Neutron Diffusion

Jacopo Taddei

University of Bologna, Department of Physics

22/07/2023

Introduction

In this presentation we are concerned with the diffusion of neutrons in fissile material where collisions between these particles and nuclei result in the release of secondary neutrons. As the fissile material increases in size the radioactive core will become critical when the total density of neutrons increases exponentially. The result is a runaway nuclear reaction that can lead to an intense explosion.

Nuclear Reaction - Generalities

Consider the diffusion of U^{235} of (Pu^{239}), neutrons can be described mathematically on a domain Ω with boundary $\partial\Omega$ by the following equation:

$$\frac{\partial n}{\partial t} = \mu \nabla^2 n + \eta n \tag{1}$$

where $n = n(t, \vec{x})$ represent the neutron density function. $(\mu, \eta) > 0$ are called respectively the *Diffusion constant* $[m^2/s]$ and the *Neutron rate of formation* $[1/s]$. For each system we also have to impose suitable boundary and initial conditions.

The Three Systems

In this analysis we consider three different systems with two choices of the boundary conditions:

- 1D Cartesian coordinates - Dirichlet BCs.
- 3D Cartesian coordinates - Dirichlet BCs.
- 3D Spherical coordinates - Neumann BCs.

1D Cartesian coordinates - Dirichlet BCs

The equation (1), plus Dirichlet boundary conditions, in one dimension becomes:

$$\frac{\partial n}{\partial t} = \mu \frac{\partial^2 n}{\partial x^2} n + n\eta \tag{2}$$

$$BCs : n(t,0) = n(t,L) = 0 \qquad IC : n(0,x) = f(x)$$

We can now postulate a form for the solution that transforms this PDE into two different ODEs.

$$n = T(t)X(x) \longrightarrow \begin{cases} -\mu \dfrac{d^2 X}{dx^2} = \alpha X \\ \dfrac{dT}{dt} = (\eta - \alpha) T \end{cases} \tag{3}$$

Theoretical Considerations

From the analytical point of view these two equations have simple solutions:

$$T = A\exp((\eta - \alpha)t); \qquad X = B\sin\left(\sqrt{\frac{\alpha}{\mu}}x\right); \qquad (4)$$

Note: Here we used the BC to have $n = 0$ at $x = 0$. From BCs we can also understand that the argument of the sin function must equal $p\pi x/L$ for $p \in \mathbb{Z}$. Using the *superposition principle* the total solution becomes:

$$n = \sum_{p=1}^{\infty} a_p \exp((\eta - \alpha)t)\sin\left(\frac{p\pi}{L}x\right) \qquad (5)$$

Numerical Analysis - Initialization

The first step is to initialize the problem with the needed constants, to set the limits of space and time and to discretize them:

```
#Uranium
mu=2.3446*10**5
eta=1.8958*10**8

#Initialization of the problem
nx, nt=100, 100  # N. of space and time points
L=0.12 #Interval
t_f=4*10**(-7) #Final time
N=30 #Number of eigenvalues and eigenvectors

#Discretization of space and time
x=np.linspace(0,L,nx)
t=np.linspace(0,t_f,nt)
```

Numerical Analysis - Spatial ODE

The spatial ODE is solved as an eigenvalue problem, indeed the equation is discretized in the following manner:

$$-\mu \frac{d^2}{dx^2} X = \alpha X \longrightarrow D_{ij} X_j = \alpha_i X_i \tag{6}$$

where the matrix D_{ij} is the distretization of the Laplacian matrix using the finite difference calculus:

$$-\frac{\mu}{h^2} \begin{pmatrix} -2 & 1 & \cdots & \cdots & & 0 \\ 1 & -2 & 1 & \cdots & & \vdots \\ \vdots & & & \ddots & & \\ 0 & \cdots & \cdots & & 1 & -2 \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{pmatrix} = \alpha_i \begin{pmatrix} X_1 \\ X_2 \\ \vdots \\ X_{N-1} \end{pmatrix} \tag{7}$$

Numerical Analysis - Spatial ODE

Numerically this is implemented as:

```
    def X_ode(L, mu, N):
    #Discretization of space
    x=np.linspace(0,L,nx)
    h=x[1]-x[0]
    #Creation of Kinetic Matrix
    T=np.zeros((nx-2)**2).reshape(nx-2,nx-2)
    for i in range(nx-2):
        for j in range(nx-2):
            if i==j:
                T[i,j]=-2
            elif np.abs(i-j)==1:
                T[i,j]=1
            else:
                T[i,j]=0
    T*=(-mu/(h**2))
```

Numerical Analysis - Spatial ODE


```
#Eigenvalues and Eigenvectors
val, vec=np.linalg.eig(T)
#Ordering of Eigenvalues
z=np.argsort(val)
#First N Eigenvalues
z=z[0:N]
# Addition of zeros at the borders of
    Eigenvectors
vec_new = np.zeros((nx, len(z)))
for i in range(len(z)):
    vec_new[:, i] = np.concatenate(([0],
        vec[:, z[i]], [0]))
return val[z], vec_new
```

In this way we find the values of α_i with the correspondent eigenvectors.

