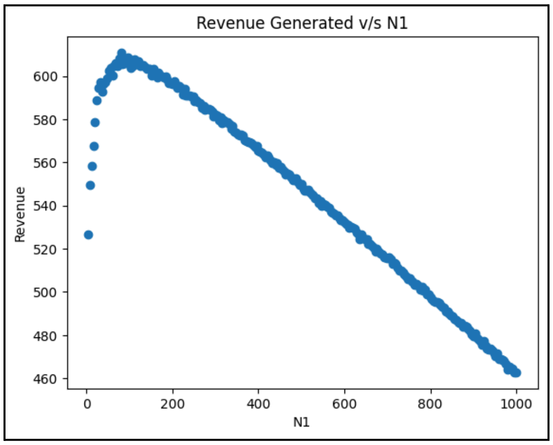
PROBABILITY AND RANDOM PROCESSES

EE325 PROGRAMMING ASSIGNMENT - 3

Aminesh Gogia - 23B1210
Rushabh Gayakwad - 23B2464
Jaswin Reddy M - 23B1289

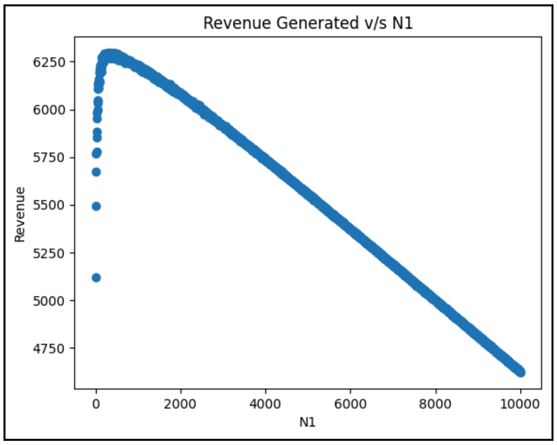
Algorithm-A:

(a)Selecting the right N1



Optimal N1 = 80

(N = 1000)

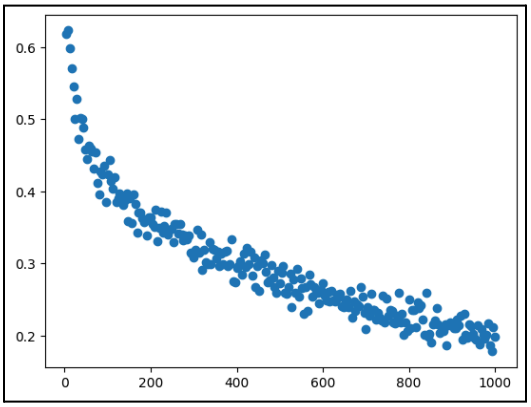


Optimal N1 = 336

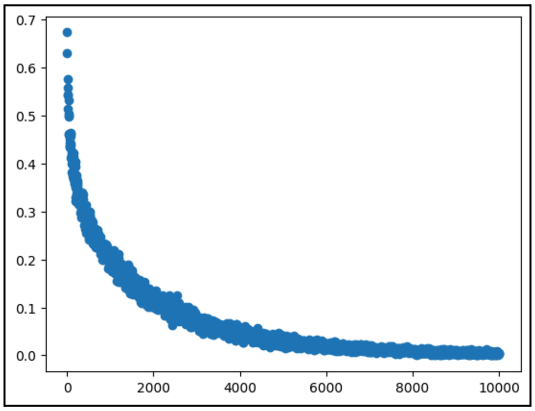
(N = 10000)

(b) Prob(wrong vid type selected for a given N1)

Empirical Results:



N = 1000

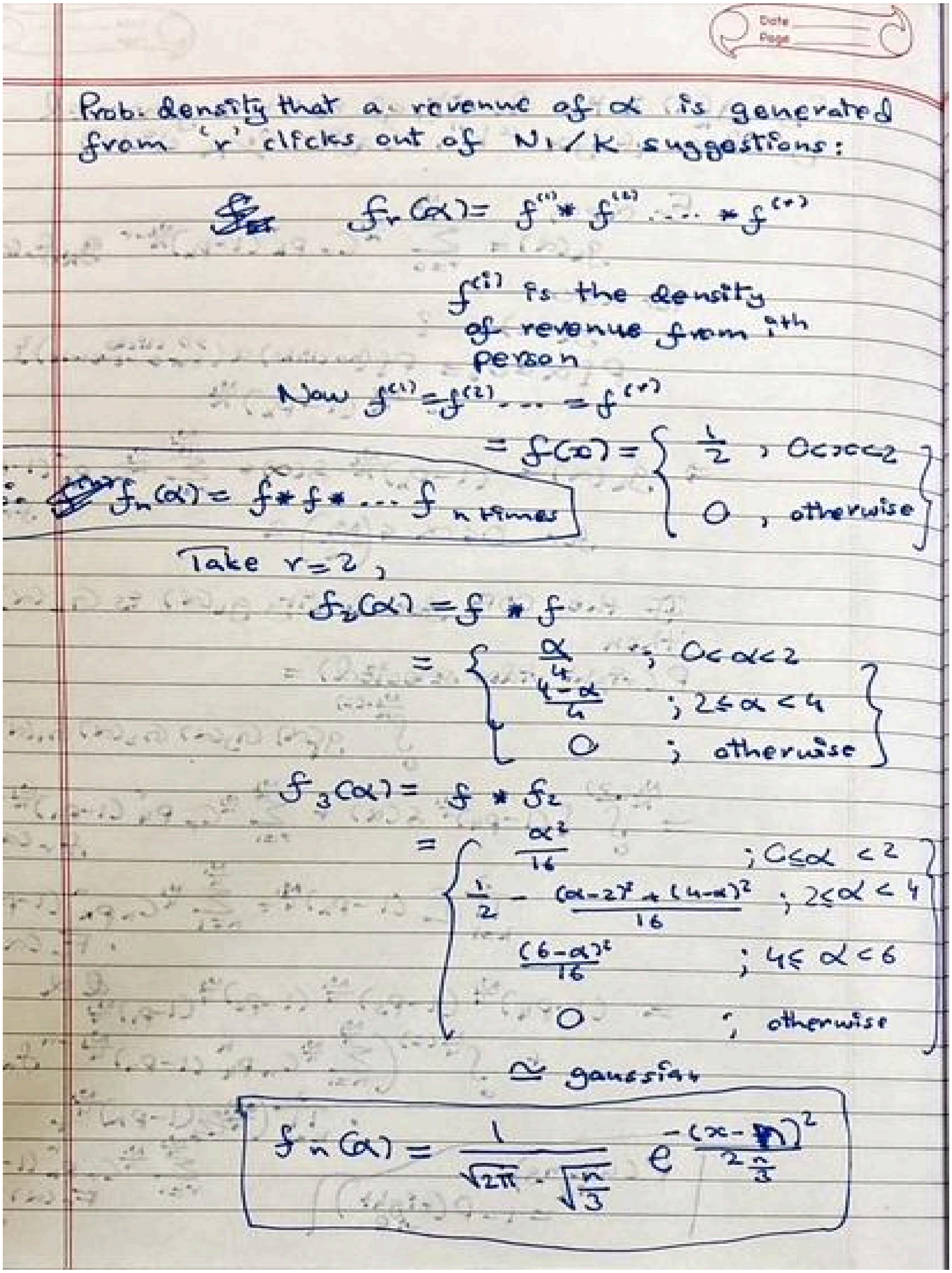


N = 10000

Higher N1 implies more opportunity to explore and determine the video-type that generates the most revenue, and thus a lesser chance to pick the wrong type.

Theoretical Expression:

The distribution of the sum of i.i.d. Uniform Random Variables is approximated to be a Gaussian, to make the computation feasible.



Prob. density that a revenue of α is generated from 'r' clicks out of N1/K suggestions:

$$f_r(\alpha) = f^{(1)} * f^{(2)} \cdots * f^{(r)}$$

$f^{(i)}$ is the density of revenue from i^{th} person

Now $f^{(1)} = f^{(2)} \cdots = f^{(r)}$

$$= f(x) = \left\{ \begin{array}{ll} \frac{1}{2} , & 0 < x < 2 \\ 0 , & \text{otherwise} \end{array} \right\}$$

$$\therefore \boxed{f_n(\alpha) = f * f * \cdots f_{\text{n times}}}$$

Take $r = 2$,

$$f_2(\alpha) = f * f$$

$$= \left\{ \begin{array}{ll} \frac{\alpha}{4} & ; 0 < \alpha < 2 \\ \frac{4-\alpha}{4} & ; 2 \leq \alpha < 4 \\ 0 & ; \text{otherwise} \end{array} \right\}$$

$$f_3(\alpha) = f * f_2$$

$$= \left\{ \begin{array}{ll} \frac{\alpha^2}{16} & ; 0 \leq \alpha < 2 \\ \frac{1}{2} - \frac{(\alpha-2)^2 + (4-\alpha)^2}{16} & ; 2 \leq \alpha < 4 \\ \frac{(6-\alpha)^2}{16} & ; 4 \leq \alpha < 6 \\ 0 & ; \text{otherwise} \end{array} \right\}$$

$\simeq$ gaussian

$$\boxed{f_n(\alpha) = \frac{1}{\sqrt{2\pi} \cdot \sqrt{\frac{n}{3}}} e^{-\frac{(x-n)^2}{2\frac{n}{3}}}}$$

Say $g_k(\alpha)$ is the density of revenue generated by videos of type "k". (after N_1/K clicks)

For $\alpha > 0$,

$$g_k(\alpha) = \sum_{r=0}^{\frac{N_1}{K}} {}^{\frac{N_1}{K}}C_r \, p_k^r (1-p_k)^{\frac{N_1}{K}-r} f_r(\alpha)$$

For $\alpha = 0$

$$g_k(\alpha) = \, ?$$

$$P\{\alpha = 0\} = P\{(\text{no clicks}) \cup (\substack{>0 \text{ clicks} \\ \text{but } 0 \text{ revenue}})\}$$

$$= (1-p_k)^{\frac{N_1}{K}}$$

$$\Rightarrow g_k(\alpha) = (1-p_k)^{\frac{N_1}{K}} \delta(\alpha) + \sum_{r=0}^{\frac{N_1}{K}} {}^{\frac{N_1}{K}}C_r \, p_k^r (1-p_k)^{\frac{N_1}{K}-r} f_r(\alpha)$$

for $0 \leq \alpha \leq \left(\frac{N_1}{K}\right) 2$

If the CDF for density $g_k(\alpha)$ is $G_k(\alpha)$, then

P(right video selected) =

$$\int_0^{\frac{N_1}{K}\cdot(2)} g_4(\alpha) \, G_3(\alpha) \, G_2(\alpha) \, G_1(\alpha) \, d\alpha$$

$$= \int_0^{\frac{N_1}{K}(2)} \Big[(1-p_4)^{\frac{N_1}{4}} \delta(\alpha) + \sum_{r=1}^{\frac{N_1}{4}} {}^{\frac{N_1}{4}}C_r \, p_4^r (1-p_4)^{\frac{N_1}{4}-r} f_r(\alpha)\Big]$$

$$\prod_{k=1}^{3} \Big[(1-p_k)^{\frac{N_1}{4}} + \sum_{n=1}^{\frac{N_1}{4}} {}^{\frac{N_1}{4}}C_n \, p_k^n (1-p_k)^{\frac{N_1}{4}-n} \cdot F_r(\alpha)\Big]$$

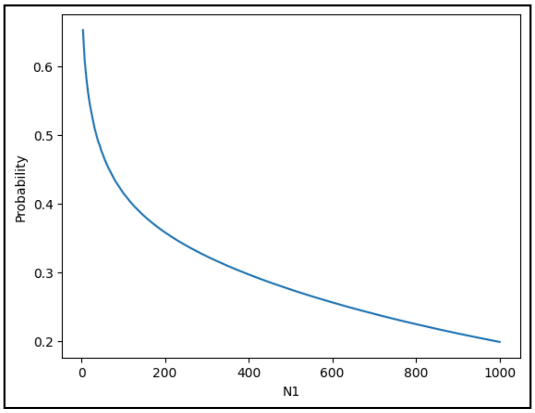
$$= (1-p_4)^{\frac{N_1}{4}} (1-p_3)^{\frac{N_1}{4}} (1-p_2)^{\frac{N_1}{4}} (1-p_1)^{\frac{N_1}{4}} \, d\alpha$$

$$+ \int_0^{\frac{N_1}{4}(2)} \left(\sum_{n=1}^{\frac{N_1}{4}} {}^{\frac{N_1}{4}}C_n \, p_4^n (1-p_4)^{\frac{N_1}{4}-n} f_n(\alpha)\right)$$

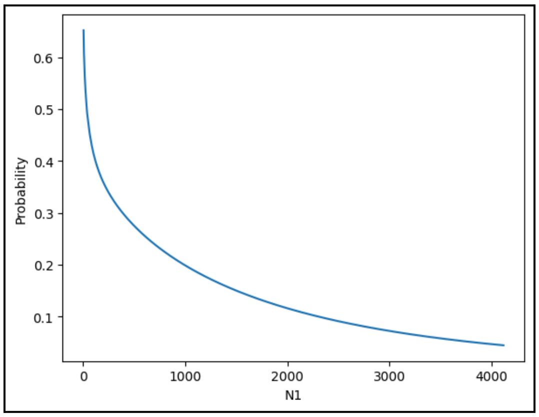
$$\cdot \prod_{k=1}^{3} \left((1-p_k)^{\frac{N_1}{4}} + \sum_{r=1}^{\frac{N_1}{4}} {}^{\frac{N_1}{4}}C_r \, p_k^r (1-p_k)^{\frac{N_1}{4}} \right) F_r(\alpha) \, d\alpha$$

$$\boxed{P(\text{wrong vid}) = 1 - P(\text{right vid})}$$

Theoretical Expression



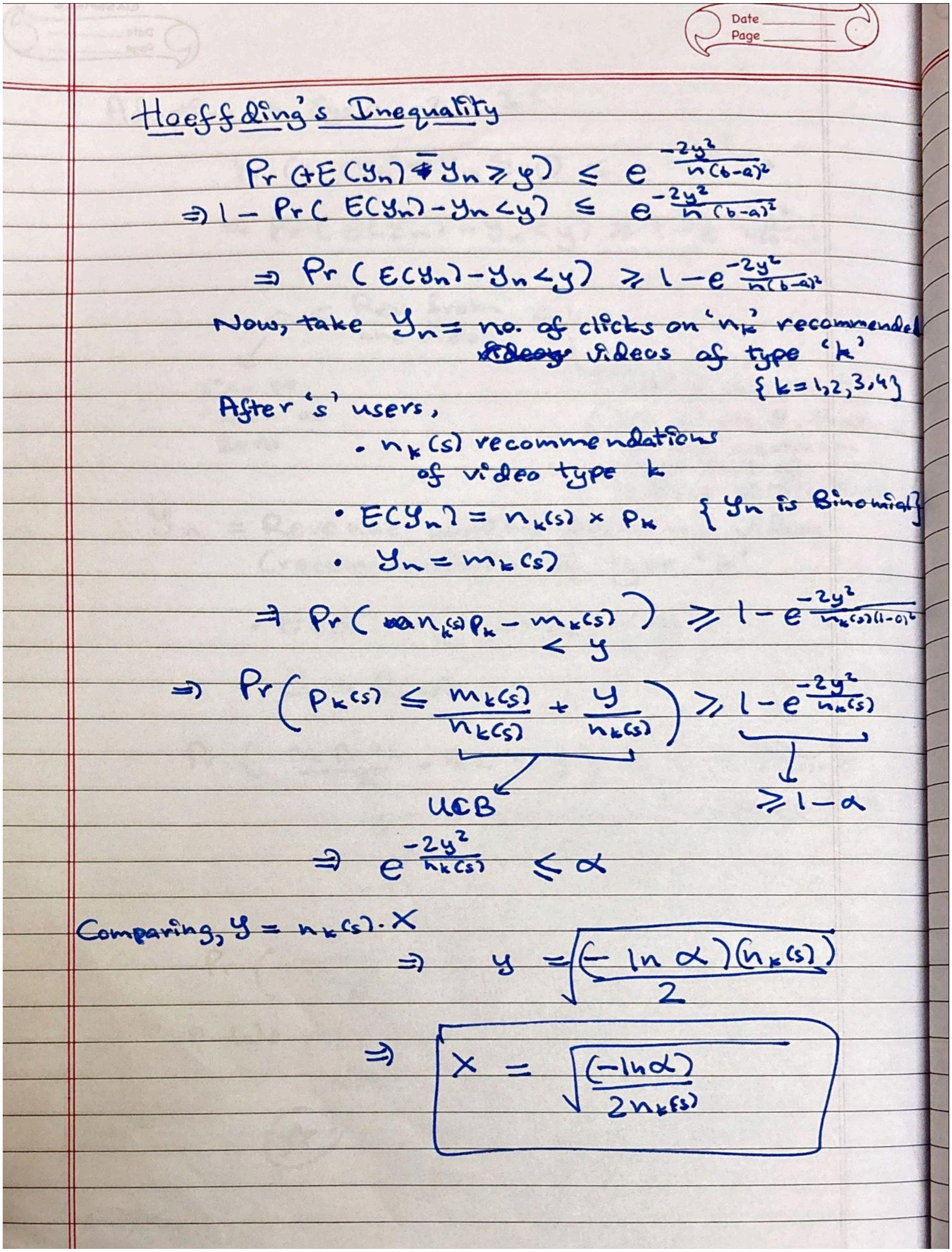
N = 1000



N = 10000

The plots obtained support the simulation results.

Algorithm-B:



Date

Page

Hoeffding's Inequality

$$\Pr(\cancel{+}E(Y_n) \cancel{+} Y_n \geq y) \leq e^{\frac{-2y^2}{n(b-a)^2}}$$

$$\Rightarrow 1 - \Pr(E(Y_n) - Y_n < y) \leq e^{-\frac{2y^2}{n(b-a)^2}}$$

$$\Rightarrow \Pr(E(Y_n) - Y_n < y) \geq 1 - e^{\frac{-2y^2}{n(b-a)^2}}$$

Now, take Y_n = no. of clicks on 'n_k' recommended ~~ideos~~ videos of type 'k'

$\{k = 1, 2, 3, 4\}$

After 's' users,

- $n_k(s)$ recommendations of video type k
- $E(Y_n) = n_k(s) \times p_k$ $\{Y_n$ is Binomial$\}$
- $Y_n = m_k(s)$

$$\Rightarrow \Pr(\cancel{an}\, n_k(s) p_k - m_k(s) < y) \geq 1 - e^{\frac{-2y^2}{n_k(s)(1-0)^2}}$$

$$\Rightarrow \Pr\left(p_k(s) \leq \underbrace{\frac{m_k(s)}{n_k(s)} + \frac{y}{n_k(s)}}_{\text{UCB}}\right) \geq \underbrace{1 - e^{\frac{-2y^2}{n_k(s)}}}_{\geq 1-\alpha}$$

$$\Rightarrow e^{\frac{-2y^2}{n_k(s)}} \leq \alpha$$

Comparing, $y = n_k(s) \cdot X$

$$\Rightarrow y = \sqrt{\frac{(-\ln\alpha)(n_k(s))}{2}}$$

$$\Rightarrow \boxed{X = \sqrt{\frac{(-\ln\alpha)}{2n_k(s)}}}$$

Adapting to System-2 & 3:

$$Pr\,(+E(Y_n) - Y_n \geq y) \leq e^{\frac{-2y^2}{n(b-a)^2}}$$

$$\Rightarrow Pr\,(E(Y_n) - Y_n < y) \geq 1 - e^{\frac{-2y^2}{n(b-a)^2}}$$

$a \leq$ Rev. from one video $\leq b$

a → Can be taken as zero

b → Taking this as 20 (2.5 times 8, that is the maximum upper limit of revenue distributions)

Y_n = Revenue coming from n_k videos (recommended) of type 'k'

- $E(Y_n) = (n_k(s) \cdot p_k) \cdot \left(\frac{a_k}{2}\right)$
- $Y_n = R_k(s)$

$$Pr\left(\frac{n_k p_k a_k}{2} - R_k < y\right) \geq 1 - e^{\frac{-2y^2}{n(20)^2}} \geq \cancel{\geq 1-\alpha}\ 1-\alpha$$

$$\boxed{y = \sqrt{\frac{(-\ln\alpha) \cdot n \cdot 20^2}{2}}}$$

$$Pr\left(\frac{p_k a_k}{2} - \frac{R_k}{n_k} < \frac{y}{n_k}\right) \geq 1-\alpha$$

⇒ We recommend video with highest value of

$$\bigstar \quad \frac{R_k}{n_k} + \sqrt{\frac{(-\ln\alpha) \cdot 20^2}{2 n_k}}$$

N = 10000 (System - 1)

Alpha	Total Revenue
0.01	6453.20
0.05	6460.01
0.1	6458.37

Best Expected Revenue: (2/2) * 0.65 * 1000 = 6500

N = 1000 (System - 1)

Alpha	Total Revenue
0.01	625.08
0.05	628.12
0.1	629.92

Best Expected Revenue: (2/2) * 0.65 * 1000 = 650

(All plots for Algorithm B are included along with the programs)

Effect of Alpha:

Lower values of alpha made the ratio m_k/n_k converge to p_k better, keeping the number of users the same.

