```python
from scipy import optimize
from scipy.special import erfc
from scipy.special import erf
from cmath import pi, sqrt, exp

# Roots of "An Accurate Approximation of the Two-Phase Stefan Problem with Coefficient Smoothing"
g = -20
u0 = 10
k1 = 2.26
k2 = 0.59
c1 = 4.182E6
c2 = 4.182E6
D = 3.35E8
a1 = sqrt(k1/c1)
a2 = sqrt(k2/c2)

def func(x):
   return ((((k1/a1) * g * exp(-(x/(2*a1))**2)) / erf(x/(2*a1))) + ((k2/a2) * u0 * (exp(-(x/(2*a2))**2)) /
(1.0-erf(x/(2*a2)))) + ((x * D * sqrt(pi)) / 2))

sol = optimize.root_scalar(func, rtol=1E-12, method='secant', x0=-0.1, x1=0.0005)
lambd = sol.root.real
print("An Accurate Approximation of the Two-Phase Stefan Problem with Coefficient Smoothing: ", lambd)
#
#  Roots of "Numerical study of solid-liquid phase change by phase field method"
tm = 0.15
t0 = 10
tb = -20
L = 335000
cps = 4182
k1 = 2.26
k2 = 0.59
rho1 = 916.8
rho2 = 999.8
a1 = k1/(rho1*cps)
a2 = k2/(rho2*cps)

def f(x):
   return ((exp(-(x**2))/erf(x)) + (k2/k1) * sqrt(a1/a2) * ((tm-t0)/(tm-tb)) *
(exp(-(a1/a2)*(x**2))/erfc(x*sqrt(a1/a2))) - (x*L*sqrt(pi))/(cps*(tm-tb)))

sol1 = optimize.root_scalar(f, rtol=1E-12, method='secant', x0=0.1, x1=0.5)
lambd1 = sol1.root.real

import os
import re
from tkinter import Tk
from tkinter.filedialog import askdirectory
import csv
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
path = askdirectory(title='Select Folder')
```

```python
for root, dirs, files in os.walk(path):
    for i in files:
        if i == 'mesh1_TX_950s.xlsx':
            dfs1 = pd.read_excel(root+'/'+i)
            T = []
            x = []
            idx1 = np.where(dfs1.columns == "T")[0][0]
            T = dfs1.values[:,idx1]
            T = T - 273.15
            idx2 = np.where(dfs1.columns == "X")[0][0]
            x = dfs1.values[:,idx2]

            t = np.linspace(0,50,len(x))
            g = -20
            u0 = 10
            k1 = 2.26
            k2 = 0.59
            c1 = 4.182E6
            c2 = 4.182E6
            D = 3.33E8
            a1 = (sqrt(k1/c1))
            a2 = (sqrt(k2/c2))
            fxt = []
            den = []
            psi = []
            fyt = []
            # An Accurate Approximation of the Two-Phase Stefan Problem with Coefficient Smoothing
            j = 0
            for i in x:
                psi = np.append(psi, lambd*sqrt(t[j]))
                if i <= psi[j]:
                    den = np.append(den, 2*a1*sqrt(t[j]))
                    if (t[j]==0.0):
                        fxt = g
                    else:
                        fxt = np.append(fxt, (g * (erf(psi[j]/den[j])-erf(x[j]/den[j])))/(erf(psi[j]/den[j])))

                else:
                    den = np.append(den, 2*a2*sqrt(t[j]))
                    if (t[j]==0.0):
                        fxt = 0.0
                    else:
                        fxt = np.append(fxt, (u0 * (erf(x[j]/den[j])-erf(psi[j]/den[j])))/(1-erf(psi[j]/den[j])))

                j = j + 1

            # Numerical study of solid-liquid phase change by phase field method

            tm = 0.15
            t0 = 10
            tb = -20
            L = 335000
            cps = 4182
            k1 = 2.26
            k2 = 0.59
            rho1 = 916.8
            rho2 = 999.8
            a1 = k1/(rho1*cps)
            a2 = k2/(rho2*cps)

            j = 0
            for i in x:
                psi = np.append(psi, 2*lambd1*sqrt(a1*x[j]))
                if i <= psi[j]:
                    if (t[j]==0.0):
                        fyt = tb
                    else:
                        fyt = np.append(fyt, (erf(x[j]/(2*sqrt(a1*t[j])))/erf(lambd1)) * (tm-tb) + tb)
                else:
                    if (t[j]==0.0):
                        fyt = 0.0
                    else:
                        fyt = np.append(fyt, (erfc(x[j]/(2*sqrt(a2*t[j])))/erfc(lambd1*sqrt(a1/a2))) * (tm-t0) + t0)

                j = j + 1

        else:
            continue

        num_solution = T
        ana_solution1 = fxt
        ana_solution2 = fyt
        numL2 = []
        denL2 = []
        L2 = []
        for k in range(0,len(T)):
            numL2 = np.append(numL2, abs(num_solution[k]-ana_solution2[k])/ana_solution2[k])
            L2 = numL2

dydx = np.gradient(fyt.real, x)
dydxT = np.gradient(T,x)

f1 = plt.figure()
f2 = plt.figure()
f3 = plt.figure()

ax1 = f1.add_subplot(111)
ax1.plot(x, fyt.real, 'r--', label='Neumann solution')
ax1.plot(x, T, 'g--', label='Numerical solution')
ax1.set(xlabel='x [m]', ylabel= 'T [ºC]')
ax1.grid(True)
ax1.legend()
L2 = L2.real/len(T)
ax2 = f2.add_subplot(111)
ax2.plot(x[1:], L2[1:], 'r--', label='Relative error')
ax2.set(xlabel='x [m]', ylabel= 'Relative error')
ax2.grid(True)
ax2.legend()

ax3 = f3.add_subplot(111)
```

```
ax3.plot(x,dydx,'r--', label='Neumann solution')
ax3.plot(x,dydxT,'g--', label='Numerical solution')
ax3.set(xlabel='x [m]', ylabel= 'dy/dx')
ax3.grid(True)
ax3.legend()

plt.show()
```