Steady-State 1D Conduction in a Cylindrical Wall with Internal Heat Generation

This report presents the analysis of heat conduction in a cylindrical tube used to transport hot fluids, for example, in a heating system. The tube in question has the following thermal characteristics:

- **Matherial**: carbon steel with thermal conductivity $\lambda = 45\ W/mK$.
- **Tube dimensions**:
  - internal radius $R_{int} = 0.05\ m$;
  - external radius $R_{ext} = 0.1\ m$;
  - cylinder height $H = 2\ m$.
- **Boundary conditions**:
  - internally, the tube is in contact with warm water at $T_A = 150\ °C$ and heat transfer coefficient of $\alpha_A = 600\ W/m^2K$;
  - externally, the tube is exposed to ambient air at $T_B = 25\ °C$ and heat transfer coefficient $\alpha_B = 20\ W/m^2K$.
- **Internal heat generation**:
  - It is assumed that there is heat generation due to energy losses, equal to $q_v = 10^5\ W/m^3$.

Analytical Solution

As a first approach to the problem, the analytical solution is proposed. The following is the heat conduction equation in cylindrical coordinates.

$$\rho C_p \frac{\partial T}{\partial t} = \frac{1}{r}\frac{\partial}{\partial r}\left(\lambda r \frac{\partial T}{\partial r}\right) + \frac{1}{r^2}\frac{\partial}{\partial \theta}\left(\lambda \frac{\partial T}{\partial \theta}\right) + \frac{\partial}{\partial z}\left(\lambda \frac{\partial T}{\partial z}\right) + \dot{q}_v$$

First, this expression can be simplified considering the heat flux dependent only on the radius direction, then it can be reduced more knowing that:

- $\lambda = constant$;
- $\dot{q}_v = constant$;
- $\frac{\partial T}{\partial t} = 0\ \ permanent\ regime.$

$$\frac{d^2T}{dr^2} + \frac{1}{r}\frac{dT}{dr} + \frac{\dot{q}_v}{\lambda} = 0$$

This differential equation has an analytical solution which can be found deducing the integrating constants C_1 and C_2 applying the boundary conditions of this case.

$$T = -\frac{\dot{q}_v}{4\lambda} r^2 + C_1 \ln r + C_2$$

Boundary Conditions

- At $r = R_{int}$

$$-\lambda \left(\frac{dT}{dr}\right)_{r=R_{int}} = \alpha_A \left(T_A - T_{r=R_{int}}\right)$$

$$T_{r=R_{int}} = -\frac{\dot{q}_v}{4\lambda} R_{int}^2 + C_1 \ln R_{int} + C_2$$

- At $r = R_{ext}$

$$-\lambda \left(\frac{dT}{dr}\right)_{r=R_{ext}} = \alpha_B \left(T_{r=R_{ext}} - T_B\right)$$

$$T_{r=R_{ext}} = -\frac{\dot{q}_v}{4\lambda} R_{ext}^2 + C_1 \ln R_{ext} + C_2$$

The temperature distribution through the cylinder has been found implementing these equations in a Python code.

Analytical Results

In order to compare the results obtained by the analytical solution with those obtained by numerical analysis, the temperatures have been calculated at the corresponding radius of the nodes used for the numerical solution, which will be explained later in this report.

