Discrete Event Simulation

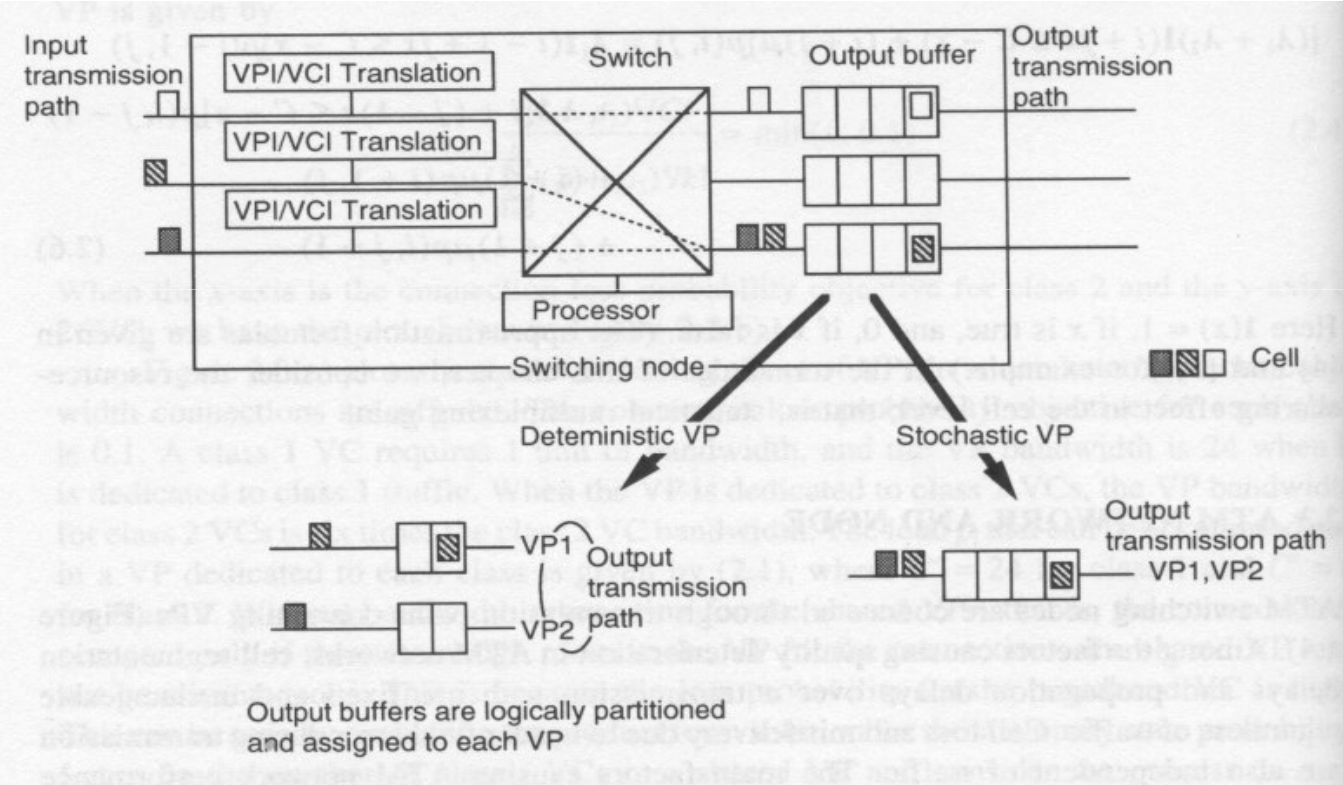
Prof Nelson Fonseca

State University of Campinas,
Brazil

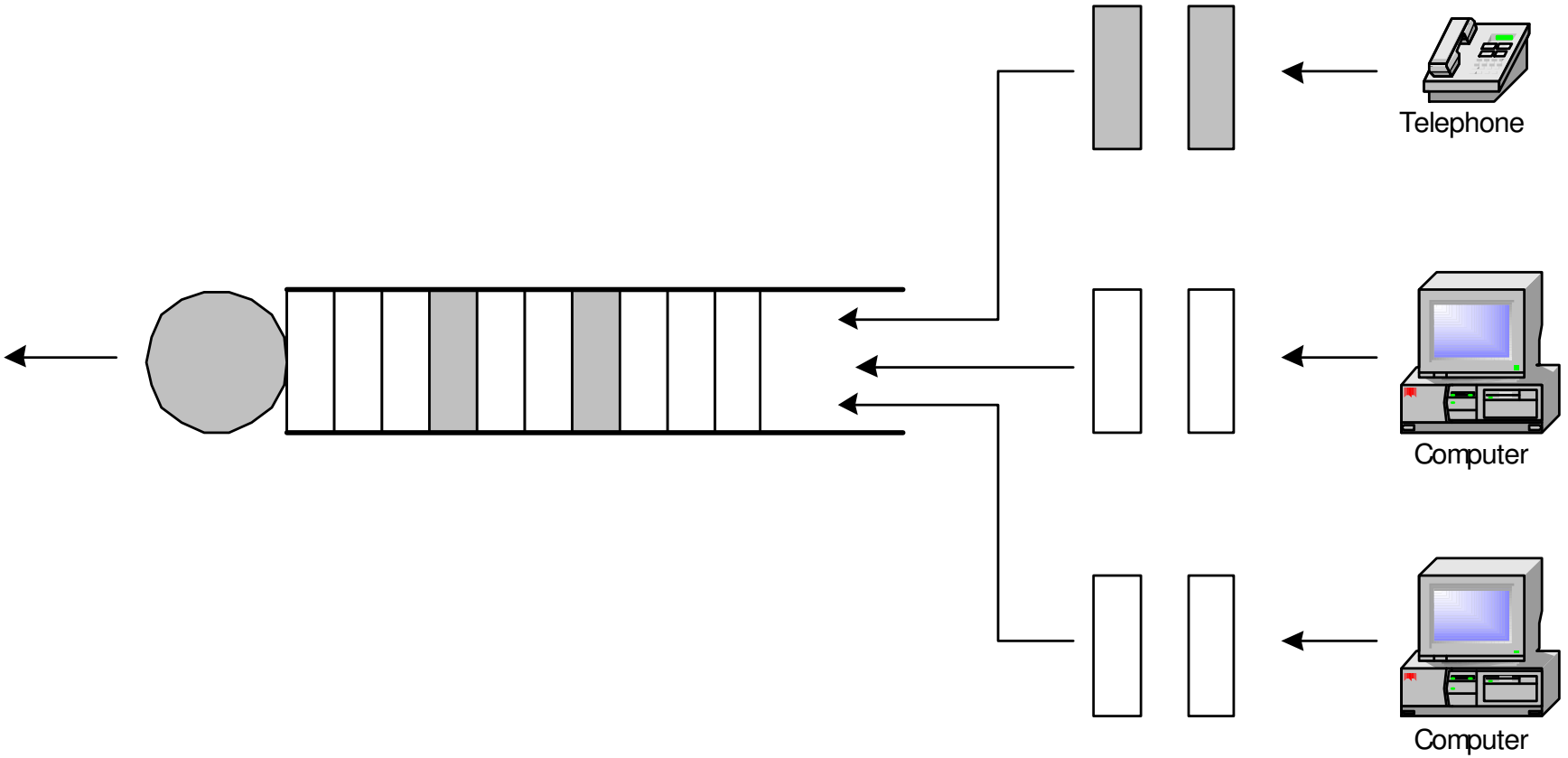
Simulation

- Emulation – hardware/firmware simulation
- Monte-carlo simulation – static simulation, typically for evaluation of numerical expressions
- Discrete event simulation – dynamic system, synthetic load
- Trace driven simulation – dynamic systems, traces of real data as input

Networks & Queues



Queuing



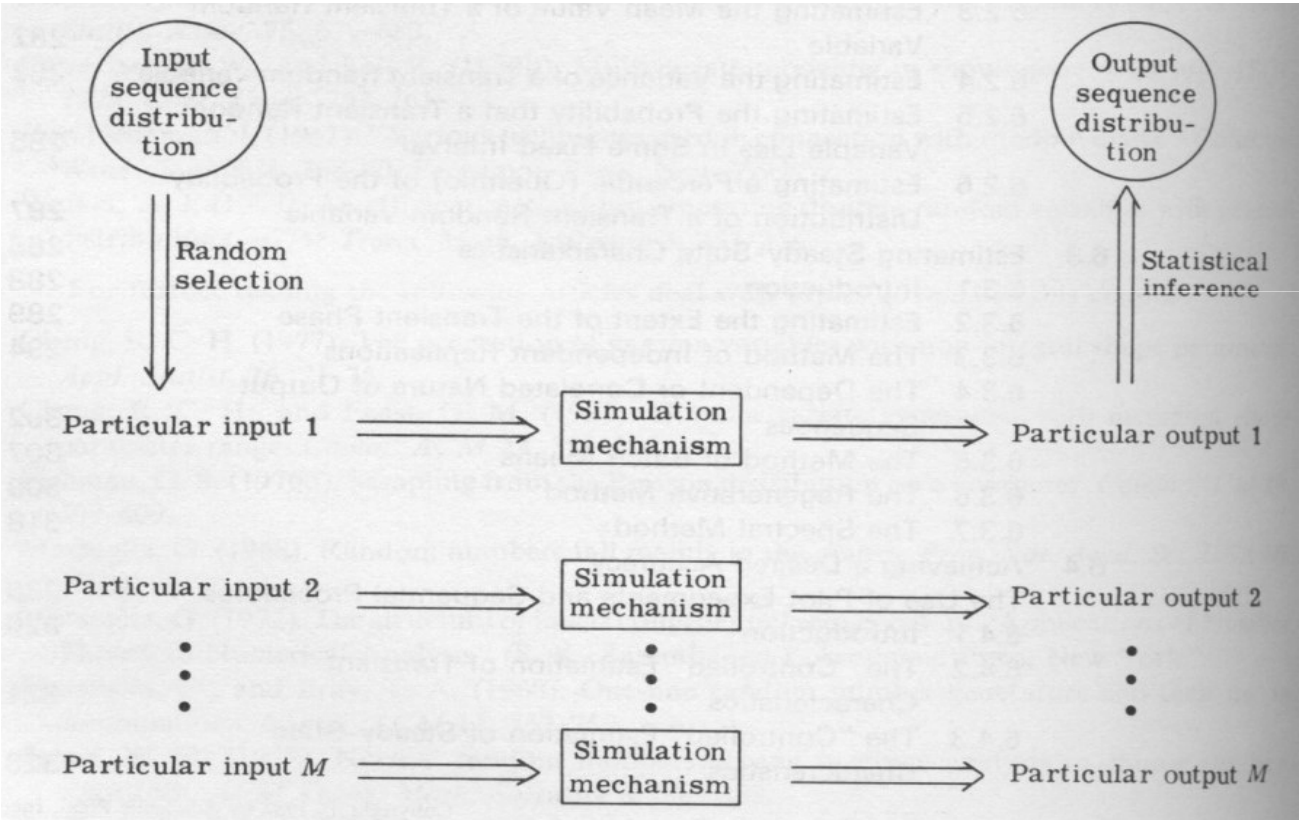
Measures of Interest

- Waiting time in the queue
- Waiting time in the system
- Queue length distribution
- Server utilization
- Overflow probability

Discrete Event Simulation

- Represents the stochastic nature of the system being modeled
- Driven by the occurrence of events
- Statistical experiment

Discrete Event Simulation



Events

- State Variables – Define the state of the system
  - Example: length of the queue
- Event: change in the system
  - Examples: arrival of a client, departure of a client

Discrete Events

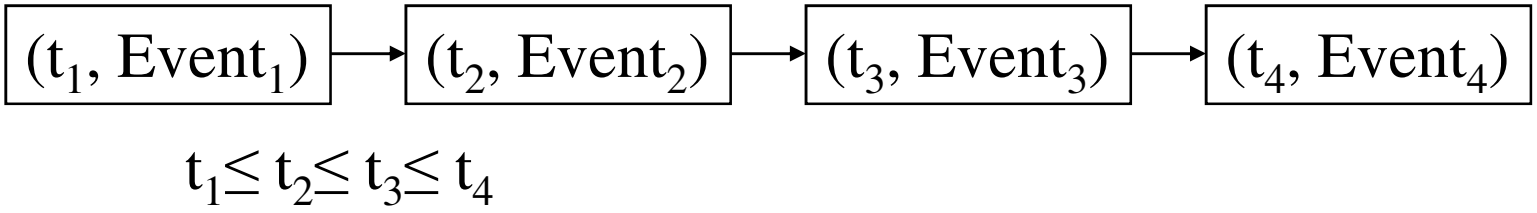
- Occurance of event – needs to reflect the changes in the system due to the occurance of that event

Discrete Events

- Primary event – an event which occurrence is scheduled at a certain time
- Conditional event → an event triggered by a certain condition becoming true

