```
%% PRE-SCRIPT PREP %%
clc; %clear command line
clear; %clear all variables
close all; %close all other previously opened figures and windows

%% CONSTANTS %%
T_infinity = 20; %degrees Celsius
guess = 300; %later add input command here
deltaX = 1*10^-3;
promptX = {'Enter x-value as shown on the diagram:'};
promptY = {'Enter y-value as shown on the diagram:'};
defaultInput = {'1'};
dialogBoxDimensions = [1 40];

%% VARIABLES %%
count = 0; %number of iterations taken to meet convergence criteria
sum = 0; %variable for storing the sum of values in the 4 nodes around node to be↙
evaluated
tripped = false; %checks if an individual node met convergence criteria
flag = true; %flag which goes down when convergence criteria for every node is met,↙
allowing exit of Gauss-Seidel iteration loop
run = true;%flag variable to keep the program running
qPrime = 0;% storage variable for q'
qPrime1Vals = zeros(14,1); %storage matrix for
qPrimeConv = zeros(14);
qPrimeConvSum = 0;
nodeReq = true;

%% MATRIX SETUP %%
T_old = zeros(14); %create new matrix with 14 rows and columns populated by the number↙
0
for i=7:length(T_old)   %add T_infinity values. Optional extra, zeros can be replaced↙
with T_infinity to fill outer cells
        T_old(1,i) = 0;
        T_old(2,i) = 0;
        T_old(3,i) = 0;
        T_old(7,i) = 0;
        T_old(8,i) = 0;
        T_old(12,i) = 0;
        T_old(13,i) = 0;
        T_old(14,i) = 0;
end
h = zeros(14,1);

%% MAIN LOOP %%
while(run)
    %Prompt User Input%
    k = str2num(cell2mat(inputdlg('Enter value of k in W/mK','Please Input x-Value',↙
dialogBoxDimensions,defaultInput)));
    h_0 = str2num(cell2mat(inputdlg('Enter value of h-0 in W/(K*m^2)','Please Input x-↙
```

```
Value',dialogBoxDimensions,defaultInput)));
    convCriteria = str2num(cell2mat(inputdlg('Enter value of desired Convection↙
Criteria','Please Input x-Value',dialogBoxDimensions,defaultInput))); %#ok<*ST2NM>
    for i=1:length(T_old) %%adds guess value for nodes to be populated
        T_old(i,:) = guess;
    end %%for loop

    %% MAIN LOGIC %%
    for i=1:length(T_old) %add left wall boundary values
        T_old(i,1) = 300+10*(14-i);%%add left wall values where T(0,y)=(300+10y)
    end %%for loop
    T_new = T_old;

    while(flag)
         %create second array to store new values at the beginning of loop, copy new↙
values to T_old before changing the values in T_new again
        for i=1:length(T_new) %start for loop for y indices
            for j=1:length(T_new) %start for loop for x indices
                h(j) = h_0*(1+((j-1)-5)/(8));
                %X=2 to x=5
                if 1<j && j<6 %if x is between 1 and 6...
                    if i==1 % if it is a top node
                        T_new(i,j) = (1/2)*(0.5*T_old(i,j-1) + 0.5*T_old(i,j+1) + T_old↙
(i+1,j)); %eq. A
                    end
                    if i==14 % if it is a bottom node
                        T_new(i,j) = (1/2)*(0.5*T_old(i,j-1) + 0.5*T_old(i,j+1) + T_old↙
(i-1,j)); %eq. B
                    end
                    if 1<i && i<14 %if it is a central node
                        T_new(i,j) = (1/4)*(T_old(i,j-1) + T_old(i+1,j) + T_old(i,j+1)↙
+ T_old(i-1,j));%eq. C
                    end
                end
                if j==6 %if x = 6...
