Subcritical Measles Outbreak Size

```
# requires: $(Box_Models)/boxmodel.py $(Box_Models)/boxmodelproduct.py
# requires: $(SageDynamics)/dynamicalsystems.py $(SageUtils)/latex_output.py
# produces: measles-model.sobj
from sage.all import *
import os, sys
sys.path.append( os.environ['SageDynamics'] )
#sys.path.append( os.environ['SageUtils'] )
sys.path.append( os.environ['Box_Models'] )
import boxmodel, boxmodelproduct

S, I, R, beta, mu, u, v = SR.var( 'S I R beta mu u v' )
SIR = boxmodel.BoxModel(
    DiGraph( [ (S, I, beta*S*I), (I, R, mu*I) ] ),
    [S,I,R]
)
uv = boxmodel.BoxModel(
    DiGraph( { v:(), u:() } ),
    [v,u]
)

measles_homogeneous_general = SIR
measles_heterogeneous_general = boxmodelproduct.BoxModelProduct( SIR, uv )

save_session( 'measles-model' )
```

```
# requires: $(Box_Models)/boxmodel.py $(Box_Models)/boxmarkov.py
# requires: $(SageDynamics)/dynamicalsystems.py $(SageUtils)/latex_output.py
# requires: measles-model.sobj
# produces: measles-run.tex measles-run.sobj
from sage.all import *
import os, sys
sys.path.append( os.environ['SageDynamics'] )
sys.path.append( os.environ['SageUtils'] )
sys.path.append( os.environ['Box_Models'] )
import latex_output, dynamicalsystems, boxmodel, boxmarkov

load_session( 'measles-model' )

ltx = latex_output.latex_output( 'measles-run.tex' )

  # homogeneous assumptions
```

```
  #  p_v is proportion of population vaccinated
  #  everyone has same rate of contacts per time \alpha
  #  \sigma is probability a contact is adequate
p_v, alpha, sigma = SR.var( 'p_v alpha sigma' )

homogeneous_params = dynamicalsystems.Bindings(
  beta = alpha * sigma
)

measles_homogeneous = measles_homogeneous_general.bind( homogeneous_params )

  # heterogeneous assumptions:
  #  two classes: u = unvaccinated, v = vaccinated
  #  \sigma_x is probability a class x person is infected by contact
  #  alpha_x_y is rate of x-y contacts
sigma_u, sigma_v, alpha_u_u, alpha_v_v = SR.var( 'sigma_u sigma_v alpha_u_u alpha_v_v' )

#print measles_heterogeneous_general.ode(), '\n'

heterogeneous_params = homogeneous_params + dynamicalsystems.Bindings(
  # \alpha_x_y is a derived quantity, the number of x-y contacts per time
  # Since
  #    \alpha_u_v = \alpha_v_u
  #    \alpha = sum( N_x N_y \alpha_x_y ) / sum( N_x N_y )
  #      = (1-p_v)^2 \alpha_u_u + 2p_v(1-p_v) \alpha_u_v + p_v^2 \alpha_v_v
  # from these things we can end up with
  beta_u_u = alpha_u_u * sigma_u,
  beta_v_u = SR('alpha_v_u') * sigma_v,
  beta_u_v = SR('alpha_u_v') * sigma_u,
  beta_v_v = alpha_v_v * sigma_v,
  # mu is recovery rate
  mu_u = mu,
  mu_v = mu
) + dynamicalsystems.Bindings(
  alpha_v_u = (alpha - (1-p_v)^2 * alpha_u_u - p_v^2 * alpha_v_v)/(2*p_v*(1-p_v)),
) + dynamicalsystems.Bindings(
  alpha_u_v = SR('alpha_v_u'),
  # also, since \sigma = p_v \sigma_v + (1-p_v) \sigma_u,
  sigma_v = (sigma - (1 - p_v) * sigma_u) / p_v
)

measles_heterogeneous = measles_heterogeneous_general.bind( heterogeneous_params )

if False:
    ## homogeneous model should be equal to heterogeneous model when
    ## \alpha_u_u = \alpha_p_p = \alpha
    ## and \sigma_u = \sigma_v = \sigma
    het_check = measles_heterogeneous.bind( dynamicalsystems.Bindings(
      alpha_u_u = alpha,
      alpha_v_v = alpha,
      sigma_u = sigma,
      sigma_v = sigma
    ) )
    hco = het_check.ode()
    print 'heterogeneous check:\n', hco, '\n'
    hom_check = dynamicalsystems.ODESystem( {
      S : simplify( hco._flow[SR.symbol('S_u')] + hco._flow[SR.symbol('S_v')] ),
```

