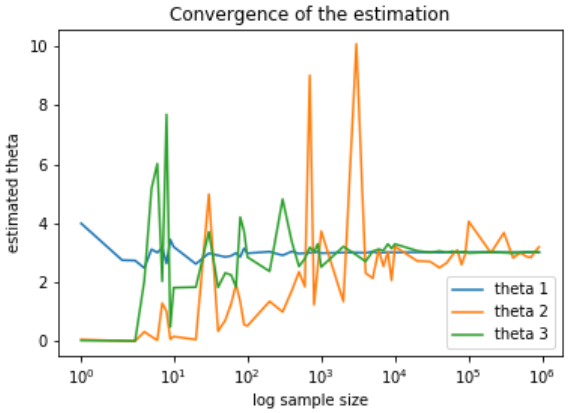
202C - HW1

Problem 1

i)

I think step 3 can be more effective, because a larger proportion of samples are effective (bigger $\pi(x, y)$ and bigger importance weight $\pi(x, y)/g(x, y)$). The distribution of step 3 is similar to that in the middle of Fig. 2.4 in the notebook, while that of step 2 has the same mean and lower variance hence is farther from the truth.

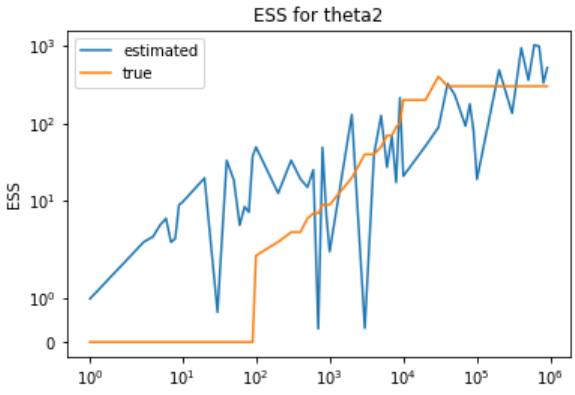
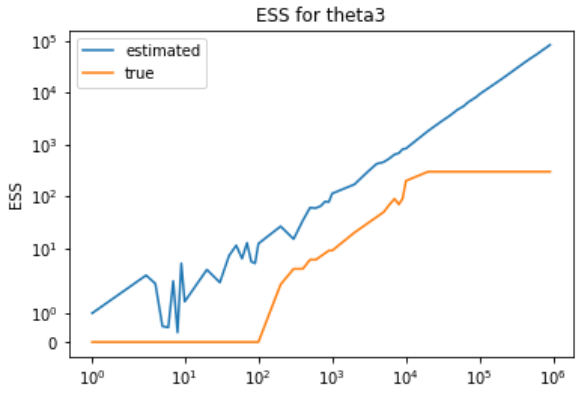


The plot above confirms that guess. It can be observed that: step 1 fluctuates around the ture value within less than 100 samples, and gets more and more stable as the sample size increase, which is justified by law of large number; step 2 does not converge to the same level until tens of thousands of samples, and the fluatuation is severe; step 3 converge in approximately thousands of samples, and the estimation gets stable with millions of samples. It can be concluded that with a sampling distribution closer to the original, the convergnece is faster and much more stable (smaller variance in estimation).

ii)

We use 500 simulations to get a stable estimate of "true" effective sample size. The sampling processures in question i) is repeated to average over estimation errors (absolute distance to the truth) with different sample sizes. A plot of the errors is displayed in the appendix.

We consider error less than .05 as convergence, and find the estimates of "true" ESS, the log plots in question are displayed below:

Step 2	**Step 3**
	

It can be observed that in general the estimator $ess(n)$ increases at a similar rate to that of the "truth". Therefore, it can be used as a general estimation. However, it is always greater than 0 and keeps increase even after convergence, which indicates that it tends to exaggerate the true effective sample size. Moreover, the variance of this estimation is depedent on the sampling process, as θ_2 and $ess(n_2)$ are both very volatile due to poor sampling distribution.

Problem 2

Designs

Design 1: naive

For each move, there are equal possibilities to move to all available neighbors. The walks stops only when there is no available actions.

Design 2: .05 probability to stop

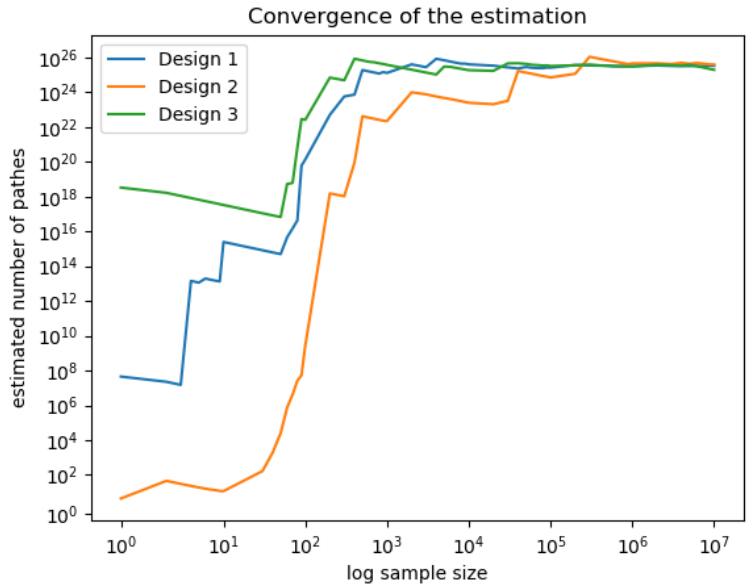
Based on Design 1, now there is also a 0.05 probablity for the walk to stop for each move. The probablities of walking directions are modiofied accordingly to ensure a probabilities sum of 1.

