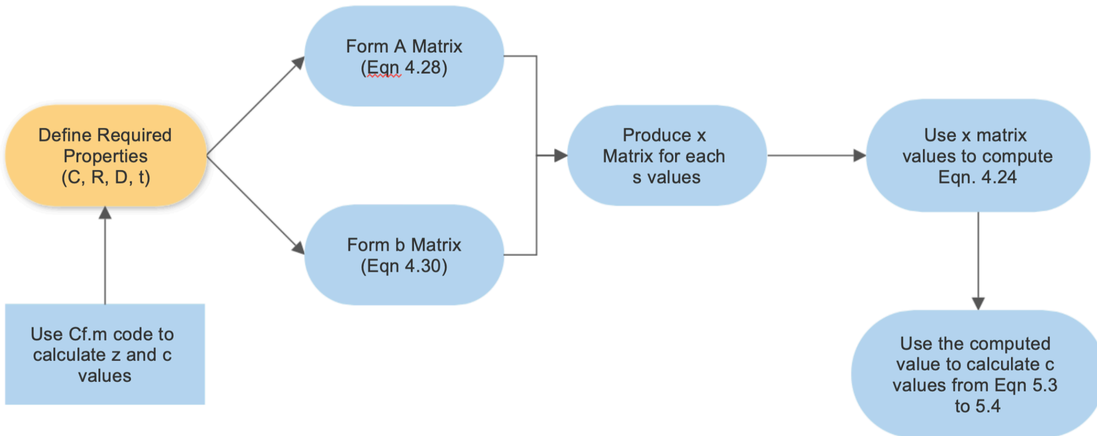
Multi-Layer Sphere Diffusion Code Report

Amirreza Mottafegh

May 28, 2021

1- Flowchart



2- Code Analysis

```python

class diffusion():

    def __init__(self,R,D,C,t,n,De,Ce):
        self.X=[]
        self.R=R
        self.D=D
        self.C=C
        self.t=t
        self.n=n     #Number of Layers
        self.De=De
        self.A=np.zeros((self.n,self.n),dtype=np.complex_)
        self.b=np.zeros((self.n),dtype=np.complex_)
        self.G=[]
        self.Ce=Ce


        self.R1=self.R[-1]
        self.call=np.copy(self.C)
        self.call=np.append(self.call,self.Ce)
        self.dall=np.copy(self.D)
        self.dall=np.append(self.dall,self.De)
        self.Dmax=np.max(self.dall)
        self.Cmax=np.max(self.call)
        self.C=self.C/self.Cmax
        self.R=np.asarray(self.R)/self.R1
        self.D=np.asarray(self.D)/self.Dmax
        self.De=self.De/self.Dmax
        self.Ce=self.Ce/self.Cmax
        z = spio.loadmat('z.mat', squeeze_me=True)
        c = spio.loadmat('c.mat', squeeze_me=True)
        self.p=math.inf
        z = z['z']
        c= c['c']
        self.c = c[0:-1:2]
        self.s= np.asarray(z[0:-1:2])/self.t

```

In the part above, variables are defined and introduced to the Diffusion class. Through lines 37 to 43, zk and c values from cf.m code, are imported. These values are used to compute inverse Laplacian.

```python
    def ai0(self,i,r,s):
        mew = np.sqrt(np.abs(s/self.D[i]))
        SS1= np.power(self.R[i-1],2)*(mew*self.R[i]*math.cosh(mew*(r-self.R[i]))+math.sinh(mew*(r-self.R[i])))
        dRi = self.R[i]-self.R[i-1]
        SS2 = r*(self.D[i]*mew*dRi*math.cosh(mew*dRi)+(s*self.R[i]*self.R[i-1]-self.D[i])*math.sinh(mew*dRi))
        return SS1/SS2

    def ai1(self,i,r,s):
        mew = np.sqrt(np.abs(s/self.D[i]))
        SS1= np.power(self.R[i],2)*(mew*self.R[i-1]*math.cosh(mew*(r-self.R[i-1]))+math.sinh(mew*(r-self.R[i-1])))
        dRi = self.R[i]-self.R[i-1]
        SS2 = r*(self.D[i]*mew*dRi*math.cosh(mew*dRi)+(s*self.R[i]*self.R[i-1]-self.D[i])*math.sinh(mew*dRi))
        return -SS1/SS2

    def a01(    ,r,s):
        mew = np.sqrt(np.abs(s/self.D[0]))
        SS1= np.power(self.R[0],2)*math.sinh(mew*r)
        SS2 = r*self.D[0]*(math.cosh(mew*self.R[0])*mew*self.R[0]-math.sinh(mew*self.R[0]))
        return -SS1/SS2

    def ae0(self,r,s):
        mew = np.sqrt(np.abs(s/self.De))
        SS1= np.power(self.R[-1],2)*np.exp(-mew*(r-self.R[-1]))
        SS2 = r*self.De*(1+mew*self.R[-1])
        return SS1/SS2
```

