Proceedings of the 2020 Winter Simulation Conference
K.-H. Bae, B. Feng, S. Kim, S. Lazarova-Molnar, Z. Zheng, T. Roeder, and R. Thiesing, eds.

ANIMATION FOR SIMULATION EDUCATION IN R

Vadim Kudlay
Barry Lawson

Department of Mathematics and Computer Science
University of Richmond
Richmond, VA 23173, USA

Lawrence M. Leemis

Department of Mathematics
The College of William & Mary
Williamsburg, VA 23187, USA

ABSTRACT

R is freely-available software for statistical computing, providing a variety of statistical analysis functionality. In prior work, we introduced and released the `simEd` package for R, focusing on functions for generating discrete and continuous variates via inversion, for extensible single- and multiple-server queueing simulation, and including real-world data sets for input modeling and analysis. In this current work, we significantly enhance and extend the `simEd` package, primarily through the introduction of a variety of animation and visualization utilities to aid in simulation education. These include animations of event-driven simulation details for a single-server queueing model, of random-variate generation for a variety of distributions, of a Lehmer random number generator, of variate generation via acceptance-rejection, and of generating a non-homogeneous arrival process via thinning.

1 INTRODUCTION

In previous work, we argued for the use of R for a first course in discrete-event simulation that emphasizes programming in a high-level language while also covering probabilistic and statistical aspects in simulation. We first presented R functions for queueing-specific contexts (Lawson and Leemis 2015) and then discussed the release of a publicly-available R package, `simEd` (Lawson and Leemis 2017a; Lawson and Leemis 2017b). We, the authors, have repeatedly used these utilities in undergraduate- and graduate-level courses in simulation at our respective institutions, as well as in various external workshops we have offered. It is in the context of using these primarily text-based utilities in teaching, while simultaneously discussing concepts using hand-sketched board diagrams and/or slideshow presentations, that we realized the benefit that easy-to-use interactive computer animations of these concepts would provide. Therefore, we have developed a suite of new functions to add animation and visualization to the `simEd` package, and have extended some of the previous functions to now include animation.

In this paper, we introduce significant animation capabilities for the `simEd` package, written with a focus on simulation pedagogy. The updated package includes:

- an interactive function specifically focused on an event-driven implementation of an $M/M/1$ queue, which includes animation of event calendar updates, of corresponding variate generation, of the queue itself, and of time-persistent and observation-based statistics (see Table 1 in Appendix A);
- updates to text-only $M/M/\{1,k\}$ queueing functions to now include animation (see Table 1);
- an interactive function for visualizing $U(0,1)$ variate generation using a Lehmer random number generator (see Table 2);
- interactive functions for visualizing variate generation via acceptance-rejection and for visualizing generation of a non-homogeneous Poisson process via thinning (see Table 2); and

- additional and improved functions for visualizing inversion for variate generation of various discrete and continuous distributions (see Table 3).

These functions are described in more detail throughout this paper, with illustrative examples. The `simEd` package is available on the Comprehensive R Archive Network (CRAN) [1] (The R Foundation 2017).

As before, our work does not focus on the practice of simulation, as other R packages do, e.g., the process-oriented general simulation framework `simmer` (Ucar and Smeets 2020), or the `arena2r` package for interactively visualizing results from Arena in R (de Lima 2018). Rather, we focus on tools specifically to be used for teaching Monte Carlo and discrete-event simulation.

2 ANIMATION OF EVENT-DRIVEN QUEUING SIMULATION DETAILS

In the updated `simEd` package, we have developed a custom interactive animation of an event-driven implementation of an $M/M/1$ queuing model. With the new function `ssqvis`, the user can rely on default exponential interarrival and service functions, or can define and provide their own custom functions to pass to `ssqvis`. The animation is split into three rows:

- the top row depicts the event calendar along with generating arrival and completion-of-service times using inversion;
- the middle row depicts a fairly typical queue visualization (with customers arriving, queueing, entering service, and departing the system), along with the simulation clock and progress meter;
- the bottom row depicts the number in system, queue, and service across time (i.e., so-called "skyline" functions) along with observation-based statistics.

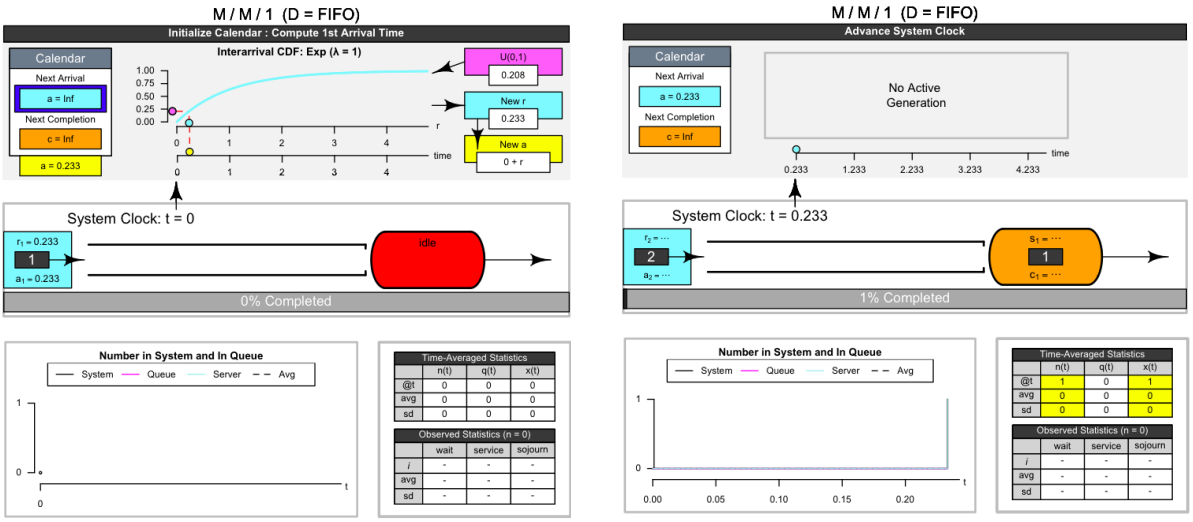
Via the R console, the user is able to proceed step-by-step from simulation initialization, through processing various types of events; is able to inspect the statistics of individual customers; and is able to jump ahead in simulated time to a particular customer.

