Milestone Sensitivity Analysis and Experiment Design*

Azadeh Kooshesh and Tom Krüger

MSc Informatics at Rostock University

1 Exercise

Since its first appearance, the SARS Coronavirus type 2 has developed several variants due to changes/mutations in the virus's genes. Some of these mutations greatly alter the properties of the virus and change the dynamics of the pandemic, e.g., due to an increased transmission or higher lethality (more severe disease). Apply a global sensitivity analysis (SA) using the Method of Morris with at least 4 model parameters and quantify their impact on the model outcome ("number of death"). In your analysis include at least one model parameter, from which you expect a small impact and at least one parameter, from which you expect a strong impact on the mortality. What is worse - a mutation yielding a virus variant with a higher lethality or a higher contagiousness? Repeat the analysis using another SA method that is based on a Latin Hypercube sample, and present and interpret the results. For your analysis you can use one of the available sensitivity analysis libraries in Python, such as SALib. Remember to run multiple replications per design point as the simulation is stochastic, and document your code carefully!

2 Implementation Background

2.1 SALib

SALib contains Python implementations of commonly used sensitivity analysis methods, including Sobol, Morris, and FAST. It is useful in systems modeling to calculate the effects of model inputs or exogenous factors on outputs of interest [4].

Prerequisites

1. Numpy library
2. Scipy library
3. Matplotlib library

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

SALib installation process

1. pip install SALib

2.2 Covasim parameters

We applied three different sensitivity analysis methods to the Covasim simulation with the help of SALib. As Covasim parameters to analyze we have chosen the following [2]:

1. Beta β: Basic disease transmission per symptomatic contact
2. asymp_factor: Multiplication factor for Beta for asymptomatic cases
3. rel_severe_prob: Scale factor for proportion of symptomatic cases that become severe
4. real_death_prob: Scale factor for proportion of critical cases that result in death

Beta sets the contagiousness of the virus. It should obviously highly influence the dynamics of the pandemic. The parameter should therefore significantly influence the number of people who die. The Covasim documentation links to a paper which suggests that the relative transmissibility of asymptomatic cases could be significantly smaller than that of the symptomatic cases [3]. We want to know the influence of different relative transmissibilities on the number of people who die but believe that the linear influence is rather low. Nevertheless, there should be some non-linear or interaction effects because people without symptoms can infect other people. The scaling factors for the probabilities for individuals to transfer from the Mild to the Severe or from Severe to Death state should have an influence because more or fewer people transfer to the Death state after they have been infected. Different settings of these two parameters map different levels of dangerousness (lethality) of the virus.

2.3 Helper functions

To make our implementation reusable we created a few helper functions which can be found in section A. The function *run_design_point* takes one configuration of Covasim parameters, runs the simulation a given number of times, and returns the mean of the accumulated results of a specified output of interest. The function *run_simulations* is the one from the previous milestone. We also reused our Covasim baseline parameters. For our tests we always set the output of interest as the number of people died. For a given configuration of Covasim parameters the simulation was replicated 10 times. For each replication the total number of people died was taken at the end of the simulation. The mean over these 10 replications was used as output for the given configuration.

To make our defaults and baseline easy to use, and apply *run_design_point* to an ordered list of parameter configurations, we created the function *design_point_executor*. It binds a function to a given list of Covasim parameter configurations and parameters of *run_design_point*. The returned function is then callable with values of the Covasim parameters of interest.

3 Global sensitivity analysis with Method of Morris

Sensitivity analysis (SA) is the study of how a variety of uncertain input parameters impacts the uncertainty of the response of interest. Many screening techniques exist for SA and the most widely used are:

1. One-at-a-time(OAT)
2. Morris method

In this section, the focus is on the Morris method which is based on a replication set of randomized OAT designs. The method calculates the elemental effects EE_n:

$$EE_n = \frac{f(x_1, \ldots, x_n + \Delta_n, \ldots, x_K) - f(x_1, \ldots, x_{n-1}, x_n, \ldots, x_K)}{\Delta_n}$$

Elementary effects are evaluated by perturbing the n-th parameter of a point $x = (x_1, \ldots, x_K) \in \Omega$ by Δ_n. In addition, to avoid the dependency on the initial point x, the EE_n are computed for R trajectories. Ω is the region of experimentation and Δ_n is a value which comes from the partitioning of the model parameter space into a uniform grid of points (with a given number of levels) [7].

