TerraPCA

Telluric Empirical Reduction & Reconstruction Analysis via PCA

Noriyuki Matsunaga
2025 September 17
(Version 1.0)

Abstract

Ground-based near-infrared (NIR) spectra are strongly affected by time-variable telluric absorption. Direct modeling is challenging due to the complexity and variability of atmospheric conditions. To model this absorption in observational spectra empirically, the PCA approach is useful and commonly used now. Based on spectra of a sample of hot, feature–poor standard stars under various airmass and weather conditions, we can model the telluric lines with a small set of principal components (PCs). Here we provide the scripts and a data set for constructing telluric absorption models of WINERED spectra. With six PCs, the telluric profiles are reproduced with root-mean-square (RMS) residuals typically below 1% over 11 WINERED WIDE-mode orders (43–48, 52, 55–58). We ignore the orders 53 and 54, in which the telluric absorption is almost negligible. This toolkit facilitates robust telluric correction for WINERED WIDE-mode spectra and is adaptable to other instruments.

Analysis flow and helper scripts

Step 1. PCA basis (once)

```
build_models.py          # build PCA basis spectra using spectra of standards
```

Step 2. Telluric masks (once, all orders)

```
list_tel_abs.py          # detect and list strong telluric absorption pixels
```

Step 3. Quick self-check (optional, per order)

```
check_models.py          # validate the PCA basis on the training dataset
```

Step 4. Fit science spectra

```
fit_model.py             # fit telluric model to science spectra
```

Some utility scripts

```
list_standard_data.py    # List available spectra of standard stars for a given setting
fits_to_txt.py           # Convert FITS to text spectra
```

```
mask_from_spectrum.py          # List wavelength ranges with lines (or continuum ranges) to mask
calc_sp_offset.py              # Fit RV/flux offset between spectra
crosscorr_sp_offset.py         # Cross-correlate two spectra to estimate RV shift
measure_telluric_offsets.py    # Measure wavelength offsets based on telluric lines across orders
```

Many scripts print out required arguments and options with the “-h” option. Some scripts give more

Instrumental Settings and Applicability to Other Spectrographs

Although **TerraPCA** was originally developed for the WINERED/WIDE mode, it is designed to be flexible and adaptable to other instrumental configurations (e.g. WINERED/HIRES-Y, or completely different spectrographs).

Setting configuration files

Each spectrograph configuration is described in a text file named:
`settings/setting_<label>.txt`
where `<label>` is an arbitrary name such as `WINERED_WIDE` or `HIRES_Y`.
Each line in this file defines one spectral order and its basic parameters:

```
# order   wmin      wmax      n_pix  n_base
m44       12570.0   12930.0   3072   6
```

- **order** – identifier of the order (free string, e.g. `m44`)
- **wmin / wmax** – expected wavelength coverage of the order (in Å, vacuum or air)
- **n_pix** – number of pixels in the rebinned wavelength grid used for PCA
- **n_base** – number of PCA components (eigenvectors) to retain for this order

The setting file thus defines the analysis grid and PCA dimensionality for all orders.

How it is used

All major scripts take a `--setting` argument (default: `utils.default_setting`), and internally load the configuration via:

```
setting_orders = utils.load_setting(setting_label)
```

This governs:

- the wavelength grid created by `make_order_waves()`
- which files are read from `data/{setting_label}/{order}/`

- where the PCA models are saved:
  `models_txt/{setting_label}/base_{order}.txt` and
  `ave_{order}_{vac|air}.txt`

Scripts also support the `--vac_air` option to distinguish between **air** and **vacuum** wavelength scales; the model grids and average spectra are saved in both scales.

How to add a new setting

1. Prepare a directory `data/<setting_label>/<order>/` for each order.
   Place all telluric standard spectra (FITS or 2-column text) there.
2. Create a `settings/setting_<setting_label>.txt` file describing all orders.
   Adjust `n_pix` and `n_base` based on the pixel scale and expected PCA complexity.
3. Run:
   ```
   python3 build_models.py --setting <setting_label> --vac_air vac
   bash make_tel_abs_<setting_label>.sh
   ```
4. Use the resulting models for telluric correction of science spectra with:
   ```
   python3 fit_model.py --setting <setting_label> ...
   ```

Notes

- Orders can be freely named, as long as they are consistent across the setting file and directory structure.
- The package does not assume any particular dispersion or instrument-specific header keywords.
- This structure allows easy support of multiple instruments in parallel, simply by selecting the appropriate `--setting`.

