Theorem 1:

We prove that for all $x$, \( \hat{\theta}(x) = 2^x - 1 \) is an estimator of $\theta$ with $n$.

This theorem states that when $n$ is chosen to be

\[
\hat{\theta}_{MLE}(x) = \text{argmax}_n f(X:n)
\]

then $x = 1, \ldots, 10$ when $n$ is chosen to be

\[
\hat{\theta}_{MSE}(X) = (1 - 2^{-x}) f(x; n - 1) + 2^{-(x-1)} f(x - 1; n - 1)
\]

We shall prove that the estimator

\[
f(x; n) = (1 - 2^{-x}) f(x; n - 1) + 2^{-(x-1)} f(x - 1; n - 1)
\]

is unbiased and consistent for $\theta$.

We shall prove that

\[
f(x > n, n) = 0
\]

and that

\[
f(x < n, n) = \frac{1}{2^{n-1}}
\]

for all $x$. We shall prove that

\[
f(x, n) = \frac{1}{2^{n-1}}
\]
\[ \Delta(x, n) = f(x; n + 1) - f(x; n) \\
= (1 - 2^{-x})(f(x; n) - f(x; n - 1)) \\
+ 2^{-x-1}(f(x - 1; n) - f(x - 1; n - 1)) \\
= (1 - 2^{-x})\Delta(x, n - 1) + 2^{-x-1}\Delta(x - 1, n - 1) \]

If the negative recursion \( (\Delta(x, n - 1) \leq 0 \text{ for all } n) \), then \( \hat{f}_{MLE} \) is a monotonic decreasing function of \( n \). If \( \Delta(x - 1, n - 1) \leq 0 \), then:

\[ \Delta(x, n) = f(x; n + 1) - f(x; n) \leq 0 \]

We found that for any given \( x \), the function \( f(x, n) \) is decreasing with respect to \( n \).

We have:

\[ \hat{f}_{MLE} \]

To find the maximum likelihood estimate at each point, we use dynamic programming. The results:

<table>
<thead>
<tr>
<th>( x )</th>
<th>( \hat{f}_{MLE} )</th>
<th>Max-Likelihood</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>0.625</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
<td>0.523670197</td>
</tr>
<tr>
<td>4</td>
<td>19</td>
<td>0.486695161</td>
</tr>
<tr>
<td>5</td>
<td>39</td>
<td>0.470369477</td>
</tr>
<tr>
<td>6</td>
<td>80</td>
<td>0.462512782</td>
</tr>
<tr>
<td>7</td>
<td>162</td>
<td>0.45869213</td>
</tr>
<tr>
<td>8</td>
<td>325</td>
<td>0.456812012</td>
</tr>
<tr>
<td>9</td>
<td>652</td>
<td>0.455876987</td>
</tr>
<tr>
<td>10</td>
<td>1306</td>
<td>0.45541105</td>
</tr>
</tbody>
</table>

The code is attached at the end of the answers (Appendix A) - consult the supplementary files for the complete code.
<table>
<thead>
<tr>
<th>n</th>
<th>Bias($\hat{n}_{MLE}, n$)</th>
<th>MSE($\hat{n}_{MLE}$)</th>
<th>NRMSE($\hat{n}_{MLE}$)</th>
<th>CV[$\hat{n}$]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>2</td>
<td>0.0</td>
<td>1.0</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>3</td>
<td>0.125</td>
<td>4.125</td>
<td>0.67703200386</td>
<td>0.57735026919</td>
</tr>
<tr>
<td>4</td>
<td>0.328125</td>
<td>9.484375</td>
<td>0.769917812172</td>
<td>0.612372435696</td>
</tr>
<tr>
<td>5</td>
<td>0.580078125</td>
<td>16.966796875</td>
<td>0.823815437462</td>
<td>0.632455532034</td>
</tr>
<tr>
<td>6</td>
<td>0.862091064433</td>
<td>26.4124450684</td>
<td>0.856550657975</td>
<td>0.645497224368</td>
</tr>
<tr>
<td>7</td>
<td>1.16192817688</td>
<td>37.68080616</td>
<td>0.876924195174</td>
<td>0.654653670708</td>
</tr>
<tr>
<td>8</td>
<td>1.47153985128</td>
<td>50.6706994362</td>
<td>0.889791929999</td>
<td>0.661437827766</td>
</tr>
<tr>
<td>9</td>
<td>1.78563172738</td>
<td>65.320132343</td>
<td>0.898009680878</td>
<td>0.666666666667</td>
</tr>
<tr>
<td>10</td>
<td>2.10075975438</td>
<td>81.598977533</td>
<td>0.903321580354</td>
<td>0.67082039325</td>
</tr>
</tbody>
</table>

\[
\text{NRMSE}(B) = \text{NRMSE}(A) = \frac{\text{CV}(\hat{n})}{\sqrt{k}}
\]

\[
\text{NRMSE}(B) = \text{CV}(\hat{n}) = \text{NRMSE}(\hat{n})
\]

