CSC 508
System Performance & Evaluation

Simulation

Outline

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

Introduction

- 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 performance evaluation knowledge

Common Mistakes in Simulation

- *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

- *Improper language*
  - Choice of language can have significant impact on time to develop
  - Special-purpose languages can make implementation, verification and analysis easier
  - C++Sim, JavaSim, SimPy(thon) …

- *Unverified models*
  - Simulations generally large computer programs
  - Unless special steps taken, bugs or errors

Common Mistakes in Simulation

- *Invalid models*
  - No errors, but does not represent real system
  - Need to validate models by analytic, measurement or intuition

- *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

Common Mistakes in Simulation

- *Too short simulation runs*
  - Attempt to save time
  - Makes even *more* dependent upon initial conditions
  - Correct length depends upon the accuracy desired (confidence intervals)

- *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

More Causes of Failure

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

- *Large software*
  - The quote above apply to software development projects, including simulations
  - If large simulation efforts is not managed properly, it can fail

- *Inadequate time estimate*
  - Need time for *validation* and *verification*
  - Time needed can often grow as more details added

More Causes of Failure

- *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

- 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

- 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?

Outline

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

Terminology

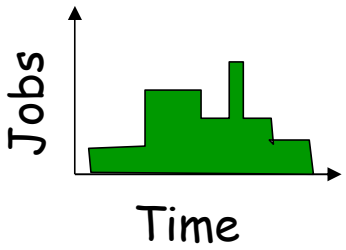
- Introduce terms using an example of simulating CPU scheduling
  - Study various scheduling techniques given job characteristics, ignoring disks, display…

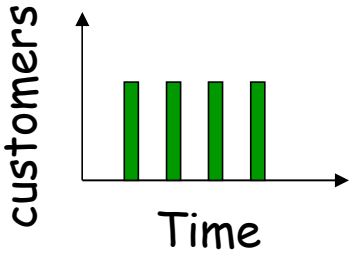
- *State variables*
  - Variables whose values define current state of system
  - Saving can allow simulation to be stopped and restarted later by restoring all state variables
  - Ex: may be length of the job queue

Terminology

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

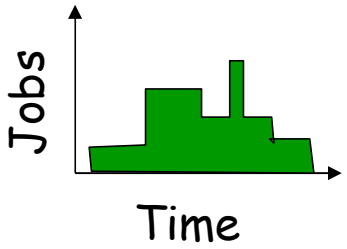
- *Continuous-time and discrete-time models*
  - If state defined at all times → continuous
  - If state defined only at instants → discrete
  - Ex: class that arrives in a banking hall is discrete since every customer arrives at their own times.

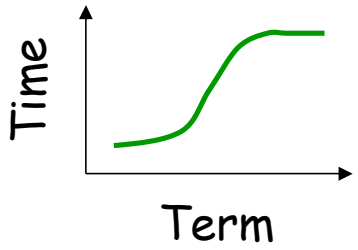




Terminology

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



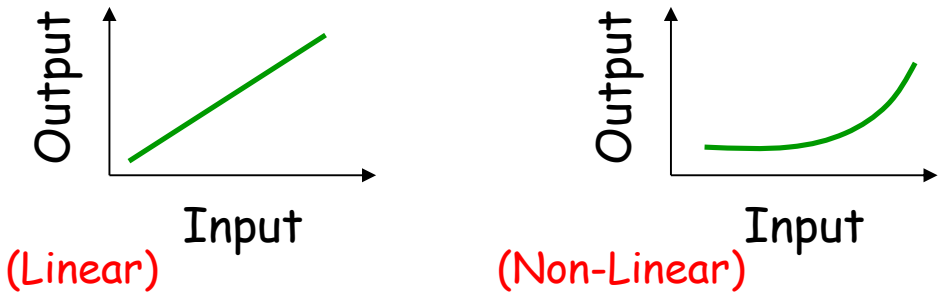


Terminology

- *Deterministic and probabilistic models*
  - If output predicted with certainty → *deterministic*
    - *return on a 5-year investment with an annual interest rate of 7%, compounded monthly.*
    - *Model is the equation*
  - If output different for different repetitions → *probabilistic*
    - Simulating who wins a given type of game is probabilistic

