Thermalift v. 1.0

Users' Guide

Maciej Miecznik

Instytut Gospodarki Surowcami Mineralnymi i Energią Polskiej Akademii Nauk

February 2025

Thermalift tool is a part of project "Optimal management of low-temperature geothermal reservoirs – Polish-Icelandic cooperation on reservoir modelling" (acronym GeoModel), financed under the Fund for Bilateral Relations through the European Economic Area Financial Mechanism (EEA FM) and the Norwegian Financial Mechanism (NFM) 2014-2021 under the Environment, Energy and Climate Change Programme.

Table of contents

1. Introduction

In documenting geothermal water resources, it is important to properly determine the filtration parameters, mainly reservoir transmissivity. Its calculation is based on the results of pumping tests or short term production. This is the most reliable method to date for determining the filtration properties of an aquifer. Geothermal waters usually occur at great depths, where there is high temperature and pressure, therefore, these factors cause a number of interpretation difficulties. They are related, among others, to the non-isothermal flow of water in the borehole, variation of flow rate and wellhead temperature during pumping and the release of gases dissolved in water as a result of the pressure dropping below the saturation pressure. In this article, the authors address the issue of the influence of wellhead temperature variations during pumping on the recorded water level or wellhead pressure. This phenomenon is called thermal lift effect.

It often happens that in the case of deep wells with good filtration parameters of the aquifer, the shape of the drawdown curve is far from expectations, because the effect of water expansions masks the true drawdown. The deeper the well and the higher the bottomhole temperature, the more significant this effect is. In any case, the thermal lift effect always causes the drawdown to be smaller than it would be observed if there was no volumetric expansion of water. This in turn can lead to overestimation of resources and unsustainable exploitation, because the real drawdown is often miscalculated. One way to get rid of this problem is to measure the water level (or the wellhead pressure) in observation wells. Another way is to measure the bottomhole pressure. Unfortunately, both of these solutions are rarely available, especially in low- and medium-temperature sedimentary systems. It is usually simply too expensive to drill observation wells 2–3 km deep or conduct long-term bottomhole pressure monitoring. However, it is not only a good practice, but sometimes a necessity, to separate the thermal lift effect from the raw data when interpreting both short- and long-term pumping data. This allows to filter out noise, correctly assess the actual drawdown, and as a result - the correct transmissivity of the aquifer.

The thermal lift effect, to the authors' knowledge, was first mathematically described in the literature in the mid-1990s by Kawecki (1994, 1995). Later, application of his approach was applied to geothermal wells in sedimentary formations in Poland, proving its usefulness in interpreting hydrodynamic tests and long-term historical data (Bielec & Miecznik 2012, Miecznik 2017). However, for all these years there was no tool that would automate the calculations and allow for a graphical comparison of the water level before and after thermal lift correction. The opportunity to develop and publicly share such a tool appeared with the start of the GeoModel project (www.geomodel.pl/en), in which Polish and Icelandic specialists decided to develop a set of Python scripts to support the work of geothermal reservoir engineers. THERMALIFT is a Python package that allows for the correction of raw pumping data and graphical representation of results on graphs. The only input data required are

temperature and water level or wellhead pressure during pumping and the temperature profile along the well under natural conditions.

2. Theory of thermal lift

Below formulation is valid for both vertical and deviated wells with conduction as a dominant heat transfer process in the formation. These are typical conditions for majority of sedimentary formations. Figure 1 shows the temperature profile in the well in static (natural) conditions and during pumping. Under static conditions, the temperature profile along the well is the same as the temperature distribution in the rock formation. During pumping, well is gradually heated. The temperature at the wellhead is closer to the bottomhole temperature the longer the production takes or the higher the flow rate of the pumped liquid.

The hydrostatic pressure exerted by the water column is:

$$p = g \int_0^{H_{MD}} \rho_w(z) dz \tag{1}$$

where p - hydrostatic pressure, g - Earth's gravitational acceleration, ρ_w - the density of the water column in the borehole, H_{MD} – measured depth of the borehole, z - integration variable (depth).

Since the pressure exerted at the bottom of the well is independent of the temperature of the liquid inside it, the multiplication of the height of the water column and its density will be the same for the static and operating well:

$$H_d \overline{\rho_w^d} = H_s \overline{\rho_w^s} \tag{2}$$

where H_d - height of the water column in dynamic conditions, H_s - height of the water column in static conditions, $\overline{\rho_w^d}$ - average density of the water column under dynamic conditions, $\overline{\rho_w^s}$ - average density of the water column under static conditions.

