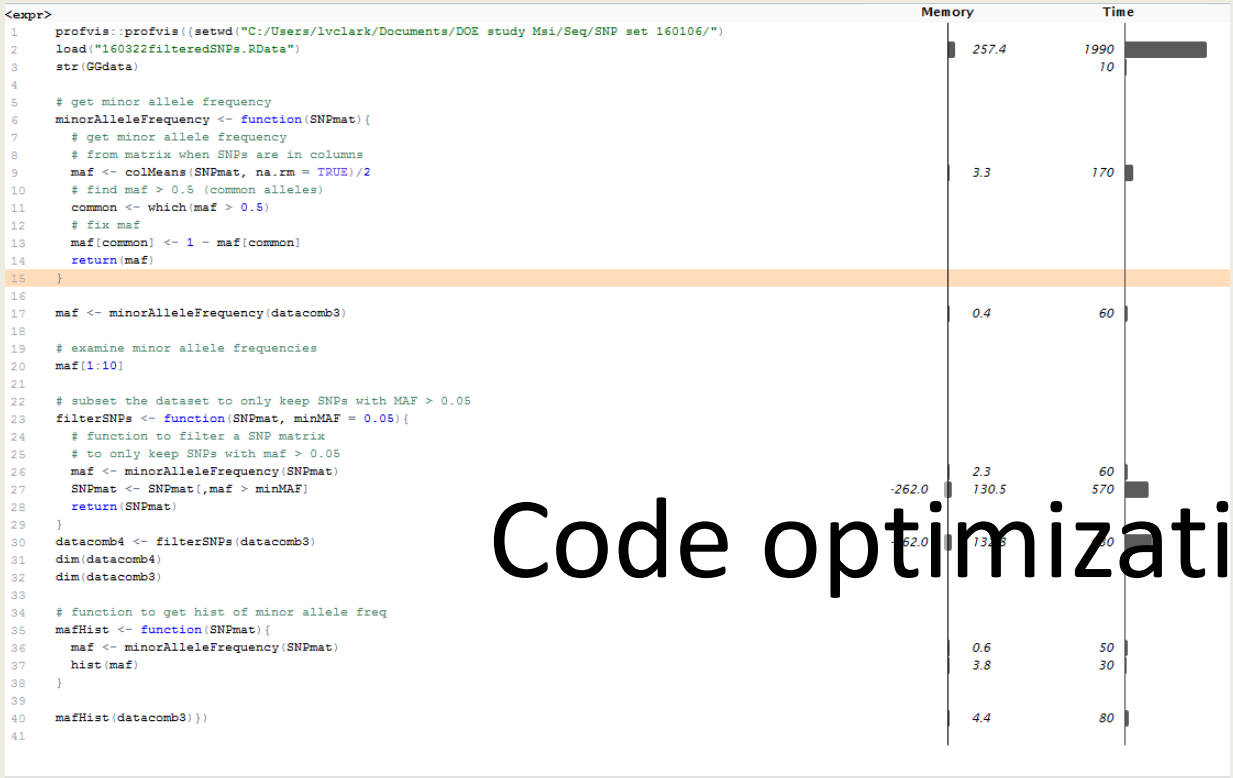
Code optimization

Lecture 10, CPSC Fall 2018



Priorities when writing code

1. Code is easy for a human to understand
2. Code is efficient in terms of computation time and memory

Don’t go out of your way to make code more efficient (especially at the expense of making it readable) unless it is running slowly

R is not fast

- R is designed for interactive use
- Being interpreted rather than compiled slows it down a lot

- BUT there are some tricks we can use to make it go faster

Testing how fast your code is running

- "microbenchmark" package for comparing short pieces of code
  - `microbenchmark(x / 2, 0.5 * x)`
- "Profile" in RStudio for scripts of any length that take ≥ ~0.5 seconds to run

