DA 605 - Final Exam

Dan Fanelli

May 14, 2016

---

1. Review of Essential Concepts - 15 points

---

1) What is the rank of the following matrix?

```
##      [,1] [,2] [,3] [,4]
## [1,]    1   -1    3   -5
## [2,]    2    1    5   -9
## [3,]    6   -1   -2    4
```

The matrix rank is: **3**

---

2) What is the reduced row-echelon form of the above matrix?

The rref of the above matrix is:

```
##      [,1] [,2] [,3]        [,4]
## [1,]    1    0    0  0.03636364
## [2,]    0    1    0 -0.25454545
## [3,]    0    0    1 -1.76363636
```

---

3) Define orthonormal basis vectors. Please write down at least one orthonormal basis for the 4-dimensional vector space R4.

Orthonormal basis vectors are unit vectors that are orthogonal to each other. (their dot product == 0)

```
q3_v1 <- c(1/3,2/3,2/3,0)
q3_v2 <- c(2/3,1/3,-2/3,0)
sum(q3_v1 * q3_v2)
```

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

---

4) Given the following matrix, what is its characteristic polynomial?

x^3 - 13x^2 + 46x - 26

```
##      [,1] [,2] [,3]
## [1,]    6    1    1
## [2,]    0    7   -1
## [3,]   -1    3    0
```

```
## [1]   1 -13  46 -26
```

---

5) What are its eigenvectors and eigenvalues? Please note that it is possible to get complex vectors as eigenvectors.

Eigenvalues:

```
## [1] 6.7856670 5.5202305 0.6941025
```

Eigenvector #1:

```
## [1] 0.8339613 0.5395686 0.1156473
```

Eigenvector #2:

```
## [1]  0.9451655 -0.1828644 -0.2705972
```

Eigenvector #3:

```
## [1]  0.2108156 -0.1531045 -0.9654614
```

---

6) Given a column stochastic matrix (all col values add to 1 and no negatives) of links between URLs, what can you say about the PageRank of this set of URLs? How is it related to its eigendecomposition?

PageRank is be done by estimating the Principal Eigenvector (the eigenvector corresponding to the largest eigenvalue) of the column stochastic matrix generated by the Eigen Decomposition.

---

7) Assuming that we are repeatedly sampling sets of numbers (each set is of size n) from an unknown probability density function. What can we say about the mean value of each set?

The mean value of the each set will be normally distributed and have a mean that is approximately the mean of the parent set.

---

8) What is the derivative of (e^x)*((sin(x))^2) ?

Product Rule:

= (e^x)'* ((sin(x))^2) + (e^x) * ((sin(x))^2)'

= ((e^x) dx * ((sin(x))^2) + (e^x) * 2 * sin(x) * cos(x)) dx

---

9) What is the derivative of log(x^3 + sin(x))?

= ((1/log(x^3 + sin(x)))*(d/dx x^3 + sin(x))) dx

= ((1/log(x^3 + sin(x)))*(3x^2 + cos(x))) dx

---

10) What is integral(e^x cos(x) + sin(x)dx) ? Don't forget the constant of integration.

= integral(e^x cos(x) + sin(x) dx)

= integral(sin(x)dx) + integral(e^x cos(x) dx)

= -cos(x) + integral(e^x cos(x))

(call the above the "***left***" and the "***right***", now just work on the right:)

---

integral (e^x cos(x)) dx

(right:) integration by parts I, let:

u = e^x, so du = e^x dx

dv = cos(x) dx, so v = sin(x)

so integral(e^x cos(x)) dx = e^x * sin(x) - integral(e^x sin(x))

(right:) integration by parts II, let:

u = e^x, so du = e^x dx

dv = sin(x) dx, so v = - cos(x)

so integral(e^x sin(x) dx) = - e^x cos(x) - integral(- e^x cos(x) dx)

so integral(e^x sin(x) dx) = - e^x cos(x) + integral(e^x cos(x) dx)

(right:) substitution:

So now let's fully write out the "right":

integral (e^x cos(x)) dx = e^x * sin(x) - [integral(e^x * sin(x) dx)]

integral (e^x cos(x)) dx = e^x * sin(x) - [- e^x cos(x) + integral(e^x cos(x) dx)]

integral (e^x cos(x)) dx = e^x * sin(x) + e^x cos(x) - integral(e^x cos(x) dx)

(right:) integral (e^x cos(x)) is on both sides of equation, so solve:

2 * integral (e^x cos(x)) = e^x * sin(x) + e^x cos(x)

integral (e^x cos(x)) = (1/2)(e^x * sin(x) + e^x cos(x)) + C

Final answer: combine the left and right:

integral(e^x cos(x) + sin(x)dx) = -cos(x) + (e^x * sin(x) + e^x cos(x))/2 + C

---

2. Mini-coding assignments - 15 points

2.1. Sampling from function.

Assume that you have a function that generates integers between 0 and 50 with the following probability distribution:

P(x == k) = (50 choose k) * (p^k) * (50^(q-k))

where p = 0.2 and q = 1 - p = 0.8 and x is in [0,50].

This is also known as a Binomial Distribution.

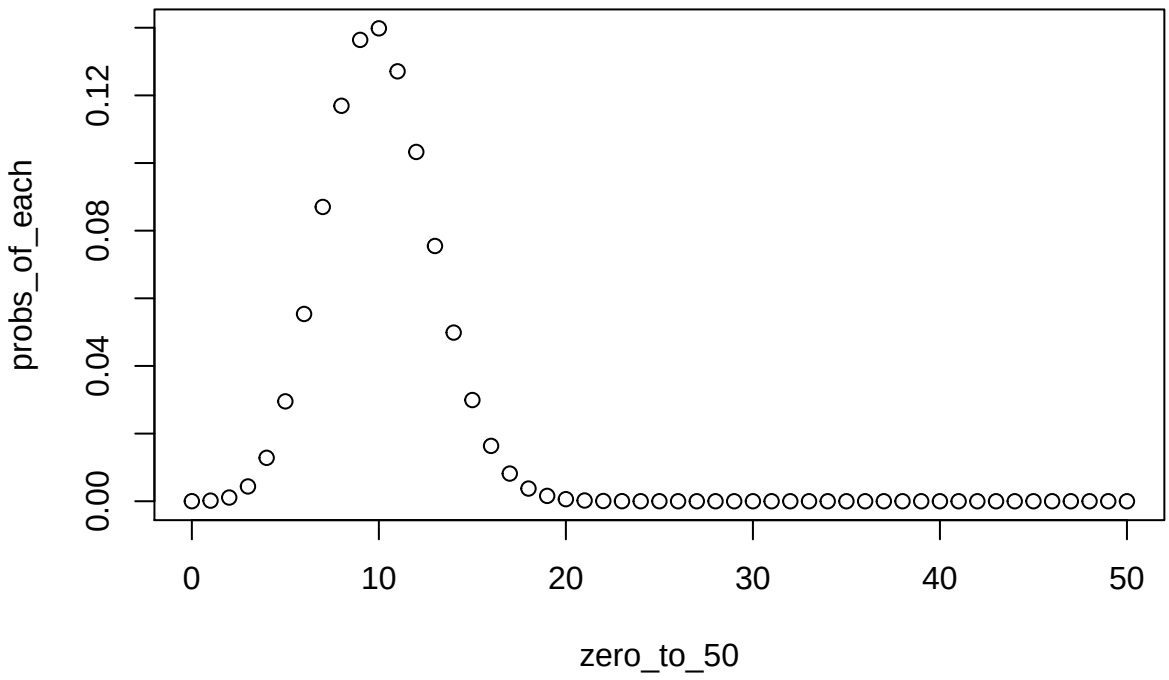
- Write a function to sample from this distribution.
- After that, generate 1000 samples from this distribution and plot the histogram of the sample.

