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
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 intended 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). According to the `dlstats` package (Yu 2019) available on the Comprehensive R Archive Network (CRAN), as of July 2020 the `simEd` package has more than 12,500 downloads since May 2017, with 4400+ downloads in 2019 and more than 4900 in the first half of 2020. We, the authors, have regularly 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 (which, collectively, would account for a few hundred downloads). It is in the context of using those primarily text-based utilities in teaching, while simultaneously discussing concepts using hand-sketched 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 animation and visualization capabilities for the `simEd` package, written with a focus on simulation pedagogy. While the animation and visualization *techniques* used are not in themselves novel, our contribution lies in providing a single educational software package in R that is significantly enhanced by the new capabilities. When designing, we had two primary considerations:

- To have the animations and visualizations entirely contained within R, even if other external software might provide more modern-looking interfaces. From an ease-of-use viewpoint across multiple platforms, the user will therefore need to install and interact with only the one R package (`simEd`.)

- To allow the user to interact directly with the animations, using a simple function-call format consistent with the other functions provided in the `simEd` package.

The updated package is available on the CRAN (The R Foundation 2017) and includes:

- an interactive animation 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 also include animation (see Table 1);
- an interactive function for visualizing $U(0,1)$ variate generation using a (dated, but pedagogically meaningful) Lehmer random number generator (see Table 1);
- interactive functions for visualizing variate generation via acceptance-rejection and for visualizing generation of a non-homogeneous Poisson process via thinning (see Table 1); and
- additional and improved functions for visualizing inversion for variate generation of various discrete and continuous distributions (see Table 2).

As before, our `simEd` package 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). Additionally, our work does not focus on advancing the state-of-the-art in visualization techniques for simulation. Rather, we focus on developing tools in R for teaching general concepts in Monte Carlo and discrete-event simulation.

In providing easy-to-use but extensible functions in R via the `simEd` package, our aim is consistent with advice provided by some of the well-known experts on a simulation-education panel (Altiok et al. 2001): that of advocating for "generic courses in stochastic modeling and analysis" (B. Nelson) and that "using a standard language has important advantages" (P. L'Ecuyer). In an even more recent panel (Smith et al. 2017), experts indicate that event-oriented simulation in a high-level language "enhanced students' understanding of the discrete-event simulation mechanism" (J. Smith) and that "intuition comes first, and mathematics (or whatever background is needed) comes second" (S. Henderson). Indeed, our goal is to provide easy-to-use functions that help early with student intuition, yet can be extended with a bit of knowledge of programming in R.

Other work provides guidance that is consistent with our goal. Kashefi et al. (2018) suggest that more attention should be paid on understanding system behavior rather than learning how to build simulation models. In a recent survey of simulation educators (de Mesquita et al. 2019), a majority of respondents indicated that probability and statistics should be prerequisites for the simulation discipline, and while computer programming did not receive a similar majority, it still ranked rather highly. One of our primary challenges is that many of those teaching simulation already use commercial packages (e.g., Arena, Simio, AnyLogic), although from the same survey results (Question 10), it is clear that respondents believe "other" software should be included as well. Moreover, of the topics that at least 50% of respondents indicated should be part of the disciplinary syllabus (Question 6), our `simEd` package and its new animations and visualizations can help address the vast majority of those topics in a simulation course.

Much work exists in the scope of simulation education, including recent work on web-based user interfaces for supporting use and development of various kinds of simulation (Wagner 2017); simulation games and gamification, e.g., by Padilla et al. (2016); and storytelling and case studies, e.g., by Padilla et al. (2017). Our work differs in that the goal is not to build specific models, but to provide utilities that aid in intuition and understanding of underlying simulation, probabilistic, and statistical concepts.

Much of our approach is consistent with recommendations by Naps et al. (2002), who propose a taxonomy of learner engagement with visualization technology (No Viewing, Viewing, Responding, Changing, Constructing, and Presenting), hypothesizing that engagement level, and consequently learning outcomes, increase when moving from "No Viewing" toward "Presenting". The `simEd` package is intended specifically for "Responding" (interacting with the visualization) and "Changing" (interacting by

changing variable parameters), as well as "Constructing" (building/extending to create new versions). Using Naps' taxonomy in the context of computer programming visualization, other recent work considered two different engagement levels involving instructor-provided visualizations: active engagement and passive viewing (Banerjee et al. 2015). For the group actively engaged with visualizations, the authors found significant increase in student behavioral engagement, perception of learning, and cognitive achievement in terms of rate of problem solving. Although controlled assessment experiments using `simEd` are left for future work, these existing works provide motivation and justification for our current work.

