

Package ‘Delaporte’

August 24, 2026

Type Package

Title Statistical Functions for the Delaporte Distribution

Version 9.0.0

Date 2026-08-23

Description Provides probability mass, distribution, quantile, random-variate generation, and method-of-moments parameter-estimation functions for the Delaporte distribution with parameterization based on Vose (2008) <isbn:9780470512845>. The Delaporte is a discrete probability distribution which can be considered the convolution of a negative binomial distribution with a Poisson distribution. Alternatively, it can be considered a counting distribution with both Poisson and negative binomial components. It has been studied in actuarial science as a frequency distribution which has more variability than the Poisson, but less than the negative binomial.

License BSD_2_clause + file LICENSE

URL <https://github.com/aadler/Delaporte>

BugReports <https://github.com/aadler/Delaporte/issues>

Depends R (>= 3.6.0)

Imports stats, parallel

Suggests covr, tinytest

ByteCompile yes

NeedsCompilation yes

UseLTO yes

SystemRequirements A version of Fortran supporting the LOG_GAMMA Intrinsic and the ieee_arithmetic module.

Author Avraham Adler [aut, cph, cre] (ORCID:
<<https://orcid.org/0000-0002-3039-0703>>)

Maintainer Avraham Adler <Avraham.Adler@gmail.com>

Repository CRAN

Date/Publication 2026-08-24 05:20:53 UTC

Contents

Delaporte-package	2
Delaporte	3
Delaporte-defunct	8
setDelapThreads	8

Index	10
--------------	-----------

Delaporte-package	<i>Statistical Functions for the Delaporte Distribution</i>
-------------------	---

Description

Provides probability mass, distribution, quantile, random-variate generation, and method-of-moments parameter-estimation functions for the Delaporte distribution with parameterization based on Vose (2008) <isbn:9780470512845>. The Delaporte is a discrete probability distribution which can be considered the convolution of a negative binomial distribution with a Poisson distribution. Alternatively, it can be considered a counting distribution with both Poisson and negative binomial components. It has been studied in actuarial science as a frequency distribution which has more variability than the Poisson, but less than the negative binomial.

Details

The DESCRIPTION file:

```
Package:           Delaporte
Type:              Package
Title:             Statistical Functions for the Delaporte Distribution
Version:           9.0.0
Date:              2026-08-23
Authors@R:         c(person(given="Avraham", family="Adler", role=c("aut", "cph", "cre"), email="Avraham.Adler@gmail.com"))
Description:        Provides probability mass, distribution, quantile, random-variate generation, and method-of-moments
License:           BSD_2_clause + file LICENSE
URL:               https://github.com/aadler/Delaporte
BugReports:        https://github.com/aadler/Delaporte/issues
Depends:           R (>= 3.6.0)
Imports:           stats, parallel
Suggests:          covr, tinytest
ByteCompile:       yes
NeedsCompilation:  yes
UseLTO:            yes
SystemRequirements: A version of Fortran supporting the LOG_GAMMA Intrinsic and the ieee_arithmetic module.
Author:            Avraham Adler [aut, cph, cre] (ORCID: <https://orcid.org/0000-0002-3039-0703>)
Maintainer:        Avraham Adler <Avraham.Adler@gmail.com>
Archs:             x64
```

Index of help topics:

Delaporte	The Delaporte Distribution
Delaporte-defunct	Defunct functions in package 'Delaporte'
Delaporte-package	Statistical Functions for the Delaporte Distribution
setDelapThreads	Set or get the number of OpenMP threads 'Delaporte' should use

Author(s)

Avraham Adler [aut, cph, cre] (ORCID: <<https://orcid.org/0000-0002-3039-0703>>)

Maintainer: Avraham Adler <Avraham.Adler@gmail.com>

Delaporte	<i>The Delaporte Distribution</i>
-----------	-----------------------------------

Description

Density, distribution, quantile, random variate generation, and method of moments parameter estimation functions for the Delaporte distribution with parameters alpha, beta, and lambda.

