

CARLETON UNIVERSITY  
SCHOOL OF  
MATHEMATICS AND STATISTICS  
HONOURS PROJECT



TITLE: Approximation of Queueing Models

AUTHOR: Lucas Pellegroms

SUPERVISOR: Dr. Gennady Shaikhet

DATE: April 25<sup>th</sup> , 2019



**CONSENT FOR DISCLOSURE OF FOUR YEAR HONOURS PROJECT**

**I authorize the**

<b>School of Mathematics and Statistics</b>
<b>Office/Program/Individual</b>

**To use my Four Year Honours Project**

**Submitted on**

<b>April 25<sup>th</sup> , 2019</b>
<b>Date submitted to Honours Coordinator</b>

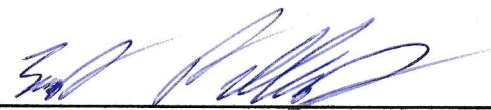
**For the purpose of**

<b>Education</b>
<b>State specific purpose of information release</b>

**In the period**

<b>Indefinite</b>
<b>State date range for which permission will exist</b>

<b>Full Name:</b>	<b>Lucas Pellegroms</b>
<b>Student I.D. #:</b>	<b>100982087</b>
<b>Date:</b>	<b>April 25<sup>th</sup> , 2019</b>

**Signature** 

**Protection of Privacy**  
The personal information requested on this form is collected under the authority of Section 42 (R.S.O.) 1990, c. F.31) of the *Freedom of Information and Protection of Privacy Act* and will be protected under Part 3 of that *Act*. It will be used for the purpose of managing the consent for disclosure of personal information process. Direct any questions about this collection to: [contact position, full address, and business telephone number].

# 1 Introduction

To begin, I would first like to cover all necessary background knowledge that will be helpful in understanding our queueing networks later on. Firstly, I will introduce the concept of Brownian Motion Processes that will be helpful in understanding the behaviour of our systems and random walks in general.

**Definition 1.1** (Standard Brownian Motion Process). .

A process  $W(t), t \geq 0$  is said to be a **Standard Brownian Motion** process if;

1.  $W(0) = 0$
2.  $W(t)$  is continuous.
3.  $W(t)$  has independent and stationary increments.
4.  $W(t) \sim N(0, t), \quad \forall t > 0.$

This gives us that for each independent interval  $[t, t + s]$  we have

$$W(t + s) - W(s) = W_s \sim N(0, s).$$

A standard Brownian Motion can also be generalized to a *Brownian Motion process*.

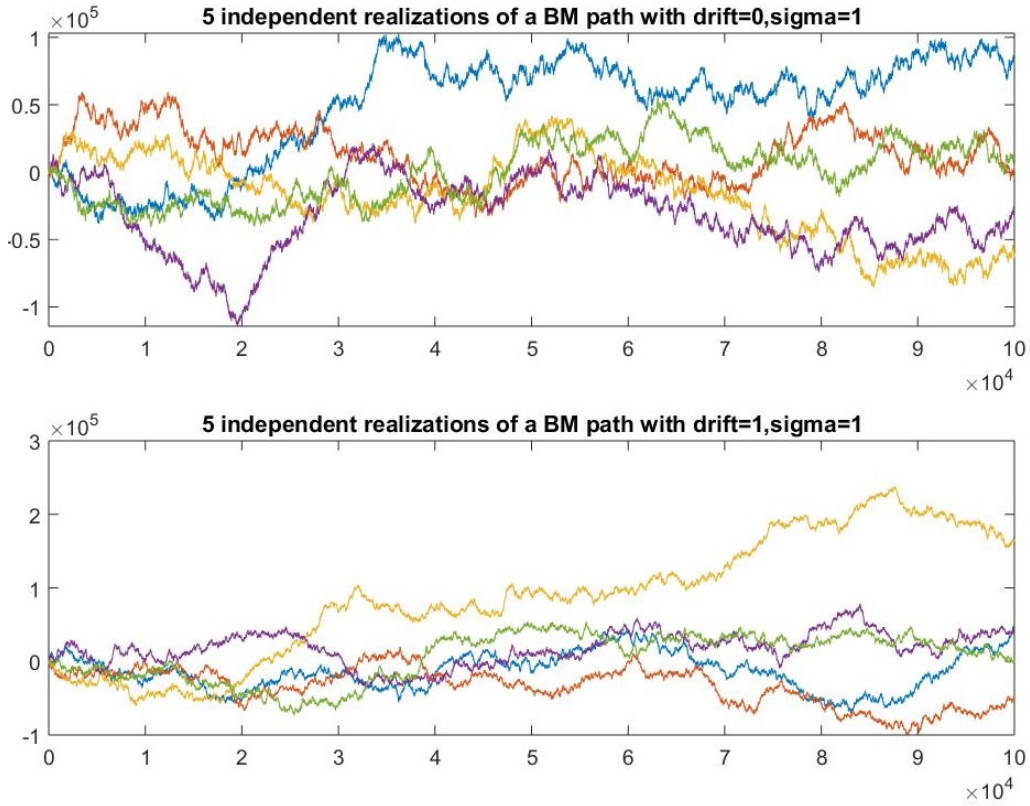
**Definition 1.2** (Brownian Motion Process). .

A process  $B = \{B(t), t \geq 0\}$  is said to be a **Brownian Motion** process (sometimes called a BM) if

$$B(t) = B(0) + \theta t + \sigma W(t)$$

Where  $W(t)$  is a *standard Brownian Motion*,  $\theta$  is our *drift* coefficient and  $\sigma$  is our *variance*.

Here, If we let  $B(0) = 0, \theta = 0$  and  $\sigma = 1$  then we recover our *standard Brownian Motion*.



(See codes 'BrownianMotion.m' and 'PlotBrownianMotion.m' in appendix)

Hopefully with these notions defined and the graphs above depicting their behaviour it is now easier to see that a Brownian motion is actually just a random walk with

$$B(t) = \sum_{i=1}^t X_i$$

where  $X_i$ 's are independent Normal random variables. In order to extend this to the continuous case all we do is choose our intervals to be  $(\Delta t, \Delta s)$  and define the continuous process

$$X(t) = \Delta s(X_1 + \dots + X_{\lfloor \frac{t}{\Delta t} \rfloor})$$

with  $\Delta s = \sigma\sqrt{\Delta t}$  in order to assure convergence to a continuous process.

**Definition 1.3** (Almost Surely convergence). .

We say that  $X_n$  converges almost surely (or with probability one) to  $X$  as  $n \rightarrow \infty$  if  $P(\lim_{n \rightarrow \infty} \|X_n - X\| = 0) = 1$ , and denote it by  $X_n \xrightarrow{a.s.} X$ .

Next, I will go over a few useful definitions and theorems that will be useful in analysing the behaviour of our queues.

**Definition 1.4** (Uniform convergence on all compact sets). .

If for each  $\omega \in \Omega$  we have that  $X_n(\omega)$  converges to  $X(\omega)$  uniformly in  $[0, T]$  as  $n \rightarrow \infty$  then  $X_n$  converges uniformly on all compact sets (u.o.c) to  $X$  as  $n \rightarrow \infty$  and denote it by  $X_n \rightarrow X$ , u.o.c.

**Definition 1.5** (Convergence in distribution). .

If  $G$  is a proper cdf and  $\lim_{n \rightarrow \infty} F_{X_n}(x) = F_Z(x)$  is a function that matches  $G$  at all points of continuity then  $X_n$  converges in distribution to  $G$  denoted by  $X_n \xrightarrow{d} G$ .

We will next introduce *Donsker's theorem* which is a functional extension of the central limit theorem and will help us relate the limit of a scaled i.i.d. summation to a standard Brownian motion.

**Theorem 1.1** (Donsker's Theorem). .

Let  $X_1, X_2, \dots$  be a sequence of i.i.d. random variables with mean  $\mu$  and variance  $\sigma^2$  such that  $\mu, \sigma^2 < \infty$ . Then for each  $n \geq 1$ , we define

$$X_n(t) = \frac{1}{\sqrt{n\sigma^2}} \sum_{i=1}^{\lfloor nt \rfloor} [X_i - \mu]$$

where  $\sum_{i=1}^{\lfloor nt \rfloor} [X_i - \mu] = 0$  for  $nt < 1$ . Now,

$$X \xrightarrow{d} W, \text{ as } n \rightarrow \infty$$

such that  $W$  is a standard Brownian motion as we wanted.

**Example 1.1.**

Define  $X_i, i = 1, \dots, n$  to be a sequence of independent, identically distributed exponential random variables (i.e.  $X_i \sim \text{expo}(\lambda)$ ). We can then define a process  $X(t)$  to be the sum of our exponential r.v.s up to time  $t$  (i.e.  $X(t) = \sum_{i=1}^t X_i$ ). In this case, by the previous theorem the inverse process  $Y(t)$  actually turns out to be a poisson process with rate  $\lambda$ !

For the context of the following Lemma which we will state without proof, define  $X$  to be a non-decreasing process, and denote  $Y$  to be its inverse such that

$$Y(t) = \sup\{s \geq 0 : X(s) \leq t\}, \quad 0 \leq t \leq \infty$$

**Lemma 1.1.** .

*Consider the pair  $(X, Y)$  as defined above and suppose*

$$\frac{X(t)}{t} \xrightarrow{a.s.} m, \quad \text{as } t \rightarrow \infty$$

*for some  $m > 0$  and set  $\mu = \frac{1}{m}$ .*

$$\Rightarrow \frac{Y(t)}{t} \xrightarrow{a.s.} \mu, \quad \text{as } t \rightarrow \infty$$

*As  $n \rightarrow \infty$*

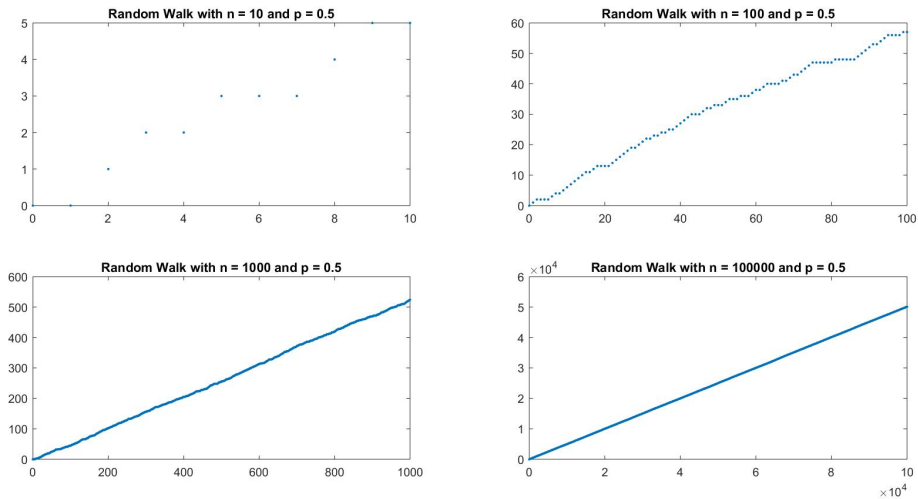
$$\bar{X}^n(t) = \frac{1}{n}X(nt) \xrightarrow{a.s.} mt, \quad \text{u.o.c.},$$

$$\bar{Y}^n(t) = \frac{1}{n}Y(nt) \xrightarrow{a.s.} \mu t, \quad \text{u.o.c.}$$

This lemma shows us that  $\bar{X}^n(t)$  and  $\bar{Y}^n(t)$  are processes that use time scaling (multiplying by  $n$ ) and space scaling (dividing by  $n$ ). We will next introduce some notation for similar processes that involve the same time scaling but instead use  $\sqrt{n}$  as a space scale.

