CS168 Spring Assignment 7

SUNet ID(s): 05794739
Name(s): Luis A. Perez
Collaborators:

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

Part 1

(a) (a) The circle graph with $n = 10$ is periodic and irreducible.

This is because it is bipartite. Consider partitions A, B where even nodes are in A and odd nodes in B. Then the edges (u, v) exists only if $u \in A, v \in B$. As such, it has a period of 2.

It is irreducible because one can reach any state from any other state.

(b) The circle graph with $n = 9$ is aperiodic and irreducible.

Consider any pair of distinct nodes i, j. WLOG, suppose $i < j$. Then we can reach j from i in $j - i$ steps as well as $9 - (j - i)$. Note that $j - i \in \{1, 2, 3, 4, 5, 6, 7, 8\}$ and, correspondingly, we'd have $9 - (j - i) \in \{8, 7, 6, 5, 4, 3, 2, 1\}$. The only pairs for which the GCD is not 1 are $(3, 6), (6, 3)$. However, if we can reach a node in 3 steps, we can reach it in 5 (just go back once and foward again). As such, even for these nodes, the GCD for times at which they are reachable is 1.

It is irreducible because one can reach any state from any other state.

The stationary distribution π is the uniform distribution over all states. To see why, note that all states are symmetric and interchangeable, so the stationary distribution must be uniform. There are 9 states, so we have:

$$\pi_i = \frac{1}{9} \text{ for all } 1 \leq i \leq 9$$

(c) The circle graph with $n = 9$ and an extra edges connecting nodes 1 and 5 is aperiodic and irreducible.

Ignoring the extra edge, we already know it's aperiodic as per (2). As such, we only need to verify that the states 1 and 5 are still aperiodic. They are, since we can reach them in either 4 steps or 5 steps.

It is irreducible because one can reach any state from any other state.

The stationary distribution π is no longer uniform. However, we can compute it easily. We have 7 nodes with two edges, and two nodes with 3. As such, we have $2 * 7 + 3 * 2 = 20$ distributions. As such, we have:

$$\pi_i = \frac{1}{10} \quad i \neq 1, 5 \pi_i = \frac{3}{20} \quad i = 1, 5$$

(b) We plot the variational distance from the distribution at time t to the stationary distribution (we use the inform distribution for graphs which are either periodic or reducible). For the results, see Figure 1.

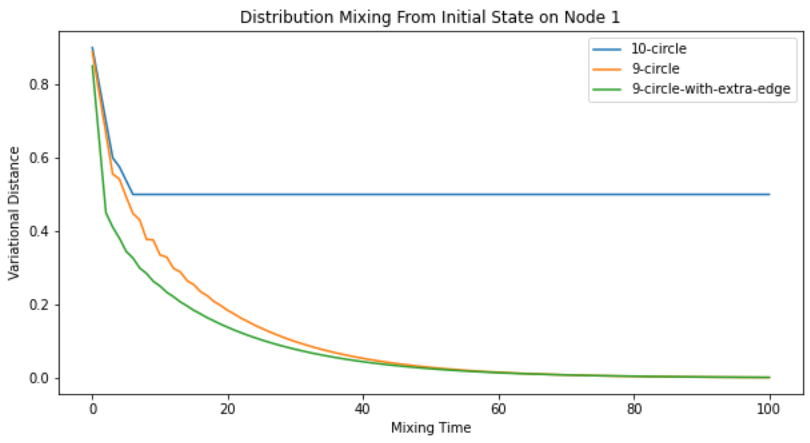


Figure 1: Plot of variational distance between true distribution at time t and stationary distribution for each of the discussed graphs. If no stationary distribution exists (eg, for the 10 node circle graph), we use the uniform distribution.

