Cookbook for Python Tools for Time-series Analysis of Subsurface Thermal Records to Infer Fluid Flow Rates

T. Wu[1] and A. T. Fisher[2]

[1] Applied Mathematics, University of California, Santa Cruz

[2] Earth and Planetary Sciences, University of California, Santa Cruz

Version 1.01, last updated 8/6/25

Document prepared as part of a CITRIS Workforce Innovation Program internship

User Guide to Analyzing Time-series Thermal Data to Derive Seepage Rates

I. Overview and Goals

A. Introduction and Purpose

The purpose of programs being developed is to calculate rates of vertical fluid seepage (downward or upward) in saturated sediments, as might be found below rivers, streams, infiltration basins, wetlands, and other hydrologic features. There are numerous applications for this kind of analysis, including assessment of surface water–groundwater interactions, rates of groundwater recharge, and links with biogeochemical transformations that impact water quality. The goal for this software development is to improve a workflow associated with a general data-analysis approach introduced by UCSC researchers (Hatch et al., 2006), which has since been adopted, modified, and applied in numerous studies.

Most data to be analyzed was collected with a single "thermal probe" that contains two separate data loggers and sensors, programmed and operated autonomously, with no real-time data transmission or control. The UCSC Hydrogeology group builds the thermal probes using PVC tubing and fittings, and suspends data loggers inside the tubing on braided cord, holding the sensors at a fixed spacing; 0.15 m separation typically works well, placing sensors 5 cm and 20 cm below the streambed or base of an infiltration basin. Data loggers used in thermal probes are programmed to start operating at a specified time and with a fixed data collection interval (usually 15 or 20 minutes). Probes are deployed and left in the field for weeks to months at a time, then recovered and returned to the lab so that data can be downloaded, processed, and interpreted.

B. Software Overview

The analysis workflow is supported by a set of custom-built Python scripts and Dash-based applications. Each tool in the suite was designed and authored to perform a focused step in the processing of temperature time-series data collected from dual-sensor thermal probes. These tools work together to guide the user from raw logger data to final seepage rate calculations with built-in logging, plotting, and error handling at each step.

Tools in the Suite:

1. **tts_mergpars.py / tts_mergpars_leg.py**

   These scripts read CSV files from shallow and deep loggers, and merges the data into a unified time series using a decimal "WaterDay" format. The primary version (**tts_mergpars.py**) provides a web-based interface (using the same dash functionality as other scripts) to view and select stable intervals for further analysis. Output is one or more composite files with matched time and temperature records from a pair of temperature loggers, each of which records data from a different depth. The legacy version (**tts_mergpars_leg.py**) was developed before we began using dash, is retained for completeness, but there is little need for use during standard processing.

2. **tts_freqfilt.py**

   This tool performs frequency analysis (e.g., using standard signal processing methods) to visualize the dominant frequencies in a set of temperature data, allowing the user to decide how best to filter the data. User selects a frequency band (typically around 1 cycle/day), which is applied to observational data using a bandpass filter; additional parameters are also used to determine how the filter is to be applied. The data is resampled after filtering (typically to 1-minute resolution), which can help to improve subsequent processing.

3. **tts_peakpick.py**

   This script detects and visualizes the local temperature peaks in filtered data sets from both shallow and deep sensors. Several peak-picking algorithms are available, and users can interactively clean up peak selections. The tool has some error checking to avoid problems with subsequent processing. The tool outputs matched peak time and temperature values needed for the next step.

4. **tts_seepage.py**

   This script calculates vertical fluid seepage rates by analyzing differences between shallow and deep temperature peaks. It uses both amplitude and phase information to estimate vertical fluid seepage rates, once per cycle (typically once per day), and includes multiple nonlinear solvers and error-checking routines. If the uncertainties package is

installed, it can also propagate potential errors in parameter selection through to the final seepage estimates.

All tools are designed to run from a common working directory and are configured using .par (parameter) files, plain-text files that define filenames, logger spacing, filter bands, and other options. Each tool also generates a detailed log of all operations, making results easier to reproduce and debug.

C. Software Setup and Required Libraries

To run this full suite of applications, including data parsing, filtering, peak picking, and seepage estimation, the Python and additional libraries must be installed. These packages support GUI-based interaction, data processing, plotting, and numerical modeling.

These applications were created on Visual Studio, Windows 10, using these specific library versions noted below, see *VI. Appendix For Additional Information*. We have done some additional testing on Mac Intel and Silicon machines, running OS 12.7 and OS 13.7, mostly with positive results.

Recommended Python Version: Python 3.12.2

All scripts were developed and tested on this version. Other 3.10+ versions may work but are not guaranteed. **NB:** We have confirmed that Python 3.9 will not work, as there are small differences in expected syntax that cause errors. We have also encountered some problems with dependencies between libraries when using Python 3.9.

pandas==2.2.2
numpy==1.26.4
matplotlib==3.9.1
dash==3.1.1
dash-bootstrap-components==2.0.3
scipy==1.14.0
PyWavelets==1.8.0
mne==1.9.0
uncertainties==3.2.3

numba==0.61.2

plotly==6.2.0

Install All Required Packages (One Line)

If you're using **pip**, run this command in your terminal or environment:

pip install dash==3.1.1 dash-bootstrap-components==2.0.3 plotly==6.2.0 pandas==2.2.2 numpy==1.26.4 matplotlib==3.9.1 scipy==1.14.0 uncertainties==3.2.3 PyWavelets == 1.8.0 numba==0.61.2

macOS, Jupyter Notebook, or Anaconda Users

Depending on your setup, here are some alternate install options:

conda install -c conda-forge dash dash-bootstrap-components plotly pandas numpy matplotlib scipy uncertainties pywavelets numba

For JupyterLab/Notebook users:
Make sure you're in the right environment (or kernel), then run:

!pip or conda install dash==3.1.1 dash-bootstrap-components==2.0.3 plotly==6.2.0 pandas==2.2.2 numpy==1.26.4 matplotlib==3.9.1 scipy==1.14.0 uncertainties==3.2.3 PyWavelets == 1.8.0 numba==0.61.2

If you're using **Jupyter on macOS**, the same commands apply. If you get permissions errors, you could try prepending !pip/conda install ... with !pip/conda install --user ....

Libraries Used by Each Script

Script	Libraries Used

tts_mergpars.py	numpy, pandas, dash, dash-boostrap-components
tts_mergpars_leg.py [a]	pandas, numpy, matplotlib
tts_freqfilt.py	numpy, pandas, plotly, dash, dash-bootstrap-components, scipy, optional: MNE*
tts_peakpick.py	dash, plotly, pandas, numpy, scipy, pywavelets
tts_seepage.py	dash, plotly, pandas, numpy, scipy, optional but recommended for best results: uncertainties*, numba*

[a] This is a legacy version of tts_mergpars, created before we started using dash. It has no functionality beyond that available with the script that uses dash, tts_mergpars.py.

System Dependencies

tkinter

- Usually comes with Python 3.12 by default
- If you're on **Linux** and get import errors, you might need:
    - Ubuntu/Debian
        - sudo apt-get install python3-tk
    - CentOS/RHEL/Fedora
        - sudo yum install tkinter
    - or
        - sudo dnf install python3-tkinter

Notes:

You must have an internet connection when installing packages unless you're working offline with pre-downloaded wheels. If you're using an older version of Python, not all packages may

install correctly, and/or errors will be encountered during execution. Dash apps are run in the browser (locally), so a default web browser must be available.
It should be possible to run all four applications simultaneously if you run them on four different terminals, resulting in opening of four browser windows. This may be useful for "recursive" processing, returning to earlier steps to update parameters or data selection.

II. tts_mergpars.py: Open, check, view, merge, parse, output thermal data

A. Overview

This program is the first in a suite of Python tools designed to process thermal time-series data collected from a dual-logger "thermal probe" — typically used in hydrogeologic settings such as rivers, wetlands, or infiltration basins. tts_mergpars.py is a graphical Python application built with Dash, and it replaces the legacy script, **tts_mergpars_leg.py**, by integrating all its functionalities in a more accessible and interactive interface. The use of dash is also consistent with the other Python tools developed for this project.
tts_mergpars_dash.py allows users to:
Load raw data from shallow and deep temperature loggers

- View full records in an interactive plot
- Zoom, pan, and explore the data visually
- Select specific time intervals for analysis
- Export data from selected intervals as merged “composite” CSV files, used for subsequent processing and analysis

This tool automates several key preprocessing steps that were formerly done in MATLAB or manually. It's the foundation for the entire seepage analysis workflow, so careful data checking and export here helps reduce errors in all downstream steps.

> **Note**: It is assumed the user is working within a configured Python environment.
> See the *Overview section* for required dependencies and setup instructions.

B. Input Data File Structure

The program expects two raw CSV files (shallow and deep) as input. These are exported from loggers (or similar devices) and follow this basic structure:

```
"Plot Title: PR24_TP04-S"
"#","Date Time, GMT-08:00","Temp, °C (LGR S/N: 20444286, ...)","Other Metadata...",...
1,08/08/24 12:00:00 AM,20.246,...
2,08/08/24 12:20:00 AM,20.150,...
...
```

You should prepare these files with meaningful filenames like:

PR24_TP04-S.csv ← Shallow logger

PR24_TP04-D.csv ← Deep logger

The script expects:

- Column 1: Row Number
- Column 2: Timestamp (MM/DD/YY HH:MM:SS AM/PM)
- Column 3: Temperature (typically in °C, but °F is also accepted with conversion)
- Columns 4–X: Ignored metadata (filtered automatically)

C. Parameter File (tts_mergpars.par)

This file tells the program which files to load and how to interpret the data. It should be saved in the working directory and include:

filename_shallow = PR24_TP04-S.csv

filename_deep = PR24_TP04-D.csv

water_year = 2024

interval_minutes = 20

convert_to_celsius = 0

- filename_shallow: Path to shallow logger file
- filename_deep: Path to deep logger file

- water_year: Reference year to anchor WaterDay time conversion (starts at Oct 1, YYYY-1)
- interval_minutes: Sampling interval (typically 15 or 20 minutes)
- convert_to_celsius: 0 = no conversion; 1 = convert from °F to °C

If data were recorded in °F or contain AM/PM formatting issues, the script handles them internally as long as the parameters are set correctly.

D. Steps for typical workflow

- **Start Application**
  - Run in terminal: python tts_mergpars.py
  - Browser opens at: http://127.0.0.1:8050
- **Load Data**
  - **Option 1**: Upload .par file → Click "Load from .par" for both shallow and deep datasets
  - **Option 2**: Directly upload shallow and deep CSV files via upload boxes
- **Verify Parameters**
  - Check "Water Year," "Data Interval," and temperature conversion in the **left control panel**.
  - Adjust as needed before processing.
- **Process Data**
  - Click **"Process Data"** to:
    - Convert timestamps to **Water Days**.
    - Validate date ranges for the specified water year.
    - Merge shallow and deep records using nearest-neighbor matching.
    - Detect and report data gaps.
- **Visual Inspection**
  - Explore the merged time series with **zoom, pan, and hover**.
  - Enable optional "Temperature Difference" view if you want Waterdays to start from 0.
- **Select Water Day Range**

    - **Graphical**: Click "Select Water Days" → choose start and end points on the plot.
    - **Manual**: Enter *Start WD* and *End WD* in numeric fields → click "Apply Range."
- **Export Data**
    - Enter output filename and folder (default: processed/).
    - **Export Options**:
        - Full dataset (leave selection blank) - generally best to do this first, before you start selecting data ranges.
        - Selected range only.
    - **Outputs:**
        - {filename}.csv (merged composite)
        - {filename}-S.wyo (shallow)
        - {filename}-D.wyo (deep)
- **Session Logging**
    - All actions and warnings are recorded in logs/tts_mergpars_session_YYYYMMDD_HHMMSS.log.

III. tts_freqfilt: Finding dominant frequencies, filtering composite data files

A. Overview

tts_freqfilt.py is an interactive, browser-based application for **frequency analysis and filtering** of thermal time-series data. It identifies dominant frequency components (e.g., diurnal cycle near 1 day^{-1}) in merged shallow–deep datasets and applies band-pass filters to isolate signals of interest.

Key features:

- **Power Spectral Density (PSD)** analysis with multiple methods: MATLAB-style Multitaper (default), Welch, and smoothed Welch. Other options could be added.
- **Graphical frequency band selection** directly from the PSD plot (5-decimal precision).
- **Configurable filtering** (Butterworth default) with ramp/taper transitions and adjustable filter order.
- **Trend removal options** (DC offset, linear, polynomial, high-pass, moving average).
- **High-resolution resampling** (e.g., 20 min → 1 min) for improved peak/trough detection.

- **Direct export** of filtered data in CSV format, compatible with peakpicker and seepage analysis tools.
- **Comprehensive logging** of all operations for reproducibility and troubleshooting.

