# Project 4: Finned Cooling Plate

Nicholas McLaughlin

MECH.5410 – Advanced Heat Transfer

Professor Juan Pablo Trelles

November 10th, 2020

## Table of Contents

## Variables Used

| Variable | Description | Value | Unit |
|---|---|---|---|
| $T = T(i,j)$ | Local temperature of the fluid | N/A | [°C] |
| $T = T(1,j)$ | Fluid temperature directly above the plate | N/A | [°C] |
| $L_x$ | Length of the plate | [0.5 1.5] | [m] |
| $L_f$ | Length of the fins | [0.01 0.10] | [m] |
| $L_y$ | Length of domain along $y$ | 0.2 | [m] |
| $T_0$ | Inflow temperature of the top fluid | 100 | [°C] |
| $U_0$ | Inflow velocity | 0.001 | [m/s] |
| $h$ | Heat transfer coefficient of the top fluid | N/A | [W/m$^2$/K] |
| $k$ | Thermal conductivity of the top fluid | 0.5 | [W/m/K] |
| $\rho$ | Density of the flowing fluid | 800 | [kg/m$^3$] |
| $C_p$ | Flowing fluid heat capacity | 2000 | [J/kg/K] |
| $\mu$ | Viscosity of the flowing fluid | 0.001 | [Pa/s] |
| $Q_f$ | Heat transfer from the fins | N/A | [W] |
| $Q_b$ | Heat transfer from the unfinned area below the plate | N/A | [W] |
| $Q_t$ | Heat transfer from the flowing fluid above the plate | N/A | [W] |
| $T_\infty$ | Ambient fluid temperature below the plate | 10 | [°C] |
| $h_c$ | Convective heat transfer coefficient below the plate | 10 | [W/m$^2$/K] |
| $\delta$ | Velocity boundary layer | N/A | [m] |
| $\delta_T$ | Thermal boundary layer | N/A | [m] |
| $T_\infty$ | Ambient fluid temperature below the plate | 10 | [°C] |
| $h_c$ | Convective heat transfer coefficient below the plate | 10 | [W/m$^2$/K] |
| $k_f$ | Thermal conductivity of the fins | 200 | [W/m/K] |
| $P/A$ | Ratio of perimeter to cross-sectional area of the fins | 100 | [m$^{-1}$] |
| $A_f$ | Total cross-sectional area of the fins | ½ A | [m$^2$] |
| $A$ | Total area of the plate | N/A | [m$^2$] |

# 1. Model Formulation

## Schematic Diagram

A schematic of the model being considered is presented below in Figure 1.

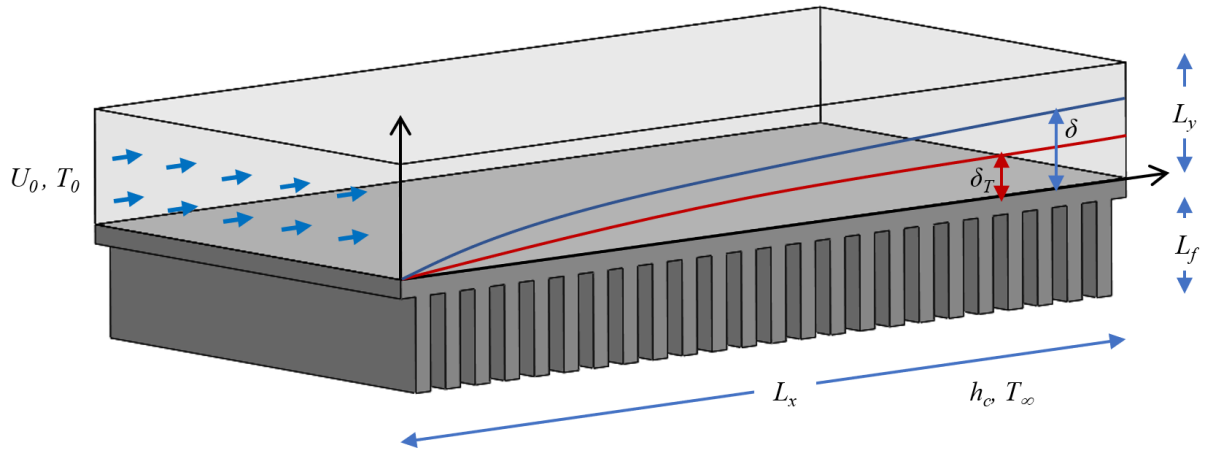


Figure 1. Schematic of a Horizontal Finned-Flat Plate with Fluid Flowing Over

## Equations

The total heat transfer through a fin with negligible heat loss at the tip is given by Equation 1.

$$Q_f = A_f k_f m \tanh(mL_f)(T - T_\infty), \qquad \text{where } m = \sqrt{\frac{P}{A}\frac{h_c}{k_f}} \tag{1}$$

This equation is useful for fins with sufficiently large aspect ratios (length/width) but is not accurate for more stout fins. This is an important consideration to keep in mind.

The heat transfer for the area below the plate where there are no fins is assumed to be equivalent to that of a flat plate with the heat transfer coefficient prescribed (Equation 2).

$$Q_b = h_c A_f (T - T_\infty) \tag{2}$$

It is also known that the fins take up precisely half the area of the plate, hence the area of the area without fins is also equal to $A_f$.

The heat transfer from the topside fluid is equal to Equation 3. Note, in this equation $A$ is equivalent to $2A_f$ with the previous logic.

$$Q_t = hA(T_o - T) \tag{3.a}$$

$$Q_t = 2hA_f(T_{1j} - T_{bottom}) \tag{3.b}$$

Equation 3.a is the heat transfer considering the initial fluid temperature – a more holistic approach, whereas Equation 3.b describes the heat transfer within the context of the finite volume method being used. $T_{1j}$ represents the temperature of the bottommost finite volumes.

Considering an energy balance, Equation 4 may be obtained. This simply equates the heat transfer from the top surface to the bottom area and fins.

$$Q_t = Q_f + Q_b \tag{4}$$

Equations 1, 2, and 3.b are substituted in Equation 4 and $A_f$ is divided out yielding Equation 5. Note, since the area was removed, these expressions now constitute equating the heat fluxes (heat transfer per unit area).

$$\begin{gathered} 2h\left(T_{1j} - T_{bottom}\right) = C_1(T - T_\infty) + h_c(T - T_\infty) \\ \text{where } C_1 = k_f m \tanh\left(mL_f\right) \end{gathered} \tag{5}$$

Solving this for *h*, Equation 6 is obtained.

$$h = \frac{1}{2}(C_1 + h_c)\frac{(T - T_\infty)}{(T_{1j} - T_{abottom})} \tag{6}$$

The expression $\frac{1}{2}(C_1 + h_c)$ is the average heat transfer coefficient along the bottom side of the plate. This is emergent from the algebra but coincides nicely with what is to be expected.

With a more rigorous consideration of the finite volume method, *h* can be represented by Equation 7.

$$h = \left(\frac{T_{2j} - T_{1j}}{T_{1j} - T_{abottom}}\right)\frac{k}{dy} \tag{7}$$

This equation effectively equates the heat transfer coefficient to $k(dT/dy)/\Delta T$. This is, in fact, the exact definition of *h*.

Now, equating Equations 6 and 7 and solving for $T_{1j}$, Equation 8 is obtained. Here, $T_{1j}$ is the temperature of the finite volumes along the bottom boundary

$$T_{1j} = \frac{\frac{1}{2}(C_1 + h_c) + \frac{k}{dy}T_{2,j}}{\frac{1}{2}(C_1 + h_c) + \frac{k}{dy}} \tag{8}$$

Note, this equation does not depend on the heat transfer coefficient at the top (*h*) or a prescribed temperature at the bottom ($T_{abottom}$). This is ideal since both values are unknown. At this point, Equation 8 is precisely the boundary condition needed to use the finite difference method.

## Rigorous Solution

As previously mentioned, Equation 1 works well for fins of sufficiently large aspect ratios. This means the numerical model will not be applicable for small fin lengths. For a more generalized equation, Equation 1 will need to be replaced with an expression for $Q_f$ that provides reasonable results as $L_f$ approaches 0 m. From Özişik, this general fin equation is defined as [1]:

$$Q_f = A_f k_f m \left( \frac{\sinh(mL_f) + \left(\frac{h_e}{mk_f}\right)\cosh(mL_f)}{\cosh(mL_f) + \left(\frac{h_e}{mk_f}\right)\sinh(mL_f)} \right)(T - T_\infty), \quad \text{where } m = \sqrt{\frac{P}{A}\frac{h_c}{k_f}} \tag{9}$$

When $L_f$ = 0, $Q_f$ reduces to $hA_f(T - T_\infty)$, precisely the expression for heat transfer along a flat plate.

To implement this equation into the previous expressions, $C_1$ can be specified to be

$$C_{1^*} = k_f m \left( \frac{\sinh(mL_f) + \left(\frac{h_e}{mk_f}\right)\cosh(mL_f)}{\cosh(mL_f) + \left(\frac{h_e}{mk_f}\right)\sinh(mL_f)} \right) \tag{10}$$

in Equation 8. This is the only adjustment needed to implement this new fin equation.

## 2. Model Solution

Through the modification of Professor Juan Pablo Trelles' code "heatconvection2d_fvm_gs.m", a numerical solution using the finite volume method can be obtained for this analysis. Implementing Equation 8 in MATLAB, the bottom boundary temperature was defined to be:

```
T( i, j ) = ( T( i+1, j )*(k/dy) + Tinf*0.5*(kmt + hc)) / ...
 ( 0.5*(kmt + hc)+ k/dy);
```

This is specified within the boundary nodes section of the code, within the iterative solver. Of note, this does not depend on the successive temperature iterations – it provides the same boundary temperature at each step.

Because this line does not make use of a specified bottom temperature or heat transfer coefficient, they do not need to be included as boundary conditions. All requisite boundary conditions are as follows:

```
% Boundary conditions:
Taleft   =   T0;    % [C]       ambient temperature - left
Taright  =   0;     % [C]       ambient temperature - right
Tatop    =   0;     % [C]       ambient temperature - top

hleft    = 1e20;    % [W/m^2/K] convective coefficient - left
hright   = 0.0;     % [W/m^2/K] convective coefficient - right
htop     = 0.0;     % [W/m^2/K] convective coefficient - top
```

The left convective heat transfer coefficient is defined as an absurdly large number since this results in a fixed left boundary temperature.

