Proceedings of the 2015 Winter Simulation Conference
L. Yilmaz, W. K. V. Chan, I. Moon, T. M. K. Roeder, C. Macal, and M. D. Rossetti, eds.

THE MNO–PQRS POISSON POINT PROCESS: GENERATING THE NEXT EVENT TIME

Huifen Chen
Department of Industrial and Systems Engineering
Chung-Yuan University
Chung-Li, 320 Taoyuan, TAIWAN

Bruce Schmeiser
School of Industrial Engineering
Purdue University
West Lafayette, IN 47907, USA

ABSTRACT

We discuss the problem of generating the time of the next event of a nonhomogeneous Poisson process with an MNO-PQRS rate function. A PQRS function is piecewise quadratic. At every time point, an MNO-PQRS rate function is the maximum of zero and a piecewise-quadratic function. We take as given the three quadratic coefficients of every time interval. In addition, we take as given the time of the previous event. The problem is then to generate the time of the next event. We review thinning algorithms, but focus on presenting an efficient inverse-transformation algorithm that converts a single pseudorandom number to the next-event time.

1 INTRODUCTION

Rate functions arise in probability models of stochastic systems, often to describe the rate of arrivals to the system. More generally, an *event* can be any change of state that occurs at a point in time. We use the term *time* throughout, even though the parameter of interest might be space. In general a rate function is nonnegative and its integral from time t_1 to time t_2 is the expected number of events in the interval (t_1,t_2). Because the integral at a time point is zero, the probability of an event at any time t is zero; nevertheless, an event might occur at any time t for which the rate function is positive.

We assume that the point process of interest is a nonhomogeneous Poisson process. Therefore, the rate function's integral between two adjacent events is exponential with a mean of one. Further, these integrals are independent of each other.

We further assume the particular form of Poisson process rate function, MNO–PQRS, developed in Chen and Schmeiser (2014, 2015) and reviewed in Section 2. Their purpose in creating the MNO–PQRS rate function, which we refer to as τ, is to replace a specified piecewise-constant rate function with something smoother subject to the *mean constraint*; that is, maintaining the fundamental property that the expected number of events is unchanged for every interval. In particular, PQRS yields a piecewise-quadratic fit that might sometimes be negative and the MNO–PQRS rate function is the maximum of zero and a modified PQRS fit.

Figure 1 is an example based on 24 hours of traffic-count data from New York State Department of Transportation (2015) for Tuesday, August 30, 2011, Station 118781, Direction 7, Lane 1. The counts are for ninety-six 15-minute intervals from midnight to midnight. Superimposed over the piecewise-constant rate function from the count data is the corresponding MNO–PQRS rate function. Middle-of-the-night counts are low, so the resulting PQRS fit is sometimes negative; therefore the MNO–PQRS rate function has some zero rates for parts of some time intervals.

Because the count data fluctuate, some smoothing maybe would be reasonable; for example, pairs of intervals could be combined to create 48 half-hour intervals. Any such preliminary data analysis, though, is application dependent. The piecewise-quadratic smoothing of MNO–PQRS, which maintains the mean constraint of each time interval, simply smooths while mimicking the given count-data fluctuations. In

