CHAPTER 8

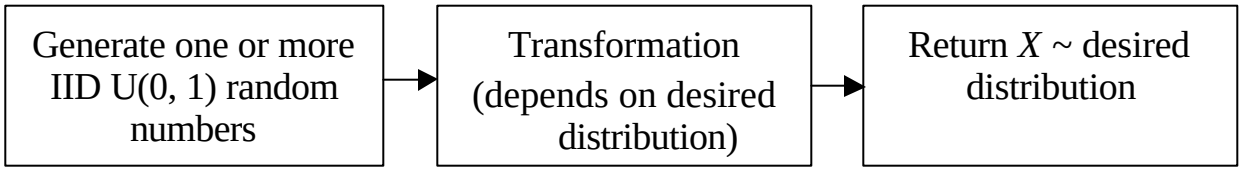
Generating Random Variates

8.1 Introduction

Algorithms to produce observations ("variates") from some desired input distribution (exponential, gamma, etc.)

Formal algorithm—depends on desired distribution

But *all* algorithms have the same general form:



Note critical importance of a good random-number generator (Chap. 7)

May be several algorithms for a desired input distribution form; want:

Exact: X has *exactly* (not approximately) the desired distribution

Example of approximate algorithm:

Treat $Z = U_1 + U_2 + \cdots + U_{12} - 6$ as N(0, 1)

Mean, variance correct; rely on CLT for *approximate* normality

Range clearly incorrect

Efficient: Low storage

Fast (marginal, setup)

Efficient regardless of parameter values (*robust*)

Simple: Understand, implement (often tradeoff against efficiency)

Requires *only* U(0, 1) input

One $U \rightarrow$ one X

(if possible—for speed, synchronization in variance reduction)

8.2 General Approaches to Generating Random Variates

Five general approaches to generating a univariate RV from a distribution:

Inverse transform
Composition
Convolution
Acceptance-rejection
Special properties

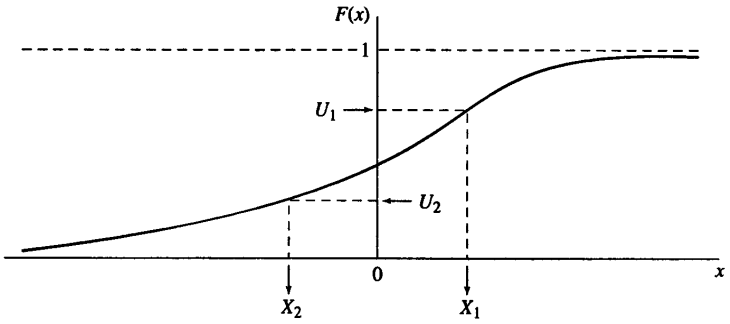
8.2.1 Inverse Transform

Simplest (in principle), "best" method in some ways; known to Gauss

Continuous Case

Suppose X is *continuous* with cumulative distribution function (CDF)

$F(x) = P(X \le x)$ for all real numbers x that is strictly increasing over all x



Algorithm:

1. Generate $U \sim U(0, 1)$ (random-number generator)
2. Find X such that $F(X) = U$ and return this value X

Step 2 involves solving the equation $F(X) = U$ for X; the solution is written

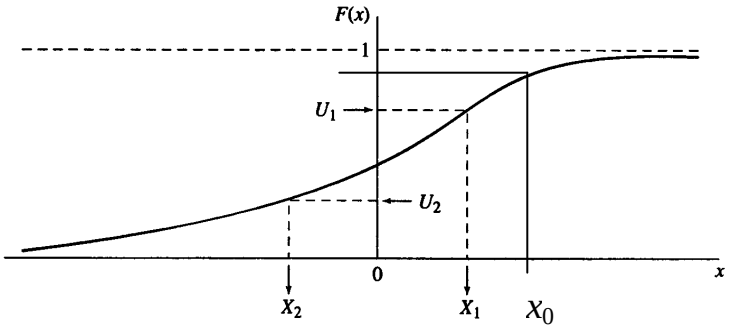
$X = F^{-1}(U)$, i.e., we must *invert* the CDF F

Inverting F might be easy (exponential), or difficult (normal) in which case numerical methods might be necessary (and worthwhile—can be made "exact" up to machine accuracy)

Proof: (Assume F is strictly increasing for all x.) For a fixed value x_0,

$$
\begin{aligned}
P(\text{returned } X \text{ is} \le x_0) &= P(F^{-1}(U) \le x_0) && \text{(def. of } X \text{ in algorithm)} \\
&= P(F(F^{-1}(U)) \le F(x_0)) && (F \text{ is monotone } \uparrow) \\
&= P(U \le F(x_0)) && \text{(def. of inverse function)} \\
&= P(0 \le U \le F(x_0)) && (U \ge 0 \text{ for sure}) \\
&= F(x_0) - 0 && (U \sim \mathrm{U}(0,1)) \\
&= F(x_0) && \text{(as desired)}
\end{aligned}
$$

Proof by Picture:



Pick a fixed value x_0

$X_1 \le x_0$ if and only if $U_1 \le F(x_0)$, so

$$
\begin{aligned}
P(X_1 \le x_0) &= P(U_1 \le F(x_0)) \\
&= F(x_0), && \text{by definition of CDFs}
\end{aligned}
$$

Example of Continuous Inverse-Transform Algorithm Derivation

Weibull (a, b) distribution, parameters $a > 0$ and $b > 0$

Density function is $f(x) = \begin{cases} a\,b^{-a}\,x^{a-1}e^{-(x/b)^a} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

CDF is $F(x) = \int_{-\infty}^{x} f(t)dt = \begin{cases} 1 - e^{-(x/b)^a} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$

Solve $U = F(X)$ for X:

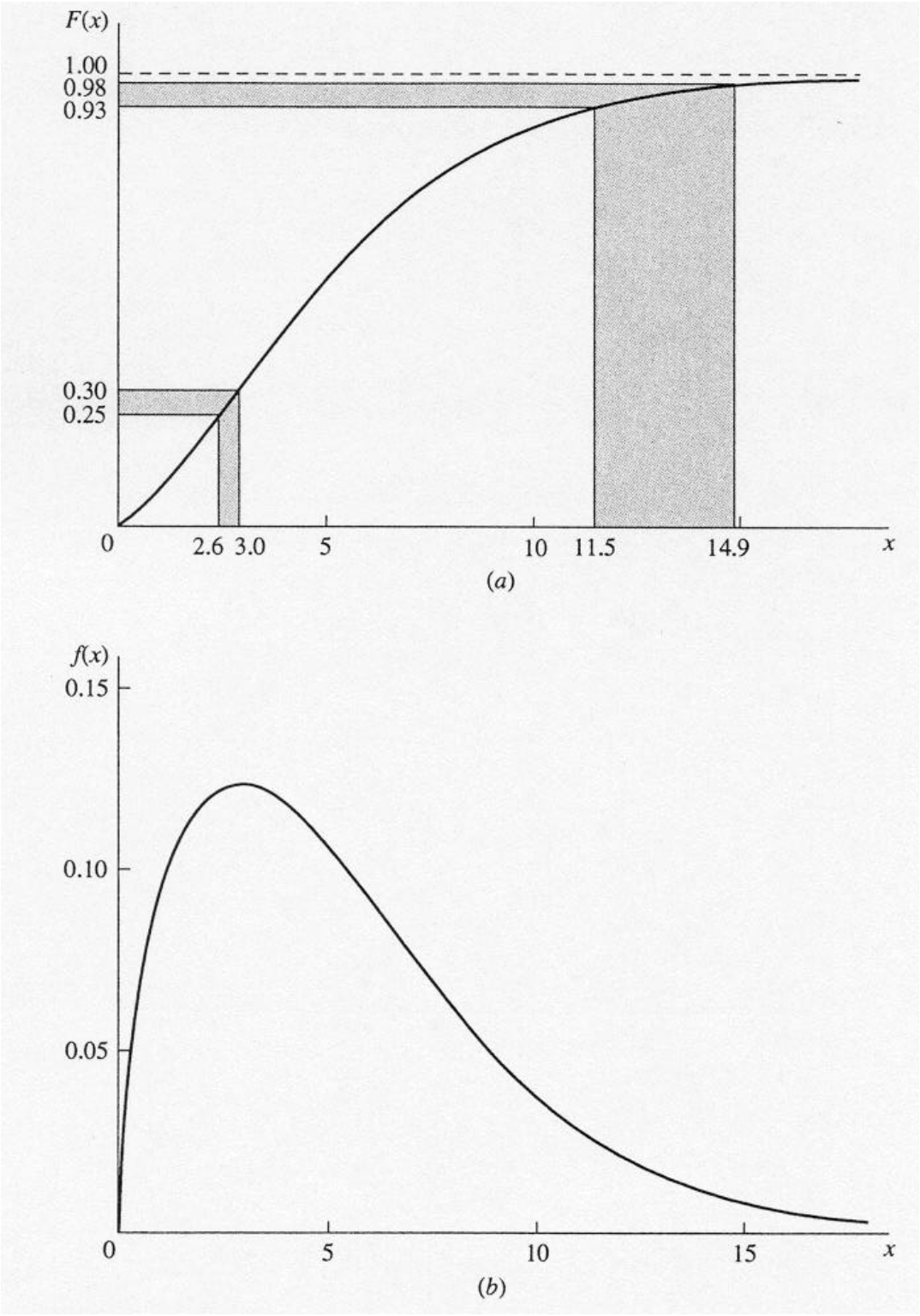
$$\begin{aligned} U &= 1 - e^{-(X/b)^a} \\ e^{-(X/b)^a} &= 1 - U \\ -(X/b)^a &= \ln(1-U) \\ X/b &= [-\ln(1-U)]^{1/a} \\ X &= b[-\ln(1-U)]^{1/a} \end{aligned}$$

Since $1 - U \sim$ U(0, 1) as well, can replace $1 - U$ by U to get the final algorithm:

