A PRACTICAL GUIDE FOR THE INITIALISATION OF MULTI-AGENT SYSTEMS WITH RANDOM NUMBER SEQUENCES FROM AGGREGATED CORRELATION DATA

Volker Nissen$^{(a)}$, Danilo Saft$^{(b)}$

$^{(a)}$ $^{(b)}$ Ilmenau Technical University, Faculty of Economics, Institute for Commercial Information Technology,
Chair of Information Systems for Services (WI 2),
Postfach 100565, 98684 Ilmenau, Germany

$^{(a)}$volker.nissen@tu-ilmenau.de, $^{(b)}$danilo.saft@tu-ilmenau.de

ABSTRACT
This article describes a scalable way to initialise a simulation model with correlated random numbers. The focus is on the nontrivial issue of creating predefined multidimensional correlations amongst those numbers. A multi-agent model serves as a basis for practical demonstrations in this paper, while the method itself may be interesting for an even wider audience within the modelling and simulation community beyond the field of agent-based modelling. In particular, we show how researchers can create streams of correlated random numbers for different empirically-based model parameters when just given aggregated statistics in the form of a correlation matrix. An example initialisation procedure is demonstrated using the open source statistical computing software "R" as well as the open source multi-agent simulation software "Repast Simphony".

Keywords: MAS Parameterisation, Correlated Random Numbers, R-Project, Repast

1. INTRODUCTION
The simulation of a model may sometimes require a large amount of parameters, which influence its outcome (significantly). In a subset of these cases, the parameters may be interdependent in such a way that the initialisation of a model needs two or more parameters to correlate in a predefined manner. A procedure to generate and utilise such numbers will be explained in the following. We use the example of an agent-based model, since one of our main research areas is the field of agent-based economics. In this research, we regularly find illustrative scenarios to which the concept presented in this paper is applicable. Note that while the statements here will be kept limited to agent-based models for scientific validity, these explanations can easily be transfered to the initialisation of other types of simulation models.

In a variety of multi-agents systems, the model to be simulated may consist of a large number of heterogeneous agents. Heterogeneity can come in the form of different spatial positions of individual agents, different network connections, opinions, etc. In general, each of these agents possesses a set of parameters with different initialisation values. Researchers may want to relay data acquired from the real world (e.g. through measurement series, questionnaires or statistical archives) to initialise their agents with according parameter values for reasons of testing, prognosis, or simply for validity.

In the case of social or economic simulations, an agent may possess variables such as income, reputation, job satisfaction, household size, etc. Scientists may however not in all cases be lucky enough to find real-world-data at a level that is as detailed as their desired simulation setup may require. They, therefore, may need to evade to more aggregated forms of simulation, matching the aggregation level of the empirical data available. This option can be unsatisfactory as important details of the micro-level to be simulated and/or the emerging micro-macro-links within such a simulation might need to stay unaddressed.

One alternative is testing different random number distributions where detailed data is missing. This latter approach holds chances, but also challenges, such as the possible availability of empirical data that cannot give numbers on a detailed level, but gives aggregated distributions and correlations of different variables found in an empirical study. Table 1 (Oreg, 2006, p. 88) shows an instance of the results such empirical surveys yield. The values shown in table 1 were put forth by Oreg in 2006 and will serve as a data example throughout this paper. Table 1 gives descriptive statistics and correlations between parameters important to individual behaviour in the context of organisational change. The statistics were extracted from a series of questionnaires given to individuals within a company undergoing several organisational adjustments. The researches recorded variables thought to be important to an individual's opinion formation about an organisational change. They derived a static statistical model of interdependencies of individual properties (tb. 1, variables 1 to 9), the resistance an individual develops towards an organisational change (tb. 1, variables 10 to 12) and the behavioural outcome its opinion has on its specific job (tb. 1, variables 13 to 15).