2 QUEUEING ANIMATION: DETAILS OF EVENT-DRIVEN SIMULATION

In the updated `simEd` package, we have developed a custom interactive animation highlighting the details of an event-driven implementation of an $M/M/1$ queueing model with default arrival rate $\lambda = 1$ and default service rate $\mu = 10/9$. With the new function `ssqvis`, the user can rely on default exponential interarrival and service functions, or can define and provide their own custom interarrival and service functions to pass to `ssqvis`. The animation is split into three rows:

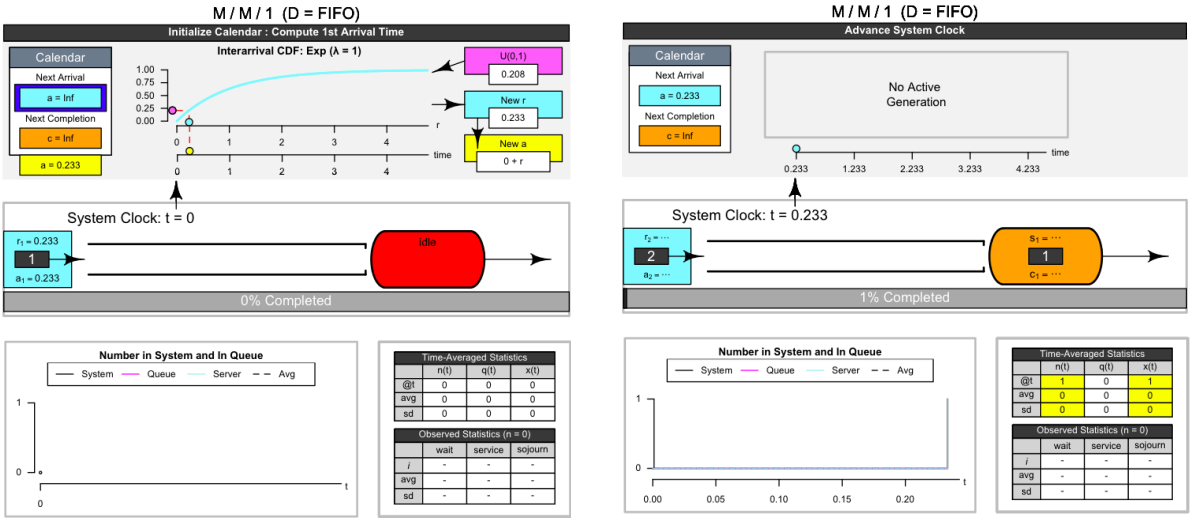
- the top row depicts the event calendar and the generation of arrival and service times using inversion;
- the middle row depicts a fairly typical queue visualization (with customers arriving, queueing, entering service, and departing the system), along with the simulation clock and a progress meter;
- 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 view 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.

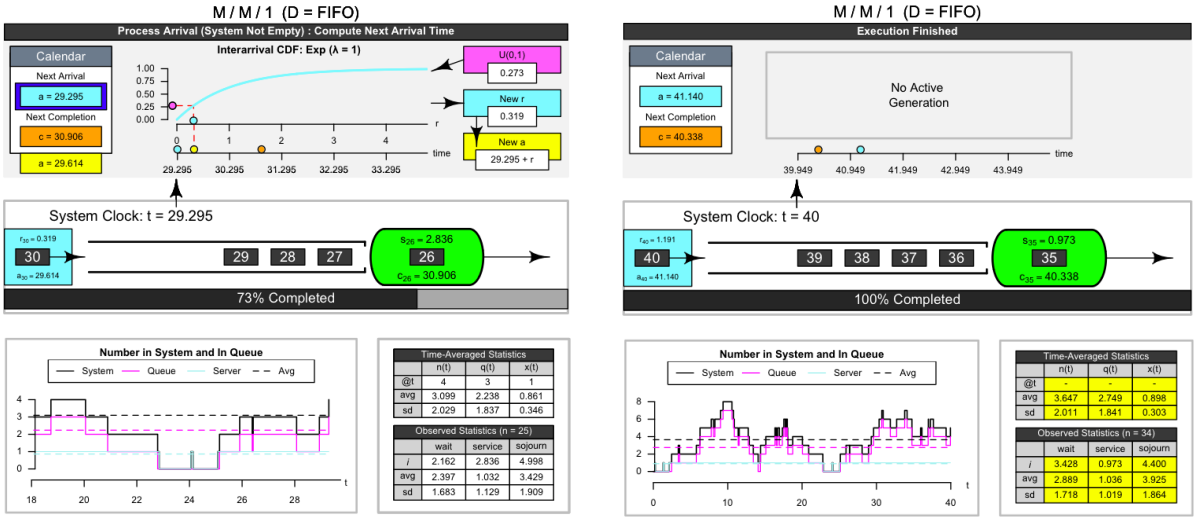
Figure 2(a) depicts the animation after having skipped ahead to the 29th customer. Note that this customer has been queued (middle row), the arrival time for the 30th customer has been generated and is being placed on the calendar (top row), and skyline functions updated appropriately (third row). Finally, Figure 2(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).

