Earthline Journal of Mathematical Sciences
E-ISSN: 2581-8147
Volume 16, Number 3, May 2026, Pages 487-513
https://doi.org/10.34198/ejms.16326.33.487513

Study, Implementation, and Application of the Bivariate Cosine Gaussian Distribution

Julien Samyn[1], Soan Bailly[2] and Christophe Chesneau[3,*]

[1] Department of Mathematics, LMNO, University of Caen-Normandie, 14032 Caen, France
e-mail: juliensamyn14@gmail.com

[2] Department of Mathematics, LMNO, University of Caen-Normandie, 14032 Caen, France
e-mail: soan.bailly@orange.fr

[3] Department of Mathematics, LMNO, University of Caen-Normandie, 14032 Caen, France
e-mail: christophe.chesneau@gmail.com

Abstract

This article presents a comprehensive study of a newly introduced probability distribution, the bivariate cosine Gaussian distribution, ranging from theory to practical application. This distribution is characterized by a single parameter, which controls a trigonometric component. It allows for the modelling of nonlinear structural dependencies that are inaccessible to classical Gaussian models. In this study, we translate this mathematical framework into a suite of statistical tools implemented in R, developing a data generator that uses accept-reject sampling and a maximum likelihood estimation method secured by particle swarm optimization. We also develop a robust goodness-of-fit test based on the energy distance via bootstrap. We validate the relevance of our approach using a real-world dataset from electric motor engineering. Combining the energy test with model selection via information criteria demonstrates that the oscillatory nature of this distribution is particularly well-suited to capturing threshold effects and discrete variations imposed by motor control algorithms.

1 Introduction

In multivariate statistics, the bivariate Gaussian distribution is considered the gold standard due to its simplicity and analytical properties. However, its fundamentally elliptical structure and reliance on linear correlation as the sole measure of dependence mean that it is ineffective for modelling complex physical phenomena. In fields such as engineering, robotics, and sensor analysis, data often exhibits clustered tiers, threshold effects, or mutual exclusion relationships. To address these limitations, several directions have been explored. See [1–5]. In particular, a recent theoretical work by C. Chesneau [2] introduced a tractable Gaussian distribution incorporating an oscillatory cosine term.

This article is based on [2] and translates its mathematical framework into operational statistical tools

Received: March 7, 2026; Accepted: April 16, 2026; Published online: April 24, 2026
2020 Mathematics Subject Classification: 62H15, 62P30.
Keywords and phrases: bivariate cosine Gaussian distribution, modified Gaussian distribution, nonlinear structural dependence, particle swarm optimization, parametric bootstrap, Goodness-of-fit.
*Corresponding author

for real-world applications. Specifically, the objective is to apply the theory in [2] to data science by creating an end-to-end algorithmic processing pipeline in the R programming language. This pipeline covers all stages, from simulation to inference and rigorous statistical validation. To demonstrate the practical value of the model, it is applied to optimizing the performance of electric motors. Using strict selection criteria, we demonstrate the empirical superiority of the model over classical bivariate models by showing that its oscillatory geometry uniquely captures the discrete tiers and zones of mutual exclusion inherent in motor control algorithms.

The remainder of this paper is organized as follows: Section 2 introduces the theoretical framework of the distribution of interest. Section 3 details the accept-reject simulation procedure. Section 4 presents the statistical inference and the particle swarm optimization (PSO) for parameter estimation. Section 5 outlines the goodness-of-fit testing methodology based on the energy distance. Section 6 describes the model selection framework using information criteria. Section 7 presents the empirical results on the motor control dataset. Finally, Section 8 concludes the article.

2 Theoretical Framework

This section is entirely based on the foundational work of C. Chesneau [2] introducing the bivariate cosine Gaussian (BCG) distribution.

2.1 Definition of the BCG distribution

Let $\theta \in \mathbb{R}$ be an angle parameter. The BCG distribution is defined by the following probability density function on $\mathbb{R}^2$:

$$f(x,y) = C^{-1} e^{-\frac{x^2+y^2}{2}} \cos^2(\theta xy), \quad (x,y) \in \mathbb{R}^2.$$

To ensure that the integral of this probability density function over $\mathbb{R}^2$ equals 1, the normalizing constant C was mathematically determined in [2] as

$$C = \pi \left(1 + \frac{1}{\sqrt{1+4\theta^2}}\right).$$

It is worth noting that modifications to the bivariate Gaussian distribution are well known. See, for example, [1, 3–5]. However, examples incorporating a trigonometric component, such as the BCG distribution, are rare. Some of its fundamental properties are summarised below.

2.2 Fundamental properties

The probability density function f possesses several remarkable structural properties demonstrated in [2]:

- **Symmetry**: Since the cosine and squared functions are even, f exhibits axial symmetry, i.e., $f(-x, y) = f(x, -y) = f(x, y)$, as well as symmetry by variable exchange $f(x, y) = f(y, x)$.

- **Limiting case ($\theta = 0$)**: If $\theta = 0$, the constant C becomes 2π and the $\cos^2(0)$ term equals 1. The probability density function f then exactly reduces to that of the standard independent bivariate Gaussian distribution.

2.3 Marginal and conditional distributions

One of the major results in [2] lies in the explicit expression of its marginal distributions. Let (X, Y) be a random vector that follows the BCG distribution. By integrating the probability density function $f(x, y)$ with respect to y, the marginal probability density function of X is written as

$$g(x) = D^{-1} e^{-\frac{x^2}{2}} \left(1 + e^{-2\theta^2 x^2}\right), \quad x \in \mathbb{R},$$

where D is a new normalizing constant specific to the marginal, defined by

$$D = \sqrt{2\pi} \left(1 + \frac{1}{\sqrt{1 + 4\theta^2}}\right).$$

As highlighted by C. Chesneau [2], this marginal probability density function g can be interpreted as a mixture of two Gaussian distributions (Gaussian mixture). It can generate bimodality (two distinct peaks) when $\theta \neq 0$, illustrating the mass concentration induced by the cosine.

Similarly, the conditional distribution of Y given $X = x$ is given by

$$k(y|x) = \sqrt{\frac{2}{\pi}} \frac{e^{-\frac{y^2}{2}} \cos^2(\theta x y)}{1 + e^{-2\theta^2 x^2}}, \quad y \in \mathbb{R}.$$

2.4 Dependence analysis

The introduction of the oscillatory term disrupts the classical notion of independence. According to [2]:

- **Stochastic independence**: The variables X and Y are stochastically independent, i.e., $f(x, y) = g(x)g(y)$ for any $(x, y) \in \mathbb{R}^2$, if and only if $\theta = 0$.

- **Linear independence**: Surprisingly, for any $\theta \in \mathbb{R}$, the linear covariance remains strictly zero, i.e., $\mathrm{Cov}(X, Y) = 0$.

