EE 214 Machine Problem 1.4

Paul Vincent S. Nonat
Electrical and Electronics Engineering Institute
University of the Philippines Diliman
paul.vincent.nonat@eee.upd.edu.ph

1. The Problem

The Zipf(n,$\alpha = 1$) random variable X has PMF,

$$P_X(x) = \Big\{ c(n)/x x = 1, 2, ..., n \tag{1}$$

where the constant $c(n)$ is set so that $\sum_{x=1}^{n} P_X(x) = 1$. This function is often used to model the popularity of a collection of n objects. For example a Web server can deliver one of n web pages. The pages are numbered such that page 1 is the most requested page, page 2 is the $2^n d$ most requested page, page 3 is the $3^r d$ most requested page, and so on. If page k is requested then $X = k$.

To reduce external network traffic, an ISP gateway caches the k most popular pages. Write a Python function **zipfunc (. . .)** to calculate, as a function of n for $1 \leq n \leq 1000$, how large k must be to ensure that the cache can deliver a page with a probability of 0.70, 0.80, and 0.90.

2. The Solution

The problem asks us to find the smallest value of k such that $P[X_n \leq k] \geq p$ where $p = 0.70$, 0.80 and 0.90. That is if the server caches the k most popular files with $P[X_n \leq k]$. To solve this, we define a function that computes the number of k files for any probability p. Which is presented in eqn. 2.

$$k = min\{k' | P[X_n \leq k'] \geq p\} \tag{2}$$

The Zipf distribution is hard to analyze since there was no closed form expression for

$$c(n) = ((\sum_{x}^{n} (\frac{1}{x}))^{-1} \tag{3}$$

Thus, we use python to grind this numerical calculations. An excellent way to solve this problem is through the use of Zipf PDF

$$P[X_n \leq k'] = c(n) \sum_{k=1}^{k'} (\frac{1}{x}) = \frac{c(n)}{c(k')} \tag{4}$$

In which we wish to find the minimum k such that

$$k = min\{k' | \frac{1}{c(k')} \geq \frac{p}{c(n)}\} \tag{5}$$

The definition of k implies that

$$\frac{1}{c(k')} < \frac{p}{c(n)} \tag{6}$$

$$for\ k' = 1, \ldots\ldots, k-1 \tag{7}$$

Thus, to find k for any probability p we calculate

$$k = 1 + |k\{\frac{1}{c(k')} < \frac{p}{c(n)}\}| \tag{8}$$

3. Results and Discussion

The eqn. 3 and eqn. 8 were used as the basis of the python program for calculating the size of cache k for $1 \leq n \leq 1000$.

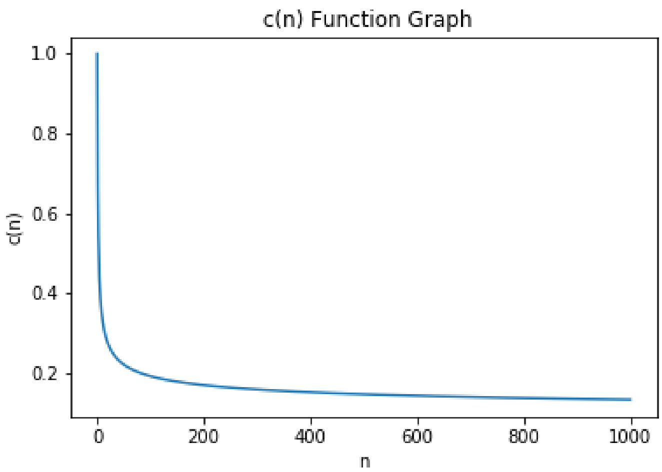

Fig. 1. Characteristic of c(n) function.

```
def constant_set(n):
    reciprocal_cn=0 #reciprocal of cn
    for i in range(n):
        reciprocal_cn= reciprocal_cn+ (1/(i+1))

    cn = 1/reciprocal_cn
```

```
    return cn;

def zipfunc(n,p):
    k=np.zeros(n)
    for m in range (n):
        k_prime=1
        print("Calculating K at n="+str(m+1))
        a= 1/constant_set(k_prime)
        b=p/constant_set(m+1)
        while a < b:
            print("k = "+str(k_prime)+" at n = "+str(m+1))
            print("%f < %f " % (a,b))
            k_prime = k_prime + 1
            k[m] =1+k_prime
            a= 1/constant_set(k_prime)
            b=p/constant_set(m+1)
    return k;
```

The constant set function calculates $c(n)$ and $c(k')$ for all input n and k'. This function produce an output as shown in Fig 1. Which we can see that the function approaches 0 as n increase from $1 \leq n \leq \infty$.

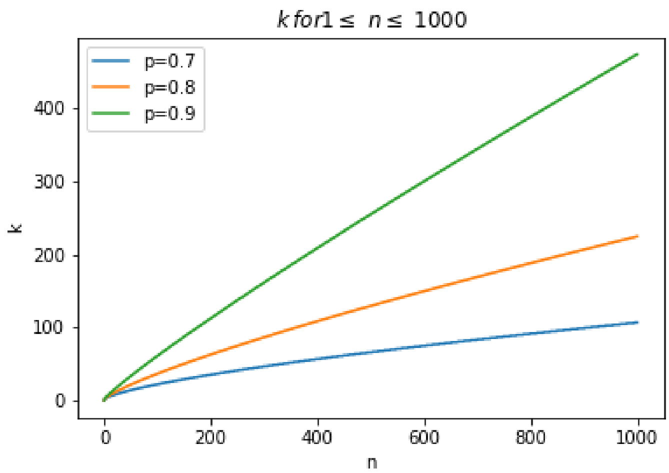

Fig. 2. Characteristic of c(n) function.

To determine k for $1 \leq n \leq 1000$ at a given probability p, the *zipfunc(n,p)* iterates from 1 to n while simultaneously calculating for $\frac{1}{c(k')}$ and $\frac{p}{c(n)}$. Each set of calculation terminates whenever eqn. 6 no longer holds true, that is if $\frac{1}{c(k')} > \frac{p}{c(n)}$ which gives us the value of k'. k is then calculated using eqn. 8 for every n until $n = 1000$. The size of k for $1 \leq n \leq 1000$ for $p = 0.70$,$p = 0.80$,$p = 0.90$ was presented in Fig 2. In there, we can see that the number of required cache k increases as the probability p of requesting page n increases.

4. Conclusion

We have shown how much k is needed for $1 \leq n \leq 1000$. The $zipfunc(n, p)$ was able to determine k in order to deliver a page with a given probability p. The *constantset* function was also able to model $c(n)$ in which $\sum_{x=1}^{n} P_X(x) = 1$ holds true for all n. Lastly, the $zipfunc(n, p)$ were able to make it easier to compute k for all n instead of creating a function that determines k with a given probability p on a particular n only.