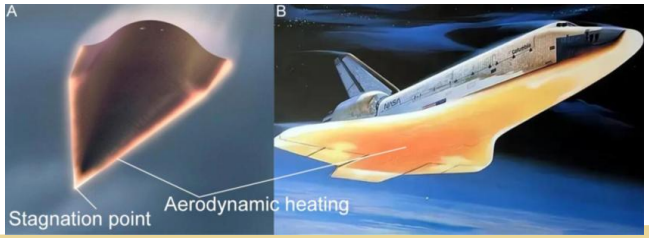
Thermal Distribution in hypersonic leading edge

2D Transient Conduction Numerical Analysis

Rithwik Shankar Raj



Outline

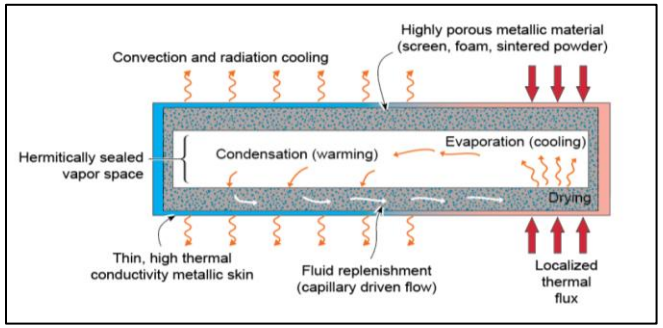
- Project Objective and expected outcomes
- Schematic Domain and Boundary Conditions
- Governing Equations
- Results
- Conclusions

Project Overview and Objective

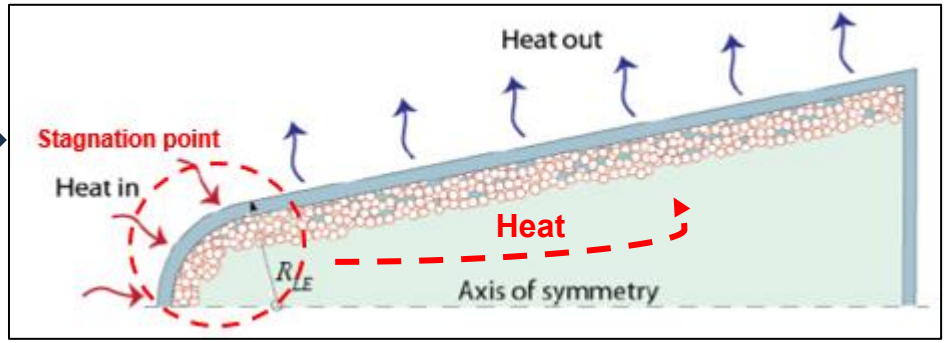
- Aerodynamic heating at the leading edges cause overheating and may damage interior components in hypersonic vehicles.
- **Objective:** Analyze the temperature distribution of a heat-pipe based TPS (Thermal Protection System) to redistribute heat from high-flux areas.

Expected outcomes:

- A 2D transient heat conduction model for the leading edge.
- Optimize thermal behavior by comparison of different materials and geometry.



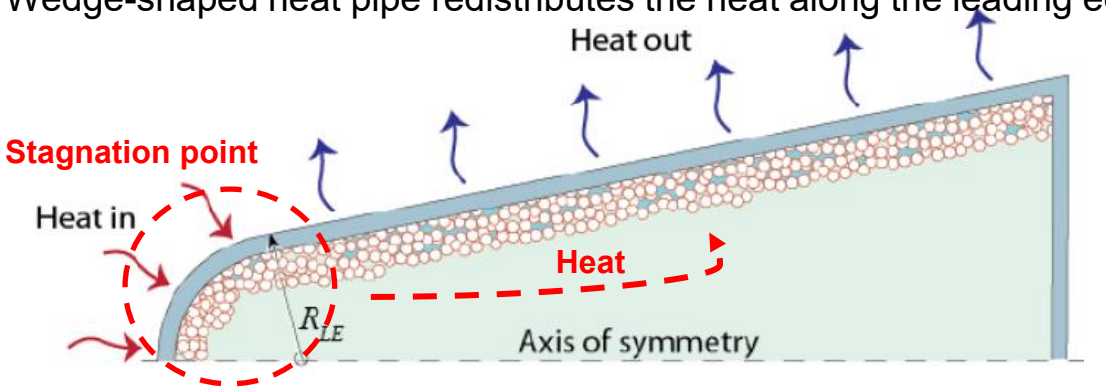
Cross-section of a cylindrical heat pipe



Wedge-shaped heat pipe (leading edge)

Schematic representation of domain and boundary conditions

- Wedge-shaped heat pipe redistributes the heat along the leading edge.



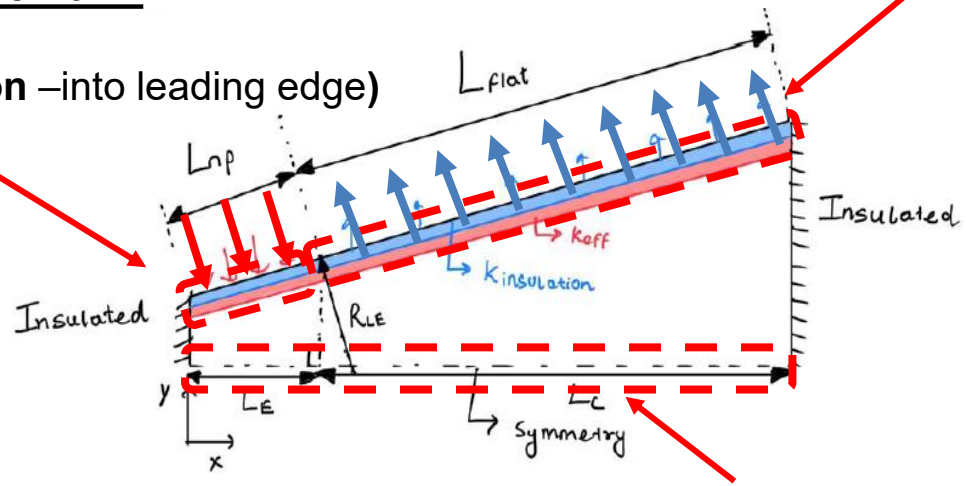
- **Heat is removed and redistributed from the stagnation point to lower temperature areas through heat pipes via evaporation-condensation mechanisms.**
- **This effect reduce thermal stresses and avoids hot spots on the stagnation point.**

$$q_{st} = 1.83x10^{-8}(\frac{\rho_\infty}{R_{LE}})^{1/2}{V_\infty}^3$$

Radiation and Convection (out of top Surface)

