Fin System, Inc.

Company Report

Temperature Profile Calculators

Team 1
J. C. Stewards, Lead
A. B. Williams, Documentation
M. D. Daily, Programmer

Submitted in Fulfillment of Management Requirements
August 26, 2018

Definition and Information

The company Fin Systems, Inc. produces rectangular extensions as heat transfer conduits between two temperature baths characteristically at T_o and T_L. The fins have different sizes by width w, measured thickness t_m, and length between baths L. They are made of different materials with thermal conductivities k and are used in different external conditions of temperature T_∞ and convection coefficient h. The company has two pressing needs. First, they want a way to predict the behavior of an extension when all operating conditions are specified. In particle, they want to produce a graph of the temperature profile is along the entire extension. Secondly, they want a function that will allow their field engineers to confirm the operation of the extension. In particular, given all the input conditions, they want a field engineer to be able to measure the temperature T at certain distances x along the extension and compare it with expectations.

Provide the code to solve the problems. Test it for input parameters below.

geometry: $L = 1$ m, $t_m = 1$ mm, $w = 50$ cm
temperatures: $T_o = 100$ ^{o}C, $T_\infty = 20$ ^{o}C, $T_L = 30$ ^{o}C
material and conditions: $k = 400$ W/m ^{o}C, $h = 10$ W/m^2 ^{o}C

Proposed Approach

The temperature profile along an extension with fixed end temperatures is given by

$$\Theta = (\Theta_L - \exp(-mL)) \left(\frac{\exp(mx) - \exp(-mx)}{\exp(mL) - \exp(-mL)} \right) + \exp(-mx)$$

with the following definitions:

$$\Theta = (T - T_\infty) / (T_o - T_\infty) \qquad m^2 = hP/kA \text{ where } P = 2(t_m + w) \text{ and } A = t_m w$$

The first demand can be met by creating a function that will graph Θ versus x/L given all inputs. The second demand can be met by creating a function that will return T given all inputs.

Results - Igor Pro

The plotting function has this format

```
plot_ThetavsZL(L, tm, w, h, k, thetaL)
```

The inputs a fin length, measured thickness, width, convection coefficient, thermal conduction coefficient, and relative temperature at the end. It produces a plot titled with the value of m and θ_L.

A plot of Θ versus x/L is provided in Figure 1 for the conditions specified. In this case, $\Theta_L = (30-20)/(100-20) = 0.125$. The function call from the command line was

```
plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
```

The calculation function has this format

```
calc_Tofz(L, tm, w, h, k, z, To, Tinf, TL)
```

It returns the value of T for the input conditions. A test result from the command line gives

```
print calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30) -> 21.73
```

