solarhouse

Release 0.0.4

Sep 13, 2022

Contents:

CHAPTER 1

About Solarhouse project

This projects allows you to calculate how many solar energy you can collect on faces of your house and it changes heating season.

For make it you need to load mesh file (.stl or .obj) which represents form of your house and specify some parameters of the house. After that just start calculation and get plots of temperatures of elements inside house.

For work with faces of mesh of house used library PyMesh

To calculate solar power on each face of house with different tilt and azimuth in py-solarhouse uses PVLIB This library makes it possible to take the weather into account when calculating power.

All thermal processes in the house calculated by models. These models are described here: Thermal theory

Substituting different parameters of the house, you can carry out the calculation for each configuration and choose the best combination of parameters to save energy for heating.

CHAPTER 2

Installation

Solarhouse requires python3 and python3-pip. Also project requires follows packages:

1. numpy;
2. scipy;
3. trimesh;
4. pvlib;
5. pandas;
6. matplotlib;
7. mpld3;
8. shapely;
9. jinja2;
10. netCDF4;
11. siphon;
12. tables;

All of these packages are in file requirements.txt

2.1 Installation from pypi:

All requirements will be installed automatically

```
$ pip install solarhouse
```

2.2 Installation from github

Requires python3 and python3-pip

```
$ git clone https://github.com/yaricp/py-solarhouse.git
$ cd py-solarhouse
$./install.sh
```

then you start ./install.sh

CHAPTER 3

Quick start

After installation of package you can use it in you code.

Firstly you need to create mesh file which represent shape of house.

It can be create in Free SketchUp

Also it can be create on any 3D editors which can formed files .obj and .stl

After that put this mesh file to .files/ folder.

For example:

file main.py:

```
import os

import uuid

import settings
from solarhouse.building import Building
from solarhouse.calculation import Calculation
import solarhouse.export as export


def main():
    calc = Calculation(
        tz=settings.TZ,
        geo=settings.GEO,
        building=Building(
            mesh_file=settings.PATH_FILE_OBJECT,
            geo=settings.GEO,
            wall_material=settings.WALL_MATERIAL,
            wall_thickness=settings.WALL_THICKNESS,
            start_temp_in=settings.TEMPERATURE_START,
            power_heat_inside=settings.POWER_HEAT_INSIDE,
            efficiency=settings.EFF,
```

(continues on next page)

(continued from previous page)

```
            heat_accumulator={
                'volume': 0.032,
                'material': 'water',
            },
            windows={
                'area': 0.3,
                'therm_r': 5.0,
            },
            floor={
                'area': 1.0,
                'material': 'adobe',
                'thickness': 0.2,
                't_out': 4.0,
            },
        ),
                    )
    data_frame = calc.compute(date=22, month=12, year=2019, with_weather=False)
    calc_id = str(uuid.uuid4())
    output_dir = os.path.join(settings.PATH_OUTPUT, calc_id)
    os.makedirs(output_dir, exist_ok=True)
    csv_file_path = export.as_file(data_frame, 'csv', output_dir)
    export.as_html(data_frame, output_dir)

if __name__ == "__main__":
    main()
```

file settings.py:

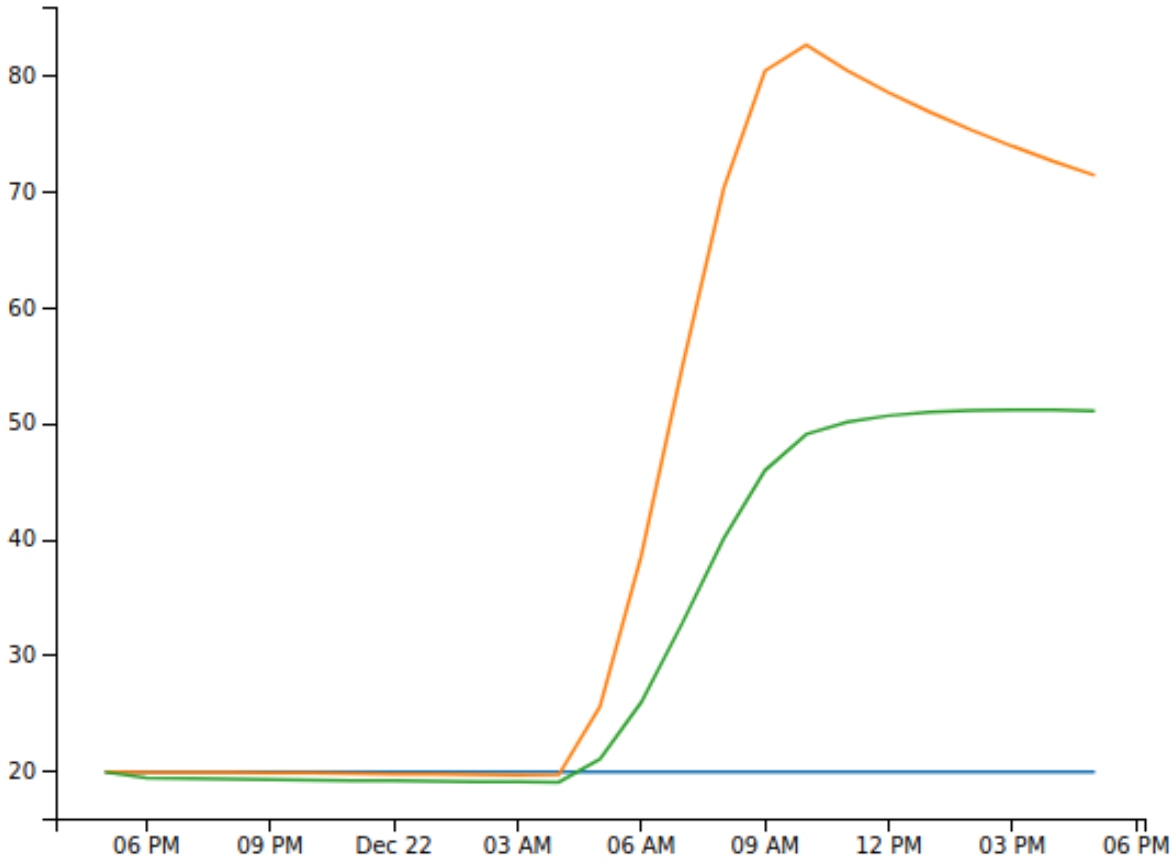
```
import os
import pathlib


_this_dir = pathlib.Path(__file__).parent.absolute()

PATH_FILE_OBJECT = os.path.join(_this_dir, 'files/cube.obj')
TIME_TICK = 1    #1 hours
WALL_THICKNESS = 0.3
TEMPERATURE_START = 20  #celcium
POWER_HEAT_INSIDE = 0   #kWtt
MASS_INSIDE = 500   #kg
PATH_FILE_TEMPERATURE_OUTSIDE_FILE = os.path.join(_this_dir, 'files/temp_table.csv')
PATH_EXPORT_THERMO_RESULT_FILE = os.path.join(_this_dir, 'files/results.csv')
SPACE_POWER_ON_METER = 1000
WALL_MATERIAL = 'adobe'
EFF = 75        #in percents
EFF_ANG = 85.0
GEO = {
    'latitude': 54.841426,
    'longitude': 83.264479,
}
TZ = 'Asia/Novosibirsk'
COUNT_FACES_FOR_PARALLEL_CALC = 1000
PATH_OUTPUT = os.path.join(_this_dir, 'output')
```

All parameters of a house (mesh, thickness of wall, material of walls and etc.) sets in file settings.py

After that you can start calculation:

```
$python3 main.py
```

As result you get two files in folder with output/<calc_id> : data.csv and plot.html like on pictures:



CHAPTER 4

Thermal theory and Modeling.

Theory based on heat equation: wiki

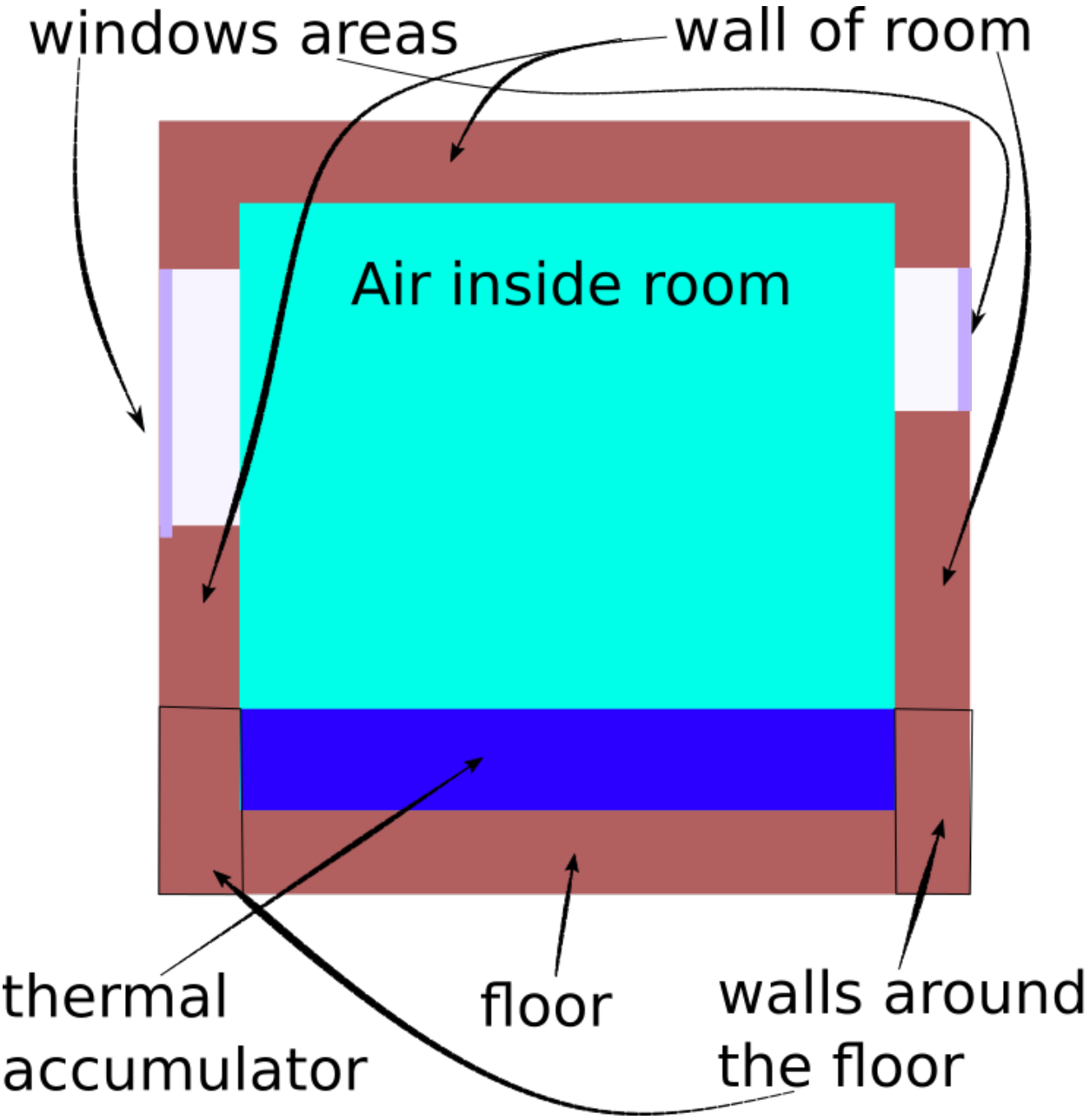
All procces heat spreading is described by the formula:

$$du/dt = \alpha(d^2u/dx^2 + d^2u/dy^2 + d^2u/dz^2)$$

Calculations using this formula are very complex and time consuming.

This why needs to simplify calculations. For this we can create simple thermal model of house.

Simple model of a house is a thermal shell with a massive object inside like on picture:

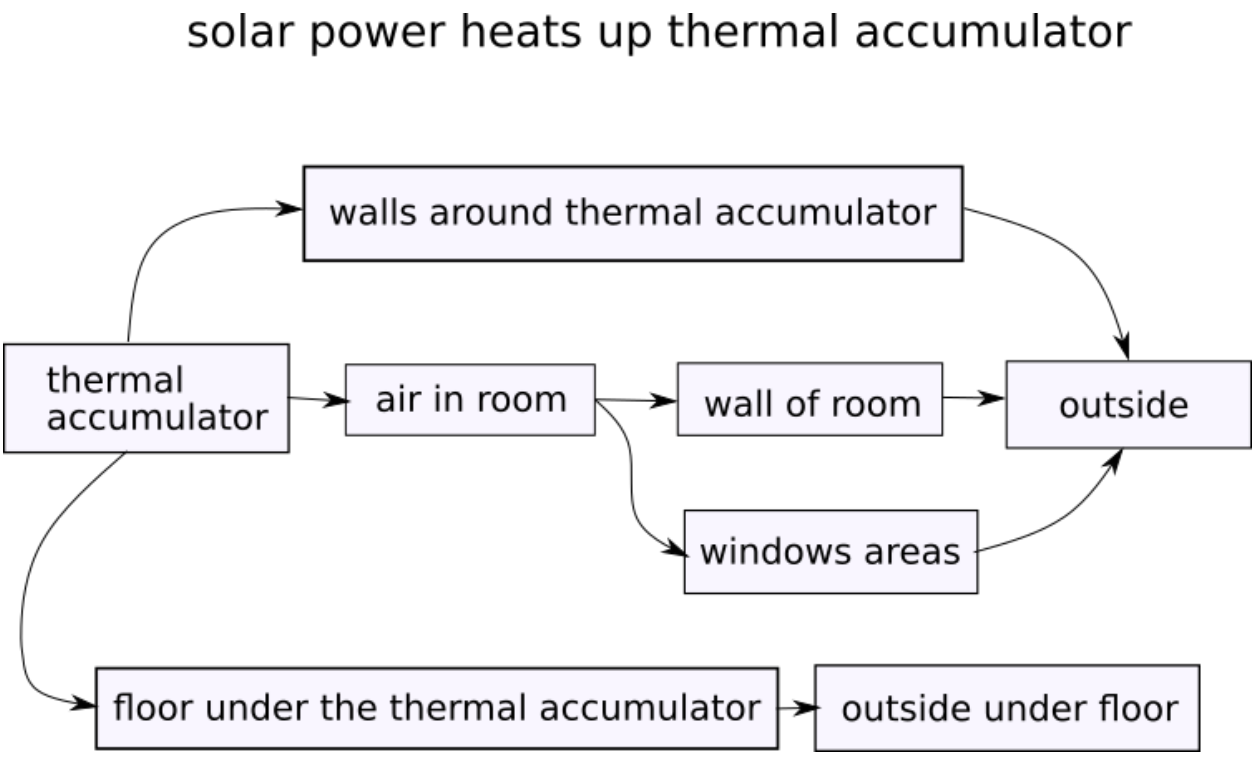


All elements of this model we can represent as a thermo point or a set of thermal layers. For each elements where temperature distribution by element volume is not important can be considered a thermal point. For example inside water thermal accumulator temperature distribution by element volume does very quickly. For elements which situated in thermal shell of model of house needs to take into account the temperature distribution of the layers from the inside to outside. Each of layers can be considered a thermal point.

