Basic Full Waveform LiDAR Tools

Mark Warren
Emma Carolan
NERC-ARF-DAN

1 Introduction

This practical aims to introduce you to a number of full waveform LiDAR tools in order to view and access the data stored in the LAS1.3 files. As well as those developed in-house by NERC-ARF-DAN and PML, we shall introduce some other freely available tools that may be of use. We will also show you one method that can be used to extract extra information from the full waveform files. This document assumes that you have a working knowledge of LiDAR and a basic understanding of the LAS1.3 file specifications.

2 PML developed software

This demonstration document covers the basic use of two libraries developed at Plymouth Marine Laboratory. The first one is a C++ library, based on the LAS13Reader developed by Milto Miltiadou, but with some additional functionality. There are also Python bindings for this library that allow it to be called from Python scripts, utilising the speed of C++ with the simplicity of Python. The second is a Python based tool which extracts waveforms from a bounded area within a LAS1.3 file, writing them into ASCII files.

3 Background information

The following sections are included for background information and if you are doing this tutorial using the NERC-ARF virtual machine then you can skip the installation as this has already been done for you. Please skip forward to section 4.3.

3.1 Las13reader C++ and Python

The development of the C++ Las13reader is based on the code from Milto Miltiadou. The library can be downloaded from:
https://github.com/pmlrsg/arsf_tools/tree/master/las13

The full install of the library consists of the C++ las13reader code, the Python bindings created from SWIG, and the las13.py library built upon the Python bindings to give a more intuitive Python interface to work with. This demonstration will only cover the active use of the Python las13.py, library but to understand the underlying workings we shall start with an introduction to the classes within the C++ code.

3.1.1 Pulse

One of the main classes which holds data is the Pulse class. This contains the information related to individual pulses, such as the sampled waveform intensities, locations, sampling rate, point classifications etc.

3.1.2 PulseManager

The PulseManager is the class that is utilised for storing Pulses in. It also manages header information from the LAS1.3 file. Any operation that returns a result from the library will most likely return a PulseManager.

3.1.3 Las1_3_handler

The final class to introduce here is the Las1_3_handler class. This is the class which actually deals with the LAS1.3 file and is the main user interface to the file, including search functions that allow to return pulses with a particular classification value or pulses that are found within particular bounds. There is also the possibility to just read the file sequentially and return a user defined chunk of pulses. All of these functions return the pulses within a PulseManager object.

3.1.4 Las13.py

The Python library contains the functionality of the C++ library but with some added extras. The library contains a class, las13, that is essentially a wrapper for the Las1_3_handler class. It also contains a couple of extra functions that allow plots of the waveforms to be produced with very little user effort.

4 Exercise

In this section we will build the (Python) libraries and do some basic manipulations on a LAS1.3 file. This will require the Python, SWIG and C++ compiler distributions to be installed before hand. If not already installed they can be downloaded from:
Python: http://www.python.org/downloads/
SWIG: http://www.swig.org/
C++ compiler: https://gcc.gnu.org/

You will also require the PyLab and MatPlotLib Python libraries to be installed.

4.1 Linux instructions to build las13 library

Assuming the above development packages have been installed, download the las13 code (las13reader, swig and las13.py files) from:
https://github.com/pmlrsg/arsf_tools/tree/master/las13

Go into the swig directory and run:

```
make
```

This will create two files that are required: las13reader.py and _las13reader.so. Both of these need to be added to your Python path, i.e. move them into a suitable directory already within your Python path or export the current directory (note this will only remain in your Python path for this session):

```
export PYTHONPATH=your-directory-path-to-the-build:$PYTHONPATH
```

You should also move the las13.py into the same directory or another within your Python path. You should now be able to access the las13 library within your Python environments.

4.2 Windows instructions to build las13 library

The library has been successfully built using the MinGW32 environment on a computer running Windows 7. In addition to the above packages you need to install MinGW32. Then make sure you install SWIG and GCC through MinGW32.

4.3 View the help

In Linux, start up ipython (an interactive python session) with the following command from a terminal:

```
ipython
```

Else in Windows start up the 'cmd' prompt and type the full path to the executable:

```
C:\Python27\python.exe
```

This starts a python session. The following commands should now be the same on either Linux or Windows operating systems. We start by importing the library. If you have not added the directory containing the library files to your Python path, you can just copy the library files into your current directory (where you are running the Python command from). The current directory is always in the Python path:

```
import las13
```

This should output nothing if it has worked correctly. Now let us view the help for this library:

```
help(las13)
```

This will list the contents of the library (available functions, classes etc) with a brief description of how to use them / what they do.