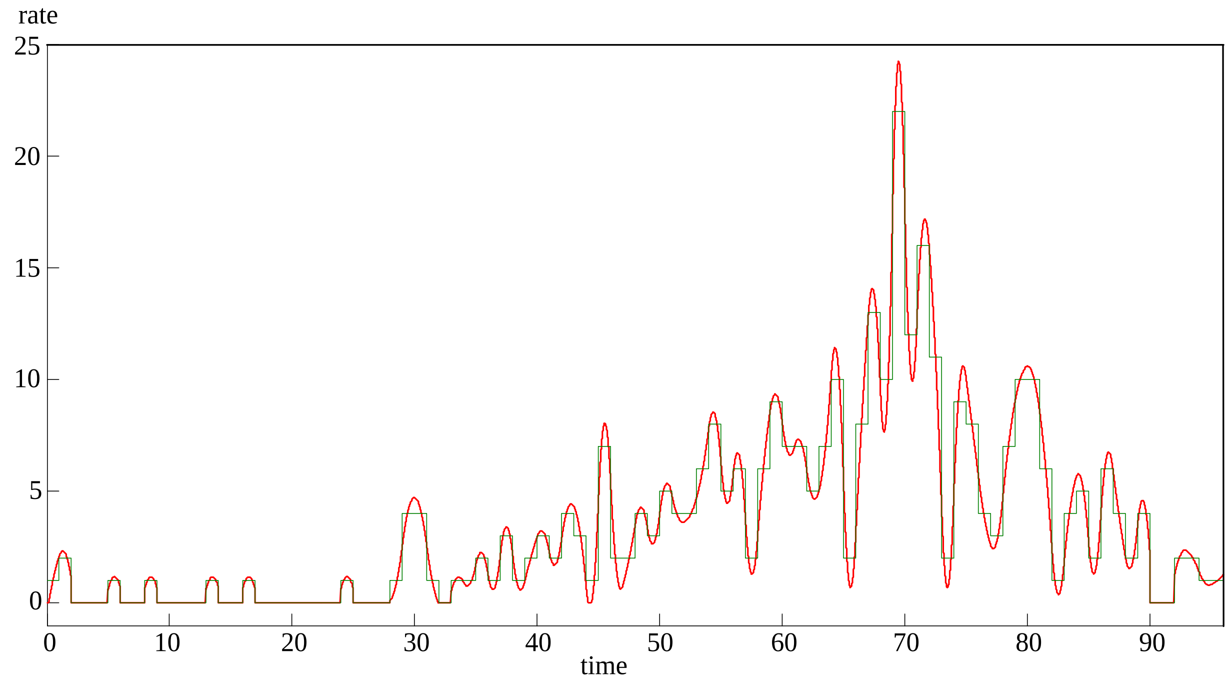

Figure 1: New York State Department of Transportation traffic-count data and the corresponding MNO–PQRS rate function.

addition, MNO–PQRS assumes that all time intervals are length one and starting time is zero; the user needs to make whatever time transformation is necessary, in this case time $t = 0$ being midnight, time $t = 48$ being noon, and $t = 96$ being the next midnight.

Our purpose here is to develop efficient and exact process-generation logic for MNO–PQRS rate functions. In Section 3 we develop inverse-transformation logic that converts a sequence of independent uniform (0,1) random numbers into a sequence of event times $t_0, t_1, t_2, \ldots$ from the Poisson process with specified MNO–PQRS rate function.

The rest of this paper is organized as follows. Section 2 reviews MNO–PQRS rate functions. Section 3 discusses process generation for Poisson processes in general and explains why we don't pursue thinning. Section 4 contains our inverse-transformation logic for generating the next event time for MNO–PQRS rate functions. Section 5 is a discussion.

2 THE MNO–PQRS RATE FUNCTION

The PQRS (Piecewise-Quadratic Rate Smoothing) function is defined on the interval $[0,k]$, partitioned into intervals $(i-1,i]$, for $i = 1,2,\ldots,k$. Each interval's rate function is a quadratic function.

For times x in the unit interval $(0,1]$, define the quadratic function to be

$$q(x : a,b,c) = ax^2 + bx + c.$$

The PQRS function is then

$$\tau_0(t : \underline{a}, \underline{b}, \underline{c}) = q(x : a_i, b_i, c_i),$$

where $j = \lfloor t/k \rfloor$ is the number of completed cycles, $i = \max\{1, \lceil t - jk \rceil\}$ is the interval number, and $x = t - jk - i + 1$ is the fractional time within interval i (except that the time intervals are closed on the right, so $x = 1$ when t is integer). The function τ_0 then has $3k$ parameters: $\underline{a} = (a_1, a_2, \ldots, a_k)$, $\underline{b} = (b_1, b_2, \ldots, b_k)$, and $\underline{c} = (c_1, c_2, \ldots, c_k)$.

The PQRS logic in Chen and Schmeiser (2014) uses $O(k^2)$ computation and $O(k)$ storage to choose the parameters $(\underline{a},\underline{b},\underline{c})$ to make τ continuous, to make its first derivatives continuous, and to maintain a specified expected number of events for each interval. If $\tau_0(t:\underline{a},\underline{b},\underline{c})$ is nonnegative for all $t \in [0,k]$, $\tau = \tau_0$ is returned as the smoothed rate function.

