Queueing Theory: Applications of Sakasegawa's formula

EBB074A05

Nicky D. van Foreest

2022:01:19

1 General info

This file contains the code and the results that go with the YouTube movies mentioned in the README file in this directory.

You should read the relevant section of my queueing book to understand what it going on. Most of it is very easy, but without background a bit cryptic (I believe).

I included a number of exercises to help you think about the code. Keep your answers short; you don't have to win the Nobel prize on literature.

2 Various models compared

We have (at least) three different types of models for a queueing system: simulation in discrete time, simulation in continuous, and Sakasegawa's formula to compute the expected waiting time in queue. Let's see how these models compare.

2.1 Discrete time simulation

We have machine that can serve c_k jobs on day k. When the period length is T, then $c_k \sim \text{Pois}(\mu T)$. The number jobs that arrive on k is $a_k \sim \text{Pois}(\lambda T)$. The arrivals in period k cannot be served on day k. The code to simulate this should be familiar to you by now.

Python Code

```
import numpy as np

np.random.seed(3)

labda = 3
mu = 1.2 * labda
T = 8  # period length, 8 hours in a day

num = 1000

a = np.random.poisson(labda * T, size=num)
c = np.random.poisson(mu * T, size=num)
L = np.zeros_like(a, dtype=int)

L[0] = 5
for i in range(1, num):
    d = min(c[i], L[i - 1])
    L[i] = L[i - 1] + a[i] - d
```

```
19
20
21 print(L.mean(), L.std())
```

Ex 2.1. Run this code (and write down the results). Then change the code such that the arrivals can be served on the day they arrive. Rerun and compare the results.

Ex 2.2. Now change the period time, which was 8 hours in a day, to $T = 1$ (so that we concentrate on what happens during an hour instead of a day). Rerun the code with and without the arrivals being served on the period of arrival, and compare to the previous exercise. Explain the difference.

Ex 2.3. What are the advantages and disadvantages of using small values for T?

2.2 Contiuous time simulation

Here is the code.

Python Code

```python
import numpy as np

np.random.seed(3)

labda = 3
mu = 1.2 * labda
num = 1000
X = np.random.exponential(scale=1 / labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
S = np.random.exponential(scale=1 / mu, size=len(A))
S[0] = 0
D = np.zeros_like(A)

for k in range(1, len(A)):
    D[k] = max(D[k - 1], A[k]) + S[k]

J = D - A
print(labda * J.mean())
```

Ex 2.4. Which result did we use to compute $\mathsf{E}[L]$? Can we use that to estimate $\mathsf{V}[L]$?

Ex 2.5. Run the code and compare with the discrete time simulation.

2.3 Sakasegawa's formula

Here we use Sakasegawa's formula to compute $\mathsf{E}[L]$.

Python Code

```python
import numpy as np


def sakasegawa(labda, ES, Ca2=1, Cs2=1, c=1):
    rho = labda * ES
    V = (Ca2 + Cs2) / 2
    U = rho ** (np.sqrt(2 * (c + 1)) - 1) / (1 - rho)
    T = ES / c
    return V * U * T
```

```
10
11
12  labda = 3
13  mu = 1.2 * labda
14  ES = 1 / mu
15  W = sakasegawa(labda, 1 / mu, 1, 1, 1)
16  L = labda * (W + ES)
17  print(L)
```

Ex 2.6. Explain the code, in particular the values of the parameters.

Ex 2.7. Run the code, and compare the value with the discrete and continuous time simulations.

Ex 2.8. What are advantages and disadvantages of using Sakasegawa's formula?

3 Server Adjustments

Consider now a queueing system in which the server needs an adjustments with probability p (see the section on server adjustments in the book). The repair times are assumed constant, at first, with mean 2. Here is the simulator.

3.1 Check work

First we should check the formulas for $\mathsf{E}[S]$ and $\mathsf{V}[S]$.

Python Code

```
1   import numpy as np
2
3   np.random.seed(3)
4
5   labda = 3
6   mu = 2 * labda
7   ES0 = 1 / mu
8   p = 1 / 20
9   num = 10000
10
11
12  S0 = np.random.exponential(scale=ES0, size=num)
13  R = 2 * np.ones_like(S0)
14  I = np.random.uniform(size=len(R)) <= p
15  S = S0 + R * I
16  print(S.mean(), S.var())
17
18  ER = R.mean()
19  ES = ES0 + p * ER
20  VS0 = ES0 * ES0
21  VR = R.var()
22  VS = VS0 + p * VR + p * (1 - p) * ER * ER
23  print(ES, VS)
```

Ex 3.1. Explain what is `I`. Then explain the line `S0 + R * I` works.

Ex 3.2. Run the code; include the numbers in your assignment. Are the numbers nearly the same?

