

BSM 420 – BİLGİSAYAR MİMARİLERİ

Simulation

12.Hafta

Introduction (1 of 3)

The best advice to those about to embark on a very large simulation is often the same as Punch's famous advice to those about to marry: 'Don't!'
– Bratley, Fox and Schrage (1986)

- System to be characterized may not be available
 - During design or procurement stage
- Still want to predict performance
- Or, may have system but want to evaluate wide-range of workloads
 - Simulation
- However, simulations may fail
 - Need good programming, statistical analysis and perf eval knowledge

Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Common Mistakes in Simulation

(1 of 4)

- *Inappropriate level of detail*
 - Level of detail often potentially unlimited
 - But more detail requires more time to develop
 - And often to run!
 - Can introduce more bugs, making more inaccurate not less!
 - Often, more detailed viewed as “better” but may not be the case
 - More detail requires more knowledge of input parameters
 - Getting input parameters wrong may lead to more inaccuracy (Ex: disk service times exponential vs. simulating sector and arm movement)
 - Start with less detail, study sensitivities and introduce detail in high impact areas

Common Mistakes in Simulation

(2 of 4)

- *Improper language*
 - Choice of language can have significant impact on time to develop (Ch 24.4)
 - Special-purpose languages can make implementation, verification and analysis easier
 - C++Sim (<http://cxxsim.ncl.ac.uk/>), JavaSim (<http://javasim.ncl.ac.uk/>), SimPy(thon) (<http://simpy.sourceforge.net/>) ...
- *Unverified models*
 - Simulations generally large computer programs
 - Unless special steps taken, bugs or errors
 - Techniques to verify simulation models in Ch 25.1

Common Mistakes in Simulation

(3 of 4)

- *Invalid models*
 - No errors, but does not represent real system
 - Need to validate models by analytic, measurement or intuition
 - Techniques to verify simulation models in Ch 25.2
- *Improperly handled initial conditions*
 - Often, initial trajectory not representative of steady state
 - Including can lead to inaccurate results
 - Typically want to discard, but need method to do so effectively
 - Techniques to select initial state in Ch 25.4

Common Mistakes in Simulation

(4 of 4)

- *Too short simulation runs*
 - Attempt to save time
 - Makes even *more* dependent upon initial conditions
 - Correct length depends upon the accuracy desired (confidence intervals)
 - Variance estimates in 25.5
- *Poor random number generators and seeds*
 - “Home grown” are often not random enough
 - Makes artifacts
 - Best to use well-known one
 - Choose seeds that are different (Ch 26)

More Causes of Failure (1 of 2)

Any given program, when running, is obsolete. If a program is useful, it will have to be changed. Program complexity grows until it exceeds the capacity of the programmer who must maintain it. - Datamation 1968

Adding manpower to a late software project makes it later.
- Fred Brooks

- *Large software*
 - Quotations above apply to software development projects, including simulations
 - If large simulation efforts not managed properly, can fail
- *Inadequate time estimate*
 - Need time for *validation* and *verification*
 - Time needed can often grow as more details added

More Causes of Failure (2 of 2)

- *No achievable goal*
 - Common example is “model X”
 - But there are many levels of detail for X
 - Goals: Specific, Measurable, Achievable, Repeatable, Through (SMART, Section 3.5)
 - Project without goals continues indefinitely
- *Incomplete mix of essential skills*
 - Team needs one or more individuals with certain skills
 - Need: leadership, modeling and statistics, programming, knowledge of modeled system

Simulation Checklist (1 of 2)

- Checks before developing simulation
 - Is the goal properly specified?
 - Is detail in model appropriate for goal?
 - Does team include right mix (leader, modeling, programming, background)?
 - Has sufficient time been planned?
- Checks during simulation development
 - Is random number random?
 - Is model reviewed regularly?
 - Is model documented?

Simulation Checklist (2 of 2)

- Checks after simulation is running
 - Is simulation length appropriate?
 - Are initial transients removed?
 - Has model been verified?
 - Has model been validated?
 - Are there any surprising results? If yes, have they been validated?

(Plus, see previous checklist (Box2.1) for performance evaluation projects)

Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Terminology (1 of 7)

- Introduce terms using an example of simulating CPU scheduling
 - Study various scheduling techniques given job characteristics, ignoring disks, display...
- *State variables*
 - Variables whose values define current state of system
 - Saving can allow simulation to be stopped and restarted later by restoring all state variables
 - Ex: may be length of the job queue

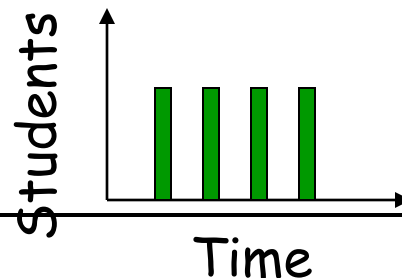
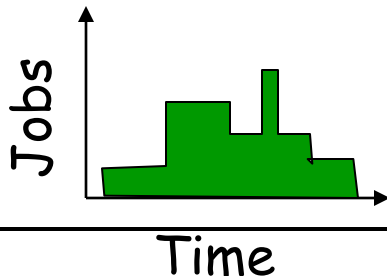
Terminology (2 of 7)

- *Event*

- A change in system state
- Ex: Three events: arrival of job, beginning of new execution, departure of job

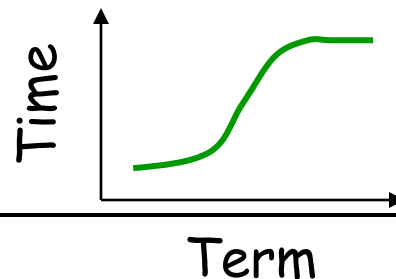
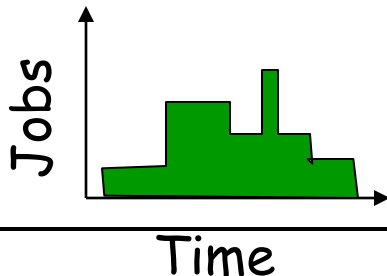
- *Continuous-time and discrete-time models*

- If state defined at all times → continuous
- If state defined only at instants → discrete
- Ex: class that meets M-F 2-3 is discrete since not defined other times



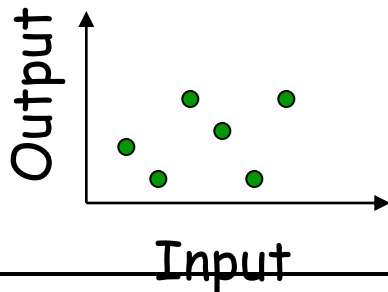
Terminology (3 of 7)

- *Continuous-state and discrete-state models*
 - If uncountably infinite → *continuous*
 - Ex: time spent by students on hw
 - If countable → *discrete*
 - Ex: jobs in CPU queue
 - Note, continuous time does not necessarily imply continuous state and vice-versa
 - All combinations possible

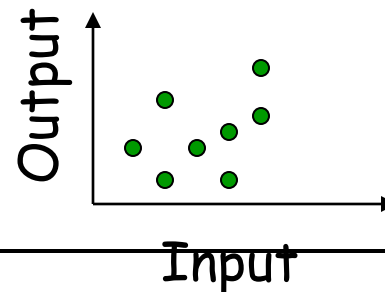


Terminology (4 of 7)

