Calculation and Optimization of Heat Generator Systems

Dipl.-Ing. (FH) Jonas Pfeiffer

2024-09-09

1 Introduction

This document describes the mathematical models and optimization functions used to calculate and evaluate various heat generation systems. The code focuses on optimizing energy efficiency and economic viability, applying the VDI 2067 guidelines for cost analysis.

2 Economic Analysis According to VDI 2067

2.1 Einleitung

Die Berechnung der Wirtschaftlichkeit technischer Anlagen ist ein zentraler Bestandteil des Energiemanagements. Die Annuitätsmethode gemäß VDI 2067 ermöglicht es, die Gesamtkosten einer technischen Anlage über die gesamte Nutzungsdauer zu erfassen und zu bewerten. Die Kosten umfassen die kapitalgebundenen, bedarfsgebundenen und betriebsgebundenen Kosten, sowie Erlöse.

2.2 Die Annuität

Die Annuität bezeichnet eine jährliche Zahlung, die Kapital- und Betriebskosten sowie Wartungskosten und gegebenenfalls Erlöse berücksichtigt. Die Berechnung der Annuität basiert auf den folgenden Komponenten:

2.2.1 Formel zur Berechnung der Annuität

Die Annuität A_N wird durch die Summe der folgenden Komponenten bestimmt:

$$A_N = A_{N,K} + A_{N,V} + A_{N,B} + A_{N,S} - A_{N,E}$$

wobei:

- $A_{N,K}$: Kapitalgebundene Kosten
- $A_{N,V}$: Bedarfsgebundene Kosten

- $A_{N,B}$: Betriebsgebundene Kosten
- $A_{N,S}$: Sonstige Kosten
- $A_{N,E}$: Erlöse

2.2.2 Kapitalgebundene Kosten

Die kapitalgebundenen Kosten $A_{N,K}$ umfassen die Investitionskosten und den Restwert der Anlage:

$$A_{N,K} = (A_0 - R_W) \cdot a$$

wobei:

- A_0 die Anfangsinvestition ist,
- R_W der Restwert der Anlage nach Ablauf der Nutzungsdauer T ist,
- a der Annuitätsfaktor ist:
$$a = \frac{q-1}{1-q^{-T}}$$
- q der Zinsfaktor ist, also $q = 1 + \text{Zinssatz}$.

2.2.3 Bedarfsgebundene Kosten

Die bedarfsgebundenen Kosten $A_{N,V}$ werden aus dem Energiebedarf und den Energiekosten berechnet:

$$A_{N,V} = \text{Energiebedarf} \cdot \text{Energiekosten} \cdot a \cdot b_V$$

wobei:

$$b_V = \frac{1 - \left(\frac{r}{q}\right)^T}{q - r}$$

und r der Preissteigerungsfaktor (Inflation) ist.

2.2.4 Betriebsgebundene Kosten

Die betriebsgebundenen Kosten $A_{N,B}$ setzen sich aus den Betriebskosten und den Wartungskosten zusammen:

$$A_{N,B} = (\text{Bedienaufwand} \cdot \text{Stundensatz} + A_0 \cdot (f_{\text{Inst}} + f_{\text{W\_Insp}})/100) \cdot a \cdot b_B$$

wobei:

- f_{Inst}: Installationsfaktor,
- $f_{\text{W\_Insp}}$: Wartungs- und Inspektionsfaktor.

2.2.5 Sonstige Kosten

Sonstige Kosten $A_{N,S}$ können in ähnlicher Weise berechnet werden, wobei keine weiteren Parameter in diesem Beispiel angegeben sind.

2.2.6 Erlöse

Falls Erlöse $A_{N,E}$ vorhanden sind (z.B. durch den Verkauf von Energie), werden diese von der Annuität abgezogen:

$$A_{N,E} = E_1 \cdot a \cdot b_E$$

2.2.7 Rückgabewert

Die Gesamtannuität wird als Summe der Komponenten berechnet. Sie ergibt die jährlichen Gesamtkosten oder Erträge der Anlage:

$$A_N = -(A_{N,K} + A_{N,V} + A_{N,B} + A_{N,S} - A_{N,E})$$

2.3 Zusammenfassung

Die Annuitätsberechnung gemäß VDI 2067 bietet eine umfassende Methode, um die Kosten und Erlöse einer technischen Anlage über die gesamte Nutzungsdauer zu bewerten. Durch die Anwendung von Kapitalwertfaktoren und Preissteigerungsfaktoren können die jährlichen Belastungen und Einsparungen realitätsnah abgebildet werden.

3 Optimierungsfunktion für den Erzeugermix

3.1 Einleitung

Die Berechnungsfunktion `Berechnung_Erzeugermix` ermittelt die optimale Energieerzeugung für einen vorgegebenen Mix an Technologien. Ziel ist es, die Wärmeerzeugung für ein bestimmtes Lastprofil unter Einbeziehung verschiedener Kosten-, Effizienz- und Emissionsfaktoren zu berechnen.

3.2 Mathematisches Modell

3.2.1 Eingangsparameter

Die Berechnungsfunktion nimmt eine Reihe von Eingangsparametern an, die die technologischen und ökonomischen Bedingungen beschreiben. Diese beinhalten unter anderem:

- **tech_order**: Liste der zu betrachtenden Technologien.
- **initial_data**: Tuple bestehend aus Zeitpunkten, Lastprofil, Vorlauf- und Rücklauftemperaturen.

- **Gaspreis**, **Strompreis**, **Holzpreis**: Energiekosten in €/kWh.
- **BEW**: Spezifische CO2-Emissionen des Strommixes in kg CO2/kWh.
- **kapitalzins**, **preissteigerungsrate**, **betrachtungszeitraum**: Finanzielle Parameter für die Kostenberechnung.

