Study of the Empirical Distribution of Ties in Irrational Numbers

Eduarda Chagas

Nov 10, 2020

The purpose of this study is describing the distribution of ties in irrational numbers when considering sequences of embedding dimension D (and $\tau = 1$). The finding will help building a model for simulating such occurrences and, then, assessing imputation techniques in a Monte Carlo study. Thus, in this report we will make an exploratory analysis of the data, analyzing the percentage of tied sequences for each embedding dimension value.

Load packages and sources

```
if(!require(ggpubr)){
  install.packages("ggpubr")
  require(ggpubr)
}
```

```
## Loading required package: ggpubr
```

```
## Loading required package: ggplot2
```

```
if(!require(ggplot2)){
  install.packages("ggplot2")
  require(ggplot2)
}
if(!require(ggthemes)){
  install.packages("ggthemes")
  require(ggthemes)
}
```

```
## Loading required package: ggthemes
```

```
source('Bandt-Pompe.R')
```

```
## Loading required package: gtools
```

Reading the numbers

We have produced to files with Mathematica: e.txt, pi.txt, and sqrt2.txt. They contain expansions of these irrational numbers with $100,000$ digits.

```
e.data = as.character(read.table('../Data/e.txt', stringsAsFactors=FALSE, fileEncoding="latin1"))
pi.data = as.character(unlist(read.table('../Data/pi.txt', stringsAsFactors=FALSE, fileEncoding="latin1
sqrt2.data = as.character(unlist(read.table('../Data/sqrt2.txt', stringsAsFactors=FALSE, fileEncoding="l

e.vector = as.numeric(unlist(strsplit(e.data, ""))[3:100011])
pi.vector = as.numeric(unlist(strsplit(pi.data, ""))[3:100011])
sqrt2.vector = as.numeric(unlist(strsplit(sqrt2.data, ""))[3:100030])
```

Computing the number of tied sequences for each value of $D \in \{3, 4, 5, 6\}$

```
D = 3
Tau = 1

e.elements.D3 = formationPattern(e.vector, D, Tau, 1)
e.percent.D3 = percentual.equalities(e.elements.D3)

pi.elements.D3 = formationPattern(pi.vector, D, Tau, 1)
pi.percent.D3 = percentual.equalities(pi.elements.D3)

sqrt2.elements.D3 = formationPattern(sqrt2.vector, D, Tau, 1)
sqrt2.percent.D3 = percentual.equalities(sqrt2.elements.D3)

cat("Number of tied sequences \ne: ", round(e.percent.D3*100, 3), "%\npi: ", round(pi.percent.D3*100, 3
```

```
## Number of tied sequences
## e:  28.115 %
## pi:  27.913 %
## sqrt2:  27.845 %
```

```
D = 4
e.elements.D4 = formationPattern(e.vector, D, Tau, 1)
e.percent.D4 = percentual.equalities(e.elements.D4)

pi.elements.D4 = formationPattern(pi.vector, D, Tau, 1)
pi.percent.D4 = percentual.equalities(pi.elements.D4)

sqrt2.elements.D4 = formationPattern(sqrt2.vector, D, Tau, 1)
sqrt2.percent.D4 = percentual.equalities(sqrt2.elements.D4)

cat("Number of tied sequences \ne: ", round(e.percent.D4*100, 3), "%\npi: ", round(pi.percent.D4*100, 3
```

```
## Number of tied sequences
## e:  49.622 %
## pi:  49.538 %
## sqrt2:  49.487 %
```

```
D = 5
e.elements.D5 = formationPattern(e.vector, D, Tau, 1)
e.percent.D5 = percentual.equalities(e.elements.D5)

pi.elements.D5 = formationPattern(pi.vector, D, Tau, 1)
pi.percent.D5 = percentual.equalities(pi.elements.D5)

sqrt2.elements.D5 = formationPattern(sqrt2.vector, D, Tau, 1)
sqrt2.percent.D5 = percentual.equalities(sqrt2.elements.D5)

cat("Number of tied sequences \ne: ", round(e.percent.D5*100, 3), "%\npi: ", round(pi.percent.D5*100, 3
```

```
## Number of tied sequences
## e:  69.866 %
## pi:  69.92 %
## sqrt2:  69.757 %
```

```
D = 6
e.elements.D6 = formationPattern(e.vector, D, Tau, 1)
e.percent.D6 = percentual.equalities(e.elements.D6)

pi.elements.D6 = formationPattern(pi.vector, D, Tau, 1)
pi.percent.D6 = percentual.equalities(pi.elements.D6)

sqrt2.elements.D6 = formationPattern(sqrt2.vector, D, Tau, 1)
sqrt2.percent.D6 = percentual.equalities(sqrt2.elements.D6)

cat("Number of tied sequences \ne: ", round(e.percent.D6*100, 3), "%\npi: ", round(pi.percent.D6*100, 3
```

```
## Number of tied sequences
## e:  85.122 %
## pi:  85.145 %
## sqrt2:  84.956 %
```

