Assignment 2: PDE e Trasformate di Fourier

Matteo Raffaele Vacca

14 Luglio 2023

Abstract

Nel documento corrente vengono riportati differenti programmi realizzati in python in cui ci si propone di definire:
a) Il profilo di temperatura di un contenitore di acciaio posto a contatto con due materiali a differente temperatura ed eseguire un confronto utilizzando una differente diffusività termica posta nelle stesse condizioni iniziali
b) Utilizzando le trasfomrate di Fourier pulire una immagine dal rumore

1 Introduzione

Per comprendere quali siano state le procedure seguite al fine di raggiungere gli scopi preposti bisogna effettuare una introduzione in merito alla costruzione del problema:

- ***Equazioni Differenziali a Derivate Parziali***: dette PDE con l'acronimo inglese. A differenza delle ODE le PDE presentano nelle proprie funzioni variabili indipendenti tra loro. L'applicazione generale avviene in presenza di funzioni o campi, non necessariamente conservativi. Negli studi fisici sono presenti in maggioranza campi o funzioni a tre dimensioni spaziali ed una dimensione temporale, per cui l'equazione risulta

$$f(x, y, z, t) \vee U(x, y, z, t) \tag{1}$$

il chè implica una variazione di posizione nel tempo e quindi una evoluzione del sistema fisico. La forma più generale a due variabili indipendenti di una PDE risulta essere

$$A\frac{\partial^2 f}{\partial x^2} + 2B\frac{\partial^2 f}{\partial y^2} + C\frac{\partial^2 f}{\partial x \partial y} + D\frac{\partial f}{\partial x} + E\frac{\partial f}{\partial y} + F = 0 \tag{2}$$

dove le lettere in maiuscolo rappresentano funzioni arbitrarie delle variabili x e y. Le PDE vengono classificate in base ai propri discriminanti, che discernono dallo studio delle equazioni di secondo grado delle coniche in algebra lineare, e ciascuna svolge un ruolo:

  - *Ellittica*: Equazioni di Poisson

$$\begin{aligned} d &= AC - B^2 > 0 \\ \nabla^2 f(x) &= -4\rho(x) \end{aligned} \tag{3}$$

– *Iperbolica*: Equazioni d'onda

$$\begin{aligned} d &= AC - B^2 < 0 \\ \nabla^2 f(x,t) &= c^{-2}\frac{\partial^2 f}{\partial t^2} \end{aligned} \tag{4}$$

– *Parabolica*: Equazioni del calore

$$\begin{aligned} d &= AC - B^2 = 0 \\ \nabla^2 f(x,t) &= a\frac{\partial f}{\partial t} \end{aligned} \tag{5}$$

quest'ultimo è il caso di interesse che verrà analizzato utilizzando appropriate condizioni al contorno

▷ ***Equazione del Calore***: Detto H il tasso di flusso del calore, K la conducività termica, T la temperatura e $\boldsymbol{x}$ le variabili posizionali, si può scrivere:

$$H = -K\nabla T(\boldsymbol{x}, t) \tag{6}$$

Mentre Q si ottiene dall'integrale del calore specifico con densità ρ e temperatura dipedenti dalle variabili spaziali

$$Q(t) = \int C\rho(\boldsymbol{x})T(\boldsymbol{x}, t)\, d\boldsymbol{x} \tag{7}$$

Applicando il teorema della divergenza si ottiene l'equazione del calore

$$\frac{\partial T(\boldsymbol{x})}{\partial t} = \frac{K}{C\rho}\nabla^2 T(\boldsymbol{x}, t) \tag{8}$$

L'equazione ha forma analitica risolvibile imponendo che la soluzione sia divisibile in funzioni di tempo e spazio

$$\begin{aligned} T(\boldsymbol{x}, t) &= X(\boldsymbol{x})\mathcal{T}(t) \\ \frac{\partial^2 X(\boldsymbol{x})}{\partial \boldsymbol{x}^2} &+ k^2 X(\boldsymbol{x}) = 0 \\ \frac{\partial \mathcal{T}(t)}{\partial t} &+ k^2 \frac{C}{C\rho}\mathcal{T}(t) = 0 \end{aligned} \tag{9}$$

con k costante da determiare. Con determinate condizioni al contorno si ottengono i modi normali che restituiscono i risultati per le variabili separate, ma nel caso in esame si tratta di eseguire una simulazione per ottenere valori espliciti

▷ ***Metodo FTCS***: dall'inglese *'forward time central space'* che utilizza l'approssimazione in cui le derivate temporali sono ottenute tramite schema *'forward difference'* mentre quelle spaziali con schema *'central difference'*, quest'ultima restituisce una accuratezza del secondo ordine per lo spazio.

$$\begin{aligned} \frac{\partial T}{\partial t} &\approx \frac{T(x, t+\Delta t) - T(x,t)}{\Delta t} \\ \frac{\partial^2 T}{\partial x^2} &\approx \frac{T(x+\Delta x, t) - 2T(x,t) + T(x-\Delta x, t)}{(\Delta x)^2} \end{aligned} \tag{10}$$

fatto ciò si sostituiscono queste approssimazioni nell'equazione (8) ottenendo la forma

$$T_{i,j+1} = T_{i,j} + \eta[T_{i+1,j} + T_{i-1,j} - 2T_{i,j}]$$
$$\eta = \frac{K\Delta t}{C\rho\Delta x^2} \tag{11}$$

che è un algoritmo esplicito.
Si applica inoltre la *condizione di Von Neumann*: affinchè uno schema numerico risulti stabile è necessario che il fattore di apmlificazione degli errori a ciascun *time step* sia minore o uguale ad 1. Nello specifico si assume che sia possibile scrivere i modi propri di oscillazione della temperatura come dipendenti da una funzione complessa $\xi(k)$, motivo per cui si denota il termine $\xi(k)^j$ come *fattore di amplificazione* che aumenta esponenzialmente i modi propri della temperatura nel tempo. Se quanto appena detto è vero, per evitare un aumento esponenziale degli errori, ξ non può crescere al crescere di j, pertanto deve valere

$$|\xi(k)| \leq 1 \tag{12}$$

da cui in seguito alle dovute sostituzioni si trova:

$$\eta = \frac{K\Delta t}{C\rho\Delta x^2} \leq \frac{1}{2} \tag{13}$$

tale condizione è anche nota come *analisi di stabilità di Von Neumann*

▷ ***Metodo Crank-Nicholson***: tale algoritmo fa uso della approssimazione di derivata temporale a differenza centrale. Il metodo utilizza uno *splitting* nel *time step* da t a $t + \Delta t$, così facendo, siccome è noto un singolo valore temporale, non è necessario introdurre errore nel *time step* iniziale:

$$\frac{\partial T}{\partial t}(x, t + \frac{\Delta t}{2}) \approx \frac{T(x, t + \Delta t) - T(x, t)}{\Delta t} + \mathcal{O}(\Delta t^2) \tag{14}$$

dove l'ultimo termine si riferisce all'errore derivato dalla troncatura dello sviluppo, essendo un termine quadratico esso risulta più preciso degli algoritmi con errore di troncatura a grado 1, in modo analogo per la derivata spaziale al tempo $t + \Delta t$ vale:

$$\begin{aligned} 2(\Delta x)^2 \frac{\partial^2 T}{\partial x^2}(x, t + \frac{\Delta t}{2}) \approx & [T(x - \Delta x, t + \Delta t) - 2T(x, t + \Delta t) \\ & + T(x + \Delta x, t + \Delta t)] + [T(x - \Delta x, t) - 2T(x, t) \\ & + T(x + \Delta x, t)] + \mathcal{O}(\Delta x^2) \end{aligned} \tag{15}$$

in questi termini sostituendo e raggruppando termini per stessa temperatura si ottiene l'equazione

$$-T_{i-1,j+1} + \left(\frac{2}{\eta} + 2\right) T_{i,j+1} - T_{i+1,j+1} = T_{i-1,j} + \left(\frac{2}{\eta} - 2\right) T_{i,j} + T_{i+1,j} \tag{16}$$

in cui la parte sinistra rappresenta l'evoluzione nel tempo futuro mentre la destra rappresenta il presente. Tale algoritmo è detto implicito perchè è necessario risolvere contemporaneamente più equazioni per ottenere tutte le soluzioni spaziali. Notiamo

che ciò è contrario a quanto detto per il metodo FTCS che invece è esplicito. Si può riscrivere in forma matriciale

$$\begin{bmatrix} (2/\eta+2) & -1 & 0 & \dots & \dots & 0 \\ -1 & (2/\eta+2) & -1 & 0 & \dots & 0 \\ 0 & 1 & (2/\eta+2) & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \vdots \\ 0 & \dots & \dots & 1 & (2/\eta+2) & -1 \\ 0 & \dots & \dots & \dots & -1 & (2/\eta+2) \end{bmatrix} \begin{bmatrix} T_{i-1,j+1} \\ T_{i,j+1} \\ T_{i+1,j+1} \\ \vdots \\ T_{N-2,j+1} \\ T_{N-1,j+1} \end{bmatrix} = \begin{bmatrix} T_{i-1,j} + (2/\eta-2)T_{i,j} + T_{i+1,j} \\ T_{i,j} + (2/\eta-2)T_{i+1,j} + T_{i+2,j} \\ T_{i+1,j} + (2/\eta-2)T_{i+2,j} + T_{i+3,j} \\ \vdots \\ T_{N-3,j} + (2/\eta-2)T_{N-2,j} + T_{N-1,j} \\ T_{N-2,j} + (2/\eta-2)T_{N-1,j} + T_{N,j} \end{bmatrix} \tag{17}$$

in cui la matrice non tiene conto di eventuali condizioni al contorno.
L'algoritmo CN presenta implicitamente l'analisi di stabilità di *Von Neumann*, in cui $\xi(k)$ è dipendente da sin^2 pertanto sempre minore o uguale a 1

- ***Trasformata di Fourier***: trasformazione matematica che consente di convertire una funzione definita su un dominio in un'altra definita in un altro dominio per mezzo di un integrale. La trasformazione avviene da una funzione temporale ad una con base esponenziale, tale per cui i coefficienti dello sviluppo rappresentano la funzione nel dominio delle frequenze, costituendo lo *spettro* della funzione. La teoria di fondo discende dalle serie di Fourier, ovvero la rappresentazione di una funzione quadrato sommabile a mezzo di una combinazione lineare dei vettori di base di uno spazio trigonometrico.
I coefficienti di Fourier sono le proiezioni ortogonali della funzione sugli spazi generati dai singoli elementi del sistema e definiti su un intervallo sufficientemente grande da contenere la funzione, a mezzo dell'integrale si elide la discretizzazione:

$$c_k = \frac{1}{T}\int_{-T/2}^{T/2} f(t)e^{-i\omega_k t}\,dt \tag{18}$$

con ω_k frequenze discrete. Per quanto detto sopra si definiscono rispettivamente la trasformata e l'antitrasformata di Fourier come:

$$\begin{aligned} F(\omega) &= \int_{-T/2}^{T/2} f(t)e^{-i\omega t}\,dt \\ f(t) &= \frac{1}{T}\int_{-\infty}^{\infty} F(\omega)e^{i\omega t}\,d\omega \end{aligned} \tag{19}$$

Le proprietà di linearità, convoluzione nel dominio del tempo (moltiplicazione nel dominio delle frequenze) e il teorema della convoluzione consentono di elaborare i segnali, nel caso in esame: la trasformata viene utilizzata per analizzare le componenti di frequenza presenti nell'immagine sporca e nel rumore generato, consentendo di separare il rumore dal segnale. L'antitrasformata viene utilizzata per ottenere l'immagine ripulita combinando le informazioni di frequenza del segnale originale e del rumore.

▷ ***Metodo di Box-Muller***: ideato per generare due numeri casuali indipendenti, x_1 e x_2, normalmente distribuiti nell'intervallo (0, 1] e avranno media 0 e varianza 1. Definite due variabili aleatorie u e v si definiscono i numeri nell'intervallo $(0, 1]$:

$$\begin{aligned} Z_0 &= Rcos\theta = \sqrt{-2ln(u)}cos(2\pi v) \\ Z_1 &= Rsin\theta = \sqrt{-2ln(v)}sin(2\pi u) \end{aligned} \tag{20}$$

Detto u numero esponenziale, sia l'intervallo definito come $(0, a]$, θ numero uniforme generato nell'intervallo, i numeri di Box-Muller dell'immagine da analizzare sono:

$$\begin{aligned} x_1 &= \sqrt{2u}sin\theta \\ x_2 &= \sqrt{2u}cos\theta \end{aligned} \tag{21}$$

Nel caso in esame vengono generati al fine di coprire una immagine con del rumore che dovrà essere pulita usando le condizioni di generazione e le trasformate di Fourier.

2 Esecuzione

Problema A:

▷ ***FTCS-CN*** : Confronto tra i metodi.

