HW 1 Math 463 Spring 2022

Ian McConachie

Covid Data

Using the data

https://pages.uoregon.edu/dlevin/DATA/Covid.csv

discussed in class, compare the rate of initial exponential growth for the three countries, Italy, UK, and USA. Choose a single time frame from initial growth for all countries (use the same interval length for all countries, although onset time may differ by country) where the growth is clearly exponential. Can the difference between countries in rates be attributed to random variation, or represents a true difference?#

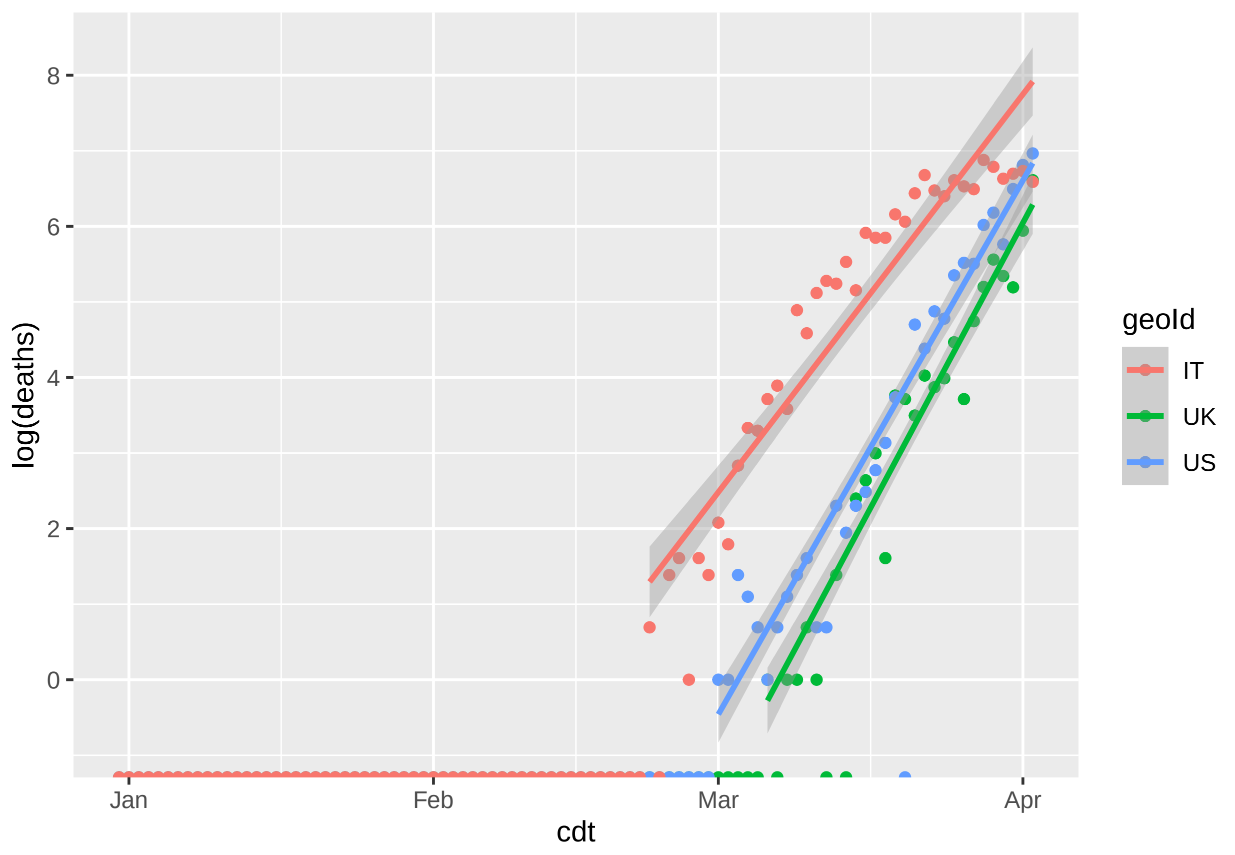
SOLUTION

First, let's graph the data as we did before, on a log-based scale with all days included in the regression fit and days with zero deaths set to 0 as to not cause problems with taking the natural log of 0.

```r
library(ggplot2)
# read.csv creates a data frame in R
covid <- read.csv("https://pages.uoregon.edu/dlevin/DATA/Covid.csv")
# dates can be tricky, cdt is a standard format for dates that R can understand
covid$cdt <- as.Date(covid$dateRep,format = "%d/%m/%Y")
# This ggplot function plots the data visually
ggplot(covid, aes(cdt,log(deaths),color=geoId)) + geom_point() + geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 186 rows containing non-finite values (stat_smooth).
```



