ME 250b

Advanced Convective Transport and Computational methods

Project 1: Fully Developed Heat Transfer in Channels with Non-circular Cross-sections

Edmund Chen
Matthew Eliceiri

March 5, 2020

ME 250b

University of California, Berkeley

Contents

List of Figures

Basic Model 1

§1.a. Equation Derivation

Definition 1.a.1 *Our system of equations governing flow in the tube is given by*

$$\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} = \frac{1}{\rho v}\frac{dP}{dz} \tag{1.1}$$

$$\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} = \frac{w}{\alpha}\left(\frac{dT_m}{dz}\right) \tag{1.2}$$

We adopt a central difference scheme for the derivatives on the LHS, yielding

$$\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{(\Delta x)^2} + \frac{w_{i,j+1} - 2w_{i,j} + w_{i,j-1}}{(\Delta y)^2} = \frac{1}{\rho v}\left(\frac{dP}{dz}\right)$$

$$\frac{T_{i+1,j} - 2T_{i,j} + T_{i-1,j}}{(\Delta x)^2} + \frac{T_{i,j+1} - 2T_{i,j} + T_{i,j-1}}{(\Delta y)^2} = \frac{w}{\alpha}\left(\frac{dT_m}{dz}\right)$$

Quite straightforwardly, we solve for the $w_{i,j}$ and $T_{i,j}$ respectively

$$w_{i,j}(-2((\Delta y)^2 + (\Delta x)^2)) = \frac{(\Delta x)^2(\Delta y)^2}{\rho v}\frac{dP}{dz} - (\Delta y)^2(w_{i+1,j} + w_{i-1,k}) - (\Delta x)^2(w_{i,j+1} + w_{i,j-1})$$

$$\begin{aligned} w_{i,j} &= -\frac{(\Delta x)^2(\Delta y)^2}{2((\Delta y)^2 + (\Delta x)^2)\rho v}\frac{dP}{dz} + \frac{(\Delta y)^2(w_{i+1,j} + w_{i-1,k}) + (\Delta x)^2(w_{i,j+1} + w_{i,j-1})}{2((\Delta y)^2 + (\Delta x)^2)} \\ &= -\frac{(\Delta y)^2}{(2(\Delta y/\Delta x)^2 + 2)\rho v}\frac{dP}{dz} + \frac{(\Delta y/\Delta x)^2(w_{i+1,j} + w_{i-1,k}) + w_{i,j+1} + w_{i,j-1}}{2(\Delta y/\Delta x)^2 + 2} \end{aligned}$$

hence the form described. An analogous process for T follows

$$T_{i,j}(-2((\Delta y)^2 + (\Delta x)^2)) = \frac{w_{i,j}(\Delta x)^2(\Delta y)^2}{\alpha}\frac{dT_m}{dz} - (\Delta y)^2(T_{i+1,j} + T_{i-1,k}) - (\Delta x)^2(T_{i,j+1} + T_{i,j-1})$$

$$\begin{aligned} T_{i,j} &= -\frac{w_{i,j}(\Delta x)^2(\Delta y)^2}{2((\Delta y)^2 + (\Delta x)^2)\alpha}\frac{dT_m}{dz} + \frac{(\Delta y)^2(T_{i+1,j} + T_{i-1,k}) + (\Delta x)^2(T_{i,j+1} + T_{i,j-1})}{2((\Delta y)^2 + (\Delta x)^2)} \\ &= -\frac{w_{i,j}(\Delta y)^2}{(2(\Delta y/\Delta x)^2 + 2)\alpha}\frac{dT_m}{dz} + \frac{(\Delta y/\Delta x)^2(T_{i+1,j} + T_{i-1,k}) + T_{i,j+1} + T_{i,j-1}}{2(\Delta y/\Delta x)^2 + 2} \end{aligned}$$

§1.b. Scripting Gauss-Seidel FDM

Iteration Calculation

The following is described in **julia**, however a subsequent version in Matlab was also partially used in developing the code. We first de-

fine the relevant constants needed as stated by the problem statement in (i)

```
dx = .45e-3;#m
dy = .45e-3;#m
dpdz=-17;#Pa/m
dtdz=7;#C/m
Tw = 85;#C
a = 1.46e-7
rho = 997
v = 8.26e-7
cp = 4164
k=.608
e_w = .0005
e_t = .05
Nx = 21
Ny = 61
```

Initialization of the W,T matrices and respective coordinate vectors can be seen by

```
W = zeros(Nx,Ny)
T = 70*ones(Nx,Ny)
x = collect(0:0.45:27)
y = collect(0:0.45:9)
```

Iterating across the grid for the W values as described by the [2] step while keeping an error index ϵ_w gives the following loop

```
while true
    e_max = 0
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
        W[i,j] = Wp;
    end
    if(e_max<e_w) break end
end
```

and similarly for [3] the same is done with the temperature

```
while true
    e_max = 0
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+T[i,j
    -1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy/dx)
    ^2+2))
    if(abs(T[i,j]-Tp)>e_max) e_max = abs(T[i,j]-Tp) end
    T[i,j] = Tp
    end
    if(e_max<e_t) break end
end
```

This yields the converged versions of T and W for the first task[1]

We induce the boundary wall temperature through the following code

```
T[[1,end],:] .= Tw
T[:,[1,end]] .= Tw
```

1: This will be defined as a function as it will be reused in the following questions

Relevant Statistics

1. To find the mean velocity and temperature we have

$$W_m = \int_\Omega W \qquad T_m = \frac{1}{W_m A_c} \int W T dA_c$$

As the boundary values are constant we can omit the boundaries with 1 indices and simply average the rest of the values divided by the size of the domain. Appropriately, we have the code

```
Wm = dx*dy*sum(W[2:end,2:end])/ys/xs
Tm=sum((dx*dy)*T[2:end,2:end].*W[2:end,2:end])/Wm/
    xs/ys
```

For this specific case, this yields the mean values

$$\boxed{w_m = .09369} \qquad \boxed{T_m = 44.8958}$$

2. Heat flux is determined by Fourier's law computed numerically using FDM

$$q_w = -k \nabla T$$

Expanding, we have

$$q_w = -k \left(\frac{\partial T}{\partial x} + \frac{\partial T}{\partial y} \right)$$

and since the boundary walls are either parallel to the horizontal or vertical axis, $\partial_x T$ or $\partial_y T$ will be negligible at any boundary point. Thus on the horizontal boundaries we evaluate

$$q = -k \frac{T_{i,j\pm1} - T_{i,j}}{\Delta y}$$

and respectively for the vertical boundaries we evaluate

$$q = -k \frac{T_{i\pm1,j} - T_{i,j}}{\Delta x}$$

Then the local heat transfer coefficient can be determined by[2]

$$h = \frac{q}{T_w - T_m}$$

2: T_w and T_m are wall and mean temperature respectively

In code, this is accomplished by

```
#heat flux
q = zeros(Nx,Ny)
q[1,:] = k.*(T[1,:]-T[2,:])./dx
q[:,1] = k.*(T[:,1]-T[:,2])./dy
q[end,:] = k.*(T[end,:]-T[end-1,:])./dx
q[:,end] = k.*(T[:,end]-T[:,end-1])./dy

#heat transfer coeff
hc=q./(Tw-Tm)
```

3. Lastly, the average heat transfer coefficient is found by

```
hmc=(sum(q[1,:]*dx)+sum(q[end,:]*dx)+sum(q[:,end]*
    dy)+sum(q[:,1]*dy))/(Tw-Tm)/(2*(.027+.009))
```

