Rearrangement Algorithm

Release 0.1.1

Karl-Ludwig Besser

Oct 25, 2021

CONTENTS:

1 Installation **1**

2 Examples **3**
- 2.1 Basics . . . 3
- 2.2 Comonotonic Rearrangement . . . 4
- 2.3 Bounds on the Value-at-Risk (VaR) . . . 4
- 2.4 Bounds on the Expected Value . . . 5
- 2.5 Bounds on the Survival Probability . . . 5

3 rearrangement_algorithm Package **7**
- 3.1 Functions . . . 7

4 Indices and tables **13**

Python Module Index **15**

Index **17**

CHAPTER

ONE

INSTALLATION

You can install the package via pip

```
pip install rearrangement-algorithm
```

If you want to install the latest (unstable) version, you can install the package from source

```
git clone https://gitlab.com/klb2/rearrangement-algorithm.git
cd rearrangement-algorithm
git checkout dev
pip install .
```

You can test, if the installation was successful, by importing the package

```
import rearrangement_algorithm as ra
print(ra.__version__)
```

CHAPTER

TWO

EXAMPLES

In the following, some simple usage examples are given.

2.1 Basics

We will start with some simple basics that are necessary for all following examples.

Besides the actual `rearrangement_algorithm` package, we will use the scipy.stats module to specify the marginal distributions.

The `.ppf` method of a distribution implements the quantile function, which we will heavily use in the following examples.

2.1.1 Basic Rearrange

```
import numpy as np
import rearrangement_algorithm as ra

# create a matrix (10x3) where each columns contains numbers 0 to .9
X = np.tile(np.linspace(0, 1, 10), (3, 1)).T
print(X)

# the row sums are varying a lot
print(np.sum(X, axis=1))

# rearrange it
X_ra = ra.basic_rearrange(X, min)
print(X_ra)

# the row sums are now very balanced
print(np.sum(X_ra, axis=1))
```

2.2 Comonotonic Rearrangement

The first example is to generate a comonotonic rearrangement of random variables with given marginals. For the following example, we will use $X_1 \sim \exp(1)$, $X_2 \sim \mathcal{N}(0,1)$, and $X_3 \sim \mathcal{U}[0,1]$.

The important function to create the comonotonic rearrangement is *rearrangement_algorithm.create_comonotonic_ra()*.

```
from scipy import stats
import rearrangement_algorithm as ra

# create the random variables
X1 = stats.expon()  # exponential distribution
X2 = stats.norm()  # normal distribution
X3 = stats.uniform()  # uniform distribution

# quantile functions are given by the .ppf method in scipy
qf = [X1.ppf, X2.ppf, X3.ppf]

# create rearrangement
level = 0.5
X_ra_lower, X_ra_upper = ra.create_comonotonic_ra(level, qf, 10)

# all columns are ordered in increasing order
print(X_ra_lower)
print(X_ra_upper)
```

2.3 Bounds on the Value-at-Risk (VaR)

One relevant use case of the rearrangement algorithm is the numerical approximation of bounds on the value-at-risk (VaR) of dependent risks.

The following example illustrates the use of the *rearrangement_algorithm.bounds_VaR()* function using the following example from the paper "Computation of sharp bounds on the distribution of a function of dependent risks" (Puccetti, Rüschendorf, 2012).

Given three Pareto(2)-distributed random variables, we calcuate the lower bound on the VaR (based on the sum of the three random variables). For comparison, the numbers can be found in the above paper (Table 1).

```
from scipy import stats
import rearrangement_algorithm as ra

# create the quantile functions
qf = [stats.pareto(2, loc=-1).ppf]*3

# set the parameters
alpha = 0.5102
num_steps = 500

# calcuate the bounds on the VaR
lower_under, lower_over = ra.bounds_VaR(1.-alpha, qf, method="lower",
                                        num_steps=num_steps)
```

(continues on next page)

(continued from previous page)

```
VaR_under = lower_under[0]
VaR_over = lower_over[0]

# the expected value is around 0.5
expected = 0.5
print("{:.5f} <= {:.3f} <= {:.5f}".format(VaR_under, expected, VaR_over))
```

2.4 Bounds on the Expected Value

TODO

2.5 Bounds on the Survival Probability

TODO

CHAPTER

THREE

REARRANGEMENT_ALGORITHM PACKAGE

3.1 Functions

basic_rearrange(x_mat, optim_func[, ...])	Implementation of the matrix rearrangement algorithm.
bounds_expectation_supermod(quant[, ...])	Computing the lower/upper bounds on the expectation of supermodular functions.
bounds_surv_probability(quant, s_level[, ...])	Computing the lower/upper bounds on the survival probability of a function of dependent risks
bounds_VaR(level, quant[, num_steps, ...])	Computing the lower/upper bounds for the best and worst VaR
create_comonotonic_ra(level, quant[, num_steps])	Creating a matrix with comonotonic random variables
create_matrix_from_quantile(quant, prob[, level])	Create a matrix approximation from marginal quantile functions

3.1.1 basic_rearrange

`rearrangement_algorithm.`**`basic_rearrange`**(*x_mat*, *optim_func*, *lookback=0*, *tol=0.0*, *tol_type='absolute'*, *max_ra=0*, *cost_func=<function sum>*, *is_sorted=False*, **args*, ***kwargs*)

Implementation of the matrix rearrangement algorithm.