4.4 Read and examine a pulse

Now that we have tested that the library has imported we can use it to do some manipulations of a LAS1.3 file. Throughout the following we shall use the example file LDR120726_094707_4.LAS. Firstly, if not already, import the las13 library in the python session:

```
import las13
```

Now we need to open the file. To do this we create a las13 object, this is defined in the las13.py library. The object requires the filename of the LAS1.3 file that we wish to open. **Note you need to include the path to the file** - in this case we assume the file to be in the current directory. Create a las13 object named 'las':

```
las=las13.las13("LDR120726_094707_4.LAS")
```

If we want to remind ourselves what functions 'las' contains we can use the Python command 'dir':

```
dir(las)
```

The output should look like the following, most of these functions will be described through this tutorial:

```
>>> dir(las)
['__doc__', '__init__', '__module__', 'get_pulse_info', 'plot_all_pulses',
'points_in_bounds', 'points_with_classification', 'quick_plot_pulse', 'read
_like_book', 'reader', 'tidy', 'waveform']
```

The first thing we need to do is retrieve some pulses from the file. We shall get all the pulses that fall within the area defined by the bounds 240166 240165 483292 483291 (N,S,E,W). To do this we can use the 'points_in_bounds' function. We can view the help for this function:

```
help(las.points_in_bounds)
```

which states that we need to pass a list describing north, south, west and east bounds, i.e.:

```
points=las.points_in_bounds([240166,240165,483291,483292])
```

This returns a PulseManager object called 'points'. We can query this PulseManager to see how many pulses it contains:

```
points.getNumOfPulses()
```

in this case it returns 2 - so we have 2 waveforms in this 1^2 m area. Let us examine the first one. We can access it like so:

```
p=points[0]
```

and use the 'get_pulse_info' function to return some specific information based on a keyword. The list of possible keywords are:

'time' = time [float]
'nreturns' = number of returns [int]
'nsamples' = number of samples in waveform [int]
'origin' = origin of waveform (first sample position) [list]
'offset' = vector to traverse the waveform [list]
'scanangle' = scan angle rank [int]
'classification' = classification of waveform [int]

'returnlocs' = offset in ns from origin to discrete return [list]
'disint' = discrete return point intensities [list]

To find the time of the pulse we can run:

```
las.get_pulse_info(p,"time")
```

```
>>> las.get_pulse_info(p,'time')
380962.59289583645
>>>
```

and similarly for all the other keywords. Now run the command for all the different keywords on both pulses.

4.5 Plot the waveform

We can also inspect the waveform for each of these pulses. One way to do this is extract the data into a more usable form such as a Python dictionary. We can use the waveform command to do this:

```
wave=las.waveform(p)
```

will extract the waveform data from pulse p and return it as a dictionary object with the keys:

'x' = X position of each sample [list]
'y' = Y position of each sample [list]
'z' = Z position of each sample [list]
'intensity' = intensity of each sample [list]

We could print out the intensity sample at position 50 like so:

```
print wave["intensity"][50]
```

