# Maintaining, supporting, and sustaining scientific software

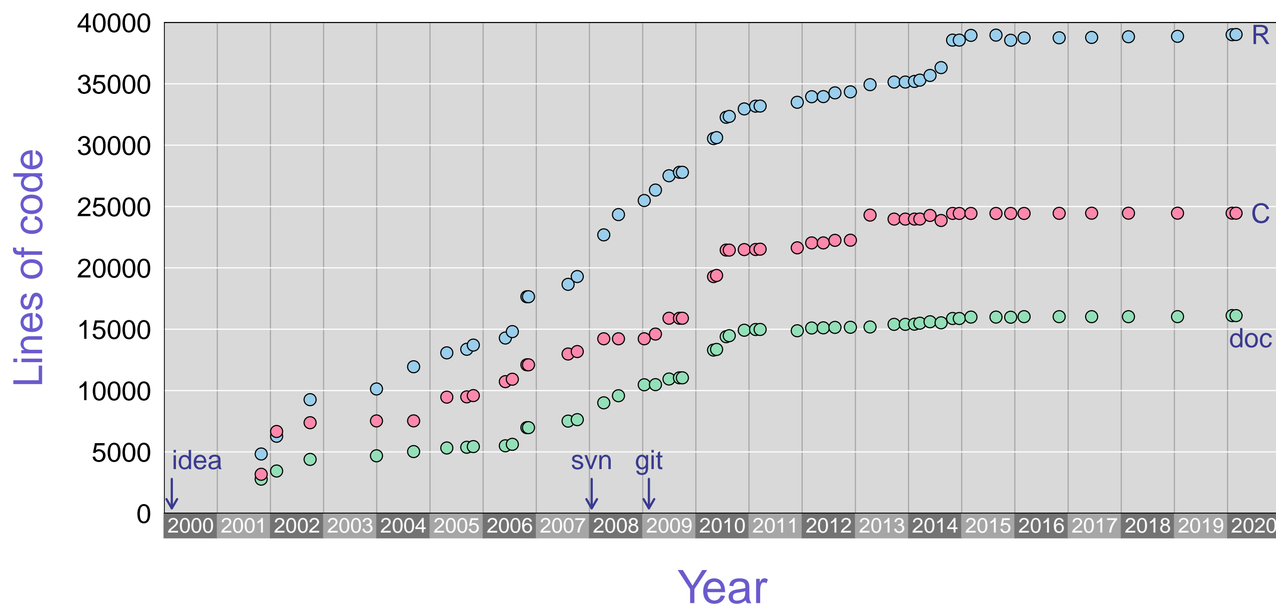
Biostatistics & Medical Informatics, UW–Madison

kbroman.org
github.com/kbroman
@kwbroman

Course web: kbroman.org/AdvData

This lecture is about developing and maintaining scientific software, and the value of this endeavor for academic data scientists.

# 20 years of R/qtl






For more than 20 years, I've been developing an R package for QTL mapping, R/qtl. This graph shows the number of lines of R code, C code, and documentation in each release of the package.

If I'd known what I was getting into, I maybe wouldn't have started the project. But it's actually been hugely valuable for me. I've made a lot of friends and formed a lot of great collaborations from the work. And it's been useful to the community, as well.

Today I want to talk about this experience.

# Why?



I've put a lot of effort into this software, and not just in developing it, but also in providing support to users. Why do so?

First, the software is important for my own research work, and particularly for many of my collaborative research, and I've come to learn that improvements aimed at making the software simpler to use have also made it simpler for me to use.

Second, if you want people to use your software, you need to help them get started. And not just in providing them tutorials and other documentation, but also in that first nasty step of getting their data into the right form. If they get frustrated at that first step, they'll just give up and look elsewhere.

Third, I've formed a lot of new collaborations through the software. People use it but find that their own data present difficulties that they can't quite figure out, and they ask me for help. Some of that short-term helps turns into longer collaborations.

Finally, the software has been a great platform for me to provide my own methods developments. If I want people to use my methods ideas, I need to provide them software; software integrated into a widely-used tool will be easier to pick up.

# Good things

- some of the code
- basics of the user interface
- diagnostics and data visualization
- quite comprehensive
- quite flexible



When you've been working on a project for a long time, you start to focus primarily on its weaknesses, and it can be difficult to identify any good aspects anymore.

But there are a number of good things about R/qtl. Some of the code, particularly the HMM for handling missing genotype information, is quite good. And the basic user interface was rather a nice idea; it hides the complexities while still making them accessible to an advanced user.