The velocity distribution throughout the domain was defined is a nested function, shown below.

```
% Boundary layer function
function ux = UXFUNCTION(x,y)
mu =  0.001;
rho = 800;
Uxmax = 0.001;

del = @(x) ((280/13)*(mu*x/(rho*Uxmax)))^.5;
if y >= del(x)
    ux = Uxmax;
else
    ux = Uxmax*((y/del(x)) - 0.5*(y/del(x))^3);
end
end
```

This function takes the input positions *x* and *y* and considers them along the velocity boundary $\delta(x)$ (or `del` in this code). If the *y* position is greater than or equal to $\delta(x)$, then the velocity is set to be the maximum. In this case, it is 0.001 m/s. If the y position is less than $\delta(x)$, the velocity is then set to be equal to the following function.

$$u_x(x,y) = U_0\left(\frac{3}{2}\left(\frac{y}{\delta(x)}\right) - \frac{1}{2}\left(\frac{y}{\delta(x)}\right)^3\right) \qquad (11)$$

$$\text{where } \delta(x) = \left(\frac{280}{13}\frac{\mu x}{\rho U_0}\right)^{1/2}$$

The rest of this code is provided in Appendix A.1.

## 3. Model Verification

The verification code is provided in Appendix A.2. This is effectively the same as the finned heat transfer code in Appendix A.1 but substitutes the variable $C_1$ (or `kmt` in the code) for a very large number. This effectively makes the fins perfect heat conductors with a negligible temperature gradient resulting in the bottom temperature boundary being equal to 10 °C.

Considering the temperature distributions at $x = L$ for nodes $N_x$ = 100, 200, and 400 and $N_y$ = 20, 40 and 80, the following plots in Figure 2 were obtained.

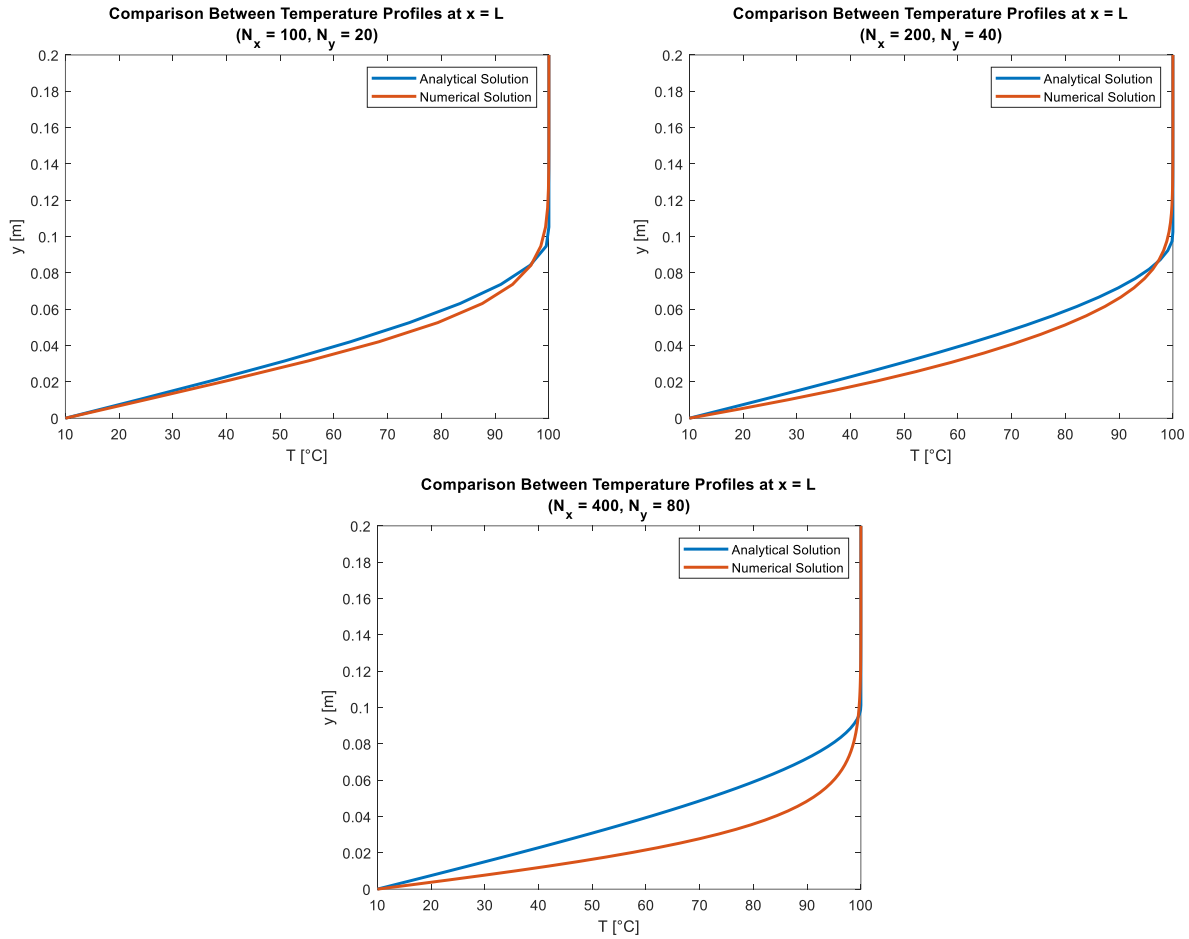


Figure 2. Temperature Profiles at $x = L$ for Various ($N_x$, $N_y$) Node Combinations (Top Left: (100, 20), Top Right: (200, 40), Bottom: (400, 80))

From these plots, there appears to be close agreement between the analytical and numerical solutions at $x = L$. This is especially so for lesser node counts. The primary difference between the analytical and numerical solutions is the shape of the curve. The analytical solution has a much more pronounced bend whereas the numerical solution presents a more gradual curve. This may indicate that the numerical solution does not completely capture the heat transfer phenomena in its entirety. This discrepancy appears to be amplified at higher node counts (the third plot in Figure 2).

The accuracy of the top two plots in Figure 2 indicates reasonable results may be obtained at moderate node amounts. So, for $N_x$ = 200 and $N_y$ = 40, the following temperature distributions across the 2D domain may be obtained for both the numerical and analytical solutions.

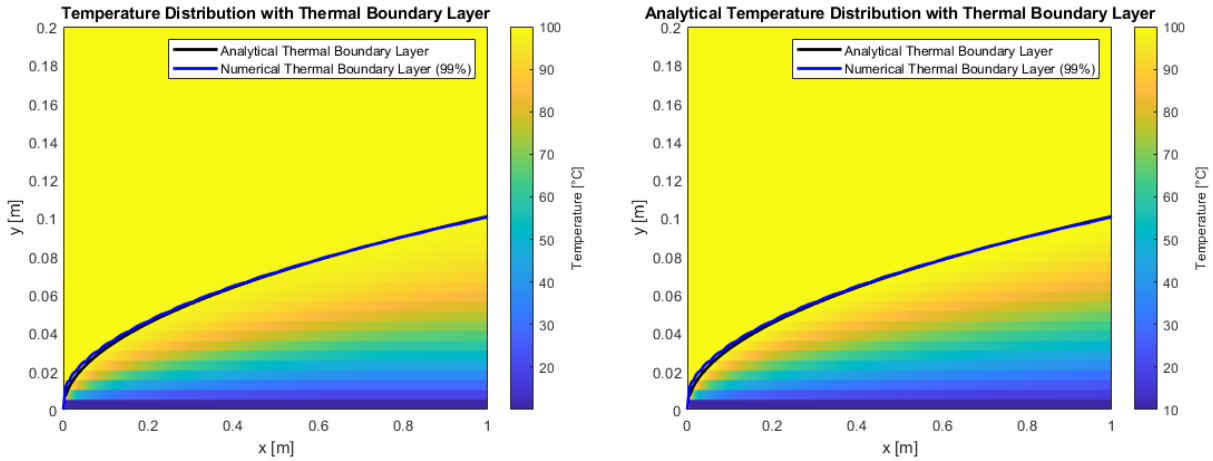


Figure 3. Temperature Distributions for Numerical and Analytical Methods at a Base Plate Temperature of 10 °C

In Figure 3, the two plots are nearly identical. There is no discernable difference between the gradients and the thermal boundary layers overlap similarly in both plots. The black line is the analytical thermal boundary layer given by $\delta_t = \frac{4.53\,x}{(\sqrt{Re_x})(\sqrt[3]{Pr})}$ whereas the blue line is the boundary from the numerical solution where the temperature is 99% of the maximum temperature for each x position.

# 4. Optimal Design

## Optimization Algorithm

The code used to determine the optimal design is provided in Appendix A.3.

Upon initial investigations into the behavior of the finned plate as a function of $L_x$ and $L_y$, it became evident that there was a variety of potential combinations that would result in a mean fluid temperature decrease of 10 °C at $x = L$. This temperature decrease will be abbreviated as $\Delta T$. This is inherently problematic because most 2D optimization algorithms are meant to find a discrete point in the domain – not a distribution of points.

To combat this, a novel 2D optimization method was devised. This algorithm creates a coarse node array, of size $N_f$ by $N_x$, consisting of $L_f$ and $L_x$ dimensions in which the $\Delta T$ value is calculated for each node. The algorithm then interpolates along each $L_f$ column to determine approximately where $\Delta T$ = 10 °C. This position in the $L_x$ dimension is rounded to the closest node ($N_c$), and two more nodes are created above and below $N_c$. The node above $N_c$ is ½ ($N_c$ + $N_{c+1}$) whereas the node below is ½ ($N_c$ + $N_{c-1}$). This algorithm is iterated, creating increasingly refined meshes. A power function is then fit to this refined data and an expression relating $L_f$ to $L_x$ is obtained for resulting $\Delta T$ = 10 °C.

This algorithm is limited insofar as the number of nodes in the $L_f$ dimension are fixed. However, this was necessary for the interpolation method used.