4. For plot generation we use the surf function from PyPlot

```
surf(x,y,W,facecolors=get_cmap("jet")(W/maximum(W))
    , linewidth=0, antialiased=false, shade=false)
```

which can be accordingly done for temperature aswell

§1.c. Results

We run the program described above to yield the plots

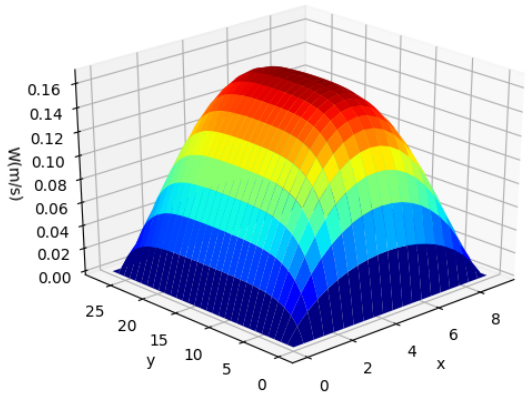

Figure 1.1: velocity 3d surface plot

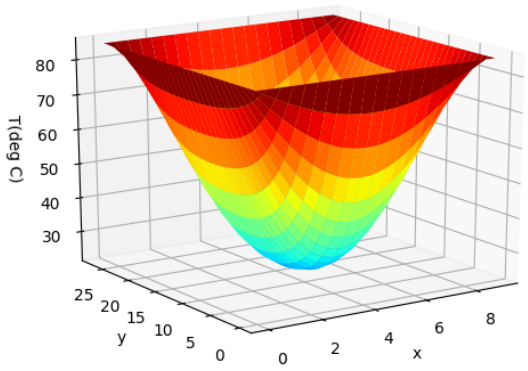

Figure 1.2: temp 3d surface plot

We plot the heat transfer coefficient values on the long and short walls as

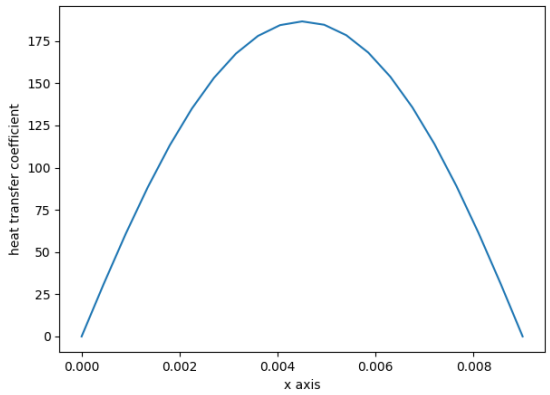

Figure 1.3: heat coeff. on x wall

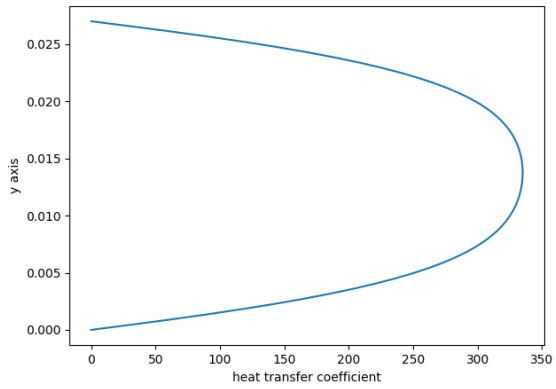

Figure 1.4: heat coeff. on y wall

The maximum and minimum values of the heat transfer coefficient are then seen to be

$$hc_{max} = hc(9\text{mm}, 14.5\text{mm}) = 339.3079 \qquad hc_{min} = hc(\text{corner nodes}) = 0$$

Additionally, the average heat transfer coefficient is seen by evaluating

```
hcm=dx*sum(q)/(Tw-Tm)/(2*(ys+xs))
```

which yields for this problem an average of

$$\text{heat transfer coeff}_{\text{mean}} = 158.6935$$

Through simple algebra, the hydraulic diameter is seen to be

$$D_H = \frac{4A_0}{p_w} = \frac{4 \times 0.000243m^3}{.072m} = 0.0135$$

The Nusselt number in this situation is calculated with a characteristic length equal to the hydraulic diameter, yielding

$$\text{Nu}_L = \frac{hL}{k} = \frac{158.6395 * 0.0135}{0.608} = 3.5224$$

Plotting this against the geometries described in the problem set, where the x axis denotes the 1:x scale rectangular domain, we have a rough consistency with Nusselt numbers

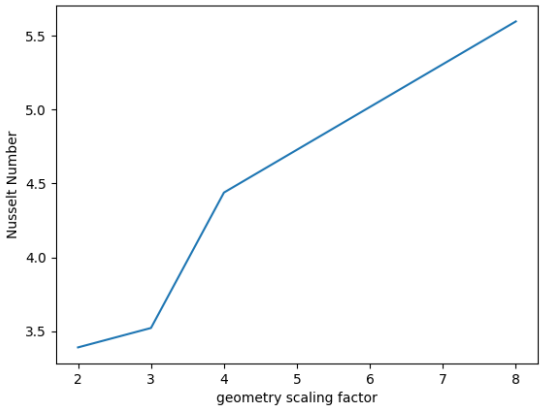

Figure 1.5: Nusselt Number Comparison

Similarly, for the Reynolds number we have the calculation

$$\mathrm{Re} = \frac{W_m \times D_h}{\nu} = \frac{0.0135 \times 0.09369}{8.26 \times 10^{-7}} = 1531.326$$

this is all scripted into the code as general equations, so we can easily return a set of statistics for the following tasks.

§1.d. Ethylene Glycol Mixture

We modify the above code for the given constants, and the respective plots yield

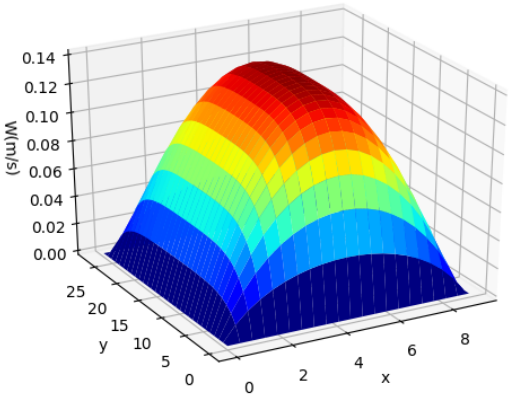

Figure 1.6: velocity 3d surface plot

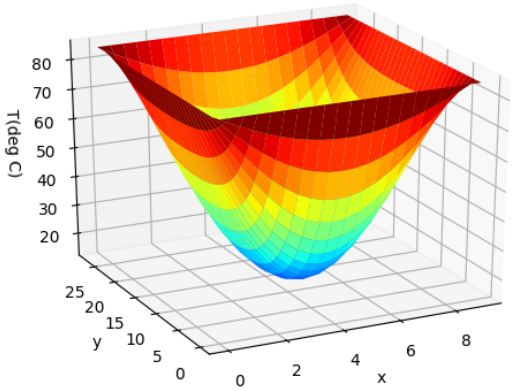

Figure 1.7: temp 3d surface plot

and the heat transfer coefficient, Nusselt numbers of

$$hc_{mean} = 106.3821 \qquad \mathrm{Nu}_L = 3.5286$$

Adiabatic Model 2

§2.a. Adiabatic Model

We modify the previous code to fit our new constraints, adding an additional line of code in the temperature loop to enforce a zero-flux condition on the right wall

```
T[end,:] = T[end-1,:]
```

The changed mesh size is simply accomplished by redefining the dimensions of the mesh to the appropriate values, $Nx = 21$ and $Ny = 41$ This yields the relevant 3d plots for velocity and temperature,

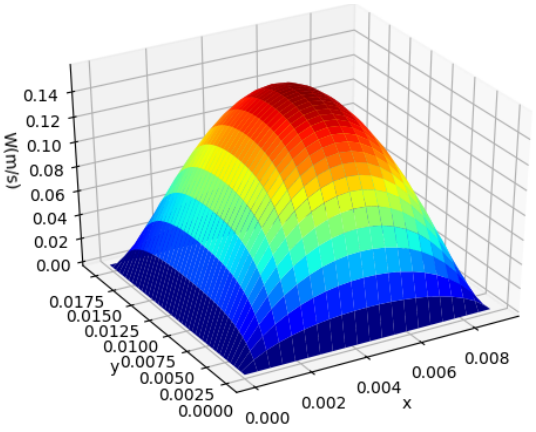

Figure 2.1: velocity 3d surface plot

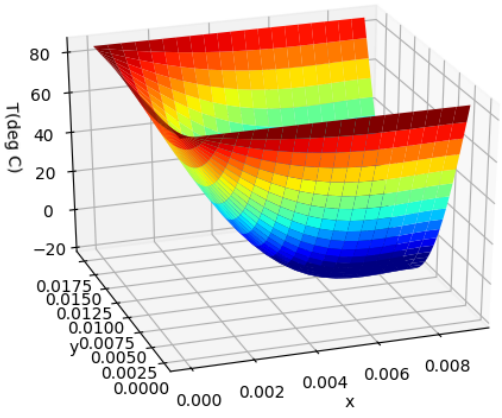

Figure 2.2: temp 3d surface plot

The zero flux condition on the right boundary as shown allows temperature to decrease on the RHS when converged, as compared to the previous examples.

§2.b. Statistics

Our relevant statistics are computed with the same formulas as in the previous problem as there are no null spaces in the computed domain, yielding

Statistic	Value
dh	0.012
Nu_L	2.5890
hc_m	109.290
Re	1191.0287

The Nusselt number of this problem is comparable with the Nusselt number for a square domain given.