- *Deterministic and probabilistic models*
 - If output predicted with certainty → *deterministic*
 - If output different for different repetitions → *probabilistic*
 - Ex: For proj1, dog type-1 makes simulation deterministic but dog type-2 makes simulation probabilistic



(Deterministic)

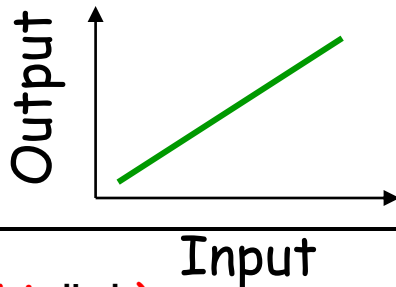


(Probabilistic)

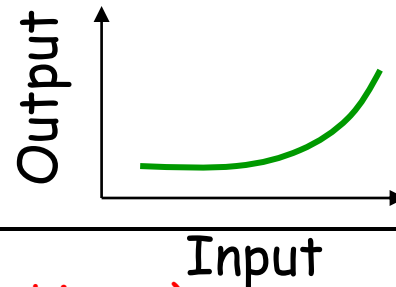
(vertical lines)

Terminology (5 of 7)

- *Static and dynamic models*
 - Time is not a variable → *static*
 - If changes with time → *dynamic*
 - Ex: CPU scheduler is dynamic, while matter-to-energy model $E=mc^2$ is static
- *Linear and nonlinear models*
 - Output is linear combination of input → *linear*
 - Otherwise → *nonlinear*



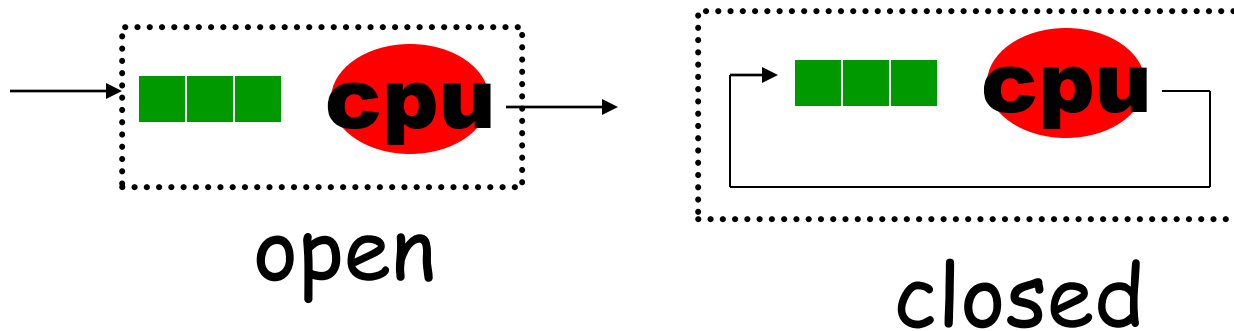
(Linear)



(Non-Linear)

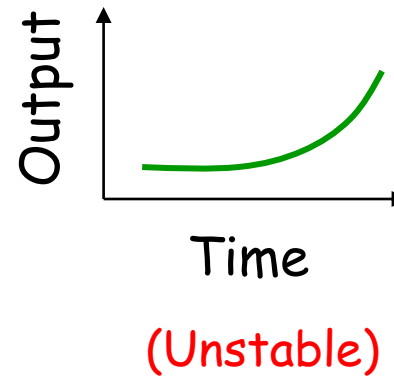
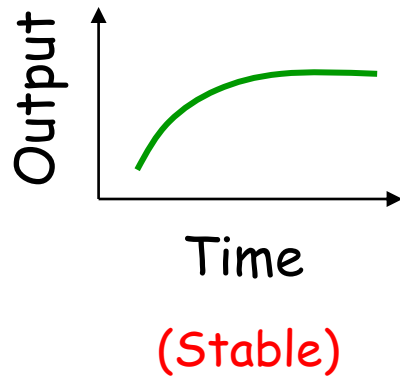
Terminology (6 of 7)

- *Open and closed models*
 - Input is external and independent → *open*
 - Closed model has no external input
 - Ex: if same jobs leave and re-enter queue then closed, while if new jobs enter system then open



Terminology (7 of 7)

- *Stable and unstable*
 - Model output settles down → *stable*
 - Model output always changes → *unstable*



Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Selecting a Simulation Language

(1 of 2)

- Four choices: simulation language, general-purpose language, extension of general purpose, simulation package
- *Simulation language* – built in facilities for time steps, event scheduling, data collection, reporting
- *General-purpose* – known to developer, available on more systems, flexible
- The major difference is the *cost tradeoff* – simulation language requires startup time to learn, while general purpose may require more time to add simulation flexibility
 - Recommendation may be for all analysts to learn one simulation language so understand those “costs” and can compare

Selecting a Simulation Language

(2 of 2)

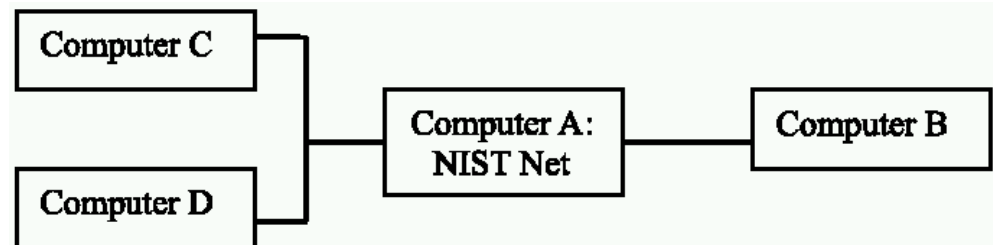
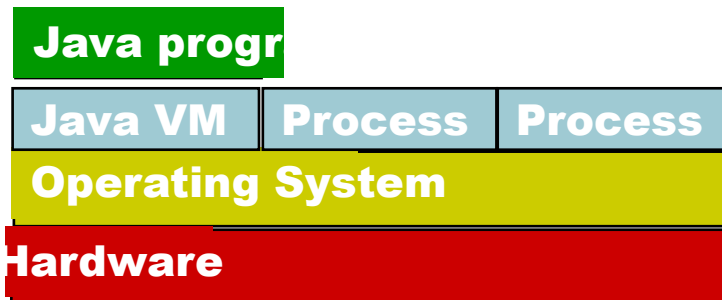
- *Extension of general-purpose* – collection of routines and tasks commonly used. Often, base language with extra libraries that can be called
- *Simulation packages* – allow definition of model in interactive fashion. Get results in one day
- Tradeoff is in *flexibility*, where packages can only do what developer envisioned, but if that is what is needed then is quicker to do so

Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- **Types of Simulations**
- Verification and Validation
- Transient Removal
- Termination

Types of Simulations

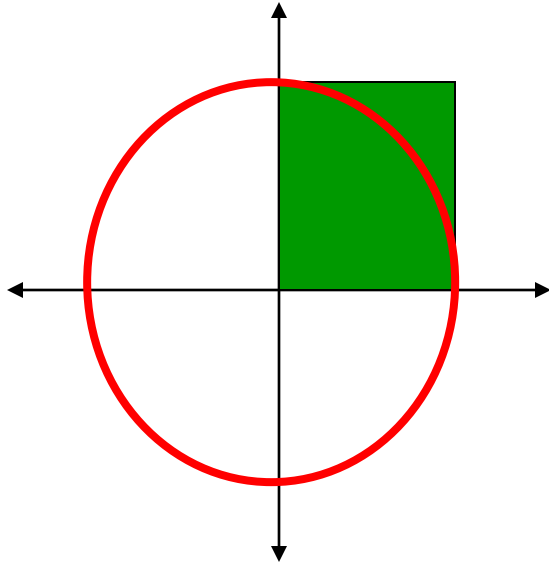
- Variety of types, but main: emulation, Monte Carlo, trace driven, and discrete-event
- *Emulation*
 - Simulation that runs on a computer to make it appear to be something else
 - Examples: JVM, NIST Net