\[
\text{NRMSE}(B) = \frac{0.6614}{\sqrt{8}} \quad \Rightarrow \quad n = 8
\]

\[
\text{NRMSE}(A) = \frac{\sqrt{\text{MSE}(A)}}{n} = \frac{\sqrt{\text{Var}[A] + \text{Bias}(A,n)^2}}{n} = \frac{1}{\sqrt{k}} \frac{\text{Var}[\hat{n}_{MLE}] + \text{Bias}(\hat{n}_{MLE},n)^2}{n}
\]

\[
\text{NRMSE}(A) = \frac{1}{\sqrt{k}} \cdot 0.7579 + 0.0338 \quad \Rightarrow \quad n = 8
\]
2. \( \hat{c}_i \leq c_i \) for all stream-\( k \)-streams. Let us denote \( \hat{c}_k^k \) the \( k \)-th \( \hat{c}_i \) value for the \( i \)-th stream.

We will prove that for all \( k \)-streams, \( \hat{c}_1^k \leq c_i \) holds.

Proof:

1. If \( j \neq i \), then the \( c \) values remain unchanged, \( c_i^{k+1} = c_i^k \).
2. If \( j = i \), then we have two sub-cases:
   a. If \( \hat{c}_i^k \) is unchanged, \( c_i^{k+1} = c_i^k + \Delta \).
   b. If \( \hat{c}_i^k \) changes, \( c_i^{k+1} = c_i^k + \Delta \).

We will prove the following statements:

- For all \( j \neq i \), \( c_i^{k+1} = c_i^k \).
- For all \( j = i \), \( c_i^{k+1} = c_i^k + \Delta \).
\[ c_{i+1}^{k+1} = \max\{0, c_i^k - 1\} \geq 0 \]

\[ \hat{c}_{i+1}^{k+1} = 0 \leq c_i^k \]

If \( \hat{c}_{i+1}^{k+1} \leq c_i^k + 1 \), then:

\[ \hat{c}_{i+1}^{k+1} = 0 \leq c_i^k \]

Therefore, by mathematical induction, the statement holds:

\[ \hat{c}_i \leq c_i \]

Let \( x \) be the closest upper bound of \( \hat{c}_i \).

We notice that the difference will increase when we receive another element with \( \hat{c}_i \) not found, and also \( c_i \), and throughout this we will decrease the total sum containing this element.

We will define the number of times the element \( c_i \) occurs as a \( decrement \), and note it in the sequence.

Therefore, the difference will decrease when we receive another element with \( \hat{c}_i \) not found, and also \( c_i \). In all other cases, the difference will remain constant.

The maximum difference for a single element:

\[ \hat{c}_i - c_i \leq d \]

Now, we can define two variables:

- \( m = \sum c_i \) - the total number of elements received
- \( m' = \sum \hat{c}_i \) - the total number of elements received with \( \hat{c}_i \)

Hence:

\[ m - m' = d(k+1) - (\Delta - d) \]

\[ \Delta = \Delta - (m - m') = d(k + 1) - (\Delta - d) \]

\[ d = \frac{\Delta - m' - d}{k + 1} \]

\[ \hat{c}_i - c_i \leq d = \frac{\Delta - m' - d}{k + 1} \]

The maximum number of elements to be transferred for each element is

\[ d_{max} = m - m' \]

And the maximum number of elements to be transferred is

\[ d_{max} = m - m' \]

We can define the number of times the element \( c_i \) occurs as a \( decrement \), and note it in the sequence.

We notice that the difference will increase when we receive another element with \( \hat{c}_i \) not found, and also \( c_i \). In all other cases, the difference will remain constant.

The maximum difference for a single element:

\[ \hat{c}_i - c_i \leq d \]

Now, we can define two variables:

- \( m = \sum c_i \) - the total number of elements received
- \( m' = \sum \hat{c}_i \) - the total number of elements received with \( \hat{c}_i \)

Hence:

\[ m - m' = d(k+1) - (\Delta - d) \]

\[ \Delta = \Delta - (m - m') = d(k + 1) - (\Delta - d) \]

\[ d = \frac{\Delta - m' - d}{k + 1} \]

\[ \hat{c}_i - c_i \leq d = \frac{\Delta - m' - d}{k + 1} \]

And the maximum number of elements to be transferred for each element is

\[ d_{max} = m - m' \]

And the maximum number of elements to be transferred is

\[ d_{max} = m - m' \]

We can define the number of times the element \( c_i \) occurs as a \( decrement \), and note it in the sequence.

We notice that the difference will increase when we receive another element with \( \hat{c}_i \) not found, and also \( c_i \). In all other cases, the difference will remain constant.
A. Committed unbiased count $c_i$ (for flow $i$), that may or may not contain a clipper.

