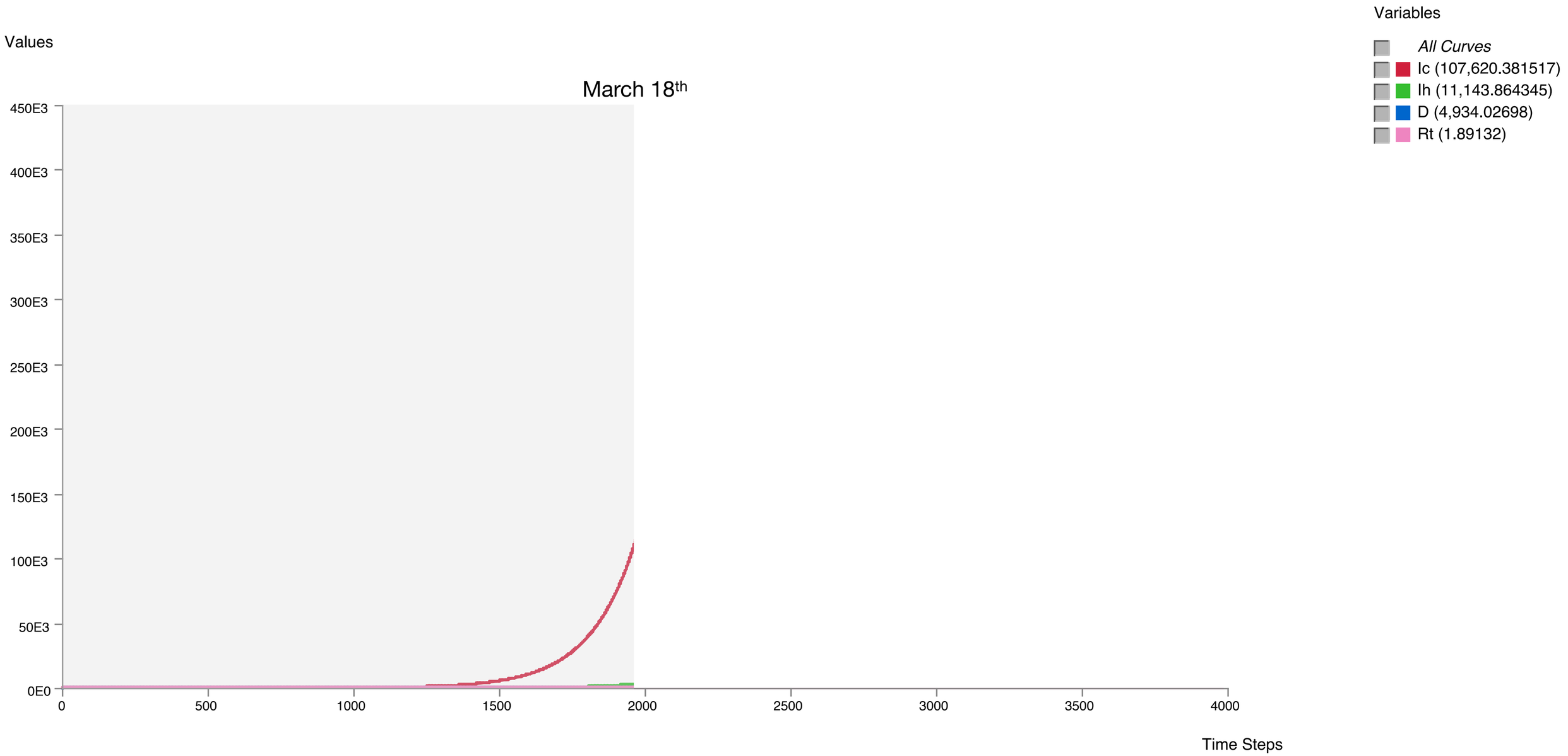
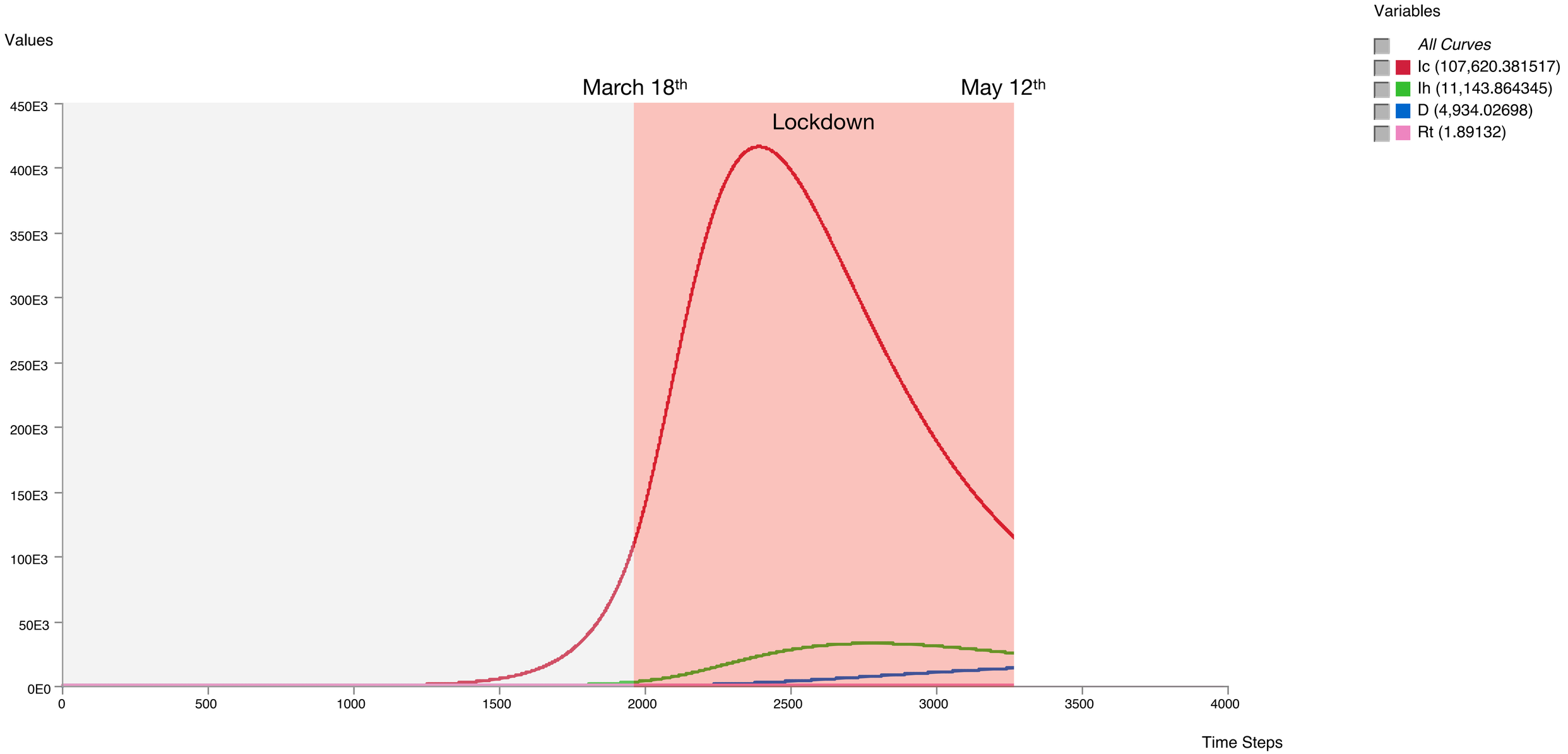
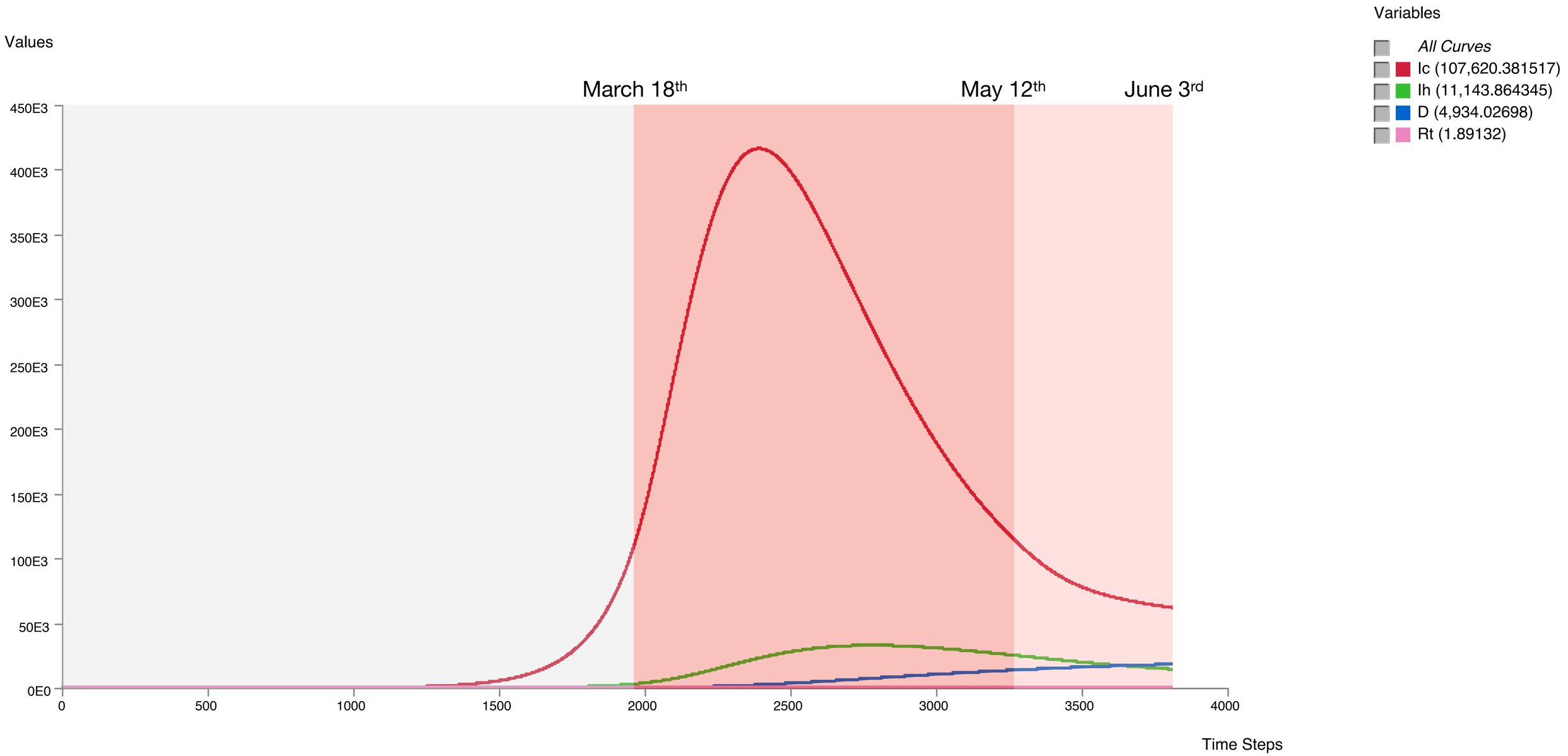
COVID-19 in France