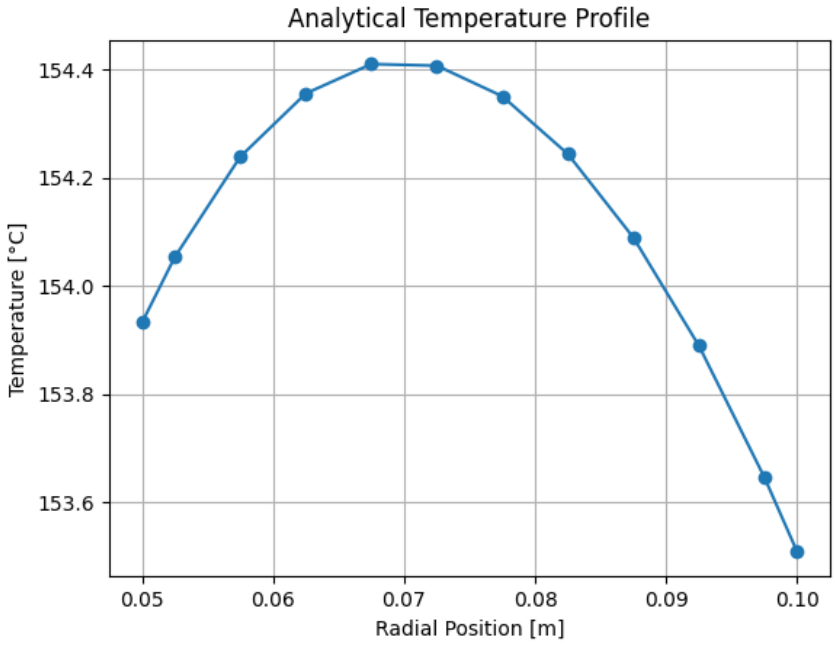

Figure 1. Temperature plot obtained analytically.

Table 1. Temperature distribution obtained analytically.

Node	Radius [m]	Temperature [°C]
1	0.0500	153.932746
2	0.0525	154.053833
3	0.0575	154.239490
4	0.0625	154.356384
5	0.0675	154.410833
6	0.0725	154.407794

7	0.0775	154.351235
8	0.0825	154.244376
9	0.0875	154.089869
10	0.0925	153.889925
11	0.0975	153.646403
12	0.1000	153.508803

Numerical Solution

The problem has been solved also numerically, through the implementation of a code using two different resolution methods:

- Gauss-Seidel algorithm which iteratively updates temperature at each node using neighboring values until convergence;
- Tri-Diagonal Matrix Algorithm (TDMA) which solves tridiagonal systems created by the discretized 1D heat conduction equation.

Firstly, the domain has been divided in control volumes, a node has been positioned in the center of each control volume, with two additional nodes at the boundaries. As for the analytical solutions, λ and $\dot{q}_v$ have been assumed nondependent on the temperature and the tube is in permanent regime.

The heat conduction equation is discretized as shown below:

$$a_P T_P = a_E T_E + a_W T_W + bp$$

Once the mesh has been defined it is possible to apply the two different solvers to obtain the temperature profile of the tube.

Code Structure

The code is structured as follows:

1. Input Data:
    - Physical: $R_{int}, R_{ext}, H, \lambda, T_A, T_B, \alpha_A, \alpha_B, \dot{q}_v$;
    - Numerical: number of control volumes (N), initial temperature guess (T_{start}), tolerance (δ), solver (Gauss-Seidel or TDMA).
2. Previous calculations: initialize the mesh as described above.
3. Initial temperature (Gauss-Seidel only): the initial temperature condition is applied to every node of the domain.
4. Evaluation of the discretization coefficients and P, R factors (TDMA).
5. Apply solver: find the temperature at each node using Gauss-Seidel or TDMA.
6. Convergence check: only for Gauss-Seidel, if it's not convergent it is necessary to do another iteration going back to step 5.
7. Plotting of the temperature distribution and comparison with the analytical solution.
8. Validation check: global energy balance.
9. End

Numerical Results

The numerical resolution uses the same number of nodes as the analytical approach to allow for a straightforward and accurate comparison of results. In this case the mesh has been defined with 10 control volumes, corresponding to 12 nodes.

The Gauss-Seidel and TDMA solutions for the reference case converge to similar temperature distributions. TDMA proved more computationally efficient, while Gauss-Seidel required more iterations to reach convergence.

Gauss-Seidel

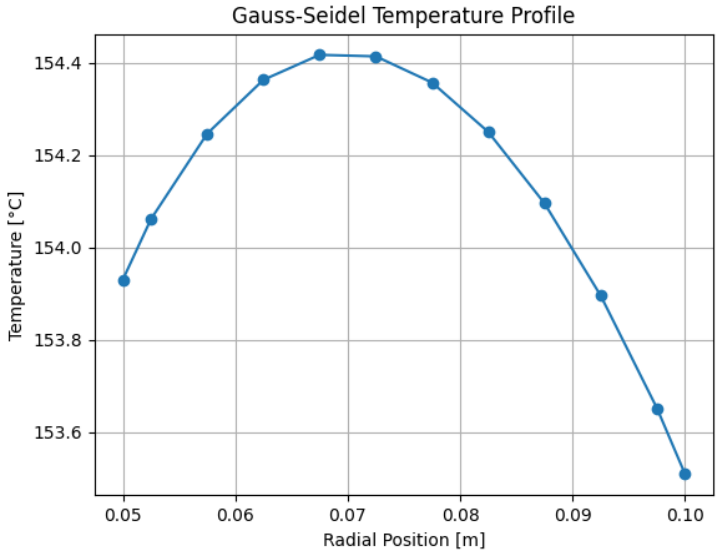

Figure2. Temperature plot obtained using Gauss-Seidel algorithm.