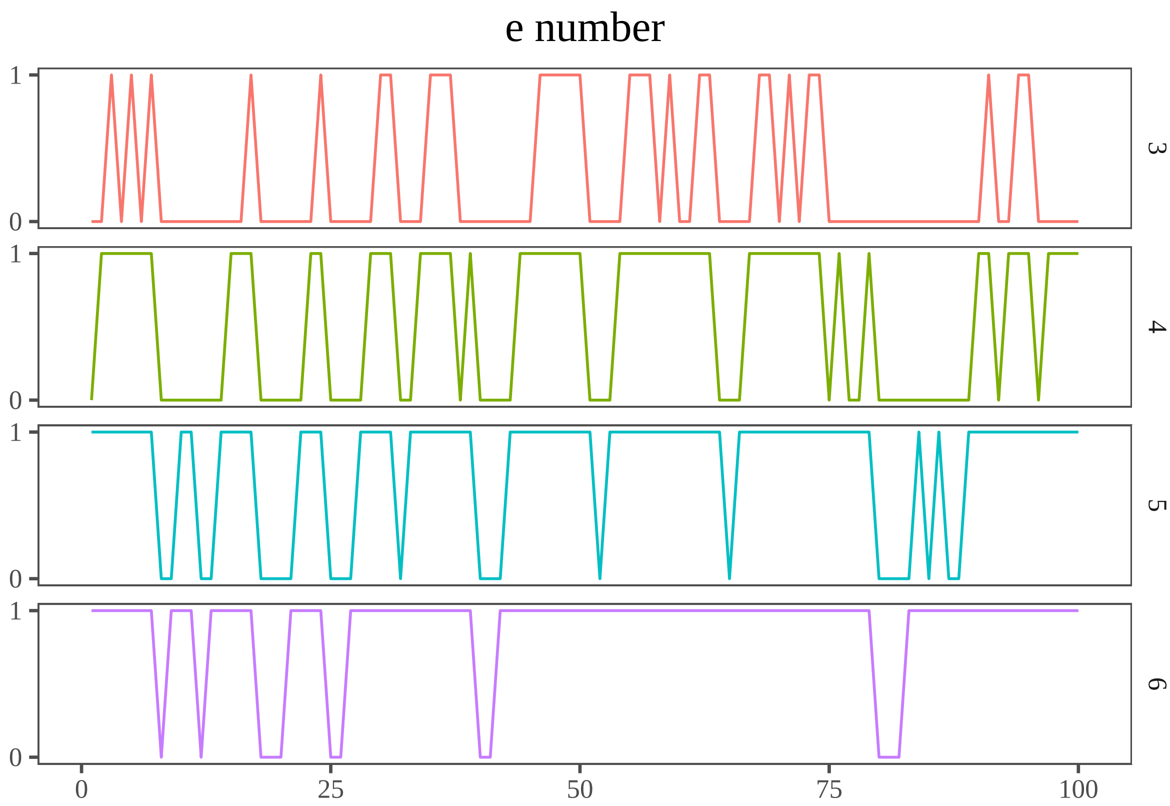
Analyzing the binary vectors with the position of the tied sequences

```
e.binary.D3 = binary.equalities(e.elements.D3)
e.binary.D4 = binary.equalities(e.elements.D4)
e.binary.D5 = binary.equalities(e.elements.D5)
e.binary.D6 = binary.equalities(e.elements.D6)

n.elements = 100

e.binary.df = data.frame('series' = c(e.binary.D3[1:n.elements], e.binary.D4[1:n.elements], e.binary.D5
                         'elements' = rep(c(1:n.elements), 4),
                         'D' = as.factor(c(rep(3, n.elements), rep(4, n.elements), rep(5, n.elements), 

ggplot(e.binary.df, mapping = aes(x = elements, y = series, group = D, color = D)) +
    xlab("") + ylab("") +
    ggtitle("e number") +
    geom_line(position = position_dodge(0.8)) +
    theme_few(base_size = 13, base_family = "serif") +
    facet_grid(facets = D~.) +
    scale_y_continuous(breaks=c(0, 1)) +
    theme(plot.title = element_text(hjust=0.5), legend.position = "none")
```