3.2.2 Berechnungslogik

Die Funktion berechnet zunächst die Jahreswärmebedarfe basierend auf dem Lastprofil L und der zeitlichen Auflösung:

$$\text{Jahreswärmebedarf} = \frac{\sum L}{1000} \cdot \text{duration}$$

Die Wärmebedarfsfunktion läuft über eine Schleife für jede Technologie in der `tech_order`. Je nach Art der Technologie (Solarthermie, Abwärme, Geothermie usw.) wird ein spezifisches Berechnungsmodell angewandt.

3.2.3 Technologiespezifische Berechnung

Jede Technologie verwendet unterschiedliche Berechnungsmodelle:

- **Solarthermie**: Berechnet den Ertrag basierend auf der Vorlauftemperatur und der solaren Einstrahlung aus dem Testreferenzjahr (TRY).
- **Wärmepumpen** und **Abwärme**: Verwenden den COP-Wert (*Coefficient of Performance*) und Strompreis zur Ermittlung der Betriebsaufwendungen.
- **Blockheizkraftwerke (BHKW)**: Berücksichtigen sowohl thermische als auch elektrische Leistungen, sowie den Brennstoffverbrauch.

3.3 Kapital- und Emissionskosten

Neben den Betriebskosten werden auch kapitalgebundene und emissionsbasierte Kosten berechnet. Der kapitalgebundene Kostenanteil ergibt sich aus:

$$A_{N,K} = A_0 \cdot \frac{(q-1)}{1-q^{-T}}$$

wobei $q = 1 + \text{Zinsrate}$.

Die spezifischen CO2-Emissionen werden pro erzeugte Wärmemenge berechnet:

$$\text{CO2\_Emissionen} = \frac{\sum \text{Wärmemenge}_i \cdot \text{spec\_co2}_i}{\text{Jahreswärmebedarf}}$$

3.4 Zusammenfassung

Die Funktion `Berechnung_Erzeugermix` führt eine detaillierte Berechnung der Energieerzeugung durch, indem sie mehrere Technologien gleichzeitig berücksichtigt. Die Berechnung erfolgt basierend auf stündlichen Daten für Lastprofile, Temperaturen und Emissionen.

4 Berechnungsfunktion für den Erzeugermix

4.1 Einleitung

Die Optimierungsfunktion `optimize_mix` verwendet mathematische Optimierungstechniken, um den Mix aus Energieerzeugungstechnologien zu optimieren. Das Ziel der Optimierung ist es, die Kosten, CO2-Emissionen und den Primärenergieverbrauch zu minimieren, indem verschiedene Technologien mit unterschiedlichen Parametern berücksichtigt werden.

4.2 Mathematisches Modell

4.2.1 Zielgrößen

Die Optimierung basiert auf der Minimierung einer gewichteten Summe von drei Zielgrößen:

$$\text{Ziel} = w_{\text{WGK}} \cdot \text{WGK\_Gesamt} + w_{\text{CO2}} \cdot \text{CO2\_Emissionen\_Gesamt} + w_{\text{Primärenergie}} \cdot \text{Primärenergie\_Faktor\_Gesamt}$$

Hierbei sind $w_{\text{WGK}}, w_{\text{CO2}}, w_{\text{Primärenergie}}$ die Gewichte, die den Einfluss der jeweiligen Zielgröße auf das Gesamtergebnis steuern.

4.2.2 Optimierungsverfahren

Die Optimierung erfolgt mittels des `SLSQP`-Algorithmus, der für nichtlineare Probleme mit Nebenbedingungen geeignet ist. Der Algorithmus sucht nach den optimalen Parametern für die Technologien (z.B. Fläche für Solarthermie, Leistung für BHKW), die die gewichtete Summe der Zielgrößen minimieren.

4.2.3 Nebenbedingungen

Für jede Technologie werden Schranken (*bounds*) für die zu optimierenden Parameter definiert, um physikalisch sinnvolle Werte sicherzustellen. Zum Beispiel:

- Für die Fläche eines Solarthermie-Systems: $\text{min\_area} \leq \text{Fläche} \leq \text{max\_area}$
- Für die Leistung eines BHKW: $\text{min\_Leistung} \leq \text{Leistung} \leq \text{max\_Leistung}$

4.3 Ergebnis

Nach erfolgreicher Optimierung gibt die Funktion die optimierten Parameter für jede Technologie zurück. Diese Parameter minimieren die gewichtete Summe der Kosten, CO2-Emissionen und des Primärenergieverbrauchs.

4.4 Zusammenfassung

Die Funktion `optimize_mix` erlaubt eine simultane Optimierung mehrerer Technologien basierend auf benutzerdefinierten Zielgrößen. Durch die Verwendung von mathematischen Optimierungsverfahren wie `SLSQP` werden die besten Kombinationen von Technologien und Parametern ermittelt.

5 HeatPump Class

The `HeatPump` class represents a heat pump system and provides methods to calculate various performance and economic metrics. The class is highly modular, making it possible to adapt it to different types of heat sources and use cases. The primary attributes and methods of the class are detailed below:

5.1 Attributes

- `name (str)`: The name of the heat pump.
- `spezifische_Investitionskosten_WP (float)`: Specific investment costs of the heat pump per kW. Default is 1000 €/kW.
- `Nutzungsdauer_WP (int)`: Useful life of the heat pump in years. Default is 20 years.
- `f_Inst_WP (float)`: Installation factor for the heat pump. Default is 1.
- `f_W_Insp_WP (float)`: Maintenance and inspection factor for the heat pump. Default is 1.5.
- `Bedienaufwand_WP (float)`: Operating effort for the heat pump in hours. Default is 0.
- `f_Inst_WQ (float)`: Installation factor for the heat source. Default is 0.5.
- `f_W_Insp_WQ (float)`: Maintenance and inspection factor for the heat source. Default is 0.5.
- `Bedienaufwand_WQ (float)`: Operating effort for the heat source in hours. Default is 0.
- `Nutzungsdauer_WQ_dict (dict)`: Dictionary containing useful life of different heat sources (e.g., waste heat, river water).
- `co2_factor_electricity (float)`: CO_2 emission factor for electricity in tCO_2/MWh. Default is 2.4.