(Please note that the Binomial distribution is a discrete distribution and takes values only at integer values of x between x in [0,50]. Sampling from a discrete distribution with finite values is very simple but it is not the same as sampling from a continuous distribution.

```
prob_of_chooseing <- function(number_k){
  p <- 0.2
  q <- 1-p
  fifty_choose_k <- factorial(50)/(factorial(number_k)*factorial(50-number_k))
  p_to_the_k <- p^number_k
  q_to_the_50_minus_k <- q^(50-number_k)
  return (fifty_choose_k * p_to_the_k * q_to_the_50_minus_k)
}

#-----------------------------------------------------------
# not a sample, just the probability distribution plotted:
#-----------------------------------------------------------
zero_to_50 <- c(0:50)
probs_of_each <- prob_of_chooseing(zero_to_50)
# confirm it all sums to 1
sum(probs_of_each)
```

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

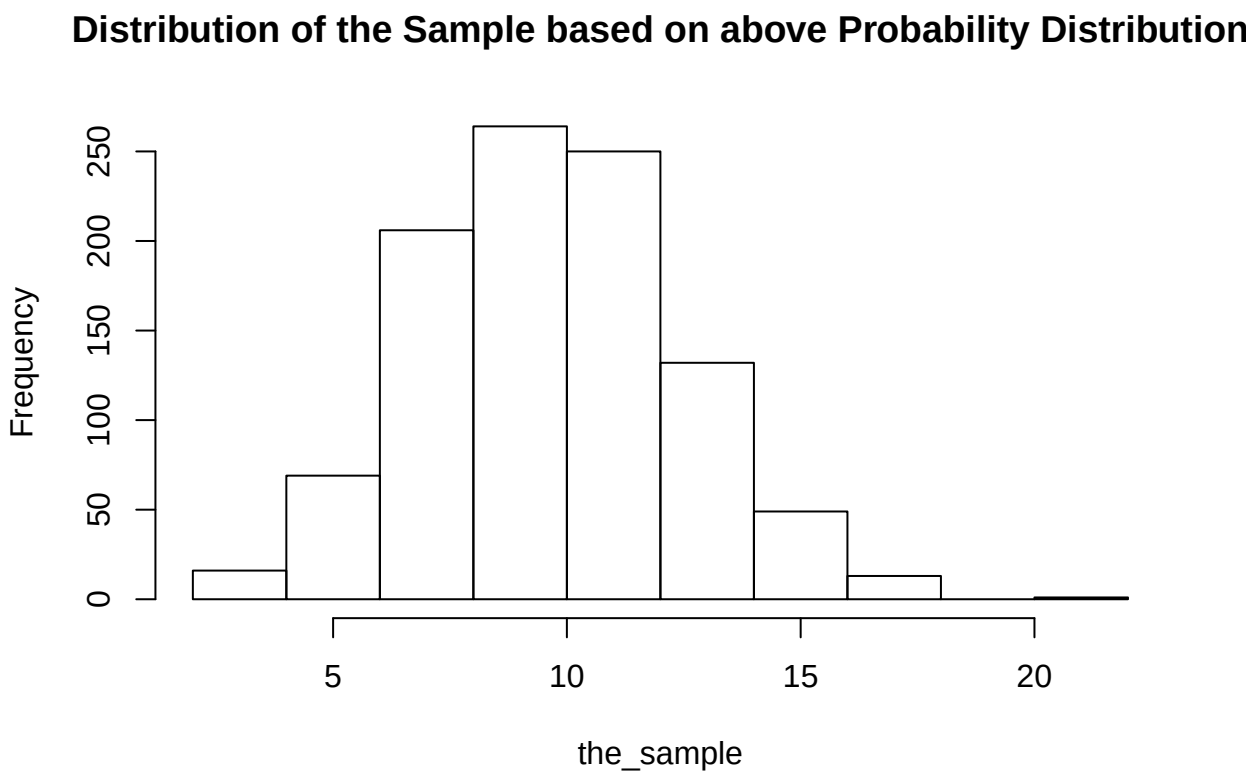
```
plot(zero_to_50, probs_of_each, main = "Plot of PROBABILITIES of Choosing Number.")
```




```
#-------------------------------------------------------
# now the sample:
#-------------------------------------------------------
the_sample <- sample(zero_to_50, size = 1000, replace = TRUE, prob=probs_of_each)
head(the_sample, n=40)
```

```
##  [1]  7 12 10 12 14 11 10 12  4 12 12 12 10 13 15  9 14  8 12  6 11 13  8
## [24] 13 16  8 10 15 13 10  5  7  9 15 14 11  9 13 11 10
```

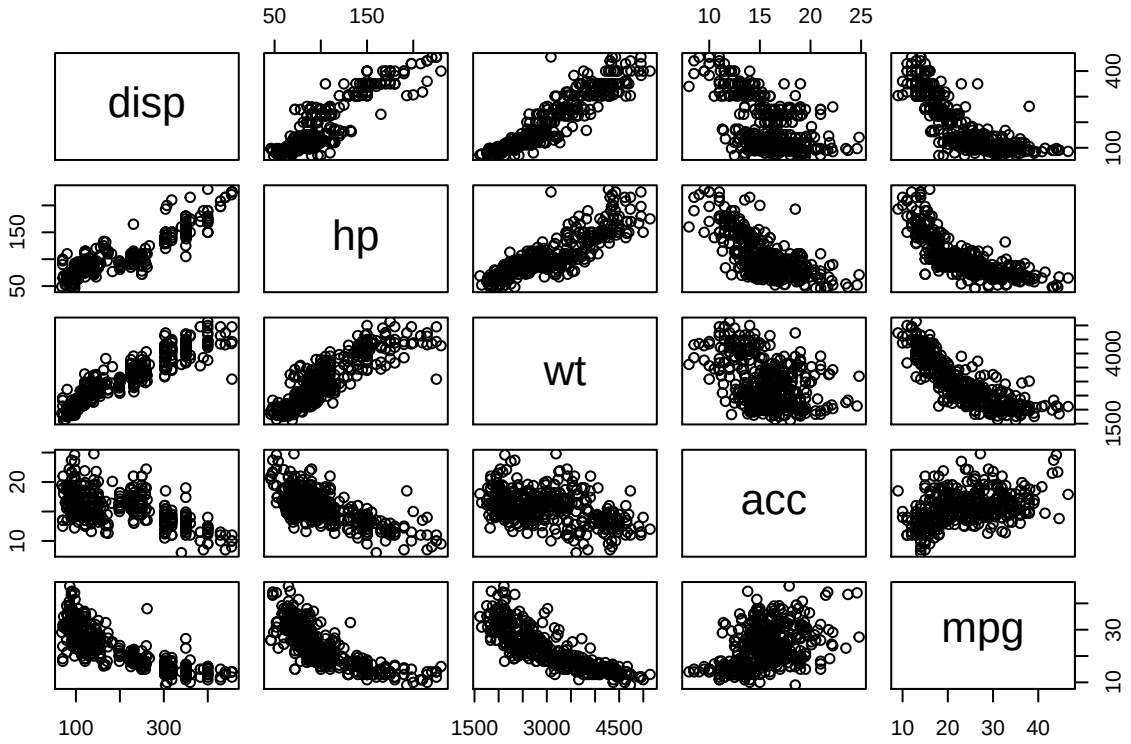
```
hist(the_sample, main = "Distribution of the Sample based on above Probability Distribution")
```