Physical domain:

(**Conduction** –into leading edge)



$$\left.\frac{\partial T}{\partial y}\right|_{y=0} = 0$$

Boundary Conditions:

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = 0$$

$$\left.\frac{\partial T}{\partial x}\right|_{x=L} = 0$$

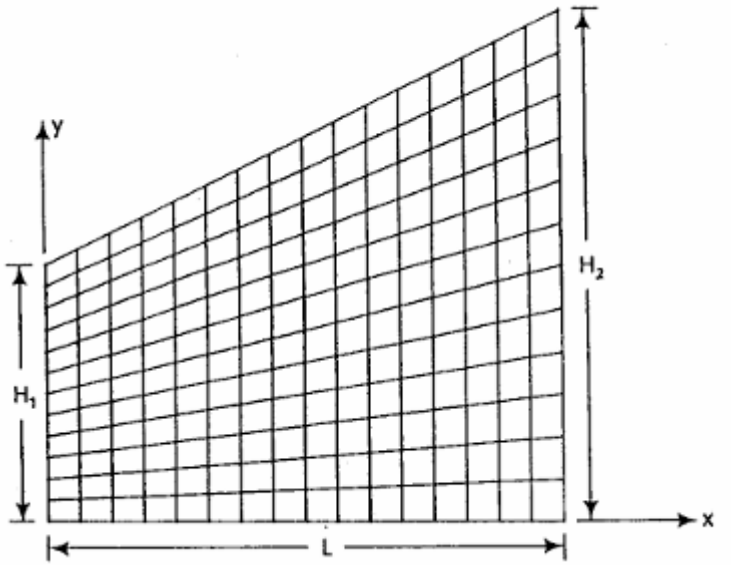
$$-k\left.\frac{\partial T}{\partial y}\right|(top\ surface) = \begin{cases} -{q_1}'', & x < L_{np} \\ {q_2}'', & L_{np} \le x \le L_{flat} \end{cases}$$

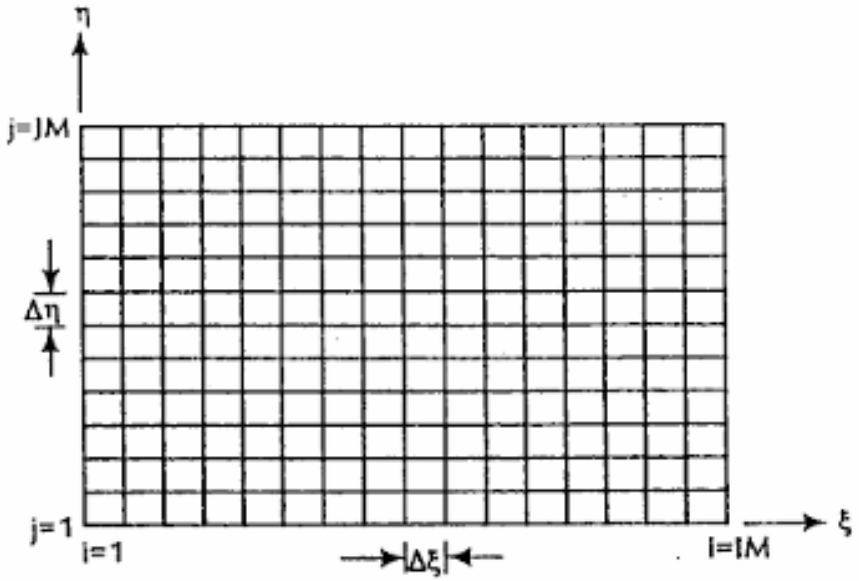
$$where\ {q_1}'' = 460\ W/cm^2$$

$$and\ {q_2}'' = \frac{T_{boundary} - T_\infty}{R_{insulation,eff} + (\frac{R_{conv} * R_{rad}}{R_{conv} + R_{rad}})}$$

Transformation of the Grid

Physical Space → Computational Domain





Linear Approximation:

$$\xi = x$$

$$\eta = \frac{y}{H_1 + (H_1 - H_1)\frac{x}{L}}$$

Energy Balance

Inner Nodes: (i,j)

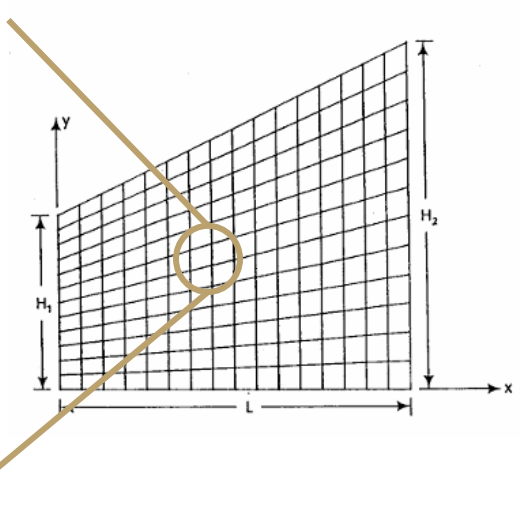
$$\kappa\frac{\Delta x}{\Delta y}\left(T_{i-1,j}-T_{i,j}\right)+\kappa\frac{\Delta x}{\Delta y}\left(T_{i+1,j}-T_{i,j}\right)+\kappa\frac{\Delta y}{\Delta x}\left(T_{i,j-1}-T_{i,j}\right)+\kappa\frac{\Delta y}{\Delta x}\left(T_{i,j+1}-T_{i,j}\right)=\rho c\Delta x\Delta y\frac{\partial T}{\partial t}$$

$$\frac{dT}{dt}=\frac{\alpha}{\Delta y^2}\left(T_{i-1,j}-T_{i,j}+T_{i+1,j}-T_{i,j}\right)+\frac{\alpha}{\Delta x^2}\left(T_{i,j-1}-T_{i,j}+T_{i,j+1}-T_{i,j}\right)$$

$$\text{if } d_1=\frac{2\alpha\Delta t}{\Delta x^2} \text{ and } d_2=\frac{2\alpha\Delta t}{\Delta y^2}$$

$$T_{32}^{n+1}=T_{32}^{n}+\frac{d_2}{2}\left(T_{i-1,j}{}^{n+1}-T_{i,j}{}^{n+1}+T_{i+1,j}{}^{n+1}-T_{i,j}{}^{n+1}\right)+\frac{d_1}{2}\left(T_{i,j-1}{}^{n+1}-T_{i,j}{}^{n+1}+T_{i,j+1}{}^{n+1}-T_{i,j}{}^{n+1}\right)$$

$$T_{i,j}^{n+1}(1+d_1+d_2)+T_{i-1,j}^{n+1}\left(\frac{-d_2}{2}\right)+T_{i+1,j}^{n+1}\left(\frac{-d_2}{2}\right)+T_{i,j-1}^{n+1}\left(\frac{-d_1}{2}\right)+T_{i,j+1}^{n+1}\left(\frac{-d_1}{2}\right)=T_{i,j}^{n}$$