From a multi-agent modelling perspective, it becomes possible to analyse the *dynamic* behaviour of a simulated company as a whole by modelling individuals

Table 1: Exemplary descriptive statistics and correlations for the variables of an empirical study by Oreg (2006, p. 88)

Variable	*Mean*	*SD*	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*
1. Age	45	12	1													
2. Manager	0.54	0.50	.17*	1												
3. Dispositional resistance to change	3.19	0.75	.19*	−.04	1											
4. Improvement in power-prestige	2.91	0.74	.11	.09	.05	1										
5. Improvement in job security	2.81	0.68	.20**	.11	.02	.48**	1									
6. Improvement in intrinsic rewards	3.12	0.67	.10	.14	.18*	.68**	.30**	1								
7. Trust in management	3.82	1.39	−.01	.09	−.03	.32**	.25**	.31*	1							
8. Information	3.93	1.34	.13	.19*	.04	.13	.05	.08	.45**	1						
9. Social influence (against the change)	3.71	1.44	−.01	−.13	−.05	−.27**	−.06	−.09	−.15	−.03	1					
10. Affective resistance	3.02	1.15	−.09	−.01	.31**	−.43**	−.31**	−.32**	−.33**	−.02	.33**	1				
11. Behavioural resistance	2.30	1.19	.01	.17*	.12	−.30**	−.19*	−.24**	−.30**	.03	.26**	.60**	1			
12. Cognitive resistance	4.21	1.11	.03	−.04	−.03	−.55**	−.24**	−.52**	−.52**	−.09	.23**	.46**	.50**	1		
13. Job satisfaction	5.74	0.92	.15	−.04	.00	.17*	.06	.21**	.17*	.19*	−.17*	−.15*	−.18*	−.02	1	
14. Intention to quit	2.42	1.26	−.15	.03	−.06	−.17*	−.16*	−.16*	−.24**	−.19*	.13	.23**	.15	.15*	−.49**	1
15. Continuance commitment	3.71	1.09	−.01	−.15	.33**	.10	−.03	.10	.02	−.04	.02	−.01	.11	−.12	−.11	.10

$^*p < .05$, $^{**}p < .01$.

with heterogeneous specific opinions and the (direct or indirect) influence individuals have on each other, e.g. through social interactions and information exchange (cmp. tb. 1, variables 8 and 9) or other behaviour affecting organisational "neighbours". In this example, the goal of researchers in the field of multi-agent-based simulation (MABS) would be to better understand the process of organisational change or analyse the effect of certain formal and informal hierarchies and network structures on company performance. In fact, such investigations are taking place within our own MABS research of which a first part has already been published (Nissen and Saft, 2010). We utilise the real-world study of "resistance to change" behaviour in organisations to initialise our own multi-agent simulation of a virtual organisation in order to better understand how resistance to change spreads and can be influenced by management. This requires (here: agent-based) modelling at a more detailed level than the aggregated statistical data in table 1 provides. To this end we can however use the values in table 1 to yield specific initialisation data for a (large) number of simulated members of a virtual organisation. To initialise each agent in our simulation model with its own personality values, it is necessary to extract sequences with specific random numbers while accounting for the multidimensional correlations given in table 1 in order to yield correct number sequences. This challenge is trivial only when one needs to generate two correlated series of random numbers, i.e. when the agents in the simulation only possess pairs of two correlated properties.

2. A LITTLE BIT OF MATH

For only two numbers (parameters) to be correlated, there is a simple approach to retrieve two correlated random numbers from a set of uncorrelated random numbers:

$$y_1 = \sigma_1 * x_1 \tag{1a}$$

$$y_2 = c * \sigma_2 + \sqrt{(1 - c^2)} * \sigma_2 * x_2 \tag{1b}$$

where x_1 and x_2 are two uncorrelated random numbers from a given distribution, σ_1 and σ_2 are their standard deviations, and c is the desired correlation coefficient between $y_1$and y_2; i.e. the resulting correlated random numbers.

With more than two sequences of correlated random numbers to generate, one can use a variety of mathematical approaches that are more or less difficult to go through manually. In our case, an Eigenvector-decomposition was employed as we found it to be a process that is robust and offers good performance. There also is the option of using the so-called Cholesky-decomposition (Lloyd and Trefethen, 1997, pp. 172 - 178) which will however not be explained further here. Given a correlation matrix C (see columns "1" through "14" in table 1) one can define a matrix

$$V = E_i Diag(\sqrt{\lambda_\mathrm{i}}) \tag{2}$$

where E_i are the eigenvectors of C and λ_i are the eigenvalues of C.