Discrete Event Simulation

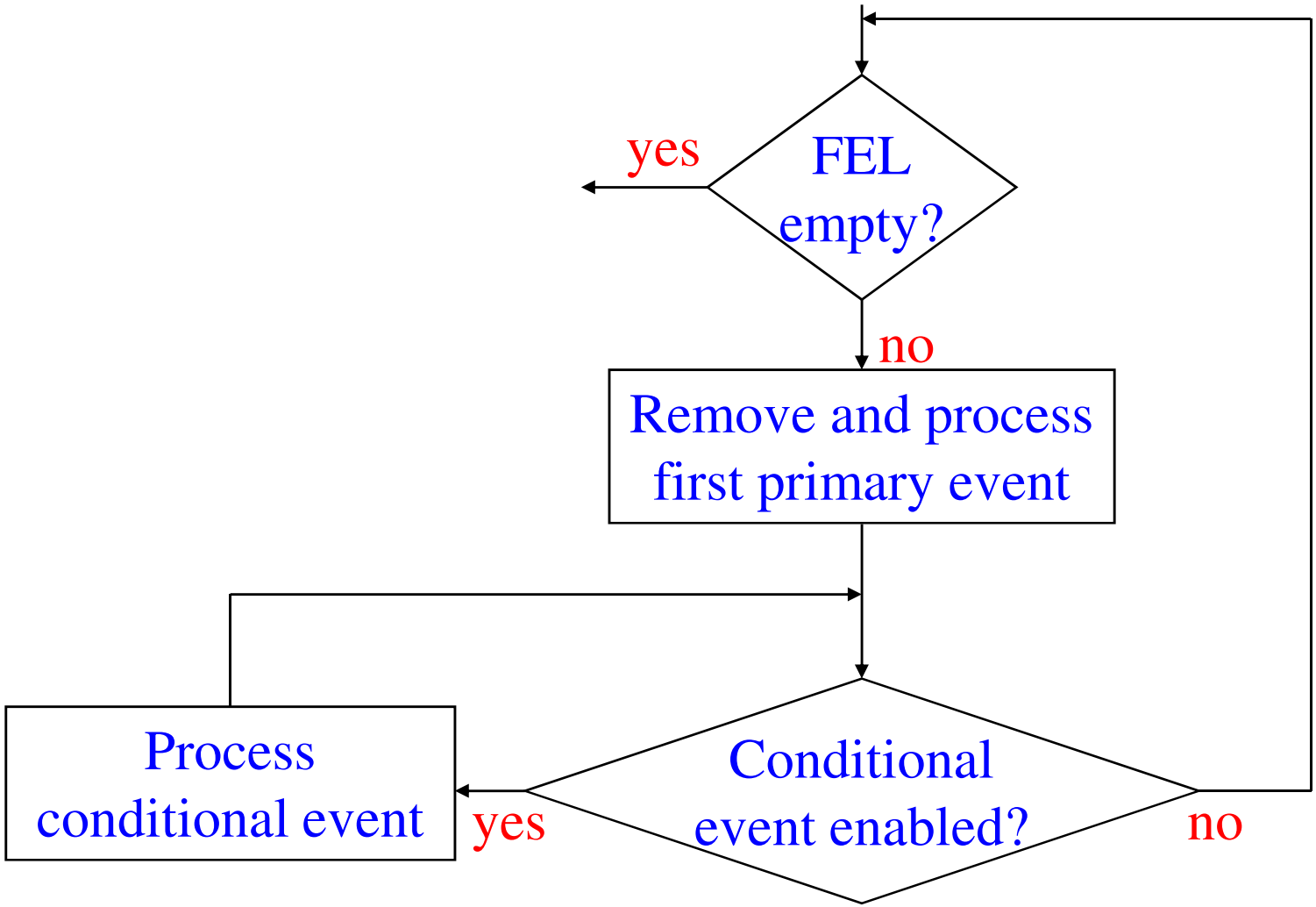
- The future event list (FEL) …
  - Controls the simulation
  - Contains all future events that are scheduled
  - Is ordered by increasing time of event notice
  - Contains only primary events
- Example FEL for some simulation time $t \leq T_1$:



Discrete Event Simulation

- Operations on the FEL:
  - Insert an event into FEL (at appropriate position)
  - Remove first event from FEL for processing
  - Delete an event from the FEL
- The FEL is thus usually stored as a linked list
- The simulator spends a lot of time processing the FEL
  - Efficiency is thus very important!

DES

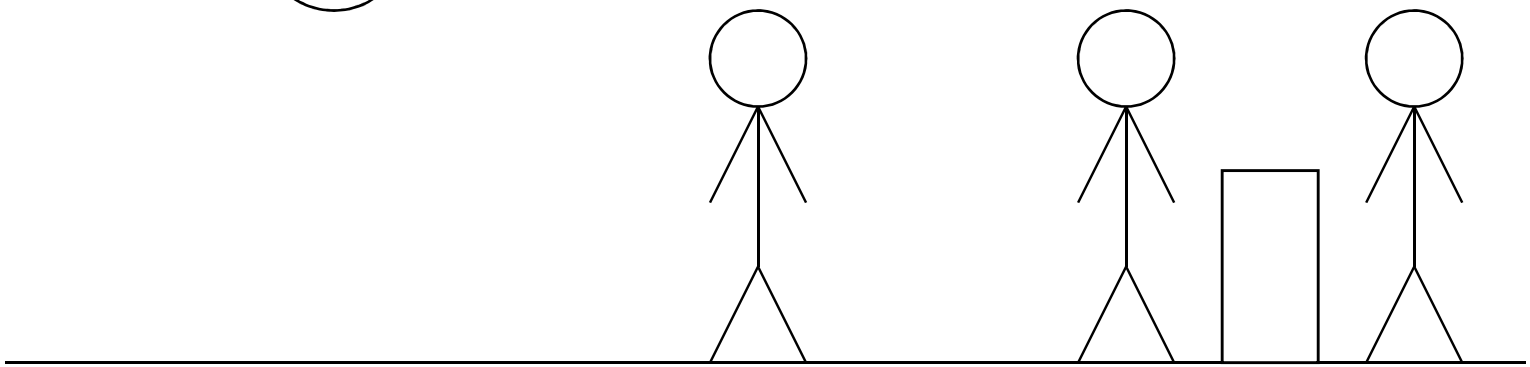

FEL
empty?
yes
no
Remove and process
first primary event
Process
conditional event
Conditional
event enabled?
yes
no

DES

- Simulation clock → register virtual time, not real time
- Can simulate one century in a second

DES

Simulation clock: t_2



$(t_2, \text{Arrival}) \rightarrow (t_3, \text{Service complete})$

Book Keeping

- Procedures that collect information (logs) about the dynamics of the simulated system to generate reports
- Can collect information at the occurance of every event or every fixed number of events

Simulating a Queue

Simulation clock: 15

Arrival interval	Customer arrives	Begin service	Service duration	Service complete
5	5	5	2	7
1	6	7	4	11
3	9	11	3	14
3	12	14	1	15

Computing Statistics

Average waiting time for a customer: (0+1+2+2)/4=1.25

Arrival interval	Customer arrives		Begin service	Service duration	Service complete
5	5	←0→	5	2	7
1	6	←1→	7	4	11
3	9	←2→	11	3	14
3	12	←2→	14	1	15

Computing Statistics

P(customer has to wait): =3/4=0.75

Arrival interval	Customer arrives		Begin service	Service duration	Service complete
5	5		5	2	7
1	6	←W→	7	4	11
3	9	←W→	11	3	14
3	12	←W→	14	1	15

Computing Statistics

P(Server busy): 10/15=0.66

Arrival interval	Customer arrives	Begin service	Service duration	Service complete
5	5	5	2	7
1	6	7	4	11
3	9	11	3	14
3	12	14	1	15

Computing Statistics

Average queue length: =(1*1+2*1+2*1)/15=0.33

Arrival interval		Customer arrives		Begin service	Service duration		Service complete
5		5	0→	5	2		7
1	0→	6	1→	7	4		11
3	0→	9	1→	11	3		14
3	0→	12	1→	14	1	0→	15

How To Generate A Random Variable?

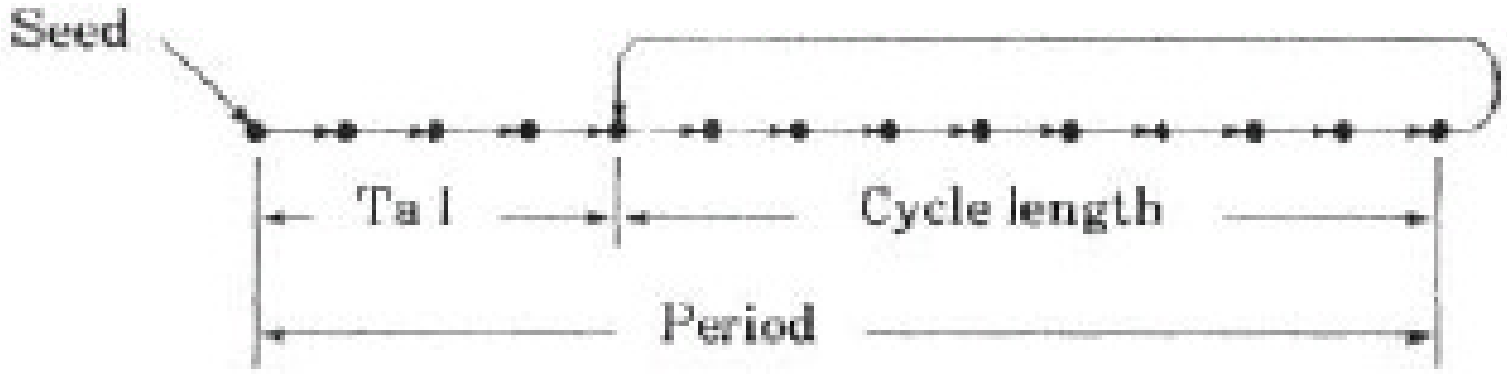
1, 3, 9, 27, 19, 26, 16, 17, 20, 29, 25, 13, 8,
24, 10, 30, 28, 22, 4, 12, 5, 15, 14, 11, 2, 6,
18, 23, 7, 21, 1, ...

How To Generate A Random Variable?

10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5
10, 3, 0, 1, 6, 15, 12, 13, 2, 11, 8, 9, 14, 7, 4, 5.

By dividing x's by 16:
0.6250, 0.1875, 0.0000, 0.0625, 0.3750, 0.9375,
0.7500, 0.8125, 0.1250, 0.6875, 0.5000, 0.5625,
0.8750, 0.4375, 0.2500, 0.3125, 0.6250, 0.1875,
0.0000, 0.0625, 0.3750, 0.9375, 0.7500, 0.8125,
0.1250, 0.6875, 0.5000, 0.5625, 0.8750, 0.4375,
0.2500, 0.3125.

How To Generate A Random Variable?



Random Number Generator

- Efficiently computable;
- The period (cycle length) should be large;
- The successful values should be independent and uniformly distributed;

How To Generate A Random Variable?

- Linear congruential method

- $X_{n+1} = (a\,X_n + b) \text{ modulo } m$

Random Variable Generation

- Let $X_0 = a = b = 7$, and $m = 10$
- This gives the pseudo-random sequence {7,6,9,0,7,6,9,0,…}
- What went wrong?
- The choice of the values is critical to the performance of the algorithm
- Also demonstrates that these methods always "get into a loop"

Linear Congruential Method

- *a*, *b* and *m* affect the period and autocorrelation
- Value depend on the size of memory word
- The modulus *m* should be large – the period can never be more than *m*
- For efficiency *m* should be power of 2
  - mod *m* can be obtained by truncation

Linear Congruential Method

- If *b* is non-zero, the maximum possible period *m* is obtained if and only if:
  - *m* and *b* are relatively prime, i.e., has non common factor rather than 1
  - Every prime number that is a factor of *m* should be a factor of *a-1*

Linear Congruential Method

- If m is a multiple of 4, $a-1$ should be a multiple of 4;

- All conditions are met if:
  - $m = 2^k$, $a = 4c + 1$
  - c, b and k are positive integer

Multiplicative Congruential Method

- b=0 period reduced, faster

$$X_n = a X_{n-1} \text{ modulo } m$$

- $m = 2^k$ – maximum period 2^{k-2}
- m prime number – with proper multiplier a maximum period $m-1$

Unix

- m= 248
- a = 0x5DEECE66D
- b = 0xB
- errand48(), lrand48(), nrand48(), mrand48(), jrand48()

Period

A generator that has the maximum possible period is called a full-period generator.

$$x_n = (2^{34} + 1)x_{n-1} + 1 \mod 2^{35} \quad (26.5)$$

$$x_n = (2^{18} + 1)x_{n-1} + 1 \mod 2^{35} \quad (26.6)$$

Lower autocorrelation between successive numbers are preferable.

