Closed Form Solution to Part 2

Formula

To get the probability that there are no compromises, we compute the probability that there is at least one compromise between the first two suicides. Denote it as P, and we have the formula that

$$
\begin{aligned}
P = & \left(\frac{1}{0!}\sum_{k=0}^{\infty} q^k\right) P_1 \\
& + \left(\frac{1}{1!}\sum_{k=0}^{\infty} \frac{(k+1)!}{k!} q^k\right) P_2 \\
& + \left(\frac{1}{2!}\sum_{k=0}^{\infty} \frac{(k+2)!}{k!} q^k\right) P_3 \\
& + \cdots + \left(\frac{1}{11!}\sum_{k=0}^{\infty} \frac{(k+11)!}{k!} q^k\right) P_{12}
\end{aligned}
$$

where q is the probability of drawing a $C_j = 0$ from the empirical distribution, C_n^r denotes the combinatorial number (i.e. the number of distinct ways to choose r terms from n terms), and we will define P_n in the derivation part.

Derivation

We divide P into different situations by the number of compromises to reach "at least one compromise". For each number of compromises, we further distinguish the case with some $C_j = 0$ and the case with all $C_j \neq 0$. Denote the corresponding probability of the later cases as P_1 to P_{12}. We will show that these 12 terms have indeed exhausted all cases with nonzero C_j.

Case 1: One compromise, and we already have one compromise between the first two suicides. Note C_j cannot be 0, so the probability for one term reaching at least one compromises is just P_1.

$$P_1 = Pr(S_1 < C_1 < S_1 + S_2)$$

Case 2: Two compromises. Get P_2 by

$$P_2 = Pr(C_1 \leq S_1 < C_1 + C_2 < S_1 + S_2)$$

Then the probability for two terms reaching at least one compromise is:

$$P_2 + qP_1C_1^1$$

That is, we either have two nonzero terms, or one zero and one term from case 1. Note 0 can be anywhere in the first 1 term, so we use C_1^k to choose its place.

Case 3: Similarly, P_3 is

$$P_3 = Pr(C_1 + C_2 \leq S_1 < C_1 + C_2 + C_3 < S_1 + S_2)$$

and the probability for 3 terms reaching at least one compromise is:

$$P_3 + P_2qC_2^1 + P_1qC_2^2$$

Again, use combinatorial number to choose the place of 0.

Case 12: Following this argument, P_{12} is

$$P_{12} = Pr(\sum_{j=1}^{11} C_j \leq S_1 < \sum_{j=1}^{11} C_j + C_{12} < S_1 + S_2)$$

and the probability for 12 terms reaching at least one compromise is:

$$P_{12} + P_{11}qC_{11}^1 + P_{10}q^2C_{11}^2 + P_9q^3C_{11}^3 + \cdots + P_2q^{10}C_{11}^{10} + P_1q^{11}C_{11}^{11}$$

Note S_1 can only be one of $6, 10, 11$, so case 12 is the last case where each $C_j \neq 0$. For later cases, we have the following.

- Probability for 13 terms reaching at least one compromise

  $$P_{12}qC_{12}^1 + P_{11}q^2C_{12}^2 + P_{10}q^3C_{12}^3 + P_9q^4C_{12}^4 + \cdots + P_2q^{11}C_{12}^{11} + P_1q^{12}C_{12}^{12}$$

- Probability for 14 terms reaching at least one compromise

  $$P_{13}q^2C_{13}^2 + P_{12}q^3C_{13}^3 + P_{11}q^4C_{13}^4 + P_9q^5C_{13}^5 + \cdots + P_2q^{12}C_{13}^{12} + P_1q^{13}C_{13}^{13}$$

After rearrangement, the final probability of having at least one compromise between the first two suicides is

$$
\begin{aligned}
P = & P_1[q^0C_0^0 + q^1C_1^1 + q^2C_2^2 + \dots] \\
& + P_2[q^0C_1^0 + q^1C_2^1 + q^2C_3^2 + \dots] \\
& + P_3[q^0C_2^0 + q^1C_3^1 + q^2C_4^2 + \dots] \\
& + \dots \\
& + P_{11}[q^0C_{10}^0 + q^1C_{11}^1 + q^2C_{12}^2 + \dots] \\
& + P_{12}[q^0C_{11}^0 + q^1C_{12}^1 + q^2C_{13}^2 + \dots]
\end{aligned}
$$

Rewrite combinatorial numbers with factorials, we get the formula stated above.

Computation

Infinite Sum: It's straightforward to check (by ratio test) that all infinite sums involved are convergent. We have easy formula for the first two. For the rest, we use the result:

Lemma
Let $S = \sum_{k=1}^{\infty} a_k, S_n = \sum_{k=1}^{n} a_k$. Suppose $\{a_k\}$ is a positive decreasing sequence and $\lim_{k\to\infty} \frac{a_{k+1}}{a_k} = L < 1$. If $\frac{a_{k+1}}{a_k}$ decreases to limit L, then

$$
S_n + a_n \frac{L}{1-L} < S < S_n + \frac{a_{n+1}}{1 - \frac{a_{n+1}}{a_n}}
$$

With this lemma, we can approximate the infinite sum with very low error. In our case, let $n = 50$ can already control the error below 10^{-10}.