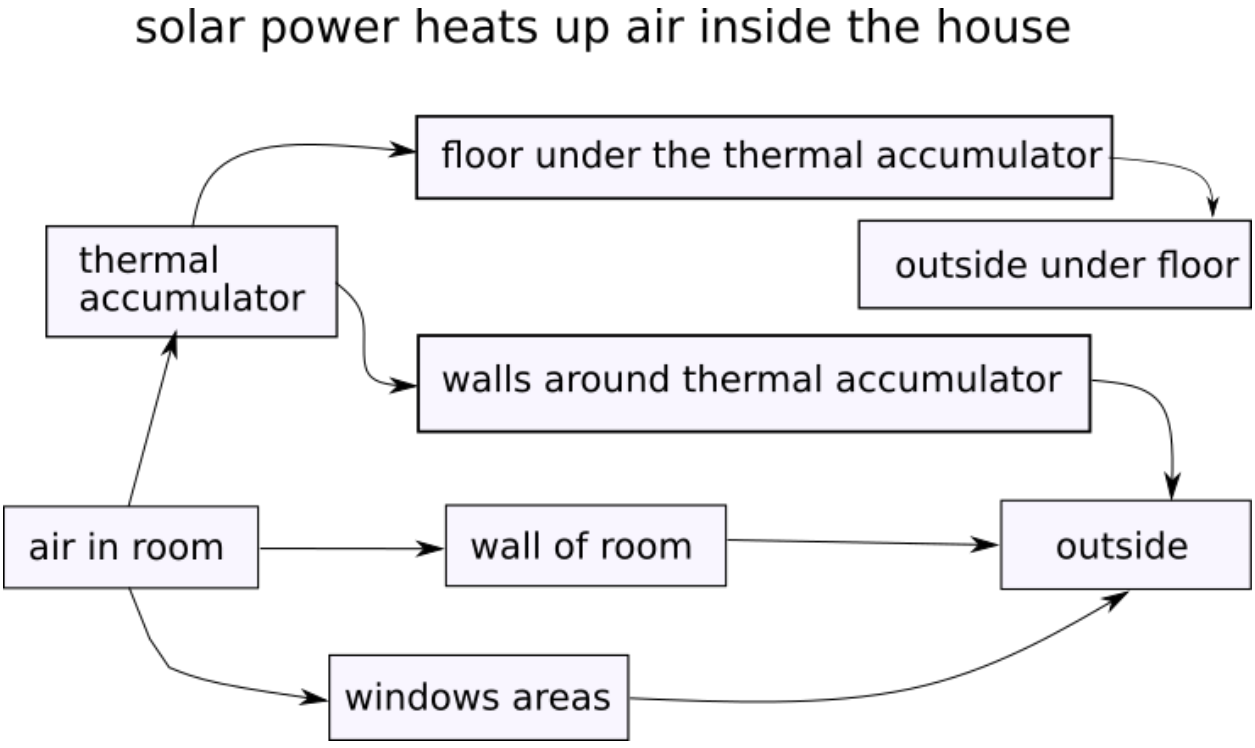
For calculate all model needs to link all elements to scheme.

Depends of type of solar collectors the begin point of heat spreading can be thermal accumulator or air inside house.

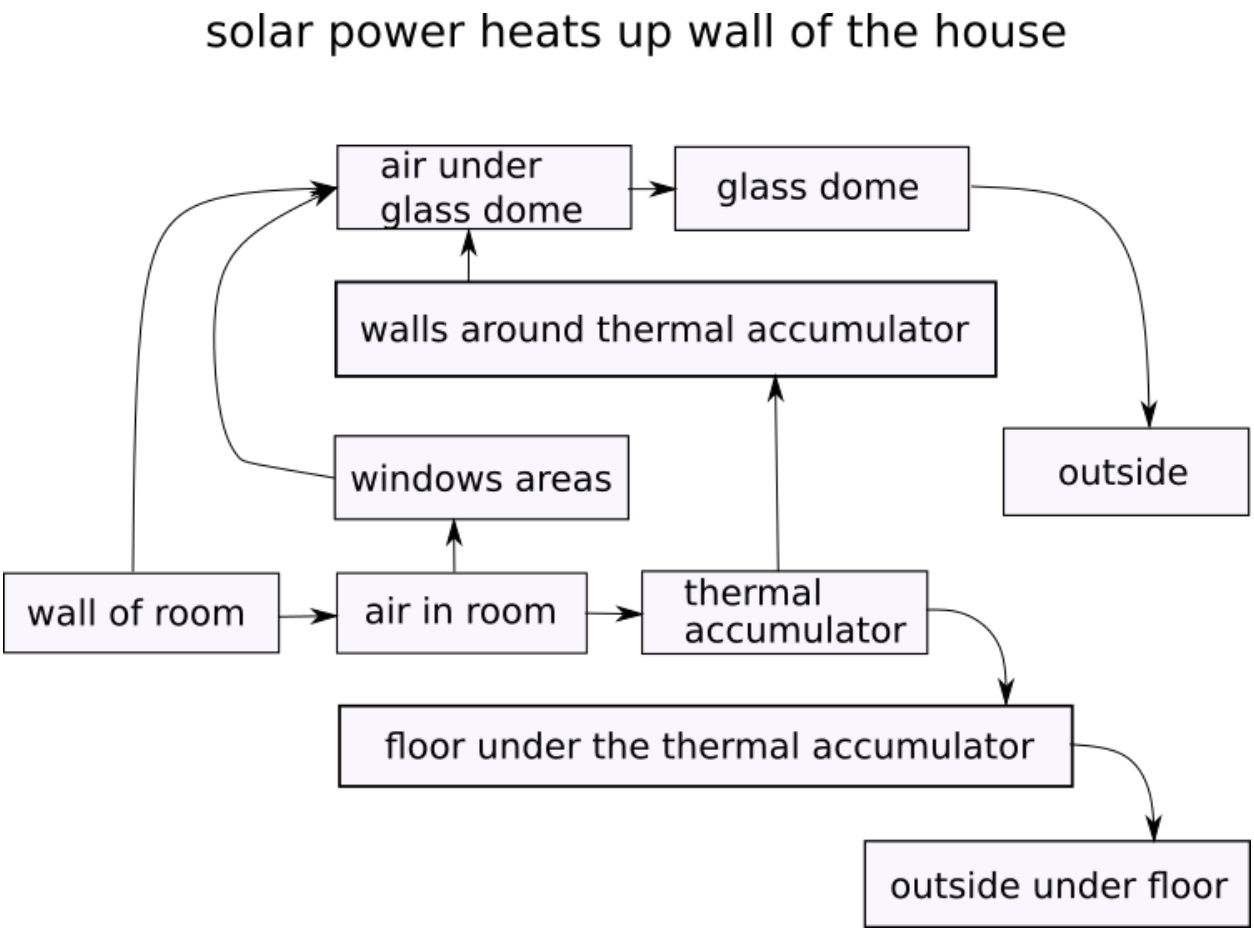
Scheme of heat spreading from thermal accumulator:



Scheme of heat spreading from air inside:



Scheme of heat spreading from walls of house:



Note: This model works best for regular convex house shapes. If the shape is not regular and has many protruding parts, then the model does not work well.

4.1 Calculation of heat spreading from point to point

The calculation is performed over a very small period of time. If the calculation is done for a layered element, then the layers are also very small.

Heat balance of each thermal point can be described by the formula:

$$Qinside = cm(t2 - t1)$$

$$Qlost = \alpha * A(Tin - Tout)$$

$$Qenter = Qinside + Qlost = cm(t2 - t1) + \alpha * A(Tin - Tout)$$

c - heat capacity of material of point;

m - mass of thermal point (or of one layer);

t2 - temperature in finish of dt;

t1 - temperature in begin of dt;

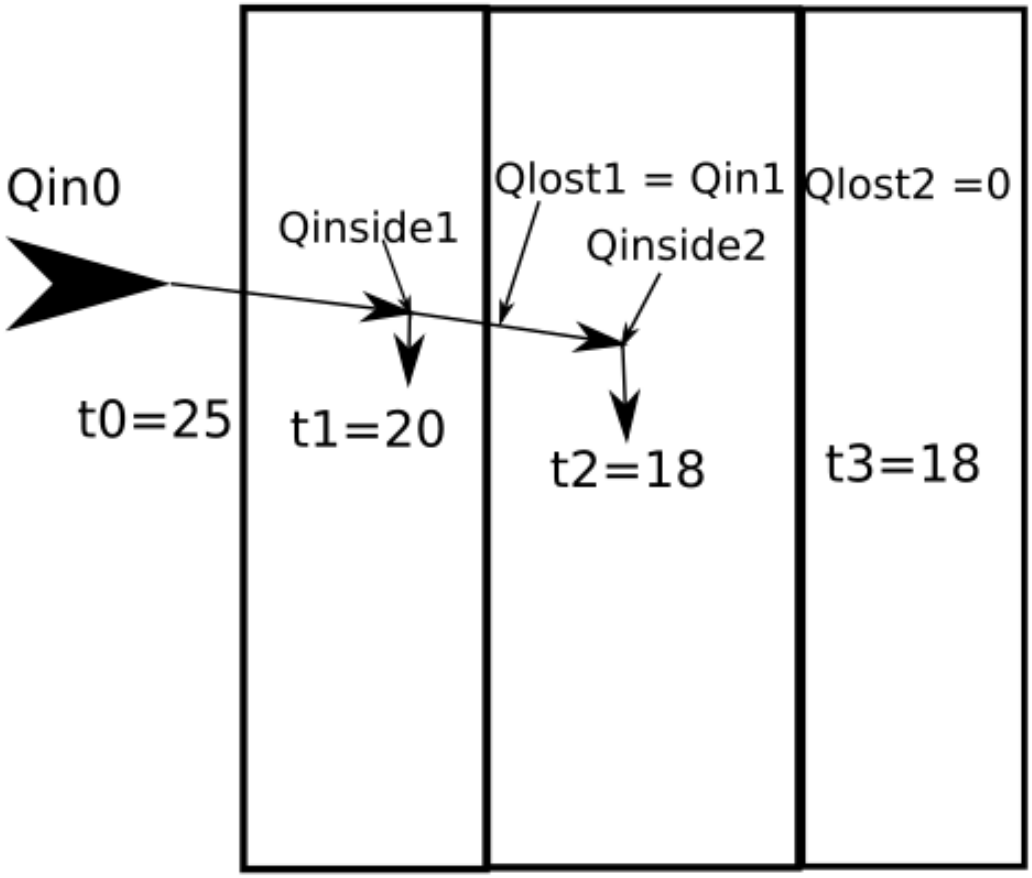
alpha - thermal diffusivity of material of point;