Both generators have the same full period, but the first one has a correlation of 0.25 between x_{n-1} and x_n, whereas the second one has a negligible correlation of less than 2^{-18}

Seeds

- Initial value – right choice to maximize period length
- Depends on *a*, *b* and *m*

Seeds

$$x_n = (25173x_{n-1} + 13849) \bmod 2^{16}$$

Starting with a seed of $x_0 = 1$

n	x_n		
	Decimal		Binary
1	25173	01100010	01010101
2	12345	00110000	00111001
3	54509	11010100	11101101
4	27825	01101100	10110001
5	55493	11011000	11000101
6	25449	01100011	01101001
7	13277	00110011	11011101
8	53857	11010010	01100001
9	64565	11111100	00110101
10	1945	00000111	10011001
11	6093	00010111	11001101
12	24849	01100001	00010001
13	48293	10111100	10100101
14	52425	11001100	11001001
15	61629	11110000	10111101
16	18625	01001000	11000001
17	2581	00001010	00010101
18	25337	01100010	11111001
19	11949	00101110	10101101
20	47473	10111001	01110001

Multiple Streams of Random Number

- Avoid correlation of events
- Single queue: Different streams for arrival and service time
- Multiple queues: multiple streams
- Do not subdivide a stream
- Do not generate successive seeds to initially feed multiple streams

Multiple Streams of Random Number

- Use non-overlaping treams
- Reuse successive seeds in different replications
- Don't use random seeds

8. *Select* $\{u_0, u_{100,000}, u_{200,000}, \ldots\}$

$$x_n = a^n x_0 + \frac{c(a^n - 1)}{a - 1} \bmod m$$

Table of Seeds

$$x_n = 7^5 x_{n-1} \bmod (2^{31} - 1)$$

To be read along rows. x_{100000} is 46,831,694

$x_{100000i}$	$x_{100000(i+1)}$	$x_{100000(i+2)}$	$x_{100000(i+3)}$
1	46831694	1841581359	11931163244
727633698	933588178	804159733	1671059989
10612888424	1961692154	1227283347	1171034773
276090261	10667280669	209208115	554590007
721958466	1371272478	675466456	1095462486
1808217256	2095021727	1769349045	904914315
373135028	717419739	881155353	1489529863
1521138112	298370230	11402794300	1335826707
706178559	1103566601	884434366	962338209
1341315363	709314158	591449447	431918286
851767375	606179079	15008692001	1434868289
263032577	753643799	202794285	715851524

Random Number Generators

- Tausworthe Generator
- Extended Fibonacci Generator
- Combined generator

Random Variate Generation

- We have a sequence of pseudo-random uniform variates. How do we generate variates from different distributions?
- Random behavior can be programmed so that the random variables appear to have been drawn from a particular probability distribution
- If f(x) is the desired pdf, then consider the CDF

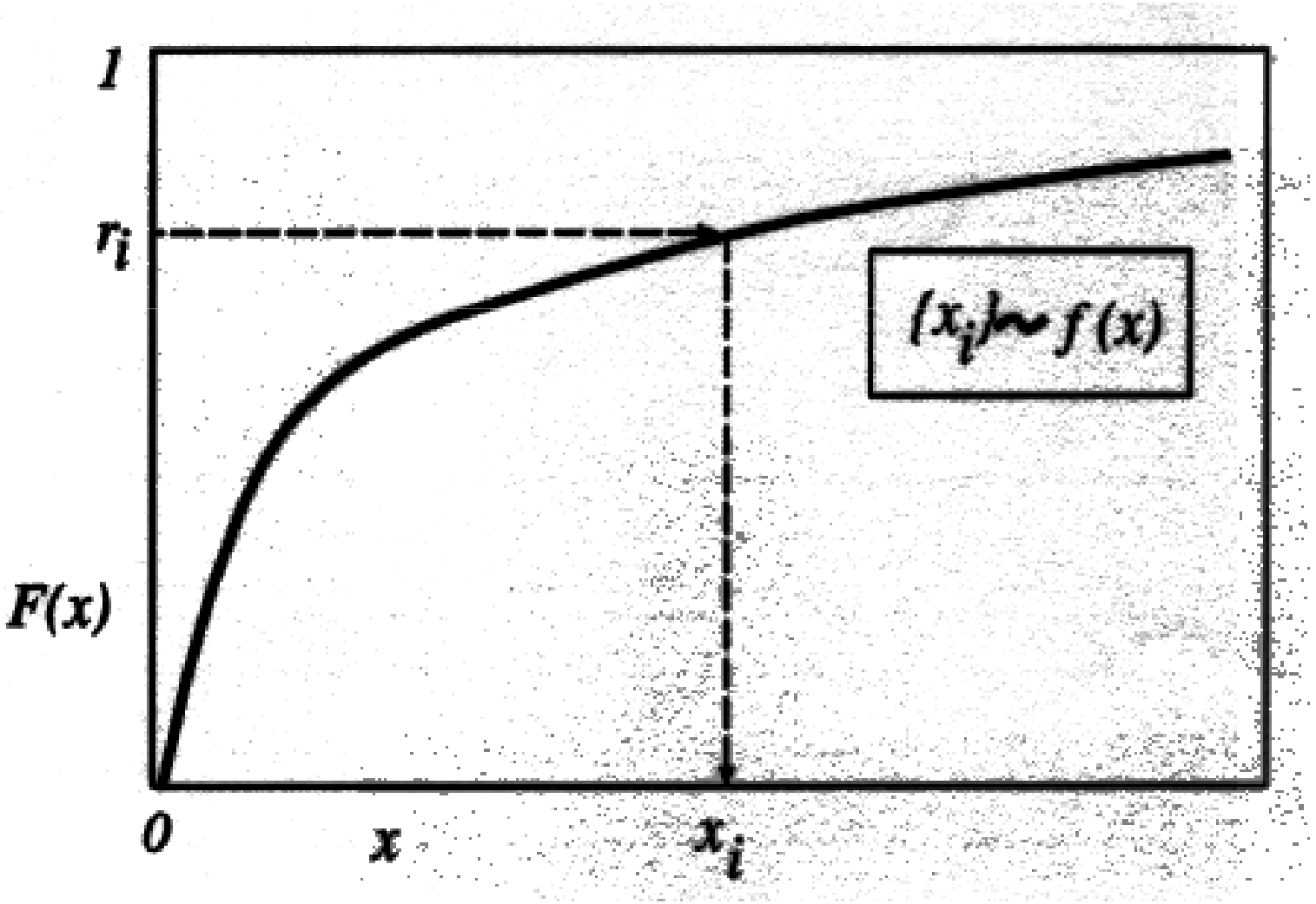
$$F_x(x) = \int_{-\infty}^{x} f(x)dx$$

- This is non-decreasing and lies between 0 and 1

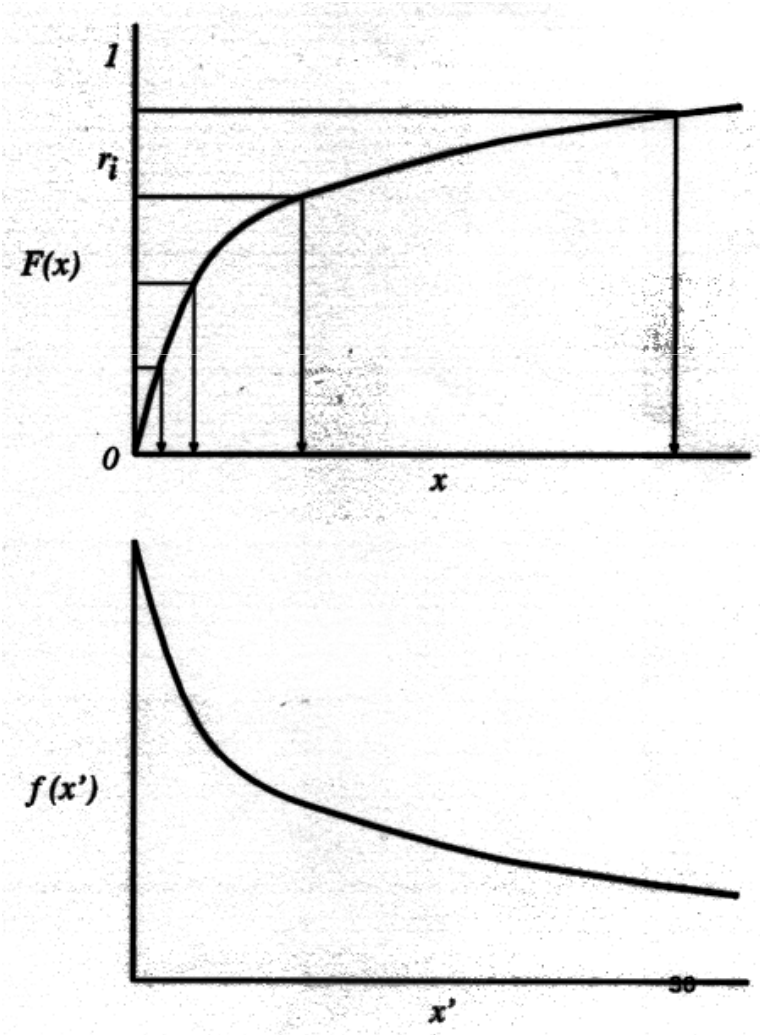
Random Variate Generation

- Given a sequence of random numbers r_i distributed over the same range (0,1)
- Let each value of r_i be a value of the function $F_x(x)$
- Then the corresponding value x_i is uniquely determined
- The sequence x_i is randomly distributed and has the probability density function $f(x)$

Random Variate Generation



Random Variate Generation



Method of Inverse

- For the exponential distribution

$$F_x(x_i) = 1 - e^{-\lambda x_i}$$

- For positive x_i
- Thus

$$r_i = 1 - e^{-\lambda x_i}$$

$$1 - r_i = e^{-\lambda x_i}$$

$$\ln(1 - r_i) = -\lambda x_i$$

$$x_i = -\frac{1}{\lambda}\ln(1 - r_i)$$

Method of Inverse

- Note that r_i has the same distribution as 1-r_i so we would in reality use

$$x_i = -\frac{1}{\lambda}\ln(r_i)$$

- Other random variates can be derivated in a similar fashion.

Method of Inverse

Example 28.1

- The packet sizes (trimodal) probabilities:

Size	Probability
64 Bytes	0.7
128 Bytes	0.1
512 Bytes	0.2

- The CDF for this distribution is:

$$F(x) = \begin{cases} 0.0 & 0 \le x < 64 \\ 0.7 & 64 \le x < 128 \\ 0.8 & 128 \le x < 512 \\ 1.0 & 512 \le x \end{cases}$$

- The inverse function is:

$$F^{-1}(u) = \begin{cases} 64 & 0 < u \le 0.7 \\ 128 & 0.7 < u \le 0.8 \\ 512 & 0.8 < u \le 1 \end{cases}$$

- Generate $u \sim U(0,1)$
  $u \le 0.7 \Rightarrow$ Size $= 64$
  $0.7 < u \le 0.8 \Rightarrow$ size $=128$
  $0.8 < u \Rightarrow$ size $= 512$

Method of Inverse

Applications of the Inverse-Transformation Technique

Distribution	CDF $F(x)$	Inverse
Exponential	$1 - e^{-x/a}$	$-a \ln(u)$
Extreme value	$1 - e^{-e^{\frac{x-a}{b}}}$	$a + b \ln \ln u$
Geometric	$1 - (1-p)^x$	$\left\lceil \frac{\ln(u)}{\ln(1-p)} \right\rceil$
Logistic	$1 - \frac{1}{1+e^{\frac{x-\mu}{b}}}$	$\mu - b \ln(\frac{1}{u} - 1)$
Pareto	$1 - x^{-a}$	$1/u^{1/a}$
Weibull	$1 - e^{(x/a)^b}$	$a(\ln u)^{1/b}$

Rejection-acceptance

Rejection

- Can be used if a pdf $g(x)$ exists such that $cg(x)$ majorizes the pdf $f(x)$
  $\Rightarrow cg(x) \geq f(x)\ \forall x$.
- Steps:
  1. Generate x with pdf $g(x)$.
  2. Generate y uniform on $[0, cg(x)]$.
  3. If $y \leq f(x)$, then output x and return. Otherwise, repeat from step 1.

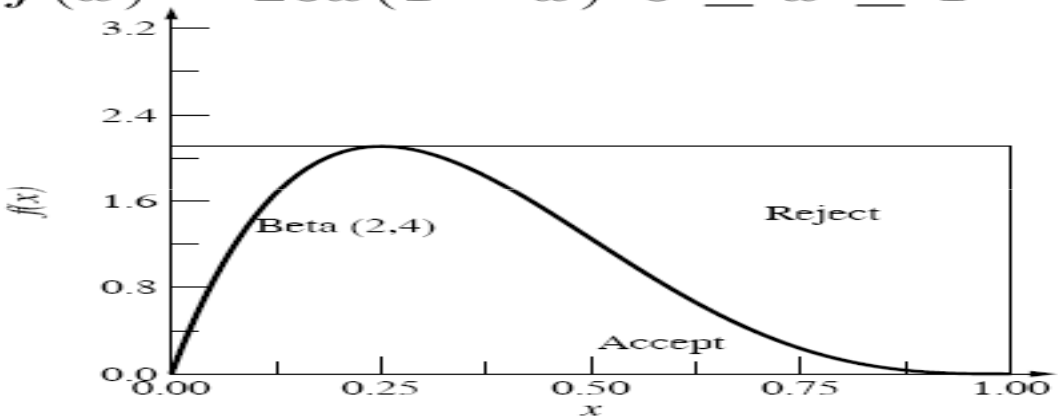
  $\Rightarrow$ Continue *rejecting* the random variates x and y until $y \leq f(x)$

Rejection-acceptance

Example 28.2

- Beta(2,4) density function:

$$f(x) = 20x(1-x)^3 \quad 0 \le x \le 1$$



- Bounded inside a rectangle of height 2.11 $\Rightarrow$ c=2.11 and $g(x) = 10 \le x \le 1$
- Steps:
  1. Generate x uniform on [0, 1].
  2. Generate y uniform on [0, 2.11].
  3. If $y \le 20x(1-x)^3$, then output x and return. Otherwise repeat from step 1.

Rejection-acceptance

Steps 1 and 2 generate a point (x, y) uniformly distributed over the rectangle. If the point falls above the beta pdf, then step 3 rejects x.

- Efficiency = how closely $cg(x)$ envelopes $f(x)$ Large area between $cg(x)$ and $f(x)$ $\Rightarrow$ Large percentage of (x, y) generated in steps 1 and 2 are rejected
- If generation of $g(x)$ is complex, this method may not be efficient.