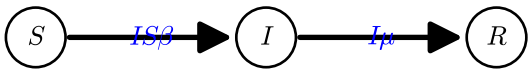
```
      I : simplify( hco._flow[SR.symbol('I_u')] + hco._flow[SR.symbol('I_v')] ),
      R : simplify( hco._flow[SR.symbol('R_u')] + hco._flow[SR.symbol('R_v')] )
    }, [S,I,R] )
    print hom_check, '\n'

#print measles_heterogeneous.ode(), '\n'

def print_dist( states, dist, sb ):
    st = '\n'
    for s,p in zip(states,dist):
        if p > 0:
            st += 'P(' + str(sb(s)) + ') = ' + str(p) + '\n'
    return st

def calc_final_size_distribution( model, N, init_cond, bindings=dynamicalsystems.Bindings() ):
    #print model.ode(), '+', bindings
    model = model.bind( bindings )
    print 'p_v =', model._bindings( p_v )
    print 'sigma_v =', model._bindings( sigma_v )
    print 'alpha_u_v =', model._bindings( 'alpha_u_v' )
    print model.ode()
    M = model.embedded_discrete_markov_matrix( N, RDF )
    #print '||M|| =', max( i for j in M for i in j )
    states = model.stochastic_states( N )
    sb = model.stochastic_state_binding_function()
    state_indexes = { s:i for i,s in enumerate(states) }
    print 'N =', N, ',', len(states), 'states'
    initial_dist_b = bindings( init_cond )
    initial_state = vector( ( initial_dist_b(v) for v in model._vars ) )
    #print initial_state
    initial_dist = vector( ( 1 if s == initial_state else 0 for s in states ) )
    #print initial_dist
    #print 'p_0:', print_dist( states, initial_dist, sb )
    if initial_dist.norm() == 0: raise ValueError, 'bad initial state'
    #print 'p_1:', print_dist( states, M*initial_dist, sb )
    #print model._bindings
    if False:
        M2N = M^(2*N)
        #print '||M^2N|| =', max( i for j in M2N for i in j )
        #print M2N.str( zero='.' )
        final_dist = M2N * initial_dist
    elif False: # see if this scales better
        its = M.iterates( initial_dist, 2*N, rows=False )
        final_dist = its.column(-1)
    else: # or how about with numpy
        import scipy.linalg, numpy
        m = numpy.array(M)
        final_dist = numpy.linalg.matrix_power( m, int(2*N) ).dot( numpy.array( initial_dist ) )
    #print 'p_2N:', print_dist( states, final_dist, sb ); sys.stdout.flush()
    pR = {}
    Rx = model._bindings('R')
    for s,i in state_indexes.iteritems():
        if final_dist[i] > 0:
            Rb = sb(s)(Rx)
            pR[Rb] = pR.get( Rb, 0 ) + final_dist[i]
    pRpts = [ (k,pR[k]) for k in sorted(pR.keys()) ]
    #print pRpts
```