This implies that the dependence captured by the BCG distribution is purely nonlinear and "oscillatory and circular" in nature, a phenomenon undetectable by classical metrics like Pearson correlation.

3 Accept-reject Simulation

3.1 Simulation procedure

Since the BCG distribution does not have an easily invertible cumulative distribution function, the generation of a dataset relies on the accept-reject simulation procedure detailed by C. Chesneau [2].

The approach proposed by the author consists of using the standard independent bivariate Gaussian distribution, with probability density function $\phi(x,y) = (1/(2\pi))e^{-\frac{x^2+y^2}{2}}$, $(x,y) \in \mathbb{R}^2$, as the proposal distribution (or instrumental distribution). By comparing the target probability density function $f(x,y)$ to this reference distribution $\phi(x,y)$, we observe the following upper bound:

$$Cf(x,y) = 2\pi\phi(x,y)\cos^2(\theta xy) \leq 2\pi\phi(x,y)$$

since the squared cosine function is naturally bounded by 1.

The generation algorithm thus boils down to a particularly elegant procedure: we draw a candidate (x^*, y^*) from the standard bivariate Gaussian distribution, along with a value u from a standard uniform distribution on the interval $(0,1)$. This candidate is accepted if and only if the acceptance condition is met:

$$u < \frac{Cf(x^*,y^*)}{2\pi\phi(x^*,y^*)}$$

that is

$$u < \cos^2(\theta x^* y^*).$$

Implementation in R. We have implemented this algorithm numerically in the R programming language. This implementation creates sampling loops of Gaussian candidates until the desired sample size n is reached, enabling us to simulate the studied distribution for any value of θ:

```
rbcg <- function(n, theta = 1) {
  mu <- c(0, 0)
  Sigma <- diag(2)
  out <- matrix(NA_real_, nrow = 2, ncol = n)
  rownames(out) <- c("X", "Y")
  i <- 1
  while (i <= n) {
    v <- as.numeric(MASS::mvrnorm(1, mu = mu, Sigma = Sigma))
    if (runif(1) <= cos(theta * v[1] * v[2])^2) {
      out[, i] <- v
      i <- i + 1
    }
  }
  out
}
```

Listing 1: Simulation algorithm in R.

Generation of the proposal distribution (MASS Package). The practical implementation of this algorithm relies on the ability to computationally simulate candidates from the standard independent bivariate Gaussian distribution. To perform this step rigorously, we use the `mvrnorm` function from the `MASS` library in R. See [6].

This function is specifically designed for sampling multivariate Gaussian random vectors. In our algorithm, the function is called via the command `MASS::mvrnorm(1, mu = c(0,0), Sigma = diag(2))`, whose parameters exactly reflect the theoretical framework:

- `n = 1`: generates a single candidate vector (x^*, y^*) at each iteration of the `while` loop.
- `mu = c(0, 0)`: defines the mathematical expectation vector, which perfectly centers our proposal distribution at the origin $(0, 0)$.
- `Sigma = diag(2)`: defines the variance-covariance matrix. The use of the 2-dimensional identity matrix imposes a unit variance (1) on the X and Y axes, and a strictly zero covariance (0) off the diagonal. This guarantees perfect independence between the two variables of the generated candidate.

The use of the `mvrnorm` function thus allows for a highly faithful translation of the mathematical proposal model into a reliable computational generator. The candidates generated this way are then subjected to the trigonometric filter of the rejection step.

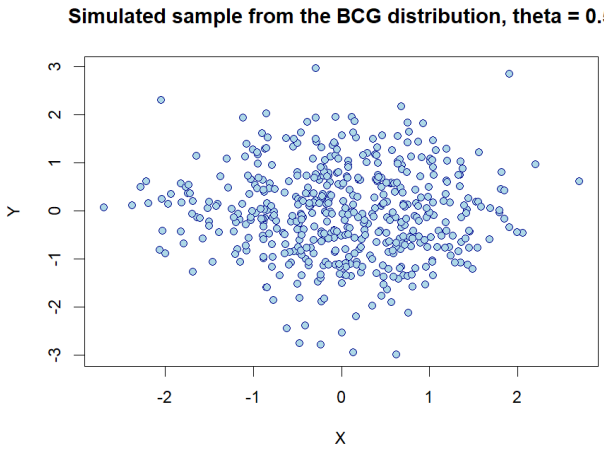
3.2 Sample generation and visualization of dependence structures

To validate our implementation of the accept-reject algorithm, we used our `rbcg` function to generate samples of size $n = 500$. To observe the influence of the oscillatory component on the dependence structure, we varied the parameter θ across the values $0.5, 1, 2$, and 5. The corresponding scatter plots are presented in Figure 1.

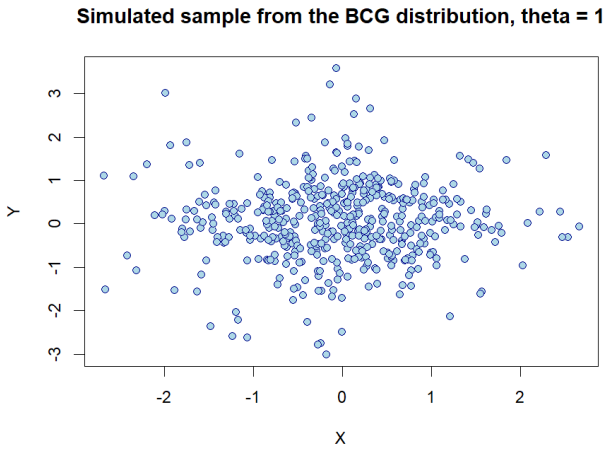
```
simulation <- rbcg(500, theta = 0.5)

plot(simulation[1,], simulation[2,],
main = "Simulated sample from the BCG distribution,
theta = 0.5", pch = 21, col = "darkblue", bg = "lightblue", xlab = "X",
ylab = "Y")
```

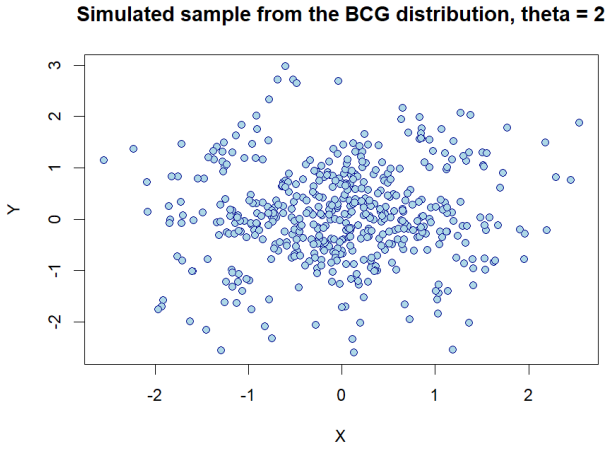
Listing 2: Simulation script and plotting of a scatter plot in R.