5.2 Methods

- `COP_WP(VLT_L, QT, COP_data)`: Calculates the coefficient of performance (COP) of the heat pump by interpolating the COP data based on flow temperatures (`VLT_L`) and source temperatures (`QT`).
- `WGK(Wärmeleistung, Wärmemenge, Strombedarf, spez_Investitionskosten_WQ, Strompreis, q, r, T, BEW, stundensatz)`: Calculates the weighted average cost of heat generation (WGK) based on thermal performance, investment, and operational costs.

6 Geothermal Class

The `Geothermal` class models a geothermal heat pump system, inheriting from the `HeatPump` base class. It includes methods for simulating the geothermal heat extraction process and calculating various economic and environmental metrics.

6.1 Attributes

- `Fläche (float)`: Area available for the geothermal installation in square meters.
- `Bohrtiefe (float)`: Drilling depth for geothermal wells in meters.
- `Temperatur_Geothermie (float)`: Temperature of the geothermal source in degrees Celsius.
- `spez_Bohrkosten (float)`: Specific drilling costs per meter. Default is 100 €/m.
- `spez_Entzugsleistung (float)`: Specific heat extraction rate per meter. Default is 50 W/m.
- `Vollbenutzungsstunden (float)`: Full utilization hours per year. Default is 2400 hours.
- `Abstand_Sonden (float)`: Distance between probes in meters. Default is 10 m.
- `min_Teillast (float)`: Minimum partial load as a fraction of nominal load. Default is 0.2.
- `co2_factor_electricity (float)`: CO_2 factor for electricity consumption, in tCO_2/MWh. Default is 0.4 tCO_2/MWh.
- `primärenergiefaktor (float)`: Primary energy factor for electricity usage. Default is 2.4.

6.2 Methods

- `Geothermie(Last_L, VLT_L, COP_data, duration)`: Simulates the geothermal heat extraction process and calculates the heat output, electricity demand, and system performance metrics.
  - **Last_L (array-like)**: Load demand profile in kW.
  - **VLT_L (array-like)**: Flow temperatures in degrees Celsius.
  - **COP_data (array-like)**: Coefficient of performance (COP) data for interpolation.
  - **duration (float)**: Time step duration in hours.

  Returns the heat energy produced, electricity demand, heat output, and electric power output.

- `calculate(VLT_L, COP_data, Strompreis, q, r, T, BEW, stundensatz, duration, general_results)`: Calculates the economic and environmental metrics for the geothermal heat pump system.
  - **VLT_L (array-like)**: Flow temperatures in degrees Celsius.
  - **COP_data (array-like)**: COP data for performance calculation.
  - **Strompreis (float)**: Electricity price in €/MWh.
  - **q (float)**: Capital recovery factor.
  - **r (float)**: Price escalation factor.
  - **T (int)**: Consideration period in years.
  - **BEW (float)**: Discount rate for operational costs.
  - **stundensatz (float)**: Hourly labor rate in €/hour.
  - **duration (float)**: Duration of each time step in hours.
  - **general_results (dict)**: General results dictionary, including the load demand profile.

  Returns a dictionary containing heat output, electricity demand, weighted average cost of heat generation (WGK), specific CO_2 emissions, and primary energy consumption.

- `to_dict()`: Converts the object attributes into a dictionary for serialization.

- `from_dict(data)`: Creates an object from a dictionary of attributes.

6.3 Economic and Environmental Considerations

The `Geothermal` class calculates the weighted average cost of heat generation (WGK), which accounts for the costs of drilling, installation, operation, and electricity consumption. It also calculates specific CO_2 emissions based on electricity usage, as well as primary energy consumption using a primary energy factor.

6.4 Usage Example

The following example demonstrates how to initialize and use the `Geothermal` class to calculate the performance of a geothermal system:

```
geothermal_system = Geothermal(
    name="Geothermal Heat Pump",
    Fläche=500,  # m²
    Bohrtiefe=150,  # m
    Temperatur_Geothermie=10,  # °C
    spez_Bohrkosten=120,  # €/m
    spez_Entzugsleistung=55  # W/m
)
results = geothermal_system.calculate(
    VLT_L=temperature_profile,
    COP_data=cop_profile,
    Strompreis=100,  # €/MWh
    q=0.04, r=0.02, T=20,
    BEW=0.9,
    stundensatz=50,
    duration=1,
    general_results=load_data
)
```

This example demonstrates how to create a geothermal system with 500 m^2 area, 150 m drilling depth, and a geothermal source temperature of 10°C. The system's performance and economic metrics are calculated using the provided data.

7 WasteHeatPump Class

The `WasteHeatPump` class models a waste heat recovery heat pump system, inheriting from the `HeatPump` base class. It includes methods for simulating the performance of the heat pump and calculating various economic and environmental metrics based on waste heat recovery.

7.1 Attributes

- `Kühlleistung_Abwärme (float)`: Cooling capacity of the waste heat pump in kW.
- `Temperatur_Abwärme (float)`: Temperature of the waste heat source in degrees Celsius.
- `spez_Investitionskosten_Abwärme (float)`: Specific investment costs for the waste heat pump per kW. Default is 500 €/kW.

- `spezifische_Investitionskosten_WP (float)`: Specific investment costs of the heat pump per kW. Default is 1000 €/kW.
- `min_Teillast (float)`: Minimum partial load as a fraction of nominal load. Default is 0.2.
- `co2_factor_electricity (float)`: CO_2 factor for electricity consumption, in tCO_2/MWh. Default is 0.4 tCO_2/MWh.
- `primärenergiefaktor (float)`: Primary energy factor for electricity. Default is 2.4.

7.2 Methods

- `Berechnung_WP(VLT_L, COP_data)`: Calculates the heat load, electric power consumption, and adjusted flow temperatures for the waste heat pump.
  - **VLT_L (array-like)**: Flow temperatures in degrees Celsius.
  - **COP_data (array-like)**: Coefficient of performance (COP) data for interpolation.