3.1 Calculate absolute value of mean and standard deviation in Morris Method

$\mu_n = \frac{1}{R} \sum_{l=1}^{R} |EE_n^l|$

1. The mean μ_n assesses the overall impact of the parameter on the response.
2. A large mean indicates large changes in the response when perturbing a parameter.

$\sigma_n^2 = \frac{1}{R-1} \sum_{l=1}^{R} (EE_n^l - \mu_n)^2$

1. The standard deviation σ_n is a measure of non-linear and interaction effects.
2. A large value indicates that the elementary effects depend highly on the choice of the sample point at which it is computed.
3. A small σ_n indicates similar values in an elementary effect for different sample points, implying that the effect is independent of other parameter values.

3.2 Applying the Method of Morris in Python

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from SALib.analyze.morris import analyze as morris_analyze
from SALib.sample.morris import sample as morris_sample
from helpers.helpers import design_point_executor
```

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

[2]:
```
r = 10  # number of replications with 4<=r<=10
num_levels = 6  # parameter for method of morris
par_list = ['beta', 'asymp_factor', 'rel_severe_prob',␣
 ↪'rel_death_prob']
k = len(par_list)  # number of covasim variables to analyse
problem = {
    'num_vars': k,
    'names': par_list,
    'bounds': [[0.005, 0.005 * num_levels],
               [0.5, 0.5 * num_levels],
               [0.75, 0.75 * num_levels],
               [0.75, 0.75 * num_levels]]
}
```

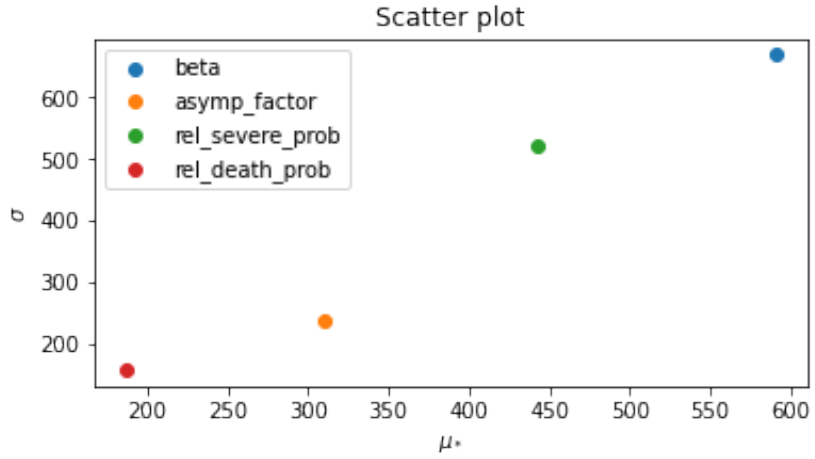
[3]:
```
# sample the r trajectories
X = morris_sample(problem, r, num_levels=num_levels)
print(f"Created {len(X)} design points")
print("First design point: ", X[0], "\n Last design point: ",␣
 ↪X[-1])
executor = design_point_executor(par_list, n_reps=10)
results = map(executor, X)
# execute the simulations
Y = np.fromiter(results, dtype=np.float64)
print("Average number of dead people over 10 simulation runs:")
print("For first design point:", Y[0])
print("For last design point:", Y[-1])
```

