In [ ]:

```
import numpy as np
import math
import numpy.random as rnd


def metropolis_hastings(g,N,m, burn_in = None):
    burn_in = burn_in if burn_in is not None else N // 10
    #X and Y are random integers from 0 to m
    U = np.random.uniform(0,1,N + burn_in)
    X = np.zeros(N + burn_in,dtype = int)
    X[0] = 3
    #prob_of_sampling = np.minimum(g(X)/g(Y),np.ones(N))
    #sample X with probability prob_of_sampling
    for i in range(1,N+burn_in):
        x = X[i-1]
        y = np.random.randint(0,m+1)

        if U[i] <= min(g(y) / g(x),1) :
            X[i] = y
        else:
            X[i] = x

    # X = np.where(U<prob_of_sampling,X,Y)
    X = X[burn_in:]
    return X

def truncated_poisson_samples(lam, low, high, size=1, numvars = 1):
    samples = []
    while len(samples) < size:
        x = np.random.poisson(lam, numvars)
        if low <= x <= high:
            samples.append(x)
    return np.array(samples)

def y_sampling_function(m):
        def y_sampling():
            y1 = np.random.randint(0,m+1)
            y2 = np.random.randint(0,m+1-y1)
            #create y as (y1,y2) or (y2,y1) with equal probability
            return (y1,y2) if np.random.uniform(0,1) < 0.5 else (y2,y1)
        return y_sampling


def metropolis_hastings_joint(g_joint,N, burn_in = None, y_sampling_func = y_sampling_function(10)):
    burn_in = burn_in if burn_in is not None else N

    #X and Y are random integers from 0 to m
    U = np.random.uniform(0,1,N + burn_in)
    X = np.zeros((N + burn_in,2),dtype = float)
    X[0] = (1.0,1.0)
    #prob_of_sampling = np.minimum(g(X)/g(Y),np.ones(N))
    #sample X with probability prob_of_sampling
    for i in range(1,N+burn_in):
        x = X[i-1]
        y = y_sampling_func()
        if U[i] <= min(g_joint(y[0],y[1]) / g_joint(x[0],x[1]),1) :
            X[i] = y
        else:
            X[i] = x
    # X = np.where(U<prob_of_sampling,X,Y)
    X = X[burn_in:]
    return X

def Gibbs(As, n, x0, m):
    xs = [x0[0]]
    ys = [x0[1]]
    A1 = As[0]
    A2 = As[1]
    for k in range(1,n):
        # Generate i and sample from j
        i = xs[k-1]
        num_classes_j = int(m - i + 1)
        ps = np.zeros(num_classes_j)
```

```
        k = 0
        for j in range(num_classes_j):
            ps[j] = A2**j / math.factorial(j)
            k += A2**j / math.factorial(j)
        ps /= k
        j = rnd.choice(a=np.arange(num_classes_j), size= 1, p = ps)[0]

        ys.append(j)
        # Newest j has already been found
        num_classes_i = int(m-j + 1)
        ps = np.zeros(num_classes_i )
        k = 0
        for i in range(num_classes_i):
            ps[i] = A2**i / math.factorial(i)
            k += A2**i / math.factorial(i)
        ps /= k
        i = rnd.choice(a=np.arange(num_classes_i), size=1, p = ps)[0]
        xs.append(i)
    return np.array(xs), np.array(ys)
```

In [ ]:

```
import numpy as np
from scipy.stats import uniform
from scipy.stats import norm
from scipy.stats import chi2

def LCG(xval, M, a, c, N):
    x = np.zeros(N)
    for i in range(N):
        xval = (a*xval + c) % M
        x[i] = xval
    x/=M
    return x

def KS_test(randn):
    Ftrue = np.arange(0,1,1/len(randn))
    #import uniform distribution cdf
    F = uniform.cdf(randn)
    #calculate max difference
    D = np.max(np.abs(Ftrue - F))
    #calculate p value
    n = len(randn)
    pval = 1 - np.exp(-2*n*D**2)

    return D, pval

#calculate chi squared
def chisquare_test(randn, k):
    n = len(randn)
    p = 1/k
    test = 0
    for i in range(k):
        xval = randn[(i/k < randn)*(randn < (i+1)/k)]
        ni = len(xval)
        test += (ni - n*p)**2/(n*p)
    pval = 1 - chi2.cdf(test, k-1)
    return test, pval

def run_test_1(randn):
    #median value
    median = np.median(randn)
    #number of observations below median
    possamps = randn < median
    negsamps = randn > median
    posruns = 0
    negruns = 0
    n1 = np.sum(possamps)
    n2 = np.sum(negsamps)
    n = n1+n2
    for i in range(len(randn)):
        if i == 0:
            continue
        if possamps[i] != possamps[i-1] and possamps[i] == True:
            posruns += 1
        if negsamps[i] != negsamps[i-1] and negsamps[i] == True:
            negruns += 1
```

```
    T = posruns + negruns
    #normal cdf

    mean = 2*n1*n2/n + 1
    var = np.exp(np.log(2)+np.log(n1)+np.log(n2)+np.log(2*n1*n2 - n)-np.log(n**2)-np.log((n-1)))
    Z = (T-mean)/np.sqrt(var)