Mini-exercise

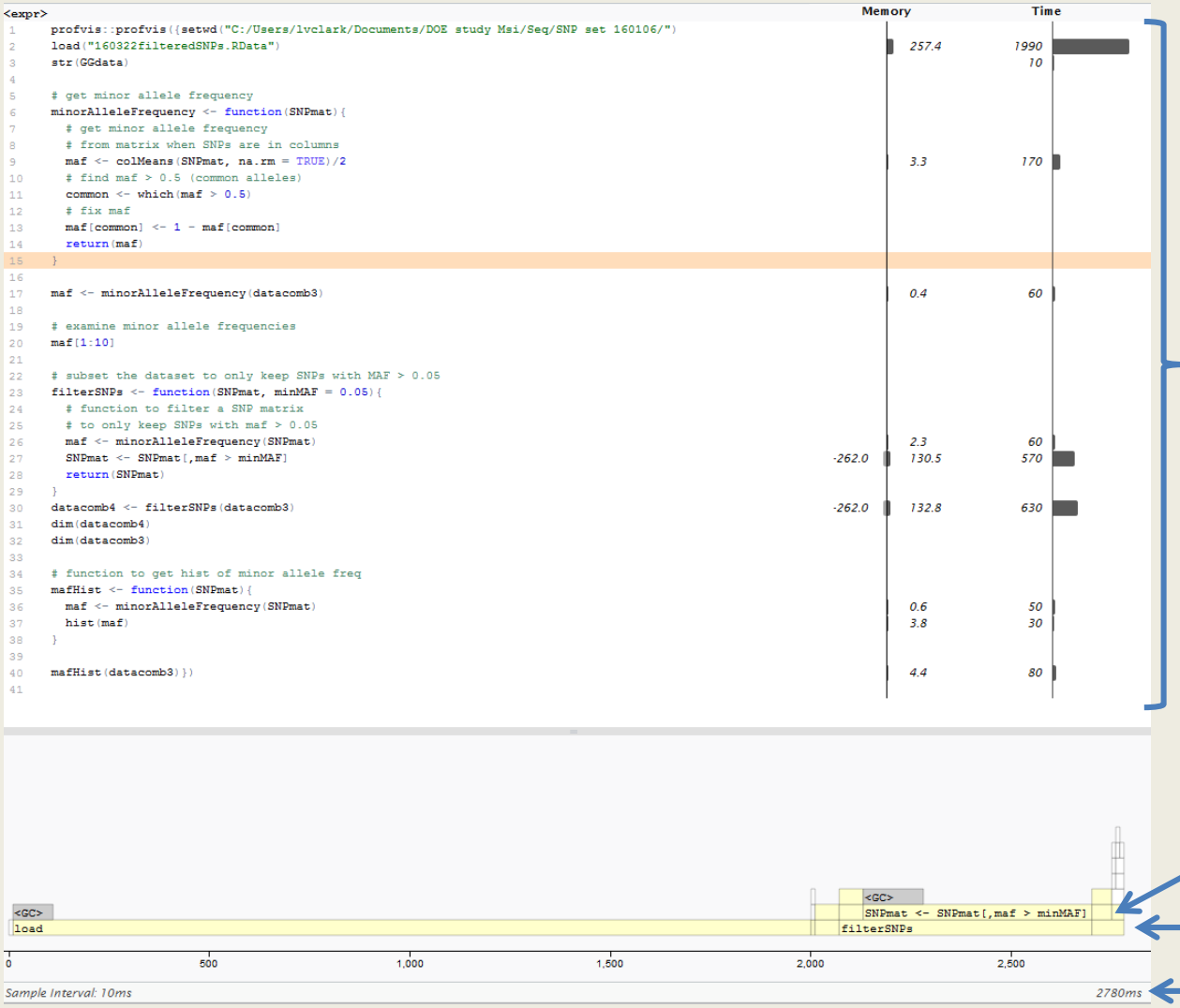
- Microbenchmarking to determine best indexing method
- Make vector
  - `vect <- 1:100`
  - `names(vect) <- paste("Sample", 1:100)`
- How does `vect[50]` compare to `vect["Sample 50"]`?

Profiling code

This will show you what parts of your code are eating up the most time and memory

1) Highlight some code that you want to run
2) In RStudio do Profile → Profile Selected Lines
3) View the resulting report

Output of profiler



Graphs of memory and time by line of code

Functions called by those functions

Functions that were called

Total time to run

Loops are slow (in R)

- Code is reinterpreted with each iteration
- If you can use vectorized math or vectorized functions, do that instead
- If you have to do a loop, move computations out of the loop if they don't absolutely have to be inside it
- `apply` functions are not necessarily faster

Mini-exercise

- Vectorizing as much as possible instead of writing loop

```
rando <- runif(400)
for(i in 1:length(rando)){
  rando[i] <- rando[i] * 3 ^ -6
}
```

