CSMMA16 - Assignment

Deyan Dyankov (student id 24885531)

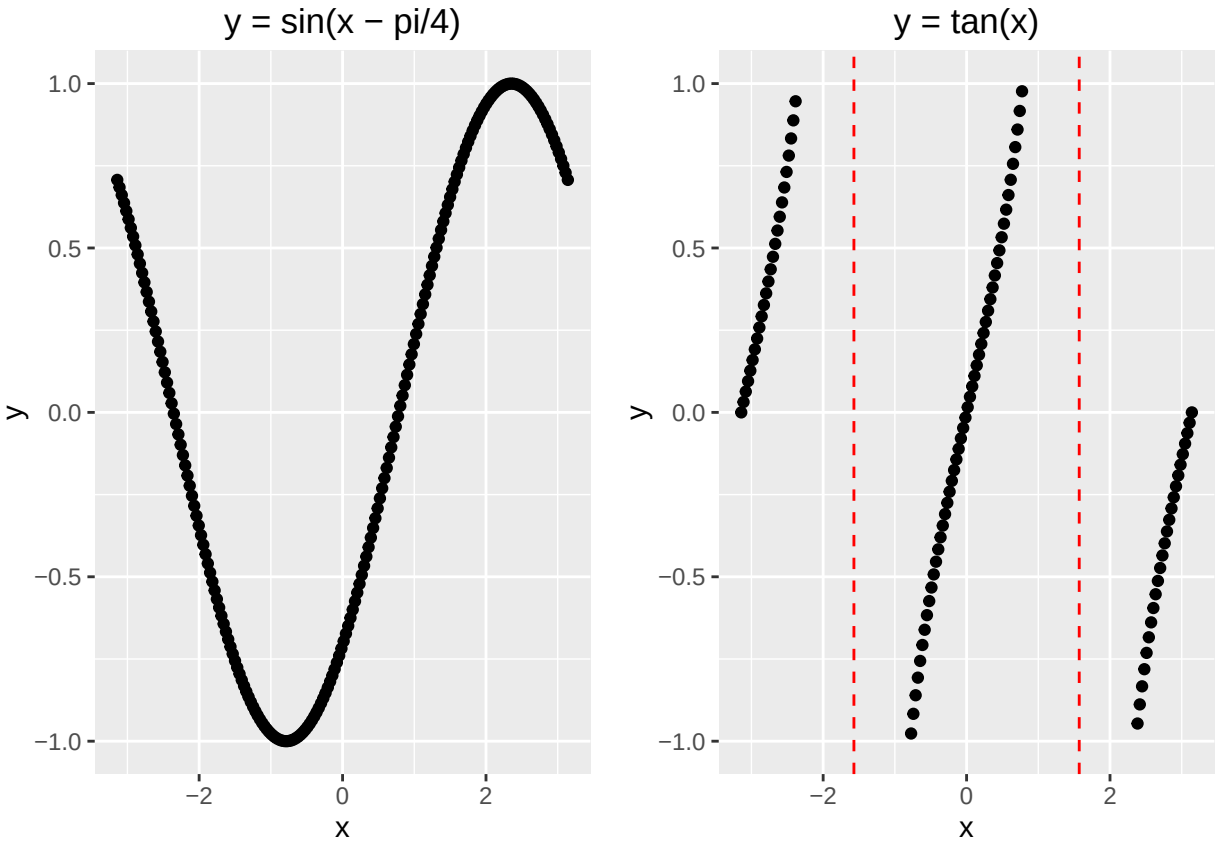
11 October 2016

Question 1

Construct plots of $\sin(x - \pi/4)$, and $\tan(x)$ for x between $-\pi$ and π on two separate plots in the same figure. Use abline to show the asymptotes. Add appropriate titles and labels to your plots.