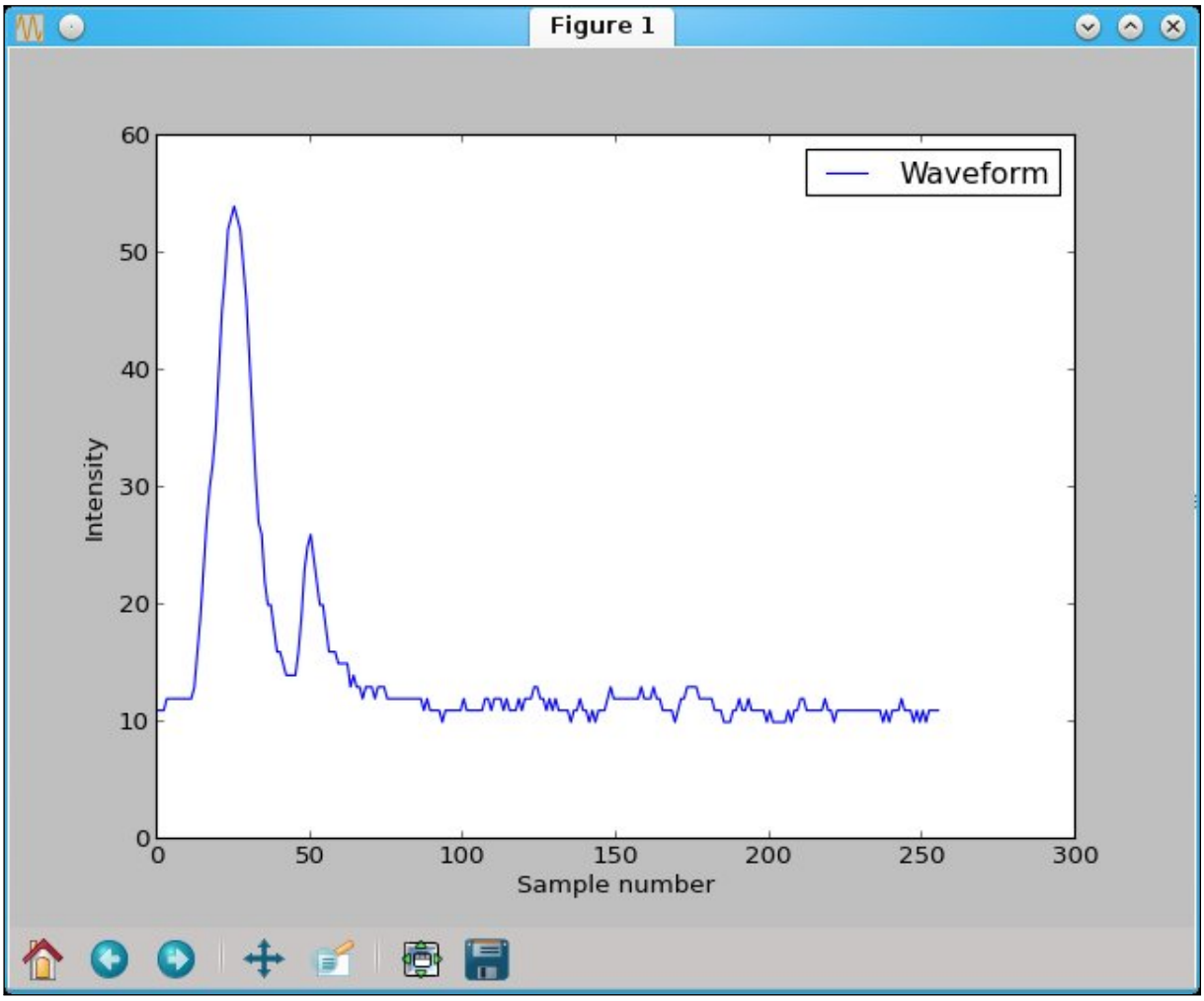
which should print the value '26'. To check the full 256 samples are present, we could print out the length of one of the lists:

```
print len(wave["intensity"])
```

We could use this data for more complex procedures, for example create a 3-dimensional plot of the intensity, but here we will just plot the intensity waveform using a built-in function to the las13 library:

```
las.quick_plot_pulse(p)
```

This should create a plot as in the below figure. The title can be specified and there is also the possibility to print to file rather than screen.



If all waveforms from a PulseManager are to be plotted then one can use the plot_all_pulses function, passing a PulseManager and output PDF filename. All the plots are added into the same PDF file, the plot title given by the GPS time of the pulse. Try that now:

```
las.plot_all_pulses(points,"pulses.pdf")
```

5 FWExtract Python

The full waveform extraction tool written by NERC-ARF-DAN to extract individual waveforms from LAS1.3 files. It requires the user to specify an area of interest (N,S,E,W) and it creates an ascii file containing the waveform information for each point within the specified area (one file for each point).

In order to use this tool you will need to have python installed on your computer. If you already have python then you can skip this section.

Download: http://www.python.org/downloads/

You will also need to add the directory of your Python distribution to your PATH environment variable.

More information: http://docs.python.org/2/using/windows.html#excursus-setting-environment-variables

The fw_extract.py script can be downloaded from the NERC-ARF github page: https://github.com/pmlrsg/arsf_tools

6 Exercise - use fw_extract to extract some waveform data

The full waveform extraction tool does not have an associated GUI so needs to be run from the command line.