Figure 3 depicts a typical R session for generating Figures 1–2. In this figure, user input is denoted in boldface, and demonstrated are the step-by-step default functionality, ability to jump ahead, and optionally to step backward in execution. In the first example run, end-of-run output is disabled, requesting only that the computed average sojourn be displayed at the end; the default value of –1 for `plotDelay` results in interactive mode. In the second example run, `plotDelay` is explicitly set to 0.5 (seconds) so that the animation will automatically update without user input. (This use of the `plotDelay` parameter is consistent across the animation functions in `simEd`: –1 for interactive, 0 to proceed directly to the end, or $s > 0$ to delay s seconds between plot updates.)



(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) 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.

```
> library(simEd)
> ssqvis(maxTime = 40, seed = 8675309, showOutput = FALSE)$avgSojourn
Hit 'ENTER' to proceed, 'q' to end, or 'help' for more: <enter>
Hit 'ENTER' to proceed, 'q' to end, or 'help' for more: <enter>

    ...

Hit 'ENTER' to proceed, 'q' to end, or 'help' for more: help
   'n'/'next'/ENTER  = proceed to next stage
   'j'/'jump'        = jump to the nth initialization
   'b'/'back'        = step back by one plot snapshot
   'q'/'quit'        = change plotDelay to 0
Hit 'ENTER' to proceed, 'q' to end, or 'help' for more: j 30
   Finished jumping
Hit 'ENTER' to proceed, 'q' to end, or 'help' for more: q
[1] 3.0153

> ssqvis(maxTime = 40, seed = 8675309, showOutput = FALSE, plotDelay = 0.5)$avgSojourn
[1] 3.0153
```

Figure 3: Example R code demonstrating `ssqvis` for generating Figures 1–2. User input is in boldface. Note that, in both the interactive and non-interactive runs, the resulting `$avgSojourn` is displayed.

3 QUEUEING ANIMATION: $M/M/1$ AND $M/M/k$

In addition to the new function visualizing event-driven simulation details 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.

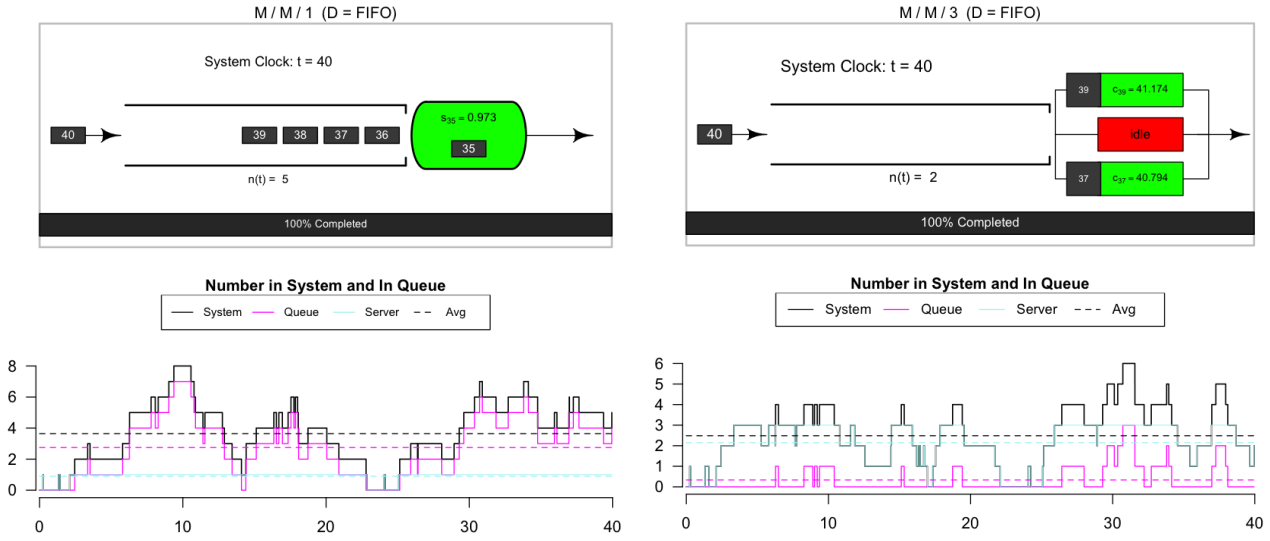
Figure 5 depicts a typical R session for executing the `ssq` and `msq` functions with animation turned on, used to generate Figures 4(a)–(b). As before, user input is shown in boldface. Note that in the `msq` example, a custom *exponential* service function having rate 0.5 is used rather than the default service model.

