David Zhao
Benjamin Quiring
MATH7241: Project Report November 27, 2018

1 Data

We downloaded a history of terminal commands for a UNIX computer from Perdue back in 1998. A sample of the data looked like:

```
cd
<1>
ll
vi
<1>
ll
emacs
```

Where a `<#>` represents the number of arguments given to a command. This was done to preserve the user's privacy. We cleaned the data as follows:

1. We combined a command with the number of arguments passed with it. For example `cd<1>` is different from `cd` which is also different from `cd<2>`.
2. We ignored operators like `; , -, |` and didn't consider them commands.
3. We ignored the `**SOF**` and `**EOF**` commands because they were added and not actually typed by the user.

After cleaning up the data as described above, we noticed that there were over 700 unique commands. We chose the 28 most common occuring commands to be the states in our Markov chain and proceeded to further filter the data by deleting commands that are not the 28 most common. The resulting dataset contained only the top 28. In order to preserve some semblance of the chain, if there was a sequence $Y \to X_1 \to ... \to X_n \to Z$ with Y and Z in the top 28 and $X_1, ..., X_n$ not, then this reduced to $Y \to Z$; i.e. we tried to keep edges. If we did not do this, then we would just have very strange fragments (and the plot of the chain would not work). Although it loses information, it seems to work pretty well.

Additionally, there were nine different users (two were the same person, but different machines). Since each person uses different commands (different preferences in text editors, as well as different alias's for commands). We ended up choosing the user with the most data and ignored the rest, as that would not really be a Markov process.

There were over twenty thousand time steps in the filtered data, which should be enough to have a meaningful chain.

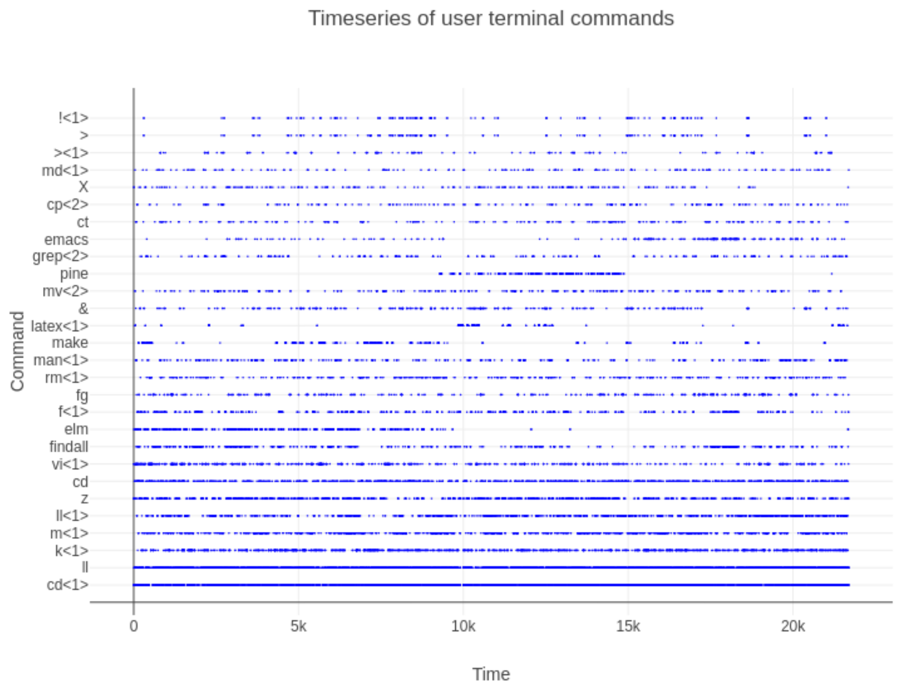
We plotted the entire time series and the first 250 steps. As the entire time series is very wild, we include the 250 step one for analysis.

