# Package 'lbaModel'

September 15, 2025

**Type** Package

**Title** The Linear Ballistic Accumulation Model

**Version** 0.2.9.2

**Date** 2025-09-10

**Maintainer** Yi-Shin Lin <yishinlin001@gmail.com>

**Description** Provides density, distribution and random generation functions for the Linear Ballistic Accumulation (LBA) model, a widely used choice response time model in cognitive psychology. The package supports model specifications, parameter estimation, and likelihood computation, facilitating simulation and statistical inference for LBA-based experiments. For details on the LBA model, see Brown and Heathcote (2008) <doi:10.1016/j.cogpsych.2007.12.002>.

**License** GPL (>= 2)

**Imports** ggdmcPrior, ggdmcModel, Rcpp (>= 1.0.7), methods

**Depends** R (>= 3.3.0)

**LinkingTo** Rcpp (>= 1.0.7), RcppArmadillo (>= 0.10.7.5.0), ggdmcHeaders

**Suggests** testthat

**RoxygenNote** 7.3.2

**Encoding** UTF-8

**NeedsCompilation** yes

**Author** Yi-Shin Lin [aut, cre]

**Repository** CRAN

**Date/Publication** 2025-09-15 07:30:07 UTC

## Contents

dlba *LBA Distribution Functions*

## Description

The functions, dlba, plba, theoretical_dlba, and theoretical_plba, calculate probability distributions for the Linear Ballistic Accumulator model:

- dlba - Probability density function (PDF) for **observed** choice response times
- plba - Cumulative distribution function (CDF) for **observed** choice response times
- theoretical_dlba - Theoretical density function sets a time range and computes the density for this range.
- theoretical_plba - Theoretical cumulative distribution function sets a time range and computes the cumulative density for this range.

## Usage

```
dlba(rt_r, parameter_r, is_positive_drift_r, debug = FALSE)

plba(rt_r, parameter_r, is_positive_drift_r, time_parameter_r, debug = FALSE)

theoretical_dlba(
  parameter_r,
  is_positive_drift_r,
  time_parameter_r,
  debug = FALSE
)

theoretical_plba(
  parameter_r,
  is_positive_drift_r,
  time_parameter_r,
  debug = FALSE
)
```

## Arguments

| | |
|---|---|
| rt_r | For dlba and plba: A numeric vector of **observed** response times |
| parameter_r | A numeric matrix of parameters where rows represent:<br>• Starting point variability (A)<br>• Thresholds (b)<br>• Mean drift rates (mean_v)<br>• The standard deviation of the drift rates (sd_v)<br>• The variability of the non-decision time (st0)<br>• Non-decision time (t0). |

Each column represents parameters for an accumulator.

`is_positive_drift_r`
: A logical vector indicating whether drift rates are strictly positive

`debug`
: A logical value indicating whether to print debug information.

`time_parameter_r`
: For theoretical functions, a numeric vector to set minimal decision time, maximal decision time, and the step size (dt). The internal mechanism uses this vector to set fine time points for evaluation.

## Details

These functions provide the computational foundation for the LBA model:

`dlba` Computes the probability density for observed response times, used during model fitting and likelihood calculations

`plba` Computes the cumulative probability for observed response times, useful for model validation and goodness-of-fit tests

`theoretical_dlba` Computes the theoretical PDF for diagnostic purposes and prediction

`theoretical_plba` Computes the theoretical CDF for model validation and comparison with empirical data

## Value

For all functions: A list containing:

- **values** - The computed distribution values
- **time_points** - The time points used (for theoretical functions and **plba**)
- Additional diagnostic information when applicable