Data preparation

Throughout the analyses with TerraPCA, spectra should be normalized, at least approximately, to have the continuum to be the unity. Each telluric line needs to be aligned in wavelength, although small offsets may be adjusted within each task. Scripts **`calc_sp_offset.py`** and **`measure_telluric_offsets.py`** would help to check the offsets. Wavelengths can be either **vacuum or air**, specified when building models.

- Directory structure for telluric standard spectra:
  ```
  data/{setting_label}/{order}/*.txt or *.fits
  ```
- Both `.txt` (2-column wavelength, flux) and `.fits` files are accepted.

- Spectra are located automatically using `utils.list_telluric_files(setting, order)`.
- A utility tool, **`list_standard_data.py`** (see below for more details) is available to check available data for a given setting.

Construction of basis spectra using PCA

`build_models.py` builds PCA models from telluric standards:

- Reads `setting_[label].txt`, which gives per-order configuration:
  `m44 12570 12930 800 6` (order, wmin, wmax, n_pix, n_base)
- User needs to specify `--vac_air vac|air`
- Collects spectra from `data/{setting}/{order}/`
- Optionally aligns each spectrum to the running mean using `calc_sp_offset`
- Interpolates to a fixed wavelength grid (`make_order_waves`)
- Performs PCA to produce:
  - `base_{order}.txt`
  - `ave_{order}_{vac_air}.txt`
  - `waves_{order}_{vac_air}.txt` (and converted to the opposite scale)
- Diagnostic plots (eigenvalues and basis spectra) are saved in `models_plot/{setting}/`.

Notes on Construction of basis spectra using PCA

The PCA basis spectra used to model telluric absorption are constructed using the script **`build_models.py`**, which combines a set of observed spectra of telluric standard stars taken under different atmospheric conditions. For each instrumental configuration (called a *setting*), the basic information about each spectral order—including its wavelength coverage, number of pixels, and number of PCA components—is provided in a configuration file (`settings/setting_<setting>.txt`). The script first gathers all available standard spectra stored under `data/<setting>/<order>/`, either in FITS or plain-text format, and constructs a fixed wavelength grid of `n_pix` points between the specified `wmin` and `wmax`. The user must indicate whether these spectra are in vacuum or air wavelength (`--vac_air`), and the code saves both the grid and its conversion to the opposite scale (using `utils.vac_to_air` and `utils.air_to_vac`) so the resulting models can be applied to spectra in either scale.

Each spectrum is interpolated onto this grid using a cubic spline, with any pixels outside the observed range filled with a continuum value of 1.0. To reduce alignment errors, spectra after the first can be shifted in wavelength and flux to minimize residuals relative to the running mean spectrum using `calc_sp_offset`. The rebinned spectra are then standardized (mean-subtracted and divided by their standard deviation) and stacked into a matrix **X** (m

spectra × n pixels). A covariance matrix of shape m×m is computed from X, and eigenvalues and eigenvectors are obtained via symmetric eigen-decomposition. The leading `n_base` eigenvectors (those with the largest eigenvalues) are projected back to the pixel space as basis vectors `A = Xᵀ @ Ep` (n × n_base), which represent the main modes of variation in the telluric lines. Their signs are flipped where necessary so that absorption features consistently point downward. The mean of the original (non-standardized) spectra is then fitted as a linear combination of these basis vectors to produce an average telluric spectrum. These outputs are saved as `base_{order}.txt` (basis), `waves_{order}_{vac|air}.txt` (grid), and `ave_{order}_{vac|air}.txt` (average spectrum) under `models_txt/<setting>/`. For diagnostics, `build_models.py` also produces plots of the eigenvalue spectrum and of the PCA basis vectors overlaid with the mean spectrum under `models_plot/<setting>/`.

We acknowledge the PCA decomposition technique, applied to stellar spectra, presented in Singh et al. (1998, *MNRAS*, 295, 312), adapted to robustly model the telluric absorption features under varying atmospheric conditions and to support different instrumental setups.