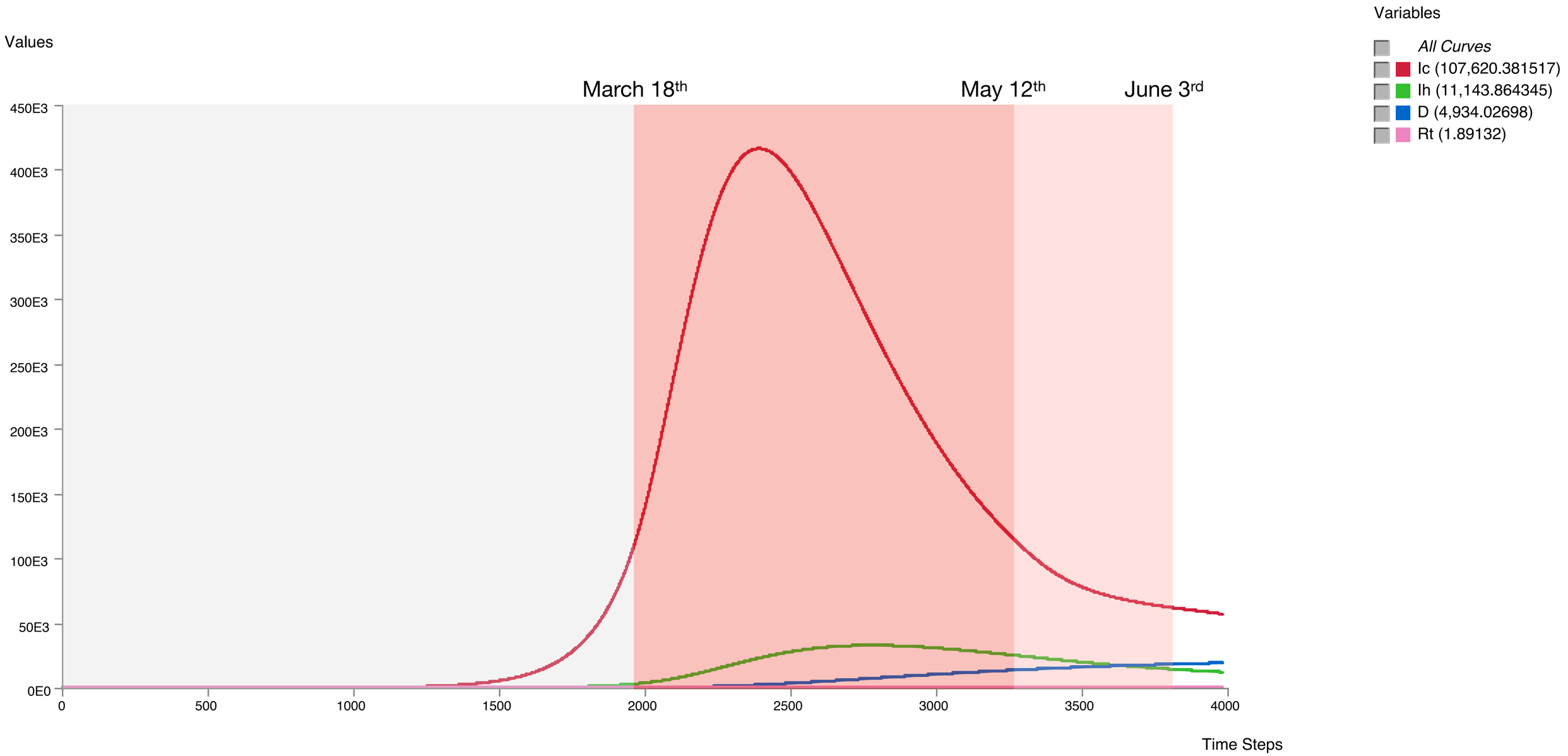
COVID-19 in France



COVID-19 in France

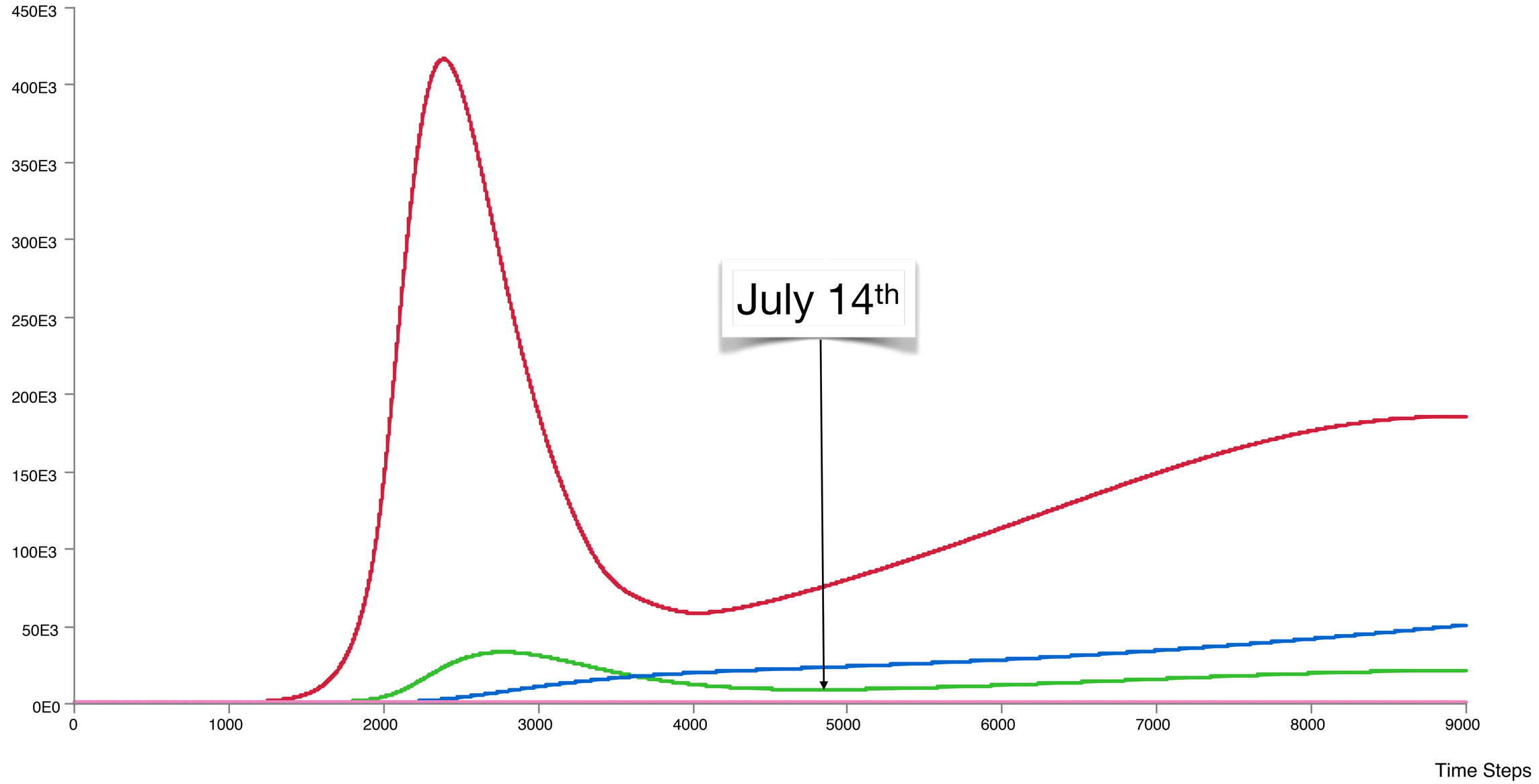


COVID-19 in France



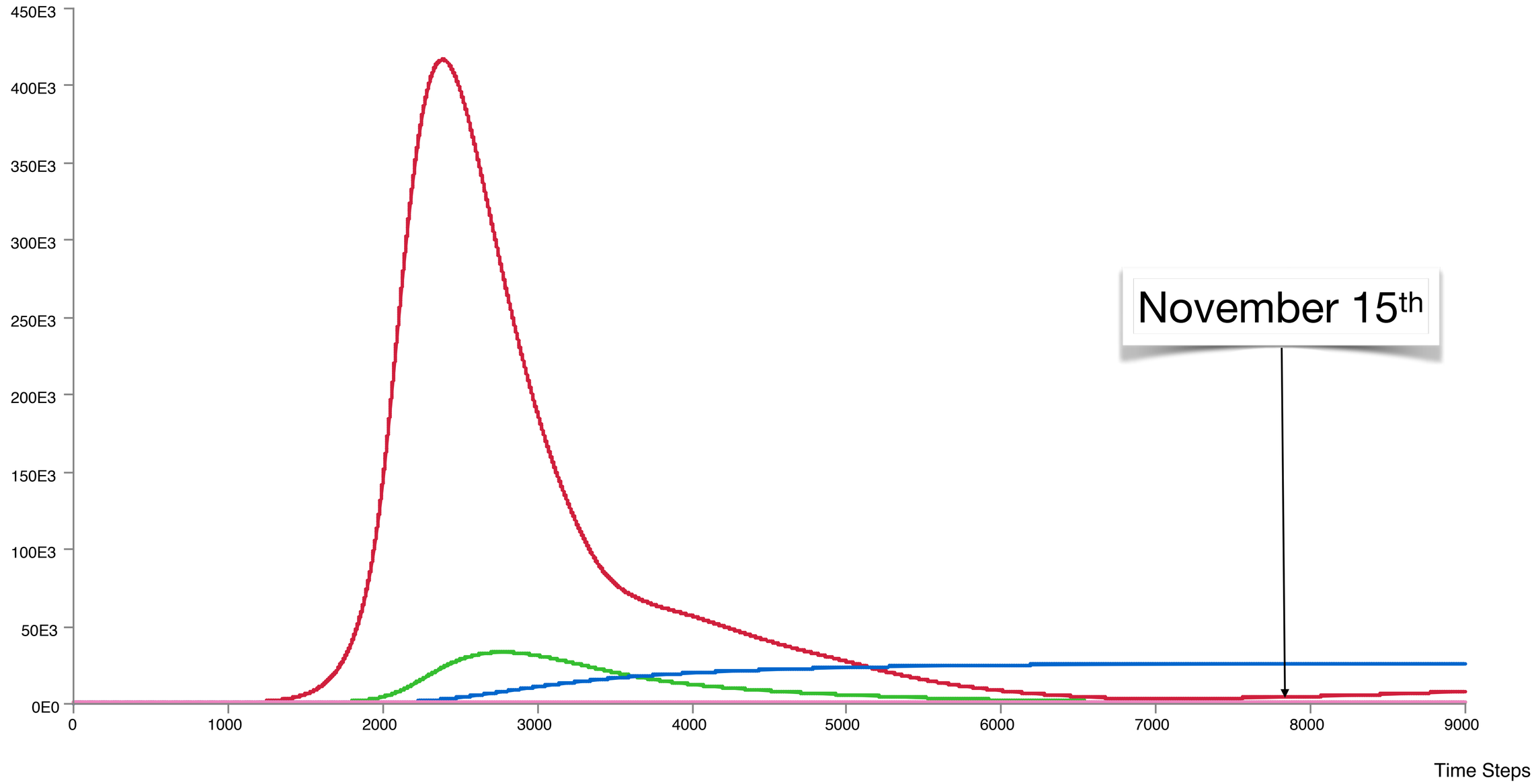
Scenario 1

3816	?Rsm?:mb{1.156} //Barriers



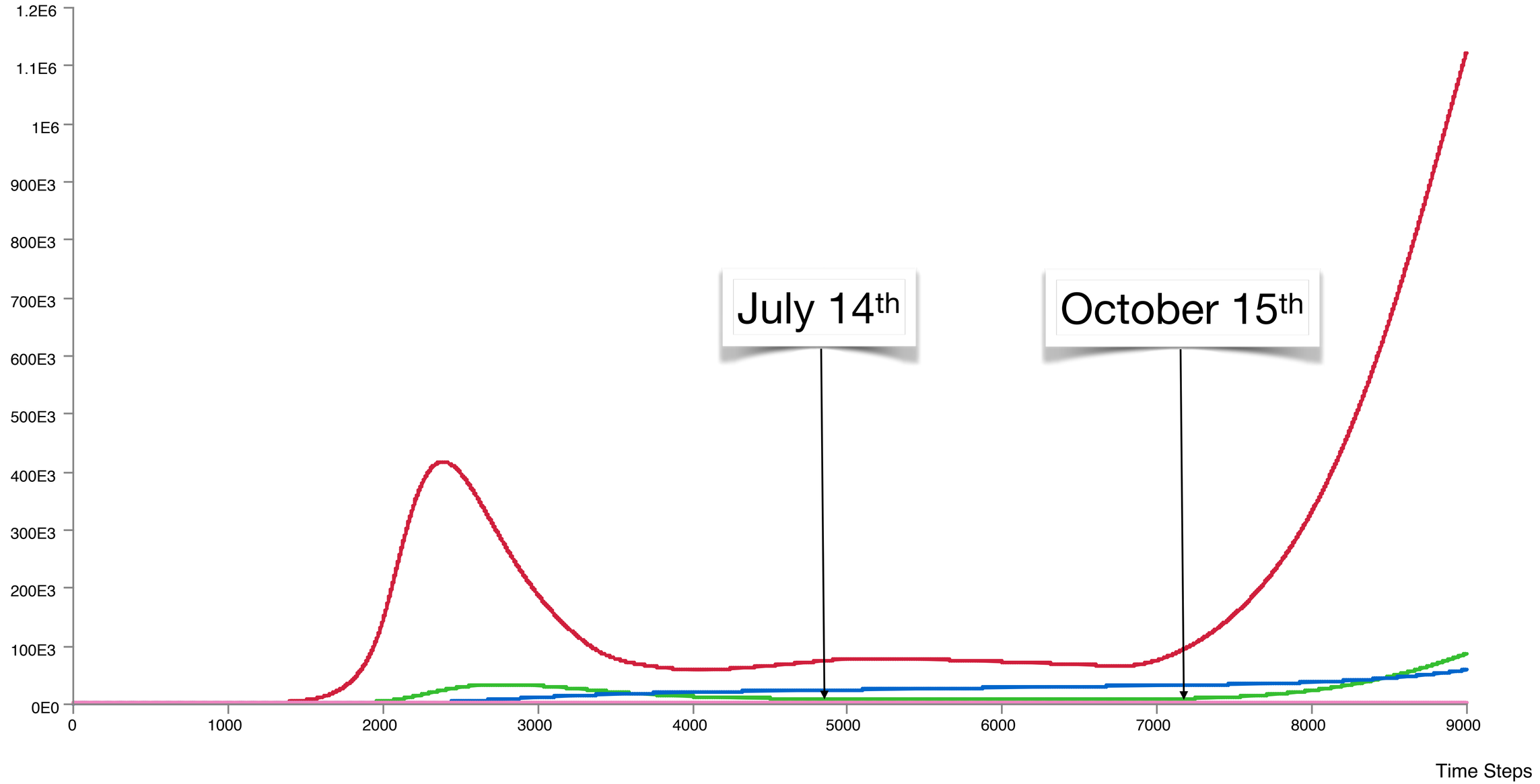
Scenario 2

3816	?Rsm?:mb{0.9} //Summer
4800	?Rsm?:mb{0.8} //Summer2
6696	?Rsm?:mb{1.2} //Automn1