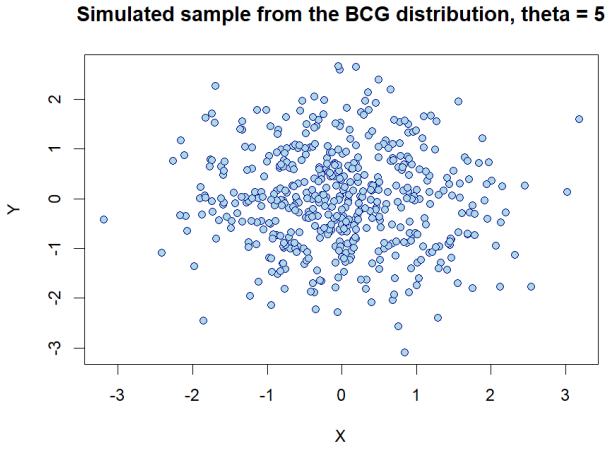
(a) $\theta = 0.5$



(b) $\theta = 1$



(c) $\theta = 2$



(d) $\theta = 5$

Figure 1: Simulated scatter plots via the accept-reject algorithm ($n = 500$) for the BCG distribution. The increase of θ fragments the initial "Gaussian cloud".

4 Statistical Inference and Estimation of Parameter θ

4.1 Negative log-likelihood function

To estimate the parameter θ on a real dataset of size n, denoted $(x_1, y_1), \ldots, (x_n, y_n)$, the reference method is the maximum likelihood method.

By definition, under the assumption of independent and identically distributed (i.i.d.) observations,

the likelihood function $L(\theta)$ is written as the following product:

$$L(\theta) = \prod_{i=1}^{n} f(x_i, y_i|\theta),$$

where $f(x_i, y_i|\theta) = f(x_i, y_i)$ and θ specifies the unknown parameter. To simplify the calculations and transform this product into a sum, it is a standard practice to take the natural logarithm, which yields the log-likelihood function $\ell(\theta) = \ln L(\theta)$ given by

$$\ell(\theta) = \sum_{i=1}^{n} \ln\big(f(x_i, y_i|\theta)\big).$$

By substituting $f(x_i, y_i|\theta)$ with the equation of our distribution, we obtain

$$\ell(\theta) = \sum_{i=1}^{n} \ln\left(C^{-1} e^{-\frac{x_i^2+y_i^2}{2}} \cos^2(\theta x_i y_i)\right).$$

Using the basic properties of the logarithm, this expression is formally expanded as follows:

$$\ell(\theta) = \sum_{i=1}^{n} \left[-\ln(C) - \frac{x_i^2+y_i^2}{2} + \ln\big(\cos^2(\theta x_i y_i)\big)\right].$$

In numerical optimization, the standard convention for algorithms (such as Brent's method or PSO) is to search for a global minimum. Maximizing the log-likelihood function is strictly equivalent to minimizing its opposite, called the negative log-likelihood (NLL) function, given by

$$\mathrm{NLL}(\theta) = -\ell(\theta) = -\sum_{i=1}^{n} \left[-\ln(C) - \frac{x_i^2+y_i^2}{2} + \ln\big(\cos^2(\theta x_i y_i)\big)\right].$$

Numerical stability trick. From a computational perspective, this theoretical formula has a fatal flaw. If for a given point (x_i, y_i), the term $\theta x_i y_i$ equals an odd multiple of $\pi/2$, then $\cos^2(\theta x_i y_i) = 0$. Calculating $\ln(0)$ will then return infinity (`-Inf` in R), which will irreparably crash the optimization algorithm.

To bypass this issue and ensure code robustness, we introduce an infinitely small regularization constant $\epsilon = 10^{-300}$ inside the logarithm. The final function implemented for minimization thus becomes:

$$\mathrm{NLL}_{optim}(\theta) = -\sum_{i=1}^{n} \left[-\ln(C) - \frac{x_i^2+y_i^2}{2} + \ln\big(\cos^2(\theta x_i y_i) + \epsilon\big)\right].$$

Implementation in R. To translate this mathematical equation faithfully into code, we opted for a modular approach. We programmed the calculation of the negative log-likelihood via a generic `nll` function.

This function takes `dens_fun` as an argument, which represents the probability density function to be evaluated. In the context of our study, we pass our `bcg` function by default, which implements the exact formula for $f(x, y|\theta)$. The numerical safeguard ϵ mentioned earlier is handled using the `pmax` function in R, which prevents the probability density function from dropping below a critical threshold (here set to 10^{-300}) before computing the logarithm:

```r
# 1. Probability density function of our distribution (f(x,y))
bcg <- function(par, x, y) {
  theta <- par[1]
  C <- pi * (1 + 1 / sqrt(1 + 4 * theta^2))
  (1 / C) * exp(- (x^2 + y^2) / 2) * cos(theta * x * y)^2
}

# 2. Calculation of the NLL
nll <- function(par, data, dens_fun = bcg, eps = 1e-300) {
  x <- data[1, ]
  y <- data[2, ]

  # Calculate the probability density function for all points
  vals <- dens_fun(par, x, y)

  # Numerical safeguard before log
  vals <- pmax(vals, eps)

  return(-sum(log(vals)))
}
```

Listing 3: Modular implementation of the NLL function and the bivariate probability density function.

4.2 PSO

Due to the oscillations of the cosine term, the topology of the likelihood function of this distribution can present multiple local minima. To ensure robust estimation, a PSO metaheuristic was implemented. This algorithm simulates a population of solutions that explore the parameter space in order to converge towards the global minimum.

```r
# Objective function
objective <- function(par) nll(par, data = D, dens_fun = bcg)

# Launching PSO
fit <- pso(func = objective, lim_inf = c(0), lim_sup = c(100),
           S = 120, N = 400, prop = 0.25)
theta_estime <- fit$par
```

Listing 4: Using the PSO algorithm for estimation.

5 Goodness-of-fit Tests

Classical goodness-of-fit tests (such as the Kolmogorov-Smirnov test) prove to be inadequate for verifying whether an empirical dataset follows the BCG distribution. This is because they generalise poorly to dimensions greater than one and require a cumulative distribution function in a simple analytical form. We therefore opted for a modern, robust multivariate analysis approach: the energy distance test.

5.1 The energy statistic (energy distance)

The energy distance is a measure of global statistical distance between two probability distributions. In our context, we adopt a two-sample approach (2-sample test) to compare:

1. The real empirical sample (the observed data of size n).
2. A "perfect" sample of the same size n, computationally generated by our accept-reject algorithm, using the parameter $\hat{\theta}$ previously estimated on the real data via PSO.

The R package `energy` (via the `eqdist.e` function) calculates the energy statistic T_{obs}, evaluating the structural discrepancy between these two point clouds.

