User manual

DTR*i*

JOŽEF STEFAN INSTITUTE

E6 – Department of Communications systems

Contents

What is DTRi

In February 2014 a severe icing storm hit Slovenia, and caused damage in order of 8.5 million € only on the power transmission network. Based on feasibility study performed by Jožef Stefan Institute (JSI), Milan Vidmar Electric Power Research Institute (EIMV), Slovenian Environment Agency (SEA), and ELES, which confirmed that using Joule heating could prevent icing JSI further developed an operative software termed DTRi (Dynamic Thermal Rating - icing). The core of the DTRi is a physical model for the simulation of heat transfer within the transmission power line under realistic weather conditions related to the icing, i.e., ambient temperatures between -5 ^{o}C and 5 ^{o}C with super cooled rain present. The DTRi comprises the Joule heating, convection, solar heating, evaporation, radiation and impinging super cooled precipitation. Basically, the DTRi solves the heat transport equation (second order partial differential equation) with non-linear boundary conditions describing different heat terms due to the weather conditions. The results obtained with DTRi have been compared against available published data as well as measurements provided by EIMV, with promising results.

DTRi can run in two modes, i.e. as **standalone** software and as an **embedded system** within the **SUMO** framework, which is a heterogeneous collection of subsystem from different vendors that was developed to increase safety and security as well as the capacity of the existing transmission network. Its core is the integration platform, SUMO BUS, which is an enterprise integration bus, used for orchestrating the subsystems and facilitating data exchange between them. The communication between SUMO BUS and the subsystems is based on web services. More precisely subsystems communicate with the bus via SOAP/HTTP interfaces. This technology enables different subsystem vendors to quickly and efficiently connect their subsystems using standardized and open means of communication and exchanging data. For example, SUMO allows different Dynamic Thermal Rating (DTR) vendors to be incorporated into the system, each serving a different part of the grid. Currently there are 17 services implemented. They are providing approximately 115 methods to the clients (subsystems). System's internal state is held in a relational SQL database.

Physical model

The backbone of the DTRi is a physical model that in general solves energy transport through the conductor as

$$\lambda \frac{1}{r}\frac{\partial}{\partial r} r \frac{\partial}{\partial r} T + q_i\left(T\right) = \rho c_p \frac{\partial T}{\partial t} \quad \left.\frac{\partial T}{\partial r}\right|_{r=0} = 0\,, \quad -\lambda \left.\frac{\partial T}{\partial r}\right|_{r=r_2} = \sum_j q_j\,, T\left(r,t=0\right) = T_a\,, T\left(r,t=0\right) = T_a\,,$$

where λ, r, T, q_i, ρ and c_p stand for thermal conductivity, radii, temperature, heat source term, density and specific heat capacity, respectively, q_j describe different heat terms due to the weather conditions and T_a stands for ambient temperature. The heat terms are modelled as follows:

Joule heating	$q_j = \frac{4}{\pi D^2} I^2 R(T,r) \left[\frac{\mathrm{W}}{\mathrm{m}^3}\right]$	with temperature dependent conductivity and D standing for line diameter.
solar heating	$q_s = \frac{\alpha_s I_T}{\pi} \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right],$	where I_T stands for measured solar intensity and α_s for absorptivity
radiation	$q_r = -\sigma_B \varepsilon_s \left(T_s^4 - T_a^4\right) \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$	where σ_B, ε_s and T_s stand for Stefan-Boltzmann constant, emissivity, and power line skin temperature, respectively.
convection	$q_c = -h(T_s - T_a) \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$	with h standing for Nusselt number
impinging	$q_{im} = \frac{0.71}{\pi} c_w \frac{d\rho_I}{dt} \left(T_s - T_a\right) \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$	with c_w standing for specific heat of water
rain mass flux	$\frac{d\rho_I}{dt} = \sqrt{\left(\frac{10^{-3}}{3600} P \rho_w\right)^2 + \left(u 6.710^{-5} P^{0.84}\right)^2} \left[\frac{\mathrm{kg}}{\mathrm{m}^2\mathrm{s}}\right]$	with u standing for wind velocity, P rain rate and ρ_W water density
evaporation	$q_e = -\frac{1}{2} h \frac{k L_e}{c_p} \left(\frac{(1-r) e_s}{p}\right) \left[\frac{\mathrm{W}}{\mathrm{m}^2}\right]$	with L_e standing for evaporation latent heat, c_p for specific heat of air, p for air pressure, k = 0.62 is the ratio of molecular weights of water vapor and dry air
saturation pressure	$e_s(T) = e_{s0} e^{\left(\frac{L_e}{R_v}\left(\frac{1}{T_0} - \frac{1}{T}\right)\right)}$	with e_{s0} = 6.1 hPa and Rv=461 J/KgK, where the maximal evaporation rate is limited by rain mass flux.

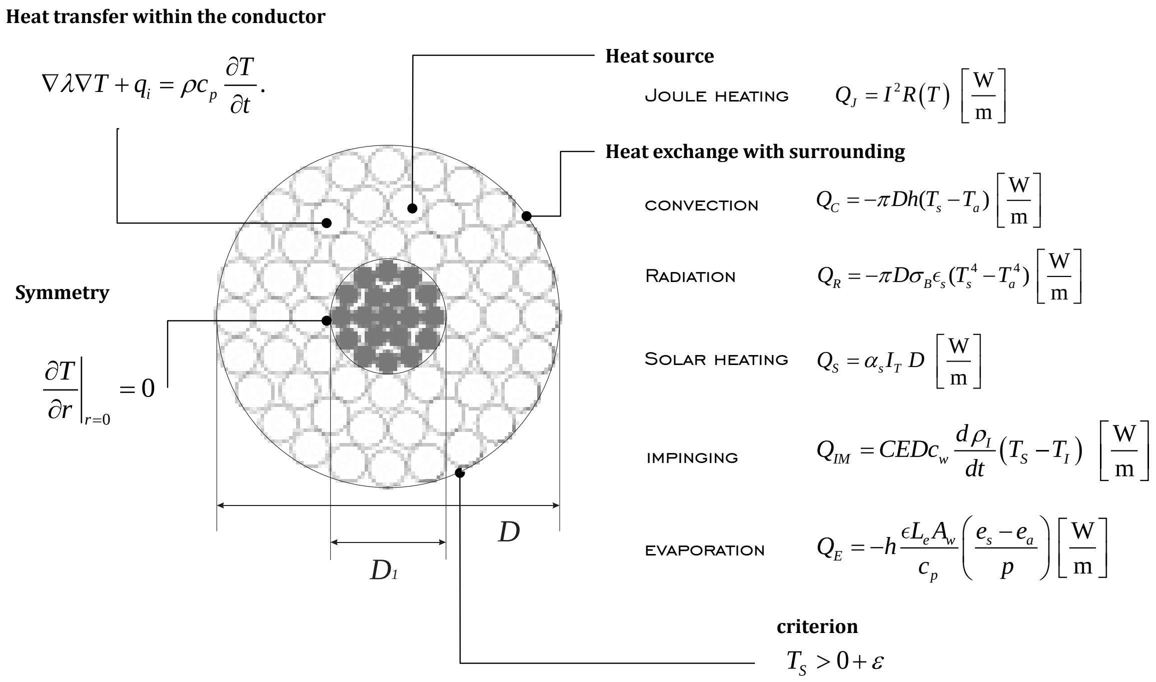

Heat transfer within the conductor
$\nabla \lambda \nabla T + q_i = \rho c_p \frac{\partial T}{\partial t}.$
Heat source
Joule heating
$Q_J = I^2 R(T) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
Heat exchange with surrounding
Convection
$Q_C = -\pi D h (T_s - T_a) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
Radiation
$Q_R = -\pi D \sigma_B \epsilon_s (T_s^4 - T_a^4) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
Symmetry
$\left.\frac{\partial T}{\partial r}\right|_{r=0} = 0$
Solar heating
$Q_S = \alpha_s I_T D \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
Impinging
$Q_{IM} = CEDc_w \frac{d\rho_I}{dt} (T_S - T_I) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
D
D_1
Evaporation
$Q_E = -h \frac{\epsilon L_e A_w}{c_p} \left(\frac{e_s - e_a}{p}\right) \left[\frac{\mathrm{W}}{\mathrm{m}}\right]$
criterion
$T_S > 0 + \varepsilon$

Figure 1: Scheme of DTRi physical model

Model parameters

Model parameters are supplied via web interface or the configuration file. Options are divided into several categories. Most of the parameters are required, but the ones marked with * are **optional**.

Line properties

This category describes options regarding the overhead line, made out of two materials, the inner and outer material.

Parameter name	Parameter description	Unit
line_altitude	Altitude (height above sea level) of the part of the line for which the simulation is run.	m
line_angle	Line angle with respect to the ground measured from a fixed line from 0° to 360°.	deg
thermal_conductivity	Thermal conductivity of the line (outer material).	W / m K
num_outer_strands	Number of outer strands in the line.	1
single_strand_radius	Radius of a single strand.	m
radius_correction	Correction of line radius due to strand packing	1
wetted_factor	Ratio of wetted area of conductor (used for evaporation).	1
impinging_factor	Ratio of impinging water that reaches the skin temp. 0.7 [Zsolt].	1
recovery_factor	Recovery factor (= 0.79) (used for friction heating).	1
skin_effect	Skin effect factor (used to determine roughness).	1
emissivity	Emissivity of the line (outer material).	1
absorptivity	Absorptivity of the line (outer material).	1
nusselt_mode	How to calculate the convection coefficient -- depends on line type. It can be either AL240FE40 or AL490FE65.	/

For inner and outer material the following properties have to be specified:

Parameter name	Parameter description	Unit
density	Density of the material.	kg / m3
specific_heat	Isobaric specific heat capacity.	J / kg K
specific_heat_alpha	Specific heat linear temperature coefficient.	1 / K
area	Cross section area of the material.	m2
resistivity_alpha	Resistivity temperature coefficient.	1 / K
electric_conductivity	Specific electric conductivity.	1 / m Ω

Weather data

Weather measurements at one or more time points must be provided. Each weather point contains a time of measurement and the values of measured quantities:

Parameter name	Parameter description	Unit
time	Time when data was measured (time relative to a chosen starting point).	s
ambient_temperature	Ambient air temperature.	°C
droplet_temperature*	Droplet temperature. If not specified, defaults to ambient temperature.	°C
droplet_MDV	Median of the droplet diameter.	m
wind_velocity	Wind velocity.	m / s
wind_angle	Wind angle relative to the ground using the same coordinate system as above.	deg
pressure	Air pressure.	Pa
rain_rate	Rain rate.	mm / h
humidity	Humidity of the surrounding air.	%
solar_irradiance	Solar irradiance.	W / m^2

Constants of nature

Certain constants of nature are also required by the model. Although these are constants, for the sake of generality, the DTRi considers them as an input.

	Parameter name	Parameter description	Unit
	stefan_constant	Stephan constant	W / m^2 K^4
	molar_mass_ratio	Ratio of molar masses of water vapor and dry air	
	gas_constant	Gas constant	J / K mol
	kelvin_celsius_diff	Difference between Celsius and kelvin unit (positive number)	°C
water	density	Water density	kg / m^3
	latent_heat_fusion	Latent heat of fusion (freezing)	J / kg
	latent_heat_evaporation	Latent heat of evaporation	J / kg
	latent_heat_sublimation	Latent heat of sublimation	J / kg
	specific_heat_ice	Specific heat of ice (cp)	J / kg K
	specific_heat_water	Specific heat of water	J / kg K
air	specific_heat	Specific heat of air.	J / kg K
	density*	Density of air – if not present, computed during simulation.	kg / m^3
	viscosity*	Kinematic viscosity of air – if not present, computed during simulation.	m^2 / s
	thermal_conductivity*	Thermal conductivity of air – if not present, computed during simulation	W / m K

Numerical setup

These options control the numerical aspect of calculations.