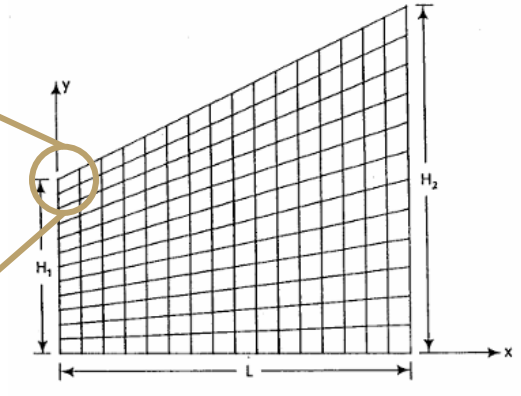
Top wall: (H,1)

$$\kappa\frac{\Delta x/2}{\Delta y}\left(T_{H-1,1}-T_{H,1}\right)+\kappa\frac{\Delta y/2}{\Delta x}\left(T_{H,2}-T_{H,1}\right)+q''\frac{\Delta x}{2}=\rho c\frac{\Delta x}{2}\frac{\Delta y}{2}\frac{\partial T}{\partial t}$$

$$\frac{dT}{dt}=\frac{2\alpha}{\Delta y^2}\left(T_{H-1,1}-T_{H,1}\right)+\frac{2\alpha}{\Delta x^2}\left(T_{H,2}-T_{H,1}\right)+\frac{2q''}{\rho c\Delta y}$$

$$T_{H,1}^{n+1}=T_{H,1}^{n}+d_2\left(T_{H-1,1}{}^{n+1}-T_{H,1}^{n+1}\right)+d_1\left(T^{n+1}{}_{H,2}-T_{H,1}^{n+1}\right)+\frac{2q''\Delta t}{\rho c\Delta y}$$

$$T_{H,1}^{n+1}(1+d_1+d_2)+T_{H-1,1}^{n+1}(-d_2)+T_{H,2}^{n+1}(-d_1)=T_{H,1}^{n}+\frac{2q''\Delta t}{\rho c\Delta y}$$



Finite Difference Formulations

Gauss-Seidel Successive Over-Relaxation (SOR) Equation:

$$T_{i,j}^{n+1} = \frac{\omega\left[A\left(T_{i+1,j}^{n+1} + T_{i-1,j}^{n+1}\right) + B\left(T_{i,j+1}^{n+1} + T_{i,j-1}^{n+1}\right) + T_{i,j}^{n}\right]}{C} + (1-\omega)T_{i,j}^{n+1}$$

Where,

ω is the relaxation factor

$A=\frac{\alpha\Delta t}{\Delta x^2}$ is the contribution from neighbors in x-direction

$B=\frac{\alpha\Delta t}{\Delta y^2}$ is the contribution from neighbors in y-direction

C=1+2A+2B is the normalization coefficient

Boundary Conditions:

$$-k\frac{\partial T}{\partial y}\Big|(top\ surface) = \begin{cases} -q_1", & x < L_{np} \\ q_2", & L_{np} \le x \le L_{flat} \end{cases}$$

$$where\ q_1" = 460\ W/cm^2$$

$$and\ q_2" = \frac{T_{boundary} - T_\infty}{R_{insulation,eff} + \left(\frac{R_{conv} * R_{rad}}{R_{conv} + R_{rad}}\right)}$$

$$\left.\frac{\partial T}{\partial x}\right|_{x=0} = 0 \qquad \left.\frac{\partial T}{\partial y}\right|_{y=0} = 0$$

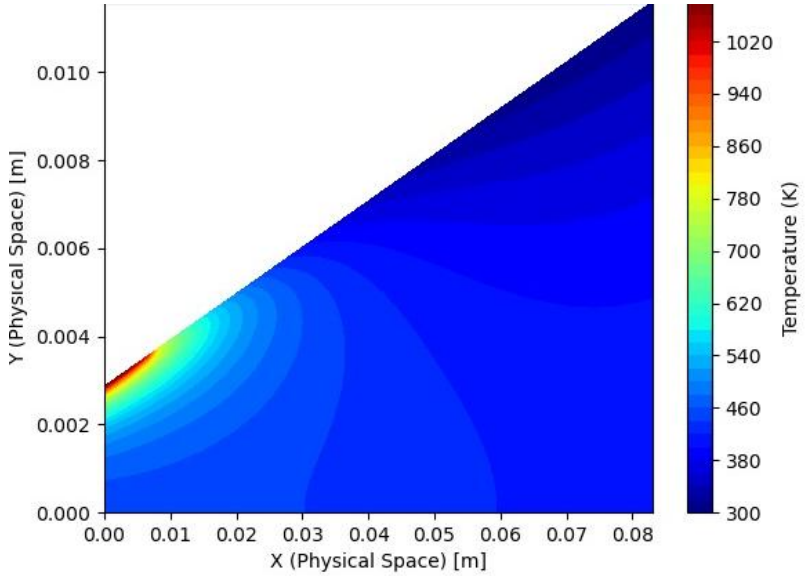
$$\left.\frac{\partial T}{\partial x}\right|_{x=L} = 0$$

Laplace Smoothing:

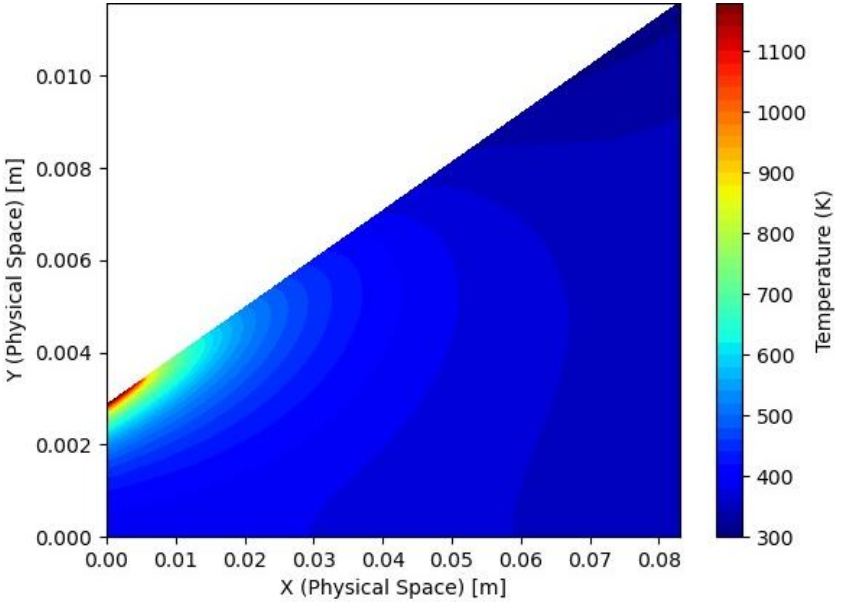
$$T_{i,j}^{n+1} = \frac{1}{4}\left(T_{i+1,j}^{n} + T_{i-1,j}^{n} + T_{i,j+1}^{n} + T_{i,j-1}^{n}\right)$$

Comparing the two methods

Energy Balance



FDM

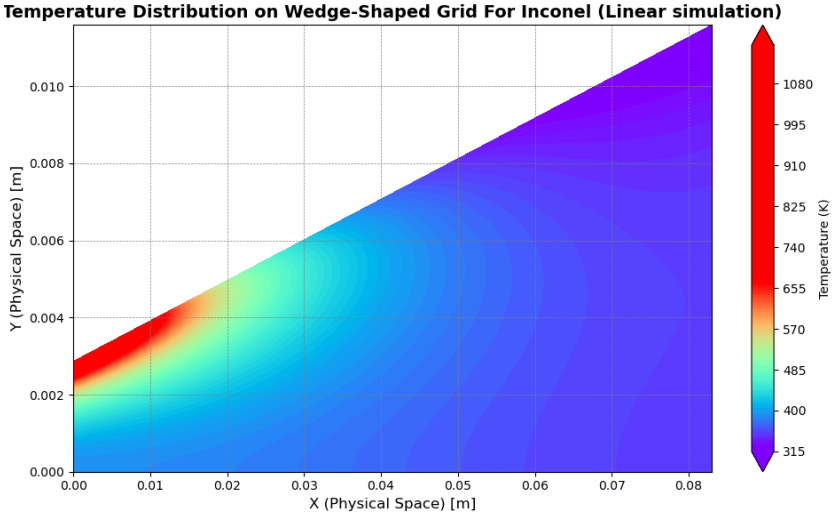


Non-linearity

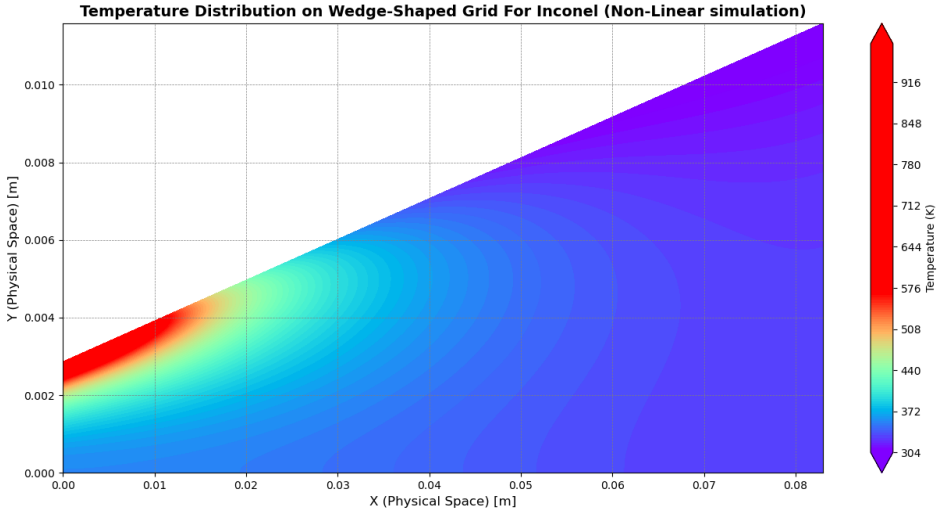
- Conductivity and Specific heat for Inconel 718 are Temperature Dependent!
- Becomes a non-linear problem
- Convergence criteria to be met at every time step!!

Alloy	Equation (T in °C, k in W/mK)
IN718	$k = \begin{cases} 10.803 + 0.0162949\,T - 2.53206 \times 10^{-7}\,T^2 & ,T < 1093°C \\ 28.31 & T \geq 1093°C \end{cases}$