Table 2. Temperature distribution obtained using Gauss-Seidel algorithm.

Node	Radius [m]	Temperature [°C]
1	0.0500	153.929982
2	0.0525	154.060990
3	0.0575	154.246163
4	0.0625	154.362674
5	0.0675	154.416813
6	0.0725	154.413523
7	0.0775	154.356758
8	0.0825	154.249731
9	0.0875	154.095088
10	0.0925	153.895034
11	0.0975	153.651425
12	0.1000	153.508638

Converged after 2327 iterations.

TDMA

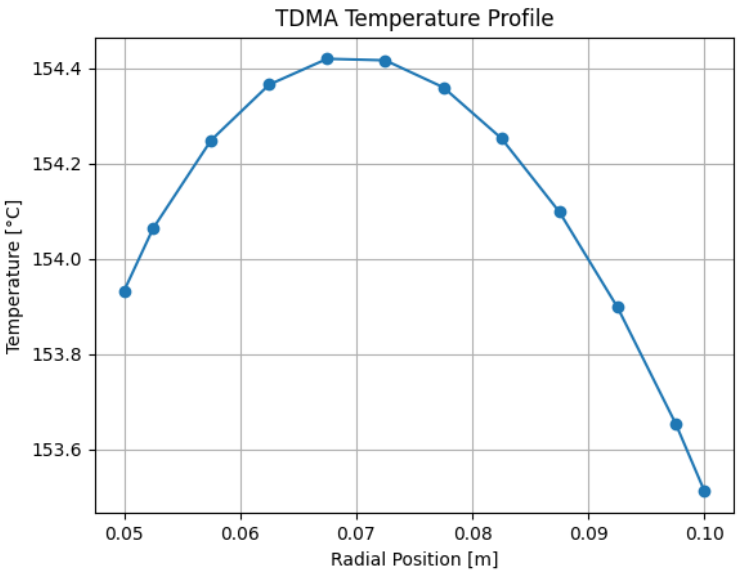

Figure 3. Temperature plot obtained using TDMA algorithm.

Table 3. Temperature distribution obtained using TDMA algorithm.

Node	Radius [m]	Temperature [°C]
1	0.0500	153.932542
2	0.0525	154.063626
3	0.0575	154.248932
4	0.0625	154.365555
5	0.0675	154.419788
6	0.0725	154.416575
7	0.0775	154.359873
8	0.0825	154.252896
9	0.0875	154.098290
10	0.0925	153.898260
11	0.0975	153.654665
12	0.1000	153.511874

Results Comparison

The following plot shows a comparison of the temperature profiles obtained using analytical and numerical methods, Gauss-Seidel and TDMA.