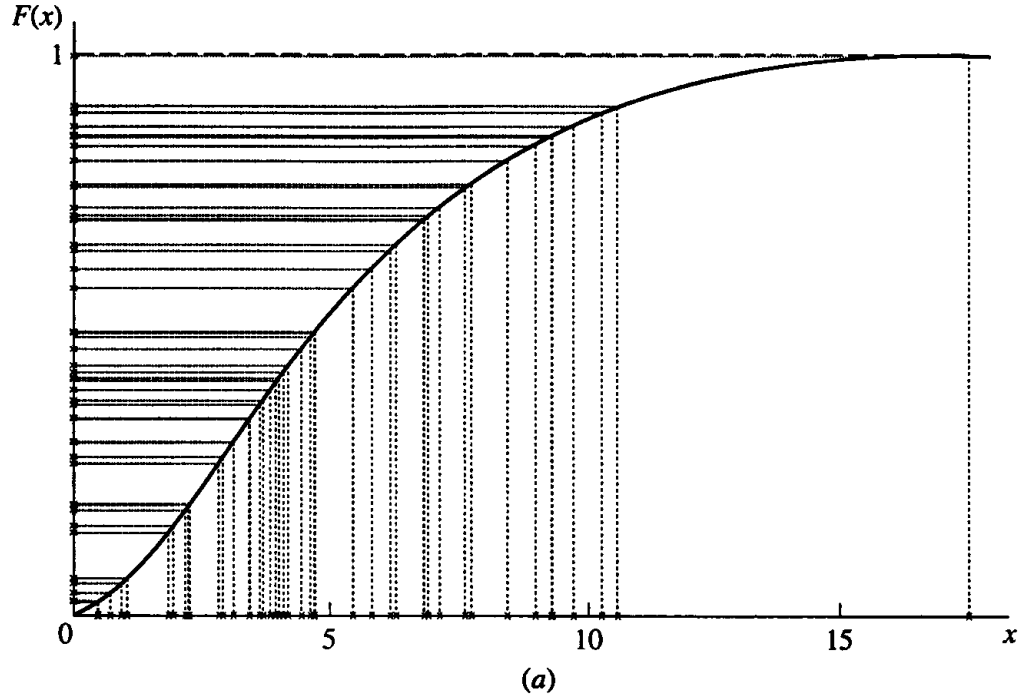
1. Generate $U \sim$ U(0, 1)
2. Return $X = b(-\ln U)^{1/a}$

Intuition Behind Inverse-Transform Method

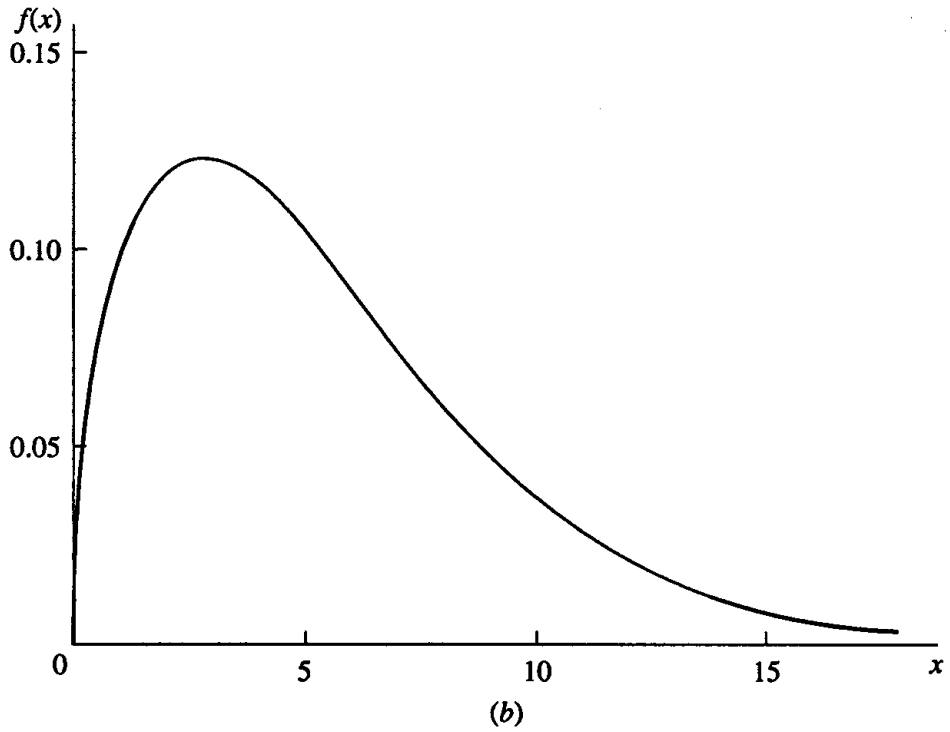
(Weibull (a = 1.5, b = 6) example)



The algorithm in action:



(a)



(b)

Discrete Case

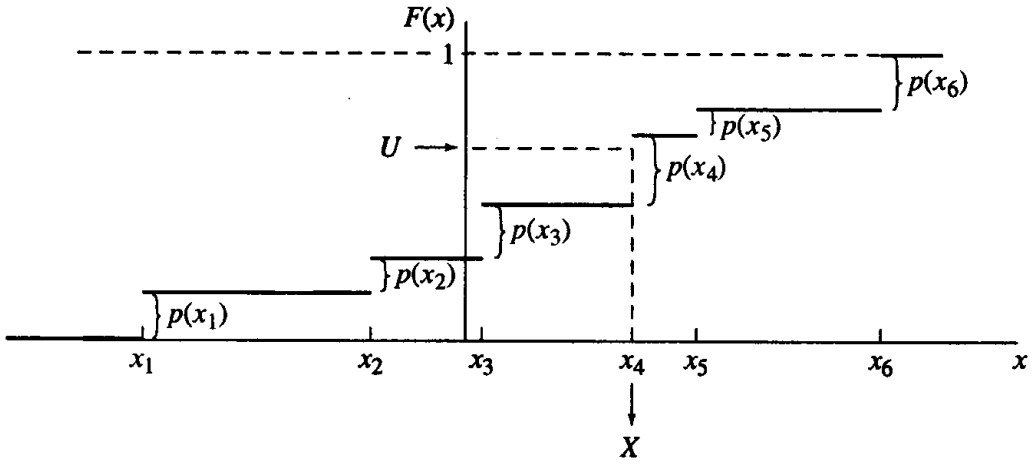
Suppose X is *discrete* with cumulative distribution function (CDF)

$$F(x) = P(X \le x) \text{ for all real numbers } x$$

and probability mass function

$$p(x_i) = P(X = x_i),$$

where $x_1, x_2, \ldots$ are the possible values X can take on



Algorithm:

1. Generate $U \sim U(0,1)$ (random-number generator)
2. Find the smallest positive integer I such that $U \le F(x_I)$
3. Return $X = x_I$

Step 2 involves a "search" of some kind; several computational options:

Direct left-to-right search—if $p(x_i)$'s fairly constant

If $p(x_i)$'s vary a lot, first sort into decreasing order, look at biggest one first, ..., smallest one last—more likely to terminate quickly

Exploit special properties of the form of the $p(x_i)$'s, if possible

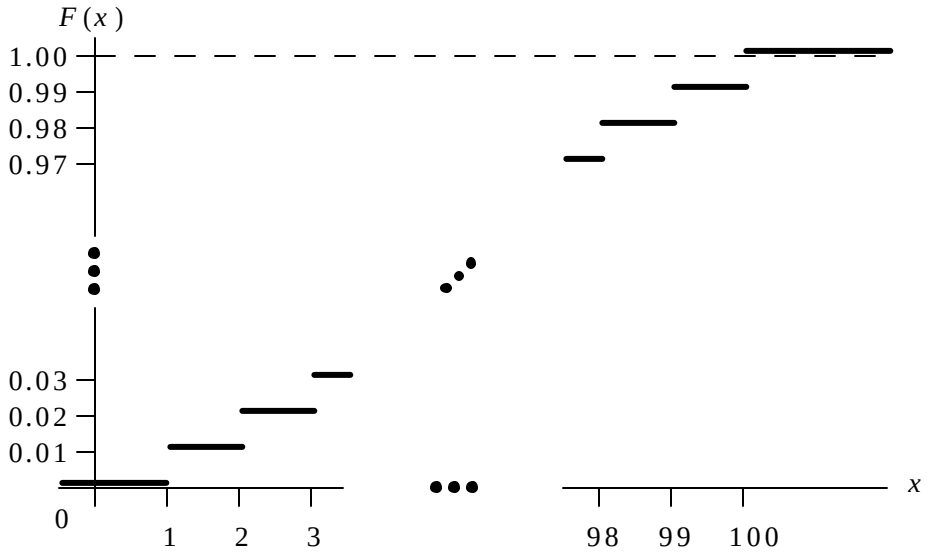
Unlike the continuous case, the discrete inverse-transform method can *always* be used for any discrete distribution (but it may not be the most efficient approach)

Proof: From the above picture, $P(X = x_i) = p(x_i)$ in every case

Example of Discrete Inverse-Transform Method

Discrete uniform distribution on 1, 2, ..., 100

$x_i = i$, and $p(x_i) = p(i) = P(X = i) = 0.01$ for $i = 1, 2, ..., 100$



"Literal" inverse transform search:

1. Generate $U \sim U(0,1)$
2. If $U \leq 0.01$ return $X = 1$ and stop; else go on
3. If $U \leq 0.02$ return $X = 2$ and stop; else go on
4. If $U \leq 0.03$ return $X = 3$ and stop; else go on
.
.
.
100. If $U \leq 0.99$ return $X = 99$ and stop; else go on
101. Return $X = 100$

Equivalently (on a U-for-U basis):

1. Generate $U \sim U(0,1)$
2. Return $X = \lfloor 100\, U \rfloor + 1$

Generalized Inverse-Transform Method

Valid for *any* CDF $F(x)$: return $X = \min\{x: F(x) \geq U\}$, where $U \sim$ U(0,1)
- Continuous, possibly with flat spots (i.e., not strictly increasing)
- Discrete
- Mixed continuous-discrete

Problems with Inverse-Transform Approach

Must invert CDF, which may be difficult (numerical methods)
May not be the fastest or simplest approach for a given distribution

Advantages of Inverse-Transform Approach

Facilitates variance-reduction techniques; an example:
- Study effect of speeding up a bottleneck machine
- Compare current machine with faster one
- To model speedup: Change service-time distribution for this machine
- Run 1 (baseline, current system): Use inverse transform to generate service-time variates for existing machine
- Run 2 (with faster machine): Use inverse transform to generate service-time variates for proposed machine
- Use the *very same* uniform U(0,1) random numbers for both variates:
  - Service times will be *positively correlated* with each other
  - Inverse-transform makes this correlation *as strong as possible*, in comparison with other variate-generation methods
  - Reduces variability in estimate of effect of faster machine
  - Main reason why inverse transform is often regarded as "the best" method
  - *Common random numbers*
  - Can have big effect on estimate quality (or computational effort required)

Generating from truncated distributions (pp. 447-448 and Problem 8.4 of *SMA*)
Generating order statistics without sorting, for reliability models:
- $Y_1, Y_2, \ldots, Y_n$ IID $\sim F$; want $Y_{(i)}$ — directly, generate Y_i's, and sort
- Alternatively, return $Y_{(i)} = F^{-1}(V)$ where $V \sim$ beta$(i, n - i + 1)$

