Proceedings of the 1992 Winter Simulation Conference
ed. J. J. Swain, D. Goldsman, R. C. Crain, and J. R. Wilson

SIMULATION OF POISSON PROCESSES WITH TRIGONOMETRIC RATES

Huifen Chen
Bruce W. Schmeiser

School of Industrial Engineering
Purdue University
West Lafayette, Indiana 47907-1287, U.S.A.

ABSTRACT

We develop logic and a subroutine (*RCOSPP*) to generate the next event time, given the previous event time, of a Poisson process whose given rate function is cyclic, being composed of a constant and a trigonometric component. The event time is generated using the inverse transformation method, which requires a numerical search. We develop easy-to-compute and accurate bounds to initiate the search. *RCOSPP* is compared to off-the-shelf algorithms, also using these bounds.

1 INTRODUCTION

We consider the nonhomogeneous Poisson process $\{X(t), t \in (-\infty, \infty)\}$ with rate function

$$\lambda(t) = \mu + a\cos(2\pi(ct + b)), \qquad (1)$$

with given constants μ, a, b and c. Since $\lambda(t)$ must be nonnegative, $|a| \le \mu$. This simple process is of interest to create simple nonhomogenous systems (e.g., Taaffe and Schmeiser (1992) and Church and Uszoy (1992)), for use in some frequency-domain experimental-design methods (Schruben and Cogliano (1987), Mitra and Park (1991)) and occasionally for modeling real-world processes. Klein and Roberts (1984) used inverse transformation to derive a nonhomogeneous Poisson generator whose rate function is continuous and piecewise-linear. More general rate functions are considered in Johnson, Lee and Wilson (1991) and Lewis and Shedler (1976, 1979a).

Three approaches can be used to generate random points from the process (for example, Devroye (1986)). Thinning (for example, Lewis and Shedler (1979b)) from the homogeneous rate $\lambda = \mu + |a|$ is easy to implement. Composition from the homogeneous rate $\mu - |a|$ and the nonhomogenous rate $|a| + a\cos(2\pi(ct + b))$ is also easy to implement, using thinning from the nonhomogeneous rate and the closed-form inverse transformation for the homogeneous rate.

The third approach, which we pursue, is the inverse (or time-scale) transformation, which is required for some variance-reduction and frequency-domain methods. Let

$$m(s, r) = \int_s^r \lambda(t)\, dt.$$

Then, the number of events occurring in time interval $[s, r]$ is $X(r) - X(s)$, which is Poisson with mean $m(s, r)$. If T_i denotes the occurrence time of the i^{th} event, then the distribution function of T_i given $T_{i-1} = t_{i-1}$ is

$$F_{T_i|T_{i-1}=t_{i-1}}(t_i) = 1 - \exp\{\, -m(t_{i-1}, t_i)\, \}.$$

Then to generate T_i by the inverse-transformation method, set $u = F_{T_i|T_{i-1}=t_{i-1}}(t_i)$, where u is a uniform $(0, 1)$ random number, and solve for t_i.

In Section 2 we discuss ideas and bounds and state algorithm *RCOSPP* for efficiently solving for t_i in the inverse transformation. In Section 3 we analyze the sensitivity of *RCOSPP* to parameter values. In Section 4 we compare *RCOSPP* to a good-quality general-purpose inversion algorithm.

2 METHOD

We now discuss solving the equation $u = F_{T_i|T_{i-1}=t_{i-1}}(t_i)$ for t_i. We develop bounds for t_i and state algorithm *RCOSPP*, based on the bounds and Newton iterations.

The Poisson process is homogeneous if either $a = 0$ or $c = 0$, since then the rate function $\lambda(t)$ is a constant with respect to t. The mean number of occurrences in $[t_{i-1}, t_i]$ is then

$$m(t_{i-1}, t_i) = [\mu + a\cos(2\pi b)]\,(t_i - t_{i-1})$$

and the inverse transformation has the closed-form solution $t_i = t_{i-1} - \dfrac{\ln(1-u)}{\mu + a\cos(2\pi b)}$.

When $ac \neq 0$, the process is non-homogeneous. The distribution function of T_i given $T_{i-1} = t_{i-1}$ is

$$\begin{aligned} F_{T_i|T_{i-1}=t_{i-1}}(t_i) &= 1 - \exp\{-m(t_{i-1}, t_i)\} \\ &= 1 - \exp\{-\mu(t_i - t_{i-1}) - \\ &\quad \frac{a}{2\pi c}\{\sin(2\pi(ct_i + b)) - \sin(2\pi(ct_{i-1} + b))\}\}. \end{aligned}$$

The inverse-transformation method requires us to solve equation $u = F_{T_i|T_{i-1}=t_{i-1}}(t_i)$. The root t_i is unique, since $F(\cdot)$ is increasing. Collecting constants and simplifying, we solve $f(t_i) = 0$, where

$$f(x) = \mu x + \frac{a}{2\pi c}\sin(2\pi(cx + b)) + \delta(t_{i-1}, u) \qquad (2)$$

and

$$\delta(t_{i-1}, u) = \ln(1-u) - \mu t_{i-1} - \frac{a}{2\pi c}\sin(2\pi(ct_{i-1} + b)).$$

Given u and t_{i-1}, $\delta(t_{i-1}, u)$ is a constant, but nevertheless the solution for t_i is not closed form. We pursue a numerical solution, which requires an initial solution, a method of iterating to t_i, and a stopping rule.

