelihood for each observation corresponding to the input vector, `y`.
Similarly, `logLik_fft` and `logLik_fft2` take also `y`, `yhat`, `h` and `m` arguments. They take also three other arguments:

1. `p`: adjusting grid size for the SPDF as a power of 2.
2. `ns`: the number of simulated samples
3. `defected`: a boolean switch to use `ns` when approximating defected densities.

3. A Real-world Example

Piecewise linear ballistic accumulator model (PLBA) is a cognitive model, accounting for the decision process when one evaluates changing information pertinent to a decision. For example, on a motor way, a motorist may want to drive as fast as permited to save journey time, and also to maintain a safe distance from its preceding vehicle. In simple term, the relevant information is the distance between the two vehicles, which influences the motorist's decision to accelerate or decelerate. The relatively stable decision information may abruptly change, when for example another vehicle switches lane and the original information favoring accelerating suddenly becomes the other way around.

PLBA uses LBA model as a building block to account for this type of decision making process. Because of the abrupt information switching and unknown cognitive processes that may happen during the switching, it is less straighforward to derive an analytic likelihood function for PLBA. We can, relatively speaking, easily derive its MC simulation process, which is implemented as `rplba` in **cpda**.

3.1. Estimating piece-wise LBA via Bayesian Inference

4. Performance comparison

4.1. cpda vs. MATLAB

4.2. cpda vs R code

4.3. cpda vs gpda

4.4. Turning of grid and block sizes

Tesla K80

4.5. Optimal resampling interval to mitigate chain stagnanion

4.6. Optimal bandwidth

4.7. How many MC simulations are needed

5. On-going development

5.1. Linear inteprepolation in GPU

5.2. Shipping only empirical data into GPU memory

5.3. More random number generators for other cognitive models

6. Summary

Acknowledgments

References

Beaumont MA (2010). "Approximate Bayesian Computation in Evolution and Ecology." *Annual Review of Ecology, Evolution, and Systematics*, **41**(1), 379–406. doi:10.1146/annurev-ecolsys-102209-144621. URL http://dx.doi.org/10.1146/annurev-ecolsys-102209-144621.

Braak CJFT (2006). "A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces." *Statistics and Computing*, **16**(3), 239–249. ISSN 1573-1375. doi:10.1007/s11222-006-8769-1. URL http://dx.doi.org/10.1007/s11222-006-8769-1.

Brown SD, Heathcote A (2008). "The simplest complete model of choice response time: linear ballistic accumulation." *Cognitive psychology*, **57**(3), 153–178. doi:doi:10.1016/j.cogpsych.2007.12.002.

Dawson MRW (1988). "Fitting the ex-Gaussian equation to reaction time distributions." *Behavior Research Methods, Instruments, & Computers*, **20**(1), 54–57. ISSN 0743-3808, 1532-5970. doi:10.3758/BF03202603. URL https://link.springer.com/article/10.3758/BF03202603.

Gelman A (2014). *Bayesian data analysis*. CRC Press, Boca Raton. ISBN 978-1-4398-4095-5 978-1-4398-4096-2. OCLC: 864304245.

Hoffman MD, Gelman A (2014). "The no-u-turn sampler: Adaptively setting path lengths in Hamiltonian Monte Carlo." *Journal of Machine Learning Research*, **15**(1).

Holmes WR (2015). "A practical guide to the Probability Density Approximation (PDA) with improved implementation and error characterization." *Journal of Mathematical Psychology*, **68–69**, 13–24. doi:10.1016/j.jmp.2015.08.006. URL http://www.sciencedirect.com/science/article/pii/S0022249615000541.

Silverman BW (1986). *Density estimation for statistics and data analysis*, volume 26. CRC press.

Sisson SA, Fan Y (2010). "Likelihood-free Markov chain Monte Carlo." *arXiv:1001.2058 [stat]*. ArXiv: 1001.2058, URL http://arxiv.org/abs/1001.2058.

Turner BM, Dennis S, Van Zandt T (2013). "Likelihood-free Bayesian analysis of memory models." *Psychological Review*, **120**(3), 667–678. ISSN 1939-1471 0033-295X. doi:10.1037/a0032458.

Turner BM, Sederberg PB (2014). "A generalized, likelihood-free method for posterior estimation." *Psychonomic Bulletin & Review*, **21**(2), 227–250. ISSN 1069-9384, 1531-5320. doi:10.3758/s13423-013-0530-0. URL https://link.springer.com/article/10.3758/s13423-013-0530-0.

Affiliation:

Yi-Shin Lin
Division of Psychology, School of Medicine
University of Tasmania
Private Bag 30 Hobart TAS 7005,
Australia
E-mail: yishin.lin@utas.edu.au
URL: http://www.tascl.org/yi-shin-lin.html

Andrew Heathcote
School of Psychology, University of Newcastle
Psychology Building, University Avenue, Callaghan, 2308,NSW, Australia
E-mail: andrew.heathcote@newcastle.edu.au
URL: http://www.newcl.org/Heathcote/

William Holmes
Department of Physics and Astronomy, Vanderbilt University
Nashville, TN 37212, United States of America
E-mail: william.holmes@vanderbilt.edu