CODE Toolbox

Technical Software Manual

February 2026

Contents

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

1 Overview

The **CODE Toolbox** is a software platform designed to support numerical experiments on complex social systems involving structured populations, epidemic spreading, and opinion dynamics on networks.

The toolbox integrates several computational components that allow the user to:

- generate synthetic networks with community structure,
- simulate epidemic spreading processes on heterogeneous populations,
- study the evolution of opinions influenced by social interactions and epidemic risk,
- interactively explore model parameters through a graphical interface.

The software was developed as a computational support tool for the research activities of the CODE project and is intended for controlled simulation experiments rather than large-scale production deployments.

1.1 Main Components

The toolbox consists of three main modeling modules together with a graphical user interface:

- **Network generation module**

  Generates synthetic networks with community structure using a Random Hyperbolic Block Model (RHBM). The module can either generate networks from scratch or randomize an observed graph while preserving structural constraints.

- **Epidemic simulation module**

  Two epidemic simulators are provided, supporting both behavioral multi-type epidemics and vaccination-driven epidemic dynamics. The multi-type model supports SIR and SIS families, optional exposed compartments, and both mean-field and stochastic network-based simulations. The S3I2 model instead focuses on vaccination-driven epidemic dynamics, incorporating multi-dose immunization, imperfect protection, waning immunity, and heterogeneous vaccination rates across communities on an explicit contact network.

- **Opinion dynamics module**

  Implements a dynamical model of opinion formation on networks coupled with epidemic risk perception. Opinions evolve through social interactions, community alignment, and external risk signals derived from epidemic dynamics.

- **Interactive interface**

  A web-based interface implemented using the `streamlit` framework provides interactive access to the models, allowing users to modify parameters and run simulations without directly interacting with the command line.

1.2 Design Philosophy

The CODE Toolbox follows a modular architecture where each modeling component is implemented as an independent Python module. The graphical interface acts as a lightweight orchestration layer that allows users to configure parameters and run simulations interactively.

All models are configured through JSON files that define simulation parameters, network inputs, and model-specific settings. This approach ensures reproducibility and allows simulations to be executed either through the graphical interface or directly from the command line.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

1.3 Scope of the Documentation

This document provides a technical description of the software architecture and of the computational modules implemented in the CODE Toolbox.

The emphasis of the manual is on:

- software structure and execution workflow,
- configuration of simulation parameters,
- interaction between the different modules of the toolbox.

The mathematical formulation of the models is described in the associated scientific publications and is therefore only summarized here to the extent necessary to understand the software implementation.

2 Software Architecture

The CODE Toolbox follows a modular architecture in which each modeling component is implemented as an independent Python module. The graphical interface provides a lightweight orchestration layer that connects user inputs to the underlying simulation engines.

The main components of the software correspond to the following functional blocks:

- network generation,
- epidemic simulation,
- opinion dynamics simulation,
- graphical user interface.

Each component can be executed independently through the command line, while the graphical interface integrates them into a single interactive workflow.

2.1 Repository Structure

The repository is organized into several directories corresponding to the main modeling modules:

```
CODE Toolbox
|
|-- app.py
|
|-- network/
|    |-- geometric_block_model/
|
|-- epidemic/
|    |-- multi_type_epidemic.py
|    |-- s3i2_simulator.py
|
|-- opinion/
|    |-- opinion_dynamics.py
|
|-- gui/
|
|-- noncompliance/
|
```

The most important components are:

- `app.py`

  Main entry point for the Streamlit graphical interface.

- `network/`

  Contains the implementation of the Random Hyperbolic Block Model used to generate synthetic networks with community structure.

- `epidemic/`

  Contains the implementation of the multi-type epidemic model and the S3I2 epidemic model:

  - `multi_type_epidemic.py`
  - `s3i2_simulator.py`

- `opinion/`

  Contains the implementation of the opinion dynamics simulator and its coupling with epidemic risk perception.

- `noncompliance/`

  Provides additional analysis tools used for empirical data exploration.

2.2 Module Interaction

The modeling components interact through a shared data representation of the contact network.

A typical workflow involves the following steps:

1. Generation or import of a contact network.
2. Simulation of epidemic dynamics on the network.
3. Simulation of opinion dynamics influenced by epidemic risk.
4. Visualization and analysis of the resulting observables.

The contact network is represented through two input files:

- a node list with community membership,
- an edge list describing the network structure.

These files can either be generated internally by the network generator or provided by the user.

2.3 Configuration System

All models are configured using JSON configuration files. These files specify:

- simulation parameters (time step, number of iterations, random seed),
- model parameters (rates, coupling strengths, behavioral parameters),
- network input files.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

The configuration files allow the same simulation to be executed either:

- from the command line, or
- through the Streamlit graphical interface.