Solution

```
# prepare the data
x <- seq(from = -pi, to = pi, length.out = 200)
df <- data.frame(
        x = x,
        y_sin = sin(x - pi/4),
        y_tan = tan(x)
)

# plots
library(ggplot2)
# y = sin(x = pi/4), no asymptotes
plot_sin <- ggplot(df, aes(x, y_sin)) +
        ggtitle("y = sin(x - pi/4)") +
        ylab("y") +
        ylim(c(-1, 1)) +
        xlim(c(-pi, pi)) +
        geom_point()

# y = tan(x), vertical asymptotes at -pi/2 and pi/2
plot_tan <- ggplot(df, aes(x, y_tan)) +
        ggtitle("y = tan(x)") +
        ylab("y") +
        ylim(c(-1, 1)) +
        xlim(c(-pi, pi)) +
        geom_point() +
        geom_vline(xintercept = -pi/2, col = "red", linetype = "dashed") +
        geom_vline(xintercept = pi/2, col = "red", linetype = "dashed")

# display plots
source("multiplot.R")
multiplot(plot_sin, plot_tan, cols=2)
```



Question 2

Calculate by hand the eigenvalues and eigenvectors of $A = \begin{pmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{pmatrix}$ and check your answer in R. Comment on any differences. Use R to obtain singular values and matrices in the singular value decomposition of A and compare those with the eigenvalues and eigenvectors in the eigen decomposition. Comment on the results. Determine if A is positive definite.

Solution

First, let's calculate the eigenvalues for A by hand. This involves finding $det(A - \lambda I_3) = 0$ where

$$A - \lambda I_3 = \begin{vmatrix} 1-\lambda & -1 & 0 \\ -1 & 2-\lambda & -1 \\ 0 & -1 & 1-\lambda \end{vmatrix}$$

Therefore

$$det(A - \lambda I_3) = (1-\lambda) \times \begin{vmatrix} 2-\lambda & -1 \\ -1 & 1-\lambda \end{vmatrix} + \begin{vmatrix} -1 & 0 \\ -1 & 1-\lambda \end{vmatrix} =$$

$$(1-\lambda) \times (2 - 2\lambda - \lambda + \lambda^2 - 1) + (-1 + \lambda) = (1-\lambda) \times (\lambda^2 - 3\lambda + 1) - 1(1-\lambda)$$

So

$$det(A - \lambda I_3) = (1-\lambda) \times (\lambda^2 - 3\lambda) = 0$$

This is a third degree polynomial with three solutions which are our eigenvalues:

$\lambda = 0$ $\lambda = 1$ $\lambda = 3$

This is in line with what R gives us as eigenvalues:

```
A = matrix(c(1, -1, 0, -1, 2, -1, 0, -1, 1), nrow = 3, ncol = 3)
e <- eigen(A)
e$values
```

```
## [1] 3.000000e+00 1.000000e+00 2.664535e-15
```

To calculate eigenvectors of A we need to solve $(A - \lambda I_3)x = 0$ for every λ

Let $\lambda = 0$

$$\begin{vmatrix} 1-0 & -1 & 0 \\ -1 & 2-0 & -1 \\ 0 & -1 & 1-0 \end{vmatrix} \times \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix}$$

This translates to the following linear system of equations.

$x_1 - x_2 = 0$

$-x_1 + 2x_2 - x_3 = 0$

$-x_2 + x_3 = 0$

from which we get that $x_1 = x_2 = x_3$ and we can set our first eigen vector to whatever values we like as long as they are equal to each other. If we choose 1 to be the value, then we get $(1, 1, 1)$ as the first eigenvector.

Let $\lambda = 1$

$$\begin{vmatrix} 1-1 & -1 & 0 \\ -1 & 2-1 & -1 \\ 0 & -1 & 1-1 \end{vmatrix} \times \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix}$$

therefore

$-x_2 = 0$

$-x_1 + x_2 - x_3 = 0$

$-x_2 = 0$

The solutions are $x_1 = -x_3$, $x_2 = 0$, $x_3 = -x_1$. We then set our second eigen vector to $(-1, 0, 1)$.

