Problem Set 1

Marc Eskew

4/6/2022

Problem 1

By conducting 10,000 replications of the marble drop we can calculate $P(\omega \in (C \cup S)^c)$. First the observations of u and v are generated and classified if they are within S or C:

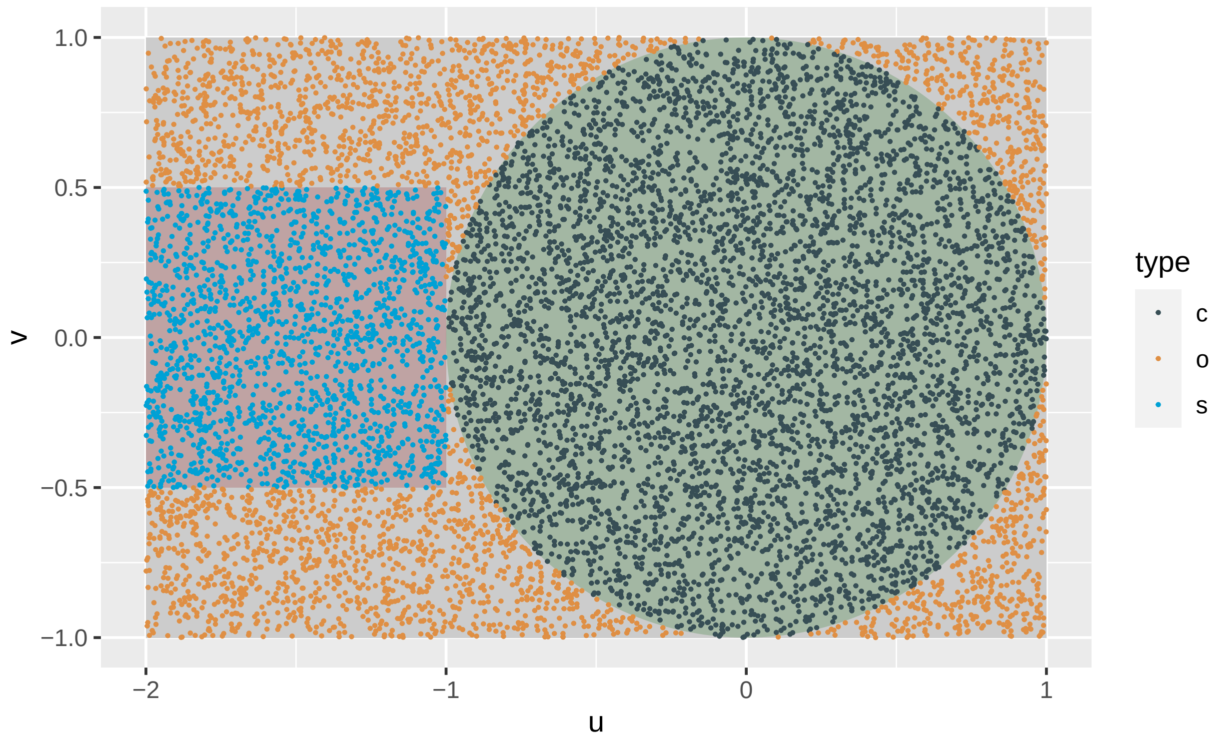
```
umin <- -2
umax <- 1
vmin <- -1
vmax <- 1

u <- 3*runif(10000)-2
v <- 2*runif(10000)-1

df1 <- data.frame(cbind(u,v)) %>%
  mutate(type = case_when(
    u^2 +v^2 <= 1 ~ "c",
    u >= -2 & u <= -1 & v <= .5 & v >= -.5 ~ "s",
    T ~ "o"
  ))
```

We can visualize this on a plot:

```
ggplot() +
  geom_rect(aes(xmin = umin, xmax = umax, ymin = vmin, ymax = vmax),
            fill = "grey80") +
  geom_circle(aes(x0 = 0, y0 = 0, r=1),fill = "darkgreen",
              color = "transparent",alpha = .2) +
  geom_rect(aes(xmin = -2, xmax = -1, ymin = -.5, ymax = .5),fill = "darkred",
            color = "transparent",alpha = .2) +
  geom_point(data=df1,aes(x = u, y = v, color = type),size = .25) +
  ggsci::scale_color_jama() +
  coord_fixed()
```



The probability can be calculated through the marbles not containted within C or S.

```
df1a <- df1 %>%
  group_by(type) %>%
  summarise(num = n()) %>%
  filter(type == "o")

probest <- df1a$num/10000

probest
```

```
## [1] 0.313
```

The probability of a wasted marble based on this simulation is **0.313**, which is close to the true value $\frac{5-\pi}{6} \approx .3097$.