In Subsection 2.1 we develop bounds for t_i based on the known values of u and t_{i-1}. From these bounds an initial point sufficient for Newton iterations to converge is derived in Subsection 2.2. A stopping rule is proposed in Subsection 2.3. The Newton-iteration algorithm *RCOSPP* is in Subsection 2.4.

2.1 Bounds

Here we establish an interval $[x_l, x_h]$ that bounds the root t_i. We know that

$$f(t_i) = \mu t_i + \frac{a}{2\pi c}\sin(2\pi(ct_i + b)) + \delta(t_{i-1}, u) = 0.$$

Then,

$$|\mu t_i + \delta(t_{i-1}, u)| = |\frac{a}{2\pi c}\sin(2\pi(ct_i + b))| \leq |\frac{a}{2\pi c}|,$$

and hence,

$$\frac{-|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu} \leq t_i \leq \frac{|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu}.$$

Furthermore, by the properties of the Poisson process, the next arrival time should be larger than last arrival time; that is, $t_i > t_{i-1}$. Then the root t_i is bounded by the points

$$x_l = \max(\frac{-|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu}, t_{i-1}),$$

and

$$x_h = \frac{|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu}.$$

2.2 Initial Solution

We now find an initial point x_0 that guarantees convergence of Newton iterations. Since $f(x)$ is twice continuously differentiable on $[x_l, x_h]$, the guarantee follows from either Condition 1 or 2.

Condition 1 *The point x_0 satisfies $t_i \leq x_0 \leq x_h$ and for all $x \in (t_i, x_h)$: (a) $f(x) \geq 0$, (b) $f'(x) > 0$, and (c) $f''(x) \geq 0$.*

Condition 2 *The point x_0 satisfies $x_l \leq x_0 \leq t_i$, and for all $x \in (x_l, t_i)$: (a) $f(x) \leq 0$, (b) $f'(x) > 0$, and (c) $f''(x) \leq 0$.*

Under Condition 1 convergence is monotonic from the right; under Condition 2 convergence is monotonic from the left. A proof of the sufficiency of Condition 1 is in Wendroff (1969, p. 36); Condition 2 has a similar proof.

The first derivative is $f'(x) = \lambda(x)$; therefore $f'(x) \geq 0$ and $f(x)$ is nondecreasing. The second derivative is

$$f''(x) = -2\pi ac\sin(2\pi(cx + b)).$$

We now proceed to find an initial solution x_0 that satisfies one of the two conditions. We consider two cases: the second derivatives at the bounds (1) have the same sign or zero and (2) have different signs.

Case 1: $f''(x_l)f''(x_h) \geq 0$

We first argue that if the second-derivative signs at the bounds are the same, then the second derivative at every point x within the bounds has the same sign.

The period of the sine function in $f''(x)$ is $1/|c|$ and the length of the interval $[x_l, x_h]$ is

$$x_h - x_l \leq 2|\frac{a}{2\pi c\mu}| \leq \frac{1}{|\pi c|} < \frac{1}{2|c|},$$

since $|a| \leq \mu$. Hence, the distance between x_l and x_h is less than an half period of sine function. Note that $\sin(2\pi(cx + b))$, and hence $f''(x)$, has constant sign for x in $[x_l, x_h]$, if $f''(x_l)f''(x_h) \geq 0$. That is, either

$$f''(x) \geq 0 \quad \text{for all } x \in [x_l, x_h]$$

or

$$f''(x) \leq 0 \quad \text{for all } x \in [x_l, x_h] .$$

The constant sign of the second derivative implies the monotonic first derivative in the interval. There are three subcases based on the first derivative.

(i) If $f'(x_l)f'(x_h) \neq 0$, then
$f'(x) > 0$ for all $x \in [x_l, x_h]$ due to the monotonic property. Since $f(x_l) \leq 0$ and $f(x_h) \geq 0$, then by Condition 1 and 2 set

$$x_0 = \begin{cases} x_h & \text{if } f''(x) \geq 0 \text{ for all } x \in [x_l, x_h] \\ x_l & \text{if } f''(x) \leq 0 \text{ for all } x \in [x_l, x_h] \end{cases}$$

(ii) If $f'(x_l) = 0$, then
$f'(x)$ is positive and increasing for all $x \in (x_l, x_h]$, since $f'(x)$ is monotonic in the interval and can not decrease to a negative value from x_l. This also implies that $f''(x) \geq 0$ for all x in the interval. Since $f(x_h) \geq 0$, then x_h satisfies Condition 1. So, set $x_0 = x_h$.

