Performance Modeling of Computer Systems and Networks

Prof. Vittoria de Nitto Personè

Continuous Random Variates: applications

Università degli studi di Roma Tor Vergata
Department of Civil Engineering and Computer Science Engineering

1

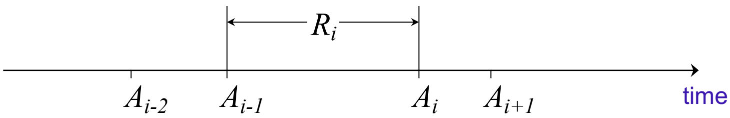
For our application framework, we will look at:

- arrival process model
- service process model

2

Arrival process model

- Model *interarrival* times as RV sequence $R_1, R_2, R_3, \ldots$
- Construct corresponding arrival times $A_1, A_2, A_3, \ldots$ defined by
  $A_0 = 0$ and $A_i = A_{i-1} + R_i \quad i=1, 2, \ldots$
- by induction, $A_i = R_1 + R_2 + \ldots + R_i \quad i=1, 2, \ldots$
- since $R_i > 0$, $0 = A_0 < A_1 < A_2 < A_3 < \ldots$



Example

- programs `ssq2` and `ssq3` generate job arrivals in this way, where R_1, $R_2, R_3, \ldots$ are *Exponential*$(1/\lambda)$.
  In both programs, the arrival rate is equal to $\lambda = 0.5$ jobs per unit time

```
 double GetArrival()
{ static double arrival = START;
  SelectStream(0);
  arrival += Exponential(2.0);
  return (arrival);}
```

Example

- programs `sis3` and `sis4` generate demand instances in this way, with *Exponential*(1/λ) interdemand times.
  The demand rate corresponds to an average of
  - $\lambda = 30.00$ actual demands per time interval in `sis3`
  - $\lambda = 120.00$ potential demands per time interval in `sis4`

```
double GetDemand(long *amount)
/* ----------------------------------------
 * generate a demand instance with rate 120
 * and generate a corresponding demand amount
 * ----------------------------------------
 */
{
  static double time = START;

  SelectStream(0);
  time += Exponential(1.0 / 120.0);      /* demand instance */
  SelectStream(2);
  *amount = Geometric(0.2);              /* demand amount   */
  return (time);
}
```

```
double GetDemand(void)
/* ------------------------------------------------------------
 * generate the next demand instance (time) with rate 30 per time
 * interval and exactly one unit of demand per demand instance
 * ------------------------------------------------------------
 */
{
  static double time = START;

  SelectStream(0);
  time += Exponential(1.0 / 30.0);
  return (time);
}
```


Def stationary arrival process

If $R_1, R_2, R_3, \ldots$ is an *iid* sequence of positive interarrival times with $E[R_i] = 1/\lambda > 0$, then the corresponding sequence of arrival times $A_1, A_2, A_3, \ldots$ is a *stationary arrival process* with rate λ

- also known as
  - *Renewal* processes
  - *Homogeneous* arrival processes
- arrival rate λ has units "arrivals per unit time"
  - If average interarrival time is 0.1 minutes,
  - then the arrival rate is 10 arrivals per minute
- stationary arrival processes are "convenient fiction"

important theoretically
good approximation
understand for nonstationary

- If the arrival rate λ varies with time, the arrival process is *nonstationary*

stationary Poisson arrival process

random events!

- As in `ssq2`, `ssq3`, `sis3`, `sis4`, with lack of information it is usually most appropriate to assume that the interarrival times are *Exponential*($1/\lambda$)

- If $R_1, R_2, R_3, \dots$ is an *iid* sequence of *Exponential*($1/\lambda$) interarrival times, the corresponding sequence $A_1, A_2, A_3, \dots$ of arrival times is a stationary *Poisson* arrival process with rate λ Equivalently, for $i=1, 2, 3, \dots$, the arrival time A_i is an *Erlang*(i,$1/\lambda$) random variable

$$A_i = R_1 + R_2 + \dots + R_i$$

Algorithm 1

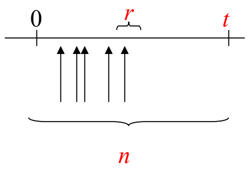
Given $\lambda > 0$ and $t > 0$, this algorithm generates a *realization* of a stationary Posson arrival process with rate λ over $(0, t)$

```
a0 = 0.0;            /* a convention */
n = 0;
while(an < t) {
    an+1 = an + Exponential(1 / λ);
    n++;
}
return a1, a2, a3, . . . , an;
```

Random Arrivals

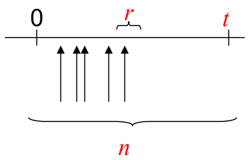
We now demonstrate the interrelation between Uniform, Exponential and Poisson random variables.