Design 3: higher probablity towards (n, n)

In this design, unequal possibilities to move to the neighbors are used to favor pathes that hits (n,n). The probablities of moving left and down are set to be 0.2, when the current state is father from (n,n) horizontally than vertically, there is 0.35 probability moving right and 0.25 otherwise.

Task i)

The 3 designs and the results with $M = 10^7$ samples are as follow:



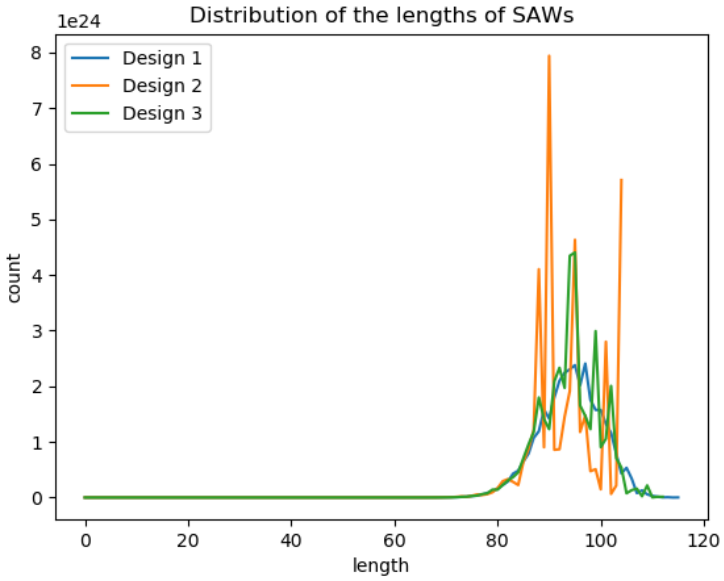
$K_1 = 3.369 \times 10^{25}, K_2 = 3.962 \times 10^{25}, K_3 = 3.760 \times 10^{25}$.

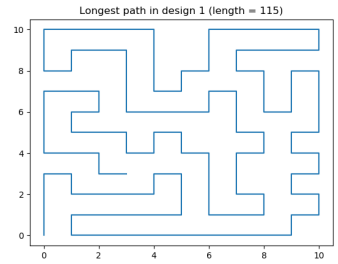
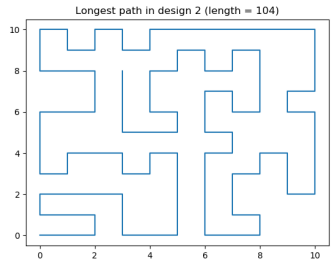
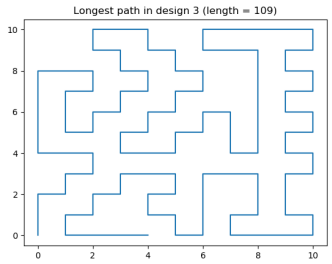
Design 1 is the most naive method, it is like sampling from the original distribution. It yields a good result while the program can be slow due to the hard sampling process. Possible improvements are discussed with the other two designs.

Design 2 favors shorter pathes by introducing a probability of early termination, so as to reduce the number of move-sampling steps required to improve the sampling efficiency. Whereas, the convergence plot shows that it requires a larger sample size to converge to the same level as Design 1.

Design 3 seems to be the most efficient as it converges the fastest, however the difference with Design 1 is very small. Actually, it is very similar to Design 1 with only a small amount of probability change, while move-sampling with different probability requires more computation power. The advantage of this design is more obvious in the nest task, as we will discuss later.

When plotting the length distribution, I think we still need to weight with $w_l = \frac{1}{g(x_l)} / \sum_{i=1}^{M} \frac{1}{g(x_i)} = \frac{1}{g(x_l)M}$. The distributions of path lengths and the longest pathes is displayed below:



Design 1	Design 2	Design 3
		

Using equal probability and no length constraints, Design 1 obtains a distribution resembling gaussian. As for Design 2, the distribution is biased towards left which confirms that this design favors shorter paths. Moreover, as the probability of getting a long path is low, hence corresponding weight is high, as a result, there is a peak at length = 104 where there is only one sample. Design 3 obtains a distribution very similar to that in Design 1, yet with smaller varaince due to the constraint we added.

