ResLoadSIM

Version 6.2.0

Christoph Troyer
Heinz Wilkening

04 November 2024

Content

Introduction

ResLoadSIM is a stochastic bottom-up simulation tool for predicting electric residential load profiles of individual households with a high time resolution. For each household some individual characteristics are defined, e.g. number of residents, the size of the house or apartment and the devices used according to some statistical information. Each device is characterized by its energy class. In the simulation devices are used according to the estimated habits of the inhabitants. Weekly and seasonal variations are also captured. Beside using the tool for load prediction it is also possible to study the potential for load shifting in private households by implementing different control mechanisms based e.g. on variable pricing.

ResLoadSIM is released under the terms of the 3-Clause BSD License:

1 Installation

1.1 Requirements

Compiler

The most basic requirement is a C++ compiler. On a Linux box it is very likely that the GNU Compiler Collection (GCC) is already installed. If not, get it from there: http://gcc.gnu.org

Under Windows ResLoadSIM has been successfully compiled with Visual Studio, which you can get here for free: https://visualstudio.microsoft.com

Users of macOS get a free development environment called Xcode. It can be downloaded from

Apple's App Store.

CMake (optional)

CMake is a set of tools that help to manage the build process of a software package. It isn't a requirement for building ResLoadSIM, but it makes things a lot easier. If you like to setup your makefiles by hand for example, or use an IDE like Xcode, which uses its own build system, then you can bypass CMake, otherwise get it from here: http://www.cmake.org. Visual Studio ships with an integrated CMake, so you don‘t need to worry about it.

MPI (optional)

To compile and execute the parallel version of ResLoadSIM you need to install an implementation of the Message Passing Interface (MPI). One possibility is Open MPI, which can be found here: http://www.open-mpi.org

PETSc (optional)

If ResLoadSIM is intended to be used with grid voltage control enabled, the PETSc package needs to be installed. It can be obtained via git:

```
git clone https://bitbucket.org/petsc/petsc.git
```

PETSc contains a power flow solver named pflow. The pflow code has been modified by Shrirang Abhyankar to make it work together with ResLoadSIM. This updated version of pflow is available at: https://github.com/abhyshr/pflow.git .

libcurl (optional)

This library allows ResLoadSIM to download data from the PVGIS site automatically.

1.2 Compiling and Linking

The ResLoadSIM source package is distributed as a compressed archive (*.tar.gz). The first thing you have to do is to uncompress the archive and extract the source code by executing the following command on the command-line:

```
tar xvfz resLoadSIM-x.y.z-Source.tar.gz
```

Unix/Linux/macOS

Change into the directory of the decompressed package

```
cd resLoadSIM-x.y.z-Source
```

and create a build directory there, like this

```
mkdir build
```

You can name the directory as you wish and you can have more than one build directory for different builds (e.g. debug, parallel). Change into your newly created build directory and start the build process using CMake:

```
cd build
cmake ..
```

CMake checks whether everything needed for ResLoadSIM is in place. If everything is ok, then CMake generates the makefiles needed for compiling the package. Start the compilation using make:

```
make
```

To install ResLoadSIM type

```
make install
```

If no errors occur, then at this point ResLoadSIM is compiled, linked and ready to be used.

CMake will by default create a makefile for the optimized version of ResLoadSIM. If you need the debug version you have to call CMake with the following option:

```
cmake -DCMAKE_BUILD_TYPE=DEBUG ..
```

To get the parallel version of ResLoadSIM call

```
cmake -DPARALLEL:BOOL=ON ..
```

Windows

Under Windows you have several possibilities to build ResLoadSIM, but we assume that you use Visual Studio. It is the only option we have tried so far and it works pretty easy. Just fire it up and select the *open local directory* option in the start window. Navigate to the previously decompressed source directory (resLoadSIM-x.y.z-Source) and select it. Visual Studio will automatically invoke CMake. After that just press F7 and watch how ResLoadSIM is being compiled and linked.

2 Execution

Most of the parameters are read from configuration files, but two of them have to be passed to ResLoadSIM on the command-line. These are the number of households and the number of days. A simulation of 10.000 households over a period of three days would be initiated as follows:

```
resLoadSIM 10000 3
```

The parallel version is initiated by using the *mpirun* command, which is part of the MPI installation, e.g.:

```
mpirun -np 4 resLoadSIM 10000 3
```

In the above eample ResLoadSIM would be executed on 4 cpus (or cores). Using the -v option prints the current version (number, debug/optimized/parallel) of ResLoadSIM:

```
resLoadSIM -v
```

If you dont‘t want to see any status output on screen while simulating then run ResLoadSIM in silent mode by using the -s option (warnings and error messages will still be displayed):

```
resLoadSIM -s 1000 365
```

ResLoadSIM depends on a number of input files which are covered in the following chapter. It also depends on a certain directory structure being in place such that it can find those files. Have a look at the *example* directory, which demonstrates the minimum input setup for a simulation.

3 Configuration Files

3.1 resLoadSIM.json

This is the main configuration file. It contains parameters which control the behavior of the simulation and is basically a list of keyword-value pairs, so called settings, which are organized in groups enclosed by curly braces. The settings can be given in any order, and not all of them need to be specified. It is actually possible to run ResLoadSIM without *resLoadSIM.json* at all. In that case missing settings are substituted by hardcoded default values. Information about what values have been used is written to a log file named *resLoadSIM.json.log*. The format of this log file is the same as the format of the configuration file itself. Therefore it is possible to use the log file as the configuration for a subsequent simulation just by renaming it to *resLoadSIM.json*. This is especially handy when you start from scratch without your own configuration. Just run a short dummy simulation, edit the settings in the resulting logfile (all defaults) according to your needs, and rename the file to *resLoadSIM.json*.

The following settings are known to ResLoadSIM:

location

String value
Default: Hannover
The following locations are available:
Amsterdam, Berlin, Copenhagen, Hannover, Oslo, Rome, Ulm, Vienna and Warsaw.
You can add a location by adding a subdirectory with the name of the new location under the locations directory. This new subdirectory must contain a file named *location.json* and optionally a PVGIS data file that contains the temperature and irradiation data of the new location.

pv_data_file_name

String value
The name of a PVGIS timeseries file. If no file name is provided, ResLoadSIM will try to fetch suitable data from the PVGIS site [1] based on the coordinates specified in *location.json*. This works only in case ResLoadSIM was compiled with libcurl support. Without libcurl installed the user has to download the data manually.

pv_forecast_file_name

String value
The name of a solar irradiation forecast file. It has the same format as the PVGIS file and is used only in case a battery charging strategy is used together with the production forecast method set to 3.

solar_production_reference_year

Array of integer values
Default: first year of the PVGIS timeseries
If the solar modules‘ production ratio given in *tech.json* is > 0, ResLoadSIM performs a prerun simulation to calculate the necessary nominal power of the solar modules. It uses the solar irradiation data from the year specified in this array. If the array consists more than one value, then the average irradiation values from all years in the array are used.

battery_charging
{

strategy

Integer value in the range [0, 3]
Default: 0
0: Charge whenever the solar production exceeds the consumption.
1: Deprecated method - not supported anymore.
2: First feed to grid, then charge the batteries.
3: Like strategy 0 as long as the battery charge is below a given threshold. Above that threshold switch to strategy 2.

production_forecast_method

Integer value in the range [0, 5]
Default: 0
0: No forecast (makes only sense with strategy = 0).
1: Perfect forecast of solar production (use PVGIS data).
2: Use the production of the previous day as a forecast.
3: Read the solar forecast data from a file (see pv_forecast_file_name).

feed_in_limit

Decimal value in the range [0.0, 1.0]
Default: 0.5
Fraction of a solar module‘s nominal power. If the solar production exceeds the feed-in limit, then all production above the limit is sent to the batteries in order to protect the network.

precharge_threshold

Decimal value in the range [0.0, 1.0]
Default: 0.1
Used only if strategy 3 is selected. This is the fraction of a battery‘s total capacity up to which the battery is charged in the morning before the strategy switches to 2.

shared

Boolean value
Default: FALSE
If set to TRUE neighbouring households can share batteries.