In this section, a functions are defined. These functions are as Eqns. 4.13, 4.18, 4.19, and 4.20. These functions are used to compute concentration distribution through the sphere layers.

```python
    def former(self,n,s):
        for i in range(self.n-1):
            self.A[i,i]=self.ai1(i,self.R[i],s)-self.ai0(i+1,self.R[i],s)
            self.A[i+1,i]=self.ai0(i+1,self.R[i+1],s)
            self.A[i,i+1]=-self.ai1(i+1,self.R[i],s)
            self.b[i]=(self.C[i+1]-self.C[i])/s
        self.A[self.n-1,self.n-1]=self.ai1(self.n-1,self.R[self.n-1],s)-self.ae0(self.R[self.n-1],s)
        self.A[0,0]=self.a01(self.R[0],s)-self.ai0(1,self.R[0],s)
        self.b[-1]=(self.Ce-self.C[-1])/s
        G= np.matmul(np.linalg.inv(self.A),self.b)
        return G
```

Former function, is a function which forms A, b and x matrixes. X matrix is the matrix where g values are in. This function uses a functions to calculate concentration respect to radius.

```
    def c_0(self,r):
        res=[]
        for k in range(len(self.s)):
            G=self.former(self.n,self.s[k])[0]
            res.append(self.c[k]*self.a01(r,self.s[k])*G/self.t)
        return (-2*np.sum(res).real)+self.C[0]

    def c_i(self,i,r):
        res0,res1=[],[]
        for j in range(len(self.s)):
            G0=self.former(self.n,self.s[j])[i-1]
            G1=self.former(self.n,self.s[j])[i]
            res0.append(self.c[j]*self.ai0(i,r,self.s[j])*G0/self.t)
            res1.append(self.c[j]*self.ai1(i,r,self.s[j])*G1/self.t)
        return (-2*np.sum(res0).real)+(-2*np.sum(res1).real)+self.C[i]


    def c_e(self,r):
        res=[]
        for i in      (len(self.s)):
            G=self.former(self.n,self.s[i])[-1]
            res.append(self.c[i]*self.ae0(r,self.s[i])*G/self.t)
        return (-2*np.sum(res).real)+self.Ce
```

These three functions are as functions Eqn. 5.3, 5.4, and 5.5 from the paper. These functions are the final functions to calculate concentration through the sphere.

```
111
112 R=[1.5e-3,1.7e-3,2e-3]    #radius of layers [mm] , [R0,R1,R2,..]
113 C=[1,1,1]      #initial concentration of layers  , [C0,C1,C2,..]
114 D=[30e-11,30e-11,30e-11]     #diffusion coefficients of layers   , [D0,D1,D2,..]
115 Ce=0     #medium Concentration
116 De=30e-11    #medium Diffusion coefficient
117 t=0.75     #time (hour)
118 n=3     #number of layer
119 aa=diffusion(R,D,C,t,n,De,Ce)    #main class
120
```

Here, problem properties are defined and as line 119, the main class is initialized.