The code used to generate the above is here:

```
def circle_graph(n=10):
    """Returns transition matrix for a circle graph."""
    A = np.zeros((n, n))
    for i in range(n):
        A[i, (i + 1) % n] = 1
        A[(i + 1) % n, i] = 1
    return 0.5 * A, 1 / n * np.ones((1, n))

def circle_graph_with_extra_edge(n=10, a=1, b=5):
    """Same as above but for a circle graph with an additional edge (a,b)."""
    P, _ = circle_graph(n=n)
    P[a-1, b-1] = 0.5
    P[b-1, a-1] = 0.5
    P[a-1, :] = P[a-1, :] * 2 / 3
    P[b-1, :] = P[b-1, :] * 2 / 3
    denom = 2*(n-2) + 3*2
    pi = 2 * np.ones((1, n)) / denom
    pi[0, a - 1] = 3 / denom
    pi[0, b - 1] = 3 / denom
    return P, pi

def variation_distance(x, y):
    return 1 / 2 * np.sum(np.abs(x - y))

def chain_mixing(T, pi):
    s = np.zeros((1, np.shape(T)[0]))
    s[0, 0] = 1
    distance = [variation_distance(s, pi)]
    for t in range(1, 101):
        s = np.dot(s, T)
        distance.append(variation_distance(s, pi))

    return distance

def problem1b():
    graphs = {
        '10-circle' : circle_graph(n=10),
        '9-circle': circle_graph(n=9),
        '9-circle-with-extra-edge': circle_graph_with_extra_edge(n=9)
    }
    for name, (T, pi) in graphs.items():
        dists = chain_mixing(T, pi)
        plt.plot(range(len(dists)), dists, label=name)
    plt.legend()
    plt.title('Distribution Mixing From Initial State on Node 1')
    plt.ylabel('Variational Distance')
    plt.xlabel('Mixing Time')
    plt.savefig('figures/mixing_time.png', format='png')
    plt.close()

problem1b()
```

(c) 10-circle has $\lambda_2 = 0.8090169943749482$.

9-circle has $\lambda_2 = 0.766044443118978$.

9-circle-with-extra-edge has $\lambda_2 = 0.7675918792439985$.

(d) While normally we'd expect a smaller λ_2 to indicates a faster convergence rate, for our specific initial state we see from the above that the 9-graph with an extra edge actually has the faster convergence (despite a slightly larger second largest eigenvalue). The reason for this relates to the power iteration algorithm. We know that the ratio of the second largest eigenvector to the largest eigenvector determines the converged. In the case of a transtion matrix, this ratio entirely depends on the second largest eigenvector (the largest eigenvector is always 1). So as this value is smaller, we'd expect fewer iteration.

However, this is a lose upperbound, and as we can see from our results above, it does not always hold.

Part 2

(a) Each state corresponds to a unique permutation of the parks we visit. As such, there are a total of 30! states, which is an extremely large number of possible states.

For the casee where $T = 0$, we will not see all possible routes no matter how large MAXITER is. The reason for this is that we will be doing only a local search. Note that for a given route, there are only n possible transitions out of that route. If our current route is a local minimum amongst these, we'll never escape.

On the other hand, when $T > 0$, we will eventually see all possible routes. The reason is that there is a non-zero probability of taking, essentially, a random move. As such, there's a non-zero probability that we reach any state.

The code implementing the algorithm is presented below:

```
def mcmc(D, n_iters, T, route_fn):
    """Implements MCMC for Eucledian Traveling Salesmen on Distance Matrix D."""
    route = np.array(range(D.shape[0]))
    np.random.shuffle(route)
    best = route.copy()

    selected = np.random.choice(range(D.shape[0]), replace=True, size=n_iters)
    selected2 = np.random.choice(range(D.shape[0]), replace=True, size=n_iters)
    route_dists = []
    for idx, idx2 in zip(selected, selected2):
        newroute = route.copy()
        route_fn(route, newroute, idx, idx2)
        route_dist = route_length(D, route.tolist() + [route[0]])
        newroute_dist = route_length(D, newroute.tolist() + [newroute[0]])
        route_dists.append(route_dist)
        delta_dist = newroute_dist - route_dist
        if delta_dist < 0 or (T > 0 and np.random.uniform() < np.exp(-delta_dist / 
            route = newroute
        if route_length(D, route.tolist() + [route[0]]) < route_length(D, best.toli
            best = route
    return best, route_dists
```

(b) We try four values of $T \in \{0, 1, 10, 100\}$ in Figure 2. From the figure, we see that the value with $T = 10$ appears to perform the best. This also holds since over the ten trials, $T = 10$ finds a tour of length 279.21.

(c) We repeat the experiment with the new mechanism for choosing paths in Figure 3. It turns out that in this case, the best temperature parameter appears to be $T = 1$ which finds a tour length of 166.23.