To run the extraction tool under Linux type

```
fwf_extract.py <input_las_file.LAS>
```

Alternatively to run under Windows precede the command with 'python', e.g.

```
python fwf_extract.py <input_las_file.LAS>
```

Run the command using the example file LDR120726_094707_4.LAS. When prompted, enter the following limits as the area to extract data from: 240166 240165 483292 483291

For example:

```
emca@pmpc1323-bash:~/scratch/workshop/workshop_docs/2014/fwf_example<946>fwf_extract.py LDR120726_094707_4.LAS

Output directory was not specified, will use current directory

Enter limits of the area to extract:
Enter Max North (Range 239950.041000-240469.431000)240166
Enter Min North (Range 239950.041000-240166.000000)240165
Enter Max East (Range 482995.250000-483507.803000)483292
Enter Min East (Range 482995.250000-483292.000000)483291
Starting to process 779869 points
Will output ASCII files to  /data/weddell1/scratch/emca/workshop/workshop_docs/2014/fwf_example/
Number of extracted waves: 3
```

You can also specify an area of interest on the command line using the '--area' argument, and an output directory using the '-o' argument.

`fwf_extract.py <input_las_file.LAS> -area <North><South><East><West> -o <output_directory >`

For example:

```
emca@pmpc1323-bash:~/scratch/workshop/workshop_docs/2014/fwf_example<947>fwf_extract.py LDR120726_094707_4.LAS
-o waveforms/ --area 240166 240165 483292 483291
Starting to process 779869 points
Will output ASCII files to  /data/weddell1/scratch/emca/workshop/workshop_docs/2014/fwf_example/waveforms/
Number of extracted waves: 3
```

Note the output folder needs to exist first.

You can examine the extracted waveform files, which will be of the format waveform_<GPS_seconds>_<GPS_microseconds>_<return_type>.txt, e.g waveform_380962_592896_1.txt. In this case the file contains information relating to a point at GPS time 380962.592896, return number 1.

Note that there is also a file called waveform_380962_592896_2.txt, which contains information relating to return 2 of the same pulse. The sampled waveform data will be the same for both of these points.

The following describes the values represented in the ascii file:

Point:	XYZ position of associated discrete point
Return Number:	Return number of associated discrete point
Number of returns for this pulse:	Total Number of returns for this pulse
Time:	GPS time (Seconds of week) of associated discrete point
Scan Angle:	Scan angle of associated discrete point
Classification:	Classification of associated discrete point
Temporal Sample Spacing:	Sampling rate in nano seconds
AGC gain:	Automatic Gain Control
Digitiser Gain/Offset:	Digitiser gain and offset. Used to convert the raw digitiser value to an absolute digitiser voltage
No. of Samples:	Number of waveform samples
Sample Length:	Sample separation in metres
Return Point Location:	Offset in nano seconds from first digitised point to the associated discrete point
Point in Waveform:	Offset in metres between first digitised point and the associated discrete point
X/Y/Z Offset:	XYZ offset in metres between samples
Intensity X Y Z:	List of the sampled waveform data

7 Alternative tools

The following is a list of some additional tools you may find useful that support full waveform LiDAR data. Some are freely available but others require a licence to be purchased.

PulseWaves: http://rapidlasso.com/category/pulsewaves/
Fityk: Windows - https://github.com/wojdyr/fityk/downloads/
Linux - http://download.opensuse.org/repositories/home://wojdyr/
SPDLib: https://bitbucket.org/petebunting/spdlib
OPALS: http://geo.tuwien.ac.at/opals/html/index.html
TerraScan: http://www.terrasolid.com/products/terrascanpage.html

In the following exercise we shall use the fityk package to fit peaks to a waveform and extract a new 'return'.

8 Exercise - extract new return

In this exercise we shall use the fityk package together with fw_extract to identify a possible 'return' from a lidar pulse that was not detected by the discrete system. Create a folder called 'waveforms_2' and run the extraction tool again on a second area 240283 240282 483269 483268, using the '-o' and '-- area' parameters.

8.1 Fityk

Fityk will be used to visualise the returned waveform and to extract additional information from the waveform data. This is done by trying to identify peaks in the waveform.

