1 习题 3.17

考虑一维杆热传导问题。杆长 $l = 20m$,横截面面积 $A = 1m^2$,导热系数 $k = 5W°C^{-1}m^{-1}$,热源 $s = 100Wm^{-1}$,边界条件为 $T(x=0) = 0°C$,$\bar{q}(x = 20m) = 0$。本问题的精确解为 $T(x) = -10x^2 + 400x$。

- 采用 2 个等长度线性单元求解此问题,绘制有限元解 $T^h(x)$ 和 $\frac{dT^h(x)}{dx}$,并于精确解比较。
- 将杆分别划分 4、8、和 16 个等长度线性单元,用 bar1D-python 程序求解,绘制有限元解 $L2$ 范数误差 $\|e\|_{L_2} - h$ 的双对数曲线图和自然边界条件误差 $e_n - h$ 曲线图,并考察温度和自然边界条件的收敛特性。

一维杆热传导问题的微分方程为

$$
\begin{aligned}
\frac{d}{dx}(Ak\frac{dT}{dx}) + s = 0, &\quad x \in \Omega \\
qn = \bar{q}, &\quad x \in \Gamma_q \\
T = \bar{T}, &\quad x \in \Gamma_T
\end{aligned}
$$

要使用有限元方法求解,我们首先得到一维热传导方程的 Bublov-Galerkin 弱形式。在处理关于热流通量 q 的自然边界条件时,我们利用了傅立叶定律

$$
q \ = \ -k\frac{dT}{dx}
$$

对原微分方程的等效积分形式进行分布积分,并考虑关于 q 的自然边界条件和关于 T 的本质边界条件,我们得到了一维热传导问题的弱形式

$$
\int_\Omega \frac{d\omega}{dx}\, Ak\, \frac{dT}{dx}\, d\Omega \ = \ \int_\Omega \omega s\, d\Omega \ - \ (A\omega\bar{q})|_{\tau_q}, \quad \forall\omega, \omega|_{\tau_T} = 0
$$

有限元的求解过程可分为以下几步:

- 前处理:将杆件划分为多个单元,对节点进行全局编号。
- 单元分析:利用边界条件,建立单元形函数 $T^e(x)$,得到单元热传导矩阵 K^e,单元节点通量列阵 $\boldsymbol{f}^e$。
- 单元组装:组装单元形函数,得到总体有限元近似函数 $T^h(x)$,总体热传导矩阵 K,总体节点通量列阵 $\boldsymbol{f}$。
- 方程求解:将 $T^h(x)$ 带入弱形式,求解 $\mathbf{Kd} = \mathbf{f} + \mathbf{r}$,其中 $\mathbf{r}$ 为自然边界条件。

1.1 2 个等长度线性单元求解

使用两个等长度线性单元求解。

前处理：将杆件分为两个单元，$element1 : x \in [0, 10]$; $element2 : x \in [10, 20]$

单元分析：Bublov-Galerkin 方法要求试探函数和权函数来自同一函数空间。为了满足有限元解的完备性和连续性要求，试探函数至少为 m 次完备多项式 (m 为弱形式中求导最高次，在这里 $m = 1$)，且试探函数在节点处 C^0 连续。由完备性和线性单元的要求，

$$T^e(x) = \alpha_0^e + \alpha_1^e x$$
$$T^{(1)}(x) = \alpha_0^{(1)} + \alpha_1^{(1)} x$$
$$T^{(2)}(x) = \alpha_0^{(2)} + \alpha_1^{(2)} x$$

由连续性要求，

$$T^{(e)}(0) = \alpha_0^{(e)} + \alpha^{(e)} \times 0 = t_1^{(e)}$$
$$T^{(e)}(10) = \alpha_0^{(e)} + \alpha_1^{(e)} \times 10 = t_2^{(e)}$$

求解两个单元的形函数 $\boldsymbol{N^e}$

$$\boldsymbol{M^{(e)}} = \begin{bmatrix} 1 & 0 \\ 1 & 10 \end{bmatrix}, \quad \boldsymbol{p(x)} = \begin{bmatrix} 1 & x \end{bmatrix}$$

$$\boldsymbol{N^e = p(x)(M^e)^{-1}} = \tfrac{1}{10}\begin{bmatrix} 10 - x & x \end{bmatrix}$$
$$\boldsymbol{B^e} = \boldsymbol{\tfrac{dN^e}{dx}} = \tfrac{1}{10}\begin{bmatrix} -1 & 1 \end{bmatrix}$$

单元刚度矩阵 $\boldsymbol{K^e}$

$$\boldsymbol{K^e} = \int_0^{10} (B^e)^T A^e k^e B^e dx = \frac{1 \times 5}{10}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

