How big is an outbreak likely to be? Methods for epidemic final size calculation **Electronic Supplementary Material**

Thomas House[1], Joshua V Ross[2], and David Sirl[3]

[1]Warwick Mathematics Institute, University of Warwick, Gibbet Hill Road, Coventry, CV4 7AL, UK. T.A.House@warwick.ac.uk
[2]Stochastic Modelling Group, School of Mathematical Sciences, The University of Adelaide, Adelaide SA 5005, Australia. joshua.ross@adelaide.edu.au
[3]Mathematics Education Centre, Loughborough University, Loughborough, LE11 3TU, UK. d.sirl@lboro.ac.uk

General notes on the code

This code is provided to give examples of the methods discussed in the main paper. The algorithms are intended as bare-bones implementations, without additional numerical optimisation or error handling. We ask that anyone making use of the code cites this paper, and the other work that we cite as appropriate.

ESM.1 Monte-Carlo methods

ESM.1.1 Gillespie's method

```
function [T,X,Y] = gil_mc(beta,gamma,X,Y)
Parameters = [beta gamma];
T = 0;
P(1,:) = [X Y];
old = [X Y];
loop = 1;
while (P(loop,2) > 0)
    [step,new] = Iterate(old,Parameters);
    loop = loop+1;
    T(loop) = T(loop-1)+step;
    P(loop,:) = old;
    loop = loop+1;
    T(loop) = T(loop-1);
    P(loop,:) = new;
    old=new;
    if (loop >= length(T))
        T(loop*2) = 0;
        P(loop*2,:) = 0;
    end
end
T = T(1:loop);
X = P(1:loop,1);
Y = P(1:loop,2);
end

function [step, new_value]=Iterate(old, Parameters)
beta = Parameters(1);
gamma = Parameters(2);
X = old(1);
Y = old(2);
Rate(1) = beta*X*Y;
Change(1,:) = [-1 +1];
Rate(2) = gamma*Y;
Change(2,:) = [0 -1];
R1 = rand;
R2 = rand;
step = -log(R2)/(sum(Rate));
m = find(cumsum(Rate) >= (R1*sum(Rate)), 1);
new_value = old + Change(m,:);
end
```

ESM.1.2 Sellke's method

ESM.1.2.1 `sel_mc.m`

```
function [Z Q Pr T T0] = sel_mc(s0,i0,beta,distname,varargin)
if strcmp(distname,'const')
    T0 = varargin{1}*ones(1,i0);
    T = varargin{1}*ones(1,s0);
else
    switch size(varargin,2)
        case 1
            T0 = random(distname,varargin{1},1,i0);
            T = random(distname,varargin{1},1,s0);
        case 2
            T0 = random(distname,varargin{1},varargin{2},1,i0);
            T = random(distname,varargin{1},varargin{2},1,s0);
        case 3
         T0 = random(distname,varargin{1},varargin{2},varargin{3},1,i0);
          T = random(distname,varargin{1},varargin{2},varargin{3},1,s0);
        otherwise
            error('Disallowed number of distribution parameters');
    end
end
Q = sort(exprnd(1,1,s0));
ST0 = sum(T0);
Pr = [ST0, (ST0 + cumsum(T(1:(s0-1))))]*beta;
Z = find(Q > Pr, 1);
if isempty(Z)
    Z = i0+s0;
else
    Z = Z+i0-1;
end
end
```

ESM.1.2.2 Temporal dynamics

Temporal dynamics can be obtained by appending the following code to the function `sel_mc` before the final `end` statement. The vectors `t`, `St` and `It` contain the times, number of susceptibles and number of infected individuals respectively.

```
TT = [T0 T];
QQ = [0*T0 Q];
R = T(1:i0)';
t = zeros(2*Z,1);
St = zeros(2*Z,1);
It = zeros(2*Z,1);
It(1:i0) = (1:i0)';
St(1:i0) = s0+1-(1:i0)';
tt = 0;
La = 0;
j = i0+1;
k = j;
while (It(k-1)>0)
    [minR,i] = min(R);
    dtprop = minR-tt;
    Laprop = La + beta*It(k-1)*dtprop;
    if (QQ(j) > Laprop)
        R = R([1:(i-1) (i+1):end]);
        tt = minR;
        t(k) = minR;
        It(k) = It(k-1)-1;
        St(k) = St(k-1);
        La = Laprop;
    else
        tt = tt + ((QQ(j)-La)/(Laprop-La))*dtprop;
        La = QQ(j);
        t(k) = tt;
        It(k) = It(k-1)+1;
        St(k) = St(k-1)-1;
        R = [R; (tt+TT(j))];
        j = j+1;
    end
    k = k+1;
end
```