  Returns the heat load and electric power consumption for the waste heat pump.

- `abwärme(Last_L, VLT_L, COP_data, duration)`: Calculates the waste heat and other performance metrics for the heat pump.
  - **Last_L (array-like)**: Load demand in kW.
  - **VLT_L (array-like)**: Flow temperatures in degrees Celsius.
  - **COP_data (array-like)**: COP data for performance calculation.
  - **duration (float)**: Duration of each time step in hours.

  Returns the heat energy produced, electricity demand, heat output, and electric power output.

- `calculate(VLT_L, COP_data, Strompreis, q, r, T, BEW, stundensatz, duration, general_results)`: Calculates the economic and environmental metrics for the waste heat pump system.
  - **VLT_L (array-like)**: Flow temperatures in degrees Celsius.
  - **COP_data (array-like)**: COP data for performance calculation.
  - **Strompreis (float)**: Price of electricity in €/MWh.
  - **q (float)**: Capital recovery factor.
  - **r (float)**: Price escalation factor.
  - **T (int)**: Consideration period in years.
  - **BEW (float)**: Discount rate for operational costs.

  - **stundensatz (float)**: Hourly labor rate in €/hour.
  - **duration (float)**: Time step duration in hours.
  - **general_results (dict)**: General results dictionary, including the load demand profile.

  Returns a dictionary of the calculated metrics, including heat output, electricity demand, CO_2 emissions, and primary energy consumption.

- `to_dict()`: Converts the object attributes into a dictionary for serialization.
- `from_dict(data)`: Creates an object from a dictionary of attributes.

7.3 Economic and Environmental Considerations

The `WasteHeatPump` class calculates the weighted average cost of heat generation (WGK) for the waste heat pump, which accounts for the costs of installation, operation, and electricity consumption. The class also computes specific CO_2 emissions based on electricity usage, as well as the primary energy consumption of the system.

7.4 Usage Example

The following example demonstrates how to initialize and use the `WasteHeatPump` class to calculate the performance of a waste heat recovery system:

```
waste_heat_pump = WasteHeatPump(
    name="Waste Heat Pump System",
    Kühlleistung_Abwärme=100,  # kW
    Temperatur_Abwärme=60  # °C
)
results = waste_heat_pump.calculate(
    VLT_L=temperature_profile,
    COP_data=cop_profile,
    Strompreis=150,  # €/MWh
    q=0.05, r=0.02, T=20,
    BEW=0.85,
    stundensatz=45,
    duration=1,
    general_results=load_profile
)
```

This example creates a waste heat recovery system with a cooling capacity of 100 kW and a waste heat source temperature of 60°C. The performance metrics and economic evaluations are calculated based on the input data.

8 RiverHeatPump Class

The `RiverHeatPump` class models a river water heat pump system, providing methods to calculate both performance and economic metrics. It extends the base `HeatPump` class.

8.1 Attributes

- `Wärmeleistung_FW_WP (float)`: Heat output of the river water heat pump.
- `Temperatur_FW_WP (float)`: Temperature of the river water.
- `dT (float)`: Temperature difference for operation. Default is 0.
- `spez_Investitionskosten_Flusswasser (float)`: Specific investment costs for river water heat pump in €/kW. Default is 1000 €/kW.
- `spezifische_Investitionskosten_WP (float)`: Specific investment costs for the heat pump in €/kW. Default is 1000 €/kW.
- `min_Teillast (float)`: Minimum partial load as a fraction of nominal load. Default is 0.2.
- `co2_factor_electricity (float)`: CO_2 emission factor for electricity in tCO_2/MWh. Default is 0.4.
- `primärenergiefaktor (float)`: Primary energy factor for electricity. Default is 2.4.

8.2 Methods

- `Berechnung_WP(Wärmeleistung_L, VLT_L, COP_data)`: Calculates the cooling load, electric power consumption, and adjusted flow temperatures.
  - **Wärmeleistung_L (array-like)**: Heat output load.
  - **VLT_L (array-like)**: Flow temperatures.
  - **COP_data (array-like)**: COP data for interpolation.

  Returns the cooling load, electric power consumption, and adjusted flow temperatures.

- `abwärme(Last_L, VLT_L, COP_data, duration)`: Calculates the waste heat and other performance metrics for the river heat pump.
  - **Last_L (array-like)**: Load demand in kW.
  - **VLT_L (array-like)**: Flow temperatures.
  - **COP_data (array-like)**: COP data for interpolation.
  - **duration (float)**: Duration of each time step in hours.

Returns the heat energy, electricity demand, heat output, electric power, cooling energy, and cooling load.

- `calculate(VLT_L, COP_data, Strompreis, q, r, T, BEW, stundensatz, duration, general_results)`: Calculates the economic and environmental metrics for the river heat pump.
  - **VLT_L (array-like)**: Flow temperatures.
  - **COP_data (array-like)**: COP data for interpolation.
  - **Strompreis (float)**: Price of electricity in €/MWh.
  - **q (float)**, **r (float)**, **T (int)**, **BEW (float)**, **stundensatz (float)**: Economic parameters.
  - **duration (float)**: Time duration for the simulation in hours.
  - **general_results (dict)**: Dictionary containing load profiles and other results.

  Returns a dictionary of calculated results, including economic and environmental metrics.
- `to_dict()`: Converts the object attributes to a dictionary for serialization.
- `from_dict(data)`: Creates an object from a dictionary of attributes.

8.3 Economic and Environmental Considerations

The `RiverHeatPump` class provides a method to calculate the **weighted average cost of heat generation (WGK)**, which takes into account investment costs, electricity consumption, and operational factors. The class also calculates the specific CO_2 emissions and primary energy usage for the heat pump.