Numerical Analysis - Spatial ODE

After this we rescaled the eigenvectors in such a way that they are sin functions:

```
#Usefull rewrite of Eigenvectors and rescaling
def separation(Xi,N):
    X_q=[]
    for i in range(N):
        X_q_i=Xi[:,i]
        X_q.append(X_q_i)
        #Here we rescale the Eigenvectors as
            the sin function
        scale=1/(max(abs(X_q[i])))
        X_q[i]*=scale
    return X_q
```

Numerical Analysis - Temporal ODE

To solve the temporal ODE we used the library **scipy.integrate.odeint**:

```
1  #Model of the temporal differential equation
2  def model(y, t, eta, alfa):
3      dydt=(eta-alfa)*y
4      return dydt
5
6  #Time ODE solver
7  def T_ode(t, eta, alfa): #t is a float Number
8      t_vec=np.linspace(0, t, nt)
9      #Initial condition
10     y0=1
11     T_solution=np.squeeze(odeint(model, y0,
           t_vec, args=(eta,alfa)))
12     return T_solution
```

Numerical Analysis - Critical Length

At this point we can solve both of the ODEs and the second step is to find the value of the critical length, i.e. when the argument of the exponent becomes positive, thus divergent.

```
#Critical length finder
def find_the_boom(mu, eta):
    #accuracy
    e=10**(-4)
    #Shift in L (1mm)
    dL=0.001
    #L temporal for loops
    L_0, L_boom=0, 0
```

Now we can create a loop that analyzes the derivative of the temporal function.

Numerical Analysis - Critical Length


```
while dL>=e:
    L_boom=L_0+dL
    #First we solve the X ode to find
    #the minimum eigenvalue
    alfas=X_ode(L_boom, mu, N)[0]
    #Minimum eigenvalue: the worst
    alfa_min=alfas[0]
    T_0=T_ode(t_f, eta, alfa_min)
    T_0_der=np.gradient(T_0)
    sign_check=np.average(T_0_der)
    #Evaluation of the change in
    #the slope of the solution
    if sign_check>0:
        dL=dL/2
    else:
        L_0=L_boom
return L_boom
```

Numerical Analysis - Critical Length

From the theory we expect a critical length equal to:

$$L_{th} = \pi \sqrt{\frac{\mu}{\eta}} = 11.050 cm \tag{8}$$

Numerically we find:

$$L_{num} = 11.048 cm \tag{9}$$

Numerical Analysis - a_p Coefficients

Now, using as Initial condition a Gaussian function well picked in the middle of the interval we can compute the coefficients:

```
1  def f(x):
2      lam=100
3      l=L/2
4      f=np.exp(-lam*((x-l)**2/(l**2)))
5      return f
6
7  #Coefficients of the series expansion
8  def coefficients(f,X,N):
9      ap=np.zeros(N)
10     cost=2/L
11     for i in range(N):
12         fun= f*X[i]
13         coeff= cost * trapz(fun, x)
14         ap[i]= coeff
15     return ap
```

Numerical Analysis - a_p Coefficients

The resulting coefficients are in great accordance with [1], except for the sign. [1] Here we report the non zero coefficients among the first thirty:

p	a_p
1	0.1762
3	0.1679
5	-0.1519
7	-0.1310
9	-0.1086
11	0.0853
13	0.0625
15	-0.0443

p	a_p
17	-0.0298
19	0.0191
21	0.0117
23	0.0068
25	0.0038
27	-0.0020
29	-0.0010

[1] The signs differs because of a phase displacement of 180° between the solution that we extract from the eigenvalue problem and the analytical sin functions. The coefficients automatically peak a correct minus sign when this displacement is present.