Consider:
- $t > 0$, defines a fixed time interval $(0, t)$
- n rappresents the number of arrivals in the interval $(0, t)$
- $r > 0$, is the length of a small subinterval located at random interior to $(0, t)$



Correspondingly:
- $\lambda = n / t$ is the arrival rate
- $p = r / t$ is the probability that a particular arrival will be in the subinterval
- $np = nr / t = \lambda r$ is the expected number of arrivals in the subinterval

random arrivals → Poisson



Theorem 1

Let:
- $A_1, A_2, A_3, \ldots$ be an iid sequence of *Uniform*(0, t) random variables ("unsorted" arrivals).
- the discrete random variable X be the number of A_i that fall in a fixed subinterval of length $r = pt$ interior to $(0, t)$

If n is large and r / t small, X is indistinguishable from a *Poisson(λr)* random variable with $\lambda = n / t$

Conclusions on random arrivals

- if *many* arrivals occur *at random* with a rate of λ, the number of arrivals X that will occurr in an interval of length r is *Poisson(λr)*
- The probability of x arrivals in an interval with length r is

$$Pr(X = x) = \frac{e^{-\lambda r}(\lambda r)^x}{x!} \qquad x = 0,1,2,..$$

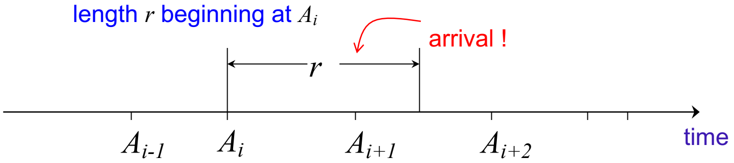
- The probability of <u>no arrivals</u> is: $Pr(X = 0) = e^{-\lambda r}$
- The probability of <u>at least one arrival</u> is:

$$Pr(X > 0) = 1 - Pr(X = 0) = 1 - e^{-\lambda r}$$

For a fixed λ, the probability of at least one arrival increases with increasing interval length r

Random Arrivals $\rightarrow$ Exponential Interarrivals

- If R represents the time between consecutive arrivals, the possible values of R are $r > 0$
- Consider arrival time A_i selected at random and an interval of length r beginning at A_i



- $R = A_{i+1} - A_i$ will be less than r iff there is at least one arrival in this interval
- the cdf of R is

$$Pr(R \le r) = Pr\,(\text{at least one arrival in } r) = 1 - e^{-\lambda r}$$

- R is an *Exponential($1/\lambda$)* random variable

Theorem 2

If arrivals occur at random with rate λ, the corresponding interarrival times form an iid sequence of *Exponential(1/λ)* RVs.

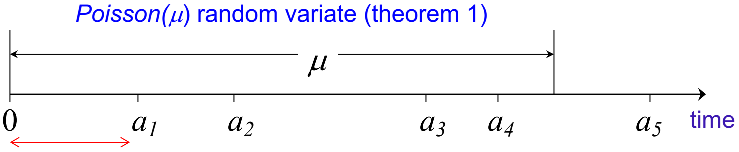
This result justifies the use of *Exponential* interarrival times in programs `ssq2`, `ssq3`, `sis3`, `sis4`

- If we know only that arrivals occur at random with a constant rate λ, the function `GetArrival` in `ssq2` and `ssq3` is appropriate
- If we know only that demand instances occur at random with a constant rate λ, the function `GetDemand` in `sis3` and `sis4` is appropriate

Generating Poisson random *variates*

Observation:

- If arrivals occur at random with rate λ=1
- the number of arrivals X in an interval of length μ will be a *Poisson(μ)* random variate (theorem 1)



exponential

```
a0 = 0.0;
x = 0;
while (a < μ) {
     a += Exponential(1.0);
     x++;
}
return x-1;
```

$a_5 > \mu$

Summary of Poisson arrival processes

Given a fixed time interval $(0, t)$, there are two ways of generating a realization of a stationary Poisson arrival process with rate λ

1. Generate the number of arrivals: n = *Poisson*(λt)
   Generate a *Uniform*(0,t) random variate sample of size n and sort to form $0 < a_1 < a_2 < a_3 < \ldots < a_n$

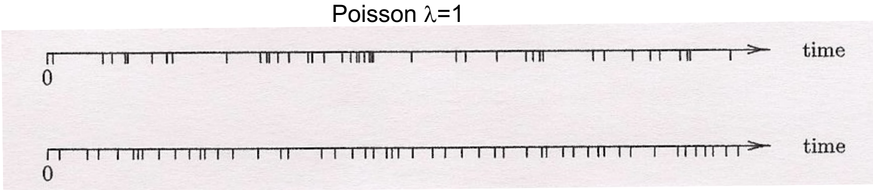
2. use algorithm 1 with *Exponential*($1/\lambda t$)