With a matrix I_u consisting of formerly uncorrelated random numbers, we can derive a matrix

$$I_c = I_u V^T \tag{3}$$

where V^T is the transpose of V .

I_c then contains random numbers with correct correlations.

I_u can consist of any number of random values where each line can represent the initialisation values for a single agent and each column stands for one of the correlated parameters of an agent. This offers great flexibility since one only needs to choose the number of rows in the original matrix I_u as big as the number of agents one wishes to instantiate/simulate. I_u should already include the general distribution properties such as – in our case - the mean and standard deviations given in table 1.

3. A PRACTICAL GUIDE FOR THE UTILISATION OF CORRELATED RANDOM NUMBER SEQUENCES USING “R” AND “REPAST”

While the pattern to generate correlated random numbers shown in section 2 can be time-consuming to do by hand, it is a very easy process once one employs supporting software. The popular and well-documented open source program “R” (Hrishikesh, 2010; Jones, 2009) is able to make the necessary calculations (for all practical purposes implied here) in just a fraction of a second. The tool, along with many additional packages, can be downloaded freely for a variety of platforms (The R Project, 2010).

Below, we will present exemplary step-by-step R code to generate random number sequences for the three correlated parameters “improvement in power-prestige”, “improvement in job security”, and “trust in management” listed in table 1. The code is to generate correctly correlated random numbers for 2500 agents as virtual “employees” of a simulated organisation. Note that the code pattern is scalable to a large number of parameters and agents.

The first step is to generate a matrix of uncorrelated random numbers for each agent, already taking into account the correct mean and standard deviation properties (in this case assuming the Gaussian distribution from table 1):

```
//create a matrix with 3 columns for 3
//parameters and 2500 rows for 2500
//agents:
Iu <- matrix(, ncol=3, nrow=2500)

//fill in random values for “power and
//prestige”,…:
Iu[,1] <- rnorm(2500, 2.91, 0.74)

//…then “job security”,…:
Iu[,2] <- rnorm(2500, 2.81, 0.68)

//… and finally “trust in management”:
Iu[,3] <- rnorm(2500, 3.82, 1.39)
```

Next, the correlations for these parameters need to be filled into another matrix *C*:

```
//create a squared matrix and fill in the
//correlations for each of the three
//correlated parameters, using the order
//“power-prestige”, “job satisfaction”,
//and “trust in management”:
C <- matrix(, ncol=3, nrow=3)
C[1,] <- c(1, 0.48, 0.32)
C[2,] <- c(0.48, 1, 0.25)
C[3,] <- c(0.32, 0.25, 1)
```

R offers a simple command to calculate both the eigenvalues and eigenvectors of a matrix. We will save the result of this operation in an object *E*. The parameter “symmetric” refers to *C* being a symmetrical matrix so that only the lower triangle of the matrix needs to be used in the calculations:

```
E <- eigen(C, symmetric=TRUE)
```

The call to *E$vectors* will then give us the eigenvectors of *C* and *E$values* will return the eigenvalues accordingly. We use the command “diag” to construct a fictive matrix diagonal from the square roots of the three eigenvalues of *C*. This call is necessary for a valid multiplication. Note that for the calculation of the square roots to be valid, all eigenvalues of *C* must be positive. This will however implicitly be the case for valid correlation matrices. We can then create the matrix V according to equation 2 given in section 2:

```
V <- E$vectors %*% diag(sqrt(E$values))
```

Finally, we can multiply our formerly uncorrelated random number matrix I_u with the transpose of V in order to receive a matrix I_c with 2500 rows each containing three correctly correlated values in three columns, where (in our example) the first column stands for the parameter “power and prestige”, the second for “job security”, and the last one for “trust in management”:

```
Ic <- Iu %*% t(V)
```

The result of this code can be saved in a CSV-File. In order to do this, we can use the “write.table” command included in R. “write.table” takes several parameters of which the first is the matrix to write into a file and the second is the file’s name on disk. The parameter “sep” defines a character by which the values of the matrix are to be separated in the output file. We use “col.names=FALSE” and “row.names=FALSE” to specify that we do not wish to export any column or row names. In order to not put any values of the matrix in quotes, we set “quote=FALSE”. In case a value is not set in the matrix (which, following the aforementioned steps, ought to be irrelevant in our case), we can specify a string value written to the file in its place. Here, for instance, we could use the Java-compatible “NaN” string for “not a number” by setting the parameter “na” accordingly:

```
write.table(Ic, "C:/filename.csv", sep=",",
col.names=FALSE, row.names=FALSE, quote=FALSE,
na="NaN")
```

We can now use the random number sequences in this file for use in any external program, in our case the “Recursive Porous Agent Simulation Toolkit” Repast Simphony (North et al., 2006). It is a Java-based open source software with seemingly growing popularity in the MABS-research community (Barnes, 2010) and is available as a free download (REPAST, 2010).