Problem 2

Replicating this procedure 100 times to estimate π and calculating the error:

```
simulapi <- function(iter=10000) {  #MC function to estimate pi

  u <- 3*runif(iter)-2
  v <- 2*runif(iter)-1

  df1 <- data.frame(cbind(u,v)) %>%
    mutate(type = case_when(
      u^2 +v^2 <= 1 ~ "c",
```

```
      u >= -2 & u <= -1 & v <= .5 & v >= -.5 ~ "s",
      T ~ "o"
    )) %>%
    group_by(type) %>%
    summarise(num = n())

   c_val <- filter(df1, type == 'c')[,2]/iter
   s_val <- filter(df1, type == 's')[,2]/iter

   c_val/s_val

}


df_iters <- data.frame() #Initiate empty dataframe

for(i in 1:100){ #Loop simulapi function and append result to df

  val <- simulapi()

  df_iters <- bind_rows(df_iters,val)
}

head(df_iters) #Show some values of pi
```

```
##        num
## 1 3.179408
## 2 3.228294
## 3 3.177352
## 4 3.064478
## 5 3.057625
## 6 3.144643
```

```
out1 <- (max(df_iters$num) - min(df_iters$num))/ pi #Calculate error
out1
```

```
## [1] 0.1914538
```

The magnitude of error between extreme estimates is **0.1914538**.

