20th International Congress on Modelling and Simulation, Adelaide, Australia, 1–6 December 2013
www.mssanz.org.au/modsim2013

RoseDist: Generalized Tool for Simulating with Non-Standard Probability Distributions

Jonathan Feinberg[a], Stuart Clark[a]

[a]*Simula Research Laboratory, PO 134, 1325 Lysaker, Norway*
Email: jonathfe@simula.no

Abstract: Monte Carlo simulation is the most popular technique for performing uncertainty quantification for being easy to implement and requiring very few assumption on the behavior of the model. However, in cases where model evaluations are computational costly, the technique can be become too expensive since Monte Carlo requires a high number of evaluations to get reasonable accuracy. To mitigate this cost, various variance reduction techniques have been introduced to increase the convergence rate. Unfortunately these techniques are only just making in-roads into computational modelling because of their inherent complexity and interdependence. `RoseDist` is a software toolbox in `Python` designed to make most variance reduction technique accessible, in an object-oriented sense, to numerical modellers from various disciplines.

Keywords: *Monte Carlo simulation, Quasi-Monte Carlo, Rosenblatt transformations, Copulas, Variance re-duction, Custom constructor*

1 INTRODUCTION

By far the most popular technique for performing uncertainty quantification on numerical modelling is the Monte Carlo method. This method is relatively simple to setup and versatile enough to be applied to many different models. However in cases where model evaluations have a high computational cost, Monte Carlo becomes prohibitive because the method requires a high number of evaluations to obtain reasonable accuracy. Numerous alternative uncertainty quantification methods have been developed that significantly improve the convergence rate using variance reduction techniques. These techniques include quasi-Monte Carlo, Latin hypercube sampling, antithetic variables and control variables. An overview of the techniques can for example be found in (Kroese et al., 2011). However, these techniques increase the complexity of programming involved and most of the methods assume that the input variables are either uniform on a hypercube, or simple in each dimension.

In this paper we introduce `RoseDist`, a software toolbox in `Python` that allows for construction of Monte Carlo schemes with most variance reduction techniques combined with multiple techniques for modeling dependencies between the variables. In `RoseDist` advanced dependencies can be modelled easily with both a large built-in collection of probability distributions and an easy to use constructor of custom distribution as building blocks. Methods for modelling such dependencies include a full transformation like the Rosenblatt method (Rosenblatt, 1952), which can map any samples to and from the unit hypercube, or an approximation like the Nataf (Nataf, 1962). Copulas are also used to determine parameter dependencies (Nelsen, 1999).

The software created is open source and can be downloaded from bitbucket as part of a larger package called `polychaos`; it can be downloaded from the webpage `https://bitbucket.org/jonathf/polychaos`. The reference documentation can be found at `http://home.simula.no/~jonathfe/polychaos/dist`. Importing the software is done as follows:

```
>>> from polychaos import dist as di
>>> import numpy as np
```

This paper is structured as follows. Section 2 outlines a tutorial giving an overview over how the software works through an example. In Section 3, the theoretical background of Rosenblatt transforms is described together with it's implementation counterpart. How to construct function estimates and inverses is described in Section 4, whereas Section 5 discusses how copulas can be integrated into the framework. Section 6 will describe how the software can be used to perform the various Monte Carlo schemes. Finally, conclusions and suggestion to further work is discussed in 7.

2 TUTORIAL

This section will illustrate how the toolbox `RoseDist` can be used to simulate a simple test case using various variance reduction techniques. To start off, consider the following as our model solver:

```
>>> def model_solver(q):
...     return q[0]*np.e**-q[1]
```

where `q` represents the model parameters. To define this as a uncertainty quantification problem let q be defined as a bivariate log-uniform random variable Q with an Ali-Mikhail-Haq copula; our goal is to quantify the model mean.

Constructing this copula in `RoseDist` can be defined as follows:

```
>>> dist = di.Iid(di.Loguniform(0,1), 2)
>>> Q = di.Ali_mikhail_haq(dist, theta=0.5)
```

To reduce the variance, we use Latin hypercube samples instead of standard psuedo-random generated samples. These samples can then be used to evaluate our model solver. These samples are generated as follows:

```
>>> samples_Q = Q.sample(1000, "L")
>>> samples_out = np.array([model_solver(s) for s in samples_Q.T])
```

Figure 1 shows both random samples and the density function.