## Fin Equation with Negligible Tip Heat Transfer

The resulting plot and function for this using the equation for a fin with negligible tip heat transfer is presented below in Figure 4.

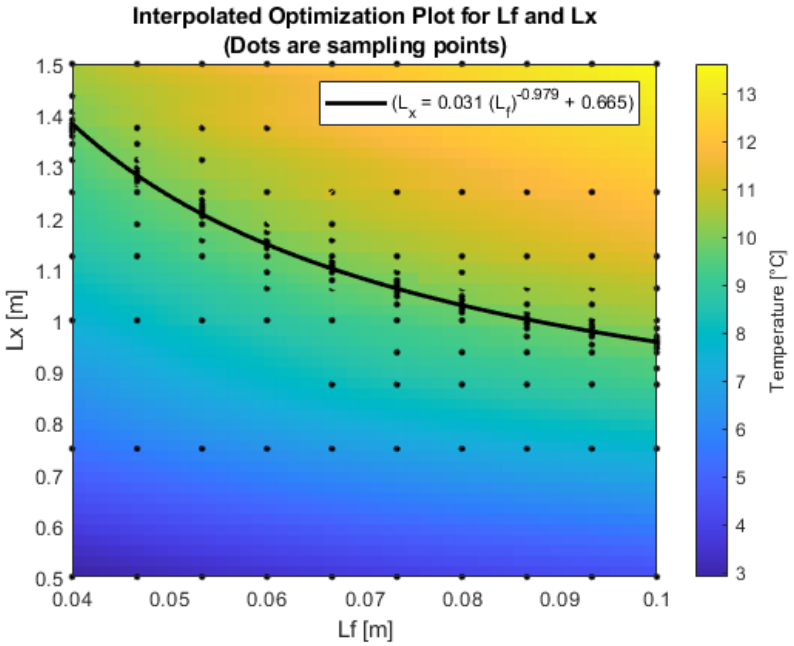


Figure 4. Optimization Plot and Function for $\Delta T$ = 10 °C in the $L_f$, $L_f$ Domain (Fin with Negligible Tip Heat Transfer, 10 Iterations)

From this plot, the algorithm works precisely as intended – with an increasingly refined mesh around the line where $\Delta T$ = 10 °C. Using this function, the following $\Delta T$ values were obtained for a select few $L_f$ values. As an aside, this graph began at $L_f$ = 0.04 m because this optimization algorithm operates poorly around $L_f$ = 0.01 m.

Table 1. Resulting Temperatures for Select $L_f$ Lengths Using the Equation from Figure 4

| $L_f$ | [m] | 0.05 | 0.06 | 0.07 | 0.08 | 0.09 | 0.1 |
|---|---|---|---|---|---|---|---|
| $L_x$ | [m] | 1.242 | 1.148 | 1.080 | 1.030 | 0.990 | 0.958 |
| $\Delta T$ | [°C] | 10.006 | 9.988 | 10.066 | 10.083 | 10.056 | 9.994 |
| **% Error** | [ ] | 0.06% | -0.12% | 0.66% | 0.83% | 0.56% | -0.06% |

The percent errors, which are all less than 1%, indicate that the equation fit to the data in Figure 4 accurately maps the behavior between $L_f$, $L_x$, and $\Delta T$. These errors average 0.32% with a 0.37% standard deviation. Certainly, any one of these six combinations of $L_f$ and $L_x$ could be used and a very close $\Delta T$ would be obtained.

## Generalized Fin Equation

This optimization can also be run considering the more rigorous fin heat transfer expression from Equation 10. This is the general fin expression that considers fins even with small aspect ratios. The plot resulting is provided below in Figure 5.

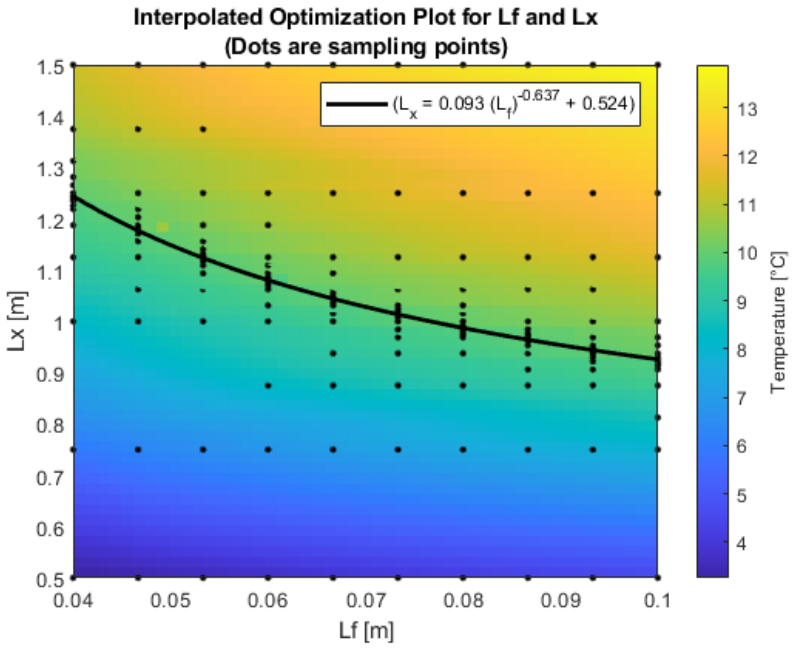


Figure 5. Optimization Plot and Function for $\Delta T$ = 10 °C in the $L_f$, $L_f$ Domain (General Fin Equation, 5 Iterations)

This plot is very similar to the plot in Figure 4. However, the function fit to the data begins around $L_x$ = 1.25 m whereas the fit function in Figure 4 begins around 1.4 m. Both functions end around 0.92 m at $L_x$ = 0.1 m.

Though initially concerning, this result is rather unremarkable. It is known that the heat transfer equation for negligible heat loss at the tip becomes increasingly inaccurate at smaller aspect ratios. This behavior, therefore, accounts for the discrepancy between the two functions at smaller fin lengths. Furthermore, as expected, the difference between the general fin equation and the fin equation with negligible tip heat transfer decreases as the aspect ratio increases.

Now considering the function from Figure 5 and the general fin equation, the following table was constructed for various fin lengths.

Table 2. Resulting Temperatures for Select $L_f$ Lengths Using the Equation from Figure 5

| $\boldsymbol{L_f}$ | [m] | 0.050 | 0.060 | 0.070 | 0.080 | 0.090 | 0.100 |
|---|---|---|---|---|---|---|---|
| $\boldsymbol{L_x}$ | [m] | 1.149 | 1.080 | 1.028 | 0.987 | 0.954 | 0.926 |
| $\boldsymbol{\Delta T}$ | [°C] | 9.990 | 10.066 | 10.078 | 10.044 | 9.973 | 10.082 |
| **% Error** | [ ] | -0.10% | 0.66% | 0.78% | 0.44% | -0.27% | 0.82% |

Ironically, with the more rigorous method, there are greater magnitude percent errors for each selected fin length. These errors average -0.39% with a 0.42% standard deviation. This is very similar to the mean and variability as the standard fin equation in Table 1. So, if the generalized fin equation were to be the accepted formula, the $L_f$ and $L_x$ combinations presented in Table 2 would work excellently.

## References

[1] M. N. Özişik, "One-Dimensional, Steady-State Heat Conduction," in *Heat Transfer*, MEDTECH, 2018, pp. 42-100.

[2] J. P. Trelles, "Project 4: Design of a finned cooling plate," University of Massachusetts; Lowell, Lowell, MA, 2020.