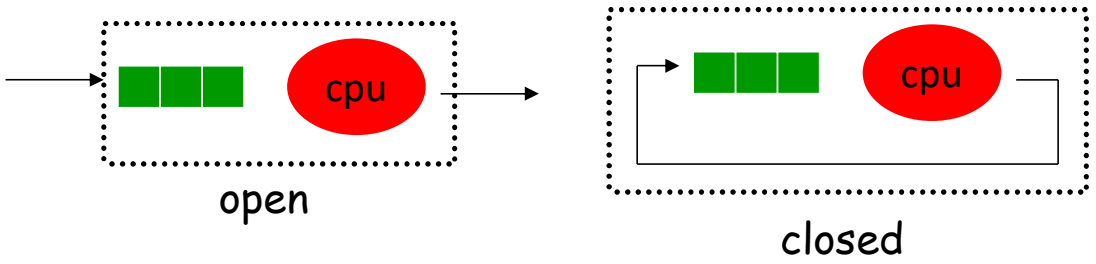
Terminology

- *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



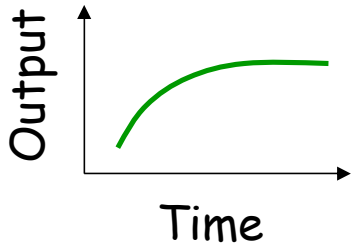
Terminology

- *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, it is closed, while if new jobs enter system then open

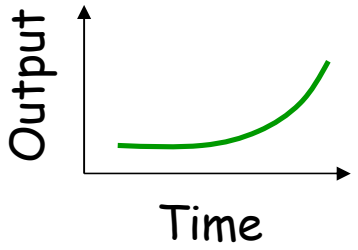


Terminology

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



(Stable)



(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

- 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

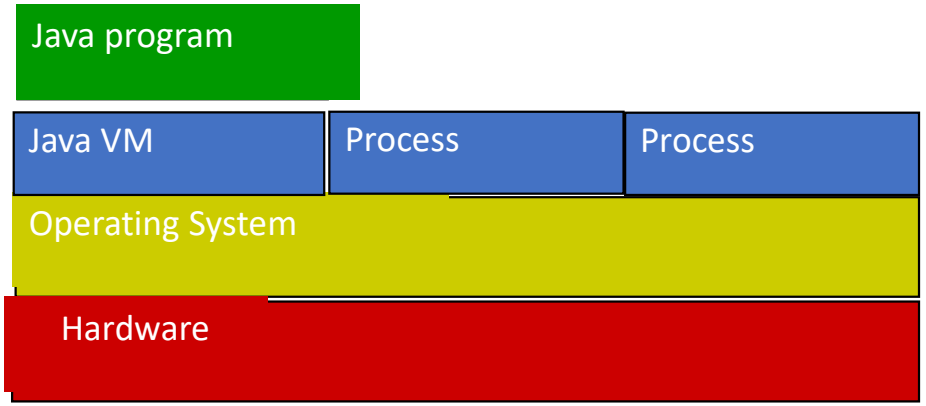
- *Extension of general-purpose* – collection of routines and tasks commonly used. Often, base language with extra libraries that can be called
- *Simulation packages* – allow definition of model in interactive fashion. Get results in one day
- Tradeoff is in *flexibility*, where packages can only do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