Now we can use the following code to get the days that each country started to experience non-zero values for deaths from COVID—a.k.a. the onset time for each country's own COVID epidemic. These values will be stored in IT_start, UK_start, and US_start for Italy, the UK, and the US respectively.

```r
num_rows = nrow(covid)
max_day = covid[2, "day"]
# initializing variables for the for loop
IT_start = max_day + 1; UK_start = max_day + 1; US_start = max_day + 1

for (i in 2:num_rows) {
  # some variables we'll use a lot in this loop
  row = covid[i,]
  deaths = row$deaths

  # if statement to catch if it is the first non-zero day for Italy deaths
  if ((row$geoId == "IT") & (deaths > 0) & (IT_start > row$day)) {
    IT_start = row$day
  }
  # if statement to catch if it is the first non-zero day for Italy deaths
  if ((row$geoId == "UK") & (deaths > 0) & (UK_start > row$day)) {
    UK_start = row$day
  }
  # if statement to catch if it is the first non-zero day for Italy deaths
  if ((row$geoId == "US") & (deaths > 0) & (US_start > row$day)) {
    US_start = row$day
  }
}
```

Now we want to take the smallest time frame that an epidemic was "active" in a country, where an "active" epidemic is defined as the period of time after the first death from COVID-19. Because we have data for all countries that extends to April 2nd of 2020, we can find the smallest time frame of available data by finding the latest start of an epidemic (the max of the _start variables) and then taking the interval from that start date to April 2nd as our standard interval.

```
latest_start = max(IT_start, UK_start, US_start)
interval_len = max_day - latest_start
```

Now that we have the length of the time interval we will be looking at for each country's data, we can filter out the data so that we only look at the data that goes 26 days from the respective starts of the epidemics in each country. The filter function is a little complicated, but when we standardize the length of each epidemic we get the picture below:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

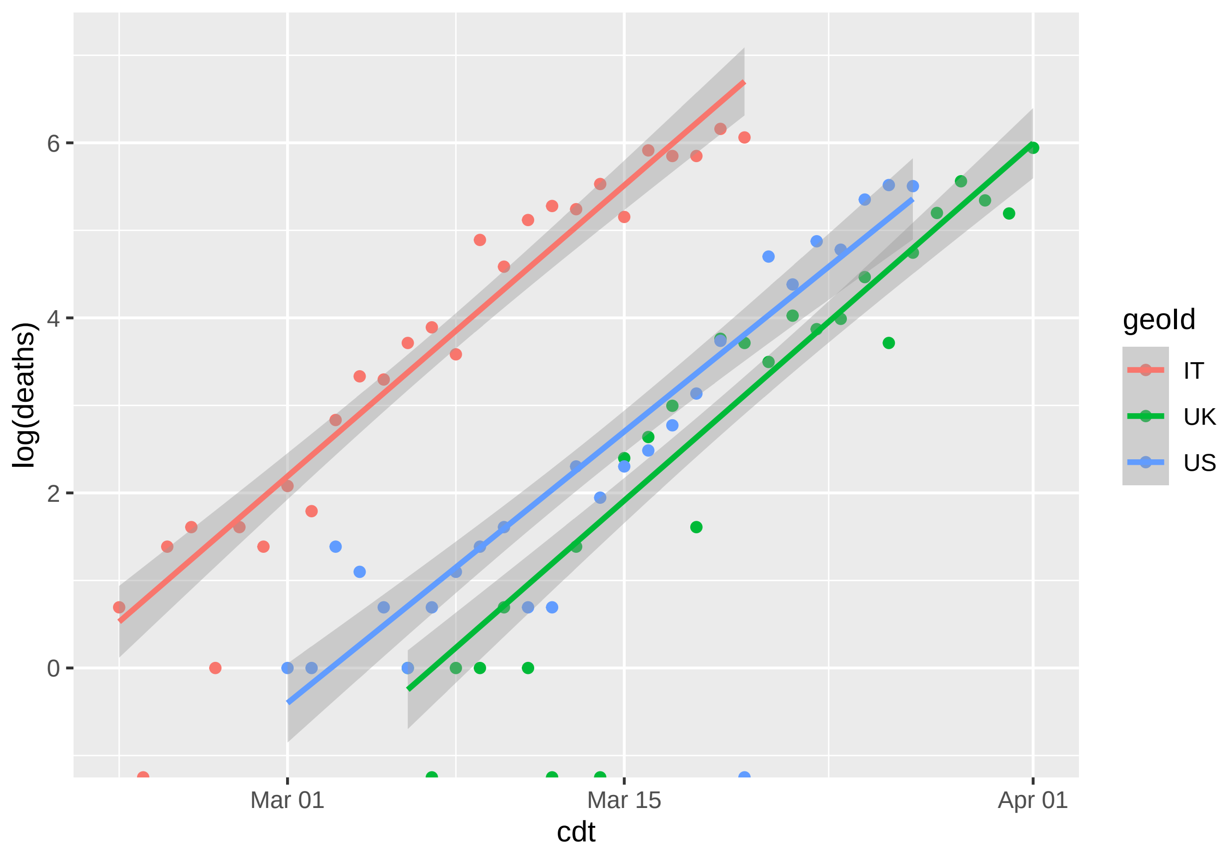
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
# Here we filter the data so we only have points within the time frames we're interested in
covid_subset = filter(covid, ((geoId == "IT") & (day <= (IT_start+interval_len) & (IT_start <= day))) |
                      ((geoId == "UK") & (day <= (UK_start+interval_len)) & (UK_start <= day)) |
                      ((geoId == "US") & (day <= (US_start+interval_len)) & (US_start <= day)))
# Then we can plot this subset of the data with ggplot
ggplot(covid_subset, aes(cdt,log(deaths),color=geoId)) + geom_point() + geom_smooth(method="lm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning: Removed 5 rows containing non-finite values (stat_smooth).
```



Then we can compare the least squares lines that we have generated for our modified/standardized COVID death data. We can do this by looking at summaries of each least-squares line with the following code:

```r
# Separating the data based n individual countries
country_deaths <- group_split(covid_subset,geoId)
IT_deaths <- country_deaths[[1]]
UK_deaths <- country_deaths[[2]]
US_deaths <- country_deaths[[3]]

# Fitting least-squares lines to each country's individual data
fit_it <- lm(log(deaths)~cdt, data=subset(IT_deaths, deaths>0))
fit_uk <- lm(log(deaths)~cdt, data=subset(UK_deaths, deaths>0))
fit_us <- lm(log(deaths)~cdt, data=subset(US_deaths, deaths>0))

# Generate some tables that summarize each of the countries respective least
# square lines and statistics associated with those lines
IT_summ = summary(fit_it)
UK_summ = summary(fit_uk)
US_summ = summary(fit_us)
```

In the summaries we have generated above, we have information about the estimated slope of each of the three country lines, as well as information about the standard error of these estimates. Using these statistics, we can compare two lines (let's say lineA and lineB) by testing to see if a 95% confidence interval for (Aslope - Bslope) contains 0 which would indicate the lines' slopes are sufficiently similar.

We can generate these confidence intervals by noting that the test statistic (Aslope_hat - Bslope_hat) has an approximately normal distribution with mean (Aslope - Bslope) and variance:

$$\sqrt(SE(Aslope)^2 + SE(Bslope)^2)$$

Below is code that takes in two linear model summaries and outputs a boolean value that indicates whether the slopes of the linear models are "sufficiently similar;" or in other words, have a difference that could be attributed to random variation.

```r
compareSlopes <- function(summA, summB) {
  ret = FALSE

  # First get estimated difference by subtracting slopeB-hat from slopeA-hat
  estDiff = summA$coefficients["cdt", 1] - summB$coefficients["cdt", 1]
  # Then get the variance of the difference estimate by adding together their
  # respective variances and taking the square root of the sum
  sdDiff = sqrt((summA$coefficients["cdt", 2])^2 + (summB$coefficients["cdt", 2])^2)

  # Calculate the lower and upper bound of the confience interval for comparing
  # the LSR slopes summarized in summA and summB
  ubound = estDiff + (2*sdDiff)
  lbound = estDiff - (2*sdDiff)

  # If 0 is in the confidence interval return TRUE
  if ((lbound < 0) & (ubound > 0)) {
    ret = TRUE
  }
  return(ret)
}

# We can just use a 2 standard deviation confidence interval
# want to test every pair to see if the three are identical
#       there is a different hypothesis test that asks if b_IT = b_US = b_UK, but
```

We can then input each of the 3 possible pairs of countries from Italy, UK, and US into this function and see if they are sufficiently similar by the previously defined metric. Note that with the function defined above if compareSlopes(sA, sB) returns TRUE, then so will compareSlopes(sB, sA).

If all three pairs of country slopes return TRUE, then it would be reasonable to assume that the difference in rates between countries could be attributed to random variation rather than true difference.

```r
if (compareSlopes(IT_summ, UK_summ) & compareSlopes(IT_summ, US_summ) &
    compareSlopes(UK_summ, US_summ)) {
  print("All three rates are reasonably similar")
} else {
  print("All three rates are not reasonably similar")
}
```

