TECHNISCHER BERICHT

Gebäudemodellierung

für das EnOB-Verbundvorhaben

BUiLD.DIGITiZED

IoT und BIM für die Inbetriebnahme und den Betrieb
von netzdienlichen Niedrigstenergiegebäuden

1.7.2020 – 30.6.2023

MONITORING.digital
FKZ: 03EN1021A

Hochschule Offenburg | Institut für Energiesystemtechnik
Badstraße 24 | 77652 Offenburg

INBETRIEBNAHME.digital
FKZ: 03EN1021B

Fraunhofer-Institut für Solare Energiesysteme ISE
Heidenhofstraße 2 | 79110 Freiburg

EVALUATION.digital
FKZ: 03EN1021D

Mondas GmbH
Emmy-Noether-Str. 2 | 79110 Freiburg

Luftqualität
FKZ: 03EN1021F

Testo SE & Co. KGaA
Celsiusstraße 2 | 79822 Titisee-Neustadt

Anlagenplanung mit BIM
FKZ: 03EN1021C

Konzmann Gebäudetechnik GmbH
Niederwiesenstraße 34 | 78050 VS-Villingen

Betriebsführung mit BIM
FKZ: 03EN1021E

Maurer Energie- und Ingenieurleistungen GmbH & Co. KG
Dr.-Kurt-Steim-Straße 7 | 78713 Schramberg-Sulgen

Autor dieses Berichtes:

Danny Carvajal, M.Sc.

Hochschule Offenburg | Institut für Energiesystemtechnik
Badstraße 24 | 77652 Offenburg

danny.carvajal@hs-offenburg.de

Offenburg, 14. Februar 2023

Kurzfassung

Ein digitaler Zwilling ist in der Gebäudetechnik ein komplettes, digitales Abbild eines Gebäudes (mit seiner gesamten Anlagentechnik). Im Rahmen des Projekts BUiLD.DIGITiZED gibt es zwei digitale Zwillinge des *RIZ Energie*:

- BIM-Modell (auch „statischer digitaler Zwilling“), das aus allen unveränderlichen Informationen des Gebäudes und der technischen Gebäudeausrüstung besteht, wie z. B. den Wärmedurchgangskoeffizienten (U-Werte) der Gebäudehülle oder der Leistungszahl (COP) der Wärmepumpe
- Gebäude- und Anlagensimulation (auch „dynamischer digitaler Zwilling“), die aus Grey-Box-Modellen besteht, mit dem BIM-Modell parametriert wird und unter Verwendung von Input-Daten aus dem Monitoring für die Betriebsoptimierung des Gebäudes verwendet werden kann. Dazu werden die Simulationsergebnisse mit Monitoring-Daten verglichen.

Dieser Bericht konzentriert sich auf den „dynamischen digitalen Zwilling“ auf Basis einer gekoppelten Gebäude- und Anlagensimulation.

1 Digitaler Zwilling zur betriebsbegleitenden Optimierung

Abbildung 1 zeigt den Entwicklungsprozess des digitalen Zwillings in zwei Stufen:

1. Simulationsmodell „Design“ nach ISO 13790 zur Parametrierung aus dem BIM-Modell sowie zur Validierung (in Kombination mit dem zweiten Schritt). Die Simulation mit dem Testreferenzjahr für Offenburg liefert den simulierten Nutzenergiebedarf für Heizung und Kühlung. Im Vergleich zum Nutzenergiebedarf nach EN 12831 aus dem GEG-Nachweis gelingt eine erste Parametrierung aus dem Abgleich mit Planungsdaten.
2. Simulationsmodell „Operation“ mit Kopplung von bereits parametriertem Gebäudemodell und Anlagenmodellen (insb. Lüftungsanlage, Wärmetauscher, Wärmepumpe, Wärmespeicher, Bauteilaktivierung und Grundwasserpumpen) zur Berechnung des aktuellen Innenraumklimas und Endenergie*bedarfs* (Strom). Im Vergleich mit dem gemessenen Innenraumklima und Endenergie*verbrauch* kann das Modell zunächst validiert, dann trainiert und danach adaptiert werden, um sowohl eine Fehleranalyse als auch eine betriebsbegleitende Optimierung realisieren zu können. Hier werden die aktuellen Wetterdaten, Grundwassertemperaturdaten und Präsenzdaten als Inputdaten genutzt.

Der digitale Zwilling dient damit zunächst für eine BIM-gestützte Inbetriebnahme und kann dann für die betriebsbegleitende Optimierung oder automatische Fehlererkennung genutzt werden.

Hinweis: Im Rahmen von BUiLD.DIGITiZED wird dieses Modell im zweiten Projektjahr für den netzdienlichen Betrieb und im dritten Projektjahr für die Einbindung in ein Energienetz weiterentwickelt und genutzt.

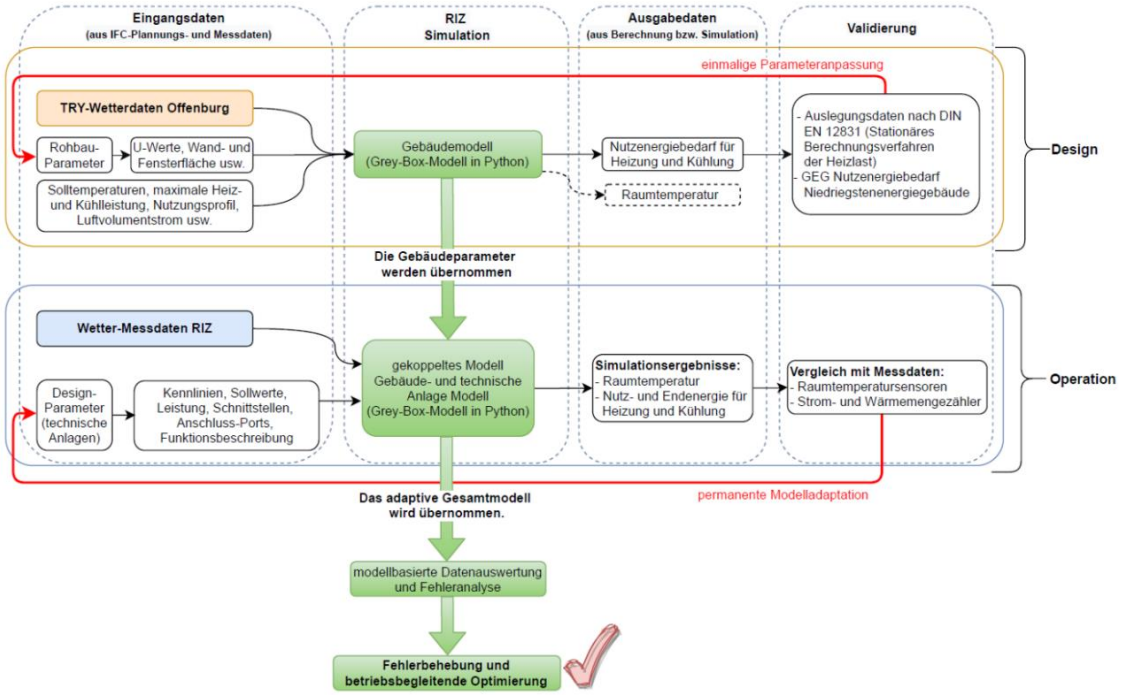

Abbildung 1 Vereinfachtes Datenflussdiagramm für die Entwicklung, Implementierung und Anwendung des digitalen Zwillings, hier nur für die gekoppelte Gebäude- und Anlagensimulation.

2 Simulationsverfahren

Das Grey-Box-Modell wurde als signalflussbasierter Simulationsprozess in Python programmiert:

INPUT → Gebäudemodell(e) → MSR-Modul → Anlagenmodell(e) → OUTPUT

Als Eingangsgrößen sollen ausschließlich Wetter- und ggf. einzelne Nutzungsdaten verwendet werden. So arbeitet das System im Januar beispielsweise im Heizbetrieb und nutzt die Wärmepumpe, während das MSR-Modul im Juli in den Kühlbetrieb über die Grundwasserbrunnen wechselt. Dabei wird ein gerichteter Datenfluss von Modell zu Modell realisiert. Eine numerische Iteration ist also nicht möglich und wird ggf. durch entsprechende Annahmen (insb. Ping-pong- statt Onion-Prinzip) umgesetzt.

Abbildung 2 zeigt einen Ausschnitt aus dem Heizsystem. Hier sind drei der Hauptkomponenten zu sehen: Wärmetauscher, Wärmepumpe und Wärmespeicher. Hier wird beispielsweise am Wärmetauscher die Ausgangstemperatur der Sekundärseite berechnet und wird als Eingangstemperatur des Verdampfers der Wärmepumpe genutzt. Die berechnete Wärmeleistung der Wärmepumpe wird dann als Eingangsleistung in den Wärmespeicher genutzt. Diese Energiewandlung wird von Komponenten zu Komponente fortgeführt, wobei keine einheitlichen Übergabeprotokolle vereinbart werden müssen und auch nicht genutzt werden.

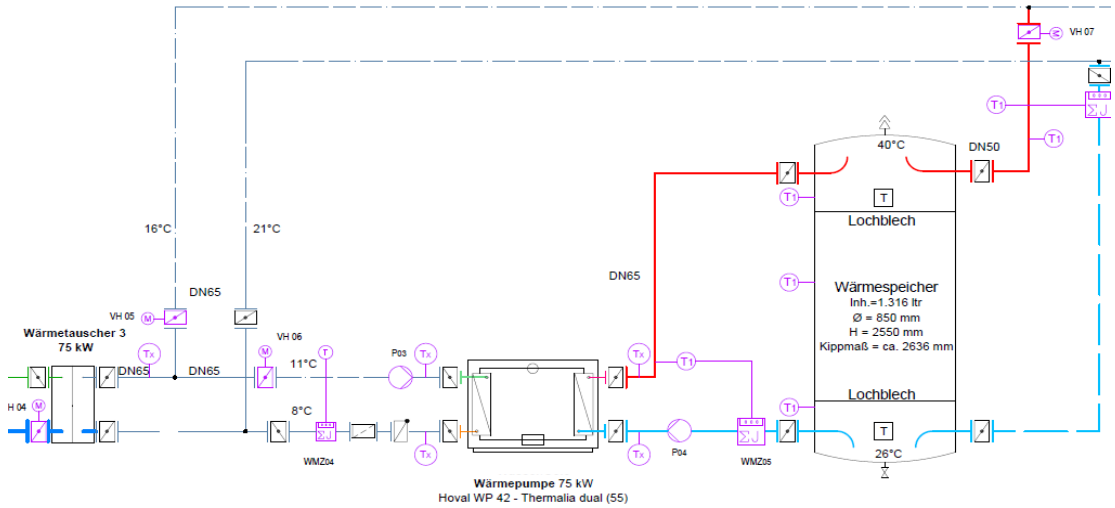

Abbildung 2 Schema der Heizungsanlage am RIZ Energie (Ausschnitt)

Abbildung 3 zeigt in einem Datenflussdiagramm wie alle Modelle in Bezug auf Eingabe- und Ausgabedaten miteinander verbunden sind. In Gelb sind alle gemessenen Eingangsdaten, in orange alle Modelle, in grau die Kontrollmodule und in Rot die Simulationsergebnisse.

