

Today

- Recap & questions from homework
- Functions
- Coding a descent algorithm (training)
 - using `optim()`

Git skills

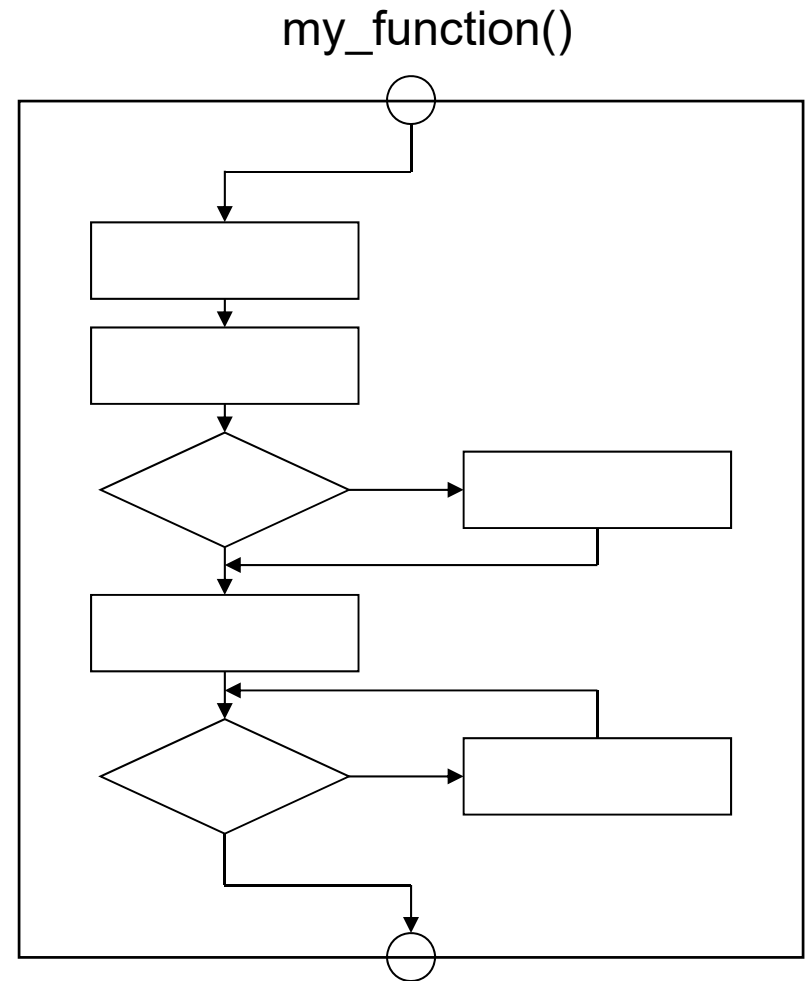
- `git amend`
- `.gitignore`
- `gitgui`
- `gitk`

Fit ecological model

- Natural process culture of data science
- Paramecium logistic growth
- Parameters: r , K , $N(0)$
- Grid search

Programming: functions

- A function **encapsulates an algorithm**
- Functions break a program down into **modules**
- Modularized programs are easier to write, debug, maintain, and modify
- Functions make algorithms easier to **reuse**



Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {  
  expression  
  return(object)  
}
```

Class style

← explicit return

← indent (4 spaces)

← closing brace aligns with first letter of function name

Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {  
  expression  
  return(object)  
}
```

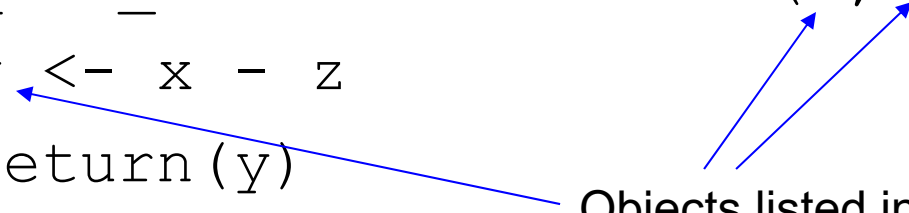
```
diff_two_nums <- function(x, z) {  
  y <- x - z  
  return(y)  
}
```

Making a function in R

- ?"function" – only the bare bones

```
function_name <- function(arguments) {  
  expression  
  return(object)  
}
```

```
diff_two_nums <- function(x, z) {  
  y <- x - z  
  return(y)  
}
```



Objects listed in the arguments or defined in the function can only be seen inside the function. These are called **local variables**.
Concept: **scope**.