ESM.1.3 Ludwig's method

```
function [Z Cvec] = lud_mc(s0,i0,beta,distname,varargin)
S = s0;
I = i0;
g = 0;
while I>0
    if strcmp(distname,'const')
        T = varargin{1}*ones(1,I);
    else
        switch size(varargin,2)
            case 1
                T = random(distname,varargin{1},1,I);
            case 2
                T = random(distname,varargin{1},varargin{2},1,I);
            case 3
           T = random(distname,varargin{1},varargin{2},varargin{3},1,I);
            otherwise
                error('Disallowed number of distribution parameters');
        end
    end
    g = g+1;
    Cvec(g) = 0;
    Inew = 0;
    for i=1:I
        p = 1 - exp(-beta*T(i));
        C = binornd(S,p);
        Cvec(g) = Cvec(g) + C;
        Inew = Inew + C;
        S = S - C;
    end
    I = Inew;
end
Z = s0 + i0 - S;
end
```

ESM.2 Machine-precision, Markov-chain methods

ESM.2.1 Brute force methods

Given the function `SIRS.m`, which generates `M = Q'`, from Ross et al. (2010), these methods start with code to generate this matrix, the index of the initial state, and the probability vector for the initial state:

```
[M,ss,ii] = SIRS(N,beta,1,0,0,[0 N],[0 N]);
Iind = find(ii==i0 & ss==s0);
v0 = 0*ii;
v0(Iind) = 1;
```

The `mexpv.m` function from EXPOKIT (Sidje, 1998) can then be used to generate a final probability vector, given a time `t_large` significantly larger than the mean hitting time:

```
pf = mexpv(t_large,M,v0);
```

Alternatively, the jump matrix P can be formed and taken to a sufficiently high power.

```
D=diag(M);
d=D;
d(d==0)=1;
p=(diag(D)-M)./(sparse(ones(length(ss),1))*d');
d(d<0)=0;
P=p+diag(d);
Pinf=P^((2*s0)+i0);
pf = Pinf(:,Iind);
```

ESM.2.2 Neuts and Li's algorithms

ESM.2.2.1 Bailey's method for final size

```
function P = bailey_fs(s0,i0,beta)
N=s0+i0;
a = zeros(N+1);
a(i0+1,s0+1) = 1;
for i=(i0-1):(-1):0
    if i==0
        a(i+1,s0+1) = a(i+2,s0+1)*((1)/(1+beta*s0));
    else
        a(i+1,s0+1) = a(i+2,s0+1)*((i)/(i+beta*i*s0));
    end
end
for k=1:s0
    a(i0+k+1,s0-k+1) = a(i0+k,s0-k+2)*((beta*(i0+k-1)*(s0-k+1))/...
        (i0+k-1 + beta*(i0+k-1)*(s0-k+1)));
    for i=(i0+k-1):(-1):2
      a(i+1,s0-k+1) = a(i+2,s0-k+1)*((i+1)/(i+1+beta*(i+1)*(s0-k))) +...
          a(i,s0-k+2)*((beta*(i-1)*(s0-k+1))/(i-1+beta*(i-1)*(s0-k+1)));
    end
    for i=1:(-1):0
        a(i+1,s0-k+1) = a(i+2,s0-k+1)*((i+1)/(i+1+beta*(i+1)*(s0-k)));
    end
end
P = a(1,(end-1):(-1):1)';
end
```

ESM.2.2.2 Peak height

This must be called for given `I0`, and returns the probability that the peak height is greater than or equal to this integer.

```
function P = NL_Imax(s0,i0,beta,I0)
P = 0;
N = s0+i0;
a = zeros(N+1);
a(i0+1,s0+1) = 1;
for i=(i0-1):(-1):0
    if i==0
        a(i+1,s0+1) = a(i+2,s0+1)*((1)/(1+beta*s0));
    else
        a(i+1,s0+1) = a(i+2,s0+1)*((i)/(i+beta*i*s0));
    end
end
for k=1:s0
    a(i0+k+1,s0-k+1) = a(i0+k,s0-k+2)*((beta*(i0+k-1)*(s0-k+1))/...
        (i0+k-1 + beta*(i0+k-1)*(s0-k+1)));
    for i=(i0+k-1):(-1):2
      a(i+1,s0-k+1) = a(i+2,s0-k+1)*((i+1)/(i+1+beta*(i+1)*(s0-k))) +...
          a(i,s0-k+2)*((beta*(i-1)*(s0-k+1))/(i-1+beta*(i-1)*(s0-k+1)));
        if i==I0
            P = P + a(i+1,s0-k+1);
            a(i+1,s0-k+1) = 0;
        end
    end
    i=1;
    a(i+1,s0-k+1) = a(i+2,s0-k+1)*((i+1)/(i+1+beta*(i+1)*(s0-k)));
    if i==I0
        P = P + a(i+1,s0-k+1);
        a(i+1,s0-k+1) = 0;
    end
end
end
```