单元节点通量列阵 f^e

$$\boldsymbol{f^e} = \int_0^{10} (\boldsymbol{N^e})^{\boldsymbol{T}} s dx = 500 \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

单元组装：使用提取矩阵将单元刚度矩阵组装为总体热传导矩阵。提取矩阵的行数等于单元自由度数目，列数等于总体自由度数目。两个单元的提取矩阵为

$$\boldsymbol{L^{(1)}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$\boldsymbol{L^{(2)}} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

因此总体热传导矩阵为

$$\boldsymbol{K} = \textstyle\sum_{i=1,2} (L^{(i)})^T K^{(i)} L^{(i)} = \frac{1}{2} \begin{bmatrix} 1 & -1 & 0 \\ -1 & 2 & -1 \\ 0 & -1 & 1 \end{bmatrix}$$

总体节点通量列阵为

$$\boldsymbol{f} = \sum_{i=1,2} (L^{(i)})^T f^{(i)} = 500 \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix}$$

有限元求解：将总体热传导、总体节点通量列阵、总体节点温度列阵等带入弱形式

$$\boldsymbol{t}=\begin{bmatrix}t^{(1)}\\t^{(2)}\\t^{(3)}\end{bmatrix}=\begin{bmatrix}0\\t^{(2)}\\t^{(3)}\end{bmatrix},\quad \boldsymbol{r}=\begin{bmatrix}r^{(1)}\\0\\0\end{bmatrix}$$

$$\boldsymbol{Kt}=\tfrac{1}{2}\begin{bmatrix}1&-1&0\\-1&2&-1\\0&-1&1\end{bmatrix}\begin{bmatrix}0\\t^{(2)}\\t^{(3)}\end{bmatrix}=\boldsymbol{f}+\boldsymbol{r}=500\begin{bmatrix}1\\2\\1\end{bmatrix}+\begin{bmatrix}r^{(1)}\\0\\0\end{bmatrix}$$

本质边界条件处的通量 $r^{(1)}$ 是未知的，我们使用缩减法求解上述线性方程组：先求解非本质边界条件处的温度

$$\frac{1}{2}\begin{bmatrix}2&-1\\-1&1\end{bmatrix}\begin{bmatrix}t^{(2)}\\t^{(3)}\end{bmatrix}=$$