```
Created 50 design points
First design point:  [0.02 2.   1.5  3.75]
 Last design point:  [0.025 1.    3.    1.5  ]
Average number of dead people over 10 simulation runs:
For first design point: 425.9
For last design point: 584.1
```

[4]:
```
# calculate elementary effects
Si = morris_analyze(problem, X, Y, num_levels=num_levels,␣
 ↪print_to_console=True)
```

	mu	mu_star	sigma	mu_star_conf
beta	590.433333	590.433333	670.721972	398.015672
asymp_factor	310.450000	310.450000	237.902058	139.851188
rel_severe_prob	443.050000	443.050000	521.698923	288.361945
rel_death_prob	187.233333	187.233333	155.356047	89.839352

[5]:
```
# plot first against higher order elementary effects
mu_star = Si['mu_star'].data
sigma = Si['sigma']
plt.figure(figsize=(6, 3))
plt.title("Scatter plot")
for par, m, s in zip(par_list, mu_star, sigma):
    plt.scatter(m, s, label=par)
plt.xlabel(r'$\mu_*$')
plt.ylabel(r'$\sigma$')
plt.legend();
```



4 Global SA with RBD-Fast and Delta

4.1 RBD-Fast

RBD-Fast is an abbreviation for Random Balance Designs Fourier Amplitude Sensitivity Test. This method is known to be very robust for the computation of global sensitivity indices but their computational cost remains prohibitive for complex and large dimensional models. Developments in the implementation of FAST by use of the random balance designs (RBD) technique have allowed significant reduction of the computational cost. The drawback of this improvement is that only individual first-order sensitivity indices can be computed [6].

4.2 Delta

The Delta method uses a global SA indicator δ that considers the entire distribution both of the input and of the output (global) in a moment independent fashion [1, Fig. 1]. For each parameter i the method derives a global importance measure δ_i. Even though the paper [1] also defines δ_{ij} to allow statements about the influence between parameters, the implementation in SALib only gives the global influence δ_i.

4.3 Applying RBD-Fast and Delta in Python

[1]:
```
import matplotlib.pyplot as plt
import numpy as np
from SALib.analyze.delta import analyze as delta_analyze
from SALib.analyze.rbd_fast import analyze as rbd_fast_analyze
from SALib.sample.latin import sample as latin_sample

from helpers.helpers import design_point_executor
```

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

[2]:
```
num_samples = 100  # Latin Hypercube samples
par_list = ['beta', 'asymp_factor', 'rel_severe_prob',␣
 ↪'rel_death_prob']
k = len(par_list)  # number of covasim variables to analyse
problem = {
    'num_vars': k,
    'names': par_list,
    'bounds': [[0.005, 0.005 * 6],
               [0.5, 0.5 * 6],
               [0.75, 0.75 * 6],
               [0.75, 0.75 * 6]]
}
```

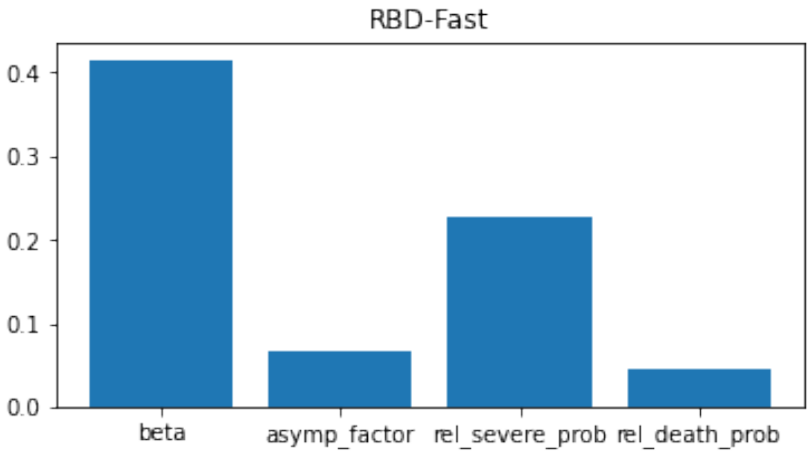
[3]:
```
# sample design points
X = latin_sample(problem, num_samples)
print(f"Created {len(X)} design points")
print("First design point: ", X[0], "\n Last design point: ",␣
 ↪X[-1])
executor = design_point_executor(par_list)
results = map(executor, X)
# execute the simulations
Y = np.fromiter(results, dtype=np.float64)
print("Average number of dead people over 10 simulation runs:")
print("For first design point:", Y[0], "\n Last design point:",␣
 ↪Y[-1])
```