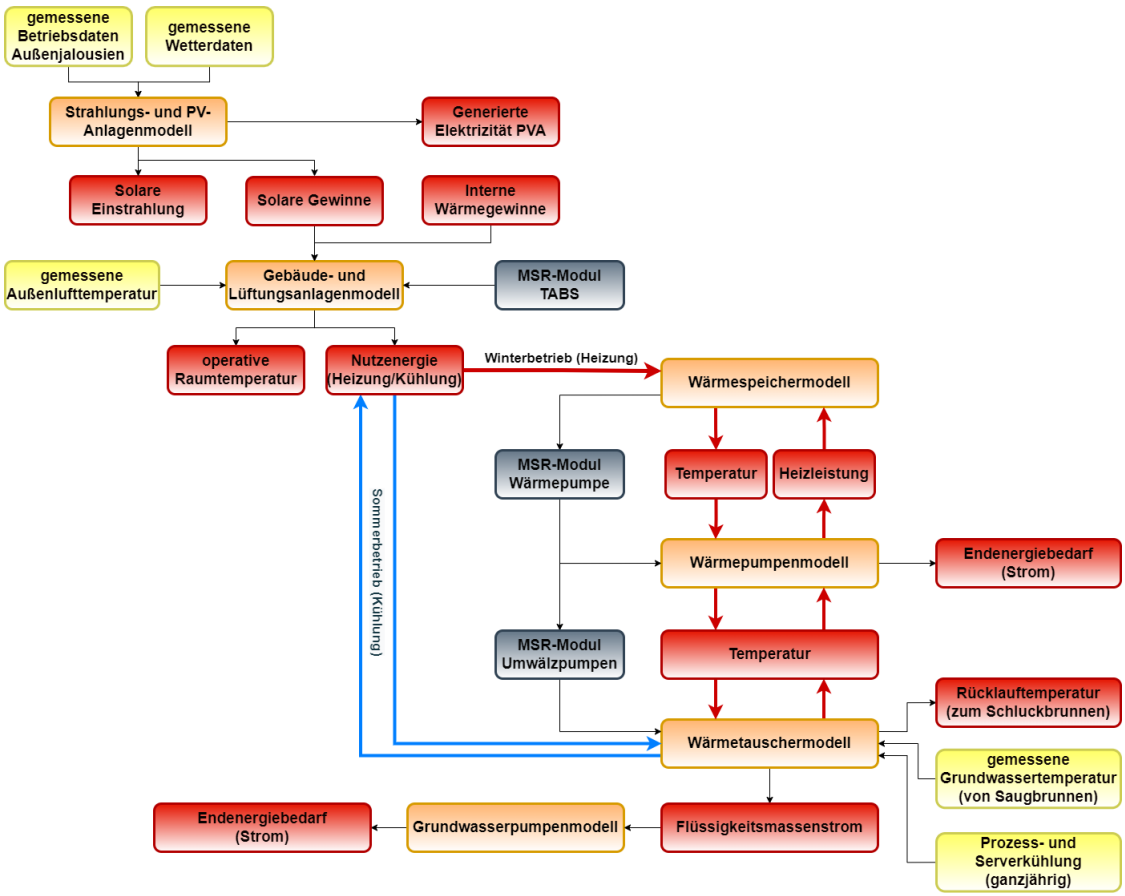

Abbildung 3 Datenflussdiagramm der Python-Modelle

Die Simulation enthält alle fünf Zonen des Gebäudes (ohne interzonalen Wärmestrom), siehe Abbildung 4:

- Zone 1 (blau) = Technikum, dessen Raumhöhe sich über drei Geschosse erstreckt
- Zone 2 (grün) = Lagerräume und Werkstätten im Erdgeschoss.
- Zone 3 (gelb) und Zone 4 (violett) = Büroebenen im 1. und 2. Obergeschoss, mit gleichen Parametern und Eigenschaften
- Zone 5 (rot) = Büroebene (3. Obergeschoss) mit Dach und Außenwand zum Außenlabor

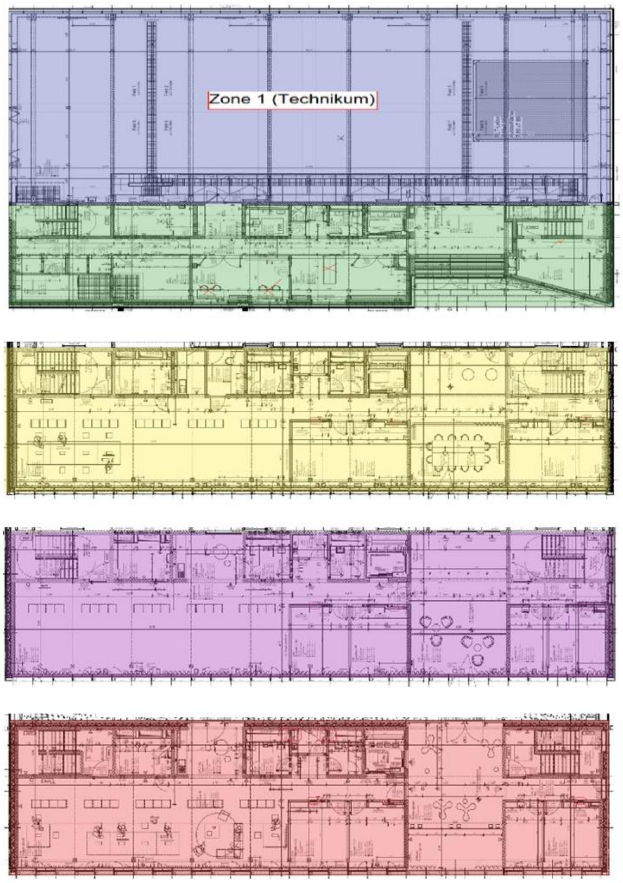

Abbildung 4 Von oben nach unten: Zone 1 (Technikum), Zone 2 (Erdgeschoss), Zone 3 (1. Obergeschoss), Zone 4 (2. Obergeschoss) und Zone 5 (3. Obergeschoss, ohne Technikzentrale).

3 Simulationsmodelle „Operation“

Alle Simulationsmodelle sind gleichungsbasierte, in Python programmierte Grey-Box-Modelle und werden aus dem BIM-Modell heraus parametriert. Nachfolgend werden die Modelle der wichtigsten technischen Gebäudeausrüstung (TGA) und des Gebäudes kurz erläutert, dabei werden mitunter Ausschnitte aus den Python-Skripten zur Veranschaulichung verwendet. Die Modelle werden ständig weiterentwickelt, um das reale Verhalten der TGA besser abbilden zu können.

3.1 Strahlungs- und Photovoltaikanlagenmodell

Die solare Einstrahlung und die dadurch resultierenden solaren Wärmegewinne werden durch den Sonnenstand bestimmt. Der Sonnenstand ist abhängig von Tag und Jahreszeit und wird, wie in Abbildung 5 zu sehen, von der sich ändernden Sonnenhöhe (γ_s) und dem Sonnenazimut (α_s) beschrieben. Basis der Sonnenhöhe und des Sonnenazimutes ist der Standort des zu berechnenden Bauwerkes, angegeben in geografischem Breitengrad und geografischem Längengrad [1], [3].

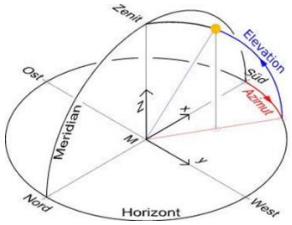

Abbildung 5 Sonnenverlauf und deren Richtgrößen (Quelle: Astronomische Koordinatensysteme, biancahoegel.de)

3.1.1 Gesamtstrahlung auf eine geneigte Oberfläche mit einer bestimmten Ausrichtung

Für die Berechnung der Gesamtstrahlung auf eine geneigte Oberfläche (*I_ges*) werden *I_bT* (Direktstrahlung), *I_gT* (Strahlung durch Bodenreflektion) und *I_dT* (Diffusstrahlung) benötigt. Diese werden in Python im Funktionsaufruf *ItotSur* berechnet. Die Berechnungen sind von nachfolgenden Parametern abhängig:

- Orientierung (*sur_a*)
- Neigung der betrachteten Flächen (*slope*)
- Gemessene Sonnenhöhe (γ_s = *sun_h*)
- Gemessenes Sonnenazimut (α_s = *sun_a*)
- Zenithwinkel ($Cos(\theta_z)$ = *CosTetha*)
- Gemessene direkte Normalstrahlung auf horizontaler Oberfläche (*I_dir*)
- Gemessene Globalstrahlung auf horizontaler Oberfläche (*I_glob*)

Mit den beiden letztgenannten Werten kann die Diffusstrahlung auf horizontaler Oberfläche berechnet werden (*I_diff* = *I_glob* - *I_dir*).

Für die Berechnung von *I_bT*, werden die direkte Strahlung auf horizontale Oberfläche (*I_dir*) und der Einstrahlwinkel (ζ = *CosTheta*) miteinander multipliziert. Eine Kondition gibt die Anweisung, dass *I_dir* nur dann berechnet wird, wenn *CosTheta* oder *sun_h* größer 0 sind. Ansonsten wird *I_dir* gleich 0 gesetzt. Eine weitere, anschließende Kondition stellt sicher, dass *I_bT* im Falle eines negativen Berechnungswertes gleich 0 gesetzt wird. Die im Anschluss angegebene Funktion *min* vermeidet, dass *I_bT* den maximalen Einstrahlwert *I_extraTer* übersteigt.

```
# Irradiance Walls
for i in range(Range):

    def ItotSur(
        I_dir, I_dif, I_Globalhor, sun_h, sun_a, slope, sur_a):

        CosTheta = sin(rad(sun_h)) * cos(rad(slope)) + cos(rad(sun_h)) \
                   * sin(rad(slope)) * cos(rad(abs(sur_a-sun_a)))

        if CosTheta < 0 or sun_h <= 0:
            Ibeam = I_bT = 0
        else:
            Ibeam = (I_dir/sin(rad(sun_h))) # [W/m²]
            Ibeam = min(Ibeam, Pdir_norm)
            I_bT = Ibeam * CosTheta # [W/m²]

        # Uniform diffuse sky model (Liu and Jordan's Model)
        I_dT = I_dif * 0.5 * (1+cos(rad(slope)))
        if I_dT < 0:
            I_dT = 0

        # Hourly ground reflected Radiation on an Angled Surface
        I_gT = I_Globalhor * params_ground_Refl * 0.5 * (1-cos(rad(slope)))

        # Global solar radiation incident on an inclined surface
        I_ges = I_bT + I_gT + I_dT

        return(I_ges, I_bT, I_gT, I_dT, Ibeam) # [W/m²]
```

Umrechnung der direkten Normalstrahlung auf horizontale Flächen (I_{dir}) in direkte Strahlung auf horizontale Flächen (I_{beam}) in W/m² entsprechend Abbildung 6

- $I_{beam} = \frac{I_{dir}}{\sin \gamma_s}$, da $\sin \gamma_s = \frac{I_{dir}}{I_{beam}}$

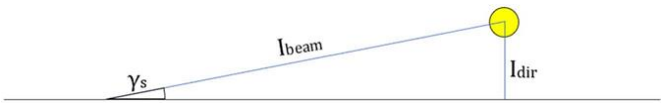

Abbildung 6 Schematische Darstellung I_{beam}

Berechnung des Einstrahlwinkels (ζ) auf eine beliebig orientierte Fläche

- $\cos \zeta = \sin \gamma_s \cdot \cos \gamma_F + \cos \gamma_s \cdot \sin \gamma_F \cdot \cos(abs(\alpha_F - \alpha_s))$

Festlegung der Orientierung der betrachteten Fläche für die Himmelsrichtung:

α_F = 0° ≙ Norden; α_F = 90° ≙ Osten; α_F = 180° ≙ Süden; α_F = 270° ≙ Westen

Festlegung der Neigung der betrachteten Fläche für die Himmelsrichtung:

γ_F = 0° ≙ waagerecht oben; γ_F = 90° ≙ senkrecht; γ_F = 180° ≙ waagerecht nach unten

Berechnung der Direktstrahlung auf eine geneigte Oberfläche entsprechend Abbildung 7

- $I_{bT} = \cos\zeta \cdot I_{beam}$, da $\cos\zeta = \frac{I_{bT}}{I_{beam}}$

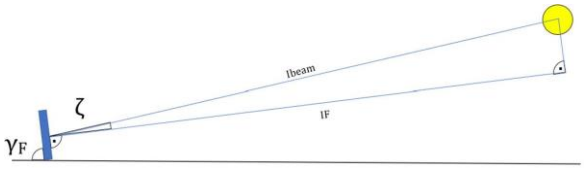

Abbildung 7 Schematische Darstellung des Einstrahlwinkels

Berechnung der Strahlungsbestandteil I_{gT} in W/m² durch Bodenreflexion (Reflexion der Umgebung)

- $I_{gT} = (I_{dif} + I_{dir}) \cdot 0{,}5 \cdot \rho_{gT} \cdot (1 - \cos\gamma_F)$.

Dabei ist die Albedo ρ_{gT} = 0,2 (VDI 6007, Kapitel 7.3)

Berechnung der Anteil der diffusen Horizontalstrahlung auf eine geneigte Oberfläche I_{dT} in W/m²

- $I_{dT} = I_{dif} \cdot 0{,}5 \cdot (1 + \cos\gamma_F)$

Gesamtstrahlung (Globalstrahlung) auf eine geneigte Oberfläche I_{ges} in W/m²

- $I_{ges} = I_{bT} + I_{gT} + I_{dT}$

Für die Berechnung der Gesamtstrahlung auf die PV-Anlage auf dem Dach des Gebäudes wird dieselbe Herangehensweise verwendet. Einziger Unterschied ist das *I_ges* hier lediglich aus der Summe von *I_bT* (Direktstrahlung) und *I_dT* (Diffusstrahlung) besteht und *I_gT* somit hier nicht berechnet wird.