Effect of N:

Higher average revenue per user is earned, because a lot more number of users are available once the most lucrative video type has been detected.

Effect of Upper Bound of Revenue:

While implementing Algorithm B to systems 2 and 3, an upper bound on the revenue from a video was needed. We assumed that the upper bound was the same for each of the video types (the upper bound was taken to be 20). Taking a very high upper bound, results in significantly lower revenue but good convergence of m/n. To make the algorithm work best, we require the tightest bound, and that is when the algorithm would be able to calculate UCBs accurately and suggest video types more effectively.

A key assumption in this exercise is that the users do not change their preferences as the s increases from 1 to N. How realistic is that assumption? Suggest ways to capture the effect of the recommendation sequence on the change in pk.

- Right now, the UCB is updated only when there is a change in m_k and n_k. This makes it possible for the algorithm to ignore increase in popularity (and maybe an increase in prospective revenue) from other video types. Periodic recommendation of a randomly selected video type other than that with the highest UCB will help incorporate changes in popularity in the video suggestion algorithm.
- We can test other solutions by varying p_k in the simulations, and reviewing the impact of changes in probability on the revenue-maximising performance of the algorithm.

The program with the plots for Algorithm B follows:

assignment-3

October 12, 2024

0.1 Part-1: Algorithm A

[28]:
```
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
K = 4
p_k = [0.2, 0.4, 0.6, 0.65]
a_k = [2, 2, 2, 2]
```

[3]:
```
N = 10000
N1 = 100
```

[4]:
```
def rev_algo_A(N, K, a_k, p_k, N1):
    n_tiny = N1//K
    num_clicks = np.ndarray((K, n_tiny))
    rev = np.zeros(K)

    for i in range(K):
        # equal_sugg_res[i, :] = np.random.uniform(low = 0.0, high = a_k[i],␣
 ↪size = (1, n_tiny))
        p = p_k[i]
        num_clicks[i, :] = np.random.choice(a=[0, 1], size = N1//K, p = [(1 -␣
 ↪p), p])
        rev[i] = np.sum(np.random.uniform(low = 0.0, high = a_k[i], size =␣
 ↪n_tiny) @ num_clicks[i])
    # print(np.sum(num_clicks, axis = 1))
    # rev = np.sum(equal_sugg_res @ num_clicks.T, axis = 1)
    # print(rev)
    best_vid = np.argmax(rev)

    # print(best_vid)

    temp = np.random.choice(a=[0, 1], p = [(1 - p_k[best_vid]), p_k[best_vid]],␣
 ↪size = N - N1)
    # rev_new = np.random.uniform(low = 0.0, high = a_k[i], size = (1, N - N1))

    rev_new = np.random.uniform(low = 0.0, high = a_k[best_vid], size = N - N1)
    rev_more = rev_new @ temp
    # print(rev_more)

    rev[best_vid] += rev_more

    return rev, best_vid
```

[ ]:

```
N1_test = tuple(range(4, 1004, 4))
rev_avg_2 = [0 for i in N1_test]
N = 1000

actual_best_vid = np.argmax(p_k)
wrong_sims = np.zeros(len(N1_test))
num_iter = 1000
for i in range(num_iter):
    for j in range(len(N1_test)):
        rev_typewise, sel_index = rev_algo_A(N, K, a_k, p_k, N1_test[j])
        if sel_index != actual_best_vid:
            wrong_sims[j] += 1
        rev_avg_2[j] += np.sum(rev_typewise)/num_iter
    if i % 10 == 0:
        print(i)

# np.argmax(rev_avg)
```

[ ]:

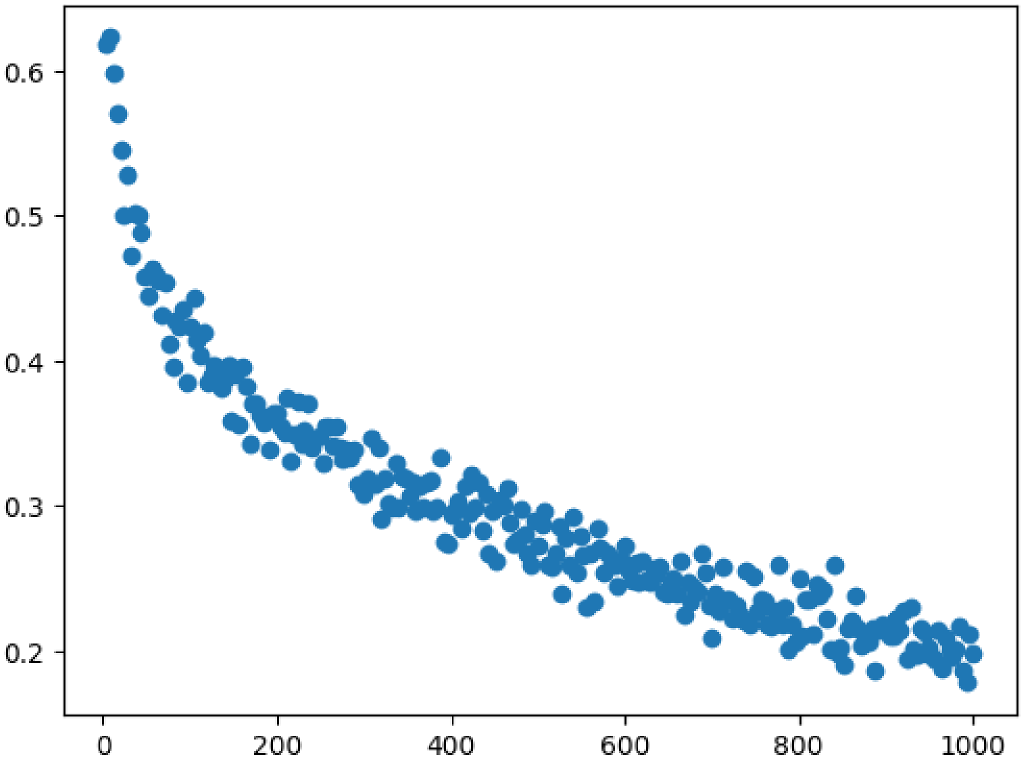
```
r_n2 = dict(zip(N1_test, rev_avg_2))
r_n2
```

[22]:

```
import matplotlib.pyplot as plt

plt.scatter(N1_test, wrong_sims/num_iter)
## Emperical Prob of selecting wrong vid type vs N1, N = 1000
```

[22]:

```
<matplotlib.collections.PathCollection at 0x2552ff42030>
```

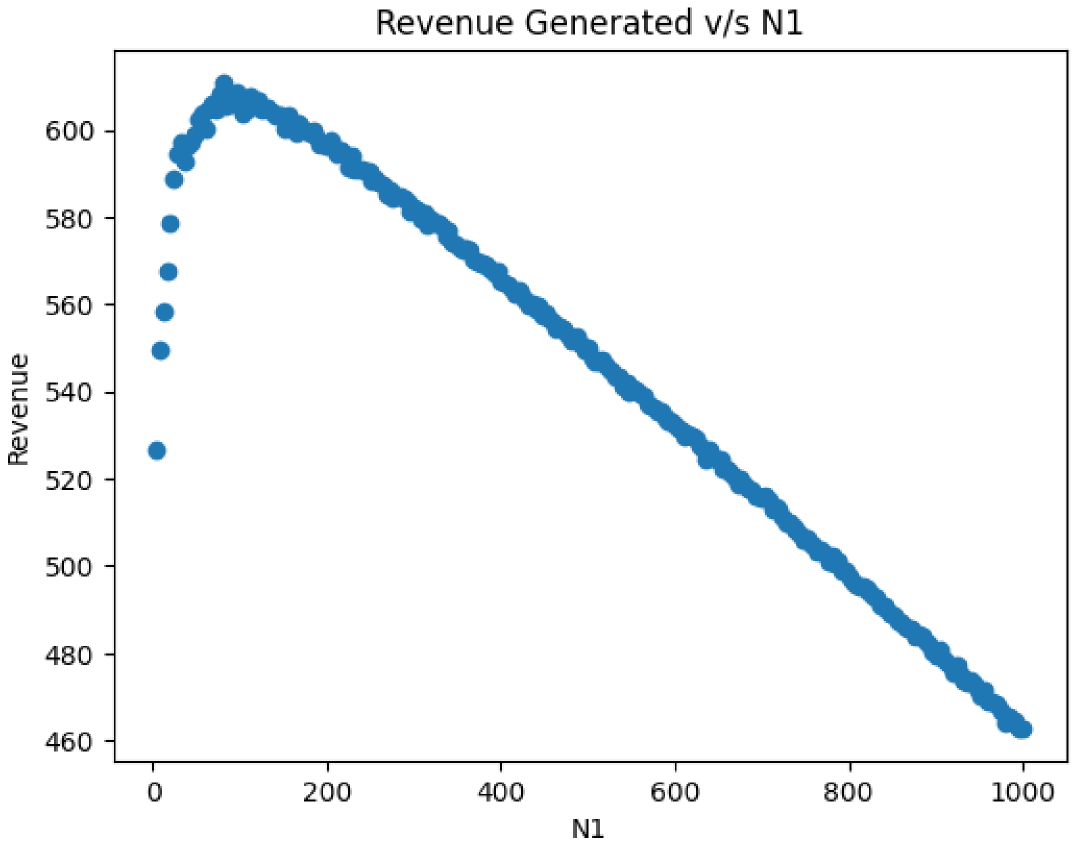


[29]:
```
len(rev_avg_2)
maxima_1000 = 4 * np.argmax(rev_avg_2) + 4
maxima_1000
```

[29]:
```
80
```

[24]:
```
import matplotlib.pyplot as plt

r_n2 = dict(zip(N1_test, rev_avg_2))
plt.scatter(r_n2.keys(), r_n2.values())
plt.title('Revenue Generated v/s N1')
plt.xlabel('N1')
plt.ylabel('Revenue')
## N = 1000
```

[24]:
```
Text(0, 0.5, 'Revenue')
```



[ ]:

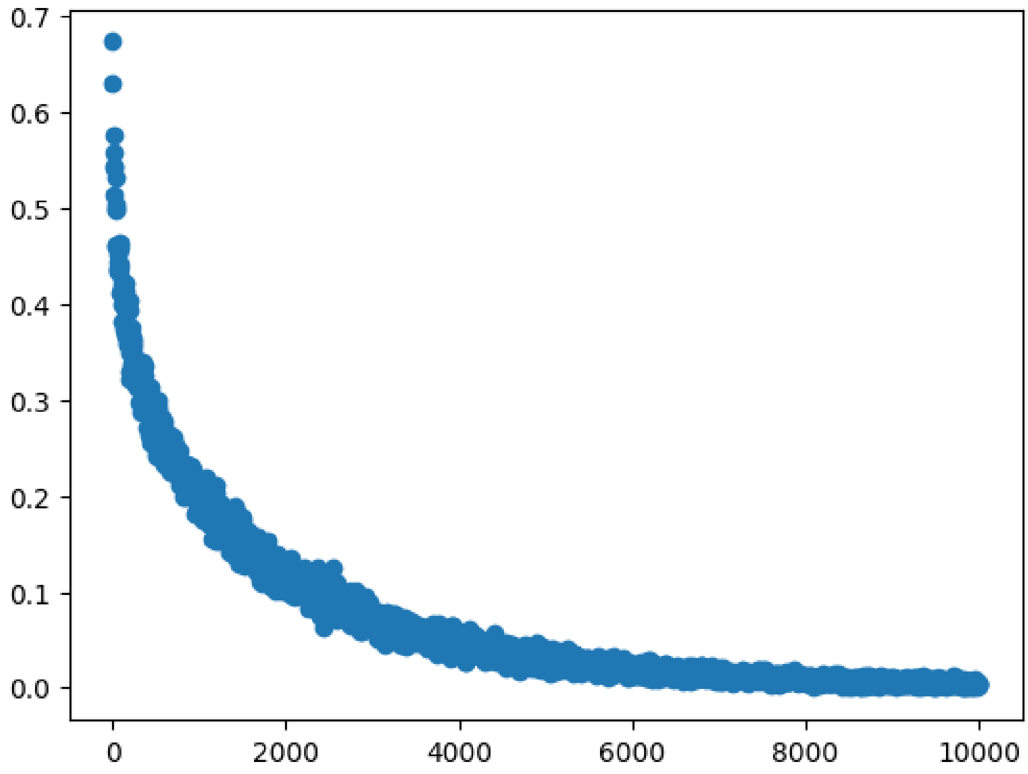
```
N = 10000
N1_test_10k = tuple(range(4, 10004, 4))
rev_avg_10k = [0 for i in N1_test_10k]
# rev_avg = rev_algo_A(N, K, a_k, p_k, 100)
# rev_avg
# rev_algo_A(N, K, a_k, p_k, 100)
actual_best_vid_10k = np.argmax(p_k)
wrong_sims_10k = np.zeros(len(N1_test_10k))
num_iter = 1000
for i in range(num_iter):
    for j in range(len(N1_test_10k)):
        rev_typewise, sel_index = rev_algo_A(N, K, a_k, p_k, N1_test_10k[j])
        if sel_index != actual_best_vid_10k:
            wrong_sims_10k[j] += 1
        rev_avg_10k[j] += np.sum(rev_typewise)/num_iter
    if i % 10 == 0:
        print(i)

# np.argmax(rev_avg)
```

[28]:

```
import matplotlib.pyplot as plt

plt.scatter(N1_test_10k, wrong_sims_10k/num_iter)
## Emperical Prob of selecting wrong vid type vs N1, N = 10000
```

[28]:

```
<matplotlib.collections.PathCollection at 0x2553311c920>
```



[30]:

```
len(rev_avg_10k)
maxima_10000 = 4 * np.argmax(rev_avg_10k) + 4
maxima_10000
```

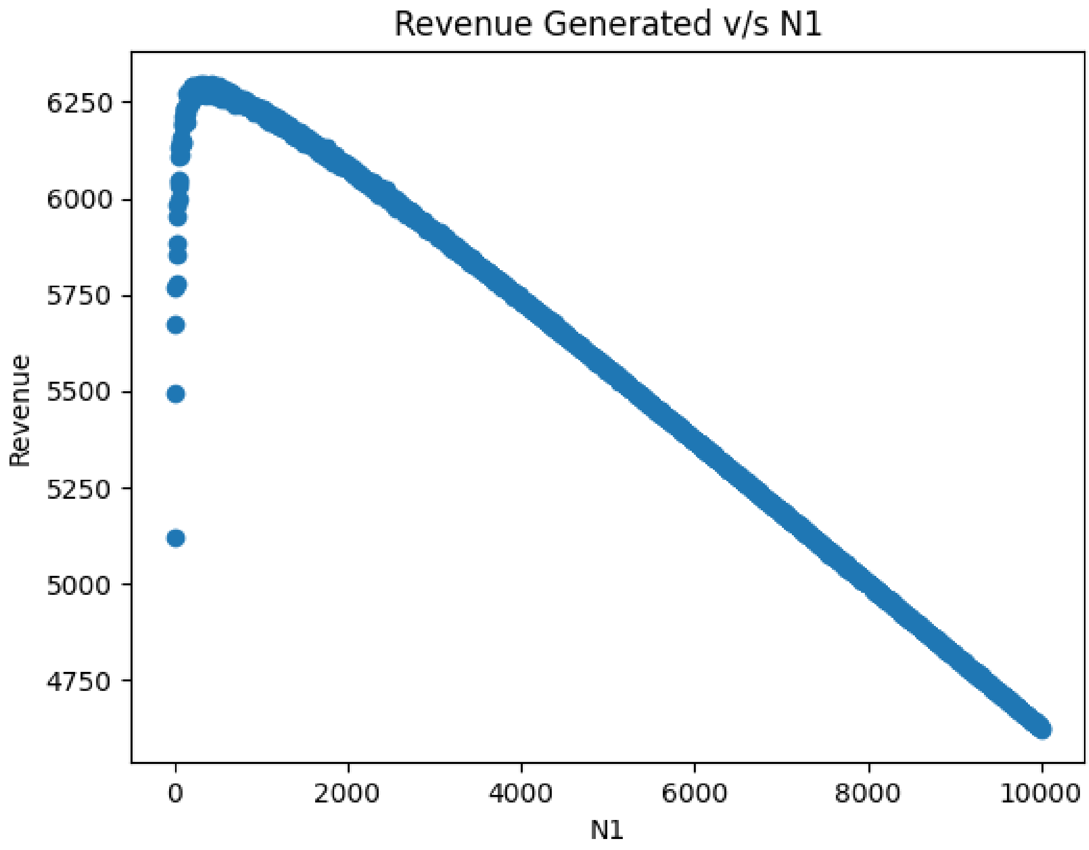
[30]:

```
336
```

[33]:

```
import matplotlib.pyplot as plt

r_n2_10k = dict(zip(N1_test_10k, rev_avg_10k))
plt.scatter(r_n2_10k.keys(), r_n2_10k.values())
plt.title('Revenue Generated v/s N1')
plt.xlabel('N1')
plt.ylabel('Revenue')
## N = 10000
```

```
[33]: Text(0, 0.5, 'Revenue')
```



```
[515]: L=1000

def f_E_n(n, M):
    alphas = np.linspace(0, 2 * M, L)
    func = np.zeros(L)
    if n==1:
        func[: int(2/(2*M)*L)]=1/2
    if n==2:
        func[: int(2/(2*M)*L)]=alphas[:int(2/(2*M)*L)]/4
        func[int(2/(2*M)*L) : int(4/(2*M)*L)] = 1 - alphas[int(2/(2*M)*L) :int(4/(2*M)*L)]/4
    if n>=3:
        func[:]=np.exp(-3/2*np.square(alphas-n)/n) / np.sqrt(2/3*n*np.pi)
    return func

def F_E_n(n, M):
    return np.cumsum(f_E_n(n, M)*(2*M/L))
```

```
from scipy.special import comb
def term_4(M):
    term=np.zeros(L)
    for n in np.arange(1,M+1):
        term += comb(M,n) * (p_k[3]**n) * ((1-p_k[3])**(M-n)) * f_E_n(n, M)
    return term

def other_terms(M):
    term=np.ones(L)
    for k in (0,1,2):
        term_k=np.zeros(L)
        for n in np.arange(1,M+1):
            term_k += comb(M,n) * (p_k[k]**n) * ((1-p_k[k])**(M-n)) * F_E_n(n,␣
 ↪M)
        term_k += (1-p_k[k])**M
        term *= term_k
    return term

def theor_Prob_wrong(M):
    return 1 -  np.sum(term_4(M)*other_terms(M)*(2*M/L)) # -␣
 ↪((1-pk[0])*(1-pk[1])*(1-pk[2])*(1-pk[3]))**M
```

[ ]:

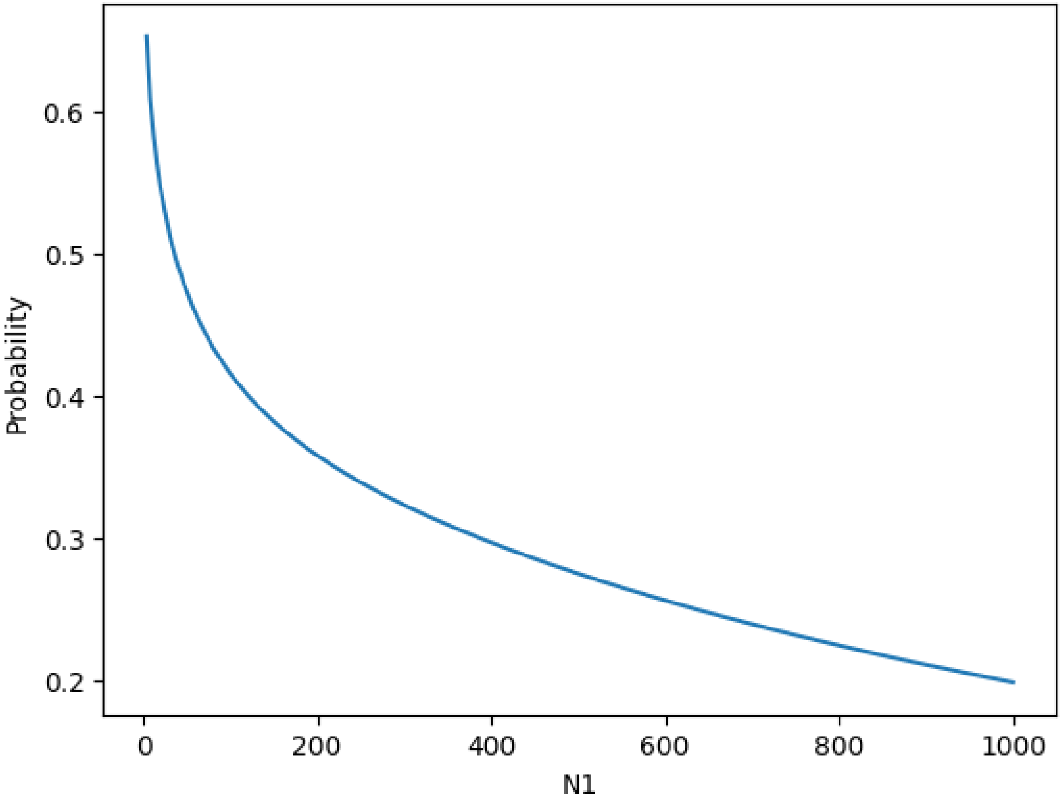
```
N1_test = np.arange(4, 1004, 4)
theor_list=np.empty(N1_test.size)
done = 0
count = 0
for N1 in N1_test:
    theor_list[N1//K-1]=theor_Prob_wrong(N1//K)
    if int(count/N1_test.size*100)>done:
        done=int(count/N1_test.size*100)
        # print(done)
    count+=1
    print(N1)
```

