SOP and tutorial for code used in:

“A universally applicable method of calculating confidence bands for ice nucleation spectra derived from droplet freezing experiments”

Prepared by William D. Fahy

Version 0.1

# 1 Purpose of this document

This SOP is intended to document the various functionalities and limitations of the Carnegie Mellon University ice nucleation statistics software package, and is provided to supplement the extensive comments within the code itself. This software is used to calculate, interpolate, and calculate statistics on ice nucleation (IN) spectra derived from droplet freezing experiments. The major innovations within this program are the use of continuous interpolations instead of binning of IN spectra and the use of empirical bootstrapping to calculate accurate confidence intervals and statistics for these spectra.

Section 2 is a step-by-step guide for how to set up Python and the various dependencies of this software on your own computer using the Anaconda software package. In Section 3, I will outline and discuss the existing functions available, their inputs and outputs, and use cases for each. For more traditional Python documentation for these functions, there is a markdown file accompanying this document along with example code to reproduce the figures from the associated publication (the markdown file is much easier to read for this section, but I wrote plaintext too for completeness). In Section 4, a tutorial is provided on how to reproduce the figures from the accompanied publication. There are also two tables provided: Table 1 outlines the variables contained within Freeze objects, and Table 2 summarizes the functions available in the software and their use cases.

This software is provided with two caveats. First, using this code as-is requires a basic knowledge of python, as it is run entirely from a python console. Second, while we have taken great care to ensure that the code included with this document is functional with minimal bugs, we cannot guarantee that it will work perfectly without some additional work from you. Always double-check the results you get to make sure they make sense.

# 2. Setting up Python

This program was developed and runs entirely in Python 3.7. If you have your own installation of Python, great! You will need pandas, numpy, scipy, matplotlib, and the basic built-in libraries installed to ensure the code will work. Note that if you use a console different from the one I developed it in (Spyder), the plotting functions might work quite differently for you than they do for me – you will have to work that out yourself.

If you don’t have an installation of Python, that’s okay! I recommend you install Anaconda. You can find the latest version for free here: <https://www.anaconda.com/products/individual>. Simply download and install this software and both Python and the required library and console will be installed for you. You should then be able to open the main python file, CMU-INstats.py, and run it out of the box with no additional modifications. If you have any issues, there is a huge open source userbase for both Python and Anaconda with well-documented support online that I encourage you to check out to get your installation up and running.

**3. CMU-INstats functionality**

3.1 Defining, initializing, and manipulating Freeze objects

The software itself is object-based, developed around the class Freeze. A Freeze represents one freezing assay and has a variety of attributed as defined in Table 1.

**Table 1:** Summary of attributes of the Freeze class

|  |  |  |  |
| --- | --- | --- | --- |
| Variable name | Variable type | Value | Defined when… |
| freeze | pd.DataFrame | The original data for the freeze itself | the freeze is initialized |
| name | dict | The name of the freeze | the freeze is initialized |
| m | float | The mass of ice nucleating material suspended | the freeze is initialized |
| mW | float | The mass of water the material is suspended in | the freeze is initialized |
| dV | float | Average droplet volume in mL | the freeze is initialized |
| suswt | float | Suspension weight% | the freeze is initialized |
| BET | float | The BETSSA for the sample | the freeze is initialized |
| nD | float | The number of droplets in the array | the freeze is initialized |
| bounds | list | The highest and lowest freezing temperatures observed | CalcINAS() is called |
| rawSpectra | pd.DataFrame | Raw IN spectra | CalcINAS() is called |
| interpSpectra | pd.DataFrame | Interpolated IN spectra | interpolateSpectra() is called |
| interpType | string | The interpolation method used | interpolateSpectra() is called |
| interpStats | pd.DataFrame | Interpolated statistics (including confidence intervals) across the IN spectrum | calcCI() is called |
| CI | string | The type of confidence intervals used | calcCI() is called |
| sims | pd.DataFrame | All interpolated and simulated spectra from bootstrapping | bootStats() is called |
| rawsims | pd.DataFrame | All raw simulated IN spectra from bootstrapping | bootStats() is called |
| resims | pd.DataFrame | All interpolated resimulated spectra used in calculating the studentized confidence intervals | bootStats() is called |
| att | set | A set containing searchable terms that describe the freeze | the freeze is initialized |
| path | string | The path that the freeze was loaded from or saved to | the freeze is loaded or first saved |
| bkgd | Freeze | The filtered water background experiment that applies to Freeze | the freeze is initialized |
| bkgdSub | Boolean | Whether bkgd has already been subtracted | the freeze is initialized |
| isDiff | Boolean | Whether the freeze is a difference (or ratio) | the freeze is initialized |
| isbkgd | Boolean | Whether the freeze is a filtered water background | the freeze is initialized |