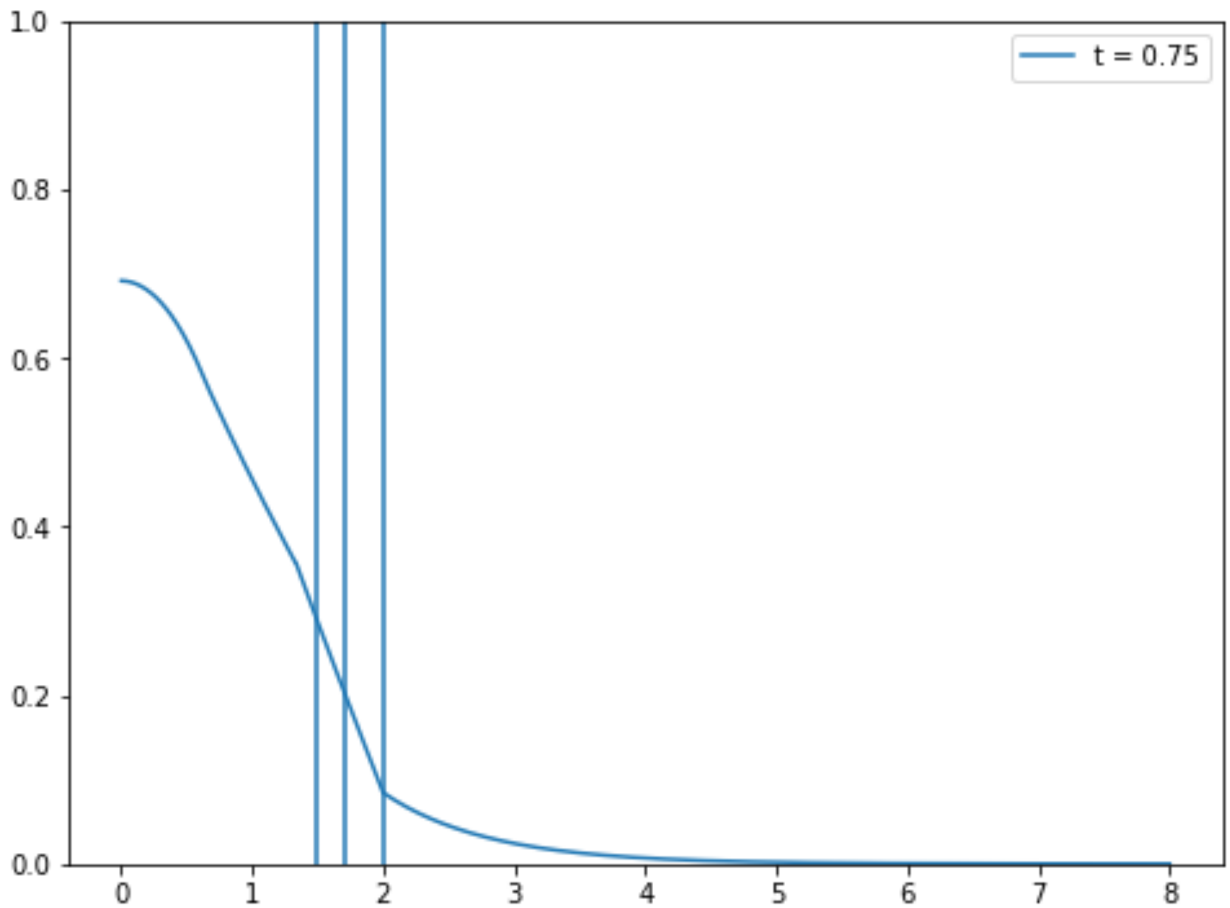
```python
def plotter(obj):
    start=0
    xxx=np.linspace(start,obj.R[0],200)
    yyy=[]
    for pp in xxx:
        yyy.append(obj.c_0(pp))
    start+=obj.R[0]
    for i in range(1,len(obj.R)):
        print(start)
        xxx=np.linspace(start,obj.R[i],200)
        for pp in xxx:
            yyy.append(obj.c_i(i,pp))
        start=obj.R[i]
    xxx1=np.linspace(obj.R[-1],4*obj.R[-1],200)
    for pp in xxx1:
        yyy.append(obj.c_e(pp))


    xxx=np.linspace(0,obj.R[-1],obj.n*200)
    xxx=np.hstack((xxx,xxx1))
    plt.figure(figsize=(8,6))
    plt.plot(xxx*1000*obj.R1,yyy,label='t = '+str(obj.t))
    plt.legend()
    plt.ylim(0,obj.Cmax)
    for i in obj.R:
        plt.axvline(x=i*1000*obj.R1)
    return yyy
```

