```python
# -*- coding: utf-8 -*-
"""
Created on Thu Oct 15 16:21:37 2020

@author: Donald
"""

# -*- coding: utf-8 -*-
"""
Created on Thu Oct  1 10:30:03 2020

@author: Donald
"""
import numpy as np
import matplotlib.pyplot as plt

def analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, titile_2, title_3):
    print('Case: ', case)
    # define initial precursor
    if S != 0:
        n_init = -(S * MGT)/rho_init
    else:
        n_init = 1
    c_init = beta_/(lambda_ * MGT)
    print('Inital value n(0): ', n_init)
    print('Inital value c(0): ', c_init)

    # Write down quadratic and give s1, s2
    disc = (MGT*lambda_+beta_-rho)**2 + 4*rho*lambda_*MGT
```

```
s1 = (rho-beta_-lambda_*MGT + np.sqrt(disc)) / (2*MGT)
s2 = (rho-beta_-lambda_*MGT - np.sqrt(disc)) / (2*MGT)
print('rooot s1: ', s1)
print('rooot s2: ', s2)

# Check if particular solution is needed and if so solve for it
if S == 0:
    n_part = 0
    c_part = 0
    print('Neutron population particular solution: ', n_part)
if S != 0:
    # check if final reactivity is 0 if not constant solutions if so linear solitions
    if rho != 0:
        n_part = -S * MGT / rho
        c_part = beta_ * (lambda_ * MGT) * n_part
        print('Neutron population particular solution: ', n_part)
    else:
        a = (S * lambda_) / ((-beta_ / MGT) + lambda_)
        b = a - S
        d = b / -lambda_
        print('Neutron population particular solution = ', a, 't')

# solve for the coefficients A1 and A2
mat = np.array([[1,1],[lambda_/(lambda_+s1),lambda_/(lambda_+s2)]])
rhs = np.ones(2)
coef = np.linalg.solve(mat, rhs)
# difine the A_1 and A_2 coeffeicints
A_1 = coef[0]
A_2 = coef[1]
```

```
# difine the B_1 and B_2 coeffeicints
B_1 = beta_/MGT * (coef[0]/(lambda_*s1))
B_2 = beta_/MGT * (coef[1]/(lambda_*s2))
print('Amplitude A1 = ', A_1)
print('Amplitude A2 = ', A_2)
print('Amplitude B1 = ', B_1)
print('Amplitude B2 = ', B_2)

# Caluculate the initial promt jump
P_J = (rho_init - beta_) / (rho - beta_)
print('Value of prompt jump = ', P_J)

# Calcualte n(5) and n(30)
if S != 0:
    if rho != 0:
        n_5 = A_1*np.exp(s1*5) + A_2*np.exp(s2*5) + n_part
        n_30 = A_1*np.exp(s1*30) + A_2*np.exp(s2*30) + n_part
        n_500 = A_1*np.exp(s1*500) + A_2*np.exp(s2*500) + n_part
    else:
        n_5 =  A_1*np.exp(s1*5) + A_2*np.exp(s2*5) + a * 5
        n_30 = A_1*np.exp(s1*30) + A_2*np.exp(s2*30) + a * 30
        n_500 =  A_1*np.exp(s1*500) + A_2*np.exp(s2*500) + a * 500
else:
    n_5 =  A_1*np.exp(s1*5) + A_2*np.exp(s2*5)
    n_30 = A_1*np.exp(s1*30) + A_2*np.exp(s2*30)
    n_500 =  A_1*np.exp(s1*500) + A_2*np.exp(s2*500)

print('n(5 sec) = ', n_5)
print('n(30 sec) = ', n_30)
```

```
if five == 'true':
    print('n(500 sec) = ', n_500)

# create function for ploting
if S != 0:
    if rho != 0:
        n = lambda time: A_1*np.exp(s1*time) + A_2*np.exp(s2*time) + n_part
        c = lambda time: B_1*np.exp(s1*time) + B_2*np.exp(s2*time) + c_part
    else:
        n = lambda time: A_1*np.exp(s1*time) + A_2*np.exp(s2*time) + a * time
        c = lambda time: B_1*np.exp(s1*time) + B_2*np.exp(s2*time) + b * time + d
else:
    n = lambda time: A_1*np.exp(s1*time) + A_2*np.exp(s2*time)
    c = lambda time: B_1*np.exp(s1*time) + B_2*np.exp(s2*time)

t = np.linspace(0,30,1000)
plt.figure(dpi=250)
plt.plot(t,n(t)/n(0),label='n')
plt.plot(t,c(t)/c(0),label='c')
plt.xlabel('time [s]')
plt.ylabel('Neutron population')
plt.title(title_1)
plt.grid()
plt.legend()
plt.show()

if log == 'true':
    t = np.linspace(0,500,10000)
    plt.figure(dpi=250)
```

```
        plt.plot(t,n(t)/n(0),label='n')
        plt.plot(t,c(t)/c(0),label='c')
        plt.xlabel('time [s]')
        plt.ylabel('Neutron population')
        plt.title(title_2)
        plt.grid()
        plt.legend()
        plt.show()

        t = np.linspace(0,500,10000)
        plt.figure(dpi=250)
        plt.semilogy(t,n(t)/n(0),label='n')
        plt.semilogy(t,c(t)/c(0),label='c')
        plt.xlabel('time [s]')
        plt.ylabel('Neutron population')
        plt.title(title_3)