P_1 **to** P_{12} Each of P_1 to P_{12} can be computed by probability of achieving $S \in \{6, 10, 11\}$ and $C \in \{1, 5, 7, 10\}$. Since bootstrap process would not enlarge the range of S and C (if we let the appended value for the case where no C or S is larger than Δ to be the largest one of the range), this computation only needs to be done once. Nevertheless, this computation is extremely prone to mistakes, and it took me several days to verify it, including referring it to simulation results of P_n

Result

We first use the formula to estimate the target probability with given data, then we construct 10^5 bootstrap samples, and estimate the target probability with each sample to get the error, and eventually get the empirical cumulative distribution of error. R codes are attached in the end.

Scenario 1

- Estimation: 0.108148
- Bias: 0.014379
- MSE: 0.013275
- sd: 0.114319
- MAD: 0.080779

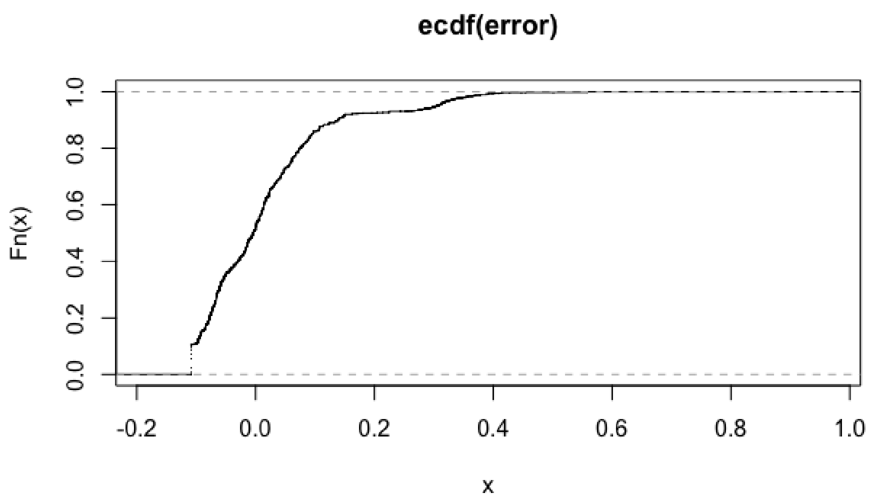

Figure 1: Scenario 1

Scenario 2

- Estimation: 0.121602
- Bias: 0.042069
- MSE: 0.022619
- sd: 0.144394
- MAD: 0.100593