Monte Carlo Simulation (1 of 2)

- A static simulation has no time parameter
 - Runs until some equilibrium state reached
- Used to model physical phenomena, evaluate probabilistic system, numerically estimate complex mathematical expression
- Driven with random number generator
 - So “Monte Carlo” (after casinos) simulation
- Example, consider numerically determining the value of π
- Area of circle = π^2 for radius 1

Monte Carlo Simulation (2 of 2)



- Unit square area of 1
- Ratio of area in quarter to area in square = R
 - $\pi = 4R$

- Imagine throwing dart at square
 - Random x (0,1)
 - Random y (0,1)
- Count if inside
 - $\text{sqrt}(x^2+y^2) < 1$
- Compute ratio R
 - $\text{in} / (\text{in} + \text{out})$
- Can repeat as many times as needed to get arbitrary precision

Trace-Driven Simulation

- Uses time-ordered record of events on real system as input
 - Ex: to compare memory management, use trace of page reference patterns as input, and can model and simulate page replacement algorithms
- Note, need trace to be independent of system
 - Ex: if had trace of disk events, could not be used to study page replacement since events are dependent upon current algorithm

Trace-Driven Simulation

Advantages

- *Credibility* – easier to sell than random inputs
- *Easy validation* – when gathering trace, often get performance stats and can validate with those
- *Accurate workload* – preserves correlation of events, don't need to simplify as for workload model
- *Less randomness* – input is deterministic, so output may be (or will at least have less non-determinism)
- *Fair comparison* – allows comparison of alternatives under the same input stream
- *Similarity to actual implementation* – often simulated system needs to be similar to real one so can get accurate idea of how complex

Trace-Driven Simulation

Disadvantages

- *Complexity* – requires more detailed implementation
- *Representativeness* – trace from one system may not represent all traces
- *Finiteness* – can be long, so often limited by space but then that time may not represent other times
- *Single point of validation* – need to be careful that validation of performance gathered during a trace represents only 1 case
- *Trade-off* – it is difficult to change workload since cannot change trace. Changing trace would first need workload model

Discrete-Event Simulations (1 of 3)

- Continuous events are simulations like weather or chemical reactions, while computers usually discrete events
- Typical components:
- *Event scheduler* – linked list of events
 - Schedule event X at time T
 - Hold event X for interval dt
 - Cancel previously scheduled event X
 - Hold event X indefinitely until scheduled by other event
 - Schedule an indefinitely scheduled event
 - Note, event scheduler executed often, so has significant impact on performance

Discrete-Event Simulations (1 of 3)

- *Simulation clock and time advancing*
 - Global variable with time
 - Scheduler advances time
 - *Unit time* – increments time by small amount and see if any events
 - *Event-driven* – increments time to next event and executes (typical)
- *System state variables*
 - Global variables describing state
 - Can be used to save and restore

Discrete-Event Simulations (2 of 3)

- *Event routines*

- Specific routines to handle event
- Ex: job arrival, job scheduling, job departure
- Often handled by call-back from event scheduler

- *Input routines*

- Get input from user (or config file, or script)
- Often get all input before simulation starts
- May allow range of inputs (from 1-9 ms) and number or repetitions, etc.

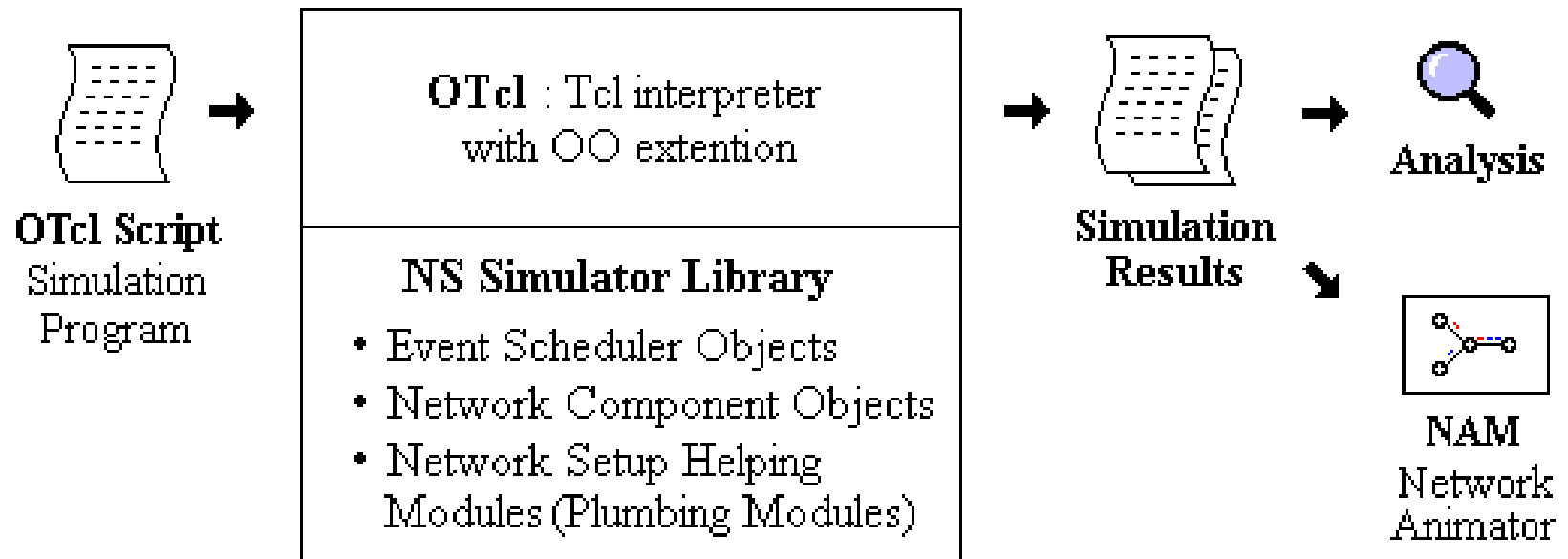
Discrete-Event Simulations (3 of 3)

- *Report generators*
 - Routines executed at end of simulation, final result and print
 - Can include graphical representation, too
 - Ex: may compute total wait time in queue or number of processes scheduled

Discrete Event Simulation

Example NS - (1 of 4)

- NS-2, network simulator
 - Government funded initially, Open source
- Wildly popular for IP network simulations

