**Magneto - Scale and Standardization Project**

Includes:

Final Report, Functions

By: Jessica Dobbin

Completed for: Natural Resources Canada

Supervising Professor: Dr. Wesley Burr

Trent Community Research Centre Project Coordinator: Brittany Finigan

Course Code: MATH 4851

Course Name: Community-Based Research Project (Sc)

Completion Date: April, 2023

Project ID: 6010


Suite 3.10, Trent University Student Centre

1600 West Bank Drive

Peterborough, ON K9L 0G2

Phone: (705) 748-1093

Email: tcrc@trentu.ca

Website: trentu.ca/tcrc

# Magneto: Scale Standardization

A Community-Based Research Project
Completed by: Jessica Dobbin
In collaboration with Natural Resources Canada
Supervised by Dr. Wesley Burr
And Coordinated by Brittany Finigan of the Trent Community Research Centre

## Contents



## List of Figures

# 1 Background

## 1.1 Magnetograms

Continuous daily observations of the local magnetic field fluctuations are recorded in magnetograms. Historically, these have been recorded on paper, then photographed and preserved on microfilm [1]. Microfilm corresponding to the Agincourt Observatory for the December 1944 - December 1965 time frame were recently scanned at Trent University. It is this collection of microfilm from which the test image used originates.

## 1.2 Magneto and TIS Algorithm

There has been an ongoing effort to digitize these scanned images. One such effort resulted in the magneto package [2] which uses the Trace Identification by Separation (TIS) algorithm to take a file containing a scanned image and construct a plot of the corresponding time series.

# 2 Motivation

While previous work has proposed various solutions to the ongoing digitization effort, there remains the issue of units. In order to interpret the digitized magnetograms, it is necessary to know the scale they were plotted on and the millimeter to pixel conversion. The 1 *mm* to *y* conversion rate is often included in the associated documentation of the magnetograms or the magnetograms themselves. Therefore, there remains the issue of determining how many pixels it takes in a given magnetogram to represent 1 *mm*. This paper suggests one possible solution for finding this scale value.

# 3 Finding the Scale Value Using find_scale()

This section will first outline the functions necessary to build the find_scale() and their respective inputs. Once these have been clearly defined, there will be an overview of the inputs and structure of the find_scale() function which returns the sought after scale value. Finally, there will be some remarks on the chosen test image and the implementation of this process on it.

## 3.1 Functions

### 3.1.1 conversion()

The conversion() function takes an array (*imageArray*) as input and returns a matrix. This will take a raster array representing an RBG image to a matrix representing the image in grey-scale.

### 3.1.2 first_line()

The first_line() function takes a matrix (*imageMatrix*) and a value (*thresh*) as input and returns a linear model object. After a heuristic examination of the available data, the default for thresh was set to 0.4. This function takes *imageMatrix* and, starting from the top, finds the first position in every column which has a value that is at least as dark as *thresh*. A linear model is then fitted to these positions and returned to the user. The resulting line will trace the top of the spine of the ruler.

#### 3.1.3 second_line()

The second_line() function has six inputs: a matrix (*imageMatrix*), a string specifying the location of the contrasted image (*filePath*), two values (*thresh* which is passed to the first_line() and *thresh2*), a positive integer (*bound*), and a logical (*plot*) indicating whether a plot should be the output.

The function first calls first_line() and moves the fitted line down until it bisects with the ticks indicating increments of 1 *cm* and 5 *mm* on the ruler. This stopping point is determined by finding the first instance that the line has moved down at least $\frac{1}{5}$ of the maximum number of shifts specified by *bound*. Additionally, the difference between the 80% quantile of the original line and the shifted line must be at least *thresh2*.

If plot is true, the function returns a plot of the line found by first_line() in red and it's shifted counterpart in green overlaying the original image.

If plot is false, then the function returns a matrix where the first two columns are the x and y values of the shifted line, respectively. The third column is the value of the image matrix at the point specified by the x and y values.

#### 3.1.4 find_valley()

The find_valley() function takes three inputs: a vector (*lin*), a minimum distance between valleys (*dist*), and a minimum height depth cutoff (*height*).