Composition

Composition

- Can be used if CDF $F(x)$ = Weighted sum of n other CDFs.

$$F(x) = \sum_{i=1}^{n} p_i F_i(x)$$

  Here, $p_i \geq 0$, $\sum_{i=1}^{n} p_i = 1$,
  and F_i's are distribution functions.
- n CDFs are composed together to form the desired CDF
  Hence, the name of the technique.
- The desired CDF is decomposed into several other CDFs
  $\Rightarrow$ Also called decomposition.
- Can also be used if the pdf $f(x)$ is a weighted sum of n other pdfs:

$$f(x) = \sum_{i=1}^{n} p_i f_i(x)$$

Composition

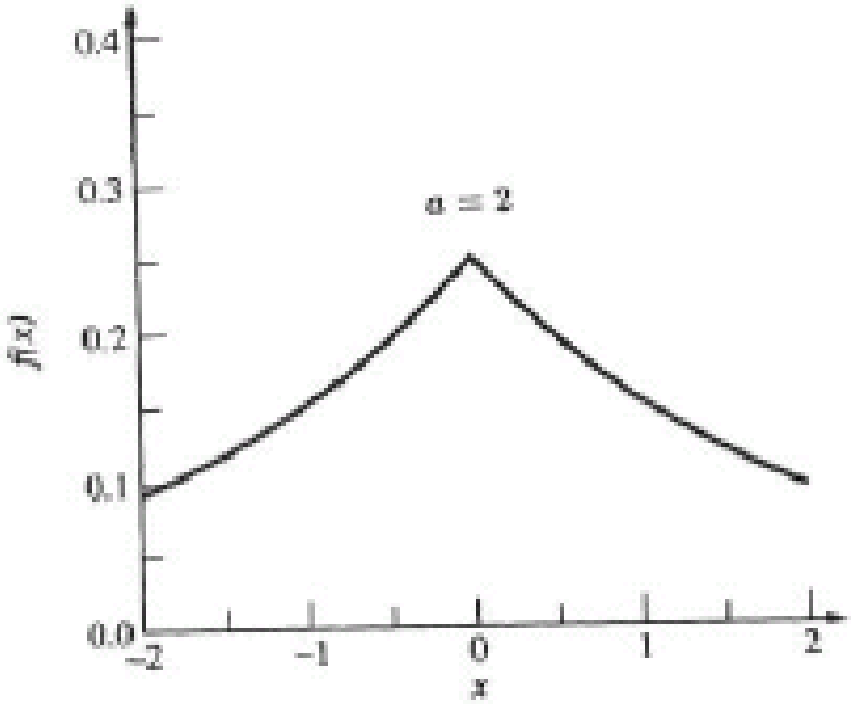


FIGURE 28.3 Laplace density function.

Convolution

- Random variable is given by the sun of independent random variable
- Examples: erlang, binomial, chi-square

Convolution

- Example: Erlang random variable is the sum of independent exponentially distributed random variables

$$f_x(x) = e^{-\lambda x} \lambda^k x^{k-1} / (k-1)!$$

Step1: Generate $U_1, U_2, \ldots U_k$ independent and uniformly distributed between 0 and 1

Step2: Compute $X = -\lambda^{-1} \ln(U_1 U_2 \ldots U_k)$

Convolution

- An Erlang-k variate is the sum of k exponential variates Thus, it can be obtained by generating k exponential variates and summing them.
- A binomial variate with parameters n and p is a sum of n Bernoulli variates with success probability p. Thus, a binomial variate can be obtained by generated n $U(0,1)$ random numbers and returning the number of random numbers that are less than p.
- The chi-square distribution with ν degrees of freedom is a sum of squares of ν unit normal $N(0,1)$ variates.
- The sum of two gamma variates with parameters (a,b_1) and (a,b_2) is a gamma variate with parameter (a,b_1+b_2). Thus, a gamma variate with a noninteger value of b parameter can be obtained by adding two gamma variates—one with integer b and the other with the fractional b.
- The sum of a large number of variates from any distribution has a normal distribution. This fact is used to generate normal variates by adding a suitable number of $U(0,1)$ variates.
- The sum of m geometric variates is a Pascal variate.
- The sum of two uniform variates has a triangular density.

Characterization

- Algorithm tailored to the variate by drawing from transformation, etc
- Example: Poisson can be generated by continuosly generating exponential distribution until exceeds a certain value

Characterization

- Pollar Method – exact for Normal distribution
- Generate U_1 and U_2 independent uniformly distributed
- Step1: $V_1 = 2U_1 - 1$ and $V_2 = 2U_2 - 1$
- Step 2: If $(S = V_1^2 + V_2^2) >= 1$
- reject U_1 and U_2 repeat Step1
- Otherwise $X_1 = V_1[(-2\ln S)/S]^{1/2}$

Random Variate Generation

TABLE 5.1

Generation of Nonuniform Random Numbers (Continuous Distributions)[a]

Exponential (5.7)	inverse transformation 5.1	relatively slow, easy to implement
	decomposition	fast, relatively hard to implement; see Learmonth and Lewis (1973a)
	comparison 5.19	suitable for small machines; see Ahrens and Dieter (1973b) for refinements
Hyperexponential (5.38)	decomposition 5.16	all methods for exponential random numbers available
Normal (5.25)	characterization 5.12	polar method; relatively slow, easy to implement
	decomposition	fast, relatively hard to implement; see Learmonth and Lewis (1973a) and Knuth (1969)
	comparison	fast, reasonably easy to implement; see Ahrens and Dieter (1973b)
Erlang (5.14)	composition 5.6	easy to implement
Gamma (5.14)	rejection–acceptance	shape parameter determines best methods; see Atkinson and Pearce (1976), Fishman (1976a), Marsaglia (1977), and Schmeiser and Lal (1980a)
Beta (5.32)	characterization 5.13	gamma sampling; arbitrary a, b; easy to implement
	characterization 5.14	uses rejection–acceptance; arbitrary a, b; easy to implement
	characterization 5.15	ordering; a, b integers; uses partial sorting
	characterization	uses rejection–acceptance, normal sampling, $a, b > 1$; fast for $a = b$; see Ahrens and Dieter (1973b)
	rejection–acceptance	arbitrary a, b; speed largely independent of parameter values; see Cheng (1979)
	rejection–acceptance	$a, b > 1$; uses exponential majorizing functions; see Schmeiser and Babu (1980)

[a] In Tables 5.1 and 5.2, citations in parentheses, e.g., (5.7), refer to equations in Section 5.2; citations without parentheses, e.g., 5.8, refer to algorithms of Section 5.2.

Random Variate Generation

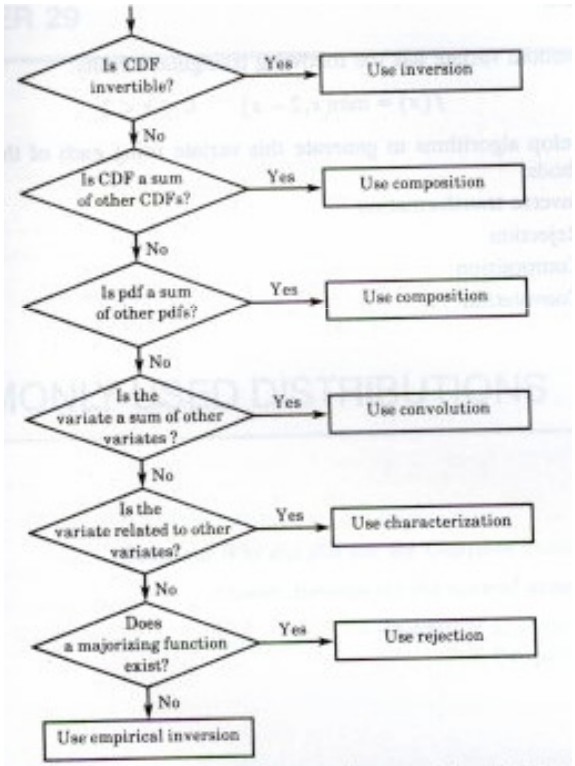


FIGURE 28.4 Finding a random-variate generation technique

Steady State Distribution

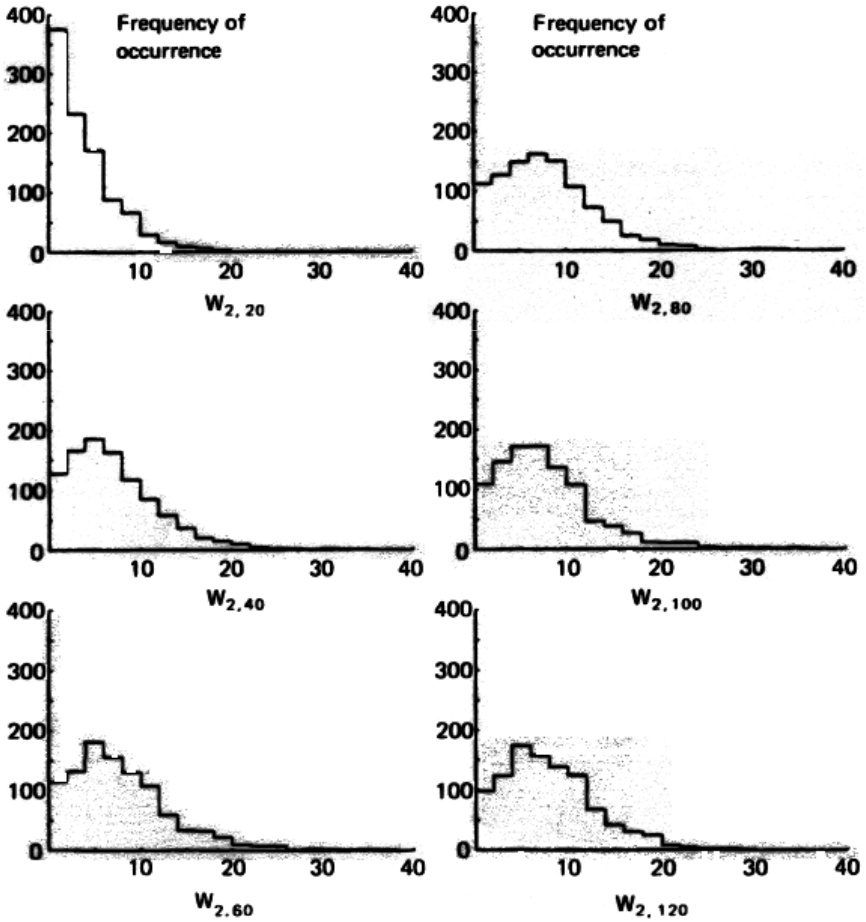


Fig. 6.7 Histograms of $W_{2,20}$, $W_{2,40}$, $W_{2,60}$, $W_{2,80}$, $W_{2,100}$, and $W_{2,120}$: all customers initially beginning terminal response times.

Transient Removal

- Identifying the end of transient state
- Long runs
- Proper initialization
- Truncations
- Initial data collection
- Moving average of independent replication
- Batch means

Transient Removal
Long Runs

- To neutralize the transient effects
- Waste of resources

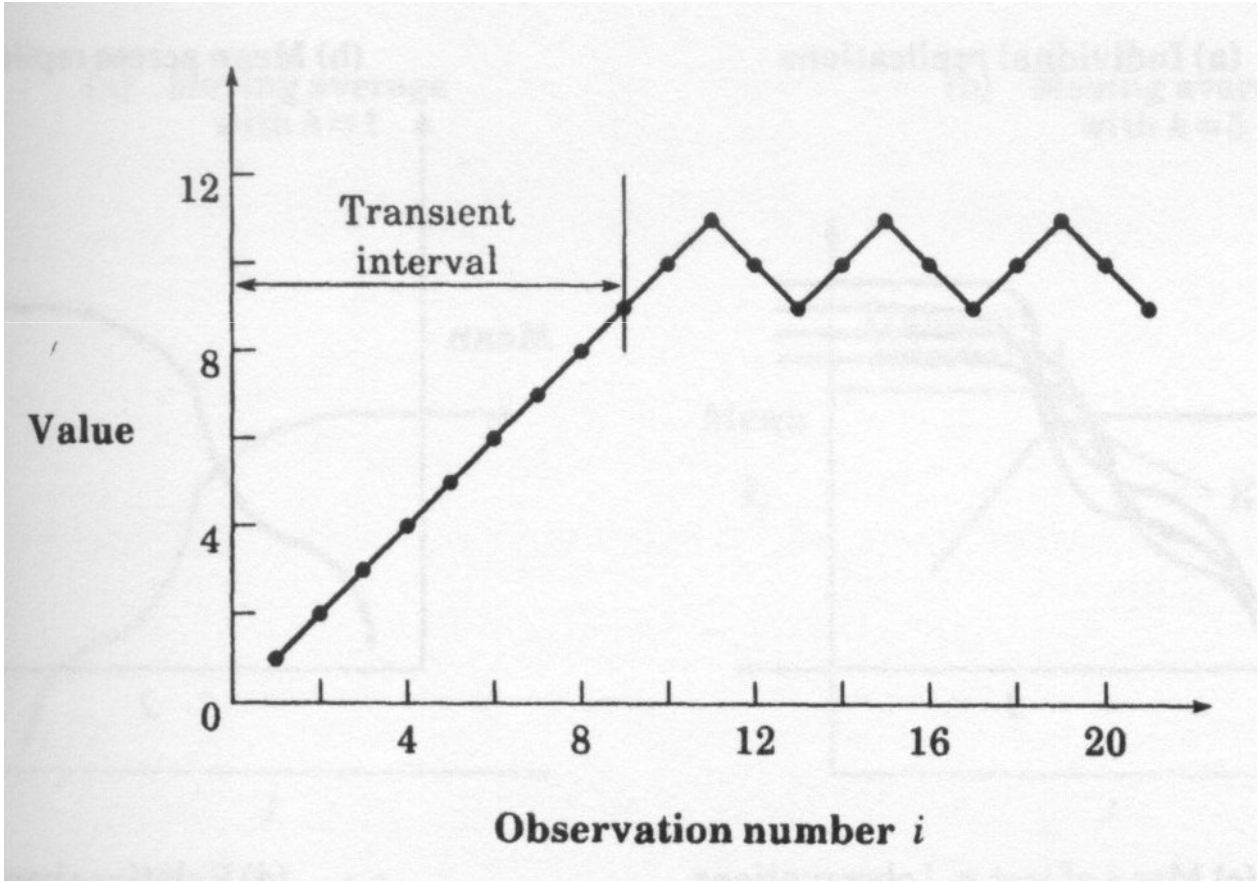
- Proper initialization – choice of a initial state that reduces transients effects