5.2 Resolution via parametric bootstrap

The use of an estimated parameter, $\hat{\theta}$, to generate the second sample alters the asymptotic distribution of the test statistic under the null hypothesis. The standard p-value returned by the test is therefore no longer valid. To correct this bias, we implemented a rigorous parametric bootstrap procedure:

- **Initialization:** Calculation of the observed statistic T_{obs} against a sample simulated with the initial $\hat{\theta}$.
- **Iterations ($B = 199$):** At each iteration b, we generate a new so-called "bootstrap" sample with the initial $\hat{\theta}$. We then re-estimate a new parameter $\hat{\theta}_b$ specifically on this sample via PSO. Finally, we calculate the statistic $T_{boot}^{(b)}$ against a new sample generated with $\hat{\theta}_b$.
- **Decision:** The final p-value is obtained by the proportion of cases where the bootstrap statistics exceed the initially observed statistic:

$$p = \frac{1 + \sum_{b=1}^{B} \mathbf{1}(T_{boot}^{(b)} \geq T_{obs})}{B + 1}.$$

Hypothesis testing framework. Formally, the goodness-of-fit procedure is conducted under the composite null hypothesis

$$\mathcal{H}_0 : \text{ there is, } \theta \in \mathbb{R} \text{ such that } (X, Y) \text{ follows the BCG distribution,}$$

against the general alternative

$$\mathcal{H}_1 : (X, Y) \text{ does not follow any BCG distribution.}$$

Since the parameter θ is unknown, it is first estimated from the data, yielding $\hat{\theta}$. The parametric bootstrap then approximates the null distribution of the test statistic under $\mathcal{H}_0$ while accounting for parameter estimation. In each bootstrap replication, both the data-generating step and the re-estimation step are reproduced, thereby mimicking the full inferential procedure.

Implementation in R. This entire logic, coupling PSO optimization for re-fitting, accept-reject simulation, and energy distance, was coded as follows:

```
# 1. Calculate the energy statistic between two samples
energy_stat_2sample <- function(D1, D2) {
  Z <- rbind(t(D1), t(D2))
  sizes <- c(ncol(D1), ncol(D2))
  T_stat <- energy::eqdist.e(Z, sizes = sizes)
  return(as.numeric(T_stat))
}

# 2. Main test function with Bootstrap
gof_energy_boot <- function(D, lim_inf = 0, lim_sup = 100,
                            S = 120, N = 400, prop = 0.25,
                            B = 199, seed = 123) {
  set.seed(seed)
  n <- ncol(D)

  # A) Estimation on original data via PSO
  theta_hat <- theta_hat_pso(D, lim_inf, lim_sup, S, N, prop)

  # B) Calculation of the observed statistic
  Y0 <- rbcg(n, theta_hat)
  T_obs <- energy_stat_2sample(D, Y0)

  # C) Parametric Bootstrap loop WITH re-fit
  T_boot <- numeric(B)
  for (b in seq_len(B)) {
    # Generation of the bootstrap sample
    Db <- rbcg(n, theta_hat)
```

```
    # RE-ESTIMATION of the parameter (crucial to correct bias)
    theta_b <- theta_hat_pso(Db, lim_inf, lim_sup, S, N, prop)

    # New reference sample and energy calculation
    Yb <- rbcg(n, theta_b)
    T_boot[b] <- energy_stat_2sample(Db, Yb)
  }

  # D) Calculation of the final p-value
  pval <- (1 + sum(T_boot >= T_obs)) / (B + 1)

  return(list(theta_hat = theta_hat, T_obs = T_obs, p_value = pval))
}
```

Listing 5: Goodness-of-fit test by the energy distance and parametric bootstrap.

6 Model Selection and Information Criteria

To validate the practical utility of our trigonometric distribution, we must prove that it can model real data more precisely than existing classical distributions. We therefore set up a model selection procedure in which we compared our distribution with nine other reference bivariate distributions. These were selected in order to rule out specific alternative hypotheses. Gaussian distributions (with and without covariance) were used as a baseline to ensure that dependence was not simply linear correlation. The bivariate Student's t-distribution, which is characterised by its "heavy tails", ensures that the superiority of our model truly stems from its oscillatory geometry and not merely from its tolerance of extreme values. Finally, robust independent distributions (Laplace, Cauchy and logistic) enable us to verify that the complexity of the scatter plot reveals genuine joint dependence and not merely atypical marginal distributions.

Directly comparing the maximum log-likelihood ($\ln \hat{L}$) of these models would be statistically biased. Indeed, a model with many parameters (such as the bivariate Student's t-distribution, which has 6) will mechanically always fit the data better than a simple model (like our distribution, which has only one, θ).

To overcome this overfitting problem, we use information criteria that impose a trade-off between the goodness-of-fit and the complexity of the model.

1. The Akaike information criterion (AIC). The AIC estimates the relative information loss when a model is used to represent the real data-generating process. It is defined by

$$\mathrm{AIC} = 2k - 2\ln(\hat{L}),$$

where k represents the number of estimated parameters of the model and $\hat{L}$ the maximum value of the likelihood function. The best model is the one that minimizes this value. The $2k$ term acts as a strict penalty against adding unnecessary parameters.

2. The Bayesian information criterion (BIC). The BIC relies on a similar approach but penalizes complexity much more heavily when the sample size n is large. It is given by

$$\text{BIC} = k\ln(n) - 2\ln(\hat{L}).$$

In our study on motor data (where n is often greater than several hundreds), the BIC is an extremely strict referee that will favor parsimony.

3. AIC differences (Δ_i) and Akaike weights (w_i). Since the absolute value of the AIC has no intrinsic meaning, we evaluate the models by calculating the AIC difference between each model i and the best model of the tournament (the one with the lowest score, AIC_{min}). It is defined by

$$\Delta_i = \text{AIC}_i - \text{AIC}_{min}.$$

As a general rule, a model with $\Delta_i > 10$ has no statistical support and can be discarded.

To make these results more intuitive, we transform these differences into Akaike weights. They normalize the scores to provide a probability, ranging from 0 to 1 (or 0% to 100%), that model i is the best among the set of tested models. It is given by

$$w_i = \frac{e^{-\frac{1}{2}\Delta_i}}{\sum_{j=1}^{M} e^{-\frac{1}{2}\Delta_j}},$$

where M is the total number of competing models (here $M = 10$). This probability w_i will serve as our primary indicator to identify the most suitable model for each studied pair of variables.

7 Results

7.1 Study on the "Motor Control Performance" dataset

To validate the practical utility of our model, we applied it to the Motor Control Performance Dataset [7]. This public dataset, consisting of 700 simulated observations, is specifically designed to evaluate the performance of Direct Torque Control (DTC) and the optimization of PID controllers in induction motors.

