CEACOV RSA - Convert Model Input Files

September 10, 2020

1 Overview

I made this python notebook to clearly document the changes I make to Julia's CEACOV input files, as well as to have a way to quickly and efficiently generate several input files with specific changes to model parameters. The other nice aspect of doing things in a somewhat "automated" way is that it becomes much easier to check for differences in model structure or parameter values between several different input sheets, as this task is challenging to do by hand.

2 Copying input parameters into a new model version

The task that I was given was to reproduce all of the runs that Julia did for the manuscript, including a number of sensitivity analyses, but using an updated version of the model structure. This was complicated by that fact that in order to implement these neccessary changes, one of the things I had to do was to move from `v0.6` to `v0.6_9_intvs`. This would have made copying over previously derived input parameters by hand tedious and error-prone, so I decided to write this script that would do it for me in a well-documented and reproducible manner.

[1]:
```
# Packages we'll need
import json
import os
import datetime
import numpy as np
```

[2]:
```
# Get current directory
pwd = os.getcwd()
path_to_old_input_files = pwd + "\\R22 6.3.20\\BC 2M"      # Directory with␣
 ↪input files to convert

# Julia's old input files are written for model v0.6, which has 8 interventions
# For our analysis, we need to use the model version of v0.6 that has 9␣
 ↪interventions
v6_9_intvs_template = pwd + "\\Templates\\v6_9_intvs_template.json"

# Old input file names
old_input_names = ['batch 0\\BC.json',
                   'batch 1\\CTA.json',
                   'batch 2\\CTA + IC.json',
```

```
                    'batch 4\\CTA + IC + MSS.json',
                    'batch 3\\CTA + IC +Q.json',
                    'batch 5\\CTA + IC + MSS+ Q.json']
```

[3]:
```
# Append file name to file path
old_input_names = [path_to_old_input_files + "\\" + x for x in old_input_names]
n = len(old_input_names)
old_inputs = []
new_inputs = []

for i in range(n):

    # Copy the old input file
    with open(old_input_names[i]) as f1:
        old_inputs.append(json.load(f1))
        f1.close()

    # Create a blank template for our new input file
    with open(v6_9_intvs_template) as f2:
        new_inputs.append(json.load(f2))
        f2.close()
```

[4]:
```
def get_items(mydict):

    """
    This function takes CEACOV input files (represented as a nested␣
 ↪dictionary),
    and extracts each input value as well as it's "filepath" in the dictionary.
    """

    paths = []
    items = []

    path_stack = [[]]
    item_stack = [mydict]

    while len(item_stack) > 0:
        x = item_stack.pop(0)
        p = path_stack.pop(0)

        k_list = list(x.keys())

        for k in k_list:
            if isinstance(x[k],dict):
                p2 = p.copy()
                p2.append(k)
                path_stack.append(p2)
```

```
                item_stack.append(x[k])
            else:
                p2 = p.copy()
                p2.append(k)
                paths.append(p2)
                items.append(x[k])

    return(paths,items)
```

[5]:
```
def update_item(mydict,mypath,myvalue):

    """
    This function updates the value of an item in a nested dictionary.
    The "filepath" of the item is specified by the variable mypath,
    and the new value of the item is specified by myvalue.
    """
    p = mypath.copy()

    if len(p) > 1:
        k = p.pop(0)
        mydict[k] = update_item(mydict[k],p,myvalue)

    else:
        k = p.pop(0)
        mydict[k] = myvalue

    return(mydict)
```

[6]:
```
# Copy parameters that are common to old input files and new input files

for i in range(len(old_inputs)):
    paths,items = get_items(old_inputs[i])
    for j in range(len(items)):
        new_inputs[i] = update_item(new_inputs[i],paths[j],items[j])
```

3 Updating Natural History Parameters

[7]:
```
# Age distribution
for i in range(len(new_inputs)):
    new_inputs[i]['initial state']['subpopulation dist']['0-19y'] = 0.4026
    new_inputs[i]['initial state']['subpopulation dist']['20-59y'] = 0.5148
    new_inputs[i]['initial state']['subpopulation dist']['>=60y'] = 0.0826
```