}

powerflow
{

step_size

Integer value >= 0
Default: 0
If set to 0, the powerflow solver is not used. Any value greater than 0 is interpreted as the number of timesteps between two invocations of the powerflow solver.

case_file_name

String value
Default: empty string
Name of the case file used by the powerflow solver.

uv_control

Boolean value
Default: FALSE
Turn on/off undervoltage control

uv_lower_limit

Decimal value
Default: 0.910

Lower limit for undervoltage control

uv_upper_limit

Decimal value
Default: 0.925
Upper limit for undervoltage control

ov_control

Boolean value
Default: FALSE
Turn on/off overvoltage control

ov_lower_limit

Decimal value
Default: 1.075
Lower limit for overvoltage control

ov_upper_limit

Decimal value
Default: 1.090
Upper limit for overvoltage control

output_level

Integer value
Default: 1
0: no output related to the power flow solver
1: transformer files only
2: transformer files, partial input/output of the power flow solver
3: transformer files, full input/output of the power flow solver

}

control

Integer value in the range [0, 4]
Default: 0
This parameter sets the control mode of the producer.

0: No control. The simulation of the producer is simply bypassed.
1: Peak shaving. The producer tries to keep the load below a given limit (see peak_shaving_threshold).
2: The producer tries to control the load such that it matches a predefined load profile. The load profile is read by ResLoadSIM from a file named *profile*.
3: The producer tries to compensate a gap between projected and actual production.
4: Decentralized control via the electricity tariff.

peak_shaving
{

relative

Boolean value
Default: TRUE

threshold

Decimal value
Default: 85.0
This value is used only if the setting **control** = 1
When **relative** is TRUE, **threshold** is interpreted as a percentage (a value between 0 and 100) of the maximum load. If **relative** is FALSE, **threshold** is given in kWh.

}

seed

Integer value
Default: 0
Seed of the random number generator. If set to 0, the current time is used as seed. Any value greater than 0 is interpreted as seed itself.

output

Integer value in the range [0, 2]
Default: 1
0: All output data is written to a single file.
1: One output file per appliance.
2: Both. One file per appliance plus one file with all data.

start
{

These settings define the date and time for the simulation start.

day

Integer value in the range [1, 31]
Default: 1

month

Integer value in the range [1, 12]
Default: 4

year

Integer value
Default: 2015

time

Decimal value in the range [0.0, 24.0]
Default: 0.0

}

transient_time

Decimal value
Default: 1.0
The number of days of the pre-simulation run. In some cases it is desirable to add a few simulation days so that potential transient oscillations have time to settle before the actual simulation starts. In other cases this is even a necessity:
• When using peak-shaving we need to have a reference value for the maximum power peak
• In case there are households with a PV installation with batteries we need to initialize the batteries properly.
• Some appliances like the TVs need one day for initialization
• If the annual production of the PV modules is given as a fraction of the annual household consumption, the pre-simulation run has to be extended by a whole year to get values for the annual consumption.

daylight_saving_time

Integer value in the range [0, 2]
Default: 1
0: no daylight saving time (wintertime only)
1: standard daylight saving time
2: permanent daylight saving time (summertime only)

timestep_size

Decimal value
Default: 60.0
The timestep size given in seconds.

simulate_heating

Boolean value
Default: TRUE
Activate the simulation of the space heating demand?

ventilation_model

Boolean value
Default: FALSE
Activate/deactivate air ventilation in the space heating simulation

variable_load

Boolean value
Default: FALSE
Activate/deactivate the simulation of a variable load for some appliances

comments_in_logfiles

Boolean value
Default: TRUE
Turn on/off comments in logfiles. Comments make it easier to edit a configuration file, but when a logfile is intended to be used as the input for a subsequent simulation, the comments have to be removed, because JSON does not support comments.

energy_classes_2021

Boolean value
Default: TRUE
Use the new energy efficiency class labels as introduced in 2021?

price_grid
{

profiles

A list of price profiles. Each profile is enclosed in parentheses and contains a set of arrays. Each array consists of 3 floating point values: start time, end time (in hours) and the price per kWh. A profile covers a whole day.
Default: ([0.00, 6.00, 0.20][6.00, 22.00, 1.00][22.00, 24.00, 0.20])

sequence

An array of integer values.
Default: [1]
The values in this array are indices pointing to members of the list of profiles. The sequence [1, 3, 2] for example means that the simulation will take the first profile for day 1, the third profile for day 2 and the second profile for day 3. After that the sequence will be repeated, so day 4 uses the first profile, and so on.

}

price_solar
{
profiles
For a description refer to **price_grid** above
Default: ([0.0, 24.0, 0.10])
sequence
For a description refer to **price_grid** above
Default: [1]
}

3.2 location.json

This file contains location dependent information and resides in the *locations* directory. There is a subdirectory for each location like Hannover, Rome, and so on. The file contains the following settings:

country	The ISO code of the country
type	can be ‚urban' or ‚rural'; influences the simulation of e-vehicles
latitude	
longitude	the location's coordinates
utc_offset	difference in hours from Universal Coordinated Time

The *location.json* file is accompanied by a data file that contains temperature and irradiation data for this specific location. This file is usually downloaded from the PVGIS site [1] and its name can be specified in *resLoadSIM.json* (pv_data_file_name). If no name is provided ResLoadSIM assumes a filename according to the naming convention used by PVGIS. If no file with that name can be found in the *locations* directory, ResLoadSIM will try to download the data from the PVGIS site (libcurl has to be installed).

3.3 households.json

The parameters, which influence the behaviour of the households, are stored in the file *households.json.* They are:

size_distribution[]
This vector stores the distribution of households of different sizes (in %). E.g. *size_distribution[3] = 12* means, that 12% of all households have 4 residents (vector index starts at 0).

retired_1
Percentage of single-person households where the resident is retired.
retired_2
Percentage of two person households where the residents are retired.

min_area[]
max_area[]
Minimum and maximum household area depending on the number of residents.

set_temperature_heating_day
Set room temperature in °C used for calculating the space heating demand during daytime.

set_temperature_heating_night
Set room temperature in °C used for calculating the space heating demand during the night.

set_temperature_cooling
Set room temperature in °C used for calculating the space cooling demand.
reduce_heat
Percentage of households which have a lower set temperature for the space heating during the night

heating_period_start_day
heating_period_start_month
The date at which the heaing period starts

heating_period_end_day
heating_period_end_month
The date at which the heaing period ends

min_init_laundry
max_init_laundry
Minimum and maximum amount of laundry (in kg) at simulation start.

min_delta_laundry[]
max_delta_laundry[]
The minimum and maximum amount of laundry, which gets added to the current amount of laundry every day. The actual value is a random number between the two. Both parameters are vectors, because these values depend on the number of residents.

second_fridge[]
Probability of having a second fridge (in %), depending on the number of residents.

second_tv[]
Probability of having a second TV (in %), depending on the number of residents.

third_tv[]
Probability of having a third TV (in %), depending on the number of residents.

second_computer[]
Probability of having a second computer (in %), depending on the number of residents.

min_vacuum_interval
max_vacuum_interval
Minimum and maximum interval of vacuum cleaning in days.

light_factor[]
Used to calculate the number of lamps in a household. Depends on the number of residents.

rnd_wakeup[4]
rnd_wakeup_weekend[4]
rnd_wakeup_retired[4]
Used to calculate the wakeup time of a household. The four parameters of each vector are the mean, sigma, lower and upper limit of a normal distributed random number. There are differnt vectors for different occasions (weekend, household with retired residents). All values are in seconds.