It gathers a multitude of critical metrics for evaluating motor control, divided into several categories:

- **Operational and electrical parameters:** time (*Time*), rotational speed of the motor (*Speed* in RPM), output torque (*Torque* in Nm), magnetic flux (*Flux* in Wb), as well as voltage (*Voltage*), current (*Current*), and output power (*Power*).

- **PID controller gains:** proportional (K_p), integral (K_i), and derivative (K_d) parameters, essential for regulation.
- **Performance and error metrics:** speed overshoot (*Speed Overshoot*), torque ripples (*Torque Ripple*) and flux ripples (*Flux Ripple*), integral squared error (*ISE*), and stabilization time after disturbance.

Before being fed into our inference and model selection algorithms (AIC/BIC), all quantitative variables were previously centered and scaled. The comparison script then evaluated all possible pairs of these variables (for example, Speed vs Power, Torque vs Control Constants) to isolate the combinations where the BCG distribution provided the best statistical fit.

7.2 Analysis approach and variable filtering

Faced with the richness of this dataset, we implemented a systematic "funnel" exploratory approach to identify the pairs of variables for which our mathematical model was the most relevant:

1. **Combinatorial exploration.** After cleaning the data (removing categorical or redundant variables), we retained 18 quantitative variables. We generated all possible combinations of pairs among these variables, resulting in a total of $\binom{18}{2} = 153$ distinct tested pairs.
2. **Goodness-of-Fit filtering.** Each pair was subjected to our energy distance goodness-of-fit test. We only kept the pairs showing a satisfactory p-value (non-rejection of the goodness-of-fit hypothesis) and for which the estimated parameter $\hat{\theta}$ was significantly different from zero. This second condition was crucial to exclude trivial cases where our model simply reduced to a standard bivariate Gaussian distribution. At the end of this step, 121 candidate pairs were validated.
3. **Validation via information criteria.** For these 121 remaining pairs, we calculated the AIC, BIC, and Akaike weights, in order to pit them in direct competition with the 9 other reference bivariate distributions.

Filtering summary. Out of the 153 initial pairs, and after going through our two statistical filters (energy then AIC/BIC), our trigonometric distribution established itself in 3 specific cases as by far the best probabilistic model according to the information criteria. We detail the results for one of these best fitting pairs below.

7.3 Results: validation and model selection

In accordance with our funnel approach, the 153 pairs of variables from the electric motor dataset were first subjected to the goodness-of-fit filter before being compared to the other distributions.

1. Validation via the energy test. The energy distance test (via parametric bootstrap) allowed us to identify the pairs for which the empirical distribution does not reject our trigonometric model (P-value > 0.05). To avoid the trivial case where the distribution would reduce to a classical Gaussian distribution, we also required the estimated parameter $\hat{\theta}$ to be significantly different from zero.

At the end of this stage, 121 pairs were statistically validated. Table 1 illustrates these results for three relevant pairs of variables characterizing the state and control of the machine.

Variable Crossing (Motor)	**Estimated parameter ($\hat{\theta}$)**	T_{obs}	**P-value**
Flux (Wb) vs Torque Ripple (Nm)	0.018	0.9280	0.84
Flux Ripple (Wb) vs Load Disturbance (Nm)	0.182	1.0692	0.82
Speed (RPM) vs Voltage (V)	0.084	1.1685	0.64

Table 1: Goodness-of-fit test results (energy) on a sample of pairs from the electric motor.

2. Competition and model selection (AIC/BIC). For these 121 candidate pairs, we then put our distribution in direct competition with 9 other classical bivariate distributions (Gaussian, Student, Laplace, etc.). The evaluation relies on the AIC and BIC, which penalize model complexity. These scores are converted into Akaike weights (w_i), representing the probability that the model is the best performing one.

The final filtering reveals that our model significantly outperforms classical distributions on several pairs of variables. Table 2 details the ranking for the pair Flux Ripple vs Load Disturbance, where our distribution vastly outperforms standard distributions.

Bivariate Model	k	**log-likelihood**	**AIC**	**BIC**	**Δ AIC**	**Akaike Weight (w_i)**
1. Bivariate Cosine Gaussian	**1**	**-1983.3**	**3968.7**	**3973.2**	**0.0**	**74.87 %**
2. Standard Gaussian (Baseline)	0	-1985.5	3971.0	3971.0	2.3	23.33 %
3. Bivariate Gaussian ($\mu = 0$)	3	-1985.5	3977.0	3990.7	8.3	1.16 %
4. Independent Gaussians	4	-1985.5	3979.0	3997.2	10.3	0.43 %
5. Bivariate Gaussian	5	-1985.5	3981.0	4003.8	12.3	0.16 %
6. Independent Students	6	-1986.1	3984.1	4011.4	15.4	0.03 %
7. Bivariate Student	6	-1986.8	3985.5	4012.8	16.8	0.02 %
8. Independent Logistics	4	-2003.2	4014.3	4032.5	45.6	0.0 %
9. Independent Laplaces	4	-2062.3	4132.6	4150.8	163.9	0.0 %
10. Independent Cauchys	4	-2254.8	4517.6	4535.8	548.9	0.0 %

Table 2: Ranking by information criteria on the Flux Ripple / Load Disturbance pair.

Physical interpretation. The success of our mathematical model on this specific pair can be explained by the nature of motor control (DTC or PID control). The controller tolerates variations in magnetic

flux (ripples) until an external load disturbance reaches the limits of its tolerance band (hysteresis). This regulation mechanism creates data structured in discrete clusters and tiers, with clear exclusion zones. Thanks to the geometric properties of its cosine, our distribution perfectly captures these mutual exclusions in a highly parsimonious manner ($k = 1$), whereas the classical Gaussian distribution draws a continuous, blurry ellipse that is unsuited to the discrete behavior of the machine.

8 Conclusion

This work demonstrates that the trigonometric extension of the Gaussian distribution proposed in [2] is a mathematically rigorous model, whose inference is fully achievable computationally. The statistical tools developed in R (PSO, parametric bootstrap on the energy distance, selection via Akaike weights) have allowed us to effectively confront this theory with reality.

The application to the electric motor dataset proves the practical value of the model: its oscillatory component gives it a unique ability to fit physical data exhibiting thresholds, discrete tiers, or periodic concentrations, where standard Gaussian models prove to be inadequate.

References

[1] Chesneau, C. (2023). On two-dimensional functions with an integral equal to zero over a rectangle: Application to a modified Gaussian distribution. *Asian Journal of Mathematics and Applications*, *5*, 1–23.