(iii) If $f'(x_h) = 0$, then
similarly $f'(x)$ is positive and decreasing for all $x \in [x_l, x_h)$. Hence, $f''(x) \leq 0$ for all x in the interval. So, satisfy Condition 2 by setting $x_0 = x_l$.

Case 2: $f''(x_l)f''(x_h) < 0$
Define the time

$$y = \begin{cases} \dfrac{\lfloor 2(cx_l + b) \rfloor + 1 - 2b}{2c} & \text{if } cx_l + b > 0 \\ \dfrac{\lfloor 2(cx_l + b) \rfloor - 2b}{2c} & \text{if } cx_l + b < 0 \end{cases}.$$

Then, $y \in (x_l, x_h)$ and $f''(y) = 0$. The second derivative $f''(x)$ has constant but different signs for $x \in [x_l, y)$ and $x \in (y, x_h]$. The signs depend on whether $f''(x_l)$ is positive or negative. The Intermediate Value Theorem (e.g., Rudin (1976, Theorem 4.23)) implies whether the root t_i is greater or less than y. If $t_i \in [x_l, y]$, replace x_h by y; otherwise $t_i \in [y, x_h]$, then replace x_l by y. Since the new interval satisfies Case 1, use the Case 1 rules to find the initial solution.

2.3 Stopping Rules

The choice of stopping rule is problem and context dependent. However, we need to choose some rule for the empirical comparisons in the next section. Our implementation refines the value of t_i until its accuracy is known within $\pm\varepsilon/\mu$; that is, the algorithm stops and returns $t_i = x_j$, if $\dfrac{|f(x_{j-1})/f'(x_{j-1})|}{1/\mu} < \varepsilon$, where ε is a given tolerance constant. We use the relative error, instead of the step size, so that the stopping rule is not a function of the choice of time unit. In addition, we restrict the maximum number of iterations to a fixed limit.

2.4 Algorithm *RCOSPP*

Our Fortran implementation of Algorithm *RCOSPP* is listed in Appendix B. Double precision is used to avoid numerical error in computing the stopping rule, as well as to allow times with many digits.

Algorithm: Given μ, a, b, c, t_{i-1} and ε, find t_i.

Step 0: Generate $u \sim U(0,1)$.

Step 1: Bound t_i with

$$x_l = \max\left(\frac{-|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu}, t_{i-1}\right)$$

and

$$x_h = \frac{|\frac{a}{2\pi c}| - \delta(t_{i-1}, u)}{\mu},$$

where $\delta(t_{i-1}, u) = \ln(1-u) - \mu t_{i-1} - \dfrac{a}{2\pi c}\sin(2\pi(ct_{i-1} + b))$.

Step 2: Find the initial point x_0 using the logic of Figure 1. Set j = 1.

Step 3: Find the next iterate using
$$x_j = x_{j-1} - \frac{f(x_{j-1})}{f'(x_{j-1})}.$$

Step 4: Stopping rule.
If $\dfrac{|f(x_{j-1})/f'(x_{j-1})|}{1/\mu} < \varepsilon$, then return $t_i = x_j$.
Otherwise, $j = j + 1$ and go to Step 3.

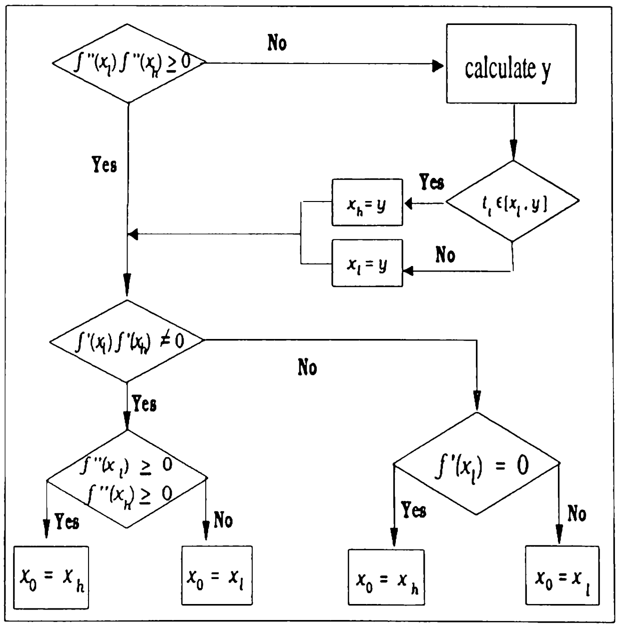

Figure 1: Flow Chart for Finding the Initial Point x_0

3 ANALYSIS

Here we investigate the robustness of *RCOSPP* to changes in the parameter values of the five parameters, μ, a, b, c, and t_{i-1}. As a location parameter of rate function $\lambda(t)$, b has no influence on the efficiency of *RCOSPP*; hence, we arbitrarily set $b = 1$ in this and the next section. Theorem 1 says that μ is also irrelevant, in that we can transform the parameter values and the result so that $\mu = 1$.