3.2 The simulations

Now that we checked the formulas to compute $\mathsf{E}[S]$ and $\mathsf{V}[S]$, we can see how well Sakasegawa's formula applies for a queueing system in which a server requires regular, but random, adjustments.

Python Code

```
import numpy as np

np.random.seed(3)

labda = 3
mu = 2 * labda
p = 1 / 20
num = 10000

X = np.random.exponential(scale=1 / labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
S = np.random.exponential(scale=1 / mu, size=len(A))
R = 2 * np.ones(len(S))
I = np.random.uniform(size=len(S)) <= p
D = np.zeros_like(S)

for k in range(1, len(A)):
    D[k] = max(D[k - 1], A[k]) + S[k] + R[k] * I[k]

W = D - A - S
print(W.mean())
```

Ex 3.3. Explain how `D` is computed.

To see how the approximation works, glue the next code below the previous code.

Python Code

```
def sakasegawa(labda, ES, Ca2=1, Cs2=1, c=1):
    rho = labda * ES
    V = (Ca2 + Cs2) / 2
    U = rho ** (np.sqrt(2 * (c + 1)) - 1) / (1 - rho)
    T = ES / c
    return V * U * T


ES0 = 1 / mu
VS0 = ES0 * ES0
ER = R.mean()
ES = ES0 + p * ER
rho = labda * ES
assert rho < 1, "rho >= 1"
VR = R.var()
VS = VS0 + p * VR + p * (1 - p) * ER * ER
Cs2 = VS / ES / ES
W = sakasegawa(labda, ES, 1, Cs2, 1)
print(W)
```

Ex 3.4. What does the `assert` statement? Why is it there? Check what happens if you would set `labda=10` and `mu=3`.

Ex 3.5. To test the code I set at first `R = 0 * np.ones(len(A))`. Why is this a good test?

Ex 3.6. Now run the code and compare the answers. Then set `num = 100000`, i.e., 10 times as large. What is the effect?

Ex 3.7. Now set `R = np.random.exponential(scale=2, size=len(A))`. What does this change mean? What is the effect on $\mathsf{E}[W]$?

Ex 3.8. What is the model behind this code?

Python Code

```
import numpy as np

np.random.seed(3)

labda = 3
mu = 4
N = 1000

X = np.random.exponential(scale=1 / labda, size=N)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
S = np.random.exponential(scale=1 / mu, size=len(A))
R = np.random.uniform(0, 0.1, size=len(A))

D = np.zeros_like(A)
for k in range(1, len(A)):
    D[k] = max(D[k - 1], A[k]) + S[k] + R[k]

W = D - A - S
print(W.mean(), W.std())
```

Ex 3.9. In stead of

Python Code

```
for k in range(1, len(A)):
    D[k] = max(D[k - 1], A[k]) + S[k] + R[k] * I[k]
```

we could write

Python Code

```
for k in range(1, len(A)):
    D[k] = max(D[k - 1] + R[k] * I[k], A[k]) + S[k]
```

What modeling choice would this change reflect? Which of these two models makes the sojourn smaller?

4 Server failures

This time we focus on a server that can fail; again check the queueing book for the formulas. Here we just implement them.

4.1 Check work

Again, first we need to check that our (implementtion of the) formulas for $\mathsf{E}[S]$ and $\mathsf{V}[S]$ are correct.

Python Code

```python
import numpy as np
from scipy.stats import expon

np.random.seed(3)

labda = 3
mu = 2 * labda
ES0 = 1 / mu
labda_f = 2
ER = 0.5
num = 10000

S0 = np.random.exponential(scale=ES0, size=num + 1)
N = np.random.poisson(labda_f * ES0, len(S0))
R = expon(scale=ER)
S = np.zeros_like(S0)
for i in range(len(S0)):
    S[i] = S0[i] + R.rvs(N[i]).sum()

A = 1 / (1 + labda_f * ER)
ES = ES0 / A
print(ES, S.mean(), S0.mean() + N.mean() * R.mean())

C02 = 1
Cs2 = C02 + 2 * A * (1 - A) * ER / ES
print(Cs2, S.var() / (S.mean() ** 2))
```

Ex 4.1. Run this code, and check the result. Then chance `num` to 100000 to see that the estimate improves.

Ex 4.2. Explain how we compute `S[i]`.

4.2 The simulations

Here is all the code to compare the results of a simulation to Sakasegawa's formula.