Scenario 3

3816	?Rsm?:mb{1.15} //Summer
4800	?Rsm?:mb{1.05} //Summer2
6696	?Rsm?:mb{1.5} //Automn1



Agenda

Markov Chain vs DBN

Differential Equations vs DBN

Function Nodes and Constants

Python vs DBN

S2E3IRD for France

Memory Nodes

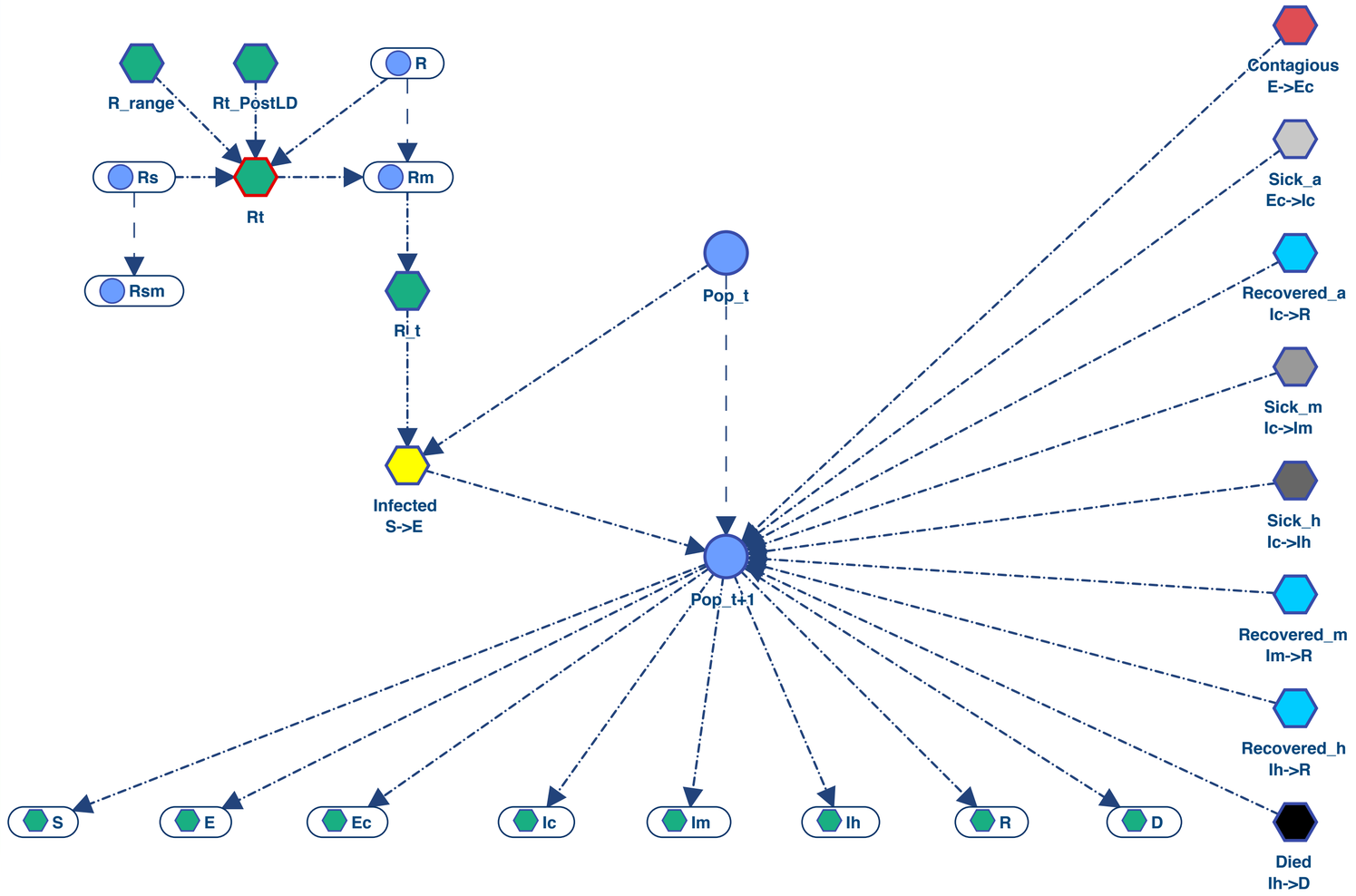
Optimization

Within BayesiaLab

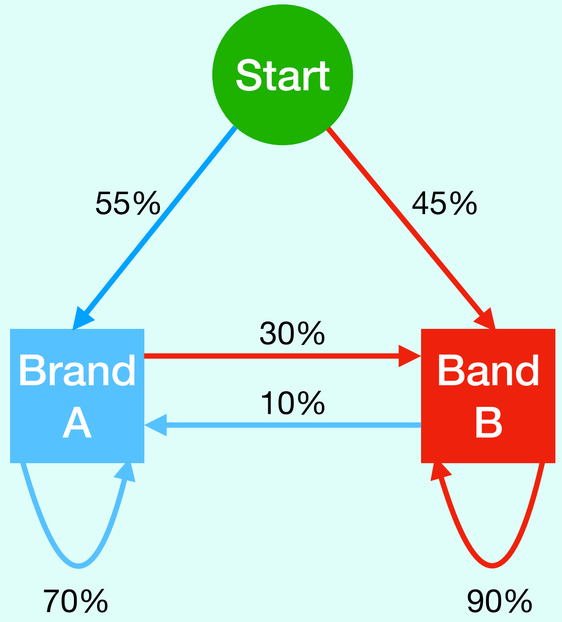
With the API

Simulation

Scenarios

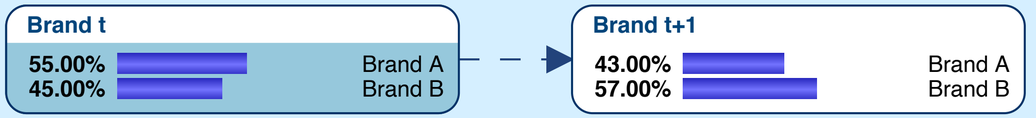


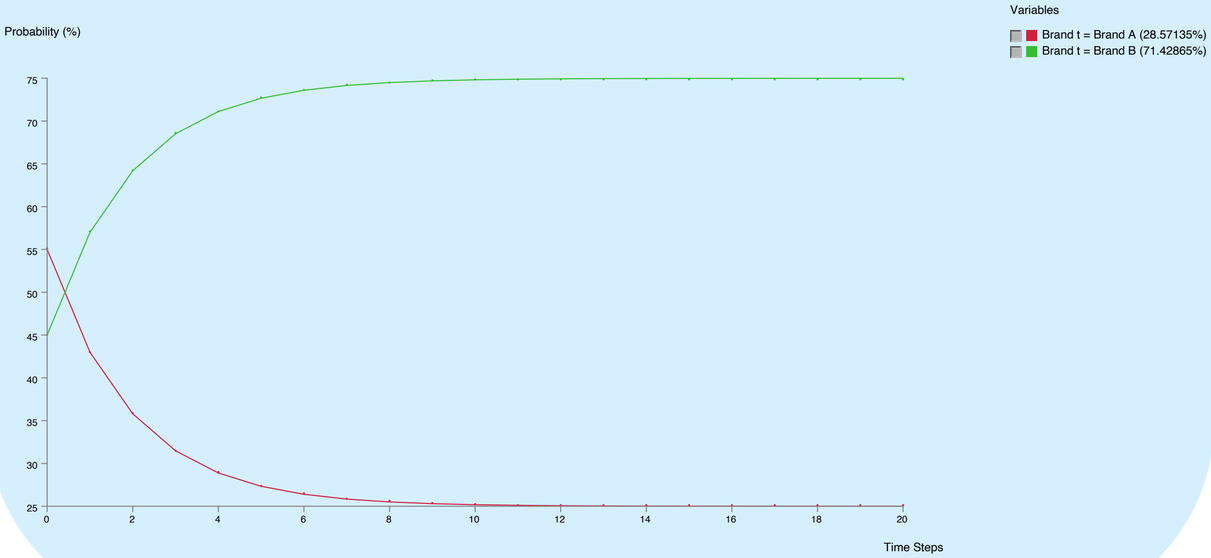
Markov Chain



Dynamic Bayesian Network

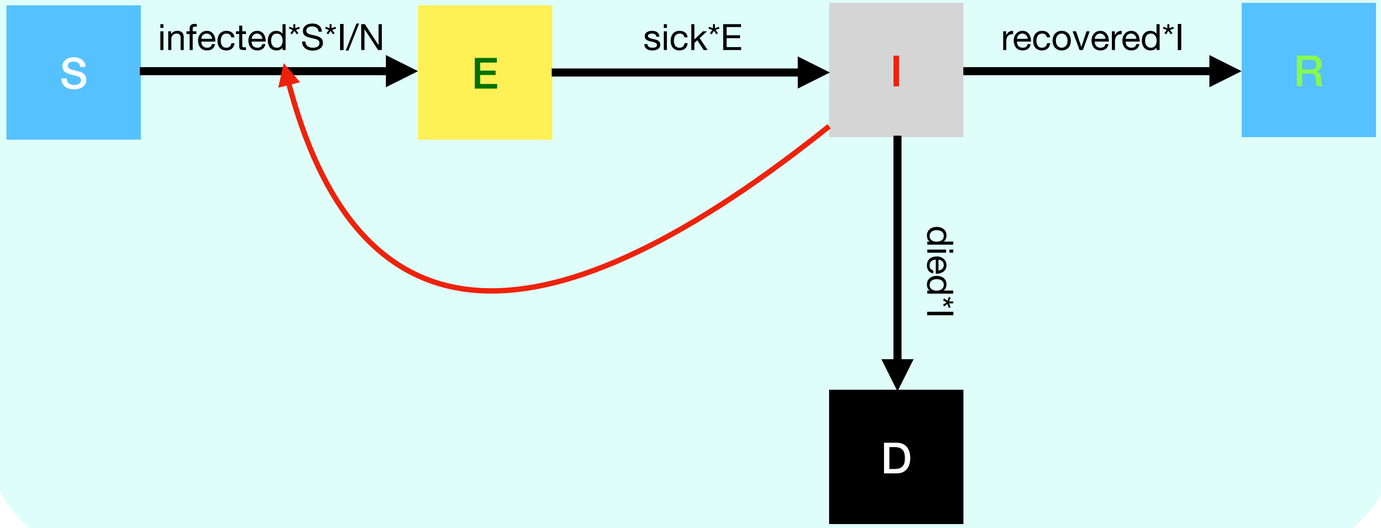
Brand t	Brand A	Brand B
Brand A	70.000	30.000
Brand B	10.000	90.000