Within the graphical interface, parameters defined in the configuration files can be interactively modified using sidebar widgets. The updated configuration is then passed directly to the simulation modules.

2.4 Execution Layers

The architecture of the software can be interpreted as a three-layer structure:

- **Model layer**
  Implements the core simulation logic, including the epidemic and opinion dynamics models.
- **Execution layer**
  Handles configuration loading, simulation loops, and result generation.
- **Interface layer**
  Provides interactive access to the simulation modules through the Streamlit graphical interface.

This separation simplifies experimentation and allows the simulation models to be reused in different contexts without modifying their internal implementation.

3 Network Generation

The CODE Toolbox provides a module for generating synthetic networks with community structure. These networks are used as the contact structures on which epidemic and opinion dynamics simulations are performed.

The generator is based on a **Random Hyperbolic Block Model (RHBM)**, which combines geometric embedding with community-based connectivity rules. The model produces heterogeneous networks with tunable clustering, heavy-tailed degree distributions, and assortative mixing between communities.

Only a brief description of the model is provided here. The mathematical formulation is described in the associated scientific publications.

3.1 Implementation

The network generation functionality is implemented through external Python scripts located in:

```
network/geometric_block_model/src/rhbm/
```

The two main scripts are:

```
rhbm_generate.py
rhbm_randomize.py
```

Their roles are:

- **rhbm_generate.py**

  Generates synthetic networks from model parameters.

- **rhbm_randomize.py**

  Randomizes an observed network while preserving structural constraints imposed by the RHBM.

These scripts can be executed either directly from the command line or through the graphical interface.

3.2 Synthetic Network Generation

Synthetic networks can be generated by executing:

```
python rhbm_generate.py [parameters]
```

Typical parameters include:

Parameter	Description
N	number of nodes
k	target average degree
γ	power-law exponent of the degree distribution
n	number of communities
p	assortativity parameter
q	community distance decay parameter
β	clustering control parameter

Example:

```
python rhbm_generate.py \
-N 1000 \
-k 10 \
-g 2.5 \
-n 5 \
-p 0.5 \
-q 1.0 \
-b 2.0 \
-o output_folder
```

3.3 Randomization of Observed Networks

An observed graph can be randomized using:

```
python rhbm_randomize.py \
-i input.graphml \
-o output_folder \
-b 2.0 \
--n_runs 1 \
--n_graphs 10
```

The input graph must be provided in GraphML format and must contain a node attribute specifying community membership.

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

3.4 Output Files

The generator produces an output directory containing:

- `node_list.txt`

  List of nodes and their community assignments.
- edge list files

  Text files containing the edges of the generated networks.
- optional metadata files

  Additional files containing statistics or mixing matrices.

These files provide the contact network required by the epidemic and opinion dynamics modules.

3.5 Integration with the CODE Toolbox

Within the Streamlit interface, the network generation scripts are invoked through a subprocess call.

The graphical interface:

1. collects model parameters through interactive widgets,
2. constructs the command-line invocation of the generator,
3. executes the script using `subprocess.run()`,
4. compresses the output directory into a downloadable archive.

The generated node list and edge list are stored in the application session state so that they can be directly reused by the epidemic or opinion dynamics modules.

4 Epidemic Models

The CODE Toolbox includes two independent epidemic simulators designed for different modeling scenarios:

- **Multi-Type Epidemic Model**

  A community-structured epidemic model supporting SIR/SIS dynamics, behavioral modulation, and both mean-field and network-based simulations.
- **S3I2 Network Epidemic Simulator**

  A stochastic network model describing epidemic spreading in the presence of multi-dose vaccination, imperfect protection, and waning immunity.

Both simulators operate on contact networks represented by a node list (with community membership) and an edge list describing the network topology.

The two models are implemented in:

```
epidemic/multi_type_epidemic.py
epidemic/s3i2_simulator.py
```

They can be executed either from the command line, used as Python libraries, or invoked through the Streamlit interface.

Only a concise description of the models is provided here. The mathematical formulation and theoretical analysis are described in the associated scientific publications.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

4.1 Multi-Type Epidemic Model

The model describes epidemic spreading in a population divided into communities.

Each community may have different behavioral parameters and may respond differently to epidemic prevalence and media signals.

The model supports two epidemiological families:

- **SIR** dynamics (permanent immunity after recovery)
- **SIS** dynamics (recovered individuals return to the susceptible state)

An optional exposed compartment can also be enabled, producing SEIR or SEIS dynamics. The model can operate in two simulation modes:

- **Mean-field mode**

  Deterministic compartmental dynamics defined at the community level.