```
## [1] "All three rates are reasonably similar"
```

We therefore see that, from a classical statistics perspective, all 3 slopes are reasonably similar and their differences could be attributed to random variation. This seems to suggest, with only taking this data into account, that these three countries saw similar spread of COVID in the first 26 days of their respective epidemics.

Simulation of regression model

Problems 6.2, 6.3, 6.4; In 6.4, "median" and "mad sd" are mentioned. You should replace this with "estimate" and "standard error". Do not include the raw output of the simulations in your write-up, but summarize in a coherent way the results.

6.2: Programming fake-data simulation

Problem 6.2 asks us to write an R function that simulates n data points from the model

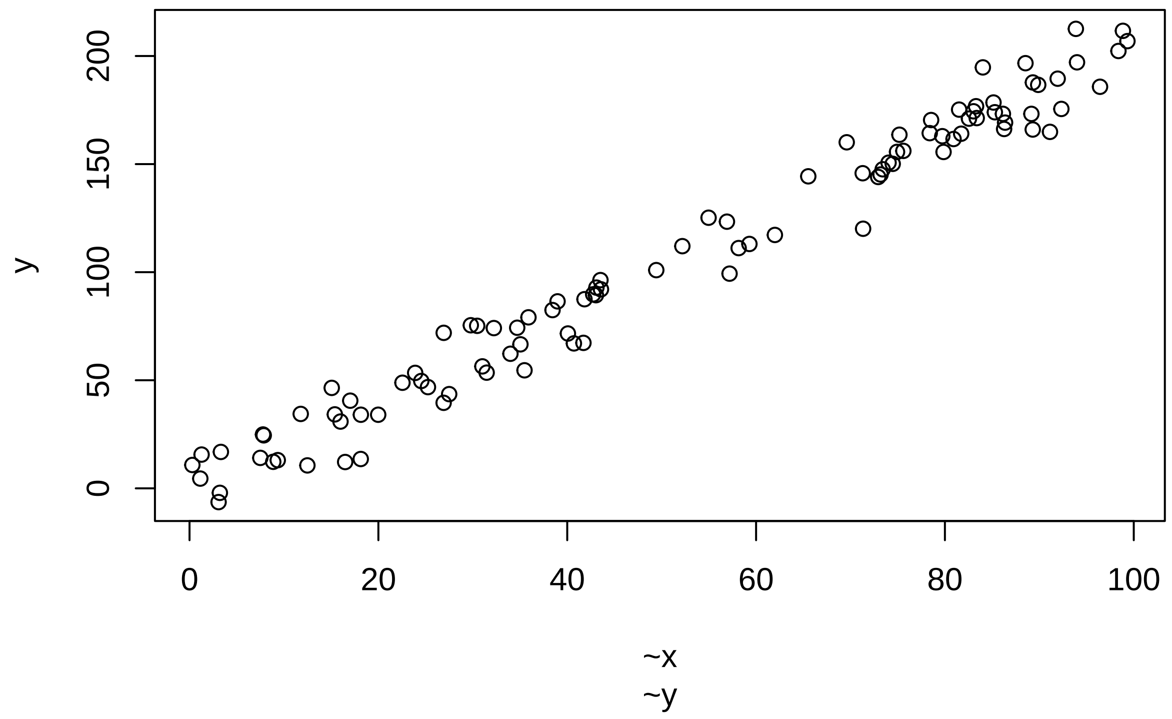
$$y = a + bx + error$$

with points x uniformly sampled from the range [0,100] and with errors drawn independently from a normal distribution with mean 0 and standard deviation sig (short for sigma). To do this part of the problem, I made a, b, n, and sig parameters passed into the function and used some built-in "random" variable functions provided by R.

Our function is then supposed to fit a linear regression line to the simulated data. In the function below, this is done with the lm function which uses a classical statistics approach to regression (i.e. least-squares).

Finally, the function prints out a scatter plot of the data with the fitted regression line graphed along with it. The return value of this function is a summary of the regression model applied to the points.