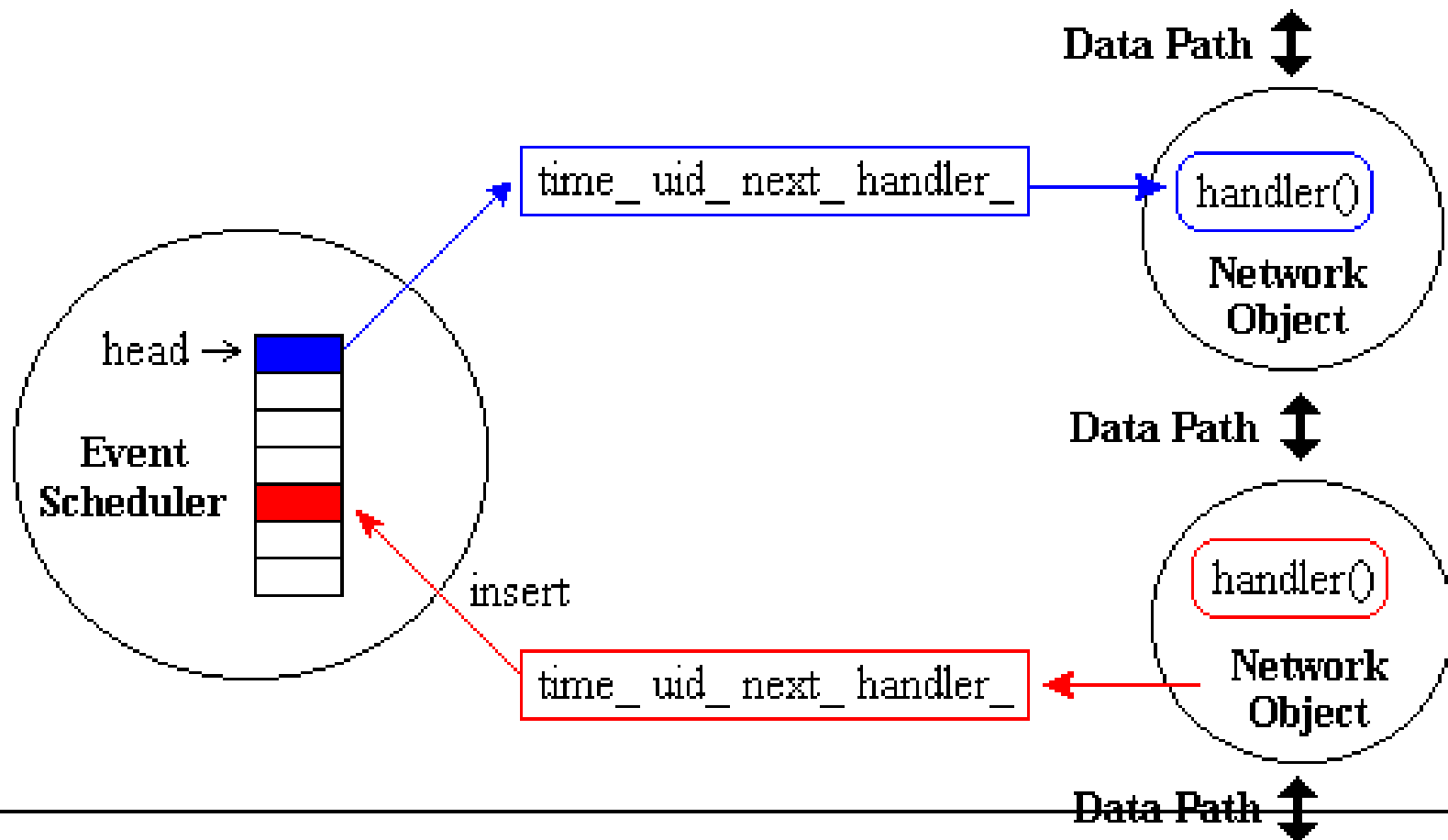


(<http://perform.wpi.edu/NS/>)

Discrete Event Simulation

Example NS - (2 of 4)

(Event scheduler is core of simulator)



Discrete Event Simulation

Example NS - (3 of 4)

Open the NAM trace file

```
set nf [open out.nam w]
```

```
$ns namtrace-all $nf
```

Define a 'finish' procedure

```
proc finish {} {
```

```
    global ns nf
```

```
    $ns flush-trace
```

```
    # Close the trace file
```

```
    close $nf
```

```
    #Execute NAM on file
```

```
    exec nam out.nam &
```

```
    exit 0
```

```
}
```

Setup a FTP

```
set ftp [new Application/FTP]
```

```
$ftp attach-agent $tcp
```

```
$ftp set type_ FTP
```

Initial schedule events

```
$ns at 0.1 "$cbr start"
```

```
$ns at 1.0 "$ftp start"
```

```
$ns at 4.0 "$ftp stop"
```

```
$ns at 4.5 "$cbr stop"
```

Finish after 5 sec (sim time)

```
$ns at 5.0 "finish"
```

Run the simulation

```
$ns run
```

Discrete Event Simulation

Example NS - (4 of 4)

- Output in text file, can be processed with Unix command line tools

event	time	from node	to node	pkt type	pkt size	flags	fid	src addr	dst addr	seq num	pkt id
-------	------	-----------	---------	----------	----------	-------	-----	----------	----------	---------	--------

```
r : receive (at to_node)
+ : enqueue (at queue)      src_addr : node.port (3.0)
- : dequeue (at queue)      dst_addr : node.port (0.0)
d : drop    (at queue)
```

```
r 1.3556 3 2 ack 40 ----- 1 3.0 0.0 15 201
+ 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
- 1.3556 2 0 ack 40 ----- 1 3.0 0.0 15 201
r 1.35576 0 2 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
d 1.35576 2 3 tcp 1000 ----- 1 0.0 3.0 29 199
+ 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207
- 1.356 1 2 cbr 1000 ----- 2 1.0 3.1 157 207
```

(Hey, run sample!)

Questions (1 of 2)

- Identify all relevant states:
 - continuous state vs. discrete time
 - deterministic vs. probabilistic
 - linear vs. non-linear
 - stable vs. unstable
- $y(t) = t + 0.2$
- $y(t) = 1/t^2$
- $y(t+1) = y(t) + \Delta$
 - For integer $\Delta \geq 1$
- $\underline{y}(t+1) = \underline{y}(t) + \Delta^t$
 - For $\Delta < 1$

Questions (2 of 2)

- Which type of simulation for each of:
 - Model requester address patterns to a server where large number of factors determine requester
 - Model scheduling in a multiprocessor with request arrivals from known distribution
 - Complex mathematical integral

Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Analysis of Simulation Results

Always assume that your assumption is invalid.

– Robert F. Tatman

- Would like model output to be close to that of real system
- Made assumptions about behavior of real systems
- 1st step, test if assumptions are reasonable
 - *Validation*, or representativeness of assumptions
- 2nd step, test whether model implements assumptions
 - *Verification*, or *correctness*
- Mutually exclusive.
 - Ex: what was your project 1?

