Macroinvertebrate Community Index

Justin Pomeranz - J Pomz

April 25, 2017

Introduction

The macroinvertebrate community index (MCI) is a biotic index commonly used to rate the quality of fresh waters in New Zealand. Each freshwater taxon is given a tolerance value (1 to 10) based on sensitivity to organic pollution. These values are averaged in order to calculate a site score. When quantitative and qualitative abundance data are available, the QMCI and SQMCI can be calculated, respectively. More information on the MCI and its variants is available at: http://www.mfe.govt.nz/sites/default/files/mci-user-guide-may07.pdf

This tutorial will show you how to easily calculate the MCI and its variants using R, using the suite of packages known as the `tidyverse`.

Although the MCI was originally created to only assess organic pollution of freshwaters, it is often used in any monitoring of freshwaters in New Zealand. Another index has been developed to assess the effects of acid mine drainage, the AMDI. The underlying logic, and the calculation of these indices is similar, so we will also go over how to calculate the AMDI value. More information on the AMDI is available at: http://www.tandfonline.com/doi/abs/10.1080/00288330.2012.663764

The following tutorial is primarily designed for freshwater ecologists and practioners with little experience using the R language. With this in mind I have included lots of tips and tricks that I found useful when learning R. At the end of the tutorial I provide the raw code with minimal comments and in fewer steps for calculating the indices as a quick reference.

Prerequisites

At this point I would like to mention that this tutorial takes a "tidyverse" centric view. You should be able to follow along without needing too much background information. However, if you're not familiar with the pipe operator `%>%` I would recommend doing a quick google search just to familiarize yourself with it. Briefly, it takes the output from the function on the left side of the operator and puts it in as the first argument in the function on the right side of the operator. I find it helpful to think *then* whenever you see a `%>%` operator. It will make the more complicated multi-step code chunks lower down easier to understand.

If you would like to learn more about the tidyverse packages and philosophy, I highly recommend *R for Data Science* by Hadley Wickham and Garret Grolemund, available free at: http://r4ds.had.co.nz/.

Setup

To begin, let's setup our R studio session. If you don't already have these packages loaded, you will first need to run `install.packages("tidyverse")`. Then you will need to load these libraries:

```
library(tidyverse)
```

Data

There are two .csv files in my MCI github repository: bugs.csv, and indices.csv. Download these two .csv files and load them into your environment.

```
bugs <- read_csv("bugs.csv")
#> Parsed with column specification:
#> cols(
#>   site = col_character(),
#>   surber = col_character(),
#>   taxa = col_character(),
#>   number = col_integer()
#> )
indices <- read_csv("indices.csv")
#> Parsed with column specification:
#> cols(
#>   taxa = col_character(),
#>   MCI = col_integer(),
#>   AMDI = col_integer()
#> )
```

note 1: I use the `readr::read_csv()` command to read in my .csv files because it automatically reads them in as tibble objects. Tibbles are a useful data structure to work with, and are designed specifically to work with the tidyverse, but a full description is out of the scope of this tutorial.

note 2: your path names to the .csv files may differ. Make sure you know where you saved the files to (e.g. "C:/Desktop/bugs.csv"), and modify your path code within the quotes accordingly. The reason I can just type in `read_csv("bugs.csv")` is because I'm using Rprojects and saved my data directly into my Rproject directory. I highly recommend working in Rprojects, but again a run down on that is outside the scope of this tutorial. If you want to learn more, I recommend reading Chapter 8 in the *R for Data Science* book mentioned above.

The `bugs` data frame is a toy dataset and contains community composition data from 14 "sites". This data is made up, but based on actual surveys conducted on the west coast of the South Island of New Zealand. Always start out by taking a look at the data. It's good to have a mental image of the data structure. since the `read_csv()` command reads the data in as a tibble, simply typing the name of the object is a useful place to start.

Other useful commands that I often use are `base::names();base::str() and dplyr::glimpse()`. Try running these commads and see how the data output changes.