Task ii): hitting (n,n)

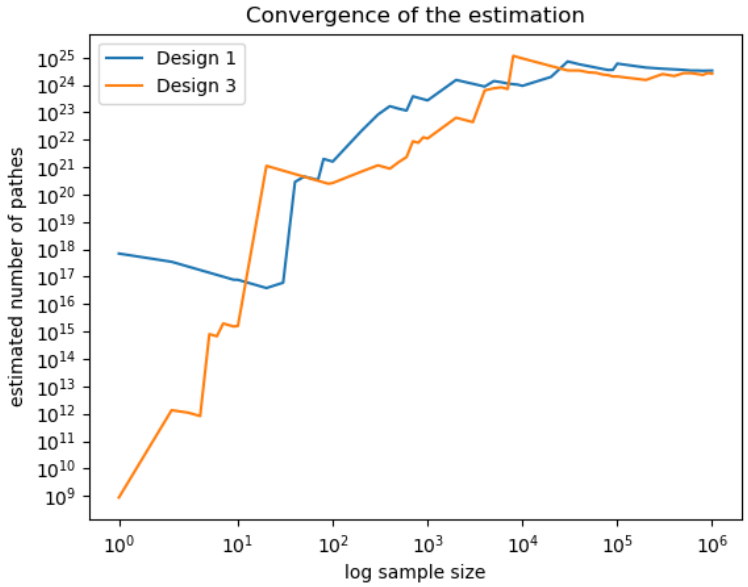
In this task, we focus on pathes that hit target (n,n), in general the sampling converges faster with less than 10^6 samples. However, multiple attempts are required to get a sample in this case.

On average, in Design 1 about 10 attempts are need to get a sample hitting (n,n), so 10^7 attempts are required in total, the program takes a similar amount of time as in task i) (in fact the two task can be accomplished in one program simultaneously).

Design 2 favors shorter pathes, which makes it a lot harder to hit (n, n). On avarage 68 attempts are need for one sample. In the appendix, a plot of estimated K over number of attempts indicates that, Design 2 can not converge within 10^7 attempts, when only 146022 samples are achieved.

In contract, Design 3 explicitly favors pathes that hits (n,n) by giving a higher probability for such move direction. We limit this difference of probablity to a small amount (.1 difference at most) so that the resulting samples are still diverse enough. Whereas, with such a small change, the average number of attempts need to get a sample reduced to 5.

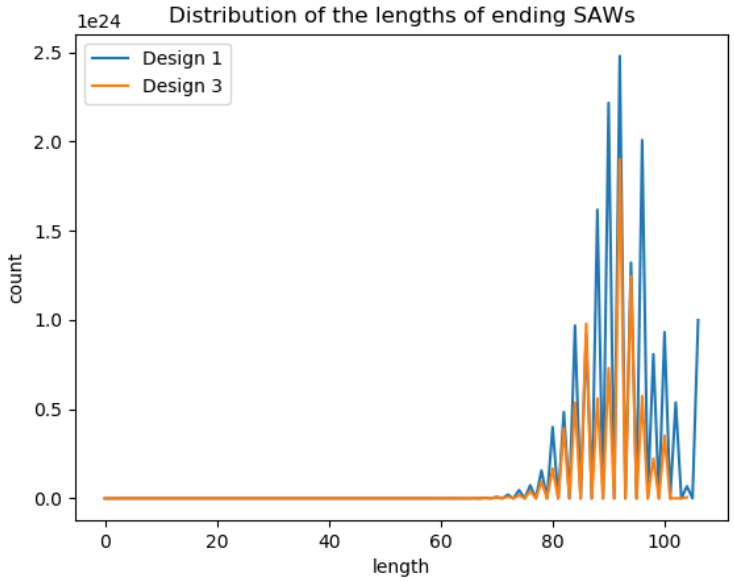
Therefore, we try Design 1 and Design 3 for this task, and the results with $M = 10^6$ samples are as follow:

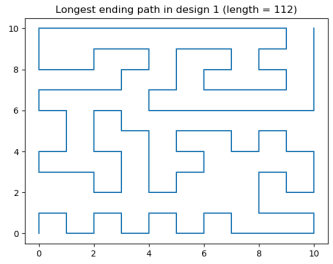
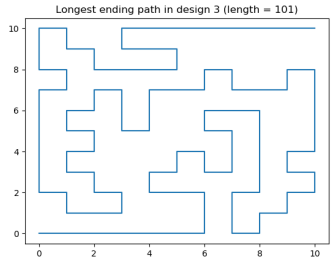


$K_1 = 3.392 \times 10^{24}, K_3 = 2.596 \times 10^{24}$.

The rates of convergence against sample size are similar, whereas, as we discussed before, Design 3 takes around half attempts to get the same level of sample size.