3.1.2 Beschattungsanlagenmodell (Außenjalousien)

Um zu verhindern, dass zu viel Strahlung in das Gebäude eindringt, ist im ersten, zweiten und dritten Stock eine Beschattungsanlage (Außenjalousien) installiert, die bei Überschreitung eines bestimmten Maximalwertes (6 - 18 Uhr: 150 W/m²) aktiviert wird. Die gemessenen Werte, wann die Jalousien aktiviert sind und bei welchem Prozentsatz der Öffnung die Jalousien in jedem Stockwerk sind, werden dann zusammen mit den oben genannten Parametern und der berechneten Gesamtstrahlung (Globalstrahlung) auf eine geneigte Oberfläche mit bestimmter Ausrichtung (*I_Nor*, *I_Sud*, *I_Ost*, *I_Wes*) in die Simulation eingespeist, um die Menge der solaren Gewinne durch die transparenten Bauteile in jeder Zone des Gebäudes zu berechnen. Der nachfolgende Ausschnitt aus dem Python-Skript stellt zunächst die Bestimmung allgemeiner Parameter sowie zonenspezifischer Parameter (hier beispielhaft für das Technikum) der transparenten Bauteile dar.

```
# General parameters:
params_ground_Refl = 0.2       # [-] Ground Reflection
I_extraTer = 1367              # [W/m²] Extraterrestrial irradiance
params_g_win_norm = 0.5        # [-] Solar Heat Gain Transmittance
params_g_tot_SoSchu = 0.05     # [-] Energy transmittance of glazing
params_F_f = 0.7         # [-] Reduction factor frame portion
params_F_w = 0.9         # [-] Red. f. non-perpendicular radiation incidence
params_P_v = 0.9         # [-] Reduction factor due to dirt
params_F_c = 0.7         # [-] Red. f. due to the curtains in the offices
params_P_s = 0.8         # [-] Reduction factor due to structural shading
# [-] Shading factor:
params_FC_SoSchu = params_g_tot_SoSchu/params_g_win_norm

# Parameters Zone 1 (Technikum):
params_A_win_Nor_1 = 78.19         # [m²] Window area to the north
params_A_win_Ost_1 = 181.26        # [m²] Window area to the east
params_A_win_Sud_1 = 78.19         # [m²] Window area to the south
params_A_win_Wes_1 = 0             # [m²] Window area to the west
params_A_win_Hor_1 = 11.25         # [m²] Window area on the roof

# [m²] Total window area:
A_win_1 = (params_A_win_Nor_1 + params_A_win_Ost_1 + params_A_win_Sud_1
           + params_A_win_Wes_1 + params_A_win_Hor_1)
```

Anschließend folgt die Berechnung des Sonnenschutzfaktors (*params_g_win_angle*).

```
# Calculation of the sun protection factor:
param_q = 4    # category parameter for type of window
param_p = 3    # number of panes in the configuration
param_a = 8
param_b = 0.25/param_q
param_c = (1-param_a-param_b)
param_alpha = 5.2 + 0.7*param_q
param_beta = 2
param_gamma = (5.26 + 0.06*param_p)+(0.73 + 0.04*param_p)*param_q
param_z = sun_h_list[i]/90

params_g_win_angle = (params_g_win_norm * (1 - param_a*param_z**param_alpha
                                           - param_b*param_z**param_beta
                                           - param_c*param_z**param_gamma))
```

Berechnung der Solargewinne durch transparente, nicht streuende Bauteile „k“ in W

- $\Phi_{sol,w,k} = F_{sh,ob,k} \cdot g_{tot} \cdot F_c \cdot F_W \cdot (1 - F_F) \cdot A_k \cdot I_{Sol(Nor,Sud,Wes,Ost)}$

$F_{sh,ob,k}$ Verschattungsfaktor durch außen liegende Hindernisse nach VDI 6007-3 11.4.4
$g_{tot} \cdot F_c$ Gesamtenergiedurchlassgrad und DIN 4108-2 8.3.2 und Tabelle 7
F_W Korrekturfaktor für nicht streuende Bauteile = 0,9 (VDI 6007-3 11.4.1)
F_F Rahmenanteil des Bauteils
A_k Fläche des Bauteils
I_{sol} Solare Einstrahlung in W/m² nach VDI 6007-3.

Der nachfolgende Ausschnitt aus dem Python-Skript zeigt die Ermittlung der solaren Gewinne (*PHI_sol*) beispielhaft für das Technikum.

```
# wirksamer Gesamtenergiedurchlassgrad des Fensters (Technikum -> keine
Sonnenschutzvorrichtung):
g_eff_OhneSoSch = Params_F_s*params_F_w*Params_F_v*params_g_win_angle

# Calculation of PHI_sol Zone 1 (Technikum):
PHI_sol_Nor_1 = params_F_f*params_A_win_Nor_1*g_eff_OhneSoSch*I_Nor
PHI_sol_Ost_1 = params_F_f*params_A_win_Ost_1*g_eff_OhneSoSch*I_Ost
PHI_sol_Sud_1 = params_F_f*params_A_win_Sud_1*g_eff_OhneSoSch*I_Sud
PHI_sol_Wes_1 = params_F_f*params_A_win_Wes_1*g_eff_OhneSoSch*I_Wes
PHI_sol_Hor_1 = params_F_f*params_A_win_Hor_1*g_eff_OhneSoSch*I_Hor

PHI_sol_1 = (PHI_sol_Nor_1 + PHI_sol_Ost_1 + PHI_sol_Sud_1 + PHI_sol_Wes_1
             + PHI_sol_Hor_1)
```

Abbildung 8 stellt eine beispielhafte Visualisierung der berechneten solaren Gewinne für das dritte Obergeschoss während dem Zeitraum Januar bis Dezember 2022 dar.

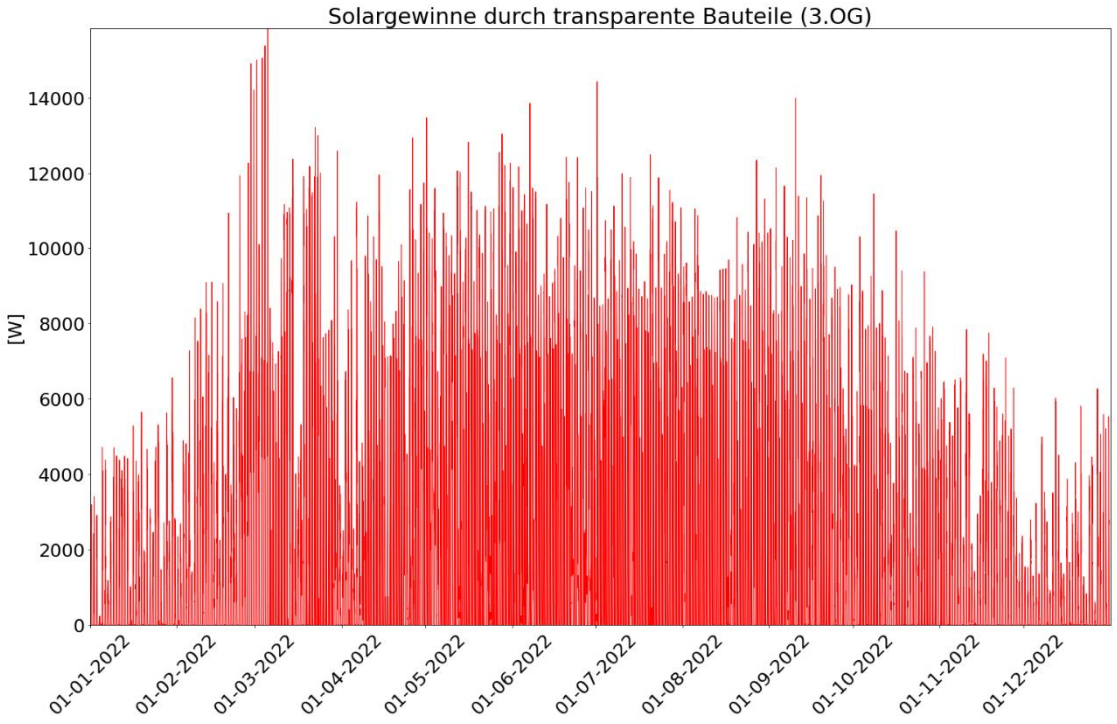

Abbildung 8 Solargewinne durch transparente Bauteile (3. OG.) - Zeitraum: Januar bis Dezember 2022

3.1.3 Photovoltaikanlagenmodell

Das Photovoltaikanlagenmodell ermittelt die generierte Elektrizität der Photovoltaikanlage (PVA) auf dem Dach des RIZ Energie über einen beliebigen Zeitraum, also die elektrische Energie in Wattstunden. Bei der Berechnung werden dabei zum einen die Einstrahlungswerte aus dem Strahlungsmodell, zum anderen anlagenspezifischer Parameter verwendet, die nachfolgend aufgelistet sind. Abbildung 9 vergleicht die gemessene und simulierte Stromerzeugung für den Zeitraum von Janur bis März 2022.

```
# Parameters PV System
Par_PVA_eta = 0.184 # [%] Solar panel yield (Efficiency)
Par_PVA_PR = 0.9    # [%] Performance Ratio, coefficient for losses
Par_PVA_A = 180     # [m²] Total solar panel area
```

Berechnung des Energieertrags der Photovoltaikanlage

- $E = I_{sol} \cdot A \cdot r \cdot PR$

E Energie [kWh] (*PVA_Pel*)
I_{sol} Solare Einstrahlung in W/m² (*I*)
A Gesamtfläche Solarpanel (*PVA_Area*)
r Effizienz (*PVA_eta*)
PR Nutzungsgrad / Verlustkoeffizient (*PVA_PR*)

Dementsprechend ergibt sich folgende Umsetzung im Modell:

```
def PV_Anlage(I, PVA_Area, PVA_eta, PVA_PR):

    PVA_Pel = (I*PVA_Area*PVA_eta*PVA_PR)
    return(PVA_Pel) #[Wh]

for i in range(Range):

    PVA_Pel_Ost = PV_Anlage(
        I = PV_I_Ost_list[i],
        PVA_Area = Par_PVA_A/2,
        PVA_eta = Par_PVA_eta,
        PVA_PR = Par_PVA_PR)

    PVA_Pel_Ost_list.append(PVA_Pel_Ost)

    PVA_Pel_West = PV_Anlage(
        I = PV_I_West_list[i],
        PVA_Area = Par_PVA_A/2,
        PVA_eta = Par_PVA_eta,
        PVA_PR = Par_PVA_PR)

    PVA_Pel_West_list.append(PVA_Pel_West)

PVA_Pel_SimList_W = [x1 + x2  for (x1, x2) in zip(PVA_Pel_Ost_list,
                                                  PVA_Pel_West_list)] #[W]
PVA_Pel_SimList_kW = [x/1000 for x in PVA_Pel_SimList_W] #[kW]
PVA_Pel_Sim_kWh = sum(x*0.25 for x in PVA_Pel_SimList_kW) #[kWh]
```

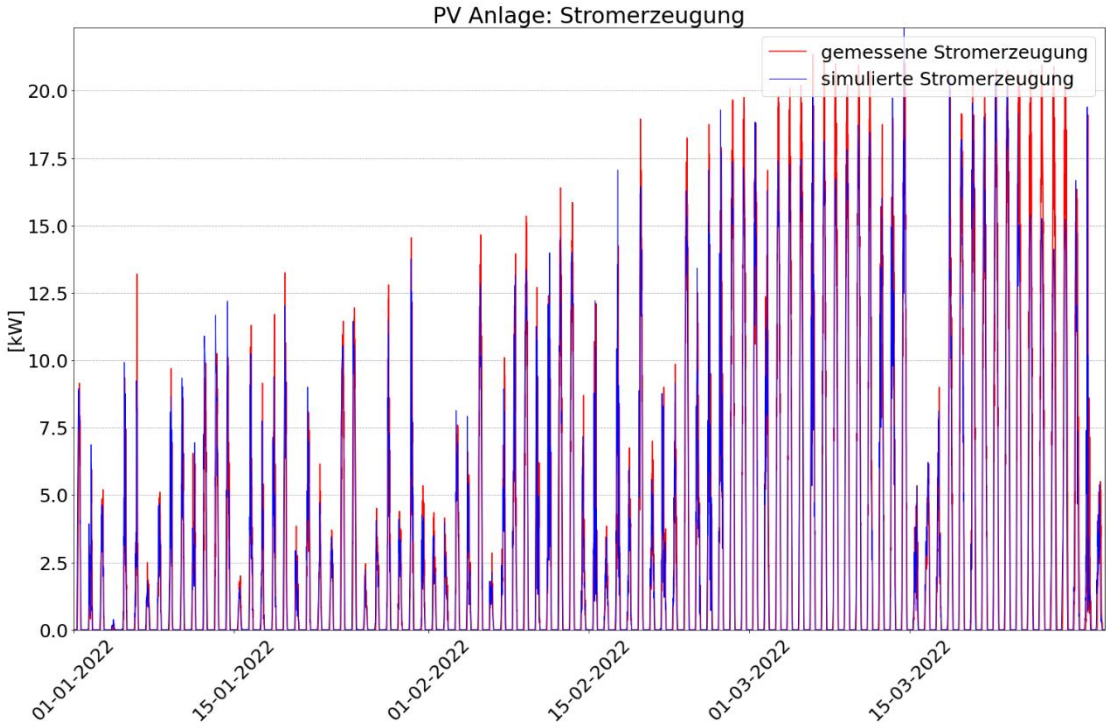

Abbildung 9 Gemessene und simulierte Stromerzeugung der PV-Anlage - Zeitraum: Janur bis März 2022

3.2 Gebäude- und Lüftungsanlagenmodell

Das Gebäudemodell liefert sowohl die operative Temperatur als auch die Nutzenergie (Heizung/Kühlung) für alle fünf in Kapitel 2 definierten Zonen. Für die Berechnungen der im Python-Skript verwendeten Gleichungen, werden zonenspezifische Parameter aus dem BIM-Modell und allgemeine Wandmaterialeigenschaften definiert.