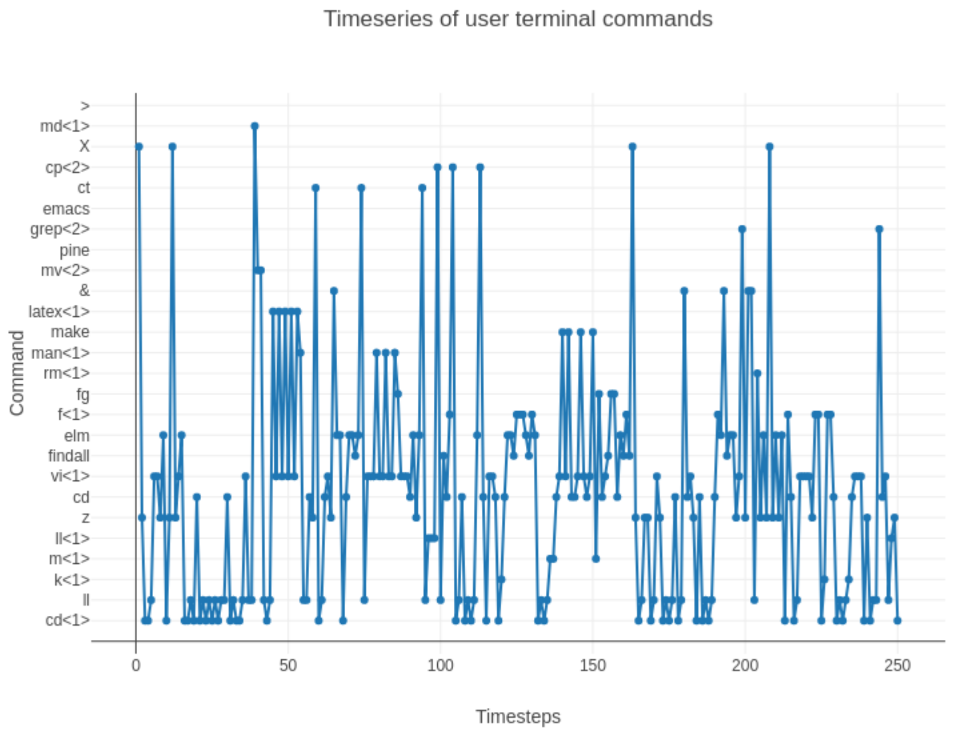
Also, note for the timeseries plot (on 250 steps) how there are several back and forths between states; i.e. when trying to solve a problem usually alternating between two commands (ignoring other sparsely used commands) usually does the trick. As an example, consider `cd` and `ll` (note this happens quite a bit in the chain). If I am navigating around a coding project looking for files, I will alternate between moving to a new directory (`cd`) and looking at the files in that directory (`ll`) to see where to go next. Once I've found my file, I'll switch to another command such as the text editor.

As another example, switching between `vi`, i.e. text editting, and `latex` or `make` (which builds code) is also a common occurence.

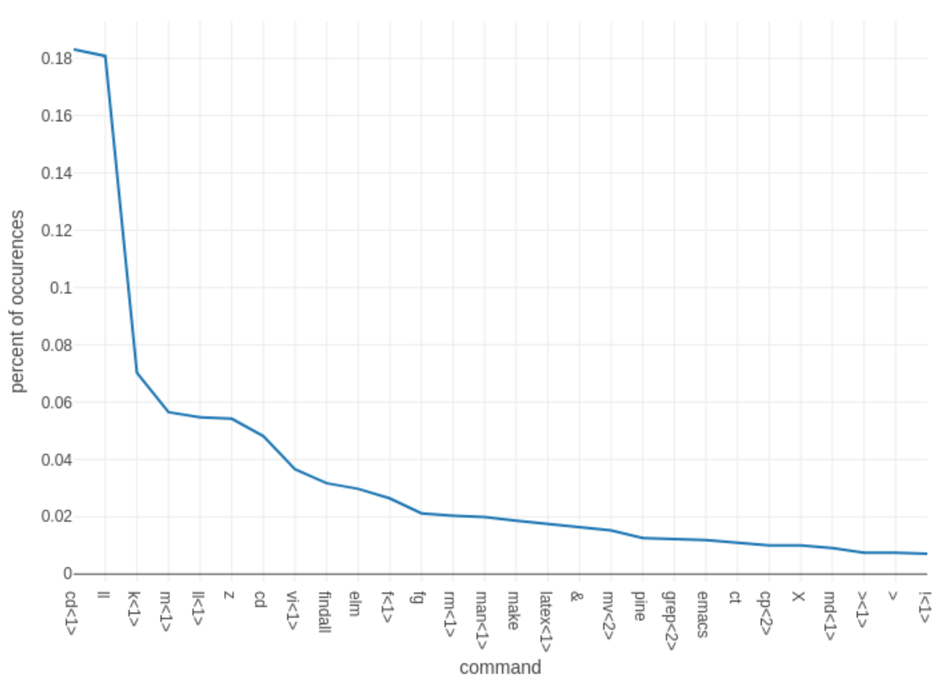
We calculated the occupation frequencies of the emperical data by counting the number of times each state was visited and graphed the distribution.