Theorem 1 *Given $\tilde{\mu}, \tilde{a}, \tilde{b}, \tilde{c}$, and $\tilde{t}_{i-1}$, let $\mu = \tilde{\mu} / \tilde{\mu} = 1$, $a = \tilde{a} / \tilde{\mu}$, $b = \tilde{b}$, $c = \tilde{c} / \tilde{\mu}$, $t_{i-1} = \tilde{t}_{i-1}\tilde{\mu}$. Substitute the new parameter values μ, a, b, c, and t_{i-1} into RCOSPP to generate t_i. Set $\tilde{t}_i = t_i/\tilde{\mu}$. Then $\tilde{t}_i$ is the arrival time that would be obtained using RCOSPP with parameter values $\tilde{\mu}, \tilde{a}, \tilde{b}, \tilde{c}$, and $\tilde{t}_{i-1}$.*

Recall that $\tilde{\mu}$ is positive for positive rate function and so the transformations above are applicable. The theorem is about the mathematics of *RCOSPP*, not the computing, since numerical error might cause minor differences between the two sets of parameter values. The proof of Theorem 1 is in Appendix A.

We further reduce the parameter space by considering only nonnegative values of the three remaining parameters a, c, and t_{i-1}, and we restrict c and t_{i-1} to satisfy $0 \le ct_{i-1} \le 1$. Although each can be negative, zero, or positive, the various combinations of signs are redundant in that the values of the rate function, and therefore computational performance, do not change.

The Monte Carlo experiment considers 25 design points, corresponding to $\mu = 1$, $b = 1$, $\varepsilon = 10^{-5}$, $a \in \{ 0.5, 1 \}$, $c \in \{ 0.001, 1, 100, 10^5 \}$, and $t_{i-1} \in \{ 0, 0.25/c, 0.5/c, 0.75/c \}$. When $c = 10^5$, the length of interval $[x_l, x_h]$ is so small (less than $\frac{1}{2|c|}$) that every point in $[x_l, x_h]$ satisfies the stopping rule, in which case we arbitrarily return t_i as the middle point of $[x_l, x_h]$. No, or almost no, iteration is taken. Hence, the levels of a and t_{i-1} make no difference on CPU time. Therefore, only $a = .5$ and $t_{i-1} = 0$ is used for $c = 10^5$. Recall that in *RCOSPP*, the length of the interval $[x_l, x_h]$, which brackets the root, depends on c (see Case 1 in Subsection 2.2). Furthermore, the initial point x_0 is in $[x_l, x_h]$. Therefore, the x_0 is very close to the root t_i when c is particularly large, causing Newton's method to converge fast.

At each design point (as defined by the levels of the three factors), twenty macroreplications of ten thousand arrival times with the same last arrival time are generated. We estimate the expected CPU time and the expected number of iterations to generate one arrival time on a Sun 4/390 under OS 4.1.1. The empirical estimates of expected CPU time are listed in Table 1 and the expected number of iterations in Table 2. All the standard errors for the empirical expected CPU times and expected number of iterations are less than .03, so the second decimal digit is meaningful, but not necessarily correct. The value of a has little practical influence on the expected CPU time and expected number of iterations, except for $c = .001$ and $t_{i-1} = .5/c = 500$.

Table 1: Empirical Expected CPU Time (in milliseconds): $\mu = 1, b = 1, \varepsilon = 10^{-5}$

		t_{i-1}			
a	c	0	$0.25/c$	$0.5/c$	$0.75/c$
.5	.001	.16	.18	.28	.19
.5	1	.22	.23	.23	.23
.5	100	.20	.19	.19	.19
.5	10^5	.08	–	–	–
1	.001	.16	.20	.43	.19
1	1	.22	.23	.23	.23
1	100	.21	.21	.21	.20

Table 2: Expected Number of Iterations: $\mu = 1, b = 1, \varepsilon = 10^{-5}$

		t_{i-1}			
a	c	0	$0.25/c$	$0.5/c$	$0.75/c$
.5	.001	2.08	1.92	4.00	1.92
.5	1	2.83	2.86	2.85	2.82
.5	100	2.03	2.03	2.03	2.03
.5	10^5	0	–	–	–
1	.001	2.05	1.96	7.69	1.96
1	1	2.93	2.97	2.94	2.91
1	100	2.35	2.33	2.35	2.34

The effect of t_{i-1} is not obvious when $c = 1$ or $c = 100$ over all values of a. The initial bias of t_{i-1} is negligible when the expected elapsed time crosses at least one cycle of rate function. Recall that the expected number of arrivals in one time unit is $\mu = 1$; therefore, the average elapsed time is 1 time unit. One rate-function cycle length is $1/|c|$. Then if $|c| \ge 1$, the expected elapsed time is longer than one cycle and the effect of t_{i-1} is negligible. When $c = .001$ (the smallest c studied), *RCOSPP* performance is worst at $t_{i-1} = .5/c = 500$ (at the valley of rates), especially when $a = 1$.