And the data diagnostics and data visualizations are quite good. And overall the software is both quite comprehensive and quite flexible. It has served me well, overall.

# Bad things



Bad things are more easy to identify. Really, it's hard not to just see all of the bad things.

Some of the code is really terrible and complicated. Short-term efforts to fix problems were accomplished via band-aids that made the code more complicated and confusing and harder to maintain.

And there was never any defined specifications on the data format or describing the software design, which makes it much harder for others to get involved.

Small changes in one place have implications throughout the code base that are to identify, making bug fixes or further developments much more difficult.

# Input file

| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | liver | spleen | sex | pgm | D1Mit18 | D1Mit80 | D1Mit17 | D2Mit379 | D2Mit75 |
| 2 | | | | | 1 | 1 | 1 | 2 | 2 |
| 3 | | | | | 27.3 | 51.4 | 110.4 | 38.3 | 48.1 |
| 4 | 61.92 | 153.16 | m | 1 | BB | SB | SB | SB | SB |
| 5 | 88.33 | 178.58 | m | 1 | – | – | – | BB | BB |
| 6 | 58 | 131.91 | m | 1 | BB | SB | SB | SB | SB |
| 7 | 78.06 | 126.13 | m | 1 | SB | SB | BB | SS | SS |
| 8 | 65.31 | 181.05 | m | 1 | – | – | – | SB | SB |
| 9 | 59.26 | 191.54 | m | 1 | – | – | – | SS | SS |
| 10 | 59.47 | 154.88 | m | 1 | BB | BB | BB | SB | SB |
| 11 | 65.63 | 184.12 | m | 1 | – | – | – | SB | SB |
| 12 | 38.64 | 133.05 | m | 1 | SB | BB | SB | SB | SB |
| 13 | 60.94 | 275.63 | m | 1 | – | – | – | SB | BB |
| 14 | 51.48 | 395.25 | m | 1 | – | – | – | SB | BB |
| 15 | 47.12 | 260.45 | m | 1 | BB | SB | SB | BB | BB |



To illustrate one of the bad things, let's first look at the data input format. A single comma-delimited file contains the three aspects of the data: the phenotypes, the genotypes, and the marker data.

Each row is an individual, the initial columns are the phenotypes, and the subsequent columns are the marker genotypes. For the markers, the second row indicates the chromosome assignments, and the third row contains the locations. The initial cells in the second and third rows need to be completely blank.

The first thing, upon reading in the data, is to split the data into the three parts: the phenotypes from the genotypes and then the marker map from the genotypes. We do that first by identifying the first non-blank cell in the second row.

When R/qtl started to be used for larger data sets (such as with gene expression phenotypes, say on 30,000 genes), users complained about how long it took to load a file. I'd ask "Well, how much data do you have?" and then say "Well, that's a big file, it's going to take a while."

But then I had a data set with like 1500 traits and 200 mice and it took a full minute to load, and that just seemed wrong, and so I finally looked to see what was happening.

## Stupidest code ever

```
n <- ncol(data)
temp <- rep(FALSE,n)
for(i in 1:n) {
  temp[i] <- all(data[2,1:i]=="")
  if(!temp[i]) break
}
if(!any(temp)) stop("...")
n.phe <- max((1:n)[temp])
```

kbroman.org/blog/2011/08/17/the-stupidest-r-code-ever



And this is what I found. In the `for` loop, I'm look at the first cell, and then the first two cells, then the first three cells, etc., until I get to a point where they are not all empty. Then I brake out of the loop.

It turned out that for my file with 1500 traits and 200 mice, it took like 2 seconds to read the data but 58 seconds to find this spot to split the phenotypes and genotypes.

This code is no longer in R/qtl, but it's embarrassing to see how long it was part of the package.

Open source means
everyone can see my stupid mistakes

Version control means
everyone can see every stupid mistake I've ever made



This is one of my favorite things to say.

## More typically bad code

The `scantwo()` function is 1446 lines long.

The related C code is 20% of the C code in R/qtl.



This sort of thing really needs to be broken up into smaller pieces.

As it is, it's really hard to extend and maintain.

# Baroque data structures

```
attr(mycross$geno[["X"]]$probs, "map")
```



The data structures in R/qtl really got out of control: far too deeply nested.

Attributes can be great, but when I learned about them, I started to pile all sorts of shit in there.

# Documentation



I've learned a lot in the last 20 years. First, regarding documentation. I've written a lot of detailed documentation. But most of it is not at all read. All those help files carefully describing the use of each function? They're mostly not read. Users do like the examples, but the rest is generally too technical really just useful as a reference rather than for learning or trouble-shooting.