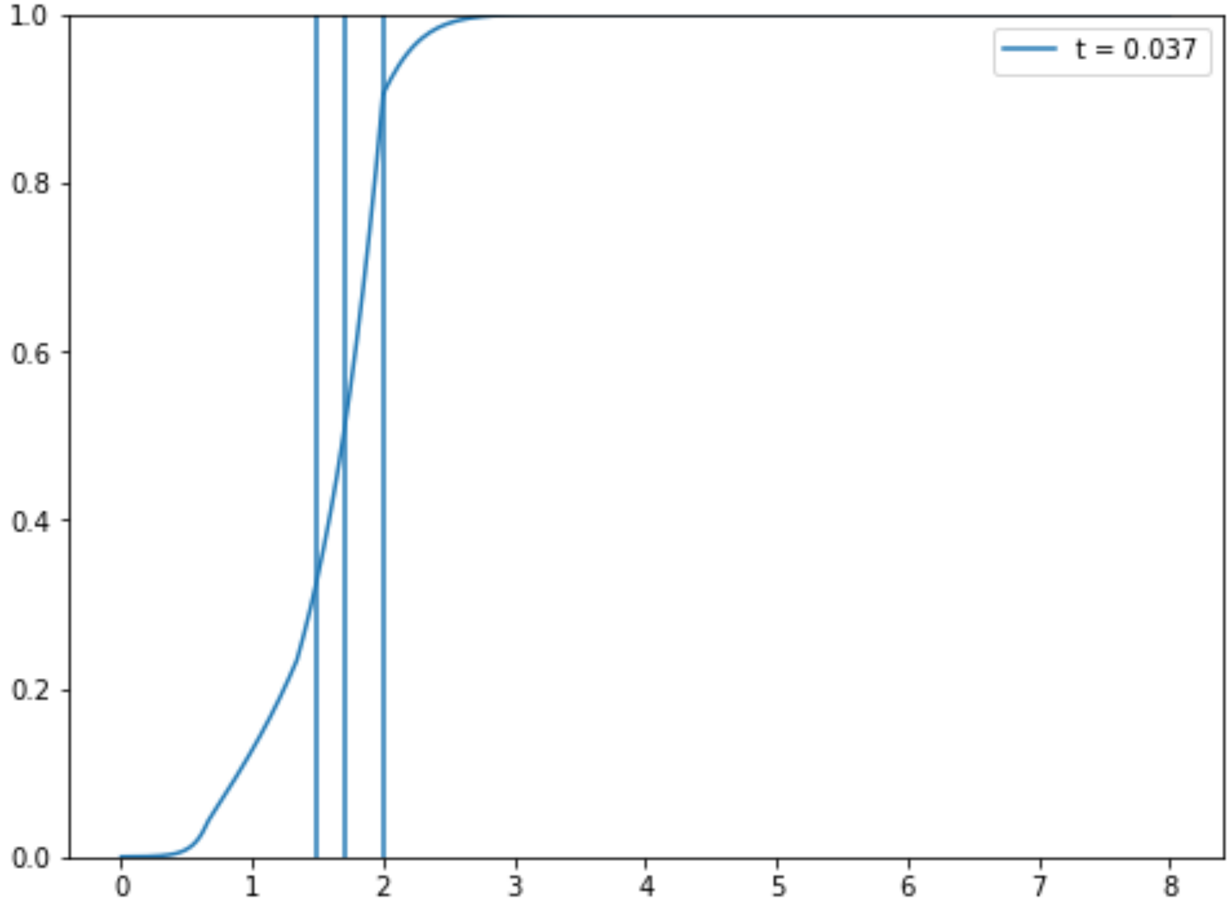
And finally for the case of visualization, the function above is defined. In this function, the initialized class is imported and concentration values as well as plot is exported.