Python Code

```python
import numpy as np
from scipy.stats import expon

np.random.seed(3)

labda = 2
mu = 6
ES0 = 1 / mu
labda_f = 2
ER = 0.5
num = 10000

S0 = np.random.exponential(scale=ES0, size=num + 1)
N = np.random.poisson(labda_f * ES0, len(S0))
R = expon(scale=ER)
S = np.zeros_like(S0)
for i in range(len(S0)):
    S[i] = S0[i] + R.rvs(N[i]).sum()


```

```python
X = np.random.exponential(scale=1 / labda, size=num)
A = np.zeros(len(X) + 1)
A[1:] = X.cumsum()
D = np.zeros_like(A)
for k in range(1, len(A)):
    D[k] = max(D[k - 1], A[k]) + S[k]

W = D - A - S
print(W.mean())


def sakasegawa(labda, ES, Ca2=1, Cs2=1, c=1):
    rho = labda * ES
    V = (Ca2 + Cs2) / 2
    U = rho ** (np.sqrt(2 * (c + 1)) - 1) / (1 - rho)
    T = ES / c
    return V * U * T


A = 1 / (1 + labda_f * ER)
ES = ES0 / A
C02 = 1
Cs2 = C02 + 2 * A * (1 - A) * ER / ES
rho = labda * ES
assert rho < 1, "rho >= 1"
W = sakasegawa(labda, ES, 1, Cs2, 1)
print(W)
```

Ex 4.3. Run the code and include the results in your assignment.

Ex 4.4. Suppose you can choose between two alternative ways to improve the system. Increase λ_f, and decrease $\mathsf{E}[R]$, but such that $\lambda_f \mathsf{E}[R]$ remains constant; or the other way around, decrease λ_f and increase $\mathsf{E}[R]$. Which alternative has better influence on $\mathsf{E}[W]$? (Use Sakasegawa's formula for this; you don't have to do the simulations. In general, computing functions is much faster than simulation.)

5 A simple tandem network

We have two queues in tandem. Again we compare the approximations with simulation.

5.1 Simulation in continuous time

This code simulates two queues in tandem in continuous time.

Python Code

```python
import numpy as np

np.random.seed(4)

labda = 3
mu1 = 1.2 * labda
num = 100000
X = np.random.exponential(scale=1 / labda, size=num)
A1 = np.zeros(len(X) + 1)
A1[1:] = X.cumsum()
S1 = np.random.exponential(scale=1 / mu1, size=len(A1))
```

```python
D1 = np.zeros_like(A1)

for k in range(1, len(A1)):
    D1[k] = max(D1[k - 1], A1[k]) + S1[k]

W1 = D1 - A1 - S1

# queue two
A2 = D1
mu2 = 1.1 * labda
S2 = np.random.exponential(scale=1 / mu2, size=len(A2))
D2 = np.zeros_like(A2)

for k in range(1, len(A2)):
    D2[k] = max(D2[k - 1], A2[k]) + S2[k]

W2 = D2 - A2 - S2

J = D2 - A1
print(W1.mean(), S1.mean(), W2.mean(), S2.mean(), J.mean())
```

Ex 5.1. Explain how the code works. Why did I choose these parameters?

5.2 Sakasegawa's formula plus tandem formula

Now we can check the quality of the approximations.

Python Code

```python
import numpy as np


def sakasegawa(labda, ES, Ca2=1, Cs2=1, c=1):
    rho = labda * ES
    V = (Ca2 + Cs2) / 2
    U = rho ** (np.sqrt(2 * (c + 1)) - 1) / (1 - rho)
    T = ES / c
    return V * U * T


labda = 3
mu1 = 1.2 * labda
ES1 = 1 / mu1
rho1 = labda * ES1
Ca2 = 1
Cs2 = 1
W1 = sakasegawa(labda, ES1, Ca2, Cs2, 1)
Cd2 = rho1 ** 2 * Ca2 + (1 - rho1 ** 2) * Cs2

Ca2 = Cd2
mu2 = 1.1 * labda
ES2 = 1 / mu2
Cs2 = 1
W2 = sakasegawa(labda, ES2, Ca2, Cs2, 1)


J = W1 + ES1 + W2 + ES2
print(W1, ES1, W2, ES2, J)
```

Ex 5.2. How does the code work? Compare the results obtained from simulation and by formula.

Ex 5.3. Assume that the service times at the first queue are constant and equal to $1/\mu_1$. What should be the parameter values for Sakasegawa's formula (explain). Then run both codes and comment on the results.

6 Hints

h.2.1. Replace the relevant line by `d = min(c[i], L[i - 1] + a[i])`.

h.3.9. What is the influence on the setup? Do we still require that the setup has to be done immediately before a service starts?

h.4.4. For instance, set `labda_f = 4` and `ER = 0.25`.

h.5.1. For what situations are the formulas exact?

h.5.3. In the simulation, replace the line with `S1` by `S1 = np.ones(len(A1)) / mu1`. In the formula's, don't forget to update C_s^2 for the relevant queue.