Figure 1(a)'s top row depicts initialization of the event-driven simulation, in which the calendar is updated using the first (inter)arrival time obtained by inversion. Note that the inversion figure depicts a $U(0,1)$ variate on the vertical axis, the corresponding interarrival time inverted across the exponential cdf, with the corresponding arrival time depicted on the time axis displayed below the inversion figure. Specific values and details are displayed on the right, with the arrival time ultimately pushed onto the calendar on the left. In Figure 1(b), all three rows of the visualization are updated to demonstrate processing of that first arriving customer, including advancing the system clock (top row), customer entering service (second row), and statistics updating (third row). Note that the customer has entered service (orange), but no service time has yet been generated.

Figure 2(a) proceeds to depict further processing of the first arriving customer, in which the service time and completion time for the first customer in the system are generated. The top row shows generation of the service time using inversion, and updating of the calendar. The middle row shows updating of the first customer in service (now green), as a service time and completion time are now known. Figure 2(b) continues the processing of the first arriving customer, in which the arrival time of the next customer is generated. The top row shows generation of the (inter)arrival time and updating of the calendar. Note that in the middle row, the next customer (on the left, yet to arrive) is also indicated by the future arrival time. Note also that the skyline functions and observation-based statistics are updated in the third row.

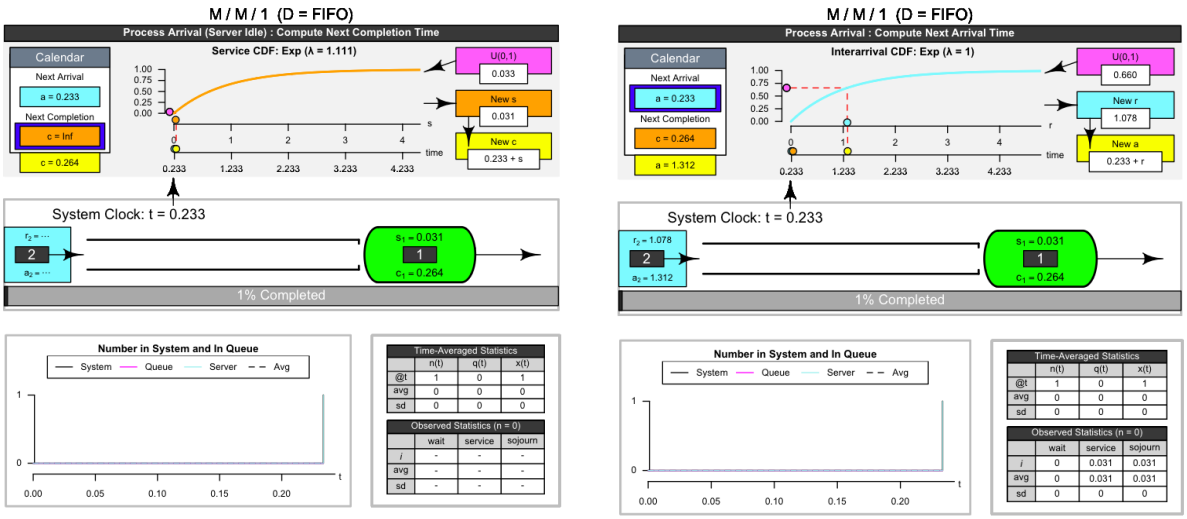
Figure 3(a) depicts the animation after having skipped ahead to the 29th customer. Note that this customer has been queued (middle row), the arrival time for the 30th customer has been generated and is being placed on the calendar (top row), and skyline functions updated appropriately (third row). Finally,

[1] Note to Reviewers: As of submission, the fully-updated package is not yet posted to the CRAN, but is available at http://bit.ly/simEd_wsc2020.



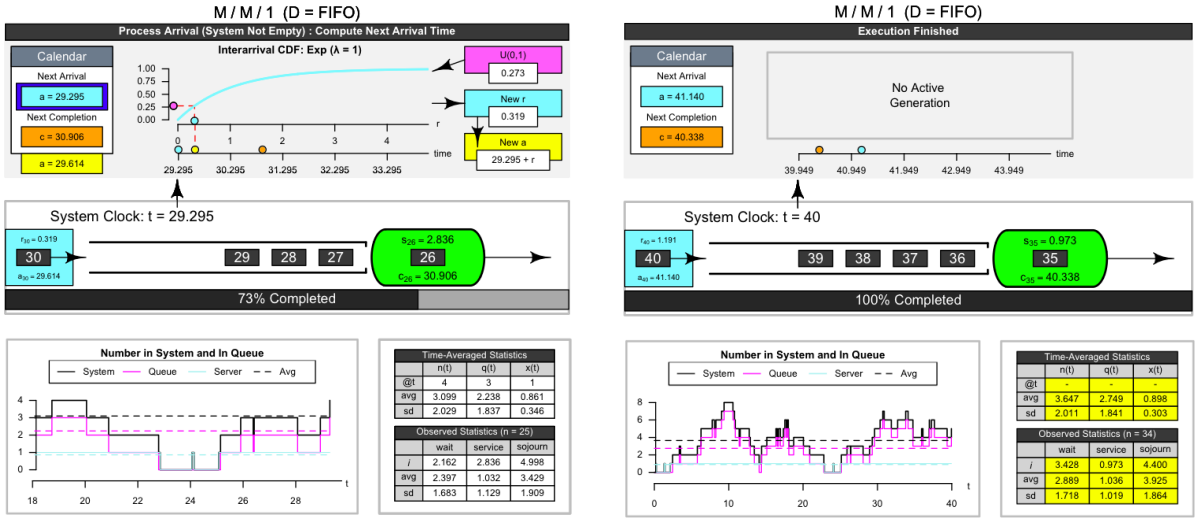
(a) (b)

