[**Chapter 1:First Day meeting resourses**](#_6lvg1vwaz0uq) **3**

[**1.1 Introduction to Single cell:**](#_lbqydunje4o) **3**

[**https://www.embopress.org/doi/10.15252/msb.20188746**](#_lbqydunje4o) **3**

[**1.2 A step by step guide to single cell pipeline:**](#_lbqydunje4o) **3**

[**https://f1000research.com/articles/5-2122**](#_lbqydunje4o) **3**

[**1.3 Sanger course on single cell:**](#_lbqydunje4o) **3**

[**https://scrnaseq-course.cog.sanger.ac.uk/website/introduction-to-single-cell-rna-seq.html**](#_lbqydunje4o) **3**

[**1.4 Valentine Svensson PhD thesis:**](#_lbqydunje4o) **3**

[**https://www.repository.cam.ac.uk/handle/1810/267937**](#_lbqydunje4o) **3**

[**1.5 A large collection of Single cell materials (I will just ask to gloss over this):**](#_lbqydunje4o) **3**

[**https://www.repository.cam.ac.uk/handle/1810/267937**](#_lbqydunje4o) **3**

[**Chapter 2: Portable Shiny App Based on Linux**](#_okcxz9jjuff7) **4**

[**How to Deploy Interactive R Apps with Shiny Server: https://www.linode.com/docs/development/r/how-to-deploy-rshiny-server-on-ubuntu-and-debian/**](#_yc6xvfygzj1w) **4**

[Deploy a Shiny App to a Remote Server](#_nd2z02rv9xvf) 4

[Deploy Your App](#_67o89hmh57ga) 4

[Configure Shiny Server](#_oebu6jezs60f) 4

[**Deploying R shiny app as a standalone application:**](#_cdzczpr6ybv) **4**

[**Develop shiny apps on linux and deploying them as desktop app on a windows machine: https://stackoverflow.com/questions/45724143/develop-shiny-apps-on-linux-and-deploying-them-as-desktop-app-on-a-windows-machi**](#_ntwub8o4s0ye) **4**

[Desktop DeployR: https://oddhypothesis.blogspot.com/2016/04/desktop-deployr.html](#_4poe4un54gxb) 4

[**Chapter 3: Optimizing R Shiny Application**](#_jw69drdag4lk) **6**

[**Optimizing Your Application:https://bookdown.org/hadrien/how\_to\_build\_a\_shiny\_app\_from\_scratch/optimizing-your-application.html**](#_2pz86q9mvvok) **6**

[**Improving shiny app loading speed:https://community.rstudio.com/t/improving-shiny-app-loading-speed/5126**](#_yk58sg7q2dms) **6**

[**Scaling and Performance Tuning with shinyapps.io: https://shiny.rstudio.com/articles/scaling-and-tuning.html**](#_kuz907vb4lo9) **6**

[**What are the best practices to make shiny application run faster?-https://stackoverflow.com/questions/38092398/what-are-the-best-practices-to-make-shiny-application-run-faster**](#_gxk5cje4ux84) **6**

[**4 Tips to Make Your Shiny Dashboard Faster: https://www.r-craft.org/r-news/4-tips-to-make-your-shiny-dashboard-faster/**](#_7wzc5edeizhi) **6**

[**The Need for Optimization:https://engineering-shiny.org/when-optimize.html**](#_d9ruxb8bmylp) **6**

[**Optimizing Shiny Code**](#_by1i4pd92jow) **7**

# 

# **Chapter 1:First Day meeting resourses**

# **1.1 Introduction to Single cell:**

# [**https://www.embopress.org/doi/10.15252/msb.20188746**](https://www.embopress.org/doi/10.15252/msb.20188746)

# **1.2 A step by step guide to single cell pipeline:**

# [**https://f1000research.com/articles/5-2122**](https://f1000research.com/articles/5-2122)

# **1.3 Sanger course on single cell:**

# [**https://scrnaseq-course.cog.sanger.ac.uk/website/introduction-to-single-cell-rna-seq.html**](https://scrnaseq-course.cog.sanger.ac.uk/website/introduction-to-single-cell-rna-seq.html)

# **1.4 Valentine Svensson PhD thesis:**

# [**https://www.repository.cam.ac.uk/handle/1810/267937**](https://www.repository.cam.ac.uk/handle/1810/267937)

# **1.5 A large collection of Single cell materials (I will just ask to gloss over this):**

# **<https://www.repository.cam.ac.uk/handle/1810/267937>**

# **Chapter 2: Second Day Meeting Resources**

**Meeting Minutes (13 August 2020, Thursday)**

Cells: 28306

Samples: 12

Clusters: 7

For this dataset, the app took 20-25 seconds to load the image.