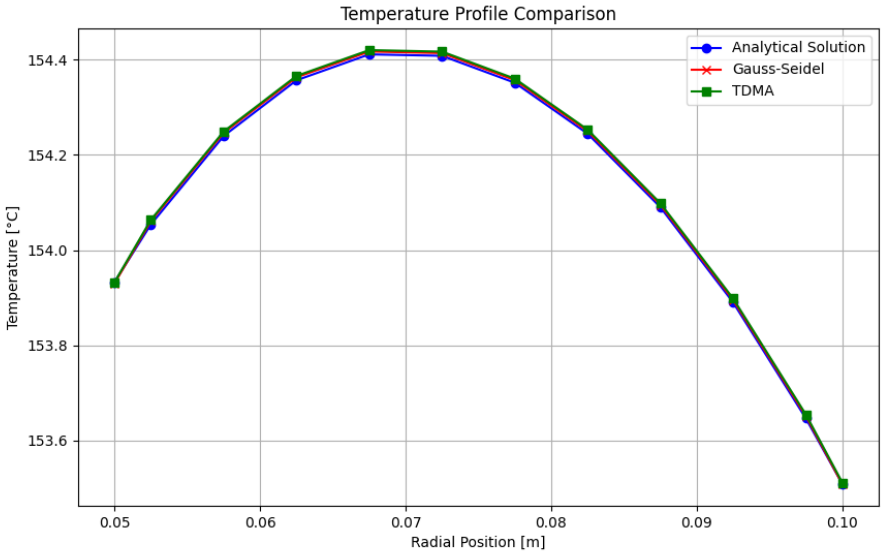

Figure 4. Temperature profile comparison. Analytical, Gauss-Seidel, TDMA.

All three curves closely overlap, indicating that both numerical methods provide results that are in good agreement with the analytical solution.

Validation Check and Mesh Refinement

As mentioned before, the code must be verified by doing a global energy balance that assures that the energy entering and leaving the system is equal to the energy generated internally.

For steady-state conditions:

$$\dot{Q}_{gen} + \dot{Q}_{in} \approx \dot{Q}_{out}$$

where:

$$\dot{Q}_{gen} = q_v \cdot \pi(R_B^2 - R_A^2) \cdot H$$

$$\dot{Q}_{in} = \alpha_a \cdot 2\pi R_A H\left(T_A - T_{R_{int}}\right)$$

$$\dot{Q}_{out} = \alpha_B \cdot 2\pi R_B H\left(T_{R_{ext}} - T_B\right)$$

The results obtained showed that for the analytical and numerical solutions this balance was satisfied.

The error between the analytical and numerical solutions was then calculated to demonstrate how refining the mesh can reduce this discrepancy.

$$ppm\ Error = \frac{\left|T_{analytical} - T_{numerical}\right|}{\left|T_{analytical}\right|} \times 10^6$$

This equation represents the pointwise error in ppm, comparing temperature values from the analytical solution and numerical solutions at each node. Then, the maximum pointwise error across all radial positions was calculated to provide an overall accuracy of the numerical methods.

Simulations were conducted with varying numbers of nodes (N = 10, 50, 100, 500) to assess the effect of mesh density. Results show that as N increases, the error between the numerical and analytical solutions decreases, confirming convergence.

ppm Error			
	N=10	N=50	N=100
Gauss-Seidel	63.570594	2.590153	0.647853
TDMA	63.570611	2.590493	0.649166

Parameter changes

To further analyze the impact of key parameters on the temperature profile, variations in thermal conductivity, internal heat generation, and external fluid properties were explored. These changings help to illustrate how each parameter influences the heat transfer behavior in the cylindrical wall.

Thermal Conductivity

By significantly lowering the thermal conductivity, to simulate for example a tube made of PVC, the temperature profile shows a steeper gradient across the wall. This occurs because a lower thermal conductivity restricts heat flow through the material, resulting in greater resistance to heat transfer.

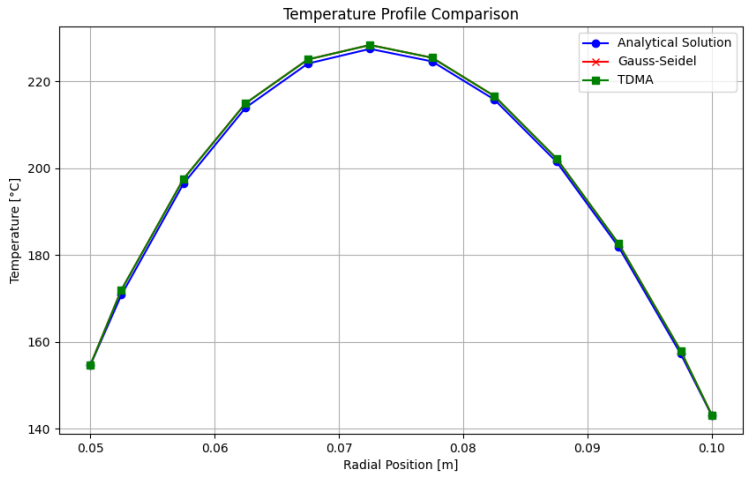

Figure 5. Temperature profile assuming $\lambda = 0.4 \ \ W/mK$.