```
Created 100 design points
First design point:  [0.00938976 2.54279189 1.63089604 0.9255386 ]
 Last design point:  [0.00674854 1.40885867 4.29869811 3.252052  ]
Average number of dead people over 10 simulation runs:
For first design point: 72.3
 Last design point: 49.5
```

[4]:
```
# apply RBD-Fast
Si = rbd_fast_analyze(problem, X, Y, print_to_console=True)
```

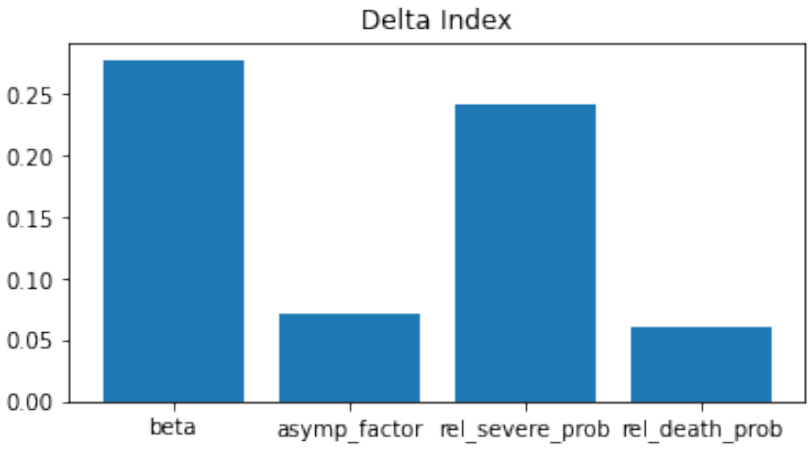
```
# plot first order effects of the parameters
plt.figure(figsize=(6, 3))
plt.title("RBD-Fast")
plt.bar(par_list, Si['S1']);
```

	S1	S1_conf
beta	0.415621	0.290685
asymp_factor	0.066794	0.335682
rel_severe_prob	0.226657	0.336032
rel_death_prob	0.045249	0.351219



[5]:
```
# apply Delta
Si = delta_analyze(problem, X, Y, print_to_console=True)
# plot Delta Moment-Independent Measure of parameters
plt.figure(figsize=(6, 3))
plt.title("Delta Index")
plt.bar(par_list, Si['delta']);
```

	delta	delta_conf	S1	S1_conf
beta	0.278506	0.071730	0.474535	0.152394
asymp_factor	0.072211	0.048075	0.071139	0.119462
rel_severe_prob	0.241161	0.055041	0.315106	0.129124
rel_death_prob	0.061120	0.050009	0.017516	0.082873


Delta Index
0.25
0.20
0.15
0.10
0.05
0.00
beta
asymp_factor
rel_severe_prob
rel_death_prob

5 Interpretation

The mean elementary effects from the method of Morris show the first order impact of each parameter on the number of people who died. The RBD-Fast implementation in SALib provides first-order sensitivity indices. As the Morris Index, they show the linear impact of each parameter. The moment independent Delta indices show the global importance of each parameter to the output of interest.

Figure 1 shows the indices in a normalized form for better comparison. In all methods, the transmissibility of the virus has the greatest influence on the response. The relative factor for transferring to the death state has the lowest influence. The relative factor for transferring to the severe state has a much higher influence. By comparing the influence of β with the relative factors for transferring to the severe and death states we infer that a virus variant with higher contagiousness is worse than a variant with a higher lethality.

Even though we thought the asymptomatic factor would not have a great influence at least the Morris index indicates that it has an influence that should not be underestimated.

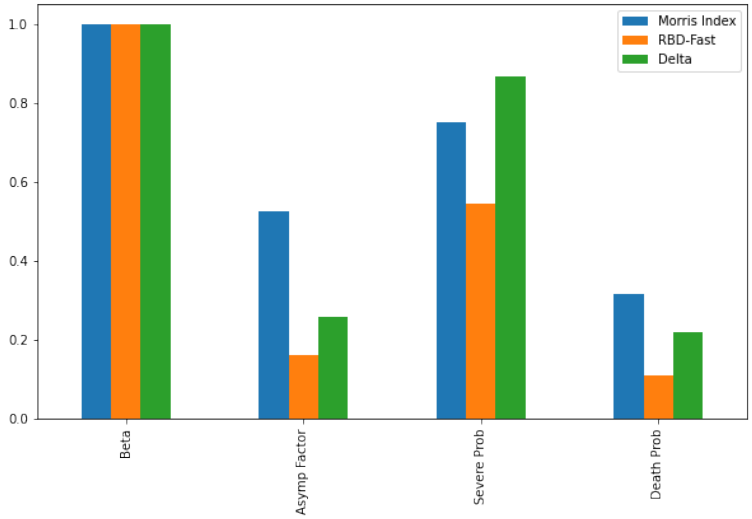


Fig. 1. Matplotlib bar plot of the normalized indices for the three applied sensitivity analysis methods.

The scatter plot from the Method of Morris shows non-linear and interaction effects of the parameters. The high standard deviation σ of the elementary effects for β indicates that the parameter has non-linear effects and interacts with other parameters. This makes sense because Dead is the last state in the Covasim

model and β influences the transition from the Exposed state to the Presymptomatic state [5, Fig. 1]. On the other hand, transmissibility also influences the number of infected people without symptoms. Another very important factor for the probability of death is the age of the individual. The contagiousness β has a high impact on the dynamics of the pandemic which includes the number of deaths. The scale factor for the proportion of symptomatic cases that become severe also shows non-linear and interaction effects. This can be explained similarly because an individual transfers from Mild to Death state over Severe and Critical. Therefore the relationship is influenced by other parameters, but not as strong as the relationship between β and the number of death. The low σ for the scale factor for proportion of critical cases that result in death suggests a linear relationship to the response. This is clear because it directly influences the number of people that transfer from Critical to the Death state.

A Helper Functions

[ ]:
```
from typing import List
import covasim as cv
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