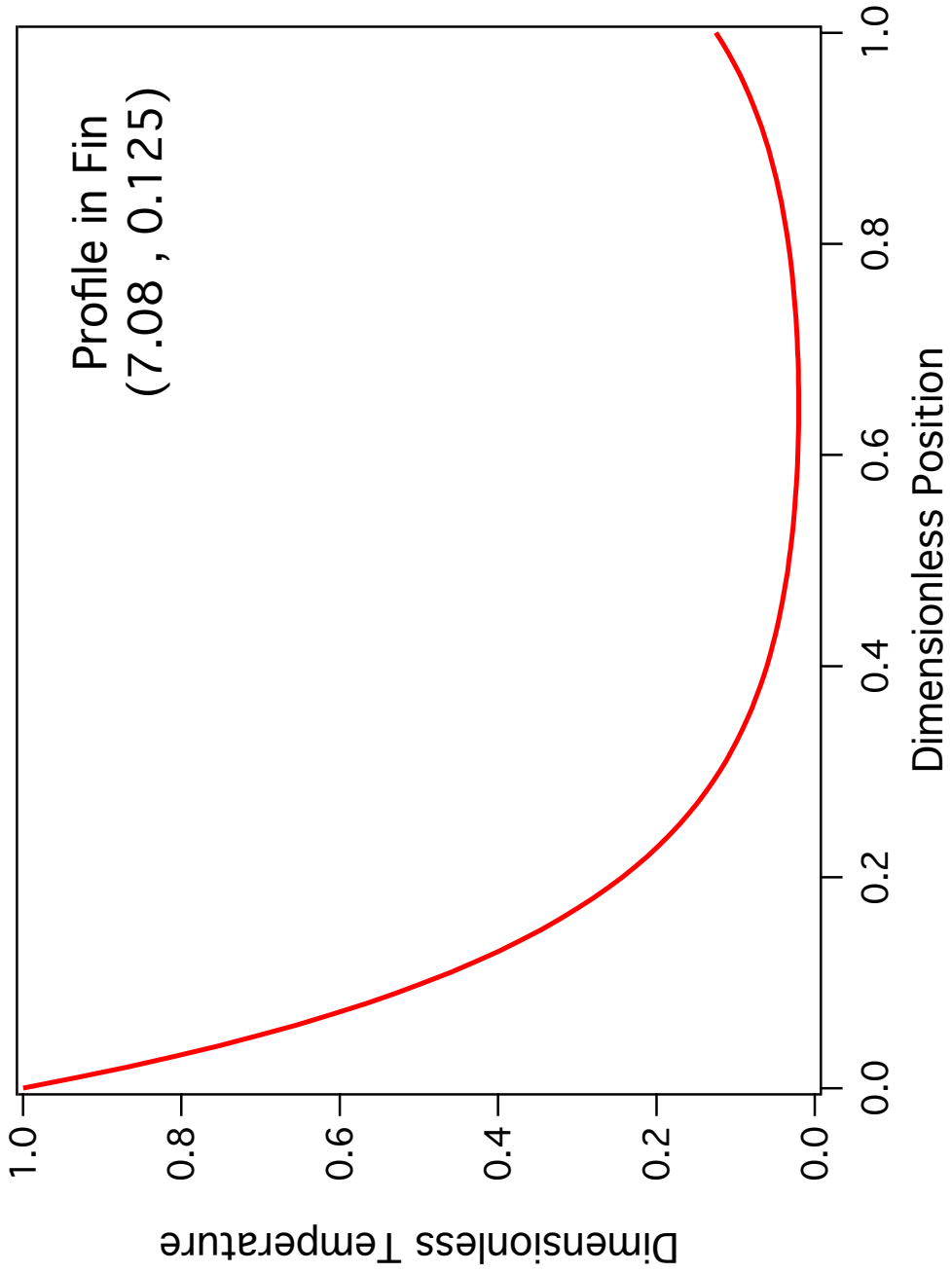

Figure 1: Plot of dimensionless temperature versus dimensionless distance using Igor Pro. Specifications and the approach used are given in the text. Values for (m, Θ_L) are given on the plot.

Code - Igor Pro

The code was generated in Igor Pro (WaveMetrics, Inc) version 7.08. The functions are documented in line.

```
// Temperature Profiler for an Extended Surface (Fin)
// version 18.08.26
// j j weimer

// The functions in this procedure calculate, return, and
// display the temperature across an extended surface (fin).
// The temperature at the end of the fin is pinned.

// Igor Pro Settings

#pragma rtGlobals=3

// Functions

// plot_ThetavsZL(Le, tm, w, h, k, thetaL)
// inputs
// Le - length of extension
// tm - measured thickness
// w - width
// h - convection coefficient
// k - thermal conductivity
// thetaL - relative temperature at end of fin
// returns a plot of Theta(z/L)

Function plot_ThetavsZL(Le, tm, w, h, k, thetaL)
  variable Le, tm, w, h, k, thetaL

  // local variables
  variable P, A, m, mL
  string plottxt

  // perimeter, area, and m
  P = 2*(tm + w)
  A = tm*w
  m = sqrt(h*P/(k*A))

  // dimensionless values
```

```
  mL = m*Le

  // create the wave (overwrite as needed) and set its x-scaling
  make/O/N=101/D theta
  SetScale/P x, 0, 0.01, theta

  // fill the wave and display it
  theta = f_Theta(x/Le, mL, thetaL)
  sprintf plottxt,"Dimensionless Temperture at (%3.2f , %3.3g)", mL, thetaL
  display theta as plottxt

  // clean up the graph