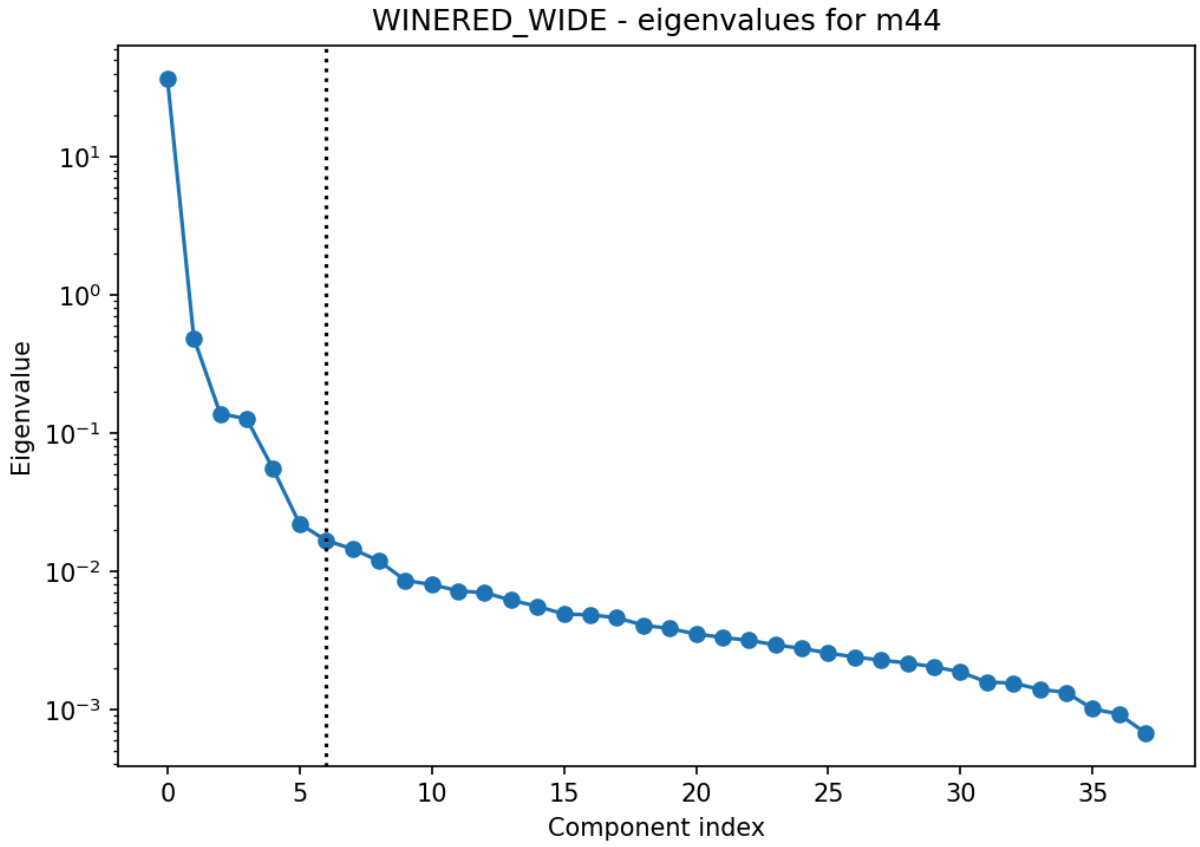

Figure 1 Eigen-value plot (log-scale) showing that the first six PCs capture >99 % of the variance.

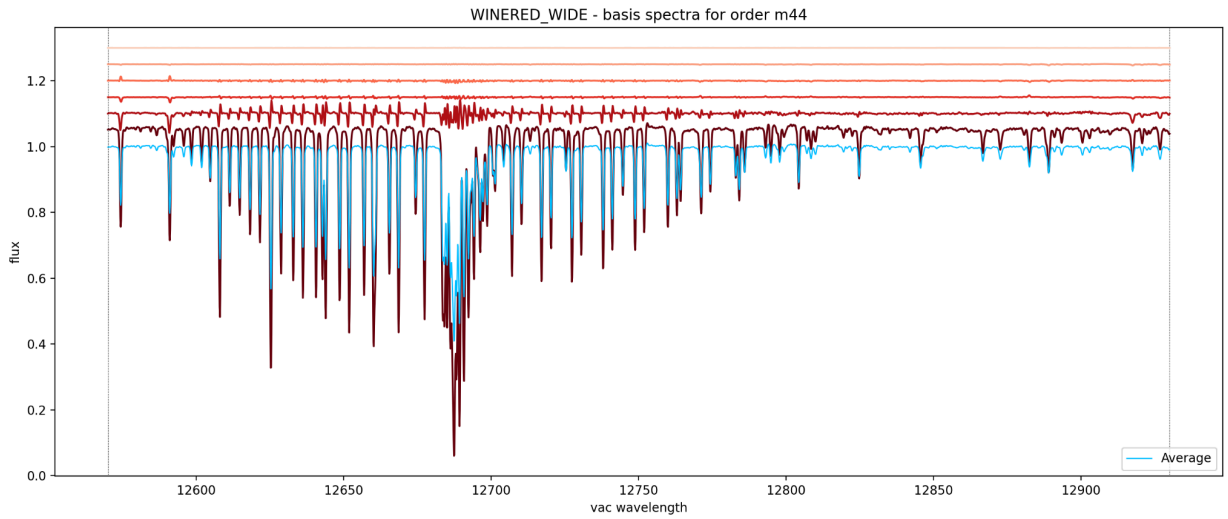

Figure 2 Basis spectra overlaid with average spectrum (black) and PCA reconstruction (cyan).