Figure 1: Screen captures of the `ssqvis` animation depicting event-driven details of an $M/M/1$ queue implementation. (a) The top row of the animation display depicts, interactively, event calendar initialization, generating the first customer arrival time using inversion. (b) All three rows are updated to depict processing of the first arriving customer, including advancing the system clock (top row), customer entering service (second row), and statistics updating (third row).



(a) (b)

Figure 2: Screen captures of the `ssqvis` animation depicting event-driven details of an $M/M/1$ queue implementation. (a) The first and second rows are updated to depict generating the service time and completion time for the first customer in the system. (b) All three rows are updated to depict scheduling the second arriving customer. Note also that observed statistics are updated based on the previously generated service time.



(a) (b)

Figure 3: Screen captures of the `ssqvis` animation depicting event-driven details of an $M/M/1$ queue implementation. (a) Processing of the 29th customer to arrive, which waits third in the queue with a customer already in service. (b) State of the simulation at the end of the specified maximum simulation time of 40. Note that the skyline functions and observation-based statistics represent final values.

Figure 3(b) depicts the animation at the end, for a maximum simulated time of 40. Note that the skyline functions and observation statistics are updated to represent the entirety of the simulation run (third row).

3 QUEUEING ANIMATION

In addition to the new function visualizing event-driven simulation discussed in the previous section, we have also extended the previously text-only but extensible functions `ssq` (defaulting to $M/M/1$) and `msq` (defaulting to $M/M/k$) to now include animation as an additional feature.

Figure 4(a) depicts queueing animation using the `ssq` function with default arrival and service processes ($\lambda = 1$, $\mu = 10/9$) and a maximum simulated time of 40. As shown in the top row of the visualization, the 39th customer has arrived and been queued, as the 35th customer is still in service at time 40. The skyline functions for number in the system, number in queue, and number in service are also displayed in the bottom row. Similar to the `ssqvis` function, the user can interactively step one customer at a time.

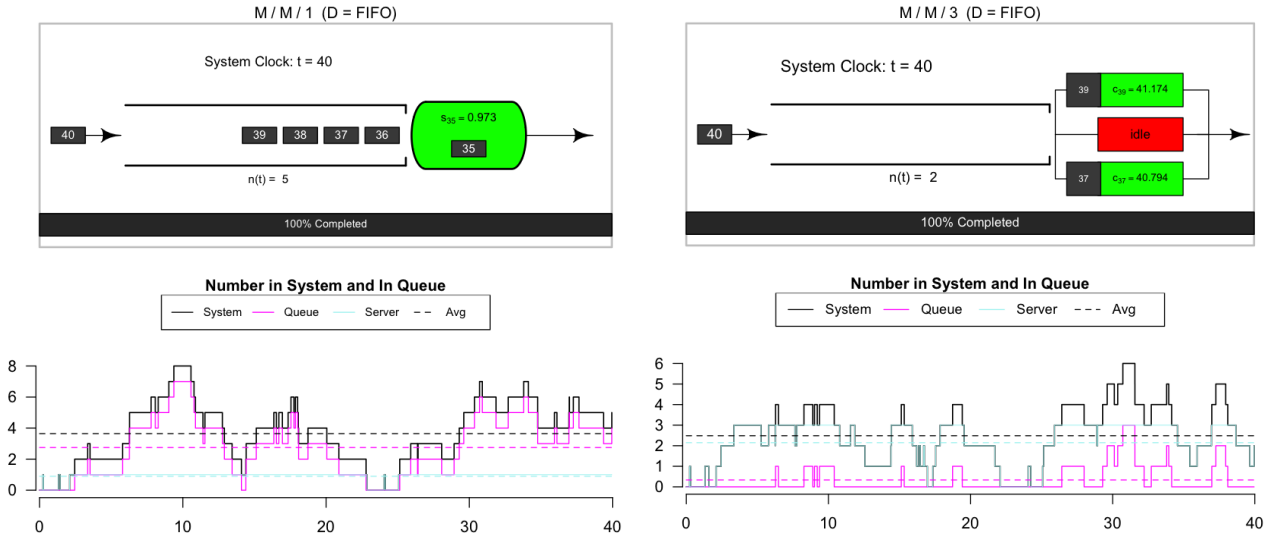
Similarly, Figure 4(b) depicts queueing animation using the `msq` function with a default arrival process ($\lambda = 1$) and a custom service process ($\mu = 1/2$) having higher average service time than for `ssq` above, but with three servers and a maximum simulated time of 40. In this example, the 37th and 39th customers are still in service, the 38th customer has already departed, with corresponding skyline functions shown below for the entire simulation run.

4 ADDITIONAL ANIMATION ROUTINES

In this section, we describe additional routines added to the `simEd` library that are particularly useful in an introductory simulation course.

