The Double Queue to
The Generalized
Join-the-Shortest-Queue Model

by
Brendan Maher

an honours project submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Bachelor of Science in
Probability and Statistics

Carleton University
Ottawa, Ontario

Contents

1.	Acknowledgements	1
2.	Abstract	1
3.	Introduction	1
4.	The Double Queue	2
5.	The DQBD Process	2-3
6.	The Kernel Method	4-5
7.	Iterative Power Method	5
8.	Results	5
9.	Bibliography	5
10.	Appendix: Python Class DQ	6-7

1. Acknowledgements

Backer Carleton University, Supervisor Yiqiang Q. Zhao

2. Abstract

The problem that I am working is to finding the decay rates of a random walk in the first and second quadrants. We have both the DQBD process and the Kernel method to find these rates, which may end up giving different answers. I will by applying these methods to a simple double queue with one axis on the minimum queue and the other axis on the difference between the queues. If discrepancies arise I will use the iterative power method to find the correct method.

3. Introduction

Motivation: There is a Markov chain describing a random walk on first and second quadrants. The decay rates of the probability of position along the borders need to be found. The problem is that there are two different ways of finding the decay rates such as the Kernel Method and the DQBD Process. This problem can be simplified by using a level two shortest queue model.

Level two Shortest Queue Model: There are two queues that are providing a service. The number of people in the queues can be described with one number for people in the minimum queue and one number for the difference of people in the two queues.

Kernel Method: There are generating functions for each of the borders. The decay rates are simply the inverses of the smallest singularities greater than one for each generating function.

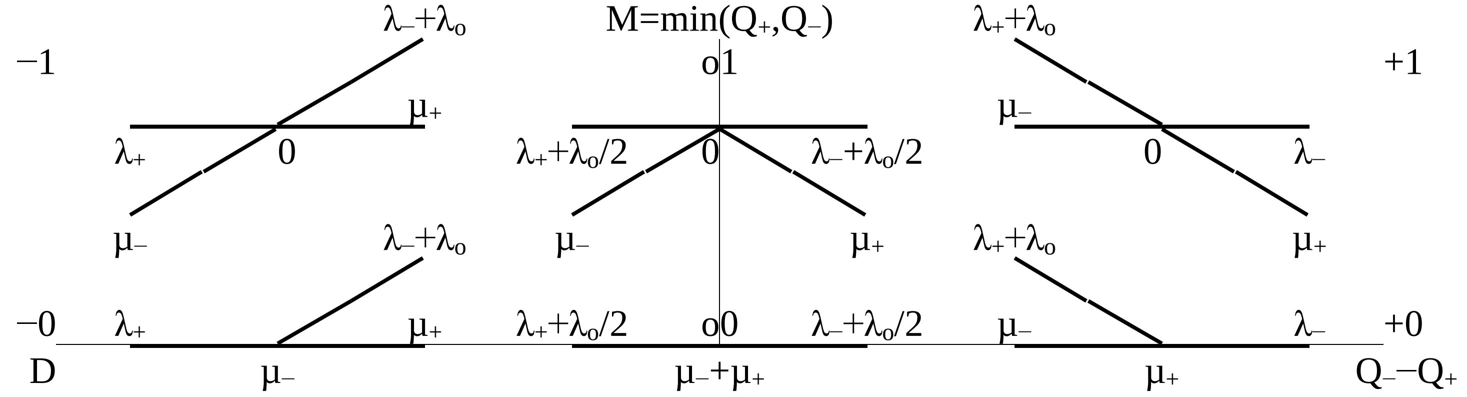
DQBD Process: The double sided quasi birth death process is a continuous time Markov chain with a level and background state. This Markov chain can be described with ellipses and curves passing through the origin for both quadrants, with which agreement leads to the decay rates.

Contributions: Through this paper the discrepancies will be found, and by using the created Python class DQ, the Kernel Method will be shown to succeed and while the DQBD Process fails.

4. The Double Queue

There are people arriving and being served at two queues (Q_+,Q_-) with $D=Q_- - Q_+$ and $M=\min(Q_+,Q_-)$. We need to find rates r_o and r_+ and r_- at which $M\rightarrow\infty$ and $D\rightarrow\infty$ and $D\rightarrow-\infty$. To keep things simple arrival and serving rates add to one and follow exponential distributions so that the stationary probabilities match. People arrive at are served at Q_+ by rates λ_+ and μ_+. People arrive at are served at Q_- by rates λ_- and μ_-. People will join the shortest queue at rate λ_o.

Let $\lambda_+ + \lambda_- + \lambda_o + \mu_+ + \mu_- = 1$ This gives the following transition diagram



This gives the following balance equations:

$\pi_{0,0}=\pi_{0,0}(\mu_-+\mu_+)+\pi_{-1,0}\mu_+ + \pi_{1,0}\mu_-$

$\pi_{1,0}= \pi_{2,0}\mu_- + (\pi_{1,0}+\pi_{0,1})\mu_+ + \pi_{0,0}(\lambda_- + \lambda_o/2)$

$\pi_{n>1,0}=\pi_{n+1,0}\mu_- + (\pi_{n,0}+\pi_{n-1,m+1})\mu_+ + \pi_{n-1,0}\lambda_-$