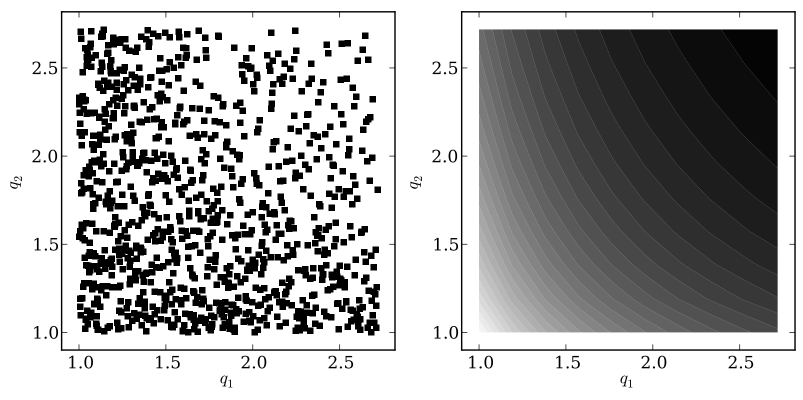

Figure 1: Scatters of Latin hypercube samples and probability density function of a log-uniform random variable with an Ali-Mikhail-Haq copula.

When estimating the mean, the use of a control variable reduces the variance still further. First, find a random variable R closely dependent on the estimate of interest with a known mean. In `RoseDist` dependent variables can be constructed through iso-probabilistic transformations. Any random variable can then be chosen as long as it is sufficently close to the quantity of interest. For our example, we choose to use a log-uniform distribution, since the distribution is already a part of the definition of the variable. As iso-probabilistic transformations do not change the number of variables, we select two control variables, one along each axis.

To create samples for the control variables, we use forward and inverse transformations between Q and R:

```
>>> R = di.Loguniform(0,1)
>>> samples_R1, samples_R2 = R.inv( Q.fwd( samples_Q ) )
```

We use this to create or control variables and implement them:

```
>>> alpha1 = np.cov([samples_out, samples_R1])[0,1]/di.Var(R)
>>> samples_out -= alpha1 * (samples_R1 - di.E(R))
>>>
>>> alpha2 = np.cov([samples_out, samples_R2])[0,1]/di.Var(R)
>>> samples_out -= alpha2 * (samples_R2 - di.E(R))
```

After the samples are ready, they can be used to create a mean:

```
>>> print np.mean(samples_out)
0.353670384341
```

3 Rosenblatt Transformations

Forward transform $\boldsymbol{F_Q}$	Inverse transform $\boldsymbol{F_Q^{-1}}$
$s_1{=}F_{Q_1}(r_1)$	$r_1{=}F^{-1}_{Q_1}(s_1)$
$s_2{=}F_{Q_2\mid Q_1}(r_2 \mid r_1)$	$r_2{=}F^{-1}_{Q_2\mid Q_1}(s_2 \mid r_1)$
$s_3{=}F_{Q_3\mid Q_1,Q_2}(r_3 \mid r_1, r_2)$	$r_3{=}F^{-1}_{Q_3\mid Q_1,Q_2}(s_3 \mid r_1, r_2)$
$\dots$	$\dots$
$s_N{=}F_{Q_N\mid Q_1,\dots,Q_N}(r_N \mid r_1,\dots,r_{N-1})$	$r_N{=}F^{-1}_{Q_N\mid Q_1,\dots,Q_N}(s_N \mid r_1,\dots,r_{N-1})$

Table 1: Forward and inverse Rosenblatt transformation

The fundamental building block of a random variable in `RoseDist` is the invertible Rosenblatt transform. Rosenblatt transforms are applied to a probability distribution to give an orthogonal decomposition that is measure preserving (Rosenblatt, 1991). The transformation has applications in goodness of fit tests and the modelling of residuals (Brockwell, 2007). Given a random variable $\boldsymbol{Q}$ the transformation is defined from the probability density function decomposition:

$$p_{\boldsymbol{Q}} = p_{Q_1,\ldots,Q_N} = p_{Q_1} p_{Q_2|Q_1} p_{Q_3|Q_1,Q_2} \cdots p_{Q_N|Q_1,\ldots,Q_{N-1}}. \tag{1}$$

By integrating each element in the vector along it's respective axis we get

$$F_{Q_n|Q_1,\ldots,Q_{n-1}}(q_n \mid q_1,\ldots,q_{n-1}) = \int_{-\infty}^{q_n} p_{Q_n|Q_2,\ldots,Q_{n-1}}(q \mid q_1,\ldots,q_{n-1}) \partial q. \tag{2}$$

This vector of cumulative distribution is used as a transformation. Table 1 shows a generic Rosenblatt transformation $\boldsymbol{F_Q} : \boldsymbol{r} \mapsto \boldsymbol{s}$ and its inverse. $\boldsymbol{F_Q}^{-1}$.