  SetAxis left, 0, 1
  SetAxis bottom, 0, 1
  ModifyGraph mirror=2,fSize=14
  Label left "\\Z16Dimensionless Temperature"
  Label bottom "\\Z16Dimensionless Position"
  ModifyGraph width=432,height={Aspect,0.75}, lsize=2
  sprintf plottxt,"\\Z18\\JCProfile in Fin\r(%3.2f , %3.3g)", mL, thetaL
  TextBox/C/N=text0/F=0 plottxt
  return 0

end

// calc_Tofz(Le, tm, w, h, k, z, To, Tinf, TL)
// inputs
// Le - length of extension
// tm - measured thickness
// w - width
// h - convection coefficient
// k - thermal conductivity
// z - position
// To - wall temperature
// Tinf - external temperature
// TL - temperture at end of fin
// returns T(z)

Function calc_Tofz(Le, tm, w, h, k, z, To, Tinf, TL)
  variable Le, tm, w, h, k, z, To, Tinf, TL

  // local variables
  variable P, A, m, zdim, mL, thetaL, theta, Trtn
```

```
  // perimeter, area, and m
  P = 2*(tm + w)
  A = tm*w
  m = sqrt(h*P/(k*A))

  // dimensionless values
  zdim = z/Le
  mL = m*Le
  thetaL = (TL - Tinf)/(To - Tinf)

  // dimensionless temperature
  theta = f_Theta(zdim, mL, thetaL)

  // actual temperature
  Trtn = theta*(To - Tinf) + Tinf

  return (Trtn)

end

// f_Theta(zdim, mL, thetaL)
// inputs: position, convection/conduction ratio, end temperature
// returns: theta
Function f_Theta(zdim, mL, thetaL)
  variable zdim, mL, thetaL

  variable term1 = thetaL - exp(-mL)
  variable num = exp(mL*zdim) - exp(-mL*zdim)
  variable den = exp(mL) - exp(-mL)

  variable rtn = term1*(num/den) + exp(-mL*zdim)

  return rtn

end
```

