Suppose we want the arrival rate $\lambda$ to change over time: $\lambda(t)$

Recall the algorithm to generate a *stationary* Poisson process:

**Stationary Poisson Process**

\[
\begin{align*}
a_0 &= 0.0; \\
n &= 0; \\
\text{while } (a_n < \tau) \{ \\
    a_{n+1} &= a_n + \text{Exponential}(1 / \lambda); \\
    n++;
\}
\]

return $a_1, a_2, \ldots, a_{n-1}$;

Above algorithm generates a stationary Poisson process

- Time interval is $0 \leq t < \tau$
- Event times are $a_1, a_2, a_3, \ldots$
- Process has constant rate $\lambda$
Incorrect Algorithm

- Change constant $\lambda$ to function:

Incorrect Algorithm

```plaintext
a_0 = 0.0;
n = 0;
while (a_n < \tau) {
    a_{n+1} = a_n + \text{Exponential}(1 / \lambda(a_n));
n++;
}
return a_1, a_2, ..., a_{n-1};
```

- Incorrect: ignores future evolution of $\lambda(t)$ after $t = a_n$
- If $\lambda(a_n) < \lambda(a_{n+1})$ then $a_{n+1} - a_n$ will tend to be too large
- If $\lambda(a_n) > \lambda(a_{n+1})$ then $a_{n+1} - a_n$ will tend to be too small
- “Inertia error” will be small only if $\lambda(t)$ varies slowly with $t$
Example 7.5.1

Piecewise-constant rate function

\[ \lambda(t) = \begin{cases} 
1 & 0 \leq t < 15 \\
2 & 15 \leq t < 35 \\
1 & 35 \leq t < 50 
\end{cases} \]

- Simulated using incorrect algorithm for \( \tau = 50 \)
- Process replicated 10000 times
- Partitioned interval \( 0 \leq t \leq 50 \) into 50 bins
- Counted number of events for each bin, divided by 10000
- Result is \( \hat{\lambda}(t) \), an estimate of \( \lambda(t) \)
Incorrect algorithm has inertia error:

- \( \hat{\lambda}(t) \) under-estimates \( \lambda(t) \) after the rate increase
- \( \hat{\lambda}(t) \) over-estimates \( \lambda(t) \) after the rate decrease

Use one of 2 correct algorithms instead
Thinning method

- Due to Lewis and Shedler, 1979
- Uses an upper bound $\lambda_{\text{max}} \geq \lambda(t)$ for $0 \leq t < \tau$
- Generates a stationary Poisson process with rate $\lambda_{\text{max}}$
- Discards (thins) some events, probabilistically
  - Event at time $s$ is kept with probability $\lambda(s)/\lambda_{\text{max}}$

- Efficiency depends on $\lambda_{\text{max}}$ being a tight bound
Algorithm 7.5.1

\begin{algorithm}
\begin{algorithmic}
\STATE $a_0 = 0.0;$
\STATE $n = 0;$
\WHILE {$(a_n < \tau)$}
\STATE $s = a_n;$
\DO
\STATE $s = s + \text{Exponential}(1 / \lambda_{\text{max}});$ 
\STATE $u = \text{Uniform}(0, \lambda_{\text{max}});$ 
\ENDDO \WHILE {$(u > \lambda(s))$}
\STATE $a_{n+1} = s;$
\STATE $n++;$
\ENDWHILE
\ENDWHILE
\RETURN $a_1, a_2, \ldots, a_{n-1};$
\end{algorithmic}
\end{algorithm}

When $\lambda(s)$ is low,
\begin{itemize}
\item The event at time $s$ is more likely to be discarded
\item The number of loop iterations is more likely to be large
\end{itemize}
Example 7.5.2

- The thinning method was applied to Example 7.5.1, using $\lambda_{\text{max}} = 2$
- Computation time increased by a factor of about 2.2
- The algorithm is not synchronized
  - Even if a separate stream is used for Uniform
Inversion Method

- Due to Çinlar, 1975
- Similar to inversion for random variate generation
- Requires only one call to Random per event
- Based upon the cumulative event rate function:

\[ \Lambda(t) = \int_0^t \lambda(s) \, ds \quad 0 \leq t < \tau \]

- \( \Lambda(t) \) represents the expected number of events in interval \([0, t)\)
- If \( \lambda(t) > 0 \) then
  - \( \Lambda(\cdot) \) is strictly monotone increasing
  - There exists an inverse \( \Lambda^{-1}(\cdot) \)
Generates a stationary “unit” Poisson process \( u_1, u_2, u_3, \ldots \)

- Equivalent to \( n \) random points in interval \( 0 < u_i < \Lambda(\tau) \)
- Each \( u_i \) is transformed into \( a_i \) using \( \Lambda^{-1}(\cdot) \)
Algorithm 7.5.2: details