ESM.2.3 Path sum / integral methods

ESM.2.3.1 Final size from path sum

Code to calculate this is available as detailed in Ross (2011).

ESM.2.3.2 Hitting times from path integral

This code creates a function handle to the Laplace transform of the hitting time distribution, `phi`, which can then be used in numerical Laplace transform inversion.

```
[M,ss,ii] = SIRS(N,beta,1,0,0,[0 N],[0 N]);
Cind = find(ii>0);
Iind = find(Cind==find(ii==1 & ss==(N-1)));
Q = full(M)';
Qc = Q(Cind,Cind);
f = ones(length(Cind),1);
F = diag(f);
qi = sum(Q(Cind,ii==0),2);
phi = @(s) get_lap(s,Qc,F,qi,Iind);

function l = get_lap(s,Qc,F,qi,Iind)
y = (Qc - s*F)\(-qi);
l = y(Iind);
end
```

ESM.2.4 Null space method

This requries the `spspaces.m` code from Kowal (2006), but once that is available then the calculation is straightforward:

```
[M,ss,ii] = SIRS(N,beta,1,0,0,[0 N],[0 N]);
Iind = find(ii==i0 & ss==s0);
Mn = spspaces(M,1);
M1 = Mn{1};
M3 = Mn{3};
P = M1(M3(N:-1:1),Iind);
```

ESM.3 Machine-precision, arbitrary infectious period methods

ESM.3.1 Direct substitution

The following code evaluates the Ball final-size equations for exponentially distributed recovery via direct substitution at 32-digit precision, and is readily adapted to machine precision or different infectious period distributions.

```
digits(32)
P = vpa(zeros(s0+1,1));
syms k l la m n ss pp
for L=0:s0
    display(['Row ' num2str(L) ' of ' num2str(s0) ' done.']);
    s = vpa('0');
    for K=0:(L-1)
        s = vpa(...
            subs('ss + pp*(((l!)/(k!*(l-k)!)) /...
                (((n!)/(k!*(n-k)!))*( (1/(1+(la*(n-l))))^(k+m) )))',...
                {k,l,la,m,n,ss,pp},{K,L,beta,i0,s0,s,P(K+1)}));
    end
    P(L+1) = vpa(...
        subs('(1.0 - ss)*(((n!)/(k!*(n-k)!))*((...
            (1/(1+(la*(n-l))))^(k+m) )))',...
            {k,l,la,m,n,ss},{L,L,beta,i0,s0,s}));
end
```

ESM.3.2 Matrix-based methods

The simplest matrix-based approach is

```
phi = @(x) 1/(1+x);
B = ball_matrix(s0,i0,beta,phi);
ii = ones(s0+1,1);
P = B\ii;

function B = ball_matrix(s0,i0,beta,phi)
B = zeros(s0+1);
for l=0:s0
    for k=0:l
        B(l+1,k+1) = (nchoosek(l,k) /...
            (nchoosek(s0,k)*((phi(beta*(s0-l)))^(k+i0))));
    end
end
end
```

Given an initial approximation (typically from asymptotic results) `x0`, a tolerance `tol` and maximum number of iterations `iter`, the PCG method is

```
A = (B'*B);
b = (B'*ii);
M1 = diag(diag(A));
P = pcg(A,b,tol,iter,M1,[],x0);
```

We set `tol=eps; iter=2e4;` for the most demanding benchmark (II). This benchmark also requires a good initial guess, `x0`, which we constructed as below.

```
q = zeros(s0(j)+1,1);
L=1;
q(L) = 1/B(1,1);
while (q(L) > 0)
    L = L + 1;
    q(L) = (1-B(L,1:(L-1))*q(1:(L-1)))/B(L,L);
    if q(L) > q(L+1)
        q(L)=-1;
    end
end
q(L) = 0;
Q = sum(q(1:(L-1)));
ff = @(z) (z - 1 + exp(-R0(j)*z));
tau = fzero(ff,[eps 1-eps]);
rho = 1 - tau;
vv = N(j)*(rho*(1-rho) + (R0(j)^2)*tau*(rho^2))/((1-(R0(j)*rho))^2);
y = normpdf(1:(s0(j)+1),s0(j)*tau,sqrt(vv))';
x0 = q + (y*(1-Q)/sum(y));
```

