Lab 1

Jonathan Eng

11:59PM February 8, 2020

You should have RStudio installed to edit this file. You will write code in places marked "TO-DO" to complete the problems. Some of this will be a pure programming assignment. The tools for the solutions to these problems can be found in the class practice lectures. I want you to use the methods I taught you, not for you to google and come up with whatever works. You won't learn that way.

To "hand in" the homework, you should compile or publish this file into a PDF that includes output of your code. Once it's done, push by the deadline to your repository in a directory called "labs".

- Print out the numerical constant pi with ten digits after the decimal point using the internal constant `pi`.

```
options(digits=11)
pi
```

```
## [1] 3.1415926536
```

- Sum up the first 100 terms of the series $1 + 1/2 + 1/4 + 1/8 + \ldots$

```
sum(1 / 2^(0:99))
```

```
## [1] 2
```

- Find the product of the first 20 terms of `1/3 * 1/6 * 1/9 *` $\ldots$

```
prod(1 / seq(3, 60, by = 3))
```

```
## [1] 1.1788275817e-28
```

- Find the product of the first 500 terms of `1 * 1/2 * 1/4 * 1/8 *` $\ldots$

```
prod(1 / 2^(0:499))
```

```
## [1] 0
```

Is this answer *exactly* correct?

NO - because of floating point error

- Figure out a means to express the answer more exactly. Not compute exactly, but express more exactly.

```
exp(sum(log(1 / 2^(0:499))))
```

```
## [1] 0
```

- Create the sequence `x = [Inf, 20, 18, ..., -20]`.

```
x = c(Inf, seq(from = 20, to = -20, by = -2))
x
```

```
##  [1] Inf  20  18  16  14  12  10   8   6   4   2   0  -2  -4  -6  -8 -10 -12 -14
## [20] -16 -18 -20
```

Create the sequence `x = [log_3(Inf), log_3(100), log_3(98), ... log_3(-20)]`.

```
x = log(c(Inf, seq( from = 100, to = -20, by = -2)), base = 3)
```

```
## Warning: NaNs produced
```

```
x
```

```
##  [1]           Inf 4.19180654858 4.17341725189 4.15464876786 4.13548512895
##  [6] 4.11590933734 4.09590327429 4.07544759936 4.05452163807 4.03310325630
## [11] 4.01116871959 3.98869253500 3.96564727304 3.94200336639 3.91772888179
## [16] 3.89278926071 3.86714702345 3.84076143031 3.81358809222 3.78557852143
## [21] 3.75667961083 3.72683302786 3.69597450568 3.66403300988 3.63092975357
## [26] 3.59657702662 3.56087679501 3.52371901429 3.48497958377 3.44451784579
## [31] 3.40217350273 3.35776278143 3.31107361282 3.26185950714 3.20983167673
## [36] 3.15464876786 3.09590327429 3.03310325630 2.96564727304 2.89278926071
## [41] 2.81358809222 2.72683302786 2.63092975357 2.52371901429 2.40217350273
## [46] 2.26185950714 2.09590327429 1.89278926071 1.63092975357 1.26185950714
## [51] 0.63092975357          -Inf           NaN           NaN           NaN
## [56]           NaN           NaN           NaN           NaN           NaN
## [61]           NaN           NaN
```

Comment on the appropriateness of the non-numeric values.

NaNs come from log of negative numbers log3(-number) and Inf comes from log3(Inf) and -Inf comes from log3(0)

- Create a vector of booleans where the entry is true if `x[i]` is positive and finite.

```
is_positive_real = (x > 0) & (x != Inf) & (!is.nan(x))
is_positive_real
```

```
##  [1] FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [13]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [25]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [37]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [49]  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
## [61] FALSE FALSE
```

- Locate the indices of the non-numbers in this vector. Hint: use the `which` function.

```
which(!is_positive_real)
```

```
##  [1]  1 52 53 54 55 56 57 58 59 60 61 62
```