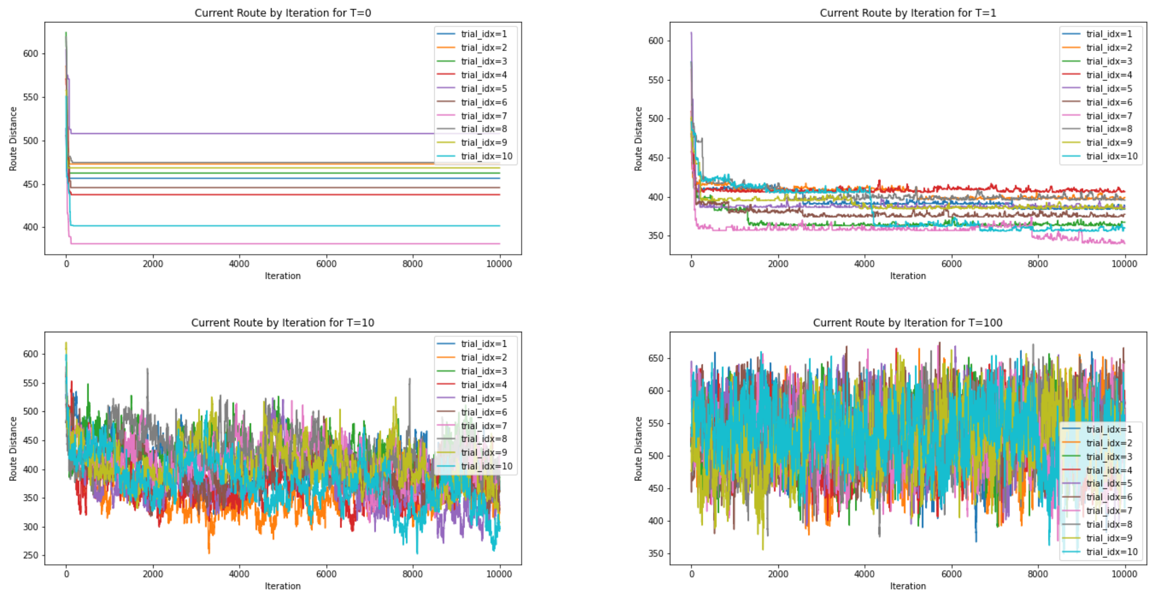


Figure 2: Searched Routes and Their Corresponding Tour Distance by Iteration for Different Temperature Values using Markov Chain Monte Carlo.

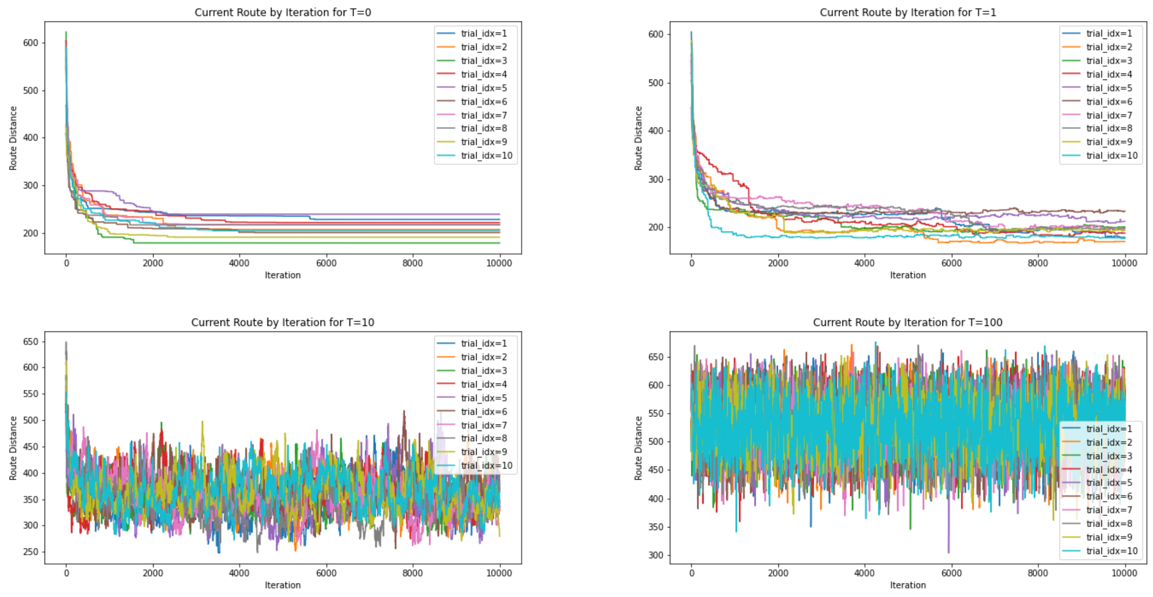


Figure 3: Searched Routes and Their Corresponding Tour Distance by Iteration for Different Temperature Values using Markov Chain Monte Carlo.

