University of Nevada, Reno

peak.gas: An R package for data wrangling and plotting trace gas concentrations from instantaneous output produced by benchtop instruments

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Natural Resources and Environmental Science

by

Jacob Anderson

Dr. Benjamin W. Sullivan/Thesis Advisor

December 2021

THE GRADUATE SCHOOL

We recommend that the thesis
prepared under our supervision by

JACOB ANDERSON

entitled

peak.gas: An R package for data wrangling and plotting trace gas concentrations from instantaneous output produced by benchtop instruments

be accepted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

Benjamin Sullivan, Ph.D.
Advisor

Erin Hanan, Ph.D.
Committee Member

Elizabeth G. Pringle, Ph.D.
Graduate School Representative

David W. Zeh, Ph.D., Dean
Graduate School

December, 2021

Abstract

1. Some analytical scientific instruments, such as infrared gas analyzers (IRGA), elemental analyzers (EA), gas chromatographs (GC), and flow injection analyzers, provide instantaneous output of timeseries data but may require further processing by the user to estimate injected standard or sample concentrations. Such processing can be time-consuming and prone to error. Here, we developed an open-source package (peak.gas) that integrates time-series data from an IRGA, used in benchtop mode (injecting discreet samples into a carrier gas flow network), into peak areas from which concentrations can be calculated.
2. The peak.gas package was written in the open-source language R. The package is designed for users with varying degrees of familiarity with the R programming language. At its simplest, the package will produce output simply by setting a working directory and executing the function. The package can also easily plot instrument output for diagnostic purposes. There are warnings if values exceed set limits embedded in our functions that will alert the user to issues such as variations in standard performance when compared with check standards, files not properly formatted for processing, or standard curves not provided.
3. When the peak.gas package is used with the associated protocol (Appendix 1) describing benchtop use of the LI-8100A IRGA, the functions in the package will batch process a folder of text files containing sample names, date and time of recording, and carbon dioxide (CO_2) concentration. The functions identify and extract analytical peaks and calculate standard curves to convert peak areas into accurate CO_2 concentrations calculated by using area under the curve (AUC). The user can view any output using the plotting functions included within the package.

4. While the peak.gas package is designed to work seamlessly with the described protocol and instrumentation, it can be adapted by the user to different analytical instruments that produce similar output to the IRGA used here (sample, datetime, concentration recorded) regardless of compounds measured.

Acknowledgements

I would like to thank Dr. Paul Verburg for generously sharing his lab and equipment to generate data, and Dr. Elizabeth Huenupi for a previous contribution to an earlier version of the operating protocol presented in my thesis. I would also like to thank my advisor Dr. Benjamin Sullivan for his steadfast commitment towards enriching my scientific knowledge and practice. He has been adaptable, kind, and considerate throughout my degree even when faced with the COVID-19 pandemic and has genuinely wanted to help me and my lab mates become the best scientists we can be. I would also like to thank my lab mates for their brilliant contributions and companionship throughout my progress and the creation of my thesis, I am honored to be a part of such a wonderful group of people, thank you all!

Table of Contents

List of Tables

List of Figures

Preface

This thesis is formatted for submission to *Methods in Ecology and Evolution* under Applications and Practical Tools. When submitted, the research will include coauthors so I have used the pronouns “we” and “our,” instead of “I” and “my”.

Introduction

Many scientific disciplines measure concentrations of gas or liquid analytes on specialized instrumentation, including atmospheric chemistry, biology and ecology, engineering, food and agriculture, and occupational health and safety. Despite measuring gas concentrations for myriad purposes, many analytical methods detect a signal associated with the compound of interest, such as the absorption of an infra-red light in an infra-red gas analyzer (IRGA) used to measure carbon dioxide (CO_2) gas. When a discrete sample is injected into an instrument, the concentration of the analyte in the sample is equivalent to a peak created relative to a baseline measured when no analyte is detected. In many cases, proprietary software packages available from the instrument manufacturer will integrate detected peak areas. However, in other situations, like when an instrument is used in an application beyond the specifications associated with the software, analytical peaks must be integrated from instantaneous output (Kuang et al., 2009, Sherrod et al., 2012, Joos et al., 2008).

The LI-COR 8100 (Lincoln, NE, USA) is designed to measure CO_2 fluxes in a chamber over time by calculating the change in CO_2 concentration (detected by an IRGA). Proprietary software (LI-8100A, v. 4.0.0) supports this analysis. However, when using the LI-COR LI-8100 in a benchtop application (in which a discreet sample is injected into a CO_2-free carrier-gas flow network), to measure CO_2 concentrations of the sample, the proprietary software associated with the instrument (LI -8100A, v. 4.0.0) records carbon dioxide (CO_2) concentrations at one-second intervals and allows the user to export a raw .TXT file of the instantaneous values. However, these instantaneous values are not integrated. Therefore, there is no single value associated with a given peak. Though peak areas can be integrated manually, for efficiency and to reduce mistakes, a pipeline to process the data output would be optimal.

Here, we introduce the peak.gas (package name and all function names are case-sensitive) data analysis package designed to measure CO_2 concentrations of samples and standards injected on a LI-COR 8100 in benchtop mode. We used R, an open-source platform for statistical computing and data visualization (R Core Team 2021). Our R package can manipulate and process datasets in a fraction of the time spent calculating and visualizing CO_2 concentrations by hand, in a way that limits human induced error.

While the peak.gas package was designed to handle CO_2 analysis, the code is open-source and can be modified to measure concentrations of any molecule using any instrument that analyzes discrete sample peaks along a baseline during analysis. The package automates the post-data-collection processing by allowing the user to run the primary functions with a single command line. The package is available to users at (https://github.com/andersonjake1988/peak.gas).

Methods

The peak.gas package contains four functions (Table 1) and was built using R version 4.1.0 (R Core Team, 2021) and created using the tidyverse package (Wickham et al., 2019). peak.gas is designed to allow the user to quickly calculate peak height, area under the curve (AUC) for each peak, and concentration values (in parts per million) based on standard curves of known CO_2 concentrations, when the standard curves are annotated in a manner consistent with the protocol we developed for the LI-COR LI-8100 used in benchtop mode (Appendix 1).

Inputs

Files input into peak.gas processing functions (extract.peaks and timeseries.peaks (Table 1)) should be .TXT files that are structured into 3 columns. The first column should be the test column which counts the number of recorded measurements starting from zero and spanning the length of the text file. The sample names will be included in this column, as they are

automatically spliced into it when the user inputs the annotation while using the LI-8100A software. The second column should be the date and time in second intervals that the concentration value was recorded by the instrument. The third column should be the measured concentration (units are pre-selected by user when running the instrument) of the molecule of interest, recorded by the instrument. (Fig. 1)

Outputs

The output of the extract.peaks function (Table 2) takes this three column text file input and expands upon the information to return: file name, sample name, replicate peaks within that file and sample, the order the sample was run in the file, the area under the curve of each peak (AUC), the maximum peak height (recorded by the instrument) in each peak, the time the peak began, the time the peak ended, the duration of the peak, and the corrected concentration in ppm based on the AUC of the peak.

The timeseries.peaks function (Table 2) takes the three column input and outputs 4 columns: the file name the sample belongs to, the sample name associated with the instrument recording, the date and time of the instrument recording, and the value recorded by the instrument in units set by the user when operating the instrument.

Package development

After using the LI-8100, the proprietary software exports a .TXT file with the results. These files, when imported into R, are structured into three columns. The first column contains a continuous test number starting at zero and spanning the length of the file (this column also includes the user's sample annotations input into the LI-COR during sample processing). The second column contains the continuous date and time (in second intervals) spanning the length of the .TXT file. The third column contains the value recorded, which depending on instrument setup may be raw values associated with absorbance of infra-red light or instantaneous CO_2

concentration (Fig 1). The extract.peaks() function takes these data and extracts the sample names from the annotations in the continuous integer column into a new column, assigning each sample a number to preserve the order in which they were run in the case of duplicate sample injections (Fig 2). The function then isolates each of the sample peaks by cleaving the data at a user-specified cut off point (the cut.off = argument in the function call, (Table 1)), and assigning each peak in each sample a replicate number (Fig 3). The cut-off point sets a new baseline and should be specified as a point above instrument noise which can be investigated beforehand using the timeseries.peaks and Plot.timeseries functions to visualize any drifting baseline concerns or outlying data points. The AUC for each individual peak (Fig 4) is calculated using the formula (Eq. 1):

$$Area = \sum \left(\left(X_n * (T_{n+1} - T_n) \right) + \frac{(X_{n+1} - X_n) * (T_{n+1} - T_n)}{2} \right) \qquad \text{(Eq. 1)}$$

where X is the CO_2 value recorded minus the user-specified cutoff value and T is the time of recording. This area calculation operates on the premise that the values recorded represent a single point in time, but capturing the accurate area under the curve requires inclusion of the subfractions of time. To do this we take the area of the column of time, using change in time as the X component and the concentration recorded as the Y component. Then because the concentration values are either increasing or decreasing as we move along the X axis, the area of a right triangle is added to the area of the column when the value of $Y_n < Y_{n+1}$ and subtracted when the value of $Y_n > Y_{n+1}$.

This package uses a calibration curve that is user-generated from injected standards of known concentrations to transform the LI-COR data to concentration values (in parts per million (ppm)). Typically, standards to be used for curve analysis are injected near the beginning of

analysis. Best practices suggest injecting standards after every 10-15 samples to "check" that the instrument is performing in the same manner as when the "curve" standards were injected. Our functions require that the user input the key words "curve" and "check" with the standard name to distinguish which standards will be used in the standard curve calibration or used as check standards. Similarly, the function requires that the known ppm concentration of the standards should be indicated in the name (Appendix 1). For example, for standards known to be 300 ppm CO_2, the names should appear similar to "300curve" or "300check." The names associated with samples of unknown concentration do not have to follow a specific convention.

We equated the AUC for each standard peak to the known standard concentration and interpolated unknown samples based on either linear or logarithmic equations. To do this, all standards containing the word "curve" are extracted from the main dataset and the concentration values are further extracted from the name (Fig 5). If the standards used, range across multiple orders of magnitude, a linear relationship may become less accurate than a logarithmic equation, as the relationship between peak area and known concentration may begin to form a slight upward parabolic trend (Fig. 6, Fig. 7). To use a logarithmic (base-10) equation, the user can specify method = "log" in the function call. The standards to be used to calculate the standard curve in each file (or $\log_{10}$-transformed standards, if that option is selected) are linearly regressed and the slope, y intercept, and adjusted r^2 value are stored as a new object (Fig 8). The function then checks all files to make sure they contain a standards to create a standard curve (Fig 9) (samples of two or more concentrations containing the key word "curve"). When batch-processing multiple .TXT files from the instrument, sometimes an individual file does not include a standard curve. If a file doesn't contain a standard curve (Fig 10), the most recent standard curve generated within the batch file will be used (Fig 11). In the case that no standard curves were run, the function will generate a warning message indicating that no standard curves were

found and thus concentration data could not be calculated. Once all standards have a slope formula associated with them, the AUC values of samples of unknown concentration are transformed to ppm values using linear (Eq. 2) or logarithmic (Eq. 3) formulas:

$$AUC(ppm) = \frac{AUC - b}{m} \quad \text{(Eq. 2)}$$

$$AUC(ppm) = e^{\frac{log(AUC) - b}{m}}. \quad \text{(Eq. 3)}$$

Summary standard curve statistics including mean, standard deviation, coefficient of variance, linear equation statistics, and mean ppm (post interpolation). These will be output for the user if desired by setting the argument standard.sum = TRUE (Table 1) in the function call (Fig 12).

Discussion

As technology advances, scientists may obtain more data faster than ever before. This has led to an exponentially increasing volume of data with time. The ability to generate large amounts of data has given rise to the necessity of faster data processing. By using the peak.gas package and the associated protocol, the user will be able to consolidate, summarize, and visualize data efficiently. While the code is designed to work smoothly with the LI-8100 IRGA instrument and our protocol (Appendix 1), there are many analytical instruments available that, at their simplest, provide instantaneous output of concentrations or detector signals associated with an analyte, such as elemental analyzers, gas chromatographs, and flow injection analyzers. With slight modification of the open-source code, a user can apply this package to other instruments and analytes.

One strength of our package is the ability to fit, assess, and automatically apply a standard curve to generate ppm data of unknown samples. However, due to the automated

process, these standard curves should be investigated by the researcher to ensure proper fit of the standard curve. To assist in the process, we have included two visualization functions in the peak.gas package. The function Plot.extracted() (Table 1), is useful when determining sample patterns post-standard curve calibration, whereas the function Plot.timeseries() (Table 1) is useful in finding outliers and potential sampling errors before the standard curve is applied to the unknown samples. In addition to visualizing the data using the Plot.extracted() and Plot.timeseries() functions (Table 1), the user may set the "standard.sum" argument to TRUE in the extract.peaks function. In this case, the extract.peaks function will generate a summary report of the standard curve statistics, and the average concentrations after the curve calibration is applied to assist the user in determining the quality of the standard curve.

Analytical instruments provide instantaneous output even when analyte is not detected, forming a "baseline" from which peaks diverge. In most cases, baselines fluctuate slightly, and deviations from baseline can lead to the detection of false peaks not associated with a sample. To eliminate false peaks, we created a feature within the extract.peaks() function that allows the user to customize their analysis by setting a user-specified cutoff point in the function call (Table 1). While the default is set at 2, the user can redefine this to any value to increase or decrease the baseline. This feature allows the user to set a baseline from which the peaks are integrated, effectively trimming noise in the baseline unrelated to sample injections that could create false peaks. Because the AUC of a sample is equated to an AUC of known standard concentrations that should bracket the sample concentration, we can assume that the calibration will remain consistent even with higher cutoff values than 2, so long as the low-concentration standard AUCs are still detected. When injecting high concentrations, a higher cutoff point may be preferred relative to when injecting lower concentrations into the instrument. Lower cutoff values will increase the risk of false peaks. It is up to the researcher to verify that the samples listed in the

output match the samples processed. If there are more samples than expected, the selected cutoff value may have been too low. If there are less samples than expected, the selected cutoff value may have been too high.

In our experience, the baseline of the LI-8100 does not fluctuate substantially even during a multi-hour period of sample injections, but numerous factors could cause baselines to change during analysis on this or other instruments. Check standards values that differ from values associated with the standard curve can indicate baseline drift, among other potential problems. While check standards are recommended in our protocol (Appendix 1), our software also provides warnings to the user if one or more check standards differ from the confidence interval associated with the standard curve (or a user defined interval).

peak.gas provides more accurate estimation of concentrations by using area under the curve to calculate standard curves. This method produces values with less variability, thus leading to more accurate standard curves than simply summing the measurements recorded during the peak range or using the maximum peak height value. Summing the measurements recorded during the peak range, while still accurate, doesn't account for the true area under the peak, it only represents slices of the whole area. Though as instrument recording interval decreases the more negligible the difference between summed values and area under the curve becomes. Maximum peak values provided the least accuracy due to the injection duration variations between samples, which can change the shape of the peak substantially. One researcher may inject samples faster creating higher and narrower peaks, while another may draw it out creating shorter yet wider peaks. Even injections from the same researcher can fluctuate as they work through samples. In this case despite the samples having similar areas under the curve they would have different maximum peak values which if used to calculate concentrations produces erroneous results.