What folks really want are very tailored tutorials: basically case studies showing how to use the software and how to interpret the results, ideally with data and with analysis goals that are very close to the users' applications.

So by all means write the detailed documentation, but focus most of your effort on writing vignettes or other tutorials that match what your users are trying to do with the software.

# User support

12

Supporting the software, by answering users' questions, is time consuming and can be frustrating (both for me and for the users), but it is where you can have the greatest impact. Your responses to questions can be the difference between users continuing with the software and giving up in frustration.

Many times users don't really seem to understand how to ask questions. I generally want to some details about the nature of the data, and then the exact code and output and error messages.

I often get questions like, "Can you look at the attached 20 page Word document and let me know if I'm doing this okay?" Or, "I tried X and it didn't work." I've found that it's best to take a short break between when I read a question and when I respond. Often I'll be really annoyed by a question initially, but if I wait 30 minutes, I can return to it and respond in a much more positive way.

I try to focus on my own frustrating experiences with software, and recall that while for me it may be the 100th time I've heard a particular question, for the user it's the first time.

I'd hope to build a "community" of people answering each other's questions, but it continues to be largely me doing the answering.

## Incorporating others' code



It's great to get contributions from others. But once you've incorporated their features, you're responsible for maintaining and supporting it.

I now think that big things should mostly be made to be separate packages.

# Version control



How on earth did I get by before git?

Really critical for incorporating others' changes, and for trying out new things without breaking what's working.

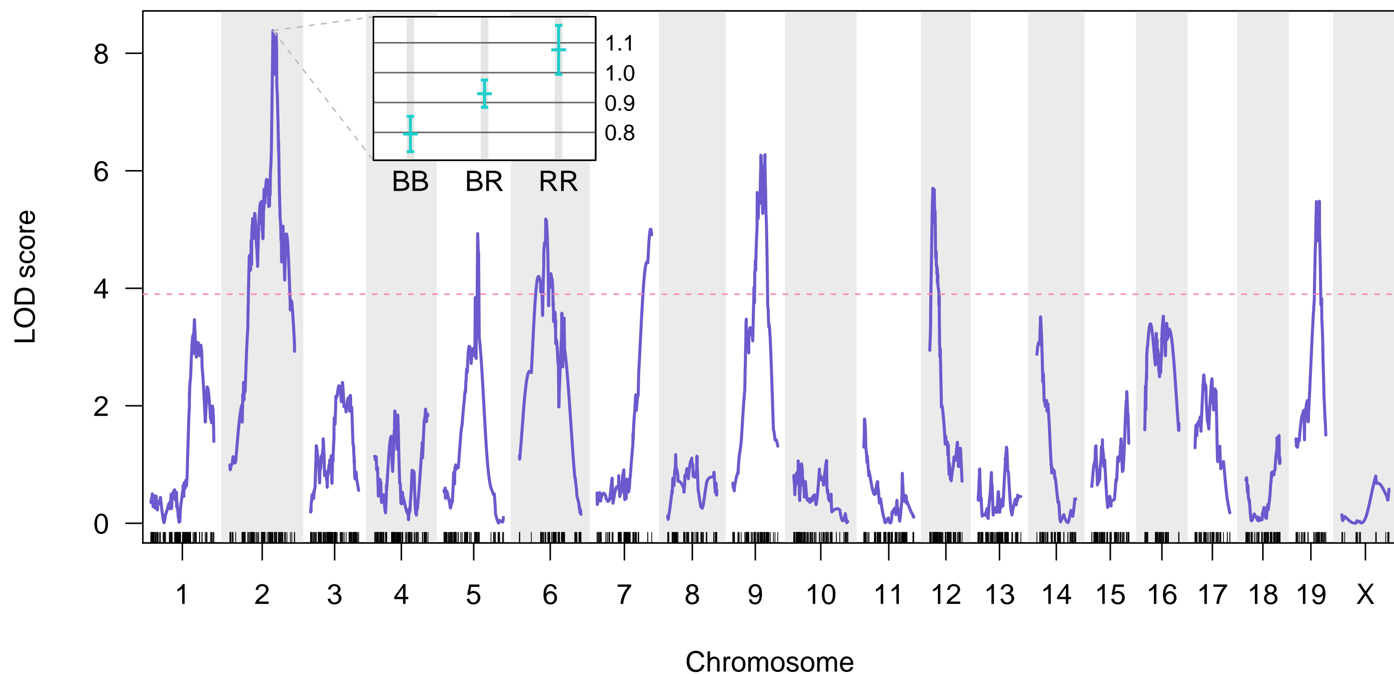
# Tests

15

How on earth does any of this work? There are basically no tests that the code works.