(d) It appears that generally speaking, the second approach of just selecting two random parks to swap tends to lead to discovering shorter tours. The reason for this is that the Markov Chain in (c) has a more distributed transition matrix, where each state has connections to $O(n^2)$ other states instead of just $O(n)$ as in (b). As such, the starting state is less important in this case, which further allows the system to mix faster.

This also explain why (c) requires a lower temperature. The easier mixing means that we need less randomess in order to avoid getting stuck in local minima. As such, we approach the stationary distribution much more quickly and easily. Too higher a temperature leads to suboptimal decisions.

Part 3

(a) Does not need to be submitted.

(b) For the full code, please see the Appendix section. The input plan is given as per the below:

```
array([ 0.36,  0.49,  0.88,  0.85,  1.  ,  1.  ,  0.74,  1.  , -1.  ,
       -0.83,  0.29,  0.16,  0.8 ,  0.73,  0.57, -0.61,  0.77,  0.95,
       -0.96, -0.08, -0.3 , -1.  , -0.82, -1.  ,  0.95,  0.48, -1.  ,
        0.72, -0.78, -1.  , -0.11, -0.9 ,  0.78, -0.26,  0.57,  0.15,
        0.09, -0.88, -0.86, -0.5 ])
```

A video of the above can be found here: https://youtu.be/kQFOffXiVMM

(c) We achieve a distance of 9.198867470630958. Our method is a simple modification of the MCMC method used in previous parts of the assignmet. We use $T = 0$ and define neighboring states as adding a uniform random value in the range $[-0.1, 0.1]$ (so changing the vector by at most 10%). Our local search implies that the nearby values are also good, as long as the inputs aren't changed too much.

HW7

May 23, 2020

1 CS 168 Spring Assignment 7

SUNet ID(s): 05794739

Name(s): Luis A. Perez

Collaborators: None

By turning in this assignment, I agree by the Stanford honor code and declare that all of this is my own work.

2 Imports

[114]:
```
import collections
import matplotlib.pyplot as plt
import scipy

import numpy as np
from PIL import Image
from sklearn import decomposition
import pandas as pd
import seaborn as sns
import os
import warnings

import IPython

from typing import Dict, List, Text, Tuple

# Make figure larger
plt.rcParams['figure.figsize'] = [10, 5]

# Set numpy seed for consistent results.
np.random.seed(1)
```

[2]:
```
class Globals:
    DATA_PATH = 'data/'
```

3 Part 1

3.1 Part 1b

[105]:
```
def circle_graph(n=10):
    """Returns transition matrix for a circle graph."""
    A = np.zeros((n, n))
    for i in range(n):
        A[i, (i + 1) % n] = 1
        A[(i + 1) % n, i] = 1
    return 0.5 * A, 1 / n * np.ones((1, n))

def circle_graph_with_extra_edge(n=10, a=1, b=5):
    """Same as above but for a circle graph with an additional edge (a,b)."""
    P, _ = circle_graph(n=n)
    P[a-1, b-1] = 0.5
    P[b-1, a-1] = 0.5
    P[a-1, :] = P[a-1, :] * 2 / 3
    P[b-1, :] = P[b-1, :] * 2 / 3
    denom = 2*(n-2) + 3*2
    pi = 2 * np.ones((1, n)) / denom
    pi[0, a - 1] = 3 / denom
    pi[0, b - 1] = 3 / denom

    return P, pi
```

[ ]:
```
def variation_distance(x, y):
    return 1 / 2 * np.sum(np.abs(x - y))

def chain_mixing(T, pi):
    s = np.zeros((1, np.shape(T)[0]))
    s[0, 0] = 1
    distance = [variation_distance(s, pi)]
    for t in range(1, 101):
        s = np.dot(s, T)
        distance.append(variation_distance(s, pi))

    return distance
```

[110]:
```
def problem1b():
    graphs = {
        '10-circle' : circle_graph(n=10),
        '9-circle': circle_graph(n=9),
        '9-circle-with-extra-edge': circle_graph_with_extra_edge(n=9)
    }
    for name, (T, pi) in graphs.items():
        dists = chain_mixing(T, pi)
        plt.plot(range(len(dists)), dists, label=name)
    plt.legend()
    plt.title('Distribution Mixing From Initial State on Node 1')
    plt.ylabel('Variational Distance')
    plt.xlabel('Mixing Time')
    plt.savefig('figures/mixing_time.png', format='png')
    plt.close()
```