2.2. Principal Components Analysis.

First take a look at the data:

```
auto_df <- as.data.frame(read.table("auto-mpg.data"))
colnames(auto_df) <- c("disp","hp","wt","acc","mpg")
pairs(auto_df)
```



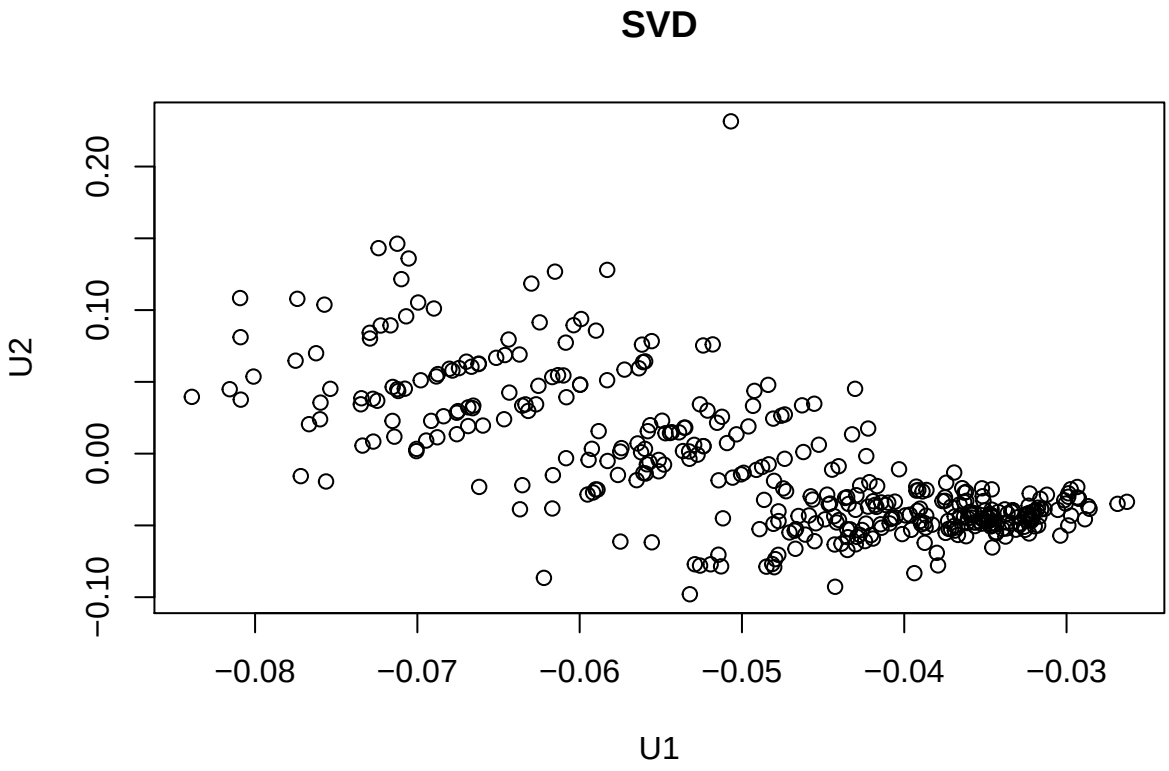

```
# Seems like highly correlated, good candidate for PCA.
```

- For the auto data set attached with the final exam, please perform a Principal Components Analysis by performing an SVD on the 4 independent variables (with mpg as the dependent variable) and select the top 2 directions.

```
# nv param tells it to only give back 2 directions
auto_df_svd <- svd(auto_df[,1:4], nv=2)
```

