Milestone What-if Analysis[*]

Azadeh Kooshesh and Tom Krüger

MSc Informatics at Rostock University

1 Familarization

In the following, we list how we got familiar with the Covasim source code, and what conclusions we draw within the learning process:

1.1 Learning

- Read the Covasim overview https://docs.idmod.org/projects/covasim/en/latest/overview.html
- Installing Covasim 3.0.7 from source
  1. fork Covasim to Tom's GitHub account https://github.com/tomtuamnuq/covasim
  2. Clone repository locally with git
  3. Install from source (pip install -e .)
- Walk through the tutorials T1-T5
  1. T1- get to know Basic Parameters
  2. T2- Covasim's included plotting functionality
  3. T3- Decide to use Multisim and not Scenarios since Multisim seems to be more flexible for testing interventions
  4. T3 - wondering what the bands in MultiSim object plots show and how MultiSim.mean method works
  5. Reading MultiSim docs and source code https://docs.idmod.org/projects/covasim/en/latest/covasim.run.html - the bands show the simulation runs of the MultiSim with respect to the given bounds (which is kind of a confidence interval, but independent of confidence level and number of runs)
  6. T4 - explore people attribute of Simulation objects and the different layers
  7. T4 - decide to use hybrid population generation type because it provides a good balance between accuracy and run-time
  8. T4 - decide to use Germany as location
  9. T5 - run different population sizes and simulation duration with the simple "change beta" intervention
  10. decide to simulate a population of 10.000 people over 60 days since the simulation can take quite some time, especially for 100 replications

[*] Data-Driven Modeling and Simulation Project: Working with an Agent-based Model of COVID-19 dynamics and interventions

- Read Covasim API (https://docs.idmod.org/projects/covasim/en/latest/modules.html)
  1. Base module
  2. Interventions module
  3. Parameters module
  4. Population module
  5. Plotting module
  6. Sim module
- Read T8 because it explains advanced vaccination and virus variant implementations

1.2 Conclusions

The documentation and tutorials for the Covasim project provide a quick introduction to the use of Covasim. The source code is documented well and there exist many examples. It is straightforward to run own sets of interventions or scenarios for small populations. The available plotting functions allow a prompt analysis of the simulations. However, the overall plotting process seems quite complex and it is not easy to plot the main trajectories with 90% confidence interval bands. Using the *mean* method of a *MultiSim* object calculates the statistics by reducing the data of all simulation runs into one simulation. It creates the mean data per day over all runs and saves attributes *low* and *high* per day over all runs with respect to the *bounds* parameter. The calculations are done for all results, i.e., saved simulation data fields in the *results* attribute. The parameter *bounds*=:b is the multiplier on the standard deviation for upper and lower bounds, and the attribute values calculated by the following formula (see lines 276-281 in covasim/run.py):

$$\overline{X}(n) \pm bS(n)$$

Here, $\overline{X}$ is the calculated mean, n the number of runs, and S the standard deviation. By setting bounds to $\frac{t_{n-1,1-\alpha/2}}{\sqrt{n}}$, we get the confidence intervals saved in the attributes *low* and *high*. The value for $t_{n-1,1-\alpha/2}$ can be found in precalculated tables (e.g. https://getcalc.com/statistics-two-tailed-tdistribution-table.htm) or calculated with *scipy.stats.t.interval* function from Scipy (https://www.statology.org/confidence-intervals-python/).
A Jupyter Notebook with some calculations is given in appendix section B.
We implemented the calculations as *helper* functions in module
data_driven/Interventions/what-if-scenarios/helpers/what_if_helpers.py, stored in our gitlab repository. The source code is appended in section A. The function *what_if_helpers.run_simulations* runs a given Simulation a specified number of times. Each run uses a different random seed. Afterwards, $t_{n-1,1-\alpha/2}$ is calculated by using the *t-distribution*. From this, the *mean* method of the *MultiSim* object is called with the *bounds* parameter. The function *run_simulations* returns the reduced simulation so that the results are the calculated means, and

the *low* and *high* attributes are set to the lower and upper bound of the confidence intervals for each day and simulation data result. It uses 100 simulation runs and 90% confidence per default. Thus, calling *plot* on the *MultiSim* object draws the confidence interval bands (see lines 406-407 in *covasim/plotting.py*).