[111]:
```
problem1b()
```

3.2 Part 1c

[136]:
```
def problem1c():
    graphs = {
        '10-circle' : circle_graph(n=10),
        '9-circle': circle_graph(n=9),
        '9-circle-with-extra-edge': circle_graph_with_extra_edge(n=9)
    }
    for name, (T, _) in graphs.items():
        w, _ = np.linalg.eig(T.T)
        second_max_idx = len(w) - 2
        lambda2 = np.partition(w, second_max_idx)[second_max_idx]
        print(f'{name}')
        IPython.display.display(IPython.display.Math(f'\lambda_2={lambda2}'))
```

[137]:
```
problem1c()
```

```
10-circle
```

$\lambda_2 = 0.8090169943749482$

```
9-circle
```

$\lambda_2 = 0.766044443118978$

```
9-circle-with-extra-edge
```

$\lambda_2 = 0.7675918792439985$

4 Part 2

[197]:
```
def route_length(D, idx):
    """Computes total length of route visiting the specified idx in order."""
    return sum([D[idx[i], idx[i+1]] for i in range(len(idx) - 1)])

def load_distance():
    """Loads a matrix containing all pair-wise distances between all parks."""
    data = pd.read_csv(os.path.join(Globals.DATA_PATH, 'parks.csv'))
    X = np.stack((data['Longitude'], data['Latitude'])).T
    n = np.shape(X)[0]
    X2 = np.diag(np.dot(X, X.T)).reshape((n, 1))
    Y2 = X2.reshape((1, n))
    XY = np.dot(X, X.T)
    D = np.sqrt(X2 + Y2 - 2*XY)
    park_to_idx = {name: i for i, name in enumerate(data['Name'])}
    idx_to_park = {i : name for name, i in park_to_idx.items()}

    # Sanity checks.
    assert np.allclose(D[park_to_idx['Acadia'], park_to_idx['Arches']], 41.75,␣
↪rtol=1e-4)
    assert np.allclose(route_length(D, list(range(n)) + [0]), 491.92)
    return D, park_to_idx, idx_to_park
```

4.1 Part 2a

```
[287]: def mcmc(D, n_iters, T, route_fn):
           """Implements MCMC for Eucledian Traveling Salesmen on Distance Matrix D."""
           route = np.array(range(D.shape[0]))
           np.random.shuffle(route)
           best = route.copy()

           selected = np.random.choice(range(D.shape[0]), replace=True, size=n_iters)
           selected2 = np.random.choice(range(D.shape[0]), replace=True, size=n_iters)
           route_dists = []
           for idx, idx2 in zip(selected, selected2):
               newroute = route.copy()
               route_fn(route, newroute, idx, idx2)
               route_dist = route_length(D, route.tolist() + [route[0]])
               newroute_dist = route_length(D, newroute.tolist() + [newroute[0]])
               route_dists.append(route_dist)
               delta_dist = newroute_dist - route_dist
               if delta_dist < 0 or (T > 0 and np.random.uniform() < np.
       ↪exp(-delta_dist / T)):
                   route = newroute
               if route_length(D, route.tolist() + [route[0]]) < route_length(D, best.
       ↪tolist() + [best[0]]):
                   best = route
           return best, route_dists
```

4.2 Part 2b

```
[225]: def sequential_modify(route, newroute, idx, idx2):
           del idx2
           next_idx = (idx + 1) % D.shape[0]
           newroute[idx] = route[next_idx]
           newroute[next_idx] = route[idx]
```

```
def problem2b():
    D, park_to_idx, idx_to_park = load_distance()
    for T in [0, 1, 10, 100]:
        best_length = None
        for trial in range(10):
            _, y = mcmc(D, n_iters=10000, T=T, route_fn=sequential_modify)
            plt.plot(range(len(y)), y, label=f'trial_idx={trial + 1}')
            if best_length is None or min(y) < best_length:
                best_length = min(y)

        plt.legend()
        plt.title(f'Current Route by Iteration for T={T}')
        plt.xlabel('Iteration')
        plt.ylabel('Route Distance')
        plt.savefig(f'figures/route_by_iteration_t={T}.png', format='png')
        plt.close()

        print(f'[T={T}]The best found route over all trials was: {best_length:.
 ↪2f}.')
```

[226]:
```
problem2b()
```

```
[T=0]The best found route over all trials was: 375.42.
[T=1]The best found route over all trials was: 320.64.
[T=10]The best found route over all trials was: 279.21.
[T=100]The best found route over all trials was: 358.89.
```