Freeze objects should not be manually created – they should instead be initialized either by calling initFreeze() or by loading a freeze using the loadAll() or loadFreeze() command. Both take in a path for a directory or a .xslx/.csv file respectively (for loadAll this can be specified in a dialogue prompt) and return a dictionary of all freezes within the directory or the singular freeze itself respectively. The function loadAll can take in a dictionary to instead add the freezes to that dictionary, and both have the optional parameter ‘s’ which can be set to true if you wish to load the interpolated spectra and statistics from a file as well instead of having to recalculate them.

Freeze objects can be loaded only if they are saved in the correct format, usually by the program itself by calling Freeze.save(). If it is the first time this freeze has been saved (determined by whether the Freeze has a defined path attribute), or if saveas is set to True in the save function call, you can define the path to be saved to in a dialogue prompt. Then, if the Freeze has been interpolated, the save function will create an additional ‘spectra’ directory within the specified path directory with interpolated spectra saved (the normalization technique for ice active site density spectra is determined by defining the optional variable specType, with ‘ns’ being the default).

To initialize a freeze object that has not been previously saved, the function ‘initFreeze’ should be called, which will then prompt for a file path. This is the least generalizable part of the program, as it depends on the formats of the data files you are using, and may require you to change the code to fit your own needs. As written, the file specified should be an .xlsx file with the first excel sheet containing columns named ‘Frozen Fraction’, ‘Temperature, and ‘Radii’, containing the freezing temperatures, accompanying frozen fractions, and a list of droplet radii for your experiment (the only parameter the program currently uses is the length of this list to get the number of droplets frozen). You should check the section of this function that specifies the variable ‘dV’ to ensure that it sets dV to your appropriate droplet volume in mL, whether you have data to calculate that or you just set it to the droplet volume specified on your pipet. The program will then ask you a series of questions. If you don’t have an answer (e.g. you don’t have the BETSSA of your sample), simply enter ‘1’ to pass this question by. In this case your IN spectra might not have the correct normalization scheme, but you should then know which spectrum you’re trying to use and can avoid using the spectra you don’t have all the data to properly calculate. The function ‘initFreeze’ will then return a freeze object. Finally, you can change variables using the changeVar function, which will prompt you once called and allows you to in-program change the parameters of the freeze.

Finally, there is a way to combine freezes that are nominally identical (e.g. they are replicates of the same experiment). Call the function combineLikeFreezes(), which takes in freezes, a list of the freezes you want to combine, and the optional variable name, the desired name of the combined freeze as a string (you can specify this later in a prompt if you don’’t define it in the function call). This function will treat the freezing events of all of the freezes as one long freezing assay, and will output an object called an Exp (short for experiment). This is a subclass that inherits everything from the Freeze object, but also contains some data about which individual freezes are contained within it – otherwise it acts exactly the same. This function combines the attributes of all the freezes included in it and also calculates the IN spectra and interpolations of the IN spectra as a bonus. Experiments can be treated like a Freeze for any other function calls – the difference is mostly internal so that plotting and other specifics of the implementation actually work.

3.2 Using a freeze object to calculate statistics

Once you have a freeze object, you need to calculate the ice nucleation spectra, interpolate them, and bootstrap to calculate statistics and confidence intervals. To do this, you must call functions in the correct order. Each function is defined within the Freeze class and act as wrapper functions for functions outside of the Freeze class – this is just for convenience. You could manually call the base functions without using the Freeze class versions, but I don’t recommend it.

First, call calcINAS(), which will calculate the raw cumulative and differential spectra and initialize the rawSpectra variable. There are no inputs to this function. Cumulative spectra are calculated as normal, with each freezing event constituting a datapoint. Differential spectra are calculated without coarse binning by creating one bin for each temperature where at least one droplet freezes and then creating bins around those freezing events by calculating half the distance between each of the adjacent freezing events. By doing this, each freezing event has a datapoint that corresponds to the data in the cumulative spectrum. Depending on your interpolation technique, this may or may not be important for your final interpolated spectra.