B. Input Data File Structure

One composite CSV file is required, typically generated by tts_mergpars.py or tts_mergpars_leg.py, with this format:

```
WaterDay,TempShallow,TempDeep
312.000000,20.246000,21.199000
312.013889,20.150000,21.151000
```

...

- **WaterDay** – Decimal day of water year.
- **TempShallow / TempDeep** – Temperatures in °C (°F supported if converted upstream).
- Alternative column names (Shallow.Temp, Deep.Temp) are recognized automatically.
- Optional WaterDay limits (wd_lower_limit, wd_upper_limit) can clip data for analysis.

C. Parameter File (tts_freqfilt.par)

The .par file sets default values and paths; parameters can be edited in the GUI and re-saved. Example contents:

```
filename_composite_data = PR24_TP04-SD1.csv
data_interval_minutes = 20
time_bandwidth_parameter = 4
wd_lower_limit = -1
wd_upper_limit = -1
start_band_pass = 0.8
end_band_pass = 1.2
ramp_fraction = 0.1
filter_order = 3
resample_interval = 1
output_folder = FilteredData
```

Parameter descriptions:

- **filename_composite_data** – Input CSV filename.
- **data_interval_minutes** – Original sampling interval.
- **time_bandwidth_parameter** – Frequency resolution control for multitaper methods.
- **wd_lower_limit / wd_upper_limit** – Optional WaterDay range for analysis (-1 = full range).
- **start_band_pass / end_band_pass** – Frequency bounds (day^{-1}).
- **ramp_fraction** – Proportion of band used for ramp up/down (0–0.5).
- **filter_order** – Filter sharpness (default = 3, see Filter Order Guide in README).
- **resample_interval** – Output interval in minutes (1 min recommended for 20:1 improvement).
- **output_folder** – Directory for filtered output files.

D. Steps For Typical Workflow

- **Start Application**
  - python tts_freqfilt.py
  - Browser opens at http://127.0.0.1:8051
- **Load Data**
  - **Load Default File** (from .par) or
  - **Browse Different File** (native file dialog) or

    Drag and Drop into the upload area.

    Raw shallow–deep temperature plot appears immediately for making sure right file is chosen.
  - **Known Bugs****
    - On MacOS/Jupyter, there might errors that could cause the program to crash if one tries to use the blue "Browse Different File" button. To avoid the crash, use the “Drag and Drop or Select Files” option to browse for files instead of the blue “Browse Different File” button.
- **Inspect PSD**
  - Select PSD method (default: MATLAB Multitaper).
  - Adjust method-specific parameters (e.g., time bandwidth, segment length).

    - PSD updates automatically on parameter changes.
- **Select Frequency Band**
    - Click “Enter Frequency Selection Mode.”
    - Click twice on PSD plot to set **lower** and **upper** bounds.
    - Bounds populate the filter parameter fields.
- **Set Filter Parameters**
    - Choose filter type (Butterworth default).
    - Adjust filter order (recommend starting at 3) and ramp fraction.
    - Optional: enable trend removal before filtering.
    - Validate parameters (warnings shown if range/order unsuitable).
- **Apply Filter and Review**
    - Filtered data plotted below raw data for visual comparison.
    - Adjust parameters and re‑filter as needed.
- **Resample (Optional)**
    - Default is 1‑minute resampling for higher resolution.
    - Improves accuracy of amplitude and phase calculations.
- **Export Results**
    - Set output folder and filename.
    - Click “Save Filtered Data” to save CSV in format:

      WaterDay,Shallow.Temp.Filt,Deep.Temp.Filt
    - All parameter values and filter characteristics logged for reproducibility and saved into a log folder in the same directory as tts_freqfilt.

IV. tts_peakpick: Picking peaks across the filtered data

A. Overview

tts_peakpick.py is an interactive dash‑based tool for identifying and validating peaks and troughs in filtered shallow–deep thermal probe datasets. It supports multiple detection algorithms (SciPy, Wavelet, Derivative, Prominence, Combined, Bootstrap) and allows full manual editing of selections.

The application is designed to:

- Mark and color‑code shallow vs. deep peaks/troughs.

- Let the user refine picks through zooming, panning, and direct selection/deselection.
- Perform **automatic error checks** for alternation, amplitude ratio, and phase shift.
- Export both a **basic peak list** and a **formatted amplitude/phase file** compatible with downstream seepage analysis.

Typical uses:

- Daily forcing (1 day^{-1}) signals, but adjustable for other target frequencies.
- Datasets with varying quality, from clean, high-amplitude signals to noisy, weak responses.

B. Input Data File Structure

Requires **one CSV file** — typically the output of tts_freqfilt.py — containing:

```
WaterDay,Shallow.Temp.Filt,Deep.Temp.Filt
312.000000000,0.007140,0.018250
312.000694302,0.001050,0.015070
...
```

Columns:

- **WaterDay** – Decimal day of water year.
- **Shallow.Temp.Filt** – Filtered shallow probe temperature (°C).
- **Deep.Temp.Filt** – Filtered deep probe temperature (°C).

The file **must** already be filtered and, if needed, resampled in previous steps.

C. Parameter File (tts_peakpick.par)

The .par file stores default settings. These can be loaded, modified in-app, and re-saved. As an example of file contents:

```
[PARAMETERS]
data_file = PR24_TP04-SD1-filtr.csv
sensor_spacing = 0.18
target_period = 1.0
search_tolerance = 0.1
slope_threshold = 0.001
```

```
min_distance = 20
prominence_factor = 0.15
output_folder = peak_analysis_output
peak_method = combined
shallow_color = blue
deep_color = red
peak_size = 12
trough_size = 10
line_width = 2
```

Key settings:

- **sensor_spacing** – Distance between sensors (m).
- **target_period** – Expected time between peaks (days).
- **search_tolerance** – ±days allowed for bootstrap detection.
- **slope_threshold** – Derivative limit for peak/trough detection.
- **prominence_factor** – Sensitivity (0–1) for prominence methods.
- **min_distance** – Minimum samples between peaks.
- **peak_method** – One of: combined, wavelet, scipy, derivative, prominence, bootstrap, manual.
- **output_folder** – Where exported results will be saved.

D. Steps for Typical Workflow

- **Launch the Application**
  - python tts_peakpick.py
  - Opens in browser at http://127.0.0.1:8052.
- **Load Filtered Data**
  - Auto-load from .par file or
  - Upload a .csv with required columns.
- **Choose Peak Detection Method**
  - Default is Manual.
  - Switch to SciPy, Wavelet, Derivative, Prominence, Combined or Bootstrap depending on data quality.

    - Adjust method parameters in-app.
- **Inspect and Refine Picks**
    - Shallow peaks: default blue; Deep peaks: default red.
    - Troughs marked separately.
    - Zoom, pan, or click points to toggle selection.
    - Add/remove peaks manually if needed.
- **Run Error Checks**
    - Alternation: Shallow → Deep sequence.
    - Amplitude Ratio: Deep amplitude < Shallow amplitude.
    - Phase Shift: Positive and within limits.
    - Errors flagged visually and in log.
- **Review & Correct**
    - Use flagged markers to adjust picks.
    - Rerun error checks until clean.
    - *Errors are up to the user to decide whether or not the errors are meaningful or not.*
- **Export Results**
    - Basic CSV (e.g., PR24_TP04-SD1-pick.csv)
    - Formatted .dAf (e.g., PR24_TP04-SD1-pickr.dAf): includes sensor spacing, amplitude ratios, phase shifts, uncertainties. **NB: The dAf file should be used with the seepage analysis program, tts_seepage.**
    - All parameter values and filter characteristics logged for reproducibility which are saved into an automatically created log folder in the same directory as tts_peakpick.

V. tts_seepage: Calculating apparent seepage values

A. Overview

tts_seepage.py computes **apparent vertical fluid flux** from the **amplitude ratio** and **phase shift** data produced by tts_peakpick.py, following the one-dimensional heat transport approach described in Hatch et al. (2006).

The tool:

- Reads **picked peak data** in MATLAB-style .dAf format.
- Uses thermal properties and geometric spacing to solve for seepage velocity and direction via **amplitude** (preferred) and **phase** equations.
- Performs **iterative root-finding** until convergence.
- Flags physically unrealistic or numerically unstable values.
- Outputs seepage rate time series and supporting detail files.
- Provides an interactive Dash interface to review inputs, adjust parameters, and plot results.

Flow direction is determined from the **amplitude-based** solution; the phase-based method provides magnitude only. For more information about the theory, see Hatch et al. (2006).

B. Input Data File Structure

Input is the .dAf file generated by tts_peakpick.py ("Pick Reformatted" output), using this example format:

```
SEEPAGE RATES DATA FILE: PEAKPICKER OUTPUT
-----------------------------------------------------------------
0.180 is the relative distance (in m) between sensors.
-----------------------------------------------------------------
Data_Year    Water_Day    Ad_As    A_Uncertainty    Phase_Shift(days)
f_Uncertainty
2025    312.77777    0.57347131    1.00000000e-05    0.11111100
0.00100000
...
```

Fields:

- **Relative distance** – Sensor spacing in meters.
- **Data_Year** – Water year of record.
- **Water_Day** – Decimal day in water year.
- **Ad_As** – Amplitude ratio (Deep/Shallow).
- **A_Uncertainty** – Amplitude ratio uncertainty.
- **Phase_Shift(days)** – Phase lag between shallow and deep peaks.
- **f_Uncertainty** – Phase shift uncertainty.

C. Parameter File (tts_seepage.par)

Contains physical and numerical parameters for seepage calculations.

Example:

```
1.4             ---thermal conductivity (W/m*°C)
0.001, 0.001    ---transverse and longitudinal dispersivity (m)
996.5, 4179     ---density (kg/m^3) and heat capacity (J/kg°C) of fluid
2650, 800       ---density (kg/m^3) and heat capacity (J/kg°C) of
grains
0.40            ---porosity
86400           ---period of analysis (s)  [=1 day for 1 day⁻¹ signals]
1.0E-13, 1.0E-13  ---tolerance for Amp iterations, uncertainty iter
1.0E-12, 1.0E-12  ---tolerance for Phase iterations, uncertainty iter
0.0, 1.0          ---min, max permitted value for A function
0.001             ---limit on (dAr/dv) slope for numerical Amp limits
0.0, 2.0          ---min, max permitted value for f function (days)
60                ---limit on (df/dv) slope for numerical Phase
limits
```

Key parameters:

- **Thermal conductivity** (lambda)
- **Transverse/Longitudinal dispersivity** (alpha_T, alpha_L)
- **Fluid density & heat capacity** (rho_f, c_f)
- **Solid density & heat capacity** (rho_s, c_s)
- **Porosity** (n)
- **Analysis period** (seconds; e.g., 86400 for daily signals)
- **Iteration tolerances** (amplitude & phase solvers)
- **Value limits** for amplitude and phase functions
- **Slope limits** for numerical stability in solvers.

D. Steps for Typical Workflow

- Launch the Application
    - python tts_seepage.py
    - Opens browser at http://127.0.0.1:8053.

- **Load Input Data**
  - Upload .dAf file from tts_peakpick.py.
  - Upload tts_seepage.par or edit parameters in-app.
- **Review Parameters**
  - Confirm physical properties and sensor spacing.
  - Adjust tolerances or limits if flagged results are excessive.
- **Run Seepage Calculations**
  - Solvers iterate until convergence for both **amplitude-based** and **phase-based** velocities.
  - Amplitude solution is used for **direction**; both methods provide **magnitude**.
- **Inspect Results**
  - Interactive plot shows time-series seepage rates (m/day).
  - Flags identify invalid or out-of-range calculations.
- **Export Outputs**
  - .sep – Main seepage rate results (both methods).
  - .spA – Detailed amplitude-based solver output.
  - .spf – Detailed phase-based solver output.
  - .spo – Summary of parameters used.
  - All parameter values and filter characteristics logged for reproducibility which are saved into an automatically created log folder in the same directory as tts_seepage.