4.3 Problem 2c

[231]:
```
def random_modify(route, newroute, idx, idx2):
    newroute[idx] = route[idx2]
    newroute[idx2] = route[idx]

def problem2c():
    D, park_to_idx, idx_to_park = load_distance()
    for T in [0, 1, 10, 100]:
        best_length = None
        for trial in range(10):
            _, y = mcmc(D, n_iters=10000, T=T, route_fn=random_modify)
            if best_length is None or min(y) < best_length:
                best_length = min(y)
            plt.plot(range(len(y)), y, label=f'trial_idx={trial + 1}')
        plt.legend()
        plt.title(f'Current Route by Iteration for T={T}')
        plt.xlabel('Iteration')
        plt.ylabel('Route Distance')
        plt.savefig(f'figures/route_by_iteration_t={T}_random.png',␣
↪format='png')
        plt.close()

        print(f'[T={T}]The best found route over all trials was: {best_length:.
↪2f}.')
```

[232]:
```
problem2c()
```

```
[T=0]The best found route over all trials was: 200.58.
[T=1]The best found route over all trials was: 166.23.
[T=10]The best found route over all trials was: 252.79.
[T=100]The best found route over all trials was: 323.23.
```

5 Part 3

6 Part 3a

[294]:
```
import importlib
import qwop
importlib.reload(qwop)
from matplotlib import animation
```

[282]:
```
def draw_simulation(data, path):
    fig = plt.figure()
    fig.set_dpi(100)
    fig.set_size_inches(12, 3)

    ax = plt.axes(xlim=(-1, 10), ylim=(0, 3))

    joints = [5, 0, 1, 2, 1, 0, 3, 4]
    patch = plt.Polygon([[0, 0], [0, 0]], closed=None, fill=None, edgecolor='k')
    head = plt.Circle((0, 0), radius=0.15, fc='k', ec='k')

    def init():
        ax.add_patch(patch)
        ax.add_patch(head)
        return patch, head

    def animate(j):
        points = zip([data[j][0][i] for i in joints],
                     [data[j][1][i] for i in joints])
        patch.set_xy(list(points))
        head.center = (data[j][0][5], data[j][1][5])
        return patch, head
    anim = animation.FuncAnimation(fig,
                                   animate,
                                   init_func=init,
                                   frames=len(data),
                                   interval=20)


    anim.save(f'{path}.mp4', fps=50)
    plt.close()
```

```
[350]: def mcmc_sim(n_iters):
           """Implements MCMC QWOP Simulation"""
           route = np.random.uniform(low=-1, high=1, size=40)
           route_value = qwop.sim(route, include_data=False)[0]
           best = route.copy()
           best_value = route_value
           route_dists = [route_value]
           for i in range(n_iters):
               newroute = np.clip(route + np.random.uniform(-0.1, 0.1, size=route.
        ↪shape), -1, 1)
               newroute_value = qwop.sim(newroute, include_data=False)[0]
               if newroute_value > route_value:
                   route = newroute
                   route_value = newroute_value
               if route_value > best_value:
                   best = route
                   best_value = route_value
                   print(f'New best: {best_value:.2f} at iteration {i + 1}.')
           return best, best_value, route_dists
```

```
[ ]: best_plan = None
     best_value = 0
     for _ in range(10):
         small_plan, small_value, _ = mcmc_sim(n_iters=20000)
         if best_plan is None or best_value < small_value:
             best_plan = plan
             best_value = small_value
```

```
New best: 1.39 at iteration 1.
New best: 3.13 at iteration 2.
New best: 3.23 at iteration 307.
New best: 3.26 at iteration 517.
New best: 3.31 at iteration 696.
New best: 3.32 at iteration 746.
New best: 3.49 at iteration 915.
New best: 3.79 at iteration 948.
New best: 3.93 at iteration 955.
New best: 4.02 at iteration 1111.
New best: 4.12 at iteration 1445.
New best: 4.69 at iteration 1479.
New best: 4.88 at iteration 1795.
```