If, however, $\tau_0(t:\underline{a},\underline{b},\underline{c})$ is not nonnegative everywhere, the PQRS function is replaced with the maximum of zero and a modified piecewise-quadratic function. That is, we replace τ_0 with

$$\tau^+(t:\underline{a}^+,\underline{b}^+,\underline{c}^+) = \max\{0, q(x:a_i^+,b_i^+,c_i^+)\},$$

where j, i, and x are defined above. The parameters $(\underline{a}^+,\underline{b}^+,\underline{c}^+)$ are computed using MNO (max nonnegativity ordering) logic, which maintains the expected number of events and the other conditions to the extent possible. Computation and storage are $O(k)$. In this case, $\tau = \tau^+$ is the MNO–PQRS rate function.

Chen and Schmeiser (2014) consider only the *cyclic* context, where for times $t > k$ the rate is $\tau(t - jk)$, where j is the number of completed cycles, as defined above. Chen and Schmeiser (2015) generalize to consider also four *finite-horizon* contexts, for which always $t \le k$. For process generation, the only difference is that in the finite-horizon context no event is generated after time k.

3 POISSON-PROCESS GENERATION

Although nothing about the rate function τ is specific to Poisson processes, they are the motivating application of MNO–PQRS. We discuss here the problem of generating events from an MNO–PQRS Poisson process.

Generation algorithms are available for other point processes. For nonhomogeneous non-Poisson processes with a specified rate function and an asymptotic mean-to-variance (dispersion) ratio, Gerhardt and Nelson (2009) propose an inverse-transformation method; Liu (2013) proposes a combination of inverse-transformation and thinning methods. Saltzman et al. (2012) discuss multivariate nonhomogeneous Poisson processes.

Our approach is to assume that the previous event time, say t_p, is known and that the problem is to generate the next event time, say t. In addition to t_p, we are given the MNO–PQRS rate function τ and an infinite sequence of independent uniform (0,1) random numbers, $u_1, u_2, \ldots$. Two classic approaches (e.g., Schmeiser 1980) are available: thinning and the inverse transformation.

3.1 Thinning

Thinning algorithms (e.g., Lewis and Shedler 1979b) from the rate function τ are easy to implement. The design issue is the choice of a majorizing rate function $\overline{\tau}$ that satisfies $\overline{\tau}(t) \ge \tau(t)$ for every time t. Algorithm efficiency depends upon two characteristics of $\overline{\tau}$: (1) keeping the integral of $\overline{\tau}$ not much larger than the integral of τ and (2) being able to easily generate event times from $\overline{\tau}$. A simple choice would be the constant rate function $\overline{\tau}(t) = \max\{\tau(t)\}$. The thinning algorithm, given the previous event time t_0, generates the next event time t_1 from $\overline{\tau}$. The time t_1 is returned as the next event time, T, from τ with probability $\tau(t_1)/\overline{\tau}(t_1)$; otherwise the logic is repeated with t_0 replaced by t_1.

A disadvantage of thinning algorithms is that, to generate the next event time, they use a random number of random numbers. Variance-reduction ideas such as common random numbers then do not work well. More fundamental, however, is that the expected number of random numbers, can be arbitrarily large. Choosing a more-efficient majorizing function, such as the piecewise-constant function that is the maximum of each interval's rates, is tempting. But still the expected number of random numbers can be arbitrarily large. Still more-efficient majorizing functions could be chosen, such as using multiple piecewise-linear rates (analogous to Nicol and Leemis 2014a and 2014b). Despite the growing complication of fitting the more-efficient majorizing function, for a fixed number of linear pieces the expected number of random numbers remains unbounded over the set of all MNO–PQRS rate functions (although for any given MNO-PQRS rate function the expected number of random numbers is finite).

3.2 Inverse Transformation

In contrast, inverse-transformation algorithms generate the next event time using exactly one random number. Klein and Roberts (1984) discuss the Poisson process inverse transformation with piecewise-linear rate functions. Chen and Schmeiser (1992) do the same for trigonometric rate functions. Lewis and Shedler (1976, 1979a) do the same for two specific rate function forms. The inverse transformation is based on the general result that the area $\int_{t_0}^{T} \tau(t)dt$ between adjacent Poisson event times has an exponential distribution with mean one, independent of previous event times. Therefore, the inverse transformation algorithms generate an exponential random variate with mean one, typically with $Y = -\log(1-U)$ where $U \equiv U(0,1)$. They then do whatever is necessary to solve for the next event time T such that $\int_{t_0}^{T} \tau(t)dt = Y$. In our application, the integral is a cubic equation, which can be solved either analytically (Abramowitz and Stegun, 1972) or with Newton search. The piecewise bookkeeping is similar to that used in Klein and Roberts (1984).