4.1 Animation of a Lehmer Random Number Generator

Although outdated for modern simulation applications, because of its simplicity, a Lehmer (multiplicative linear congruential) random number generator provides an ideal entry point for students to understand



(a) (b)

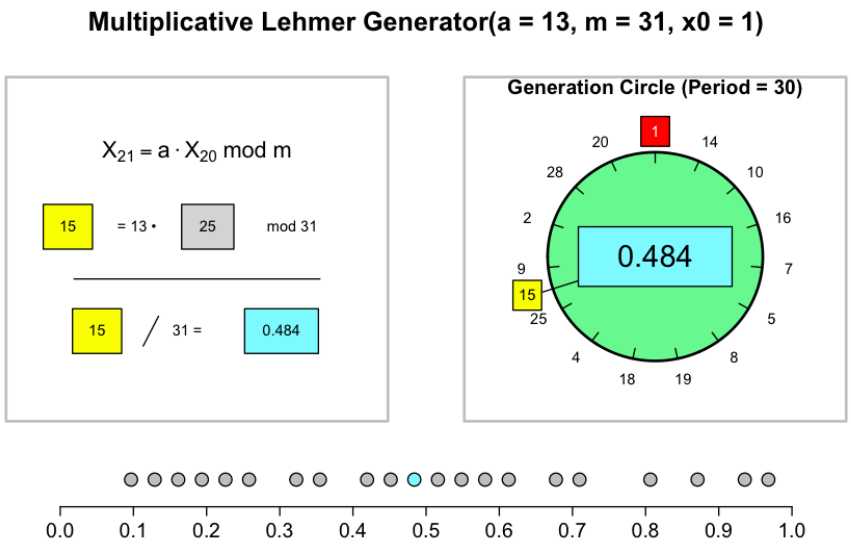
Figure 4: The (a) `ssq` function and (b) `msq` function, each with animation activated, depicting the queueing node and the "skyline" functions of time-persistent statistics for an $M/M/1$ and $M/M/k$ queue respectively. In interactive mode, the node and skyline functions are updated with each arriving and departing customer.

details of random number generation. Visualizing the period of the generator as a virtual circle (based on the choices of multiplier a, modulus m, and initial seed x_0) allows student to easily grasp the notions of generator periodicity, period length, choice of initial seed, and drawing integers without replacement to produce $U(0,1)$ random variates. In our experience, a general understanding of a Lehmer generator helps students with an implicit understanding of more modern generators (e.g., Mersenne twister) even without fully exploring the underlying complex details of the latter.

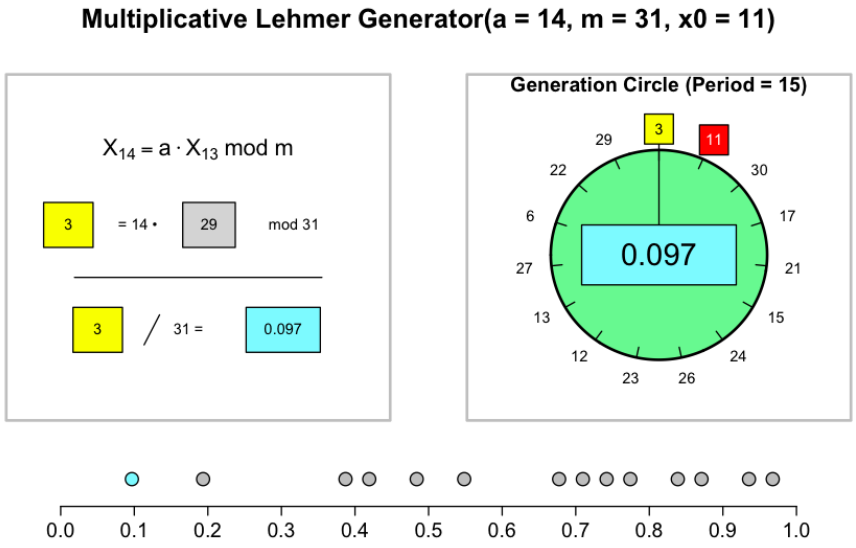
For this reason, we have provided the `lehmer` function for visualizing a Lehmer random number generator. Figure 5(a) depicts a Lehmer generator with multiplier $a = 13$, prime modulus $m = 31$, and initial seed $x_0 = 1$. The sequence of integers drawn without replacement from $\{1, 2, \ldots, m-1\}$ are depicted in the circle on the right, with red indicating the initial seed and yellow indicating the current state of the generator. The box on the left shows the generation of the current integer state using the previous state, and also shows mapping of that integer into a real-valued number in $(0,1)$. As each new random variate is generated, it is depicted by a filled circle at the corresponding location on the $(0,1)$ axis at the bottom, with teal indicating the current value, and (eventually) red indicating the reappearance of the initial seed. Figure 5(b) demonstrates how the `lehmer` function can be used to experiment with periodicity, in this case changing the multiplier to $a = 14$, which results in a less-than-full period corresponding to $m = 31$.

4.2 Animation of Variate Generation via Acceptance-Rejection

When discussing variate generation, an instructor might typically cover inversion via closed-form expression (e.g., for uniform or exponential), convolution (e.g., for Erlang), as well as numerical approximation (e.g, for normal). Other distributions, such as Beta, motivate an acceptance-rejection approach, for which we have provided the `arsample` function in `simEd`. With the `arsample` function, the user can rely on the default pdf (i.e., Beta($\alpha = 3$, $\beta = 2$)) and default constant majorizing function, or can specify custom pdf and/or majorizing function — see Table 2. The function then interactively demonstrates variate generation via acceptance-rejection.