[2] Chesneau, C. (2025). Theory on a new bivariate trigonometric Gaussian distribution. *Innovation in Statistics and Probability*, *1*(2), 1–17. https://doi.org/10.64389/isp.2025.01223

[3] Jwo, D.-J., Cho, T.-S., & Biswal, A. (2023). Geometric insights into the multivariate Gaussian distribution and its entropy and mutual information. *Entropy*, *25*(8), 1177. https://doi.org/10.3390/e25081177

[4] Mahmoudi, E., & Mahmoodian, H. (2017). A new bivariate distribution obtained by compounding the bivariate normal and geometric distributions. *Journal of Statistical Theory and Applications*, *16*, 198–208. https://doi.org/10.2991/jsta.2017.16.2.5

[5] Mathai, A., Provost, S., & Haubold, H. (2022). The multivariate Gaussian and related distributions. In *Multivariate statistical analysis in the real and complex domains*. Springer, Cham. https://doi.org/10.1007/978-3-030-95864-0

[6] Venables, W. N., & Ripley, B. D. (2002). *Modern applied statistics with S* (4th ed.). Springer. https://doi.org/10.1007/978-0-387-21706-2

[7] Ziya. (2023). *Motor control performance dataset* [Data set]. Kaggle. https://www.kaggle.com/datasets/ziya07/motor-control-performance-dataset