$\pi_{n<-1,0}=\pi_{n-1,0}\mu_+ + (\pi_{n,0}+\pi_{n+1,m+1})\mu_- + \pi_{n+1,0}\lambda_+$

$\pi_{-1,0}= \pi_{-2,0}\mu_+ + (\pi_{-1,0}+\pi_{0,1})\mu_- + \pi_{0,0}(\lambda_+ + \lambda_o/2)$

$\pi_{-1,m>0}=\pi_{-2,m-1}(\lambda_- + \lambda_o)+\pi_{-2,m}\mu_+ + \pi_{0,m}(\lambda_+ + \lambda_o/2)+\pi_{0,m+1}\mu_-$

$\pi_{1,m>0}=\pi_{2,m-1}(\lambda_+ + \lambda_o)+\pi_{2,m}\mu_- + \pi_{0,m}(\lambda_- + \lambda_o/2)+\pi_{0,m+1}\mu_+$

$\pi_{0,m>0}=\pi_{-1,m-1}(\lambda_- + \lambda_o)+\pi_{-1,m}\mu_+ + \pi_{1,m}\mu_- + \pi_{1,m-1}(\lambda_+ + \lambda_o)$

$\pi_{n<-1,m>0}=\pi_{n-1,m-1}(\lambda_- + \lambda_o)+\pi_{n-1,m}\mu_+ + \pi_{n+1,m}\lambda_+ + \pi_{n+1,m+1}\mu_-$

$\pi_{n>1,m>0}=\pi_{n+1,m-1}(\lambda_+ + \lambda_o)+\pi_{n+1,m}\mu_- + \pi_{n-1,m}\lambda_- + \pi_{n-1,m+1}\mu_+$

5. The DQBD Process

The Double sided Quasi Birth Death Process is a continuous time Markov chain with a level and background state which is shown below. $k=0,\pm1$ $i=o,\pm$ $j=0,1$

$$X_k^{(i)}=\left|\begin{array}{ccccc} p^{(i0)}_{k,0} & p^{(i0)}_{k,1} & 0 & \dots & \\ p^{(i1)}_{k,-1} & p^{(i1)}_{k,0} & p^{(i1)}_{k,1} & 0 & \dots \\ 0 & p^{(i1)}_{k,-1} & p^{(i1)}_{k,0} & p^{(i1)}_{k,1} & 0 \\ | & & & & \end{array}\right| \qquad Y_k^{(j)}=\left|\begin{array}{ccccc} p^{(-j)}_{-1,k} & p^{(-j)}_{0,k} & p^{(-j)}_{1,k} & 0 & \dots \\ 0 & p^{(oj)}_{-1,k} & p^{(oj)}_{0,k} & p^{(oj)}_{1,k} & 0 \\ \dots & 0 & p^{(+j)}_{-1,k} & p^{(+j)}_{0,k} & p^{(+j)}_{1,k} \end{array}\right|$$

P_X has the difference for level and min for background

$$P_X=\left|\begin{array}{ccccc} X_{-1}^{(-)} & X_0^{(-)} & X_1^{(-)} & 0 & \dots \\ 0 & X_{-1}^{(o)} & X_0^{(o)} & X_1^{(o)} & 0 \\ \dots & 0 & X_{-1}^{(+)} & X_0^{(+)} & X_1^{(+)} \end{array}\right| \qquad P_Y=\left|\begin{array}{ccccc} Y_0^{(0)} & Y_1^{(0)} & 0 & \dots & \\ Y_{-1}^{(1)} & Y_0^{(1)} & Y_1^{(1)} & 0 & \dots \\ 0 & Y_{-1}^{(1)} & Y_0^{(1)} & Y_1^{(1)} & 0 \\ | & & & & \end{array}\right|$$

P_Y has the min for level and difference for background

$p_o=(\lambda_+ +\lambda_- +\lambda_o)/(\mu_- +\mu_+)$ $\quad p_+=\lambda_+/\mu_+ \quad q_+=p_+(\lambda_- +\lambda_o)/\mu_- \quad s_-=\mu_- p_o^2+\lambda_+$

$\quad p_-=\lambda_-/\mu_- \quad q_-=p_-(\lambda_+ +\lambda_o)/\mu_+ \quad s_+=\mu_+ p_o^2+\lambda_-$

For both quadrants the DQBD Process is described with the following curves through the origin. $E(\exp(\theta D^{(-)}+\varphi M^{(-)}))=1=E(\exp(\theta D^{(+)}+\varphi M^{(+)}))$ which are extended upon and made to agree below.

$$(\theta_{-c},\varphi_{-c})=\begin{cases}(-\ln(p_-),-\ln(p_-)) & \mu_->\lambda_- +\lambda_o\\ (\theta_{-\max},\varphi_{-\max}) & \mu_-\le\lambda_- +\lambda_o\end{cases} \qquad (\theta_{+c},\varphi_{+c})=\begin{cases}(-\ln(p_+),-\ln(p_+)) & \mu_+>\lambda_+ +\lambda_o\\ (\theta_{+\max},\varphi_{+\max}) & \mu_+\le\lambda_+ +\lambda_o\end{cases}$$

$\theta_{-\max}=\ln(1-2\text{ʒ}_-^{1/2}+(1+4\text{ʒ}_- -\text{ʒ}_-^{1/2}-\lambda_+\mu_+)^{1/2})-\ln2-\ln\lambda_+ \qquad \text{ʒ}_-=\mu_-(\lambda_- +\lambda_o)$

