HOMEWORK 1

Group 12

Estimation of Constants and Randomness Testing

This document focuses on estimating two mathematical constants, Euler's number e and π, using probabilistic methods. Additionally, we perform a statistical test to verify the randomness of a dataset. The document integrates theoretical explanations and practical implementations.

Q1: Estimation of Euler's Number(e)

Euler's number, approximately 2.718, is the base of natural logarithms. It frequently appears in growth models, calculus, and statistics. We use three different methods to estimate e and compare their rate convergence.

Method 1: Using Uniform Distribution

Theory

If $U_1, U_2, \ldots, U_n$ are i.i.d Uniform(0,1), then sample geometric mean $\prod_{i=1}^{n} U_i^{1/n}$ converges in probability to $1/e$. So, reciprocal of sample geometric mean will converge to e in probability for large n.

Implementation

```r
e.est1 <- function(n) {
  # Generate 'n' random binomial values
  dat <- runif(n)
  # Sample geometric mean
  geo.mean <- prod(dat^(1/n))
  e.hat <- 1/geo.mean
  return(e.hat)
}
```

Method 2: Using Binomial Distribution

Theory

For a binomial random variable $X \sim Binomial(n, 1/n)$ the probability of $X = 0$ approaches $1/e$ as n becomes large. Thus, e can be estimated as,

$$\hat{e} = \frac{n}{\text{number of zeros in the sample}}$$

Implementation

```r
e.est2 <- function(n) {
  # Generate 'n' random binomial values
  dat <- rbinom(n, n, 1/n)
  # Count the number of zeros and calculate e
  e.hat <- n / sum(dat == 0)
  return(e.hat)
}
```

Method 3: Using Card Shuffling

Theory

A derangement is a permutation in which no element appears in its original position. The probability of a derangement for a set of n elements is approximately $(1/e)$. Using this property,