$\boldsymbol{T^{(2)}}=\boldsymbol{N^{(2)}L^{(2)}t}=3000+100x$

计算单元中温度场的微分：

$$\frac{dT^{(1)}}{dx}=300$$
$$\frac{dT^{(2)}}{dx}=100$$

绘制有限元求解的温度场、温度场的微分，以及精确温度场和其微分的图像如下

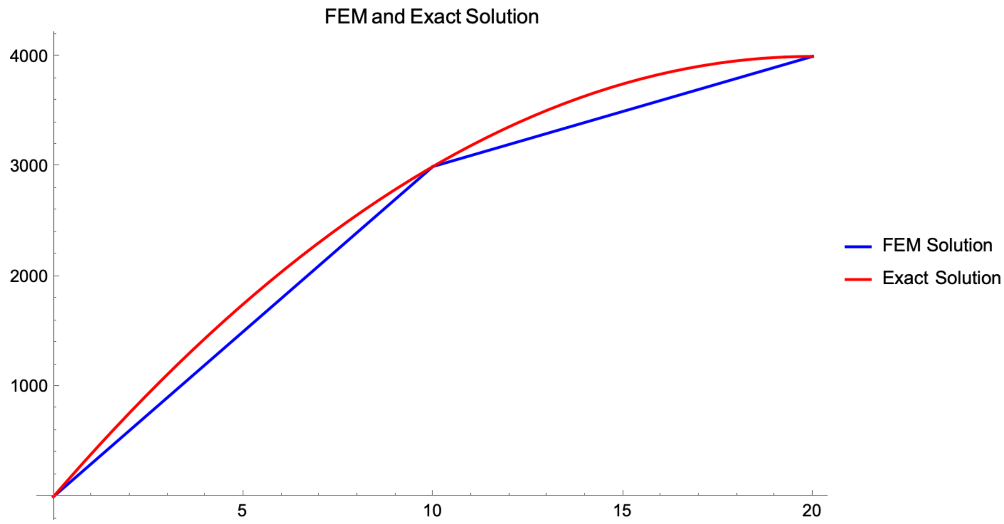

图 1: 有限元温度场和精确温度场

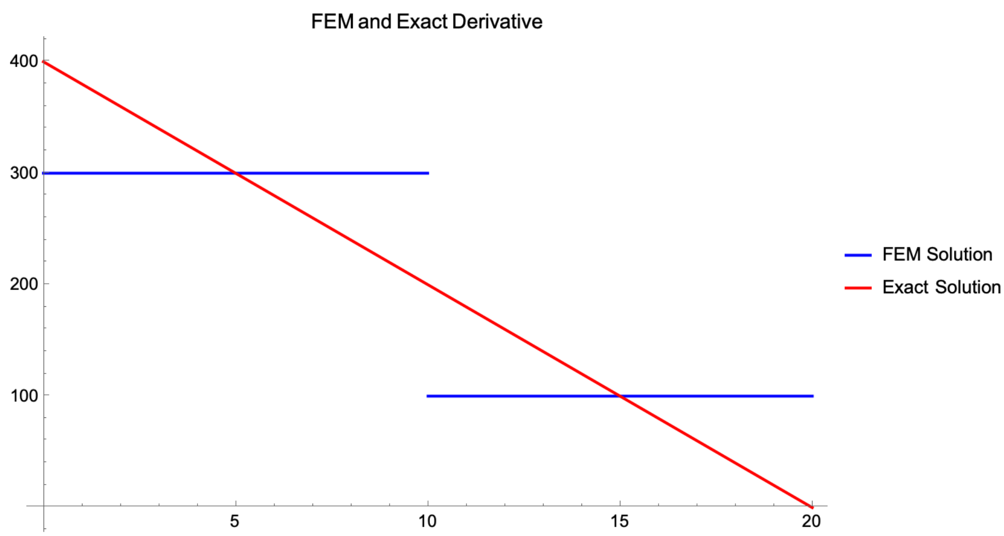

图 2: 有限元温度场和精确温度场微分

1.2 4,8,16 个等长线性单元求解

将原来的杆件分为 4，8，16 个相同的线性单元，使用 bar1D-python 程序求其温度场和温度场的微分 $\frac{dT}{dx}$ 并绘制有限元解和精确解。我们首先在 exact.py 中添加热传导模型的精确解绘制函数:

```python
def ExactSolution_HeatConduction(ax1, ax2):
"""
Plots the exact temperature and derivative of temperature of a elastic bar under
stable heat source in axes ax1 and ax2, respectively.

Args:
    ax1 : axis to draw temperature distribution
    ax2 : axis to draw derivative distribution
"""
# bar parameters
length = 20.0

# region of the bar
x = np.arange(0, length, length / 100)
# exact temperature field
T = -10 * np.power(x, 2) + 400 * x

# plot temperature
ax1.plot(x, T, '--r', label='Exact')

# exact temperature derivative for x
dT = -20 * x + 400

# plot temperature derivative
ax2.plot(x, dT, '--r', label='Exact')
```

并在 Prepost.py 的 postprocessor() 函数中添加该函数的调用:

```python
# plot the exact solution
if model.Exact == "TaperedBar":
    ExactSolution_TaperedBar(ax1,ax2)
elif model.Exact == "CompressionBar":
    ExactSolution_CompressionBar(ax1,ax2)
elif model.Exact == "ConcentratedForce":
    ExactSolution_ConcentratedForce(ax1, ax2)
elif model.Exact == "HeatConduction":
```

```
        ExactSolution_HeatConduction(ax1, ax2)
    elif model.Exact != None:
        print('Error: Exact solution for %s is not available'%(model.Exact))
```

另外，由于 bar1D-python 程序被设计用于计算线弹性问题，因此我们可以通过稍加更改应力的计算，达到直接输出 $\frac{dT}{dx}$ 的目的。以下代码块来自 PrePost.py 中的 disp_and_stress() 函数，在最后一行 stress[i] 处，删除了 B@de 之前乘的杨氏模量 Ee：

```
   # compute displacement and stresses
    displacement = np.zeros(model.nplot)
    stress = np.zeros(model.nplot)
    for i in range(model.nplot):
        xi = xplot[i]               # current coordinate
        N  = Nmatrix1D(xi,xe)       # shape functions
        B  = Bmatrix1D(xi,xe)       # derivative of shape functions

        Ee = N@model.E[IENe]       # Young's modulus
        displacement[i] = N@de      # displacement output
        stress[i]       = B@de   # stress output s
```