```r
fake_data_sim <- function(a,b,n,sig) {
  # Simulate n data points uniformly distributed over [0,100]
  x <- runif(n,0,100)
  # Simulate y values corresponding with x according to a + bx linear relation
  # and with added error of ep ~ N(0,sig)
  y <- a + b*x + rnorm(n,0,sig)
  # Fit linear model to the data and find slope value
  b <- coef(lm(y~x))[2]
  df = data.frame(x, y)
  plot(df, aes(x,y))+ geom_point() + geom_smooth(method="lm")
  return(summary(lm(y~x)))
}
fake_data_sim(1,2,100,10)
```



6.3: Variation, uncertainty, and sample size:

As we increase the number of points n, we can notice that the estimate given by our regression model returned in the function gets closer and closer to the b value we input into the function.

This makes sense because of the weak law of large numbers which states that as we observe more and more instances of the error random variable (the only source of randomness in this model), the sample mean of those instances approaches the expected value of the error term. The expected value of the error term here is 0, so the more y values we observe, the more error terms we observe, and the less the error term impacts the data as error trends towards 0. In the situation where there is no error, the least squares regression line will exactly fit the line where the data comes from, which explains why b-hat approaches b.

As we increase the number of observed points n, we can also see that the standard error of the estimate b-hat decreases.

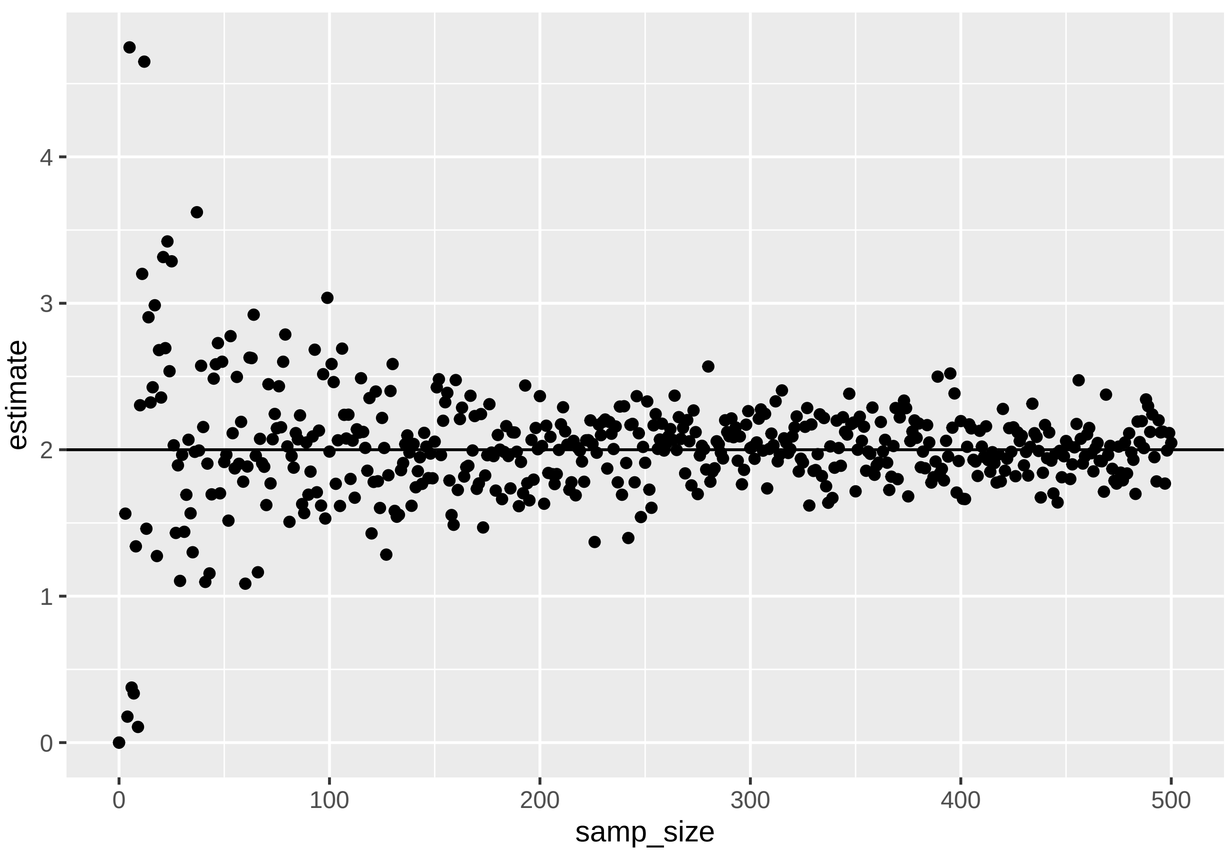
This is intuitively explained by the reasoning given above about error trending towards 0. However, we can also note that the equation for standard error of a regression slope is scaled by the inverse of the square root of the number of samples n. This implies that as n increases, standard error should decrease.

6.4: Simulation study

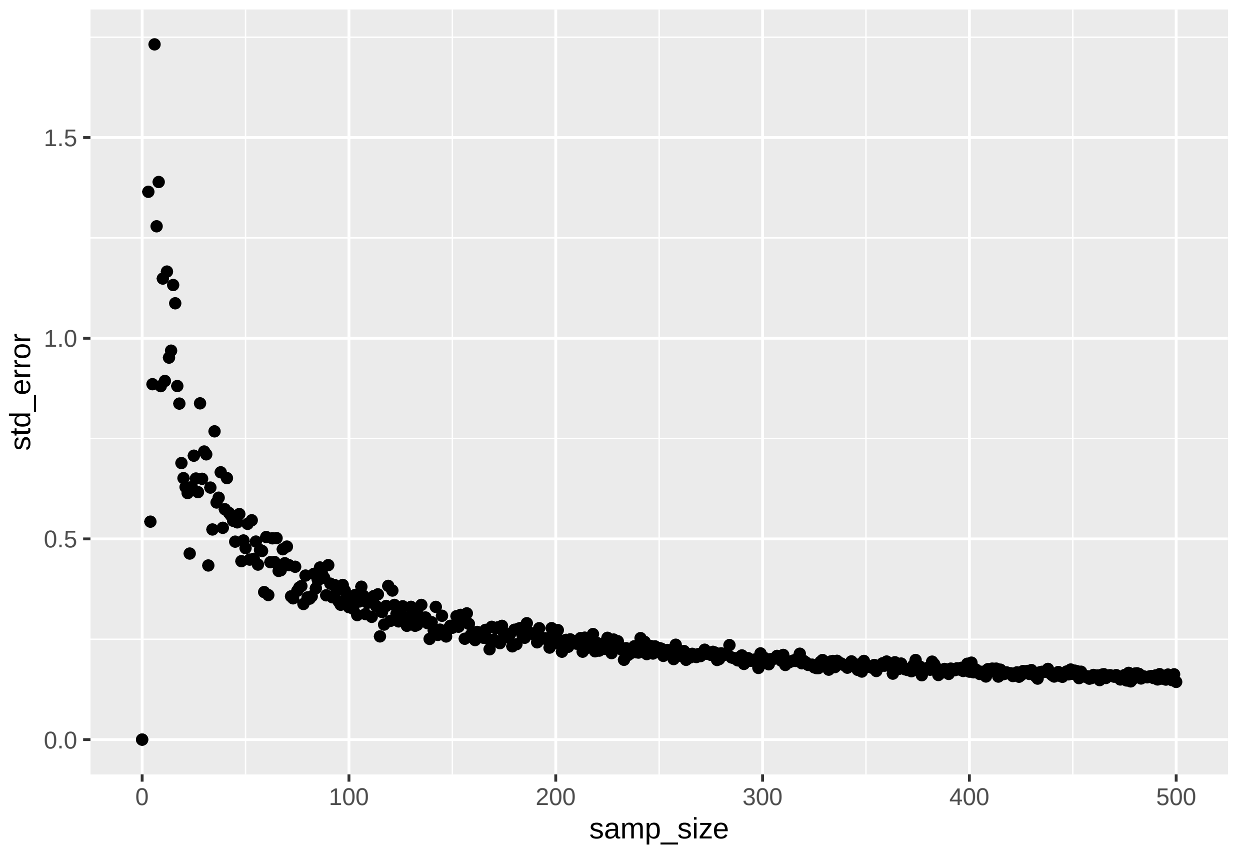
Problem 6.4 asks for us to perform the analysis we did above in a more systematic manner by graphing the change in estimate and standard error produced by fake_data_sim with a fixed a, b, and sig, but with n changing.

The R code below does just that varying n from 3 to 1000 and graphing the slope estimate and standard error of the slope estimate against n using ggplot2. Note that the fake_data_sim function is redefined without the plotting functionality so that each individual fake data simulation doesn't produce a new graph in this iterative setting. The graph starts at 3 and not 1 because 1 and 2 produce regression models with NULL/NaN standard errors that cause problems when graphing.