rnd_bedtime[2]
rnd_bedtime_weekend[2]
rnd_bedtime_retired[2]
Used to calculate the bedtime of a household. The two parameters of each vector are mean and sigma of a normal distributed random number. There are different vectors for different circumstances. All values are in seconds.

at_home_param[7]
These parameters are used to calculate the elements of the *at_home* matrix inside the household class. This matrix stores information about how many of the residents are at home at a given daytime. The first column of the matrix contains the time in seconds, the second column represents the number of residents. Whenever the number of residents changes, because someone leaves or returns home, this is represented as one row in the matrix. There is also one entry for the end of a day. Consider a single-person household for example, where the resident goes to work at 8 a.m. and returns home at 5 p.m. This would be stored in the *at_home* matrix this way:

at_home[0][0] = 28800; // until 8 a.m.
at_home[0][1] = 1; // the resident is at home
at_home[1][0] = 61200; // between 8 a.m. and 5 p.m.
at_home[1][1] = 0; // he is not at home
at_home[2][0] = 86400; // for the rest of the day
at_home[2][1] = 1; // he is at home again

Parts of this matrix are initialized randomly. This is where the parameter array *at_home_params* comes into play:

at_home_param[0] is the interval in seconds between wakeup and the time someone leaves home.
at_home_param[1] and *at_home_param[2]* are the lower and upper limit of a randomly chosen interval of time in which one or more residents are away from home. *at_home_param[3]* is the probability that in a two person household both residents go to work. All parameters so far refer to households with less than three residents.
at_home_param[4], *[5]* and *[6]* are for households with more than two residents and have the same meaning as *at_home_param[0]*, *[1]* and *[2]*.

energy_class[9]
This vector defines the percentages of households, which have the energy label A+, A, … , H.

rnd_heat_source[6]
A vector of probabilities used to determine what heat sources a household uses to satisfy its heat demand for space heating (SH) and domestic hot water. (DHW). The 6 currently implemented heat sources are:
oil, gas and district heating (for both SH and DHW), heat pumps with electric heating as backup, solar collectors (together with heat pumps as backup and a water storage as buffer) and finally electric heating as single heat source.

min_temperature_DHW
max_temperature_DHW
Minimum and maximum temperature [C°] of domestic hot water

min_volume_handwash
max_volume_handwash
min_volume_shower
max_volume_shower

min_volume_bath
max_volume_bath
Minimum and maximum volume of water used for washing hands, showering and bathing.

urban_car_percentage
Percentage of urban households that have a car

rural_car_percentage
Percentage of rural households that have a car

prevalence
This is a group settings. It contains the prevalence arrays for all household appliances. For example, the prevalence array for freezers could look like this:
freezer = [17.00, 40.00, 50.00, 55.00, 60.00, 65.00];
That means that 17% of all single person households, 40% of all two person households, and so on, own a freezer.

3.4 tech.json

The core of the simulation are the appliances' *simulate* functions. They model the appliances' behaviour depending on several, partially random, factors.
Each appliance can be in one of two states: ON or OFF. All the *simulate* function does, is to determine the points of time, when an appliance has to change its state. This is usually done in advance at the start of each day (daytime == 0). During the simulation the current daytime is then compared to those saved points of time. In some cases though a timer is used to keep track of an appliance‘s state. For example when a washing machine is turned on we know the cycle time and set a timer accordingly.

All appliances, which are currently turned on, contribute to the total power consumption of a household. This is taken care of at the very end of each *simulate* function. In the following a description of all application related configuration parameters is presented for all appliances. All parameters are defined in the country dependent technical configuration *tech.json*.

3.4.1 Fridges

Fridges are controlled by the following parameters:

smartgrid_enabled
Percentage of smartgrid enabled fridges

smart
Percentage of smart fridges. These are fridges which try to make use of photovoltaic overproduction.

min_temperature
The minimum temperature of a fridge

max_temperature
The maximum temperature of a fridge

delta_t_rise_factor
delta_t_rise_mean
delta_t_rise_sigma
Parameters used to calculate how much the temperature of a fridge rises within one timestep according to this formula: *factor * N(mean, sigma)*
N(mean, sigma) is a normal distributed random number.

delta_t_drop_factor
delta_t_drop_mean
delta_t_drop_sigma
These parameters are used to calculate how much the temperature of a fridge drops within one timestep.

Vc_mean[]
Vc_sigma[]
Vc_low[]
Vc_high[]
Parameters to define the size of the fridge compartment. The size is a normal distributed random number with a lower and an upper limit. All parameters are given as vectors, because they depend on the number of residents per household.

Tc
This is the nominal temperature of the fridge compartment, which is used in the formula to calculate the equivalent fridge compartment volume, which in turn is used to calculate the standard annual energy consumption. The latter is part of the formula to calculate the fridge‘s power consumption. All that formulas can be found in annex XIII of this document:
http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32010R1060&from=EN

energy_classes[7] (when **energy_classes_2021** = TRUE)
or
energy_classes[10] (when **energy_classes_2021** = FALSE)
This vector defines the percentages of fridges, which have the energy labels A to G or A+++ to G.

power_factor
The ratio of real power to apparent power.

3.4.2 Freezers

The parameters for freezers are the same as those for fridges in the previous section, without the parameters **Vc_mean**, **Vc_sigma**, **Vc_low** and **Vc_high**, and with the addition of:

Vc_per_resident
Freezer compartment size per resident.

mn_percentage
Part of the formula for the standard annual energy consumption are the parameters M and N, which come in two versions (M=0.539, N=315 and M=0.472, N=286). **mn_percentage** is interpreted as a percentage. For example: if it is set to 70, it means that for 70% of all freezers the first set of numbers for M and N is used.
The formula for the standard annual energy consumption can be found in annex XIII of this document:

http://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32010R1060&from=EN

3.4.3 Boilers

power_factor
The ratio of real power to apparent power.

3.4.4 Air conditioners

min_eff
max_eff
Minimum and maximum efficiency

kW_per_m2
Power of the air conditioner per square meter room area.

power_factor
The ratio of real power to apparent power.

3.4.5 Dishwashers

smartgrid_enabled
Percentage of smartgrid enabled fridges

smart
Percentage of smart dishwashers. These are appliances, which try to make use of photovoltaic overproduction.

energy_classes[7]
This vector defines the percentages of dishwashers, which have the energy label A to G.

hours_per_cycle
Time in hours for one cycle

place_settings[]
The dishwasher‘s capacity depending on the number of residents.

SAEc_small[2]
Two parameters used to calculate the standard annual energy consumption for small dishwashers.

SAEc_big[2]
Two parameters used to calculate the standard annual energy consumption for big dishwashers.

factor
Multiplier used in the formula for the dishwasher‘s power.

probability[2]
Two parameters used to calculate the chances that the dishwasher is used at a given day.

ignore_price
In a simulation with price control enabled this number stands for the probability (in percent) that a

dishwasher will ignore the current price for electricity.

fraction
It is assumed that most dishwashers are used mainly shortly after lunchtime, or later in the evening. The fraction parameter is interpreted as the percentage of dishwashers, which are used rather earlier than later.

timer_1_mean
timer_1_sigma
The parameters of a normal distributed random number which is used as a timer. This timer determines when a dishwasher is turned on. Timer_1 is used for dishwashers that start rather early.

timer_2_mean
timer_2_sigma
The parameters of a normal distributed random number which is used as a timer. This timer determines when a dishwasher is turned on. Timer_2 is used for dishwashers that start later in the day.

timer_3_mean
timer_3_sigma
Parameters of a normal distributed random number used as a timer for smart dishwashers, which are turned on after sunset. It is used to set the amount of time (in seconds) between sunset and the start of the dishwashers.

preview_length
This is the number of minutes a dishwasher under price control will look into the future to find the lowest price for electricity.

peak_delay
If peak shaving is enabled and the electricity provider has issued a stop signal, then smartgrid enabled dishwashers will delay their start for a number of seconds, which is defined by this parameter.

power_factor
The ratio of real power to apparent power.