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 mathematical simplicity a Lehmer (multiplicative linear congruential) random number generator provides an ideal entry point for students to understand 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 students 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



(a) (b)

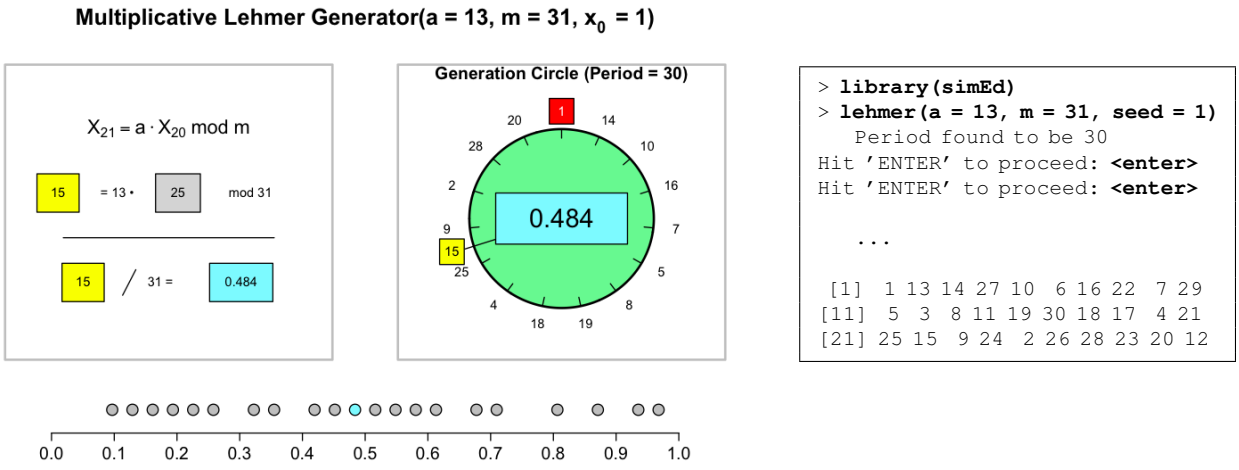
Figure 4: The (a) `ssq` function and (b) `msq` function, each with animation activated, depicting the queueing node and the "skyline" functions of time-persistent statistics for an $M/M/1$ and $M/M/k$ queue respectively. In interactive mode, the node and skyline functions are updated with each arriving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

```
> library(simEd)

> ssq(maxTime = 40, seed = 8675309, animate = TRUE, showOutput = FALSE)$utilization
  |======================================================================| 100%
[1] 0.8980306

> mySvc <- function() vexp(1, rate = 0.5, stream = 2)
> msq(maxTime = 40, seed = 8675309, numServers = 3, serviceFcn = mySvc, animate = TRUE,
+   showOutput = FALSE)$utilization
  |======================================================================| 100%
[1] 0.7076551 0.6988794 0.7420438
```

Figure 5: Example R code demonstrating `ssq` and `msq` for generating Figure 4. User input is in boldface. Note that, for `msq`, a custom *exponential* service function is used rather than the default service model.

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 6(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. Corresponding R code is given in Figure 6(b). By then simply changing the multiplier to $a = 14$, the function can be used to explore less than full periodicity.



(a) (b)

Figure 6: The `lehmer` function depicts, interactively, the generation of a $U(0,1)$ random variate using a multiplicative linear congruential generator. (a) Interactively generating variates using a small prime modulus $m = 31$ with full-period multiplier $a = 13$ and initial seed $x_0 = 1$. (b) Example R code demonstrating `lehmer` for generating Figure 6(a). (Interactive input/output shortened for brevity.)