VI. Appendix For Additional Information

A. Original Library Versions

This is the full library list where the applications were created. If for any reason errors occur due to libraries then try (re)installing libraries to mimic the original environment, or perhaps establish a new environment using desired versions.

absl-py==2.1.0
altair==5.5.0
anyio==4.4.0
argon2-cffi==23.1.0
argon2-cffi-bindings==21.2.0
arrow==1.3.0
asttokens==2.4.1
astunparse==1.6.3
async-lru==2.0.4
attrs==23.2.0
Babel==2.15.0
beautifulsoup4==4.12.3
bleach==6.1.0
blinker==1.9.0
cachetools==6.1.0
certifi==2024.6.2
cffi==1.16.0
cftime==1.6.4.post1
charset-normalizer==3.3.2
click==8.2.1
colorama==0.4.6
comm==0.2.2
contourpy==1.2.1
cycler==0.12.1

dash==3.1.1
dash-bootstrap-components==2.0.3
debugpy==1.8.2
decorator==5.1.1
defusedxml==0.7.1
executing==2.0.1
fastjsonschema==2.20.0
filelock==3.15.4
Flask==3.1.1
flatbuffers==24.3.25
fonttools==4.53.1
fortls==3.2.2
fqdn==1.5.1
fsspec==2024.6.0
gast==0.5.5
gitdb==4.0.12
GitPython==3.1.44
google-pasta==0.2.0
grpcio==1.64.1
h11==0.14.0
h5py==3.11.0
httpcore==1.0.5
httpx==0.27.0
idna==3.7
importlib_metadata==8.7.0
intel-openmp==2021.4.0
ipykernel==6.29.4
ipython==8.25.0
ipywidgets==8.1.3
isoduration==20.11.0
itsdangerous==2.2.0

jedi==0.19.1
Jinja2==3.1.4
joblib==1.4.2
json5==0.9.25
jsonpointer==3.0.0
jsonschema==4.22.0
jsonschema-specifications==2023.12.1
jupyter==1.0.0
jupyter-console==6.6.3
jupyter-events==0.10.0
jupyter-lsp==2.2.5
jupyter_client==8.6.2
jupyter_core==5.7.2
jupyter_server==2.14.1
jupyter_server_terminals==0.5.3
jupyterlab==4.2.2
jupyterlab_pygments==0.3.0
jupyterlab_server==2.27.2
jupyterlab_widgets==3.0.11
keras==3.4.0
kiwisolver==1.4.5
lazy_loader==0.4
libclang==18.1.1
llvmlite==0.44.0
Markdown==3.6
markdown-it-py==3.0.0
MarkupSafe==2.1.5
matplotlib==3.9.1
matplotlib-inline==0.1.7
mdurl==0.1.2
mistune==3.0.2

mkl==2021.4.0
ml-dtypes==0.3.2
mne==1.9.0
mpmath==1.3.0
namex==0.0.8
narwhals==1.44.0
nbclient==0.10.0
nbconvert==7.16.4
nbformat==5.10.4
nest-asyncio==1.6.0
netCDF4==1.7.2
networkx==3.3
notebook==7.2.1
notebook_shim==0.2.4
numba==0.61.2
numpy==1.26.4
opt-einsum==3.3.0
optree==0.14.1
overrides==7.7.0
packaging==24.1
pandas==2.2.2
pandocfilters==1.5.1
parso==0.8.4
pillow==10.3.0
platformdirs==4.2.2
plotly==6.2.0
pooch==1.8.2
prometheus_client==0.20.0
prompt_toolkit==3.0.47
protobuf==4.25.3
psutil==6.0.0

pure-eval==0.2.2
pyarrow==20.0.0
pycparser==2.22
pydeck==0.9.1
Pygments==2.18.0
pyparsing==3.1.2
python-dateutil==2.9.0.post0
python-json-logger==2.0.7
pytz==2024.1
PyWavelets==1.8.0
pywin32==306
pywinpty==2.0.13
PyYAML==6.0.1
pyzmq==26.0.3
qtconsole==5.5.2
QtPy==2.4.1
referencing==0.35.1
requests==2.32.3
retrying==1.4.0
rfc3339-validator==0.1.4
rfc3986-validator==0.1.1
rich==13.7.1
rpds-py==0.18.1
scikit-learn==1.5.0
scipy==1.14.0
Send2Trash==1.8.3
setuptools==70.1.1
six==1.16.0
smmap==5.0.2
sniffio==1.3.1
soupsieve==2.5

stack-data==0.6.3

streamlit==1.46.1

sympy==1.13.1

tbb==2021.13.0

tenacity==9.1.2

tensorboard==2.16.2

tensorboard-data-server==0.7.2

tensorflow==2.16.1

tensorflow-intel==2.16.1

termcolor==2.4.0

terminado==0.18.1

threadpoolctl==3.5.0

tinycss2==1.3.0

toml==0.10.2

torch==2.6.0+cu118

torchaudio==2.6.0+cu118

torchvision==0.21.0+cu118

tornado==6.4.1

tqdm==4.67.1

traitlets==5.14.3

types-python-dateutil==2.9.0.20240316

typing_extensions==4.12.2

tzdata==2024.1

uncertainties==3.2.3

uri-template==1.3.0

urllib3==2.2.2

watchdog==6.0.0

wcwidth==0.2.13

webcolors==24.6.0

webencodings==0.5.1

websocket-client==1.8.0

Werkzeug==3.1.3

wheel==0.43.0

widgetsnbextension==4.0.11

wrapt==1.16.0

ydf==0.5.0

zipp==3.23.0

B. tts_mergpars troubleshooting

File and Data Issues

CSV File Format Problems

- **Error**: "No valid data found in filename.csv"
- **Cause**: Incorrect CSV structure, missing required columns, or corrupted file
- **Solutions**:
    - Verify CSV has at least 3 columns: ID, DateTime, Temperature
    - Check for proper comma separation (not semicolons or tabs)
    - Ensure file is not corrupted by opening in text editor
    - Remove any extra header rows beyond the first

Example correct format:

```
ID,DateTime,Temperature
1,10/01/23 12:00:00 AM,15.25
2,10/01/23 12:15:00 AM,15.30
```

Date/Time Parsing Failures

- **Error**: "Could not parse date 'XX/XX/XX XX:XX:XX' on line N"
- **Cause**: Unsupported date format or corrupted timestamp data
- **Solutions**:
    - Convert dates to supported formats (MM/DD/YY or MM/DD/YYYY)
    - Check for missing AM/PM indicators in 12-hour format
    - Verify consistent date formatting throughout file
    - Remove any non-standard characters in date strings
    - Supported formats:

        - MM/DD/YY HH:MM:SS AM/PM
        - MM/DD/YY HH:MM:SS (24-hour)
        - MM/DD/YYYY HH:MM:SS AM/PM
        - YYYY-MM-DD HH:MM:SS (ISO format)

Temperature Data Issues

- **Error**: "Could not parse temperature 'XX.XX' on line N"
- **Cause**: Non-numeric characters in temperature column
- **Solutions**:
    - Remove any text annotations from temperature values
    - Replace missing values with numeric placeholders
    - Check for degree symbols (°) or units (C, F) in data
    - Verify decimal point format (use periods, not commas)

File Access and Permissions

- **Error**: "Shallow file 'filename.csv' not found"
- **Cause**: Incorrect file path, missing file, or permission issues
- **Solutions**:
    - Verify file exists in working directory
    - Check file name spelling and case sensitivity
    - Ensure read permissions on file and directory
    - Use absolute paths if relative paths fail
    - For web version: upload files directly rather than using file paths

Parameter Configuration Issues

Parameter File Problems

- **Error**: "Parameter file not found" or "Missing required parameter"
- **Cause**: Incorrect .par file format or missing required fields
- **Solutions**:
    - Ensure tts_mergpars.par is in same directory as script
    - Verify required parameters are present:

```
filename_shallow = shallow_data.csv
filename_deep = deep_data.csv
water_year = 2024
data_interval_min = 20.0
```

- Check for typos in parameter names
- Remove special characters from file paths
- Use forward slashes (/) in paths, even on Windows

Water Year Configuration Errors

- **Error**: Data contains records outside specified water year
- **Cause**: Data spans multiple water years or incorrect water year specified
- **Solutions**:
  - Verify water year matches your data (Oct 1 to Sep 30)
  - Split multi-year datasets into separate files
  - Check for data logger clock drift or incorrect time settings
  - Use water year containing majority of your data

Interval and Gap Detection Issues

- **Error**: Excessive gaps detected or unexpected gap warnings
- **Cause**: Irregular sampling intervals or data logger malfunctions
- **Solutions**:
  - Adjust gap_threshold_factor parameter (default 1.5)
  - Verify data_interval_min matches actual sampling rate
  - Check for logger battery failures or memory issues
  - Consider data logger synchronization problems

Web Interface Specific Issues (Dash Version)

Browser Compatibility Problems

- **Error**: Plot not displaying or controls not responding
- **Cause**: Outdated browser or JavaScript disabled
- **Solutions**:

- Use modern browsers: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+
- Enable JavaScript in browser settings
- Clear browser cache and cookies
- Try incognito/private browsing mode
- Disable browser extensions that might interfere

File Upload Failures

- **Error**: Upload hangs or files not processing
- **Cause**: Large file size, network issues, or browser limitations
- **Solutions**:
  - Limit files to <100MB for optimal performance
  - Check network connection stability
  - Try uploading one file at a time
  - Use "Load from .par" option for local files
  - Refresh page and retry upload

Application Startup Issues

- **Error**: "Address already in use" or port conflicts
- **Cause**: Another instance running or port 8050 occupied
- **Solutions**:
  - Close other instances of tts_mergpars.py
  - Kill processes using port 8050: `netstat -ano | findstr :8050`
  - Restart application after closing all instances
  - Check for other Dash applications running

Data Quality and Processing Issues

Large Data Gaps

- **Error**: Significant time periods with no data
- **Cause**: Logger malfunction, battery failure, or data corruption
- **Solutions**:
  - Document gap periods for analysis consideration

    - Check if gaps align with known logger maintenance
    - Consider interpolation for small gaps (<1 hour)
    - Split dataset around large gaps for separate analysis

Temperature Unit Confusion

- **Error**: Temperature values outside reasonable range
- **Cause**: Mixing Celsius and Fahrenheit data or incorrect conversion
- **Solutions**:
    - Verify temperature units in original logger files
    - Use convert_f_to_c = 1 parameter if data is in Fahrenheit
    - Check for reasonable temperature ranges (typically 0-30°C for water)
    - Manually verify conversion: °C = (°F - 32) × 5/9

Clock Synchronization Issues

- **Error**: Shallow and deep data timestamps don't align properly
- **Cause**: Logger clocks set to different times or time zones
- **Solutions**:
    - Check logger clock synchronization at deployment
    - Document any known time offset between loggers
    - Consider manual time correction before processing
    - Verify time zone consistency across all loggers

Memory and Performance Issues

Application Crashes with Large Datasets

- **Error**: Python crashes or "MemoryError"
- **Cause**: Insufficient RAM for large datasets
- **Solutions**:
    - Process smaller time windows separately
    - Increase system virtual memory
    - Close other memory-intensive applications
    - Consider data decimation for initial processing

Slow Processing Performance

- **Error**: Very slow data loading or processing
- **Cause**: Large files, old hardware, or inefficient data structures
- **Solutions**:
    - Pre-process data to remove unnecessary columns
    - Use SSD storage for faster file access
    - Process data in chunks rather than all at once
    - Ensure adequate RAM (8GB+ recommended for large datasets)

Output and Export Issues

File Creation Errors

- **Error**: "Permission denied" or "Cannot create output file"
- **Cause**: Insufficient write permissions or disk space
- **Solutions**:
    - Check available disk space
    - Verify write permissions in output directory
    - Avoid special characters in output filenames
    - Use different output directory if needed

Incorrect Output Format

- **Error**: CSV files with wrong formatting or precision
- **Cause**: Software configuration or data type issues
- **Solutions**:
    - Verify decimal precision in output (6 decimals for WaterDay)
    - Check column headers match expected format
    - Ensure proper CSV delimiter (commas, not semicolons)
    - Test with smaller dataset first

System and Environment Issues

Python Library Conflicts

- **Error**: ImportError or version compatibility issues

- **Cause**: Incompatible library versions or missing dependencies
- **Solutions**:
    - Use recommended library versions from installation guide
    - Create isolated Python environment with virtualenv or create an virtual enviroment
    - Reinstall all required packages: pip install --force-reinstall
    - Check for Python version compatibility (3.12 preferred)

Operating System Compatibility

- **Error**: Script fails on specific OS or architecture
- **Cause**: OS-specific file path or library issues
- **Solutions**:
    - Use forward slashes in file paths universally
    - Test with administrator/root privileges if needed
    - Check for OS-specific library variants
    - Consider using Windows Subsystem for Linux (WSL) on Windows

Network and Firewall Issues (Web Version)

- **Error**: Cannot access http://127.0.0.1:8050
- **Cause**: Firewall blocking local server or port conflicts
- **Solutions**:
    - Add exception for Python in firewall settings
    - Temporarily disable firewall for testing
    - Check if antivirus software is blocking connections
    - Try different port by modifying app.run() parameters

Advanced Troubleshooting

Data Validation Failures

- **Error**: Water year validation warnings or data outside expected ranges
- **Diagnosis Steps**:
    - Check original logger programming dates

    - Verify deployment and retrieval dates
    - Examine data for systematic errors or drift
    - Compare with field notes and maintenance records

Merge Algorithm Issues

- **Error**: Poor alignment between shallow and deep temperature records
- **Diagnosis Steps**:
    - Verify both loggers used same sampling interval
    - Check for missing data periods in either dataset
    - Examine timestamp precision and rounding
    - Consider logger internal clock accuracy

Performance Optimization

- **For Large Datasets (>50,000 records)**:
    - Monitor system memory usage during processing
    - Consider data preprocessing to remove outliers
    - Use chunked processing for extremely large files
    - Implement data compression for storage efficiency

Mathematical Background

Water Year Calculations

Leap Year Determination The application uses the Gregorian calendar leap year rules:

is_leap_year(year) = (year % 4 == 0 AND year % 100 != 0) OR (year % 400 == 0)

Examples:

- 2024: LEAP (divisible by 4, not by 100)
- 2100: NOT LEAP (divisible by 100, not by 400)
- 2400: LEAP (divisible by 400)

Water Year Length Calculation

water_year_length = 366 if is_leap_year(water_year) else 365

For Water Year N, the leap day occurs in calendar year N (February 29).

