**Lab 03**

|  |  |
| --- | --- |
| Name: | 劉邦彬 |
| Student ID: | R11631033 |
| Total Score: |  |

**Note:**

Most of the explanations in this lab is optional. However, giving reasonable explanations to your answer or programs will earn you partial credits when your answer is incorrect.

1. **Multiple Choice (10 points, 5 points each question)**

|  |  |  |  |
| --- | --- | --- | --- |
| # | Answer | Explanation (Optional) | Score |
| 1 | (d) |  |  |
| 2 | (b) |  |  |

1. **A Piece of Cake (28 points, 4 points each question)**

|  |  |  |
| --- | --- | --- |
| # | Explanation (Optional) | Score |
| 1 | import numpy as np  # b-1  np.set\_printoptions(suppress=True, formatter={"float": "{:0.1f}".format})  b\_1 = np.arange(5, -5.2, -0.2)  print(b\_1)  # b-2  a = np.arange(0, 1.01, 0.01)  b\_2 = np.power(10, a)  print(b\_2)  # b-3  b\_3 = np.arange(1, 101).reshape(10, 10, order="F")  print(b\_3)  # b\_4  b\_4 = np.arange(1, 13).reshape(12, 1) \* np.arange(1, 13)  print(b\_4) |  |
| 2 | def E(X):      return X[::2, ::2]  def F(X):      return X[1:-1, 1:-1]  def G(X):      # The shape of return value will be (M, 2)      above = np.diag(X, k=1)      below = np.diag(X, k=-1)      return np.column\_stack((above, below)) |  |

1. **Programming Exercise (30 points, 6 points each question)**

|  |  |  |
| --- | --- | --- |
| # | Explanation (Optional) | Score |
| 1 | def swap\_rows(x, r1, r2):      # no return values needed      # no loops allowed      # do not declare new variables, manipulate x directly      x[[r1, r2], :] = x[[r2, r1], :]      pass |  |
| 2 | def most\_value(x):      # no loops allowed      return np.argmax(np.bincount(x)) |  |
| 3 | def top\_n(x, n):      tops = np.sort(x)[-n:]      # no loops allowed      return tops |  |
| 4 | def pythagorean(x):      if x.ndim == 1:          if np.size(x, 1) != 2:              raise ValueError          a, b = x          return np.sqrt(a\*\*2 + b\*\*2)      elif x.ndim == 2:          if np.size(x, 1) != 2:              raise ValueError          return np.sqrt(x[:, 0] \*\* 2 + x[:, 1] \*\* 2)  # .reshape(-1,1)      else:          raise ValueError |  |
| 5 | def replace\_me(v, a, b=None, c=None):      if b is None:          b = 0      if c is None:          c = b      return np.insert(v, np.where(np.isin(v, a))[0], [b, c]) |  |

