Estimation of Tumors Grown Rate

Zijia Cao, Entong Li, Tom Sweeney

4/23/2020

Description:

A data set is given by the cancer cilent, it has 116 obersevations totally. Each observation represents the time from detection of an initial tumor to the time of detection of a recurrent tumor for one breast cancer patient. It's already known that the time `p` is generated by the following process with given parameters $\theta = (\theta_1, \theta_2, \theta_3, \theta_4)$:

```
cancer.dat <- scan("cancer data.txt")
simulate.fuc <- function(the, sample.size = 1) {
  time.p = rep(NA, sample.size)
  for (i in 1:sample.size) {
    repeat {
      u <- runif(4,0,1)
      Detect1 <- log(1-(the[1]/the[3])*log(u[1]))/the[1]
      Metastasis <- log(1-(the[1]/the[4])*log(u[2]))/the[1]
      NewSystemic <- (-log(u[3]))/the[2]
      if (Metastasis > Detect1){
        Second <- NewSystemic}else{
          Second <- min(Metastasis,NewSystemic)
        }
      Detect2 <- log(1-(the[1]/the[3])*log(u[4]))/the[1]
      p <- Second + Detect2 - Detect1
      if (p>0) {
        time.p[i] = round(p) ; break
        }
    }
  }
  return(time.p)
}
```

However, while other three θ are given, the θ_1 used to generated the given dataset now is missing. The main purpose of this report is to estimate the value of θ_1, with given $\theta_2 = 1/730$, $\theta_3 = \theta_4 = 1/2^{24}$.

Methods used to estimate θ_1:

1. Kolmogorov–Smirnov test:

Kolmogorov–Smirnov test is one of the nonparametric test used to compare a sample with given probability distribution. The null and alternative hypothesis of the test are:

$$H_0 : F = F_0 \quad vs. \quad H_1 : F \neq F_0,$$

where F is the cumulative distribution function of the given dataset, and F_0 is the given empirical cumulative distribution function.

The test statistic is the supremum distance between F and F_0, which use the formula:

$$D_n := \sqrt{n} \sup_{x \in \mathbb{R}} |F_n(x) - F_0(x)|.$$

If $F = F_0$, then the test statistic should be small, otherwise, it will reject the null hypothesis.

2. Kullback-Leibler Divergence:

A non-parametric test method to directly compare the difference between the original distribution and the generated distribution.

The reason to choose **Kullback-Leibler Divergence** is because, when people use another probability distribution to approximate the original data distribution, the information must be missing, KL divergence is used to measure this lack of information.

The formula of **Kullback-Leibler Divergence** is: $D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) log \frac{p(x_i)}{q(x_i)}$

p is the original probability dsitribution and q is the simulated probability distribution in this project.

If differeces between two distributions are very small, then the test statistic, or KL divergence, will be very small.

3. Cramer-von Mises Test:

Cramer-von mises test is one of the nonparameteric goodness-of-fit test for distribution models. The null and alternative hypothesis of the test are:

$$H_0 : F = F_0 \quad vs. \quad H_1 : F \neq F_0,$$

where F is the cumulative distribution function of the given dataset, and F_0 is the given empirical cumulative distribution function.

The test statistic is the quadratic distance between F and F_0, which use the formula:

$$W_n^2 = n \int (F_n(x) - F_0(x))^2 dF_0(x).$$

Thus, if the null hypothesis $F = F_0$ is true, then the test statistic W_n^2 should be very small, otherwise it will reject the null hypothesis.

After comparing these three methods, we find that Cramer-von Mises test has higher accuracy in predicting the θ_1 than other two methods. Thus we decide to use Cramer-von Mises test to continue the rest part of this report.

