Tutorial – Application of Image & Signal Processing to Ice Core Layer Count

by Kasia Staniszewska, RENR 690 – Winter 2021

Video seminar: http://tinyurl.com/mv690/seminars/imagej
Data: http://tinyurl.com/mv690/seminars/imagej/data

BACKGROUND

Ice cores contain valuable proxy information about the past including records of past climate, dust storms, atmospheric carbon dioxide concentrations, and historic contaminant emissions. One way to establish context for the environmental and geochemical data we can obtain from an ice core record is by constructing an age profile for the ice core. Counting annual layers of alternating dense summer and loose winter ice is one method of determining the approximate age of an ice core. In this lab we will image and signal process a 51 cm segment of an ice core from the Columbia Icefield, Alberta. The ice core has been imaged at the Canadian Ice Core Archive on a Line Scanner which allows light through a slice of ice core, and measures light scatter intensity – recorded by an 8-bit gray-value. The light scatter intensity is a measure of ice density and particle interference, which varies seasonally, consisting of couplets of (dark) summer and (bright) winter precipitation. We will use the software ImageJ to clean and enhance the image. Then, we will export 1 dimension of data (gray-value with depth) from ImageJ into R where we will build and test a Blackman Windowed Sinc Filter, and peak counting function to count annual layers contained in the ice core segment.

Figure 1. an example of an ice core, extracted at Mt. Logan by the University of Alberta Canadian Ice Core Archive crew: Image available at: https://www.folio.ca/the-ice-cores-cometh/

STAGE 1 – IMAGE PROCESSING IN IMAGEJ

1. Download the open access software ImageJ at https://imagej.nih.gov/ij/download.html
2. Run the program. The program looks like a small window:

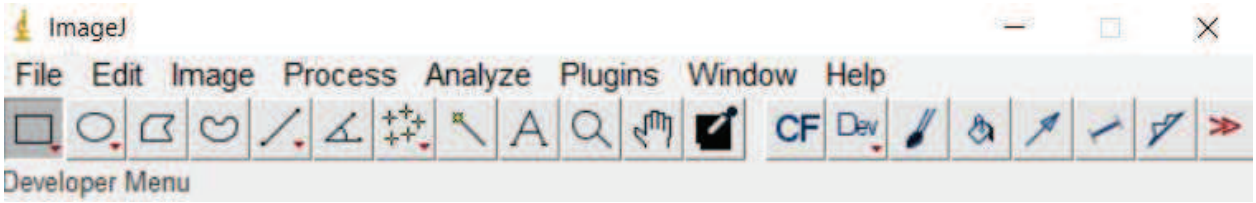

Figure 2. The ImageJ environment

3. Open the image of the Ice Core Segment attached with the lab

> File > Open… OR drag image from your file folder into the ImageJ Dialog Box

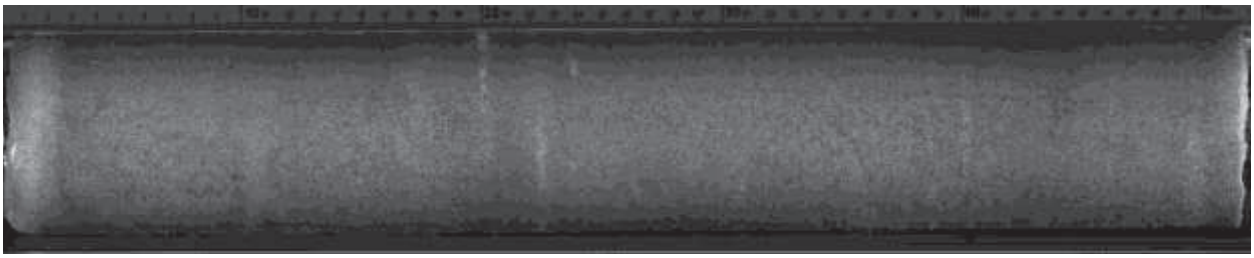

Figure 3. The Ice Core Image from the Line Scanner – our starting point for image processing. The top of the core (depth = 0), is on the left. The bottom of the core (depth = 51.5 cm) is on the right.

4. Make a note of how many centimeters versus pixels there are in the long axis of the ice core. The ruler is located along the top of the core. This is the pixel to depth conversion: 51.5 cm = 10260 pixels. This means 1 pixel represents 0.0000502 m
5. Use the Free Draw tool to delete the purely black area at the very top and very bottom of the ice core. This region is not real data, and we do not want to include it in our analysis. In an 8-bit image this will set the deleted area to "black = 0".