- Please scatter plot the data set after it has been projected to these two dimensions.
- Your code should print out the two orthogonal vectors and also perform the scatter plot of the data after it has been projected to these two dimensions.

```
# columns of U from the SVD correspond to the principal components
plot(auto_df_svd$u[, 1], auto_df_svd$u[, 2], main = "SVD", xlab = "U1", ylab = "U2")
```




```
# first eigen vector:
auto_df_svd$u[,1]
```

```
##   [1] -0.05723835 -0.06037128 -0.05615942 -0.05609517 -0.05634653
##   [6] -0.07098619 -0.07123753 -0.07053790 -0.07239290 -0.06297601
##  [11] -0.05830296 -0.05899484 -0.06152114 -0.05068035 -0.03864535
##  [16] -0.04621590 -0.04526145 -0.04222342 -0.03469923 -0.02989201
##  [21] -0.04350224 -0.03957623 -0.03868411 -0.03642671 -0.04321424
##  [26] -0.07536205 -0.07141868 -0.07153391 -0.07718429 -0.03469923
##  [31] -0.03692148 -0.03631024 -0.04302976 -0.05607846 -0.05431987
##  [36] -0.05387517 -0.05363489 -0.06873851 -0.07293514 -0.06784101
##  [41] -0.06686180 -0.08088079 -0.07750508 -0.08389686 -0.04838333
##  [46] -0.03924644 -0.05355771 -0.05123196 -0.03618543 -0.03460811
##  [51] -0.03376106 -0.03362826 -0.02886833 -0.02627719 -0.02988372
##  [56] -0.03184493 -0.03712113 -0.03463053 -0.03669113 -0.03925681
##  [61] -0.03628291 -0.06979252 -0.07165416 -0.06749424 -0.06743558
##  [66] -0.05997064 -0.07572693 -0.07348403 -0.07274098 -0.07226278
##  [71] -0.03791719 -0.06353813 -0.06687051 -0.07004905 -0.06655378
##  [76] -0.04776092 -0.04089759 -0.04849175 -0.03564419 -0.03902330
##  [81] -0.03726377 -0.04082718 -0.03524713 -0.03421292 -0.06697684
##  [86] -0.05997064 -0.06514348 -0.06596090 -0.06168902 -0.08089398
##  [91] -0.07292083 -0.07123288 -0.06914830 -0.07739725 -0.08092233
##  [96] -0.06246376 -0.05092201 -0.05349308 -0.04807301 -0.04931869
## [101] -0.04736724 -0.03175683 -0.08156385 -0.08009788 -0.07596855
## [106] -0.07344965 -0.04554331 -0.03711569 -0.03913297 -0.03875280
## [111] -0.03457278 -0.03764460 -0.04032074 -0.03689067 -0.06666774
```

```
## [116] -0.06995048 -0.03038009 -0.03516715 -0.04204935 -0.04670699
## [121] -0.05555944 -0.04333291 -0.04576262 -0.05990960 -0.05057795
## [126] -0.04735953 -0.05443350 -0.03174858 -0.03992796 -0.02989006
## [131] -0.04142089 -0.06164939 -0.05924806 -0.05890003 -0.06756792
## [136] -0.07667569 -0.07271563 -0.07562718 -0.06945701 -0.03614081
## [141] -0.03195912 -0.03745013 -0.02685556 -0.03260904 -0.03460286
## [146] -0.03432727 -0.03659400 -0.04055155 -0.03894674 -0.03255913
## [151] -0.05323505 -0.05643080 -0.05597442 -0.05153126 -0.07624024
## [156] -0.07247296 -0.07338055 -0.07599453 -0.06367732 -0.06353338
## [161] -0.06083725 -0.06168355 -0.04960181 -0.05258762 -0.05179999
## [166] -0.03535672 -0.04299838 -0.04757033 -0.04223350 -0.04402044
## [171] -0.03618983 -0.04145865 -0.04863528 -0.03155143 -0.05238067
## [176] -0.04386898 -0.04813539 -0.04794747 -0.04351411 -0.02924046
## [181] -0.04012533 -0.03617607 -0.04191695 -0.03672237 -0.03586842
## [186] -0.06877115 -0.06838606 -0.06465610 -0.06882952 -0.05273519
## [191] -0.05471184 -0.04911284 -0.05033756 -0.03312525 -0.03523619
## [196] -0.03155145 -0.02924046 -0.05951344 -0.05828047 -0.05944977
## [201] -0.05212083 -0.02974357 -0.03240549 -0.03509730 -0.04179186
## [206] -0.05128412 -0.06433220 -0.05321122 -0.04774924 -0.06220145
## [211] -0.07151998 -0.06622984 -0.06316786 -0.06133239 -0.03331086
## [216] -0.03511573 -0.02971645 -0.03748853 -0.03167577 -0.06334179
## [221] -0.06619061 -0.06757247 -0.07005944 -0.05742284 -0.05585827
## [226] -0.05917292 -0.05749723 -0.06898140 -0.06802785 -0.07068985
## [231] -0.07077379 -0.03161255 -0.04465126 -0.03688115 -0.04488273
## [236] -0.03340517 -0.03380574 -0.03233608 -0.03566639 -0.04586687
## [241] -0.04235988 -0.04425990 -0.03231773 -0.02933660 -0.03230643
## [246] -0.03370285 -0.02932547 -0.05492057 -0.06100228 -0.05830821
## [251] -0.05764215 -0.05143974 -0.04835287 -0.04431459 -0.05592978
## [256] -0.05236447 -0.05512849 -0.05005556 -0.05901658 -0.05565374
## [261] -0.05596379 -0.05621675 -0.05238943 -0.06659667 -0.03509441
## [266] -0.04171732 -0.03748568 -0.03632228 -0.04098766 -0.04474775
## [271] -0.04651444 -0.03918832 -0.04609686 -0.05117195 -0.04552500
## [276] -0.05555489 -0.03241038 -0.03477011 -0.05294510 -0.04875844
## [281] -0.04707137 -0.05325640 -0.05480039 -0.06268478 -0.06081591
## [286] -0.06460549 -0.06254004 -0.07118153 -0.06621320 -0.05882870
## [291] -0.06437920 -0.03135630 -0.03216037 -0.03120942 -0.04347792
## [296] -0.05749133 -0.06370538 -0.05192795 -0.05580145 -0.03583284
## [301] -0.03502220 -0.03288925 -0.03468162 -0.04351732 -0.04233978
## [306] -0.04404253 -0.04166854 -0.03492050 -0.03204762 -0.03452794
## [311] -0.03287394 -0.04364708 -0.04674712 -0.04891740 -0.05512960
## [316] -0.03563409 -0.04416309 -0.04139854 -0.03965571 -0.03689323
## [321] -0.03434967 -0.04563942 -0.03434865 -0.03393929 -0.03799332
## [326] -0.04801117 -0.05290396 -0.03014007 -0.03493068 -0.03005400
## [331] -0.04745183 -0.03937823 -0.04072700 -0.03729592 -0.04057715
## [336] -0.04294637 -0.04271317 -0.04444502 -0.03887431 -0.02858123
## [341] -0.03053834 -0.02866564 -0.03363342 -0.03215957 -0.03337833
## [346] -0.03232990 -0.03607219 -0.03330896 -0.03874150 -0.03567306
## [351] -0.03600020 -0.03827167 -0.04259529 -0.04290607 -0.05258140
## [356] -0.05144844 -0.04728058 -0.04774480 -0.05569890 -0.06085631
## [361] -0.04989516 -0.05648867 -0.04241878 -0.04298624 -0.03901340
## [366] -0.04193033 -0.04114472 -0.04457148 -0.04666821 -0.03226789
## [371] -0.03297869 -0.03208679 -0.03461278 -0.03460916 -0.03521145
## [376] -0.03591913 -0.03656618 -0.03200494 -0.03200500 -0.03249150
## [381] -0.04802168 -0.04923315 -0.04214564 -0.04629609 -0.04343169
```

```
## [386] -0.03863109 -0.04805776 -0.04544854 -0.03467934 -0.03741482
## [391] -0.04274680 -0.04428794
```

```
# second eigen vector:
auto_df_svd$u[,2]
```

```