8.2.2 Composition

Want to generate from CDF F, but inverse transform is difficult or slow

Suppose we can find other CDFs F_1, F_2, ... (finite or infinite list) and weights p_1, p_2, ... ($p_j \geq 0$ and $p_1 + p_2 + \cdots = 1$) such that for all x,

$$F(x) = p_1 F_1(x) + p_2 F_2(x) + \cdots$$

(Equivalently, can decompose density $f(x)$ or mass function $p(x)$ into convex combination of other density or mass functions)

Algorithm:

1. Generate a positive random integer J such that $P(J = j) = p_j$
2. Return X with CDF F_J (given $J = j$, X is generated independent of J)

Proof: For fixed x,

$$\begin{aligned}
\text{P(returned } X \leq x) &= \sum_j P(X \leq x \mid J = j) P(J = j) && \text{(condition on } J = j) \\
&= \sum_j P(X \leq x \mid J = j) p_j && \text{(distribution of } J) \\
&= \sum_j F_j(x) p_j && \text{(given } J = j,\ X \sim F_j) \\
&= F(x) && \text{(decomposition of } F)
\end{aligned}$$

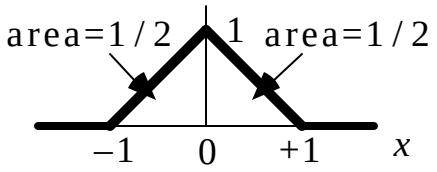
The trick is to find F_j's from which generation is easy and fast

Sometimes can use geometry of distribution to suggest a decomposition

Example 1 of Composition Method (divide area under density vertically)

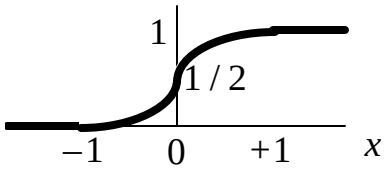
Symmetric triangular distribution on [–1, +1]:

$$\text{Density:}\quad f(x)=\begin{cases} x+1 & \text{if } -1\le x\le 0 \\ -x+1 & \text{if } 0\le x\le +1 \\ 0 & \text{otherwise} \end{cases}$$



CDF:

$$F(x)=\begin{cases} 0 & \text{if } x<-1 \\ x^2/2+x+1/2 & \text{if } -1\le x\le 0 \\ -x^2/2+x+1/2 & \text{if } 0<x\le +1 \\ 1 & \text{if } x>+1 \end{cases}$$



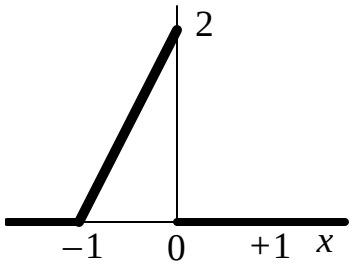
Inverse-transform:

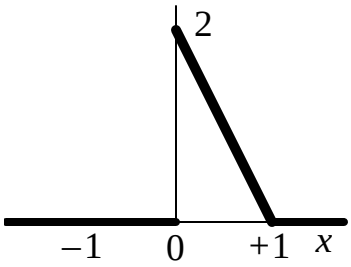
$$U=F(X)=\begin{cases} X^2/2+X+1/2 & \text{if } U<1/2 \\ -X^2/2+X+1/2 & \text{if } U\ge 1/2 \end{cases}\text{; solve for}$$

$$X=\begin{cases} \sqrt{2U}-1 & \text{if } U<1/2 \\ 1-\sqrt{2(1-U)} & \text{if } U\ge 1/2 \end{cases}$$

Composition: Define *indicator function* for the set A as $I_A(x) = \begin{cases} 1 & \text{if } x \in A \\ 0 & \text{if } x \notin A \end{cases}$

$$\begin{aligned} f(x) &= (x+1)I_{[-1,0]}(x) + (-x+1)I_{[0,+1]}(x) \\ &= \underbrace{0.5}_{p_1}\underbrace{\{2(x+1)I_{[-1,0]}(x)\}}_{f_1(x)} + \underbrace{0.5}_{p_2}\underbrace{\{2(-x+1)I_{[0,+1]}(x)\}}_{f_2(x)} \end{aligned}$$





$F_1(x) = x^2 + 2x + 1$ $\qquad$ $F_2(x) = -x^2 + 2x$

$F_1^{-1}(U) = \sqrt{U} - 1$ $\qquad$ $F_2^{-1}(U) = 1 - \sqrt{1-U}$

Composition algorithm:

1. Generate U_1, $U_2 \sim$ U(0,1) independently

2. If $U_1 < 1/2$, return $X = \sqrt{U_2} - 1$

Otherwise, return $X = 1 - \sqrt{1-U_2}$

(Can eliminate $\sqrt{\ }$'s)

Comparison of algorithms (expectations):

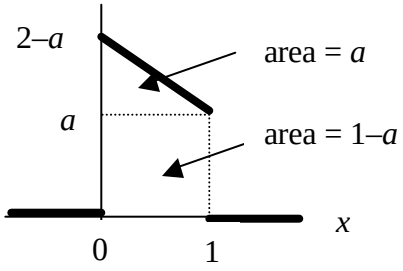
Method	U's	Compares	Adds	Multiplies	$\sqrt{\ }$'s
Inv. trnsfrm.	1	1	1.5	1	1
Composition	2	1	1.5	0	1

So composition needs one more U, one fewer multiply—faster if RNG is fast

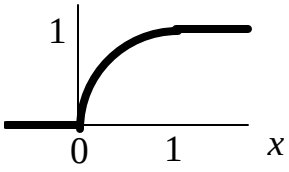
Example 2 of Composition Method (divide area under density horizontally)

Trapezoidal distribution on [0, 1] with parameter a ($0 < a < 1$):

Density: $f(x) = \begin{cases} 2 - a - 2(1-a)x & \text{if } 0 \le x \le 1 \\ 0 & \text{otherwise} \end{cases}$



CDF: $F(x) = \begin{cases} 0 & \text{if } x < 0 \\ (2-a)x - (1-a)x^2 & \text{if } 0 \le x \le 1 \\ 1 & \text{if } x > 1 \end{cases}$



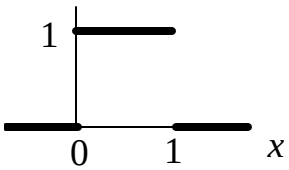
Inverse-transform:

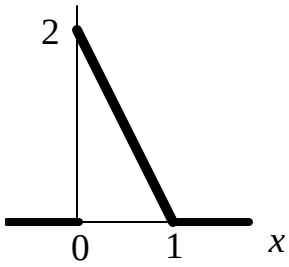
$U = F(X) = (2-a)X - (1-a)X^2$; solve for $X = \dfrac{2-a}{2(1-a)} - \sqrt{\dfrac{(a-2)^2}{4(1-a)^2} - \dfrac{U}{1-a}}$

Composition:

$$f(x) = \underbrace{a}_{p_1}\underbrace{\{I_{[0,1]}(x)\}}_{f_1(x)} + \underbrace{(1-a)}_{p_2}\underbrace{\{2(1-x)I_{[0,1]}(x)\}}_{f_2(x)}$$

Just U(0,1)





$F_1(x) = x$

$F_1^{-1}(U) = U$

$F_2(x) = -x^2 + 2x$

$F_2^{-1}(U) = 1 - \sqrt{1-U}$

Composition algorithm:

1. Generate U_1, $U_2 \sim$ U(0,1) independently
2. If $U_1 < a$, return $X = U_2$

   Otherwise, return $X = 1 - \sqrt{1-U_2}$ (Can eliminate $\sqrt{\ }$)

Comparison of algorithms (expectations):

Method	U's	Compares	Adds	Multiplies	$\sqrt{\ }$'s
Inv. trnsfrm.	1	0	2	1	1
Composition	2	1	$2(1-a)$	0	$1-a$

Composition better for large a, where F is nearly U(0,1) and it avoids the $\sqrt{\ }$

8.2.3 Convolution

Suppose desired RV X has same distribution as $Y_1 + Y_2 + \cdots + Y_m$, where the Y_j's are IID and m is fixed and finite

Write: $X \sim Y_1 + Y_2 + \cdots + Y_m$, called *m-fold convolution* of the distribution of Y_j

Contrast with composition:

Composition: Expressed the *distribution function* (or density or mass) as a (weighted) sum of other distribution functions (or densities or masses)

Convolution: Express the *random variable itself* as the sum of other random variables

Algorithm (obvious):

1. Generate $Y_1, Y_2, \cdots, Y_m$ independently from their distribution

2. Return $X = Y_1 + Y_2 + \cdots + Y_m$

Example 1 of Convolution Method

$X \sim m$-Erlang with mean $b > 0$

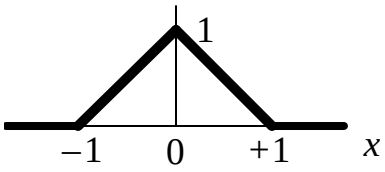
Express $X = Y_1 + Y_2 + \cdots + Y_m$ where Y_j's ~ IID exponential with mean b/m

Note that the speed of this algorithm is *not* robust to the parameter m

Example 2 of Convolution Method

Symmetric triangular distribution on [–1, +1] (again):

$$\text{Density:}\quad f(x) = \begin{cases} x+1 & \text{if } -1 \le x \le 0 \\ -x+1 & \text{if } 0 \le x \le +1 \\ 0 & \text{otherwise} \end{cases}$$



By simple conditional probability: If U_1, U_2 ~ IID U(0,1), then $U_1 + U_2$ ~ symmetric triangular on [0, 2], so just shift left by 1:

$$\begin{aligned} X &= U_1 + U_2 - 1 \\ &= \underbrace{(U_1 - 0.5)}_{Y_1} + \underbrace{(U_2 - 0.5)}_{Y_2} \end{aligned}$$

2 *U*s, 2 adds; no compares, multiplies, or $\sqrt{\ }$ s—clearly beats inverse transform, composition

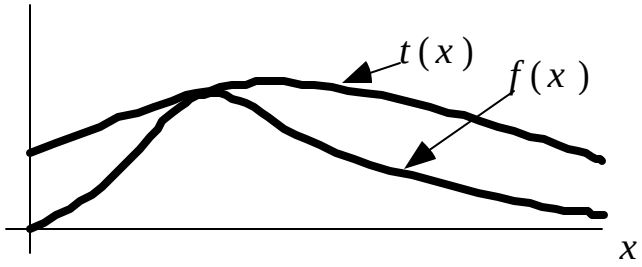
8.2.4 Acceptance-Rejection

Usually used when inverse transform is not directly applicable or is inefficient (e.g., gamma, beta)