Transient Removal
Truncation

- Low variability in steady state
- Plots max-min n – j (j = 1, 2..) observations
- When (j+1)th observation is neither the minimum nor the maximum – transient ended

Example: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9,
10, 11, 10, 9, 10, 11, 10, 9, ...
At $l = 9$, Range = (9,11), next observation =

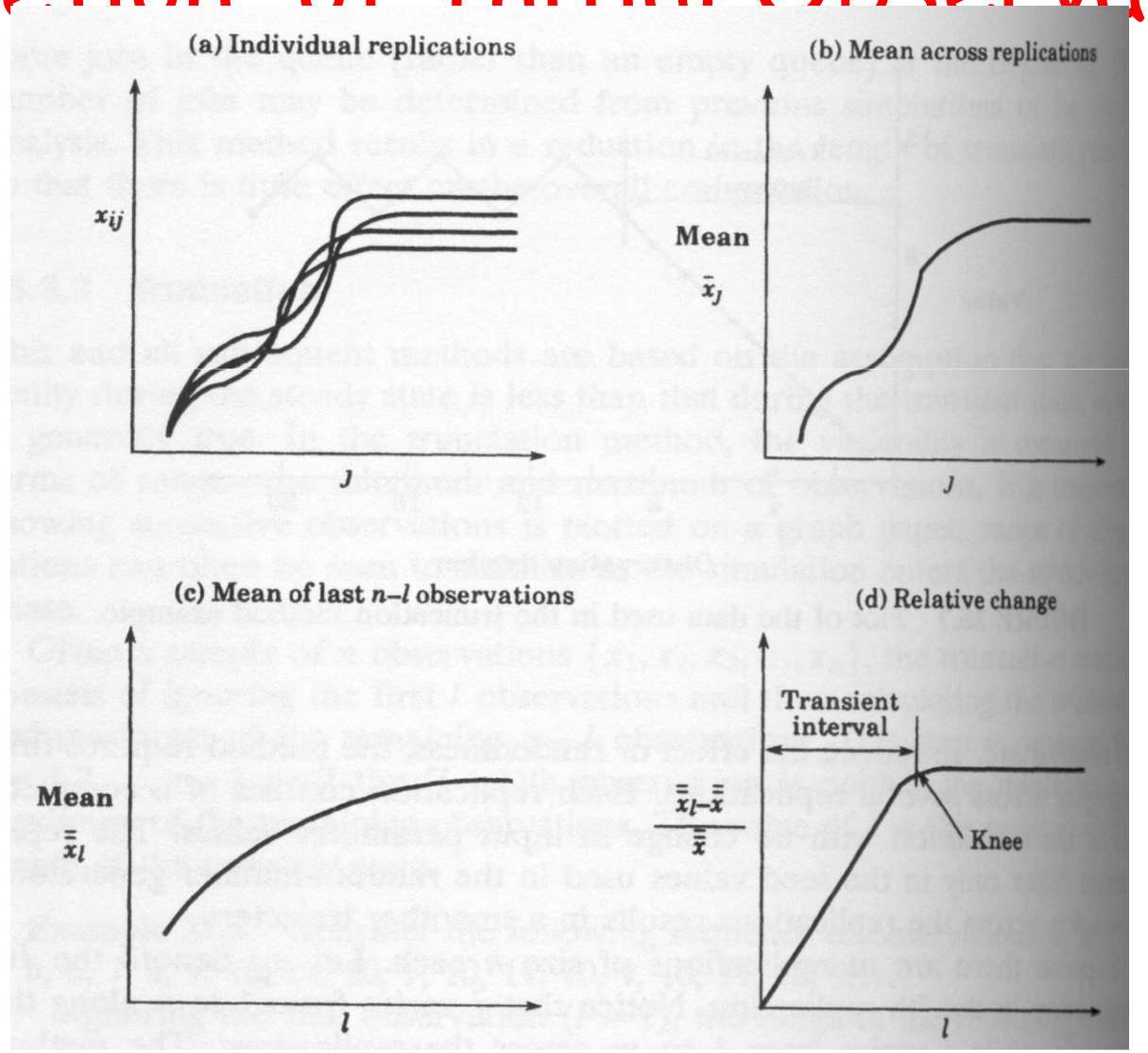
Truncation


12
8
4
0
Value
Transient
interval
4
8
12
16
20
Observation number i

Transient Removal
Deletion of Initial Observation

- No change on average value – steady state
- Produce several replications
- Compute the mean
- Delete j observation and check whether the sample mean was achieved. When found such j the duration of transient is determined

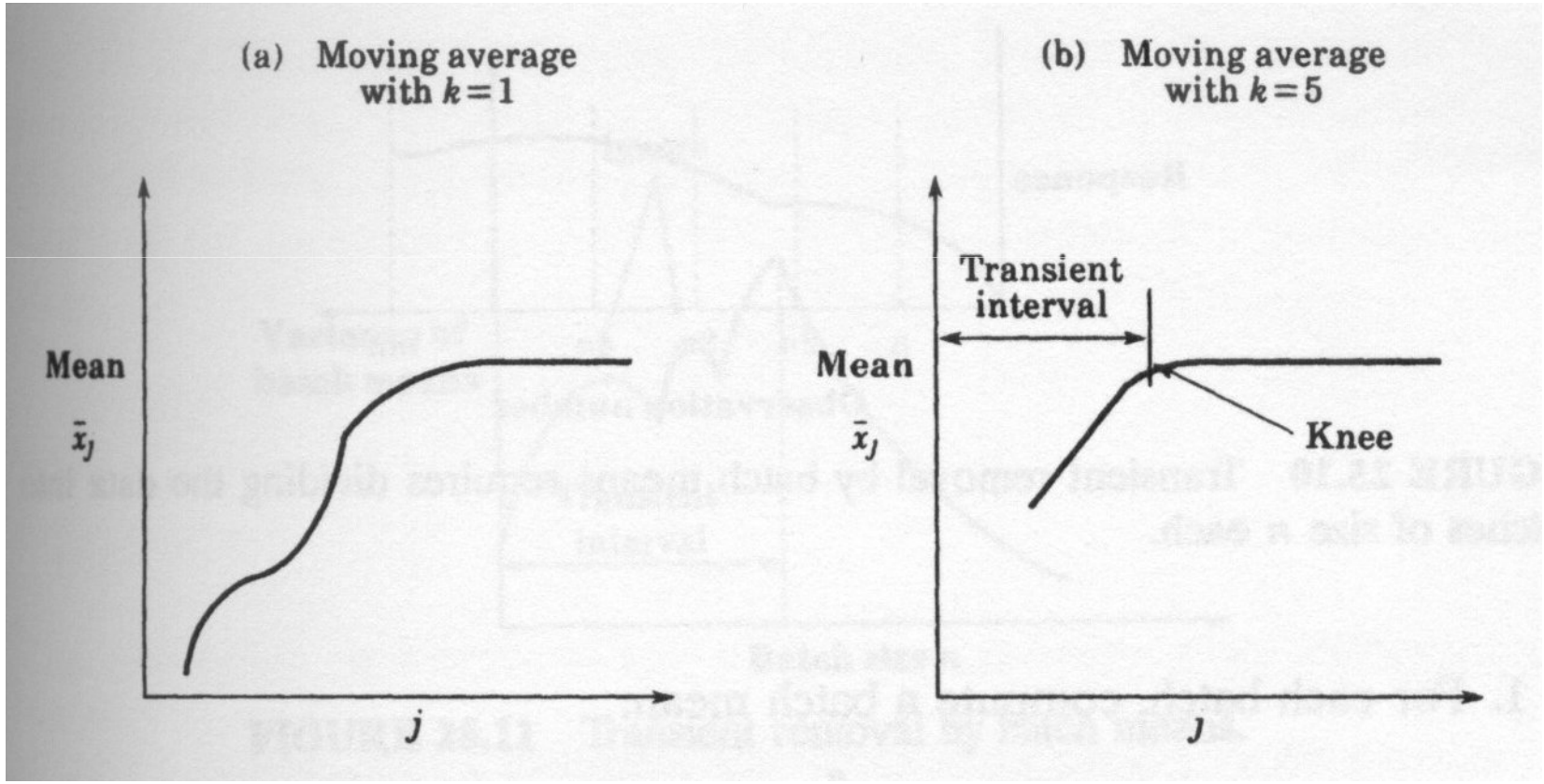
Transient Removal
Deletion of Initial Observation



Transient Removal
Moving average independent replication

- Similar to initial deletion method but the mean is computed over moving time interval instead of overall mean

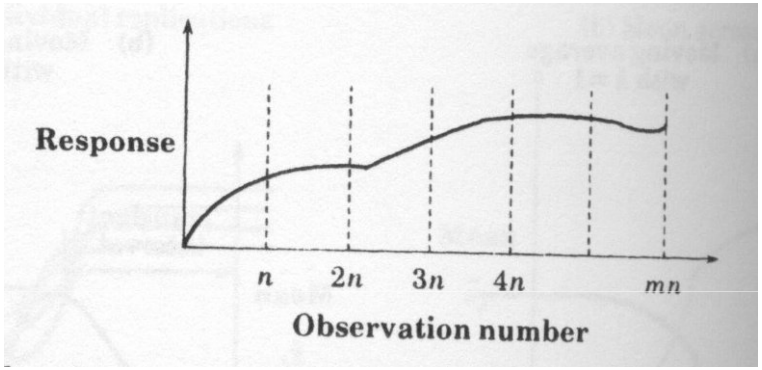
Transient Removal
Moving average independent replication

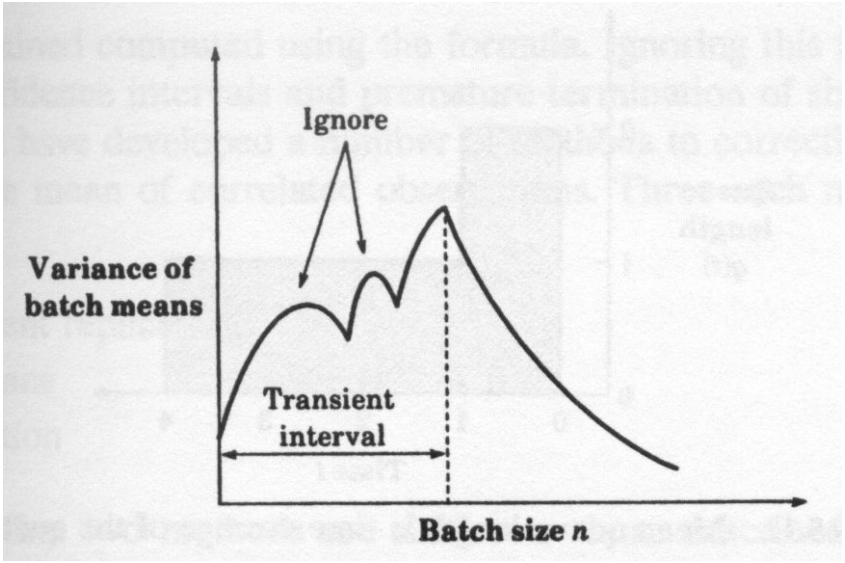


Transient Removal
Batch Mean

- Take a long simulation run
- Divide the observation into intervals
- Compute the mean of this intervals
- Try different sizes of batches
- When variance of batch mean starts to decrease – found the size of transient

Transient Removal
Batch Mean





Simulation: A Statistical Experiment

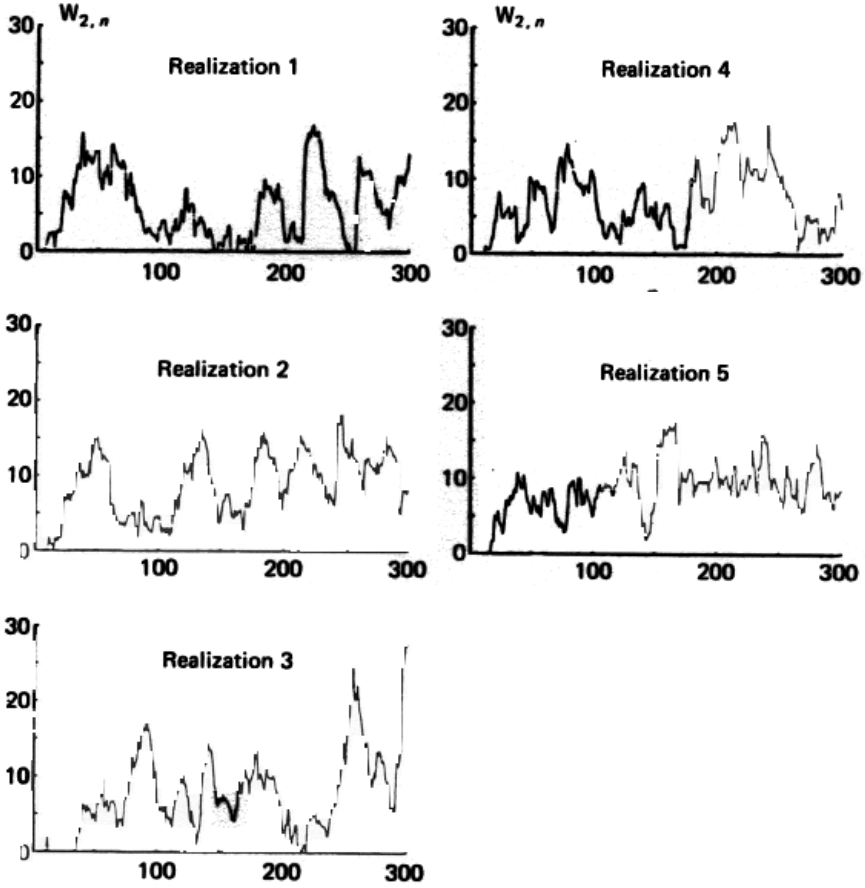


Fig. 6.3 Five sample realizations of the sequence $\{W_{2,n}\}$.