Distributions
Arcsin, Beta, Gamma, Hyperbolic Secant, Laplace, Logistic, Lognormal, Normal, Raised Cosine, Student-t, Triangle, Uniform, Weibull, Wigner

Table 2: List of built-in probability distributions.

The Rosenblatt transform is constructed via the cumulative distributions defined in (2), while predetermined distributions are listed in table 2. In addition, it is possible to construct user-defined distributions through the `di.construct` function. For example the minimum code to create an uniform random variable is as follows:

```
>>> Uniform_minimum = di.construct(
...     cdf=lambda self,q,a,b: (q-a)/(b-a),
...     bnd=lambda self,a,b: (a,b))
```

Here `cdf` is the dependent cumulative distribution function as defined in equation (2), `bnd` is a function returning the lower and upper bounds, and `a` and `b` are distribution parameters. As an alternative to a minimalistic construction, it is possible to make a more complex variable by adding a few additional keywords:

```
>>> Uniform = di.construct(
...     cdf=lambda self,q,a,b: (q-a)/(b-a),
...     bnd=lambda self,a,b: (a,b),
...     pdf=lambda self,q,a,b: 1./(b-a),
...     ppf=lambda self,u,a,b: u*(b-a)+a,
...     mom=lambda self,k,a,b: (b**(k+1)-a**(k+1))/(k+1)/(b-a),
...     defaults=dict(a=0., b=1.),
...     str=lambda self,a,b: ("u%s%s" % (a,b)))
```

If the keywords are not provided during construction, the distribution parameters are estimated as far as possible.

Creating dependencies in the model is done by inserting another distribution as a model parameter during initialisation. For example, to create a normal distribution with a uncertain mean, using our custom uniform variable, can be done as follows:

```
>>> u1 = Uniform(a=1, b=2)
>>> u2 = Uniform(a=0, b=u1)
>>> joint = di.J(u1, u2)
```

Note that the uniform distribution's upper parameter is a scaling parameter given that the lower parameter is 0. Since scaling is equivalent to multiplying the random variable, we could reformulize the joint distribution as follows:

```
>>> u3 = Uniform(a=0, b=1)
>>> joint = di.J(u1, u1*u3)
```

Since the primary building block is the Rosenblatt transformation, we are required to construct the structure given Table 1. This requirement is fulfilled by creating a bivariate distribution with one independent and one dependent variable. With the transform constructed, the sofware will be able to sort out all the dependencies at runtime.

4 Inverse Transformations and Approximations

One of the fundamental properties of the Rosenblatt transformation is that it can map random samples between distributions. Given that a random variable $\boldsymbol{Q}$ is mapped through its associated Rosenblatt transformation $\boldsymbol{F_Q}$, then the variable $\boldsymbol{U} = \boldsymbol{F_Q}(\boldsymbol{Q})$ will be uniformly distributed on the $[0,1]^N$ hypercube. The components $U_1, \dots, U_N$ will also be stochastically independent, irrespectively if $\boldsymbol{Q}$ was so to begin with. Likewise if $\boldsymbol{U}$ is uniformly distributed on the unit hypercube, using the inverse transformation $\boldsymbol{Q} = \boldsymbol{F}_{\boldsymbol{Q}}^{-1}(\boldsymbol{U})$ will have density $p_{\boldsymbol{Q}}$. With a double transformation, the random samples $\boldsymbol{Q} \sim p_{\boldsymbol{Q}}$ can be mapped to any density $p_{\boldsymbol{S}}$: First, map the samples to the unit hypercube $\boldsymbol{U} = \boldsymbol{F_Q}$ and then map them to the target distribution $\boldsymbol{S} = \boldsymbol{F}^{-1}(\boldsymbol{U})$. A difficulty with this procedure is that the inverse transformation is needed and in may be unavailable.

In `di.construct` an inverse can be provided through the `ppf` argument (point percentile function). If `ppf` is not provided, it is instead estimated using a root-finding algorithm. In this case, a hybrid between Newton-Raphson and the bisection method will be used. The former is used because of it's rapid convergence; the latter for it's convergence guarantee. If a Newton-Raphson iteration fails to converge, the bisection method provides a new start location a significant distance from the beginning of the previous iteration.