Model Verification Techniques

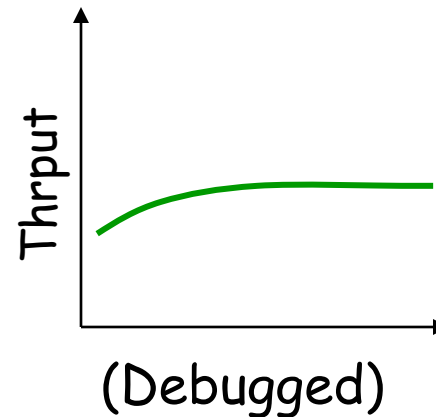
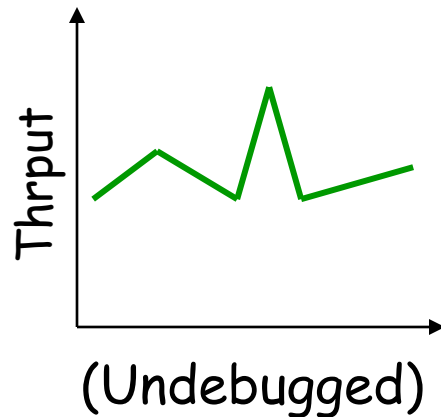
(1 of 3)

- Good software engineering practices will result in fewer bugs
- Top-down, modular design
- Assertions (antibugging)
 - Say, total packets = packets sent + packets received
 - If not, can halt or warn
- Structured walk-through
- Simplified, deterministic cases
 - Even if end-simulation will be complicated and non-deterministic, use simple repeatable values (maybe fixed seeds) to debug
- Tracing (via print statements or debugger)

Model Verification Techniques (2 of 3)

■ Continuity tests

- Slight change in input should yield slight change in output, otherwise error



■ Degeneracy tests

- Try extremes (lowest and highest) since may reveal bugs

Model Verification Techniques (3 of 3)

- Consistency tests – similar inputs produce similar outputs
 - Ex: 2 sources at 50 pkts/sec produce same total as 1 source at 100 pkts/sec
- Seed independence – random number generator starting value should not affect final conclusion (maybe individual output, but not overall conclusion)

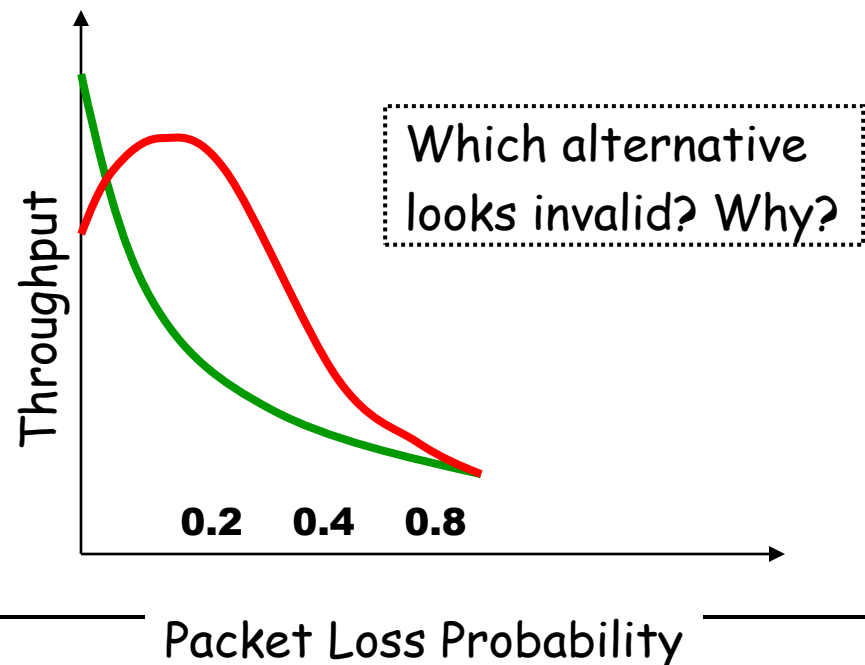
Model Validation Techniques

- Ensure assumptions used are reasonable
 - Want final simulated system to be like real system
- Unlike verification, techniques to validate one simulation may be different from one model to another
- Three key aspects to validate:
 - Assumptions
 - Input parameter values and distributions
 - Output values and conclusions
- Compare validity of each to one or more of:
 - Expert intuition
 - Real system measurements
 - Theoretical results

→ 9 combinations
- Not all are
always possible,
however

Model Validation Techniques - Expert Intuition

- Most practical, most common
- “Brainstorm” with people knowledgeable in area
 - Ex: mail carriers and dog owners for Proj1
- Assumptions validated first, followed soon after by input. Output validated as soon as output is available (and verified), even if preliminary
- Present measured results and compare to simulated results (can see if experts can tell the difference)



Model Validation Techniques - Real System Measurements

- Most reliable and preferred
- May be unfeasible because system does not exist or too expensive to measure
 - That could be why simulating in the first place!
- But even one or two measurements add an enormous amount to the validity of the simulation
- Should compare input values, output values, workload characterization
 - Use multiple traces for trace-driven simulations
- Can use statistical techniques (confidence intervals) to determine if simulated values different than measured values

Model Validation Techniques - Theoretical Results

- Can be used to compare a simplified system with simulated results
- May not be useful for sole validation but can be used to complement measurements or expert intuition
 - Ex: measurement validates for one processor, while analytic model validates for many processors
- Note, there is no such thing as a “fully validated” model
 - Would require too many resources and may be impossible
 - Can only show is invalid
- Instead, show validation in a few select cases, to lend *confidence* to the overall model results

Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Transient Removal

- Most simulations only want steady state
 - Remove initial *transient* state
- Trouble is, not possible to define exactly what constitutes end of transient state
- Use heuristics:
 - Long runs
 - Proper initialization
 - Truncation
 - Initial data deletion
 - Moving average of replications
 - Batch means

Long Runs

- Use very long runs
- Effects of transient state will be amortized
- But ... wastes resources
- And tough to choose how long is “enough”
- Recommendation ... don't use long runs alone

Proper Initialization

- Start simulation in state close to expected state
- Ex: CPU scheduler may start with some jobs in the queue
- Determine starting conditions by previous simulations or simple analysis
- May result in decreased run length, but still may not provide confidence that are in stable condition

Truncation

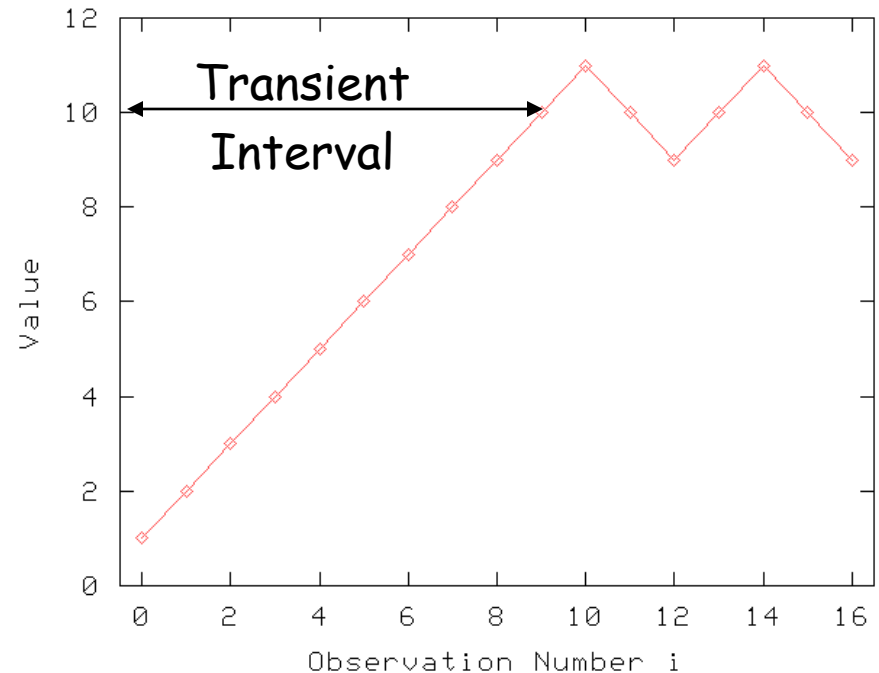
- Assume variability during steady state is less than during transient state
- Variability measured in terms of range
 - (min, max)
- If a trajectory of range stabilizes, then assume that in stable state
- Method:
 - Given n observations $\{x_1, x_2, \dots, x_n\}$
 - ignore first l observations
 - Calculate (min,max) of remaining $n-l$
 - Repeat for $l = 1 \dots n$
 - Stop when $l+1$ th observation is neither min nor max