Cramer-von Mises Test:

We begin by creating a function that takes a proposed value of θ_1 and returns the Cramer-von Mises test statistic from a comparison between the cancer data and simulated data using the given θ_1.

```
the2 <- 1/730
the3 <- 1/2^24
the4 <- 1/2^24

#Function to get the statistic from Cramer-von Mises test for a specific theta1

get.M2 <- function(the1) {
  the <- c(the1, 1/730, 1/2^24,1/2^24)
  fit <- simulate.fuc(the, sample.size = 5000)
  F0 <- ecdf(fit)
  test <- cvm.test(x = cancer.dat, null = F0)
  M2 <- test$statistic
  return(M2)
}
```

Note: Higher values of the sample size parameter (`sample.size`) in `simulate.fuc()` will reduce variation caused by the randomness in the simulation. Ideally, the empirical distribution of the simulated data would be exactly the same with each iteration of `get.M2()` as long as the value of `the1` does not change. However, the simulation randomness makes iterative usage of `get.M2()` noisy. We set `sample.size` equal to 5000, but slower machines may face long computation times with high values.

Narrowing down the range of possible θ_1 values

We don't know the range of possible θ_1 values. We are told that θ_1 is the exponential rate at which a tumor grows, starting from a single cell. Therefore, we know that θ_1 will be greater than zero, but we don't know what a reasonable upper bound would be.

Our goal now is to choose an upper bound for $theta_1$.

Let the initial size for tumor j be equal to one cell. Then, the unit size of tumor j at time t is:

$$Y_j(t) = e^{\theta_1 t}$$

The data provided to us are the observed times from detection of an initial tumor to the time of detection of a recurrent tumor. Therefore, we can get a sense of the observed range of t by looking at the range of the data provided. The maximum value of the observed times is:

```
max(cancer.dat)
```

```
## [1] 2419
```

Therefore, a reasonable value of θ_1 would therefore be able to provide a reasonable value of $e^{\theta_1 \cdot 2419}$ for the number of cells in a tumor. Then we ask: How many cells can a large tumor have?

According to a Physician Data Query (2018) text published by the U.S. government's National Cancer Institute (2018), a tumor the size of a grapefruit would have a diameter of 10cm. A cube with length, width, and height of 10cm would have volume 1000cm^3. According to Del Monte (2009), a tumor reaching the size of 1cm^3 (approximately 1g wet weight) is "commonly assumed" to contain 1×10^9 cells (but most tumors of this volume actually contain far fewer cells).

Putting these facts together, a good upper estimate of the number of cells in a large tumor is $1000 \times 10^9 = 1 \times 10^{12}$, or one trillion cells. Indeed, the entire human body consists of about 40 trillion cells, according to Bianconi et al. (2013). So we can have confidence in this upper limit.

A value of θ_1 greater than or equal to 0.2 would therefore be very unreasonable:

```
proposed.theta1 <- c(0.2, 0.5, 1, 2)
exp(proposed.theta1*2419)
```