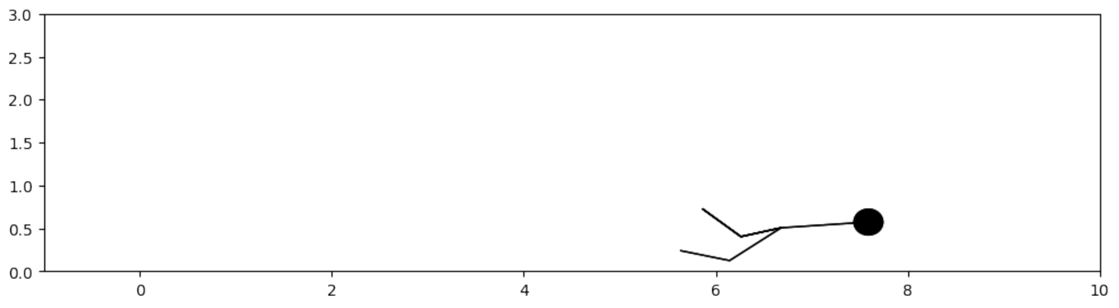
```
[342]: plan, value, _ = mcmc_sim(n_iters=10000, T=0)
```

```
New best: 1.62 at iteration 2.
New best: 1.69 at iteration 4.
New best: 1.74 at iteration 5.
New best: 1.90 at iteration 6.
New best: 2.19 at iteration 7.
New best: 2.56 at iteration 12.
New best: 2.65 at iteration 27.
New best: 2.68 at iteration 87.
New best: 2.76 at iteration 88.
New best: 3.11 at iteration 89.
New best: 3.12 at iteration 99.
New best: 3.65 at iteration 117.
New best: 3.84 at iteration 119.
New best: 4.02 at iteration 159.
New best: 4.05 at iteration 169.
New best: 4.27 at iteration 247.
New best: 4.33 at iteration 331.
New best: 4.35 at iteration 433.
New best: 4.51 at iteration 437.
New best: 4.83 at iteration 451.
New best: 4.91 at iteration 541.
New best: 5.04 at iteration 661.
New best: 5.06 at iteration 689.
New best: 5.08 at iteration 1124.
New best: 5.13 at iteration 1337.
New best: 5.22 at iteration 1393.
New best: 5.32 at iteration 1568.
New best: 5.54 at iteration 1595.
New best: 5.77 at iteration 1759.
New best: 5.87 at iteration 1906.
New best: 6.00 at iteration 2039.
New best: 6.01 at iteration 2127.
New best: 6.05 at iteration 2190.
New best: 6.11 at iteration 2711.
New best: 6.29 at iteration 2809.
New best: 6.32 at iteration 2907.
New best: 6.41 at iteration 2951.
New best: 6.41 at iteration 3013.
New best: 6.62 at iteration 3094.
New best: 6.68 at iteration 3191.
New best: 6.73 at iteration 3309.
New best: 6.82 at iteration 3343.
New best: 6.87 at iteration 4398.
New best: 6.94 at iteration 5045.
New best: 7.06 at iteration 5134.
New best: 7.25 at iteration 5290.
New best: 7.26 at iteration 5310.
New best: 7.35 at iteration 9263.
New best: 7.38 at iteration 9364.
New best: 7.59 at iteration 9954.
```

```
[357]: plan
```

```
[357]: array([ 3.32046582e-01,  8.64347977e-01, -5.36958882e-01,  1.33757192e-01,
              -7.46647497e-02,  5.20644096e-02,  1.38249558e-01,  4.68027199e-01,
               2.38875307e-01,  5.97549589e-01, -1.31011744e-01,  3.00758848e-01,
              -8.75100380e-01, -1.00000000e+00,  5.33726082e-01,  7.10140454e-01,
               8.44800416e-01,  1.00000000e+00, -6.50942301e-01, -8.15635219e-01,
              -6.90576364e-01,  4.99978093e-01, -9.96742325e-01, -6.18842000e-01,
               2.73972411e-01, -5.08626797e-01, -5.25919503e-01, -3.93425059e-01,
              -8.71606742e-01, -3.69966358e-01,  8.81972954e-02, -2.36058686e-01,
              -9.45380799e-01, -4.99008952e-01,  4.76229721e-01, -6.72647790e-04,
               7.42384034e-01, -5.09676613e-01, -8.47599711e-01,  8.12253360e-01])
```

```
[356]: _, data = qwop.sim(plan)
       draw_simulation(data, 'figures/best_mcmc')
```



```
[298]: best, routes[]
```

```
[298]: (array([-0.78027422,  0.0309237 ,  0.01693435, -0.655999  , -0.95880106,
               -0.96315125, -0.76193845, -0.86024011,  0.75840799,  0.66953856,
                0.85208282, -0.47579035,  0.82885493,  0.73460846, -0.15858533,
                0.98560189,  0.42121195,  0.57024489, -0.79955204, -0.128188  ,
               -0.86051161, -0.77008358,  0.43200857, -0.4883419 ,  0.48698762,
                0.12144517,  0.56585249, -0.29193376,  0.53723171,  0.58063392,
                0.51053619,  0.60100035, -0.80468913, -0.43719093,  0.64147052,
                0.56480972, -0.64413858,  0.44152763, -0.33662148,  0.27307277]),
[-0.371670043066106,
 -0.371670043066106,
 -0.371670043066106,
 -0.371670043066106,
 -0.371670043066106,
 -0.27609286547633904,
 -0.27609286547633904,
 -0.1123038006366276,
 -0.1123038006366276,
 -0.1123038006366276,
 -0.1123038006366276,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09960110423113729,
 -0.09470513131941656,
 -0.09470513131941656,
 -0.09470513131941656,
 -0.08191527547612243,
 -0.08191527547612243,
 0.00018221104613973943,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.010806260723276742,
 0.013684976142552749,
 0.013684976142552749,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.021271837437210082,
 0.10612414679026541,
 0.10612414679026541,
 0.10612414679026541,
 0.10612414679026541,
 0.10612414679026541,
 0.10612414679026541,
 0.12543391255586805,
 0.12543391255586805,
 0.12543391255586805,
 0.14810368605370622,
 0.14810368605370622,
 0.14810368605370622,
 0.14810368605370622,
 0.34441804030782863,
 0.34441804030782863,
 0.34441804030782863,
 0.34441804030782863,
 0.34441804030782863,
 0.34441804030782863,