## Examples

```
#------------------------------------------------#
# Example 1: theoretical_dlba and theoretical_plba
#------------------------------------------------#
# Tiny helper to build the parameter matrix
# (rows = A, b, mean_v, sd_v, st0, t0)
param_list2mat <- function(x) do.call(rbind, x)

params <- param_list2mat(list(
  A = c(0.5, 0.5),
  b = c(1.0, 1.0),
  mean_v = c(2.0, 1.0),
  sd_v = c(1.0, 1.0),
  st0 = c(0.0, 0.0),
  t0 = c(0.2, 0.2)
))
is_pos <- rep(TRUE, ncol(params))

# Use a coarse, short grid so it runs quickly
```

```
dt <- 0.05
time_param <- c(0, 2, dt)

# Theoretical densities/cdfs for two accumulators
pdfs <- theoretical_dlba(params, is_pos, time_param)
cdfs <- theoretical_plba(params, is_pos, time_param)

# Combine to check mass and monotonicity quickly
mass_from_pdf <- sum((pdfs[[1]] + pdfs[[2]]) * dt)
tail_cdf_sum <- tail(cdfs[[1]] + cdfs[[2]], 1)

# Both should be close to 1 (within coarse-grid tolerance)
round(c(mass_from_pdf = mass_from_pdf, tail_cdf_sum = tail_cdf_sum), 3)

# Spot-check dlba/plba at a few RTs (shifted by t0)
RT <- c(0.25, 0.50, 0.75) + params[6, 1]
pl <- plba(RT, params, is_pos, time_param)
head(pl[[1]])

## ---- Extended (plots; only in interactive sessions) --------------------

{
  oldpar <- par(no.readonly = TRUE)
  on.exit(par(oldpar), add = TRUE)

  # Reuse objects from above; create a denser grid for nicer plots
  dt2 <- 0.01
  time_param2 <- c(0, 5, dt2)
  DT <- seq(time_param2[1], time_param2[2], by = time_param2[3])

  pdfs2 <- theoretical_dlba(params, is_pos, time_param2)
  cdfs2 <- theoretical_plba(params, is_pos, time_param2)
  pdf_all <- (pdfs2[[1]] + pdfs2[[2]]) * dt2
  cdf_all <- cdfs2[[1]] + cdfs2[[2]]

  par(mfrow = c(2, 1), mar = c(5, 5, 2, 1))
  # PDF
  plot(DT, pdfs2[[1]] * dt2,
    type = "l",
    xlab = "DT", ylab = "Density", main = "Theoretical PDF"
  )
  lines(DT, pdfs2[[2]] * dt2, lty = 2)
  legend("topright", legend = c("Acc 1", "Acc 2"), lty = c(1, 2), bty = "n")

  # Cumulated PDF vs CDF
  plot(DT, cumsum(pdf_all),
    type = "l",
    xlab = "DT", ylab = "Cumulative", main = "Cumulated PDF and CDF",
    ylim = c(0, 1)
  )
  lines(DT, cdf_all, lty = 2)
  legend("bottomright",
    legend = c("Cumulated PDF", "CDF"),
```

```
  lty = c(1, 2), bty = "n"
)

# Optional: grid-comparison for plba
RT2 <- seq(0, 3, by = 0.002) + params[6, 1]
c1 <- plba(RT2, params, is_pos, c(0, 10, 0.01))
c2 <- plba(RT2, params, is_pos, c(0, 5, 0.10))
c3 <- plba(RT2, params, is_pos, c(0, 5, 0.20))

# Okabe-Ito color-blind-safe palette
col1 <- "#0072B2" # dt=0.01
col2 <- "#D55E00" # dt=0.1
col3 <- "#009E73" # dt=0.2

par(mfrow = c(1, 1), mar = c(5, 5, 2, 2))
plot(RT2, c1[[1]],
  type = "l", ylim = c(0, 1), xlab = "RT", ylab = "CDF",
  main = "LBA CDF Estimates under Different Time Grids", lwd = 2,
  col = col1
)
lines(RT2, c1[[2]], lwd = 2, lty = 2, col = col1)

lines(RT2, c2[[1]], lwd = 2, col = col2)
lines(RT2, c2[[2]], lwd = 2, lty = 2, col = col2)
lines(RT2, c3[[1]], lwd = 2, col = col3)
lines(RT2, c3[[2]], lwd = 2, lty = 2, col = col3)
legend("bottomright",
  legend = c(
    "Acc1 (dt=0.01)", "Acc2 (dt=0.01)",
    "Acc1 (dt=0.1)", "Acc2 (dt=0.1)",
    "Acc1 (dt=0.2)", "Acc2 (dt=0.2)"
  ),
  col = c(col1, col1, col2, col2, col3, col3),
  lty = c(1, 2, 1, 2, 1, 2), lwd = 2, bty = "n"
)
}
```