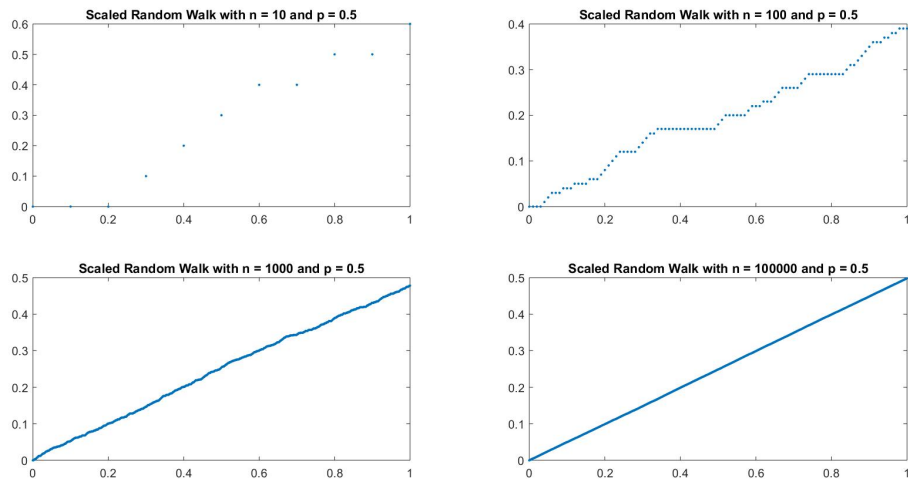
$$\hat{X}^n(t) = \sqrt{n}[\bar{X}^n(t) - \bar{X}(t)], \quad \text{and}$$

$$\hat{Y}^n(t) = \sqrt{n}[\bar{Y}^n(t) - \bar{Y}(t)]$$



(See code 'RandomWalk.m' in appendix)

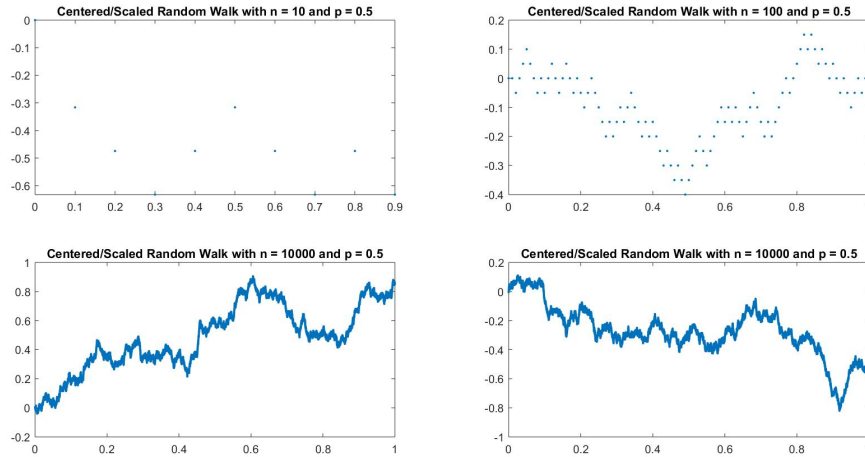
Note here that as we increase  $n$  our "Random Walk" just turns into a straight line (uninteresting) and we a large range of data modelled over a prolonged period of time (not super useful).



(See code 'ScaledRandomWalk.m' in appendix)

Now that we have scaled our Random Walk we now have just as much data

but modelled over a much smaller range and within a shorter time period!  
 (Still looks uninteresting but an improvement none the less)



(See code 'CenteredScaledRandomWalk.m' in appendix)

Wow! Now that we have centred our data we finally have something that is starting to look interesting (sort of looks like Brownian motion) and have it nicely plotted over a reasonable range within a short time period!

**Theorem 1.2** (Functional Strong Law of Large Numbers). (*FsLLN*).  
 Assuming that  $X_i$  has mean  $m > 0$ . Then as  $n \rightarrow \infty$

$$(\bar{X}^n, \bar{Y}^n) \xrightarrow{a.s.} (\bar{X}, \bar{Y}), \text{ u.o.c.}$$

**Theorem 1.3** (Functional Central Limit Theorem). (*FCLT*).  
 Assuming that  $X_i$  has variance  $\sigma^2 < \infty$ . Then,

$$(\hat{X}^n, \hat{Y}^n) \xrightarrow{d} (\hat{X}, \hat{Y}),$$

Where  $\hat{X}(t) = \sigma W(t)$ ,  $\hat{Y}(t) = -\mu \hat{X}(\mu t)$ , and  $W$  is a standard Brownian motion.

**Proof.**

From *Theorem 1.1 (Donsker's Theorem)* we have that

$$\hat{X}^n = \sqrt{n}[\bar{X}^n - \bar{X}] \xrightarrow{d} \hat{X}$$



as  $n \rightarrow \infty$ . Now by applying the *Theorem 1.2 (FSLLN)* gives us that

$$(\bar{X}^n, \bar{Y}^n) \xrightarrow{a.s.} (\bar{X}, \bar{Y}), \text{ u.o.c.}$$

as  $n \rightarrow \infty$ . The random time-change theorem also tells us that under certain conditions if  $(X_n, Y_n) \rightarrow (X, Y)$  then  $X_n(Y_n) \rightarrow X(Y)$  where  $X(Y)$  is  $X$  evaluated at  $Y$ . It can be shown that in our current situation, said conditions have been met and therefore,

$$\hat{X}^n(\hat{Y}^n) \xrightarrow{d} \hat{X}(\bar{Y})$$

Next, recall from *Lemma 1.2* that

$$\begin{aligned} \bar{X}(t) &= mt \\ \Rightarrow \bar{Y}^n &\xrightarrow{d} \hat{Y} \\ \text{if, } \sqrt{n}[\bar{X}^n(\bar{Y}^n(t)) - t] &\xrightarrow{a.s.} 0 \end{aligned}$$

where the last line follows from  $|\bar{X}^n(\bar{Y}^n(t)) - t| \leq \frac{1}{n}, \quad \forall t \geq 0$ .

## 2 G/G/1 Queueing

Throughout this section we will explore some interesting facts and consequences of single stage queueing with generally distributed arrival rates and generally distributed service rates; without hesitation let us begin with introducing some helpful notation

Define the sequence  $\{u_i, i = 1, 2, \dots\}$  to be the inter-arrival times and  $\{v_i, i = 1, 2, \dots\}$  to be the length of the job requirements such that

$$\begin{aligned} U(0) &= 0, \quad V(0) = 0 \\ U(k) &= \sum_{i=1}^k u_i, \quad V(k) = \sum_{i=1}^k v_i, \quad k \geq 1. \end{aligned}$$

**Definition 2.1** (Arrival Process). .

We define,

$$A(t) = \sup\{k; U(k) \leq t\}$$

to be the number of arrivals up up to time  $t$ .

**Definition 2.2** (Service Process). .

We define,

$$S(t) = \sup\{k; V(k) \leq t\}$$

to be the number of jobs that can **potentially** be completed during the first  $t$  units of service time.

Now that we have defined our Arrival and Service processes we can introduce  $Q(t)$ ; the number of jobs currently in the system at time  $t$  as

$$Q(t) = Q(0) + A(t) - S(B(t))$$

where  $B(t) = \int_0^t I_{\{Q(s) > 0\}} ds$  is the busy time process (throws away idle time).

Now define,

$$\begin{aligned} X(t) &= Q(0) + A(t) - S(B(t)) + \mu[B(t) - t] \\ Y(t) &= \mu[t - B(t)] \end{aligned}$$

such that  $Y(t)$  is the cumulative idle time which satisfies  $Q(t)dY(t) = 0$  and notice that we can rewrite  $Q(t)$  as

$$\begin{aligned} Q(t) &= Q(0) + A(t) - S(B(t)) \\ &= Q(0) + A(t) - S(B(t)) + \mu[B(t) - t] - \mu[B(t) - t] \\ &= Q(0) + A(t) - S(B(t)) + \mu[B(t) - t] + \mu[t - B(t)] \\ &= X(t) + Y(t) \end{aligned}$$

and let us examine these forms further in attempts to be able to better understand them, and help with modelling.

$Q(0)$  : is our initial state space (amount of people in the system at time 0)

$A(t)$  : is our arrival process as defined previously

$S(t)$  : is our service process as defined previously, however we are not taking

$S(t)$  here, instead we are taking  $S(B(t))$

$B(t)$  : is our busy time process which is the integral of the indicator function for when our queue is non-zero

Which means that  $S(B(t))$  is the number of jobs that can potentially be completed during the first  $t$  units of time for which the queue is non-zero

$$\begin{aligned} \text{i.e. } S(B(t)) &= \sup\{k; V(k) \leq B(t)\} \\ &= \sup\{k; V(k) \leq \int_0^t I_{\{Q(s) > 0\}} ds\} \end{aligned}$$

In order to properly model our queue we will be required to model  $S(B(t))$  which seems to be quite nasty. In the following sections we will further examine the G/G/1 queueing system in hopes of coming up with intuitive approaches to better understanding and modelling it.

## 2.1 Approximating the G/G/1 Queue

For the purposes of this queueing system we shall rewrite the queueing process as

$$\begin{aligned} Q(t) &= Q(0) + A(t) - S(B(t)) \\ &= X(t) + Y(t) \end{aligned}$$

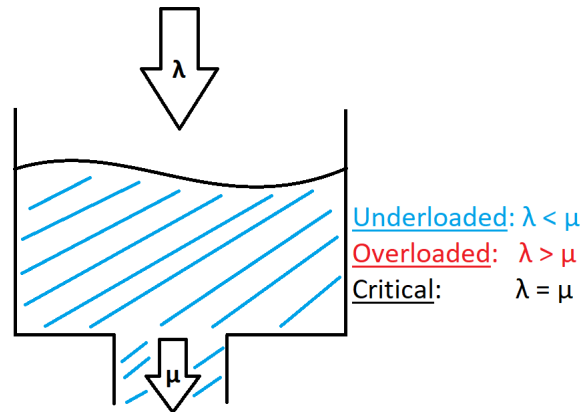
such that,

$$\begin{aligned} X(t) &= Q(0) + [A(t) - \lambda t] - [S(B(t)) - \mu B(t)] + (\lambda - \mu)t \\ Y(t) &= \mu[t - B(t)] \end{aligned}$$

Where once again,  $B$  is the busy time process and  $Y/\mu$  is the cumulative idle time, and so we have that  $Q(t)dY(t) = 0$ .