2 ways for optimization:

1. To make the code efficient

2. Using the maximum feature from HDF5

**2.1 Making R-portable**

<https://github.com/ColumbusCollaboratory/electron-quick-start/blob/master/R-Portable-Mac/bin/R>

**2.2 How to make electron-app**

<https://github.com/ColumbusCollaboratory/electron-quick-start>

**2.3 Electron video**

<https://www.youtube.com/watch?v=ARrbbviGvjc>

**2.4 HDF5 documentation**

<https://support.hdfgroup.org/HDF5/doc/index.html>

**2.5 Shiny-app link ( you need to get through VPN)**

[http://singlecell-2.ls.manchester.ac.uk:6787](http://singlecell-2.ls.manchester.ac.uk:6787/)

**Problem:**

1. Error in library(scater) : there is no package called ‘scater’ ( After installing R 3.6.3)

1. Failed to connect to singlecell-2.ls.manchester.ac.uk port 6787: No route to host

# 

# **Chapter 3: Portable Shiny App Based on Linux**

# How to Deploy Interactive R Apps with Shiny Server: <https://www.linode.com/docs/development/r/how-to-deploy-rshiny-server-on-ubuntu-and-debian/>

## Deploy a Shiny App to a Remote Server

### Deploy Your App

### Configure Shiny Server

# **[Deploying R shiny app as a standalone application](https://stackoverflow.com/questions/33513544/deploying-r-shiny-app-as-a-standalone-application):**

a)Deploying Desktop Apps with R:[**https://www.r-bloggers.com/deploying-desktop-apps-with-r/**](https://www.r-bloggers.com/deploying-desktop-apps-with-r/)

**b)** Packaging your Shiny App as an Windows desktop app:<http://blog.analytixware.com/2014/03/packaging-your-shiny-app-as-windows.html>

c)DesktopDeployR: A framework for deploying self-contained R-based applications to the desktop -<https://github.com/wleepang/DesktopDeployR>

Check this: <https://stackoverflow.com/questions/33513544/deploying-r-shiny-app-as-a-standalone-application>

# [Develop shiny apps on linux and deploying them as desktop app on a windows machine](https://stackoverflow.com/questions/45724143/develop-shiny-apps-on-linux-and-deploying-them-as-desktop-app-on-a-windows-machi): [**https://stackoverflow.com/questions/45724143/develop-shiny-apps-on-linux-and-deploying-them-as-desktop-app-on-a-windows-machi**](https://stackoverflow.com/questions/45724143/develop-shiny-apps-on-linux-and-deploying-them-as-desktop-app-on-a-windows-machi)

### Desktop DeployR: <https://oddhypothesis.blogspot.com/2016/04/desktop-deployr.html>

No 1 is the most useful resource so far

# Chapter 4: Optimizing R Shiny Application

1. Make Shiny fast by doing as little work as possible:<https://rstudio.com/resources/rstudioconf-2018/make-shiny-fast-by-doing-as-little-work-as-possible/>

a)Rprof and profvis

* “Feels slow” usually means R is busy
* Rprof: sample what R is doing
  + Computing (ggplot2, dplyr)
  + Waiting (database, network, disk)
* profvis: visualize Rprof output

# Optimizing Your Application:<https://bookdown.org/hadrien/how_to_build_a_shiny_app_from_scratch/optimizing-your-application.html>

a)Profvis — Interactive Visualizations for Profiling R Code:<https://rstudio.github.io/profvis/>

# [Improving shiny app loading speed](https://community.rstudio.com/t/improving-shiny-app-loading-speed/5126):<https://community.rstudio.com/t/improving-shiny-app-loading-speed/5126>

# Scaling and Performance Tuning with shinyapps.io: <https://shiny.rstudio.com/articles/scaling-and-tuning.html>

# What are the best practices to make shiny application run faster?-<https://stackoverflow.com/questions/38092398/what-are-the-best-practices-to-make-shiny-application-run-faster>

# 4 Tips to Make Your Shiny Dashboard Faster: <https://www.r-craft.org/r-news/4-tips-to-make-your-shiny-dashboard-faster/>

# The Need for Optimization:<https://engineering-shiny.org/when-optimize.html>

1. Performance:<https://mastering-shiny.org/performance.html>

First 3 examples are almost same

5-7 no point are better

# **Optimizing Shiny Code**

Shiny runs R code on the server side. So to be efficient, the R code computing your values and returning results also has to be optimized.

Optimizing R code is such a broad topic that it would be possible to write a full book about it, and in fact a lot of books about R already cover this topic. Instead of re-writing these books, we will try to point to some crucial resources you can refer to if you want to get started optimizing your R code.