8.4 Usage Example

The following example demonstrates how the `RiverHeatPump` class can be initialized and used to simulate the performance of a river water heat pump:

```
river_heat_pump = RiverHeatPump(
    name="Flusswärmepumpe",
    Wärmeleistung_FW_WP=300,  # kW
    Temperatur_FW_WP=12  # °C
)
results = river_heat_pump.calculate(
    VLT_L=temperature_forward,
    COP_data=cop_data,
    Strompreis=100,  # €/MWh
    q=0.03, r=0.02, T=20, BEW=0.8,
    stundensatz=50,
    duration=1,
    general_results=load_profile
)
```

In this example, a river water heat pump with a heat output of 300 kW and a river water temperature of 12°C is simulated. The performance metrics are calculated based on the provided data.

9 CHP Class

The `CHP` class models a Combined Heat and Power (CHP) system, allowing for the calculation of both performance metrics and economic evaluations. The system can operate with or without storage and can simulate gas or wood gas-powered CHPs.

9.1 Attributes

- `name (str)`: Name of the CHP system.
- `th_Leistung_BHKW (float)`: Thermal power of the CHP system in kW.
- `spez_Investitionskosten_GBHKW (float)`: Specific investment costs for gas CHP in €/kW. Default is 1500 €/kW.
- `spez_Investitionskosten_HBHKW (float)`: Specific investment costs for wood gas CHP in €/kW. Default is 1850 €/kW.
- `el_Wirkungsgrad (float)`: Electrical efficiency of the CHP system. Default is 0.33.
- `KWK_Wirkungsgrad (float)`: Combined heat and power efficiency. Default is 0.9.
- `min_Teillast (float)`: Minimum partial load as a fraction of nominal load. Default is 0.7.
- `speicher_aktiv (bool)`: Indicates if a storage system is active. Default is False.
- `Speicher_Volumen_BHKW (float)`: Storage volume in cubic meters. Default is 20 m^3.
- `T_vorlauf (float)`: Flow temperature in degrees Celsius. Default is 90°C.
- `T_ruecklauf (float)`: Return temperature in degrees Celsius. Default is 60°C.
- `initial_fill (float)`: Initial fill level of the storage. Default is 0.0.

- `min_fill (float)`: Minimum fill level of the storage. Default is 0.2.
- `max_fill (float)`: Maximum fill level of the storage. Default is 0.8.
- `spez_Investitionskosten_Speicher (float)`: Specific investment costs for storage in €/m^3. Default is 750 €/m^3.
- `BHKW_an (bool)`: Indicates if the CHP is on. Default is True.
- `thermischer_Wirkungsgrad (float)`: Thermal efficiency of the CHP system.
- `el_Leistung_Soll (float)`: Desired electrical power output of the CHP system in kW.
- `Nutzungsdauer (int)`: Lifespan of the CHP system in years. Default is 15 years.
- `f_Inst (float)`: Installation factor.
- `f_W_Insp (float)`: Inspection factor.
- `Bedienaufwand (float)`: Operational effort.
- `co2_factor_fuel (float)`: CO_2 emission factor for fuel in tCO_2/MWh.
- `primärenergiefaktor (float)`: Primary energy factor.
- `co2_factor_electricity (float)`: CO_2 emission factor for electricity in tCO_2/MWh. Default is 0.4 tCO_2/MWh.

9.2 Methods

- `BHKW(Last_L, duration)`: Calculates the power and heat output of the CHP system without storage.
  - **Last_L (array-like)**: Load demand in kW.
  - **duration (float)**: Duration of the time step in hours.

  Returns the heat output, electrical output, fuel consumption, and number of starts.
- `storage(Last_L, duration)`: Calculates the power and heat output of the CHP system with storage.
  - **Last_L (array-like)**: Load demand in kW.
  - **duration (float)**: Duration of the time step in hours.

  Returns the heat output, electrical output, storage status, and fuel consumption.

- `WGK(Wärmemenge, Strommenge, Brennstoffbedarf, Brennstoffkosten, Strompreis, q, r, T, BEW, stundensatz)`: Calculates the weighted average cost of energy for the CHP system.
  - **Wärmemenge (float)**: Total heat generated in MWh.
  - **Strommenge (float)**: Total electricity generated in MWh.
  - **Brennstoffbedarf (float)**: Fuel consumption in MWh.
  - **Brennstoffkosten (float)**: Fuel cost in €/MWh.
  - **Strompreis (float)**: Electricity price in €/MWh.
  - **q (float)**, **r (float)**: Factors for capital recovery and price escalation.
  - **T (int)**: Time period in years.
  - **BEW (float)**: Discount rate.
  - **stundensatz (float)**: Hourly rate for labor in €/hour.
- `calculate(Gaspreis, Holzpreis, Strompreis, q, r, T, BEW, stundensatz, duration, general_results)`: Simulates the CHP system's economic and environmental performance over a time period.
  - **Gaspreis (float)**: Gas price in €/MWh.
  - **Holzpreis (float)**: Wood price in €/MWh.
  - **Strompreis (float)**: Electricity price in €/MWh.
  - **q (float)**, **r (float)**, **T (int)**, **BEW (float)**, **stundensatz (float)**: Cost parameters.
  - **duration (float)**: Duration of the simulation in hours.
  - **general_results (dict)**: Dictionary containing load profiles and other metrics.
- `to_dict()`: Converts the object attributes into a dictionary for serialization.
- `from_dict(data)`: Creates an object from a dictionary of attributes.

9.3 Economic and Environmental Considerations

The `CHP` class calculates both the weighted average cost of heat generation (WGK) and the specific CO_2 emissions for a CHP system. These calculations consider fuel costs, electricity generation, and operational factors. The class also accounts for the CO_2 savings achieved through electricity generation and estimates the system's primary energy usage.

