Projektdokumentation Softwareprojekt

„**DistrictHeatSim**“

SMWK-NEUES TG70

Jonas Pfeiffer

# Einleitung

Inhalt

[1. Einleitung 1](#_Toc157763167)

[2. Zielstellung 2](#_Toc157763168)

[3. Installation und Setup 2](#_Toc157763169)

[4. Benutzerhandbuch 2](#_Toc157763170)

[5. Technische Dokumentation 2](#_Toc157763171)

[5.1. Grundlegendes 2](#_Toc157763172)

[5.2. Geocoding 2](#_Toc157763173)

[5.2.1. Grundlegendes 2](#_Toc157763174)

[5.3. Net generation 3](#_Toc157763175)

[5.4. Heat requirement 6](#_Toc157763176)

[5.5. Osm data 11](#_Toc157763177)

[5.6. Net generation pandapipes 15](#_Toc157763178)

[5.7. Heat generators 24](#_Toc157763179)

[5.8. Weiteres 29](#_Toc157763180)

[6. Validierung und Testergebnisse 29](#_Toc157763181)

[7. Anwendungsfälle und Fallstudien 29](#_Toc157763182)

[8. FAQ und Troubleshooting 29](#_Toc157763183)

[9. Zukunftsperspektiven 29](#_Toc157763184)

[10. Literaturverzeichnis 29](#_Toc157763185)

[11. Anhänge 29](#_Toc157763186)

# Zielstellung

# Installation und Setup

# Benutzerhandbuch

# Technische Dokumentation

## Grundlegendes

## Geocoding

### Grundlegendes

**Skript:** „geocodingETRS89.py“:

Dieses Skript ermöglicht die Geokodierung von Adressen aus einer CSV-Datei und transformiert die Koordinaten in das ETRS89 / UTM Zone 33N-Koordinatensystem.

**Funktion:** get\_coordinates(address)

**Beschreibung**: Diese Funktion nimmt eine Adresse als Eingabe entgegen und versucht, die entsprechenden geografischen Koordinaten (Längen- und Breitengrad) mithilfe des Nominatim-Geokodierungsdienstes abzurufen. Sie transformiert auch diese Koordinaten vom WGS84 (GPS) Koordinatensystem in das ETRS89 / UTM Zone 33N Koordinatensystem.

**Parameter**:

address (str): Die Adresse, für die Koordinaten abgerufen werden sollen.

**Rückgabewert**:

\_X und UTM\_Y) in Metern enthält, nämlich (utm\_x, utm\_y). Wenn die Geokodierung erfolgreich ist, werden die Koordinaten zurückgegeben, ansonsten (None, None).

**Funktion:** process\_data(input\_csv, output\_csv)

**Beschreibung**: Diese Funktion liest Daten aus einer Eingabe-CSV-Datei, verarbeitet jede Zeile, geokodiert die Adresse in der Zeile, um UTM-Koordinaten zu erhalten, und schreibt die Originaldaten zusammen mit den transformierten UTM-Koordinaten in eine Ausgabe-CSV-Datei.

**Parameter**:

input\_csv (str): Der Pfad zur Eingabe-CSV-Datei, die die zu verarbeitenden Daten enthält. Die CSV-Datei sollte eine bestimmte Struktur mit den Spalten "Land," "Bundesland," "Stadt," "Adresse" und zusätzlichen Spalten haben.

output\_csv (str): Der Pfad zur Ausgabe-CSV-Datei, in der die verarbeiteten Daten mit UTM-Koordinaten geschrieben werden.

**Verhalten**:

Die Funktion öffnet die Eingabe-CSV-Datei zum Lesen und die Ausgabe-CSV-Datei zum Schreiben.

Sie liest die Daten aus der Eingabe-CSV-Datei Zeile für Zeile, extrahiert relevante Informationen wie Land, Bundesland, Stadt und Adresse.

Sie kombiniert diese Informationen, um einen vollständigen Adressstring zu erstellen.

Sie ruft die get\_coordinates-Funktion auf, um die vollständige Adresse zu geokodieren und UTM-Koordinaten zu erhalten.

Die ursprünglichen Datensätze werden um zwei zusätzliche Spalten, "UTM\_X" und "UTM\_Y", erweitert, die die UTM-Koordinaten enthalten.

Die verarbeiteten Daten werden in die Ausgabe-CSV-Datei geschrieben, einschließlich der neu hinzugefügten UTM-Koordinaten.

Die Funktion gibt "Verarbeitung abgeschlossen" aus, wenn sie fertig ist.

**Hinweis**:

Der Code-Schnipsel am Ende des Skripts zeigt, wie die process\_data-Funktion aufgerufen wird, indem die Pfade zu Eingabe- und Ausgabe-CSV-Dateien angegeben werden. Sie können diese Zeilen aktivieren und anpassen, um die Geokodierung und die Datenverarbeitung auszuführen.

Diese Funktion ermöglicht die Geokodierung von Adressen aus einer Eingabe-CSV-Datei, das Abrufen von UTM-Koordinaten und das Erstellen einer neuen CSV-Datei mit den Originaldaten, die um UTM-Koordinaten erweitert wurden, für weitere räumliche Analysen oder Kartenanwendungen.

## Net generation

**net\_generation.py**

Die allgemeine Funktionalität der bereitgestellten Skripte und wie sie zusammenarbeiten, besteht darin, Netzwerkschichten für Heizungsleitungen auf der Grundlage von geografischen Daten zu generieren. Hier ist eine Übersicht darüber, wie der Prozess funktioniert:

1. Dateneingabe und Konfiguration:
   * Die Skripte beginnen mit dem Importieren der erforderlichen Bibliotheken für die Manipulation von geografischen Daten, wie GeoPandas und Shapely, sowie zusätzlichen Bibliotheken für mathematische Berechnungen und Netzwerkanalyse.
2. Dateneingabe:
   * Der Benutzer stellt projektspezifische Eingaben bereit, indem er die Variable "projekt" festlegt, um das Projekt (z. B. "Zittau" oder "Görlitz") zu spezifizieren, an dem gearbeitet wird. Diese Eingaben bestimmen die Quelle der geografischen Daten und die Koordinaten des Herstellers.
3. Laden der geografischen Daten:
   * Das Skript lädt die folgenden geografischen Datenschichten und Informationen:
     + OpenStreetMap (OSM) Straßenschicht im GeoJSON-Format, die das Straßennetz repräsentiert.
     + Datenpunkte aus einer CSV-Datei, die wahrscheinlich Informationen über Heizungsanforderungen oder andere relevante Daten enthält.
     + Koordinaten für den Standort des Herstellers (x\_coord und y\_coord).
4. Generierung von Netzwerkschichten:
   * Die Skripte verwenden eine Reihe von Funktionen, die hauptsächlich im Skript "simple\_MST.py" zu finden sind, um verschiedene Arten von Netzwerkschichten für Heizungsleitungen zu generieren:
     + generate\_lines: Generiert Linien basierend auf Eingabedatenpunkten und Attributen.
     + generate\_return\_lines: Generiert Rückleitungen für Heizungsleitungen.
     + generate\_network\_fl: Generiert Netzwerkschichten für Vorlaufleitungen.
     + generate\_network\_rl: Generiert Netzwerkschichten für Rücklaufleitungen.
     + generate\_mst: Generiert minimal spannende Bäume (MST), um Netzwerkverbindungen zu optimieren.
5. Koordinatentransformation:
   * Die generierten Netzwerkschichten werden mit dem Koordinatenbezugssystem (CRS) versehen, um eine korrekte räumliche Ausrichtung und Kompatibilität sicherzustellen.
6. Export von Netzwerkschichten:
   * Der letzte Schritt beinhaltet den Export der generierten Netzwerkschichten als GeoJSON-Dateien. Diese exportierten Schichten umfassen:
     + Vorlaufleitungen (Flow Lines)
     + Rücklaufleitungen (Return Lines)
     + Informationen zur Heizungsnachfrage (Wärmebedarf)
     + Produktionsanlagen (Producer Facilities)
7. Ausführung des Skripts:
   * Der Benutzer führt das Skript aus, und basierend auf dem ausgewählten Projekt lädt das Skript die entsprechenden Daten, generiert Netzwerkschichten, setzt das CRS und exportiert die Schichten in GeoJSON-Dateien.

Im Wesentlichen bieten diese Skripte einen Rahmen für die Generierung und Analyse von Netzwerkschichten für Heizungsleitungen. Sie nehmen projektspezifische Daten, transformieren sie in Netzwerkschichten und exportieren die Ergebnisse zur weiteren Analyse oder Verwendung in einem Geoinformationssystem (GIS). Der Prozess ist modular und kann durch Ändern von Eingabeparametern und Datenquellen an verschiedene Projekte angepasst werden.

**Dokumentation für import\_and\_create\_layers.py**:

Dieses Skript ist verantwortlich für das Laden von geografischen Daten, die Generierung von Netzwerkschichten für Heizungsleitungen und den Export der Ergebnisse in GeoJSON-Dateien. Es verwendet Funktionen aus dem Modul "simple\_MST", um Netzwerkschichten zu erstellen.