Bent Model 3

§3.a. Bent Model

For the bent domain, we accordingly split each of the loops to iterate across their own rectangular sections, yielding the code

```
while true
    e_max = 0
    for i = 2:15; j = 2:20
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
    W[i,j] = Wp
    end
    for i = 7:15; j = 21:41
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
    W[i,j] = Wp
    end
    for i = 7:20; j = 42:60
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
    W[i,j] = Wp
    end
    if(e_max<e_w) break end
end
```

An analogous process can be followed for T, modified with the appropriate formula that was derived earlier. These calculations will yield the graphs

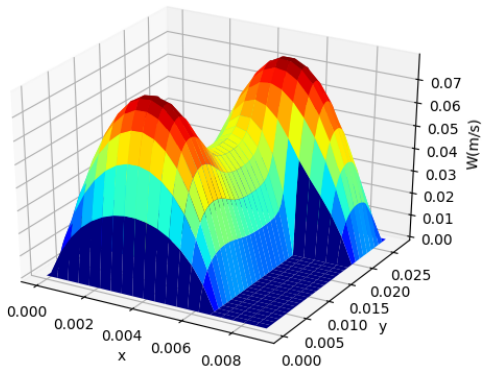

Figure 3.1: velocity 3d surface plot

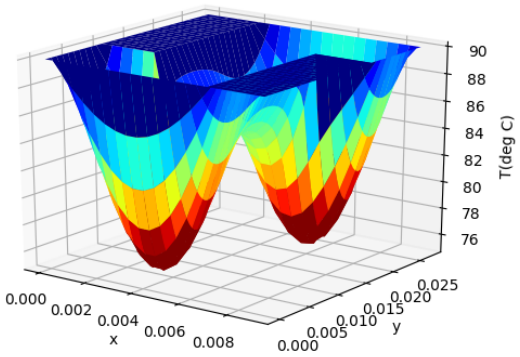

Figure 3.2: temp 3d surface plot

§3.b. Statistics

We require several modifications to properly evaluate the relevant statistics. Because the domain leaves some negative space, some simple algebra gives a new area of $0.000162m^2$ but a consistent perimeter. We introduce a simple loop to define the temperature at the negative space to the wall temperature

```
indices = findall(x->x==70,T)
for i in 1:size(indices)[1]
    T[indices[i]] = Tw
end
```

Note that as W_m of the negative space datapoints are 0, the $W \times T$ computation while determining T_m means no additional modifications to average temperature are needed. We only substitute the area calculation with a previously determined area variable, as the averages are multiplied across their respective local cross sections. The heat flux is determined on the selected boundaries by

```
q[1,1:21] = k.*(T[1,1:21] - T[2,1:21])./dx
q[end,41:end] = k.*(T[end,41:end] - T[end-1,41:end])./dx
q[6,21:end] = k.*(T[6,21:end] - T[7,21:end])./dx
q[16,1:41] = k.*(T[16,1:41] - T[15,1:41])./dx

q[1:6,21] = k.*(T[1:6,21] - T[1:6,20])./dy
q[1:16,1] = k.*(T[1:16,1] - T[1:16,2])./dy
q[6:end,end] = k.*(T[6:end,end] - T[6:end,end-1])./dy
q[16:end,41] = k.*(T[16:end,41] - T[16:end,42])./dy
```

This yields the relevant statistics

Statistic	Value
Nu_L	4.8822
hc_m	329.8242
Re	439.0593
dh	0.009
W_m	0.0402
T_m	81.399

Custom Model 4

§4.a. Determination and Scripts

We choose to make a plus shaped geometry. The dimensions will be a 8 node width on each side with a maximum length/width of 61 nodes. The step sizes are maintained from previous tasks. Our domains are then split accordingly for the W_p

```
for c = 1:30000
    e_max = 0
    for i = 27:35
        for j = 2:26
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end
    for i = 2:60
        for j = 27:35
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end
     for i = 27:35
        for j = 36:60
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end
    if(e_max<e_w)
        break
    end

end
```

An analogous process can be followed to converge the temperature, with the same equations that we derived earlier. This yields

the graphs for velocity and temperature as

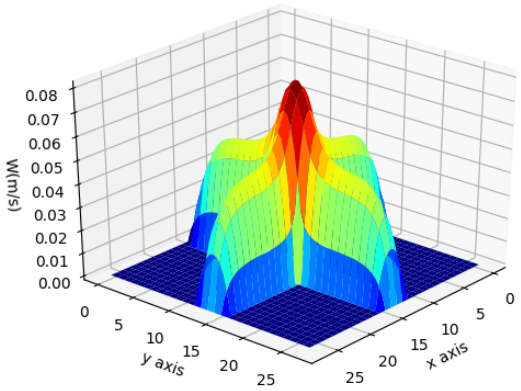

Figure 4.1: velocity 3d surface plot

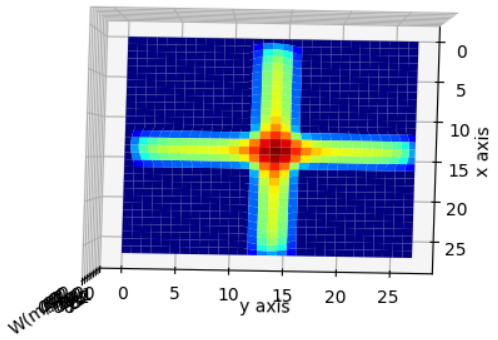

Figure 4.2: velocity 2d surface plot

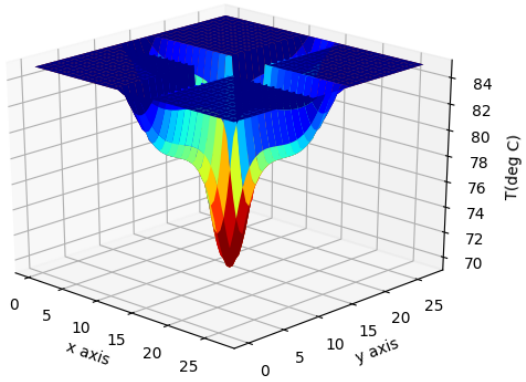

Figure 4.3: temp 3d surface plot

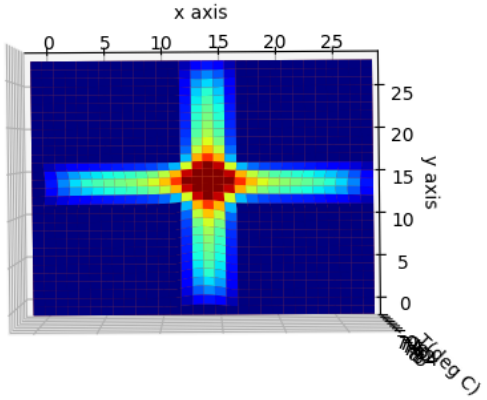

Figure 4.4: temp 2d surface plot

§4.b. Statistics

We use the same format as from the previous calculations with the bent model and get the following code for heat flux

```
q = zeros(Nx,Ny)

#fix pm coeffs
q[1:26,26] = k.*(T[1:26,26] - T[1:26,27])./dy
q[26,1:26] = k.*(T[26,1:26] - T[27,1:26])./dx

q[26:36,1] = k.*(T[26:36,1] - T[26:36,2])./dy
q[1,26:36] = k.*(T[1,26:36] - T[2,26:36])./dx

q[1:26,36] = k.*(T[1:26,36] - T[1:26,35])./dy
q[36,1:26] = k.*(T[36,1:26] - T[35,1:26])./dx

q[30:61,26] = k.*(T[30:61,26] - T[30:61,27])./dy
q[26,30:61] = k.*(T[26,30:61] - T[27,30:61])./dx

q[36:61,36] = k.*(T[36:61,36] - T[36:61,35])./dy
q[36,36:61] = k.*(T[36,36:61] - T[35,36:61])./dx

q[26:36,61] = k.*(T[26:36,61] - T[26:36,60])./dy
q[61,26:36] = k.*(T[61,26:36] - T[60,26:36])./dy
```

Statistic	Value
Nu_L	4.0194
hc_m	363.6684
Re	334.7014
dh	0.00672
W_m	0.0411
T_m	78.7803

Work Organization 5

For this project, Ed first did the equation derivations. Matthew developed and ran the converging solutions for T and W across all the four different tasks, whereupon Ed took those results to calculate the relevant statistics and writeup conclusions. Our division of work made it easy for both of us to work on the project. Please find any related flow charts below.