3.4.6 E-Vehicles

smartgrid_enabled
Percentage of smartgrid enabled e-vehicles.

smart
Percentage of e-vehicles, which try to charge their batteries when there is a surplus production of solar electricity.

departure_delay
Time interval in seconds between household wakeup time and departure time of the vehicle.
The following settings are organized in groups named ‚model n‘. There can be a maximum of 10 models numbered from 1 to 10. Each group contains the settings of one e-car model.

The following settings are arranged in groups with the name „model 1“, „model 2“, and so on.

name
Name of the car model.

consumption_per_100km
The consumption per 100 km in kWh

battery_capacity
The battery capacity in kWh

max_charge_power_AC
max_charge_power_DC
Maximum charging power of the AC/DC charger

charging_curve
This is a vector of 21 values given in kW. Each value represents a point on the charging curve which is characteristic for this model of e-car.

3.4.7 Circulation Pumps

power_per_size
Given in kW. Used to calculate the pump‘s power:
*power = power_per_size * household_area*

power_factor
The ratio of real power to apparent power.

controlled
Percentage of controlled circulation pumps, which are turned on and off during daytime, as opposed to uncontrolled pumps, which just run permanently between morning and evening.

rnd_first_day[4]
The first day a pump gets turned on is calculated as a normal distributed random number *N(rnd_first_day[0], rnd_first_day[1])*. *rnd_first_day[2]* and *rnd_first_day[3]* are the lower and upper limit.

rnd_last_day[4]
Analogous to *rnd_first_day[4],* these parameters are used to calculate the day when a pump gets turned off again.

first_month
The first month in a year a pump gets turned on.

last_month
The month when the pump gets deactivated.

rnd_time_on[2]
The length of the time interval a controlled pump is turned ON is calculated as a random number, which is uniformly distributed between *rnd_time_on[0]* and *rnd_time_on[1]*. The numbers are in seconds.

rnd_time_off[2]
The length of the time interval a controlled pump is turned OFF is calculated as a random number, which is uniformly distributed between *rnd_time_off[0]* and *rnd_time_off[1]*. The numbers are in seconds.

time_1
Daytime in seconds when a non-controlled pump is turned OFF

time_2
Daytime in seconds when a non-controlled pump is turned ON

3.4.8 Computers

power
Power drawn by a computer in kW.

power_factor
The ratio of real power to apparent power.

duration_mean
duration_sigma
The total time a computer is turned on during the day is calculated as a normal distributed random number with these two parameters. Both are in seconds.

duration_fraction
The total runtime is split into two intervals. The length of the first interval is given as a fraction of the total runtime, therefore *duration_fraction* is a value between 0 and 1.

duration_fraction_saturday
This is the *duration_fraction* used on Saturdays.

duration_fraction_sunday
This is the *duration_fraction* used on Sundays.

time_offset[3]
time_offset_saturday[3]
time_offset_sunday[3]
Interval in seconds between the household wakeup time and the first time the computer gets booted. These values come in sets of three to make things a bit more interesting. Which one of the values is used is determined by the random number generator.

rnd[3]
rnd_saturday[3]
rnd_sunday[3]
These are the values used by the random number generator to select the previously described *time_offset*. All values are read as percentages.

time_2_mean
time_2_sigma
Daytime in seconds when the second runtime interval of a computer starts. Calculated as a normal distributed random number as usual.

3.4.9 Stoves

power[]
The power of a cooker dependent on the number of residents in the households.

power_factor
The ratio of real power to apparent power.

duration_1_percent
The total time per day a cooker is in use is split in parts. *duration_1_percent* is the probabilty that the cooker is used in the morning.

duration_2_percent
duration_2_percent_saturday
duration_2_percent_sunday
This is the probabilty that the cooker is used for preparing lunch. The values can differ depending whether it is a weekday, a Saturday or a Sunday.

time_offset
This is the time in second between the household wakeup time and the time the cooker gets activated.

rnd_duration_1[4]
rnd_duration_2[4]
rnd_duration_3[4]
The duration a cooker is turned on is calculated as a normal distributed random number. Each array contains four values: mean, sigma, lower and upper limit of this random number. *rnd_duration_1* is used in the morning, *rnd_duration_2* at lunchtime, and *rnd_duration_3* in the evening.

time_2_mean
time_2_sigma
Parameters (in seconds) for a normal distributed random number which is the daytime when the cooker is turned on for preparing lunch.

time_3_mean
time_3_sigma
Parameters (in seconds) for a normal distributed random number which is the daytime when the cooker is turned on for preparing dinner.

3.4.10 E-Heating

smartgrid_enabled
Percentage of smartgrid enabled electrical heatings

kw_per_m2
Power of the electrical heating per square meter household area in kW.

power_factor
The ratio of real power to apparent power.

3.4.11 Lights

energy_classes[7]
This vector defines the percentages of lights, which have the energy labels A to G, or A+++ to D under the old labeling scheme.

luminous_flux_mean
luminous_flux_sigma
luminous_flux_min
luminous_flux_max
The luminous flux, which is used to calculate the power consumption, is calculated as a normal distributed random number with mean, sigma, lower and upper limit as parameters.

sigma_morning
sigma_evening
Used to calculate for how long lights get turned on in the morning and in the evening.

power_factor
The ratio of real power to apparent power.

3.4.12 Tumble Dryers

smartgrid_enabled
Percentage of smartgrid enabled tumble dryers.

energy_classes[7]
This vector defines the percentages of lights, which have the energy labels A to G, or A+++ to D under the old labeling scheme.

hours_per_cycle
Time in hours for one drying cycle.

capacity[]
Capacity in kg depending on the number of residents.

ignore_price
Percentage of dryers which ignore the price of electricity when under price control.

peak_delay
If peak shaving is enabled and the electricity provider has issued a stop signal, then smartgrid enabled dryers will delay their start for a number of seconds, which is defined by this parameter.

power_factor
The ratio of real power to apparent power.

3.4.13 TVs

energy_classes[7] (when **energy_classes_2021** = TRUE)
or

energy_classes[10] (when **energy_classes_2021** = FALSE)
This vector defines the percentages of TVs, which have the energy labels A to G, or A+++ to G under the old labeling scheme.

diagonal_1
diagonal_2
diagonal_3
Size of a TV in inch for the first, second and third TV in a household.

avg_duration[]
Average time (seconds) a TV is turned on per day, depending on the number of residents in a household.
avg_duration[0] → single-person household, *avg_duration[1]* → two person household, ...

factor_mean
factor_sigma
The total time a TV is turned on per day is a normal distributed random number *N(factor_mean*avg_duration, factor_sigma*avg_duration)*.

factor_mean_we
factor_sigma_we
Same as above, but used for Saturday/Sunday.

duration_factor
duration_factor_sat
duration_factor_sun
Fraction of the total time a TV is on (weekday, Saturday and Sunday).

random[3]
random_sat[3]
random_sun[3]
These are the values used by the random number generator to determine when to turn on the TV for the first time during the day. All values are read as percentages.

delay[3]
delay_sat[3]
delay_sun[3]
When TVs are turned on in the morning, this happens *delay* seconds after the household wakeup time. Values differ depending on whether it is a weekday or Saturday/Sunday.

time_2_mean
time_2_sigma
Parameters for a normal distributed random number, which is interpreted as the daytime when a TV is turned on for the second time. Values are in seconds.

power_factor
The ratio of real power to apparent power.