[8]:
```
# Transition probabilities

path1 = [0.319, # pre-inf to asymptomatic
```

```
         0.100] # asymptomatic to recovered

path2 = [0.319, # pre-inf to asymptomatic
         0.393, # asymptomatic to mild/moderate
         0.095] # mild/moderate to recovered

path3 = [0.319, # pre-inf to asymptomatic
         0.393, # asymptomatic to mild/moderate
         0.143, # mild/moderate to severe
         0.091] # severe to recovered

path4 = [0.319, # pre-inf to asymptomatic
         0.393, # asymptomatic to mild/moderate
         0.283, # mild/moderate to severe
         0.202, # severe to critical
         0.096, # critical to recuperation
         0.161] # recuperation to recovered

# The only real difference between these numbers and the original input
# file is for severe to critical, and critical to recuperation in path 4
# Rather than changing all those numbers by what amounts to a rounding error,
# only change the severe/critical state in path 4

path4_severe_to_recovered = 0.055
critical_to_death = [0.000006,0.0038,0.0500]  # by age: 0-19, 20-59, 60+

for i in range(len(new_inputs)):
    new_paths,new_items = get_items(new_inputs[i])

    dis_new_paths = [new_paths[i] for i in range(len(new_paths)) \
             if "disease progression" in new_paths[i]]

    dis_new_items = [new_items[i] for i in range(len(new_paths)) \
             if "disease progression" in new_paths[i]]

    for j in range(len(dis_new_paths)):
        if dis_new_items[j] != 0:
            if "for severity = critical" in dis_new_paths[j] and "from severe␣
↪to critical" in dis_new_paths[j]:
                dis_new_items[j] = path4[3]
            if "for severity = critical" in dis_new_paths[j] and "from critical␣
↪to recuperation" in dis_new_paths[j]:
                dis_new_items[j] = path4[4]
            if "daily prob recovery from severe state in critical path" in␣
↪dis_new_paths[j]:
                dis_new_items[j] = path4_severe_to_recovered
```

```
    for k in range(len(dis_new_paths)):
        new_inputs[i] =␣
↪update_item(new_inputs[i],dis_new_paths[k],dis_new_items[k])
```

4 PCR Test Sensitivity / Return Time

[9]:
```
PCR_return_time = 4    # BC: 4 SA: 0 or 6
PCR_sens = 0.7  # BC: 0.7 SA: 0.5 or 0.9

for i in range(len(new_inputs)):
    new_inputs[i]['tests']['test 1']['result return time'] = PCR_return_time

    new_inputs[i]['tests']['test 1']\
    ['probability of positive result']['for asymptomatic'] = PCR_sens
    new_inputs[i]['tests']['test 1']\
    ['probability of positive result']['for mild/moderate'] = PCR_sens
    new_inputs[i]['tests']['test 1']\
    ['probability of positive result']['for severe'] = PCR_sens
    new_inputs[i]['tests']['test 1']\
    ['probability of positive result']['for critical'] = PCR_sens
    new_inputs[i]['tests']['test 1']\
    ['probability of positive result']['for recuperation'] = PCR_sens
```