Has continuous and discrete versions (we'll just do continuous; discrete is similar)

Goal: Generate X with density function f

Specify a function $t(x)$ that *majorizes* $f(x)$, i.e., $t(x) \geq f(x)$ for all x



Then $t(x) \geq 0$ for all x, but $\int_{-\infty}^{\infty} t(x)\,dx \geq \int_{-\infty}^{\infty} f(x)\,dx = 1$ so $t(x)$ is *not* a density

Set $c = \int_{-\infty}^{\infty} t(x)\,dx \geq 1$

Define $r(x) = t(x)/c$ for all x

Thus, $r(x)$ *is* a density (integrates to 1)

Algorithm:

1. Generate Y having density r
2. Generate $U \sim U(0,1)$ (independent of Y in Step 1)
3. If $U \leq f(Y)/t(Y)$, return $X = Y$ and stop;
   else go back to Step 1 and try again

(Repeat 1— 2 — 3 until acceptance finally occurs in Step 3)

Since t majorizes f, $f(Y)/t(Y) \leq 1$ so "$U \leq f(Y)/t(Y)$" may or may not occur for a given U

Must be able to generate Y with density r, hopefully easily—choice of t

On each pass, P(acceptance) = $1/c$, so want small c = area under $t(x)$, so want t to "fit" down on top of f closely (i.e., want t and thus r to resemble f closely)

Tradeoff between ease of generation from r, and closeness of fit to f

Proof: Key—we *get* an X only *conditional* on acceptance in step 3. So

$$\begin{aligned} P(\text{generated } X \le x) &= P(Y \le x \mid \text{acceptance}) \\ &= \frac{P(\text{acceptance}, Y \le x)}{P(\text{acceptance})} \quad \text{(def. of cond'l. prob.)} \quad * \end{aligned}$$

(Evaluate top and bottom of *.)

For any y,

$$P(\text{acceptance} \mid Y = y) = P(U \le f(y)/t(y)) = f(y)/t(y)$$

since $U \sim$ U(0,1), Y is independent of U, and $t(y) > f(y)$. Thus,

$$\begin{aligned} P(\text{acceptance}, Y \le x) &= \int_{-\infty}^{\infty} P(\text{acceptance}, Y \le x \mid Y = y) r(y)\, dy \\ &= \underbrace{\int_{-\infty}^{x} P(\text{acceptance}, Y \le x \mid Y = y) r(y)\, dy}_{Y \le x \text{ on this range, guaranteeing } Y \le x \text{ in the probability}} + \\ &\quad \underbrace{\int_{x}^{\infty} P(\text{acceptance}, Y \le x \mid Y = y) r(y)\, dy}_{Y \ge x \text{ on this range, contradicting } Y \le x \text{ in the probability}} \\ &= \int_{-\infty}^{x} P(\text{acceptance}, Y \le x \mid Y = y) r(y)\, dy \\ &= \frac{1}{c} \int_{-\infty}^{x} \frac{f(y)}{t(y)} t(y)\, dy \quad \text{(def. of } r(y)) \\ &= F(x)/c \qquad ** \end{aligned}$$

Next,

$$
\begin{aligned}
P(\text{acceptance}) &= \int_{-\infty}^{\infty} P(\text{acceptance} \mid Y = y) r(y)\, dy \\
&= \frac{1}{c} \int_{-\infty}^{\infty} \frac{f(y)}{t(y)} t(y)\, dy \\
&= \frac{1}{c} \int_{-\infty}^{\infty} f(y)\, dy \\
&= 1/c \qquad\qquad *** \\
\end{aligned}
$$

since f is a density and so integrates to 1. Putting ** and *** back into *,

$$
\begin{aligned}
P(\text{generated } X \le x) &= \frac{P(\text{acceptance}, Y \le x)}{P(\text{acceptance})} \\
&= \frac{F(x)/c}{1/c} \\
&= F(x),
\end{aligned}
$$

as desired.

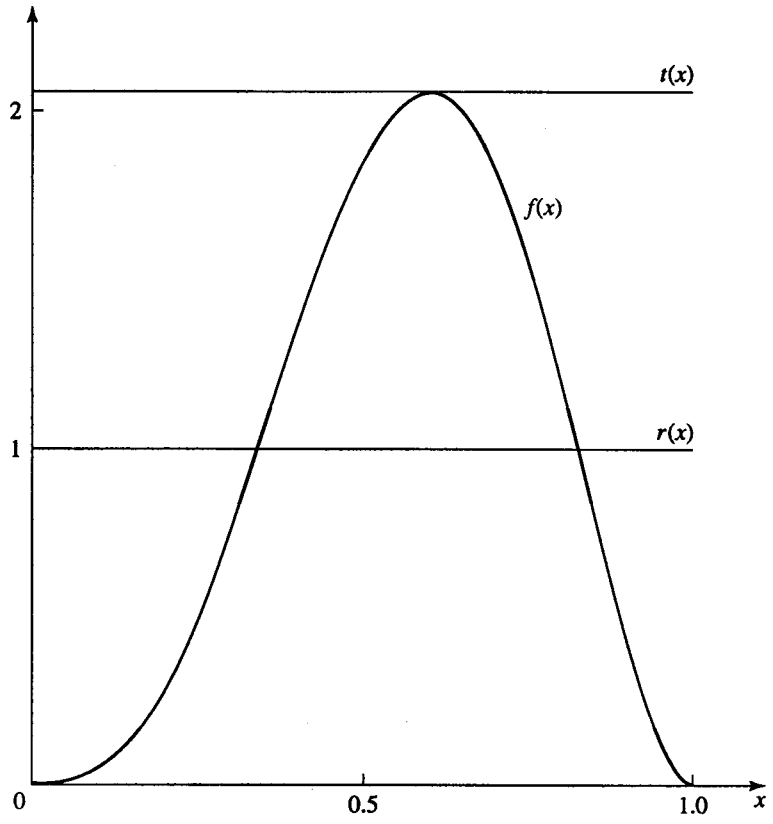
(Depressing footnote: John von Neumann, in his 1951 paper developing this idea, needed only a couple of sentences of words — no math or even notation — to see that this method is valid.)

Example of Acceptance-Rejection

Beta(4,3) distribution, density is $f(x) = 60\ x^3\ (1 - x)^2$ for $0 \le x \le 1$

Top of density is $f(0.6) = 2.0736$ (exactly), so let $t(x) = 2.0736$ for $0 \le x \le 1$

Thus, $c = 2.0736$, and r is the U(0,1) density function

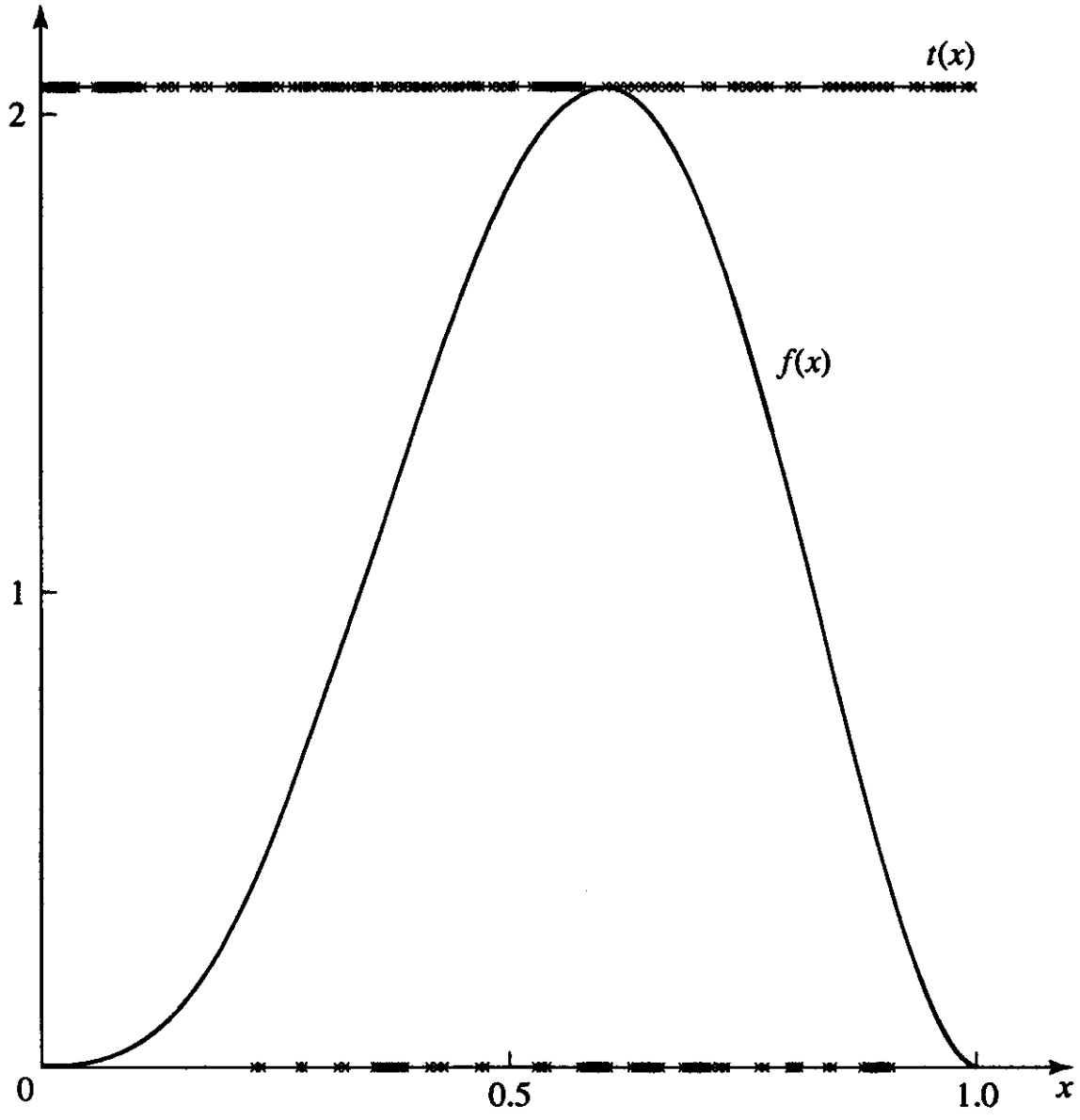


Algorithm:

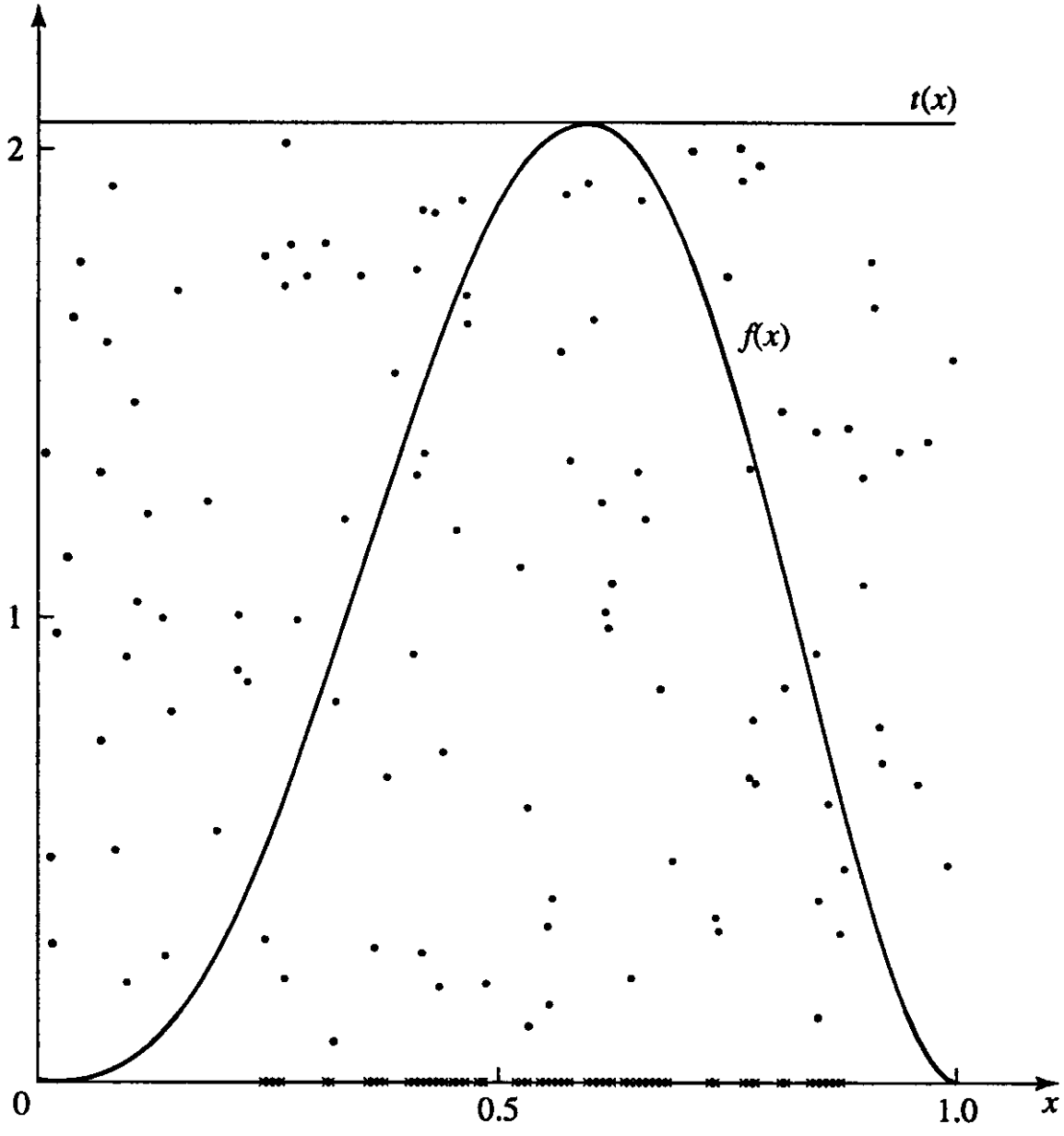
1. Generate $Y \sim$ U(0,1)
2. Generate $U \sim$ U(0,1) independent of Y
3. If $U \le 60\ Y^3\ (1 - Y)^2 / 2.0736$, return $X = Y$ and stop;
   else go back to step 1 and try again

P(acceptance) in step 3 is 1/2.0736 = 0.48

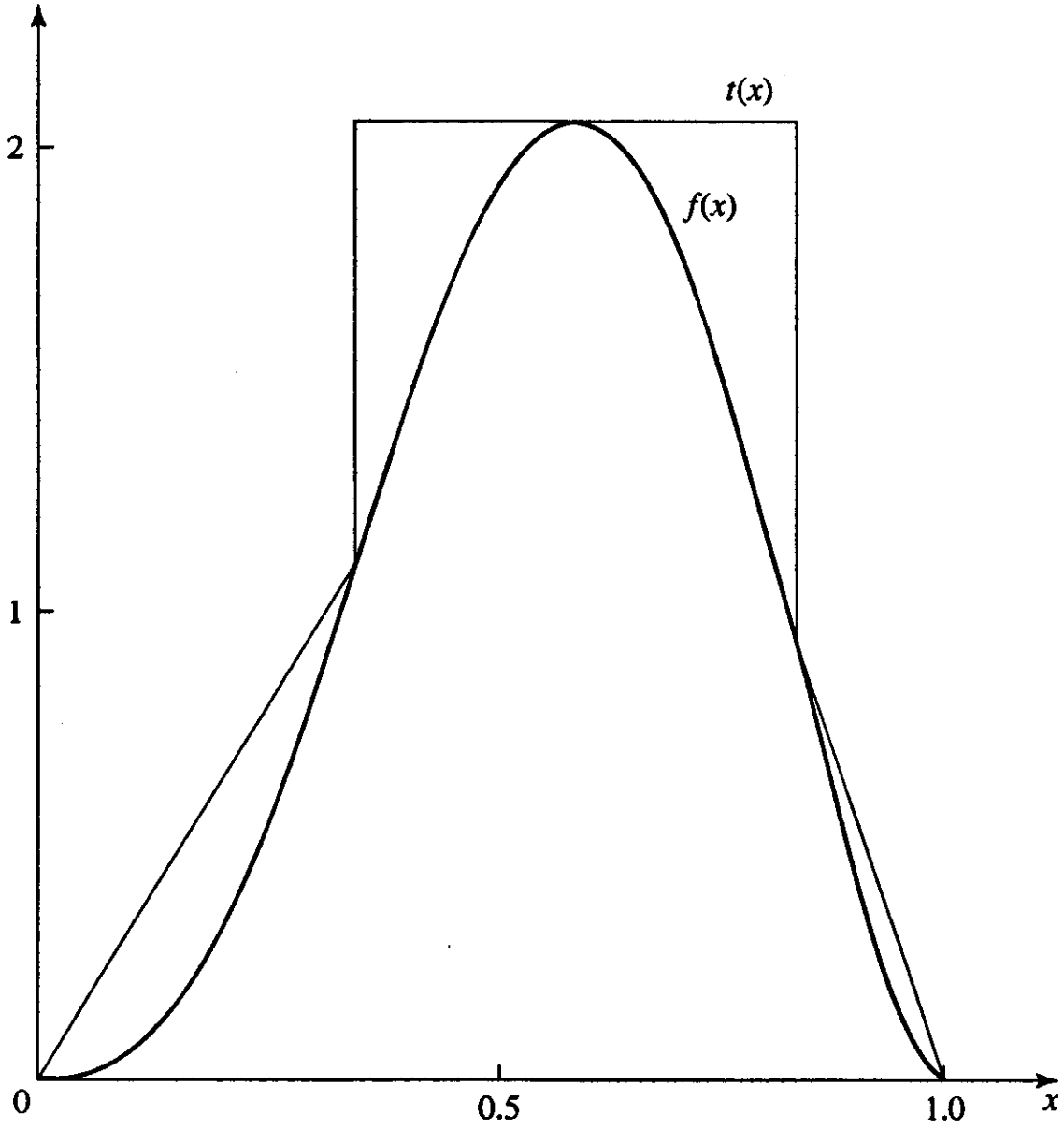
Intuition



A different way to look at it—accept Y if $U\,t(Y) \le f(Y)$, so plot the pairs $(Y, U\,t(Y))$ and accept the Y's for which the pair is under the f curve



A closer-fitting majorizing function:



Higher acceptance probability on a given pass

Harder to generate Y with density shaped like t (composition)

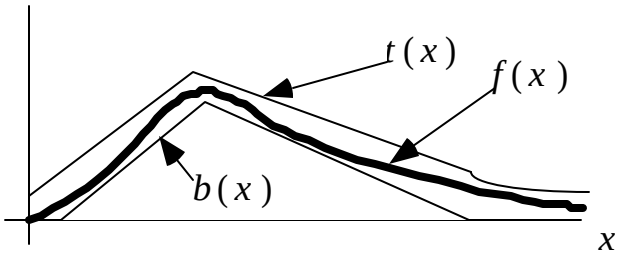
Better ???

Squeeze Methods

Possible slow spot in A-R is evaluating $f(Y)$ in step 3, if f is complicated

Add a fast *pre-test* for acceptance just before step 3—if pre-test is passed we know that the test in step 3 would be passed, so can quit without actually doing the test (and evaluating $f(Y)$).

One way to do this — put a *minorizing* function $b(x)$ under $f(x)$:



Since $b(x) \le f(x)$, pre-test is first to check if $U \le b(Y)/t(Y)$; if so accept Y right away (if not, have to go on and do the actual test in step 3)

Good choice for $b(x)$:

Close to $f(x)$ (so pre-test and step 3 test agree most of the time)

Fast and easy to evaluate $b(x)$

8.2.5 Special Properties

Simply "tricks" that rely completely on a given distribution's form

Often, combine several "component" variates algebraically (like convolution)

Must be able to figure out distributions of functions of random variables

No coherent general form — only examples

Example 1: Geometric

Physical "model" for $X \sim$ geometric with parameter p $(0 < p < 1)$:

X = number of "failures" before first success in Bernoulli trials with $P(\text{success}) = p$

Algorithm: Generate Bernoulli(p) variates and count the number of failures before first success

Clearly inefficient if p is close to 0

Example 2: Beta

If $Y_1 \sim \text{gamma}(\alpha_1, 1)$, $Y_2 \sim \text{gamma}(\alpha_2, 1)$, and they are independent, then

$$X = Y_1/(Y_1 + Y_2) \sim \text{beta}(\alpha_1, \alpha_2)$$

Thus, we effectively have a beta generator if we have a gamma generator

8.3 Generating Continuous Random Variates

Sixteen families of continuous distributions found useful for modeling simulation input processes

Correspond to distributions defined in Chap. 6

At least one variate-generation algorithm for each is specifically given on pp. 459–471 of *SMA*

Algorithms selected considering exactness, speed, and simplicity — often there are tradeoffs involved among these criteria

8.4 Generating Discrete Random Variates

Seven families of discrete distributions found useful for modeling simulation input processes

Correspond to distributions defined in Chap. 6

At least one variate-generation algorithm for each is specifically given on pp. 471–478 of *SMA*

Algorithms selected considering exactness, speed, and simplicity — often there are tradeoffs involved among these criteria

One of these seven is completely general if the range of the random variable is finite, and will be discussed separately in Sec. 8.4.3 ...