[522]:

```
plt.plot(N1_test, theor_list)
plt.xlabel("N1")
plt.ylabel("Probability")
## N = 1000
```

[522]:

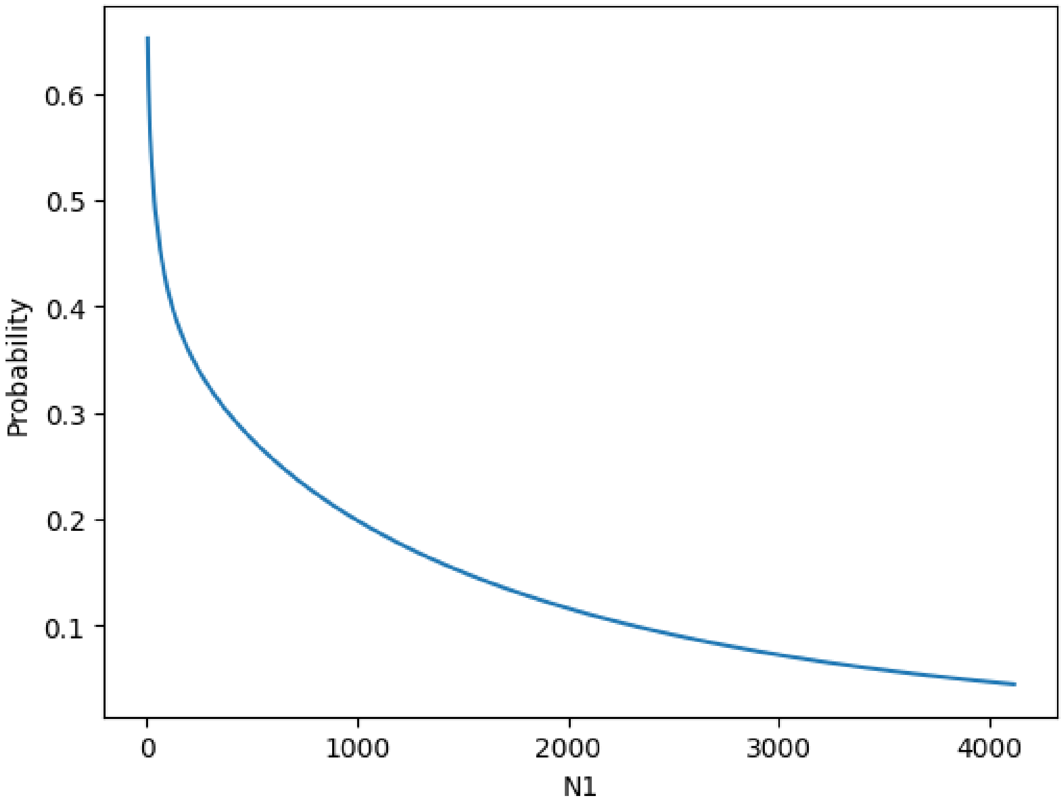
```
Text(0, 0.5, 'Probability')
```



[ ]:
```
N1_test = np.arange(4, 10004, 4)
theor_list=np.empty(N1_test.size)
done = 0
count = 0
for N1 in N1_test:
    theor_list[N1//K-1]=theor_Prob_wrong(N1//K)
    if int(count/N1_test.size*100)>done:
        done=int(count/N1_test.size*100)
        # print(done)
    count+=1
    print(N1)
```

[524]:
```
plt.plot(N1_test, theor_list)
plt.xlabel("N1")
plt.ylabel("Probability")

## N = 10000
```

[524]:
```
Text(0, 0.5, 'Probability')
```



0.2 Part-2: Hoeffding's Inequality

[11]:
```
K = 4
p_k = [0.2, 0.4, 0.6, 0.65]
a_k = [2, 2, 2, 2]
N = 10000
alpha = 0.1
```

[445]:
```
def ucb_prob(N, p_k, a_k, K, alpha):
    ucb =  np.ones((K, N + 1)) ## np.zeros
    n_k_s = np.zeros_like(ucb)
    m_k_s = np.zeros_like(ucb)
    m_by_n_s = np.zeros_like(ucb)
    # r_by_n_s = np.zeros_like(ucb)
    rev_total = np.zeros(K)
    rev_storer = np.zeros_like(ucb)

    for i in range(N - 1):
        max_rev = np.max(ucb[:, i])
        poss_ind = np.where(ucb[:, i] == max_rev)
        sel_ind = np.random.choice(poss_ind[0])
        n_k_s[sel_ind, (i + 1):] += 1
        if np.random.rand() <= p_k[sel_ind]:
            m_k_s[sel_ind, (i + 1):] += 1
            m_by_n_s[sel_ind, (i + 1):] = m_k_s[sel_ind, (i + 1)] /␣
↪n_k_s[sel_ind, (i + 1)]
            rev = np.random.uniform(low = 0.0, high = a_k[sel_ind])
            rev_total[sel_ind] += rev
        # r_by_n_s[sel_ind, (i + 1):] = rev_total[sel_ind]/n_k_s[sel_ind, i + 1]
        ucb[sel_ind, (i + 1):] = m_by_n_s[sel_ind, (i + 1)] + (-np.log(alpha)/
↪(2 * n_k_s[sel_ind, (i + 1)]))** 0.5
        rev_storer[:, (i + 1)] = rev_total
    return ucb, m_by_n_s, rev_storer[:, :N + 1]
```

0.2.1 System-1

N = 10000

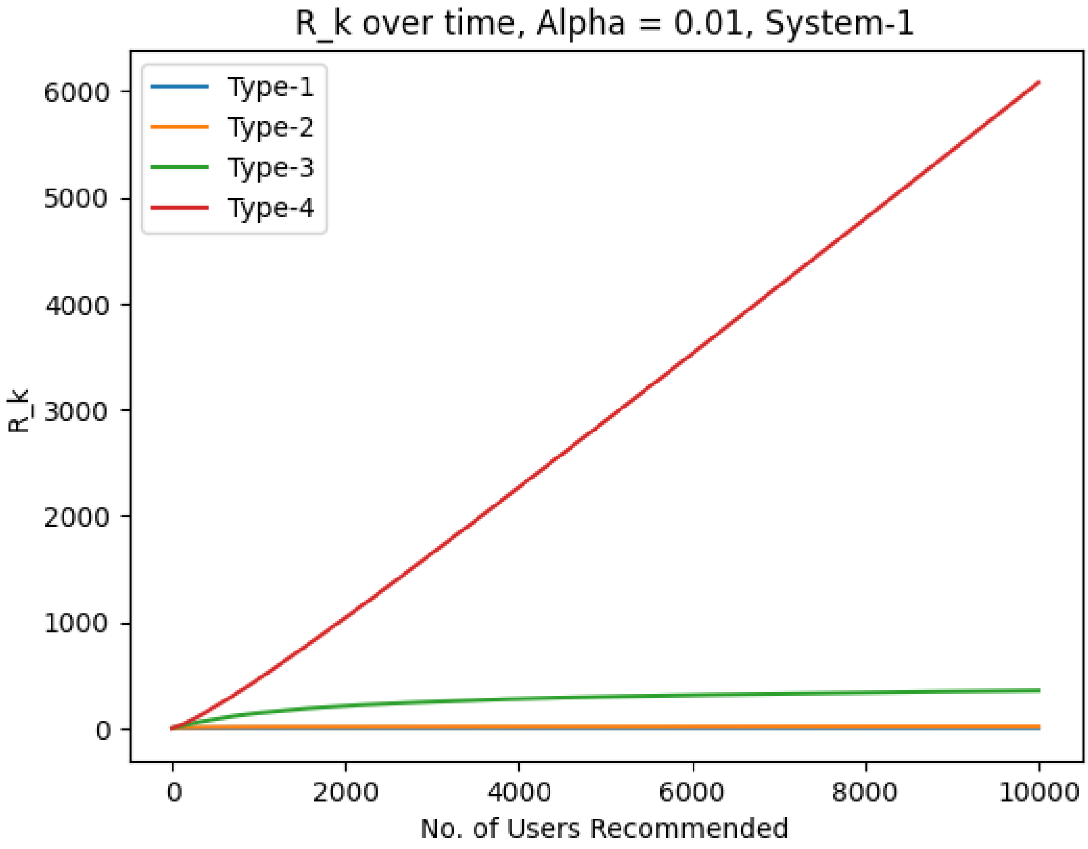
[ ]:

```
N = 10000
K = 4
ucb_p001 = np.zeros((K, N + 1))
num_iter = 1000
count_p001 = np.zeros(K)
rev_avgd_001 = np.zeros_like(ucb_p001)
m_by_nk_avgd_001 = np.zeros_like(ucb_p001)
a_k = [2, 2, 2, 2]
p_k = [0.2, 0.4, 0.6, 0.65]

for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, 0.01)
    ucb_p001 += res
    rev_avgd_001 += rev_storer
    m_by_nk_avgd_001 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p001[j] += 1
    if i % 10 == 0:
        print(i)
```

[505]:

```
plot_it = (rev_avgd_001.T/count_p001).T
for j in range(K):
    plt.plot(plot_it[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.01, System-1')
plot_it[:, -2]
```

```
[505]: array([3.07559183e+00, 1.53371686e+01, 3.55555348e+02, 6.07923582e+03])
```

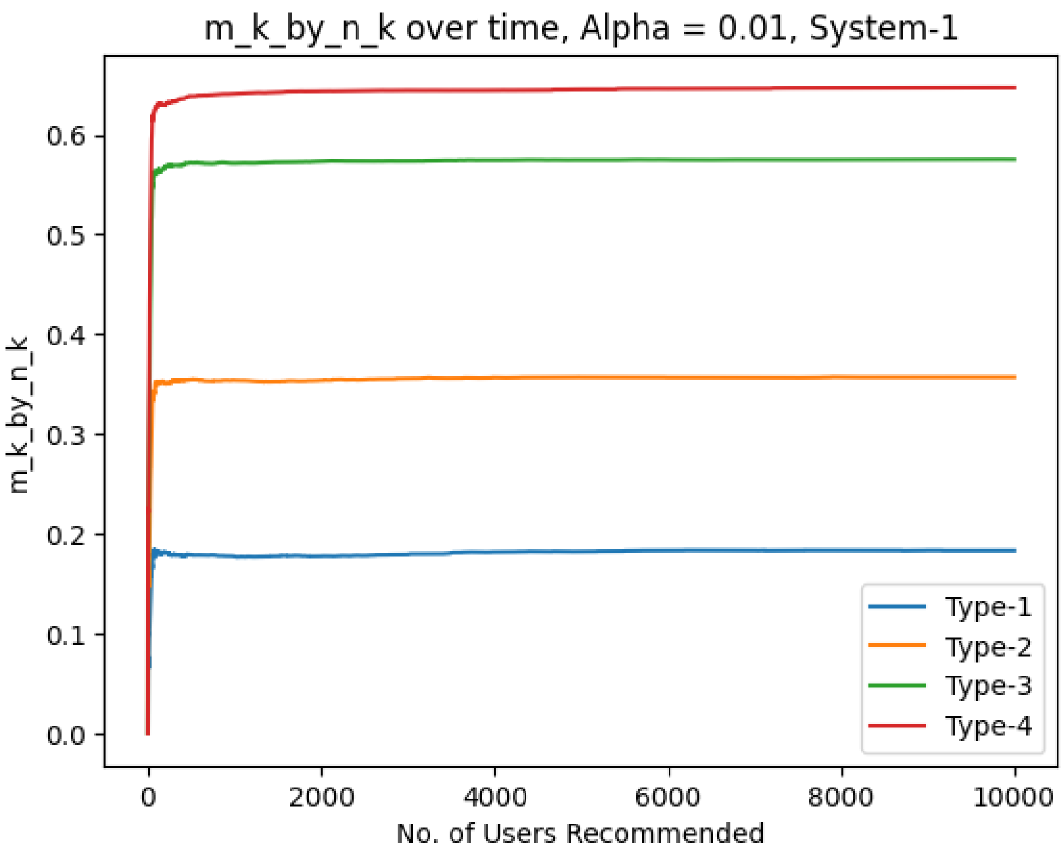


```
[506]: np.sum(plot_it[:, -2]) ## Total Revenue
```

```
[506]: 6453.203926346409
```

```
[454]: plot_it = (m_by_nk_avgd_001.T/count_p001).T
       for j in range(K):
           plt.plot(plot_it[j, :])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("m_k_by_n_k")
       plt.title('m_k_by_n_k over time, Alpha = 0.01, System-1')
       plot_it[:, -1]
```

```
[454]: array([0.18331361, 0.35715075, 0.57543826, 0.64748015])
```



[ ]:

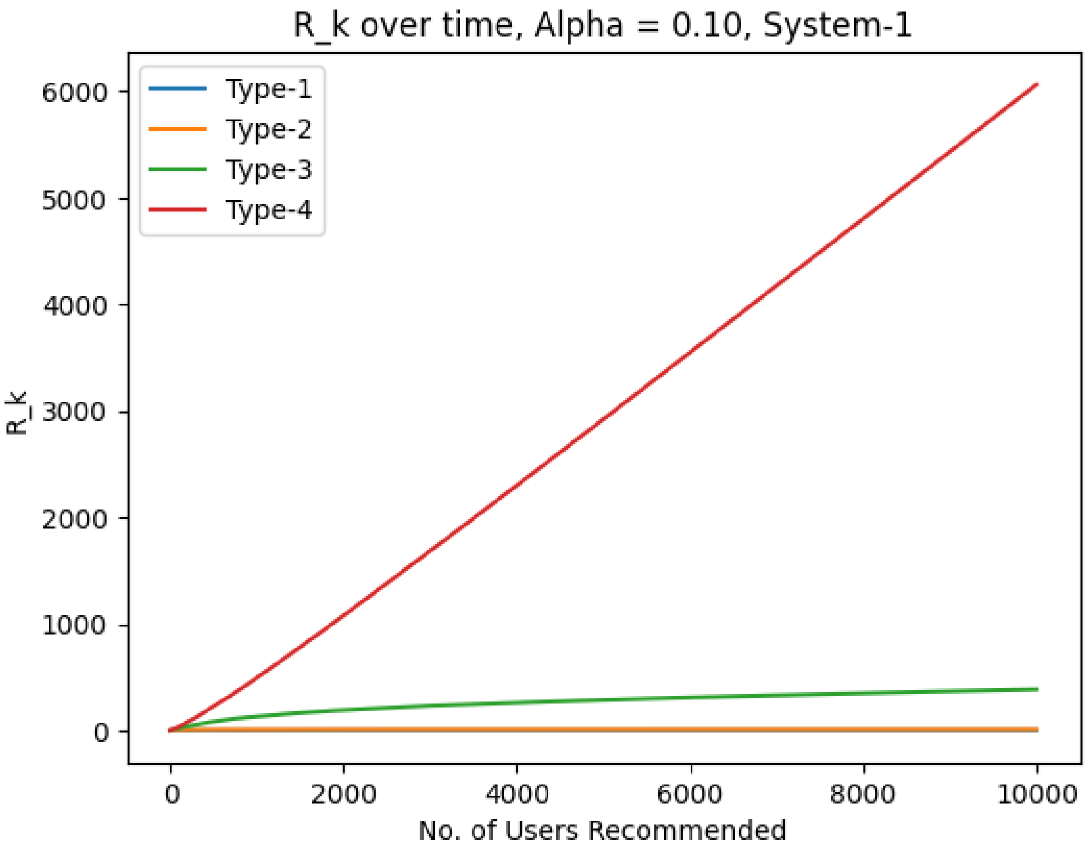
```
N = 10000
K = 4
ucb_p010 = np.zeros((K, N + 1))
num_iter = 1000
count_p010 = np.zeros(K)
rev_avgd_010 = np.zeros_like(ucb_p010)
m_by_nk_avgd_010 = np.zeros_like(ucb_p010)
a_k = [2, 2, 2, 2]

for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, 0.1)
    ucb_p010 += res
    rev_avgd_010 += rev_storer
    m_by_nk_avgd_010 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p010[j] += 1
    if i % 10 == 0:
        print(i)
```

[509]:

```
plot_it_3 = (rev_avgd_010.T/count_p010).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.10, System-1')

np.sum(plot_it_3[:, -2])
```

[509]:

```
6458.367861990006
```



[ ]:

```
plot_it_3 = (rev_avgd_010.T/count_p010).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.10, System-1')
```

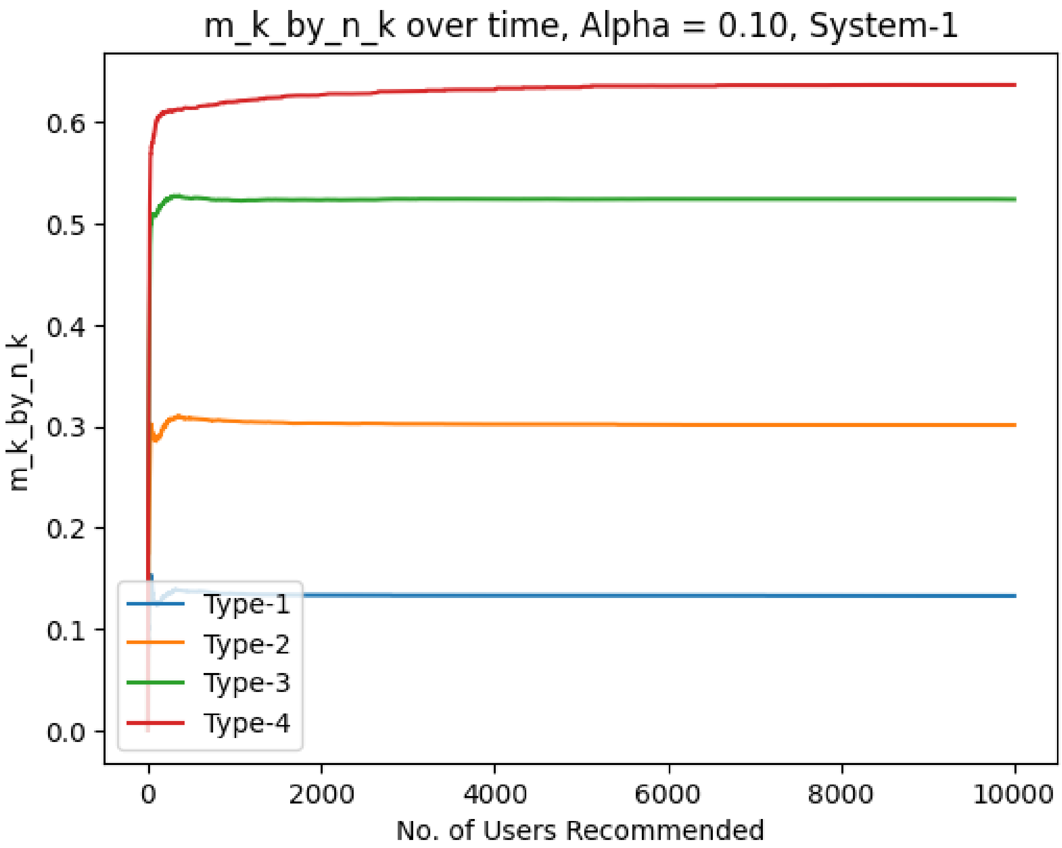
```
np.sum(plot_it_3[:, -2])
```

[510]:

```
plot_it = (m_by_nk_avgd_010.T/count_p010).T
for j in range(K):
    plt.plot(plot_it[j, :])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k_by_n_k")
plt.title('m_k_by_n_k over time, Alpha = 0.10, System-1')
plot_it[:, -1]
```

[510]:

```
array([0.13264819, 0.30122275, 0.52414134, 0.63686586])
```



[ ]:

```
N = 10000
K = 4
ucb_p005 = np.zeros((K, N + 1))
num_iter = 1000
count_p005 = np.zeros(K)
rev_avgd_005 = np.zeros_like(ucb_p005)
m_by_nk_avgd_005 = np.zeros_like(ucb_p005)
```