---

`dlba_inverse_external` *Density Function for the LBA Model Using Inverse Transform Method*

---

## Description

Computes the probability density of response times for the Linear Ballistic Accumulator (LBA) model using an inverse transform formulation.

## Usage

```
dlba_inverse_external(
  rt_r,
```

```
  response_r,
  parameter_r,
  is_positive_drift_r,
  time_parameter_r
)
```

## Arguments

| | |
|---|---|
| `rt_r` | A numeric vector of response times (RTs) for which the density should be computed. |
| `response_r` | An integer vector of response choices (indices of winning accumulators), same length as `rt_r`. |
| `parameter_r` | A numeric matrix of LBA parameters. Each column corresponds to an accumulator, and each row corresponds to a parameter, with the expected order being:<br>• `A` - Starting point variability<br>• `b` - Thresholds<br>• `mean_v` - Mean drift rates<br>• `sd_v` - Standard deviation drift rates<br>• `st0` - Variability of the non-decision<br>• `t0` - Non-decision time |
| `is_positive_drift_r` | A logical vector indicating whether the drift rate must be positive for each trial. |
| `time_parameter_r` | A numeric vector to set the simulated time grid, with the expected value for: minimal and maximum decision times and the difference time (i.e., `min dt`, `max dt` and `dt`). |

## Details

This function is a lower-level computational routine intended to be used inside higher-level LBA model evaluation or fitting functions. It uses the inverse transform method to compute exact LBA densities in a numerically stable and efficient way. The function assumes consistent input dimensions across all arguments.

## Value

A numeric vector of LBA densities, one per trial.

---

`fptpdf` *First_Passage_Time*

---

## Description

The functions, `fptpdf`, `fptcdf`, and `n1PDF`, calculate first passage time distributions for the Linear Ballistic Accumulation model. This includes the probability density function `fptpdf`, the cumulative distribution function `fptcdf`, and the node 1 density function, `n1PDF`.

## Usage

```
fptpdf(rt_r, parameter_r, is_positive_drift_r, verbose = FALSE)

fptcdf(rt_r, parameter_r, is_positive_drift_r, verbose = FALSE)

n1PDF(rt_r, parameter_r, is_positive_drift_r, verbose = FALSE)
```

## Arguments

| | |
|---|---|
| rt_r | A numeric vector of response times (RTs). |
| parameter_r | A numeric matrix of model parameters, with each row representing a core model parameter and each column representing an accumulator. Expected row (in fixed order): A, b, mean_v, sd_v, st0, and t0. |
| is_positive_drift_r | A logical vector indicating whether the drift rate must be positive for each trial. |
| verbose | A logical value indicating whether to print debug information. |

## Details

The three functions, respectively, are designed to:

- fptpdf: Calculates the probability density function for (PDF; 1 accumulator).
- fptcdf: Calculates the cumulative distribution function (CDF; 1 accumulator).
- n1PDF: Calculates the node 1 PDF for the multi-accumulator LBA model freeing the non-decision time. **node 1** refers to the accumulator passing the threshold first.

## Value

A numeric vector of probabilities corresponding to input response times.