1. import\_osm\_street\_layer(osm\_street\_layer\_geojson\_file):
   * Diese Funktion lädt eine OpenStreetMap (OSM) Straßenschicht aus einer GeoJSON-Datei.
   * Sie gibt ein GeoDataFrame zurück, das die Geometrie der Straßenschicht enthält.
2. generate\_lines(layer, distance, angle\_degrees, df=None):
   * Diese Funktion generiert Linien basierend auf Eingabedatenpunkten und Attributen.
   * Sie berechnet den Wärmebedarf basierend auf dem bereitgestellten DataFrame (df) und erstellt Versatzlinien.
   * Gibt ein GeoDataFrame zurück, das die generierten Linien enthält.
3. load\_layers(osm\_street\_layer\_geojson\_file, data\_csv\_file\_name, x\_coord, y\_coord):
   * Diese Funktion lädt verschiedene Schichten und Daten, die für die Netzwerkgenerierung erforderlich sind.
   * Sie lädt die Straßenschicht, Datenpunkte aus einer CSV-Datei und den Standort des Herstellers.
   * Gibt die geladenen Schichten und Daten als GeoDataFrames und DataFrames zurück.
4. generate\_and\_export\_layers(osm\_street\_layer\_geojson\_file\_name, data\_csv\_file\_name, x\_coord, y\_coord, fixed\_angle=0, fixed\_distance=1):
   * Dies ist die Hauptfunktion, die den Prozess der Generierung von Netzwerkschichten orchestriert.
   * Sie ruft die erforderlichen Funktionen auf, um Linien für verschiedene Arten von Heizungsleitungen (HAST, Rücklauf, Vorlauf, Erzeugeranlagen) zu generieren.
   * Setzt das Koordinatenbezugssystem (CRS) und exportiert die resultierenden GeoDataFrames in GeoJSON-Dateien.

**Dokumentation für simple\_MST.py**:

Dieses Skript enthält verschiedene Funktionen, die zur Generierung von Netzwerkschichten verwendet werden, einschließlich Vorlaufleitungen (Flow Lines) und Rücklaufleitungen (Return Lines), sowie zur Suche nach minimal spannenden Bäumen (MST) zur Netzwerkoptimierung.

1. create\_offset\_points(point, distance, angle\_degrees):
   * Diese Funktion berechnet und gibt einen Versatzpunkt von einem gegebenen Punkt basierend auf einem angegebenen Abstand und Winkel in Grad zurück.
2. find\_nearest\_line(point, line\_layer):
   * Diese Funktion findet die nächstgelegene Linie in einer gegebenen Schicht zu einem bestimmten Punkt.
   * Sie gibt die Geometrie der nächsten Linie zurück.
3. create\_perpendicular\_line(point, line):
   * Diese Funktion erstellt eine senkrechte Linie von einem gegebenen Punkt zu einer angegebenen Linie.
   * Gibt die senkrechte Linie als LineString zurück.
4. process\_layer\_points(layer, layer\_lines):
   * Diese Funktion verarbeitet Schichtpunkte und findet senkrechte Linien zu benachbarten Linien.
   * Sie gibt eine Menge von Straßenendpunkten zurück, an denen sich senkrechte Linien mit benachbarten Linien schneiden.
5. generate\_return\_lines(layer, distance, angle\_degrees, layer\_lines):
   * Diese Funktion generiert Rücklaufleitungen für Heizungsleitungen basierend auf Eingabepunkten der Schicht, Abstand und Winkel.
   * Gibt eine Menge von Straßenendpunkten für Rücklaufleitungen zurück.
6. generate\_mst(points):
   * Diese Funktion generiert ein Netzwerk mit minimal spannenden Bäumen (MST) aus einer Menge von Punkten.
   * Sie erstellt einen Graphen, bei dem jeder Punkt ein Knoten ist und die Kanten auf der Entfernung zwischen den Punkten basieren.
   * Gibt den MST als GeoDataFrame zurück, das LineString-Geometrien enthält.
7. generate\_network\_fl(layer\_points\_fl, layer\_wea, layer\_lines):
   * Diese Funktion generiert Netzwerkschichten für Vorlaufleitungen (flow lines) im Heizungsnetz.
   * Sie berechnet senkrechte Linien für Punkte in den Schichten layer\_points\_fl und layer\_wea.
   * Gibt ein GeoDataFrame zurück, das die endgültige Netzwerkschicht enthält.
8. generate\_network\_rl(layer\_points\_rl, layer\_wea, fixed\_distance\_rl, fixed\_angle\_rl, layer\_lines):
   * Diese Funktion generiert Netzwerkschichten für Rücklaufleitungen (return lines) im Heizungsnetz.
   * Sie berechnet senkrechte Linien für Punkte in den Schichten layer\_points\_rl und layer\_wea mit festgelegtem Abstand und Winkel.
   * Gibt ein GeoDataFrame zurück, das die endgültige Netzwerkschicht enthält.

Diese Funktionen unterstützen gemeinsam die Generierung von Heizungsnetzwerkschichten, einschließlich Vorlauf- und Rücklaufleitungen, sowie die Optimierung dieser Netzwerke mithilfe minimal spannender Bäume (MST).

## Heat requirement

**Skriptname: heat\_requirement\_BDEW.py**

Beschreibung: Dieses Skript ist dafür konzipiert, das Wärmebedarfsprofil (Lastgang) für ein gegebenes Jahr auf der Grundlage verschiedener Parameter zu berechnen, einschließlich Temperaturdaten und Lastprofilkoeffizienten. Die Berechnungen erfolgen sowohl auf täglicher als auch auf stündlicher Ebene.

Funktionen:

1. import\_TRY(filename)
   * Beschreibung: Diese Funktion importiert Temperaturdaten aus einer spezifischen Datei im TRY-Format und extrahiert die stündlichen Temperaturwerte.
   * Parameter:
     + filename (str): Der Pfad zur TRY (Temperature Reference Year) Datendatei.
   * Rückgabewert:
     + temperature (numpy.ndarray): Ein Array mit den stündlichen Temperaturwerten.
2. generate\_year\_months\_days\_weekdays(year)
   * Beschreibung: Generiert Arrays, die Tage des Jahres, Monate, Tage des Monats und Wochentage (1-7) für ein gegebenes Jahr repräsentieren.
   * Parameter:
     + year (int): Das Jahr, für das die Arrays generiert werden sollen.
   * Rückgabewert:
     + Vier Arrays, die days\_of\_year, months, days und daily\_weekdays repräsentieren.
3. calculate\_daily\_averages(temperature)
   * Beschreibung: Berechnet tägliche Durchschnittstemperaturen aus stündlichen Temperaturdaten.
   * Parameter:
     + temperature (numpy.ndarray): Ein Array mit stündlichen Temperaturwerten.
   * Rückgabewert:
     + daily\_avg\_temperature (numpy.ndarray): Ein Array mit den täglichen Durchschnittstemperaturen.
4. calculate\_quarter\_hourly\_intervals(year)
   * Beschreibung: Generiert ein Array von viertelstündlichen Intervallen für das angegebene Jahr.
   * Parameter:
     + year (int): Das Jahr, für das viertelstündliche Intervalle generiert werden sollen.
   * Rückgabewert:
     + hourly\_intervals (numpy.ndarray): Ein Array von viertelstündlichen Intervallen.
5. get\_coefficients(lastprofiltyp, subtyp, daily\_data)
   * Beschreibung: Ruft Lastprofilkoeffizienten (A, B, C, D, mH, bH, mW, bW) für ein bestimmtes Lastprofil ab.
   * Parameter:
     + lastprofiltyp (str): Der Haupttyp des Lastprofils.
     + subtyp (str): Der Untertyp innerhalb des Haupttyps des Lastprofils.
     + daily\_data (pandas.DataFrame): Ein DataFrame mit Lastprofilkoeffizienten.
   * Rückgabewert:
     + Koeffizientenwerte als einzelne Floats.
6. get\_weekday\_factor(daily\_weekdays, lastprofiltyp, subtyp, daily\_data)
   * Beschreibung: Ruft Wochentagsfaktoren für ein bestimmtes Lastprofil ab, basierend auf den Wochentagen und Informationen zum Lastprofil.
   * Parameter:
     + daily\_weekdays (numpy.ndarray): Ein Array, das die Wochentage (1-7) für jeden Tag repräsentiert.
     + lastprofiltyp (str): Der Haupttyp des Lastprofils.
     + subtyp (str): Der Untertyp innerhalb des Haupttyps des Lastprofils.
     + daily\_data (pandas.DataFrame): Ein DataFrame mit Lastprofilkoeffizienten.
   * Rückgabewert:
     + weekday\_factors (numpy.ndarray): Ein Array von Wochentagsfaktoren für jeden Tag.
7. berechnung\_lastgang(weather\_data, JWB\_kWh, lastprofiltyp, subtyp, Feiertage, year=2019)
   * Beschreibung: Berechnet das Wärmebedarfsprofil (Lastgang) für ein angegebenes Jahr unter Verwendung von Temperaturdaten und Lastprofilkoeffizienten.
   * Parameter:
     + weather\_data (str): Der Pfad zur Temperaturdatendatei (TRY oder andere).
     + JWB\_kWh (float): Der jährliche Heizenergiebedarf in Kilowattstunden (kWh).
     + lastprofiltyp (str): Der Haupttyp des Lastprofils.
     + subtyp (str): Der Untertyp innerhalb des Haupttyps des Lastprofils.
     + Feiertage (numpy.ndarray): Ein Array von Feiertagen im Format 'datetime64[D]'.
     + year (int, optional): Das Jahr, für das der Lastgang berechnet werden soll (Standardwert ist 2019).
   * Rückgabewert:
     + hourly\_intervals (numpy.ndarray): Ein Array von viertelstündlichen Intervallen.
     + hourly\_heat\_demand (numpy.ndarray): Ein Array von stündlichen Wärmebedarfswerten (kW).
     + hourly\_temperature (numpy.ndarray): Ein Array von stündlichen Temperaturwerten (°C).
8. Jahresdauerlinie(hourly\_intervals, hourly\_heat\_demand)
   * Beschreibung: Zeichnet die jährliche Dauerkurve (Jahresdauerlinie) für das berechnete Wärmebedarfsprofil.
   * Parameter:
     + hourly\_intervals (numpy.ndarray): Ein Array von viertelstündlichen Intervallen.
     + hourly\_heat\_demand (numpy.ndarray): Ein Array von stündlichen Wärmebedarfswerten (kW).
9. calculate(JWB\_kWh=10000, lastprofiltyp="HMF", subtyp="03", year=2021)
   * Beschreibung: Einstiegspunkt-Funktion zur Berechnung des Lastgangs.
   * Parameter (mit Standardwerten):
     + JWB\_kWh (float): Der jährliche Heizenergiebedarf in Kilowattstunden (Standardwert ist 10.000 kWh).
     + lastprofiltyp (str): Der Haupttyp des Lastprofils (Standardwert ist "HMF").
     + subtyp (str): Der Untertyp innerhalb des Haupttyps des Lastprofils (Standardwert ist "03").
     + year (int): Das Jahr, für das der Lastgang berechnet werden soll (Standardwert ist 2021).
   * Rückgabewert:
     + Die berechneten Arrays hourly\_intervals, hourly\_heat\_demand und hourly\_temperature.