Internal Heat Generation

Reducing the internal heat generation by half brings the temperature profile closer to linear, highlighting that the internal and external fluids have a greater influence on the temperature distribution across the tube wall. This shift indicates that with less internal heat, heat transfer is more controlled by the boundary conditions than by the internal heat source.

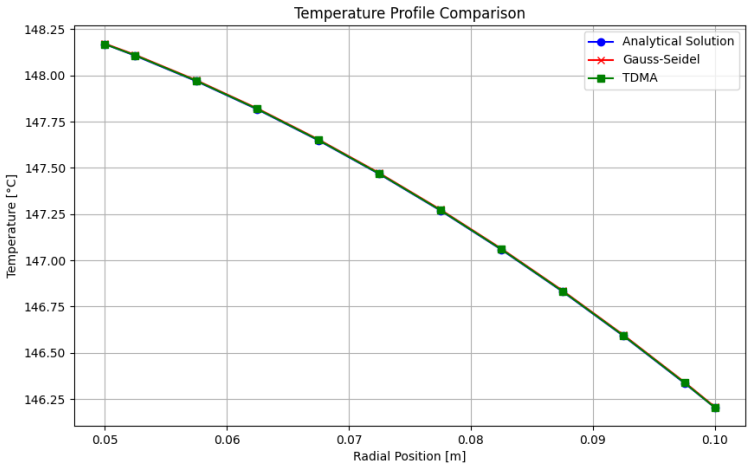

Figure 6. Temperature profile assuming $\dot{q}_v = 50000\ W/m^3$.

External fluid

Increasing the external heat transfer coefficient α_B, to simulate a more effective cooling fluid around the tube, results in a flatter temperature profile. This change reflects how enhancing heat dissipation reduces the overall temperature within the tube wall.

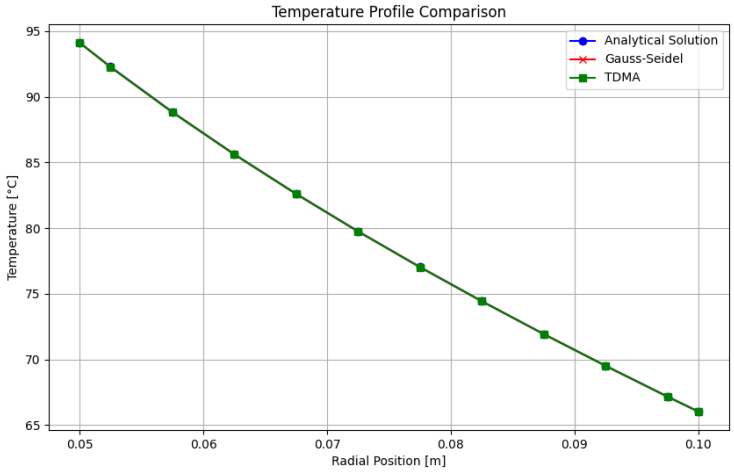

Figure 7. Temperature profile assuming $\alpha_B = 500\ \ W/m^2K$.

Conclusions

This study modeled steady-state, one-dimensional conduction in a cylindrical wall with internal heat generation where numerical results align closely with the analytical solution, confirming the accuracy of the model. Parameter studies demonstrated the effect of thermal conductivity, heat generation, and convection coefficients on the temperature profile.

The TDMA method is recommended for similar problems due to its computational efficiency, while Gauss-Seidel is useful for systems without tridiagonal structure.