4 MNO–PQRS INVERSE TRANSFORMATION

Our inverse-transformation logic is a bit more general than for MNO-PQRS Poisson process rate functions. In particular, we don't require that the rate function is continuous or differentiable at the intervals' end points.

We discuss our logic in five parts. Section 4.1 is about partitioning each interval into segments with zero rates and positive rates. Section 4.2 is about integrating the positive-rate interval segments. Section 4.3 is about numerically inverting the cubic cumulative rate function. Section 4.4 contains an example driver program. Finally Section 4.5 contains the routine that generates the next event time.

Unless mentioned separately, in the pseudocodes all variables can be thought of as double-precision variables. Alternatively, variables that begin with the letters i, j, k, l, m, and n can be thought of integers.

4.1 Partitioning

Fundamental to dealing with the MNO–PQRS rate function for interval i is the need to know where the τ is zero and where it is positive. Our solution is the routine *qpartit*, which partitions the unit interval with three points, $0 \le \alpha \le \beta \le \gamma \le 1$. As before, let x denote the fractional part of t. The values of (α, β, γ) are chosen so that $q_i(x : a_i^+, b_i^+, c_i^+)$ is positive in the intervals (α, β) and $(\gamma, 1)$ and zero in the intervals $(0, \alpha)$ and (β, γ). The routine *qpartit* is also needed for the MNO fitting logic.

```
provided: (a, b, c), the quadratic coefficients
compute:  (alpha, beta, gamma), the segment boundaries

routine qpartit:

  alpha = 0
  beta  = 0
  gamma = 1
  if (a = 0) then
    if (b = 0) then
      if (c > 0) beta = 1
    else
      root = - c / b
      if (b < 0) then
        beta  = min( 1, root )
      else
        gamma = max( 0, root )
      endif
    endif
  else
    char = b^2 - 4*a*c
    if (char < 0) then
      if (a > 0) beta = 1
    else
      schar = sqrt( char )
      root1 = (- b - schar) / (2*a)
      root2 = (- b + schar) / (2*a)
      beta  = max( 0, min( 1, root1 ) )
      if (a < 0) then
        alpha = max( 0, min( 1, root2 ) )
      else
        gamma = max( 0, min( 1, root2 ) )
      endif
    endif
  endif
  return alpha, beta, gamma
```

4.2 Integration

Each interval has zero, one, or two subintervals of positive rates. We refer to the integral of (α,β) as the left area and the integral of $(\gamma,1)$ as the right area. Either or both can be zero. Their sum is the interval's mean. Any positive area between the lower bound and upper bound is computed with the routine *qarea*

```
provided: (a, b, c), the quadratic coefficients
provided: (boundl, boundu), the lower and upper bounds
compute:  qarea, the area between the bounds

routine qarea:

  if (boundu <= boundl) then
    qarea = 0
  else
    qarea =   (a/3) * (boundu^3 - boundl^3)
            + (b/2) * (boundu^2 - boundl^2)
            +  c    * (boundu   - boundl )
  endif
  return qarea
```

4.3 Cubic Inversion

The intervals with positive quadratic rates have associated cubic cumulative rates. For a specified interval with quadratic coefficients (a,b,c) and a specified area v, the routine *cubic1* contains inversion logic to compute the time x so that $\int_0^x q(y:a,b,c)dy = v$. The logic assumes that a lower bound and upper bound are known; these values are easily available from α, β, and γ. Given these bounds, the root x is unique.