The command line inputs to generate the results were

```
plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
print calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30)
```

Results - Python

The plotting function has this format

```
plot_ThetavsZL(L, tm, w, h, k, thetaL)
```

The inputs a fin length, measured thickness, width, convection coefficient, thermal conduction coefficient, and relative temperature at the end. It produces a plot titled with the value of m and θ_L.

A plot of Θ versus x/L is provided in Figure 2 for the conditions specified. In this case, $\Theta_L = (30-20)/(100-20) = 0.125$. The function call was

```
plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
```

The calculation function has this format

```
calc_Tofz(L, tm, w, h, k, z, To, Tinf, TL)
```

It returns the value of T for the input conditions. A test result gives

```
calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30) -> 21.73
```

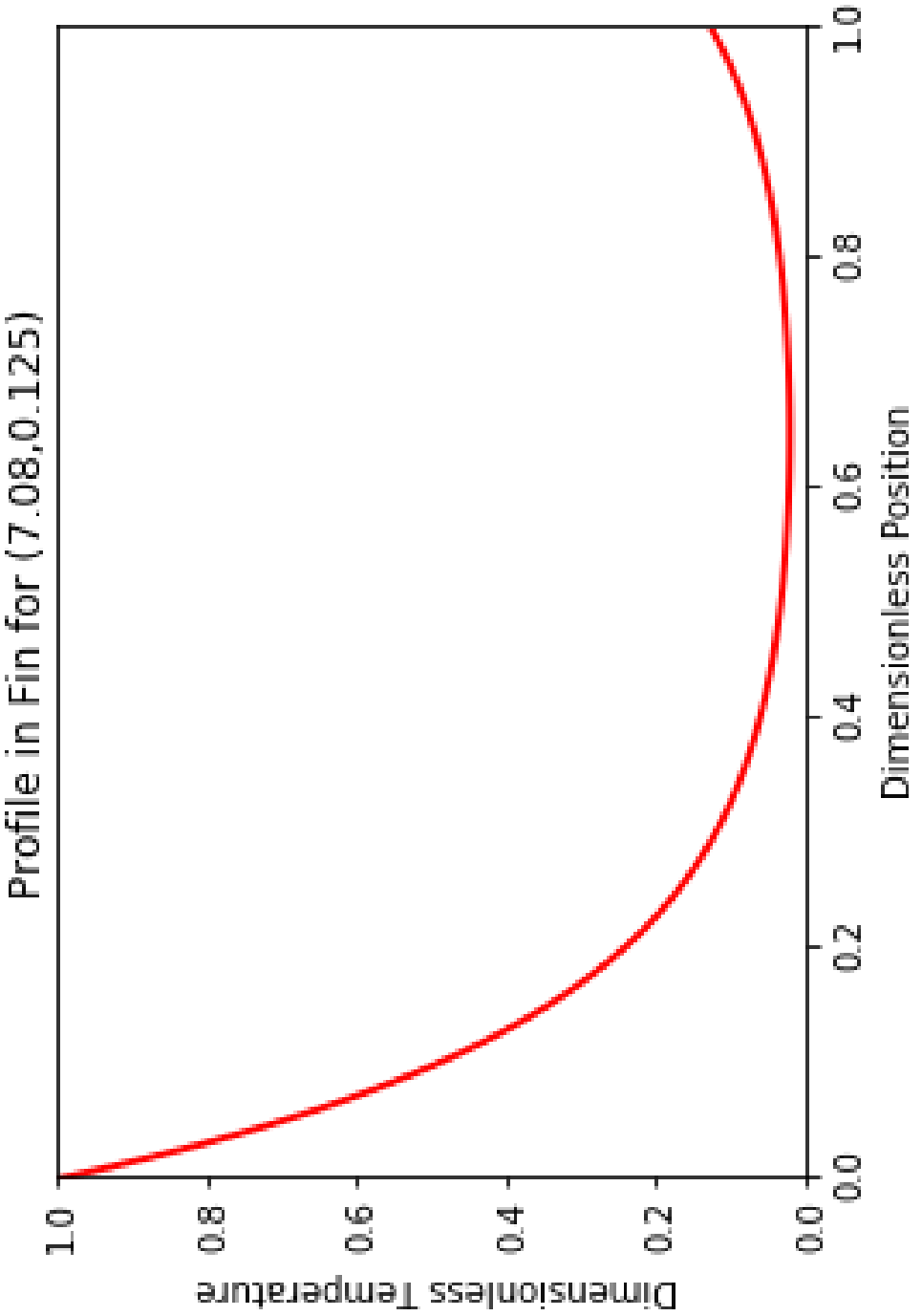

Figure 2: Plot of dimensionless temperature versus dimensionless distance produced using python (in a Jupyter notebook). Specifications and the approach used are given in the text. Values for (m, Θ_L) are given on the plot.

Code - Python

The code was generated in a Jupyter notebook. The definitions are documented in line.

Temperature Profiler for an Extended Surface (Fin)
version 18.08.26
author j j weimer

The functions in this procedure calculate, return, and display the temperature across an extended surface (fin). The temperature at the end of the fin is pinned.

```
# import modules
import math
import numpy as np
import matplotlib.pyplot as plt

# global configuration parameters for graph
startx = 0; starty = 0
endx = 1; endy = 1
npx = 101; npy = 101

# global x, y arrays for graph
x = np.linspace(startx,endx,npx)
y = np.linspace(starty,endy,npy)

# calculate temperature at position
def calc_Tofz(Le, tm, w, h, k, z, To, Tinf, TL):

    # perimeter, area, and m
    P = 2*(tm + w)
    A = tm*w
    m = math.sqrt(h*P/(k*A))

    # dimensionless values
    zdim = z/Le
    mL = m*Le
    thetaL = (TL - Tinf)/(To - Tinf)

    # dimensionless temperature
```

```
    theta = f_Theta(zdim, mL, thetaL)

    # actual temperature
    Trtn = theta*(To - Tinf) + Tinf

    return Trtn

# generate the dimensionless temperature
def f_Theta(zdim, mL, thetaL):

    term1 = thetaL - np.exp(-mL)
    num = np.exp(mL*zdim) - np.exp(-mL*zdim)
    den = np.exp(mL) - np.exp(-mL)
    rtn = term1*(num/den) + np.exp(-mL*zdim)
    return rtn

# plot the dimensionless temperature profile
def plot_ThetavsZL(Le, tm, w, h, k, thetaL):

    # perimeter, area, and m
    P = 2*(tm + w)
    A = tm*w
    m = math.sqrt(h*P/(k*A))

    # dimensionless values
    mL = m*Le

    # generate y vs x data
    #global x, y
    y = f_Theta(x, mL, thetaL)

    # create plot
    plt.plot(x,y,color='red',linewidth=2)
    plt.axis((startx,endx,starty,endy))
    plottxt = "Profile␣in␣Fin␣for␣({:.2f},{:.3f})".format(mL,thetaL)
    plt.title(plottxt)
    plt.xlabel('Dimensionless␣Position')
    plt.ylabel('Dimensionless␣Temperature')

    plt.show()

plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30)
```

Results - Maple

The plotting function has this format

```
plot_ThetavsZL(L, tm, w, h, k, thetaL)
```

The inputs a fin length, measured thickness, width, convection coefficient, thermal conduction coefficient, and relative temperature at the end. It produces a plot titled with the value of m and θ_L.