- **Network mode**

  Stochastic node-level simulation on an explicit contact network.

Transmission rates may be modulated by:

- media-driven awareness signals,
- local infection prevalence,
- community-specific behavioral parameters.

Configuration

All simulation parameters are defined in the JSON configuration file:

```
epidemic/config.json
```

The configuration file specifies:

- simulation parameters (time step, number of iterations, random seed),
- epidemiological rates,
- behavioral modulation parameters,
- community-specific parameters,
- network input files.

The configuration file can be edited manually or modified dynamically by the graphical interface.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Command Line Execution

The epidemic simulator can be executed directly as a Python script:

```
python multi_type_epidemic.py
```

The program performs the following steps:

1. Load the configuration file.
2. Initialize the epidemic model.
3. Run the simulation.
4. Generate a visualization of the results.

The output consists of time series describing the epidemic evolution and a figure summarizing the main observables.

Using the Module as a Python Library

The epidemic simulator can also be used programmatically by importing the class `MultiTypeEpidemic`.

Example:

```
from multi_type_epidemic import MultiTypeEpidemic

model = MultiTypeEpidemic(config)

results = model.run()

fig = MultiTypeEpidemic.plot_results(results)
```

The returned `results` dictionary contains time series for the main epidemic observables, including total prevalence and community-level infection fractions.

Network Input

When running in network mode, the simulator requires a contact network provided through two input files:

- a node list specifying community membership,
- an edge list specifying the network topology.

These files can either be:

- generated using the network generation module,
- provided by the user.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Integration with the Streamlit Interface

Within the graphical interface, the epidemic simulator is executed by directly instantiating the `MultiTypeEpidemic` class.

The workflow implemented in the interface is:

1. Load the configuration file.
2. Update parameters through sidebar widgets.
3. Instantiate the epidemic model.
4. Execute the simulation.
5. Display the results.

The graphical interface therefore acts only as a front-end for parameter selection and visualization, while the simulation logic remains entirely implemented in the Python module.

4.2 S3I2 Network Epidemic Simulator

The S3I2 epidemic simulator models epidemic spreading in the presence of vaccination and waning immunity.

The implementation is contained in:

```
epidemic/s3i2_simulator.py
```

The model describes individuals belonging to five epidemiological states:

- S_0 : fully susceptible individuals
- S_1 : partially immunized individuals
- S_2 : maximally immunized individuals
- I_1 : infected after zero or one vaccination events
- I_2 : infected after two or more vaccination events

The model includes:

- vaccination and booster doses,
- imperfect protection after vaccination,
- waning immunity,
- heterogeneous vaccination rates across communities.

The population is represented by an undirected contact network, where nodes correspond to individuals and edges represent potentially infectious contacts.

Each node belongs to a community, and vaccination rates may vary across communities.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

Configuration

The simulator is controlled through a JSON configuration file specifying:

- network input files,
- epidemiological parameters,
- vaccination parameters,
- simulation length,
- plotting options.

Typical parameters include:

Parameter	Description
β	transmission rate
γ	recovery rate
σ_1	susceptibility reduction after first vaccination
σ_2	susceptibility reduction after second vaccination
η_1	waning rate from S_1
η_2	waning rate from S_2

Command Line Execution

The simulator can be executed using:

```
python s3i2_simulator.py config.json
```

The program loads the configuration file, performs the simulation, and prints the time evolution of the epidemic states.

If plotting is enabled in the configuration file, figures summarizing the global and community-level epidemic dynamics are generated.

Using the Module as a Python Library

The simulator can also be used programmatically:

```
from s3i2_simulator import run_simulation

history, history_com = run_simulation(config)
```

The returned objects contain time series describing the evolution of the five epidemiological states at both the global and community levels.

Integration with the Streamlit Interface

Within the Streamlit interface, the simulator can be invoked using the same configuration files used for standalone execution. The graphical interface allows users to select parameters interactively and visualize the resulting epidemic dynamics.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

5 Opinion Dynamics Model

The CODE Toolbox includes a simulator for opinion dynamics on networks coupled with epidemic risk perception.

The implementation is contained in the file:

```
opinion/opinion_dynamics.py
```

The model describes the evolution of individual opinions on a contact network where agents interact socially and respond to perceived epidemic risk.

The module can be used in three different ways:

- executed as a standalone Python program,
- imported as a Python library,
- invoked through the Streamlit graphical interface.

The detailed theoretical formulation of the model is presented in the associated scientific publications. This section focuses on the software implementation and execution workflow.