Regardless of the cases with $c = .001$ at which *RCOSPP* depends on t_{i-1}, the performance of *RCOSPP* increases with the values of c. When $c = 10^5$, no iteration is taken and *RCOSPP* converges very fast, as mentioned before. In extreme cases, such as c and ε both very small, numerical error can cause unnecessary infinite looping. Our implementation restricts looping to fifty iterations.

4 COMPARISON

We compared three other algorithms with *RCOSPP*: bisection search, subroutine *zreal* in the IMSL (1989) library, and the combination of bisection and Newton-Raphson discussed in Press, *et al.* (1986, p. 258). We found that of these reasonable off-the-shelf algorithms, the combination of bisection and Newton-Raphson performs better than the other two. After describing the combination of bisection and Newton-Raphson algorithm here, in Section 4.1 we compare it to our algorithm *RCOSPP*.

The combination of bisection and Newton-Raphson takes a bisection step whenever Newton's Method would take the solution out of bounds x_l and x_h or whenever Newton's Method is not reducing the size of the brackets rapidly enough. Here, x_l and x_h are the bounds bracketing t_i and defined as those in step 1 of *RCOSPP*. Its initial point is the center of the interval $[x_l, x_h]$. We use the same stopping rule as that in *RCOSPP*. This algorithm always converges.

4.1 Results and Analysis

In this section, at each design point, twenty macroreplications of ten thousand *consecutive* arrival times are generated. We estimate the expected CPU time and the expected number of iterations to generate ten thousand *consecutive* arrival times on a Sun 4/390 under OS 4.1.1. We have mentioned in Section 3 that μ can be transformed to 1 and b is irrelevant. Futhermore, if the replication number is so large that the generated *consecutive* t_i's cross many cycles of the rate function, the initial bias of t_{i-1} is negligible. Therefore, we arbitrary set $\mu = 1, b = 1$ and $t_{i-1} = 0$. Then, only two factors, a and c, affect the goodness of algorithms. We use the same sample spaces of a and c here as the last section except we delete $c = 10^5$, which makes no difference for these two algorithms because of the short length of $[x_l, x_h]$. Six design points with different random numbers are taken to make the experiment. Each row uses common random numbers to compare the goodness of two algorithms. The numbers in Table 3 show only meaningful digits; the last digits are not reliable.

The columns *CPU* and *iteration* show the empirical estimates of expected CPU time and the expected number of iterations needed to generate ten thousand *consecutive* arrival times. *RCOSPP* converges faster than the combination of bisection and Newton-Raphson, especially when $c = .001$.

Table 3: Empirical Expected CPU Time (in seconds) and Expected Number of Iterations (in ten thousands): $\mu = 1,\ b = 1,\ t_{i-1} = 0,\ \varepsilon = 10^{-5}$

		RCOSPP		Bisection/Newton	
a	c	CPU	iteration	CPU	iteration
.5	.001	2.6	3.19	4.1	5.75
.5	1	2.5	2.84	3.3	4.02
.5	100	2.3	2.04	3.0	3.42
1	.001	2.6	3.30	4.4	6.22
1	1	2.5	2.94	3.9	5.06
1	100	2.3	2.34	3.5	4.38

As discussed earlier, *RCOSPP*'s performance depends on c, but less on a. The combination of bisection and Newton-Raphson is more sensitive to a.

ACKNOWLEDGEMENTS

The authors received support from NSF grant DMS-8717799 to Purdue University.

APPENDIX A: PROOF OF THEOREM 1

We prove Theorem 1 only for non-homogeneous Poisson processes, since the proof for homogeneous Poisson process is trivial. Given $\tilde{\mu}, \tilde{a}, \tilde{b}, \tilde{c}$, and $\tilde{t}_{i-1}$, let $\tilde{t}_i$ denote the next arrival time. Then, the mean number of arrivals occurring in time $[\tilde{t}_{i-1}, \tilde{t}_i]$ is