$$\hat{e} = \frac{\text{number of shuffles}}{\text{number of derangements}}$$

Implementation

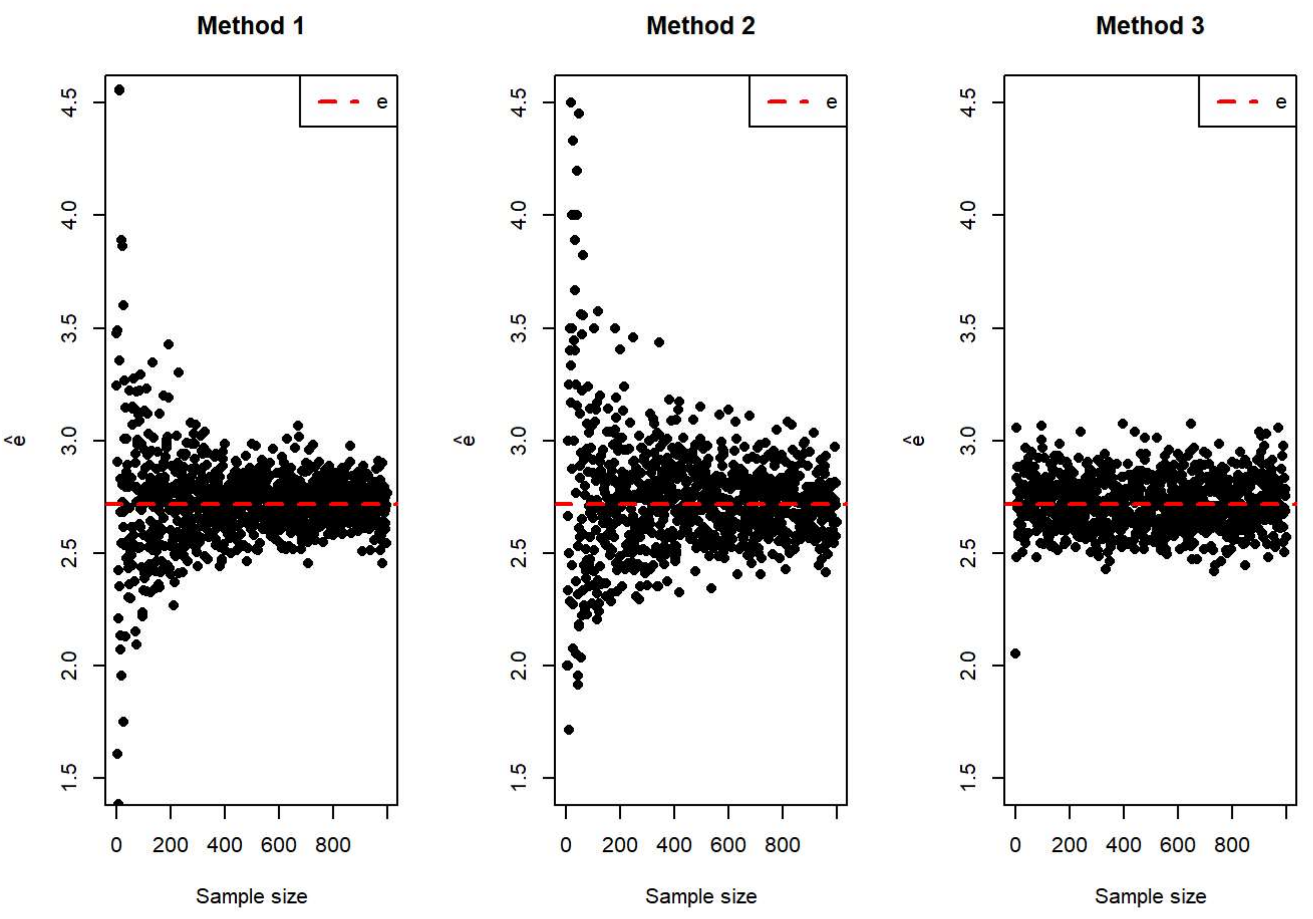
```r
e.est3 <- function(n)  # n = #of cards
{
  arranged.deck <- 1:n
  n.shuffle <- 1e3
  derangements <- 0
  for(i in 1:n.shuffle)
  {
    shuffled.deck <- sample(arranged.deck)
    if(sum(shuffled.deck == arranged.deck) == 0)
    {
      derangements <- derangements + 1
    }
  }
  e.hat <- n.shuffle / derangements
  return(e.hat)
}
```

Comparision of methods

We now compare these three methods by estimating e for increasing sample sizes (up to 1000). We plot the results to analyze the rate of convergence.

```r
set.seed(123)

# Initialize vectors to store estimates
m <- 1e3
e.hat1 <- numeric(m)
e.hat2 <- numeric(m)
e.hat3 <- numeric(m)
```

```r
# Compute estimates for each method
for (i in 1:m) {
  e.hat1[i] <- e.est1(i)
  e.hat2[i] <- e.est2(i)
  e.hat3[i] <- e.est3(i)
}

# Plot the results
par(mfrow = c(1, 3))
methods <- list(e.hat1, e.hat2, e.hat3)
for (i in 1:3) {
  plot(1:m, methods[[i]],
       pch = 16, ylim = c(1.5, 4.5),
       main = paste("Method", i),
       xlab = "Sample size", ylab = expression(hat(e)))
  abline(h = exp(1), col = "red", lty = 2, lwd = 2)
  legend("topright", legend = expression(e), lty = 2, col = "red", lwd = 2)
}
```



These side by side plots suggests that the rate of convergence of Method 3 is very much faster than the other two methods and hence Method 3 is most efficient. And we can say that the rate of convergence of Method 1 is slightly faster than Method 2.

So to estimate e we can use Method 3 with number of cards 1000.

```
set.seed(123)

estimated.e <- e.est3(1000)
estimated.e
```

```
[1] 2.710027
```

Q2: Estimation of π

Theory

In a 2D plane, the ratio of points inside a unit circle to the total points in a square approximates $\frac{\pi}{4}$. Generalizing this to higher dimensions, we use the volume of a hyper sphere to estimate π.

Volume of a hyper cuboid of edge length d is 2^d and volume of a unit d-dimesional sphere is $\frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}$.