Hinweis:

* Das Skript bietet Funktionen zur Berechnung des Lastgangs auf der Grundlage von Temperaturdaten, Lastprofilkoeffizienten und verschiedenen Eingabeparametern.
* Die calculate-Funktion kann als Einstiegspunkt verwendet werden, um die Berechnung für verschiedene Szenarien anzupassen.

Verwendung:

1. Setzen Sie die gewünschten Eingabeparameter (z. B. JWB\_kWh, lastprofiltyp, subtyp, Jahr und weather\_data) in der calculate-Funktion.
2. Rufen Sie die calculate-Funktion auf, um die Ergebnisse des Lastgangs zu erhalten.
3. Optional können Sie die Jahresdauerlinie mithilfe der Jahresdauerlinie-Funktion erstellen. Um das Skript auszuführen, entfernen Sie die Kommentare in der letzten Zeile des Skripts und geben Sie die gewünschten Parameter für die Berechnung an.

**Skriptname: heat\_requirement\_VDI4655.py**

Verwendete Bibliotheken: • pandas (als pd): Zur Handhabung von Daten im tabellarischen Format. • numpy (als np): Für numerische Berechnungen. • matplotlib.pyplot (als plt): Zur Erstellung von Diagrammen. Hauptfunktionen und Zweck:

1. import\_TRY(filename)

* Importiert Temperatur- und Bewölkungsdaten aus einer TRY (Temperature Reference Year)-Datei.
* Gibt Arrays mit Temperatur- und Bewölkungswerten zurück.

1. import\_csv(filename)

* Importiert Daten aus einer CSV-Datei.
* Gibt ein DataFrame mit den Daten zurück.

1. generate\_year\_months\_days\_weekdays(year)

* Generiert Arrays, die Tage des Jahres, Monate, Tage des Monats und Wochentage für ein bestimmtes Jahr repräsentieren.
* Gibt diese Arrays zurück.

1. calculate\_daily\_averages(temperature, cloud\_cover)

* Berechnet tägliche Durchschnittstemperatur und Bewölkungsabdeckung aus stündlichen Daten.
* Gibt Arrays mit den täglichen Durchschnittswerten zurück.

1. calculate\_quarter\_hourly\_intervals(year)

* Generiert ein Array von viertelstündlichen Intervallen für ein bestimmtes Jahr.
* Gibt dieses Array zurück.

1. quarter\_hourly\_data(data)

* Erstellt ein Array mit viertelstündlichen Daten für ein Jahr.
* Gibt dieses Array zurück.

1. standardized\_quarter\_hourly\_profile(year, building\_type, days\_of\_year, type\_days)

* Berechnet ein standardisiertes viertelstündliches Profil für einen spezifischen Gebäudetyp unter Berücksichtigung von Faktoren wie Jahreszeit, Tageszeit und Bewölkungsabdeckung.
* Gibt viertelstündliche Intervalle und Nachfrageprofile für Strom, Heizung und Warmwasser zurück.

1. berechnung\_lastgang(weather\_data, factors, building\_type, anzahl\_personen\_haushalt, JEB\_Strom\_kWh, JEB\_Heizwärme\_kWh, JEB\_Trinkwarmwasser\_kWh, Feiertage, klimazone="9", year=2019)

* Berechnet die Energiebedarfsprofile (Strom, Heizung und Warmwasser) für ein angegebenes Jahr basierend auf Wetterdaten, Gebäudetyp und anderen Parametern.
* Gibt viertelstündliche Intervalle und normierte Nachfrageprofile zurück.

1. Jahresdauerlinie(weather\_data, factors, building\_type, anzahl\_personen\_haushalt, JEB\_Strom\_kWh, JEB\_Heizwärme\_kWh, JEB\_Trinkwarmwasser\_kWh, Feiertage, klimazone="9", year=2019)

* Zeichnet die jährliche Dauerkurve für die Energiebedarfsprofile.

1. calculate(JEB\_Heizwärme\_kWh, JEB\_Trinkwarmwasser\_kWh, JEB\_Strom\_kWh=1, building\_type="MFH", personen\_pro\_haushalt=2, year=2019, Klimazone="9")

* Haupt-Einstiegspunkt zur Berechnung der Energiebedarfsprofile.
* Ruft berechnung\_lastgang auf, um Nachfrageprofile zu berechnen, und erstellt dann die jährliche Dauerkurve mithilfe von Jahresdauerlinie.

Verwendung:

1. Setzen Sie die gewünschten Eingabeparameter wie Gebäudetyp, Anzahl der Bewohner, Energiebedarf, Jahr und Wetterdatendatei.
2. Rufen Sie die calculate-Funktion auf, um Energiebedarfsprofile zu berechnen.
3. Die jährliche Dauerkurve wird gezeichnet, um die Ergebnisse zu visualisieren.

Um das Skript auszuführen, entfernen Sie die Kommentare in den letzten Zeilen des Skripts und geben Sie die gewünschten Parameter für die Berechnung an. Mit diesem Skript können Energiebedarfsmuster für verschiedene Szenarien analysiert und visualisiert werden.

## Osm data

**Skriptname: import\_osm\_building\_layer.py**

Dieses Python-Skript mit dem Namen import\_osm\_building\_layer.py ist darauf ausgelegt, OpenStreetMap (OSM) Gebäudedaten aus einem bestimmten Bereich zu importieren und zu verarbeiten. Es verwendet die Bibliotheken geopandas, pandas und numpy, um geospatiale Daten zu verarbeiten und verschiedene Datenfilterung und Berechnungen durchzuführen. Nachfolgend finden Sie eine Dokumentation des Skripts:

Übersicht des Skripts: Das Skript führt die folgenden Aufgaben aus:

1. Importiert OSM Gebäudedaten aus einem bestimmten Bereich mithilfe einer Overpass-Abfrage und speichert sie als GeoJSON-Datei.
2. Liest die GeoJSON-Datei ein und fügt eine 'full\_address'-Spalte hinzu.
3. Filtert die Gebäudedaten basierend auf einer Liste von zu ignorierenden Gebäudetypen.
4. Konvertiert Koordinaten in ein anderes Koordinatenreferenzsystem (CRS).
5. Berechnet die Fläche jedes Gebäudes und fügt Spalten für den spezifischen Wärmebedarf, die Anzahl der Stockwerke und den jährlichen Heizbedarf hinzu.
6. Exportiert die gefilterten und berechneten Daten als neue GeoJSON-Datei.

Funktionen:

1. import\_and\_filter\_building()
   * Lädt OSM-Gebäudedaten mithilfe einer Overpass-Abfrage herunter und speichert sie als GeoJSON-Datei.
   * Liest die GeoJSON-Datei ein und fügt eine 'full\_address'-Spalte hinzu.
   * Filtert die Gebäudedaten basierend auf Adressen aus einer CSV-Datei.
   * Speichert die gefilterten Daten als neue GeoJSON-Datei.
2. filter\_building\_data(geojson\_file, output\_file)
   * Liest eine Eingabe-GeoJSON-Datei mit Gebäudedaten.
   * Filtert Gebäude anhand angegebener zu ignorierender Gebäudetypen aus.
   * Exportiert die gefilterten Daten in eine neue GeoJSON-Datei.
3. calculate\_building\_data(geojson\_file, output\_file)
   * Liest eine Eingabe-GeoJSON-Datei mit Gebäudedaten.
   * Konvertiert das Koordinatenreferenzsystem (CRS) in EPSG 25833.
   * Berechnet die Fläche jedes Gebäudes in Quadratmetern.
   * Fügt Spalten für den spezifischen Wärmebedarf, die Anzahl der Stockwerke und den jährlichen Heizbedarf basierend auf zufälligen Werten hinzu.
   * Exportiert das erweiterte GeoDataFrame in eine neue GeoJSON-Datei.

