Mathematical Software for the Evaluation of Distribution Functions of Quadratic Forms of Gaussian Random Variables

Denis Arzelier[1][0000−0003−0149−1369], Florent Bréhard[2][0000−0003−3909−2099], and Mioara Joldes[3][0000−0001−7781−3745]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
arzelier@laas.fr
[2] Univ. Lille, CNRS, Centrale Lille, UMR 9189 CRIStAL, F-59000 Lille, France
florent.brehard@univ-lille.fr
[3] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
joldes@laas.fr

Abstract. We describe a mathematical software package for the fast and reliable evaluation of the cumulative distribution function (cdf) of positive semidefinite quadratic forms in Gaussian random variables. Building on our previous theoretical work, the package implements several symbolic-numeric algorithms based on the closed-form modified Laplace transform of the cdf and its D-finite structure: a classical recurrence, a positivity-preserving recurrence that eliminates the cancellation phenomenon between coefficients alternating in sign, and a faster scheme combining transposed multipoint evaluation with FFT-based power-series exponentiation. The paper focuses on the software aspects of these developments: algorithm implementation, numerical robustness, precision management, and practical handling of difficult parameter regimes. We detail the implementation in Julia and MATLAB, discuss the overall evaluation workflow, and compare different operating modes for balancing reliability and speed of execution. Numerical experiments on benchmark instances show that the positivity-preserving implementation provides the best overall robustness, while the fast implementation opens the way to scalable high-dimensional computation.

Keywords: mathematical software · quadratic forms in Gaussian variables · symbolic-numeric computation · numerical reliability · Julia and MATLAB implementations

1 Introduction

Quadratic forms in Gaussian random variables arise in many areas of applied mathematics and scientific computing, including multivariate statistics, signal processing and control, uncertainty quantification, and risk assessment [5,6,8]. In such applications, a central task is the evaluation of the cumulative distribution function

$$F_Q(q) = \mathbb{P}(Q \le q), \qquad q \ge 0,$$

for a positive semidefinite Gaussian quadratic form

$$Q = \Xi^\top A \Xi, \qquad \Xi \sim \mathcal{N}(\mu, \Sigma), \quad A \succeq 0,$$

where $\Xi \in \mathbb{R}^d$ is a normal random vector. Writing the spectral decomposition

$$\Sigma^{1/2} A \Sigma^{1/2} = P \Lambda P^\top, \qquad \Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_d), \quad \lambda_i \ge 0,$$

and defining $b := P^\top \Sigma^{-1/2} \mu$, one obtains the classical reduction

$$Q = \sum_{i=1}^{d} \lambda_i (Z_i + b_i)^2 = \sum_{i=1}^{d} X_i^2, \tag{1}$$

where $Z_1, \ldots, Z_d$ are independent standard normal variables and $X_i \sim \mathcal{N}(\nu_i, \sigma_i^2)$ are independent with

$$\nu_i = \sqrt{\lambda_i}\, b_i, \qquad \sigma_i^2 = \lambda_i.$$

The random variable Q is distributed as a generalized noncentral chi-square variable [3].

Existing approaches include numerical inversion of characteristic functions, saddlepoint approximations, and convergent series or mixture methods, see, e.g., [5,4,7,9,6,8]. Our companion paper [1] proposes an approach based on Laplace transform of the cdf and algebraic properties of D-finite functions. The present paper is concerned with the corresponding software artifact. The main practical issue is reliable finite-precision evaluation, especially in regimes where cancellation may seriously degrade the result. In larger numerical pipelines, robustness and reproducibility are as important as execution speed.

We present a package implementing three evaluation modes: a direct recurrence CdfRec, a positivity-preserving recurrence CdfPosRec, and a faster accelerated scheme based on transposed multipoint evaluation and FFT-based power-series exponentiation CdfFast. The package is available in MATLAB and Julia. Our goal is to show that the approach of [1] can be turned into practical mathematical software, and to assess experimentally the trade-offs between simplicity, robustness, and speed.