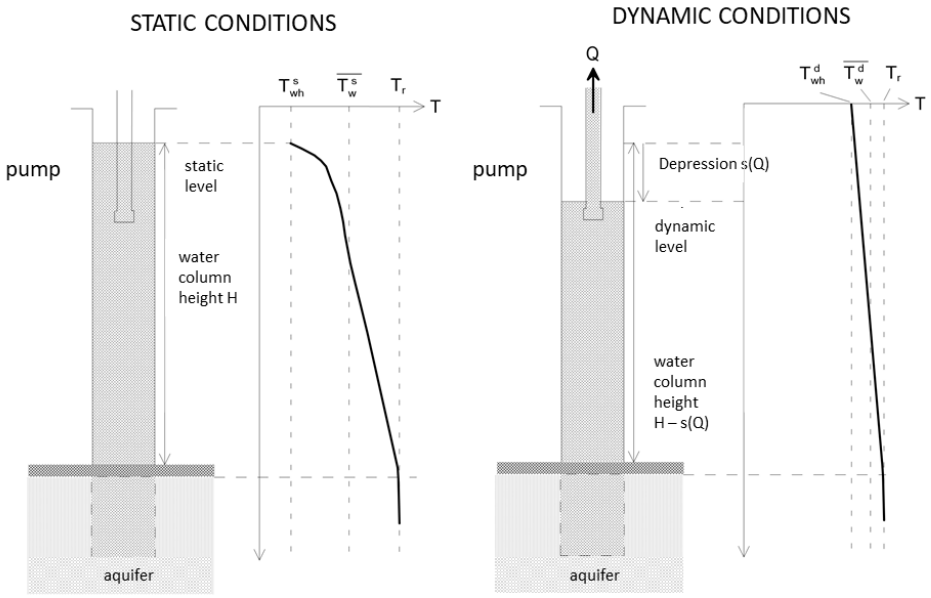

Figure 1: Diagram showing the temperature profile in the well under static and dynamic conditions (adopted from Kawecki 1995)

The average density of water in the well is as follows:

$$\overline{\rho_w} = \frac{1}{H_{MD}} \int_0^{H_{MD}} \rho_w dz \tag{3}$$

Although in general the density of water is a non-linear function of temperature, in the temperature range from 40 to 95°C (typical temperatures of geothermal water in liquid systems) the approximation by a linear function can be considered sufficiently accurate. By omitting some transformations of Equation 3 which are provided by Kawecki (1995), one obtains a relation which says that the average density of the water column is the density of water for the average (weighted) temperature of the water column:

$$\overline{\rho_w} = \rho_w\left(\overline{T_w}\right) \tag{4}$$

By inserting equation 4 into equation 2, the following relation is obtained:

$$H_d \rho_w \left(\overline{T_w^d}\right) = H_s \rho_w \left(\overline{T_w^s}\right) \tag{5}$$

where $\overline{T_w^d}$ - average water column temperature under dynamic conditions, $\overline{T_w^s}$ - average water column temperature under static conditions.

The pressure measured at the bottom of the well under static conditions is as follows:

$$p_{r,0} = \rho_w\left(\overline{T_w^s}\right) H_{TVD} g + p_{wh,0} \tag{6}$$

where $p_{r,0}$ - pressure at the bottom of the well under static conditions (no extraction), $p_{wh,0}$ - wellhead pressure under static conditions.

The bottomhole pressure during operation can be expressed in the form of equation 7, which is analogous to equation 6:

$$p_r = \rho_w \left(\overline{T_w^d}\right) H_{TVD} g + p_{wh} \tag{7}$$

where p_r - bottomhole pressure under dynamic conditions, p_{wh} - wellhead pressure under dynamic conditions.

The value of the water drawdown $s\left(\overline{T_w^s}\right)$ in a well is the difference between the static and dynamic bottomhole pressures:

$$s\left(\overline{T_w^s}\right) = \frac{p_{r,0} - p_r}{\rho_w\left(\overline{T_w^s}\right) g} \tag{8}$$

By substituting equations 6 and 7 into equation 8, one obtains a formula that allows to calculate the actual depression excluding the influence of the thermal lift, knowing only the static and dynamic wellhead pressure and the average water column temperature under static and dynamic conditions:

$$s\left(\overline{T_w^s}\right) = \frac{p_{wh,0} - p_{wh}}{\rho_w\left(\overline{T_w^s}\right) g} + \left[1 - \frac{\rho_w\left(\overline{T_w^d}\right)}{\rho_w\left(\overline{T_w^s}\right)}\right] H_{TVD} \tag{9}$$