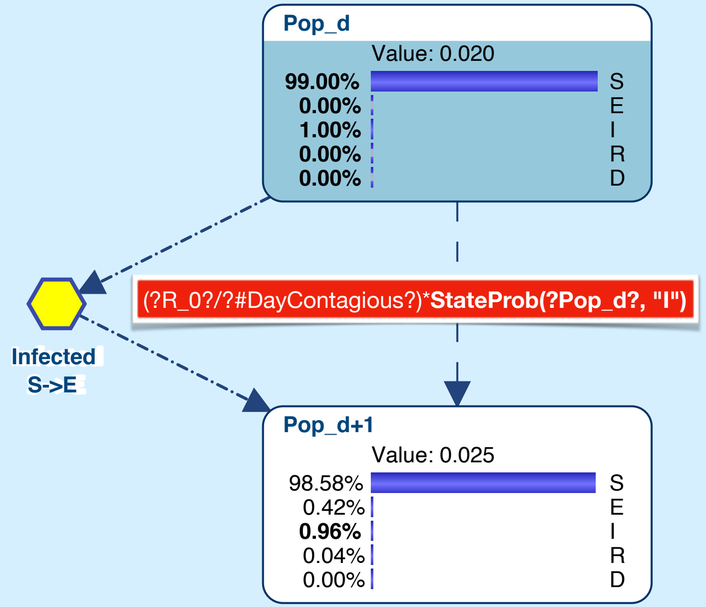


Compartmental Epidemiological Model

Differential Equations



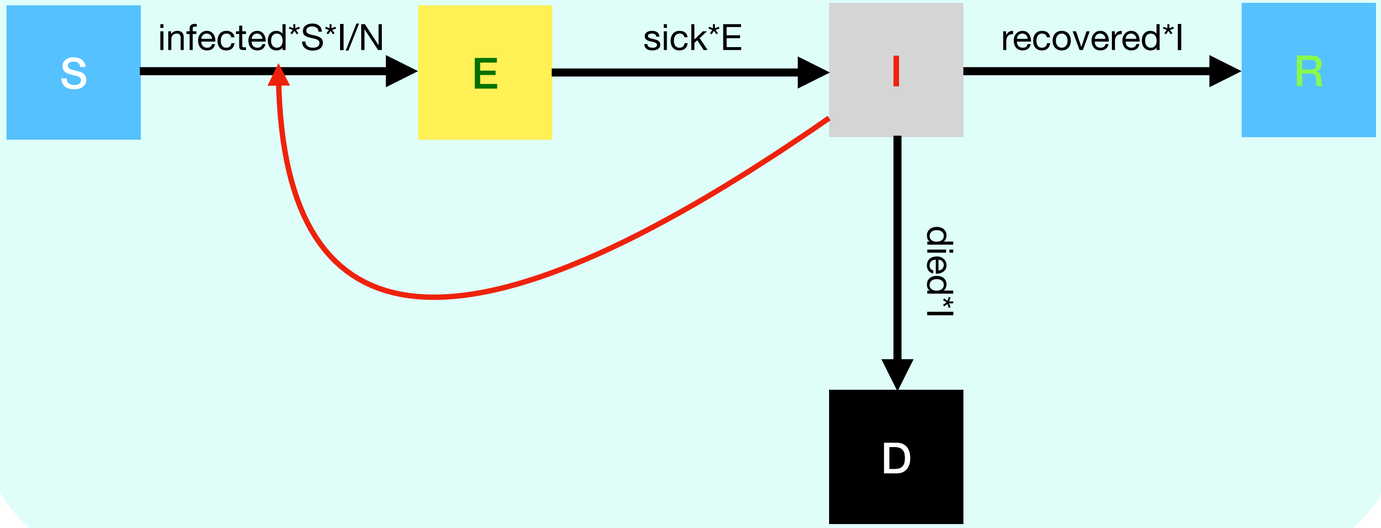
Dynamic Bayesian Network



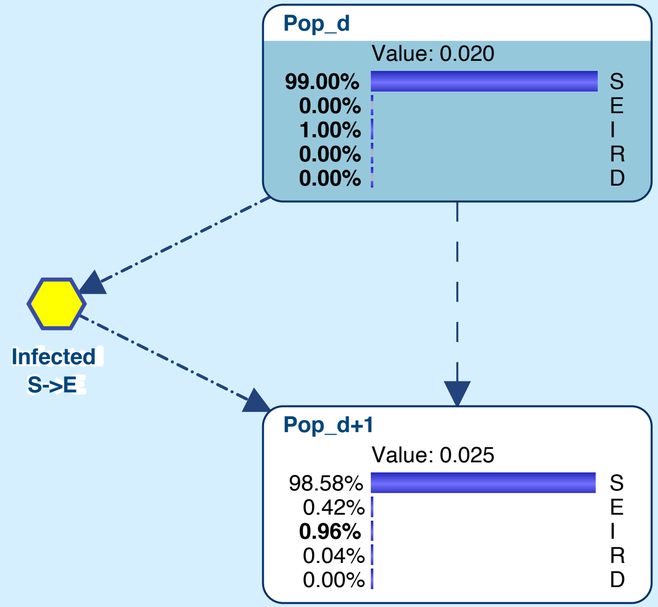
Pop_d	S	E	I	R	D
S	99.571	0.429	0.000	0.000	0.000

Compartmental Model in Epidemiology

Differential Equations



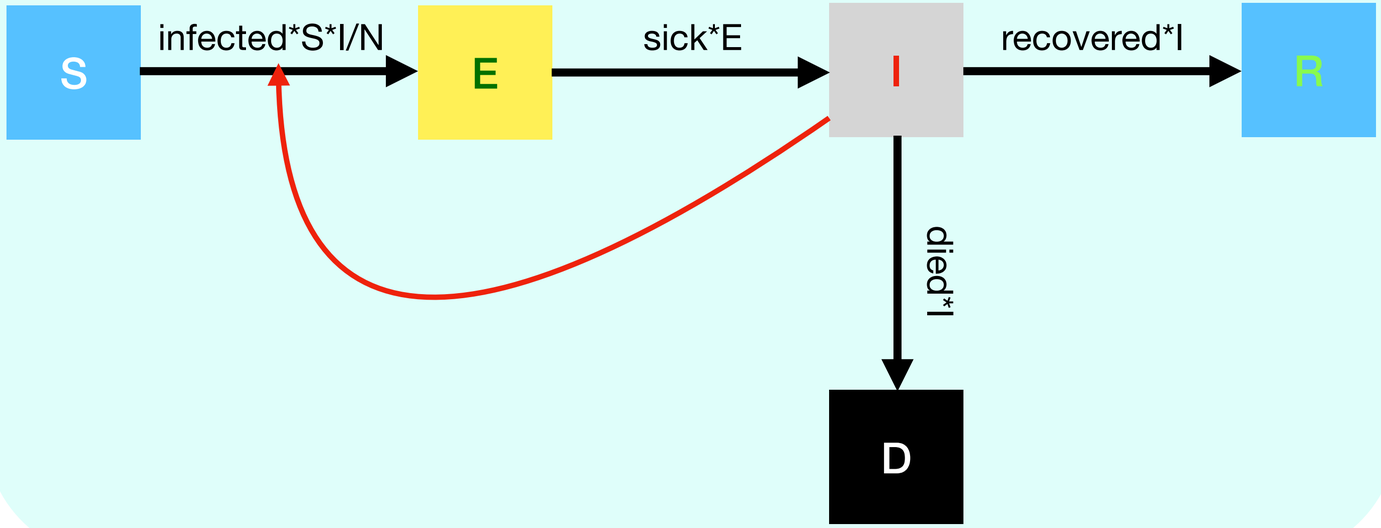
Dynamic Bayesian Network



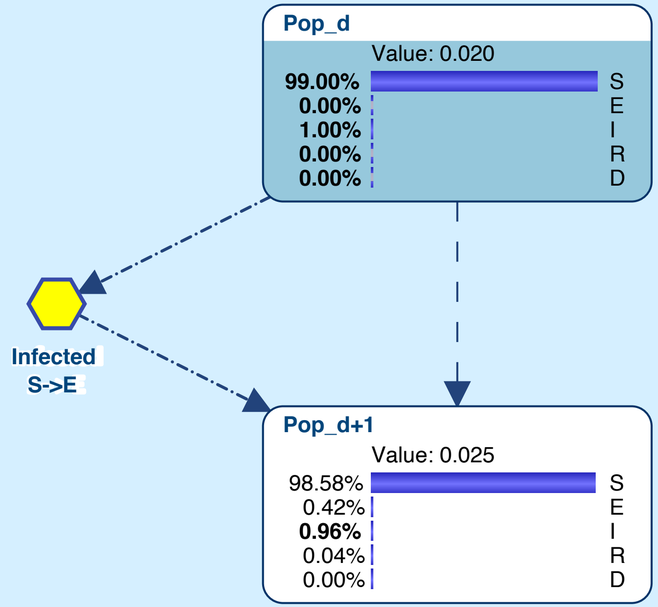
Pop_d	S	E	I	R	D
S	99.571	0.429	0.000	0.000	0.000
E	0.000	80.000	20.000	0.000	0.000

Compartmental Model in Epidemiology

Differential Equations



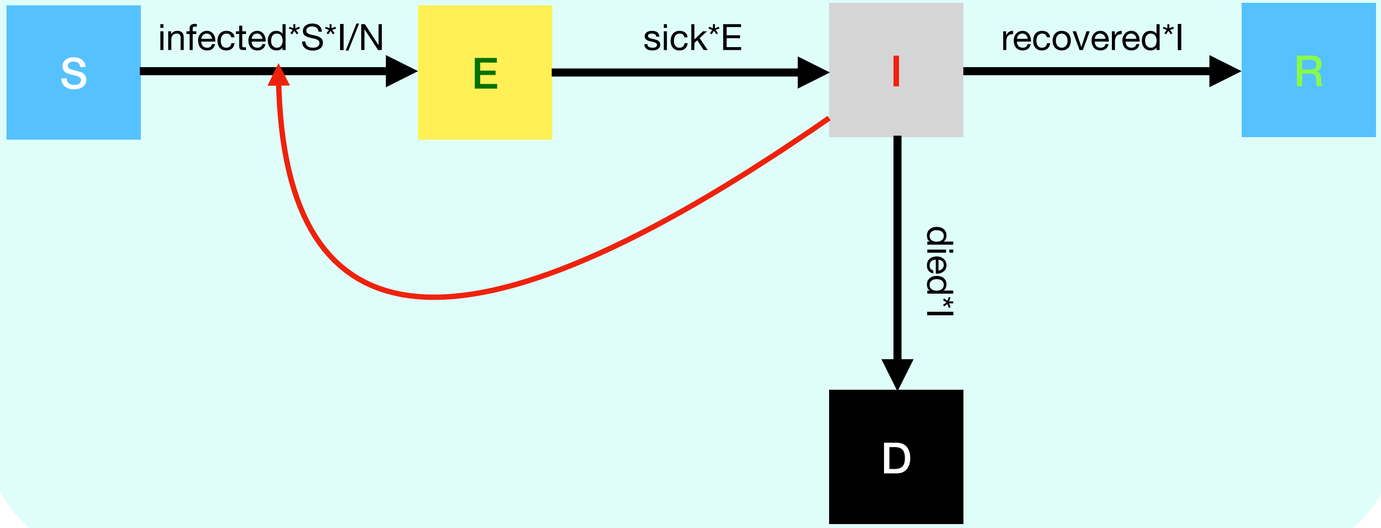
Dynamic Bayesian Network



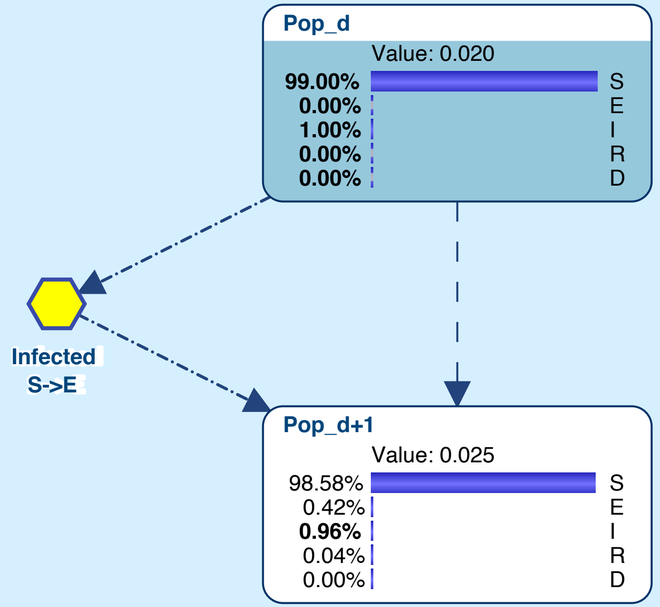
Pop_d	S	E	I	R	D
S	99.571	0.429	0.000	0.000	0.000
E	0.000	80.000	20.000	0.000	0.000
I	0.000	0.000	96.000	3.979	0.021

