UNIVERSITY OF COPENHAGEN

Variations on Stochastic Gradient Descent

Computational Statistics

Johan Larsson

Department of Mathematical Sciences, University of Copenhagen

October 23, 2024

Last Time

Today

Distributing and Organizing Code

- Reproducibility
- R packages

Today

Distributing and Organizing Code

- Reproducibility
- R packages

Course Summary

What did we actually do?

Today

Distributing and Organizing Code

- Reproducibility
- R packages

Course Summary

What did we actually do?

Oral Examination Prep

What to think of during examination

Organizing Code as an R Package

Organizing Code

Components

- Code for experiments
- Source code for functions (which we should be able to reuse)
- Tests
- Rcpp code
- Data

There is a plethora of ways to organize this. Which one to choose?

Organizing Code

Components

- Code for experiments
- Source code for functions (which we should be able to reuse)
- Tests
- Rcpp code
- Data

There is a plethora of ways to organize this. Which one to choose?

R Package

One way is to make an R package, which helps in many ways:

- Easy to connect to C++ code through Rcpp.
- Built-in support for automatic testing
- Documentation
- Declare dependencies (other packages, R version)

R Packages

Different approaches, but we will follow **R Packages** (Wickham and Bryan 2023), which is based around the **devtools** package.

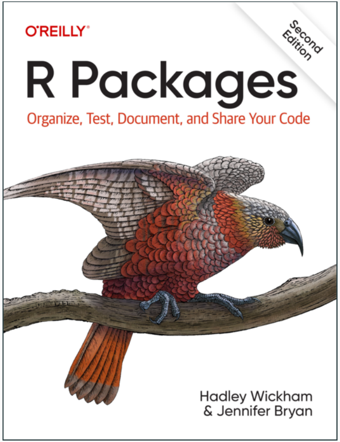

Figure 1: R Packages

Devtools

Meta-package for various helpers that aid in developing R packages (and projects).
First off, install and load **devtools**:

```r
install.packages("devtools")
library(devtools)
```

This loads other packages that will be useful for setting up your package, most importantly the **usethis** package.

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min\left((a - x_1)^2 + b(x_2 - x_1^2)^2\right).$$

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min \left((a - x_1)^2 + b(x_2 - x_1^2)^2 \right).$$

What We Will Learn

- Adding R functions to our package

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min \left((a - x_1)^2 + b(x_2 - x_1^2)^2 \right).$$

What We Will Learn

- Adding R functions to our package
- Interfacing with Rcpp

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min \left((a - x_1)^2 + b(x_2 - x_1^2)^2 \right).$$

What We Will Learn

- Adding R functions to our package
- Interfacing with Rcpp
- Testing our code

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min \left((a - x_1)^2 + b(x_2 - x_1^2)^2 \right).$$

What We Will Learn

- Adding R functions to our package
- Interfacing with Rcpp
- Testing our code
- Adding dependencies to other packages

A Toy Example

Rosenbrock Package

Let's build a simple package that solves the Rosenbrock optimization problem, i.e. find

$$x^* = \arg\min \left((a - x_1)^2 + b(x_2 - x_1^2)^2 \right).$$

What We Will Learn

- Adding R functions to our package
- Interfacing with Rcpp
- Testing our code
- Adding dependencies to other packages
- Licensing our package

A First Package

Create It
Call

```
usethis::create_package("rosenbrock")
```

or use `File > New Project > New Directory > R Package using devtools` in R Studio.

A First Package

Create It
Call

```
usethis::create_package("rosenbrock")
```

or use `File > New Project > New Directory > R Package using devtools` in R Studio.

This gives you a minimal package:

```
rosenbrock/
├── R/
├── DESCRIPTION
└── NAMESPACE
```

A First Package

Create It

Call

```
usethis::create_package("rosenbrock")
```

or use `File > New Project > New Directory > R Package using devtools` in R Studio.

This gives you a minimal package:

```
rosenbrock/
├── R/
├── DESCRIPTION
└── NAMESPACE
```

You may also have `.Rbuildignore` and `.rosenbrock.Rproj` depending on how you created the package.