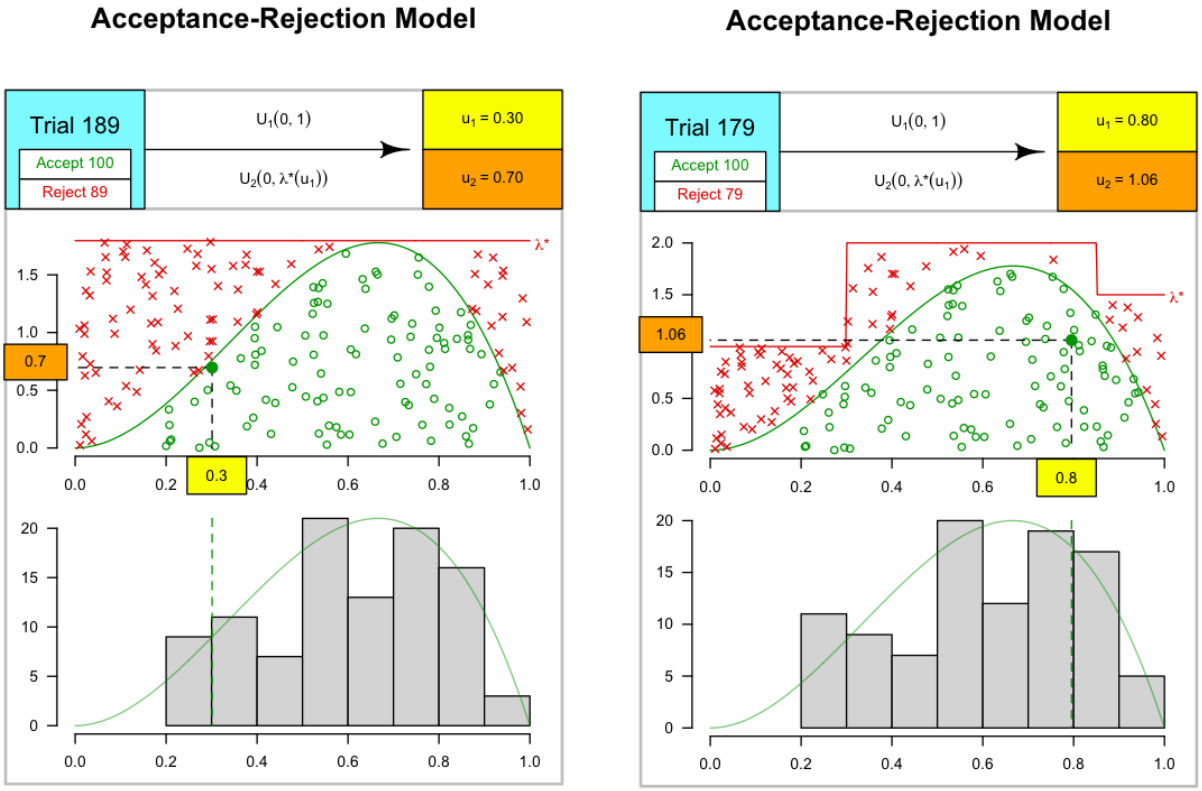


(a)



(b)

Figure 5: The `lehmer` function depicts, interactively, the generation of a $U(0,1)$ random variate using a multiplicative linear congruential generator. The user can select the choice of multiplier a, modulus m, and initial seed x_0. This figures depicts interactively generating variates using a small prime modulus $m = 31$ with (a) full-period multiplier $a = 13$ and initial seed $x_0 = 1$ or (b) less-than-full-period multiplier $a = 14$ and initial seed $x_0 = 11$.



(a) (b)

Figure 6: The `arsample` function interactively demonstrates variate generation via acceptance-rejection, here using the default Beta($\alpha = 3$, $\beta = 2$) distribution. The user can specify the distribution from which to generate, as well as a corresponding majorizing function. (a) A constant λ^* majorizing function. (b) A piecewise-constant $\lambda^*(t)$ majorizing function.

Figure 6(a) depicts the generation of 100 Beta($\alpha = 3$, $\beta = 2$) variates using acceptance-rejection with a constant majorizing function $\lambda^*(t)$. At each step, a $u_1 = U(0,1)$ variate is generated (yellow) to determine location on the horizontal axis, and then a $u_2 = U(0, \lambda^*(u_1))$ variate is generated. If u_2 is below the provided pdf (green curve), the u_1 variate is accepted (indicated by an open green circle) and contributes to the updated histogram in the bottom row. If u_2 is above the provided pdf, the u_1 variate is rejected (indicated by a red X). Note that the total number accepted and rejected is shown in the top of the display.

Figure 6(b) depicts the generation of 100 variates from the same Beta($\alpha = 3$, $\beta = 2$) distribution but using a piecewise-constant majorizing function instead. Notice that in this case, the number of rejections is improved, even with a simple change in efficiency to "squeezing" the majorizing function.

4.3 Animation of NHPP Generation via Thinning

We also provide a function `thinning` to visualize the generation of a non-homogeneous Poisson process using the thinning method. The function provides a default intensity function (meant to mimic a typical 24-hour set of rush-arrival periods), but custom intensity functions can be specified by the user. Similarly, a default constant majorizing function is provided, but custom piecewise-constant majorizing functions can also be specified by the user.