To help motivate the following section, consider our system to be a tub that is filled by a tap and empties out a drain. In this case our arrival rate will be the rate at which liquid comes from the tap into the tub; and our departure rate is the rate at which the liquid leaves the tub through the drain. Here we will have a very fluid cycle of system states, liquid constantly entering and leaving with an excess of liquid pooling in the tub. This type of system is precisely what **fluid limits** is used for to help analyse.

### 2.1.1 G/G/1 - Fluid Limits



With this simple diagram we can see that when the arrival rate ( $\lambda$ ) exceeds the service(or departure) rate ( $\mu$ ) that the system will be **overloaded** and our tub will overflow.

Or, when our service rate exceeds the arrival rate our system will be **underloaded** and the tub will consistently be empty.

And lastly, when both rates are equal we will have a **critical** system and the total amount of fluid in the system will remain roughly constant.

In this simple example we are observing a small tub over a short amount of time to see what will happen which is rather trivial and quite frankly unimportant. So suppose that we are interested analysing the long term behaviour of a more complex system  $\mathbf{Q}(\mathbf{nt})$ , and consider scaling **time** as in *Lemma 1.1* such that

$$\bar{Q}^n(t) = \frac{Q(\mathbf{nt})}{\mathbf{n}}$$

where we know that

$$\bar{A}^n(t) = \frac{A(\mathbf{nt})}{\mathbf{n}} \rightarrow \lambda t, \quad \bar{S}^n(t) = \frac{S(\mathbf{nt})}{\mathbf{n}} \rightarrow \mu t$$

from the FLLN(*Theorem 1.2*). Which leads us to suspect that

$$\bar{Q}^n(t) \rightarrow (\bar{Q}(0) + (\lambda - \mu)t)^+ \quad (\text{How?})$$

Now instead of trying to better understand the long term behaviour by scaling time, let's try scaling our **rates** as in *Lemma 1.1* such that

$$\lambda^n = \mathbf{n}\lambda, \quad \mu^n = \mathbf{n}\mu$$

however, it's still not completely clear why

$$\bar{Q}^n(t) \rightarrow (\bar{Q}(0) + (\lambda - \mu)t)^+$$

To help better understand this let's break it down piece by piece. Logically, over time we would expect our system to have what we started with ( $\bar{Q}^n(0)$ ) plus all of our arrivals up to time  $t$  ( $\bar{A}^n(t)$ ) and then minus all of our departures up to time  $t$  ( $\bar{S}^n(t)$ ), giving us  $\bar{Q}^n(t) = \bar{Q}^n(0) + \bar{A}^n(t) - \bar{S}^n(t)$ . But we know from the *FSLLN* and *Lemma 1.1* that

$$\bar{A}^n(t) = \frac{A(t)}{\mathbf{n}} \rightarrow \lambda t, \quad \bar{S}^n(t) = \frac{S(t)}{\mathbf{n}} \rightarrow \mu t$$

and so

$$\begin{aligned} \bar{Q}^n(t) &= \bar{Q}^n(0) + \bar{A}^n(t) - \bar{S}^n(t) \\ &\rightarrow \bar{Q}(0) + \lambda t - \mu t \\ &= \bar{Q}(0) + (\lambda - \mu)t \end{aligned}$$

But this is still missing the  $(\dots)^+$ , so why is it there?

Well, if we are in the case of where  $\lambda < \mu$  then then quantity  $(\lambda - \mu)$  can be negative and while generally  $(\lambda - \mu)$  is OK to be negative (it's to be expected that we will sometimes be decreasing) what happens to our system when  $(\lambda - \mu)$  is negative and  $(\lambda - \mu) > \bar{Q}(0)$ ? This would mean that  $\bar{Q}^n(t) < 0$  (i.e. our system would have less than zero occupancy) which obviously cannot happen and so we define  $\bar{Q}^n(t)$  to only be the positive part of  $\bar{Q}(0) + (\lambda - \mu)t$  such that when  $\bar{Q}(0) + (\lambda - \mu)t < 0$  our system defaults to 0, finally yielding

$$\bar{Q}^n(t) \rightarrow (\bar{Q}(0) + (\lambda - \mu)t)^+$$

Next, consider applying our space/rate scaling to the way in which we have defined our system at the beginning of G/G/1 queueing to get

$$\bar{Q}^n(t) = \bar{X}^n(t) + \bar{Y}^n(t)$$

s.t.

$$\begin{aligned}\bar{X}^n(t) &= \bar{Q}^n(0) + [\bar{A}^n(t) - \lambda t] - [\bar{S}^n(\bar{B}^n(t)) - \mu \bar{B}^n(t)] + (\lambda - \mu)t \\ \bar{Y}^n(t) &= \mu[t - \bar{B}^n(t)]\end{aligned}$$

$$\text{and, } Q(t)dY(t) = 0 \tag{2.1.1}$$

The next theorem that we introduce will play a vital role in approximating our queue and effectively modelling  $S(B(t))$ .

**Theorem 2.1.** (*One-Dimension Skorokhod Mapping*).

Suppose  $x \in D$  such that  $D$  is a right continuous function. Then there exists a unique pair of  $y \in D$  and  $z \in D$  such that the following hold for all  $t$ .

1.  $z(t) = x(t) + y(t) \geq 0$ ;
2.  $y(t)$  is non-decreasing in  $t$  with  $y(0) = 0$ ;
3.  $\int_0^\infty z(t)dy(t) = 0$ ;

In fact, the unique pair can be re-written as

$$\begin{aligned}y(t) &= \sup_{0 \leq s \leq t} [-x(s)]^+ \\ z(t) &= x(t) + \sup_{0 \leq s \leq t} [-x(s)]^+\end{aligned}$$

**Proof**

It is clear from how  $z$  and  $y$  are defined in the last two lines that  $z$  is non-negative and that  $y$  is strictly increasing and starting at 0, thus 1. and 2. are satisfied. Note that for any  $t > 0$  we have that  $dy(t) > 0$

$$\begin{aligned}\Rightarrow \sup_{0 \leq s \leq t} [-x(s)]^+ &\text{ is attained at } s = t \\ \Rightarrow z(t) &= 0.\end{aligned}$$

and thus 3. is satisfied as well.

To show the uniqueness, consider  $y'$  and  $z'$  to be another pair satisfying the theorem and take  $z - z' = y - y'$  to be the difference between two non-

decreasing functions and that  $z(0) - z'(0) = 0$ . So,

$$\begin{aligned} \frac{1}{2}(z(t) - z'(t))^2 &= \int_0^t (z(u) - z'(u)) d(z(u) - z'(u)) \\ &= \int_0^t (z(u) - z'(u)) d(y(u) - y'(u)) \\ &= - \int_0^t (z'(u) dy(u) + z(u) dy'(u)) \\ &\leq 0, \end{aligned}$$

$$\begin{aligned} \text{since } z dy &= 0 \text{ and } z' dy' = 0, \text{ and } z, z', dy, dy' \geq 0 \\ &\Rightarrow z \equiv z' \text{ and } y \equiv y' \end{aligned}$$

Hence, the uniqueness of  $z$  and  $y$ .

**Definition 2.3** (Reflected Process). .

In the theorem above we call  $z$  the **reflected process** of  $x$  and denote it by

$$z = \Phi(x)$$

**Definition 2.4** (Regulator). .

In the theorem above if we let  $x$  be a Brownian motion, then we call  $y$  the **regulator** of  $x$  and denote it by

$$y = \Psi(x)$$

Intuitively,  $y$  is meant to be able to provide enough of an add on to ensure that the modified process  $z = x + y$  remains  $\geq 0$ , given that the process  $x$  isn't guaranteed to be non-negative.

Now, thinking back to how we defined our queue and it's properties from line **(2.1.1)**, they in fact satisfy all three conditions required for Skorokhod mapping and so we can apply it to get,

$$\begin{aligned} \bar{X}^n(t) &\rightarrow x(t) \\ \Rightarrow \bar{Y}^n(t) &= \Psi(\bar{X}^n(t)) \rightarrow \Psi(x(t)) = \sup_{0 \leq s \leq t} [-x(s)]^+ \\ \Rightarrow \bar{Q}^n(t) &= \Phi(\bar{X}^n(t)) \rightarrow \Phi(x(t)) = x(t) + \sup_{0 \leq s \leq t} [-x(s)]^+ \end{aligned}$$

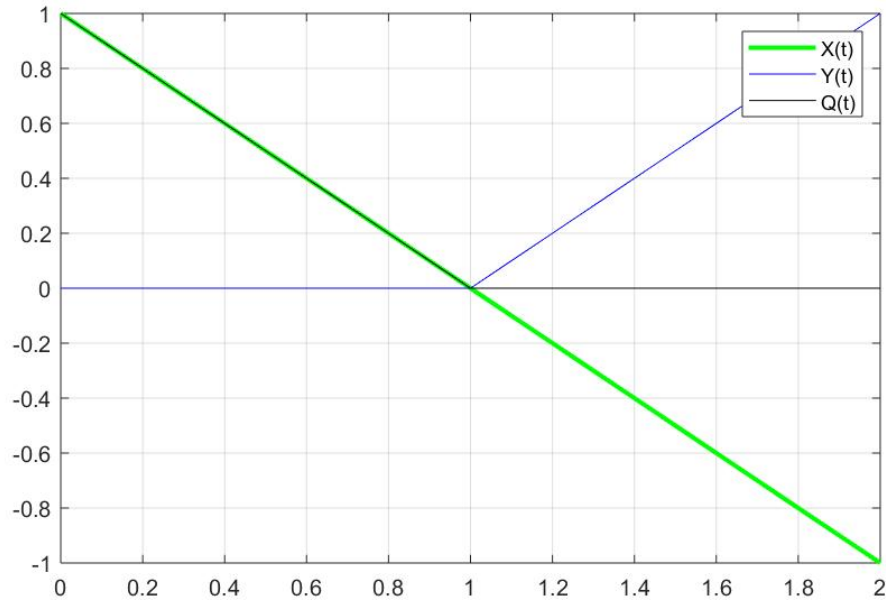
And so

$$\bar{Y}^n(t) = \Psi(\bar{X}^n(t)) \rightarrow \Psi(\bar{X}(t))$$

is our associated regulator, and

$$\bar{Q}^n(t) = \Phi(\bar{X}^n(t)) \rightarrow \Phi(\bar{X}(t)) = (\bar{Q}(0) + (\lambda - \mu)t)^+$$

is our reflected affine function.