A powerful feature of this package is its ability to preserve the order in which the samples are sequentially run through the instrument regardless of annotation redundancies. When using summarizing functions in R, a common problem is the automatic reordering of the summarized output which makes it more challenging to link sampling notes with the output. By preserving the order in which standards and samples were injected into the instrument, we make it easier for the user to compare sample notes to output data and address any discrepancies (such as false peaks caused by an improper cutoff point, noted in the paragraph above).

The ability of our package to process through an entire folder of text file results while showing a progress bar grants the user peace of mind that the function is operating properly, but this automation carries with it some inherent problems. If one file out of the folder contains an error or is not identical to the rest, the function will break down and no output will be generated. If an error occurs, the user will be shown an error message stating which file wasn't formatted properly, and they will have to make the necessary corrections.

This package will continue to be maintained and improved upon with community feedback. We hope to continue developing helpful warning messages to inform and guide the user to more accurate analysis. We are developing more arguments in the plotting functions to allow for easy baseline drift investigation. We already offer warning messages for check standards that fall outside of our curve equation confidence interval, though we are hoping to develop automatic baseline correction before peak extraction to allow for greater accuracy and less variability. Arguments suggested by the user community via GitHub will continue to be developed and added to our data processing functions to allow for greater customization for the user. Additionally, we intend to record and produce a video demonstrating the protocol and integration of the software.

In conclusion, this package will help those in that community searching for more efficient ways to process their gas data. The peak.gas function is but one R package in a pantheon of tools that are available for streamlining data processing (e.g., Ottensmann 2020, Vivo-Truyols et al., 2005). This package development is made possible by efforts of the broader R community, which is committed to open-source publication, and sites like github.com that allow for easy package publication and utilization.

References

Anderson J.F., andersonjake1988/peak.gas, (2021) GitHub repository, https://github.com/andersonjake1988/peak.gas

Joos, O., Saurer, M., Heim, A., Hagedorn, F., Schmidt, M. W., & Siegwolf, R. T. (2008). Can we use the CO2 concentrations determined by continuous-flow isotope ratio mass spectrometry from small samples for the Keeling plot approach?. *Rapid Communications in Mass Spectrometry: An International Journal Devoted to the Rapid Dissemination of Up-to-the-Minute Research in Mass Spectrometry*, 22(24), 4029-4034.

Kuang, X., Shankar, T. J., Bi, X. T., Lim, C. J., Sokhansanj, S., & Melin, S. (2009). Rate and peak concentrations of off-gas emissions in stored wood pellets—sensitivities to temperature, relative humidity, and headspace volume. Annals of Occupational Hygiene, 53(8), 789-796.

LI-COR. (2021). LI-COR, Inc. https://www.licor.com/

LI-COR. (n.d.). *8100A and LI-8150 Soil Co2 Flux System*. LI. Retrieved September 9, 2021, from https://www.licor.com/env/support/LI-8100A/topics/analyzer-control-unit.html.

Ottensmann, Meinolf, mottensmann/GCalignR (2020) GitHub repository, https://github.com/mottensmann/GCalignR

R Core Team (2021). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Sherrod, L. A., Reeder, J. D., Hunter, W., & Ahuja, L. R. (2012). Rapid and cost-effective method for soil carbon mineralization in static laboratory incubations. *Communications in soil science and plant analysis*, 43(6), 958-972.

The r project for statistical computing. R. (n.d.). Retrieved September 15, 2021, from https://www.r-project.org/.

Vivo-Truyols et al. "Automatic program for peak detection and deconvolution of multi-overlapped chromatographic signals, Part I: Peak detection", Journal of Chromatography A, vol. 1096 (2005) pp 133-145

Wickham H, Averick M, Bryan J, Chang W, McGowan LD, François R, Grolemund G, Hayes A, Henry L, Hester J, Kuhn M, Pedersen TL, Miller E, Bache SM, Müller K, Ooms J, Robinson D, Seidel DP, Spinu V, Takahashi K, Vaughan D, Wilke C, Woo K, Yutani H (2019). "Welcome to the tidyverse." *Journal of Open Source Software*, **4**(43), 1686. doi: 10.21105/joss.01686.

Tables

Table 1: A table of the functions contained within peak.gas and their available arguments.

Function	Arguments and defaults	Description
extract.peaks	cut.off = 2	Cut off value used to define when the peak should start and end. (Where to place the new baseline).
	method = "linear"	Select regression method used for the standard curve, either "linear", or "log".
	standard.sum = F	Specifies whether or not the user wants to output a summary of the standard curve statistics.
	check.stand = F	Logical argument to specify if the user want to compare "check" standards to the "curve" slope.
	check.alpha = .05	Number to set the confidence interval, defaults to 95% confidence interval.
	ci.meth = "avg"	Argument to compare average ("avg") check standards or individual ("indiv") check standards.
Plot.extracted	data	Output from the extract.peaks() function.
	file = NULL	The name of the file the user wishes to plot.
	sample = NULL	The specific sample the user wishes to plot.
	std.curve = F	A logical argument that specifies whether to plot the standard curve or not.
	method = "linear"	Specify whether you want to plot the "linear" relationship or the "log" transformed relationship of standard curve data.
timeseries.peaks	none	NA
Plot.timeseries	data	output from the timeseries.peaks() function
	file	the name of the file the user wishes to plot
	sample = NULL	the specific sample the user wishes to plot
	time.start = NULL	option to specify when you want the plot to start
	time.stop = NULL	option to specify when you want the plot to stop

Table 2: A table describing the functions included in peak.gas

Function	Function Description	Output	Output Description
extract.peaks	This function will batch process a folder of identically structured text files, to combine and extract useful peak information.	File_Name	The name of the text file the peak was taken from.
		Sample	The name of the sample annotation created when using the LI-8100A.
		Replicate	Number system to indicate how many peaks each sample name holds.
		Order_Run	A number to indicate the order in which the samples were run according to the text file.
		AUC	The sum of the area under each peak.
		Peak	The maximum peak value in each peak range.
		Time_Peak_Start	The time the peak started in datetime format.
		Time_Peak_End	The time the peak ended in datetime format.
		Timespan_(s)	The total duration of the peak in seconds.
		AUC_ppm	Corrected ppm values for area under the curve based on standards supplied by the user.
Plot.extracted	Uses output generated from the extract.peaks() function to provide a visualization tool helpful in accessing quality of data post processing.	Plot	A user generated plot based on arguments selected in the function call
timeseries.peaks	This function will batch process a folder of identically structured text files, to combine and assign file and sample names to each instrument reading while preserving the time series structure of the files.	File_Name	The name of the file the sample was taken from.
		Sample	The name of the sample annotation created when using the LI-8100A.
		Time	The time in date time format that a concentration value was recorded by the instrument.
		CO2	A value recorded by the instrument specifying the concentration of the CO_2 in units specified by user before measured recording.
Plot.timeseries	Uses output generated from the timeseries.peaks() function to provide a visualization tool helpful in accessing timeseries data before peaks are identified.	Plot	A user generated plot based on arguments selected in the function call

Figures

Etime	Time	CO2 (µmol/mol)
0	2021-08-02 12:19:47	-0.75
1	2021-08-02 12:19:48	-0.74
2	2021-08-02 12:19:49	-0.88
3	2021-08-02 12:19:50	-0.78
4	2021-08-02 12:19:51	-0.8
5	2021-08-02 12:19:52	-0.8
6	2021-08-02 12:19:53	-0.73
7	2021-08-02 12:19:54	-0.69
8	2021-08-02 12:19:55	-0.69
9	2021-08-02 12:19:56	-0.74
10	2021-08-02 12:19:57	-0.69
11	2021-08-02 12:19:58	-0.72
12	2021-08-02 12:19:59	-0.75
13	2021-08-02 12:20:00	-0.72
14	2021-08-02 12:20:01	-0.74
15	2021-08-02 12:20:02	-0.93
16	2021-08-02 12:20:03	-0.8
17	2021-08-02 12:20:05	-0.61
18	2021-08-02 12:20:06	-0.55
19	2021-08-02 12:20:07	-0.6
20	2021-08-02 12:20:08	-0.72
21	2021-08-02 12:20:09	-0.75
22	2021-08-02 12:20:10	-0.77
23	2021-08-02 12:20:11	-0.77
500curve		
24	2021-08-02 12:20:12	-0.75
25	2021-08-02 12:20:13	-0.71
26	2021-08-02 12:20:14	-0.64
27	2021-08-02 12:20:15	-0.6

Figure 1. An example carbon dioxide timeseries IRGA text file output from an LI-8100 gas analyzer

Sample	Test	Time	CO2
500curve/1	24	2021-08-02 12:20:12	-0.75
500curve/1	25	2021-08-02 12:20:13	-0.71
500curve/1	26	2021-08-02 12:20:14	-0.64
500curve/1	27	2021-08-02 12:20:15	-0.60
500curve/1	28	2021-08-02 12:20:16	-0.56
500curve/1	29	2021-08-02 12:20:17	-0.63
500curve/1	30	2021-08-02 12:20:18	-0.75
500curve/1	31	2021-08-02 12:20:19	-0.74
500curve/1	32	2021-08-02 12:20:20	-0.77
500curve/1	33	2021-08-02 12:20:21	-0.76
500curve/1	34	2021-08-02 12:20:22	-0.73
500curve/1	35	2021-08-02 12:20:23	-0.75
500curve/1	36	2021-08-02 12:20:24	-0.85
500curve/1	37	2021-08-02 12:20:25	-0.87
500curve/1	38	2021-08-02 12:20:26	-0.77
500curve/1	39	2021-08-02 12:20:27	-0.74
500curve/1	40	2021-08-02 12:20:28	-0.75
500curve/1	41	2021-08-02 12:20:29	-0.74
500curve/1	42	2021-08-02 12:20:30	-0.80
500curve/1	43	2021-08-02 12:20:31	-0.80
500curve/1	44	2021-08-02 12:20:32	-0.79

Figure 2. The first process in extract.peaks() function extracts the sample name and assigns a number to preserve sample order

Sample	Peaks	Value	Replicate	Time
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:22
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:23
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:24
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:25
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:26
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:27
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:28
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:29
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:30
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:31
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:32
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:33
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:34
500curve/1	2.00	FALSE	1	2021-08-02 12:20:35
500curve/1	10.52	TRUE	1	2021-08-02 12:20:36
500curve/1	15.51	TRUE	1	2021-08-02 12:20:37
500curve/1	15.34	TRUE	1	2021-08-02 12:20:38
500curve/1	13.44	TRUE	1	2021-08-02 12:20:39
500curve/1	3.70	TRUE	1	2021-08-02 12:20:40
500curve/1	2.00	FALSE	1	2021-08-02 12:20:41
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:42
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:43
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:44
500curve/1	*NA*	FALSE	*NA*	2021-08-02 12:20:45

Figure 3. The second process in extract.peaks() function cleaves the data at the specified cutoff point and assigns a replicate number to the peak

File_Name	Sample	Order_Run	Area_Under_Curve	Peak	Time_Peak_Start	Time_Peak_End
vn_veg_07292021.txt	500curve. 1	1	48.510	15.51	2021-08-02 12:20:35	2021-08-02 12:20:41
vn_veg_07292021.txt	500curve. 2	1	49.050	15.46	2021-08-02 12:21:00	2021-08-02 12:21:06
vn_veg_07292021.txt	500curve. 3	1	41.550	13.88	2021-08-02 12:21:37	2021-08-02 12:21:42
vn_veg_07292021.txt	500curve. 4	1	42.730	13.77	2021-08-02 12:22:07	2021-08-02 12:22:13
vn_veg_07292021.txt	1000curve. 1	2	57.080	17.85	2021-08-02 12:22:46	2021-08-02 12:22:52
vn_veg_07292021.txt	1000curve. 2	2	57.710	17.46	2021-08-02 12:23:20	2021-08-02 12:23:26
vn_veg_07292021.txt	1000curve. 3	2	66.350	19.94	2021-08-02 12:23:51	2021-08-02 12:23:57
vn_veg_07292021.txt	1000curve. 4	2	84.250	22.38	2021-08-02 12:24:20	2021-08-02 12:24:27
vn_veg_07292021.txt	1000curve. 5	2	78.460	22.97	2021-08-02 12:24:50	2021-08-02 12:24:56
vn_veg_07292021.txt	1000curve. 6	2	97.480	27.65	2021-08-02 12:25:16	2021-08-02 12:25:22
vn_veg_07292021.txt	1000curve. 7	2	86.990	22.59	2021-08-02 12:25:47	2021-08-02 12:25:54
vn_veg_07292021.txt	5000curve. 1	3	540.920	142.00	2021-08-02 12:26:25	2021-08-02 12:26:33
vn_veg_07292021.txt	5000curve. 2	3	431.850	113.54	2021-08-02 12:26:56	2021-08-02 12:27:03
vn_veg_07292021.txt	5000curve. 3	3	564.470	147.66	2021-08-02 12:27:26	2021-08-02 12:27:33
vn_veg_07292021.txt	5000curve. 4	3	570.720	136.42	2021-08-02 12:28:00	2021-08-02 12:28:07
vn_veg_07292021.txt	NN_Veg. 1	4	90.800	25.79	2021-08-02 12:29:08	2021-08-02 12:29:14
vn_veg_07292021.txt	NN_Veg. 2	4	110.620	30.75	2021-08-02 12:29:30	2021-08-02 12:29:36
vn_veg_07292021.txt	NN_Veg. 3	4	144.930	40.37	2021-08-02 12:29:52	2021-08-02 12:29:58
vn_veg_07292021.txt	NN_Veg. 4	4	50.290	16.16	2021-08-02 12:30:15	2021-08-02 12:30:20
vn_veg_07292021.txt	NN_Veg. 5	4	78.440	23.05	2021-08-02 12:30:34	2021-08-02 12:30:40
vn_veg_07292021.txt	NN_Veg. 6	4	112.370	32.47	2021-08-02 12:30:56	2021-08-02 12:31:03

Figure 4. The third process in extract.peaks() function summarizes the maximum peak value, Area under the curve, and start/stop time for each peak