$\theta_{+\max}=\ln(1-2\text{ʒ}_+^{1/2}+(1+4\text{ʒ}_+ -\text{ʒ}_+^{1/2}-\lambda_-\mu_-)^{1/2})-\ln2-\ln\lambda_- \qquad \text{ʒ}_+=\mu_+(\lambda_+ +\lambda_o)$

$\varphi_{-\max}=\ln(1-4(\lambda_+\mu_+ +\text{ʒ}_-)-(1-4(\lambda_+\mu_+ +\text{ʒ}_-)^2-64\lambda_+\mu_+\text{ʒ}_-)^{1/2})-\ln8-\ln\lambda_+-\ln(\lambda_- +\lambda_o)$

$\varphi_{+\max}=\ln(1-4(\lambda_-\mu_- +\text{ʒ}_+)-(1-4(\lambda_-\mu_- +\text{ʒ}_+)^2-64\lambda_-\mu_-\text{ʒ}_+)^{1/2})-\ln8-\ln\lambda_--\ln(\lambda_+ +\lambda_o)$

For both quadrants the DQBD Process is described with the following ellipses through the origin. $E(\exp(\eta_- D^{(-)}+\eta_o M^{(-)}))=1=E(\exp(\eta_+ D^{(+)}+\eta_o M^{(+)}))$ which are extended and made to agree below.

$e^{\eta_-}(\lambda_+ +\mu_-/e^{\eta_o})+(e^{\eta_o}(\lambda_o+\lambda_-)+\mu_+)/e^{\eta_-}=1=e^{\eta_+}(\lambda_- +\mu_+/e^{\eta_o})+(e^{\eta_o}(\lambda_o+\lambda_+)+\mu_-)/e^{\eta_+}$

$((\lambda_+ +\lambda_o/2)+\mu_-/e^{\eta_o})/e^{\eta_-}+e^{\eta_+}((\lambda_- +\lambda_o/2)+\mu_+/e^{\eta_o})=1=((\lambda_- +\lambda_o/2)+\mu_+/e^{\eta_o})/e^{\eta_+}+e^{\eta_-}((\lambda_+ +\lambda_o/2)+\mu_-/e^{\eta_o})$

$t_-(v)=\min(z-1|\lambda_+ +\mu_- v=((\lambda_+ +\lambda_o)/v+\mu_-) \qquad t_+(v)=\min(z-1|\lambda_- +\mu_+ v=((\lambda_- +\lambda_o)/v+\mu_+)$

$t_-(r_o)=(r_o(\lambda_+ +\mu_- r_o)/(\mu_+ r_o+\lambda_- +\lambda_o))^{1/2} \qquad t_+(e^{-\eta_o})=((\lambda_- e^{\eta_o}+\mu_+)/(\mu_- e^{\eta_o}+\lambda_+ +\lambda_o))^{1/2}$

Calculating decay rates can now be done with the equations below split into nine cases.

caseA~$(s_- +\lambda_o>s_+,\ s_+ +\lambda_o>s_-)$:

$\quad \eta_o=-2\ln p_o \qquad \eta_{-1}=-\ln p_o=\eta_{+1}$

$\quad$caseAa~$(p_-<p_o,\ p_+<p_o)$:

$$r_o=p_o^2 \qquad r_+=\begin{cases}e^{-\theta_{+c}} & \varphi_{+c}\le-\ln p_o\\ p_o s_+/(s_- +\lambda_o) & \varphi_{+c}>-\ln p_o\end{cases} \qquad r_-=\begin{cases}e^{-\theta_{-c}} & \varphi_{-c}\le-\ln p_o\\ p_o s_-/(s_+ +\lambda_o) & \varphi_{-c}>-\ln p_o\end{cases}$$

$\quad$caseAb~$(p_-\ge p_o,\ p_+<p_o)$:

$$r_o=q_- \qquad r_+=p_+ \qquad r_-=\begin{cases}e\wedge-\theta_1^{(-c)} & \varphi_{-\max}\le-\ln r_o\\ t_-(r_o) & \varphi_{-\max}>-\ln r_o\end{cases}$$

$\quad$caseAc~$(p_-<p_o,p_+\ge p_o)$: by symmetry the reverse of caseAB

caseB~$(s_- +\lambda_o\le s_+,\ s_+ +\lambda_o>s_-)$:

$\quad \eta_o=\varphi_{-\max} \qquad \eta_{-1}=\eta_o-\ln2-\ln(\lambda_+ e^{\eta_o}+\mu_-)$

$\quad \eta_{+1}=\ln(1-(\lambda_+ +\lambda_o/2+\mu_-/e^{\eta_o})/e^{\eta_-})-\ln(\mu_+/e^{\eta_o}+\lambda_- +\lambda_o/2)$

$\quad$caseBa~$(p_-<p_o,\ p_+<p_o)$:

$$r_-=e^{-\theta_{-c}} \qquad (r_o,r_+)=\begin{cases}(q_-,p_-) & \theta_{+c}<\eta_{+1}\\ (e^{-\varphi_{-\max}},\min(e^{-\theta_{+c}},\ t_+(e^{-\eta_o}))) & \theta_{+c}\ge\eta_{+1}\end{cases}$$

