Straxferno: a Perilous Journey to the Heart of our Analysis Framework

Find materials in the course repository here

Introduction

This is an online course on strax, where we dive deeper and deeper into the details that make our analysis framework go. The level increases throughout the course.

To get the most out of this course, expect to spend 5-10 hours a week. Each week has a 1-1.5 hour discussion section, a few hours of reading, and exercises which will take several hours to complete. Everyone is welcome to participate, even if you only have time to attend the discussion section.

In detail, we have:

- **Discussion**: a 1 to 1.5 hour zoom session in which we discuss the material together. Bring questions!! This is NOT a lecture, Jelle is far too lazy to make ten hour-long presentations, and it would not help you learn well anyway.
    - We understand there are no timeslots that are convenient for everyone in the collaboration. The sessions will be recorded so you can rewatch them. Feel free to discuss in the strax chat too.
- **Reading**: usually code from strax(en), but sometimes technical documentation. For optimal results, go through this before the discussion session – but don't be too shy to attend discussion if you couldn't.
- **Study questions**: questions to guide the reading. We will discuss these in the discussion section.
- **Exercises**: You can do these before or after the discussion section to help advance your understanding. 'Official' solutions might become available after the discussion, but you are more than welcome to share your own solutions in the course repo!

Prerequisites

We assume you are already familiar with python, to the point that you know idioms such as dictionary comprehensions, `**kwargs`, and simple classes. If you're not, see the list of Python training and resource materials, from the High-Energy Physics Software Foundation.

Why is it called straxferno? What is all this 'circle' stuff about?

It is an allegory

Schedule

Session	Date	Intro slides	Recording
0: Using strax	Friday 15 May 14:30	Link	`/project2/lgrandi/strax_tutorial/0_using_strax.mp4`
1: High-level processing	Tuesday 19 May 15:00	Link	`/project2/lgrandi/strax_tutorial/1_high_level_processing.mp4`
2: Low-level processing	Tuesday 26 May 15:00	Link	`/project2/lgrandi/strax_tutorial/2_low_level_processing.mp4`
3: Loading data and mini-analyses	Tuesday 2 June 15:00		`/project2/lgrandi/strax_tutorial/3_data_loading_and_mini_analysis.mp4`
4: Contexts, run selection, configuration	Tuesday 9 June 15:00	link	`/project2/lgrandi/strax_tutorial/4_context_runs_configuration.mp4`
5: Storage	Tuesday 16 June 15:00	link	`/project2/lgrandi/strax_tutorial/5_storage_system.mp4`
6: Plugin implementation	Tuesday 23 June 15:00		`/project2/lgrandi/strax_tutorial/6_plugins.mp4`
7: Data flow	Tuesday 30 June 15:00		`/project2/lgrandi/strax_tutorial/7_dataflow.mp4`
8: Processing setup	Tuesday 7 July 15:00		`/project2/lgrandi/strax_tutorial/8_processing_setup.mp4`
9: Mailboxes	Tuesday 14 July 15:00	Link	`/project2/lgrandi/strax_tutorial/9_mailboxes.mp4`

Vestibule

Circle 0: Using strax

This is a preparatory session to go over the main points of *using* strax for anyone interested.

Reading:

- Analysis guide (if you have not read it already)
- Strax presentation (slides 1-11)
    - This presentation may also be useful
- Straxen tutorials: strax_demo, run_selection and pulse_analysis. Try to actually *run* the notebooks: they are automatically copied to the `strax_tutorials` folder in your home directory if you've followed the analysis guide.
- Straxen datastructure (skim)

Study questions:

- Why does your notebook crash if you try to load `raw_records` for a normal one-hour run? What options do you have available to do analysis on low-level data?
- Someone announces on gitter that the straxen processing algorithms have just been updated. Looking in your notebook, you see the data you want to load is not available anymore. What is the most likely cause? What options do you have for accessing the data anyway?

- Why can you not load `peak_basics` and `event_info` together in one `get_array` or `get_df` call? How else can you compute e.g. the sum area of S2s in an event?

Exercises:

- Copy the notebook `/home/aalbers/strax_training/elife_exercise.ipynb` to one of your own directories. Go through the exercises in there.

