

---

# **InfraRisk Simulation Platform**

***Release 1.0***

**Srijith Balakrishnan**

**Jan 18, 2023**



## CONTENTS:

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Methodology and Architecture</b>	<b>3</b>
2.1	Integrated infrastructure network . . . . .	5
2.1.1	Power system model . . . . .	5
2.1.2	Water system model . . . . .	6
2.1.3	Transport system model . . . . .	6
2.1.4	Modeling interdependencies . . . . .	7
2.2	Hazard initiation and vulnerability modeling . . . . .	7
2.3	Recovery modeling . . . . .	8
2.3.1	Heuristics-based repair sequences . . . . .	8
2.3.2	Recovery optimization . . . . .	8
2.4	Simulation of direct and indirect effects . . . . .	9
2.5	Resilience quantification . . . . .	9
<b>3</b>	<b>Data requirements</b>	<b>11</b>
3.1	Infrastructre Networks . . . . .	11
3.1.1	Water distribution system . . . . .	11
3.1.2	Power systems . . . . .	12
3.1.3	Transportation system . . . . .	12
3.2	Infrastructure dependencies . . . . .	12
3.3	Infrastructure disruption data . . . . .	13
<b>4</b>	<b>Installation</b>	<b>15</b>
4.1	From sources . . . . .	15
<b>5</b>	<b>Getting started</b>	<b>17</b>
5.1	Shelby county simulation demonstration . . . . .	17
5.1.1	Load required modules and libraries . . . . .	17
5.1.2	Create an IntegratedNetwork object and load inidividual networks . . . . .	18
5.1.3	Load socioeconomic data for Shelby County . . . . .	20
5.1.4	Generate disaster scenario and set infrastructure component disruptions . . . . .	20
5.1.5	Set initial locations of restoratio ncrews . . . . .	21
5.1.6	Simulation of restoration and recovery process . . . . .	21
5.1.7	Quanitification of loss using resilience metrics . . . . .	22
5.1.8	Plot simulation results . . . . .	22
5.2	Additional tutorials . . . . .	23
<b>6</b>	<b>Support</b>	<b>25</b>
<b>7</b>	<b>API Documentation</b>	<b>27</b>

7.1	infrarisk.src package . . . . .	27
7.1.1	Subpackages . . . . .	27
7.1.1.1	infrarisk.src.hazards package . . . . .	27
7.1.1.1.1	infrarisk.src.hazards.custom module . . . . .	27
7.1.1.1.2	infrarisk.src.hazards.fragility_based module . . . . .	28
7.1.1.1.3	infrarisk.src.hazards.radial module . . . . .	31
7.1.1.1.4	infrarisk.src.hazards.random module . . . . .	33
7.1.1.1.5	infrarisk.src.hazards.track module . . . . .	33
7.1.1.2	infrarisk.src.physical package . . . . .	35
7.1.1.2.1	Subpackages . . . . .	35
7.1.1.2.1.1	infrarisk.src.physical.power package . . . . .	35
7.1.1.2.1.2	Submodules . . . . .	35
7.1.1.2.1.3	infrarisk.src.physical.power.power_system_model module . . . . .	35
7.1.1.2.1.4	Module contents . . . . .	36
7.1.1.2.1.5	infrarisk.src.physical.transportation package . . . . .	36
7.1.1.2.1.6	infrarisk.src.physical.transportation.network module . . . . .	36
7.1.1.2.1.7	infrarisk.src.physical.transportation.transpo_compons module . . . . .	39
7.1.1.2.1.8	infrarisk.src.physical.transportation.utils module . . . . .	40
7.1.1.2.1.9	Module contents . . . . .	40
7.1.1.2.1.10	infrarisk.src.physical.water package . . . . .	40
7.1.1.2.1.11	infrarisk.src.physical.water.water_network_model module . . . . .	40
7.1.1.2.1.12	Module contents . . . . .	43
7.1.1.2.2	Submodules . . . . .	43
7.1.1.2.3	infrarisk.src.physical.integrated_network module . . . . .	43
7.1.1.2.4	infrarisk.src.physical.interdependencies module . . . . .	47
7.1.1.2.5	Module contents . . . . .	49
7.1.1.3	infrarisk.src.socioeconomic package . . . . .	49
7.1.1.3.1	Submodules . . . . .	49
7.1.1.3.2	infrarisk.src.socioeconomic.se_analysis module . . . . .	49
7.1.1.3.3	Module contents . . . . .	50
7.1.2	infrarisk.src.network_recovery module . . . . .	50
7.1.3	infrarisk.src.optimizer module . . . . .	52
7.1.4	infrarisk.src.plots module . . . . .	53
7.1.5	infrarisk.src.recovery_strategies module . . . . .	54
7.1.6	infrarisk.src.repair_crews module . . . . .	55
7.1.7	infrarisk.src.resilience_metrics module . . . . .	57
7.1.8	infrarisk.src.simulation module . . . . .	58
7.1.9	Module contents . . . . .	59
8	SEC Disclaimer . . . . .	61
9	Funding Statement . . . . .	63
10	Indices and tables . . . . .	65
	Bibliography . . . . .	67
	Python Module Index . . . . .	69

## INTRODUCTION

This package was developed as part of the Disaster REsilience Assessment, Modelling, and INno-variation Singapore (DREAMIN'SG) project at the Future Resilient Systems of the Singapore-ETH Centre. The DREAMIN'SG project is funded by the National Research Foundation, Singapore under the Intra-CREATE grant program. In the wake of increasing threats posed by climate change on urban infrastructure<sup>1</sup>, this project aims at studying the effects of policy interventions and network characteristics on the resilience of urban infrastructure networks. Given the critical nature of the above infrastructure systems, their interdependencies need to be considered in pre- and post-disaster resilience actions.

In specific, the DREAMIN'SG project envisaged to build a platform to assess and predict the resilience of urban infrastructure systems and propose new pathways to develop innovative technologies and services for its improvement. The urban infrastructure system is modeled as an interdependent power-, water-, and transportation network that interact with each other before, during and after a disaster. The researchers are developing an integrated simulation model to study the performance of the interdependent infrastructure network under various disruption and recovery scenarios. Based on the simulation-generated datasets, machine learning algorithms are implemented to understand the causal relationship between topological and policy-related interventions and disaster risks. The results of the research are intended to support local governments and system managers to improve infrastructure resilience against weather-related disruptions. The methodology adopted in the study is presented in Fig. 1.

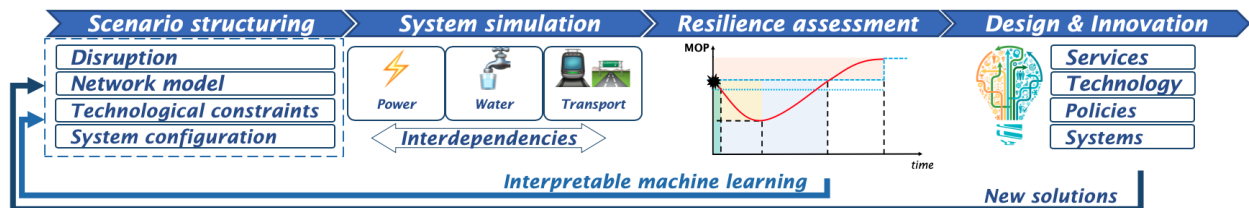


Fig. 1: Figure 1. Methodological framework of DREAMIN'SG project

The steps of the project are summarized as follows:

1. Multiple scenarios are generated by considering different disruptions, network models, technological constraints and system configurations.
2. A simulation model is created for the interdependent power grid, water distribution system, and road transportation system.
3. Resilience is assessed based on the simulated performance of the three systems.
4. An interpretable machine learning algorithm is implemented to analyze the scenarios and extract information related to key system features that influence resilience.

<sup>1</sup> Nissen, K. M. and U. Ulbrich (2017). Increasing frequencies and changing characteristics of heavy precipitation events threatening infrastructure in Europe under climate change. *Natural Hazards and Earth System Sciences*, 17(7), 1177–1190. ISSN 16849981.

5. The identified system features inform the design of new services, technologies, and products that are able to simultaneously enhance resilience and accommodate the technological constraints.

For further information and updates on the project, please visit the [DREAMIN'SG webpage](#). In the rest of the documentation, the details of the interdependent infrastructure simulation platform, including its modeling, installation, and usage are discussed.

## METHODOLOGY AND ARCHITECTURE

The integrated simulation model has been developed as a Python-based package consisting of modules for simulation of system- and network-level cascading effects resulting from component failures. The overall methodological framework of the integrated simulation platform is illustrated in Figure 2.

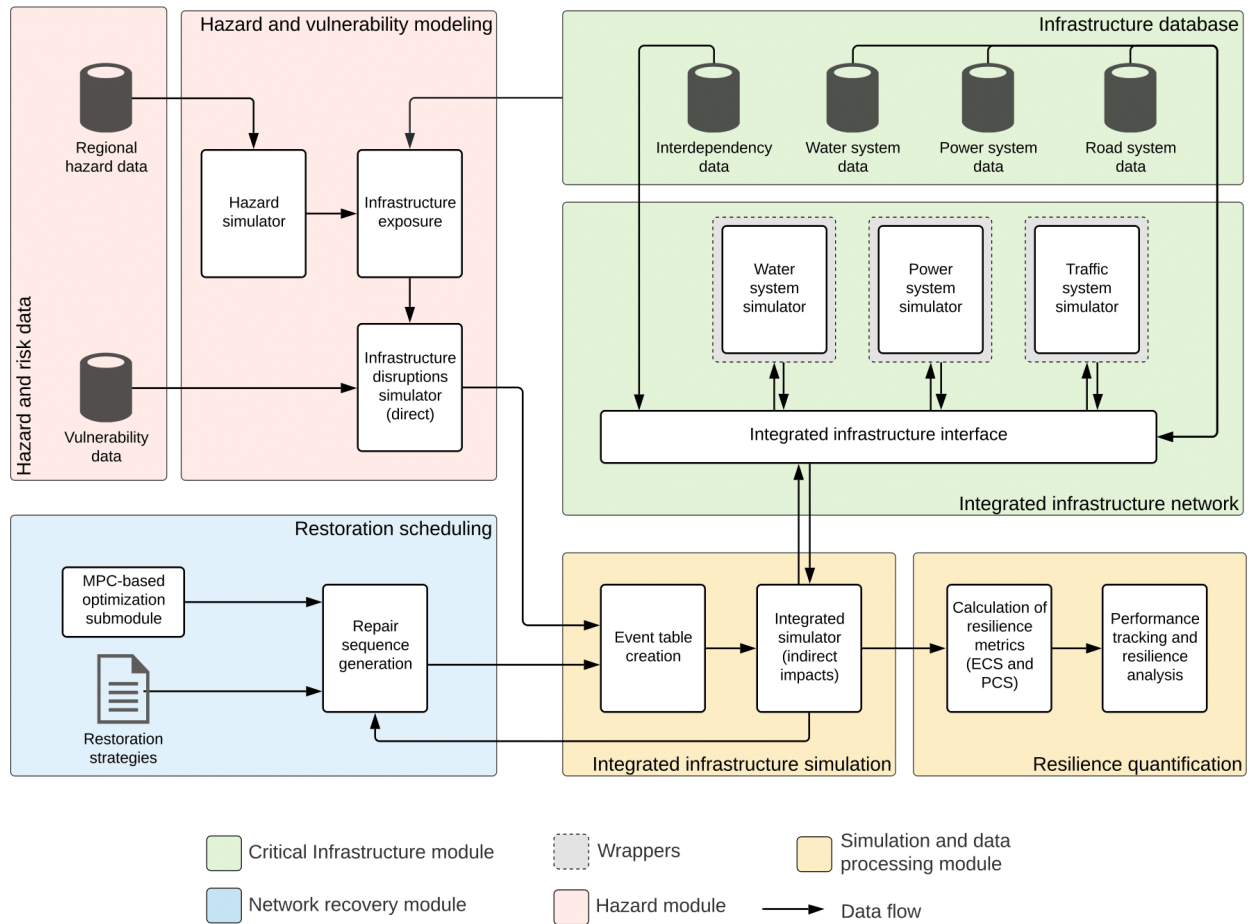


Fig. 1: Figure 2. InfraRisk integrated simulation platform structure

The platform is based on the widely accepted risk- and resilience analysis frameworks as presented in [Argyroudis2020] and [Balakrishnan2020]. In this framework, the most important component is an interdependent infrastructure model that consists of various infrastructure system submodels of interest. In addition, the major hazards in the region can also be modeled. Further, the vulnerabilities in the network to those hazards are mapped and the direct impacts (physical and functional failures in the infrastructure components) are simulated using the hazard model. For scheduling post-disaster

restoration/repair actions, a recovery model is also developed. The restoration actions are prioritized based on specific recovery strategies or optimization methods. The indirect failures in the network are simulated using the interdependent infrastructure model based on the initial failure events and the subsequent repair actions. The component- and system-level operational performance are quantified and tracked using appropriate resilience metrics.

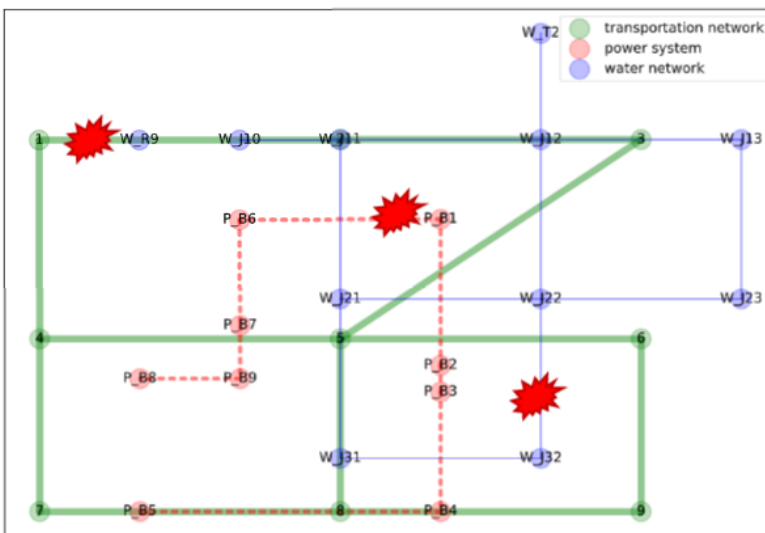
The basic idea behind the InfraRisk simulation package is to integrate existing infrastructure-specific simulation models through an object-oriented interface so that interdependent infrastructure simulation can be achieved. Interfacing requires identifying and modeling the dependencies among various infrastructure components and time-synchronization among infrastructure simulation models. To address the above challenges, InfraRisk is built using a sequential simulation framework (Figure 2). The advantage of this approach is that it simplifies the efforts for data preparation and enables the complete utilization of component-level modeling features of the domain-specific infrastructure models.

InfraRisk consists of five modules, namely,

1. integrated infrastructure network simulation
2. hazard initiation and vulnerability modeling
3. recovery modeling
4. simulation of direct and indirect effects
5. resilience quantification.

In the rest of the section, a detailed discussion on each of the above modules is provided.