$\quad$caseBb~$(p_-\ge p_o,p_+<p_o)$:

$$r_o=q_- \qquad r_+=p_- \qquad r_-=\begin{cases}e^{-\theta_{-c}} & \theta_2^{(-c)}<-\ln r_o\\ t_-(r_o) & \theta_2^{(-c)}\ge-\ln r_o\end{cases}$$

$\quad$caseBc~$(p_-<p_o,p_+\ge p_o)$:

$$(r_-,r_o,r_+)=\left|\begin{array}{l}(p_+,q_+,\min(e^{-\theta_{+c}},t_+(r_o)))\begin{cases}\eta_{-1}\ge-\ln p_+,\eta_{+1}<\theta_1^{(+c)}\\ \eta_{-1}\ge-\ln p_+,\eta_{+1}\ge\theta_1^{(+c)},q_+<q_-\end{cases}\\ (p_+,\ e^{-\varphi_{-\max}},\ e^{-\theta_{+c}})\text{---------}\ \eta_{-1}<-\ln p_+,\eta_{+1}<\theta_1^{(+c)}\\ (\min(e^{-\theta_{-c}},t_-(r_o)),q_-,p_+)\begin{cases}\eta_{-1}<-\ln p_+,\eta_{+1}\ge\theta_1^{(+c)}\\ \eta_{-1}\ge-\ln p_+,\eta_{+1}\ge\theta_1^{(+c)},q_+\ge q_-\end{cases}\end{array}\right.$$

caseC~$(s_- +\lambda_o>s_+,s_+ +\lambda_o\le s_-)$: by symmetry the reverse of caseB

6. The Kernel Method

The Kernel equations are given below.

$h_{+1}(x,y)=1-\sum_{i,j=0,\pm1}p^{(+1)}_{j,i}x^iy^j=1-\mu_+y/x-(\lambda_++\lambda_o)x/y-\mu_-/y-\lambda_-y$

$h_{-1}(x,y)=1-\sum_{i,j=0,\pm1}p^{(-1)}_{j,i}x^iy^{-j}=1-\mu_-y/x-(\lambda_-+\lambda_o)x/y-\mu_+/y-\lambda_+y$

$h_{+0}(x,y)=\sum_{i=0,1}\sum_{j=0,\pm1}p^{(+0)}_{j,i}x^iy^j-1=\lambda_-y+\mu_++\mu_-/y-(\lambda_++\lambda_o)x/y-1$

$h_{-0}(x,y)=\sum_{i=0,1}\sum_{j=0,\pm1}p^{(-0)}_{j,i}x^iy^{-j}-1=\lambda_+y+\mu_-+\mu_+/y-(\lambda_-+\lambda_o)x/y-1$

$h_{+o1}(x,y)=\sum_{i=0,\pm1}\sum_{j=0,1}p^{(o1)}_{j,i}x^iy^j-1=\mu_+y/x+(\lambda_-+\lambda_o/2)y-1$

$h_{-o1}(x,y)=\sum_{i=0,\pm1}p^{(o1)}_{-1,i}x^iy=\mu_+y/x+(\lambda_-+\lambda_o/2)y$

$h_{+o0}(x,y)=\sum_{i,j=0,1}p^{(o0)}_{j,i}x^iy^j-1=\mu_-+\mu_++(\lambda_-+\lambda_o/2)y-1$

$h_{-o0}(x,y)=\sum_{i,=0,1}p^{(o0)}_{-1,i}x^iy=(\lambda_++\lambda_o/2)y$

The Kernel Method gets the decay rates from the inverse of the dominant singularities (smallest singularity greater than one) of the generating functions below.

$P_0(x)=(f_+(x)+f_-(x)+g_0(x))/g_1(x)\quad Q_{+0}(y)=(k_{+o1}(y)+k_{+o0}(y))/g_{+2}(y)\quad Q_{-0}(y)=(k_{-o1}(y)+k_{-o0}(y))/g_{-2}(y)$

These require the following equations.

$g_{o0}(x)=-\pi_{0,0}(h_{+o0}(x,Y_{+0}(x))+h_{-o0}(x,Y_{-0}(x)))\quad g_{o1}(x)=h_{+o1}(x,Y_{+0}(x))+h_{-o1}(x,Y_{-0}(x))$

$f_+(x)=-Q_{+0}(Y_{+0}(x))h_{+0}(x,Y_{+0}(x))\quad k_{+o1}(y)=-P_0(X_{+0}(y))h_{+o1}(X_{+0}(y),y)\quad g_{+0}(y)=h_{+0}(X_{+0}(y),y)$

$f_-(x)=-Q_{-0}(Y_{-0}(x))h_{-0}(x,Y_{-0}(x))\quad k_{-o1}(y)=-P_0(X_{-0}(y))h_{-o1}(X_{-0}(y),y)\quad g_{-0}(y)=h_{-0}(X_{-0}(y),y)$

$k_{+o0}(y)=-\pi_{0,0}h_{+o0}(x,Y_{+0}(x))-P_{+1}(X_{+0}(y))A_+(X_{+0}(y))-\pi_{-1,0}B+(X_{+0}(y))$

$k_{-o0}(y)=-\pi_{0,0}h_{-o0}(x,Y_{-0}(x))-P_{-1}(X_{-0}(y))A_-(X_{-0}(y))-\pi_{+1,0}B+(X_{-0}(y))$

The Kernel equations $h_{+1}(x, y)=0=h_{-1}(x, y)$ give algebraic curves which turn out to be Riemann surfaces. The zeros of these Kernel equations are called branch points of the fundamental form.