Usage

```
ddelap(x, alpha, beta, lambda, log = FALSE)
pdelap(q, alpha, beta, lambda, lower.tail = TRUE, log.p = FALSE)
qdelap(p, alpha, beta, lambda, lower.tail = TRUE, log.p = FALSE, exact = TRUE)
rdelap(n, alpha, beta, lambda, exact = TRUE)
```

```
MoMdelap(x, type = 2L)
```

Arguments

x	vector of (non-negative integer) quantiles.
q	vector of quantiles.
p	vector of probabilities.
n	number of observations.
alpha	vector of alpha parameters of the gamma portion of the Delaporte distribution. Must be strictly positive, but need not be integer.
beta	vector of beta parameters of the gamma portion of the Delaporte distribution. Must be strictly positive, but need not be integer.
lambda	vector of lambda parameters of the Poisson portion of the Delaporte distribution. Must be strictly positive, but need not be integer.
log, log.p	logical; if TRUE, probabilities p are given as log(p).
lower.tail	logical; if TRUE (default), probabilities are $P[X \leq x]$, otherwise, $P[X > x]$.
exact	logical; if TRUE uses double summation to generate quantiles or random variates. Otherwise uses Poisson-negative binomial approximation.
type	integer; 1L will return g1 , 2L will return G1 , and 3L will return b1 , as per skewness .

Details

Definition: The Delaporte distribution with parameters α , β , and λ is a discrete probability distribution which can be considered the convolution of a negative binomial distribution with a Poisson distribution. Alternatively, it can be considered a counting distribution with both Poisson and negative binomial components. The Delaporte’s probability mass function, called via `ddelap`, is:

$$p(n) = \sum_{i=0}^n \frac{\Gamma(\alpha + i) \beta^i \lambda^{n-i} e^{-\lambda}}{\Gamma(\alpha) i! (1 + \beta)^{\alpha+i} (n - i)!}$$

for $n = 0, 1, 2, \dots; \alpha, \beta, \lambda > 0$.

If an element of `x` is negative—including `-Inf`—the result of `ddelap` is zero ($-\infty$ on the log scale) and no warning is issued, matching base R `dpois`. Likewise, for any `q` below the support `pdelap` returns a cumulative probability of zero and a survival probability of one, also in agreement with base R.

The Delaporte’s cumulative distribution function, `pdelap`, is calculated through double summation:

$$CDF(n) = \sum_{j=0}^n \sum_{i=0}^j \frac{\Gamma(\alpha + i) \beta^i \lambda^{j-i} e^{-\lambda}}{\Gamma(\alpha) i! (1 + \beta)^{\alpha+i} (j - i)!}$$

for $n = 0, 1, 2, \dots; \alpha, \beta, \lambda > 0$.

If only singleton values for the parameters are passed in, the function uses a shortcut. It identifies the largest value passed to it, computes a vector of CDF values for all integers up to and including that value, and reads the remaining results from this vector. This requires only one double summation instead of `length(q)` such summations. If at least one of the parameters is itself a vector of length greater than 1, the function has to build the double summation for each entry in `q`.

Distributional Functions:

Density and Distribution: `ddelap` will return 0 for all values $> 2^{63}$ whereas `pdelap` will not run at all, due to the limitations of integer representation. For values $> 2^{24}$, or $> 2^{15}$ when slower code paths must be used, when using R interactively, `pdelap` will ask for positive input from the user to continue, as otherwise, depending on the parameters, the function can take hours to complete. When run as a script, `pdelap` at those values will stop with an error, and offer the user the option to set an environment variable to proceed nevertheless.

`ddelap` with `log = TRUE` and `pdelap` with `log.p = TRUE` compute log-probabilities directly in log space rather than taking `log()` of a linear-space result, so deep-tail log-probabilities and log-CDF/log-survival values remain accurate to near full double precision even where the corresponding linear-space value underflows to 0. For example, `pdelap(2000, 1, 1, 1, lower.tail = FALSE, log.p = TRUE)` returns approximately -1386 rather than `-Inf`.

`pdelap` with `lower.tail = FALSE` likewise computes the survival function by direct tail summation once it is small enough that $1 - \text{CDF}$ would lose precision to catastrophic cancellation, rather than always subtracting from 1. Near the opposite ends of the range—log-CDF values within roughly `sqrt(.Machine$double.eps)` of 0 and their log-survival mirror—`pdelap` derives the value from the complementary tail via `log1p(-exp(.))`, so log-scale results remain accurate to near full double precision throughout both saturated bands.