A plot of Θ versus x/L is provided in Figure 3 for the conditions specified. In this case, $\Theta_L = (30-20)/(100-20) = 0.125$. The function call from the command line was

```
plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
```

The calculation function has this format

```
calc_Tofz(L, tm, w, h, k, z, To, Tinf, TL)
```

It returns the value of T for the input conditions. A test result from the command line gives

```
calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30) -> 21.73
```

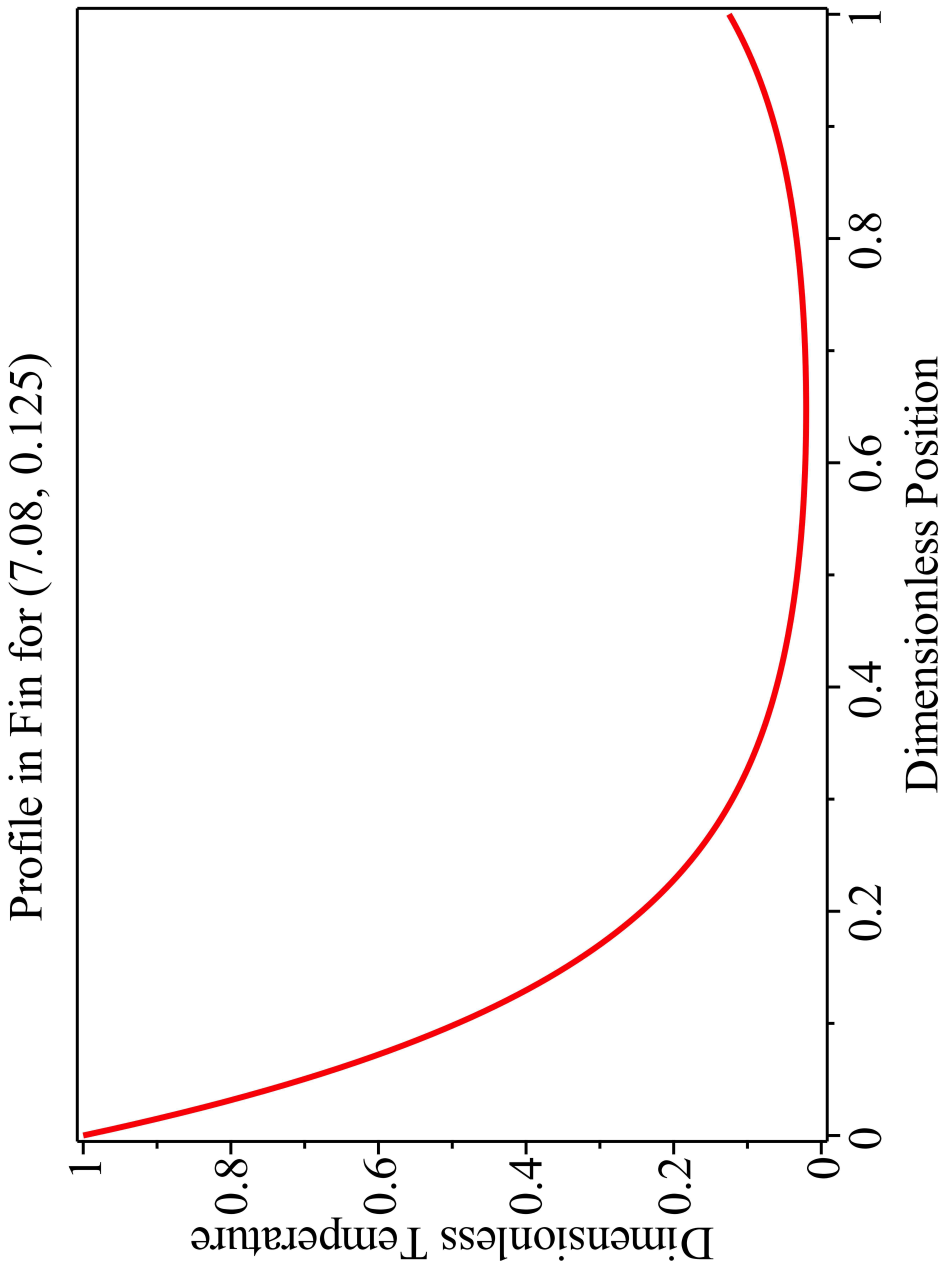

Figure 3: Plot of dimensionless temperature versus dimensionless distance using Maple. Specifications and the approach used are given in the text. Values for (m, Θ_L) are given on the plot.