- Locate the indices of the infinite quantities in this vector. Hint: use the `which` function.

```
which(x == Inf | x == -Inf)
```

```
## [1]  1 52
```

- Locate the indices of the min and max in this vector. Hint: use the `which.min` and `which.max` functions.

```
y = x
y[is.infinite(y)] = NA
which.min(y)
```

```
## [1] 51
```

```
which.max(y)
```

```
## [1] 2
```

- Count the number of unique values in `x`.

```
length(unique(x))
```

```
## [1] 53
```

- Cast `x` to a factor. Do the number of levels make sense?

```
factor(x)
```

```
##  [1] Inf               4.19180654857877  4.1734172518943   4.15464876785729
##  [5] 4.13548512895119  4.11590933734319  4.09590327428938  4.07544759935851
##  [9] 4.05452163806914  4.03310325630434  4.01116871959141  3.98869253500376
## [13] 3.96564727304425  3.94200336638929  3.91772888178973  3.89278926071437
## [17] 3.86714702345081  3.84076143030548  3.81358809221559  3.78557852142874
## [21] 3.75667961082847  3.72683302786084  3.69597450568212  3.66403300987579
## [25] 3.63092975357146  3.59657702661571  3.56087679500731  3.52371901428583
## [29] 3.48497958377173  3.44451784578705  3.40217350273288  3.3577627814323
## [33] 3.31107361281783  3.26185950714291  3.20983167673402  3.15464876785729
## [37] 3.09590327428938  3.03310325630434  2.96564727304425  2.89278926071437
## [41] 2.8135880922156   2.72683302786084  2.63092975357146  2.52371901428583
## [45] 2.40217350273288  2.26185950714291  2.09590327428938  1.89278926071437
## [49] 1.63092975357146  1.26185950714291  0.630929753571457 -Inf
## [53] NaN               NaN               NaN               NaN
## [57] NaN               NaN               NaN               NaN
## [61] NaN               NaN
## 53 Levels: -Inf 0.630929753571457 1.26185950714291 ... NaN
```

Casting x to a factor gave each unique number its own level, which does make sense because the numbers are different values, though the usefulness of having similar values be considered different levels would vary on the purpose.

- Cast `x` to integers. What do we learn about R's infinity representation in the integer data type?

```
as.integer(x)
```

```
## Warning: NAs introduced by coercion to integer range
```

```
##  [1] NA  4  4  4  4  4  4  4  4  4  4  3  3  3  3  3  3  3  3  3  3  3  3  3
## [26]  3  3  3  3  3  3  3  3  3  3  3  3  2  2  2  2  2  2  2  2  2  1  1  1
## [51]  0 NA NA NA NA NA NA NA NA NA NA NA
```

They are converted to NA - they don't exist.

- Use `x` to create a new vector `y` containing only real numbers.

```
y <- x[is.finite(x)]
y
```

```
##  [1] 4.19180654858 4.17341725189 4.15464876786 4.13548512895 4.11590933734
##  [6] 4.09590327429 4.07544759936 4.05452163807 4.03310325630 4.01116871959
## [11] 3.98869253500 3.96564727304 3.94200336639 3.91772888179 3.89278926071
## [16] 3.86714702345 3.84076143031 3.81358809222 3.78557852143 3.75667961083
## [21] 3.72683302786 3.69597450568 3.66403300988 3.63092975357 3.59657702662
## [26] 3.56087679501 3.52371901429 3.48497958377 3.44451784579 3.40217350273
## [31] 3.35776278143 3.31107361282 3.26185950714 3.20983167673 3.15464876786
## [36] 3.09590327429 3.03310325630 2.96564727304 2.89278926071 2.81358809222
## [41] 2.72683302786 2.63092975357 2.52371901429 2.40217350273 2.26185950714
## [46] 2.09590327429 1.89278926071 1.63092975357 1.26185950714 0.63092975357
```