(Example next)

Truncation Example

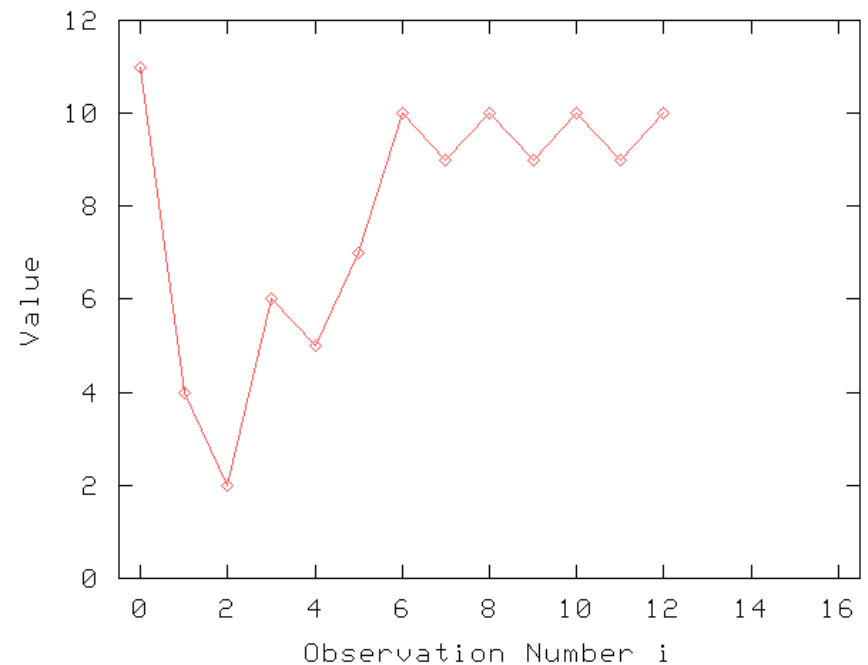
- Sequence: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 10, 9, 10, 11, 10, 9...
- Ignore first ($l=1$), range is (2, 11) and 2nd observation ($l+1$) is the min
- Ignore second ($l=2$), range is (3, 11) and 3rd observation ($l+1$) is min
- Finally, $l=9$ and range is (9, 11) and 10th observation is neither min nor max

- So, discard first 9 observations



Truncation Example 2 (1 of 2)

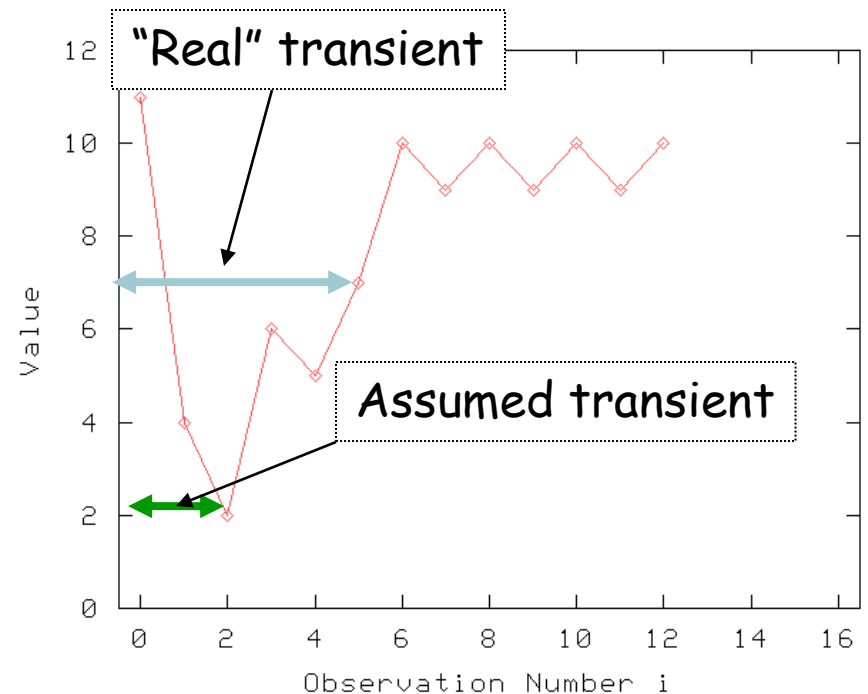
- Find duration of transient interval for:
11, 4, 2, 6, 5, 7, 10, 9,
10, 9, 10, 9, 10



Truncation Example 2

- Find duration of transient interval for:
11, 4, 2, 6, 5, 7, 10, 9,
10, 9, 10, 9, 10
- When $l=3$, range is (5,10) and 4th (6) is not min or max

- So, discard only 3 instead of 6



Initial Data Deletion (1 of 3)

- Study average after some initial observations are deleted from sample
 - If average does not change much, must be deleting from steady state
 - However, since randomness can cause some fluctuations during steady state, need multiple runs (w/different seeds)
- Given m replications size n each with x_{ij} j th observation of i th replication
 - Note j varies along time axis and i varies across replications

Initial Data Deletion (2 of 3)

- Get mean trajectory:

$$\underline{x}_j = (1/m)\sum x_{ij} \quad j=1,2,\dots,n$$

- Get overall mean:

$$\underline{x} = (1/n)\sum \underline{x}_j \quad j=1,2,\dots,n$$

- Set $l=1$. Assume transient state l long, delete first l and repeat for remaining $n-l$

