# SimJoint: simulate joint distribution given marginals and correlation matrix

Charlie Wusuo Liu

June 9, 2019

**Abstract**

This R package simulates joint distribution given nonparametric marginals and their covariance structure characterized by a Pearson or Spearman correlation matrix. The simulator engages the problem from a purely computational perspective. It assumes no statistical models such as copulas or parametric distributions, and can approximate the target correlations regardless of theoretical feasibility. The algorithm integrates and advances the Iman-Conover (1982) approach [1] and the Ruscio-Kaczetow (2008) iteration [2]. Additionally, a simple heuristic algorithm is designed to optimize the joint distribution in the post-simulation stage. This heuristic demonstrated not only strong capability of cost reduction, but also good potential of achieving the same level of precision of approximation without the enhanced Iman-Conover-Ruscio-Kaczetow.

## Contents



## 1 Iman-Conover review

In this document, a data matrix is seen as a joint distribution whose marginals are the columns. Correlation matrix of the data matrix refers to correlations between columns. If unspecified, correlation means Pearson.

Let $\Sigma$ be a $K \times K$ positive semi-definite correlation matrix. Let $S$ be an $N \times K$ noise matrix whose columns are uncorrelated and have equal variances — the covariance matrix of $S$ is diagonal of equal entries. The following holds: for any decomposition in the form of $\Sigma = U^\top U$, the correlation matrix of $Y = SU$ is $\Sigma$.

Iman-Conover transformation builds on the above fact. Let $X$ be an arbitrary $N \times K$ matrix. If every column of $X$ is reordered such that $X$ and $Y$ have the same Spearman rank correlation matrix, then they *should* have close Pearson correlation matrices, namely the reordered $X$ *should* have a correlation matrix close to $\Sigma$. The algorithm goes:

1. Cholesky-decompose $\Sigma = U^\top U$ where $U$ is the upper-triangle. Any decomposition in the form of $\Sigma = U^\top U$ is admissible, although Cholesky decomposition is arguably the cheapest for positive definite matrices.
2. Let $Y = SU$. $Y$ will be an $N \times K$ matrix whose correlation matrix equals $\Sigma$.
3. Reorder elements in the $k$th column, $k = 0, 1, \ldots, K-1$, of $X$ such that it perfectly rank-correlates the $k$th column of $Y$. Denote the reordered $X$ by $X^{(1)}$. Its correlation matrix will be *close* to $\Sigma$.

### 1.1 Enhancement

An eigen-decomposition can substitute the Cholesky decomposition if $\Sigma$ is not postive-definite: let $\Sigma = (\Lambda^{\frac{1}{2}}Q)^\top(\Lambda^{\frac{1}{2}}Q)$ and set $Y = S(\Lambda^{\frac{1}{2}}Q)$.

## 2 Ruscio-Kaczetow iteration

Two sources of errors can prevent $X^{(1)}$ from having a Pearson correlation matrix exactly equal to $\Sigma$: (i) columns of the noise matrix $S$ can hardly be perfectly uncorrelated (limited by random number generator), (ii) $X^{(1)}$ and $Y$ having identical rank correlation matrices does not imply they have exactly the same Pearson correlation matrix.

Let $\Sigma^{(1)}$ be the Pearson correlation matrix of $X^{(1)}$. Let $e(\cdot)$ be an error function. Feeding a different target correlation matrix $\Sigma_{\text{target}}$ in Step 1 Section 1 may end up with a lower error $e(\Sigma - \Sigma^{(1)})$. Ruscio and Kaczetow (2008) proposed an iterative algorithm for adjusting $\Sigma_{\text{target}}$ according to $\Delta = \Sigma - \Sigma^{(1)}$ until $e(\Delta)$ converges. The algorithm can be categorized as a variant of Nelder–Mead [3].

### 2.1 Enhancement

For some reason, Ruscio and Kaczetow embedded a quite expensive factor analysis to obtain the matrix that is equivalent to $Y$ in Iman-Conover. My guesses are (i) they might be concerned by the fact that Cholesky decomposition only applies to positive definite matrix, or (ii) they simply intended to threshold the dimensionality with principal components — the number of factors for reproducing $\Sigma$ is less than $K$. Such a dimension reduction in the middle however has no impact on the final result, because we do need $K$ correlated vectors spanned from those factors to rank-order the marginals.

