STAT 33B Workbook 10

Yuanrui Zhu (3034615728)

Nov 5, 2020

This workbook is due **Nov 5, 2020** by 11:59pm PT.

The workbook is organized into sections that correspond to the lecture videos for the week. Watch a video, then do the corresponding exercises *before* moving on to the next video.

Workbooks are graded for completeness, so as long as you make a clear effort to solve each problem, you'll get full credit. That said, make sure you understand the concepts here, because they're likely to reappear in homeworks, quizzes, and later lectures.

As you work, write your answers in this notebook. Answer questions with complete sentences, and put code in code chunks. You can make as many new code chunks as you like.

In the notebook, you can run the line of code where the cursor is by pressing `Ctrl` + `Enter` on Windows or `Cmd` + `Enter` on Mac OS X. You can run an entire code chunk by clicking on the green arrow in the upper right corner of the code chunk.

Please do not delete the exercises already in this notebook, because it may interfere with our grading tools.

You need to submit your work in two places:

- Submit this Rmd file with your edits on bCourses.
- Knit and submit the generated PDF file on Gradescope.

If you have any last-minute trouble knitting, **DON'T PANIC**. Submit your Rmd file on time and follow up in office hours or on Piazza to sort out the PDF.

Attributes

Watch the "Attributes" lecture video.

No exercises for this section.

The S3 Object System

Watch the "The S3 Object System" lecture video.

No exercises for this section.

S3 Objects

Watch the "S3 Objects" lecture video.

The goal of the next 2 exercises is to create an S3 class `polynomial` that can represent polynomials of any degree.

Exercise 1

Create a function `poly` that constructs an S3 object with class `polynomial`. The `poly` function should accepts any number of coefficients as arguments, in order from lowest to highest degree. The number of coefficients provided determines the degree of the polynomial.

The `polynomial` object should have a field called `coefs` that stores the coefficients as a vector, in order from lowest to highest degree.

For example, the call `poly(3, 1, 0, 2)` should construct a `polynomial` object that represents the polynomial $3 + x + 2x^3$. The `coefs` field should contain the vector `(3, 1, 0, 2)`.

If the `poly` function is called without arguments, or if any of the coefficients are not numbers, the function should produce an error message instead of returning a `polynomial` object.

Hint 1: You'll need to use the `...` parameter here.

Hint 2: You can convert a list to a vector with `unlist()` or `as.vector()`.

YOUR ANSWER GOES HERE:

```
poly = function(...) {
  temp = list(...)
  polynomial = list(coefs = as.vector(temp))
  structure(polynomial, class = "poly")
}
x = poly(3, 1, 0, 3)
```

S3 Generics & Methods

Watch the "S3 Generics & Methods" lecture video.

Exercise 2

Create a `print` method for `polynomial` objects that prints them similar to how you would write a polynomial mathematically. That is:

- Terms should be printed from lowest to highest degree.
- Every coefficient should be rounded to show at most 2 decimal places.
- It is okay to print terms where the coefficient is 0.
- The first term should just be a number. Only negative numbers should begin with a sign. Examples: `-0.67` and `2`.
- The second term should:
  - Begin with two spaces and a sign
  - End with `x`
  - Examples: `-0.67x` and `+2x`.
- The third term and all subsequent terms should:
  - Begin with two spaces and a sign
  - End with `x^d`, where `d` is the degree of the term

  - Examples: `-0.67x^2` and `+2x^4`

For example, the object created by `polynomial(2/3, 0, -1)` should print as `0.67  +0x  -1x^2`.

Test your print method by for several different `polynomial` objects.

Your method should work even if `print` is not called explicitly. For example, this code should implicitly call your `print` method:

```
p = poly(1, 3, 2)
p
```

Hint 1: Make sure your method's parameters match the generic ***`print`*** *function's parameters.*

Hint 2: Use vectorized operations to assemble the output string. Create a prefix, coefficient, and suffix string for each term. Then use ***`paste`*** *to paste everything together.*

Hint 3: The ***`format`*** *function can convert a number to a string. The* ***`digits`***, ***`dropOtrailing`***, *and* ***`trim`*** *parameters are especially useful for this exercise.*

Hint 4: Until your ***`print`*** *function is working correctly, you may want to give it a different name. That way you can still see R's default printout for* ***`polynomial`*** *objects while testing your function.*

YOUR ANSWER GOES HERE:

```
print = function(x) {
  UseMethod("print")
}

print.poly = function(x) {
  power = 0
  msg = ""
  while (power < length(x$coefs)) {
    if (power == 0) {
      msg = paste(msg, toString(x$coefs[power + 1]))
    } else {
    msg = paste(msg, "+", toString(x$coefs[power + 1]), "x^", toString(power))
    }
    power = power + 1
  }
  msg
}
print(x)
```

```
## [1] " 3 + 1 x^ 1 + 0 x^ 2 + 3 x^ 3"
```

Other Object Systems

Watch the "Other Object Systems" lecture video.

Exercise 3

Read Chapter 16 of Advanced R.

Is threading state a problem in Homework 5, Exercise 1? Why or why not?

Based on this, is R6 strictly necessary to avoid the threading state problem?

Hint: For more context, read Chapter 14 as well.

YOUR ANSWER GOES HERE:

Yes, I believe R6 is strictly necessary to avoid the threading state problem, because $pop() can modify the object in place, and return only the top-most value. R3 is only able to modify the functions in the outer environment and thus need a pass-in argument. There is not a concept of "instance attribute" in R3, which makes it different to do reassignment.