##   [1]  0.0585528544  0.0894777492  0.0759477421  0.0637492927  0.0594730161
##   [6]  0.1215621917  0.1462481114  0.1359498085  0.1431553084  0.1184899301
##  [11]  0.1279961517  0.0857008121  0.1267792961  0.2314905452 -0.0432276851
##  [16]  0.0011178620  0.0062053523  0.0174920175 -0.0424069368 -0.0300269438
##  [21] -0.0671121151 -0.0530834076 -0.0515375269 -0.0240545741  0.0134528297
##  [26]  0.0451407041  0.0118286743  0.0228039728 -0.0156694287 -0.0424069368
##  [31] -0.0131704237 -0.0336509409  0.0451451407 -0.0135254684  0.0149422703
##  [36]  0.0148320633  0.0018077911  0.0553982306  0.0841882911  0.0578998041
##  [41]  0.0322530227  0.0376252676  0.0647579044  0.0395534044  0.0479233452
##  [46] -0.0257765885  0.0180813662  0.0256622575 -0.0266220954 -0.0248721924
##  [51] -0.0577735533 -0.0479298901 -0.0457055900 -0.0335628731 -0.0276389403
##  [56] -0.0394403831 -0.0370648629 -0.0431563050 -0.0566386027 -0.0229779069
##  [61] -0.0272102042  0.0511101123  0.0893615476  0.0296416952  0.0595875369
##  [66]  0.0480206681  0.1037979198  0.0343771554  0.0381999176  0.0892464441
##  [71] -0.0778646033  0.0334301239  0.0192122244  0.0033044569  0.0334295596
##  [76] -0.0704797833 -0.0485834716 -0.0787194468 -0.0504431419 -0.0383208552
##  [81] -0.0523921701 -0.0455543871 -0.0450693986 -0.0405816313  0.0640883500
##  [86]  0.0480206681  0.0667485441  0.0195716748  0.0533362443  0.0811385103
##  [91]  0.0802093650  0.0449384050  0.0228357350  0.1078517716  0.1083776228
##  [96]  0.0914477059  0.0073764128  0.0181145115  0.0243974050  0.0332930155
## [101] -0.0036045390 -0.0376520711  0.0448929690  0.0536912235  0.0355519008
## [106]  0.0385924512  0.0347658019 -0.0525827318 -0.0253532477 -0.0483622742
## [111] -0.0653778984 -0.0330632172 -0.0108981031 -0.0501947434  0.0605472653
## [116]  0.1052817657 -0.0571071079 -0.0296608305 -0.0567552073 -0.0662685779
## [121]  0.0783886709 -0.0527464675 -0.0297157061  0.0937936479 -0.0167052636
## [126]  0.0273001312  0.0143651035 -0.0500275429 -0.0429994033 -0.0500155921
## [131] -0.0339900270 -0.0149920140  0.0033766329 -0.0250813442  0.0135518536
## [136]  0.0205158515  0.0083990169 -0.0193895845  0.0091199515 -0.0483403195
## [141] -0.0506159711 -0.0551982033 -0.0349821380 -0.0509498786 -0.0502858506
## [146] -0.0492412557 -0.0353509134 -0.0444713357 -0.0492346326 -0.0530953565
## [151] -0.0035951788  0.0071156712  0.0033109047  0.0215023421  0.0699421029
## [156]  0.0368527162  0.0056172908  0.0239282717 -0.0387550835 -0.0219719314
## [161] -0.0031654431 -0.0381968352  0.0189692664  0.0343566858  0.0760957342
## [166] -0.0472761636 -0.0391282686  0.0264425076 -0.0369310927 -0.0462459693
## [171] -0.0575353146 -0.0491995012 -0.0322105545 -0.0386329305  0.0051960047
## [176] -0.0627104421 -0.0769181825 -0.0734780989 -0.0526455986 -0.0313337954
## [181] -0.0560338115 -0.0329130011 -0.0331281219 -0.0514380481 -0.0444866340
## [186]  0.0113867909  0.0260517439  0.0239759086  0.0537717903 -0.0007158346
## [191]  0.0142239720 -0.0112992149  0.0134622093 -0.0529657215 -0.0487678565
## [196] -0.0386483131 -0.0313261041 -0.0284686336 -0.0051112718 -0.0043822329
## [201]  0.0299890249 -0.0247883859 -0.0467599385 -0.0462513920 -0.0357187134
## [206] -0.0785443385  0.0425291240 -0.0979439950 -0.0401906343 -0.0865462114
## [211]  0.0464737714  0.0624053966  0.0297769075  0.0546722380 -0.0393794645
## [216] -0.0330207877 -0.0431689655 -0.0304389611 -0.0437758586  0.0343136433
## [221] -0.0231759684  0.0285083634  0.0017469023  0.0038487658 -0.0074297877
## [226] -0.0270832415  0.0014275114  0.1011040866  0.0591351985  0.0956687099
## [231]  0.0452516558 -0.0314486250 -0.0352467703 -0.0536466509 -0.0457429022
## [236] -0.0404480152 -0.0387943405 -0.0362972636 -0.0479106225 -0.0430604640
```

```
## [241] -0.0486959089 -0.0927446620 -0.0458440016 -0.0232164514 -0.0556115374
## [246] -0.0521606834 -0.0304757149  0.0229431072  0.0544770015  0.0511560003
## [251] -0.0148635032 -0.0185725600 -0.0074296570 -0.0435607690 -0.0138505749
## [256]  0.0052465932 -0.0043764862 -0.0144258390 -0.0249242562  0.0198214416
## [261]  0.0642766323  0.0009771138  0.0753584946  0.0316785325 -0.0464824541
## [266] -0.0370895379 -0.0328596939 -0.0440134006 -0.0341669885 -0.0285511419
## [271] -0.0434293583 -0.0398020369 -0.0564114055 -0.0450591762 -0.0609952175
## [276] -0.0617796181 -0.0429209597 -0.0451707244  0.0061365587 -0.0092635245
## [281] -0.0549219823  0.0015412724 -0.0076102657  0.0343651559  0.0393081518
## [286]  0.0686868353  0.0472678318  0.0436373664  0.0627015722  0.0156725181
## [291]  0.0795028552 -0.0385636198 -0.0455400626 -0.0285964592 -0.0582116546
## [296] -0.0612543084  0.0690632037 -0.0770897036  0.0156470067 -0.0427223762
## [301] -0.0395545757 -0.0496963725 -0.0506968404 -0.0303135592 -0.0017046347
## [306] -0.0087546558 -0.0225775031 -0.0443573789 -0.0435033639 -0.0437813168
## [311] -0.0485350844 -0.0308797841 -0.0536640917 -0.0525972623 -0.0123095508
## [316] -0.0479094325 -0.0479392474 -0.0516276972 -0.0425109825 -0.0437363444
## [321] -0.0546538195 -0.0320026498 -0.0556337671 -0.0524564892 -0.0691030779
## [326] -0.0791094666 -0.0770478994 -0.0324709621 -0.0469756925 -0.0348044917
## [331] -0.0242134910 -0.0832522285 -0.0448633563 -0.0468768811 -0.0334407798
## [336] -0.0289810503 -0.0221769049 -0.0111689521 -0.0262984632 -0.0384202398
## [341] -0.0391928719 -0.0366134058 -0.0416826259 -0.0467430579 -0.0484823830
## [346] -0.0413886438 -0.0449460255 -0.0396754210 -0.0621218314 -0.0415091035
## [351] -0.0409263670 -0.0494669445 -0.0531754098 -0.0579821540 -0.0779715594
## [356] -0.0704026879 -0.0261679767 -0.0469680637 -0.0058973790  0.0773327010
## [361] -0.0133240374 -0.0184776957 -0.0609360319 -0.0631681056 -0.0469590504
## [366] -0.0591682525 -0.0357728405 -0.0347555035 -0.0525730092 -0.0277397871
## [371] -0.0441966944 -0.0405221388 -0.0389932483 -0.0442496154 -0.0241469726
## [376] -0.0406042030 -0.0433357694 -0.0401499285 -0.0402037676 -0.0422213556
## [381] -0.0189351671  0.0437370572 -0.0198756094  0.0336456329 -0.0350361988
## [386] -0.0253165887 -0.0488854403 -0.0485093627 -0.0488580342 -0.0202610817
## [391] -0.0565574235 -0.0632933080
```

```
# orthogal vectors have dot product of zero
round(auto_df_svd$u[,1] %*% auto_df_svd$u[,2])
```