So the probability of choosing a point from the hyper cuboid which also lies inside the hyper sphere is,

$$p = \frac{\frac{\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2}+1)}}{2^d}$$

From this, we can estimate the value of π.

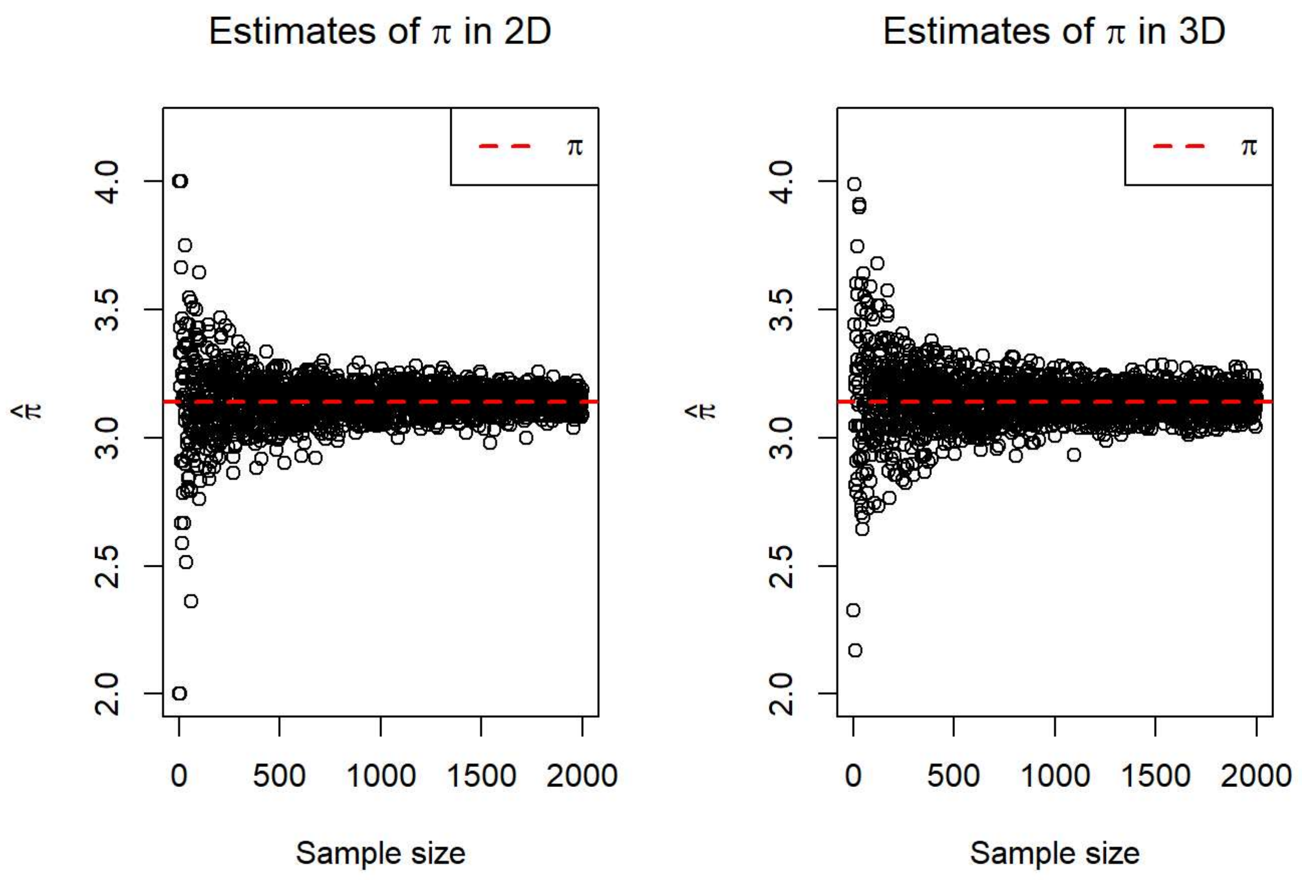
Implementation

```
pi.est <- function(n, d)  # n = sample size, d = dimension
{
  sum <- 0
  for(i in 1:n)
  {
    point <- runif(d, -1, 1)
    if(norm(point, "2") < 1) sum <- sum + 1
  }
  p <- sum / n
  pi.hat <- (p * 2^d * gamma(d/2+1))^(2/d)
  return(pi.hat)
}
```

Estimating π in 2D and 3D for varying sample size.