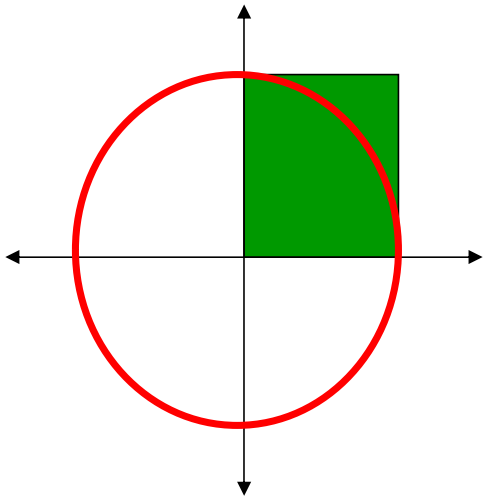
- Variety of types, but main: emulation, Monte Carlo, trace driven, and discrete-event
- *Emulation*
  - Simulation that runs on a computer to make it appear to be something else
  - Example: JVM



Monte Carlo Simulation

- 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



- 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
  - in / (in + 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

- 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

- *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

- *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

- *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

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

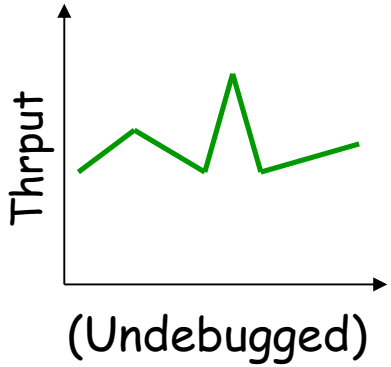
- Would like model output to be close to that of real system
- Made assumptions about behavior of real systems
- 1st step, test if assumptions are reasonable
  - *Validation*, or representativeness of assumptions
- 2nd step, test whether model implements assumptions
  - *Verification*, or *correctness*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

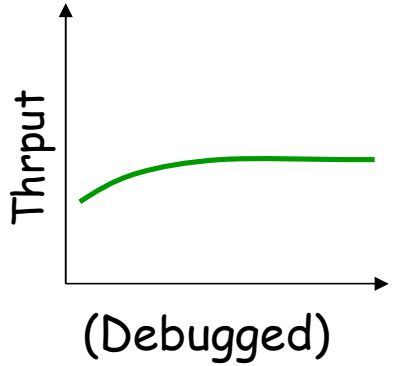
Model Verification Techniques

- 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

- 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

- 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)

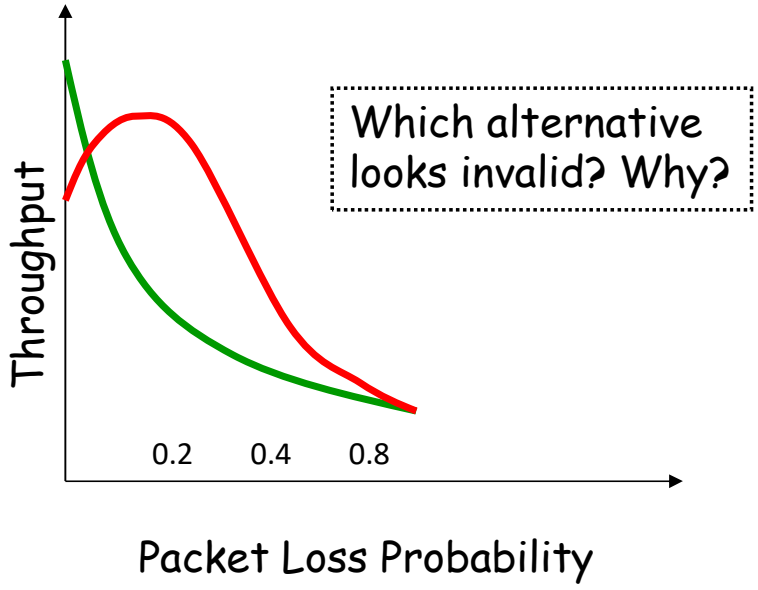
Model Validation Techniques

- 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

Model Validation Techniques - Expert Intuition

- Most practical, most common
- "Brainstorm" with people knowledgeable in area
- 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, it 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

