```python
import numpy as np
import numpy.random as npr
import scipy.stats as stats
import matplotlib.pyplot as plt



## Define functions for 1

def G(i):
    num = (A ** i) / (np.math.factorial(i))
    return num

def TP(i):
    num = (A ** i) / (np.math.factorial(i))
    denom = np.array([(A ** j) / (np.math.factorial(j)) for j
in range((n+1))])
    return num / np.sum(denom)

def chisquared(observed, expected):
    nclass = len(expected)
    T = sum([(observed[i]-expected[i])**2/expected[i] for i in
 range(nclass)])
    return T

# Simple metropolis hastings - 10000 iterations
A = 8
n = 10
N = 10000
Burn_in = 1000
X = 0
Xi_simple = np.zeros(N)
n_sample_iter = 100

for i in range(N*n_sample_iter+Burn_in):

    Y = np.sign(npr.rand() - 1/2) + X # Jump -1 or 1
    # Y = np.int(npr.rand() * (n+1)) # Jump to any of the
other values

    if Y > n:
        Y = 0
    elif Y < 0:
        Y = n

    ratio = G(Y) / G(X)

    if ratio >= 1:
        X = Y
```

```
    else:
        U = npr.rand()
        if ratio > U:
            X = Y
        else:
            X = X

    if i >= Burn_in and i % n_sample_iter == 0:
        Xi_simple[(i-Burn_in) // n_sample_iter] = X

# Make figure
plt.figure()
plt.hist(Xi_simple,density=True,bins=11,ec='k')
plt.show()

ps = np.array([TP(i) for i in range((n+1))])

Theoretical = ps * N
Result = np.array([np.count_nonzero(Xi_simple == j) for j in
range(n+1)])

ChiSq = chisquared(Result,Theoretical)
pval = 1 - stats.chi2.cdf(ChiSq,n-1)


## Exercise 2
N = 20000
Burn_in = 1000
Xi_direct = np.zeros((N,2))
n_sample_iter = 100
A1 = 4
A2 = 4
n = 10

X1 = 5
X2 = 5

# Normalization constant calculated in Maple
K = 3830591/1575

def G(i,j):
    return (A1 ** i) / np.math.factorial(i) * (A2 ** j) / np.
math.factorial(j)

def P(i,j):
    return (A1 ** i) / np.math.factorial(i) * (A2 ** j) / np.
math.factorial(j)  * 1 / K


# Test if normalization constant from crappy Maple is actually
```

```
 normalization constant and also save probas

Sum = 0
Probas = np.zeros((n+1,n+1))

# Do for j

for i in range(10+1):
    for j in range(11-i):
        if j >= 0:
            p = P(i,j)
            Sum += p
            Probas[i,j] = p


# Metropolis hastings - both coordinates at the same time
for i in range(N*n_sample_iter+Burn_in):

    # Sample the jumps [-1,0,1] uniformly
    Y1 = np.int(npr.rand() * 3) - 1 + X1
    Y2 = np.int(npr.rand() * 3) - 1 + X2

    # Make special jump for the point (0,0) to ensure
symmetric jumping distribution
    if X1 == 0 and X2 == 0:
        if Y1 + Y2 < 0:
            Y1 = 5
            Y2 = 5
        elif Y1 + Y2 == 0:
            Y1 = 0
            Y2 = 0

    # Make special jump for X1 = 5 and X2 = 5
    elif X1 == 5 and X2 == 5 and (Y1 + Y2 > n):
        Y1 = 0
        Y2 = 0

    # Make looping markov chain for the other cases where we
interchange coordinates for jump
    elif (Y1 + Y2) > n or Y1 < 0 or Y2 < 0 or Y1 > n or Y2 > n
:
        Y1 = X2
        Y2 = X1

    ratio = G(Y1,Y2) / G(X1,X2)
    if ratio >= 1:
        X1 = Y1
        X2 = Y2
    else:
        U = npr.rand()
```

```
        if ratio > U:
            X1 = Y1
            X2 = Y2
        else:
            X1 = X1
            X2 = X2
    if i >= Burn_in and i % n_sample_iter == 0:
        Xi_direct[(i-Burn_in) // n_sample_iter,:] = X1,X2

# Metropolis Hastings each coordinate
N = 20000
