Adaptive Randomization with Arm-Elimination

Li Xiaocheng
Renmin University of China
Economics and English

Adaptive Randomization with Arm-Elimination
GLR Statistics and test

Multi-Arm Bandit with Side Information

Review

treatment effect PDF: $f_\theta(x) = e^{\theta x - \psi(\theta)}$

mean: $\psi'(\theta)$ and variance: $\psi''(\theta)$

Special case: Bernoulli Distribution, $\theta = log(\frac{p}{1-p})$ and $\psi(\theta) = \log(e^\theta + 1)$

Kullback-Leibler Information

$$I(\mu, \mu') = \mathbb{E}_{\theta_\mu}\left\{\log(\frac{f_{\theta_\mu}(X)}{f_{\theta_{\mu'}}(X)})\right\} = (\theta_\mu - \theta_{\mu'})\mu - [\psi(\theta_\mu) - \psi(\theta_{\mu'})]$$

Generalized Likelihood Ratio(GLR) Statistics

$$l_j^i(k, k') = n_{ijk}\left\{\hat{\mu}_{ijk}\theta_{\hat{\mu}_{ijk}} - \psi\left(\theta_{\hat{\mu}_{ijk}}\right)\right\} + n_{ijk'}\left\{\hat{\mu}_{ijk'}\theta_{\hat{\mu}_{ijk'}} - \psi\left(\theta_{\hat{\mu}_{ijk'}}\right)\right\} - (n_{ijk} + n_{ijk'})\{\bar{\mu}\theta_{\bar{\mu}} - \psi(\theta_{\bar{\mu}})\},$$

where $\bar{\mu} = \frac{n_{ijk}\hat{\mu}_{ijk} + n_{ijk'}\hat{\mu}_{ijk'}}{n_{ijk} + n_{ijk'}}$ of biomarker j and strategy k in interim i. Note that $n_{ij} = \sum_{k=1}^{K} n_{ijk}$.

Adaptive Randomization with Arm-Elimination

Arm Elimination

Multi-Arm Bandit with Side Information

Elimination Rule

After trials of interim i , arm(a.k.a strategy) $k \neq \hat{k}_{ij}$ within biomarker j is eliminated if

$$l_j^i(\hat{k}_{ij}, k) \geq a_\alpha$$

where $\hat{k}_{ij}$ refers to the best treatment of biomarker j in interim i

(Note that $l(\lambda, \lambda) = 0$)

a_α and α

- The the best strategy for each biomarker class is not eliminated is guaranteed with the probability $1 - \alpha$
- In my simulation, the best strategy is presumed to be unique
  - An alternative is to choose the best strategy set, with guaranteed probability not being eliminated $1 - \alpha$

Adaptive Randomization with Arm-Elimination
Arm Elimination — Computation of a_α

Multi-Arm Bandit with Side Information

Restatement of α

A_j $\rightarrow$ the best treatment of biomarker j is eliminated

$$A_j = \{\max_{k \neq \hat{k}_{ij}} [l_j^i(\hat{k}_{ij}, k) I_{\{\hat{\mu}_{ijk} > \hat{\mu}_{ij\hat{k}_{ij}}\}}] \geq a_\alpha \text{ for some } 1 \leq i \leq I\}$$

Thus, $\alpha = \mathrm{P}\left(\bigcup_{j=1}^{J} A_j\right)$

Determine a_α

- P_* is used as the probability measure satisfying $\theta_{j1} = \cdots = \theta_{jK}$,
  - Thus α can be computed through the De Morgan's Law
- we can let $1 = \hat{k}_{ij}$, n_{ijk} can be approximated by $\frac{\left(1+o_p(1)\right)n_{ij}}{K}$

Adaptive Randomization with Arm-Elimination

Arm Elimination — Computation of a_α

Central Limit Theorem

- $l_j^i(1, k)$ can be approximated by $\left(1 + O_p(1)\right)\frac{n_{ij}}{4K\psi''(\theta_{\mu j1})}(\hat{\mu}_{ijk} - \hat{\mu}_{ij1})^2 = \frac{1}{2}(\Delta_{jk}^i)^2$
  - where $\Delta_{jk}^i = \left(\frac{n_{ij}}{4K\psi''(\theta_{\mu j1})}\right)^{1/2}(\hat{\mu}_{ijk} - \hat{\mu}_{ij1})$

- By applying Central Limit Theorem, $\Delta_{jk}^i = \frac{\hat{\mu}_{ijk} - \hat{\mu}_{ij1}}{\sqrt{2\frac{\psi''(\theta_{\mu j1})}{\frac{n_{ij}}{K}}}} \sim \mathcal{N}(0, 1)$

Adaptive Randomization with Arm-Elimination
Arm Elimination — Computation of a_α

Multi-Arm Bandit with Side Information

Multivariate Markov Chain

$i \in \{1,2,3,4\}$, given $(\Delta_{j2}^{i}, \dots, \Delta_{jK}^{i})$,

$$\Delta_{jk}^{i+1} = \frac{\hat{\mu}_{i+1,jk} - \hat{\mu}_{i+1,j1}}{\sqrt{2\frac{\psi''(\theta_{\mu j1})}{\frac{n_{i+1,j}}{K}}}} = \sqrt{\frac{\frac{n_{i+1,j}}{K}}{2\psi''(\theta_{\mu j1})}}\ \frac{\hat{S}_{i+1,jk} - \hat{S}_{i+1,j1}}{\frac{n_{i+1,j}}{K}}$$

$$= \sqrt{\frac{n_{ij}}{n_{i+1,j}}}\sqrt{\frac{\frac{n_{ij}}{K}}{2\psi''(\theta_{\mu j1})}}\ \left(\frac{\hat{S}_{ijk}}{\frac{n_{i,j}}{K}} - \frac{\hat{S}_{i,j1}}{\frac{n_{i,j}}{K}}\right) + \sqrt{\frac{n_{i+1,j} - n_{i+1,j}}{n_{i+1,j}}}\sqrt{\frac{\frac{n_{i+1,j} - n_{i+1,j}}{K}}{2\psi''(\theta_{\mu j1})}}\ \left[\frac{(\hat{S}_{i+1,jk} - \hat{S}_{ijk})}{\frac{n_{i+1,j} - n_{i,j}}{K}} - \frac{(\hat{S}_{i+1,j1} - \hat{S}_{ij1})}{\frac{n_{i+1,j} - n_{i,j}}{K}}\right]$$

$$= \sqrt{\frac{n_{ij}}{n_{i+1,j}}}\,\Delta_{jk}^{i} + \sqrt{\frac{n_{i+1,j} - n_{i+1,j}}{n_{i+1,j}}}\sqrt{\frac{\frac{n_{i+1,j} - n_{i+1,j}}{K}}{2\psi''(\theta_{\mu j1})}}\ \left[\frac{(\hat{S}_{i+1,jk} - \hat{S}_{ijk})}{\frac{n_{i+1,j} - n_{i,j}}{K}} - \frac{(\hat{S}_{i+1,j1} - \hat{S}_{ij1})}{\frac{n_{i+1,j} - n_{i,j}}{K}}\right]$$

Adaptive Randomization with Arm-Elimination
Arm Elimination — Computation of a_α

Multi-Arm Bandit with Side Information

- $$\Delta_{jk}^{i+1} = \sqrt{\frac{n_{ij}}{n_{i+1,j}}}\,\Delta_{jk}^{i} + \sqrt{\frac{n_{i+1,j}-n_{i+1,j}}{n_{i+1,j}}}\sqrt{\frac{\frac{n_{i+1,j}-n_{i+1,j}}{K}}{2\psi''(\theta_{\mu j1})}}\;\left[\frac{(\hat{S}_{i+1,jk}-\hat{S}_{ijk})}{\frac{n_{i+1,j}-n_{i,j}}{K}} - \frac{(\hat{S}_{i+1,j1}-\hat{S}_{ij1})}{\frac{n_{i+1,j}-n_{i,j}}{K}}\right]$$

Where $\sqrt{\frac{\frac{n_{i+1,j}-n_{i+1,j}}{K}}{2\psi''(\theta_{\mu j1})}}\;\left[\frac{(\hat{S}_{i+1,jk}-\hat{S}_{ijk})}{\frac{n_{i+1,j}-n_{i,j}}{K}} - \frac{(\hat{S}_{i+1,j1}-\hat{S}_{ij1})}{\frac{n_{i+1,j}-n_{i,j}}{K}}\right] \sim \mathcal{N}(0\,,1)$