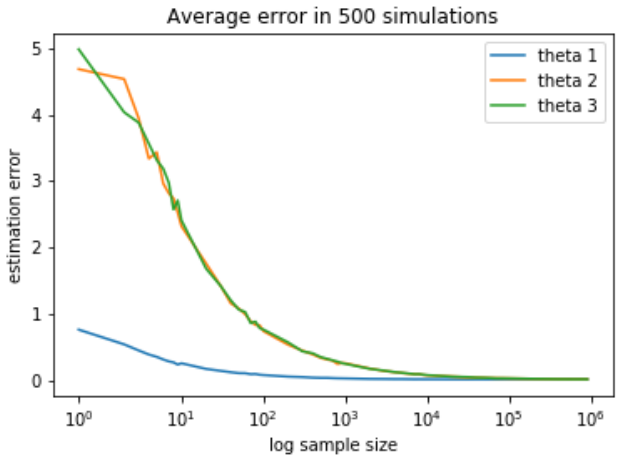
The distributions of path lengths and the longest pathes is displayed below:

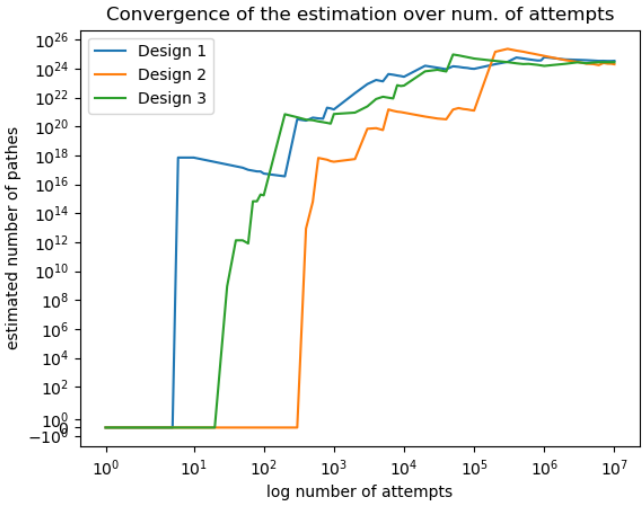


Design 1	Design 3
	

We always get lengths of even number because we only ends at (n,n), apart from that the distributions are similar to that in task i) similar to that of gaussian.

Appendix





Codes

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

1

i)

In [ ]:
```
# define \pi(x) and g(x)
def p(x, y):
    return 0.5/np.pi*np.exp(-0.5*((x-2)**2+(y-2)**2))
def g(x, y, sigma):
    return 0.5/np.pi/sigma**2*np.exp(-0.5/sigma**2*(x**2+y**2))
ns = []
for i in range(6):
    ns += list(range(int(10**(i)),int(10**(i+1)),int(10**(i))))
##ns = np.logspace(1,6,6, dtype=int)
# importance sampling
np.random.seed(7)
cache1 = []
for n in ns:
    x = np.random.normal(2,1,n)
    y = np.random.normal(2,1,n)
    theta1 = np.mean(np.sqrt(x**2+y**2))
    cache1.append(theta1)
cache2 = []
ws2 = []
ess_est2 = []
sigma = 1
for n in ns:
    x = np.random.normal(0,sigma,n)
    y = np.random.normal(0,sigma,n)
    w = p(x,y)/g(x,y,sigma)
    ws2.append(w)
    ess_est2.append(n/(1+np.var(w,ddof=1)))
    theta2 = np.mean(np.sqrt(x**2+y**2)*w)
    cache2.append(theta2)
cache3 = []
ws3 = []
ess_est3 = []
sigma = 4
for n in ns:
    x = np.random.normal(0,sigma,n)
    y = np.random.normal(0,sigma,n)
    w = p(x,y)/g(x,y,sigma)
    ws3.append(w)
    ess_est3.append(n/(1+np.var(w,ddof=1)))
    theta3 = np.mean(np.sqrt(x**2+y**2)*w)
    cache3.append(theta3)
```

In [ ]:
```
# Figure 1-1 for one simulation seed=7
plt.figure(3)
plt.title('Convergence of the estimation')
plt.xscale('symlog')
plt.xlabel('log sample size')
plt.ylabel('estimated theta')
plt.plot(ns, cache1, label='theta 1')
plt.plot(ns, cache2, label='theta 2')
plt.plot(ns, cache3, label='theta 3')
plt.legend(loc='lower right')
plt.savefig("1-1.png")
```

ii)

In [ ]:
```
# For a more stable error measure: 500 simulation
def avg_err():
    cache1 = []
    cache2 = []
    cache3 = []
    for n in ns:
        x = np.random.normal(2,1,n)
        y = np.random.normal(2,1,n)
        theta1 = np.mean(np.sqrt(x**2+y**2))
        cache1.append(theta1)
        x = np.random.normal(0,sigma,n)
        y = np.random.normal(0,sigma,n)
        w = p(x,y)/g(x,y,sigma)
        theta2 = np.mean(np.sqrt(x**2+y**2)*w)
        cache2.append(theta2)
        x = np.random.normal(0,4,n)
        y = np.random.normal(0,4,n)
        w = p(x,y)/g(x,y,sigma)
        theta3 = np.mean(np.sqrt(x**2+y**2)*w)
        cache3.append(theta3)
    err1.append(np.abs([x-3 for x in cache1]))
    err2.append(np.abs([x-3 for x in cache2]))
    err3.append(np.abs([x-3 for x in cache3]))

err1 = []
err2 = []
err3 = []
for i in range(500):
    avg_err()
avg_err1 = np.mean(err1, axis=0)
avg_err2 = np.mean(err2, axis=0)
avg_err3 = np.mean(err3, axis=0)
print("Done")
```