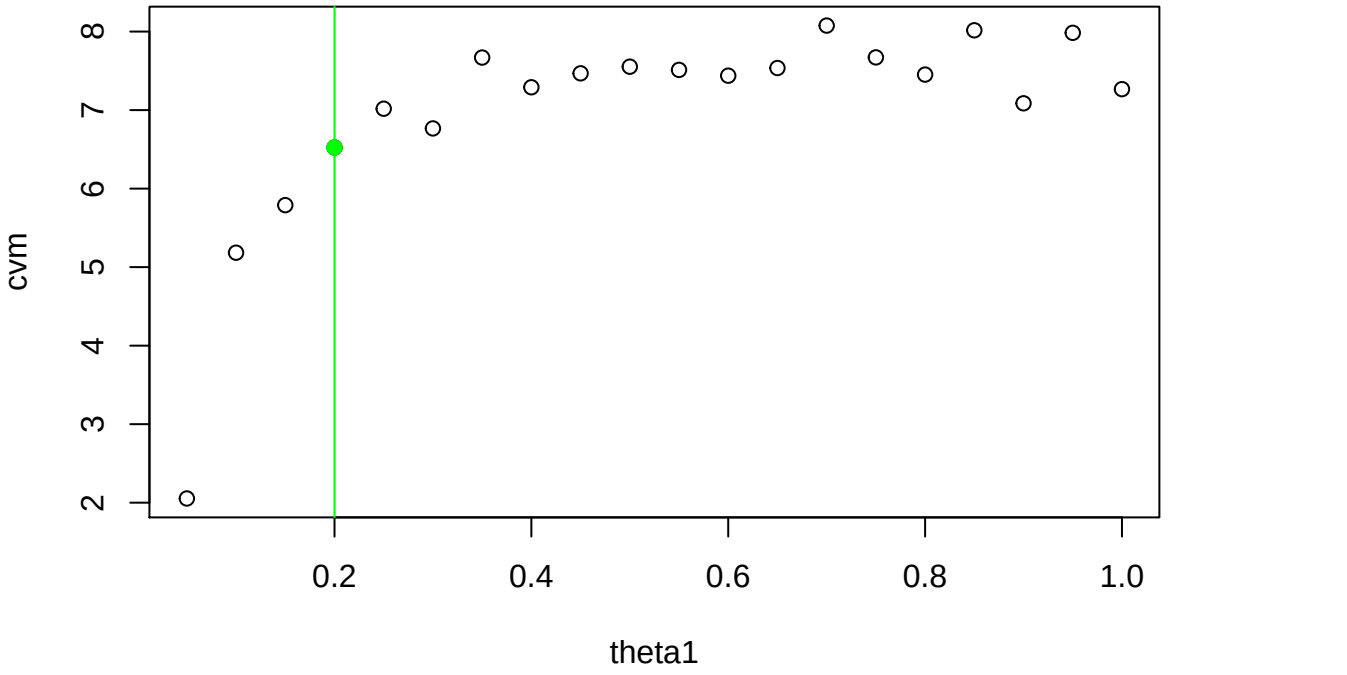
```
## [1] 1.293214e+210           Inf           Inf           Inf
```

Therefore, we restrict our attention to values of θ_1 between 0 and 0.2.

To confirm that this is reasonable for our data, we check Cramer-von Mises test statistics using our function for 20 values of θ_1 between 0 and 1. We print a plot of the CVM statistics associated with each value of θ_1. The CVM statistic associated with 0.2 is shown with a green dot:

```
theta1 <- seq(0.05,1, by = 0.05)

cvm = rep(NA, length(theta1))
for (i in 1:length(theta1)) {
  the1 <- theta1[i]
  cvm[i] <- get.M2(the1)
}
plot(x = theta1, y = cvm)
points(x = theta1[4], y = cvm[4], col = "green", pch = 19); abline(v = theta1[4], col = "green")
```



The plot shows that a value of $\theta_1 = 0.2$ clearly produces a distribution that differs from the distribution of the provided data much more than values less than 0.2. At values greater than 0.2, the CVM statistic plateaus. Therefore, we should be satisifed with an upper bound of 0.2. For the remainder of this investigation, we will restrict our search to values between zero and 0.2.

```
# Setting upper bound of theta1
upper <- 0.2
```

We will now examine four different methods for finding an optimized value of θ_1. The four methods are:

(1) Naive / simple search,
(2) R function `optimize()`,
(3) Fibonacci search, and
(4) Golden-section search.

Naive method for finding the value minimize the test statistic:

The least efficient of the four methods is the naive search method. This finds an optimal value by testing a large number of values between 0 and 0.2 using a for-loop. This method requires simulating a new distribution for each iteration of the for loop. The number of iterations is equal to the number of tested values of θ_1.