File_Name	Sample	Order_Run	Area_Under_Curve	Peak	Time_Peak_Start	Time_Peak_End	Timespan_(s)	Standard
vn_veg_07292021.txt	500curve. 1	1	48.51	15.51	2021-08-02 12:20:35	2021-08-02 12:20:41	6	500
vn_veg_07292021.txt	500curve. 2	1	49.05	15.46	2021-08-02 12:21:00	2021-08-02 12:21:06	6	500
vn_veg_07292021.txt	500curve. 3	1	41.55	13.88	2021-08-02 12:21:37	2021-08-02 12:21:42	5	500
vn_veg_07292021.txt	500curve. 4	1	42.73	13.77	2021-08-02 12:22:07	2021-08-02 12:22:13	6	500
vn_veg_07292021.txt	1000curve. 1	2	57.08	17.85	2021-08-02 12:22:46	2021-08-02 12:22:52	6	1000
vn_veg_07292021.txt	1000curve. 2	2	57.71	17.46	2021-08-02 12:23:20	2021-08-02 12:23:26	6	1000
vn_veg_07292021.txt	1000curve. 3	2	66.35	19.94	2021-08-02 12:23:51	2021-08-02 12:23:57	6	1000
vn_veg_07292021.txt	1000curve. 4	2	84.25	22.38	2021-08-02 12:24:20	2021-08-02 12:24:27	7	1000
vn_veg_07292021.txt	1000curve. 5	2	78.46	22.97	2021-08-02 12:24:50	2021-08-02 12:24:56	6	1000
vn_veg_07292021.txt	1000curve. 6	2	97.48	27.65	2021-08-02 12:25:16	2021-08-02 12:25:22	6	1000
vn_veg_07292021.txt	1000curve. 7	2	86.99	22.59	2021-08-02 12:25:47	2021-08-02 12:25:54	7	1000
vn_veg_07292021.txt	5000curve. 1	3	540.92	142.00	2021-08-02 12:26:25	2021-08-02 12:26:33	8	5000
vn_veg_07292021.txt	5000curve. 2	3	431.85	113.54	2021-08-02 12:26:56	2021-08-02 12:27:03	7	5000
vn_veg_07292021.txt	5000curve. 3	3	564.47	147.66	2021-08-02 12:27:26	2021-08-02 12:27:33	7	5000
vn_veg_07292021.txt	5000curve. 4	3	570.72	136.42	2021-08-02 12:28:00	2021-08-02 12:28:07	7	5000

Figure 5. The fourth process in extract.peaks() function extracts the known standard concentrations from the data to run a calibration curve

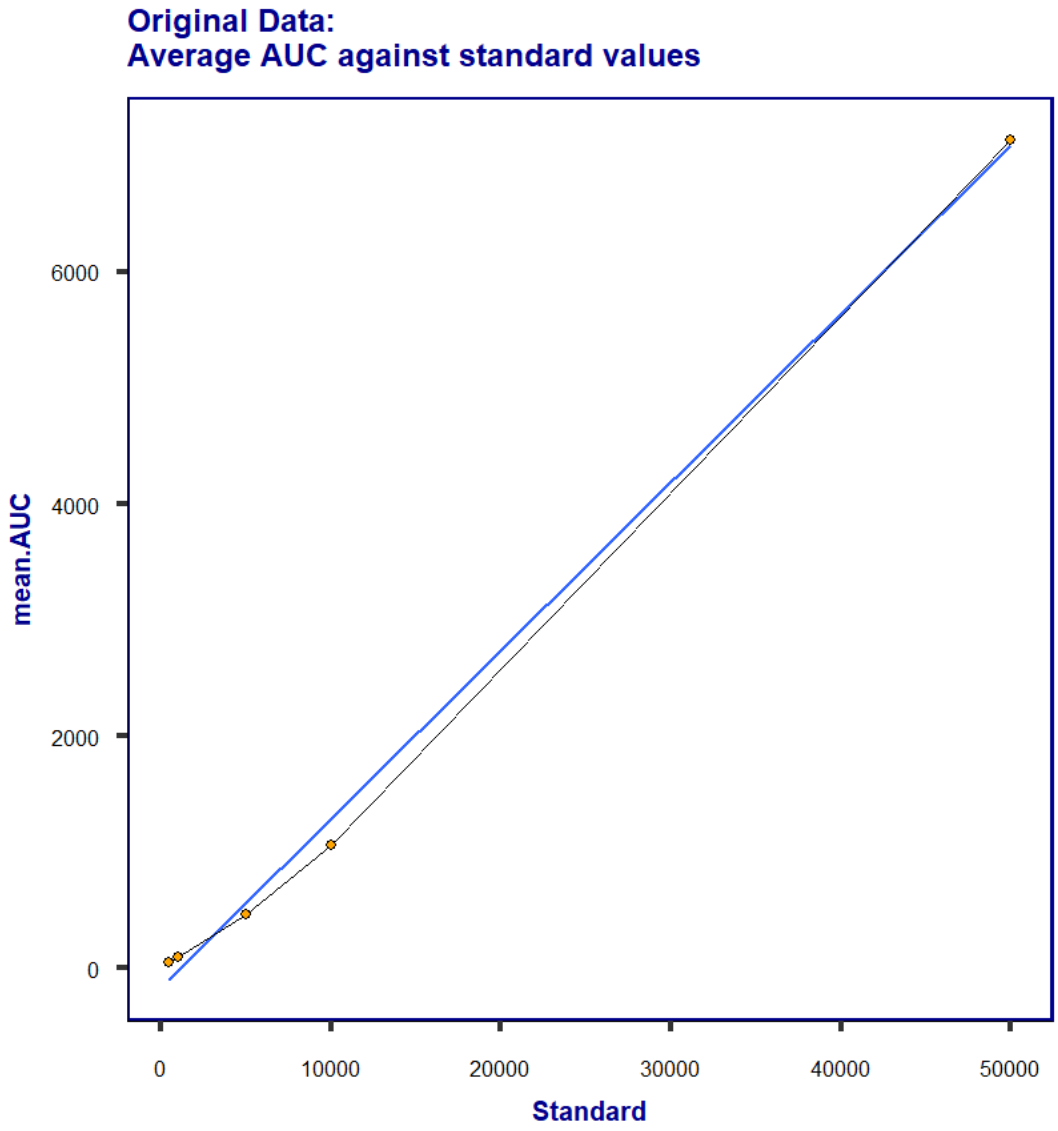


Figure 6. Average area under the curve values across 78 standard samples taken from incubation experiments plotted against the know ppm values (500, 1000, 5000, 10000, 50000), demonstrating how standard values form a slight upward parabolic relationship as known standard values increase in orders of magnitude.

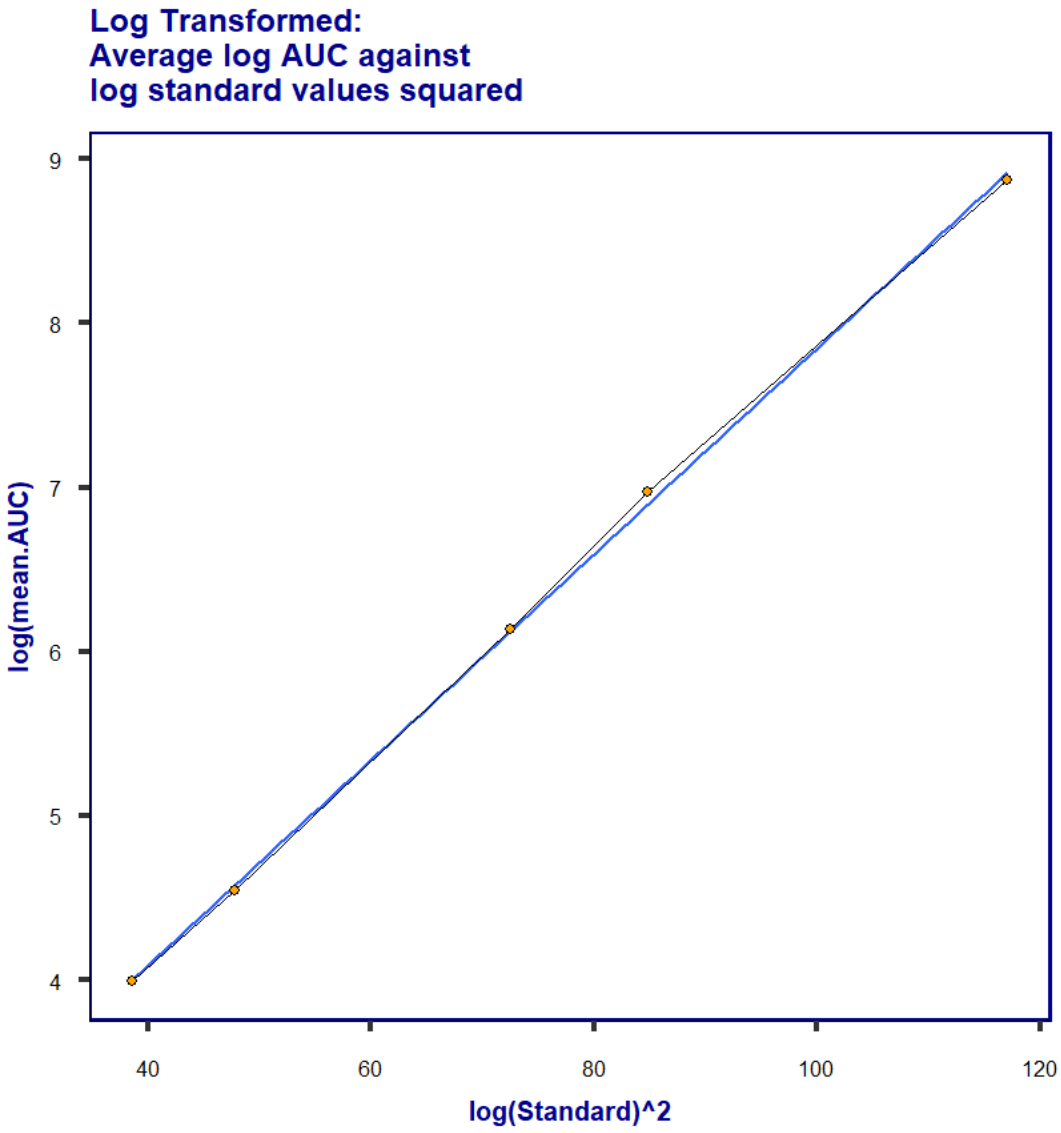


Figure 7. Average log transformed area under the curve values using 78 standard samples taken from incubation experiments, plotted against log transformed standard values (500, 1000, 5000, 10000, 50000) squared. Demonstrating correction of the upward parabolic relationship formed with the original data(Fig 6).

File_Name	Y.intercept	Slope	R.squared
vn_clear_071621.txt	63.45522	9.291256	0.9469338
vn_clear_07292021.txt	106.89702	7.628548	0.9862539
vn_veg_07292021.txt	266.34159	8.872893	0.9756778

Figure 8. The fifth process in extract.peaks() function calculates and extracts the linear equation statistics for all files containing known standard concentrations

File_Name	n	n2	curve
vn_clear_071621.txt	72	66	TRUE
vn_clear_07292021.txt	83	71	TRUE
vn_darkveg_071621.txt	108	108	FALSE
vn_veg_07292021.txt	364	334	TRUE

Figure 9. The sixth process in extract.peaks() function checks if a file contains any standard curves

File_Name	Y.intercept	Slope	R.squared
vn_clear_071621.txt	63.4552238573522	9.29125595571659	0.94693383615345
vn_darkveg_071621.txt	*NA*	*NA*	*NA*
vn_clear_07292021.txt	106.897020257476	7.6285479094335	0.986253882414221
vn_veg_07292021.txt	266.341594905358	8.87289259379289	0.975677848949178

Figure 10. The seventh process in extract.peaks() function orders the files by the time the sample was collected

File_Name	Y.intercept	Slope	R.squared
vn_clear_071621.txt	63.45522	9.291256	0.9469338
vn_darkveg_071621.txt	63.45522	9.291256	0.9469338
vn_clear_07292021.txt	106.89702	7.628548	0.9862539
vn_veg_07292021.txt	266.34159	8.872893	0.9756778

Figure 11. The eighth process in extract.peaks() function fills in linear equation statistics for files that didn't contain standard curves.

File_Name	Standard	Mean	std.dev	COV	Y.intercept	Slope	R.squared	Mean_ppm
vn_clear_071621.txt	300	26.03000	3.896421	0.14968964	-4.54500	0.1019167	0.9469338	300.0000
vn_clear_071621.txt	500	46.41333	1.510143	0.03253685	-4.54500	0.1019167	0.9469338	500.0000
vn_clear_07292021.txt	300	25.72750	1.969626	0.07655721	-12.93160	0.1292846	0.9862539	299.0232
vn_clear_07292021.txt	500	51.88750	2.646613	0.05100675	-12.93160	0.1292846	0.9862539	501.3675
vn_clear_07292021.txt	1000	116.30250	8.355227	0.07184048	-12.93160	0.1292846	0.9862539	999.6093
vn_veg_07292021.txt	500	45.46000	3.870039	0.08513064	-24.71784	0.1099616	0.9756778	638.2029
vn_veg_07292021.txt	1000	75.47429	15.491734	0.20525844	-24.71784	0.1099616	0.9756778	911.1553
vn_veg_07292021.txt	5000	526.99000	64.711482	0.12279452	-24.71784	0.1099616	0.9756778	5017.2754

Figure 12. The ninth process in extract.peaks() function which collects and calculates standard curve summary statistics to be displayed to the user if specified using the argument std.summary= TRUE

File_Name	Sample	Replicate	Order_Run	Time_Peak_Start	Time_Peak_End	Timespan_(s)	AUC	Peak	AUC_ppm
vn_clear_071621.txt	300curve	1	1	2021-07-16 11:32:07	2021-07-16 11:32:13	6	30.470	10.51	343.5650
vn_clear_071621.txt	300curve	2	1	2021-07-16 11:32:51	2021-07-16 11:32:57	6	24.440	9.02	284.3990
vn_clear_071621.txt	300curve	3	1	2021-07-16 11:33:19	2021-07-16 11:33:24	5	23.180	8.69	272.0360
vn_clear_071621.txt	500curve	1	2	2021-07-16 11:34:25	2021-07-16 11:34:31	6	44.760	14.44	483.7776
vn_clear_071621.txt	500curve	2	2	2021-07-16 11:35:18	2021-07-16 11:35:24	6	46.760	14.89	503.4015
vn_clear_071621.txt	500curve	3	2	2021-07-16 11:36:03	2021-07-16 11:36:09	6	47.720	15.38	512.8209
vn_clear_071621.txt	n-castle	1	3	2021-07-16 11:38:19	2021-07-16 11:38:25	6	40.920	13.32	446.0998
vn_clear_071621.txt	n-castle	2	3	2021-07-16 11:39:45	2021-07-16 11:39:51	6	32.420	11.40	362.6983
vn_clear_071621.txt	n-castle	3	3	2021-07-16 11:40:26	2021-07-16 11:40:31	5	34.185	12.54	380.0164
vn_clear_071621.txt	n-castle	4	3	2021-07-16 11:41:08	2021-07-16 11:41:14	6	40.890	13.64	445.8054
vn_clear_071621.txt	n-castle	5	3	2021-07-16 11:44:34	2021-07-16 11:44:40	6	42.070	12.56	457.3835
vn_clear_071621.txt	n-castle	6	3	2021-07-16 11:45:07	2021-07-16 11:45:13	6	29.980	10.80	338.7572
vn_clear_071621.txt	n-castle	7	3	2021-07-16 11:45:46	2021-07-16 11:45:52	6	34.730	11.95	385.3639
vn_clear_071621.txt	n-castle	8	3	2021-07-16 11:46:22	2021-07-16 11:46:29	7	34.100	11.48	379.1823
vn_clear_071621.txt	n-castle	9	3	2021-07-16 11:47:26	2021-07-16 11:47:32	6	38.270	12.74	420.0981
vn_clear_071621.txt	n-castle	10	3	2021-07-16 11:48:01	2021-07-16 11:48:06	5	35.370	12.40	391.6435
vn_clear_071621.txt	n-castle	11	3	2021-07-16 11:48:36	2021-07-16 11:48:42	6	41.530	12.82	452.0850
vn_clear_071621.txt	n-castle	12	3	2021-07-16 11:49:08	2021-07-16 11:49:14	6	37.890	13.00	416.3696
vn_clear_071621.txt	n-castle	13	3	2021-07-16 11:49:54	2021-07-16 11:49:59	5	40.450	13.87	441.4881
vn_clear_071621.txt	n-castle	14	3	2021-07-16 11:50:29	2021-07-16 11:50:35	6	36.630	12.60	404.0065
vn_clear_071621.txt	n-castle	15	3	2021-07-16 11:50:58	2021-07-16 11:51:04	6	34.800	12.36	386.0507