Sample integrated network



Performance tracking

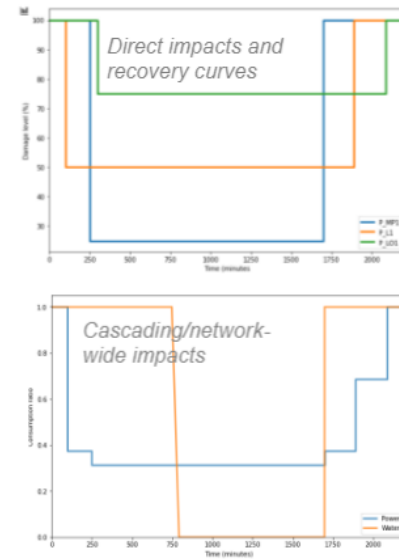


Fig. 2: Figure 3. Implementation of the simulation platform to generate performance curves



## 2.1 Integrated infrastructure network

This module houses the three infrastructure models to simulate power-, water-, and transport systems. These models are developed using existing Python-based packages. In order to model the power system, *pandapower* is employed [Thurner2018]. The water distribution system is modeled using *wntr* package [Klise2018]. The traffic model provides the travel costs for traveling from one point in the network to another and is modeled using the static traffic assignment method [Boyles2020]. All three packages have network-flow optimization models that identify the steady-state resource flows in the respective systems considering the operational constraints. The details of the packages are presented in Table 2.

Table 2. Infrastructure packages used in the simulation model

Infrastructure	Package	Capabilities
Power	<i>pandapower</i>	<ul style="list-style-type: none"> <li>• Capable of generating power distribution systems with standard components, such as lines, buses, and transformers based on design data.</li> <li>• Capable of performing power-flow analysis.</li> </ul>
Water	<i>wntr</i>	<ul style="list-style-type: none"> <li>• Capable of generating water distribution systems with standard components such as pipes, tanks, and nodes based on design data.</li> <li>• Capable of performing pressure dependent demand or demand-driven hydraulic</li> </ul>
Transport	static traffic assignment package	<ul style="list-style-type: none"> <li>• Capable of implementing static traffic assignment and computing travel times between origin-destination pairs.</li> </ul>

### 2.1.1 Power system model

The *pandapower* package can be used to determine the steady-state optimal power flow for a given set of system conditions. The optimal power flow problem, solved by *pandapower*, attempts to minimize the total power distribution costs in the system under load flow-, branch-, bus-, and operational power constraints (Equation 1)

$$\begin{aligned}
& \min \sum_{i \in \text{gen}, \text{sgen}, \text{load}, \text{extgrid}} P_i \times f_i(P_i) \\
& \text{s.t. } P_{\min, i} \leq P_i \leq P_{\max, i} & \forall i \in \text{gen}, \text{sgen}, \text{extgrid}, \text{load} \\
& Q_{\min, i} \leq Q_i \leq Q_{\max, i} & \forall i \in \text{gen}, \text{sgen}, \text{extgrid}, \text{load} \\
& V_{\min, j} \leq V_j \leq V_{\max, j} & \forall j \in \text{bus} \\
& L_k \leq L_{\max, k} & \forall k \in \text{trafo}, \text{line}, \text{trafo3w}
\end{aligned}$$

where  $i, j$ , and  $k$  are the power system components,  $gen$  is the set of generators,  $sgen$  is the set of static generators,  $load$  is the set of loads,  $extgrid$  is the set of external grid connections,  $bus$  is the set of bus bars,  $trafo$  is the set of transformers,  $line$  is the set of lines, and  $trafo3w$  is the set of three winding transformers,  $f_i(\cdot)$  is the cost function,  $P_i$  is the active power in  $i$ ,  $Q_i$  is the reactive power in  $i$ ,  $V_j$  is the voltage in  $j$  and  $L_k$  is the loading percentage in  $k$ .

## 2.1.2 Water system model

The *wntr* package can simulate water flows in water distribution systems using two common approaches, namely, demand-driven analysis (DDA) and pressure-dependent demand analysis (PDA). While DDA assigns pipe flows based on the demands irrespective of the pressure at demand nodes, PDA assumes that the demand is a function of the pressure at which water is supplied. The PDA approach is more suitable for pressure-deficient situations, such as disaster-induced disruptions to water infrastructure. In the case of PDA, the actual node demands is computed as a function of available the water pressure at the nodes as in Equation.

The *wntr* package can simulate water flows in water distribution systems using two common approaches, namely, demand-driven analysis (DDA) and pressure-dependent demand analysis (PDA). While DDA assigns pipe flows based on the demands irrespective of the pressure at demand nodes, PDA assumes that the demand is a function of the pressure at which water is supplied. The PDA approach is more suitable for pressure-deficient situations, such as disaster-induced disruptions to water infrastructure. In the case of PDA, the actual node demands is computed as a function of available the water pressure at the nodes as in Equation~ref{eq:PDA} cite{Klise2020}.

$$d_i(t) = \begin{cases} 0 & p_i(t) \leq P_0 \\ D_i(t) \left( \frac{p_i(t) - P_0}{P_f - P_0} \right)^{\frac{1}{e}} & P_0 < p_i(t) \leq P_f \\ D_i & p_i(t) > P_f \end{cases}$$

where  $d_i(t)$  is the actual demand at node  $i$  at time  $t$ ,  $D_i(t)$  is the desired demand at a node  $i$  at  $t$ ,  $p_i(t)$  is the available pressure in node  $i$  at  $t$ ,  $P_f$  is the nominal pressure, and  $P_0$  is the lower pressure threshold, below which no water is consumed. In InfraRisk, the hydraulic simulation is performed using the PDA approach.

## 2.1.3 Transport system model

The traffic condition in the transport system is modeled using the static traffic assignment method based on the principle of user-equilibrium. Under user-equilibrium, every user tries to minimize their travel costs. The traffic assignment problem considered in InfraRisk package is formulated as follows (Equation~ref{eq:staeqs}) cite{Boyles2020}.

$$\begin{aligned} \min_{\mathbf{x}, \mathbf{h}} \quad & \sum_{(i,j) \in A} \int_0^{x_{ij}} t_{ij}(x_{ij}) dx \\ \text{s.t.} \quad & x_{ij} = \sum_{\pi \in \Pi} h^\pi \delta_{ij}^\pi \quad \forall (i,j) \in A \\ & \sum_{\pi \in \Pi^{rs}} h^\pi = d^{rs} \quad \forall (r,s) \in Z^2 \\ & h^\pi \geq 0 \quad \forall \pi \in \Pi \end{aligned}$$

where  $A$  is the set of all road links with  $i$  and  $j$  as the tail and head nodes,  $t_{ij}$  is the travel cost on link  $(i,j)$ ,  $x_{ij}$  is the traffic flow on link  $(i,j)$ ,  $h^\pi$  is the flow on path  $\pi \in \Pi$ ,  $\delta_{ij}^\pi$  is an indicator variable that denotes whether  $(i,j)$  is part of  $\pi$ ,  $d^{rs}$  is the total flow between origin-destination pair  $(r,s)$ .

## 2.1.4 Modeling interdependencies

The module also consists of an interdependency layer which serves as an interface between infrastructure systems. The interdependency layer stipulates the different pieces of information that can be exchanged among individual infrastructure systems and their respective formats. The interdependency submodule also stores information related to the various component-to-component couplings between infrastructure systems. The module facilitates the communication between infrastructure systems and enables information transfer triggered by dependencies. Currently the following dependencies are considered.

1. Power-water dependencies, which include dependency of water pumps on electric motors and generators on reservoirs (hydro-power).
2. Dependencies also exist between road traffic system and the other two infrastructure systems, as the former provides access to the latter. The disruptions to transport infrastructure components and their recovery are key considerations that influence the restoration and recovery of all other infrastructure systems. The module also stores the functional details of all infrastructure components, including their operational status after a disaster.

The interdependency layer communicates with the infrastructure simulators through inbuilt functions (wrappers).

## 2.2 Hazard initiation and vulnerability modeling

The hazard module generates disaster scenarios and initiates disaster-induced infrastructure failures based on their vulnerability. The hazard initiation and the resulting infrastructure component failures is the first step in the interdependent infrastructure simulation. The probabilistic failure of an infrastructure component is modeled as follows (Equation~ref{eq:failureprob}):

$$p(\text{failure}_i) = p(\text{hazard}) \times p(\text{exposure}_i | \text{hazard}) \times p(\text{failure}_i | \text{exposure}_i)$$

where  $i$  is the component,  $p(\cdot)$  is the probability. The probability of failure of a component is computed as the product of the probability of the hazard, the probability of the component being exposed to the hazard if it occurs, and the probability of failure of the component if it is exposed to the event.

In InfraRisk, infrastructure component failures can be induced using five types of hazards.

1. Point events (e.g., explosions)
2. Track-based events (e.g., hurricanes and floods)
3. Random disruption events (e.g., seven random road link failures)
4. Custom events (e.g., fail five specific pipelines)
5. Fragility-based events (e.g., earthquakes)

Among these, the first four two of event types are agnostic to infrastructure vulnerability and focus on the proximity of the components to the location of the event. The random events are generated randomly based on user requirements. Custom events can generate disruptions based on the user-defined lists. The fragility-based events are generated based on the component fragility curves and considers the vulnerability characteristics of the components.

## 2.3 Recovery modeling

The third module, which is the recovery module, determines how the repair actions are sequenced and implemented. The three major factors that influence recovery are the availability of repair crews, repair times of components, and the criteria used for selecting subsequent components to restore. In InfraRisk, the user can specify the number of crews deployed for restoration of the three infrastructure systems, their initial locations in the network, and the repair times of the infrastructure components.

The repair sequence can be derived using two approaches as follows.

### 2.3.1 Heuristics-based repair sequences

The first approach is to adopt pre-defined repair strategies based on performance- and network-based heuristics. Currently, there are three inbuilt strategies based on the following criteria:

1. **Maximum flow handled:** The resource-flow during normal operating conditions could reflect the importance of an infrastructure component to the system. The maximum resource-flow handled by a component, considering the temporal fluctuations, can be used as a performance-based heuristics to prioritize failed components for restoration.
2. **Betweenness centrality:** Centrality is a graph-based measure that is used to denote the relative importance of components (nodes and links) in a system. Betweenness centrality is often cited as an effective measure to identify critical infrastructure components cite{Almoghathawi2019}.
3. **Landuse/zone:** Certain regions of a network may have consumers with large demands or critical to the functioning of the whole city. Industrial zones and central business districts are critical from both societal and economic perspectives.

While it is comparatively easier to derive repair sequences based on heuristics, they may not guarantee optimal recovery of the system or the network.

### 2.3.2 Recovery optimization

The second approach is an optimization model leveraging on the concept of model predictive control (MPC) cite{Camacho2007}. In this approach, first, out of  $n$  repair steps, the solution considering only  $k$  steps (called the prediction horizon) is computed. Next, the first step of the obtained solution is applied to the system and then the process is repeated for the remaining  $n-1$  components until all components are scheduled for repair. In the context of the integrated infrastructure simulation, the optimizer module evaluates repair sequences of the length of the prediction horizon for each infrastructure (assuming that each infrastructure has a separate recovery crew) based on a chosen resilience metric cite{Kottmann2021}. The optimal repair sequence is found by maximizing the resilience metric. At this stage, the optimal repair action in each prediction horizon is computed using a brute-force approach where the resilience metric is evaluated for each of the possible repair sequences. The major limitation of MPC is that it is suitable only for small disruptions involving a few component failures; MPC becomes computationally expensive to derive optimal restoration sequences for larger disruptions due to the large number of repair permutations it has to simulate.

## 2.4 Simulation of direct and indirect effects

The simulation module implements the integrated infrastructure simulation in two steps, namely, event table generation and interdependent infrastructure simulation. The objective of the event table is to provide a reference object to schedule all the disruptions and repair actions for implementing the interdependent network simulation.

The component failures, repair actions, and the respective time-stamps, are recorded in an event table for later use in the simulation module. The simulation platform uses the event table as a reference to modify the operational status of system components during the simulation, so that the consequences of disaster events and repair actions are reflected in the system performance. The recovery module also stores details including the number of repair crews for every infrastructure system, and their initial locations.

The next step is to simulate the interdependent effects resulting from the component disruptions and the subsequent restoration efforts. One of the main challenge in simulating the interdependent effects using a platform that integrates multiple infrastructure models is the time synchronization.

In order to synchronize the times, the power- and water- system models are run successively for every subsequent time-interval in the event table. The required water system metrics are collected for every one minute of simulation time from the *wntr* model, whereas power system characteristics at the start of every time interval is recorded from the *pandapower* model. The power flow characteristics are assumed to remain unchanged unless there is any modification to the power system in the subsequent time-steps in the simulation.

## 2.5 Resilience quantification

Currently, the model has two measures of performance (MOP), namely, equitable consumer serviceability (ECS) and prioritized consumer serviceability (PCS), to quantify the system- and network steady-state performances. The above MOPs are based on the well-known concepts of satisfied demand cite{Didier2017} and productivity cite{Poulin2021}. Th MOPs are used as the basis for defining the resilience metrics.

Consider an interdependent infrastructure network  $\mathbb{K}$  consisting of a set of infrastructure systems denoted by  $K : K \in \mathbb{K}$ . There are  $N$  consumers who are connected to  $\mathbb{K}$  and the resource supply from a system  $K$  to consumer  $i$  in  $N$  at time  $t$  under normal operating conditions is represented by  $S_i^K(t)$ .

The ECS approach assumes equal importance to all the consumers dependent on the system irrespective of the quantity of resources consumed from the system. For an infrastructure system, the ECS at time  $t$  is given by Equation-ref{eq:ecs}.

$$\text{ECS}_K(t) = \left( \sum_{\forall i: S_i^K(t) > 0} \frac{s_i^K(t)}{S_i^K(t)} \right) / n_K(t), \quad \text{where } 0 \leq s_i^K(t) \leq S_i^K(t)$$

where  $s_i$  is the resource supply at time  $t$  under stressed system conditions and  $n_K(t)$  is the number of consumers with a non-zero normal demand at time  $t$ .

In the case of PCS, the consumers are weighted by the quantity of resources drawn by them. This approach assumes that disruptions to serviceability of large-scale consumers, such as manufacturing sector, have larger effect to the whole region compared to small-scale consumers such as residential buildings. The PCS metric of an infrastructure system at time  $t$  is given by the Equation-ref{eq:pcs}.

$$\text{PCS}_K(t) = \left( \frac{\sum_{\forall i: S_i^K(t) > 0} s_i^K(t)}{\sum_{\forall i: S_i^K(t) > 0} S_i^K(t)} \right), \quad \text{where } 0 \leq s_i^K(t) \leq S_i^K(t)$$

The normal serviceability component ( $S_i^K(t)$ ) makes both ECS and PCS metrics unaffected by the intrinsic design inefficiencies as well as the temporal fluctuations in demand.

For water distribution systems, pressure-driven approach is chosen as it is reported to be most ideal for the hydraulic simulation under pressure deficient situations. The component resource supply values for water systems are computed as in Equations~ref{eq:water\_t\_pda}~ref{eq:water\_base\_pda}.

$$\begin{aligned} s_i^{water}(t) &= Q_i(t) \\ S_i^{water}(t) &= Q_i^0(t) \end{aligned}$$

where  $Q_i(t)$  and  $Q_i^0(t)$  are the water supplied to consumer  $i$  during stressed and normal system conditions, respectively.