A First Package

Create It

Call

```r
usethis::create_package("rosenbrock")
```

or use `File > New Project > New Directory > R Package using devtools` in R Studio.

This gives you a minimal package:

```
rosenbrock/
├── R/
├── DESCRIPTION
└── NAMESPACE
```

You may also have `.Rbuildignore` and `.rosenbrock.Rproj` depending on how you created the package.

Install It

Open up the package in your editor (R Studio[a]).

```r
devtools::install()
```

Voila, you have made an R package!

[a] In which case it should alread be opened.

Thank you!

R Code

`.R/`

- All R code should live in `.R`-files in `R/`.
- These files should (almost) always contain only functions.
- Many ways to organize your files: one function per file, all functions of a certain S3 class in one file etc.

R Code

`.R/`

- All R code should live in `.R`-files in `R/`.
- These files should (almost) always contain only functions.
- Many ways to organize your files: one function per file, all functions of a certain S3 class in one file etc.

Let's create a first file: `R/objective.R`. Use `usethis::use_r("objective")` and insert this:

```
objective <- function(x, a = 1, b = 100) {
  (a - x[1])^2 + b * (x[2] - x[1]^2)^2
}
```

Workflow

We have created a first R file, but how do we use it? Two major options:

`devtools::install()`
Installs the package, like calling
`install.packages()`.

Workflow

We have created a first R file, but how do we use it? Two major options:

`devtools::install()`
Installs the package, like calling `install.packages()`.

Robust but slow. Need to call `library(rosenbrock)` to load package[a].

[a] Done automatically in R Studio

Workflow

We have created a first R file, but how do we use it? Two major options:

`devtools::install()`

Installs the package, like calling `install.packages()`.

Robust but slow. Need to call `library(rosenbrock)` to load package[a].

[a] Done automatically in R Studio

`devtools::load_all()`

Sources all of your code.

Quick but not as robust.

Workflow

We have created a first R file, but how do we use it? Two major options:

`devtools::install()`

Installs the package, like calling `install.packages()`.

Robust but slow. Need to call `library(rosenbrock)` to load package[a].

[a] Done automatically in R Studio

`devtools::load_all()`

Sources all of your code.

Quick but not as robust.

Try It

Try both options and see if you can call your newly defined function, `objective()`.

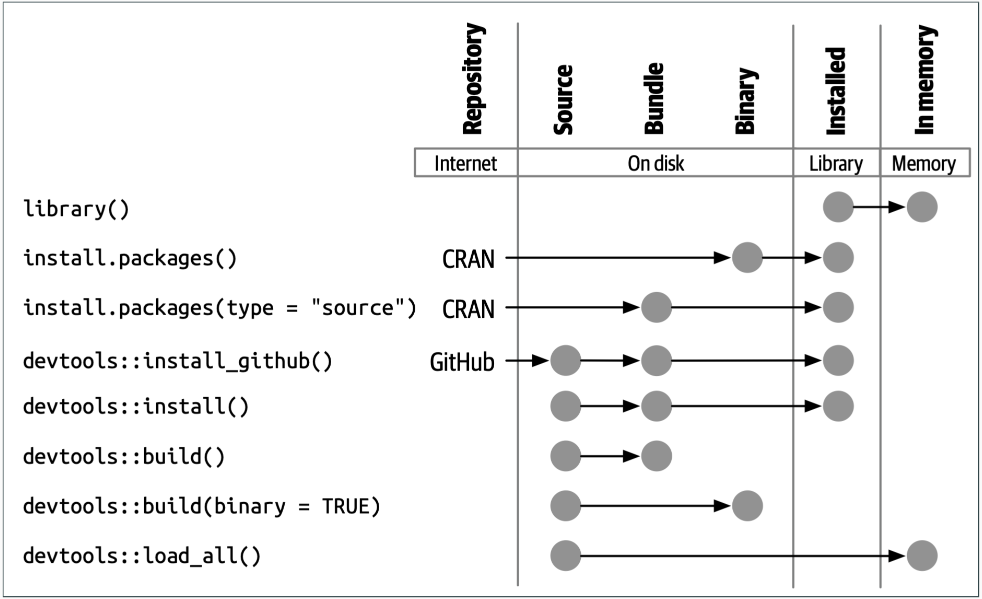

Figure 2: The various states of a package and how to move between them.