$0=xyh_{+1}(x,y)=a_{+x}(x)y^2+b_{+x}(x)y+c_{+x}(x)=a_{+y}(y)x^2+b_{+y}(y)x+c_{+y}(y)$

$0=xyh_{-1}(x,y)=a_{-x}(x)y^2+b_{-x}(x)y+c_{-x}(x)=a_{-y}(y)x^2+b_{-y}(y)x+c_{-y}(y)$

$$Y_{\pm0}(x)=\begin{matrix}(b_{\pm x}(x)+d_{\pm x}(x)^{1/2})/(-2a_{\pm x}(x)) & b_{\pm x}(x)>0\\ (b_{\pm x}(x)-d_{\pm x}(x)^{1/2})/(-2a_{\pm x}(x)) & b_{\pm x}(x)<0\end{matrix}\qquad X_{\pm0}(y)=\begin{matrix}(b_{\pm y}(y)+d_{\pm y}(y)^{1/2})/(-2a_{\pm y}(y)) & b_{\pm y}(y)>0\\ (b_{\pm y}(y)-d_{\pm y}(y)^{1/2})/(-2a_{\pm y}(y)) & b_{\pm y}(y)<0\end{matrix}$$

$$Y_{\pm1}(x)=\begin{matrix}(b_{\pm x}(x)-d_{\pm x}(x)^{1/2})/(-2a_{\pm x}(x)) & b_{\pm x}(x)>0\\ (b_{\pm x}(x)+d_{\pm x}(x)^{1/2})/(-2a_{\pm x}(x)) & b_{\pm x}(x)<0\end{matrix}\qquad X_{\pm1}(y)=\begin{matrix}(b_{\pm y}(y)-d_{\pm y}(y)^{1/2})/(-2a_{\pm y}(y)) & b_{\pm y}(y)>0\\ (b_{\pm y}(y)+d_{\pm y}(y)^{1/2})/(-2a_{\pm y}(y)) & b_{\pm y}(y)<0\end{matrix}$$

The fundamental forms above are described with the following equations.

$a_{+x}(x)=-p_{1,1}^{(+1)}x^2-p_{1,0}^{(+1)}\,x-p_{1,-1}^{(+1)}=-\lambda_-x-\mu_+\qquad a_{+y}(y)=-p_{1,1}^{(+1)}y^2-p_{0,1}^{(+1)}y-p_{-1,\,1}^{(+1)}=-\lambda_+-\lambda_o$

$a_{-x}(x)=-p_{-1,1}^{(-1)}x^2-p_{-1,0}^{(-1)}\,x-p_{-1,-1}^{(-1)}=-\lambda_+x-\mu_-\quad a_{-y}(y)=-p_{-1,1}^{(-1)}y^2-p_{0,1}^{(-1)}y-p_{1,1}^{(-1)}=-\lambda_--\lambda_o$

$b_{+x}(x)=-p_{0,1}^{(+1)}x^2+x-p_{0,-1}^{(+1)}=x\qquad b_{+y}(y)=-p_{1,0}^{(+1)}\,y^2+y-p_{-1,0}^{(+1)}=-\lambda_-y^2+y-\mu_-$

$b_{-x}(x)=-p_{0,1}^{(-1)}x^2+x-p_{0,-1}^{(-1)}=x\qquad b_{-y}(y)=-p_{-1,0}^{(-1)}y^2+y-p_{1,0}^{(-1)}=-\lambda_+y^2+y-\mu_+$

$c_{+x}(x)=-p_{1,1}^{(+1)}x^2-p_{1,0}^{(+1)}x-p_{1,-1}^{(+1)}=-\lambda_-x-\mu_+\qquad c_{+y}(y)=-p_{1,1}^{(+1)}y^2-p_{0,1}^{(+1)}y-p_{-1,1}^{(+1)}=-y^2(\lambda_++\lambda_o)$

$c_{-x}(x)=-p_{-1,1}^{(-1)}x^2-p_{-1,0}^{(-1)}x-p_{-1,-1}^{(-1)}=-\lambda_+x-\mu_-\quad c_{+y}(y)=-p_{-1,1}^{(-1)}y^2-p_{0,1}^{(-1)}y-p_{1,1}^{(-1)}=-y^2(\lambda_-+\lambda_o)$

$0=d_{\pm z}(z)^2=b_{\pm z}(z)^2-4a_{\pm z}(z)c_{\pm z}(z)$ $z=x,y$ which give at most four branch points $|z_{\pm1}|\le|z_{\pm2}|\le|z_{\pm3}|\le|z_{\pm4}|\le\infty$

$$p_{0,1}^{(+1)2}\begin{matrix}>\\=\\<\end{matrix}4p_{1,1}^{(+1)}p_{-1,\,1}^{(+1)}\quad 1<x_{+3}\begin{matrix} & <\infty\\ <x_{+4} & =\infty\\ \le -x_{+4} & <\infty\end{matrix}\qquad p_{0,-1}^{(+1)2}\begin{matrix}>\\=\\<\end{matrix}4p_{1,-1}^{(+1)}p_{-1,-1}^{(+1)}\quad\begin{matrix}0< & \\ 0= & x_{+1}<\\ 0<-x_{+1}\le & \end{matrix}x_{+2}<1$$