# Appendix A

## A.1: Finned Heat Transfer Script

```
% Heat Transfer of a Finned Plate
%
% MECH.5410 - Advanced Heat Transfer
% Project 4
% Nicholas McLaughlin
% 11/30/2020

% NOTE ------------------------------------------------------------------
% This code is a modification of Professor Jaun Pablo Trelles' 2020 code
% "heatconvection2d_fvm_gs.m". Slight modifications have been made to suit
% this analysis but the core functionality remains Professor Trelles' work.
% No credit is meant to be taken for his work.

% PURPOSE ---------------------------------------------------------------
% This code considers a flat, horizontal plate with uniform fins on the
% underside. The top surface is heated by a transverse flowing fluid
% whereas the bottom surface is cooled by the ambient environment. The
% temperature distribution of the fluid is of primary interest for this
% investigation.

% CODE DESCRIPTION ------------------------------------------------------
% The following code makes use of an iterative Gauss-Seidel method to
% evaluate the temperature distribution of the fluid using a finite volume
% method.

% INPUTS ----------------------------------------------------------------
% Lx: Length of plate
% Lf: Length of fins
% convcond: specify which fin equation to use

% OUTPUTS ---------------------------------------------------------------
% Total number of iterations
% Perclet number
% Overall mean temperature
% Overall maximum temperature
% Plot of the temperature distribution
% Plot of the temperature profile at particular instants along the x axis
% Plot of the temperature distribution with the analytical and numerically
%       calculated thermal boundary layer

% FIN EQUATIONS
% This code makes use of two different equations that can be specified by
% the user
% 1. Fin with Negligible Tip Heat Transfer
%       Qf = A*kf*m*tanh(m*Lf)*(T-Tinf)
%       where m = sqrt(P*hc/A*kf)
% 2. General Fin Equation
%       Qf = kf*m*((sinh(mL) + (he/(m*kf))*cosh(mL))/(cosh(mL) + ...
%           (he/(m*kf))*sinh(mL)))*(T-Tinf);
%       where m = sqrt(P*hc/A*kf)
%
% It is important to note that equation 2 reduces to equation 1 at
```

```
% sufficiently long fin lenghts

% BOUNDARY CONDITIONS ----------------------------------------------------
% The following are the only boundary conditions that need to be specified
% in order for this code to properly run.
% Left temperature:
%   T(i,j) = 100;
% Bottom temperature:
%   T(i,j) = (0.5(kmt + hc)Tinf + T(i+1,j)*k/dy)/(0.5*(kmt + hc) + k/dy)
%       where kmt = kf*m*tanh(m*Lf)
%       where m = sqrt(P*hc/A*kf)
% Left convective heat transfer coefficient
%   h = 1e20
%       Note: this is a large magnitude in order for the temperature at the
%       left boundary to be fixed

% ORIGINAL CODE DESCRIPTION ----------------------------------------------
% heatconvection2d_fvm_gs.m:
%
% Finite Volume Method (FVM) solution of the heat convection
% in a two-dimensional (2D) Cartesian domain.
%
% Numerical solution attained using the Gauss-Seidel (GS) iterative
% procedure instead of a direct solution of the linear system A * T = b.
%
% The temperature distribution T(x,y) due to heat convection through a
% rectangular domain of size Lx x Ly, with thermal conductivity k,
% volumetric heat capacity rhoCp, and volumetric heat generation g is
% described by:
%
%    rhoCp * Ux * dT/dx + rhoCp * Uy * dT/dy ...
%  - k * dT^2/dx^2 - k * dT^2/dy^2 + g = 0,    0 < x < Lx, 0 < y < Ly
%
% The boundary conditions for the problem are:
%
%    @ x = 0  (left  ):  k * dT/dx = hleft   * ( T - Taleft   )
%    @ x = Lx (right ): -k * dT/dx = hright  * ( T - Tariright  )
%    @ y = 0  (bottom):  k * dT/dy = hbottom * ( T - Tabottom )
%    @ y = Ly (top   ): -k * dT/dy = htop    * ( T - Tatop    )
%
% where hleft and hright are convective heat transfer coefficients, and
% Taleft and Tariright reference temperatures, respectively.
%
% Finite Volume Method (FVM) approximation: ==============================
%
% Physical domain:
%        ^
%        |
% (0, Ly)|           (top)           (Lx, Ly)
%        o---------------------------o
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%  (left)|                           |(right)
```

```
%         |                            |
%         |                            |
%         |                            |
%         |                            |
%         |                            |
%         o----------------------------o ----> x
%  (0, 0)           (bottom)         (Lx, 0)
%
% Index domain:
% Finite Volume discrettization - capital letters: nodes, lowercase: faces
%
%  (1, 1)            (bottom)         (1, N)
%         o----------------------------o -----> j (equivalent to x axis)
%         |                            |
%         |                            |
%         |           (South, S)       |
%         |            i-1,j           |
%         |              |             |
%  (left) |    i,j-1 - i,j - i,j+1     |(right)
%         | (West, W)    |   (East, E) |
%         |            i+1,j           |
%         |           (North, N)       |
%         |                            |
%         |                            |
%         o----------------------------o
%   (N,1) |             (top)          (N, N)
%         |
%         i (equivalent to y axis)
%
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
% Initialize code
%-------------------------------------------------------------------------

clear
close all
convcond = menu('Select Fin Equation','Fin with negligible tip convection'...
    ,'Generalized fin equation');

%-------------------------------------------------------------------------
% Inputs
%-------------------------------------------------------------------------

% material properties:
rhoCp    = 800*2000;    % [J/m^3/K] volumetric heat capacity (1e3)
k        = 0.5;    % [W/m/K]   thermal conductivity (1.0)
g        = 0.0;    % [W/m^3]   volumetric heat generation (0.0)
mu =  0.001;
rho = 800;
Cp = 2000;

% geometry:
Lx       = 1.0;    % [m]       length of domain along x (1.0)
Ly       = 0.2;    % [m]       length of domain along y (0.1, 0.2)
```

```
Lf       = 0.01;

% Dimensionless Parametrs
Rex = @(x,v) rho*x*v/mu;
Pr = Cp*mu/k;

% Additional conditions
T0 = 100;          % [C]
Tinf = 10;         % [C]
kf = 200;
hc = 10;
PdivA = 100;

% Velocity:
Uxmax = 0.001;       % [m/s]    maximum velocity
Uxfun = @(x,y) UXFUNCTION(x,y);
Uyfun = @( x, y )(0);% [m/s]

% Boundary layer functions
del = @(x) ((280/13)*mu*x/(rho*Uxmax))^.5;
delT = @(x) 4.53*x/(((Rex(x, Uxmax))^.5)*(Pr^(1/3)));

% Additional parameters
he = hc;
m = sqrt(PdivA*(hc/kf));
mL = m*Lf;
D1 = kf*m*((sinh(mL) + (he/(m*kf))*cosh(mL))/(cosh(mL) + ...
    (he/(m*kf))*sinh(mL)));

% Specify which fin equation to use
if convcond == 1
    kmt = kf*m*tanh(m*Lf);
elseif convcond == 2
    kmt = D1;
else
    disp('Error')
end

% Boundary conditions:
Taleft   =   T0;   % [C]       ambient temperature - left
Taright  =   0;    % [C]       ambient temperature - right
Tatop    =   0;    % [C]       ambient temperature - top

hleft    = 1e20;   % [W/m^2/K] convective coefficient - left
hright   = 0.0;    % [W/m^2/K] convective coefficient - right
htop     = 0.0;    % [W/m^2/K] convective coefficient - top


% Solution parameters:
Nx       = 200;    % number of nodes along x (200)
Ny       = 40;     % number of nodes along y (20, 40)
tol      = 1.0e-8; % tolerance for solution (1.0e-5)

% Upwind function:
```

```
funUpwind = @( P )( max( 0.0, ( 1.0 - 0.1 * abs( P ) )^5 ) ); % power-law

%--------------------------------------------------------------------------
% Set up iterative solution
%--------------------------------------------------------------------------

dx = Lx / ( Nx - 1 );      % discrete x differential
x  = 0 : dx : Lx;          % discrete x axis, x(1) = 0, x(Nx) = Lx

dy = Ly / ( Ny - 1 );      % discrete y differential
y  = 0 : dy : Ly;          % discrete y axis, y(1) = 0, y(Ny) = Ly

T = zeros( Ny, Nx );       % discrete solution, initial guess

% Create an arbitrary initial temperature distribution guess
for i = 1:Ny
    for j = 1:Nx
        T(i,j)    = 100 + 1*sin(i*pi/10) + 1*cos(j*pi/10);
    end
end

% Define parameters to initialize the iterations
Told    = T + 0.5;         % auxiliary array to store previous solution
error   = 100 * tol;       % initialize error (any large number > tol)
nite    = 0;               % iteration counter
nitemax = 1000;            % maximum number of iterations

%--------------------------------------------------------------------------
% Iterative solver
%--------------------------------------------------------------------------

tic
while ( error > tol ) && ( nite < nitemax )

    % Update iteration counter:
    nite = nite + 1;

    % Update temperature from the old values:
    for i = 1 : Ny
        for j = 1 : Nx

            % Boundary nodes:
            if     j ==  1 % Left
                T( i, j ) = ( T( i, j+1 ) + ( hleft   * dx / k ) * Taleft   )
/ ...
                    ( 1           + ( hleft   * dx / k ) );

            elseif j == Nx % Right
                T( i, j ) = ( T( i, j-1 ) + ( hright  * dx / k ) * Taright  )
/ ...
                    ( 1           + ( hright  * dx / k ) );

            elseif i ==  1 % Bottom
                T( i, j ) = ( T( i+1, j )*(k/dy) + Tinf*0.5*(kmt + hc)) / ...
```

```
            ( 0.5*(kmt + hc)+ k/dy);

        elseif i == Ny % Top
            T( i, j ) = ( T( i-1, j ) + ( htop    * dy / k ) * Tatop    )
/ ...
            ( 1           + ( htop    * dy / k ) );

        else % Inside nodes:

            % Coordinates around control volume:
            xe = x( j ) - dx / 2; ye = y( i )         ;
            xw = x( j ) + dx / 2; yw = y( i )         ;
            xn = x( j )         ; yn = y( i ) + dy / 2;
            xs = x( j )         ; ys = y( i ) - dy / 2;

            % Velocities normal the control volume faces:
            Uxe = Uxfun( xe, ye );
            Uxw = Uxfun( xw, yw );
            Uyn = Uyfun( xn, yn );
            Uys = Uyfun( xs, ys );

            % Discrete advective fluxes:
            Fe = rhoCp * Uxe * dy;
            Fw = rhoCp * Uxw * dy;
            Fn = rhoCp * Uyn * dx;
            Fs = rhoCp * Uys * dx;

            % Discrete diffusive fluxes:
            De = k * dy / dx;
            Dw = k * dy / dx;
            Dn = k * dx / dy;
            Ds = k * dx / dy;

            % Local Peclet number on each face:
            Pe = Fe / De;
            Pw = Fw / Dw;
            Pn = Fn / Dn;
            Ps = Fs / Ds;

            % Coefficients:
            aE = De * funUpwind( Pe ) + max( -Fe, 0.0 );
            aW = Dw * funUpwind( Pw ) + max(  Fw, 0.0 );
            aN = Dn * funUpwind( Pn ) + max( -Fn, 0.0 );
            aS = Ds * funUpwind( Ps ) + max(  Fs, 0.0 );
            aP = aE + aW + aN + aS;
            b  = g * dx * dy;

            % Gauss-Seidel update: T( y, x )
            T( i, j ) = ( aW * T( i    , j - 1 ) + ...
                aE * T( i    , j + 1 ) + ...
                aS * T( i - 1, j     ) + ...
                aN * T( i + 1, j     ) + b ) / aP;

        end
```

```
        end
    end

    % Update error:
    error = max( T(:) - Told(:) );

    % Store the solution from this iteration:
    Told = T;
end
toc % Stop timer

%--------------------------------------------------------------------------
% Post Processing
%--------------------------------------------------------------------------

% Screen output:
fprintf(' total number of iterations: %i \n', nite );

% Report characteristics of problem and solution:
fprintf(' Peclet number: %2.2f \n'      , rhoCp * Uxmax * Ly / k );
fprintf(' mean temperature: %2.2f \n'   , mean( T( : ) ) );
fprintf(' maximum temperature: %2.2f \n', max(  T( : ) ) );

%--------------------------------------------------------------------------
% Plotting
%--------------------------------------------------------------------------

% Surface plot to scale
[ X, Y ] = meshgrid( x, y );
figure
surfc( X, Y, T, 'edgecolor', 'none' )
view(2)
c = colorbar;
c.Label.String = 'Temperature [°C]';
xlabel(' x [m] ')
ylabel(' y [m] ')
title(' Temperature Distribution')
hold on
axis tight
axis equal

% Overlay velocity and thermal boundary layers
dell = zeros(1,Nx);
delTT = dell;
for j = 1:Nx
    dell(j) = (4.96*x(j))/(Rex(x(j),Uxmax))^.5;
    delTT(j) = 4.53*x(j)/(((Rex(x(j),Uxmax))^.5)*(Pr^(1/3)));
end

% Line plot:
Nx1 = 1;                  % node for x ~   0% Lx
Nx2 = round( 0.2 * Nx );  % node for x ~  20% Lx
Nx3 = round( 0.5 * Nx );  % node for x ~  50% Lx
Nx4 = Nx;                 % node for x ~ 100% Lx
```

```
figure
plot( T( :, Nx1 ), y, '-r', ...
    T( :, Nx2 ), y, '-m', ...
    T( :, Nx3 ), y, '-g', ...
    T( :, Nx4 ), y, '-b'  )
legend( 'Inlet', '20%%L_x', '50%%L_x', 'L_x', ...
    'location', 'northwest' );
box on
xlabel(' T [C] ')
ylabel(' y [m] ')
title(' Temperature Profile T(y) ')

% Calculate thermal boundary layer
eps = zeros(Ny,1);
Bnd = zeros(Ny,1);
for j = 2:Nx
    for i = 1:Ny
        eps(i) = 1e-9*i;
    end
    TCol = T(:,j) + eps;
    TBnd = 0.990*(max(T(:,j)) - min(T(:,j))) + min(T(:,j));
    Bnd(j) = interp1(TCol,y',TBnd);
end

% Surface plot with thermal boundary overlay
figure
surfc( X, Y, T, 'edgecolor', 'none' )
view(2)
axdim = axis;
c = colorbar;
c.Label.String = 'Temperature [°C]';
xlabel(' x [m] ')
ylabel(' y [m] ')
title('Temperature Distribution with Overlaid Thermal Boundary Layer')
hold on

ax = axes;
plot(x,delTT,'black',x,Bnd,'Blue','LineWidth',2)
axis([0 Lx 0 Ly])
legend('Analytical Thermal Boundary Layer','Numerical Thermal Boundary Layer
(99%)')
ax.Color = 'none';
ax.Position(3) = 0.68;
ax.Position(1) = 0.114;
ax.XTick = [];
ax.YTick = [];

%-------------------------------------------------------------------------
% Nested functions
%-------------------------------------------------------------------------

% Boundary layer function
function ux = UXFUNCTION(x,y)
mu =  0.001;
```

```
rho = 800;
Uxmax = 0.001;

del = @(x) ((280/13)*(mu*x/(rho*Uxmax)))^.5;
if y >= del(x)
    ux = Uxmax;
else
    ux = Uxmax*((y/del(x)) - 0.5*(y/del(x))^3);
end
end
```

