```
% Name: Kathryn Atherton

% Food Material: Pumpkin Pie Filling

%--------------------------------------------------------------------------
%USE THIS TEMPLATE FOR EACH SCRIPT/FUNCTION THAT YOU WRITE
%For each block of code (5-10 lines) in the model provide the
 following:
%       HOW: How does this block of code work?
%       WHAT: What does the block of code represent?
%       WHY: Why is this block of code included in the model?
%For more details, an example of good commenting and bad commenting
 has
%been uploaded to Blackboard.
%
 --------------------------------------------------------------------------

% These lines clear the command window and the workspace and close any
% previously made images to ensure the code runs smoothly.

clc;
clear;
close all;

%---------------Thermal/Physical Properties of the System
 -----------------
%Here place any/all thermal, physical, or kinetic properties of the
 system.
%For each block make sure to answer the HOW/WHAT/WHY questions above.

% Define how heat moves through the material so that the equations
 later in
% the code can be used. Most of the variables came from the memo/
assignment
% document but the food composition data came from the USDA database.
% User-defined functions (described below) converted Imperial units to
 SI
% units for easier calculations.

% Filling Properties
fill_temp = 180; % deg F, given in blueprint

fill_temp = f_to_c(fill_temp); % deg C

% Full Steam Immersion Retort Properties
steam_temp = 250; % deg F, given in blueprint

steam_temp = f_to_c(steam_temp); % deg C

% Cooling Water Immersion Properties
exit_temp = 100; % deg F, given in blueprint
cool_water_temp_min = 50; % deg F, given in blueprint
```

```
cool_water_temp_max = 60; % deg F, given in blueprint

exit_temp = f_to_c(exit_temp); % deg C
cool_water_temp_min = f_to_c(cool_water_temp_min); % deg C
cool_water_temp_max = f_to_c(cool_water_temp_max); % deg C
cool_water_temp = mean([cool_water_temp_min, cool_water_temp_max]); %
 deg C

% Production Line Time Breakdown Properties
moisture_content_min = 45; % percent, given in properties of food
 materials and microbes
moisture_content_max = 50; % percent, given in properties of food
 materials and microbes
can_diameter = 4.25; % inches, from https://
www.everythingkitchens.com/wisconsin-aluminum-1502-senior-flywheel-
can-sealer.html

can_diameter = in_to_cm(can_diameter); % cm

% Kinetic Properties of Food Components - selected "worst case
 scenario"
% values
t_ea = 27; % kcal/mol, given in properties of food materials and
 microbes
t_d250 = 246.9; % min, given in properties of food materials and
 microbes
aa_ea = 24; % kcal/mol, given in properties of food materials and
 microbes
aa_d250 = 1.12; % days, given in properties of food materials and
 microbes
c_ea = 24; % kcal/mol, given in properties of food materials and
 microbes
c_d250 = 1.94; % days, given in properties of food materials and
 microbes

t_ea = kcal_to_j(t_ea); % J/mol
aa_ea = kcal_to_j(aa_ea); % J/mol
c_ea = kcal_to_j(c_ea); % J/mol
aa_d250 = days_to_min(aa_d250); % min
c_d250 = days_to_min(c_d250); % min

% Kinetic Properties of Microorganisms
cb_ea = 64; % kcal/mol, given in properties of food materials and
 microbes
cb_d250 = 0.2; % min, given in properties of food materials and
 microbes
cp_ea = 72; % kcal/mol, given in properties of food materials and
 microbes
cp_d250 = 0.04; % min, given in properties of food materials and
 microbes
bc_ea = 65; % kcal/mol, given in properties of food materials and
 microbes
bc_d250 = 0.008; % min, given in properties of food materials and
 microbes
```

```
cb_ea = kcal_to_j(cb_ea); % J/mol
cp_ea = kcal_to_j(cp_ea); % J/mol
bc_ea = kcal_to_j(bc_ea); % J/mol


% Nutrient Content of Food Material, Data from USDA Food Composition
% Database (ndb.nal.usda.gov/nd), using data from Stonewall Kitchen
 Ltd.