Scope

- See examples in functions.R
- Good programming practice: **avoid global variables**
 - Define **local variables** by including in argument list or initializing within the function
 - Global variables make programs harder to maintain and debug

Make a function

```
function_name <- (arguments) {  
  expression  
  return(object)  
}
```

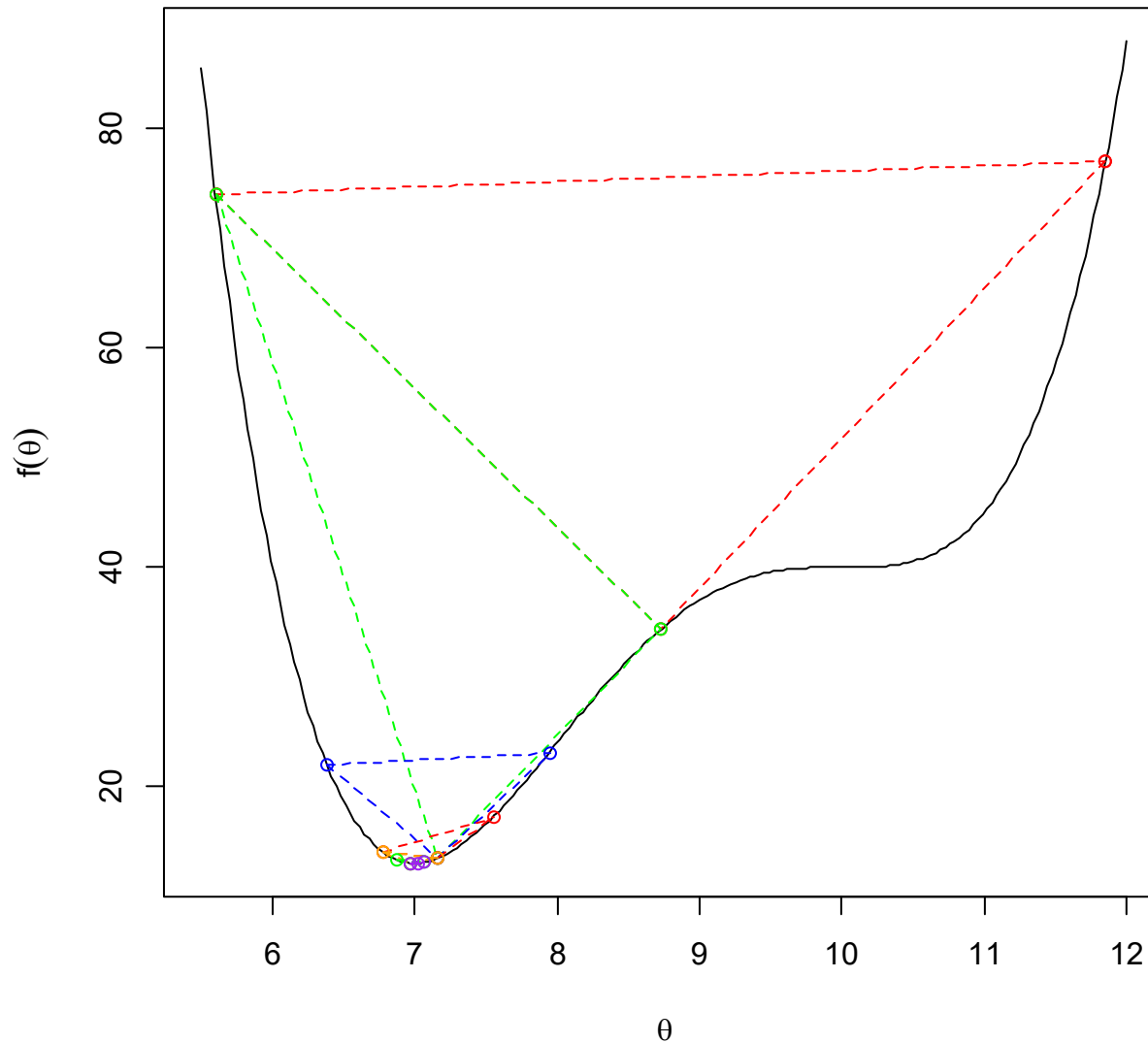
Exercise:

Make a function to calculate the linear model given the model parameters and a vector of x data. In other words, turn the following into a function:

$$y \leftarrow b_0 + b_1 * x$$

Descent algorithms

Optimize θ : find θ such that $f(\theta)$ is minimum



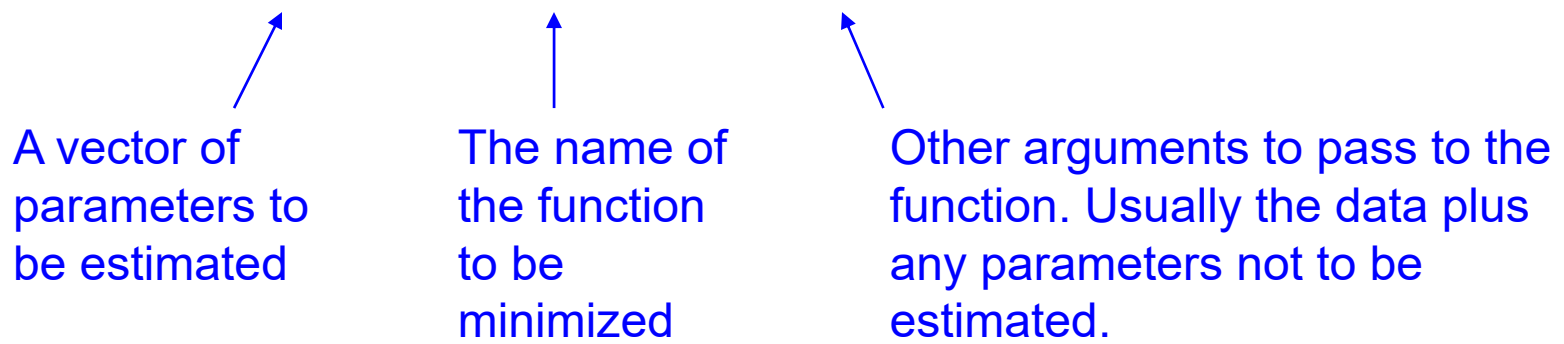
Narrowing in:

keep changing
parameters in the
direction that leads to
lower SSQ

optim()

- Has various descent and Monte Carlo methods
- **Nelder-Mead** algorithm is default
(method="Nelder-Mead")

```
optim(par, fn, ...)
```



A vector of
parameters to
be estimated

The name of
the function
to be
minimized

Other arguments to pass to the
function. Usually the data plus
any parameters not to be
estimated.

Training models: general recipe

- 1) biology function
 - complex mechanistic to abstract pattern
- 2) error function
 - e.g. SSQ: distance of the model from the data
$$\text{sum}((\text{observed} - \text{predicted})^2)$$
- 3) optimize
 - find biology parameters that minimize the error
- This recipe is the same no matter how complicated the process model or error function

Code (train_ssq_optim.R)

Biology function (linear)

```
linmod <- function(b_0, b_1, x) {  
  y <- b_0 + b_1 * x  
  return(y)  
}
```

Parameters are first argument

Response data

Error function (SSQ)

```
ssq_linmod <- function(p, y, x) {  
  y_pred <- linmod(b_0=p[1], b_1=p[2], x)  
  e <- y - y_pred  
  ssq <- sum(e^2)  
  return(ssq)  
}
```

Auxiliary data

Use the biology
function to get
predicted values

Compare predicted
to the data

"Unpack" the parameters
(self documenting)

Use optim to optimize error function

```
par <- c(b_0_start, b_1_start) Starting values for parameters  
fit <- optim(par, ssq_linmod, y=data$y, x=data$x)
```

Need "=" sign