Let $\lambda = 3$

$$\begin{vmatrix} 1-3 & -1 & 0 \\ -1 & 2-3 & -1 \\ 0 & -1 & 1-3 \end{vmatrix} \times \begin{vmatrix} x_1 \\ x_2 \\ x_3 \end{vmatrix}$$

therefore

$-2x_1 - x_2 = 0$

$-x_1 - x_2 - x_3 = 0$

$-x_2 - 2x_3 = 0$

The solutions are $x_1 = \frac{-x_2}{2}$, $x_2 = -2x_3$, and x_3 can have any value. We can then set our third eigen vector to $(1, -2, 1)$.

R is choosing different eigen vectors but they still map to our values:

```r
A = matrix(c(1, -1, 0, -1, 2, -1, 0, -1, 1), nrow = 3, ncol = 3)
e <- eigen(A)
e$values
```

```
## [1] 3.000000e+00 1.000000e+00 2.664535e-15
```

```r
e$vectors
```

```
##            [,1]          [,2]      [,3]
## [1,] -0.4082483 -7.071068e-01 0.5773503
## [2,]  0.8164966  9.420555e-16 0.5773503
## [3,] -0.4082483  7.071068e-01 0.5773503
```

For example R has chosen (0.5773503, 0.5773503, 0.5773503) where we chose (1, 1, 1), etc.

Question 3

Write your own R function to return the angle (in degrees) between two arbitrary vectors α and β.

Solution

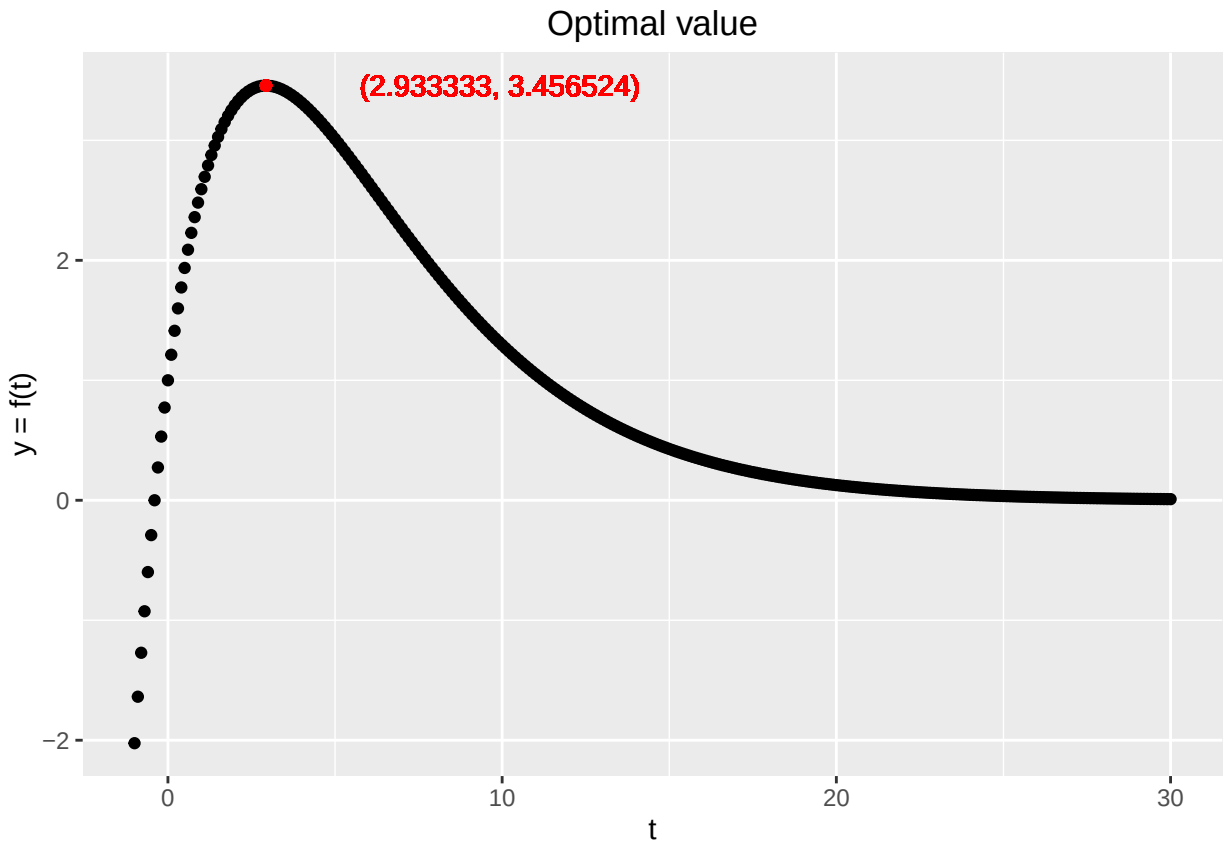
```r
# testing library, we later use expect_equal() to test our function
library(testthat)
vector_angle <- function(a, b) {
        # this function is undefined for the (0, 0) point
        if (identical(a, c(0, 0)) || identical(b, c(0, 0))) return(NA)
        rad <- acos(sum(a * b) / (sqrt(sum(a^2)) * sqrt(sum(b^2))))
        angle <- rad * 180/pi
        return(angle)
}

# produces an error unless the angle between alpha (1st param)
# and beta (2nd param) equals the 3rd parameter
expect_equal(vector_angle(c(1, 0), c(0, 1)), 90)
expect_equal(vector_angle(c(1, 0), c(1, 1)), 45)
expect_equal(vector_angle(c(1, 0), c(-1, 0)), 180)
expect_equal(vector_angle(c(1, 0), c(0, -1)), 90)
```