% Pumpkin Pie Filling
protein = 0.00; % g protein / 100 g
fat = 0.00; % g fat / 100 g
carbohydrate = 51.11 / 100; % g carbs / gram
fiber = 1.1 / 100; % g fiber / gram
moisture_content = 1 - protein - fat - carbohydrate - fiber; % g
 water / gram
ash = 0.00 / 100; % g ash / gram
if moisture_content < (moisture_content_min / 100)
    moisture_content = moisture_content_min / 100;
elseif moisture_content > (moisture_content_max / 100)
    moisture_content = (moisture_content_max / 100);
    ash = 1 - protein - fat - carbohydrate - fiber - moisture_content;
end

%--------------------Numerical/Analytical Calculations
 ---------------------
%Here place any calculations, mathematical structures, looping
 structures,
%etc that are necessary for the modeling activity. For block of code
 or
%structure make sure to answer the HOW/WHAT/WHY questions above.

% This works by assuming a small delta t value (1 second) and an M
 value of
% 4 and then calculating delta x from these values before dividing the
% radius of the can by the number of layers. This must be done to set
 the
% number of iterations the program must run through for each time
 iteration.

delt = 1; % s
M = 4;
t_food = fill_temp; % deg C
alpha = choi_okos_alpha(t_food, protein, fat, carbohydrate, fiber,
 ash, moisture_content); % m^2/s
delx = sqrt(M * delt * alpha); % m
delx_cm = delx * 100;
layers = round((can_diameter / 2) / delx_cm) + 1;
n = layers; % layers, currently at surface

% This uses the Choi-Okos equation for the given initial temperature
 in
% order to start the process of calculating the heat transfer
 behavior.
```

```
k = choi_okos_k(t_food, protein, fat, carbohydrate, fiber, ash,
 moisture_content); % W/m.C
rho = choi_okos_density(t_food, protein, fat, carbohydrate, fiber,
 ash, moisture_content); % kg/m^3, from https://www.aqua-calc.com/
calculate/food-volume-to-weight
cp = choi_okos_cp(t_food, protein, fat, carbohydrate, fiber, ash,
 moisture_content); % J/kg.C, from https://www.engineeringtoolbox.com/
specific-heat-capacity-food-d_295.html
h_steam = 1/(1/6000 + 1/166); % W/m^2.C, from ASHRAE Handbook Chapter
 9: Thermal Properties of Food
h_water = 1/(1/300 + 1/166);
t = 0; % s
T = ones(1, layers) * t_food; % deg C
T_new = T; % deg C

% The arrays are a helpful way to store information to be used later
 for
% each iteration when an unknown number of iterations will occur as
 before
% the loop it is unknown how much time it will take to reach the
 minimum
% desired sterilization level.
heating_center = [];
k_cb = thermal_reduction(cb_d250, cb_ea, T(1));
k_cp = thermal_reduction(cp_d250, cp_ea, T(1));
k_bc = thermal_reduction(bc_d250, bc_ea, T(1));
k_t = thermal_reduction(t_d250, t_ea, T(1));
k_aa = thermal_reduction(aa_d250, aa_ea, T(1));
k_c = thermal_reduction(c_d250, c_ea, T(1));
t_avg = [];
aa_avg = [];
c_avg = [];
k_t_avg = [];
k_aa_avg = [];
k_c_avg = [];
lnreduction_cb = 0;
lnreduction_cp = 0;
lnreduction_bc = 0;
lnreduction_t = 0;
lnreduction_aa = 0;
lnreduction_c = 0;


% This loop first steps the time and then goes through each node from
 the
% surface to the center and calculates the temperature at that node by
% first calculating all the parameters needed via the Choi-Okos using
 the
% last temperature at that node. Then a new temperature is calculated
 with
% the equations defined in Geankoplis. The desired temperatures are
 stored
% in an array. The k value for each microorganism and nutrient is then
% calculated. At the end of the loop, the desired data points are
 saved,
% the average nutrient content is found for that time point and saved,
 and
% the total reduction in each microorganism and nutrient is calculated
 with
% the trapezoidal method. Then, the parameters such as time and node
 number
% are reset for the next iteration. This continues until the targeted
% microorganism, C. Botulinum, reaches the minimum required reduction
 to
% meet sterilization requirements.