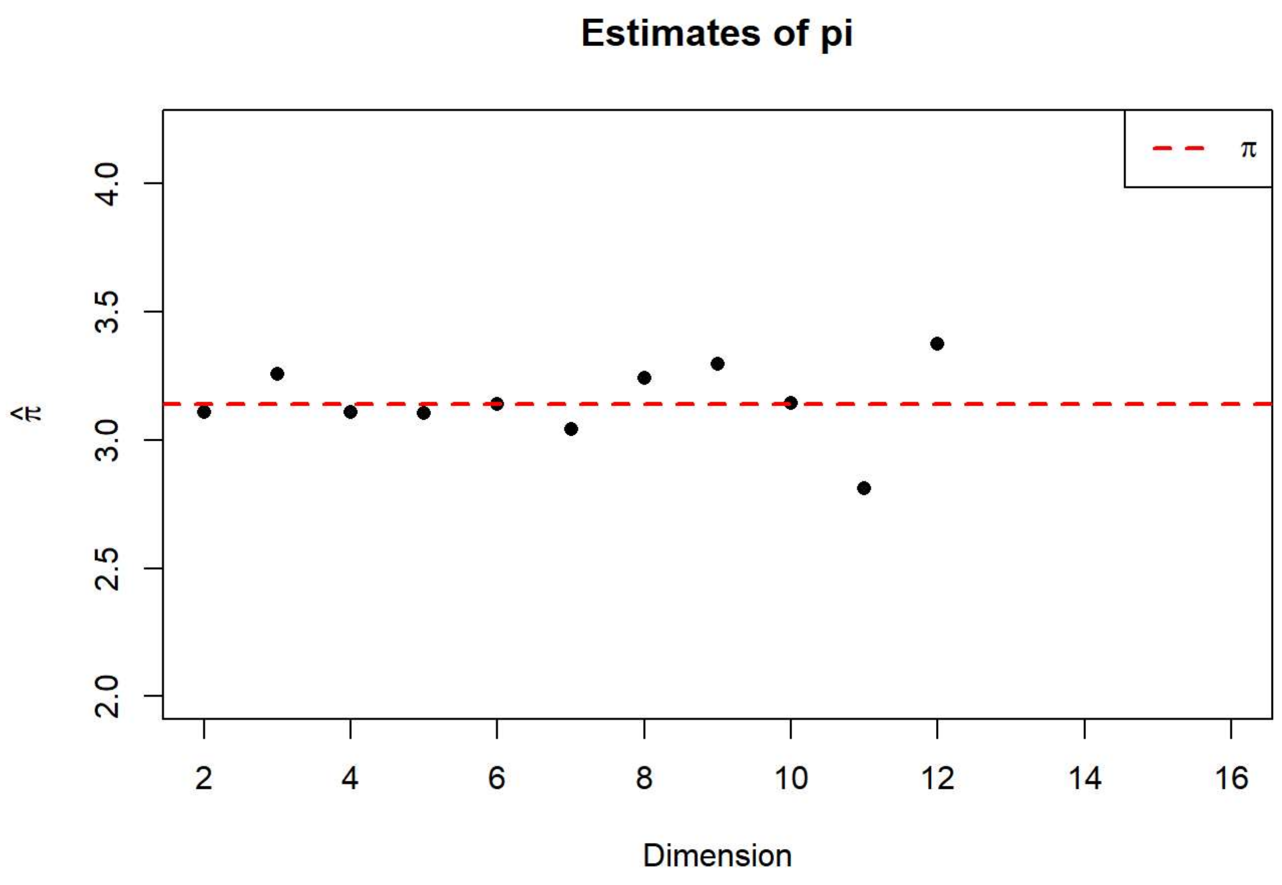
```
set.seed(123)

par(mfrow = c(1,2))
#plotting pi.hat estimated in 2D with increasing sample size
n <- 1:2e3  #vector of sample sizes
```

```r
pi.hat.2D <- numeric(length = 2e3)
for(i in n)
{
  pi.hat.2D[i] <- pi.est(i, 2)
}
plot(n, pi.hat.2D, ylim = c(2,4.2),
     main = expression(paste("Estimates of ", pi," in 2D")),
     xlab = "Sample size", ylab = expression(hat(pi)))
abline(h = pi, col = "red", lty = 2, lwd = 2)
legend("topright", legend = expression(pi), lty = 2, lwd = 2, col = "red")

#plotting pi.hat estimated in 3D with increasing sample size
n <- 1:2e3
pi.hat.3D <- numeric(length = 2e3)
for(i in n)
{
  pi.hat.3D[i] <- pi.est(i, 3)
}
plot(n, pi.hat.3D, ylim = c(2,4.2),
     main = expression(paste("Estimates of ", pi," in 3D")),
     xlab = "Sample size", ylab = expression(hat(pi)))
abline(h = pi, col = "red", lty = 2, lwd = 2)
legend("topright", legend = expression(pi), lty = 2, lwd = 2, col = "red")
```