ESM.3.3 Polynomial methods

Here, the main function calculates the final size distribution using polynomial methods.

```
function P = gont_fs(s0,i0,beta,Phi)
U = zeros(1,s0+i0+1);
for i=0:s0+i0
    U(i+1) = Phi(i*beta);
end
p = zeros(1,s0+1);
for j=0:s0
    x = gontFact(s0-j,0,U(j+1:end));
    S = 0;
    for k=0:(s0-j)
        pp = U(k+j+1)^i0;
        num = s0:-1:(j+1);
        for i=1:(s0-j-k)
            num(i) = num(i)*U(k+j+1);
        end
        den = [s0-j-k:-1:1, k:-1:1];
        for i=1:(s0-j)
            pp = pp*(num(i)/den(i));
        end
        S = S+pp*x(k+1);
    end
    p(j+1)=S;
end
P=p(end:-1:1);
end
```

It calls the sub-function below, which returns the first $k+1$ Gontcharoff polynomials scaled by a factorial, so the $(k+1)$-th term is $k!G_k(x|U)$. The optional 4th argument can be the first part of the sequence to avoid recalculation.

```
function y = gontFact(k,x,U,varargin)
y = zeros(1,k+1);
if isempty(varargin)
    yy = [];
    startIndex = 1;
    y(1) = 1;
else
    yy = varargin{1};
    startIndex = length(yy);
    y(1:startIndex) = yy;
end
for i=startIndex:k
    y(i+1) = x^i;
    for j=0:(i-1)
        prod = 1;
        for k=0:(i-j-1)
            prod = prod*(U(j+1)*((i-k)/(i-j-k)));
        end
        y(i+1) = y(i+1) - prod*y(j+1);
    end
end
end
```

ESM.4 Asymptotic results

The following code reproduces Figure 2 of Ball et al. (1997), using a jump matrix approach.

```
N=5;
laGrange = [
    0.7 1 1.5 2;
    0.54 1 1.5 2;
    0.34 0.4 0.5 1;
    0.25 0.35 0.5 1
    ];
laLrange = [0.125 0.1875 0.5 1.25];
pimax = 1-1e-6;
mm = 1e2;
for i=1:4
    tau = laLrange(i);
    for j=1:4
        beta = laGrange(i,j);
        gamma = 1;
        [M,ss,ii] = SIRS(N,tau,gamma,0,0,[0 N],[0 N]);
        rr=(N-ss-ii);
        D=diag(M);
        d=D;
        d(d==0)=1;
        p=(diag(D)-M)./(sparse(ones(length(ss),1))*d');
        d(d<0)=0;
        P=p+diag(d);
        G=P^(2*N);

        z = @(pp) G*((rr==0).*binopdf(ss,N,pp));
        g = @(pp) (exp(-(beta/(gamma))*(rr'/N)*z(pp)));
        pi=pimax;
        for m=1:mm
            pi = g(pi);
        end
        q = z(pi);
        [~, jj] = sort(rr(ii==0));
        Q = q(ii==0);

        subplot(4,4,j+4*(i-1))
        bar(0:N,Q(jj))
    end
end
```

ESM.5 Epidemics on networks

ESM.5.1 Monte Carlo methods

For simplicity, these methods are implemented for exponential recovery only, but can clearly be extended as in the function above to arbitrary distributions. The network is size N with adjacency matrix G.

ESM.5.1.1 sel_net.m

```
function R = sel_net(n,m,tau,gamma,G)
N = length(n);
T = random('exp',gamma,1,N);
R = zeros(1,N);
Q = exprnd(1,1,N);
Inew = m;
while ~isempty(find(Inew>0, 1))
    R = R + Inew;
    La = tau*(T.*R)*G;
    Inew = (La>Q).*(1-R);
end
end
```