In [ ]:
```
# plot the average error
plt.title('Average error in 500 simulations')
plt.xscale('symlog')
plt.xlabel('log sample size')
plt.ylabel('estimation error')
plt.plot(ns, avg_err1, label='theta 1')
plt.plot(ns, avg_err2, label='theta 2')
plt.plot(ns, avg_err3, label='theta 3')
plt.legend(loc='upper right')
plt.savefig("Apdx1.png")
```

In [ ]:
```
# estimate "true" ess
ess_true2 = []
ess_true3 = []
cvg1 = np.argwhere(avg_err1 < 5e-2)[0] #20
cvg2 = np.argwhere(avg_err2 < 5e-2)[0] #38
cvg3 = np.argwhere(avg_err3 < 5e-2)[0] #38
maxi = np.max(avg_err1)
for i in range(len(ns)):
    if i > cvg2:
        ess_true2.append(ns[cvg1[0]])
    else:
        if avg_err2[i] > maxi:
            ess_true2.append(0)
        else:
            reach = np.argwhere(avg_err1 <= avg_err2[i])
            if len(reach) > 1:
                reach = reach[0]
            ess_true2.append(ns[reach[0]])
    if i > cvg3:
        ess_true3.append(ns[cvg1[0]])
    else:
        if avg_err3[i] > maxi:
            ess_true3.append(0)
        else:
            reach = np.argwhere(avg_err1 <= avg_err3[i])
            if len(reach) > 1:
                reach = reach[0]
            ess_true3.append(ns[reach[0]])
```

In [ ]:
```
plt.figure(2)
plt.title('ESS for theta2')
plt.xlabel('')
plt.ylabel('ESS')
plt.yscale('symlog')
plt.xscale('symlog')
plt.plot(ns, ess_est2, label='estimated')
plt.plot(ns, ess_true2, label='true')
plt.legend(loc='upper left')
#plt.savefig("1-2-a.png")
```

In [ ]:
```
plt.figure(2)
plt.title('ESS for theta3')
plt.xlabel('')
plt.ylabel('ESS')
plt.yscale('symlog')
plt.xscale('symlog')
plt.plot(ns, ess_est3, label='estimated')
plt.plot(ns, ess_true3, label='true')
plt.legend(loc='upper left')
#plt.savefig("1-2-b.png")
```

ii)

In [ ]:
```
## Parallel computation
from queue import Queue
from threading import Thread
class Worker(Thread):
    """ Thread executing tasks from a given tasks queue """
    def __init__(self, tasks):
        Thread.__init__(self)
        self.tasks = tasks
        self.daemon = True
        self.start()
    def run(self):
        while True:
            func, args, kargs = self.tasks.get()
            try:
                func(*args, **kargs)
            except Exception as e:
                # An exception happened in this thread
                print(e)
            finally:
                # Mark this task as done, whether an exception happened or not
                self.tasks.task_done()
class ThreadPool:
    """ Pool of threads consuming tasks from a queue """
    def __init__(self, num_threads):
        self.tasks = Queue(num_threads)
        for _ in range(num_threads):
            Worker(self.tasks)
    def add_task(self, func, *args, **kargs):
        """ Add a task to the queue """
        self.tasks.put((func, args, kargs))
    def map(self, func, args_list):
        """ Add a list of tasks to the queue """
        for args in args_list:
            self.add_task(func, args)
    def wait_completion(self):
        """ Wait for completion of all the tasks in the queue """
        self.tasks.join()\

pool = ThreadPool(50)
```

2

Design 1

In [ ]:
```
def saw1(i):
    state = np.array([0,0])
    path = []
    p = 1
    end = False
    r, l, u, d = np.array([1,0]), np.array([-1,0]), np.array([0,1]), np.array([0,-1])
    while list(state) not in path:
        last = np.copy(state)
        path.append(list(last))
        # candidates
        moves = []
        if all(state+r >= 0) and all(state+r <= 10) and list(state+r) not in path:
            moves.append(r)
        if all(state+l >= 0) and all(state+l <= 10) and list(state+l) not in path:
            moves.append(l)
        if all(state+u >= 0) and all(state+u <= 10) and list(state+u) not in path:
            moves.append(u)
        if all(state+d >= 0) and all(state+d <= 10) and list(state+d) not in path:
            moves.append(d)
        # choose move
        num = len(moves)
        if num == 0:
            break
        else:
            idx = np.random.randint(num, size=1)
            prob = 1/num
            move = moves[idx[0]]
            # take move, cumulate p(r)
            state += move
            p = p*prob #g(x)
    # whether end
        if all(state == 10):
            end = np.copy(p)
            pathe = np.copy(path)
            le = len(pathe)
    if length > len(path1):
        path1 = np.copy(path)
    if le > len(pathe1):
        pathe1 = pathe
#    s = len(design1) # attempts
#    se = sum([x[0]!=0 for x in design1])
#    if s in ns or (le!=0 and se in ns):
#        # i)
#        K_all = sum([1/x[1] for x in design1])/s
#        # ii)
#        w = np.array([1/x[0] for x in design1 if x[0]!=0]) # w_0
#        ends = np.array([x[0] for x in design1])
#        att = np.argwhere(ends!=0) # cumulative attempts
#        u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
#        weights = w/u #weights: 1/g(x)
#        K_end = sum(weights)/se
#        cache1.append((s, se, K_all, K_end))
#        print("Sample[%.1e] Sample_end[%.1e] Design3: %.3e, %.3e" % (s, se, K_all, K_end))
```