2 Algorithms and Software

The software follows the reduction of [1]. For a fixed threshold q, define

$$g(\xi) := \mathbb{P}(Q \le \xi q), \qquad \xi \ge 0,$$

so that $F_Q(q) = g(1)$. Using (1), this becomes

$$g(\xi) = \mathbb{P}\left(\sum_{i=1}^{d} X_i^2 \le \xi q\right),$$

and the initial problem is reduced to the evaluation of $g(1)$. The key mathematical ingredient is a modified Laplace transform of g, which is a D-finite function: it satisfies a linear differential equation with polynomial coefficients, and its Taylor coefficients therefore satisfy a linear recurrence with polynomial coefficients in the index variable. Furthermore, this function is also available in closed form in terms of the reduced parameters (p_i, m_i) [10,1]:

$$p_i = \frac{q}{2\sigma_i^2} = \frac{q}{2\lambda_i}, \qquad m_i = \frac{\nu_i^2}{2\sigma_i^2} = \frac{b_i^2}{2}.$$

After preprocessing, the parameters are ordered so that $p_1 = \max_i p_i$, and a preconditioning step factors out the dominant exponential contribution. The remaining expression then depends on the shifted quantities $\alpha_i = p_1 - p_i$ and on the noncentrality parameters m_i. This rewriting is numerically important: it separates the largest exponential scale to ensure the positivity of the Taylor coefficients of the remaining function, thereby reducing cancellation in the final summation. The preconditioned Laplace transform remains D-finite. As a result, the evaluation of $F_Q(q) = g(1)$ is reduced to the computation of a finite number of coefficients, followed by the summation of a truncated series and the restoration of the prefactor e^{-p_1}. Moreover, the explicit remainder bounds derived in [1] allow the truncation order N to be selected automatically.

- Algorithm CdfRec is properly formulated in [1, Sec. 2] as the generalization in any dimension d of the scalar recurrence approach of [10]. Despite the positivity of the Taylor coefficients, because the recurrence generating them involves coefficients with alternating signs, cancellation may severely degrade numerical accuracy as the dimension d or the truncation order N increases.

- Algorithm CdfPosRec in [1, Sec. 3] replaces the scalar recurrence by an equivalent positivity-preserving coupled recurrence, which strongly reduces cancellation and gives the best finite-precision robustness in our experiments.
- Finally, Algorithm CdfFast in [1, Sec. 4] accelerates coefficient generation by transposed multipoint evaluation followed by an FFT-based power-series exponentiation. Its asymptotic cost is more favorable than the purely recurrence-based approaches, at the price of a larger constant factor and a more sophisticated implementation. This makes it most beneficial for high-dimensional problems, when both d and N get large. In addition, the error analysis detailed in [1] demonstrates that this method only guarantees an *absolute* error on $g(1) = F_Q(q)$.

These algorithms are implemented [2] as the routines `cdf_rec`, `cdf_pos_rec` and `cdf_fast` in a Julia package called `Quadratiques.jl`. They take as input the values of the parameters ν_i, σ_i of the distribution, together with $R = \sqrt{q}$. An additional integer parameter t in `cdf_fast` controls the degree of intermediate power series in the transposed multipoint evaluation scheme. Using Julia's multiple dispatch, computations can be carried out seamlessly in machine floating-point double-precision arithmetic (`Float64` datatype) or arbitrary precision arithmetic using the integrated `BigFloat` wrapper for the C MPFR library. For machine precision however, specific routines `cdf_rec_exponent` and `cdf_pos_rec_exponent` implement a finer control of the exponent of `Float64` numbers to reduce the risk of overflow and underflow (this has not been implemented for `cdf_fast` yet). Note also that `cdf_fast` does not currently implement all the necessary fast subroutines (notably FFT-based polynomial multiplication) to reach the complexity claimed by [1, Thm. 1.2]. Nonetheless, significant speedups compared to `cdf_rec` and `cdf_pos_rec` on high-dimensional instances are already observed, as detailed in the next section.

```
function cdf_rec(R::T, d::Int, nu::Vector{T}, sigma::Vector{T},   N::Int) where
     {T<:Real}

function cdf_rec_exponent(R::Float64, d::Int, nu::Vector{Float64},sigma::Vector
    {Float64}, N::Int; max_expo::Int=800)

function cdf_pos_rec(R::T, d::Int, nu::Vector{T}, sigma::Vector{T}, N::Int)
    where {T<:Real}

function cdf_pos_rec_exponent(R::Float64, d::Int, nu::Vector{Float64}, sigma::
    Vector{Float64}, N::Int; max_expo::Int=800)

function cdf_fast(R::T, d::Int, nu::Vector{T}, sigma::Vector{T},  N::Int, t::
    Int) where {T<:Real}
```