```r
fake_data_sim <- function(a,b,n,sig) {
  # Simulate n data points uniformly distributed over [0,100]
  x <- runif(n,0,100)
  # Simulate y values corresponding with x according to a + bx linear relation
  # and with added error of ep ~ N(0,sig)
  y <- a + b*x + rnorm(n,0,sig)
  # Fit linear model to the data and find slope value
  b <- coef(lm(y~x))[2]
  df = data.frame(x, y)
  #plot(df, aes(x,y))+ geom_point() + geom_smooth(method="lm")
  return(summary(lm(y~x)))
}


max_samp_size = 500;

estimate = numeric(max_samp_size)
std_error = numeric(max_samp_size)
samp_size = numeric(max_samp_size)

for (i in 3:max_samp_size) {
  summ = fake_data_sim(1,2,i,100)
  std_error[i] = summ$coefficients[2,2]
  estimate[i] = summ$coefficients[2,1]
  samp_size[i] = i
}
df2 = data.frame(samp_size, estimate, std_error)
ggplot(df2, aes(samp_size,estimate))+ geom_point() + geom_hline(yintercept=2)
```



```
ggplot(df2, aes(samp_size,std_error))+ geom_point()
```



We can see from the graphing of the estimated slope against the sample size n that as n increases, the points converge to the horizontal line on the plot which represents the true value for b which we set to 2 in simulating our data. This graphical representation is what we would expect given the conclusions we can draw from the weak law of large numbers as discussed earlier.

For the graph of standard error against sample size, we can see visually that the data roughly follows an inverse (exponential) curve. This is also exactly what we would expect given our earlier observations about the equation of standard error being scaled by the inverse of n raised to (1/2) [i.e. the square root of n].

Using categorical variables in a regression

Problem 7.6 in text.

Problem 7.6 asks us to use the election forecasting data provided in the textbook (which I have read into Rstudio from the authors' GitHub) and create a binary indicator for economic growth where the indicator equals 1 if the growth was 2% or more and 0 otherwise. The code below imports the data and creates a new column with this binary indicator.