Numerical Analysis - Final Solution

At the end it is sufficient to combine all of these results:

```
#Combination of space an time solutions (
    without coefficients yet)
def Space_time_eig(t, X, A, N):
    for  i in range(N):
        X[i]*=T_ode(t, eta, A[i])
    return X

#Complete solution with coefficients
def function_exp(ap,t,X,A, N):
    X_st=Space_time_eig(t,X,A,N)
    for i in range(N):
        X_st[i]*=ap[i]
    return X_st
```

Numerical Analysis - Results

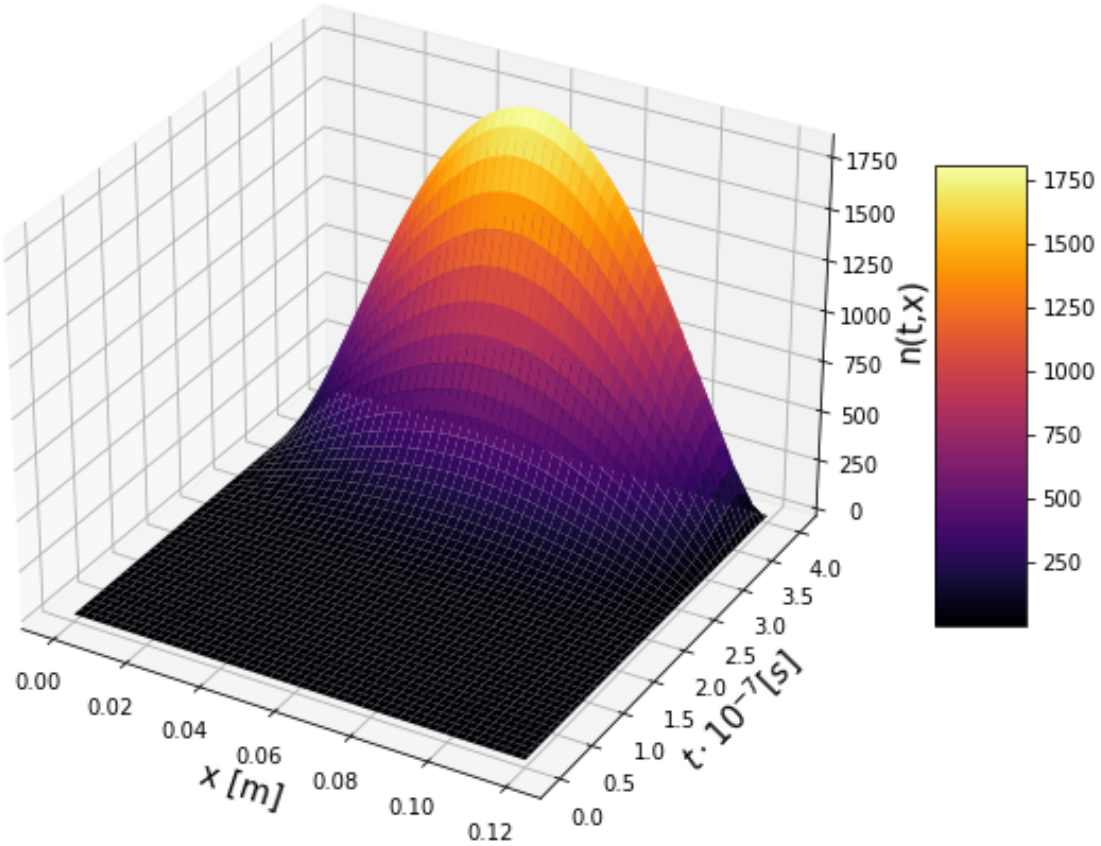


Figure: Neutron density evolution in time span $t \in [0, 0.4]\mu s$ for a overcritical length L=12.00 cm

3D Cartesian coordinates - Dirichlet BCs

This case is the straight generalization of the previous toy model. The equation for the neutron diffusion (1) becomes:

$$\frac{\partial n}{\partial t} = \mu \left(\frac{\partial^2 n}{\partial x^2} + \frac{\partial^2 n}{\partial y^2} + \frac{\partial^2 n}{\partial z^2} \right) + \eta n \tag{10}$$

We deal with a cubic volume of fissile material and the boundary conditions this time set to zero the neutron density on all of the faces of the cube. We make the following ansatz about the form of the solution:

$$n = T(t)X(x)Y(y)Z(z) \tag{11}$$

We divide in this way a PDE into four ODEs.

Theoretical Considerations

The difference with the previous case is that now we have to solve three spatial equations instead of just one:

$$-\mu \frac{d^2}{dx^2} X = \alpha_x X \quad -\mu \frac{d^2}{dy^2} Y = \alpha_y Y \quad -\mu \frac{d^2}{dz^2} Z = \alpha_z Z \tag{12}$$

The final solution is the following:

$$n = \sum_{p,q,r}^{\infty} a_{p,q,r} \exp([\eta - \alpha]t) \sin\left(\frac{p\pi}{L} x\right) \sin\left(\frac{q\pi}{L} y\right) \sin\left(\frac{r\pi}{L} z\right) \tag{13}$$