Verwendung:

1. Stellen Sie sicher, dass Sie die erforderlichen Python-Bibliotheken (geopandas, pandas und numpy) installiert haben.
2. Passen Sie die Overpass-Abfrage an, um den gewünschten Bereich und die OSM-Datentypen festzulegen.
3. Legen Sie die Pfade für Eingabe- und Ausgabe-GeoJSON-Dateien fest.
4. Entkommentieren Sie die gewünschten Funktionsaufrufe am Ende des Skripts (z. B. import\_and\_filter\_building(), filter\_building\_data(...) oder calculate\_building\_data(...)).

Hinweis: • Das Skript kann verwendet werden, um OSM-Gebäudedaten für bestimmte Gebäudetypen zu filtern und zu analysieren sowie zusätzliche Informationen wie den Heizbedarf zu berechnen.

Passen Sie bitte die Parameter und Funktionen des Skripts an Ihre spezifischen Anforderungen und den Interessenbereich Ihrer OSM-Gebäudedatenanalyse an.

**Skriptname: import\_osm\_data\_geojson.py**

Übersicht des Skripts: Das Skript ist darauf ausgelegt, OpenStreetMap (OSM)-Daten herunterzuladen und in das GeoJSON-Format umzuwandeln, wobei der Fokus auf bestimmten Elementen (z. B. Straßen oder Gebäuden) innerhalb eines angegebenen Stadtgebiets liegt. Es enthält Funktionen zum Erstellen einer Overpass-Abfrage, zum Herunterladen von Daten, zur Verarbeitung der Daten in GeoJSON-Features und zum Speichern der resultierenden GeoJSON-Datei.

Funktionen:

1. build\_query(city\_name, tags, element\_type="way")

* Generiert eine Overpass-Abfrage, um OSM-Daten basierend auf festgelegten Kriterien wie Stadtname, Tags (Schlüssel-Wert-Paare) und Elementtyp (Standardwert ist "way" für Straßen) abzurufen.
* Gibt die erstellte Abfrage als Zeichenfolge zurück.

1. download\_data(query, element\_type)

* Führt die Overpass-Abfrage aus, um OSM-Daten abzurufen.
* Konvertiert die abgerufenen Daten in GeoJSON-Features.
* Unterstützt zwei Elementtypen: "way" (für Straßen) und "building" (für Gebäude).
* Gibt eine GeoJSON-FeatureCollection zurück, die die heruntergeladenen Daten enthält.

1. json\_serial(obj)

* JSON-Serialisierer zum Umgang mit Objekten, die nicht nativ in JSON serialisierbar sind.
* Konvertiert Decimal-Objekte in Gleitkommazahlen.
* Wird in der Funktion save\_to\_file verwendet.

1. save\_to\_file(geojson\_data, filename)

* Speichert die GeoJSON-Daten in einer angegebenen Datei mit richtiger Formatierung.
* Verwendet die Funktion json\_serial zur Serialisierung von Decimal-Objekten.

1. run\_here()

* Definiert Standardparameter für Stadtname und Tags (z. B. Straßentyp).
* Erstellt eine Overpass-Abfrage, lädt OSM-Daten herunter und speichert sie als GeoJSON-Datei.
* Diese Funktion kann entkommentiert und ausgeführt werden, um das Skript mit vordefinierten Einstellungen auszuführen.

Verwendung:

1. Stellen Sie sicher, dass Sie die erforderlichen Python-Bibliotheken (overpy, json, decimal und geojson) installiert haben.
2. Passen Sie in der run\_here()-Funktion den Stadtname, die Tags und den Elementtyp an, um die spezifischen benötigten OSM-Daten zu zielen.
3. Führen Sie das Skript aus, um OSM-Daten herunterzuladen und als GeoJSON zu speichern.

Hinweis: • Das Skript ermöglicht es Ihnen, Ihre Datenextraktion anzupassen, indem Sie die Stadt, OSM-Tags und den Elementtyp (way oder building) festlegen. • Sie können die run\_here()-Funktion entkommentieren und ausführen, um das Skript mit vordefinierten Parametern auszuführen.

Dieses Skript ist ein nützliches Werkzeug, um OSM-Daten abzurufen und in das GeoJSON-Format umzuwandeln, wodurch sie für verschiedene geospatiale Anwendungen und Analysen geeignet sind.

**Skriptname: Wärmeversorgungsgebiete.py**

Übersicht des Skripts: Das Skript zielt darauf ab, Wärmeversorgungsgebiete basierend auf spezifischen Gebäudedaten und Eigenschaften zu bestimmen. Es verwendet Clustering-Techniken, um Gebäude in Quartiere zu gruppieren, und teilt dann überlappende Gebiete zwischen Quartieren mit unterschiedlichen Wärmeversorgungsmethoden zu. Das Endergebnis wird als GeoJSON-Datei gespeichert, die die Versorgungsmethode für jedes Quartier angibt.

Funktionen:

1. versorgungsgebiet\_bestimmen(flaechenspezifischer\_waermebedarf, schwell-wert\_waermenetz, schwellwert\_wasserstoff)

* Bestimmt die Wärmeversorgungsmethode für jedes Quartier basierend auf spezi-fischem Wärmebedarf und vordefinierten Schwellenwerten.
* Gibt die Wärmeversorgungsmethode als Zeichenfolge zurück: "Wärmenetzversor-gung", "Wasserstoffversorgung" oder "Einzelversorgungslösung".

1. clustering\_quartiere\_hdbscan(gdf, buffer\_size=10, min\_cluster\_size=30, min\_samples=1, schwellwert\_waermenetz=90, schwellwert\_wasserstoff=60)

* Führt ein Clustering der Gebäude in Quartiere mithilfe des HDBSCAN-Algorithmus durch.
* Berechnet den Wärmebedarf auf Quartierebene und die Versorgungsmethode.
* Löst überlappende Gebiete zwischen Quartieren mit unterschiedlichen Versor-gungsmethoden auf.
* Gibt ein GeoDataFrame mit Quartiersinformationen zurück.

1. postprocessing\_hdbscan(quartiere)

* Führt eine Nachverarbeitung durch, um überlappende Gebiete zwischen Quartieren weiter aufzulösen.
* Kombiniert Quartiere mit derselben Versorgungsmethode, die sich überschnei-den.
* Gibt ein GeoDataFrame mit nachverarbeiteten Quartiersinformationen zurück.

1. allocate\_overlapping\_area(quartiere)

* Weist überlappende Gebiete zwischen Quartieren mit unterschiedlichen Versor-gungsmethoden zu.
* Stellt sicher, dass jedes überlappende Gebiet auf der Grundlage des Wärmebedarfs zu einem Quartier gehört.
* Gibt ein GeoDataFrame mit zugewiesenen Quartiersinformationen zurück.

1. calculate\_building\_data(gdf, output\_filename)

* Berechnet zusätzliche Gebäudedaten, einschließlich Fläche, spezifischem Wärme-bedarf, Anzahl der Stockwerke und jährlichem Wärmebedarf.
* Speichert die aktualisierten Gebäudedaten als GeoJSON-Datei.
* Gibt das aktualisierte GeoDataFrame zurück.

1. run\_here()

* Führt den Hauptablauf des Skripts aus.
* Lädt Gebäudedaten aus einer GeoJSON-Datei.
* Berechnet Gebäudedaten, Quartiere und führt die Nachverarbeitung der Ergeb-nisse durch.
* Speichert die abschließende GeoJSON-Datei mit zugewiesenen Quartieren.

Verwendung:

1. Stellen Sie sicher, dass Sie die erforderlichen Python-Bibliotheken (numpy, pandas, geopandas, sklearn und hdbscan) installiert haben. Um hdbscan zu installieren, folgen Sie den in den Skriptkommentaren angegebenen Anweisungen.
2. Passen Sie die Parameter und Dateipfade bei Bedarf in der run\_here()-Funktion an.
3. Führen Sie das Skript aus, um das Clustering, die Nachverarbeitung und die Zuweisung der Quartiere durchzuführen.

Hinweis: • Das Skript bietet einen umfassenden Workflow zur Bestimmung von Wärmeversor-gungsgebieten basierend auf Gebäudeeigenschaften. • Es verwendet das HDBSCAN-Clustering, um Gebäude in Quartiere zu gruppieren und überlappende Gebiete aufzulösen. • Stellen Sie sicher, dass Sie Ihre Gebäudedaten im GeoJSON-Format haben und die Eingabe- und Ausgabedateipfade entsprechend angeben. • Das Skript ist flexibel und anpassbar gestaltet, sodass Sie die Clustering- und Schwellenwertparameter an Ihre speziellen Anforderungen anpassen können.

Dieses Skript kann ein wertvolles Werkzeug für Stadtplanung, Energieverteilung und Infrastrukturentwicklung sein, da es effizient Wärmeversorgungsgebiete in einer Stadt oder Region identifiziert und zuweist.

## Net generation pandapipes

**Skriptname: config\_plot.py**

Skriptübersicht: Es enthält Funktionen zum Darstellen von Netzwerkinformationen auf einer Karte.

Funktionen:

1. config\_plot(net, ax, show\_junctions=True, show\_pipes=True, show\_flow\_controls=True, show\_heat\_exchangers=True, show\_plot=False)
   1. Konfiguriert und zeigt die Informationen des Netzwerks auf einer Karte an.
   2. Zeigt Verbindungspunkte, Rohre, Durchflussregelungselemente und Wärmetauscher mit relevanten Details wie Druck, Temperatur, Durchflussrate und Rohreigenschaften an.
   3. Bietet Optionen zur Steuerung, welche Netzwerkelemente angezeigt werden. o Erfordert ein Pandapipes-Netzwerkobjekt (net) und ein Achsenobjekt (ax) zum Plotten.
   4. Setzen Sie show\_plot auf True, um die Grafik anzuzeigen.