Note le condizioni si considera il problema monodimensionale assumendo che tutti gli elementi che compongono il bordo del contenitore abbiano spessore $l = 1cm$. Nota la diffusività termica dell'acciaio $k_a = 4.25 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}$, si converte l'elemento di area in cm^2 per non avere problemi di unità di misura. Si sceglie divisione dello spessore $N = 100$ e si parte da un basso *time step* $tstep = 200$ in modo da enfatizzare eventuali differenze tra i due metodi. Si applica condizione di *Von Neumann* tale che la variazione temporale sia la medesima per entrambi i metodi

$$\begin{aligned} \Delta t &= \frac{\Delta x^2}{2k} \\ k &= \frac{K}{C\rho} \end{aligned} \tag{22}$$

Si definisce un parametro di differenza tra i due istanti di tempo nel ciclo for

$$\mathrm{diff} = \max\left(|T_{i,\mathrm{time}} - T_{i,\mathrm{time}+1}|\right) \tag{23}$$

a cui si aggiunge la condizione

$$\begin{aligned} &\text{if } \mathrm{diff}_1 \neq 0. \\ &\mathrm{time} = \mathrm{time} + 1 \end{aligned} \tag{24}$$

che consente di determinare quando la temperatura del materiale va a convergenza.
Nel metodo CN bisogna riscrivere le matrici in modo che A venga utilizzata per moltiplicare il vettore delle incognite al tempo $t + 1$, mentre la matrice B moltiplichi il vettore delle incognite al tempo t, pertanto vale l'uguaglianza con l'equazione (17)

$$A\boldsymbol{x} = B \tag{25}$$

Il Codice Python risulta dunque il seguente:

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from matplotlib import cm

#inizializzazione
N = 100 #numero partizioni
tstep = 200 #time step
k = 4.25 * 10**(-2) #diffusività termica acciaio e legno
l = 1 #lunghezza sbarra

#funzione FTCS
def ftcs(N, tstep, k, l):
    h = l / N #suddivisione spaziale
    t_1 = (h**2) / (2 * k) #tempo con criterio di stabilità

    #condizioni al contorno
    T_1 = np.zeros((N, tstep))
    T_1[0, :] = 5
    T_1[-1, :] = 50
    T_1[1:-1, 0] = 27
    time_1 = 0 #counter per somma a convergenza FTCS

    diff_1 = float(np.amax(abs(T_1[:,time_1]-T_1[:,time_1+1]))) \
    #parametro Temperatura-tempo per equilibrio


    #ciclo per FTCS
    for time in range(1, tstep):
        for i in range(1, N-1):
            T_1[i, time] = T_1[i, time-1] + k * (t_1 / h**2) * (T_1[i-1, time-1] + \
            T_1[i+1, time-1] - 2 * T_1[i, time-1])

        diff_1 = (np.amax(abs(T_1[:,time-1]-T_1[:,time])))
        if diff_1 != 0.:
          time_1 += 1 #condizione if fino a raggiungimeno convergenza

    FTCS = [T_1, time_1]

```

```
    if FTCS[1] < tstep:
      print("Nel metodo FTCS l'acciaio converge al passo", FTCS[1], \
            ' nel tempo t_1 = ', FTCS[1]*t_1*1e2, 'secondi')\
             #conversione per errore dato dai cm invece che m
    else:
      print("Nel metodo FTCS l'acciaio tende alla divergenza nel passo",  \
            FTCS[1], ' nel tempo t_1 = ', FTCS[1]*t_1*1e2, 'secondi')

    return FTCS

#funzione Crank-Nicholson
def crank_nicholson(N, tstep, k, l):
    h = l / N #suddivisione spaziale
    t_1 = (h**2) / (2 * k) #tempo con criterio di stabilità

    #condizioni al contorno
    T_2 = np.zeros((N, tstep))
    T_2[0, :] = 5
    T_2[-1, :] = 50
    T_2[1:-1, 0] = 27

    time_2 = 0 #counter per somma a convergenza CN

    diff_2 = float(np.amax(abs(T_2[:,time_2]-T_2[:,time_2+1])))

    A = np.zeros((N, N)) #matrice 1 CN
    B = np.zeros((N, N)) #matrice 2 CN

    #condizioni matrici
    A[0, 0] = 1
    A[-1, -1] = 1

    B[0, 0] = 1
    B[-1, -1] = 1

    #ciclo C-N per le matrici
    for i in range(1, N-1):
        A[i, i-1] = -k * t_1 / h**2
        A[i, i] = (2 * k * t_1 / h**2) + 1
        A[i, i+1] = -k * t_1 / h**2
```

```
80
81          B[i, i-1] = k * t_1 / h**2
82          B[i, i] = -(2 * k * t_1 / h**2) + 1
83          B[i, i+1] = k * t_1 / h**2
84
85      #ciclo C-N per la Temperatura
86      for time in range(1, tstep):
87          T_2[:, time] = np.linalg.solve(A, np.dot(B, T_2[:, time-1]))
88
89          diff_2 = (np.amax(abs(T_2[:,time-1]-T_2[:,time])))
90
91          if diff_2 != 0.:
92                time_2 += 1
93
94      CN = [T_2, time_2]
95
96      if CN[1] < tstep:
97        print("Nel metodo CN l'acciaio converge al passo", CN[1], \
98              ' nel tempo t_2 = ', CN[1]*t_1*1e2, 'secondi') \
99              #conversione per errore dato dai cm invece che m
100     else:
101       print("Nel metodo CN l'acciaio tende alla divergenza nel passo",  CN[1], \
102             'nel tempo t_2 = ', CN[1]*t_1*1e2, 'secondi')
103
104     return CN
```

Da cui si ricavano valori di tempo di convergenza identici per FTCS e CN, giacchè l'intervallo temporale del *time step* è troppo breve, è comunque interessante notare il grafico del confronto tra i due metodi generato attraverso il codice:

```
1  T_ftcs = ftcs(N, tstep, k, l)
2  T_cn = crank_nicholson(N, tstep, k, l)
3
4  plt.rcParams['font.family'] = 'serif'
5
6  fig_compar = plt.figure(figsize=(8, 6))
7  fig_compar.suptitle('Comparazione FTCS - Crank-Nicolson' , fontsize = 12 , y=0.9 )
8  ax_compar = fig_compar.add_subplot(111, projection="3d")
9  ax_compar.plot_wireframe(gridx_cn, gridy_cn, T_cn[0], color='blue', linewidth=0.5)
10 ax_compar.plot_wireframe(gridx_ftcs, gridy_ftcs, T_ftcs[0], color='darkorange', linew
11 ax_compar.set_xlabel('Tempo (s)')
```