For reference, we have implemented the calculations and plotting without the Covasim *reduce* and *plot* procedures. The functions are *calculate_mean_and_confidence* and *plot_with_bands* in module *what_if_helpers*. They produce the identical data, which is exemplary shown in appendix section B. In our implementation, the pre-calculation of $t_{99,0.9}$ is not necessary, because we use the *interval* function from Scipy's *t-distribution* by setting the parameters *df*, *loc* and *scale* to the number of runs, as well as mean and standard deviation of the given data (see the implementation of *interval* in lines 1401-1431 of https://github.com/scipy/scipy/blob/v1.7.1/scipy/stats/_distn_infrastructure.py). The function *interval* then calculates the confidence interval. *plot_with_bands* calculates the means and confidence interval for a data set of interest, and directly plots the mean trajectory with the confidence band by using *matplotlib* functions *plot* and *fill_between*.

2 Implementation of What-If Scenarios

In this section, we will first describe the common baseline parameters for all scenarios and show some traits of our baseline simulation. Afterward, we present 4 different interventions in comparison to the common baseline simulation. The 4 comparisons show how either the type, strictness, timing or combination of interventions affects the dynamics of the pandemic.

2.1 Baseline

Each of our intervention scenarios uses the following setup:

[1]:
```
from helpers.what_if_helpers import run_base_and_intervention,␣
↪baseline_pars
import covasim as cv
cv.options.set(dpi=100, show=False, close=True, verbose=0)  #␣
↪Standard options for Jupyter notebook
```

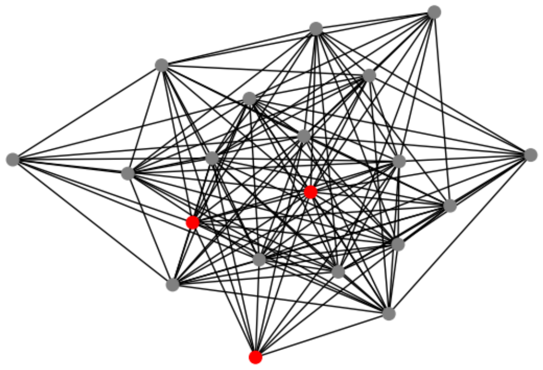
```
Covasim 3.0.7 (2021-06-29) | © 2021 by IDM
```

Herein, the Python dictionary sets the following Covasim parameters, which reflect the current situation in Germany for a subset of 10.000 people:

[1]:
```
# Define baseline parameters
baseline_pars = dict(
    start_day='2022-01-01',
    n_days=60,
    pop_type='hybrid',
    pop_size=10_000,
    pop_infected=int(get_current_infected_ratio() * 10000),
    location='Germany',
    use_waning=True,  # use dynamically calculated immunity
    n_beds_hosp=80,
    n_beds_icu=62,
    variants=[delta_variant],
)
```

The current ratio of infected people in Germany is about 0.78%. The delta variant is considered as the pandemic driver. Details and sources are given in Appendix section A. The population type *hybrid* adds 4 contact layers with respect to certain age distributions to the simulation. The contacts of a small subset of 20 people are shown in the following *NetworkX* graph. Each node represents one person, an edge represents a contact between two people. Each individual has several contacts. We exemplary colored people with contacts in the school layer red.

[2]:
```
import networkx as nx
pars = dict(
    pop_type='hybrid',
    pop_size=20,
    pop_infected=1,
    location='Germany'
)
partial_sim = cv.Sim(pars)
partial_sim.initialize()
G = partial_sim.people.to_graph()
edges = [(i,j) for i,j in G.edges() if i!=j] # remove self loops
school_layer_nodes = set(row['p1'] for _, row in partial_sim.
 ↪people.contacts['s'].to_df().iterrows())
node_colors = ['red' if n in school_layer_nodes else 'grey' for␣
 ↪n in G.nodes()]
simple_G = nx.Graph()
simple_G.add_edges_from(edges)
nx.draw(simple_G,  node_color=node_colors, node_size=70)
```