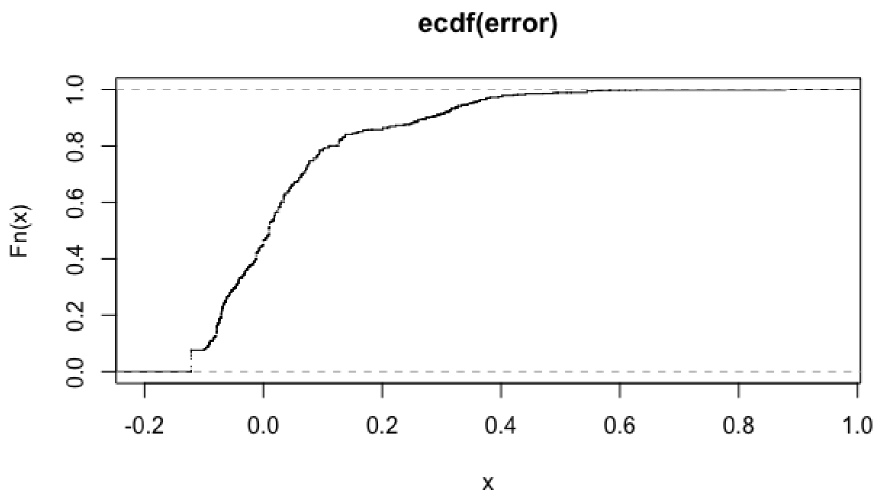

Figure 2: Scenario 2

```
#### Environment ####

# Huizhu's data
suicide <- c(6,10,11)
compromise <- c(10,5,1,1,1,0,7)
d.prob <- c(1/6, 1/6, 1/8, 1/8, 1/8, 1/8, 1/6)

#####

# Q1 by Closed Form

#### Estimation ####
# basic probabilities
q = 1/8 #prob for 0
# suicides
ps6 = 1/3
ps10 = 1/3
ps11 = 1/3
# compromises
pc1 = 1/8 * 3
pc5 = 1/6
pc7 = 1/6
pc10 = 1/6

# compute P1 - P12
{
  # P1
  P1 = ps6 * (pc7+pc10)

  # P2
  P2 = ps6^2 * (pc1*(pc7+pc10) + pc5*pc5) +
    ps6*ps10 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc1+pc5)) +
    ps10^2 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc1+pc5+pc7))
+
    ps10*ps11 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) +
pc10*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc5)) +
    ps11*ps10 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10)) +
    ps11^2 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10))

  # P3
  P3 = ps6^2 * (pc1^2*(pc5+pc7) + pc1*pc5*2*(pc1+pc5)) +
    ps6*ps10 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7) + pc1*pc7*2*(pc5+pc7) +
pc5^2*(pc1+pc5)) +
    ps10^2 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7) +
pc1*pc10*2*(pc1+pc5) + pc5^2*(pc5)) +
    ps11*ps10 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7) + pc5^2*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^2 *pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7+pc10) + pc5^2*(pc5+pc7+pc10))

  # P4
  P4 = ps6^2 * (pc1^3*(pc5+pc7)) +
    ps6*ps10 * (pc1^3*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^3*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1^2*pc7*3*(pc5)) +
    ps10^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))
+
    ps10*ps11 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1*pc5^2*3*(pc1+pc5) +
pc1^2*pc7*3*(pc5+pc7)) +
    ps11*ps10 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
```

```
pc1*pc5^2*3*(pc1+pc5+pc7) + pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))

  # P5
  P5 = ps6^2 * (pc1^4*(pc5+pc7)) +
    ps6*ps10 * (pc1^4*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^4*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc1+pc5)) +
    ps10^2 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc5)) +
    ps11*ps10 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10))

  # P6
  P6 = ps6^2 * (pc1^5*(pc5)) +
    ps6*ps10 * (pc1^5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5)) +
    ps10^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7) + pc1^4*pc7*5*(pc1+pc5)) +
    ps11*ps10 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7+pc10))

  # P7
  P7 = ps6^2 * (pc1^6*(pc1+pc5)) +
    ps6*ps10 * (pc1^6*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^6*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^6*(pc5+pc7) + pc1^5*pc5*6*(pc1+pc5)) +
    ps10^2 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5)) +
    ps11*ps10 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10))

  # P8
  P8 = ps10*ps6 * (pc1^7*(pc5+pc7)) +
    ps10^2 * (pc1^7*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^7*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^7*(pc5+pc7) + pc1^6*pc5*7*(pc1+pc5)) +
    ps11*ps10 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7+pc10))

  # P9
  P9 = ps10*ps6 * (pc1^8*(pc5+pc7)) +
    ps10^2 * (pc1^8*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^8*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^8*(pc5+pc7)) +
    ps11*ps10 * (pc1^8*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^8*(pc5+pc7+pc10))

  # P10
  P10 = ps10*ps6 * (pc1^9*(pc5)) +
    ps10^2 * (pc1^9*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^9*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^9*(pc5+pc7)) +
    ps11*ps10 * (pc1^9*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^9*(pc5+pc7+pc10))

  # P11
  P11 = ps10*ps6 * (pc1^10*(pc1+pc5)) +
    ps10^2 * (pc1^10*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^10*(pc1+pc5+pc7+pc10)) +
```

```
    ps11*ps6 * (pc1^10*(pc5)) +
    ps11*ps10 * (pc1^10*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^10*(pc5+pc7+pc10))

  # P12
  P12 = ps11*ps6 * (pc1^11*(pc1+pc5)) +
    ps11*ps10 * (pc1^11*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^11*(pc1+pc5+pc7+pc10))
}
PP <- c(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)
remove(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)

# Fine-tuning part (find appropriate n)
{
  potential_n <- c(50, 100, 500, 1000, 5000, 10^4, 1.5*10^4)
  # start while loop
  epsilon = 100
  i = 1
  # control the error within 10^(-10)
  # use largest t = 11 get the bound
  while(epsilon >= 10^(-10)){
    n = potential_n[i]
    coeff = prod(seq(from = n+1, to = n+11, by = 1)) # product of the vector
    epsilon = coeff * q^n * ( q/((n+1)/(11+n+1) - q) - q/(1-q) ) # error of sum
    i = i + 1
  }
  remove(i, coeff, epsilon, potential_n)
}

# use partial sum to approximate infinite sum
{
  i <- 1:n
  PPC <- vector()
  # P1 coefficient (P1C)
  PPC <- append(PPC, 1/(1-q))