ESM.5.1.2 lud_net.m

```
function R = lud_net(N,G,tau,gamma,i0)
S = ones(N,1);
I = zeros(N,1);
J = zeros(N,1);
R = zeros(N,1);
T = random('exp',gamma,N,1);
S(i0) = 0;
I(i0) = 1;
i = sum(I);
while(i>0)
    ii = find(I);
    i = sum(I);
    for j=1:i
        p = 1 - exp(-tau*T(ii(j)));
        s = find(G(:,ii(j))&S);
        ss = (rand(length(s),1)<p);
        S(s(ss)) = 0;
        J(s(ss)) = 1;
        I(ii(j)) = 0;
        R(ii(j)) = 1;
    end
    I = J;
    J = 0*J;
end
end
```

ESM.5.2 Machine precision methods

ESM.5.2.1 Neal's equations

This code generates a matrix N that can then be used to calculate final size probabilities for on a Bernoulli graph with link probabilities p.

```
function N = neal_matrix(s0,i0,beta,phi,p)
q = zeros(s0+1,1);
for l=0:s0
    for k=0:l
       q(l+1) = q(l+1) + nchoosek(l,k)*(p^k)*phi(beta*k)*((1-p)^(l-k));
    end
end
N = zeros(s0+1);
for l=0:s0
    for k=0:l
     N(l+1,k+1) = (nchoosek(l,k)/(nchoosek(s0,k)*((q(s0-l+1))^(k+i0))));
    end
end
end
```

ESM.5.2.2 Ball multitype formula

```
function [B x] = multitype_matrix(n,m,beta,phi)
kk = length(n);
ll = prod(n+1);
B = zeros(ll);
x = zeros(ll,kk);
C = [1 cumprod(n(1:(end-1))+1)];
v = 0*n;
for l=1:ll
    for i=kk:-1:2
        if (v(i) > n(i))
            v(i) = 0;
            v(i-1) = v(i-1) + 1;
        end
    end
    j = sum(C.*v)+1;
    x(j,1:kk) = v;
    u = 0*n;
    for w=1:prod(v+1)
        for i=kk:-1:2
            if (u(i) > v(i))
                u(i) = 0;
                u(i-1) = u(i-1) + 1;
            end
        end
        k = sum(C.*u)+1;
        B(j,k) = multi_nchoosek(n-u,v-u)/...
            (prod((phi((beta*(n-v)'))).^(u+m))*multi_nchoosek(n,v));
        u(kk) = u(kk) + 1;
    end
    v(kk) = v(kk) + 1;
end
end

function c = multi_nchoosek(a,b)
c = 1;
for i = 1:length(a)
    c = c*nchoosek(a(i),b(i));
end
end
```

ESM.5.2.3 Markovian network model

```
function [Q x] = Q_net_SIR(tau,gamma,G)
% 1 is S, 2 is I, 3 is R
[N, ~] = size(G);
c = [1 cumprod(3*ones(1,N-1))];
C = c(end:-1:1);
M = 3^N;
Q = sparse(M,M);
x = zeros(M,N);
v = ones(1,N);
for i=1:M
    x(i,:) = v;
    ss = find(v==1);
    ii = find(v==2);
    for k=ii
        Q(i,i) = Q(i,i) - gamma;
        u=v;
        u(k)=3;
        j = sum(C.*(u-1))+1;
        Q(i,j) = Q(i,j) + gamma;
    end
    for k=ss
        lambk = sum(tau*G(ii,k));
        Q(i,i) = Q(i,i) - lambk;
        u=v;
        u(k)=2;
        j = sum(C.*(u-1))+1;
        Q(i,j) = Q(i,j) + lambk;
    end

    v(N) = v(N)+1;
    for k=N:-1:2
        if (v(k) > 3)
            v(k) = 1;
            v(k-1) = v(k-1) + 1;
        end
    end
end
end
```

References

F. Ball, D. Mollison, and G. Scalia-Tomba. Epidemics with two levels of mixing. *The Annals of Applied Probability*, 7(1):46--89, 1997.

P. Kowal. Null space of a sparse matrix. MATLAB File Exchange: http://www.mathworks.fr/matlabcentral/fileexchange/11120-null-space-of-a-sparse-matrix, 2006.

J. V. Ross. Invasion of infectious diseases in finite homogeneous populations. *Journal of Theoretical Biology*, 289:83--89, 2011. Code available at http://www.maths.adelaide.edu.au/joshua.ross/Research/Code.html.

J. V. Ross, T. House, and M. J. Keeling. Calculation of disease dynamics in a population of households. *PLoS ONE*, 5:e9666, Jan 2010.

R. B. Sidje. Expokit. A software package for computing matrix exponentials. *ACM Transactions on Mathematical Software*, 24(1):130--156, 1998.