```
ax_compar.set_ylabel('Lunghezza (cm$^{-2}$)')
ax_compar.set_zlabel('Temperatura (°C)', labelpad=-4)
ax_compar.view_init(elev=30, azim=220)
ax_compar.tick_params(axis='z', pad=-2)
ax_compar.legend(['Crank-Nicolson', 'FTCS'] , loc='upper left', bbox_to_anchor=(0.7, 
plt.subplots_adjust(left=0.35)

plt.show()
```

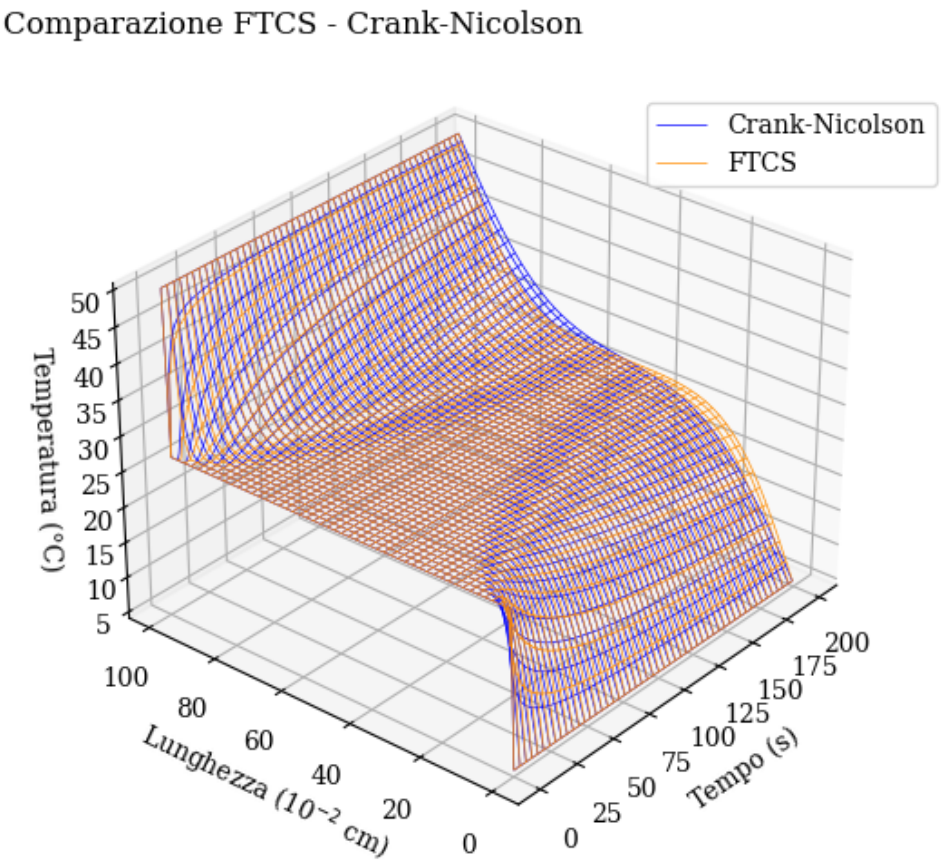

Figure 1: Confronto grafici Acciaio FTCS-CN, $N = 100$, $tstep = 200$

Dove sull'asse verticale è riportata la temperatura, sull'asse delle ascisse il tempo in secondi e sulle ordinate la lunghezza in centesimi di centimetro (scelta derivante dal numero di divisioni selezionato all'inizio).
Non vi è grossa differenza tra i due grafici, si notano alcune differenze nei punti e leggera variazione nelle curve, le figure sono distinguibili ma molto affini tra loro.
Portando il *time step* al suo massimo valore $tstep = 100.000$ si può eseguire ulteriore confronto per determinare se esiste una differenza nei grafici generati singolarmente per ciascun metodo:

```
fig_ftcs = plt.figure(figsize=(8, 6))
```

```
fig_ftcs.suptitle('Profilo di Temperatura FTCS' , fontsize = 12 , y=0.9)
ax_ftcs = fig_ftcs.add_subplot( projection="3d")
gridx_ftcs, gridy_ftcs = np.meshgrid(range(tstep), range(N))
ax_ftcs.plot_surface(gridx_ftcs, gridy_ftcs, T_ftcs[0], cmap=cm.coolwarm, \
                     linewidth=0, antialiased=False)
ax_ftcs.set_xlabel('Tempo (s)')
ax_ftcs.set_ylabel('Lunghezza (10$^{-2}$ cm)')
ax_ftcs.set_zlabel('Temperatura (°C)' , labelpad= -9)
ax_ftcs.view_init(elev=30, azim=220)
ax_ftcs.tick_params(axis='z', pad=-4.5)


fig_cn = plt.figure(figsize=(8, 6))
fig_cn.suptitle('Profilo di Temperatura C-N' , fontsize = 12 , y=0.9)
ax_cn = fig_cn.add_subplot( projection="3d")
gridx_cn, gridy_cn = np.meshgrid(range(tstep), range(N))
ax_cn.plot_surface(gridx_cn, gridy_cn, T_cn[0], cmap=cm.coolwarm, linewidth=0, \
                   antialiased=False)
ax_cn.set_xlabel('Tempo (s)')
ax_cn.set_ylabel('Lunghezza (10$^{-2}$ cm)')
ax_cn.set_zlabel('Temperatura (°C)' , labelpad= -9)
ax_cn.view_init(elev=30, azim=220)
ax_cn.tick_params(axis='z', pad=-4.5)