  # P2
  PPC <- append(PPC, q/(1-q)^2 + 1/(1-q) )
  # P3 - P12 (by approximation)
  PPT <- sapply(seq(from = 2, to = 11, by = 1), function(t){
    # factorial part
    coeff <- sapply(i, function(k){
      prod(seq(from = k+1, to = k+t, by = 1))
    })
    # q^k part
    product <- rep(q, n)^i
    # combine
    return(1 + 1/factorial(t) * ( sum(coeff * product) +
    prod(seq(from = n+1, to = n+t, by = 1)) * q^n * q/(1-q)) )
  })
  PPC <- append(PPC, PPT)
  remove(n, PPT, i)
}
remove(q, ps6, ps10, ps11, pc1, pc5, pc7, pc10)

# get probability: compromise between first and second suicides
pr_target <- 1 - sum(PP*PPC)
pr_target_ss <- 1 - sum(PPS*PPC)


#####

#### Bootstrap ####

# get bootstrap cdf of estimation error
# that is, first construct 10^5 bootstrap samples,
BB = 10^5
# list containing bootstrap samples
SS_data <- list()
CC_data <- list()
# construct 10^5 bootstrap samples
set.seed(1019)
```

```
for (j in 1:BB){
  # construct three suicides
  SS <- sample(suicide, size = 3, replace = T, prob = NULL)
  # construct compromises
  i = 0
  CC <- vector()
  while (i <= sum(SS)){
    CC <- append(CC, sample(compromise, size = 1, replace = T, prob = d.prob))
    i <- sum(CC)
  }
  # append
  SS_data[[j]] <- SS
  CC_data[[j]] <- CC[1:length(CC)-1]
}
remove(CC, SS, i, j)


## get the target probability from each of these sample
# vector containing errors
error <- vector()
for (u in 1:BB){
# prepare probability for compromises
delta <- sum(SS_data[[u]]) - sum(CC_data[[u]])
diff <- CC_data[[u]] - delta
Nplus <- sum(diff > 0)
# how many terms of C are strictly larger than delta?
# with this, get probabilities
lowprob <- 1/(length(CC_data[[u]]) + 1 )
# there is a case, that we don't get any C larger than delta
# under this case, Nplus is 0, highprob is infinity
highprob <- 1/(length(CC_data[[u]]) + 1 ) * (1 + 1/Nplus )
dd.prob <- rep(lowprob, times = length(CC_data[[u]]))
# but no term satisfies diff > 0 if Nplus is 0, so every term is (1/N+1)
dd.prob[which(diff > 0)] <- highprob

# because previous calculation is carried with 0,1,5,7,10 in mind
# we make the arbitrary term to be 10 for the case
# "no term has diff > 0"
append_value = 10
if (Nplus > 0){
  CC_update <- CC_data[[u]]
  dd.prob_update <- dd.prob
} else if (Nplus == 0) {
  CC_update <- append(CC_data[[u]], append_value)
  dd.prob_update <- append(dd.prob, lowprob)
  # add low prob s.t. total probability is 1
} else {
  print("error Nplus")
}



# prepare probability for suicides
Lowprob <- 1/(length(SS_data[[u]]))
ss.prob <- rep(Lowprob, times = length(SS_data[[u]]))



# basic probabilities
q = sum(dd.prob_update[which(CC_data[[u]] == 0)]) #prob for 0
# suicides
ps6 = sum(ss.prob[which(SS_data[[u]] == 6)])
ps10 = sum(ss.prob[which(SS_data[[u]] == 10)])
ps11 = sum(ss.prob[which(SS_data[[u]] == 11)])
# compromises
pc1 = sum(dd.prob_update[which(CC_data[[u]] == 1)])
pc5 = sum(dd.prob_update[which(CC_data[[u]] == 5)])
pc7 = sum(dd.prob_update[which(CC_data[[u]] == 7)])
pc10 = sum(dd.prob_update[which(CC_data[[u]] == 10)])

# compute P1 - P12
```

```
{
  # P1
  P1 = ps6 * (pc7+pc10)

  # P2
  P2 = ps6^2 * (pc1*(pc7+pc10) + pc5*pc5) +
    ps6*ps10 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc1+pc5)) +
    ps10^2 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc1+pc5+pc7))
+
    ps10*ps11 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) +
pc10*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc5)) +
    ps11*ps10 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10)) +
    ps11^2 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10))

  # P3
  P3 = ps6^2 * (pc1^2*(pc5+pc7) + pc1*pc5*2*(pc1+pc5)) +
    ps6*ps10 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7) + pc1*pc7*2*(pc5+pc7) +
pc5^2*(pc1+pc5)) +
    ps10^2 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7) +
pc1*pc10*2*(pc1+pc5) + pc5^2*(pc5)) +
    ps11*ps10 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7) + pc5^2*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^2 *pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7+pc10) + pc5^2*(pc5+pc7+pc10))

  # P4
  P4 = ps6^2 * (pc1^3*(pc5+pc7)) +
    ps6*ps10 * (pc1^3*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^3*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1^2*pc7*3*(pc5)) +
    ps10^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))
+
    ps10*ps11 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1*pc5^2*3*(pc1+pc5) +
pc1^2*pc7*3*(pc5+pc7)) +
    ps11*ps10 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7) + pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))

  # P5
  P5 = ps6^2 * (pc1^4*(pc5+pc7)) +
    ps6*ps10 * (pc1^4*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^4*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc1+pc5)) +
    ps10^2 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc5)) +
    ps11*ps10 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10))

  # P6
  P6 = ps6^2 * (pc1^5*(pc5)) +
    ps6*ps10 * (pc1^5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5)) +