Parameter name	Parameter description	Unit
num_nodes	Number of nodes in discretization.	1
time_step	Time step of the implicit Euler.	s
steady_state_crit	Finish when temperature changes less than this (negative value disables it). When in bisection mode this parameter is set to a positive number equal to 0.1 × bisection precision.	°C
start_time*	Start simulation at this time, not the first one in the data. When running in bisection mode, find current only for weather data points later than this time.	s
end_time*	End simulation at this time, not the last one in the data. When running in bisection mode, find current only for weather data points earlier than this time.	s
radial_distribution	Do we use radial distribution of lambda or not. It can be either true or false.	/
use_effective_radius	Do we use effective radius due strand packing. It can be either true or false.	/
output_rate	Write data every output_rate time steps, if 0, don't write at all.	1
debug_level	How verbose do you want the output to be from 0 – 9.	1

Run setup

The first choice of the run setup is the run mode. The model can either simulate the temperature, given the current, or find the current which produces the wanted temperature. These modes are called SIMULATION and BISECTION, respectfully.

Parameter name	Parameter description	Unit
run_mode	Specifies in which mode the model should be run. It can be either "SIMULATION" or "BISECTION".	/

If the simulation mode was chosen, the simulation parameters must be specified.

Parameter name	Parameter description	Unit
electrical_current	A value of the current flowing through the line for every time, at which the weather was specified.	A

Likewise, if the bisection mode was chosen, the bisection parameters must be specified.

Parameter name	Parameter description	Unit
target_temperature	Target skin temperature.	°C
min_current	Bisection min current.	A
max_current	Bisection max current.	A
precision	How precisely to determine the temperature.	A
max_iterations	Maximal number of iterations of bisection, this implies the precision of the current, as it is equal to $0.5^{\text{max\_iterations}}$.	1

DTRi IMPLEMENTATION

The whole DTRi system comprises the back end computing the physical model, middle end handling outputs from backend and **communicating with SUMO**, and finally web front end providing users appealing control over the **standalone executions**. The implementation is schematically presented in Figure 2.

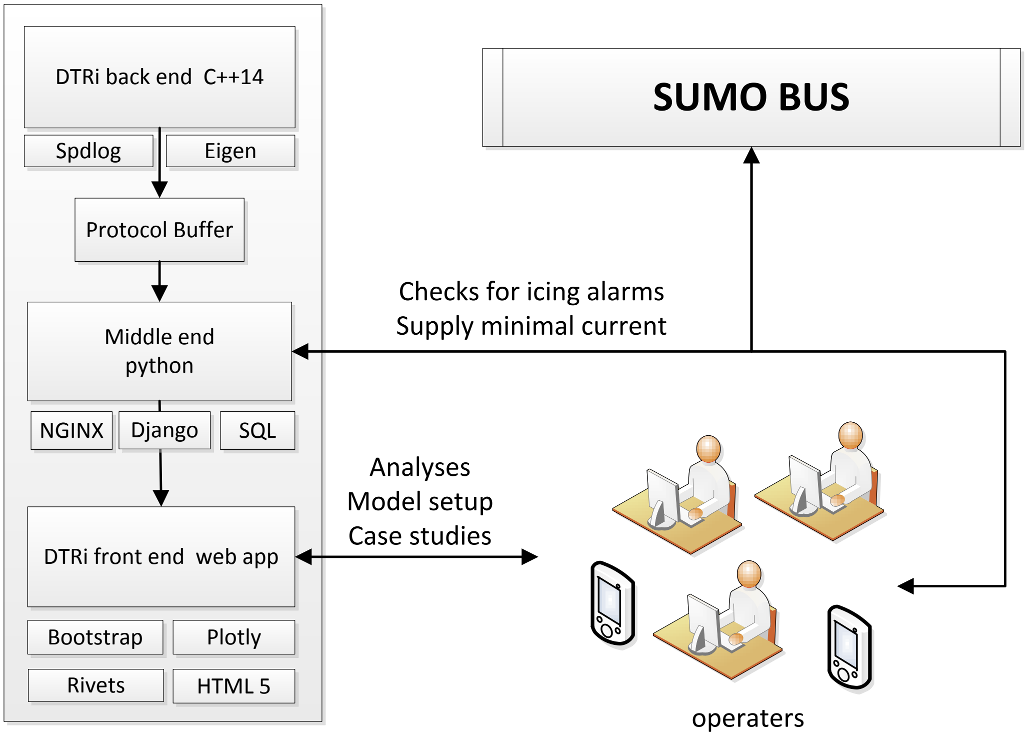


Figure 2: Scheme of DTRi

Backend

The backend is written in **C++14**. All involved matrix operations are performed with **Eigen** numerical library[1]. Communication with the server is implemented using **Protocol Buffers** and **spdlog** library is used for logging.

Console execution

The backend binary can be run directly from the console. First, make sure the binary is compiled. Got to `cpp/` folder and run

```
mkdir –p build/
cd build/
```

[1] http://eigen.tuxfamily.org/

make dtri

To execute go to **cpp/** folder and run **./bin/dtri.**

The `dtri` executable takes one positional argument that specifies the mode. It can be either "**run**" or "**test**". The "run" mode is used for execution by the server. It accepts the serialized protocol buffer containing input data to the standard input and prints serialized protocol buffer with results to the standard output. The "test" mode is useful for manual execution. It takes another positional argument specifying the configuration file and prints results in human readable form to `stdout`, with additional logs to `stderr`.

Example run:

```
user@computer ~/dtri/cpp $ ./bin/dtri test data/test_case_1.conf
[2016-04-25 15:16:23.598] [console] [notice] Started.
[2016-04-25 15:16:23.598] [console] [notice] Parsed successfully!
[2016-04-25 15:16:23.598] [console] [notice] Setting defaults...
[2016-04-25 15:16:23.598] [console] [notice] Defaults set!
[2016-04-25 15:16:23.598] [console] [notice] Checking validity...
[2016-04-25 15:16:23.598] [console] [notice] Message valid!
[2016-04-25 15:16:23.598] [console] [notice] Initializing...
[2016-04-25 15:16:23.598] [console] [notice] Bisection mode chosen.
[2016-04-25 15:16:23.604] [console] [notice] temperature at low current of 0 A is -0.352063 deg C
[2016-04-25 15:16:23.604] [console] [notice] temperature at high current of 4000 A is 743.673 deg C
[2016-04-25 15:16:23.664] [console] [notice] Found current: 149.109 A at time 0 s
[2016-04-25 15:16:23.669] [console] [notice] temperature at low current of 0 A is -2.2156 deg C
[2016-04-25 15:16:23.669] [console] [notice] temperature at high current of 4000 A is 742.678 deg C
[2016-04-25 15:16:23.727] [console] [notice] Found current: 371.104 A at time 100 s
[2016-04-25 15:16:23.727] [console] [notice] Sending response...
time: 0
time: 100
electrical_current: 149.109
electrical_current: 371.104
core_temperature: 0.11580593639820913
core_temperature: 0.71872955547329687
skin_temperature: -2.3780966901822732e-06
skin_temperature: 3.12489564365419e-06
heat_flux_radiation: -0.016517339392556674
heat_flux_radiation: -0.32692405169670463
heat_flux_convection: -0.45616735335532721
heat_flux_convection: -9.1235782893736221
heat_flux_solar: 0.47422139828232979
heat_flux_solar: 0.47422139828232979
heat_flux_friction: 0.016136452806899747
heat_flux_friction: 0.016136452806899747
heat_flux_impinging: -0.025718398580340099
heat_flux_impinging: -0.51250772648009268
heat_flux_evaporation: -2.2906344862048704
heat_flux_evaporation: -4.7868499190929557
heat_flux_joule_heating: 2.4087248839692652
heat_flux_joule_heating: 14.944556836615945
num_iterations: 22
[2016-04-25 15:16:23.727] [console] [notice] Response sent.
[2016-04-25 15:16:23.727] [console] [notice] All done, exiting.
user@computer ~/dtri/cpp $
```

Code structure

Code is separated in four source files:

- `dtri.cpp`
- `dtri.hpp`
- `dtri_main.cpp`
- `dtri_test.cpp`

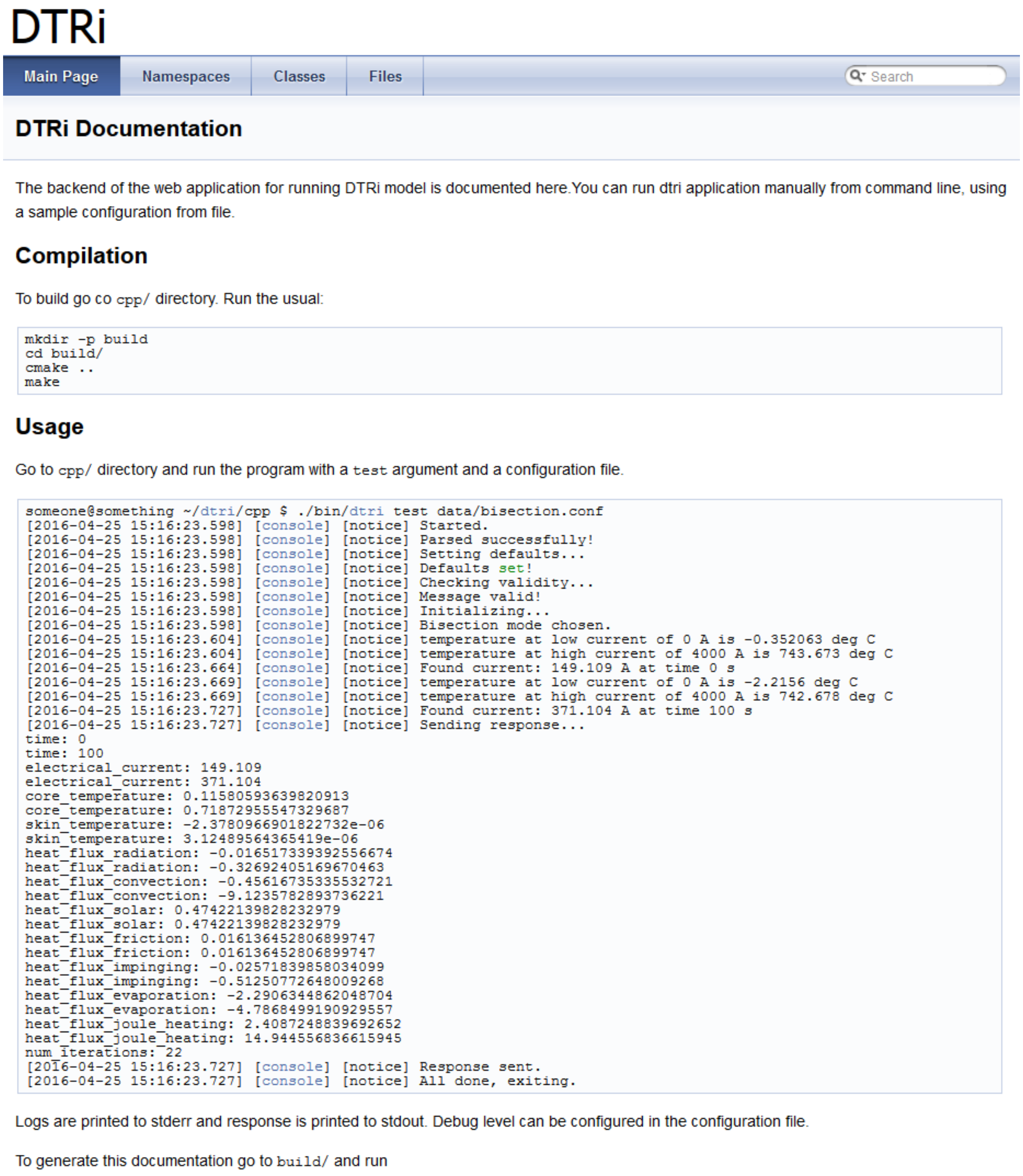
The `dtri_test.cpp` file contains tests as described in section Unit tests. The `dtri_main.cpp` contains the `main` function, which takes care of argument handling and communication. It reads and writes configurations from files or serialized and deserializes data from standard input. It also detects **run_mode** and appropriately runs bisection of simulation.

The declarations and documentation of the classes and functions used to implement simulation and bisection modes are located in `dtri.hpp` file in the `dtri` namespace. The corresponding implementations are in the `dtri.cpp` file.

Main class for simulation is called `DTRi`. It takes all parameters in the constructor and runs the simulation when the `simulate` method is called. The results are stored in the `response` attribute, which can then be directly printed as a result. The intermediate results are also accessible and are stored in the **SimulationData** class.

Main class for bisection is called **BisectionRunner**. Similarly to `DTRi`, it takes all parameters as input and performs bisection to find the current at each time point given. The intermediate results are stored in the **BisectionData** class. The bisection can be run using **run_bisection** method. The method also takes the log level of the simulation and the log level of bisection which specify the verbosity of logging during the run. The results are stored in the `response` attribute, which can be returned directly serialized to the server.

For more precise technical documentation see the doxygen generated docs. It can be built by going to `build/` folder and running `make docs`. This will populate the `docs/` folder with html and latex documentations.



DTRi Documentation

The backend of the web application for running DTRi model is documented here.You can run dtri application manually from command line, using a sample configuration from file.

Compilation

To build go co `cpp/` directory. Run the usual:

```
mkdir -p build
cd build/
cmake ..
make
```

Usage

Go to `cpp/` directory and run the program with a `test` argument and a configuration file.

```
someone@something ~/dtri/cpp $ ./bin/dtri test data/bisection.conf
[2016-04-25 15:16:23.598] [console] [notice] Started.
[2016-04-25 15:16:23.598] [console] [notice] Parsed successfully!
[2016-04-25 15:16:23.598] [console] [notice] Setting defaults...
[2016-04-25 15:16:23.598] [console] [notice] Defaults set!
[2016-04-25 15:16:23.598] [console] [notice] Checking validity...
[2016-04-25 15:16:23.598] [console] [notice] Message valid!
[2016-04-25 15:16:23.598] [console] [notice] Initializing...
[2016-04-25 15:16:23.598] [console] [notice] Bisection mode chosen.
[2016-04-25 15:16:23.604] [console] [notice] temperature at low current of 0 A is -0.352063 deg C
[2016-04-25 15:16:23.604] [console] [notice] temperature at high current of 4000 A is 743.673 deg C
[2016-04-25 15:16:23.664] [console] [notice] Found current: 149.109 A at time 0 s
[2016-04-25 15:16:23.669] [console] [notice] temperature at low current of 0 A is -2.2156 deg C
[2016-04-25 15:16:23.669] [console] [notice] temperature at high current of 4000 A is 742.678 deg C
[2016-04-25 15:16:23.727] [console] [notice] Found current: 371.104 A at time 100 s
[2016-04-25 15:16:23.727] [console] [notice] Sending response...
time: 0
time: 100
electrical_current: 149.109
electrical_current: 371.104
core_temperature: 0.1158059363982091З
core_temperature: 0.71872955547329687
skin_temperature: -2.3780966901822732e-06
skin_temperature: 3.12489564365419e-06
heat_flux_radiation: -0.016517339392556674
heat_flux_radiation: -0.32692405169670463
heat_flux_convection: -0.45616735335532721
heat_flux_convection: -9.1235782893736221
heat_flux_solar: 0.47422139828232979
heat_flux_solar: 0.47422139828232979
heat_flux_friction: 0.016136452806899747
heat_flux_friction: 0.016136452806899747
heat_flux_impinging: -0.025718398580З4099
heat_flux_impinging: -0.51250772648009268
heat_flux_evaporation: -2.2906344862048704
heat_flux_evaporation: -4.7868499190929557
heat_flux_joule_heating: 2.4087248839692652
heat_flux_joule_heating: 14.944556836615945
num_iterations: 22
[2016-04-25 15:16:23.727] [console] [notice] Response sent.
[2016-04-25 15:16:23.727] [console] [notice] All done, exiting.
```

Logs are printed to stderr and response is printed to stdout. Debug level can be configured in the configuration file.

To generate this documentation go to `build/` and run

Figure 3: Example source documentation screenshot

Logging

Spdlog header only library[2] is used for logging. The library is optimized for speed and offers several convenient features such as formatting, severity levels and file logging; refer to the supplied link for more details. In DTRi code logging is structured as follows:

Level	#	Purpose
emergency	1	Currently unused.
alert	2	Currently unused.
critical	3	For extreme situations when the program cannot continue, i.e. "cannot parse input", "time step negative".
error	4	Used for semantical error regarding the input, i.e. computation ended in a NaN.
warning	5	Used for strange computations which do not terminate the computation, i.e. air pressure is extremely high, temperature is extremely low of over the model limit, external and Joule heat flux are not approximately equal in steady state.
notice	6	This level logs the general execution stages of the model with messages such as "Simulation started", "Sending response…"
info	7	This level logs more detailed execution stages as well as some basic intermediate results. Useful to inspect the behavior of simulation or bisection iterations.
debug	8	Outputs most intermediate results, useful when debugging wrong computation results. One can easily trace the origin of the error by backtracking over the path of wrong results.
trace	9	Extremely detailed output: for every calculated quantity, the quantities it was calculated from are printed above. It is extremely useful for tracking numerical errors but the output gets very large when a lot of iterations are performed.
off	0	There is no output.

A usual log line features log location, date and time, level and the message. An example is shown below:

```
[2016-04-25 15:16:23.604] [console] [notice] temperature at low current of 0 A is -0.352063 deg C
```

Unit tests

Unit tests are written in Google test framework[3], a unit testing library for the C++ programming language, based on the xUnit architecture. The library is released under the BSD 3-clause license and it can be compiled for a variety of POSIX and Windows platforms, allowing unit-testing of 'C' sources as well as C++ with minimal source modification. The tests themselves could be run one at a time, or even be called to run all at once. This simplifies the debugging process and caters to the need of many programmers and coders alike. It also safeguard against regressions in the new version of the model, as it can be easily tested that the new functionality did not cause any errors in the already existing features.

Following tests are performed every time before a new version of DTRi is deployed:

- test for tridiagonal solver
- test for correct matrix of implicit scheme
- test of a single iteration of simulation
- test of two iterations of a simulation

[2] https://github.com/gabime/spdlog/

[3] https://github.com/google/googletest

- complete simulation with rough numerical parameters
- complete simulation with fine numerical parameters
- complete simulation without steady state criterion
- complete simulation without effective radius
- complete simulation without radial distribution of heat
- test for one iteration for bisection
- test for bisection with rough numerical approximation and single weather point
- test for bisection with fine numerical approximation and multiple weather points

The unit tests are implemented in `dtri_test.cpp` file and can be compiled by running `make test_dtri` in the `build` folder. The test can be run by running `./bin/test_dtri`. The desired output is below:

```
user@computer ~/dtri/cpp $ ./bin/test_dtri
Running main() from gtest_main.cc
[==========] Running 12 tests from 3 test cases.
[----------] Global test environment set-up.
[----------] 1 test from TridiagonalSolver
[ RUN      ] TridiagonalSolver.Solve
[       OK ] TridiagonalSolver.Solve (0 ms)
[----------] 1 test from TridiagonalSolver (0 ms total)

[----------] 8 tests from Simulation
[ RUN      ] Simulation.BuildsCorrectMatrix
[       OK ] Simulation.BuildsCorrectMatrix (5 ms)
[ RUN      ] Simulation.OneIteration
[2016-05-23 12:48:10.393] [console] [warning] Qj more than 0.1 different than Qi. Qj = 19.85 W/m, Qi = -14.6051 W/m
[       OK ] Simulation.OneIteration (0 ms)
[ RUN      ] Simulation.TwoIterations
[       OK ] Simulation.TwoIterations (0 ms)
[ RUN      ] Simulation.RoughNumeric
[       OK ] Simulation.RoughNumeric (1 ms)
[ RUN      ] Simulation.FineNumeric
[       OK ] Simulation.FineNumeric (14 ms)
[ RUN      ] Simulation.NoSteadyState
[       OK ] Simulation.NoSteadyState (32 ms)
[ RUN      ] Simulation.NoEffectiveRaduis
[       OK ] Simulation.NoEffectiveRaduis (8 ms)
[ RUN      ] Simulation.NoRadialDistribution
[       OK ] Simulation.NoRadialDistribution (0 ms)
[----------] 8 tests from Simulation (60 ms total)

[----------] 3 tests from Bisection
[ RUN      ] Bisection.OneIteration
[2016-05-23 12:48:10.462] [console] [warning] Bisection could not achieve selected precision after 1 iterations. Last precision:
12.8892 deg C
[       OK ] Bisection.OneIteration (14 ms)
[ RUN      ] Bisection.Small
[       OK ] Bisection.Small (7 ms)
[ RUN      ] Bisection.Long
[       OK ] Bisection.Long (361 ms)
[----------] 3 tests from Bisection (382 ms total)

[----------] Global test environment tear-down
[==========] 12 tests from 3 test cases ran. (442 ms total)
[  PASSED  ] 12 tests.
user@computer ~/dtri/cpp $
```

Errors and error codes

Error code name	code	Error description
`INVALID_MODE`	1	The program did not recognize the test mode specified. It must be either “run” or “test”.

MISSING_SIMULATION_PARAMETERS	2	The specified run mode was “simulation” but the simulation parameters were either missing or incomplete.
WEATHER_AND_CURRENT_DATA_MISMATCH	3	The weather and current data were not of the same length. There must always be an electrical current entry for every weather point specified.
MISSING_BISECTION_PARAMETERS	4	The specified run mode was “bisection” but the bisection parameters were either missing or incomplete.
NUMBER_OF_NODES_TOO_SMALL	5	The number of nodes in the discretization of the line radius was too small to perform a reliable simulation. This does not imply that all simulations above this limit are reliable!
NEGATIVE_OR_ZERO_TIME_STEP	6	The time step was negative or zero, so it was impossible to run the simulation.
MISSING_WEATHER_DATA	7	Weather data is missing or is incomplete.
NON_INCREASING_TIME_STAMPS	8	Time stamps in weather data were not strictly increasing. Check if the data is sorted correctly, and remove any possible duplicates.
START_TIME_TOO_SMALL	9	The specified start time was before the first weather entry, so there is no way to deduce the weather before first entry, and the simulation could not be run.
END_TIME_BEFORE_START_TIME	10	The end time of the simulation is before the start time, so there is no simulation to be done. Note that is they are the same, a simulation is run at only that time.
BISECTION_PRECISION_TOO_SMALL	11	Precision to which the temperature should be determined is too small: it is either negative or below 1e-9.
BISECTION_LOW_NOT_SMALLER_THAN_BISECTION_HIGH	12	The highest possible current allowed in bisection is lower than the lowest allowed current. It is impossible to run the bisection.
TARGET_TEMPERATURE_NOT_ACHIEVABLE	13	Even if the highest current were used, the desired temperature could not be achieved. Try rising the high limit if it makes sense; otherwise that temperature is simply not achievable. The reversed case, when even if no current is flowing the temperature of the line is higher than specified only results in a warning.
TARGET_TEMPERATURE_OUT_OF_MODELLING_RANGE	14	The requested target temperature is out of modelling range – it is probably too high and does not fit in the icing conditions.
MAX_NUMBER_OF_BISECTION_ITERATIONS_TOO_SMALL	15	The maximal number of iterations that bisection is allowed to perform must be at least 1.
ERROR_DURING_EXECUTION	16	Used for other errors during execution, that did not cause fatal termination, i.e. values of quantities were nan. Usually additional explanations from other errors accompany and help explain this error.
NO_WEATHER_DATA_IN_TIME_RANGE	17	This error occurs only in bisection mode. The time range you specified does not include any weather data point and the calculation could not be performed.
LOADING_JSON_FROM_POST_FAILED	29	The model parameters data could not be transferred from the frontend to the server. Either the connection was reset of there is something wrong in the application. You can try running again, but if the problem persists also when using different data, contact the developers.
FAILED_POPULATING_MODEL_PARAMETERS	30	There was invalid data when creating basic container for model parameters.
FAILED_POPULATING_CONSTANTS_OF_NATURE	31	There was invalid data in the constants of nature section and the

		message could not be created.
FAILED_POPULATING_WATER_CONSTANTS	32	There was invalid data in the water constants section and the message could not be created.
FAILED_POPULATING_AIR_CONSTANTS	33	There was invalid data in the air constants section and the message could not be created.
FAILED_POPULATING_LINE_PROPERTIES	34	There was invalid data in the line properties section and the message could not be created.
FAILED_POPULATING_INNER_MATERIAL_PROPERTIES	35	There was invalid data in the inner material properties section and the message could not be created.
FAILED_POPULATING_OUTER_MATERIAL_PROPERTIES	36	There was invalid data in the outer material properties section and the message could not be created.
FAILED_POPULATING_NUMERICAL_SETUP	37	There was invalid data in the numerical setup section and the message could not be created.
FAILED_POPULATING_BISECTION_PARAMETERS	38	There was invalid data in the bisection parameters section and the message could not be created.
FAILED_POPULATING_SIMULATION_PARAMETERS	39	There was invalid data in the simulation parameters section and the message could not be created.
FAILED_POPULATING_WEATHER_DATA	40	There was invalid data in the water data section and the message could not be created.
WEATHER_DATA_EMPTY	41	There was no weather data specified.
FAILED_POPULATING_WEATHER_POINT	42	There was invalid data in one of the weather points and the message could not be created
EXECUTABLE_NOT_FOUND	43	The model executable was not found. This error should not occur and it means something is wrong with the setup of the application. Possible causes for this error are that the lookup path for the executable is wrong or it has not been compiled yet.
UNPARSABLE_DTRI_RESPONSE	44	The response from the executable was unparsable. This error should not occur and it indicates that something is wrong with the executable itself. You can try recompiling, inspecting the logs to detect unusual output or run manually.
EXECUTABLE_FAILED_TO_FINISH_ON_TIME	45	The run of the model took more that the specified time limit (currently one minute) and was cancelled. Try running with a different setup (less nodes, greater time step).
COULD_NOT_SERIALIZE	46	The created message could not be serialized. This error should not occur normally and it indicated there is probably something wrong with this part of the server.
TOO_MANY_SUBPROCESSES_AT_THE_SAME_TIME	47	The server detected too many executables running at the same time and disallowed the run. Try waiting for 30 seconds to let other ruins finish and try again.
UNKNOWN_ERROR	96	The error was not of the above errors and was not predicted to happen. We ask you to submit the error to the developers by email, as well as the steps to reproduce it, the configuration used and the rest of the run details (logs, user details, etc.).

Warnings are also issued then temperature exceeds modelling range in simulation, input data is valid but strange (high pressure, extreme humidity, low temperatures). Another case for warning is when the specified target temperature could not be achieved, because even if not current is flowing the temperature of the line would be higher.

Middle end

The middle communicates with the DTRi simulation executable, SUMO and front end. The core of middle end is written in Python. Django web framework is used to run the application. It is fast, secure and scalable. The backend currently uses SQLite database. The application is secured with a login using salted and hashed password storage and cross site forgery tokens. The middle end limits the number of `dtri` instances running to a fixed number. For seamless communication with frontend AJAX and RESTful APIs are used.

Protocol buffers

The data transfer between back-end, middle-end and frontend is implemented with Protocol Buffers[4] that are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, and then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages, in present case C++, Python and JS.

For the message definition refer to appendix section.

Frontend

Technical description

Front end is prepared in HTML5/CSS3 with Bootstrap, Plotly and Rivets additional libraries. Template design is taken from w3layouts and uses Bootstrap for responsive and visually appealing design. Rivets two-way binging library is used to swiftly update different parts of the application on interaction for the user or upon receiving new data from the backend. For plotting, the Plotly library is used, which provides a rich set of plot types and features as well as speed and stability. The font face used is Lato from Google Fonts and icons are taken from Font Awesome and Linearicons.

Usage overview

The dashboard's main menu is located on the left side of the screen and can be toggled using the round red hamburger button.

Dashboard: this is the default view of the application. It is divided into two parts vertically. Above, there is the setup for the run, where user inputs the data either by choosing a preset instance, uploading it from file or loading it from SUMO by clicking the link icon. For the meaning of the setup options refer to Model parameters section. After the wanted setup was chosen, click the run button to execute it. This send the selected data through the middle end to the model in the backend and returns the results on a successful run of errors on failure. The status code below the button is 0 for successful runs and nonzero otherwise. For nonzero exit codes consult the Errors and error codes section. The logs from the model are shown in the resizable log box. To the right, potential error or warning boxes are shown as well as a box with small statistical report on running time. On a successful run, results are displayed below on two plots and in the table. The plots support change of quantities on both axis and the table can be downloaded as a csv file for further analysis. This process is more precisely illustrated in the Standalone execution section.

[4] https://developers.google.com/protocol-buffers/

Sumo connect: this tab controls the application's interaction with SUMO system. A small status report is shown and some settings of interactions are offered. A way to trigger the communication manually is offered.

Edit models: this tab provides basic editing capabilities for saved models. You can view, edit and save the line and weather data. Any changes made are permanent.

Manual: this tab links to the pdf version of this manual located on the server.

Profile: this tab displays and allows you to control some basic settings of your profile, such as your email, name or password. The changes are reflected immediately after pushing the submit button.

Admin: this tab links to Django administration that allows one to edit the database including models, saved lines and weather data, run counts, users and groups. It should be used with care as the effects are irreversible.

Icon glossary

Link icon: 🔗

Hamburger button: ▤ ≡

Installation

Although the DTRi implementation is platform independent, in this manual a Debian-based system with aptitude package manager is assumed.

Prerequisites

Make sure you have **git**, the correct version of **python**, **pip** and **virtualenv** installed on the system.

- `sudo apt-get install git python3.4 python3.4-dev python3-pip`
- `git clone git@bitbucket.org:GregorKosec/dtri.git`

Python virtual environments

At its core, the main purpose of Python virtual environments is to create an isolated environment for Python projects. This means that each project can have its own dependencies, regardless of what dependencies every other project has. **To add Python virtual environment** make sure it is installed

- `sudo pip3 install virtualenv`

and then execute

- `virtualenv -p python3.4 venv`
- `source venv/bin/activate`
- `pip install -r requirements.txt`

Django

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

- Create a secrets file `web/dtri_web/dtri_web/SECRETS.py` and populate it with debug value and a secret key.
- `DEBUG = True`
- `SECRET_KEY = '[choose your secret key]'`

Initialize a **database** while in a virtual environment

- `cd web/dtri_web`
- `python manage.py migrate`
- Create an admin superuser: `python manage.py createsuperuser`

While in virtual environment run

- `python manage.py runserver`

Nginx

Engine x – Nginx is an HTTP and reverse proxy server, a mail proxy server, and a generic TCP/UDP proxy server. To set up Nginx follow:

- http://uwsgi-docs.readthedocs.org/en/latest/tutorials/Django_and_nginx.html
- `sudo apt-get install nginx`
- `sudo pip3 install uwsgi`

C++

To compile and generate docs, usual build tools are needed. That includes `cmake`, `make`, `g++` and `doxygen` for documentation generation. Protobuf packages are needed for compilation and development.

- `sudo apt-get install cmake make g++ doxygen protobuf libprotobuf-dev`

All other libraries are header only and therefore installation is not required.

Standalone execution

Main dashboard

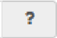
Standalone execution of a DTRi model is invoked through a web interface that gives user full control over all parameters and at the same time offers appealing presentation of results. Front end is optimized for **Chrome** browser; however other browsers might also support it.

User can execute DTRi in two main modes, namely **simulation mode** and **bisection mode**. In the simulation mode user provides the weather data and data about current, and DTRi computes skin and core temperatures. In the bisection mode user again provide weather data, but this time instead of current, a desired target skin temperature of the power line is required. According to the supplied parameters DTRi computes minimal required currents to maintain target temperature.

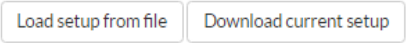
Besides physical properties user also controls all numerical parameters. All these controls can be found at the **main dashboard** (Figure 4) that is divided on two main parts; first for the parameter setup and second for analysis of computed results. The parameters are divided into the following subgroups (Figure 5)

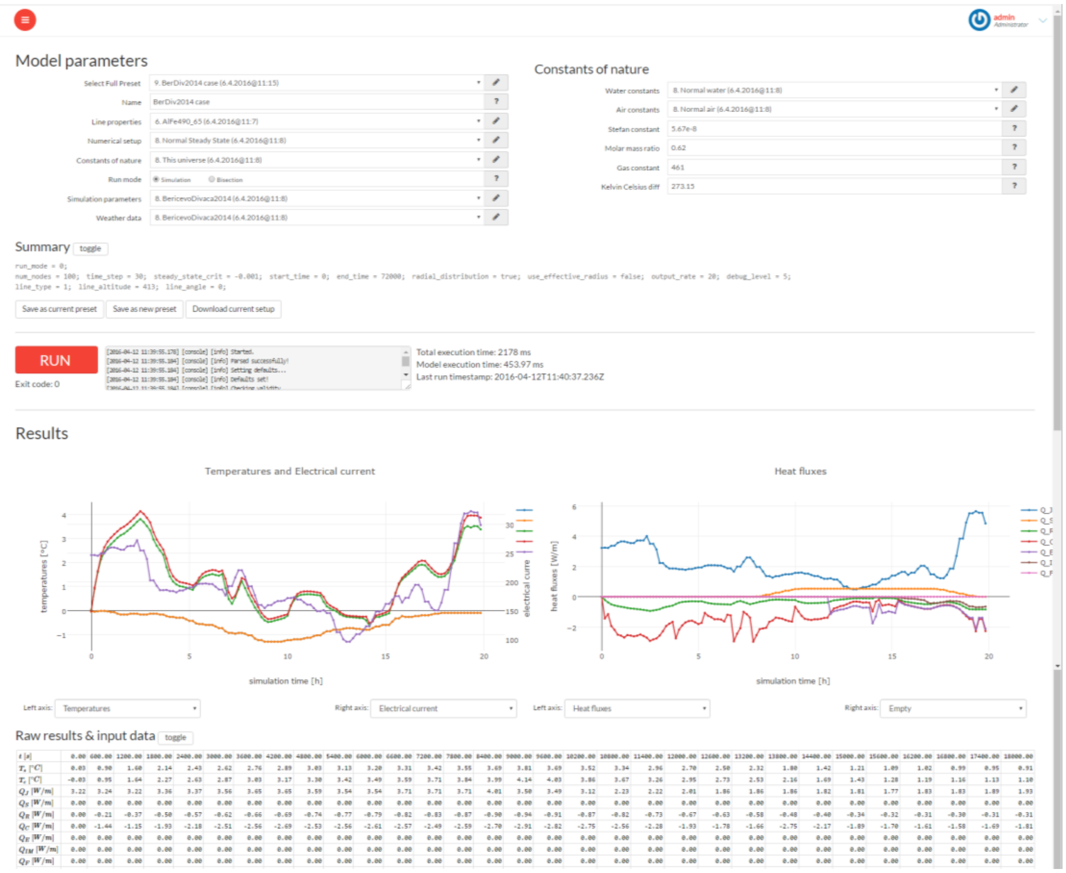
Line properties	Describe physical properties of selected overhead line -
Numerical setup	Defines all parameters for implicit Finite Differences Solution
Constants	Define all constants like Stefan constant, Gas constant, etc.
bisection parameters	Define target temperature and bisection limits
Weather data	Define all required weather data, e.g. rain rate, humidity, etc.

For detailed explanation of parameters please refer to the Feasibility study **Analiza preprečevanja nastajanja žleda z obratovalnimi ukrepi**. For **instant help** on specified parameter **hover** over **help** icon next to the parameter name. All paraemeters are summarized in **summary** section as user changes them.



Each collection of parameters can be stored in a preset. Presets can be made and edit only by administrators. However, User can download a full preset in a local file that can be later uploaded, this can be considered as a **save as/open** operations.





Model parameters

Select Full Preset: 9. BerDiv2014 case (6.4.2016@11:15)

Name: BerDiv2014 case

Line properties: 6. AlFe490_65 (6.4.2016@11:7)

Numerical setup: 8. Normal Steady State (6.4.2016@11:8)

Constants of nature: 8. This universe (6.4.2016@11:8)

Run mode: ● Simulation ○ Bisection

Simulation parameters: 8. BericevoDivaca2014 (6.4.2016@11:8)

Weather data: 8. BericevoDivaca2014 (6.4.2016@11:8)

Constants of nature

Water constants: 8. Normal water (6.4.2016@11:8)

Air constants: 8. Normal air (6.4.2016@11:8)

Stefan constant: 5.67e-8

Molar mass ratio: 0.62

Gas constant: 461

Kelvin Celsius diff: 273.15

Summary

toggle

```
run_mode = 0;
num_nodes = 100;  time_step = 30;  steady_state_crit = -0.001;  start_time = 0;  end_time = 72000;  radial_distribution = true;  use_effective_radius = false;  output_rate = 20;  debug_level = 5;
line_type = 1;  line_altitude = 413;  line_angle = 0;
```

Save as current preset | Save as new preset | Download current setup

RUN

Exit code: 0

```
[2016-04-12 11:39:55.178] [console] [info] Started.
[2016-04-12 11:39:55.184] [console] [info] Parsed successfully!
[2016-04-12 11:39:55.184] [console] [info] Setting defaults...
[2016-04-12 11:39:55.184] [console] [info] Defaults set!
[2016-04-12 11:39:55.184] [console] [info] Checking validity
```

Total execution time: 2178 ms

Model execution time: 453.97 ms

Last run timestamp: 2016-04-12T11:40:37.236Z

Results

Left axis: Temperatures

Right axis: Electrical current

Left axis: Heat fluxes

Right axis: Empty

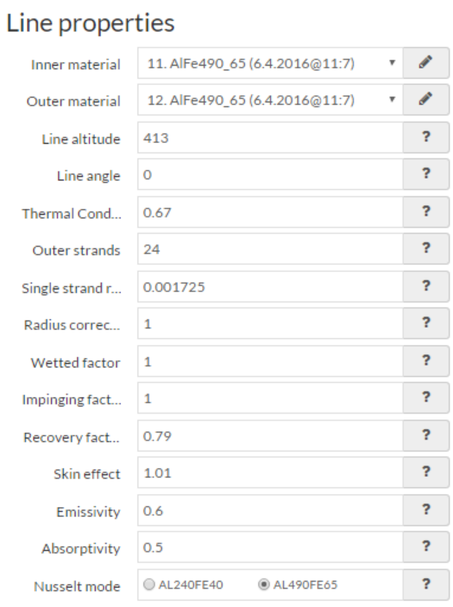
Raw results & input data

toggle

t [s]	0.00	600.00	1200.00	1800.00	2400.00	3000.00	3600.00	4200.00	4800.00	5400.00	6000.00	6600.00	7200.00	7800.00	8400.00	9000.00
T_s [°C]	0.03	0.90	1.60	2.14	2.43	2.62	2.76	2.89	3.03	3.13	3.20	3.31	3.42	3.55	3.69	3.81
T_c [°C]	-0.03	0.95	1.64	2.27	2.63	2.87	3.03	3.17	3.30	3.42	3.49	3.59	3.71	3.84	3.99	4.14
Q_J [W/m]	3.22	3.24	3.22	3.36	3.37	3.56	3.65	3.65	3.59	3.54	3.54	3.71	3.71	3.71	4.01	3.50
Q_S [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_R [W/m]	0.00	-0.21	-0.37	-0.50	-0.57	-0.62	-0.66	-0.69	-0.74	-0.77	-0.79	-0.82	-0.83	-0.87	-0.90	-0.94
Q_C [W/m]	0.00	-1.44	-1.15	-1.93	-2.18	-2.51	-2.56	-2.69	-2.53	-2.56	-2.61	-2.57	-2.49	-2.59	-2.70	-2.91
Q_E [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_{IM} [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_F [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

t [s]	9600.00	10200.00	10800.00	11400.00	12000.00	12600.00	13200.00	13800.00	14400.00	15000.00	15600.00	16200.00	16800.00	17400.00	18000.00
T_s [°C]	3.69	3.52	3.34	2.96	2.70	2.50	2.32	1.80	1.42	1.21	1.09	1.02	0.99	0.95	0.91
T_c [°C]	4.03	3.86	3.67	3.26	2.95	2.73	2.53	2.16	1.69	1.43	1.28	1.19	1.16	1.13	1.10
Q_J [W/m]	3.49	3.12	2.23	2.22	2.01	1.86	1.86	1.86	1.82	1.81	1.77	1.83	1.83	1.89	1.93
Q_S [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_R [W/m]	-0.91	-0.87	-0.82	-0.73	-0.67	-0.63	-0.58	-0.48	-0.40	-0.34	-0.32	-0.31	-0.30	-0.31	-0.31
Q_C [W/m]	-2.82	-2.75	-2.56	-2.28	-1.93	-1.78	-1.66	-2.75	-2.17	-1.89	-1.70	-1.61	-1.58	-1.69	-1.81
Q_E [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_{IM} [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_F [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 4: Main dashboard



Line properties

Inner material: 11. AlFe490_65 (6.4.2016@11:7)

Outer material: 12. AlFe490_65 (6.4.2016@11:7)

Line altitude: 413

Line angle: 0

Thermal Cond...: 0.67

Outer strands: 24

Single strand r...: 0.001725

Radius correc...: 1

Wetted factor: 1

Impinging fact...: 1

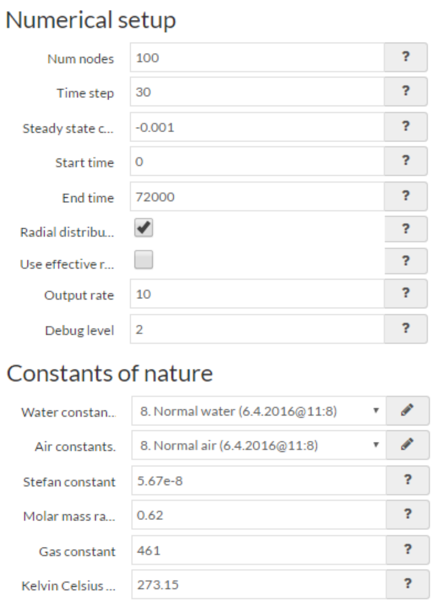
Recovery fact...: 0.79

Skin effect: 1.01

Emissivity: 0.6

Absorptivity: 0.5

Nusselt mode: ○ AL240FE40 ● AL490FE65



Numerical setup

Num nodes: 100

Time step: 30

Steady state c...: -0.001

Start time: 0

End time: 72000

Radial distribu...: ☑

Use effective r...: ☐

Output rate: 10

Debug level: 2

Constants of nature

Water constan..: 8. Normal water (6.4.2016@11:8)

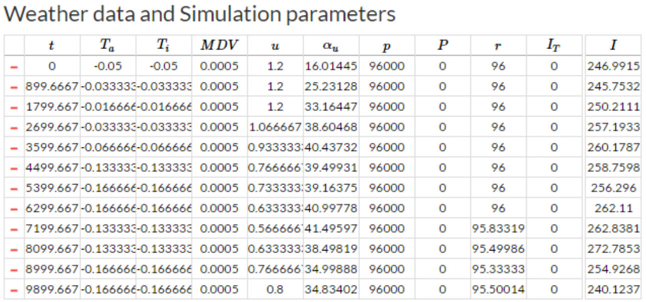
Air constants.: 8. Normal air (6.4.2016@11:8)

Stefan constant: 5.67e-8

Molar mass ra...: 0.62

Gas constant: 461

Kelvin Celsius ...: 273.15



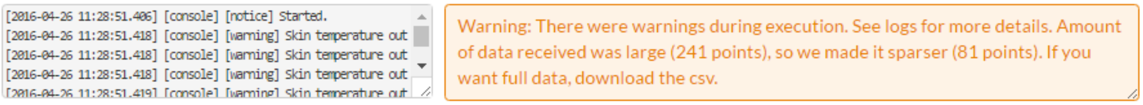
Weather data and Simulation parameters

	t	T_a	T_i	MDV	u	α_u	p	P	r	I_T	I
–	0	-0.05	-0.05	0.0005	1.2	16.01445	96000	0	96	0	246.9915
–	899.6667	-0.033333	-0.033333	0.0005	1.2	25.23128	96000	0	96	0	245.7532
–	1799.667	-0.016666	-0.016666	0.0005	1.2	33.16447	96000	0	96	0	250.2111
–	2699.667	-0.033333	-0.033333	0.0005	1.066667	38.60468	96000	0	96	0	257.1933
–	3599.667	-0.066666	-0.066666	0.0005	0.933333	40.43732	96000	0	96	0	260.1787
–	4499.667	-0.133333	-0.133333	0.0005	0.766666	39.49931	96000	0	96	0	258.7598
–	5399.667	-0.166666	-0.166666	0.0005	0.733333	39.16375	96000	0	96	0	256.296
–	6299.667	-0.166666	-0.166666	0.0005	0.633333	40.99778	96000	0	96	0	262.11
–	7199.667	-0.133333	-0.133333	0.0005	0.566666	41.49597	96000	0	95.83319	0	262.8381
–	8099.667	-0.133333	-0.133333	0.0005	0.633333	38.49819	96000	0	95.49986	0	272.7853
–	8999.667	-0.166666	-0.166666	0.0005	0.766666	34.99888	96000	0	95.33333	0	254.9268
–	9899.667	-0.166666	-0.166666	0.0005	0.8	34.83402	96000	0	95.50014	0	240.1237

Figure 5: Parameter editable tables

The computation is executed with a **RUN** button that invokes the back. Immediate after the execution the report is rendered in the resizable log window (Figure 6). Note that the level of **logging details** can be adjusted in numerical parameters section under the **debug level** (5 means all reports, 0 no logging). Also note that using debug level 5 might results in a **slow front end response** due to large number of log entries. Besides log also warning window might appear, reporting about potential inconstancies occurred during computation, typically resulting from a **bad parameter selection**.

RUN



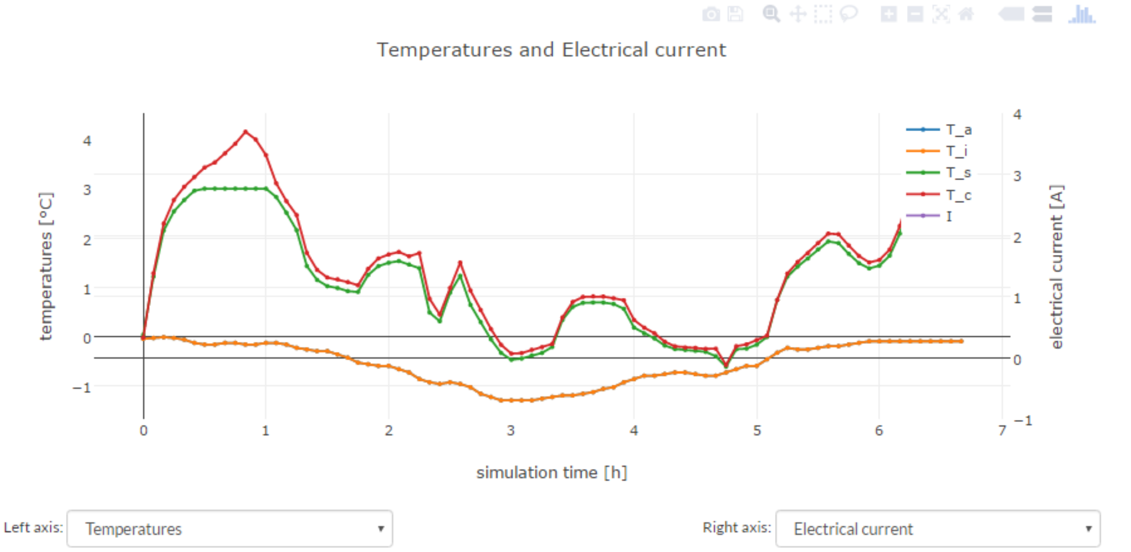
```
[2016-04-26 11:28:51.406] [console] [notice] Started.
[2016-04-26 11:28:51.418] [console] [warning] Skin temperature out
[2016-04-26 11:28:51.418] [console] [warning] Skin temperature out
[2016-04-26 11:28:51.418] [console] [warning] Skin temperature out
[2016-04-26 11:28:51.419] [console] [warning] Skin temperature out
```

Warning: There were warnings during execution. See logs for more details. Amount of data received was large (241 points), so we made it sparser (81 points). If you want full data, download the csv.

Figure 6: Log and warning windows

The **graphical representation of results** (Figure 7) follows log and warning windows. Two figures, each with two different axes offer possibility to review **four different quantities simultaneity**. Under each figure there are two dropdown menus, where user can choose which quantities he would like to review. Figures enable **scaling, panning, removing/adding series, exporting to bitmap files, comparing data on hover and inspecting data**. If user cannot analyze everything on prepared figures, he can always download pure data in a csv format and use more sophisticated software for further processing. Besides graphical representation also tabular data is presented below figures (Figure 8). Note, that **DTRi might reduce the resolution of data** for the web GUI, however, downloaded data will remain of original size.

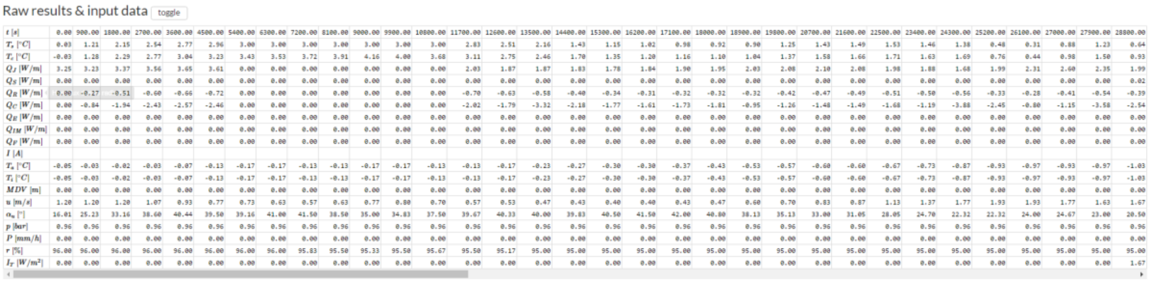
Download as CSV



Left axis: Temperatures

Right axis: Electrical current

Figure 7: Graphical representation of results.



Raw results & input data

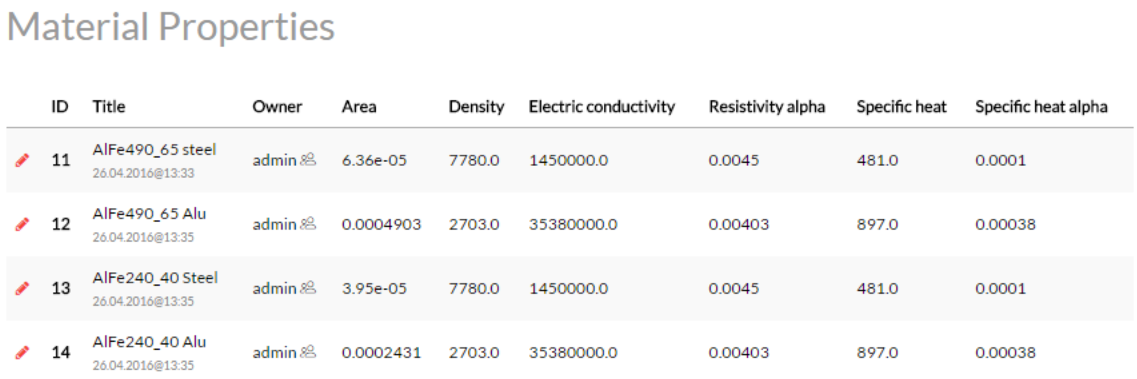
t [s]	0.00	900.00	1800.00	2700.00	3600.00	4500.00	5400.00	6300.00	7200.00	8100.00	9000.00	9900.00	10800.00	11700.00	12600.00	13500.00	14400.00
T_s [°C]	0.03	1.21	2.15	2.54	2.77	2.96	3.00	3.00	3.00	3.00	3.00	3.00	3.00	2.83	2.51	2.16	1.43
T_c [°C]	-0.03	1.28	2.29	2.77	3.04	3.23	3.43	3.53	3.72	3.91	4.16	4.00	3.68	3.11	2.75	2.46	1.70
Q_J [W/m]	3.25	3.23	3.37	3.56	3.65	3.61	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.03	1.87	1.87	1.83
Q_S [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_R [W/m]	0.00	-0.27	-0.51	-0.60	-0.66	-0.72	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.70	-0.63	-0.58	-0.40
Q_C [W/m]	0.00	-0.84	-1.94	-2.43	-2.57	-2.46	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-2.02	-1.79	-3.32	-2.18
Q_E [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_{IM} [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_P [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I [A]																	
T_a [°C]	-0.05	-0.03	-0.02	-0.03	-0.07	-0.13	-0.17	-0.17	-0.13	-0.13	-0.17	-0.17	-0.13	-0.13	-0.17	-0.23	-0.27
T_i [°C]	-0.05	-0.03	-0.02	-0.03	-0.07	-0.13	-0.17	-0.17	-0.13	-0.13	-0.17	-0.17	-0.13	-0.13	-0.17	-0.23	-0.27
MDV [m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
u [m/s]	1.20	1.20	1.20	1.07	0.93	0.77	0.73	0.63	0.57	0.63	0.77	0.80	0.70	0.57	0.53	0.47	0.43
α_u [°]	16.01	25.23	33.16	38.60	40.44	39.50	39.16	41.00	41.50	38.50	35.00	34.83	37.50	39.67	40.33	40.00	39.83
p [bar]	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
P [mm/h]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r [%]	96.00	96.00	96.00	96.00	96.00	96.00	96.00	96.00	95.83	95.50	95.33	95.50	95.67	95.50	95.17	95.00	95.00
I_T [W/m^2]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

t [s]	15300.00	16200.00	17100.00	18000.00	18900.00	19800.00	20700.00	21600.00	22500.00	23400.00	24300.00	25200.00	26100.00	27000.00	27900.00	28800.00
T_s [°C]	1.15	1.02	0.98	0.92	0.90	1.25	1.43	1.49	1.53	1.46	1.38	0.48	0.31	0.88	1.23	0.64
T_c [°C]	1.35	1.20	1.16	1.10	1.04	1.37	1.58	1.66	1.71	1.63	1.69	0.76	0.44	0.98	1.50	0.93
Q_J [W/m]	1.78	1.84	1.90	1.95	2.03	2.08	2.10	2.08	1.98	1.88	1.68	1.99	2.31	2.60	2.35	1.99
Q_S [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02
Q_R [W/m]	-0.34	-0.31	-0.32	-0.32	-0.32	-0.42	-0.47	-0.49	-0.51	-0.50	-0.56	-0.33	-0.28	-0.41	-0.54	-0.39
Q_C [W/m]	-1.77	-1.61	-1.73	-1.81	-0.95	-1.26	-1.48	-1.49	-1.68	-1.19	-3.88	-2.45	-0.80	-1.15	-3.58	-2.54
Q_E [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_{IM} [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Q_P [W/m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
I [A]																
T_a [°C]	-0.30	-0.30	-0.37	-0.43	-0.53	-0.57	-0.60	-0.60	-0.67	-0.73	-0.87	-0.93	-0.97	-0.93	-0.97	-1.03
T_i [°C]	-0.30	-0.30	-0.37	-0.43	-0.53	-0.57	-0.60	-0.60	-0.67	-0.73	-0.87	-0.93	-0.97	-0.93	-0.97	-1.03
MDV [m]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
u [m/s]	0.40	0.40	0.43	0.47	0.60	0.70	0.83	0.87	1.13	1.37	1.77	1.93	1.93	1.77	1.63	1.67
α_u [°]	40.50	41.50	42.00	40.00	38.13	35.13	33.00	31.05	28.05	24.70	22.32	22.32	24.00	24.67	23.00	20.50
p [bar]	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96	0.96
P [mm/h]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
r [%]	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00	95.00
I_T [W/m^2]	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.67

Figure 8: Tabular form of results.

Editing properties

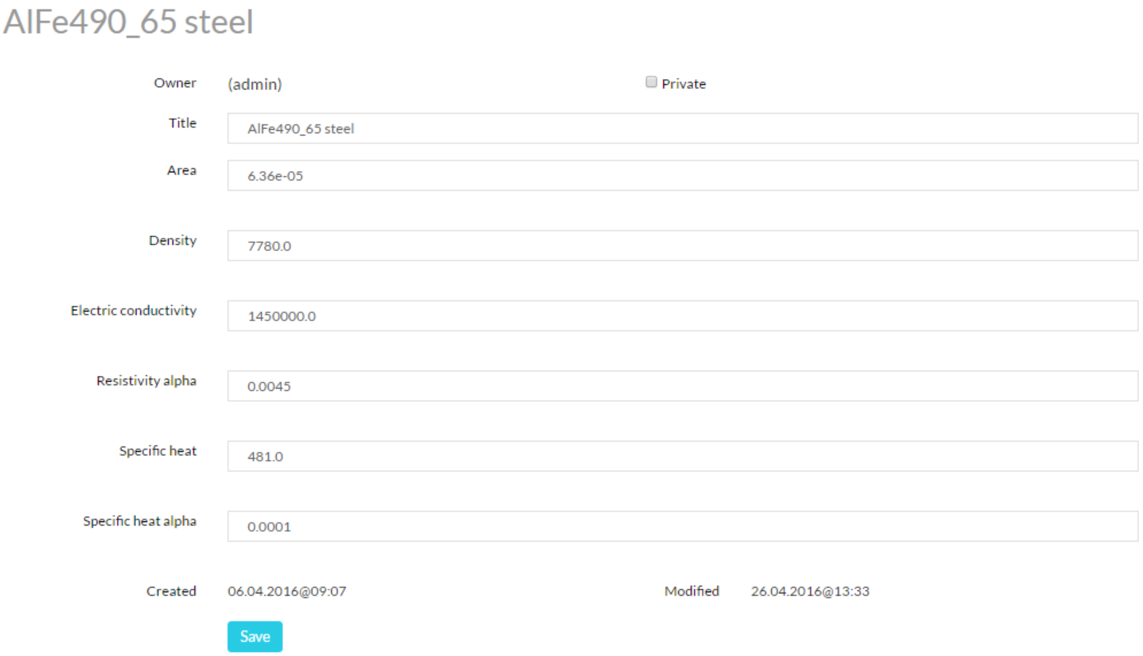
User can access and edit most of the preset data, however, he cannot add new and delete data elements that is possible only with administration access. Example of line material properties is presented in Figure 9. By pressing an **edit icon** user starts editing the selected data Figure 10.



Material Properties

	ID	Title	Owner	Area	Density	Electric conductivity	Resistivity alpha	Specific heat	Specific heat alpha
✎	11	AlFe490_65 steel 26.04.2016@13:33	admin	6.36e-05	7780.0	1450000.0	0.0045	481.0	0.0001
✎	12	AlFe490_65 Alu 26.04.2016@13:35	admin	0.0004903	2703.0	35380000.0	0.00403	897.0	0.00038
✎	13	AlFe240_40 Steel 26.04.2016@13:35	admin	3.95e-05	7780.0	1450000.0	0.0045	481.0	0.0001
✎	14	AlFe240_40 Alu 26.04.2016@13:35	admin	0.0002431	2703.0	35380000.0	0.00403	897.0	0.00038

Figure 9: List of material properties in edit mode



AlFe490_65 steel

Owner	(admin)	☐ Private	
Title	AlFe490_65 steel		
Area	6.36e-05		
Density	7780.0		
Electric conductivity	1450000.0		
Resistivity alpha	0.0045		
Specific heat	481.0		
Specific heat alpha	0.0001		
Created	06.04.2016@09:07	Modified	26.04.2016@13:33

Figure 10: Example of editing data form

SUMO CONNECT

Basic concept

The aim of the SUMO system is to improve reliability and safety of operating of the transmission system, e.g. in cases of sudden increases of power flows, and to better utilize the existing transmission system infrastructure, e.g. when new transfer capacities, such as new power lines are introduced. The SUMO system combines different subsystems into a meaningful and helpful power grid operation support tool. It comprises the following functions (Figure 11):

- Measurements: currents from **SCADA**, measured data from weather stations, gridded weather data applied to micro locations using weather model and terrain data (**OIAP**),
- Reliability analyses: N-1 analyses, Line Outage Distribution Factors – **LODF** (power flow calculations).
- Forecasts: short term load flow forecasts (**NOV**), short term weather forecasts for corridors of power lines (**OIAP**).
- Dynamic thermal ratings (**DTR**) – calculations based on: current weather and forecasted weather (t0 ... t0+3h).
- Exceptional weather events.
- Visualization: **ODIN**.
- Integration platform and data exchange: **SUMO BUS**

Data such as line loading, physical properties of the conductors, corridor geography, weather data needed by DTRi are available on the SUMO BUS for classical DTR and OIAP module (alarming on exceptional weather data). All information can be accessed through implementation of the standardized web services (WS).

On the web interface user can select load from SUMO BUS button to access data on the SUMO BUS.

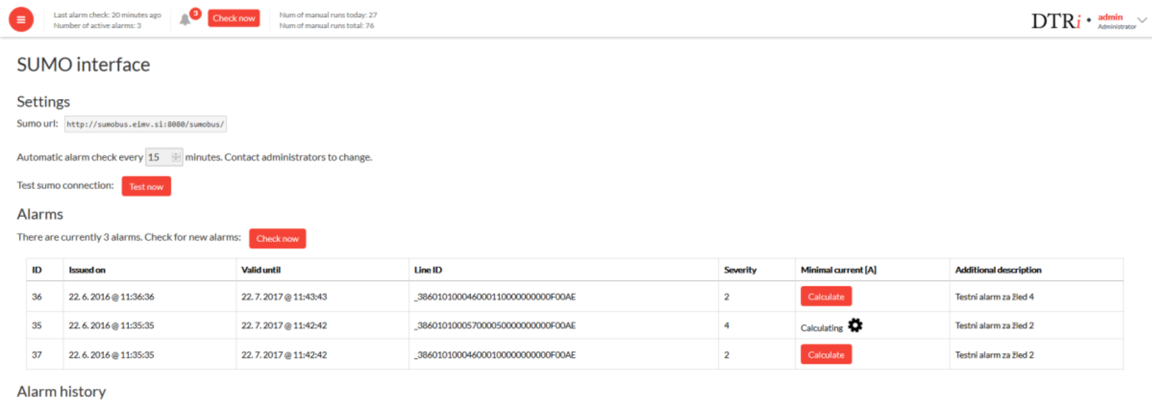
SUMO interface

Interaction with SUMO system is available via the SUMO interface tab. The SUMO BUS url and alarm checking frequency are displayed. This can be changed in the files `sumo/automatic.py` and in `sumo/cron.py`. The connection can be tested by clicking the Test now button.

Alarm checking

Our system periodically checks the SUMO system for alarms. Only alarms for glaze are filtered out and save in our database. The alarm check can also be performed manually from the top bar of every page. In the SUMO connect page, a history of all alarm checks and a list of past and currently active alarms can be seen.

The automatic alarms checks also automatically run the model for the line in consideration and find the minimal possible current that prevents glazing. For alarms that were found manually, the calculation can be started by hand by clicking the calculate button. If the calculation is already running it will not start again. The automatic runs and checks are performed by a user called `system` and this username is reserved.



Last alarm check: 20 minutes ago
Number of active alarms: 3

Num of manual runs today: 27
Num of manual runs total: 76

SUMO interface

Settings

Sumo url: `http://sumobus.eimv.si:8080/sumobus/`

Automatic alarm check every 15 minutes. Contact administrators to change.

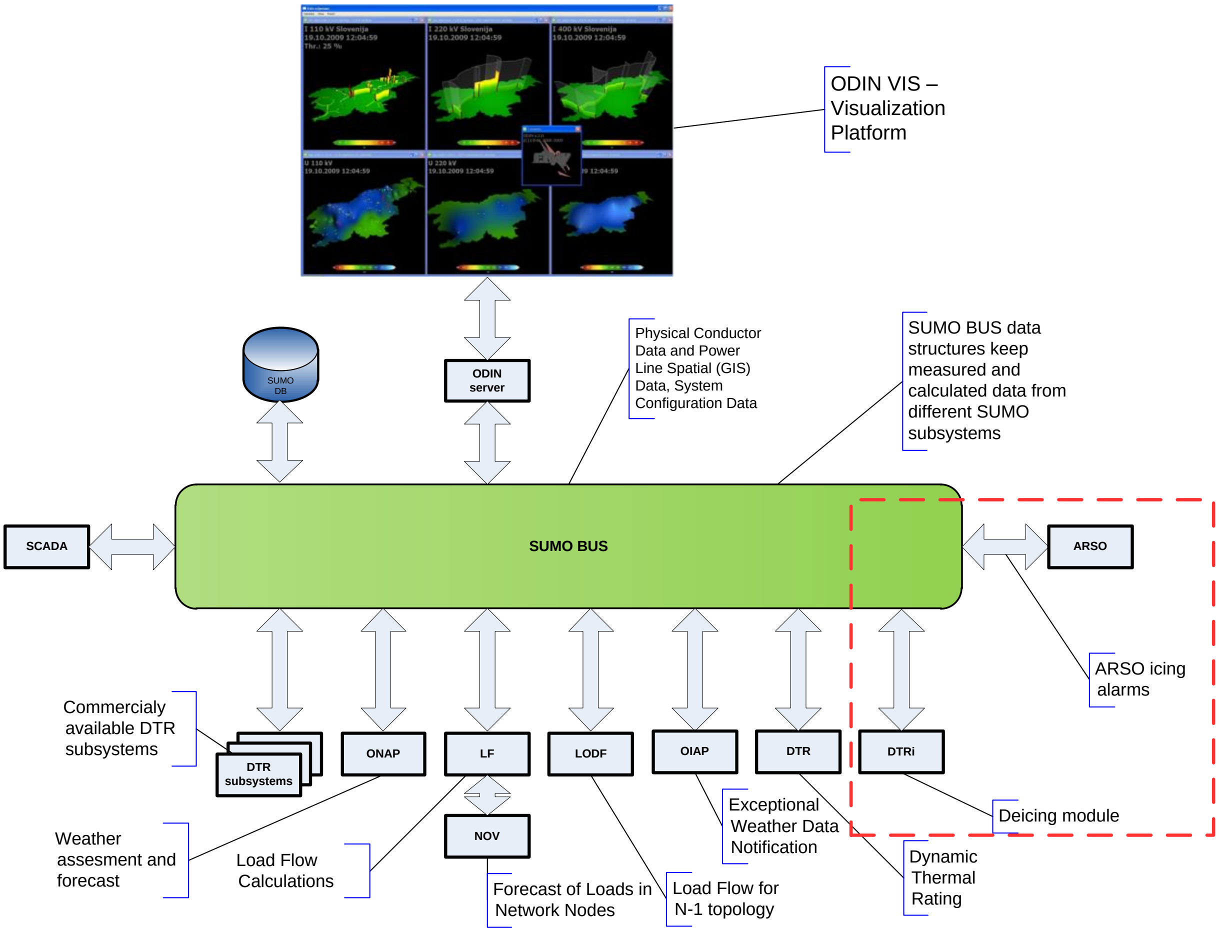
Test sumo connection:

Alarms

There are currently 3 alarms. Check for new alarms:

ID	Issued on	Valid until	Line ID	Severity	Minimal current [A]	Additional description
36	22. 6. 2016 @ 11:36:36	22. 7. 2017 @ 11:43:43	_38601010004600011000000000F00AE	2	Calculate	Testni alarm za žled 4
35	22. 6. 2016 @ 11:35:35	22. 7. 2017 @ 11:42:42	_38601010005700005000000000F00AE	4	Calculating	Testni alarm za žled 2
37	22. 6. 2016 @ 11:35:35	22. 7. 2017 @ 11:42:42	_38601010004600010000000000F00AE	2	Calculate	Testni alarm za žled 2

Alarm history



I 110 kV Slovenija
19.10.2009 12:04:59
Thr.: 25 %
I 220 kV Slovenija
19.10.2009 12:04:59
I 400 kV Slovenija
19.10.2009 12:04:59
U 110 kV
19.10.2009 12:04:59
U 220 kV
19.10.2009 12:04:59
ODIN VIS – Visualization Platform
SUMO DB
ODIN server
Physical Conductor Data and Power Line Spatial (GIS) Data, System Configuration Data
SUMO BUS data structures keep measured and calculated data from different SUMO subsystems
SCADA
SUMO BUS
ARSO
ARSO icing alarms
Commercialy available DTR subsystems
DTR subsystems
ONAP
LF
LODF
OIAP
DTR
DTRi
NOV
Deicing module
Weather assesment and forecast
Load Flow Calculations
Forecast of Loads in Network Nodes
Load Flow for N-1 topology
Exceptional Weather Data Notification
Dynamic Thermal Rating

Figure 11: SUMO concept

Appendix

Protocol buffer message definition

```
package message;
// Parameters for the DTRi model.
message ModelParameters {
    required LineProperties line_properties      = 1;  // properties of the overhead line
    required WeatherData weather_data              = 2;  // weather data array
    required ConstantsOfNature constants_of_nature = 3;  // physical constants
    required NumericalSetup numerical_setup        = 4;  // all numerical parameters
    // Which mode to run in.
    enum RunMode {
        SIMULATION = 0;  // either numerical simulation of temperature with given current
        BISECTION = 1;   // or finding current to achieve given temperature
    }
    required RunMode run_mode                           = 5;  // which mode to run in
    optional SimulationParameters simulation_parameters = 7;  // exactly one of these should be set
    optional BisectionParameters bisection_parameters   = 6;  // otherwise data is considered invalid
}

// Message representing properties of the overhead line and the materials is built from.
message LineProperties {
    required MaterialProperties inner_material = 1;  // properties of the inner material
    required MaterialProperties outer_material = 2;  // properties of the outer material

    required double line_altitude          = 3;    // altitude (height above sea level) of the part of the line for which the
simulation is run
    required double line_angle           = 4;   // line angle with respect to oriented map NW = 90, SE = 0 [deg]
    required double thermal_conductivity = 5;   // thermal conductivity of the line (outer material) [W / m K]
    required double num_outer_strands    = 6;   // number of outer strands
    required double single_strand_radius = 7;   // radius of a single strand [m]
    required double radius_correction    = 8;   // correction of line radius due to strand packing
    required double wetted_factor        = 9;   // ratio of wetted area of conductor (used for evaporation)
    required double impinging_factor     = 10;  // ratio of impinging water that reaches the skin temp. 0.7 [Zsolt]
    required double recovery_factor      = 11;  // recovery factor (= 0.79) (friction heating)
    required double skin_effect          = 12;  // skin effect factor
    required double emissivity           = 13;  // emissivity of the line (outer material)
    required double absorptivity         = 14;  // absorptivity of the line (outer material)

    // Different lines have different ways of calculating Nusselt number.
    enum NusseltType {
        AL240FE40 = 0;
        AL490FE65 = 1;
    }
    required NusseltType nusselt_mode    = 15;  // how to calculate the convection coefficient -- depends on line type
}

// Representing physical properties of a material.
message MaterialProperties {
    required double density               = 1;   // density of the material [kg / m^3]
    required double specific_heat         = 2;   // isobaric specific heat capacity [J / kg K]
    required double specific_heat_alpha   = 3;   // specific heat linear temperature coefficient [1 / K]
    required double area                  = 5;   // cross section area of the material [m^2]
    required double resistivity_alpha     = 6;   // resistivity temperature coefficient [1 / K]
    required double electric_conductivity = 7;   // specific electric conductivity [1 / m ohm]
}
// A level of nesting for nicer gui generation and extensibility.
message WeatherData {
    repeated WeatherPoint weather_points = 1;  // list of weather measurements
}
// A single weather measurement.
message WeatherPoint {
    required double time                = 1;   // time when data was measured (timestamp from epoch) [s]
    required double ambient_temperature = 2;   // ambient air temperature [deg C]
    optional double droplet_temperature = 3;   // droplet temperature [deg C]
    required double droplet_MDV         = 4;   // median of the droplet diameter [m]
    required double wind_velocity       = 5;   // wind velocity [m / s]
    required double wind_angle          = 6;   // wind angle relative to absolute map NW = 90, SE = 0 [deg]
    required double pressure            = 7;   // air pressure [Pa]
    required double rain_rate           = 8;   // rain rate [mm / h]
    required double humidity            = 9;   // humidity [%]
    required double solar_irradiance    = 10;  // solar irradiance [W / m^2]
}

// Class representing physical constants.
message ConstantsOfNature {
    required WaterConstants water       = 1;  // water constants
    required AirConstants air           = 2;  // air constants
    required double stefan_constant     = 3;  // Stephan constant [W / m^2 K^4]
    required double molar_mass_ratio    = 4;  // ratio of molar masses of water vapor and dry air
    required double gas_constant        = 5;  // gas constant [J / K mol]
    required double kelvin_celsius_diff = 6;  // difference between Celsius and kelvin unit (positive number) [deg C]
}

// Class representing physical constants of water.
message WaterConstants {
    required double density                 = 1;  // water density [kg / m^3]
    required double latent_heat_fusion      = 2;  // latent heat of fusion (freezing) [J / kg]
    required double latent_heat_evaporation = 3;  // latent heat of evaporation [J / kg]
    required double latent_heat_sublimation = 4;  // latent heat of sublimation [J / kg]
    required double specific_heat_ice       = 5;  // specific heat of ice (cp) [J / kg K]
    required double specific_heat_water     = 6;  // specific heat of water [J / kg K]
}

// Class representing physical constants of air.
message AirConstants {
    required double specific_heat        = 1;  // specific heat of air [J / kg K]
    optional double density              = 2;  // density of air -- if not present, computed during simulation [kg / m^3]
    optional double viscosity            = 3;  // kinematic viscosity of air -- as above [m^2 / s]
    optional double thermal_conductivity = 4;  // thermal conductivity of air -- as above [W / m K]
}

// Class containg data about the numerical setup.
message NumericalSetup {
    required int32 num_nodes           = 1;  // number of nodes in discretization
    required double time_step          = 2;  // time step of the implicit Euler [s]
    required double steady_state_crit  = 3;  // finish when temperature changes less than this (negative value disables it)
    optional double start_time         = 4;  // start simulation at this time, not the first one in the data
    optional double end_time           = 5;  // end simulation at this time, not the last one in the data
    required bool radial_distribution  = 6;  // do we use radial distribution of lambda or not
    required bool use_effective_radius = 7;  // do we use effective radius due strand packing
    required int32 output_rate         = 8;  // write data every `output_rate` time steps, if 0, don't write at all
    required int32 debug_level          = 9;  // how verbose do you want the output to be (0 = no output, 2 = error, 5 = info, 7 =
full trace)
}

// If bisection mode was chosen, this parameters must be specified.
message BisectionParameters {
    required double target_temperature = 1;  // target skin temperature [deg C]
    required double min_current        = 2;  // bisection min current [A]
    required double max_current        = 3;  // bisection max current [A]
    required double precision          = 4;  // how precisely to determine the current [A]
}
```

```
// If simulation mode was chosen, electrical current data must be provided.
message SimulationParameters {
    repeated double electrical_current = 1;  // current at each time point
}
// Results of a bisection run.
message BisectionResponse {
    repeated double time                    = 1;   // time of this data entry
    repeated double electrical_current      = 2;   // current to use at each time to achieve desired temperature
    repeated double core_temperature        = 3;   // calculated temperature of the line core
    repeated double skin_temperature        = 4;   // calculated temperature of the line surface
    repeated double heat_flux_radiation     = 5;   // heat fluxes out of line due to radiation per meter length [W / m]
    repeated double heat_flux_convection    = 6;   // heat fluxes out of line due to convection per meter length [W / m]
    repeated double heat_flux_solar         = 7;   // heat fluxes out of line due to solar irradiation per meter length [W / m]
    repeated double heat_flux_friction      = 8;   // heat fluxes out of line due to friction per meter length [W / m]
    repeated double heat_flux_impinging     = 9;   // heat fluxes out of line due to impinging per meter length [W / m]
    repeated double heat_flux_evaporation   = 10;  // heat fluxes out of line due to evaporation per meter length [W / m]
    repeated double heat_flux_joule_heating = 11;  // heat flux from electrical current due to joule heating per meter length [W /
m]
    optional int32 num_iterations           = 12;  // number of iterations of bisection at each time step
}

// Results of a simulation run.
message SimulationResponse {
    repeated double time                    = 1;   // time of this data entry
    repeated double core_temperature        = 2;   // calculated temperature of the line core
    repeated double skin_temperature        = 3;   // calculated temperature of the line surface
    repeated double heat_flux_radiation     = 4;   // heat fluxes out of line due to radiation per meter length [W / m]
    repeated double heat_flux_convection    = 5;   // heat fluxes out of line due to convection per meter length [W / m]
    repeated double heat_flux_solar         = 6;   // heat fluxes out of line due to solar irradiation per meter length [W / m]
    repeated double heat_flux_friction      = 7;   // heat fluxes out of line due to friction per meter length [W / m]
    repeated double heat_flux_impinging     = 8;   // heat fluxes out of line due to impinging per meter length [W / m]
    repeated double heat_flux_evaporation   = 9;   // heat fluxes out of line due to evaporation per meter length [W / m]
    repeated double heat_flux_joule_heating = 10;  // heat flux from electrical current due to joule heating per meter length [W /
m]
}

// List of errors and respective error codes for cpp. The codes returned have
// the same value but are negative.
enum Error {
    INVALID_MODE = 1;
    MISSING_SIMULATION_PARAMETERS = 2;
    WEATHER_AND_CURRENT_DATA_MISMATCH = 3;
    MISSING_BISECTION_PARAMETERS = 4;
    NUMBER_OF_NODES_TOO_SMALL = 5;
    NEGATIVE_OR_ZERO_TIME_STEP = 6;
    MISSING_WEATHER_DATA = 7;
    NON_INCREASING_TIME_STAMPS = 8;
    START_TIME_TOO_SMALL = 9;
    END_TIME_BEFORE_OR_EQUAL_START_TIME = 10;
    BISECTION_PRECISION_TOO_SMALL = 12;
    BISECTION_LOW_NOT_SMALLER_THAN_BISECTION_HIGH = 13;
    TARGET_TEMPERATURE_NOT_ACHIEVABLE = 14;
    TARGET_TEMPERATURE_OUT_OF_MODELLING_RANGE = 15;
};
```