Even though the data is from 1998 the top 5 commands seem reasonable given our own experience using UNIX based operating systems. The majority of the time a user will be navigating their computer's file


Timeseries of user terminal commands
!<1>
>
><1>
md<1>
X
cp<2>
ct
emacs
grep<2>
pine
mv<2>
&
latex<1>
make
man<1>
rm<1>
fg
f<1>
elm
findall
vi<1>
cd
z
ll<1>
m<1>
k<1>
ll
cd<1>
Command
0
5k
10k
15k
20k
Time


Timeseries of user terminal commands
>
md<1>
X
cp<2>
ct
emacs
grep<2>
pine
mv<2>
&
latex<1>
make
man<1>
rm<1>
fg
f<1>
elm
findall
vi<1>
cd
z
ll<1>
m<1>
k<1>
ll
cd<1>
Command
0
50
100
150
200
250
Timesteps

Command Occupation Frequency Distribution



directory with commands like `cd` and `ll`. `cd<1>` navigates to the given directory, while `ll` gives a detailed description of files in current directory (name, owner, modified date, size, etc.). Other common commands include `rm`, `cp`, and mv for file manipulation, `grep` for searching, `latex` for latex, and `emacs` and `vi` for text editting.

Next we computed the transition matrix by counting the number of times the chain went from state i to state j followed by dividing each count P_{ij} by the number of times we moved out of state i.

We plotted the transition matrix as a heatmap, as we believe that this provides a much better understanding of the transitions of the chain.