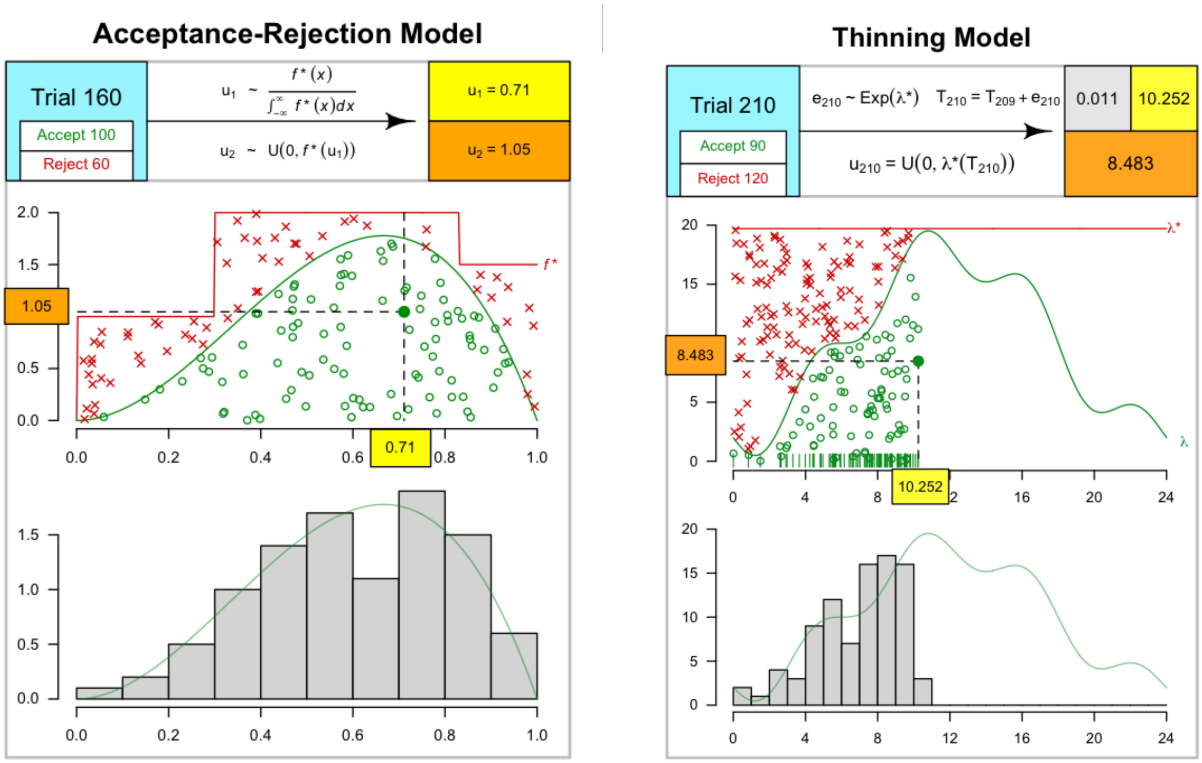
4.2 Animation of Variate Generation via Acceptance-Rejection

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 `accrej` function in `simEd`. With the `accrej` 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 custom piecewise-constant or piecewise-linear majorizing function. The function then interactively demonstrates variate generation via acceptance-rejection.

Figure 7(a) depicts the generation of 100 Beta($\alpha = 3$, $\beta = 2$) variates using acceptance-rejection with a custom piecewise-constant majorizing function $f^*(x)$. At each step, a random variate u_1 is generated (yellow) from the scaled majorizing function to determine the location on the horizontal axis, and then a $u_2 = U(0, f^*(u_1))$ 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 $\times$). Note that the total number accepted and rejected is shown in the top of the display, allowing the instructor to discuss "squeezing" using the majorizing function. Example R code using the `accrej` function is given in Figure 8(a). Note that the custom majorizing function is supplied as an R data frame, with x values corresponding to the endpoints of the segments defining the piecewise-constant function, and y values to the associated heights.

4.3 Animation of NHPP Generation via Thinning

We also provide a function named `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.



(a) (b)

Figure 7: (a) The `accrej` function interactively demonstrates variate generation via acceptance-rejection, here using the default Beta($\alpha = 3$, $\beta = 2$) distribution and a corresponding piecewise-constant majorizing function $f^*(x)$, and requesting 100 samples. (b) The `thinning` function visualizes the generation of a non-homogeneous Poisson process via the thinning process, based on acceptance-rejection. Accepted arrival times are indicated by green ticks above the horizontal axis.

```
> library(simEd)

> majorizing <- data.frame(
+                  x = c(0.0, 0.3, 0.83, 1.0),
+                  y = c(0.0, 1.0, 2.0, 1.5))
> samples <- accrej(n = 100, seed = 8675309,
+    majorizingFcn = majorizing,
+    majorizingFcnType = "pwc",
+    plotDelay = 0)

> head(samples, n = 4)
[1] 0.06683874 0.25434546 1.58941606 0.31508686
```