8.4.3 Arbitrary Discrete Distribution

Common situation: Generate discrete $X \in \{0, 1, 2, ..., n\}$ with mass function $p(i) = P(X = i)$, $i = 0, 1, 2, ..., n$

In its own right to represent, say, lot sizes in a manufacturing simulation

As part of other variate-generation methods (e.g., composition)

Why restrict to range $\{0, 1, 2, ..., n\}$ rather than general $\{x_1, x_2, ..., x_m\}$?

Not as restrictive as it seems:

Really want a general range $\{x_1, x_2, ..., x_m\}$

Let $n = m - 1$ and let $p(j - 1) = P(X = x_j)$, $j = 1, 2, ..., m$ $(= n + 1)$
(so $j - 1 = 0, 1, ..., m - 1$ $(= n)$)

Algorithm:

1. Generate J on $\{0, 1, 2, ..., n\}$ with mass function $p(j)$
2. Return $X = x_{J+1}$

Have already seen one method to do this: Inverse transform

Always works

But may be slow, especially for large range (n)

Table Lookup

Assume that each $p(i)$ can be represented as (say) a 2-place decimal

Example:

i	0	1	2	3	
$p(i)$	0.15	0.20	0.37	0.28	1.00 = sum of $p(i)$'s (must be exact — no roundoff allowed)

Initialize a vector $(m_1, m_2, ..., m_{100})$ with

$m_1 = m_2 = \cdots = m_{15} = 0$ (first $100p(0)$ m_j's set to 0)

$m_{16} = m_{17} = \cdots = m_{35} = 1$ (next $100p(1)$ m_j's set to 1)

$m_{36} = m_{37} = \cdots = m_{72} = 2$ (next $100p(2)$ m_j's set to 2)

$m_{73} = m_{74} = \cdots = m_{100} = 3$ (last $100p(3)$ m_j's set to 3)

Algorithm (obvious):

1. Generate J uniformly on $\{1, 2, ..., 100\}$ $(J = \lfloor 100U \rfloor + 1)$

2. Return $X = m_J$

Advantages:

Extremely simple

Extremely fast (marginal)

Drawbacks:

Limited accuracy on $p(i)$'s—use 3 or 4 decimals instead?

(Few decimals OK if $p(i)$'s are themselves inaccurate estimates)

Storage is 10^d, where d = number of decimals used

Setup required (but not much)

Marsaglia Tables

As above, assume $p(i)$'s are q-place decimals ($q = 2$ above); set up tables

But will use less storage, a little more time

Same example:

i	0	1	2	3	
$p(i)$	*0.15*	*0.20*	*0.37*	*0.28*	1.00 = sum of $p(i)$'s (must be exact—no roundoff allowed)

Initialize a vector for each decimal place (here, need $q = 2$ vectors):

"Tenths" vector: Look at tenths place in each $p(i)$; put in that many copies of the associated i

```
0 1 1 2 2 2 3 3   ← 10ths vector
1  2   3   2
```

"Hundredths" vector: Look at hundredths place in each $p(i)$; put in that many copies of the associated i

```
0 0 0 0 0 ___ 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3   ← 100ths
  vector
   5       0       7             8
```

Total storage = 8 + 20 = 28 = sum of all the digits in the $p(i)$'s

Was 100 for table-lookup method

Algorithm:

1. Pick 10ths vector with prob. 1/10 × (sum of 10ths digits) = 8/10.
If picked, return X = one of the entries in this vector with equal probability (1/8 here).
If not picked, go on to step 2.

2. Pick 100ths vector with prob. 1/100 × (sum of 100ths digits) = 20/100.
If picked, return X = one of the entries in this vector with equal probability (1/20 here).
If not picked, go on to step 3. (Won't happen here.)

3. (Not present here.) Pick 1000ths vector with prob. 1/1000 × (sum of 1000ths digits)
If picked, return X = one of the entries in this vector with equal probability.
If not picked, go on to step 4.

etc.

Proof (by example for $i = 2$; other cases exactly analogous):

$$\begin{aligned} P(\text{generated } X = 2) &= P(X = 2 \mid \text{pick 10ths vector})P(\text{pick 10ths vector}) + \\ &\quad P(X = 2 \mid \text{pick 100ths vector})P(\text{pick 100ths vector}) \\ &= \frac{3}{8} \times \frac{8}{10} \quad + \quad \frac{7}{20} \times \frac{20}{100} \\ &= 0.37, \quad \text{as desired.} \end{aligned}$$

Main advantage: Less storage than table-lookup, especially for large number of decimals required in the $p(i)$'s

The Alias Method

Improvement over A-R: If we "reject," we don't give up and start all over, but instead return the *alias* of the generated Y

Set up two vectors of length $n + 1$ each:

Aliases $L_0, L_1, ..., L_n \in \{0, 1, ..., n\}$

Cutoffs $F_0, F_1, ..., F_n \in [0, 1]$

(Setup methods for aliases and cutoffs on pp. 490–491 of *SMA*.)

Algorithm:

1. Generate I uniformly on $\{0, 1, ..., n\}$ $\quad (I = \lfloor (n + 1)U \rfloor)$
2. Generate $U_0 \sim U(0,1)$ independent of I
3. If $U_0 \le F_I$, return $X = I$; otherwise, return $X = L_I$

Note that a "rejection" in step 3 results in returning the alias of I, rather than throwing I out and starting all over, as in A-R

Proof: Complicated; embodied in algorithms to set up aliases and cutoffs

Intuition:

Approximate $p(i)$'s initially by simple discrete uniform I on $\{0, 1, ..., n\}$

Thus, $I = i$ with probability $1/(n + 1)$ for each $i \in \{0, 1, ..., n\}$

For i with $p(i) << 1/(n + 1)$, cutoff F_i is small, so will probably "move away" to another value L_i for which $p(L_i) >> 1/(n + 1)$

For i with $p(i) >> 1/(n + 1)$, cutoff F_i is large (like 1) or alias of i is itself ($L_i = i$), so will probably "keep" all the $I = i$ values, as well as receive more from other values with low desired probabilities

Clever part: can get an algorithm for cutoffs and aliases so that this shifting works out to exactly the desired $p(i)$'s in the end

Advantage: Very fast

Drawbacks:

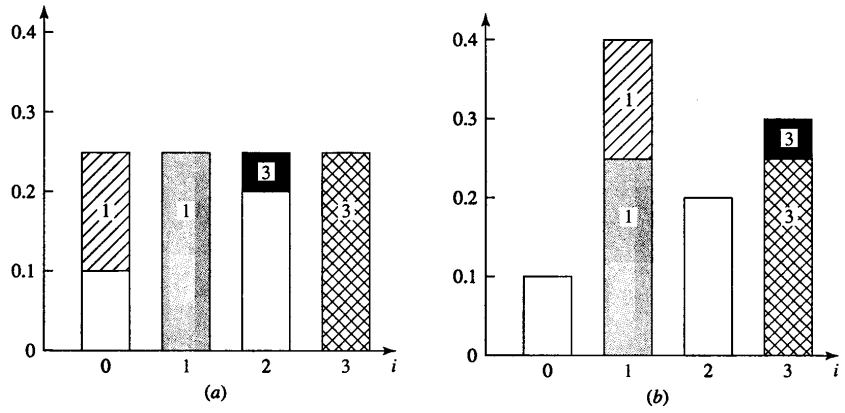
Requires storage of $2(n + 1)$; can be reduced to $n + 1$, still problematic

Requires the initial setup of aliases and cutoffs

Example of Alias method:

i	0	1	2	3
$p(i)$	0.1	0.4	0.2	0.3
F_i	0.4	0.0	0.8	0.0
L_i	1	1	3	3

Picture:



How can the algorithm return $X = 2$?

Since 2 is not the alias of anything else, can get $X = 2$ only if we generate $I = 2$ and keep it (i.e., don't change $I = 2$ to its alias $F_2 = 3$)

Thus,

$$P(X = 2) = P(I = 2 \text{ and } U \le 0.8)$$
$$= P(I = 2)\,P(U \le 0.8) \text{ since } I \text{ and } U \text{ are generated independently}$$
$$= 0.25 \times 0.8 = 0.2, \text{ as desired}$$

How can the algorithm return $X = 3$?

Can get $X = 3$ in two mutually exclusive ways:

Generate $I = 3$ (since alias of 3 is $L_3 = 3$, will always get a 3 here)

Generate $I = 2$ but change it to its alias 3

Thus,

$$P(X = 3) = P(I = 3) + P(I = 2 \text{ and } U > 0.8)$$
$$= 0.25 + 0.25 \times 0.2 = 0.3, \text{ as desired}$$

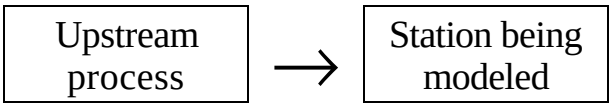
8.5 Generating Random Vectors, Correlated Random Variates, and Stochastic Processes

So far: IID univariate RVs

Sometimes have correlation between RVs in reality:

A = interarrival time of a job from an upstream process

S = service time of job at the station being modeled



Perhaps a large A means that the job is "large," taking a lot of time upstream—then it probably will take a lot of time here too (S large)

i.e., $\text{Cor}(A, S) > 0$

Ignoring this correlation can lead to serious errors in output validity (see notes for Chap. 6 for some specific numerical examples)

Need ways to generate it in the simulation

May want to simulate entire joint distribution of a random *vector* for input:

Multivariate normal in econometric or statistical simulation

Important distinction:

Full joint distribution of random vector $\mathbf{X} = (X_1, X_2, ..., X_n)^T \in \Re^n$	vs.	Marginal distributions of each X_i and all covariances or correlations

These are the same thing only in the case of multivariate normal

Could want either in simulation

8.5.1 Using Conditional Distributions

Suppose we know the entire joint distribution for a random vector X, i.e., for a fixed $\mathbf{x} = (x_1, x_2, ..., x_n)^T \in \Re^n$, we know the value of the joint CDF

$$F(\mathbf{x}) = P(\mathbf{X} \le \mathbf{x}) = P(X_1 \le x_1, ..., X_n \le x_n)$$

This determines all the covariances and correlations

General method:

For $i = 1, 2, ..., n$, let $F_i(\bullet)$ be the marginal distribution of X_i

For $k = 2, 3, ..., n$, let $F_k(\bullet \mid X_1, X_2, ..., X_{k-1})$ be the conditional distribution of X_k given $X_1, X_2, ..., X_{k-1}$

Algorithm:

1. Generate X_1 (marginally) from F_1

2. Generate X_2 from $F_2(\bullet \mid X_1)$

3. Generate X_3 from $F_3(\bullet \mid X_1, X_2)$

.