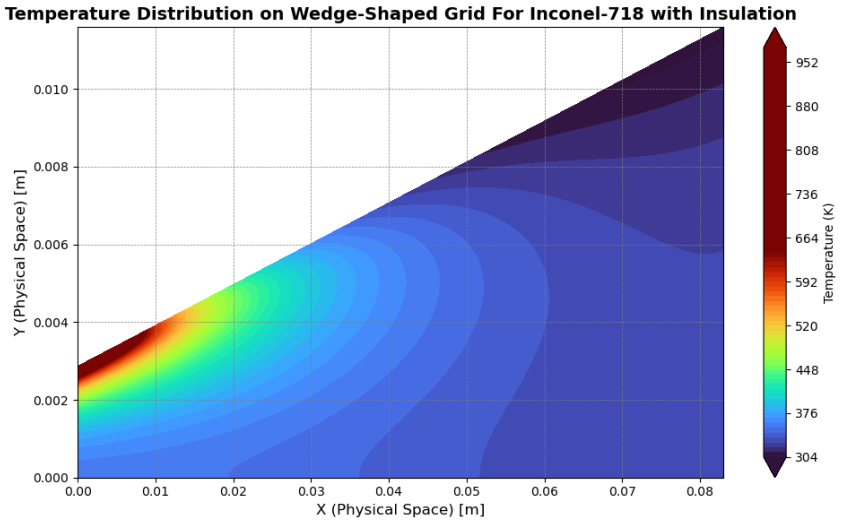
Linear

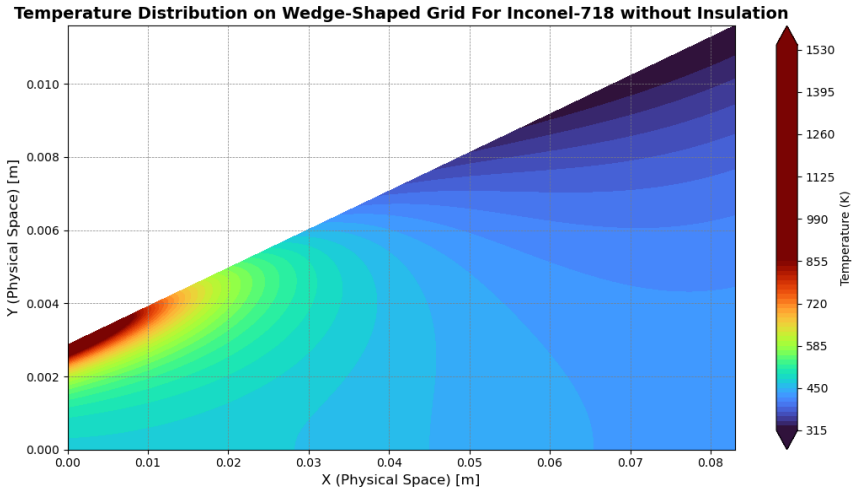


Non-Linear



Insulation





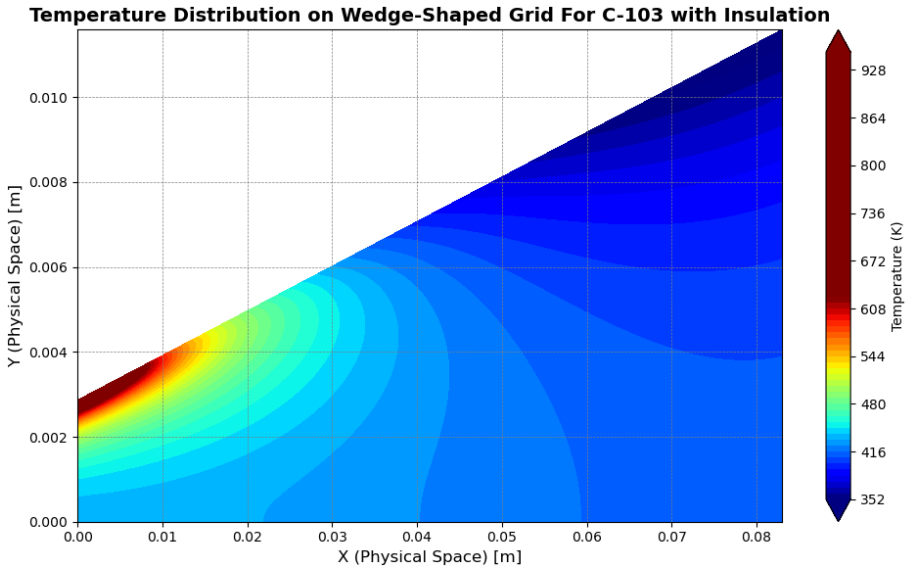
- Comparing max temperatures with and without insulation

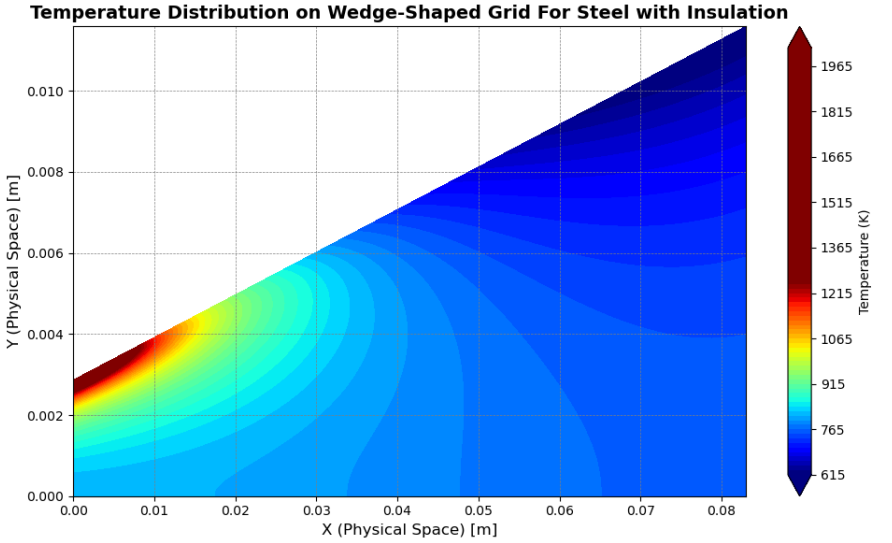
Max Temperature Without Insulation (K)	Max Temperature With Insulation (K)	Reduction (%)	Material
1554	981	37	INC-718
1325	960	28	C-103
2465	2027	18	AISI-304

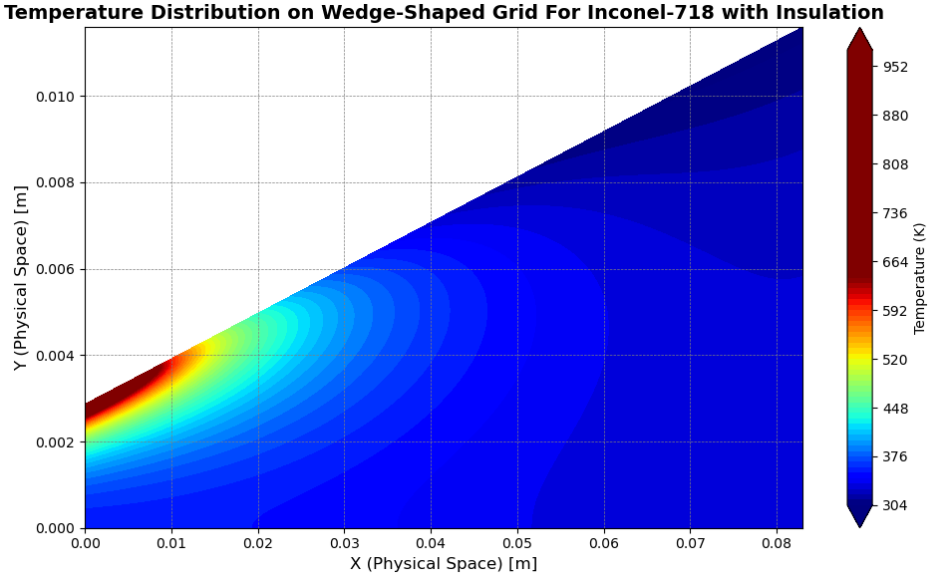
- Comparing max temperatures for varying insulation thickness (INC-718)