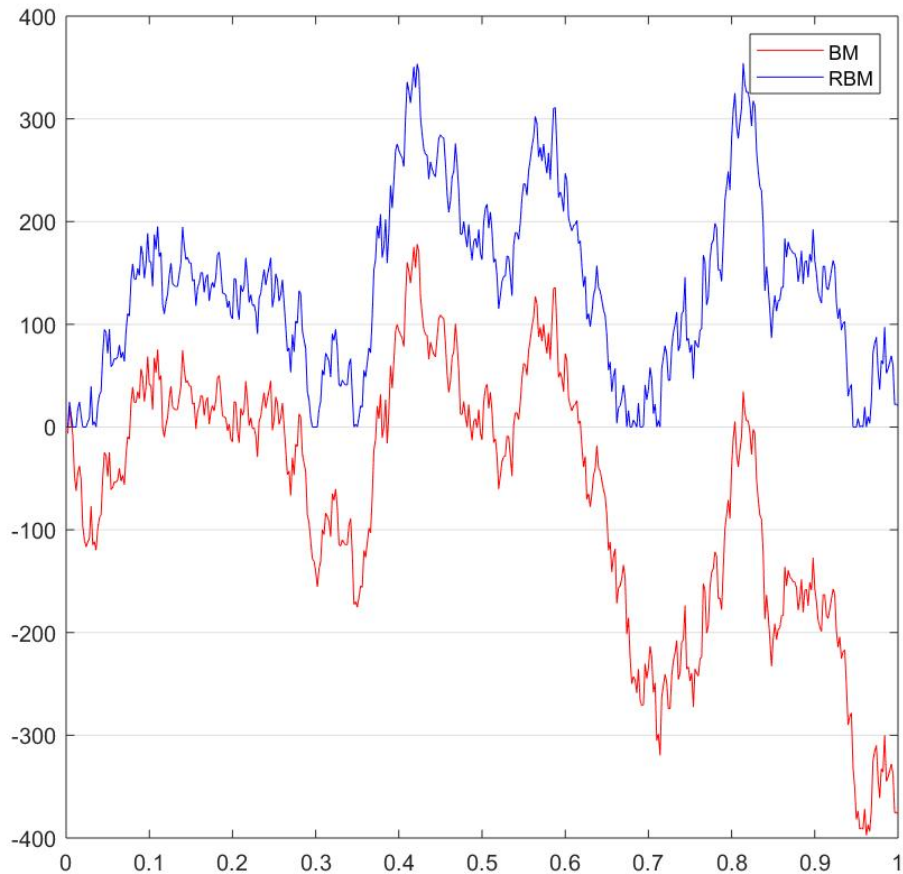


### 2.1.2 G/G/1 - Diffusion Approximation

**Definition 2.5** (Reflected Brownian Motion). (RBM).

In the one-dimensional Skorokhod mapping theorem, if we let the input process  $(X)$  be a Brownian motion  $X(t)$ ,  $X(t) > 0$ . Then we call  $Z = \Phi(X)$  the **one-dimensional reflected Brownian motion**. When  $X$  has drift  $\theta$  and variance  $\sigma^2$  we write  $Z = RMB(\theta, \sigma^2)$ .

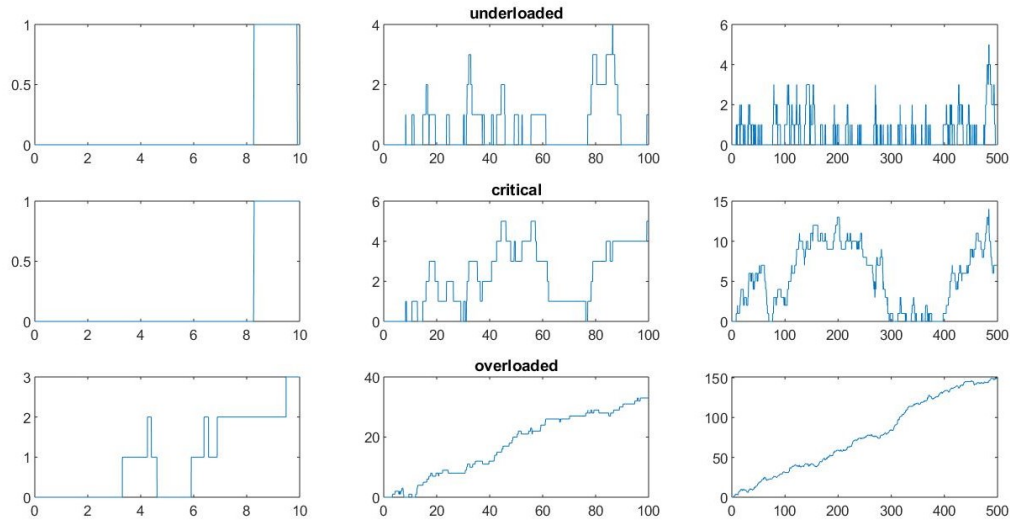




**Theorem 2.2.** .

*A RBM  $Z(t) = RBM(\theta, \sigma^2)$ , converges in distribution to some limiting variable  $Z \iff \theta < 0$ . In this case  $Z$  is exponentially distributed with parameter  $\frac{-2\theta}{\sigma^2}$ .*

Recall the simple diagram of the tub from the beginning of section 2.1.1 - Fluid Limits that illustrates a tub filled with liquid that arrives with rate  $\lambda$  and departs with rate  $\mu$  with three possibilities; under loaded, critical, or overloaded. Next, we will demonstrate what our system looks like under each of those cases and what we want to understand is the fluctuations around those linear (affine) fluid approximations.



underloaded:  $\lambda = 3, \mu = 5$   
critical:  $\lambda = 5, \mu = 5$   
overloaded:  $\lambda = 5.30, \mu = 5$

We can see that when  $T$  is 'small' that in all three cases there does not seem to be much going on or of interest.

As  $T$  increases, in the case of an under loaded or critical system there seems to be a bit more of interest happening, but the overloaded system has already began exploding and there does not seem to much to analyse.

Then, by increasing  $T$  even more we can truly start to get an idea for the long term behaviour of each system. Our under loaded system is heavily centred about 0 (a lot of idle time); in our critical system we start to see something a bit more interesting (kind of looks like Brownian Motion!) that varies from bursts of busy to a bit of idle time; and as suspected, our overloaded system explodes even more following what seems to be a linear trend upwards.

(Note the range of our data! in the underloaded system we only see a max of 5 people in the system, in critical we reach 15 and in overloaded we explode to 150 (and still increasing) Now that we have a bit better of an understanding of the long term behaviour that some of these systems can have, it seems like an under loaded system could be useful in attempting to model some kind of rare event scenario, critical systems seem to be the most generally applicable, and overloaded systems seeming useless as they strictly explode to  $(\lambda - \mu)t$  (in our case we can see it converge to  $(5.3 - 5)(500) = 150$ ).

Now, let us focus our attention to the case where  $\lambda \approx \mu$  (critical system) and to do this we will consider the system

$$\begin{aligned}\hat{Q}^n(t) &= \frac{1}{\sqrt{\mathbf{n}}}Q(\mathbf{nt}) \\ \hat{X}^n(t) &= \frac{1}{\sqrt{\mathbf{n}}}X(\mathbf{nt}) \\ \hat{Y}^n(t) &= \frac{1}{\sqrt{\mathbf{n}}}Y(\mathbf{nt})\end{aligned}\tag{2.1.2}$$

under the assumption that

$$\sqrt{\mathbf{n}}(\lambda_n - \mu_n) = \theta_n \rightarrow \theta \in \Re.$$

with,

$$\hat{Q}^n(t) = \hat{X}^n(t) + \hat{Y}^n(t),\tag{as in 2.1.1}$$

and,

$$\begin{aligned}\hat{X}^n(t) &= \hat{Q}^n(0) + \frac{A(nt) - n\lambda t}{\sqrt{\mathbf{n}}} - \frac{S(B(nt)) - \mu B(nt)}{\sqrt{\mathbf{n}}} + \sqrt{\mathbf{n}}(\lambda - \mu)t \\ \hat{Y}^n(t) &= \mu \frac{nt - B(nt)}{\sqrt{\mathbf{n}}}\end{aligned}\tag{from Lemma 1.1}$$

where the term  $\sqrt{\mathbf{n}}(\lambda - \mu)t = \theta$  is  $o(1)$  (order 1) since we are in the case  $\lambda \approx \mu$  and

$$\begin{aligned}\sqrt{\mathbf{n}}(\lambda - \mu)t &= \theta \\ \Rightarrow \lambda &= \mu + \frac{\theta}{\sqrt{\mathbf{n}}}\end{aligned}$$

$$\begin{aligned}
\text{i.e. } \frac{\lambda_n}{\mu_n} &= \frac{\mu + \frac{\theta}{\sqrt{n}}}{\mu} \\
&= 1 + \frac{\theta}{\mu\sqrt{n}} \\
\therefore \frac{\lambda_n}{\mu_n} &\rightarrow 1 & \therefore \left(\frac{\theta}{\mu}\right) \left(\frac{1}{\sqrt{n}}\right) &\rightarrow 0 \text{ as } n \rightarrow \infty \\
\text{and } \theta_n &= \sqrt{n}[\lambda_n - \mu_n] \\
&= \sqrt{n}[\mu_n + \frac{\theta}{\sqrt{n}} - \mu_n] \\
&= \sqrt{n}[\frac{\theta}{\sqrt{n}}] \\
\therefore \theta_n &\rightarrow \theta
\end{aligned}$$

Once again, all three conditions of Skorokhod Mapping are satisfied and so we can apply it to get the reflection mapping

$$\hat{Q}^n(t) = \Phi(\hat{X}^n(t)) \quad | \quad \hat{Y}^n(t) = \Psi(\hat{X}^n(t))$$

Which can be simplified to

$$\begin{aligned}
\hat{X}^n(t) &= \hat{Q}^n(0) + \hat{A}^n(t) - \hat{S}^n(\bar{B}^n(t)) + \theta_n t \\
\hat{Y}^n(t) &= \sqrt{n}\mu[t - \bar{B}^n(t)]
\end{aligned}$$

where by the FCLT

$$\begin{aligned}
\hat{A}^n(t) &\rightarrow \hat{A} \\
\hat{B}^n(t) &\rightarrow t \\
\text{and as a result, } \hat{S}^n(t) &\rightarrow \hat{S}
\end{aligned}$$

Now because our system is critical, it is consistently busy and

$$\begin{aligned}
\hat{X}^n(t) &= \hat{Q}^n(0) + \hat{A}^n(t) - \hat{S}^n(\bar{B}^n(t)) + \theta_n t \\
\Rightarrow \hat{X}^n(t) &\rightarrow \hat{Q}(0) + \hat{A}(t) - \hat{S}(t) + \theta t
\end{aligned}$$

Thus,  $\hat{X}^n(t)$  is the difference of two independent Brownian motion processes (arrival and service) and in such is a Brownian motion itself with drift

$\theta = 0$  (as shown above) and variance  $\lambda c_a^2 + \mu c_s^2 = \lambda(c_a^2 + c_s^2)$ ; satisfying required conditions for Skorokhod mapping!  
Hence,

$$\begin{aligned}\hat{X}^n(t) &\rightarrow \tilde{W}(t); \\ \hat{Y}^n(t) &= \Psi(\hat{X}^n(t)) \rightarrow \Psi(\tilde{W}(t)) \\ \hat{Q}^n(t) &= \Phi(\hat{X}^n(t)) \rightarrow \Phi(\tilde{W}(t))\end{aligned}$$

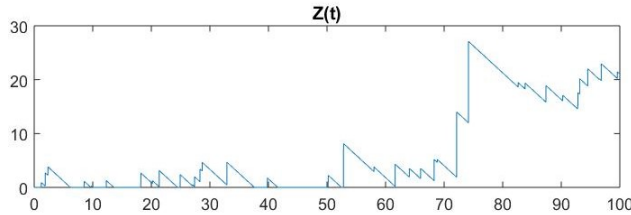
where  $\tilde{W}(t)$  is a Brownian motion, and  $\hat{Y}^n(t)$  and  $\hat{Q}^n(t)$  are the associated regulator and reflection (respectively) which can now be leveraged and utilized using what we have learned about Skorokhod mapping!