By symmetry of the above lemma $\quad 0=x_{\pm1}<x_{\pm2}<1<x_{\pm3}<x_{\pm4}=\infty\qquad 0<y_{\pm1}<y_{\pm2}<1<y_{\pm3}<y_{\pm4}<\infty$

$x_{\pm3}$ and $y_{\pm3}$ give the first four singularities

The zeros of $g_{o1}(x)$ and $g_{\pm0}(y)$ will give the next three singularities x_0 and $y_{\pm0}$
$0=g_{+0}(y)=h_{+0}(X_{+0}(y),y)=\lambda_-y+\mu_++\mu_-/y+X_{+0}(y)(\lambda_++\lambda_o)-1$
$\lambda_-y^3(\lambda_-y-2)+y^2(1+2\lambda_-\mu_-4(\lambda_++\lambda_o)\mu_+)=d_{+y}(y)^2=(\lambda_-y^2+(2\mu_+-1)y+\mu_-)^2$
This gives solution $y_{+0}=\mu_-/\lambda_-=1/p_-$ and by symmetry $y_{-0}=\mu_+/\lambda_+=1/p_+$ which led me to hypothesize and then confirm that $x_0=p_o^{-2}$ The final four singularities are provided by $x_{\pm y}=X_{\pm1}(y_{\pm0})$ and $y_{\pm x}=Y_{\pm1}(x_0)$ The decay rates can now be calculated as
$r_o=1/\min_{x>1}(x_{+3}, x_{-3}, x_0, x_{+y}, x_{-y})$ $r_+=1/\min_{y>1}(y_{+3}, y_{+0}, y_{+x})$ $r_-=1/\min_{y>1}(y_{-3}, y_{-0}, y_{-x})$

7. Iterative Power Method

The Power Method is simply taking a starting point and then using the balance equations to get the position probabilities after so many steps. For instance starting at the origin
$p_{0,0}^0=1$, $p_{0,0}^1=p_{0,0}^0(\mu_-+\mu_+)$, $p_{0,0}^2=p_{0,0}^1(\mu_-+\mu_+)+p_{-1,0}^1\mu_++p_{1,0}^1\mu_-,\ldots,p_{0,0}^*=\pi_{0,0}$ this also expresses the surrounding positions. For the decay rates I picked border points far from the origin.

8. Results

By using the provided python class DQ($\lambda_+,\lambda_-,\lambda_o,\mu_+,\mu_-$), many cases can be tested in succession. The case rt=DQ(0.1,0.2,0.1,0.3,0.3), rt.tails() returns.

```
[[0.358974358974359, 0.4444444444444444, 0.6666666666666667, 'Kernel'],
 [0.57112773836058, -0.00454860520502576, 0.326186893524866, '-Power'],
 [0.65963284315925, 6.6713261685059e-05, 0.435115487774358, 'oPower'],
 [0.81224870404007, -0.000224390689413934, 0.65974795721478, '+Power'],
 [0.489199590470229, 0.4444444444444444, 0.6666666666666667, 'DQBD']]
```

r_o and r_+ both agree for the Kernel Method and DQBD Process but r_- does not. The Iterative Power Method does not seem to work for either. Although by instead using the inversed square roots of the dominant singularities from the Kernel Method, we get following relative errors.

```
(0.049055490169546, 0.010486897877028, 0.005229773672183) #rt[0][1:3]**0.5/ rt[1:3][2]−1
```

I suspect the success of the much more intuitive Kernel Method comes from the generating functions staying within their respective quadrants.

9. Bibliography

- Zafar Zafari, University of British Columbia, Yiqiang Q. Zhao, Carleton University, Javad Tavakoli, University of British Columbia, 26 March 2012, THE EXACT TAIL ASYMPTOTICS ANALYSIS OF A GENERAL RANDOM WALK MODEL IN THE FIRST AND FOURTH QUADRENTS: KERNEL METHOD APPROACH
- Masakiyo Miyazawa, Department of Information Sciences, Tokyo University of Science, 2009, Two Sided DQBD Process and Solutions to the Tail Decay Rate Problem and Their Applications to the Generalized Join Shortest Queue, *Advances in Queueing Theory and Network Applications*, 3-33