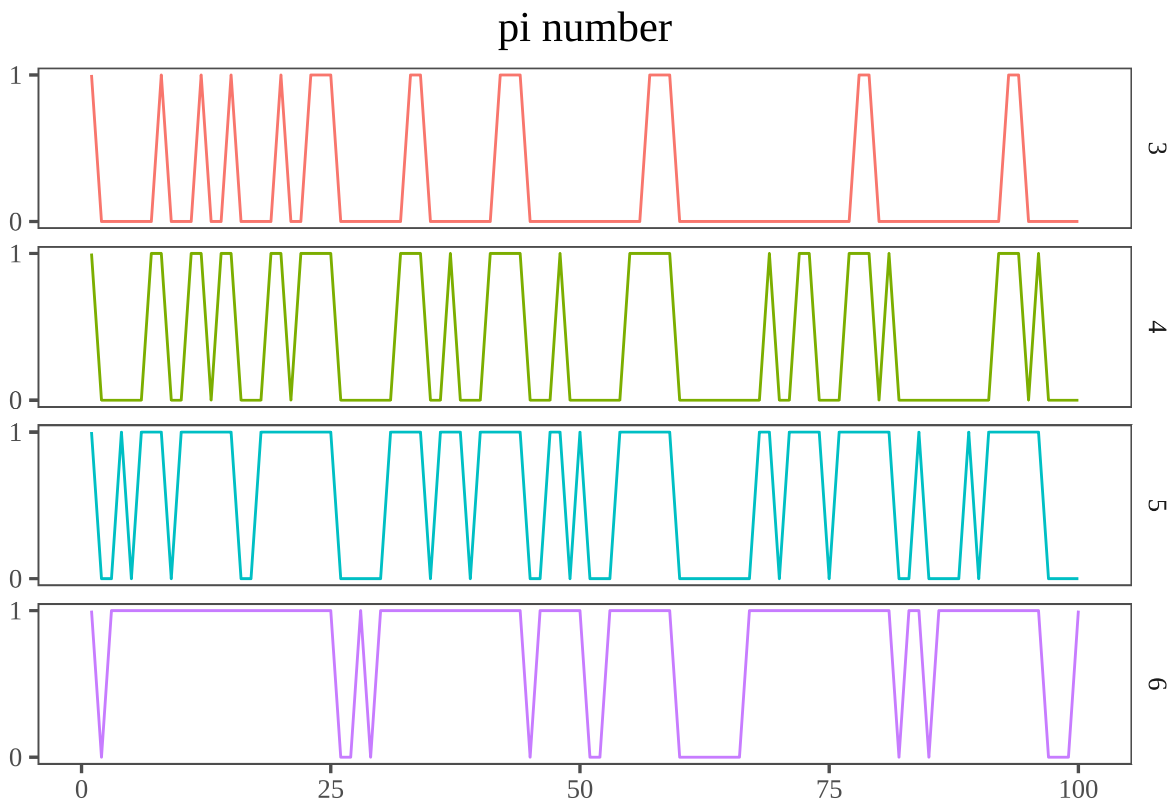



```
pi.binary.D3 = binary.equalities(pi.elements.D3)
pi.binary.D4 = binary.equalities(pi.elements.D4)
pi.binary.D5 = binary.equalities(pi.elements.D5)
pi.binary.D6 = binary.equalities(pi.elements.D6)

n.elements = 100

pi.binary.df = data.frame('series' = c(pi.binary.D3[1:n.elements], pi.binary.D4[1:n.elements], pi.binary
                         'elements' = rep(c(1:n.elements), 4),
                         'D' = as.factor(c(rep(3, n.elements), rep(4, n.elements), rep(5, n.elements), r

ggplot(pi.binary.df, mapping = aes(x = elements, y = series, group = D, color = D)) +
    xlab("") + ylab("") +
    ggtitle("pi number") +
    geom_line(position = position_dodge(0.8)) +
    theme_few(base_size = 13, base_family = "serif") +
    facet_grid(facets = D~.) +
    scale_y_continuous(breaks=c(0, 1)) +
    theme(plot.title = element_text(hjust=0.5), legend.position = "none")
```