3.4.14 Vacuum Cleaners

energy_classes[7] (when **energy_classes_2021** = TRUE)
or
energy_classes[10] (when **energy_classes_2021** = FALSE)
Although at the time of writing there is actually no energy efficiency labeling for vacuum cleaners in effect, we still use energy classes as a way of specifying a distribution of vacuum cleaners with different levels of energy consumption.

timer_min
timer_max
The daytime (in seconds) when to use the vacuum cleaner is calculated as a random number between *timer_min* and *timer_max*.

timer_factor
Both parameters are used to calculate for how long to turn on the vacuum cleaner (in seconds).

power_factor
The ratio of real power to apparent power.

3.4.15 Washing Machines

smartgrid_enabled
Percentage of smartgrid enabled washing machines.

smart
Percentage of washing machines, which try to turn themselves on when there is a surplus production of solar electricity.

energy_classes[7] (when **energy_classes_2021** = TRUE)
or
energy_classes[10] (when **energy_classes_2021** = FALSE)
This vector defines the percentages of washing machines, which have the energy labels A to G, or A+++ to D under the old labeling scheme.

hours_per_cycle
Time in hours for one washing cycle.

capacity[]
How much laundry fits into the washing machine (in kg), depending on the number of residents.

random_limit
This value is interpreted as a percentage. Together with other factors this parameter is used to determine whether the washing machine gets fired up at a given day.

ignore_price
In a simulation with price control enabled this number stands for the probability (in percent) that a washing machine will ignore the current price for electricity.

best_price_lookahead
The number of minutes a washing machine will look into the future to find the lowest price for electricity.

timer_mean
timer_sigma
Both parameters are in seconds and are used to set a timer for smart washing machines that had no chance to make use of a surplus production of the solarmodules. Those machines will be turned on a randomly chosen time after sunset.

peak_delay
If peak shaving is enabled and the electricity provider has issued a stop signal, then smartgrid enabled washing machines will delay their start for a number of seconds, which is defined by this parameter.

power_factor
The ratio of real power to apparent power.

3.4.16 Heat Pumps

min_eff
max_eff
The efficiency of a heat pump is a random number between *min_eff* and *max_eff*. Both are values between 0 and 1.

min_temperature
max_temperature
The hot temperature of a heat pump is randomly choosen within min and max temperature.

kW_per_m2
Power per square meter household area in kW.

power_factor
The ratio of real power to apparent power.

3.4.17 Heat Storage

liter_per_m2
The capacity of the heat storage in relation to the household area.

max_temperature
Maximum water temperature of the storage in C°.

max_heat_power
Maximum power output of the storage in kW.

3.4.18 Solarmodules and Batteries

The simulation of solar modules and attached batteries differs considerably from the simulation of

the household appliances. The power of an appliance drawn from the grid or the battery is assumed to be constant over time, while this appliance is turned on, and can be calculated before the simulation starts. The power delivered from a solar module on the other hand varies over time and has to be updated every timestep. It can be safely assumed that the power before sunrise and after sunset equals zero. Between sunrise and sunset the power output of a solar module is calculated using its nominal power and the solar irradiation values at the given location. The latter values can be obtained by using PVGIS [1].

Once the power output of the module has been calculated, it is compared to the household's current consumption. Any surplus production is fed into the grid or used to charge the battery, if available. In case the consumption is higher than the production, power is drawn from the battery. The simulation takes into account that there are losses while charging or discharging a battery, and that the battery's capacity, the charging and discharging power are limited.
Parameters related to the solarmodules are:

system_loss
Estimated system losses of the PV installation (in %)

production_ratio
The production ratio is the ratio of solar production to household consumption. If this value is zero, ResLoadSIM will set the nominal power of all solar modules to random values depending only on the household size, and then perform an ordinary simulation. For any value bigger than zero the nominal power of the solar modules can be calculated precisely as long as the household consumption is known. Therefore, when production_ratio > 0, ResLoadSIM will extend the transient time by 365 days.

min_area
max_area
The size of a solarmodule per household resident is a random number between *min_area* and *max_area* (in m^2).

min_eff
max_eff
The efficiency of a solarmodule is a random number between *min_eff* and *max_eff*. Both are values between 0 and 1.

Parameters for the batteries are:

frequency_solar
The frequency (in %) of batteries in households with a photovoltaic installation.

frequency_non_solar
The frequency (in %) of batteries in households without a photovoltaic installation.

capacity_in_days
If this parameter is bigger than zero, it is interpreted as the number of days a fully charged battery can provide energy to a household without the need of a recharge. This requires to know a household‘s average consumption per day, which in turn makes it necessary to extend the simulation‘s transient time by 365 days (which is done automatically by ResLoadSIM). If this parameter is set to zero, then the following two parameters are used to calculate the battery‘s capacity.

min_capacity_per_resident
max_capacity_per_resident
The capacity is set to a random value between *min_capacity_per_resident* and *max_capacity_per_resident* multiplied by the number of residents.
smartgrid_enabled
Percentage of smartgrid enabled batteries.

allow_grid_charge_solar
Boolean flag to choose whether solar batteries are allowed to be charged with electricity from the grid.

installation_costs
Fixed installation costs for a solar battery (in Euro)

avg_lifetime
The average lifetime of a battery in years.

min_price
max_price
The minimum and maximum retail price per kWh capacity of a battery in Euro.

min_capacity_per_resident
max_capacity_per_resident
The minimum and maximum battery capacity per resident in kWh.

min_eff_charging
max_eff_charging
The minimum and maximum efficiency while charging.

min_eff_discharging
max_eff_discharging
The minimum and maximum efficiency while discharging.

max_power_charging
The maximum power used for charging, measured as a fraction of the maximum capacity.

max_power_discharging
The maximum power while discharging, measured as a fraction of the maximum capacity.

3.5 vacation.json

This configuration file allows to define the times when households are on vacation. Household appliances are in general turned off during these times with freezers being the exception. The file contains an array of name-value pairs where the name is a date (day.month) and the values represent the percentages of households being on vacation starting with the given date and being in effect until the next following date in the file. The values are given as arrays where each member stands for a different household size. The first value in the array refers to single-person households and so on. Thus the name-value pairs define a staircase function. This is illustrated with the following example of a vaction.json file:

```
{
   "vacation_percentage":
   {
        "1.1":   [3.5, 3.5, 3.5, 3.5, 3.5, 3.5],
        "7.1":   [2.0, 2.0, 2.0, 2.0, 2.0, 2.0],
        "1.7":   [13.4, 13.4, 13.4, 13.4, 13.4, 13.4],
        "1.9":   [5.7, 5.7, 5.7, 5.7, 5.7, 5.7]
   }
}
```

In this example 3.5% of the households (all sizes) are on vacation beginning with the first of January until and including the 6th of January. Starting with the 7th of January only 2% of the households are on holidays. The 5.7% of households being on vacation starting with the first of September is valid for the rest of the year.
In the simulation a household going on vacation gets assigned a random number of holidays. This number is decreased at the beginning of each day until it eventually is down to zero again, which means the residents have returned home.
The statistical data about how many households are on holidays at a given time is usually available on a monthly basis. To avoid sudden changes in the power consumption at the transition from one month to the following one (e.g. the first of July in the above example) it is recommended to define a more fine-grained staircase function. There is a vacation.json file in example/countries/AT where the percentage value is defined for each single day of the year. This file was generated by using the monthly data and feeding it to the ‚smooth csplines‘ function of gnuplot. The gnuplot output was then converted to the json format.

3.6 public_holidays.json

In resLoadSIM public holidays are treated in the same way as Sundays. To let the simulation know when there is a holiday the user has to provide a file named 'public_holidays.json' either in the current working directory or in the respective country directory. This file has the following format:

```
{
   "2020": ["13.4", "21.5", "1.6", "11.6"],
   "2021": ["5.4", "13.5", "24.5", "3.6"],
   "2022": ["18.4", "26.5", "6.6", "16.6"],
   "2023": ["10.4", "18.5", "29.5", "8.6"],
   "2024": ["1.4", "9.5", "20.5", "30.5"],
   "default": ["1.1", "6.1", "1.5", "15.8", "26.10", "1.11", "8.12", "25.12", "26.12"]
}
```

The lines starting with the name of a year contain the public holidays of that specific year, whereas the 'default' line lists those holidays which always fall on the same date no matter what year. When a simulation is run for the year 2022 for example, then all dates from the '2022' line plus the dates from the 'default' line are interpreted as public holidays. If the simulation is run for the year 2015, then only the default holidays are considered in the above example, unless the user adds an extra line for the year 2015.