Figure 13. An example of the final output from the extract.peaks() function

Appendix 1

LI-COR LI-8100A CO_2 IRGA Benchtop Protocol

Reagents, materials and equipment

1. Carrier gas: N_2 or CO_2-free air.
2. Carrier gas regulator valve
3. A rotameter to control carrier gas flow
4. Evacuated exetainers (for storing standards)
5. Sharpie for labeling exetainers
6. Notebook/pencil to record standard curve peak heights for later reference, sample injection orders and other observations or issues.
7. Syringe. The size depends on the available volume of the sample
    a. Typically, we inject 1 mL of sample using a 1mL syringe
    b. a 20 mL syringe is used to prep standards into evacuated exetainers.
    c. Stopcocks and needles (multiple of both)
8. CO_2 standards within the range of the CO_2 content of samples under analysis
9. LI-8100A Automated Soil CO_2 Flux System
10. Computer with the current LI-8100A instrument software installed from https://www.licor.com/env/support/LI-8100A/software.html

Procedure

1. Connect the gas tank (N_2 or CO_2-free air) system to the LI-8100A at the “air in” port (Appendix Fig 1). If chamber is connected, first proceed to disconnect the hoses: to remove the Air In and Bellows hoses (Appendix Fig 1), slide the collar on the fittings and pull straight out; to remove the Air Out unscrew the fitting, same as the cable that connects the

chamber to the Analyzer Control Unit. When the chamber is disconnected, insert the hose from the tank to the Air In port until you feel the fitting snaps into place.

   a. Ensure the wall-power supply adapter is attached.
   b. Change the septum on the inlet apparatus if necessary. Tighten the nut down firmly (be careful not to over tighten as this could crack the connector tube).

2. Start up the system
   a. Open the LI-8100A case
   b. Connect the LI-8100A to the computer in case is not connected
   c. Turn on the LI-8100A (the power and ready LEDs will light)
   d. Turn on the computer
   e. Access the LI-8100A software on the computer connected to the instrument. The Main Window appears.
      i. Go to the Communication tab and select **Connect** (Ctrl + C). You are asked to select the serial port to which the LI-8100A is connected, just press the Connect button.
      ii. Go to the Utilities tab and select **Manual Controls**, then select the Flow tab. Press the **8100 Flow OFF** button (the flow of the carrier gas is adjusted manually and set to a known rate, the rotameter available in this setting let us adjust to a maximum of 0.5 LPM, however, flow rates between 0.5 and 1.5 LPM are suggested to work well when making injections in the LI-8100A)
   f. Open the CO_2-free air tank and adjust the regulator valve at ~40 psi
   g. Adjust the regulator before the rotameter at ~14 psi, and the rotameter to 0.5 LPM

h. Let the system equilibrate and the IRGA to be ready (Appendix Fig 2). The "IRGA ready" LED will light. This step can take 10+ minutes. We often prep standards during this step.

   NOTE: Steps f and g should be set previously.

3. Data acquisition. The procedure that follows is convenient in terms of that you can directly log files to the PC instead of pulling files off the instrument later; and also, this will not impact the pump, so once it is turned off it will stay off. The stored file will be a delimited text file.
   a. Go to the Utilities tab and select **PC Data Logging**
   b. Under the Data Values select the data you are interested on collecting such as Time and CO_2 (umol/mol)
   c. Under Field Delimiter, select Tab
   d. Under Log Frequency select 1 second
   e. Under Controls, press the Start button. You are prompted for a file name and destination for the log file. Note where the file is being saved to. The data logging begins.
4. Annotating samples: To best utilize sample processing code (below), standards and samples should be annotated differently, but all included on the same run.
   a. Annotating is done by typing in text into the box at the bottom of the PC datalogging window and clicking "annotate." *NOTE* you will not receive any form of confirmation that your annotation has been added
      i. DO NOT CLICK OK! Leave the window open.
   b. Annotating standards correctly:
      i. Note the CO_2 concentration followed by "curve"

ii. Should look like “300curve”

c. Annotating samples and check standards:

i. Each sample should be annotated, for it will be run in triplicate. The triplicates do not get annotated.

ii. If running fluxes, always run samples in the order of the flux (e.g., T0, T1, T2 should be in order). This prevents instrument drift impacts that could arise if all the T0s were run together, followed by all the T1s).

iii. Check standards should be noted as the concentration followed by “check” (E.g., 300check.)

d. Keep a paper record of your standard sample injections. Try to watch for the approximate peak height of the standard curves so you can visually compare with the check standards later in the run

5. Injecting standards and samples: All standards and samples should be stored in exetainers (unless injecting straight from incubation jars) and should be injected in triplicate. If using exetainers, they should be positively pressurized (We usually inject 18 mL into a 12 mL exetainer). Standards and samples should be injected similarly. Leaving the needles in the septa reduces the chances of the needle becoming clogged by septa rubber (a common problem we’ve experienced). To do this follow the process below.

a. Make two stopcock-needle combos:

i. Put a needle on a stopcock. Close the stopcock. Insert the needle and stopcock into the septum.

ii. Put another needle on another stopcock. Close that stopcock. You’ll insert that into your exetainers or incubation jars.

b. Put a third stopcock (without a needle) on a syringe (typically this will be 1mL).

c. Put the syringe & stopcock onto the exetainer/jar with the needle/stopcock. You'll now have two stopcocks next to each other with a needle on one side and a syringe on the other.
d. Open both stopcocks.
e. Withdraw > 1 mL of gas
f. Close both stopcocks.
g. Separate the two stopcocks. A little twist can help.
h. While pushing up towards the 1mL line, open the stopcock attached to the syringe. Vent to 1mL. Close the stopcock.
i. Attach the 1mL syringe and stopcock to the stopcock/needle in the septum on the IRGA.
j. Open both stopcocks. Inject your sample at a consistent speed finishing in 1 second or a little under. Say "one-one thousand" to get a sense for how long it took you to inject. We usually inject at a pace of "one-one-thou."
k. Close both stopcocks.
l. Remove the syringe-stopcock combo. Start again at step 4d, repeating twice more. That standard or sample has now been run in triplicate.

6. Check standards
    a. Check standards should be run every 10-15 samples.
    b. Check standards should be taken randomly from the standards used in the standard curve
    c. Store check standards in Exetainers. Inject them the same way as the samples and standards.
7. Stopping and shutting down:

a. Press the stop button on the software to end the session
b. Go to the Communication tab and select **Disconnect**
c. Close the program
d. Turn off the computer
e. Turn off the LI-8100A
f. Make sure the CO_2-free air tank is closed
g. Close the LI-8100A case

8. Sample analysis:
    a. Get your data off the computer attached to the LI-COR with a thumb drive and onto a machine with R.
    b. Consolidate all your text file outputs into a single folder. There should be no other TXT files in this folder.
    c. Using Rstudio, set your working directory to the folder containing your TXT files using the command setwd("path to folder") or by clicking on session/set working directory/choose directory (Appendix Fig 3)
    d. Load the peak.gas package into your working environment
    e. Specify where you want to cut the peaks by setting the cut.off argument and store as an object such as: output <- extract.peaks(cut.off = 2)
    f. *NOTE* The cut.off point should be specified high enough to eliminate sample noise, if set too low there may be "false" peaks (Appendix Fig 4)
    g. After you run the extract.peaks() function you will see a loading bar as the function loops through the folder. Generating an output containing 10 columns (Figure 10, Table 2):

h. If you want to export the output to a csv use the command: write.csv(output, "Licor output.csv"), this will save a .csv file called “Licor output” into the folder you set as your working directory

i. If you want to generate standard curve plots or look at a subsample of the data to see if the patterns you expect are present, you can use the Plot.extracted() function included in the package. Plot.extracted() has defaults set as Plot.extracted(data, file = NULL, sample = NULL, std.curve = FALSE, method = “linear”)

j. for standard curve plots you simply need to specify the std.curve argument = TRUE, if a file name is not specified all standard curves will be plotted with check standards for reference, otherwise you can select a specific file name to plot (Appendix Fig 5-7)

Troubleshooting:

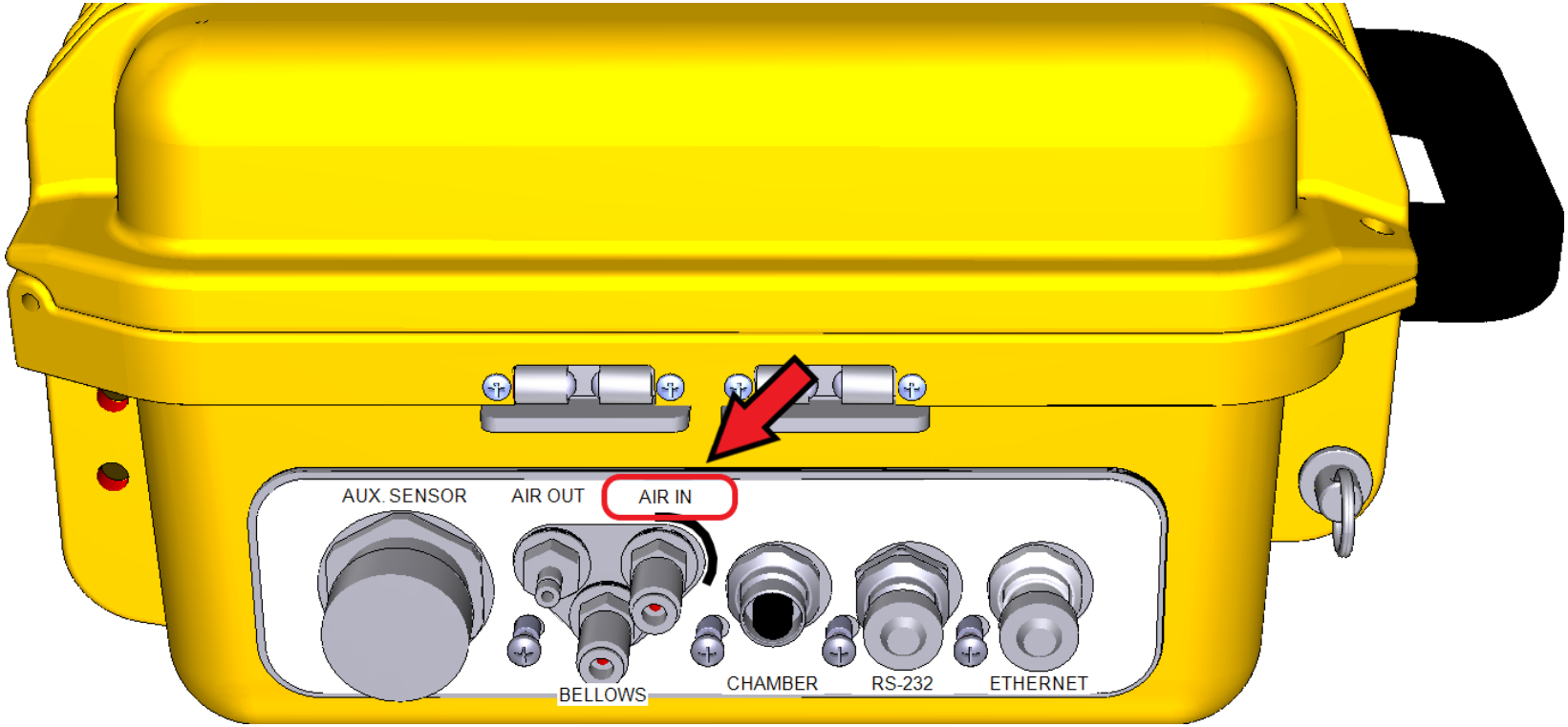
1) Stay organized. Store and inject samples in an order that makes sense. Double check labels.
2) If check standards seem strange, or your standard curve has errors, you may have a chunk of septa in the needle that is restricting airflow. Change the needle or clear out the stuck bit if possible (that can often be achieved by filling a 60 mL syringe with lab air, attaching it to the stuck needle, and pushing hard on the plunger).
3) Know where you saved your file.

Best Practices

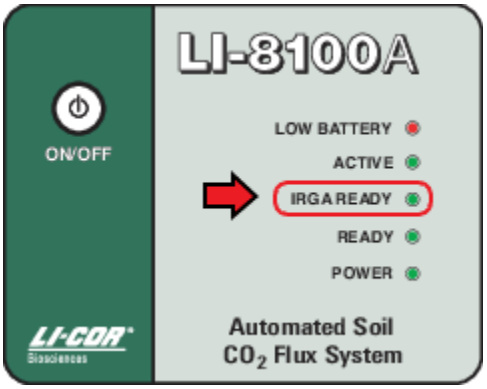
To save time and prevent re-running samples, the user should run the known standard concentrations first to determine if adjustments or recalibration are required before processing their unknown samples. It is recommended that the user first look at the raw timeseries data by using timeseries.peaks() followed by Plot.timeseries() to investigate baseline drift and to

determine the appropriate point to set as the cut off before extracting the peaks. A known check standard should be run every 10 -15 samples to help with determining baseline drift.