Three surber samples (area = 0.06 m^2) were taken at each site (s1, s2, s3). The number column represents the number of individuals of a given taxa found within a sample.

The `indices` data frame is simply a list of taxa with corresponding tolerance scores for the MCI and AMDI, respectively.

```
indices
#> # A tibble: 216 × 3
#>              taxa   MCI  AMDI
#>             <chr> <int> <int>
#> 1 Acanthophlebia     7    NA
#> 2          Acari     5    NA
#> 3      Acroperla     5    NA
#> 4         Aeshna     5    NA
#> # ... with 212 more rows
```

MCI

Step 1: combine data frames

The first thing we want to is combine these data frames so that the `bugs` object has the MCI and AMDI tolerance scores by taxa. In order to do this we will use the `dplyr::left_join()` function. Using the `left_join()` function ensures that all observations in the `bugs` data frame will be kept, even if no corresponding taxa observation occurs in the `indices` data frame.

```
bugs.index <- left_join(bugs, indices, by = "taxa")
```

I made sure that all of the taxa in `bugs` has a corresponding taxa in the `indices` data, but it's a good idea to check this whenever you're merging two data frames together. There are a number of ways to do this, but I generally use the `dplyr::setdiff()` function.

```
setdiff(bugs$taxa, indices$taxa)
#> character(0)
```

This shows us that there are no observations in `bugs$taxa` that *do not* occur in `indices`. Try running the following: `setdiff(indices$taxa, bugs$taxa)` (output not shown in this document).

This gives us a character vector containing all of the values of `indices$taxa` that *do not* occur in `bugs$taxa`.

Step 2: calculate MCI

The MCI can be calculated using presence / absence data with the following formula:

$$\sum_{i=1}^{S} a_i / S * 20$$

Where a_i is the tolerance value for taxa i, and S is the number of taxa within a sample.

Values of MCI can range from 0 to 200, but are rarely <50 or >150.

In order to calculate the MCI score for a site, we first need to calculate the score by surber sample. This is done with the `group_by()` command. This "groups" the data so that the subsequent `summarise` command is applied to each group independently.

```
mci.surber <-
  bugs.index %>% # our combined data fram
  group_by(site, surber) %>% # group the data
  summarise(MCI = (sum(MCI) / n()) * 20) # calculate the MCI
```

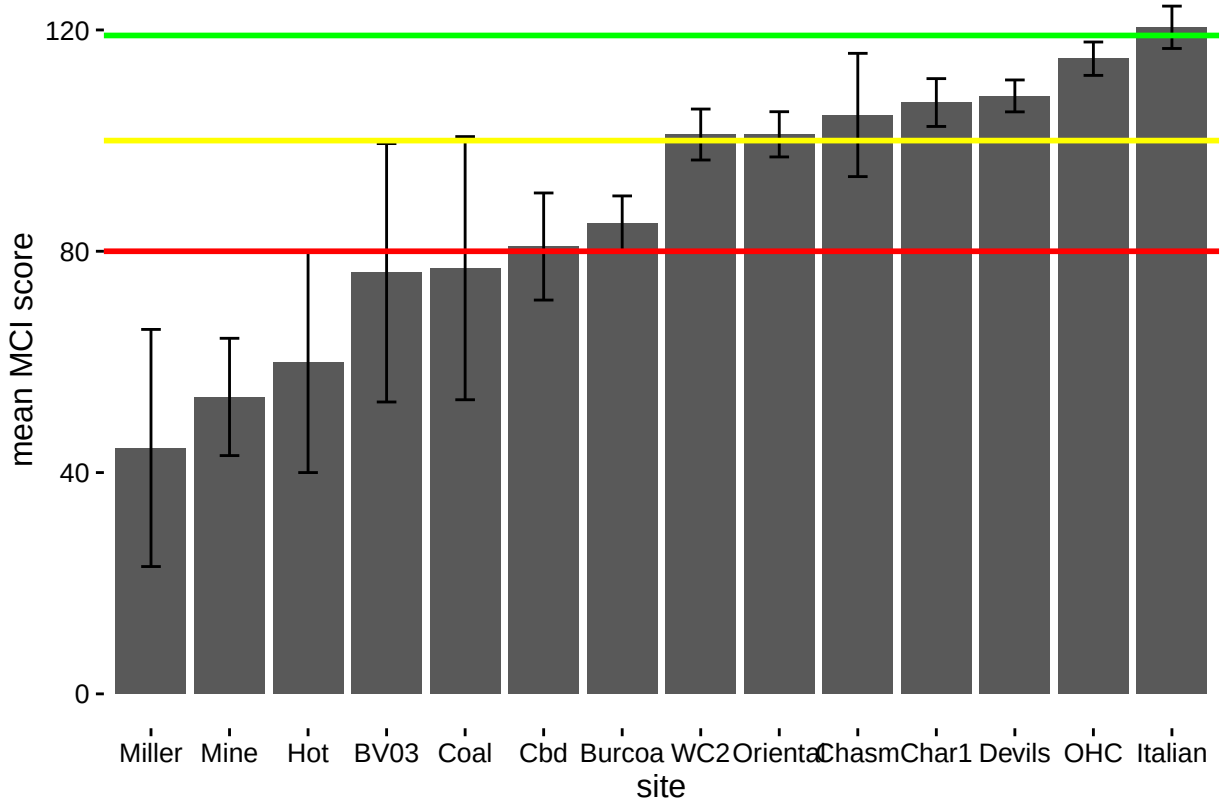
Then, calculate the average MCI score and standard deviation by site.

```
mci.site <-
  mci.surber %>% # our scores by surber calculated above
  group_by(site) %>% # we drop the 'surber' argument and calculate just by site
  summarise(mean.MCI = mean(MCI), sd.MCI = sd(MCI)) # calculate mean and sd MCI
```

