```
#!/usr/bin/env python3

# Imports
from os import stat_result
import numpy as np
import matplotlib.pyplot as plt
from numpy.core.numeric import array_equal
from numpy.core.shape_base import atleast_1d

# Cloud Generated Information
# Matrix A:
# copyable: {{0.91, 0, 0.06}, {0.09, 0.93, 0}, {0, 0.07, 0.94}}
#
# formatted:
# │  0.91  0.00  0.06  │
# │  0.09  0.93  0.00  │
# │  0.00  0.07  0.94  │
#
# n = 21

# 1.
A = np.array([[0.91, 0, 0.06], [0.09, 0.93, 0], [0, 0.07, 0.94]])

# Compute power trick dot products
x2 = A @ A
x4 = x2 @ x2
x8 = x4 @ x4
x16 = x8 @ x8
x32 = x16 @ x16
x64 = x32 @ x32
x_diff = sum(sum(abs(x2 - A)))
x2_diff = sum(sum(abs(x4-x2)))
x4_diff = sum(sum(abs(x8-x4)))
x8_diff = sum(sum(abs(x16-x8)))
x16_diff = sum(sum(abs(x32-x16)))
x32_diff = sum(sum(abs(x64-x32)))
# print("x2 = {}\nx4 = {}\nx8 = {}\nx16 = {}\nx32 = {}\nx64 = {}".
#   format(x2, x4, x8, x16, x32, x64))
# print("x_diff = {}\nx2_diff = {}\nx4_diff = {}\nx8_diff = {}\nx16_diff = {}\nx32_diff = {
}".
#   format(x_diff, x2_diff, x4_diff, x8_diff, x16_diff, x32_diff))
# Output:
# x2 = [[0.8281 0.0042 0.111 ]
#  [0.1656 0.8649 0.0054]
#  [0.0063 0.1309 0.8836]]
# x4 = [[0.68714443 0.0216405  0.19002138]
#  [0.28039482 0.74945439 0.0278235 ]
#  [0.03246075 0.22890511 0.78215512]]
# x8 = [[0.48440359 0.07458558 0.27980044]
#  [0.40371804 0.57411871 0.09589575]
#  [0.11187837 0.35129571 0.62430381]]
# x16 = [[0.296062   0.1772432  0.31736926]
#  [0.43807321 0.3934116  0.22788411]
#  [0.26586479 0.4293452  0.45474663]]
# x32 = [[0.24967552 0.25846547 0.27867449]
#  [0.36262627 0.33025913 0.33231275]
#  [0.38769821 0.41127539 0.38901276]]
# x64 = [[0.26410584 0.26450504 0.2638775 ]
#  [0.3391366  0.33946952 0.34007791]
#  [0.39675757 0.39602543 0.39604459]]
#
# To calculate the closeness of these matrices, I computed the sum of their
# absolute elementwise-difference. The output shows that initially, the matrices
# increase, but thafter x16 the elements begin to become much closer, with the
# element-wise total difference between x64 and x32 only being 0.107.
# Output:
# x_diff = 0.40679999999999994
# x2_diff = 0.7156921200000002
# x4_diff = 1.0718556680104574
# x8_diff = 1.0772117528973573
# x16_diff = 0.6149686760406601
# x32_diff = 0.10707324031657456

# 2.
# n = 21
x21 = x16 @ x4 @ A
# print("x21 = {}".format(x21))
# Output:
# x21 = [[0.25879821 0.21871728 0.30666258]
#  [0.41312588 0.35060396 0.28120793]
#  [0.32807591 0.43067877 0.41212949]]

# 3.
p0 = np.array([0.5, 0, 0])

# 3. a)
p1 = A @ p0
p2 = x2 @ p0
p3 = x2 @ A @ p0
# print("p1 = {}\np2 = {}\np3 = {}".format(p1, p2, p3))
# Output:
# p1 = [0.455 0.045 0.   ]
# p2 = [0.41405 0.0828  0.00315]
# p3 = [0.3769745 0.1142685 0.008757 ]
time_axis = np.linspace(0, 3, 4)

artemis = np.array([p0[0], p1[0], p2[0], p3[0]])
luna = np.array([p0[1], p1[1], p2[1], p3[1]])
selene = np.array([p0[2], p1[2], p2[2], p3[2]])
plt.title("Timeseries Plot")
plt.grid()
plt.xticks([0, 1, 2, 3])
plt.xlabel("Day")
plt.ylabel("Probability of Disease")
plt.plot(time_axis, artemis, color='r')
plt.plot(time_axis, luna, color='g')
plt.plot(time_axis, selene, color='b')
plt.savefig("images/timeseries.png")

# 3. b)
# day n = 21
p21 = x21 @ p0
# print("p21 = {}".format(p21))
# Output:
# p21 = [0.1293991  0.20656294 0.16403796]

# 3. c)
eigvals, eigvecs = np.linalg.eig(A)
# print("eigenvalues: \n{}".format(eigvals))
# print("largest eigenvalue: {}".format(max(eigvals)))
# Output:
# [0.89+0.06164414j, 0.89-0.06164414j, 1. 0.j]

# print("eigenvectors: \n{}".format(eigvecs))
# Output:
# [-0.28426762+0.43808583j, -0.28426762-0.43808583j, 0.4516129 +0.j],
#       [ 0.63960215+0.j, 0.63960215-0.j, 0.58064516+0.j],
#       [-0.35533453-0.43808583j, -0.35533453+0.43808583j, 0.67741935+0.j]

# The third column is the dominant eigenvector because its eigenvalue is 1.
dom = np.real(eigvecs[:,2])
# print(dom)
# Dominant eigenvector [0.4516129  0.58064516 0.67741935]
dom_probability = dom / sum(dom)
# print(dom_probability)
# Dominant eigenvector as probability vec [0.26415094 0.33962264 0.39622642]
# 3. d)
# Looking at A, B, and C we can observe that C yields the true steady state
# while part A displays the initial values and trends, and B shows the state at
# 21 days. They are similar in the way that part A shows a trend towards C, and
# part B shows a much closer approach to C. This shows that as more days pass, the
# probability vector is stabilizing and approaching the steady state given by C. They are
# different in that A begins at the initial state which is far from the states given by B o
r
# C. None of these are truly equal because they all yield different values, although trendi
ng
# similarly.

# 3. e)
# print("eigenvectors: \n{}".format(eigvecs))
# Output:
# [0.23615094 0.23615094 0.23615094]
# [0.33962264 0.33962264 0.33962264]
# [0.39622642 0.39622642 0.39622642]

# 3. f)
# print(x64)
#  [0.26410584 0.26450504 0.2638775 ]
#  [0.3391366  0.33946952 0.34007791]
#  [0.39675757 0.39602543 0.39604459]
# These matrices are quite similar at this point, given we have gone through 64
# applications of the transition matrix. It is clear that the elements of the x64
# matrix are approaching those of the matrix composed of the dominant eigenvector,
# the steady state. They are different in that the values are not yet equal and they
# still differ by a notable amount, up to ~0.3

# 4.
# My long term prediction is that the values yielded by the applications of the
# transition matrix will converge to the values in the matrix from part 3e).
# Since the columns of the matrix from 3e) are given by the dominant eigenvector
# whose eigenvalue is 1, this is the steady state, and repeated applcations of
# the matrix A to the probability vector yielded by x64 dot v will continue to
# approach, but never exactly reach the steady state.
```