    ps10^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
```

```
    ps10*ps11 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7) + pc1^4*pc7*5*(pc1+pc5)) +
    ps11*ps10 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7+pc10))

  # P7
  P7 = ps6^2 * (pc1^6*(pc1+pc5)) +
    ps6*ps10 * (pc1^6*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^6*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^6*(pc5+pc7) + pc1^5*pc5*6*(pc1+pc5)) +
    ps10^2 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5)) +
    ps11*ps10 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10))

  # P8
  P8 = ps10*ps6 * (pc1^7*(pc5+pc7)) +
    ps10^2 * (pc1^7*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^7*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^7*(pc5+pc7) + pc1^6*pc5*7*(pc1+pc5)) +
    ps11*ps10 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7+pc10))

  # P9
  P9 = ps10*ps6 * (pc1^8*(pc5+pc7)) +
    ps10^2 * (pc1^8*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^8*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^8*(pc5+pc7)) +
    ps11*ps10 * (pc1^8*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^8*(pc5+pc7+pc10))

  # P10
  P10 = ps10*ps6 * (pc1^9*(pc5)) +
    ps10^2 * (pc1^9*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^9*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^9*(pc5+pc7)) +
    ps11*ps10 * (pc1^9*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^9*(pc5+pc7+pc10))

  # P11
  P11 = ps10*ps6 * (pc1^10*(pc1+pc5)) +
    ps10^2 * (pc1^10*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^10*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^10*(pc5)) +
    ps11*ps10 * (pc1^10*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^10*(pc5+pc7+pc10))

  # P12
  P12 = ps11*ps6 * (pc1^11*(pc1+pc5)) +
    ps11*ps10 * (pc1^11*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^11*(pc1+pc5+pc7+pc10))
}
PP <- c(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)
remove(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)

# Fine-tuning part (find appropriate n)
{
  potential_n <- c(50, 100, 500, 1000, 5000, 10^4, 1.5*10^4)
  # start while loop
  epsilon = 100
  i = 1
  # control the error within 10^(-10)
  # use largest t = 11 get the bound
  while(epsilon >= 10^(-10)){
    n = potential_n[i]
    coeff = prod(seq(from = n+1, to = n+11, by = 1)) # product of the vector
    epsilon = coeff * q^n * ( q/((n+1)/(11+n+1) - q) - q/(1-q) ) # error of sum
    i = i + 1
```

```
  }
  remove(i, coeff, epsilon, potential_n)
}

# use partial sum to approximate infinite sum
{
  i <- 1:n
  PPC <- vector()
  # P1 coefficient (P1C)
  PPC <- append(PPC, 1/(1-q))
  # P2
  PPC <- append(PPC, q/(1-q)^2 + 1/(1-q) )
  # P3 - P12 (by approximation)
  PPT <- sapply(seq(from = 2, to = 11, by = 1), function(t){
    # factorial part
    coeff <- sapply(i, function(k){
      prod(seq(from = k+1, to = k+t, by = 1))
    })
    # q^k part
    product <- rep(q, n)^i
    # combine
    return(1 + 1/factorial(t) * ( sum(coeff * product) +
                                    prod(seq(from = n+1, to = n+t, by = 1)) * q^n *
q/(1-q)) )
  })
  PPC <- append(PPC, PPT)
  remove(n, PPT, i)
}
remove(q, ps6, ps10, ps11, pc1, pc5, pc7, pc10)

# get probability: compromise between first and second suicides
pr_boot <- 1 - sum(PP*PPC)
error <- append(error, pr_boot - pr_target)
remove(PP, PPC)



# Progress Indicator
if (u%%5000==0) {cat(" *",u)}
}

#####

# error distribution

saveRDS(error, "error_distribution_closeQ1.rds")
write.csv(error, file = "error_distribution_closeQ1.csv")

# plot
#error <- readRDS("error_distribution_closeQ1.rds")
CDF <- ecdf(error)
plot(CDF)

{
# Bias
mean(error)
# MSE
mean(error^2)
# sd
sd(error)
# MAD
mean(abs(error))
}
```

```
#### Environment ####

# Huizhu's data
suicide <- c(6,10,10)
s.prob <- c(1/3, 1/3, 1/3)
compromise <- c(10,5,1,1,1,0,7)
d.prob <- c(1/6, 1/6, 1/8, 1/8, 1/8, 1/8, 1/6)


#####

# Q1 by Closed Form

#### Estimation ####
# basic probabilities
q = 1/8 #prob for 0
# suicides
ps6 = 1/3
ps10 = 2/3
ps11 = 0
# compromises
pc1 = 1/8 * 3
pc5 = 1/6
pc7 = 1/6
pc10 = 1/6

# compute P1 - P12
{
  # P1
  P1 = ps6 * (pc7+pc10)

  # P2
  P2 = ps6^2 * (pc1*(pc7+pc10) + pc5*pc5) +
    ps6*ps10 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc1+pc5)) +
    ps10^2 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc1+pc5+pc7))
+
    ps10*ps11 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) +
pc10*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc5)) +
    ps11*ps10 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10)) +
    ps11^2 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10))

  # P3
  P3 = ps6^2 * (pc1^2*(pc5+pc7) + pc1*pc5*2*(pc1+pc5)) +
    ps6*ps10 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7) + pc1*pc7*2*(pc5+pc7) +
pc5^2*(pc1+pc5)) +
    ps10^2 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7) +
pc1*pc10*2*(pc1+pc5) + pc5^2*(pc5)) +