1. **Markov Chains (32 points, 6 / 6 / 4 / 2 / 4 / 6 / 4)**

|  |  |  |
| --- | --- | --- |
| # | Explanation (Optional for questions 1, 2, and 5) | Score |
| 1 | def transition\_matrix(n):      # TODO\_D      P = np.zeros((n, n))      state = np.arange(n)      cond0 = state == 0      P[cond0, state[cond0] + 1] = 0.6      P[state[cond0] + 1, state[cond0] + 1] = 0.35      cond1 = (state < n / 2) & (state > 0)      P[cond1, state[cond1] + 1] = 0.6      P[cond1, state[cond1] - 1] = 0.35      P[cond1, 0] = 0.05      cond2 = (state >= n / 2) & (state < n - 1)      P[cond2, state[cond2] + 1] = 0.5      P[cond2, state[cond2] - 1] = 0.4      P[cond2, 0] = 0.1      P[0, 0] = 0.4      cond3 = state == (n - 1)      P[n - 1, n - 1] = 0.5      P[cond3, state[cond3] - 1] = 0.4      P[cond3, 0] = 0.1      return P |  |
| 2 | def propagate(x0, P, k):      xk = None      xk = x0 @ np.linalg.matrix\_power(P, k)      # TODO\_D2      return xk |  |
| 3 | Which state has the highest probability in step 8?  State0  What is the probability to reach the final state () in step 8?  0  # paste Lab04\_D3.png here    def main():      P = transition\_matrix(n=10)      # TODO\_D3, D4, D5,and D7      x0 = np.zeros(10)      x0[0] = 1      x8 = propagate(x0, P, k=8)      print(x8) |  |
| 4 | How many steps does it take for the probability of being in the final state (i=9) to be at least 1%?  Step 11  def main():      P = transition\_matrix(n=10)      # TODO\_D3, D4, D5,and D7      x0 = np.zeros(10)      x0[0] = 1      x11 = propagate(x0, P, k=11)      print(x11)      plot\_distribution(x11)  輸出:[0.18437179 0.17832461 0.14034025 0.16202376 0.09939172 0.1255268 0.04091187 0.04673376 0.01012824 0.0122472 ]此時state(i=9)>1% |  |
| 5 | Try initialing x0 with random numbers and keep , will the probability distribution be different? Give an explanation.      # TODO\_D5  x0 = np.random.rand(10)      x0 = x0 / np.sum(x0)      print(x0)  一開始不同，隨著propagate(x0, P, k=11)中的k增加應該會接近 |  |
| 6 | def create\_sample(s0, P, k):      trajectories = []      # # TODO\_D6      prev\_state = s0      for i in range(k+1):          trajectories.append(prev\_state)          current\_state = np.random.choice(np.arange(10), p=P[prev\_state])          prev\_state = current\_state      return trajectories |  |
| 7 | # TODO\_D7      last\_steps = []      for i in range(1000):          sample = create\_sample(s0=0, P=P, k=8)          state\_last= sample[-1]          last\_steps.append(state\_last)      plot\_histogram(last\_steps)  state has the highest probability in step 8?  State0  What do you observe from the two figures? Does the result meet your expectation?  看起來很像 , Yes。  # paste Lab04\_D7.png here |  |

**03\_b**

import numpy as np

from random import randrange

# b-1

np.set\_printoptions(suppress=True) #, formatter={"float": "{:0.1f}".format}

b\_1 = np.arange(5, -5.2, -0.2)

print(b\_1)

# b-2

a = np.arange(0, 1.01, 0.01)

b\_2 = np.power(10, a)

print(b\_2)

# b-3

b\_3 = np.arange(1, 101).reshape(10, 10, order="F")

print(b\_3)

# b\_4

b\_4 = np.arange(1, 13).reshape(12, 1) \* np.arange(1, 13)

print(b\_4)

A = None

B = None

C = None

D = None

def E(X):

    return X[::2, ::2]

def F(X):

    return X[1:-1, 1:-1]

def G(X):

    # The shape of return value will be (M, 2)

    above = np.diag(X, k=1)

    below = np.diag(X, k=-1)

    return np.column\_stack((above, below))

# Do NOT modifiy the main function

def main():

    print('A: \n', A, '\n')

    print('B: \n', B, '\n')

    print('C: \n', C, '\n')

    print('D: \n', D, '\n')

    M = randrange(3, 8)

    X = np.random.randint(10, size=(M, M))

    print('X: \n', X, '\n')

    print('E: \n', E(X), '\n')

    print('F: \n', F(X), '\n')

    print('G: \n', G(X), '\n')

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**03\_C**

import numpy as np

def swap\_rows(x, r1, r2):

    # no return values needed

    # no loops allowed

    # do not declare new variables, manipulate x directly

    x[[r1, r2], :] = x[[r2, r1], :]

    pass

def most\_value(x):

    # no loops allowed

    return np.argmax(np.bincount(x))

def top\_n(x, n):

    tops = np.sort(x)[-n:]

    # no loops allowed

    return tops

def pythagorean(x):

    if x.ndim == 1:

        if np.size(x, 1) != 2:

            raise ValueError

        a, b = x

        return np.sqrt(a\*\*2 + b\*\*2)

    elif x.ndim == 2:

        if np.size(x, 1) != 2:

            raise ValueError

        return np.sqrt(x[:, 0] \*\* 2 + x[:, 1] \*\* 2)  # .reshape(-1,1)

    else:

        raise ValueError

def replace\_me(v, a, b=None, c=None):

    if b is None:

        b = 0

    if c is None:

        c = b

    return np.insert(v, np.where(np.isin(v, a))[0], [b, c])

# You may test your function here

def main():