(a)

```
> library(simEd)

> arrivals <- thinning(seed = 8675309)
Hit 'ENTER' to proceed and 'q' to end: help
   'n'/'next'/ENTER  = proceed to next stage
   'j'/'jump' = jump to the nth initialization
   'b'/'back' = step back by one plot snapshot
   'q'/'quit' = change plotDelay to 0
Hit 'ENTER' to proceed and 'q' to end: j 90
Hit 'ENTER' to proceed and 'q' to end: q

> head(arrivals, n = 4)
[1] 0.01184767 0.84760644 1.50284590 2.59057128
> tail(arrivals, n = 4)
[1] 22.46144 22.53142 22.70214 23.72117
```

(b)

Figure 8: Example R code demonstrating `accrej` and `thinning` for generating Figures 7(a)–(b). User input is in boldface. (a) The `accrej` example uses the default Beta($\alpha = 3, \beta = 2$) pdf and a custom piecewise-constant majorizing function. (b) The `thinning` example uses the default intensity function and default constant majorizing function. Note the value of 0 for `plotDelay` causes the animation to proceed directly to the end, while the default requires user interaction.

Figure 7(b) 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 NHPP 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. An example of R code using the `thinning` function is given in Figure 8(b), where boldface again denotes user input. This example demonstrates jumping ahead to the 90th acceptance (shown in Figure 7(b)), immediately followed by completing the full realization.

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 2) 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.

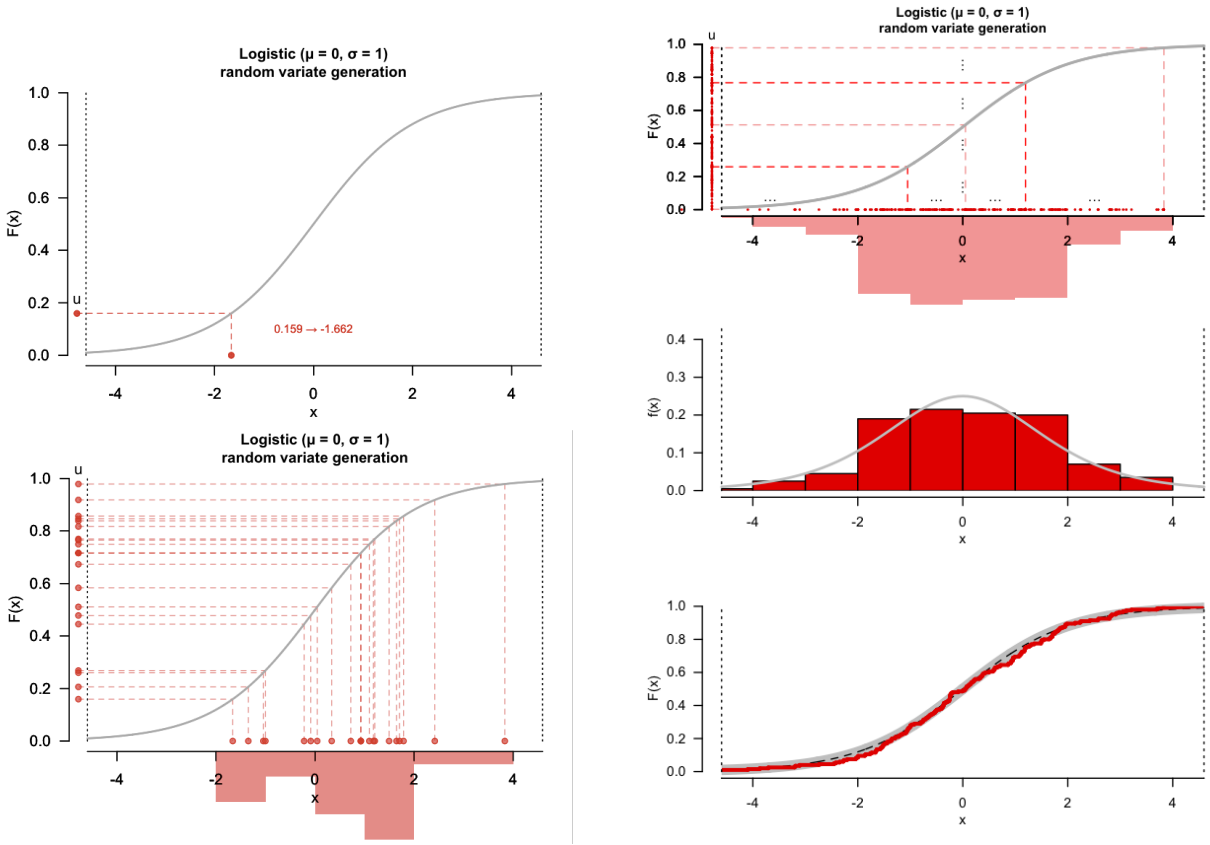
Figure 9(a) depicts the `ilogis` function for one and for twenty logistic variates generated, with only the inversion display shown. Note that in the case of one variate only, the value of the $U(0,1)$ variate and the inverted value are both displayed. In the case of multiple variates, an inverted histogram is displayed below the horizontal axis. Figure 9(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. In each case, the function returns to the console all variates generated. Figure 10 shows R code that produces the inversion graphs in Figure 9. Similar results hold for the other discrete and continuous distributions in Table 2.