Verwendung:

1. Stellen Sie sicher, dass Sie die erforderlichen Bibliotheken installiert haben. Sie können Pandapipes, Geopandas und andere Abhängigkeiten nach Bedarf installieren.
2. Pandapipes ist eine leistungsstarke Bibliothek für die hydraulische Modellierung von Rohrnetzwerken, die häufig für Wasser, Gas oder Fernwärmesysteme verwendet wird.
3. Führen Sie das Skript aus, um das Rohrnetzwerk zu generieren und zu analysieren. Das Skript zeigt Netzwerkinformationen auf einer Karte an.

Dieses Skript dient als praktisches Werkzeug zur Generierung, Visualisierung und Optimierung von Rohrnetzwerken unter Verwendung von Pandapipes und Geopandas. Es eignet sich für Anwendungen im Bereich der Hydraulik, Netzwerkgestaltung und -optimierung.

**Skriptname: net\_simulation\_calculation.py**

Skriptübersicht: Das Skript verwendet die Pandapipes-Bibliothek für die hydraulische Modellierung, um Fernwärmenetzwerke zu erstellen, zu analysieren und zu optimieren. Es enthält Funktionen zum Generieren von Netzwerkkomponenten, Korrigieren von Flussrichtungen, Optimieren von Rohrparameter (einschließlich Durchmesser und Typ), Erstellen von Reglern und Exportieren von Netzwerkd aten im GeoJSON-Format.

Funktionen:

1. get\_line\_coords\_and\_lengths(gdf)
   1. Extrahiert Linienkoordinaten und Längen aus einem GeoDataFrame.
   2. Berechnet die Länge jeder Linie im GeoDataFrame.
   3. Gibt Listen von Linienkoordinaten und Längen zurück.
2. get\_all\_point\_coords\_from\_line\_cords(all\_line\_coords)
   1. Ruft eindeutige Punktkoordinaten aus einer Liste von Linienkoordinaten ab.
   2. Gibt eine Liste eindeutiger Punktkoordinaten zurück.
3. create\_network(gdf\_vorlauf, gdf\_rl, gdf\_hast, gdf\_wea, qext\_w, ...)
   1. Generiert ein Fernwärmenetzwerk basierend auf geospatialen Daten unter Verwendung von Pandapipes.
   2. Importiert geospatiale Daten für verschiedene Netzwerkkomponenten (Vorlauf, Rücklauf, HAST, Erzeugeranlagen) aus GeoJSON-Dateien.
   3. Erstellt ein Pandapipes-Netzwerk, korrigiert Flussrichtungen und optimiert Durchmesserparameter für Rohre.
   4. Erstellt Wärmetauscher und Umwälzpumpen mit festgelegten Parametern.
   5. Gibt das generierte Netzwerk zurück.
4. create\_controllers(net, qext\_w, target\_temperature)
   1. Erstellt Regler für Wärmetauscher im Netzwerk.
   2. Richtet Controller ein, um die externe Wärmezufuhr des Wärmetauschers (qext\_w) basierend auf der Zieltemperatur anzupassen.
   3. Gibt das aktualisierte Netzwerk mit Controllern zurück.
5. correct\_flow\_directions(net)
   1. Korrigiert Flussrichtungen im Netzwerk, um physikalisch konsistente Ergebnisse sicherzustellen.
   2. Tauscht Verbindungspunkte für Rohre mit negativen Geschwindigkeiten aus.
   3. Berechnet die Rohrflüsse neu, um aktualisierte Ergebnisse zu erhalten.
   4. Gibt das Netzwerk mit korrigierten Flussrichtungen zurück.
6. optimize\_diameter\_parameters(initial\_net, element="pipe", v\_max=2.2, v\_min=1.8, dx=0.001)
   1. Optimiert Rohrparameter (Durchmesser) basierend auf angegebenen Geschwindigkeitsbeschränkungen.
   2. Passt Rohrdurchmesser iterativ an, bis die Geschwindigkeiten im definierten Bereich (v\_max und v\_min) liegen.
   3. Gibt das Netzwerk mit optimierten Rohrparametern zurück.
7. optimize\_diameter\_types(net, v\_max=1.0)
   1. Optimiert Rohrparameter (Rohrtyp) basierend auf angegebenen Geschwindigkeitsbeschränkungen.
   2. Passt Rohrtypen an, um Zielgeschwindigkeiten (v\_max) zu erreichen.
   3. Gibt das Netzwerk mit optimierten Rohrtypen zurück.
8. export\_net\_geojson(net)
   1. Exportiert die geografischen Daten des Netzwerks, einschließlich Rohren und Rohrattributen, in eine GeoJSON-Datei.
   2. Konvertiert Linienkoordinaten in LineString-Geometrien und setzt das Koordinatenreferenzsystem (CRS).
   3. Exportiert den GeoDataFrame in eine GeoJSON-Datei.
9. calculate\_worst\_point(net)
   1. Berechnet den ungünstigsten Punkt im Fernwärmenetzwerk, definiert als der Wärmetauscher mit dem niedrigsten Druckunterschied zwischen den Vorlauf- und Rücklaufleitungen
   2. Gibt den minimalen Druckunterschied (dp\_min) und den entsprechenden Index des Wärmetauschers (idx\_dp\_min) zurück.

Verwendung:

1. Stellen Sie sicher, dass Sie die erforderlichen Bibliotheken installiert haben. Sie können Pandapipes, Pandapower, Geopandas und andere Abhängigkeiten nach Bedarf installieren.
2. Passen Sie die Netzwerkgenerierungsparameter, einschließlich geospatialer Daten, Einstellungen für Controller und Optimierungskriterien, innerhalb der Funktionen create\_network() und create\_controllers() an.
3. Führen Sie das Skript aus, um das Fernwärmenetzwerk zu generieren, zu analysieren und zu optimieren. Es kann auch Flussrichtungen korrigieren und Netzwerkdaten zur weiteren Analyse exportieren.

Hinweis: • Dieses Skript bietet einen umfassenden Rahmen für die Erstellung, Steuerung und Optimierung von Fernwärmenetzwerken unter Verwendung von Pandapipes und Geopandas. • Sie können dieses Skript an Ihre eigenen geospatialen Daten, Netzwerkgestaltung und Optimierungsanforderungen anpassen. • Es enthält Funktionen zur Sicherstellung physikalisch konsistenter Ergebnisse, zur Optimierung von Rohrparametern und zur Exportierung von Netzwerkdaten zur Visualisierung und Analyse.

Dieses Skript ist ein wertvolles Werkzeug für Ingenieure und Forscher, die an der Gestaltung, Optimierung und Analyse von Fernwärmenetzwerken arbeiten. Es ermöglicht die Erstellung realistischer Netzwerkmodelle und die Feinabstimmung von Netzwerkparametern, um spezifische Leistungskriterien zu erfüllen.

**Skriptname: net\_simulation.py**

Das Skript "net\_simulation.py" ist für die Simulation von Heiznetzwerksystemen mit der Verwendung der Python-Bibliothek "pandapipes" verantwortlich. Es enthält Funktionen zum Generieren von Profilen, zur Initialisierung von Netzwerken, zur Durchführung von zeitbasierten Simulationen, zur Aktualisierung von Steuerungen und zur Berechnung sowie Speicherung von Ergebnissen. Im Folgenden wird jede Hauptfunktion und ihr Zweck im Skript erläutert:

1. generate\_profiles\_from\_geojson

Die Funktion generate\_profiles\_from\_geojson generiert Heizlastprofile auf Grundlage eines GeoJSON-Datensatzes, der Gebäudeinformationen und Heizbedarfsdaten enthält. Die Funktion kann verschiedene Berechnungsmethoden verwenden, um die Heizlastprofile zu erstellen.

* gdf\_HAST: Ein GeoDataFrame, das Gebäudeinformationen und Heizbedarfsdaten enthält.
* building\_type: Der Gebäudetyp, für den die Heizlastprofile berechnet werden sollen.
* calc\_method: Die Berechnungsmethode, die für die Erstellung der Profile verwendet werden soll (z. B. VDI 4655 oder BDEW).

2. initialize\_net\_geojson

Die Funktion initialize\_net\_geojson initialisiert ein Heiznetzwerk unter Verwendung von GeoJSON-Daten. Sie setzt Steuerungen, Flussrichtungen und andere Netzwerkparameter.

* gdf\_vl, gdf\_rl, gdf\_HAST, gdf\_WEA: GeoDataFrames, die die Vorlauf-, Rücklauf-, Gebäude- und Wärmequellendaten enthalten.
* qext\_w: Die externe Wärmezufuhr.
* return\_temperature, supply\_temperature, flow\_pressure\_pump, lift\_pressure\_pump: Verschiedene Parameter für die Netzwerkinitialisierung.
* diameter\_mm, pipetype, k, alpha, pipe\_creation\_mode: Parameter für Rohrleitungsattribute und -erstellung.

3. init\_timeseries\_opt

Die Funktion init\_timeseries\_opt bereitet die zeitbasierte Simulation für das Netzwerk vor. Sie legt die Controller-Daten und die Zieltemperatur fest.

* net: Das initialisierte Heiznetzwerk.
* qext\_w\_profiles: Profile für externe Wärmezufuhr über die Zeit.
* time\_steps: Zeitstufen der Simulation.
* target\_temperature: Die Zieltemperatur für die Return-Temperatur-Steuerungen.