    pval = 1 - norm.cdf(Z)
    return T, pval

def run_test_2(randn):
    # up/down
    run_lengths = []
    current_run_length = 1
    for i in range(1, len(randn)):
        if randn[i] > randn[i-1]:
            current_run_length = min(current_run_length + 1, 6)
        else:
            run_lengths.append(current_run_length)
            current_run_length = 1
    #get vector of count for each run length
    R, _ = np.histogram(run_lengths, bins=6)

    A  = np.array([
        [4529.4, 9044.9, 13568, 18091, 22615, 27892],
        [9044.9, 18097, 27139, 36187, 45234, 55789],
        [13568, 27139, 40721, 54281, 67852, 83685],
        [18091, 36187, 54281, 72414, 90470, 111580],
        [22615, 45234, 67852, 90470, 113262, 139476],
        [27892, 55789, 83685, 111580, 139476, 172860]
        ])
    B = np.array([1/6, 5/24, 11/120, 19/720, 29/5040, 1/840])
    n = len(randn)
    Z = (R-B*n).T@A@(R-B*n)/(n-6)
    #chisquare test
    pval = 1 - chi2.cdf(Z, 6)
    return Z, pval

def run_test_3(randn):
    n = len(randn)
    updown = np.zeros(n-1, dtype=bool)
    for i in range(1, n):
        updown[i-1] = randn[i] > randn[i-1]
    #count number of runs
    runs = []
    current_run = 1
    for i in range(1, n-1):
        if updown[i] != updown[i-1]:
            runs.append(current_run)
            current_run = 1
        else:
            current_run += 1
    # number of unique runs
    X = len(runs)
    Z = (X - (2*n-1)/3)/np.sqrt((16*n-29)/90)
    p = 2*(1 - norm.cdf(abs(Z)))
    return Z, p

def correlation_coefficient(randn, h=2):
    c = np.sum(randn[:-h]*randn[h:])/len(randn)
    Z = (c - 1/4)/np.sqrt(7/(144*len(randn)))
    p = 2*(1 - norm.cdf(abs(Z)))
    return c, p

def do_all_tests(randn):
    D, pval1 = KS_test(randn)
    test1, pval2 = chisquare_test(randn, len(randn)//20)
    test2, pval3 = run_test_1(randn)
    test3, pval4 = run_test_2(randn)
    test4, pval5 = run_test_3(randn)
    c, pval6 = correlation_coefficient(randn)
    # print results
    print('KS test: D =', D, 'p-value =', pval1)
    print('Chi-square test: test =', test1, 'p-value =', pval2)
    print('Run test 1: test =', test2, 'p-value =', pval3)
    print('Run test 2: test =', test3, 'p-value =', pval4)
```

```
    print('Run test 3: test =', test4, 'p-value =', pval5)
    print('Correlation coefficient: c =', c, 'p-value =', pval6)
    # dictionary of results
    results = {'KS test': (D, pval1), 'Chi-square test': (test1, pval2), 'Run test 1': (test2, pval3), '
    return results
```

In [ ]:

```
import numpy as np
import bisect
import math


class Customer:
    def __init__(self, arrival_time, service_time):
        self.service_time = service_time
        self.blocked = False

        self.event = "arrival"
        self.event_time = arrival_time


    def arrive(self, servers, event_list):
        if servers < 1:
            self.blocked = True
            return servers
        else:
            servers -= 1
            servers = max(servers, 0)
            self.event = "departure"
            self.event_time += self.service_time
            bisect.insort(event_list, self, key=lambda x: x.event_time)
            return servers

    def depart(self, servers, m):
        servers += 1
        servers = min(servers, m)
        return servers


def main_loop(arrival_interval, service_time, m, repititions = 10):
    blocked = np.zeros(repititions)
    for i in range(repititions):
        arrival_intervals = arrival_interval()
        service_times = service_time()
        arrival_times = np.cumsum(arrival_intervals)
        event_list = [Customer(arrival_times[i],service_times[i]) for i in range(len(arrival_times))]
        event_list.sort(key=lambda x: x.event_time)
        open_servers = m
        while event_list:
            event = event_list.pop(0)
            if event.event == "arrival":
                open_servers = event.arrive(open_servers, event_list)
                blocked[i] += event.blocked
            elif event.event == "departure":
                open_servers = event.depart(open_servers, m)
    return blocked


def confidence_intervals(samples):
    emp_mean = np.mean(samples)
    emp_std = np.std(samples)
    t = 1.96
    return (emp_mean - t*emp_std/np.sqrt(len(samples)), emp_mean + t*emp_std/np.sqrt(len(samples)))

#Erlang B formula
def erlang_b(m, A):
    return (A**m/math.factorial(m))/np.sum([A**i/math.factorial(i) for i in range(m+1)])
```