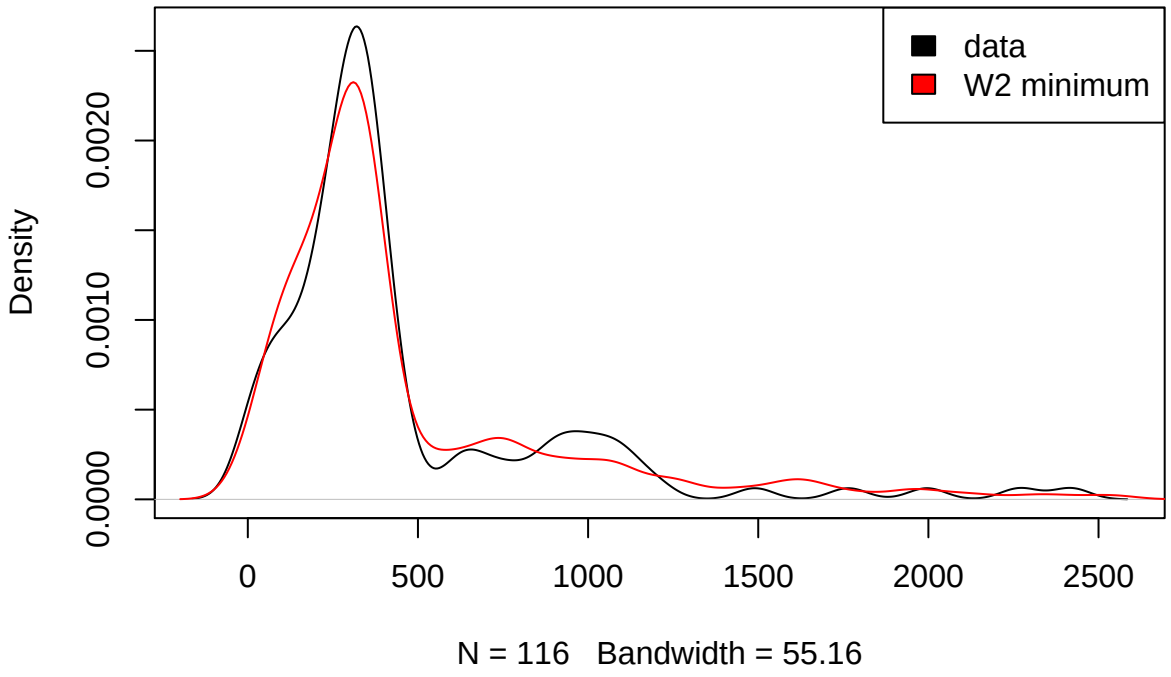
The estimated optimal value is printed below, along with a plot displaying the distribution of the given data and the distribution of the optimal simulated data. We also print the time taken to run the optimization code.

```
# setting parameters
leng <- 500
nums <- seq(0.001, upper, length.out = leng)

# method implementation
start.time <- Sys.time()
cvm = rep(NA, length(nums))
for (i in 1:length(nums)) {
  the1 <- nums[i]
  cvm[i] <- get.M2(the1)
}
mini <- nums[which.min(cvm)]
time.taken <- Sys.time() - start.time
mini
```

```
## [1] 0.03529659
```

```
# comparing distributions
simu1 <- simulate.fuc(c(mini,the2, the3, the4),leng)
plot(density(cancer.dat), main = "Naive / simple search")
lines(density(simu1), col = "red")
legend("topright",c("data","W2 minimum"),fill = c("black","red"))
```



