Queueing Theory and Simulation, lecture 12

Nicky van Foreest

June 22, 2021

Contents

1 Status and Plan

1.1 Plan

- Past:
  - $\mathsf{E}[W]$ for the $M/G/1$ queue, the so-called Pollaczek-Khinchine equation
  - Stationary distribution of $p(n)$ $(= \pi(n)$ by PASTA) of the $M^X/M/1$ queue
- Now:
  - Numerical methods to compute stationary distribution of $p(n)$ $(= \pi(n))$ of the $M^X/M/1$ queue.
  - Stationary distribution of $p(n)$ $(= \pi(n))$ of the $M/G/1$ queue.
- Next:
  - Simple queueing control

2 Computing the stationary distribution $p(n)$ for $M^X/M/1$

2.1 $M^X/M/1$ Model

- Jobs arrive as a Poisson process with rate λ.
- Job batch sizes are iid $\sim B$, $f(k) = \mathsf{P}[B = k]$, $G(k) = \mathsf{P}[B > k]$.
- Items are served individually
- Item service times iid $S \sim \text{Exp}(\mu)$.
- $c = 1$
- $\rho = \lambda \mathsf{E}[B]\ \mathsf{E}[S]$.

2.2 Recursion (recall)

$$\lambda \sum_{m=0}^{n} \pi(m) G(n-m) = \mu \pi(n+1) \tag{1}$$

- Which arguments have we used to derive this?
  - level-crossing
  - PASTA
  - rate-stability
- Watch out for off-by-one errors in formulas like this: $G(n-m)$ or $G(n-m-1)$ or $G(n-m+1)$, etc.

2.3 How to solve this recursion

- A first easy win:

$$\begin{aligned}
\mu \pi(n+1) &= \lambda \sum_{m=0}^{n} \pi(m) G(n-m) && (2)\\
&= \lambda \sum_{m=0}^{n} \pi(n-m) G(m) && (3)\\
&= \lambda \sum_{m=0}^{\min\{n,N\}} \pi(n-m) G(m) && (4)
\end{aligned}$$

where N is such that $G(N) = 0$.

- Why does this imply that $G(k) = 0$ for $k > N$?
- Of course N is finite since we are going to use the computer.

2.4 Method 1: code

Python Code

```python
import numpy as np

def compute_pi(f, M):
    pi = np.ones(M + 1)
    F = np.cumsum(f)
    G = np.ones_like(F) - F
    N = len(G)

    for n in range(M):
        R = sum(pi[n - m] * G[m] for m in range(min(n + 1, N)))
        pi[n + 1] = R * labda / mu # keep code on the slide
    return pi / pi.sum() # normalize
```

2.5 Method 1: results

Python Code

```
labda, mu = 1, 3
f = np.array([0, 1, 1, 1])
f = f / f.sum()

pi = compute_pi(f, M=10)
EL = sum(n * pi[n] for n in range(len(pi)))
print(EL)
```

2.6 Method 1, comparison to earlier work, $M = 10$

Python Code

```
EB = sum(k * fk for k, fk in enumerate(f))
EB2 = sum(k * k * fk for k, fk in enumerate(f))
VB = EB2 - EB * EB
C2 = VB / EB / EB
rho = labda * EB / mu
EL_exact = (1 + C2) / 2 * rho / (1 - rho) * EB
EL_exact += rho / (1 - rho) / 2
print(EL, EL_exact)
```

2.7 Method 2, better idea

Python Code

```
print(pi)
```

- Observation: $\pi(n) \to 0$ for $n \to \infty$.
- Idea: Stop when $\pi(n) < \epsilon$, for some $0 < \epsilon \ll 1$.

2.8 Method 2, code

Python Code

```
def compute_pi_2(f, eps):
    F = np.cumsum(f)
    G = np.ones_like(F) - F
    pi, n, N = {}, 0, len(G)
    pi[0] = 1

    while pi[n] > eps:
        R = sum(pi[n - m] * G[m] for m in range(min(n + 1, N)))
        pi[n + 1] = R * labda / mu
        n += 1

    norm = sum(pi[n] for n in pi.keys())
    return {n: p / norm for n, p in pi.items()}
    #return {n: pi[n] / norm for n in pi.keys()}
```

2.9 Method 2: results

Python Code

```python
pi = compute_pi_2(f, eps=0.001)
# EL = sum(n * pi[n] for n in pi.keys())
EL = sum(n * p for n, p  in pi.items())
print(EL, EL_exact, len(pi))
```

2.10 Observations

- Just setting M to some value is not smart; no guarantee on quality.
- Set some ϵ is better.
  - We run op to $n = 29$, which is doable (for a computer)
  - But we still don't have a guarantee on the quality of the estimation of $\mathsf{E}[L]$.