Figure 7 depicts visualization of the thinning process using the default intensity function (green) and default constant majorizing function (red). In this example, an interarrival time is generated as an e_i = Exp(λ^*) variate, resulting in the corresponding (cumulative) arrival time T_i. Then a $u_i = U(0, \lambda^*)$ variate is generated. If u_i is less than the intensity function evaluated at T_i, then T_i is accepted, denoted as an arrival time by a green tick at the horizontal axis with corresponding open green circle at height u_i above. If u_i is greater than the intensity function evaluated at T_i, then T_i is rejected. The corresponding realization of the non-homogeneous Poisson process is given by the left-to-right sequence of ticks above the horizontal axis (which, in general, should be consistent with the rise and fall of the intensity function), and the corresponding time values are returned by the function to the R console.

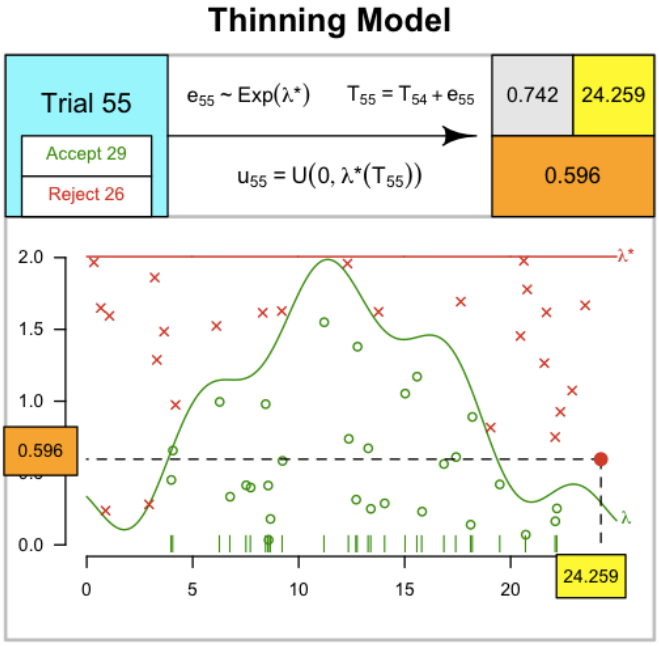


Figure 7: The `thinning` function visualizing the generation of a non-homogeneous Poisson process via the thinning process, based on acceptance-rejection. This example depicts a custom default intensity function (green) and default constant majorizing function (red). Accepted arrival times are indicated by green ticks above the horizontal axis with corresponding open circles above, while rejected times are indicated by corresponding red X's above.

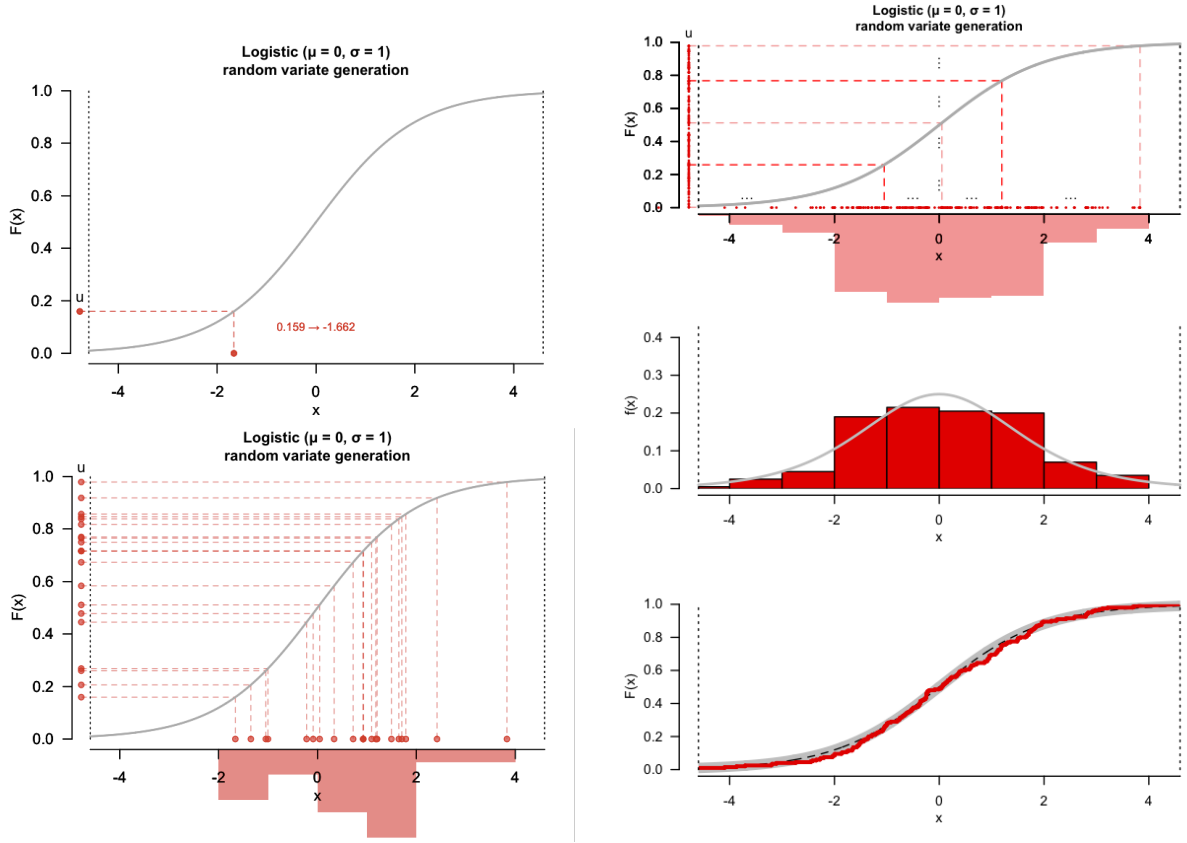
5 VARIATE GENERATION VIA INVERSION

5.1 Animation of Variate Generation via Inversion

In the updated version of `simEd`, we also have eight new functions for visualizing variate generation via inversion (e.g., `ibeta`, `icauchy`, etc. — see Table 3) and have improved the animation of the previous "`i*`" functions.