For power systems, the power supplied to components under normal and stressed system conditions can be calculated using Equations~ref{eq:power\_t}~ref{eq:power\_base}.

$$\begin{aligned} s_i^{power}(t) &= p_i(t) \\ S_i^{power}(t) &= p_i^0(t) \end{aligned}$$

where  $p_i(t)$  and  $p_i^0(t)$  are the power supplied to consumer  $i$  under stressed and normal power system conditions.

The ECS and PCS time series can be used to profile the effect of the disruption on any of the infrastructure systems. To quantify the system-level cumulative performance loss, a resilience metric called Equivalent Outage Hours (EOH), based on the well-known concept of *resilience triangle*’ cite{Bruneau2003}, is introduced. *EOH of an infrastructure system due to disaster event*  $\gamma^K$  is calculated as in Equation~ref{eq:system\_eoh}.

$$\gamma^K = \frac{1}{3600} \int_{t_0}^{t_{max}} [1 - PCS_K(t)] dt \quad \text{or} \quad \gamma^K = \frac{1}{3600} \int_{t_0}^{t_{max}} [1 - ECS_K(t)] dt$$

where  $t_0$  is the time of the disaster event in the simulation and  $t_{max}$  is the maximum simulation time (both in seconds). In Equation~ref{eq:system\_eoh}, system performance during normal operating conditions is 1 due to the expression of the MOP used (see Equations ~ref{eq:ecs}~ref{eq:pcs}).

EOH of an infrastructure system can be interpreted as the duration (in hours) of a full infrastructure service outage that would result in an equivalent quantity of reduced consumption of the same service by all consumers during a disaster. The larger the EOH value, the larger the impact on the infrastructure system and thereby on the consumers due to the disruptive event. The EOH metric can effectively capture the response and resilience of the infrastructure system (Equation~ref{eq:system\_eoh}), according to the serviceability criteria chosen by the user.

Similar to EOH of a system, the consumer-level EOH can also be quantified, which indicates the equivalent duration of infrastructure service outage experienced by each consumer (Equation~ref{eq:consum\_eoh}).

$$\gamma_i^K = \int_{T_0}^T \left[ 1 - \frac{s_i^K(t)}{S_i^K(t)} \right] dt$$

Finally, in order to compute the resilience of the interdependent infrastructure network, a weighted EOH metric is derived (Equation~ref{eq:wEOH}).

$$\bar{\gamma} = \sum_{K \in \mathbb{K}} w^K \gamma^K$$

By default, equal weights are applied to both water and power systems.

## DATA REQUIREMENTS

Currently, network generation and hazard generation are not automated in the integrated simulation model. Therefore, in order to use the model, three types of data are to be manually fed into the model, namely the water, power and transportation networks, interdependency data, and infrastructure disruption data. These datasets need to be in specific formats which are compatible with the model. In this chapter, the details such as file formats and information to be included in the input files are discussed in detail.

### 3.1 Infrastructure Networks

The infrastructure network data must be compatible with the respective infrastructure model packages enlisted in Table 1. In addition, disruptions to certain components belonging to the infrastructure systems are not supported as of now. The individual networks must be constructed taking into the above aspects into consideration.

#### 3.1.1 Water distribution system

Since the integrated simulation model handles the water distribution network models using wntr package, the input file must be in .inp format. Some examples of water network files can be found in wntr Github repository<sup>1</sup>. The water network can be either built using EPANET or wntr package. Currently, the water network simulation is performed using WNTRSimulator in the wntr package. Therefore, there are certain exceptions which must be considered while generating the water network file. These exceptions can be found in wntr's software framework and limitations page.

In addition, the integrated simulation model identifies the component details such as infrastructure type, component type, etc. using the component names. The name of the components must follow the nomenclature. The details of nomenclature and whether a component can be failed in the model is presented in Table 2. For example, a water pump can be named as W\_WP1. Integers must be followed by the prefix to name components belonging to same category.

Table 2. Nomenclature for water network components

Prefix	Component name	Disruption supported	Prefix	Component name	Disruption supported
W_WP	Pump	Yes	W_PMA	Main Pipe	Yes
W_R	Reservoir	No	W_PHC	Hydrant Connection Pipe	Yes
W_P	Pipe	Yes	W_PV	Valve converted to Pipe	Yes
W_PSC	Service Connection Pipe	Yes	W_J	Junction	No
W_T	Tank	Yes			

### 3.1.2 Power systems

The power system is modeled using pandapower package, and therefore the network file must be in \*.json format. Several tutorials for generating power system networks using pandapower are available in the package's tutorial page<sup>3</sup>. Similar to water network components, naming of power system components must also follow the stipulated nomenclature presented in Table 3. Currently, power networks are modeled as three-phase systems. Therefore, single-phase components in pandapower are not supported. Integers must be followed by the prefix to name components belonging to same category.

Table 3. Nomenclature for power network components

Pre-fix	Component name	Disruption supported	Pre-fix	Component name	Disruption supported
P_B	Bus	Yes	P_L	Line	Yes
P_BLO	Bus connected to load	Yes	P_TF	Transformer	Yes
P_BS	Bus connected to switch	Yes	P_TFEG	Transformer connected to external grid	Yes
P_BEG	Bus connected to external grid connection	yes	P_S	Switch	Yes
P_LO	Load	Yes	P_SEG	Switch connected to external grid	Yes
P_MP	Motor	Yes	P_SLI	Switch connected to Line	Yes
P_G	Generator	No	P_EG	External Grid connection	Yes

### 3.1.3 Transportation system

The transportation system is simulated using the static traffic assignment model developed by Prof. Stephen Boyles of Department of Civil, Architectural and Environmental Engineering, The University of Texas at Austin. The transportation network data must be in the TNTP data format. More details on the format and example networks can be found in Ben Stabler's Github page<sup>4</sup>. The naming of the transportation network components must follow the nomenclature presented in Table 4. Integers must be followed by the prefix to name components belonging to same category.

Table 4. Nomenclature for transportation network components

Prefix	Component name	Disruption supported
T_J	Junction	No
T_L	Link	Yes

## 3.2 Infrastructure dependencies

Data related to water-power dependencies<sup>5</sup> between infrastructure components must be provided separately in a .csv file. The file must contain water\_id and power\_id fields which represent the water- and power component names (Table 5). The model will determine the type of the water and power components and construct the dependencies accordingly.



### 3.3 Infrastructure disruption data

The third data input is related to disrupted components. Similar to the dependency input file, the disruption data also needs to be provided in \*.csv format. The file should have three fields, namely time\_stamp (time of disruption in seconds), components (name of the component that is included in any of the three networks), and fail\_perc (percentage of damage). An example for the dependency data is presented in Table 6.

Table 6. Format of disruption data

water_id	power_id
W_WP9	P_MP1
W_R9	P_G3

While the percentage of damage of component is not used in the current model, this may be incorporated in the model to find the repair duration estimate in the future versions.



## INSTALLATION

### 4.1 From sources

The sources for dreaminsg-integrated-model can be downloaded from the [Github repo](#).

You can either clone the public repository:

```
$ git clone git://github.com/srijithabalakrishnan/dreaminsg_integrated_model
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

You may need to create a new Python environment that has all required packages and dependencies installed before start using the package. Run the following command.

```
$ conda env create --name infrarisk --file=environment.yml
```

We have observed that in certain instances, a few package conflicts when installing the environment. If so, please install the packages manually.



## GETTING STARTED

This chapter provides a step-by-step tutorial for running a model predictive control (MPC) method to identify the optimal repair strategy after a disruptive event. It is suggested to create a Jupyter notebook inside the notebooks folder as below:

```
$ cd notebooks
$ conda activate redcar
(infrarisk) $ conda install ipykernel
(infrarisk) $ ipython kernel install -user -name=redcar
```

The browser would open showing available notebook kernels. Select redcar from options to open a Jupyter notebook that can run the integrated simulation model.

### 5.1 Shelby county simulation demonstration

#### 5.1.1 Load required modules and libraries

Now run the following IPython extensions to reload all modules within the simulation package and set the Jupyter notebook.

```
%load_ext autoreload
%autoreload 2

from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

Now, load the required modules and external libraries manually as follows:

```
from pathlib import Path
from infrarisk.src.network_recovery import *
import infrarisk.src.simulation as simulation
from infrarisk.src.physical.integrated_network import *
import infrarisk.src.recovery_strategies as strategies
import infrarisk.src.socioeconomic.se_analysis as se_analysis
from infrarisk.src.physical.interdependencies import *

import infrarisk.src.plots as model_plots
```

(continues on next page)

(continued from previous page)

```
import warnings
warnings.filterwarnings('ignore')
```

## 5.1.2 Create an IntegratedNetwork object and load individual networks

In order to create the integrated infrastructure model, first we need to initiate the IntegratedNetwork object from the integrated\_network module. Then the individual models should be loaded one by one to the IntegratedNetwork object. For this study, we use a simplified integrated network consisting of a few nodes and links.

```
shelby_network = IntegratedNetwork(name = "Shelby")

MAIN_DIR = Path('../..')
SIM_STEP = 60

network_dir= 'infrarisk/data/networks/shelby'
water_folder = MAIN_DIR/f'{network_dir}/water'
power_folder = MAIN_DIR/f'{network_dir}/power'
transp_folder = MAIN_DIR/f'{network_dir}/transportation/reduced'

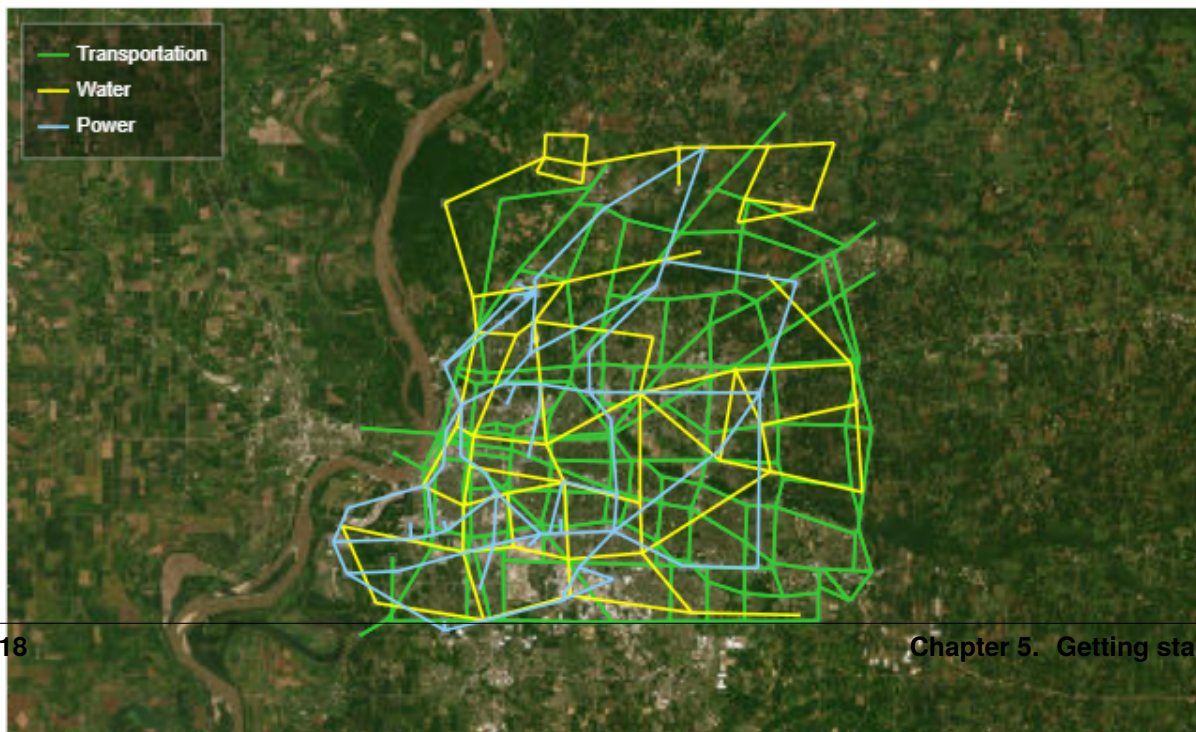
# load all infrastructure networks
shelby_network.load_networks(water_folder=water_folder,
                             power_folder=power_folder,
                             transp_folder=transp_folder,
                             sim_step=SIM_STEP)
```

Next, the individual infrastructure networks will be converted into a networkX object for plotting purposes.

```
shelby_network.generate_integrated_graph(basemap = True)
```

The above method will create the object and outputs the integrated graph (Fig. 4) consisting of the topologies of all the three infrastructure networks that were loaded.

Shelby integrated network



In the next step, we need to build the inter-dependencies within the integrated

net-  
work.  
In  
this  
model,  
three  
types  
of  
in-

terdependencies are considered.

1. Water pump on electric motor (water system on power system dependency)
2. Power generator on reservoir (power system on water system dependency)
3. All power and water system components on nearest transportation network node (power and water systems on transportation system dependency)

The information related to the first two types of dependencies must be provided in the form of csv file whereas the third set of dependencies will be automatically identified by the simulation model.

```
dependency_file = MAIN_DIR/f"{network_dir}/dependencies.csv"
shelby_network.generate_dependency_table(dependency_file = dependency_file)
```

The dependencies are referenced using two pandas dataframes in the model.

- `wp_table` stores water - power dependencies.
- `access_table` stores transportation access dependencies.

In order to view the `wp_table`, the following line of code may be implemented. It will return the table shown in Table 7.

```
shelby_network.dependency_table.wp_table.head()
```

water_id	power_id	water_type	power_type
W_WP81	P_MP81	Pump	Motor
W_WP82	P_MP82	Pump	Motor
W_WP83	P_MP83	Pump	Motor
W_WP84	P_MP84	Pump	Motor
W_WP85	P_MP85	Pump	Motor

Similarly, the `access_table` can be printed which will return a table as shown in Table 8.

```
shelby_network.dependency_table.access_table.head()
```

origin_id	transp_id	origin_cat	origin_type	access_dist
P_B40L0	T_J153	power	Bus connected to load	2202.69
P_B57L0	T_J153	power	Bus connected to load	3411.97
P_B2L0	T_J58	power	Bus connected to load	1262.27
P_B12L0	T_J183	power	Bus connected to load	1195.67
P_B49L0	T_J54	power	Bus connected to load	698.73

### 5.1.3 Load socioeconomic data for Shelby County

Before we initiate the simulations, we need to download the socio-economic data for Shelby county. The following code does that. Please note that socioeconomic analysis is only specific to Shlby county and may not work in the case of other available networks in this package. The user may use the socio-economic module for retrieving data for other counties in the United States.

```
year, tract, county, state = 2017, '*', 157, 47
county = 157
se_dir = MAIN_DIR/f"{network_dir}/gis/se_data"
if not os.path.exists(se_dir):
    os.makedirs(se_dir)

ShelbySE = se_analysis.SocioEconomicTable(name = 'Shelby', year = year,
                                           tract = tract, state = state,
                                           county = county, dir = se_dir)

ShelbySE.download_se_data(force_download = False)
ShelbySE.create_setable()
```

```
ShelbySE.plot_interactive(type = "annual receipts")
```

The above code would produce an interactive map showing the annual industrial output from various zipcodes of Shelby County in Tennessee.