Annex 1. Python Code

```
# Simulation of a steady, 1D conduction heat transfer in a cylindrical wall with
internal heat production

import numpy as np
import matplotlib.pyplot as plt
from sympy import symbols, Eq, ln, solve

#### Input data ####

# Physical data
ta=150
Ta=ta+273.15
tb=25
Tb=tb+273.15
Ra=0.05
Rb=0.1
H=2
lambda_=45
alfaa=600
alfab=20
qv=1e5

# Numerical data
N=10
tol=10e-12
tstart=150
Tstart = tstart + 273.15

#### Analytical Solution ####

# Definition of variables
r = symbols('r')
C1, C2 = symbols('C1 C2')

# Solution in cylindrical coordinates
T = -qv / (4 * lambda_) * r**2 + C1 * ln(r) + C2

# Derivative of temperature
dT_dr = T.diff(r)

# Boundary conditions
# R=Ra
eq1 = Eq(-lambda_ * dT_dr.subs(r, Ra), alfaa * (Ta - T.subs(r, Ra)))
# R=Rb
eq2 = Eq(-lambda_ * dT_dr.subs(r, Rb), alfab * (T.subs(r, Rb) - Tb))

# Find solution
```

```
solutions = solve((eq1, eq2), (C1, C2))
T_sol = T.subs(solutions)

# Definition of the position of the nodes
Dr_a=(Rb-Ra)/N

rcv_a=np.zeros(N+1)
rP_a=np.zeros(N+2)

for i in range (0, N+1, 1):
    rcv_a[i]=Ra+(i)*Dr_a

rP_a[0]=Ra
for i in range (1, N+1, 1):
    rP_a[i] = (rcv_a[i]+rcv_a[i-1])/2
rP_a[N+1]=Rb


T_a = np.zeros(N + 2)
def T_sol_numeric(r_value):
    return T_sol.subs(r, r_value).evalf()

# Temperature at each point
for i in range(0, N + 2, 1):
    T_a[i] = T_sol_numeric(rP_a[i]) - 273.15

# Temperature distribution Analytical
print("Analytical Temperature Distribution:")
for i in range(0, N + 2, 1):
    print(f"At r = {rP_a[i]:.4f} T: {T_a[i]:.6f} °C")

# Plotting the temperature profile Analytical
plt.plot(rP_a, T_a, marker='o')
plt.xlabel('Radial Position [m]')
plt.ylabel('Temperature [°C]')
plt.title('Analytical Temperature Profile')
plt.grid()
plt.show()

#### Numerical Solution ####

# Discretization of the mesh
Dr=(Rb-Ra)/N

rcv=np.zeros(N+1)
rP=np.zeros(N+2)
vP=np.zeros(N+1)

for i in range (0, N+1, 1):
    rcv[i]=Ra+(i)*Dr
```

```
rP[0]=Ra
for i in range (1, N+1, 1):
    rP[i] = (rcv[i]+rcv[i-1])/2
rP[N+1]=Rb

for i in range (1, N+1, 1):
    vP[i]= np.pi*(rcv[i]**2-rcv[i-1]**2)*H

# Initial temperature guess
Tg=np.zeros(N+2)
for i in range (0, N+2, 1):
    Tg[i]=Tstart

# Discretization of coefficients
aw = np.zeros (N + 2)
ae = np.zeros (N + 2)
bp = np.zeros (N + 2)
ap = np.zeros (N + 2)

# Internal nodes
for i in range (1, N+1, 1):
    aw[i] = lambda_*2*np.pi*rcv[i-1]*H/(rP[i]-rP[i-1])
    ae[i] = lambda_*2*np.pi*rcv[i]*H/(rP[i+1]-rP[i])
    ap[i] = aw[i] + ae[i]
    bp[i] = qv*vP[i]

# Boundary nodes
aw[0] = 0
ae[0] = lambda_/(rP[1]-rP[0])
ap[0] = ae[0] + alfaa
bp[0] = alfaa*Ta

aw[N+1] = lambda_/(rP[N+1]-rP[N])
ae[N+1] = 0
ap[N+1] = aw[N+1] + alfab
bp[N+1] = alfab*Tb

#### Resolution with Gauss-Seidel ####

T_ng = np.zeros(N + 2)
convergence = tol + 1
iteration = 0

while convergence > tol:
    convergence = 0
    for i in range(0, N + 2, 1):
        if i == 0:  # Boundary condition at Ra
            T_ng[i] = (ae[i] * Tg[i + 1] + bp[i]) / ap[i]
        elif i == N + 1:  # Boundary condition at Rb
```

```
            T_ng[i] = (aw[i] * T_ng[i - 1] + bp[i]) / ap[i]