Identifying telluric absorption pixels

`list_tel_abs.py` marks regions of strong telluric absorption from the average spectrum:

- Input: `ave_{order}_{vac_air}.txt`
- Finds pixels with flux below threshold, expands ±3 pixels (controllable with the -b option), merges adjacent regions
- Produces:
  `models_txt/<setting>/tel_abs_{order}_vac.txt`
  `models_txt/<setting>/tel_abs_{order}_air.txt`
- Saves diagnostic plots to:
  `models_plot/{setting}/tel_abs_{order}.png`
- A bash script, **`list_tel_abs.sh`**, is prepared for the WINERED_WIDE setting.

These regions are used later to constrain RV alignment and to mark regions strongly affected by telluric lines.

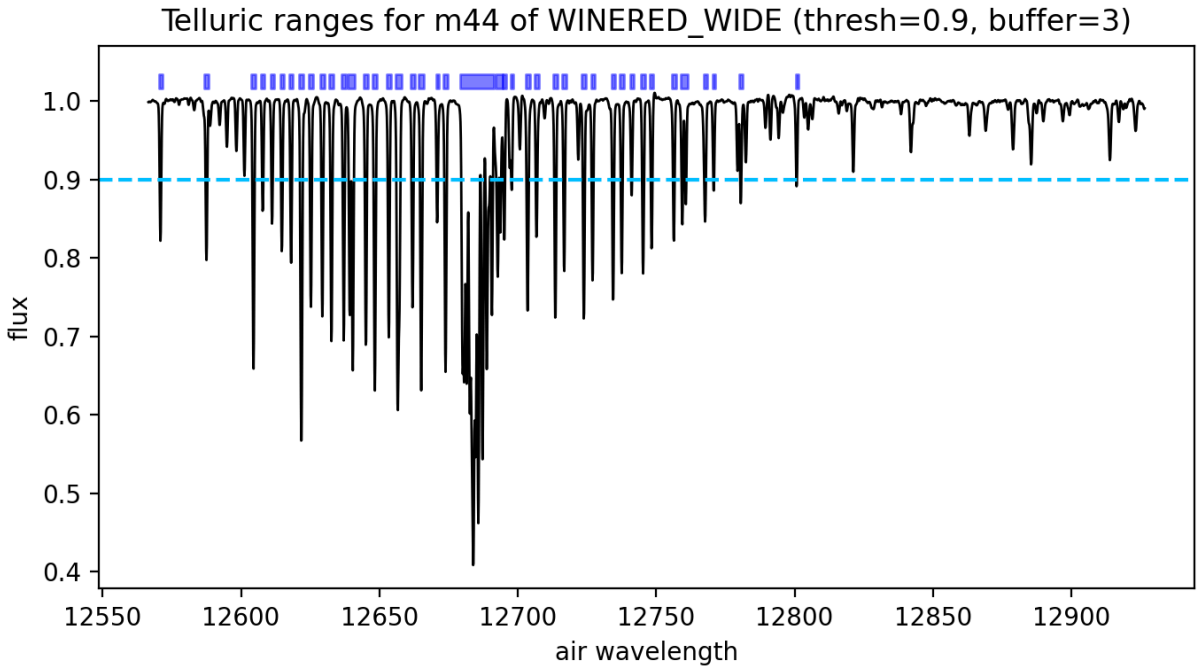

Figure 3 Wavelength ranges with significant telluric absorption.

Modeling telluric absorption in the training set

`check_models.py` validates the PCA basis:

- Fits each telluric standard spectrum using `base_{order}.txt`
- Optionally aligns with `calc_sp_offset`
- Reports the RMS of residuals
- Saves diagnostic plots to:
  `check_models/{setting}/{order}/`
- A bash script, **`check_models.sh`**, is prepared for the WINERED_WIDE setting.

Modeling telluric absorption in science spectra

The script **`fit_model.py`** fits the PCA-based telluric absorption model to an observed spectrum for a single spectral order. It uses the PCA basis vectors (`base_{order}.txt`), wavelength grid (`waves_{order}_{vac|air}.txt`), average telluric spectrum (`ave_{order}_{vac|air}.txt`), and telluric absorption regions (`tel_abs_{order}_{vac|air}.txt`) that were previously generated by `build_models.py` and `list_tel_abs.py`.