Exporting Functions

If you called `devtools::load_all()` then everything is sourced and you can just call `objective()` directly.

But if you use `devtools::install()` and `library(rosenbrock)`, the you would need to use `rosenbrock:::objective()`. The reason is that the function is not yet exported.

Exporting Functions

If you called `devtools::load_all()` then everything is sourced and you can just call `objective()` directly.

But if you use `devtools::install()` and `library(rosenbrock)`, the you would need to use `rosenbrock::objective()`. The reason is that the function is not yet exported.

`NAMESPACE`
Decides what functions you want exported. But right now it just contains a comment:

```
# Generated by roxygen2: do not edit by hand
```

Exporting Functions

If you called `devtools::load_all()` then everything is sourced and you can just call `objective()` directly.

But if you use `devtools::install()` and `library(rosenbrock)`, the you would need to use `rosenbrock::objective()`. The reason is that the function is not yet exported.

NAMESPACE
Decides what functions you want exported. But right now it just contains a comment:

```
# Generated by roxygen2: do not edit by hand
```

If you want to just export everything, you can remove this file and recreate it with this content:

```
exportPattern("^[[:alpha:]]+")
```

roxygen2

roxygen2 is a package that helps with package documentation[1], but it can also be used for handling the namespace.

[1] More on this later.

roxygen2

roxygen2 is a package that helps with package documentation[1], but it can also be used for handling the namespace.

To export a function, you need to place a special roxygen2 comment just before the function:

```r
#' @export
```

[1]More on this later.

roxygen2 is a package that helps with package documentation[1], but it can also be used for handling the namespace.

To export a function, you need to place a special roxygen2 comment just before the function:

```
#' @export
```

Go ahead and place this before your `objective()` definition. Then run `devtools::document()` to roxygenize your package.

[1]More on this later.

roxygen2

roxygen2 is a package that helps with package documentation[1], but it can also be used for handling the namespace.

To export a function, you need to place a special roxygen2 comment just before the function:

```
#' @export
```

Go ahead and place this before your `objective()` definition. Then run `devtools::document()` to roxygenize your package.

Now `NAMESPACE` will (should) contain this:

```
export(objective)
```

[1] More on this later.

roxygen2 is a package that helps with package documentation[1], but it can also be used for handling the namespace.

To export a function, you need to place a special roxygen2 comment just before the function:

```
#' @export
```

Go ahead and place this before your `objective()` definition. Then run `devtools::document()` to roxygenize your package.

Now `NAMESPACE` will (should) contain this:

```
export(objective)
```

Reinstall the package and see if you can call `objective()` after loading it.

[1] More on this later.

Tests

testthat

- We have already encountered **testthat** for writing tests in a formalized way.
- But **testthat** was actually written especially for packages.

testthat

- We have already encountered **testthat** for writing tests in a formalized way.
- But **testthat** was actually written especially for packages.

Let's start using **testthat** with our package:

```
usethis::use_testthat()
```

testthat

- We have already encountered **testthat** for writing tests in a formalized way.
- But **testthat** was actually written especially for packages.

Let's start using **testthat** with our package:

```
usethis::use_testthat()
```

This creates some new files and directories:

```
rosenbrock/
└── tests/
    ├── testthat/
    │   └── test-<some_fun>.R .................... Your test file for some_fun()
    └── testthat.R
```

A First Simple Test

For the Rosenbrock function, $f^* = f(a, a^2) = f(1,1) = 0$. Let's make sure this is the case for us too!

A First Simple Test

For the Rosenbrock function, $f^* = f(a, a^2) = f(1, 1) = 0$. Let's make sure this is the case for us too!

To create a test, we can use `usethis::use_test()`.

A First Simple Test

For the Rosenbrock function, $f^* = f(a, a^2) = f(1, 1) = 0$. Let's make sure this is the case for us too!