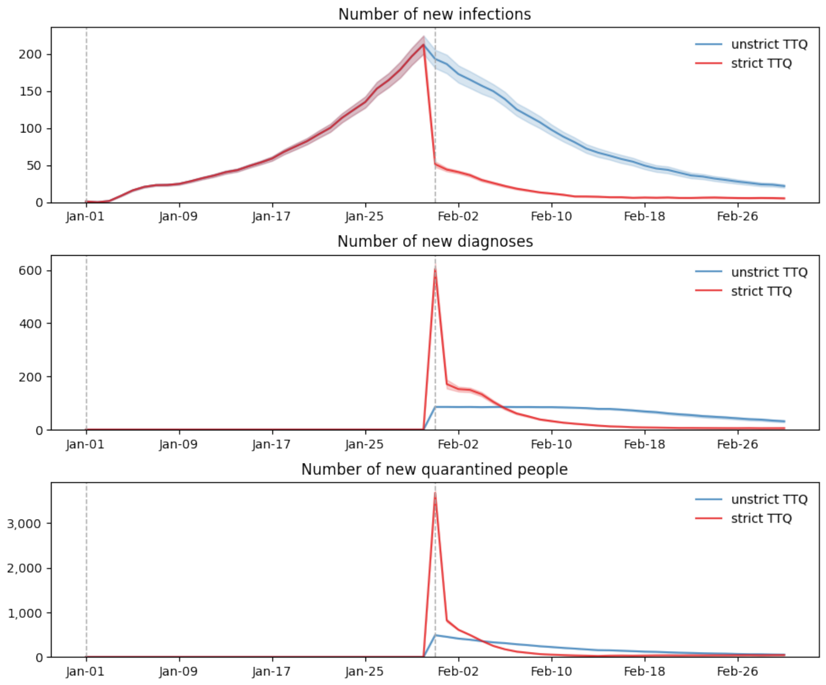
2.2 Strictness of Intervention

We want to show the influence of the strictness of the interventions test-trace-quarantine. An unstrict scenario with a lower number of tests, and lower tracing probabilities, is compared to a strict scenario with people doing a lot of tests and an efficient tracing. The strictness is able to break the dynamics very fast. In the unstrict version, the breaking of the dynamics takes rather long.

[2]:
```
tn_low = cv.test_num(daily_tests=100, start_day=30)
ct_low = cv.contact_tracing(trace_probs=dict(h=0.5, s=0.25, w=0.
 ↪25, c=0.15))
tn_high = cv.test_num(daily_tests=1000, start_day=30) # 10␣
 ↪times more tests
ct_high = cv.contact_tracing(trace_probs=dict(h=1.0, s=0.5, w=0.
 ↪5, c=0.3)) # double probabilities

unstrict_sim = cv.Sim(baseline_pars, interventions=[tn_low,␣
 ↪ct_low], label='unstrict TTQ')
strict_sim = cv.Sim(baseline_pars, interventions=[tn_high,␣
 ↪ct_high], label='strict TTQ')
multi_sim = run_base_and_intervention(unstrict_sim, strict_sim)
multi_sim.plot(to_plot=['new_infections', 'new_diagnoses',␣
 ↪'new_quarantined'])
```

[2]:



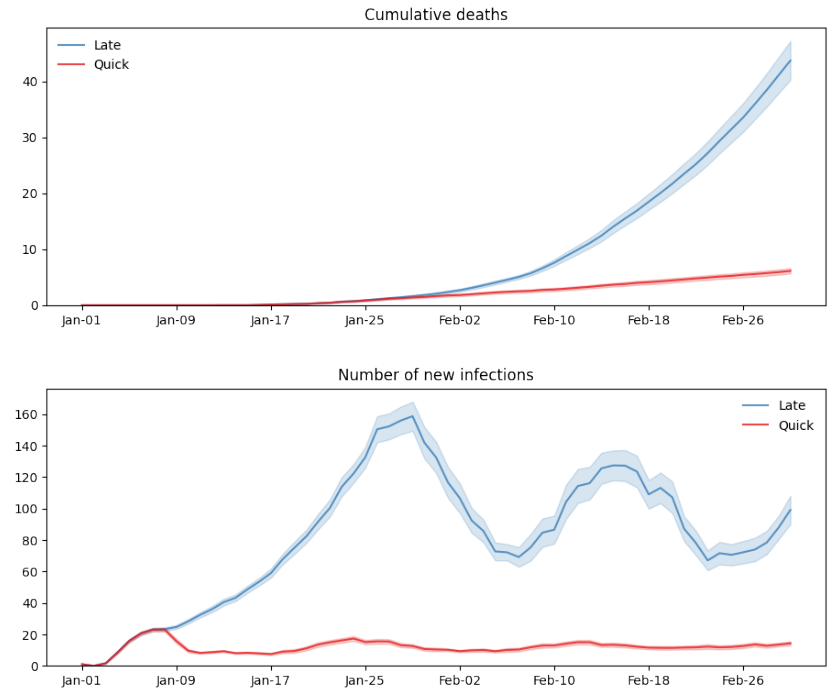
2.3 Timing of Intervention

To investigate the influence of the correct timing we used the contact constraint (lockdown). Contact restriction is simulated by reducing contacts in all layers by 75%. The measure is introduced dynamically when a certain number of infected people is reached. The threshold implies the timing. In the "Late" scenario the intervention is applied when there are 1000 infections. This is too late since many more people die. A timely lockdown causes a sustained decline in the number of infected persons.

[2]:
```
from helpers.what_if_helpers import␣
 ↪init_intervention_for_inf_thresh, inf_thresh_callback
changes = [1 / 4, 1.0]  # remove 75 % of contacts
ce_late = cv.clip_edges(days=inf_thresh_callback(1000),␣
 ↪changes=changes)
ce_quick = cv.clip_edges(days=inf_thresh_callback(100),␣
 ↪changes=changes)
init_intervention_for_inf_thresh(ce_late)
init_intervention_for_inf_thresh(ce_quick)
```

```
base_sim = cv.Sim(baseline_pars, interventions=ce_late,␣
↪label='Late')  # late lockdown
lockdown_sim = cv.Sim(baseline_pars, interventions=ce_quick,␣
↪label='Quick')
multi_sim = run_base_and_intervention(base_sim, lockdown_sim)
multi_sim.plot(to_plot=["cum_deaths", "new_infections"])
```

[2]:



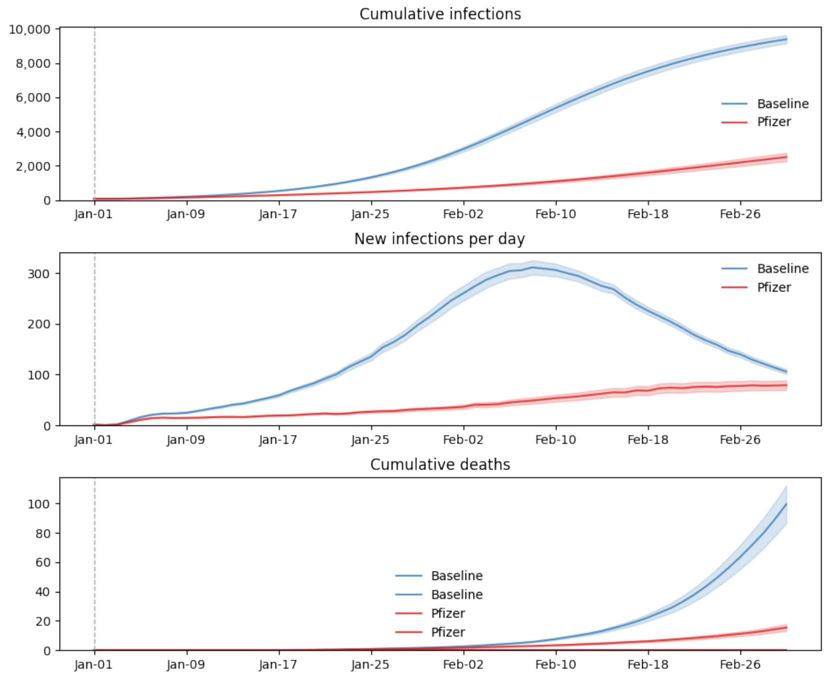
2.4 Type and Combination of Interventions

The vaccination of the population is considered the most important intervention against the pandemic. We compared what happens without any interventions against a scenario with half of the people vaccinated.