## A.2: Verification of Finned Heat Transfer Script

```
% Verification: Heat Transfer of a Finned Plate
%
% MECH.5410 - Advanced Heat Transfer
% Project 4
% Nicholas McLaughlin
% 11/30/2020

% NOTE ------------------------------------------------------------------
% This code is a modification of Professor Jaun Pablo Trelles' 2020 code
% "heatconvection2d_fvm_gs.m". Slight modifications have been made to suit
% this analysis but the core functionality remains Professor Trelles' work.
% No credit is meant to be taken for his work.

% PURPOSE ---------------------------------------------------------------
% This code contains a very slight modification to the Heat Transfer of a
% Finned Plate previously established. See that code for all relevant
% inputs and considerations.

% CODE DESCRIPTION ------------------------------------------------------
% The following code makes use of an iterative Gauss-Seidel method to
% evaluate the temperature distribution of the fluid using a finite volume
% method.

% OUTPUTS ---------------------------------------------------------------
% Total number of iterations
% Perclet number
% Overall mean temperature
% Overall maximum temperature
% Plot of the temperature distribution
% Plot of the temperature profile at particular instants along the x axis
% Plot of the temperature distribution with the analytical and numerically
%       calculated thermal boundary layer
% Plot of the analytical temperature distribution with the analytical and
%       numerically calculated thermal boundary layer

% BOUNDARY CONDITIONS ---------------------------------------------------
% The following are the only boundary conditions that need to be specified
% in order for this code to properly run.
% Left temperature:
%   T(i,j) = 100;
% Bottom temperature:
%   T(i,j) = (0.5(kmt + hc)Tinf + T(i+1,j)*k/dy)/(0.5*(kmt + hc) + k/dy)
%       where kmt = 1e12
%       Note, this is a very high magnitude so the top of the plate has
%           the same temperature as Tinf. This is all that needed to be
%           changed in order to run this code at Tbottom = 10 C.
% Left convective heat transfer coefficient
%   h = 1e20
%       Note: this is a large magnitude in order for the temperature at the
%           left boundary to be fixed

% ORIGINAL CODE DESCRIPTION ---------------------------------------------
% heatconvection2d_fvm_gs.m:
%
```

```
% Finite Volume Method (FVM) solution of the heat convection
% in a two-dimensional (2D) Cartesian domain.
%
% Numerical solution attained using the Gauss-Seidel (GS) iterative
% procedure instead of a direct solution of the linear system A * T = b.
%
% The temperature distribution T(x,y) due to heat convection through a
% rectangular domain of size Lx x Ly, with thermal conductivity k,
% volumetric heat capacity rhoCp, and volumetric heat generation g is
% described by:
%
%    rhoCp * Ux * dT/dx + rhoCp * Uy * dT/dy ...
%  - k * dT^2/dx^2 - k * dT^2/dy^2 + g = 0,    0 < x < Lx, 0 < y < Ly
%
% The boundary conditions for the problem are:
%
%    @ x = 0  (left  ):  k * dT/dx = hleft   * ( T - Taleft   )
%    @ x = Lx (right ): -k * dT/dx = hright  * ( T - Taright  )
%    @ y = 0  (bottom):  k * dT/dy = hbottom * ( T - Tabottom )
%    @ y = Ly (top   ): -k * dT/dy = htop    * ( T - Tatop    )
%
% where hleft and hright are convective heat transfer coefficients, and
% Taleft and Taright reference temperatures, respectively.
%
% Finite Volume Method (FVM) approximation: ==============================
%
% Physical domain:
%        ^
%        |
% (0, Ly)|           (top)           (Lx, Ly)
%        o---------------------------o
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%  (left)|                           |(right)
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%        o---------------------------o ----> x
%  (0, 0)           (bottom)         (Lx, 0)
%
% Index domain:
% Finite Volume discrettization - capital letters: nodes, lowercase: faces
%
%  (1, 1)            (bottom)         (1, N)
%        o---------------------------o -----> j (equivalent to x axis)
%        |                           |
%        |                           |
%        |           (South, S)      |
%        |            i-1,j          |
%        |              |            |
%  (left)|    i,j-1 - i,j - i,j+1    |(right)
```

```
%         | (West, W)    |    (East, E) |
%         |            i+1,j            |
%         |           (North, N)        |
%         |                             |
%         |                             |
%         o-----------------------------o
%    (N,1)|              (top)          (N, N)
%         |
%         i (equivalent to y axis)
%
%--------------------------------------------------------------------------

%--------------------------------------------------------------------------
% Initialize code
%--------------------------------------------------------------------------

clear
close all

%--------------------------------------------------------------------------
% Inputs
%--------------------------------------------------------------------------

% material properties:
rhoCp    = 800*2000;    % [J/m^3/K] volumetric heat capacity (1e3)
k        = 0.5;     % [W/m/K]   thermal conductivity (1.0)
g        = 0.0;     % [W/m^3]   volumetric heat generation (0.0)
mu =  0.001;
rho = 800;
Cp = 2000;

% geometry:
Lx       = 1.0;     % [m]       length of domain along x (1.0)
Ly       = 0.2;     % [m]       length of domain along y (0.1, 0.2)
Lf       = 0.01;

% Dimensionless Parametrs
Rex = @(x,v) rho*x*v/mu;
Pr = Cp*mu/k;

% Additional conditions
T0 = 100;           % [C]
Tinf = 10;          % [C]
kf = 200;
hc = 10;
PdivA = 100;

% velocity:
Uxmax = 0.001;      % [m/s]     maximum velocity

% Boundary layer functions
del = @(x) ((280/13)*mu*x/(rho*Uxmax))^.5;
delT = @(x) 4.53*x/(((Rex(x, Uxmax))^.5)*(Pr^(1/3)));
```

```
% Velocity:
Uxmax = 0.001;       % [m/s]    maximum velocity
Uxfun = @(x,y) UXFUNCTION(x,y);
Uyfun = @( x, y )(0);% [m/s]

% Additional parameters
he = hc;
m = sqrt(PdivA*(hc/kf));
mL = m*Lf;
D1 = kf*m*((sinh(mL) + (he/(m*kf))*cosh(mL))/(cosh(mL) + ...
    (he/(m*kf))*sinh(mL)));

kmt = 1e12;

% Boundary conditions:
Taleft   =   100;   % [C]       ambient temperature - left (100, 0)
Taright  =   0;     % [C]       ambient temperature - right
%Tabottom = @(T,hbot) (T*(0.5*(kmt + hc) - hbot) - 0.5*(kmt + hc)*Tinf)/hbot;
Tabottom = @(T,hbot) 10;
Tatop    =   0;    % [C]       ambient temperature - top

hleft    = 1e20;    % [W/m^2/K] convective coefficient - left (1e8)
hright   = 0.0;    % [W/m^2/K] convective coefficient - right (0)
htop     = 0.0;    % [W/m^2/K] convective coefficient - top (0)


% Solution parameters:
Nx       = 200;    % number of nodes along x (200)
Ny       = 40;     % number of nodes along y (20, 40)
tol      = 1.0e-8; % tolerance for solution (1.0e-5)

% upwind function:
funUpwind = @( P )( max( 0.0, ( 1.0 - 0.1 * abs( P ) )^5 ) ); % power-law

%--------------------------------------------------------------------------
% Set up iterative solution
%--------------------------------------------------------------------------

dx = Lx / ( Nx - 1 );      % discrete x differential
x  = 0 : dx : Lx;          % discrete x axis, x(1) = 0, x(Nx) = Lx

dy = Ly / ( Ny - 1 );      % discrete y differential
y  = 0 : dy : Ly;          % discrete y axis, y(1) = 0, y(Ny) = Ly

T = zeros( Ny, Nx );       % discrete solution, initial guess

% Create an arbitrary initial temperature distribution guess
for i = 1:Ny
    for j = 1:Nx