## Examples

```
#------------------#
# n1PDF example
#------------------#
if (requireNamespace("ggdmcModel", quietly = TRUE)) {
  BuildModel <- getFromNamespace("BuildModel", "ggdmcModel")

  model <- BuildModel(
    p_map = list(
      A = "1", B = "1", t0 = "1", mean_v = "M", sd_v = "1",
      st0 = "1"
    ),
    match_map = list(M = list("s1" = "r1", "s2" = "r2")),
    factors = list(S = c("s1", "s2")),
    constants = c(sd_v = 1, st0 = 0),
    accumulators = c("r1", "r2"),
    type = "lba"
  )
```

```
}


#---------------------------------------#
pop_mean <- c(
  A = .4, B = 1.6, mean_v.true = 3.5, mean_v.false = 2.38,
  t0 = 0.05
)
pop_scale <- c(
  A = 2, B = .2, mean_v.true = .25, mean_v.false = .1,
  t0 = 0.01
)

if (requireNamespace("ggdmcPrior", quietly = TRUE)) {
  BuildPrior <- getFromNamespace("BuildPrior", "ggdmcPrior")

  pop_dist <- BuildPrior(
    p0 = pop_mean,
    p1 = pop_scale,
    lower = rep(NA, model@npar),
    upper = rep(NA, model@npar),
    dists = rep("tnorm", model@npar),
    log_p = rep(FALSE, model@npar)
  )
}

#---------------------------------------#
rt_model <- setLBA(model, population_distribution = pop_dist)
res_process_model <- simulate(rt_model)

param_list2mat <- function(param_list) {
  n_row <- length(param_list[[1]])
  n_col <- length(param_list)

  tmp <- matrix(NA, nrow = n_row, ncol = n_col)

  for (i in seq_len(n_col)) {
    tmp[, i] <- param_list[[i]]
  }
  t(tmp)
}

params_tmp <- list(
  A = c(0.74, 0.74),
  b = c(1.25 + 0.74, 1.25 + 0.74),
  mean_v = c(2.52, 1.50),
  sd_v = c(1.0, 1.0),
  st0 = c(0.0, 0.0),
  t0 = c(0.04, 0.04)
)

# Store the LBA parameter as a list of vectors
print(params_tmp)
```

```
# $A: 0.74 0.74
# $b: 1.99 1.99
# $mean_v: 2.52 1.50
# $sd_v: 1 1
# $st0: 0 0
# $t0: 0.04 0.04

# Convert it to a matrix, each row represents a parameter,
# and each column corresponds to an accumulator.
params <- param_list2mat(params_tmp)

print(params)
#      [,1] [,2]
# [1,] 0.74 0.74
# [2,] 1.99 1.99
# [3,] 2.52 1.50
# [4,] 1.00 1.00
# [5,] 0.00 0.00
# [6,] 0.04 0.04

n_acc <- 2
is_positive_drift <- rep(TRUE, n_acc)
res <- n1PDF(res_process_model$RT, params, is_positive_drift, TRUE)
cat("Print first 30 density values from the LBA node 1 function: \n")
print(head(round(res, 2), 30))

#----------------------------#
# fptpdf computes only 1 RT
#----------------------------#
mean_v <- 2.4
A <- 1.2
b <- 2.7
t0 <- .2
sd_v <- 1
st0 <- 0
positive_drift_r <- TRUE

params <- list(
  A = rep(A, 2),
  b = rep(b, 2),
  mean_v = rep(mean_v, 2),
  sd_v = rep(sd_v, 2),
  st0 = rep(st0, 2),
  t0 = rep(t0, 2)
)

n_accumulator <- length(params$A)
is_positive_drift <- rep(TRUE, n_accumulator)
params_mat <- param_list2mat(params)

RT <- 0.3
result <- fptpdf(RT, params_mat, is_positive_drift, TRUE)
```

```
#--------------------------------------#
# fptpdf and fptcdf computes a range RTs
#--------------------------------------#
RT <- seq(0, 10, .01) + t0
result1 <- fptpdf(RT, params_mat, is_positive_drift)
result2 <- fptcdf(RT, params_mat, is_positive_drift)
```

---

`lba-class` *An S4 Class of the* `lba` *Object*

---

## Description

The `lba` class represents an LBA model with slots for model specification, population distribution, and other necessary components. The `setLBA` function is the constructor for creating `lba` objects.