Now we will plot the results using ggplot. We will reorder the sites based on their mean.MCI score, from lowest to highest, using the `reorder()` function. We will also add reference lines using the `geom_hline()` command for delineating which sites are considered excellent, good, fair, and poor.

```
ggplot(mci.site, aes(reorder(site, mean.MCI), mean.MCI)) +
# reorder() function organizes sites based on mean.MCI score, lowest to highest
  geom_bar(stat = "identity") + # plot bar
  geom_errorbar( # this command adds error bars
```

```
  aes(ymax = mean.MCI + sd.MCI, ymin = mean.MCI - sd.MCI), width = 0.25) +
# geom_hline() adds reference lines at a given value of y
geom_hline(yintercept = 119, color = "green", size = 1) +
geom_hline(yintercept = 100, color = "yellow", size = 1) +
geom_hline(yintercept = 80, color = "red", size = 1) +
labs(x = "site", y = "mean MCI score") +# change default axis labels
theme_classic() # gets rid of grey background, grid lines, etc.
```




```
# see ?ggtheme for more
```

Site scores > 119 are considered excellent, 100-119 are considered good, 80-99 are considered fair, and < 80 are considered poor.

QMCI

Our data also contains the abundance of taxa per surber sample `bugs.index$number`, allowing us to calculate the quantitative macroinvertebrate index, or QMCI. The QMCI is calculated using a similar formula, with a modification taking into account the abundance of a given taxa:

$$\sum_{i=1}^{S}(a_i * n_i)/N$$

Where S is the total number of taxa, n_i is the abundance for the ith taxa, and N is the total abundance in the sample.

The QMCI ranges from 0 to 10. Note that this is a different scale than the MCI. This is by design because these metrics should not be directly compared to one another.