# Define baseline parameters
baseline_pars = dict(
    verbose=0,
    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
)

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_design_point(pars: dict, output_of_interest: str,␣
↪n_runs: int, confidence_level: float = 0.9) -> float:
    """Runs one design point n_runs times and gets the output␣
↪of interest on the last simulation point in time."""
    msim = run_simulations(cv.Sim(pars), n_runs,␣
↪confidence_level)
    return msim.results[output_of_interest].values[-1]

def design_point_executor(par_list: List[str],␣
↪output_of_interest: str = "cum_deaths", n_reps: int = 10) ->␣
↪callable:
    """Creates a callable function that executes a design point.
↪"""
    def get_result_to_design_point(design_point: np.ndarray) ->␣
↪float:
        design_pars = dict(zip(par_list, design_point))
        pars = {**baseline_pars, **design_pars}
        return run_design_point(pars, output_of_interest,␣
↪n_reps)

    return get_result_to_design_point
```

References

1. Borgonovo, E.: A new uncertainty importance measure. Reliability Engineering and System Safety **92**(6), 771–784 (2007). https://doi.org/https://doi.org/10.1016/j.ress.2006.04.015, https://www.sciencedirect.com/science/article/pii/S0951832006000883
2. Foundation, B..M.G.: parameters. https://docs.idmod.org/projects/covasim/en/latest/parameters.html (2021)
3. He, D., Zhao, S., Lin, Q., Zhuang, Z., Cao, P., Wang, M.H., Yang, L.: The relative transmissibility of asymptomatic covid-19 infections among close contacts. International Journal of Infectious Diseases **94**, 145–147 (2020). https://doi.org/https://doi.org/10.1016/j.ijid.2020.04.034, https://www.sciencedirect.com/science/article/pii/S1201971220302502
4. Jon Herman, W.U.: Salib: An open-source python library for sensitivity analysis. JOSS (2017). https://doi.org/10.21105/joss.00097, https://joss.theoj.org/papers/10.21105/joss.00097
5. Kerr, C.C., Stuart, R.M., Mistry, D., Abeysuriya, R.G., Rosenfeld, K., Hart, G.R., Nunez, R.C., Cohen, J.A., Selvaraj, P., Hagedorn, B., George, L., Jastrzebski, M., Izzo, A., Fowler, G., Palmer, A., Delport, D., Scott, N., Kelly, S., Bennette, C.S., Wagner, B., Chang, S., Oron, A.P., Wenger, E., Panovska-Griffiths, J., Famulare, M., Klein, D.J.: Covasim: an agent-based model of covid-19 dynamics and interventions. medRxiv (2021). https://doi.org/10.1101/2020.05.10.20097469, https://www.medrxiv.org/content/early/2021/04/01/2020.05.10.20097469
6. Mara, T.A.: Extension of the rbd-fast method to the computation of global sensitivity indices. HAL (2014). https://doi.org/hal-01093036, https://hal.archives-ouvertes.fr/hal-01093036
7. Wicaksono, D.: gsa-module. https://gsa-module.readthedocs.io/en/stable/implementation/morris_screening_method.html (2016)