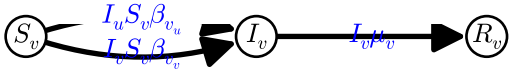
```
    print
    sys.stdout.flush()
    return pRpts

hom_parameters = dynamicalsystems.Bindings(
  sigma = 0.5,
  alpha = 1,
  mu = 0.7
)

def calc_hom(N):
    # parameters for outbreak-size experiment
    init_I = 1/N  # initial number infected
    init_S = N - init_I
    init_cond = dynamicalsystems.Bindings(
        S = 1 - init_I,
        I = init_I,
        R = 0
    )
    return calc_final_size_distribution(
        measles_homogeneous,
        N,
        init_cond,
        hom_parameters
    )

hom_runs = {
    N:calc_hom(N)
    for N in (3,5,7,9,11)
}

het_parameters = hom_parameters + dynamicalsystems.Bindings(
  alpha_u_u = 2,
  alpha_v_v = 1.1,
  sigma_u = 1
)

def calc_het(N):
    # parameters for outbreak-size experiment
    init_I_v = 0    # initial number infected among vaccinated
    init_I_u = 1/N  # initial number infected among unvaccinated
    init_S = N - init_I_v + init_I_u
    p_v = floor( 9/10 * N ) / N
    init_cond = dynamicalsystems.Bindings(
        I_u = init_I_u,
        I_v = init_I_v,
        S_u = 1 - p_v - init_I_u,
        S_v = p_v - init_I_v,
        R_u = 0,
        R_v = 0
    )
    return calc_final_size_distribution(
        measles_heterogeneous,
        N,
        init_cond,
        het_parameters.merge( p_v = p_v )
    )
```

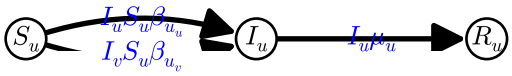
```
het_runs = {
    N:calc_het(N)
    for N in (3,5,7,9,11)
}

ltx.close()

save_session( 'measles-run' )
```

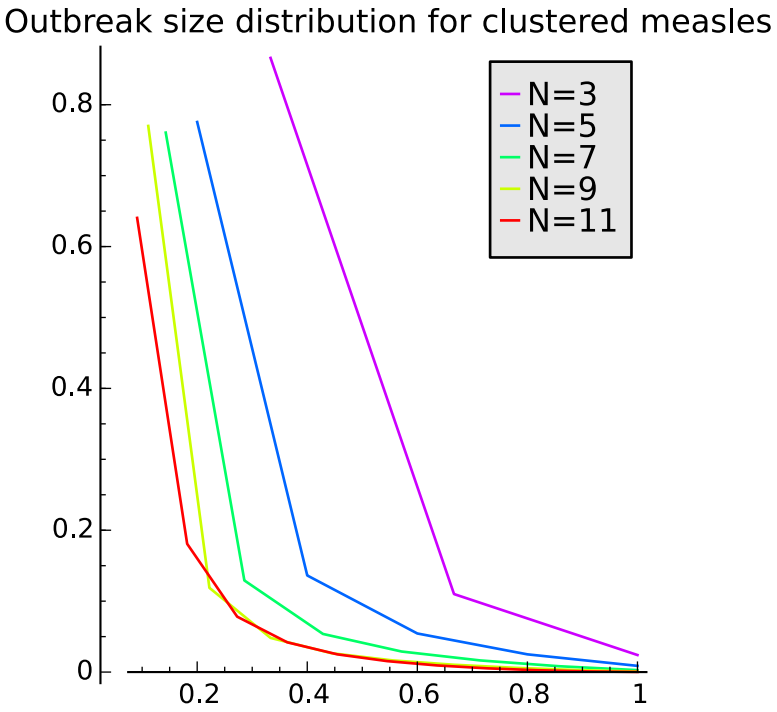
```
# requires: $(Box_Models)/boxmodel.py $(Box_Models)/boxmodelproduct.py
# requires: $(SageDynamics)/dynamicalsystems.py $(SageUtils)/latex_output.py
# requires: measles-run.sobj
# produces: measles-output.tex measles-het.svg measles-hom.svg measles-p.svg
from sage.all import *
import os, sys
sys.path.append( os.environ['SageDynamics'] )
sys.path.append( os.environ['SageUtils'] )
sys.path.append( os.environ['Box_Models'] )
import latex_output, boxmodel, boxmodelproduct

load_session( 'measles-run' )

measles_homogeneous_general.plot().save( filename='measles-hom.svg', figsize=(3,3) )

measles_heterogeneous_general.plot().save( filename='measles-het.svg', figsize=(3,3) )

ltx = latex_output.latex_output( 'measles-output.tex' )
#ltx.write_equality( SR.symbol('M'), M )
#ltx.write_equality( SR( 'M^N' ), MN )
#ltx.write_equality( SR( 'M^N p_0' ), final_dist )
#ltx.write_equality( SR( 'n(M^N)' ), MNb )
#ltx.write_equality( SR( 'n(M^N p_0)' ), final_b )

ltx.close()

colors = rainbow( len(hom_runs) )
G = Graphics()
for N in sorted(hom_runs.keys()):
    G += list_plot( hom_runs[N], plotjoined=True, color=colors.pop(), legend_label='N='+str(N)
G.save(
  filename='measles-hom-p.svg',
  #ymax=0.01,
  title='Outbreak size distribution for homogeneous measles',
  figsize=(4,4)
)

colors = rainbow( len(het_runs) )
G = Graphics()
for N in sorted(het_runs.keys()):
    G += list_plot( het_runs[N], plotjoined=True, color=colors.pop(), legend_label='N='+str(N)
G.save(
  filename='measles-het-p.svg',
  #ymax=0.01,
  title='Outbreak size distribution for clustered measles',
```