[2]:
```
delta_sim = cv.Sim(baseline_pars, label='Baseline') # no␣
↪vaccination
pfizer = cv.vaccinate_prob(vaccine='pfizer', days=0, prob=0.5)␣
↪# 50% of the population is vaccinated
pfizer_sim = cv.Sim(baseline_pars, interventions=pfizer,␣
↪label='Pfizer')
```

```
multi_sim = run_base_and_intervention(delta_sim, pfizer_sim)
multi_sim.plot()
```

[2]:

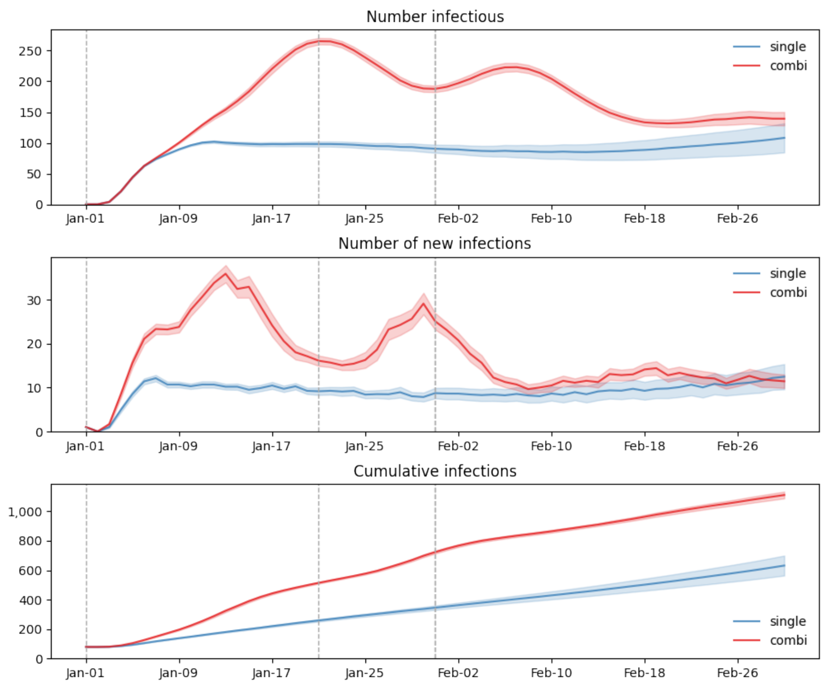


The intervention type vaccination has a huge impact on the dynamics of the pandemic. Without it, everyone gets infected within two months. To achieve a similar effect with interventions such as TTQ or contact reduction alone, these must be implemented and maintained very strictly. This is only possible to a limited extent due to the social and economic impact. However, for a limited amount of time a combination of the interventions TTQ, contact reduction, together with a beginning vaccination, can break the rising number of new infections. We compared two scenarios in this regard. The first reflects the current vaccination ratio and rate in Germany without further measures. The second is based on the beginning of vaccinations. There are still few people vaccinated, but more vaccinated people are being added every day. There are ongoing contact reductions when certain infection numbers are reached. A satisfactory test and quarantine infrastructure are being established (here on day 30).

[2]:
```
from helpers.what_if_helpers import␣
 ↪init_intervention_for_inf_thresh, inf_thresh_callback
pfizer_high = cv.vaccinate_prob(vaccine='pfizer', days=0,␣
 ↪prob=0.682)  # https://impfdashboard.de/
```

```
pfizer_low = cv.vaccinate_prob(vaccine='pfizer', days=[i for i␣
 ↪in range(baseline_pars['n_days'])], prob=1 / 180, ␣
 ↪do_plot=False)
single_sim = cv.Sim(baseline_pars, interventions=[pfizer_low,␣
 ↪pfizer_high], label='single')
tn_low = cv.test_num(daily_tests=50, start_day=30)
ct_low = cv.contact_tracing(trace_probs=dict(h=0.5, s=0.25, w=0.
 ↪25, c=0.15))
ce = cv.clip_edges(days=inf_thresh_callback(200), changes=[1 /␣
 ↪4, 1.0])
init_intervention_for_inf_thresh(ce)
combi_sim = cv.Sim(baseline_pars, interventions=[pfizer_low,␣
 ↪ce, tn_low, ct_low], label='combi')
multi_sim = run_base_and_intervention(single_sim, combi_sim)
multi_sim.plot(to_plot=["n_infectious", "new_infections",␣
 ↪"cum_infections"])
```

[2]:



The dynamically triggered contact reduction leads to a noticeable drop in infection chains. This is visible around the second vertical dot-line. The start of TTQ on the day 30 (third vertical line) has a lasting effect on the dynamic. They are leading to a decline in new infections and control of the pandemic, which is comparable to the effect of vaccinations alone. However, the total number of infected (and damaged/dead) people is higher.