Adaptive Randomization with Arm-Elimination
Arm Elimination — Computation of a_α

Multi-Arm Bandit with Side Information

- Moreover, let $\left(\frac{n_{ij}}{4K\psi''(\theta_{\mu j1})}\right)^{1/2} = \sigma_{\mu j1}$. For $k \neq k'$, $\operatorname{cov}(\Delta^i_{jk}, \Delta^i_{jk'}) = (\sigma_{\mu j1})^2 \operatorname{cov}\left(\hat{\mu}_{ijk} - \hat{\mu}_{ij1}, \hat{\mu}_{ijk'} - \hat{\mu}_{ij1}\right)$

$$= (\sigma_{\mu j1})^2 \operatorname{Var}\left(\hat{\mu}_{ij1}\right) = \frac{1}{2}\operatorname{Var}\left(\Delta^i_{jk}\right)$$

- Therefore, given $(\Delta^i_{j2}, \dots, \Delta^i_{jK})$, the distribution of $(\Delta^{i+1}_{j2}, \dots, \Delta^{i+1}_{jK})$ is

$$\mathcal{N}\left(\sqrt{\frac{n_{ij}}{n_{i+1,j}}}\begin{bmatrix}\Delta^i_{j2}\\ \vdots \\ \Delta^i_{jK}\end{bmatrix}, \frac{n_{i+1,j} - n_{i+1,j}}{n_{i+1,j}}\begin{bmatrix}1 & \cdots & \frac{1}{2}\\ \vdots & \ddots & \vdots\\ \frac{1}{2} & \cdots & 1\end{bmatrix}\right)$$

Adaptive Randomization with Arm-Elimination

Arm Elimination — Computation of a_α

Monte Carlo Method

- Obviously, $P_*(A_j)$ is non-increasing in a_α.
- It is also easy to prove that $P_*(\bigcup_{j=1}^{J} A_j)$ is non-increasing in a_α.
- Monte Carlo Simulation(I = 5, K = 3, J = 3, t= 100,000):
  - (1) Generate k-1-dimensional multivariate normal Markov Chain with period t = 5, store the maximum value.
  - (2) Repeat (1) t times for J biomarker respectively.
  - (3) Use Bisection Method to find a_α within a prescribed error.
- Result:

```
a_alpha: 2.592155653417598
```

- Alternative Method in Computing a_α: Recursive Numerical Integration.

Adaptive Randomization with Arm-Elimination
Simulation

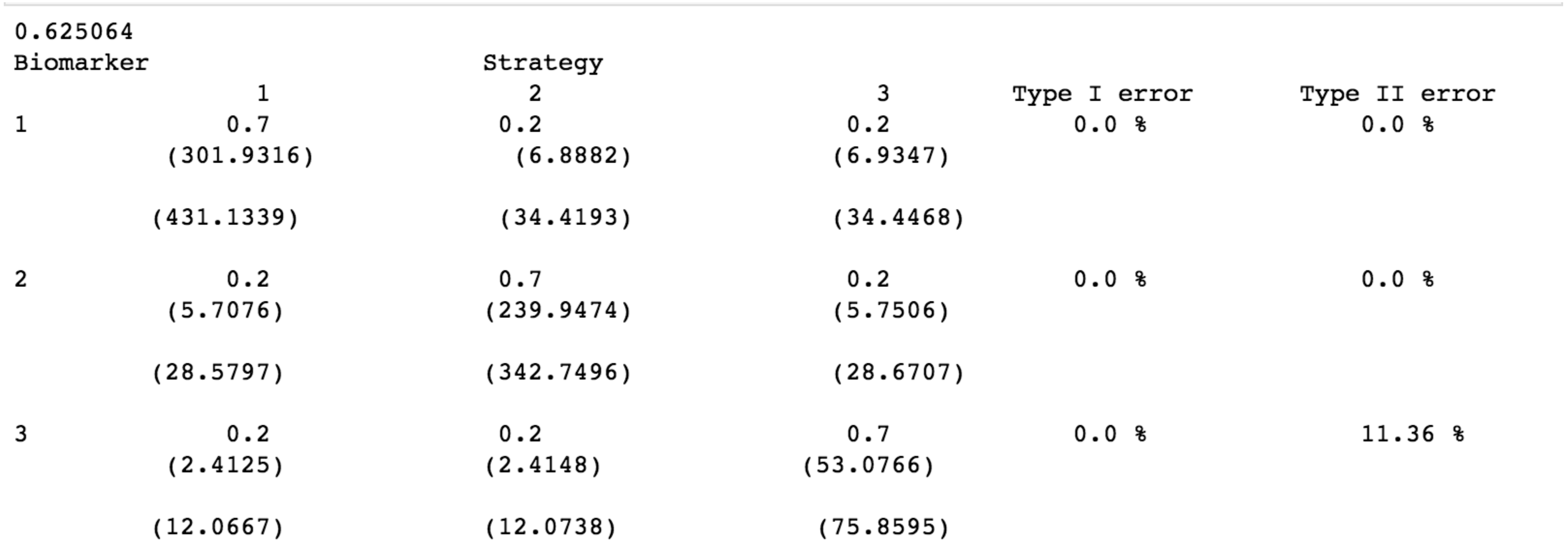
Multi-Arm Bandit with Side Information

Recap of AR (with elimination)

```
Compute $a_\alpha$
For $i \in \{1, \dots, I\}$ do
    For $j \in \{1, \dots, J\}, t \in \{1, \dots, n_{ij} - n_{i-1,j}\}$ do
        if $i = 1$ then $\phi_{jt} \leftarrow$ random $\{1, \dots, K\}$
        else
            $\phi_{jt} \begin{cases} \mathcal{B}_j \ \ with\, probability\ \frac{(1-|\mathcal{H}_j \backslash \mathcal{B}_j|\epsilon)}{|\mathcal{B}_j|} \\ \mathcal{H}_j \backslash \mathcal{B}_j \ \ with\, probbability\ \epsilon \end{cases}$
        end if
    if $l_j^i(\hat{k}_j, k_j) \geq a_\alpha$ then eliminate $k_j$ end if
```

Adaptive Randomization with Arm-Elimination
Simulation

Multi-Arm Bandit with Side Information



```
0.625064
Biomarker                    Strategy
                 1                  2                      3        Type I error        Type II error
1               0.7                0.2                    0.2          0.0 %               0.0 %
            (301.9316)          (6.8882)               (6.9347)

           (431.1339)          (34.4193)               (34.4468)

2               0.2                0.7                    0.2          0.0 %               0.0 %
            (5.7076)           (239.9474)              (5.7506)

           (28.5797)           (342.7496)              (28.6707)

3               0.2                0.2                    0.7          0.0 %               11.36 %
            (2.4125)           (2.4148)              (53.0766)

           (12.0667)           (12.0738)              (75.8595)
```