5 Transmission Multipliers

[10]:
```
# Rate multiplier thresholds
t0 = 10000
t1 = 10010
t2 = 10020
t3 = 10030
t4 = 10040

# Time varying Re: 1.5 for 0-30, 1.0 for 30-60, 1.125 for 60-90, 1.3 for 90+

desired_R0_0 = 1.5 # 1.5, 0.9, 1.1, 1.2, 2.6
desired_R0_1 = 1.5
desired_R0_2 = 1.5
desired_R0_3 = 1.5
desired_R0_4 = 1.5

my_mult_0 = desired_R0_0/1.5
my_mult_1 = desired_R0_1/1.5
my_mult_2 = desired_R0_2/1.5
my_mult_3 = desired_R0_3/1.5
my_mult_4 = desired_R0_4/1.5
```

```
for i in range(len(new_inputs)):

    paths,items = get_items(new_inputs[i])

    new_inputs[i]['transmissions']\
    ['rate multiplier thresholds']['t0'] = t0

    new_inputs[i]['transmissions']\
    ['rate multiplier thresholds']['t1'] = t1

    new_inputs[i]['transmissions']\
    ['rate multiplier thresholds']['t2'] = t2

    new_inputs[i]['transmissions']\
    ['rate multiplier thresholds']['t3'] = t3

    # Make all rate multipliers the same as in 0 < day < t0
    # Before they were all 1.0 since they weren't being used, but now we're␣
 ↪using them

    for j in range(len(items)):
        if 'rate multipliers' in paths[j]:
            intv_name = paths[j][len(paths[j])-1]
            items[j] = new_inputs[i]['transmissions']['rate multipliers']\
            ['for 0 <= day# < t0'][intv_name]

    # Update rate multipliers

    for j in range(len(items)):
        if 'rate multipliers' in paths[j] and 'for 0 <= day# < t0' in paths[j]:
            items[j] = items[j]*my_mult_0
        if 'rate multipliers' in paths[j] and 'for t0 <= day# < t1' in paths[j]:
            items[j] = items[j]*my_mult_1
        if 'rate multipliers' in paths[j] and 'for t1 <= day# < t2' in paths[j]:
            items[j] = items[j]*my_mult_2
        if 'rate multipliers' in paths[j] and 'for t2 <= day# < t3' in paths[j]:
            items[j] = items[j]*my_mult_3
        if 'rate multipliers' in paths[j] and 'day# > t4' in paths[j]:
            items[j] = items[j]*my_mult_4

    for k in range(len(items)):
        new_inputs[i] = update_item(new_inputs[i],paths[k],items[k])
```

6 Effectiveness of IC and QC

[11]:
```
# Home isolation/quarantine effectiveness

E_home = 0.5    # BC: 0.5 - 0.25, 0.75
E_center = 0.05 # BC: 0.05 - 0.01, 0.1, 0.15, 0.25, 0.35

# BC
for i in [0,1]:
    paths,items = get_items(new_inputs[i])
    for j in range(len(items)):
        if 'rate multipliers' in paths[j] and ('for intervention 1' in paths[j]␣
 ↪or 'for intervention 2' in paths[j]):
            items[j] = items[j]/0.5*E_home
    for k in range(len(items)):
        new_inputs[i] = update_item(new_inputs[i],paths[k],items[k])

# CT+IC / CT+IC+MSS
for i in [2,3]:
    paths,items = get_items(new_inputs[i])
    for j in range(len(items)):
        if 'rate multipliers' in paths[j] and 'for intervention 3' in paths[j]:
            items[j] = items[j]/0.5*E_home
        if 'rate multipliers' in paths[j] and ('for intervention 1' in paths[j]␣
 ↪or 'for intervention 2' in paths[j]):
            items[j] = items[j]/0.05*E_center
    for k in range(len(items)):
        new_inputs[i] = update_item(new_inputs[i],paths[k],items[k])

for i in [4,5]:
    paths,items = get_items(new_inputs[i])
    for j in range(len(items)):
        if 'rate multipliers' in paths[j] and ('for intervention 1' in paths[j]␣
 ↪or 'for intervention 2' in paths[j] or 'for intervention 3' in paths[j]):
            items[j] = items[j]/0.05*E_center
    for k in range(len(items)):
        new_inputs[i] = update_item(new_inputs[i],paths[k],items[k])
```

7 Probability of presenting to care if susceptible or recovered

One of the issues with the previous analysis was how we dealt with non-infected cases presenting to care due to ILI or contact tracing. We now deal with it outside of the model, so need to set the probability of presenting to care if susceptible / recovered to zero in our input files.

[12]:
```
# Paths to variables we are setting to zero
```

```
p0 = ['testing strategies',
      'no intervention',
      'probability of presenting to care']

health_states = ['while susceptible',
                 'while pre-infectious incubation',
                 'while asymptomatic',
                 'while mild/moderate',
                 'while severe',
                 'while critical',
                 'while recuperation',
                 'while recovered']

hs_list = [p0 + [health_states[i]] for i in range(len(health_states))]

# Average length of each health state
# Determined by taking wieghted average over the age/severity dist
# For susceptible and recovered, put -1 to indicate that it's just there
# to make the math work (doesn't matter since p2c prob is zero)

t1 = [-1.000, # susceptible
       2.600,  # pre-infectious incubation (Julia: 3.1)
       4.710,  # asymptomatic (Julia: 4.26)
       9.890,  # mild/moderate (Julia: 10.39)
       9.050,  # severe
       11.90,  # critical
       5.700,  # recuperation
      -1.000]  # recovered
```