Simulation: A Statistical Experiment

- "Any estimate will be a random variable. Consequently a fixed, deterministic quantity must be estimated by a random quantity"
- "The experimenter must generate from the simulation not only an estimate but also enough information about the probability distribution so that reasonable confidence on the unknown value can be achieved"

Statistical Analysis of Results

- Given that each independent replication of a simulation experiment will yield a different outcome…
- To make a statement the about accuracy we have to estimate the distribution of the estimator
- Need to determine that the distribution becomes asymptotically centered around the true value

Statistical Analysis of Results

- Cannot be established with certainty in the case of a finite simulation

- The usual method used to estimate variability is to produce ***"confidence interval"*** estimates

Confidence Interval

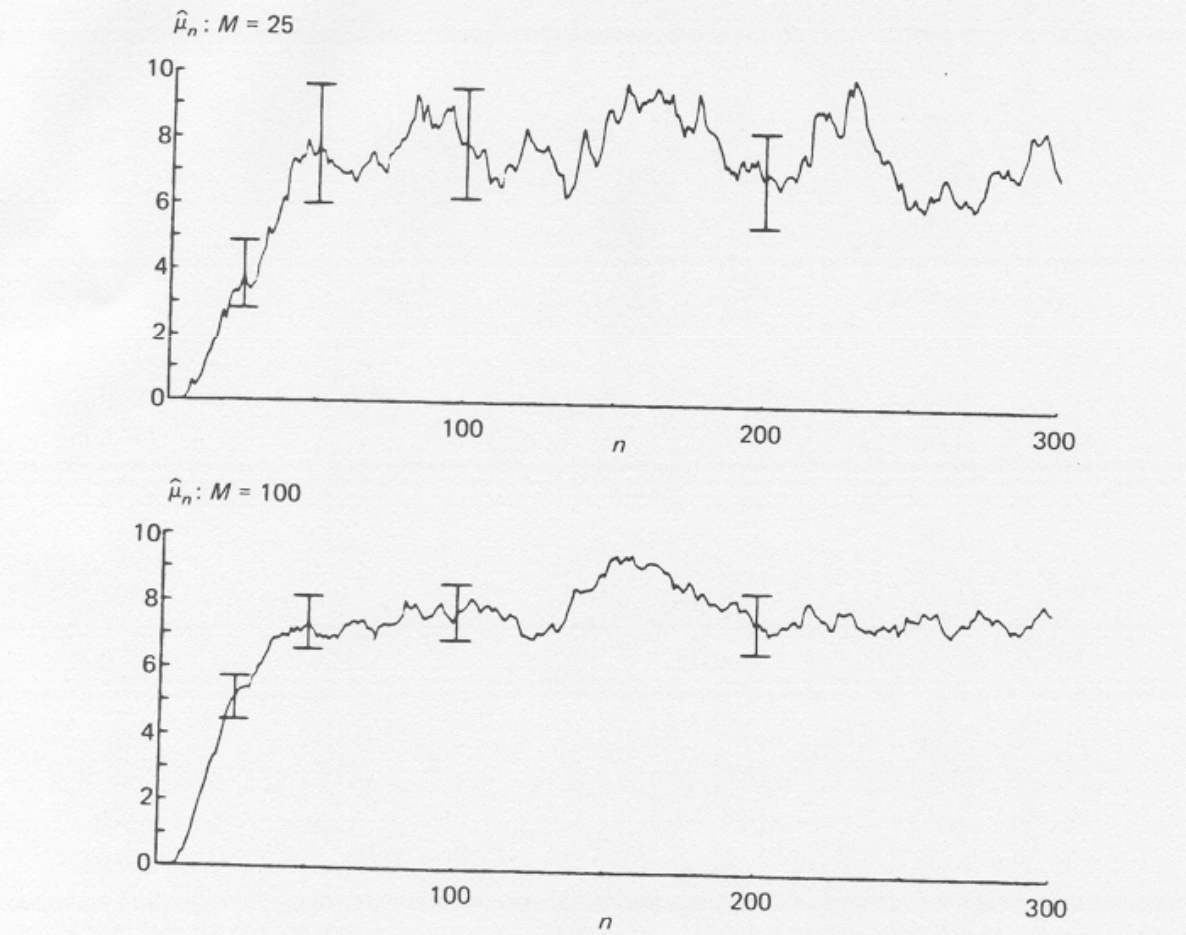
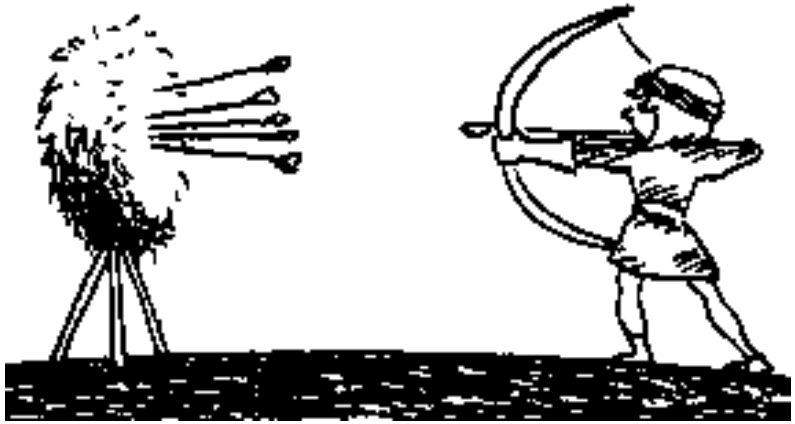


Fig. 6.11 Realizations of $\{\hat{\mu}_n\}$ obtained from 25 and 100 independent realizations of $\{W_{2,n}\}$.

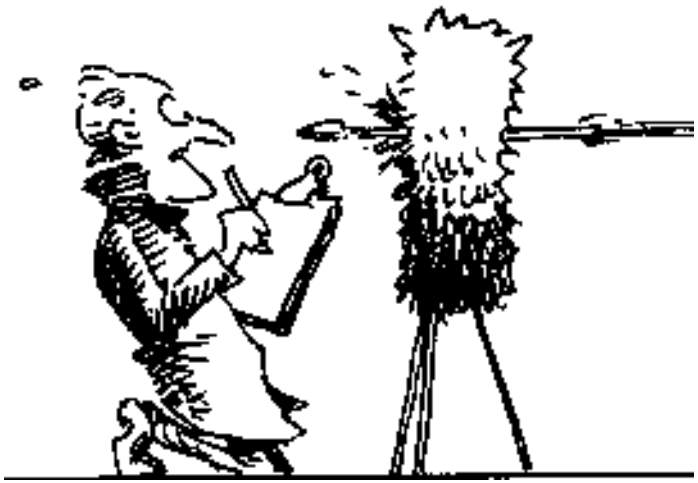
Confidence Intervals

- Given some point estimate p a we produce a confidence interval $(p-\delta, p+\delta)$
- The "true" value is estimated to be contained within the interval with some chosen probability, e.g. 0.9
- The value δ depends on the confidence level – the greater the confidence, the larger the value of δ

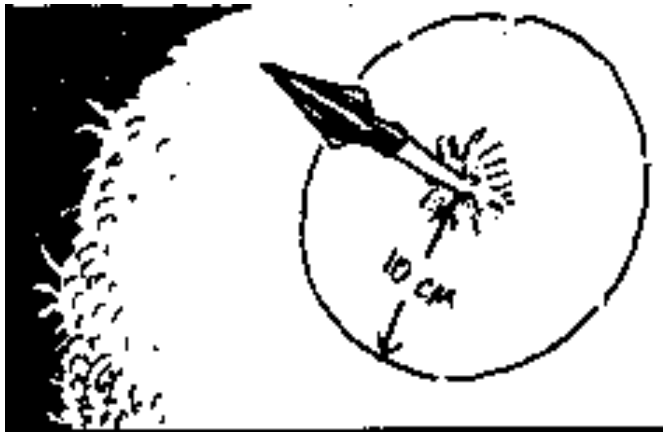
The central circle has a radiu of 20 cm, only 5% of the arrows are thrown out of the circle



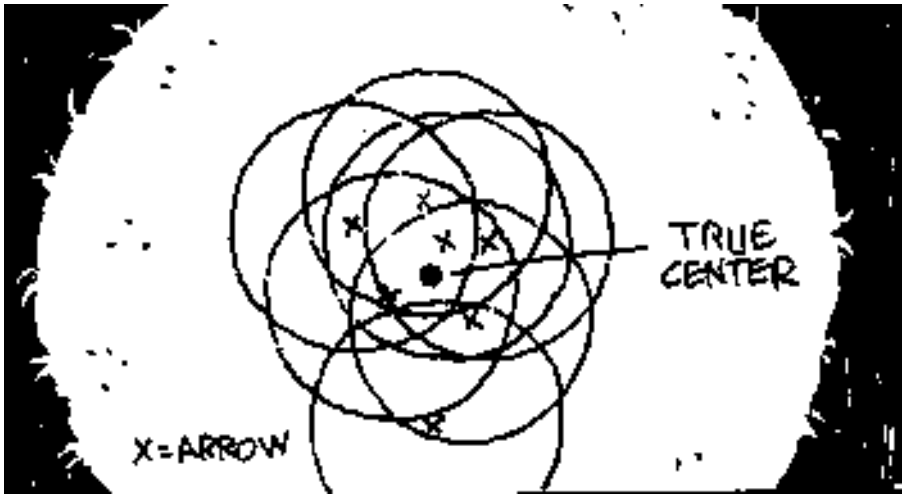
An observer does not know where the circle is centered



The observer draws a circle around each point
on the board made by the arrow.



After drawing several circle the position of the target point lays
In the intersection of all circles



Confidence Intervals

- Let $x_1, x_2, \ldots, x_n$ be the values of a random sample from a population determined by the random variable X
- Let the mean of X be $\mu = E(X)$ and variance σ^2
- Assume: either X is normally distributed or n is large
- Then: by the law of large numbers, $\bar{X} \approx$ normally distributed

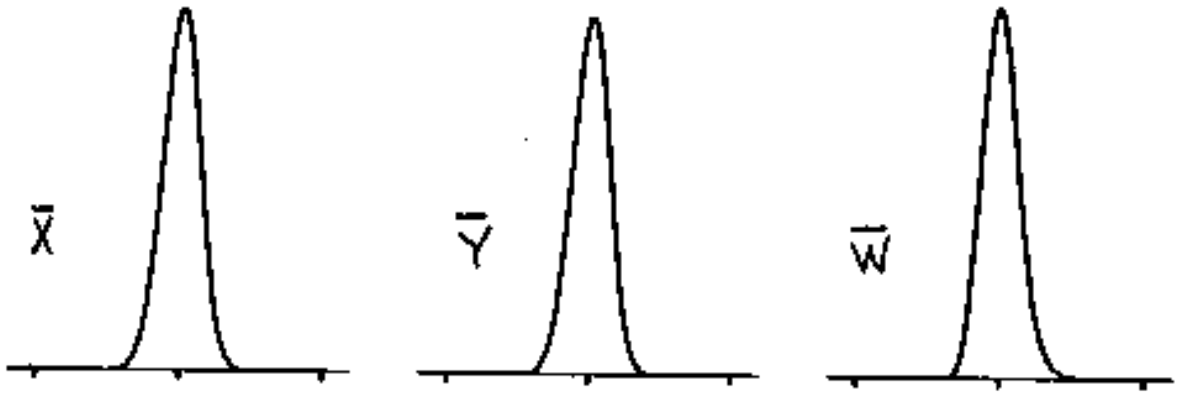
Central Limit Theorem

- The sum of a large number of independent observations from any distribution tends to have a normal distribution:

$$\bar{x} \sim N(\mu, \sigma / \sqrt{n})$$

Standard deviation

Central Limit Theorem



Confidence Intervals

- Then, given σ the 100(1-α)% confidence interval is given by

$$x \pm \delta$$

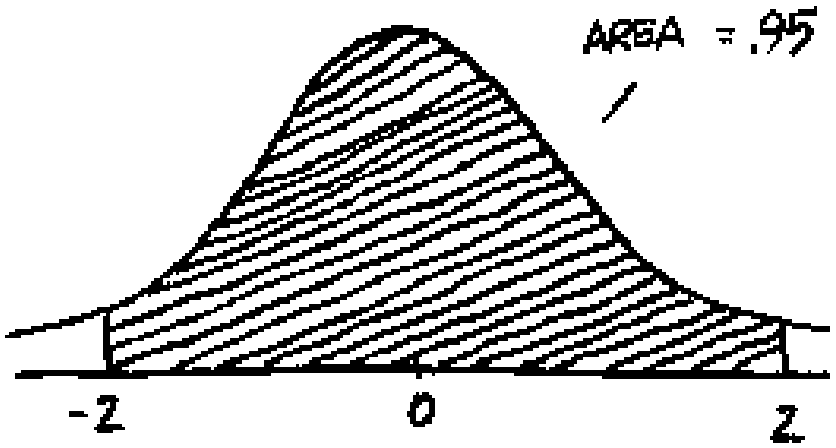
where

$$\delta = \frac{z_\alpha / 2^\sigma}{\sqrt{n}} \qquad (2)$$

- z_α is defined to be the largest value of z such that P(Z>z)=α and Z is the standard normal random variable