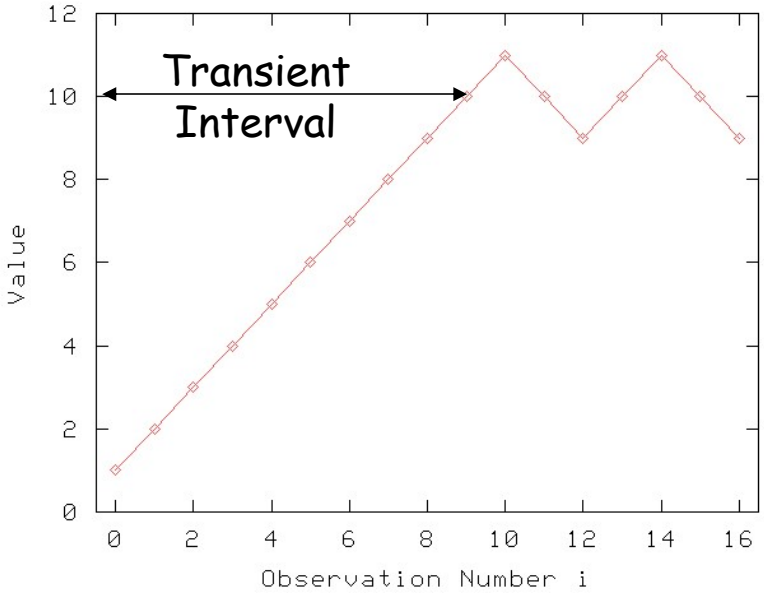
- Assume variability during steady state is less than during transient state
- Variability measured in terms of range
  - (min, max)
- If a trajectory of range stabilizes, then assume that in stable state

- Method:
  - Given n observations $\{x_1, x_2, \ldots, x_n\}$
  - ignore first i observations
  - Calculate (min,max) of remaining n-i
  - Repeat for i = 1…n
  - Stop when i+1th observation is neither min nor max

