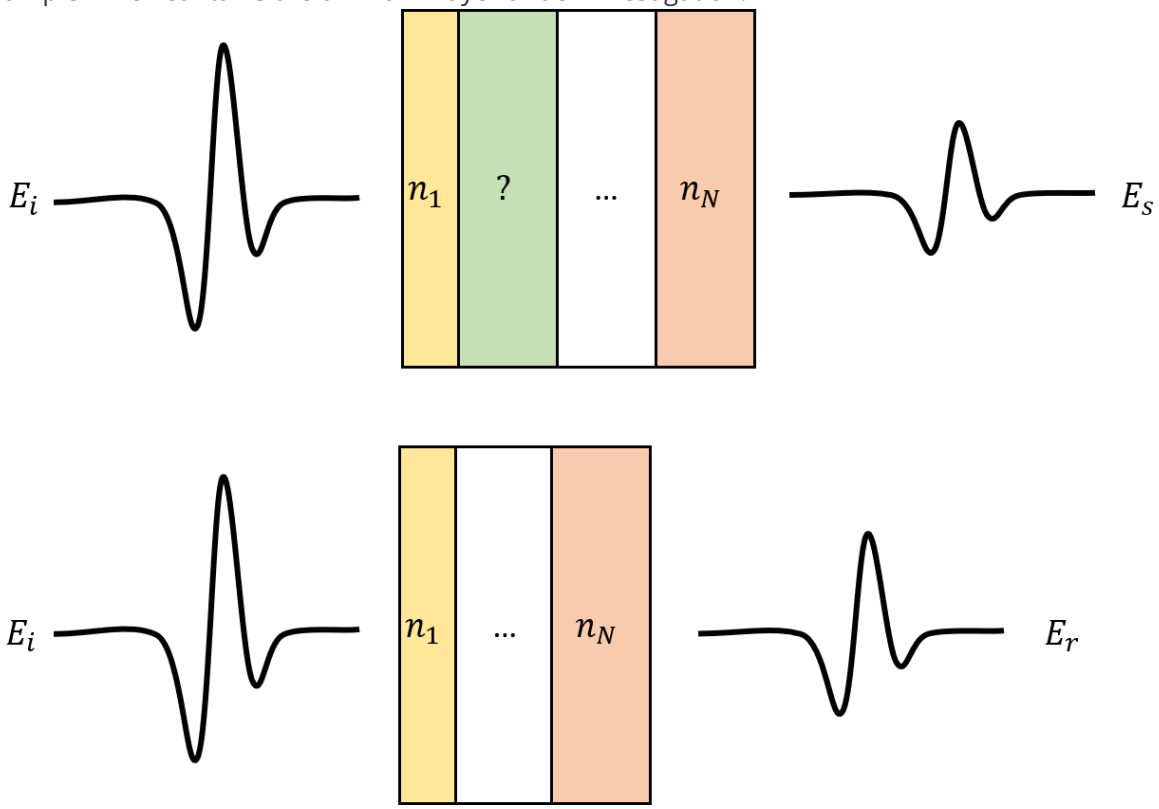
Overview

Nelly is a package for numerically extracting the refractive indices of materials from time-domain THz (TDS) data. These measurements typically consist of measuring a terahertz pulse after it's passed through a particular sample, and comparing this with a terahertz pulse that's passed through a known sample. The picture below depicts this general setup, with a THz pulse passing through a layered reference in which all the layers are well characterized, as well as through a sample which contains the unknown layer under investigation.



We can then Fourier transform the pulses and and see how the amplitude and phase of each Fourier component changes when passing through the sample (compared with the reference). We can express this as $\frac{\tilde{E}_s}{\tilde{E}_r}(\omega)$, the complex ratio of the sample and reference. This then allows us to extract the photoconductivity since the change in amplitude and phase can be related to each of the layer's **refractive index** and **thickness**. This can be expressed as follows

> $$\frac{\tilde{E}_s}{\tilde{E}_r}(\omega) = TF(\omega, n_{\text{solve}}, n_1, n_2, \dots)$$

where n_{solve} is the unknown refractive index and the transfer function $TF_s(\omega, n_{\text{solve}})$ is a function made up of Fresnel coefficients and propagation terms.[1] The complex ratio $\frac{\tilde{E}_s}{\tilde{E}_r}(\omega)$ is measured experimentally, so once we have the transfer function, we can go frequency-by-frequency and find the refractive index n_{solve} which best reproduces the experimental value.