        T(i,j)    = 100 + 1*sin(i*pi/10) + 1*cos(j*pi/10);
    end
end

% Define parameters to initialize the iterations
```

```
Told    = T + 0.5;          % auxiliary array to store previous solution
error   = 100 * tol;        % initialize error (any large number > tol)
nite    = 0;                % iteration counter
nitemax = 1000;             % maximum number of iterations

%--------------------------------------------------------------------------
% Iterative solver
%--------------------------------------------------------------------------

tic
while ( error > tol ) && ( nite < nitemax )

    % Update iteration counter:
    nite = nite + 1;

    % Update temperature from the old values:
    for i = 1 : Ny
        for j = 1 : Nx

            % Boundary nodes:
            if     j ==  1 % Left
                T( i, j ) = ( T( i, j+1 ) + ( hleft   * dx / k ) * Taleft   )
/ ...
                    ( 1           + ( hleft   * dx / k ) );

            elseif j == Nx % Right
                T( i, j ) = ( T( i, j-1 ) + ( hright  * dx / k ) * Taright  )
/ ...
                    ( 1           + ( hright  * dx / k ) );

            elseif i ==  1 % Bottom
                T( i, j ) = ( T( i+1, j )*(k/dy) + Tinf*0.5*(kmt + hc)) / ...
                    ( 0.5*(kmt + hc)+ k/dy);

            elseif i == Ny % Top
                T( i, j ) = ( T( i-1, j ) + ( htop    * dy / k ) * Tatop    )
/ ...
                    ( 1           + ( htop    * dy / k ) );

            else % Inside nodes:

                % Coordinates around control volume:
                xe = x( j ) - dx / 2; ye = y( i )         ;
                xw = x( j ) + dx / 2; yw = y( i )         ;
                xn = x( j )         ; yn = y( i ) + dy / 2;
                xs = x( j )         ; ys = y( i ) - dy / 2;

                % Velocities normal the control volume faces:
                Uxe = Uxfun( xe, ye );
                Uxw = Uxfun( xw, yw );
                Uyn = Uyfun( xn, yn );
                Uys = Uyfun( xs, ys );

                % Discrete advective fluxes:
```

```
                Fe = rhoCp * Uxe * dy;
                Fw = rhoCp * Uxw * dy;
                Fn = rhoCp * Uyn * dx;
                Fs = rhoCp * Uys * dx;

                % Discrete diffusive fluxes:
                De = k * dy / dx;
                Dw = k * dy / dx;
                Dn = k * dx / dy;
                Ds = k * dx / dy;

                % Local Peclet number on each face:
                Pe = Fe / De;
                Pw = Fw / Dw;
                Pn = Fn / Dn;
                Ps = Fs / Ds;

                % Coefficients:
                aE = De * funUpwind( Pe ) + max( -Fe, 0.0 );
                aW = Dw * funUpwind( Pw ) + max(  Fw, 0.0 );
                aN = Dn * funUpwind( Pn ) + max( -Fn, 0.0 );
                aS = Ds * funUpwind( Ps ) + max(  Fs, 0.0 );
                aP = aE + aW + aN + aS;
                b  = g * dx * dy;

                % Gauss-Seidel update: T( y, x )
                T( i, j ) = ( aW * T( i    , j - 1 ) + ...
                    aE * T( i    , j + 1 ) + ...
                    aS * T( i - 1, j     ) + ...
                    aN * T( i + 1, j     ) + b ) / aP;

            end
        end
    end

    % Update error:
    error = max( T(:) - Told(:) );

    % Store the solution from this iteration:
    Told = T;
end
toc % Stop timer

%--------------------------------------------------------------------------
% Post processing
%--------------------------------------------------------------------------

% Screen output:
fprintf(' total number of iterations: %i \n', nite );

% Report characteristics of problem and solution:
fprintf(' Peclet number: %2.2f \n'      , rhoCp * Uxmax * Ly / k );
fprintf(' mean temperature: %2.2f \n'   , mean( T( : ) ) );
fprintf(' maximum temperature: %2.2f \n', max(  T( : ) ) );
```

```
%--------------------------------------------------------------------------
% Plotting
%--------------------------------------------------------------------------

% Surface plot to scale
[ X, Y ] = meshgrid( x, y );
figure
surfc( X, Y, T, 'edgecolor', 'none' )
view(2)
c = colorbar;
c.Label.String = 'Temperature [°C]';
xlabel(' x [m] ')
ylabel(' y [m] ')
title(' Temperature Distribution')
hold on
axis tight
axis equal

% Overlay velocity and thermal boundary layers
dell = zeros(1,Nx);
delTT = dell;
for j = 1:Nx
    dell(j) = (4.96*x(j))/(Rex(x(j),Uxmax))^.5;
    delTT(j) = 4.53*x(j)/(((Rex(x(j),Uxmax))^.5)*(Pr^(1/3)));
end

% Line plot:
Nx1 = 1;                  % node for x ~   0% Lx
Nx2 = round( 0.2 * Nx );  % node for x ~  20% Lx
Nx3 = round( 0.5 * Nx );  % node for x ~  50% Lx
Nx4 = Nx;                 % node for x ~ 100% Lx

figure
plot( T( :, Nx1 ), y, '-r', ...
    T( :, Nx2 ), y, '-m', ...
    T( :, Nx3 ), y, '-g', ...
    T( :, Nx4 ), y, '-b'  )
legend( 'Inlet', '20%%L_x', '50%%L_x', 'L_x', ...
    'location', 'northwest' );
box on
xlabel(' T [°C] ')
ylabel(' y [m] ')
title(' Temperature Profile T(y) ')

% Calculate thermal boundary layer
eps = zeros(Ny,1);
Bnd = zeros(Ny,1);
for j = 2:Nx
    for i = 1:Ny
        eps(i) = 1e-9*i;
    end
    TCol = T(:,j) + eps;
    TBnd = 0.990*(max(T(:,j)) - min(T(:,j))) + min(T(:,j));
    Bnd(j) = interp1(TCol,y',TBnd);
```

```
end

% Surface plot with thermal boundary overlay
figure
surfc( X, Y, T, 'edgecolor', 'none' )
view(2)
axdim = axis;
c = colorbar;
c.Label.String = 'Temperature [°C]';
xlabel(' x [m] ')
ylabel(' y [m] ')
title('Temperature Distribution with Thermal Boundary Layer')
hold on

ax = axes;
plot(x,delTT,'black',x,Bnd,'Blue','LineWidth',2)
axis([0 Lx 0 Ly])
legend('Analytical Thermal Boundary Layer','Numerical Thermal Boundary Layer
(99%)')
ax.Color = 'none';
ax.Position(3) = 0.68;
ax.Position(1) = 0.114;
ax.XTick = [];
ax.YTick = [];

%--------------------------------------------------------------------------
% Verification
%--------------------------------------------------------------------------

clear x y
x = linspace(0,Lx,Nx);
y = linspace(0,Ly,Ny);
VdelT = zeros(1,Nx);
TVer = zeros(Nx,Ny);
for i = 1:Nx
    Re = Rex(x(i),Uxmax);
    VdelT(i) = 4.53*x(i)/(Re^(.5) * Pr^(1/3));
    for j = 1:Ny
        if y(j) <= VdelT(i)
            TVer(i,j) = (1.5*(y(j)/VdelT(i)) - 0.5*(y(j)/VdelT(i))^3)*(T0 -
Tinf) + Tinf;
         else
             TVer(i,j) = T0;
        end
    end
end

% Plot surface with overlaid thermal boundary layers
figure
s = surf(x,y,TVer');
s.EdgeColor = 'none';
view(2)
c = colorbar;
c.Label.String = 'Temperature [°C]';
title('Analytical Temperature Distribution with Thermal Boundary Layer')
```

```
xlabel('x [m]')
ylabel('y [m]')

ax = axes;
plot(x,VdelT,'black',x,Bnd,'Blue','LineWidth',2)
axis([0 Lx 0 Ly])
legend('Analytical Thermal Boundary Layer','Numerical Thermal Boundary Layer
(99%)')
ax.Color = 'none';
ax.Position(3) = 0.68;
ax.Position(1) = 0.114;
ax.XTick = [];
ax.YTick = [];

% Plot temperature distributions at x = L
figure
plot(TVer(end,:),y,T(:,end),y,'lineWidth',2);
legend('Analytical Solution','Numerical Solution')
title('Comparison Between Temperature Profiles for Various Solutions at x =
L')
ylabel('y [m]')
xlabel('T [°C]')

%-------------------------------------------------------------------------
% Nested functions
%-------------------------------------------------------------------------

% Boundary layer function
function ux = UXFUNCTION(x,y)
mu =  0.001;
rho = 800;
Uxmax = 0.001;

del = @(x) ((280/13)*(mu*x/(rho*Uxmax)))^.5;
if y >= del(x)
    ux = Uxmax;
else
    ux = Uxmax*((y/del(x)) - 0.5*(y/del(x))^3);
end
end
```