- Statistically, the two approaches are equivalent
- The first approach is computationally more expensive, especially for large n
- The second approach is always preferred

Summary of Poisson arrival processes

- The *mode* of the exponential distribution is 0
  → A stationary Poisson arrival process exhibits "clustering"

Poisson λ=1



Interarrival times *Erlang*(4,1/4)

The stationary Poisson arrival process generalizes to

- a stationary arrival process when exponential interarrival times are replaced by any continuous RV with positive support
- a nonstationary Poisson arrival process when λ varies over time

For our application framework, we will look at:

- arrival process model
- service process model

Service Process Models

differently from the case of arrival processes, there are no well-defined "default", only application-dependent guidelines:

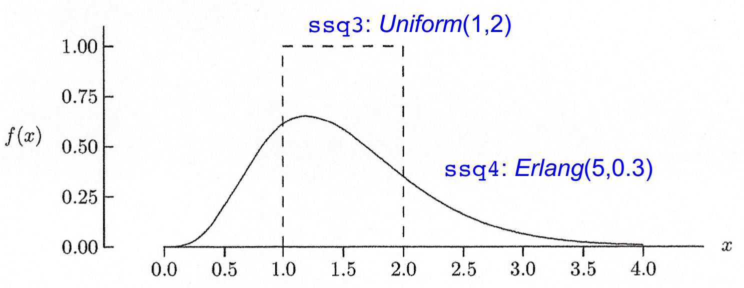
- *Uniform*(a, b) service times are usually inappropriate since they rarely "cut off" at a maximum value b
- Service times are positive, so they cannot be *Normal*(μ,σ) unless truncated to positive values
- Positive probability models "with tails", such as the *Lognormal*(a, b) distribution, are candidates
  - jobs UNIX
  - web file size
  - Internet topology
  - IP packet flow
  - …
- If service times are the sum of n iid *Exponential*(b) sub-task times, then the *Erlang*(n, b) model is appropriate

Program ssq4

- `ssq4` is based on program `ssq3`, but with a more realistic *Erlang*(5, 0.3) service time model
  The corresponding service rate is 2/3
- As in program `ssq3`, `ssq4` uses *Exponential*(2) random variate interarrivals.
  The corresponding arrival rate is 1/2

Example

- For both `ssq3` and `ssq4`, the arrival rate is λ=0.5 and the service rate is ν= 2/3 ≈ 0.667
- The distribution of service times for two programs is very different



first conjecture how the steady-state statistics will differ for these two programs and then investigate the correctness of your conjecture via discrete-event simulation

Example

- suppose using a *Normal*(1.5,2.0) random variable to model service times
- Truncate distribution so that
  - Service times are non-negative (a=0)
  - Service times are less than 4 (b=4)

```
α = cdfNormal(1.5, 2.0, a);         /* a is 0.0 */
β = 1.0 - cdfNormal(1.5, 2.0, b);   /* b is 4.0 */
```

- the result: $\alpha = 0.2266$ and $\beta = 0.1056$

NB
the *truncated Normal*(1.5,2.0) random variable has a mean of 1.85 (not 1.5) and a standard deviation of 1.07 (not 2.0)

Why is the mean increased?????

Constrained Inversion

once α and β are determined, the corresponding truncated random variate can be generated by using constrained inversion

```
u = Uniform(α, 1.0 - β);
return F⁻¹(u);
```

Exercise:

- generate n truncated random variates;
- compute sample mean and standard deviation (by Welford)
- verify for which n the statistics converge to the theoretical values

Example (cont.)

- The idf capability in `rvms` can be used to generate the *truncated Normal*(1.5,2.0) RV

```
α = 0.2266274;
β = 0.1056498;
u = Uniform(α, 1.0 - β);
return idfNormal(1.5, 2.0, u);
```

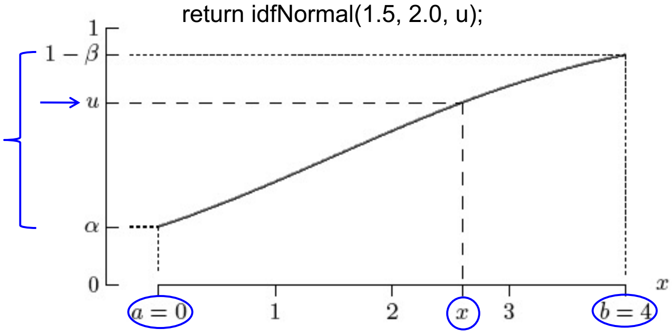


Figure shows u=0.7090 and x=2.601

Exercises

- Exercise 7.3.1