5.1 Model Overview

The opinion dynamics model describes the evolution of a continuous opinion variable associated with each node of a contact network.

Opinions evolve through three main mechanisms:

- **Social influence**

  Agents adjust their opinions in response to the average opinion of their neighbors.

- **Community alignment**

  Agents are attracted toward the average opinion of their community.

- **External risk influence**

  Epidemic prevalence generates a perceived risk signal that acts as an external field affecting opinion dynamics.

The perceived risk signal evolves dynamically and depends on the epidemic prevalence and on media amplification effects.

The simulator integrates the following components:

- a deterministic SIR epidemic model,
- a risk perception module,
- a network-based opinion dynamics model.

These components evolve jointly during the simulation.

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

5.2 Configuration

All parameters are defined in the configuration file:

```
opinion/config.json
```

The configuration specifies:

- network input files,
- simulation control parameters,
- opinion dynamics parameters,
- epidemic model parameters,
- media bias and risk perception parameters.

The configuration file allows the same simulation to be executed either directly from the command line or through the graphical interface.

5.3 Running the Simulation from the Command Line

The simulator can be executed directly as a Python script:

```
python opinion_dynamics.py config.json
```

If no configuration file is provided, the default configuration file located in the `opinion` directory is used.

The execution workflow is:

1. load the configuration file,
2. construct the contact network,
3. initialize epidemic and opinion states,
4. run the simulation loop,
5. generate a visualization of the results.

The output consists of time series describing the evolution of epidemic variables, perceived risk, and opinion statistics.

5.4 Using the Module as a Python Library

The simulator can also be used programmatically by importing the simulation class.

Example:

```
from opinion_dynamics import Simulation
from opinion_dynamics import plot_results

sim = Simulation(config)

results = sim.run()

fig = plot_results(results)
```

Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

The returned results dictionary contains time series for:

- epidemic variables,
- perceived risk,
- global mean opinion,
- community-level opinion statistics.

5.5 Network Input

The simulator requires a network described by two input files:

- a node list containing node identifiers and community labels,
- an edge list describing the network topology.

These files can either be provided by the user or generated using the network generation module of the toolbox.

5.6 Integration with the Streamlit Interface

Within the Streamlit interface, the opinion dynamics simulator is executed by instantiating the `Simulation` class.

The graphical interface performs the following steps:

1. load the configuration file,
2. update parameters using sidebar controls,
3. instantiate the simulation object,
4. execute the simulation,
5. display the resulting figures.

The graphical interface therefore acts as a parameter selection and visualization layer, while the simulation logic remains entirely implemented in the Python module.

6 Streamlit Interface

The CODE Toolbox provides an interactive graphical interface implemented using the `streamlit` framework. The application is defined in the file `app.py`, which acts as the main entry point of the graphical user interface.

The interface exposes the main computational components of the toolbox through a web-based dashboard that allows users to configure parameters and run simulations without directly interacting with the command line.

The application is organized into four independent modules:

- Network generation
- Epidemic model simulation
- Opinion dynamics simulation
- Noncompliance analysis

The module selection is performed through a sidebar menu, which dynamically activates the corresponding interface.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

6.1 Architecture

The graphical interface acts as a lightweight orchestration layer that connects user inputs to the underlying simulation engines implemented in Python.

The architecture can be summarized as follows:

$$\text{Streamlit GUI} \rightarrow \text{Simulation Modules} \rightarrow \text{Visualization}$$

More specifically:

- The **network generation module** invokes external scripts implementing the Random Hyperbolic Block Model (RHBM).
- The **epidemic module** directly instantiates the class `MultiTypeEpidemic`.
- The **opinion dynamics module** invokes the `Simulation` class from `opinion_dynamics.py`.
- The **noncompliance module** loads an external Streamlit page defined in the `noncompliance` package.

The interface does not modify the internal logic of the simulation models. Instead, it dynamically updates the configuration parameters passed to each model.

6.2 Network Generation Module

The network generator produces synthetic networks based on the Random Hyperbolic Block Model (RHBM).

Two operational modes are available:

- **Generate synthetic network:** a network is generated from scratch using model parameters such as network size, average degree, number of communities, and assortativity.
- **Randomize observed network:** an uploaded graph (in GraphML format) is randomized while preserving structural constraints imposed by the RHBM.

The interface internally calls the scripts:

```
rhbm_generate.py
rhbm_randomize.py
```

which are executed through a subprocess invocation.

The generated output folder is automatically compressed into a ZIP archive and made available for download.

To facilitate interoperability with other modules, the interface stores the most recently generated network in the session state.