The observed spectrum (in text or FITS format) is first loaded and optionally trimmed at its edges (not altering the input file). The user must specify whether its wavelength scale is in vacuum or air with the `--vac_air` option. The code then:

- Interpolates the spectrum onto the PCA model grid.
- Optionally aligns it in wavelength by calling `calc_sp_offset`, using only pixels within the strong telluric regions.
- Iteratively fits the observed spectrum as a linear combination of the PCA basis spectra plus an additive offset.
    - After each iteration, pixels with large residuals are sigma-clipped (via `clip_parts`) and masked out in the next iteration.
- Pixels predicted to have strong intrinsic object absorption can be excluded by providing a model spectrum to `--mask_obj`.
- The continuum level of the spectrum is measured (via `utils.calc_continuum`) and used to normalize the result.
- The final best-fit telluric model is saved on the same wavelength grid as the input, either as a text file or a new FITS file.
- The script also prints the RMS values of the residuals (`sigma1` and `sigma2`) and the numbers of pixels actually used in the fit, and can optionally append this information to a user-specified diagnostics file. Excluding rejected pixels, `sigma1` considers pixels within the `tel_abs` ranges only, while `sigma2` considers the entire range.
- It supports verbose mode for detailed logging and accepts user-defined rejection regions (`--reject1` and `--reject2`).

Typical usage:

```
python3 fit_model.py m44 object_m44.txt model_m44.txt \
  --setting WINERED_WIDE --vac_air vac --xadjust \
  --plot_out fit_m44.png --reject1 mask.txt
```

Notes

- Two estimates of the standard deviation (RMS) of the difference between the observed and model spectra for each star are printed as a diagnostic, one for the pixels within the `tel_abs` ranges and the other for the entire range, excluding masked pixels. Usually, the two values are similar.
- While we expect the PCA telluric fitting works reasonably well even with some contamination of intrinsic lines of the target, the RMS calculation can be easily affected by the intrinsic lines unless they are marked out with the `--mask_obj` or `--reject1` option.In case of the WINERED WIDE data, the standard deviation is usually about 1-2 % for stars with intermediate spectral types with using one of the rough masks in the `data/obj_model` directory.

- It would be difficult to find a good telluric model fit to spectra with too many intrinsic lines.Telluric absorption would be overestimated unless the task finds appropriate wavelength ranges with telluric lines but without significant contamination of intrinsic lines. A more robust approach in such a case may be to fit the PCA-based telluric model to the spectrum of a telluric standard observed in a similar condition (time and air mass) and adjust this model to reproduce the telluric absorption in the target's spectrum using a classical tool like IRAF/telluric.

Plot description

If `--plot_out` is given, the script produces a diagnostic plot with two panels:

- **Top panel:**
  - Observed spectrum (red) after continuum normalization
  - Best-fit telluric model (blue)
  - Shaded bars show masked regions:
    - Blue = strong telluric absorption (`tel_abs`)
    - Gray = sigma-clipped (`reject2`)
    - Red = user-defined (`reject1`)
  - (Optional) Predicted object absorption (`--mask_obj`) is shown as dashed green lines.
- **Bottom panel:**
  - Residual spectrum (observed − model) in gray
  - The blue curve shows the model offset from unity
  - Same shaded regions as the top panel
  - The reported RMS of the residuals (σ) and the wavelength coverage are also annotated.

This plot provides a quick visual validation of the fitting quality and highlights which pixels were included or rejected in the fit.