Thickness (mm)	Max Temperature (K)
1	1300
2	1100
3	981

Trying different materials…

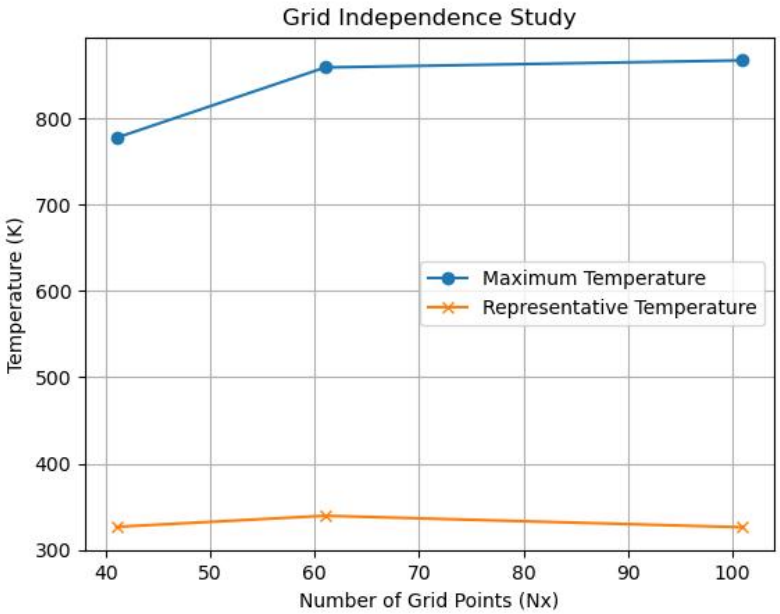


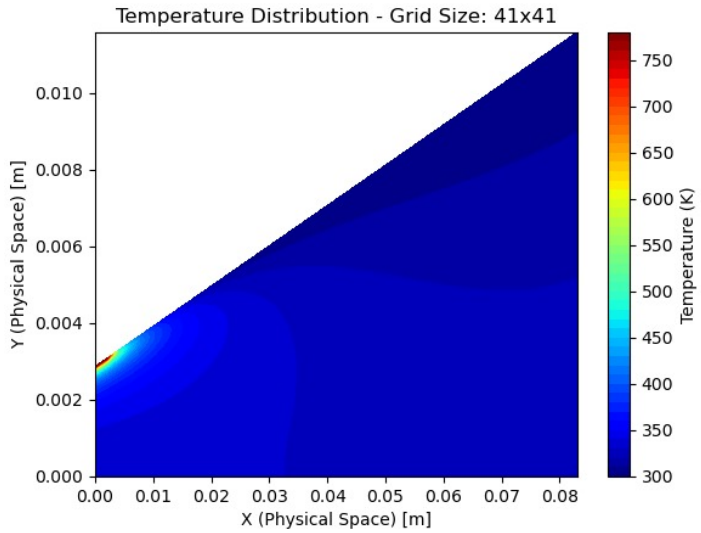


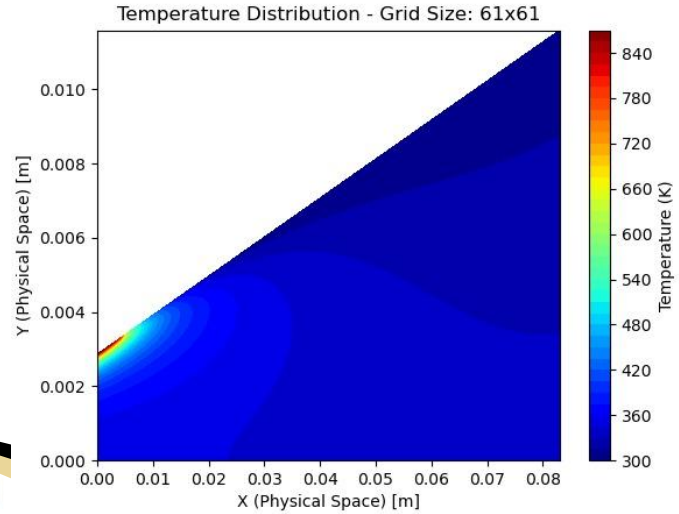


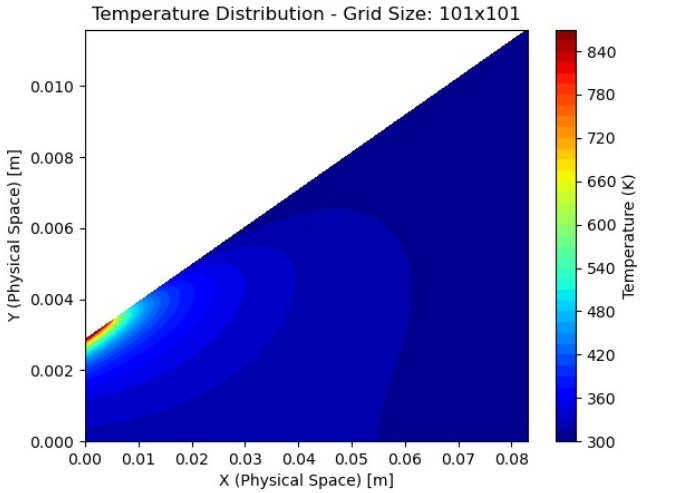
IN718	$C_p = \begin{cases} 526.727 - 0.0476148\ \mathrm{T} - 8.92138 \times \frac{10^6}{(273.15+\mathrm{T})^2} & T \le 507°\mathrm{C} \\ 2.71 \frac{(-133.249+\mathrm{T})(399965,-711.241\ \mathrm{T}+\mathrm{T}^2)}{(273.15+\mathrm{T})^2} & 507 < T \le 787°\mathrm{C} \\ 725 & T > 787°\mathrm{C} \end{cases}$	[138]
C-103	$C_p = \begin{cases} 269.053 + 0.0323504\ \mathrm{T} + 9.79386 \times 10^{-6}\ \mathrm{T}^2 & \mathrm{T} \le 1000°\mathrm{C} \\ 311 & \mathrm{T} > 1000°\mathrm{C} \end{cases}$	[139]
IN718	$k = \begin{cases} 10.803 + 0.0162949\ T - 2.53206 \times 10^{-7}\ T^2 & ,T < 1093°\mathrm{C} \\ 28.31 & T \ge 1093°\mathrm{C} \end{cases}$	[143]
C-103	$k = 43.2$	[139]