The algorithm is synchronized

Useful when $\Lambda^{-1}(\cdot)$ can be evaluated efficiently
Example 7.5.3

- Use the rate function from Example 7.5.2

\[ \lambda(t) = \begin{cases} 
1 & 0 \leq t < 15 \\
2 & 15 \leq t < 35 \\
1 & 35 \leq t < 50 
\end{cases} \]

- By integration we obtain

\[ \Lambda(t) = \begin{cases} 
t & 0 \leq t < 15 \\
2t - 15 & 15 \leq t < 35 \\
t + 20 & 35 \leq t < 50 
\end{cases} \]

- Solving \( u = \Lambda(t) \) for \( t \) we obtain

\[ \Lambda^{-1}(u) = \begin{cases} 
u & 0 \leq u < 15 \\
(u + 15)/2 & 15 \leq u < 35 \\
u - 20 & 35 \leq u < 50 
\end{cases} \]
Generation time using inversion is similar to the incorrect algorithm.

For this example, $\Lambda(t)$ was easily inverted.

If $\Lambda(t)$ cannot be inverted in closed form:

- Use thinning if $\lambda_{\text{max}}$ can be found.
- Use numerical methods to invert $\Lambda(t)$.
- Use an approximation.
The three algorithms must be adapted for next-event simulation:

- Given current event time $t$, generate next event time

Incorrect Algorithm

```
arrival = t + Exponential(1 / \lambda(t));
```

Thinning Method

```
arrival = t;
do {
    arrival = arrival + Exponential(1 / \lambda_{\text{max}});
    u = Uniform(0, \lambda_{\text{max}});
} while (u > \lambda(arrival));
```

Inversion Method

```
arrival = \Lambda^{-1}(\Lambda(t) + \text{Exponential}(1.0));
```
A piecewise-constant rate function is (usually) unrealistic.

Obtaining an accurate estimate of $\lambda(t)$ is difficult.
- Requires lots of data — see Section 9.3.

We will examine piecewise-linear $\lambda(t)$ functions.
- Can be specified as a sequence of “knot pairs” $(t_j, \lambda_j)$.
Algorithm 7.5.3

Algorithm 7.5.3 Step 1

- Given \( k + 1 \) knot pairs \((t_j, \lambda_j)\) with
  - \( 0 = t_0 < t_1 < \cdots < t_k = \tau \)
  - \( \lambda_j \geq 0 \)
- Four steps to construct
  - Piecewise-linear \( \lambda(t) \)
  - Piecewise-quadratic \( \Lambda(t) \)
  - \( \Lambda^{-1}(u) \)
- Define the slope of each segment

\[
s_j = \frac{\lambda_{j+1} - \lambda_j}{t_{j+1} - t_j} \quad j = 0, 1, \ldots, k - 1
\]
Algorithm 7.5.3 Step 2

Define the *cumulative rate* for each knot point as

$$\Lambda_j = \int_0^{t_j} \lambda(t) \, dt \quad j = 0, 1, \ldots, k$$

These can be computed recursively with

$$\begin{align*}
\Lambda_0 &= 0 \\
\Lambda_j &= \Lambda_{j-1} + \frac{1}{2}(\lambda_j + \lambda_{j-1})(t_j - t_{j-1})
\end{align*}$$
Algorithm 7.5.3 Step 3

- For subinterval $t_j \leq t < t_{j+1}$

\[
\lambda(t) = \lambda_j + s_j(t - t_j)
\]

\[
\Lambda(t) = \Lambda_j + \lambda_j(t - t_j) + \frac{1}{2} s_j(t - t_j)^2
\]

If $s_j \neq 0$ then
- $\lambda(t)$ is linear
- $\Lambda(t)$ is quadratic

If $s_j = 0$ then
- $\lambda(t)$ is constant
- $\Lambda(t)$ is linear
Algorithm 7.5.3 Step 4

For subinterval $\Lambda_j \leq u < \Lambda_{j+1}$

$$\Lambda^{-1}(u) = t_j + \frac{2(u - \Lambda_j)}{\lambda_j + \sqrt{\lambda_j^2 + 2s_j(u - \Lambda_j)}}$$

If $s_j = 0$ then the above reduces to

$$\Lambda^{-1}(u) = t_j + \frac{(u - \Lambda_j)}{\lambda_j}$$
Algorithm 7.5.4: Inversion with piecewise-linear $\lambda(t)$

- Modified algorithm 7.5.2
- Also keeps track of index $j$, the current segment

```
Algorithm 7.5.4

$a_0 = 0.0;$
$u_0 = 0.0;$
$n = 0;$
$j = 0;$
while ($a_n < \tau$) {
    $u_{n+1} = u_n + \text{Exponential}(1.0);$ 
    while ($($\Lambda_{j+1} < u_{n+1}$) \text{ and } (j < k)$) 
        $j++;$
    $a_{n+1} = \Lambda^{-1}(u_{n+1})$; /* $\Lambda_j < u_{n+1} \leq \Lambda_{j+1}$ */
    $n++;$
} 
return $a_1, a_2, \ldots, a_{n-1};$
```