Where the α in the exponent is $\alpha = \alpha_x + \alpha_y + \alpha_z$

Numerical Analysis - Critical Length

The critical length is found with the same procedure as before, but in 3D:

```
 #Critical length finder
def find_the_boom(mu, eta):
    #Accuracy
    e=10**(-4)
    #Shift in L (1mm)
    dL=0.001
    #L temporal for loops
    L_0, L_boom=0, 0
    while dL>=e:
        L_boom=L_0+dL
        #First we solve the X, Y, Z ODEs to
            find the minimum eigenvalue
        alfa_x=X_ode(L_boom, mu, N)[0]
        alfa_y=Y_ode(L_boom, mu, N)[0]
```

Numerical Analysis - Critical Length


```
        alfa_z=Z_ode(L_boom, mu, N)[0]
        alfas=alfa_x+alfa_y+alfa_z
        #Minimum eigenvalue: the worst
        alfa_min=alfas[0]
        T_0=T_ode(t_f,L_boom,eta, alfa_min)
        T_0_der=np.gradient(T_0)
        sign_check=np.average(T_0_der)
        #Evaluation of the change in the slope
            of the solution
        if sign_check>0:
            dL=dL/2
        else:
            L_0=L_boom
    return L_boom
```

Numerical Analysis - Critical Length

From the theory we expect a critical length equal to:

$$L_{th} = \pi \sqrt{\frac{3\mu}{\eta}} = 19.136cm \tag{14}$$

Numerically we find:

$$L_{num} = 19.138cm \tag{15}$$

Numerical Analysis - $a_{p,q,r}$ Coefficients

The initial function this time is:

```
#Initial condition
def f(x,y,z):
    return (8/L**3)*x*y*z*(1-x/L)*(1-y/L)*(1-z/
        L)
```

Considering the separability of the integral for the computation of the coefficients we can divide it in pieces and compute the three integrals autonomously.

Numerical Analysis - $a_{p,q,r}$ Coefficients


```
#Coefficients of the series expansion
def coef_finder(N):
    ap=np.zeros((N,N,N))
    for i in range(N):
        for j in range(N):
            for k in range(N):
                fx=x*(1-x/L)*X[i]
                fy=y*(1-y/L)*Y[j]
                fz=z*(1-z/L)*Z[k]
                ap[i, j, k] =(8/L**3)**2 *
                    trapz(fx,x)*trapz(fy,y)*
                    trapz(fz,z)
                #print(i+1,j+1,k+1,
                    '---------->', ap[i,j,k])
    return ap
```

Numerical Analysis - $a_{p,q,r}$ Coefficients

The results are in accordance with [1], we report the non zero coefficients among the first 64:

p	**q**	**r**	$a_{p,q,r}$
1	1	1	1.3741E-01
1	1	3	-5.0962E-03
1	3	1	-5.0962E-03
1	3	3	1.8893E-04
3	1	1	-5.0962E-03
3	1	3	1.8893E-04
3	3	1	1.8893E-04
3	3	3	-7.0048E-06

Note the great number of null combinations. These coefficients are calculated at a length L=19.14 cm.

Numerical Analysis - Final Solution

Now we have just to merge all the pieces to get the final solution. At first we combine the spatial eigenvectors, then we add the temporal part and at the end the coefficients:

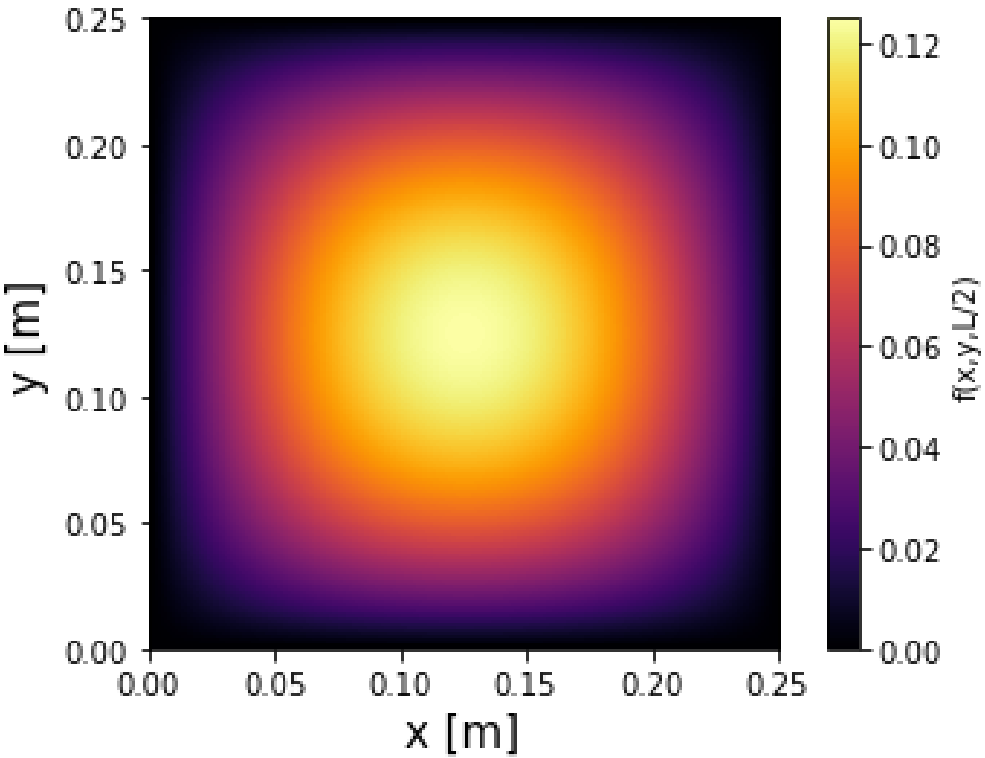
```
1 #Combination of all of the spatial solutions
2 def combined_eigenvec(X,Y,Z,N):
3     R=np.zeros((N,N,N,100))
4     for i in range(N):
5         for j in range(N):
6             for k in range(N):
7                 R[i,j,k]=X[i]*Y[j]*Z[k]
8     return R
```