4. update\_const\_controls

Die Funktion update\_const\_controls aktualisiert während der Simulation die konstanten Steuerungen mit neuen Daten.

* net: Das Heiznetzwerk.
* qext\_w\_profiles: Profile für externe Wärmezufuhr über die Zeit.
* time\_steps: Zeitstufen der Simulation.
* start, end: Indizes für den Zeitraum der Aktualisierung.

5. update\_return\_temperature\_controller

Die Funktion update\_return\_temperature\_controller aktualisiert die Zieltemperatur für den Return-Temperatur-Regler.

* net: Das Heiznetzwerk.
* temperature\_target: Die neue Zieltemperatur für die Return-Temperatur-Regler.

6. update\_supply\_temperature\_controls

Die Funktion update\_supply\_temperature\_controls aktualisiert die Steuerungen für die Versorgungstemperatur, wenn entsprechende Daten vorhanden sind.

* net: Das Heiznetzwerk.
* supply\_temperature: Die Versorgungstemperaturdaten.
* time\_steps: Zeitstufen der Simulation.
* start, end: Indizes für den Zeitraum der Aktualisierung.

7. create\_log\_variables

Die Funktion create\_log\_variables erstellt eine Liste von zu protokollierenden Variablen während der Zeitreihensimulation.

8. thermohydraulic\_time\_series\_net

Die Funktion thermohydraulic\_time\_series\_net führt eine zeitbasierte Simulation für das Heiznetzwerk durch. Sie berücksichtigt unterschiedliche Einstellungen für Steuerungen und Temperaturen.

* net: Das initialisierte Heiznetzwerk.
* yearly\_time\_steps: Zeitstufen für die Simulation.
* qext\_w\_profiles: Profile für externe Wärmezufuhr über die Zeit.
* start, end: Indizes für den Zeitraum der Simulation.
* supply\_temperature: Daten zur Versorgungstemperatur (optional).
* target\_temp: Die Zieltemperatur für die Return-Temperatur-Steuerungen.

9. calculate\_results

Die Funktion calculate\_results berechnet verschiedene Ergebnisse aus der Zeitreihensimulation, wie Massenfluss, Temperaturen und Druckdaten.

* net: Das Heiznetzwerk.
* net\_results: Die Ergebnisse der Simulation.

10. save\_results\_csv

Die Funktion save\_results\_csv speichert die Simulationsergebnisse in einer CSV-Datei.

* time\_steps: Zeitstufen der Simulation.
* qext\_kW: Externe Wärmezufuhr in kW.
* flow\_temp\_circ\_pump: Vorlauftemperatur der Umwälzpumpe.
* return\_temp\_circ\_pump: Rücklauftemperatur der Umwälzpumpe.
* filename: Der Dateiname für die CSV-Datei.

11. import\_results\_csv

Die Funktion import\_results\_csv importiert zuvor gespeicherte Simulationsergebnisse aus einer CSV-Datei.

* filename: Der Dateiname der CSV-Datei.

Diese Dokumentation bietet einen Überblick über die Funktionalität des Skripts "net\_simulation.py" und soll die Verwendung und Anpassung für Ihre speziellen Anforderungen erleichtern. Weitere detaillierte Informationen finden sich in den Funktionen selbst sowie in den Kommentaren im Skript.

**Skriptname: controllers.py**

Das Skript "controllers.py" definiert zwei benutzerdefinierte Regler für Heizungsnetzsimulationen unter Verwendung der pandapipes-Bibliothek. Diese Regler sind für die Steuerung des Drucks und der Rücklauftemperatur im Heizungsnetz ausgelegt. Im Folgenden werden detaillierte Erklärungen zu jedem Regler und seiner Funktionalität bereitgestellt.

1. WorstPointPressureController-Klasse Die WorstPointPressureController-Klasse ist für die Steuerung des Drucks an der ungünstigsten Stelle im Heizungsnetz verantwortlich. Sie erweitert die BasicCtrl-Klasse aus dem Modul "pandapower.control.basic\_controller".

Konstruktorparameter:

• net: Das Modell des Heizungsnetzes.

• worst\_point\_idx: Der Index der ungünstigsten Stelle (Wärmetauscher) im Netzwerk.

• circ\_pump\_pressure\_idx: Der Index des Druckregelungselements für Umwälzpumpen (Standardwert ist 0).

• target\_dp\_min\_bar: Der Zielwert für den minimalen Druckunterschied in bar.

• Toleranz: Toleranz für die Druckregelung.

• proportional\_gain: Proportionaler Verstärkungsfaktor für die Druckregelung.

• \*\*kwargs: Zusätzliche Schlüsselwortargumente für den Konstruktor der Basisklasse.

Methoden: time\_step(self, net, time\_step)

• Setzt den Iterationszähler zurück und gibt den aktuellen Zeitschritt zurück. is\_converged(self, net)

• Überprüft, ob die Druckregelung innerhalb der Toleranz konvergiert ist. control\_step(self, net)

• Implementiert die Druckregellogik zur Anpassung des Pumpendrucks basierend auf dem Ziel-Druckunterschied.

1. ReturnTemperatureController-Klasse Die ReturnTemperatureController-Klasse steuert die Rücklauftemperatur am Einlass des Wärmetauschers. Sie erweitert die BasicCtrl-Klasse aus dem Modul "pandapower.control.basic\_controller".

Konstruktorparameter:

• net: Das Modell des Heizungsnetzes.

• heat\_exchanger\_idx: Der Index des Wärmetauschers im Netzwerk.

• target\_temperature: Die Ziel-Rücklauftemperatur in Grad Celsius.

• Toleranz: Toleranz für die Temperaturregelung.

• min\_velocity: Minimale Durchflussgeschwindigkeit für die Massenstromberechnung.

• max\_velocity: Maximale Durchflussgeschwindigkeit für die Massenstromberechnung.

• \*\*kwargs: Zusätzliche Schlüsselwortargumente für den Konstruktor der Basisklasse.

Methoden: time\_step(self, net, time\_step)

• Setzt den Iterationszähler zurück und löscht die vorherigen Temperaturdaten. update\_state(self, net)

• Aktualisiert den Zustand des Reglers mit den aktuellen Einlauftemperaturen. calculate\_mass\_flow\_limits(self, net)

• Berechnet die minimalen und maximalen Massenstromgrenzwerte basierend auf dem Durchmesser des Wärmetauschers und den Geschwindigkeitsbeschränkungen. is\_converged(self, net)

• Überprüft, ob die Temperaturregelung innerhalb der Toleranz konvergiert ist. control\_step(self, net)

• Implementiert die Temperaturregellogik zur Anpassung der Massenstromrate am Einlass des Wärmetauschers.

Attribute:

• initial\_target\_temperature: Die anfängliche Ziel-Rücklauftemperatur.

• min\_mass\_flow: Der berechnete minimale Massenstrom basierend auf Geschwindigkeitsbeschränkungen.

• max\_mass\_flow: Der berechnete maximale Massenstrom basierend auf Geschwindigkeitsbeschränkungen.

• at\_min\_mass\_flow\_limit: Gibt an, ob der Regler am minimalen Massenstromlimit liegt.

• at\_max\_mass\_flow\_limit: Gibt an, ob der Regler am maximalen Massenstromlimit liegt.

Diese benutzerdefinierten Regler bieten erweiterte Steuermöglichkeiten für Heizungsnetzsimulationen unter Verwendung der pandapipes-Bibliothek und ermöglichen es Benutzern, die Leistung ihrer Heizungssysteme zu optimieren und feinzustellen.

**Skriptname: stanet\_import\_pandapipes.py**

Das Skript "stanet\_import\_pandapipes.py" wird verwendet, um Daten aus einer spezifischen CSV-Datei zu importieren, die eine Konfiguration eines Heizungsnetzes darstellt, und sie in ein pandapipes-Netzwerkmodell umzuwandeln. Das Skript führt auch eine Koordinatentransformation durch und integriert zusätzliche Daten für Wärmetauscher. Im Folgenden wird die Funktionalität des Skripts detailliert erläutert.

Importieren der erforderlichen Bibliotheken

Das Skript beginnt mit dem Importieren der notwendigen Python-Bibliotheken:

• pandas (pd): Wird zur Datenmanipulation verwendet.

• pandapipes (pp): Wird zur Erstellung des Heizungsnetzwerkmodells verwendet.

• numpy (np): Wird für numerische Operationen verwendet.

• pyproj.Transformer: Wird zur Koordinatentransformation verwendet.

• sys: Wird verwendet, um dem Systempfad ein benutzerdefiniertes Verzeichnis hinzuzufügen, um ein Modul zu importieren.

Funktion: create\_net\_from\_stanet\_csv

Diese Funktion ist der Hauptzugangspunkt für den Import und die Umwandlung der Daten aus der CSV-Datei in ein pandapipes-Netzwerkmodell. Sie akzeptiert mehrere Parameter:

• file\_path: Der Pfad zur CSV-Datei mit den Daten des Heizungsnetzwerks.

• supply\_temperature: Die Vorlauftemperatur des Heizungsnetzwerks.

• flow\_pressure\_pump: Der Druck-Sollwert für Umwälzpumpen für den Durchfluss.

• lift\_pressure\_pump: Der Druck-Sollwert für Umwälzpumpen für den Auftrieb.

Importieren und Verarbeiten der CSV-Daten

1. Das Skript liest die gesamte CSV-Datei in eine Liste von Zeilen ein und ignoriert dabei fehlerhafte Zeilen, da die gesamte Datei als einzelne Spalte gelesen wird.