9.4 Usage Example

The following example demonstrates the initialization and use of the `CHP` class to simulate the performance of a gas-powered CHP system:

```
chp_system = CHP(
    name="Gas-BHKW",
    th_Leistung_BHKW=200,  # kW
    speicher_aktiv=True,
    Speicher_Volumen_BHKW=30  # m³
)
results = chp_system.calculate(
    Gaspreis=60,  # €/MWh
    Holzpreis=40,  # €/MWh
    Strompreis=100,  # €/MWh
    q=0.03, r=0.02, T=15, BEW=0.8,
    stundensatz=50,
    duration=1,
    general_results=load_profile
)
```

This example initializes a gas-powered CHP system with 200 kW thermal power and a 30 m^3 storage volume. The economic and environmental performance is calculated based on the provided inputs.

10 BiomassBoiler Class

The `BiomassBoiler` class models a biomass boiler system and includes methods for simulating the boiler's performance, fuel consumption, storage integration, and economic and environmental analysis.

10.1 Attributes

- `name (str)`: Name of the biomass boiler system.
- `P_BMK (float)`: Boiler power in kW.
- `Größe_Holzlager (float)`: Size of the wood storage in cubic meters.
- `spez_Investitionskosten (float)`: Specific investment costs for the boiler in €/kW.
- `spez_Investitionskosten_Holzlager (float)`: Specific investment costs for wood storage in €/m^3.
- `Nutzungsgrad_BMK (float)`: Efficiency of the biomass boiler.
- `min_Teillast (float)`: Minimum part-load operation as a fraction of the nominal load.

- `speicher_aktiv (bool)`: Indicates whether a storage system is active.
- `Speicher_Volumen (float)`: Volume of the thermal storage in cubic meters.
- `T_vorlauf (float)`: Supply temperature in degrees Celsius.
- `T_ruecklauf (float)`: Return temperature in degrees Celsius.
- `initial_fill (float)`: Initial fill level of the storage as a fraction of total volume.
- `min_fill (float)`: Minimum fill level of the storage as a fraction of total volume.
- `max_fill (float)`: Maximum fill level of the storage as a fraction of total volume.
- `spez_Investitionskosten_Speicher (float)`: Specific investment costs for the thermal storage in €/m^3.
- `BMK_an (bool)`: Indicates whether the boiler is on.
- `opt_BMK_min (float)`: Minimum boiler capacity for optimization.
- `opt_BMK_max (float)`: Maximum boiler capacity for optimization.
- `opt_Speicher_min (float)`: Minimum storage capacity for optimization.
- `opt_Speicher_max (float)`: Maximum storage capacity for optimization.
- `Nutzungsdauer (int)`: Service life of the biomass boiler in years.
- `f_Inst (float)`: Installation factor.
- `f_W_Insp (float)`: Maintenance and inspection factor.
- `Bedienaufwand (float)`: Operational effort for the system.
- `co2_factor_fuel (float)`: CO_2 factor for the fuel in tCO_2/MWh.
- `primärenergiefaktor (float)`: Primary energy factor for the fuel.

10.2 Methods

- `Biomassekessel(Last_L, duration)`: Simulates the operation of the biomass boiler over a given load profile and duration.
  - **Last_L (array)**: Load profile of the system in kW.
  - **duration (float)**: Duration of each time step in hours.
- `storage(Last_L, duration)`: Simulates the operation of the storage system, adjusting boiler output to optimize storage usage.

  - **Last_L (array)**: Load profile of the system in kW.
  - **duration (float)**: Duration of each time step in hours.

- `WGK(Wärmemenge, Brennstoffbedarf, Brennstoffkosten, q, r, T, BEW, stundensatz)`: Calculates the weighted average cost of heat generation (WGK) based on the system's investment costs, fuel costs, and operating costs.
  - **Wärmemenge (float)**: Amount of heat generated in kWh.
  - **Brennstoffbedarf (float)**: Fuel consumption in MWh.
  - **Brennstoffkosten (float)**: Cost of the biomass fuel in €/MWh.
  - **q (float)**: Factor for capital recovery.
  - **r (float)**: Price escalation factor.
  - **T (int)**: Time period in years for the calculation.
  - **BEW (float)**: Operational cost factor.
  - **stundensatz (float)**: Hourly labor rate for operational efforts.

- `calculate(Holzpreis, q, r, T, BEW, stundensatz, duration, general_results)`: Simulates the performance of the biomass boiler, calculating both heat generation and economic parameters.
  - **Holzpreis (float)**: Price of wood fuel in €/MWh.
  - **q (float)**: Factor for capital recovery.
  - **r (float)**: Price escalation factor.
  - **T (int)**: Time period in years for the calculation.
  - **BEW (float)**: Operational cost factor.
  - **stundensatz (float)**: Hourly labor rate for operational efforts.
  - **duration (float)**: Duration of each time step in hours.
  - **general_results (dict)**: A dictionary containing general results from the simulation, such as remaining loads.

  Returns a dictionary with key results such as fuel consumption, heat output, specific CO_2 emissions, and primary energy usage.

- `to_dict()`: Converts the `BiomassBoiler` object to a dictionary for serialization and storage.

- `from_dict(data)`: Initializes a `BiomassBoiler` object from a dictionary.

10.3 Economic and Environmental Considerations

The `BiomassBoiler` class includes methods to calculate the system's **weighted average cost of heat generation (WGK)**. This takes into account the investment, installation, operational costs, and fuel consumption. The system's specific CO_2 emissions are calculated based on the amount of fuel burned, and its **primary energy consumption** is calculated based on the heat output and the primary energy factor.

10.4 Usage Example

The `BiomassBoiler` class can be used to simulate the performance of a biomass heating system with or without a storage unit. Below is an example of how to initialize and use the class:

```
biomass_boiler = BiomassBoiler(
    name="Biomassekessel",
    P_BMK=500,  # kW
    Größe_Holzlager=50,  # m³
    Nutzungsgrad_BMK=0.85,
    Speicher_Volumen=100,  # m³
    speicher_aktiv=True
)
results = biomass_boiler.calculate(
    Holzpreis=20,  # €/MWh
    q=0.05, r=0.03, T=15, BEW=1.1,
    stundensatz=50,
    duration=1,
    general_results=load_profile
)
```

In this example, a biomass boiler with a power rating of 500 kW and a wood storage volume of 50 m^3 is simulated. The system includes a 100 m^3 thermal storage unit. Performance and cost metrics are calculated based on the provided inputs.