To do so, we will take a step back and begin to look at our system from a **Workload** perspective, that is instead of viewing our system as the amount of people in the system, we will view it as the amount of work in the system (that is brought by all those in queue). In order to be able to model our queue using a series of primitive equations.

### 2.1.3 Diffusion (Workload)

Define a process  $Z(t) = \sum_{i=1}^{A(t)} v_i - B(t)$  that we can rewrite as

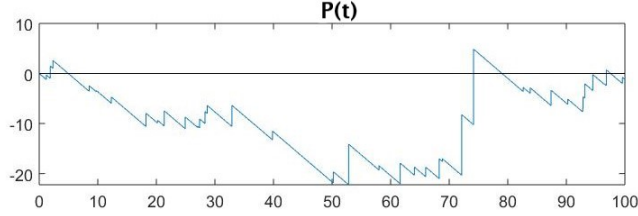
$$\begin{aligned}Z(t) &= \sum_{i=1}^{A(t)} v_i - B(t) \\ &= \sum_{i=1}^{A(t)} v_i - t + t - B(t) \\ &= \left( \sum_{i=1}^{A(t)} v_i - t \right) + \left( t - B(t) \right) \\ &= P(t) + Y(t)\end{aligned}\tag{2.1.3}$$



where,

$$P(t) = \sum_{i=1}^{A(t)} v_i - t,$$

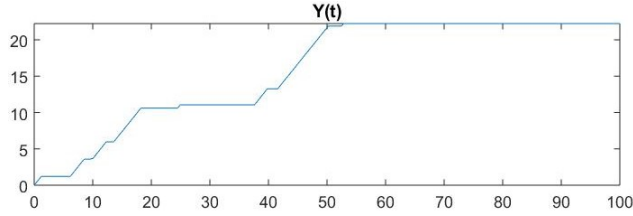
is the **potential outflow**,



and,

$$Y(t) = t - B(t)$$

is the **cumulative idle time**.



Note that from how these are defined, they satisfy all conditions for Skorokhod mapping where  $P(t)$  is a right continuous function, thus we can simplify  $Y(t)$  to

$$Y(t) = \sup_{0 \leq s \leq t} [-P(s)]^+$$

as  $Y(t)$  is the **regulator (definition 2.4)** of  $P(t)$  and  $Z(t)$  is the **reflected process (definition 2.3)** of  $P(t)$

$$\begin{aligned} \Rightarrow Z(t) &= P(t) + Y(t) \\ &= \sum_{i=1}^{A(t)} v_i - t + \sup_{0 \leq s \leq t} [-P(s)]^+ \end{aligned}$$

Which are precisely the types of relationships we wish to further examine and utilize in our fluid and diffusion approximations.

Take  $Q(t)$  as stated at the beginning of section 2.1 to be the total number of jobs in the system at time  $t$ ,  $A(t)$  to be the number of arrivals by time  $t$  and lastly  $v_i$  to be the service time required for the  $i$ 'th job. Now in the context of a single server system and using the **workload**, **potential outflow** and **cumulative idle time** as described in our motivation for workload approximations, we have seen that  $Z = \Phi(P)$  and  $Y = \Psi(P)$  are indeed reflections and regulators (respectively) of  $X$  from the uniqueness of Skorokhod mapping.

From this, we now we focus our attention onto

$$Z(t) = V(Q(0) + A(t)) - B(t) \quad (\text{i.e. the workload})$$

(still in the case of  $\lambda \approx \mu$ )

Where,

$A(t)$  is our arrival process,

$B(t)$  is our busy time process,

$Z(t)$  is the total amount of work remaining in the system at time  $t$ ,

and  $V(k)$  is the cumulative sum of the first  $k$  job requirements.

Which makes sense as our definition of workload says that the total amount of work left over is the sum of the job requirements that started in the system plus all the jobs that arrived minus the total amount of time the system was busy (working on job requirements). And using our knowledge of the Skorokhod mapping, FCLT, FSLLN, and space/rate scaling we can see, In order to better understand the behaviour of  $Z(t)$  in the context of fluid approximations, we first need to better understand the behaviour of  $P(t)$ .

Denote  $\sum_{j=1}^{Q(0)} v_j$  to be the amount of work brought by our initial condition (i.e.

amount of work that the system has from the initial  $Q(0)$  people)

$$\begin{aligned}
P(t) &= \sum_{j=1}^{Q(0)} v_j + \sum_{i=1}^{A(t)} v_i - t \\
\Rightarrow \bar{P}^n(t) &= \frac{P(nt)}{n} \\
&= \frac{Q(0)}{n} \frac{1}{Q(0)} \sum_{j=1}^{Q(0)} v_j + \frac{A(nt)}{n} \frac{1}{A(nt)} \sum_{i=1}^{A(nt)} v_i - t
\end{aligned}$$

where,

$$\begin{aligned}
\frac{1}{Q(0)} \sum_{j=1}^{Q(0)} v_j &\rightarrow E(v_j) = \frac{1}{\mu} = m \\
\frac{1}{A(nt)} \sum_{i=1}^{A(nt)} v_i &\rightarrow E(v_i) = \frac{1}{\mu} = m \\
\Rightarrow \bar{P}^n(t) &\rightarrow \bar{Q}(0)m + m\lambda t - t \\
&= m(\bar{Q}(0) + \lambda t - \frac{1}{m}t) \\
&= m(\bar{Q}(0) + (\lambda - \mu)t)
\end{aligned}$$

thus,

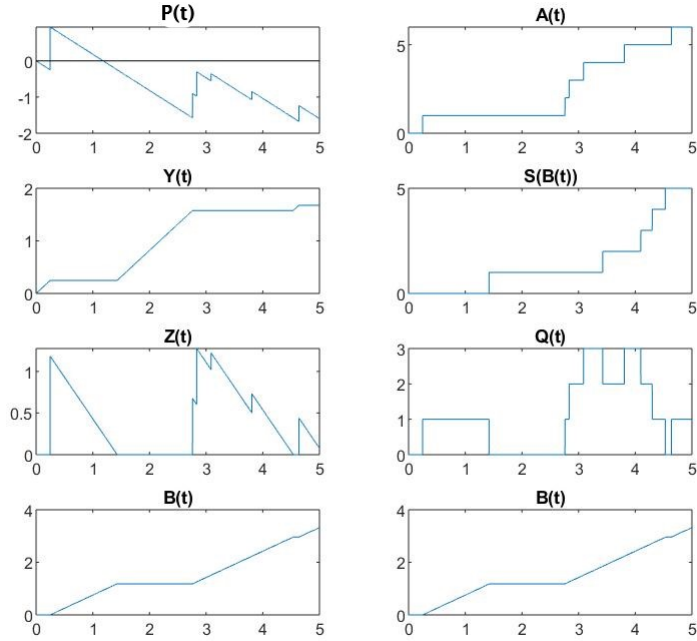
$$\begin{aligned}
\bar{P}^n(t) &\rightarrow (\bar{Q}(0) + (\lambda - \mu)t)^+ = \bar{P}(t) \\
\Rightarrow \bar{Z}^n(t) = \Phi(\bar{P}^n(t)) &\rightarrow \Phi(\bar{P}(t)) = m\bar{P}(t)
\end{aligned}$$

Where it can be shown similarly for diffusion that,

$$\Rightarrow \hat{Z}^n(t) = \Phi(\hat{P}^n(t)) \rightarrow \Phi(\hat{P}(t)) = m\hat{P}(t)$$

using **(2.1.2)**





(see code

*'WorkloadDiffusion.m'* in appendix)

Where we are able to easily get  $P(t)$  using just our arrival and service times. From here we were able to easily model  $Y(t)$  from  $P(t)$  using Skorokhod mapping (i.e.  $Y(t) = \sup_{0 \leq s \leq t} [-P(s)]^+$ ).

Now  $Z(t) = P(t) + Y(t)$  follows swiftly as it is just the sum of  $P(t)$  and  $Y(t)$  (which we already have).

And now as we have  $P(t), Y(t)$  and  $Z(t)$  we can leverage this to get  $B(t)$  since

$$\begin{aligned}
 Y(t) &= t - B(t) && \text{as seen in (2.1.3)} \\
 \Rightarrow B(t) &= t - Y(t)
 \end{aligned}$$

which makes evaluating  $S(B(t))$  simple.

And lastly, putting it all together we are now able to get  $Q(t)$  as  $Q(t) = A(t) - S(B(t))!!$

Thus taking a series of simple, intuitive steps we were able to utilize many different aspects of our system in order to effectively model and approximate our queue while avoiding some of the elements that can be a bit more difficult

to deal with!

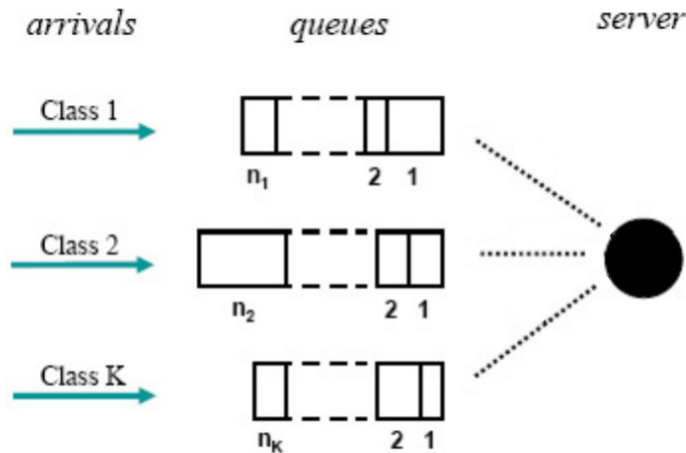
To help motivate the following section on multi-class networks, recall that so far we have been working in the case of  $G/G/1$  which represents a system with **G**enerally distributed arrivals, **G**enerally distributed departures(or services), and **1** server. Since both our arrivals and departures are generally distributed let's consider a case where our arrivals come from distribution  $A(x)$  s.t.

$$A(x) = \sum_{i=1}^K p_i F_x(x)^{(i)}; \quad \sum_{i=1}^K p_i = 1 \text{ and } F_x(x)^{(i)} \text{ are } i \text{ proper cdfs}$$

that is,  $A(x)$  is a composition of distributions with associated probabilities  $p_i$  of coming from each. WLOG assume that our departures also come from a distribution that is a composition of other distributions each with their own associated weights (arrivals and departures both being compositions of the same number of distributions)

For what may be starting to seem like a complex model of a rare case from our  $G/G/1$  system, this scenario is actually very useful and much more common than one may think as it represents a **single server, multi-class model**.