Grid Independence Test

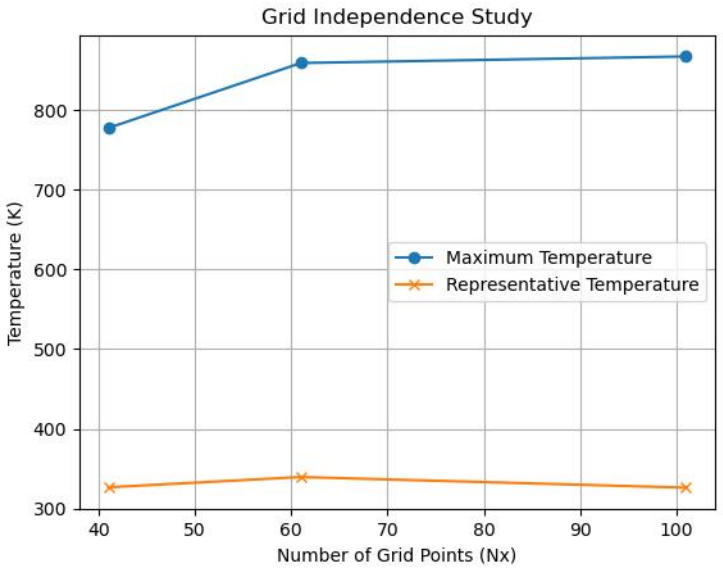








Grid and Time Independence



$$L_2 \text{ Error} = \frac{\sqrt{\sum_{i=1}^{N}\left(T_i^{n+1} - T_i^{n}\right)^2}}{\sqrt{\sum_{i=1}^{N}\left(T_i^{n+1}\right)^2}}$$

$$\text{Error} = \max(|T_{\text{new}} - T_{\text{old}}|)$$

- Grid independence achieved for a grid size of 61 x 61
- Normalized L2 Norm and absolute temperature difference are used to analyze time independence
- Since an implicit method is used, a larger time step was possible.
- However, the implicit method is still not unconditionally stable since the problem is non-linear.
- Therefore, for dt=0.001, the L2 Error is 1e-9 and the absolute error reaches 1e-4

Conclusion

What was achieved:

- C-103 and INC-718 material was the best.
- Steel couldn’t withstand the high heat fluxes
- Achieved grid independence!

Scope for improvements:

- Curved geometry
- Heat pipe not considered

References

[1] Moran, M. J., and Shapiro, H. N., *Fundamentals of Engineering Thermodynamics*, 9th ed., Wiley, Hoboken, New Jersey, 2021. Published May 13, 2021.

[2] Kasen, S. D., "Analysis and Design of Heat Pipe Based Thermal Protection Systems for Hypersonic Vehicles," Ph.D. thesis, Georgia Institute of Technology, Atlanta, Georgia, 2013. Ph.D. Dissertation.

[3] Metals, S., "Special Metals INCONEL® Alloy 600 - MatWeb," , 2023. URL https://www.matweb.com/search/datasheet.aspx?matguid=022a4773b91543d6a181cb2efcc6801d, retrieved from MatWeb.

[4] MatWeb, "Niobium C-103 (WC 103; 89Nb-10Hf-1Ti), Cold-Rolled - MatWeb," , 2023. URL https://www.matweb.com/search/datasheet.aspx?matguid=c7dc39be19554f77940d330fd78a3560, retrieved from MatWeb.

[5] MatWeb, "AISI 1020 Steel - MatWeb," , 2023. URL https://www.matweb.com/search/datasheet.aspx?matguid=b93598c78b2a48d08e4dc4f4c97a6b3b, retrieved from MatWeb.

[6] Metals, S., "Inconel 600 Alloy (UNS N06600) - Composition, Properties, and Uses," *AZoM - Materials Science*, 2023. URL https://www.azom.com/article.aspx?ArticleID=12822.

[7] Corporation, M., "Materion's high-strength niobium C-103 alloy," *Materion Technical Library*, 2023. URL https://materion.com/products/niobium-alloys/niobium-c103.

[8] Toolbox, E., "Emissivity Coefficients of Metals," , 2023. URL https://www.engineeringtoolbox.com/emissivity-coefficients-d_447.html.