Basic implementation of the rearrangement algorithm to get a permutation of a matrix X such that each column j is oppositely ordered to the vector derived by applying a function ψ to the (sub-)matrix without column j.

The implementation is based on the R library `qrmtools`[1]. A detailed description can be found in[2].

Parameters

- **x_mat** (*2D-array or matrix*) – Matrix of shape *(num_samples, num_var)* that will get rearranged.
- **optim_func** (*func*) – Internal optimization function. It needs to take a vector-argument and return a scalar. This usually is either `min()` or `max()`.
- **lookback** (*int, optional*) – Number of rearrangement steps to look back to determine convergence.
- **tol** (*float, optional*) – Tolerance to determine convergence.

[1] M. Hofert, K. Hornik, and A. J. McNeil, "qrmtools: Tools for Quantitative Risk Management." Version 0.0-13 (https://cran.r-project.org/web/packages/qrmtools/index.html)

[2] M. Hofert, "Implementing the Rearrangement Algorithm: An Example from Computational Risk Management," Risks, vol. 8, no. 2, May 2020.

- **tol_type** (*str, optional*) – Tolerance function used. Possible options are *"absolute"* and *"relative"*.
- **max_ra** (*int, optional*) – Number of maximum column rearrangements. If `0`, there will be no limit.
- **cost_func** (*func, optional*) – Cost function ψ that is applied to each row. It takes a 2D numpy.array as input and outputs a vector that results by applying the function to each row. It needs to take the keyword argument `axis` in the numpy style. An example for the sum is `numpy.sum()`.
- **is_sorted** (*bool, optional*) – Indicates wheter the columns of the matrix `x_mat` are sorted in increasing order.

Returns **x_rearranged** – Rearranged matrix of shape *(num_samples, num_var)*.

Return type numpy.array

References

3.1.2 bounds_expectation_supermod

`rearrangement_algorithm.`**`bounds_expectation_supermod`**(*quant, num_steps: int = 10, abstol: float = 0, lookback: int = 0, max_ra: int = 0, supermod_func=<function sum>, method: str = 'lower'*)

Computing the lower/upper bounds on the expectation of supermodular functions.

This function performs the RA and calculates the lower and upper bounds on the expected value of a supermodular function of dependent random variables. For mathematical details, see[1].

Parameters

- **quant** (*list*) – List of marginal quantile functions
- **num_steps** (*int*) – Number of discretization points
- **abstol** (*float*) – Absolute convergence tolerance
- **lookback** (*int*) – Number of column rearrangements to look back for deciding about convergence. Must be a number in $\{1, ...,$
- *lower*: for the lower bound
- *upper*: for the upper bound

Returns

- **bound_low** (*float*) – Lower bound on the approximated bound of the expected value
- **x_ra_low** (*numpy.array*) – Rearranged matrix for the lower bound
- **bound_up** (*float*) – Upper bound on the approximated bound of the expected value
- **x_ra_up** (*numpy.array*) – Rearranged matrix for the upper bound

[1] G. Puccetti and L. Rüschendorf, "Computation of Sharp Bounds on the Expected Value of a Supermodular Function of Risks with Given Marginals," Communications in Statistics - Simulation and Computation, vol. 44, no. 3, pp. 705-718, Mar. 2015.

References

3.1.3 bounds_surv_probability

`rearrangement_algorithm.bounds_surv_probability(quant, s_level: float, num_steps: int = 10, abstol: float = 0, lookback: int = 0, max_ra: int = 0, cost_func=<function sum>, method: str = 'lower', sample: bool = True)`

Computing the lower/upper bounds on the survival probability of a function of dependent risks

This function performs the RA and calculates the lower and upper bounds on the survival probability of function of dependent random variables. For mathematical details, see[1].