        plt.grid()
        plt.legend()
        plt.show()

def numericalSolution(S, lambda_, beta_, MGT, rho_init, rho, title):
    # initial values for n and c
    if S !=0:
        X0 = np.array([1., beta_/(lambda_*MGT)])*(-S * MGT)/MGT
    else:
        X0 = np.array([1., beta_/(lambda_*MGT)])
    # end time for simulation
    Tend = 30
    # number of time steps
```

```
n_steps = 30
# time step size
dt = Tend / n_steps


# identity matrix
I = np.eye(2)
A = np.array([[(rho-beta_)/MGT, lambda_], [beta_/MGT, -lambda_]])
# final form of the linear system matrix
M = I - dt*A

# storage place for plotting solution later
sol = np.zeros((2,n_steps+1))
sol[:,0] = X0

# loop through time steps
for i in range(n_steps):
    # end time steps values
    X1 = np.linalg.solve(M,X0)
    # store
    sol[:,i+1] = X1
    # X1 becomes initial value for next time step
    X0 = np.copy(X1)

# initial values for n and c
X0_1 = np.array([1., beta_/(lambda_*MGT)])
# end time for simulation
Tend_1 = 30
# number of time steps
```

```
n_steps_1 = 3000
# time step size
dt_1 = Tend_1 / n_steps_1



# identity matrix
I_1 = np.eye(2)
A_1 = np.array([[(rho-beta_)/MGT, lambda_], [beta_/MGT, -lambda_]])
# final form of the linear system matrix
M_1 = I_1 - dt_1*A_1

# storage place for plotting solution later
sol_1 = np.zeros((2,n_steps_1+1))
sol_1[:,0] = X0_1

# loop through time steps
for i in range(n_steps_1):
    # end time steps values
    X1_1 = np.linalg.solve(M_1,X0_1)
    # store
    sol_1[:,i+1] = X1_1
    # X1 becomes initial value for next time step
    X0_1 = np.copy(X1_1)

time_30 = np.linspace(0,Tend,n_steps+1)
time_3000 = np.linspace(0,Tend,n_steps_1+1)
plt.figure(2, dpi=250)
plt.plot(time_30,sol[0,:]/sol[0,0],label='n dt=1')
plt.plot(time_30,sol[1,:]/sol[1,0],label='cd t=1')
```

```
    plt.plot(time_3000,sol_1[0,:]/sol_1[0,0],label='n dt=0.01')
    plt.plot(time_3000,sol_1[1,:]/sol_1[1,0],label='cd t=0.01')
    plt.xlabel('time [s]')
    plt.ylabel('Neutron population')
    plt.title(title)
    plt.grid()
    plt.legend()
    plt.show()

def numericalSolutionComparison():
    # decay constant for DNP groups, 1/s
    lambda_i = np.array([0.0124 , 0.0305  , 0.111   , 0.301   , 1.14    , 3.01   ])
    # delayed neutron fractions per DNP froups
    beta_i   = np.array([0.00021, 0.00142 , 0.00127 , 0.00258 , 0.00075 , 0.00027])
    beta_tot = sum(beta_i)
    n_dnp = 6
    MGT = 2e-4
    rho = beta_tot/11
    # initial values for n and c
    X0 = np.array([1., beta_i[0]/(lambda_i[0]*MGT), beta_i[1]/(lambda_i[1]*MGT)\
                , beta_i[2]/(lambda_i[2]*MGT), beta_i[3]/(lambda_i[3]*MGT)\
                , beta_i[4]/(lambda_i[4]*MGT), beta_i[5]/(lambda_i[5]*MGT)])
    # end time for simulation
    Tend = 30
    # number of time steps
    n_steps = 30
    # time step size
    dt = Tend / n_steps
```

```
# identity matrix
I = np.eye(n_dnp+1)
A = np.zeros((n_dnp+1,n_dnp+1))
A[0,0] = (rho-beta_tot)/MGT
for i in range(n_dnp):
   A[0,i+1] = lambda_i[i]
   A[i+1,0] = beta_i[i]/MGT
   A[i+1,i+1] = -lambda_i[i]
# final form of the linear system matrix
M = I - dt*A

# storage place for plotting solution later
sol = np.zeros((7,n_steps+1))
sol[:,0] = X0

# loop through time steps
for i in range(n_steps):
      # end time steps values
      X1 = np.linalg.solve(M,X0)
      # store
      sol[:,i+1] = X1
      # X1 becomes initial value for next time step
      X0 = np.copy(X1)

# Make plot with 1 neutron group
beta_ = 650e-5
lambda_ = 0.3
# initial values for n and c
```

```
X0_1 = np.array([1., beta_/(lambda_*MGT)])

# end time for simulation

Tend_1 = 30

# number of time steps

n_steps_1 = 3000

# time step size

dt_1 = Tend_1 / n_steps_1



# identity matrix

I_1 = np.eye(2)

A_1 = np.array([[(rho-beta_)/MGT, lambda_], [beta_/MGT, -lambda_]])

# final form of the linear system matrix

M_1 = I_1 - dt_1*A_1


# storage place for plotting solution later

sol_1 = np.zeros((2,n_steps_1+1))

sol_1[:,0] = X0_1


# loop through time steps

for i in range(n_steps_1):

    # end time steps values

    X1_1 = np.linalg.solve(M_1,X0_1)