Quantile: The quantile function, `qdelap`, is right continuous: `qdelap(q, alpha, beta, lambda)` is the smallest integer x such that $P(X \leq x) \geq q$. This function has two versions:

1. When `exact = TRUE`, the function builds a CDF vector and the first value for which the CDF is greater than or equal to q is returned as the quantile.
2. When `exact = FALSE` and there are no vector-valued parameters or NA/NaN in the data, the Delaporte’s definition as a counting distribution with both a Poisson and a negative binomial component is used. Based on Karlis & Xekalaki (2005), the function generates n gamma variates Γ with shape α and scale β , then n pseudo-Delaporte variates as Poisson random variables with parameter $\lambda + \Gamma$, and finally calls the `quantile` (with `type = 8`) on the result. The number of variates n is set to five orders of magnitude greater than the mean, with a minimum of 10K variates and a maximum of 10M variates.

The “exact” method is **always** more accurate and, since the table-building algorithm was rewritten to build the CDF vector in $O(K)$ rather than $O(K^2)$ time, orders of magnitude faster than `exact = FALSE` for essentially all practical parameter ranges. Benchmarks show `exact = TRUE` outperforming the approximate version by roughly 30- to 50,000-fold across means from single digits up to the low millions. The “exact” method *must* be used when passing parameter vectors, since pooling observations across distinct parameter triplets would be intractable.

There remains a niche use case for `exact = FALSE`. Since `exact = TRUE` builds an explicit table of length K , its time and memory cost grow with K , while `exact = FALSE` has a roughly fixed cost bounded by its capped simulation size. K depends on both the mean and dispersion of the distribution:

$$K \approx \alpha\beta + \lambda + 10\sqrt{\alpha\beta(1 + \beta) + \lambda}$$

A distribution with a modest mean but high overdispersion, such as one with a small α relative to β , can require as large a table as one with a much larger, near-Poisson mean.

Current benchmarks indicate `exact = FALSE` is worth considering only once K exceeds approximately 1.5×10^7 , the point at which the exact method’s memory use of roughly 17 bytes per unit of K begins to exceed the bounded cost of the approximate method. This supersedes the older ad-hoc guidance of using `exact = FALSE` once $\alpha\beta + \lambda \approx 2500$. This older guidance is now four to five orders of magnitude too conservative given the current algorithm, and was in any case based on the mean alone rather than K .

Both versions return NaN for quantiles < 0 or > 1 , 0 for quantiles $= 0$, and Inf for quantiles exactly $= 1$, following the conventions of `qpois`.

`qdelap` with `lower.tail = FALSE` and/or `log.p = TRUE` searches in the native space of the supplied probabilities, so survival and log-scale targets resolve at their full representable depth. Round trips through `pdelap` in the same mode are exact wherever the probability is representable.

Parameter sets extreme enough to defeat the internal recurrence table are handled by explicit summation whose cost grows with the *square* of the answer. When the true quantile is computationally unreachable that way – in practice, when it exceeds 2^{15} , and such parameter sets generally place it beyond 2^{53} , past exact integer representation in a double – `qdelap` returns NaN for the affected entries with the warning “quantile too large to compute exactly for these parameter values; NaN returned”, rather than an arbitrarily wrong bound. Entries the summation did resolve, and the $p = 0$ and $p = 1$ boundaries, are returned normally.

Random Variate Generation: The random variate generator, `rdelap`, also has multiple versions. When `exact = TRUE`, it uses inversion by creating a vector of n uniformly distributed random variates between 0 and 1. If all the parameters are singletons, a single CDF vector is constructed as per the quantile function, and the entries corresponding to the uniform variates are read off

of the constructed vector. If the parameters are themselves vectors, it then passes the entire uniform variate vector to `qdelap`, which is slower. When `exact = FALSE`, regardless of the length of the parameters, it generates n gamma variates Γ with shape α and scale β and then n pseudo-Delaporte variates as Poisson random variables with parameter $\lambda + \Gamma$. As there is no pooling, each individual random variate reflects the parameter triplet which generated it. The non-inversion method is usually faster.

Method of Moments Fitting: `MoMdelap` uses the definition of the Delaporte's mean, variance, and skew to calculate the method of moments estimates of α , β , and λ , which it returns as a numeric vector. This estimate is also a reasonable starting point for maximum likelihood estimation using nonlinear optimizers such as `optim` or `nloptr`. If the data is clustered near 0, there are times when method of moments would result in a non-positive parameter. In these cases `MoMdelap` will throw an error. For the sample skew, the user has the choice to select g_1 , G_1 , or b_1 as defined in Joanes & Gill (1997) and found in `skewness`. The selection defaults to option 2, G_1 , which Joanes & Gill found to have the least mean-square error for non-normal distributions.