We also provide a MATLAB implementation of CdfRec and CdfPosRec in machine double precision, which is targeted for moderate precision and engineering applications. This allows for a more direct comparison with the state-of-the-art package of Das [3].

One of the main practical benefits of the power series approach is that truncation can be controlled by explicit analytic bounds. Rather than exposing a raw number of series terms as a user parameter, the next version of the package will accept a target tolerance and computes a truncation order N that is sufficient for that tolerance. Furthermore, the rounding error analysis in [1] allows for considering also a bound for the rounding error.

3 Numerical Experimental Evaluation

The goal of the numerical experiments is not only to measure execution times, but to evaluate the package as a mathematical software. We therefore focus on three questions:

1. How do the dimension d and the parameters σ_i, ν_i, q influence the practical performances of our algorithms in terms of computation time and accuracy?
2. Does the positive recurrence-based routine `cdf_pos_rec` truly offer superior floating-point rounding error performance compared to `cdf_rec` in practice?
3. In which regimes does the accelerated routine `cdf_fast` become advantageous?

The benchmarks are organized around families of instances chosen to stress different aspects of the problem. Typical dimensions range from small and moderate values, where all routines are expected to succeed, to larger values for which coefficients generation becomes more expensive. The package `Quadratiques.jl` [2] is distributed together with benchmark scripts and input data in the `examples/` directory, so that reported timings and numerical comparisons can be regenerated systematically. Results reported below were obtained using a laptop equipped with an Intel i7-1185G7 CPU (4 cores, 4.800GHz), 32GB of RAM, running Ubuntu 22.04.5 LTS and Julia v1.11.1.

3.1 Low-dimensional test cases

In this first numerical experiment, implemented in `ExamplesCdfPosRec.jl`, we evaluate the performance of `cdf_rec_exponent` and `cdf_pos_rec_exponent`. Below is a minimal working example of how to invoke these routines:

```
using Quadratiques

nu = [0.2, -0.5, 1.0]
sigma = [1.0, 0.7, 1.5]
d = length(nu)

q = 4.0
R = sqrt(q)
N = 80

p_rec = cdf_rec_exponent(R, d, nu, sigma, N)
p_pos_rec = cdf_pos_rec_exponent(R, d, nu, sigma, N)
```

The benchmark is composed of a mixed set of synthetic examples, constructed directly in the (R, d, ν, σ) parametrization used by the implementation and a subset of external examples taken from the Das benchmark collection [3]. It was designed to cover a range of numerical regimes with a small but structured set of test cases on which `cdf_rec_exponent` and `cdf_pos_rec_exponent` are compared in double-precision floating-point arithmetic. The first synthetic examples were organized so as to vary the difficulty progressively: isotropic central cases ($\nu_i = 0$ and $\sigma_i = \sigma$ for all i) were used as baselines; anisotropic central cases ($\nu_i = 0,\ \forall\ i$) were then introduced with increasing dispersion in the standard deviations σ_i; anisotropic noncentral cases were also obtained by fixing the spectrum and increasing the magnitude of ν. Additional cases in dimension $d = 30$ were included to test whether the same trends persist in a higher-dimensional setting. Finally, four challenging cases illustrate how N grows in presence of small σ_i: `hard-1` (anisotropic central with $d = 10$ and $\min_i \sigma_i = 0.05$), `hard-2` (same with $d = 30$), `hard-3` (anisotropic central with $d = 10$ and $\min_i \sigma_i = 0.01$) and `hard-4` (anisotropic noncentral with $d = 30$ and

$\min_i \sigma_i = 0.01$). In addition, the external Das' examples provide an independent comparison set representative of instances already used in the literature.