- Use the left rectangle method to numerically integrate x^2 from 0 to 1 with rectangle size 1e-6.

```
sum(seq(0, 1 - 1e-6, by = 1e-6)^2) * 1e-6
```

```
## [1] 0.33333283333
```

- Calculate the average of 100 realizations of standard Bernoullis in one line using the `sample` function.

```
mean(sample(c(0,1), size = 100, replace = TRUE))
```

```
## [1] 0.6
```

- Calculate the average of 500 realizations of Bernoullis with p = 0.9 in one line using the `sample` function.

```
mean(sample(c(0,1), size = 500, replace = TRUE, prob = c(.1, .9)))
```

```
## [1] 0.9
```

- In class we considered a variable `x_3` which measured "criminality". We imagined L = 4 levels "none", "infraction", "misdimeanor" and "felony". Create a variable `x3` here with 100 random elements (equally probable). Create it as a nominal (i.e. unordered) factor.

```
x_3 = factor(sample(1:4, 100, replace = TRUE), labels = c("none", "infraction", "misdimeanor", "felony"))
x_3
```

```
##   [1] infraction  felony      misdimeanor felony      felony      infraction
##   [7] infraction  felony      none        none        felony      infraction
##  [13] misdimeanor felony      infraction  infraction  none        none
##  [19] infraction  none        infraction  infraction  none        infraction
##  [25] infraction  misdimeanor misdimeanor none        felony      felony
##  [31] misdimeanor infraction  misdimeanor felony      misdimeanor none
##  [37] felony      none        misdimeanor infraction  felony      felony
##  [43] none        misdimeanor infraction  none        misdimeanor none
##  [49] misdimeanor infraction  felony      infraction  felony      infraction
##  [55] misdimeanor felony      felony      felony      none        misdimeanor
##  [61] infraction  none        none        misdimeanor misdimeanor misdimeanor
##  [67] felony      infraction  infraction  infraction  infraction  misdimeanor
##  [73] none        infraction  none        infraction  infraction  misdimeanor
##  [79] felony      misdimeanor infraction  infraction  infraction  infraction
##  [85] infraction  infraction  felony      none        misdimeanor felony
##  [91] infraction  misdimeanor none        none        none        misdimeanor
##  [97] none        misdimeanor felony      none
## Levels: none infraction misdimeanor felony
```

- Use `x_3` to create `x_3_bin`, a binary feature where 0 is no crime and 1 is any crime.

```
x_3_bin = as.numeric( x_3 != "none")
x_3_bin
```

```
##   [1] 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 1 1 1 1 1 1 1 0 1
##  [38] 0 1 1 1 1 0 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 0 0 1 1 1 1 1 1 1 1 1 0 1
##  [75] 0 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 0 0 0 1 0 1 1 0
```

- Use `x_3` to create `x_3_ord`, an ordered, nominal factor variable. Ensure the proper ordinal ordering.

```
x_3_ord = sort(x_3)
x_3_ord
```

```
##   [1] none        none        none        none        none        none
##   [7] none        none        none        none        none        none
##  [13] none        none        none        none        none        none
##  [19] none        none        none        none        none        infraction
##  [25] infraction  infraction  infraction  infraction  infraction  infraction
##  [31] infraction  infraction  infraction  infraction  infraction  infraction
```

```
##  [37] infraction  infraction  infraction  infraction  infraction  infraction
##  [43] infraction  infraction  infraction  infraction  infraction  infraction
##  [49] infraction  infraction  infraction  infraction  infraction  infraction
##  [55] infraction  misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor
##  [61] misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor
##  [67] misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor
##  [73] misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor misdimeanor
##  [79] felony      felony      felony      felony      felony      felony
##  [85] felony      felony      felony      felony      felony      felony
##  [91] felony      felony      felony      felony      felony      felony
##  [97] felony      felony      felony      felony
## Levels: none infraction misdimeanor felony
```