```r
# Reading in the data
myfile <- "https://raw.githubusercontent.com/avehtari/ROS-Examples/master/ElectionsEconomy/data/hibbs.da
hibbs <- read.table(myfile, header = T)

nrows = nrow(hibbs)
bin_indicator = numeric(nrows)
# Generating the binary indicators for each row in the hibbs dataframe
for (i in 1:nrows) {
  if (hibbs$growth[i] > 2) {
    bin_indicator[i] = 1
  } else {
    bin_indicator[i] = 0
  }
}
# Adding this indicator column to the dataframe
hibbs$b_ind <- bin_indicator
```

Now, we are asked to compute the difference in the vote share on average for the two groups defined by the binary indicator. The code below uses the dplyr library to separate the election data into the two groups.

It then uses the mean function to find the average vote percentage going towards the incumbent in each group. The average difference in vote percentage can then be calculated by finding the difference between the averages for the two groups.

```
# Here splitting the data and finding the average vote percentage for each group
# avg_1 being the vote percentage when the binary indicator is 1 and avg_0
# being when the indicator is 0
library(dplyr)
hibbs_1 = filter(hibbs, bin_indicator == 1)
hibbs_0 = filter(hibbs, bin_indicator == 0)
avg_1 = mean(hibbs_1$vote)
avg_0 = mean(hibbs_0$vote)

# The average difference is then the difference between the two group avgs
avg_diff = avg_1 - avg_0
print("Average difference calculted using averages for each group")
print(avg_diff)
```

The problem then asks us to find the standard error of this difference in averages. We can calculate the standard error of the difference between two sample means by finding the square root of the sum of their respective variances, which we can estimate with the sample variance.

```
# The var function here calculates the sample variance for both groups
se <- function(x) sqrt(var(x)/length(x))
samp_var_0 = (var(hibbs_0$vote))/nrow(hibbs_0)
samp_var_1 = (var(hibbs_1$vote))/nrow(hibbs_1)
std_err = sqrt(samp_var_0 + samp_var_1)
print("Standard error using sample variances for each group")
```

```
## [1] "Standard error using sample variances for each group"
```

```
print(std_err)
```

```
## [1] 2.502052
```

We are then asked to use linear regression to estimate a slope and its standard error for a linear relationship between the binary indicator and the vote percentage. This is done below using the lm function (classical statistics) and then the estimate of the slope and its standard error is printed.

```r
fit_b_ind <- lm(vote~b_ind, data=hibbs)
b_summ = summary(fit_b_ind)
print("Estimated average difference:")
print(coef(b_summ)[2,1])
print("Standard error for estimated average difference:")
print(coef(b_summ)[2,2])
```

Note that the estimate and its standard error are the same in both the averaging the difference calculation and the regression slope calculation.

Another "fake data" example

Problem 7.7

Part (a) of this problem asks us to generate "fake" data of 100 points from the model

$$y = 2 + 3x + error$$

where predictors x are drawn from a uniform distribution from 0 to 20 and error is modeled with a standard normal distribution that has a standard deviation of 5.

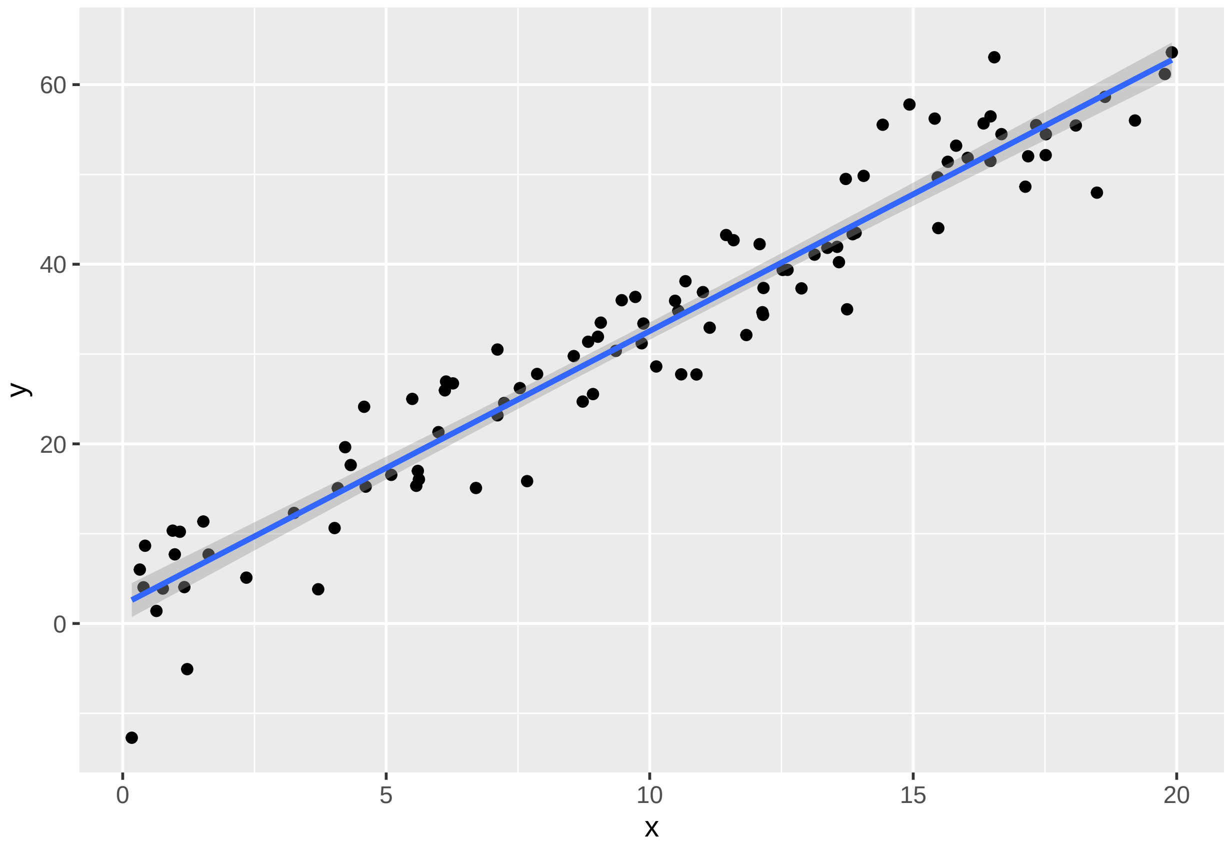
The code below does just that and then uses stan_glm to plot a regression line on the data. stan_glm is a linear model function that uses a Bayesian approach in contrast to lm's more traditional statistical approach.