.

.

***n*.** Generate X_n from $F_n(\bullet \mid X_1, X_2, ..., X_{n-1})$

***n* + 1.** Return $\mathbf{X} = (X_1, X_2, ..., X_n)^T$

Proof: Tedious but straightforward manipulation of definitions of marginal and conditional distributions

Completely general concept

Requires a lot of input information

8.5.2 Multivariate Normal and Multivariate Lognormal

One case where knowing the marginal distributions and all the covariances/correlations is equivalent to knowing the entire joint distribution

Want to generate multivariate normal random vector $\mathbf{X}$ with:

Mean vector $\mathtt{m} = (m_1, m_2, ..., m_n)^T$

Covariance matrix $\mathrm{S} = [s_{ij}]_{n \times n}$

Since S must be positive definite, there is a unique lower-triangular $n \times n$ matrix $\mathbf{C}$ such that $\mathrm{S} = \mathbf{C}\mathbf{C}^T$ (there are linear-algebra algorithms to do this)

Algorithm:

1. Generate $Z_1, Z_2, ..., Z_n$ as IID N(0,1), and let $\mathrm{Z} = (Z_1, Z_2, ..., Z_n)^T$
2. Return $\mathbf{X} = \mathtt{m} + \mathbf{C}\mathbf{Z}$

Note that the final step is just higher-dimensional version of the familiar transformation $X = m + sZ$ to get $X \sim \mathrm{N}(m, s)$ from $Z \sim \mathrm{N}(0,1)$

Can be modified to generate a multivariate lognormal random vector

8.5.3 Correlated Gamma Random Variates

Want to generate $\mathbf{X} = (X_1, X_2, ..., X_n)^T$ where

$X_i \sim$ gamma(a_i, b_i), a_i's, b_i's specified

Cor(X_i, X_j) = r_{ij} (specified)

Useful ability, since gamma distribution is flexible (many different shapes)

Note that we are *not* specifying the whole joint distribution, so there may be different **X**'s that will satisfy the above but have different joint distributions

Difficulties:

The a_i's place limitations on what r_{ij}'s are theoretically possible — that is, there may not even be such a distribution and associated random vector

Even if the desired **X** *is* theoretically possible, there may not be an algorithm known that will work

Even the known algorithms do not have control over the joint distribution

One known case:

Bivariate ($n = 2$); let $r = r_{12}$

$0 \le r \le \min\{a_1, a_1\} / \sqrt{a_1 a_1}$

Positive correlation, bounded above

If $a_1 = a_2$ the upper bound is 1, i.e., is removed

If $a_1 = a_2 = 1$, have any two positively correlated exponentials

Algorithm (*trivariate reduction*):

1. Generate $Y_1 \sim$ gamma($a_1 - r\sqrt{a_1 a_2}$, 1)

2. Generate $Y_2 \sim$ gamma($a_2 - r\sqrt{a_1 a_2}$, 1)

3. Generate $Y_3 \sim$ gamma($r\sqrt{a_1 a_2}$, 1)

4. Return $\mathbf{X} = (X_1, X_2)^T = (b_1(Y_1 + Y_3), b_2(Y_2 + Y_3))^T$

Correlation is carried by Y_3, common to both X_1 and X_2

Other solved problems:

Bivariate gamma with any theoretically possible correlation (+ or –)

General n-dimensional gamma, but with restrictions on the correlations that are more severe than those imposed by existence

Negatively correlated gammas with common a_i's and b_i's

Any theoretically possible set of marginal gamma distributions and correlation structure (see Sec. 8.5.5 below)

8.5.4 Generating from Multivariate Families

Methods exist for generating from:

Multivariate Johnson-translation families — generated vectors match empirical marginal moments, and have cross-correlations close to sample correlations

Bivariate Bézier — limited extension to higher dimensions

8.5.5 Generating Random Vectors with Arbitrarily Specified Marginal Distributions and Correlations

Very general structure

- Arbitrary marginal distributions — need not be from same family; can even have some continuous and some discrete
- Arbitrary cross-correlation matrix — there are, however, constraints imposed by the set of marginal distributions on what cross correlations are theoretically feasible

Variate-generation method — *normal-to-anything* (NORTA)

- Transform a generated multivariate normal random vector (which is easy to generate) to get the desired marginals and cross-correlation matrix
- $F_1, F_2, ..., F_d$ are the desired marginal cumulative distribution functions (d dimensions)
- $r_{ij}(\mathbf{X})$ = desired correlation between the generated X_i and X_j (i and j are the coordinates of the generated d-dimensional random vector $\mathbf{X}$)
- Generate multivariate normal vector $\mathbf{Z} = (Z_1, Z_2, ..., Z_d)^T$ with $Z_i \sim N(0, 1)$ and correlations $r_{ij}(\mathbf{Z}) = \text{Cor}(Z_i, Z_j)$ specified as discussed below
- For $i = 1, 2, ..., d$, set $X_i = F_i^{-1}(\Phi(Z_i))$, where Φ is the standard normal cumulative distribution function (CDF)
  - Since Z_i has CDF Φ, $\Phi(Z_i) \sim U(0, 1)$, so $X_i = F_i^{-1}(\Phi(Z_i))$ is the inverse-transform method of generation from F_i, but with a roundabout way of getting the U(0, 1) variate
  - Evaluation of Φ and possibly F_i^{-1} would have to be numerical
- Main task is to pre-compute the normal correlations $r_{ij}(\mathbf{Z})$, based on the desired output correlations $r_{ij}(\mathbf{X})$, so that after the Z_i's are transformed via Φ and then F_i^{-1}, the resultant X_i's will have the desired correlation structure
- This computation is done numerically via algorithms in the original Cario/Nelson paper

Despite the numerical work, NORTA is very attractive due to its complete generality

8.5.6 Generating Stochastic Processes

Need for auto-correlated input processes was demonstrated in examples in Chap. 6

AR, ARMA, ARIMA models can be generated directly from their definitions, which are constructive

Gamma processes — gamma-distributed marginal variates with an autocorrelation structure

- Includes exponential autoregressive (EAR) processes as a special case

TES (Transform-Expand-Sample) processes

- Flexible marginal distribution, approximate matching of autocorrelation structure empirical observation
- Generate sequence of U(0, 1)'s that are autocorrelated
- Transform via inverse transform to desired marginal distribution
- Correlation-structure matching proceeds via interactive software
- Has been applied to telecommunications models

ARTA (Autoregressive To Anything)

- Similar to NORTA (finite-dimensional) random-vector generation
- Want generated X_i to have (marginal) distribution F_i, specified autocorrelation structure
- Generate AR(p) base process with N(0, 1) marginals and autocorrelation specified so that $X_i = F_i^{-1}(\Phi(Z_i))$ will have the desired final autocorrelation structure
- Numerical method to find appropriate correlation structure of the base process

8.6 Generating Arrival Processes

Want to simulate a sequence of events occurring over time (e.g., arrivals of customers or jobs)

Event times: $t_1, t_2, t_3, \ldots$ governed by a specified stochastic process

For convenience in dynamic simulations, want recursive algorithms that generate t_i from t_{i-1}

8.6.1 Poisson Process

Rate = $\lambda > 0$

Inter-event times: $A_i = t_i - t_{i-1} \sim$ exponential with mean $1/\lambda$

Algorithm (recursive, get t_i from t_{i-1}):

1. Generate $U \sim U(0,1)$ independently

2. Return $t_i = t_{i-1} - (\ln U)/\lambda$

Note that $-(\ln U)/\lambda$ is the desired exponential variate with mean $1/\lambda$

Obviously generalized to any *renewal process* where the A_i's are arbitrary positive RVs

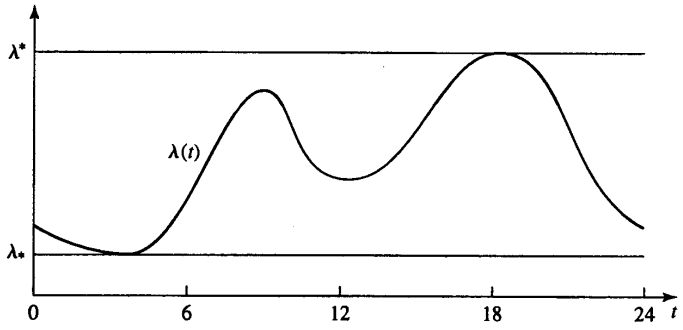
8.6.2 Nonstationary Poisson Process

When events (arrivals, accidents, etc.) occur at a varying rate over time

Noon rush, freeways, etc.

Ignoring nonstationarity can lead to serious modeling/design/analysis errors

$l(t)$ = mean rate of process (e.g., arrivals) at time t



Definition of process:

Let $N(a, b)$ be the number of events in the time interval $[a, b]$ $(a < b)$

Then $N(a, b) \sim$ Poisson with mean $\int_a^b l(t)\,dt$

Reasonable (but wrong) idea to generate:

Recursively, have an arrival at time t

Time of next arrival is t + expo (mean = $1/l(t)$)

Why this is wrong:

In above figure, suppose an arrival occurs at time 5, when $l(t)$ is low

Then $1/l(t)$ is large, making the exponential mean large (probably)

Likely to miss the first "rush hour"

A Correct Idea: *Thinning*

Let $\lambda^* = \max_t \lambda(t)$, the "peak" arrival rate; "thin" out arrivals at this rate

Generate "trial" arrivals at the (too-rapid) rate λ^*

For a "trial" arrival at time t, accept it as a "real" arrival with prob. $\lambda(t)/\lambda^*$

Algorithm (recursive—have a "real" arrival at time t_{i-1}, want to generate time t_i of the next "real" arrival):

1. Set $t = t_{i-1}$

2. Generate $U_1, U_2 \sim$ U(0,1) independently

3. Replace t by $t - (1/\lambda^*) \ln U_1$

4. If $U_2 \le \lambda(t)/\lambda^*$, set $t_i = t$ and stop; else go back to step 2 and go on

Proof that Thinning Correctly Generates a Nonstationary Poisson Process:

Verify directly that the number of retained events in any time interval $[a, b]$ is a Poisson RV with mean $\int_a^b \lambda(t)\,dt$

Condition on the number of rate λ^* trial events in $[a, b]$

Given this number, the trial events are U(a, b)