Water Day Conversion Given a datetime dt and water year wy:

```
water_year_start = datetime(wy - 1, 10, 1)  # October 1 of previous calendar year
water_day = (dt - water_year_start).total_seconds() / 86400.0
```

This calculation provides decimal precision where:

- Water Day 0.0 = October 1, 00:00:00
- Water Day 0.5 = October 1, 12:00:00
- Water Day 1.0 = October 2, 00:00:00

Calendar Year Assignment for WYO Files

```
if water_day < water_year_length:
    calendar_year = water_year - 1
else:
    calendar_year = water_year
```

This ensures proper temporal continuity in WYO format files.

Data Merging Algorithm

Nearest-Neighbor Timestamp Matching

The application uses pandas merge_asof with direction='nearest':

```
merged = pd.merge_asof(
    shallow_sorted,     # Primary timeline (shallow data)
    deep_sorted,        # Secondary data (deep data)
    on='WaterDay',      # Merge key
    direction='nearest' # Find closest match
)
```

Matching Tolerance For each shallow temperature record at time t_s, finds deep temperature record at time t_d where:

```
|t_s - t_d| = min(|t_s - t_i|) for all i in deep_data
```

This approach handles minor clock drift between loggers while maintaining data integrity.

Gap Detection Algorithm

Gap Threshold Calculation

gap_threshold = data_interval_min × gap_threshold_factor

Default values:

- data_interval_min = 20.0 minutes (typical logger interval)
- gap_threshold_factor = 1.5
- Resulting threshold = 30.0 minutes

Gap Identification

time_diffs = df['DateTime'].diff().dt.total_seconds() / 60.0

gaps = time_diffs > gap_threshold

Gaps are identified where consecutive measurements are separated by more than the threshold time.

Date Format Parsing Priority

The application attempts date parsing in order of likelihood:

1. %m/%d/%y %I:%M:%S %p (12-hour with AM/PM) - Most common logger format
2. %m/%d/%y %H:%M:%S (24-hour) - Alternative logger format
3. %m/%d/%Y %I:%M:%S %p (4-digit year with AM/PM)
4. %m/%d/%Y %H:%M:%S (4-digit year 24-hour)
5. %Y-%m-%d %H:%M:%S (ISO format) - International standard
6. %m/%d/%y %H:%M (No seconds) - Simplified format
7. %m/%d/%Y %H:%M (4-digit year no seconds)

Parse Success Rate Optimization. The order prioritizes formats most commonly produced by temperature loggers, reducing parse time for typical datasets.

Temperature Conversion Formula

Fahrenheit to Celsius Conversion

temperature_celsius = (temperature_fahrenheit - 32.0) / 1.8

Applied when convert_f_to_c = 1 in parameter file.

Precision Maintenance All temperature calculations maintain floating-point precision throughout processing, with formatting applied only during file output:

- CSV output: 5 decimal places for temperatures
- WYO output: 5 decimal places for temperatures
- Water Day output: 6 decimal places for temporal precision

Memory Management and Optimization

Data Structure Efficiency

- Uses pandas DataFrame for vectorized operations
- Sorts data by WaterDay for efficient merge operations
- Removes duplicate columns after merging to reduce memory footprint

Processing Complexity

- Time complexity: O(n log n) for sorting + O(n) for merging
- Space complexity: O(n) where n = number of temperature records
- Optimal for datasets up to ~100,000 records on typical hardware

C. tts_freqfilt troubleshooting

File and Data Issues

Input Data Format Problems

- **Error**: "No data loaded" or "CSV must contain columns: WaterDay, TempShallow, TempDeep"
- **Cause**: Incorrect CSV structure or missing required columns
- **Solutions**:
    - Verify CSV has columns: WaterDay, TempShallow, TempDeep
    - Alternative column names (Shallow.Temp, Deep.Temp) are automatically recognized
    - Ensure WaterDay is in decimal format (e.g., 312.000000)
    - Check that temperature values are numeric (no text annotations)
    - Remove any extra header rows beyond column names

Example correct format:

```
WaterDay,TempShallow,TempDeep
312.000000,20.246000,21.199000
312.013889,20.150000,21.151000
...
```

Data Range and Clipping Issues

- **Error**: "Data clipped from X to Y records" warnings
- **Cause**: WaterDay limits applied or data outside specified range
- **Solutions**:
    - Check wd_lower_limit and wd_upper_limit parameters
    - Use -1 for both limits to include all data
    - Verify WaterDay values are in expected range for your dataset
    - Check that data interval matches actual sampling rate

File Access and Loading Failures

- **Error**: "Default file not found" or file browser crashes
- **Cause**: Missing input file or permission issues
- **Solutions**:
    - Verify filename_composite_data parameter in .par file
    - Check file exists in current directory, script directory, or specified path
    - Use "Browse Different File" button instead of "Load Default File"
    - On macOS/Jupyter: Use "Drag and Drop" option instead of blue "Browse Different File" button to avoid crashes

Parameter Configuration Issues

Parameter File Problems

- **Error**: "No tts_freqfilt.par found" or parameter loading failures
- **Cause**: Missing or incorrectly formatted parameter file
- **Solutions**:
    - Create tts_freqfilt.par in same directory as script

  Verify required parameters are present:

```
filename_composite_data = temperature_composite_data.csv
data_interval_minutes = 20
time_bandwidth_parameter = 4
```

```
start_band_pass = 0.8
end_band_pass = 1.2
filter_order = 3
resample_interval = 1
```

    - Check for typos in parameter names
    - Use forward slashes (/) in paths, even on Windows

Sampling Frequency Configuration Errors

- **Error**: Incorrect frequency axis or PSD scaling issues
- **Cause**: Wrong data_interval_minutes parameter
- **Solutions**:
    - Verify data_interval_minutes matches actual logger sampling rate
    - Common values: 15, 20, or 30 minutes
    - Check that sampling_freq = (24 × 60) / interval_minutes
    - For 20-minute intervals: sampling_freq = 72 measurements/day

Resampling Configuration Issues

- **Error**: No resolution improvement or excessive memory usage
- **Cause**: Incorrect resample_interval parameter
- **Solutions**:
    - Set resample_interval = 1 for 20:1 resolution improvement
    - For 20-minute data: improvement = 20/1 = 20:1 enhancement
    - Larger datasets may require resample_interval = 5 or 10
    - Monitor memory usage with high-resolution resampling

PSD Analysis and Method Issues

MATLAB MTM Method Problems

- **Error**: PSD computation fails or produces unexpected results
- **Cause**: Incorrect time-bandwidth parameter or NFFT settings
- **Solutions**:

    - Use time_bandwidth = 4 to match R/MATLAB (default)
    - Set matlab_nfft = 2048 for good frequency resolution
    - Ensure data length > NFFT value
    - Try reducing NFFT if getting memory errors
    - Check that number of tapers = 2 × time_bandwidth - 1

Multitaper (R-style) Method Issues

- **Error**: "Multitaper method not available" or import errors
- **Cause**: MNE library not installed or incompatible version
- **Solutions**:
    - Install MNE library: pip install mne==1.9.0
    - Use alternative methods (MATLAB MTM, Welch) if MNE installation fails
    - Check Python version compatibility (3.9+ required), but other libraries might fail, at the time of writing, no need to install if MNE is not used
    - Verify MNE imports correctly: import mne.time_frequency

Welch Method Parameter Issues

- **Error**: Poor PSD quality or excessive noise
- **Cause**: Inappropriate segment length parameters
- **Solutions**:
    - Adjust nperseg_divider (default: 4) - lower values = longer segments
    - Modify nperseg_max (default: 520) - limits maximum segment length
    - Try different window types: tukey, hann, hamming, blackman
    - For noisy data: increase overlap, use larger segments
    - For smooth data: decrease overlap, use smaller segments

PSD Plot Display Issues

- **Error**: PSD plot not updating or showing errors
- **Cause**: Parameter conflicts or computational errors
- **Solutions**:
    - Check that PSD method is supported and available

    - Verify all method-specific parameters are valid
    - Try switching to different PSD method temporarily
    - Clear browser cache and refresh page
    - Check console for JavaScript errors

Frequency Selection and Filter Design Issues

Frequency Selection Problems

- **Error**: Cannot select frequency bands or clicks not registering
- **Cause**: Not in frequency selection mode or plot interaction issues
- **Solutions**:
    - Click "Enter Frequency Selection Mode" button first
    - Click exactly twice on PSD plot to set bounds
    - Ensure clicks are on actual plot area, not margins
    - Check that frequency values are reasonable (0.1-10 day^{-1})
    - Exit and re-enter selection mode if having issues

Filter Design and Validation Errors

- **Error**: "Filter validation failed" or parameter warnings
- **Cause**: Invalid frequency ranges or filter order issues
- **Solutions**:
    - Ensure low frequency < high frequency
    - Check that frequencies < Nyquist frequency (sampling_freq/2)
    - Verify filter order is appropriate for data length
    - Use filter order 2-3 for short datasets, 3-6 for longer datasets
    - Adjust ramp_fraction (0.0-0.5) for smoother transitions

Filter Order Selection Issues

- **Error**: Artificial oscillations, ringing, or insufficient filtering
- **Cause**: Inappropriate filter order for data characteristics
- **Solutions**:
    - **For ringing/artifacts**: Reduce filter order (6→3→2)

- **For insufficient filtering**: Increase filter order (3→4→6)
- **For short datasets** (<7 days): Use order 2
- **For standard datasets** (7+ days): Use order 3 (recommended)
- **For long datasets** (30+ days): Order 6 acceptable
- **Never exceed order 8** unless >60 days of data

Web Interface Specific Issues

Browser Compatibility Problems

- **Error**: Interface not loading or controls unresponsive
- **Cause**: Outdated browser or JavaScript issues
- **Solutions**:
  - Use modern browsers: Chrome 90+, Firefox 88+, Safari 14+, Edge 90+, etc
  - Enable JavaScript in browser settings
  - Clear browser cache and cookies
  - Try incognito/private browsing mode
  - Disable browser extensions that might interfere

Plot Rendering Issues

- **Error**: Plots not displaying or interactive features not working
- **Cause**: Plotly compatibility or rendering problems
- **Solutions**:
  - Refresh page and reload data
  - Check browser console for errors
  - Try different browser
  - Ensure adequate RAM for large datasets
  - Close other browser tabs to free memory

Application Startup Issues

- **Error**: "Address already in use" or port 8051 conflicts
- **Cause**: Another instance running or port occupied
- **Solutions**:

- Close other instances of tts_freqfilt.py
- Kill processes using port 8051: netstat -ano | findstr :8051
- Wait 30 seconds after closing before restarting
- Change port in code if persistent conflicts

Data Quality and Processing Issues

Filter Application Failures

- **Error**: "Filtering failed" or numerical errors
- **Cause**: Filter design problems or data issues
- **Solutions**:
  - Check for NaN values in temperature data
  - Verify data has sufficient length for filter order
  - Try simpler filter (Butterworth instead of Elliptic)
  - Reduce filter order if getting numerical instabilities
  - Apply trend removal before filtering

Memory and Performance Issues

- **Error**: Application crashes or very slow processing
- **Cause**: Large datasets or insufficient system resources
- **Solutions**:
  - Use data clipping to analyze smaller time windows
  - Reduce resampling resolution (1→5→10 minutes)
  - Close other memory-intensive applications
  - Process datasets <50,000 points for optimal performance
  - Consider 8GB+ RAM for large datasets

Edge Effects and Data Quality

- **Error**: Artificial features at data boundaries
- **Cause**: Insufficient data buffer for filter order
- **Solutions**:
  - Ensure data extends well beyond analysis period

- Use lower filter order for shorter datasets
- Apply appropriate trend removal before filtering
- Check that analysis period is at least 3× filter duration from edges

Output and Export Issues

File Saving Problems

- **Error**: "Failed to create output folder" or save errors
- **Cause**: Insufficient permissions or path issues
- **Solutions**:
  - Check write permissions in output directory
  - Avoid special characters in output filenames
  - Use "Change" button to select different output folder
  - Ensure adequate disk space available