- Without Elimination: 0.586

Adaptive Randomization with Arm-Elimination
Simulation

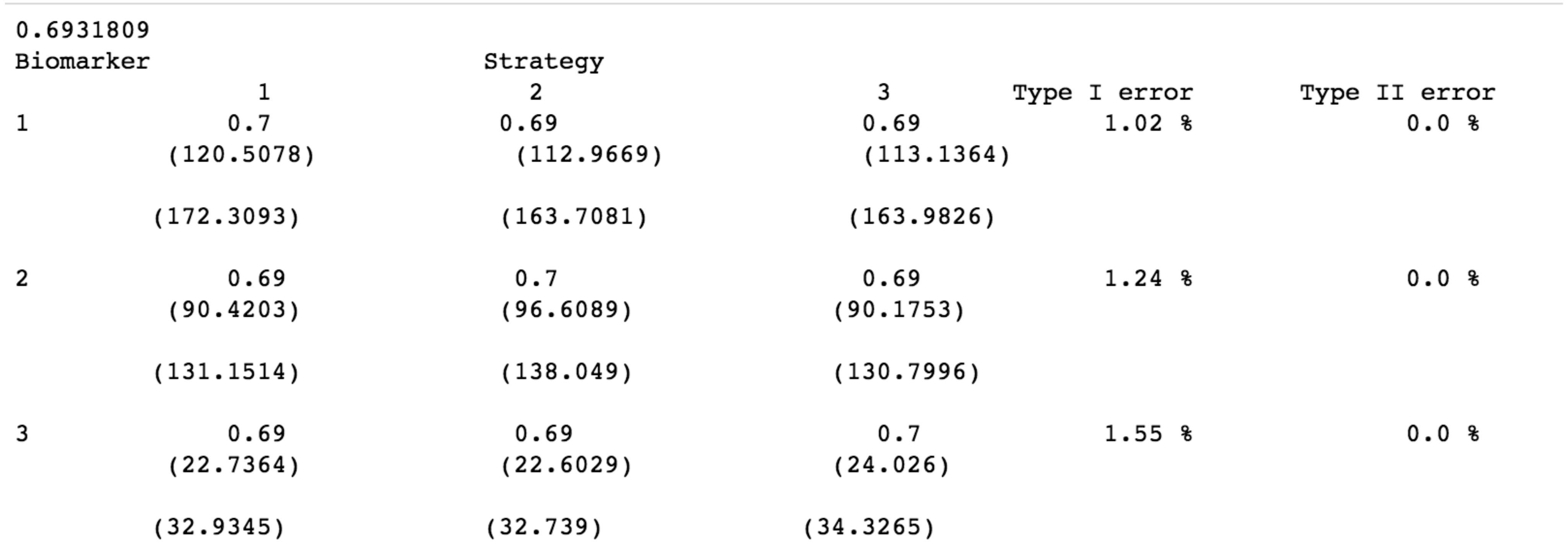
Multi-Arm Bandit with Side Information

```
0.6406417307599259
Biomarker                       Strategy
                1                 2                      3       Type I error       Type II error
1              0.7               0.5                    0.2          0.0 %             7.1 %
            (259.975)         (47.1925)              (6.8121)

           (371.5823)         (94.3107)              (3.4107)

2              0.2               0.7                    0.5          0.0 %             11.88 %
            (5.5921)         (199.2923)             (43.5972)

           (28.0188)         (284.8465)             (87.1347)

3              0.5               0.2                    0.7          0.0 %             72.22 %
            (14.1288)         (2.3334)              (42.053)

           (28.2274)         (11.6848)             (60.0878)
```

- Without Elimination: 0.592

Adaptive Randomization with Arm-Elimination
Simulation

Multi-Arm Bandit with Side Information



```
0.6931809
Biomarker                      Strategy
                1                  2                      3         Type I error         Type II error
1              0.7               0.69                   0.69          1.02 %                 0.0 %
            (120.5078)         (112.9669)             (113.1364)

           (172.3093)          (163.7081)             (163.9826)

2               0.69              0.7                   0.69          1.24 %                 0.0 %
            (90.4203)          (96.6089)              (90.1753)

           (131.1514)          (138.049)              (130.7996)

3               0.69              0.69                   0.7          1.55 %                 0.0 %
            (22.7364)          (22.6029)              (24.026)

           (32.9345)          (32.739)              (34.3265)
```

Non-Parametric Method — Simulation & Comparison

Li Xiaocheng
Renmin University of China
Economics and English

Non-Parametric Method — Simulation

A Novel Method

Histogram Method with Local Linear Regression

- The basic non-parametric method uses bins to split side information, then compute mean for each bin
- Yet for regions where choosing strategy is complex, this method engenders problems.
- A novel method:

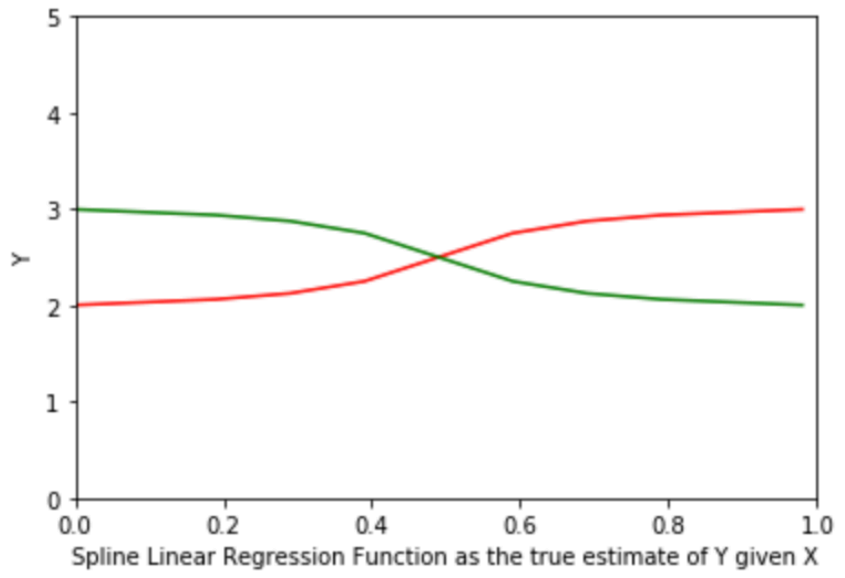
  Choose a consecutive set bins whose proportion of surviving strategies exceeds δ (δ is close to 1)
- Arm elimination criterion: At stage t, assume that $X_t \in B_i, N_r \in a^r$ for $a^r > 1$.

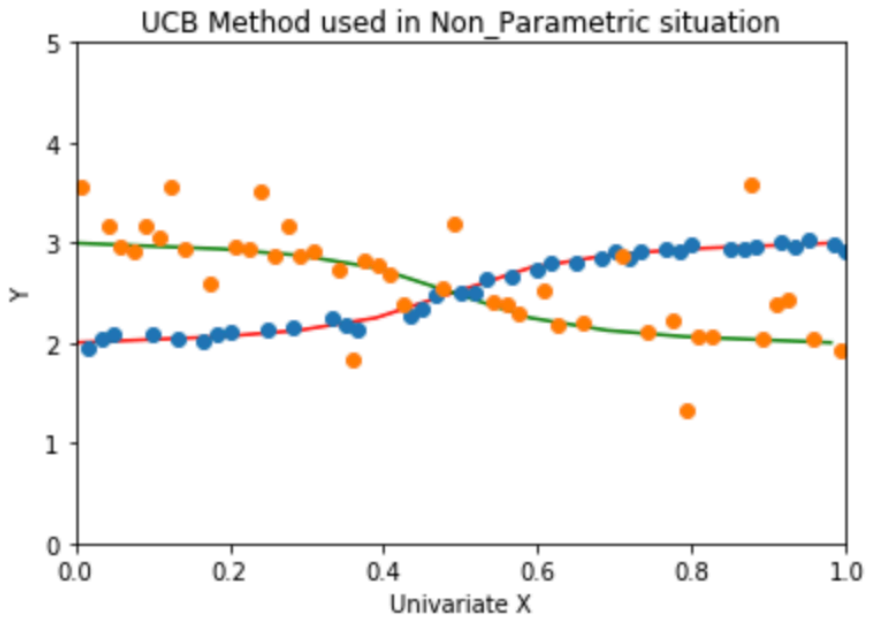
$$l_{j,t}^2(i) > g^2(n_{ij}(\mathrm{t})/N_r)$$

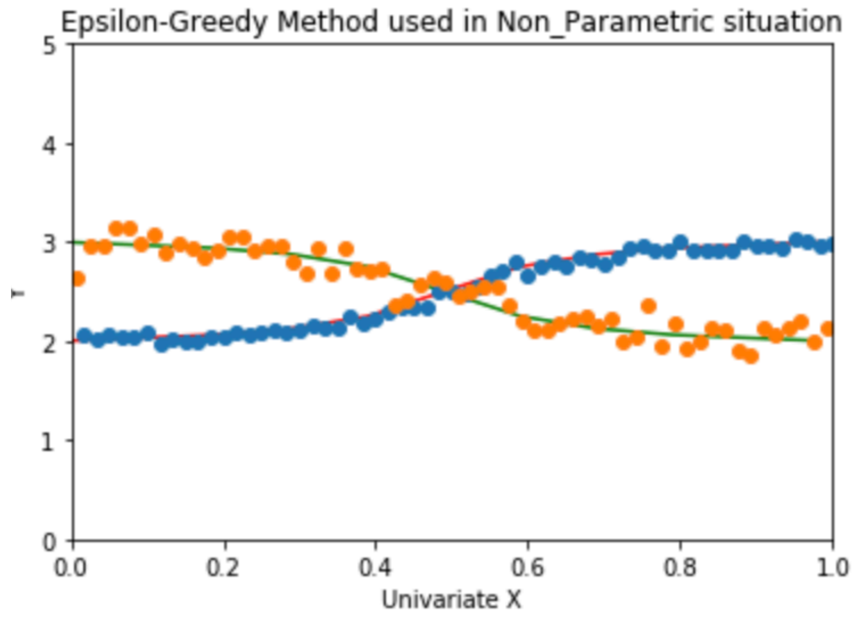
where $l_{j,t}^2(i) = (\bar{Y}_{\hat{j}n_{i\hat{j}}(\mathrm{t})} - \bar{Y}_{jn_{ij}(\mathrm{t})}) / \sqrt{\frac{s^2_{\hat{j}n_{i\hat{j}}(\mathrm{t})}}{n_{i\hat{j}}(\mathrm{t})} + \frac{s^2_{jn_{ij}(\mathrm{t})}}{n_{ij}(\mathrm{t})}}$

