CS168, Spring 2017

Mini-Project #7

Due by 11:59 PM on Tuesday, May 23rd.

Instructions

- You can work individually or with one partner. If you work in a pair, both partners will receive the same grade.
- If you've written code to solve a certain part of a problem, or if the part explicitly asks you to implement an algorithm, you must also include the code in your pdf submission. See the problem parts below for instructions on where in your writeup to put the code.
- Make sure plots you submit are easy to read at a normal zoom level.
- Detailed submission instruction can be found on the course website (`http://cs168.stanford.edu`) under the "Coursework - Assignment" section. If you work in pairs, only one member should submit all of the relevant files.
- a) Use 12pt or higher font for your writeup. b) Code marked as "Deliverable" gets pasted into the relevant section, rather than into the appendix (though feel free to put it in both). Keep variable names consistent with those used in the problem statement, and with general conventions. No need to include import statements and other scaffolding, if it is clear from context. Also, please use the `verbatim` environment to paste code in LaTeX from now on, rather than the listings package:

```
def example():
print "Your code should be formatted like this."
```

- **Reminder:** No late assignments will be accepted, but we will drop your lowest mini-project grade when calculating your final grade.

Part 1: Markov Chains

Goal: In this exercise you will build intuition on Markov chains and their properties.

Description: For each of the following three graphs, consider a Markov chain defined by a random walk over the nodes of the graph (specifically, the states of the chain correspond to nodes of the graph, and at each time step, the chain transitions to a random neighbor of the current state).

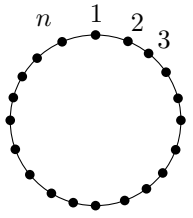

Figure 1: The circle graph.

1. The circle graph (Figure 1) with $n = 10$.

2. The circle graph with $n = 9$.

3. The circle graph with $n = 9$, with an extra edge connecting nodes 1 and 5.

(a) (3 points) For each Markov chain, explain whether it is aperiodic and irreducible. If it is aperiodic and irreducible, what is its stationary distribution, π? (Recall that a Markov chain is irreducible if for states s_i, s_j, it is possible to eventually get to s_j starting from s_i).

The *total variation distance* between two probability distributions D_1, D_2 supported on a domain S is defined as $\|D_1 - D_2\|_{TV} = 0.5 \sum_{x \in S} |D_1(x) - D_2(x)|$.

(b) (5 points) Given a Markov chain and an initial state s, let $D_s(t)$ denote the distribution of the state of the chain after t transitions, given that it is in state s at time $t = 0$ (hence $D_s(0)$ is the distribution that assigns probability 1 to state s, and 0 to all other states). Choose an initial state s, and plot $\|D_s(t) - \pi\|_{TV}$ for $t \in \{0, 1, 2, \ldots, 100\}$ for each of the three Markov chains (corresponding to the three graphs). Plot the three lines on the same plot, clearly labeling which is which. For the Markov chains that are not aperiodic and irreducible, set π to be the uniform distribution in the total variation distance calculations.

(c) (3 points) Report the values of λ_2, the second largest eigenvalue of the *transition matrix*, for each Markov chain.

(d) (6 points) Discuss the results of parts (b) and (c), and their connections. You should specifically mention the mixing time and the power iteration algorithm.

Deliverables: Answers for part (a). Plot, code for part (b). Eigenvalues for part (c). Discussion for part (d).

Background

How long does it take to visit 30 national parks?[1] The TAs plan on having an epic road trip this summer, and have picked 30 of the 59 national parks to visit. A proper route starts and ends at the same national park, and visits each park exactly once. Your goal in this project is to help them find the shortest route possible.

In the file `parks.csv`, you'll find the longitude and latitude of the 30 national parks *in alphabetical order of the park names*. To simplify the problem, we assume that the distance between two national parks is as follows:[2]

$$\text{distance} = \sqrt{(\text{longitude}_1 - \text{longitude}_2)^2 + (\text{latitude}_1 - \text{latitude}_2)^2}$$

As a sanity check, the distance between Acadia to Arches (the first two in the file) is 41.75 units, and the length of the route that visits all parks in alphabetical order (Acadia $\to$ Arches $\to$ Badlands $\to \cdots \to$ Acadia) is 491.92 units.

Part 2: Markov Chain Monte Carlo (MCMC)

Goal: To gain experience exploring a vast search space using Markov Chain Monte Carlo (MCMC). Also, to explore the cost-benefit trade-off as one interpolates between local search and random search via a "temperature" parameter.

[1] This problem is a version of the Traveling Salesman Problem (TSP): `http://en.wikipedia.org/wiki/Travelling_salesman_problem`.

[2] The proper measure (without considerations for traffic, elevation, or the existence of roads) is Vicenty's formulae (`http://en.wikipedia.org/wiki/Vincenty's_formulae`), but we'll spare you the trouble.