In [ ]:
```
# record
design1 = [] # probablity of hitting (n,n) - 0 if not hitting, probability, length, length of ending p
ath
path1 = [] # longest path
pathe1 = [] # longest path hitting (n,n)
cache1 = []
# run
M = int(1e7)
pool.map(saw1, range(M))
pool.wait_completion()
```

In [ ]:
```
# i)
K_all = np.mean([1/x[1] for x in design1[:int(1e7)]]) #3.3689065344252283e+25
# ii)
w = np.array([1/x[0] for x in design1 if x[0]!=0]) # w_0
ends = np.array([x[0] for x in design1])
att = np.argwhere(ends!=0) # cumulative attempts
u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
weights = w/u #weights: 1/g(x)
K_end = sum(weights)/se #
#pe = [x[2] for x in design1 if x[0]]
#l = [len(x[2]) for x in design1]
#le = [len(x[2]) for x in design1 if x[0]]
#path1 = p[np.argmax(l)]
#pathe1 = pe[np.argmax(le)]
#e, g, p = zip(*design1)
#ll = tuple(l)
#design1 = list(zip(*[e,g,ll]))
```

Design 2

In [ ]:
```
def saw2(eps):
    state = np.array([0,0])
    path = []
    p = 1
    end = 0
    le = 0
    r, l, u, d = np.array([1,0]), np.array([-1,0]), np.array([0,1]), np.array([0,-1])
    while list(state) not in path:
        last = np.copy(state)
        path.append(list(last))
        # candidates
        moves = [np.array([0,0])]
        if all(state+r >= 0) and all(state+r <= 10) and list(state+r) not in path:
            moves.append(r)
        if all(state+l >= 0) and all(state+l <= 10) and list(state+l) not in path:
            moves.append(l)
        if all(state+u >= 0) and all(state+u <= 10) and list(state+u) not in path:
            moves.append(u)
        if all(state+d >= 0) and all(state+d <= 10) and list(state+d) not in path:
            moves.append(d)
        # choose move
        num = len(moves)
        if num == 1:
            break
        else:
            probs = [eps] + list(np.repeat((1-eps)/(num-1),num-1))
            idx = np.random.choice(num, 1, p = probs)
            choice = idx[0]
            if choice == 0:
                break
            move = moves[choice]
            prob = probs[choice]
            # take move, cumulate p(r)
            state += move
            p = p*prob #g(x)
    # whether end
#         if all(state == 10):
#             end = np.copy(p)
#             pathe = np.copy(path)
#             le = len(pathe)
    length = len(path)
    design2.append((end,p,length,le))
    if length > len(path2):
        path2 = np.copy(path)
#     if le > len(pathe2):
#         pathe2 = pathe
#     s = len(design2) # attempts
#     se = sum([x[0]!=0 for x in design2])
#     if s in ns or (le!=0 and se in ns):
#         # i)
#         K_all = sum([1/x[1] for x in design2])/s
#         # ii)
#         w = np.array([1/x[0] for x in design2 if x[0]!=0]) # w_0
#         ends = np.array([x[0] for x in design2])
#         att = np.argwhere(ends!=0) # cumulative attempts
#         u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
#         weights = w/u #weights: 1/g(x)
#         K_end = sum(weights)/se
#         cache2.append((s, se, K_all, K_end))
#         print("Sample[%.1e] Sample_end[%.1e] Design3: %.3e, %.3e" % (s, se, K_all, K_end))
```

In [ ]:
```
# record
design2 = [] # probablity of hitting (n,n) - 0 if not hitting, probability, length, length of ending p
ath
path2 = [] # longest path
#pathe2 = [] # longest path hitting (n,n)
cache2 = []
# run
M = int(1e7)
pool.map(saw2, range(M))
pool.wait_completion()
```

In [ ]:
```
# i)
K_all = np.mean([1/x[1] for x in design2[:int(1e7)]]) #3.9621050204425594e+25
# ii)
#w = np.array([1/x[0] for x in design if x[0]!=0]) # w_0
#ends = np.array([x[0] for x in design2])
#att = np.argwhere(ends!=0) # cumulative attempts
#u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
#weights = w/u #weights: 1/g(x)
#K_end = sum(weights)/se #
```