Subtracting the actual depression value $s\left(\overline{T_w^s}\right)$ from the static wellhead pressure, taking into account the thermal lift effect during extraction, the so-called reduced wellhead pressure value p_{wh}^{red} is obtained:

$$p_{wh}^{red} = p_{wh,0} - s\left(\overline{T_w^s}\right) \cdot \rho_w\left(\overline{T_w^s}\right) \cdot g = p_{wh} - \left[1 - \frac{\rho_w\left(\overline{T_w^d}\right)}{\rho_w\left(\overline{T_w^s}\right)}\right] \rho_w\left(\overline{T_w^s}\right) H_{TVD} g \tag{10}$$

In case of the non-artesian well, following formulas are applicable. Having a pressure sensor in the submersible pump, the water level below the ground level h_m is given by equation 11:

$$h_m = h_{pump} - \frac{p - p_{atm}}{g \cdot \rho_w} \tag{11}$$

where h_{pump} - depth of the pressure sensor, p - pressure measured by the pressure sensor, p_{atm} - atmospheric pressure of 101325 Pa, g - Earth's gravitational acceleration, ρ_w - average density of the water column above the pressure sensor.

By analogy with equation 10, the reduced water level h_{red} in the production well is:

$$h_{red} = h_m + \left(1 - \frac{\rho_w(\overline{T_w^d})}{\rho_w(\overline{T_w^s})}\right) \cdot H_{TVD} \quad (12)$$

Hence, the true drawdown s_{red} in the aquifer is:

$$s_{red} = h_{red} - h_{red,\ min} \quad (13)$$

where $h_{red,\ min}$ is the water level (b.g.l.) at the closed wellhead.

3. Structure of Thermalift package

3.1. System requirements

Following packages are needed to run Thermalift code:

```
matplotlib==3.10.1
numpy==2.2.3
pandas==2.2.3
scipy==1.15.1
```

They can be installed with the following command:

```
pip install matplotlib==3.10.1 numpy==2.2.3 pandas==2.2.3 scipy==1.15.1
```

3.2. Folders and files tree

`Thermalift` package contains 2 subdirectories:

`lib`: contains all necessary modules, i.e. `thermalift.py` and brine_density.py

`examples`: example Python files with input csv or Excel files to show how `Thermalift` package works in practice.

3.3. The main module thermalift.py

Module `thermalift.py` contains classes and functions which allows to create objects, apply methods (functions) to them, create plots and save files.

The are two classes defined in this module:

`class Formation:` is used to create **`Formation`** class objects for evaluating the temperature profile in geothermal well in the natural state

`class Well:` is used to create **`Well`** class objects, where different methods are used to eliminate the thermal lift effect from the pumping data, allowing to calculate the corrected water level, corrected drawdown or reduced wellhead pressure.

Below is the list of methods that can be applied to object belonging to both classes.

3.3.1. class Formation

```
__init__(formation_temp_file)
```

`__init__` method is used to initialize objects of a `class Formation`. It is also called a constructor. `__init__` method doesn't return value.

Parameters:

- `formation_temp_file [string]`: name of the *.csv file

```
read_formation_data()
```

Reads a 2 or 3 columns `formation_temp_file` which stores natural temperature profile of the formation vs the true vertical depth (TVD) of the well and/or measured depth (MD). Seperator is semicolon sign ";", while the dot sign "." is decimal point.

Return type:

`[DataFrame] object`

```
temp_interpolation(zmin, zmax, dz, method="cubic")
```

In case of missing temperature recordings, interpolates them between `zmin` and `zmax` and interval `dz`.

Parameters:

- `zmin` [float]: lowest depth point for temperature interpolation
- `zmax` [float]: uppermost point for temperature interpolation
- `dz` [float]: depth interval used for interpolation
- `method [string]`: interpolation method used by `pandas.DataFrame.interpolate()` method; default "cubic"

Return type:

`[DataFrame]` object

```
formation_temperature_plot(formation, formation_new)
```

Plots a temperature profile of the formation in the natural state, both the raw and interpolated data.

Parameters:

- `formation [DataFrame]`: original temperature profile of the formation

- `formation_new [DateFrame]`: interpolated temperature profile of the formation

Return type:

Matplotlib object

3.3.2. class Well

```
__init__(pumping_input, name="", salinity=0.0)
```

`__init__` method is used to initialize objects of a `class Well`. `__init__` method doesn't return value.