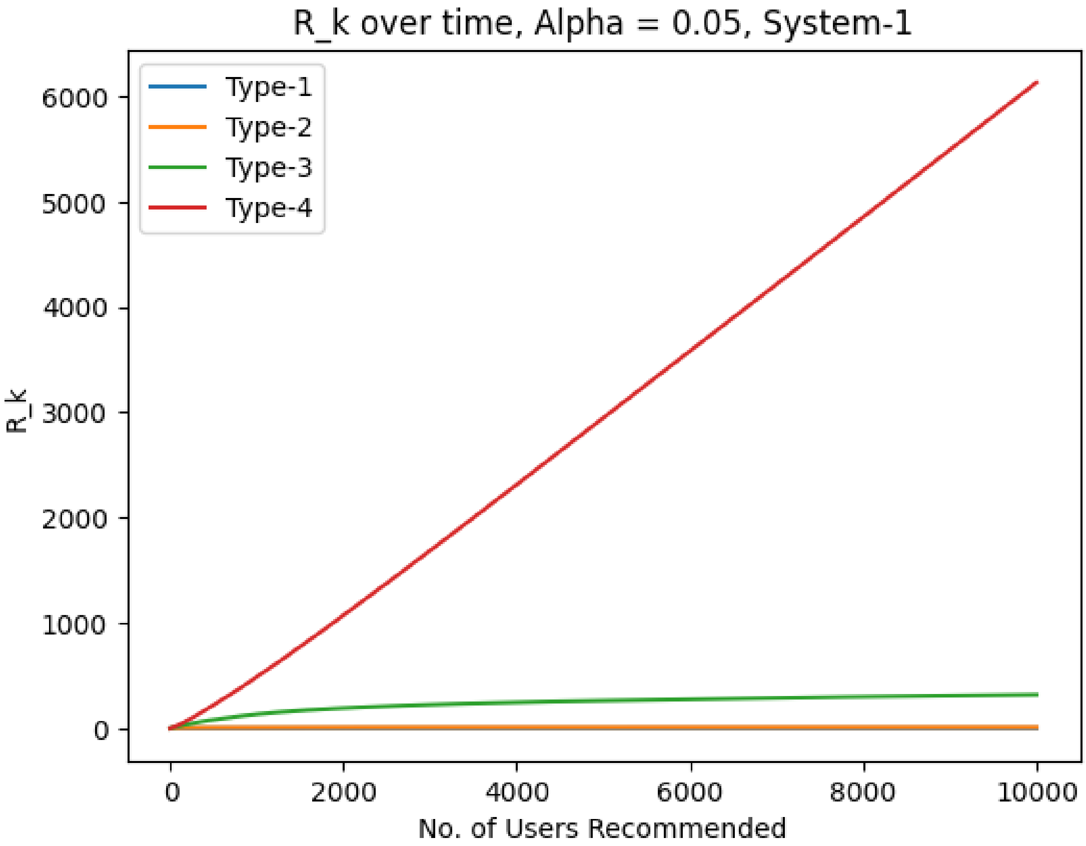
```
a_k = [2, 2, 2, 2]

for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, 0.05)
    ucb_p005 += res
    rev_avgd_005 += rev_storer
    m_by_nk_avgd_005 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005[j] += 1
    if i % 10 == 0:
        print(i)
```

[513]:

```
plot_it_3 = (rev_avgd_005.T/count_p005).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-1')

np.sum(plot_it_3[:, -2])
```

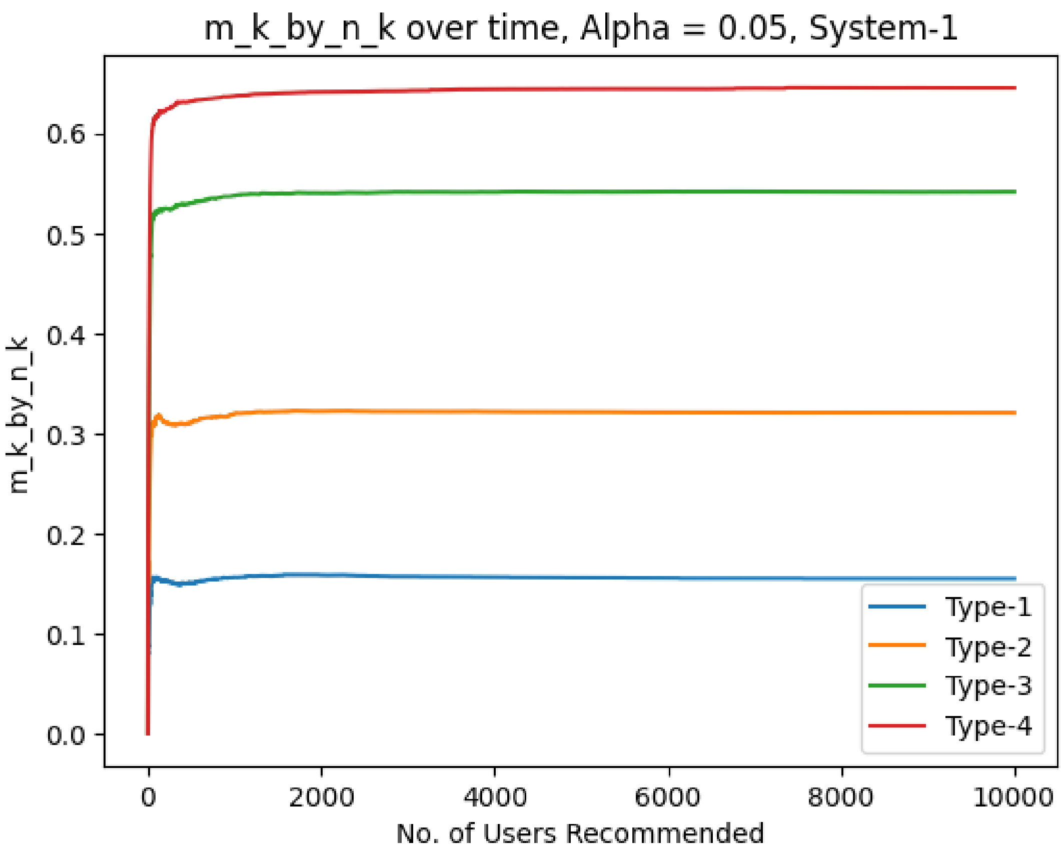
[513]:

```
6460.013029443621
```



```
[514]: plot_it = (m_by_nk_avgd_005.T/count_p005).T
       for j in range(K):
           plt.plot(plot_it[j, :])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("m_k_by_n_k")
       plt.title('m_k_by_n_k over time, Alpha = 0.05, System-1')
       plot_it[:, -1]
```

```
[514]: array([0.15468209, 0.32059725, 0.54166548, 0.64560754])
```



N = 1000

```
[ ]: N = 1000
     K = 4
     ucb_p001_1k = np.zeros((K, N + 1))
     num_iter = 1000
     count_p001_1k = np.zeros(K)
     rev_avgd_001_1k = np.zeros_like(ucb_p001_1k)
     m_by_nk_avgd_001_1k = np.zeros_like(ucb_p001_1k)
     a_k = [2, 2, 2, 2]
     p_k = [0.2, 0.4, 0.6, 0.65]

     for i in range(num_iter):
         res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, 0.01)
         ucb_p001_1k += res
         rev_avgd_001_1k += rev_storer
         m_by_nk_avgd_001_1k += m_by_n_storer
         # rev_avgd += rev_add/num_iter
         for j in range(K):
             if res[j, -1] != 0:
```

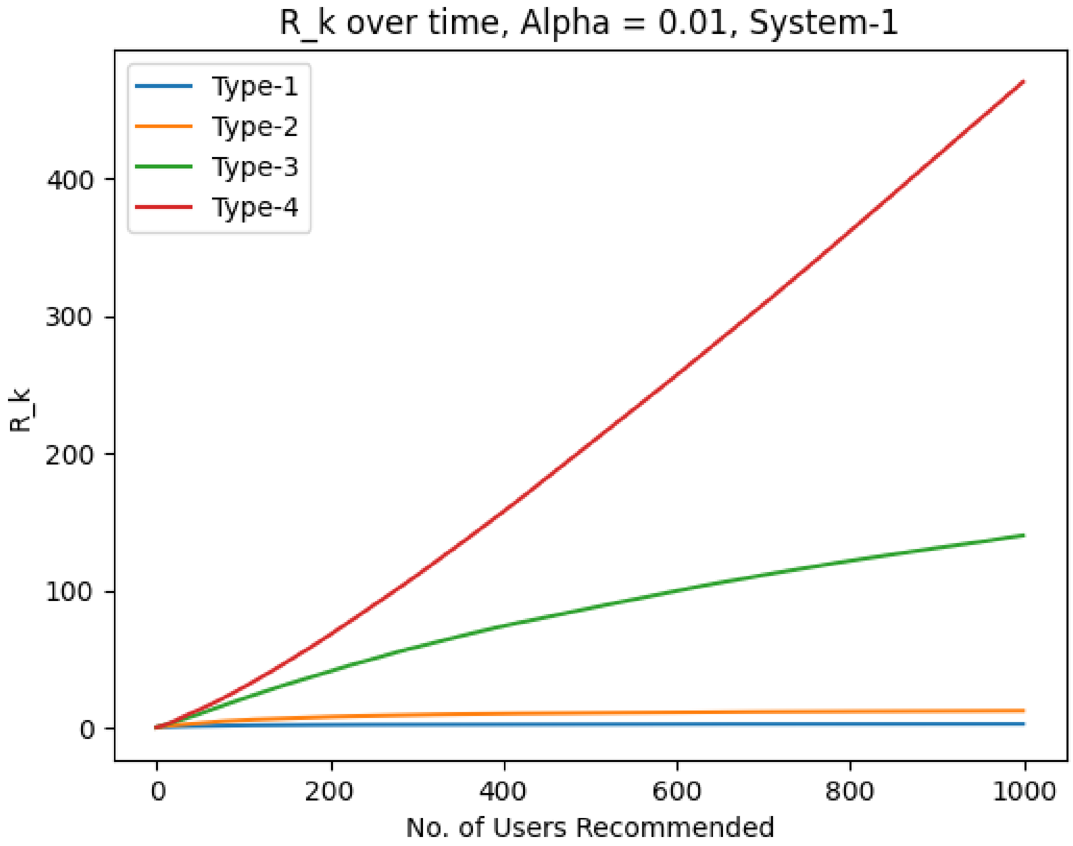
```
            count_p001_1k[j] += 1
    if i % 10 == 0:
        print(i)
```

[531]:

```
plot_it_3 = (rev_avgd_001_1k.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.01, System-1')

np.sum(plot_it_3[:, -2])
```

[531]:

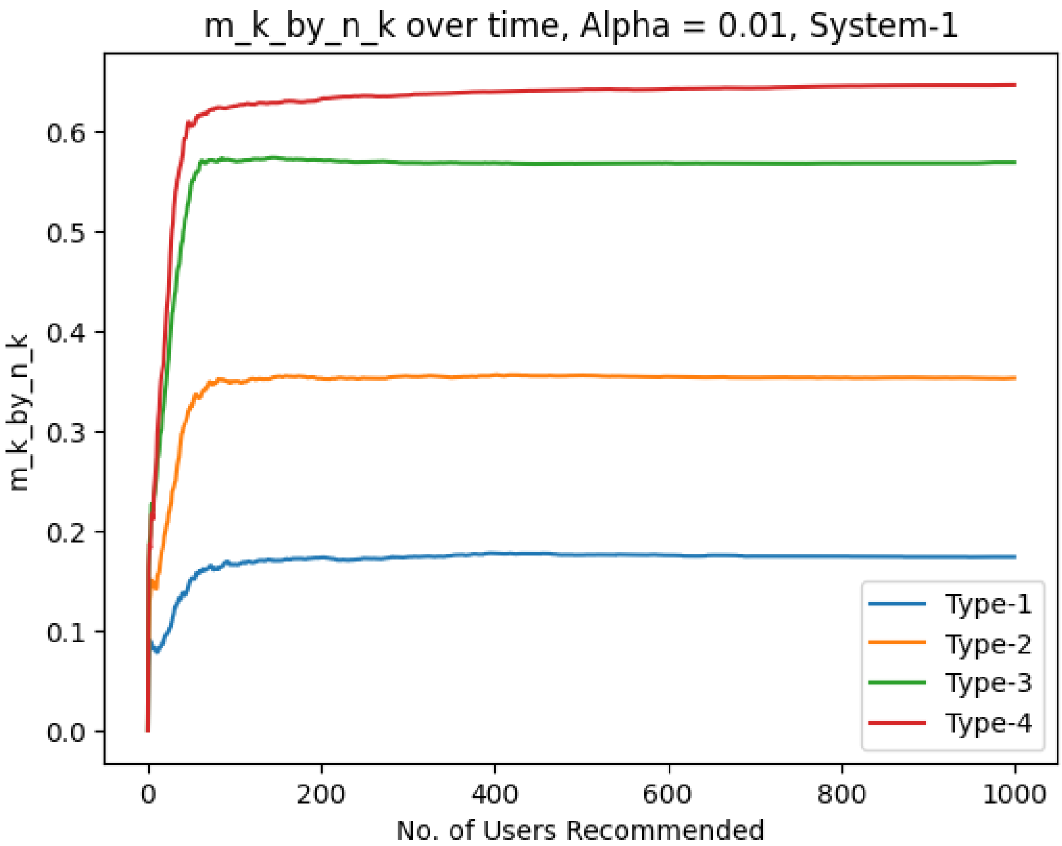
```
625.0826963331743
```



[530]:

```
plot_it = (m_by_nk_avgd_001_1k.T/num_iter).T
for j in range(K):
    plt.plot(plot_it[j, :])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k_by_n_k")
plt.title('m_k_by_n_k over time, Alpha = 0.01, System-1')
plot_it[:, -1]
```

```
[530]: array([0.17379813, 0.35302485, 0.56931958, 0.6467305 ])
```

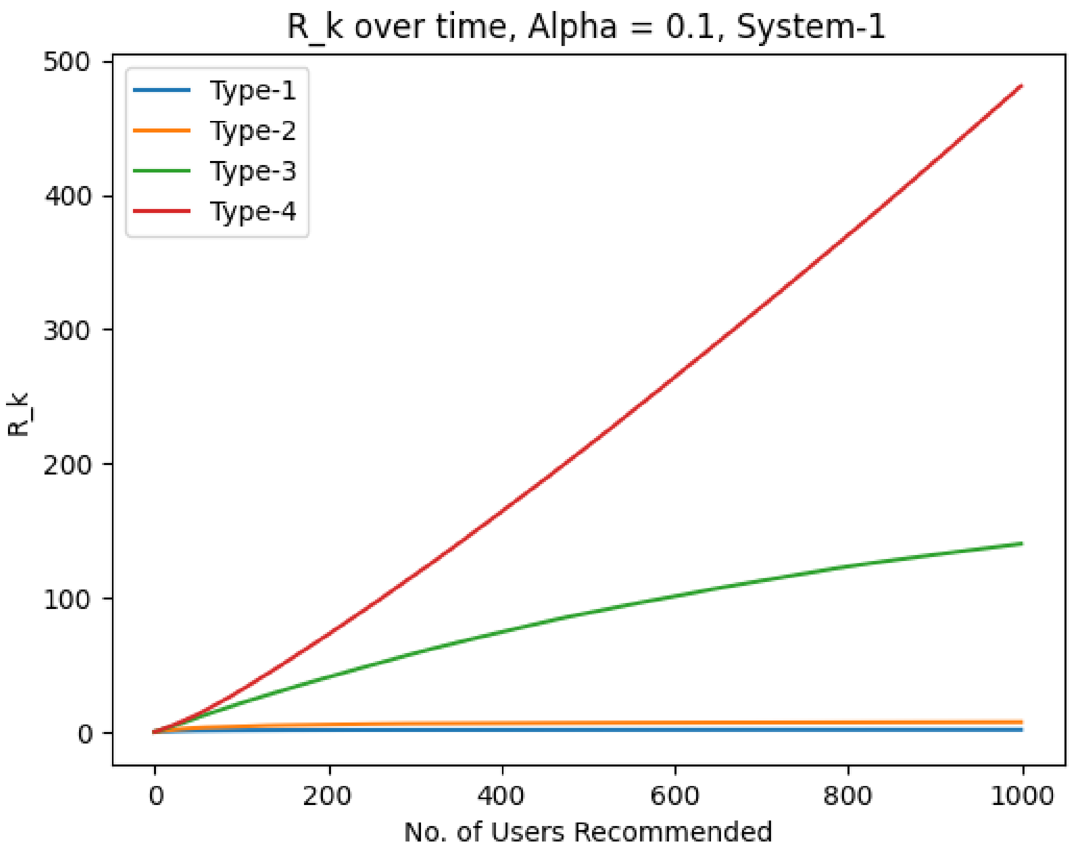


```
[ ]: N = 1000
     K = 4
     alpha = 0.1
     ucb_p01_1k = np.zeros((K, N + 1))
     num_iter = 1000
     count_p01_1k = np.zeros(K)
     rev_avgd_01_1k = np.zeros_like(ucb_p01_1k)
     m_by_nk_avgd_01_1k = np.zeros_like(ucb_p01_1k)
     a_k = [2, 2, 2, 2]
     p_k = [0.2, 0.4, 0.6, 0.65]

     for i in range(num_iter):
         res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, alpha)
```

```
    ucb_p01_1k += res
    rev_avgd_01_1k += rev_storer
    m_by_nk_avgd_01_1k += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p01_1k[j] += 1
    if i % 10 == 0:
        print(i)
```

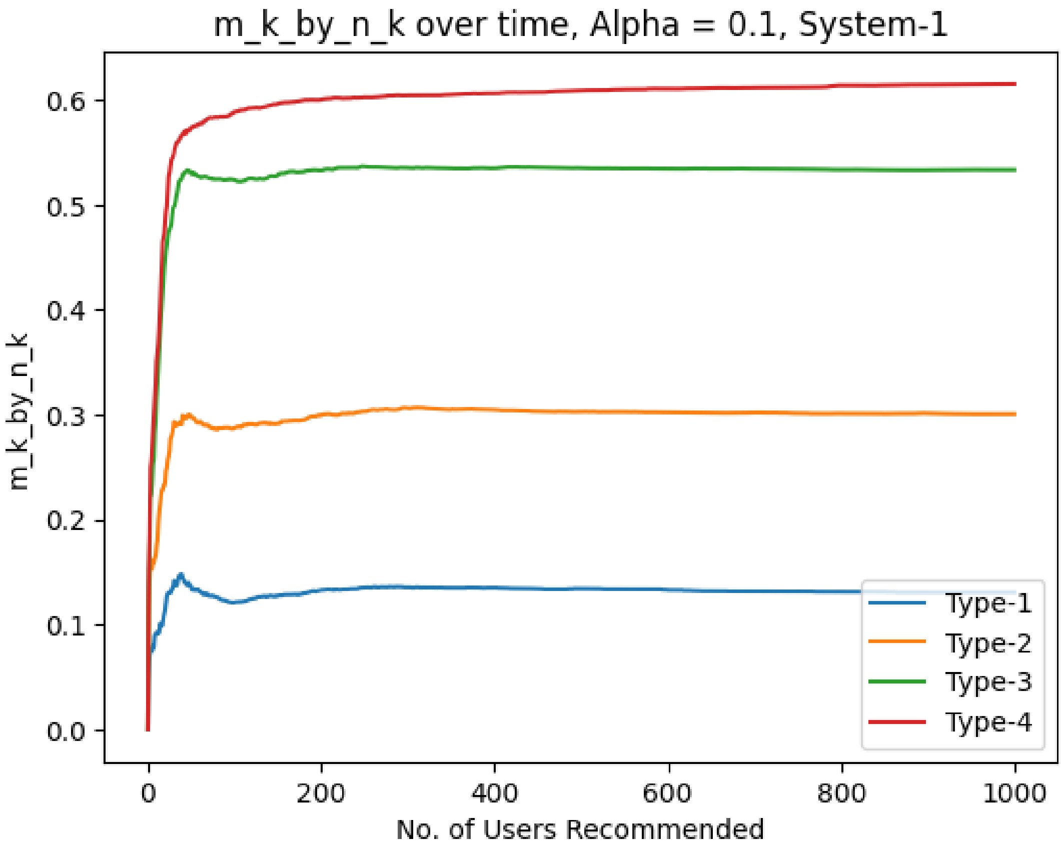
```
[535]: plot_it_3 = (rev_avgd_01_1k.T/num_iter).T
       for j in range(K):
           plt.plot(plot_it_3[j, :-1])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("R_k")
       plt.title('R_k over time, Alpha = 0.1, System-1')

       np.sum(plot_it_3[:, -2])
```

```
[535]: 629.9157937353215
```



```
[537]: plot_it = (m_by_nk_avgd_01_1k.T/num_iter).T
       for j in range(K):
           plt.plot(plot_it[j, :])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("m_k_by_n_k")
       plt.title('m_k_by_n_k over time, Alpha = 0.1, System-1')
       plot_it[:, -1]
```

```
[537]: array([0.13056207, 0.30054811, 0.53332349, 0.61517836])
```



```
[ ]: N = 1000
     K = 4
     alpha = 0.05
     ucb_p005_1k = np.zeros((K, N + 1))
     num_iter = 1000
     count_p005_1k = np.zeros(K)
     rev_avgd_005_1k = np.zeros_like(ucb_p005_1k)
```