Adaptive Randomization with Arm-Elimination

Multi-Arm Bandit with Side Information

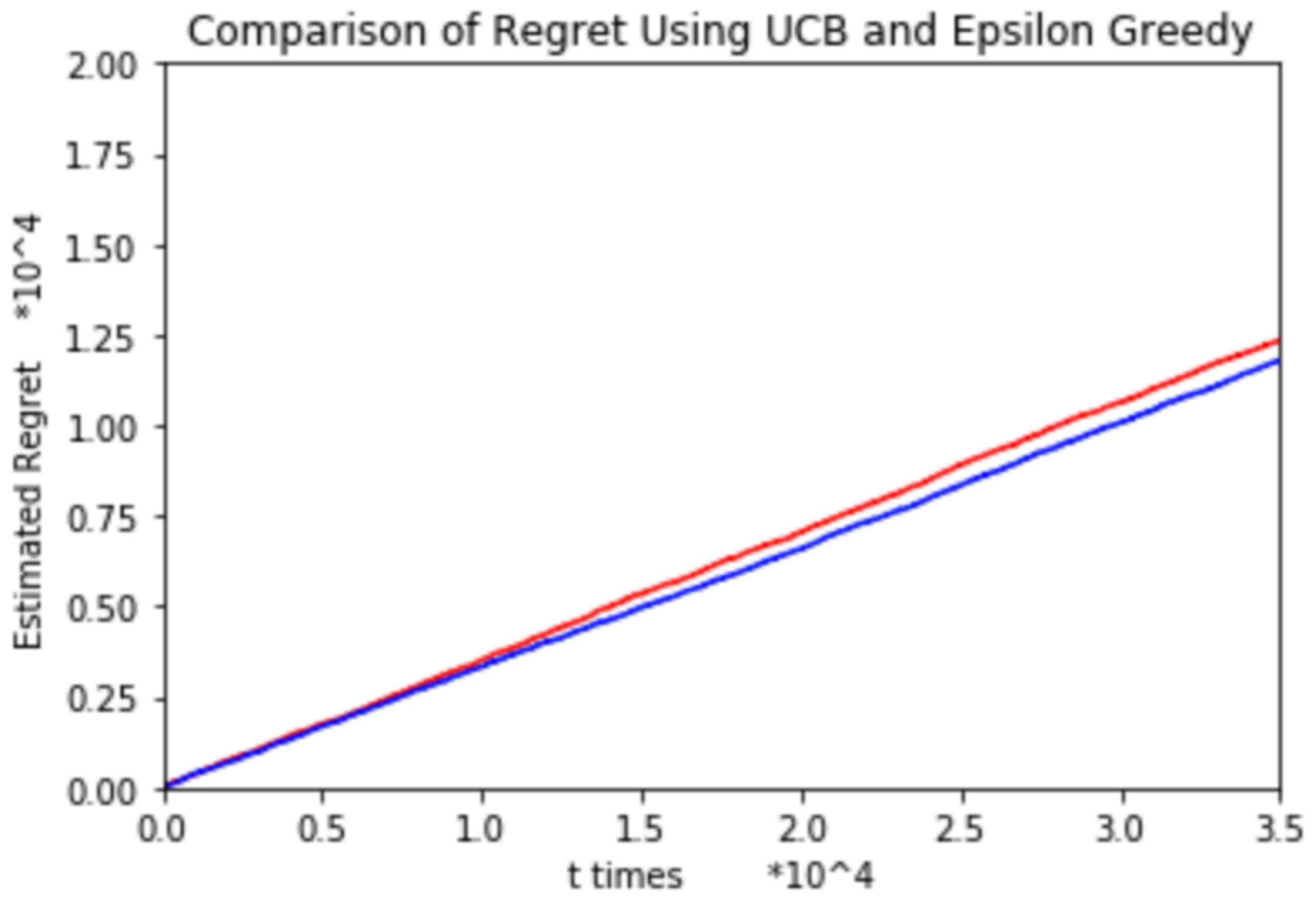






Adaptive Randomization with Arm-Elimination

Multi-Arm Bandit with Side Information



Red: Epsilon-Greedy

Blue: UCB

Future Plan

1

Theory

2

Application

Adaptive Randomization with Arm-Elimination

GLR Statistics and test

Multi-Arm Bandit with Side Information

Gnereralized Likelihood Ratio: Distribution and Approximation

A more effective AR method with elimination as a non-parametric method

A more efficient threshold for eliminating inferior strategies

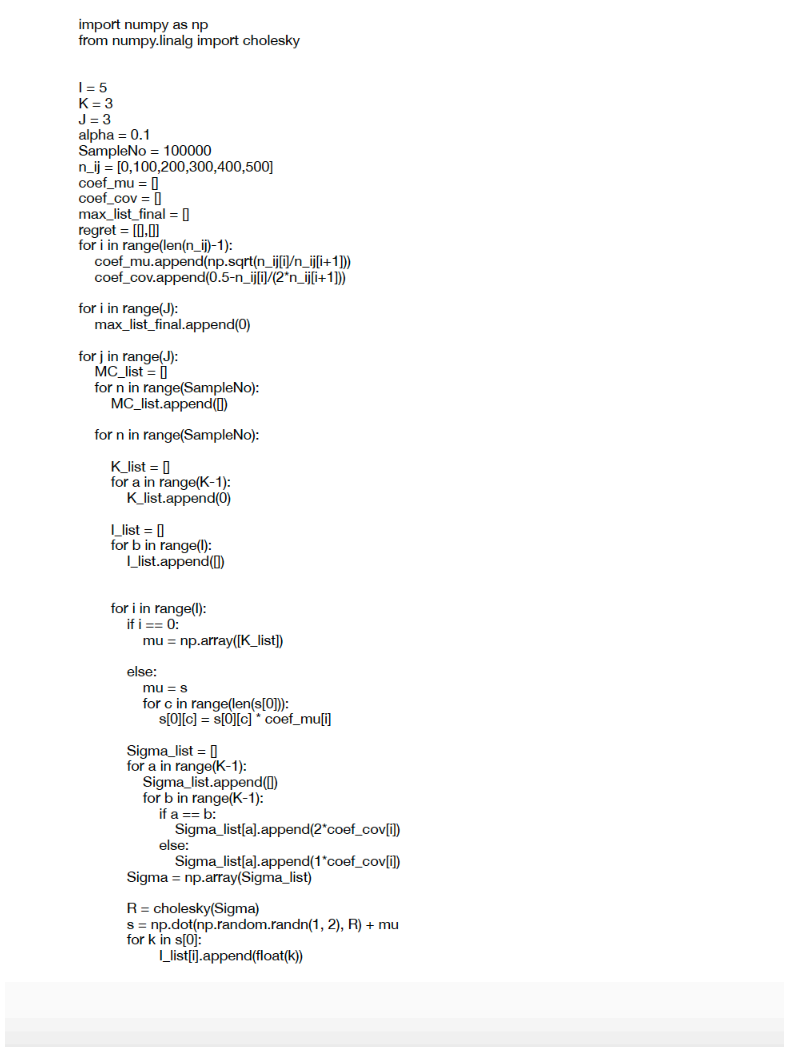
Business Insight : How will a new MoBike allocate their bikes

- The Emergence of "Sharing Economy"
- How can you find the nearest bike or charger?
- Cost-effective strategy
- Gaming between the juggernauts?