Another drawback of Ruscio and Kaczetow's iteration is that the correlation adjustments, analogous to step sizes in gradient descent, are deterministic and identical for every entry of the correlation matrix. Simulations have shown random step sizes can largely lower the converged $e(\Delta)$. This package uses a uniform kernel to generate the stochastic steps. Other choices

**Algorithm 1** Simulate joint distribution given marginals and correlation matrix

**Input:** (i) an $N \times K$ matrix $X$ of samples from $K$ marginal distributions; (ii) a $K \times K$ correlation matrix $\Sigma$; (iii) maximal iteration $i_{\max}$; (iv) range of the step size for correcting correlations $[l, u]$ — self-explanatory in the algorithm, default [0, 1]; (v) convergence tail size $h$ — self-explanatory in the algorithm.

**Optional input:** an $N \times K$ noise matrix $S$ where columns are uncorrelated (approximately). One can populate this matrix with uniform random numbers.

**Operator and function definitions:**

$\odot$: element-wise multiplication for vectors or matrices.

$\sigma(X)$: export the column-wise Pearson correlation matrix of $X$ .

$\phi(\mathbf{x})$: given a vector $\mathbf{x}$ of size $\frac{1}{2}(K-1)K$, export a $K \times K$ symmetric matrix where the diagonal equals 1 and the lower triangle entries equal $\mathbf{x}$ .

$e(\Delta)$: given a matrix $\Delta$, export a scalar, e.g. the sum of all squared elements in $\Delta$.

$\pi(X, Y)$: given $N \times K$ matrices $X$ and $Y$, reorder elements in the $k$th column of $X$ such that it perfectly rank-correlates the $k$th column of $Y$, $k = 0, 1, \ldots, K-1$ . Export the reordered $X$.

If $S$ is not given, copy $X$ to $S$ and permute each column of $S$ at random.
$\epsilon^{\text{optimal}} \leftarrow \infty$; $i \leftarrow 0$; $\Sigma^{(i)}_{\text{target}} \leftarrow \Sigma$;
Fill an array $\{\epsilon_0, \ldots, \epsilon_{h-1}\}$ with arbitrary unequal values.
Allocate vector $\mathbf{r}$ of size $\frac{1}{2}(K-1)K$ for storing stochastic step ratio.
**while** $i < i_{\max}$ **do**
  Cholesky (primary) or eigen (secondary) decomposition: $\Sigma^{(i)}_{\text{target}} = A^\top A$ .
  $Y \leftarrow SA$ .
  $X^{(i)} \leftarrow \pi(X, Y)$ .
  $\Sigma^{(i)} \leftarrow \sigma(X^{(i)})$ .
  $\Delta \leftarrow \Sigma - \Sigma^{(i)}$ .
  $\epsilon \leftarrow e(\Delta)$ .
  $\{\epsilon_0, \ldots, \epsilon_{h-1}\} \leftarrow \{\epsilon_1, \ldots, \epsilon_{h-2}, \epsilon\}$ .
  **if** $\epsilon_0 = \epsilon_1 = \ldots = \epsilon_{h-1}$ **then**
    **break** ▷ Algorithm converged.
  **end if**
  **loop**
    **if** $\epsilon < \epsilon^{\text{optimal}}$ **then**
      $\Sigma^{\text{optimal}} \leftarrow \Sigma^{(i)}$ .
      $\epsilon^{\text{optimal}} \leftarrow \epsilon$ .
      $\Sigma^{\text{base}}_{\text{target}} \leftarrow \Sigma^{(i)}_{\text{target}}$ .
      $\mathbf{r} \leftarrow \mathbf{1}$ .
    **else**
      Populate a vector $\mathbf{x}$ of size $\frac{1}{2}(K-1)K$ with random uniforms in $[l, u]$ .
      $\mathbf{r} \leftarrow \mathbf{r} \odot \mathbf{x}$
    **end if**
    $i \leftarrow i + 1$ .
    $\Sigma^{(i)}_{\text{target}} \leftarrow \min\left(\mathbf{1}, \max\left(-\mathbf{1}, \Sigma^{\text{base}}_{\text{target}} + \phi(\mathbf{r}) \odot (\Sigma - \Sigma^{\text{optimal}})\right)\right)$ .
    **if** $\Sigma^{(i)}_{\text{target}}$ is positive semi-definite **then** ▷ merged with Step 6.
      **break** .
    **end if**
  **end loop**
**end while**
**return** $X^{(i)}$ .

of kernels might lead to faster convergence in terms of iterations, but they are usually more expensive which tends to prolong the overall computing time.

Ruscio and Kaczetow's codes published with their paper were erroneous. Instructions for correction are made in the example section for SJpearson() in the package manual.

The full improved algorithm is depicted in Algorithm 1.