Mit diesen Parametern können in einem vorbereitenden Schritt Werte für opake und transparente Bauteile, die Wärmetransferkoeffizienten, sowie die massebezogene Fläche und Speicherfähigkeit aller Zonen des Gebäudes berechnet werden. Zudem wird der Lüftungswärmetransferkoeffizient ermittelt.

```
"""-----------------------------------------------------------------------
Precalculated Model Variables
-----------------------------------------------------------------------"""
# Default values for dynamic parameters
RIZ_factor_A = 3.0    # [m²] table 12 DIN 13790 (Heavy)
RIZ_factor_C = 260000 # [m²] table 12 DIN 13790 (Heavy)

def CalcVar(
        RIZ_factor_A, RIZ_factor_C, Par_Build_Af, Par_Build_LambdaAt, \
        Par_Build_Awall_Ext, Par_Build_Uwall_Ext, Par_Build_Aroof, \
        Par_Build_Uroof, Par_Build_Afloor, Par_Build_Ufloor, \
        Par_Build_Awin, Par_Build_Uwin):

    # Effective mass area [m²]
    A_m = RIZ_factor_A * Par_Build_Af
    # Internal heat capacity [J/K]
    C_m = RIZ_factor_C * Par_Build_Af
    # Area facing building zone [m²]
    A_t = Par_Build_LambdaAt * Par_Build_Af
```

```
    # Heat transfer coefficients
    H_tr_op = ((Par_Build_Awall_Ext * Par_Build_Uwall_Ext)
               + (Par_Build_Aroof * Par_Build_Uroof)
               + (Par_Build_Afloor * Par_Build_Ufloor)) # [W/K]

    H_tr_w  = Par_Build_Awin * Par_Build_Uwin     # [W/K]
    H_tr_is = A_t * 3.45                          # [W/K]
    H_tr_ms = A_m * 9.1                           # [W/K]
    H_tr_em = 1.0 / (1.0/H_tr_op - 1.0/H_tr_ms)   # [W/K]

    ua = (1/((1/Par_TABS_h) + (Par_TABS_d/Prop_Conc_k))) * Par_Build_Af
    mc = Par_Build_Af * Par_TABS_s * Prop_Conc_Rho * Prop_Conc_C

    return(
        A_m, C_m, A_t, H_tr_op, H_tr_w, H_tr_is, H_tr_ms, H_tr_em, ua, mc)

Par_AHU_etaHRC = 0.84  # Heat recovery coefficient [-]
Par_AHU_fve = 1

def AHU(AHU_Vdot_Sup):

    b_ve_k  = 1 - (Par_AHU_fve*Par_AHU_etaHRC) # DIN EN ISO 13790
    H_ve = (b_ve_k * (Prop_Air_Rho*Prop_Air_Cp*1000)
            * (AHU_Vdot_Sup*0.25/3600))

    if H_ve == 0:
        H_ve = 1 * math.exp(-7)

    return(H_ve)
```

3.2.1 Ermittlung der operativen Raumtemperatur

Abbildung 10 zeigt die Modellformulierung analog zu der von elektrischen Netzwerken. Dabei werden Annahmen zur Struktur des thermischen Systems in ein Netzwerk aus fünf thermischen Widerständen (R) und einer Kapazität (C) überführt mit

- Wärmetransferkoeffizienten für Transmission $H_{tr,w}$ der Türen, Fenster, Vorhangfassaden und verglasten Wände und $H_{tr,op}$ opaker Bauteile unterteilt in $H_{tr,em}$ und $H_{tr,ms}$,
- Lüftungswärmetransferkoeffizienten H_{ve} und Zulufttemperatur θ_{sup},
- dem thermischen Kopplungsleitwert $H_{tr,is}$,
- der inneren Wärmespeicherfähigkeit C_m,
- Heiz- bzw. Kühlbedarf $\Phi_{HC,nd}$,
- Wärmeströme der inneren und solaren Wärmequellen Φ_{int} und Φ_{sol} sowie
- der Knoten-Außentemperatur θ_e, der Oberflächentemperatur der Innenwand θ_s, und der Lufttemperatur θ_{air}.

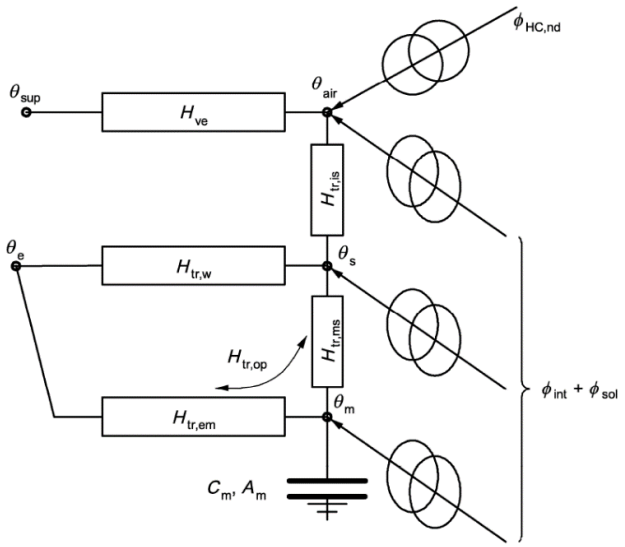

Abbildung 10 Modell mit fünf Widerständen und einer Kapazität (5R1C) nach ISO 13790 [1].

Der nachfolgende Ausschnitt des Python-Skripts zeigt die Gleichungen des Gebäudemodells (C1 bis C12). Dieses Modell liefert die operative Temperatur der Zone, die auf der Grundlage der Temperatur der Luft und der Oberfläche der Innenwände berechnet wird. Diese Temperatur wird dann mit der gemessenen Temperatur verglichen, um das Modell zu validieren. Weitere Informationen zum Modell und den Gleichungen sind in ISO 13790 Anhang C zu finden [1].

```
"""-----------------------------------------------------------------------
Building model - ISO 13790
-----------------------------------------------------------------------
Time [h]            --> |     |
PHI_sol [W]         --> |     |
PHI_Int [W]         --> |     |
Wea_thetaAmb [°C]   --> |     |--> Build_thetaAir_actual [°C]
theta_sup [°C]      --> | SHM |--> THETA_op [°C]
H_ve [W/K]          --> |     |--> THETA_m_t [°C]
PHI_HC              --> |     |
theta_m_t_prev [°C] --> |     |
-----------------------------------------------------------------------"""

# Equations C1 to C3
def equaC1C3(PHI_Int, PHI_sol, A_m, A_t, H_tr_w):

    PHI_ia = 0.5*PHI_Int
    PHI_m  = (A_m/A_t) * (0.5*PHI_Int + PHI_sol)
    PHI_st = ((1 - (A_m/A_t) - (H_tr_w/(9.1*A_t)))
              * (0.5*PHI_Int + PHI_sol))

    return(PHI_ia, PHI_m, PHI_st)

# Equations C4 to C12
def equaC4C12(
        theta_m_t_prev, Wea_thetaAmb, theta_sup,
        PHI_m, PHI_st, PHI_ia, PHI_HC,
        H_ve, H_tr_is, H_tr_w, H_tr_ms, H_tr_em, C_m):

    H_tr_1 = 1 / ((1/H_ve) + (1 / H_tr_is))
    H_tr_2 = H_tr_1 + H_tr_w
    H_tr_3 = 1 / ((1/H_tr_2) + (1/H_tr_ms))
```

```
    PHI_m_tot = ((PHI_m + (H_tr_em*Wea_thetaAmb)
                + (H_tr_3 * ((PHI_st + (H_tr_w*Wea_thetaAmb)
                + (H_tr_1 * (((PHI_ia+PHI_HC)/H_ve) + theta_sup)))
                / H_tr_2))))

    # Temperatures
    theta_m_t = ((theta_m_t_prev*((C_m/900) - 0.5 * (H_tr_3+H_tr_em))
                + PHI_m_tot) / ((C_m/900) + 0.5 * (H_tr_3+H_tr_em)))
    theta_m = (theta_m_t+theta_m_t_prev) / 2
    theta_s = (((H_tr_ms*theta_m) + PHI_st + (H_tr_w*Wea_thetaAmb)
              + H_tr_1 * (theta_sup + (PHI_ia+PHI_HC)/H_ve))
              / (H_tr_ms+H_tr_w+H_tr_1))
    Build_thetaAir = (((H_tr_is*theta_s) + (H_ve*theta_sup)
                      + PHI_ia+PHI_HC) / (H_tr_is+H_ve))
    theta_op = 0.3 * Build_thetaAir + 0.7 * theta_s

    return(theta_m, theta_s, Build_thetaAir, theta_m_t, theta_op)
```

Abbildung 11 zeigt beispielhaft den Vergleich zwischen der simulierten und der gemessenen Raumtemperatur im 2. und 3.OG (Zone 4 und Zone 5) des RIZ Energie. Diese Abbildung zeigt auch die gemessene Umgebungstemperatur.

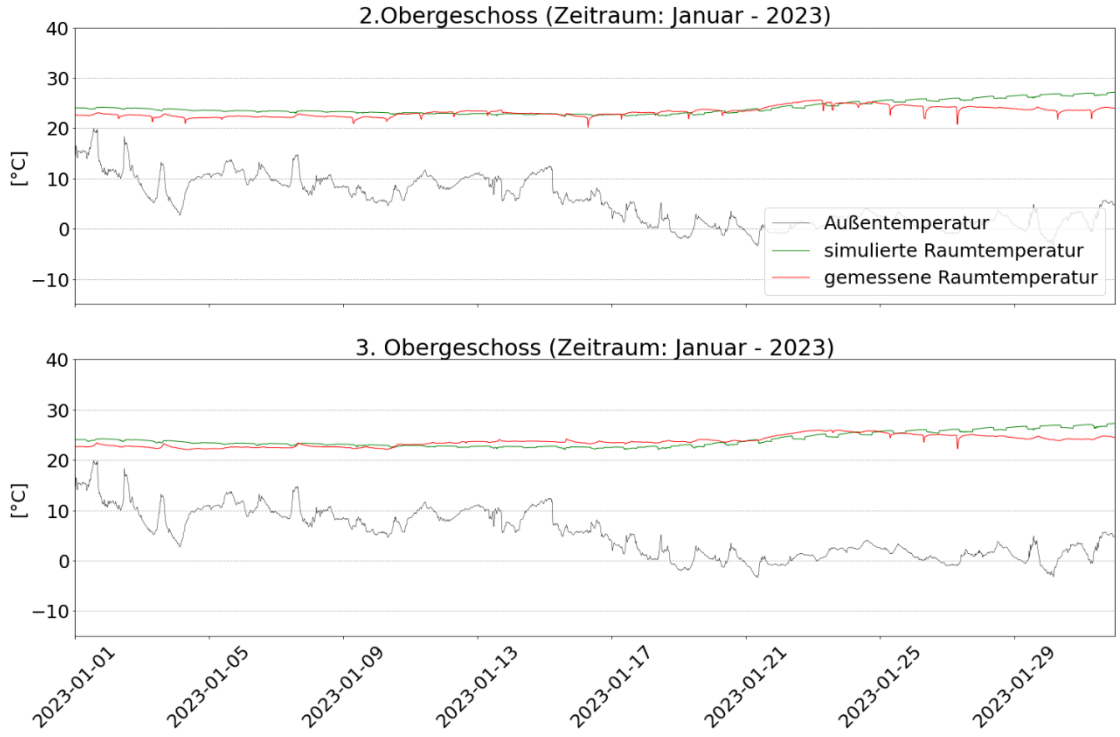

Abbildung 11 Vergleich der simulierten mit der im 2. und 3.OG gemessenen Raumtemperatur

3.2.2 Lüftungsanlage (Air Handling Unit/AHU)

Die Lüftungsanlage wird als Teil des Gebäudemodells simuliert, da sie laufend in Abhängigkeit der operativen Temperaturen (*theta_op*) gesteuert wird. Sie ist mit zwei Heizstufen konzipiert, die im Winter genutzt werden (im Sommer gibt es einen Bypass-Modus und die Luft kommt direkt mit der Außentemperatur), um die Solltemperaturen der Zuluft zu erreichen. Zuerst wird ein Wärmerückgewinnungssystem (WRG) genutzt, um die Außentemperatur mit Hilfe der

Ablufttemperatur, die aus den Räumen kommt, zu erhöhen. Sollte dies nicht ausreichen, wird ein Wärmetauscher (Heizregister) eingesetzt, der die Temperatur auf die Solltemperatur erhöht.