3- Results

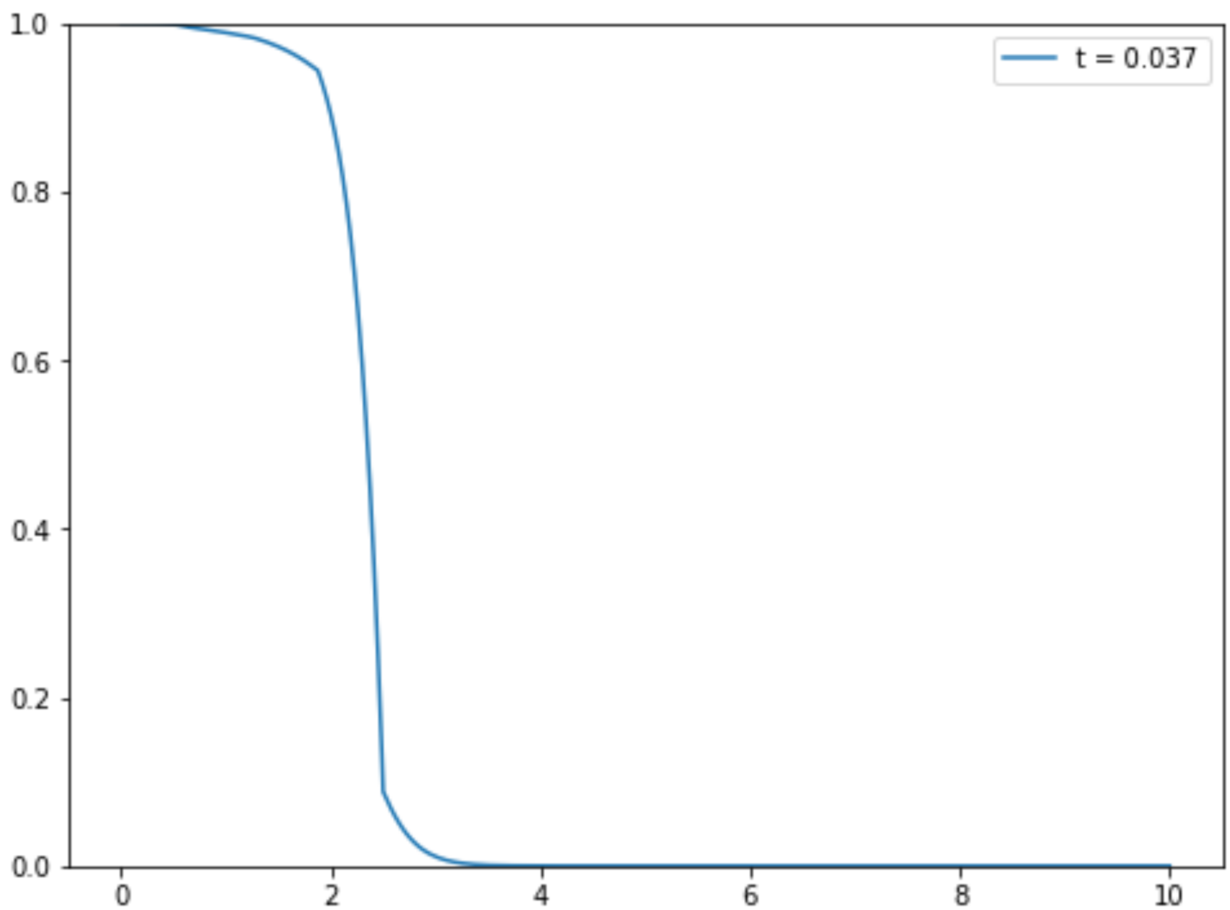
```
R=[1.5e-3,1.7e-3,2e-3]   #radius of layers [mm] , [R0,R1,R2,..]
C=[1,1,1]       #initial concentration of layers  , [C0,C1,C2,..]
D=[30e-11,30e-11,30e-11]     #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=0     #medium Concentration
De=30e-11   #medium Diffusion coefficient
t=0.75    #time (hour)
n=3    #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)   #main class
```