APPENDIX

Appendix A

§1.a. Julia Code, Task 1

```
using PyPlot

param = 2 # 1 for water only 2 for mix with ethy

if (param==1)
    dx = .45e-3
    dy = .45e-3
    dpdz=-17;
    dtdz=7;
    Tw = 85;
    a = 1.46e-7
    rho = 997
    v = 8.26e-7
    cp = 4164
    k=.608
    e_w = .0005
    e_t = .05
    title = "water only"
else
    # ethyn mix
    dx = .45e-3;
    dy = .45e-3;
    dpdz=-17;
    dtdz=7;
    Tw = 85;
    a = 1.08e-7;
    rho = 1055;
    v = 9e-7;
    cp = 3559;
    k=.407;
    e_w = .0005;
    e_t = .05;
    title = "ethy and water mix"
end

Nx = 21;
Ny = 61;
xs = .009
ys = .027

#datadeclaration
W = zeros(Nx,Ny)
T = 70*ones(Nx,Ny)
T[[1,end],:] .= Tw
T[:,[1,end]] .= Tw

while true
    e_max = 0
```

```
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
    W[i,j] = Wp;
    end
    if(e_max<e_w) break end
end

while true
    e_max = 0
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+T[i,j
    -1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy/dx)
    ^2+2))
    if(abs(T[i,j]-Tp)>e_max) e_max = abs(T[i,j]-Tp) end
    T[i,j] = Tp
    end
    if(e_max<e_t) break end
end


#mean velocity and temperature
Wm = dx*dy*sum(W[2:end,2:end])/ys/xs
Tm=sum((dx*dy)*T[2:end,2:end].*W[2:end,2:end])/Wm/xs/ys
#Tm = dx*dy*sum(T[2:end,2:end])/ys/xs

#heat flux
q = zeros(Nx,Ny)
q[1,:] = k.*(T[1,:]-T[2,:])./dx
q[:,1] = -k.*(T[:,1]-T[:,2])./dy
q[end,:] = k.*(T[end,:]-T[end-1,:])./dx
q[:,end] = -k.*(T[:,end]-T[:,end-1])./dy

#heat transfer coeff
hc=q./(Tw-Tm)

#mean hc
hcm=(sum(q[1,:]*dx)+sum(q[end,:]*dx)+sum(q[:,end]*dy)+
    sum(q[:,1]*dy))/(Tw-Tm)/(2*(ys+xs))

#graphers
W=W';T=T'
x = collect(0:dx:xs)
y = collect(0:dy:ys)
figure()
plt1=surf(x,y,W,facecolors=get_cmap("jet")(W/maximum(W))
    , linewidth=0, shade=false)
xlabel("x"); ylabel("y"); zlabel("W(m/s)")
figure()
plt2=surf(x,y,T,facecolors=get_cmap("jet")(T/maximum(T))
    , linewidth=0, shade=false)
xlabel("x"); ylabel("y"); zlabel("T(deg C)")

#dimensionless numbers
dh = (4*xs*ys)/(2*(xs+ys))
Nu = (hcm*dh)/k
```

```
Re = (Wm*dh)/v

println(title)
println("Reynolds Number = ", Re)
println("Nusselt Number = ", Nu)
println("Heat transfer Coeff = ", hcm)
```

§1.b. Julia Code, Task 2

```
#water, adibatic

dx = .45e-3;#m
dy = .45e-3;#m
dpdz=-17;#Pa/m
dtdz=7;#C/m
Tw = 85;#C
a = 1.46e-7
rho = 997
v = 8.26e-7
cp = 4164
k=.608
e_w = .0005
e_t = .05
Nx = 21
Ny = 41
xs = .009
ys = .018

W = zeros(Nx,Ny)
T = 70*ones(Nx,Ny)
T[[1,end],:] .= Tw
T[:,[1,end]] .= Tw

while true
    e_max = 0
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+W[i,j
    -1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx)
    ^2+2))
    if(abs(W[i,j]-Wp)>e_max) e_max = abs(W[i,j]-Wp) end
    W[i,j] = Wp;
    end
    if(e_max<e_w) break end
end

while true
    e_max = 0
    for i = 2:(Nx-1), j = 2:(Ny-1)
    Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+T[i,j
    -1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy/dx)
    ^2+2))
    if(abs(T[i,j]-Tp)>e_max) e_max = abs(T[i,j]-Tp) end
    T[i,j] = Tp
    end
    T[end,:] = T[end-1,:]
    if(e_max<e_t) break end
end
```

```
45
46 #mean velocity and temperature
47 Wm = dx*dy*sum(W[2:end,2:end])/ys/xs
48 Tm=sum((dx*dy)*T[2:end,2:end].*W[2:end,2:end])/Wm/xs/ys
49
50 #heat flux
51 q = zeros(Nx,Ny)
52 q[1,:] = k.*(T[1,:]-T[2,:])./dx
53 q[:,1] = k.*(T[:,1]-T[:,2])./dy
54 q[end,:] = k.*(T[end,:]-T[end-1,:])./dx
55 q[:,end] = k.*(T[:,end]-T[:,end-1])./dy
56
57 #heat transfer coeff
58 hc=q./(Tw-Tm)
59
60 #mean hc
61 hcm=(sum(q[1,:]*dx)+sum(q[end,:]*dx)+sum(q[:,end]*dy)+
       sum(q[:,1]*dy))/(Tw-Tm)/(2*(ys+xs))
62
63 #graphers
64 W=W';T=T'
65 x = collect(0:dx:xs)
66 y = collect(0:dy:ys)
67 figure()
68 plt1=surf(x,y,W,facecolors=get_cmap("jet")(W/maximum(W))
       , linewidth=0, shade=false)
69 xlabel("x"); ylabel("y"); zlabel("W(m/s)")
70 figure()
71 plt2=surf(x,y,T,facecolors=get_cmap("jet")(T/maximum(T))
       , linewidth=0, shade=false)
72 xlabel("x"); ylabel("y"); zlabel("T(deg C)")
73
74 #dimensionless numbers
75 dh = (4*xs*ys)/(2*(xs+ys))
76 Nu = (hcm*dh)/k
77 Re = (Wm*dh)/v
78
79 println("adibatic water")
80 println("Reynolds Number = ", Re)
81 println("Nusselt Number = ", Nu)
82 println("Heat transfer Coeff = ", hcm)
```

§1.c. Julia Code, Task 3

```
1  dx = .45e-3;#m
2  dy = .45e-3;#m
3  dpdz=-17;#Pa/m
4  dtdz=7;#C/m
5  Tw = 90;#C
6  a = 1.46e-7
7  rho = 997
8  v = 8.26e-7
9  cp = 4164
10 k=.608
11 e_w = .0005
12 e_t = .05
13 Nx = 21
```

```
Ny = 61
xs = .009
ys = .027
area = xs*ys-(dx*dy*5*40*2)

W = zeros(Nx,Ny)
T = 70*ones(Nx,Ny)

for c = 1:30000
   e_max = 0
   for i = 2:15
       for j = 2:20
           Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+
    W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx
    )^2+2))
           if(abs(W[i,j]-Wp)>e_max)
               e_max = abs(W[i,j]-Wp)
           end
           W[i,j] = Wp
       end
   end
   for i = 7:15
       for j = 21:41
           Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+
    W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx
    )^2+2))
           if(abs(W[i,j]-Wp)>e_max)
               e_max = abs(W[i,j]-Wp)
           end
           W[i,j] = Wp
       end
   end
    for i = 7:20
       for j = 42:60
           Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j+1]+
    W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(dy/dx
    )^2+2))
           if(abs(W[i,j]-Wp)>e_max)
               e_max = abs(W[i,j]-Wp)
           end
           W[i,j] = Wp
       end
   end



   if(e_max<e_w)
       break
   end

end


#temp

T[1,1:21] .= Tw
T[end,41:end] .= Tw
T[1:16,1] .= Tw
```

```
T[6:end,end] .= Tw
T[1:6,21] .= Tw
T[6,21:end] .= Tw
T[16,1:41].=Tw
T[16:end,41] .= Tw


for c = 1:30000
   e_max = 0



    for i = 2:15
       for j = 2:20
           Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+
    T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy
    /dx)^2+2))
           if(abs(T[i,j]-Tp)>e_max)
               e_max = abs(T[i,j]-Tp);
           end
           T[i,j] = Tp
       end
   end
   for i = 7:15
       for j = 21:41
           Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+
    T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy
    /dx)^2+2))
           if(abs(T[i,j]-Tp)>e_max)
               e_max = abs(T[i,j]-Tp);
           end
           T[i,j] = Tp
       end
   end
    for i = 7:20
       for j = 42:60
           Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j+1]+
    T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a*(2*(dy
    /dx)^2+2))
           if(abs(T[i,j]-Tp)>e_max)
               e_max = abs(T[i,j]-Tp);
           end
           T[i,j] = Tp
       end
   end


   if(e_max<e_t)
       break
   end

end

#isolate the non-domain points
indices = findall(x->x==70,T)
for i in 1:size(indices)[1]
    T[indices[i]] = 90
end
```

```
118
119 #mean velocity and temperature
120 Wm = dx*dy*(sum(W[2:end,2:end])-sum(W[indices[:]]))/(
        area)
121 Tm=sum((dx*dy)*T[2:end,2:end].*W[2:end,2:end])/Wm/area
122
123 #heat flux
124 q = zeros(Nx,Ny)
125
126 #fix pm coeffs
127 q[1,1:21] = k.*(T[1,1:21] - T[2,1:21])./dx
128 q[end,41:end] = k.*(T[end,41:end] - T[end-1,41:end])./dx
129 q[6,21:end] = k.*(T[6,21:end] - T[7,21:end])./dx
130 q[16,1:41] = k.*(T[16,1:41] - T[15,1:41])./dx
131
132 q[1:6,21] = k.*(T[1:6,21] - T[1:6,20])./dy
133 q[1:16,1] = k.*(T[1:16,1] - T[1:16,2])./dy
134 q[6:end,end] = k.*(T[6:end,end] - T[6:end,end-1])./dy
135 q[16:end,41] = k.*(T[16:end,41] - T[16:end,42])./dy
136
137 #heat transfer coeff
138 hc=q./(Tw-Tm)
139
140 #mean hc
141 hcm=(sum(q)*dx)/(Tw-Tm)/(2*(ys+xs))
142
143 #dimensionless numbers
144 dh = (4*area)/(2*(xs+ys))
145 Nu = (hcm*dh)/k
146 Re = (Wm*dh)/v
147
148 println("adibatic water")
149 println("Reynolds Number = ", Re)
150 println("Nusselt Number = ", Nu)
151 println("Heat transfer Coeff = ", hcm)
152
153 #graphers
154 W=W';T=T'
155 x = collect(0:dx:xs)
156 y = collect(0:dy:ys)
157 figure()
158 plt1=surf(x,y,W,facecolors=get_cmap("jet")(W/maximum(W))
        , linewidth=0, shade=false)
159 xlabel("x"); ylabel("y"); zlabel("W(m/s)")
160 figure()
161 plt2=surf(x,y,T,facecolors=get_cmap("jet")((maximum(T).-
        T)/12), linewidth=0, shade=false)
162 xlabel("x"); ylabel("y"); zlabel("T(deg C)")
```