There are four cases for the cumulative rate function: cubic ($a \neq 0$), quadratic ($a = 0, b \neq 0$), linear ($a = b = 0, c \neq 0$), and zero ($a = b = c = 0$.) The last three are trivial, but included here for completeness. The cubic case has multiple analytical solutions; a classic reference is Abramowitz and Stegun (1972). Because in the general cubic case there are three roots, the logic becomes complicated. We use Newton's method with convergence guaranteed with bisection search (Conte and deBoor 1980), which is simpler to implement in our situation, because we know that there is a single root between the bounds.

```
provided: (a, b, c), the quadratic coefficients
provided: v, the random area
provided: (boundl, boundu), bounds on the value of x
compute: x, the fractional time

routine cubic1:

  tolerance = 10^(-12)
  if (a != 0) then
    comment: rewrite as f(t) = t^3 + a2*t^2 + a1*t + a0 = 0.
    a2 =  3*b / (2*a)
    a1 =  3*c / a
    a0 = -3*v / a
    comment: newton's method
    f = 2 * tolerance
    x = (boundl + boundu) / 2
    do while (abs( f ) > tolerance)
      f = x * (x *(x + a2) + a1) + a0
      d = x * (3 * x + 2 * a2) + a1
      xnewton = x - (f / d)
      if (xnewton < boundl) then
        boundu = x
        x = (x + boundl) / 2
      elseif (xnewton > boundu) then
        boundl = x
        x = (x + boundu) / 2
      else
        x = xnewton
      endif
    enddo
  elseif (b != 0) then
     cdb = c / b
     x   = -cdb + sqrt( cdb^2 + (2*v / b) )
  elseif (c != 0) then
     x   =  v / c
  else
     x   = 1
  endif
  return x
```

4.4 Driver Program

An illustrative driver program inputs the number of intervals k, the k-dimensional MNO-PQRS rate vectors a, b, and c. In addition, its random-number generator is initialized with a seed value and the number of replications is specified. The first event time is set to $t = 0$. The random-generation routine *poisrvg* is called repeatedly, each time returning the next event time t. This driver program is replaced with user-written logic for the user's application.

```
driver program:
```

```
print: Each interval is length one, starting at zero.
print: Enter number of intervals.
input: k
comment: input 1xk vectors a, b, c
do i=1,k
  print: Enter (a, b, c) for interval i.
  input:  a(i), b(i), c(i)
enddo
print: Enter random-number seed and number of event times.
input:  seed, nreps
t = 0
do irep=1,nreps
  call poisrvg( k, a, b, c, seed, t )
  print: t
enddo
stop
```

4.5 Generation

We are now ready to discuss the generation routine *poisrvg*, which is called from some higher-level program within a simulation. The routine *poisrvg* requires *qpartit*, *qarea*, and *cubic*1, all discussed above. In addition, we require a uniform(0,1) random-number generator, which we refer to as *rngenerator*; in our implementation we used the generator of L'Ecuyer (1999).

As with all inversion-transformation logic, for a single random number u, the cumulative distribution function (cdf) $P(T \leq t) = u$ is inverted to find the value of t. For any Poisson process, the inversion determines the time t that has an exponential (with mean one) area Y between it and the previous event time, t_p.

To avoid unnecessary re-computation, the logical variable *isaved(i)* is initially *false* for every interval i. Whenever an intervals left and right areas have been computed, they are saved and *isaved(i)* is set to *true*. This strategy allows the code to be used for multiple rate functions, since nothing is saved between calls to *poisrvg*. If only one rate function is being used, the initialization of the *isaved* vector can be omitted.