Parameters

- **quant** (*list*) – List of marginal quantile functions
- **s_level** (*float*) – Value of the function of the random variables at which the survival probability bounds are calculated.
- **num_steps** (*int*) – Number of discretization points
- **abstol** (*float*) – Absolute convergence tolerance
- **lookback** (*int*) – Number of column rearrangements to look back for deciding about convergence. Must be a number in $\{1, \ldots,$
- *lower*: for the lower bound
- *upper*: for the upper bound

sample (*bool*) – Indication whether each column of the two working matrices is randomly permuted before the rearrangements begin

Returns

- **bound_low** (*float*) – Lower bound on the survival probability
- **x_ra_low** (*numpy.array*) – Rearranged matrix for the lower bound
- **bound_up** (*float*) – Upper bound on the survival probability
- **x_ra_up** (*numpy.array*) – Rearranged matrix for the upper bound

References

3.1.4 bounds_VaR

`rearrangement_algorithm.bounds_VaR(level: float, quant, num_steps: int = 10, abstol: float = 0, lookback: int = 0, max_ra: int = 0, method: str = 'lower', sample: bool = True, cost_func=<function sum>)`

Computing the lower/upper bounds for the best and worst VaR

This function performs the RA and calculates the lower and upper bounds on the worst or best case VaR for a given confidence level. For mathematical details, see[1].

Parameters

[1] G. Puccetti and L. Rüschendorf, "Computation of sharp bounds on the distribution of a function of dependent risks," Journal of Computational and Applied Mathematics, vol. 236, no. 7, pp. 1833-1840, Jan. 2012.

[1] P. Embrechts, G. Puccetti, and L. Rüschendorf, "Model uncertainty and VaR aggregation," Journal of Banking & Finance, vol. 37, no. 8, pp. 2750-2764, Aug. 2013.

- **level** (*float*) – Confidence level between 0 and 1.
- **quant** (*list*) – List of marginal quantile functions
- **num_steps** (*int*) – Number of discretization points
- **abstol** (*float*) – Absolute convergence tolerance
- **lookback** (*int*) – Number of column rearrangements to look back for deciding about convergence. Must be a number in $\{1, ...,$
- *lower* or *best.VaR*: for best VaR
- *upper* or *worst.VaR*: for worst VaR

sample (*bool*) – Indication whether each column of the two working matrices is randomly permuted before the rearrangements begin

Returns

- **bound_low** (*float*) – Lower bound on the VaR
- **x_ra_low** (*numpy.array*) – Rearranged matrix for the lower bound on the VaR
- **bound_up** (*float*) – Upper bound on the VaR
- **x_ra_up** (*numpy.array*) – Rearranged matrix for the upper bound on the VaR

References

3.1.5 create_comonotonic_ra

`rearrangement_algorithm.create_comonotonic_ra(level: float, quant, num_steps: int = 10)`

Creating a matrix with comonotonic random variables

This function creates a rearrangement matrix that represents a joint distribution of comonotonic random variables for a given confidence level. Both upper and lower bound approximations are returned. For mathematical details, see[1]

Parameters

- **level** (*float*) – Confidence level between 0 and 1.
- **quant** (*list*) – List of marginal quantile functions
- **num_steps** (*int*) – Number of discretization points

Returns

- **x_ra_low** (*numpy.array*) – Lower bound approximation of the rearrangement matrix.
- **x_ra_up** (*numpy.array*) – Upper bound approximation of the rearrangement matrix.

[1] P. Embrechts, G. Puccetti, and L. Rüschendorf, "Model uncertainty and VaR aggregation," Journal of Banking & Finance, vol. 37, no. 8, pp. 2750-2764, Aug. 2013.

References

3.1.6 create_matrix_from_quantile

`rearrangement_algorithm.`**`create_matrix_from_quantile`**(*quant*, *prob*, *level=1.0*)

Create a matrix approximation from marginal quantile functions

Given a set of n quantile functions, this function returns a matrix where each column represents a random variable with the specified marginal distribution. Each row represents the value of the random variable at a given probability. For details, see[1].

Parameters

- **quant** (*list*) – List of marginal quantile functions
- **prob** (*list of float*) – Probability values at which the quantile functions are evaluated.
- **level** (*float*) – Confidence level between 0 and 1.

Returns **x_mat** – Matrix with the values of the random variables at the specified probabilites.

Return type numpy.array

References

[1] M. Hofert, "Implementing the Rearrangement Algorithm: An Example from Computational Risk Management," Risks, vol. 8, no. 2, May 2020.

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

r

rearrangement_algorithm, 7

INDEX

B

basic_rearrange() (*in module rearrangement_algorithm*), 7
bounds_expectation_supermod() (*in module rearrangement_algorithm*), 8
bounds_surv_probability() (*in module rearrangement_algorithm*), 9
bounds_VaR() (*in module rearrangement_algorithm*), 9

C

create_comonotonic_ra() (*in module rearrangement_algorithm*), 10
create_matrix_from_quantile() (*in module rearrangement_algorithm*), 11

M

module
    rearrangement_algorithm, 7

R

rearrangement_algorithm
    module, 7