Numerical Analysis - Final Solution

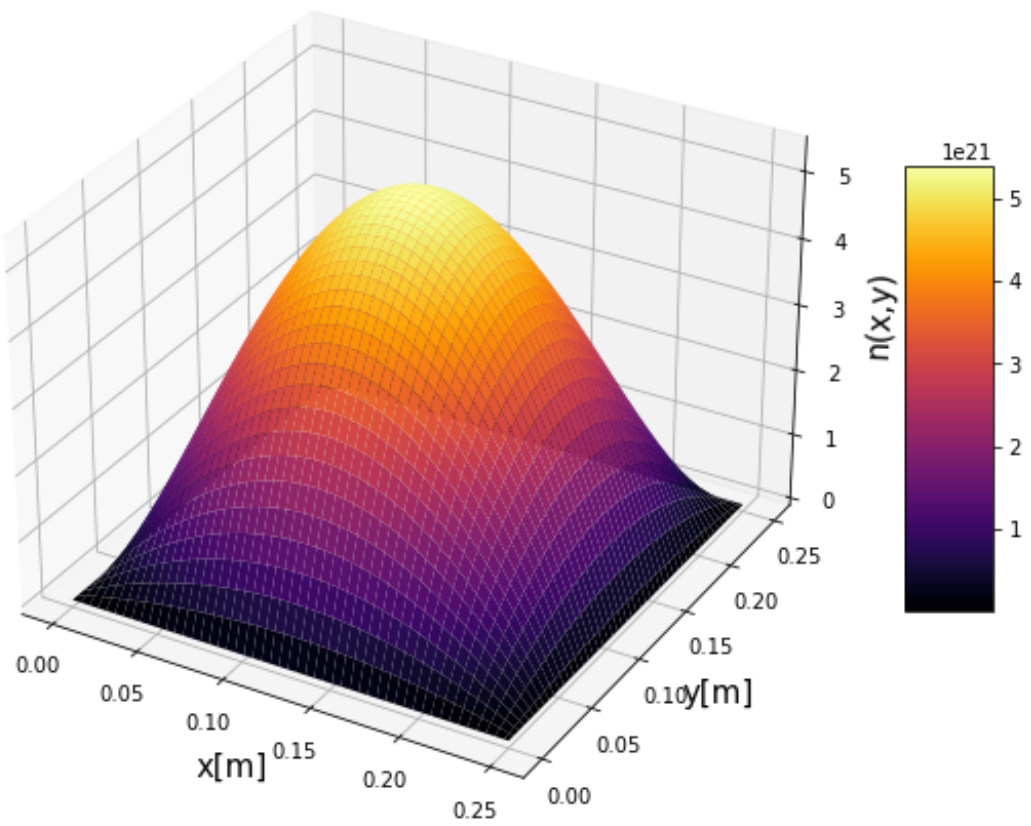

```
#S.T. solution (without coefficients yet)
def Space_time_eig(t,R,Ax,Ay,Az,N):
    for i in range(N):
        for j in range(N):
            for k in range(N):
                alfa=Ax[i]+Ay[j]+Az[k]
                R[i,j,k]*=T_ode(t, L, eta,
                    alfa)
    return R
#Complete solution with coefficients
def function_exp(ap,t,R,Ax,Ay,Az,N):
    R_st=Space_time_eig(t, R, Ax, Ay, Az, N)
    for i in range(N):
        for j in range(N):
            for k in range(N):
                R_st[i,j,k]*=ap[i,j,k]
    return R_st
```

Numerical Analysis - Results

Summing along the indices we obtain the evolution of the neutron density. We report a plot of the initial condition (at time t=0) and the evolution for a overcritical length (L=25.0 cm) after a time t=8.0E-07. To visualize the solution we perform a slice at the height z=L/2.