## A.3: Optimization of $L_x$ and $L_f$ Script

```
% Optimization Code for a Finned Plate
%
% MECH.5410 - Advanced Heat Transfer
% Project 4
% Nicholas McLaughlin
% 11/30/2020

%--------------------------------------------------------------------------
% Initialize code
%--------------------------------------------------------------------------

clear
close all

%--------------------------------------------------------------------------
% Define solver parameters
%--------------------------------------------------------------------------

Lxmin = 0.5;     % [m]   Minimum Lx dimension
Lxmax = 1.5;     % [m]   Maximum Lx dimension

Lfmin = 0.04;    % [m]   Minimum Lf dimension
Lfmax = 0.1;     % [m]   Maximum Lf dimension

kfin = 5;        % [ ]   Number of iterations desired
meshdim = 50;    % [ ]   Dimension for final mesh (Size should be
inconsequential)

NLx = 5;         % [ ]   Number of starting Lx nodes
                 %       Note: This will increase by 2 for each iteration
NLf = 10;         % [ ]   Number of Lf nodes (this is fixed throughout the
code)

%--------------------------------------------------------------------------
% Initialize solver
%--------------------------------------------------------------------------

% Create dimension vectors to be tested
Lx = linspace(Lxmin,Lxmax,NLx);
Lf = linspace(Lfmin,Lfmax,NLf);
Lx = repelem(Lx',1,NLf);

%Preallocate matrix
StoreDeltaT = zeros(NLx, NLf);

% Find number of nodes
[NLx,NLf] = size(Lx);

%--------------------------------------------------------------------------
% Iterative solver
%--------------------------------------------------------------------------
```

```
for k = 1:kfin
    tic
    for i = 1:NLx
        for j = 1:NLf
            % This if statement makes it such that the code only calculates
            % the new nodes - there is no recalculation of past nodes
            if StoreDeltaT(i,j) == 0
                [deltaT,Mean,Max,iter] = funConv(Lx(i,j),Lf(j));
                StoreDeltaT(i,j) = deltaT;
            else
            end
        end
    end
    toc

    % Display these arrays to observe the code operating
    disp('DeltaT Array')
    disp(StoreDeltaT)
    disp('Lx Array')
    disp(Lx)

    % Test where the DeltaT is greater than 10 C
    LfExist = zeros(1,NLf);
    for j = 1:NLf
        if max(StoreDeltaT(:,j)) >= 10
            LfExist(j) = 1;
        else
            LfExist(j) = 0;
        end
    end

    % Estimate where DeltaT = 10 C through interpolation
    [~,J] = max(LfExist);
    Lfbnd = zeros(2,(NLf - J));
    for i = J:NLf
        Lfbnd(2,i - (J)+1) = Lf(i);
        Lfbnd(1,i - (J)+1) = interp1(StoreDeltaT(:,i),Lx(:,i),10);
    end
    Bnd = Lfbnd;
    Bnd = sortrows(Bnd');

    % Fit a power function to the data
    FO = fit(Bnd(:,1),Bnd(:,2),'power2');
    Fo = coeffvalues(FO);

    % Create the fit power function and evaluate
    func = @(x) Fo(1) .* x.^Fo(2) + Fo(3);
    x = linspace(Bnd(1,1),Bnd(end,1),NLx*2);
    y = func(x);

    % Determine the inverse of func(x)
    g = @(y) ((y-Fo(3))./Fo(1)).^(1./(Fo(2)));

    % Determine where new nodes need to be created
    [M,N] = size(StoreDeltaT);
```

```
    clear res Lxnew DeltaTnew
    if k < kfin
        for j = 1:NLf
            clear res
            res(:,j) = abs(StoreDeltaT(:,j) - 10);
            [~,I1] = min(res(:,j));
            I(j) = I1;
            if I1 == M
                I1 = M-1;
            elseif I1 == 1
                I1 = 2;
            else
                I1 = I(j);
            end
            a = 0.5*(Lx(I1+1,j) + Lx(I1,j));
            b = 0.5*(Lx(I1,j) + Lx(I1-1,j));
            A = 0;
            B = 0;
            col1 = Lx(1:(I1-1),j);
            col2 = Lx(I1+1:NLx,j);
            Tcol1 = StoreDeltaT(1:(I1-1),j);
            Tcol2 = StoreDeltaT(I1+1:NLx,j);
            disp(I1)

            % Create the new nodes in here
            Lxnew(:,j) = [col1; b; Lx(I1,j); a; col2];
            Lxnew(:,j) = sort(Lxnew(:,j));
            DeltaTnew(:,j) = [Tcol1; B; StoreDeltaT(I1,j); A; Tcol2];
        end
        Lx = Lxnew;
        StoreDeltaT = DeltaTnew;
        [NLx,NLf] = size(Lx);
    else
    end
end

%-------------------------------------------------------------------------
% Plotting
%-------------------------------------------------------------------------

% Plot the DeltaT values for each Lf used
figure
hold on
for i = 1:NLf
    Lx(:,i) = sort(Lx(:,i));
    StoreDeltaT(:,i) = sort(StoreDeltaT(:,i));
    plot(Lx(:,i),StoreDeltaT(:,i),'lineWidth',2)
end
legend(strcat('Lf=',num2str(Lf')),'location','southeast')
title('Delta T for each Input L_f')
xlabel('L_x [m]');
ylabel('Delta T [°C]');

% Create structures for final surface
LF = repelem(Lf,NLx,1);
xlin = linspace(min(LF,[],'all'),max(LF,[],'all'),meshdim);
```

```
ylin = linspace(min(Lx,[],'all'),max(Lx,[],'all'),meshdim);
[X,Y] = meshgrid(xlin,ylin);

% Convert data into a plottable surface
Z = griddata(LF,Lx,StoreDeltaT,X,Y,'cubic');

% Plot surface
figure
s = surf(X,Y,Z);
s.EdgeColor = 'none';
c = colorbar;
c.Label.String = 'Temperature [°C]';
title({'Interpolated Optimization Plot for Lf and Lx','(Dots are sampling
points)'})
xlabel('Lf [m]')
ylabel('Lx [m]')
view(2);
grid off

% Calculate where delta T = 10 C
clear Bnd
Lfbnd = zeros(2,meshdim);
for i = 1:meshdim
    Lfbnd(2,i) = X(1,i);
    Lfbnd(1,i) = interp1(Z(:,i),Y(:,i),10);
end

% Fit and plot a power function
Bnd = Lfbnd';
FO = fit(Bnd(:,2),Bnd(:,1),'power2');
Fo = coeffvalues(FO);
func = @(x) Fo(1) .* x.^Fo(2) + Fo(3); % So this gives us Lx(Lf)
x = X(1,:);
y = func(x);
hold on

% Plot where all the nodes were
plot3(Lf,Lx,StoreDeltaT,'.k','MarkerSize',10)
view(2)
ax = axes;
fplot(func,[min(x),max(x)],'k','lineWidth',2)
axis([min(x) max(x) min(Y,[],'all') max(Y,[],'all')])

% Add additional details
a = Fo(1);
b = Fo(2);
c = Fo(3);
write = sprintf('(L_x = %.3f (L_f)^{%.3f} + %.3f)',a,b,c);
legend(write)
ax.Color = 'none';
ax.Position(3) = 0.676;
ax.Position(1) = 0.113;
ax.Position(4) = 0.79;
ax.XTick = [];
ax.YTick = [];
```

```
% End of code
```