Das Simulationsmodell in Python verwendet folgende Eingangsdaten:

- Gemessene Außenlufttemperatur *Wea_thetaAmb*
- Gemessener Luftvolumenstrom *AHU_Vdot_Sup*
- Simulierte Ablufttemperatur *AHU_theta_Ret*, berechnet aus den operativen, simulierten Raumtemperaturen

Außerdem werden die nachstehenden Berechnungsparameter benutzt:

- Wärmerückgewinnungskoeffizient (*Par_AHU_etaHRC*)
- Sollwerttemperatur Lüftungsanlage (*ThetaSupSet*)

Das Lüftungsanlagenmodell bestimmt dann die Zulufttemperatur (*AHU_theta_Sup*) und die Fortlufttemperatur (*AHU_theta_Exha*) als Ausgangsdaten und berücksichtigt mithilfe von Konditionen alle Heizstufen und Modi der Lüftungsanlage.

```
"""--------------------------------------------------------------------------
AHU Model (including heat recovery and internal regulation/control):
--------------------------------------------------------------------------
Intake (Außentemp) [°C] --> |     | <-- Return [°C] (Abluft)
                            | AHU |
Exhaust (Fortluft) [°C] <-- |     | --> Supply [°C] (Zuluft)
--------------------------------------------------------------------------"""
# Set temperature AHU (after HX)
if Stunde_kumuliert_list[i] <= StopHE:
    ThetaSupSet = 21
elif Stunde_kumuliert_list[i] > StopHE and Monat_list[i] < 9:
    ThetaSupSet = 19
else:
    ThetaSupSet = 16

Par_AHU_etaHRC =  Par_AHU_etaHRC_list[-1]
theta_op_mean = [Temperatur_Technikum[-1], theta_op_list_EG[-1],
                 theta_op_list_1OG[-1], theta_op_list_2OG[-1],
                 theta_op_list_3OG[-1]]

if AHU_Vdot_Mea_list[i] != 0:
    AHU_theta_Ret_SimMean = statistics.mean(theta_op_mean)
else:
    AHU_theta_Ret_SimMean = 0

AHU_theta_Ret_SimMean_list.append(AHU_theta_Ret_SimMean)

# AHU model function
def AHU(Wea_thetaAmb, AHU_theta_Ret, H, AHU_Vdot_Sup, Stunde):

    if AHU_Vdot_Sup == 0 or AHU_theta_Ret == 0:
         AHU_Qdot_HX = P09_mdot = AHU_theta_Sup_AHR = AHU_theta_Sup = 0
    else:
        #Temperature after heat recovery system and before HX
        AHU_theta_Sup_AHR = (Wea_thetaAmb + Par_AHU_etaHRC
                             * (AHU_theta_Ret - Wea_thetaAmb))
        # Heat recovery system plus HX
        if ((AHU_theta_Sup_AHR < ThetaSupSet)
            and (H >= StartHE or H <= StopHE)):
            # water volume flow rate [kg/s]
```

```
            P09_mdot = 1*0.28
            # Power needed to get the set temperature after HX [kW]
            AHU_Qdot_HX = ((AHU_Vdot_Sup/3600)*Prop_Air_Rho
                           *Prop_Air_Cp*(ThetaSupSet - AHU_theta_Sup_AHR))
            # set supply temperature after HX
            AHU_theta_Sup = ThetaSupSet
        # Heat recovery system (without HX)
        else:
            AHU_Qdot_HX = P09_mdot = 0
            if Par_AHU_etaHRC == 0:
                AHU_theta_Sup = Wea_thetaAmb - 1
            else:
                if (Wea_thetaAmb < ThetaSupSet
                    and AHU_theta_Ret >= ThetaSupSet):
                    AHU_theta_Sup = ThetaSupSet
                elif (Wea_thetaAmb > ThetaSupSet
                      and AHU_theta_Ret <= ThetaSupSet):
                    AHU_theta_Sup = ThetaSupSet
                else:
                    AHU_theta_Sup = AHU_theta_Sup_AHR

    AHU_theta_Exha = AHU_theta_Ret - (AHU_theta_Sup - Wea_thetaAmb)

    return(AHU_theta_Sup_AHR, AHU_theta_Sup, AHU_theta_Exha,
           AHU_Qdot_HX, P09_mdot)
```

Abbildung 12 zeigt die gemessene und simulierte Zuluf- (ZUL) und Abluftemperatur (ABL) von Januar bis Dezember 2022.

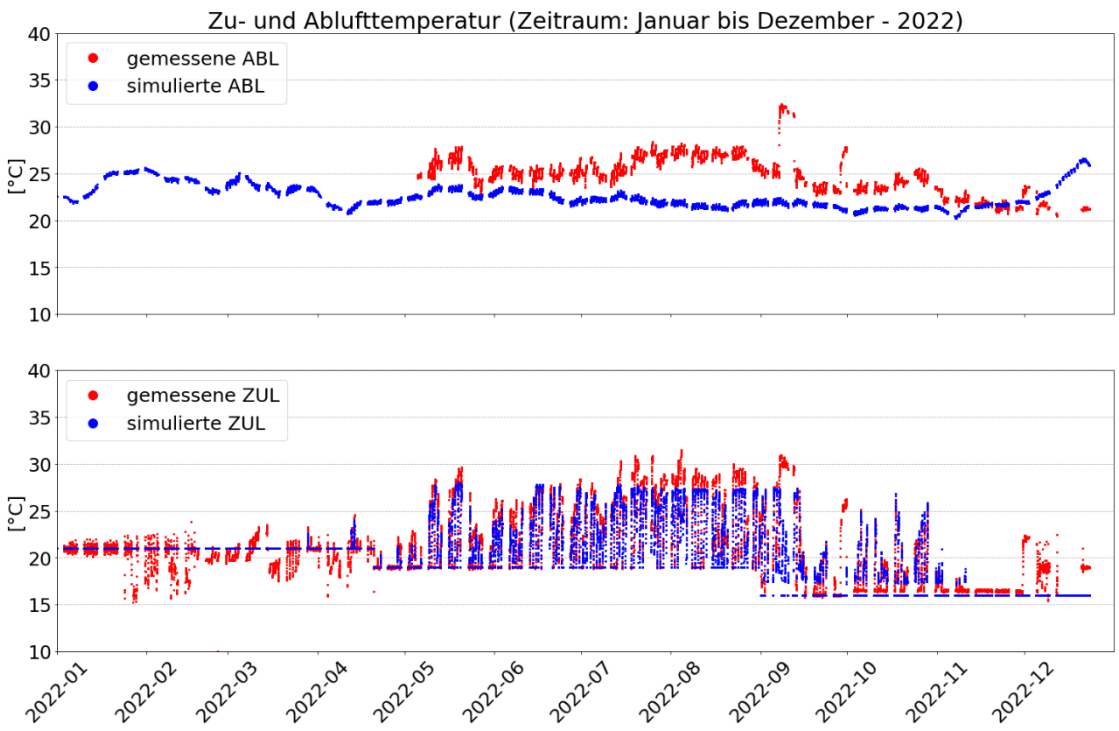

Abbildung 12 Zulufttemperatur der Lüftungsanlage nach der WRG-Stufe und Außentemperatur - Zeitraum: Januar bis Dezember 2022

3.2.3 MSR-Modul Thermoaktive Bauteilsysteme (TABS)

Das RIZ Energie wird mithilfe von thermoaktiven Bauteilsysteme (TABS) klimatisiert. TABS sind wasserdurchströmte Rohrsysteme, die in Bauteile der Gebäudestruktur in Mittellage integriert oder oberflächennah in den Deckenputz oder den Estrich eingebettet werden. Bevor die Nutzenergie (Heiz- bzw. Kühlleistung) der TABS ermittelt werden kann, wird zunächst das MSR-Modul der TABS in die Simulation implementiert. Dabei wird die Aktivierung bzw. Kontrollfunktion der verschiedenen TABS-Modi simuliert. Zur Differenzierung erhält der Beheizungsmodus hierin den Wert 1 und der Kühlungsmodus den Wert -1, während Inaktivität gleich 0 gesetzt wird.

Im Verwaltungstrakt (Büros) des RIZ Energie wird ein zweilagiges TABS-System angewendet: die BKT in Mittellage zur Grundtemperierung und die oberflächennahe Bauteilaktivierung in der Randzone der Betondecke zur dynamischen Raumregelung. Im Erdgeschoss, also im Technikum sowie in den Lagern und Werkstätten, kommt eine Fußbodenheizung bzw. -kühlung als Raumübergabesystem zum Einsatz. Die verschiedenen Heiz- bzw. Kühlkreise werden über Zeitprogramme in der Gebäudeleittechnik aktiviert:

- Tagbetrieb: Heiz-/Kühlkreis Büro der oberflächennahen BKT:
  Mo. - Fr. 08:00 Uhr bis 18:00 Uhr
- Nachtbetrieb: Heiz-/Kühlkreis Büro BKT in Mittellage:
  Mo. - So. 20:00 Uhr bis 08:00 Uhr
- Forschungsbetrieb: Industrieflächentemperierung Heiz-/Kühlkreis Technikum:
  Mo. - So. 00:00 Uhr bis 24:00 Uhr

Diese Zeitprogramme werden im MSR-Modul für die TABS festgelegt. Weiterhin werden die Zeitperioden von Heiz- und Kühlbetrieb stundengenau berücksichtigt und definiert.

```
# Heating and cooling periods - Büros
StartCH = 2847 # Hour of the year where the cooling starts (29.04.)
StopCH  = 6768 # Hour of the year where the cooling stops (11.10.)
StartHH = 7464 # Hour of the year where the heating starts (08.11.)
StopHH  = 1536 # Hour of the year where the heating stops (05.03.)

# Heating and cooling periods - EG und Technikum
StartCH_EGT = 2847 # Hour of the year where the cooling starts (29.04.)
StopCH_EGT  = 6792 # Hour of the year where the cooling stops (11.10.)
StartHH_EGT = 7464 # Hour of the year where the heating starts (08.11.)
StopHH_EGT  = 1536 # Hour of the year where the heating stops (05.03.)

# TABS Control functions
def BA_TABS_Bueros(H, Stunde, theta_op):
    if (H >= StartHH or H <= StopHH) and (Stunde <= 8 or Stunde >= 20):
        Control_TABS = 1  # Switched on heating
    elif (StartCH <= H <= StopCH) and (Stunde <= 8 or Stunde >= 20):
        Control_TABS = -1 # Switched on Cooling
    else:
        Control_TABS = 0  # Switched off
    return(Control_TABS)

def BA_TABS_Technikum(H, Stunde, theta_op):
    if (H >= StartHH_EGT or H <= StopHH_EGT):
        Control_TABS = 1  # Switched on heating
    elif (StartCH_EGT <= H <= StopCH_EGT):
        Control_TABS = -1 # Switched on Cooling
    else:
        Control_TABS = 0  # Switched off
    return(Control_TABS)
```

3.2.4 Ermittlung der Nutzenergie: Thermoaktive Bauteilsysteme (TABS)-Modell

Das TABS-Modell besitzt als Eingabedaten die Raumlufttemperatur (*Build_thetaAir*), die vorherige Betontemperatur (*TABS_thetaAC_prev*) und die konstanten Massenströme der einzelnen Gebäudezonen (*TABS_mdot*) und die Widerstände aufgrund der Rohranordnung im Bauteil (*Rx*) sowie zur Berücksichtigung der Wärmeleitung durch die Rohrwand (*Rr*).