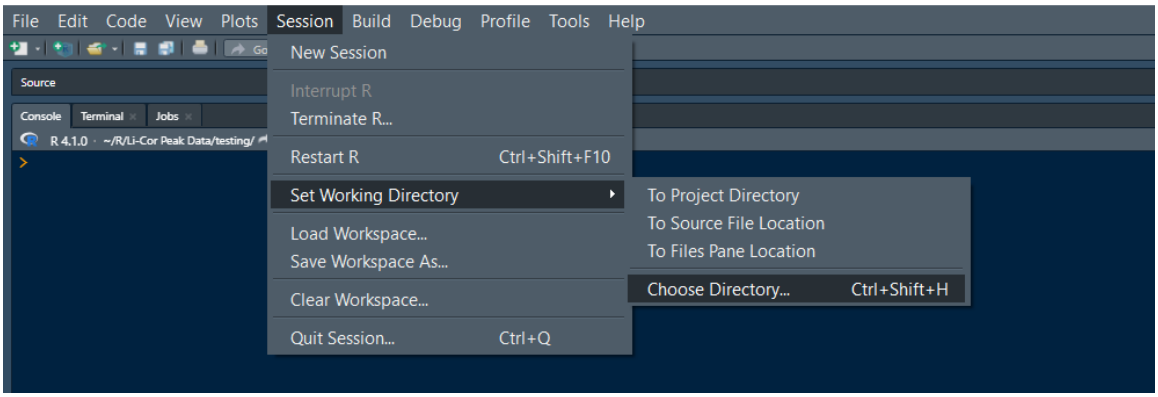
Appendix 1 Figures



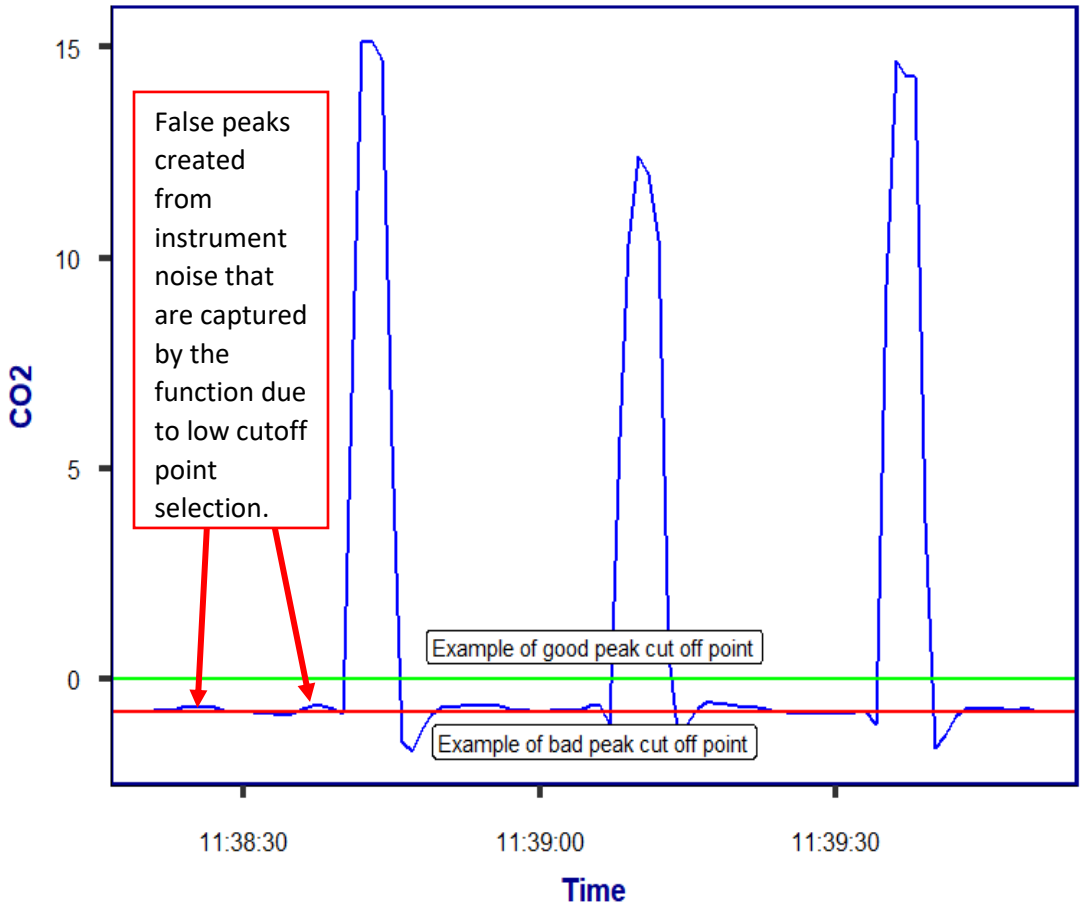
Appendix 1, Figure 1. Modified connector diagram taken from https://www.licor.com/env/support/LI-8100A/topics/8150-connect.html.



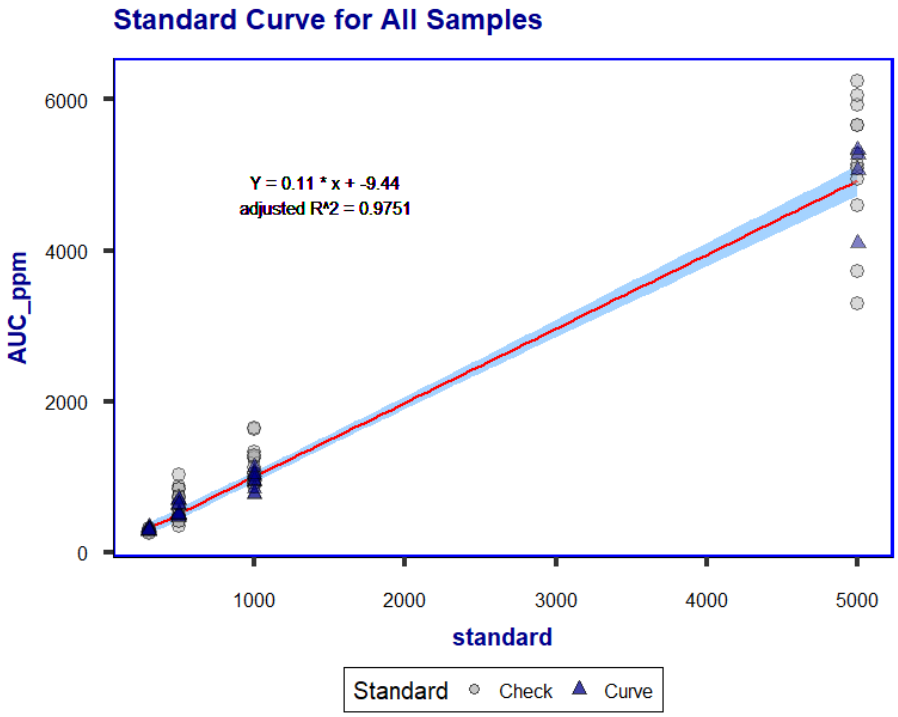
Appendix 1, Figure 2. Indicator panel inside the Analyzer Control Unit modified from https://www.licor.com/env/support/LI-8100A/topics/analyzer-control-unit.html.



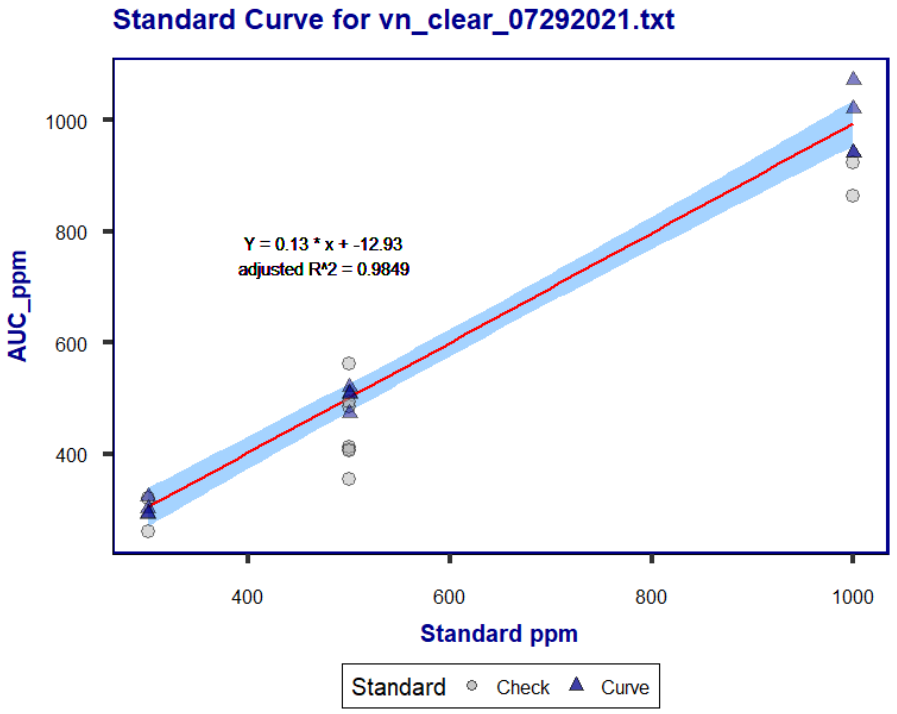
Appendix 1, Figure 3. A pathway to take in order to set your working directory within Rstudio.



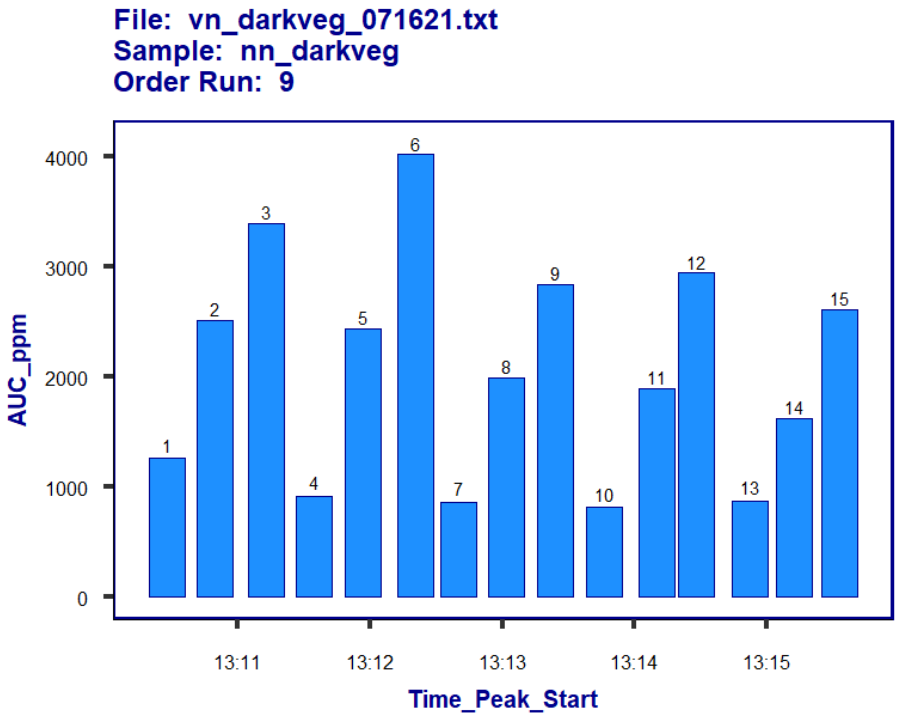
Appendix 1, Figure 4. A sample of three peaks taken from a text file output from an LI-8100A. The values are [CO_2] (µmol/mol) at the time recorded, with each recording collected in second increments. The peaks represent a sample passing through the instrument



Appendix 1, Figure 5. The output from running Plot.extracted(output, std.curve = TRUE)



Appendix 1, Figure 6. The output from running Plot.extracted(output, file = "vn_clear_07292021.txt", std.curve = TRUE)



Appendix 1, Figure 7. Output from running Plot.extracted(output), which selected a random file and sample