[13]:
```
# Base case
p1 = [[0.0,0.000,0.000,0.30,1.0,1.0,1.0,0.0],  # BC
      [0.0,0.100,0.100,0.35,1.0,1.0,1.0,0.0],  # CT
      [0.0,0.100,0.100,0.35,1.0,1.0,1.0,0.0],  # CT+IC
      [0.0,0.125,0.125,0.40,1.0,1.0,1.0,0.0],  # CT+IC+MSS
      [0.0,0.100,0.100,0.35,1.0,1.0,1.0,0.0],  # CT+IC+QC
      [0.0,0.125,0.125,0.40,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[14]:
```
# CT+MSS effectiveness 50% of BC
#p1 = [[0.0,0.000,0.000,0.300,1.0,1.0,1.0,0.0],  # BC
#       [0.0,0.050,0.050,0.325,1.0,1.0,1.0,0.0],  # CT
#       [0.0,0.050,0.050,0.325,1.0,1.0,1.0,0.0],  # CT+IC
#       [0.0,0.0625,0.0625,0.35,1.0,1.0,1.0,0.0],  # CT+IC+MSS
#       [0.0,0.050,0.050,0.325,1.0,1.0,1.0,0.0],  # CT+IC+QC
#       [0.0,0.0625,0.0625,0.35,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[15]:
```
# CT+MSS effectiveness 75% of BC
#p1 = [[0.0,0.000,0.000,0.30,1.0,1.0,1.0,0.0],  # BC
```

```
#       [0.0,0.075,0.075,0.3375,1.0,1.0,1.0,0.0],  # CT
#       [0.0,0.075,0.075,0.3375,1.0,1.0,1.0,0.0],  # CT+IC
#       [0.0,0.09375,0.09375,0.375,1.0,1.0,1.0,0.0],  # CT+IC+MSS
#       [0.0,0.075,0.075,0.3375,1.0,1.0,1.0,0.0],  # CT+IC+QC
#       [0.0,0.09375,0.09375,0.375,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[16]:
```
# 125% of BC
#p1 = [[0.0,0.000,0.000,0.30,1.0,1.0,1.0,0.0],  # BC
#       [0.0,0.1250,0.1250,0.3625,1.0,1.0,1.0,0.0],  # CT
#       [0.0,0.1250,0.1250,0.3625,1.0,1.0,1.0,0.0],  # CT+IC
#       [0.0,0.15625,0.15625,0.425,1.0,1.0,1.0,0.0],  # CT+IC+MSS
#       [0.0,0.1250,0.1250,0.3625,1.0,1.0,1.0,0.0],  # CT+IC+QC
#       [0.0,0.15625,0.15625,0.425,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[17]:
```
# 150% of BC
#p1 = [[0.0,0.000,0.000,0.30,1.0,1.0,1.0,0.0],  # BC
#       [0.0,0.150,0.150,0.375,1.0,1.0,1.0,0.0],  # CT
#       [0.0,0.150,0.150,0.375,1.0,1.0,1.0,0.0],  # CT+IC
#       [0.0,0.1875,0.1875,0.45,1.0,1.0,1.0,0.0],  # CT+IC+MSS
#       [0.0,0.150,0.150,0.375,1.0,1.0,1.0,0.0],  # CT+IC+QC
#       [0.0,0.1875,0.1875,0.45,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[18]:
```
# 200 % of BC
#p1 = [[0.0,0.000,0.000,0.30,1.0,1.0,1.0,0.0],  # BC
#       [0.0,0.20,0.20,0.40,1.0,1.0,1.0,0.0],  # CT
#       [0.0,0.20,0.20,0.40,1.0,1.0,1.0,0.0],  # CT+IC
#       [0.0,0.25,0.25,0.50,1.0,1.0,1.0,0.0],  # CT+IC+MSS
#       [0.0,0.20,0.20,0.40,1.0,1.0,1.0,0.0],  # CT+IC+QC
#       [0.0,0.25,0.25,0.50,1.0,1.0,1.0,0.0]]  # CT+IC+QC+MSS
```

[19]:
```
# Convert to daily probability
p2 = [[1 - (1 - p1[i][j])**(1/t1[j]) for j in range(len(p1[i]))] for i in␣
 ↪range(len(p1))]

