STAT6011/7611/6111/3317
COMPUTATIONAL STATISTICS (2016 Fall)
Assignment 4

Kei Ikegami (u3535947)

November 22, 2016

1

The code is below. I denote the 'Fair' by 0 and 'Loaded' by 1 in this code.

Listing 1: problem 1

```
import numpy as np

def vitervi(transition):
    tosses = "2513263442124633665655356145665236655613263456214432352132634614354 21"
    prob_loaded = {"1": 1/12, "2": 1/12, "3": 1/12, "4": 1/12, "5":1/3, "6":1/3}
    prob_fair = {"1":1/6, "2":1/6, "3":1/6, "4":1/6, "5":1/6, "6":1/6}
    score_loaded = np.ones(len(tosses))
    score_fair = np.ones(len(tosses))
    score_loaded[0] = 0.5 * prob_loaded[tosses[0]]
    score_fair[0] = 0.5 * prob_fair[tosses[0]]

    for i in range(1,len(tosses)):
        score_loaded[i] = max(score_loaded[i-1] * transition[1,1], score_fair[i-1] * transition[1,0]) * prob_loaded[tosses[i
            ]]
        score_fair[i] = max(score_loaded[i-1] * transition[0,1], score_fair[i-1] * transition[0,0]) * prob_fair[tosses[i]]

    state_estimation = np.ones(len(tosses))
    for n, j in enumerate(score_loaded - score_fair):
        if j < 0:
            state_estimation[n] = 0

    return(state_estimation)

transition1 = np.array([[0.8, 0.3], [0.2, 0.7]])
transition2 = np.array([[0.9, 0.15], [0.1, 0.85]])
transition3 = np.array([[0.95, 0.05], [0.6, 0.4]])
transition4 = np.array([[0.5, 0.5], [0.5, 0.5]])

vitervi(transition1)
vitervi(transition2)
vitervi(transition3)
vitervi(transition4)
```

And the results are as follows.

(a) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

(b) [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

(c) [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

(d) [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0]

2

Listing 2: problem 2

```
import numpy as np
from scipy.special import eval_hermite
from scipy.special import hermite
from scipy.special import h_roots
from scipy.special import he_roots
from scipy.stats import norm

def GH(m):
    points = h_roots(m)[0]
    weights = h_roots(m)[1]
    return sum([weights[i] * np.exp(points[i]**2) * norm.pdf(points[i]) for i in range(len(points))])

def GH_weights(m):
    points = h_roots(m)[0]
    weights = h_roots(m)[1]
    return [weights[i] * np.exp(points[i]**2) for i in range(len(points))]

a = [5, 10, 20, 30]

# approximation result
for m in a:
    print(GH(m))

# weights
for m in a:
    print(GH_weights(m))

# points
for m in a:
    print(h_roots(m)[0])
```

2.1

I prove the integral of standard normal probability distribution function is equal to 1.

2.2

I use the above code to compute the approximation of the integral.

3

Listing 3: problem 3

```
import numpy as np
from scipy.special import p_roots
from scipy.special import u_roots
from scipy.special import t_roots

def f(x):
    return (x**9) / np.sqrt(x**2 + 1)

def GL(m, a, b):
    points = p_roots(m)[0]
    weights = p_roots(m)[1]
    return (b-a)*sum([weights[i]*f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(points))])/2

def Cheby1(m, a, b):
    points = t_roots(m)[0]
    weights = t_roots(m)[1]
    return (b-a)*sum([weights[i]*np.sqrt(1-(points[i])**2) * f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(points))])/2

def Cheby2(m, a, b):
    points = u_roots(m)[0]
    weights = u_roots(m)[1]
```

```
    return (b-a)*sum([(weights[i]/np.sqrt(1-(points[i])**2)) * f((b-a)*points[i]/2 + (b+a)/2) for i in range(len(
        points))])/2

a = [5, 10]

for m in a:
    # approximation results
    print(GL(m))
    print(Cheby1(m))
    print(Cheby2(m))

    # nodes
    print(p_roots(m)[0])
    print(t_roots(m)[0])
    print(u_roots(m)[0])

    # weights
    print(p_roots(m)[1])
    print(t_roots(m)[1])
    print(u_roots(m)[1])
```

4

Listing 4: problem 4

```
import numpy as np
from scipy.stats import binom

n = 25

iteration = 100
values = np.ones((iteration, 3))
initial_value = [1/3, 1/3, 1/3]
values[0, :] = initial_value

for ite in range(1, iteration):
    q_a = (2*values[ite-1, 2]) / (2*values[ite-1, 2] + values[ite-1, 0])
    q_b = (2*values[ite-1, 2]) / (2*values[ite-1, 2] + values[ite-1, 1])
    under = sum([binom.pmf(i, n, q_a) * binom.pmf(j, n, q_b) * (3*n - i) for i in range(n+1) for j in range(n+1)])
    upper = sum([binom.pmf(i, n, q_a) * binom.pmf(j, n, q_b) * (2*n + i + j) for i in range(n+1) for j in range(n+1)
        ])
    values[ite, 0] = 1/(2+(upper/under))
    values[ite, 1] = 1/(2+(upper/under))
    values[ite, 2] = 1 - 2*values[ite, 0]
```

5

Listing 5: problem 5

```
# data generate
import numpy as np

I = 100
J = 2
beta_0 = -1
beta_1 = 1
sigma_u = 0.5
sigma_eps = 1

np.random.seed(12345)

value_x = np.random.normal(0, 1, (I, J))
u = np.random.normal(0, sigma_u, I)
value_u = np.ones((I, J))
value_u[:, 0] = u
value_u[:, 1] = u
value_eps = np.random.normal(0, sigma_eps, (I, J))
value_y = beta_0 + beta_1 * value_x + value_u + value_eps
```

```
# implement EM algorithm
# para = (beta_0, beta_1, sigma_eps, sigma_u)
# sigma is variance
iteration = 500
XY = sum(sum(value_y * value_x))
X = sum(sum(value_x))
Y = sum(sum(value_y))
X_2 = sum(sum(value_x * value_x))

def xE(E_t):
    return sum(sum(value_x * np.array([E_t, E_t]).T))

def eps_right(E_t, beta0, beta1):
    return (sum(sum((value_y - beta0-beta1*value_x)**2)) + J * sum(E_t**2) - 2*sum(sum((value_y - beta0 - beta1*value_x) * np.array([E_t, E_t]).T)))/(I*J)

initial = [0, 0, 5, 5]
E_t = np.ones(I)
estimation = np.ones((iteration, 4))
estimation[0, :] = initial

for ite in range(1,iteration):
    u = estimation[ite-1, 3]
    eps = estimation[ite-1, 2]
    beta0 = estimation[ite-1, 0]
    beta1 = estimation[ite-1, 1]
    for i in range(I):
        E_t[i] = u*sum([value_y[i, j] - beta0 - beta1*value_x[i, j] for j in range(J)]) / (J*u + eps)
    V_t = eps*u/(J*u + eps)
    estimation[ite, 1] = (I*J *XY - Y*X - J * (I*xE(E_t) - sum(E_t)*X))/(I*J*X_2 - X**2)
    estimation[ite, 0] = (Y-beta1*X - J* sum(E_t))/(I*J)
    estimation[ite, 2] = V_t + eps_right(E_t, estimation[ite-1, 0], estimation[ite-1, 1])
    estimation[ite, 3] = V_t + sum(E_t**2)/I

# change the last variance into standard deviation
estimation[:, 2] = np.sqrt(estimation[:, 2])
estimation[:, 3] = np.sqrt(estimation[:, 3])

# compute the bias and std
bias = np.average(estimation, axis = 0) - [-1, 1, 1, 0.5]
std = np.std(estimation, axis = 0)
print("bias :", bias)
print("std :", std)
```