Truncation Example

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

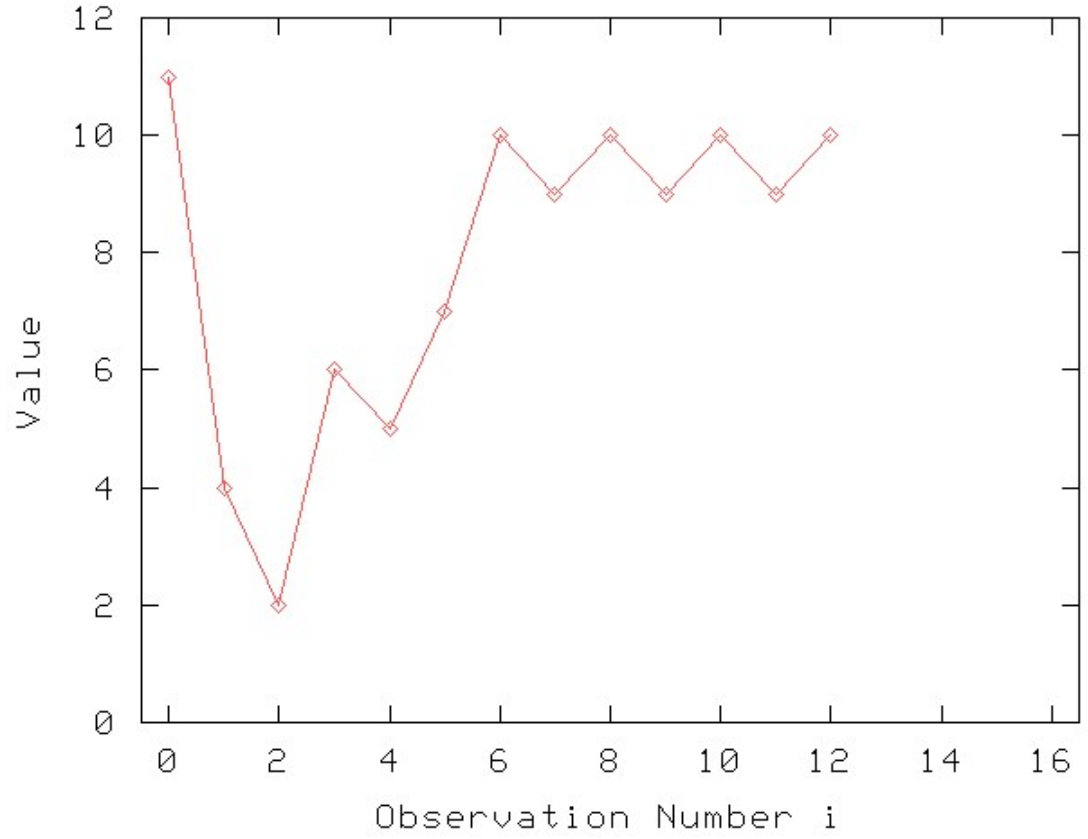
- So, discard first 9 observations



Truncation Example 2

- Find duration of transient interval for:

  11, 4, 2, 6, 5, 7, 10, 9, 10, 9, 10, 9, 10

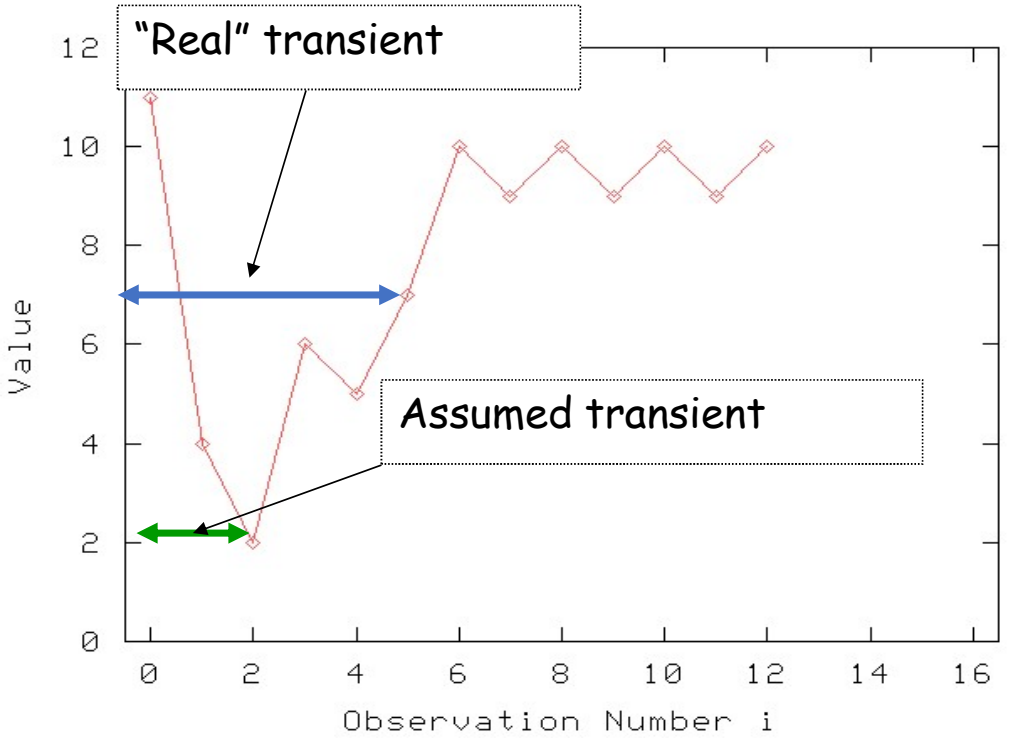


Truncation Example 2

- 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

- 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

- Get mean trajectory:

$$\underline{x}_j = (1/m)\Sigma x_{ij} \qquad j=1,2,\ldots,n$$

- Get overall mean:

$$\underline{x} = (1/n)\Sigma \underline{x}_j \; j=1,2,\ldots,n$$

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

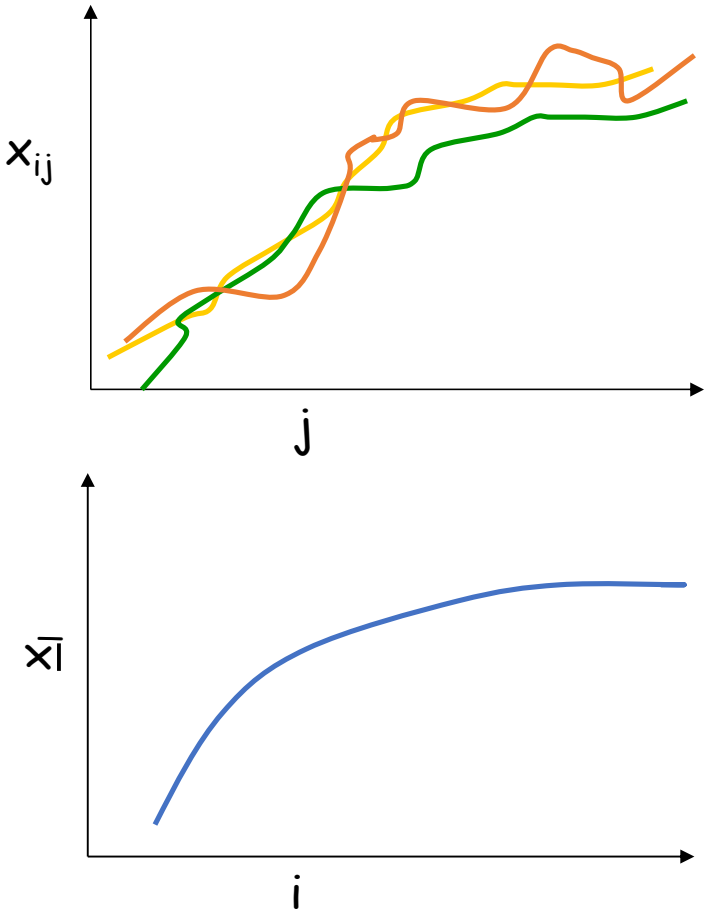
$$\underline{x}_l = (1/(n-i))\Sigma \underline{x}_j \qquad j=i+1,\ldots,n$$