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

2.3 Sampling in Bootstrapping.

As we discussed in class, in bootstrapping we start with n data points and repeatedly sample many times with replacement. Each time, we generate a candidate data set of size n from the original data set. All parameter estimations are performed on these candidate data sets. It can be easily shown that any particular data set generated by sampling n points from an original set of size n covers roughly 63.2% of the original data set. Using probability theory and limits, please prove that this is true. After that, write a program to perform this sampling and show that the empirical observation also agrees this.

Proof (with probability theory and limits)

- For collection of integers from 1 to n, let S be collection of those integers drawn randomly and with replacement

- One any one particular sample from S, the probability of drawing the number s1 is 1/n.
- Therefore, the probability of NOT drawing the number s1 on that sample is 1-(1/n)
- Likewise, the probability of NOT drawing the number s2 on that sample is also 1-(1/n)
- Therefore, the probability of NOT drawing the number s1 OR s2 on that sample is (1-(1/n))^2
- Thus for a subset of size n of the original set S, the probability of not drawing ANY of the integers in that size-n set is (1-(1/n))^n
- Now take the limit as n -> infinity of (1-(1/n))^n, and you get 1-(1/e), or about 0.632

Proof (empirical)

```
boostrap_sample <- function(n_num_data_points){
  the_sample <- sample.int(n_num_data_points, size = n_num_data_points, replace = TRUE)
  the_sample <- sort(unique(the_sample))
  pct_found <- length(the_sample)/n_num_data_points
  return (pct_found)
}

boostrap_sample(100)
```

```
## [1] 0.67
```

```
boostrap_sample(1000)
```

```
## [1] 0.628
```

```
boostrap_sample(10000)
```

```
## [1] 0.6305
```

---

3. Mini-project - 20 points

In this mini project, you'll perform a Multivariate Linear Regression analysis using Stochastic Gradient Descent. The data set consists of two predictor variables and one response variable. The predictor variables are living area in square feet and number of bedrooms. The response variable is the price of the house. You have 47 data points in total.

Since both the number of rooms and the living area are in different units, it makes it hard to compare them in relative terms. One way to compensate for this is to standardize the variables. In order to standardize, you estimate the mean and standard deviation of each variable and then compute new versions of these variables. For instance, if you have a variable x, then then standardized version of x is xstd = (x - mew)/sigma where mew and sigma are the mean and standard deviation of x, respectively.

```
x_predictors <- read.table("mini-project-data/ex3x.dat", header=FALSE)
colnames(x_predictors) <- c("x1","x2")

y_response <- read.table("mini-project-data/ex3y.dat", header=FALSE)
colnames(y_response) <- c("y")
```

```r
mtrx <- cbind(y_response, x_predictors$x1, x_predictors$x2)
colnames(mtrx) <- c("y","x1","x2")
head(mtrx)
```

```
##        y   x1 x2
## 1 399900 2104  3
## 2 329900 1600  3
## 3 369000 2400  3
## 4 232000 1416  2
## 5 539900 3000  4
## 6 299900 1985  4
```

