ORIE 4580/5580: Simulation Modeling and Analysis

ORIE 5581: Monte Carlo Simulation

Unit 12: General Discrete-Event Simulation

Sid Banerjee

School of ORIE, Cornell University

the simulation hierarchy

closed-form solutions

- eg. queueing models
- have formulas for steady-state performance measures
- restrictive assumptions

- Excellent debugging tool

Markovian models

- need inter-event times to be exponentially distributed
- easier to simulate (no event-list needed)
- can give spurious insights, hide critical issues

discrete-event simulations

- most general framework
- allows detailed modeling, general distributions
- complex. takes time to code up

complex CTMCs

the Markov models we have seen have 2 common features:

– inter-event rates are time homogeneous

– inter-event times are exponential

we now see how to go beyond these models

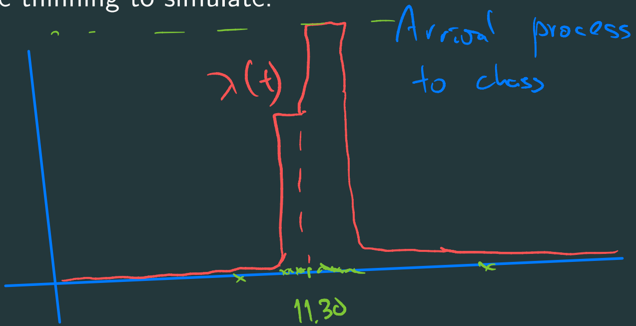
complex ctmcs

- use non-stationary Poisson process
- use 'phase-type' distributions instead of exponential

incorporating time-inhomogeneity

the easiest way to model time-inhomogeneity in a simulation model is to use a non-stationary Poisson process

- example: arrivals to a $m/m/1$ queue follow a Poisson process with rate $\lambda(t)$ (with $\lambda(t) \leq \lambda^*$ for all t)
- can use thinning to simulate:



fitting Poisson processes from data

question

have collected arrival times to a coffee shop over the time interval from 6:00 am to 8:00 pm

arrival time data $t_1, t_2, \ldots \Rightarrow$ interarrival times $a_1 = t_1, a_2 = t_2 - t_1, \ldots$

suppose we believe the data is from a stationary Poisson process

how can we learn λ?

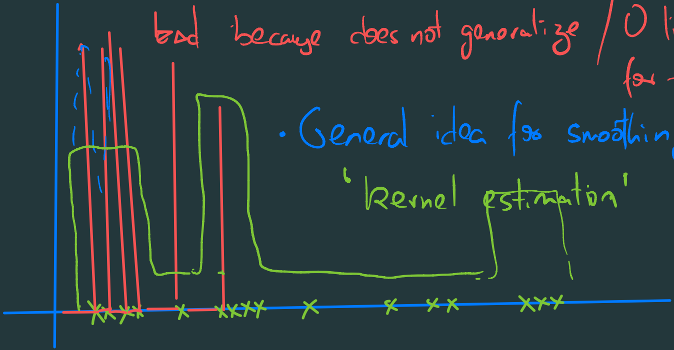
$$\lambda_{MLE} = \frac{n}{\sum_{i=1}^{n} a_i}$$

fitting non-stationary Poisson processes from data

question

have collected arrival times to a coffee shop over the time interval from 6:00 am to 8:00 pm

suppose we believe the data is from a non-stationary Poisson process



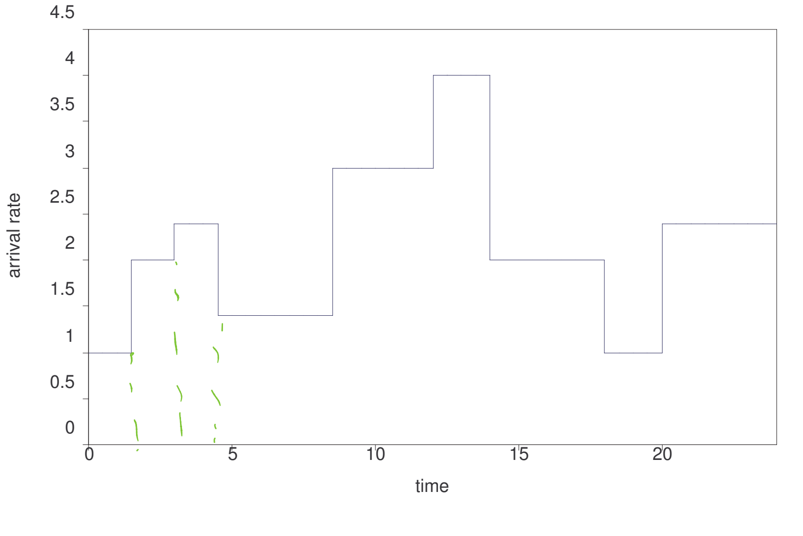
bad because does not generalize / 0 likelihood for future data