Again, we start out by calculating the QMCI score per sample `group_by(site, surber)`:

```
qmci.surber <- bugs.index %>%
  group_by(site, surber) %>%
  summarise(QMCI = (sum(MCI * number) / sum(number)))
```

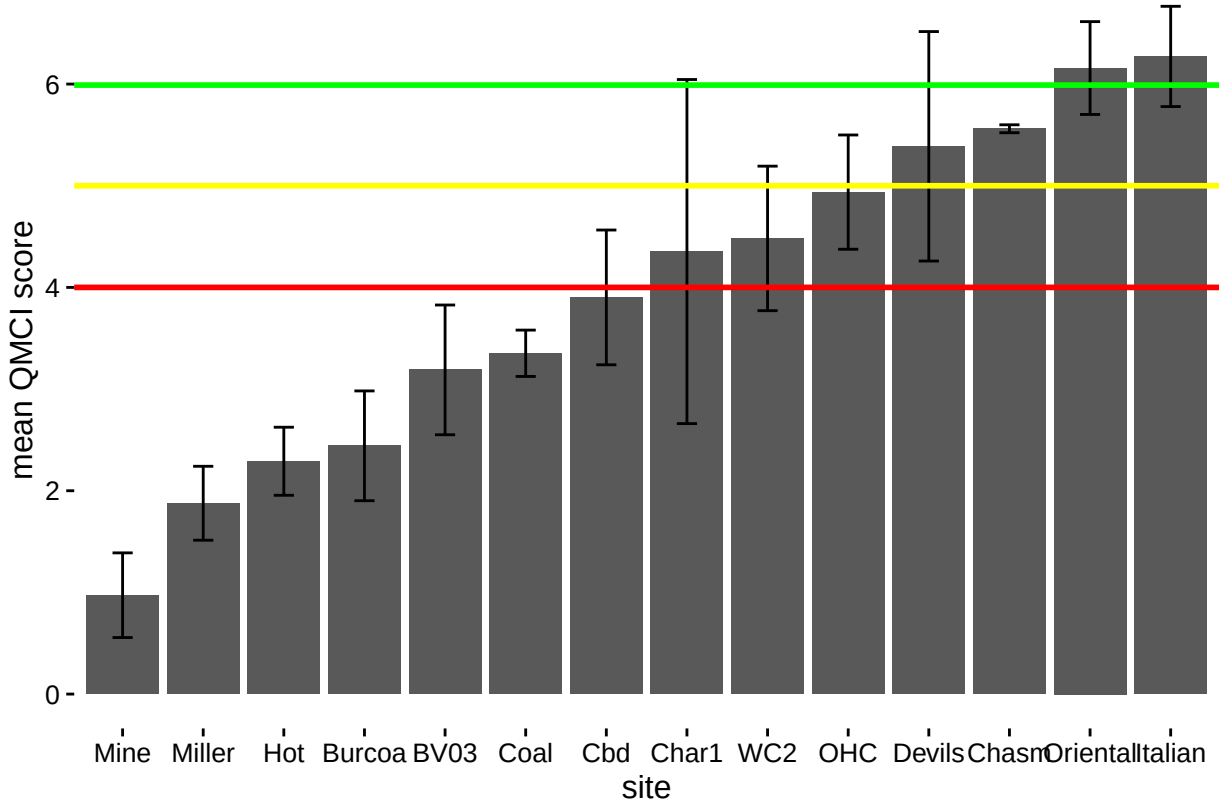
Then we average the individual sample scores by site:

```
qmci.site <-
  qmci.surber %>%
  group_by(site) %>%
  summarise(mean.QMCI = mean(QMCI), sd.QMCI = sd(QMCI))
```

QMCI values >5.99 are considered excellent, 5.90 - 5.00 good, 4.00 - 4.99 fair, and < 4 poor. metrics are purposefully on different scales so they cannot be compared directly.

We plot the results using the same template as above. Make sure you're using all of the right objects and variables! (e.g. MCI –> QMCI, mean.MCI –> QMCI, etc)

```
ggplot(qmci.site, aes(reorder(site, mean.QMCI), mean.QMCI)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
    aes(ymax = mean.QMCI + sd.QMCI, ymin = mean.QMCI - sd.QMCI), width = 0.25) +
  geom_hline(yintercept = 5.99, color = "green", size = 1) +
  geom_hline(yintercept = 5, color = "yellow", size = 1) +
  geom_hline(yintercept = 4, color = "red", size = 1) +
  labs(x = "site", y = "mean QMCI score") +
  theme_classic()
```



Notice the differences in the conclusions we would draw based on the QMCI. Which sites appear to be in "better" condition using the QMCI? Which sites appear "worse"?