# Set presentation to care probs
for i in range(len(new_inputs)):
    for j in range(len(hs_list)):
        new_inputs[i] = update_item(new_inputs[i],hs_list[j],p2[i][j])
```

8 Adding an intervention to send a fixed percentage of mild / moderate cases to quarantine centers

People who have mild/moderate cases of COVID-19 can present to care due to either their symptoms, a contact tracing event, or due to mass symptom screening. Contact tracing and mass symptom screening aren't modeled explicity, but manifest as an increase in the probability of presenting to care over the duration of a person's health state, shown in the table below.

	BC	CT+IC+QC	CT+IC+QC+MSS
Probability of presenting to care	30%	35%	40%

Of mild/moderate cases presenting to care, some number will have a false-negative PCR test. Of those with a negative PCR test, only those who are contacts of someone with COVID-19 are eligible to go to a quarantine center. Because of this, we need to figure out what proportion of mild/moderate cases testing negative are expected to have been contact traced, and thus need to be sent to a quarantine center.

I assume that presenting due to symptoms, contact tracing, and mass symtom screening are independent but not mutually-exclusive events. This allows us to invoke the following probability rule:

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) = P(A) + P(B) - P(A)P(B)$$

Let's say that event A represents presenting to care due to symptoms, and event B represents presenting to care due to contact tracing. From the above table, we can see that $P(A) = 0.3$ and $P(A \cup B) = 0.35$. Substituting these values into the above equation allows us to solve for $P(B)$

$$(0.35) = (0.3) + P(B) - (0.3)P(B)$$

$$P(B) = \frac{1}{14} \approx 0.0714$$

For interventions with mass symptom screening, we have a third event (C) which represents a person presenting to care due to a symptom screen. This allows us to invoke a similar probability rule as before:

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$$

If we assume all three eventes to be independent, we can re-write as follows and solve for $P(C)$, the probability of being mass symptom screened.

$$P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A)P(B) - P(A)P(C) - P(B)P(C) + P(A)P(B)P(C)$$

$$(0.4) = (0.3) + (0.0714) + P(C) - (0.3)(0.0714) - (0.3)P(C) - (0.0714)P(C) + (0.3)(0.0714)P(C)$$

$$P(C) \approx 0.0769$$

This implies that mild/moderate cases have a 7.69% chance of being mass-symptom screened over the duration of their health state. We might want to check if this number makes sense.

We now want to know the fraction of mild/moderate presenting to care who were contact traced (and thus eligible for quarantine). For interventions with contact tracing, the numerator of this fraction is $P(B)$, while for those without contact tracing the numerator is zero. The denominator of this fraction is the total probability of presenting to care, given by either $P(A \cup B)$ or $P(A \cup B \cup C)$, depending on whether or not there is mass symptom screening.

$$\frac{P(B)}{P(A \cup B)} = \frac{0.0714}{0.35} = 0.2041 \qquad \frac{P(B)}{P(A \cup B \cup C)} = \frac{0.0714}{0.40} = 0.1786$$

This gives us the percentage of mild/moderate cases testing negative that should go to a quarantine center, shown below

	BC	CT+IC+QC	CT+IC+QC+MSS
Percentage to quarantine centers	0%	20.41%	17.86%

In order to implement this in the model, we will have to create an additional test, as well as an intervention that will serve as a branching point for mild/moderate cases going in different directions.

It's worth noting that we only have to do this procedure for mild/moderate cases, since 100% of asymptomatic cases that present are considered to be contacts, severe/critical cases go to the hospital instead of quarantine, and we handle the susceptible/recovered cases outside of the model.

[20]:
```
# Fraction of mild/moderate going to quarantine centers
#f1 = (1/14)/0.35   # CT+IC+QC
#f2 = (1/14)/0.40   # CT+IC+QC+MSS
#f1 = round(f1,6)   # Round to nearest 1e-6
#f2 = round(f2,6)