Compartmental Model in Epidemiology

Differential Equations



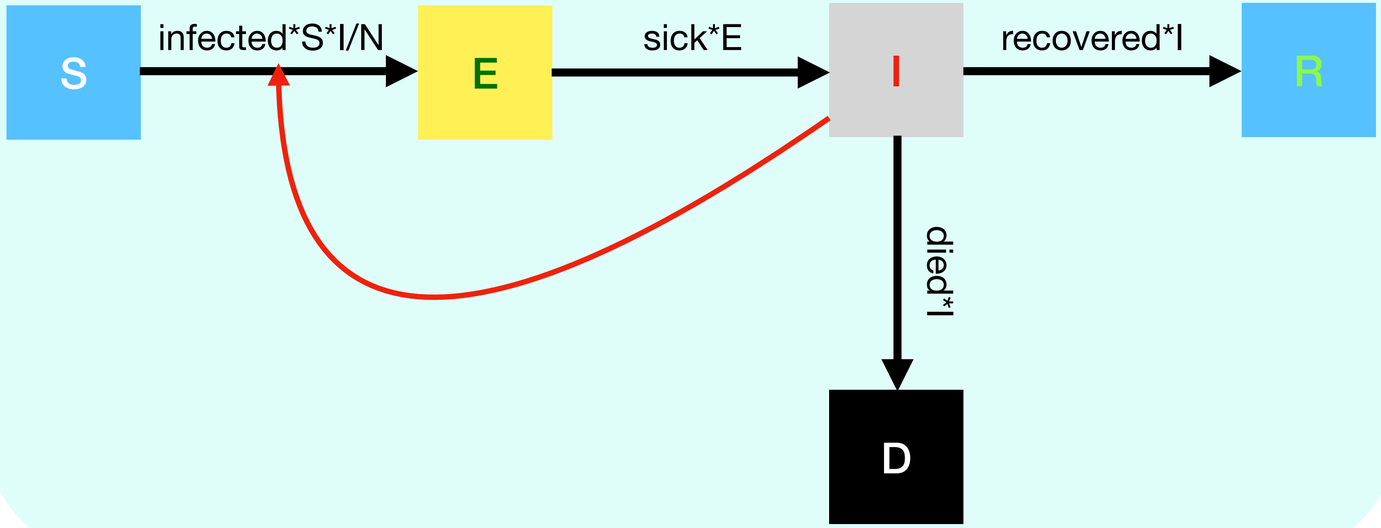
Dynamic Bayesian Network



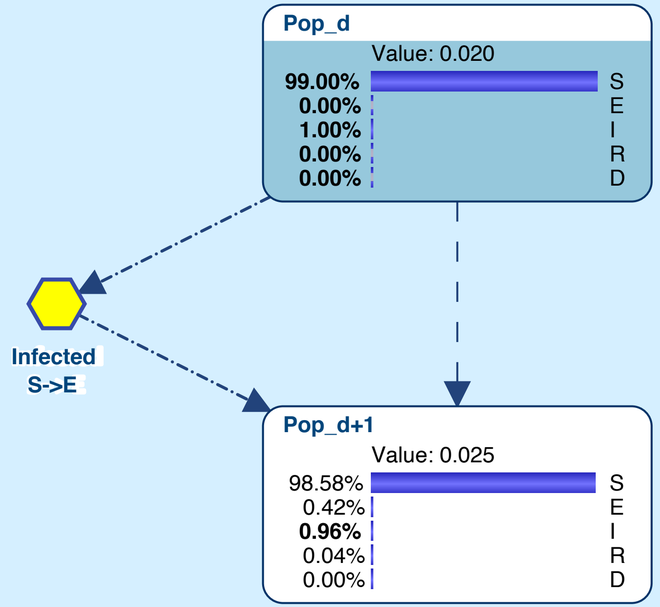
Pop_d	S	E	I	R	D
S	99.571	0.429	0.000	0.000	0.000
E	0.000	80.000	20.000	0.000	0.000
I	0.000	0.000	96.000	3.979	0.021
R	0.000	0.000	0.000	100.000	0.000

Compartmental Model in Epidemiology

Differential Equations



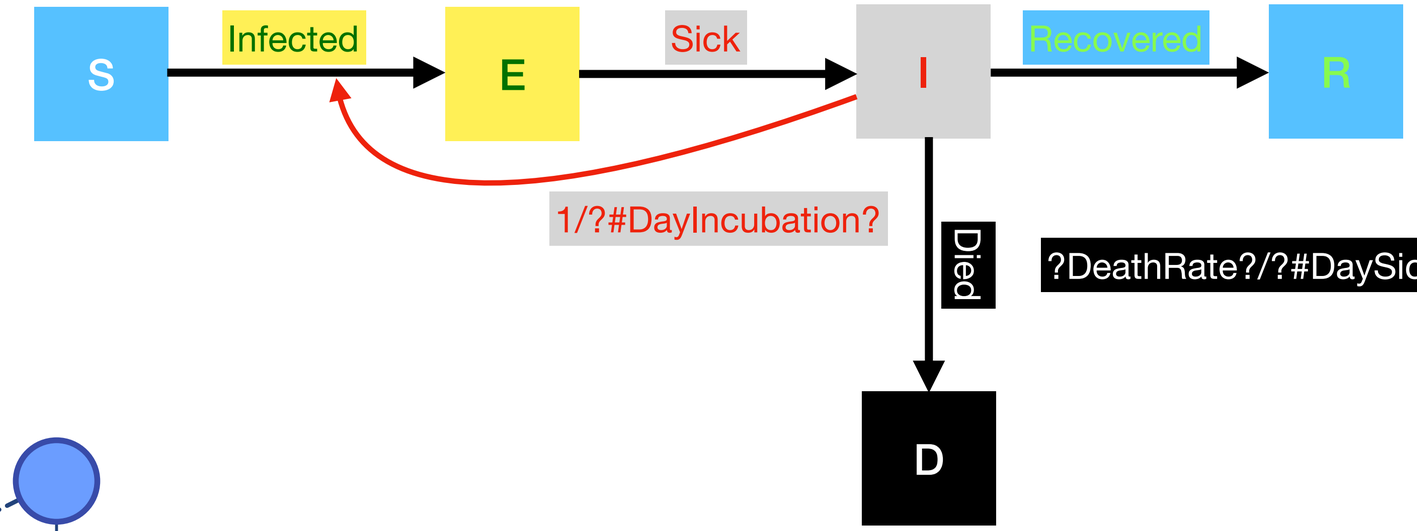
Dynamic Bayesian Network



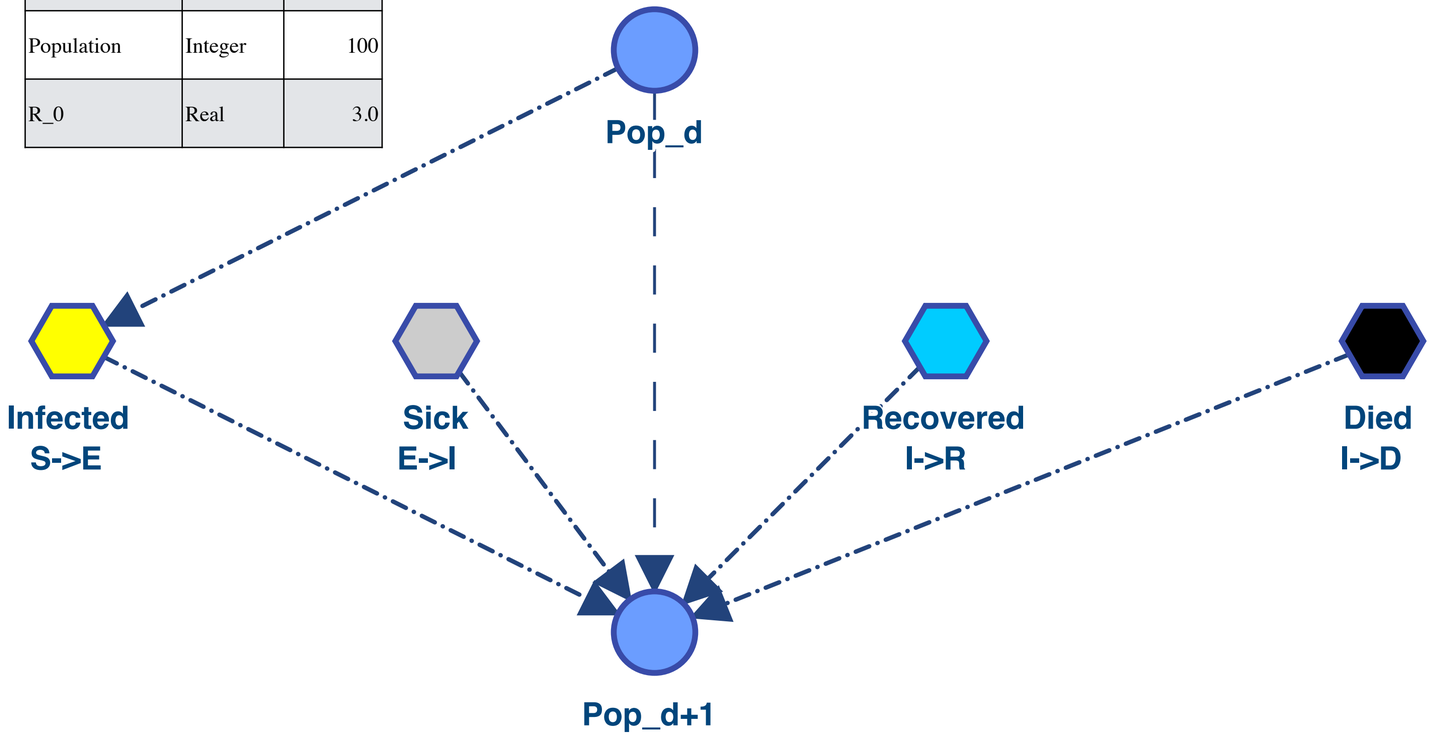
Pop_d	S	E	I	R	D
S	99.571	0.429	0.000	0.000	0.000
E	0.000	80.000	20.000	0.000	0.000
I	0.000	0.000	96.000	3.979	0.021
R	0.000	0.000	0.000	100.000	0.000
D	0.000	0.000	0.000	0.000	100.000

(?R_0?/?#DayContagious?)*StateProb(?Pop_t?, "I")

(1-?DeathRate?)/?#DaySick?



Name (6)	Type	Value
#DayContagious	Real	7
#DayIncubation	Real	5
#DaySick	Real	25
DeathRate	Real	0.0053
Population	Integer	100
R_0	Real	3.0



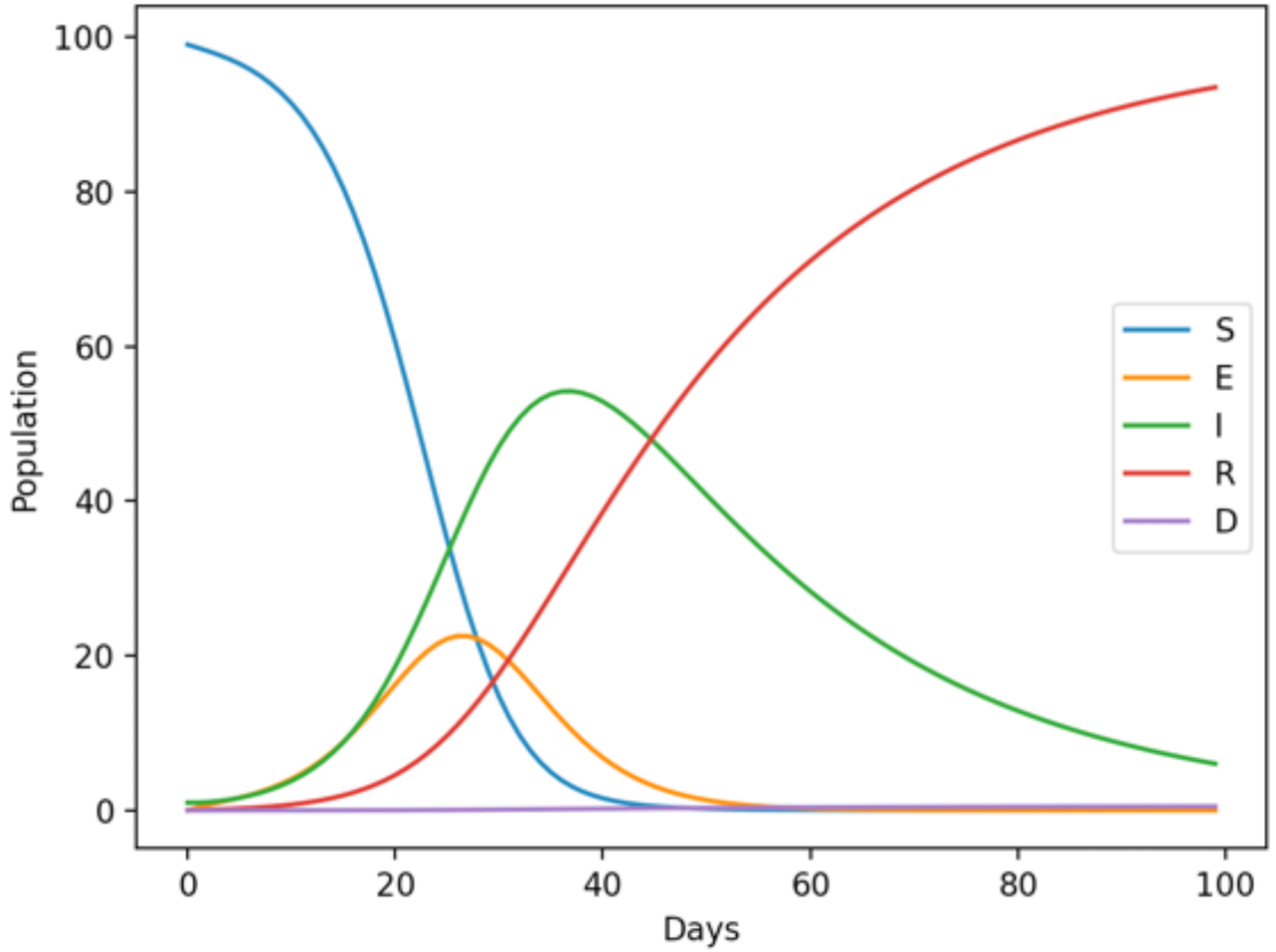
Equation Type: Deterministic Probabilistic