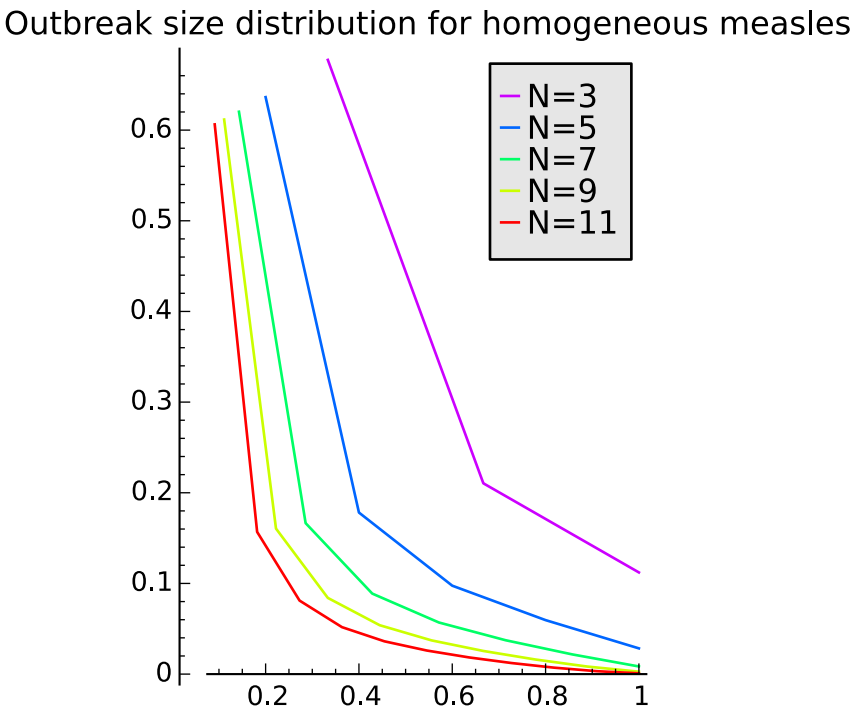
```
  figsize=(4,4)
)
```


S
$IS\beta$
I
$I\mu$
R


S_v
$I_u S_v \beta_{v_u}$
$I_v S_v \beta_{v_v}$
I_v
$I_v \mu_v$
R_v


S_u
$I_u S_u \beta_{u_u}$
$I_v S_u \beta_{u_v}$
I_u
$I_u \mu_u$
R_u


Outbreak size distribution for clustered measles
N=3
N=5
N=7
N=9
N=11
0.8
0.6
0.4
0.2
0
0.2
0.4
0.6
0.8
1


Outbreak size distribution for homogeneous measles
N=3
N=5
N=7
N=9
N=11
0
0.1
0.2
0.3
0.4
0.5
0.6
0.2
0.4
0.6
0.8
1