按照规则编写 4，8，16 单元的.json 文件，即可计算并绘制有限元温度场与精确温度场的图线：

- 4 单元图线：

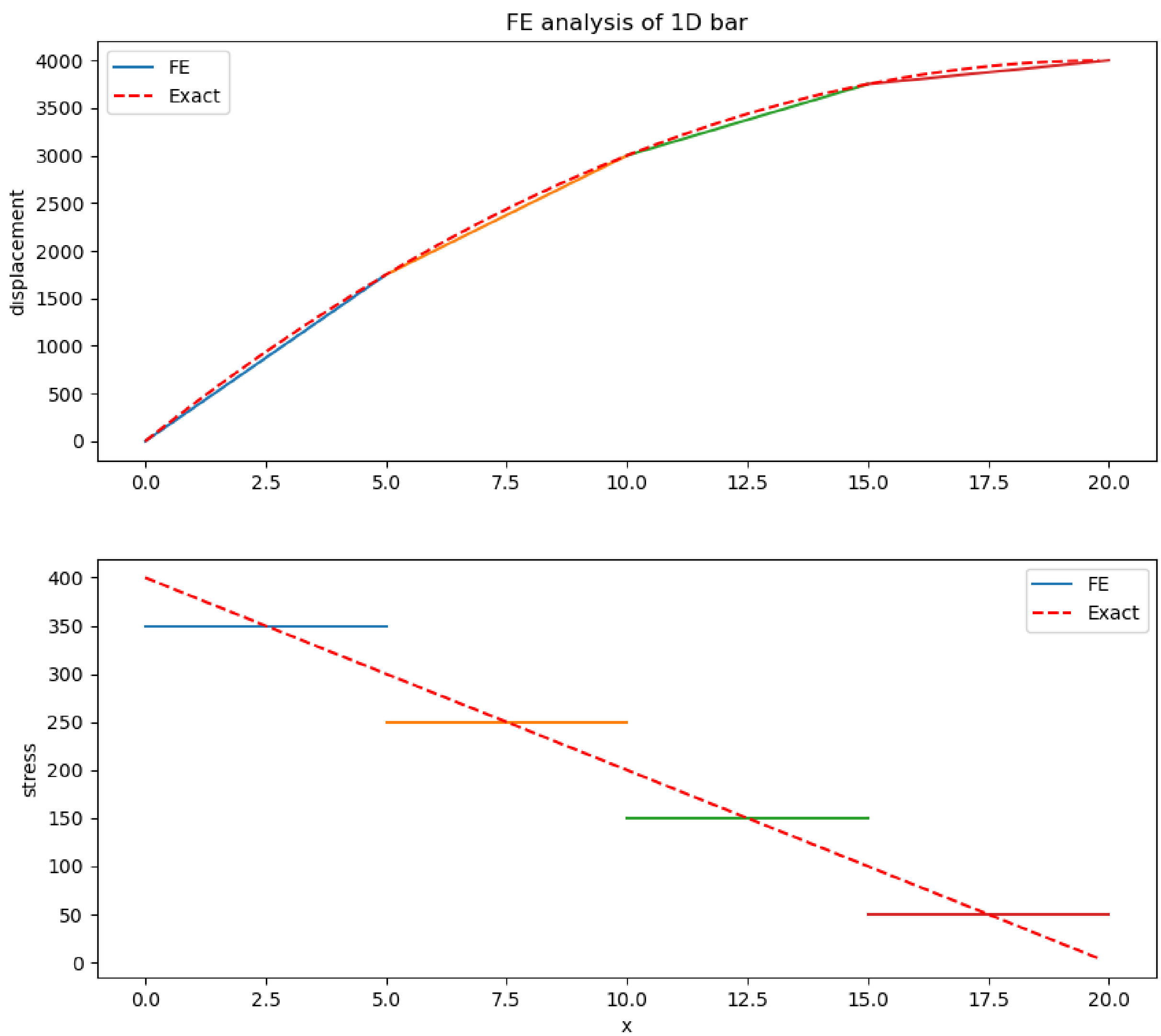

图 3: 4 节点温度场、温度场微分

- 8 单元图线：

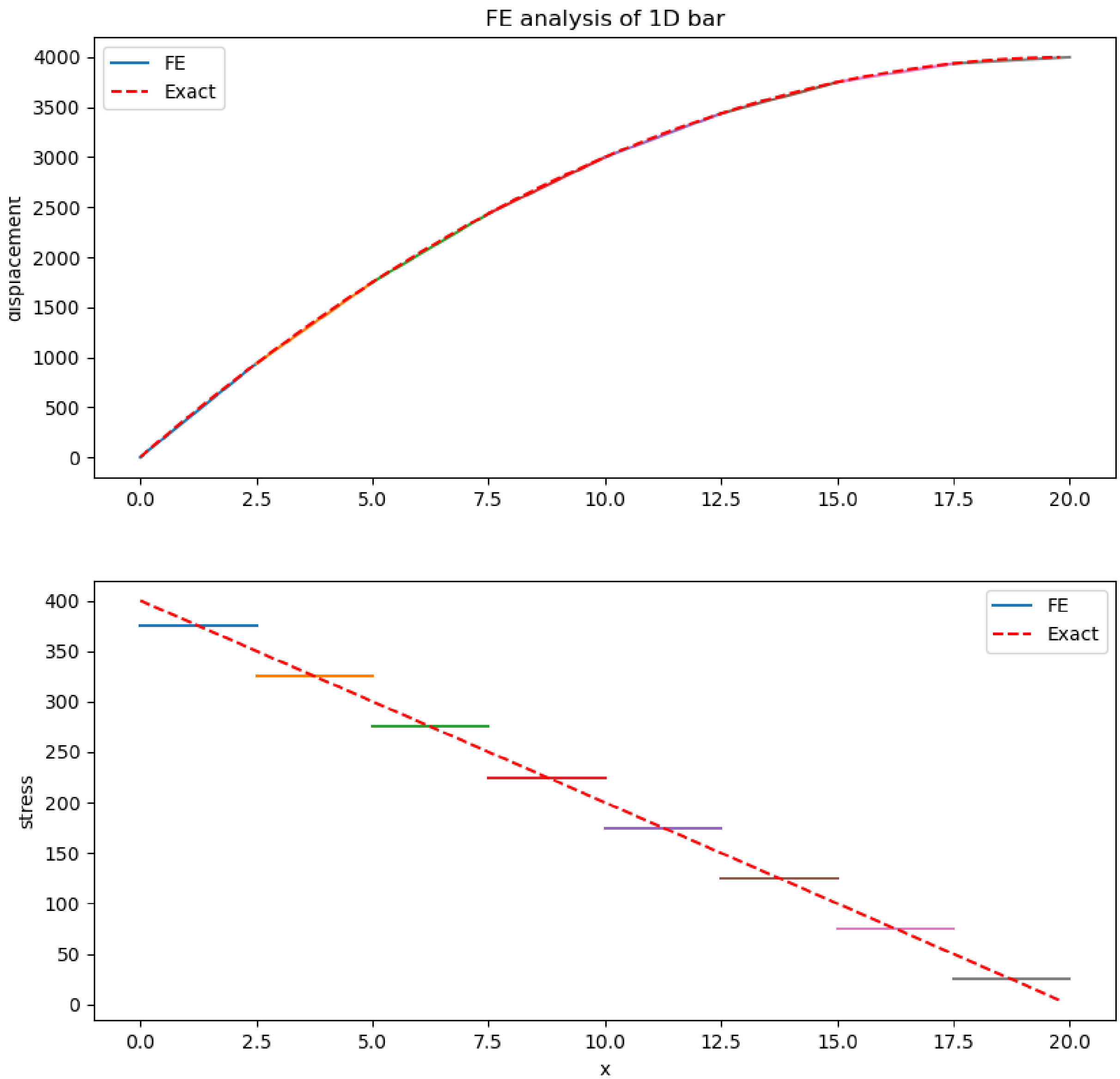

图 4: 8 节点温度场、温度场微分

- 16 单元图线：

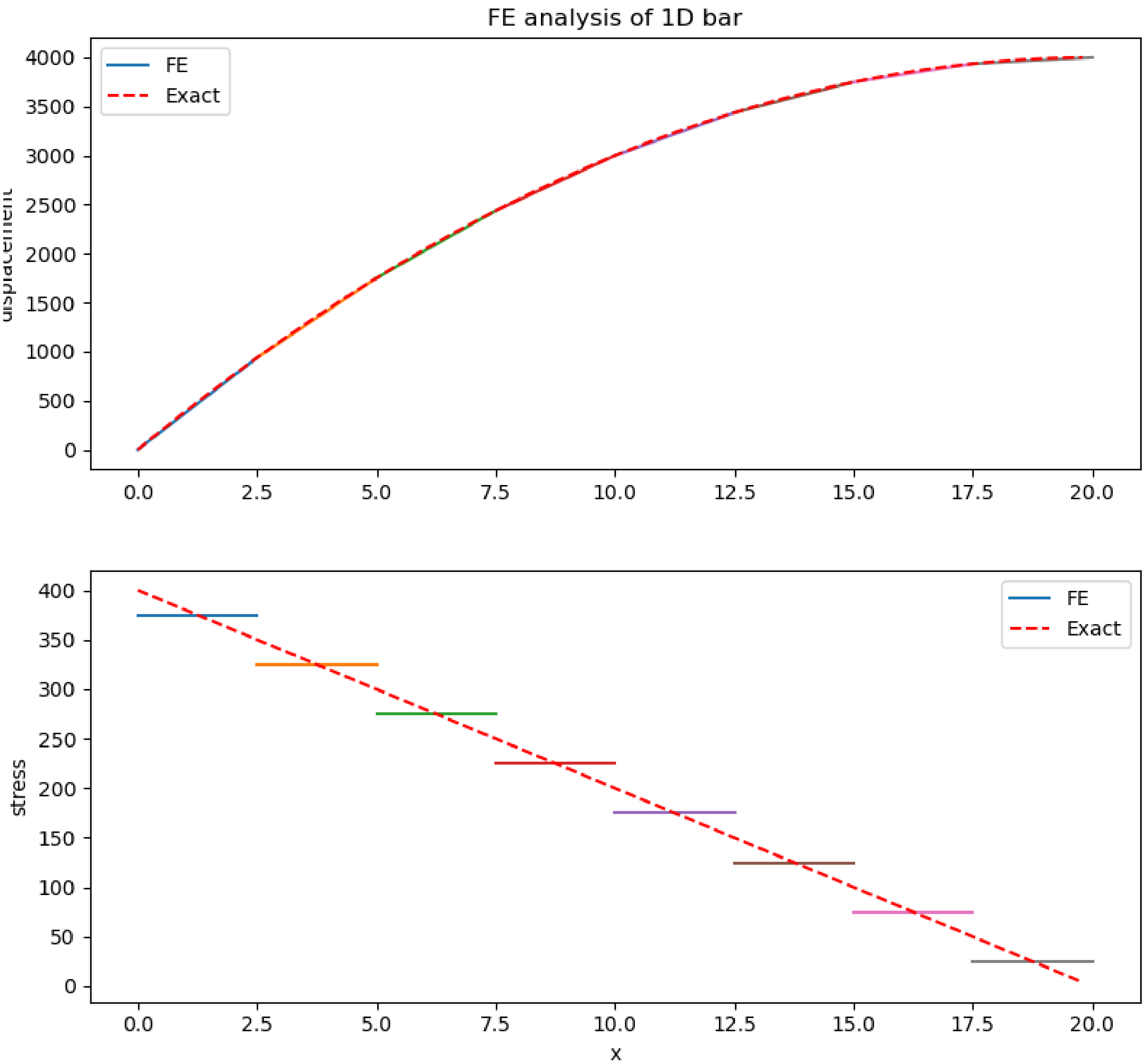

图 5: 16 节点温度场、温度场微分

我们仿照 bar1D-python 中的 ConvergeCompressionBar.py 程序，计算一维热传导问题的 L2 范数误差和自然边界条件误差。首先需要在原程序的 Exact.py 中添加适用于一维热传导问题的误差计算函数：

```python
def ErrorNorm_HeatConduction():
"""
Calculate and print the error norm (L2 norm) of the elastic
bar under heat conduction in practice 3.17 for convergence study
"""

ngp = 3
[w, gp] = gauss(ngp)    # extract Gauss points and weights

bar_length = 20         # total length of the bar

L2Norm = 0
natural_bc_error = 0   # nature boundary condition error, FE solution at nature boundary - exact nature boundary

L2NormEx = 0


for e in range(model.nel):

    de = model.d[model.LM[:,e]-1] # extract element nodal displacements