                    if (1<i && i<4) || (6<i && i<9) || (11<i && i<14) %if vertical↙
convection node
                        T_new(i,j) = (k/(2*k+h(j)*deltaX))*(T_old(i,j-1) + 0.5*T_old↙
(i+1,j) + 0.5*T_old(i-1,j) + ((h(j)*deltaX)/k)*T_infinity); %eq D
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end
                    if i==4 || i==9 %if top left fin corner
                        T_new(i,j) = (1/(3+((h(j)*deltaX)/k)))*(0.5*T_old(i-1,j) +↙
T_old(i,j-1) + T_old(i+1,j) + 0.5*T_old(i,j+1) + ((h(j)*deltaX)/k)*T_infinity); %eq. I
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end
                    if i==5 || i==10 %if fin base internal node
                        T_new(i,j) = (1/4)*(T_old(i,j-1) + T_old(i+1,j) + T_old(i,j+1)↙
+ T_old(i-1,j));%eq. C
                    end
```

```
                    if i==6 || i==11 %if bottom left fin corner
                        T_new(i,j) = (1/(3+((h(j)*deltaX)/k)))*(T_old(i-1,j) + T_old(i,↙
j-1) + 0.5*T_old(i+1,j) + 0.5*T_old(i,j+1) + ((h(j)*deltaX)/k)*T_infinity);%eq. J
                        qPrimeConv(i,j)= h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end
                    if i==1 %if top corner node (y=1)
                        T_new(i,j) = (1/(2*k+h(j)*deltaX))*(k*T_old(i,j-1) + k*T_old↙
(i+1,j) + h(j)*deltaX*T_infinity);%eq. G
                        qPrimeConv(i,j) = h(j)*deltaX*(0.5)*(T_new(i,j)-T_infinity);
                    end
                    if i==14 %if bottom corner node (y=14)
                        T_new(i,j) = (1/(2*k+h(j)*deltaX))*(k*T_old(i,j-1) + k*T_old(i-↙
1,j) + h(j)*deltaX*T_infinity);%eq. H
                        qPrimeConv(i,j) = h(j)*deltaX*(0.5)*(T_new(i,j)-T_infinity);
                    end
                end
                if 6<j && j<14 %if x is between 6 and 14
                    if i==4 || i==9 %if top fin surface node
                        T_new(i,j) = (1/(2*k+h(j)*deltaX))*((k/2)*T_old(i,j-1) +↙
k*T_old(i+1,j) + (k/2)*T_old(i,j+1) + h(j)*deltaX*T_infinity);%eq. E
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end %if i==4...
                    if i==5 || i==10 %if fin internal node
                        T_new(i,j) = (1/4)*(T_old(i,j-1) + T_old(i+1,j) + T_old(i,j+1)↙
+ T_old(i-1,j));%eq. C
                    end %if i==5
                    if i==6 || i==11 %if bottom fin surface node
                        T_new(i,j) = (1/(2*k+h(j)*deltaX))*((k/2)*T_old(i,j-1) +↙
k*T_old(i-1,j) + (k/2)*T_old(i,j+1) + h(j)*deltaX*T_infinity); %eq. F
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end %if i==6
                end %if 6<j...
                if j==14 %if it is a far right node
                    if i==4 || i==9 %if fin top right corner
                        T_new(i,j) = (1/(1+((h(j)*deltaX)/k)))*(0.5*T_old(i,j-1) + 0.5↙
*T_old(i+1,j) + ((h(j)*deltaX)/k)*T_infinity); %eq. K
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end %if i==4...
                    if i==5 || i==10 %if fin right vertical
                        T_new(i,j) = (k/(2*k+h(j)*deltaX))*(T_old(i,j-1) + 0.5*T_old↙
(i+1,j) + 0.5*T_old(i-1,j) + ((h(j)*deltaX)/k)*T_infinity); %eq D
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end %if i==5
                    if i==6 || i==11 %if fin bottom right corner
                        T_new(i,j) = (1/(1+((h(j)*deltaX)/k)))*(0.5*T_old(i,j-1) + 0.5↙
*T_old(i-1,j) + ((h(j)*deltaX)/k)*T_infinity); %eq. L
                        qPrimeConv(i,j) = h(j)*deltaX*(T_new(i,j)-T_infinity);
                    end %if i==6
                end %if j==14
                if j==1 %calculate q' using left wall
```

```
                    if i==1 || i==14
                        qPrime1Vals(i) = 0.5*k*(-T_new(i,j+1)+T_new(i,j));
                    else
                        qPrime1Vals(i) = k*(-T_new(i,j+1)+T_new(i,j));
                    end
                end
                %Check convergence criteria:
                if abs(T_new(i,j)-T_old(i,j))>convCriteria
                    tripped = true; %if any value is above conv criteria, this
                                    %becomes true and flag will stay up
                end %if abs(...