With our "`i*`" functions, the user can visualize variate generation by inversion using user-specified $U(0,1)$ values or by allowing `runif` or `vunif` to generate variates automatically. The user can also specify which of the following to display: (*i*) inversion across the cdf, (*ii*) corresponding histogram versus population pdf, and (*iii*) corresponding ecdf versus population cdf. The functions also allow the user to choose to interactively proceed step-by-step with each variate generated, in which case the inversion, histogram, and ecdf update immediately with each new variate generated.

Figure 8(a) depicts the `ilogis` function for one and for twenty variates generated, with only the inversion display shown. Note that in the case of one variate only, the value of the $U(0,1)$ variate and the inverted value are both displayed. In the case of multiple variates, an inverted histogram is displayed below the horizontal axis. Figure 8(b) depicts the same `ilogis` function for 200 variates generated, with all three displays shown. Note that in the case of many variates generated, the upper row displays the dashed inversion lines at quantile values only, resulting in a cleaner display. In each case, the function returns to the console all variates generated. Similar results hold for the other discrete and continuous distributions in Table 3.



(a) (b)

Figure 8: (a) The `ilogis` function visualizing the generation of 1 and 20 Logistic($\mu = 0$, $s = 1$) random variates via inversion. (b) The `ilogis` function visualizing the generation of 200 Logistic($\mu = 0$, $s = 1$) random variates via inversion. The middle row depicts the resulting histogram versus the population pdf, while the bottom row depicts the resulting ecdf versus the population cdf. In interactive mode, the inversion, histogram, and ecdf update immediately with each new variate generated.

5.2 Variate Generation via Inversion

Consistent with the "`i*`" functions, in the updated `simEd` package we have also introduced eight new "`v*`" functions for directly generating random variates. As in our previous work, we have chosen naming and parameter conventions similar to the variate generation functions available by default in R via the `stats` package, but replacing the leading 'r' in each function name with 'v' (for variate) instead. Each of the functions generates n $U(0,1)$ random variates, and then inverts using the appropriate quantile functions available in `stats`. Each of the functions also provides the capability for independent streams of random

numbers, based on the `rstream` package implementation (Leydold 2020), and for antithetic variates. Details are discussed in our previous `simEd` work (Lawson and Leemis 2017a).

6 CONCLUSIONS

In this paper, we have discussed significant enhancements to our `simEd` package for R, having a focus on simulation pedagogy. These enhancements focus primarily on, but are not limited to, animation and visualization for teaching simulation. We have included a function for animating and visualizing the details of an event-driven implementation of an $M/M/1$ queue, and have updated existing queueing functions to include more traditional-style queuing animations. We have included animation functions for visualizing, and experimenting with the parameters of, a Lehmer random number generator; for visualizing variate generation via acceptance-rejection; and for visualizing generation of a non-homogeneous Poisson process using thinning. In addition, we have expanded the set of, and improved the animation of, various "`i*`" functions for visualizing variate generation via inversion; and correspondingly have expanded the set of "`v*`" functions for generating variates, with streams and antithetic variates capabilities. These additions significantly enhance the `simEd` package for use in an introductory simulation course.

A ASSOCIATED FUNCTIONS IN THE `simEd` PACKAGE

The following tables provide overviews of function signatures of each of the functions associated with animations presented earlier.

Table 1: Queueing simulation functions with animation. The `ssqvis` function is new, and animates details of an event-driven implementation. The `ssq` and `msq` functions have been enhanced to include animation of the queue and "skyline" functions.

Queue Model	Function Signature

Event-Driven Details for Single-Server	`ssqvis(maxArrivals = Inf, seed = NA,`
	`  interarrivalFcn = NA, serviceFcn = NA,`
	`  arrivalType = "M", serviceType = "M",`
	`  maxTime = Inf, maxDepartures = Inf,`
	`  maxInQueue = Inf, maxInSkyline = 15,`
	`  showSkylineQueue = TRUE, showSkylineSystem = TRUE,`
	`  showSkylineServer = TRUE, showSkyline = NULL,`
	`  customerImage = NA, plotDelay = -1)`

Single-Server	`ssq(maxArrivals = Inf, seed = NA,`
	`  interarrivalFcn = NA, serviceFcn = NA,`
	`  arrivalType = "M", serviceType = "M",`
	`  maxTime = Inf, maxDepartures = Inf, ...,`
	`  showSkylineQueue = TRUE, showSkylineSystem = TRUE,`
	`  showSkylineServer = TRUE, showSkyline = NULL,`
	`  customerImage = NA, plotDelay = 0, respectLayout = FALSE)`

Multiple-Server	`msq(maxArrivals = Inf, seed = NA, numServers = 2,` `  serverSelection = c("LRU", "LFU", "CYC", "RAN", "ORD"), ...)`

REFERENCES

de Lima, P. N. 2018. "Arena2R: Discrete-Event Simulation for R". https://cran.r-project.org/package=arena2r/.

Lawson, B., and L. Leemis. 2015. "Discrete-Event Simulation Using R". In *Proceedings of the 2015 Winter Simulation Conference*, edited by L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, 3502–3513. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Lawson, B., and L. Leemis. 2017a. "An R Package for Simulation Education". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. DAmbrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 1–12. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc. Article No. 346.