P(?Pop_d+1? | ?Pop_d?) =

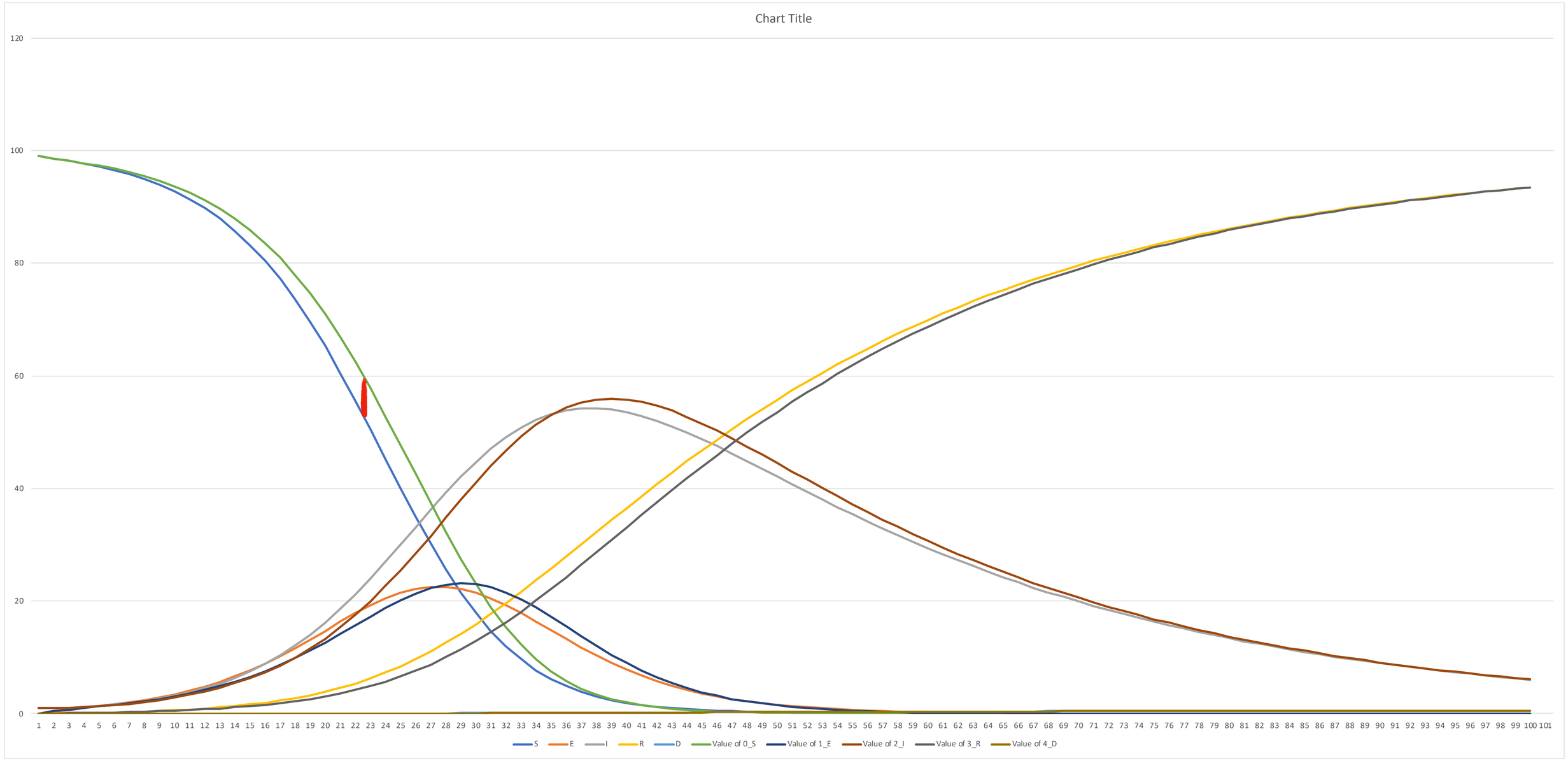
```
Switch(?Pop_d?,
      "S", Switch(?Pop_d+1?,
            "S", 1-?Infected S->E?,
            "E", ?Infected S->E?,
            0),
      "E", Switch(?Pop_d+1?,
            "E", 1-?Sick E->I?,
            "I", ?Sick E->I?,
            0),
      "I", Switch(?Pop_d+1?,
            "I", 1-?Recovered I->R?-?Died I->D?,
            "R", ?Recovered I->R?,
            "D", ?Died I->D?,
            0),
      "R", Switch(?Pop_d+1?, "R", 1, 0),
      Switch(?Pop_d+1?, "D", 1, 0)
)
```

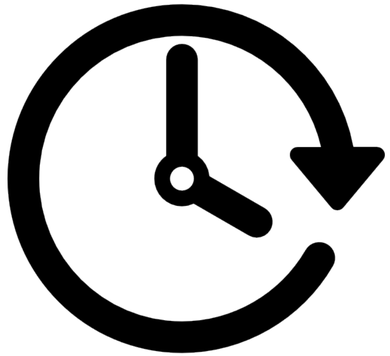
Python Code (SciPy) for Solving this System of Differential Equations

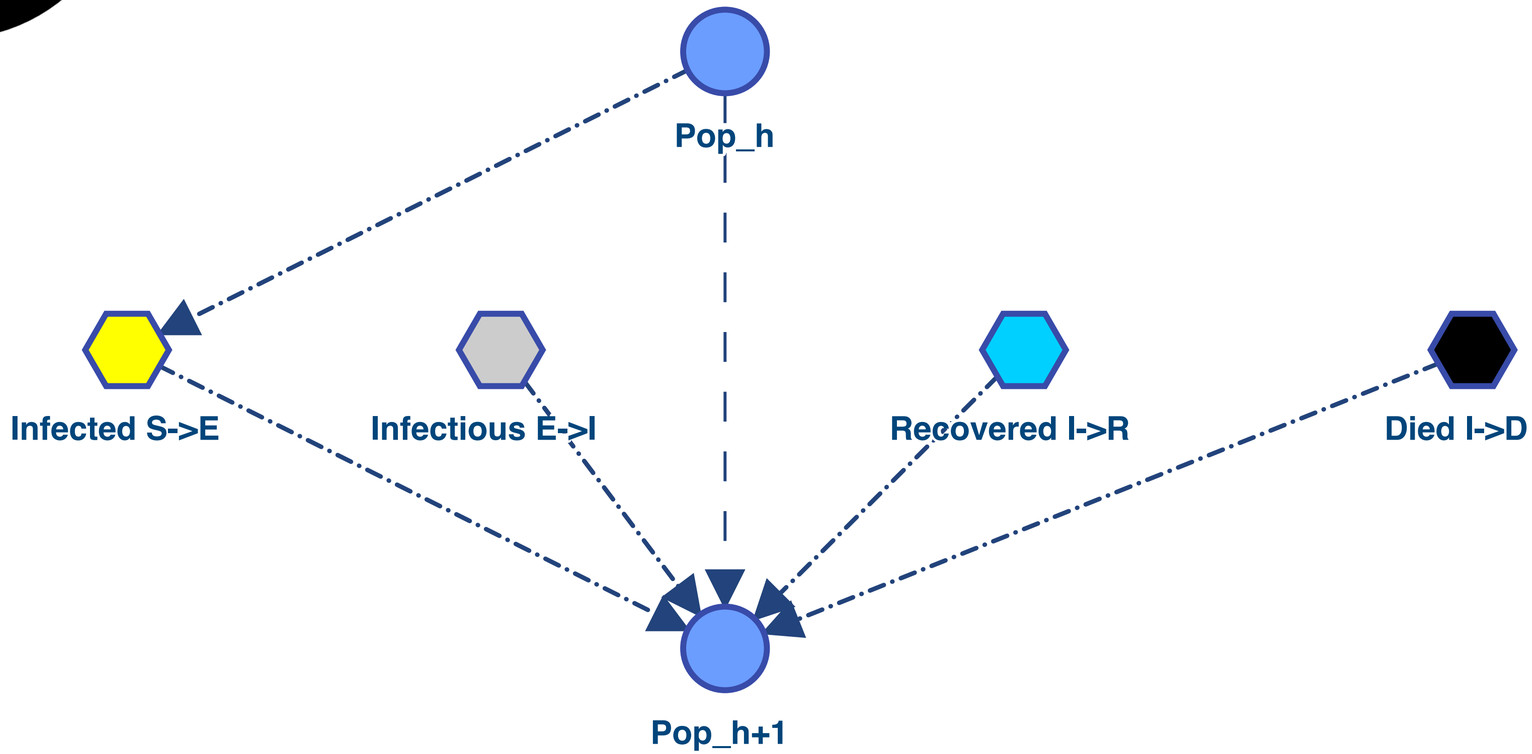
```python
import numpy as np
import matplotlib.pyplot as plt


from scipy.integrate import odeint
from pylab import *


def ode_model(z, t, infected, sick, recovered, died):
    S, E, I, R, D = z
    N = S + E + I + R + D
    dSdt = -infected*S*I/N
    dEdt = infected*S*I/N - sick*E
    dIdt = sick*E - (recovered+died)*I
    dRdt = recovered*I
    dDdt = died*I
    return [dSdt, dEdt, dIdt, dRdt, dDdt]


def ode_solver(t, initial_conditions, params):
    initE, initI, initR, initD, initN = initial_conditions
    infected, sick, recovered, died = params
    initS = initN - (initE + initI + initR + initD)
    res = odeint(ode_model, [initS, initE, initI, initR, initD], t, args=(infected, sick, recovered, died))
    return res

days = 100
population = 100
deathRate = 0.0053
r_0 = 3
dayContagious = 7
dayIncubation = 5
daySick = 25
initial_conditions = [0, 1, 0, 0, population]
params = [r_0/dayContagious, 1/dayIncubation, (1-deathRate)/daySick, deathRate/daySick]
tspan = np.arange(0, days, 1)
sol = ode_solver(tspan, initial_conditions, params)

S=sol[:,0]
E=sol[:,1]
I=sol[:,2]
R=sol[:,3]
D=sol[:,4]

l1 = plt.plot(S, label = "S")
l2 = plt.plot(E, label = "E")
l3 = plt.plot(I, label = "I")
l4 = plt.plot(R, label = "R")
l5 = plt.plot(D, label = "D")

plt.legend()
plt.xlabel("Days")
plt.ylabel("Population")
plt.show()
```