§1.d. Julia Code, Task 4

```
1 ## Cross tube
2 dx = .45e-3;#m
3 dy = .45e-3;#m
4 dpdz=-17;#Pa/m
5 dtdz=7;#C/m
6 Tw = 85;#C
```

```
a = 1.46e-7
rho = 997
v = 8.26e-7
cp = 4164
k=.608
e_w = .0005
e_t = .05
Nx = 61
Ny = 61
xs=ys=.027
area= .00018144

W = zeros(Nx,Ny)
T = 70*ones(Nx,Ny)

for c = 1:30000
    e_max = 0
    for i = 27:35
        for j = 2:26
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end
    for i = 2:60
        for j = 27:35
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end
     for i = 27:35
        for j = 36:60
            Wp = ((dy/dx)^2*(W[i+1,j]+W[i-1,j])+W[i,j
    +1]+W[i,j-1])/(2*(dy/dx)^2+2)-dpdz*dy^2/(rho*v*(2*(
    dy/dx)^2+2))
            if(abs(W[i,j]-Wp)>e_max)
                e_max = abs(W[i,j]-Wp)
            end
            W[i,j] = Wp
        end
    end



    if(e_max<e_w)
        break
    end

end
```

```
59
60
61 #temp
62
63 T[1:26,26] .= Tw
64 T[26,1:26] .= Tw
65
66 T[1,26:36] .= Tw
67 T[26:36,1] .= Tw
68
69 T[1:26,36] .= Tw
70 T[36,1:26] .= Tw
71
72 T[26,36:61].= Tw
73 T[36:61,26].= Tw
74
75 T[36:61,36] .= Tw
76 T[36,36:61] .= Tw
77
78 T[61,26:36].=Tw
79 T[26:36,61].=Tw
80
81 for c = 1:30000
82     e_max = 0
83     for i = 27:35
84         for j = 2:26
85             Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j
    +1]+T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a
    *(2*(dy/dx)^2+2))
86             if(abs(T[i,j]-Tp)>e_max)
87                 e_max = abs(T[i,j]-Tp);
88             end
89             T[i,j] = Tp
90         end
91     end
92     for i = 2:60
93         for j = 27:35
94             Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j
    +1]+T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a
    *(2*(dy/dx)^2+2))
95             if(abs(T[i,j]-Tp)>e_max)
96                 e_max = abs(T[i,j]-Tp);
97             end
98             T[i,j] = Tp
99         end
100    end
101     for i = 27:35
102        for j = 36:60
103            Tp = ((dy/dx)^2*(T[i+1,j]+T[i-1,j])+T[i,j
    +1]+T[i,j-1])/(2*(dy/dx)^2+2)-W[i,j]*dtdz*dy^2/(a
    *(2*(dy/dx)^2+2))
104            if(abs(T[i,j]-Tp)>e_max)
105                e_max = abs(T[i,j]-Tp);
106            end
107            T[i,j] = Tp
108        end
109    end
110
```

```
111
112     if(e_max<e_t)
113         break
114     end
115
116 end
117
118 #isolate the non-domain points
119 indices = findall(x->x==70,T)
120 for i in 1:size(indices)[1]
121 T[indices[i]] = 85
122 end
123
124 #mean velocity and temperature
125 Wm = dx*dy*(sum(W[2:end,2:end])-sum(W[indices[:]]))/(
        area)
126 Tm=sum((dx*dy)*T[2:end,2:end].*W[2:end,2:end])/Wm/area
127
128 #heat flux
129 q = zeros(Nx,Ny)
130
131 #fix pm coeffs
132 q[1:26,26] = k.*(T[1:26,26] - T[1:26,27])./dy
133 q[26,1:26] = k.*(T[26,1:26] - T[27,1:26])./dx
134
135 q[26:36,1] = k.*(T[26:36,1] - T[26:36,2])./dy
136 q[1,26:36] = k.*(T[1,26:36] - T[2,26:36])./dx
137
138 q[1:26,36] = k.*(T[1:26,36] - T[1:26,35])./dy
139 q[36,1:26] = k.*(T[36,1:26] - T[35,1:26])./dx
140
141 q[30:61,26] = k.*(T[30:61,26] - T[30:61,27])./dy
142 q[26,30:61] = k.*(T[26,30:61] - T[27,30:61])./dx
143
144 q[36:61,36] = k.*(T[36:61,36] - T[36:61,35])./dy
145 q[36,36:61] = k.*(T[36,36:61] - T[35,36:61])./dx
146
147 q[26:36,61] = k.*(T[26:36,61] - T[26:36,60])./dy
148 q[61,26:36] = k.*(T[61,26:36] - T[60,26:36])./dy
149
150 #heat transfer coeff
151 hc=q./(Tw-Tm)
152
153 #mean hc
154 hcm=(sum(q)*dx)/(Tw-Tm)/(2*(ys+xs))
155
156 #dimensionless numbers
157 dh = (4*area)/(2*(xs+ys))
158 Nu = (hcm*dh)/k
159 Re = (Wm*dh)/v
160
161 println("adibatic water")
162 println("Reynolds Number = ", Re)
163 println("Nusselt Number = ", Nu)
164 println("Heat transfer Coeff = ", hcm)
165
166 #graphers
167 W=W';T=T'
```

```
168 x = collect(0:dx:xs)
169 y = collect(0:dy:ys)
170 figure()
171 plt1=surf(x,y,W,facecolors=get_cmap("jet")(W/maximum(W))
        , linewidth=0, shade=false)
172 xlabel("x"); ylabel("y"); zlabel("W(m/s)")
173 figure()
174 plt2=surf(x,y,T,facecolors=get_cmap("jet")((maximum(T).-
        T)/12), linewidth=0, shade=false)
175 xlabel("x"); ylabel("y"); zlabel("T(deg C)")
```

ME 250b

Advanced Convective Transport and Computational Methods

Project 2: Natural Convection Boundary Layer Flow

Edmund Chen
Matthew Eliceiri

April 14, 2020

ME 250b

University of California, Berkeley

Contents

List of Figures

Numerical Analysis of Boundary Layer Flow 1

§1.1. Basic Model Code

We seek to solve the following problem

Definition 1.1.1 (2d Convection Heat Transfer) *The system of equations invoking the usual boundary layer approximations, the Boussinesq approximations, yields the system of equations given by*