Repast employs a so-called Context Creator to initialise simulations. Within this class, agents can be created and parameters may be set before the simulation begins. Skipping over most of the code of our initialisation routine, we will again present exemplary code to assign the generated correlated values to each agent using the Context Creator. In our example, we wish to create a virtual organisation with 2500 employees, each having different, but correlated parameters as explained above. We utilise a CSV-reader class that simply returns the matrix saved by R as a two-dimensional Java Double array[1].

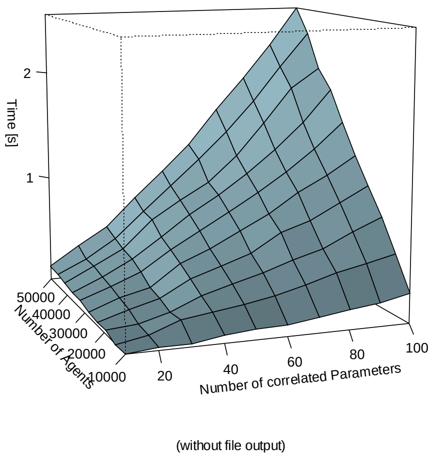
```
Double[][] correlatedRandomNumbers =
CSV_Reader.readFile("C:/filename.csv");
```

All we then need to do is to create our agents and read out the correlated random values in the correct order:

```
//stylised iteration:

for (int i=0; i<2500; i++) {
  EmployeeAgent ea = new EmployeeAgent();
  ea.powerPrestige =
correlatedRandomNumbers[i,0];
  ea.jobSecurity =
correlatedRandomNumbers[i,1];
  ea.trustInMgmt =
correlatedRandomNumbers[i,2];
  …
}
```

The approach itself is very flexible and scalable to a large number of agents and correlated parameters. Performance tests were conducted on an Intel Core2-Duo pc with 4GB memory and a hard disk spinning at 5400rpm. Figure 1 shows a 3D-mesh-plot portraying the execution times for the generation of 10.000 to 50.000 correlated vectors (i.e. agents) for respectively 10 to 100 parameters for each individual. The upper part of figure 1 displays execution times for the calculations themselves without disk output to a CSV file. The bottom plot shows execution times including the disk output. The data shows that even for the calculation of 100 correlated parameters for 50.000 agents it only takes slightly over two seconds to retrieve all necessary values using a code analogous to the example code listed above. Also, the computation complexity seems to rise only linearly with a rising number of agents and parameters which makes this approach interesting for large scale simulations with either a large number of agents or parameters in one simulation, or a large number of simulations running in parallel (e.g. for parameter optimisation purposes). Note that there is, however, a performance-bottleneck where the files need to be written to disk. Therefore, using many parameters or agents, one should refer to Lang (2005) or JRI (2011) for a way to directly call R-functions from within Java-



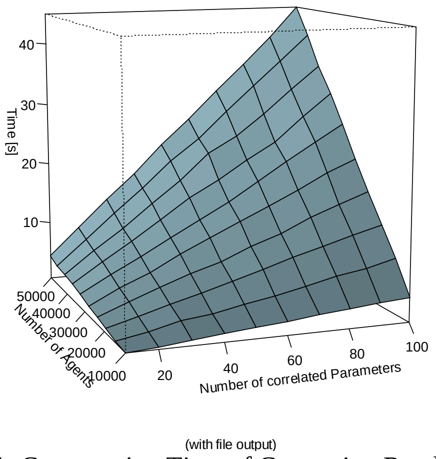


Figure 1: Computation Time of Generating Random Correlated Number Sequences for Varying Numbers of Agents and Parameters with (bottom) and without (top) Disk Output to a CSV File.

based applications such as Repast. We will explain this process briefly in the following.