plt.show()
```

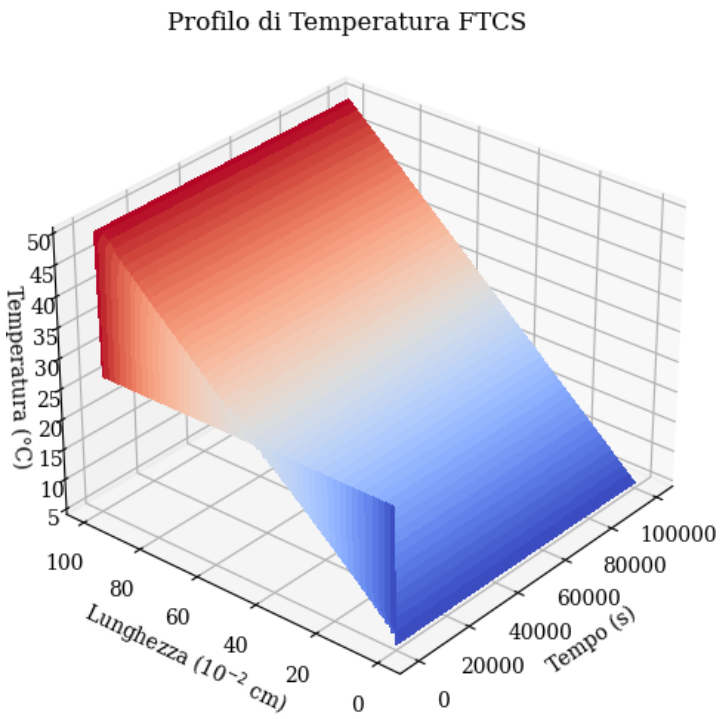

Figure 2: Grafico Acciaio FTCS, $N = 100$, $tstep = 100.000$

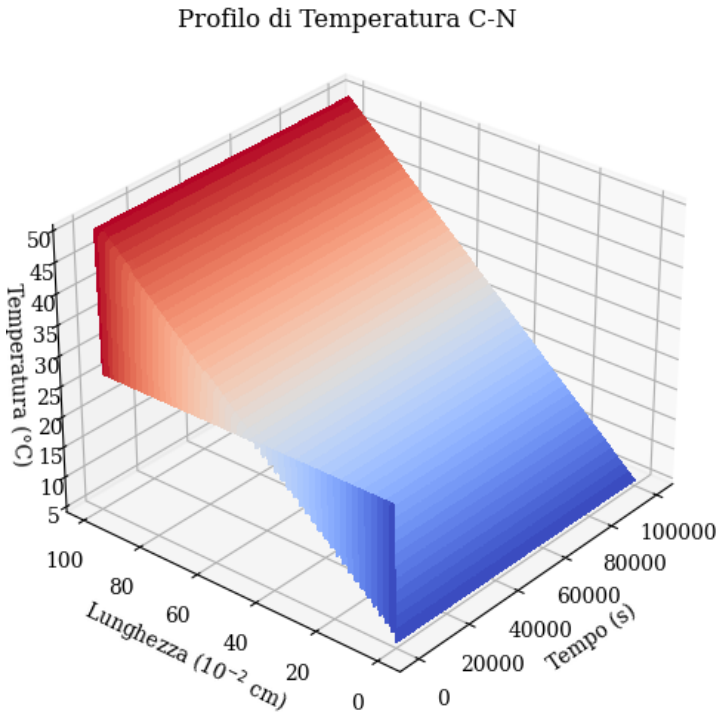

Figure 3: Grafico Acciaio CN, $N = 100$, $tstep = 100.000$

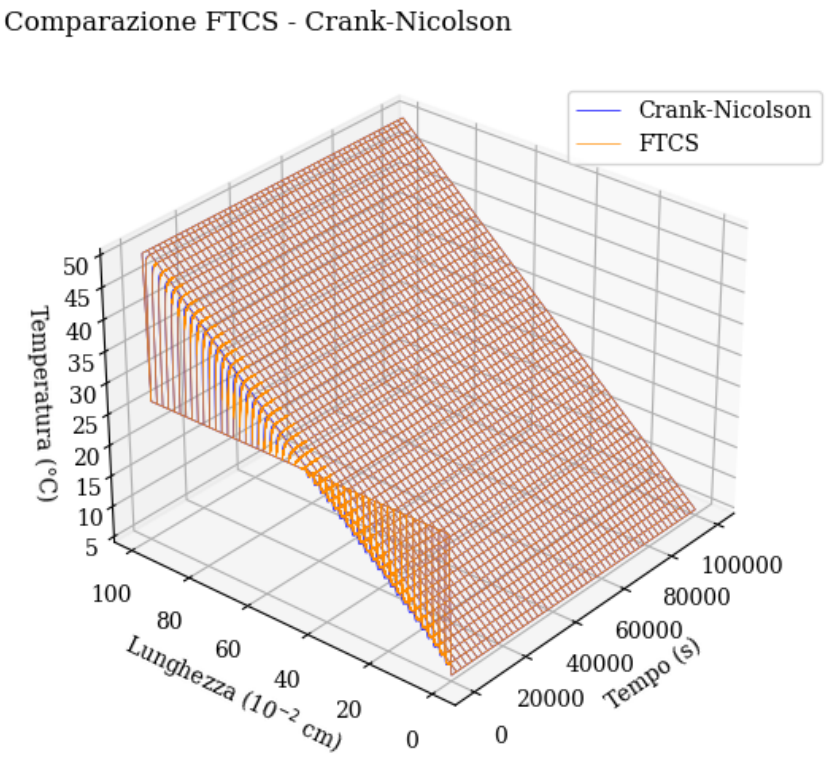

Figure 4: Confronto grafici Acciaio FTCS-CN, $N = 100$, $tstep = 100.000$

Rispetto al grafico 1 la differenza tra il metodo FTCS e CN si assottiglia terribilmente, difatti nella figura 4 la maggior parte dei punti coincidono sovrapponendosi gli uni sugli altri. Ci si chiede quale dei due metodi sia il più preciso, o il più adatto. La prima stima che si può dare è relativa ai tempi di convergenza che ciascun metodo ricava a parità di condizioni iniziali, comprese divisioni della barra e step temporali:

$$\begin{aligned} t_{FTCS} &= 6.111s \\ t_{CN} &= 3.047s \end{aligned} \tag{26}$$

Che ci mostrano un problema: nel metodo FTCS l'acciaio va in equilibrio dopo 100 ore mentre nel metodo CN ne impiega 50. Questi tempi scala sono assurdamente grandi, e, a condizione di non aver sbagliato le conversioni (cosa che è possibile), ciò è imputabile allo scarto imposto in *counter-timestep* dove si richiede $t_i < tstep$. La spiegazione alternativa è che tale tempo è derivante dal metodo discretizzato (cosa che risulta più probabile), che calcola il profilo di temperatura, e poichè è più preciso per definizione matematica il metodo CN, il tempo riscontrato con quest'ultimo è più vicino a quello reale. Il problema non è però risolto perchè non è possibile fare una comparazione effettiva con una sbarra d'acciaio, non vi sono tabelle con questa diffusività termica che riportino il tempo necessario ad una sbarra di 1 cm di raggiungere lo stato di equilibrio termico.
Un secondo confronto potrebbe venire dall'analisi degli errori in merito ai punti calcolati, per esempio calcolando lo scarto quadratico medio dei punti per entrambe le simulazioni, ma ancora una volta non ci sono riscontri con parametri reali per poter asserire che il metodo CN sia il migliore da utilizzare.
Il dilemma è stato risolto in seguito al consulto col tutor, il quale ha ritenuto che per gli scopi didattici preposti la miglior soluzione fosse quella di adoperare il metodo FTCS poichè è il metodo adoperato nelle esercitazioni, ulteriormete ha ritenuto più utile concentrarsi su una analisi fisica dei fenomeni che la simulazione può consentire di determinare.

Problema A:

▷ ***k variabile*** : Confronto tra due diffusività termiche.

La differenza con la parte precedente sta nel fatto che viene introdotta una nuova diffusività termica $k_l = 0.13 \times 10^{-6}\,\mathrm{m}^2/\mathrm{s}$, pertanto nel criterio di stabilità bisogna scegliere un Δt comune per entrambe le diffusivià in modo da poter effettuare un confronto, il chè lascia due scelte:

1. Si sceglie un Δt sufficientemente piccolo da poter essere usato per entrambe a priori
2. Dal criterio di stabilità si determinano i Δt per ciascun materiale e poi si prende il più piccolo per entrambe

Si è optato per la seconda scelta