Appendix: R code for model comparison and combinatorial search

The following code illustrates the complete evaluation engine developed in R to compare our trigonometric distribution against 9 classical probability models. It includes the calculation of Akaike weights and the combinatorial loop used to automatically test all possible variable pairs in the dataset.

```
#=====================================================================
# MULTI-MODEL ANALYSIS: TEN BIVARIATE DISTRIBUTIONS
#=====================================================================

# Required packages
library(MASS)
library(mvtnorm)
library(dplyr)

#---------------------------------------------------------------------
# Function: evaluate_bivariate_models
# Purpose : Fit a collection of bivariate models to (x, y),
#           compute AIC/BIC, and return a sorted table with Akaike weights.
#---------------------------------------------------------------------
evaluate_bivariate_models <- function(dataset_label, x, y) {

  #-------------------------------------------------------------------
  # Data preprocessing (numeric coercion + removal of missing values)
  #-------------------------------------------------------------------
  data_xy <- na.omit(data.frame(x = as.numeric(x), y = as.numeric(y)))
  x <- data_xy$x
  y <- data_xy$y
  n <- length(x)

  if (n < 10) stop("Insufficient number of valid observations (n < 10).")

  # Container for model outputs
  model_table_list <- list()

  #-------------------------------------------------------------------
  # Internal helper: append one model result to the container
  #-------------------------------------------------------------------
  append_model <- function(model_name, n_parameters,
  nll_value, theta_hat = NA_real_) {
    if (is.null(nll_value) || length(nll_value) == 0) nll_value <- NA_real_
    model_table_list[[length(model_table_list) + 1]] <<- data.frame(
      Model = model_name,
      Estimated_Theta = theta_hat,
      K = n_parameters,
```

```
    NLL = as.numeric(nll_value)
  )
}

#---------------------------------------------------------------
# 1) Bivariate Cosine Gaussian (BCG): one-parameter model (theta)
#---------------------------------------------------------------
nll_bcg <- function(theta) {
  normalizing_const <- pi * (1 + 1 / sqrt(1 + 4 * theta^2))
  -sum(
    -log(normalizing_const) -
      (x^2 + y^2) / 2 +
      log(cos(theta * x * y)^2 + 1e-300)  # numerical safeguard
  )
}

opt_bcg <- optimize(nll_bcg, interval = c(-10, 10))
append_model(
  model_name   = "1. Bivariate Cosine Gaussian",
  n_parameters = 1,
  nll_value    = opt_bcg$objective,
  theta_hat    = round(opt_bcg$minimum, 3)
)

#---------------------------------------------------------------
# 2) Unrestricted bivariate Gaussian: 5 parameters
#    (mu_x, mu_y, sigma_x, sigma_y, rho)
#---------------------------------------------------------------
mu_hat <- c(mean(x), mean(y))
Sigma_hat <- cov(data_xy)
nll_bvn <- -sum(dmvnorm(data_xy, mean = mu_hat, sigma = Sigma_hat, log = TRUE))
append_model("2. Bivariate Gaussian", 5, nll_bvn)

#---------------------------------------------------------------
# 3) Centered bivariate Gaussian: 3 parameters
#    (sigma_x, sigma_y, rho) with mean fixed at (0,0)
#---------------------------------------------------------------
nll_bvn_centered <- -sum(dmvnorm(data_xy, mean = c(0, 0),
sigma = Sigma_hat, log = TRUE))
append_model("3. Bivariate Gaussian (Centered)", 3, nll_bvn_centered)

#---------------------------------------------------------------
# 4) Standard independent Gaussian baseline: 0 parameter
#    (mean 0, variance 1, independence)
#---------------------------------------------------------------
```

```r
nll_std_ind_norm <- -sum(dnorm(x, 0, 1, log = TRUE) +
dnorm(y, 0, 1, log = TRUE))
append_model("4. Standard Independent Gaussian (Baseline)", 0, nll_std_ind_norm)

#-----------------------------------------------------------------
# 5) Independent Gaussians (univariate MLE per margin): 4 parameters
#-----------------------------------------------------------------
nll_ind_norm <- -sum(
  dnorm(x, mean(x), sd(x), log = TRUE) +
    dnorm(y, mean(y), sd(y), log = TRUE)
)
append_model("5. Independent Gaussians", 4, nll_ind_norm)

#-----------------------------------------------------------------
# 6) Bivariate Student t: 6 parameters
#    (mu_x, mu_y, sigma_x, sigma_y, rho, nu)
#    nu is estimated numerically (bounded optimization).
#-----------------------------------------------------------------
nll_bvt <- tryCatch({
  res_optim <- optim(
    par = 5,
    fn = function(nu) {
      if (nu <= 2) return(1e9)  # ensures finite variance
      -sum(dmvt(data_xy, delta = mu_hat, sigma = Sigma_hat,
      df = nu, log = TRUE))
    },
    method = "L-BFGS-B",
    lower = 2.1,
    upper = 100
  )
  res_optim$value
}, error = function(e) NA_real_)

append_model("6. Bivariate Student t", 6, nll_bvt)

#-----------------------------------------------------------------
# Auxiliary: independent marginal fitting via MASS::fitdistr
# Used for models 7--9.
#-----------------------------------------------------------------
nll_independent_marginals <- function(dist_name) {
  tryCatch({
    fit_x <- fitdistr(x, dist_name)
    fit_y <- fitdistr(y, dist_name)

    if (dist_name == "t") {
```

```
      # fitdistr for "t": (m, s, df). Use standardized
      dt and add Jacobian term -log(s).
      nll_x <- -sum(dt((x - fit_x$estimate[1]) / fit_x$estimate[2],
          df = fit_x$estimate[3], log = TRUE) - log(fit_x$estimate[2]))
      nll_y <- -sum(dt((y - fit_y$estimate[1]) / fit_y$estimate[2],
          df = fit_y$estimate[3], log = TRUE) - log(fit_y$estimate[2]))
      return(nll_x + nll_y)
    }

    if (dist_name == "logistic") {
      return(-sum(
        dlogis(x, fit_x$estimate[1], fit_x$estimate[2], log = TRUE) +
          dlogis(y, fit_y$estimate[1], fit_y$estimate[2], log = TRUE)
      ))
    }

    if (dist_name == "cauchy") {
      return(-sum(
        dcauchy(x, fit_x$estimate[1], fit_x$estimate[2], log = TRUE) +
          dcauchy(y, fit_y$estimate[1], fit_y$estimate[2], log = TRUE)
      ))
    }

    NA_real_
  }, error = function(e) NA_real_)
}

append_model("7. Independent Student t marginals", 6,
nll_independent_marginals("t"))
append_model("8. Independent Logistic marginals", 4,
nll_independent_marginals("logistic"))
append_model("9. Independent Cauchy marginals", 4,
nll_independent_marginals("cauchy"))

#-----------------------------------------------------------------
# 10) Independent Laplace marginals: 4 parameters
#     Location estimated by median; scale by mean absolute deviation.
#-----------------------------------------------------------------
loc_x <- median(x); scale_x <- mean(abs(x - loc_x))
loc_y <- median(y); scale_y <- mean(abs(y - loc_y))
nll_laplace <- -sum(-log(2 * scale_x) - abs(x - loc_x) / scale_x) -
  sum(-log(2 * scale_y) - abs(y - loc_y) / scale_y)

append_model("10. Independent Laplace marginals", 4, nll_laplace)
```

```
  #=================================================================
  # Information criteria and sorting
  #=================================================================
  df <- do.call(rbind, model_table_list)
  df <- df[!is.na(df$NLL), ]  # discard failed fits

  df$LogLik <- -df$NLL
  df$AIC <- 2 * df$K + 2 * df$NLL
  df$BIC <- df$K * log(n) + 2 * df$NLL

  df <- df[order(df$AIC), ]

  min_aic <- min(df$AIC)
  df$Delta_AIC <- df$AIC - min_aic
  w_raw <- exp(-0.5 * df$Delta_AIC)
  df$Akaike_Weight_Pct <- round((w_raw / sum(w_raw)) * 100, 2)

  # Presentation cleanup
  df$NLL <- NULL
  df$AIC <- round(df$AIC, 1)
  df$BIC <- round(df$BIC, 1)
  df$LogLik <- round(df$LogLik, 1)
  df$Delta_AIC <- round(df$Delta_AIC, 1)
  rownames(df) <- NULL

  return(df)
}

#=====================================================================
# EXHAUSTIVE PAIRWISE SCREENING OVER A MULTIVARIATE DATASET
#=====================================================================

# Assumption: 'motor' is already loaded in the workspace
X <- motor %>%
  select(-12, -20, -21, -22, -23) %>%
  mutate(across(everything(), as.numeric))

# Standardization (centering and scaling)
X <- scale(X, center = TRUE, scale = TRUE)
variable_names <- colnames(X)

# Enumerate all unordered pairs of variables
pair_index <- combn(1:ncol(X), 2)
n_pairs <- ncol(pair_index)
```

```
# Build a list of bivariate samples (2 x n matrices)
pair_samples <- vector("list", n_pairs)
for (i in seq_len(n_pairs)) {
  j1 <- pair_index[1, i]
  j2 <- pair_index[2, i]
  pair_samples[[i]] <- rbind(X[, j1], X[, j2])
  rownames(pair_samples[[i]]) <- variable_names[c(j1, j2)]
}

#-----------------------------------------------------------------------
# Summary table: pairs for which the BCG model yields the lowest AIC
#-----------------------------------------------------------------------
bcg_best_fit_summary <- data.frame(
  Variable_1 = character(),
  Variable_2 = character(),
  Estimated_Theta = numeric(),
  Akaike_Weight_Pct = numeric(),
  Delta_AIC_2nd = numeric(),
  stringsAsFactors = FALSE
)

bcg_best_fit_tables <- list()

cat("Running model comparison over", n_pairs, "pairs...\n")

for (i in seq_len(n_pairs)) {
  j1 <- pair_index[1, i]
  j2 <- pair_index[2, i]
  pair_label <- paste(variable_names[j1], "and", variable_names[j2])

  # Silence intermediate output to avoid console clutter
  capture.output({
    model_evaluation <- evaluate_bivariate_models(pair_label,
    pair_samples[[i]][1, ], pair_samples[[i]][2, ])
  })

  # Retain pairs for which BCG provides the best fit (lowest AIC)
  if (model_evaluation$Model[1] == "1. Bivariate Cosine Gaussian") {
    bcg_best_fit_summary <- rbind(bcg_best_fit_summary, data.frame(
      Variable_1 = variable_names[j1],
      Variable_2 = variable_names[j2],
      Estimated_Theta = model_evaluation$Estimated_Theta[1],
      Akaike_Weight_Pct = model_evaluation$Akaike_Weight_Pct[1],
      Delta_AIC_2nd = model_evaluation$Delta_AIC[2],
      stringsAsFactors = FALSE
```

```
    ))
    bcg_best_fit_tables[[pair_label]] <- model_evaluation
  }
}

cat("Completed. The BCG model provides the best fit for",
nrow(bcg_best_fit_summary), "pairs.\n\n")

# Display the summary (sorted by decreasing Akaike weight)
if (nrow(bcg_best_fit_summary) > 0) {
  bcg_best_fit_summary <- bcg_best_fit_summary[order(
  bcg_best_fit_summary$Akaike_Weight_Pct), ]
  print(bcg_best_fit_summary)
} else {
  cat("No pair yields the BCG model as the best fitting model.\n")
}
```