# *Sigh*...just make them all 20%

f1=0.20
f2=0.20
```

[21]:
```
# Create test to facilitate moving people around

def create_test_5(x,fraction_to_QC):
    paths,items = get_items(x)
    indices = [i for i,x in enumerate(paths) if 'tests' in paths[i] and 'test␣
 ↪5' in paths[i]]
    t5_paths = [paths[i] for i in indices]
    t5_items = [items[i] for i in indices]

    # This test has a return time of zero days
    # Mild/moderate tests postive at rate corresponding to fraction going to␣
 ↪quarantine centers
    # Everyone else tests postive 100% of the time
```

```
    t5_items[:2] = [0]*2
    t5_items[2:] = [1.0]*(len(t5_items[2:]))
    t5_items[5] = fraction_to_QC

    for i in range(len(t5_items)):
        x = update_item(x,t5_paths[i],t5_items[i])

    return(x)

new_inputs[4] = create_test_5(new_inputs[4],f1) # CT+QC+IC
new_inputs[5] = create_test_5(new_inputs[5],f2) # CT+QC+IC+MSS

# Check that it worked properly
new_inputs[4]['tests']['test 5']
```

[21]:
```
{'result return time': 0,
 'probability of positive result': {'for susceptible': 1.0,
  'for pre-infectious incubation': 1.0,
  'for asymptomatic': 1.0,
  'for mild/moderate': 0.2,
  'for severe': 1.0,
  'for critical': 1.0,
  'for recuperation': 1.0,
  'for recovered': 1.0},
 'delay to test': 0}
```

[22]:
```
# Create the extra intervention we need and insert it between intvs 1 and 3 in␣
↪our model

def create_intervention_8(x):

    paths,items = get_items(x)

    indices = [i for i,x in enumerate(paths) \
               if 'testing strategies' in paths[i] and 'intervention 8' in␣
↪paths[i]]

    i8_paths = [paths[i] for i in indices]
    i8_items = [items[i] for i in indices]

    i8_items = [1]*len(i8_items) # Let's make everything 1 to start
    i8_items[8:10] = [3]*2   # no symptoms or mild/moderate with positive␣
↪result go to QC
    i8_items[10:12] = [5]*2  # severe/critical to hospital
    i8_items[12] = 3         # recuperation/recovered to QC
```

```
    i8_items[14] = 0          # mild/moderate with negative result go to home␣
↪quarantine
    i8_items[18:23] = [0]*5  # No symptoms, severe, critical, and recovered get␣
↪test 0
    i8_items[19] = 5          # Mild/moderate cases get test 5

    # Update values to reflect the test we've created
    for i in range(len(i8_items)):
        x = update_item(x,i8_paths[i],i8_items[i])

    # Reroute those who might go to QC to first pass through intervention 8
    x["testing strategies"]["intervention 1"]\
    ["switch to intervention on negative test result"]["if observed no␣
↪symptoms"] = 8
    x["testing strategies"]["intervention 1"]\
    ["switch to intervention on negative test result"]["if observed mild/
↪moderate"] = 8

    return(x)

new_inputs[4] = create_intervention_8(new_inputs[4]) # CT+QC+IC
new_inputs[5] = create_intervention_8(new_inputs[5]) # CT+QC+IC+MSS
```

[23]:
```
# Now we need to make it so that intervention 8 has the same disease␣
↪progression,
# transmission, mortality, etc. as intervntion 1

def make_eight_into_one(x):
    paths,items = get_items(x)

    # Get indices of parameters corresponding to disease progression for␣
↪interventions 1 and 8

    i1_disease_inds = [i for i,x in enumerate(paths) if 'disease progression'␣
↪in paths[i] and \
            'daily disease progression probability for intervention 1' in␣
↪paths[i]]

    i8_disease_inds = [i for i,x in enumerate(paths) if 'disease progression'␣
↪in paths[i] and \
            'daily disease progression probability for intervention 8' in␣
↪paths[i]]

    i1_mort_inds = [i for i,x in enumerate(paths) if 'disease mortality' in␣
↪paths[i] and \
            'on intervention 1' in paths[i]]
```