2. Die Daten werden in verschiedene Dictionaries kategorisiert, basierend auf verschiedenen Objekttypen wie 'KNO' (Knoten), 'LEI' (Leitungen), 'WAE' (Wärmetauscher), 'HEA' (Heizquellen) und 'ZAE' (Verbraucher).

3. Für jeden Objekttyp erstellt das Skript ein DataFrame und stellt sicher, dass die Anzahl der Spalten zwischen Kopfzeile und Datenzeilen konsistent ist.

Koordinatentransformation

1. Das Skript definiert eine Koordinatentransformation von EPSG:31465 auf EPSG:25833 mithilfe der Klasse pyproj.Transformer.

2. Sie wendet diese Transformation auf die Koordinaten von Knoten, Leitungen und Wärmetauschern an.

Erstellung des pandapipes-Netzwerks

1. Ein neues leeres pandapipes-Netzwerkmodell wird mit "Wasser" als Fluidtyp erstellt.

2. Für jeden Knoten in den CSV-Daten wird ein Knoten im pandapipes-Netzwerk erstellt, mit Eigenschaften wie Druck und Temperatur.

3. Für jede Leitung wird ein Leitungselement im pandapipes-Netzwerk erstellt, einschließlich Eigenschaften wie Leitungslänge, Durchmesser und Wärmeleitfähigkeit.

4. Für jede Heizquelle wird eine Umwälzpumpe mit konstantem Druck im pandapipes-Netzwerk erstellt, die die Vorlauf- und Rücklaufknoten verbindet.

5. Für jeden Wärmetauscher wird ein Durchflussregelungselement und ein Wärmetauscher-Element im pandapipes-Netzwerk erstellt. Das Durchflussregelungselement wird zwischen dem Einlass des Wärmetauschers und einem neuen Knoten mit Zwischenkoordinaten platziert.

6. Das Wärmetauscher-Element wird dann zwischen dem Zwischenknoten und dem Auslass des Wärmetauschers platziert.

7. Die Wärmeanforderungen für jeden Wärmetauscher werden mithilfe einer externen Funktion (heat\_requirement\_BDEW.calculate) berechnet und in das Modell integriert.

pandapipes-Simulation

1. Das Skript startet eine pandapipes-Netzwerksimulation unter Verwendung der Funktion pp.pipeflow.

2. Das resultierende pandapipes-Netzwerk, die Zeitschritte und die Wärmeanforderungsdaten werden für weitere Analysen oder Simulationen zurückgegeben.

Zusätzliche Hinweise

• Das Skript enthält Fehlerbehandlung, um Ausnahmen während des Datenimportprozesses abzufangen und etwaige Fehlermeldungen auszugeben.

• Die Transformation zwischen Koordinatenreferenzsystemen (CRS) geht davon aus, dass die CSV-Daten in EPSG:31465 vorliegen und konvertiert die Koordinaten in EPSG:25833.

• Das Skript berechnet die Wärmeanforderungen für Wärmetauscher unter Berücksichtigung von Gebäudeprofilen und Verbrauchsdaten aus einer externen Funktion (heat\_requirement\_BDEW.calculate). Diese Werte werden als Parameter für Wärmetauscher im pandapipes-Netzwerk verwendet.

• Der Code ist teilweise kommentiert, um wichtige Abschnitte des Skripts zu erläutern.

Insgesamt erleichtert das Skript "stanet\_import\_pandapipes.py" die Umwandlung von CSV-basierten Heizungsnetzwerksdaten in ein pandapipes-Netzwerkmodell, was weitere Analysen und Simulationen des Verhaltens des Heizungsnetzwerks ermöglicht.

## Heat generators

„heat\_generator\_classes.py“

Das Skript "heat\_generator\_classes.py" enthält Python-Funktionen und Klassen, die für die Berechnung und Analyse von verschiedenen Wärmeerzeugungssystemen verwendet werden. Hier ist eine inhaltliche Beschreibung der wichtigsten Funktionen und Klassen im Skript:

1. **annuität()**: Diese Funktion berechnet die Annuität für verschiedene Kostenkomponenten eines Wärmeerzeugungssystems über die Nutzungsdauer. Sie wird in verschiedenen Wärmeerzeugungsklassen für die Wirtschaftlichkeitsberechnung verwendet und berücksichtigt Investitionskosten, Energiebedarf, Energiekosten und weitere Parameter.
2. **HeatPump-Klasse**: Diese Klasse stellt eine allgemeine Wärmepumpe dar und enthält Methoden zur Berechnung des Leistungskoeffizienten (COP) und der Wärmegewinnungskosten (WGK) für Wärmepumpensysteme. Sie enthält auch eine Methode zur Berechnung der Abwärme, die von der Wärmepumpe erzeugt wird.
3. **RiverHeatPump-Klasse**: Diese Unterklasse von HeatPump ist spezialisiert auf Wärmepumpensysteme, die Flusswasser als Wärmequelle nutzen. Sie enthält Methoden zur Berechnung der Wärme- und Stromproduktion sowie der WGK für solche Systeme.
4. **WasteHeatPump-Klasse**: Diese Unterklasse von HeatPump ist für Abwärmepumpensysteme vorgesehen und berechnet die Leistung und Kosten solcher Systeme basierend auf Abwärmequellen.
5. **Geothermal-Klasse**: Diese Unterklasse von HeatPump ist auf geothermische Wärmequellen spezialisiert. Sie enthält Methoden zur Berechnung der geothermischen Leistung und Kosten sowie zur Bestimmung der optimalen Betriebsstunden.
6. **CHP-Klasse**: Diese Klasse repräsentiert kombinierte Wärme- und Stromerzeugungssysteme, wie Blockheizkraftwerke (BHKW) und Holzgas-BHKW. Sie berechnet die Leistung, den Brennstoffverbrauch und die WGK für solche Systeme.
7. **BiomassBoiler-Klasse**: Diese Klasse stellt Biomassekessel dar und berechnet deren Leistung, Brennstoffbedarf und WGK.
8. **GasBoiler-Klasse**: Diese Klasse repräsentiert Gasheizkessel und berechnet deren Leistung, Brennstoffbedarf und WGK.
9. **SolarThermal-Klasse**: Diese Klasse ist für Solarthermie-Systeme verantwortlich. Sie berechnet die Wärmegewinnungskosten und berücksichtigt die Art des Solarkollektors und die Speichergröße.

Zusammengefasst ermöglicht das Skript "heat\_generator\_classes.py" die Berechnung und Analyse verschiedener Wärmeerzeugungssysteme unter Berücksichtigung von Investitionskosten, Energieeffizienz, Wirtschaftlichkeit und Umweltauswirkungen. Es bietet eine flexible Grundlage zur Untersuchung und Bewertung verschiedener Technologien zur Wärmeerzeugung.

Die Funktionen calculate\_factors, Berechnung\_Erzeugermix und optimize\_mix sind Teil des Skripts und dienen dazu, verschiedene Berechnungen und Optimierungen für Wärmeerzeugungssysteme durchzuführen. Hier ist eine Beschreibung der Funktionen:

1. calculate\_factors(Kapitalzins, Preissteigerungsrate, Betrachtungszeitraum): Diese Funktion berechnet die Faktoren q, r und T auf Grundlage von Kapitalzins, Preissteigerungsrate und Betrachtungszeitraum. Diese Faktoren werden in verschiedenen Berechnungen verwendet und repräsentieren den Zinsfaktor, den Preissteigerungsfaktor und die Betrachtungsdauer. Sie werden aus den gegebenen Prozentwerten umgewandelt und zurückgegeben.
2. Berechnung\_Erzeugermix(tech\_order, initial\_data, calc1, calc2, TRY, COP\_data, Gaspreis, Strompreis, Holzpreis, BEW, variables=[], variables\_order=[], kapitalzins=5, preissteigerungsrate=3, betrachtungszeitraum=20): Diese Funktion berechnet verschiedene Kennzahlen und Ergebnisse für einen Mix von Wärmeerzeugungstechnologien. Sie verwendet eine Reihe von Parametern, darunter die Reihenfolge der Technologien (tech\_order), Anfangsdaten (initial\_data), Berechnungsfaktoren (calc1 und calc2), den Jahresenergieverbrauch (TRY), Leistungskoeffizientendaten für Wärmepumpen (COP\_data), Gaspreis, Strompreis, Holzpreis, BEW, und verschiedene optionale Variablen für die Anpassung der Technologien.

Die Funktion führt eine Vielzahl von Berechnungen durch, um den Wärme- und Stromertrag, die Wirtschaftlichkeit, die Emissionen und andere Kennzahlen für jeden Technologietyp im Mix zu bestimmen. Sie verwendet auch die zuvor definierten Faktoren q, r und T, um die Ergebnisse über den Betrachtungszeitraum zu diskontieren und an die Preisentwicklung anzupassen. Die berechneten Ergebnisse werden in einem Python Dictionary (general\_results) gespeichert und am Ende zurückgegeben.

1. optimize\_mix(tech\_order, initial\_data, calc1, calc2, TRY, COP\_data, Gaspreis, Strompreis, Holzpreis, BEW, kapitalzins, preissteigerungsrate, betrachtungszeitraum): Diese Funktion ist anscheinend für die Optimierung des besten Technologiemixes für die Wärmeerzeugung verantwortlich. Sie verwendet die gleichen Parameter wie Berechnung\_Erzeugermix, einschließlich der Liste der Technologien (tech\_order) und der anderen Eingangsdaten. Die genaue Implementierung der Optimierung ist in der gegebenen Funktion nicht ersichtlich, da dies von den spezifischen Anforderungen und Zielen abhängen würde.