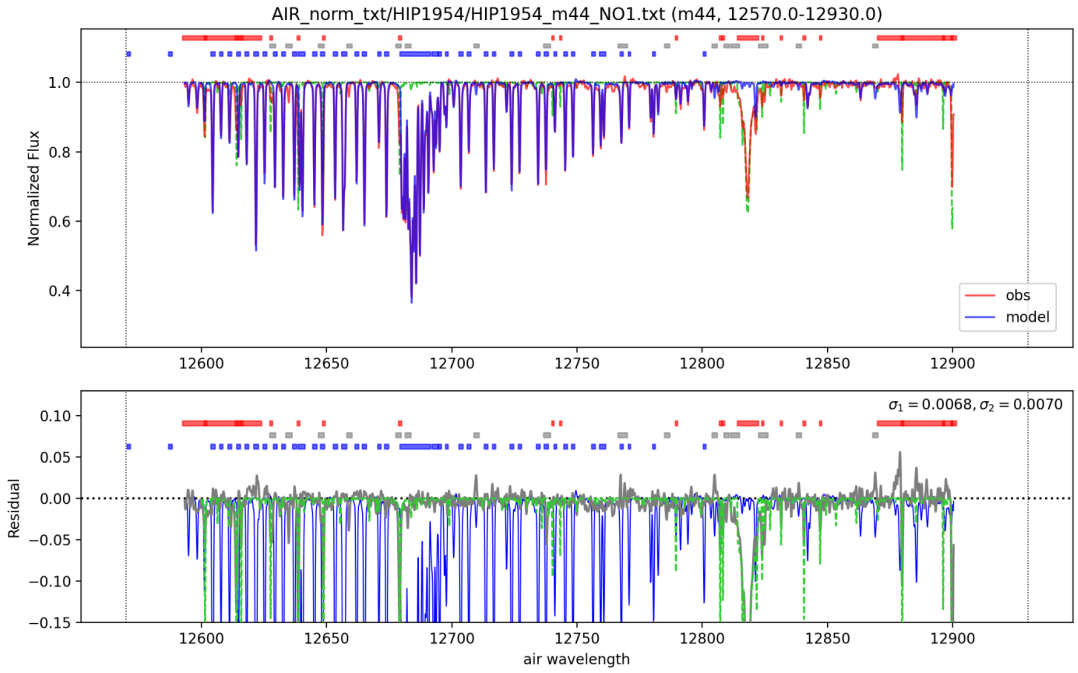

Figure 4 Output from `fit_model.py` showing the best-fit telluric model (blue) compared to the observed spectrum (red). Shaded bars mark masked pixels: blue (telluric absorption), gray (sigma-clipped), red (user-defined). The lower panel shows residuals (observed − model) with the RMS (σ) annotated. The green curve is the user-supplied predicted object spectrum with the –mask_obj option (typically a stellar or planetary atmospheric model) plotted for reference.

Additional utility tools

Besides the main PCA modelling pipeline, this package includes several utility scripts that help identify spectral features, measure wavelength offsets, and mask intrinsic stellar lines during telluric correction. These tools are often useful for preparing and diagnosing input data before fitting.

`fits_to_txt.py` - Converting a FITS spectrum to a text file

Purpose:
This utility converts a **1D FITS spectrum** into a simple **2-column text file** containing wavelength and flux values, which are easier to inspect, plot, or use in downstream analysis scripts (like PCA-based telluric modeling).

How it works:

- The script opens the input FITS file and reads the **flux array** and **header keywords** (`CRVAL1`, `CRPIX1`, `CDELT1`).
- It computes the **wavelength scale** from the FITS World Coordinate System (WCS) keywords as:
  $\lambda_i = \mathrm{CRVAL1} + (i - \mathrm{CRPIX1}) \times \mathrm{CDELT1}$
- It writes these wavelengths and fluxes to the specified text file in two columns: `wavelength[Å] flux`.

Optional features:

- `--write_header`: When enabled, all FITS header keywords and values are written at the top of the output text file as comment lines (`# ...`).
  - These comment lines are **ignored by `numpy.loadtxt`** (and similar readers) when loading the text later, so they do not interfere with data reading.
- `--plot`: Opens a simple Matplotlib plot of the spectrum for quick inspection.

Typical usage:

```
python3 fits_to_txt.py input.fits output.txt --write_header --plot
```

This will create `output.txt` containing:

```
# input.fits
# CRVAL1 = ...
# CDELT1 = ...
...
5000.000000   1.234567
5000.050000   1.210432
...
```

Notes:

- Only supports **1D spectra** (a single flux array in the primary HDU).
- Wavelengths are assumed to be **linear** and in the same units as `CRVAL1`/`CDELT1` (usually Angstroms).

`list_standard_data.py` — Listing Available Telluric Standard Spectra

This script is a **data inventory and quality check utility**.
It scans the directory structure for all **telluric standard star spectra** associated with a given

instrumental setting (defined in `setting_<label>.txt`) and **summarizes their wavelength coverage order by order**.

This is especially useful when you are preparing to build the PCA basis (`build_models.py`): it lets you confirm that enough standard spectra are available and that they sufficiently cover the expected wavelength ranges for each order.

How It Works

1. **Load Order Definitions**
   - The script first loads the `setting_<label>.txt` file via `utils.load_setting()`.
   - This file defines, for each order, its expected wavelength range (`wmin`, `wmax`) and the number of pixels (`n_pix`) used to build the model.

2. **Find Available Files**
   - For each order, it uses `utils.list_telluric_files(setting, order)` to gather all standard-star spectra files associated with that order.