Value

`ddelap` gives the probability mass function, `pdelap` gives the cumulative distribution function, `qdelap` gives the quantile function, and `rdelap` generates random deviates. Non-positive or non-finite values for α , β , or λ return `NaN` for that particular entry.

`MoMdelap` returns a triplet comprising a method-of-moments based estimate of α , β , and λ .

Arguments to `ddelap` and `pdelap` that fall below the support (negative values and $-\text{Inf}$) return 0 for the density and the lower-tail CDF ($-\text{Inf}$ on the log scale) and 1 for the survival function (0 on the log scale), consistent with base R distributions. Invalid quantiles passed to `qdelap` will result in return values of `NaN` or `Inf` as appropriate.

The length of the result is determined by `x` for `ddelap`, `q` for `pdelap`, `p` for `qdelap`, and `n` for `rdelap`. The distributional parameters (α, β, λ) are recycled as necessary to the length of the result.

A Note on Accuracy

Three limits at the edges of double-precision representation are inherent to these algorithms.

First, `pdelap` with `lower.tail = FALSE` builds its survival values by backward accumulation over an internal table sized to `max(q)`, so a given survival probability may differ, at the level of its accumulated rounding, depending on the other elements of `q` supplied in the same call. The relative rounding error is of order $K * \text{.Machinedouble.eps}$ for a table of length K , and up to about `sqrt(.Machine$double.eps)` where the survival is derived from $1 - \text{CDF}$.

Second, in *linear* space (`log.p = FALSE`), survival targets within about $K * \text{.Machine$double.eps}$ of 1 sit below the accumulation's resolution, and targets within about `sqrt(.Machine$double.eps)` of 1 may round-trip through `qdelap` one support point high. Log-space inputs are not subject to either limit.

Third, `qdelap` with vector-valued parameters follows base R's linear-space convention (transforming log and upper-tail probabilities before searching, with base R's $64 * \text{EPS}$ fuzz), so deep-tail `log.p` and `lower.tail = FALSE` targets saturate there in the same way base R's quantile functions did before R 4.1; the full-depth native-space search applies to scalar parameters only.

Performance

Deep upper-tail quantiles on the log scale are exact but not free: the internal table must extend until the log-survival reaches the target, so the cost of `qdelap(lp, ..., lower.tail = FALSE, log.p = TRUE)` grows with $|lp|$ divided by the tail's decay rate $\log\left(\frac{1+\beta}{\beta}\right)$. At high dispersion this is material: with $\beta = 40$, a target of -2000 requires a table of roughly 8×10^4 support points and several seconds of direct tail summation for its seeds.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

References

- Joanes, D. N. and Gill, C. A. (1998) Comparing Measures of Sample Skewness and Kurtosis. *Journal of the Royal Statistical Society. Series D (The Statistician)* **47**(1), 183–189. doi:[10.1111/1467-9884.00122](https://doi.org/10.1111/1467-9884.00122)
- Johnson, N. L., Kemp, A. W. and Kotz, S. (2005) *Univariate discrete distributions* (Third ed.). John Wiley & Sons. pp. 241–242. ISBN 978-0-471-27246-5.
- Karlis, D. and Xekalaki, E. (2005) Mixed Poisson Distributions. *International Statistical Review* **73**(1), 35–58. <https://projecteuclid.org/euclid.isr/1112304811>
- Vose, D. (2008) *Risk analysis: a quantitative guide* (Third, illustrated ed.). John Wiley & Sons. pp. 618–619. ISBN 978-0-470-51284-5

See Also

[Distributions](#) for standard distributions, including [dnbinom](#) for the negative binomial distribution and [dpois](#) for the Poisson distribution, and [skewness](#) for skew options.