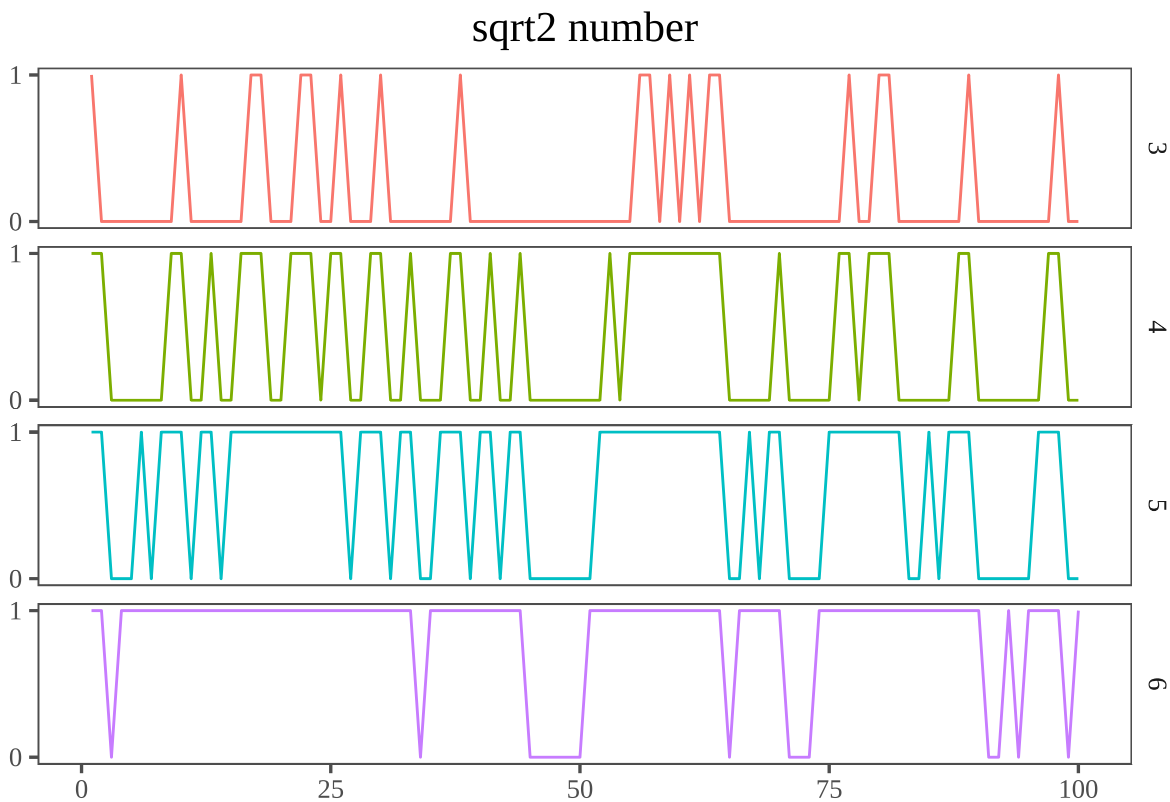



```
sqrt2.binary.D3 = binary.equalities(sqrt2.elements.D3)
sqrt2.binary.D4 = binary.equalities(sqrt2.elements.D4)
sqrt2.binary.D5 = binary.equalities(sqrt2.elements.D5)
sqrt2.binary.D6 = binary.equalities(sqrt2.elements.D6)

n.elements = 100

sqrt2.binary.df = data.frame('series' = c(sqrt2.binary.D3[1:n.elements], sqrt2.binary.D4[1:n.elements],
                         'elements' = rep(c(1:n.elements), 4),
                         'D' = as.factor(c(rep(3, n.elements), rep(4, n.elements), rep(5, n.elements),

ggplot(sqrt2.binary.df, mapping = aes(x = elements, y = series, group = D, color = D)) +
    xlab("") + ylab("") +
    ggtitle("sqrt2 number") +
    geom_line(position = position_dodge(0.8)) +
    theme_few(base_size = 13, base_family = "serif") +
    facet_grid(facets = D~.) +
    scale_y_continuous(breaks=c(0, 1)) +
    theme(plot.title = element_text(hjust=0.5), legend.position = "none")
```