while lnreduction_cb < 12
    t = t + delt;
   while n > 0
       if n == 1
           alpha = choi_okos_alpha(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           M = m_calc(delt, alpha, delx);
           T_new(1) = T_center(M, T(2), T(1)); % deg C
           heating_center = [heating_center, T_new(1)];
           k_cb = [k_cb, thermal_reduction(cb_d250, cb_ea, T_new(1))];
           k_cp = [k_cp, thermal_reduction(cp_d250, cp_ea, T_new(1))];
           k_bc = [k_bc, thermal_reduction(bc_d250, bc_ea, T_new(1))];
           k_t = [k_t, thermal_reduction(t_d250, t_ea, T_new(1))];
           k_aa = [k_aa, thermal_reduction(aa_d250, aa_ea, T_new(1))];
           k_c = [k_c, thermal_reduction(c_d250, c_ea, T_new(1))];
           k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
           k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
           k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
       elseif n == layers
           k = choi_okos_k(T(n), protein, fat, carbohydrate, fiber,
 ash, moisture_content);
           rho = choi_okos_density(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           cp = choi_okos_cp(T(n), protein, fat, carbohydrate, fiber,
 ash, moisture_content);
           T_new(n) = T_outside(T(n), h_steam, n, delx, steam_temp, k,
 T(n-1), rho, cp, delt) + T(n); % deg C
           k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
           k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
           k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
       else
           alpha = choi_okos_alpha(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           M = m_calc(delt, alpha, delx);
           T_new(n) = T_other(n, M, T(n+1), T(n), T(n-1)); % deg C
```

```
          k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
          k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
          k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
      end
      n = n - 1;
  end
  T = T_new;
  n = layers;
  t_avg = [t_avg, mean(k_t_avg)];
  aa_avg = [aa_avg, mean(k_aa_avg)];
  c_avg = [c_avg, mean(k_c_avg)];
  k_t_avg = [];
  k_aa_avg = [];
  k_c_avg = [];
  lnreduction_cb = trapz(k_cb); % log reduction
  lnreduction_cp = trapz(k_cp); % log reduction
  lnreduction_bc = trapz(k_bc); % log reduction
  lnreduction_t = trapz(k_t); % log reduction
  lnreduction_aa = trapz(k_aa); % log reduction
  lnreduction_c = trapz(k_c); % log reduction
end

% These inform the user that the code is progressing as it should and
% outputs important data points such as the optimum temperature and
% sterilization time.
t_hours = t / 3600;
fprintf('The time to reach the minimum sterilization requirement is
 %.2f hours.\n', t_hours);
fprintf('The maximum temperature of the center of the can is %.1f
 degrees C.\n', T(1));

% This loop first steps the time and then goes through each node from
 the
% surface to the center and calculates the temperature at that node by
% first calculating all the parameters needed via the Choi-Okos using
 the
% last temperature at that node. Then a new temperature is calculated
 with
% the equations defined in Geankoplis. The desired temperatures are
 stored
% in an array. The k value for each microorganism and nutrient is then
% calculated. At the end of the loop, the desired data points are
 saved,
% the average nutrient content is found for that time point and saved,
 and
% the total reduction in each microorganism and nutrient is calculated
 with
% the trapezoidal method. The average temperature of the can is
 calculated.