## A.4: Finned Heat Transfer Function

```
% Function: Heat Transfer of a Finned Plate
%
% MECH.5410 - Advanced Heat Transfer
% Project 4
% Nicholas McLaughlin
% 11/30/2020

% NOTE ------------------------------------------------------------------
% This code is a modification of Professor Jaun Pablo Trelles' 2020 code
% "heatconvection2d_fvm_gs.m". Slight modifications have been made to suit
% this analysis but the core functionality remains Professor Trelles' work.
% No credit is meant to be taken for his work.

% PURPOSE ---------------------------------------------------------------
% This code considers a flat, horizontal plate with uniform fins on the
% underside. The top surface is heated by a transverse flowing fluid
% whereas the bottom surface is cooled by the ambient environment. The
% temperature distribution of the fluid is of primary interest for this
% investigation.

% CODE DESCRIPTION ------------------------------------------------------
% The following code makes use of an iterative Gauss-Seidel method to
% evaluate the temperature distribution of the fluid using a finite volume
% method.

% INPUTS ----------------------------------------------------------------
% Lx: Length of plate
% Lf: Length of fins
% convcond: specify which fin equation to use

% OUTPUTS ---------------------------------------------------------------
% Total number of iterations
% Perclet number
% Overall mean temperature
% Overall maximum temperature
% Plot of the temperature distribution
% Plot of the temperature profile at particular instants along the x axis
% Plot of the temperature distribution with the analytical and numerically
%       calculated thermal boundary layer

% FIN EQUATIONS
% THis code makes use of two different equations that can be specified by
% the user
% 1. Fin with Negligible Tip Heat Transfer
%       Qf = A*kf*m*tanh(m*Lf)*(T-Tinf)
%       where m = sqrt(P*hc/A*kf)
% 2. General Fin Equation
%       Qf = kf*m*((sinh(mL) + (he/(m*kf))*cosh(mL))/(cosh(mL) + ...
%            (he/(m*kf))*sinh(mL)))*(T-Tinf);
%       where m = sqrt(P*hc/A*kf)
%
% It is important to note that equation 2 reduces to equation 1 at
% sufficiently long fin lenghts
```

```
% BOUNDARY CONDITIONS ----------------------------------------------------
% The following are the only boundary conditions that need to be specified
% in order for this code to properly run.
% Left temperature:
%   T(i,j) = 100;
% Bottom temperature:
%   T(i,j) = (0.5(kmt + hc)Tinf + T(i+1,j)*k/dy)/(0.5*(kmt + hc) + k/dy)
%       where kmt = kf*m*tanh(m*Lf)
%       where m = sqrt(P*hc/A*kf)
% Left convective heat transfer coefficient
%   h = 1e20
%       Note: this is a large magnitude in order for the temperature at the
%       left boundary to be fixed

% ORIGINAL CODE DESCRIPTION -----------------------------------------------
% heatconvection2d_fvm_gs.m:
%
% Finite Volume Method (FVM) solution of the heat convection
% in a two-dimensional (2D) Cartesian domain.
%
% Numerical solution attained using the Gauss-Seidel (GS) iterative
% procedure instead of a direct solution of the linear system A * T = b.
%
% The temperature distribution T(x,y) due to heat convection through a
% rectangular domain of size Lx x Ly, with thermal conductivity k,
% volumetric heat capacity rhoCp, and volumetric heat generation g is
% described by:
%
%    rhoCp * Ux * dT/dx + rhoCp * Uy * dT/dy ...
%  - k * dT^2/dx^2 - k * dT^2/dy^2 + g = 0,    0 < x < Lx, 0 < y < Ly
%
% The boundary conditions for the problem are:
%
%    @ x = 0  (left  ):  k * dT/dx = hleft   * ( T - Taleft   )
%    @ x = Lx (right ): -k * dT/dx = hright  * ( T - Tariright  )
%    @ y = 0  (bottom):  k * dT/dy = hbottom * ( T - Tabottom )
%    @ y = Ly (top   ): -k * dT/dy = htop    * ( T - Tatop    )
%
% where hleft and hright are convective heat transfer coefficients, and
% Taleft and Tariright reference temperatures, respectively.
%
% Finite Volume Method (FVM) approximation: ==============================
%
% Physical domain:
%        ^
%        |
% (0, Ly)|           (top)           (Lx, Ly)
%        o---------------------------o
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%        |                           |
%  (left)|                           |(right)
%        |                           |
%        |                           |
```

```
%         |                            |
%         |                            |
%         |                            |
%         o----------------------------o ----> x
%  (0, 0)           (bottom)         (Lx, 0)
%
% Index domain:
% Finite Volume discrettization - capital letters: nodes, lowercase: faces
%
%  (1, 1)            (bottom)         (1, N)
%         o----------------------------o -----> j (equivalent to x axis)
%         |                            |
%         |                            |
%         |           (South, S)       |
%         |            i-1,j           |
%         |              |             |
%  (left) |    i,j-1 - i,j - i,j+1     |(right)
%         | (West, W)    |   (East, E) |
%         |            i+1,j           |
%         |           (North, N)       |
%         |                            |
%         |                            |
%         o----------------------------o
%   (N,1) |             (top)          (N, N)
%         |
%         i (equivalent to y axis)
%
%-------------------------------------------------------------------------

function [deltaT,Mean,Max,iter] = funConv(Lx,Lf)

%-------------------------------------------------------------------------
% Inputs
%-------------------------------------------------------------------------

% material properties:
rhoCp    = 800*2000;    % [J/m^3/K] volumetric heat capacity (1e3)
k        = 0.5;     % [W/m/K]   thermal conductivity (1.0)
g        = 0.0;     % [W/m^3]   volumetric heat generation (0.0)
mu =  0.001;
rho = 800;
Cp = 2000;

% geometry:
Ly       = 0.2;     % [m]       length of domain along y (0.1, 0.2)

% Dimensionless Parametrs
Rex = @(x,v) rho*x*v/mu;
Pr = Cp*mu/k;

% Additional conditions
T0 = 100;          % [C]
Tinf = 10;         % [C]
kf = 200;
hc = 10;
```

```
PdivA = 100;

% Velocity:
Uxmax = 0.001;       % [m/s]    maximum velocity
Uxfun = @(x,y) UXFUNCTION(x,y);
Uyfun = @( x, y )(0);% [m/s]

% Boundary layer functions
del = @(x) ((280/13)*mu*x/(rho*Uxmax))^.5;
delT = @(x) 4.53*x/(((Rex(x, Uxmax))^.5)*(Pr^(1/3)));

% Additional parameters
he = hc;
m = sqrt(PdivA*(hc/kf));
mL = m*Lf;
D1 = kf*m*((sinh(mL) + (he/(m*kf))*cosh(mL))/(cosh(mL) + ...
    (he/(m*kf))*sinh(mL)));

%kmt = kf*m*tanh(m*Lf); % Uncomment this for simple fin equation
kmt = D1;              % Uncomment this for general fin equation

% Boundary conditions:
Taleft   =   T0;    % [C]       ambient temperature - left (100, 0)
Taright  =   0;     % [C]       ambient temperature - right
Tatop    =   0;     % [C]       ambient temperature - top

hleft    = 1e20;    % [W/m^2/K] convective coefficient - left (1e8)
hright   = 0.0;     % [W/m^2/K] convective coefficient - right (0)
htop     = 0.0;     % [W/m^2/K] convective coefficient - top (0)

% solution parameters: ==================================================
Nx       = 200;    % number of nodes along x (200)
Ny       = 40;    % number of nodes along y (20, 40)
tol      = 1.0e-4; % tolerance for solution (1.0e-5)

% Upwind function:
funUpwind = @( P )( max( 0.0, ( 1.0 - 0.1 * abs( P ) )^5 ) ); % power-law
% upwind

%-------------------------------------------------------------------------
% Set up iterative solution
%-------------------------------------------------------------------------

dx = Lx / ( Nx - 1 );      % discrete x differential
x  = 0 : dx : Lx;          % discrete x axis, x(1) = 0, x(Nx) = Lx

dy = Ly / ( Ny - 1 );      % discrete y differential
y  = 0 : dy : Ly;          % discrete y axis, y(1) = 0, y(Ny) = Ly

T = zeros( Ny, Nx );       % discrete solution, initial guess

% Create an arbitrary initial temperature distribution guess
for i = 1:Ny
    for j = 1:Nx
```

```
        T(i,j)    = 100 + 1*sin(i*pi/10) + 1*cos(j*pi/10);
    end
end

% Define parameters to initialize the iterations
Told    = T + 0.5;              % auxiliary array to store previous solution
error   = 100 * tol;       % initialize error (any large number > tol)
nite    = 0;               % iteration counter
nitemax = 1000;            % maximum number of iterations

%--------------------------------------------------------------------------
% Iterative solver
%--------------------------------------------------------------------------

tic
while ( error > tol ) && ( nite < nitemax )

    % Update iteration counter:
    nite = nite + 1;

    % Update temperature from the old values:
    for i = 1 : Ny
        for j = 1 : Nx

            % Boundary nodes:
            if     j ==  1 % Left
                T( i, j ) = ( T( i, j+1 ) + ( hleft   * dx / k ) * Taleft   )
/ ...
                    ( 1           + ( hleft   * dx / k ) );

            elseif j == Nx % Right
                T( i, j ) = ( T( i, j-1 ) + ( hright  * dx / k ) * Taright  )
/ ...
                    ( 1           + ( hright  * dx / k ) );

            elseif i ==  1 % Bottom
                T( i, j ) = ( T( i+1, j )*(k/dy) + Tinf*0.5*(kmt + hc)) / ...
                    ( 0.5*(kmt + hc)+ k/dy);

            elseif i == Ny % Top
                T( i, j ) = ( T( i-1, j ) + ( htop    * dy / k ) * Tatop    )
/ ...
                    ( 1           + ( htop    * dy / k ) );

            else % Inside nodes:

                % Coordinates around control volume:
                xe = x( j ) - dx / 2; ye = y( i )         ;
                xw = x( j ) + dx / 2; yw = y( i )         ;
                xn = x( j )         ; yn = y( i ) + dy / 2;
                xs = x( j )         ; ys = y( i ) - dy / 2;

                % Velocities normal the control volume faces:
                Uxe = Uxfun( xe, ye );
```

```
                Uxw = Uxfun( xw, yw );
                Uyn = Uyfun( xn, yn );
                Uys = Uyfun( xs, ys );

                % Discrete advective fluxes:
                Fe = rhoCp * Uxe * dy;
                Fw = rhoCp * Uxw * dy;
                Fn = rhoCp * Uyn * dx;
                Fs = rhoCp * Uys * dx;

                % Discrete diffusive fluxes:
                De = k * dy / dx;
                Dw = k * dy / dx;
                Dn = k * dx / dy;
                Ds = k * dx / dy;

                % Local Peclet number on each face:
                Pe = Fe / De;
                Pw = Fw / Dw;
                Pn = Fn / Dn;
                Ps = Fs / Ds;

                % Coefficients:
                aE = De * funUpwind( Pe ) + max( -Fe, 0.0 );
                aW = Dw * funUpwind( Pw ) + max(  Fw, 0.0 );
                aN = Dn * funUpwind( Pn ) + max( -Fn, 0.0 );
                aS = Ds * funUpwind( Ps ) + max(  Fs, 0.0 );
                aP = aE + aW + aN + aS;
                b  = g * dx * dy;

                % Gauss-Seidel update: T( y, x )
                T( i, j ) = ( aW * T( i    , j - 1 ) + ...
                    aE * T( i    , j + 1 ) + ...
                    aS * T( i - 1, j     ) + ...
                    aN * T( i + 1, j     ) + b ) / aP;

            end
        end
    end

    % Update error:
    error = max( T(:) - Told(:) );

    % Store the solution from this iteration:
    Told = T;
end
toc, % stop timer

%--------------------------------------------------------------------------
% Post Processing
%--------------------------------------------------------------------------

% Screen output:
fprintf(' total number of iterations: %i \n', nite );
```

```
% Report characteristics of problem and solution:
fprintf(' Peclet number: %2.2f \n'      , rhoCp * Uxmax * Ly / k );
fprintf(' mean temperature: %2.2f \n'   , mean( T( : ) ) );
fprintf(' maximum temperature: %2.2f \n', max(  T( : ) ) );

%--------------------------------------------------------------------------
% Calculate parameters to be output
%--------------------------------------------------------------------------

dell = zeros(1,Nx);
delTT = dell;
for j = 1:Nx
    dell(j) = (4.96*x(j))/(Rex(x(j),Uxmax))^.5;
    delTT(j) = 4.53*x(j)/(((Rex(x(j),Uxmax))^.5)*(Pr^(1/3)));
end

eps = zeros(Ny,1);
Bnd = zeros(Ny,1);
for j = 2:Nx
    for i = 1:Ny
        eps(i) = 1e-9*i;
    end
    TCol = T(:,j) + eps;
    TBnd = 0.990*(max(T(:,j)) - min(T(:,j))) + min(T(:,j));
    Bnd(j) = interp1(TCol,y',TBnd);
end

iter = nite;
Mean =  mean( T( : ) );
Max =  max(  T( :,end ) );
deltaT =max(T,[],'all') - mean(T(:,end));

%--------------------------------------------------------------------------
% Nested functions
%--------------------------------------------------------------------------

% Boundary layer function
function ux = UXFUNCTION(x,y)
mu =  0.001;
rho = 800;
Uxmax = 0.001;

del = @(x) ((280/13)*(mu*x/(rho*Uxmax)))^.5;
if y >= del(x)
    ux = Uxmax;
else
    ux = Uxmax*((y/del(x)) - 0.5*(y/del(x))^3);
end
end
end
```