3.6 business.json

Businesses are simulated by using standard load profiles. In *business.json*, which has to be placed in the current working directory or in the country directory, one can define different classes of businesses. Each class can refer to its own standard load profile. A simple examle with just one business class could look as follows:

```
{
  {
    "description": "supermarket",
    "count": 10,
    "load_profile_name": "G3",
    "scale_factor": [1.00, 1.60],
    "power_factor": [0.80, 1.00],
    "has_fridge": TRUE
  }
}
```

description
is currently not used by resLoadSIM, but helps the user to identify different business classes. The JSON standard does not allow comments, so this is kind of a workaround.

count
The number of businesses of this class.

load_profile_name
The name of the standard load profile to be used. In the above example resLoadSIM expects a file named 'G3.csv' to be located in the directory *countries/[ISO-country-code]/standard_load_profiles*. The load profiles are saved as CSV files, which contain a timestamp and nine values for the power drawn. The nine values refer to Saturday, Sunday and weekday in winter, summer and a transition period. Examples can be found in *example/countries/AT/standard_load_profiles*.

scale_factor
The power values in the .load profile files are normalized and need to be be scaled using this factor.

power_factor
The ratio of real power to apparent power.

has_fridge
If set to TRUE then the power consumption of a business is calculated by adding the power drawn by a fridge to the power values from the standard load profile.

4 Output Files

The simulation results are stored in a set of output files, some of which can be used as input for a data plotting program like Gnuplot. The files are:

appliances.[year].[residents]

One file per year and per household size is generated. The output is triggered every year at December 31st and once when the simulation ends. Each file contains the yearly consumption of all appliances for the given year and size of households. There is one line per household, and the consumption values within a line are grouped by appliance type in alphabetical order and within each appliance type split with regard to the energy classes. That means the first values in each line refer to the air conditioners, followed by boilers, circulation pumps and so on. The number of values for each appliance corresponds to the number of energy classes of this appliance. For appliances, which have no energy classes (or for which energy classes are not implemented yet), only one value is print to the file.

battery.[year]

This one is produced only in case there are households with a photovoltaic installation including a battery. The 6 data columns are:

- the time in hours
- the mean charge of all batteries in percent
- the current power used for charging in kWh
- the current power drawn from all batteries in kWh
- the power loss while charging in kWh
- the power loss while discharging in kWh

This file is generated once per year (December 31st) and at the very end of the simulation.

consumption.[year]

This is a table that shows the total consumption of electricity in kWh during the year quoted in the file name. This is not necessarily a whole year, depending on the start and end date of the simulation. Each column stands for one household category (1 person, 2 persons, …). The rows show the minimum, average and maximum consumption together with the median and the standard deviation for all household appliances. For each appliance the number of households with such an appliance and the total number of appliances of this type are given.

costs.[year]

There are two tables stored in this output file, both showing the mean consumption and the mean costs for all household categories. The two tables refer to households with and without photovoltaic installations.

dist.[year].[residents]

For each year and household category one distribution file is produced. The distribution is generated in the following way:

The interval between the minimum consumption and the maximum consumption of the households for the given category is divided in subintervals with constant length. Each household determines to which subinterval its own consumption belongs to. The middle point of each subinterval together with the number of households that belong to that interval are written to the file.

gridbalance.[year]

The five columns are:
the time (hours)
power into grid minus power from grid (kWh)
power from grid (kWh)
power into grid (kWh)
power from grid, which is used for charging the batteries (kWh).

households.[year].[residents]

One line per household is printed. Each line consists of the following values:
Household id
Number of residents
Total consumption in kWh
Own consumption of electricity produced by solar module in kWh
Energy drawn from battery in kWh
Number of TVs
Nominal power of the solar module in kW (0 if not installed)
Production of the solarmodule in kWh
Battery capacity in kWh
Efficiency of charging
Efficiency of discharging
Costs for energy drawn from the grid
Income for energy fed into the grid

households.stats.[residents]

One line per household. Each line consists of:
Household id
Minimum daily consumption
Maximum daily consumption
Mean consumption
Median of the daily consumption
Standard deviation of the daily consumption

max.[year].[residents]

One line per household without a solar module. Each household stores the 3 highest power values (in kW) together with the times when they occured (in hours). One line consists of the following values:
Household id
Time 1
Power value (highest)
Time 2
Power value (2^{nd} highest)
Time 3
Power value (3^{rd} highest)

max_sol.[year].[residents]

One line per household. Only households with a photovoltaic installation are considered. Times are in hours and power in kW. The lines consist of:
Household id
Time 1
Household power at time 1 (highest)
Solar power at time 1

Time 2
Household power at time 2 (2nd highest)
Solar power at time 2
Time 3
Household power at time 3 (3rd highest)
Solar power at time 3
Time 4
Max. power drawn from grid at time 4 (highest)
Household power at time 4
Solar power at time 4
Time 5
Max. power drawn from grid at time 5 (2nd highest)
Household power at time 5
Solar power at time 5
Time 6
Max. power drawn from grid at time6 (3rd highest)
Household power at time 6
Solar power at time 6

power.[year].[appliance]

For each year and appliance there is one power file written to the disk. On top of that there is one power file for the households, which is named *power.[year].Total*.
The first column in each file is the time in hours. The second column is the power of all appliances of a certain type (e.g. all TVs) in all households, which are turned on at that time. In the next six columns the power is broken down in the six household categories. The file *power.[year].Solarmodule* contains an additional column showing the produced power that is used up by the household immediately. All power values are in kW, the time is given in hours.

summary.[year]

There are several sections. The first one shows the total consumption of all electrical appliances plus batteries during the year quoted in the file name. The second part refers to the photovoltaic installations. It shows the total solar electricity production, the part that is fed into the grid and the part that is used immediately. The third section shows the amount of heat generated by the solar collectors and how much is used for space heating and domestig hot water respectively. The following two sections show the amount of energy used for space heating and domestic hot water broken down by the type of heat source used.

If the powerflow simulation is used and the parameter *powerflow.output_level* is not set to 0, then additional output is generated:

trafo.[nr]

There is one file for each transformer in the grid. The values in the columns are:
the time (hours)
number of the bus with minimum voltage
minimum voltage magnitude
fraction of households in energy conservation mode
number of the bus with maximum voltage
maximum voltage magnitude
fraction of households in energy burst mode
power transmitted by transformer
maximum power

total power drawn by all households attached to this transformer
total solar production of all households attached to this transformer
household with maximum consumption
bus with maximum consumption

pfin/

A directory that contains the case data files, which are generated by ResLoadSIM and used by the powerflow solver as input.

pfout/

This directory contains the output generated by the powerflow solver.

5 Load Shifting

There are several modes of smart grid control implemented in ResLoadSIM. They are user selectable via the setting *control* in the configuration file *resLaodSIM.cfg*. The control modes are:

None: This is the trivial case. Smartgrid control is disabled.

Peak Shaving: In this mode the simulation tries to keep the maximum consumption below a user specified threshold value (peak_shaving.threshold in *resLoadSIM.json*). This threshold refers to the maximum peak value, which has been calculated during the first day of the simulation. This has an effect on all smartgrid enabled appliances, which can be fridges, freezers, e-vehicles, washing machines, tumble dryers and dishwashers. The first three of those can be turned on and off by centralized control at any time, under the condition that the temperature of fridges and freezers does not exceed a user defined limit. For the rest of the smartgrid enabled appliances the producer sends a global stop signal whenever the total load reaches the peak shaving threshold. No washing machine, tumble dryer or dishwasher is turned on while this signal is in effect, but already running ones aren't interrupted.