For each test case, the benchmark reads the tuple (R, d, ν, σ) and first computes a reference value p_{ref} for $F_Q(q) = g(1)$ using `cdf_pos_rec` in 1024-bit `BigFloat` arithmetic with a conservatively large truncation order $N_{\text{ref}} = 10^6$. Then, a calibration routine also based on `cdf_pos_rec` with `BigFloat` returns an optimal truncation order N so that the computed value stays within the requested relative tolerance `errmax` $= 2^{-53} \approx 10^{-16}$ with respect to p_{ref}. The two timed methods, `cdf_pos_rec_exponent` and `cdf_rec_exponent`, are finally both run in double-precision arithmetic with the calibrated truncation order N. For each case, the table reports:

- the reference probability p_{ref},
- the calibrated truncation order N,
- the execution times $\text{T}_{\texttt{posrec}}$ and $\text{T}_{\texttt{rec}}$ of `cdf_pos_rec` and `cdf_rec`,
- the number of accurate digits Dig. $_{\texttt{posrec}}$ and Dig. $_{\texttt{rec}}$ for both routines, computed from the relative errors,

$$\text{digits} = \max\big(0, -\log_{10}(\text{relative error})\big).$$

Case	Type	d	p_{ref}	N	$\text{T}_{\texttt{posrec}}$ (s)	$\text{T}_{\texttt{rec}}$ (s)	Dig. $_{\texttt{posrec}}$	Dig. $_{\texttt{rec}}$
S1	`easy`	10	0.3555	30	1.01e-5	5.78e-5	15.9	15.9
S2	`easy`	30	0.4908	50	3.41e-5	1.71e-4	15.9	15.5
S3	`aniso-1`	10	0.9974	70	2.38e-5	1.61e-4	14.9	10.3
S4	`aniso-2`	10	0.8746	100	3.33e-5	2.38e-4	14.5	4.5
S5	`aniso-3`	10	0.5960	150	4.99e-5	3.67e-4	14.2	0.0
S6	`aniso-4`	10	0.2553	330	1.18e-4	8.68e-4	15.8	0.0
S7	`noncent-1`	10	0.5799	150	5.29e-5	3.85e-4	14.7	0.0
S8	`noncent-2`	10	0.5288	150	5.33e-5	3.82e-4	14.8	0.1
S9	`noncent-3`	10	0.4352	150	5.21e-5	3.85e-4	14.0	0.4
S10	`noncent-4`	10	0.2957	150	5.34e-5	3.84e-4	14.1	1.0
S11	`aniso30-2`	30	0.9606	200	1.46e-4	1.36e-3	14.0	0.0
S12	`aniso30-3`	30	0.5599	340	2.49e-4	2.45e-3	14.2	0.0
S13	`noncent30-2`	30	0.5068	340	2.50e-4	2.44e-3	13.7	0.0
S14	`noncent30-4`	30	0.3383	340	2.48e-4	2.49e-3	14.6	0.0
S15	`hard-1`	10	4.451e-04	7,910	2.93e-3	2.30e-2	12.8	0.0
S16	`hard-2`	30	2.162e-13	7,900	6.03e-3	7.44e-2	12.8	0.0
S17	`hard-3`	10	4.101e-06	183,500	9.94e-2	6.21e-1	10.7	0.0
S18	`hard-4`	30	4.515e-23	183,500	1.96e-1	1.80e+0	10.7	0.0
D1	`das`	3	0.4936	30	9.04e-6	2.90e-5	15.9	15.0
D5	`das`	8	0.5913	60	1.99e-5	1.18e-4	15.2	11.4
D9	`das`	10	0.5848	80	2.87e-5	1.93e-4	14.4	10.0
D11	`das`	34	0.5736	210	1.79e-4	1.56e-3	15.5	0.0

For the easy isotropic cases, both methods are highly accurate. For the anisotropic cases, the results show a transition from similarity (`aniso-1`) to a clear advantage to `cdf_pos_rec` (`aniso-2`, `aniso-3`, `aniso-4`), while the noncentrality transition shows the progressive loss of accuracy with `cdf_rec` as the noncentral shift increases. Finally, the four hard cases show that `cdf_pos_rec` still makes it possible to compute the probability in `Float64` arithmetic to at least 10 digits of accuracy in less than a second, even when several thousands of terms are needed.