#### 2.1.4 Multi-class Model



For a multi-class model as above where there are  $K$  classes each with respective queues ( $Q_i$ ) and arrival/service rates  $\lambda_i$ , and  $\mu_i$   $i = 1, \dots, K$ . In

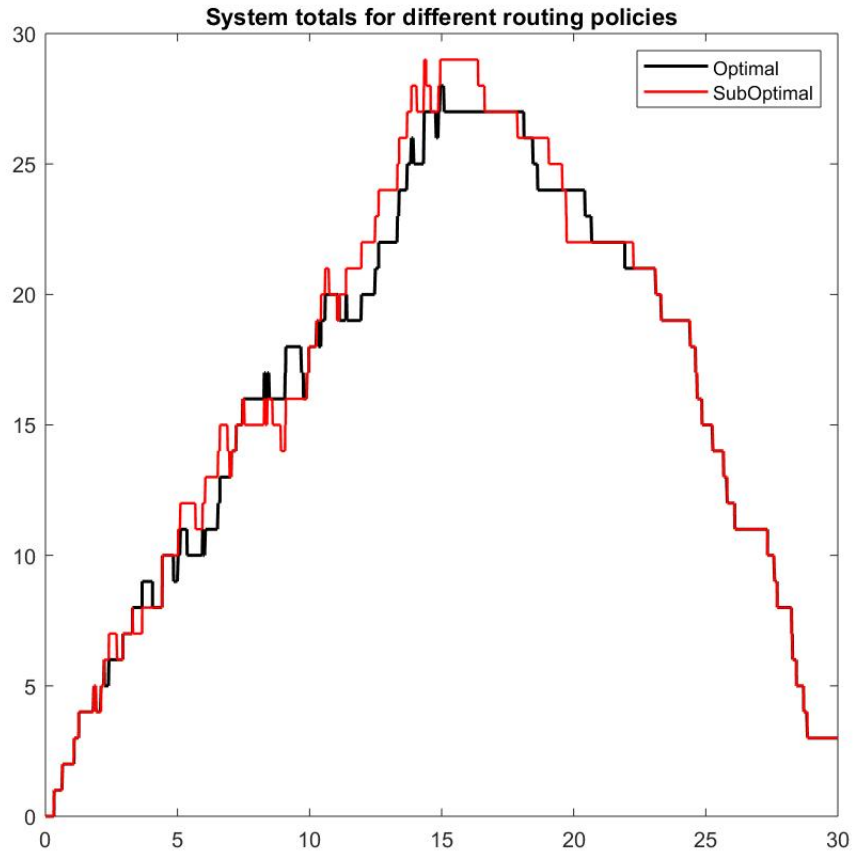
the diagram above, each  $n_i$  will arrive based on their associated rates ( $\lambda_i = E[F_x(x)^{(i)}]$ ) with probability  $p_i$  of occurring. In the diagram it may not be clear what our  $p_i$ 's are however they are nothing more than  $p_i = P(n_i < n_j) \quad \forall j \neq i$  (i.e. the probability that  $n_i$  occurred before any other arrival). It is worth noting that using the same intuition that lead us to discover the idea of a multi-class model we can work backwards to show that a multi-class model can be represented as a  $G/G/1$  system where we already know a lot about them and how to deal with them! In such, define the workload

$$Z(t) = \sum_{i=1}^K V_i(A_i(t)) - t + Y(t)$$

Where  $Y(t)$  is the cumulative idle time as before, then consider a state-space collapse.

Using the fact that,

$$\begin{aligned} \hat{Z}^n(t) &\rightarrow \tilde{Z}(t) \text{ - our R.B.M.} \\ \sum_{i=1}^K \left( \frac{1}{\mu_i} \hat{Q}_i^n \right) &\rightarrow \tilde{Z} \text{ - our state space collapse} \\ &\text{(i.e. condensing our system back to G/G/1)} \end{aligned}$$



Here we see that regardless of routing policy that the total system behaves very similarly, demonstrating the independence of routing policy from the **state space collapse**

It should also be noted that when in a multi-class scenario that our system

will be critically loaded when  $\sum_{i=1}^K \frac{\lambda_i}{\mu_i} = 1$  because,

$$\begin{aligned}
 p_i &= \frac{\lambda_i}{\sum_{i=1}^K \lambda_i} \\
 \lambda &= \sum_{i=1}^K \lambda_i \\
 \mu &= \frac{1}{\sum_{i=1}^K \frac{p_i}{\mu_i}} \\
 &= \frac{1}{\sum_{i=1}^K \frac{\lambda_i}{\mu_i \sum_{i=1}^K \lambda_i}} \\
 &= \frac{\sum_{i=1}^K \lambda_i}{\sum_{i=1}^K \frac{\lambda_i}{\mu_i}} \\
 \Rightarrow \frac{\lambda}{\mu} &= \frac{\sum_{i=1}^K \lambda_i}{\frac{\sum_{i=1}^K \lambda_i}{\sum_{i=1}^K \frac{\lambda_i}{\mu_i}}} \\
 &= \sum_{i=1}^K \frac{\lambda_i}{\mu_i}
 \end{aligned}$$

Hence,

$$\frac{\lambda}{\mu} = 1 \Rightarrow \sum_{i=1}^K \frac{\lambda_i}{\mu_i} = 1$$

Multi-class models are particularly useful as they allow us to define our system in a more detailed way allowing us to examine them further and help optimize their performance based on a variety of different conditions. One example could be if we had 2 classes, one of VIPs and one of regular customers and we want to try and optimize our system such that VIPs spend

as little time in the system as possible.

If we consider our system to be  $M/G/1$  (that is, exponential services with general arrivals) then one way we can do this is by prioritizing the classes to make asymptotic control easy for any work-conserving control discipline using the  $C\mu$ -rule.

To help motivate and set up for the  $C\mu$ -rule we shall first begin to examine some of the underlying foundations of single server queues working with priorities.

To do so, define

$$\rho_i = \lambda_i E[X_i] \quad \because \text{Little's Law}$$

$$L_k \equiv \text{number of customers in queue from class } k$$

$$W_k \equiv \text{waiting time for class } k$$

$$R_k \equiv \text{residual service time for class } k$$

And

$$\begin{aligned} E[R] &= \sum_{k=1}^K \rho_k E[R_k] \\ &= \frac{1}{2} \sum_{k=1}^K \frac{\rho_k E[X_k^2]}{E[X_k]} \\ &= \frac{1}{2} \sum_{k=1}^K \lambda_k E[X_k^2] \end{aligned}$$

then also from Little's Law we have that  $E[L_i] = \lambda_i E[W_i]$  so,

$$\begin{aligned}
E[W_1] &= E[L_1]E[X_1] + E[R] \\
&= \lambda_1 E[W_1]E[X_1] + E[R] \\
&= \rho_1 E[W_1] + E[R] \\
&= \frac{E[R]}{1 - \rho_1} \\
E[W_2] &= E[L_1]E[X_1] + E[L_2]E[X_2] + \lambda_1 E[W_2]E[X_1] + E[R] \\
&= \rho_1 E[W_1] + \rho_2 E[W_2] + \rho_1 E[W_2] + E[R] \\
&= \frac{\rho_1 \frac{E[R]}{1 - \rho_1} + E[R]}{1 - \rho_1 - \rho_2} \\
&= \frac{E[R]}{(1 - \rho_1)(1 - \rho_1 - \rho_2)}
\end{aligned}$$

and similarly,

$$E[W_3] = \frac{E[R]}{(1 - \rho_1 - \rho_2)(1 - \rho_1 - \rho_2 - \rho_3)}$$

Using this recursive relation we can see that

$$E[W_k] = \frac{E[R]}{(1 - \rho_1 - \dots - \rho_{k-1})(1 - \rho_1 - \dots - \rho_k)}$$

And now, assuming that class  $i$  has an associated cost  $C_i$  of being in the system then the total cost becomes

$$\begin{aligned}
\sum_{i=1}^K C_i E[N_i] &= \sum_{i=1}^K C_i \lambda_i E[W_i] \\
&= \sum_{i=1}^K \frac{C_i}{E[X_i]} \rho_i E[W_i] \\
&= \sum_{i=1}^K (C_i \mu_i) \rho_i E[W_i], \quad \mu_i = \frac{1}{E[X_i]}
\end{aligned}$$

**Lemma 2.1** ( $C\mu$  - Rule).

The  $C\mu$ -rule states to assign priorities based on  $C_1 \mu_1 \geq C_2 \mu_2 \geq \dots \geq C_n \mu_n$  where  $C_i$  is the associated costs to class  $i$ , and  $\mu_i = \frac{1}{E[X_i]}$ .