Parameters:

- `pumping input [DataFrame]: Pandas DataFrame` containing pumping data, of which datetime, pumping rate, water level and water temperature are minimum required columns
- `name [string]: name of the well, optional, default ""`
- `salinity [float]:` total dissolved solids (salinity); can be expressed by either relative concentration (kg of NaCl/kg H_2O) or absolute concentration (g NaCl/dm^3 H_2O). If `salinity < 0.35`, then relative concentration is used, otherwise (`salinity ≥ 0.35`) absolute concentration is used; default 0.0.

```
temp_static(formation_temp)
```

Returns mean temperature of the formation along the well's track.

Parameters:

- `formation_temp [DataFrame]: DataFrame` containing temperature of the rock formation at equal depth intervals

Return type:

`[float]`

```
temp_flowing(formation_temp, temp_col, pumping_input, wellhead_temp_col)
```

Returns mean temperatures of the water column in the flowing well.

Parameters:

- `formation_temp [DataFrame]: Pandas DataFrame` containing temperature of the rock formation at equal depth intervals
- `temp_col [int]:` column number containing formation temperature data

- `pumping_input [DataFrame]: Pandas DataFrame` containing pumping data, of which datetime, pumping rate, water level and water temperature are minimum required columns
- `wellhead_temp_col [int]:` column number containing wellhead temperature data

Return type:

`[Pandas Series]`

```
dens_static(mean_temp_static, salinity)
```

Returns mean density of water in non-flowing well (mean water temperature of the water column in natural conditions).

Parameters:

- `mean_temp_static [float]:` mean temperature of the formation along the well's track
- `salinity [float]:` total dissolved solids (salinity); can be expressed by either relative concentration (kg of NaCl/kg H_2O) or absolute concentration (g NaCl/dm^3 H_2O). If `salinity < 0.35`, then relative concentration is used, otherwise (`salinity ≥ 0.35`) absolute concentration is used; default 0.0.

Return type:

`[Float]`

```
dens_dynamic(mean_temp_flowing, salinity)
```

Returns mean densities of water in the flowing well.

Parameters:

- `mean_temp_flowing [float]:` mean temperature of the water column in the flowing well
- `salinity [float]:` total dissolved solids (salinity); can be expressed by either relative concentration (kg of NaCl/kg H_2O) or absolute concentration (g NaCl/dm^3 H_2O). If `salinity < 0.35`, then relative concentration is used, otherwise (`salinity ≥ 0.35`) absolute concentration is used; default 0.0.

Return type:

`[Pandas Series] object`

```
water_level(pressure_level, flow_dens, probe_depth, atmo_pressure)
```

Converts pressure sensor readings to depth to the water table.

Parameters:

- `pressure_level [Pandas Series]:` pressure sensor readings
- `flow_dens [Pandas Series]:` mean densities of water in the flowing well
- `probe_depth [float]:`depth to the pressure sensor, expressed usually in meters b.g.l.
- `atmo_pressure [float]:`average atmospheric pressure in Pa

Return type:

`[Pandas Series] object`

```
measured_drawdown(water_level)
```

Returns water table drawdown calculated for the raw pumping data.

Parameters:

- `water_level [Pandas Series]:` calculated depth to the water table before elimination of the thermal lift effect

Return type:

`[Pandas Series] object`

```
true_water_level(water_level, flow_dens, stat_dens, depth_max)
```

Returns corrected water level that would be measured without thermal lift effect.

Parameters:

- `water_level [Pandas Series]:` calculated depth to the water table before elimination of the thermal lift effect
- `flow_dens [Pandas Series]:` mean densities of water in the flowing well
- `stat_dens [float]:` mean density of water in non-flowing well
- `depth_max [float]: true vertical depth of the well`

Return type:

`[Pandas Series] object`

```
true_drawdown(true_water_level):
```

Returns corrected water table drawdown, after elimination of the thermal lift effect caused by water thermal expansion.

Parameters:

- `true_water_level [Pandas Series]:` corrected water level that would be measured without thermal lift effect

Return type:

`[Pandas Series] object`

```
polynomial(x, a, b, c)
```

Returns instance of 2nd degree polynomial.

Parameters:

- `x []:` polynomial variable
- `a [float]:` 2nd degree polynomial coefficient
- `b [float]:` 1st degree polynomial coefficient
- `c [float]:` 0th degree polynomial coefficient

Return type:

`Model instance`

```
polyfit(x, y, bounds=True)
```

Returns the polynomial coefficients that best fit the data.

Parameters:

- `x [Pandas Series]:` independent variable, i.e. flow rate
- `y [Pandas Series]:` dependent variable, i.e. water drawdown
- `bounds [bool]:` bool value, either `True` or `False`. If `True`, set bounds on polynomial coefficients; in case of drawdown = f(flow rate), all polynomial coefficients must be non-negative; defaults to `True`

Return type:

`1D array of floats`

```
r_square(x, y)
```

Returns the coefficient of determination R^2 of model fit (polynomial) to data.