    IENe = model.IEN[:,e]-1       # extract local connectivity information
    xe = model.x[IENe]            # extract element x coordinates
    J = (xe[-1] - xe[0])/2        # compute Jacobian
    print("Jacobina of element ",e, "is: ",J)

    for i in range(ngp):
        xt = 0.5*(xe[0]+xe[-1])+J*gp[i]  # Gauss points in physical coordinates

        N = Nmatrix1D(xt,xe)      # shape functions matrix

        uh  = N@de                # temperature at gauss point
        uex = -10 * np.power(xt, 2) + 400 * xt # Exact temperature
        L2Norm += J*w[i]*(uex - uh)**2
        L2NormEx += J*w[i]*(uex)**2

```

```python
        if e == model.nel - 1:
            B = Bmatrix1D(bar_length / model.nel, xe)     # derivative of shape functions matrix

            Ee = N@model.E[IENe]     # Heat Conductiion coefficient k at element gauss points


            dT  = B@de               # derivative of temperature at Gauss points
            q_h = -1 * Ee * dT
            q_bc = 0                    # Exact natural boundary condition

            natural_bc_error = np.abs(q_h - q_bc)       # natural bc at the end of bar is 0 in practice 3.17

    L2Norm = sqrt(L2Norm)



    # print stresses at element gauss points
    print('\nError norms')
    print('%13s %13s %13s %13s '
          %('h','L2Norm','L2NormRel','NaturalBC'))
    print('%13.6E %13.6E %13.6E %13.6E\n'
          %(bar_length / model.nel, L2Norm, L2Norm/L2NormEx, natural_bc_error))

    return bar_length / model.nel, L2Norm, natural_bc_error
```

以及 ConvergeHeatConduction.py 主程序，用于调用计算函数、绘图、图像斜率计算：

```python
    #!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Convergence analysis for 2L bar element using the bar under heat conduction in practice 3.17.
Plot the element length - L2 norm cuvers in logarithm scale for both
linear and quadratic elements, and obtain their convergence rates and the
intercepts by linear regression.