Free Draw Tool > Delete

6. We will next have to change the image type from 8-bit (range of 0 – 255) to a 32-bit image to change the deleted area from a value of 0, to a value of NaN.

> Image > Type > 32-bit

7. Use the wand tool to select the deleted areas (one at a time), then set the black values to NaN.

> Process > Math > Set As… > Value: input **NaN**

6. Use the rectangle tool to select the area of the ice core with data to enhance the image contrast – avoid the ruler at the top and the empty space at the base of the image. A rectangle one-third the width of the core, stretched around the entire length of core will do.

Rectangle Tool > Select Core
> Process > Enhance contrast…

Figure 4. An example rectangular selection, and output after the Enhance Function

7. Again, use the rectangle tool to select an area one-third the width of the core, stretched across the entire length of core, then analyze and extract the gray-value profile. The ImageJ program will take

the average pixel gray value of a column of pixels within the rectangular selection, for the length of the core, and plot a pixel index (depth) versus gray value profile. We will import this profile into R. While looking at the Plot Profile, check for values of pure white or black (255 or 0), these should not be present.

Rectangle Tool > Select Core
> Analyze > Plot Profile > Data>> > Save Data… > save CSV Save As > Values.csv

We're moving on from ImageJ but while you're here, explore the Image, Process, Analyze, and Plugins tabs to get an idea of what ImageJ can do. For example, you can upload a photo, then under ">Process" test the filtering options for smoothing or enhancing the photo, under ">Process>Math" you can brighten, darken, transform, find maxima, minima, averages. Find an image with cells or clusters and try the thresholding function! You can look at the distribution of values along the core length by using ">Analyze>Histogram" (or by core width by first rotating using ">Image>Transform")

Like R, ImageJ is highly customizable. You can install custom plugins (like a package in R) by downloading the plugin into your ImageJ plugins folder. There's a whole community of people who write plugins for ImageJ (mostly biologists) and a whole suite of tutorials on YouTube.

STAGE 2 – SIGNAL PROCESSING IN R

1. Read the gray-value versus pixel position data into R

```
core <- read.csv(file = "Values.csv "")
View(core) # Open csv viewer within the R GUI.
head(core) # view only the top of the spreadsheet
tail(core) # view only the base of the spreadsheet
```

Confirm that the number of pixels is still 10260 –your rectangle selection may not have spanned the entire core length – jot down the index of the last pixel. You'll need this number for the filter!

```
core$Gray_Value[is.na(core$Gray_Value)] <- 128 # Remove NA values by plugging in
# the average  value of an 8-bit image: 128.
core$Gray_Scaled <- scale(core$Gray_Value) # Normalize gray-value and add it
# as a  third column  in your data frame.
```

2. Turn the index into a vector that represents core depth, and add it to your data frame as column 4.

```
core <- cbind(core, as.numeric(rownames(core)))  # The function rownames()
# collectes the index of our file. The function as.numeric() ensures this is a
# number, and in the function cbind() we specify the file we want to bind this
# new vector to.
colnames(core)[4] <- "indX"  # The function colnames(variable)[column number]
# will rename the column to text of our choice, i.e., "indX".
core$Depth_m <- core$indX*0.0000502 # This is the conversion between pixel and
# depth we calculated in ImageJ. (Stage 1 step 4).
```

3. Check out your work so far by plotting Scatter Intensity (core$Gray_Scaled) by depth:

```
plot(core$Depth_m~core$Gray_Scaled, ylim=c(0.51,0), xlim=c(-5,5),
type="n", ylab = "Depth (m)", xlab = "Scatter Intensity (z-score of Gray
Value)", xaxs = "i", yaxs="i")
title(main = "Normalized Scatter Intensity Profile")
points(core$Depth_m~core$Gray_Scaled, type="b", pch=16, cex=0.1, col="black")
```

You'll notice the profile is very noisy. Can you tell how many up-down patterns (annual layers) there are? Jot down your best guess.