    # Lab04\_C1 Swap rows

    print("Lab04\_C1 Swap rows:")

    x1 = np.arange(9).reshape(3, 3)

    swap\_rows(x1, 0, 1)

    print(x1, "\n")

    # Lab04\_C2 Find most frequent value

    print("Lab04\_C2 Find most frequent value:")

    x2 = np.array([1, 2, 2, 1, 3, 2, 4, 1, 2])

    print("The most frequent value is: ", most\_value(x2), "\n")

    # Lab04\_C3 top n

    print("Lab04\_C3 Top n:")

    x3 = np.array([1, 0, 3, 5, 7, 3, 2, 8, 9, 2, 8])

    print("The 3 largest values are: ", top\_n(x3, n=3), "\n")

    # Lab04\_C4 pythagorean

    print("Lab04\_C4 pythagorean:")

    x4 = np.array([[3, 4], [5, 12]])

    print(pythagorean(x4))

    try:

        pythagorean(np.array([12]))

        print("If you see this line, you may not check the input array", "\n")

    except:

        print("\n")

    # Lab04\_C5 replace\_me

    print("Lab04\_C5 replace\_me:")

    x5 = np.array([1, 2, 3])

    print(replace\_me(x5, 2, 4, 5), "\n")

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**03\_D**

import numpy as np

import matplotlib.pyplot as plt

def transition\_matrix(n):

    # TODO\_D1

    P = np.zeros((n, n))

    state = np.arange(n)

    cond0 = state == 0

    P[cond0, state[cond0] + 1] = 0.6

    cond1 = (state < n / 2) & (state > 0)

    P[cond1, state[cond1] + 1] = 0.6

    P[cond1, state[cond1] - 1] = 0.35

    P[cond1, 0] = 0.05

    P[state[cond0] + 1, cond0] = 0.4

    cond2 = (state >= n / 2) & (state < n - 1)

    P[cond2, state[cond2] + 1] = 0.5

    P[cond2, state[cond2] - 1] = 0.4

    P[cond2, 0] = 0.1

    P[0, 0] = 0.4

    cond3 = state == (n - 1)

    P[n - 1, n - 1] = 0.5

    P[cond3, state[cond3] - 1] = 0.4

    P[cond3, 0] = 0.1

    return P

def propagate(x0, P, k):

    xk = None

    xk = x0 @ np.linalg.matrix\_power(P, k)

    # TODO\_D2

    return xk

def create\_sample(s0, P, k):

    trajectories = []

    # # TODO\_D6

    prev\_state = s0

    for i in range(k + 1):

        trajectories.append(prev\_state)

        current\_state = np.random.choice(np.arange(10), p=P[prev\_state])

        prev\_state = current\_state

    return trajectories

def plot\_distribution(x):

    plt.plot(x)

    plt.xticks(np.arange(0, len(x), step=1))

    plt.ylim(0, max(x) + 0.1)

    plt.xlabel("State (i)")

    plt.ylabel("Probability")

    plt.title("Probability Distribution")

    plt.savefig("Lab04\_D3.png", dpi=150)

def plot\_histogram(x):

    plt.hist(x, bins=max(x) + 1, range=(-0.5, max(x) + 0.5))

    plt.xticks(np.arange(0, max(x) + 1, step=1))

    plt.xlabel("State (i)")

    plt.ylabel("Number of smaple")

    plt.title("State Histogram")

    plt.savefig("Lab04\_D7.png", dpi=150)

def main():

    P = transition\_matrix(n=10)

    # TODO\_D3

    x0 = np.zeros(10)

    x0[0] = 1

    x8 = propagate(x0, P, k=8)

    print(x8)

    plot\_distribution(x8)

    # TODO\_D4

    x11 = propagate(x0, P, k=11)

    print(x11)

    plot\_distribution(x11)

    # TODO\_D5

    x0 = np.random.rand(10)

    x0 = x0 / np.sum(x0)

    print(x0)

    # TODO\_D6

    a = create\_sample(s0=0, P=P, k=20)

    print(a)

    # TODO\_D7

    last\_steps = []

    for i in range(1000):

        sample = create\_sample(s0=0, P=P, k=8)

        state\_last = sample[-1]

        last\_steps.append(state\_last)

    plot\_histogram(last\_steps)

if \_\_name\_\_ == "\_\_main\_\_":

    main()