```
    i8_mort_inds = [i for i,x in enumerate(paths) if 'disease mortality' in␣
↪paths[i] and \
             'on intervention 8' in paths[i]]

    i1_trans_inds = [i for i,x in enumerate(paths) if 'transmissions' in␣
↪paths[i] and \
             'for intervention 1' in paths[i]]

    i8_trans_inds = [i for i,x in enumerate(paths) if 'transmissions' in␣
↪paths[i] and \
             'for intervention 8' in paths[i]]

    # Combine all those indices so that you get a big list of the parameters␣
↪you need to change
    i1_indices = i1_disease_inds + i1_mort_inds + i1_trans_inds
    i8_indices = i8_disease_inds + i8_mort_inds + i8_trans_inds

    i1_paths = [paths[i] for i in i1_indices]
    i1_items = [items[i] for i in i1_indices]

    i8_paths = [paths[i] for i in i8_indices]
    i8_items = [items[i] for i in i8_indices]

    # Now make it all the parameters associated with intv 8 the same as in intv␣
↪1

    for i in range(len(i8_paths)):
        x = update_item(x,i8_paths[i],i1_items[i])

    return(x)
```

[24]:
```
new_inputs[4] = make_eight_into_one(new_inputs[4]) # CT+QC+IC
new_inputs[5] = make_eight_into_one(new_inputs[5]) # CT+QC+IC+MSS

# Check that it worked
A = new_inputs[4]['disease progression']\
['daily disease progression probability for intervention 1']
B = new_inputs[5]['disease progression']\
['daily disease progression probability for intervention 8']

print(A == B)
```

```
True
```

9 Scaling resources to reflect simulation size

For the final model runs, we will likely run a simulation on the order of 1-2 million patients. However, for debugging purposes, it is helpful to run things with a smaller cohort of 100,000 patients. The code below takes the resource availability from Julia's original input files and scales them to the desired simulation size.

[25]:
```
def scale_resources(x,sim_size,fixed_seed,time_horizon):

    # Make sure we have an integer number of patients
    sim_size = round(sim_size)

    original_size = x['simulation parameters']['cohort size'] # get old size
    x['simulation parameters']['cohort size'] = sim_size      # assign new size
    x['simulation parameters']['fixed seed'] = fixed_seed     # True or false
    x['simulation parameters']['time horizon'] = time_horizon # 365 or 730 days

    paths,items = get_items(x)
    indices = [i for i,x in enumerate(paths) if 'resources' in paths[i] and \
           'resource availabilities' in paths[i]]

    r_paths = [paths[i] for i in indices]
    r_items = [items[i] for i in indices]

    scaled_items = [round(r/original_size*sim_size) for r in r_items]

    for i in range(len(r_paths)):
        x = update_item(x,r_paths[i],scaled_items[i])

    return(x)
```

[26]:
```
# Scale resources to reflect simulation of 1,000,000 people
sim_size = 1000000

for i in range(len(new_inputs)):
    new_inputs[i] = scale_resources(new_inputs[i],sim_size,True,365)

#KZN_pop = 11531628

# S5 - IC and QC to 30k
#nICQC = int(30000/KZN_pop*sim_size)

#for i in range(len(new_inputs)):
#    if new_inputs[i]['resources']['resource availabilities']['resource 3'] !=
 ↪0:
#        new_inputs[i]['resources']['resource availabilities']['resource 3'] =
 ↪nICQC
```

```
#    if new_inputs[i]['resources']['resource availabilities']['resource 4'] !=␣
 ↪0:
#        new_inputs[i]['resources']['resource availabilities']['resource 4'] =␣
 ↪nICQC

# S6 - Hospital beds 22275/11M, ICU beds 371/11M
#nHbed = int(22275/KZN_pop*sim_size)
#nICUbed = int(371/KZN_pop*sim_size)

#for i in range(len(new_inputs)):
#    new_inputs[i]['resources']['resource availabilities']['resource 0'] = nHbed
#    new_inputs[i]['resources']['resource availabilities']['resource 1'] =␣
 ↪nICUbed
#    new_inputs[i]['resources']['resource availabilities']['resource 2'] =␣
 ↪nICUbed