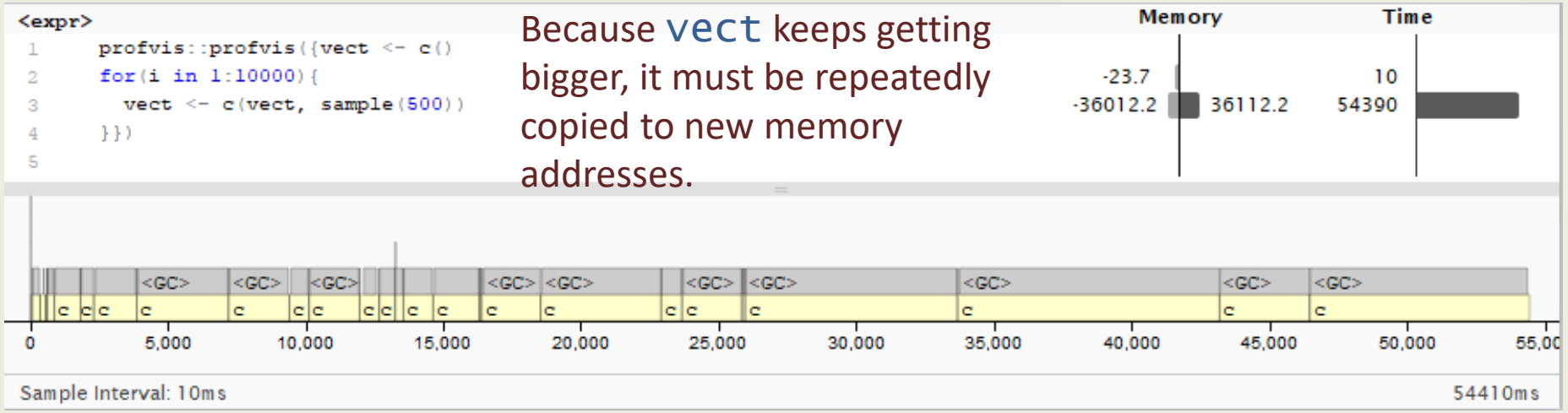
- How can this be sped up?

Pre-allocating space for objects

- It is possible to make a vector, matrix, or data frame bigger on each iteration of a loop
- **Faster**: set up a blank vector/matrix/data frame that is the final size you need, then fill it in by indexing

Pre-allocation continued

- If you see a lot of <GC> in your profiler output, that means **garbage collection**
  - (deletion of objects in memory if they no longer have names in a namespace)
  - Indicates that you could speed things up by preallocating.
- You can use `tracemem` to debug and confirm which object is being repeatedly moved



Mini exercise

- Make a matrix with some random numbers and add columns repeatedly in a loop

```
mat <- matrix(runif(500),
              nrow = 500, ncol = 1)
for(i in 1:100){
  mat <- cbind(mat,
               matrix(runif(500),
               nrow = 500, ncol = 1)
}
```

- Keeping the loop, how could you fix the preallocation issue?

Assorted other tips

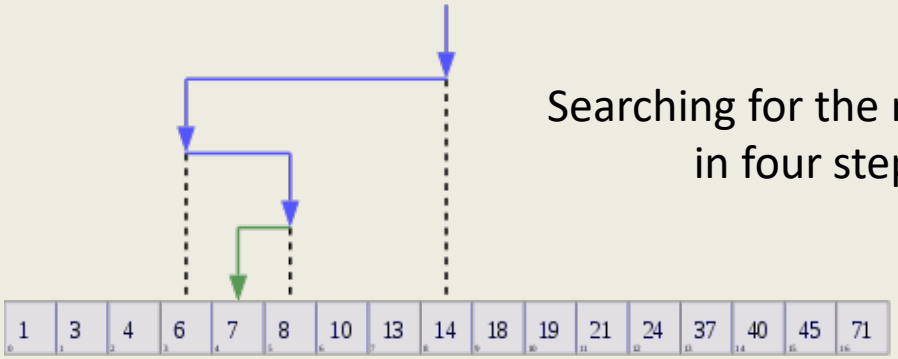
- Matrices are faster than data frames if data are all one type
  - "data.table" package for faster computations on big data frames
- `scan` is faster than `read.table` if you already know the file format (esp. for numbers)
- `colSums`, `colMeans`, `rowSums`, `rowMeans` much faster than `apply`

Quick matching of text

- Built-in `match` function finds first instance of a value within a vector
- The "fastmatch" package has an `fmatch` function that is like a much faster version of match if you are doing multiple searches
  - Uses hash tables
  - Also has `%fin%` to replace `%in%`
  - `ctapply` to replace `tapply` if all groups are contiguous