Differential Equations vs Dynamic Bayesian Network



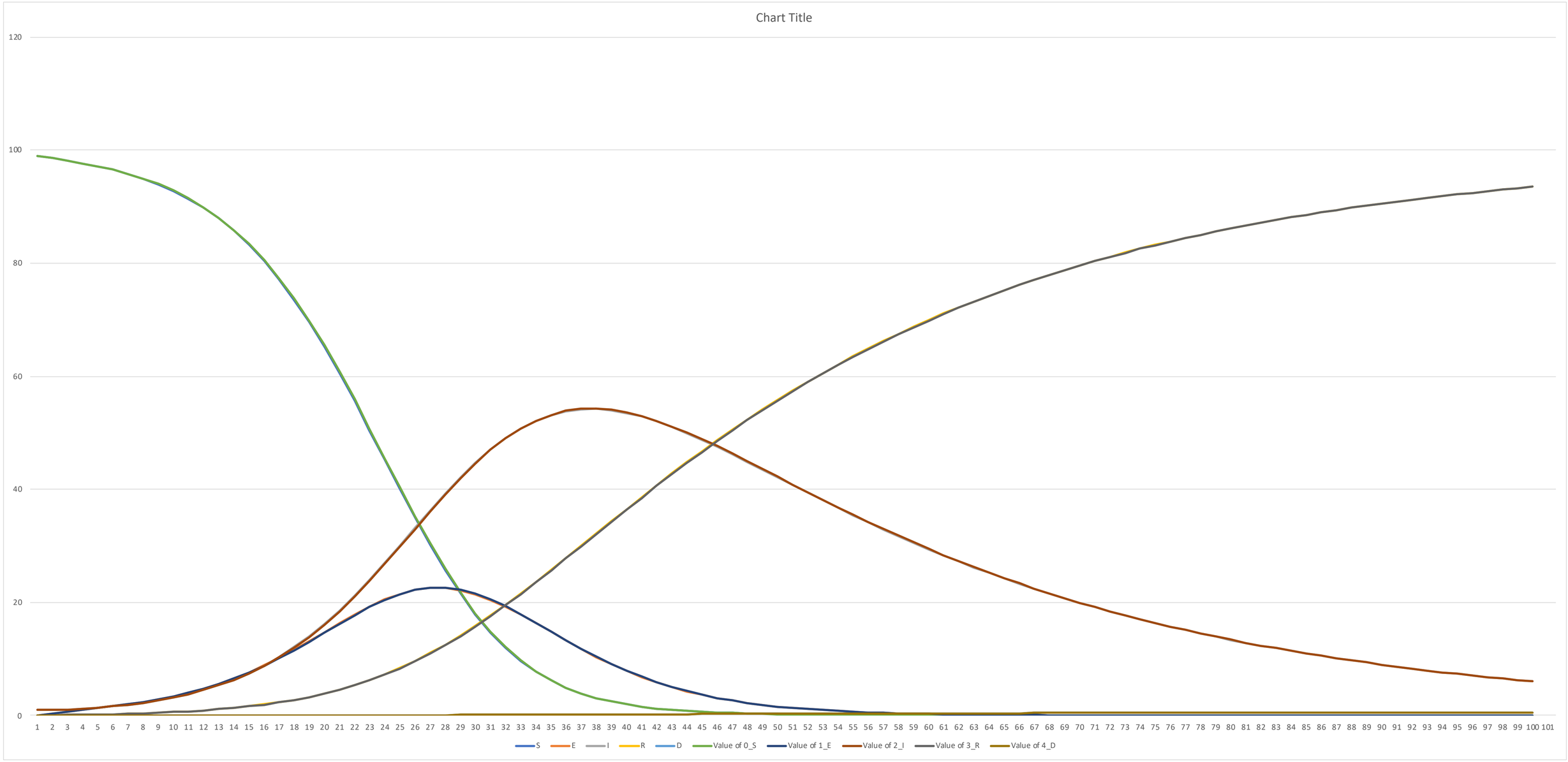




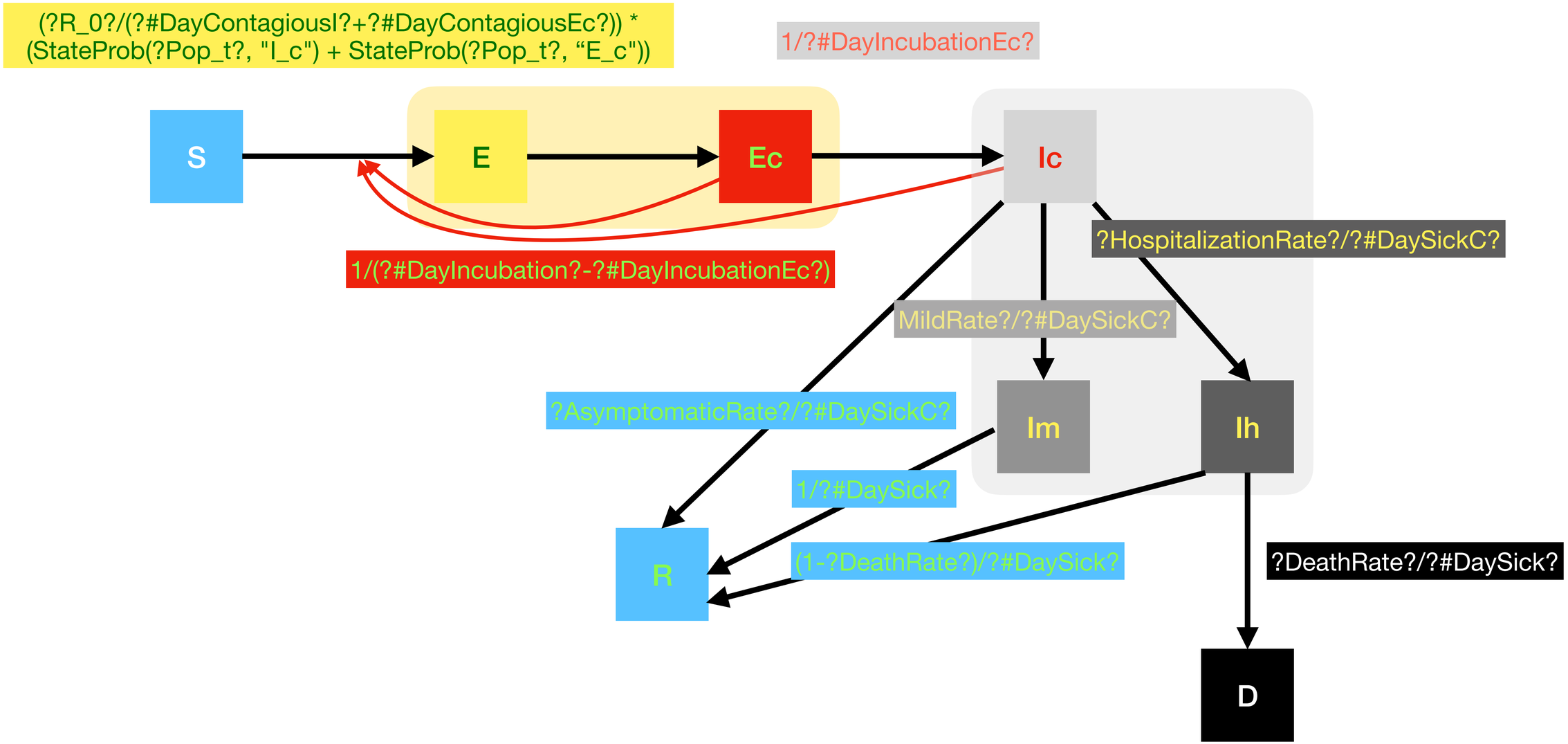
Name (6)	Type	Value
#HourContagious	Real	168.0
#HourIncubation	Real	120.0
#HourSick	Real	600.0
DeathRate	Real	0.0053
Population	Integer	100
R_0	Real	3.0

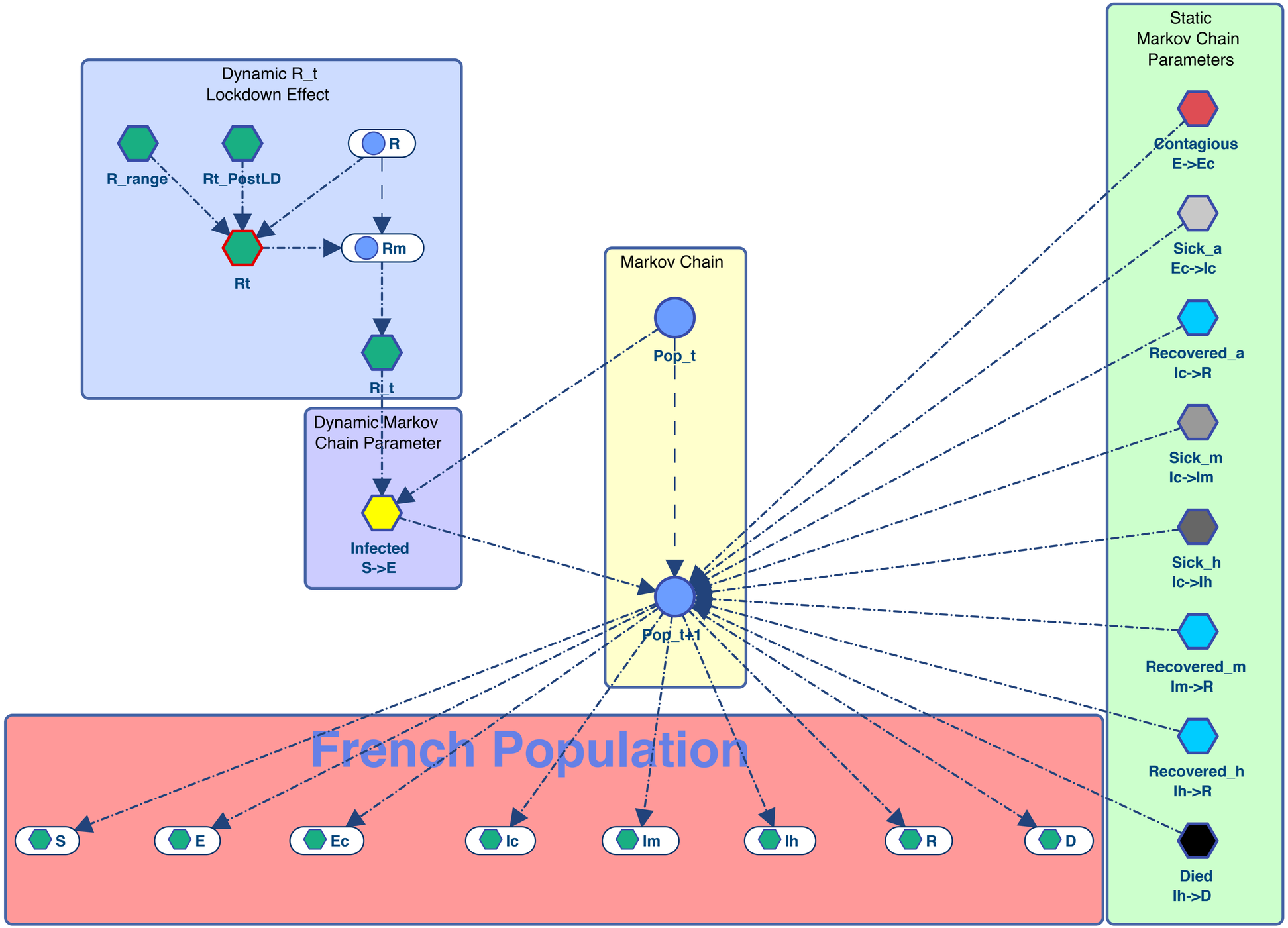
Pop_h	S	E	I	R	D
S	99.999	0.001	0.000	0.000	0.000
E	0.000	99.167	0.833	0.000	0.000
I	0.000	0.000	99.833	0.166	0.001
R	0.000	0.000	0.000	100.000	0.000
D	0.000	0.000	0.000	0.000	100.000

Differential Equations vs Dynamic Bayesian Network (hours)