Output Format Issues

- **Error**: Incorrect CSV format or precision problems
- **Cause**: Formatting configuration problems
- **Solutions**:
  - Verify output format: WaterDay,Shallow.Temp.Filt,Deep.Temp.Filt
  - Check decimal precision (9 places for WaterDay, 6 for temperatures)
  - Ensure proper CSV delimiter (commas, not semicolons)
  - Test with smaller dataset first

Mathematical Background and Theory

Power Spectral Density Methods

MATLAB-Style Multitaper Method: The default method uses Thomson's multitaper approach with Discrete Prolate Spheroidal Sequences (DPSS): $|H(\omega)|^2 = (1/K) \Sigma_i |Y_i(\omega)|^2$

where:

- $K = 2 \times NW - 1$ (number of tapers)
- NW = time_bandwidth parameter (default: 4)
- $Y_i(\omega)$ = DFT of data windowed with i-th DPSS taper

Mathematical advantages:

- Superior bias-variance tradeoff
- Optimal frequency resolution for given data length
- Reduced spectral leakage compared to single-window methods

Welch Method: Classical periodogram averaging approach:

$P(\omega) = (1/M) \Sigma\square |X\square(\omega)|^2$

Where:

- M = number of overlapping segments
- $X\square(\omega)$ = DFT of j-th windowed segment
- Segment length = min(data_length/nperseg_divider, nperseg_max)

Filter Design Mathematics

Butterworth Filter Response: The fundamental equation for filter order N:

$|H(\omega)|^2 = 1 / (1 + (\omega/\omega c)^{(2N)})$

Key mathematical relationships:

1. **Rolloff Rate**: -20N dB/decade = -6N dB/octave
    - Order 2: -40 dB/decade
    - Order 3: -60 dB/decade (recommended default)
    - Order 6: -120 dB/decade
2. **Stopband Rejection at 2× cutoff frequency**:
    - Order 1: 84% rejection
    - Order 3: 99% rejection (optimal balance)
    - Order 6: 99.98% rejection
3. **Impulse Response Duration**: $N/(2\pi \times fc)$
    - Order 3 at 1.0 day^{-1}: ~0.48 days = 11.5 hours

Ramp/Taper Implementation: Following specification Table 2 format:

$f\_start\_ramp = f\_low - (ramp\_width/2)$

$f\_start\_pass = f\_low + (ramp\_width/2)$

$f\_end\_pass = f\_high - (ramp\_width/2)$

$f\_end\_ramp = f\_high + (ramp\_width/2)$

Where: ramp_width = (f_high - f_low) × ramp_fraction

Filter Order Selection Mathematics

Mathematical Validation for Order 3 Default:

For typical diurnal analysis (0.8-1.2 day^{-1} band):

1. **Frequency Selectivity**:
    - At 0.4 day^{-1}: $|H(0.4)|^2 = 1/(1 + (0.4/0.8)^6) = 98.4\%$ rejection
    - At 2.4 day^{-1}: $|H(2.4)|^2 = 1/(1 + (2.4/1.2)^6) = 98.4\%$ rejection
2. **Time Domain Response**:
    - Filter duration: 11.5 hours
    - For daily cycles (24h): 11.5/24 = 48% overlap (acceptable)
3. **Data Length Requirements**:
    - Minimum: 3 × 11.5h = 1.4 days
    - Recommended: 7+ days for stable performance

Order Optimization Guidelines:

Order	Mathematical Properties	Best Applications	Stability
2	$(\omega/\omega c)^4$ rolloff	Noisy data, <10 days	Excellent
3	$(\omega/\omega c)^6$ rolloff	Standard analysis, 7+ days	Optimal
6	$(\omega/\omega c)^{12}$ rolloff	High precision, 20+ days	Good
8+	$(\omega/\omega c)^{16}$+ rolloff	Very long series, 60+ days	Caution

Resampling Mathematics

Resolution Improvement Calculation:

improvement_factor = original_interval / resample_interval

new_length = original_length × improvement_factor

For 20-minute → 1-minute resampling:

- Improvement: 20/1 = 20:1

- Enhanced peak/trough detection accuracy
- Better amplitude and phase calculations

Time Axis Scaling:

```
new_time_axis = linspace(start_time, end_time, new_length)
resampled_data = signal.resample(original_data, new_length)
```

Trend Removal Mathematics

DC Offset Removal:

```
detrended = data - mean(data)
```

Linear Trend Removal:

```
coeffs = polyfit(time_index, data, 1)
trend = polyval(coeffs, time_index)
detrended = data - trend
```

High-pass Filter Trend Removal:

```
cutoff_normalized = 0.1 / (sampling_freq/2)  # 0.1 day⁻¹
b, a = butter(3, cutoff_normalized, btype='high')
detrended = filtfilt(b, a, data)
```

System and Environment Issues

Python Library Conflicts

- **Error**: ImportError or version compatibility issues
- **Cause**: Incompatible library versions
- **Solutions**:
    - Use recommended versions: scipy==1.14.0, numpy==1.26.4, plotly==6.2.0
    - Create isolated environment: `python -m venv freqfilt_env`
    - Reinstall packages: `pip install --force-reinstall -r requirements.txt`
    - Check Python version (3.8+ required, 3.12.2 recommended)

Operating System Compatibility

- **Error**: Platform-specific failures
- **Cause**: OS-specific library or path issues
- **Solutions**:

    - Use forward slashes in all file paths
    - Install OS-specific scipy/numpy builds
    - On Linux: may need sudo apt-get install python3-tk
    - On macOS: use Homebrew Python for best compatibility

Memory Management

- **Error**: "MemoryError" or application crashes
- **Cause**: Insufficient RAM for large datasets or high-resolution resampling
- **Solutions**:
    - Monitor memory usage during processing
    - Use data clipping for analysis of specific periods
    - Reduce resampling resolution for very large datasets
    - Close other memory-intensive applications
    - Recommend 8GB+ RAM for datasets >20,000 points with resampling

Advanced Troubleshooting

Numerical Stability Issues

- **Error**: Filter produces NaN values or extreme oscillations
- **Diagnosis Steps**:
    - Check for very small frequency bands (< 0.01 day^{-1})
    - Verify filter order is appropriate for band width
    - Examine data for discontinuities or outliers
    - Test with synthetic sinusoidal data
- **Solutions**:
    - Use wider frequency bands
    - Apply robust trend removal before filtering
    - Consider different filter types (Chebyshev vs Butterworth)

Spectral Analysis Validation

- **Error**: PSD results don't match expectations
- **Diagnosis Steps**:

- Compare different PSD methods on same data
- Check units and scaling factors
- Verify sampling frequency calculation
- Test with known synthetic signals

- **Solutions**:
  - Use MATLAB MTM for best compatibility with established methods
  - Verify time_bandwidth parameter matches R/MATLAB settings
  - Check that data is properly detrended before PSD calculation

Performance Optimization

- **For datasets >10,000 points**:
  - Use data clipping to focus on specific analysis periods
  - Consider reducing PSD resolution (lower NFFT) for faster computation
  - Monitor browser memory usage and restart if needed
  - Process multiple smaller time windows separately

Mathematical Verification Procedures

Filter Response Verification:

```
 # Test filter with synthetic sinusoid
test_freq = 1.0  # day⁻¹
test_signal = sin(2π × test_freq × time_array)
filtered_signal = apply_filter(test_signal)
amplitude_ratio = max(filtered_signal) / max(test_signal)
```

- **Expected results for 0.8-1.2 day⁻¹ bandpass**:
  - 1.0 day⁻¹ sinusoid: amplitude_ratio ≈ 1.0
  - 0.5 day⁻¹ sinusoid: amplitude_ratio < 0.1
  - 2.0 day⁻¹ sinusoid: amplitude_ratio < 0.1

D. tts_peakpick troubleshooting

File and Data Issues

Filtered Data File Problems

- **Error**: "Required columns not found in CSV file"
- **Cause**: Input file missing WaterDay, Shallow.Temp.Filt, or Deep.Temp.Filt columns
- **Solutions**:
    - Verify file was processed through tts_freqfilt.py first
    - Check column headers match exactly: WaterDay,Shallow.Temp.Filt,Deep.Temp.Filt
    - Ensure CSV uses comma delimiters, not semicolons or tabs
    - Remove any extra header rows or metadata

Unfiltered Data Issues

- **Error**: "Data appears unfiltered - high frequency noise detected"
- **Cause**: Attempting to use raw merged data instead of filtered data
- **Solutions**:
    - Process data through tts_freqfilt.py before peak picking
    - Apply appropriate bandpass filter (typically 0.8-1.2 cycles/day for diurnal signals)
    - Verify filtered data shows smooth sinusoidal patterns
    - Check filter parameters match your target frequency

Data Range and Quality Problems

- **Error**: "No valid data points in specified range"
- **Cause**: Data outside expected WaterDay range or all NaN values
- **Solutions**:
    - Check WaterDay values are reasonable (0-366 for single water year)
    - Verify temperature values are numeric and not text
    - Remove or interpolate NaN values before processing
    - Ensure sufficient data length (minimum 5-10 cycles for reliable detection)

Peak Detection Algorithm Issues and Mathematical Background

SciPy find_peaks Method

Mathematical Foundation: The SciPy method uses scipy.signal.find_peaks with adaptive parameters:

```
# Remove DC offset
data_centered = data - np.mean(data)

# Calculate prominence threshold
data_std = np.std(data_centered)
prominence = data_std * prominence_factor

# Find peaks
peaks, properties = find_peaks(data_centered,
                               distance=min_distance,
                               prominence=prominence)
```

Key Parameters:

- **Distance**: Minimum samples between peaks = (target_period × sampling_rate) × 0.5
- **Prominence**: Peak height above surrounding minima = std(data) × prominence_factor

Common Issues:

- **Error**: "No peaks detected with current prominence"
- **Cause**: Prominence threshold too high for signal amplitude
- **Solutions**:
    - Reduce prominence_factor from 0.15 to 0.05-0.1
    - Check data filtering - signal may be over-smoothed
    - Verify data contains clear diurnal cycles
    - Use "Show All Local Maxima" mode to see all potential peaks
- **Error**: "Too many peaks detected"
- **Cause**: Prominence threshold too low or noise in signal
- **Solutions**:
    - Increase prominence_factor to 0.2-0.3
    - Increase min_distance parameter
    - Apply additional smoothing in frequency domain
    - Use exclusion ranges to skip noisy periods

Wavelet-based Detection

Mathematical Foundation: Uses Continuous Wavelet Transform (CWT) with Mexican Hat wavelet:

```
# CWT with scale-dependent analysis
coefficients = pywt.cwt(data, scales, 'mexh')[0]

# Ridge detection in coefficient space
ridges = identify_ridgelines(coefficients, scales)

# Peak extraction from ridgelines
peaks = extract_peaks_from_ridges(ridges, data)
```

Wavelet Theory:

- **Mexican Hat Wavelet**: $\psi(t) = (2/\sqrt{(3\sigma)}\pi^{(1/4)}) \times (1 - t^2/\sigma^2) \times \exp(-t^2/2\sigma^2)$
- **Scale Selection**: Scales = $2\pi f_0/f$ where f_0 = center frequency, f = target frequencies
- **Ridge Extraction**: Follows local maxima across scales

Common Issues:

- **Error**: "Wavelet detection failed - no ridgelines found"
- **Cause**: Signal-to-noise ratio too low or inappropriate scale range
- **Solutions**:
    - Increase data filtering (narrower bandpass)
    - Adjust wavelet scales: `np.arange(5, min(100, len(data)//5), 2)`
    - Check for data gaps that break wavelet continuity
    - Consider using combined method instead
- **Error**: "Wavelet processing very slow"
- **Cause**: Large dataset or too many scales
- **Solutions**:
    - Reduce scale range: use 20-50 scales maximum
    - Decimate data for initial analysis
    - Use SciPy method for large datasets (>10,000 points)

Custom Derivative Method

Mathematical Foundation: Detects peaks using first and second derivative analysis:

```
# Gaussian smoothing
sigma = max(2, min(5, len(data) // 100))
smoothed = gaussian_filter1d(data, sigma=sigma)

# First derivative (slope)
dt = np.gradient(time)
dT_dt = np.gradient(smoothed) / dt

# Second derivative (curvature)
d2T_dt2 = np.gradient(dT_dt) / dt

# Peak detection logic
for i in range(2, len(data) - 2):
    # Peak: dT/dt changes from + to - AND d²T/dt² < 0
    if (dT_dt[i-1] > 0 and dT_dt[i+1] < 0 and d2T_dt2[i] < 0):
        peaks.append(i)
    # Trough: dT/dt changes from - to + AND d²T/dt² > 0
    elif (dT_dt[i-1] < 0 and dT_dt[i+1] > 0 and d2T_dt2[i] > 0):
        troughs.append(i)
```