Burn_in = 40000
Xi_coordinate = np.zeros((N,2))
n_sample_iter = 300
A1 = 4
A2 = 4
n = 10

X1 = 5
X2 = 5


for i in range(N*n_sample_iter+Burn_in):

    # Sample the jumps [-1,0,1] uniformly
    Y1 = X1
    Y2 = X2

    if i % 2 == 0:
        Y1 = np.int(npr.rand() * 3) - 1 + X1
    else:
        Y2 = np.int(npr.rand() * 3) - 1 + X2

    # Make special jump for the point (0,0) to ensure
symmetric jumping distribution
    if X1 == 0 and X2 == 0:
        if Y1 + Y2 < 0:
            Y1 = 5
            Y2 = 5
        elif Y1 + Y2 == 0:
            Y1 = 0
            Y2 = 0

    # Make special jump for X1 = 5 and X2 = 5
    elif X1 == 5 and X2 == 5 and (Y1 + Y2 > n):
        Y1 = 0
        Y2 = 0

    # Make looping markov chain for the other cases where we
interchange coordinates for jump
```

```
    elif (Y1 + Y2) > n or Y1 < 0 or Y2 < 0 or Y1 > n or Y2 > n
:
        Y1 = X2
        Y2 = X1
    ratio = G(Y1, Y2) / G(X1, X2)
    if ratio >= 1:
        X1 = Y1
        X2 = Y2
    else:
        U = npr.rand()
        if ratio > U:
            X1 = Y1
            X2 = Y2
        else:
            X1 = X1
            X2 = X2
    if i >= Burn_in and i % n_sample_iter == 0:
        Xi_coordinate[(i-Burn_in) // n_sample_iter,:] = X1,X2


#### Now we do coordinate wise Gibbs sampling
N = 20000
Burn_in = 1000
Xi_gibbs = np.zeros((N,2))
n_sample_iter = 100
A1 = 4
A2 = 4
n = 10

X1 = 5
X2 = 5

Y1 = X1
Y2 = X2

### Gibbs sampling
def condist(i,j):
    num = (4**i) / np.math.factorial(i)
    denom = np.array([(4**k) / np.math.factorial(k) for k in
range((n+1) - j)])
    return num / np.sum(denom)

### Make matrix of conditional distributions
condmatrix = np.zeros((n+1,n+1))

for i in range(n+1):
    for j in range((n+1)-i):
        condmatrix[i,j] = condist(i,j)
### Cumulative distribution function for sampling
cummatrix = np.cumsum(condmatrix,0)
```

```
for i in range(N*n_sample_iter+Burn_in):

    # Gibbs sample each coordinate - start with X1
    U1 = npr.rand()
    cumdist = cummatrix[:,X2]
    X1 = np.nonzero(cumdist >= U1)[0][0]

    U2 = npr.rand()
    cumdist = cummatrix[:,X1]
    X2 = np.nonzero(cumdist >= U2)[0][0]

    if i >= Burn_in and i % n_sample_iter == 0:
        Xi_gibbs[(i-Burn_in) // n_sample_iter,:] = X1,X2


### Perform T-testing for all 3 multi-dimensional methods
predicted = Probas * N

observed_direct = np.zeros((n+1,n+1))
observed_coordinate = np.zeros((n+1,n+1))
observed_gibbs = np.zeros((n+1,n+1))

for i in range(N):
    coord = Xi_direct[i,:]
    X = np.int(coord[0])
    Y = np.int(coord[1])
    observed_direct[X,Y] += 1

    coord = Xi_coordinate[i,:]
    X = np.int(coord[0])
    Y = np.int(coord[1])
    observed_coordinate[X,Y] += 1

    coord = Xi_gibbs[i,:]
    X = np.int(coord[0])
    Y = np.int(coord[1])
    observed_gibbs[X,Y] += 1

T_gibbs = 0
T_direct = 0
T_coordinate = 0
df = 66 - 1

for i in range((n+1)):
    for j in range((10-i)):
        T_gibbs += (observed_gibbs[i,j] - predicted[i,j]) ** 2
 / predicted[i,j]
        T_direct += (observed_direct[i,j] - predicted[i,j]) **
 2 / predicted[i,j]
```

```
        T_coordinate += (observed_coordinate[i,j] - predicted[
i,j]) ** 2 / predicted[i,j]

p_direct = 1 - stats.chi2.cdf(T_direct,df)
p_coordinate = 1 - stats.chi2.cdf(T_coordinate,df)
p_gibbs = 1 - stats.chi2.cdf(T_gibbs,df)
```