Hinzu kommen einige Parameter des Gebäudes wie der Gesamtwärmeübergangskoeffizient des Betons multipliziert mit der konditionierten Bodenfläche (*UA*) und *mc*, welcher aus der Multiplikation der Wärmekapazität des Wandmaterials mit der Wanddicke, der Dichte des Betons und der konditionierten Bodenfläche resultiert.

Die aktuelle Temperatur des aktiven Bauteils (*TABS_thetaAC*), welches die Zone heizt oder kühlt, und die aktuelle Heiz-/Kühlleistung (*Speicher2TABS_Qdot*) werden anschließend berechnet.

```
def TABS_Model(Rx, Rr, Dx, Da, Dr, L, TABS_thetaVL, TABS_thetaRL,
               TABS_mdot, Control_TABS, H, TABS_thetaAC_prev,
               Build_thetaAir, UA, mc):

    if Control_TABS != 0 and TABS_mdot != 0 and TABS_thetaVL != 0:

        TABS_mdot_sp = TABS_mdot / (Dx*L) # [kg/m²s]
        Di = Da - (2*Dr) # Inner diameter
        w = TABS_mdot / Prop_Water_Rho / (Di**2*pi/4) # Water Velocity
        TW = (TABS_thetaVL + TABS_thetaRL)/2 # Mean temperature water
        # Heat transfer coefficient between the fluid and the pipe wall
        alphaw = 2040 * (1 + 0.015*TW) * w ** 0.87 / ((Da - 2*Dr) ** 0.13)

        Rw = Dx / (alphaw * (Da - 2*Dr) * pi)

        U1 = 1 / ((1/Prop_Air_h_UO) + (0.18/lambdaB))
        U2 = 1 / ((1/Prop_Air_h_OU) + (0.1/lambdaB))
        # Resistor to account for nonlinear fluid temperature distribution
        Rz = (1 / (TABS_mdot_sp * Prop_Water_Cp
                   * (1 - exp(-1 / ((Rw + Rr + Rx +1 / (U1+U2))
                                    * TABS_mdot_sp * Prop_Water_Cp))))
              - (Rw + Rr + Rx +1 / (U1+U2)))

        Rt = Rz + Rx + Rw + Rr # [m²K/W] Total thermal resistance

        if H >= StartHH or H <= StopHH: # Winter (Heating mode)
            TABS_thetaRL = (TABS_thetaVL
                            - (1 / (Rt*1.5)
                               * (TABS_thetaVL-TABS_thetaAC_prev)
                               / (TABS_mdot_sp*Prop_Water_Cp)))
        else:
            TABS_thetaRL = (TABS_thetaVL
                            + (1 / (Rt*1.5)
                               * (TABS_thetaAC_prev-TABS_thetaVL)
                               / (TABS_mdot_sp*Prop_Water_Cp)))
            # Avoid peaks in Qdot due to large DeltaT
            if TABS_thetaRL > TABS_thetaVL + 5:
                TABS_thetaRL = TABS_thetaVL + 5

        Speicher2TABS_Qdot = TABS_mdot * Prop_Water_Cp
        * (TABS_thetaVL-TABS_thetaRL) / 1000 # [kW]
```

```
        TABS_thetaAC = (TABS_thetaAC_prev +
                        ((Speicher2TABS_Qdot +
                          (UA * (Build_thetaAir-TABS_thetaAC_prev)))
                         * 0.25 * 900) / mc)

        if (((H >= StartHH or H <= StopHH) and Speicher2TABS_Qdot < 0)
                or ((StartCH <= H <= StopCH) and Speicher2TABS_Qdot > 0)):
                Speicher2TABS_Qdot = 0

            TABS_DeltaT = abs(TABS_thetaVL-TABS_thetaRL)

        if ((Control_TABS == 0) or abs(TABS_thetaVL - TABS_thetaRL) < 0.5
            or (StopHH <= H <= StartCH) or (StopCH <= H <= StartHH)
            or (TABS_mdot == 0) or (TABS_thetaVL == 0)):

           Speicher2TABS_Qdot = 0
           TABS_thetaAC = (TABS_thetaAC_prev
                           + ((Speicher2TABS_Qdot
                               + (UA * (Build_thetaAir-TABS_thetaAC_prev)))
                              *0.25*900)/mc)
           TABS_thetaRL = 0
           TABS_DeltaT = 0

        return(Speicher2TABS_Qdot, TABS_thetaRL, TABS_thetaAC, TABS_DeltaT)
```

Abbildung 13 zeigt beispielhaft die simulierte Heizleistung des TABS-Systems im Januar. Die Operation im Tag-Nacht-Betrieb ist klar ersichtlich und kann durch das MSR-Modul (siehe Abschnitt 3.3.3) simuliert werden.

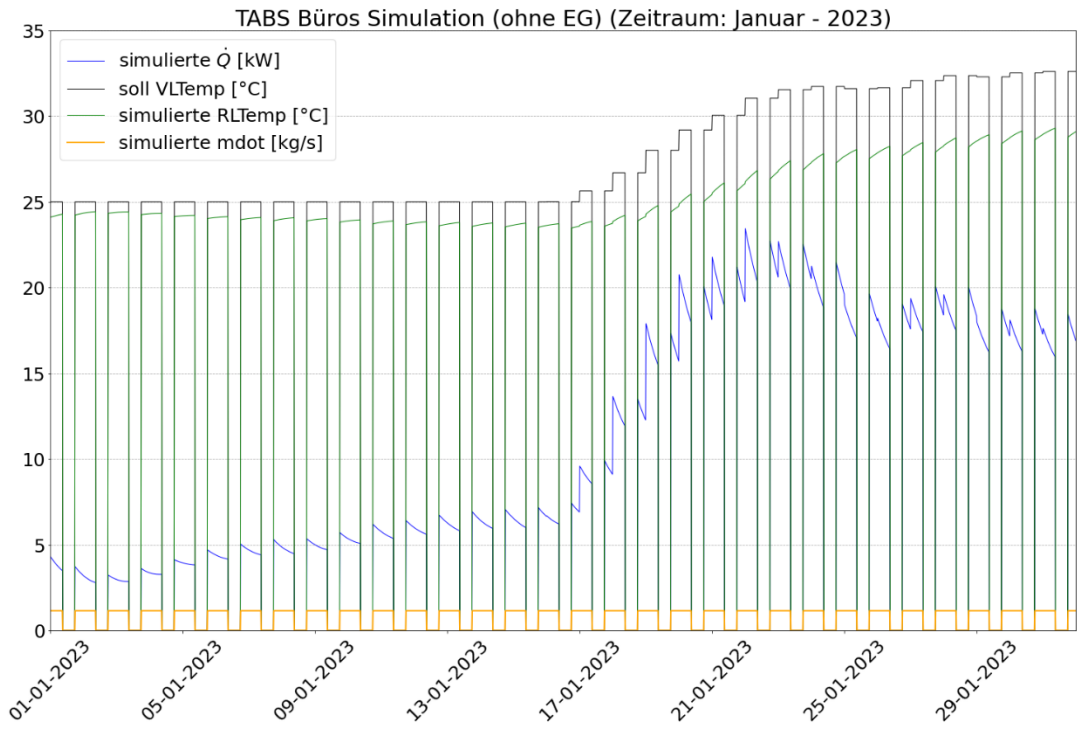

Abbildung 13 Heizleistung der TABS (Thermoaktive Bauteilsysteme) - Zeitraum: Janur 2023

3.3 Wärmespeichermodell

Die Modellierung von Wärmespeichern ist aufgrund der physikalischen Effekte der thermischen Schichtung, der erzwungenen Konvektion oder der laminaren Strömungen, die je nach Konstruktion des Tanks auftreten können, komplex. Deshalb sollte zumindest ein einfaches Schichtenmodell verwendet werden. Hier wird ein dynamisches 1-D-Mehrschichtmodell unter Verwendung der Fourier Gleichung nach [4] an den dynamischen digitalen Zwilling angepasst.

Dieses analytische Modell fasst den komplexen Wärmestrom unter Verwendung eines effektiven vertikalen Wärmeleitfähigkeitskoeffizienten λ_{eff} zusammen. Der Heißwassertank wird als ein vertikal geschichteter zylindrischer Tank betrachtet, mit benutzerdefinierten Abmessungsparametern wie beispielsweise Tankdurchmesser und -höhe, Dicke der Tankwand oder der Anzahl der Schichten in Längsrichtung.

```
# Water
Prop_Water_Cp  = 4.19    # Specific heat capacity [kJ/kg K]
Prop_Water_Rho = 1000    # Density [kg/m³]

# Thermal emergy storage (TES) model parameters
Par_TES_Vstor  = 1.316   # tank volume [m³]
Par_TES_rtank  = 0.85/2  # tank radius [m]
Par_TES_htank  = 2.55    # tank height [m]
Par_TES_dtank  = 0.85    # tank diameter [m]
Par_TES_Atank  = 2*math.pi*Par_TES_rtank*(Par_TES_rtank
                                          +Par_TES_htank)  # tank area [m²]
Par_TES_thtank = 0.16    # wall thickness [m]
Par_TES_Utank  = 0.2     # heat transition coefficient [W/m²K]
Par_TES_N      = 3       # number of layers
k              = 0.002   # heat transfer coeff. of tank envelope [W/m²K]
lambda_eff     = 0.0015  # eff. vertical heat conductivity of water [W/m²K]
```

Aus diesen Parametern werden die relevanten Dimensionsgrößen für die einzelnen Schichten berechnet und je Schicht eine initiale Wassertemperatur festgelegt:

```
# layer parameters
zi    = Par_TES_htank / Par_TES_N      # height of a layer [m]
Ai    = (math.pi*
         (Par_TES_dtank
          - 2*Par_TES_thtank)**2) / 4  # cross-section area of a layer [m²]
Aexti = math.pi*Par_TES_dtank*zi  # ext. heat tr. surface area / layer [m²]
mi    = Ai*zi*Prop_Water_Rho           # mass of a layer [kg]

# initial value for layers of TES
if (H[i] <= StopHH or H[i] >= StartHH):
    TES_theta_list = np.linspace(19, 23, Par_TES_N).tolist() # [°C]
else:
    TES_theta_list = np.linspace(theta_op_list_EG[i],
                                 theta_op_list_EG[i],
                                 Par_TES_N).tolist()         # [°C]
```

Für jede Schicht wird anschließend mit dem dynamischen Schichtenmodell ein effektiver Massenstrom und die allgemeine Energiebilanz berechnet. Eingangsgrößen sind dabei die Austrittstemperatur des Verflüssigers der Wärmepumpe, die von den TABS ausgehende Rücklauftemperatur, die untere Wassertemperatur des Tanks (zur Quelle, also zur Wärmepumpe), die obere Wassertemperatur des Tanks (zur Last, also zu den TABS) und die Massenströme.

```
m_dot_HP   = P04_mdot_list[-1]
m_dot_load = P07_mdot_list[-1] + P08_mdot_list[-1] + P09_mdot_list[-1]
m_dot      = m_dot_HP - m_dot_load # >0: TES charged, <0: TES discharged
m_dot_list.append(m_dot)

def TES(Par_TES_N, Tfs, Trl, m_dot, TES_top, TES_bottom, Tamb):

    Ti = np.linspace(TES_bottom, TES_top, Par_TES_N).tolist()
    t  = np.linspace(0, 15, 900)  # Time Grid (15 minutes time stamp)

    # bottom layer
    def energybalance_1(y, t):
        Ti[i] = y

        if m_dot > 0:
            dTdt = (((Ai*lambda_eff/zi) * (Ti[i+1] - Ti[i]))
                    + (m_dot * Prop_Water_Cp * (Ti[i+1] - Ti[i]))
                    - (k * Aexti * (Ti[i] - Tamb))) / (mi*Prop_Water_Cp)
        else:
            dTdt = (((Ai*lambda_eff/zi)*(Ti[i] - Ti[i+1]))
                    + (m_dot * Prop_Water_Cp * (Ti[i] - Trl))
                    - (k*Aexti*(Ti[i] - Tamb))) / (mi*Prop_Water_Cp)- 0.03
        return dTdt

    i  = 0      # bottom layer index
    y0 = Ti[i]  # initial condition
    Ti[i] = float(odeint(energybalance_1, y0, t)[-1]) # solve ODE

    # middle layer
    def energybalance_2(y, t): # Function for the middle layer
        Ti[i] = y

        if m_dot > 0:
            dTdt = ((((Ai*lambda_eff)/zi) * (Ti[i+1] - 2*Ti[i] + Ti[i-1]))
                    + (m_dot*Prop_Water_Cp*(Ti[i+1] - Ti[i]))
                    - (k*Aexti*(Ti[i] - Tamb))) / (mi*Prop_Water_Cp)

        else:
            dTdt = ((((Ai*lambda_eff)/zi) * (Ti[i+1] - 2*Ti[i] + Ti[i-1]))
                    + (m_dot*Prop_Water_Cp*(Ti[i] - Ti[i-1]))
                    - (k*Aexti*(Ti[i] - Tamb))) / (mi*Prop_Water_Cp)
        return dTdt

    for i in range(1, Par_TES_N - 1): # middle layers index
        if  m_dot > 0:
            y0 = Ti[i+1] # layer above current layer as initial condition
        else:
            y0 = Ti[i-1] # layer under current layer as initial condition

        Ti[i] = float(odeint(energybalance_2, y0, t)[-1]) # solve ODE

    # top layer
    def energybalance_3(y, t): # Function for the top layer
        Ti[i] = y

        if m_dot>0:
            dTdt = (((Ai*lambda_eff/zi)*(Ti[i-1] - Ti[i]))
                    + (m_dot * Prop_Water_Cp * (Tfs - Ti[i]))
                    - (k*Aexti*(Ti[i] - Tamb))) / (mi*Prop_Water_Cp)
```

```
        else:
            dTdt = (((Ai*lambda_eff/zi)*(Ti[i] - Ti[i-1]))
                    + (m_dot * Prop_Water_Cp * (Ti[i] - Ti[i-1]))
                    - (k*Aexti*(Ti[i] - Tamb))) / (mi*Prop_Water_Cp)
        return dTdt

    i  = Par_TES_N - 1 # top layer index
    y0 = Ti[i]         # initial condition
    Ti[i] = float(odeint(energybalance_3, y0, t)[-1]) # solve ODE

    return Ti
```