Description: In this part, you will implement MCMC to explore the space of traveling salesman tours of the national parks. The MCMC search will be parameterized by a "temperature" parameter, T, which governs the trade-off between local search and random search. To begin with, consider the MCMC algorithm that starts with a random permutation of the N parks; at each iteration, randomly select two successive parks, and propose the route defined by switching their order. If the proposed route has a lower total distance than the current route, set the current route to be the proposed route; otherwise, if $T > 0$, with probability $e^{-\Delta_d/T}$, update the current route to be the proposed route, where Δ_d is the increase in total distance.

Repeat this process MAXITER times. The algorithm outputs the shortest route found during the MAXITER iterations (which need not be the last route). The following is pseudo-code for the above algorithm:

---

route ← random permutation of the N parks
best ← route
for i = 1, 2, . . . , MAXITER **do**
    route$_{new}$ ← route with the order two random successive parks reversed
    Δ_d = distance(route$_{new}$) - distance(route)
    if $\Delta_d < 0$ OR ($T > 0$ AND random() $< e^{-\Delta_d/T}$) **then**
        route ← route$_{new}$
    end if
    if distance(route) < distance(best) **then**
        best ← route
    end if
end for

(a) (3 points) Implement the MCMC algorithm. How many states are in this Markov chain? If MAXITER tends to infinity, will you eventually see all possible routes? [Hint: consider separately the cases of $T = 0$ and $T > 0$.]

(b) (5 points) Set MAXITER to be $10,000$. Run the algorithm in four different regimes, with $T \in \{0, 1, 10, 100\}$. For each value of T, run 10 trials. Produce one figure for each value of T. In each figure, plot a line for each trial: the iteration number against the length of the current value of route during that iteration. (Your solution to this part should have 4 figures, and each plot should contain 10 lines.) Among the 4 values of T, which one seems to work the best?

(c) (5 points) Modify the above MCMC algorithm as follows: in each iteration select two parks at random (*not necessarily successive parks*) and propose the route obtained by switching them. Repeat the experiments from part (b). Among the 4 values of T, which one seems to work the best now?

(d) (5 points) Describe whatever differences you see relative to part (b), and propose one or more explanations for why this Markov Chain performs differently from the previous one. (You may wish to think about this part in the context of the investigation of the mixing times of the different Markov chains in the first problem of this mini-project.)

Deliverables: Code and answers for part (a). Figures and answer for parts (b) and (c). Discussion for part (d).

Part 3: QWOP[3]

Goal: In this exercise you will get some hands-on experience solving a tricky optimization problem that captures some of the difficulties of some real-world robotics problems.

[3]This part of the assignment, and the implementations of the QWOP physics engine, are based on an assignment developed by Paul Valiant at Brown University, and we thank him for sharing his code, etc.

Description: In 2010, the QWOP game[4] took the Internet by storm. The purpose of the game is to get your avatar to run the 100-meter event at the Olympic Games by controlling their thigh and knee angles. If you haven't played the game, give it a try—this assignment will be much more amusing if you have first spent a few minutes with the flash game; even getting your runner to fall forwards rather than backwards can be quite a challenge.

Rather than learning how to play the game, you'll just write a computer program to do it for you! We have implemented the QWOP physics engine in Python and MATLAB. These functions take a list of 40 floating point numbers as input (corresponding to 20 instructions for the angle of the thighs and 20 for the angle of the knees). The output of the function is a single number, representing the final $x-$position of the head of the avatar. The more efficiently you get the avatar to run, the further it will go and the larger the output number. The goal of this problem is for you to find an input that makes the avatar run as far as possible—namely find a length 40 vector that results in the QWOP code evaluating to as large a number as possible.

For your amusement, the MATLAB and Python implementations also provide an animation of the movements of the avatar on any given input.

(a) *[do not hand in]* Make sure you can call either the python `qwop(plan)` function, or the MATLAB `qwop(plan)` functions with a `plan` consisting of 40 floating point numbers in the range $[-1, 1]$. The MATLAB and Python implementations come with a visualization of the game, and we highly encourage you to look at what actually happens. (In the MATLAB implementation, you can suppress the animation by including an additional argument of "0" in the function call, ie `qwop(plan, 0)`.)

(b) (20 points) Optimize QWOP. Let d be the distance you get your avatar to go. The number of points you receive will be $\frac{d^2}{5}$ with anything above 20 points regarded as bonus. There will be a small prize for the person/group that gets the furthest, and for the group with the most amusing result. [Upload a video of your final run to YouTube—see the "Deliverables" section below.]

Note: Please do NOT use optimization code that you found online. Design your own MCMCs, variants of gradient descent, etc.

(c) (5 points) Describe which techniques you tried, and whether or not they were successful. Also report the best distance you get.

Deliverables: For part (b), the code, input and a link to a YouTube video of the run (the Python code gives the option to save to an mp4 video; you may need to install ffmpeg for the code to work; if you are using MATLAB or do not want to install the extra python package, just record the animation on your phone and upload it). The input `plan` *has* to be formatted as either a MATLAB vector or Python list: e.g. `[0.1, -0.3, ..., 0.21]`. Discussion for part (c) and best distance your method achieves.

[4] `http://www.foddy.net/Athletics.html`