Second, call interpolateSpectra(), which has one optional input ‘interp’, which must be a string and is set by default to ‘smoothedPCHIP’. The options for interp are currently ‘PCHIP’, ‘smoothedPCHIP’, ‘splinederiv’, ‘splineint’, ‘chebyshevint’, and ‘powerint’. PCHIP calculates a piecewise cubic Hermite interpolating polynomial, and is very similar to calculating a normal spline fit but with a monotonicity requirement. SmoothedPCHIP calculates a PCHIP interpolation and then re-interpolates using a simple cubic spline fit to incorporate a smoothing factor. To re-interpolate, a grid of temperatures Is calculated and the smoothedPCHIP interpolation is used to calculate corresponding ice active site densities. This grid of points is then used in the second round of interpolation. Splinederiv and splineint use simple spline fits, chebyshevint uses a Chebyshev polynomial fit, and powerint uses a power function. The PCHIP and splinederiv algorithms interpolate on the cumulative raw spectrum and then take the negative derivative to calculate the interpolations of the differential spectra, and the remaining functions interpolate on the differential spectra which is then inverted across the x-axis and integrated to produce the cumulative spectrum as a function of positive instead of negative temperatures (which can be dealt with later in the program while plotting). This approach of only interpolating one of the two types of spectrum was selected to minimize the number of interpolations that had to be done and to ensure that the cumulative and differential IN spectra agreed with each other. Note that the splineint function currently doesn’t work with studentized confidence intervals or for calculating differences between spectra, and this likely won’t be fixed because the splineint is quite inaccurate to begin with. The function interpolateSpectra returns the variable interpSpectra, which contains the interpolated functions which can later be used for plotting or generating data to export. The variable ‘interp’ is also initialized.

Third, call calcCI(), which has many inputs. The variable nSim (default 1000) determines how many bootstrapped simulations the function will calculate, and nresim determines how many second-order bootstrapped simulations the function will calculate if studentized CIs are desired. The alpha level specifies the confidence level you want (e.g. alpha=0.05, the default, will give you 95% confidence intervals). The method variable (default ‘Empirical’) requires a string specifying how the bootstrapped spectra will be calculated. Right now the only one implemented is the empirical technique so don’t change this! The variable moreStats determines whether you get quantiles as well as mean, standard deviation, and confidence intervals in the output. The variable CI should be set to a string describing how you want CIs to be calculated, and interp as above should be an interpolation technique used to interpolate the simulated spectra. The values for CI can be ‘pivot’, ‘quantile’, ‘tboot’, ‘tskew’, ‘expandedquantile’, or ‘studentized’. Pivot is a reverse quantile interval, quantile just uses the quantiles as a confidence interval, expandedquantile tries to correct the narrowness bias of the normal quantile intervals, tboot constructs a simple t-interval, tskew constructs a t-interval with a skew correction factor, and studentized constructs the bootstrap T intervals. See the accompanied publication for more information on these. calcCI actually acts as a wrapper for two functions, bootStats() and calcCIbody(). If you prefer, you can call Freeze.bootStats alone to only do the bootstrapping without calculating statistics, but there aren’t many situations where this is useful. CalcCI() initializes the variables ‘sims’, ‘rawsims’, and ‘interpStats’.

All three of these functions can be called using the wrapper function calc(), which takes in a single freeze and the variables CI, interp, and nSim and does everything for you. There is a little less flexibility when using this, but it’s faster and easier and is usually sufficient. This function also has my recommended defaults already input – the smoothedPCHIP with tskew intervals and nSim=1000.

**\*\*\*IMPORTANT\*\*\* -** calculating confidence intervals using bootstrapping takes *time*, especially if using studentized confidence intervals. I have a relatively powerful laptop, and it takes between 30 minutes to 1 hour for studentized confidence intervals on an array with ~300 droplets. On more powerful computers it is much faster, but I recommend calculating studentized intervals only on a few spectra to ensure that your other selection of intervals (tskew or expandedquantile) are accurate.

3.3 Plotting ice nucleation spectra

This program has built-in plotting capability using the Python library matplotlib (mpl). The main function you will need is makePlot(), which takes in the required variable ‘freezes’, which must a list of the freezes you want to include in the plot. Then, what should be set to a string describing which spectra you want to plot (‘ns’, ‘diffns’, ‘nm’, ‘diffnm’, ‘k’, or ‘K’), where should be a matplotlib frame if there’s an existing plot where you want to add the spectra to, ‘raw’ determines whether you’re plotting the raw or interpolated data, ‘CI’ determines whether confidence intervals are plotted, ‘mean’ determines whether you plot the bootstrapped mean or the true interpolated measured data, ‘ scale’ can be ‘auto’ or a list containing ymin, ymax, xmin, xmax in that order, ‘alsoraw’ determines whether you plot both raw data and the interpolated data or not, and ‘legend’ can be True (mpl default), False, or ‘external’, in which case a legend is superimposed on top of the plot to be later cropped out and used in figures outside of the plot itself. PlotSpectrum is the function makePlot calls, as it is applied only to a single spectrum. It has mostly the same inputs, but the variable ‘c’ can be defined to determine what color that spectrum should be.