Abbildung 14 zeigt exemplarisch den simulierten Lade- und Entladevorgang über einen Monat während des Winterbetriebs.

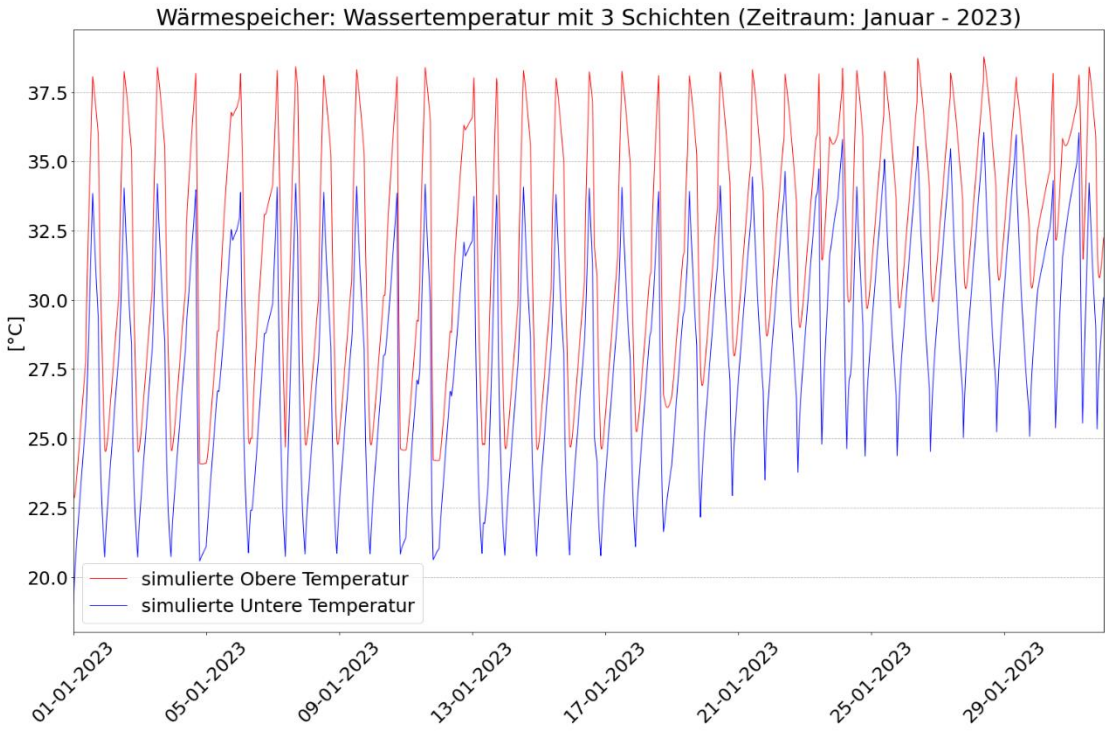

Abbildung 14 Wärmespeicher Temperatur- Zeitraum: Januar 2023

3.4 Wärmepumpenmodell

Dieses Modell basiert auf einem multiplen linearen Regressionsmodell gemäß der folgenden Gleichung:

- $Z = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 Y + \beta_4 Y^2$

In diesem Fall wurden die für das Grey-Box-Modell verwendeten Daten dem Datenblatt der Wärmepumpe entnommen, wobei die beiden unabhängigen Variablen *X* und *Y* die Eintrittstemperatur des Verdampfers (*HP_theta_Eva_In*) und die Austrittstemperatur des Verflüssigers (*HP_theta_Con_Out*) sind. Mit diesen Variablen werden die Koeffizienten β_0, β_1, β_2, β_3 und β_4 (*HC_Coeff*) der Regression gefunden, um die abhängigen Variablen *Z*, also die Heizleistung (*HP_Qdot_Con*) und die elektrische Leistung (*HP_Pel*) der Wärmepumpe zu berechnen.

```
"""-------------------------------------------------------------------------
Heat Pump Model
-------------------------------------------------------------------------
HP_theta_Eva_In [°C] --> |     | --> HP_Qdot_Con: Th. power condenser [kW]
HP_theta_Con_In [°C] --> |     | --> HP_Pel: Electrical power  [kW]
HP_mdot_Eva [kg/s]   --> | HP  | --> HP_Qdot_Eva: Heat absorption [kW]
HP_mdot_Con [kg/s]   --> |     | --> HP_theta_Eva_Out: Outlet temp. [°C]
HP_Control (On/Off)  --> |     | --> HP_theta_Con_Out: Outlet temp. [°C]
-------------------------------------------------------------------------"""

def HP(HP_Control, HP_theta_Eva_In, HP_theta_Con_In, HP_mdot_Eva,
       HP_mdot_Con, HP_theta_Con_Out_prev, HP_theta_Eva_Out_prev,
       TES_theta_set, TES_theta_Top, Tamb):

    if HP_Control == 0:
        HP_Qdot_Con = HP_Qdot_Eva = HP_Pel = 0

        if HP_theta_Con_Out_prev > Tamb:
            HP_theta_Con_Out = HP_theta_Con_Out_prev - 1
        else:
            HP_theta_Con_Out = Tamb

        if HP_theta_Eva_Out_prev < Tamb:
            HP_theta_Eva_Out = HP_theta_Eva_Out_prev + 1
        else:
            HP_theta_Eva_Out = Tamb
    # Multiple linear regression models
    elif HP_Control == 1:
        HP_Qdot_Con = (HC_Coeff[0] + (HC_Coeff[1]*HP_theta_Eva_In)
                       + (HC_Coeff[2]*HP_theta_Con_In)
                       + (HC_Coeff[3]*HP_theta_Eva_In**2)
                       + (HC_Coeff[4]*HP_theta_Con_In**2))

        HP_Pel = (Pel_Coeff[0] + (Pel_Coeff[1]*HP_theta_Eva_In)
                  + (Pel_Coeff[2]*HP_theta_Con_In)
                  + (Pel_Coeff[3]*HP_theta_Eva_In**2)
                  + (Pel_Coeff[4]*HP_theta_Con_In**2))
```

Die Berechnung der Austrittstemperaturen des Verflüssigers (*HP_theta_Con_Out*) und des Verdampfers (*HP_theta_Eva_Out*) erfolgt anschließend mit einer Energiebilanz über den Wärmepumpenkreislauf:

```
        # Energy balance over heat pump cycle
        HP_Qdot_Eva = HP_Pel - HP_Qdot_Con
        # Outlet temperatures
        HP_theta_Con_Out = (HP_theta_Con_In
                            + (HP_Qdot_Con / (HP_mdot_Con*Prop_Water_Cp)))
        HP_theta_Eva_Out = (HP_theta_Eva_In
                            + (HP_Qdot_Eva / (HP_mdot_Eva*Prop_Water_Cp)))
    return(HP_Qdot_Con, HP_Qdot_Eva, HP_Pel,
           HP_theta_Eva_Out, HP_theta_Con_Out)
```

Dabei sind *HP_mdot_Eva* und *HP_mdot_Con* die Massenströme durch den Verdampfer bzw. Verflüssiger und *Prop_Water_Cp* ist die spezifische Wärmekapazität des Wassers bei konstantem Druck.

3.5 Wärmetauschermodell

Das Wärmetauschermodell basiert auf der NTU-Methode (Number of Transfer Units Method), die zur Ermittlung der Wärmeübertragungsrate von Wärmetauschern dient. Die Eingabedaten sind die Massenströme der Primär- (heißes Fluid: *PlateHX_mdot_Prim*) und Sekundärseite (kaltes Fluid: *PlateHX_mdot_Sec*), die Eintrittstemperaturen beider Seiten (*PlateHX_theta_Prim_In*, *PlateHX_theta_Sec_In*), die spezifischen Wärmekapazitäten beider Seiten (*PlateHX_Cp_Prim* = *PlateHX_Cp_Sec* = *Prop_Water_Cp*) und der Gesamtwärmeübergangskoeffizient multipliziert mit der Wärmeübertragungsfläche (*Par_PlateHX_AU*).

```
"""--------------------------------------------------------------------------
Plate Heat Exchanger Model: NTU Method
--------------------------------------------------------------------------
PlateHX_mdot_Prim [kg/s]  --> |      |
PlateHX_mdot_Sec [kg/s]   --> |      |
PlateHX_Cp_Prim [kJ/kg K] --> |      | --> PlateHX_theta_Prim_Out [°C]
PlateHX_Cp_Sec [kJ/kg K]  --> |  HX  | --> PlateHX_theta_Sec_Out [°C]
PlateHX_theta_Prim_In [K] --> |      | --> PlateHX_Qdot [kW]
PlateHX_theta_Sec_In [K]  --> |      |
Par_PlateHX_AU [kW/K]     --> |      |
--------------------------------------------------------------------------"""

PlateHX_Cp_Prim = Prop_Water_Cp
PlateHX_Cp_Sec = Prop_Water_Cp
```

Für die Berechnung muss zunächst ermittelt werden, welches das heiße und welches das kalte Fluid ist. Dann muss die maximale Temperaturdifferenz innerhalb des Wärmetauschers (*DT_max*) berechnet werden. Mit der minimalen und maximalen spezifischen Wärmekapazität der Fluide (*C_min* und *C_max*) werden dann die NTU-Zahl und der maximal mögliche Wärmeübergang (*Qdot_max*) im Wärmetauscher berechnet.

```
def HX(
        PlateHX_mdot_Prim, PlateHX_mdot_Sec,
        PlateHX_theta_Prim_In, PlateHX_theta_Sec_In, Par_PlateHX_AU):

    if PlateHX_theta_Prim_In > PlateHX_theta_Sec_In:
        theta_hf_in = PlateHX_theta_Prim_In  # hf: Hot fluid
        theta_cf_in = PlateHX_theta_Sec_In   # cf: Cold fluid
        Cp_hf = PlateHX_Cp_Prim*PlateHX_mdot_Prim  # Heat capacity hf
        Cp_cf = PlateHX_Cp_Sec*PlateHX_mdot_Sec    # Heat capacity cf
    else:
        theta_hf_in = PlateHX_theta_Sec_In
        theta_cf_in = PlateHX_theta_Prim_In
        Cp_hf = PlateHX_Cp_Sec*PlateHX_mdot_Sec
        Cp_cf = PlateHX_Cp_Prim*PlateHX_mdot_Prim

    DT_max = theta_hf_in-theta_cf_in # Delta max temp. in HX
    C_min = min(Cp_hf, Cp_cf) #[kW/K] Minimum Heat capacity
    C_max = max(Cp_hf, Cp_cf) #[kW/K] Maximum Heat capacity
    Cr = C_min/C_max # Capacity Rate
    NTU = Par_PlateHX_AU/C_min
    Qdot_max = C_min*DT_max
```