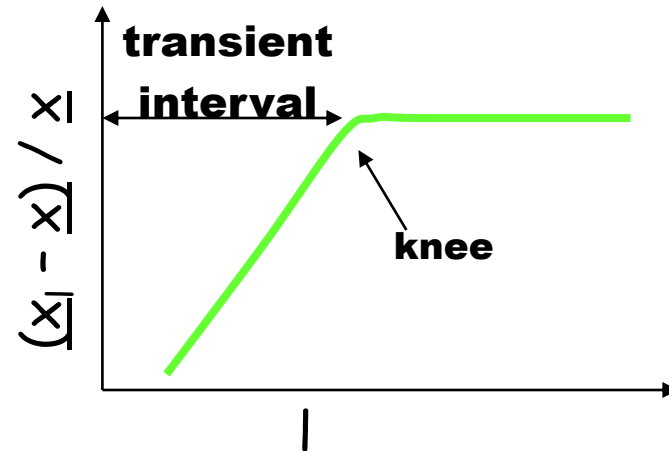
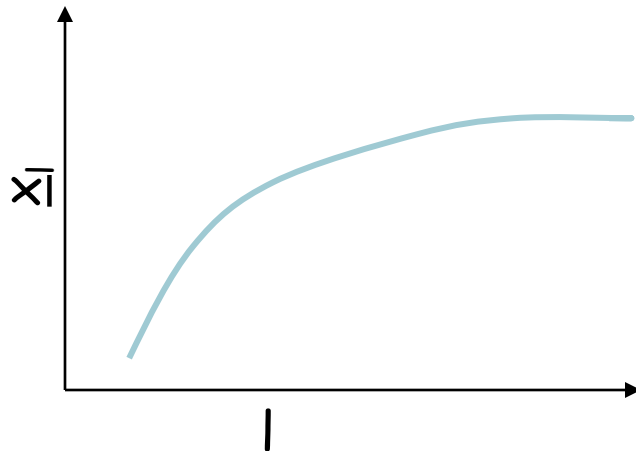
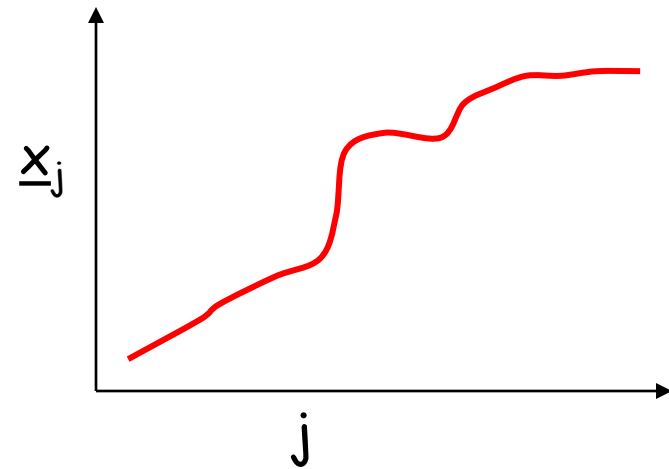
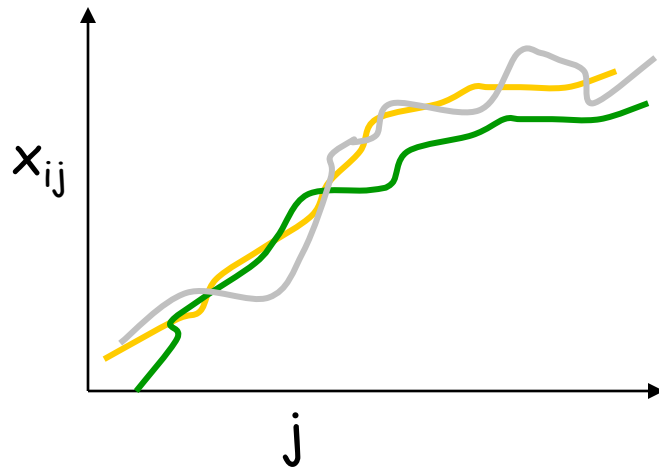
$$\underline{x}_l = (1/(n-l))\sum \underline{x}_j \quad j=l+1,\dots,n$$

- Compute relative change

$$(\underline{x}_l - \underline{x}) / \underline{x}$$

- Repeat with l from 1 to $n-1$. Plot. Relative change graph will stabilize at knee. Choose l there and delete 1 through l

Initial Data Deletion (3 of 3)



Moving Average of Independent Replications

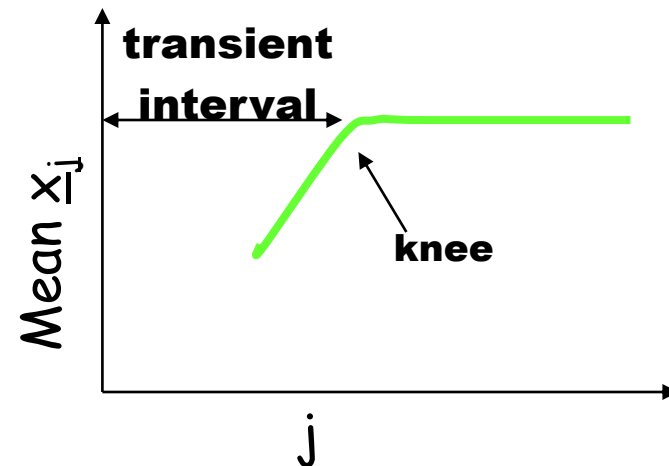
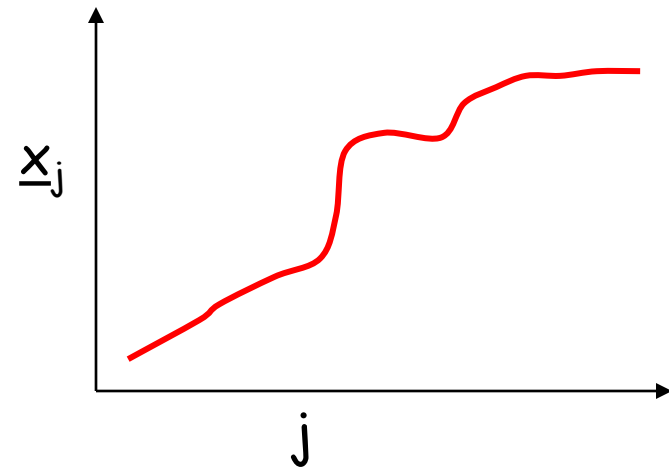
- Compute mean over moving time window
- Get mean trajectory

$$\underline{x}_j = (1/m) \sum x_{ij} \quad j=1,2,\dots,n$$
- Set $k=1$. Plot moving average of $2k+1$ values:

$$\text{Mean } \underline{x}_j = 1/(2k+1) \sum (\underline{x}_j + l)$$

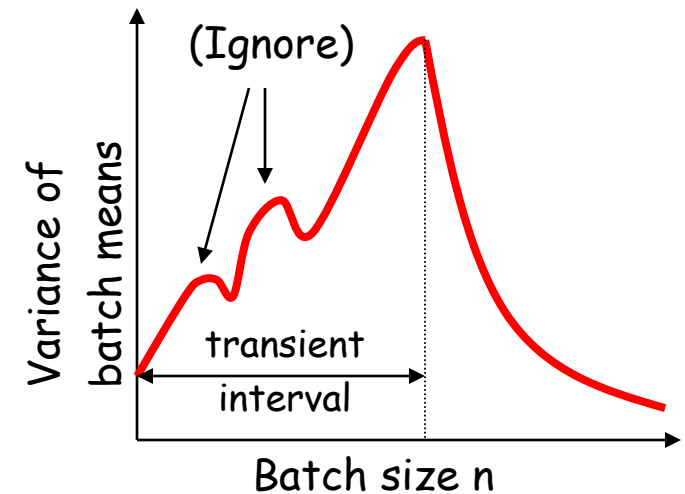
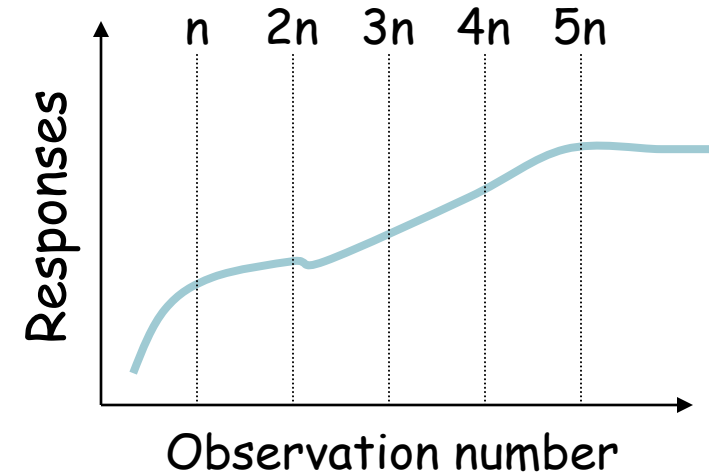
With $j=k+1, k+2, \dots, n-k$

With $l=-k$ to k
- Repeat for $k=2,3,\dots$ and plot until smooth
- Find knee. Value at j is length of transient phase.