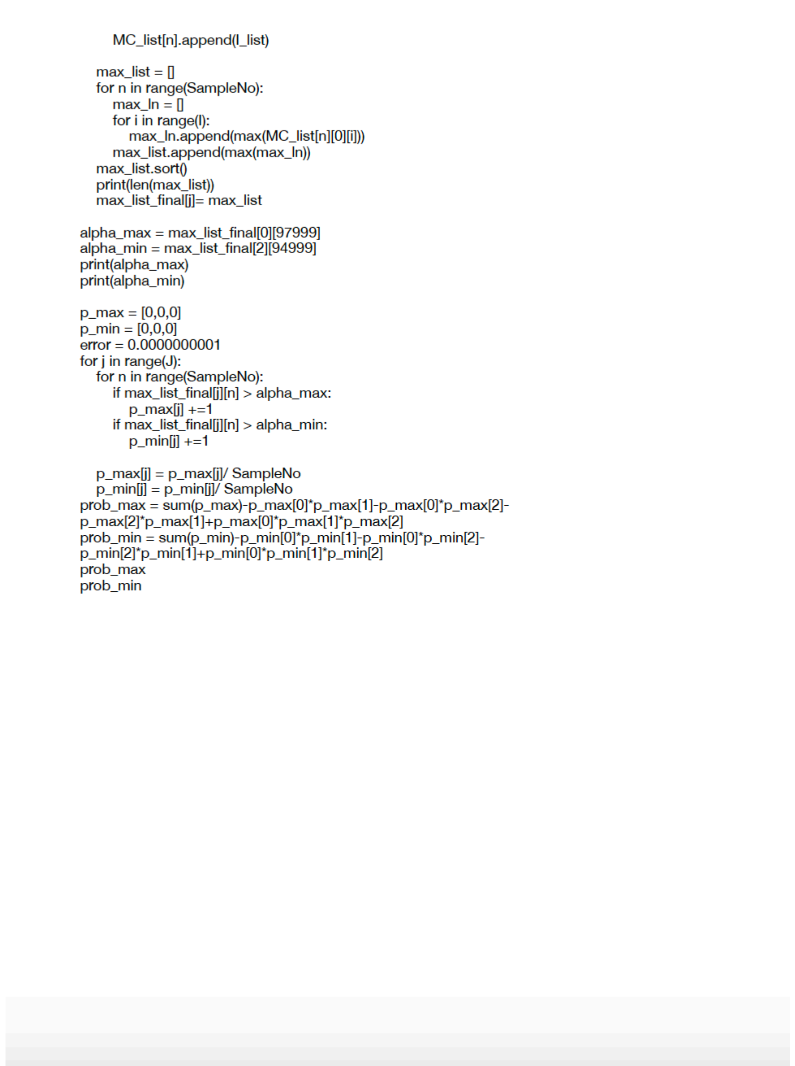
Appendix

A Snapshot of Codes

Computing a-alpha By Xiaocheng Li



```
import numpy as np
from numpy.linalg import cholesky


I = 5
K = 3
J = 3
alpha = 0.1
SampleNo = 100000
n_ij = [0,100,200,300,400,500]
coef_mu = []
coef_cov = []
max_list_final = []
regret = [[],[]]
for i in range(len(n_ij)-1):
    coef_mu.append(np.sqrt(n_ij[i]/n_ij[i+1]))
    coef_cov.append(0.5-n_ij[i]/(2*n_ij[i+1]))

for i in range(J):
    max_list_final.append(0)

for j in range(J):
    MC_list = []
    for n in range(SampleNo):
        MC_list.append([])

    for n in range(SampleNo):

        K_list = []
        for a in range(K-1):
            K_list.append(0)

        I_list = []
        for b in range(I):
            I_list.append([])


        for i in range(I):
            if i == 0:
                mu = np.array([K_list])

            else:
                mu = s
                for c in range(len(s[0])):
                    s[0][c] = s[0][c] * coef_mu[i]

            Sigma_list = []
            for a in range(K-1):
                Sigma_list.append([])
                for b in range(K-1):
                    if a == b:
                        Sigma_list[a].append(2*coef_cov[i])
                    else:
                        Sigma_list[a].append(1*coef_cov[i])
            Sigma = np.array(Sigma_list)

            R = cholesky(Sigma)
            s = np.dot(np.random.randn(1, 2), R) + mu
            for k in s[0]:
                    I_list[i].append(float(k))
```



```
        MC_list[n].append(I_list)

    max_list = []
    for n in range(SampleNo):
        max_In = []
        for i in range(I):
            max_In.append(max(MC_list[n][0][i]))
        max_list.append(max(max_In))
    max_list.sort()
    print(len(max_list))
    max_list_final[j]= max_list

alpha_max = max_list_final[0][97999]
alpha_min = max_list_final[2][94999]
print(alpha_max)
print(alpha_min)

p_max = [0,0,0]
p_min = [0,0,0]
error = 0.0000000001
for j in range(J):
    for n in range(SampleNo):
        if max_list_final[j][n] > alpha_max:
            p_max[j] +=1
        if max_list_final[j][n] > alpha_min:
            p_min[j] +=1

    p_max[j] = p_max[j]/ SampleNo
    p_min[j] = p_min[j]/ SampleNo
prob_max = sum(p_max)-p_max[0]*p_max[1]-p_max[0]*p_max[2]-
p_max[2]*p_max[1]+p_max[0]*p_max[1]*p_max[2]
prob_min = sum(p_min)-p_min[0]*p_min[1]-p_min[0]*p_min[2]-
p_min[2]*p_min[1]+p_min[0]*p_min[1]*p_min[2]
prob_max
prob_min
```

A Snapshot of Codes

Multi-Arm Bandit with Side Information

AR with Elimination By Xiaocheng Li

```
#准备工作
import numpy as np
import re
J = 3
K = 3
n_interval = 200
I = 5
epsilon = 0.1
a_alpha = 2.592155653417598

#计算GLR的中间步骤
def theta(mu_st,n):
    import numpy as np
    theta = 0
    theta = mu_st * (np.log(mu_st)-np.log(1-mu_st)) + np.log(1-mu_st)
    theta = theta * n
    return theta

#计算GLR
def GLR(success_1,trials_1,J,K):
    list1 = []
    none_final = 0
    import numpy as np
    for j in range(J):
        list1.append([])
        for k in range(K):
            if j == k:
                list1[j].append(0)
            else:
                if trials_1[j][k] == 0:
                    list1[j].append(none_final)
                    break
                none_1 = []
                none_2 = []
                none_3 = []
                none_1 = [success_1[j][k], trials_1[j][k]]
                none_2 = [max(success_1[j]), trials_1[j][np.argmax(success_1[j])]]
                none_3 = [none_1[0] + none_2[0], none_1[1] + none_2[1]]
                none_final = theta(none_1[0]/none_1[1], none_1[1]) + theta(none_2[0]/none_2[1],
none_2[1])
                none_final = none_final - theta(none_3[0]/none_3[1], none_3[1])
                list1[j].append(none_final)
    return list1

def
AR_elimination_loop(stage,J,K,eps,Trials_Begin,Acc_Trials,Success,Mu,a_alpha_index,elimination)
:

    suc = 0
    tot = 0

    trials = Trials_Begin
    # Create Patients Data
    Patients = {}
    for j in range(J):
        Patients.setdefault(j+1,{})
        for k in range(K):
            Patients[j+1].setdefault(k+1,[])
```

```
    # Update Patients Data and other Matrixes
    for j in Patients:
        if j == 1:
            trials_j = int(trials * 0.5)
        elif j ==2:
            trials_j = int(trials * 0.4)
        else:
            trials_j = int(trials * 0.1)

        acc_trials_j = []
        for k in range(K):
            acc_trials_j.append((Acc_Trials[j-1][k]))

        acc_success_j = []
        for k in range(K):
            acc_success_j.append((Success[j-1][k]))

        n_inferior = 0
        superior_choice = []
        inferior_choice = []
        for k in range(K):
            if elimination[j-1][k] == 'superior':
                superior_choice.append(k)
            elif elimination[j-1][k] == 'inferior':
                n_inferior += 1
                inferior_choice.append(k)
        #Begin the trial
        for n in range(trials_j):

            prob = 0
            prob = np.random.uniform(0,1)
            choose_k = 0

            if stage == 0:
            #注明：这里默认只有3个treatment
                choice_k = int(np.random.choice([1,2,3]))
            else:
                prob_2 = 0
                prob_2 = np.random.uniform(0,1)
                if prob_2 <= n_inferior * eps:
                    choice_k = int(np.random.choice(inferior_choice))+1
                else:
                    choice_k = int(np.random.choice(superior_choice))+1


            if prob < p_jk[j-1][choice_k-1]:
                Patients[j][choice_k].append(1)
                acc_success_j[choice_k-1] +=1
                Success[j-1][choice_k-1] +=1
            else:
                Patients[j][choice_k].append(0)

            acc_trials_j[choice_k-1] +=1
            Acc_Trials[j-1][choice_k-1] +=1

        # Update mu matrix
        for k in range(len(Mu[j-1])):
            if acc_trials_j[k] == 0:
                Mu[j-1][k] = 0
            else:
```

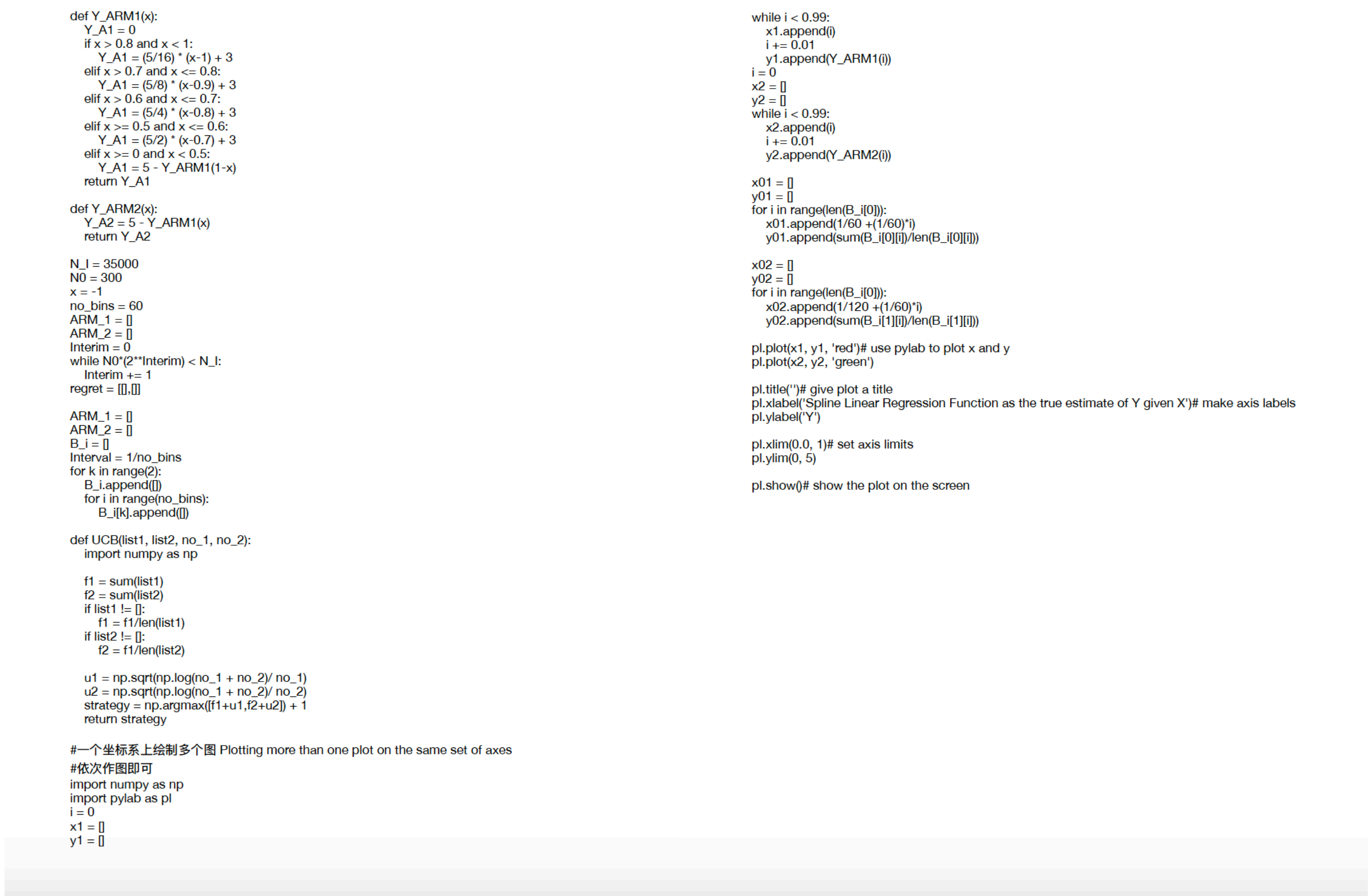
```
                Mu[j-1][k] = (acc_success_j[k])/(acc_trials_j[k])
    for j in Patients:
        # Update mu matrix, decide elimination
        GLR_stats = 0
        GLR_stats = GLR(Success,Acc_Trials,J,K)
        for k in range(K):
            if GLR_stats[j-1][k] >= a_alpha_index:

                elimination[j-1][k] = 'none'

        for k in range(K):
            if elimination[j-1][k] != 'none':
                if abs(Mu[j-1][k]-max(Mu[j-1])) <= (sum(acc_trials_j))**(-0.4):
                    elimination[j-1][k] = 'superior'
                else:
                    elimination[j-1][k] = 'inferior'

        suc += (sum(acc_success_j))
        tot += (sum(acc_trials_j))

    return(Patients,Acc_Trials,Success,Mu, suc/tot, elimination)

#试10000次
count = 0
for simulation in range(400):
    J = 3
    K = 3
    n_interval = 200
    I = 5
    epsilon = 0.1
    a_alpha = 2.592155653417598
    p_jk = [[0.7,0.69,0.69], [0.69,0.7,0.69],[0.69,0.69,0.7]]

    success_jk = []
    for j in range(J):
        success_jk.append([])
        for k in range(K):
            success_jk[j].append(0)

    trials_jk = []
    for j in range(J):
        trials_jk.append([])
        for k in range(K):
            trials_jk[j].append(0)

    mu_jk = []
    for j in range(J):
        mu_jk.append([])
        for k in range(K):
            mu_jk[j].append(0)

    eli = []
    for j in range(J):
        eli.append([])
        for k in range(K):
            eli[j].append('start')

    for i in range(I):
        Patients,trials_jk,success_jk,mu_jk,rate,eli =
AR_elimination_loop(i,J,K,epsilon,n_interval,trials_jk,success_jk,mu_jk,a_alpha,eli)
```

```
    for j in range(J):
        for k in range(K):
            cul_suc[j][k] += success_jk[j][k]
            cul_trial[j][k] += trials_jk[j][k]
    for j in range(J):
            if eli[j][j] == 'none':
                type_I[j] += 1
    count +=1

print(ave_suc)
print(ave_trial)
sum(ave_trial[0])+sum(ave_trial[1])+sum(ave_trial[2])
```

A Snapshot of Codes

Multi-Arm Bandit with Side Information

Non-Parametric UCB By Xiaocheng Li

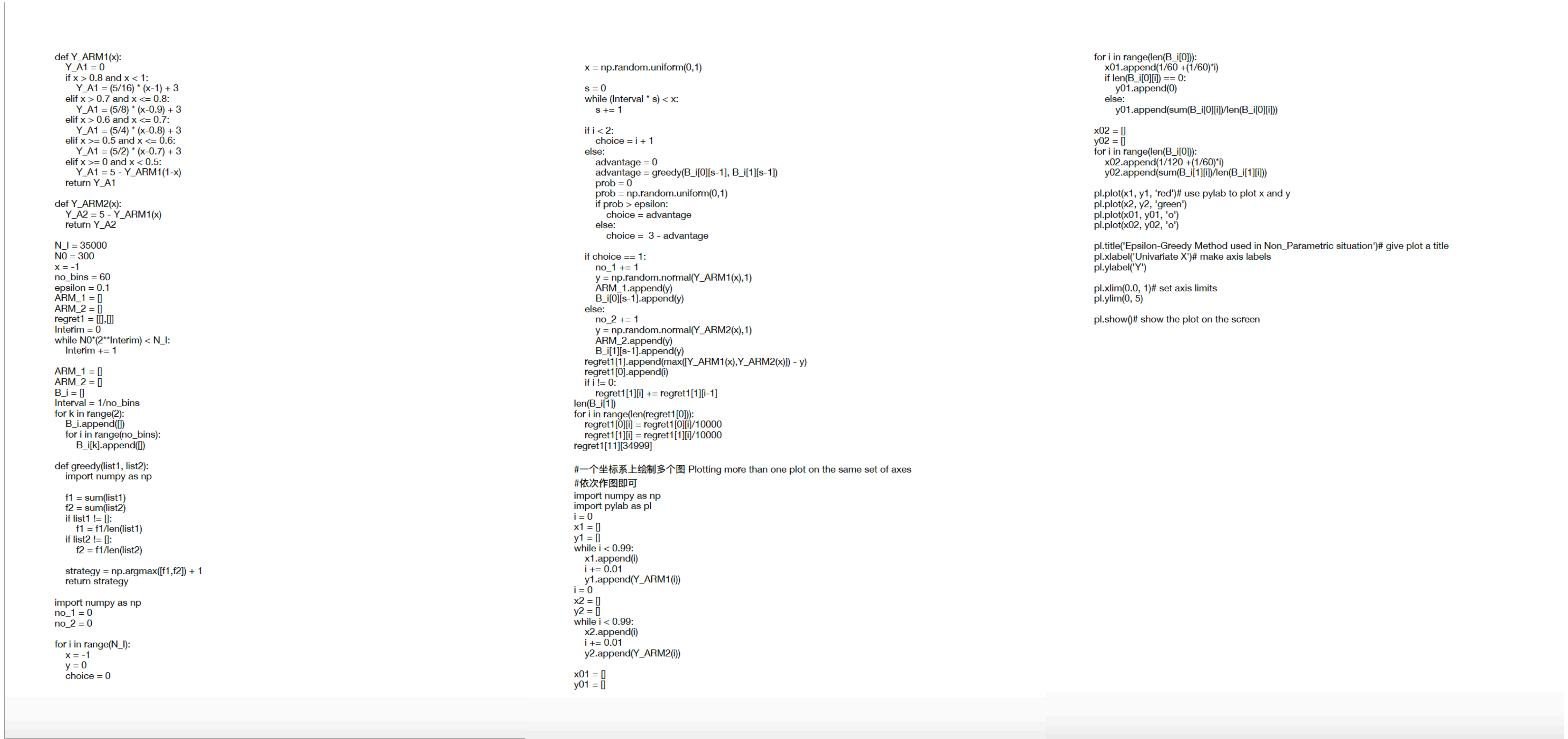


```
def Y_ARM1(x):
    Y_A1 = 0
    if x > 0.8 and x < 1:
        Y_A1 = (5/16) * (x-1) + 3
    elif x > 0.7 and x <= 0.8:
        Y_A1 = (5/8) * (x-0.9) + 3
    elif x > 0.6 and x <= 0.7:
        Y_A1 = (5/4) * (x-0.8) + 3
    elif x >= 0.5 and x <= 0.6:
        Y_A1 = (5/2) * (x-0.7) + 3
    elif x >= 0 and x < 0.5:
        Y_A1 = 5 - Y_ARM1(1-x)
    return Y_A1

def Y_ARM2(x):
    Y_A2 = 5 - Y_ARM1(x)
    return Y_A2

N_I = 35000
N0 = 300
x = -1
no_bins = 60
ARM_1 = []
ARM_2 = []
Interim = 0
while N0*(2**Interim) < N_I:
    Interim += 1
regret = [[],[]]

ARM_1 = []
ARM_2 = []
B_i = []
Interval = 1/no_bins
for k in range(2):
    B_i.append([])
    for i in range(no_bins):
        B_i[k].append([])

def UCB(list1, list2, no_1, no_2):
    import numpy as np

    f1 = sum(list1)
    f2 = sum(list2)
    if list1 != []:
        f1 = f1/len(list1)
    if list2 != []:
        f2 = f1/len(list2)

    u1 = np.sqrt(np.log(no_1 + no_2)/ no_1)
    u2 = np.sqrt(np.log(no_1 + no_2)/ no_2)
    strategy = np.argmax([f1+u1,f2+u2]) + 1
    return strategy

#一个坐标系上绘制多个图 Plotting more than one plot on the same set of axes
#依次作图即可
import numpy as np
import pylab as pl
i = 0
x1 = []
y1 = []
while i < 0.99:
    x1.append(i)
    i += 0.01
    y1.append(Y_ARM1(i))
i = 0
x2 = []
y2 = []
while i < 0.99:
    x2.append(i)
    i += 0.01
    y2.append(Y_ARM2(i))

x01 = []
y01 = []
for i in range(len(B_i[0])):
    x01.append(1/60 +(1/60)*i)
    y01.append(sum(B_i[0][i])/len(B_i[0][i]))

x02 = []
y02 = []
for i in range(len(B_i[0])):
    x02.append(1/120 +(1/60)*i)
    y02.append(sum(B_i[1][i])/len(B_i[1][i]))

pl.plot(x1, y1, 'red')# use pylab to plot x and y
pl.plot(x2, y2, 'green')

pl.title('')# give plot a title
pl.xlabel('Spline Linear Regression Function as the true estimate of Y given X')# make axis labels
pl.ylabel('Y')

pl.xlim(0.0, 1)# set axis limits
pl.ylim(0, 5)

pl.show()# show the plot on the screen
```

A Snapshot of Codes

Multi-Arm Bandit with Side Information

Non-Parametric epsilon-greedy By Xiaocheng Li

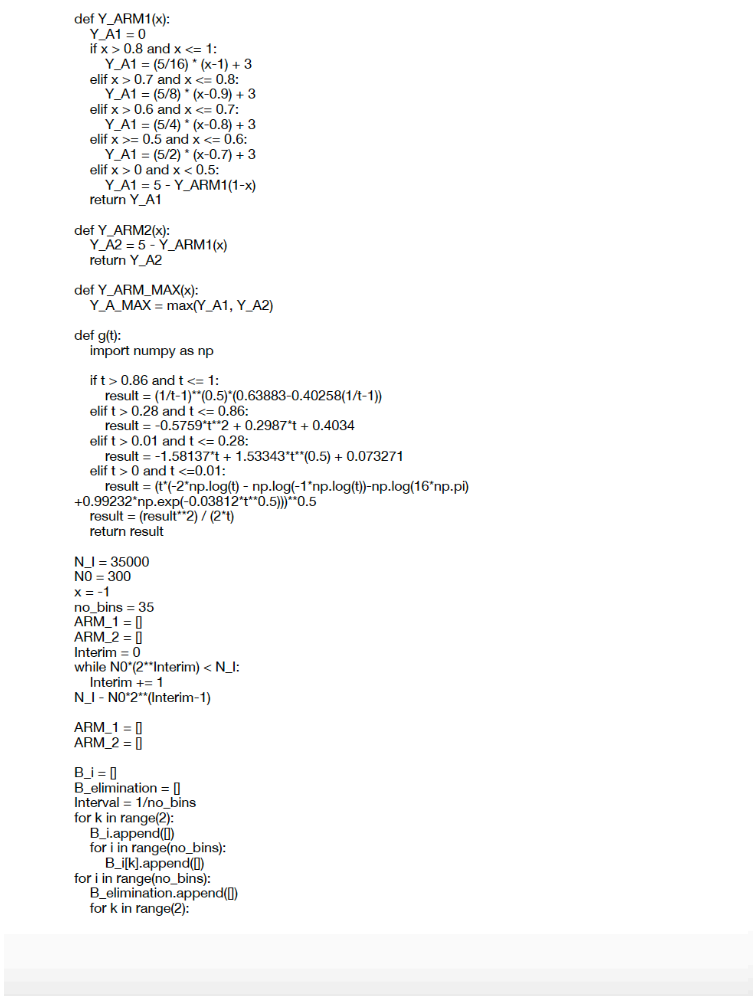


```
def Y_ARM1(x):
    Y_A1 = 0
    if x > 0.8 and x < 1:
        Y_A1 = (5/16) * (x-1) + 3
    elif x > 0.7 and x <= 0.8:
        Y_A1 = (5/8) * (x-0.9) + 3
    elif x > 0.6 and x <= 0.7:
        Y_A1 = (5/4) * (x-0.8) + 3
    elif x >= 0.5 and x <= 0.6:
        Y_A1 = (5/2) * (x-0.7) + 3
    elif x >= 0 and x < 0.5:
        Y_A1 = 5 - Y_ARM1(1-x)
    return Y_A1

def Y_ARM2(x):
    Y_A2 = 5 - Y_ARM1(x)
    return Y_A2

N_I = 35000
N0 = 300
x = -1
no_bins = 60
epsilon = 0.1
ARM_1 = []
ARM_2 = []
regret1 = [[],[]]
Interim = 0
while N0*(2**Interim) < N_I:
    Interim += 1

ARM_1 = []
ARM_2 = []
B_i = []
Interval = 1/no_bins
for k in range(2):
    B_i.append([])
    for i in range(no_bins):
        B_i[k].append([])

def greedy(list1, list2):
    import numpy as np

    f1 = sum(list1)
    f2 = sum(list2)
    if list1 != []:
        f1 = f1/len(list1)
    if list2 != []:
        f2 = f1/len(list2)

    strategy = np.argmax([f1,f2]) + 1
    return strategy

import numpy as np
no_1 = 0
no_2 = 0

for i in range(N_I):
    x = -1
    y = 0
    choice = 0

    x = np.random.uniform(0,1)

    s = 0
    while (Interval * s) < x:
        s += 1

    if i < 2:
        choice = i + 1
    else:
        advantage = 0
        advantage = greedy(B_i[0][s-1], B_i[1][s-1])
        prob = 0
        prob = np.random.uniform(0,1)
        if prob > epsilon:
            choice = advantage
        else:
            choice =  3 - advantage

    if choice == 1:
        no_1 += 1
        y = np.random.normal(Y_ARM1(x),1)
        ARM_1.append(y)
        B_i[0][s-1].append(y)
    else:
        no_2 += 1
        y = np.random.normal(Y_ARM2(x),1)
        ARM_2.append(y)
        B_i[1][s-1].append(y)
    regret1[1].append(max([Y_ARM1(x),Y_ARM2(x)]) - y)
    regret1[0].append(i)
    if i != 0:
        regret1[1][i] += regret1[1][i-1]
len(B_i[1])
for i in range(len(regret1[0])):
    regret1[0][i] = regret1[0][i]/10000
    regret1[1][i] = regret1[1][i]/10000
regret1[11][34999]

#一个坐标系上绘制多个图 Plotting more than one plot on the same set of axes
#依次作图即可
import numpy as np
import pylab as pl
i = 0
x1 = []
y1 = []
while i < 0.99:
    x1.append(i)
    i += 0.01
    y1.append(Y_ARM1(i))
i = 0
x2 = []
y2 = []
while i < 0.99:
    x2.append(i)
    i += 0.01
    y2.append(Y_ARM2(i))

x01 = []
y01 = []
for i in range(len(B_i[0])):
    x01.append(1/60 +(1/60)*i)
    if len(B_i[0][i]) == 0:
        y01.append(0)
    else:
        y01.append(sum(B_i[0][i])/len(B_i[0][i]))

x02 = []
y02 = []
for i in range(len(B_i[0])):
    x02.append(1/120 +(1/60)*i)
    y02.append(sum(B_i[1][i])/len(B_i[1][i]))

pl.plot(x1, y1, 'red')# use pylab to plot x and y
pl.plot(x2, y2, 'green')
pl.plot(x01, y01, 'o')
pl.plot(x02, y02, 'o')

pl.title('Epsilon-Greedy Method used in Non_Parametric situation')# give plot a title
pl.xlabel('Univariate X')# make axis labels
pl.ylabel('Y')

pl.xlim(0.0, 1)# set axis limits
pl.ylim(0, 5)

pl.show()# show the plot on the screen
```

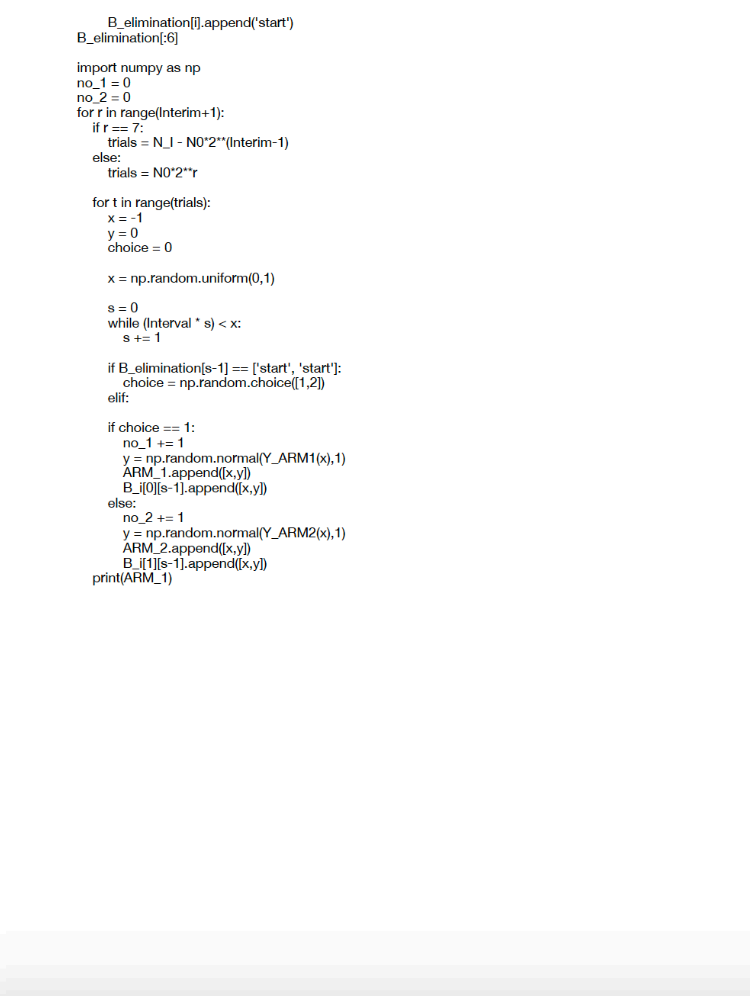
A Snapshot of Codes

Multi-Arm Bandit with Side Information

Non-Paremetric AR Method By Xiaocheng Li (Working)



```
def Y_ARM1(x):
    Y_A1 = 0
    if x > 0.8 and x <= 1:
        Y_A1 = (5/16) * (x-1) + 3
    elif x > 0.7 and x <= 0.8:
        Y_A1 = (5/8) * (x-0.9) + 3
    elif x > 0.6 and x <= 0.7:
        Y_A1 = (5/4) * (x-0.8) + 3
    elif x >= 0.5 and x <= 0.6:
        Y_A1 = (5/2) * (x-0.7) + 3
    elif x > 0 and x < 0.5:
        Y_A1 = 5 - Y_ARM1(1-x)
    return Y_A1

def Y_ARM2(x):
    Y_A2 = 5 - Y_ARM1(x)
    return Y_A2

def Y_ARM_MAX(x):
    Y_A_MAX = max(Y_A1, Y_A2)

def g(t):
    import numpy as np

    if t > 0.86 and t <= 1:
        result = (1/t-1)**(0.5)*(0.63883-0.40258(1/t-1))
    elif t > 0.28 and t <= 0.86:
        result = -0.5759*t**2 + 0.2987*t + 0.4034
    elif t > 0.01 and t <= 0.28:
        result = -1.58137*t + 1.53343*t**(0.5) + 0.073271
    elif t > 0 and t <=0.01:
        result = (t*(-2*np.log(t) - np.log(-1*np.log(t))-np.log(16*np.pi)
+0.99232*np.exp(-0.03812*t**0.5)))**0.5
    result = (result**2) / (2*t)
    return result

N_I = 35000
N0 = 300
x = -1
no_bins = 35
ARM_1 = []
ARM_2 = []
Interim = 0
while N0*(2**Interim) < N_I:
    Interim += 1
N_I - N0*2**(Interim-1)

ARM_1 = []
ARM_2 = []

B_i = []
B_elimination = []
Interval = 1/no_bins
for k in range(2):
    B_i.append([])
    for i in range(no_bins):
        B_i[k].append([])
for i in range(no_bins):
    B_elimination.append([])
    for k in range(2):
```



```
        B_elimination[i].append('start')
B_elimination[:6]

import numpy as np
no_1 = 0
no_2 = 0
for r in range(Interim+1):
    if r == 7:
        trials = N_I - N0*2**(Interim-1)
    else:
        trials = N0*2**r

    for t in range(trials):
        x = -1
        y = 0
        choice = 0

        x = np.random.uniform(0,1)

        s = 0
        while (Interval * s) < x:
            s += 1

        if B_elimination[s-1] == ['start', 'start']:
            choice = np.random.choice([1,2])
        elif:

        if choice == 1:
            no_1 += 1
            y = np.random.normal(Y_ARM1(x),1)
            ARM_1.append([x,y])
            B_i[0][s-1].append([x,y])
        else:
            no_2 += 1
            y = np.random.normal(Y_ARM2(x),1)
            ARM_2.append([x,y])
            B_i[1][s-1].append([x,y])
    print(ARM_1)
```