To create a test, we can use `usethis::use_test()`.

Call `use_test("objective")`[2] and insert this:

```r
test_that("multiplication works", {
  # add a test using expect_equal()
})
```

[2] It's good practice to name the test file the same as the file where the function you're testing is defined.

A First Simple Test

For the Rosenbrock function, $f^* = f(a, a^2) = f(1, 1) = 0$. Let's make sure this is the case for us too!

To create a test, we can use `usethis::use_test()`.

Call `use_test("objective")`[2] and insert this:

```
test_that("multiplication works", {
  # add a test using expect_equal()
})
```

Check That Everything Works

Run `devtools::test()`, and hopefully see:

```
[ FAIL 0 | WARN 0 | SKIP 0 | PASS 1 ]
```

[2]It's good practice to name the test file the same as the file where the function you're testing is defined.

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

ERROR Major problem with your package

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

ERROR Major problem with your package
WARNING Something that is most likely not great but not critical

Checking

`R CMD check`

R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

ERROR	Major problem with your package
WARNING	Something that is most likely not great but not critical
NOTE	Typically small issues with your package

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

ERROR Major problem with your package
WARNING Something that is most likely not great but not critical
NOTE Typically small issues with your package

Checking

`R CMD check`
R contains functionality for checking that your package is built correctly and you can access this functionality through `devtools::check()`.

No requirement that your package needs to pass these checks (if you're using it as a project), but it's good practice to make sure it does.

ERROR	Major problem with your package
WARNING	Something that is most likely not great but not critical
NOTE	Typically small issues with your package

Now run `devtools::check()`. Is there a problem? Yes, let's fix it!

The metadata for your package lives in DESCRIPTION. Right now it looks like this:

```
Package: rosenbrock
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person("First", "Last", , "first.last@example.com", role =
        c("aut", "cre"),
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends
    to pick a
    license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.3.2
```

Metadata

The metadata for your package lives in DESCRIPTION. Right now it looks like this:

```
Package: rosenbrock
Title: What the Package Does (One Line, Title Case)
Version: 0.0.0.9000
Authors@R:
    person("First", "Last", , "first.last@example.com", role =
        c("aut", "cre"),
           comment = c(ORCID = "YOUR-ORCID-ID"))
Description: What the package does (one paragraph).
License: `use_mit_license()`, `use_gpl3_license()` or friends
   to pick a
    license
Encoding: UTF-8
Roxygen: list(markdown = TRUE)
RoxygenNote: 7.3.2
```

For now we'll leave most of these files alone, but let's fix one thing: the license

Licensing

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.
- If you do not provide a license, this generally means that nobody is allowed to copy, distribute, or modify your code.

Licensing

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.
- If you do not provide a license, this generally means that nobody is allowed to copy, distribute, or modify your code.
- If you have other contributors, then "nobody" includes you too!

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.
- If you do not provide a license, this generally means that nobody is allowed to copy, distribute, or modify your code.
- If you have other contributors, then “nobody” includes you too!

Licensing

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.
- If you do not provide a license, this generally means that nobody is allowed to copy, distribute, or modify your code.
- If you have other contributors, then "nobody" includes you too!

Choosing a License

So we need to pick a license: for now we'll pick the MIT license.[3]

[3] Read more about picking a license at https://choosealicense.com.

Licensing

Why Do You Need a License?

- Licensing software tells other people about how they are allowed to reuse your code.
- If you do not provide a license, this generally means that nobody is allowed to copy, distribute, or modify your code.
- If you have other contributors, then “nobody” includes you too!

Choosing a License

So we need to pick a license: for now we’ll pick the MIT license.[3]

```
usethis::use_mit_license()
```

This will add new files to your package: `LICENSE`, `LICENSE.md`, and modify `DESCRIPTION`, in which you should see:

```
License: MIT + file LICENSE
```

[3] Read more about picking a license at https://choosealicense.com.

Dependencies

In R packages, you make dependencies explicit, defined in `DESCRIPTION`

Dependencies

In R packages, you make dependencies explicit, defined in `DESCRIPTION`

Gradient
Let's say that we want to compute the gradient for the Rosenbrock function.