```
    h = l / N
    t_1 = (h**2) / (2 * k_1) #tempo acciaio con criterio di stabilità
    t_2 = (h**2) / (2 * k_2) #tempo legno con criterio di stabilità
    tau = min(t_1,t_2) #valore minimo per entrambi i materiali
```

Il resto della procedura è totalmente analogo a quanto già visto, semplicemente viene effettuato il ciclo per entrambi materiali in contemporanea. Si evita di riportare l'intero codice onde appesantire la scrittura, lo si può tuttavia consultare negli allegati. Vengono riportati comunque i il profilo di temperatura per il legno e la sovrapposizione dei profili calcolati con entrambe le diffusività

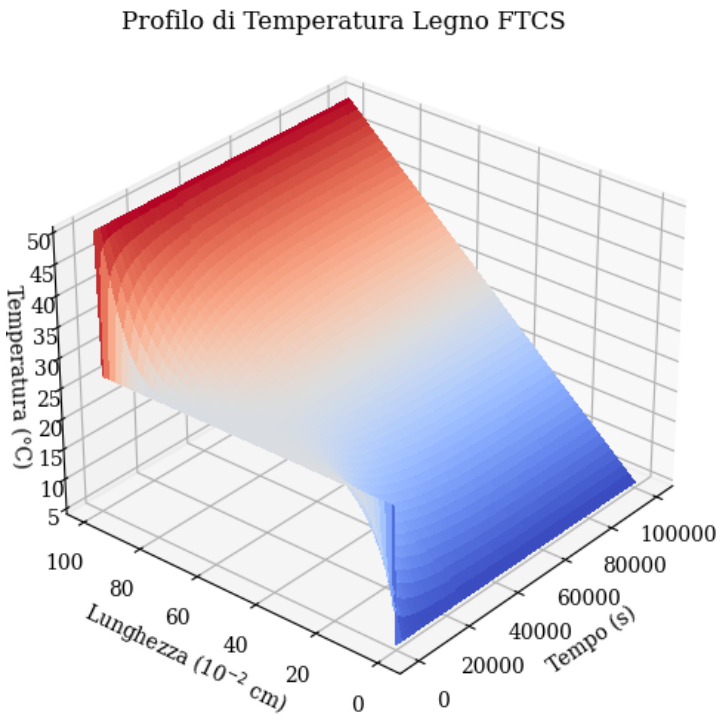

Figure 5: Grafico Legno FTCS, $N = 100$, $tstep = 100.000$

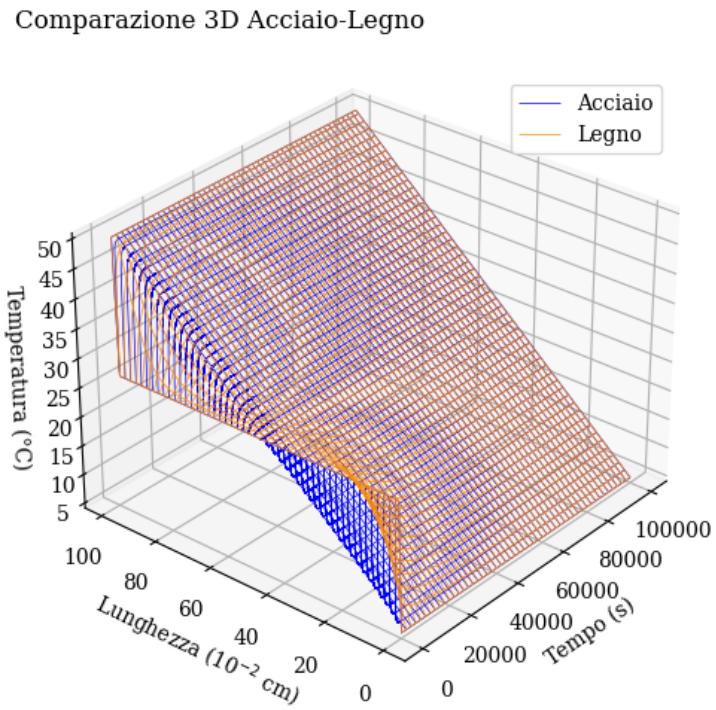

Figure 6: Grafico Comparazione Legno-Acciaio FTCS, $N = 100$, $tstep = 100.000$

I profili ricavati in figura 6 sembrano simili per via dell'inclinazione e del tempo di calcolo, le differenze si notano nel momento in cui si analizzano i tempi di equilibrio calcolati che valgono:

$$\begin{aligned} t_{acciaio} &= 6.111s \\ t_{legno} &= 11.764s \end{aligned} \tag{27}$$

Ovvero il legno raggiunge stabilità solo a fine ciclo, dopo quasi 200 ore, comportamento derivante in prima analisi da quanto detto in precedenza per il metodo FTCS poi derivante dal fatto che il legno non è un buon conduttore termico.

Problema B:

▷ ***Trasformate di Fourier*** : Pulizia di una immagine dal rumore

Per realizzare il programma in esame sono stati riscontrati alcuni problemi, il primo tra tutti la definizione del numero esponenziale u, il quale si discostava dalla definizione canonica, un termine $\frac{1}{2}$ doveva essere ulteriormente aggiunto per creare il rumore corretto. In seguito a consulto col tutor è stato possibile costruire il programma corretto ma senza il quale non sarebbe stato possibile.
La prima porzione di codice relativa alla pulizia dell'immagine è la seguente:

```
#Pulizia immagine
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rn

dirty = np.load('/img_modificata.npy') #immagine
```

```
7
8  L = np.shape(dirty)[0] #larghezza
9  H = np.shape(dirty)[1] #altezza
10 C = np.shape(dirty)[2] #numero canali di colore
11 s  = 1642147
12
13 noise = np.zeros((L,H,C)) #array del rumore
14 ref = np.array([0.05627879, 0.22818874, 0.44774995]) \
15 #l'array con il riferimento
16 diff = list()
17 A = list()
18
19 #per determinare a ottimale
20 for a in np.arange (1.0,2.0,0.1): #intervallo di definizione di a
21     rn.seed(s)
22     for l in range(L):
23         for h in range(H):
24             for c in range(C):
25                 u = -np.log(1-rn.uniform(0,1))\
26                  #numero logaritmico per Box Muller
27                 theta = rn.uniform(0,a)
28                 noise[l,h,c] = (0.5*np.sqrt(2*u)*(np.cos(theta)\
29                                 + np.sin(theta))) #creo il rumore casuale
30     norm = 1./np.amax(noise)
31     noise *= norm #normalizzo
32     diff.append(abs(ref[0]-noise[0,0,0]))
33     A.append(a)
34
35 plt.rcParams['font.family'] = 'serif'
36
37 #Immagine sporca
38 plt.figure(1)
39 plt.title('Immagine sporca')
40 plt.imshow(dirty.astype(np.uint8))
41 plt.axis('off')
42
43 #Riferimento-Rumore
44 plt.figure(2, figsize=(6, 4))
45 plt.title('Riferimento-Rumore', fontsize=12)
46 plt.plot(A, diff)
```