Parameters:

- `x [Pandas Series]:` independent variable, i.e. flow rate
- `y [Pandas Series]:` dependent variable, i.e. water drawdown

Return type:

`[float]`

```
rmse(x, y)
```

Returns the root mean square error (RMSE) of model fit (polynomial) to data.

Parameters:

- `x [Pandas Series]:` independent variable, i.e. flow rate
- `y [Pandas Series]:` dependent variable, i.e. water drawdown

Return type:

`[float]`

```
save_results(filename)
```

Exports `DataFrame` to Excel file.

Parameters:

- `filename [string]:` name of the *.xlsx file

Return type:

*.xlsx disk file

```
raw_data_plot(time, flowrate, temperature, water_level, title="",
figsize=(16, 10))
```

Creates Matplotlib subplot object consisting of 3 subplots stacked horizontally and saves it on a disk as a *.png file.

Parameters:

- `time [Pandas Series]:` time series of raw pumping data
- `flowrate [Pandas Series]:` flow rate series of raw pumping data
- `temperature [Pandas Series]:` temperature series of raw pumping data
- `water_level [bool]:` converted pressure sensor readings to depth to the water table
- `title [string]:` fraction of the *.png filename
- `figsize [tuple]:` size of the displayed figure, default `(16, 10)`

Return type:

`Matplotlib` object, *.png disk file

```
mosaic_plot(time, flowrate, temperature, density, water_level,
corrected_water_level, recorded_drawdown, corrected_drawdown,
bounds=True, show_fit=False, title="", figsize=(25, 12))
```

Creates Matplotlib mosaic object consisting of 5 subplots stacked horizontally, 1 subplot stacked vertical and saves it on a disk as a *.png file.

Parameters:

- `time [Pandas Series]`: time series of raw pumping data
- `flowrate [Pandas Series]`: flow rate series of raw pumping data
- `temperature [Pandas Series]`: temperature series of raw pumping data
- `density [Pandas Series]`: calculated densities of water in the flowing well
- `water_level [Pandas Series]`: converted pressure sensor readings to depth to the water table
- `corrected_water_level [Pandas Series]`: corrected water level that would be measured without thermal lift effect
- `recorded_drawdown [Pandas Series]`: water table drawdown calculated for the raw pumping data
- `corrected_drawdown [Pandas Series]`: corrected water table drawdown, after elimination of the thermal lift effect
- `bounds [bool]`: bool value, either `True` or `False`. If `True`, set bounds on polynomial coefficients; in case of drawdown = f(flow rate), all polynomial coefficients must be non-negative; defaults to `True`
- `show_fit [bool]`: bool value, whether to show or not polynomial curve fit with statistics (R^2 and RMSE)
- `title [string]`: fraction of the *.png filename
- `figsize [tuple]`: size of the displayed figure, default `(16, 10)`

Return type:

Matplotlib object, *.png disk file

4. Example of using the Thermalift calculator

Below example shows step by step use of the Thermalift calculator. Following steps are included:

a) Import of files containing raw data;