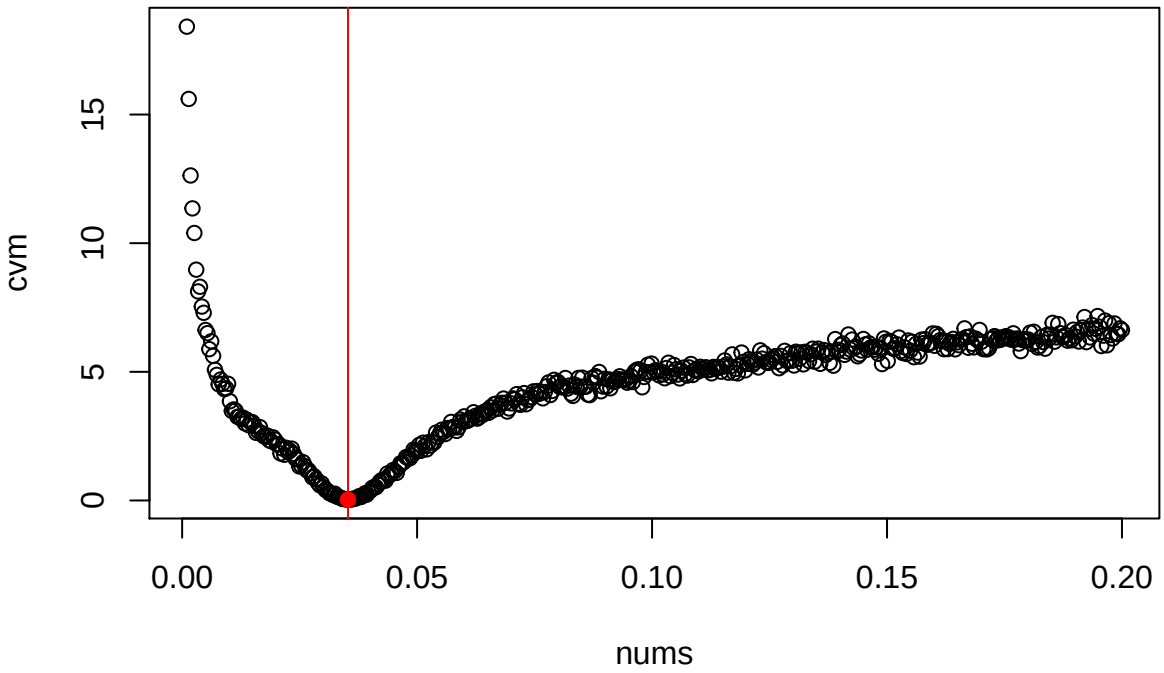

```
# computation time
time.taken
```

```
## Time difference of 8.189365 secs
```

This method required a computation time of: 8.1893649.

Now that we've collected a simulated distribution for each proposed value of θ_1, let's look at all the CVM statistics. The CVM statistic associated with the optimal value of θ_1 is shown with a red dot:

```
plot(x = nums, y = cvm)
points(x = mini, y = min(cvm), col = "red", pch = 19); abline(v = mini, col = "red")
```



We see that the optimal value sits in a relatively deep basin, and there are no other basins. There are no othe extrema or anything close to another extremum. We should have high confidence in our estimate.

1D-optimize:

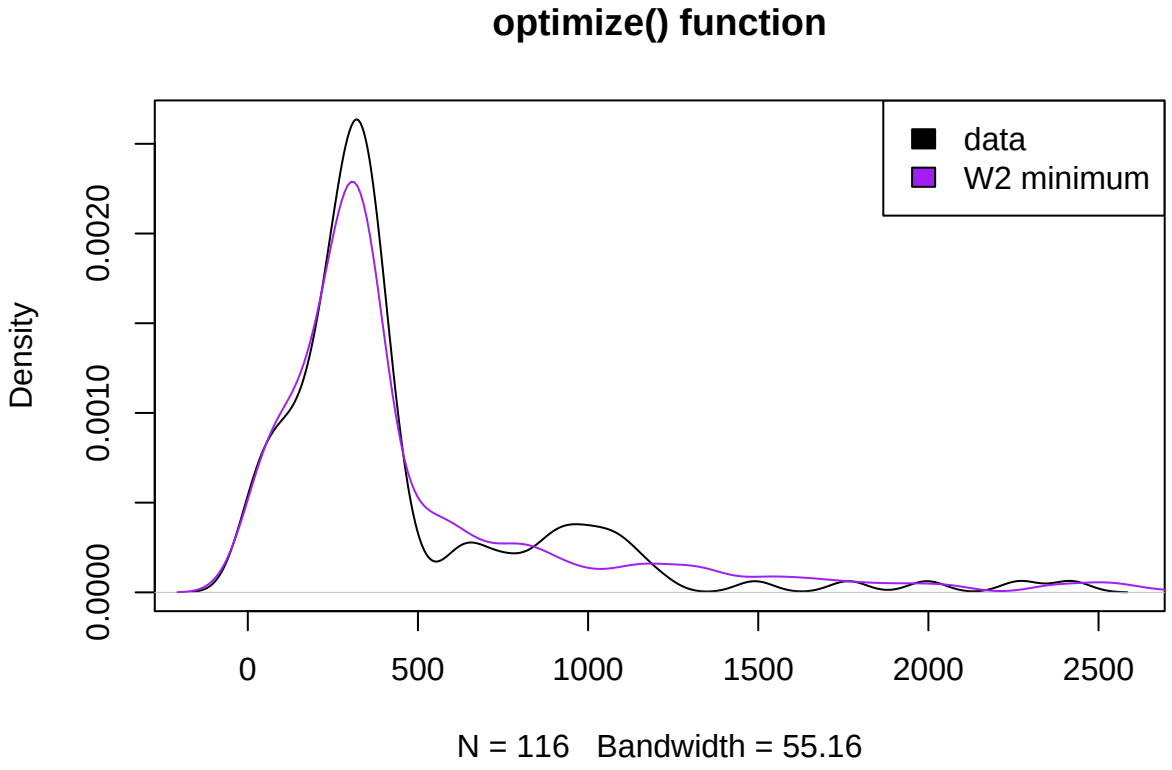
The defualt function in `R` used to find the minimum or maximum of the given function in the ginve interval, but the accuracy of this function is not high. Although this method has more noise than the rest of two methods, it could provide a good first guess of θ_1, or use its output to shrink the inverval used to search for the optimize value. For example, here the `optimize` function will return a value close to the estimated value with the naive search method.

The estimated optimal value is printed below, along with a plot displaying the distribution of the given data and the distribution of the optimal simulated data. We also print the time taken to run the optimization code.

```
# method implementation
start.time <- Sys.time()
opt.1d <- optimize(get.M2, c(0, upper), tol = 10^(-5))
time.taken <- Sys.time() - start.time
mini <- opt.1d$minimum
mini
```

```
## [1] 0.03587017
```

```
# comparing distributions
simu2 <- simulate.fuc(c(mini,the2, the3, the4),leng)
plot(density(cancer.dat), main = "optimize() function")
lines(density(simu2), col = "purple")
legend("topright",c("data","W2 minimum"),fill = c("black","purple"))
```




```
# computation time
time.taken
```

```
## Time difference of 0.3015852 secs
```

This method required a computation time of: 0.3015852.

Fibonacci search:

Fibonacci search is an optimal search method with the least number of function estimates. The reason is that Fibonacci search divides the array into unequal parts and operations involved in this search are addition and subtraction only, which means it reduces the work load of the computing machine.

The formula for Fibonacci search:

$$F[k] = F[k-1] + F[k-2]$$

$$F[k] - 1 = (F[k-1] - 1) + 1 + (F[k-2] - 1)$$

1. key = mid, Success 2. key < mid, the new search interval [low, F[k-1]-1] 3. key > mid, the new search interval [F[k-2]-1, high]

The difference between bisection and Fibonacci is that bisection take the midpoint of two numbers, the lower part and the upper part are even while Fibonacci has uneven lower part and upper part, based on the Fibonacci sequence. This saves computation time by avoiding calculation of midpoints.

We use the function `fibsearch` in library `pramca`.