Profile: The producer tries to control the total consumption such that it follows a user defined profile. It influences all smartgrid enabled appliances in the same way as described in the last paragraph. The profile is a file with 2 columns of values, which are the time in hours and the total consumption of all households at any given time.

Price: This is a decentralized way of influencing the load distribution by using the electricity tariff as an incentive for the households. Only the washing machines, the dishwashers and the tumble dryers are effected by this option.

6 Grid Voltage Control

If the power flow solver is used and the parameter **powerflow.case_file_name** is not the empty string, then ResLoadSIM will use the file with the given name as the powerflow solver‘s case data file. A description of the Matpower case file format can be found here:
http://www.pserc.cornell.edu/matpower/docs/ref/matpower5.0/caseformat.html
ResLoadSIM specific extensions to the case data can be defined in a separate file which has to have the same name as the case data file with the appended suffix '.ext'. This file contains the relation between bus numbers and household numbers. It is possible to assign several households to a single bus. Each line in the file contains the following information:

The bus number
A boolean flag ('f', 'F', 't' or 'T') to specify whether it's a bus with solar households
The number of attached households
The list of household numbers

By using the PETSc‘s power flow solver ResLoadSIM is able to simulate a reaction to grid voltage levels, which raise above or drop below predefined limits. The communication between ResLoadSIM and pflow is done via files. ResLoadSIM writes the grid topology together with the households‘ consumption values to a file in MATPOWER format. Pflow reads that file, calculates the voltage levels for each node in the grid and writes the results to a file, which is then read by ResLoadSIM.
If undervoltage control is enabled, ResLoadSIM will try to reduce the consumption of affected households by sending them a ‚reduce consumption‘ signal. When voltage levels rise back to normal, this signal is revoked. In order to avoid oscillations two mechanisms have been implemented. First, the signal is sent only to a fraction of all affected households (e.g. 10%) per timestep. Secondly, before the ‚reduce consumption‘ signal is revoked, the voltage has to rise above a limit, which is higher compared to the limit which triggered the signal in the first place (hysteresis).
Overvoltage control works analogously. In this case households, which are affected by voltage levels above nominal voltage, are sent a ‚raise consumption‘ signal.

7 Space Heating and Cooling

The simulation of the space heating/cooling demand is based on the hourly calculation procedure presented in the European Standard prEN ISO 52016-1 [3]. Since ResLoadSIM is used to simulate thousands of households of which we do not have detailled information regarding the layout, some simplifications had to be applied. E.g. it is assumed that all housholds are detached houses without inner walls.

Each building is modelled as a set of building elements, which can be walls, floors, ceilings, doors or windows. Each element in turn is modelled as a set of nodes. There are 5 nodes (2 surface nodes and 3 inner nodes) for walls, ceilings and floors, and only 2 nodes (on the surface) for doors and windows. Each time step the temperatures of all nodes plus the inside air temperature are calculated by solving the equations derived from EN ISO 52017-1 [4]. There are energy balance equations for

- nodes at inner surfaces
- inner nodes
- nodes at the outer surfaces
- nodes at the surface of dividing walls (not considered in ResLoadSIM)

There is one additional equation for the energy balance at zone level. A building can have different zones with different set temperatures, like sleeping rooms or garages. In ResLoadSIM all buildings consist of only one zone (no dividing walls).

Further simplifications are made at the level of the energy balance equations. ResLoadSIM does not take into account:

- thermal bridges
- the influence of air humidity
- solar heat input

The simplified equations used in ResLoadSIM are:

1) for nodes at the inner surfaces of the elements *eli* (*pli* = 1)

$$-h_{pli;eli}\,\theta_{pli+1;eli;t}+[h_{ci;eli}+h_{ri;eli}\sum_{elk=1}^{eln}(\frac{A_{elk}}{A_{tot}})+h_{pli;eli}]\,\theta_{pli;eli;t}-h_{ci;eli}\,\theta_{\mathrm{int};air;t}-\sum_{elk=1}^{eln}(\frac{A_{elk}}{A_{tot}}h_{ri;eli}\,\theta_{pli;elk;t})=$$
$$\frac{1}{A_{tot}}[(1-f_{\mathrm{int};c})\,\phi_{\mathrm{int};t}+(1-f_{H;c})\,\phi_{H;t}]$$

2) for all inner nodes of the elements *eli* (*pli* = 2 … 4)

$$-h_{pli;eli}\,\theta_{pli+1;eli;t}+[\frac{\kappa_{pli;eli}}{\Delta t}+h_{pli-1;eli}+h_{pli;eli}]\,\theta_{pli;eli;t}-h_{pli-1;eli}\,\theta_{pli-1;eli;t}=\frac{\kappa_{pli;eli}}{\Delta t}\,\theta_{pli;eli;t-1}$$

3) for nodes at the outer surfaces (*pli* = 5)

$$(h_{ce;eli}+h_{\mathrm{re};eli}+h_{pli-1;eli})\,\theta_{pli;eli;t}-h_{pli-1;eli}\,\theta_{pli-1;eli;t}=$$
$$(h_{ce;eli}+h_{\mathrm{re};eli})\,\theta_{ext;air;t}-\phi_{sky;eli}$$

4) energy balance at zone level

$$[\frac{C_{\mathrm{int}}}{\Delta t}+\sum_{eli=1}^{eln}(A_{eli}h_{ci;eli})]\,\theta_{\mathrm{int};air;t}-\sum_{eli=1}^{eln}(A_{eli}h_{ci;eli}\,\theta_{1;eli;t})=\frac{C_{\mathrm{int}}}{\Delta t}\,\theta_{\mathrm{int};air;t-1}+f_{\mathrm{int};c}\,\phi_{\mathrm{int};t}+f_{H;c}\,\phi_{H;t}$$

$h_{pli;eli}$	thermal conductance between node *pli* and *pli*+1 of element *eli*
$h_{ci;eli}$	convective heat transfer coefficient at the inner surface of element *eli*
$h_{\mathrm{ri};eli}$	radiation heat transfer coefficient at the inner surface of element *eli*
$\theta_{pli;eli;t}$	temperature at node *pli* of element *eli* at time step *t*
$\theta_{pli+1;eli;t}$	temperature at node *pli*+1 of element *eli* at time step *t*
$\theta_{pli-1;eli;t}$	temperature at node *pli*-1 of element *eli* at time step *t*
$\theta_{\mathrm{int};air;t}$	room temperature at time step *t*
A_{elk}	surface area of element *elk*
A_{tot}	total surface area of all elements
$f_{\mathrm{int};c}$	convective fraction of internal heat gains
$f_{H;c}$	convective fraction of the heating system
$\phi_{\mathrm{int};t}$	internal heat gains (e.g. residents' body heat and the waste heat from appliances)
$\phi_{H;t}$	heat power from the heating system in time step *t*
$\phi_{sky;eli}$	radiation from element *eli* to the sky
$\kappa_{pli;eli}$	specific heat capacity of node *pli* of an opaque element *eli*
C_{int}	inner heat capacity of the zone
Δt	time step size
$\theta_{\mathrm{int};air;t}$	internal air temperature at time step *t*

The input data for the model is either calculated by ResLoadSIM itself (e.g. the areas of the elements depending on the household size, or the heat transfer coefficients depending on the building's energy class), or they are defined in configuration files. The ambient air temperature, which is used to calculate Φ_{sky}, is read from the PVGIS database [1].

Each time step the square system of linear equations (number of nodes+1 for the room temperature)

is solved using standard LU decomposition. The system matrix does not change over time and therefore needs to be factorised only once. ResLoadSIM resorts to the GNU Scientific Library for solving the linear system.