3.2 High-dimensional test cases

Theorem 1.2 in [1] predicts that for large d and N, CdfFast is asymptotically faster than CdfPosRec (for a given *absolute* error target). In order to illustrate this point, it is crucial to generate instances that are as uniform as possible w.r.t. the dimension d (ranging from 100 to 10,000 in what follows) to allow for consistent comparisons. Moreover, the value $g(1)$ must

lie sufficiently far away from 0 and 1 for the target absolute error model to make sense. In the numerical experiment implemented in `ExamplesCdfFast.jl`, for values of d ranging from 100 to 10,000, we generate uniformly distributed random values σ'_i between 1 and 2. The ν'_i follow a standard normal distribution. The final values σ_i and ν_i are obtained by rescaling σ'_i, ν'_i by λ^{-1}, where $\lambda = \left(\sum_{i=1}^{d} \sigma_i'^2 + \nu_i'^2\right)^{\frac{1}{2}}$. For all the tested values of d, the resulting $g(1)$ is close to 0.5, and the optimal truncation index N always lies between d and $2d$, for a target absolute accuracy of 10^{-10}. We therefore run `cdf_fast` and `cdf_pos_rec` on those instances with $N = 2d$ and `BigFloat` with 64-bit precision[4]. The additional parameter t of `cdf_fast` was also set to 25.

```
using Quadratiques
setprecision(BigFloat, 64)

d = 1000
N = 2*d
R = BigFloat(1)

sigma = BigFloat.(sort(rand(d))) .+ 1
nu = BigFloat.(randn(d))
lambda = sqrt(sum(nu .^ 2 + sigma .^ 2))
sigma /= lambda
nu /= lambda

t = 25

p = cdf_fast(R, d, nu, sigma, N, t)
```

The obtained numerical results show that the target accuracy of 10^{-10} is satisfied on all instances, with a slight advantage to `cdf_pos_rec`. However, as soon as $d = 300$, `cdf_fast` runs faster and scales much better with high dimension d. The speedups depicted in Figure 1 reach a factor of 30 for $d = 10,000$, reducing the execution time for that case from 104 s to 3.5 s. It also reveals a characteristic "sawtooth pattern" due to the use of FFT for the series exponentiation in CdfFast.

Although this numerical experiment is far from representing all the diversity of the possible instances of the problem in high dimension d, the promising results give practical consistence to the theoretical complexity results proved in [1, Sec. 4], and a motivation for future improvements in both the method and implementation.

4 Conclusion and Perspectives

The main contribution of the paper is not a new theoretical derivation, but the realization of the ideas in [1] as reusable software. We described a common implementation framework together with three evaluation modes. The experiments show that the positivity-preserving mode provides the best overall robustness, while the fast mode opens promising perspectives for larger and more demanding computations. More broadly, the package illustrates how holonomic and symbolic-numeric ideas can be turned into effective mathematical software. This is, in our view, the main message of the paper: the analytic structure of a difficult probabilistic problem can be exploited not only to derive elegant formulas, but to build dependable computational tools that are useful in practice.

Several directions remain for future work. On the algorithmic side, it would be useful to refine automatic mode selection so that the software can infer from the instance itself whether

[4] Using `Float64` arithmetic leads to overflows, even with `cdf_pos_rec_exponent`. In contrast, the much larger exponent range of `BigFloat` turns out to be sufficient for this experiment.