Outer circles

Circle 1: High-level processing

Reading:

- straxen/plugins/event_processing.py (406 lines)
- straxen/plugins/peak_processing.py (256 lines)
- Straxen datastructure (for reference)

Study questions:

- How exactly is a strax event defined? What determines what the main and alternate S1 and S2 are?
- Why does event_basics' compute function take only two arguments (beside self)? It depends on four other plugins!
- Strax applies four corrections – one to the position, and three to the S1 and S2 areas. Can you find where each of these are applied?
- What is the difference between a Plugin, a LoopPlugin, and an OverlapWindowPlugin? There is not much documentation on this, try to figure it out from the code. If it helps, peek at the base classes in strax/plugins.py (here and here).

Exercises:

- An analyst requests that we make a field n_hits available in peak_basics. Add the field in your custom straxen installation (locally or on dali/midway) and process XENON1T run 180215_1029 to test that it works. (For extra credit, also add the time at which the peak's sum waveform reaches its maximum.)
- Modify the event plugins so that the n_hits information is present for the main and alternate S1s and S2s. Verify it works again on run 180215_1029. Hint: there are many ways to do it, but it can be done by only adding one line.[1)]
- In a notebook, make a new plugin that reconstruct S2 positions as the position of the maximum PMT. (Hint: use straxen.pmt_positions.) Alternatively, reconstruct positions as the area-weighted mean of the hitpattern. (Hint: use np.average.) Reprocess run 180215_1029 and compare the resulting event positions. You can do this without changing the higher-level plugins that depend on peak_positions.

Circle 2: Low-level processing

Reading:

- straxen/plugins/peaklet_processing.py (342 lines)
    - Skip natural_breaks_threshold and get_merge_instructions, unless you are Joey/Evan/Joran;
    - It might be helpful to review the clustering/classification note first.
- PulseProcessing from straxen/plugins/pulse_processing.py (191 lines)
    - You can skip the functions defined below the class (e.g. software_he_veto, count_pulses, etc.) or not, depending on your interests.

Study questions:

- Consider the different steps in PulseProcessing. Are there any whose order could be (inter)changed?
- Why are we doing find_hits twice? Are the results guaranteed to be the same?
- Why is the peakfinding a three-step process? Why do we do a rough gapsize clustering before natural breaks?
- Is any information lost when merging peaks together?
- What is the purpose of integrate_lone_hits?

As a homework exercise, we're going to split up the low-level processing algorithms in strax between each other.

1. **Sign up** for one of the strax functions in this spreadsheet
    - Easy: O(10) lines. Choose this if you are new to strax, python or analysis.
    - Medium: O(50) lines. Probably a few hours of work, suitable for most people.
    - Hard: > 100 lines. If you know strax already and want a challenge.
2. **Study the function**, answering e.g. the following questions:
    - Start with the docstring (if there is one!) and maybe skim the code first
    - Find out where is it used. Look in strax and straxen. Compare with the docstring – what goes in, what goes out?
    - How is it tested? Check the strax unit tests. If it isn't tested, can you think of a way to test it?
    - Study through the implementation, paying special attention to comments. Anything unusual, risky, or remarkable?
        - If it helps, you could try to run the function in a notebook to try out some behaviors or modifications.
3. **Choose one of the following assignments**:
    - A: Add **one-slide** on the function to this presentation, with a short summary, e.g. your answers to the questions above, or other things you would like to highlight.
    - B: Make a **strax pull request** improving some part of the function. E.g. add some comments or a simple unit test.

Circle 3: Loading data and mini-analyses

Reading:

- plot_peak_classification from straxen/analyses/quick_checks.py (17 lines)
    - You can also look at other mini-analyses in the straxen/analyses folder, depending on your interests.
- apply_selection, make, get_df, and get_array from context.py (122 lines)
- straxen/mini_analysis.py (175 lines)

    - If you're not familiar with python decorators, you may want to look up a tutorial while diving into this. (There are many, use your search engine)
    - We also have documentation on mini-analyses here; apologies for the messy formatting.
- *Optional, for those interested in the DAQ: straxen/plugins/daqreader.py (360 lines).*
    - We may not discuss this in the discussion section; it will depend on time.