Dependencies

In R packages, you make dependencies explicit, defined in `DESCRIPTION`

Gradient
Let's say that we want to compute the gradient for the Rosenbrock function.

One way to do so is to use numerical differentiation through the **numDeriv** package:

```
gradient <- function(x, a = 1, b = 100) {
  numDeriv::grad(objective, x, a = a, b = b)
}
```

Now our package depends on **numDeriv**, so we need to add it to `DESCRIPTION`:

```
usethis::use_package("numDeriv")
```

Dependencies

In R packages, you make dependencies explicit, defined in `DESCRIPTION`

Gradient
Let's say that we want to compute the gradient for the Rosenbrock function.

One way to do so is to use numerical differentiation through the **numDeriv** package:

```
gradient <- function(x, a = 1, b = 100) {
  numDeriv::grad(objective, x, a = a, b = b)
}
```

Now our package depends on **numDeriv**, so we need to add it to `DESCRIPTION`:

```
usethis::use_package("numDeriv")
```

In `DESCRIPTION`, you should now see this:

```
Imports:
  numDeriv
```

Rcpp

Rcpp works best in a package:

- No more manual sourcing (no need to call `Rcpp::sourceCpp()`)
- You don't need to add directives for dependencies to **RcppArmadillo** and other packages.

Rcpp works best in a package:

- No more manual sourcing (no need to call `Rcpp::sourceCpp()`)
- You don't need to add directives for dependencies to **RcppArmadillo** and other packages.

We will rely on **roxygen2**. First, call

```
usethis::use_package_doc()
```

to set up a package doc file in `R/rosenbrock-package.R`.

Rcpp

Rcpp works best in a package:

- No more manual sourcing (no need to call `Rcpp::sourceCpp()`)
- You don't need to add directives for dependencies to **RcppArmadillo** and other packages.

We will rely on **roxygen2**. First, call

```
usethis::use_package_doc()
```

to set up a package doc file in `R/rosenbrock-package.R`.

Then use `usethis::use_rcpp()` to put the pieces in place:

```
rosenbrock/
└─ src/
   └─ slop-package.cpp/
```

Rcpp

Rcpp works best in a package:

- No more manual sourcing (no need to call `Rcpp::sourceCpp()`)
- You don't need to add directives for dependencies to **RcppArmadillo** and other packages.

We will rely on **roxygen2**. First, call

```
usethis::use_package_doc()
```

to set up a package doc file in `R/rosenbrock-package.R`.

Then use `usethis::use_rcpp()` to put the pieces in place:

```
rosenbrock/
└── src/
    └── slop-package.cpp/
```

Now just need to run `devtools::document()` and `devtools::load_all()` or `devtools::install()` and no your code is available (but not exported).

Rcpp

Rcpp works best in a package:

- No more manual sourcing (no need to call `Rcpp::sourceCpp()`)
- You don't need to add directives for dependencies to **RcppArmadillo** and other packages.

We will rely on **roxygen2**. First, call

```
usethis::use_package_doc()
```

to set up a package doc file in `R/rosenbrock-package.R`.

Then use `usethis::use_rcpp()` to put the pieces in place:

```
rosenbrock/
└─src/
   └─slop-package.cpp/
```

Now just need to run `devtools::document()` and `devtools::load_all()` or `devtools::install()` and no your code is available (but not exported).

To export, easiest is to write an R wrapper.

Documentation

Writing documentation is useful for others who want to use your code as well as for your future self.

Documentation

Writing documentation is useful for others who want to use your code as well as for your future self.

Many aspects of documentation

- Comments in code
- Manual (help files)
- Long-form articles (vignettes)

Documentation

Writing documentation is useful for others who want to use your code as well as for your future self.

Many aspects of documentation

- Comments in code
- Manual (help files)
- Long-form articles (vignettes)

For the manual part, you can use **roxygen2** (it's main purpose).