Die Effizienz des Wärmetauschers (*PlateHX_Eff*), die Austrittstemperaturen beider Seiten (*theta_hf_out* und *theta_cf_out*) und die Wärmeübertragung im Wärmetauscher (*PlateHX_Qdot*) werden danach wie folgt ermittelt:

```
    if Cr == 1:
       PlateHX_Eff = NTU / (1+NTU) # Effectiveness in counterflow
    else:
       PlateHX_Eff = ((1 - math.exp(-NTU * (1-Cr)))
                      / (1 - Cr*math.exp(-NTU * (1-Cr))))

    theta_hf_out = (((PlateHX_Eff*Qdot_max) / Cp_hf) - theta_hf_in)*-1 #[K]
    theta_cf_out = ((PlateHX_Eff*Qdot_max) / Cp_cf) + theta_cf_in      #[K]
    PlateHX_Qdot = (Cp_hf * (theta_hf_in-theta_hf_out)) #[kW]

    if PlateHX_theta_Prim_In > PlateHX_theta_Sec_In:
       PlateHX_theta_Prim_Out = theta_hf_out
       PlateHX_theta_Sec_Out = theta_cf_out
    else:
       PlateHX_theta_Sec_Out = theta_hf_out
       PlateHX_theta_Prim_Out = theta_cf_out

    return(PlateHX_Qdot, PlateHX_theta_Prim_Out, PlateHX_theta_Sec_Out)
```

3.6 Grundwasserpumpenmodell

Das Grundwasserpumpenmodell basiert auf einer Polynomregression dritter Ordnung gemäß der folgenden Gleichung:

- $Z = \beta_0 + \beta_1 X + \beta_2 X^2 + \beta_3 X^3$

In diesem Fall wurden die für das Grey-Box-Modell verwendeten Daten aus den Datenblattkennlinien der Grundwasserpumpen entnommen, wobei mit dem Volumenstrom *X* als unabhängige Variable die Koeffizienten β0, β1, β2 und β3 der Regressionen für die Förderhöhe (*GWP_Head_Coeff)*, den hydraulischen Wirkungsgrad (*GWP_EffHyd_Coeff)* und die Wellenleistung (*GWP_PMech_Coeff*) der jeweiligen Pumpe berechnet wurden. Mit diesen Koeffizienten ist es möglich, die Werte für die Förderhöhe (*GWP_Head*), den hydraulischen Wirkungsgrad (*GWP_EffHyd*) und die Wellenleistung (*GWP_PMech*) zu ermitteln.

```
"""-------------------------------------------------------------------------
Datasheet Data
-------------------------------------------------------------------------"""
GWP1_Vdot   = GWP1_Daten.iloc[:,0]
GWP1_Head   = GWP1_Daten.iloc[:,1]
GWP1_EffHyd = GWP1_Daten.iloc[:,3]
GWP1_PMech  = GWP1_Daten.iloc[:,4]
GWP2_Vdot   = GWP2_Daten.iloc[:,0]
GWP2_Head   = GWP2_Daten.iloc[:,1]
GWP2_EffHyd = GWP2_Daten.iloc[:,3]
GWP2_PMech  = GWP2_Daten.iloc[:,4]
"""-------------------------------------------------------------------------
Regression Coefficients: Polynomial linear regression 3rd Order
-------------------------------------------------------------------------"""
# find coefficients of the regression model
GWP1_Head_Coeff   = np.polyfit(GWP1_Vdot,GWP1_Head,3)
GWP1_EffHyd_Coeff = np.polyfit(GWP1_Vdot,GWP1_EffHyd,3)
GWP1_PMech_Coeff  = np.polyfit(GWP1_Vdot,GWP1_PMech,3)
GWP2_Head_Coeff   = np.polyfit(GWP2_Vdot,GWP2_Head,3)
GWP2_EffHyd_Coeff = np.polyfit(GWP2_Vdot,GWP2_EffHyd,3)
GWP2_PMech_Coeff  = np.polyfit(GWP2_Vdot,GWP2_PMech,3)
```

```
# polynomial regression models
GWP1_Head_Model   = np.poly1d(GWP1_Head_Coeff)     # [m]
GWP1_EffHyd_Model = np.poly1d(GWP1_EffHyd_Coeff)   # [-]
GWP1_PMech_Model  = np.poly1d(GWP1_PMech_Coeff)    # [kW]
GWP2_Head_Model   = np.poly1d(GWP2_Head_Coeff)     # [m]
GWP2_EffHyd_Model = np.poly1d(GWP2_EffHyd_Coeff)   # [-]
GWP2_PMech_Model  = np.poly1d(GWP2_PMech_Coeff)    # [kW]
```

Das Modell erhält außerdem als Eingangsdaten die Flüssigkeitsmassenströme aus dem Wärmetauschermodell, um den Volumenstrom (*GWP_Vdot*) zu ermitteln.

```
GWP_mdot_list = [sum(x) for x in zip(PlateHX3_mdot_Prim_list,
                                     PlateHX1_mdot_Prim_list,
                                     PlateHX2_mdot_Prim_list)] # [kg/s]

GWP_Vdot_list = [x/(0.28) for x in GWP_mdot_list]              # [m³/h]
```

Im Modell gibt es vier Betriebsarten, die vom Volumenstrom (*GWP_Vdot*) abhängen, der von den Wärmetauschern der Anlage benötigt wird:

- Keine Pumpe arbeitet (*GWP_Vdot* = 0)
- Nur die kleine Pumpe (GWP2) arbeitet (*GWP_Vdot* <= *GWP2_Vdot_Opt*)
- Nur die große Pumpe (GWP1) arbeitet (*GWP2_Vdot_Opt* < *GWP_Vdot* <= *GWP1_Vdot_Opt*)
- Beide Pumpen arbeiten gleichzeitig (*GWP_Vdot* > *GWP1_Vdot_Opt*)

Mit der berechneten mechanischen Leistung und dem im Datenblatt angegebenen Wirkungsgrad des Getriebes und des Elektromotors lässt sich die hydraulische Leistung (*GWP_Phydro*), die benötigte elektrische Leistung (*GWP_Pel*) und der Gesamtwirkungsgrad (*GWP_EffTot*) für alle Betriebsarten wie folgt berechnen:

```
# Optimum Operation Points - Datasheet
GWP1_Vdot_Opt = 23.91  #[m³/h]
GWP1_PMech_Opt = 2.377 #[kW]
GWP1_Pel_Opt = 3.127   #[kW]
GWP1_EffMT = GWP1_PMech_Opt/GWP1_Pel_Opt

GWP2_Vdot_Opt = 12.31  #[m³/h]
GWP2_PMech_Opt = 1.326 #[kW]
GWP2_Pel_Opt = 1.817   #[kW]
GWP2_EffMT = GWP2_PMech_Opt/GWP2_Pel_Opt

"""-------------------------------------------------------------------------
Groundwater Pump (GWP) Model
-------------------------------------------------------------------------
GWP_mdot [kg/s] --> | GWP | --> GWP_Pel: Electrical power [kW]
-------------------------------------------------------------------------"""
for i in range(Range):

    if (GWP_Vdot_list[i] == 0) or (WMZ_2_Vdot_list[i] == 0
                                   and WMZ_3_Vdot_list[i] == 0
                                   and HP_Control_list[i] == 0):
        GWP1_Control=GWP1_EffTot=GWP2_Control=GWP2_EffTot=GWP_Pel=0
```

```
    elif WMZ_2_Vdot_list[i] > 0 and HP_Control_list[i] == 0:    # Only GWP2
        GWP1_Control = GWP1_EffTot = 0
        GWP2_Control = 1
        GWP2_Head    = GWP2_Head_Model(GWP_Vdot_list[i])
        GWP2_DP      = GWP2_Head*0.09804139432 # [bar]
        GWP2_EffHyd  = GWP2_EffHyd_Model(GWP_Vdot_list[i])
        GWP2_Phydro  = ((GWP_Vdot_list[i]/3600)
                        * GWP2_Head*Prop_Water_Rho*g) / 1000    # [kW]
        GWP2_PMech   = GWP2_PMech_Model(GWP_Vdot_list[i])
        GWP2_Pel     = GWP2_PMech/GWP2_EffMT  # [kW] El. power from grid
        GWP2_EffTot  = GWP2_Phydro/GWP2_Pel   # Overall pump efficiency
        GWP_Pel      = GWP2_Pel
    elif (HP_Control_list[i] != 0) or (WMZ_2_Vdot_list[i] > 0  # Only GWP1
                                       and HP_Control_list[i] != 0):
        GWP2_Control = GWP2_EffTot = 0
        GWP1_Control = 1
        GWP1_Head    = GWP1_Head_Model(GWP_Vdot_list[i])
        GWP1_DP      = GWP1_Head*0.09804139432 # [bar]
        GWP1_EffHyd  = GWP1_EffHyd_Model(GWP_Vdot_list[i])
        GWP1_Phydro  = ((GWP_Vdot_list[i]/3600)
                        * GWP1_Head*Prop_Water_Rho*g) / 1000    # [kW]
        GWP1_PMech   = GWP1_PMech_Model(GWP_Vdot_list[i])
        GWP1_Pel     = GWP1_PMech/GWP1_EffMT   # [kW] El. power from grid
        GWP1_EffTot  = GWP1_Phydro/GWP1_Pel    # Overall pump efficiency
        GWP_Pel      = GWP1_Pel
    elif (WMZ_2_Vdot_list[i] > 0 and WMZ_3_Vdot_list[i] > 0    # GWP1+GWP2
          and HP_Control_list[i] != 0):
        # GWP2
        GWP2_Control = 1
        GWP2_Head    = GWP2_Head_Model(GWP_Vdot_list[i])
        GWP2_DP      = GWP2_Head*0.09804139432 # [bar]
        GWP2_EffHyd  = GWP2_EffHyd_Model(GWP_Vdot_list[i])
        GWP2_Phydro  = ((GWP_Vdot_list[i]/3600)
                        * GWP2_Head*Prop_Water_Rho*g) / 1000    # [kW]
        GWP2_PMech   = GWP2_PMech_Model(GWP_Vdot_list[i])
        GWP2_Pel     = GWP2_PMech/GWP2_EffMT   # [kW] El. power from grid
        GWP2_EffTot  = GWP2_Phydro/GWP2_Pel    # Overall pump efficiency
        # GWP1
        GWP1_Control = 1
        GWP1_Head    = GWP1_Head_Model(GWP_Vdot_list[i])
        GWP1_DP      = GWP1_Head*0.09804139432 # [bar]
        GWP1_EffHyd  = GWP1_EffHyd_Model(GWP_Vdot_list[i])
        GWP1_Phydro  = ((GWP_Vdot_list[i]/3600)
                        * GWP1_Head*Prop_Water_Rho*g) / 1000    # [kW]
        GWP1_PMech   = GWP1_PMech_Model(GWP_Vdot_list[i])
        GWP1_Pel     = GWP1_PMech/GWP1_EffMT   # [kW] El. power from grid
        GWP1_EffTot  = GWP1_Phydro/GWP1_Pel    # Overall pump efficiency
        GWP_Pel = GWP1_Pel + GWP2_Pel
```

Idealerweise sollte die Validierung und Anpassung der Simulationsmodelle mit ausreichenden Messdaten durchgeführt werden, um das Verhalten der verschiedenen Variablen während der vier Jahreszeiten beobachten zu können. Es handelt sich also um einen kontinuierlichen Prozess, bei dem Messungen durchgeführt werden, die Modelle mit diesen Messungen gespeist werden, die gemessenen mit den simulierten Werten verglichen und die Modelle angepasst werden. Diese Modelle müssen weiterentwickelt werden, um weitere Steuerungsebenen hinzuzufügen, wie z.B. die Stellung von Ventilen und deren Auswirkung auf das System oder die Steuerung, die Thermostate geben können. Die nächste Phase der Validierung des Simulationsmodells wird auf der Grundlage einer Energiebilanz des Systems durchgeführt, für die derzeit alle Daten der Wärme- und Stromzähler des Gebäudes erfasst werden.

Literaturverzeichnis

[1] Energieeffizienz von Gebäuden – Berechnung des Energiebedarfs für Heizung und Kühlung (ISO 13790:2008);. Ausgabe September 2008.

[2] VDI 6007-3. Ausgabe Juni 2015.

[3] *Gassel, A.:* Beiträge zur Berechnung solarthermischer und exergieeffizienter Energiesysteme – Abschnitt 2.2. Strahlungsberechnung; (1997), S. 15-27.

[4] *Sawant, P. et al.:* Development and experimental evaluation of grey-box models of a microscale polygeneration system for application in optimal control; (2020), S.7-8.