% Then, the parameters such as time and node number are reset for the
 next
```

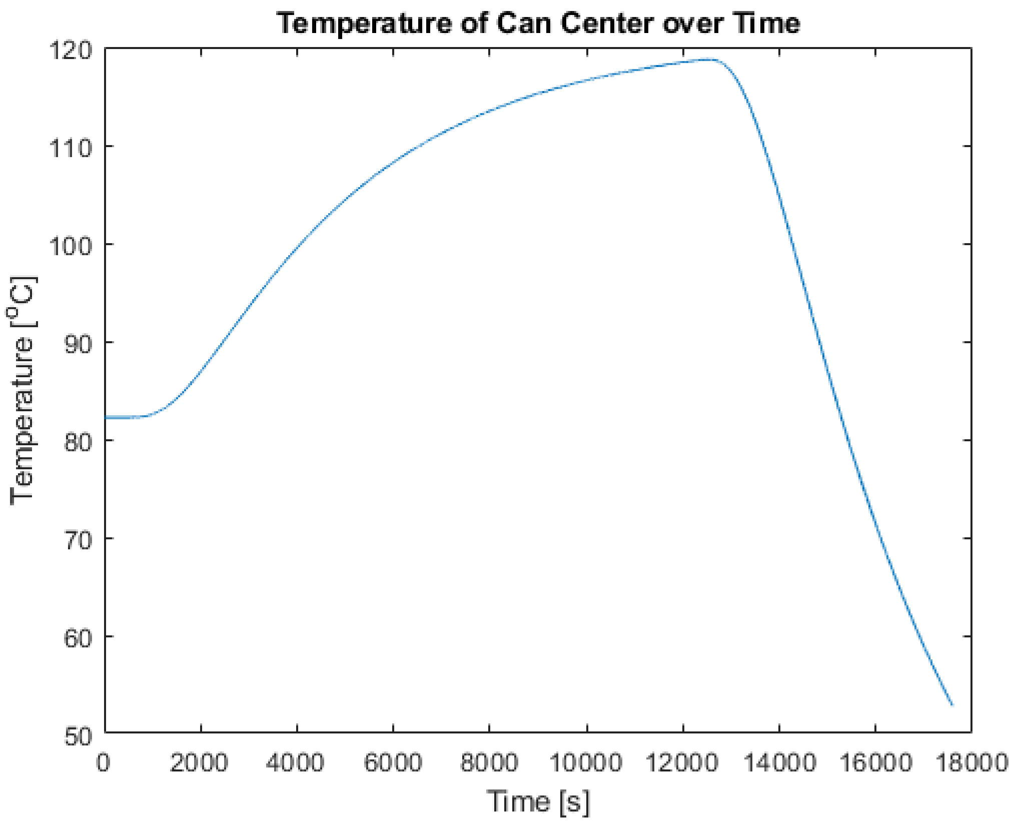
```
% iteration. This continues until the average temperature of the can
% reaches the desired temperature.