```
plt.xlabel('Parametro "a"', fontsize=10)
plt.ylabel('Differenza' , fontsize=10)
plt.grid(True)
```

Si noti che a fine codice vengono mostrate sia l'immagine sporca che il grafico per i valori di a ottenuti dallo scarto del rumore con il riferimento

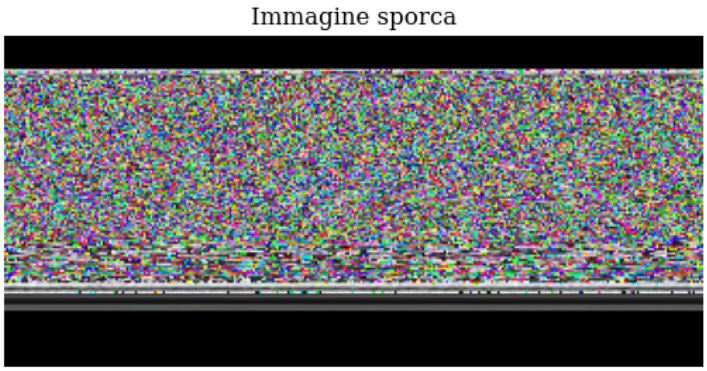

Figure 7: Immagine sporca da analizzare

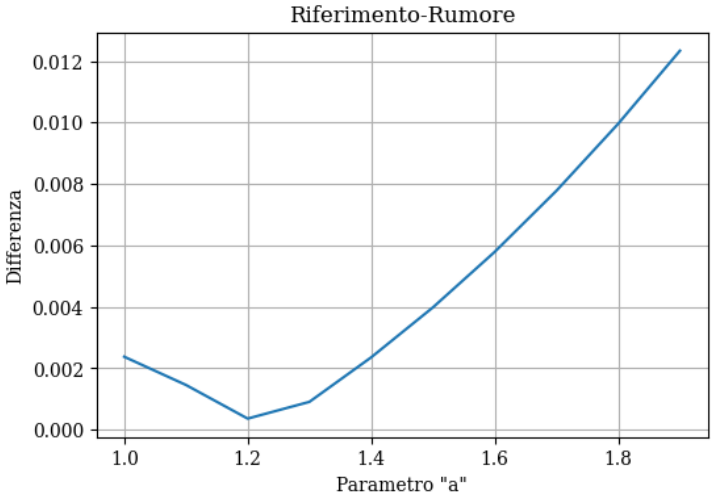

Figure 8: Grafico Riferimento-Rumore per visualizzare l'andamento da cui si ricava a

Infine, definito a ottimale ricavato nell'intervallo $(1, 2)$ si devono riscrivere i numeri di *Box-Muller* per generare il rumore e finalmente si possono utilizzare rispettivamente trasformata ed antitrasformata di *Fourier* per ricavare l'immagine pulita

```
#Il rumore va ricalcolato con "a" ottimale
a = min(A)
rn.seed(s)

for l in range(L):
      for h in range(H):
```

```
        for c in range(C):
            u = -np.log(1-rn.uniform(0,1))
            theta = rn.uniform(0,a)
            noise[l,h,c] = (0.5*np.sqrt(2*u)*(np.cos(theta)+np.sin(theta)))
norm = 1./np.amax(noise)
noise *= norm

#conversione per trasformata
FT_dirty = np.fft.fft2(dirty)  #nel dominio delle frequenze
FT_noise = np.fft.fft2(noise) #nel dominio spazio/tempo
clean = abs(np.fft.ifft2(FT_dirty/FT_noise)) #immagine pulita

plt.figure(3)

fig = plt.imshow(clean.astype(np.uint8))
fig.axes.get_xaxis().set_visible(False)
fig.axes.get_yaxis().set_visible(False)
plt.show()
```

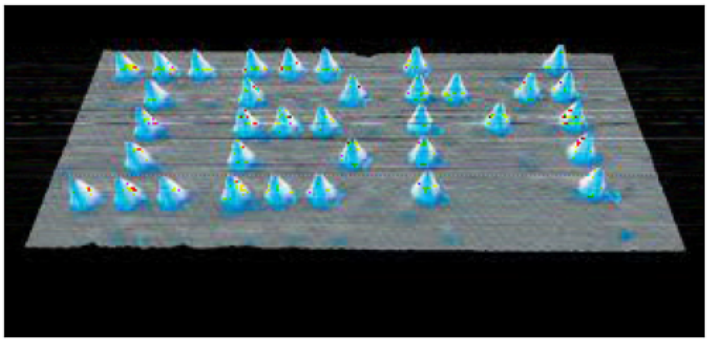

Figure 9: Immagine pulita attraverso trasformata di *Fourier*

3 Analisi dei dati:

Per quanto concerne il *Problema A* rimanevano ancora delle possibili analisi da fare sul comportamento fisico di acciaio e legno, quindi sotto consulto del tutor si è optato per la valutazione dell'andamento della temperatura in funzione della distanza al variare del *time step*:

```
#Andamento della temperatura lungo la sbarra
fig_3, (ax_1, ax_2) = plt.subplots(1, 2)