Now let’s compare the value of estimated π with increasing dimensions.

```r
set.seed(123)

# Plotting pi.hat estimated with fixed sample size 2000 in different dimensions
d <- 2:16   #vector of dimensions
pi.hat <- numeric(length = 15)
for(i in d)
{
  pi.hat[i-1] <- pi.est(2e3, i)
}
plot(d, pi.hat, pch = 16, ylim = c(2, 4.2),
     main = "Estimates of pi",
     xlab = "Dimension", ylab = expression(hat(pi)))
abline(h = pi, col = "red", lty = 2, lwd = 2)
legend("topright", legend = expression(pi), lty = 2, lwd = 2, col = "red")
```



Clearly estimation error is getting bigger in higher dimensions. Also the previous plots suggests large sample size will give a better estimate.

Hence in 2D, 2000 sample size can be taken to get an estimate of π.

```r
set.seed(238)

estimated.pi <- pi.est(2000, 2)
estimated.pi
```

```
[1] 3.124
```

Q3: Test of Randomness

Theory

The run test checks whether a sequence of data is random by evaluating the number of runs (consecutive identical elements) in a binary sequence.

Here, we generates 100 observations from N(0,1) distribution and test whether the generated data is random. We will split the data in several partitions and apply run test on each of them.

Steps

1. The data is split into n partitions.
2. Run test is applied to each partition(For each partition, the test checks whether the sequence of values above or below the median is random or not).

```
test.randomness <- function(data, n)    # n = #partitions of the data
{
  #splitting the data into n many parts
  l = length(data)
  subdata <- vector(mode = 'list', length = n)   #empty list of length n
  if(n > 1)
  {
    for(i in 1:(n-1))
    {
      subdata[[i]] <- data[((i-1)*floor(l/n)+1) : (i*floor(l/n))]
    }
  }
  temp <- data[(n-1)*floor(l/n)+1 : l]
  subdata[[n]] <- temp[!is.na(temp)]

  #applying run test on each of the partition
  p.val <- numeric(length = n)
  for(i in 1:n)
  {
    dat <- subdata[[i]]
    p.val[i] <- runs.test(as.factor(dat > median(dat)))$p.value
  }
  return(p.val)
}

set.seed(123)
data <- rnorm(100)
for(i in 1:6)
{
  print(test.randomness(data, i))
}
```

```
[1] 0.6876578
[1] 0.04545529 0.08641073
[1] 0.5990772 0.3789732 0.4860340
[1] 0.06671289 0.31009132 0.83146037 0.83146037
[1] 1.0000000 0.0660811 0.6458979 0.3581287 0.3581287
[1] 0.604772854 1.000000000 0.009660623 0.604772854 1.000000000 0.358128743
```

If $p > 0.05$: Fail to reject Ho. There is no significant evidence to suggest that the data is non-random.

Here the p values suggest that there is no significant evidence to reject H0 (:Data is random), at 5% level of significance. Hence we conclude that the data is random.