while mean(T) > exit_temp
    t = t + delt;
   while n > 0
       if n == 1
           alpha = choi_okos_alpha(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           M = m_calc(delt, alpha, delx);
           T_new(1) = T_center(M, T(2), T(1)); % deg C
           heating_center = [heating_center, T_new(1)];
           k_cb = [k_cb, thermal_reduction(cb_d250, cb_ea, T_new(1))];
           k_cp = [k_cp, thermal_reduction(cp_d250, cp_ea, T_new(1))];
           k_bc = [k_bc, thermal_reduction(bc_d250, bc_ea, T_new(1))];
           k_t = [k_t, thermal_reduction(t_d250, t_ea, T_new(1))];
           k_aa = [k_aa, thermal_reduction(aa_d250, aa_ea, T_new(1))];
           k_c = [k_c, thermal_reduction(c_d250, c_ea, T_new(1))];
           k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
           k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
           k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
       elseif n == layers
           k = choi_okos_k(T(n), protein, fat, carbohydrate, fiber,
 ash, moisture_content);
           rho = choi_okos_density(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           cp = choi_okos_cp(T(n), protein, fat, carbohydrate, fiber,
 ash, moisture_content);
           T_new(n) = T_outside(T(n), h_water, n, delx,
 cool_water_temp, k, T(n-1), rho, cp, delt) + T(n); % deg C
           k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
           k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
           k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
       else
           alpha = choi_okos_alpha(T(n), protein, fat, carbohydrate,
 fiber, ash, moisture_content);
           M = m_calc(delt, alpha, delx);
           T_new(n) = T_other(n, M, T(n+1), T(n), T(n-1)); % deg C
           k_t_avg = [k_t_avg, thermal_reduction(t_d250, t_ea,
 T_new(n))];
           k_aa_avg = [k_aa, thermal_reduction(aa_d250, aa_ea,
 T_new(n))];
           k_c_avg = [k_c_avg, thermal_reduction(c_d250, c_ea,
 T_new(n))];
       end
       n = n - 1;
   end
   t_avg = [t_avg, mean(k_t_avg)];
   aa_avg = [aa_avg, mean(k_aa_avg)];
   c_avg = [c_avg, mean(k_c_avg)];
   k_t_avg = [];
   k_aa_avg = [];
   k_c_avg = [];
   T = T_new;
   n = layers;
   lnreduction_cb = trapz(k_cb); % log reduction
   lnreduction_cp = trapz(k_cp); % log reduction
   lnreduction_bc = trapz(k_bc); % log reduction
   lnreduction_t = trapz(k_t); % log reduction
   lnreduction_aa = trapz(k_aa); % log reduction
   lnreduction_c = trapz(k_c); % log reduction
end

% These inform the user that the code is progressing as it should and
% outputs important data points such as the reduction of the
 microorganisms
% and nutrients and the total sterilization and cooling time.
t_hours = t / 3600; % hours
fprintf('The total sterilization and cooling process takes %.2f hours.
\n', t_hours);
fprintf('The total reduction in C. Botulinum is %.2f.\n',
 lnreduction_cb);
fprintf('The total reduction in C. Perfringens is %.2f.\n',
 lnreduction_cp);
fprintf('The total reduction in B. Cereus is %.2f.\n', lnreduction_bc)
fprintf('The total reduction in Thiamine is %.2f.\n', lnreduction_t)
fprintf('The total reduction in Ascorbic Acid is %.2f.\n',
 lnreduction_aa)
fprintf('The total reduction in Cobalamin is %.2f.\n', lnreduction_c)

% This section is performed to be able to plot the average nutrient
% activity of the can.
t_logred_avg = t_avg .* t;
aa_logred_avg = aa_avg .* t;
c_logred_avg = c_avg .* t;

%-------------------------- Graphical/Numerical Output
 --------------------
%Any output parameters should go here such as display of graphs,
 matrices,
%values, variables, or tables. For each block of code make sure to
 answer the
%HOW/WHAT/WHY questions above.

% The plots are a visual way to see what occurs during the
 sterilization
% and cooling processes with regards to the temperature and
 microorganism
% and nutrient content.