$$\begin{aligned} m(\tilde{t}_{i-1}, \tilde{t}_i;\ \tilde{\mu}, \tilde{a}, \tilde{b}, \tilde{c}) &= \int_{\tilde{t}_{i-1}}^{\tilde{t}_i} \lambda(t;\ \tilde{\mu}, \tilde{a}, \tilde{b}, \tilde{c})\, dt \\ &= \tilde{\mu}(\tilde{t}_i - \tilde{t}_{i-1}) - \\ &\quad \tfrac{\tilde{a}}{2\pi \tilde{c}}[\sin(2\pi(\tilde{c}\tilde{t}_i + \tilde{b})) - \sin(2\pi(\tilde{c}\tilde{t}_{i-1} + \tilde{b}))] \\ &= (\tilde{\mu}\tilde{t}_i - \tilde{\mu}\tilde{t}_{i-1}) - \\ &\quad \tfrac{\tilde{a}/\tilde{\mu}}{2\pi\tilde{c}/\tilde{\mu}}[\sin(2\pi(\tfrac{\tilde{c}}{\tilde{\mu}}\tilde{\mu}\tilde{t}_i + \tilde{b})) - \sin(2\pi(\tfrac{\tilde{c}}{\tilde{\mu}}\tilde{\mu}\tilde{t}_{i-1} + \tilde{b}))] \\ &= (t_i - t_{i-1}) - \\ &\quad \tfrac{a}{2\pi c}[\sin(2\pi(ct_i + b)) - \sin(2\pi(ct_{i-1} + b))] \\ &= m(t_{i-1}, t_i;\ \mu, a, b, c). \end{aligned}$$

Hence, for $\tilde{t}_i = t_i/\tilde{\mu}$ the cdf of $\tilde{T}_i|\tilde{T}_{i-1}$ is the same as the cdf of T_i given T_{i-1} of the Poisson process with the corresponding parameter values.