Batch Means

- Run for long time
 - N observations
- Divide up into batches
 - m batches size n each so $m = N/n$
- Compute batch mean (\underline{x}_i)
- Compute var of batch means as function of batch size (\underline{X} is overall mean)
 - $\text{Var}(x) = (1/(m-1))\Sigma(x_i - \underline{X})^2$
- Plot variance versus size n
- When n starts decreasing, have transient



Outline

- Introduction
- Common Mistakes in Simulation
- Terminology
- Selecting a Simulation Language
- Types of Simulations
- Verification and Validation
- Transient Removal
- Termination

Terminating Simulations

- For some simulations, transition state is of interest no transient removals required
- Sometimes upon termination you also get final conditions that do not reflect steady state
 - Can apply transition removal conditions to end of simulation
- Take care when gathering at end of simulation
 - Ex: mean service time should include only those that finish
- Also, take care of values at event times
 - Ex: queue length needs to consider area under curve
 - Say $t=0$ two jobs arrive, $t=1$ one leaves, $t=4$ 2nd leaves
 - qlengths $q_0=2$, $q_1=1$ $q_4=0$ but q average not $(2+1+0)/3=1$
 - Instead, area is $2 + 1 + 1 + 1$ so q average $5/4=1.25$

Stopping Criteria

- Important to run long enough
 - Stopping too short may give variable results
 - Stopping too long may waste resources
- Should get confidence intervals on mean to desired width:

$$\underline{x} \pm z_{1-\alpha/2} \text{Var}(\underline{x})$$

- Variance of sample mean of independent observations

$$\text{Var}(\underline{x}) = \text{Var}(x) / n$$

- But only if observations independent! Most simulations not
 - Ex: if queuing delay for packet i is large then will likely be large for packet $i+1$
- So, use: independent replications, batch means, regeneration (all next)

Independent Replications

- Assume replications are independent
 - Different random seed values
- Collect m replications of size $n+n_0$ each
 - n_0 is length of transient phase
- Mean for each replication
$$\underline{x}_i = (1/n)\sum x_{ij} \quad i=1,2,\dots,m \quad j=n_0+1,\dots,n_0+n$$
- Overall mean for all replications
$$\underline{x} = (1/m)\sum \underline{x}_i \quad i=1,2,\dots,m$$
- Calculate variance of replicate means
$$\text{Var}(\underline{x}) = (1/(m-1))\sum (x_i - \underline{x})^2 \quad i=1,\dots,m$$
- Confidence interval is
$$\underline{x} \pm z_{1-\alpha/2} \text{Var}(\underline{x})$$
- Note, width proportional to \sqrt{mn} , but reduce “waste” of mn_0 observations, increase length n

Batch Means (1 of 2)

- Collect long run of N samples + n_0
 - n_0 is length of transient phase
- Divide into m batches of n observations each
 - n large enough so little correlation between
- Mean for each batch

$$\underline{x}_i = (1/n) \sum_{j=1, \dots, n} x_{ij} \quad i=1, 2, \dots, m$$

- Overall mean for all replications

$$\underline{x} = (1/m) \sum_{i=1, 2, \dots, m} \underline{x}_i$$

- Calculate variance of replicate means

$$\text{Var}(\underline{x}) = (1/(m-1)) \sum_{i=1, \dots, m} (x_i - \underline{x})^2$$

- Confidence interval is

$$\underline{x} \pm z_{1-\alpha/2} \text{Var}(\underline{x})$$

- Note, similar to independent replications but less waste (only n_0)

Batch Means (2 of 2)

- How to choose n? Want covariance of successive means small compared to variance

$$\text{Cov}(\underline{x}_i, \underline{x}_{i+1}) = 1/(m-2)\Sigma[(\underline{x}_i-\underline{x})(\underline{x}_{i+1}-\underline{x})]$$

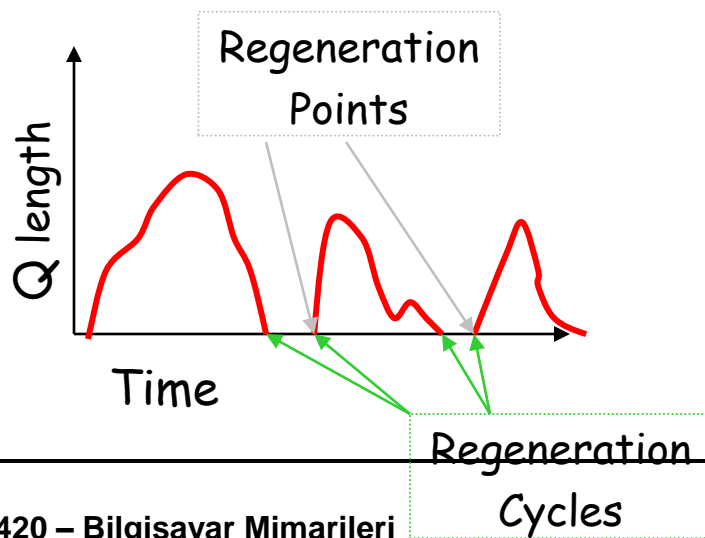
- Start n=1, then double n
- Example:

Size	Cov	Var
1	-0.187	1.799
2	0.026	0.811
4	0.110	0.420
...		
64	0.00010	0.06066

- Becomes less than 1%, so can use n=64 as batch size

Method of Regeneration (1 of 2)

- Consider CPU scheduling
- Jobs arriving after queue empty not dependent upon previous
- Note, system with two queues would need both to be idle
- Not all systems are regenerative
 - Not those with “long” memories
- Note, unlike in batch methods, the cycles can be of different lengths



Method of Regeneration (2 of 2)

- m cycles of length n_1, n_2, \dots, n_m
- Cycle means:
 $\underline{x}_i = (1/n_i) \sum_{j=1}^{n_i} x_{ij} \quad j=1 \text{ to } n_i$
- Note, overall mean is *not*:
 $\underline{x} \neq (1/m) \sum \underline{x}_i$
- Since cycles are different length
- So, to compute confidence intervals:

- Compute sums:
 $y_i = 1/n_i \sum x_{ij} \quad j=1 \text{ to } n_i$
- Compute overall mean:
 $\underline{x} = (\sum y_i) / (\sum n_i) \quad i=1 \text{ to } m$
- Calculate difference between expected and observed sums
 $w_i = y_i - n_i \underline{x} \quad i=1 \text{ to } m$
- Calculate variance of differences
 $s_w^2 = 1/(m-1) \sum w_i^2$
- Compute mean cycle length:
 $\underline{n} = 1/m \sum n_i \quad i=1 \text{ to } m$
- Conf interval:
 $\underline{x} \pm z_{1-\alpha/2} (s_w / (\underline{n} \sqrt{m}))$

Question

- Imagine you are called in as an expert to review a simulation study. Which of the following would you consider non-intuitive and would want extra validation?
 1. Throughput increases as load increases
 2. Throughput decreases as load increases
 3. Response time increases as load increases
 4. Response time decreases as load increases
 5. Loss rate decreases as load increases