4. Next we make a Blackman Windowed Low Pass Filter to reduce noise in our signal.

```
fc = 0.####  # The frequency cut-off (fc) is the primary way in which we
# customize the filter, to eliminate noise above a set "frequency".
```

The value of fc is the inverse of your desired smoothing filter width divided by the inverse of your real world sampling width, i.e., ideally, given enough information, you would input a number or a polynomial equation representative of the expected annual layer width in real space. If you expect an annual layer to be greater than 5 cm wide, and you sampled at 50 µm wide pixels, you might set your fc to 0.001.

$$fc = \frac{1}{smooth\ width} \Big/ \frac{1}{sampling\ width} = \frac{1}{0.05\ m} \Big/ \frac{1}{0.00005\ m} = 0.001$$

Explore values of fc between 0 and 0.5. If fc ~ 0.5 there will be minimal to no filtering; i.e., most noise will pass through. If fc < 0.00005, the resulting signal will essentially be a straight line, i.e., all signal will be eliminated. It is very important to consider the value of fc carefully. Too strong a filter cut-off will eliminate the desired signal, but too weak a filter (e.g., fc=0.5) will have no effect.

```
N = 8 # value related to the transition bandwidth of the filter – a low value
# (8) will make the filter strict – frequencies around your frequency cut-off
# won't pass through the filter. Deviate from 8, plot to test its effect.
n = seq(1, 10260, 1) # n, pixel index as a sequence from 1:10260, by a step of 1.
# Note that 10260 is the number of pixels in the original data check how many
# pixels you imported using str(core) or tail(core)

h = (sin(2*fc*(n-(N-1)/2))) / (2*fc*(n-(N-1)/2))
# The formula to compute the ideal Sinc filter.
w = 0.42 - 0.5*cos(2*pi*n/(N-1)) + 0.08*cos(4*pi*n/(N-1))
# The formula to compute a Blackman window, needed for discreet filters.
h = h * w # Multiply the Sinc filter function by the Blackman window function.
h = h / sum(h) # Normalize the result.
GS_sinc = convolve(core$Gray_Scaled, h) # Apply the filter to the normalized
# Gray Values. This step may take a few seconds.
core$GS_sinc <- as.vector(GS_sinc) # Add the filtered data to the data frame
```

5. Plot the filtered Gray Values.

```
plot(core$Depth_m~core$GS_sinc, ylim=c(10.24,0), xlim=c(-
6,6),type="n", ylab = "Depth (m) (pixel=50.2 µm)", xlab = "Relative Scatter Inten
sity (z-score)", xaxs = "i", yaxs="i")
title(main = "Sinc Filtered Normalized Scatter Intensity Profile")
abline(v=0) # add a vertical line at zero. You might expect peaks will cross 0.
points(core$Depth_m~core$GS_sinc, type="b", pch=16, cex=0.1, col=c("black"))
```

How does the output look? Do you see clear up and down patterns? If the signal is very noisy, return to stage 6 and decrease the value of fc, then re-run the code. If it is too smooth, increase the value of fc. Once you have a feel for how the filter responds to the value of fc, use `fc = 0.001` *(filter width of 5 cm)* and re-run the code.

6. Count the number of peaks with library(pracma) and the function findpeaks().

```
install.library("pracma")
library(pracma) # See how findpeaks() works with ?findpeaks.
findpeaks(GS_sinc, nups = 1, ndowns = 1, zero = "0", peakpat = "[+][-
]", minpeakheight = -
Inf, minpeakdistance = 1, threshold = 0, npeaks = 0, sortstr = FALSE)
```

The peak count is the index of the `findpeaks()` output. The last index, divided by 2, is the estimated number of years. The first column of the output is the peak amplitude, the second, third and fourth columns are the midpoint, beginning inflection point, and end inflection points' index. We have determined that any signal smaller than 10 cm is likely noise rather than an annual layer, so let's adjust `nups, ndowns, minpeakdistance` to consider changes in peak amplitude of more than 10 cm rather than one index step at a time. Remember, you can convert index into depth (meters) by multiplying by 0.0000502. (hint: try setting `minpeakdistance` = 1990).

Did you get an output of 4 peaks? Divide by 2 to get the number of years. This would mean there are 2 years of data in this 51.5 cm segment of ice core… but can you identify the peaks on the depth profile by converting index to depth again? Are any obvious peaks missed? Were any potentially fake peaks counted around the core beginning and end? How does this output compare to your estimate from Stage 2 Step 5?

To gauge if this output is realistic, we would want to compare the result to external information: glaciochemical data like $\delta^{18}O$, $\delta^{2}H$, chlorine concentration, or calibrate our age model using layers of known age (like volcanic ashes), or incorporate information about average annual accumulation rate.

Thank you for participating in this tutorial. If you have any questions of feedback, please feel free to reach out: katarzyn@ualberta.ca