Appendix 2

Code

extract.peaks

```
extract.peaks <- function(cut.off = 2, method = "linear", standard.sum = F, check.stand = F,
check.alpha = .05, ci.meth = "avg"){
 Peaks <- function(x){
  output <- vector()
  for(i in 1:length(x)){
   ifelse(x[i] >= cut.off, output[i] <- x[i], output[i] <- NA)
  }
  output
 }
 numextract <- function(string){
  as.numeric(stringr::str_extract(string, "\\-*\\d+\\.*\\d*"))
 }
 filelist <- list.files(pattern = c(".txt", ".TXT"))
 output.raw <- data.frame()
 print("Looping through Folder:")
 progress_bar <- txtProgressBar(min = 0, max = length(filelist), style = 3)
 for(a in 1:length(filelist)){
  setTxtProgressBar(progress_bar, a)
  b <- read.table(filelist[a], header = T, sep = "\t", fill = T, strip.white = T, check.names = F)
  if(length(b) != 3){
   print(' ')
   stop(c(filelist[a], ' is not formatted properly, 3 columns are required.'))
  }
  data.1 <- dplyr::mutate(b, Sample = rep(NA, nrow(b)), .before = 1)
  names(data.1) <- c("Sample", "Test", "Time", "CO2")
  data.2 <- dplyr::filter(data.1, Test != "-----------------------------------------------------------------")
  data.3 <- data.2
  for(i in 1:nrow(data.2)){
   if(is.na(data.2[i,4])==T){
    data.3[i,1] <- as.character(data.2[i,2])
   } else {
    next
   }
  }
  file.annot <- dplyr::filter(data.3, is.na(CO2)) %>%
   dplyr::mutate("temp" = c(1:length(Sample))) %>%
   dplyr::mutate("Sample" = paste0(Sample, "/", temp))
  for(i in 1:nrow(data.3)){
   if(is.na(data.3$CO2[i])){
    data.3$Sample[i] <- file.annot$Sample[1]
    file.annot <- file.annot[-1, ]
```

```
  } else {
    next
  }
}
data.4 <- na.omit(tidyr::fill(data.3, Sample, .direction = "down"))
Preserve.order <- unique(data.4$Sample)
options(dplyr.summarise.inform = FALSE)
test.2 <- data.4 %>%
  dplyr::group_by(Sample) %>%
  dplyr::mutate("Sample" = factor(Sample, levels = Preserve.order)) %>%
  dplyr::summarise("Peaks" = Peaks(CO2)) %>%
  dplyr::arrange(Sample) %>%
  dplyr::mutate("Value" = !is.na(Peaks), "Replicate" = NA)
test.2 <- cbind(test.2, "Time" = lubridate::as_datetime(data.4$Time))
test.2 <- dplyr::arrange(test.2, Time)
r <- 0
for(i in 1:(length(test.2$Value)-1)){
  if(test.2$Value[i] == T & test.2$Value[i+1] == T){
    test.2$Replicate[i] <- r
  } else if(test.2$Value[i] == F & test.2$Value[i+1] == T){
    r <- r + 1
  } else if(test.2$Value[i] == F & test.2$Value[i+1] == F){
    test.2$Replicate[i] <- NA
  } else {
    test.2$Replicate[i] <- r
  }
}
for(i in 2:(length(test.2$Value)-1)){
  if(test.2$Value[i] == F & test.2$Value[i+1] == T){
    test.2$Peaks[i] <- cut.off
    test.2$Replicate[i] <- test.2$Replicate[i+1]
  } else if(test.2$Value[i] == F & test.2$Value[i-1] == T){
    test.2$Peaks[i] <- cut.off
    test.2$Replicate[i] <- test.2$Replicate[i-1]
  } else {
    next
  }
}
test.3 <- na.omit(test.2)
test.3 <- dplyr::mutate(test.3, "Area" = 0)
test.3$diff <- c(0, diff(test.3$Peaks))
peaks <- test.3$Peaks-cut.off
diff <- c(0, diff(peaks))
for(i in 1:(length(test.3$Peaks)-1)){
  if(test.3$Replicate[i] == test.3$Replicate[i+1]){
    time <- as.numeric(difftime(test.3$Time[i+1], test.3$Time[i]))
    if(test.3$Value[i] == F & test.3$Value[i+1] == T){
      test.3$Area[i] <- (time * diff[i+1])/2
```

```
      } else if(test.3$Value[i] == F & test.3$Value[i+1] == F){
        test.3$Area[i] <- (time * diff[i])/2
      } else if(test.3$Value[i] == T){
        test.3$Area[i] <- (peaks[i] * time) + ((time * diff[i+1])/2)
      }
    } else{
      next
    }
  }
  test.4 <- test.3 %>%
    dplyr::group_by(Sample, Replicate)%>%
    dplyr::summarize("AUC" = sum(Area), "Peak" = max(Peaks), "Time_Peak_Start" =
min(Time), "Time_Peak_End" = max(Time)) %>%
    dplyr::mutate("Subsample" = c(1:length(Sample))) %>%
    tidyr::unite(col = "Sample", Sample, Subsample, sep = "/") %>%
    dplyr::arrange(Time_Peak_Start)
  test.5 <- cbind(File_Name = filelist[a], test.4)
  test.5 <- tidyr::separate(test.5, Sample, c("Sample", "Annotation", "Replicate"), sep = "/")
  test.5 <- dplyr::mutate(test.5, "Order_Run" = rep(1, dplyr::n()), .before = 3)
  g <- 1
  for(i in 2:nrow(test.5)){
    if(test.5$Sample[i] == test.5$Sample[i-1]){
      test.5$Order_Run[i] <- g
    } else {
      g <- g + 1
      test.5$Order_Run[i] <- g
    }
  }
  output.raw <- rbind(output.raw, test.5)
 }
 print("done")
 output <- output.raw %>%
  dplyr::group_by(Sample, Order_Run) %>%
  tidyr::unite("Sample", Sample, Replicate, sep = ". ") %>%
  dplyr::select(-Annotation) %>%
  dplyr::arrange(Time_Peak_Start)
 preserve.order <- unique(output$File_Name)
 output <- mutate(output, "Timespan_(s)" = as.numeric(difftime(Time_Peak_End,
Time_Peak_Start)))
 output.1 <- dplyr::filter(output, stringr::str_detect(toupper(Sample), "CURVE"))
 output.1 <- dplyr::mutate(output.1, "Standard" = numextract(Sample))
 output.2 <- dplyr::filter(output, stringr::str_detect(toupper(Sample), "CHECK"))
 output.2 <- dplyr::mutate(output.2, "Standard" = numextract(Sample))
 curve <- data.frame(matrix(ncol = 4, nrow = 0))
 colnames(curve) <- c("File_Name", "Y.intercept", "Slope", "R.squared")
 if(dim(output.1)[1] != 0){
  if(method == "linear"){
    q <- 0
```

```
    output.1 <- mutate(output.1, low = NA, high = NA)
    for(i in 1:length(unique(output.1$File_Name))){
      dat1 <- dplyr::filter(output.1, File_Name == unique(output.1$File_Name)[i])
      linmod <- lm(AUC ~ Standard, data = dat1)
      for(j in 1:nrow(dat1)){
        q <- q+1
        new.dat <- dat1[j,]
        output.1$low[q] <- predict(linmod, newdata = new.dat, interval = 'confidence', level = 1-
check.alpha)[2]
        output.1$high[q] <- predict(linmod, newdata = new.dat, interval = 'confidence', level = 1-
check.alpha)[3]
      }
      asdf <- summary(linmod)
      Y <- asdf$coefficients[1]
      M <- asdf$coefficients[2]
      R <- asdf$r.squared
      curve[i,] <- c(unique(as.character(output.1$File_Name))[i], Y, M, R)
    }
    if(check.stand == T){
      ot1 <- output.1 %>%
      group_by(File_Name, Standard) %>%
      summarize(ci.low = mean(low), ci.high = mean(high))
    if(ci.meth == "avg"){
      ot2 <- output.2 %>%
            group_by(File_Name, Standard) %>%
            summarize(mean.AUC = mean(AUC))
      ot3 <- full_join(ot1, ot2, by = c("File_Name", "Standard"))
      ot3$File_Name <- factor(ot3$File_Name, levels = preserve.order)
      ot3 <- dplyr::arrange(ot3, File_Name)
      ot3$File_Name <- as.character(ot3$File_Name)
      ot4 <- mutate(ot3, checkci = mean.AUC > ci.low & mean.AUC < ci.high)
      cierr <- data.frame(File_Name = NA, Standard = NA, ci.low = NA, ci.high = NA,
mean.AUC = NA)
      naerr <- data.frame(File_Name = NA, Standard = NA, mean.AUC = NA)
    } else if(ci.meth == "indiv"){
      ot2 <- select(output.2, File_Name, Sample, Order_Run, Standard, AUC)
      ot3 <- full_join(ot1, ot2, by = c("File_Name", "Standard"))
      ot3$File_Name <- factor(ot3$File_Name, levels = preserve.order)
      ot3 <- dplyr::arrange(ot3, File_Name, Order_Run)
      ot3$File_Name <- as.character(ot3$File_Name)
      ot4 <- mutate(ot3, checkci = AUC > ci.low & AUC < ci.high)
      cierr <- data.frame(File_Name = NA, Sample = NA, Order_Run = NA, ci.low = NA, ci.high
= NA, AUC = NA)
      naerr <- data.frame(File_Name = NA, Sample = NA, Order_Run = NA, AUC = NA)
    }
    if(any(ot4$checkci == FALSE) | any(is.na(ot4$checkci))){
      for(i in 1:nrow(ot4)){
        if(ci.meth == "avg"){
```

```
          if(is.na(ot4['checkci'][i,])){
            naerr[i,] <- select(ot4[i,], File_Name, Standard, mean.AUC)
          } else if(ot4['checkci'][i,] == FALSE){
            cierr[i,] <- select(ot4[i,], File_Name, Standard, ci.low, ci.high, mean.AUC)
          } else if(ot4['checkci'][i,] == TRUE){
            next
          }
        } else if(ci.meth == "indiv"){
          if(is.na(ot4['checkci'][i,])){
            naerr[i,] <- select(ot4[i,], File_Name, Sample, Order_Run, AUC)
          } else if(ot4['checkci'][i,] == FALSE){
            cierr[i,] <- select(ot4[i,], File_Name, Sample, Order_Run, ci.low, ci.high, AUC)
          } else if(ot4['checkci'][i,] == TRUE){
            next
          }
        }
      }
      cierr <- na.omit(cierr)
      naerr <- na.omit(naerr)
      if(nrow(cierr) != 0){
        warning(call. = F, c("\n\nCheck standards deviate from the ", 100*(1 - check.alpha), "%", "
confidence interval in the following Samples:\n"))
        for(i in 1:nrow(cierr)){
          if(ci.meth == "avg"){
          warning(call. = F, c("File: ", cierr$File_Name[i], "\tStandard: ", cierr$Standard[i],
"check",
                               "\tCI range: ", round(cierr$ci.low[i]), " to ", round(cierr$ci.high[i]), "\tAUC:
", round(cierr$mean.AUC[i], 2)))
          }
          if(ci.meth == "indiv"){
            warning(call. = F, c("File: ", cierr$File_Name[i], "\tSample: ", cierr$Sample[i],
"\tOrder_Run: ", cierr$Order_Run[i],
                                 "\tCI range: ", round(cierr$ci.low[i]), " to ", round(cierr$ci.high[i]),
"\tAUC: ", round(cierr$AUC[i], 2)))
          }
        }
      }
      if(nrow(naerr) != 0){
        warning(call. = F, "\n\nNA values for confidence interval due to missing standard 'curves'
in the following Samples:\n")
        if(ci.meth == "avg"){
          for(i in 1:nrow(naerr)){
            warning(call. = F, c("File: ", naerr$File_Name[i], "\tStandard: ", naerr$Standard[i],
"check", "\tAUC: ", round(naerr$mean.AUC[i], 2)))
          }
        }
        if(ci.meth == "indiv"){
          for(i in 1:nrow(naerr)){
```

```
          warning(call. = F, c("File: ", naerr$File_Name[i], "\tSample: ", naerr$Sample[i],
"\tOrder_Run:", naerr$Order_Run[i], "\tAUC: ", round(naerr$AUC[i], 2)))
          }
        }
      }
    } else{
      next
    }
    }
    curve[, 2:4] <- sapply(curve[, 2:4], as.numeric)
    output.0 <- dplyr::filter(output, !stringr::str_detect(toupper(Sample), "CURVE"))
    inwork <- output %>%
      dplyr::group_by(File_Name) %>%
      dplyr::summarize(n = dplyr::n())
    inwork2 <- output.0 %>%
      dplyr::group_by(File_Name) %>%
      dplyr::summarize(n = dplyr::n())
    if(dim(inwork2)[1]==0){
      curve.2 <- dplyr::mutate(inwork, n2 = 0, "curve" = (n != n2))
    } else {
      curve.2 <- dplyr::mutate(inwork, n2 = inwork2$n, "curve" = (n != n2))
    }
    yes.curve <- data.frame(matrix(ncol = 4, nrow = 0))
    colnames(yes.curve) <- c("File_Name", "Y.intercept", "Slope", "R.squared")
    for(i in 1:length(curve.2$File_Name)){
      for(j in 1:length(curve$File_Name)){
        if(curve.2$File_Name[i] == curve$File_Name[j] & curve.2$curve[i]==T){
          yes.curve[i,] <- curve[j,]
        } else if(curve.2$curve[i]==F){
          yes.curve[i,] <- c(curve.2$File_Name[i], rep(NA,3))
        }
      }
    }
    yes.curve$File_Name <- factor(yes.curve$File_Name, levels = preserve.order)
    yes.curve <- dplyr::arrange(yes.curve, File_Name)
    curve.3 <- tidyr::fill(yes.curve, "File_Name", "Y.intercept", "Slope", "R.squared", .direction =
"down")
    curve.3[, 2:4] <- sapply(curve.3[, 2:4], as.numeric)
    stand <- output.1 %>%
      dplyr::group_by(File_Name, Standard) %>%
      dplyr::summarise(Mean = mean(AUC), std.dev = sd(AUC))%>%
      dplyr::mutate(COV = std.dev/Mean)
    sum.stat <- left_join(stand, curve, by = "File_Name")
    output$AUC_ppm <- 0
    for(i in 1:nrow(output)){
      for(j in 1:nrow(curve.3)){
        if(output$File_Name[i] == curve.3$File_Name[j]){
          output$AUC_ppm[i] <- (output$AUC[i] - curve.3$Y.intercept[j])/(curve.3$Slope[j])
```

```
      }
     }
    }
   } else if(method == "log"){
    q <- 0
    output.1 <- mutate(output.1, low = NA, high = NA)
    for(i in 1:length(unique(output.1$File_Name))){
     dat1 <- dplyr::filter(output.1, File_Name == unique(output.1$File_Name)[i])
     linmod <- lm(log(AUC) ~ log(Standard), data = dat1)
     for(j in 1:nrow(dat1)){
      q <- q+1
      new.dat <- dat1[j,]
      output.1$low[q] <- predict(linmod, newdata = new.dat, interval = 'confidence', level = 1-
check.var)[2]
      output.1$high[q] <- predict(linmod, newdata = new.dat, interval = 'confidence', level = 1-
check.var)[3]
     }
     asdf <- summary(linmod)
     Y <- asdf$coefficients[1]
     M <- asdf$coefficients[2]
     R <- asdf$r.squared
     curve[i,] <- c(unique(as.character(output.1$File_Name))[i], Y, M, R)
    }
    if(check.stand == T){
     ot1 <- output.1 %>%
     group_by(File_Name, Standard) %>%
     summarize(ci.low = mean(low), ci.high = mean(high))
    if(ci.meth == "avg"){
     ot2 <- output.2 %>%
      group_by(File_Name, Standard) %>%
      summarize(mean.AUC = mean(log(AUC)))
     ot3 <- full_join(ot1, ot2, by = c("File_Name", "Standard"))
     ot3$File_Name <- factor(ot3$File_Name, levels = preserve.order)
     ot3 <- dplyr::arrange(ot3, File_Name)
     ot3$File_Name <- as.character(ot3$File_Name)
     ot4 <- mutate(ot3, checkci = mean.AUC > ci.low & mean.AUC < ci.high)
     cierr <- data.frame(File_Name = NA, Standard = NA, ci.low = NA, ci.high = NA,
mean.AUC = NA)
     naerr <- data.frame(File_Name = NA, Standard = NA, mean.AUC = NA)
    } else if(ci.meth == "indiv"){
     ot2 <- select(output.2, File_Name, Sample, Order_Run, Standard, AUC)
     ot2$AUC <- log(ot2$AUC)
     ot3 <- full_join(ot1, ot2, by = c("File_Name", "Standard"))
     ot3$File_Name <- factor(ot3$File_Name, levels = preserve.order)
     ot3 <- dplyr::arrange(ot3, File_Name, Order_Run)
     ot3$File_Name <- as.character(ot3$File_Name)
     ot4 <- mutate(ot3, checkci = AUC > ci.low & AUC < ci.high)
```

```
      cierr <- data.frame(File_Name = NA, Sample = NA, Order_Run = NA, ci.low = NA, ci.high
= NA, AUC = NA)
      naerr <- data.frame(File_Name = NA, Sample = NA, Order_Run = NA, AUC = NA)
    }
    if(any(ot4$checkci == FALSE) | any(is.na(ot4$checkci))){
      for(i in 1:nrow(ot4)){
        if(ci.meth == "avg"){
          if(is.na(ot4['checkci'][i,])){
            naerr[i,] <- select(ot4[i,], File_Name, Standard, mean.AUC)
          } else if(ot4['checkci'][i,] == FALSE){
            cierr[i,] <- select(ot4[i,], File_Name, Standard, ci.low, ci.high, mean.AUC)
          } else if(ot4['checkci'][i,] == TRUE){
            next
          }
        } else if(ci.meth == "indiv"){
          if(is.na(ot4['checkci'][i,])){
            naerr[i,] <- select(ot4[i,], File_Name, Sample, Order_Run, AUC)
          } else if(ot4['checkci'][i,] == FALSE){
            cierr[i,] <- select(ot4[i,], File_Name, Sample, Order_Run, ci.low, ci.high, AUC)
          } else if(ot4['checkci'][i,] == TRUE){
            next
          }
        }
      }
      cierr <- na.omit(cierr)
      naerr <- na.omit(naerr)
      if(nrow(cierr) != 0){
        warning(call. = F, c("\n\nCheck standards deviate from the ", 100*(1 - check.alpha), "%", "
confidence interval in the following Samples:\n"))
        for(i in 1:nrow(cierr)){
          if(ci.meth == "avg"){
            warning(call. = F, c("File: ", cierr$File_Name[i], "\tStandard: ", cierr$Standard[i],
"check",
                                 "\tCI range: ", round(cierr$ci.low[i],2), " to ", round(cierr$ci.high[i],2),
"\tAUC: ", round(cierr$mean.AUC[i], 2)))
          }
          if(ci.meth == "indiv"){
            warning(call. = F, c("File: ", cierr$File_Name[i], "\tSample: ", cierr$Sample[i],
"\tOrder_Run: ", cierr$Order_Run[i],
                                 "\tCI range: ", round(cierr$ci.low[i],2), " to ", round(cierr$ci.high[i],2),
"\tAUC: ", round(cierr$AUC[i], 2)))
          }
        }
      }
      if(nrow(naerr) != 0){
        warning(call. = F, "\n\nNA values for confidence interval due to missing standard 'curves'
in the following Samples:\n")
        if(ci.meth == "avg"){
```

```
          for(i in 1:nrow(naerr)){
            warning(call. = F, c("File: ", naerr$File_Name[i], "\tStandard: ", naerr$Standard[i],
"check", "\tAUC: ", round(naerr$mean.AUC[i], 2)))
          }
        }
        if(ci.meth == "indiv"){
          for(i in 1:nrow(naerr)){
            warning(call. = F, c("File: ", naerr$File_Name[i], "\tSample: ", naerr$Sample[i],
"\tOrder_Run:", naerr$Order_Run[i], "\tAUC: ", round(naerr$AUC[i], 2)))
          }
        }
      }
    } else{
      next
    }
    }
    curve[, 2:4] <- sapply(curve[, 2:4], as.numeric)
    output.0 <- dplyr::filter(output, !stringr::str_detect(toupper(Sample), "CURVE"))
    inwork <- output %>%
      dplyr::group_by(File_Name) %>%
      dplyr::summarize(n = dplyr::n())
    inwork2 <- output.0 %>%
      dplyr::group_by(File_Name) %>%
      dplyr::summarize(n = dplyr::n())
    curve.2 <- dplyr::mutate(inwork, n2 = inwork2$n, "curve" = (n != n2))
    yes.curve <- data.frame(matrix(ncol = 4, nrow = 0))
    colnames(yes.curve) <- c("File_Name", "Y.intercept", "Slope", "R.squared")
    for(i in 1:length(curve.2$File_Name)){
      for(j in 1:length(curve$File_Name)){
        if(curve.2$File_Name[i] == curve$File_Name[j] & curve.2$curve[i]==T){
          yes.curve[i,] <- curve[j,]
        } else if(curve.2$curve[i]==F){
          yes.curve[i,] <- c(curve.2$File_Name[i], rep(NA,3))
        }
      }
    }
    yes.curve$File_Name <- factor(yes.curve$File_Name, levels = preserve.order)
    yes.curve <- dplyr::arrange(yes.curve, File_Name)
    curve.3 <- tidyr::fill(yes.curve, "File_Name", "Y.intercept", "Slope", "R.squared", .direction =
"down")
    curve.3[, 2:4] <- sapply(curve.3[, 2:4], as.numeric)
    stand <- output.1 %>%
      dplyr::group_by(File_Name, Standard) %>%
      dplyr::summarise(Mean = mean(AUC), std.dev = sd(AUC))%>%
      dplyr::mutate(COV = std.dev/Mean)
    sum.stat <- dplyr::left_join(stand, curve, by = "File_Name")
    output$AUC_ppm <- 0
    for(i in 1:nrow(output)){
```

```
      for(j in 1:nrow(curve.3)){
        if(output$File_Name[i] == curve.3$File_Name[j]){
          output$AUC_ppm[i] <- exp((log(output$AUC[i]) -
curve.3$Y.intercept[j])/(curve.3$Slope[j]))
        }
      }
    }
  }
  sum.stat2 <- output %>%
    dplyr::filter(stringr::str_detect(toupper(Sample), "CURVE")) %>%
    dplyr::mutate(Standard = numextract(Sample)) %>%
    dplyr::group_by(Standard, File_Name) %>%
    dplyr::summarise(Mean_ppm = mean(AUC_ppm))
  standard.summary.stats <- dplyr::left_join(sum.stat, sum.stat2, by = c("File_Name",
"Standard"))
  output <- tidyr::separate(output, Sample, c("Sample", "Replicate"), sep = ". ")
  if(standard.sum == T){
    View(standard.summary.stats)
  }
  return(output)
 } else {
  print(' ')
  warning('No standard curve data found, could not compute concentration')
  output$AUC_ppm <- 0
  output <- tidyr::separate(output, Sample, c("Sample", "Replicate"), sep = ". ")
  return(output)
 }
}
```