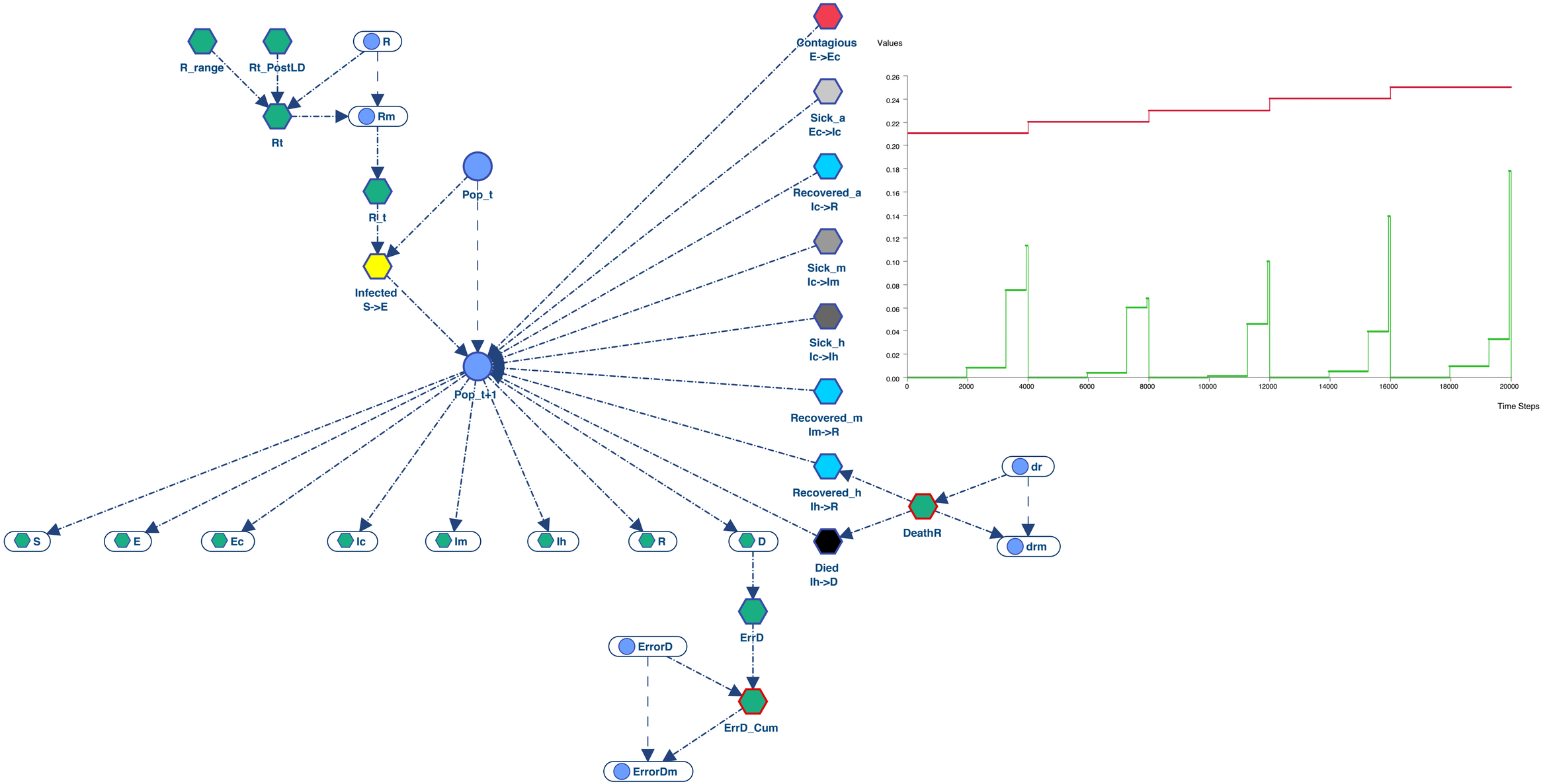


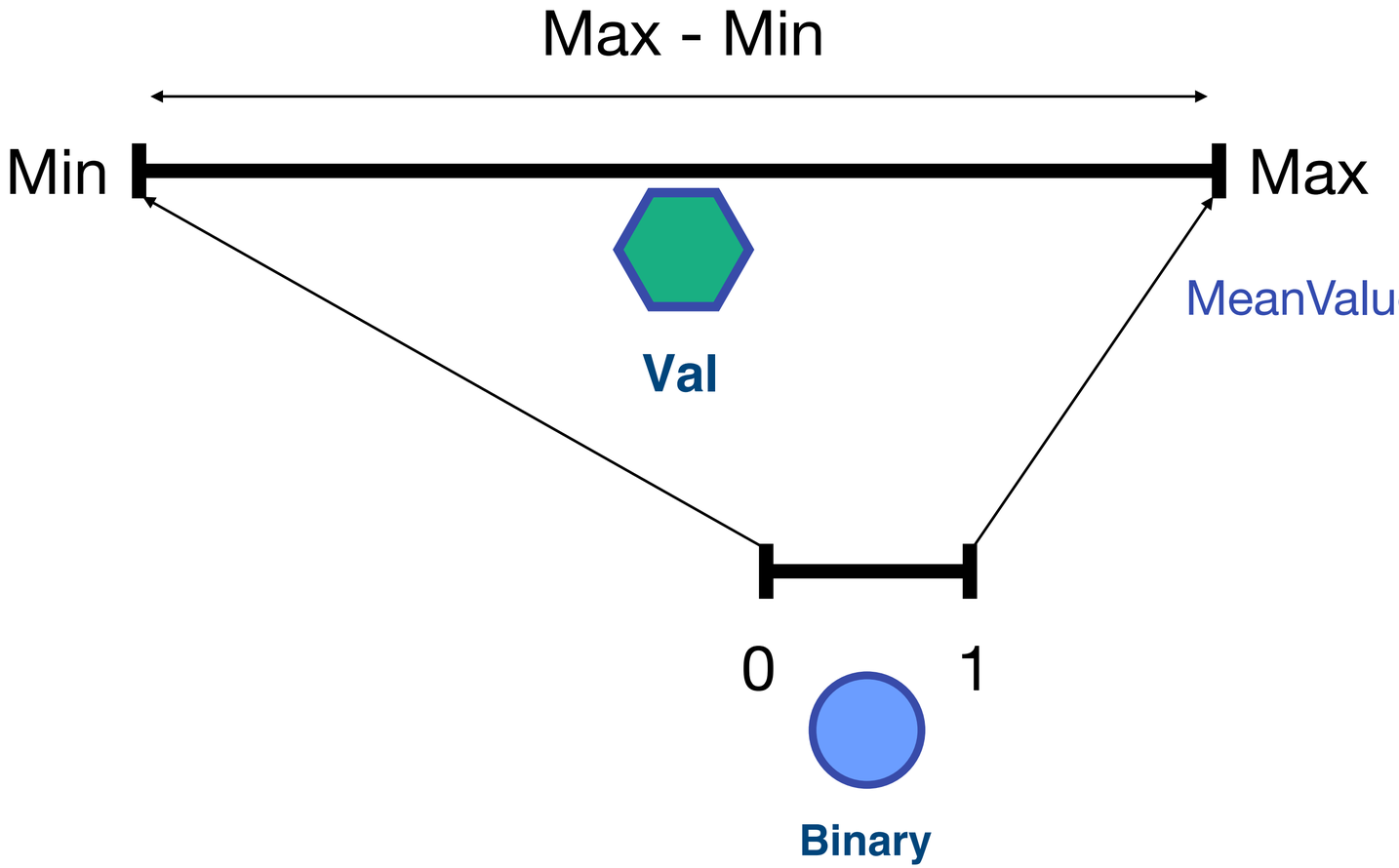
S2E3IRD Model for France




Dynamic R_t
Lockdown Effect
R_range
Rt_PostLD
R
Rt
Rm
Ri_t
Dynamic Markov
Chain Parameter
Infected
S->E
Markov Chain
Pop_t
Pop_t+1
Static
Markov Chain
Parameters
Contagious
E->Ec
Sick_a
Ec->Ic
Recovered_a
Ic->R
Sick_m
Ic->Im
Sick_h
Ic->Ih
Recovered_m
Im->R
Recovered_h
Ih->R
Died
Ih->D
French Population
S
E
Ec
Ic
Im
Ih
R
D

Internal Optimization - Death Rate





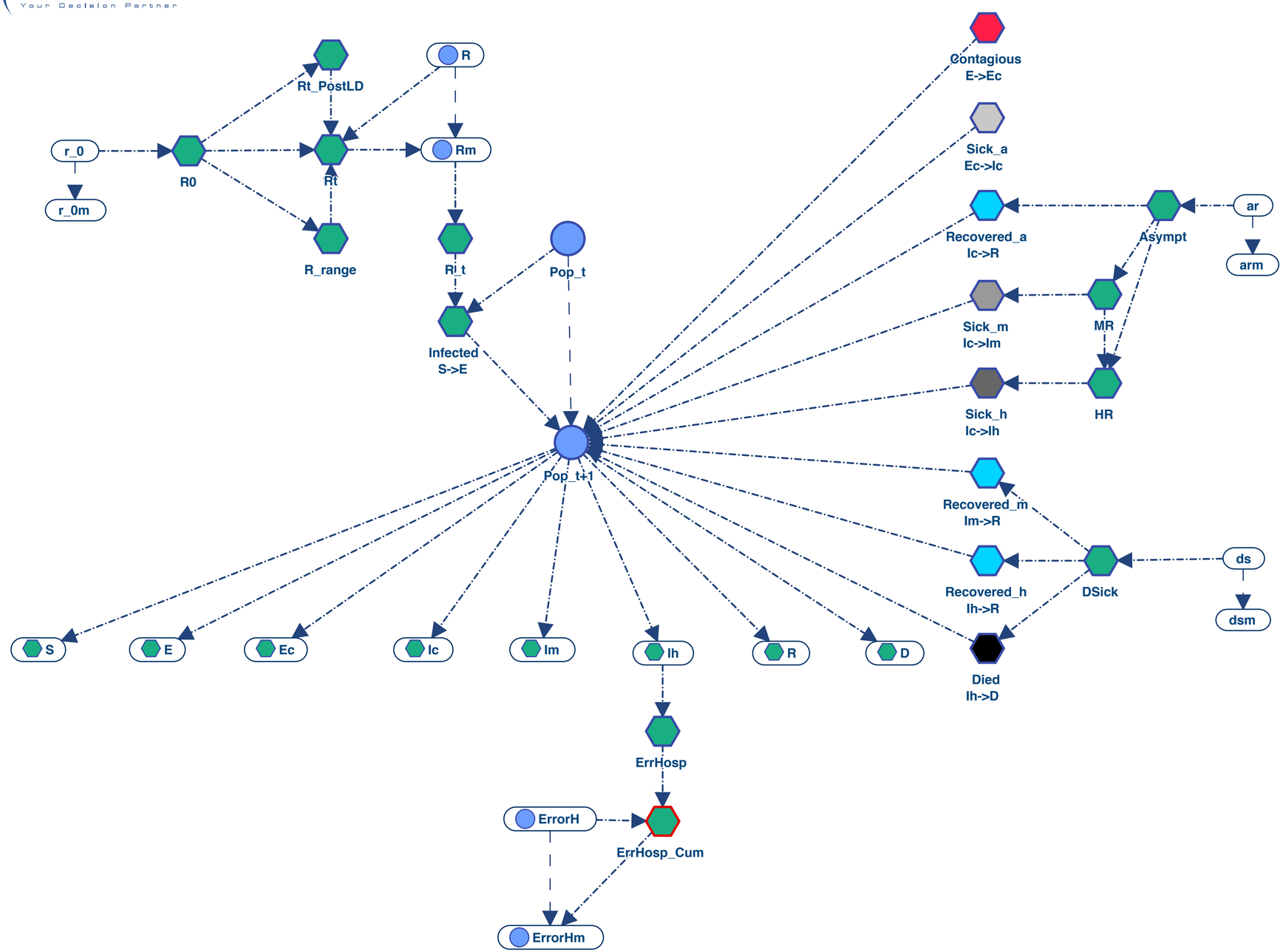
MeanValue(?Binary?) * (?Max? - ?Min?) + ?Min?

if(?Binary?=1,
(?Val? - ?Min?) / (?Max? - ?Min?),
1- (?Val? - ?Min?) / (?Max? - ?Min?))

Optimization with the Java API via Python

```
from jpype import *
from numpy import *

class Inference():
    '''
    classdocs
    '''

    def __init__(self, networkpath, licensekey):
        '''
        Constructor
        '''
        startJVM(getDefaultJVMPath(),
            classpath=
            ['lib/BayesiaEngine.jar','lib/xercesImpl.jar','lib/xml-apis.jar','lib/xml-apis-ext.jar','lib/batik-all.jar'],
            ignoreUnrecognized=True, convertStrings=True)
        apipackage = JPackage("com.bayesia.api")
        self.inference = apipackage.APIInference(networkpath, licensekey, apipackage.APIInference.EXACT_INFERENCE);

    def close(self):
        shutdownJVM()

    def getinference(self):
        return self.inference
```

Optimization with the API




```
infer = Inference('S2E3IRD_France_h2_Opt_ADR.xbl', '                                    ')
apiInf = infer.getinference()
sep = ';'
newline = '\n'
outvar = 'ErrHosp_Cum'

var1 = 'ar'
fvar1 = 'Asympt'
min1 = 0.2
max1 = 0.3
nb1 = 11
range1 = linspace(min1, max1, nb1)

var2 = 'ds'
fvar2 = 'DSick'
min2 = 0.456
max2 = 0.528
nb2 = 4
range2 = linspace(min2, max2, nb2)

var3 = 'r_0'
fvar3 = 'R0'
min3 = 0.0280
max3 = 0.0290
nb3 = 11
range3 =  linspace(min3, max3, nb3)

f = open('output_ADR.csv', 'w', buffering=1)
f.write(fvar1)
f.write(sep)
f.write(fvar2)
f.write(sep)
f.write(fvar3)
f.write(sep)
f.write(outvar)
f.write(newline)

total = nb1 * nb2 * nb3
current = 0

array1 = JArray(JDouble)([0, 0])
array2 = JArray(JDouble)([0, 0])
array3 = JArray(JDouble)([0, 0])
for p1 in range1:
    array1[0] = 1 - p1
    array1[1] = p1
    for p2 in range2:
        array2[0] = 1 - p2
        array2[1] = p2
        for p3 in range3:
            array3[0] = 1 - p3
            array3[1] = p3
            apiInf.setObservedProbabilities(var1, array1)
            apiInf.setObservedProbabilities(var2, array2)
            apiInf.setObservedProbabilities(var3, array3)

            apiInf.setTime(3999);

            f.write(str(apiInf.getFunctionValue(fvar1).getNumber().doubleValue()))
            f.write(sep)
            f.write(str(apiInf.getFunctionValue(fvar2).getNumber().doubleValue()))
            f.write(sep)
            f.write(str(apiInf.getFunctionValue(fvar3).getNumber().doubleValue()))
            f.write(sep)
            f.write(str(apiInf.getFunctionValue(outvar).getNumber().doubleValue()))
            f.write(newline)

            infer.getinference().resetTime()
            current = current + 1
            print(current / total * 100, '%')
f.close()
infer.close()
```

Simulations