AMDI

We calculate the AMDI in much the same way as the MCI. Since the `bugs.index` object already contains the AMDI tolerance values, we can start directly with this object.

The formula to calculate the AMDI is:

$$[(\sum_{i=1}^{S} b_i/S) * log10S] * 10$$

Where b is the tolerance score for the ith taxa and S is the total number of scoring taxa present at a site. If a taxa at a site does not have an AMDI tolerance score, it is not included in the calculation. AMDI scores range from 0 to 100, with higher scores indicating less acid mine drainage impacts. Note that the AMDI can only be calculated with presence/absence data, and there is currently no way of incorporating qauntitative abundance data.

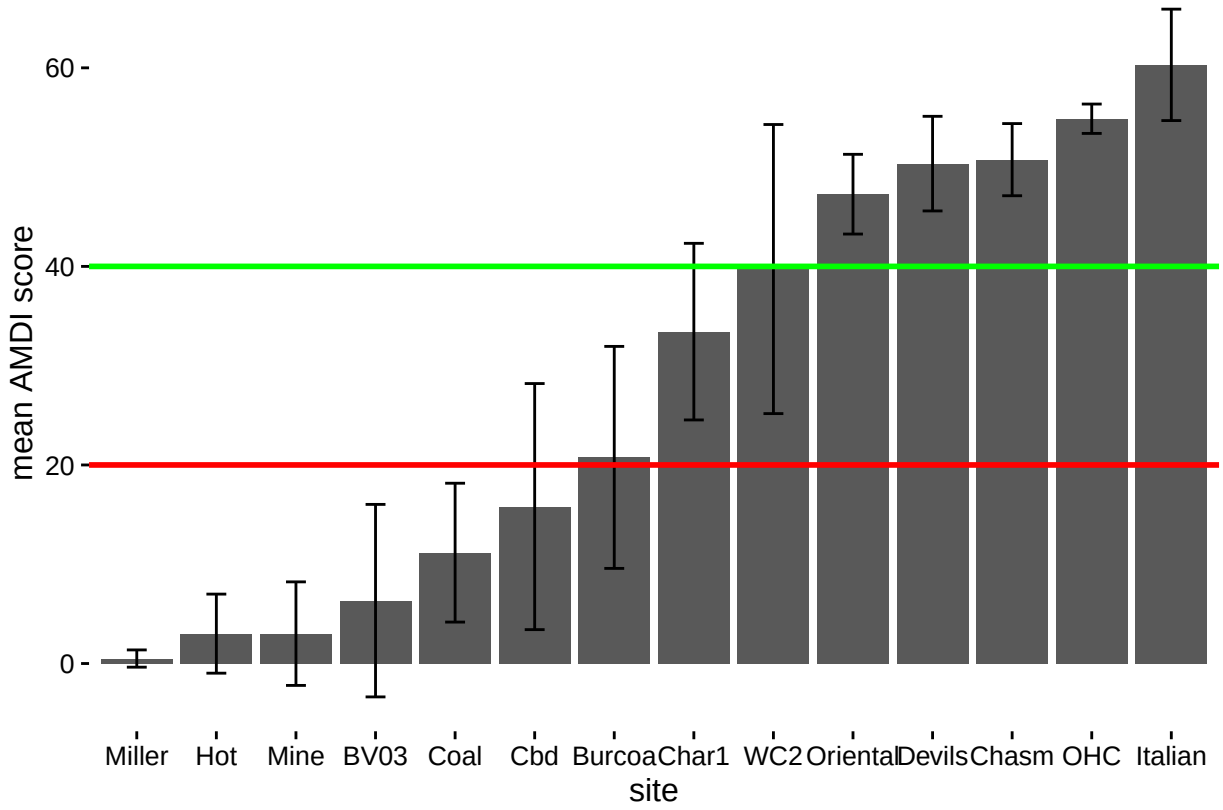
```
amdi.surber <- bugs.index %>%
  group_by(site, surber) %>%
  filter(!is.na(AMDI)) %>% # remove taxa that do not have an AMDI score
  summarise(AMDI = (sum(AMDI) / n()) * log10(n()) * 10)
```