- General idea for smoothing –

'Kernel estimation'

case study: fitting non-stationary Poisson processes

idea: assume rate function $\lambda(\cdot)$ is piecewise constant with known breakpoints



fitting non-stationary Poisson processes from data

1. divide the time interval 6:00 am-8:00 pm into subintervals, such that arrival rate over each is assumed to be
2. n_i^k : number of arrivals during i-th subinterval in day k
m_i: average number of arrivals during subinterval i.

$$m_i = \frac{\sum_k n_i^k}{\text{number of days}}.$$

3. m_i estimates $\lambda_i[t_i - t_{i-1}]$
estimate λ_i (the arrival rate over the i-th subinterval) as

$$\lambda_i = \frac{m_i}{t_i - t_{i-1}}.$$

fitting non-stationary Poisson processes from data

- there is no easy rule for picking the number and placement of subintervals
- if one picks too many, then each subinterval contains too few arrivals and estimation is meaningless
- if one picks too few, then one can end up making an unrealistic assumption that the arrival rate over the subinterval is constant

non-exponential inter-event times

problem

want to model time taken by a bus to go from collegetown to the commons

method 1: use an exponentially distributed travel time

problems:

1) speed limits $\Rightarrow$ minimum time

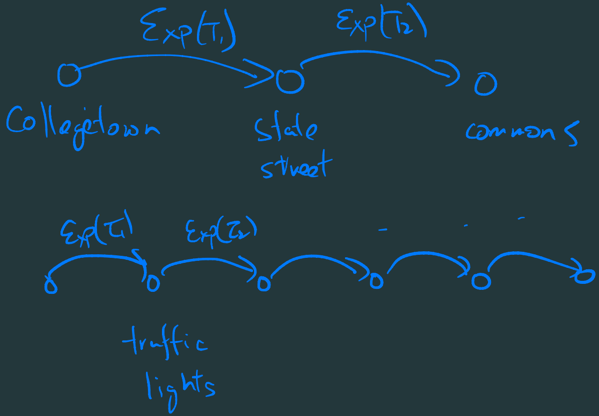
2) Traffic lights $\Rightarrow$ multi-modal distn

non-exponential inter-event times

problem

want to model time taken by a bus to go from collegetown to the commons

method 2: use a sum of exponentially distributed travel times

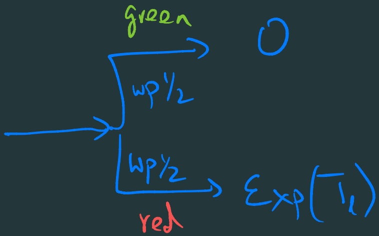


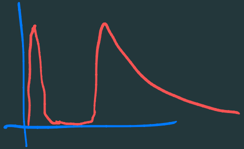
non-exponential inter-event times

problem

want to model time taken by a bus to go from collegetown to the commons

method 3: use a mixture of exponentially distributed travel times



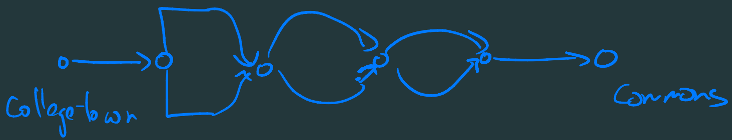


non-exponential inter-event times

problem

want to model time taken by a bus to go from collegetown to the commons

method 3: use a markov chain



phase-type distributions

'absorption time' of any finite-state ctmc with a single start state and one or more terminal states

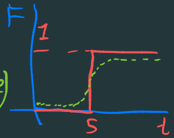
- thm: every non-negative distribution is approximately phase-type

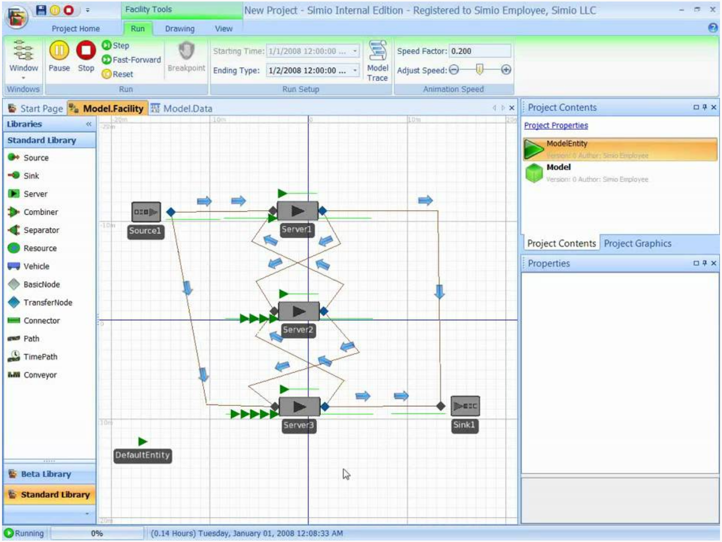
positive - end up with a Markovian simulation