Design 3

In [ ]:
```
def saw3(eps):
    state = np.array([0,0])
    path = []
    p = 1
    end = 0
    le = 0
    r, l, u, d = np.array([1,0]), np.array([-1,0]), np.array([0,1]), np.array([0,-1])
    while list(state) not in path:
        last = np.copy(state)
        path.append(list(last))
        # candidates
        moves = [r, l, u, d]
        # initialize probs with guidance
        probs = np.array([0.3,0.2,0.3,0.2])
        distance = np.array([10,10]) - state
        monitor = distance[0]-distance[1]
        if monitor > 0:
            probs[0], probs[2] = 0.35, 0.25
        if monitor <0:
            probs[0], probs[2] = 0.25, 0.35
        if any(state+r < 0) or any(state+r > 10) or list(state+r) in path:
            probs[0] = 0
        if any(state+l < 0) or any(state+l > 10) or list(state+l) in path:
            probs[1] = 0
        if any(state+u < 0) or any(state+u > 10) or list(state+u) in path:
            probs[2] = 0
        if any(state+d < 0) or any(state+d > 10) or list(state+d) in path:
            probs[3] = 0
        plus = 1 - sum(probs)
        if plus == 1:
            break
        # choose move
        else:
            probs[probs!=0] += plus/sum(probs!=0)
            idx = np.random.choice(4, 1, p = probs)
            choice = idx[0]
            move = moves[choice]
            prob = probs[choice]
            # take move, cumulate p(r)
            state += move
            p = p*prob #g(x)
    # whether end
        if all(state == 10):
            end = np.copy(p)
            pathe = np.copy(path)
            le = len(pathe)
    length = len(path)
    design3.append((end, p, length,le))
    global path3
    global pathe3
    if length > len(path3):
        path3 = np.copy(path)
    if le > len(pathe3):
        pathe3 = pathe
#    s = len(design3) # attempts
#    se = sum([x[0]!=0 for x in design3])
#    if s in ns or (le!=0 and se in ns):
#        # i)
#        K_all = sum([1/x[1] for x in design3])/s
#        # ii)
#        w = np.array([1/x[0] for x in design3 if x[0]!=0]) # w_0
#        ends = np.array([x[0] for x in design3])
#        att = np.argwhere(ends!=0) # cumulative attempts
#        u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
#        weights = w/u #weights: 1/g(x)
#        K_end = sum(weights)/se
#        cache3.append((s, se, K_all, K_end))
#        print("Sample[%.1e] Sample_end[%.1e] Design3: %.3e, %.3e" % (s, se, K_all, K_end))
```

In [ ]:
```
# record
design3 = [] # probablity of hitting (n,n) - 0 if not hitting, probability, length, length of ending p
ath
path3 = [] # longest path
pathe3 = [] # longest path hitting (n,n)
cache3 = []
# run
M = int(1e7)
pool.map(saw3, range(M))
pool.wait_completion()
```

In [ ]:
```
# i)
K_all = np.mean([1/x[1] for x in design3[:int(1e7)]]) #3.760032560502165e+25
# ii)
w = np.array([1/x[0] for x in design3 if x[0]!=0]) # w_0
ends = np.array([x[0] for x in design3])
att = np.argwhere(ends!=0) # cumulative attempts
u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
weights = w/u #weights: 1/g(x)
K_end = sum(weights)/se #
```

Plots

In [ ]:
```
# Convergence plot K against M for task i)
ns = []
for i in range(7):
    ns += list(range(int(10**(i)),int(10**(i+1)),int(10**(i))))
K_all1 = [np.mean([1/x[1] for x in design1[:ns[0]]])]
K_all2 = [np.mean([1/x[1] for x in design2[:ns[0]]])]
K_all3 = [np.mean([1/x[1] for x in design3[:ns[0]]])]
for i in range(len(ns)-1):
    n = ns[i+1]
    nl = ns[i]
    K_all1.append((K_all1[i]*nl+np.sum([1/x[1] for x in design1[nl:n]]))/n)
    K_all2.append((K_all2[i]*nl+np.sum([1/x[1] for x in design2[nl:n]]))/n)
    K_all3.append((K_all3[i]*nl+np.sum([1/x[1] for x in design3[nl:n]]))/n)
plt.figure(3)
plt.title('Convergence of the estimation')
plt.xscale('symlog')
plt.yscale('symlog')
plt.xlabel('log sample size')
plt.ylabel('estimated number of pathes')
plt.plot(ns, K_all1, label='Design 1')
plt.plot(ns, K_all2, label='Design 2')
plt.plot(ns, K_all3, label='Design 3')
plt.legend(loc='upper left')
plt.savefig("2-1-0.png")
```