And calculate mean score by site:

```r
amdi.site <-
  amdi.surber %>%
  group_by(site) %>%
  summarise(mean.AMDI = mean(AMDI), sd.AMDI = sd(AMDI))
```

AMDI scores > 40 are considered to be unimpacted by acid mine drainage, while socres between 20-40 and < 20 are considered to be impacted and severely impacted respectively.

```r
ggplot(amdi.site, aes(reorder(site, mean.AMDI), mean.AMDI)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
    aes(ymax = mean.AMDI + sd.AMDI, ymin = mean.AMDI - sd.AMDI), width = 0.25) +
  geom_hline(yintercept = 40, color = "green", size = 1) +
  geom_hline(yintercept = 20, color = "red", size = 1) +
  labs(x = "site", y = "mean AMDI score") +
  theme_classic()
```



Just give me the code!

If you have any questions, see detailed explanation above.

```r
# load library
library(tidyverse)
# read data
bugs <- read_csv("bugs.csv")
```

```r
indices <- read_csv("indices.csv")
# merge datasets
bugs.index <- left_join(bugs, indices, by = "taxa")

#### MCI ####
# MCI in one step
mci.site <-
  bugs.index %>%
  group_by(site, surber) %>%
  summarise(MCI = (sum(MCI) / n()) * 20) %>%
  group_by(site) %>%
  summarise(mean.MCI = mean(MCI), sd.MCI = sd(MCI))
# plot MCI results
ggplot(mci.site, aes(reorder(site, mean.MCI), mean.MCI)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
    aes(ymax = mean.MCI + sd.MCI, ymin = mean.MCI - sd.MCI), width = 0.25) +
  geom_hline(yintercept = 119, color = "green", size = 1) +
  geom_hline(yintercept = 100, color = "yellow", size = 1) +
  geom_hline(yintercept = 80, color = "red", size = 1) +
  labs(x = "site", y = "mean MCI score") +
  theme_classic()

#### QMCI ####
# QMCI in one step
qmci.site <- bugs.index %>%
  group_by(site, surber) %>%
  summarise(QMCI = (sum(MCI * number) / sum(number))) %>%
  group_by(site) %>%
  summarise(mean.QMCI = mean(QMCI), sd.QMCI = sd(QMCI))
# QMCI plot
ggplot(qmci.site, aes(reorder(site, mean.QMCI), mean.QMCI)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
    aes(ymax = mean.QMCI + sd.QMCI, ymin = mean.QMCI - sd.QMCI), width = 0.25) +
  geom_hline(yintercept = 5.99, color = "green", size = 1) +
  geom_hline(yintercept = 5, color = "yellow", size = 1) +
  geom_hline(yintercept = 4, color = "red", size = 1) +
  labs(x = "site", y = "mean QMCI score") +
  theme_classic()

#### AMDI ####
# AMDI in one step
amdi.site <-  bugs.index %>%
  group_by(site, surber) %>%
  filter(!is.na(AMDI)) %>%
  summarise(AMDI = (sum(AMDI) / n()) * log10(n()) * 10) %>%
  group_by(site) %>%
  summarise(mean.AMDI = mean(AMDI), sd.AMDI = sd(AMDI))
# plot AMDI results
ggplot(amdi.site, aes(reorder(site, mean.AMDI), mean.AMDI)) +
  geom_bar(stat = "identity") +
  geom_errorbar(
```

```
  aes(ymax = mean.AMDI + sd.AMDI, ymin = mean.AMDI - sd.AMDI), width = 0.25) +
geom_hline(yintercept = 40, color = "green", size = 1) +
geom_hline(yintercept = 20, color = "red", size = 1) +
labs(x = "site", y = "mean AMDI score") +
theme_classic()
```