The hybrid method is described below. The objective function and it's derivative are defined as follows:

$$\begin{aligned} g(q) &= F_{Q_n|Q_1,\dots,Q_{n-1}}(q \mid q_1, \dots, q_{n-1}) - u \\ g^{'}(q) &= p_{Q_n|Q_1,\dots,q_{n-1}}(q \mid q_1, \dots, q_{n-1}), \end{aligned}$$

where u is the input to the point percentile function. The derivative follows from the definition in equation (2). From `bnd` we can select the following initial values $q = (q_u - q_l)u + q_l$. At each iteration q is updated using a Newton-Raphson step. If the new value is on the interval $[q_l, q_u]$, then accept it, else use a bisection increment $q = (q_u + q_l)/2$. In either case, q_l and q_u are updated using q and the sign of $g(q)$. The algorithm ends if either $g(x) \in [-\varepsilon, \varepsilon]$ for some tolerance level ε or the number iterations is superseded.

For the code to work, the probability density function has to be available. If the function is available analytically, this can be done by adding the `pdf` keyword to `pc.construct`. If the density is not available, it can be estimated through a finite difference scheme:

$$\begin{aligned} &p_{Q_n|Q_1,\dots,Q_{n-1}}(q \mid q_1, \dots, q_{n-1}) \approx \\ &\frac{F_{Q_n|Q_1,\dots,Q_{n-1}}(q + h \mid q_1, \dots, q_{n-1}) - F_{Q_n|Q_1,\dots,Q_n}(q \mid q_1, \dots, q_{n-1})}{h}, \end{aligned}$$

where h is a small constant.

5 Copulas

Supported Copulas	Ali-Mikhail-Haq, Clayton, Frank, Gumbel, Normal (Nataf), Joe

Table 3: The list of supported copulas

The Rosenblatt transformation is not the only way to model dependencies through an iso-probabilty transformation; other methods, such as copulas, are seeing increasing popularity. Copulas are especially useful for various applications for which there are complex non-linear relationships between parameters. Recent examples include regional and global modelling of climate (Laux et al., 2010), iron-ore mineral parameters

(Boardman and Vanna, 2011) and the relationship between returns and assests in finance (Dobri and Schmid, 2007).

An independent multivariate cumulative distribution function made dependent through an copula can for example be defined

$$F_{Q_1,\dots,Q_n}(q_1,\dots,q_n) = C(F_{Q_1}(q_1),\dots,F_{Q_n}(q_n)),$$

where C is the copula, and F_{Q_i} are marginal distribution functions. One of the more popular classes of copulas is the Archemedian copulas (Sklar, 1996). They are defined as follows:

$$C(u_1,\dots,u_n) = \phi^{[-1]}(\phi(u_1)+\dots+\phi(u_n)),$$

where ϕ is a generator and $\phi^{[-1]}$ is its pseudo-inverse. Fitting the Arcemedian copulas into the Rosenblatt transformation framework is also possible. This combination is done as follows:

$$\begin{aligned}
C_1(u_1) &= \phi^{[-1]}(\phi(u_1)) \\
C_2(u_2 \mid u_1) &= \tfrac{\partial}{\partial u_1}\phi^{[-1]}(\phi(u_1)+\phi(u_2)) \\
C_2(u_3 \mid u_1,u_2) &= \tfrac{\partial^2}{\partial u_1 \partial u_2}\phi^{[-1]}(\phi(u_1)+\phi(u_2)+\phi(u_3)) \\
&\vdots
\end{aligned}$$

These components then follow the structure of an inverse to a Rosenblatt transform and can be used directly in our software. In table 3, there is a list of the copulas currently supported in `RoseDist`.

6 Quasi-Monte Carlo and Variance Reduction Schemes

Sampling Schemes
Antithetic variables (Rubinstein and Kroese, 2007), <u>H</u>alton sequence (Halton, 1960), Ha<u>m</u>mersley sequence (Hammersley, 1960), <u>K</u>orobov latice (Korobov, 1957), <u>L</u>atin Hypercube sampling (McKay et al., 1979), (Pseudo-)<u>R</u>andom, <u>S</u>obol sequence (Sobol, 1967)

Table 4: List of different methods for generating (pseudo-)random, quasi-random and variance reducing techniques supported in the software.