### 5.1.4 Generate disaster scenario and set infrastructure component disruptions

The information regarding the dursuptive events are also stored in the IntegratedNetwork object. The data related to disrupted components and the level of damage must be provided in csv file format.

```
scenario_folder = f"scenarios/scenario1"
disruption_file = MAIN_DIR/f"{network_dir}/{scenario_folder}/disruption_file.dat"

shelby_network.set_disrupted_components(disruption_file=disruption_file)
disrupted_components = shelby_network.get_disrupted_components()
print(*disrupted_components, sep = ", ")
```

In this example, 14 components in the integrated network are failed, including six water pipes, six power lines and two road links. The disruptive events table can be returned using the following code.

```
shelby_network.get_disruptive_events()
```

The returned pandas dataframe would like the one below (Table 6). It shows the time (time\_stamp) in seconds at which the component (components) failed and the intensity of damage in percentage (fail\_perc).



### 5.1.5 Set initial locations of restoration crews

To perform the restoration and recovery of damaged or failed components after disruptive event occurs, each of the infrastructure agency has a repair crew. Once the disaster hits, each of these crews are sent to the locations of the damaged infrastructure components based on a recovery strategy. To set the initial locations of the water- power-, and transportation system repair crews, the following method is initiated.

```
crew_count = 10
shelby_network.deploy_crews(
    init_power_crew_locs=['T_J8']*crew_count,
    init_water_crew_locs=['T_J8']*crew_count,
    init_transpo_crew_locs= ['T_J8']*crew_count
)
```

In the current example, for simplicity and demonstration purpose, the initial locations of all the three repair crews are set as T\_J8, which is a transportation node (junction).

### 5.1.6 Simulation of restoration and recovery process

The next step is to create a `NetworkRecovery` object. The `NetworkRecovery` object stores recovery related information including the start times and end times of various repair actions.

```
network_recovery = NetworkRecovery(shelby_network,
                                   sim_step=SIM_STEP,
                                   pipe_close_policy="repair",
                                   pipe_closure_delay= 12*60,
                                   line_close_policy="sensor_based_line_isolation",
                                   line_closure_delay= 12*60)
```

Now, a `NetworkSimulation` object is created which include methods to perform actions to simulate direct and indirect impacts in the network due to the external event as well as recovery actions.

```
bf_simulation = simulation.NetworkSimulation(network_recovery)
```

Now, we define repair strategy and generate the repair sequence. Here, we adopt `HandlingCapacityStrategy` in which the repair sequence is generated based of the maximum daily infrastructure service flow handled by the component under normal operating conditions.

```
capacity_strategy = strategies.HandlingCapacityStrategy(shelby_network)
capacity_strategy.set_repair_order()
repair_order = capacity_strategy.get_repair_order()

import os
if not os.path.exists(MAIN_DIR/f"{network_dir}/{scenario_folder}/capacity"):
    os.makedirs(MAIN_DIR/f"{network_dir}/{scenario_folder}/capacity")
```

The next step is to schedule the recovery actions based on the repair sequence as follows:

```
bf_simulation.network_recovery.schedule_recovery(repair_order)
bf_simulation.expand_event_table()
```

Finally, we run the integrated infrastructure simulation model to simulate the indirect effects of the initial infrastructure disruptions. The results of the simulation (including the node-level water and power demands) are stored in the `ResilienceMetrics` object.

```
resilience_metrics = bf_simulation.simulate_interdependent_effects(
    bf_simulation.network_recovery)
```

Once the simulation is completed, the results can be saved in a csv file using the following code.

```
strategy = 'capacity'
bf_simulation.write_results(f"{MAIN_DIR}/{network_dir}/{scenario_folder}/{strategy}",
    resilience_metrics)
```

## 5.1.7 Quantification of loss using resilience metrics

The ResilienceMetrics object stores the results of the simulation. The following code calculates the water and power resilience values. The average network resilience is also calculated which is the weighted sum of the water and resilience values.

```
resilience_metrics.calculate_power_resmetric(network_recovery)
resilience_metrics.calculate_water_resmetrics(network_recovery)

resilience_metrics.set_weighted_auc_metrics()
```

In addition to the functional resilience values, we can also calculate the economic costs of business disruptions if the industry data is available for the network under study. The following code calculates the sector-wise economic costs of business disruptions.

```
ShelbySE.combine_infrastructure_se_data(shelby_network, resilience_metrics)
ShelbySE.calculate_economic_costs()
ShelbySE.economic_cost_df.to_csv(f"{MAIN_DIR}/{network_dir}/{scenario_folder}/{strategy}/
↳economic_cost.csv", index=False)
```

## 5.1.8 Plot simulation results

The network performance of water and power systems during the simulation can be visualized using the following code.

```
model_plots.plot_interdependent_effects(resilience_metrics, metric = 'pcs', title = "
↳False)
```

The following code plots the map of the network showing the initial disruptions and the initial locations of the repair crews.

```
model_plots.plot_disruptions_and_crews(shelby_network, basemap = True)
```

At the same time, the following code can be used to plot the equivalent outage hours (resilience metric to quantify operational loss) of the water and power systems in various buildings or service areas.

```
split_water_sa = gpd.overlay(shelby_network.wn.service_area, ShelbySE.county_gpd_
↳truncated, how='intersection')
split_power_sa = gpd.overlay(shelby_network.pn.service_area, ShelbySE.county_gpd_
↳truncated, how='intersection')
sa_dict = {'Water': split_water_sa, 'Power': split_power_sa}

model_plots.plot_region_impact_map(resilience_metrics, sa_dict, "capacity", extends = "
↳ShelbySE.bounds)
```

The following code produces an interactive map that shows the sector-wise economic loss of business disruptions due the simulated event.

```
ShelbySE.plot_interactive(type = "economic costs")
```

## 5.2 Additional tutorials

There are several Jupyter notebooks in `infrarisk/notebooks` that demonstrate the use of various features of the package, including generating hazards and interdependent infrastructure simulations based on other networks.



**SUPPORT**

The model is still in development stage. To report any issues in the model, suggest changes, or obtain support with installation or use, please write to [srijith.balakrishnan@sec.ethz.ch](mailto:srijith.balakrishnan@sec.ethz.ch)



## API DOCUMENTATION

### 7.1 infrarisk.src package

#### 7.1.1 Subpackages

##### 7.1.1.1 infrarisk.src.hazards package

##### 7.1.1.1.1 infrarisk.src.hazards.custom module

**class** `infrarisk.src.hazards.custom.CustomDisruption`(*affected\_components*, *name*='User defined disruptions', *time\_of\_occurrence*=6000)

Bases: object

**generate\_disruption\_file**(*location*=None, *folder\_extra*=None, *minimum\_data*=0, *maximum\_data*=None)

Generates the disruption file consisting of the list of failed components, time of occurrence, and failure percentage (damage extent).

**Parameters** *location* (*string*) – The location of the file to be saved.

**get\_fail\_compon\_dict**()

Returns the dictionary of that could be failed due to a radial disaster.

**Returns** dictionary of components that could be failed.

**Return type** dictionary

**plot\_disruptions**(*integrated\_graph*)

**set\_affected\_components**(*G*, *plot\_components*=True)

**set\_fail\_compon\_dict**()

Sets the dictionary of components that could be failed due to a radial disaster.

**set\_time\_of\_occurrence**(*time\_of\_occurrence*)

Sets the time of occurrence of the disruptive event.

**Parameters** *time\_of\_occurrence* (*integer*) – Time in seconds and multiple of 60.

### 7.1.1.1.2 infrarisk.src.hazards.fragility\_based module

Class of disruption based on fragility analysis

```
class infrarisk.src.hazards.fragility_based.FragilityBasedDisruption(resilience_level='high',  
                                                                    name='Fragility based  
                                                                    disruption',  
                                                                    fragility_df=None,  
                                                                    time_of_occurrence=6000)
```

Bases: object

**ascertain\_damage\_probabilities**(*component, imt\_type, imt\_value, plotting=True*)

Ascertains the damage of a component type

**Parameters**

- **component** (*str*) – The component type
- **imt\_value** (*float*) – The intensity measure value

**ascertain\_pipe\_damage\_probabilities**(*wn, pipe, gmf\_gpd, method=1, plotting=True*)

**assign\_recovery\_time**(*component, failure\_state*)

Assigns the recovery time of a component based on the failure state.

**Parameters**

- **component** (*str*) – The component identifier
- **failure\_state** (*str*) – The failure state of the component

**Returns** The recovery time of the component

**Return type** float

**static calculate\_state\_probability**(*imt, median, beta*)

Calculates the damage state cumulative probability from the fragility functions.

**Parameters**

- **imt** (*string*) – The intensity measure used in the fragility curves.
- **median** (*float*) – The median parameter of the fragility curve corresponding to the damage state.
- **beta** (*float*) – The beta parameter of the fragility curve corresponding to the damage state.

**Returns** The cumulative probability distribution value corresponding to the damage state.

**Return type** float

**generate\_bokeh\_disruptions**(*integrated\_graph, infra, map\_extends*)

**generate\_disruption\_file**(*location=None, folder\_extra=None, minimum\_data=0,*  
 *maximum\_data=None*)

Generates the disruption file consisting of the list of failed components, time of occurrence, and failure percentage (damage extent).

**Parameters** **location** (*string*) – The location of the file to be saved.

**get\_affected\_components**()

Returns the disrupted infrastructure components.

**Returns** The disrupted infrastructure components along with their disruption states probabilities.



**Return type** pandas.DataFrame object

**get\_fail\_compon\_dict()**

Returns the dictionary of the component types that fail

**Returns** The dictionary of the component types in each infrastructure system that must be considered for disruption

**Return type** dict

**static get\_int\_at\_line**(*start\_coords*, *end\_coords*, *imt\_type*, *gmf\_gpd*)

Returns the maximum intensity measure (PGA, PGV, or PGD) along a line if corresponding fragility curves are available.

**Parameters**

- **start\_coords** (*list*) – The coordinates of the start point (latitude, longitude) in UTM
- **end\_coords** (*list*) – The coordinates of the end point (latitude, longitude) in UTM
- **imt\_type** (*str*) – The intensity measure type (PGA, PGV, PGD)
- **gmf\_gpd** (*geopandas.GeoDataFrame*) – The ground motion field data

**Returns** The maximum intensity measure along the line

**Return type** float

**static get\_int\_at\_point**(*coords*, *imt\_type*, *gmf\_gpd*)

Returns the intensity measure (PGA, PGV, or PGD) at a point if corresponding fragility curves are available.

**Parameters**

- **coords** (*list*) – The coordinates of the point (latitude, longitude) in UTM
- **imt\_type** (*str*) – The intensity measure type (PGA, PGV, PGD)
- **gmf\_gpd** (*geopandas.GeoDataFrame*) – The ground motion field data

**Returns** The intensity measure at the point

**Return type** float

**get\_int\_dict()**

Returns the mapping between infrastructure components and corresponding imt measures as a dictionary.

**Returns** The dictionary with component prefix as the keys and imt measure types as the values.

**Return type** dictionary

**plot\_disruptions**(*integrated\_graph*, *map\_extends*)

**plot\_failure\_distributions()**

**plot\_int**(*gmf\_gpd*, *imt\_column*)

Plots the fragility curves based on intensity measure type (PGA, PGV, or PGD) on the map.

**Parameters**

- **gmf\_gpd** (*geopandas.GeoDataFrame*) – The ground motion field data
- **imt\_column** (*str*) – The intensity measure type (PGA, PGV, PGD)

**plot\_state\_probabilities**(*imt\_type*, *imt\_value*, *compon\_type*, *state\_cdf*)

Plots the probabilities of a component being in different damage states for a given intensity measure value

**Parameters**

- **imt\_type** (*str*) – The intensity measure type (PGA, PGV, PGD)

- **imt\_value** (*float*) – The intensity measure value
- **compon\_type** (*str*) – The component type
- **state\_cdf** (*list*) – The cumulative probabilities of the component being in different damage states

**set\_affected\_components**(*integrated\_network, gmf\_gpd, disruption\_time, plot\_components=True*)  
Determines the disrupted infrastructure components using the calculated damage state probabilities.

**Parameters**

- **G** (*IntegratedNetwork.integrated\_graph object*) – The networkx integrated graph object
- **gmf\_gpd** (*geopandas.GeoDataFrame object*) – The geopandas GeoDataFrame that consists of the ground motion fields at available geographic points.
- **disruption\_time** (*integer*) – The time of disruption in seconds
- **plot\_components** (*bool, optional*) – boolean to determine whether to generate the disruption plots, defaults to True

**set\_all\_fragility\_and\_recovery\_curves**()

Creates a dictionary of all fragility curves that will be used in generating the disruptions.

**set\_compon\_fragility\_curves**(*compon\_prefix, imt, list\_of\_states, list\_of\_ds\_median, list\_of\_ds\_beta*)  
Sets the fragility curves for a component type

**Parameters**

- **compon\_prefix** (*str*) – The prefix of the component type
- **imt** (*str*) – The intensity measure type
- **list\_of\_states** (*list*) – The list of states
- **list\_of\_ds\_median** (*list*) – The list of median values of the fragility curves in the order of states
- **list\_of\_ds\_beta** (*list*) – The list of beta values of the fragility curves in the order of states

**set\_compon\_recovery\_curves**(*compon\_prefix, list\_of\_states, list\_of\_recov\_mean, list\_of\_recov\_sigma*)  
Sets the recovery curves for a component type

**Parameters**

- **compon\_prefix** (*str*) – The prefix of the component type
- **list\_of\_states** (*list*) – The list of states
- **list\_of\_recov\_mean** (*list*) – The list of mean values of the recovery curves in the order of states
- **list\_of\_recov\_sigma** (*list*) – The list of sigma values of the recovery curves in the order of states

**set\_fail\_compon\_dict**(*fail\_compon\_dict*)

Sets the dictionary of the component types that fail

**Parameters** **fail\_compon\_dict** (*dict*) – The dictionary of the component types in each infrastructure system that must be considered for disruption.

**set\_gmfs**(*G, gmf\_gpd*)

Calculates the ground motion field values of all components that must be considered for disruptions.

#### Parameters

- **G** (*IntegratedNetwork.integrated\_graph object*) – The integrated graph object
- **gmf\_gpd** (*geopandas.GeoDataFrame object*) – The geopandas GeoDataFrame that consists of the ground motion fields at available geographic points.

#### **set\_int\_dict()**

Sets the mapping between infrastructure components and corresponding int measures used in fragility curves, if available.

#### **set\_link\_gmfs(G, gmf\_gpd)**

Calculates the ground motion field values of all link (edge) components that must be considered for disruptions.

#### Parameters

- **G** (*IntegratedNetwork.integrated\_graph object*) – The integrated graph object
- **gmf\_gpd** (*geopandas.GeoDataFrame object*) – The geopandas GeoDataFrame that consists of the ground motion fields at available geographic points.

#### **set\_node\_gmfs(G, gmf\_gpd)**

Calculates the ground motion field values of all node components that must be considered for disruptions.

#### Parameters

- **G** (*IntegratedNetwork.integrated\_graph object*) – The integrated graph object
- **gmf\_gpd** (*geopandas.GeoDataFrame object*) – The geopandas GeoDataFrame that consists of the ground motion fields at available geographic points.

### 7.1.1.1.3 infrarisk.src.hazards.radial module

```
class infrarisk.src.hazards.radial.RadialDisruption(name='Radial disruption',
                                                    point_of_occurrence=None,
                                                    radius_of_impact=100,
                                                    time_of_occurrence=6000, intensity='high')
```

Bases: object

Class of disaster where the probability of failure of components reduces with distance from the point of occurrence of the event.

#### **assign\_link\_failure(p\_occ, start\_coords, end\_coords)**

Assigns the link failure status.

#### Parameters

- **p\_occ** (*shapely Point object*) – The point of occurrence of the disruptive event.
- **start\_coords** (*tuple*) – The coordinates of the start node of the link.
- **end\_coords** (*tuple*) – The coordinates of the end node of the link.

**Returns** The failure status of the link

**Return type** bool

#### **assign\_node\_failure(p\_occ, node\_point)**

Assigns node failure status.

#### Parameters

- **p\_occ** (*shapely Point object*) – The point of occurrence of the disruptive event.

- **node\_point** (*shapely Point object*) – The node for which the failure status is to be assigned.

**Returns** The failure status of the node.

**Return type** bool

**generate\_disruption\_file**(*location=None, folder\_extra=None, minimum\_data=0, maximum\_data=None*)

Generates the disruption file consisting of the list of failed components, time of occurrence, and failure percentage (damage extent).

**Parameters** **location** (*string*) – The location of the file to be saved.

**get\_fail\_compon\_dict**()

Returns the dictionary of that could be failed due to a radial disaster.

**Returns** dictionary of components that could be failed.

**Return type** dictionary

**plot\_disruptions**(*integrated\_graph*)

**set\_affected\_components**(*G, plot\_components=True*)

Set the affected components (nodes and link)

**Parameters**

- **G** (*Networkx object*) – The infrastructure network as a networkx graph.
- **plot\_components** (*bool, optional*) – plots affected components, defaults to True

**set\_fail\_compon\_dict**(*fail\_compon\_dict={'power': ['L'], 'transport': ['L'], 'water': ['PMA', 'WP']}*)

Sets the dictionary of components that could be failed due to a radial disaster.

**Parameters** **fail\_compon\_dict** (*\_type\_*) – A dictionary of the prefixes of different infrastructure components that must be considered for failing

**set\_intensity\_failure\_probability**()

Sets the vulnerability (probability of failure) based on the intensity of the disaster event (currently arbitrary values are used).

**set\_point\_of\_occurrence**(*point\_of\_occurrence*)

Sets the point of occurrence of the radial disruption.

**Parameters** **point\_of\_occurrence** (*tuple, optional*) – The central point (represented by a tuple of longitude and latitude) of the disruptive event, defaults to None

**set\_radius\_of\_impact**(*radius\_of\_impact*)

Sets the radius of the radial disruption.

**Parameters** **radius\_of\_impact** (*float or integer, optional*) – The radius of the impact (he probability of failure at the circumference reacher zero) in metres., defaults to None

**set\_time\_of\_occurrence**(*time\_of\_occurrence*)

Stes the time of occurrence of the disruptive event.

**Parameters** **time\_of\_occurrence** (*integer*) – Time in seconds and multiple of 60.

#### 7.1.1.1.4 infrarisk.src.hazards.random module

A class for random network disruptions

```
class infrarisk.src.hazards.random.RandomDisruption(failure_count={'power': 1, 'transpo': 1, 'water': 1},
                                                    compon_scope=None,
                                                    time_of_occurrence=6000, name='Random
                                                    disruption')
```

Bases: object

```
generate_disruption_file(location=None, folder_extra=None, minimum_data=0,
                          maximum_data=None)
```

Generates the disruption file consisting of the list of failed components, time of occurrence, and failure percentage (damage extent).

**Parameters** **location** (*string*) – The location of the file to be saved.

```
get_fail_compon_dict()
```

Returns the dictionary of that could be failed due to a radial disaster.

**Returns** dictionary of components that could be failed.

**Return type** dictionary

```
plot_disruptions(integrated_graph)
```

```
set_affected_components(G, plot_components=True)
```

```
set_fail_compon_dict(fail_compon_dict={'power': ['L'], 'transport': ['L'], 'water': ['PMA', 'WP']})
```

Sets the dictionary of components that could be failed due to a radial disaster.

**Parameters** **fail\_compon\_dict** (*\_type\_*) – A dictionary of the prefixes of different infrastructure components that must be considered for failing

```
set_time_of_occurrence(time_of_occurrence)
```

Sets the time of occurrence of the disruptive event.

**Parameters** **time\_of\_occurrence** (*integer*) – Time in seconds and multiple of 60.

#### 7.1.1.1.5 infrarisk.src.hazards.track module

```
class infrarisk.src.hazards.track.TrackDisruption(hazard_tracks=None, buffer_of_impact=25,
                                                    time_of_occurrence=6000, intensity='high',
                                                    name='Track disruption')
```

Bases: object

Class of flood disaster where the probability of failure of components reduces with the longitudinal distance from the track of the event.

```
assign_link_failure(track, link_line)
```

Assigns the link failure status.

**Parameters**

- **track** (*shapely LineString object*) – The track of the disruptive event.
- **link\_line** (*shapely LineString object*) – The link for which the failure status is to be assigned.

**Returns** The failure status of the link.

**Return type** bool

**assign\_node\_failure**(*track, node\_point*)

Assigns node failure status.

**Parameters**

- **track** (*shapely LineString object*) – The track of the disruptive event.
- **node\_point** (*shapely Point object.*) – The node for which the failure status is to be assigned.

**Returns** The failure status of the node.

**Return type** bool

**generate\_disruption\_file**(*location=None, folder\_extra=None, minimum\_data=0, maximum\_data=None*)

Generates the disruption file consisting of the list of failed components, time of occurrence, and failure percentage (damage extent).

**Parameters** **location** (*string*) – The location of the file to be saved.

**generate\_random\_track**(*loc\_extents, shape='spline'*)

Generates a random track using a spline connecting three points on the map. :param loc\_extents: The [(xmin, ymin), (xmax, ymax)] coordinates from the map denoting the occurrence of the event. :type loc\_extents: list of tuples :param shape: The method of generating the track. If “line”, generates a straight line, if “spline”, generates a smooth curve. :type shape: string :return: The disaster track. :rtype: shapely LineString object

**get\_fail\_compon\_dict**()

Returns the dictionary of that could be failed due to a radial disaster.

**Returns** dictionary of components that could be failed.

**Return type** dictionary

**plot\_disruptions**(*integrated\_graph*)

**set\_affected\_components**(*G, plot\_components=True*)

Set the affected components (nodes and link)

**Parameters**

- **G** (*Networkx object*) – The infrastructure network as a networkx graph.
- **plot\_components** (*bool, optional*) – plots affected components, defaults to True

**set\_buffer\_of\_impact**(*buffer\_of\_impact*)

Sets the impact buffer distance in meters.

**Parameters** **buffer\_of\_impact** (*integer or float*) – Impact buffer distance in meters.

**set\_fail\_compon\_dict**(*fail\_compon\_dict={'power': ['L'], 'transport': ['L'], 'water': ['PMA', 'WP']}*)

Sets the dictionary of components that could be failed due to a radial disaster.

**Parameters** **fail\_compon\_dict** (*\_type\_*) – A dictionary of the prefixes of different infrastructure components that must be considered for failing

**set\_hazard\_tracks\_from\_linestring**(*linestring\_track*)

Sets a hazard track from a LineString object.

**Parameters** **linestring\_track** (*LineString object*) – A shapely LineString object denoting the track of the hazard.

**set\_hazard\_tracks\_from\_shapefile**(*hazard\_tracks*)

Sets the tracks of the track-based hazard from a shapefile.

**Parameters** `hazard_tracks` (*shapefile*) – A shapefile that has tracks of the event as LineString objects.

**set\_intensity\_failure\_probability()**

Sets the vulnerability (probability of failure) based on the intensity of the disaster event (currently arbitrary values are used).

**set\_time\_of\_occurrence**(*time\_of\_occurrence*)

Sets the time of occurrence of the disruptive event.

**Parameters** `time_of_occurrence` (*integer*) – Time in seconds and multiple of 60.

### 7.1.1.2 infrarisk.src.physical package

#### 7.1.1.2.1 Subpackages

##### 7.1.1.2.1.1 infrarisk.src.physical.power package

##### 7.1.1.2.1.2 Submodules

##### 7.1.1.2.1.3 infrarisk.src.physical.power.power\_system\_model module

Functions to implement power systems simulations.

`infrarisk.src.physical.power.power_system_model.generate_base_supply(pn)`

`infrarisk.src.physical.power.power_system_model.get_power_control_dict()`

`infrarisk.src.physical.power.power_system_model.get_power_dict()`

Creates a dictionary of major power system components in a network. Used for naming automatically generated networks.

**Returns** Mapping of infrastructure component abbreviations to names.

**Return type** dictionary of string: dictionary of string: string

`infrarisk.src.physical.power.power_system_model.load_power_network(network_json, sim_type='1ph')`

Loads the power system model from a json file.

**Parameters**

- **network\_json** (*string*) – Location of the json power system file generated by pandapower package.
- **sim\_type** (*string*) – Type of power flow simulation: '1ph': single phase, '3ph': three phase.

**Returns** The loaded power system model object.

**Return type** pandapower network object

`infrarisk.src.physical.power.power_system_model.run_power_simulation(pn)`

Runs the power flow model for an instance.

**Parameters** `pn` (*pandapower network object*) – A power system model object generated by pandapower package.

#### 7.1.1.2.1.4 Module contents

#### 7.1.1.2.1.5 `infrarisk.src.physical.transportation` package

#### 7.1.1.2.1.6 `infrarisk.src.physical.transportation.network` module

**exception** `infrarisk.src.physical.transportation.network.BadNetworkOperationException`

Bases: `Exception`

You can raise this exception if you try a network action which is invalid (e.g., trying to find a topological order on a network with cycles.)

**class** `infrarisk.src.physical.transportation.network.Network`(*networkFile*="", *demandFile*="", *nodeFile*="")

Bases: `object`

This is the class used for transportation networks. It uses the following dictionaries to store the network; the keys are IDs for the network elements, and the values are objects of the relevant type:

`node` – network nodes; see `node.py` for description of this class  
`link` – network links; see `link.py` for description of this class  
`ODpair` – origin-destination pairs; see `od.py`  
`path` – network paths; see `path.py`.  
 Paths are NOT automatically generated

when the network is initialized (you probably wouldn't want this, the number of paths is exponential in network size.)

The network topology is expressed both in links (through the tail and head nodes) and in nodes (`forwardStar` and `reverseStar` are Node attributes storing the IDs of entering and leaving links in a list).

`numNodes`, `numLinks`, `numZones` – self-explanatory  
`firstThroughNode` – in the TNTF data format, transiting through nodes with

low IDs can be prohibited (typically for centroids; you may not want vehicles to use these as “shortcuts”). When implementing shortest path or other routefinding, you should prevent trips from using nodes with lower IDs than `firstThroughNode`, unless it is the destination.

**FrankWolfeStepSize**(*targetFlows*, *precision*=0.0001)

This method returns the step size `lambda` used by the Frank-Wolfe algorithm.

The current link flows are given in the `self.link[ij].flow` attributes, and the target flows are given in the `targetFlows` dictionary.

The precision argument dictates how close your method needs to come to finding the exact Frank-Wolfe step size: you are fine if the absolute difference between the true value, and the value returned by your method, is less than precision.

**acyclicShortestPath**(*origin*)

This method finds the shortest path in an acyclic network, from the stated origin. You can assume that a topological order has already been found, and referred to in the ‘order’ attributes of network Nodes. You can also find a list of nodes in topological order in `self.topologicalList`. (See the method `createTopologicalList` below.)

Use the ‘cost’ attribute of the Links to calculate travel times. These values are given – do not try to recalculate them based on flows, BPR functions, etc.

Be aware that both the order Node attribute and `topologicalList` respect the usual convention in network modeling that the topological order starts at 1, whereas Python starts numbering at 0.

The implementation in the text uses a vector of backnode labels. In this assignment, you should use back-LINK labels instead. The idea is exactly the same, except you are storing the ID of the last *link* in a shortest



path to each node.

The backlink and cost labels are both stored in dict's, whose keys are node IDs.

**\* BE SURE YOUR IMPLEMENTATION RESPECTS THE FIRST THROUGH NODE! \*** Travelers should not be able to use “centroid connectors” as shortcuts, **\*\*\*** and the shortest path tree should reflect this.

You should use the macro `utils.NO_PATH_EXISTS` to initialize backlink labels, and `utils.INFINITY` to initialize cost labels.

#### **allOrNothing()**

This method generates an all-or-nothing assignment using the current link cost values. It must do the following:

1. Find shortest paths from all origins to all destinations
2. For each OD pairs in the network, load its demand onto the shortest path found above. (Ties can be broken arbitrarily.)

The resulting link flows should be returned in the `allOrNothing` dict, whose keys are the link IDs.

Be aware that the network files are in the TNTF format, where nodes are numbered starting at 1, whereas Python starts numbering at 0.

Your code will not be scored based on efficiency, but you should think about different ways of finding an all-or-nothing loading, and how this might best be done.

#### **averageExcessCost()**

This method should calculate the average excess cost based on the current link flows, and return this value.

To do this, you will need to calculate both the total system travel time, and the shortest path travel time (you will find it useful to call some of the methods implemented in earlier assignments).

#### **beckmannFunction()**

This method evaluates the Beckmann function at the current link flows.

#### **calculateShortestTravelTime(*origin, destination*)**

#### **createTopologicalList()**

Takes a topological ordering of the nodes, expressed by the ‘order’ attribute of the Node objects, and creates a single list which stores the IDs of the nodes in topological order. This is essentially the inverse function of the topological order, the k-th element of this list gives you the ID of the node whose order value is k.

#### **finalize()**

Establish the forward and reverse star lists for nodes, initialize flows and costs for links and OD pairs.

#### **findLeastEnteringLinks()**

This method should return the ID of the node with the *least* number of links entering the node. Ties can be broken arbitrarily.

#### **findTopologicalOrder()**

This method should find a topological order for the network, storing the order in the ‘order’ attribute of the nodes, i.e.:

```
self.node[5].order
```

should store the topological label for node 5.

The topological order is generally not unique, this method can return any valid order. The nodes should be labeled 1, 2, 3, ... up through numNodes.

If the network has cycles, a topological order does not exist. The presence of cycles can be detected in the algorithm for finding a topological order, and you should raise an exception if this is detected.

### **formAdjacencyMatrix()**

This method should produce an adjacency matrix, with rows and columns corresponding to each node, and entries of 1 if there is a link connecting the row node to the column node, and 0 otherwise. This matrix should be stored in `self.adjacencyMatrix`, which is a dictionary of dictionaries: the first key is the “row” (tail) node, and the second key is the “column” (head) node.

### **loadPaths()**

This method should take given values of path flows (stored in the `self.path[].flow` attributes), and do the following:

1. Set link flows to correspond to these values (`self.link[].flow`)
2. Set link costs based on new flows (`self.link[].cost`), see `link.py`
3. Set path costs based on new link costs (`self.path[].cost`), see `path.py`

### **readDemandFile(*demandFileName*)**

Reads demand (OD matrix) data from a file in the TNTF format.

### **readFromFiles(*networkFile*, *demandFile*)**

Reads network data from a pair of files (`networkFile`, containing the topology, and `demandFile`, containing the OD matrix), then do some basic checks on the input data (validate) and build necessary data structures (finalize).