**Why?**

Well,

$$\begin{aligned}
\sum_{i=1}^K \rho_i E[W_i] &= \sum_{i=1}^K \rho_i \frac{E[R]}{(1 - \rho_1 - \dots - \rho_{k-1})(1 - \rho_1 - \dots - \rho_k)} \\
&= E[R] \sum_{i=1}^K \left( \frac{1}{(1 - \rho_1 - \dots - \rho_k)} - \frac{1}{(1 - \rho_1 - \dots - \rho_{k-1})} \right) \\
&= E[R] \left( \frac{1}{(1 - \rho_1 - \dots - \rho_k)} - 1 \right) \\
&= \frac{\rho}{1 - \rho} E[R]
\end{aligned}$$

Which is constant, and so minimizing total cost  $\sum_{i=1}^K (C_i \mu_i) \rho_i E[W_i]$  subject to  $\sum_{i=1}^K \rho_i E[W_i] \equiv \text{constant}$  is solved by assigning priorities based on  $C_1 \mu_1 \geq C_2 \mu_2 \geq \dots \geq C_k \mu_k$

**Example 2.1.** For purposes of this example suppose we have a G/G/1 queueing system with 3 independent classes, class 1 having inter-arrival times distributed as  $exp(3)$  random variables (r.v.'s) with  $exp(1)$  service requirements. Suppose further that class 2 has arrival times distributed as  $Beta(6, 2)$  r.v.'s with  $exp(0.5)$  and queue 3 has inter-arrival times distributed as  $Gamma(2, 0.5)$  with  $exp(1)$  service requirements. Denote  $E[u_i] = \lambda_i$  and  $E[v_i] = \mu_i$  such that

$$\begin{aligned}
\lambda_1 &= 3, & \mu_1 &= 1 \\
\lambda_2 &= 0.75, & \mu_2 &= 0.5 \\
\lambda_3 &= 1, & \mu_3 &= 1
\end{aligned}$$

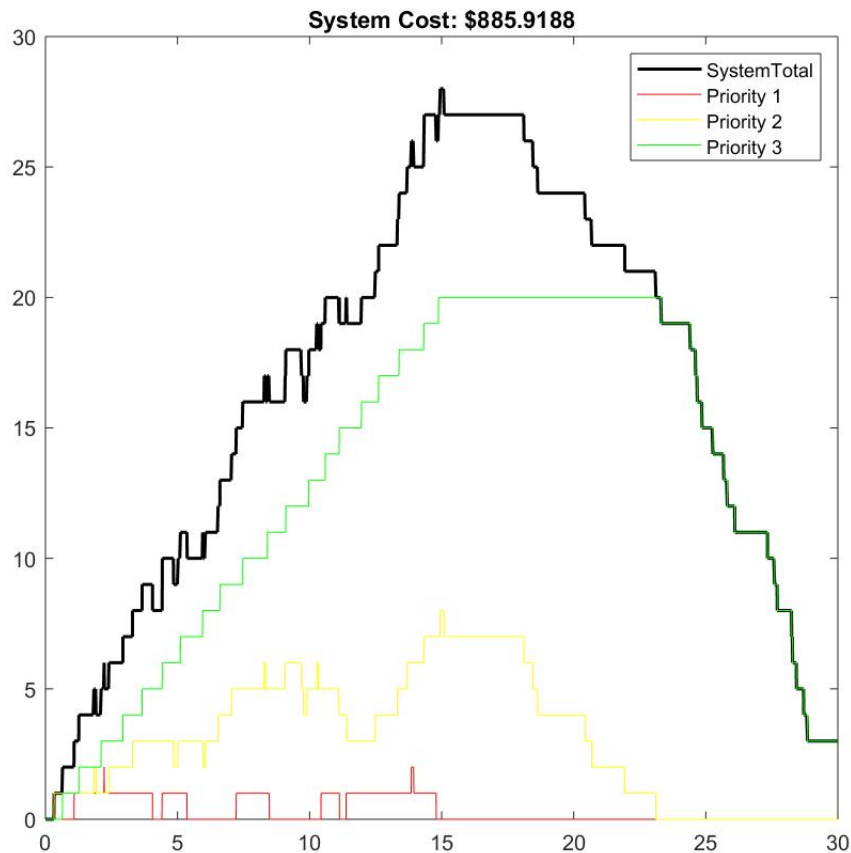
and note that our system will be overloaded loaded, and so for the first 15 units of time we accept new customers into our queues and then after that we close our doors and complete all remaining work in the system. Suppose also that customers from classes 1, 2, 3 have associated costs  $C = \{3, 1, 2\}$  of being in the system and our goal is to minimize the total cost of our system.



Thus, based on how we defined our costs and how we want to optimize our system (minimize total cost) we are able to assign priorities based on

$$C_1\mu_1 > C_3\mu_3 > C_2\mu_2$$

and so whenever there is someone from class 1 is in the queue we will serve them above any others to get them out as soon as possible (working non-preemptively, meaning if serving someone from a lower priority we will finish serving them and then take on those from other classes) .

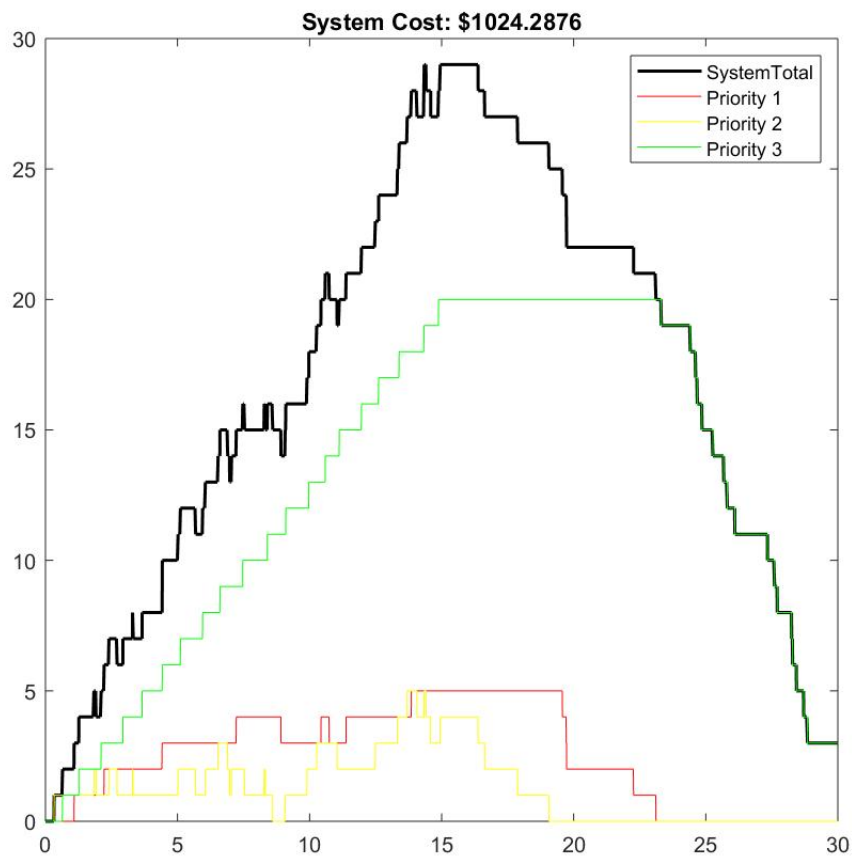


(see code '*MulticlassGG1Queue.m*' in appendix)

In the above graph the red line represents the amount of people from class 1, the yellow line represents those from class 3 and the green line class 2.

Intuitively the goal is to minimize the area under the graph where the red area is multiplied by a factor of 3, yellow by a factor of 2 and green a factor of 1 (refer back to stated costs) which in this cases produces a total system cost of \$885.92.

The code was run using seed (123) for reproducibility of similar outcomes if using sub-optimal prioritizing.



(slightly modified code from before using 3,1,2 as priority)

As we can see, using the same seed (same inter-arrival times and service requirements) it produces a drastically different outcome really outlining the effects that sub optimal prioritizing can have on a system. In this case

our total system cost is \$1024.29 which is considerably higher than optimal routing!

### **3 Appendix**

```
function [X] = BrownianMotion(X0,theta,sigma,N,T)
%This function simulates a brownian motion path taking values;
% X0, the starting point of the path
% Theta, the drift
% Sigma, the standard deviation
% N, the total number of steps
% T, the total space (range)

% the length of each interval
dt=N/T;
%creating a vector from 1->T with equal spacing
t=[0:dt:T];
%W~N(0,dt), thus W=sqrt(dt)*Z where Z~N(0,1)
W=cumsum(sqrt(dt)*normrnd(0,1,1,N));
X=X0+theta*t+sigma*W;
end
```

```
function [] = PlotBrownianMotion(numpaths,X0,theta,sigma,N,T)
% this functions plots npaths # of independent brownian motion simulations
% for BMs with inputted variables

%pre-allocating storage
x=zeros(N,numpaths);
for i = 1:numpaths
    x(:,i)=BrownianMotion(X0,theta,sigma,N,T);
end
for i = 1:numpaths
    plot(x(:,i))
    hold on
end
hold off
```

```
function [] = RandomWalk(X0,n,p)
% plots random walk with multiples of n steps, prob p of increasing and
% starting at X0

% intialize path X
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,1)
xx=0:1:n;
plot(xx,X, '.')
title(['Random Walk with n = ', num2str(n), ' and p = ', num2str(p)])

% intialize path X and increase n
n=1e01*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,2)
xx=0:1:n;
plot(xx,X, '.')
title(['Random Walk with n = ', num2str(n), ' and p = ', num2str(p)])

% intialize path X and increase n
n=1e01*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
```

```
    end
end
subplot(2,2,3)
xx=0:1:n;
plot(xx,X, '.')
title(['Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])
% initialize path X and increase n more
n=1e02*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
end
subplot(2,2,4)
xx=0:1:n;
plot(xx,X, '.')
title(['Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

end
```

```
function [] = ScaledRandomWalk(X0,n,p)
% plots random walk with multiples of n steps, prob p of increasing and
% starting at X0

% intialize path X
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,1)
t=0:1:n;
Xbar=(1/n)*X;
t=(1/n)*t;
plot(t,Xbar, '.')
title(['Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

% intialize path X and increase n
n=1e01*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,2)
t=0:1:n;
Xbar=(1/n)*X;
t=(1/n)*t;
plot(t,Xbar, '.')
title(['Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

% intialize path X and increase n
n=1e01*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
```



```
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,3)
t=0:1:n;
Xbar=(1/n)*X;
t=(1/n)*t;
plot(t,Xbar, '.')
title(['Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

% initialize path X and increase n more
n=1e02*n;
X=zeros(1,n);
% set X(0) to X0
X=[X0 X];
for i = 2:n+1
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
end
subplot(2,2,4)
t=0:1:n;
Xbar=(1/n)*X;
t=(1/n)*t;
plot(t,Xbar, '.')
title(['Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

end
```

```
function [] = CenteredScaledRandomWalk(X0,n,p)
% plots random walk with multiples of n steps, prob p of increasing and
% starting at X0

% initialize path X of length n-1
X=zeros(1,n-1);
S=zeros(1,n);
% set X(0) to X0 to make X of length n
X=[X0 X];
for i = 2:n
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
    S(i)=(p*i);
end
subplot(2,2,1)
t=0:1:n-1;
t=(1/n)*t;
Xhat = (1/sqrt(n))*(X-S);
plot(t,Xhat, '.')
title(['Centered/Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

% initialize path X and increase n
n=1e01*n;
X=zeros(1,n-1);
S=zeros(1,n);
% set X(0) to X0 to make X of length n
X=[X0 X];
for i = 2:n
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
    S(i)=(p*i);
end
subplot(2,2,2)
t=0:1:n-1;
t=(1/n)*t;
Xhat = (1/sqrt(n))*(X-S);
plot(t,Xhat, '.')
title(['Centered/Scaled Random Walk with n = ',num2str(n), ' and p = ',num2str(p)])

% initialize path X and increase n
n=1e02*n;
```

```
X=zeros(1,n-1);
S=zeros(1,n);
% set X(0) to X0 to make X of length n
X=[X0 X];
for i = 2:n
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
    S(i)=(p*i);
end
subplot(2,2,3)
t=0:1:n-1;
t=(1/n)*t;
Xhat = (1/sqrt(n))*(X-S);
plot(t,Xhat, '.')
title(['Centered/Scaled Random Walk with n = ',num2str(n),' and p = ',num2str(p)])
```

```
X=zeros(1,n-1);
S=zeros(1,n);
% set X(0) to X0 to make X of length n
X=[X0 X];
for i = 2:n
    u=rand(1);
    if u < p
        X(i)=X(i-1)+1;
    else
        X(i)=X(i-1);
    end
    S(i) = (p*i);
end
subplot(2,2,4)
t=0:1:n-1;
t=(1/n)*t;
Xhat = (1/sqrt(n))*(X-S);
plot(t,Xhat, '.')
title(['Centered/Scaled Random Walk with n = ',num2str(n),' and p = ',num2str(p)])

end
```

```
function [t Q Z] = WorkloadDiffusion(lambda,mu,T)
%This function simulates a Workload diffusion process for expo(lambda)
%arrivals and expo(mu) service rates and returns X,Y,Z for lambda approx mu
% Note that this code works for a G/G/1 System and so
% exprnd() may be replaced by other distributions so long as service rate
% is approx arrival rate.

%generate first arrival time to initialize u
u=exprnd(lambda);
%generate arrival times in [0,T]
while cumsum(u) < T
    u=[u,exprnd(lambda)];
end

%Generate the cumulative arrival times
U=cumsum(u);

% Note the last arrival occurs after time T and will be removed
U(end)=[];

% now to generate service requirements for each counted arrival
v=exprnd(mu,1,length(U));

%Create a copy of cumulative idle times and service rates for later use
U2=U;
V2=cumsum(v);
test=V2;
% We will first model our system in time [0,T]
% to make it as smooth as possible we will evaluate the system at intervals
% equal to a fraction of the minimum time between two consecutive arrivals

% extract the minimum inter-arrival time and make the spacing less
dt1=min(u)/100;
dt2=min(v)/100;
dt=min(dt1,dt2);

% evaluate on intervals less than minimum inter-arrival time
t=0:(dt):T;

% initialize:

% X (potential outflow)
X=zeros(1,length(t));

% Y (Cumulative idle time)
Y=zeros(1,length(t));

% A (Arrival Process)
A=zeros(1,length(t));
```

```

for i = 2:length(t)
    % if length(U) = 0 then there are going to be no more arrivals so the
    % system is just equal to
    if length(U) == 0 || t(i)<U(1)
        X(i) = X(i-1)-dt;
        A(i) = A(i-1);
    elseif U(1) <= t(i)
        % update potential outflow
        X(i) = X(i-1) + v(1) - dt;
        % increase total number of arrivals
        A(i) = A(i-1) + 1;
        % now that we have added our first service time at the time of
        % our first arrival they will no longer be needed for future
        % use and can be removed
        U(1)=[];
        v(1)=[];
    end
    % initialize Y(i) to be the previous value (no change)
    Y(i)=Y(i-1);

    % First check if X(i) is negative to consider increasing idle time
    % i.e. checks [x(s)]^+
    if X(i) < 0

        % if X(i) was negative check to see if the sum of X and Y is
        % negative, if yes then increase by dt
        % i.e. checks sup and increases idle time
        if X(i) + Y(i) < 0
            Y(i) = Y(i-1)+dt;
        end
    end
end

% we can now define the workload (incomplete work in system at time t)
Z = X + Y;

% and then use Z to find the busy time process B
B = (X + t) - Z;

% and now that we have our busy time process we can use it to generate S(B)
% SB = S(B(t)) (Service Process at time B(t))
SB=zeros(1,length(t));

for i = 2:length(B)
    % if length(U) = 0 then there are going to be no more arrivals so the
    % system is just equal to
    if length(V2) == 0
        SB(i) = SB(i-1);
    elseif round(V2(1),4) <= (round(B(i),4)+0.01)