Code - Maple

The formatted pages of workbook code from Maple are included directly in the pages that follow.

Capstone Example

Functions

f_Theta(zdim, mL, thetaL)

inputs: dimensionless distance, convection/conduction ratio, and end temperature
output: dimensionless temperature at position

$f\_Theta := \mathbf{proc}(zdim, mL, thetaL)$

local *term1*, *num*, *den*, *rtn* :

$term1 := thetaL - \exp(-mL)$:

$num := \exp(mL \cdot zdim) - \exp(-mL \cdot zdim)$:

$den := \exp(mL) - \exp(-mL)$:

$rtn := term1 \cdot \left(\frac{num}{den} \right) + \exp(-mL \cdot zdim)$:

return *rtn*

end proc:

calc_TofZ(Le, tm, w, h, k, z, To, Tinf, TL)

inputs:
length, measured thickness, width
convection coefficient, thermal conductivity
position
start temperature, external temperature, end temperature
output: temperature at z

$calc\_TofZ := \mathbf{proc}(Le, tm, w, h, k, z, To, Tinf, TL)$

local *P*, *A*, *m*, *zdim*, *mL*, *thetaL*, *thetaA*, *Trtn* :

perimeter, area, and m

$P := 2 \cdot (tm + w)$:

$A := tm \cdot w$:

$m := \sqrt{\frac{h \cdot P}{k \cdot A}}$:

 # dimensionless values

$zdim := \frac{z}{Le}$:

$mL := m \cdot Le$:

$thetaL := \frac{TL - Tinf}{To - Tinf}$:

 # dimensionless and actual temperature

$thetaA := f\_Theta(zdim, mL, thetaL)$:

$Trtn := thetaA \cdot (To - Tinf) + Tinf$:

return *Trtn*

end proc:

plot_ThetavsZL(Le, tm, w, h, k, thetaL)

inputs:
length, measured thickness, width
convection coefficient, thermal conductivity
dimensionless end temperature
output: plot of dimensionless temperature versus dimensionless position

$plot\_ThetavsZL := \mathbf{proc}(Le, tm, w, h, k, thetaL)$

local $P, A, m, mL, plottxt, xlabel, ylabel, pf$:

 # perimeter, area, and m

$P := 2 \cdot (tm + w)$:

$A := tm \cdot w$:

$m := \sqrt{\frac{h \cdot P}{k \cdot A}}$:

 # dimensionless values

```
    mL := m·Le :

    # create and display the plot
     plottxt := sprintf("Profile in Fin for (%3.2f, %3.3f)", mL, thetaL) :
     xlabel := "Dimensionless Position" :  ylabel := "Dimensionless Temperature" :
    plots:-setoptions(axes = boxed, title = plottxt,
            labels = [xlabel, ylabel],
           labeldirections = [horizontal, vertical],
          size = [440, 330]) :
     pf := plot(f_Theta(x, mL, thetaL), x = 0 ..1, 0 ..1, color = red, thickness = 2) :
     plots:-display(pf)

end proc:
```

Results - MatLab

The plotting function has this format

```
plot_ThetavsZL(L, tm, w, h, k, thetaL)
```

The inputs a fin length, measured thickness, width, convection coefficient, thermal conduction coefficient, and relative temperature at the end. It produces a plot titled with the value of m and θ_L.

A plot of Θ versus x/L is provided in Figure 4 for the conditions specified. In this case, $\Theta_L = (30-20)/(100-20) = 0.125$. The function call from the workspace was

```
plot_ThetavsZL(1, 0.001, 0.5, 10, 400, 0.125)
```

The calculation function has this format

```
calc_Tofz(L, tm, w, h, k, z, To, Tinf, TL)
```

It returns the value of T for the input conditions. A test result from the command line gives

```
calc_Tofz(1, 0.001, 0.5, 10, 400, 0.6, 100, 20, 30) -> 21.73
```