    # store

    sol_1[:,i+1] = X1_1

    # X1 becomes initial value for next time step

    X0_1 = np.copy(X1_1)


time_30 = np.linspace(0,Tend,n_steps+1)
```

```
    time_3000 = np.linspace(0,Tend,n_steps_1+1)
    plt.figure(2, dpi=250)
    plt.plot(time_30,sol[0,:]/sol[0,0],label='n with 6 groups')
    plt.plot(time_30,sol[1,:]/sol[1,0],label='c with 6 groups')
    plt.plot(time_3000,sol_1[0,:]/sol_1[0,0],label='n with 1 groups')
    plt.plot(time_3000,sol_1[1,:]/sol_1[1,0],label='c with 1 groups')
    plt.xlabel('time [s]')
    plt.ylabel('Neutron population')
    plt.title('Comparing 6 and 1 neutron group solution')
    plt.grid()
    plt.legend()
    plt.show()

print('Question 1, Part 1:')
case = 1
S = 0
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = 0
rho = beta_/11
log = 'false'
five = 'false'
title_1 = 'Case 1 t [0,30]'
title_2 = ''
title_3 = ''
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)

print('\n=========================================================')
```

```
print('Question 1, Part 2:')
case = 2
S = 0
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = 0
rho = 0
log = 'false'
five = 'false'
title_1 = 'Case 2 t [0,30]'
title_2 = ''
title_3 = ''
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)
print('\n==========================================================')
print('Question 1, Part 3:')
case = 3
S = 0
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = 0
rho = -beta_/9
log = 'true'
five = 'false'
title_1 = 'Case 3 t [0,30]'
title_2 = 'Case 3 t [0,500]'
title_3 = 'Case 3 t [0,500] on semilog scale in y'
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)
```

```
print('\n============================================================')
print('Question 1, Part 4:')
case = 4
S = 2
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = -beta_/9
rho = beta_/11
log = 'false'
five = 'false'
title_1 = 'Case 4 t [0,30]'
title_2 = ''
title_3 = ''
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)

print('\n============================================================')
print('Question 4, Part 5:')
case = 5
S = 2
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = -beta_/9
rho = 0
log = 'false'
five = 'false'
title_1 = 'Case 5 t [0,30]'
```

```
title_2 = ''
title_3 = ''
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)

print('\n===============================================')
print('Question 1, Part 6:')
case = 6
S = 2
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = -beta_/9
rho = -beta_/3
log = 'true'
five = 'true'
title_1 = 'Case 6 t [0,30]'
title_2 = 'Case 6 t [0,500]'
title_3 = 'Case 6 t [0,500] on semilog scale in y'
analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)

print('\n===============================================')
print('Question 1, Part 7:')
case = 7
S = 2
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = -beta_/9
rho = -beta_/18
```

```
log = 'true'

five = 'true'

title_1 = 'Case 7 t [0,30]'

title_2 = 'Case 7 t [0,500]'

title_3 = 'Case 7 t [0,500] on semilog scale in y'

analyticalSolution(case, S, lambda_, beta_, MGT, rho_init, rho, log, five, title_1, title_2,  title_3)


print('\n========================================================')

print('Question 2 case 1')

S = 0

lambda_ = 0.3

MGT = 2e-4

beta_ = 650e-5

rho_init = 0

rho = beta_/11

title = 'Case 1'

numericalSolution(S, lambda_, beta_, MGT, rho_init, rho, title)


print('\n========================================================')

print('Question 2 case 3')

S = 0

lambda_ = 0.3

MGT = 2e-4

beta_ = 650e-5

rho_init = 0

rho = -beta_/9

title = 'Case 3'

numericalSolution(S, lambda_, beta_, MGT, rho_init, rho, title)
```

```
print('\n============================================================')
print('Question 2 case 6')
S = 2
lambda_ = 0.3
MGT = 2e-4
beta_ = 650e-5
rho_init = -beta_/9
rho = -beta_/3
title = 'Case 6'
numericalSolution(S, lambda_, beta_, MGT, rho_init, rho, title)

print('\n============================================================')
print('Question 3')
numericalSolutionComparison()
```