Derivative Theory:

- **First Derivative**: Rate of temperature change (°C/day)
- **Second Derivative**: Acceleration of temperature change
- **Peak Criteria**: Zero-crossing in first derivative with negative second derivative
- **Gaussian Smoothing**: Reduces noise while preserving peak structure

Common Issues:

- **Error**: "Derivative method finds no peaks"
- **Cause**: Over-smoothing or slope_threshold too high
- **Solutions**:
    - Reduce slope_threshold from 0.001 to 0.0001

    - Decrease Gaussian sigma (less smoothing)
    - Check for adequate sampling resolution
    - Verify data has clear slope changes at peaks
- **Error**: "Too many spurious peaks from derivative method"
- **Cause**: Insufficient smoothing or noisy data
- **Solutions**:
    - Increase Gaussian sigma parameter
    - Raise slope_threshold to filter weak transitions
    - Apply additional filtering in frequency domain
    - Use minimum distance filtering between peaks

Prominence-based Detection

Mathematical Foundation: Simple threshold-based detection using signal range:

```
# Calculate prominence threshold
data_range = np.max(data) - np.min(data)
min_prominence = data_range * prominence_factor

# Find peaks above threshold
peaks, _ = find_peaks(data, prominence=min_prominence)
```

Prominence Theory:

- **Prominence**: Minimum height a peak must rise above surrounding valleys
- **Relative Threshold**: Adapts to signal amplitude automatically
- **Range-based**: Uses full data range for threshold calculation

Common Issues:

- **Error**: "Prominence method misses obvious peaks"
- **Cause**: prominence_factor too high relative to signal variability
- **Solutions**:
    - Reduce prominence_factor to 0.05-0.1
    - Check data preprocessing - may need better filtering
    - Verify signal has sufficient amplitude variation

- Consider using combined method for better sensitivity

Combined Methods (Most Sensitive)

Mathematical Foundation: Merges results from multiple detection approaches:

```
# Method 1: Local maxima detection
local_maxima = find_local_maxima(data)

# Method 2: Multiple SciPy runs with varying prominence
for prom_factor in [0.05, 0.1, 0.15, 0.2, 0.3]:
    peaks_scipy = find_peaks(data, prominence=std*prom_factor)
    all_peaks.extend(peaks_scipy)

# Method 3: Unconstrained detection
peaks_unconstrained = find_peaks(data)
all_peaks.extend(peaks_unconstrained)

# Merge and deduplicate
final_peaks = deduplicate_peaks(all_peaks, min_distance)
```

Combined Algorithm Theory:

- **Sensitivity Maximization**: Uses multiple prominence thresholds
- **Redundancy**: Different methods catch different peak characteristics
- **Deduplication**: Removes nearby duplicates using distance threshold

Common Issues:

- **Error**: "Combined method finds too many peaks"
- **Cause**: High sensitivity picks up noise as peaks
- **Solutions**:
    - Increase min_distance parameter
    - Use exclusion ranges for noisy periods
    - Filter results manually using interactive editing
    - Apply post-processing with stricter prominence

Bootstrap from Manual Selection

Mathematical Foundation: Propagates user-selected peaks using periodicity estimation:

```
# Calculate average period from manual selections
manual_times = time[manual_peak_indices]
periods = np.diff(manual_times)
avg_period = np.median(periods)  # Robust to outliers

# Forward propagation
last_time = manual_times[-1]
while last_time + avg_period <= time[-1]:
    expected_time = last_time + avg_period
    search_window = [expected_time - tolerance,
                     expected_time + tolerance]

    # Find local maximum in search window
    local_peak = find_local_max_in_window(data, search_window)
    if local_peak:
        peaks.append(local_peak)
        last_time = time[local_peak]
```

Bootstrap Theory:

- **Period Estimation**: Uses robust median of manual peak intervals
- **Temporal Propagation**: Extends pattern forward and backward in time
- **Local Search**: Finds actual peak near expected location
- **Tolerance Window**: Accommodates period variability

Common Issues:

- **Error**: "Bootstrap requires at least 2 manual peaks"
- **Cause**: Insufficient manual selections to estimate period
- **Solutions**:
    - Manually select 2-5 clear peaks across dataset
    - Ensure manual peaks represent consistent periodicity
    - Verify manual peaks are accurately placed on actual maxima

    - Check for period stability across dataset
- **Error**: "Bootstrap propagation fails after few cycles"
- **Cause**: Period instability or large gaps in data
- **Solutions**:
    - Select manual peaks from stable periods only
    - Increase search tolerance parameter
    - Check for data gaps that break periodicity
    - Use combined method for irregular periods

Parameter Configuration Issues

Sensor Spacing Problems

- **Error**: "Negative phase shifts indicate incorrect sensor configuration"
- **Cause**: Sensor spacing or labeling incorrect
- **Solutions**:
    - Verify sensor_spacing matches physical deployment (typically 0.15-0.20m)
    - Check if shallow/deep labels are swapped in data
    - Confirm sensor orientation (shallow should heat first)
    - Review field deployment notes

Target Period Mismatches

- **Error**: "Detected period differs significantly from target_period"
- **Cause**: target_period doesn't match actual signal periodicity
- **Solutions**:
    - Analyze power spectral density to identify dominant frequency
    - Adjust target_period to match observed cycles (typically 1.0 day)
    - Consider semi-diurnal (0.5 day) or tidal signals if appropriate
    - Use frequency analysis from tts_freqfilt.py results

Search Tolerance Issues

- **Error**: "Bootstrap method skips expected peaks"
- **Cause**: search_tolerance too narrow for period variability

- **Solutions**:
    - Increase search_tolerance from 0.1 to 0.2-0.3 days
    - Account for period variations due to weather patterns
    - Check for systematic drift in peak timing
    - Consider adaptive tolerance based on period stability

Interactive Interface Issues

Graph Display Problems

- **Error**: "Peaks not visible or appearing transparent"
- **Cause**: Peak visibility settings or plotting issues
- **Solutions**:
    - Click "Toggle Peak Visibility" button to restore opacity
    - Refresh browser page if display corrupted
    - Check browser compatibility (Chrome 90+, Firefox 88+ recommended)
    - Disable browser extensions that might interfere

Click Detection Problems

- **Error**: "Clicking on graph doesn't add/remove peaks"
- **Cause**: Incorrect interaction mode or browser issues
- **Solutions**:
    - Verify correct mode selected (Add Shallow/Deep/Remove)
    - Switch to "View Only" mode then back to desired mode
    - Check JavaScript is enabled in browser
    - Try clicking directly on temperature trace, not empty space

Zoom State Issues

- **Error**: "Graph zoom resets after peak edits"
- **Cause**: Zoom preservation failure in interface
- **Solutions**:
    - Set zoom level before making peak edits
    - Use manual zoom controls rather than mouse wheel

- Refresh page and re-zoom if state lost
- Use "View Only" mode for zoom operations

Error Checking and Validation Issues

Alternation Errors

- **Error**: "Consecutive peaks of same depth detected"
- **Mathematical Foundation**:

```
# Alternation check algorithm
all_peaks = [(time[i], 'shallow') for i in shallow_indices] + \
        [(time[i], 'deep') for i in deep_indices]
all_peaks.sort()

# Check for consecutive same-type peaks
for i in range(1, len(all_peaks)):
    if all_peaks[i][1] == all_peaks[i-1][1]:
        errors.append(f"Alternation violation at {all_peaks[i][0]:.2f}")
```

- **Solutions**:
    - Review peak detection parameters for over-sensitivity
    - Manually remove duplicate or spurious peaks
    - Check for data quality issues in problematic periods
    - Verify physical sensor deployment is correct

Amplitude Ratio Errors

Mathematical Foundation: Enhanced amplitude calculation using peak-trough method:

```
# Peak-trough amplitude calculation
def calculate_peak_trough_amplitude(peak_indices, trough_indices,
                                    temp_data, time_array):
    amplitudes = {}
    for peak_idx in peak_indices:
        peak_time = time_array[peak_idx]
        peak_value = temp_data[peak_idx]

        # Find closest trough within time window
```

```
        min_distance = float('inf')
        closest_trough = None

        for trough_idx in trough_indices:
            trough_time = time_array[trough_idx]
            time_diff = abs(trough_time - peak_time)

            if time_diff <= max_time_diff and time_diff < min_distance:
                min_distance = time_diff
                closest_trough = trough_idx

        if closest_trough is not None:
            trough_value = temp_data[closest_trough]
            # Enhanced amplitude: A = (Peak - Trough) / 2
            amplitude = (peak_value - trough_value) / 2
        else:
            # Fallback: A = Peak value
            amplitude = abs(peak_value)

        amplitudes[peak_idx] = amplitude

    return amplitudes

# Amplitude ratio calculation
def calculate_amplitude_ratios(shallow_amps, deep_amps):
    ratios = []
    for s_idx, s_amp in shallow_amps.items():
        s_time = time_array[s_idx]

        # Find corresponding deep peak (after shallow)
        for d_idx, d_amp in deep_amps.items():
```

```
            d_time = time_array[d_idx]
            if d_time > s_time and d_time - s_time <= tolerance:
                # Amplitude ratio: Ar = Ad/As
                ar = d_amp / s_amp if s_amp > 0 else float('inf')
                ratios.append((s_time, ar))
                break

    return ratios
```

Amplitude Ratio Theory:

- **Physical Expectation**: Deep amplitude < Shallow amplitude (Ar < 1.0)
- **Peak-Trough Method**: A = (Peak - Trough) / 2 for asymmetric signals
- **Error Criteria**: Ar ≥ (1 - tolerance) or Ar ≤ tolerance
- **Error**: "Amplitude ratio Ad/As > 1.0"
- **Cause**: Physical impossibility or measurement errors
- **Solutions**:
    - Check sensor labeling (shallow/deep may be swapped)
    - Verify peak detection accuracy - may have incorrect peaks
    - Review data filtering - over-filtering can distort amplitudes
    - Consider sensor drift or calibration issues
- **Error**: "Amplitude ratio too small (Ar < 0.001)"
- **Cause**: Very weak deep signal or measurement noise
- **Solutions**:
    - Check deep sensor functionality and placement
    - Increase filtering to improve signal-to-noise ratio
    - Verify adequate thermal forcing at surface
    - Consider longer deployment period for better signal

Phase Shift Errors

Mathematical Foundation: Phase shift calculation between paired peaks:

```
def calculate_phase_shifts(shallow_peaks, deep_peaks, time_array):
    phase_shifts = []
```

```
    for s_idx in shallow_peaks:
        s_time = time_array[s_idx]

        # Find next deep peak after shallow peak
        future_deep = deep_peaks[time_array[deep_peaks] > s_time]

        if len(future_deep) > 0:
            d_idx = future_deep[0]  # First deep peak after shallow
            d_time = time_array[d_idx]

            # Phase shift = time delay
            phase_shift = d_time - s_time
            phase_shifts.append((s_time, phase_shift))

    return phase_shifts
```

Phase Shift Theory:

- **Physical Expectation**: 0.05 ≤ Phase Shift ≤ 1.0 days
- **Heat Conduction**: Deeper sensors lag surface forcing
- **Exponential Decay**: Phase lag increases with depth
- **Error**: "Negative phase shift detected"
- **Cause**: Deep peak occurring before shallow peak
- **Solutions**:
    - Check sensor labeling and physical configuration
    - Review peak detection - may have missed or added peaks
    - Verify data time synchronization between sensors
    - Check for sensor malfunction or deployment errors
- **Error**: "Phase shift too large (>1.0 days)"
- **Cause**: Incorrect peak pairing or very slow heat conduction
- **Solutions**:
    - Review peak detection accuracy and pairing

    - Check soil/sediment thermal properties
    - Verify sensor spacing measurement
    - Consider environmental factors affecting heat transfer

Export and Output Issues

File Format Problems

- **Error**: "Cannot create .dAf formatted file"
- **Cause**: Insufficient peak pairs or data formatting issues
- **Solutions**:
    - Ensure adequate shallow-deep peak pairs exist
    - Check sensor_spacing parameter is set correctly
    - Verify amplitude and phase calculations completed
    - Review error checking - fix major errors before export

Naming Convention Errors

- **Error**: "Output files overwrite existing results"
- **Cause**: Default naming scheme conflicts
- **Solutions**:
    - Customize output filenames to include date/version
    - Use different output folders for different analyses
    - Follow suggested naming: input-pick.csv and input-pickr.dAf
    - Add site/sensor identifiers to filenames

Performance and Memory Issues

Large Dataset Processing

- **Error**: "Application becomes slow with large datasets"
- **Cause**: Memory limitations or inefficient processing
- **Solutions**:
    - Process data in smaller time windows
    - Use resampled data from tts_freqfilt.py (1-minute resolution)
    - Close other memory-intensive applications
    - Consider data decimation for initial peak detection