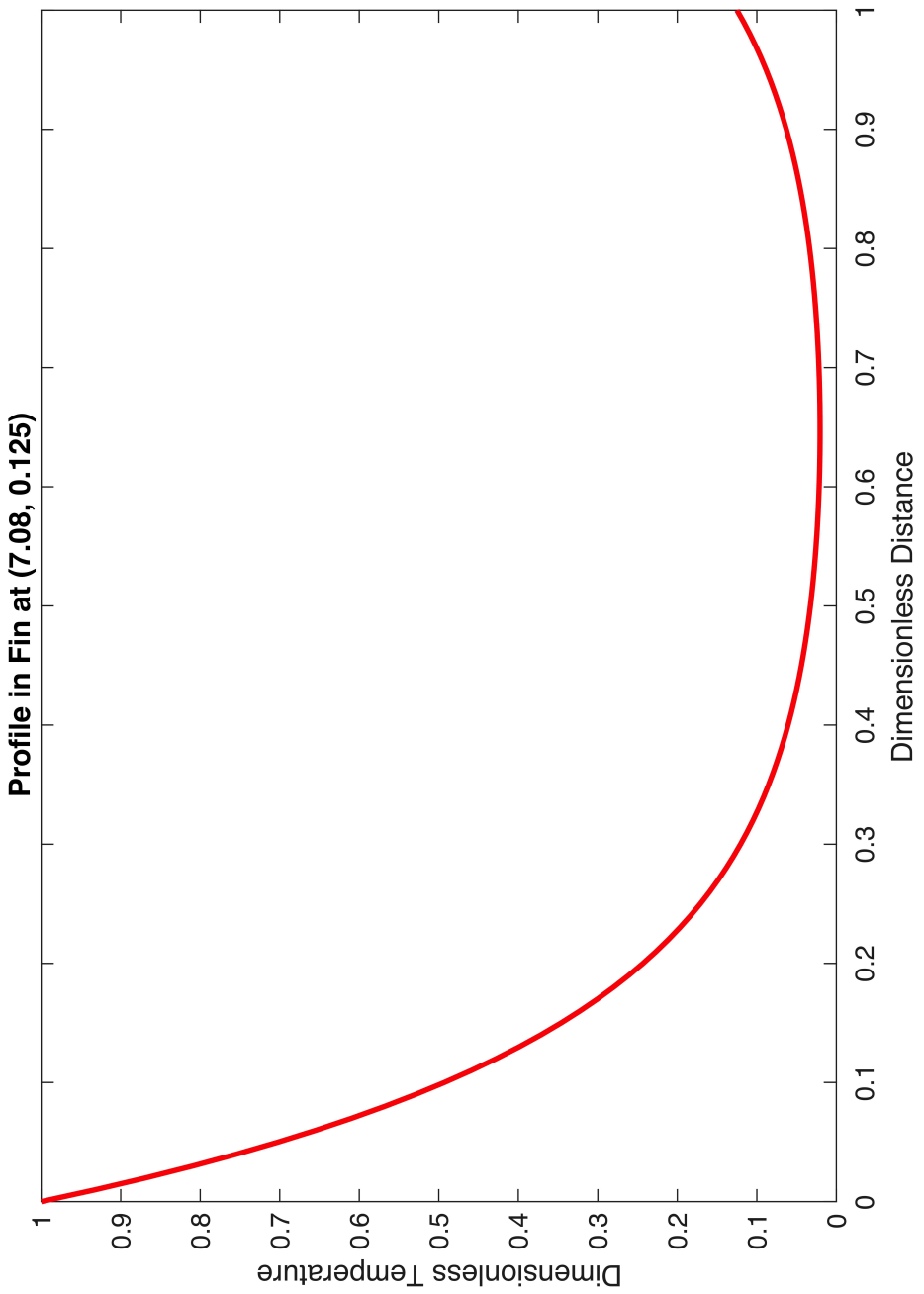

Figure 4: Plot of dimensionless temperature versus dimensionless distance using MatLab. Specifications and the approach used are given in the text. Values for (m, Θ_L) are given on the plot.