Binary search

- For quickly matching a value within a sorted vector
- If that value doesn't exist, it finds the closest one
- Divides the search space in half with each step
- Implemented in "data.table" package



Searching for the number 7
in four steps

Image: AlwaysAngry on Wikimedia Commons

Using compiled code in R

(Biggest advantage = loops run quickly)

Script, source code, and/or manually typed in commands

Data

INTERPRETER

Output (printed to console or written to file)

Source code

Data

COMPILER

Executable computer program file

Output (printed to console or written to file)

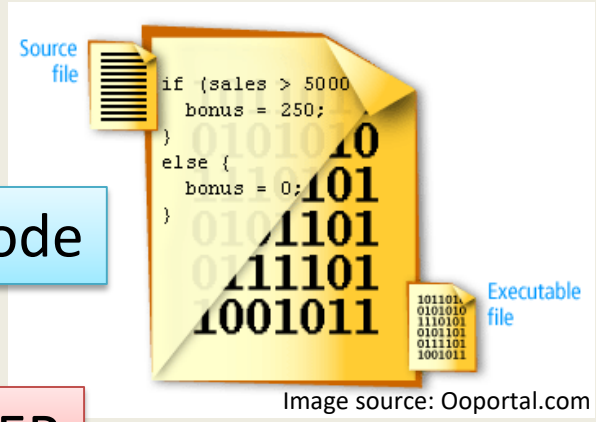


Image source: Ooportal.com

Most functions created in R aren't compiled by default

- Type name of function into console without parentheses to see what is in that object.
- If it was a function you created with `function`, it contains the R code itself.
- This includes functions installed with R packages.
- Every time the function runs, R must re-interpret the code in the functions.

Compiling simple R functions

- `library(compiler)` – installed with R
- By default, R will compile your functions when it can (as of version 3.4)
- Tends to only work on fairly simple code
- Good for cases where you are forced to do arithmetic in a loop
- If your function has a “bytecode” then it is compiled
- Now the interpreter does not re-run every time you run the function