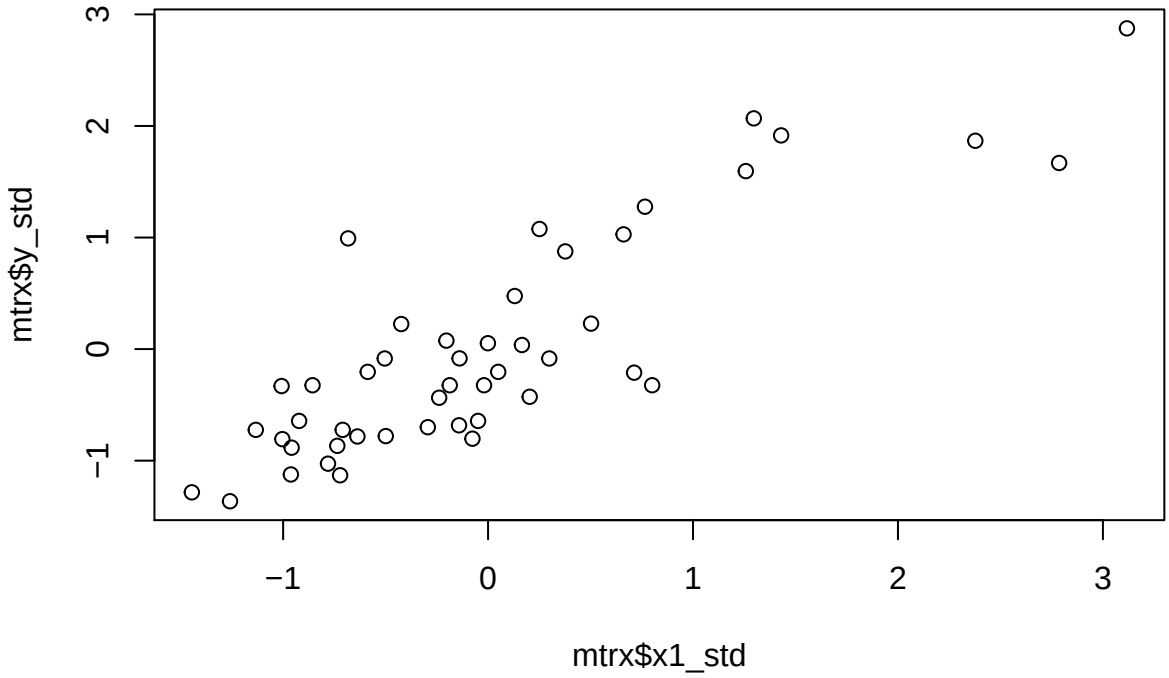
```r
nrow(mtrx)
```

```
## [1] 47
```

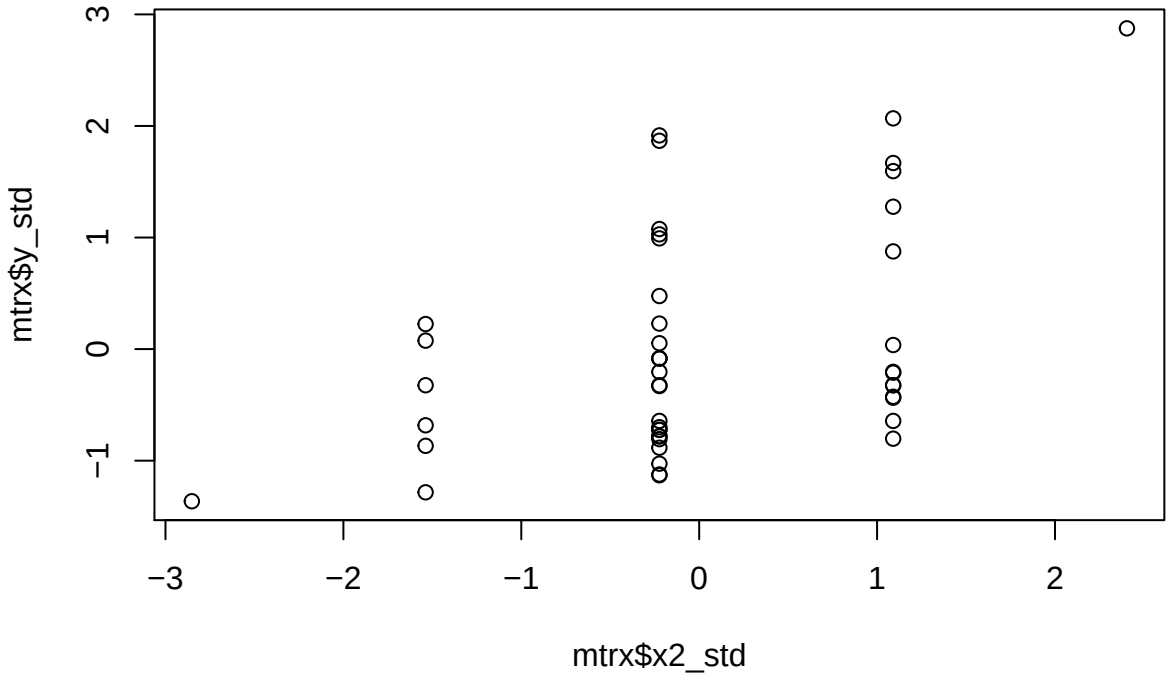
```r
mtrx$y_std <- (mtrx$y - mean(mtrx$y))/sd(mtrx$y)

mtrx$x1_std <- (mtrx$x1 - mean(mtrx$x1))/sd(mtrx$x1)
mtrx$x2_std <- (mtrx$x2 - mean(mtrx$x2))/sd(mtrx$x2)

head(mtrx)
```

```
##        y   x1 x2       y_std      x1_std     x2_std
## 1 399900 2104  3  0.47574687  0.13000987 -0.2236752
## 2 329900 1600  3 -0.08407444 -0.50418984 -0.2236752
## 3 369000 2400  3  0.22862575  0.50247636 -0.2236752
## 4 232000 1416  2 -0.86702453 -0.73572306 -1.5377669
## 5 539900 3000  4  1.59538948  1.25747602  1.0904165
## 6 299900 1985  4 -0.32399786 -0.01973173  1.0904165
```

```r
# plot the 2 independently to get the idea of them...
plot(mtrx$x1_std, mtrx$y_std)
```




```
plot(mtrx$x2_std, mtrx$y_std)
```



As we saw in the gradient descent equations, we introduce a dummy variable x0 = 1 in order to calculate the intercept term of the linear regression. Please standardize the 2 variables, introduce the dummy variable and then write a function to perform gradient descent on this data set. You'll repeat gradient descent for a range of alpha values. Please use alpha = (0:001; 0:01; 0:1; 1:0) as your choices.

```
x1_and_x2_mtrx <- as.matrix(cbind(mtrx$x1_std, mtrx$x2_std))
y_mtrx <- as.matrix(mtrx$y_std)

# define the gradient function dJ/dtheata: 1/m * (h(x)-y))*x
# where h(x) = x*theta in matrix form this is as follows:
grad <- function(x, y, theta, alpha) {
  gradient <- (1/nrow(y))* (t(x) %*% ((x %*% t(theta)) - y))
  return(t(gradient))
}

# define gradient descent update algorithm
grad.descent <- function(x, y, maxit,alpha, start_pt){
  num_iterations <- 0
  dummy_var <- c(start_pt, start_pt)
  theta <- matrix(dummy_var, nrow=1) # Initialize
  for (i in 1:maxit) {
    this_theta <- theta - alpha  * grad(x, y, theta, alpha)
    # do this so num_iterations will only get incremented if the value is different:
    x_equal <- (round(this_theta[1],digits=5) != round(theta[1],digits=5))
    y_equal <- (round(this_theta[2],digits=5) != round(theta[2],digits=5))
    if(x_equal && y_equal){
```

```
      theta <- this_theta
      num_iterations <- i
    }else{
      break
    }
  }

  return (list("theta" = theta, "num_iterations" = num_iterations))
}
```

For each value of alpha perform about 500 SGD iterations with 5 randomly picked samples in each iteration and compute J(theta) at the end of each iteration. When you perform SGD, you randomly pick a mini-batch (in this case 5 samples), use that mini-batch to compute the gradient, and then take a step to improve the objective function. You repeat this process in each iteration. It is very important to randomly pick samples in each iteration - otherwise SGD will not work.

Please plot J(theta) versus number of iterations for each of the 4 alpha choices.

```
# init some constants:
alpha_starts <- c(0.001, 0.01, 0.1, 1.0)
start_pts <- sample(-1000:1000, 5)
five_0s <- c(0,0,0,0,0)
expected_result_x <- 0.885
expected_result_y <- -0.053


calc_results_df <- data.frame(tmp1=five_0s,tmp2=five_0s,tmp3=five_0s,tmp4=five_0s)
colnames(calc_results_df) <- alpha_starts

ggplot_df <- data.frame(tmp1=five_0s,tmp2=five_0s,tmp3=five_0s,tmp4=five_0s)
colnames(ggplot_df) <- alpha_starts

rownames(calc_results_df) <- start_pts
calc_results_df
```

```
##      0.001 0.01 0.1 1
## 11       0    0   0 0
## -920     0    0   0 0
## 326      0    0   0 0
## -745     0    0   0 0
## 470      0    0   0 0
```

```
numiterations_for_alpha <- c()
start_pts_for_alpha <- c()

ALPHAS <- c()
START_PT_X <- c()
START_PT_Y <- c()
END_PT_X <- c()
END_PT_Y <- c()
ITERATIONS <- c()

# set learning rate
```

```
for (alpha in alpha_starts){
  results_for_alpha <- c()
  for(start_pt in start_pts){
    func_result <- grad.descent(x1_and_x2_mtrx, y_mtrx, 500, alpha, start_pt)
    ALPHAS <- c(ALPHAS,alpha)
    END_PT_X <- c(END_PT_X,func_result$theta[1])
    END_PT_Y <- c(END_PT_Y,func_result$theta[2])
    ITERATIONS <- c(ITERATIONS,func_result$num_iterations)
    # these are the same for every col, but helps with calculations later
    START_PT_X <- c(START_PT_X,start_pt)
    START_PT_Y <- c(START_PT_Y,start_pt)

    formatted_result <- paste('(',round(func_result$theta[1,1], digits=5),',',round(func_result$theta[1
    calc_results_df[toString(start_pt),toString(alpha)] <- formatted_result
  }

  numiterations_for_alpha <- c(numiterations_for_alpha, func_result$num_iterations)
}

kable(calc_results_df)
```

	0.001	0.01	0.1	1
11	(5.43692,5.25518)	(0.82379,0.05637)	(0.88457,-0.05299)	(0.88476,-0.05318)
-920	(-428.23474,-428.41648)	(0.41076,-0.4188)	(0.88457,-0.05299)	(0.88476,-0.05318)
326	(152.16794,151.9862)	(0.97908,0.14952)	(0.88457,-0.05299)	(0.88476,-0.05318)
-745	(-346.71751,-346.89925)	(0.49058,-0.33898)	(0.85837,-0.03839)	(0.88476,-0.05318)
470	(219.24497,219.06323)	(1.04476,0.2152)	(0.88457,-0.05299)	(0.88476,-0.05318)

```
FINAL_DF <- data.frame(alpha=ALPHAS, iterations=ITERATIONS, START_X=round(START_PT_X,digits = 3), START_

FINAL_DF$DX <- round((FINAL_DF$END_PT_X - expected_result_x),digits = 3)
FINAL_DF$DY <- round((FINAL_DF$END_PT_Y - expected_result_y),digits = 3)

FINAL_DF$SUCCESS <- (abs(FINAL_DF$DX - FINAL_DF$DY) <= 0.001)

# REMINDERS OF WHERE WE WANTED TO GET TO:
expected_result_x
```

```
## [1] 0.885
```

```
expected_result_y
```