Question 4

Find the optimal value of the function $y = (1 + 2.5t)e^{-0.3t}$ and the point $t.max$ at which this optima occurs by writing your own code in R implementing the Newton-Raphson algorithm. Use R to iterate until $t.max$ is found accurate to 4dp.

Solution

```r
f <- function(t) (1 + 2.5*t) * exp(-0.3 * t)
t <- seq(-1, 30, by = 0.1)
y <- round(f(t), 4)

# get the derivative
dx <- deriv(~ (1 + 2.5*t) * exp(-0.3 * t), c("t"), func = TRUE, hessian = TRUE)

# fetch the gradient and hessian functions returned by deriv()
grad <- function(t) attr(dx(t), "gradient")[1]
hess <- function(t) attr(dx(t), "hessian")[1]

# recursive function that allows us to loop over values of t
get_t_init <- function(t_init) {
        result <- t_init - (grad(t_init) / hess(t_init))
        if (abs(result - t_init) < 0.0001)
                return(result)
        get_t_init(result)
}

# get optimal value for t
t_init <- get_t_init(5)

# some plotting
library(ggplot2)
df <- data.frame(t = t, y = y)
v_label <- paste0("(", round(t_init, 6), ", ", round(f(t_init), 6), ")")
ggplot(df, aes(x = t, y = y)) +
        ggtitle("Optimal value") +
        ylab("y = f(t)") +
        geom_point() +
        geom_point(aes(x = t_init, y = f(t_init)), colour = "red") +
        geom_text(aes(x = t_init + 7, y = f(t_init), label = v_label), colour = "red")
```



Question 5

Consider a system with differential equations given by $\frac{d^2O}{dt^2} + 4\frac{dO}{dt} + 3O = 1$ Determine the transient and steady state solutions for O, and hence find the particular solution if $O = 1$ and $dO/dt = 0$ at $t = 0$. Using R, obtain a numerical solution and plot graphs showing the theoretical and numerical solutions for t in the interval (0, 8). Comment on the plots.