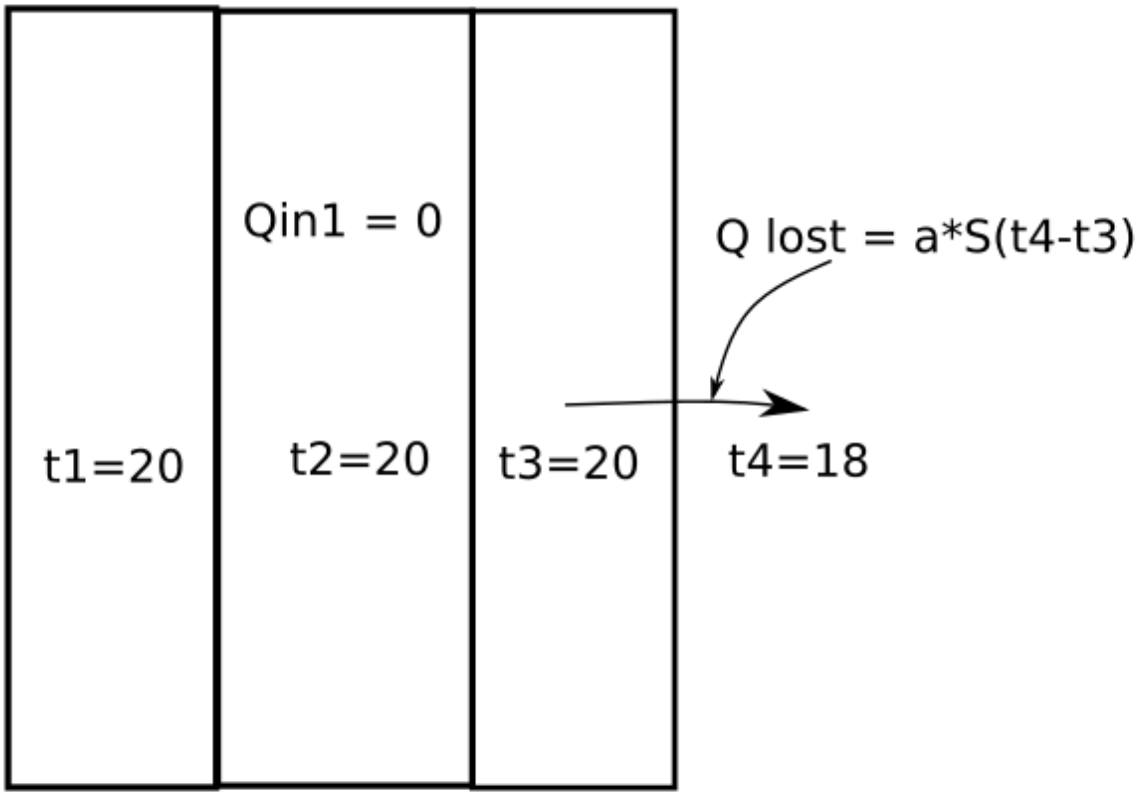
A - area of next element;

Tin - temperature of current point on begin calculation;

Tout - temperature of next point on begin calculation;

On the follows pictures you can see process for some layers:





As you can see on picture heat which lost on first layer is a enter heat of second layer.

Note: Calculation on each small period of time made on all thermal points and all layers from first element to finish element. Process running by scheme from point by point on each period of time.

Now in package we use only three scheme of elements.

Note: Last element of each scheme is a temperature of outside. There are several last elements. For example temperature outside can be not equal temperature under floor of house.

And the last assumption that we make in the calculations is that the area of layers in layered elements does not change from inside to outside.

Such an assumption is justified with a thin wall relative to the linear dimensions of the house.

If you set thick walls with small size house you get wrong result of thermal calculation.

As result if has all described above assumption you can get approximate result of temperature inside the a house.

From this you can estimate how much the heating season will be reduced.

CHAPTER 5

Calculation Class

class solarhouse.calculation.**Calculation**(*tz: str*, *geo: dict*, *building: solarhouse.building.Building*)

Class implements methods for calculate of the solar power what you can take on faces of the building. As a result you can get html page with graphics. Alternatively, you can export data in file CSV or JSON.

compute(*date: datetime.datetime = None*, *month: datetime.datetime = None*, *year: datetime.datetime = None*, *period: tuple = None*, *with_weather: bool = True*) → None

proxy method for prepare period and calculations.

start_calculation(*start: pandas._libs.tslibs.timestamps.Timestamp*, *end: pandas._libs.tslibs.timestamps.Timestamp*, *with_weather: bool = True*) → None

Start calculations.