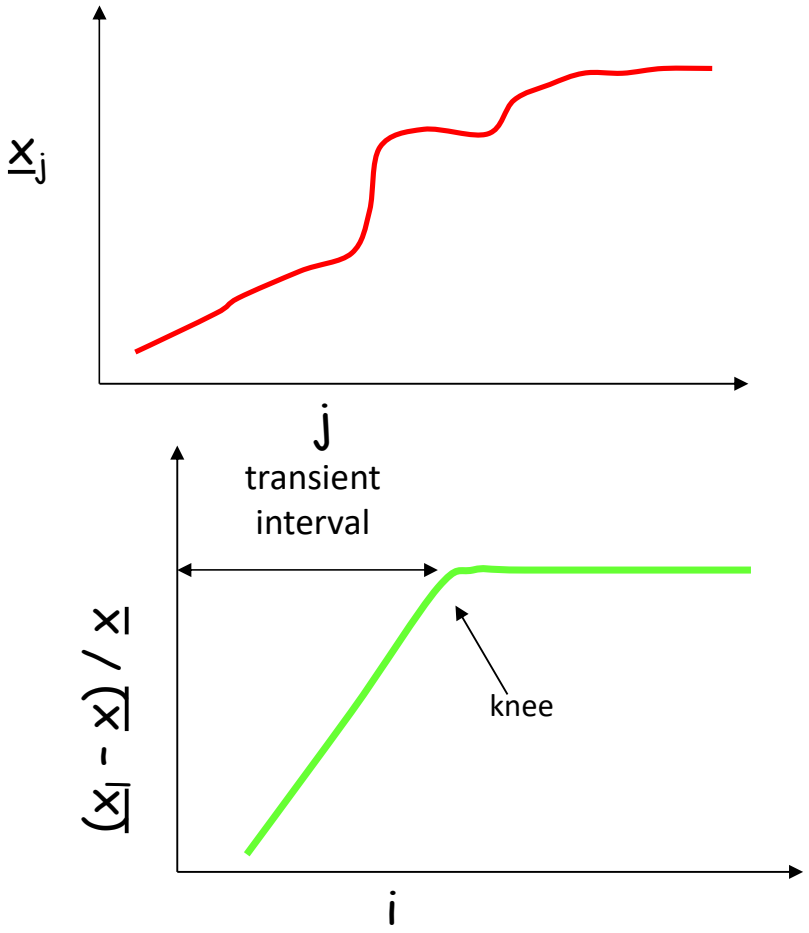
- Compute relative change

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

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

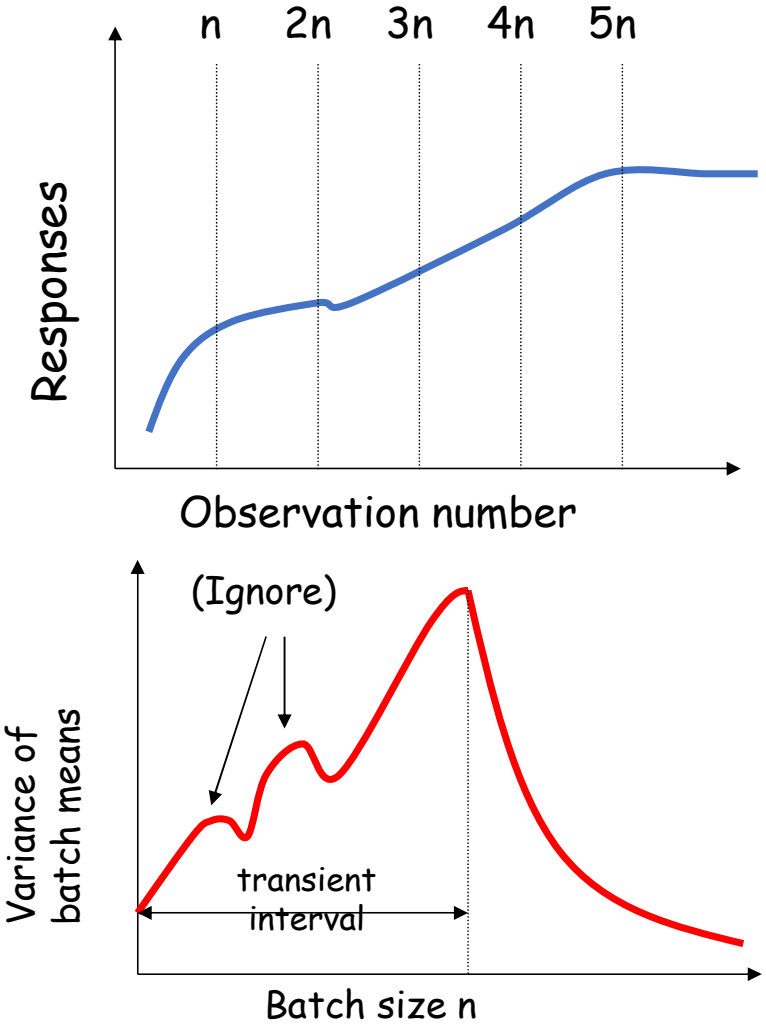
Initial Data Deletion





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



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:

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

- Variance of sample mean of independent observations

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

- But only if observations are independent! Most simulations are 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)

Assignment

- 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