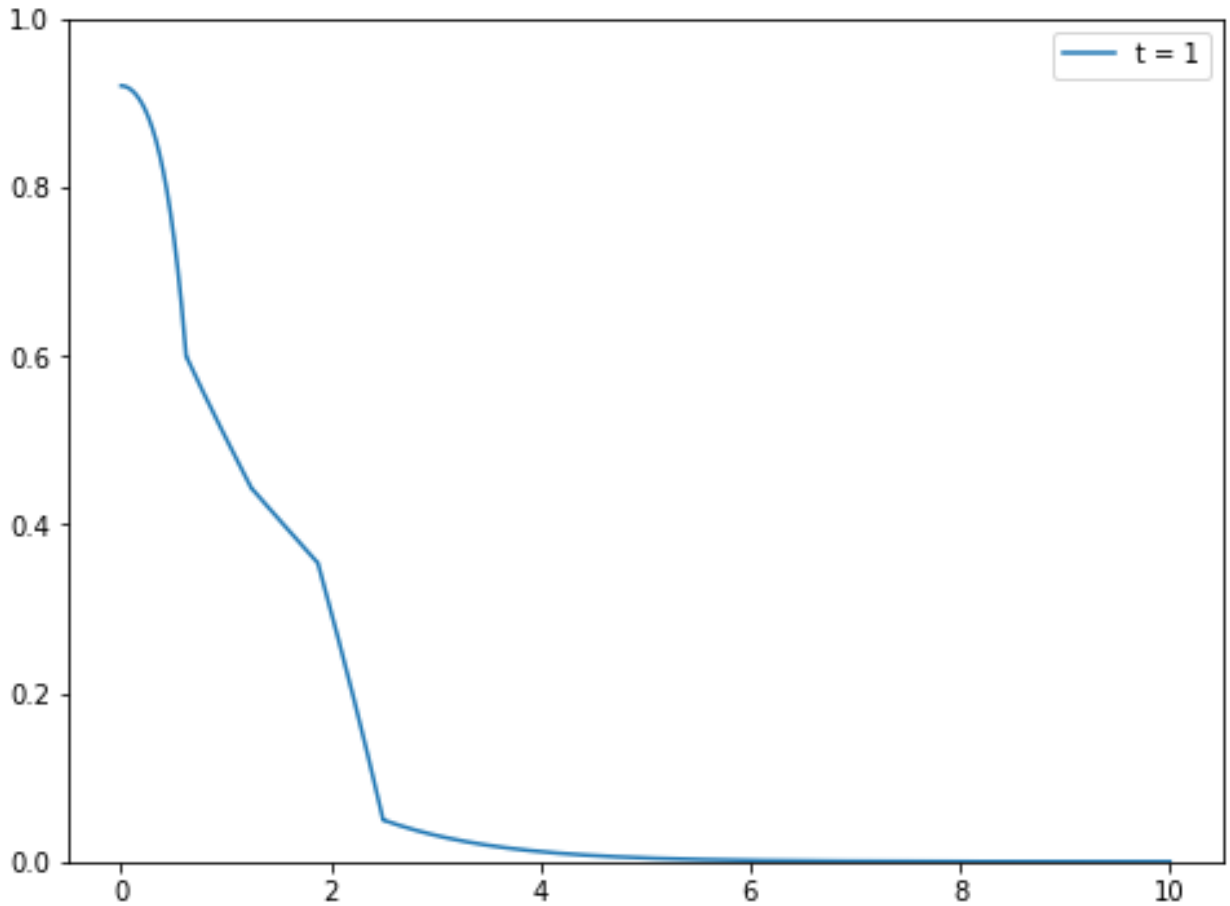
```
R=[1.5e-3,1.7e-3,2e-3]   #radius of layers [mm] , [R0,R1,R2,..]
C=[0,0,0]      #initial concentration of layers  , [C0,C1,C2,..]
D=[30e-11,30e-11,30e-11]     #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=1    #medium Concentration
De=30e-11   #medium Diffusion coefficient
t=0.037   #time (hour)
n=3    #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)   #main class
```



```
R=[1.5e-3,1.7e-3,2e-3,2.5e-3]   #radius of layers [mm] , [R0,R1,R2,..]
C=[1,1,1,1]       #initial concentration of layers  , [C0,C1,C2,..]
D=[30e-11,30e-11,30e-11,15e-11]     #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=0    #medium Concentration
De=20e-11   #medium Diffusion coefficient
t=0.037   #time (hour)
n=4   #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)   #main class
```



```
R=[1.5e-3,1.7e-3,2e-3,2.5e-3]   #radius of layers [mm] , [R0,R1,R2,..]
C=[1,1,1,1]      #initial concentration of layers  , [C0,C1,C2,..]
D=[3e-11,12e-11,30e-11,15e-11]     #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=0    #medium Concentration
De=20e-11   #medium Diffusion coefficient
t=1   #time (hour)
n=4   #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)   #main class
```