5.1 Building Class

class solarhouse.building.**Building**(*mesh_file: str*, *geo: dict*, *wall_material: str = 'adobe'*, *wall_thickness: float = 0.3*, *start_temp_in: float = 20*, *power_heat_inside: float = 0*, *efficiency: float = 60*, *cover_material: str = None*, *heat_accumulator: dict = {'material': 'water', 'volume': 0.02}*, ***kwargs*)

Class implements methods for work with buildings for which calculate sun energy. Example: create building and test some it's parameters.

```
>>> text = 'o Cube\n'
>>> text += 'v 1.000000 1.000000 -1.000000\n'
>>> text += 'v 1.000000 0.000000 -1.000000\n'
>>> text += 'v 1.000000 1.000000 0.000000\n'
>>> text += 'v 1.000000 0.000000 0.000000\n'
>>> text += 'v 0.000000 1.000000 -1.000000\n'
>>> text += 'v 0.000000 0.000000 -1.000000\n'
>>> text += 'v 0.000000 1.000000 0.000000\n'
```

(continues on next page)

(continued from previous page)

```
>>> text += 'v 0.000000 0.000000 0.000000\n'
>>> text += 's off\n'
>>> text += 'f 1/1/1 5/2/1 7/3/1 3/4/1\n'
>>> text += 'f 4/5/2 3/6/2 7/7/2 8/8/2\n'
>>> text += 'f 8/8/3 7/7/3 5/9/3 6/10/3\n'
>>> text += 'f 6/10/4 2/11/4 4/12/4 8/13/4\n'
>>> text += 'f 2/14/5 1/15/5 3/16/5 4/17/5\n'
>>> text += 'f 6/18/6 5/19/6 1/20/6 2/11/6\n'
>>> with open('test_file.obj','a') as file:            file.write(text)
418
>>> geo = {'latitude': 54.841426, 'longitude': 83.264479}
>>> vertices = [[0,0,0],[1,0,0],[1,1,0],[0,1,0],[0,1,1],[0,0,1],[1,0,1],    [1,1,
↪1]]
>>> material = {'birch': {        'density': 700.0,        'transcalency': 0.15, ␣
↪      'heat_capacity': 1250.0}}
>>> b = Building(mesh_file='test_file.obj',        geo=geo,        wall_
↪thickness=0.3,        wall_material='birch',        properties_
↪materials=material)
>>> import os
>>> os.remove('test_file.obj')
>>> b.wall_thickness
0.3
>>> b.mesh.area
6.0
>>> round(b.mesh_inside.volume, 3)
0.064
>>> b.mesh.center_mass
array([0. , 0. , 0.5])
>>> b.floor_area_outside
1.0
>>> round(b.floor_area_inside, 3)
0.16
>>> b.windows['area']
0.0
>>> b.windows['area'] = 0.5
>>> b.walls_area_outside
4.5
>>> round(b.walls_area_inside, 3)
0.3
>>> b.heat_accumulator['mass'] = 1
>>> b.heat_accumulator['density'] = 1000
>>> b.get_perimeter_floor('inside')
1.6
>>> round(b.area_mass_walls_inside, 2)
0.2
>>> round(b.volume_air_inside, 3)
0.044
>>> b.get_perimeter_floor('outside')
4.0
>>> round(b.area_mass_walls_outside, 3)
1.7
>>> import datetime, pytz
>>> date = datetime.datetime(day=22, month=6, year=2020)
```

area_mass_walls_inside
: Calculates area of the walls around the heat accumulator inside the house.

area_mass_walls_outside

Calculates area of the walls around the heat accumulator outside the house.

calc_reflect_power(*power: float*, *sun_ang: float*, *cover_material: str = 'polycarbonat'*) → float
Calculates power of reflection based on : https://majetok.blogspot.ru/2014/05/vid-na-teplicu.html. Returns: float of power of the reflection of material.

calc_sun_power_on_faces() → None
Calculates the power of sun on all faces of the building.

Returns self changed self.power_data, self.power_data_by_days

floor_area_inside
Calculates area floor inside the house

floor_area_outside
Calculates area floor outside the house

floor_thickness
Get floor thickness

get_efficient_angle(*reflect_material: dict = None*) → float
Get angle for material.

get_perimeter_floor(*where: str*) → float
Calculate perimeter of floor

Parameters **where** – 'inside' or 'outside'

Returns float value of perimeter

get_prop(*material: str*, *prop: str*) → float
Retrieve a value of property for some materials.

Parameters

- **material** – string of name material
- **prop** – string of name property

Returns float of value property

get_pv_power_face(*face_tilt: float*, *face_azimuth: float*, *face_area: float*) → float
Get Irradiation from PVLIB.

Parameters

- **face_tilt** – angle between normal of face and horizontal plane
- **face_azimuth** – angle between normal of face and north direction
- **face_area** – float value of area of face

Returns pandas DataFrame with sun power of current period.

heat_accumulator_volume
Get volume of heat accumulator

mesh_inside
Get mesh of inside walls and floor of the house

projection_on_flat(*vector: tuple*) → tuple
get vector what is projection vector on the flat plane.

volume_air_inside
Calculates volume of the air inside the house

walls_area_inside
Calculates area of walls inside the house

walls_area_outside
Calculates area of walls outside the house

5.1.1 ThermalProcess Class

class solarhouse.thermal_process.**ThermalProcess**(*t_start: float, building: solarhouse.building.Building, variant: str = 'heat_to_mass', for_plots: list = ['mass']*)

Class implements all calculations of thermal processes in a house. There are three main models of a house: 1. All solar power comes into massive body (water tank, concrete

plate, etc.) in the house with respect to efficient coefficient of water solar collector.