5.2 Functions for Variate Generation

Consistent with the "`i*`" functions, in the updated `simEd` package we have also introduced eight new "`v*`" functions for generating random variates via inversion. 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 queueing animations. We have included animation functions for visualizing



(a) (b)

Figure 9: (a) `ilogis` visualizing the generation of 1 and 20 Logistic($\mu = 0$, $s = 1$) random variates via inversion. (b) `ilogis` visualizing the generation of 200 random variates. The middle row depicts the histogram versus the population pdf; the bottom row depicts the ecdf versus the population cdf. In interactive mode, the inversion, histogram, and ecdf update immediately with each new variate generated.

```
> library(simEd)

> sample  <- ilogis(vunif(1, stream = 1), location = 0, scale = 1)
> samples <- ilogis(vunif(20, stream = 1), location = 0, scale = 1, show = 4)
> samples <- ilogis(vunif(200, stream = 1), location = 0, scale = 1, show = 7)
```

Figure 10: R code demonstrating `ilogis`, which visualizes inversion using the logistic distribution, for producing Figures 9(a)–(b). Note the use of `vunif` to allow for random number streams. Also note the `show` parameter, which uses a Unix `chmod` numerical value for choosing which of the inversion (4), pdf (2), and/or ecdf (1) graphs to display.

and experimenting with the parameters of: a Lehmer random number generator; variate generation via acceptance-rejection; and 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 via inversion, with streams and antithetic variates capabilities. These additions significantly enhance the `simEd` package for use in an introductory simulation course.

A ANIMATION AND VISUALIZATION FUNCTIONS IN THE `simEd` PACKAGE

The following tables provide overviews of signatures for each of the functions associated with animations and visualizations present in the `simEd` package.

Table 1: Package functions with animation. The `ssq` and `msq` functions have been updated to include animation of the queue and "skyline" functions. All other functions are new.

Utility	Example Function Call
Event-Driven $M/M/1$ Details	`ssqvis(maxTime = 40, jobImage = "http://bit.ly/prthwarbler")`
$M/M/1$ Model	`ssq(maxTime = 40, animate = TRUE)`
$M/M/k$ Model	`msq(maxTime = 40, numServers = 3, animate = TRUE)`
Lehmer Generator	`lehmer(a = 13, m = 31, seed = 11)`
Acceptance-Rejection	`accrej(n = 100, pdf = function(x) dbeta(x, 3, 2))`
Thinning	`thinning(maxTime = 1440, seed = 8675309)`

Table 2: Functions for variate generation and for visualizing inversion, by distribution. There are 8 new distributions (underlined) in the package. Visualization for all inversion functions has been improved.

Distribution	Example Inversion Visualization	Example Variate Generation
Beta	`ibeta(runif(10), shape1 = 3, shape2 = 2)`	`vbeta(10, shape1 = 3, shape2 = 2)`
Binomial	`ibinom(runif(10), size = 10, prob = 0.3)`	`vbinom(10, size = 10, prob = 0.3)`
Cauchy	`icauchy(runif(10), location = 0, scale = 1)`	`vcauchy(10, location = 0, scale = 1)`
Chisquare	`ichisq(runif(10), df = 7, ncp = 0)`	`vchisq(10, df = 7, ncp = 0)`
Exponential	`iexp(runif(10), rate = 1, show = 7)`	`vexp(10, rate = 1, antithetic = TRUE)`
Gamma	`igamma(runif(10), shape = 4, scale = 2)`	`vgamma(10, shape = 4, scale = 2)`
Geometric	`igeom(runif(10), prob = 0.3)`	`vgeom(10, prob = 0.3)`
Lognormal	`ilnorm(runif(10), meanlog = 0, sdlog = 1)`	`vlnorm(10, meanlog = 0, sdlog = 1)`
Logistic	`ilogis(runif(10), location = 1, scale = 0.5)`	`vlogis(10, location = 0, scale = 0.5)`
Neg. Binomial	`inbinom(runif(10), size = 4, prob = 0.4)`	`vnbinom(10, size = 4, prob = 0.4)`
Normal	`inorm(runif(10), mean = 0, sd = 1)`	`vnorm(10, mean = 0, sd = 1, stream = 1)`
Poisson	`ipois(runif(10), lambda = 0.7)`	`vpois(10, lambda = 0.7, stream = 2)`
Student's t	`it(runif(10), df = 5, ncp = 1, show = 7)`	`vt(10, df = 5, ncp = 1, stream = 3)`
Uniform	`iunif(runif(10), min = 0, max = 1)`	`vunif(10, min = 0, max = 1)`
Weibull	`iweibull(runif(10), shape = 2, scale = 1)`	`vweibull(10, shape = 2, scale = 1)`