With this in mind, we have the following tasks:

1. For a given geometry, construct the appropriate transfer function
2. Loop through a range of frequencies and fit the refractive index to the experimental value at that frequency.

This is the general procedure that Nelly follows when `nelly_main` is called.

Input File

Broadly, the input file has two parts: (1) settings, which controls various data processing parameters, and (2) sample specification, which gives information about the materials which make up the sample, and specifies the order they come in (i.e. the geometry).

The input file follows the JSON format, except that comments are allowed (beginning with `//`). A sample input file is included with the package (any of the JSON files in the `test_data` folder.

Settings

The settings part of the input file controls things like the frequency range and spacing, and parameters for the Fourier transforms. An example of this part of the transfer function is included below with comments to explain each line.

```
"settings":
{
    "a_cut": 6e-5,  // amplitude cutoff for reflections
                    // (see build_transfer_function_tree section below)

    // specifying the frequency range
    // the refractive index will be calculated at each
    // frequency between freq_lo and freq_hi with step size
    // freq_step
    "freq_lo": 0.2,
    "freq_step": 0.2,
    "freq_hi": 2.2,

    //specifying fft settings
    "fft" : {
        "windowing_type": "none",  //specifies the type of windowing
                                   //used to suppress noise in the
                                   //time domain traces.
                                   // options are:
                                   //   "gauss", "square", "none"
                                   // see TD_window.m for more info
                                   // Since the package is designed to handle
                                   // reflections and other features that may be
removed
                                   // in the windowing, windowing is
discouraged.

        "windowing_width": 2,      // the width of the window
                                   // for square windows, this is
                                   //    just the width.
                                   // for gauss windowing, this is
                                   //    the std dev

        "padding": 16,             // the base 2 log of the padding,
                                   // i.e. this pads the time
                                   // domain data to length 2^16
                                   // prior to Fourier transforming
    }
}
```

Geometry Specification

The next portion of the input file gives the geometries for the sample and reference. Each of these geometries consists of an array of layers, each containing fields for the name of the layer, the thickness of the layer (`d`), and the refractive index of the layer (`n`). There are a number of options for specifying the refractive index:

- `"solve"` (denoting the layer whose refractive index we're solving for)
- A number (including complex values)
- A path to a file containing the frequency-by-frequency refractive index. This must be in the csv format, which the first column denoting the frequency (in THz), the second column giving the real part, and the (optional) third column giving the imaginary part.

If the reference is not specified, it is assumed to be air (with the same thickness as the sample).

The example below shows the geometry specification for an experiment measuring the refractive index of water in a quartz cell with the empty cell as the reference.

```
"sample":
    [
    {"name": "air",      "d": 0,    "n": 1},
    {"name": "quartz",   "d": 1250, "n": 2},
    {"name": "water",    "d": 100,  "n": "solve"},
    {"name": "quartz",   "d": 1250, "n": 2},
    {"name": "air",      "d": 0,    "n": 1}
    ],

    "reference":
    [
    {"name": "air",      "d": 0,    "n": 1},
    {"name": "quartz",   "d": 1250, "n": 2},
    {"name": "air",      "d": 100,  "n": 1},
    {"name": "quartz",   "d": 1250, "n": 2},
    {"name": "air",      "d": 0,    "n": 1}
    ]

}
```

How to run

The main interface to the program is the `nelly_main` function, which is run as follows:

```
[freq, n_fit, freq_full, tf_full, tf_spec, tf_pred, func, spec_smp, spec_ref]...
    = nelly_main(input, t_smp, A_smp, t_ref, A_ref)
```

See below for a more detailed explanation of the input and output arguments

Functions

This section describes each of the functions in the package. For all functions, a description of the expected input is given. For important ("Primary") functions, a fuller description of the function is given as well.