2. All solar power heats up air inside the house with respect efficient coefficient of air solar collector.
3. All solar power heats up walls through a glass dome.

run_process() → dict
Start main calculation process. In the end of process it show a plots of temperatures

Returns dict data of elements in house for plots.

ThermalModel Class

class solarhouse.thermal_model.**ThermalModel**(*name*, ***kwargs*)

Class implements process of calculation of some model of thermal object which contains several thermal elements. As a result you can take plots of temperatures of some thermal elements.

make_init_conditions() → None
Initialize conditions in elements.

show_schema()
Shows schema of chain.

start(*count: int*, *dt: int*, *power: float*, *t_out: float*) → dict

Parameters

- **count** – count of calculation
- **dt** – time for calculation (seconds)
- **power** – input power in first thermal element (Watt)
- **t_out** – temperature of last element

Returns dict of data of temperatures of elements.

ThermalElement Class

class solarhouse.thermal_element.**ThermalElement**(*name*, *temp0=None*, *density=None*, *heat_capacity=None*, *volume=None*, ***kwargs*)

Implements thermal element for thermal computation. Represents a point with heat capacity. Change of tem-

perature of this point depends on sum of input and output energy and heat capacity (output energy is negative). Several elements can be connected into a chain of elements. Output energy depends on temperature of current element, temperature of next element in chain, and thermal resistance between each other. An element can have several elements of output enegry. Second and further elements must have area of input face (square meters) and input coefficient of transcalency on input face.

Also element may be represented as a wall with a variable area and with variable thermal resistance on each increment of thickness, dx. Computation is implemented in single dimension, dx (meters). All computations are performed for increments of time, dt. Example: compute temperature of 1 cubic meter of water in 1 hour with

1 kW of power applied:

```
>>> e = ThermalElement(        name='cube_water',        temp0=0,
↪density=997,        heat_capacity=4180,        volume=1    )
>>> e.count_layers
1
>>> e.compute(q_enter=1000, dt=3600)
>>> round(e.temp, 3)
0.864
>>>
Example: calculate temperature of inside face of wall of birch
with dx = 0.01 m and external power of 1 kW.
Result of test calculated manually.
>>> e = ThermalElement(        name='birch_wall',        temp0=20.0,
↪density=700.0,        heat_capacity=1250.0,        dx=0.01,        thickness=0.
↪20,        kappa=0.15,        area_inside=1.0,        area_outside=1.1    )
>>> e.count_layers
20
>>> e.dTx_list
[20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.
↪0, 20.0, 20.0, 20.0, 20.0, 20.0, 20.0]
>>> e.get_loss_dx(0)
0.0
>>> e.compute(q_enter=1000, dt=1)
>>> round(e.dTx_list[0], 3)
20.109
>>> round(e.dTx_list[1], 3)
20.0
>>> round(e.get_loss_dx(0),3)
1.714
>>> e.compute(q_enter=1000, dt=1)
>>> round(e.dTx_list[0], 3)
20.218
>>> round(e.dTx_list[1], 4)
20.0002
>>>
Example element which implementing  thin layer between two areas
>>> e = ThermalElement(        name='glass',        temp0=20.0,        area_
↪inside=1.0,        input_alpha=23,    )
>>> e.calc_loss_input_q(25.0)
115.0
```

calc_loss_input_q(*t_in: float*) → float
Calculates loss energy between current and previous elements

calc_temp(*q_enter: float*, *q_loss: float*, *iterator: int*, *dt: float*) → None
Calculates the dT on dt of current point (dx) of element. If element represent as a point then calculates. Tdx = Tdx0 + (q_enter - q_loss)/cmdx

Parameters

- **q_enter** – enter power from previouse element or source of power
- **q_loss** – total power loss from current point dx
- **iterator** – number of current dx
- **dt** – range of time for calculate

Returns Nothing returns but change temperature in list of temperatures by dx in the current point

compute(*q_enter: float*, *dt: float*) → None

Start of calculate temperature of element if it represent as a point or calculate of all temperatures by dx if element has the dx parameter

Parameters

- **q_enter** – input power
- **dt** – range of time

Returns change self.temp parameter in the end of calculation

get_loss_dx(*iterator*)

Defines loss energy from current element on dx or from all element if it represent in calculation as a point. q_loss = alpha*area_branch*(T_current - T_branch)

Parameters **iterator** – number of dx, 0 if element as a point

Returns Float value of all loss power

init_conditions(*val*)

Reduction to initial conditions

5.2 Export

solarhouse.export.**as_file**(*pd_data: pandas.core.frame.DataFrame*, *type_file: str = 'csv'*, *path: str = 'output'*) → None

Export results to file.

solarhouse.export.**as_html**(*pd_data: pandas.core.frame.DataFrame*, *output_file_dir: str*) → None

Create HTML page with graphics.

5.3 Helpers

solarhouse.helpers.**prepare_period**(*tz*, *date: datetime.datetime = None*, *month: datetime.datetime = None*, *year: datetime.datetime = None*, *period: tuple = None*) → tuple

Prepare period to retrieve data of weather and calculate sun power. :param tz: - time zone of geoposition of building :param date: :param month: :param year: :param period: :return: tuple (start , end) - begin and end of period.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

T

V

W