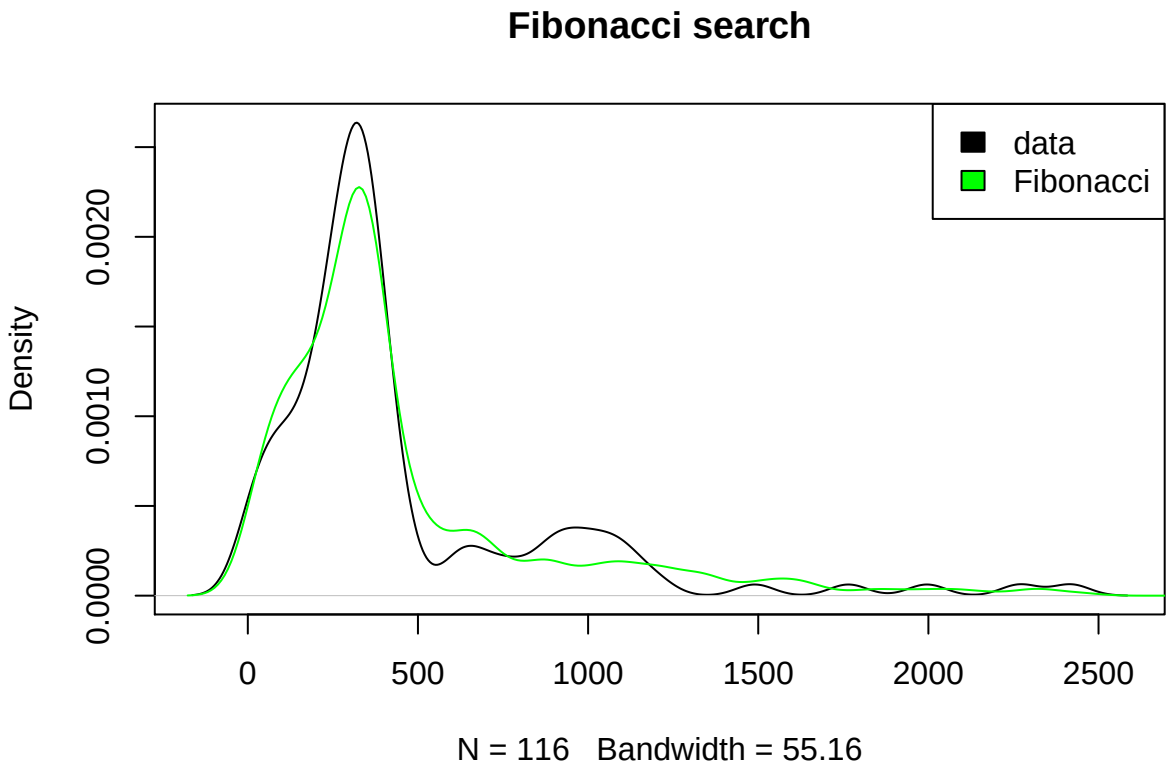
The estimated optimal value is printed below, along with a plot displaying the distribution of the given dataand the distribution of the optimal simulated data. We also print the time taken to run the optimization code.

```
# method implementation
start.time <- Sys.time()
```

```
f.search <- fibsearch(get.M2, 0, upper, endp = FALSE, tol=10^(-5))
min <- f.search$xmin
time.taken <- Sys.time() - start.time
min
```

```
## [1] 0.03468125
```

```
# comparing distributions
simu3 <- simulate.fuc(c(min, the2, the3, the4), leng)
plot(density(cancer.dat), main = "Fibonacci search")
lines(density(simu3), col = "green")
legend("topright",c("data","Fibonacci"), fill = c("black","green"))
```




```
# computation time
time.taken
```

```
## Time difference of 0.3732219 secs
```

This method required a computation time of: 0.3732219.

Golden-section search:

Golden-section search is a search method to find the maximum or minimum of a unimodal function by narrowing down the range of values, the range is determined by Golden ratio. The difference between bisection and golden-section is bisection takes the midpoint of two numbers on each iteration, and the golden-section take the golden section of two numbers. This saves computation time by avoiding calculation of midpoints.

The golden-section method is related to Fibonacci search because the golden ratio is the limit of ratio of successive terms in the Fibonacci sequence.