Numerical Analysis - Results



It is clear the strong divergence of the neutron density after less than a micro second.

3D Spherical coordinates - Neumann BCs

This system is made by a spherical volume of fissile material. The equation (1) in spherical coordinates becomes:

$$\frac{\partial n}{\partial t} = \mu \left(\frac{\partial^2}{\partial r^2} + \frac{2}{r}\frac{\partial}{\partial r} \right) n + \eta n \tag{16}$$

This time the boundary conditions allow the escape of neutrons from the system. We use the so called Neumann BC, also this time imposing a suitable initial condition.

$$BCs : \frac{dn(t, r_1)}{dr} = -\frac{3}{2}\frac{n(t, r_1)}{\lambda_t} \qquad ICs : n(0, r) = f(r) \tag{17}$$

where r_1 is the radius of the sphere and λ_t is the transport free path.

Theoretical Consideration

Again we make the ansatz that transforms the PDE (17) into two separate ODE.

$$n = T(t)R(t) \tag{18}$$

The form of the two resulting equations changes with respect to the previous cases:

$$\frac{dT}{dt} = -\alpha T; \qquad \frac{d^2R}{dr^2} + \frac{2}{r}\frac{dR}{dr} + k^2 R = 0: \tag{19}$$

where $k = \sqrt{\dfrac{\eta + \alpha}{\mu}}$.

Theoretical Consideration

From the analytical point of view the solutions to the equations (19) are well known:

$$T = A\exp(-\alpha); \qquad R = \frac{B}{r}\sin(kr); \tag{20}$$

If we plug the solution of the radial ODE into the boundary conditions we obtain the theoretical relation among r and α (or equivalently k):

$$\frac{dR}{dr} = -\frac{3}{2}\frac{R}{\lambda_t} \longrightarrow -1 + kr\cos(kr) + \frac{3}{2}\frac{r}{\lambda_t} = 0 \tag{21}$$

The criticality condition is manifest from the solution of the temporal ODE, indeed the stability point is when $\alpha = 0$.

Numerical Analysis - Critical Radius

We omit the explanation of the usual discretization of space and time which is analogous to the one presented in the first system. In this case to find the critical radius we use again the analytical relation $r(\alpha)$ and we solve it numerically in the following way:

```
#Analytical relation k(a)
def f(k,r):
    lam_t=0.0360
    #lam_t=0.0411 for Plutonium Analysis
    return k*r*(np.tan(k*r))**(-1)+3/2*r/lam_t
        -1
```

After setting $\alpha = 0$

```
#Critical radius
r_crit=fsolve(f_crit,0.05)
```

Numerical Analysis - Temporal ODE

Using the latter relation we can solve it in the opposite manner, indeed fixing a overcritical value of the radius we can extract the correct value of α and consequently solve the differential equations. The temporal ODE is resolved in the same manner as the previous cases:

```
1 #Time ODE solver
2 def T_ode(t, alfa):
3     t_vec=np.linspace(0, t, nt)
4     #initial condition
5     y0=1
6     T_solution=np.squeeze(odeint(model, y0,
          t_vec, args=alfa))
7     return T_solution
```

Numerical Analysis - Radial ODE

In order to solve this radial ODE with the Neumann boundary conditions we have to recast the ODE, which is second order, into a system of two linear differential equations.

```
#Setting for the Radial ODE
def equation(r,R):
    dR=R[1]
    ddR=-(2/r)*R[1]-k_sc**2*R[0]
    return [dR, ddR]
#This sets the final condition as dR=-3/2*R/lam and the initial condition (at the origin) = k as the limit of the analytic solution
def boundary_conditions(Ra, Rb):
    return np.array([Ra[0]-k_sc, Rb[1]+(3/(2*lam_t))*Rb[0]], dtype=object)
```