11 GasBoiler Class

The `GasBoiler` class represents a gas boiler system, designed to calculate and simulate the performance, cost, and emissions of a gas boiler in a heating system. The class incorporates key economic, operational, and environmental factors and can be used for comprehensive analysis in energy systems.

11.1 Attributes

- `name (str)`: Name of the gas boiler system.
- `spez_Investitionskosten (float)`: Specific investment costs for the gas boiler in €/kW.
- `Nutzungsgrad (float)`: Efficiency of the gas boiler, typically ranging between 0.8 and 1.0. It represents the ratio of useful heat output to the total energy input.
- `Faktor_Dimensionierung (float)`: Dimensioning factor to account for capacity oversizing.

- `Nutzungsdauer (int)`: Lifespan of the gas boiler in years. Defaults to 20 years.
- `f_Inst (float)`: Installation factor, representing additional costs due to installation complexities.
- `f_W_Insp (float)`: Inspection factor, accounting for periodic maintenance and inspection costs.
- `Bedienaufwand (float)`: Operational effort, representing the cost of operation in terms of labor.
- `co2_factor_fuel (float)`: CO_2 emission factor for the fuel (natural gas), typically measured in tCO_2/MWh.
- `primärenergiefaktor (float)`: Primary energy factor for the fuel, representing the amount of primary energy required to produce one unit of usable energy (MWh). This factor accounts for energy losses in the fuel supply chain.

11.2 Methods

- `GasBoiler(Last_L, duration)`: Simulates the operation of the gas boiler based on the load profile and duration of operation.
  - **Last_L (array)**: Array representing the heat load in kW that the gas boiler needs to meet.
  - **duration (float)**: Duration of each time step in hours for which the load is simulated.

  This method calculates the total heat output (`Wärmemenge_Gaskessel`) and gas demand (`Gasbedarf`) based on the efficiency (`Nutzungsgrad`) and the given load.

- `WGK(Brennstoffkosten, q, r, T, BEW, stundensatz)`: Calculates the weighted average cost of heat generation (**WGK**), factoring in investment costs, fuel costs, and operational expenses.
  - **Brennstoffkosten (float)**: The cost of fuel (natural gas) in €/MWh.
  - **q (float)**: Capital recovery factor.
  - **r (float)**: Factor accounting for price escalation or depreciation.
  - **T (int)**: Time period in years for the cost calculation.
  - **BEW (float)**: Subsidy or other cost-reducing factors.
  - **stundensatz (float)**: Hourly labor rate for the operation of the gas boiler.

  The method calculates total investment costs, operational costs, and returns the specific heat generation cost (`WGK_GK`).

- `calculate(Gaspreis, q, r, T, BEW, stundensatz, duration, Last_L, general_results)`: Performs a complete calculation of the gas boiler's performance and economic metrics. It simulates the boiler's operation over a given time period and computes the weighted average cost of heat generation, CO_2 emissions, and primary energy usage.

  - **Gaspreis (float)**: Price of natural gas in €/MWh.
  - **q (float)**, **r (float)**, **T (int)**, **BEW (float)**: Parameters for cost and subsidy calculation.
  - **stundensatz (float)**: Labor rate for boiler operation in €/hour.
  - **duration (float)**: Duration of each simulation time step in hours.
  - **Last_L (array)**: Load profile of the system in kW.
  - **general_results (dict)**: Dictionary containing general results such as residual load.

  This method computes:

  - `Wärmemenge`: Total heat output of the gas boiler.
  - `Wärmeleistung_L`: Time-resolved heat output.
  - `Brennstoffbedarf`: Total gas demand based on the heat output and efficiency.
  - `WGK`: Weighted average cost of heat generation.
  - `spec_co2_total`: Specific CO_2 emissions in tCO_2/MWh heat.
  - `primärenergie`: Primary energy usage of the system.

  Returns a dictionary of the results, including performance metrics, cost analysis, and emissions data.

- `to_dict()`: Converts the `GasBoiler` object to a dictionary for serialization or storage. This method is useful for saving the configuration or results of the gas boiler system.

- `from_dict(data)`: Creates a `GasBoiler` object from a dictionary containing the system's attributes. This method is essential for re-loading configurations from a stored state.

11.3 Economic and Environmental Considerations

The `GasBoiler` class is designed to simulate both the economic and environmental impacts of a gas boiler system. The **weighted average cost of heat generation (WGK)** takes into account both investment costs and operational costs, including fuel prices, labor, and maintenance. Additionally, the system's **CO_2 emissions** are calculated based on fuel consumption and the specific CO_2 factor for natural gas, allowing for an analysis of the environmental footprint of the system. The **primary energy consumption** is also calculated, offering insights into the system's overall energy efficiency and sustainability.

11.4 Usage Example

The following is an example of how the GasBoiler class can be initialized and used:

```
gas_boiler = GasBoiler(
    name="Gasheizkessel",
    spez_Investitionskosten=35,  # €/kW
    Nutzungsgrad=0.92,  # 92% efficiency
    Faktor_Dimensionierung=1.1  # Slight oversizing
)

results = gas_boiler.calculate(
    Gaspreis=30,  # €/MWh
    q=0.03, r=0.02, T=20, BEW=1,
    stundensatz=50,
    duration=1,
    Last_L=load_profile,
    general_results={'Restlast_L': residual_load}
)
```

In this example, the gas boiler is dimensioned to have an efficiency of 92% and is slightly oversized. The calculation method estimates the heat output, gas demand, CO_2 emissions, and the weighted average cost of heat generation based on a load profile and general system parameters.

12 SolarThermal Class

The SolarThermal class models a solar thermal system and includes methods for performance, economic, and environmental calculations. The class can handle various types of solar collectors (e.g., flat plate and vacuum tube collectors) and includes parameters for storage system integration.