Once the equations are solved and the node temperatures are known, an operative temperature Θ_{op} can be calculated. Θ_{op} is the arithmetic mean of the room temperature and the mean radiation temperature of the zone, which can be calculated from the node temperatures at the inner surfaces of the building elements. The output of the whole procedure so far is a temperature, but what we are interested in is the heat power Φ_H needed to achieve a predetermined set temperature Θ_{set}. The heat power is actually an input quantity in the system of equations. Following the EN ISO 52016-1 [3] the heat demand is then calculated in a stepwise manner:

Step 1
Assume that the current heat power is 0 and calculate the operative temperature $\Theta_{op;0}$.
If $\Theta_{op;0} >= \Theta_{set}$ (set temperature) then the heat demand Φ_H is obviously 0 and we are done, otherwise proceed to step 2.

Step 2
Calculate the operative temperature once more, but this time use the maximum available heat power $\Phi_{H;max}$, which the heating system can deliver, as input for the equations solver. Let the resulting temperature be $\Theta_{op;max}$.

Step 3
If $\Theta_{set} > \Theta_{op;max}$ then the available heat power is not enough to reach the set temperature in the current time step. In this case the heat demand $\Phi_H = \Phi_{H;max}$ and we are done, otherwise proceed to step 4.

Step 4
Θ_{set} is somewhere between $\Theta_{op;0}$ and $\Theta_{op;max}$. The heat demand is then calculated as follows:

$$\Phi_H = \Phi_{H;max} \frac{\Theta_{set} - \Theta_{op;0}}{\Theta_{op;max} - \Theta_{op;0}}$$

Finally update the node temperatures by solving the equations once more with using Φ_H as input.

The households calculate their heat demand for space heating and propagate the result to the heat sources. Currently five different configurations of heat sources are supported:

1. Oil heating
2. Gas heating
3. District heating
4. Heat pumps together with electrical heating and boilers
5. Solar thermal collectors together with heat pumps

The first three configurations power space heating as well as water heating. The fourth combination of heat sources lets the heat pumps take care of space heating and uses electrical heating only as a last resort when the heat pumps cannot deliver enough heat power. The boilers are used for the preparation of hot water. The last configuration is unique in the sense that it uses a water storage as a buffer. The storage has been implemented in a similar way to the batteries used in connection with the solar modules. The model used for the solar collectors is based on the European Standard prEN 15316-4-3 [5]. The main input for the solar collector model is the solar irradiation data, which is provided by the PVGIS database [1].

8 Variable Load

Some appliances have a power consumption varying over time. This behaviour can be simulated in ResLoadSIM by setting the *variable_load* flag in *resLoadSIM.json* to TRUE. Currently this affects the washing machines only. When the simulation of varying loads is activated, ResLoadSIM needs to read a file named *varload.json*, which contains a set of time-dependent power values for each energy efficiency class of an appliance with varying load. An example entry for washing machines in *varload.json* might look as follows:

```
"washing_machine“:
{
   "A+++": [[15, 0.200], [45, 0.120], [10, 0.250]],
   "A++": [[15, 0.220], [55, 0.125], [10, 0.273]],
   ...
}
```

There is a line of paired values for each energy efficiency class. The first value of each pair is to be interpreted as the number of minutes an appliance is in a certain mode. The second value is the amount of power in kW drawn from the grid while being in this mode.

9 E-Vehicles

As of version 3.7.0 ResLoadSIM uses an improved model for simulating the charging of electric vehicles based on the work of Fischer et al.[2]. The factors taken into account by the model are:

- the household's place of residence (urban or rural)
- the household type (residents are retired, working full- or part-time, students)
- the weekday
- the place of charging (at home, at the shop, ...)
- the energy consumption of the car in kWh/100km
- the battery's capacity and self-discharging rate
- the charging power of the internal charger
- user dependent charging behaviour
- availabilty of photovoltaic electricity in a household
- air temperature

Unlike all other appliances in a household, which can be only in one of two states, e.g. 'on' or 'off', the electric vehicles can be in the state 'idle', 'driving' and 'charging'. On top of that smartgrid enabled cars can also be 'forced idle' or 'forced charging' when grid voltage control is enabled. Depending on the factors in the above list a car's state can be updated at every timestep of the simulation.

A car in the state 'charging' contributes to a household's current electricity consumption. This is done in a time-dependent manner such that the actual battery load curves are reproduced correctly in the simulation.

10 Battery Charging

Using batteries for buffering a household‘s surplus solar production is an important part of an energy managment system. ResLoadSIM is able to simulate different battery charging strategies, which can be selected using the setting *battery→strategy* in *resLoadSIM.json*. The following strategies are available:

0: The batteries get charged whenever the solar production is higher than the houshold‘s current consumption. This method focusses on the optimization of the self-consumption. It does not take into account, that the batteries, especially when undersized and on a sunny day, can be fully charged at noon, which can lead to over-voltages when the surplus production is fed into the grid.

2 (strategy 1 is deprecated): This strategy tries to mitigate the problem of over-voltage by trying to make a more intelligent use of the battery buffer. The idea is to make an estimate of the amount of energy that is going to be fed into the grid every day. The battery charging occurs only after that amount of energy has been actually fed into the grid, or the current solar production is beyond a given threshold of the peak production (see *feed_in_limit* in *resLoadSIM.json*). This of course comes at the cost of a lower self-consumption, and it depends on the quality of the necessary forecast for the solar production and the electricity consumption. The different forecast methods are listed below.

3: This is a mixture of strategies 0 and 2 which employs a precharging of batteries in the morning. Batteries with a charge below a given threshold (e.g. 10% of the capacity, see *battery→precharge_threshold* in *resLoadSIM.json*) get charged according to strategy 0. Above that threshold switch to strategy 2.

The above mentioned forecast methods for the solar production can be selected with the setting *battery→production_forecast_method* in *resLoadSIM.json*. Currently the consumption of the previous day is used as a naive forecast of the current day consumption.

0: No forecast. This makes sense only in connection with *battery→strategy* = 0.

1: Use the actual solar radiation data from PVGIS [1] as a forecast.

2: Use the solar production of the previous day as a forecast.

3: Use a GEFS reforecast file (*pv_forecast_file_name* in *resLoadSIM.json*).

In all cases the extimate for the amount of energy, that has to be fed into the grid is calculated as follows: feed-to-grid = production forecast – consumption forecast – (battery capacity – battery charge).

When the flag *battery→shared* in *resLoadSIM.json* is set to TRUE, the simulation tries to store surplus solar production, that can not be buffered in an already full battery, in another houshold‘s battery.

References

[1] Šúri, M.; Huld, T.A.; Dunlop, E.D. PVGIS: a web-based solar radiation database for the calculation of PV potential in Europe. *International Journal of Sustainable Energy* 2005, 24, 2, 55-67.
https://ec.europa.eu/jrc/en/pvgis

[2] Fischer D.; Harbrecht A.; Surmann A.; McKenna R.: Electric vehicles' impacts on residential electric local profiles - A stochastic modelling approach considering socio-economic, behavioural and spatial factors. *Applied Energy* 2018, Vol. 233-234, 644-658.

[3] prEN ISO 52016-1:2015-02, Energy performance of buildings - Calculation of the energy needs for heating and cooling load in a building or building zone - Part 1: Calculation procedures (ISO/DIS 52016-1:2015)

[4] prEN ISO 52017-1:2015-01, Energy performance of buildings - Calculation of the dynamic thermal balance in a building or building zone - Part 1: Generic calculation procedure (ISO/ DIS 52017-1:2015)

[5] prEN 15316-4-3:2014-10, Heating systems and water based cooling systems in buildings - Method for calculation of system energy requirements and system efficiencies - Part 4-3: Heat generation systems, thermal solar and photovoltaic systems