In [ ]:
```
# Convergence plot K against M for task ii)
ns = []
for i in range(6):
    ns += list(range(int(10**(i)),int(10**(i+1)),int(10**(i))))
w = np.array([1/x[0] for x in design1 if x[0]!=0]) # w_0
ends = np.array([x[0] for x in design1])
att = np.argwhere(ends!=0) # cumulative attempts
u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
weights1 = w/u #weights: 1/g(x)
#w = np.array([1/x[0] for x in design2 if x[0]!=0]) # w_0
#ends = np.array([x[0] for x in design2])
#att = np.argwhere(ends!=0) # cumulative attempts
#u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
#weights2 = w/u #weights: 1/g(x)
w = np.array([1/x[0] for x in design3 if x[0]!=0]) # w_0
ends = np.array([x[0] for x in design3])
att = np.argwhere(ends!=0) # cumulative attempts
u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
weights3 = w/u #weights: 1/g(x)
K_end1 = sum(weights1[:ns[0]])/ns[0]
#K_end2 = sum(weights[:ns[0]])/ns[0]
K_end3 = sum(weights3[:ns[0]])/ns[0]
for i in range(len(ns)-1):
    n = ns[i+1]
    nl = ns[i]
    K_end1.append((K_end1[i]*nl+sum(weights1[nl:n]))/n)
#    K_end2.append((K_end2[i]*nl+sum(weights2[nl:n]))/n)
    K_end3.append((K_end3[i]*nl+sum(weights3[nl:n]))/n)
plt.figure(3)
plt.title('Convergence of the estimation')
plt.xscale('symlog')
plt.yscale('symlog')
plt.xlabel('log sample size')
plt.ylabel('estimated number of pathes')
plt.plot(ns, K_end1, label='Design 1')
#plt.plot(ns, K_all2, label='Design 2')
plt.plot(ns, K_end3, label='Design 3')
plt.legend(loc='upper left')
plt.savefig("2-2-0.png")
# K against num. of attempts
K_end1 = []
K_end2 = []
K_end3 = []
d = design[int(3052380):]
for i in ns:
    w = np.array([1/x[0] for x in d[:i] if x[0]!=0]) # w_0
    if len(w) == 0:
        K_end1.append(0)
    else:
        ends = np.array([x[0] for x in d[:i]])
        att = np.argwhere(ends!=0) # cumulative attempts
        u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
        K_end1.append(np.mean(w/u))
    w = np.array([1/x[0] for x in design2[:i] if x[0]!=0]) # w_0
    if len(w) == 0:
        K_end2.append(0)
    else:
        ends = np.array([x[0] for x in design2[:i]])
        att = np.argwhere(ends!=0) # cumulative attempts
        u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
        K_end2.append(np.mean(w/u))
    w = np.array([1/x[0] for x in design3[:i] if x[0]!=0]) # w_0
    if len(w) == 0:
        K_end3.append(0)
    else:
        ends = np.array([x[0] for x in design3[:i]])
        att = np.argwhere(ends!=0) # cumulative attempts
        u = np.append(att[0]+1,np.diff(att,axis=0)) # attempts
        K_end3.append(np.mean(w/u))
plt.figure(3)
plt.title('Convergence of the estimation over num. of attempts')
plt.xscale('symlog')
plt.yscale('symlog')
plt.xlabel('log number of attempts')
plt.ylabel('estimated number of pathes')
plt.plot(ns, K_end1, label='Design 1')
```

```
plt.plot(ns, K_end2, label='Design 2')
plt.plot(ns, K_end3, label='Design 3')
plt.legend(loc='upper left')
plt.savefig("Apdx2.png")
```

In [ ]:
```
# lengths ditribution plots
l1 = [x[2] for x in design1]
x1 = list(range(max(l1)+1))
y1 = np.bincount(l1, weights=[1/x[1] for x in design1])/len(l1)
l2 = [x[2] for x in design2]
x2 = list(range(max(l2)+1))
y2 = np.bincount(l2, weights=[1/x[1] for x in design2])/len(l2)
l3 = [x[2] for x in design3]
x3 = list(range(max(l3)+1))
y3 = np.bincount(l3, weights=[1/x[1] for x in design3])/len(l3)
plt.figure(3)
plt.title('Distribution of the lengths of SAWs')
plt.xlabel('length')
plt.ylabel('count')
plt.plot(x1, y1, label='Design 1')
plt.plot(x2, y2, label='Design 2')
plt.plot(x3, y3, label='Design 3')
plt.legend(loc='lower right')
plt.savefig("2-1-l.png")
# task ii)
l1 = [x[3] for x in design1 if x[0]!=0]
x1 = list(range(max(l1)+1))
y1 = np.bincount(l1, weights=[1/x[0] for x in design1 if x[0]!=0])/len(l1)
#l2 = [x[2] for x in design2]
#x2 = list(range(max(l2)+1))
#y2 = np.bincount(l2, weights=[1/x[1] for x in design2 if x[0]])/len(l2)
l3 = [x[3] for x in design3 if x[0]]
x3 = list(range(max(l3)+1))
y3 = np.bincount(l3, weights=[1/x[0] for x in design3 if x[0]!=0])/len(l3)
plt.figure(3)
plt.title('Distribution of the lengths of ending SAWs')
plt.xlabel('length')
plt.ylabel('count')
plt.plot(x1, y1, label='Design 1')
#plt.plot(x2, y2, label='Design 2')
plt.plot(x3, y3, label='Design 3')
plt.legend(loc='lower right')
plt.savefig("2-1-b.png")
```