The logic begins by generating the exponential area y. Then the number of completed cycles j , the interval number, and the fractional time x of t_p is computed; if the left and right areas, and the bounds of positive rates, of this interval are unknown, they are computed. Always, then, the areas to the left and right of x are computed, since the area to the left is in the past. Computing the next event time, t, proceeds interval by interval, subtracting the interval's mean from y and incrementing j whenever another cycle is completed, until the interval i containing t is found. Before calling *cubic1*, the area to the left of x is added to y. The routine *cubic1* then returns the fractional time of t, from which the next event time $t = (j*k)+(i-1)+x$ is computed.

```
required routines: qarea, qpartit, cubic1, nrgenerator
provided: k, number of time intervals
provided: (a,b,c), 1xk vectors of quadratic coefficients
provided: seed, random-number seed
provided: t, the previous event time
compute: seed, updated random-number seed
compute: t, the next event time

comment: intermediate variables...
comment: j, the number of completed cycles (j=0,1,2,...)
```

```
comment: i, the interval in which t lies (i=1,2,...,k)
comment: x, the fractional time of t in interval i
comment: area(i), the area of interval i
comment: area1(i), the left-most positive area of interval i
comment: start1(i), the left bound of left-most positive area
comment: end1(i), the right bound of left-most positive area
comment: start2(i), the left bound of right-most positive area
comment: isaved(i), "true" indicates interval-i constants computed

routine poisrvg:

  define 1xk arrays: a, b, c, area, area1, start1, end1, start2
  define 1xk logical array: isaved

  comment: set isaved to indicate that no interval is initialized.
  do i=1,k
    isaved(i) = "false"
  enddo
  comment: generate a mean-one exponential random variate, y
  u = rngenerator( seed )
  y = - log( 1-u )
  comment: for previous time t, compute the location in the cycle
  j = t / k
  i = max( 1, ceiling( t - j*k ) )
  x = t - j*k - (i - 1)
  comment: compute constants for interval i (if first time)
  if (isaved(i) = "false") then
    call qpartit( a(i), b(i), c(i), start1(i), end1(i), start2(i) )
    area1(i) =       qarea( a(i), b(i), c(i), start1(i), end1(i) )
    area(i)  = area1(i) + qarea( a(i), b(i), c(i), start2(i), 1 )
    isaved(i) = "true"
  endif
  comment: compute the areas to the left and right of x.
  if (x < start2(i)) then
    sleft =           qarea( a(i), b(i), c(i), start1(i), x )
  else
    sleft = area1(i) + qarea( a(i), b(i), c(i), start2(i), x )
  endif
  sright = area(i) - sleft
  comment: compute the next time, t
  if (y <= sright) then
    comment: in the same interval
    y = sleft + y
  else
    comment: in a future interval
    sarea = sright
    do while (y > sarea)
      y = y - sarea
      i = i + 1
```

```
            if (i > k) then
              j = j + 1
              i = 1
            endif
            if (isaved(i) = "false") then
              call qpartit( a(i),b(i),c(i), start1(i),end1(i),start2(i) )
              area1(i) = qarea( a(i), b(i), c(i), start1(i), end1(i) )
              area(i)  = area1(i) + qarea( a(i),b(i),c(i), start2(i),1 )
              isaved(i) = "true"
            endif
            sarea = area(i)
          enddo
        endif
        comment: to ignore negative rates, add back negative area
        if (y < area1(i)) then
          y = y +           qarea( a(i), b(i), c(i), 0, start1(i) )
          boundl = start1(i)
          boundu = end1(i)
        else
          y = y - area1(i) + qarea( a(i), b(i), c(i), 0, start2(i) )
          boundl = start2(i)
          boundu = 1
        endif
        comment: find the fractional time x and return the next time, t
        call cubic1( a(i), b(i), c(i), y, boundl, boundu, x )
        t = (j * k) + (i - 1) + x
        return seed, t
```

5 DISCUSSION

A disadvantage of the MNO-PQRS method of smoothing piecewise-constant rate functions is that generating the next event time is more difficult than for piecewise-constant and piecewise-linear rate functions. The generation logic developed here is conceptually straightforward, being an implementation of the usual inverse-transformation logic.

Nevertheless, seeking a balance of storage efficiency, computational efficiency, and coding simplicity is a challenge. The logic could be written to compute and save all intermediate variables as a preliminary step. The value of each interval's integral could be made available. In the cyclic context when the rates are small, code that could jump ahead to future cycles would be efficient.

The routine *poisrvg* illustrates that MNO–PQRS random-process generation is reasonable, addressing the concern that smoothing rate functions via MNO–PQRS results in slower random-process generation. Piecewise-constant, piecewise-linear, and piecewise-quadratic rate functions have the same order of random-process computation (in terms of the number of intervals). Both piecewise-linear and piecewise-quadratic logics, however, need to determine interval segments with zero rates. In addition, the inversion logic is more difficult as the order of the rate function increases.

ACKNOWLEDGMENTS

The research was aided by Chung-Yuan University, Taiwan, which supported the second author as an adjunct chaired professor. We thank two referees for comments that improved presentation.

REFERENCES

Abramowitz, M. and Stegun, I.A. 1972. *Handbook of Mathematical Functions: with Formulas, Graphs, and Mathematical Tables* (AMS55). Dover Publications, Inc., New York.

