Online Supplement for "Efficient and Reliable Computation of Birth-Death Process Performance Measures"

Armann Ingolfsson
School of Business, University of Alberta, Edmonton, Alberta T6G 2R6, Canada,
armann.ingolfsson@ualberta.ca

Ling Tang
Industrial Engineering and Operations Research Department, Columbia University, New York, NY 10027,
lt2326@columbia.edu

Figures 1-3 show pseudocode for algorithms 1 to 3, discussed in Section 2.

Initialization: compute $\pi_0, n \leftarrow 0, \text{converge} \leftarrow \text{FALSE}$
While not converge
    $n \leftarrow n + 1$
    $\pi_n \leftarrow \lambda_{n-1}/\mu_n \pi_{n-1}$
    If termination criteria are met, then converge $\leftarrow$ TRUE
$u \leftarrow n$
Return $\{\pi_0, \ldots, \pi_u\}$

Figure 1: Algorithm 1.

Initialization: $\bar{\pi}_0 \leftarrow 1, \Sigma \leftarrow 1, n \leftarrow 0, \text{converge} \leftarrow \text{FALSE}$
While not converge
    $n \leftarrow n + 1$
    $\bar{\pi}_n \leftarrow \lambda_{n-1}/\mu_n \bar{\pi}_{n-1}$
    $\Sigma \leftarrow \Sigma + \bar{\pi}_n$
    If termination criteria are met, then converge $\leftarrow$ TRUE
$u \leftarrow n$
Normalize: $\pi_i \leftarrow \bar{\pi}_i/\Sigma$ for $i = 0, \ldots, u$
Return $\{\pi_0, \ldots, \pi_u\}$

Figure 2: Algorithm 2.

Initialization: $\bar{\pi}_c \leftarrow 1, \Sigma \leftarrow 1, u \leftarrow 0, \text{converge} \leftarrow \text{FALSE}$
While not converge
$u \leftarrow u + 1$
$\bar{\pi}_{c+u} \leftarrow b_{c+u-1}\bar{\pi}_{c+u-1}$
$\Sigma \leftarrow \Sigma + \bar{\pi}_{c+u}$
If upper truncation criteria are met, then converge $\leftarrow$ TRUE
$l \leftarrow 0, \text{converge} \leftarrow \text{FALSE}$
While not converge
$l \leftarrow l + 1$
$\bar{\pi}_{c-l} \leftarrow a_{c-l+1}\bar{\pi}_{c-l+1}$
$\Sigma \leftarrow \Sigma + \bar{\pi}_{c-l}$
If lower truncation criteria are met, then converge $\leftarrow$ TRUE
Normalize: $\pi_n \leftarrow \bar{\pi}_n/\Sigma$ for $n = c - l, \ldots, c + u$
Return $\{\pi_{c-l}, \ldots, \pi_{c+u}\}$

Figure 3: Algorithm 3.

The remainder of this supplement shows Matlab code for the algorithms used in Sections 5-7 in the paper.

```
function [B,l,RE]=ErlangB(r,s,epsilon)
q=1;
B=1;
l=0;
converge=0;
while ~converge,
    l=l+1;
    q=(s-l+1)/r*q;
    Sum=1+q;
    B=B/Sum;
    q=q/Sum;
    if l>s-r/(1+q),
        RE=(s-l)*q/(B*(r-(s-l)*(1+q)));
        if RE<epsilon, converge=1; end
    end
end
end
```

```
function [C,l,u,RE]=ErlangCgeneral(lambda,mu,s,epsilon)
C=1;
l=0; u=0;
ql=1; qu=1;
converge=0;
b=lambda/(s*mu);
D1=1; D2=1;
E1=0; E2=1;
while ~converge,
    if (qu>ql)|l==s,
        u=u+1;
        qu=b*qu;
        C=C+qu;
        S=1+qu; qu=qu/S; ql=ql/S; C=C/S;
        D2=b*qu/(1-b); E2=D2;
    else
        l=l+1;
        a=(s-l+1)*mu/lambda;
        ql=a*ql;
        S=1+ql; qu=qu/S; ql=ql/S; C=C/S;
        if a<1, D1=a*ql/(1-a);end
        if l==s, D1=0; end
    end
    if (D1+D2<1),
        RE=(C*(D1+D2)+E1+E2)/(C*(1-D1-D2));
        if RE<epsilon,
            converge=1;
        end
    end
end
end
```

```
function [C,l,u,RE]=ErlangCgeometric(lambda,mu,s,epsilon)
l=0; u=0;
ql=1;
converge=0;
D1=1; D2=0;
E1=0; E2=0;
S=1+lambda/mu/(s-lambda/mu);
ql=ql/S;
C=1;
while ~converge,
    l=l+1;
    a=(s-l+1)*mu/lambda;
    ql=a*ql;
    S=1+ql; ql=ql/S; C=C/S;
    if a<1, D1=a*ql/(1-a);end
    if l==s, D1=0; end
    if (D1+D2<1),
        RE=(C*(D1+D2)+E1+E2)/(C*(1-D1-D2));
        if RE<epsilon,
            converge=1;
        end
    end
end
end
```

```
function [A,l,u,RE]=ErlangA(lambda,mu,gamma,s,t,epsilon)
l=0; u=0;
ql=1; qu=1;
phi=s*mu/gamma; xi=exp(-gamma*t); theta=-s*mu*t; kappa=1-xi;
A=exp(theta);
f=1; g=1;
converge=0;
D1=1; D2=1;
while ~converge,
    if (qu>ql)|l==s,
        u=u+1;
        qu=lambda/(s*mu+u*gamma)*qu;
        g=g*(phi+u-1)*kappa/u;f=f+g;theta=theta+log(f);
        g=g/f;f=1;
        A=A+qu*exp(theta);
        S=1+qu; qu=qu/S; ql=ql/S; A=A/S;
        if ((s*mu+u*gamma)>lambda),
            D2=lambda*qu/(s*mu+u*gamma-lambda);
        end
    else
        l=l+1;
        a=(s-l+1)*mu/lambda;
        ql=a*ql;
        S=1+ql; qu=qu/S; ql=ql/S; A=A/S;
        if l==s,
            D1=0;
        elseif ((s-l)*mu<lambda),
            D1=(s-l)*mu*ql/(lambda-(s-l)*mu);
        end
    end
    D=D1+D2;
    if D<1,
        RE=D*(1+A)/(A*(1-D));
        if RE<epsilon,
            converge=1;
        end
    end
end
end
```