* Efficient R programming (Gillespie and Lovelace [2017](https://engineering-shiny.org/optimizing-shiny-code.html#ref-colingillespie2017)), has a series of methods you can quickly put into practice for more efficient R code. (<https://englianhu.files.wordpress.com/2018/10/efficient-r-programming.pdf>)
* Advanced R (Wickham [2019b](https://engineering-shiny.org/optimizing-shiny-code.html#ref-hadleywickham2019)) has a chapter about optimizing R code (number 24). In the rest of this chapter, we will be focusing on how to optimize Shiny specifically. (<https://adv-r.hadley.nz/perf-improve.html>)
* Another one is <https://engineering-shiny.org/optimizing-shiny-code.html>

From . Efficient Programming Book (Chapter 3 ,page:80)

Many people who use R would not describe themselves as programmers. Instead, they tend to have advanced domain-level knowledge and understand standard R data structures such as vectors and data frames, but have little formal training in computing. Sound familiar? In that case, this chapter is for you. In this chapter, we will discuss “big picture” programming techniques. We cover general concepts and R programming techniques about code optimization, before describing idiomatic programming structures. We conclude the chapter by examining relatively easy ways of speeding up code using the compiler package and parallel processing using multiple CPUs.

**Top Five Tips for Efficient Programming**

1. Be careful never to grow vectors.

2. Vectorize code whenever possible.

3. Use factors when appropriate.

4. Avoid unnecessary computation by caching variables.

5. Byte compile packages for an easy performance boost.

Chapter 7. Efficient Optimization(page: 192)

**Top Five Tips for Efficient Optimization**

1. Before you start to optimize you code, ensure that you know where the bottleneck

lies; use a code profiler.

2. If the data in your data frame is all of the same type, consider converting it to a

matrix for a speed boost.

3. Use specialized row and column functions whenever possible.

4. The parallel package is ideal for Monte Carlo simulations.

5. For optimal performance, consider rewriting key parts of your code in C++.

**Code Profiling**

Often you will have working code, but simply want it to run faster. In some cases, it’s obvious

where the bottleneck lies. Sometimes you will guess, relying on intuition. A drawback of this

is that you could be wrong and waste time optimizing the wrong piece of code. To make slow

code run faster, it is important to first determine where the slow code lives. This is the

purpose of code profiling.

The Rprof() function is a built-in tool for profiling the execution of R expressions. At

regular time intervals, the profiler stops the R interpreter, records the current function call

stack, and saves the information to a file. The results from Rprof() are stochastic. Each time

we run a function R, the conditions have changed. Hence, each time you profile your code, the

result will be slightly different.

Unfortunately, Rprof() is not user-friendly. For this reason, we recommend using the profvis

package for profiling your R code. profvis provides an interactive graphical interface for

visualizing code-profiling data from Rprof().

(From Advanced R book) Chapter- 24 Improving performance

24.1 Introduction

We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil. Yet we should not pass up our opportunities in that critical 3%. A good programmer will not be lulled into complacency by such reasoning, he will be wise to look carefully at the critical code; but only after that code has been identified.

— Donald Knuth

Once you’ve used profiling to identify a bottleneck, you need to make it faster. It’s difficult to provide general advice on improving performance, but I try my best with four techniques that can be applied in many situations. I’ll also suggest a general strategy for performance optimisation that helps ensure that your faster code is still correct.

It’s easy to get caught up in trying to remove all bottlenecks. Don’t! Your time is valuable and is better spent analysing your data, not eliminating possible inefficiencies in your code. Be pragmatic: don’t spend hours of your time to save seconds of computer time. To enforce this advice, you should set a goal time for your code and optimise only up to that goal. This means you will not eliminate all bottlenecks. Some you will not get to because you’ve met your goal. Others you may need to pass over and accept either because there is no quick and easy solution or because the code is already well optimised and no significant improvement is possible. Accept these possibilities and move on to the next candidate.

If you’d like to learn more about the performance characteristics of the R language, I’d highly recommend Evaluating the Design of the R Language (Morandat et al. 2012). It draws conclusions by combining a modified R interpreter with a wide set of code found in the wild.

**Outline**

Section 24.2 teaches you how to organise your code to make optimisation as easy, and bug free, as possible.

Section 24.3 reminds you to look for existing solutions.

Section 24.4 emphasises the importance of being lazy: often the easiest way to make a function faster is to let it to do less work.

Section 24.5 concisely defines vectorisation, and shows you how to make the most of built-in functions.

Section 24.6 discusses the performance perils of copying data.

Section 24.7 pulls all the pieces together into a case study showing how to speed up repeated t-tests by about a thousand times.

Section 24.8 finishes the chapter with pointers to more resources that will help you write fast code.

UI up to 900 lines