$E[c_i] = E\left[\frac{x}{p}\right] = \frac{c_i p}{p} = c_i$

B. For each flow $i$:

$Var[c_i] = Var\left[\frac{x}{p}\right] = \frac{1}{p^2} \cdot Var[x] = \frac{1}{p^2} \cdot (c_i p (1 - p)) = c_i \cdot \frac{1 - p}{p}$

$CV[c_i] = \sqrt{\frac{Var[c_i]}{E[c_i]}} = \sqrt{\frac{1 - p}{c_i p}} \cdot \frac{1 - p}{c_i} = \sqrt{\frac{1 - p}{c_i p}}$

C. For streaming NetFlow, the net flow,

$\hat{c} = \hat{c}_1 + \hat{c}_2$

If flow is independent, if the counter is empty,

$E[\hat{c}] = c_1 + c_2$

If flow is independent, if the counter is full,

$E[\hat{c}] = E[\hat{c}_1] + E[\hat{c}_2] = c_1 + c_2$

D. Each flow's counter is independent.

For $m$ flows, the total counter:

$Pr[\text{new counter on packet } i] = p \cdot (1 - p)^{i-1}$
ברור כי ההסתברות להיות פקטהافluent stream - ולהפתוע card counter בלבפלט. יורדת עם מיומנו בולו כל פקטה, ובחום, אם מצפה ל- stream שווה 1, ונכבר כי古老 - אוזה כשכריה במקסימום.誠mong - אם נתחק فلا פקטה להפתוע card counter דרוש בלבפלט כל פקטה, p, וכל נקבה את המקסימום של התוכן.

נ.Objects:

\[ n = \sum_{f=1}^{k} \sum_{i=1}^{c} \operatorname{Pr}[\text{new counter on packet } i] = \sum_{f=1}^{k} \sum_{i=1}^{c} p \cdot (1 - p)^{i-1} \]

ברור כי ההסתברות להיות פקטה card counter הפתוע stream - ולהפתוע card counter בלבפלט. יורדת עם מיומנו בולו כל פקטה, ובחום, אם מצפה ל- stream שווה 1, ונכבר כי古老 - אוזה כשכריה במקסימום.誠mong - אם נתחק فلا פקטה להפתוע card counter דרוש בלבפלט כל פקטה, p, וכל נקבה את המקסימום של התוכן.

נ.Objects:

\[ n = \sum_{f=1}^{k} \sum_{i=1}^{c} p \cdot (1 - p)^{i-1} = \sum_{f=1}^{c} \sum_{i=1}^{1} p \cdot (1 - p)^{i-1} = \sum_{f=1}^{c} p = cp \]
MAX_X = 10 # the maximum value for x
MAX_N = 10 # the maximum value for n (for computing section B)
# Using dynamic programing we compute each value of f(x,n) once and store it inside this table:
table = [ ]
# a row in the table holds all values of possibilities for where x is the (row-index + 1)
# the index inside the row corresponds to value of n.
def f(x, n):
    # This function retrieves the value if previously computed, else computes it first.
    # f(x;n) = table[x-1][n-1]
    if len(table[x-1]) < n:
        fill_row(x-1, n)
    return table[x-1][n-1]

def compute(x, n):
    # This function computes the value of f(x,n).
    if x == 1 and n == 1:
        f_xn = 1.0
    elif x > n:
        f_xn = 0.0
    else:
        f_xn = (1 - 2**(-x)) * f(x, n - 1)
        if x > 1:
            f_xn += 2**(-(x - 1)) * f(x - 1, n - 1)
    return f_xn

def fill_row(i, max_n = -1):
    # This function fills the row (which corresponds to a value of x) until we either
    # reach the maximum likelihood for that value of x or until we reach max_n (inorder
    # to calculate another value: f(x, max_n) which is needed)
    x = i + 1
    last = 0
    value = last
    n = len(table[i]) + 1
    while not (last > value or ((max_n != -1) and (n > max_n))):
        last = value
        value = compute(x, n)
        table[i].append(value)
        n = n + 1

def fill_table(max_x = MAX_X):
    # This function fills the entire table up to the value of x = max_x
    for i in xrange(max_x):
        fill_row(i)

def print_sum():
    # prints the outcome of the run
    print "x\tn_mle\tMax-Likelihood"
    print "-" * 30
    n_mle = []
    for i in xrange(MAX_N):
        max_value = max(table[i])
        n_value = table[i].index(max(table[i])) + 1
        print i+1, "\t", n_value, "\t", max_value
        n_mle.append(n_value)
    print "" * 40
    for j in xrange(MAX_N):
        n = j + 1
        col = f(i+1, n) for i in xrange(n)]
        E_mle = sum([n_mle[i] * p for i, p in enumerate(col)])
        E_mle2 = sum([n_mle[i] ** 2 * p for i, p in enumerate(col)])
        a = E_mle - n
        b = E_mle2 - 2 * n * E_mle + n ** 2
        c = b ** 0.5 / n
        d = ((n - 1) / (2.0 * n)) ** 0.5
        print n, "\t", a, "\t", b, "\t", c, "\t", d
    print ""

def main():
    fill_table()
    print_sum()
if __name__ == "__main__":
    main()
import os
os.system("pause")