Supplement of Geosci. Model Dev., 14, 7287–7307, 2021
https://doi.org/10.5194/gmd-14-7287-2021-supplement

Geoscientific Model Development
Open Access
EGU

Supplement of

Cosmic-Ray neutron Sensor PYthon tool (crspy 1.2.1): an open-source tool for the processing of cosmic-ray neutron and soil moisture data

Daniel Power et al.

Correspondence to: Daniel Power (daniel.power@bristol.ac.uk)

How the code is structured and how it creates the outputs

We have tried to structure crspy in a modular way to organise the code, this can help when wanting to adjust certain areas without affecting other parts of the process. This supplementary document will explain how the code is structured and give some insight into how a user might update it themselves. This document relates to **crspy v1.2.0.** and can be used as a reference to understand the inner working of the code.

Main Routine

The main function that processes raw CRNS data into soil moisture estimates is contained in the **`full_process_wrapper.py`** module. Within this module is the function:

```
process_raw_data(filepath, calibrate=True, intentype=None, nld=nld)
```

where the filepath is a string pointing to the location of the raw CRNS data, calibrate is Boolean to state whether to use the calibration routine or not, intentype can be left as default or set to 'nearestGV' (see Section 2.1 of manuscript) and nld is the name of the configuration file (config.ini) where global parameters are stored. The option nld=nld is in any function that requires this config file and is called into the module at the beginning when also calling in the required packages using the ConfigParser.

This function "wraps" the steps to fully process the data. The steps are:

1. Tidy and format the data
2. Calculate the corrected neutron counting rates using the tidied data
3. (Optional) Use the corrected neutron counts with the calibration data to calibrate the sensor, acquiring the N0 number for the site.
4. Flagging and removing erroneous data (we need an N0 number to do this which is why it occurs here rather than at the very beginning of the process)
5. Quality Assessment plotting
6. Convert neutron counts into soil moisture values.

Tidy and Format Data – (**tidy_data.py**)

The first step involves formatting the data into a standard form. The function within this module is:

```
prepare_data(fileloc, intentype=None, nld=nld)
```

As before nld is required to load in the config.ini file and intentype can select another approach to correcting for neutron intensity. The fileloc is the string location of the raw CRNS data.

What follows is a run through of the steps in this function that are undertaken:

1. The country and sitenum are extracted from the file path
2. They are used to create a site_code (e.g. USA_SITE_011) that is later used to query ERA5-Land data.
3. The master time process is conducted. Here we are expanding the time series so that each hour is present in the final document (using a no value such as -999 for filling in missing data). This was done so that we can identify gaps in datasets. Additionally, we adjust the times on CRNS data so that values occur on the hour (i.e. at 10:00 instead of 10:15). This allows us to match up external datasets with the CRNS data. Currently this is done by flooring the values to the hour (see issue #1 on the GitHub page).
4. The next steps relate to ERA5-Land data (collecting using a separate process). This replaces missing sensors in some older sites with ERA5-Land data, allowing us to correct them for all currently understood influences on the signal. The function will query the user generated era5-land netcdf file, create dictionaries for the variables and if required append them into the formatted CRNS data.
5. We create a single pressure column (as sensors often have two available)
6. We create a column for vapour pressure, either as a function of dewpoint temperature (ERA5-Land) or as a function of relative humidity (in-situ data).
7. We query the NMDB.eu database to collect the reference neutron monitoring data and append to the dataframe.
8. We do some final formatting and tidying of the dataset and save it into the /data/crns_data/tidy/ folder using the country code.
9. The updated dataframe is output, which can then be fed into the next section of the process.

Correct the neutron counts - (**neutron_coeff_creation.py**)

The next function called is contained in the **neutron_coeff_creation.py** module. Here we are using the tidied and formatted dataframe and correcting the neutron signal for the known influences described in the paper.

The main function is:

```
neutcoeffs(df, country, sitenum, nmdbstation=None, nld=nld)
```

where df is the formatted dataframe, country and sitenum are the appropriate values for the site being processed, nmdbstation can be a string to select a new station (default is Jungfraujoch) and nld is the config files.

This processes the dataframe to give us the factors required correct raw N into corrected N. The neutron correction functions themselves are imported from the **neutron_correction_funcs.py** module. The output is the dataframe with the correction factors (e.g. fawv for atmospheric water vapour correction) and the corrected neutron count

(MOD_CORR) as well as the standard error of the neutron count rate (MOD_ERR). If someone wished to alter a correction method, or experiment with an additional one, changes can be made here.

N0 calibration – (**n0_calibration.py**)

If this option is turned on, it will now use the corrected neutron data to calibrate the sensor using the supplied calibration data. This process follows that outlined Schron et al., (2017). The following section will briefly outline this process that can be used along with the commented code to understand how it is being done.

The main function here is:

```
n0_calib(meta, country, sitenum, defineaccuracy, nld=nld)
```

where meta is the metadata table (output from the first tidy process), country and sitenum are also outputs from the tidy_data process that lets the code know which site to collect data for, defineaccuracy is usually set to 0.01 (this represents 1% which is the accuracy desired from the iteration that will occur in the script) and nld is the config.ini file.