The so-called Java-R-Interface (JRI, 2011), amongst other interfaces available, is able to send commands to an instance of R running in the background of a Java-based application. Since simulations in the MAS-tool Repast Simphony can be programmed in the Java language, JRI can be easily implemented for use in such multi-agent-simulations.

JRI is available as part of a package-extension of R called "rJava", which was originally designed to send data and commands in the opposite direction, i.e. from R to Java. The quickest method to utilise only the functionality of the Java-R-Interface, nevertheless, is to install the complete "rJava" package within R. Once installed, the downloaded folders within R's own extension library will contain the JRI Java-archive for implementation as a library in any Java project, too. Note that the subsequent setup of the JRI library files can be difficult. For a step by step guide based on

[1] Several similar Java-based CSV-readers are also available online.

Eclipse/Repast, please refer to Shah (2009). Once JRI is available for use in Repast, one can extend the Context Creator class of a simulation analogously to the case of CSV files described above. However, here one would directly initialise agents using the calculations made in R. For our example case, the necessary code is listed below:

As a first step, it is important to make the JRI library available to the Repast simulation project and include it in the import statements of the simulation part needing to access R, i.e. the ContextCreator.java class file in our case:

```
import org.rosuda.JRI.REXP;
import org.rosuda.JRI.Rengine;
import org.rosuda.REngine.*;
```

We are then using the initialisation routine of the Context Creator class (e.g. the "*build()*" method) to access R routines. Here, we first create an object of type *Rengine* which is the main instance passing commands and data between Java and R. The constructor of this class can pass various arguments to the instance of R to be used. Please refer to the JRI documentation (JRI, 2011) at this point in order to adjust this step for your requirements. In all cases, the "*waitForR()*" function of the newly created *Rengine* object should be called to make sure that the R thread finished its program start sequence before calculations can begin. Not including a call to this method may lead to Java exceptions being raised at this step and the failure of Repast's Context Creator initialisation routine:

```
//creating a new instance of R for
//calculations:
Rengine re = new Rengine(new
String[]{""}, false, null);
//important: waiting for the R instance
//to finish loading:
if (!re.waitForR()) {
  System.out.println("Cannot load R");
}
```

The *Rengine* type now offers the method "*eval(String command)*" (amongst numerous others) to execute and evaluate a command passed over to R in the form of a simple string parameter. This method returns an object of type *REXP* (R expression), which can subsequently be used to output or further interpret results of the command sent via "*eval*". The following code listing demonstrates several calls to send commands to R analogous to the example R-code already given above. We retrieve the final calculation of matrix I_c within the object *ex* of type *REXP*:

```
//creating instance for return values:
REXP ex;
//executing R example code as stated in
//beginning of section 3 of this paper:
re.eval("Iu <- matrix(, ncol=3,
nrow=2500)");
re.eval("Iu[,1] <- rnorm(2500, 2.91,
0.74)");
re.eval("Iu[,2] <- rnorm(2500, 2.81,
0.68)");
re.eval("Iu[,3] <- rnorm(2500, 3.82,
1.39)");
re.eval("C <- matrix(, ncol=3,
nrow=3)");
re.eval("C[1,] <- c(1, 0.48, 0.32)");
re.eval("C[2,] <- c(0.48, 1, 0.25)");
re.eval("C[3,] <- c(0.32, 0.25, 1)");
re.eval("E <- eigen(C,
symmetric=TRUE)");
re.eval("V <- E$vectors %*%
diag(sqrt(E$values))");

//catching the final result of Ic in
//"ex" for further handling:
ex =re.eval("Ic <- Iu %*% t(V)");
```

Now we only need to create an array just as one would when using CSV files. The *REXP* type has several routines for formatting the results, e.g. an "*asString()*" method for outputting its contents to the Java console. Here, we use the "*asMatrix()*" method that interprets the results as a two-dimensional array of *Double* values. We can then again use this array to iterate through it, assigning the parameter values to each of our agents:

```
Double[][] correlatedRandomNumbers =
ex.asMatrix()
//stylised iteration:
for (int i=0; i<2500; i++) {
  EmployeeAgent ea = new
  EmployeeAgent();
  ea.powerPrestige =
  correlatedRandomNumbers[i,0];
  ea.jobSecurity =
  correlatedRandomNumbers[i,1];
  ea.trustInMgmt =
  correlatedRandomNumbers[i,2];
  …
}
```

Accessing R through the JRI library is a very efficient method to compute the necessary calculations for the initialisation of a Repast simulation. Further performance tests using this method have shown that there is no loss of performance present when using JRI rather than R itself to create large numbers of correlated random values. Performance results are comparable to those shown in the top part of fig. 1.