Solution

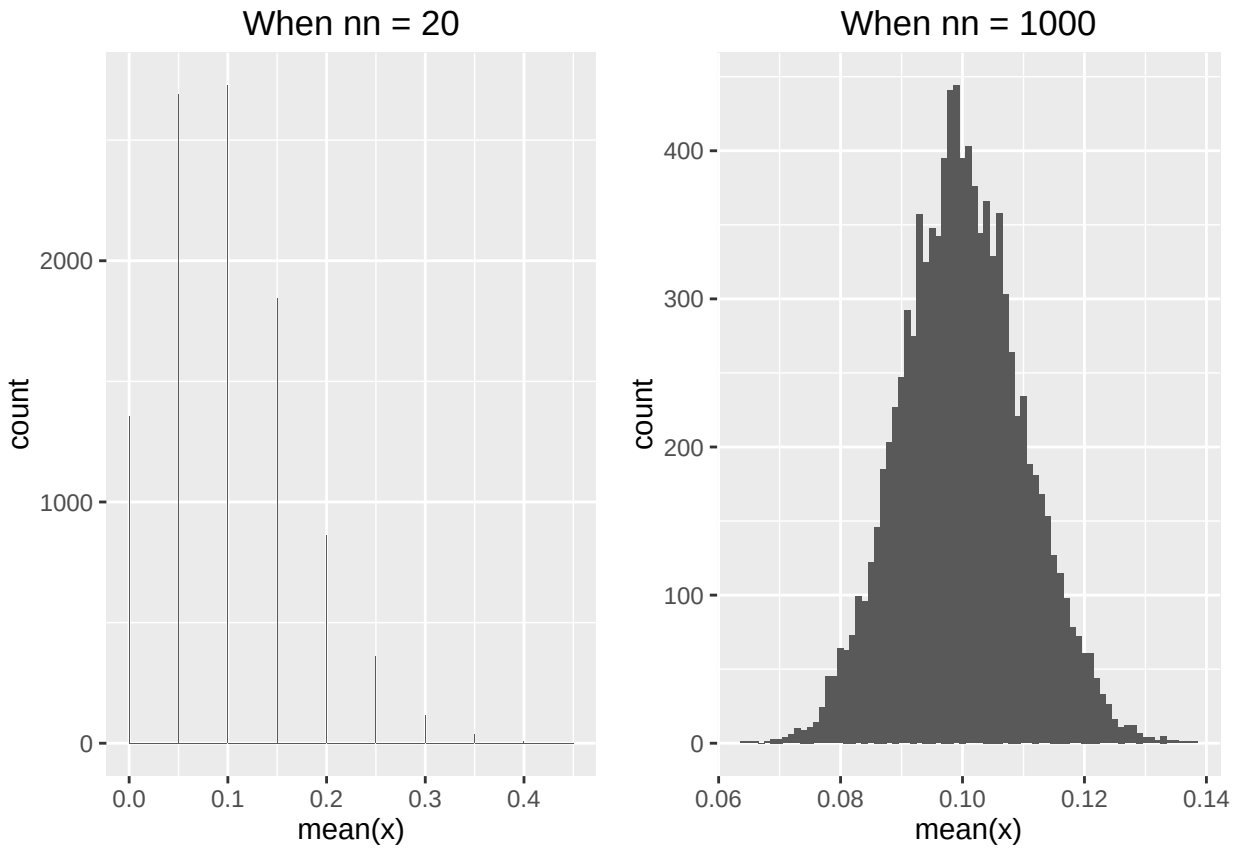
no solution currently.

Question 6

Write a function to genetate m datasets, each of sample size nn, from a binomial distribution with $p = 0.01$ and trial size 10 to calculate the mean of each dataset. Do not write your own loop in R; instead use the **apply** function. Use this function in the **apply** function to generate 10000 means each from studies with sample sizes $nn = 20$ and $nn = 100$, respectively. Plot histograms of the sample means for each sample size so that the two histograms are in the same figure. Use the Central Limit Theorem to explain your findings.