Confidence Interval

$$\left(\bar{x} - z_{1-\alpha/2}\, s/\sqrt{n}, \bar{x} + z_{1-\alpha/2}\, s/\sqrt{n}\right)$$

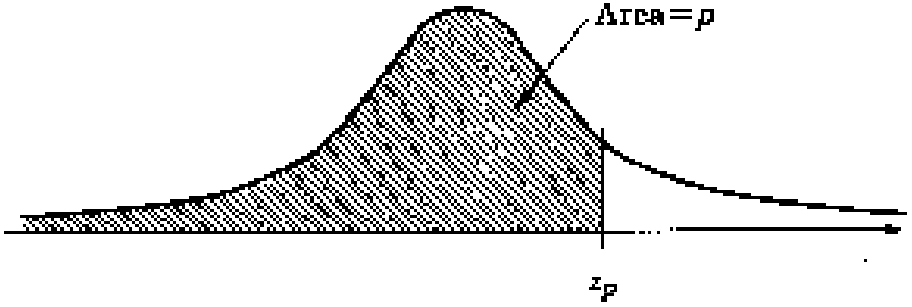


Confidence Intervals

- Can be taken from tables of the normal distribution

- For example, for a 95% confidence interval α=0.05 and $z_{\alpha/2}=z_{0.025}=1.96$

Confidence level = 95%, $\alpha = 0.05$ and $p = 1 - \alpha/2$



p	0.000	0.001	0.002	0.003	0.004	0.005	0.006	0.007	0.008	0.009
0.90	1.282	1.287	1.293	1.299	1.305	1.311	1.317	1.323	1.329	1.335
0.91	1.341	1.347	1.353	1.359	1.366	1.372	1.379	1.385	1.392	1.398
0.92	1.405	1.412	1.419	1.426	1.433	1.440	1.447	1.454	1.461	1.468
0.93	1.476	1.483	1.491	1.499	1.506	1.514	1.522	1.530	1.538	1.546
0.94	1.555	1.563	1.572	1.580	1.589	1.598	1.607	1.616	1.626	1.635
0.95	1.645	1.655	1.665	1.675	1.685	1.695	1.706	1.717	1.728	1.739
0.96	1.751	1.762	1.774	1.787	1.799	1.812	1.825	1.838	1.852	1.866
0.97	1.881	1.896	1.911	1.927	1.943	1.960	1.977	1.995	2.014	2.034
0.98	2.054	2.075	2.097	2.120	2.144	2.170	2.197	2.226	2.257	2.290

Example

- $\bar{x} = 3{,}90$; s=0,95 e n=32.
- Confidence level of 90%

$$= 3{,}90 \mp (1{,}645)(0{,}95)/\sqrt{32} = (3{,}62;4{,}17)$$

- Confidence level of 95%

$$= 3{,}90 \mp (1{,}960)(0{,}95)/\sqrt{32} = (3{,}57;4{,}23)$$

- Confidence level of 99%

$$= 3{,}90 \mp (2{,}576)(0{,}95)/\sqrt{32} = (3{,}46;4{,}33)$$

Using Student`s T

- When we know neither μ nor σ we can use the observed sample mean $\bar{x}$ and sample standard deviation s
- If n is large then we simply use s for σ in Equation (2).
- If n is small and X is normally distributed then we may use

$$\delta = \frac{t_{\alpha} / 2^{s}}{\sqrt{n}}$$

Using Student`s T

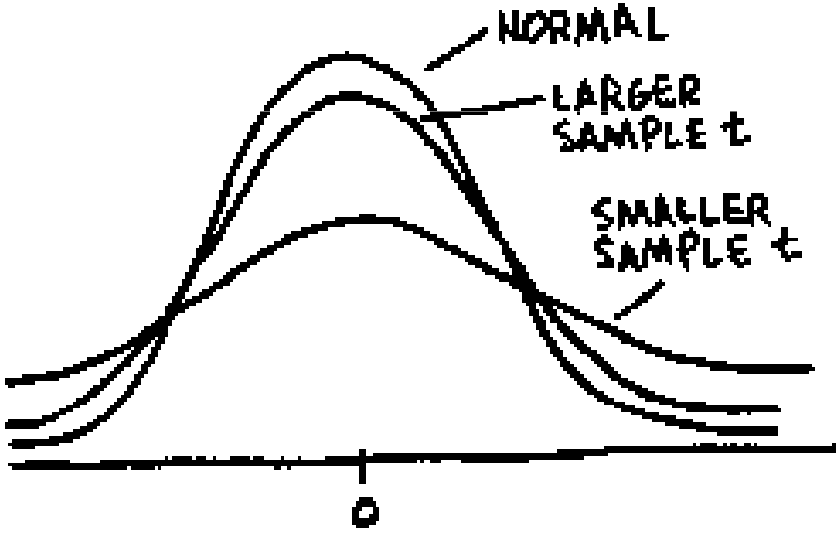
- The ratio $\frac{(\bar{x}-\mu)}{\left(S/\sqrt{n}\right)}$ for samples from normal populations follows a *t (n-1)* distribution

- $t_{\alpha/2}$ is defined by $P(T>t_{\alpha/2})=\alpha/2$

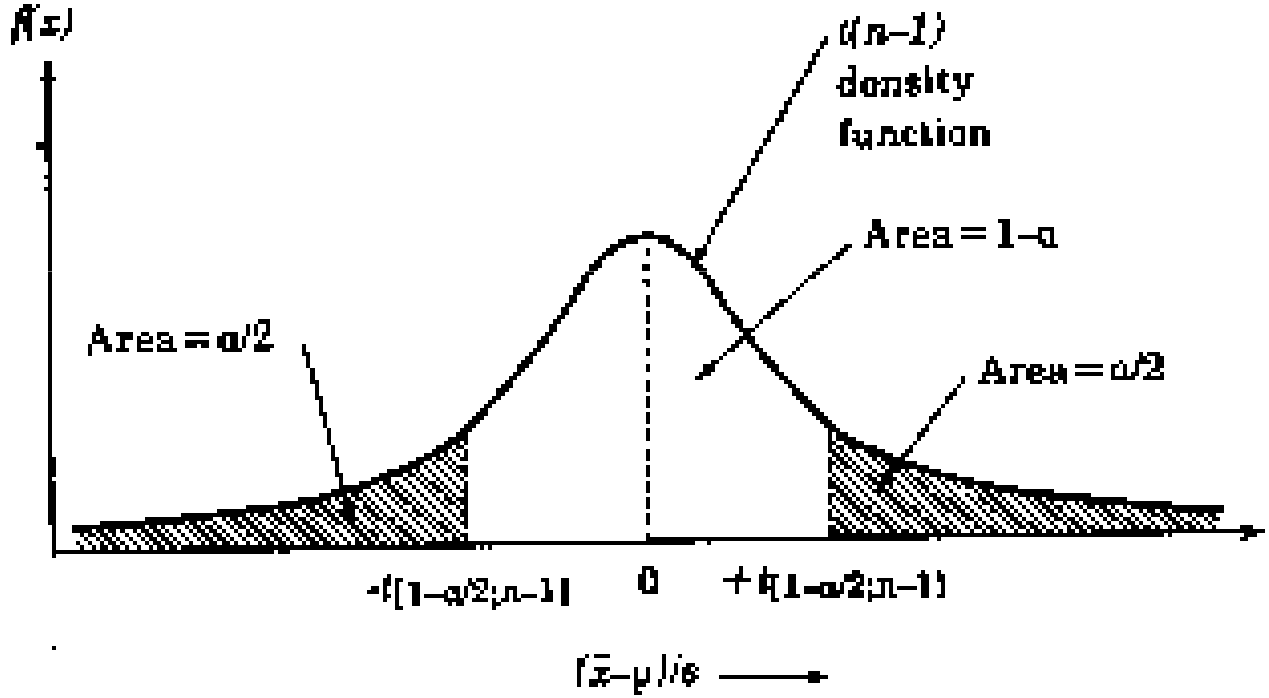
- *T* has a Student-*t* distribution with *n*-1 degrees of freedom

- This is the more frequently used formula in simulation models

t Student



t(n-1) Density Function



Confidence Interval

$$\hat{\mu} = \bar{X} = \frac{1}{M}\sum_{m=1}^{M} X_m$$

$$s^2 = \hat{\sigma}^2 = \frac{1}{M-1}\sum_{m=1}^{M}(X_m - \bar{X})^2 = \frac{1}{M-1}\sum_{m=1}^{M} X_m{}^2 - \frac{M}{M-1}(\bar{X})^2$$

$$\frac{\hat{\mu} - \mu}{s / M^{1/2}}$$

$$\text{Prob}\{\hat{\mu} - t_{M-1}(1-\alpha/2)s/M^{1/2} \le \mu \le \hat{\mu} + t_{M-1}(1-\alpha/2)s/M^{1/2}\} \approx 1-\alpha$$

Confidence Interval

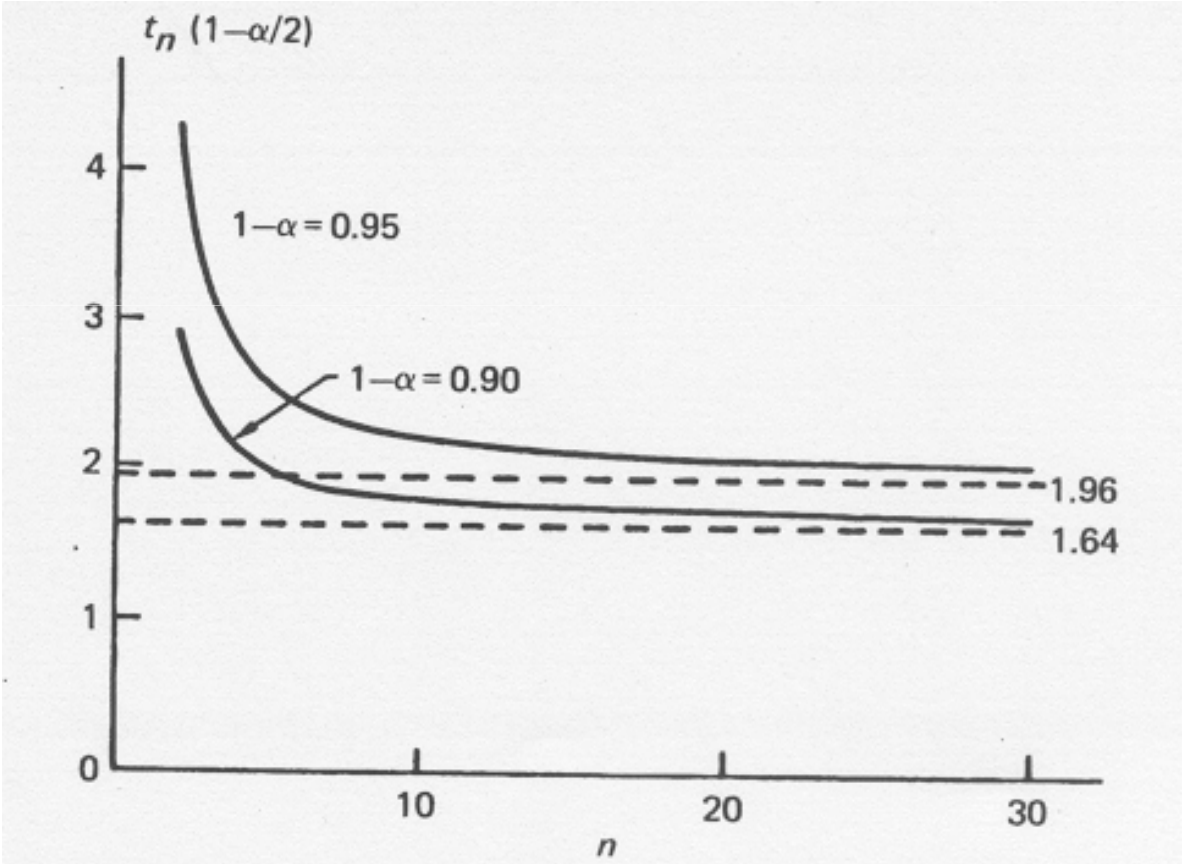


Fig. 6.9 Plots of $t_n(1 - \alpha/2)$ for $1 - \alpha = 0.9$ and $1 - \alpha = 0.95$.