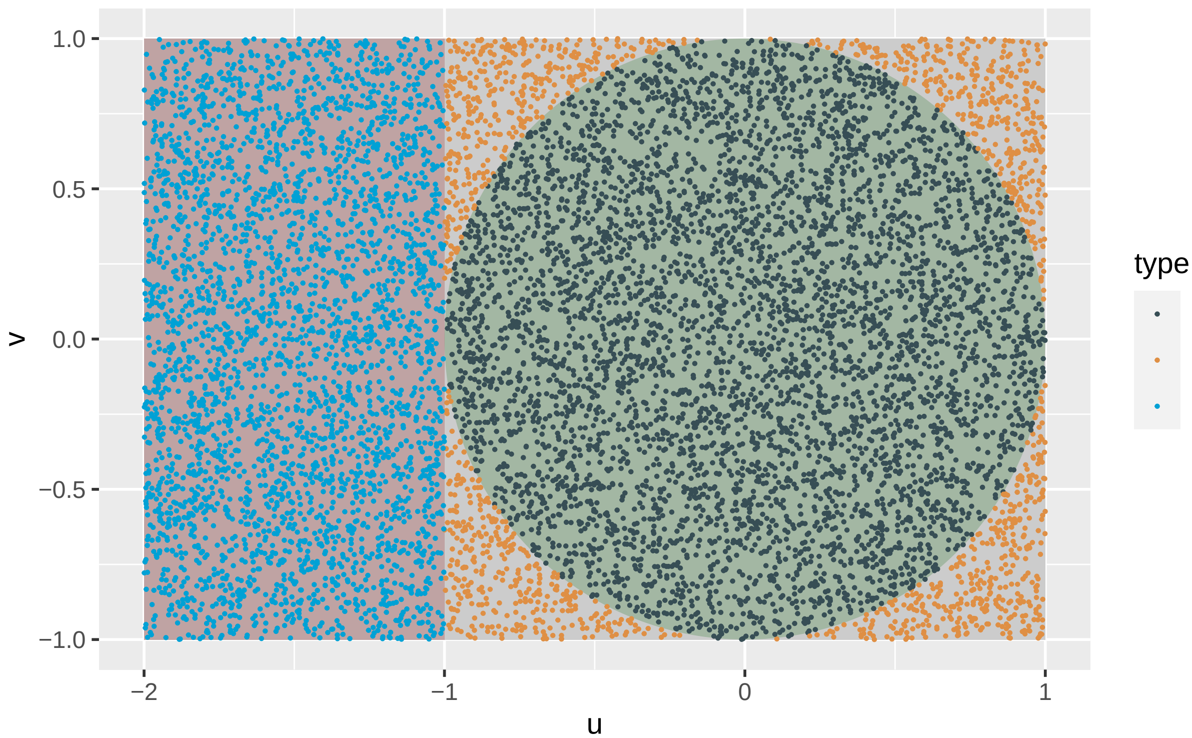
Problem 3

With the definition of new set R we can show how that effects our simulation:

```
df2 <- data.frame(cbind(u,v)) %>%
  mutate(type = case_when(
    u^2 +v^2 <= 1 ~ "c",
    u >= -2 & u <= -1 & v <= 1 & v >= -1 ~ "r",
    T ~ "o"
  ))

ggplot() +
  geom_rect(aes(xmin = umin, xmax = umax, ymin = vmin, ymax = vmax),
            fill = "grey80") +
  geom_circle(aes(x0 = 0, y0 = 0, r=1),fill = "darkgreen",
              color = "transparent",alpha = .2) +
```

```
  geom_rect(aes(xmin = -2, xmax = -1, ymin = -1, ymax = 1),fill = "darkred",
            color = "transparent",alpha = .2) +
  geom_point(data=df2,aes(x = u, y = v, color = type),size = .25) +
  ggsci::scale_color_jama() +
  coord_fixed()
```



Now, $P(\omega \in R) = \frac{1}{3}$. We modify the code understanding now that $2\frac{P(\omega \in c)}{P(\omega \in R)} = \pi$ and run the same process to sample and estimate π. With

```
simulapi2 <- function(iter=10000) {

  u <- 3*runif(iter)-2
  v <- 2*runif(iter)-1

  df1 <- data.frame(cbind(u,v)) %>%
    mutate(type = case_when(
      u^2 +v^2 <= 1 ~ "c",
      u >= -2 & u <= -1 & v <= 1 & v >= -1 ~ "s",
      T ~ "o"
    )) %>%
    group_by(type) %>%
    summarise(num = n())

  c_val <- filter(df1, type == 'c')[,2]/iter
  s_val <- filter(df1, type == 's')[,2]/iter
```

```
  2*c_val/s_val

}


df_iters2 <- data.frame()

for(i in 1:100){

  val <- simulapi2()

  df_iters2 <- bind_rows(df_iters2,val)
}

head(df_iters2)
```

```
##        num
## 1 3.103203
## 2 3.281782
## 3 3.144928
## 4 3.091662
## 5 3.286726
## 6 3.142261
```

```
out2 <- (max(df_iters2$num) - min(df_iters2$num))/ pi

out2
```

```
## [1] 0.1372213
```

The error using this method is lower; **0.1372213**. There is a **28%** reduction in error using set R instead of set S.

To examine the increase in effiency we can see that the probability of a wasted marble is reduced in this instance: $P(\omega \in (C \cup R)^c) = \frac{4-\pi}{6}$. Therefore, efficiency is increased by reducing the probability of a wasted marble by $\frac{(5-\pi)-(4-\pi)}{(5-\pi)} = \frac{1}{5-\pi}$ or approximately **54%**.

Problem 4

The most efficient way to conduct this estimation is to eliminate uninformative samples. We can do this by defining Ω and R as such:

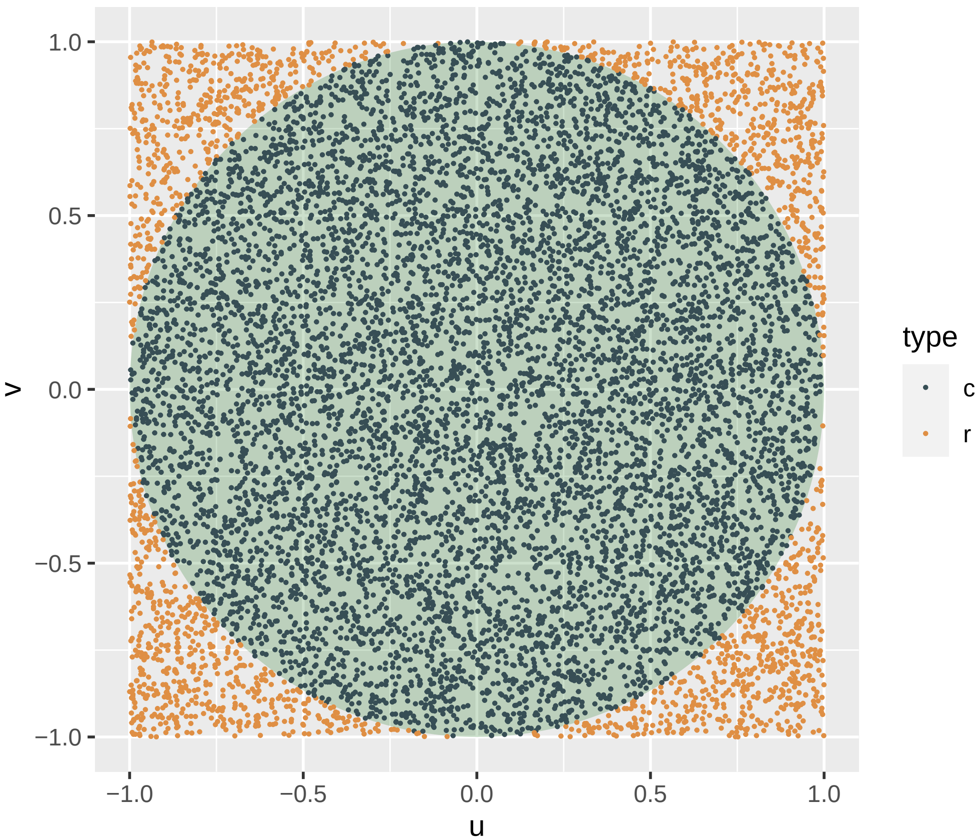
$$\Omega = \{(u,v)| -1 \leq u \leq 1, -1 \leq v \leq 1\}$$
$$R = \{(u,v) : u^2 + v^2 > 1\}$$

A simulation with these parameters would utilize all of Ω as informative space.

```
u <- 2*runif(10000)-1
v <- 2*runif(10000)-1

df1b <- data.frame(cbind(u,v)) %>%
  mutate(type = case_when(
    u^2 +v^2 <= 1 ~ "c",
    TRUE ~ "r"
  ))
```

```
ggplot() +
  geom_circle(aes(x0 = 0, y0 = 0, r=1),fill = "darkgreen",
              color = "transparent",alpha = .2) +
  geom_point(data=df1b,aes(x = u, y = v, color = type), size = .25) +
  ggsci::scale_color_jama() +
  coord_fixed()
```



The formula for value of π must be updated with the latest probabilities.

$$
\begin{aligned}
\frac{P(\omega \in C)}{P(\omega \in R)} &= \frac{\frac{\pi}{4}}{1-\frac{\pi}{4}} \\
&= \frac{\pi}{4-\pi} \\
P(\omega \in C)(4-\pi) &= \pi P(\omega \in R) \\
4P(\omega \in C) - \pi P(\omega \in C) &= \pi P(\omega \in R) \\
4P(\omega \in C) &= \pi(P(\omega \in C) + P(\omega \in R)) \\
\frac{4P(\omega \in C)}{P(\omega \in C) + P(\omega \in R)} &= \pi
\end{aligned}
$$

Modifying the functions used for the previous simulations we can again estimate π and determine an error.

```
simulapi3 <- function(iter=10000) {

  u <- 2*runif(iter)-1
  v <- 2*runif(iter)-1
```

```r
  df1 <- data.frame(cbind(u,v)) %>%
    mutate(type = case_when(
      u^2 +v^2 <= 1 ~ "c",
      T ~ "s"
    )) %>%
    group_by(type) %>%
    summarise(num = n())

  c_val <- filter(df1, type == 'c')[,2]/iter
  s_val <- filter(df1, type == 's')[,2]/iter

  4*c_val/(c_val+s_val)

}


df_iters3 <- data.frame()

for(i in 1:100){

  val <- simulapi3()

  df_iters3 <- bind_rows(df_iters3,val)
}

head(df_iters3)
```

```
##      num
## 1 3.1336
## 2 3.1792
## 3 3.1588
## 4 3.1236
## 5 3.1532
## 6 3.1224
```

```r
out3 <- (max(df_iters3$num) - min(df_iters3$num))/ pi
out3
```

```
## [1] 0.02457352
```

The error using this method, 0.0245735, is much lower due to the gain in efficiency of the experiment.