```

```python
Created on Thu Apr 30 21:05:47 2020

@author: xzhang
"""

from Bar1D import FERun
from Exact import ErrorNorm_HeatConduction

import numpy as np
import matplotlib.pyplot as plt

# Json data files for 2L element
files_2L = ("bar_3_17_2.json", "bar_3_17_4.json", "bar_3_17_8.json", "bar_3_17_16.json")



# Run FE analysis for all files using 2L element
n2L = len(files_2L)
h2 = np.zeros(n2L)
L2Norm2 = np.zeros(n2L)
natural_bc_error = np.zeros(n2L)
for i in range(n2L):
    FERun(files_2L[i])

    # Calculate error norms for convergence study
    h2[i], L2Norm2[i], natural_bc_error[i] = ErrorNorm_HeatConduction()

'''
# plot the element length - error norm curve in logarithmic scale
fig,(axs) = plt.subplots(2,2)
plt.tight_layout()

axs[0,0].set_title('Linear element', fontsize=9);
axs[0,0].set_ylabel('L_2 error', fontsize=8)
axs[0,0].xaxis.set_tick_params(labelsize=7)
axs[0,0].yaxis.set_tick_params(labelsize=7)
axs[0,0].set_xscale('log')
axs[0,0].set_yscale('log')
axs[0,0].plot(h2,L2Norm2)

```

```
plt.show()'''


fig, axs = plt.subplots(2, 1, figsize=(8, 10))

# L2-Norm error and h figure
axs[0].set_title('Linear element L2 error in logarithmic scale', fontsize=10)
axs[0].set_ylabel('L2 error', fontsize=9)
axs[0].set_xscale('log')
axs[0].set_yscale('log')
axs[0].plot(h2, L2Norm2, marker='o')
axs[0].grid(True, which="both", ls="--")
slope_L2, intercept_L2 = np.polyfit(np.log(h2), np.log(L2Norm2), 1)
print(f"L2 Error Linear Regression (log-log scale): Slope = {slope_L2},
    Intercept = {np.exp(intercept_L2)}")

# Natural BC error and h figure
axs[1].set_title('Natural BC error with element length h', fontsize=10)
axs[1].set_xlabel('Element length h', fontsize=9)
axs[1].set_ylabel('Natural BC error', fontsize=9)
axs[1].plot(h2, natural_bc_error, marker='o', color='red')
axs[1].grid(True)
slope_nbc, intercept_nbc = np.polyfit(h2, natural_bc_error, 1)
print(f"Natural BC Error Linear Regression: Slope = {slope_nbc}, Intercept = {
    intercept_nbc}")

plt.tight_layout()
plt.show()


# Linear regression
print("The L2 error norms are ")

a, C = np.polyfit(np.log(h2),np.log(L2Norm2),1)
print("    Linear element    : ||e||_L2 = %e h^%g" %(np.e**C, a))

# Convert matplotlib figures into PGFPlots figures stored in a Tikz file,
# which can be added into your LaTex source code by "\input{fe_plot.tex}"
import tikzplotlib
tikzplotlib.save("fe_convergence.tex")

```

```python

# Print error norms obtained by the linear element
#     with different element size
print("\nError norms of linear elements")
print('%13s %13s %13s' %('h','L2Norm','NaturalBC'))
for i in range(len(h2)):
    print('%13.6E %13.6E %13.6E' %(h2[i], L2Norm2[i], natural_bc_error[i]))
```

使用上述程序经计算，并得到了 L_2 范数误差以及自然边界条件的误差：

```
./ConvergeHeatConduction.py
...
    Error norms
            h        L2Norm     L2NormRel     NaturalBC
 1.250000E+00  1.275776E+01  7.475249E-08  6.250000E+01

L2 Error Linear Regression (log-log scale): Slope = 2.0000000000018296,
    Intercept = 8.164965809249374
Natural BC Error Linear Regression: Slope = 49.99999999999999, Intercept =
    -1.4261976419955643e-13
qt.xkb.compose: failed to create compose table
The L2 error norms are
    Linear element    : ||e||_L2 = 8.164966e+00 h^2

Error norms of linear elements
            h        L2Norm     NaturalBC
 1.000000E+01  8.164966E+02  5.000000E+02
 5.000000E+00  2.041241E+02  2.500000E+02
 2.500000E+00  5.103104E+01  1.250000E+02
 1.250000E+00  1.275776E+01  6.250000E+01
```