Plot.extracted

```
Plot.extracted <- function(data, file = NULL, sample = NULL, std.curve = F, method = "linear"){
  UNR <- function(){
    ggplot2::theme(text = ggplot2::element_text(color = "black", size = 15),
                   plot.title = ggplot2::element_text(face = "bold", color = "darkblue", margin =
ggplot2::margin(b = 15)),
                   plot.subtitle = ggplot2::element_text(size = 10),
                   axis.ticks = ggplot2::element_line(size = 1.5),
                   axis.title = ggplot2::element_text(face = "bold", line = 2),
                   axis.title.x = ggplot2::element_text(margin = ggplot2::margin(t = 10), color =
"darkblue", size = 15),
                   axis.title.y = ggplot2::element_text(margin = ggplot2::margin(r = 10), color =
"darkblue", size = 15),
                   axis.title.y.right = ggplot2::element_text(margin = ggplot2::margin(l = 15),color =
"grey50", size = 15),
                   axis.text = ggplot2::element_text(color = "black"),
                   axis.text.x = ggplot2::element_text(margin = ggplot2::margin(t = 15)),
                   axis.text.y = ggplot2::element_text(margin = ggplot2::margin(r = 10)),
                   axis.line = ggplot2::element_line(colour = "black"),
                   axis.ticks.length = ggplot2::unit(2, "mm"),
                   plot.caption = ggplot2::element_text(color = "black"),
                   plot.background = ggplot2::element_rect(fill = "white"),
                   plot.margin = ggplot2::margin(t = 10, r = 50, b = 10, l = 10),
                   panel.background = ggplot2::element_rect(fill = "white"),
                   panel.border = ggplot2::element_rect(fill = "NA", color = "darkblue", size = 1.5),
                   legend.background = ggplot2::element_rect(color = "black", fill = "white"),
                   legend.key =  ggplot2::element_rect(fill = "white"),
                   legend.text = ggplot2::element_text(color = "black"),
                   legend.position = "bottom",
                   strip.background = ggplot2::element_rect(color = "blue", fill = "grey75", size = 2),
                   strip.text.y = ggplot2::element_text(size = 13, face = "bold"),
                   panel.grid = ggplot2::element_blank())
  }
  numextract <- function(string){
    as.numeric(stringr::str_extract(string, "\\-*\\d+\\.*\\d*"))
  }
  ghost <- dplyr::filter(data, !stringr::str_detect(toupper(Sample), "CURVE") &
!stringr::str_detect(toupper(Sample), "CHECK"))
  ghost$Replicate <- as.numeric(ghost$Replicate)
  if(is.null(sample) & is.null(file) & std.curve == F){
    rando.group <- sample(unique(data$File_Name), size = 1)
    samp.run <- sample(unique(dplyr::filter(ghost, File_Name == rando.group)$Order_Run), size
= 1)
    verify <- dplyr::filter(ghost, File_Name == rando.group, Order_Run == samp.run)
    mean.ppm <- mean(verify$AUC_ppm)
    ggplot2::ggplot(verify, ggplot2::aes(x = Time_Peak_Start, y = AUC_ppm)) +
```

```
      ggplot2::geom_col(color = "darkblue", fill = "dodgerblue")+
      ggplot2::ggtitle(paste0("File:  ", verify$File_Name, "\nSample:  ", tolower(verify$Sample),
"\nOrder Run:  ", verify$Order_Run)) +
      ggplot2::geom_text(label = verify$Replicate, nudge_y =
(mean.ppm/log(verify$AUC_ppm)*.40))+
      UNR()
  } else if(!is.null(file) & is.null(sample) & std.curve == F){
    filt <- dplyr::filter(ghost, File_Name == file)
    samp.run <- sample(unique(filt$Order_Run), size = 1)
    verify.1 <- dplyr::filter(filt, Order_Run == samp.run)
    mean.ppm <- mean(verify.1$AUC_ppm)
    ggplot2::ggplot(verify.1, ggplot2::aes(x = Time_Peak_Start, y = AUC_ppm)) +
      ggplot2::geom_col(color = "darkblue", fill = "dodgerblue")+
      ggplot2::ggtitle(paste0("File:  ", verify.1$File_Name, "\nSample:  ",
tolower(verify.1$Sample), "\nOrder Run:  ", verify.1$Order_Run)) +
      ggplot2::geom_text(label = verify.1$Replicate, nudge_y =
(mean.ppm/log(verify.1$AUC_ppm)*.40))+
      UNR()
  } else if(is.null(file) & !is.null(sample) & std.curve == F){
    samp <- dplyr::filter(data, toupper(Sample) == toupper(sample))
    samp2 <- dplyr::filter(samp, File_Name == sample(unique(File_Name), 1))
    samp.run <- sample(unique(samp2$Order_Run), size = 1)
    verify.2 <- dplyr::filter(samp2, Order_Run == samp.run)
    mean.ppm <- mean(verify.2$AUC_ppm)
    ggplot2::ggplot(verify.2, ggplot2::aes(x = Time_Peak_Start, y = AUC_ppm)) +
      ggplot2::geom_col(color = "darkblue", fill = "dodgerblue")+
      ggplot2::ggtitle(paste0("File:  ", verify.2$File_Name, "\nSample:  ",
tolower(verify.2$Sample), "\nOrder Run:  ", verify.2$Order_Run)) +
      ggplot2::geom_text(label = verify.2$Replicate, nudge_y =
(mean.ppm/log(verify.2$AUC_ppm)*.40))+
      UNR()
  } else if(!is.null(file) & !is.null(sample) & std.curve == F){
    verify.3 <- dplyr::filter(ghost, File_Name == file, toupper(Sample) == toupper(sample))
    mean.ppm <- mean(verify.3$AUC_ppm)
    ggplot2::ggplot(verify.3, ggplot2::aes(x = Time_Peak_Start, y = AUC_ppm)) +
      ggplot2::geom_col(color = "darkblue", fill = "dodgerblue")+
      ggplot2::ggtitle(paste0("File:  ", verify.3$File_Name, "\nSample:  ",
tolower(verify.3$Sample), "\nOrder Run:  ", verify.3$Order_Run)) +
      ggplot2::geom_text(label = verify.3$Replicate, nudge_y =
(mean.ppm/log(verify.3$AUC_ppm)*.40))+
      UNR()
  } else if(is.null(file) & is.null(sample) & std.curve == T & method == "linear"){
    curv <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CURVE")) %>%
          dplyr::mutate(standard = numextract(Sample), .before = 3)
    check <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CHECK") )%>%
           dplyr::mutate(standard = numextract(Sample), .before = 3)
    asdf <- summary(lm(AUC ~ standard, data = curv))
    Y <- as.numeric(asdf$coefficients[1])
```

```
   M <- as.numeric(asdf$coefficients[2])
   R <- as.numeric(asdf$adj.r.squared)
   form <- data.frame(Y = Y, M = M, R = R)
   form2 <- paste0("Y = ", round(form$M, digits = 2), " * x", " + ", round(form$Y, digits = 2))
   r.squared <- paste0("adjusted R^2 = ", round(form$R, digits = 4))
   if(max(curv$standard) > max(check$standard)){
    ggplot2::ggplot()+
      ggplot2::geom_smooth(formula = y~x, data = curv, ggplot2::aes(x = standard, y =
AUC_ppm),
                  method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
      ggplot2::geom_point(data = check, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Check"), size = 4, shape = 21, alpha = .5) +
      ggplot2::geom_point(data = curv, ggplot2::aes(x = standard, y = AUC_ppm, fill = "Curve"),
size = 3, shape = 24, alpha = .5) +
      ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
        shape = c(21,24),
        size = c(3,3)))) +
      ggplot2::geom_text(data = curv, ggplot2::aes(x = (((max(standard)-
min(standard))*.25)+min(standard)),
                           y = (((max(AUC_ppm)-min(AUC_ppm))*.75)+min(AUC_ppm)),
                           label = paste(form2, r.squared, sep = "\n"))) +
      ggplot2::ggtitle("Linear Standard Curve for All Samples") +
      ggplot2::xlab("Standard ppm") +
      UNR()
   } else if(max(curv$standard) <= max(check$standard)){
    ggplot2::ggplot()+
      ggplot2::geom_smooth(formula = y~x, data = curv, ggplot2::aes(x = standard, y =
AUC_ppm),
                  method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
      ggplot2::geom_point(data = check, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Check"), size = 4, shape = 21, alpha = .5) +
      ggplot2::geom_point(data = curv, ggplot2::aes(x = standard, y = AUC_ppm, fill = "Curve"),
size = 3, shape = 24, alpha = .5) +
      ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
        shape = c(21,24),
        size = c(3,3)))) +
      ggplot2::geom_text(data = check, ggplot2::aes(x = (((max(standard)-
min(standard))*.25)+min(standard)),
                           y = (((max(AUC_ppm)-min(AUC_ppm))*.75)+min(AUC_ppm)),
                           label = paste(form2, r.squared, sep = "\n"))) +
      ggplot2::ggtitle("Linear Standard Curve for All Samples") +
      ggplot2::xlab("Standard ppm") +
      UNR()
   }
  } else if(!is.null(file) & is.null(sample) & std.curve == T & method == "linear"){
```

```
   curv.2 <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CURVE"), File_Name ==
file) %>%
           dplyr::mutate(standard = numextract(Sample), .before = 3)
   check.2 <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CHECK"), File_Name ==
file)%>%
            dplyr::mutate(standard = numextract(Sample), .before = 3)
   asdf.2 <- summary(lm(AUC ~ standard, data = curv.2))
   Y.2 <- as.numeric(asdf.2$coefficients[1])
   M.2 <- as.numeric(asdf.2$coefficients[2])
   R.2 <- as.numeric(asdf.2$adj.r.squared)
   form.2 <- data.frame(Y = Y.2, M = M.2, R = R.2)
   form2.2 <- paste0("Y = ", round(form.2$M, digits = 2), " * x", " + ", round(form.2$Y, digits =
2))
   r.squared.2 <- paste0("adjusted R", "^", 2,  sep = "", " = ", round(form.2$R, digits = 4))
   if(max(curv.2$standard) > max(check.2$standard)){
     ggplot2::ggplot()+
       ggplot2::geom_smooth(formula = y~x, data = curv.2, ggplot2::aes(x = standard, y =
AUC_ppm),
                  method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
       ggplot2::geom_point(data = check.2, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Check"), size = 4, shape = 21, alpha = .5) +
       ggplot2::geom_point(data = curv.2, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Curve"), size = 3, shape = 24, alpha = .5) +
       ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
         shape = c(21,24),
         size = c(3,3)))) +
       ggplot2::geom_text(data = curv.2, ggplot2::aes(x = (((max(standard)-
min(standard))*.25)+min(standard)),
                                y = (((max(AUC_ppm)-min(AUC_ppm))*.75)+min(AUC_ppm)),
                                label = paste(form2.2, r.squared.2, sep = "\n"))) +
       ggplot2::ggtitle(paste0("Linear Standard Curve for ", curv.2$File_Name)) +
       ggplot2::xlab("Standard ppm") +
       UNR()
   } else if(max(curv.2$standard) <= max(check.2$standard)){
     ggplot2::ggplot()+
       ggplot2::geom_smooth(formula = y~x, data = curv.2, ggplot2::aes(x = standard, y =
AUC_ppm),
                  method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
       ggplot2::geom_point(data = check.2, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Check"), size = 4, shape = 21, alpha = .5) +
       ggplot2::geom_point(data = curv.2, ggplot2::aes(x = standard, y = AUC_ppm, fill =
"Curve"), size = 3, shape = 24, alpha = .5) +
       ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
         shape = c(21,24),
         size = c(3,3)))) +
```

```
      ggplot2::geom_text(data = check.2, ggplot2::aes(x = (((max(standard)-
min(standard))*.25)+min(standard)),
                                 y = (((max(AUC_ppm)-min(AUC_ppm))*.75)+min(AUC_ppm)),
                                 label = paste(form2.2, r.squared.2, sep = "\n"))) +
      ggplot2::ggtitle(paste0("Linear Standard Curve for ", curv.2$File_Name)) +
      ggplot2::xlab("Standard ppm") +
      UNR()
   }
  }else if(is.null(file) & is.null(sample) & std.curve == T & method == "log"){
   curv <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CURVE")) %>%
    dplyr::mutate(standard = numextract(Sample), .before = 3)
   check <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CHECK") )%>%
    dplyr::mutate(standard = numextract(Sample), .before = 3)
   asdf <- summary(lm(log(AUC) ~ log(standard), data = curv))
   Y <- as.numeric(asdf$coefficients[1])
   M <- as.numeric(asdf$coefficients[2])
   R <- as.numeric(asdf$adj.r.squared)
   form <- data.frame(Y = Y, M = M, R = R)
   form1 <- paste(round(form$M, digits = 2), "* log(x) +", round(form$Y, digits = 2))
   form2 <- paste0("Y = ", "e^(", form1, ")")
   r.squared <- paste0("adjusted R^2 = ", round(form$R, digits = 4))
   if(max(curv$standard) > max(check$standard)){
    ggplot2::ggplot()+
      ggplot2::geom_smooth(formula = y~x, data = curv, ggplot2::aes(x = log(standard), y =
log(AUC_ppm)),
                method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
      ggplot2::geom_point(data = check, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Check"), size = 4, shape = 21, alpha = .5) +
      ggplot2::geom_point(data = curv, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill =
"Curve"), size = 3, shape = 24, alpha = .5) +
      ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
        shape = c(21,24),
        size = c(3,3)))) +
      ggplot2::geom_text(data = curv, ggplot2::aes(x = (((max(log(standard))-
min(log(standard)))*.25)+min(log(standard))),
                           y = (((max(log(AUC_ppm))-
min(log(AUC_ppm)))*.75)+min(log(AUC_ppm))),
                           label = paste(form2, r.squared, sep = "\n"))) +
      ggplot2::ggtitle("Log Standard Curve for All Samples") +
      ggplot2::xlab("log(Standard ppm)") +
      UNR()
   } else if(max(curv$standard) <= max(check$standard)){
    ggplot2::ggplot()+
      ggplot2::geom_smooth(formula = y~x, data = curv, ggplot2::aes(x = log(standard), y =
log(AUC_ppm)),
                method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
```

```
      ggplot2::geom_point(data = check, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Check"), size = 4, shape = 21, alpha = .5) +
      ggplot2::geom_point(data = curv, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill =
"Curve"), size = 3, shape = 24, alpha = .5) +
      ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
        shape = c(21,24),
        size = c(3,3)))) +
      ggplot2::geom_text(data = check, ggplot2::aes(x = (((max(log(standard))-
min(log(standard)))*.25)+min(log(standard))),
                            y = (((max(log(AUC_ppm))-
min(log(AUC_ppm)))*.75)+min(log(AUC_ppm))),
                            label = paste(form2, r.squared, sep = "\n"))) +
      ggplot2::ggtitle("Log Standard Curve for All Samples") +
      ggplot2::xlab("log(Standard ppm)") +
      UNR()
   }
  } else if(!is.null(file) & is.null(sample) & std.curve == T & method == "log"){
   curv.2 <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CURVE"), File_Name ==
file) %>%
          dplyr::mutate(standard = numextract(Sample), .before = 3)
   check.2 <- dplyr::filter(data, stringr::str_detect(toupper(Sample), "CHECK"), File_Name ==
file)%>%
           dplyr::mutate(standard = numextract(Sample), .before = 3)
   asdf.2 <- summary(lm(log(AUC) ~ log(standard), data = curv.2))
   Y.2 <- as.numeric(asdf.2$coefficients[1])
   M.2 <- as.numeric(asdf.2$coefficients[2])
   R.2 <- as.numeric(asdf.2$adj.r.squared)
   form.2 <- data.frame(Y = Y.2, M = M.2, R = R.2)
   form2.1 <- paste(round(form.2$M, digits = 2), "* log(x) +", round(form.2$Y, digits = 2))
   form2.2 <- paste0("Y = ", "e^(", form2.1, ")")
   r.squared.2 <- paste0("adjusted R", "^", 2,  sep = "", " = ", round(form.2$R, digits = 4))
   if(max(curv.2$standard) > max(check.2$standard)){
    ggplot2::ggplot()+
      ggplot2::geom_smooth(formula = y~x, data = curv.2, ggplot2::aes(x = log(standard), y =
log(AUC_ppm)),
                method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
      ggplot2::geom_point(data = check.2, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Check"), size = 4, shape = 21, alpha = .5) +
      ggplot2::geom_point(data = curv.2, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Curve"), size = 3, shape = 24, alpha = .5) +
      ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
        shape = c(21,24),
        size = c(3,3)))) +
      ggplot2::geom_text(data = curv.2, ggplot2::aes(x = (((max(log(standard))-
min(log(standard)))*.25)+min(log(standard))),
```

```
                              y = (((max(log(AUC_ppm))-
min(log(AUC_ppm)))*.75)+min(log(AUC_ppm))),
                              label = paste(form2.2, r.squared.2, sep = "\n"))) +
       ggplot2::ggtitle(paste0("Log Standard Curve for ", curv.2$File_Name)) +
       ggplot2::xlab("log(Standard ppm)") +
       UNR()
   } else if(max(curv.2$standard) <= max(check.2$standard)){
     ggplot2::ggplot()+
       ggplot2::geom_smooth(formula = y~x, data = curv.2, ggplot2::aes(x = log(standard), y =
log(AUC_ppm)),
                    method = lm, se = T, lwd = 1, color = "red", fullrange = T, fill = "dodgerblue") +
       ggplot2::geom_point(data = check.2, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Check"), size = 4, shape = 21, alpha = .5) +
       ggplot2::geom_point(data = curv.2, ggplot2::aes(x = log(standard), y = log(AUC_ppm), fill
= "Curve"), size = 3, shape = 24, alpha = .5) +
       ggplot2::scale_fill_manual(name = "Standard", values = c("grey70", "darkblue"), guide =
ggplot2::guide_legend(override.aes = list(
         shape = c(21,24),
         size = c(3,3)))) +
       ggplot2:: geom_text(data = check.2, ggplot2::aes(x = (((max(log(standard))-
min(log(standard)))*.25)+min(log(standard))),
                              y = (((max(log(AUC_ppm))-
min(log(AUC_ppm)))*.75)+min(log(AUC_ppm))),
                              label = paste(form2.2, r.squared.2, sep = "\n"))) +
       ggplot2::ggtitle(paste0("Log Standard Curve for ", curv.2$File_Name)) +
       ggplot2::xlab("log(Standard ppm)") +
       UNR()
   }
  }
}
```