Open fityk under Linux by typing:

```
fityk
```

Under Windows, navigate to the fityk folder and double click on the fityk application.

Go to Data -> Load File. Navigate to the directory containing the extracted waveforms and select file waveform_380962_592896_1.txt.

Select column 0 for x and column 1 for y. Click 'Replace @ 0' and then click 'Close'

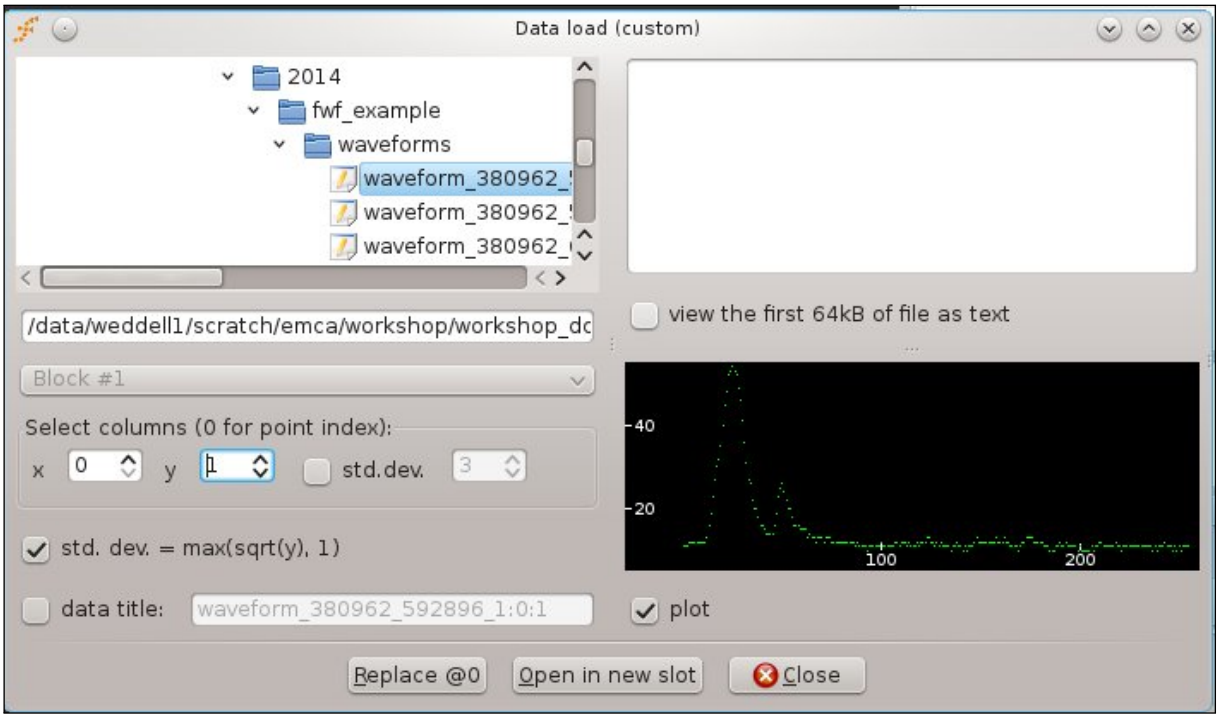


Figure 1: Fityk load screen

Tick the 'line' option on the right hand side. Note here that there are two distinct peaks visible. These correspond to discrete returns 1 and 2.

The next step is to manually fit a baseline and subtract the baseline to remove background noise.

To do this select 'Baseline Mode' from the tool bar (third icon from the left). The baseline determines where you think the noise threshold should be. Click ONCE on the graph to insert a flat baseline. Where you click determines the position of the baseline.

If you want to remove the baseline, go to GUI -> Baseline Handling -> Clear Baseline.