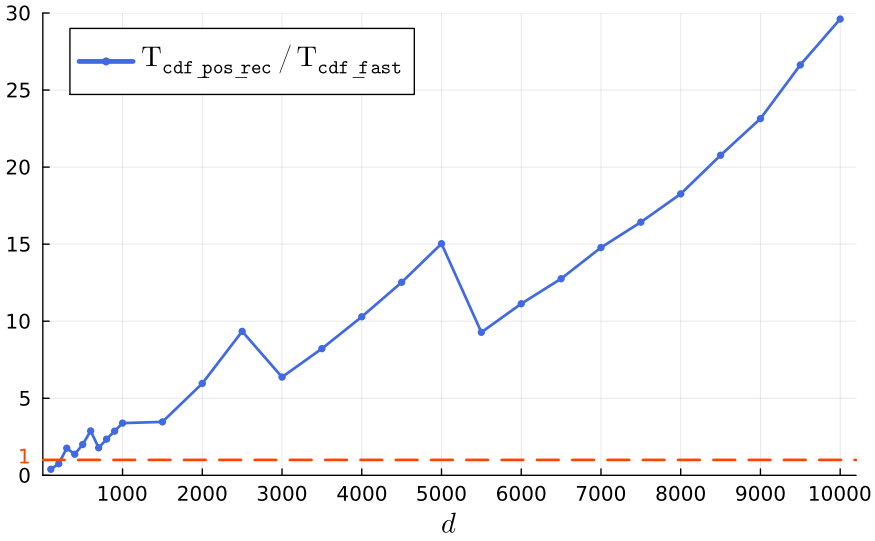

Fig. 1: Speedups obtained using CdfFast compared to CdfPosRec.

robustness-oriented or speed-oriented evaluation is preferable. On the implementation side, additional vectorization and parallelization strategies could further improve throughput in batch settings. On the application side, tighter integration with operational risk-assessment workflows and uncertainty-propagation pipelines would broaden the practical reach of the package.

References

1. Arzelier, D., Bréhard, F., Joldes, M.: Fast and reliable evaluation of the distribution of quadratic forms of Gaussian random variables. In: ISSAC 2026: Proceedings of the 51st International Symposium on Symbolic and Algebraic Computation, Oldenbourg, Germany, July 13-17, 2026. ACM Press (2026), to appear. https://hal.science/hal-05507599/
2. Arzelier, D., Bréhard, F., Joldes, M.: Fast and reliable evaluation of the distribution of quadratic forms of gaussian random variables. https://gitlab.laas.fr/arzelier/quadratic (2026), gitLab repository, accessed 19 June 2026
3. Das, A.: New methods to compute the generalized chi-square distribution. Journal of Statistical Computations **95**(3), 1–36 (May 2025)
4. Davies, R.: Numerical inversion of a characteristic function. Biometrika **60**(2), 415–417 (August 1973), https://doi.org/10.1093/biomet/60.2.415
5. Imhof, J.: Computing the distribution of quadratic forms in normal variables. Biometrika **48**(3/4), 419–426 (December 1961). https://doi.org/10.1093/biomet/48.3-4.419
6. Johnson, N., Kotz, S.: Continuous univariate distributions, volume 2. Wiley Series in Probability and Mathematical Statistics, John Wiley & Sons, New York, NY, USA (1970), https://doi.org/10.1080/00224065.1996.11979675
7. Kuonen, D.: Saddlepoint approximations for distributions of quadratic forms in normal variables. Biometrika **86**(4), 929–935 (12 1999). https://doi.org/10.1093/biomet/86.4.929, https://doi.org/10.1093/biomet/86.4.929
8. Mathai, A.M., Provost, S.B.: Quadratic forms in random variable: Theory and Applications. Statistics: Texbooks and Monographs, Marcel Dekker, New York, USA (1992), https://doi.org/10.2307/2290674
9. Ruben, H.: Probability content of regions under spherical normal distributions IV: the distribution of homogeneous and non-homogeneous quadratic functions of normal variables. The Annals of Mathematical Statistics **33**, 542–570 (1962)
10. Serra, R., Arzelier, D., Joldes, M., Lasserre, J., Rondepierre, A., Salvy, B.: Fast and accurate computation of orbital collision probability for short-term encounters. Journal of Guidance Control and Dynamics **39**(9), 1009–1021 (2016), https://doi.org/10.2514/1.G001353