### **readNetworkFile(*networkFileName*)**

Reads network topology data from the TNTF data format. In keeping with this format, the zones/centroids are assumed to have the lowest node IDs (1, 2, ..., `numZones`).

### **relativeGap()**

This method should calculate the relative gap (as defined in the course text) based on the current link flows, and return this value.

To do this, you will need to calculate both the total system travel time, and the shortest path travel time (you will find it useful to call some of the methods implemented in earlier assignments).

### **shiftFlows(*targetFlows*, *stepSize*)**

This method should update the flow on each link, by taking a weighted average of the current link flows (`self.link[ij].flow`) and the flows given in the `targetFlows` dictionary (`targetFlows[ij]`). `stepSize` indicates the weight to place on the target flows (so the weight on the current flows is  $1 - \text{stepSize}$ ).

**\* IMPORTANT: After updating the flow on a link, you should call its `updateCost` method, so that the travel time is updated to reflect the new flow value. \***

This method does not need to return a value.

### **shortestPath(*origin*)**

This method finds the shortest path in a network which may or may not have cycles; thus you cannot assume that a topological order exists.

The implementation in the text uses a vector of backnode labels. In this assignment, you should use back-LINK labels instead. The idea is exactly the same, except you are storing the ID of the last *link* in a shortest path to each node.

Use the ‘cost’ attribute of the Links to calculate travel times. These values are given – do not try to recalculate them based on flows, BPR functions, etc.

The backlink and cost labels are both stored in dict’s, whose keys are node IDs.

**\* BE SURE YOUR IMPLEMENTATION RESPECTS THE FIRST THROUGH NODE! \*** Travelers should not be able to use “centroid connectors” as shortcuts, **\*\*\*** and the shortest path tree should reflect this.

You should use the macro `utils.NO_PATH_EXISTS` to initialize backlink labels, and `utils.INFINITY` to initialize cost labels.

**userEquilibrium**(*stepSizeRule*='MSA', *maxIterations*=100, *targetGap*=1e-06, *gapFunction*=<function *Network.relativeGap*>)

This method uses the (link-based) convex combinations algorithm to solve for user equilibrium. Arguments are the following:

**stepSizeRule** – a string specifying how the step size  $\lambda$  is to be chosen. Currently 'FW' and 'MSA' are the available choices, but you can implement more if you want.

**maxIterations** – stop after this many iterations have been performed **targetGap** – stop once the gap is below this level **gapFunction** – pointer to the function used to calculate gap. After

finishing this assignment, you should be able to choose either **relativeGap** or **averageExcessCost**.

**validate()**

Perform some basic validation checking of network, link, and node data to ensure reasonableness and consistency.

#### 7.1.1.2.1.7 `infrarisk.src.physical.transportation.transpo_compons` module

```
class infrarisk.src.physical.transportation.transpo_compons.Link(network, tail, head,
                                                                capacity=99999,
                                                                length=99999,
                                                                freeFlowTime=99999,
                                                                fft_base=99999, alpha=0.15,
                                                                beta=4, speedLimit=99999,
                                                                toll=0, linkType=0)
```

Bases: object

Class for network links. As currently written, assumes costs are calculated as the sum of three factors:

1. Travel time, computed via the BPR function
2. Toll cost, the product of toll and `network.tollFactor`
3. Distance-related costs, the product of length and `network.distanceFactor`

**calculateBeckmannComponent()**

Calculates the integral of the BPR function for the link, for its contribution to the sum in the Beckmann function.

**calculateCost()**

Calculates the cost of the link using the BPR relation, adding in toll and distance-related costs. This cost is returned by the method and NOT stored in the cost attribute.

**updateCost()**

Same as `calculateCost`, except that the `link.cost` attribute is updated as well.

```
class infrarisk.src.physical.transportation.transpo_compons.Node(isZone=False)
```

Bases: object

```
class infrarisk.src.physical.transportation.transpo_compons.OD(origin, destination, demand=0)
```

Bases: object

```
class infrarisk.src.physical.transportation.transpo_compons.Path(links, network, flow=0)
```

Bases: object

A Path is an ordered sequence of adjacent links; these are expressed in the attribute ‘links’, which is a tuple of link IDs. Other attributes are as follows:

network points to the parent Network class (needed for calculating costs) flow is the number of vehicles using this path cost is the total cost of the path

**calculateCost()**

Calculates the cost of the path by summing the cost of its constituent links. This cost is returned by the method and NOT stored in the cost attribute.

**updateCost()**

Same as calculateCost, except that the path.cost attribute is updated as well.

`infrarisk.src.physical.transportation.transpo_compons.get_transpo_dict()`

### 7.1.1.2.1.8 infrarisk.src.physical.transportation.utils module

**exception** `infrarisk.src.physical.transportation.utils.BadFileFormatException`

Bases: Exception

This exception is raised if a network or demand file is in the wrong format or expresses an invalid network.

**exception** `infrarisk.src.physical.transportation.utils.NotYetAttemptedException`

Bases: Exception

This exception is used as a placeholder for code you should fill in.

`infrarisk.src.physical.transportation.utils.path2linkTuple(pathString)`

Converts a path expressed as a sequence of nodes, e.g. [1,2,3,4] into a tuple of link IDs for use with the Path object (see path.py), in this case ((1,2),(2,3),(3,4))

`infrarisk.src.physical.transportation.utils.readMetadata(lines)`

Read metadata tags and values from a TNTP file, returning a dictionary whose keys are the tags (strings between the <> characters) and corresponding values. The last metadata line (reading <END OF METADATA>) is stored with a value giving the line number this tag was found in. You can use this to proceed with reading the rest of the file after the metadata.

### 7.1.1.2.1.9 Module contents

### 7.1.1.2.1.10 infrarisk.src.physical.water package

### 7.1.1.2.1.11 infrarisk.src.physical.water.water\_network\_model module

Functions to implement water network simulations.

`infrarisk.src.physical.water.water_network_model.add_control_system(wn, list_of_tanks, list_of_pumps, controller_id=None)`

Adds a controller system to the water network.

**Parameters**

- **wn** (*wntr water network object*) – Water network model object.
- **list\_of\_tanks** (*list*) – List of tanks controlled by the controller system.
- **list\_of\_pumps** (*list*) – List of pumps controlled by the controller system.
- **id** (*integer*) – The ID of the controller to be added.

```
infrarisk.src.physical.water.water_network_model.add_discharge_pipe(wn, tank_id, elevation)
```

Adds a discharge pipe to the tank of interest.

#### Parameters

- **wn** (*wntr water network object*) – Water network model object.
- **tank\_id** (*string*) – The ID of the tank to which the discharge pipe is to be added.
- **elevation** (*float*) – The elevation of the discharge pipe.

```
infrarisk.src.physical.water.water_network_model.add_discharge_pipe_actuator_system(wn,  
                                                                                   con-  
                                                                                   troller_id,  
                                                                                   list_of_tanks)
```

Adds a discharge pipe actuator system to the water network.

**Parameters** **wn** (*wntr water network object*) – Water network model object.

```
infrarisk.src.physical.water.water_network_model.add_pump_actuator_system(wn, controller_id,  
                                                                           list_of_pumps)
```

Adds a pump actuator system to the water network.

**Parameters** **wn** (*wntr water network object*) – Water network model object.

```
infrarisk.src.physical.water.water_network_model.add_tank_level_sensor_system(wn,  
                                                                              controller_id,  
                                                                              list_of_tanks)
```

Adds a sensor system to the water network.

#### Parameters

- **wn** (*wntr water network object*) – Water network model object.
- **controller\_id** (*string*) – The ID of the controller to which the sensor system is to be added.
- **list\_of\_tanks** (*list*) – List of tank IDs for which tank level sensors are to be assigned.

```
infrarisk.src.physical.water.water_network_model.add_universal_control_system(wn, con-  
                                                                           troller_id=None)
```

Adds single control system to the water network.

**Parameters** **wn** (*wntr water network object*) – Water network model object.

```
infrarisk.src.physical.water.water_network_model.generate_base_supply(wn_original, directory)
```

Runs demand driven simulation (DDA) simulation under normal network conditions and stores the node demands.

#### Parameters

- **wn** (*wntr water network object*) – Water network model object.
- **directory** (*string*) – The directory to which the node demands are to be saved.

```
infrarisk.src.physical.water.water_network_model.generate_base_supply_pda(wn_original, dir)
```

Runs pressure-dependent demand simulation (PDA) simulation under normal network conditions and stores the node demands.

#### Parameters

- **wn** (*wntr water network object*) – Water network model object.
- **dir** (*string*) – The directory to which the node demands are to be saved.

`infrarisk.src.physical.water.water_network_model.generate_pattern_interval_dict(wn)`  
`_summary_`

**Parameters** *wn* (*\_type\_*) – *\_description\_*

`infrarisk.src.physical.water.water_network_model.get_valves_to_isolate_water_node(integrated_network,  
node)`

`infrarisk.src.physical.water.water_network_model.get_water_control_dict()`

Creates a dictionary of major water distribution system control components.

**Returns** Mapping of control component abbreviations to names.

**Return type** dictionary of string: string

`infrarisk.src.physical.water.water_network_model.get_water_dict()`

Creates a dictionary of major water distribution system components.

**Returns** Mapping of infrastructure component abbreviations to names.

**Return type** dictionary of string: string

`infrarisk.src.physical.water.water_network_model.load_water_network(network_inp,  
water_sim_type,  
initial_sim_step,  
cyber_layer=False)`

Loads the water network model from an inp file.

**Parameters**

- **network\_inp** (*string*) – Location of the inp water network file.
- **water\_sim\_type** – The type of water simulation in wntr. Available options are pressure-dependent demand analysis ('PDA') and demand driven analysis ('DDA').
- **initial\_sim\_step** (*integer*) – The initial iteration step size in seconds.

**Returns** The loaded water wntr network object.

**Return type** wntr network object

`infrarisk.src.physical.water.water_network_model.run_water_simulation(wn)`

Runs the simulation for one time step.

**Parameters** *wn* (*wntr water network object*) – Water network model object.

**Returns** Simulation results in pandas tables.

**Return type** ordered dictionary of string: pandas table

`infrarisk.src.physical.water.water_network_model.set_initial_control_system_status(wn,  
actuator_status=1,  
sensor_status=1,  
controller_status=1)`

Sets the initial status of the control system components (sensors, actuators, and sensors).

**Parameters**

- **wn** (*wntr water network object*) – Water network model object.
- **actuator\_status** (*int*, *optional*) – Initial status of the actuators, defaults to 1
- **sensor\_status** (*int*, *optional*) – Initial status of the sensors, defaults to 1

- **controller\_status** (*int*, *optional*) – Initial status of the controller, defaults to 1

`infrarisk.src.physical.water.water_network_model.set_tank_levels(wn, tank_id, tolerance, print_levels=False)`

Sets the maximum and allowable range of water levels for the tank.

#### Parameters

- **wn** (*wntr water network object*) – water network model object.
- **tank\_id** (*string*) – Name of the tank (id).
- **tolerance** (*float*) – The difference between maximum and allowable tank levels in meters.

`infrarisk.src.physical.water.water_network_model.set_tl_sensor_ef(wn, tank_id, error_factor)`

Sets the error factor for the tank level sensor. A value of 1 means no error.

#### Parameters

- **wn** (*wntr water network object*) – Water network model object.
- **tank\_id** (*str*) – Tank ID.
- **error\_factor** (*float*) – Error factor.

### 7.1.1.2.1.12 Module contents

### 7.1.1.2.2 Submodules

### 7.1.1.2.3 `infrarisk.src.physical.integrated_network` module

`class infrarisk.src.physical.integrated_network.IntegratedNetwork(name, water_folder=None, power_folder=None, transp_folder=None, power_sim_type=None, water_sim_type=None)`

Bases: `object`

An integrated infrastructure network class

`deploy_crews(init_power_crew_locs=None, power_crews_size=None, init_water_crew_locs=None, water_crews_size=None, init_transpo_crew_locs=None, transpo_crews_size=None)`

Deploys the infrastructure crews for performing recovery actions.

#### Parameters

- **init\_power\_crew\_locs** (*list of strings*) – Initial locations (nearest transportation nodes) of the power crews.
- **init\_water\_crew\_locs** (*list of strings*) – Initial locations (nearest transportation nodes) of the water crews.
- **init\_transpo\_crew\_locs** (*list of strings*) – Initial locations (nearest transportation nodes) of the transportation crews.

`generate_betweenness_centrality()`

Generates the betweenness centrality of the integrated graph.

`generate_dependency_table(dependency_file)`

Generates the dependency table from an input file.

**Parameters** **dependency\_file** (*string*) – The location of the dependency file in csv format.

**generate\_integrated\_graph**(*basemap=False*)

Generates the integrated network as a Networkx graph.

**generate\_power\_networkx\_graph**(*plot=False*)

Generates the power network as a networkx object.

**Parameters** *plot* (*bool*, *optional*) – To generate the network plot, defaults to False.

**Returns** The power network as a networkx object.

**Return type** Networkx object

**generate\_transpo\_networkx\_graph**(*plot=False*)

Generates the transportation network as a networkx object.

**Parameters** *plot* (*bool*, *optional*) – To generate the network plot, defaults to False., defaults to False.

**Returns** The transportation network as a networkx object.

**Return type** Networkx object

**generate\_water\_networkx\_graph**(*plot=False*)

Generates the water network as a networkx object.

**Parameters** *plot* (*bool*, *optional*) – To generate the network plot, defaults to False., defaults to False.

**Returns** The water network as a networkx object.

**Return type** Networkx object

**get\_disrupted\_components**()

Returns the list of disrupted components.

**Returns** current list of disrupted components.

**Return type** list of strings

**get\_disrupted\_infra\_dict**()

Returns the disrupted infrastructure components dictionary.

**Returns** The disrupted infrastructure components dictionary.

**Return type** dictionary

**get\_disruptive\_events**()

Returns the disruptive event data

**Returns:** pandas dataframe: The table with details of disrupted components and the respective damage levels.

**get\_idle\_crew**(*crew\_type*, *min\_time=None*)

Returns the idle crew of the given type.

**Parameters** *crew\_type* (*string*) – Type of the crew.

**Returns** The idle crew of the given type.

**Return type** repair\_crews.

**get\_map\_extends**()

Returns the extents of the map in the format ((xmin, ymin), (xmax, ymax)).

**Returns** The extent of the integrated graph (coordinates)

**Return type** list of tuples



**get\_node\_link\_dict()**

**get\_power\_crew\_loc()**

Returns the current power crew location.

**Returns** Power crew location

**Return type** string

**get\_transpo\_crew\_loc()**

Returns the current transportation crew location.

**Returns** Transportation crew location

**Return type** string

**get\_water\_crew\_loc()**

Returns the current water crew location.

**Returns** Water crew location

**Return type** string

**load\_networks**(*water\_folder, power\_folder, transp\_folder, power\_sim\_type='1ph', water\_sim\_type='PDA', sim\_step=60, cyber\_layer=False*)

Loads the water, power and transportation networks.