2.11 Improvement strategy

Python Code

```python
M, EL_old, EL, eps = 0, 0, 1, 1 / 1000

while abs(EL - EL_old) > eps:
    EL_old = EL
    M += 10
    pi = compute_pi(f, M)
    EL = sum(n * pi[n] for n in range(len(pi)))
    print(M, EL)
```

2.12 Better idea, prevent (unreasonably) large queues

- What does $n = 29$ mean: A pretty big queue. Perhaps unreasonably big.
- Consider $M^X/M/1/K$.
- How to 'chop off' batches that 'stick out'?

2.13 Complete rejection

- Whenever an arriving batch does not arrive in its entirety, reject it.
- To cross level n when at state m: $B > n - m$.
- To fit in when at state m: $B \leq K - m$.
- Watch out for off-by-one errors!

$$\mu\pi(n+1) = \lambda \sum_{m=0}^{n} \pi(m)\, \mathsf{P}[n - m < B \leq K - m] \tag{5}$$

3 Stationary distribution $p(n)$ for $M/G/1$

3.1 $M/G/1$ Model and notation

- Jobs arrive as a Poisson process with rate λ.
- Job service times iid $S \sim F$.
- Y is the number of jobs that arrive during a service time.
- The M in $M/G/1$ implies $Y|S \sim \text{Poi}(\lambda S)$.
- By LOTP:

$$f(j) = \mathsf{P}[Y = j] = \int_0^\infty \mathsf{P}[Y = j \,|\, S = x]\, F(\,\mathrm{d}x) \tag{6}$$

$$= \int_0^\infty e^{-\lambda x} \frac{(\lambda x)^j}{j!} F(\,\mathrm{d}x). = \int_0^\infty e^{-\lambda x} \frac{(\lambda x)^j}{j!} f(x)\,\mathrm{d}x. \tag{7}$$

- $F(\,\mathrm{d}x) = f(x)\,\mathrm{d}x$ if S a continous rv, but not if S makes jumps.
- $G(n) = \mathsf{P}[Y > n]$ is the survivor function.

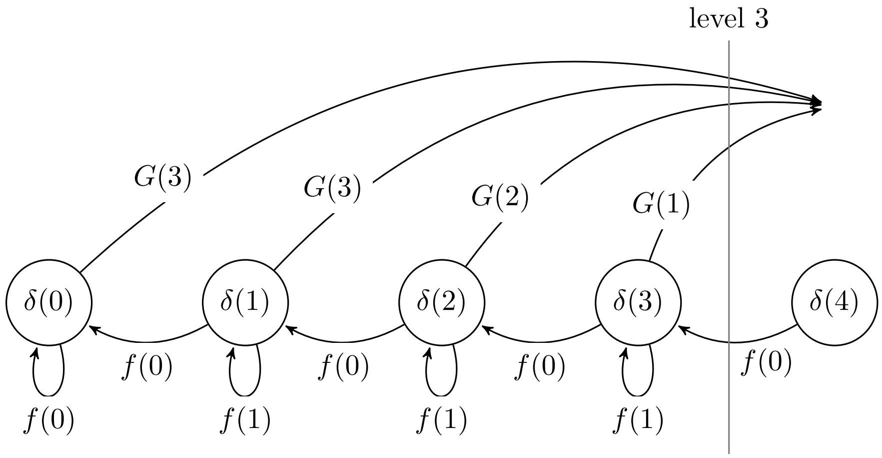
3.2 Crossing of level $n = 0$

- Focus on job departure moments
- Downcrossing occurs if in state 1 and $Y = 0$. Why?
- An upcrossing occurs if in state 0 and $Y \leq 1$. Why?
  - What does it mean when job k sees the system in state 0 upon arrival? Job $k - 1$ left an empty system behind.
  - What does it mean when $Y = 1$ That job $k+1$ came in while serving job k. Hence, $L(D_k) = 1$.
  - Hence? An upcrossing occured!

Level crossing:

$$\pi(1)f(0) = \pi(0)\,\mathsf{P}[Y \geq 1] = \pi(0)G(0). \tag{8}$$

3.3 Crossing of level $n > 0$



3.4 Result

- Recursion:

$$\delta(n+1)f(0) = \delta(0)G(n) + \sum_{m=1}^{n} \delta(m)G(n+1-m). \tag{9}$$

- For the $G/G/1$ queue, $\delta(n) = \pi(n)$.
- Hence

$$\pi(n+1)f(0) = \pi(0)G(n) + \sum_{m=1}^{n} \pi(m)G(n+1-m). \tag{10}$$

3.5 Computation

- Initially set $\pi(0) = 1$.
- Use some smart method to determine N.
- Solve the recursion for $n = 1, \ldots, N$.
- Normalize: $\alpha = \sum_{n=0}^{N} \pi(n)$. $\pi(n) /= \alpha$ (in python notation).