Additional functions within this module are related to weighting procedures and calculations provided in the Schron et al., (2017) paper.

The steps of the code are then:

1. Necessary variables are read in from the metadata table using the country and sitenum and stored for use later.
2. Folders are created for saving of outputs
3. Tidying of data such as formatting of columns is undertaken
4. The count of unique calibration dates is stored (this will be used later). It can also be a single calibration date if only one is available.
5. The radial distance from the sensor (m) is extracted from the calibration data. This is related to the format of the COSMOS data which combines direction and radius together.
6. We also require the depth of each sample, this is sometimes provided as a range and so we adjust this to give us a single number for the depth. This is taken as the mid point of the range of depth given.
7. Using dictionaries, it will now split the calibration data into separate dataframes, one for each unique calibration date.
8. Next step is to read in the formatted CRNS dataset (stored in the data/crns_data/tidy/ folder) that was created in the tidy data step previously. We use the unique dates stored earlier for each calibration date to extract the time periods of calibration and place them in separate dataframes in a dictionary.

9. For each calibration date we use the data to get average values for the calibration period for: Temperature, Pressure, Relative Humidity (if available) and Vapour Pressure (if no relative humidity is available).
10. The next step is to weight the samples appropriately in order to provide us with a field averaged soil moisture value. This follows the steps outlined in the Schron et al., (2017) paper which gives much greater detail on this. Broadly the code does the following for each separate calibration day:
    a. Creates an equal average mean and defines accuracy as 1
    b. Creates a weighted average for each soil sample profile using appropriate weighting equations from the Schron paper
    c. Calculates the absolute humidity for the calibration period using the previously collected data
    d. Calculates the weighting to be applied to each sample to account for the radius from the sensor
    e. Saves dataframes created in this process for later viewing
    f. Checks to see if the difference between the newly field averaged value is within one percent of the previous field average
    g. Iterate until this converges within the accuracy value (0.01 or 1%)
11. Once we have the field average, we can find N0. We have all the variables required to calculated soil moisture except for the N0, we also now have the soil moisture value. We iterate create a list of N0 numbers from the average of N to the average of N *2 and create a list of soil moisture values. The one with the least error is taken as the correct N0.
12. If there are multiple calibration days we create lists of the error for the calibration days, sum the error for each N0 and the one with the least error (summed) is taken as N0.
13. Next is to output a user report with some key values that have been calculated as well as the error tables.

Flag and Remove data and QA graphs – (**qa.py**)

The next step is to conduct quality assessment. We now definitely have an N0 number and so can undertake assessments such as removing N values below 30% of N0. The first function here is:

```
flag_and_remove(df, N0, country, sitenum, nld=nld)
```

where df is the neutron corrected df, N0 is taken from the calibration routine or from the metadata table using the country and sitenum and nld is the config file.

The steps here are to apply the flagging of data as outlined in the manuscript.

We also generate QA graphs for visual inspection of site variables that could aid in identifying issues.

Generating soil moisture estimates – (**theta.py**)

The final stage of processing is to provide soil moisture estimates, give a rolling average as well as error bands and output a final version of the table with all these details in it. So far, the code has taken the raw data, formatted it and collected external data, created correction factors, calibrated the data, undertaken some quality assessment and we are now ready to generate the soil moisture estimates.

The main function here is:

```
thetaprocess(df, meta, country, sitenum, yearlysmfig=True, nld=nld)
```

Here the df is the formatted dataframe with the neturon corrections applied, the metadata table, country and sitenum of the site being processed, Boolean value to state whether to output yearly figures and the config file.

1. The first step is to query the metadata table using the country and sitenum to collect necessary variables for calculating theta
2. It then reads in the dataframe of CRNS data
3. It creates error columns (+-) that are used to understand the inherit uncertainty of the signal
4. It calculates the soil moisture using the thetacalc function
5. It calculates the soil moisture if you were to add or subtract the uncertainty in the neutron counting rate
6. It cuts off soil moisture at the soil moisture max
7. It uses the D86 calculations (from the N0 calibration) to provide a depth of measurement
8. It tidies the data and saves the final table
9. It produces graphical figures of the soil moisture

Changing the code

If someone wanted to test a new equation, then they can replace/insert the required variables at the appropriate area in the code. To use an example, recent advancements have led to a revision of the base theta calculation that derives soil moisture from neutron counting rates (Kohli et al., 2021).

To change this a user could replace the theta_calc() function in the **theta.py** module. Ensure that any changes to required inputs are fully accomodated within the thetaprocess() function. Additionally a new function could be created and within the thetaprocess() function this can be added as a line, so that crspy processes the data with both versions and outputs them into a final table for comparison.

If a user wished to change correction methods for the neutron signal then they can adjust the process in the **neutron_coeff_creation.py** module.

Once desired changes to the code has been created it can be installed by opening the folder with the code in it in a terminal, ensuring your development python environment is activated and running in the terminal:

```
pip install .
```