10. Appendix: Python Class DQ

The following code describes a simple Double Queue such that people arrive at and leave the two queues according to exponential distributions where the rates add up to one. The three rates to look at are, the rate at which probability goes to zero for equal queues going to infinity, and rates at which probability goes to zero for the difference between the queues going to negative and positive infinity. A DQ class is initialized with one queue picked to be negative so that, the difference between queues will be negative when it is smaller. Methods tailsK for the Kernel Method, and tailsD for the DQBD Process, returns the rates as [r_-,r_o,r_+,“name”]. The Iterative Power Method has methods tailn and tailo and tailp, return [r,error$_r$,r^2,“name”] for r_- and r_o and r_+ respectively. Methods tailsP returns [tailn,tailo,tailp] while tails returns [tailsK]+tailsP+[tailsD].

```
class DQ:
 def __init__(self,np=0.1,nn=0.1,no=0.2,up=0.3,un=0.3):
   if (np+nn+no+up+un!=1)|(np<=0)|(nn<=0)|(no<=0)|(up<=0)|(un<=0)|(np>=up)|(nn>=un)|(up+un<=0.5): print("bad input")
   else: self.__Np,self.__Nn,self.__No,self.__Up,self.__Un=np,nn,no,up,un
 def __TailsK(np,nn,no,up,un):
   from pylab import poly1d
   po,pp,pn=(np+nn+no)/(up+un),np/up,nn/un
   axp,axn,bx,cxp,cxn=poly1d([nn,up]),poly1d([np,un]),poly1d([1.,0.]),poly1d([np+no,un,0.]),poly1d([nn+no,up,0.])
   ayp,ayn,byp,byn,cyp,cyn=np+no,nn+no,poly1d([-nn,1.,-un]),poly1d([-np,1.,-up]),poly1d([up,0.,0.]),poly1d([un,0.,0.])
   dxp,dxn,dyp,dyn=bx**2-4*axp*cxp,bx**2-4*axn*cxn,byp**2-4*ayp*cyp,byn**2-4*ayn*cyn
   kp,kn,ko=min(1/pn,dyp.r[1]),min(1/pp,dyn.r[1]),min(dxp.r[0],po**-2,dxn.r[0])
   kyp,kyn=(byp(1/pn)+dyp(1/pn)**0.5)/2/ayp,(byn(1/pp)+dyn(1/pp)**0.5)/2/ayn
   if kyp>1: ko=min(ko,kyp)
   if kyn>1: ko=min(ko,kyn)
   kxp,kxn=(po**-2+dxp(po**-2)**0.5)/2/axp(po**-2),(po**-2+dxn(po**-2)**0.5)/2/axn(po**-2)
   if kxp>1: kp=min(kp,kxp)
   if kxn>1: kn=min(kn,kxn)
   return [1/kn,1/ko,1/kp,"Kernel"]
 def tailsK(self): return DQ.__TailsK(self.__Np,self.__Nn,self.__No,self.__Up,self.__Un)
 def __TailsD(np,nn,no,up,un):
   from math import log,e
   po,pp,pn=(np+nn+no)/(up+un),np/up,nn/un
   yp,yn,qp,qn=up*po**2+nn,un*po**2+np,pp*(nn+no)/un,pn*(np+no)/up
   mn1=log(1-2*(un*(nn+no))**0.5+(1+4*(un*(nn+no))-(un*(nn+no))**0.5-up*np)**0.5)-log(2)-log(np)
   mp1=log(1-2*(up*(np+no))**0.5+(1+4*(up*(np+no))-(up*(np+no))**0.5-un*nn)**0.5)-log(2)-log(nn)
   mn2=log(1-4*(np*up+un*(nn+no))+((1-4*(np*up+un*(nn+no)))**2-64*(nn+no)*np*un*up)**0.5)-log(8*np)-log(nn+no)
   mp2=log(1-4*(nn*un+up*(np+no))+((1-4*(nn*un+up*(np+no)))**2-64*(np+no)*nn*up*un)**0.5)-log(8*nn)-log(np+no)
   sp1,sp2,sn1,sn2,co,cp,cn=mp1**1,mp2**1,mn1**1,mn2**1,-2*log(po),-log(po),-log(po)
   if un>nn+no: sn1,sn2=-log(pp),-log(pp)
   if up>np+no: sp1,sp2=-log(pn),-log(pn)
   rn,ro,rp=e**-sn1,po**2,e**-sp1
   if (yn+no>yp)&(yp+no>yn):
     if (pp<po)&(pn<po):
       if sn2>-2*log(po): rn=po*yn/(yp+no)
       if sp2>-2*log(po): rp=po*yp/(yn+no)
     elif pn<po:
       if sp2>-log(qp): rp=(qp*(nn+up*qp)/(un*qp+np+no))**0.5
       ro,rn=qp,pp
     else:
       if sn2>-log(qn): rn=(qn*(np+un*qn)/(up*qn+nn+no))**0.5
       ro,rp=qn,pn
   elif yp+no>yn:
     co,cn=mn2**1,mn2-log(2)-log(np*e**mn2+un)
     cp=log(1-(np+no/2+un/e**co)/e**cn)-log(up/e**co+nn+no/2)
     if (pp<po)&(pn<po):
       if sp1<cp: ro,rp=qn,pn