Subtract the baseline by selecting 'Strip Baseline' (icon to the left of Gaussian/Spline - see Figure 6)

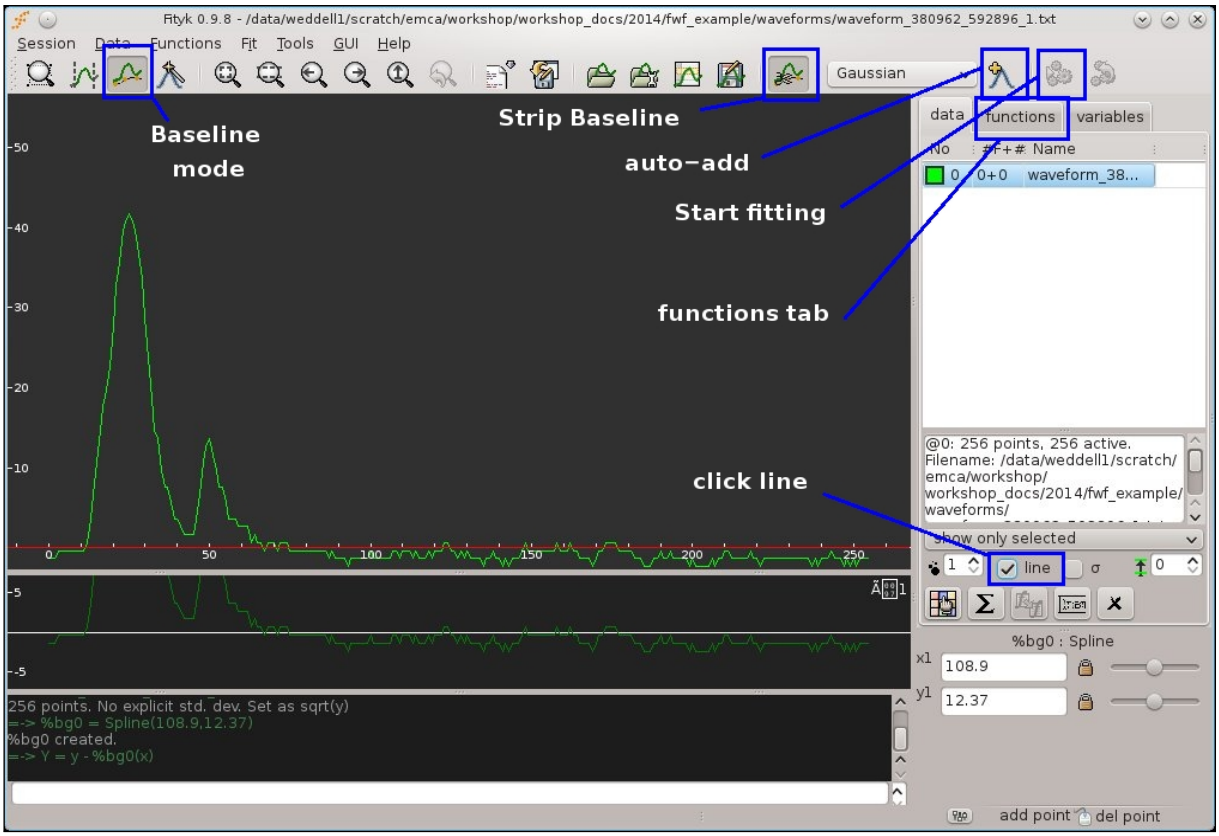


Figure 2: Fityk icon identification

Click on 'auto-add' to add peaks. Then select 'Start fitting'. Each of the fitted functions generates a number of values e.g. peak center, peak area,

height, FWHM, intensity width. You can see these values by selecting the 'functions' tab.

The value given as the center of the peak should correspond approximately to the Return Point Location value given in the extracted waveform ascii file.

To clear the window click on Session -> Reset.

Now load file waveform_380961_552645_1.txt from waveforms_2 folder in a similar manner. In this case we are looking at a single return (Number of returns for this pulse is 1).

Note that a second peak can be distinguished which was not detected by the discrete system. As before, we can fit peaks to this waveform and use the information extracted to create an additional discrete point corresponding to the second visible peak.

In order to work out the position of the new point, we use the center of the peak and the X,Y,Z offset values from the ascii file.

$$x_i = x_0 + (x_t) * (r)$$
$$y_i = y_0 + (y_t) * (r)$$
$$z_i = z_0 + (z_t) * (r)$$

where x_0 is the anchor point (first X sampled point) , x_t is the X offset and r is the return point location (center of peak).

Using the example shown here, the values we get are:

$$x = 483268.878$$
$$y = 240282.105$$
$$z = 85.226$$

Note that you may get slightly different values depending on where you place your baseline and on how many times you fit your data.