            end %for j
        end %for i
        count = count +1; %increase iteration count by 1
        if tripped
            flag = true;
            T_old = T_new; %move new values to old matrix for next iteration
        else
            flag = false;
            for i=1:length(qPrime1Vals)
                qPrime = qPrime+qPrime1Vals(i);
            end
            test = nonzeros(qPrimeConv);
            qPrimeConvSum = cumsum(test);
        end %if/else
        tripped = false; %reset tripped for next iteration
    end %while flag

    %% PRINT FINAL VALUES %%
    fprintf('\t\tBase Temperature: \tTip Temperature:\n'); %utilize fprinf function to↙
print values.
    fprintf('y=4 \t'); %this is a highly inefficient way to print things. With more↙
time, could add a more elegent solution.
    fprintf('%4.4f', T_new(4,6)); %use %4.4f delimiter to display up to 4 digits before↙
decimal, and 4 digits after
    fprintf('\t\t\t ');
    fprintf('%4.4f', T_new(4,14));
    fprintf('\ny=5 \t');
    fprintf('%4.4f', T_new(5,6));
    fprintf('\t\t\t ');
    fprintf('%4.4f', T_new(5,14));
    fprintf('\ny=6 \t');
    fprintf('%4.4f', T_new(6,6));
    fprintf('\t\t\t ');
    fprintf('%4.4f', T_new(6,14));

    fprintf('\n'); %add a space for clarity
    fprintf('\ny=9 \t');
    fprintf('%4.4f', T_new(9,6));
    fprintf('\t\t\t ');
```

```
    fprintf('%4.4f', T_new(9,14));
    fprintf('\ny=10 \t');
    fprintf('%4.4f', T_new(10,6));
    fprintf('\t\t\t ');
    fprintf('%4.4f', T_new(10,14));
    fprintf('\ny=11 \t');
    fprintf('%4.4f', T_new(11,6));
    fprintf('\t\t\t ');
    fprintf('%4.4f', T_new(11,14));
    fprintf('\n');

    %PRINT q'%
    fprintf('\nq''through left wall: ');
    fprintf('%4.4f',qPrime);
    fprintf('\nq''through convecting surface: ');
    fprintf('%4.4f',qPrimeConvSum(46));

    %PRINT NUMBER OF ITERATIONS%
    fprintf('\nNumber of Iterations: ');
    fprintf('%5d', count);
    fprintf('\n');

    %% POST PROCESS USER INTERFACE %%
    while nodeReq
        answer1 = questdlg('Would you like to see another node value?', 'Want More↙
Info?', 'Yes', 'No', 'No');
        switch answer1
            case 'Yes'
                xReq = str2num(cell2mat(inputdlg(promptX,'Please Input x-Value',↙
dialogBoxDimensions,defaultInput)));
                yReq = str2num(cell2mat(inputdlg(promptY,'Please Input y-Value',↙
dialogBoxDimensions,defaultInput)));
                fprintf("\nRequested node Temperature is: ");
                fprintf('%4.4f', T_new(yReq,xReq));
                fprintf('\n');
                nodeReq = true;
            case 'No'
                nodeReq = false;
        end
    end
    answer2 = questdlg('Would you like to try another run using different values for k,↙
h_0, or Convergence Criteria?','Run Again?','Yes','No','No');
    switch answer2
        case 'Yes'
            run = true;
        case 'No'
            run = false;
    end
end %while run
```