b) Data manipulation: filtering, interpolation, removing duplicated columns, renaming columns,
c) Converting time series of the imported data to `Pandas.Datetime` format;
d) Creation of `Formation` and `Well` classes objects;
e) Calling different methods that can be applied to `Formation` and `Well` classes objects in order to calculate the corrected water table level or corrected water table drawdown;
f) Plotting graphs and saving them to disk.

```
import sys
import pandas as pd
import thermalift

# adding lib folder to the system path
sys.path.insert(0, "..\..\lib")


# %% Import static temperature profile of the formation
# Stargard GT-7 well
rock_formation_temperature_file = "formation_temperature.csv"
rock_formation = thermalift.Formation(rock_formation_temperature_file)

rock_formation.data = rock_formation.read_formation_data()
rock_formation.profile = rock_formation.temp_interpolation(0, 3000, 30)
rock_formation.formation_temperature_plot(
    rock_formation.data, rock_formation.profile
)


# %% Import pumping data, remove duplicated columns, filter incorrect data
test_well_zenith = pd.read_csv("zenith_measerements.csv", sep=";",
decimal=",")

# Removes duplicate columns with measurement date and time
test_well_zenith = test_well_zenith.drop(
    columns=[
        "ST3_SC_GT6_1_TempWej - Czas",
        "ST3_SC_GT7_1_CisnWej - Czas",
        "ST3_SC_GT7_1_TempWej - Czas",
        "ST1_FIT_GT6_1_Wart - Czas",
        "ST2_FIT_GT7_1_Wart - Czas",
    ]
)

# Changing column names to shorter ones
test_well_zenith.rename(
    columns={
        "ST3_SC_GT6_1_CisnWej - Czas": "time",
        "ST3_SC_GT6_1_CisnWej": "gt6_pressure",
        "ST3_SC_GT6_1_TempWej": "gt6_temperature",
        "ST3_SC_GT7_1_CisnWej": "gt7_pressure",
        "ST3_SC_GT7_1_TempWej": "gt7_temperature",
        "ST1_FIT_GT6_1_Wart": "gt6_flow",
        "ST2_FIT_GT7_1_Wart": "gt7_flow",
    },
    inplace=True,
```

```
)

# Converting a time column (type: string) to type 'datetime64[ns]
test_well_zenith["time"] = pd.to_datetime(
    test_well_zenith["time"], errors="raise", dayfirst=True
)

# Data filtering: only measurement points before the Zenith probe failure
test_well_zenith = test_well_zenith[
    test_well_zenith["time"] <= "2022-10-05 12:00:00"
]

# Remove incorrect measurement from 2022-02-05 14:00:00
test_well_zenith = test_well_zenith.drop(test_well_zenith.index[9608])


# %% Create Well class object and perform calculations
test_well = thermalift.Well(test_well_zenith)

# Add properties to the object
test_well.salinity = 126

# Calculate mean static temperature in the wellbore
test_well.mean_stat_temp = test_well.temp_static(rock_formation.profile)

# Calculate mean dynamic temperature in the flowing well
test_well.mean_flowing_temp = test_well.temp_flowing(
    rock_formation.profile, 1, test_well_zenith, 4
)

# Calculate mean water column density in a non-flowing well
test_well.mean_stat_dens = test_well.dens_static(
    test_well.mean_stat_temp, test_well.salinity
)

# Calculate mean water column density in flowing well
test_well.mean_flowing_dens = test_well.dens_dynamic(
    test_well.mean_flowing_temp, test_well.salinity
)

# Calculate water level in flowing well
test_well.water_level = test_well.water_level(
    test_well.pumping_input.iloc[:, 3],
    test_well.mean_flowing_dens,
    probe_depth=272.28,
    atmo_pressure=101325,
)

# Calculate measured drawdown
test_well.recorded_drawdown = test_well.measured_drawdown(
    test_well.water_level
)

# Calculate true water level, after eliminating thermal lift
test_well.true_water_level = test_well.true_water_level(
    test_well.water_level,
    test_well.mean_flowing_dens,
    test_well.mean_stat_dens,
    depth_max=2700.0,
)
```

```
# Calculate true drawdown, after eliminating thermal lift
test_well.true_drawdown =
test_well.true_drawdown(test_well.true_water_level)

# Save results to file
test_well.save_results(filename="results.xlsx")

# %% FIGURES
test_well.raw_data_plot(
    test_well.pumping_input.iloc[:, 0],
    test_well.pumping_input.iloc[:, 5],
    test_well.pumping_input.iloc[:, 4],
    test_well.pumping_input.iloc[:, 3],
    title="Test well raw data",
)

test_well.mosaic_plot(
    test_well.pumping_input.iloc[:, 0],
    test_well.pumping_input.iloc[:, 6],
    test_well.mean_flowing_temp,
    test_well.mean_flowing_dens,
    test_well.water_level,
    test_well.true_water_level,
    test_well.recorded_drawdown,
    test_well.true_drawdown,
    bounds=True,
    show_fit=True,
    title="Test well raw and corrected pumping data",
    figsize=(25, 12),
)
```