    ps11*ps10 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7) + pc5^2*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^2 *pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7+pc10) + pc5^2*(pc5+pc7+pc10))

  # P4
  P4 = ps6^2 * (pc1^3*(pc5+pc7)) +
    ps6*ps10 * (pc1^3*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^3*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1^2*pc7*3*(pc5)) +
    ps10^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))
+
    ps10*ps11 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1*pc5^2*3*(pc1+pc5) +
```

```
pc1^2*pc7*3*(pc5+pc7)) +
    ps11*ps10 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7) + pc1^2*pc7*3*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))

  # P5
  P5 = ps6^2 * (pc1^4*(pc5+pc7)) +
    ps6*ps10 * (pc1^4*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^4*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc1+pc5)) +
    ps10^2 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc5)) +
    ps11*ps10 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10))

  # P6
  P6 = ps6^2 * (pc1^5*(pc5)) +
    ps6*ps10 * (pc1^5*(pc5+pc7+pc10)) +
    ps6*ps11 * (pc1^5*(pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5)) +
    ps10^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7) + pc1^4*pc7*5*(pc1+pc5)) +
    ps11*ps10 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7+pc10))

  # P7
  P7 = ps6^2 * (pc1^6*(pc1+pc5)) +
    ps6*ps10 * (pc1^6*(pc1+pc5+pc7)) +
    ps6*ps11 * (pc1^6*(pc1+pc5+pc7+pc10)) +
    ps10*ps6 * (pc1^6*(pc5+pc7) + pc1^5*pc5*6*(pc1+pc5)) +
    ps10^2 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5)) +
    ps11*ps10 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10))

  # P8
  P8 = ps10*ps6 * (pc1^7*(pc5+pc7)) +
    ps10^2 * (pc1^7*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^7*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^7*(pc5+pc7) + pc1^6*pc5*7*(pc1+pc5)) +
    ps11*ps10 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7+pc10))

  # P9
  P9 = ps10*ps6 * (pc1^8*(pc5+pc7)) +
    ps10^2 * (pc1^8*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^8*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^8*(pc5+pc7)) +
    ps11*ps10 * (pc1^8*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^8*(pc5+pc7+pc10))

  # P10
  P10 = ps10*ps6 * (pc1^9*(pc5)) +
    ps10^2 * (pc1^9*(pc5+pc7+pc10)) +
    ps10*ps11 * (pc1^9*(pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^9*(pc5+pc7)) +
    ps11*ps10 * (pc1^9*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^9*(pc5+pc7+pc10))

  # P11
  P11 = ps10*ps6 * (pc1^10*(pc1+pc5)) +
```

```
    ps10^2 * (pc1^10*(pc1+pc5+pc7)) +
    ps10*ps11 * (pc1^10*(pc1+pc5+pc7+pc10)) +
    ps11*ps6 * (pc1^10*(pc5)) +
    ps11*ps10 * (pc1^10*(pc5+pc7+pc10)) +
    ps11^2 * (pc1^10*(pc5+pc7+pc10))

  # P12
  P12 = ps11*ps6 * (pc1^11*(pc1+pc5)) +
    ps11*ps10 * (pc1^11*(pc1+pc5+pc7)) +
    ps11^2 * (pc1^11*(pc1+pc5+pc7+pc10))
}
PP <- c(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)
remove(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)

# Fine-tuning part (find appropriate n)
{
  potential_n <- c(50, 100, 500, 1000, 5000, 10^4, 1.5*10^4)
  # start while loop
  epsilon = 100
  i = 1
  # control the error within 10^(-10)
  # use largest t = 11 get the bound
  while(epsilon >= 10^(-10)){
    n = potential_n[i]
    coeff = prod(seq(from = n+1, to = n+11, by = 1)) # product of the vector
    epsilon = coeff * q^n * ( q/((n+1)/(11+n+1) - q) - q/(1-q) ) # error of sum
    i = i + 1
  }
  remove(i, coeff, epsilon, potential_n)
}

# use partial sum to approximate infinite sum
{
  i <- 1:n
  PPC <- vector()
  # P1 coefficient (P1C)
  PPC <- append(PPC, 1/(1-q))
  # P2
  PPC <- append(PPC, q/(1-q)^2 + 1/(1-q) )
  # P3 - P12 (by approximation)
  PPT <- sapply(seq(from = 2, to = 11, by = 1), function(t){
    # factorial part
    coeff <- sapply(i, function(k){
      prod(seq(from = k+1, to = k+t, by = 1))
    })
    # q^k part
    product <- rep(q, n)^i
    # combine
    return(1 + 1/factorial(t) * ( sum(coeff * product) +
                                    prod(seq(from = n+1, to = n+t, by = 1)) * q^n *
q/(1-q)) )
  })
  PPC <- append(PPC, PPT)
  remove(n, PPT, i)
}
remove(q, ps6, ps10, ps11, pc1, pc5, pc7, pc10)

# get probability: compromise between first and second suicides
pr_target <- 1 - sum(PP*PPC)


#####

#### Bootstrap ####

# get bootstrap cdf of estimation error
# that is, first construct 10^5 bootstrap samples,
BB = 10^5
# list containing bootstrap samples
SS_data <- list()
CC_data <- list()
```

```
# construct 10^5 bootstrap samples
set.seed(1019)
for (j in 1:BB){
  # construct three suicides
  SS <- sample(suicide, size = 3, replace = T, prob = NULL)
  # construct compromises
  i = 0
  CC <- vector()
  while (i <= sum(SS)){
    CC <- append(CC, sample(compromise, size = 1, replace = T, prob = d.prob))
    i <- sum(CC)
  }
  # append
  SS_data[[j]] <- SS
  CC_data[[j]] <- CC[1:length(CC)-1]
}
remove(CC, SS, i, j)