Primary functions

1. `nelly_main` takes in an input file name and two time traces and returns a vector for the resulting calculated refracted refractive index, as well as the one for the corresponding frequencies. Conceptually, the code and be broken up into a few steps.
    1. **Loading and processing experimental data**
        1. Load settings and geometry from the input file.
        2. Process experimental data: pad, Fourier transform, and calculated experimental transfer function
    2. **Build transfer function** This is simply a matter of taking the geometries loaded from the input file and handing them off to the `build_transfer_function_tree` function.
    3. **Fitting** Loops through the frequencies specified in the input file and finds the refractive index n that best reproduces the experimental transfer function at that frequency. This optimization is done with MATLAB's `fminsearch` function from the Optimization Toolbox.

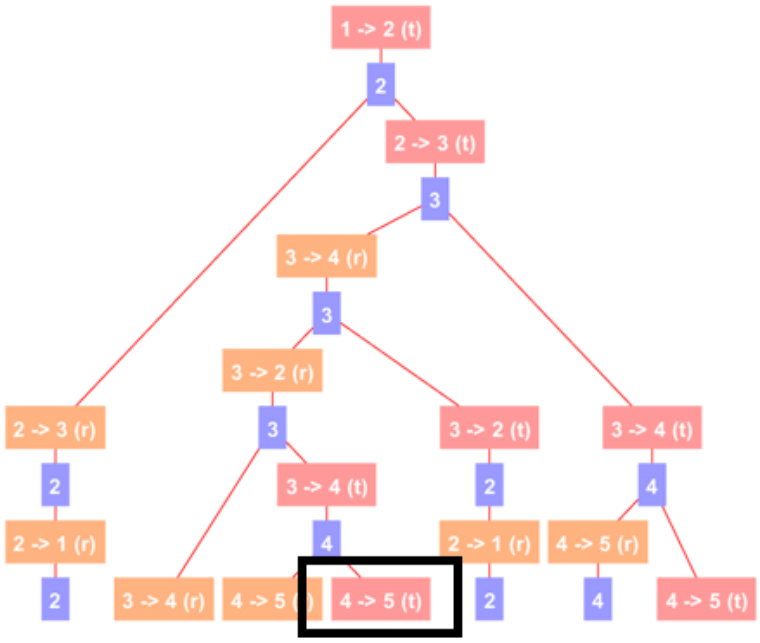
    Arguments

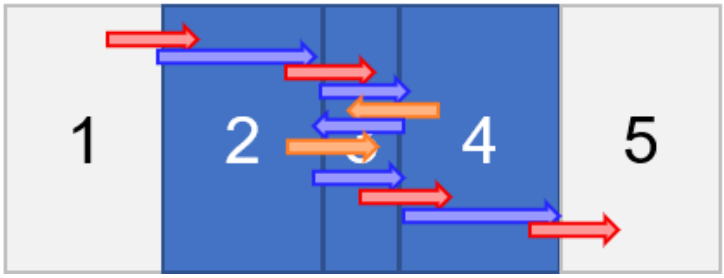
    - `input` : gives input geometry and other settings for the calculation. This can either be a filename for a JSON file or a struct containing the same information.
    - `t_smp` : time points for the reference time domain trace
    - `A_smp` : amplitude points corresponding to `t_smp`
    - `t_ref` : time points for the sample time domain trace
    - `A_ref` : amplitude points corresponding to `t_ref`

    Output

    - `freq`: an array containing the frequencies (THz) at which the refractive index was calculated
    - `n_fit`: an array of complex values for the refractive index. The i^{th} element corresponds to the i^{th} element in `freq`. For the imaginary part, positive values correspond to loss.
    - `freq_full`: an array containing a finer mesh of frequency points directly from the padded Fourier transform.
    - `tf_full`: an array containing the transfer function ($\frac{E_{smp}}{E_{ref}}$). The i^{th} element corresponds to the i^{th} element in `freq_full`
    - `tf_spec` : an array containing the transfer function ($\frac{E_{smp}}{E_{ref}}$) at a coarser spacing. The i^{th} element corresponds to the i^{th} element of `freq`
    - `tf_pred`: an array containing the predicted transfer function based on the extracted refractive index values, for use in assessing the accuracy of the extraction. The i^{th} element corresponds to the i^{th}element of `freq`
    - `func`: an anonymous function which takes two arguments -- a frequency (in THz) and the value for the unknown refractive index--and returns the predicted transfer function values at that frequency assuming that the unknown refractive index is the value given.
    - `spec_smp`: the spectrum for the sample pulse (i.e. $E_{smp}(\omega)$). The i^{th}element corresponds to the i^{th} element of `freq`.
    - `spec_ref` : the spectrum for the reference pulse (i.e. $E_{ref}(\omega)$). The element corresponds to the i^{th} element of `freq`.