downside - state space may blow up

Eg - If T = constant (5)

$\left(\text{approx} = T = \sum_{i=1}^{100} T_i,\ T_i \sim Exp\left(\frac{100}{5}\right)\right)$





discrete-event simulation

discrete-event simulation: example

the single-server $GI/GI/1$ queue

- number of servers: 1
- queue capacity: infinite
- service discipline: first-come-first-served (FCFS or FIFO)
- i.i.d. interarrival times with cdf $F_a(\cdot)$
- i.i.d. service times with cdf $F_s(\cdot)$
- independent interarrival and service times.

common features of DES models

simulation clock

variable that keeps track of the simulated time

- every DES model has a 'simulation clock'; helps model dynamics
- two methods for advancing the simulation clock:
  - fixed-increment time advance
  - next-event time advance
- fixed-increment time advance used for continuous simulations
- for next-event time advance, see the rest of this lecture!

common features of DES models

system state

collection of variables needed to DEScribe status of system at any time

- defining the system state is the first and most important steps in building a DES model
- example: single-server queue:
  $X(t) =$ number of customers in the system at time t
- example: two-server queue:

common features of DES models

events

instantaneous occurrence that modifies system state

example: for single-server queue

– set of possible events are {arrival, departure}

– customer arrives at time $t \to X(t) \to X(t) + 1$

– customer completes service and departs at $t \to X(t) \to X(t) - 1$

event list

event list contains the events that will occur in the future

example: for single-server queue at t, we may have the event list

$$(a, t+5),\ (b, t+7),\ (c, t+10)$$

- approaches for generating the events in an event list:
  - pre-generate the arrival times of all events
  - generate events as needed

common features of simulation models

timing and event-handling

- each event in event list has an 'occurrence time'
  (i.e., time stamp showing when the event is to occur)
- $\{\text{simulation clock} = t\} \implies \{\text{occ. time of } \textit{all} \text{ events in event list} \geq t\}$

timing routine

- selects the earliest event in the event list
  advances the simulation clock to the time of this event
- then the event handler:
  - processes the event
  - updates state to reflect changes induced by event
  - (maybe) generates new events

other components of a simulation model

- for any DES model, {simulation clock, state, event-list, event-handling} are enough to DEScribe the evolution of the system over time
- other important components:
  1. statistical counters (for tracking metrics)
  2. library routines
  3. reporting routines

simulating the single server queue

- when an arrival event occurs:
  1. add 1 to the number of customers in the system
  2. generate the next customer arrival event
  3. if the server was previously idle, then the newly arriving customer immediately starts to receive service

  $\implies$ generate the service completion event for this customer

simulating the single server queue

- when a service completion event occurs:
  1. reduce the number of customers in the system by 1
  2. if there are other customers in the system, then the first customer in the queue immediately starts to receive service

  $\implies$ schedule the service completion event for this customer

overall algorithm

1. initialize clock, system state, event list and counters
2. timing routine:
   - determine closest (imminent) event in the event list
   - advance clock to imminent event's occurrence time
   - event handling routine (executing the event):
     - update the system state
     - update the counters:

$N(t) = \#$ of customers served

$W(t) =$ (avg) time spent by customers in system

     - generate the future events and add them to the event list

hand simulation of the single-server queue

- state variable: $X(t) = \#$ of customers in system at time t
- initialization: set $\quad t = 0 \quad X(t) = 0$
- schedule an 'end simulation' event $(e, 120)$
- generate first interarrival t_1 time from c.d.f. $f_a(\cdot)$
  update event list to $\big((a, t_1), (e, 120)\big)$ and continue

hand simulating the single-server queue

t	$X(t)$	notes	event list	$N(t)$	$W(t)$

simulating the single-server queue: notes

- proceed from event to event, updating the simulation clock
- the clock does not always increase by the same amount
- the event list, simulation clock (t) and state ($X(t)$) completely determine the future evolution of the simulation. never need to look back at past behavior