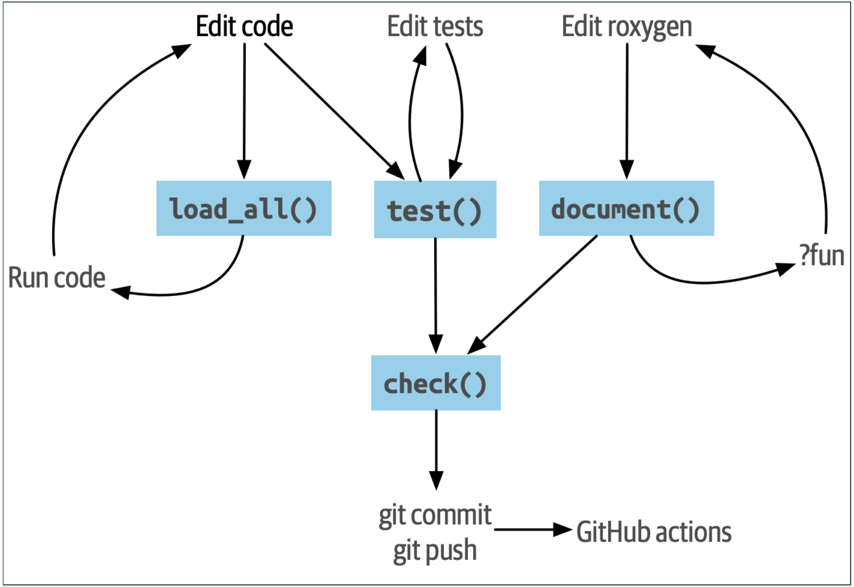

Figure 3: The whole game

Exercise: Optimize the Rosenbrock Function

Write a gradient descent (or stochastic gradient descent) that minimizes the rosenbrock function.

Exercise: Optimize the Rosenbrock Function

Write a gradient descent (or stochastic gradient descent) that minimizes the rosenbrock function.

Write the code in Rcpp. If you want, you can first write it in R to see that everything is working, and then port it.

Exercise: Optimize the Rosenbrock Function

Write a gradient descent (or stochastic gradient descent) that minimizes the rosenbrock function.

Write the code in Rcpp. If you want, you can first write it in R to see that everything is working, and then port it.

Feel free to use generative AI to write the code.

Exercise: Optimize the Rosenbrock Function

Write a gradient descent (or stochastic gradient descent) that minimizes the rosenbrock function.

Write the code in Rcpp. If you want, you can first write it in R to see that everything is working, and then port it.

Feel free to use generative AI to write the code.

Export everything and document the package.

What We Didn't Cover

- Version control through git and github

What We Didn't Cover

- Version control through git and github
- Metadata

What We Didn't Cover

- Version control through git and github
- Metadata
- Publishing to CRAN

Oral Examination Prep

The Five Points

Remember the five points:

- How can you test that your implementation is correct?
- Can you implement alternative solutions?
- Can the code be restructured e.g. by modularization, abstraction or object oriented programming to improve generality, extendability and readability?
- How does the implementation perform (benchmarking)?
- Where are the bottlenecks (profiling), and what can you do about them?

Course Summary

Course Summary

Statistical Topics

Smoothing Kernel density smoothing and splines (topic 1)

Course Summary

Statistical Topics

Smoothing Kernel density smoothing and splines (topic 1)

Simulation MC methods: rejection and importance sampling (topic 2)

Course Summary

Statistical Topics

Smoothing Kernel density smoothing and splines (topic 1)

Simulation MC methods: rejection and importance sampling (topic 2)

Optimization The EM algorithm (topic 3), gradient descent and stochastic optimization (topic 4)

Course Summary

Statistical Topics

Smoothing Kernel density smoothing and splines (topic 1)

Simulation MC methods: rejection and importance sampling (topic 2)

Optimization The EM algorithm (topic 3), gradient descent and stochastic optimization (topic 4)

Course Summary

Statistical Topics

Smoothing Kernel density smoothing and splines (topic 1)

Simulation MC methods: rejection and importance sampling (topic 2)

Optimization The EM algorithm (topic 3), gradient descent and stochastic optimization (topic 4)

Computational Topics

- Debugging
- Profiling
- Benchmarking
- Debugging
- Writing performant code