The output of the function is a matrix where the first column is the index at which the downward peak/valley occurs and the second is the value at that point.

### 3.2 find_scale()

The find_scale() function takes seven inputs, four of which (*thresh1*, *thresh2*, *bound*, and *plot*) are used solely as inputs for second_line() and have already been specified. Two of the remaining arguments (*dist1* and *height1*) are likewise passed into find_valley(). The remaining input it the location of the file (*filePath*).

Using the tiff package [3], the image is imported using readTIFF() and, (using conversion()) the image is converted to a grey-scale one (where each pixel takes on a value from 0 (black) to 1 (white)) if it is not already in that form.

If plot is true, the plot (see Figure 1) from second_line() is given.

If plot is false, the function then takes the third column of the matrix from second_line(), and uses find_valley() to identify where the black tick marks are. Then the first difference is taken from the resulting vector of indices at which a tick occurs. From the resulting differences, the average is taken and divided into 5 to get the average number of pixels in 1 *mm*.

### 3.3 Implementation

As a test image, the ruler preceding the January 24, 1945 to January 25, 1945 magnetogram was used. Before being put through the algorithm, the image was altered in a photo editor by increasing the contrast (see Figure 2). This creates a greater disparity between the values of the pixels that, to the eye, appear black and white which results in an optimal implementation of find_scale(). This contrasted image is ideal for a number of other reasons. Firstly, it is a complete image of the ruler and so none of the ruler ticks are missing. Secondly, there are no visual abnormalities such as a shadow that will affect the function's efficacy. Thirdly, this image is properly oriented: horizontal with the ruler placed at a reasonable angle.

Once this image is in the aforementioned form, the function find_scale() can be used directly to determine the scale value (number of pixels in 1 *mm*). To ensure that this is a sensible value, it is possible for the user to use find_scale() to plot and then overlay lines spaced by the scale value that it returns (see Figure 3).

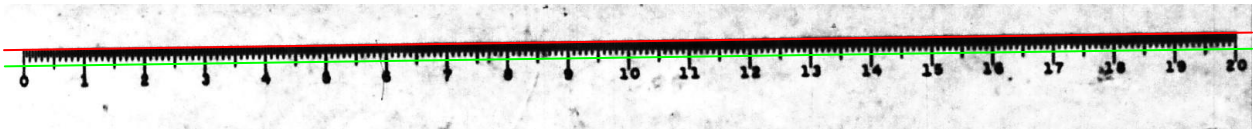

Figure 1: Plot from second_line()

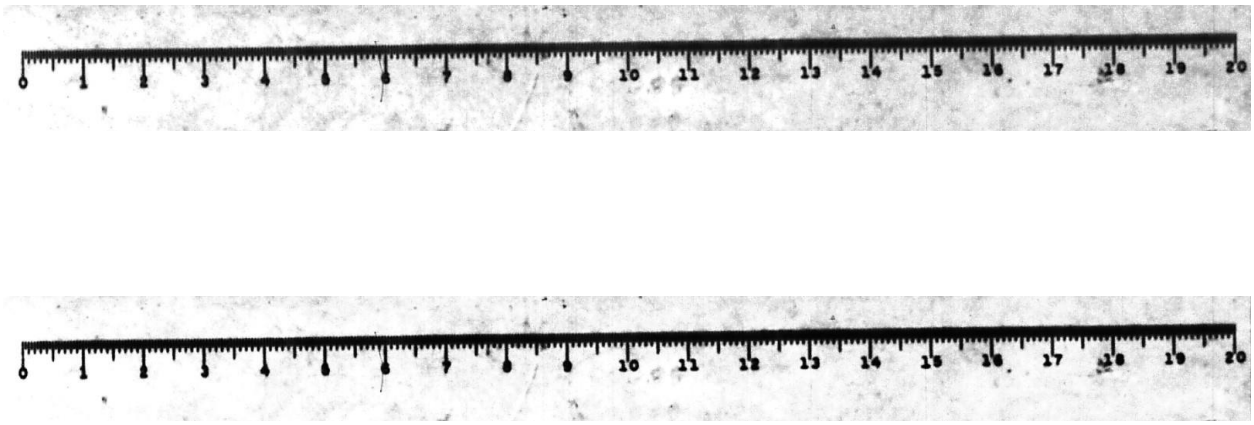

Figure 2: Original Image (above) and Contrasted Image (below)