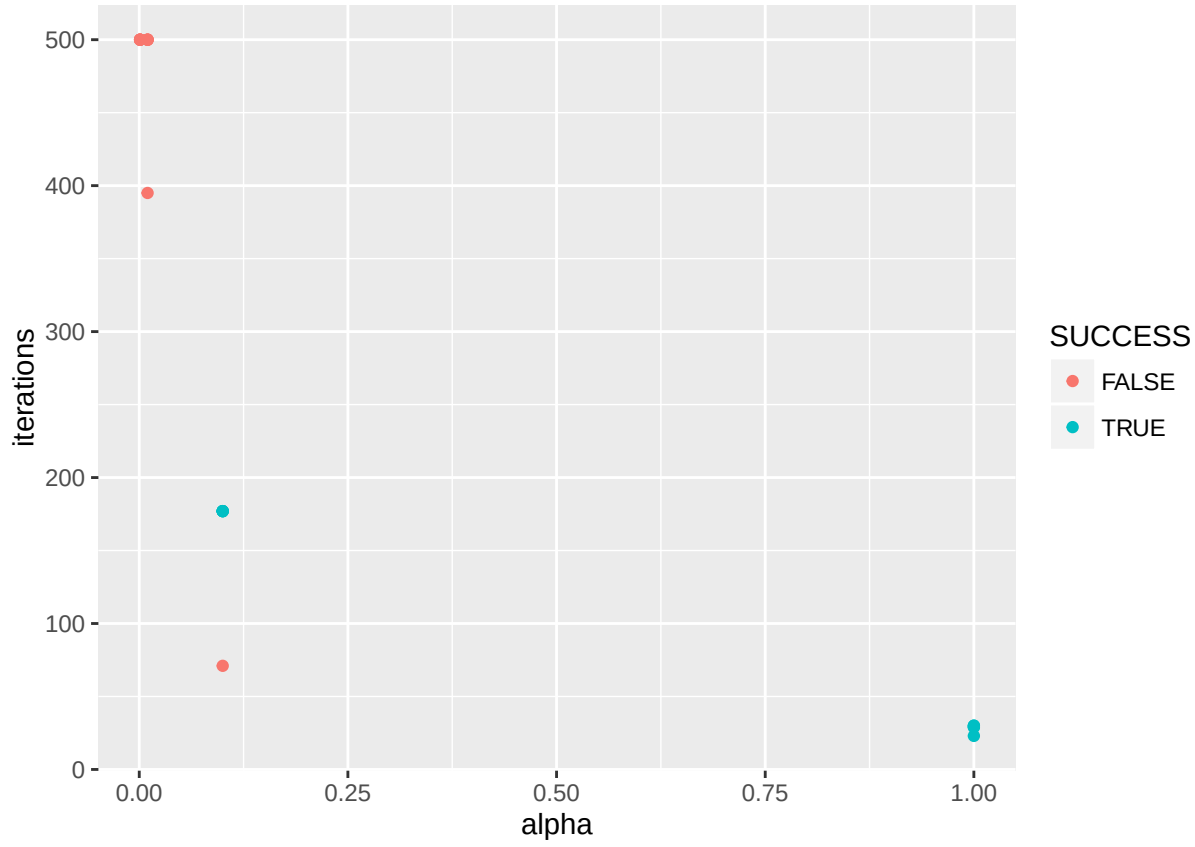
```
## [1] -0.053
```

```
kable(FINAL_DF)
```

alpha	iterations	START_X	START_Y	END_PT_X	END_PT_Y	DX	DY	SUCCESS
0.001	500	11	11	5.437	5.255	4.552	5.308	FALSE
0.001	500	-920	-920	-428.235	-428.416	-429.120	-428.363	FALSE

alpha	iterations	START_X	START_Y	END_PT_X	END_PT_Y	DX	DY	SUCCESS
0.001	500	326	326	152.168	151.986	151.283	152.039	FALSE
0.001	500	-745	-745	-346.718	-346.899	-347.603	-346.846	FALSE
0.001	500	470	470	219.245	219.063	218.360	219.116	FALSE
0.010	395	11	11	0.824	0.056	-0.061	0.109	FALSE
0.010	500	-920	-920	0.411	-0.419	-0.474	-0.366	FALSE
0.010	500	326	326	0.979	0.150	0.094	0.203	FALSE
0.010	500	-745	-745	0.491	-0.339	-0.394	-0.286	FALSE
0.010	500	470	470	1.045	0.215	0.160	0.268	FALSE
0.100	177	11	11	0.885	-0.053	0.000	0.000	TRUE
0.100	177	-920	-920	0.885	-0.053	0.000	0.000	TRUE
0.100	177	326	326	0.885	-0.053	0.000	0.000	TRUE
0.100	71	-745	-745	0.858	-0.038	-0.027	0.015	FALSE
0.100	177	470	470	0.885	-0.053	0.000	0.000	TRUE
1.000	23	11	11	0.885	-0.053	0.000	0.000	TRUE
1.000	30	-920	-920	0.885	-0.053	0.000	0.000	TRUE
1.000	29	326	326	0.885	-0.053	0.000	0.000	TRUE
1.000	30	-745	-745	0.885	-0.053	0.000	0.000	TRUE
1.000	29	470	470	0.885	-0.053	0.000	0.000	TRUE

```
ggplot(data = FINAL_DF[,c('alpha','iterations','SUCCESS')], aes(x=alpha, y=iterations)) + geom_point(aes
```



Some Poitnts about the Plot:

Interesting that sometimes the red dots appeared up at 500 iterations, other times they appeared with values lower. Values lower show: "Hey we got stuck in a ditch. . . could keep on iterating up to 500, but not going to go anywhere. . . "

Once you have your final gradient descent solution, compare this with regular linear regression (using the built-in function in R). Please document both solutions in your submission. How does the SGD solution differ from the Linear Regression solution? Are they different? If so, why? If not, why not?

```
solved <- solve(t(x1_and_x2_mtrx) %*% x1_and_x2_mtrx) %*% t(x1_and_x2_mtrx) %*% y_mtrx
solved
```

```
##             [,1]
## [1,]  0.88476599
## [2,] -0.05317882
```

Answer:

(After rounding the hand calculated results):

They are NOT the same for the smaller alpha "jumps" of 0.001 and 0.01

They ARE the same for the bigger alpha "jumps" of 0.1 and 1

This seems to be because the smaller "jumps" of 0.001 and 0.01 are not large enough to get out of the "valleys", ie - local minimums, while 0.1 and 1 are large enough to get out of those same local minimums.