Chen, H. and Schmeiser, B.W. 1992. Simulation of Poisson Processes with Trigonometric Rates. *Proceedings of the 1992 Winter Simulation Conference*, J.J. Swain, D. Goldsman, R.C. Crain, and J.R. Wilson, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., 609–617.

Chen, H. and Schmeiser, B.W. 2014. Piecewise-Quadratic Rate Functions: The Cyclic Context. *Proceedings of the 2014 Winter Simulation Conference*, A. Tolk, S. Y. Diallo, I. O. Ryzhov, L. Yilmaz, S. Buckley, and J. A. Miller, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., 486–497.

Chen, H. and Schmeiser, B.W. 2015. MNO–PQRS: Max Nonnegativity Ordering—Piecewise-Quadratic Rate Smoothing. Working paper. Available by contacting the first author with email address huifen@cycu.edu.tw.

Conte, S.D. and deBoor, C. 1980. *Elementary Numerical Analysis: An Algorithmic Approach*. New York, NY: McGraw-Hill, Inc.

Gerhardt, I. and Nelson, B.L. 2009. Transforming Renewal Processes for Simulation of Nonstationary Arrival Processes. *INFORMS Journal on Computing* 21(4): 630–640.

Klein, R.W. and Roberts, S.D. 1984. A Time-Varying Poisson Arrival Process Generator. *Simulation* 42: 193–195.

L'Ecuyer, P. 1999. Good parameters and implementations for combined multiple recursive random number generators. *Operations Research* 47, 159–164.

Lewis, P.A.W. and Shedler, G.S. 1976. Simulation of non-homogeneous Poisson processes with log-linear rate function. *Biometrika* 63, 501–505.

Lewis, P.A.W. and Shedler, G.S. 1979a. Simulation of nonhomogeneous Poisson processes with degree-two exponential polynomial rate function. *Operations Research* 27(5): 1026–1040.

Lewis, P.A.W. and Shedler, G.S. 1979b. Simulation of nonhomogeneous Poisson processes by thinning. *Naval Logistics Research Quarterly* 26(3): 403–413.

Liu, R. 2013. *Modeling and Simulation of Nonstationary Non-Poisson Processes*. Ph.D. Dissertation, Edward P. Fitts Department of Industrial and Systems Engineering, North Carolina State University.

New York State Department of Transportation. 2015. Available via https://www.dot.ny.gov/divisions/engineering/technical-services/highway-data-services/hdsb/albany [accessed June 27, 2015].

Nicol, D.M. and Leemis, L.M. 2014a. *Continuous Piecewise-Linear Intensity Function Estimation for Nonhomogeneous Poisson Process Count Data.* Technical Report, Department of Mathematics, The College of William & Mary.

Nicol, D.M. and Leemis, L.M. 2014b. A Continuous Piecewise-Linear NHPP Intensity Function Estimator. *Proceedings of the 2014 Winter Simulation Conference*, A. Tolk, S.D. Diallo, I.O. Ryzhov, L. Yilmaz, S. Buckley, and J.A. Miller, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., 498–509.

Saltzman, E.A., Drew, J.H., Leemis, L.M. and Henderson, S.G. 2012. Simulating Multivariate Nonhomogeneous Poisson Processes Using Projections. *ACM Transactions on Modeling and Computer Simulation (TOMACS)* 22(3): 1–13.

Schmeiser, B.W. 1980. Random Variate Generation: A Survey. *Proceedings of the 1980 Winter Simulation Conference*, T. I. Oren, C.M. Shub, and P. F. Roth, eds., Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc., 79–104.

AUTHOR BIOGRAPHIES

HUIFEN CHEN is a Professor in the Department of Industrial and Systems Engineering at Chung-Yuan University, Taiwan. She completed her Ph.D. in Industrial Engineering at Purdue University in 1994 and master in statistics at Purdue University in 1990. Her research interests include statistical process control, public health, and stochastic root finding. Her email address is huifen@cycu.edu.tw.

BRUCE SCHMEISER is a Professor Emeritus in the School of Industrial Engineering at Purdue University. His research interests center on developing methods for better simulation experiments. He is a fellow of INFORMS and IIE, as well as recipient of the Informs Simulation Society's 2014 Lifetime Professional Achievement Award. A long-time participant in the WSC, he served as the 1983 Program Chair and the 1988–1990 President of the Board of Directors. His e-mail address is bruceschmeiser@gmail.com.