```r
library(rstanarm)
```

```
## Loading required package: Rcpp

## This is rstanarm version 2.21.1

## - See https://mc-stan.org/rstanarm/articles/priors for changes to default priors!

## - Default priors may change, so it's safest to specify priors, even if equivalent to the defaults.

## - For execution on a local, multicore CPU with excess RAM we recommend calling

##   options(mc.cores = parallel::detectCores())

##
## Attaching package: 'rstanarm'

## The following object is masked _by_ '.GlobalEnv':
##
##     se
```

```r
# Generate 100 x values between 0 and 20
x <- runif(100,0,20)
# Generate the y data with "random" error added
y <- 2 + 3*x + rnorm(100,0,5)

# Make a data frame and fit a plot to it with stan_glm
fake = data.frame(x, y)
fake_fit = stan_glm(y ~ x, data=fake)
ggplot(fake, aes(x,y)) + geom_point() + geom_smooth(method="stan_glm")
```

```
## `geom_smooth()` using formula 'y ~ x'
```



Now that we have generated this fake data and have a regression model fit to it, we can see how "reasonably close" the estimated coefficients are to the true values used in the model by creating 68% and 95% confidence intervals for both the intercept and slope and seeing if the true values used in generating the fake data are included in these intervals.

Let's begin with the slope. We can create a 95% interval by adding a buffer of 2 standard errors on each side of the estimated coefficient value and a 68% interval by doing the same thing but with 1 standard error. If the true value for the coefficient is in the 95% confidence interval, our estimate can be considered "reasonably close." However, if it is in the 68% confidence interval, this would imply our estimate was even better and we would then consider it beyond "reasonably close."

```r
# Extracting the slope estimate and standard error from the stan_glm fit
b_hat = coef(fake_fit)["x"]
b_hat_se = fake_fit$ses[2]


# Seeing if the true value is in these confidence intervals
in_68b = abs(3 - b_hat) < b_hat_se
in_95b = abs(3 - b_hat) < (2 * b_hat_se)

if (in_95b) {
  print("Slope estimate reasonably close (true value in 95% confidence interval)")
}
if (in_68b) {
  print("Slope estimate beyond reasonably close (true value in 68% confidence interval)")
}
```

Now we can do the exact same procedure with the estimated intercept with the following code:

```
# Extracting the intercept estimate and standard error from the stan_glm fit
a_hat = coef(fake_fit)["(Intercept)"]
a_hat_se = fake_fit$ses[1]

# Seeing if the true value is in these confidence intervals
in_68a = abs(2 - b_hat) < a_hat_se
in_95a = abs(2 - b_hat) < (2 * a_hat_se)

if (in_95a) {
  print("Intercept estimate reasonably close (true value in 95% confidence interval for estimate)")
}
if (in_68a) {
  print("Intercept estimate beyond reasonably close (true value in 95% confidence interval for estimate
}
```

The results printed above vary with each running of this code, but I have found that generally both the slope and the intercept estimate have 95% confidence intervals that include the true values used in generating this data, but not 68% CIs. This suggests that generally stan_glm has "reasonably close" estimates for regression set ups with parameters similar to these.