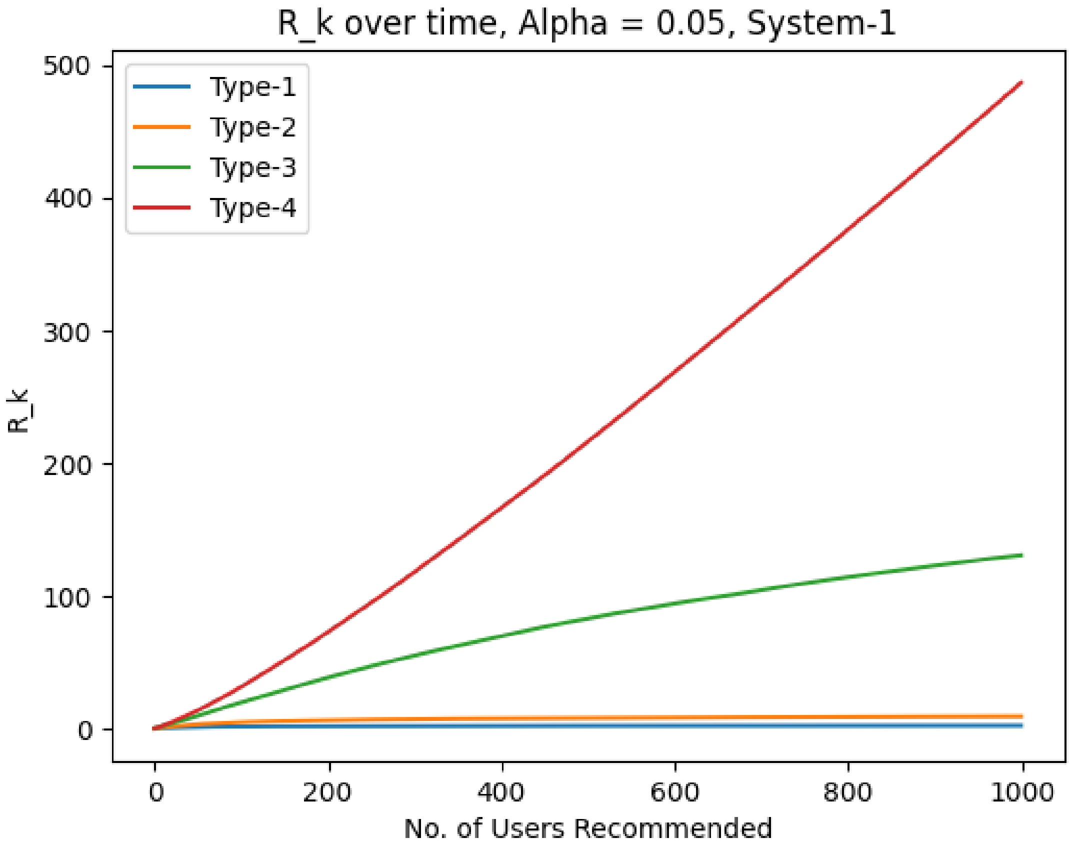
```
m_by_nk_avgd_005_1k = np.zeros_like(ucb_p005_1k)
a_k = [2, 2, 2, 2]
p_k = [0.2, 0.4, 0.6, 0.65]

for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_prob(N, p_k, a_k, 4, alpha)
    ucb_p005_1k += res
    rev_avgd_005_1k += rev_storer
    m_by_nk_avgd_005_1k += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005_1k[j] += 1
    if i % 10 == 0:
        print(i)
```

[540]:

```
plot_it_3 = (rev_avgd_005_1k.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-1')

np.sum(plot_it_3[:, -2])
```

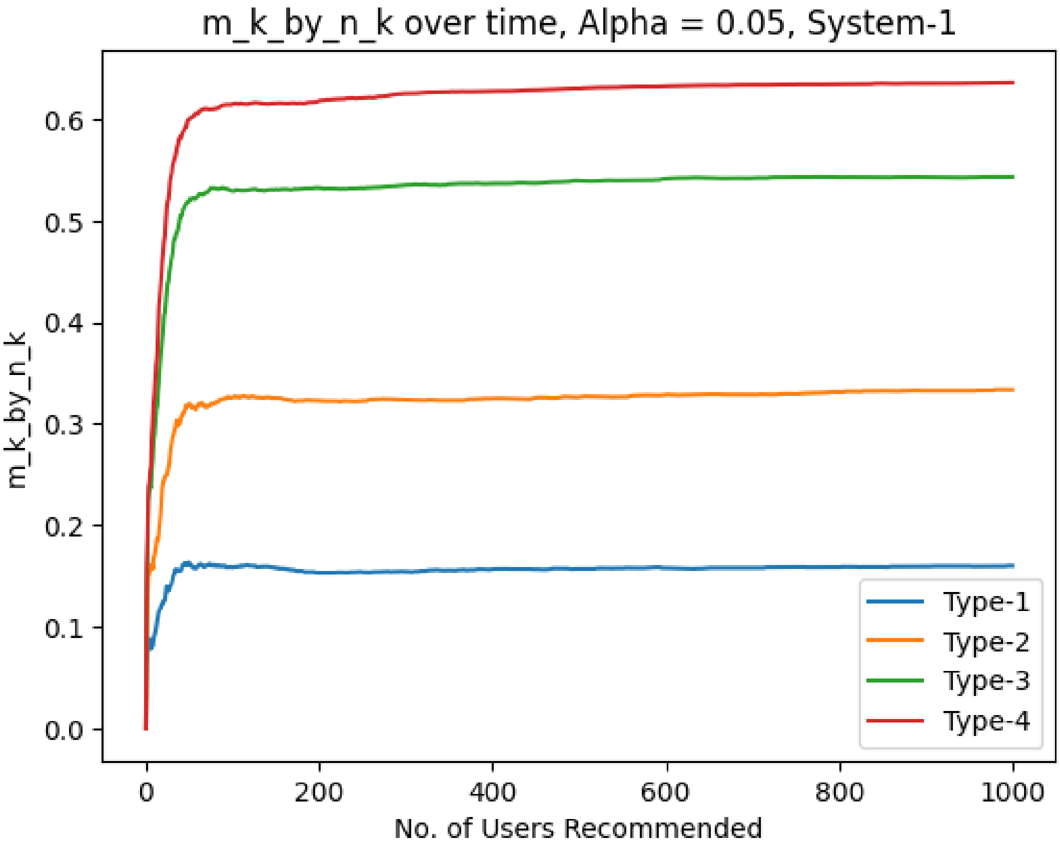
[540]:

```
628.1206769063334
```



```
[541]: plot_it = (m_by_nk_avgd_005_1k.T/num_iter).T
       for j in range(K):
           plt.plot(plot_it[j, :])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("m_k_by_n_k")
       plt.title('m_k_by_n_k over time, Alpha = 0.05, System-1')
       plot_it[:, -1]
```

```
[541]: array([0.16005569, 0.33352896, 0.5432678 , 0.6361051 ])
```



0.2.2 Modifying the algorithm for System 2 and 3

[415]:

```
def ucb_rev_shenai(N, p_k, a_k, rev_up_bound, K, alpha):
    ucb_rev = 20 * np.ones((K, N + 1))
    n_k_s = np.zeros_like(ucb_rev)
    m_k_s = np.zeros_like(ucb_rev)
    m_by_n_s = np.zeros_like(ucb_rev)
    r_by_n_s = np.zeros_like(ucb_rev)
    rev_total = np.zeros(K)
    rev_storer = np.zeros_like(ucb_rev)

    for i in range(N - 1):
        max_rev = np.max(ucb_rev[:, i])
        poss_ind = np.where(ucb_rev[:, i] == max_rev)
        sel_ind = np.random.choice(poss_ind[0])
        n_k_s[sel_ind, (i + 1):] += 1
        if np.random.rand() <= p_k[sel_ind]:
            m_k_s[sel_ind, (i + 1):] += 1
            m_by_n_s[sel_ind, (i + 1):] = m_k_s[sel_ind, (i + 1)] /␣
↪n_k_s[sel_ind, (i + 1)]
            rev = np.random.uniform(low = 0.0, high = a_k[sel_ind])
            rev_total[sel_ind] += rev
        r_by_n_s[sel_ind, (i + 1):] = rev_total[sel_ind]/n_k_s[sel_ind, i + 1]
        ucb_rev[sel_ind, (i + 1):] = r_by_n_s[sel_ind, (i + 1)] + (-np.
↪log(alpha) * rev_up_bound ** 2/(2 * n_k_s[sel_ind, (i + 1)]))** 0.5
        rev_storer[:, (i + 1)] = rev_total
    return ucb_rev, m_by_n_s, rev_storer[:, :N + 1]
```

0.2.3 System 3

N = 10000

[ ]:

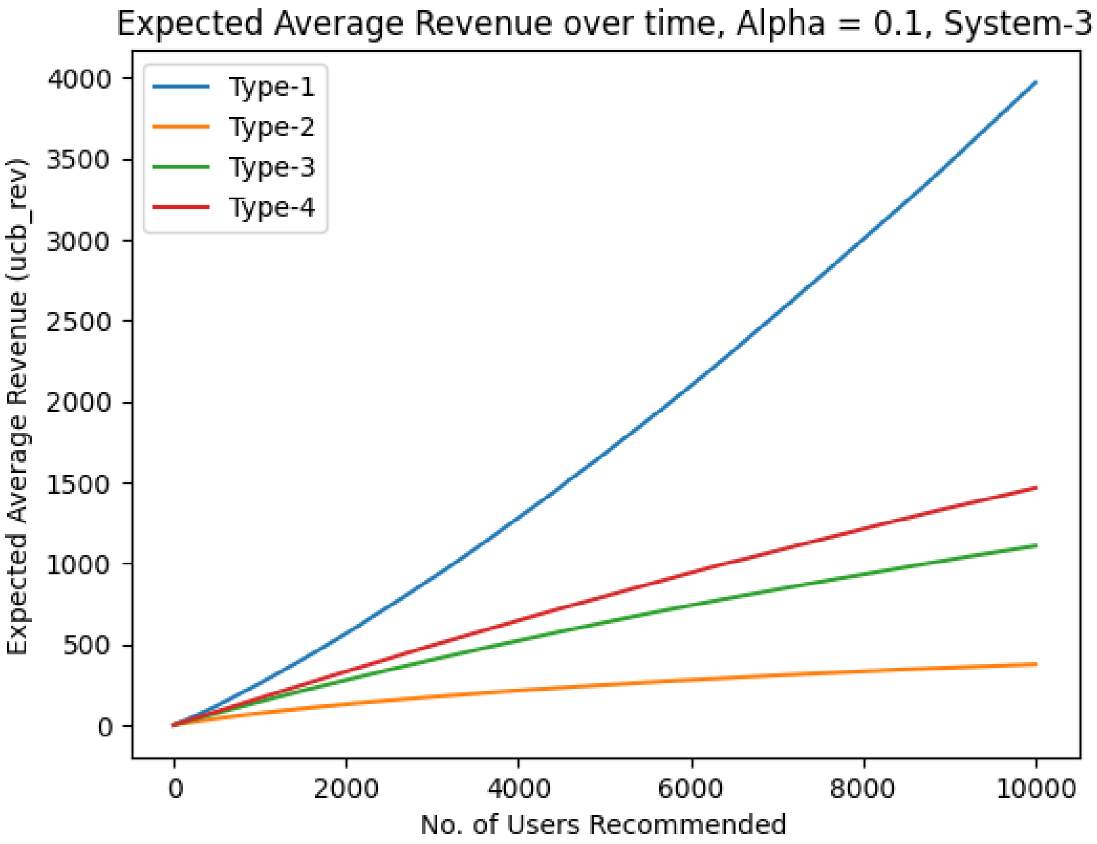
```
N = 10000
K = 4
ucb_p01_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p01_3 = np.zeros(K)
rev_avgd_01_3 = np.zeros_like(ucb_p01_3)
m_by_nk_avgd_01_3 = np.zeros_like(ucb_p01_3)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.1
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p01_3 += res
    rev_avgd_01_3 += rev_storer
    m_by_nk_avgd_01_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p01_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[495]:

```
plot_it = (rev_avgd_01_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("Expected Average Revenue (ucb_rev)")
plt.title('Expected Average Revenue over time, Alpha = 0.1, System-3')

np.sum(plot_it[:, -2])
```

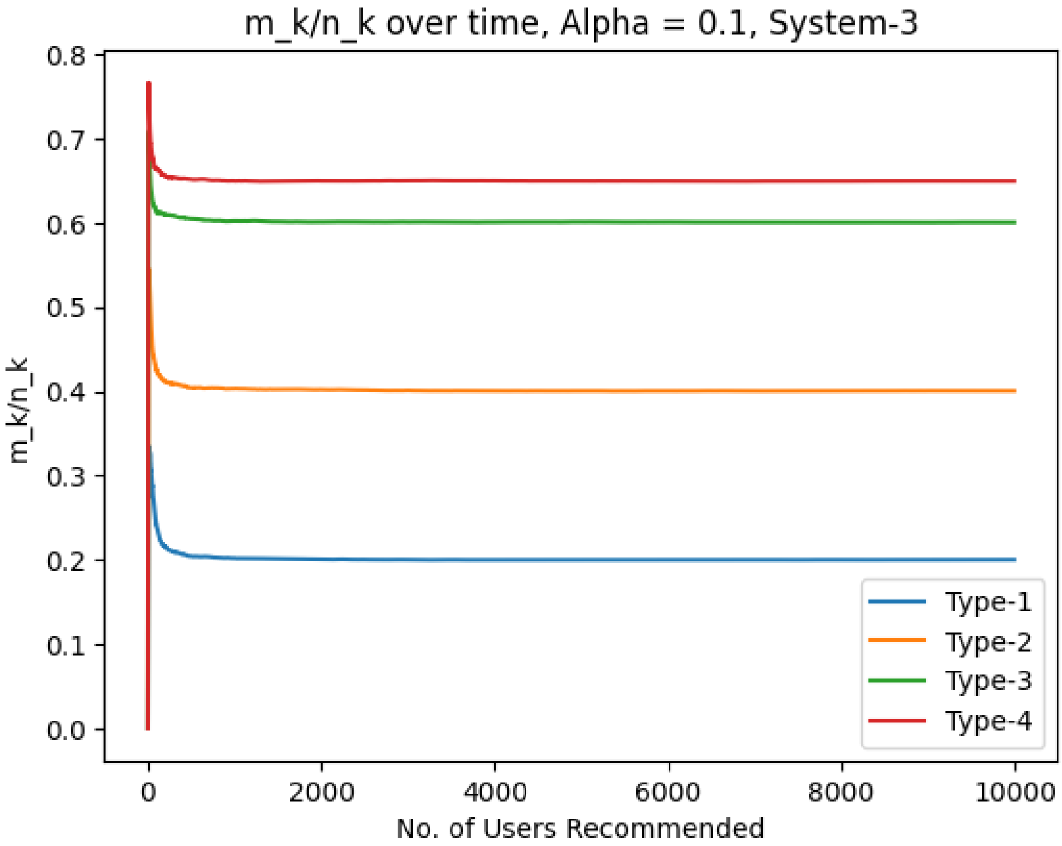
[495]:

```
6917.142377072158
```



```
[493]: plot_it_mn = (m_by_nk_avgd_01_3.T/num_iter).T
       for j in range(K):
           plt.plot(plot_it_mn[j, :N])
       plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
       plt.xlabel("No. of Users Recommended")
       plt.ylabel("m_k/n_k")
       plt.title('m_k/n_k over time, Alpha = 0.1, System-3')
       plot_it_mn[:, -1]
```

```
[493]: array([0.20003565, 0.40058125, 0.60049756, 0.64961894])
```



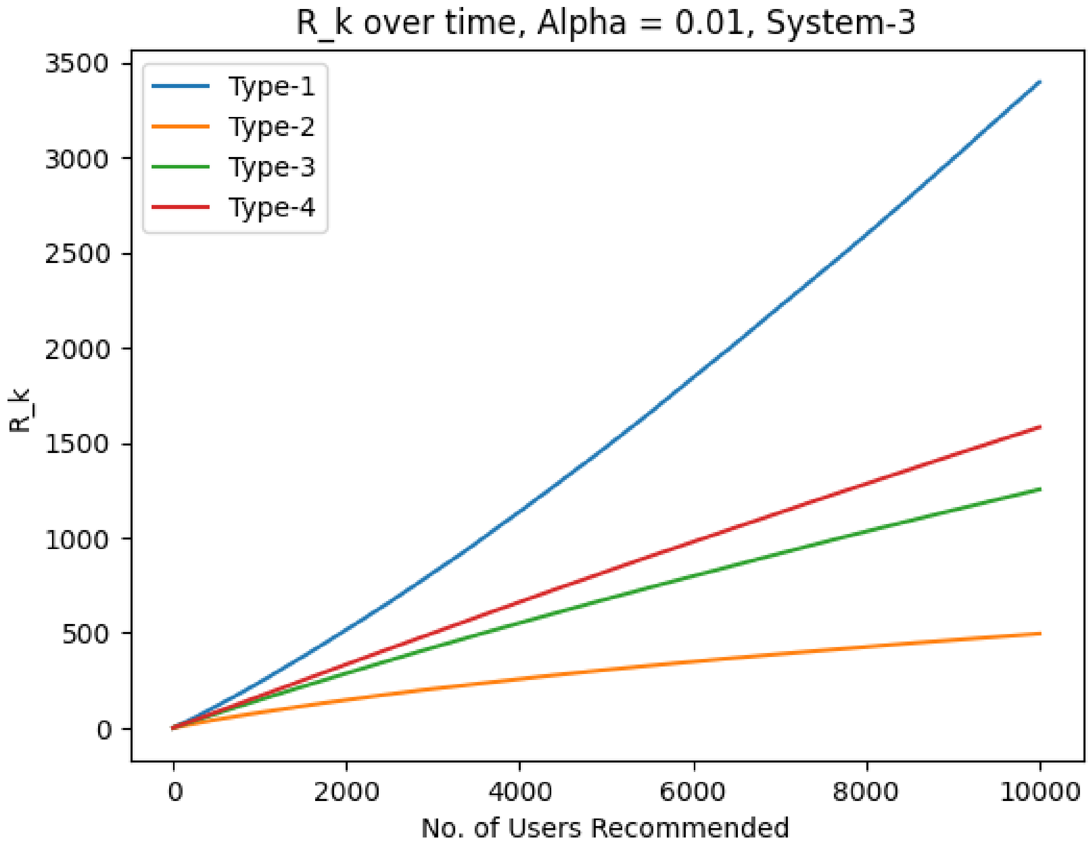
[ ]:
```
N = 10000
K = 4
ucb_p001_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p001_3 = np.zeros(K)
rev_avgd_001_3 = np.zeros_like(ucb_p001_3)
m_by_nk_avgd_001_3 = np.zeros_like(ucb_p001_3)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.01
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p001_3 += res
    rev_avgd_001_3 += rev_storer
    m_by_nk_avgd_001_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p001_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[561]:

```
plot_it_3 = (rev_avgd_001_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.01, System-3')

np.sum(plot_it_3[:, -2])
```

[561]:

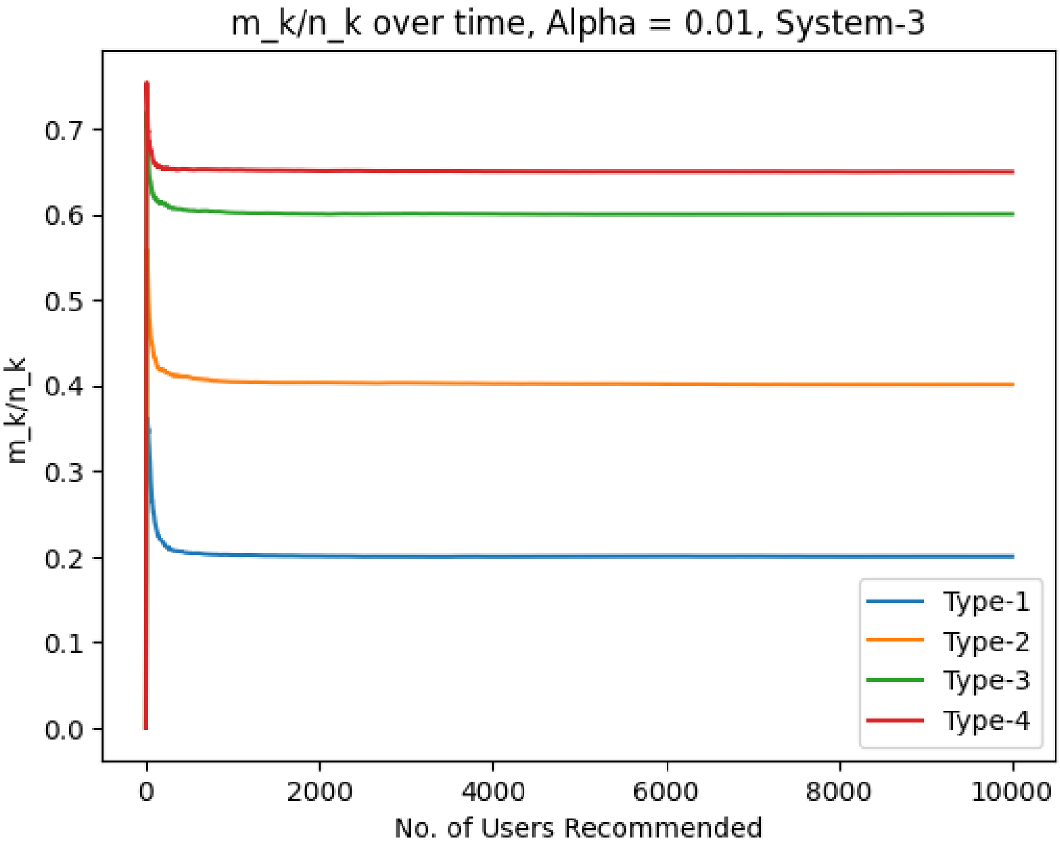
```
6727.801553507009
```



[563]:

```
plot_it_mn = (m_by_nk_avgd_001_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_mn[j, :])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.01, System-3')
plot_it_mn[:, -1]
```

```
[563]: array([0.20020646, 0.40132862, 0.60056718, 0.64998628])
```



```
[462]: N = 10000
K = 4
ucb_p001_trial = np.zeros((K, N + 1))
a_k = [2, 2.5, 2.5, 3]
num_iter = 1000
count_p001_trial = np.zeros(K)
rev_avgd_001_trial = np.zeros_like(ucb_p001_trial)
m_by_nk_avgd_001_trial = np.zeros_like(ucb_p001_trial)
p_k = [0.2, 0.4, 0.6, 0.65]

for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, 0.01)
    ucb_p001_trial += res
    rev_avgd_001_trial += rev_storer
    m_by_nk_avgd_001_trial += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p001_trial[j] += 1
    if i % 10 == 0:
        print(i)
```