P(retain a trial point in $[a, b]$)

$$= \int_a^b P(\text{retain a trial point in } [a,b] \mid \text{trial point is at time } t) \underbrace{\frac{1}{b-a}}_{\substack{\text{Density of} \\ \text{trial-point} \\ \text{location}}} dt$$

$$= \int_a^b \frac{\lambda(t)}{\lambda^*} \frac{1}{b-a}\,dt$$

$$= \frac{\int_a^b \lambda(t)\,dt}{\lambda^*(b-a)} \qquad (\text{call this } p(a,b))$$

Let $N^*(a, b)$ = number of trial events in $[a, b]$

Thus, $N^*(a, b) \sim$ Poisson with mean $l^*(b - a)$

Also, $N(a, b) \le N^*(a, b)$, clearly

Then

$$P\big(N(a,b) = n \mid N^*(a,b) = k\big) = \begin{cases} \binom{k}{n}[p(a,b)]^n [1 - p(a,b)]^{k-n} & \text{if } k \ge 1 \text{ and } 0 \le n \le k \\ 1 & \text{if } n = k = 0 \end{cases}$$

being the (binomial) probability of accepting n of the k trial points, each of which has probability $p(a, b)$ of being "accepted"

Thus, for $n > 1$ (must treat $n = 0$ case separately),

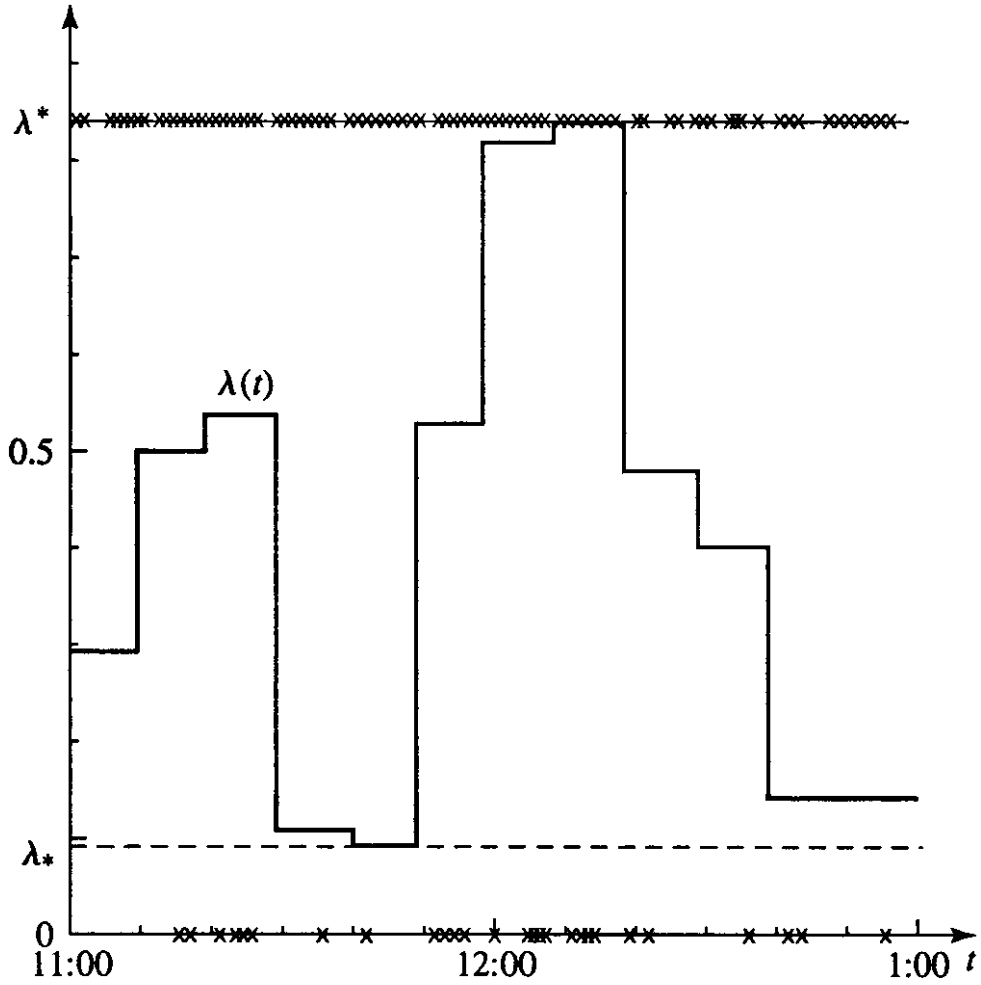
$$P(N(a,b) = n) = \sum_{k=n}^{\infty} P(N(a,b) = n \mid N^*(a,b) = k) P(N^*(a,b) = k)$$

$$= \sum_{k=n}^{\infty} \binom{k}{n} [p(a,b)]^n [1 - p(a,b)]^{k-n} \exp[-l^*(b-a)] \frac{[l^*(b-a)]^k}{k!}$$

$$\vdots \qquad \text{(several days of algebra)}$$

$$= \exp\left[-\int_a^b l(t)\,dt\right] \frac{\left[\int_a^b l(t)\,dt\right]^n}{n!} \qquad \text{as desired.}$$

Intuition: piecewise-constant $l(t)$, time from 11:00 a.m. to 1:00 p.m.



Another Algorithm to Generate NSPP:

Plot *cumulative rate function* $\Lambda(t) = \int_0^t l(y)\,dy$

Invert a rate-one stationary Poisson process (event times t_i') with respect to it

Algorithm (recursive):

1. Generate $U \sim U(0,1)$
2. Set $t_i' = t_{i-1}' - \ln U$
3. Return $t_i = \Lambda^{-1}(t_i')$

Compared to thinning:

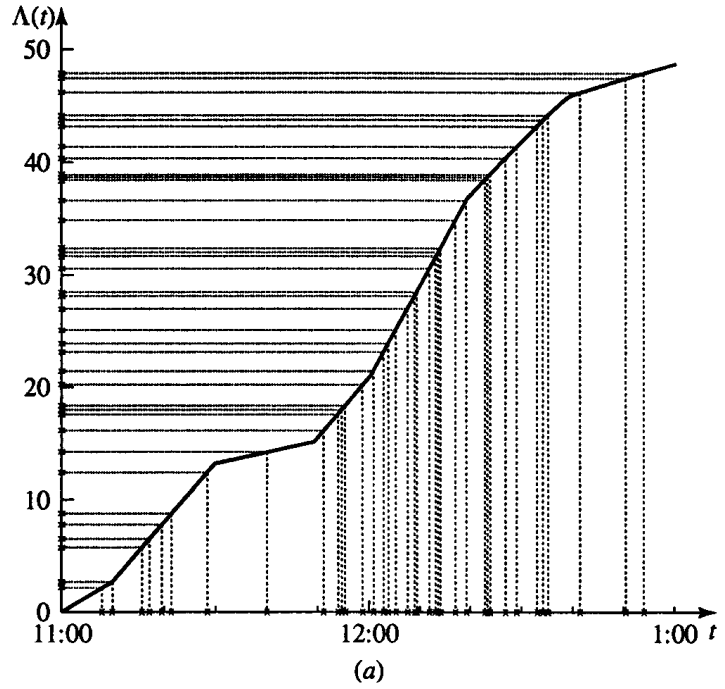
Have to compute and invert $\Lambda(t)$

Don't "waste" any trial arrivals

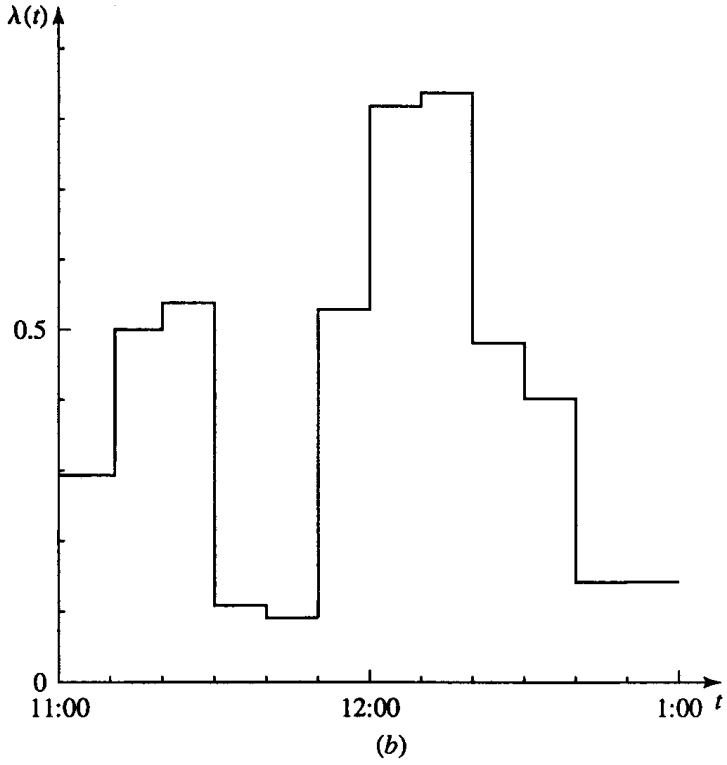
Faster if $l(t)$ has a few high spikes and is low elsewhere, and $\Lambda(t)$ is easy to invert

Corresponds to inverse-transform, good for variance reduction

Previous example: $l(t)$ piecewise constant, so $\Lambda(t)$ piecewise linear—easy to invert



(*a*)



(*b*)

Generating NSPP with piecewise-constant $l(t)$ in **SIMAN** (similar for other process-interaction languages)

Initialize global variables: **X(1)** = level of $l(t)$ on first piece
X(2) = level of $l(t)$ on second piece
etc.

Create a new "rate-changer" entity at the times when $l(t)$ jumps to a new rate
(Or, create a single such entity that cycles back when $l(t)$ jumps)

When the next rate changer arrives (or the single rate changer cycles back), increase global variable **J** by 1

Generate potential "customer arrivals" with interarrivals ~ exponential with mean $1/l^*$, and for each of these "trial" arrivals:

Generate $U \sim U(0,1)$
If $U \le$ **X(J)**$/l^*$, "accept" this as a "real" arrival and release entity into the model
Else dispose of this entity and wait for the next one

8.6.3 Batch Arrivals

At time t_i, have B_i events rather than 1; B_i a discrete RV on {1, 2, ...}

Assume B_i's are IID and independent of t_i's

Algorithm (get t_i and B_i from t_{i-1}):

1. Generate the next arrival time t_i from the Poisson process
2. Generate B_i independently
3. Return with the information that there are B_i events at time t_i

Could generalize:

Event times from some other process (renewal, nonstationary Poisson)

B_i's and t_i's correlated somehow