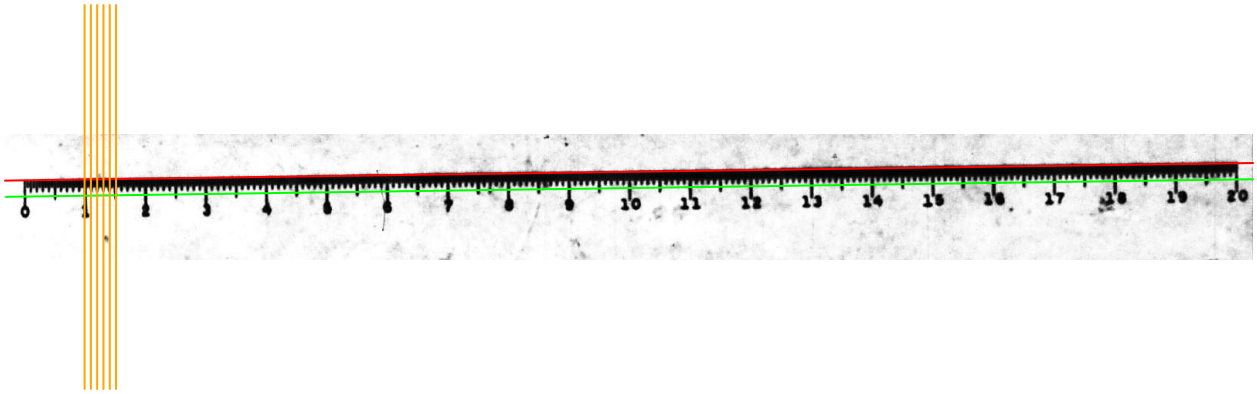

Figure 3: find_scale() plot overlaid with lines spaced by scale value

All plots are rendered using the raster package [4].

# 4 Next Steps

This was tested on a clean and complete image but not all ruler scans are in this condition. Further work can be done to deal with cases in which only a portion of the ruler is available or there is a shadow on the image.

While the default inputs for find_scale() are considered optimal for the contrasted test image, there is no reason to believe that these inputs are optimal for all images. For instance, the *bound* argument in find_scale() that is passed to second_line() has a default of 100 though there is no guarantee that the image has at least that number of rows. This is also an example of where defining sensible checks for each function would be useful to give the user clear and concise warnings and errors.

Additionally, the find_scale() function should have the ability to increase the contrast on the image without the user having to preprocess the image in a photo editor before applying the function.

This was tested on a clean and complete image but not all ruler scans are in this condition. Further work can be done to deal with cases in which only a portion of the ruler is available or there is a shadow on the image.

When these steps have been completed and the function fully tested, it is expected that this will be integrated into the TIS algorithm in the magneto package [2].

# 5 Appendix: Functions

## 5.1 conversion()

```
conversion <- function(imageArray){ if(length(dim(imageArray)) > 2) { newMat <- matrix(nrow
  = dim(imageArray)[1], ncol = dim(imageArray)[2]) for(i in 1:dim(imageArray)[1]){ for(j in
  1:dim(imageArray)[2]){ v <- imageArray[i, j, ] newMat[i,j] <- 0.299 * v[1] + 0.587 * v[2] +
  0.114 * v[3]
        }
      }
  } else { newMat <-
  imageArray
  } return(newMat)
}
```

## 5.2 first_line()

```
first_line <- function(imageMatrix, thresh = 0.4){
  test <- apply(imageMatrix, MAR = 2, FUN = function(x) {
    y <- min(which(x < thresh));
    y[!is.finite(y)] <- NA; y })

  x <- 1:ncol(imageMatrix) mod <-
  lm((nrow(imageMatrix) - test) ~ x) return(mod)
}
```