[ ]:

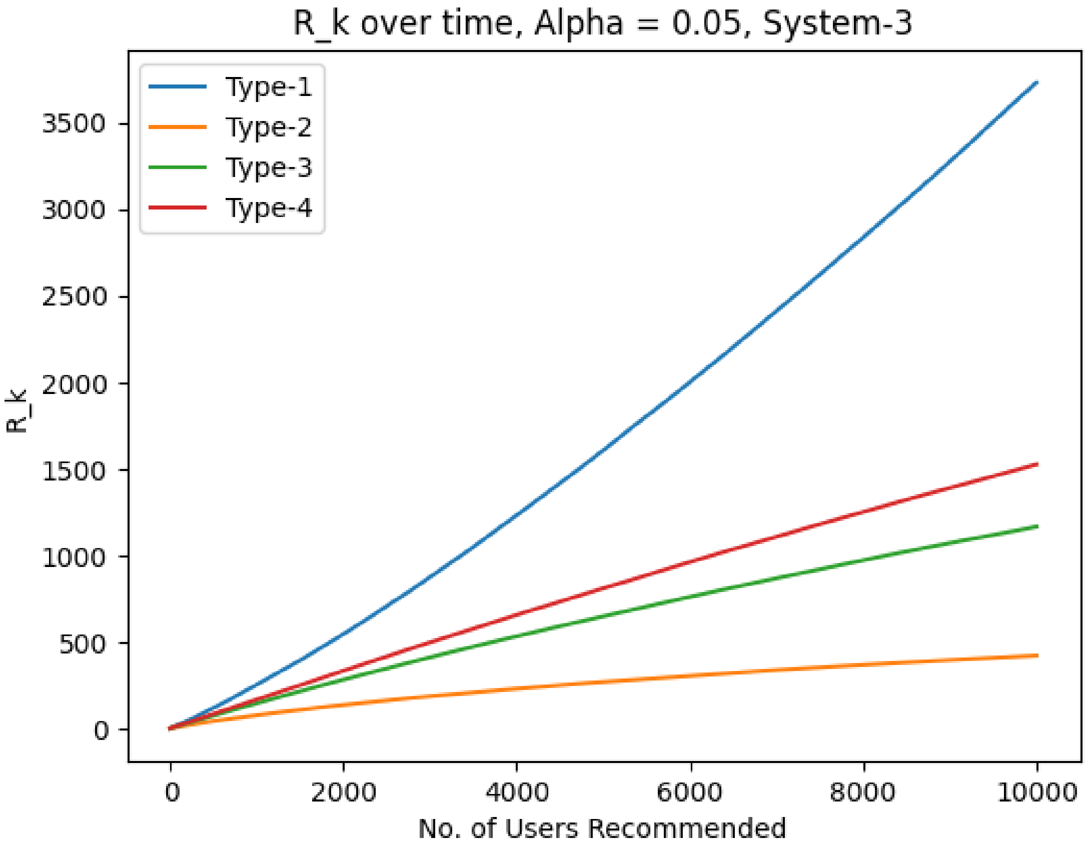
```
N = 10000
K = 4
ucb_p005_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p005_3 = np.zeros(K)
rev_avgd_005_3 = np.zeros_like(ucb_p005_3)
m_by_nk_avgd_005_3 = np.zeros_like(ucb_p005_3)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.05
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p005_3 += res
    rev_avgd_005_3 += rev_storer
    m_by_nk_avgd_005_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[503]:

```
plot_it_3 = (rev_avgd_005_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-3')

np.sum(plot_it_3[:, -2])
```

[503]:

```
6840.052325195102
```

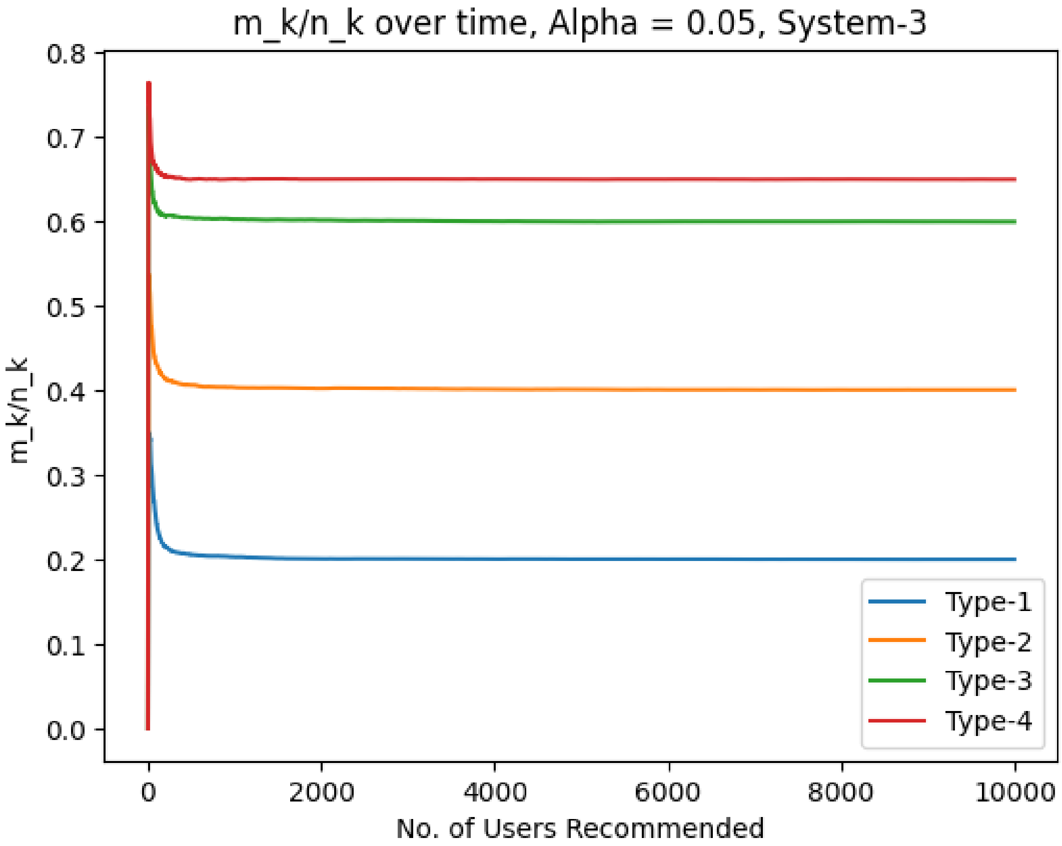


[504]:

```
plot_it_3 = (m_by_nk_avgd_005_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.05, System-3')

plot_it_3[:, -2]
```

[504]:

```
array([0.19989562, 0.40072796, 0.59970412, 0.64982323])
```

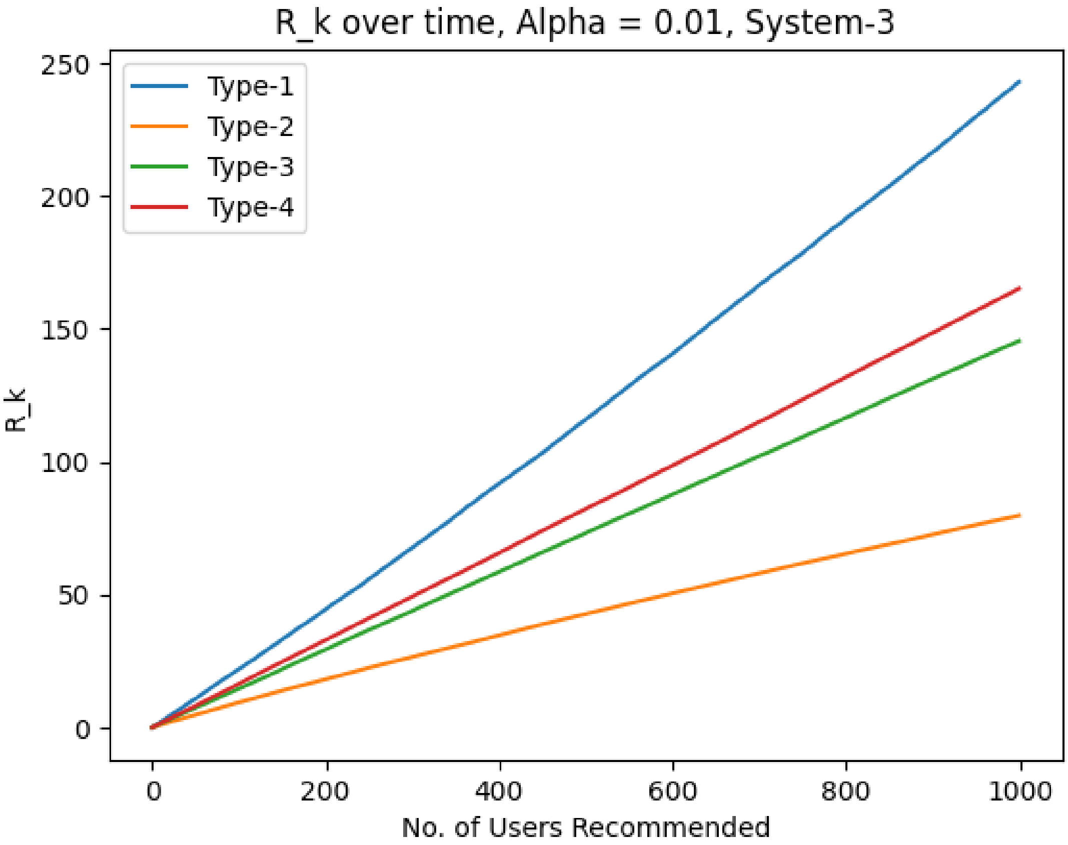


[ ]:

```
N = 1000
K = 4
ucb_p001_1k_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p001_1k_3 = np.zeros(K)
rev_avgd_001_1k_3 = np.zeros_like(ucb_p001_1k_3)
m_by_nk_avgd_001_1k_3 = np.zeros_like(ucb_p001_1k_3)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.01
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p001_1k_3 += res
    rev_avgd_001_1k_3 += rev_storer
    m_by_nk_avgd_001_1k_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p001_1k_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[544]:
```
plot_it_3 = (rev_avgd_001_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.01, System-3')
np.sum(plot_it_3[:, -2])
```

[544]:
```
633.3200935719287
```

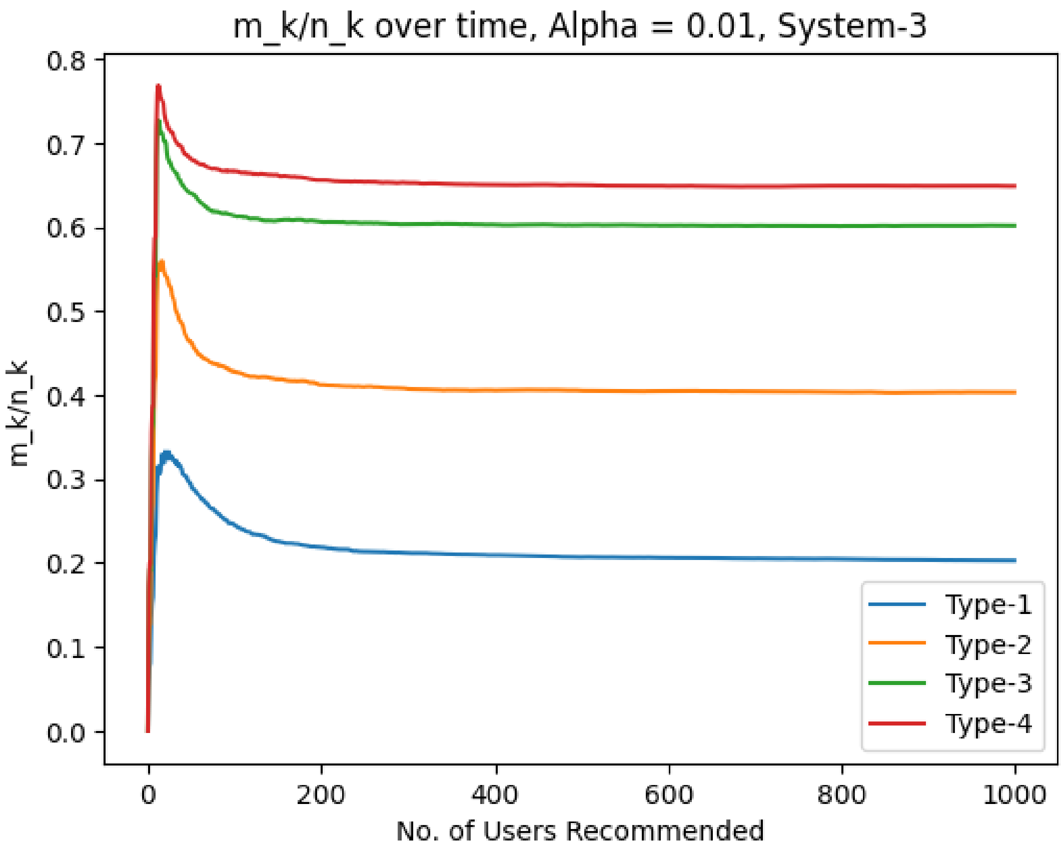


[545]:
```
plot_it_3 = (m_by_nk_avgd_001_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.01, System-3')
```

```
plot_it_3[:, -2]
```

[545]:

```
array([0.20281767, 0.40316404, 0.60173805, 0.64871968])
```



[ ]:

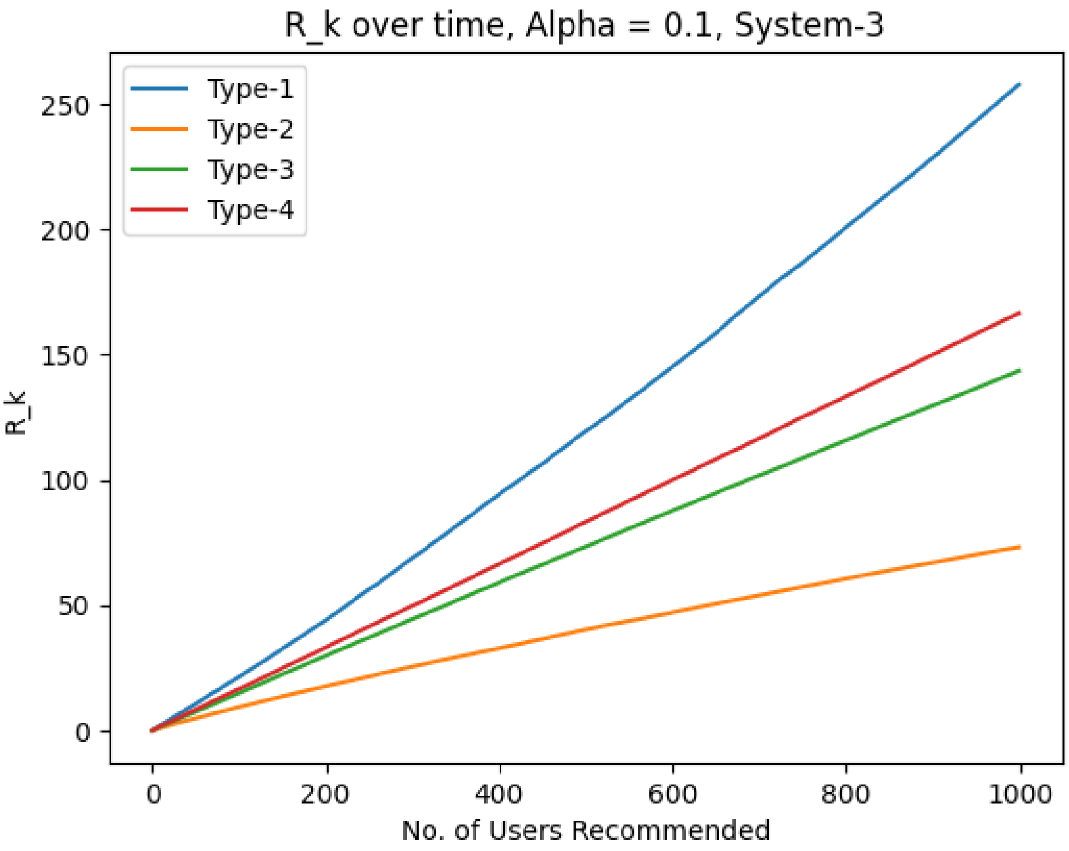
```
N = 1000
K = 4
ucb_p01_1k_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p01_1k_3 = np.zeros(K)
rev_avgd_01_1k_3 = np.zeros_like(ucb_p01_1k_3)
m_by_nk_avgd_01_1k_3 = np.zeros_like(ucb_p01_1k_3)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.1
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p01_1k_3 += res
    rev_avgd_01_1k_3 += rev_storer
    m_by_nk_avgd_01_1k_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p01_1k_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[552]:

```
plot_it_3 = (rev_avgd_01_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.1, System-3')
np.sum(plot_it_3[:, -2])
```

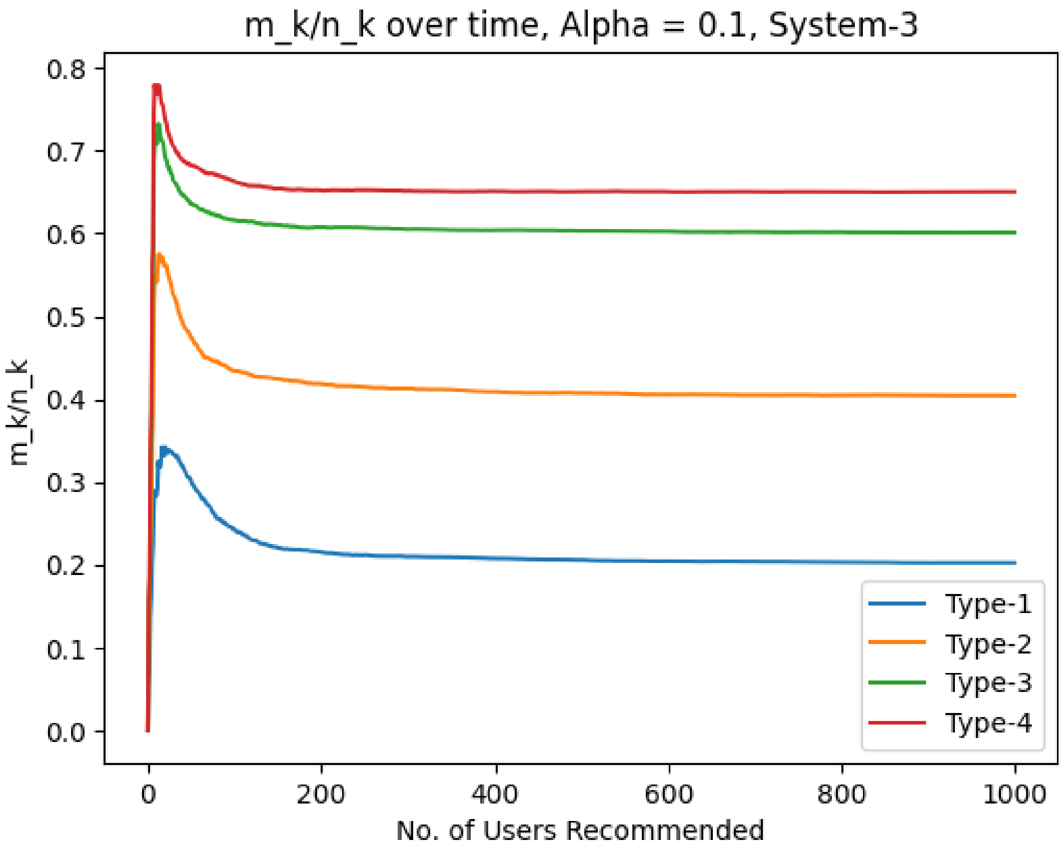
[552]:

```
640.7724190392514
```



[553]:
```
plot_it_3 = (m_by_nk_avgd_01_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.1, System-3')

plot_it_3[:, -2]
```

[553]:
```
array([0.20226618, 0.40415141, 0.60103357, 0.65042651])
```



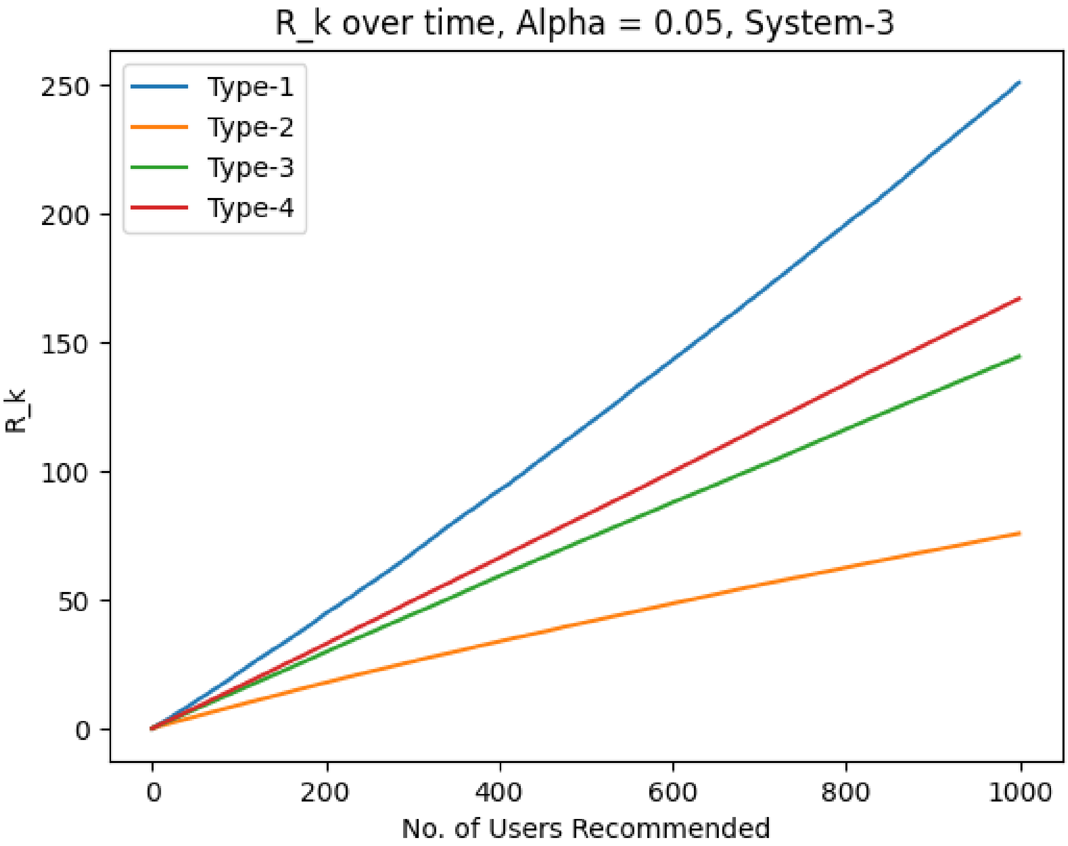
[ ]:
```
N = 1000
K = 4
ucb_p005_1k_3 = np.zeros((K, N + 1))
a_k = [8, 2, 2, 2]
num_iter = 1000
count_p005_1k_3 = np.zeros(K)
rev_avgd_005_1k_3 = np.zeros_like(ucb_p005_1k_3)
m_by_nk_avgd_005_1k_3 = np.zeros_like(ucb_p005_1k_3)
```

```
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.05
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p005_1k_3 += res
    rev_avgd_005_1k_3 += rev_storer
    m_by_nk_avgd_005_1k_3 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005_1k_3[j] += 1
    if i % 10 == 0:
        print(i)
```

[557]:

```
plot_it_3 = (rev_avgd_005_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-3')
np.sum(plot_it_3[:, -2])
```

```
[557]: 637.3876827442376
```

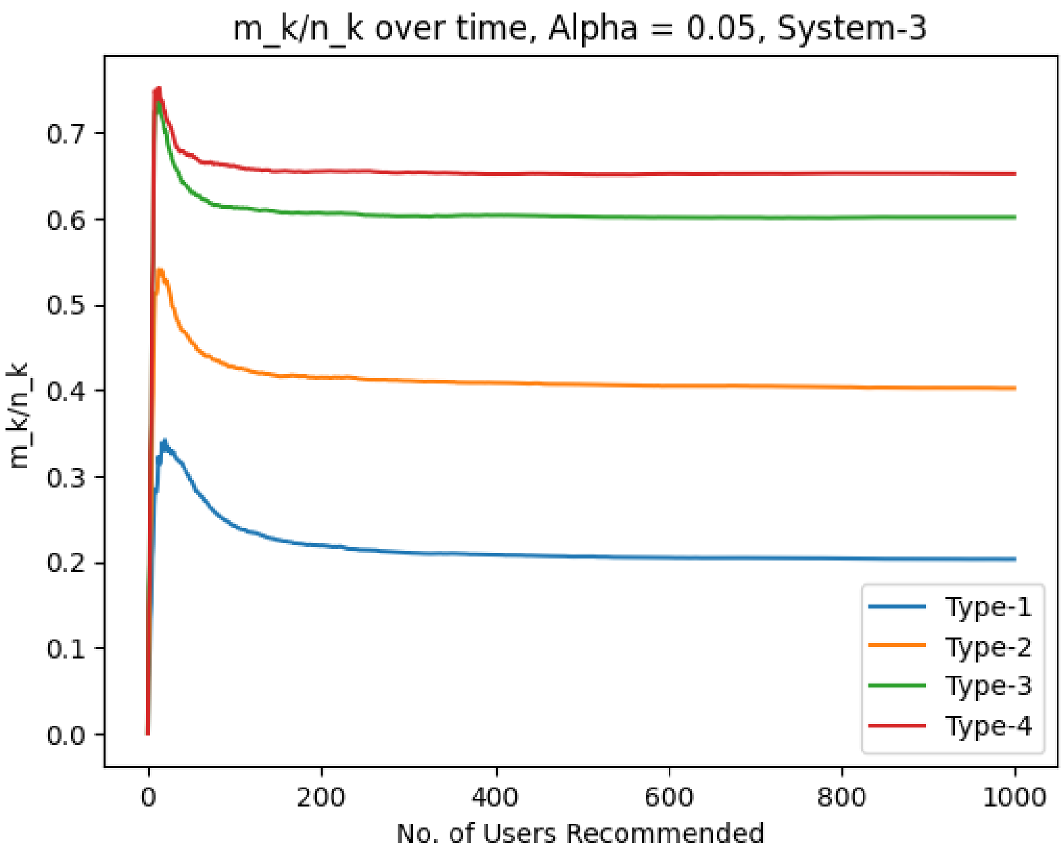


[558]:

```
plot_it_3 = (m_by_nk_avgd_005_1k_3.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.05, System-3')

plot_it_3[:, -2]
```

[558]:

```
array([0.20276187, 0.40192342, 0.60109036, 0.65152031])
```



0.2.4 System-2

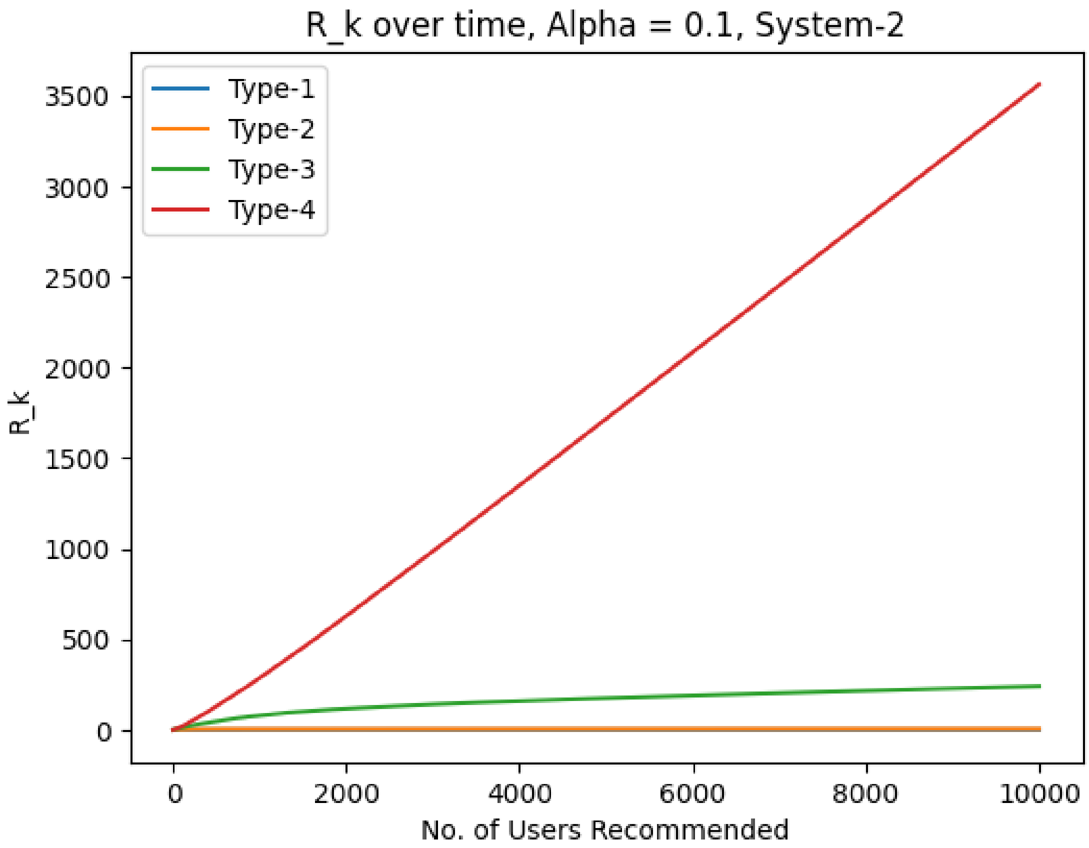
N = 10000

[ ]:

```
N = 10000
K = 4
ucb_p01_trial = np.zeros((K, N + 1))
a_k = [2, 2.5, 2.5, 3]
num_iter = 1000
count_p01_trial = np.zeros(K)
rev_avgd_01_trial = np.zeros_like(ucb_p01_trial)
m_by_nk_avgd_01_trial = np.zeros_like(ucb_p01_trial)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.1
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p01_trial += res
    rev_avgd_01_trial += rev_storer
    m_by_nk_avgd_01_trial += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p01_trial[j] += 1
    if i % 10 == 0:
        print(i)
```

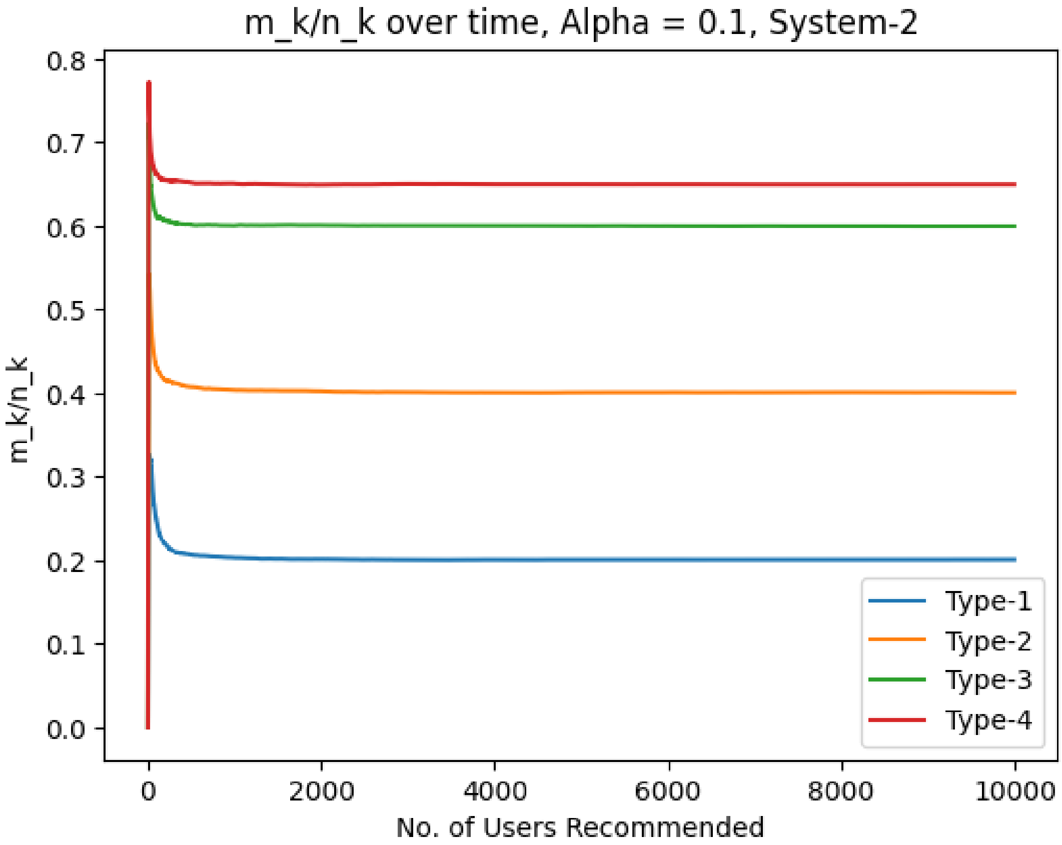
[475]:
```
plot_it_3 = (rev_avgd_010.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.1, System-2')

np.sum(plot_it_3[:, -2])
```

[475]:
```
3808.8751555479444
```



[476]:
```
plot_it_3 = (m_by_nk_avgd_001_trial.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.1, System-2')

plot_it_3[:, -2]
```

```
[476]: array([0.20090044, 0.40057571, 0.59994432, 0.65001608])
```



```
[ ]: N = 10000
     K = 4
     ucb_p005_trial = np.zeros((K, N + 1))
     a_k = [2, 2.5, 2.5, 3]
     num_iter = 1000
     count_p005_trial = np.zeros(K)
     rev_avgd_005_trial = np.zeros_like(ucb_p005_trial)
     m_by_nk_avgd_005_trial = np.zeros_like(ucb_p005_trial)
     p_k = [0.2, 0.4, 0.6, 0.65]
     alpha = 0.05
     for i in range(num_iter):
```

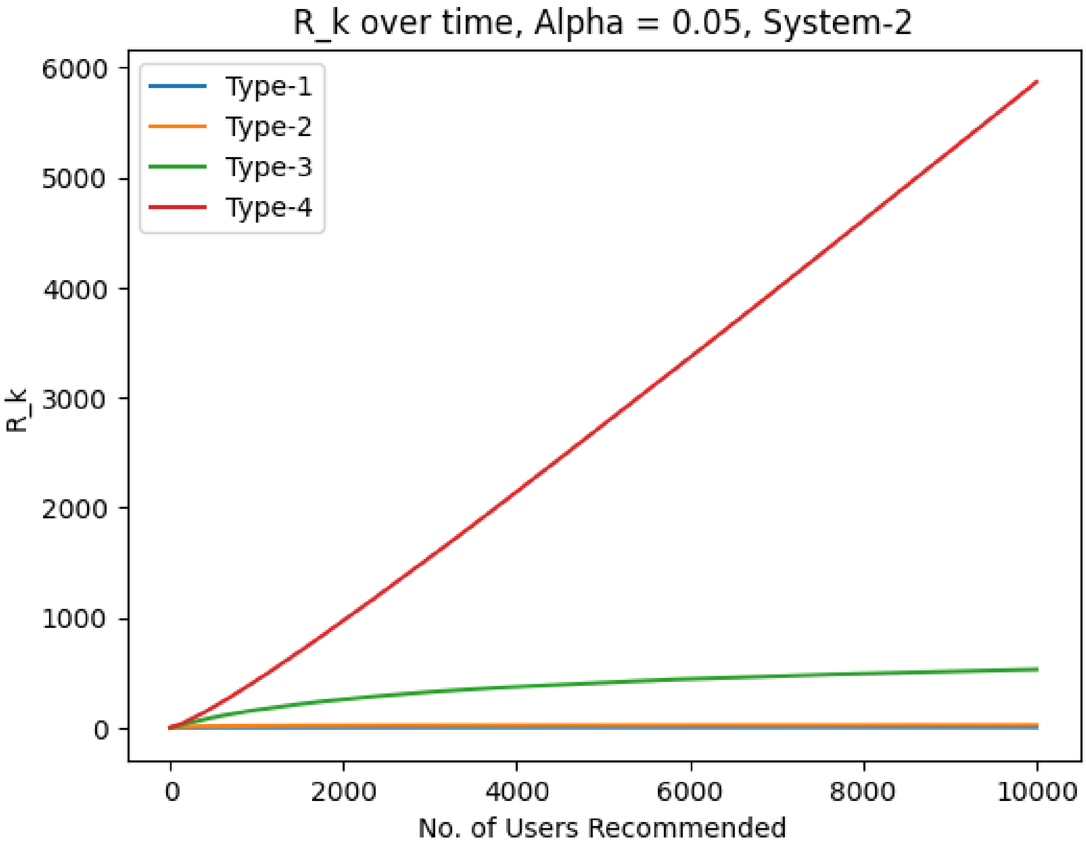
```
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p005_trial += res
    rev_avgd_005_trial += rev_storer
    m_by_nk_avgd_005_trial += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005_trial[j] += 1
    if i % 10 == 0:
        print(i)
```

[483]:

```
plot_it_3 = (rev_avgd_005.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-2')

np.sum(plot_it_3[:, -2])
```

[483]:

```
6429.694593340157
```

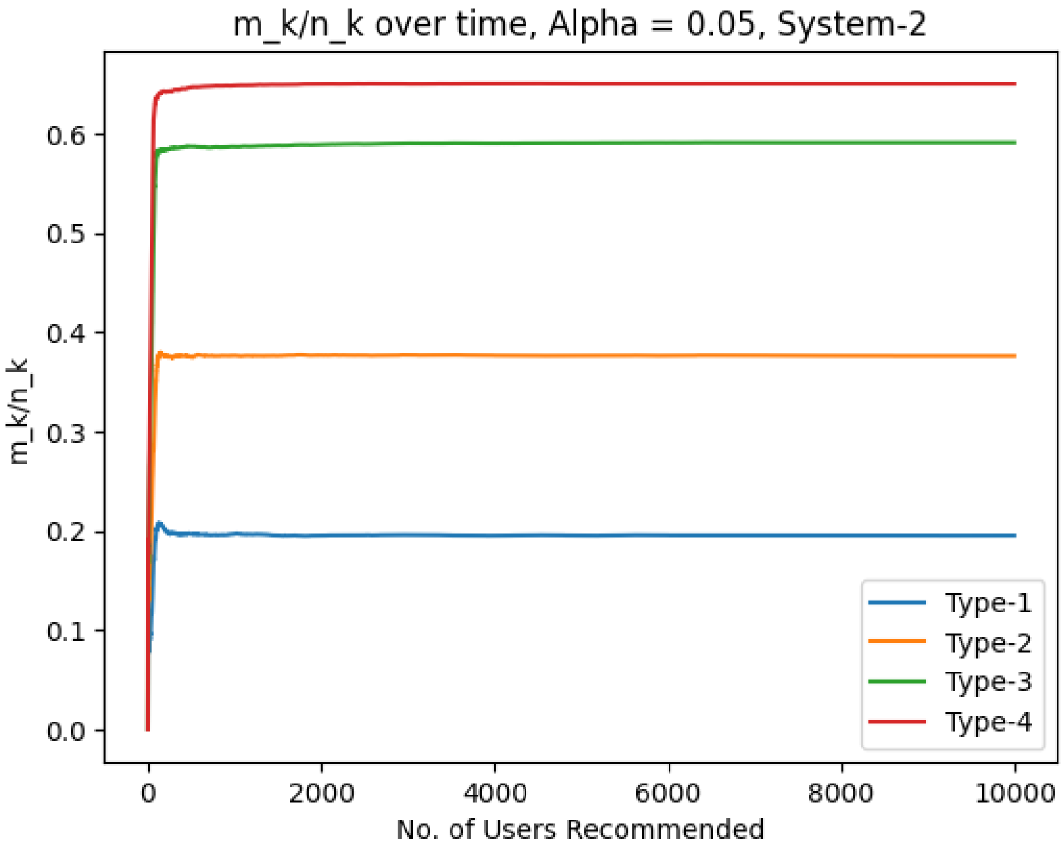


[484]:

```
plot_it_3 = (m_by_nk_avgd_005.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.05, System-2')

plot_it_3[:, -2]
```

[484]:

```
array([0.19523239, 0.37598713, 0.59084667, 0.65008565])
```



N = 1000

[ ]:

```
N = 1000
K = 4
ucb_p001_1k_2 = np.zeros((K, N + 1))
a_k = [2, 2.5, 2.5, 3]
```

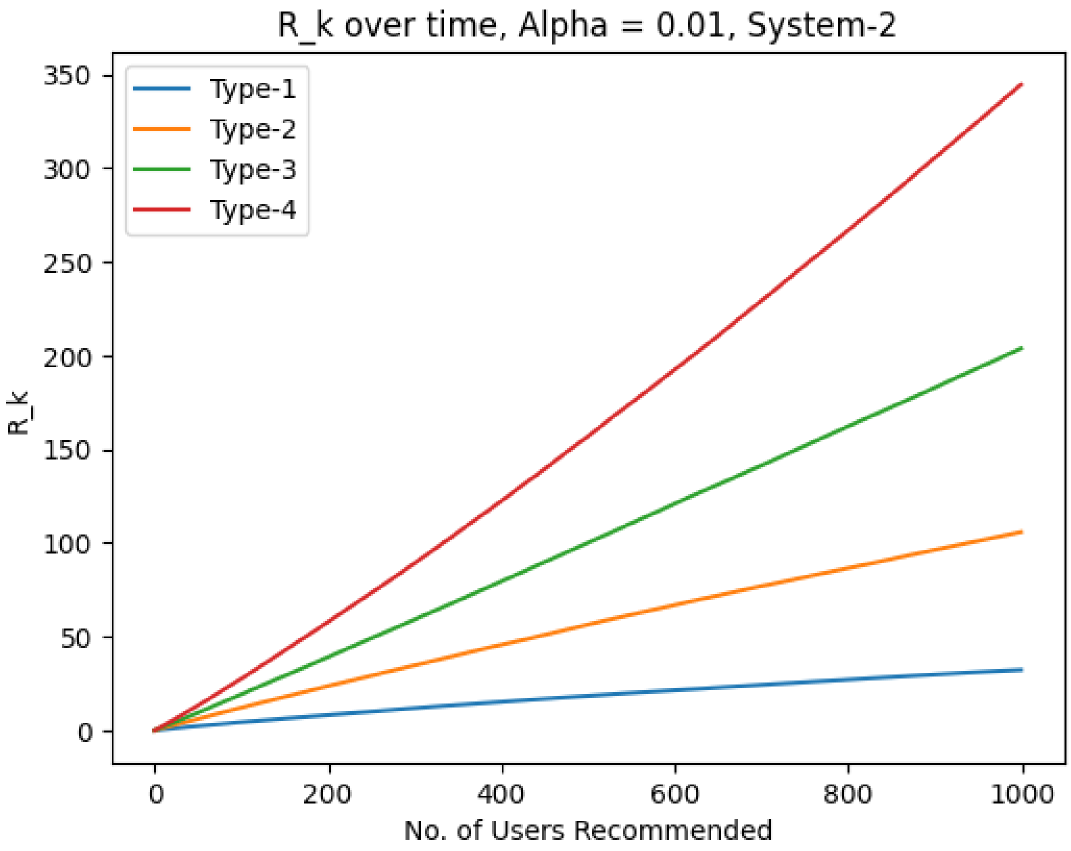
```
num_iter = 1000
count_p001_1k_2 = np.zeros(K)
rev_avgd_001_1k_2 = np.zeros_like(ucb_p001_1k_2)
m_by_nk_avgd_001_1k_2 = np.zeros_like(ucb_p001_1k_2)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.01
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p001_1k_2 += res
    rev_avgd_001_1k_2 += rev_storer
    m_by_nk_avgd_001_1k_2 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p001_1k_2[j] += 1
    if i % 10 == 0:
        print(i)
```

[568]:

```
plot_it_3 = (rev_avgd_001_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.01, System-2')

np.sum(plot_it_3[:, -2])
```

[568]:

```
686.0983332340832
```

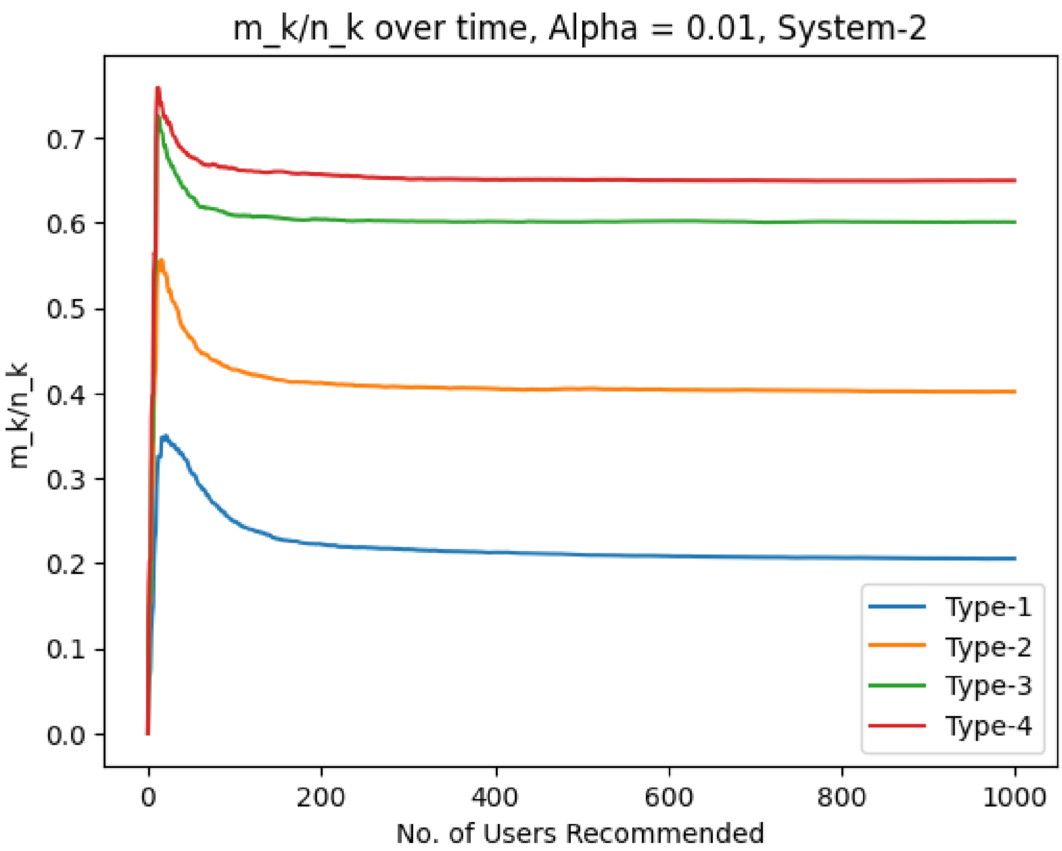


[569]:

```
plot_it_3 = (m_by_nk_avgd_001_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.01, System-2')

plot_it_3[:, -2]
```

[569]:

```
array([0.20537006, 0.40194411, 0.60127749, 0.65023866])
```

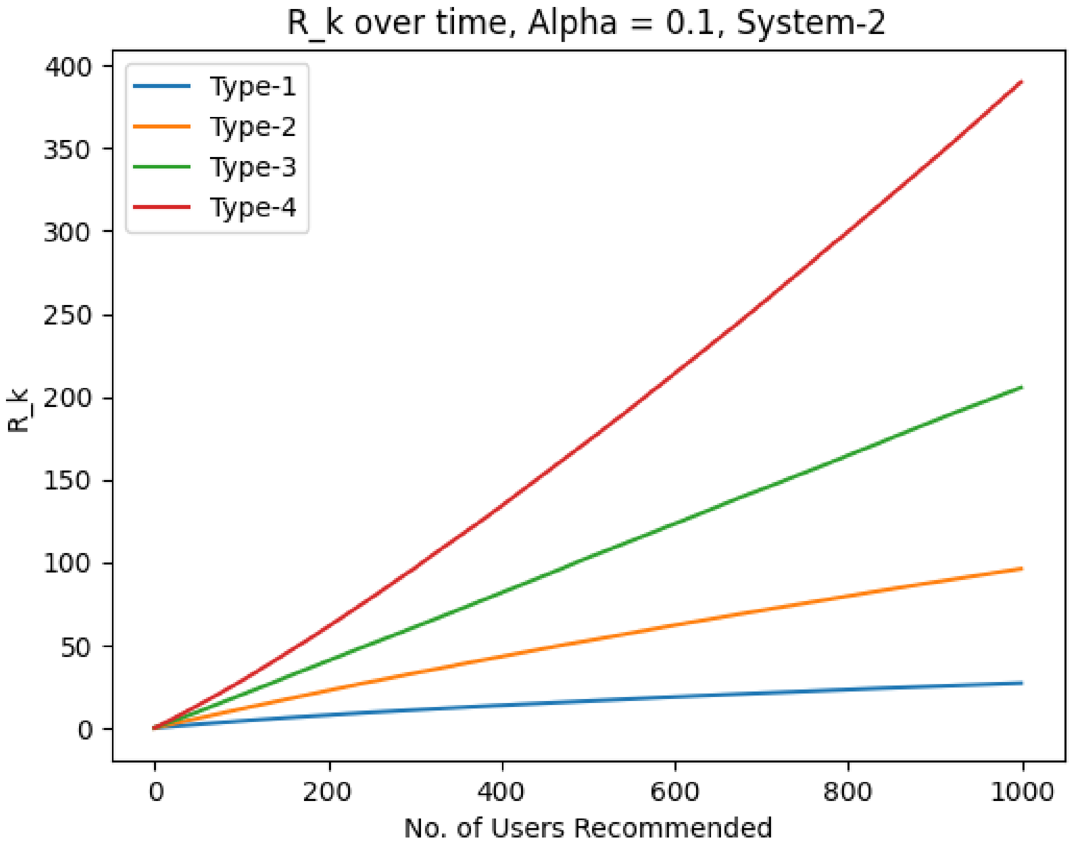


[ ]:

```
N = 1000
K = 4
ucb_p01_1k_2 = np.zeros((K, N + 1))
a_k = [2, 2.5, 2.5, 3]
num_iter = 1000
count_p01_1k_2 = np.zeros(K)
rev_avgd_01_1k_2 = np.zeros_like(ucb_p01_1k_2)
m_by_nk_avgd_01_1k_2 = np.zeros_like(ucb_p01_1k_2)
p_k = [0.2, 0.4, 0.6, 0.65]
alpha = 0.1
for i in range(num_iter):
    res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
    ucb_p01_1k_2 += res
    rev_avgd_01_1k_2 += rev_storer
    m_by_nk_avgd_01_1k_2 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p01_1k_2[j] += 1
    if i % 10 == 0:
        print(i)
```

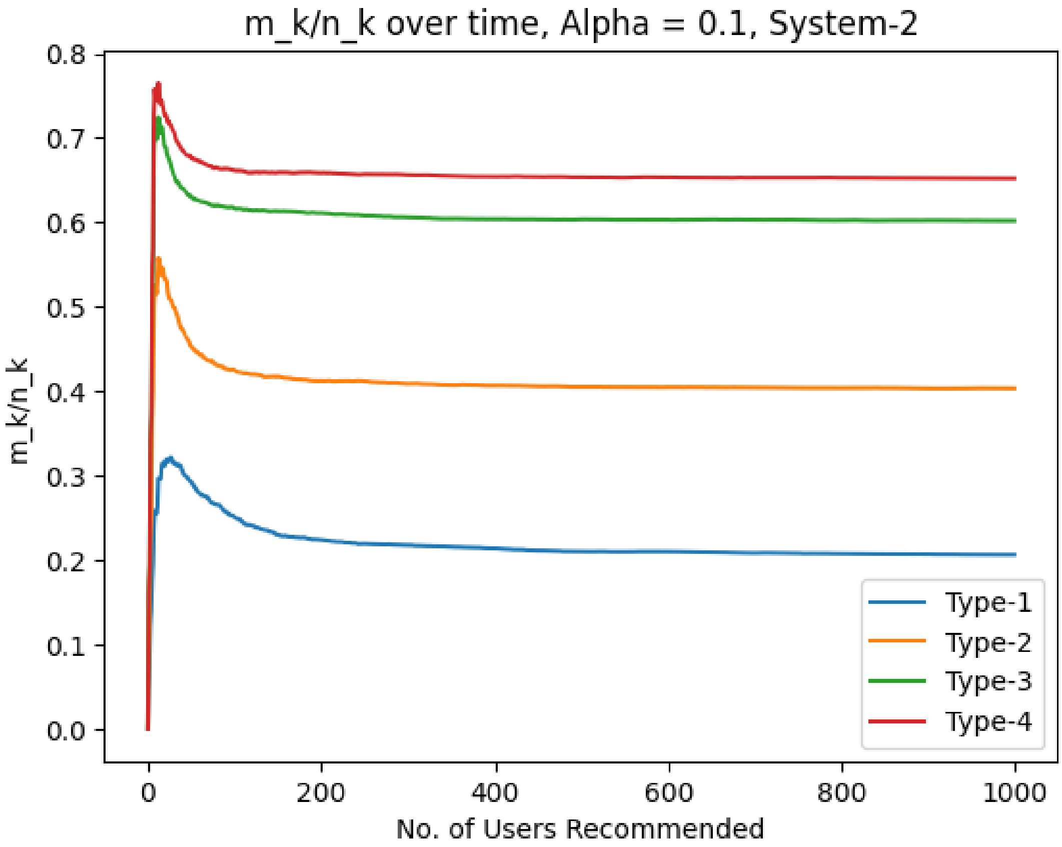
[565]:
```
plot_it_3 = (rev_avgd_01_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.1, System-2')

np.sum(plot_it_3[:, -2])
```

[565]:
```
718.5314684974713
```



[566]:
```
plot_it_3 = (m_by_nk_avgd_01_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.1, System-2')

plot_it_3[:, -2]
```

```
[566]: array([0.20631329, 0.403196  , 0.60190221, 0.65186814])
```



```
[ ]: N = 1000
     K = 4
     ucb_p005_1k_2 = np.zeros((K, N + 1))
     a_k = [2, 2.5, 2.5, 3]
     num_iter = 1000
     count_p005_1k_2 = np.zeros(K)
     rev_avgd_005_1k_2 = np.zeros_like(ucb_p005_1k_2)
     m_by_nk_avgd_005_1k_2 = np.zeros_like(ucb_p005_1k_2)
     p_k = [0.2, 0.4, 0.6, 0.65]
     alpha = 0.1
     for i in range(num_iter):
         res, m_by_n_storer, rev_storer = ucb_rev_shenai(N, p_k, a_k, 20, 4, alpha)
         ucb_p005_1k_2 += res
         rev_avgd_005_1k_2 += rev_storer
```

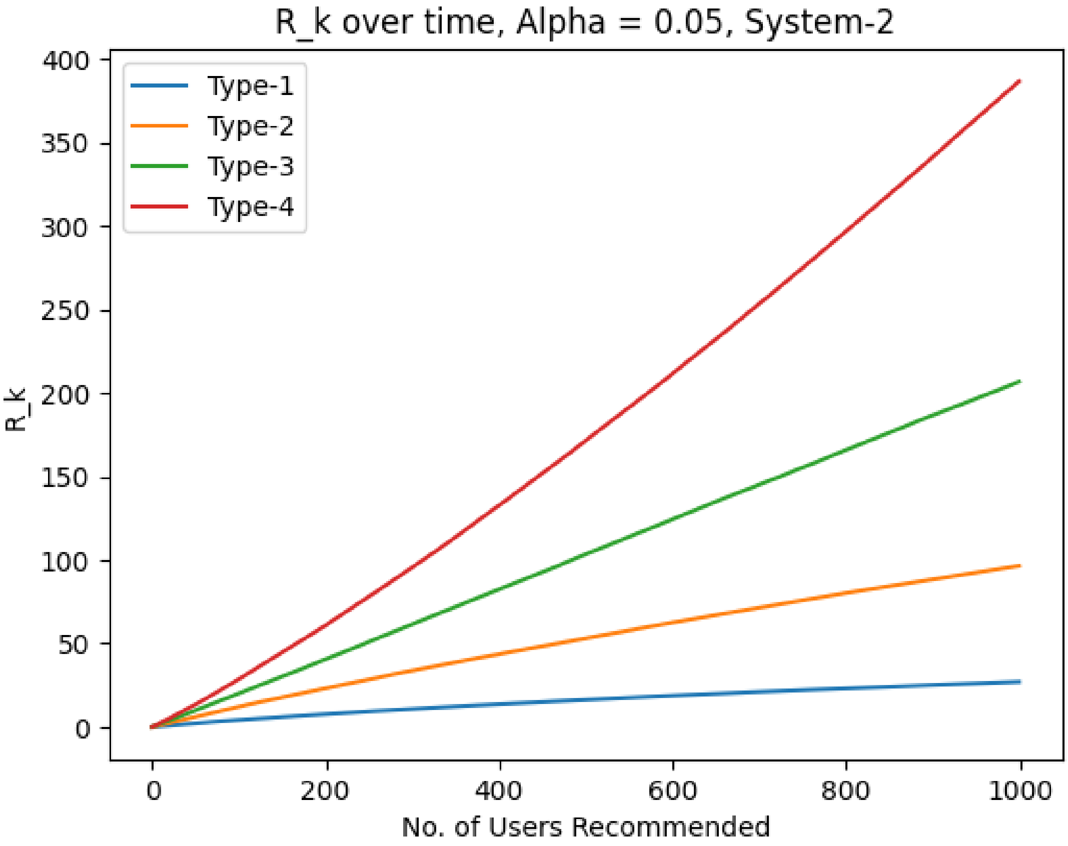
```
    m_by_nk_avgd_005_1k_2 += m_by_n_storer
    # rev_avgd += rev_add/num_iter
    for j in range(K):
        if res[j, -1] != 0:
            count_p005_1k_2[j] += 1
    if i % 10 == 0:
        print(i)
```

[571]:

```
plot_it_3 = (rev_avgd_005_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("R_k")
plt.title('R_k over time, Alpha = 0.05, System-2')

np.sum(plot_it_3[:, -2])
```

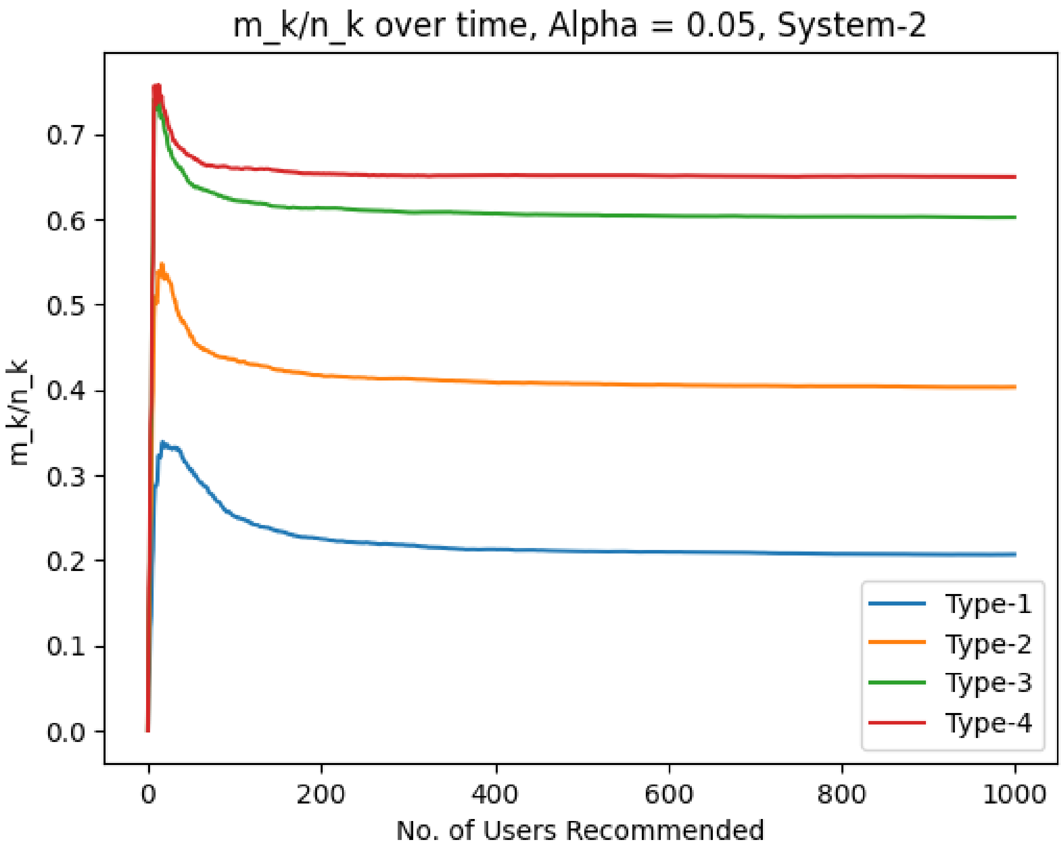
[571]:

```
716.5279268708452
```



[572]:
```python
plot_it_3 = (m_by_nk_avgd_005_1k_2.T/num_iter).T
for j in range(K):
    plt.plot(plot_it_3[j, :-1])
plt.legend(['Type-1', 'Type-2', 'Type-3', 'Type-4'])
plt.xlabel("No. of Users Recommended")
plt.ylabel("m_k/n_k")
plt.title('m_k/n_k over time, Alpha = 0.05, System-2')

plot_it_3[:, -2]
```

[572]:
```
array([0.20637956, 0.4031827 , 0.60233531, 0.64996524])
```


m_k/n_k over time, Alpha = 0.05, System-2
m_k/n_k
No. of Users Recommended
Type-1
Type-2
Type-3
Type-4