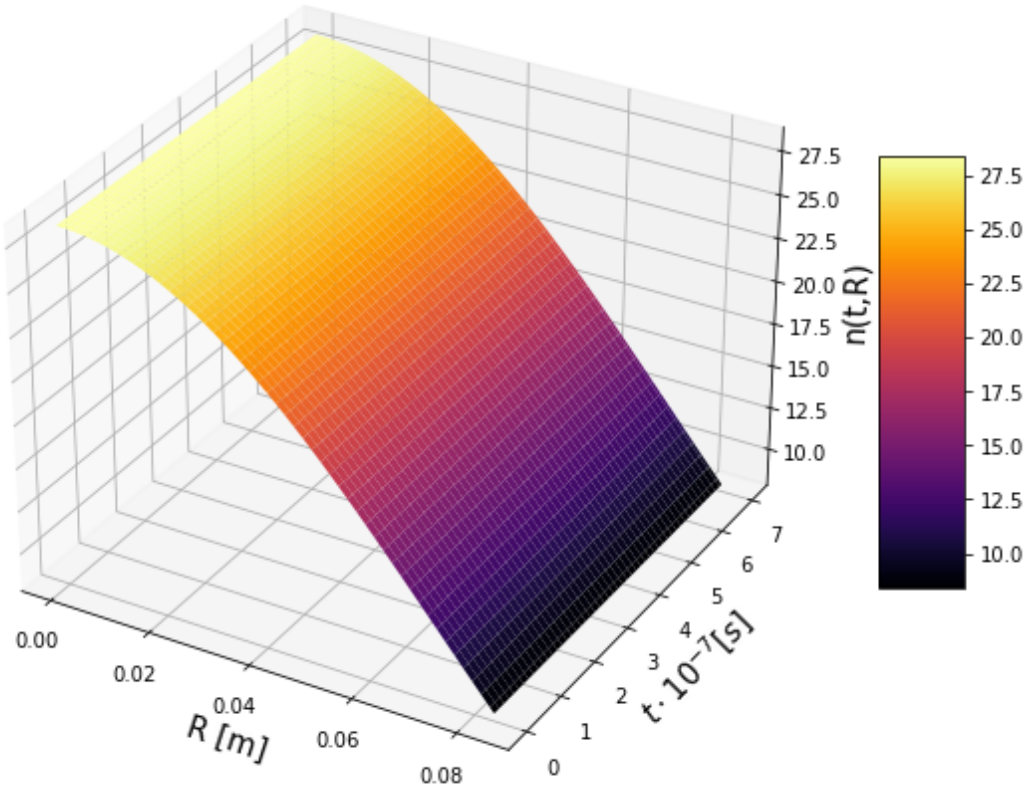
Numerical Analysis - Radial ODE

At the end we can solve the equation and combine the solutions to have the final result.

```
1 #Initial Guess for R and dR
2 R_guess=np.zeros((2,nr))
3 R_guess[0]=1
4
5 #Solution of the differential equation for R
6 solution= solve_bvp(equation,
      boundary_conditions, r, R_guess)
7 R_sol=solution.sol(r)[0]
```

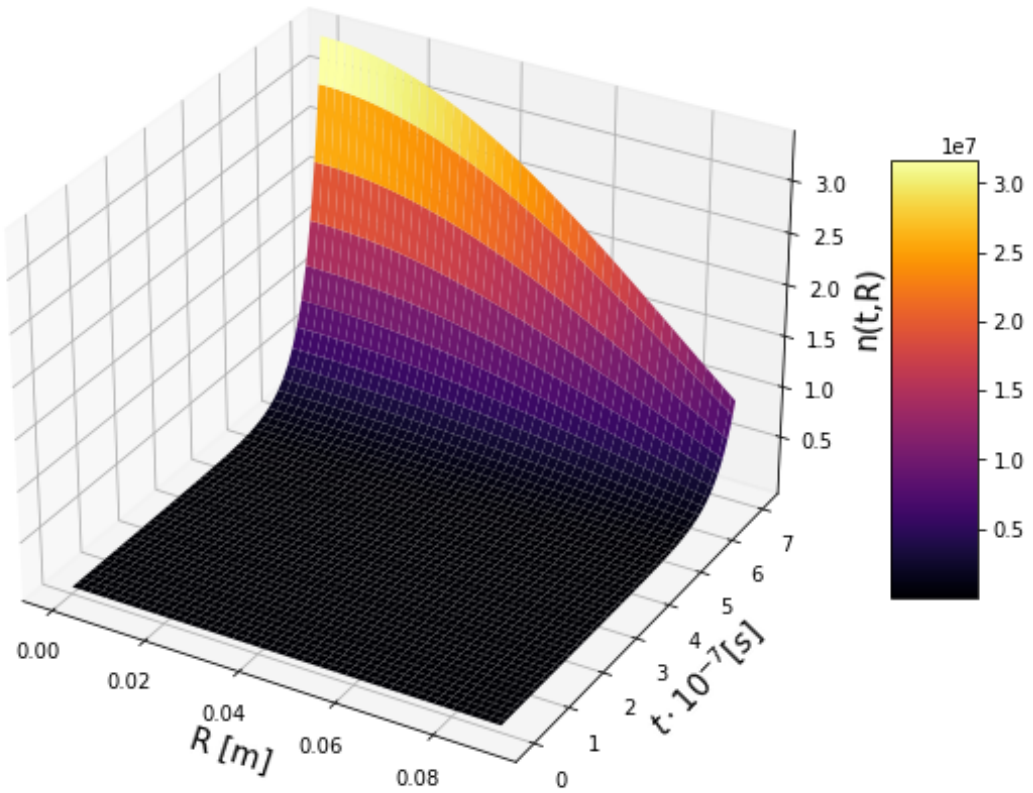
Numerical Analysis - Results

We report the plot of the neutron density at a critical length, thus in the static regime where there is not the temporal evolution since $\alpha = 0$ at $r = r_{crit} = 8.363cm$.