6.3 Epidemic Simulation Module

The epidemic module provides interactive access to the multi-type epidemic model implemented in the class `MultiTypeEpidemic`.

The model configuration is initially loaded from the JSON file:

```
epidemic/config.json
```

All key parameters (rates, behavioral modulation, media response, and simulation control parameters) can be interactively modified through sidebar widgets.

The simulation workflow is:

1. Load configuration file.
2. Update parameters using the graphical interface.
3. Instantiate the epidemic model.
4. Execute the simulation via `run()`.
5. Display results using the internal plotting routine.

Network inputs can be provided either by uploading node and edge files or by reusing the most recently generated network.

6.4 Opinion Dynamics Module

The opinion dynamics module provides an interface to the coupled opinion–epidemic simulation implemented in `opinion_dynamics.py`.

The model is configured through:

```
opinion/config.json
```

Users can interactively modify parameters controlling:

- social influence strength,
- community alignment,
- external risk coupling,
- media bias and risk perception dynamics.

After parameter selection, the simulation is executed through:

```
Simulation.run()
```

The resulting observables are visualized using the plotting routine `plot_results()`.

6.5 Session State and Data Reuse

To enable interaction between modules, the application stores certain objects in the Streamlit session state.

In particular:

- the most recently generated network archive,
- the corresponding node membership file,
- the detected edge list file.

This mechanism allows a network generated in the first module to be immediately reused in the epidemic or opinion simulations without re-uploading files.

6.6 Execution

The Streamlit interface can be launched from the project root directory using:

```
streamlit run app.py
```

The application will then be available through the local web interface provided by Streamlit.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

7 Execution Workflow

This section summarizes the typical workflow for using the CODE Toolbox to perform simulation experiments.

The toolbox can be used either through the command line, through Python scripts, or through the Streamlit graphical interface. Regardless of the interface used, the logical sequence of operations remains the same.

7.1 Typical Simulation Pipeline

A typical simulation experiment using the CODE Toolbox follows the steps below:

1. Generate or load a contact network.
2. Configure simulation parameters.
3. Run the epidemic simulation.
4. Run the opinion dynamics simulation.
5. Analyze and visualize the resulting observables.

The same network can be reused across multiple simulations in order to compare different parameter configurations.

7.2 Network Preparation

The first step consists of obtaining a contact network.

This can be achieved in two ways:

- by generating a synthetic network using the RHBM generator,
- by providing an observed network.

In both cases the network must be represented using two input files:

- a node list containing node identifiers and community labels,
- an edge list describing the network topology.

These files are then used as input for the epidemic and opinion dynamics simulations.

7.3 Configuration of Simulation Parameters

Simulation parameters are defined in JSON configuration files located in the corresponding module directories.

Examples include:

```
epidemic/config.json
opinion/config.json
```

These files specify:

- simulation control parameters,
- epidemiological or opinion model parameters,
- behavioral or media-response parameters,
- network input paths.

Users can either modify these files directly or override parameters programmatically.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

7.4 Running Simulations

Simulations can be executed in several ways.

Command Line Execution

Each module can be run independently as a Python program.

Examples:

```
python multi_type_epidemic.py
```

```
python opinion_dynamics.py config.json
```

This approach is convenient for batch experiments or automated workflows.

Python API Usage

The simulation modules can also be imported and executed within Python scripts.

Example:

```
from multi_type_epidemic import MultiTypeEpidemic

model = MultiTypeEpidemic(config)
results = model.run()
```

Similarly, the opinion dynamics simulator can be executed programmatically using the `Simulation` class.

This mode of operation allows the toolbox to be integrated into larger computational pipelines.

Graphical Interface

The toolbox also provides an interactive interface implemented with the Streamlit framework.

The interface can be launched using:

```
streamlit run app.py
```

The graphical interface allows users to:

- generate networks,
- configure simulation parameters interactively,
- run epidemic and opinion simulations,
- visualize the results immediately.

7.5 Result Visualization

Both simulation modules include built-in visualization routines that produce figures summarizing the main observables of the simulation.

Typical outputs include:

- epidemic prevalence over time,
- community-level infection dynamics,
- perceived risk evolution,
- global and community-level opinion statistics.

The plotting functions return standard `matplotlib` figures, which can be displayed interactively or saved to disk for further analysis.

Finanziato dall'Unione europea NextGenerationEU | Ministero dell'Università e della Ricerca | Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA

7.6 Reproducibility

Reproducibility of simulation experiments is ensured through:

- explicit configuration files,
- user-defined random seeds,
- deterministic execution of the simulation modules when the same configuration is used.

Users are therefore encouraged to archive the configuration files associated with each experiment in order to reproduce results in future analyses.