3. **Evaluate Wavelength Coverage**
   - It loads each spectrum using `utils.load_sp()`, then measures the min and max wavelengths present in the file.
   - It computes the **coverage fraction** = (overlap with expected order range) / (expected order width).
   - Based on this fraction, it labels the coverage as:
     - **OK** if ≥ 90% of the order's range is covered
     - **Partial** if ≥ 50% but < 90% is covered
     - **NG** if < 50% coverage or failed to load

4. **Print Summary**
   - For each file, it prints:
     `filename λmin - λmax (n px) [OK / Partial / NG, coverage=%]`
   - It also prints the total number of files classified as OK, Partial, and NG for each order.

Example Usage

```
python3 list_standard_data.py --setting WINERED_WIDE
```

Example output:

```
# Setting: WINERED_WIDE
[m44] range=10300.0:10450.0, n_pix=2200, n_base=6
```

```
  telluric_m44_20230601.txt    10295.2 - 10455.8 (2174 px) [OK,
coverage=100%]
  telluric_m44_20230602.txt    10350.7 - 10410.2 ( 740 px) [Partial,
coverage=55%]
  -> N(OK)=1
     N(Partial)=1
```

When to Use

- Before running `build_models.py`, to confirm you have **enough spectra with full coverage**.
- To check which orders have **few or no standard data**, helping plan additional observations.
- To diagnose missing or truncated files after data conversion.

`mask_from_spectrum.py` - listing the wavelength ranges of target's features or continuum

This utility identifies regions of significant absorption (or emission) in a given spectrum and outputs them as a list of wavelength intervals. These intervals can be used as masks in other scripts, such as `fit_model.py` or `calc_sp_offset.py`, to exclude object lines or emission features from the telluric fitting process.

The script takes a two-column spectrum file (wavelength, normalized flux) and finds contiguous pixels where the flux falls below a specified threshold (or above, if `--invert` is used). Optionally, each region is expanded by a given number of pixels (`--buffer_pix`) to ensure full coverage of the feature. You can also apply a radial-velocity shift (`--rv_to_add`) to match the expected object velocity and even convert between vacuum and air wavelengths (`--conversion`).

Internally, the tool:

- Shifts the wavelength grid by the requested RV offset.
- Optionally converts the wavelength scale (`vac_to_air` or `air_to_vac`).
- Creates a Boolean mask of all pixels satisfying the threshold condition.
- Groups contiguous masked pixels into ranges and merges overlapping or adjacent ranges.
- Returns a compact list of `(start, end)` wavelength intervals.

Example Usage

```
python3 mask_from_spectrum.py object_model.txt -t 0.97 -b 3 -v 25 \
    --conversion vac_to_air -p object_mask.png
```

This command:

- Masks all pixels with flux `< 0.97`.
- Expands each region by 3 pixels on both sides.
- Applies a +25 km/s radial velocity shift to the wavelengths.
- Converts wavelengths from vacuum to air scale.
- Saves a PNG diagnostic plot marking masked regions in blue.

The output is a comma-separated list of wavelength ranges:
`10420.250:10421.380,10426.100:10427.950`
You can redirect this to a file and later feed it into `fit_model.py` via `--reject1`.

Output Plot (Optional)

If `--plot_out` is given, the script produces a figure showing:

- **Black curve:** the input spectrum (after any RV shift/conversion).
- **Blue shaded regions:** the masked intervals.
- **Threshold line:** implied by the title (`f < threshold` or `f > threshold`).

This diagnostic plot is helpful to visually verify that the masked regions capture only the desired features and do not accidentally remove too much continuum.

`crosscorr_sp_offset.py` — Estimating Radial Velocity via Cross-Correlation

This script estimates the relative wavelength offset (radial velocity shift) between two spectra using normalized cross-correlation.

How It Works

- The object and reference spectra are interpolated to a common wavelength grid.
- A cross-correlation function is computed over a velocity range.
- The peak of the cross-correlation gives the best-fit velocity shift (km/s).

Example

```
python3 crosscorr_sp_offset.py obj.txt ref.txt --wmin 10420 --wmax
10460
```

Use Cases

- Quick-look estimate of spectral shifts between observed frames.
- Checking if an object spectrum is roughly aligned with the telluric reference.

This method is fast but purely correlation-based: it does not handle continuum mismatches, masking, or flux offsets, so it is best for initial estimates.