Sample Python Script

```python
import numpy as np
import matplotlib.pyplot as plt

##############Dafining Thermophysical Properties of Stainless Steel AISI 304#############################
k_steel=16.2 #W/mK
alpha_steel=1.67e-5 #m^2/s

##############Dafining Thermophysical Properties of Inconel 718#########################################
#Define thermophysical property functions
def Cp_inc(T):
    if T<=(507+273):
        return 526.727-0.04761488*(T-273)-8.92138e6/(T**2)
    elif (507+273)<T<=(787+273):
        return 2.71*(-133.249+T-273)*(399965.0-711.24*(T-273)+(T-273)**2)/(T**2)
    else:
        return 725

def k_inc(T):
    if T<=(1093+273):
        return 10.803+0.0162949*(T-273)-2.53206*10**(-7)*(T-273)**2
    else:
        return 28.31

rho_inc=8190

def alpha_inc(T):
    return k_inc(T)/(rho_inc*Cp_inc(T))

#Average temperature
T_avg=700  #Kelvin
Cp_avg=Cp_inc(T_avg)  #Specific heat
k_avg=k_inc(T_avg)  #Thermal conductivity
alpha_avg=alpha_inc(T_avg)  #Thermal diffusivity

##############Dafining Thermophysical Properties of C-103#########################################
k_c103=43.2 #W/m-K
def Cp_c103(T):
    if T<=(1000+273):
        return 269.053+0.0323504*(T-273)+9.79386e-6*(T-273)**2
    else:
        return 311

rho_c103=8850 #Kg/m^3

def alpha_c103(T):
    return k_c103(T)/(rho_c103*Cp_c103(T))

#################Effective Conductivity of the Insulation#########################################
k_eff=50 #W/m-K


# Problem Setup
Lx, Ly=0.083, 0.083  #Dimensions of the rectangular grid
Nx, Ny=61, 61  #Number of grid points
dx, dy=Lx/(Nx-1), Ly/(Ny-1)  #Grid spacing

#Initialize temperature grid
T=np.ones((Ny, Nx))*300  #Initial temperature (K)
T_new=T.copy()

#Time-stepping parameters
dt=0.001  #Time step
tol=1e-3  #Convergence tolerance
max_iter=200000  #Maximum iterations

#Physical boundary and geometry for wedge transformation
R=3e-3
angle=6
L_flat=82.3e-3
x_flux=Lx/np.cos(np.radians(angle))-L_flat
H1=R*np.cos(np.radians(angle))-x_flux*np.sin(np.radians(angle))
H2=H1+(x_flux+L_flat)*np.sin(np.radians(angle))

#Create wedge-shaped grid for visualization
x_physical=np.linspace(0, Lx, Nx)
y_physical=np.zeros((Ny, Nx))
for i in range(Ny):
    eta = i/(Ny-1)
    y_physical[i, :]=eta*(H1+x_physical*(H2-H1)/Lx)

#Boundary conditions
q_flux_in=4.6e6  #Heat flux in W/m^2
h=12  #Convection coefficient in W/m^2-K
epsilon=0.95  #Emissivity
sigma=5.67e-8  #Stefan-Boltzmann constant
T_infinity=293  #Ambient temperature in K
omega=1.4 #Relaxation factor

k_insulation1=0.01  #W/m-K (thermal conductivity of insulation layer 1)
k_insulation2=0.16  #W/m-K (thermal conductivity of insulation layer 2)
thickness_insulation1=0.003  #m (thickness of insulation layer 1)
thickness_insulation2=0.003  #m (thickness of insulation layer 2)

#Solve the heat conduction on rectangular grid
for iter in range(max_iter):
    T_old=T.copy()
    #Update interior points
    for i in range(1,Ny-1):
        for j in range(1,Nx-1):
            #Coefficients for the impkicit schema
            A=alpha_inc(T[i,j])*dt/dx**2
            B=alpha_inc(T[i,j])*dt/dy**2
            C=1+2*A+2*B
            T_new[i, j]=(omega*(A*(T_new[i+1, j]+T_new[i-1, j])+B*(T_new[i, j+1]+T_new[i, j-1])+T[i, j])/C+(1-omega)*T_new[i, j])

    #Boundary conditions
    #Left and right boundaries (adiabatic: zero gradient)
    T_new[:, 0]=T_new[:, 1]
    T_new[:, -1]=T_new[:, -2]

    #Bottom boundary (zero gradient)
    T_new[0, :]=T_new[1, :]

    #Top boundary: Mixed heat flux, convection, and radiation
    split_index=int(0.065 * Nx)  #Assuming 7% of the top boundary for heat flux
    for j in range(split_index):
        #Calculate thermal resistances for the insulation layers
        #R_insulation1=thickness_insulation1/k_insulation1
        #R_insulation2=thickness_insulation2/k_insulation2

        #Total resistance through the insulation layers
        #R_total_insulation=R_insulation1+R_insulation2

        #Heat flux through the insulation layers
        #q_flux=q_flux_in/(1+R_total_insulation)

        #Update top boundary temperature
        T_new[-1, j]=T[-2, j]+(q_flux_in*dy/k_inc(T[-2, j]))

    for j in range(split_index, Nx):
        #Estimate initial surface temperature
        T_surf=T[-1, j]

        #Stabilization loop for nonlinear terms
        for _ in range(10):  #Maximum stabilization iterations
            #Thermal resistances
            R_conv=1/h  #Convective resistance
            R_rad=1/(epsilon*sigma*(T_surf**2+T_infinity**2)*(T_surf+T_infinity))  #Radiative resistance
            R_total_boundary=R_conv*R_rad/(R_conv+R_rad)  #Combined resistance for convection and radiation

            #Effective resistance including insulation layers
            #R_insulation1=thickness_insulation1/k_insulation1
            #R_insulation2=thickness_insulation2/k_insulation2
            #R_total=R_total_boundary+R_insulation1+R_insulation2
            R_total=R_total_boundary

            #Calculate heat flux through the entire system
            q_flux=(T_surf-T_infinity)/R_total

            #Update surface temperature
            T_surf_new=T[-2, j]+(q_flux*dy/k_inc(T[-2, j]))

            #Check for convergence
            if np.abs(T_surf_new-T_surf)<1e-6:
                break

            T_surf=T_surf_new

        #Apply the stabilized surface temperature to the boundary
        T_new[-1, j]=T_surf


    #Convergence check
    error=np.max(np.abs(T_new-T))

    if iter%1000==0:
        print("Iterations", iter, "error", error)

    if error<tol:
        print(f"Converged in {iter} iterations")
        break
    T=T_new.copy()

#Add Laplace smoothing loop
smoothing_iterations = 10000
  #Number of smoothing iterations
T_smoothed=T.copy()  #Start with the final temperature field from the main loop

for _ in range(smoothing_iterations):
    T_smoothed_old = T_smoothed.copy()
```

Sample Python Script

```
    for i in range(1, Ny-1):
        for j in range(1, Nx-1):
            #Laplace smoothing: average of neighboring points
            T_smoothed[i, j]=0.25*(
                T_smoothed_old[i+1, j] +
                T_smoothed_old[i-1, j] +
                T_smoothed_old[i, j+1] +
                T_smoothed_old[i, j-1]
            )

    # Reapply boundary conditions
    T_smoothed[:, 0]=T_smoothed[:, 1]  # Left boundary (adiabatic)
    T_smoothed[:, -1]=T_smoothed[:, -2]  # Right boundary (adiabatic)
    T_smoothed[0, :]=T_smoothed[1, :] # Bottom boundary (fixed temperature)
    T_smoothed[-1, :]=T[-1, :]  # Top boundary (keep the solution)

#Create a meshgrid for x_physical
x_physical_2d, _=np.meshgrid(x_physical, np.linspace(0, 1, Ny))

#Saving the files for plotting
np.savez('Inconel718_No_Insulation.npz', T=T_smoothed, x_dom=x_physical_2d, y_dom=y_physical)


#Plotting the results
plt.figure()
plt.contourf(x_physical_2d, y_physical, T_smoothed, cmap="jet", levels=50)
plt.colorbar(label="Temperature (K)")
plt.xlabel("X (Physical Space) [m]")
plt.ylabel("Y (Physical Space) [m]")
plt.title("Temperature Distribution on Wedge-Shaped Grid For INC-718 without insulation")
plt.show()
```