APPENDIX B: *RCOSPP* CODE

```
c  reference: huifen chen and bruce
c    schmeiser, simulation of poisson
c    processes with trigonometric rates,
c    proceedings of the winter simulation
c    conference, 1992.
c  purpose: generate the next arrival time
c    of a Poisson process, given the last
c    arrival time, with rate function
c    lambda(t) = xmu+a*cos(two_pi*(c*t + b))

c.....example main program

      double precision a,b,c,eps,t,xmu

c.....test parameters
c        n:      number of points to generate
c        iseed: random number seed
c        eps:    accuracy tolerance
c        t:      previous event time
c        xmu:    process mean rate
c        a:      process amplitude
c        b:      process phase
c        c:      process frequency

      n     = 100
      iseed = 111222333
      eps   = .0001
      t     = 0.
      xmu   = 2.5
      a     = 1.1
      b     = 0.
      c     = 0.
c
c.....generate next arrival time
      do 100 i = 1,n
        call rcospp(xmu,a,b,c,eps,iseed,t,ier)
        print *, t
        if (ier .ne. 0) then
          print *,'error indicator =', ier
        endif
  100 continue
      stop
      end
```

```
      subroutine rcospp(xmu,a,b,c,eps,
     &                        iseed,t,ier)
c.....purpose: generate next arrival time
c        input:
c          xmu:    process mean
c          a:      process amplitude
c          b:      process phase
c          c:      process frequency
c          eps:    accuracy tolerance
c          iseed:  current random-number seed
c          t:      previous arrival time
c        output:
c          iseed:  new random-number seed
c          t:      next arrival time
c          ier:    error indicator
c                  0 ==> no error
c                  1 ==> too many iterations
c                  2 ==> rate function < 0
      common/xinp/delta,para
      double precision a,b,c,delta,eps,
     &        para,rtnewt,t,theta,two_pi,
     &        u,x_high,xlambda,x_low,xmu
      data two_pi/6.283185307/, maxit/50/

c.....generate U(0,1) random variate
      u = rand(iseed)

c.....for homogeneous case (a=0 or c=0),
c        generate next arrival time
      if (a*c .eq. 0.) then
         xlambda  = xmu + a*dcos(two_pi*b)
         if (xlambda .le. 0.) then
            ier = 2
            return
         endif
         t = t - dlog(1. - u) / xlambda
         return
      endif

c.....otherwise, for nonhomogeneous case,
c        solve the following equation for
c        x, the next arrival time.
c        0 = xmu*x + delta +
c          a/(two_pi*c)*sin(two_pi*(c*x+b)),
c        where delta is defined as below.

c.....bracket the root in [x_low, x_high].
      para   = a / (two_pi*c)
      call convert(b,c,t,theta)
      delta  = dlog(1.-u) - xmu*t -
     &                     para*dsin(theta)
      x_low  = (-dabs(para) - delta) / xmu
      x_high = ( dabs(para) - delta) / xmu
      if (x_low .lt. t) x_low = t
```

```
c.....if the length of [x_low, x_high] small,
c        return middle point.  otherwise,
c        use newton's method.
      if (((x_high-x_low)*xmu).lt.eps) then
         t = (x_high+x_low) / 2.
      else
         t = rtnewt(xmu,a,b,c,x_low,x_high,
     &                    eps,maxit,itr,ier)
      endif
      return
      end


      function rtnewt(xmu,a,b,c,x_low,
     &           x_high,eps,maxit,itr,ier)
c.....purpose:
c       solve f(x)=0 using newton's method.
c     input:
c       xmu,a,b,c: process parameters
c       x_low:  lower bound on the root
c       x_high: upper bound on the root
c       eps:    accuracy tolerance
c       maxit:  maximum # of iterations
c     output:
c       itr:    iteration number
c       ier:    error indicator
c                 0 ==> no error
c                 1 ==> too many iterations
c       rtnewt: root of f(x) = 0

      double precision a,b,c,df,dx,eps,
     &      error,f,rtnewt,x_high,x_low,xmu
      ier = 0
      call initial(xmu,a,b,c,
     &                   x_low,x_high,rtnewt)

c.....start newton's method
      do 10 itr = 1, maxit
         call funcd(xmu,a,b,c,rtnewt,f,df)
         dx     = f / df
         rtnewt = rtnewt - dx
         error  = dabs(dx) * xmu
         if (error .lt. eps) return
  10  continue
      itr = maxit
      ier = 1
      return
      end
```

```
      subroutine initial(xmu,a,b,c,
     &                    x_low,x_high,x0)
c.....purpose: find an initial solution
c     input:
c       xmu,a,b,c:     process parameters
c       x_low, x_high:  bounds on the root
c     output:
c       x0:  initial solution
      common/xinp/delta,para
      double precision xmu,a,b,c,cycle_h,
     &         cycle_l,delta,para,temp1,
     &         temp2,x0,x_high,x_low,y
      logical izerof1_l,izerof1_h
      data INEG,IZERO,IPOS/-1,0,1/
      temp1   = c*x_low+b
      temp2   = c*x_high+b
      cycle_l = temp1 - idint(temp1)
      cycle_h = temp2 - idint(temp2)
      call SecondDer(cycle_l,a,c,if2_l)
      call SecondDer(cycle_h,a,c,if2_h)
      call FirstDer(cycle_l,xmu,a,izerof1_l)
      call FirstDer(cycle_h,xmu,a,izerof1_h)
      if ( if2_l*if2_h .lt. 0 ) then
c        ...calculate y, in [x_low,x_high],
c              at which 2nd derivative is 0.
         y = (idint(2.*(c*x_low+b))
     &                         +1.-2.*b)/(c+c)
         if ((c*x_low+b) .lt. 0)
     &                          y = y-1./(c+c)
         if ((xmu*y+delta) .gt. 0.) then
c           ...the root is in [x_low, y]
            x_high = y
            if2_h  = IZERO
         else
c           ...the root is in [y, x_high]
            x_low = y
            if2_l = IZERO
         endif
      endif
c.....set initial point
      if     (izerof1_l .eq. .true.) then
         x0 = x_high
      elseif (izerof1_h .eq. .true.) then
         x0 = x_low
      else
         if ((if2_l .eq. IPOS) .or.
     &       (if2_h .eq. IPOS)) then
            x0 = x_high
         else
            x0 = x_low
         endif
      endif
      return
      end
      subroutine FirstDer(cycle,xmu,a,zero)
c.....purpose: check whether the first
c        derivative at point cycle is 0.
c     input:
c        cycle:  point for evaluation
c        xmu, a: process parameters
c     output:
c        zero:   logical variable.
c                if true, the first deriv-
c                ative at cycle is zero;
c                otherwise, nonzero.
      logical zero
      double precision cycle,xmu,a
      zero = .false.
      if (dabs(a) .lt. xmu) return
      if     ( a .eq. xmu) then
         if (dabs(cycle).eq..5) zero=.true.
      elseif (-a .eq. xmu) then
         if (     cycle .eq..0) zero=.true.
      endif
      return
      end


      subroutine SecondDer(cycle,a,c,isign)
c.....purpose: compute 2nd-derivative sign
c     input:
c        cycle:   evaluate at (cycle-b)/c
c        a, c:    process parameters
c     output:
c        isign:   second-derivative sign
      double precision a,ac,c,cycle
      data INEG,IZERO,IPOS/-1,0,1/
      ac = a * c
      if ( (     cycle  .eq. 0.) .or.
     &     (dabs(cycle) .eq. .5))  then
         isign = IZERO
      elseif (((cycle .ge. -.5) .and.
     &         (cycle .le. 0.)) .or.
     &         (cycle .ge. .5) )  then
         if (ac .gt. 0.) then
            isign = IPOS
         else
            isign = INEG
         endif
      else
         if (ac .gt. 0.) then
            isign = INEG
         else
            isign = IPOS
         endif
      endif
      return
      end
```

```
      subroutine funcd(xmu,a,b,c,x,f,df)
c.....purpose: evaluate function value
c         and its derivative at point x
c     input:
c         xmu,a,b,c:  process parameters
c         x:  point to evaluate
c     output:
c         f:  function value
c         df: first derivative
      common/xinp/delta,para
      double precision a,b,c,delta,df,f,
     &                para,x,xmu,xtheta
      call convert(b,c,x,xtheta)
      f  = xmu*x+para*dsin(xtheta)+delta
      df = xmu + a*dcos(xtheta)
      return
      end


      subroutine convert(b,c,x,theta)
c.....purpose: convert the angle (c*x+b)
c        into theta in [0, two_pi)
c     input:
c         b,c:   process parameters
c         x:     point to evaluate
c     output:
c         theta: converted angle
      double precision b,c,temp,theta,x
      double precision two_pi/6.28318530/
      temp  = c*x + b
      theta = two_pi*(temp - idint(temp))
      return
      end


      function rand(iseed)
c     u(0,1) random-number generator.
c     reference:  law & kelton,
c       simulation modeling and analysis,
c       mcgraw hill, 1982, p. 227.
      integer a,p,b15,b16,xhi,xalo,fhi
      data   a/16807/,      b15/32768/,
     &       p/2147483647/, b16/65536/
      xhi    = iseed / b16
      xalo   = (iseed-xhi*b16) * a
      leftlo = xalo  / b16
      fhi    = xhi*a + leftlo
      k      = fhi   / b15
      iseed  = ((( xalo-leftlo*b16 ) - p )
     &           + ( fhi-k*b15 )*b16 ) + k
      if (iseed .lt. 0)  iseed = iseed + p
      rand   = float(iseed) / 2147483647.
      return
      end
```