Solution

```r
set.seed(23) # make sure we get the same figures in every simulation

# Generate m datasets with sample size nn from a binomial distribution
gen_binomial_means <- function(nn, m) {
        trials <- 10
        binom_distributions <- matrix(rbinom(m * nn, trials, 0.01), nrow = nn, ncol = m)
        result <- apply(binom_distributions, 2, mean)
        return(result)
}

# get 10000 means for two groups of x (sizes 20 and 1000)
means <- lapply(c(20, 1000), gen_binomial_means, 10000)

# Plot histograms of the two groups of means
group1 <- data.frame(x = means[[1]])
group2 <- data.frame(x = means[[2]])

library(ggplot2)
group1_plot <- ggplot(group1, aes(x = x)) +
        geom_histogram(binwidth = 0.001) +
        xlab("mean(x)") +
        ggtitle("When nn = 20")

group2_plot <- ggplot(group2, aes(x = x)) +
        geom_histogram(binwidth = 0.001) +
        xlab("mean(x)") +
        ggtitle("When nn = 1000")

source("multiplot.R")
multiplot(group1_plot, group2_plot, cols=2)
```



To put it simply, there are only so many *unique* mean values of a dataset of length 20. For example:

```
length(unique(group1$x)) # when nn = 20
```

```
## [1] 10
```

```
length(unique(group2$x)) # when nn = 1000
```

```
## [1] 74
```

Question 7

Write a function that uses R command **t.test** to calculate and return a 9% confidence interval for the mean μ based on a randomly generated sample of size 25 from a normal distribution with mean of 10 and a standard deviation of 1. Use this function to generate 100 confidence intervals and count how often these intervals obtain the mean *mu*. Comment on your results.

Solution

```
set.seed(23) # make sure results are reproducible
sample_size <- 25
theoretical_mean <- 10
theoretical_sd <- 1
```