## Usage

```
setLBA(model, population_distribution = NULL, is_positive_drift = TRUE)
```

## Arguments

| | |
|---|---|
| `model` | A model object containing the model specification |
| `population_distribution` | Optional population distribution for parameters (default `NULL`) |
| `is_positive_drift` | a Boolean value indicating whether to use strictly positive drift rates |

## Details

The LBA model is a popular decision-making model that assumes evidence accumulates linearly toward decision thresholds. The setLBA function initialises this model by creating an S4 object with all necessary components, including automatically calculating node 1 indices and creating an indicator vector to inform whether to use strictly positive drift rates.

## Value

An object of class lba containing:

- The model specification
- Population distribution (if provided)
- Node 1 index information
- Whether to restrict drift rates to be positive

## Slots

`model` A model object containing the model specification

`population_distribution` The population distribution for parameters (can be NULL)

`node_1_index` Index information for the first node (automatically calculated)

`is_positive_drift` Logical vector indicating positive drift for each accumulator

rlba_r *Simulate Choices and Response Times*

## Description

Simulate Choices and Response Times

## Usage

```
rlba_r(
  parameter_r,
  is_positive_drift_r,
  time_parameter_r,
  n = 1L,
  use_inverse_method = FALSE,
  debug = FALSE
)

rlba(
  parameter_r,
  is_positive_drift_r,
  time_parameter_r,
  n = 1L,
  use_inverse_method = FALSE,
  debug = FALSE,
  seed = NULL
)
```

## Arguments

| | |
|---|---|
| `parameter_r` | A numeric matrix of LBA parameters. Each column corresponds to an accumulator, and each row corresponds to a parameter, with the expected order being:<br>• `A` - Starting point variability<br>• `b` - Thresholds<br>• `mean_v` - Mean drift rates<br>• `sd_v` - Standard deviation drift rates<br>• `st0` - Variability of the non-decision<br>• `t0` - Non-decision time |
| `is_positive_drift_r` | A logical vector indicating whether the drift rate (`v`) is strictly positive. |
| `time_parameter_r` | A numeric vector to set the simulated time grid, with the expected value for: minimal and maximum decision times and the difference time (i.e., `min dt`, `max dt` and `dt`). |
| `n` | Integer. Number of trials to simulate. Defaults to `1L`. |

use_inverse_method
: Logical. If TRUE, use the inverse transform sampling method; otherwise use the standard sampling method.Defaults to FALSE. TODO: fix the memory problem.

debug
: Logical. If TRUE, prints detailed messages for debugging. Defaults to FALSE.

seed
: Optional integer. If provided, sets the random seed for reproducible simulation. Defaults to NULL. Available only in the R interface.

## Details

This function generates simulated data for the LBA model. The function supports both standard and inverse transform sampling methods. The inverse method may offer better numerical behaviour in edge cases (e.g., small RTs or boundary conditions).

## Value

A data.frame. Each row corresponds to a simulated trial containing:

- trial — Trial index
- choice — Index of the winning accumulator
- rt — Simulated response time