Study questions:

- Give an example of when time_selection = 'touching' would be appropriate in an analysis or processing task, and the same for time_selection = 'fully_contained'. Can you think of other time selection conventions that might be useful?
- Why does `make` have a for loop that just does pass? How could you rewrite this as a list comprehension? Do you think this is a good idea?
- What is the use of selection_str option to get_array? Couldn't the user cut more easily cut data in their own notebook after the data is loaded?[2)]
- What does straxen do if you make your mini-analysis take `t_reference` as an argument?
- Why does mini_analysis contain two nested function definitions? Could it not be done with one, or zero?

Exercises:

- Add a (hardcoded, simplified) blinding cut to strax: remove all events with cs1 < 50 (if cs1 is present in the data being loaded).
- Create a mini-analysis to plot the rate of peaks over time during a run. How would you plot the rate of just S2s?
- Modify mini_analysis so that mini-analyses can be defined with an argument `user_defined_data`, which is True if the user passed in a data manually, and False if strax loaded new data. Test it with your mini-analysis above. Make sure it still works for mini-analyses that don't have this argument.

Circle 4: Contexts, run selection, and configuration

Reading:

- **Contexts** (256 lines)
    - straxen/contexts.py: XENONnT part (first 100 lines). (Feel free to skim the rest.)
    - strax/context.py: initial list of options, `__init__`, and `register` (156 lines)
- **Run selection** (58 lines)
    - strax/context.py: get_meta, run_metadata, add_method (36 lines)
    - strax/run_selection.py: select_runs (22 lines)
        - You may want to start by reviewing the run selection demo tutorial.
- **Configuration** (124 lines)
    - You may want to start by reviewing the configuration changes section of the main tutorial.
    - There is also a useful configuration section in the strax docs
    - strax/context.py: _set_plugin_config (14 lines)
    - strax/config.py: Option class (110 lines)

Study questions:

- The DAQReader plugin not registered in every context. In which contexts is it omitted? Why?
- Suppose you are writing a plugin that takes an option. Why is it a bad idea to call your option

`threshold`? Can you already find examples of badly named options in straxen?
- What is the difference between regular options and context options? Give an example of each. How do you modify them?
- What is the difference between run metadata and ordinary (data-level) metadata? Give an example of info you can find in each.
- Suppose you implement a per-run deadtime-corrected livetime calculation. Where would be the best place to store this information?

Exercises:

- Select all runs for which lone_hits are available, that are shorter than 10 minutes, and do not have a 'bad' tag.
- Load xenon1t_dali, then create a new context from it which:
  - Puts lone_hits for all runs into a dedicated folder (e.g. /home/yourusername/my_lone_hits);
  - Allows building records, but not raw_records;
  - Uses a hitfinder threshold of 50 ADC counts.
  - Finally, process a tiny run (e.g. 180215_1029) with this. Compare the results with those of the default context.
- Modify `register` to check that the defaults of the to-be-registered plugin's options are consistent with defaults specified by plugins that are already registered, and throw an exception otherwise.
  - If you are the first to do this, please make a pull request! If you're not, look through the pull request made by whoever was the first and see if you can suggest improvements.

Circle 5: The storage system

Reading:

- Start with the documentation page on the storage system (apologies for the poor formatting)
- The entire storage system is too large to cover in one week. Below I listed the main functions of each the layers involved (most of them relatively small), which will at least give you an idea what is done where.
- `strax/contexts.py`: `is_stored`, `key_for`, `get_meta`, `_get_partial_loader_for`
  - These are the main places a context interacts with its storage frontends.
- `strax/storage/common.py`: DataKey, and from StorageFrontend: `__init__`, `loader`, `saver`, `get_metadata`; a quick glance at `find` and the abstract methods (`_scan_runs` and onwards).
- `strax/storage/files.py`:
  - from `DataDirectory`: `__init__` and `_find`;
  - from `FileSytemBackend`: `_read_chunk`;
  - from `FileSaver`: `__init__` and `_save_chunk`.
- `strax/io.py`: `load_file` and `_load_file`.
- Optional, for those interested: have a look at straxen/rundb.py to see a different StorageFrontEnd in action.