As we can see in the graphs above, the larger the dimension used, the greater the presence of patterns with repeated elements. When we have $D = 3$, we see that there is a greater tendency for the existence of sequential patterns with the $label = 0$ (that is, without repeated elements). However, as the dimension increases, this behavior is reversed. This fact occurs because we are analyzing numbers within the small range $[0, 9]$, so the larger the symbol considered, the greater the probability of the existence of equal elements to be grouped.

Examples of tied patterns

For illustration, follow we have presented some examples of tied sequences. How we can observe, the implementation used for Bandt-Pompe symbolization considers the same approach of the time-ordered algorithm.

```
sqrt2.patterns.D3 = formationPattern(sqrt2.vector, 3, 1, 0)
sqrt2.binary.D3 = binary.equalities(sqrt2.elements.D3)
print('Elements:')
```

```
## [1] "Elements:"
```

```
print(sqrt2.elements.D3[which(sqrt2.binary.D3 == 1)[1:10],])
```

```
##       [,1] [,2] [,3]
##  [1,]    4    1    4
##  [2,]    3    7    3
##  [3,]    4    8    8
##  [4,]    8    8    0
##  [5,]    6    8    8
##  [6,]    8    8    7
```

```
##  [7,]    2    4    2
##  [8,]    9    6    9
##  [9,]    6    9    6
## [10,]    7    6    6
```

```r
print('Patterns:')
```

```
## [1] "Patterns:"
```

```r
print(sqrt2.patterns.D3[which(sqrt2.binary.D3 == 1)[1:10],])
```

```
##       [,1] [,2] [,3]
##  [1,]    1    0    2
##  [2,]    0    2    1
##  [3,]    0    1    2
##  [4,]    2    0    1
##  [5,]    0    1    2
##  [6,]    2    0    1
##  [7,]    0    2    1
##  [8,]    1    0    2
##  [9,]    0    2    1
## [10,]    1    2    0
```

```r
e.patterns.D4 = formationPattern(e.vector, 4, 1, 0)
e.binary.D4 = binary.equalities(e.elements.D4)
print('Elements:')
```

```
## [1] "Elements:"
```

```r
print(e.elements.D4[which(e.binary.D4 == 1)[1:10],])
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    1    8    2    8
##  [2,]    8    2    8    1
##  [3,]    2    8    1    8
##  [4,]    8    1    8    2
##  [5,]    1    8    2    8
##  [6,]    8    2    8    4
##  [7,]    5    2    3    5
##  [8,]    2    3    5    3
##  [9,]    3    5    3    6
## [10,]    8    7    4    7
```

```r
print('Patterns:')
```

```
## [1] "Patterns:"
```

```r
print(e.patterns.D4[which(e.binary.D4 == 1)[1:10],])
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]    0    2    1    3
##  [2,]    3    1    0    2
##  [3,]    2    0    1    3
##  [4,]    1    3    0    2
##  [5,]    0    2    1    3
##  [6,]    1    3    0    2
##  [7,]    1    2    0    3
##  [8,]    0    1    3    2
```

```
##  [9,]    0    2    1    3
## [10,]    2    1    3    0
```

```
pi.patterns.D6 = formationPattern(pi.vector, 6, 1, 0)
pi.binary.D6 = binary.equalities(pi.elements.D6)
print('Elements:')
```

```
## [1] "Elements:"
```

```
print(pi.elements.D6[which(pi.binary.D6 == 1)[1:10],])
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6]
##  [1,]    1    4    1    5    9    2
##  [2,]    1    5    9    2    6    5
##  [3,]    5    9    2    6    5    3
##  [4,]    9    2    6    5    3    5
##  [5,]    2    6    5    3    5    8
##  [6,]    6    5    3    5    8    9
##  [7,]    5    3    5    8    9    7
##  [8,]    3    5    8    9    7    9
##  [9,]    5    8    9    7    9    3
## [10,]    8    9    7    9    3    2
```

```
print('Patterns:')
```

```
## [1] "Patterns:"
```

```
print(pi.patterns.D6[which(pi.binary.D6 == 1)[1:10],])
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6]
##  [1,]    0    2    5    1    3    4
##  [2,]    0    3    1    5    4    2
##  [3,]    2    5    0    4    3    1
##  [4,]    1    4    3    5    2    0
##  [5,]    0    3    2    4    1    5
##  [6,]    2    1    3    0    4    5
##  [7,]    1    0    2    5    3    4
##  [8,]    0    1    4    2    3    5
##  [9,]    5    0    3    1    2    4
## [10,]    5    4    2    0    1    3
```