Listing 6: Complete model selection and combinatorial analysis engine

Appendix: Core R functions for the BCG distribution

The following script contains the core functions developed for this project. It includes the accept-reject sampler, the negative log-likelihood function, the custom PSO algorithm, and the parallelized goodness-of-fit test using the energy distance and parametric bootstrap.

```r
# ======================================================================
# CORE FUNCTIONS: SIMULATION, ESTIMATION, AND GOODNESS-OF-FIT
# ======================================================================

# Load required packages
library(MASS)
library(mvtnorm)
library(energy)
library(future)
library(future.apply)

# Setup parallel processing to speed up the bootstrap
plan(multisession, workers = parallel::detectCores() - 1)

# ======================================================================
# 1. DATA GENERATOR (ACCEPT-REJECT SAMPLER)
# ======================================================================
rbcg <- function(n, theta = 1) {
  mu <- c(0, 0)
  Sigma <- diag(2)
  out <- matrix(NA_real_, nrow = 2, ncol = n)
  rownames(out) <- c("X", "Y")

  i <- 1
  while (i <= n) {
    v <- as.numeric(MASS::mvrnorm(1, mu = mu, Sigma = Sigma))
    if (runif(1) <= cos(theta * v[1] * v[2])^2) {
      out[, i] <- v
      i <- i + 1
    }
  }
  out
}

# ======================================================================
# 2. DENSITY AND LIKELIHOOD FUNCTIONS
# ======================================================================

# Bivariate Cosine Gaussian (BCG) probability density function
```

```
bcg <- function(par, x, y) {
  theta <- par[1]
  C <- pi * (1 + 1 / sqrt(1 + 4 * theta^2))
  val <- (1 / C) * exp(- (x^2 + y^2) / 2) * cos(theta * x * y)^2
  return(val)
}

# NLL with numerical safeguard
nll <- function(par, data, dens_fun = bcg, eps = 1e-300) {
  stopifnot(is.matrix(data), nrow(data) == 2, ncol(data) > 0)
  x <- data[1, ] ; y <- data[2, ]

  vals <- dens_fun(par, x, y)
  if (any(!is.finite(vals))) stop("Non-finite density encountered for
  some points.")

  vals <- pmax(vals, eps)
  return(-sum(log(vals)))
}

# ======================================================================
# 3. PSO ALGORITHM
# ======================================================================
pso <- function(func, S = 350, lim_inf, lim_sup, e = 1e-4,
                N = 500, prop = 0.2,
                omega = 0.6, phi_p = 1.4, phi_g = 1.4) {

  stopifnot(length(lim_inf) == length(lim_sup))
  d <- length(lim_sup)

  X <- matrix(NA_real_, nrow = S, ncol = d)
  for (j in 1:d) X[, j] <- runif(S, lim_inf[j], lim_sup[j])
  V <- matrix(0, nrow = S, ncol = d)

  fX <- apply(X, 1, function(row) func(as.numeric(row)))
  if (any(!is.finite(fX))) stop("Objective function returned non-finite
  values at initialization.")

  P  <- X ; fP <- fX
  g_idx <- which.min(fP)
  g <- P[g_idx, , drop = FALSE]
  f_g <- fP[g_idx]

  hist_fg <- numeric(0)
  it <- 0
```

```
repeat {
  it <- it + 1
  r_p <- matrix(runif(S * d), nrow = S)
  r_g <- matrix(runif(S * d), nrow = S)

  # Velocity and position update
  V <- omega * V + phi_p * r_p * (P - X) +
    phi_g * r_g * (matrix(rep(g, each = S), nrow = S) - X)
  X <- X + V

  # Apply boundaries
  for (j in 1:d) {
    X[, j] <- pmin(pmax(X[, j], lim_inf[j]), lim_sup[j])
  }

  fX <- apply(X, 1, function(row) func(as.numeric(row)))
  fX[!is.finite(fX)] <- .Machine$double.xmax

  # Update personal bests
  improved <- fX <= fP
  if (any(improved)) {
    P[improved, ] <- X[improved, ]
    fP[improved]  <- fX[improved]
  }

  # Update global best
  m_idx <- which.min(fP)
  if (fP[m_idx] < f_g) {
    g   <- P[m_idx, , drop = FALSE]
    f_g <- fP[m_idx]
  }

  hist_fg <- c(hist_fg, f_g)

  # Convergence check
  if (length(hist_fg) >= N) {
    n_var <- ceiling(length(hist_fg) * prop)
    win   <- tail(hist_fg, n_var)
    if (var(win) <= e) break
  }

  if (it >= 5000) break
}
```

```
  return(list(par = as.numeric(g), f = hist_fg))
}

# ======================================================================
# 4. ENERGY STATISTIC AND PARAMETRIC BOOTSTRAP (2-SAMPLE TEST)
# ======================================================================

# Wrapper to estimate theta using PSO
theta_hat_pso <- function(D, lim_inf = 0, lim_sup = 100, S = 120,
N = 400, prop = 0.25) {
  objective <- function(par) nll(par, data = D, dens_fun = bcg)
  fit <- pso(objective, lim_inf = c(lim_inf), lim_sup = c(lim_sup),
  S = S, N = N, prop = prop)
  return(as.numeric(fit$par[1]))
}

# Compute the energy statistic between two samples
energy_stat_2sample <- function(D1, D2) {
  X <- t(D1)
  Y <- t(D2)
  Z <- rbind(X, Y)
  sizes <- c(nrow(X), nrow(Y))
  T_stat <- energy::eqdist.e(Z, sizes = sizes)
  if (is.list(T_stat) && !is.null(T_stat$statistic))
  return(as.numeric(T_stat$statistic))
  return(as.numeric(T_stat))
}

# Goodness-of-Fit test using parametric bootstrap (parallelized)
gof_energy_boot <- function(D, lim_inf = 0, lim_sup = 100,
                            S = 120, N = 400, prop = 0.25,
                            B = 199, seed = 123,
                            workers = max(1, parallel::detectCores() - 1)) {

  # Check dependencies
  if (!requireNamespace("future", quietly = TRUE) ||
      !requireNamespace("future.apply", quietly = TRUE)) {
    stop("Please install the required packages first:
    install.packages(c('future','future.apply'))")
  }

  set.seed(seed)
  n <- ncol(D)

  # 1) Fit on original data
```

```
  theta_hat <- theta_hat_pso(D, lim_inf, lim_sup, S, N, prop)

  # 2) Calculate observed statistic
  Y0 <- rbcg(n, theta_hat)
  T_obs <- energy_stat_2sample(D, Y0)

  # 3) Parametric bootstrap with re-fit (Parallel processing)
  old_plan <- future::plan()
  on.exit(future::plan(old_plan), add = TRUE)
  future::plan(future::multisession, workers = workers)

  # future.seed ensures reproducibility in parallel loops
  res <- future.apply::future_lapply(seq_len(B), function(b) {
    Db <- rbcg(n, theta_hat)
    theta_b <- theta_hat_pso(Db, lim_inf, lim_sup, S, N, prop)
    Yb <- rbcg(n, theta_b)

    list(theta_b = theta_b,
         T_stat = energy_stat_2sample(Db, Yb))
  }, future.seed = TRUE)

  theta_boot <- vapply(res, `[[`, numeric(1), "theta_b")
  T_boot     <- vapply(res, `[[`, numeric(1), "T_stat")

  # Final p-value calculation
  pval <- (1 + sum(T_boot >= T_obs)) / (B + 1)

  return(list(theta_hat = theta_hat,
              T_obs = T_obs,
              p_value = pval,
              theta_boot = theta_boot,
              T_boot = T_boot,
              workers = workers))
}
```

Listing 7: Core R functions