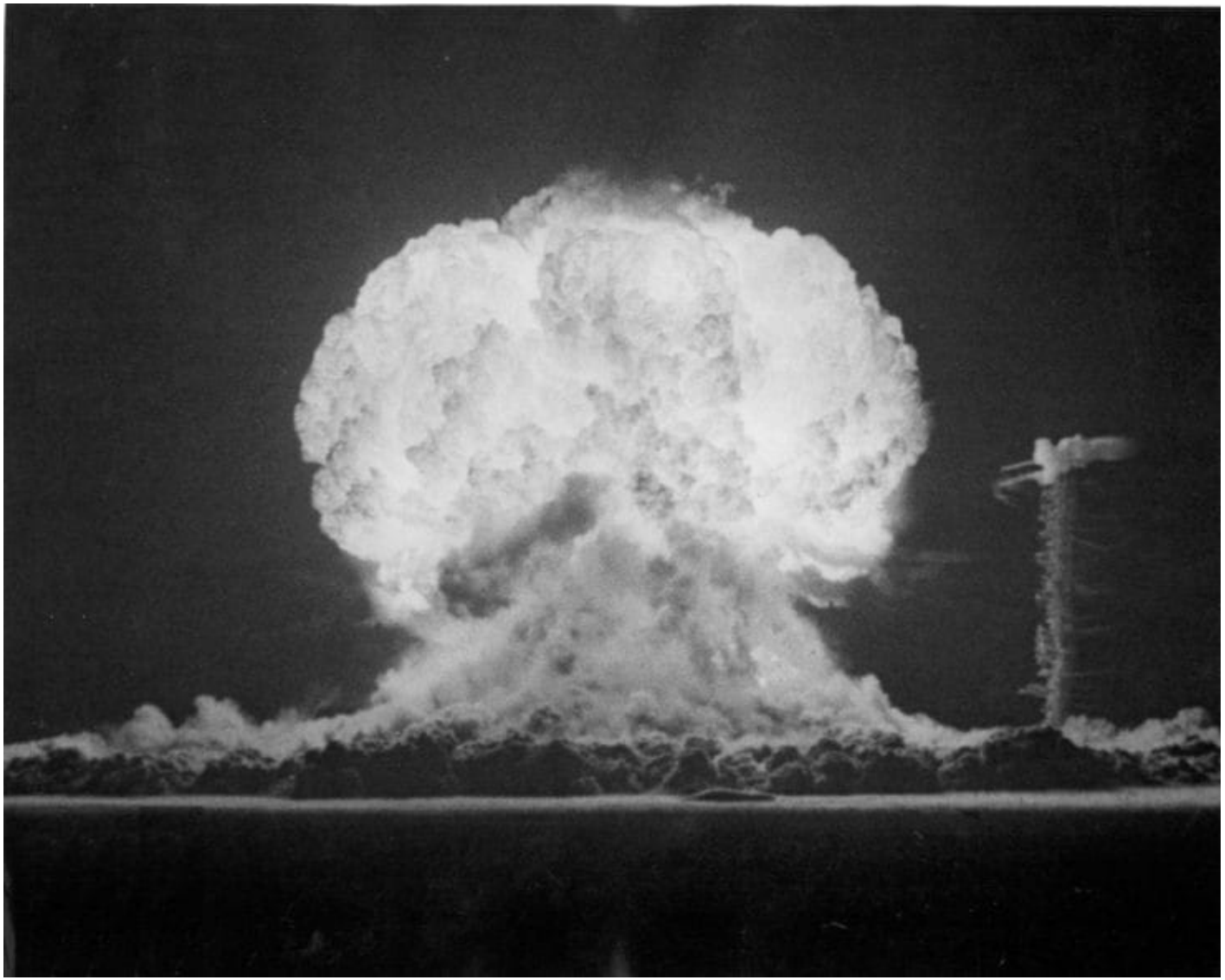


Numerical Analysis - Results

If instead we consider an overcritical value for the radius, for example $r = 9.0cm$ and we wait a time of $t = 7.0E-07$ we obtain the usual divergence of the neutron density.



Thanks For The Attention



References

[1] Graham W. Griffiths. Neutron diffusion. 2015.