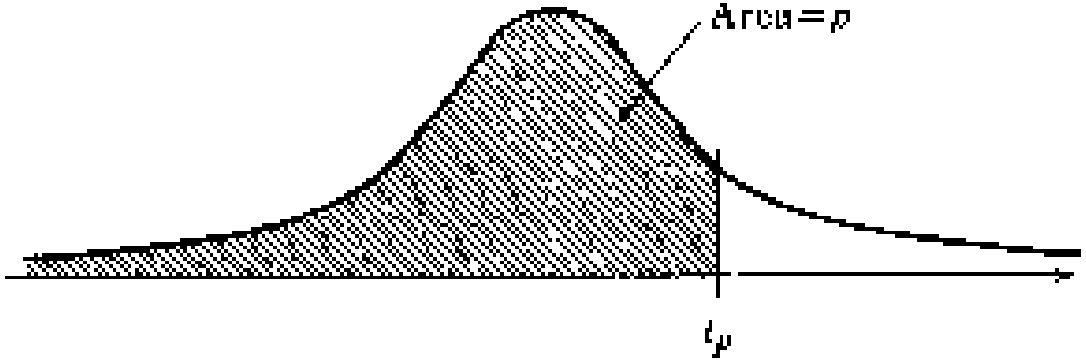


TABLE A.4 Quantiles of the t Distribution

	p							
n	0.6000	0.7000	0.8000	0.9000	0.9500	0.9750	0.9950	0.9995
1	0.325	0.727	1.377	3.078	6.314	12.706	63.657	636.619
2	0.289	0.617	1.061	1.886	2.920	4.303	9.925	31.599
3	0.277	0.584	0.978	1.638	2.353	3.182	5.841	12.924
4	0.271	0.569	0.941	1.533	2.132	2.776	4.604	8.610
5	0.267	0.559	0.920	1.476	2.015	2.571	4.032	6.869
6	0.265	0.553	0.906	1.440	1.943	2.447	3.707	5.959
7	0.263	0.549	0.896	1.415	1.895	2.365	3.499	5.408
8	0.262	0.546	0.889	1.397	1.860	2.306	3.355	5.041
9	0.261	0.543	0.883	1.383	1.833	2.262	3.250	4.781
10	0.260	0.542	0.879	1.372	1.812	2.228	3.169	4.587

Confidence Interval Variance Estimation

$$\hat{\sigma}_j^2 = \frac{1}{M-2} \sum_{m \neq j}^{M} X_m^2 - \frac{M-1}{M-2} (\overline{X})^2$$

$$\overline{X}_j = \frac{1}{M-1} \sum_{m \neq j}^{M} X_m$$

Confidence Interval
Variance Estimation

$$Z_j = M\sigma^2 - (M-1)\sigma_j^{\,2}$$

$$S_z^{\,2} = \frac{1}{M-1}\sum_{j=1}^{M}(Zj-\bar{Z})^2$$

$$\frac{\bar{Z}-\sigma^2}{s_z / M^{1/2}}$$

$$\text{Prob}\{\bar{Z} - t_{M-1}(1-\alpha/2)s_z / M^{1/2} \le \sigma^2 \le \bar{Z} + t_{M-1}(1-\alpha/2)s_z / M^{1/2}\} \approx 1-\alpha$$

Independent Replications

- Generate several sample paths for the model which are statistically independent and identically distributed.
- Reset the model performance measures at the beginning of each replication,
- Use a different random number seed for each independent replication

Independent Replications

- Distributions of the performance measures can then be assumed to have finite mean and variance
- With sufficient replications the average over the replications can be assumed to have a Normal distribution

Confidence Interval
Single run

- Sequence of output are correlated
- Many correlated observations must be taken to give the variance reduction achieved by one independent observation

$$Var[\overline{V}] \approx \frac{\sigma^2}{N}\left[\sum_{k=-\infty}^{\infty}\rho(k)\right]$$

Confidence Interval
Single run

- Batch means
- Regenerative Method
- Spectral Method

Batch Means

$$\rho(k) \approx 0 \qquad for \quad |k| \geq L$$

$$N \geq 5L$$

Batch Means

- Divide data in batches (sub-sample) and compute the mean of each batch
- The confidence interval is computed in the same way as in the independent replication method, except that samples are the batch means instead of means from different replications
- Discard lower amount of data than the replication method

Batch Means

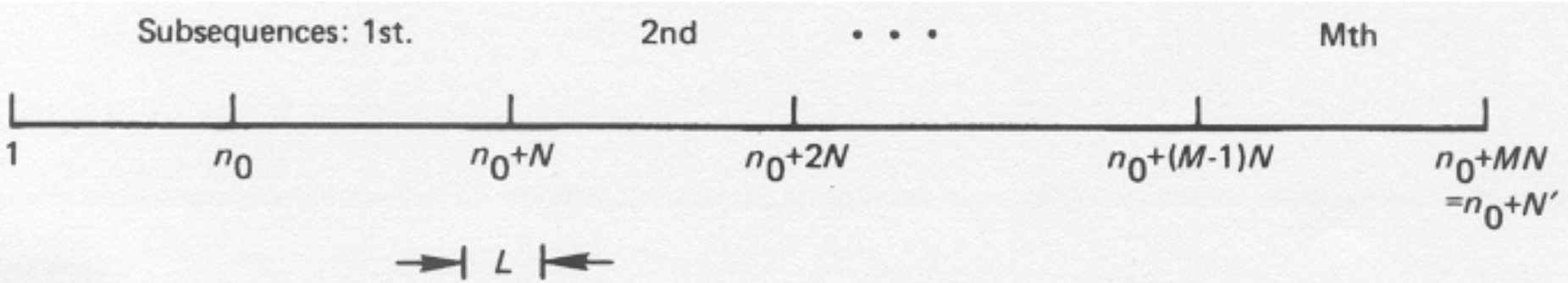


Fig. 6.14 The relationship between sequence lengths in the method of batch means.

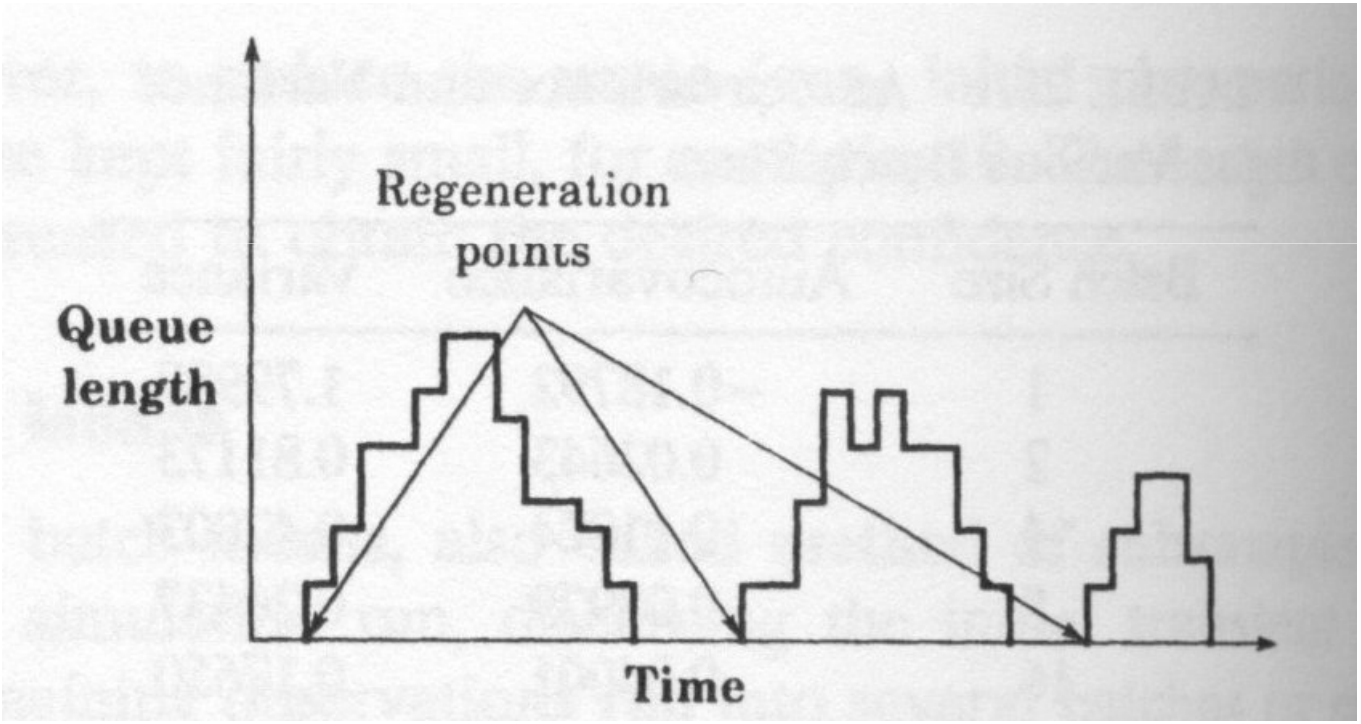
Batch Means

Batch Size	Autocovariance	Variance
1	−0.18792	1.79989
2	0.02643	0.81173
4	0.11024	0.42003
8	0.08979	0.26437
16	0.04001	0.17650
32	0.01108	0.10833
64	0.00010	0.06066
128	−0.00378	0.02992
256	0.00027	0.01133
512	0.00069	0.00503
1024	0.00078	0.00202

Regenerative Method

- Points of regeneration – no memory
- Tour - each period of regeneration
- Compute the desired value by taking the mean of the values obtained in each tour

Regenerative Method



Spectral Method

- Compute the correlation between runs
- Does not assume independent runs
- Confidence interval takes into account correlation between runs

Analysis of output data

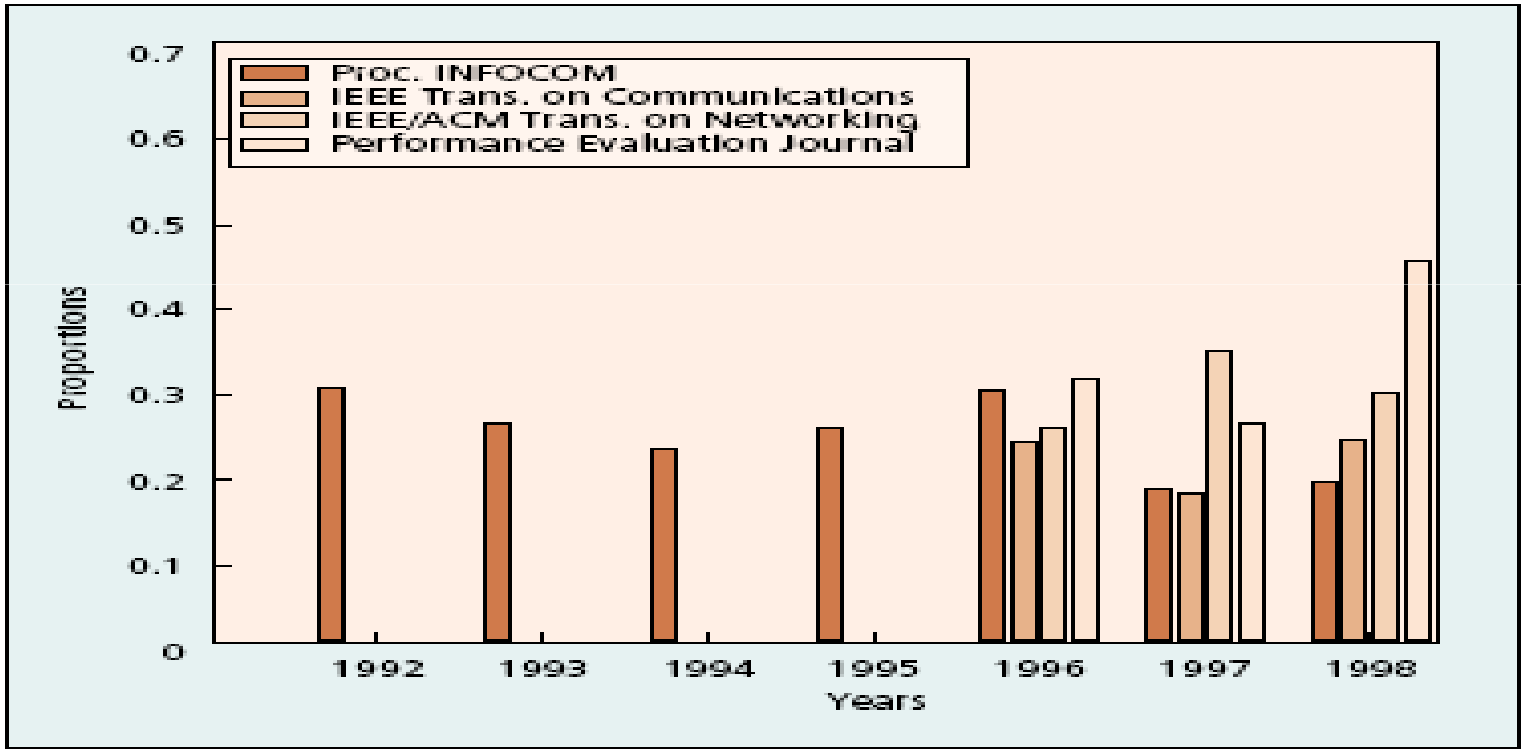


Figure 4. *Papers with* statistically analyzed *output data as a proportion of papers reporting simulation-based results.*

Trace Driven Simulation

- Trace – time ordered record of events on a system
  - Example : sequence of packets transmitted in a link
- Trace-driven simulation – trace input

Trace Driven Simulation

- Easy validation
- Accurate workload
- Less randomness
- Allow better understanding of complexity of real system

Trace Driven Simulation

- Representativiness
- Finiteness (huge amount of data)
- Difficult to collect data
- Difficult to change input parameters

Multiprocessed Simulation

- Work on a single simulation run
- Distributed Simulation
- Parallel Simulation

References

- Stephen Lavenberg, Computer Performance Modeling Handbook, Academic Press, 1983
- Raj Jain, "The art of Computer Systems Performance Analysis", John Wiley and Sons, 1991