REFERENCES

Church, L. and R. Uszoy. 1992. Personal communication.

Devroye, L. 1986. *Non-uniform Random Variate Generation.* New York: Springer-Verlag.

IMSL Library Reference Manual, ed. 1.1 (1989) IMSL Inc., 7500 Bellaire Boulevard, Houston TX 77036.

Johnson, M., S. Lee, and J. R. Wilson. (1991). Experimental evaluation of a procedure for estimating nonhomogeneous Poisson processes having cyclic behavior. In *Proceedings of the 1991 Winter Simulation Conference*, ed. B. L. Nelson, W. D. Kelton, and G. M. Clark, 958–967. Institute of Electrical and Electronics Engineers, Phoenix, Arizona.

Klein, R. W. and S. D. Roberts. 1984. A time-varying Poisson arrival process generator. *Simulation* 42:193–195.

Lee, S., J. R. Wilson, and M. M. Crawford. 1991. Modeling and simulation of a nonhomogeneous Poisson process having cyclic behavior. *Communications in Statistics — Simulation and Computation* B20:777–809.

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

Lewis, P. W. and G. S. Shedler. 1979a. Simulation of nonhomogeneous Poisson processes with degree-two exponential polynomial rate function. *Operations Research* 26:1026–1040.

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

Mitra, M. and S. K. Park. 1991. Solution to the indexing problem of frequency domain simulation experiments. *Proceedings of the Winter Simulation Conference*, ed. B. L. Nelson, W. D. Kelton, and G. M. Clark, 907–915. Institute of Electrical and Electronics Engineers, Phoenix, Arizona.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. 1986. *Numerical Recipes: The Art of Scientific Computing.* Cambridge: Cambridge University Press.

Rudin, W. 1976. *Principles of Mathematical Analysis.* New York: McGraw-Hill Book Company.

Schmeiser, B. 1990. Simulation experiments. Chapter 7 in *Handbooks in Operations Research and Management Science, Volume 2: Stochastic Models.* ed. D. P. Heyman and M. J. Sobel, 295–330. Amsterdam: North-Holland.

Schruben, L. W. and V. J. Cogliano. 1987. An experimental procedure for simulation response surface model identification. *Communications of the Association for Computing Machinery* 30:716–730.

Taaffe, M. R. and B. W. Schmeiser (1992). Correlated decomposition for analyzing dynamic stochastic systems. In *Proceedings of the First Industrial Engineering Research Conference*, ed. G. Klutke, D. A. Mitta, B. O. Nnaji, and L. M. Seiford, 457–462. Institute of Industrial Engineers, Chicago, Illinois.

Wendroff, B. 1969. *First Principles of Numerical Analysis.* New York: Addison Wesley.

AUTHOR BIOGRAPHIES

HUIFEN CHEN is a Ph.D. student in the School of Industrial Engineering at Purdue University. She received a B.S. degree in accounting from National Cheng-Kung University in Taiwan in 1986 and an M.S. degree in statistics from Purdue University in 1990. Her research interests include simulation and numerical analysis applied to quality control and reliability.

BRUCE SCHMEISER is a Professor in the School of Industrial Engineering at Purdue University. His research interests include input modeling, random-variate generation, output analysis, and variance reduction. He is the current Simulation Area Editor of *Operations Research* and a Member of the Council of the Operations Research Society of America. He is an active participant in the Winter Simulation Conference, including being Program Chairman in 1983 and Chairman of the Board of Directors during 1988-1990.