## get the target probability from each of these sample
# vector containing errors
error <- vector()
for (u in 1:BB){
  # prepare probability for suicides
  {
  deltaS <- 26 - sum(SS_data[[u]])
  diffS <- SS_data[[u]] - deltaS
  SNplus <- sum(diffS > 0)
  # how many terms of S are strictly larger than delta?
  # with this, get probabilities
  lowprobS <- 1/(length(SS_data[[u]]) + 1 )
  # there is a case, that we don't get any C larger than delta
  # under this case, Nplus is 0, highprob is infinity
  highprobS <- 1/(length(SS_data[[u]]) + 1 ) * (1 + 1/SNplus )
  ss.prob <- rep(lowprobS, times = length(SS_data[[u]]))
  # but no term satisfies diff > 0 if Nplus is 0, so every term is (1/N+1)
  ss.prob[which(diffS > 0)] <- highprobS

  # we will append an arbitrary term.
  # To keep our closed form, let it be the largest one of the range
  append_valueS = 10
  # fix the case where no S is higher than delta
  # (each term has prob 1/(N+1) but only N terms to sample)
  if (SNplus > 0){
    SS_update <- SS_data[[u]]
    ss.prob_update <- ss.prob
  } else if (SNplus == 0) {
    SS_update <- append(SS_data[[u]], append_valueS)
    ss.prob_update <- append(ss.prob, lowprobS)
  } else {
    print("error SNplus")
  }
  }

  # prepare probability for compromises
  {
  delta <- sum(SS_data[[u]]) - sum(CC_data[[u]])
  diff <- CC_data[[u]] - delta
  Nplus <- sum(diff > 0)
  # how many terms of C are strictly larger than delta?
  # with this, get probabilities
  lowprob <- 1/(length(CC_data[[u]]) + 1 )
  # there is a case, that we don't get any C larger than delta
  # under this case, Nplus is 0, highprob is infinity
  highprob <- 1/(length(CC_data[[u]]) + 1 ) * (1 + 1/Nplus )
  dd.prob <- rep(lowprob, times = length(CC_data[[u]]))
  # but no term satisfies diff > 0 if Nplus is 0, so every term is (1/N+1)
  dd.prob[which(diff > 0)] <- highprob

  # because previous calculation is carried with 0,1,5,7,10 in mind
  # we make the arbitrary term to be 10 for the case
```

```
  # "no term has diff > 0"
  append_value = 10
  if (Nplus > 0){
    CC_update <- CC_data[[u]]
    dd.prob_update <- dd.prob
  } else if (Nplus == 0) {
    CC_update <- append(CC_data[[u]], append_value)
    dd.prob_update <- append(dd.prob, lowprob)
    # add low prob s.t. total probability is 1
  } else {
    print("error Nplus")
  }
  }

  # basic probabilities
  {
  q = sum(dd.prob_update[which(CC_data[[u]] == 0)]) #prob for 0
  # suicides
  ps6 = sum(ss.prob[which(SS_data[[u]] == 6)])
  ps10 = sum(ss.prob[which(SS_data[[u]] == 10)])
  ps11 = sum(ss.prob[which(SS_data[[u]] == 11)])
  # compromises
  pc1 = sum(dd.prob_update[which(CC_data[[u]] == 1)])
  pc5 = sum(dd.prob_update[which(CC_data[[u]] == 5)])
  pc7 = sum(dd.prob_update[which(CC_data[[u]] == 7)])
  pc10 = sum(dd.prob_update[which(CC_data[[u]] == 10)])
  }

  # compute P1 - P12
  {
    # P1
    P1 = ps6 * (pc7+pc10)

    # P2
    P2 = ps6^2 * (pc1*(pc7+pc10) + pc5*pc5) +
      ps6*ps10 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
      ps6*ps11 * (pc1*(pc7+pc10) + pc5*(pc5+pc7+pc10)) +
      ps10*ps6 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc1+pc5)) +
      ps10^2 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) +
pc10*(pc1+pc5+pc7)) +
      ps10*ps11 * (pc1*pc10 + pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) +
pc10*(pc1+pc5+pc7+pc10)) +
      ps11*ps6 * (pc5*(pc7+pc10) + pc7*(pc5+pc7) + pc10*(pc5)) +
      ps11*ps10 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10)) +
      ps11^2 * (pc5*(pc7+pc10) + pc7*(pc5+pc7+pc10) + pc10*(pc5+pc7+pc10))

    # P3
    P3 = ps6^2 * (pc1^2*(pc5+pc7) + pc1*pc5*2*(pc1+pc5)) +
      ps6*ps10 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7)) +
      ps6*ps11 * (pc1^2*(pc5+pc7+pc10) + pc1*pc5*2*(pc1+pc5+pc7+pc10)) +
      ps10*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7) + pc1*pc7*2*(pc5+pc7) +
pc5^2*(pc1+pc5)) +
      ps10^2 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc5^2*(pc1+pc5+pc7)) +
      ps10*ps11 * (pc1^2*pc10 + pc1*pc5*2*(pc5+pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10)
+ pc5^2*(pc1+pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7) +
pc1*pc10*2*(pc1+pc5) + pc5^2*(pc5)) +
      ps11*ps10 * (pc1^2*pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7) + pc5^2*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^2 *pc10 + pc1*pc5*2*(pc7+pc10) + pc1*pc7*2*(pc5+pc7+pc10) +
pc1*pc10*2*(pc1+pc5+pc7+pc10) + pc5^2*(pc5+pc7+pc10))

    # P4
    P4 = ps6^2 * (pc1^3*(pc5+pc7)) +
      ps6*ps10 * (pc1^3*(pc5+pc7+pc10)) +
      ps6*ps11 * (pc1^3*(pc5+pc7+pc10)) +
      ps10*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1^2*pc7*3*(pc5)) +
      ps10^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1^2*pc7*3*(pc5+pc7+pc10)) +
      ps10*ps11 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
```

```
pc1^2*pc7*3*(pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7) + pc1*pc5^2*3*(pc1+pc5) +
pc1^2*pc7*3*(pc5+pc7)) +