## Examples

```
param_list2mat <- function(param_list) {
  n_row <- length(param_list[[1]])
  n_col <- length(param_list)
  out <- matrix(NA, nrow = n_row, ncol = n_col)

  for (i in seq_len(n_col)) {
    out[, i] <- param_list[[i]]
  }
  t(out)
}

A <- 1.2
b <- 2.7
t0 <- .2

mean_v <- c(2.4, 2.2)
sd_v <- c(1, 1)

RT <- seq(0, 3, .4) + t0
posdrift <- TRUE
nv <- length(mean_v)

st0 <- 0

params_tmp <- list(
  A = rep(A, nv),
  b = rep(b, nv),
  mean_v = mean_v,
```

```
  sd_v = sd_v,
  st0 = rep(st0, nv),
  t0 = rep(t0, nv)
)

params <- param_list2mat(params_tmp)
is_positive_drift <- rep(TRUE, nv)

n <- 1
seed <- 123

dt <- 0.01
min_dt <- 0
max_dt <- 5
time_parameter_r <- c(min_dt, max_dt, dt)


set.seed(seed)
# R interface
result1 <- rlba(params, is_positive_drift, time_parameter_r, n,
  seed = seed, debug = TRUE
)
# C++ interface. No seed argument.
result2 <- rlba_r(params, is_positive_drift, time_parameter_r, n,
  debug = TRUE
)

print(result1)
print(result2)
```

---

`simulate_lba` *Simulate Data from an LBA Model*

---

## Description

Simulate response times and choices from a Linear Ballistic Accumulation (LBA) model with a model specification (typically from `ggdmcModel::BuildModel`).

## Usage

```
## S4 method for signature 'lba'
simulate(
  object,
  nsim = 4L,
  seed = NULL,
  n_subject = 3L,
  parameter_vector = NULL,
  use_inverse_method = FALSE,
  debug = FALSE
)
```

## Arguments

| | |
|---|---|
| object | An object of class lba that defines the model structure and parameters. |
| nsim | Integer. Number of trials to simulate per subject. Defaults to 4. |
| seed | Optional integer. Sets the random seed for reproducibility. Defaults to NULL. |
| n_subject | Integer. Number of subjects to simulate. Defaults to 3. |
| parameter_vector | A named vector or list of parameters (e.g., A, b, mean_v.true, t0). Supply either parameter_vector here or a population distribution via setLBA (typically built with ggdmcPrior::BuildPrior). Defaults to NULL. |
| use_inverse_method | Logical. If TRUE, use inverse transform sampling; otherwise use the process model to sample. Defaults to FALSE. |
| debug | Logical. If TRUE, print debugging output during simulation. Defaults to FALSE. |

## Details

This method simulates data from a design-based LBA model. You can simulate multiple subjects, override default parameters, and choose between standard and inverse sampling methods. Turn on debugging mode by entering TRUE to the option, debug.

## Value

A data frame with:

- s (lowercase): subject identifiers (factor)
- R (uppercase): choices (integer/character)
- RT: response times (numeric)

Plus user-defined condition columns derived from the model

## Notes

The internal mechanism is case sensitive. The choice of using upper- or lowercase letters to denote variables is a convention (originated from DMC), rather than a strict requirement.

## See Also

simulate_lba_trials (low-level C++ back end), theoretical_dlba, plba, dlba

Other LBA simulation: validate_lba_parameters()

## Examples

```
if (requireNamespace("ggdmcModel", quietly = TRUE)) {
  BuildModel <- getFromNamespace("BuildModel", "ggdmcModel")

  model <- BuildModel(
    p_map = list(
      A = "1", B = "1", t0 = "1", mean_v = "M", sd_v = "1",
```

```
      st0 = "1"
    ),
    match_map = list(M = list("s1" = "r1", "s2" = "r2")),
    factors = list(S = c("s1", "s2")),
    constants = c(sd_v = 1, st0 = 0),
    accumulators = c("r1", "r2"),
    type = "lba"
  )
}
p_vector <- c(
  A = .75, B = 1.25, mean_v.false = 1.5, mean_v.true = 2.5,
  t0 = 0.15
)
sub_model <- setLBA(model)
sim_dat <- simulate(sub_model,
  nsim = 256, parameter_vector = p_vector,
  n_subject = 1
)
head(sim_dat)
```

# Index