**Parameters**

- **water\_folder** (*pathlib.Path object, optional*) – The directory that consists of required water network files, defaults to None
- **power\_folder** (*pathlib.Path object, optional*) – The directory that consists of required power network files, defaults to None
- **transp\_folder** (*pathlib.Path object, optional*) – The directory that consists of required traffic network files, defaults to None
- **power\_sim\_type** (*string, optional*) – Power simulation type (“1ph” for single phase networks, “3ph” for three phase networks), defaults to “1ph”
- **water\_sim\_type** (*string*) – Type of water simulation: ‘PDA’ for pressure-dependent driven analysis, ‘DDA’ for demand driven analysis
- **sim\_step** (*int, optional*) – Simulation step in seconds, defaults to 60
- **cyber\_layer** (*bool, optional*) – Whether to include cyber layer in the integrated network, defaults to False

**load\_power\_network**(*power\_folder, power\_sim\_type, cyber\_layer=False*)

Loads the power network.

**Parameters**

- **power\_file** (*string*) – The power systems file in json format
- **power\_sim\_type** (*string, optional*) – Power simulation type (“1ph” for single phase networks, “3ph” for three phase networks), defaults to “1ph”
- **service\_area** (*bool, optional*) – If True, the service area will be loaded, defaults to False

**load\_transpo\_network**(*transp\_folder*)

Loads the transportation network.

**Parameters** **transp\_folder** (*string*) – The directory that consists of required transportation network files

**load\_water\_network**(*water\_folder, water\_sim\_type, initial\_sim\_step=60, cyber\_layer=False*)

Loads the water network.

**Parameters**

- **water\_folder** (*pathlib.Path object*) – The directory that consists of required water network files
- **water\_sim\_type** (*string*) – Type of water simulation: ‘PDA’ for pressure-dependent driven analysis, ‘DDA’ for demand driven analysis
- **initial\_sim\_step** (*int, optional*) – The initial simulation step in seconds, defaults to 60
- **cyber\_layer** (*bool, optional*) – If True, the cyber layer will be loaded, defaults to False

**pipe\_leak\_node\_generator**()

Splits the directly affected pipes to induce leak during simulations.

**reset\_crew\_locs**()

Resets the location of infrastructure crews.

**set\_disrupted\_components**(*disruption\_file*)

Sets the disrupted components in the network.

**Parameters** **scenario\_file** (*string*) – The location of the physical disruption scenario file in the list.

**set\_disrupted\_cyber\_components**(*cyber\_disruption\_file*)

Sets the disrupted components in the network.

**Parameters** **cyber\_disruption\_file** – The location of the disruption scenario file in the list.

**set\_disrupted\_cyber\_dict**()

Sets the disrupted infrastructure components dictionary with infrastructure type as keys.

**set\_disrupted\_infra\_dict**()

Sets the disrupted infrastructure components dictionary with infrastructure type as keys.

**set\_map\_extends**()

Sets the extents of the map in the format ((xmin, ymin), (xmax, ymax)).

**set\_power\_crew\_loc**(*power\_crew\_loc*)

Sets the location of the power crew.

**Parameters** **power\_crew\_loc** (*string*) – The name of the location (transportation node)

**set\_transpo\_crew\_loc**(*transpo\_crew\_loc*)

Sets the location of the transportation crew.

**Parameters** **transpo\_crew\_loc** (*string*) – The name of the location (transportation node)

**set\_water\_crew\_loc**(*water\_crew\_loc*)

Sets the location of the water crew.

**Parameters** **water\_crew\_loc** (*string*) – The name of the location (transportation node)

#### 7.1.1.2.4 `infrarisk.src.physical.interdependencies` module

Classes and functions to manage dependencies in the integrated infrastructure network.

**class** `infrarisk.src.physical.interdependencies.DependencyTable`

Bases: `object`

A class to store information related to dependencies among power, water and transportation networks.

**add\_gen\_reserv\_coupling**(*water\_id*, *power\_id*)

Creates a generator-on-reservoir dependency entry in the dependency table.

**Parameters**

- **water\_id** (*string*) – The name of the reservoir in the water network model.
- **power\_id** (*string*) – The name of the generator in the power systems model.

**add\_pump\_loadmotor\_coupling**(*water\_id*, *power\_id*)

Creates a pump-on-motor dependency entry in the dependency table when motor is modled as a load.

**Parameters**

- **water\_id** (*string*) – The name of the pump in the water network model.
- **power\_id** (*string*) – The name of the motor (modeled as load in three phase pandapower networks) in the power systems model.

**add\_pump\_motor\_coupling**(*water\_id*, *power\_id*)

Creates a pump-on-motor dependency entry in the dependency table.

**Parameters**

- **water\_id** (*string*) – The name of the pump in the water network model.
- **power\_id** (*string*) – The name of the motor in the power systems model.

**add\_transpo\_access**(*integrated\_graph*)

Creates a mapping to nearest road link from every water/power network component.

**Parameters** **integrated\_graph** (*[networkx object]*) – The integrated network as networkx object.

**build\_power\_water\_dependencies**(*dependency\_file*)

Adds the power-water dependency table to the DependencyTable object.

**Parameters** **dependency\_file** (*string*) – The location of the dependency file containing dependency information.

**build\_transportation\_access**(*integrated\_graph*)

Adds the transportation access table to the DependencyTable object.

**Parameters** **integrated\_graph** (*Networkx object*) – The integrated network as Networkx object.

**update\_dependencies**(*network*, *time\_stamp*, *next\_time\_stamp*)

Updates the operational performance of all the dependent components in the integrated network.

**Parameters**

- **network** (*An IntegratedNetwork object*) – The integrated infrastructure network object.
- **time\_stamp** (*integer*) – The start time of the current iteration in seconds.
- **next\_time\_stamp** (*integer*) – The end time of the iteration.

`infrarisk.src.physical.interdependencies.find_connected_nodes(component, integrated_network)`  
 Finds the nodes to which the given component is connected to.

**Parameters**

- **component** (*string*) – Name of the component.
- **integrated\_network** (*networkx object*) – The integrated network in networkx format.

**Returns** List of connected nodes.

**Return type** list

`infrarisk.src.physical.interdependencies.find_connected_power_node(component, pn)`  
 Finds the bus to which the given power systems component is connected to. For elements which are connected to two buses, the start bus is returned.

**Parameters**

- **component** (*string*) – Name of the power systems component.
- **pn** (*pandapower network object*) – The power network the origin node belongs to.

**Returns** Name of the connected bus.

**Return type** string

`infrarisk.src.physical.interdependencies.find_connected_transpo_node(component, tn)`  
 Finds the bus to which the given power systems component is connected to. For elements which are connected to two buses, the start bus is returned.

**Parameters**

- **component** (*string*) – Name of the power systems component.
- **pn** (*pandapower network object*) – The power network the origin node belongs to.

**Returns** Name of the connected bus.

**Return type** string

`infrarisk.src.physical.interdependencies.find_connected_water_node(component, wn)`  
 Finds the water network node to which the water component is connected to.

**Parameters**

- **component** (*string*) – Name of the water network component.
- **wn** (*wntr network object*) – The water distribution network the origin node belongs to.

**Returns** Name of the water network node.

**Return type** string

`infrarisk.src.physical.interdependencies.get_compon_details(compon_name)`  
 Fetches the infrastructure type, component type, component code and component actual name.

**Parameters** **compon\_name** (*string*) – Name of the component.

**Returns** Infrastructure type, component type, component code and component actual name.

**Return type** list of strings

`infrarisk.src.physical.interdependencies.get_compon_repair_time(component)`  
 Returns the repair time of the given component.

**Parameters** **component** (*string*) – Name of the component.

**Returns** Repair time of the component.

**Return type** float

`infrarisk.src.physical.interdependencies.get_nearest_node(integrated_graph, connected_node, target_type)`

Finds the nearest node belonging to a specific family from a given node and the distance between the two.

**Parameters**

- **integrated\_graph** (*networkx object*) – The integrated network in networkx format.
- **connected\_node** (*string/integer*) – Name of the node for which the nearest node has to be identified.
- **target\_type** (*string*) – The type of the target node (power, transpo, water)

**Returns** Nearest node belonging to target type and the distance in meters.

**Return type** list

`infrarisk.src.physical.interdependencies.get_power_repair_time(component)`

Returns the repair time of the given component.

**Parameters** **component** (*string*) – Name of the component.

**Returns** Repair time of the component.

**Return type** float

`infrarisk.src.physical.interdependencies.get_transpo_repair_time(component)`

Returns the repair time of the given component.

**Parameters** **component** (*string*) – Name of the component.

**Returns** Repair time of the component.

**Return type** float

#### 7.1.1.2.5 Module contents

### 7.1.1.3 infrarisk.src.socioeconomic package

#### 7.1.1.3.1 Submodules

#### 7.1.1.3.2 infrarisk.src.socioeconomic.se\_analysis module

`class infrarisk.src.socioeconomic.se_analysis.SocioEconomicTable(name, year, tract, state, county, dir)`

Bases: object

A class for downloading and analyzing socioeconomic data for a given county.

**calculate\_economic\_costs()**

**combine\_infrastructure\_se\_data(integrated\_network, resilience\_metrics)**

**create\_setable()**

**download\_abs\_data(force\_download=False)**

**download\_acs5\_data(force\_download=False)**

**download\_cbp\_data(force\_download=False)**

**download\_ecnbasic\_data(force\_download=False)**

```
download_se_data(force_download=False)
download_zipcode_map(force_download=False)
plot_economic_costs(var, alpha)
plot_industry_output(var, alpha)
plot_interactive(type='annual receipts')
```

### 7.1.1.3.3 Module contents

## 7.1.2 infrarisk.src.network\_recovery module

```
class infrarisk.src.network_recovery.NetworkRecovery(network, sim_step, pipe_close_policy='repair',
                                                    pipe_closure_delay=None,
                                                    line_close_policy='sensor_based_line_isolation',
                                                    line_closure_delay=None)
```

Bases: object

Generate a disaster and recovery object for storing simulation-related information and settings.

```
add_additional_end_events(repair_order, extra_hours=8)
```

Add events post-recovery to the event table.

#### Parameters

- **repair\_order** (*list*) – list of components to be repaired
- **extra\_hours** (*integer, optional*) – Number of hours post-recovery to be added, defaults to 5

```
add_disruption_to_event_table()
```

Schedules the events until the initial disruption events.

```
add_recovery_to_event_table(component, recovery_start, recovery_time)
```

Adds the recovery time to the event table.

#### Parameters

- **component** (*string*) – The component name
- **recovery\_start** (*integer*) – The start time of the recovery in seconds
- **recovery\_time** – The duration of the recovery in seconds

```
calculate_recovery_start_end(links_to_repair, repair_order)
```

Calculates the start and end times for the repair of the links.

#### Parameters

- **links\_to\_repair** (*list*) – The list of components to be repaired
- **repair\_order** (*list*) – The list of components to be repaired in the order in which they are to be repaired

**Returns** The component name, start time and duration of the repair of the links

**Return type** list

```
calculate_recovery_time(component)
```

Calculates the recovery time of the given component.

**Parameters** **component** (*string*) – Component whose recovery time is to be calculated.

**Returns** Recovery time of the component in seconds.

**Return type** float

**calculate\_travel\_time\_and\_path**(*component, crew*)

Calculates the travel time and path for a component and crew.

**Parameters**

- **component** (*string*) – The component for which the travel time and path is to be calculated
- **crew** (*crew object*) – The crew for which the travel time and path is to be calculated

**Returns** The nearest node, travel time to that node for the crew and path for the component and crew

**Return type** list

**check\_route\_accessibility**(*failed\_transpo\_link\_en\_route*)

Checks when the failed transportation links along a route are repaired and the possible start time.

**Parameters** **failed\_transpo\_link\_en\_route** (*list*) – List of transportation links along the route.

**Returns** Accessibility and possible start time of the route.

**Return type** list

**fail\_transpo\_link**(*link\_compon*)

Fails the given transportation link by changing the free-flow travel time to a very large value.

**Args:** link\_compon (string): Name of the transportation link.

**get\_event\_table**()

Returns the event table.

**remove\_previous\_water\_controls**()

Removes all previous pump controls

**reset\_networks**()

Resets the IntegratedNetwork object within NetworkRecovery object.

**restore\_transpo\_link**(*link\_compon*)

Restores the disrupted transportation link by changing the free flow travel time to the original value.

**Args:** link\_compon (string): Name of the transportation link.

**schedule\_recovery**(*repair\_order*)

Generates the unexpanded event table consisting of disruptions and repair actions.

**Parameters** **repair\_order** (*list of strings.*) – The repair order considered in the current simulation.

**set\_initial\_crew\_start**()

Sets the initial start times at which the respective infrastructure crews start from their locations post-disaster.

**switch\_closure\_allowed**(*compons\_to\_repair, switch*)

Check if a switch closure is possible. Depends on whether a switch needs to be open to isolate any component whose repair is not yet performed

**Parameters**

- **network\_recovery** ([NetworkRecovery](#)) – NetworkRecovery object

- **compons\_to\_repair** (*list*) – List of components to be repaired excluding the component under consideration.
- **switch** (*string*) – Switch component

**update\_directly\_affected\_components** (*time\_stamp, next\_sim\_time*)

Updates the operational performance of directly impacted infrastructure components by the external event.

**Parameters**

- **time\_stamp** (*integer*) – Current time stamp in the event table in seconds.
- **next\_sim\_time** (*integer*) – Next time stamp in the event table in seconds.

**update\_traffic\_model** ()

Updates the static traffic assignment model based on current network conditions.

`infrarisk.src.network_recovery.add_pipe_leak(wn, leak_junc, area, discharge_coeff=0.75, start_time=None, end_time=None)`

`infrarisk.src.network_recovery.get_nearest_time_step(time_stamp, time_step)`

`infrarisk.src.network_recovery.link_close_event(wn, pipe_name, time_stamp, state)`

Closes a pipe.

**Parameters**

- **wn** (*wntr network object*) – Water network object.
- **pipe\_name** (*string*) – Name of the pipe.
- **time\_stamp** (*integer*) – Time stamp at which the pipe must be closed in seconds.
- **state** (*string*) – The state of the object.

**Returns** The modified wntr network object after pipe splits.

**Return type** wntr network object

`infrarisk.src.network_recovery.link_open_event(wn, pipe_name, time_stamp, state)`

Opens a pipe. :param wn: Water network object. :type wn: wntr network object :param pipe\_name: Name of the pipe. :type pipe\_name: string :param time\_stamp: Time stamp at which the pipe must be opened in seconds. :type time\_stamp: integer :param state: The state of the object. :type state: string :return: The modified wntr network object after pipe splits. :rtype: wntr network object

`infrarisk.src.network_recovery.pipe_leak_node_generator(network)`

Splits the directly affected pipes to induce leak during simulations.

**Parameters** **network** (*IntegratedNetwork object*) – Integrated infrastructure network object

### 7.1.3 infrarisk.src.optimizer module

**class** `infrarisk.src.optimizer.BruteForceOptimizer` (*prediction\_horizon=None*)

Bases: `infrarisk.src.optimizer.Optimizer`

A Brute Force Optimizer class

**Parameters** **Optimizer** (*Optimizer abstract class.*) – An optimizer class.

**find\_optimal\_recovery** (*simulation*)

Identifies the optimal recovery strategy using the Model Predictive Control principle.

**Parameters** **simulation** (*Simulation object.*) – The infrastructure network simulation object.



**get\_optimization\_log()**

Returns the optimization log.

**Returns** A table consisting of the AUC values from the network simulations.

**Return type** pandas dataframe.

**get\_repair\_permutations(simulation)**

Returns all possible permutations of the repair order.