Examples

```
## Density and distribution
A <- c(0, seq_len(50))
PMF <- ddelap(A, alpha = 3, beta = 4, lambda = 10)
CDF <- pdelap(A, alpha = 3, beta = 4, lambda = 10)

## Quantile
A <- seq(0, .95, .05)
qdelap(A, alpha = 3, beta = 4, lambda = 10)
A <- c(-1, A, 1, 2)
qdelap(A, alpha = 3, beta = 4, lambda = 10)

## Compare a Poisson, negative binomial, and three Delaporte distributions with the same mean:
P <- rpois(25000, 25) ## Will have the tightest spread
DP1 <- rdelap(10000, alpha = 2, beta = 2, lambda = 21) ## Close to the Poisson
DP2 <- rdelap(10000, alpha = 3, beta = 4, lambda = 13) ## In between
DP3 <- rdelap(10000, alpha = 4, beta = 5, lambda = 5) ## Close to the Negative Binomial
NB <- rnbinom(10000, size = 5, mu = 25) ## Will have the widest spread
mean(P); mean(NB); mean(DP1); mean(DP2); mean(DP3) ## Means should all be near 25
MoMdelap(DP1); MoMdelap(DP2); MoMdelap(DP3) ## Estimates should be close to originals
```

```
## Not run:
plot(density(P), col = "black", lwd = 2, main = "Distribution Comparison",
     xlab = "Value", xlim = c(0, 80))
lines(density(DP1), col = "blue", lwd = 2)
lines(density(DP2), col = "green3", lwd = 2)
lines(density(DP3), col = "orange3", lwd = 2)
lines(density(NB), col = "red", lwd = 2)
legend(x = "topright", legend = c("Poisson {l=25}", "DP {a=2, b=2, l=21}",
    "DP {a=3, b=4, l=13}", "DP {a=4, b=5, l=5}", "NegBinom {a=5, b=5}"),
     col=c("black", "blue", "green3", "orange3", "red"), lwd = 2)

## End(Not run)
```

Delaporte-defunct	<i>Defunct functions in package ‘Delaporte’</i>
-------------------	---

Description

These functions are defunct and no longer available.

Details

The following functions are defunct; use the replacement indicated below:

- As of version 5.0.0, the `old = TRUE` method for the [qdelap](#) and [rdelap](#) functions is defunct and has been removed.

setDelapThreads	<i>Set or get the number of OpenMP threads Delaporte should use</i>
-----------------	--

Description

Set or get number OpenMP threads to be used by **Delaporte** functions which are parallelized. These include `ddelap`, `pdelap`, `qdelap`, and `rdelap`.

Usage

```
setDelapThreads(n)
getDelapThreads()
```

Arguments

`n` integer: maximum number of threads to be used.

Details

The requested thread count is stored in an environment local to the package namespace, following the guidance in Writing R Extensions, and is passed to the compiled routines on each call, where it scopes only **Delaporte**'s own parallel regions. Neither the global options registry nor the process-wide OpenMP settings are touched, so other OpenMP-using packages are unaffected.

On load, the thread count is initialized to the OpenMP runtime maximum at that moment, which respects environment variables such as OMP_NUM_THREADS and OMP_THREAD_LIMIT.

For systems which return valid values for [detectCores](#), the maximum number of threads will be capped at that value.

Value

getDelapThreads returns an integer representing the maximum number of allowed threads.

setDelapThreads invisibly returns the integer value actually set after flooring and capping.

Author(s)

Avraham Adler <Avraham.Adler@gmail.com>

See Also

[detectCores](#) in **parallel** package.

Examples

```
getDelapThreads()  
setDelapThreads(2L)  
getDelapThreads()
```

Index

- * **data**
 - setDelapThreads, 8
- * **distribution**
 - Delaporte, 3
 - Delaporte-package, 2
- * **package**
 - Delaporte-package, 2
- ddelap (Delaporte), 3
- Delaporte, 3
- delaporte (Delaporte), 3
- Delaporte-defunct, 8
- Delaporte-package, 2
- detectCores, 9
- Distributions, 7
- dnbinom, 7
- dpois, 4, 7
- getDelapThreads (setDelapThreads), 8
- MoMdelap (Delaporte), 3
- nloptr, 6
- openMP (setDelapThreads), 8
- openmp (setDelapThreads), 8
- optim, 6
- pdelap (Delaporte), 3
- qdelap, 8
- qdelap (Delaporte), 3
- qpois, 5
- quantile, 5
- rdelap, 8
- rdelap (Delaporte), 3
- setDelapThreads, 8
- skewness, 3, 6, 7