### 5.3 second_line()

```
second_line <- function(imageMatrix, filePath, thresh = 0.4, thresh2 = 0.007, bound =
                                    100, plot = TRUE){
   mod <- first_line(imageMatrix = imageMatrix, thresh = thresh) cIM_n <-
   ncol(imageMatrix) x <- data.frame(x = 1:cIM_n) y_hat <- predict(mod,
   newdata = x) fl_y_hat1 <- floor(y_hat) linMod <- cbind(1:cIM_n,fl_y_hat1)
   extract <- apply(linMod, MARGIN = 1, FUN = function(x){
   imageMatrix[x[2],x[1]]
   }) linModSums <- quantile(extract, prob = 0.80) lineSums <- vector(length =
   bound) flag <- TRUE for(k in 1:bound){ if(flag == TRUE){ adj_fl_y_hat <-
   floor(y_hat - k) linMod <- cbind(1:cIM_n, adj_fl_y_hat) extract <-
   apply(linMod, MARGIN = 1, FUN = function(x){ imageMatrix[x[2],x[1]]
         }) lineSums[k] <- quantile(extract, 0.80) if(k > (1/5*bound) && (abs(lineSums[k] -
         linModSums) >= thresh2)){ y_int <- -k flag <- FALSE
         }
      }
   } xs2 <- 1:cIM_n ys2 <- y_int + mod$coefficients[[1]] + mod$coefficients[[2]]*xs2

      adj_fl_y_hat <- floor(y_hat + y_int) linMod <- cbind(1:cIM_n,
      adj_fl_y_hat) extract <- apply(linMod, MARGIN = 1, FUN = function(x){
      imageMatrix[x[2],x[1]]
      })

 if(plot == TRUE){ plotRGB(brick(filePath))
      abline(mod, col = "red") lines(xs2, ys2, col =
      "green")
    } else
 {
    return(cbind(x = xs2, y = ys2, value = extract)) }
}
```

### 5.4 find_valley()

```
find_valley <- function(lin, dist = 14, height = 0.85) { peakHeights <-
  peakIndices <- vector() index <- 1:length(lin) first <- min(lin[1:(2 *
  dist)]) minst <- min(which(lin[1:(2 * dist)] <= height)) if
  (!is.integer(minst)) {
    minst <- 2 * dist
  } if (first <= height) { peakIndices <-
  c(peakIndices, minst) peakHeights <-
  c(peakHeights, first)
  }
  if (minst <= (dist + 1)) {
    minst <- minst + dist
  } for (i in minst:(length(lin) - dist)) {
    if (lin[i] == min(lin[(i - dist):(i + dist)]) && lin[i] <= height) { peakIndices <- c(peakIndices,
  i) peakHeights <- c(peakHeights, lin[i]) } } last <- min(lin[(length(lin) - dist + 1):length(lin)])
  if (last <= height) {
    lastIn <- which(lin == last) peakIndices <- c(peakIndices,
    lastIn[length(lastIn)]) peakHeights <- c(peakHeights, last)
  } peakInfo <- cbind(peakIndices, peakHeights)
  return(peakInfo)
}
```

### 5.5 find_scale()

```
find_scale <- function(filePath, thresh1 = 0.4, thresh2 = 0.007, bound = 100, dist1 = 14, height1 = 0.85,
                                  plot = FALSE){
  testim <- tiff::readTIFF(filePath) contestIM <- conversion(testim) twoLines <-
  second_line(contestIM, filePath, thresh1, thresh2, bound, plot) if(plot == FALSE){ lins <-
  twoLines[,3] val <- find_valley(lins, dist = dist1, height = height1) return(mean(diff(val[-
  c(1,length(val)),1]))/5)
  }
}
```

## References


[1] M. Weygang, “Historic magnetogram digitization,” Master’s thesis, Trent University., Peterborough, ON, 2019.

[2] B. Ott and W. Burr, *Magneto: The building blocks for digitization of magnetograms, or images alike*. 2022. Available: https://github.com/wesleyburr/magneto

[3] S. Urbanek and K. Johnson, *Tiff: Read and write TIFF images*. 2022. Available: https://CRAN.R-

project.org/package=tiff

[4] R. J. Hijmans, *Raster: Geographic data analysis and modeling*. 2022. Available: https://CRAN.Rproject.org/package=raster