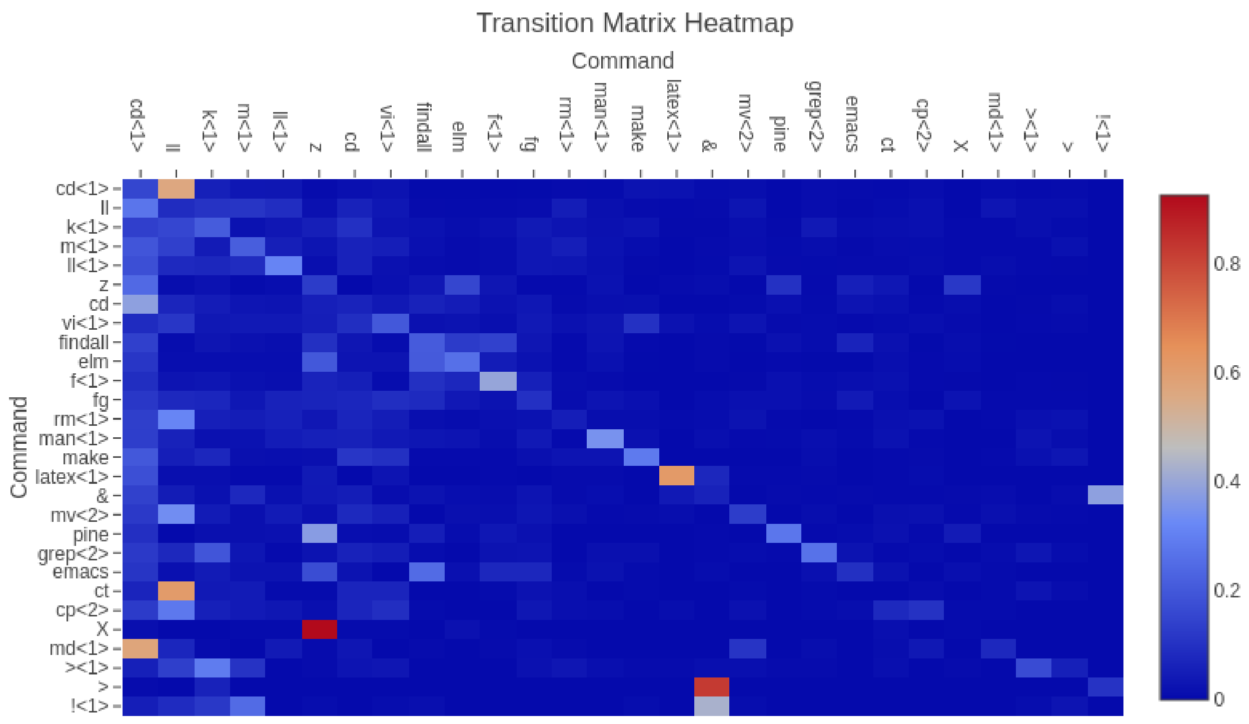
Note the high(er) frequency of the diagonal; after typing a command, you are more likely to type it again that typing most others. Additionally, the column for `cd<1>` is brighter than most; after every command you are decently likely to navigate to another directory.

We raised the transition matrix to the power 1000 to compute the stationary distribution. We argue that the original chain is irreducible because the data tracks the user's usage of the UNIX machine over two years. This means that there are many instances of logging in and out of the machine, which is done with the same set of commands every time (`rlogin` and `exit`). So the only instance where a state i is unreachable from j is if the user types a command and then doesn't proceed to logoff or doesn't proceed to log on.

Both cases are impossible because a command isn't tracked until the user has logged on and since we're tracking the 28 most common commands there is no way that one of these commands are used so frequently while the user never exit afterwrods. Therefore, since the chain always visits the state `exit` every state can reach `exit` and as a result every state can reach every other state via the `exit` state.

Additionally, as there are no 0 entries in the transition matrix to the 1000^{th} power, our cleaned chain is irreducible.

Therefore our stationary distribution is unique and when we plot it against the occupation frequency dis-



tribution, there is no surprise that the distributions are identical. The plot of the comparison is in the Appendix.

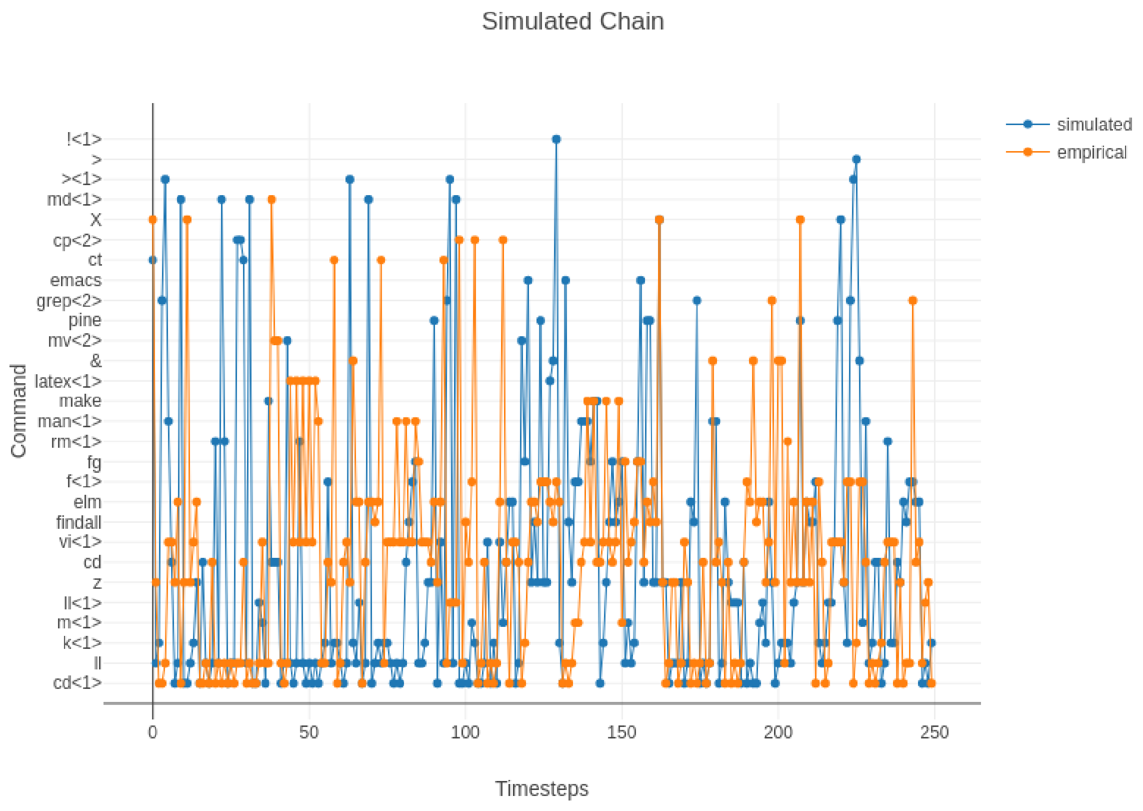
We simluated the chain for 250 time steps (it is too wild to get meaningful images with any more). We chose the initial state uniformly randomly over all 28.

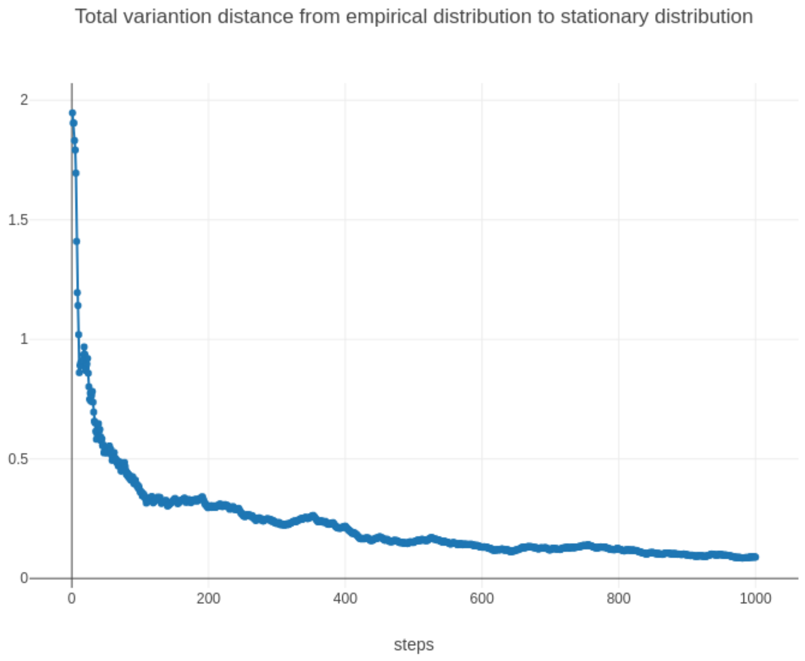
Note how the original chain has many back and forths between lesser (i.e. not `cd` and `ll`) states, as discussed earlier, while the simulated chain does not appear to capture this behavior. In general, the simulated chain appears to lose much of the structure of the original chain, bobbing around meaninglessly between states while the original chain has reoccuring patterns (though still with meaningless jumps up to less used states).

We made some attempts to display what happens in the chain clearer (as our states do not have any notion of continuity between them) but were unable to find a good visualization technique besides the scatter plot.