figure
plot(heating_center)
xlabel('Time [s]')
ylabel('Temperature [^oC]')
```

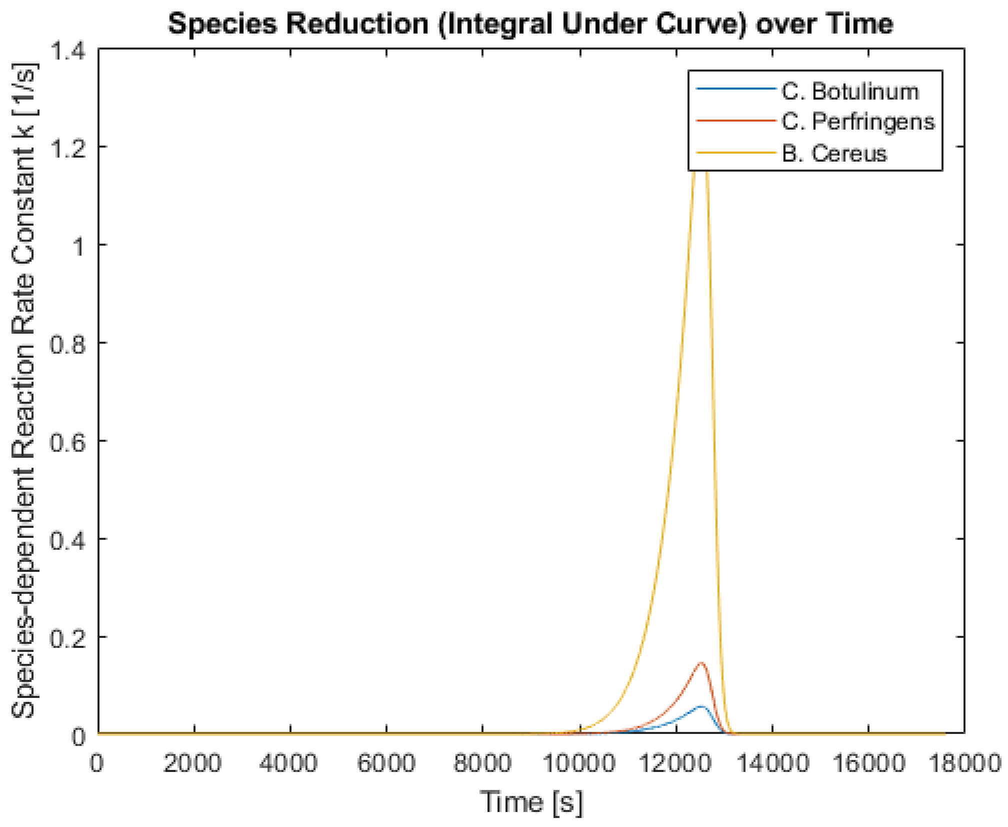
```
title('Temperature of Can Center over Time')
hold off

figure
plot(k_cb);
hold on
plot(k_cp);
plot(k_bc);
xlabel('Time [s]')
ylabel('Species-dependent Reaction Rate Constant k [1/s]')
title('Species Reduction (Integral Under Curve) over Time')
legend('C. Botulinum', 'C. Perfringens', 'B. Cereus')
hold off

figure
plot(t_logred_avg);
hold on
plot(aa_logred_avg);
plot(c_logred_avg);
xlabel('Time [s]')
ylabel('Average Log Reduction in Nutrient Content Throughout Can')
title('Average Log Reduction in Nutrient Content In Can Over Time')
legend('Thiamine', 'Ascorbic Acid', 'Cobalamin')
hold off
%-------------------------- User-Defined Functions
 ------------------------
function [t_c] = f_to_c(t_f)
    % HOW: This funciton uses the standard temperature conversion
 formulas.
    % WHAT: This function converts temperatures in degrees Fahrenheit
 to
    %        degrees Celsius.
    % WHY: The formulas to evaluate the temperature profile require
    %       temperature to be in degrees C.

    t_c = 5/9 * (t_f - 32); % convert to degrees C
end

function [cm] = in_to_cm(in)
    % HOW: This function uses standard length conversions.
    % WHAT: This function converts inches to centimeters.
    % WHY: This is necessary to put all units into SI units for easier
    %       usage in the alorithm.

    cm = in * 2.54; % centimeters
end

function [j] = kcal_to_j(kcal)
    % HOW: This function uses standard energy unit conversions.
    % WHAT: This function converts kilocalories to Joules.
    % WHY: This is necessary to put all units into SI units for easier
    %       usage in the alorithm.