With a fully functional Rosenblatt transformation available, samples can be mapped between any domain. However, there are no requirement that the samples should be of the traditional (pseudo-)random kind. Samples of any type, even quasi-Monte Carlo samples or samples from variance reduction techniques, like Latin hypercube sampling, can also be mapped. In table 4 a list of schemes for generating and manipulating samples on the unit hypercube that are supported in `RoseDist`.

Most of the methods in table 4 can be accessed by adding a letter as second optional argument to the instance method `sample`. The corresponding letter to each method is underlined. For example to create samples from the Halton sequence:

```
>>> print joint.sample(4, "H")
[[ 1.125       1.625       1.375       1.875     ]
 [ 0.5         1.26388889  0.30555556  1.04166667]]
```

The only exceptions are antithetic variables. With an antithetic variable, which axes to mirror has to be specified. In `RoseDist` the method is evoked by providing a list of boolean values as keyword argument. For examples for the user defined distribution in the last section we could mirror the first axis as follows:

```
>>> samples = joint.sample(4, antithetic=[True, False])
>>> print joint.fwd(samples)
[[ 0.38228023  0.94536045  0.61771977  0.05463955]
 [ 0.64698907  0.18033883  0.35301093  0.37072438]]
>>> print 1-joint.fwd(samples)
[[ 0.61771977  0.05463955  0.38228023  0.94536045]
 [ 0.35301093  0.81966117  0.64698907  0.62927562]]
```

By mapping the samples back the unit hypercube the antithetic effect can be observed along the first axis. In addition, constructing your own scheme is also possible. To do so, create samples on the unit hyercube and use the `inv` instance method to map samples to the target distribution.

7 CONCLUSIONS

This paper has introduced the `RoseDist` toolkit. In addition to being able to creating a Monte Carlo sampling scheme, the toolkit has support for: quasi-Monte Carlo methods; advance dependencies through Rosenblatt and copulas; and variance reduction techniques like Latin hypercube sampling, control variable and antithetic variables. The software is designed to be easy to use, but also be able to be easily extendable for user defined probability distributions and sampling scheme and is publicly available via `https://bitbucket.org/jonathf/polychaos`.

REFERENCES

Boardman, R. C. and J. E. Vanna (2011). A review of the application of copulas to improve modelling of non-bigaussian bivariate relationships (with an example using geological data). In *International Congress on Modelling and Simulation*.

Brockwell, A. E. (2007). Universal residuals: A multivariate transformation. *Statistics & probability letters 77*(14), 14731478.

Dobri, J. and F. Schmid (2007). A goodness of fit test for copulas based on rosenblatt's transformation. *Computational Statistics & Data Analysis 51*(9), 4633—4642.

Halton, J. H. (1960). On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik 2*(1), 8490.

Hammersley, J. M. (1960). Monte carlo methods for solving multivariable problems. *Annals of the New York Academy of Sciences 86*(3), 844874.

Korobov, N. M. (1957). The approximate calculation of multiple integrals using number theoretic methods dokl. *Acad. Nauk SSSR 115*, 10621065.

Kroese, D. P., T. Taimre, and Z. I. Botev (2011). *Handbook of Monte Carlo Methods*, Volume 706. John Wiley & Sons.

Laux, P., G. Jckel, R. M. Tingem, and H. Kunstmann (2010). Impact of climate change on agricultural productivity under rainfed conditions in CameroonA method to improve attainable crop yields by planting date adaptations. *Agricultural and Forest Meteorology 150*(9), 12581271.

McKay, M. D., R. J. Beckman, and W. J. Conover (1979). Comparison of three methods for selecting values of input variables in the analysis of output from a computer code. *Technometrics 21*(2), 239245.

Nataf, A. (1962). Dtermination des distributions de probabilites dont les marges sont donnes. *Comptes rendus de lacademie des sciences 225*, 42–43.

Nelsen, R. B. (1999). *An introduction to copulas*. Springer.

Rosenblatt, J. (1991). Automatic continuity is equivalent to uniqueness of invariant means. *Illinois Journal of Mathematics 35*(2), 339—348.

Rosenblatt, M. (1952). Remarks on a multivariate transformation. *The Annals of Mathematical Statistics 23*(3), 470–472.

Rubinstein, R. Y. and D. P. Kroese (2007, December). *Simulation and the Monte Carlo Method* (2 ed.). Wiley-Interscience.

Sklar, A. (1996). Random variables, distribution functions, and copulas: a personal look backward and forward. *Lecture notes-monograph series*, 114.

Sobol, I. M. (1967). On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics 7*(4), 86112.