I'm now thoroughly in love with unit tests and Hadley's testthat.

# QTL mapping



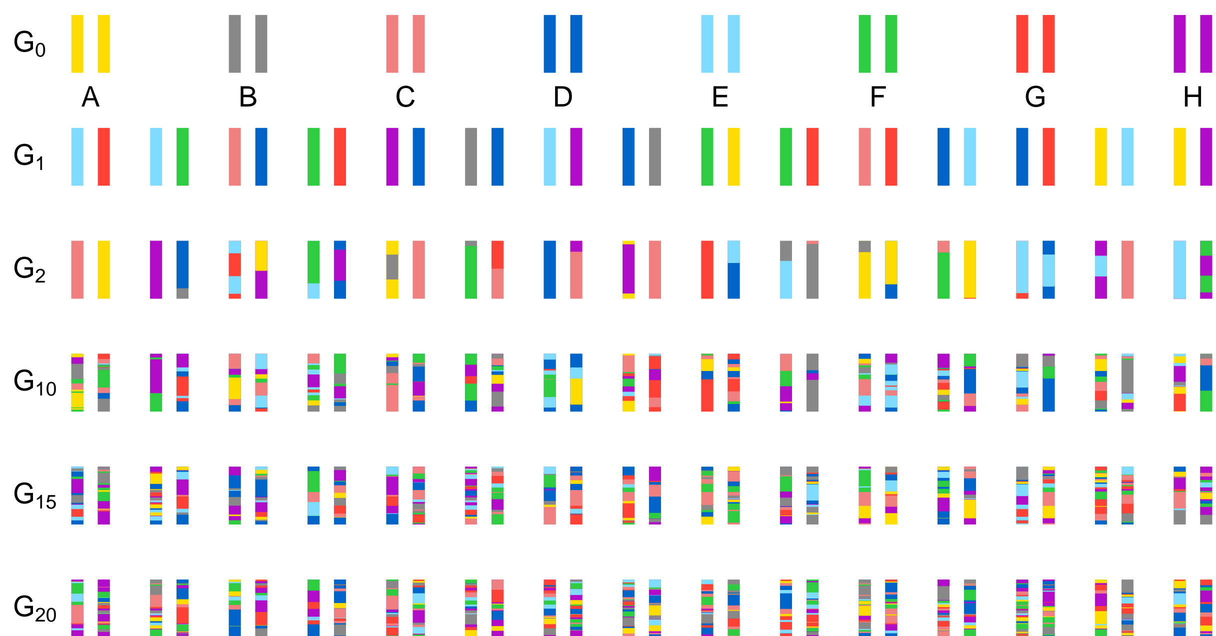



So back to the subject of the software: mapping QTL in experimental crosses.

The big problem has been the poor mapping resolution of QTL.

The solution has been first to look at multi-parent crosses and more generations. And second to look at high-dimensional, genome-scale traits.

## Heterogeneous stock




17

The central data structure in R/qtl doesn't really fit these sorts of multi-parent crosses. Revising things to work would be a ton of work.