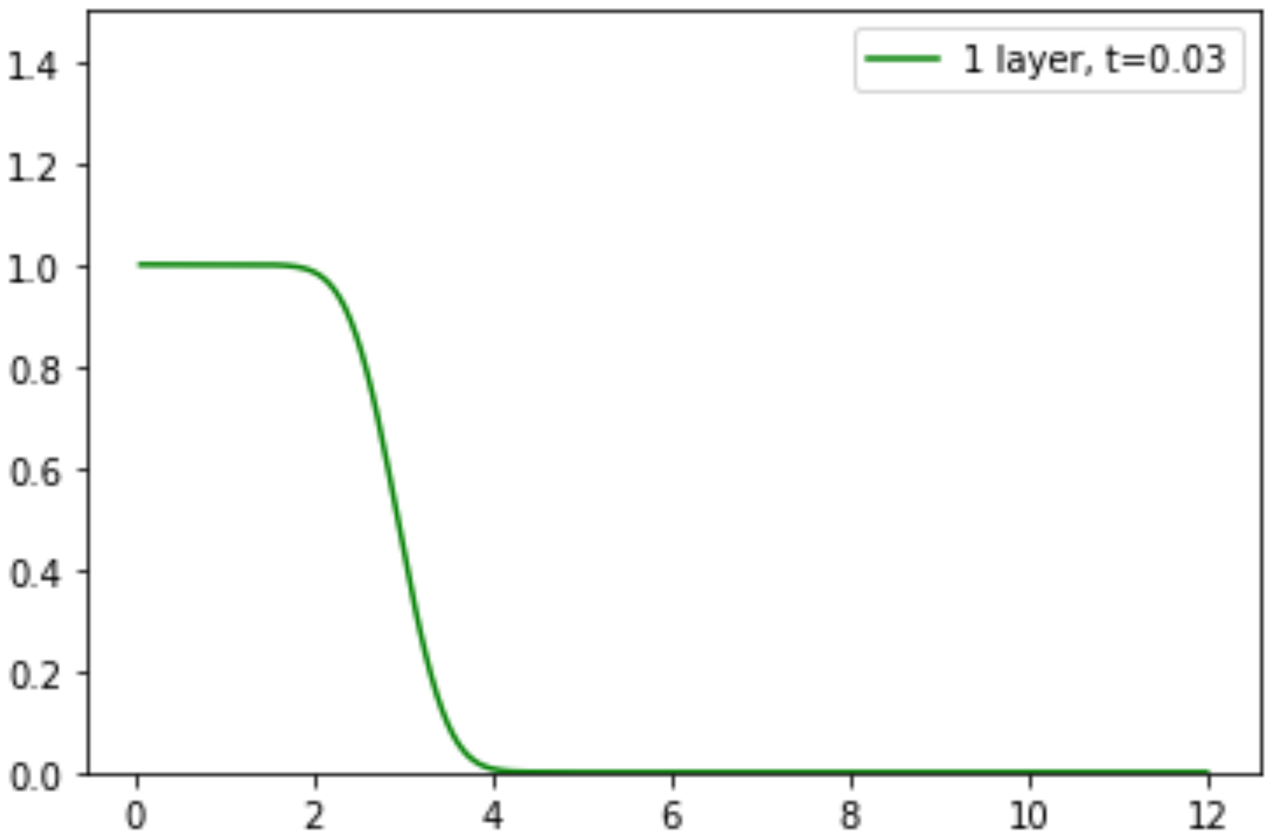


4- Comparison

In the next graph, 3 different models with similar properties are compared.