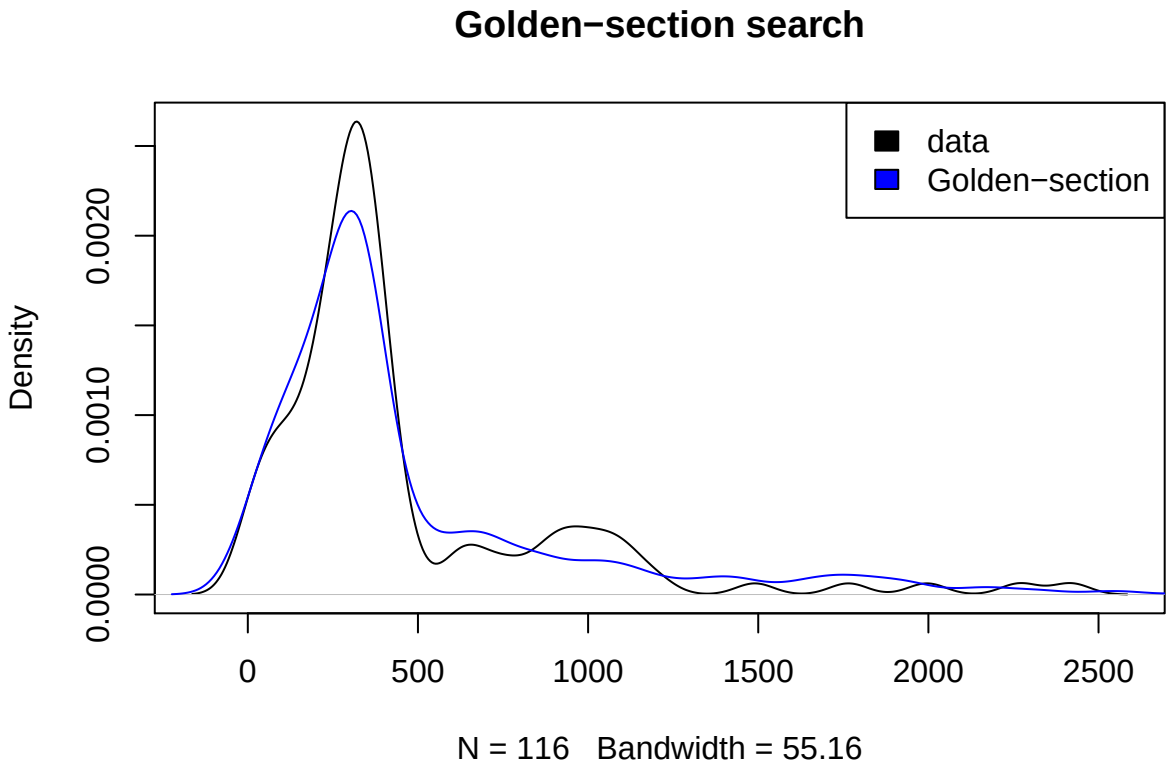
We use the function `golden_ratio` in library `pramca`.

The estimated optimal value is printed below, along with a plot displaying the distribution of the given dataand the distribution of the optimal simulated data. We also print the time taken to run the optimization code.

```
# method implementation
start.time <- Sys.time()
g.search <- golden_ratio(get.M2, 0, upper, maxiter = 500, tol =10^(-5))
min <- g.search$xmin
time.taken <- Sys.time() - start.time
min
```

```
## [1] 0.03554156
```

```
# comparing distributions
simu4 <- simulate.fuc(c(min, the2, the3, the4), leng)
plot(density(cancer.dat), main = "Golden-section search")
lines(density(simu4), col = "blue")
legend("topright",c("data","Golden-section"), fill = c("black","blue"))
```




```
# computation time
time.taken
```

```
## Time difference of 0.3590569 secs
```

This method required a computation time of: 0.3590569.

Conclusion

We estimate that the value of θ_1 for the given data is 0.035. For efficient estimation, we recommend using our function `get.M2()` along with any of the following methods: R function `optimize()`, Fibonacci search, or Golden-section search. All three methods were significantly faster than a naive search method.