Browser Performance Issues

- **Error**: "Interactive graph becomes unresponsive"
- **Cause**: Too many plot elements or browser limitations
- **Solutions**:
    - Use "Toggle Peak Visibility" to reduce plot complexity
    - Clear browser cache and restart application
    - Limit dataset size to <50,000 points for interactive work
    - Use modern browser with adequate RAM

Mathematical Background and Theory

Signal Processing Fundamentals

Thermal Signal Characteristics:

- **Diurnal Forcing**: Surface temperature follows solar heating cycle
- **Heat Conduction**: One-dimensional heat equation: $\partial T/\partial t = \alpha \nabla^2 T$
- **Exponential Decay**: Amplitude decreases with depth: $A(z) = A_0 e^{(-z/d)}$
- **Phase Lag**: Time delay increases with depth: $\varphi(z) = z/d$

Where:

- α = thermal diffusivity (m^2/s)
- d = damping depth = $\sqrt{(2\alpha/\omega)}$
- ω = angular frequency (rad/s)

Peak Detection Algorithms Comparison

Method	Sensitivity	Noise Tolerance	Speed	Best For
SciPy	Medium	Medium	Fast	Clean signals
Wavelet	High	High	Medium	Noisy data
Derivative	High	Low	Medium	Smooth transitions
Prominence	Low	High	Fast	High amplitude signals
Combined	Very High	Medium	Slow	Unknown signal quality

Bootstrap	Medium	Medium	Fast	Periodic signals

Error Metrics and Thresholds

Amplitude Ratio Bounds:

Physical Range: $0.1 \leq Ar \leq 0.9$ (typical)

Error Thresholds: $Ar \leq 0.001$ or $Ar \geq 0.999$

Warning Thresholds: $Ar \leq 0.05$ or $Ar \geq 0.95$

Phase Shift Bounds:

Physical Range: $0.05 \leq \Delta t \leq 0.5$ days (typical)

Error Thresholds: $\Delta t \leq 0.001$ or $\Delta t \geq 1.0$ days

Warning Thresholds: $\Delta t \leq 0.02$ or $\Delta t \geq 0.8$ days

Quality Control Metrics

Peak Detection Quality:

- **Completeness**: Fraction of expected peaks detected
- **Accuracy**: RMS error in peak timing
- **Precision**: Standard deviation of peak spacing
- **False Positive Rate**: Spurious peaks / total detected

Data Quality Indicators:

- **Signal-to-Noise Ratio**: $SNR = 20 \times \log_{10}(A\_signal/A\_noise)$
- **Periodicity Stability**: Coefficient of variation in peak spacing
- **Amplitude Consistency**: Variation in peak amplitudes over time

Advanced Troubleshooting

Complex Error Patterns

Multiple Simultaneous Errors:

- **Diagnosis**: Use error summary by type to identify patterns
- **Systematic Issues**: All errors of one type suggest parameter problems
- **Random Issues**: Scattered errors suggest data quality problems
- **Temporal Clustering**: Errors in specific time periods suggest environmental factors

Borderline Cases:

- **Near-Threshold Values**: Amplitude ratios ~0.95 or phase shifts ~0.95 days
- **Ambiguous Peaks**: Multiple local maxima of similar height
- **Period Variations**: Systematic changes in peak spacing over time

Environmental Factor Considerations

Weather Influences:

- **Cloud Cover**: Reduces diurnal signal amplitude
- **Precipitation**: Can cause temperature spikes or data gaps
- **Seasonal Changes**: Affects background temperature and signal amplitude
- **Flow Variations**: Alters heat transport and signal characteristics

Deployment Factors:

- **Sensor Burial Depth**: Affects expected amplitude ratios and phase shifts
- **Sediment Properties**: Thermal conductivity influences signal propagation
- **Water Level Changes**: Can expose sensors or change thermal environment
- **Biofouling**: Gradual degradation of sensor response

Data Recovery Strategies

Partial Data Loss:

- **Gap Interpolation**: Use bootstrap method to estimate missing peaks
- **Period Adjustment**: Account for systematic changes in peak spacing
- **Amplitude Correction**: Apply calibration factors for sensor drift

Quality Assessment Protocol:

1. **Visual Inspection**: Review temperature traces for obvious anomalies
2. **Statistical Analysis**: Check amplitude and period distributions
3. **Physical Validation**: Verify results match expected thermal behavior
4. **Comparison Studies**: Cross-check with nearby sensors or historical data

E. tts_seepage trouble shooting

File and Data Issues

Input Data Format Problems

● **Error**: "Invalid .dAf file format" or "Cannot parse amplitude/phase data"

● **Cause**: Incorrect file format from peakpicker or corrupted data

● **Solutions**:

○ Verify input file is the .dAf format from tts_peakpick.py ("Pick Reformatted" output)

○ Check file header contains: "SEEPAGE RATES DATA FILE: PEAKPICKER OUTPUT"

○ Ensure relative distance line format: "X.XXX is the relative distance (in m) between sensors"

○ Verify column headers: Data_Year, Water_Day, Ad_As, A_Uncertainty, Phase_Shift(days), f_Uncertainty

○ Check for proper tab or space separation in data lines

○ Remove any extra header lines or comments not in expected format

Example correct .dAf format:

```
SEEPAGE RATES DATA FILE: PEAKPICKER OUTPUT
----------------------------------------------------------------
0.180 is the relative distance (in m) between sensors.
----------------------------------------------------------------
Data_Year Water_Day Ad_As A_Uncertainty Phase_Shift(days)
f_Uncertainty
2025 312.77777 0.57347131 1.00000000e-05 0.11111100 0.00100000
```

Amplitude Ratio Validation Failures

● **Error**: "Amplitude ratios outside physical bounds" or "Invalid A_r values"

● **Cause**: Amplitude ratios ≤ 0 or ≥ 1, indicating measurement or processing errors

● **Solutions**:

○ Verify amplitude ratios are between 0.01 and 0.99 (typical range 0.3-0.9)

○ Check peak picking results for proper shallow/deep amplitude calculation

○ Review temperature data for sensor malfunction or poor thermal coupling

○ Consider noise effects in low-amplitude signals

○ Verify sensor spacing and deployment configuration

Phase Shift Validation Issues

- **Error**: "Negative phase shifts detected" or "Phase values exceed physical limits"
- **Cause**: Incorrect phase calculation or temporal aliasing
- **Solutions**:
    - Verify phase shifts are positive (deep peaks lag shallow peaks)
    - Check phase shift range: typically 0.05-0.5 days for groundwater systems
    - Review peak picking for proper peak identification and matching
    - Consider signal period vs. phase shift relationship
    - Validate sensor orientation (shallow sensor actually shallower than deep)

Parameter Configuration Issues

Parameter File Problems

- **Error**: "Parameter file not found" or "Cannot parse tts_seepage.par"
- **Cause**: Missing parameter file or incorrect format
- **Solutions**:
    - Ensure tts_seepage.par exists in same directory as script
    - Verify parameter file format matches expected structure:

```
1.4          ---thermal conductivity (W/m*degC)
0.001, 0.001  ---transverse and longitudinal dispersivity (m)
996.5, 4179   ---density (kg/m^3) and heat capacity (J/kg degC) of the fluid
2650, 800     ---density (kg/m^3) and heat capacity (J/kg degC) of the grains
0.40 ---porosity
86400         ---period of analysis (1/frequency of signal, seconds)
1.0E-13, 1.0E-13  ---tolerance for Amp iterations, uncertainty iterations
1.0E-12, 1.0E-12  ---tolerance for Phase iterations, uncertainty iterations
0.0, 1.0          ---minimum, maximum permitted value for A function
0.001 ---limit on (dAr/dv) slope for numerical Amplitude limits
0.0, 2.0          ---minimum, maximum permitted value for f function (days)
60                ---limit on (df/dv) slope for numerical Phase limits
```

Thermal Property Validation Errors

- **Error**: "Unrealistic thermal properties" or "Property values outside expected ranges"

- **Cause**: Incorrect physical parameter values
- **Solutions**:
  - **Thermal conductivity (λ)**: Typical range 0.5-3.0 W/m°C for sediments
  - **Fluid density**: ~996-1000 kg/m³ for water at typical temperatures
  - **Fluid heat capacity**: ~4179 J/kg°C for water
  - **Solid density**: 1500-2800 kg/m³ depending on sediment type
  - **Solid heat capacity**: 600-1000 J/kg°C for typical minerals
  - **Porosity**: 0.1-0.6 for natural sediments
  - **Dispersivity**: 0.0001-0.01 m for typical probe spacings

Solver Convergence Issues

Amplitude Solver Failures

- **Error**: "Amplitude solver failed to converge" or "No solution found for amplitude equation"
- **Cause**: Numerical instability or inappropriate parameter combinations
- **Solutions**:
  - **Try different solver methods**:
    - Switch from Newton-Raphson to Hybrid method (most robust)
    - Use Brent's method for guaranteed convergence
    - Try Anderson acceleration for fixed-point problems
  - **Adjust solver tolerances**:
    - Increase tolerance from 1e-13 to 1e-10 or 1e-8
    - Reduce maximum iterations if computational time is excessive
  - **Check parameter bounds**:
    - Verify amplitude ratio limits (default: 0.0 to 1.0)
    - Adjust slope limits for numerical stability
  - **Validate input data**:
    - Remove extreme amplitude ratio values
    - Check for data points near amplitude bounds

Phase Solver Failures

- **Error**: "Phase solver convergence failed" or "Complex square root in phase equation"

- **Cause**: Mathematical constraints violated or numerical instability
- **Solutions**:
    - **Mathematical validation**:
        - Ensure phase equation discriminant > 0: $\alpha - 2((\Delta\varphi \times 4\pi K\_eff)/(dz \times T))^2 \geq 0$
        - Check velocity compatibility between amplitude and phase methods
    - **Solver method selection**:
        - Use Advanced Multi-Method for maximum reliability
        - Try constrained optimization for bounded solutions
        - Fall back to MATLAB legacy method if needed
    - **Parameter adjustment**:
        - Increase phase tolerance from 1e-12 to 1e-10
        - Verify phase shift limits (default: 0.0 to 2.0 days)
    - **Physical constraints**:
        - Validate sensor spacing measurement accuracy
        - Check period specification (86400 s for daily signals)

Numerical Method Selection Issues

- **Error**: "Selected solver method not available" or "Solver method failed"
- **Cause**: Missing dependencies or inappropriate method for data characteristics
- **Solutions**:
    - **Method availability check**:
        - Numba JIT: Requires numba installation (pip install numba)
        - Advanced methods: Require scipy and numpy
    - **Method selection guidelines**:
        - **Hybrid**: Best overall performance and reliability (recommended default)
        - **Newton-Raphson**: Fast but may fail for poorly conditioned problems
        - **Brent**: Guaranteed convergence but slower
        - **Numba JIT**: Fastest for large datasets but requires compilation

- **Fallback strategy**:
  - Always try Hybrid method first
  - Use MATLAB fixed-point as last resort
  - Document method performance for reproducibility

Uncertainty Calculation Issues

Uncertainty Library Problems

- **Error**: "Uncertainties library not available" or "Cannot propagate uncertainties"
- **Cause**: Missing uncertainties package or version incompatibility
- **Solutions**:
  - Install uncertainties library: pip install uncertainties==3.2.3
  - Verify installation: python -c "import uncertainties; print('OK')"
  - Use deterministic calculation if uncertainties not critical
  - Check for library version conflicts with other packages

Unrealistic Uncertainty Values

- **Error**: "Uncertainty exceeds 100% of calculated value" or "Negative uncertainties"
- **Cause**: Inappropriate uncertainty parameters or mathematical instability
- **Solutions**:
  - **Review uncertainty parameters**:
    - dz uncertainty: typically ±0.001 m (1mm measurement precision)
    - Temperature uncertainty: ±1-2% of amplitude ratio
    - Thermal conductivity: ±0.1-0.3 W/m°C (measurement precision)
    - Porosity: ±0.02-0.05 absolute (field measurement uncertainty)
  - **Validate uncertainty propagation**:
    - Check individual parameter contributions
    - Verify correlation handling between related parameters
    - Compare with Monte Carlo validation if available
  - **Uncertainty interpretation**:
    - Typical seepage uncertainty: 5-20% relative uncertainty
    - >30% uncertainty indicates poor measurement conditions

- <1% uncertainty may indicate underestimated parameter errors

Mathematical Theory and Implementation

Amplitude Ratio Method Theory

The amplitude ratio method solves for thermal velocity from temperature amplitude attenuation:

Governing equation:

$$A_r = |A_{deep}|/|A_{shallow}| = \exp[(dz/(2K_{eff})) \times (v - \sqrt{((\alpha + v^2)/2)})]$$

Where:

- A_r = amplitude ratio (dimensionless, $0 < A_r < 1$)
- dz = sensor spacing (m)
- $K_{eff} = \kappa + \alpha_T|v|$ = effective thermal diffusivity (m^2/s)
- $\kappa = \lambda/(\rho c)_{bulk}$ = thermal diffusivity (m^2/s)
- $\alpha = \sqrt{(v^4 + (8\pi K_{eff}/T)^2)}$ = dispersion parameter (m^2/s^2)
- v = thermal velocity (m/s)
- T = signal period (s)

Derived thermal properties:

$$(\rho c)_{bulk} = \varphi(\rho c)_{fluid} + (1-\varphi)(\rho c)_{solid}$$

$$\kappa = \lambda_{thermal} / (\rho c)_{bulk}$$

Numerical solution methods:

1. **Newton-Raphson iteration**:

$$f(v) = \exp[(dz/(2K_{eff})) \times (v - \sqrt{((\alpha + v^2)/2)})] - A_r = 0$$

$$v_{n+1} = v_n - f(v_n)/f'(v_n)$$

2. **Fixed-point iteration** (MATLAB legacy):

$$v_{n+1} = (2K_{eff}/dz) \times \ln(A_r) + \sqrt{((\alpha + v_n^2)/2)}$$

Convergence criteria:

- $|f(v)| < 1e\text{-}12$ (absolute residual)
- $|v_{new} - v_{old}|/|v_{old}| < 1e\text{-}10$ (relative change)
- Maximum iterations: 50-80,000 depending on method

Phase Shift Method Theory

The phase shift method determines thermal velocity from temperature phase lag:

Governing equation:

$\Delta\varphi = (dz/(4\pi K\_eff)) \times \sqrt{(\alpha - v^2)} \times T$

Solving for thermal velocity:

$v = \sqrt{[\alpha - 2((\Delta\varphi \times 4\pi K\_eff)/(dz \times T))^2]}$

Mathematical constraints:

- Discriminant must be positive: $\alpha - 2((\Delta\varphi \times 4\pi K\_eff)/(dz \times T))^2 \geq 0$
- Phase shift must be positive: $\Delta\varphi > 0$
- Velocity must be real and finite

Physical interpretation:

- Positive phase shift: deep temperature lags shallow (physical requirement)
- Phase shift magnitude: typically 0.05-0.5 days for groundwater systems
- Relationship to flow: larger phase shifts indicate slower thermal velocities

Velocity Sign Determination:

Flow direction is determined from amplitude-based solution:

```
zero_v_alpha = (8π × κ) / T
zero_v_Ar = exp(-dz / (2κ) × √(zero_v_alpha / 2))

if A_r > zero_v_Ar:
    flow_direction = negative  # downward infiltration
else:
    flow_direction = positive  # upward discharge
```

Seepage Rate Conversion:

Convert thermal velocity to specific discharge:

```
q_seepage = [v_thermal × (ρc)_bulk / (ρc)_fluid] × 86400  # m/day
```

Sign convention:

- Positive: upward flow (groundwater discharge)
- Negative: downward flow (surface water infiltration)

Uncertainty Propagation Theory

Linear error propagation formula:

$\sigma^2\_q = \Sigma_i (\partial q/\partial x_i)^2 \sigma^2_i + 2\Sigma_i\Sigma\square (\partial q/\partial x_i)(\partial q/\partial x\square) \sigma_i\square$

Major uncertainty sources:

1. **Measurement uncertainties**:
   - Sensor spacing: ±1-2 mm typical field precision
   - Temperature amplitude: ±1-5% sensor precision
   - Phase timing: ±0.01 days typical
2. **Parameter uncertainties**:
   - Thermal conductivity: ±5-20% (literature/measurement)
   - Porosity: ±0.02-0.05 absolute (field variability)
   - Fluid properties: ±1-2% (temperature dependence)
3. **Numerical uncertainties**:
   - Solver convergence: ±0.1-1% (tolerance dependent)
   - Discretization: ±0.1% (temporal resolution)

Typical uncertainty magnitudes:

- Research quality: 5-15% relative uncertainty
- Field applications: 10-30% relative uncertainty
- Poor conditions: >30% relative uncertainty

Advanced Solver Implementation

Hybrid Method Algorithm:

```
def solve_hybrid(A_r, params):
    # Stage 1: Brent's method with bracketing
    try:
        v_low, v_high = bracket_root(amplitude_eq, v_init)
        result = brentq(amplitude_eq, v_low, v_high, xtol=1e-12)
        return result, 'brent'
    except:
        pass

    # Stage 2: Newton-Raphson with auto-differentiation
    try:
        result = newton(amplitude_eq, v_init, fprime=amplitude_eq_prime,
```

```
                tol=1e-12, maxiter=50)
        return result, 'newton'
    except:
        pass

    # Stage 3: SciPy root_scalar hybrid
    try:
        result = root_scalar(amplitude_eq, x0=v_init, method='newton')
        return result.root, 'scipy'
    except:
        pass

    # Stage 4: Robust fsolve fallback
    result = fsolve(amplitude_eq, v_init, xtol=1e-12)
    return result[0], 'fsolve'
```

Performance characteristics:

- Success rate: 90-95% for typical datasets
- Speed: 5-20× faster than MATLAB fixed-point
- Reliability: Automatic fallback prevents total failure

Data Quality and Validation Issues

Seepage Rate Range Validation

- **Error**: "Seepage rates outside realistic range"
- **Cause**: Extreme calculated values indicating measurement or processing errors
- **Solutions**:
    - **Typical seepage ranges**:
        - Groundwater discharge: 0.01-5.0 m/day
        - Infiltration rates: -0.1 to -2.0 m/day
        - Values >10 m/day: likely measurement errors
    - **Validation checks**:

        - Compare amplitude and phase method results
        - Check for consistency across time series
        - Validate against independent flow measurements
    - **Data filtering**:
        - Remove obvious outliers (>3σ from mean)
        - Flag values outside physical bounds
        - Check for systematic biases in results

Flag Generation and Interpretation

- **Error**: "Excessive flagged results" or "All calculations flagged"
- **Cause**: Poor data quality or inappropriate parameter values
- **Solutions**:
    - **Flag meanings**:
        - Flag 0: Successful calculation
        - Flag 1: Amplitude solver convergence failure
        - Flag 2: Phase solver convergence failure
        - Flag 3: Result outside physical bounds
        - Flag 4: Mathematical constraint violation
    - **Flag reduction strategies**:
        - Adjust solver tolerances (increase to 1e-10 or 1e-8)
        - Use more robust solver methods (Hybrid vs Newton)
        - Review input data quality and preprocessing
        - Check parameter values for physical reasonableness

Temporal Consistency Issues

- **Error**: "Large temporal variations in seepage rates"

- **Cause**: Natural variability, measurement noise, or processing artifacts
- **Solutions**:
    - **Expected variability**:
        - Seasonal patterns: factor of 2-5 typical
        - Event responses: order of magnitude changes possible
        - Random noise: ±10-20% of signal typical
    - **Smoothing and filtering**:
        - Apply moving average to reduce noise
        - Check for systematic trends or drift
        - Compare with hydrologic forcing (precipitation, stage)
    - **Quality assessment**:
        - Calculate temporal derivatives (dq/dt)
        - Identify and investigate extreme changes
        - Cross-validate with other measurement methods

Performance and Memory Issues

Slow Calculation Performance

- **Error**: "Calculations taking excessive time"
- **Cause**: Large datasets, complex parameter combinations, or inefficient methods
- **Solutions**:
    - **Method optimization**:
        - Use Numba JIT for maximum speed (10-100× improvement)
        - Disable uncertainty calculations if not needed
        - Use simpler solver methods for initial screening
    - **Data management**:
        - Process data in smaller time chunks
        - Remove unnecessary data points before calculation

        - Use appropriate data types (float64 vs float32)
    - **Hardware considerations**:
        - Ensure adequate RAM (4GB+ for large datasets)
        - Use SSD storage for faster file I/O
        - Close memory-intensive applications

Memory Errors with Large Datasets

- **Error**: "MemoryError" or "System out of memory"
- **Cause**: Insufficient RAM for uncertainty calculations or large arrays
- **Solutions**:
    - **Memory reduction**:
        - Disable uncertainty propagation temporarily
        - Process data in smaller batches
        - Use data chunking for very large files
    - **System optimization**:
        - Increase virtual memory/swap space
        - Close other applications
        - Use 64-bit Python for larger memory access
    - **Algorithm efficiency**:
        - Use vectorized operations where possible
        - Avoid creating unnecessary intermediate arrays
        - Clear large variables when no longer needed

Output and Export Issues

File Export Failures

- **Error**: "Cannot write output files" or "Permission denied"
- **Cause**: Insufficient permissions or disk space issues
- **Solutions**:
    - Check available disk space (ensure >100MB free)

- Verify write permissions in output directory
- Avoid special characters in output filenames
- Close any open output files in other applications
- Try different output directory if needed

Incorrect Output Format

- **Error**: "Export file format incorrect" or "Missing uncertainty columns"
- **Cause**: Software configuration or export option selection
- **Solutions**:
  - **Verify export format selection**:
    - .sep files: Main seepage results with uncertainty columns
    - .spA files: Detailed amplitude-based solver output
    - .spf files: Detailed phase-based solver output
  - **Check output structure**:
    - Header includes processing parameters and metadata
    - Uncertainty columns present when calculations enabled
    - Proper decimal precision (8 decimal places for WaterDay)
  - **Validate exported data**:
    - Compare against displayed results
    - Check for missing or corrupted data points
    - Verify timestamp and value formatting

Cross-Validation and Quality Control

Method Comparison Validation

- **Error**: "Large discrepancies between amplitude and phase methods"
- **Cause**: Mathematical inconsistencies or data quality issues
- **Solutions**:
  - **Expected agreement**:
    - Good data: methods agree within ±20%
    - Moderate data: methods agree within ±50%

        - Poor data: methods may disagree significantly
    - **Diagnostic procedures**:
        - Plot amplitude vs phase results for correlation analysis
        - Calculate systematic bias between methods
        - Check for parameter sensitivity differences
    - **Resolution strategies**:
        - Prioritize amplitude method for direction determination
        - Use phase method for magnitude validation
        - Average methods only when good agreement exists

Physical Reasonableness Checks

- **Error**: "Results violate physical constraints"
- **Cause**: Mathematical artifacts or measurement errors
- **Solutions**:
    - **Thermodynamic consistency**:
        - Heat flow direction should match calculated flow direction
        - Amplitude attenuation should be physically reasonable
        - Phase lag should be consistent with thermal properties
    - **Hydrologic reasonableness**:
        - Compare with regional groundwater flow patterns
        - Check consistency with surface water levels
        - Validate against precipitation and evapotranspiration
    - **Scale considerations**:
        - Point measurements vs. representative volumes
        - Temporal averaging vs. instantaneous values
        - Local vs. regional flow system impacts

System and Environment Issues

Browser Interface Problems

- **Error**: "Cannot access application" or "Interface not loading"
- **Cause**: Port conflicts, firewall issues, or browser compatibility
- **Solutions**:

    - **Network configuration**:
        - Verify application starts on http://127.0.0.1:8053
        - Check for firewall blocking local connections
        - Try different port if 8053 is occupied
    - **Browser compatibility**:
        - Use modern browsers: Chrome 90+, Firefox 88+, Safari 14+
        - Enable JavaScript and disable blocking extensions
        - Clear browser cache and cookies
        - Try incognito/private browsing mode

Library Version Conflicts

- **Error**: "ImportError" or "Module version incompatibility"
- **Cause**: Conflicting package versions or missing dependencies
- **Solutions**:
    - **Dependency verification**:
        - Check required package versions match documentation
        - Use virtual environment for isolated installation
        - Reinstall packages if version conflicts detected
    - **Installation troubleshooting**:
        - Use recommended installation command from documentation
        - Update pip/conda to latest version
        - Check for Python version compatibility

Operating System Compatibility

- **Error**: "Platform-specific errors" or "Path resolution issues"
- **Cause**: OS-specific file handling or library differences
- **Solutions**:
    - **Path handling**:
        - Use forward slashes (/) in file paths universally
        - Avoid spaces and special characters in filenames
        - Use absolute paths if relative paths fail
    - **Platform testing**:
        - Test on target platform before production use

- Document any OS-specific configuration requirements
- Consider using Windows Subsystem for Linux (WSL) if needed

VII. Liscence and Citation

This software is provided for research and educational purposes.
If using in published research, please cite appropriately.

Purpose: Temperature time-series processing, frequency analysis, peak picking, and seepage analysis for UCSC Hydrology

Created by Timothy Wu from the weeks of 8/23/2025 - 8/15/2025 during CITRIS WIP Internship

The software is provided as-is without warranty. Users are responsible for validating results and ensuring appropriateness for their specific applications.

All final versions of code are found here: Github