To define the color, label, and line style of spectra, the functions getLine(), getMarker(), getColor(), and getLabel() are defined. Each takes in a freeze and returns a line style, marker style, color, and label respectively. I use if/else statements that check the names and attributes of the freeze objects to choose the styles for a given freeze, but you can do whatever you want here. There is also a getRandomColor() function that provides a random color from a colorblindness-friendly palette and a adjustLightness() function that can take in a color and adjust its lightness to make plots with ranges of color strengths.

Finally, there is a function called plotBoot() in Freeze object that allows you to visualize the bootstrapped simulations around a spectrum. This creates a plot with the 2.5 and 97.5 quantiles (as dashed lines), each bootstrapped freeze (as faint lines), and the original spectrum (as a bold line) plotted in one place. This function takes in ‘where’, a mpl frame (if you want to specify it, otherwise it creates one), ‘what’, and ‘nSim’, which are exactly what you would expect from previous functions.

3.4 Calculating differences between spectra and background subtraction

There are two main functions that are useful for calculating the differences between spectra. The first is for background subtraction. Once you’ve specified the freeze that represents the filtered water background freezing array applicable to a given experiment (by defining freeze.bkgd), you can call the function bkgdSubtract() (a function attribute of the Freeze class). This will automatically perform background subtraction, including recalculating statistics and confidence intervals.

The other useful function is compareFreezes(), which is not an attribute of the Freeze class. Instead, it asks that you input two positional variables ‘samp’ and ‘ref’ which are the two freeze objects you want to compare. The second freeze will always be subtracted/divided from the first. You can also specify nSim, the bootstrapping method (‘method’; if bootstrapping hasn’t already been done), the type of confidence intervals you want to use (‘CI’), and how to interpolate the difference spectra that are calculated (‘interp’). Finally, the variable ‘diff’ can be either ‘divide’ or ‘subtract’, and just tells you whether it is dividing or subtracting the two spectra.

In either of these cases, if calcCI() or bootStats() have not already been called on both the background and sample spectra, it will prompt you to calculate those when this function is called. Also, both of these functions are wrapper functions for the function ‘bootstrapFreezeDiff’, which you can also call separately if you really want to or look into to see the implementation of how this works. These functions return the objects diffExp, a subclass of Exp, which is identical to an Exp but with a little extra internal information again. The one notable thing is that if you plot a diffExp, a line will show up representing 0 or 1 (the no difference mark for subtraction and division respectively), just to help you compare. As before, diffExp objects have confidence bands, statistics, and interpolations, so use them as normal! You could theoretically calculate a difference of diffExp objects if you really wanted.

Finally, for convenience there is a function called ‘permComp’, which takes a list of freezes and calculates all permutations of the difference between each freeze and the combination of all other freezes. This is especially useful for identifying outliers.

That’s all I have for now, although the code will likely be updated to include some other features. Keep an eye on the github page (<https://github.com/wdfahy/CMU-INstats>) for updates. To conclude this section, Table 2 summarizes the functions I think are useful to someone who wants to use this code for their own research.

**Table 2:** Summary of user-facing functions in the CMU-INstats.py file. See the text for more details about inputs and outputs of each function.