以及绘制的图像:

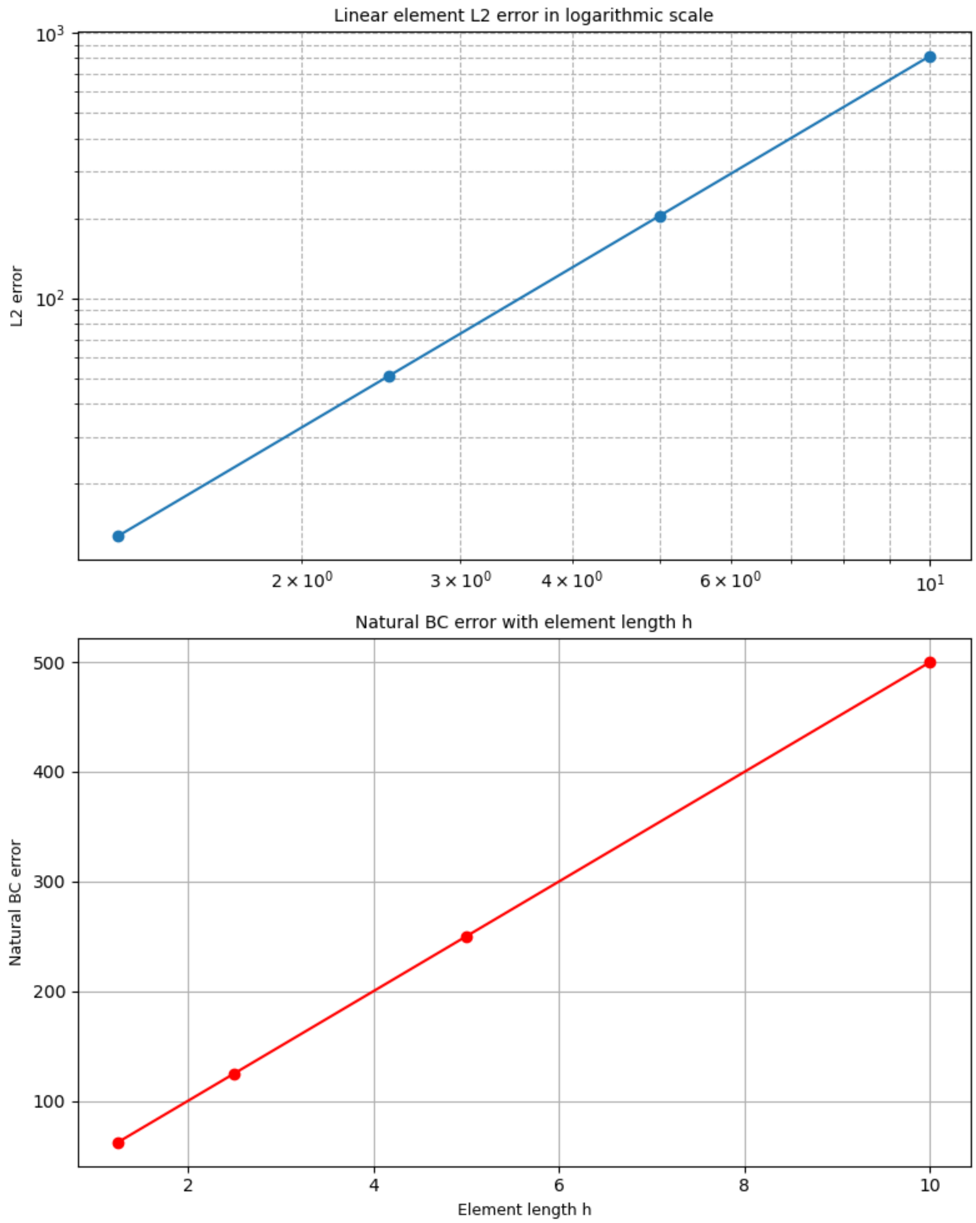

图 6: L2, 自然边界条件误差-单元长度曲线

从趋势上看，两个误差都随网格的加密（单元长度减小）而下降，图线的斜率代表收敛率。其中 L2 范数误差与单元长度对数满足线性的收敛过程，其斜率约为 2，与线性单元收敛的理论值几乎完全相等，自然边界条件误差与单元长度之间也满足线性的收敛，收敛率约为 50。