2. `build_transfer_function_tree` takes in layer information (a structure) and returns the transfer function used for the refractive index extraction. Briefly, it does this by considering every possible path the pulse can take through the sample. At each interface, the pulse can either be reflected or transmitted. This is represented in a tree which splits at each interface. This is illustrated in the diagram below:





Here the tree starts at the interface between layer 1 (air) and layer 2. After passing through layer 2, the pulse will reach the `2->3` interface and either be reflected or transmitted, so the tree splits into `2->3 (t)` and `2->3(r)` branches. This process then continues for each of the branches. Each given node of the tree corresponds to a particular path in the geometry. For example the node in the black box corresponds to the path shown in the errors in the diagram below the tree (i.e. passing though layer 2, reflecting back and forth in layer 3 then transmitting through layer 4 and finally leaving the sample). Each of these paths will correspond to a particular time delay (the time required to traverse the path) as well as a change in amplitude (losses due to absorption and at interfaces). By keeping track of these, we can terminate any branch with a time delay that would place it outside our measurement window, or with an amplitude less than `a_cut` (specified in input).

The function `build_transfer_function_tree` takes a struct containing the geometry (`geom`) as well as time and amplitude cutoffs (`t_cut` and `a_cut`), and returns two functions:

- `tf_func`, which takes a frequency and a value for the unknown refractive index and returns the predicted $\frac{E_{smp}}{E_{ref}}(\omega)$

- `tree_func`, which takes a frequency and a value for the unknown refractive index and returns an object representing the tree of pulse paths. This can be used for debugging (e.g. visualizing the tree)

Auxiliary functions/classes

1. `estimate_n` gives the starting estimate for the refractive index. The real part is estimated from the time delay between the sample and reference pulses and the imaginary part is estimated from the attenuation of the sample pulse relative to the reference pulse.
2. `exp_tf` Pads, windows, and Fourier transforms the raw time traces.
3. `fft_func` takes two arguments which define the time domain trace -- `A` and `t` -- as well as the number of points to pad the FFT (`N`). It returns the frequency vector (in THz) and result of the
4. `faxis` gives the frequency points (in THz) for the Fourier transform.
5. `load_input` loads input file and returns a structure containing relevant data. Also checks input for errors, adds air terms to geometry structure.
6. `TD_window` windows the time domain trace to suppress etalons.
7. `time_pad` pads the time traces with zeros is the time ranges don't match.
8. `tf_node` is a class used for handling the tree nodes. The two types nodes (layer and interface) are handled in two classes that inherit from `tf_node`: `layer_node` and `interface_node` respectively.

Utilities

Several post processing and debugging utilities can be found in the `utilities` folder.

1. `drude_fit` Fits the given conductivity to the Drude model
2. `drude_smith_fit` Fits the given conductivity to the Drude-Smith model
3. `error_map` Takes a transfer function, the experimental transfer function, and ranges for the real and imaginary parts of the refractive index. Generates a error map showing the deviation between the experimental value and the transfer function prediction for each refractive index in the ranges given.
4. `error_map_single` [have to check this one and maybe get rid of it] Similar to `error_map` but only generates an error map at a single frequency.
5. `just_propagation` extracts the refractive index from the experimental transfer function assuming all changes in amplitude and phase are due to propagation through the unknown layer (i.e. no reflections)
6. `n_to_photocond` takes the refractive index of a photoexcited material along with its nonphotoexcited index and gives the photoconductivity.
7. `tinkham` extracts the conductivity from the experimental transfer function using the assumptions made by Tinkham and Glover (DOI: 10.1103/PhysRev.108.243).

Testing

This package has a suite of tests, which can be run in order to ensure the code is working properly. To run these tests, run the command `runtests`.

---

1. More details can be found in a forthcoming paper and this TDS tutorial by Neu et. al. ↩