Lawson, B., and L. Leemis. 2017b. "simEd: Simulation Education". https://cran.r-project.org/package=simEd, version 1.0.3.

Leydold, J. 2020. "rstream: Streams of Random Numbers". https://cran.r-project.org/package=rstream.

The R Foundation 2017. "The Comprehensive R Archive Network". https://cran.r-project.org/.

Ucar, I., and B. Smeets. 2020. "simmer: Discrete-Event Simulation for R". http://r-simmer.org/.

AUTHOR BIOGRAPHIES

VADIM KUDLAY is currently a third-year undergraduate student majoring in Computer Science and in Mathematics at the University of Richmond. His email address is vadim.kudlay@richmond.edu. Examples of his project work are available on his GitHub site at https://github.com/VKudlay.

BARRY LAWSON is Professor of Computer Science at the University of Richmond. He received Ph.D. and M.S. degrees in Computer Science from William & Mary, and a B.S. in Mathematics from UVA's College at Wise. His interests are in agent-based simulation with biological applications, and in simulation education. His email address is blawson@richmond.edu.

LAWRENCE M. LEEMIS is Professor in the Department of Mathematics at The College of William & Mary. He received B.S. and M.S. degrees in Mathematics and a Ph.D. in Industrial Engineering from Purdue University. His interests are in reliability, simulation, and computational probability. He is a member of ASA and INFORMS. His email address is leemis@math.wm.edu.

Table 2: Other animation utilities in the `simEd` package.

Utility	Signature
Lehmer Generator	`lehmer(a = 13, m = 31, seed = 1, plotDelay = -1)`
Acceptance-Rejection	`arsample(n = 10, pdf = function(x) dbeta(x, 3, 2), majorizingFcn = NULL, support = c(0,1), seed = NA, plot = TRUE, plotDelay = -1)`
Thinning	`thinning(maxTime = 1440, majorFcn = NULL, intensityFcn = defaultCustomerProcess, seed = NA, plot = TRUE, plotDelay = -1)`

Table 3: Functions for visualizing inversion of variate generation. (The last 10 parameters are consistent in all functions, indicated by ellipses.) A null value for `u` displays distribution plots only, without inversion. Here, there are 8 new functions in the `simEd` package, and the animation in all 15 has been improved.

Distribution	Function Signature
Beta	`ibeta(u = runif(1), shape1, shape2, ncp = 0, minPlotQuantile = 0.01, maxPlotQuantile = 0.99, plot = TRUE, showCDF = TRUE, showPDF = FALSE, showECDF = FALSE, show = NULL, maxInvPlotted = 50, plotDelay = 0, respectLayout = FALSE)`
Binomial	`ibinom(u = runif(1), size, prob, ... )`
Cauchy	`icauchy(u = runif(1), location = 0, scale = 1, ... )`
Chisquare	`ichisq(u = runif(1), df, ncp = 0, ... )`
Exponential	`iexp(u = runif(1), rate = 1, ... )`
Gamma	`igamma(u = runif(1), shape, rate = 1, scale = 1/rate, ... )`
Geometric	`igeom(u = runif(1), prob, ... )`
Lognormal	`ilnorm(u = runif(1), meanlog = 0, sdlog = 1, ... )`
Logistic	`ilogis(u = runif(1), location = 0, scale = 1, ... )`
Negative Binomial	`inbinom(u = runif(1), size, prob, mu, ... )`
Normal	`inorm(u = runif(1), mean = 0, sd = 1, ... )`
Poisson	`ipois(u = runif(1), lambda, ... )`
Student's t	`it(u = runif(1), df, ncp = 0, ... )`
Uniform	`iunif(u = runif(1), min = 0, max = 1, ... )`
Weibull	`iweibull(u = runif(1), shape, scale = 1, ... )`

Table 4: Variate generation functions available in `simEd`. Associated with new package functionality, there are 8 new variate generation functions in the package, shown above the dividing line below. All these generating functions use inversion and provide capabilities for streams and for antithetic variates.

Distribution	Function Signature
Beta	`vbeta(n, shape1, shape2, ncp = 0, stream = NULL, antithetic = FALSE)`
Cauchy	`vcauchy(n, location = 0, scale = 1, stream = NULL, antithetic = FALSE)`
Chisquare	`vchisq(n, df, ncp = 0, stream = NULL, antithetic = FALSE)`
Lognormal	`vlnorm(n, meanlog = 0, sdlog = 1, stream = NULL, antithetic = FALSE)`
Logistic	`vlogis(n, location = 0, scale = 1, stream = NULL, antithetic = FALSE)`
Negative Binomial	`vnbinom(n, size, prob, mu, stream = NULL, antithetic = FALSE)`
Poisson	`vpois(n, lambda, stream = NULL, antithetic = FALSE)`
Student's t	`vt(n, df, ncp = 0, stream = NULL, antithetic = FALSE)`
Binomial	`vbinom(n, size, prob, stream = NULL, antithetic = FALSE)`
Geometric	`vgeom(n, prob, stream = NULL, antithetic = FALSE)`
Exponential	`vexp(n, rate = 1, stream = NULL, antithetic = FALSE)`
Gamma	`vgamma(n, shape, rate = 1, scale = 1/rate, stream = NULL, antithetic = FALSE)`
Normal	`vnorm(n, mean = 0, sd = 1, stream = NULL, antithetic = FALSE)`
Uniform	`vunif(n, min = 0, max = 1, stream = NULL, antithetic = FALSE)`
Weibull	`vweibull(n, shape, scale = 1, stream = NULL, antithetic = FALSE)`