4. CONCLUSIONS AND EXTENSIONS

This article dealt with the question of how to generate scalable sets of correlated random numbers for the initialisation of agent-based simulations. The authors found this to be a question both important and difficult as many empirical studies provide aggregated descriptive statistics, including correlations, while there is also a necessity for detailed simulations at the level of single and interacting individuals to explore certain issues especially when dealing with complex and/or emergent systems (Nissen and Saft, 2010, p. 113). The process of extracting correctly correlated sequences of random numbers for each agent is nontrivial and literature on this topic, especially in the form of practical guides for researchers in the (multi-agent) simulation community without a deep mathematical background, is scarce. The reader therefore was provided with a step-by-step guide for how to create a matrix containing MAS initialisation data in the form of correlated random number sets for each agent as well as with a stylised example code for the wide-spread Repast simulation software in order to access those values indirectly via file output or directly using the so-called JRI-package. This paper therefore serves as a demonstrative guide for a wide audience of researchers in the simulation community. It provides a time-saving way as well as a quick access for newcomers to the creation of correlated random number sequences for MAS parameterisation.

The steps shown here can further be enhanced by using additional packages, e.g. the R-Commander package for R, providing quick and easy access to basic R operations via a graphical user interface (Fox, 2010). There are also other options to call R directly from Java and Java-based software (Lang, 2005), eliminating the need to use CSV Files for storage. Such approaches are beneficial since one is not only able to outsource initialisation calculations, but any complex set of calculations that should rather be executed in a professional environment such as R.

REFERENCES

Barnes, D. J.; Chu, D., 2010. ABMs Using Repast and Java, *Introductio to Modeling for Biosciences*, Springer London, 79-130.

Fox, J., 2010, *The R Commander: A Basic-Statistics GUI for R*, Available from: http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/ [accessed 01 May 2011]

Hrishikesh D., 2010. Advances in Social Science Research Using R. *Lecture Notes in Statistics*. Springer.

Jones, O.; Maillardet, R., Robinson, A.:, 2009. *Introduction to Scientific Programming and Simulation Using R*. Chapman & Hall/CRC, Boca Raton (Florida, USA).

JRI (without date), *JRI* Available from: http://www.rforge.net/JRI/ [accessed 01 May 2011]

Lang, D.T., 2005, *Calling R from Java*, Available from: http://www.omegahat.org/RSJava/RFromJava.pdf [accessed 01 May 2011]

Lloyd N., Trefethen, D.B., 1997. *Numerical Linear Algebra. Society for Industrial and Applied Mathematics*

Nissen, V.; Saft, D., 2010. Social Emergence in Organisational Contexts: Benefits from Multi-Agent Simulations. In: Madey, G.R.; Sierhuis, M.; Zhang, Y. (Eds.): *Proceedings of the Agent-Directed Simulation Symposium*, San Diego: SCS, 2010, 106 – 113 (CD).

North, M.J.; Collier, N.T.; Vos, J.R., 2006. Experiences creating three implementations of the repast agent modeling toolkit. *ACM Trans. Model. Comput. Simul.*, 16(1), 1-25

Oreg, S., 2006. Personality, context, and resistance to organisational change. *The European journal of work and organisational psychology*, (15), 73-101

REPAST Developers Group, (2010), *Repast Home Page,* Available from: http://repast.sourceforge.net [accessed 01 May 2011]

Shah, M., 2009, *R and Java – JRI using eclipse*, Available from: http://mithil-tech.blogspot.com/2009/11/r-and-java-jri-via-eclipse.html [accessed 01 May 2011].

The R Project for Statstical Computing, (2010), *The R Project for Statistical Computing*, Available from: http://www.r-project.org/ [accessed 01 May 2011]