# 3 Copula

If marginals of the noise matrix $S$ are sampled from a Gaussian distribution, then the result from Algorithm 1 is a joint distribution whose dependency structure can be characterized by a Gaussian copula. In general, if marginals of $S$ are sampled from a *stable distribution*, e.g. Gaussian, Cauchy, Levy, etc., then the dependency structure of the joint can be characterized by a copula of the same name. In practice, it is recommended to take quantiles of the distribution of interest rather than random sampling for an $S$. For example, to sample a $1000 \times 5$ $S$ from Gaussian, one can shuffle at random $\left(\Phi^{-1}(0.0005), \Phi^{-1}(0.0015), \ldots, \Phi^{-1}(0.9995)\right)$ five times and bind them as columns.

Note that the joint distribution's dependency structure can no longer be characterized by the corresponding parametric copula after performing the following optimization procedure.

# 4 Post-simulation optimization

We propose a simple stochastic optimization procedure to further reduce the final cost $\epsilon$ in Algorithm 1:

1. From the output correlation matrix $\Sigma^{(i)}$, identify the $\kappa$ entries that have the $\kappa$ largest absolute difference against the target correlation matrix $\Sigma$ .

2. Select one of the $\kappa$ entries with a probability. Identify which two columns in $X^{(i)}$ result in this correlation.

   - We set the probability as an increasing function of the absolute difference.

3. According to certain criteria (below), from the two columns in Step 2, sample two rows so we have four elements.

   (a) Let $X^{(i)}[s, u]$, $X^{(i)}[t, u]$, $X^{(i)}[s, v]$, $X^{(i)}[t, v]$ represent the four elements. Let $\rho^*$ be the correlation between columns $X^{(i)}[\,, u]$ and $X^{(i)}[\,, v]$ .

   (b) We keep sampling $s$, $t$ until

   $$(X^{(i)}[s, u] - X^{(i)}[t, u]) \cdot (X^{(i)}[s, v] - X^{(i)}[t, v]) < 0 \text{ if } \rho^* \text{ is below target,} \quad (1)$$

   or,

   $$(X^{(i)}[s, u] - X^{(i)}[t, u]) \cdot (X^{(i)}[s, v] - X^{(i)}[t, v]) > 0 \text{ if } \rho^* \text{ is above target.} \quad (2)$$

4. Let $w$ be $u$ or $v$ at random. Swap $X^{(i)}[s, w]$ and $X^{(i)}[t, w]$ if this would reduce the final cost $\epsilon$.

5. Repeat Steps 1 to 4 until the last consecutive $h$ iterations failed to lower the cost.

In various experiments, feeding the input of Algorithm 1 to this procedure achieves the same level of precision of approximation.

# 5 Implementation

The package is implemented in C++ and is carefully programmed for computing speed. Memory consumption is dominated by three $N \times K$ matrices: the input $X$, the input $S$ or a permuted $X$, and $Y$. The input $X$ is normalized (column shifted and scaled) such that its Gramian matrix equals the correlation matrix. The permuted $X$ or $S$ is in row-major for promoting cache locality while left-multiplying the factorization $A$. Matrix multiplications are hand-optimized to exploit matrix symmetries and triangularities. All major steps except for Cholesky and eigen decompositions, which employed C++ library Armadillo, are crafted for multithreading.

Imposing Spearman correlations is equivalent to imposing the same Pearson correlations on ranks. If a rank correlation matrix is given, the algorithm populates three $N \times K$ single-precision matrices: $X$, a permuted $X$, and $Y$ with normalized ranks. Therefore the memory usage stays below three $N \times K$ double-precision matrices. Single-precision floats are sufficiently accurate for operations on ranks due to their uniformity.

For the post-simulation optimization, the key to high computing speed is to not recompute, but rather to regionally update, the correlation matrix and the cost function in each iteration. Programming details are presented in C++ source comments.

The package imports Rcpp, RcppArmadillo, RcppParallel for bridging C++ and R.

# References


[1] R. L. Iman and W. J. Conover, "A distribution-free approach to inducing rank correlation among input variables," *Communications in Statistics - Simulation and Computation*, 1982. [Online]. Available: https://doi.org/10.1080/03610918208812265

[2] J. Ruscio and W. Kaczetow, "Simulating multivariate nonnormal data using an iterative algorithm," *Multivariate Behavioral Research*, 2008. [Online]. Available: https://doi.org/10.1080/00273170802285693

[3] J. A. Nelder and R. Mead, "A simplex method for function minimization," *Computer Journal*, 1965. [Online]. Available: https://doi:10.1093/comjnl/7.4.308