**Parameters** **simulation** (*Simulation object*) – An integrated infrastructure network simulation object.

**Returns** A nested list of all possible repair permutations for the given list of components.

**Return type** list of lists of strings.

**get\_trackers()**

Returns the time, power consumption ratio and water consumption ratio values.

**Returns:** lists: lists of lists

**class infrarisk.src.optimizer.Optimizer(prediction\_horizon=None)**

Bases: abc.ABC

The Optimizer class defines an interface to a discrete optimizer or can be implemented as such. This optimizer takes a network object and a prediction horizon and should compute the best steps of the length of the prediction\_horizon

**abstract find\_optimal\_recovery(simulation)**

**abstract get\_optimization\_log()**

**abstract get\_trackers()**

## 7.1.4 infrarisk.src.plots module

Functions to generate infrastructure network plots and result plots.

**infrarisk.src.plots.integrate(x, y)**

Calculates the area under a curve

**Parameters**

- **x** (*array of floats*) – An array of the x values of the curve
- **y** – An array of the y values of the curve

:type y:array of floats :return: The area under the curve :rtype: float

**infrarisk.src.plots.plot\_bokeh\_from\_integrated\_graph(G, title, extent=[(1000, 1000), (8000, 6600)], basemap=False)**

Converts the integrated network into a Bokeh interactive plot.

**Parameters**

- **G** (*networkx object*) – Integrated network on which the simulation is to be performed.
- **title** (*string*) – Title of the plot.
- **extent** (*list, optional*) – Extent of the plot as a list of tuple in the format [(xmin, xmax), (ymin, ymax)], defaults to [(1000, 8000), (1000, 6600)]

**infrarisk.src.plots.plot\_bokeh\_lines(p, x, y, infra, link\_layer, link\_category, ids, line\_dash='-', color='black', alpha=1)**

`infrarisk.src.plots.plot_disruptions_and_crews(integrated_network, basemap=False)`

Generate a plot of the number of disruptions and crews for each strategy.

**Parameters** `integrated_network` (`IntegratedNetwork`) – `IntegratedNetwork` object

`infrarisk.src.plots.plot_interdependent_effects(resilience_metrics, metric, title=True)`

`infrarisk.src.plots.plot_network_impact_map(resilience_metrics, integrated_network, strategy, node_prefix, infra='power', time_index=None)`

`infrarisk.src.plots.plot_power_net(net)`

Generates the power systems plot.

**Parameters** `net` (`pandapower network object`) – The power systems network.

`infrarisk.src.plots.plot_region_impact_map(resilience_metrics, sa_dict, strategy, extends)`

`infrarisk.src.plots.plot_repair_curves(disrupt_recovery_object, scatter=False)`

Generates the direct impact and repair level plots for the failed components.

**Parameters**

- **disrupt\_recovery\_object** (`DisasterAndRecovery object`) – The disrupt\_generator.DisruptionAndRecovery object.
- **scatter** (`bool, optional`) – scatter plot, defaults to False

`infrarisk.src.plots.plot_transpo_net(transpo_folder)`

Generates the transportation network plot.

**Parameters** `transpo_folder` (`string`) – Location of the .tnp files.

`infrarisk.src.plots.plot_water_net(wn)`

Generates the water network plot.

**Parameters** `wn` (`wntr network object`) – The water network.

## 7.1.5 infrarisk.src.recovery\_strategies module

`class infrarisk.src.recovery_strategies.CentralityStrategy(integrated_network)`

Bases: object

Based on betweenness centrality of the components multiplied by capacity. Break ties randomly.

**get\_repair\_order()**

Returns the repair sequence based on the principle of betweenness centrality.

**Returns** A list of component names in the order of repair action.

**Return type** list of strings

**set\_repair\_order()**

Identifies the repair sequence based on the betweenness centrality measure. For methodology, please refer to networkx package documentation.

`class infrarisk.src.recovery_strategies.CrewDistanceStrategy(integrated_network)`

Bases: object

Based on the distance between the component and the crew location. Break ties randomly.

**get\_repair\_order()**

Returns the repair sequence based on the distance from the initial crew location.

**Returns** A list of component names in the order of repair action.

**Return type** list of strings

**set\_repair\_order()**

Identifies the repair sequence based on the distance from crew location.

**class** `infrarisk.src.recovery_strategies.HandlingCapacityStrategy(integrated_network)`

Bases: object

Based on the predetermined priority for different components

**get\_repair\_order()**

Returns the repair sequence based on the distance from the initial crew location.

**Returns** A list of component names in the order of repair action.

**Return type** list of strings

**set\_repair\_order()**

Identifies the repair sequence based on the maximum quantity of resource flow handled.

**class** `infrarisk.src.recovery_strategies.JointStrategy`

Bases: object

Optimized strategy. Capture interdependencies somehow if exist. Not an immediate priority

**class** `infrarisk.src.recovery_strategies.ZoneBasedStrategy(integrated_network, zones_shp)`

Bases: object

Based on the zone in which the components are located.

**get\_repair\_order()**

Returns the repair sequence based on the distance from the initial crew location.

**Returns** A list of component names in the order of repair action.

**Return type** list of strings

**set\_repair\_order()**

Identifies the repair sequence based on the zone of the component.

## 7.1.6 infrarisk.src.repair\_crews module

Repair crew classes

**class** `infrarisk.src.repair_crews.PowerRepairCrew(name=None, init_loc=None, crew_size=None)`

Bases: object

Power network repair crew class

**get\_crew\_loc()**

Returns the current location of the crew.

**Returns** Name of the current location of the crew

**Return type** string

**get\_next\_trip\_start()**

Returns the next trip start time for the crew.

**Parameters** **time** (*integer*) – Start time of the next trip start

**reset\_locs()**

Resets the current location of the crew.

**set\_crew\_loc(*loc*)**

Sets the current location of the crew.

**Parameters** *loc* (*string*) – Name of the current location of the crew

**set\_next\_trip\_start(*time*)**

Sets the next trip start time for the crew.

**Parameters** *time* (*integer*) – Start time of the next trip start

**class** `infrarisk.src.repair_crews.TranspoRepairCrew`(*name=None, init\_loc=None, crew\_size=None*)

Bases: `object`

Traffic network repair crew class

**get\_crew\_loc()**

Returns the current location of the crew.

**Returns** Name of the current location of the crew

**Return type** `string`

**get\_next\_trip\_start()**

Returns the next trip start time for the crew.

**Parameters** *time* (*integer*) – Start time of the next trip start

**reset\_locs()**

Resets the current location of the crew.

**set\_crew\_loc(*loc*)**

Sets the current location of the crew.

**Parameters** *loc* (*string*) – Name of the current location of the crew

**set\_next\_trip\_start(*time*)**

Sets the next trip start time for the crew.

**Parameters** *time* (*integer*) – Start time of the next trip start

**class** `infrarisk.src.repair_crews.WaterRepairCrew`(*name=None, init\_loc=None, crew\_size=None*)

Bases: `object`

Water network repair crew class

**get\_crew\_loc()**

Returns the current location of the crew.

**Returns** Name of the current location of the crew

**Return type** `string`

**get\_next\_trip\_start()**

Returns the next trip start time for the crew.

**Parameters** *time* (*integer*) – Start time of the next trip start

**reset\_locs()**

Resets the current location of the crew.

**set\_crew\_loc(*loc*)**

Sets the current location of the crew.

**Parameters** *loc* (*string*) – Name of the current location of the crew

**set\_next\_trip\_start(*time*)**

Sets the next trip start time for the crew.

Parameters **time** (*integer*) – Start time of the next trip start

## 7.1.7 infrarisk.src.resilience\_metrics module

Resilience metric classes to be used for optimizing recovery actions.

**class** `infrarisk.src.resilience_metrics.WeightedResilienceMetric`

Bases: `object`

A class that consists of methods to calculate and store weighted ILOS estimates without

**calculate\_node\_details**(*network\_recovery, wn\_results*)

Calculates the node head, demand and pressure and stores them to respective tables.

Parameters

- **network\_recovery** (*NetworkRecovery object*) – The network recovery object
- **wn\_results** (*wntr object*) – The water network simulation results for the current time interval

**calculate\_power\_load**(*network\_recovery, sim\_time*)

Calculates the power flow in loads and motor pumps.

Parameters

- **network\_recovery** (*NetworkRecovery object*) – The network recovery object.
- **sim\_time** (*integer*) – The simulation time when the data is collected.

**calculate\_power\_resmetric**(*network\_recovery*)

Calculates the power network performance timelines (pcs and ecs).

Parameters **network\_recovery** (*NetworkRecovery object*) – The network recovery object

**calculate\_pump\_energy**(*wn\_object*)

Calculates the energy consumed by each pump in kWhr in the network and stores the values in a dictionary.

Parameters **wn\_object** (*wntr water network object*) – The water network model.

**calculate\_pump\_flow**(*network\_recovery, wn\_results*)

Calculates the flowrates in pumps and stores the values to a table.

Parameters

- **network\_recovery** (*NetworkRecovery object*) – The network recovery object.
- **wn\_results** (*wntr object*) – The water network simulation results for the current time interval

**calculate\_pump\_status**(*network\_recovery, wn\_results*)

Calculates the status of pumps and stores the values to a table.

Parameters

- **network\_recovery** (*NetworkRecovery object*) – The network recovery object.
- **wn\_results** (*wntr object*) – The water network simulation results for the current time interval

**calculate\_transpo\_resmetric**(*m*)

**calculate\_water\_lost**(*network\_recovery, wn\_results*)

Calculates the flows through pipe leaks and stores it to a table.

Parameters

- **network\_recovery** (*NetworkRecovery object*) – The network recovery object
- **wn\_results** (*wntr object*) – The water network simulation results for the current time interval

**calculate\_water\_resmetrics**(*network\_recovery*)

Calculates the water network performance timelines (pcs and ecs).

**Parameters** **network\_recovery** (*NetworkRecovery object*) – The network recovery object

**get\_weighted\_auc\_metrics**()

Returns the weighted auc metrics.

**Returns** list of acs and pcs weighted auc values

**Return type** list

**integrate**(*x, y*)

Calculates the area under a curve

**Parameters**

- **x** (*array of floats*) – An array of the x values of the curve
- **y** – An array of the y values of the curve

:type y:array of floats :return: The area under the curve :rtype: float

**set\_weighted\_auc\_metrics**()

Calculates the water, power, and weighted auc values.

## 7.1.8 infrarisk.src.simulation module

Functions to implement the various steps of the interdependent infrastructure network simulations.

**class** **infrarisk.src.simulation.NetworkSimulation**(*network\_recovery*)

Bases: object

Methods to perform simulation of interdependent effects.

**expand\_event\_table**()

Expands the event table with additional time\_stamps for simulation.

**get\_components\_repaired**()

Returns the list of components that are already repaired.

**Returns** list of components which are already repaired.

**Return type** list of strings

**get\_components\_to\_repair**()

Returns the remaining components to be repaired.

**Returns** The list of components

**Return type** list of strings

**get\_sim\_times**(*network\_recovery*)

Returns the unique simulation times scheduled by the event table.

**Parameters** **network\_recovery** (*NetworkRecovery object*) – A integrated infrastructure network recovery object.

**Returns** Unique simulation time stamps

**Return type** list of integers

**simulate\_interdependent\_effects**(*network\_recovery\_original*)

Simulates the interdependent effect based on the initial disruptions and subsequent repair actions.

**Parameters** **network\_recovery\_original** (*NetworkRecovery object*) – A integrated infrastructure network recovery object.

**Returns** lists of time stamps and resilience values of power and water supply.

**Return type** lists

**update\_repaired\_components**(*component*)

Update the lists of repaired and to be repaired components.

**Parameters** **component** (*string*) – The name of the component that was recently repaired.

**write\_results**(*file\_dir, resilience\_metrics*)

Writes the results to csv files.

**Parameters**

- **file\_dir** (*string*) – The directory in which the simulation contents are to be saved.
- **resilience\_metrics** (*The WeightedResilienceMetric object*) – The object in which simulation related data are stored.

### 7.1.9 Module contents





**SEC DISCLAIMER**

This research is carried out by Singapore ETH Centre through its Future Resilient Systems module funded by the National Research Foundation (NRF) Singapore. It is subject to Agency's review and hence is for internal use only. Not the contents necessarily reflect the views of the Agency. Mention of trade names, products, or services does not convey official NRF approval, endorsement, or recommendation.



## **FUNDING STATEMENT**

The project is funded by the National Research Foundation Singapore through the Inter-CREATE program.



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## BIBLIOGRAPHY

- [Argyroudis2020] S A. Argyroudis, S. A. Mitoulis, L. Hofer, M. A. Zanini, E. Tubaldi, D. M. Frangopol, Resilience assessment framework for critical infrastructure in a multihazard environment: Case study on transport assets, *Science of the Total Environment* 714 (2020) 136854. doi:10.1016/j.scitotenv.2020.136854.
- [Balakrishnan2020] S Balakrishnan, *Methods for Risk and Resilience Evaluation in Interdependent Infrastructure Networks*, Ph.D. thesis, The University of Texas at Austin, Austin, Texas (aug 2020). doi:http://dx.doi.org/10.26153/tsw/13859.
- [Thurner2018] L Thurner, A. Scheidler, F. Schafer, J. H. Menke, J. Dollichon, F. Meier, S. Meinecke, M. Braun, Pandapower - an open-source python tool for convenient modeling, analysis, and optimization of electric power systems, *IEEE Transactions on Power Systems* 33 (6) (2018) 6510–6521. doi:10.1109/TPWRS.2018.2829021.
- [Klise2018] K Klise, D. Hart, M. Bynum, J. Hogge, T. Haxton, R. Murray, J. Burkhardt, *Water network tool for resilience (wntr) user manual.*, Tech. rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States) (2020).
- [Boyles2020] S D. Boyles, N. E. Lownes, A. Unnikrishnan, *Transportation Network Analysis*, 0th Edition, Vol. 1, 2020.





## PYTHON MODULE INDEX

### i

- `infrarisk.src`, 59
- `infrarisk.src.hazards.custom`, 27
- `infrarisk.src.hazards.fragility_based`, 28
- `infrarisk.src.hazards.radial`, 31
- `infrarisk.src.hazards.random`, 33
- `infrarisk.src.hazards.track`, 33
- `infrarisk.src.network_recovery`, 50
- `infrarisk.src.optimizer`, 52
- `infrarisk.src.physical`, 49
  - `infrarisk.src.physical.integrated_network`, 43
  - `infrarisk.src.physical.interdependencies`, 47
  - `infrarisk.src.physical.power`, 36
    - `infrarisk.src.physical.power.power_system_model`, 35
  - `infrarisk.src.physical.transportation`, 40
    - `infrarisk.src.physical.transportation.network`, 36
    - `infrarisk.src.physical.transportation.transpo_compons`, 39
    - `infrarisk.src.physical.transportation.utils`, 40
  - `infrarisk.src.physical.water`, 43
    - `infrarisk.src.physical.water.water_network_model`, 40
- `infrarisk.src.plots`, 53
- `infrarisk.src.recovery_strategies`, 54
- `infrarisk.src.repair_crews`, 55
- `infrarisk.src.resilience_metrics`, 57
- `infrarisk.src.simulation`, 58
- `infrarisk.src.socioeconomic`, 50
  - `infrarisk.src.socioeconomic.se_analysis`, 49