`calc_sp_offset.py` — Precise Offset Fitting by Minimizing Residuals

While `crosscorr_sp_offset.py` gives a quick estimate, `calc_sp_offset.py` performs a more rigorous fit of both the **radial velocity (wavelength)** and **flux** offsets between an object spectrum and a reference (typically the average telluric spectrum of an order). It minimizes the standard deviation of residuals between the two spectra after cubic-spline interpolation and optional masking.

Features

- Fits both wavelength (RV) shift and vertical flux offset.
- Accepts `--use_ranges` and `--reject_ranges` masks to focus on clean telluric regions.
- Allows `--fmin` and `--fmax` to reject low-S/N pixels.
- Supports iterative sigma clipping to remove outliers.

Example

```
python3 calc_sp_offset.py obj_m44.txt ref_m44.txt \
    --use_ranges 10430:10460 --fmin 0.2 --fmax 1.3 --plot_out
offset.png
```

The script outputs the best-fit velocity shift (km/s), flux offset, and number of used pixels.
The plot shows the reference and shifted object spectra with masked regions shaded.

This tool is used inside `fit_model.py` (when `--xadjust` is enabled) to refine the wavelength alignment before telluric fitting.

`measure_telluric_offsets.py` — Measuring Systematic Wavelength Offsets Across Orders

This script is designed to **systematically measure radial velocity (RV) offsets** between your **observed science spectra** and the **PCA-based telluric model average spectra** (`ave_{order}_{vac/air}.txt`) for **all spectral orders** defined in a given `setting_*.txt` configuration.

It automates the process of calling `calc_sp_offset.py` for each order, focusing on wavelength regions dominated by telluric absorption (`tel_abs_{order}_{vac/air}.txt`). This is particularly useful to check for **systematic wavelength calibration errors**, such as order-dependent drifts in the instrument's wavelength solution.

How It Works

1. **Setup and Input Selection**
   - The script takes a filename pattern such as `object_{order}.txt` where `{order}` is replaced with each order name (e.g. `m43`, `m44`, …).
   - You must specify the **instrumental setting** (`--setting`) and the **wavelength scale** (`--vac_air`) to ensure the correct PCA models are loaded.

2. **Loading Data**
   - For each order, it loads:
     - the **observed spectrum** (from your object file),
     - the **average telluric model** (`ave_{order}_{vac/air}.txt`),
     - and the **predefined telluric absorption ranges** (`tel_abs_{order}_{vac/air}.txt`).

3. **Defining the Analysis Region**
   - The regions of significant telluric absorption are used as `use_ranges` for fitting. This ensures the RV offset is measured **only from telluric features**, minimizing contamination from stellar lines.

4. **Measuring the Offset**
   - It calls `calc_sp_offset()` on each order, scanning across a velocity grid (set by `--vmin`, `--vmax`, `--vstep`) to find the best RV shift that minimizes residuals.
   - It also fits a flux offset and counts the number of usable pixels (`n_used`) used in the calculation.

5. **Output**
   - For each order, the script prints a summary line:

     ```
     o     v[km/s]    n_pix    filename
     m44   +1.352      583     object_m44.txt
     ```

     This gives a quick overview of the measured offsets across all orders.

Key Options

`pattern`	Filename pattern like `./object_{order}.txt`

`--vac_air`	Wavelength scale of the input (`vac` or `air`)
`--setting`	Instrumental setting label (default from `utils.default_setting`)
`--orders`	Comma-separated list to restrict the analysis to specific orders
`--vmin`, `--vmax`, `--vstep`	Velocity search range and step (km/s)
`--fmin`, `--fmax`	Flux range to include (to avoid noisy pixels)

Example Usage

```
python3 measure_telluric_offsets.py "object_{order}.txt" \
    --setting WINERED_WIDE --vac_air vac \
    --vmin -20 --vmax +20 --vstep 0.5
```

Example output:

```
# Measuring telluric offsets for 11 orders in WINERED_WIDE (vac)
# File pattern: object_{order}.txt
# o    v[km/s]  n_pix  file
m43    +0.842     601  object_m43.txt
m44    +1.356     583  object_m44.txt
m45    +1.192     598  object_m45.txt
...
```

This output quickly shows if your object spectra are consistently shifted (e.g. all orders show +1.3 km/s), or if there are order-to-order differences that may need correction.

When to Use

- Before running `fit_model.py`, to check if a global wavelength offset is present.
- To validate the accuracy of the spectrograph's wavelength solution.
- To assess if the wavelength calibration drifts between orders or nights.

Acknowledgements

I’m grateful to Hiroaki Sameshima for his suggestion to use the PCA approach for the WINERED telluric correction and following discussions. I also thank Daisuke Taniguchi for discussions on the algorithm and toolkit.