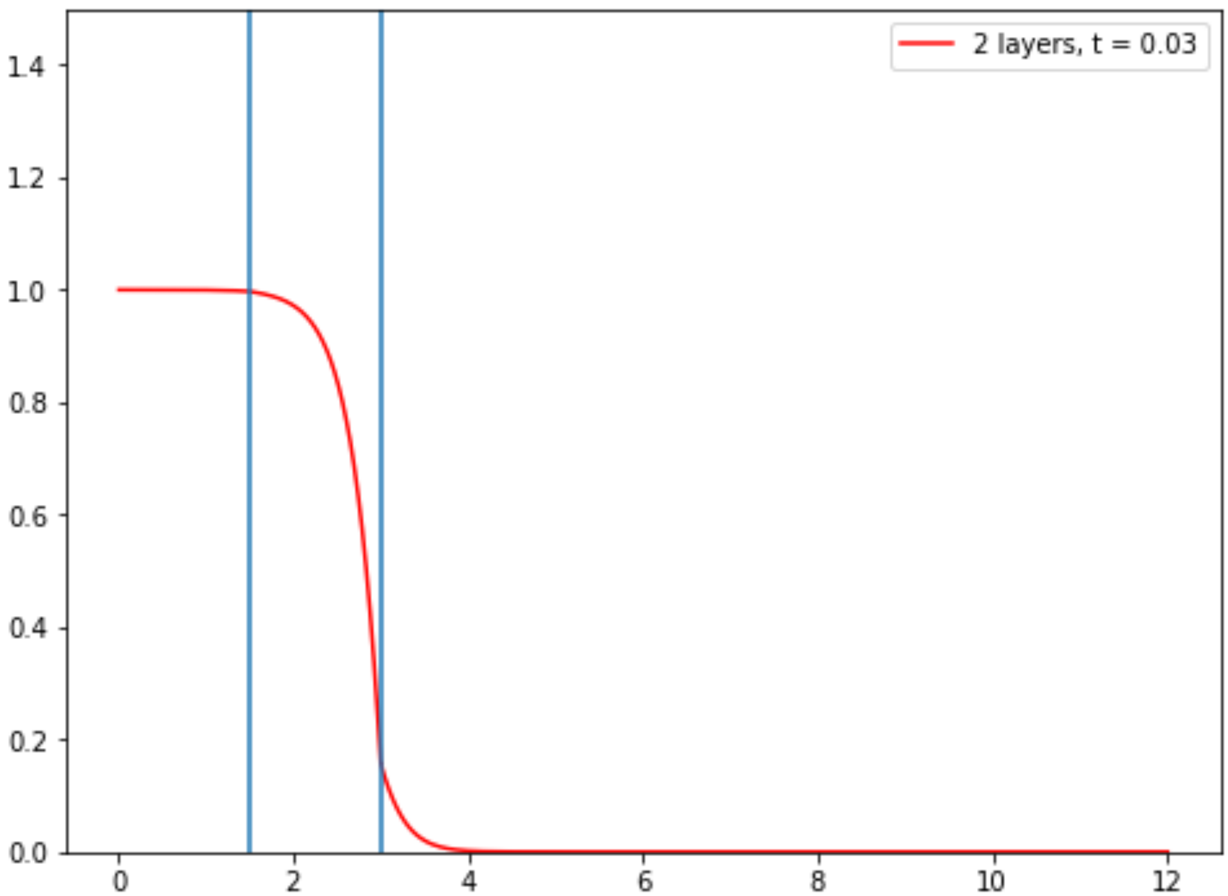
Single Layer: (one layer.py)

```
D=30e-11
R=3 #mm
t=0.03 #h
```



Two Layer: (core.py)

```
R=[1.5e-3,3e-3]    #radius of layers [mm] , [R0,R1,R2,..]
C=[1,1]       #initial concentration of layers  , [C0,C1,C2,..]
D=[30e-11,30e-11]     #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=0    #medium Concentration
De=20e-11   #medium Diffusion coefficient
t=0.03 #time (hour)
n=2 #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)    #main class
```



Three Layers (core.py):

```
R=[1e-3,2e-3,3e-3]    #radius of layers [mm] , [R0,R1,R2,..]
C=[1,1,1]       #initial concentration of layers  , [C0,C1,C2,..]
D=[30e-11,30e-11,30e-11]      #diffusion coefficients of layers  , [D0,D1,D2,..]
Ce=0     #medium Concentration
De=20e-11    #medium Diffusion coefficient
t=0.03 #time (hour)
n=3 #number of layer
aa=diffusion(R,D,C,t,n,De,Ce)    #main class
```