        else:  # Internal nodes
            T_ng[i] = (ae[i] * Tg[i + 1] + aw[i] * T_ng[i - 1] + bp[i]) / ap[i]

        # Convergence check
        convergence = max(convergence, abs(T_ng[i] - Tg[i]))

    # Update the old temperature
    Tg[:] = T_ng[:]

    iteration += 1

# Temperature distribution Gauss-Seidel
print ('Gauss-Seidel temperature distribution:')
for i in range(0, N + 2, 1):
    print(f"At r = {rP[i]:.4f} T: {T_ng[i]-273.15:.6f} °C")
print(f'Converged after {iteration} iterations.')

# Plotting the temperature profile Gauss-Seidel
plt.plot(rP, T_ng - 273.15, marker='o')
plt.xlabel('Radial Position [m]')
plt.ylabel('Temperature [°C]')
plt.title('Gauss-Seidel Temperature Profile')
plt.grid()
plt.show()

#### Resolution with TDMA ####

T_nt = np.zeros (N+2)
P = np.zeros (N+2)
R = np.zeros (N+2)

# Evaluation of the triangular matrix
for i in range(0, N + 2, 1):
    if i == 0:
        P[i] = ae[i] / ap[i]
        R[i] = bp[i] / ap[i]
    else:
        P[i] = ae[i] / (ap[i] - aw[i] * P[i - 1])
        R[i] = (bp[i] + aw[i] * R[i - 1]) / (ap[i] - aw[i] * P[i - 1])

# Temperature of each node
T_nt[N+1] = R[N+1]
for i in range(N, -1, -1):
    T_nt[i] = P[i] * T_nt[i + 1] + R[i]

# Temperature distribution TDMA
print('TDMA temperature distribution:')
for i in range(0, N + 2):
    print(f"At r = {rP[i]:.4f} T: {T_nt[i] - 273.15:.6f} °C")
```

```
# Plotting the temperature profile TDMA
plt.plot(rP, T_nt - 273.15, marker='o')
plt.xlabel('Radial Position [m]')
plt.ylabel('Temperature [°C]')
plt.title('TDMA Temperature Profile')
plt.grid()
plt.show()

# Plotting the temperature profiles for both analytical and numericals solutions
plt.figure(figsize=(10, 6))
plt.plot(rP_a, T_a, marker='o', label='Analytical Solution', color='blue')
plt.plot(rP, T_ng - 273.15, marker='x', label='Gauss-Seidel', color='red')
plt.plot(rP, T_nt - 273.15, marker='s', label='TDMA', color='green')

plt.xlabel('Radial Position [m]')
plt.ylabel('Temperature [°C]')
plt.title('Temperature Profile Comparison')
plt.grid()
plt.legend()
plt.show()

#### Global Balances ####

#Analytical
Qg_a=qv*np.pi*H*(Rb**2-Ra**2)
Qin_a=alfaa*2*np.pi*Ra*H*(Ta-T_a[0])
Qout_a=alfab*2*np.pi*Rb*H*(T_a[N+1]-Tb)
print (Qg_a+Qin_a ,'=', Qout_a)

#Gauss-Seidel
Qg_ng=qv*np.pi*H*(Rb**2-Ra**2)
Qin_ng=alfaa*2*np.pi*Ra*H*(Ta-T_ng[0])
Qout_ng=alfab*2*np.pi*Rb*H*(T_ng[N+1]-Tb)
print (Qg_ng+Qin_ng ,'=', Qout_ng)

#TDMA
Qg_nt=qv*np.pi*H*(Rb**2-Ra**2)
Qin_nt=alfaa*2*np.pi*Ra*H*(Ta-T_nt[0])
Qout_nt=alfab*2*np.pi*Rb*H*(T_nt[N+1]-Tb)
print (Qg_nt+Qin_nt ,'=', Qout_nt)

# Pointwise errors comparison with the analytical solution
# Calculate ppm errors
pointwise_errors_ng_ppm = (np.abs(T_a - (T_ng - 273.15)) / np.abs(T_a)) * 1e6  #
Gauss-Seidel
pointwise_errors_nt_ppm = (np.abs(T_a - (T_nt - 273.15)) / np.abs(T_a)) * 1e6  #
TDMA

# Maximum ppm errors for both methods
```

```python
max_error_ng_ppm = np.max(pointwise_errors_ng_ppm)
max_error_nt_ppm = np.max(pointwise_errors_nt_ppm)

# Print maximum errors
print(f'\nMaximum Gauss-Seidel Error: {max_error_ng_ppm:.6f} ppm')
print(f'Maximum TDMA Error: {max_error_nt_ppm:.6f} ppm')
```