A Helper Functions

[ ]:
```
from functools import partial
from typing import Tuple

import covasim as cv
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st


def get_current_infected_ratio():
    # Returns the current ratio of infected people in germany
    number_infected = 651500  # https://www.deutschland.de/de/
 ↪topic/politik/corona-in-deutschland-zahlen-und-fakten
    number_total = 83100000  # https://www.destatis.de/DE/
 ↪Themen/Gesellschaft-Umwelt/Bevoelkerung/Bevoelkerungsstand/
 ↪_inhalt.html
    infected_ratio = number_infected / number_total
    return infected_ratio


delta_variant = cv.variant('delta', days=0)  # delta is the␣
 ↪dominant variant in germany

# Define baseline parameters
baseline_pars = dict(
    start_day='2022-01-01',
    n_days=60,
    pop_type='hybrid',
    pop_size=10_000,
    pop_infected=int(get_current_infected_ratio() * 10000),
    location='Germany',
    use_waning=True,  # use dynamically calculated immunity
    n_beds_hosp=80,  # https://tradingeconomics.com/germany/
 ↪hospital-beds - 8 per 1000 people
    n_beds_icu=62,  # https://tradingeconomics.com/germany/
 ↪icu-beds - 620 per 100.000 people
    variants=[delta_variant],
)
```

```python
def run_simulations(sim: cv.Sim, n_runs: int, confidence_level:␣
↪float, method: str = "t") -> cv.MultiSim:
    msim = cv.MultiSim(sim)
    msim.run(n_runs=n_runs)
    if method == "t":  # use t-distribution
        bounds = st.t.interval(alpha=confidence_level,␣
↪df=n_runs - 1)[1]
    else:  # use normal distribution
        bounds = st.norm.interval(alpha=confidence_level)[1]
    bounds = bounds / np.sqrt(n_runs)
    msim.mean(bounds=bounds)
    return msim


def run_base_and_intervention(base_sim: cv.Sim,␣
↪intervention_sim: cv.Sim, n_runs: int = 100,
                              confidence_level: float = 0.9) ->␣
↪cv.MultiSim:
    base_msim = run_simulations(base_sim, n_runs,␣
↪confidence_level)
    intervention_msim = run_simulations(intervention_sim,␣
↪n_runs, confidence_level)
    return cv.MultiSim([base_msim.base_sim, intervention_msim.
↪base_sim])


# calculate by hand for reference

def calculate_mean_and_confidence(msim: cv.MultiSim, result_key:
↪ str, method: str = "t",
                                  confidence_level: float = 0.
↪9) -> Tuple[np.array, np.array, np.array]:
    data = np.array([s.results[result_key] for s in msim.sims],␣
↪dtype=float)
    data_mean = np.mean(data, axis=0)
    data_sem = st.sem(data, axis=0)
    if method == "t":
        conf_intervals = st.t.interval(alpha=confidence_level,␣
↪df=data.shape[0] - 1, loc=data_mean, scale=data_sem)
    else:
        conf_intervals = st.norm.
↪interval(alpha=confidence_level, loc=data_mean,␣
↪scale=data_sem)
    lower_band, upper_band = conf_intervals
    return data_mean, lower_band, upper_band


# plot by hand for reference

def plot_with_bands(base_msim: cv.MultiSim, intervention_msim:␣
↪cv.MultiSim, result_key: str, ax=None,
                    colors_base=("b", "c"),␣
↪colors_intervention=("r", "tab:orange"), show_dates=False):
    if ax is None:
        _, ax = plt.subplots()
        ax.set_title(result_key)
    if show_dates:
        x = base_msim.results['date']
    else:
        x = base_msim.results['t']
    for sim, c in ((base_msim, colors_base),␣
↪(intervention_msim, colors_intervention)):
        data_mean, lower_band, upper_band =␣
↪calculate_mean_and_confidence(sim, result_key)
        ax.fill_between(x, lower_band, upper_band, alpha=.75,␣
↪linewidth=0, label=f"{sim.label} band", color=c[1])
        ax.plot(x, data_mean, label=sim.label, color=c[0])
    if show_dates:
        cv.date_formatter(sim=base_msim.base_sim, ax=ax)
    else:
        # show intervention as vertical line
        for intervention in intervention_msim.base_sim.
↪get_interventions():
            intervention.plot_intervention(intervention_msim.
↪base_sim, ax)
    ax.legend()
    return ax

def _inf_thresh(self: cv.Intervention, sim: cv.Sim, thresh:␣
↪int):
    ''' Dynamically define on and off days with respect to the␣
↪number of infected people.
    See https://docs.idmod.org/projects/covasim/en/latest/
↪tutorials/tut_interventions.html#Dynamic-triggering'''
```

```
    if sim.people.infectious.sum() > thresh:
        if not self.active:
            self.active = True
            self.t_on = sim.t
            self.plot_days.append(self.t_on)
    else:
        if self.active:
            self.active = False
            self.t_off = sim.t
            self.plot_days.append(self.t_off)

    return [self.t_on, self.t_off]

def inf_thresh_callback(thresh: int = 500):
    return partial(_inf_thresh, thresh=thresh)

def init_intervention_for_inf_thresh(c: cv.Intervention):
    """Setup attributes for `inf_thresh_callback`"""
    c.t_on = np.nan
    c.t_off = np.nan
    c.active = False
    c.plot_days = []
    return c
```