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{1.1}$$

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} = g\beta(T - T_\infty) + v\frac{\partial^2 u}{\partial y^2} \tag{1.2}$$

$$\frac{\partial T}{\partial t} + u\frac{\partial T}{\partial x} + v\frac{\partial T}{\partial y} = \alpha\frac{\partial^2 T}{\partial y^2} \tag{1.3}$$

First we consider the First-Order Upwind Scheme governed by the equation given by

$$T'_{i,j} = T_{i,j} + \left[\alpha\frac{T_{i,j+1} - T_{i,j} + T_{i,j-1}}{(\Delta y)^2} - u_{i,j}\frac{T_{i,j} - T_{i-1,j}}{\Delta x} - v_{i,j}\frac{T_{i,j+1} - T_{i,j}}{\Delta y}\right]\Delta t \tag{1.4}$$

$$u'_{i,j} = u_{i,j} + \left[g\beta(T'_{i,j} - T_\infty) + v\frac{u_{i,j+1} - 2u_{i,j} + u_{i,j-1}}{(\Delta y)^2} - u_{i,j}\frac{u_{i,j} - u_{i-1,j}}{\Delta x} - v_{i,j}\frac{u_{i,j+1} - u_{i,j}}{\Delta y}\right]\Delta t \tag{1.5}$$

$$v'_{i,j} - v'_{i,j-1} - \left[\frac{u'_{i,j} - u'_{i-1,j}}{\Delta x}\right]\Delta y \tag{1.6}$$

A Forward-Time-Central-Space algorithm is described in the problem set, which advances both the temperature and velocity fields in time.

Using Matlab and implementing the aforementioned algorithms, we design a similar program implementing the constants as shown,

```
while true
    for i = 2:45
        for j = 2:45
            Tp(i,j)= T(i,j)+(a*(T(i,j+1)-2*T(i,j)+T(i,j-1))/dy^2-u(i,j)*(T(i,j)-T(i
    -1,j))/dx-v(i,j)*(T(i,j+1)-T(i,j))/dy)*dt;
            up(i,j)= u(i,j)+(g*b*(Tp(i,j)-Tinf) +nu*(u(i,j+1)-2*u(i,j)+u(i,j-1))/dy
    ^2-u(i,j)*(u(i,j)-u(i-1,j))/dx-v(i,j)*(u(i,j+1)-u(i,j))/dy)*dt;
            vp(i,j)= vp(i,j-1)-(up(i,j)-up(i-1,j))/dx*dy;
        end
        Tp(i,1)=qw/k*dy+Tp(i,2);
    end
    T_hist  = [T_hist,Tp(45,1)];
    u_hist  = [u_hist,max(up(45,:))];
    Tp(46,:)=Tp(45,:);
    vp(46,:)=vp(45,:);
    up(46,:)=up(45,:);
    if(any(T-Tp,'all'))
```

```
    else
        break
    end
    u=up; T=Tp; v=vp;
end
```

This fundamental iterative algorithm will be reused throughout, though with varying parameters and modified boundary conditions. A first iteration to determine the final steady-state temperature is completed before running the test case. For reference, our steady state Temperature field is given by the contour

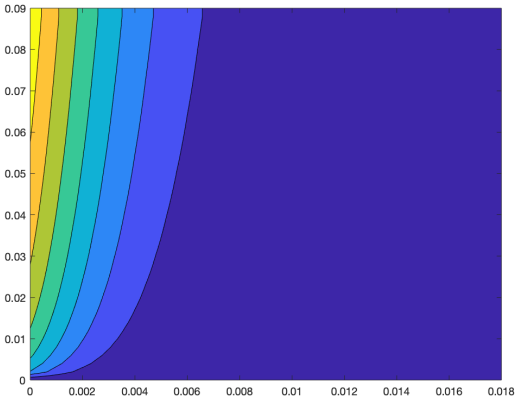

Figure 1.1: P1.1 Steady State Temperature Field

This steady state distribution is shown at time $t = 8.2465$. Again, running the time iteration with the parameter $q_w = 200W/m^2$ yields the time-dependent datapoints.

Time Datapoints

We break when the leading edge effect reaches the uppermost component through the conditional

```
if(Tp(44,1)==Tp(45,1) || flag == 1)
else
    time_leading_edge_300 = t*dt;
    flag = 1;
end
```

yielding a time of

$$t_{\text{edge=0.5195}}$$

Terminating the loop when 99% of SS is reached, we have

```
if(max(abs(T(45,:)-T_ss(45,:)))/Tinf <.01)
    break;
end
```

$$t_{99ss} = 1.115$$

Recalculating a steady state temperature field and recalculating the following time datapoints yields

$$t_{\text{edge}} = 0.447$$

$$t_{99ss} = 0.965$$

§1.2. Transients

Time Snapshots

In the previously described algorithm, we store these values in the `T_hist` and `u_list` variables, whereupon we can observe the variation of surface temperature

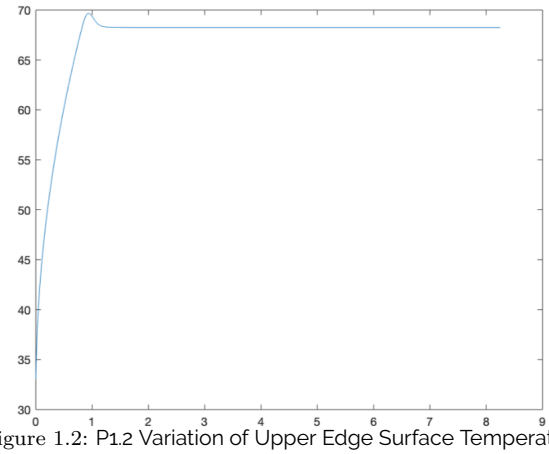

Figure 1.2: P1.2 Variation of Upper Edge Surface Temperature

As observed, there is a slight overshoot with the temperature before it reaches the steady state. ** Accordingly, the peak and steady state values of u velocity and surface temperature are as follows

$$T_{ss} = 82.8976 \qquad T_{max} = 84.9234$$

$$u_{ss} = 0.1905 \qquad u_{max} = 0.2019$$

Taking snapshots of the time distributions at $t = .164, .5595, .9365$, of which $t = .9365$ maximizes the wall temperature at the topmost x location.

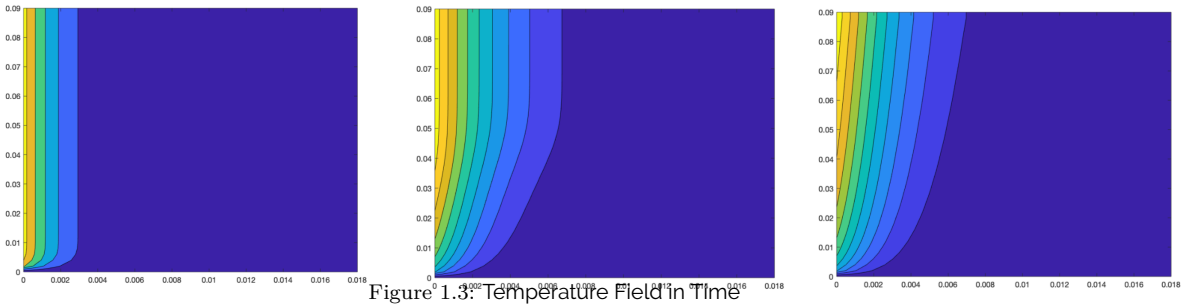

Figure 1.3: Temperature Field in Time

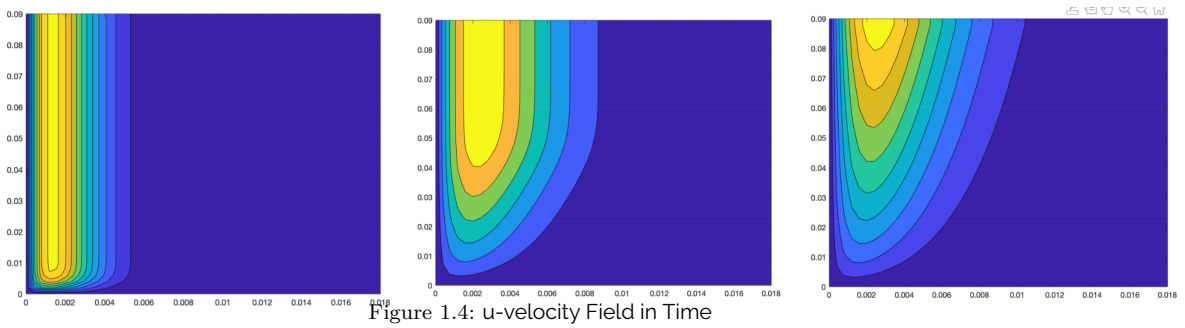

Figure 1.4: u-velocity Field in Time

They are ordered chronologically starting from the left with the three aforementioned times snapshotted going from left to right. Accordingly, the plotted v values