# Check that it worked
new_inputs[5]['resources']['resource availabilities']
```

[26]:
```
{'resource 0': 2384,
 'resource 1': 68,
 'resource 2': 68,
 'resource 3': 1000000,
 'resource 4': 1000000,
 'resource 5': 0,
 'resource 6': 0,
 'resource 7': 0}
```

10 Saving modified input files

In order to ensure reproducibility, I want to create a text document that summarizes the differences between my new input file and the original one used in the manuscript. This will be saved in the output folder alongside it's associated input file.

[27]:
```
def compare_variables(old_x,new_x):
    """
    This function looks for the differences between two input files
    """
    old_paths,old_items = get_items(old_x)
    new_paths,new_items = get_items(new_x)

    # Additional variables not previously in old input file
    added_vars = [i for i in range(len(new_paths)) if new_paths[i] not in␣
 ↪old_paths]
```

```
    # Check if values of parameters common to both input sheets have been␣
↪changed
    common_vars = [i for i in range(len(new_paths)) if new_paths[i] in␣
↪old_paths]

    index_in_old = [j for i in range(len(common_vars)) \
                    for j in range(len(old_paths)) \
                    if new_paths[common_vars[i]] == old_paths[j]]

    changed_vars = [common_vars[i] for i in range(len(common_vars)) \
                    if old_items[index_in_old[i]] != new_items[common_vars[i]]]

    old_item_values = [old_items[index_in_old[i]] for i in␣
↪range(len(common_vars)) \
                    if old_items[index_in_old[i]] != new_items[common_vars[i]]]

    added_paths = [new_paths[added_vars[i]] for i in range(len(added_vars))]
    added_items = [new_items[added_vars[i]] for i in range(len(added_vars))]
    changed_paths = [new_paths[changed_vars[i]] for i in␣
↪range(len(changed_vars))]
    changed_items = [new_items[changed_vars[i]] for i in␣
↪range(len(changed_vars))]

    return(added_paths,changed_paths,added_items,changed_items,old_item_values)
```

[28]:
```
run_names = ['BC',
             'CT',
             'CT+IC',
             'CT+IC+MSS',
             'CT+IC+QC',
             'CT+IC+QC+MSS']

output_path = pwd
output_batch_name = "Testing_NumCheck"
output_batch_path = os.path.join(pwd,output_batch_name)
os.mkdir(output_batch_path)

for i in range(len(run_names)):

    # Get current date and time and convert from UTC to EDT
    now = datetime.datetime.now() + datetime.timedelta(hours=-5)
    current_time = now.strftime("Created on %A, %B %d, %Y at %I:%M %p EDT"\
                                ).lstrip("0").replace(" 0", " ")

    # List differences between old and new input files
    added_paths,changed_paths,added_items,changed_items,old_items = \
    compare_variables(old_inputs[i],new_inputs[i])
```

```
    added_list = ['/'.join(added_paths[k]) + ': --> ' + \
                  str(added_items[k]) + '\n\n' for k in range(len(added_paths))]
    changed_list = ['/'.join(changed_paths[k]) + ': ' + str(old_items[k]) + '␣
↪--> ' + \
                     str(changed_items[k]) + '\n\n' for k in␣
↪range(len(changed_paths))]

    outdir = os.path.join(output_batch_path,run_names[i])
    outJSON = os.path.join(outdir,run_names[i]+'.json')
    outREADME = os.path.join(outdir,run_names[i]+'_README.txt')
    os.mkdir(outdir)

    with open(outJSON, 'w') as f:
        json.dump(new_inputs[i], f, indent=2)
        f.close()

    with open(outREADME, 'w') as f:
        f.write(current_time + '\n\n')
        f.write('Location of Original Input File:\n')
        f.write(old_input_names[i] + '\n\n')
        f.write('Location of Modified Input File:\n')
        f.write(outJSON + '\n\n')
        f.write('\n\n************ Variables Changed ************\n\n\n\n')
        for j in range(len(changed_list)):
            f.write(changed_list[j])
        f.write('\n\n************** Variables Added **************\n\n\n\n')
        for j in range(len(added_list)):
            f.write(added_list[j])
        f.close()
```

[ ]: