problem6

February 8, 2023

Fix some dependency issues

[ ]:
```
# !pip uninstall networkx -y
# !pip install networkx==2.6.3
```

```
Found existing installation: networkx 3.0
Uninstalling networkx-3.0:
  Successfully uninstalled networkx-3.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting networkx==2.6.3
  Downloading networkx-2.6.3-py3-none-any.whl (1.9 MB)
                             1.9/1.9 MB
29.6 MB/s eta 0:00:00
Installing collected packages: networkx
Successfully installed networkx-2.6.3
```

[ ]:
```
import numpy as np
```

1 Load Graph

Load data for Caltech graph network (upload file first)

- G <- Total graph
- H <- Subgraph of size N (first N Nodes)
- adj_list <- Adjacency list for H

1.1 IMPORTANT NOTE

This code template converts site nodes into numbers for ease of computation. Specifically, they are numbered 0 to N - 1 so they match up with indices in an array.

nodes_to_idx can be called to convert a site URL to the index that it occupies in an array of PageRanks. *nodes[i]* for an index i will give the site URL

[ ]:
```
# Number of nodes
N = 200
```

problem6

February 8, 2023

Fix some dependency issues

[ ]:
```
# !pip uninstall networkx -y
# !pip install networkx==2.6.3
```

```
Found existing installation: networkx 3.0
Uninstalling networkx-3.0:
  Successfully uninstalled networkx-3.0
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
wheels/public/simple/
Collecting networkx==2.6.3
  Downloading networkx-2.6.3-py3-none-any.whl (1.9 MB)
                             1.9/1.9 MB
29.6 MB/s eta 0:00:00
Installing collected packages: networkx
Successfully installed networkx-2.6.3
```

[ ]:
```
import numpy as np
```

1 Load Graph

Load data for Caltech graph network (upload file first)

- G <- Total graph
- H <- Subgraph of size N (first N Nodes)
- adj_list <- Adjacency list for H

1.1 IMPORTANT NOTE

This code template converts site nodes into numbers for ease of computation. Specifically, they are numbered 0 to N - 1 so they match up with indices in an array.

nodes_to_idx can be called to convert a site URL to the index that it occupies in an array of PageRanks. *nodes[i]* for an index i will give the site URL

[ ]:
```
# Number of nodes
N = 200
```

[ ]:
```
import pickle
import networkx as nx

f = open('nx_caltech_2500.pkl', 'rb')
G = pickle.load(f)

nodes = list(G.nodes())[:N]
H = G.subgraph(nodes)

nodes_to_idx = {}
for i in range(len(nodes)):
  nodes_to_idx[nodes[i]] = i

adj_list = {}
for i in range(len(nodes)):
  adj_list[i] = []
  for j in nx.neighbors(H, nodes[i]):
    adj_list[i].append(nodes_to_idx[j])
```

2 Set Parameters

[ ]:
```
# TODO: Fill in alpha values
alphas = [0.25, 0.45, 0.65, 0.85, 0.95]

# When the max difference is < tolerance, we've "converged"
tolerance = 1e-3

# How many times we run each process (to magnify time differences)
iter = 25
```

3 Potentially Useful Visualizations

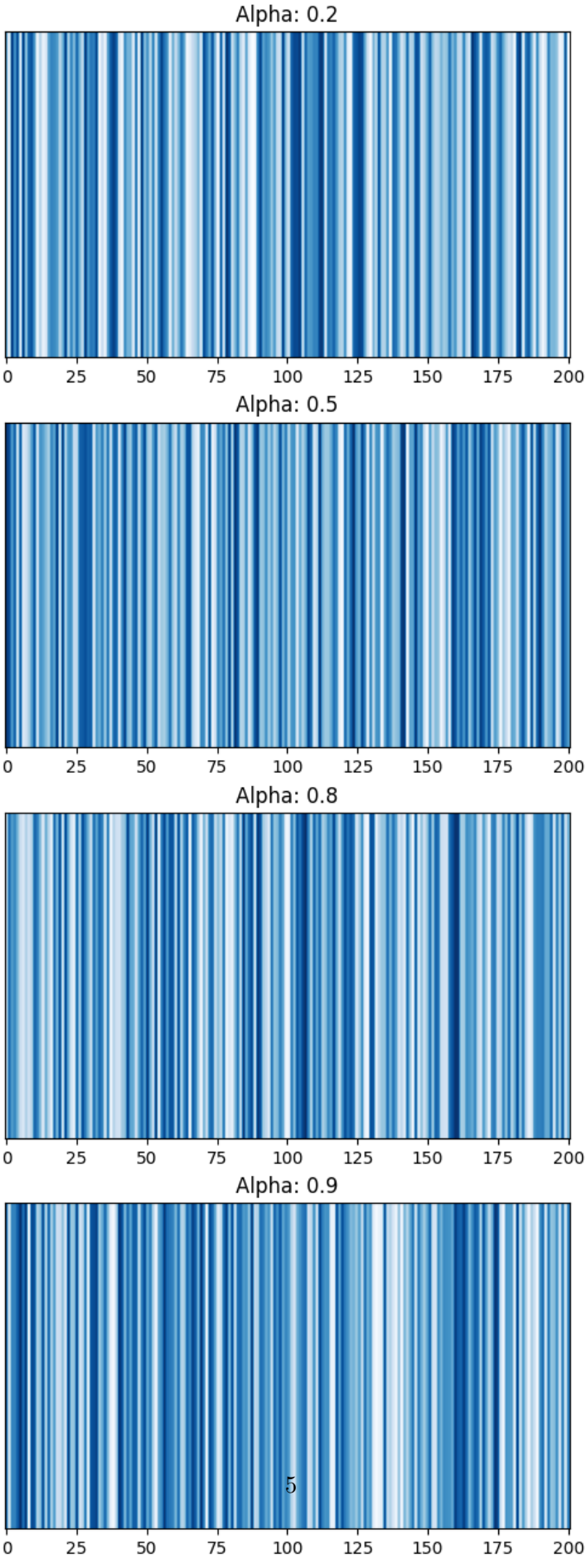
[ ]:
```
import matplotlib.pyplot as plt
from matplotlib import colors

# Plot graph {H} with node colors according to value array {v}
# Darker blue means larger value
# vmin and vmax are the bounds you want to set for the gradient
# See matplotlib vmin/vmax + colormap documentation for more details
def intensity_graph(H, v, vmin=None, vmax=None, title=""):
  vmin = vmin if vmin else np.min(v)
  vmax = vmax if vmax else np.max(v)
  plt.figure()
  plt.title(title)
  nx.draw(
```

```
    H,
    node_color=v, cmap=plt.cm.Blues, node_size=50, vmin=vmin, vmax=vmax,
    width=0.1, alpha=0.4, edge_color='grey'
  )
  plt.show()

# Let M = len(alphas) and N = # of nodes
# pi_matrix is an N x M matrix
# where row i is the PageRank of the N nodes (labelled 0 to N - 1)
# at alpha = alphas[i]
# Each row has their intensity visualized as a 1D heatmap
#
# See IMPORTANT NOTE section for how labelling can be converted to URLs
def draw_heatmap(alphas, pi_matrix):
  eps = 0.5
  extent = [-eps, len(pi_matrix[0]) + eps, 0, 1]
  M = len(alphas)
  fig, axs = plt.subplots(len(alphas), figsize=(6, 4 * M))
  norm = colors.Normalize(np.min(pi_matrix), np.max(pi_matrix))
  for i in range(len(alphas)):
    axs[i].imshow(pi_matrix[i][np.newaxis, :], cmap="Blues", aspect="auto",␣
 ↪extent=extent, norm=norm)
    axs[i].set_yticks([])
    axs[i].set_title("Alpha: {}".format(alphas[i]))
    axs[i].set_xlim(extent[0], extent[1])
  fig.show()
```

[ ]:
```
intensity_graph(H, np.random.rand(N))
```

[ ]:
```
draw_heatmap([0.2, 0.5, 0.8, 0.9], np.random.rand(4, N))
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_29900/2876918579.py:3
8: UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```


Alpha: 0.2
0
25
50
75
100
125
150
175
200
Alpha: 0.5
0
25
50
75
100
125
150
175
200
Alpha: 0.8
0
25
50
75
100
125
150
175
200
Alpha: 0.9
0
25
50
75
100
125
150
175
200

4 Random Surf

[ ]:
```
import numpy as np
import time
from tqdm import tqdm

# TODO:
# Perform one random surf until "converge" using alpha
# Return final distribution of PageRank
def random_surf(alpha):
  curr_node = nodes_to_idx[np.random.choice(nodes)]
  visits = np.zeros(N)
  visits[curr_node] += 1
  old_page_rank = N * visits / 1
  i = 1
  while True:
    if np.random.random() <= alpha:
      neighbors = list(H.neighbors(nodes[curr_node]))
      if len(neighbors) != 0:
        curr_node = nodes_to_idx[np.random.choice(neighbors)]
    else:
      curr_node = nodes_to_idx[np.random.choice(nodes)]
    visits[curr_node] += 1
    i += 1
    new_page_rank = N * visits / i
    if np.max(np.abs(old_page_rank - new_page_rank)) < tolerance:
      break
    old_page_rank = new_page_rank
  return new_page_rank



## RESULTS WILL BE STORED IN THESE ARRAYS
# Final PageRank values (averaged over each alpha)
surf_pis = []
# Time taken (averaged over each alpha)
surf_times = []

# Iterate over alphas
for alpha in alphas:
  pis = []

  # Perform process at alpha level {iter} times
  # Time whole process
```

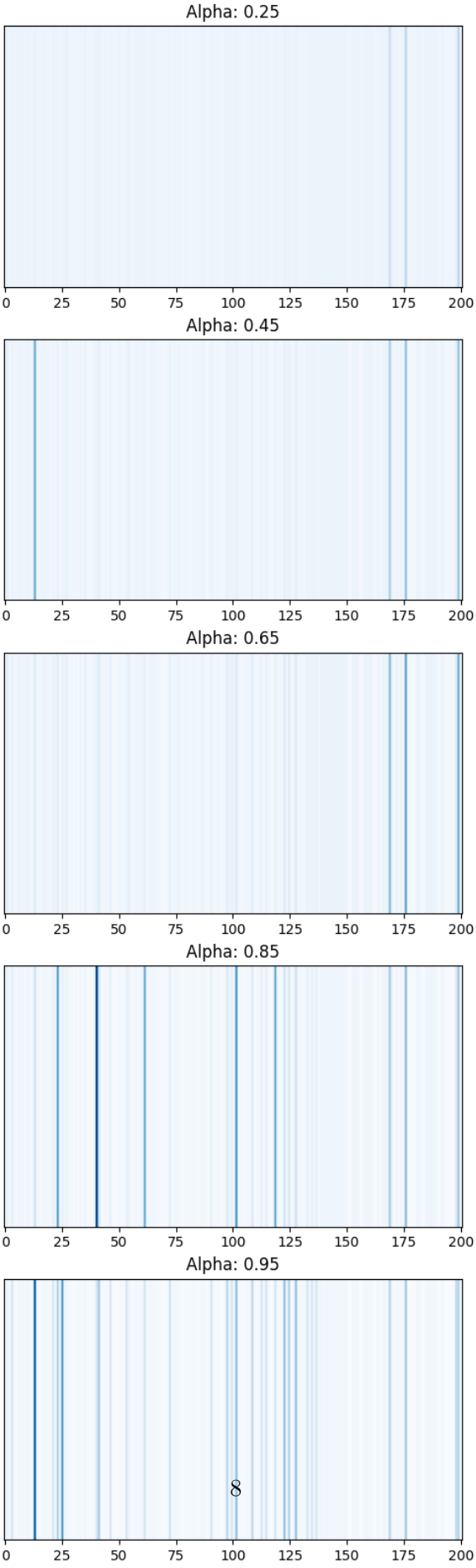
```
  start = time.time()
  for _ in tqdm(range(iter)):
    pis.append(random_surf(alpha))
  end = time.time()

  # Record average time + final value
  pis = np.array(pis)
  pi_avg = np.average(pis, axis=0)
  surf_pis.append(pi_avg)
  surf_times.append((end - start) / iter)
```

```
100%|      | 25/25 [03:22<00:00,  8.10s/it]
100%|      | 25/25 [05:10<00:00, 12.42s/it]
100%|      | 25/25 [06:34<00:00, 15.80s/it]
100%|      | 25/25 [04:18<00:00, 10.34s/it]
100%|      | 25/25 [04:04<00:00,  9.76s/it]
```

4.1 Random Surf Visualizations

[ ]:
```
# TODO: Create visualizations here

# for pi in surf_pis:
#     intensity_graph(H, pi)

draw_heatmap(alphas, surf_pis)
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_29900/2876918579.py:3
8: UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```


Alpha: 0.25
0
25
50
75
100
125
150
175
200
Alpha: 0.45
0
25
50
75
100
125
150
175
200
Alpha: 0.65
0
25
50
75
100
125
150
175
200
Alpha: 0.85
0
25
50
75
100
125
150
175
200
Alpha: 0.95
0
25
50
75
100
125
150
175
200

5 Matrix Iterations

[ ]:
```
P = np.zeros((N, N))
for i, row in enumerate(P):
    neighbors = adj_list[i]
    k = len(neighbors)
    if k != 0:
        P[i, neighbors] = 1/k
    else:
        P[i, i] = k
```

[ ]:
```
# TODO:
# Calculate PageRank via repeated multiplication of matrix using {alpha}

# TIPS:
# See np.linalg.eig for SVD decomposition

# Like Q1 on HW4, apply diagonalization with the focus on taking the diagonal
# matrix to a high power and then applying the other transforms

# Certain decompositions may carry imaginary numbers due to precision
# These numbers should die out with multiple applications of G
# So they should disappear as you take G^n for large n

def matrix_iterations(alpha):
  bigG = P * alpha + (1-alpha)/N * np.ones((N, N))
  pi0 = np.ones(N) * 1/N
  i = 1
  old_pi = (pi0 @ np.linalg.matrix_power(bigG, i))
  old_pi = old_pi / sum(old_pi)
  while True:
    i += 1
    pi = pi0 @ np.linalg.matrix_power(bigG, i)
    pi = pi / sum(pi)
    if np.max(np.abs(old_pi - pi)) < tolerance:
      break
    old_pi = pi
  return pi

# Note that as described, the PageRanks will be a factor of N off from random␣
 ↪surf approximation
matrix_pis = []
matrix_times = []
for alpha in alphas:
```

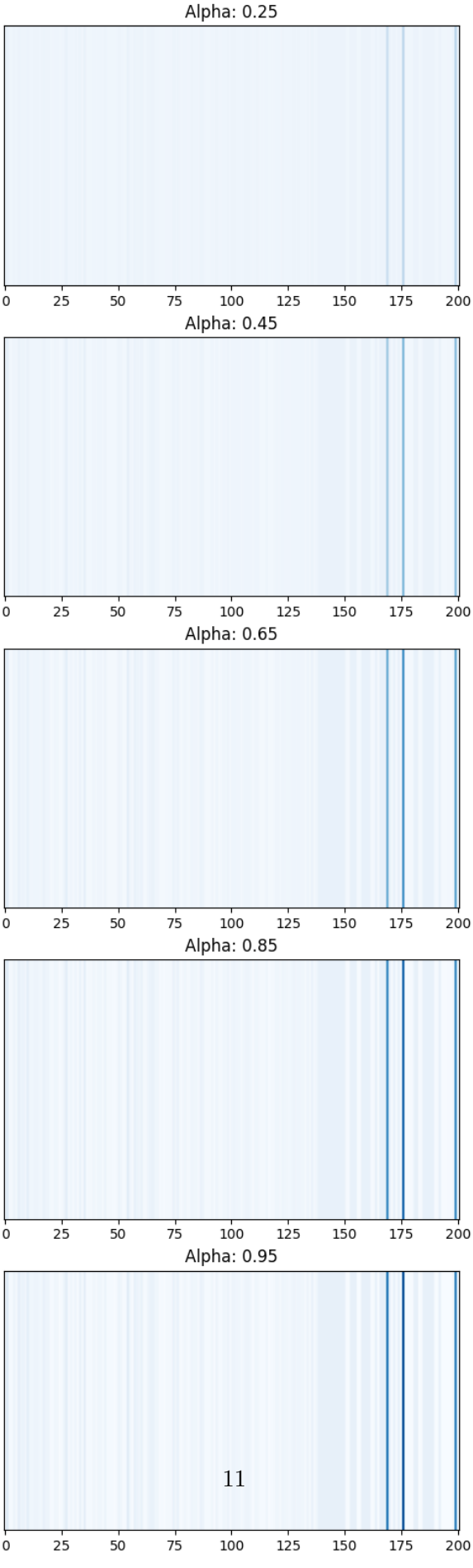
```
  pis = []

  # Even though result is the same, do multiple to average out runtime
  start = time.time()
  for _ in tqdm(range(iter)):
    pis.append(matrix_iterations(alpha))
  end = time.time()

  matrix_times.append((end - start) / iter)
  # Don't really need to have average here since we have deterministic results
  matrix_pis.append(np.average(pis, axis=0))
```

```
100%|      | 25/25 [00:00<00:00, 130.84it/s]
100%|      | 25/25 [00:00<00:00, 155.60it/s]
100%|      | 25/25 [00:00<00:00, 82.23it/s]
100%|      | 25/25 [00:03<00:00,  6.26it/s]
100%|      | 25/25 [00:02<00:00,  8.69it/s]
```

6 Matrix Iterations Visualizations

[ ]:
```
draw_heatmap(alphas, matrix_pis)
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_29900/2876918579.py:3
8: UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```


Alpha: 0.25
0
25
50
75
100
125
150
175
200
Alpha: 0.45
0
25
50
75
100
125
150
175
200
Alpha: 0.65
0
25
50
75
100
125
150
175
200
Alpha: 0.85
0
25
50
75
100
125
150
175
200
Alpha: 0.95
0
25
50
75
100
125
150
175
200

7 Diagonalization

[ ]:
```
# TODO:
# Calculate PageRank via repeated multiplication of matrix using {alpha}

# TIPS:
# See np.linalg.eig for SVD decomposition

# Like Q1 on HW4, apply diagonalization with the focus on taking the diagonal
# matrix to a high power and then applying the other transforms

# Certain decompositions may carry imaginary numbers due to precision
# These numbers should die out with multiple applications of G
# So they should disappear as you take G^n for large n

# Calculate PageRank via diagonalization
def diagonalize(alpha):
  bigG = P * alpha + (1-alpha)/(N)
  eigval, S = np.linalg.eig(bigG)
  raiseme = np.diag(eigval)

  pi0 = np.ones(N) * 1/N
  i = 1
  old_pi = np.real((pi0 @ S @ np.linalg.matrix_power(raiseme, i) @ np.linalg.
 ↪inv(S)))
  old_pi = old_pi / sum(old_pi)
  while True:
    i += 1
    pi = np.real((pi0 @ S @ np.linalg.matrix_power(raiseme, i) @ np.linalg.
 ↪inv(S)))
    pi = pi / sum(pi)
    if np.max(np.abs(old_pi - pi)) < tolerance:
      break
    old_pi = pi
  return pi

# Note that as described, the PageRanks will be a factor of N off from random␣
 ↪surf approximation
diagonalize_pis = []
diagonalize_times = []
for alpha in alphas:
  pis = []
```

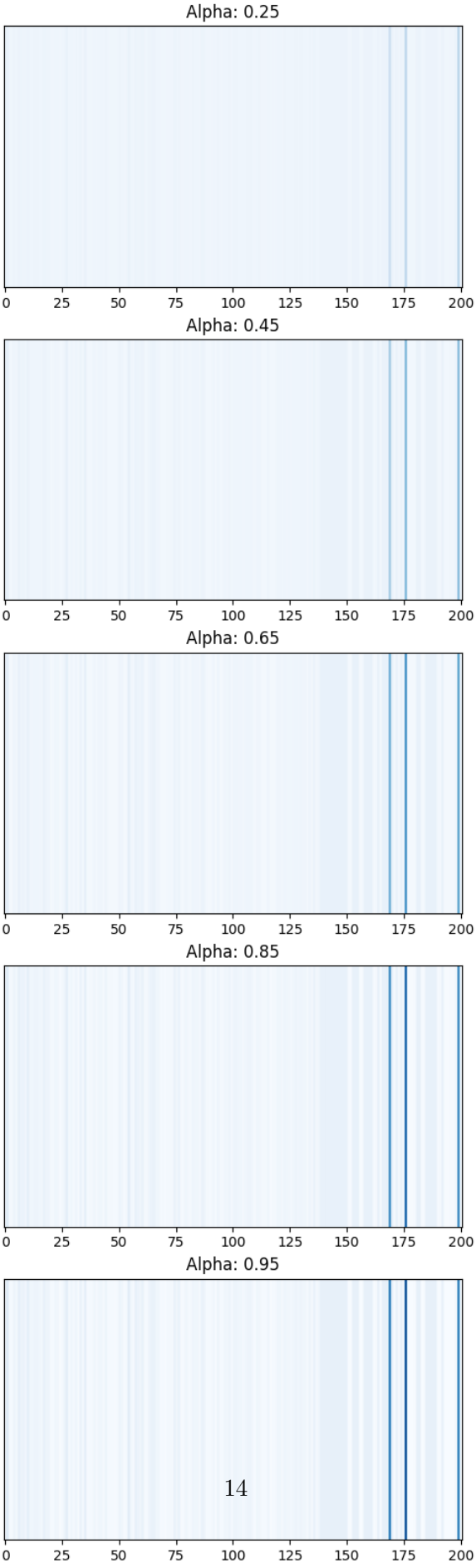
```
  # Even though result is the same, do multiple to average out runtime
  start = time.time()
  for _ in tqdm(range(iter)):
    pis.append(diagonalize(alpha))
  end = time.time()

  diagonalize_times.append((end - start) / iter)
  # Don't really need to have average here since we have deterministic results
  diagonalize_pis.append(np.average(pis, axis=0))
```

```
100%|      | 25/25 [00:09<00:00,  2.73it/s]
100%|      | 25/25 [00:14<00:00,  1.75it/s]
100%|      | 25/25 [00:09<00:00,  2.52it/s]
100%|      | 25/25 [00:18<00:00,  1.32it/s]
100%|      | 25/25 [00:12<00:00,  1.97it/s]
```

7.1 Diagonalization Visualizations

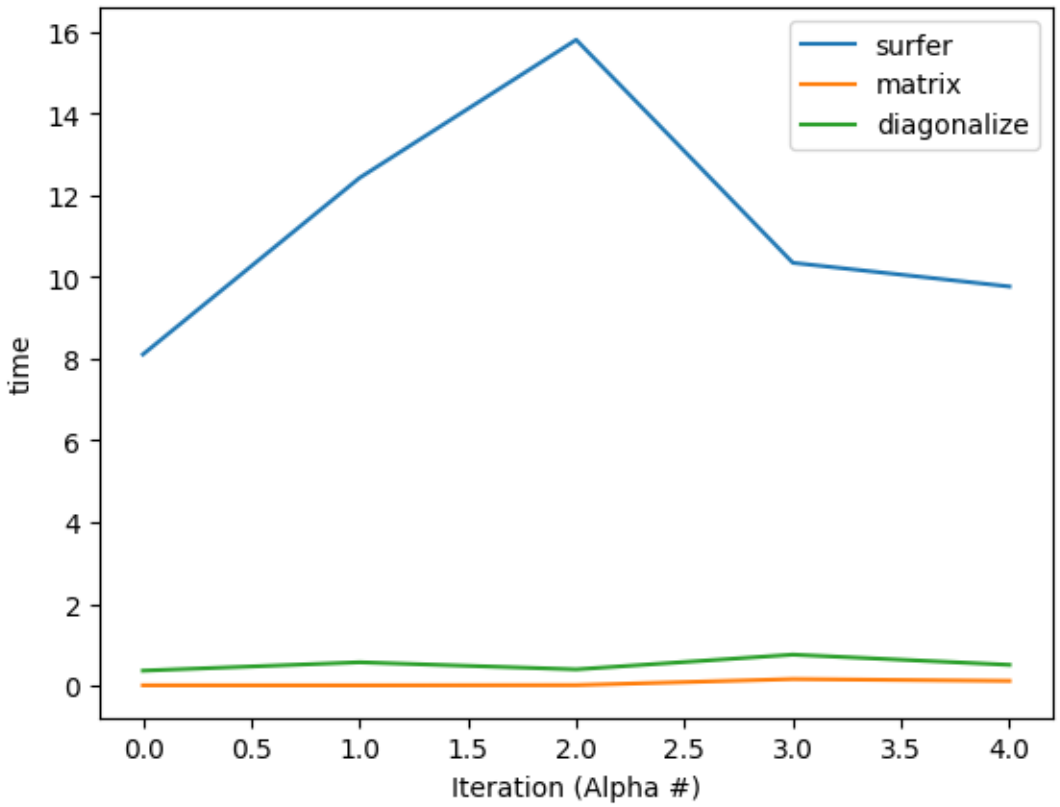
[ ]:
```
draw_heatmap(alphas, diagonalize_pis)
```

```
/var/folders/sg/4dp480wd1cjd288xvby34rpr0000gn/T/ipykernel_29900/2876918579.py:3
8: UserWarning: Matplotlib is currently using
module://matplotlib_inline.backend_inline, which is a non-GUI backend, so cannot
show the figure.
  fig.show()
```


Alpha: 0.25
0
25
50
75
100
125
150
175
200
Alpha: 0.45
0
25
50
75
100
125
150
175
200
Alpha: 0.65
0
25
50
75
100
125
150
175
200
Alpha: 0.85
0
25
50
75
100
125
150
175
200
Alpha: 0.95
0
25
50
75
100
125
150
175
200

[ ]:
```
plt.plot(surf_times, label="surfer")
plt.plot(matrix_times, label="matrix")
plt.plot(diagonalize_times, label="diagonalize")
plt.xlabel("Iteration (Alpha #)")
plt.ylabel("time")
plt.legend()
```

[ ]:
```
<matplotlib.legend.Legend at 0x11f4c8ac0>
```



It looks like the fastest method is matrix, but only a bit faster than diagonalize. Both deterministic methods are faster than the random surfer.

In terms of accuracy, both matrix and diagonalize come to very similar heatmaps, while the random surfer has some variance from both the other method while still having peaks in the same areas on the right side of the heatmap.