B Testing Calculations and Plots for Reference

[1]:
```
import warnings
warnings.simplefilter("ignore")
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st
import covasim as cv
cv.options.set(dpi=100, show=False, close=True, verbose=0)
from what_if_helpers import run_simulations, plot_with_bands,␣
 ↪calculate_mean_and_confidence
```

```
Covasim 3.0.7 (2021-06-29) | © 2021 by IDM
```

[2]:
```
alpha = 0.9  # two tail 0.9, for one tail use 0.8
num_runs = 100
mean = 0.0
std = 1.0
# https://www.statology.org/confidence-intervals-python/
```

```
interval = st.norm.interval(alpha=alpha)
print(f"The norm interval is {interval}")
interval = st.t.interval(alpha=alpha, df=num_runs - 1)
print(f"The t interval is {interval}")

# from table https://getcalc.com/
 ↪statistics-two-tailed-tdistribution-table.htm
t = 1.66  # for two tail use 1.66, one tail 1.29
# for covasim https://docs.idmod.org/projects/covasim/en/latest/
 ↪_modules/covasim/run.html#MultiSim.reduce
# reduce method
bounds = t  #  / np.sqrt(num_runs)
interval_manual = (mean - bounds * std, mean + bounds * std)
print(f"Interval by hand {interval_manual}")
```

```
The norm interval is (-1.6448536269514729, 1.6448536269514722)
The t interval is (-1.6603911559963902, 1.6603911559963895)
Interval by hand (-1.66, 1.66)
```

[3]:
```
pars = dict(
    start_day=0,
    n_days=90,
    pop_type='hybrid',
    pop_size=20_000,
    pop_infected=100,
    location='Germany'
)

base_sim = cv.Sim(pars, label='Baseline')
cb = cv.change_beta(days=[15, 30, 45, 50], changes=[1.0, 0.7, 0.
 ↪3, 0.1])
sim = cv.Sim(pars, interventions=cb, label='With beta changes')
base_msim = run_simulations(base_sim, num_runs, alpha)
msim = run_simulations(sim, num_runs, alpha)
```

C Calculate and plot without Covasim for reference

[4]:
```
# test calculations on the cumulated number of deaths
result_key = 'cum_deaths'
data = np.array([s.results[result_key] for s in msim.sims],␣
 ↪dtype=float)
data_mean = np.mean(data, axis=0)
data_sem = st.sem(data, axis=0)
```

```
t_conf_intervals = st.t.interval(alpha=alpha, df=data.shape[0]␣
↪- 1, loc=data_mean, scale=data_sem)
normal_conf_intervals = st.norm.interval(alpha=alpha,␣
↪loc=data_mean, scale=data_sem)
max_diff = max(abs(t_i-n_i) for t_i,n_i in␣
↪zip(t_conf_intervals[0], normal_conf_intervals[0]) if all(np.
↪isfinite((t_i,n_i))))
f"Maximum difference between t and normal interval left is␣
↪{max_diff}"
```

```
[4]: 'Maximum difference between t and normal interval left is 0.
↪022898136993489997'
```

```
[5]: msim.results[result_key].values[20:30]
```

```
[5]: array([0.38, 0.49, 0.71, 0.9 , 1.09, 1.26, 1.56, 1.79, 2.07, 2.
↪49])
```

```
[6]: data_mean, lower_band, upper_band =␣
↪calculate_mean_and_confidence(msim, result_key)
data_mean[20:30]
```

```
[6]: array([0.38, 0.49, 0.71, 0.9 , 1.09, 1.26, 1.56, 1.79, 2.07, 2.
↪49])
```

```
[7]: msim.results[result_key].low[50:70]
```