 0.34441804030782863,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 0.35286534820448445,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.0537216089353647,
 2.1849990574940907,
 2.1849990574940907,
 2.1849990574940907,
 2.1849990574940907,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.283366875391518,
 2.3556983192702723,
 2.3556983192702723,
 2.3556983192702723,
 2.3759417065901687,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.5401186465592787,
 2.556336811341475,
 2.556336811341475,
 2.556336811341475,
 2.556336811341475,
 2.635763751161244,
 2.635763751161244,
 2.635763751161244,
 2.635763751161244,
 2.635763751161244,
 2.782170789203952,
 2.8746280499840196,
 2.8746280499840196,
 2.8746280499840196,
 2.8749769517703188,
 3.1943690761050805,
 3.1943690761050805,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.1983341325379913,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.265926223057849,
 3.2724967881259914,
 3.2724967881259914,
 3.2724967881259914,
 3.2724967881259914,
 3.2724967881259914,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.338221284823599,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.3562120810594274,
 3.374821757005733,
 3.374821757005733,
 3.374821757005733,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.403135436070362,
 3.4895873294142263,
 3.5045571304241285,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.113717189408771,
 4.132217231101603,
 4.132217231101603,
 4.132217231101603,
 4.132217231101603,
 4.132217231101603,
 4.132217231101603,
 4.132217231101603,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.139821053083445,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.140094264726064,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.144498012790248,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.154084411429212,
 4.163179937422711,
 4.163179937422711,
 4.163179937422711,
 4.163179937422711,
 4.163266913948373,
 4.163266913948373,
 4.163266913948373,
 4.163266913948373,
 4.163266913948373,
 4.163266913948373,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.180102354107199,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.217630564292433,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.277242898993564,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.342250120054057,
 4.468937746069372,
 4.468937746069372,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.5292107469380785,
 4.58925799518465,
 4.58925799518465,
 4.58925799518465,
 4.58925799518465,
 4.58925799518465,
 4.58925799518465,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.671541803910155,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.797747960783125,
 4.919290961368887,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.359378280939781,
 5.403038345782456,
 5.403038345782456,
 5.403038345782456,
 5.462406081867705,
 5.462406081867705,
 5.462406081867705,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.509006369355819,
 5.5600880139145685,
 5.5600880139145685,
 5.5600880139145685,
 5.5600880139145685,
 5.5600880139145685,
 5.5600880139145685,
 5.5600880139145685,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.851560138186448,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.856860387402659,
 5.9150618660519045,
 5.9150618660519045,
 5.9150618660519045,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.92664982293432,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.965683295503818,
 5.977595062680409,
 5.9821723229679575,
 5.9821723229679575,
 5.9821723229679575,
 5.9821723229679575,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.005574649872068,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.023980643008962,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.045340818343567,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.0478459855747175,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.072080920034159,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.122548087076995,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.1354198806871425,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.166735969536807,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.194114419173594,
 6.341963874929001,
 6.341963874929001,
 6.341963874929001,
 6.341963874929001,
 6.341963874929001,
 6.341963874929001,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.345478847572818,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939,
 6.354647426118939])
```

```
[300]:
```

```
[300]: 8.064499999999999
```

```
[ ]:
```