      ps11*ps10 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7) + pc1^2*pc7*3*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^3*pc10 + pc1^2*pc5*3*(pc5+pc7+pc10) +
pc1*pc5^2*3*(pc1+pc5+pc7+pc10) + pc1^2*pc7*3*(pc5+pc7+pc10))

    # P5
    P5 = ps6^2 * (pc1^4*(pc5+pc7)) +
      ps6*ps10 * (pc1^4*(pc5+pc7+pc10)) +
      ps6*ps11 * (pc1^4*(pc5+pc7+pc10)) +
      ps10*ps6 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc1+pc5))
+
      ps10^2 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7)) +
      ps10*ps11 * (pc1^4*(pc7+pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc1+pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7) + pc1^3*pc7*4*(pc5)) +
      ps11*ps10 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^4*(pc10) + pc1^3*pc5*4*(pc5+pc7+pc10) +
pc1^3*pc7*4*(pc5+pc7+pc10))

    # P6
    P6 = ps6^2 * (pc1^5*(pc5)) +
      ps6*ps10 * (pc1^5*(pc5+pc7+pc10)) +
      ps6*ps11 * (pc1^5*(pc5+pc7+pc10)) +
      ps10*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5)) +
      ps10^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
      ps10*ps11 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7) + pc1^4*pc7*5*(pc1+pc5))
+
      ps11*ps10 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7)) +
      ps11^2 * (pc1^5*(pc7+pc10) + pc1^4*pc5*5*(pc5+pc7+pc10) +
pc1^4*pc7*5*(pc1+pc5+pc7+pc10))

    # P7
    P7 = ps6^2 * (pc1^6*(pc1+pc5)) +
      ps6*ps10 * (pc1^6*(pc1+pc5+pc7)) +
      ps6*ps11 * (pc1^6*(pc1+pc5+pc7+pc10)) +
      ps10*ps6 * (pc1^6*(pc5+pc7) + pc1^5*pc5*6*(pc1+pc5)) +
      ps10^2 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7)) +
      ps10*ps11 * (pc1^6*(pc5+pc7+pc10) + pc1^5*pc5*6*(pc1+pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5)) +
      ps11*ps10 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^6*(pc7+pc10) + pc1^5*pc5*6*(pc5+pc7+pc10))

    # P8
    P8 = ps10*ps6 * (pc1^7*(pc5+pc7)) +
      ps10^2 * (pc1^7*(pc5+pc7+pc10)) +
      ps10*ps11 * (pc1^7*(pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^7*(pc5+pc7) + pc1^6*pc5*7*(pc1+pc5)) +
      ps11*ps10 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7)) +
      ps11^2 * (pc1^7*(pc5+pc7+pc10) + pc1^6*pc5*7*(pc1+pc5+pc7+pc10))

    # P9
    P9 = ps10*ps6 * (pc1^8*(pc5+pc7)) +
      ps10^2 * (pc1^8*(pc5+pc7+pc10)) +
      ps10*ps11 * (pc1^8*(pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^8*(pc5+pc7)) +
      ps11*ps10 * (pc1^8*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^8*(pc5+pc7+pc10))

    # P10
    P10 = ps10*ps6 * (pc1^9*(pc5)) +
      ps10^2 * (pc1^9*(pc5+pc7+pc10)) +
      ps10*ps11 * (pc1^9*(pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^9*(pc5+pc7)) +
      ps11*ps10 * (pc1^9*(pc5+pc7+pc10)) +
```

```
      ps11^2 * (pc1^9*(pc5+pc7+pc10))

    # P11
    P11 = ps10*ps6 * (pc1^10*(pc1+pc5)) +
      ps10^2 * (pc1^10*(pc1+pc5+pc7)) +
      ps10*ps11 * (pc1^10*(pc1+pc5+pc7+pc10)) +
      ps11*ps6 * (pc1^10*(pc5)) +
      ps11*ps10 * (pc1^10*(pc5+pc7+pc10)) +
      ps11^2 * (pc1^10*(pc5+pc7+pc10))

    # P12
    P12 = ps11*ps6 * (pc1^11*(pc1+pc5)) +
      ps11*ps10 * (pc1^11*(pc1+pc5+pc7)) +
      ps11^2 * (pc1^11*(pc1+pc5+pc7+pc10))
  }
  PP <- c(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)
  remove(P1, P2, P3, P4, P5, P6, P7, P8, P9, P10, P11, P12)

  # Fine-tuning part (find appropriate n)
  {
    potential_n <- c(50, 100, 500, 1000, 5000, 10^4, 1.5*10^4)
    # start while loop
    epsilon = 100
    i = 1
    # control the error within 10^(-10)
    # use largest t = 11 get the bound
    while(epsilon >= 10^(-10)){
      n = potential_n[i]
      coeff = prod(seq(from = n+1, to = n+11, by = 1)) # product of the vector
      epsilon = coeff * q^n * ( q/((n+1)/(11+n+1) - q) - q/(1-q) ) # error of sum
      i = i + 1
    }
    remove(i, coeff, epsilon, potential_n)
  }

  # use partial sum to approximate infinite sum
  {
    i <- 1:n
    PPC <- vector()
    # P1 coefficient (P1C)
    PPC <- append(PPC, 1/(1-q))
    # P2
    PPC <- append(PPC, q/(1-q)^2 + 1/(1-q) )
    # P3 - P12 (by approximation)
    PPT <- sapply(seq(from = 2, to = 11, by = 1), function(t){
      # factorial part
      coeff <- sapply(i, function(k){
        prod(seq(from = k+1, to = k+t, by = 1))
      })
      # q^k part
      product <- rep(q, n)^i
      # combine
      return(1 + 1/factorial(t) * ( sum(coeff * product) +
                                      prod(seq(from = n+1, to = n+t, by = 1)) * q^n
* q/(1-q)) )
    })
    PPC <- append(PPC, PPT)
    remove(n, PPT, i)
  }
  remove(q, ps6, ps10, ps11, pc1, pc5, pc7, pc10)

  # get probability: compromise between first and second suicides
  pr_boot <- 1 - sum(PP*PPC)
  error <- append(error, pr_boot - pr_target)
  remove(PP, PPC)



  # Progress Indicator
  if (u%%5000==0) {cat(" *",u)}
}
```

```
#####

# error distribution

saveRDS(error, "error_distribution_closeQ2.rds")
write.csv(error, file = "error_distribution_closeQ2.csv")

# plot
CDF <- ecdf(error)
plot(CDF)

{
  # Bias
  mean(error)
  # MSE
  mean(error^2)
  # sd
  sd(error)
  # MAD
  mean(abs(error))
}
```