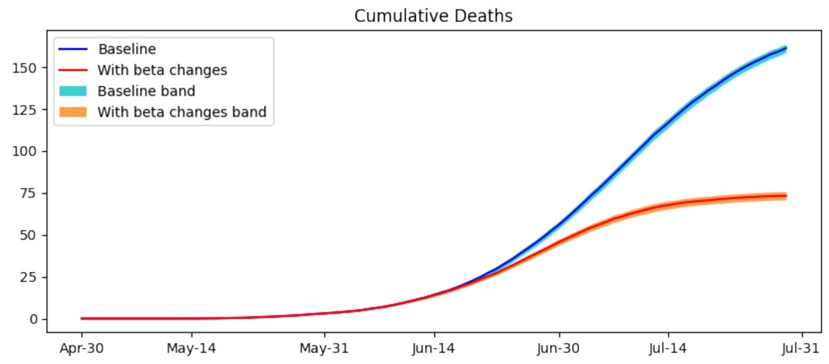
```
[7]: array([20.50689655, 22.36648348, 24.09742822, 25.88461805, 28.
↪08617791,
       30.20832213, 32.44423111, 34.6739754 , 36.96681748, 39.
↪14772451,
       41.47310121, 43.89774507, 45.94111443, 48.08928805, 50.
↪092285  ,
       52.08553926, 53.78364951, 55.46829962, 57.40343505, 58.
↪6283774 ])
```

```
[8]: lower_band[50:70]
```

```
[8]: array([20.50133931, 22.36052114, 24.09116837, 25.8779914 , 28.
↪07930723,
       30.20115998, 32.43659494, 34.66593492, 36.95853943, 39.
↪139048  ,
       41.46430066, 43.8887664 , 45.93195122, 48.07986374, 50.
↪08227126,
       52.07523964, 53.77293735, 55.45745975, 57.39241954, 58.
↪61718527])
```

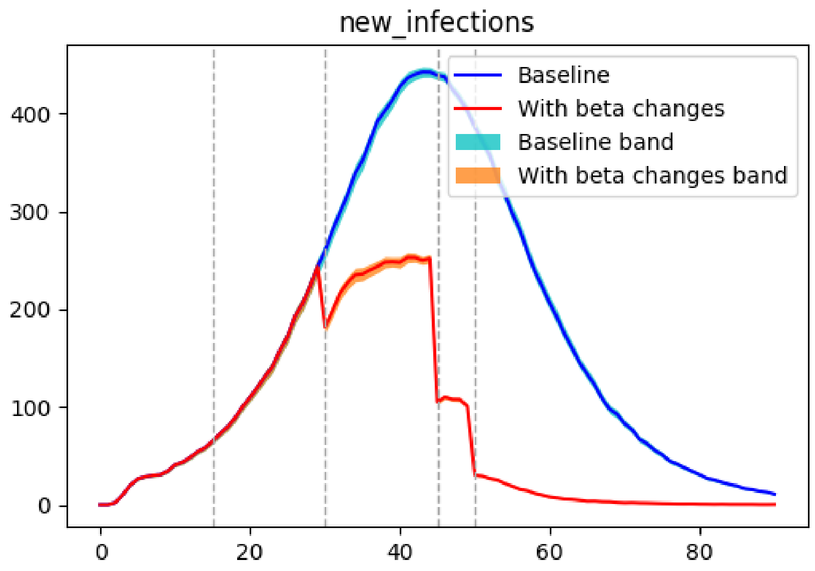
[9]:
```
fig, ax = plt.subplots(figsize=(10, 4))
ax.set_title("Cumulative Deaths")
plot_with_bands(base_msim, msim, result_key, ax,␣
 ↪show_dates=True)
```

[9]:
```
<AxesSubplot:title={'center':'Cumulative Deaths'}>
```



[10]:
```
plot_with_bands(base_msim, msim, "new_infections")
```

[10]:
```
<AxesSubplot:title={'center':'new_infections'}>
```


new_infections
Baseline
With beta changes
Baseline band
With beta changes band
0
100
200
300
400
0
20
40
60
80