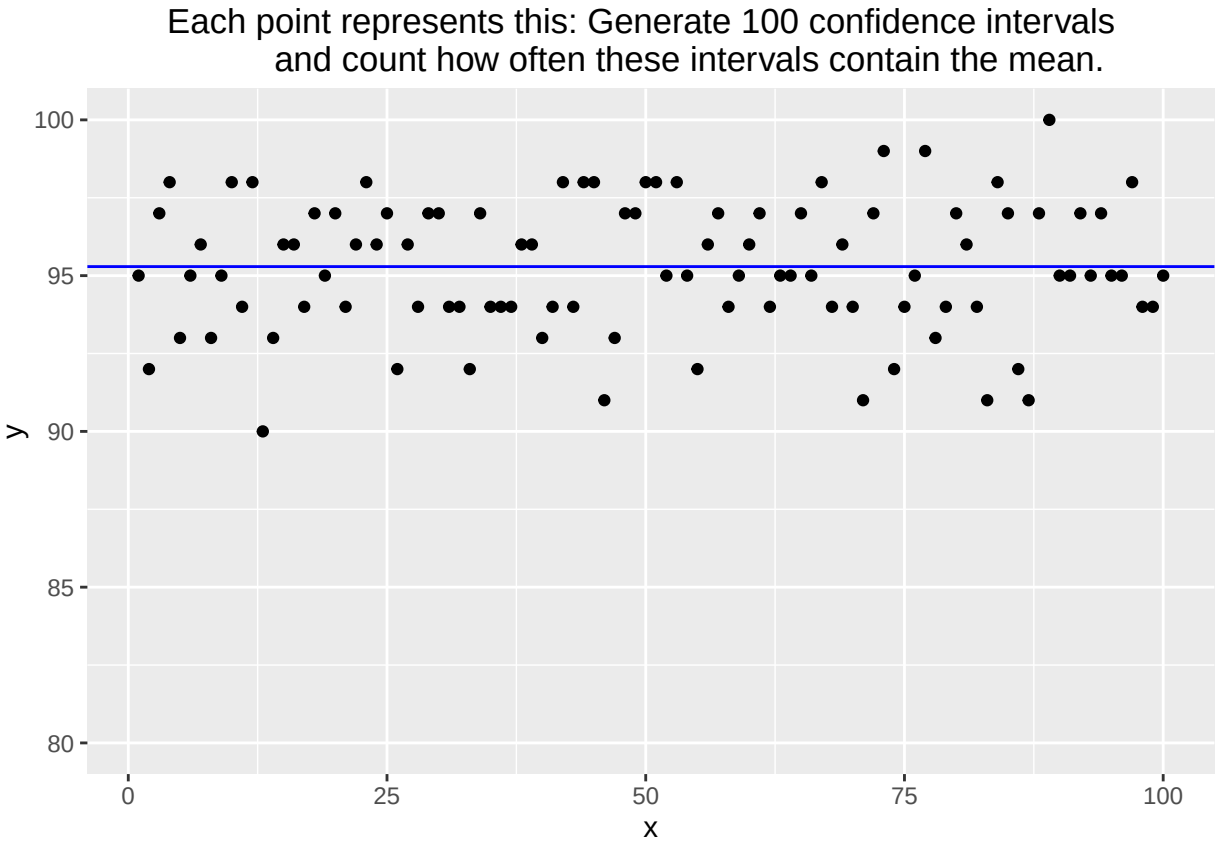
```r
samples <- 100

confidence_interval_hit <- function() {
        t <- t.test(
                rnorm(sample_size, theoretical_mean, theoretical_sd),
                alternative = c("two.sided"),
                mu = theoretical_mean
        )
        in_interval <- theoretical_mean > t$conf.int[1] & theoretical_mean < t$conf.int[2]
        return(in_interval)
}

# this number should be roughly 95
pct_hits <- sum(sapply(1:samples, function(x) confidence_interval_hit()))
print("Generate 100 confidence intervals and count how often they contain the mean")
```

```
## [1] "Generate 100 confidence intervals and count how often they contain the mean"
```

```r
print(paste0("Result: ", pct_hits))
```

```
## [1] "Result: 99"
```

```r
# everything below this line is just playing around.
# let's run the above 100 times and plot the results
# the expectation is that y will be close to 95 most of the time
y <- c()
for (i in 1:samples) {
        y <- c(y, sum(sapply(1:samples, function(x) confidence_interval_hit())))
}
library(ggplot2)
ggplot(data.frame(x = 1:length(y), y = y), aes(x = x, y = y)) +
        ggtitle("Each point represents this: Generate 100 confidence intervals \
        and count how often these intervals contain the mean.") +
        geom_point() +
        ylim(c(80, 100)) +
        geom_hline(yintercept = mean(y), col = "blue") # will always be very close to 95!
```



Question 8

The following figure shows 3 faces from a face database.

The originals can be found in Blackboard under Assignments. Calculate the *mean* face, and produce an image of it. Also, obtain and produce on one figure images of the difference of each face from the mean face.

Based on the covariance matrix of the differences, calculate eigenfaces (equal to eigenvectors) and produce the first 3 eigenfaces on the same figure.