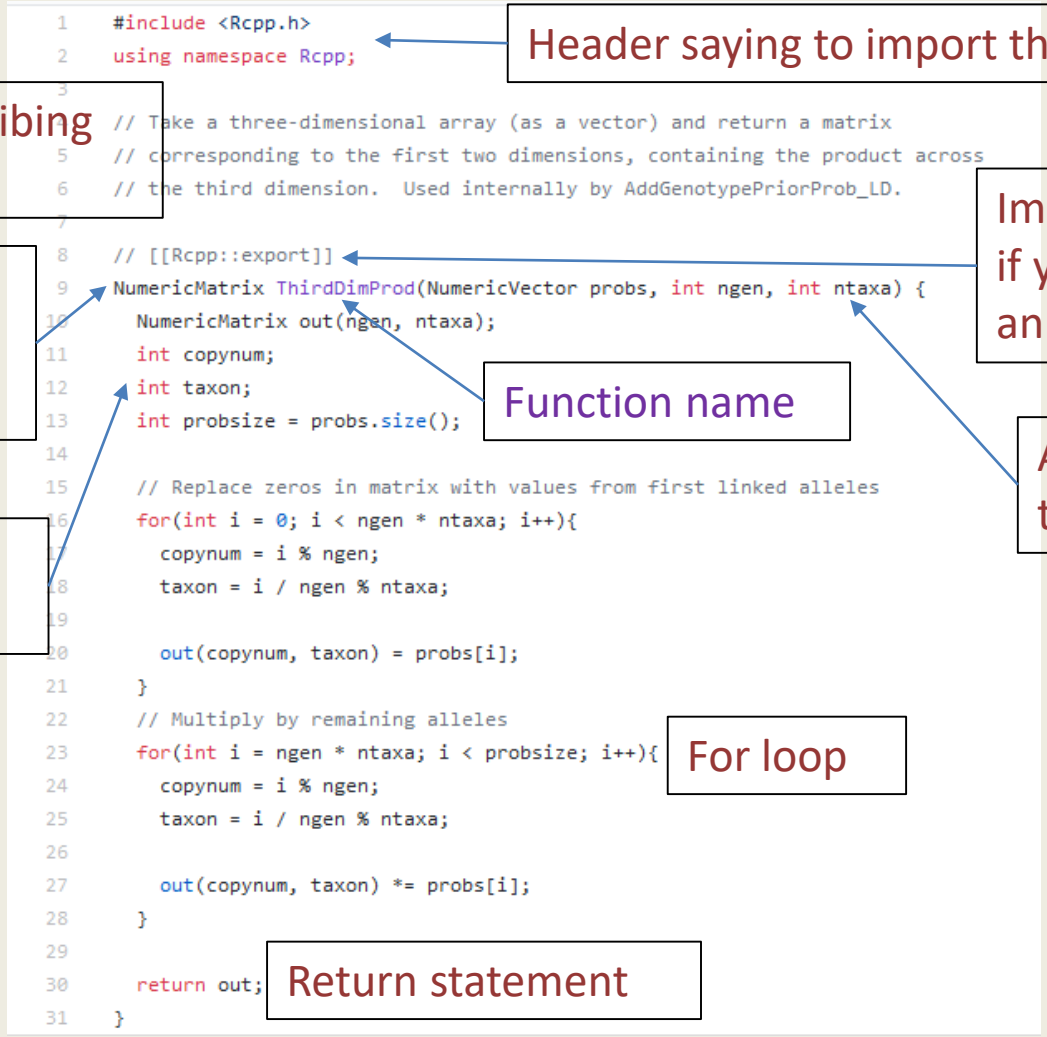
Mini exercise

- Write a small function that does math in a for loop
- Use the function once
- Did the function get compiled?

Writing functions in C++ using Rcpp

- C++ is a compiled language based on C
- You can write C++ code in text files with the extension .cpp
- The Rcpp package lets you write functions in C++, then import them as compiled functions into R
- Rcpp is popular in R packages that are optimized for performance

Anatomy of some C++ code



```cpp
#include <Rcpp.h>
using namespace Rcpp;

// Take a three-dimensional array (as a vector) and return a matrix
// corresponding to the first two dimensions, containing the product across
// the third dimension.  Used internally by AddGenotypePriorProb_LD.

// [[Rcpp::export]]
NumericMatrix ThirdDimProd(NumericVector probs, int ngen, int ntaxa) {
  NumericMatrix out(ngen, ntaxa);
  int copynum;
  int taxon;
  int probsize = probs.size();

  // Replace zeros in matrix with values from first linked alleles
  for(int i = 0; i < ngen * ntaxa; i++){
    copynum = i % ngen;
    taxon = i / ngen % ntaxa;

    out(copynum, taxon) = probs[i];
  }
  // Multiply by remaining alleles
  for(int i = ngen * ntaxa; i < probsize; i++){
    copynum = i % ngen;
    taxon = i / ngen % ntaxa;

    out(copynum, taxon) *= probs[i];
  }

  return out;
}
```

Key differences between C++ and R

	R	C++
Comment character	#	//
Assignment operator	<-	=
End of command	Line break	;
First index of vector	1	0
For loop	`for(i in 1:10)`	`for(int i 0; i < 10; i++)`
Classes of variables	Determined from context	Must be declared
Scalar variable	Vector of length 1	Different from a vector
Matrix indexing	`[ , ]`	`( , )`
Calling a method	`method(object)`	`object.method()`

Compiling a C++ function and loading it into your Global Environment

- In RStudio – “Source” button in the upper right corner of the script editor
- In a script or at the R console:
  - `library(Rcpp)`
  - `sourceCpp("mysource.cpp")`
  - Okay to get warning from `normalizePath`

Including C++ functions in your R package

- All C++ code goes into a package subdirectory called `src` (short for “source”).
- In NAMESPACE, include the line `useDynLib(mypkg)`, where mypkg is your package name.
- If the function will be accessible to user, put it in NAMESPACE and write a help file for it.
- To NAMESPACE, also add `importFrom(Rcpp, evalCpp)`

Including C++ functions in your R package (cont'd)

- To the DESCRIPTION file, add:
  ```
  Imports: Rcpp
  LinkingTo: Rcpp
  ```
- At the R console, run `compileAttributes()` before you build your package.
- See http://mirror.las.iastate.edu/CRAN/web/packages/Rcpp/vignettes/Rcpp-package.pdf for more info

When should I use Rcpp?

- When speed is important and loops or apply functions are slowing you down a lot. (i.e. when you can't find a way to vectorize in R)
- When the task is simple enough that you can write the code without taking a course in C++
- When you expect the function to be used a lot in the future (i.e. worth the time to write it)

When should I not use Rcpp?

- When the time needed to figure out how to write C++ is longer than the time it would take to just run it in R
- When other R users need to be able to understand the code easily

Thursday's lab

- Converting some R code to C++ code
- Optimizing R code within R