REFERENCES

Altiok, T., P. L'Ecuyer, B. W. Schmeiser, L. W. Schruben, W. D. Kelton, B. L. Nelson, T. J. Schriber, and J. R. Wilson. 2001. "Various Ways Academics Teach Simulation: Are They All Appropriate?". In *Proceedings of the 2001 Winter Simulation Conference*, edited by B. Peters, J. Smith, D. Medeiros, and M. Rohrer, 1580–1591. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Banerjee, G., S. Murthy, and S. Iyer. 2015. "Effect of Active Learning Using Program Visualization in Technology-Constrained College Classrooms". *RPTEL* 10(15). https://doi.org/10.1186/s41039-015-0014-0.

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

de Mesquita, M. A., B. C. da Silva, and J. V. Tomotani. 2019. "Simulation Education: A Survey of Faculty and Practitioners". In *Proceedings of the 2019 Winter Simulation Conference*, edited by N. Mustafee, K.-H.G. Bae, S. Lazarova-Molnar, M. Rabe, C.Szabo, P. Haas, and Y.-J. Son, 3344–3355. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Kashefi, A., F. Alwzinani, and D. Bell. 2018. "Perspectives on Teaching Modeling and Simulation in a Department of Computer Science". In *Proceedings of the 2018 Winter Simulation Conference*, edited by M. Rabe, A. Juan, N. Mustafee, A. Skoogh, S. Jain, and B. Johansson, 4058–4068. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

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. D'Ambrogio, 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.

Naps, T. L., G. Rößling, V. Almstrum, W. P. Dann, R. Fleischer, C. D. Hundhausen, A. Korhonen, L. Malmi, M. F. McNally, S. H. Rodger, and J. A. Velázquez-Iturbide. 2002. "Exploring the Role of Visualization and Engagement in Computer Science Education". *ACM SIGCSE Bulletin* 35(2):131–152. https://doi.org/10.1145/782941.782998.

Padilla, J. J., C. J. Lynch, S. Y. Diallo, R. J. Gore, A. Barraco, H. Kavak, and B. Jenkins. 2016. "Using Simulation Games for Teaching and Learning Discrete-Event Simulation". In *Proceedings of the 2016 Winter Simulation Conference*, edited by T. M. K. Roeder, P. I. Frazier, R. Szechtman, E. Zhou, T. Huschka, and S. E. Chick, 3375–3384. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Padilla, J. J., C. J. Lynch, H. Kavak, S. Evett, D. Nelson, C. Carson, and J. del Villar. 2017. "Storytelling and Simulation Creation". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 4288–4299. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Smith, J. S., C. Alexopoulos, S. Henderson, and L. Schruben. 2017. "Teaching Undergraduate Simulation: 4 Questions for 4 Experienced Instructors". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 4264–4275. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

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/.

Wagner, G. 2017. "sim4edu.com: Web-based Simulation for Education". In *Proceedings of the 2017 Winter Simulation Conference*, edited by W. K. V. Chan, A. D'Ambrogio, G. Zacharewicz, N. Mustafee, G. Wainer, and E. Page, 4240–4251. Piscataway, NJ: Institute of Electrical and Electronics Engineers, Inc.

Yu, G. 2019. "dlstats: Download stats of R packages". https://cran.r-project.org/package=dlstats, version 0.1.3.

AUTHOR BIOGRAPHIES

VADIM KUDLAY is currently a fourth-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 STEM education. His email address is blawson@richmond.edu and his website is http://www.mathcs.richmond.edu/∼blawson/.

LAWRENCE M. LEEMIS is Professor in the Department of Mathematics at 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 and his website is http://www.math.wm.edu/∼leemis/.