Monte Carlo study

The purpose of the study is measuring the ability of imputation methods (Data-driven and Time ordered imputation) to retrieve the underlying dynamic of a time series that has been *attacked*. In a loose sense, we will assess the *breakdown point* of the imputation techniques (study references for "breakdown point") (Donoho and Huber 1983; Yohai 1987). The **attack** consists in introducing randomly repeated values in the sequence.

```
# Define imputation techniques as functions
I1 <- function(time_series, params=...){return(imputed_time_series)}
I2 <- function(time_series){return(imputed_time_series)}

# Define attack
# Input:  "time_series" a time series without repetitions
#         "d" embedding dimension
#         "p" probability of attack
```

```r
AttackTimeSeries <- function(time_series, d, p) {
  attacked_time_series <- time_series
  foreach(d in time_series) { # Check the package foreach
    if (runif(1) <= p) { # attack with probability p
      select uniformly an observation in d, say at position i
      select uniformly a position in d different from i, say j
      replace d[j] <- d[i] # this introduces a repetition
    }
  }
  return(attacked_time_series)
}

# Define the parameter space of the Monte Carlo study
# My suggestion: white noise, for which we already have confidence intervals in the HxC plane

N <- c(N1, N2, N3, ...) # lengths of the time series to be considered
D <- c(D1, D2, D3, ...) # embedding dimensions to be considered
P <- c(p1, p2, p3, ...) # probabilities of attack to be considered

# Store all the points in the HxC plane of the following loop
for(n in N){
  for(d in D) {
    for(p in P) {

      x <- white_noise_length_n # perhaps from the true noise we already have
      hcx <- point_in_the_HC_plane(x)
      x_attack <- AttackTimeSeries(x, d, p)
      hcxattack <- point_in_the_HC_plane(x_attack)

    }
  }
}

# Analyze the points, for instance:
# Measure distances between pairs
# Count the number of points inside the confidence regions
```


Donoho, David L, and Peter J Huber. 1983. "The Notion of Breakdown Point." *A Festschrift for Erich L. Lehmann* 157184.

Yohai, Victor J. 1987. "High Breakdown-Point and High Efficiency Robust Estimates for Regression." *The Annals of Statistics*, 642–56.