|  |  |  |
| --- | --- | --- |
| Function name | Inputs & default values | Purpose |
| Freeze.save | saveas = False, specType = ‘ns’ | Saving a freeze object to a .csv file |
| loadAll | p = None, s = False | Load all freezes contained within a given directory and all sub-directories. |
| loadFreeze | p = None, s = False | Loading one freeze at path p (must be defined) |
| initFreeze | p = None, freezes = None | Initializing a freeze in its raw .xlsx data format |
| combineLikeFreezes | freezes, name=None | Combines all freezing arrays in freezes into one Exp |
| Freeze.changeVar | N/A | Convenience function to change an attribute of a freeze |
| Freeze.calcINAS | N/A | Calculates raw IN spectra |
| Freeze.interpolateSpectra | interp=’smoothedPCHIP’ | Interpolates raw IN spectra |
| Freeze.calcCI | nSim=1000, alpha = 0.05, method = ‘Empirical, moreStats = False, CI = ‘tskew’, interp = ‘smoothedPCHIP, nresim = 50 | Calculates confidence intervals and statistics for freeze. |
| Freeze.plotBoot | where = None, what = ‘ns’, nSim = 100 | Plots bootstrapped simulations around a spectrum |
| makePlot | freezes, what = None, where = None, raw = False, CI = False, mean = False, scale = ‘auto’, alsoraw = False, legend = True | Makes a plot with all freezes in the list ‘freezes’ |
| plotSpectrum | freeze, what = ‘ns’, raw = False, where = None, c = None, CI = False, mean = False, alsoraw = False | Plots one spectrum |
| getLine | freeze | Gets line for a freeze – edit to change |
| getMarker | freeze | Gets markers for a freeze – edit to change |
| getColor | freeze | Gets colors for a freeze – edit to change |
| getLabel | freeze | Gets labels for a freeze – edit to change |
| Freeze.bkgdSubtract | N/A | Subtracts the background spectrum from the sample (must have defined Freeze.bkgd). |
| compareFreezes | samp, ref, nSim = None, method = ‘empirical’, CI = ‘tskew’, diff = ‘divide’, interp = None | Divides or subtracts two freezes to compare them statistically – gives confidence intervals for the difference for statistical testing |
| permComp | lst, CI = ‘tskew’, nSim = 1000 | Calculates all permutations of differences of each freeze in lst from all the others combined – for checking for outliers |
| calc | freeze, Ci = ‘tskew’, interp = ‘smoothedPCHIP’, nSim = 1000 | Convenience function that handles all basic calculations for a single freeze at once. |

# 4 Recreating the figures from the accompanied paper

# In this section, the approach for generating each figure in the accompanied paper is outlined in plain text. The exact function calls and aesthetic choices used in the paper are not included, but these can be found in the accompanying script file ‘figure\_generator.py’, which will run automatically once the path to the appropriate downloaded data files is provided.

Figure 1

For this figure, load all the files from the ‘sep’ folder. Generate the combined water aged and unaged spectra by combining Freezes with the attribute ‘OG’ and ‘PW’ separately, then run calcINAS for each separate and combined spectra. Plot all of these freezes with ‘raw=True’ in the makePlot call.

Figure 2a, b

Load five versions of the combined water aged spectrum, and call calcINAS on each. Then, call interpolateSpectra on the first four, with interp=’splinederiv’, ‘splineint’, ‘PCHIP’, and ‘smoothedPCHIP’. The fifth is not implemented in the version of the code available, but can be reproduced using the methods described in the text. Plot each spectrum, saving the matplotlib frame, then call plotSpectrum to plot the raw spectrum on top of the rest.

Figure 2c, d

Load the combined aged and unaged spectra, then call calc() on each. Plot both using makePlot with alsoraw = True.

Figure 3

Load the combined spectra, then call plotBoot on one, saving the matplotlib frame. Call plotBoot on the other, inputting the same frame.

Figure 4

Load six versions of the combined unaged spectra. Call calc on each one with CI = ‘pivot’, ‘tboot’, ‘expandedquantile’, ‘quantile’, ‘tskew’, and ‘studentized’ respectively. Plot all freezes with makePlot, ensuing CI=True.

Figure 5

Load the combined aged and unaged spectra, and call calc on each. Then, call compareSpectra with the aged spectrum as the first freeze and the unaged spectrum as the second. For the first graph, call diff = ‘divide’, and for the second diff = ‘subtract’, being sure to save the output of compareSpectra to a variable each time. Plot each output spectrum.

Figure 6

Load the separate water aged experiments, then call permComp on a list containing all of the separate water aged experiments, setting the output to a variable. Call makePlot inputting the output of permComp.

Figure S1

Repeat the process for Figure 1, but call interpolateSpectra on each before plotting it, with raw=False.

Figure S2

Repeat the process for Figure 4, but with the unaged combined freeze.

Figure S3a

Load the separate aged freeze experiments, and create combined freezes with just the first, the first two, the first four, and all six spectra. Call calc() for all of these combined spectra and plot them.

Figure S3b

Load three copies of the combined aged experiment, and call calc() with nSim = 100, 1000, and 10000. Plot the three copies.