## Challenge: scale of results

genotypes

phenotypes

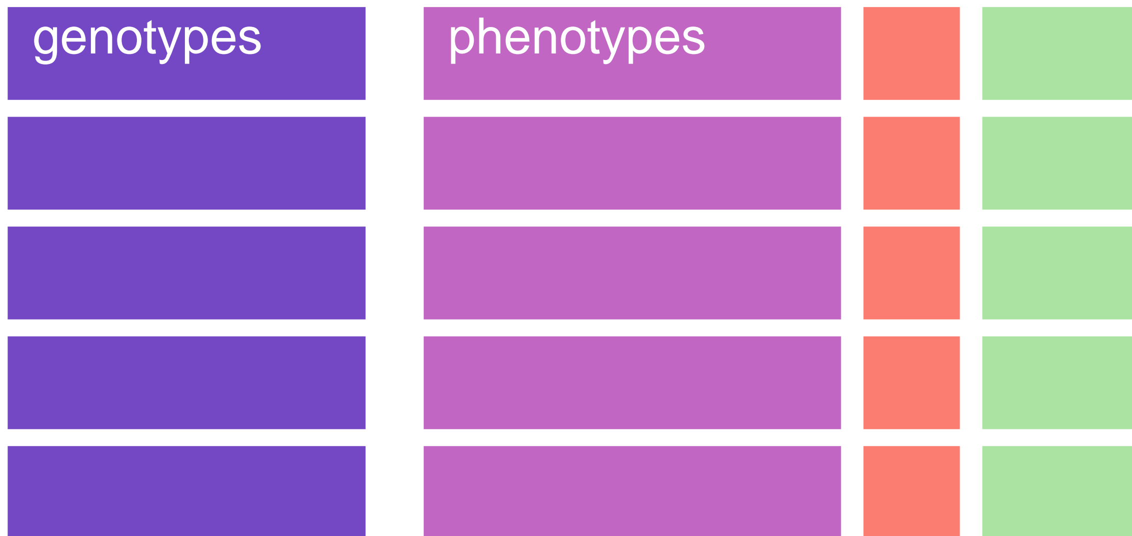
results

18

And a general challenge I have is just the scale of the results. We think of the data are being large, but even if we just associate each genotype to each phenotype, we have a results object that is considerably larger than either data set.

The problem is not just the calculation and storage of the results, but how to make sense of them: how to find the interesting bits? How to help my collaborators to explore these results on their own?

## Challenge: organizing, automating






A second challenge is the strong need to organize and automate the analysis so that it doesn't kill me. Because my collaborators are always measuring things in waves and adding on new batches of traits without thinking about the time and other resources to analyze them.

ropenscilabs.github.io/miner_book



The realities of modern high-dimensional QTL data has led me to start over with a completely new package: R/qtl2.

## R/qtl2

- High-density genotypes
- High-dimensional phenotypes
- Multi-parent populations
- Linear mixed models

`kbroman.org/qtl2`



The goals of the new package have been to handle the high-dimensional data, both genotypes and phenotypes, and to handle these multi-parent populations. Further, we need to be able to fit linear mixed models to account for population structure.

## R/qtl2: Let’s not make the same mistakes

- C++ and Rcpp
- Roxygen2 for documentation
- Unit tests
- A single “switch” for cross type
- Yet another data input format
- Flatter data structures, but still complex



In starting over, I don’t want to make the same mistakes.

And so I’m using C++ and Rcpp rather than C and .C(). This has been hugely valuable. Second, I’m using Roxygen2 for the documentation and writing extensive unit tests. Further, I’m particularly proud of having a single “switch” for cross type. In R/qtl1, I have “if backcross this, if intercross that” all over the place. That makes extending the code to new cross types a total pain. In the new code, I don’t have to do that at all.

I’m also focusing on much shorter functions and trying to cut down on some of the ugly repeated code. But I’ve not been totally successful in that, and that’s not my only bad tendency.

I’ve created yet another data input format. But here, I’m sticking with just one and forcing the user to conform to my requirements rather than trying to have code that adapts to every damn thing they might throw at me.

Finally, I’ve gone with flatter data structures, but they’re still complex. Basically I find that there’s a trade-off between the complexity of the user interface and the complexity of the data, and I think for my users it’s better to have the user interface be simple.

## Sustainable academic software



A central question here is how to have sustainable academic software?

My boss just cares about publications and grants. And really just grants. Granting agencies care about papers not software. Grant reviewers basically care just about innovation: new exciting methods rather than plodding software implementations. My colleagues, too, are mostly interested in people that solve hard, sophisticated problems, more than usefulness.

Effort on software has been super useful to me, but the advantages have all been indirect and long term.

But I think what has really made R/qtl sustainable has been that it's continued to be important for me, for my own research. Because I'm using it every day, it's been important for me to continue to maintain and extend it.

The main problem with orphan software is that academics more on to a different and totally unrelated problem, or the key student or post doc moves on to other things. The focus on constant innovation contributes to this.

## Acknowledgments

Danny Arends
Gary Churchill
Nick Furlotte
Dan Gatti
Ritsert Jansen
Pjotr Prins
Śaunak Sen
Petr Simecek
Artem Tarasov
Hao Wu
Brian Yandell
Robert Corty
Timothée Flutre
Lars Ronnegard
Rohan Shah
Laura Shannon
Quoc Tran
Aaron Wolen

NIH/NIGMS




A ton of people have participated in the development of R/qtl (and R/qtl2). In particular Hao Wu, who was a software engineer with Gary Churchill and is now a faculty member at Emory, wrote big chunks of R/qtl1. But also Gary Churchill and Śaunak Sen who I've worked with closely for many years.