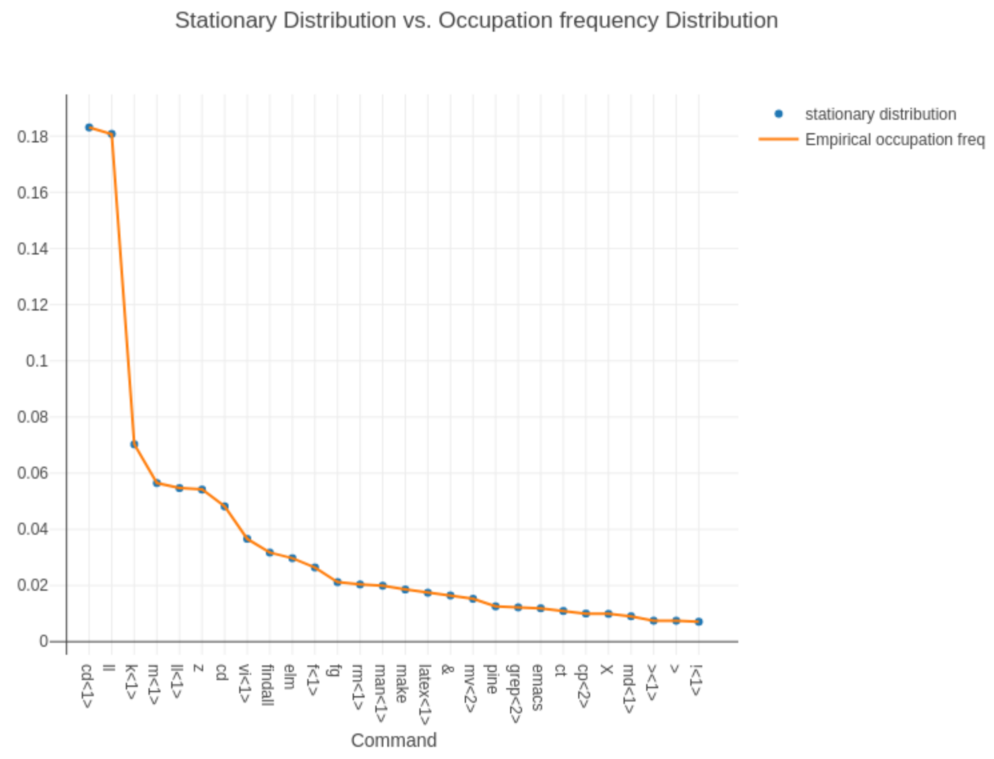
Finally we investigated the mixing time of the chain. We did this by, at each time step, computing the occupation frequency of the chain from the first to the current time step. At each step we calculate the occupational frequency of the states seen so far and subtracted this distribution with the calculated stationary distribution. We then plotted the L_1 norm of the difference.

After 900 steps it has a different of around 0.1, which is quite small. The chain gets to a difference of 0.5 after only 50. The chain appears to converge quite rapidly to the stationary distribution


Simulated Chain
simulated
empirical
Command
!<1>
>
><1>
md<1>
X
cp<2>
ct
emacs
grep<2>
pine
mv<2>
&
latex<1>
make
man<1>
rm<1>
fg
f<1>
elm
findall
vi<1>
cd
z
ll<1>
m<1>
k<1>
ll
cd<1>
0
50
100
150
200
250
Timesteps


Total variantion distance from empirical distribution to stationary distribution
2
1.5
1
0.5
0
0
200
400
600
800
1000
steps

2 Appendix


Stationary Distribution vs. Occupation frequency Distribution
stationary distribution
Empirical occupation freq
0.18
0.16
0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
cd<1>
ll
k<1>
m<1>
ll<1>
z
cd
vi<1>
findall
elm
f<1>
fg
rm<1>
man<1>
make
latex<1>
&
mv<2>
pine
grep<2>
emacs
ct
cp<2>
X
md<1>
><1>
>
l<1>
Command