timeseries.peaks

```
timeseries.peaks <- function(){
  filelist <- list.files(pattern = c(".txt", ".TXT"))
  output.raw <- data.frame()
  print("Looping through Folder:")
  progress_bar <- txtProgressBar(min = 0, max = length(filelist), style = 3)
  for(i in 1:length(filelist)){
    setTxtProgressBar(progress_bar, i)
    b <- read.table(filelist[i], header = T, sep = "\t", fill = T, strip.white = T, check.names = F)
    data.1 <- dplyr::mutate(b, Sample = NA, .before = 1)
    names(data.1) <- c("Sample", "Test", "Time", "CO2")
    data.1 <- cbind(data.1, File = filelist[i])
    data.2 <- dplyr::filter(data.1, Test != "------------------------------------------------------------")
    data.3 <- data.2
    for(i in 1:nrow(data.2)){
      if(is.na(data.2[i,4])==T){
        data.3[i,1] <- as.character(data.2[i,2])
      } else {
        next
      }
    }
    data.4 <- na.omit(tidyr::fill(data.3, Sample, .direction = "down"))
    output.raw <- rbind(output.raw, data.4)
  }
  output.raw$Time <- lubridate::as_datetime(output.raw$Time)
  return(output.raw[, c(1,3,4,5)])
}
```

Plot.timeseries

```
Plot.timeseries <- function(data, file, sample = NULL, time.start = NULL, time.stop = NULL){
  UNR <- function(){
    ggplot2::theme(text = ggplot2::element_text(color = "black", size = 15),
          plot.title = ggplot2::element_text(face = "bold", color = "darkblue", margin =
ggplot2::margin(b = 15)),
          plot.subtitle = ggplot2::element_text(size = 10),
          axis.ticks = ggplot2::element_line(size = 1.5),
          axis.title = ggplot2::element_text(face = "bold", line = 2),
          axis.title.x = ggplot2::element_text(margin = ggplot2::margin(t = 10), color = "darkblue",
size = 15),
          axis.title.y = ggplot2::element_text(margin = ggplot2::margin(r = 10), color = "darkblue",
size = 15),
          axis.title.y.right = ggplot2::element_text(margin = ggplot2::margin(l = 15),color =
"grey50", size = 15),
          axis.text = ggplot2::element_text(color = "black"),
          axis.text.x = ggplot2::element_text(margin = ggplot2::margin(t = 15)),
          axis.text.y = ggplot2::element_text(margin = ggplot2::margin(r = 10)),
          axis.line = ggplot2::element_line(colour = "black"),
          axis.ticks.length = ggplot2::unit(2, "mm"),
          plot.caption = ggplot2::element_text(color = "black"),
          plot.background = ggplot2::element_rect(fill = "white"),
          plot.margin = ggplot2::margin(t = 10, r = 50, b = 10, l = 10),
          panel.background = ggplot2::element_rect(fill = "white"),
          panel.border = ggplot2::element_rect(fill = "NA", color = "darkblue", size = 1.5),
          legend.background = ggplot2::element_rect(color = "black", fill = "white"),
          legend.key =  ggplot2::element_rect(fill = "white"),
          legend.text = ggplot2::element_text(color = "black"),
          legend.position = "bottom",
          strip.background = ggplot2::element_rect(color = "blue", fill = "grey75", size = 2),
          strip.text.y = ggplot2::element_text(size = 13, face = "bold"),
          panel.grid = ggplot2::element_blank())
  }
  if(!is.null(file) & is.null(sample) & is.null(time.start) & is.null(time.stop)){
     file.plot <- dplyr::filter(data, File == file)
     ggplot2::ggplot(file.plot, ggplot2::aes(x = Time, y = CO2)) +
       ggplot2::geom_line(color = "blue") +
       ggplot2::ggtitle(paste0("Time Series Plot for:", " ", file)) +
       UNR()
  } else if(!is.null(file) & !is.null(sample) & is.null(time.start) & is.null(time.stop)){
     file.plot <- dplyr::filter(data, File == file)
     sample.plot <- dplyr::filter(file.plot, Sample == sample)
     ggplot2::ggplot(sample.plot, ggplot2::aes(x = Time, y = CO2))+
       ggplot2::geom_line(color = "blue")+
       ggplot2::ggtitle(paste0("File: ", " ", file, "\nSample: ", sample))+
       UNR()
```

```
  } else if(!is.null(file) & !is.null(sample) & !is.null(time.start) & is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    sample.plot <- dplyr::filter(file.plot, Sample == sample)
    start <- dplyr::filter(sample.plot, Time >= lubridate::as_datetime(time.start))
    ggplot2::ggplot(start, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSample: ", sample))+
      UNR()
  } else if(!is.null(file) & is.null(sample) & !is.null(time.start) & is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    start <- dplyr::filter(file.plot, Time >= lubridate::as_datetime(time.start))
    samples <- paste0(unique(start$Sample), collapse = ', ')
    samples.2 <- stringr::str_wrap(samples, width = 40)
    ggplot2::ggplot(start, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSamples: ", samples.2))+
      UNR()
  } else if(!is.null(file) & is.null(sample) & is.null(time.start) & !is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    stop <- dplyr::filter(file.plot, Time <= lubridate::as_datetime(time.stop))
    samples <- paste0(unique(stop$Sample), collapse = ', ')
    samples.2 <- stringr::str_wrap(samples, width = 40)
    ggplot2::ggplot(stop, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSamples: ", samples.2))+
      UNR()
  } else if(!is.null(file) & !is.null(sample) & is.null(time.start) & !is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    sample.plot <- dplyr::filter(file.plot, Sample == sample)
    stop <- dplyr::filter(sample.plot, Time <= lubridate::as_datetime(time.stop))
    ggplot2::ggplot(stop, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSample: ", sample))+
      UNR()
  } else if(!is.null(file) & is.null(sample) & !is.null(time.start) & !is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    start.stop <- dplyr::filter(file.plot, Time >= lubridate::as_datetime(time.start) & Time <=
lubridate::as_datetime(time.stop))
    samples <- paste0(unique(start.stop$Sample), collapse = ', ')
    samples.2 <- stringr::str_wrap(samples, width = 40)
    ggplot2::ggplot(start.stop, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSamples: ", samples))+
      UNR()
  } else if(!is.null(file) & !is.null(sample) & !is.null(time.start) & !is.null(time.stop)){
    file.plot <- dplyr::filter(data, File == file)
    sample.plot <- dplyr::filter(file.plot, Sample == sample)
```

```
    start.stop <- dplyr::filter(sample.plot, Time >= lubridate::as_datetime(time.start) & Time <=
lubridate::as_datetime(time.stop))
    ggplot2::ggplot(start.stop, ggplot2::aes(x = Time, y = CO2))+
      ggplot2::geom_line(color = "blue")+
      ggplot2::ggtitle(paste0("File: ", " ", file, "\nSample: ", sample))+
      UNR()
  }
}
```