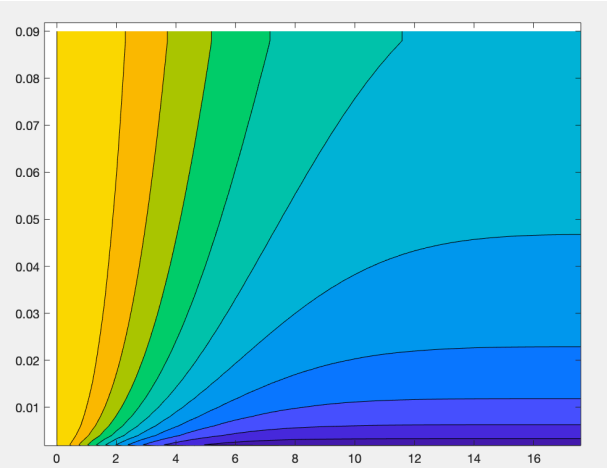

Figure 1.5: v 3d field plot

and on the outer edge of the boundary layer as a function of x

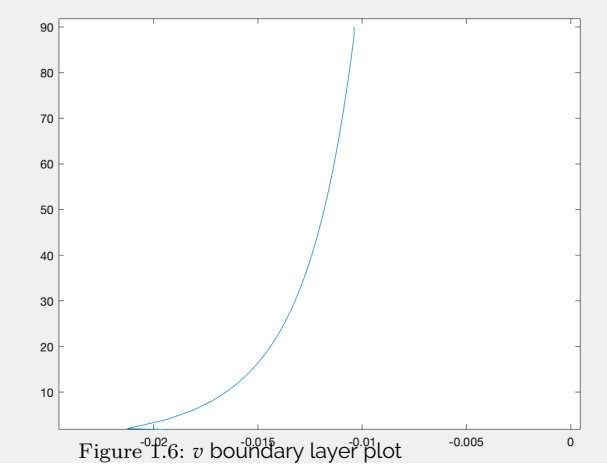

Figure 1.6: v boundary layer plot

As x approaches 0, v approaches 0.0213.

Steady-State Solutions

Firstly for $q_w = 150$ we have the following graphs

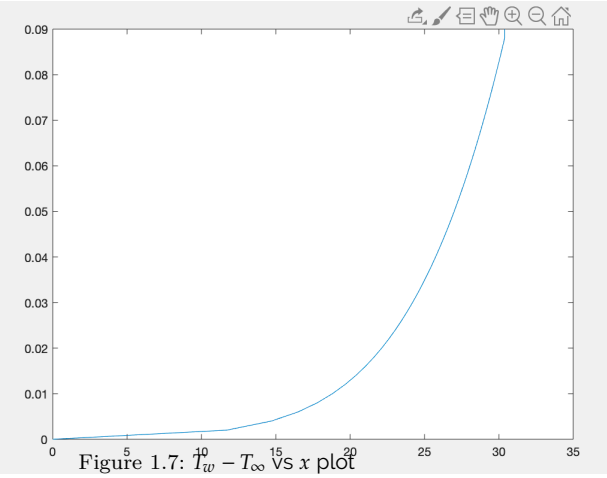

Figure 1.7: $T_w - T_\infty$ vs x plot

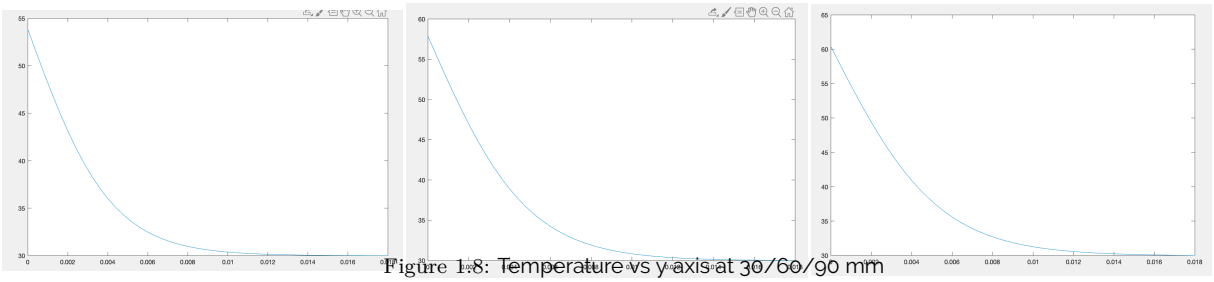

Figure 1.8: Temperature vs y axis at 30/60/90 mm

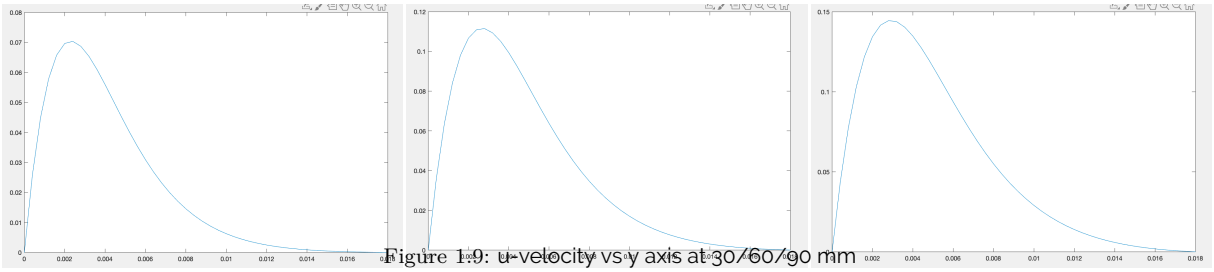

Figure 1.9: u-velocity vs y axis at 30/60/90 mm

following an analogous process for $q_w = 200$ we have

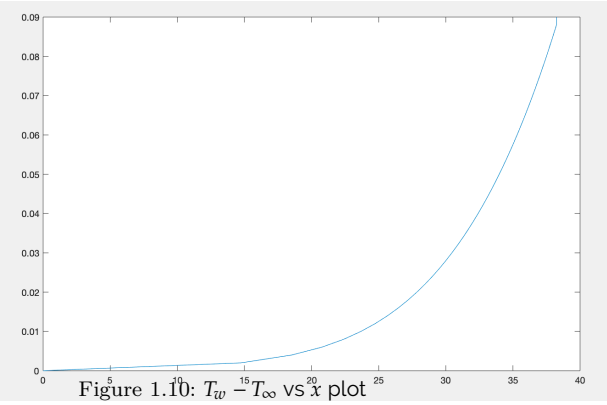

Figure 1.10: $T_w - T_\infty$ vs x plot

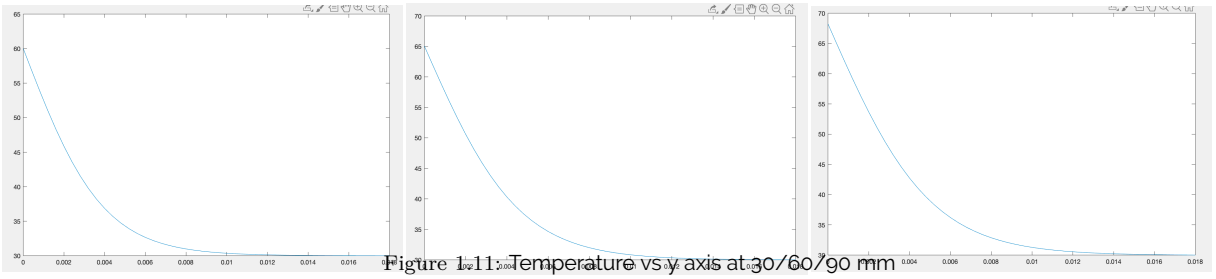

Figure 1.11: Temperature vs y axis at 30/60/90 mm

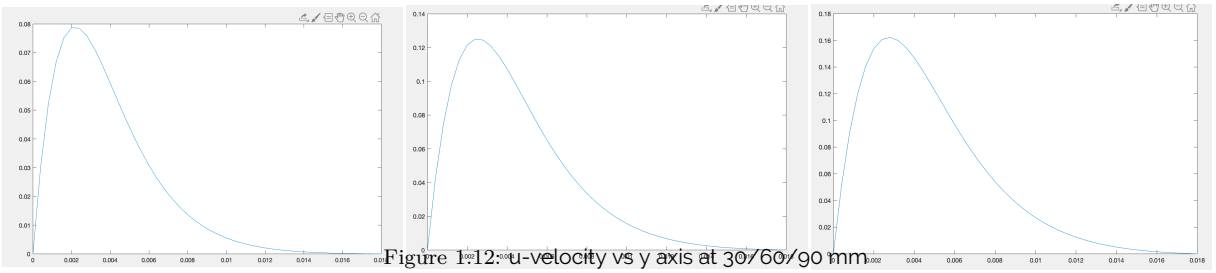

Figure 1.12: u-velocity vs y axis at 30/60/90 mm

hence all the plots specified. Laminar analysis

Integral Solution 3

§3.1. Derivation

Our initial equations present the problem

$$u = \frac{Ax^m y}{\delta}\left(1-\frac{y}{\delta}\right)^2$$
$$T = T_\infty + \frac{q_w B x^n}{2k}\left(1-\frac{y}{\delta}\right)^2$$
$$\frac{d}{dx}\int_0^\delta u^2 dy = -\nu\left[\frac{du}{dy}\right]_{y=0} + \int_0^\delta g\beta\left(T-T_\infty\right)dy$$