space = range(0, N) #suddivisione sbarra
```

```
5
6  #Variazione della temperatura nella sbarra
7  for i in [0, 100, 500, 1000, int(1e4), 40000, tstep-1]: #i = istanti di tempo
8      label_1 = "(tstep = {})".format(i)
9      label_2 = "(tstep = {})".format(i)
10     ax_1.plot(T_ftcs[0][:, i], label=label_1)
11     ax_2.plot(space, T_ftcs[1][:, i], label=label_2)
12
13 #grafici
14 fig_3.suptitle('Temperatura lungo la sbarra')
15 ax_1.set_title("Acciaio")
16 ax_1.legend()
17 ax_1.set(xlabel="Lunghezza (10$^-2$ cm)", ylabel="Temperatura (C)")
18 ax_1.set_xticks(np.linspace(0, 100, 6))
19 ax_1.grid()
20
21 ax_2.set_title("Legno")
22 ax_2.legend()
23 ax_2.set_xlabel("Lunghezza (10$^-2$ cm)")
24 ax_2.set_xticks(np.linspace(0, 100, 6))
25 ax_2.grid()
26
27 plt.show()
```

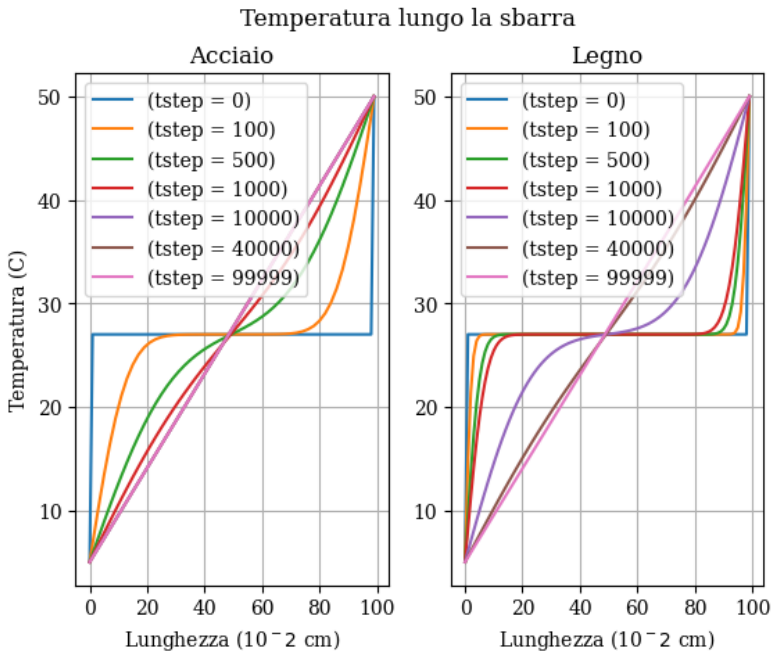

Figure 10: Variazione della Temperatura nella sbarra per vari tempi di simulazione

Come si può notare dalla figura per basso *time step* si presentano dei plateu derivanti dal comportamento di tendenza all'equilibrio dei corpi. Per bassa diffusività si nota la necessità di un processo più lungo per raggiungere un comportamento lineare, quest'ultimo rappresenta una variazione costante di temperatura ed un tempo lunghissimo per raggiungere l'equilibrio termico per i motivi già analizzati in precedenza. Di seguito viene riportato il grafico relativo a Temperatura in funzione del tempo per dati elementi della sbarra:

```
#Temperatura in funzione del tempo
time = [] #tempo per ciascun punto simulazione
for i in range(0, tstep):
    time.append(int(i)*T_ftcs[4]*1e2)

#Andamento per punto della sbarra (x = 0.3 cm)
fig_4, (ax_3, ax_4) = plt.subplots(1,2)
fig_4.suptitle('Temperatura in funzione del tempo')
ax_3.plot(time,T_ftcs[0][20,:], label = "legno")
ax_3.plot(time,T_ftcs[1][20,:], label = "acciaio")
ax_3.set_xlabel("Tempo (s)")
ax_3.set_ylabel("Temperatura(C)")
ax_3.set_title("x = 0,3 cm")
ax_3.legend()
ax_3.grid()

#Andamento per punto della sbarra (x = 0.7 cm)
ax_4.plot(time,T_ftcs[0][70,:], label = "legno")
ax_4.plot(time,T_ftcs[1][70,:], label = "acciaio")
ax_4.set_xlabel("Tempo (s)")
ax_4.set_ylabel("Temperatura(C)" , labelpad = -2)
ax_4.set_title("x = 0,7 cm")
ax_4.legend()
ax_4.grid()
```

Nel grafico generato si noterà come sia presente una variazione della concavità analoga a quanto visto in precedenza, l'andamento rappresenta come i due materiali tendano all'equilibrio in funzione del tempo in punti precisi della sbarra.

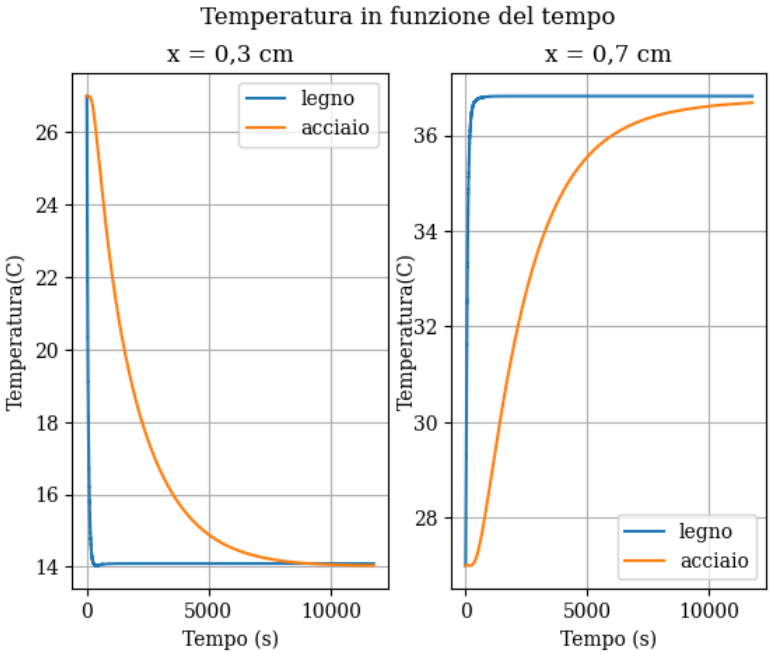

Figure 11: Variazione della Temperatura nella sbarra in funzione del tempo di simulazione

4 Conclusioni

Si è riusciti a portare a termine i compiti preposti.
Per quanto concerne il confronto tra i metodi Crank-Nicholson e FTCS, per quanto detto nell'*Esecuzione*, non è possibile stimare univocamente che il metodo CN sia più preciso nonostante la matematica della sua struttura lo dica intrinsecamente (errore quadratico). Per quanto si trovi convergenza all'equilibrio più in linea con i valori reali in CN essi sono sempre tempi fuori scala rispetto ai valori reali, derivanti da quanto detto nel paragrafo 2.
Per quanto concerne i paragoni costruiti col metodo FTCS per le due diffusività troviamo valori di convergenza per l'acciaio che invece non si trovano nel legno, o meglio si trova convergenza a tempo massimo. Il resto dei grafici e dei confonti restituiscono un profilo sensato per i tempi di simulazione, con l'acciaio che raggiunge sempre, per qualunque *time step*, equilibrio prima del legno.
Nella parte relativa alle trasformate di *Fourier* non sono state effettuate analisi fisiche per via della natura del problema, si può solo dire di essere riusciti a pulire l'immagine dal rumore.