12.1 Attributes

- name (str): Name of the solar thermal system.
- bruttofläche_STA (float): Gross collector area of the solar thermal system in square meters.
- vs (float): Volume of the storage system in cubic meters.
- Typ (str): Type of solar collector, e.g., "Flachkollektor" or "Vakuumröhrenkollektor".
- kosten_speicher_spez (float): Specific costs for the storage system in €/m^3.
- kosten_fk_spez (float): Specific costs for flat plate collectors in €/m^2.

- `kosten_vrk_spez (float)`: Specific costs for vacuum tube collectors in €/m².
- `Tsmax (float)`: Maximum storage temperature in degrees Celsius.
- `Longitude (float)`: Longitude of the installation site.
- `STD_Longitude (float)`: Standard longitude for the time zone.
- `Latitude (float)`: Latitude of the installation site.
- `East_West_collector_azimuth_angle (float)`: Azimuth angle of the solar collector in degrees.
- `Collector_tilt_angle (float)`: Tilt angle of the solar collector in degrees.
- `Tm_rl (float)`: Mean return temperature in degrees Celsius.
- `Qsa (float)`: Initial heat output.
- `Vorwärmung_K (float)`: Preheating in Kelvin.
- `DT_WT_Solar_K (float)`: Temperature difference across the solar heat exchanger in Kelvin.
- `DT_WT_Netz_K (float)`: Temperature difference across the network heat exchanger in Kelvin.
- `opt_volume_min (float)`: Minimum optimization volume in cubic meters.
- `opt_volume_max (float)`: Maximum optimization volume in cubic meters.
- `opt_area_min (float)`: Minimum optimization area in square meters.
- `opt_area_max (float)`: Maximum optimization area in square meters.
- `kosten_pro_typ (dict)`: Dictionary containing the specific costs for different types of solar collectors.
- `Kosten_STA_spez (float)`: Specific costs for the solar thermal system in €/m².
- `Nutzungsdauer (int)`: Service life of the solar thermal system in years (20 years by default).
- `f_Inst (float)`: Installation factor.
- `f_W_Insp (float)`: Maintenance and inspection factor.
- `Bedienaufwand (float)`: Operating effort for the system.

- `Anteil_Förderung_BEW (float)`: Subsidy rate for renewable energy law compliance.
- `Betriebskostenförderung_BEW (float)`: Operational cost subsidy per MWh of thermal energy under the renewable energy law.
- `co2_factor_solar (float)`: CO_2 factor for solar energy (typically 0 for solar heat).
- `primärenergiefaktor (float)`: Primary energy factor (typically 0 for solar thermal energy).

12.2 Methods

- `calc_WGK(q, r, T, BEW, stundensatz)`: Calculates the weighted average cost of heat generation (WGK) based on the system's investment costs, operational costs, and subsidy status.
  - **q (float)**: Factor for capital recovery.
  - **r (float)**: Price escalation factor.
  - **T (int)**: Time period in years for the calculation.
  - **BEW (str)**: Indicates eligibility for renewable energy subsidies ("Ja" or "Nein").
  - **stundensatz (float)**: Hourly labor rate for operational efforts.

  Returns the WGK of the system based on total investments, subsidies, and operating costs.

- `calculate(VLT_L, RLT_L, TRY, time_steps, calc1, calc2, q, r, T, BEW, stundensatz, duration, general_results)`: Simulates the solar thermal system's performance over a time period, taking into account forward and return temperatures, weather data, and operational costs.
  - **VLT_L (array)**: Array of forward temperatures in degrees Celsius.
  - **RLT_L (array)**: Array of return temperatures in degrees Celsius.
  - **TRY (array)**: Test reference year weather data.
  - **time_steps (array)**: Array of time steps for the simulation.
  - **calc1 (float)**, **calc2 (float)**: Additional calculation parameters.
  - **q (float)**, **r (float)**, **T (int)**, **BEW (str)**, **stundensatz (float)**: Parameters for cost calculation.
  - **duration (float)**: Duration of each simulation time step.
  - **general_results (dict)**: A dictionary containing general results from the simulation, such as remaining loads.

  Returns a dictionary of simulation results, including the heat output, specific CO_2 emissions, primary energy usage, and storage state.

- `to_dict()`: Converts the `SolarThermal` object to a dictionary for easy serialization and storage.
- `from_dict(data)`: Creates a `SolarThermal` object from a dictionary of attributes.

12.3 Economic and Environmental Considerations

The `SolarThermal` class includes methods to calculate the system's **weighted average cost of heat generation (WGK)**, which takes into account installation costs, operational costs, and subsidies under the renewable energy law. The system's specific CO_2 emissions are calculated as the amount of emissions per unit of heat generated, and the system's **primary energy usage** is computed based on its heat output.

12.4 Usage Example

This class is highly adaptable for different solar thermal setups. The following example demonstrates how the class can be initialized and used:

```
solar_system = SolarThermal(
    name="SolarThermie-Anlage",
    bruttofläche_STA=500,  # m²
    vs=50,  # m³ storage
    Typ="Flachkollektor",
    Tsmax=90,
    Longitude=-14.42,
    STD_Longitude=-15,
    Latitude=51.17,
    East_West_collector_azimuth_angle=0,
    Collector_tilt_angle=36
)
results = solar_system.calculate(
    VLT_L=temperature_forward,
    RLT_L=temperature_return,
    TRY=weather_data,
    time_steps=steps,
    calc1=0.8, calc2=1.2,
    q=0.03, r=0.02, T=20, BEW="Ja",
    stundensatz=50,
    duration=1,
    general_results=load_profile
)
```

This example shows a solar thermal system with flat plate collectors covering 500 m^2 and a storage volume of 50 m^3. Performance and cost metrics are calculated based on the provided inputs.

13 Conclusion

This document provides a comprehensive overview of the models and optimization strategies used for heating systems. Future work will extend the system to cover additional environmental factors and economic uncertainties.