Solution

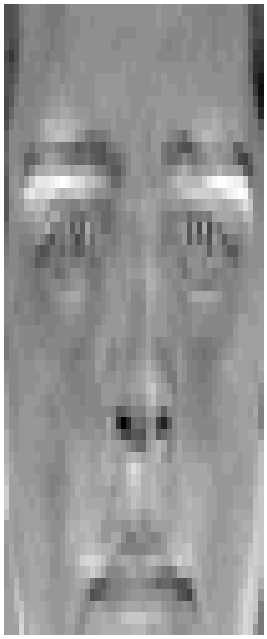
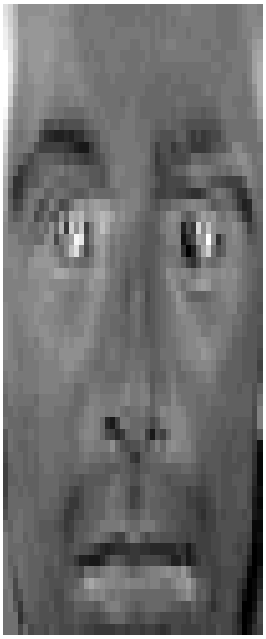
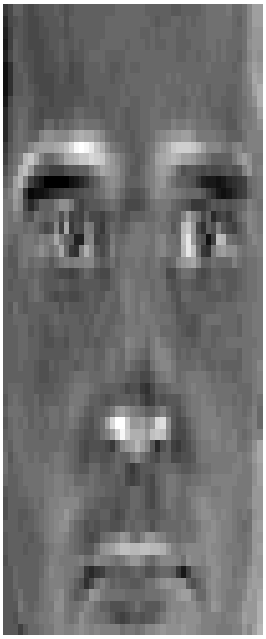
```
library(bmp) # used for load images via read.bmp()
image1 <- read.bmp("images/1.bmp")
image2 <- read.bmp("images/2.bmp")
image3 <- read.bmp("images/3.bmp")

# calculate rows and columns, they're the same for all images.
# we'll use this to create the mean_face later on.
rows <- nrow(image1) # 55
cols <- ncol(image1) # 51

# flatten the images from 2 dimentional matrices to 55*51 vectos.
f1 <- as.vector(image1)
f2 <- as.vector(image2)
f3 <- as.vector(image3)

# the three faces in a single matrix (face per column)
faces <- matrix(c(f1, f2, f3), nrow = length(f1), ncol = 3)

# the average of all three faces
mean_face <- matrix(apply(faces, 1, mean), nrow = rows, ncol = cols)

# used for plotting
rotate_matrix <- function(x) t(x)[, nrow(x):1]
```

Produce an image of the mean face

```
image(rotate_matrix(mean_face), col = grey((0:32)/32), axes = F)
```

Produce on one figure images of the difference of each face from the mean.

```
diff1 <- image1 - mean_face
diff2 <- image2 - mean_face
diff3 <- image3 - mean_face
par(mfrow = c(1, 3))
image(rotate_matrix(diff1), col = grey((0:32)/32), axes = F)
image(rotate_matrix(diff2), col = grey((0:32)/32), axes = F)
image(rotate_matrix(diff3), col = grey((0:32)/32), axes = F)
```

Based on the covariance matrix of the differences, calculate eigenfaces and produce the first three eigenfaces on the same figure.

```
differences <- matrix(
        c(as.vector(diff1), as.vector(diff2), as.vector(diff3)),
        nrow = length(f1),
        ncol = 3
)
cov_matrix <- differences %*% t(differences)
eigenfaces <- eigen(cov_matrix)$vectors
w1 <- f1 %*% eigenfaces[, 1]
w2 <- f2 %*% eigenfaces[, 2]
w3 <- f3 %*% eigenfaces[, 3]
eigenface1 <- matrix(w1 * eigenfaces[, 1] + mean_face, nrow = rows, ncol = cols)
eigenface2 <- matrix(w2 * eigenfaces[, 2] + mean_face, nrow = rows, ncol = cols)
eigenface3 <- matrix(w3 * eigenfaces[, 3] + mean_face, nrow = rows, ncol = cols)

par(mfrow = c(1, 3))
image(rotate_matrix(eigenface1), col = grey((0:32)/32), axes = F)
image(rotate_matrix(eigenface2), col = grey((0:32)/32), axes = F)
image(rotate_matrix(eigenface3), col = grey((0:32)/32), axes = F)
```