Jennifer Chu, Samnang Tep, Ryan Quigley
Section 2
Group C
MATH 267A

Problem Set 4

1. According to the help page for the function `scan()`, "a field is always delimited by an end-of-line marker unless it is quoted." Graphically, lines 1 and 2 end with the user-specified delimiter `,` but R reads `,\n`. Thus it determines that there is an empty string between these two delimiters. Additionally, the order in which the empty strings appear in the resulting vector hints at the explanation above. The last wonder pet on line 1 is Tuck and the first wonder pet on line 2 is Ming-Ming; in the resulting vector, these strings are separated by an empty string. Similarly for Ollie and The Visitor.

2. When `header = TRUE`, R expects the first line to contain the names of the variables that are associated with each column of data. The `read.table()` argument `check.names = TRUE` by default. When set to `TRUE`, this tells R to check the names of the variables in the data to ensure that they are syntactically valid variable names, i.e, name consists of letters, numbers and the dot or underline characters and starts with a letter or the dot not followed by a number (See `?read.table`). If the names are not syntactically valid, the `make.names` function is called, which will prepend the character "X" if necessary (See `?make.names`). This functionality can be overwritten by setting `check.names = FALSE`; however, the user must now be careful to user backticks when selecting individual variables by name using the `$` operator.

```
caffeine.bad <- read.table("caffeine.txt", header = TRUE, check.names = FALSE)
head(caffeine.bad, n = 2)
```

```
##     0 100 200
## 1 242 248 246
## 2 245 246 248
```

```
caffeine.bad$100
```

```
## Error: <text>:1:14: unexpected numeric constant
## 1: caffeine.bad$100
##                  ^
```

```
caffeine.bad$`100`
```

```
## [1] 248 246
```

3. **(a)** For the object `X`, the class is a matrix. Because matrices are stored as a vectors in column-major order, we can use the `length` function to determine the number of elements in `X`. There are 1,317,200 elements in `X`. For `y`, the class is a numeric vector, and there are 1,850 elements in the vector.

   (c) Using only `t`, `solve`, and `%*%`, the time that it took to compute $\hat{\beta}$ was:

```
##    user  system elapsed
##   2.190   0.036   2.250
```

   (d) Note: we passed $(X^TX)^{-1}$ and X^Ty to `crossprod` as the first and second arguments, respectively. According the `crossprod` help page, this is equivalent to calculating $((X^TX)^{-1})^TX^Ty$, which some might expect to give a different result than desired. However, X^TX is a symmetric matrix, and so $((X^TX)^{-1})^T = ((X^TX)^T)^{-1} = (X^TX)^{-1}$. Thus, using `solve` for matrix inversion and `crossprod` for matrix multiplication, the time that it took to compute $\hat{\beta}$ was:

```
##    user  system elapsed
##   1.590   0.017   1.617
```

(e) The provided formula for $\hat{\beta}$ can be rearranged in the following way,

$$\begin{aligned}
\hat{\beta} &= (X^T X)^{-1} X^T y \\
(X^T X)\hat{\beta} &= (X^T X)(X^T X)^{-1} X^T y \\
(X^T X)\hat{\beta} &= I X^T y \\
(X^T X)\hat{\beta} &= X^T y
\end{aligned}$$

Using `solve` for solving a system of linear equations and `crossprod` for matrix multiplication, the time that it took to compute $\hat{\beta}$ was:

```
##    user  system elapsed
##   0.836   0.008   0.851
```

(g) From part (c) to part (e), the time required to compute $\hat{\beta}$ gets progressively shorter, i.e, (e) is faster than (d) and (d) is faster than (c). According to the help page, `crossprod` is usually slightly faster than using `t` and `%*%`. In part (d), the equation to calculate $\hat{\beta}$ is the same, but in R we are making less function calls: in (c) we make 5 calls (1 `t`, 2 `%*%`, and 1 `solve`) whereas in (d) we make 4 calls (3 `crossprod` and 1 `solve`). Additionally, `crossprod(X)` is much faster than `t(X) %*% X` because $X^T X$ is known to be symmetric. As a result, only one triangle of the matrix needs to be calculated, and a dramatic speed improvement is expected (Bates & Eddelbuettel, 2013). In part (e), the equation rearrangment allows us to make one less `crossprod` call, so we again expect an improvement in performance over part (d).

(h) Converting `X` to a data frame and repeating part (c) results in the following error:

```
## Error in xt %*% X.df: requires numeric/complex matrix/vector arguments
```

```
## Timing stopped at: 0.068 0.009 0.079
```

Once `X` has been converted from a matrix to a data frame, $\hat{\beta}$ cannot be calculated using the code from part (c) because the matrix multiplication operator `%*%` does not work with data frames. Data frames have the ability to store values of different types, which is not appropriate for matrix multiplication. Instead of checking all of the columns of the data frame to makes sure they are all numeric (or complex), R simply checks the class of the objects being multiplied and throws an error if they are not appropriate.

In order to get it to work, we must convert `X` back to a matrix using the `as.matrix` function. Even if we include the matrix conversion in the timing of the code, the performance results are very similar to those from part (c). It would seem R can perform data frame to matrix conversion rather quickly.

```r
system.time({
    xt.5 <- t(as.matrix(X.df))
    xt.x.5 <- xt %*% as.matrix(X.df)
    xt.x.inv.5 <- solve(xt.x.5)
    xt.y.5 <- xt.5 %*% y
    beta5 <- xt.x.inv.5 %*% xt.y.5
})
```

```
##    user  system elapsed
##   2.130   0.034   2.180
```

References

Bates, D., & Eddelbuettel, D. (2013). Fast and Elegant Numerical Linear Algebra Using the RcppEigen Package. Journal of Statistical Software J. Stat. Soft., 52(5). doi:10.18637/jss.v052.i05

R Core Team (2016). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.