The first substitution gives

$$\frac{d}{dx}\int_0^\delta \left[\frac{Ax^m y}{\delta}\left(1-\frac{y}{\delta}\right)^2\right]^2 dy$$
$$= \frac{d}{dx}\int_0^\delta \left[\frac{A^2y^6x^{2m}}{\delta^6} - \frac{4A^2y^5x^{2m}}{\delta^5} + \frac{6A^2y^4x^{2m}}{\delta^4} - \frac{4A^2y^3x^{2m}}{\delta^3} + \frac{A^2y^2x^{2m}}{\delta^2}\right]dy$$
$$= \frac{d}{dx}\left[\frac{1}{105}A^2\delta x^{2m}\right]$$
$$= \frac{d}{dx}\left[\frac{1}{105}A^2Bx^{2m+n}\right]$$
$$= \frac{2m+n}{105}A^2Bx^{2m+n-1}$$
$$= \frac{d}{dx}\left[\frac{1}{105}\delta\hat{U}^2\right]$$

The second substitution gives

$$-\nu\left[\frac{du}{dy}\right]_{y=0} = -\nu\frac{d}{dy}\left[\frac{Ax^m y}{\delta}\left(1-\frac{y}{\delta}\right)^2\right]_{y=0}$$
$$= -\nu\left[\frac{Ax^m}{\delta} + \frac{3Ay^2x^m}{\delta^3} - \frac{4Ayx^m}{\delta^2}\right]$$
$$= -\frac{\nu Ax^m}{\delta}$$
$$= -\frac{\nu Ax^{m-n}}{B}$$
$$= -\frac{\nu\hat{U}}{\delta}$$

Substituting all three terms into the momentum balance equation yields:

$$\frac{1}{105}\frac{d}{dx}\left(ABx^{2m+n}\right) = -\frac{\nu Ax^{m-n}}{B} + g\beta\frac{q_w B^2x^{2n}}{6k}$$
$$\frac{2m+n}{105}A^2Bx^{2m+n-1} = -\frac{\nu Ax^{m-n}}{B} + g\beta\frac{q_w B^2x^{2n}}{6k}$$

$$\frac{1}{105}\frac{d}{dx}\left[\delta\hat{U}^2\right] = \frac{1}{3}g\beta\Delta T\delta - \frac{\nu\hat{U}}{\delta}$$

Next substitute forms of T and u into energy balance integral equation

$$\frac{d}{dx}\int_0^\delta u\,(T_\infty - T)\,dy = \alpha\left[\frac{\partial T}{\partial y}\right]_{y=0}$$

$$\frac{d}{dx}\int_0^\delta \frac{Ax^m y}{\delta}\left(1-\frac{y}{\delta}\right)^2\left(-\frac{q_w B x^n}{2k}\left(1-\frac{y}{\delta}\right)^2\right)dy = \alpha\frac{\partial}{\partial y}\left[\frac{q_w B x^n}{2k}\left(1-\frac{y}{\delta}\right)^2\right]_{y=0}$$

Left hand side

$$= \frac{d}{dx}\int_0^\delta\left[-\frac{ABqy^5x^{m+n}}{2\delta^5 k} + \frac{2ABqy^4x^{m+n}}{\delta^4 k} - \frac{3ABqy^3x^{m+n}}{\delta^3 k} + \frac{2ABqy^2x^{m+n}}{\delta^2 k} - \frac{ABqyx^{m+n}}{2\delta k}\right]dy$$

$$= \frac{d}{dx}\left[-\frac{ABq_w x^{m+n}\delta}{60k}\right]$$

$$= -\frac{AB^2 q_w(m+2n)x^{m+2n-1}}{60k}$$

$$= -\frac{1}{30}\frac{d}{dx}[\hat{U}\delta\Delta T]$$

Right hand side

$$= \alpha\frac{\partial}{\partial y}\left[\frac{q_w B x^n}{2k}\left(1-\frac{y}{\delta}\right)^2\right]_{y=0}$$

$$= \alpha\left[\frac{Bq_w x^n\left(\frac{y}{\delta^2}-\frac{1}{\delta}\right)}{k}\right]_{y=0}$$

$$= -\alpha\frac{Bq_w x^n}{\delta k}$$

$$= -\alpha\frac{q_w}{k}$$

$$= \frac{-2\alpha\Delta T}{\delta}$$

Combining above equations yields

$$-\frac{1}{30}\frac{d}{dx}[\hat{U}\delta\Delta T] = \frac{-2\alpha\Delta T}{\delta}$$

$$\frac{1}{60}\frac{d}{dx}[\hat{U}\delta\Delta T] = \frac{\alpha\Delta T}{\delta}$$

$$\frac{1}{60}\frac{d}{dx}\left(\frac{AB^2q_w x^{m+2n}}{k}\right) = \alpha\frac{q_w}{k}$$

$$\frac{AB^2(m+2n)x^{m+2n-1}}{60} = \alpha$$

to make the left hand terms have the same x dependence

$$m + 2n - 1 = 0$$

$$2m + n - 1 = 2n$$

$$m = \frac{3}{5}; n = \frac{1}{5}$$

Next plug in m and n, solve for a and b

$$\frac{2m+n}{105}A^2Bx^{2m+n-1} = -\frac{\nu Ax^{m-n}}{B} + g\beta\frac{q_w B^2 x^{2n}}{6k}$$
$$\frac{AB^2(m+2n)x^{m+2n-1}}{60} = \alpha$$
$$\frac{\frac{7}{5}}{105}A^2Bx^{\frac{2}{5}} = -\frac{\nu Ax^{\frac{2}{5}}}{B} + g\beta\frac{q_w B^2 x^{\frac{2}{5}}}{6k}$$
$$\frac{1}{75}A^2B = -\frac{\nu A}{B} + g\beta\frac{q_w B^2}{6k}$$
$$\frac{AB^2\left(\frac{3}{5}+\frac{2}{5}\right)x^0}{60} = \alpha$$
$$\frac{AB^2}{60} = \alpha$$

$$A = \frac{60\alpha}{B^2}$$
$$\frac{1}{75}\left(\frac{60\alpha}{B^2}\right)^2 B = -\frac{\nu}{B}\frac{60\alpha}{B^2} + g\beta\frac{q_w B^2}{6k}$$
$$\frac{48\alpha^2}{B^3} = -\frac{60\alpha\nu}{B^3} + g\beta\frac{q_w B^2}{6k}$$
$$48\alpha^2 = -60\alpha\nu + g\beta\frac{q_w B^5}{6k}$$
$$B = \left[\frac{6k}{g\beta q_w}\left(48\alpha^2 + 60\alpha\nu\right)\right]^{\frac{1}{5}}$$
$$A = \frac{60\alpha}{\left[\frac{6k}{g\beta q_w}\left(48\alpha^2 + 60\alpha\nu\right)\right]^{\frac{2}{5}}}$$
$$\delta = \left[\frac{6kx}{g\beta q_w}\left(48\alpha^2 + 60\alpha\nu\right)\right]^{\frac{1}{5}}$$
$$u = \frac{60\alpha x^{\frac{3}{5}}}{\left[\frac{6k}{g\beta q_w}\left(48\alpha^2 + 60\alpha\nu\right)\right]^{\frac{2}{5}}}\frac{y}{\delta}\left(1-\frac{y}{\delta}\right)^2$$
$$T = T_\infty + \frac{q_w\left[\frac{6kx}{g\beta q_w}\left(48\alpha^2 + 60\alpha\nu\right)\right]^{\frac{1}{5}}}{2k} * \left(1-\frac{y}{\delta}\right)^2$$

§3.2. Comparisons

APPENDIX