Study questions:

- When you ask for data, different messages are passed through different layers of strax. Fill in the missing entries in the flow diagram below.

    - `User –[run_id + data_type]» Context –[…]» Storage FrontEnd –[…]» … –[…]» reading code in io.py`
- Which part(s) of the storage system would you need to change to implement the following?[3)]
    - Support a new way of cataloguing which data is stored where (like the runs db or rucio)
    - Support a new file format for storing data (e.g. csvs or ROOT trees)
    - Support a different scheme of file organization, in which files for different runs are more or less randomly scattered across different folders, though there is some file or database tracking which file is where. (This is not hypothetical, unfortunately; it is what Rucio does to our files..)
- Someone gives you a path to a folder which contains amazing data you want to read, unfortunately processed with some custom settings they did not tell you (and maybe some custom plugins too). How would you go about reading it in?

Inner circles

Circle 6: Chunks and plugins

Reading:

- Documentation on the strax data model
- from `strax/chunk.py`: the `Chunk` class (243 lines)
- from `strax/plugins.py`:
    - `Plugin` class, except `iter` and `cleanup`. (320 lines, focus on the big picture)
        - You probably want to refer to some example plugins from straxen as you are reading this. See circle 1 and 2.
    - `OverlapWindowPlugin`, `LoopPlugin`, and `MergeOnlyPlugin`. (165 lines)
        - We covered the outlines of this (from a user perspective) in circle 1 and 2.
        - (Skip `ParallelSourcePlugin`. This is only used to parallelize online processing efficiently, and will make more sense to you in one of the final circles).

Circle 7: Data flow

- From `strax/plugin.py`: `iter` and `cleanup` from the `Plugin` class
- From `strax/context.py`: `get_iter`
    - Don't worry about GeneratorExit. If you are interested, see issue 252.
- From `strax/processor.py`, `iter` method of `Processor` class
    - Don't worry too much about the first few lines involving mailboxes; we'll get to them in the final circle

Study questions:

- What checks are in place to ensure plugins process all data from a run?
- What happens if data for one type stops (i.e. its last chunk has an endtime) a few milliseconds before another does?

Exercises:

- Add a progress bar to straxen, controlled by a context option. See issue #240
- Consider issue #247. What is causing the problem? Any suggestion for fixing it?

- Consider issue #222. Can you think of a way to tracking and summarizing computation time per plugin? (Wouldn't advise putting too much work into implementing it, Yossi is already looking at solutions at the streaming level that may make this easier.)

Circle 8: Processing initialization

- get_components and _get_plugins from context.py

Study questions (sorry for adding them late, we'll go through them during the discussion section if there are no more important questions).

- Please put the following significant events in the lifecycle of a plugin in order:
  - Call to __init__
  - Calls to do_compute
  - Setting complete configuration
  - Setting default for data_kind if it is absent
  - Calling infer_dtype if needed
  - Setting run_id attribute
  - Setting default for provides if it is absent
  - Call to setup
- Why is set_plugin_config called twice for each plugin?
- Find the following code goofs: an ususued variable, and two f-strings without an f. PRs welcome 

Circle 9: Mailboxes and the ThreadedMailboxProcessor

- __init_ from processor.py
- mailboxes.py, as long as you can stand its baleful gaze.
  - The descriptions here and here might be helpful

Study / discussion questions:

- Describe the three conditions in which a mailbox can be locked / initiate a waiting period. What triggers them? What resolves them?
- Does a mailbox have one or more senders? Does it have one or more subscribers? How do multi-output plugins fit in to this?
- Consider an output like lone_hits which is not required by other plugins. How is this mailbox emptied? What if lone_hits were also not saved?
- Suppose an exception occurs in a plugin's compute. Describe the steps that occur to handle this exception, until strax has shut down.

1)
… and incrementing a version number
2)
You might need to peek inside get_iter to answer this. Don't get lost in there though, there is a lot of stuff we won't discuss until much later
3)
Just to avoid all mistunderstanding, of course I'm not asking you to actually implement these as a study question!