Code - Maple

The code was written in three live function files. The files were loaded in an active session.

```
Plot Temperature Profile
This plots the dimensionless temperature profile in the fin.

function pTheta = plot_ThetavsZL(Le, tm, w, h, k, thetaL)

    % perimeter, area, and m
    P = 2*(tm + w);
    A = tm*w;
    m = sqrt(h*P/(k*A));

    % dimensionless values
    mL = m*Le

    % initialize x, y vectors
    x = linspace(0,1,101)

    % generate the curve
    y = f_Theta(x, mL, thetaL);

    % generate plot
    plottxt = sprintf("Profile␣in␣Fin␣at␣(%3.2f,␣%3.3f)",mL,thetaL);
    figure
    pTheta = plot(x,y,'red');
    pTheta.LineWidth = 2;
    ax.FontSize = 16;
    xlim([0,1]);
    ylim([0,1]);
    title(plottxt);
    xlabel('Dimensionless␣Distance');
    ylabel('Dimensionless␣Temperature');

end

Temperature Calculator
Calculates temperature at position in fin

function Trtn = calc_TofZ(Le, tm, w, h, k, z, To, Tinf, TL)

    % perimeter, area, and m
    P = 2*(tm + w);
```

```
    A = tm*w;
    m = sqrt(h*P/(k*A));

    % dimensionless values
    zdim = z/Le;
    mL = m*Le;
    thetaL = (TL - Tinf)/(To - Tinf);

    % dimensionless and actual temperature
    theta = f_Theta(zdim, mL, thetaL);
    Trtn = theta*(To - Tinf) + Tinf;

end

Theta Calculator
Calculates dimensionless temperature at position along fin

function rtn = f_Theta(zdim, mL, thetaL)

    term1 = thetaL - exp(-mL);
    num = exp(mL*zdim) - exp(-mL*zdim);
    den = exp(mL) - exp(-mL);
    rtn = (term1*(num/den)) + exp(-mL*zdim);

end
```

Summary

Coding Language

All four programs use comparable syntax to define functions and operations. Differences are in such aspects as variable declarations (explicitly required in Igor Pro, implicit in others) and types of line endings required. Python requires that specific packages be pre-loaded and commands be prefixed to them in order to access functionality that is otherwise directly accessible in the other platforms.

Ease of Coding and Use for a Novice

I contend the most difficult coding platform for a novice to use is MatLab. Different functions must each be written to their own separate file with file names that are the same as the function. Operations are handled across multiple windows (active functions and editors and plot windows), each having a plethora of button options. I admit that further experience may override this rating, although I suspect not.

By comparison, the single-window approach used in Maple and python affords significant benefits to a novice programmer. One can develop code, run it, and see the output directly all within one single window.

Igor Pro separates graphical output windows (graphs), code windows (procedures), and an input/output window (command line). Once this division is mastered, Igor Pro affords all of the advantages of Maple or python (develop, run, and observe) without those of MatLab.

Plot Quality

For the same level of coding input, I maintain the best quality plots are obtained using Igor Pro. The next best quality plots are obtained by using either python or Maple. Finally, I maintain the worst quality plots for the same coding effort are obtained using MatLab.

For graphics, the significant advantage of using Igor Pro over all three other platforms is that Igor Pro allows the user to change the format of graphs through menu commands and by user interface controls. By example, in Igor Pro, to change the thickness of a line on a graph, click on the line and use a dialog interface to change it (as well as color, line style, and a host of other properties). This means, Igor Pro is the absolute best platform at separating the process of analyzing data with code and generating graphics from the results. One must learn to code in any of the four platforms in

order to analyze data, but one does not need to learn how to code *at all* to make a publication-ready graph of the results in Igor Pro. To be fair, Maple also provides a context-menu list that allows a user to change graph formats without programming. This is cumbersome by comparison.

Advanced Level Efforts

Igor Pro, Maple, and python (in Jupyter) allow a user to include interactive widgets such buttons, variable settings, and sliders. Whether this is possible in MatLab is unexplored at this time.

Igor Pro, Maple, and python (in Jupyter) allow a user to save the effort as a self-contained demonstration package. Whether this is possible in MatLab is unexplored at this time.

Maple allows the user to style the format throughout the workbook, and the workbook can be printed or exported as PDF. Jupyter allows the user to add blocks of formatted text to the document, although obtaining printable output of the entire formatted document is cumbersome to impossible. Igor Pro allows the user to lay out graphical and tabular results in notebooks that can be printed, although the effort is not handled as elegantly as with a Maple workbook.

Recommendations

When you as a novice must learn how to code to analyze data, avoid MatLab. Its use of multiple windows is confusing because it adds extra steps to the cycle of developing, running, and observing code. Choose either python (within Jupyter), Igor Pro, or Maple because you can develop, run, and observe results within one window or within well defined containers (i.e. Igor Pro). When your analysis is to be married with the manipulation of symbolic math equations, use Maple. Its ability to show mathematical expressions in readable form (rather than in "computer language format") is unsurpassed.

When the quality of the graphical output matters, and when you do not want to spend time to learn a host of potentially idiosyncratic commands that are needed to make effective changes to the quality of your graphics, choose Igor Pro or perhaps Maple. None of the platforms that are covered here compare to Igor Pro in the ease of making publication-ready graphs with no need to know how to code.

Finally, when you want to code interactive demonstrations of functions for rapid review, choose python in Jupyter.