```

```
        % increase total number of arrivals
        SB(i) = SB(i-1) + 1;
        % now that we have added our first service time at the time of
        % our first arrival they will no longer be needed for future
        % use and can be removed
        V2(1)=[];
    elseif B(i)<V2(1)
        SB(i) = SB(i-1);
    end
end
end

Q = A - SB;

subplot(4,2,1)
plot(t,X)
title('X(t)')
hold on
xx=zeros(1,length(t));
plot(t,xx,'Black')
hold off
subplot(4,2,3)
plot(t,Y)
title('Y(t)')
subplot(4,2,5)
plot(t,Z)
title('Z(t)')

subplot(4,2,7)
plot(t,B)
title('B(t)')
subplot(4,2,8)
plot(t,B)
title('B(t)')

subplot(4,2,2)
plot(t,A)
title('A(t)')

subplot(4,2,4)
plot(t,SB)
title('S(B(t))')

subplot(4,2,6)
plot(t,Q)
title('Q(t)')
end
```

```

function [TOTAL_COST,Q1,Q2,Q3] = MulticlassGG1Queue(T,C1,C2,C3)
% This function takes a time T as well as other associated costs and
% accepts arrivals up to time T at which it closes its doors and just
% finishes remaining work in the Queue
% Note that this was designed for a G/G/1 system and so any of the
% arrival or service rates may be replaced by other distributions,
% please define your queues such that the priority list is Q1>Q2>Q3.

% It returns the total cost associated to the system as well as the paths
% for the different ques

% generate first arrival times of each que to initialize u

%--can replace all arrival distributions with ones of users choice here---%
%-(make sure to also replace where filling other inter-arrival times below)

% Exp(2) => E[X] = 3
u1 = exprnd(3);
% Gamma(2,0.5) => E[X] = 2*.5 = 1
u2 = gamrnd(2,0.5);
% Beta(2,) => E[X] = 6/(6+2) = 0.75
u3 = betarnd(6,2);

%-----FILLING INTER ARRIVAL TIMES OF FIRST QUE-----%
% while cumsum of interarrivals of first que is less than T, expand
% total system size by 1 (new interarrival time) and creates zeros in
% all newly unfilled places
while cumsum(u1) < T
    u1 = [u1 exprnd(3)];
end
while cumsum(u2) < T
    u2 = [u2 gamrnd(2,0.5)];
end
while cumsum(u3) < T
    u3 = [u3, betarnd(6,2)];
end
u1(end)=[];
u2(end)=[];
u3(end)=[];
% Generate the cumulative arrival times
U1 = cumsum(u1);
U2 = cumsum(u2);
U3 = cumsum(u3);
% Creating an indicator of all arrivals
U = [U1 U2 U3];
% Sorting to have linear timeline
U = sort(U);

```

```
% now simulating service requirements for each arrival (note that
% exponential can be replaced for any other distribution)

%-----can also replace service rates for each queue here as well-----%

v1 = exprnd(1,1,length(u1));
v2 = exprnd(1,1,length(u2));
v3 = exprnd(0.5,1,length(u3));

% We will first model our system in time [0,T]
% to make it as smooth as possible we will evaluate the system at intervals
% equal to a fraction of the minimum time between two consecutive arrivals

% extract the minimum inter-arrival time and make the spacing less
dt1=min(diff(U)/100);
dt2=min([v1 v2 v3])/100;
dt=min(dt1,dt2);

% evaluate on intervals less than minimum inter-arrival time
t=0:(dt):(2*T);
Q1=zeros(1,length(t));
Q2=zeros(1,length(t));
Q3=zeros(1,length(t));

S=0;W=0;

for i = 2:length(t)
    % i.e. no more arrivals or no arrival occurred yet
    if length(U) == 0 || t(i) < U(1)
        Q1(i) = Q1(i-1);
        Q2(i) = Q2(i-1);
        Q3(i) = Q3(i-1);
    elseif U(1) < t(i)
        % i.e. is non zero and is the minimum of all other pending arrivals
        if length(U1) ~= 0 && ...
            ( length(U2) == 0 || U1(1) < U2(1) ) && ...
            ( length(U3) == 0 || U1(1) < U3(1) )
            % then increase que by 1 and remove the current arrival from
            % pending arrivals leaving both other ques as is
            Q1(i) = Q1(i-1) + 1;
            Q2(i) = Q2(i-1);
            Q3(i) = Q3(i-1);
            U1(1) = [];
            U(1) = [];
        elseif length(U2) ~= 0 && ...
```



```

        ( length(U1) == 0 || U2(1) < U1(1) ) && ...
        ( length(U3) == 0 || U2(1) < U3(1) )
        % then increase que by 1
        Q1(i) = Q1(i-1);
        Q2(i) = Q2(i-1) + 1;
        Q3(i) = Q3(i-1);
        U2(1) = [];
        U(1) = [];
    elseif length(U3) ~= 0 && ...
        ( length(U2) == 0 || U3(1) < U2(1) ) && ...
        ( length(U1) == 0 || U3(1) < U1(1) )
        % then increase que by 1
        Q1(i) = Q1(i-1);
        Q2(i) = Q2(i-1);
        Q3(i) = Q3(i-1) + 1;
        U3(1) = [];
        U(1) = [];
    end
end

% now to deal with potential departures

if S == 1 && W == 0
    % i.e. there was someone in service from Que 1 and they have no
    % work remaining
    S = 0; % take them out of service
    Q1(i) = Q1(i) - 1; % remove one person from class 1 from system
elseif S == 2 && W == 0
    S = 0; % take them out of service
    Q2(i) = Q2(i) - 1; % remove one person from class 2 from system
elseif S == 3 && W == 0
    S = 0; % take them out of service
    Q3(i) = Q3(i) - 1; % remove one person from class 3 from system
end
if S == 0 && W == 0
    % now to check if someone is available for service and to add their
    % work requirement, checking in order of priority
    if Q1(i) > 0 && length(v1) > 0
        S = 1;
        W = v1(1);
        v1(1) = [];
    elseif Q2(i) > 0 && length(v2) > 0
        S = 2;
        W = v2(1);
        v2(1) = [];
    elseif Q3(i) > 0 && length(v3) > 0
        S = 3;

```

```
        W = v3(1);
        v3(1) = [];
    end
end

W = max(W-dt,0);
end
TOTAL_COST= C1*trapz(t,Q1) + C2*trapz(t,Q2) + C3*trapz(t,Q3);

system_total = Q1+Q2+Q3; %i.e. total number of people in system

plot(t,system_total,'black','linewidth',1.5); hold on;
plot(t,Q1,'red'); hold on;
plot(t,Q2,'yellow'); hold on;
plot(t,Q3,'green'); hold off;
legend({'SystemTotal','Priority 1','Priority 2','Priority 3'})
title(['System Cost: $', num2str(TOTAL_COST)])

end
```

## 4 References

Chen, H., & Yao, D. D. (2001). Ch.5 Technical Desiderata. In *Fundamentals of queuing networks: Performance, asymptotics, and optimization* (pp. 97-112). New York, NY: Springer.

Chen, H., & Yao, D. D. (2001). Ch.6 Single-Station Queues. In *Fundamentals of queuing networks: Performance, asymptotics, and optimization* (pp. 125-144). New York, NY: Springer.

Shaikhet, G. (2019). *Personal Communications*. Ottawa, ON: Carleton University