       else: ro,rp=e**-mn2,min(rp,((nn*e**co+up)/(un*e**co+np+no))**0.5)
     elif pp<po:
       if sn2>=-log(qn): rn=(qn*(np+un*qn)/(up*qn+nn+no))**0.5
       ro,rp=qn,pn
     else:
       if (cn<-log(pp))&(cp<sp1): ro,rn=e**-mn2,pp
       elif (cp<sp1)|(cn>=-log(pp))&(qp<qn): ro,rn,rp=qp,pp,min(rp,(qp*(nn+up*qp)/(un*qp+np+no))**0.5)
       else: ro,rp,rn=qn,pn,min(rn,(qn*(np+un*qn)/(up*qn+nn+no))**0.5)
   else:
     co,cp=mp2**1,mp2-log(2)-log(nn*e**mp2+up)
     cn=log(1-(nn+no/2+up/e**co)/e**cp)-log(un/e**co+np+no/2)
     if (pn<po)&(pp<po):
       if sn1<cn: ro,rn=qp,pp
       else: ro,rn=e**-mp2,min(rn,((np*e**co+un)/(up*e**co+nn+no))**0.5)
     elif pn<po:
       if sp2>=-log(qp): rp=(qp*(nn+up*qp)/(un*qp+np+no))**0.5
       ro,rn=qp,pp
     else:
       if (cp<-log(pn))&(cn<sn1): ro,rp=e**-mp2,pn
       elif (cn<sn1)|(cp>=-log(pn))&(qn<qp): ro,rp,rn=qn,pn,min(rn,(qn*(np+un*qn)/(up*qn+nn+no))**0.5)
       else: ro,rn,rp=qp,pp,min(rp,(qp*(nn+up*qp)/(un*qp+np+no))**0.5)
   return [rn,ro,rp,"DQBD"]
 def tailsD(self): return DQ.__TailsD(self.__Np,self.__Nn,self.__No,self.__Up,self.__Un)
 def __Tailpn(m=200,np=0.1,nn=0.1,no=0.2,up=0.3,un=0.3,k=2,mo=[[0.,0.],[1.,0.],[0.,0.]],ms=[[0.3,0.3],[0.3,0.],[0.1,0.]]):
   if k>m: return [ms[k][0]/ms[k-1][0],ms[k][0]/ms[k-1][0]-mo[k][0]/mo[k-1][0],(ms[k][0]/ms[k-1][0])**2]
   mo,ms=[mo[0]+[0.]]+mo+[mo[0]+[0.]],[mo[0]+[0.]]+ms+[mo[0]+[0.]]
   for i in range(1,2*k):
     mo[i],ms[i]=mo[i]+[0.],ms[i]+[0.]
     ms[i][0]=nn*mo[i-1][0]+un*mo[i+1][0]+up*(mo[i][0]+mo[i-1][1])
     for j in range(1,k): ms[i][j]=up*mo[i-1][j+1]+nn*mo[i-1][j]+un*mo[i+1][j]+(np+no)*mo[i+1][j-1]
   return DQ.__Tailpn(m,np,nn,no,up,un,k+1,ms,mo)
 def tailp(self,m=200): return DQ.__Tailpn(m,self.__Np,self.__Nn,self.__No,self.__Up,self.__Un)+["+Power"]
 def tailn(self,m=200): return DQ.__Tailpn(m,self.__Nn,self.__Np,self.__No,self.__Un,self.__Up)+["-Power"]
 def __Tailo(m,np=0.1,nn=0.1,no=0.2,up=0.3,un=0.3,k=1,o=1.,mo=[[0.,0.,0.],[0.,0.,0.]],ms=[[1.,0.,0.],[0.,0.,0.]]):
   mo,ms=[mo[0]+[0.,0.,0.,0.]]+mo+[mo[0]]+[mo[0]+[0.,0.,0.,0.]],[mo[0]+[0.,0.,0.,0.]]+ms+[mo[0]]+[mo[0]+[0.,0.,0.,0.]]
   if k>m: return [ms[k-1][0]/ms[k][0],ms[k-1][0]/ms[k][0]-o,(ms[k-1][0]/ms[k][0])**2,"oPower"]
   mo[1],ms[1],o=mo[1]+[0.,0.,0.,0.],ms[1]+[0.,0.,0.,0.],ms[k-1][0]/ms[k][0]
   for i in range(1,3*k):
     mo[i+1],ms[i+1]=mo[i+1]+[0.,0.,0.,0.],ms[i+1]+[0.,0.,0.,0.]
     mo[i][1]=up*ms[i-1][0]+(nn+no/2)*ms[i][0]+un*ms[i][3]+(np+no)*ms[i+1][3]
     mo[i][2]=un*ms[i-1][0]+(np+no/2)*ms[i][0]+up*ms[i][4]+(nn+no)*ms[i+1][4]
     for j in range(1,k):
       mo[i][4*j+1]=up*ms[i-1][4*j-1]+nn*ms[i][4*j-1]+un*ms[i][4*j+3]+(np+no)*ms[i+1][4*j+3]
       mo[i][4*j+2]=un*ms[i-1][4*j]+np*ms[i][4*j]+up*ms[i][4*(j+1)]+(nn+no)*ms[i+1][4*(j+1)]
   ms[0][0]=(np+no)*mo[1][1]+(nn+no)*mo[1][2]
   for i in range(1,3*k+1):
     ms[i][0]=un*mo[i][1]+up*mo[i][2]+(np+no)*mo[i+1][1]+(nn+no)*mo[i+1][2]
     for j in range(1,k+1):
       ms[i][4*j-1]=up*mo[i-1][4*j-3]+nn*mo[i][4*j-3]+un*mo[i][4*j+1]+(np+no)*mo[i+1][4*j+1]
       ms[i][4*j]=un*mo[i-1][4*j-2]+np*mo[i][4*j-2]+up*mo[i][4*j+2]+(nn+no)*mo[i+1][4*j+2]
   return DQ.__Tailo(m,np,nn,no,up,un,k+1,o,mo,ms)
 def tailo(self,m=100): return DQ.__Tailo(m,self.__Nn,self.__Np,self.__No,self.__Un,self.__Up)
 def tailsP(self,m=100): return [self.tailn(2*m),self.tailo(m),self.tailp(2*m)]
 def tails(self,m=100): return [self.tailsK()]+self.tailsP(m)+[self.tailsD()]
```