Zusammenfassend ermöglichen diese Funktionen die umfassende Analyse und Optimierung von Wärmeerzeugungssystemen unter Berücksichtigung verschiedener Parameter wie Kapitalkosten, Preissteigerungsraten, Technologieoptionen und Betrachtungszeiträume. Sie helfen bei der Bewertung der Wirtschaftlichkeit und Umweltauswirkungen verschiedener Technologiemixe für die Wärmeerzeugung.

„Solarthermie.py“

1. **Einleitung:** Dieses Python-Skript wurde von Dipl.-Ing. (FH) Jonas Pfeiffer erstellt, um den Ertrag von Solarthermieanlagen in einem Wärmenetz zu berechnen. Es nutzt die Berechnungsgrundlage "ScenoCalc Fernwärme 2.0" von <https://www.scfw.de/>. Solarthermieanlagen wandeln Sonnenenergie in Wärme um, die dann zur Beheizung von Gebäuden oder zur Warmwasserbereitung verwendet wird.
2. **Import von Bibliotheken:** Das Skript beginnt mit dem Import von verschiedenen Python-Bibliotheken, die später im Code verwendet werden. Dazu gehören math, numpy, und datetime. Einige spezifische Funktionen aus heat\_generators.Solarstrahlung werden ebenfalls importiert. Diese Funktionen sind für die Berechnung der Solarstrahlung erforderlich.
3. **Die Hauptfunktion "Berechnung\_STA":** Diese Funktion ist das Herzstück des Skripts und wird verwendet, um den Ertrag der Solarthermieanlage zu berechnen. Sie erfordert eine Vielzahl von Eingabeparametern:
   * Bruttofläche\_STA: Die Bruttofläche der Solarthermieanlage.
   * VS: Das Volumen des Speichers.
   * Typ: Der Typ der Solarthermieanlage (Flachkollektor oder Vakuumröhrenkollektor).
   * Last\_L: Eine Liste von Lasten über einen bestimmten Zeitraum.
   * VLT\_L: Eine Liste von Vorlauftemperaturen über denselben Zeitraum.
   * RLT\_L: Eine Liste von Rücklauftemperaturen über denselben Zeitraum.
   * TRY: Ein Tupel von Listen, die Temperatur, Windgeschwindigkeit, Direktstrahlung und Globalstrahlung enthalten.
   * time\_steps: Eine Liste von Zeitstempeln, die den Zeitverlauf der Berechnungen darstellen.
   * calc1 und calc2: Indices, die den Zeitbereich innerhalb der TRY-Daten angeben, der für die Berechnung verwendet werden soll.
   * duration: Die Gesamtdauer der Berechnungen.
   * Weitere optionale Parameter wie Tsmax, Longitude, STD\_Longitude, Latitude, usw.

Die Funktion beginnt mit der Anpassung der Eingangsdaten an die gewünschte Zeitauflösung und berechnet dann den Ertrag der Solaranlage über den angegebenen Zeitraum. Dabei werden die Temperaturen, Strahlungswerte und andere Parameter berücksichtigt.

1. **Berechnung der Kollektorwirkungsgrade:** Abhängig vom gewählten Kollektortyp (Flachkollektor oder Vakuumröhrenkollektor) und den spezifischen Konstanten für diesen Typ werden die Kollektorwirkungsgrade berechnet. Diese Wirkungsgrade sind entscheidend für die Berechnung des Ertrags.
2. **Berechnung der Verluste:** Das Skript berechnet verschiedene Arten von Verlusten, darunter Verluste in Verbindungsleitungen, Verluste in internen Rohrleitungen und Speicherverluste. Diese Verluste werden während der Berechnung des Ertrags berücksichtigt.
3. **Berechnung des Ertrags und der Speicherladung:** Die Hauptberechnung erfolgt innerhalb einer Schleife, die durch die Zeitpunkte in time\_steps iteriert. In jeder Iteration werden die Kollektorleistung, die Verluste und die Speicherladung neu berechnet. Der Ertrag der Solaranlage wird berechnet und in der Ausgabe erfasst.
4. **Rückgabe der Ergebnisse:** Am Ende der Berechnung gibt die Funktion die Gesamtwärmemenge und die Wärmeleistung der Solaranlage in Form von NumPy-Arrays zurück.

Dieses Skript ist äußerst nützlich für Ingenieure und Planer, die Solarthermieanlagen in Wärmenetzen planen oder optimieren möchten. Es ermöglicht die detaillierte Berechnung des Ertrags unter Berücksichtigung einer Vielzahl von Faktoren und kann zur Simulation verschiedener Szenarien und zur Optimierung von Solarthermieanlagen verwendet werden. Es stellt sicher, dass die Solaranlage unter den gegebenen Bedingungen effizient arbeitet und hilft, den Beitrag zur nachhaltigen Energieerzeugung zu maximieren.

„Solarstrahlung.py“

Das Skript "Solarstrahlung.py" dient der Berechnung der Solarstrahlung auf einer schrägen Oberfläche. Es verwendet verschiedene meteorologische und geografische Parameter, um den Einfluss von Sonnenstrahlung auf Kollektoren oder Solarpaneele zu analysieren. Hier sind die Schlüsselaspekte des Skripts:

Import von Bibliotheken:

Das Skript beginnt mit dem Import der benötigten Bibliotheken, insbesondere NumPy, das für mathematische Berechnungen verwendet wird.

Grad-Radian-Konversion:

Es wird eine Konstante DEG\_TO\_RAD definiert, um Grad in Radian umzurechnen. Diese Konversion wird in späteren Berechnungen verwendet.

Funktion deg\_to\_rad:

Eine Funktion deg\_to\_rad wird definiert, um Grad in Radian umzurechnen. Dies wird in den Berechnungen für Winkel benötigt.

Hauptfunktion Berechnung\_Solarstrahlung:

Dies ist die Hauptfunktion des Skripts, die die Solarstrahlung auf der schrägen Oberfläche berechnet. Sie erfordert eine Vielzahl von Eingabeparametern, darunter:

* Globalstrahlung\_L: Die Globalstrahlung auf horizontaler Oberfläche (W/m²).
* D\_L: Die Direktstrahlung auf horizontaler Oberfläche (W/m²).
* Tag\_des\_Jahres\_L: Der Tag des Jahres.
* time\_steps: Eine Liste von Zeitstempeln, um den Zeitverlauf der Berechnungen darzustellen.
* Longitude: Die geografische Länge (°).
* STD\_Longitude: Die Zeitzone (°).
* Latitude: Die geografische Breite (°).
* Albedo: Die Albedo der Oberfläche (Reflexionskoeffizient).
* IAM\_W und IAM\_N: Dictionaries mit Einstrahlungs-Winkelabhängigkeitsdaten für die EW- und NS-Richtung.
* EWCaa: Der Azimutwinkel des Kollektors (°).
* CTA: Der Neigungswinkel des Kollektors (°).

Die Funktion führt dann eine Reihe von Berechnungen durch, um Folgendes zu ermitteln:

* Die Sonnenzeit unter Berücksichtigung der geografischen Länge und Zeitkorrektur.
* Die Sonnendeklination und den Stundenwinkel der Sonne.
* Den Sonnenzenitwinkel und Azimutwinkel der Sonne.
* Den Einfallswinkel der Sonnenstrahlung auf den Kollektor.
* Die Einstrahlungsfaktoren IAM\_EW und IAM\_NS unter Berücksichtigung des Einfallswinkels.
* Das Verhältnis der Strahlungsintensität auf dem geneigten Kollektor zur horizontalen Oberfläche.
* Die Direktstrahlung auf horizontaler Oberfläche und die diffuse Strahlung.
* Den atmosphärischen Diffusanteil Ai.
* Die Gesamtstrahlung auf der schrägen Oberfläche GT\_H\_Gk.
* Die direkte Strahlung auf der schrägen Oberfläche GbT.
* Die diffuse Strahlung auf der schrägen Oberfläche GdT\_H\_Dk.
* Das Produkt der Einstrahlungsfaktoren K\_beam.

Zusammenfassung der Ergebnisse:

Die Funktion gibt schließlich vier wichtige Ergebnisse zurück:

1. GT\_H\_Gk: Die Gesamtstrahlung auf der schrägen Oberfläche, einschließlich des direkten und diffusen Beitrags sowie des durch Albedo reflektierten Beitrags.
2. K\_beam: Das Produkt der Einstrahlungsfaktoren für die EW- und NS-Richtung, das den Einfluss des Einfallswinkels auf die Strahlung beschreibt.
3. GbT: Die Direktstrahlung auf der schrägen Oberfläche.
4. GdT\_H\_Dk: Die diffuse Strahlung auf der schrägen Oberfläche.

Verwendungszweck:

Dieses Skript ist nützlich für Ingenieure, die sich mit der Leistungsprognose von Solarthermieanlagen oder Photovoltaiksystemen befassen. Es ermöglicht eine detaillierte Analyse der Strahlungseinflüsse auf schräge Kollektoren, um den Ertrag genauer zu bestimmen und die Anlagenausrichtung zu optimieren.

## Weiteres

# Validierung und Testergebnisse

# Anwendungsfälle und Fallstudien

# FAQ und Troubleshooting

# Zukunftsperspektiven

# Literaturverzeichnis

# Anhänge