Output graphs

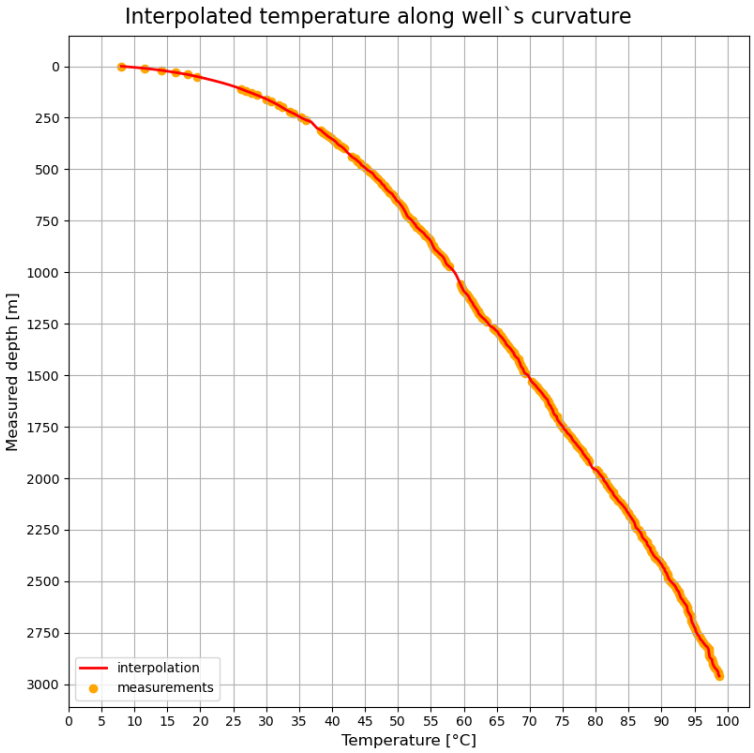

Figure 2: Plot of the rock formation temperature along the well's trajectory. Missing data were interpolated to accurately estimate the mean temperature of the rock formation.

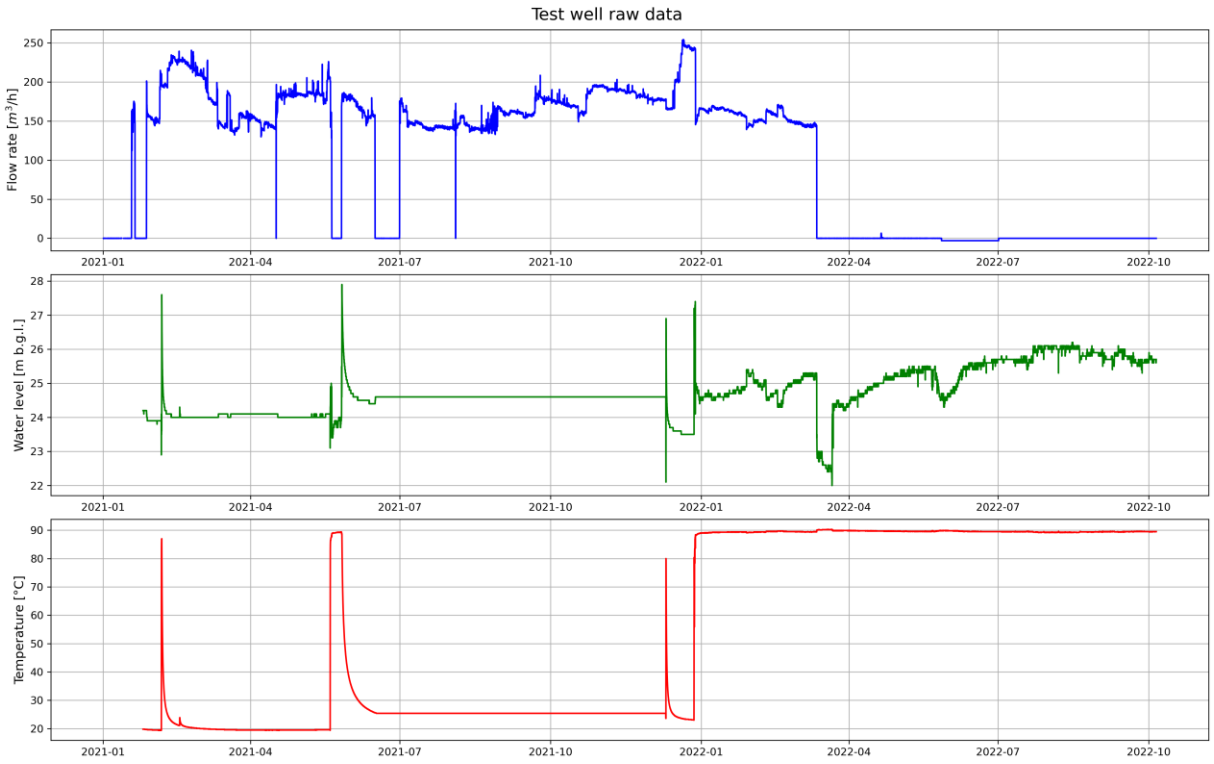

Figure 3: Plot of time series of the flow rate, depth to water table and measured wellhead temperature over a period of 21 months

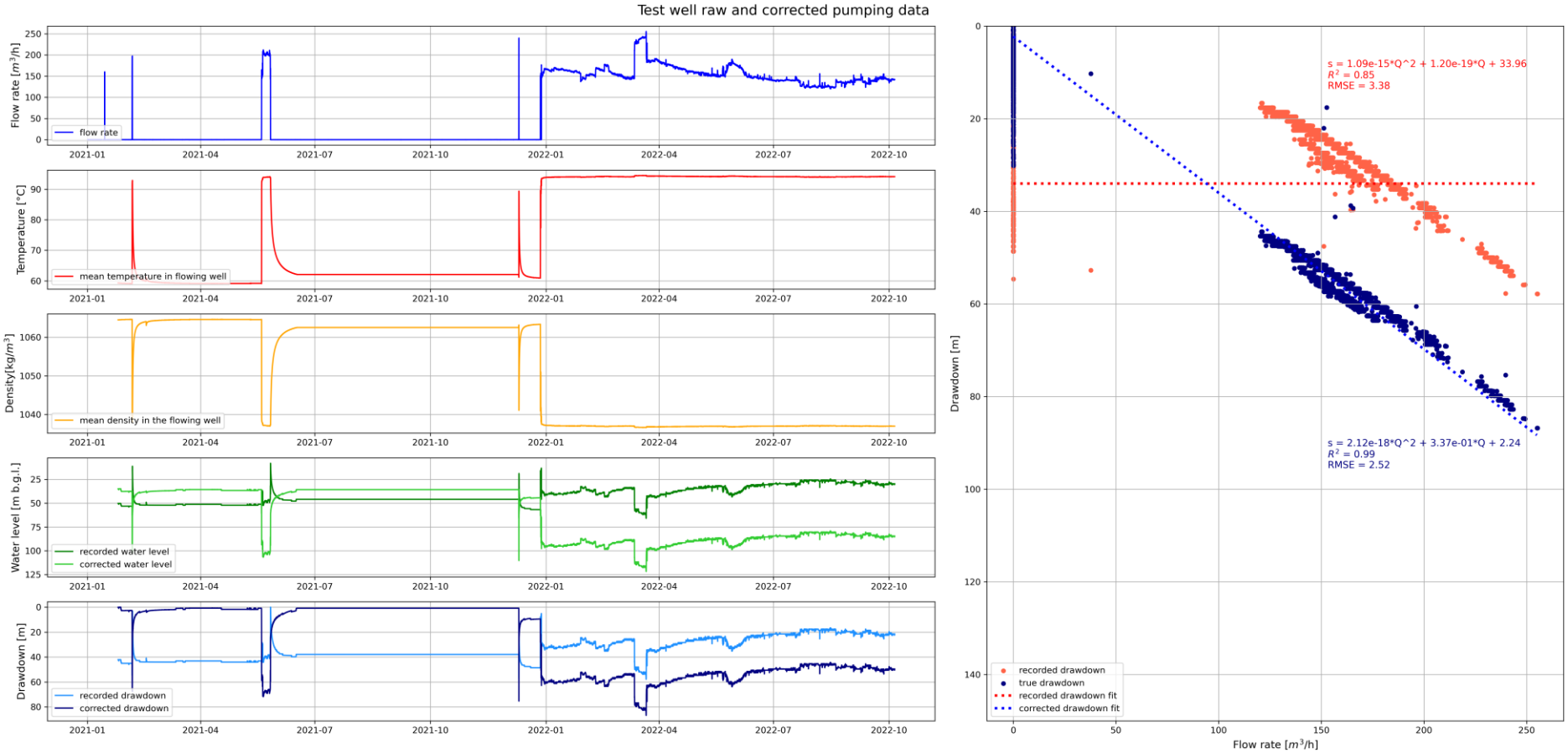

Figure 4: A 21-month time series graph comparing raw pumping data with the thermal lift-adjusted data

List of references

Bielec, B. & Miecznik, M., Efekt termiczny w obliczeniach przewodności hydraulicznej w otworach ujmujących wodę termalną, Technika Poszukiwań Geologicznych, Geotermia, Zrównoważony Rozwój, 51, 2, (2012), 45-54.

Kawecki, M.W., Temperature effect in discharge tests on deep water wells, Proceedings Second Gulf Water Conference, Bahrain, v.2 , 179-190 (1994).

Kawecki, M. W., Correction for Temperature Effect in the Recovery of a Pumped Well, Ground Water, 33, 6, (1995), 917-926.

Miecznik, M., Model zrównoważonej eksploatacji zbiornika wód geotermalnych w centralnej części Podhala do produkcji energii cieplnej i elektrycznej, Book nr 202, IGSMiE PAN, Krakow, Poland (2017).

Sun, H., Feistel, R., Koch, M. and Markoe, A., New equations for density, entropy, heat capacity, and potential temperature of a saline thermal fluid, Deep-Sea Research, I 55 (2008), 1304–1310.