    j = kcal * 4184; % Joules
```

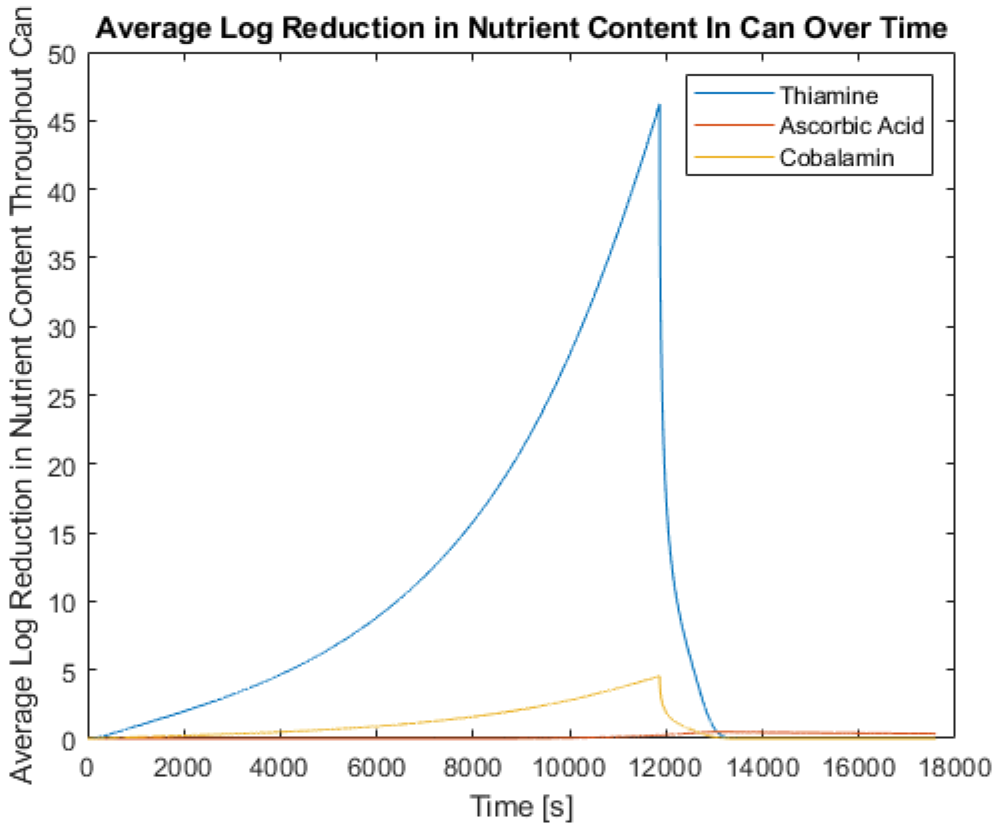
```
end

function [min] = days_to_min(days)
    % HOW: This function uses standard time conversions.
    % WHAT: This function converts days to minutes.
    % WHY: This is necessary to keep all D250 values the same time
 units
    %      for easier comparison.

    min = days * 24 * 60; % minutes
end

function [density] = choi_okos_density(t_food, protein, fat,
 carbohydrate, ...
    fiber, ash, moisture_content)
    % HOW: This function uses the Choi-Okos equation.
    % WHAT: This function calculates the density of the food using the
    %       relative composition of the food.
    % WHY: This funciton is required in order to calculate heat
 transfer.

    p_density = 1.3299e3 - 5.1840e-1 * t_food; % kg/m^3
    fat_density = 9.2559e2 - 4.1757e-1 * t_food; % kg/m^3
    c_density = 1.5991e3 - 3.1046e-1 * t_food; % kg/m^3
    fiber_density = 1.3115e3 - 3.6589e-1 * t_food; % kg/m^3
    a_density = 2.42338e3 - 2.8063e-1 * t_food; % kg/m^3
    w_density = 9.9718e2 + 3.1439e-3 * t_food - 3.7574e-3 * t_food ^
 2; % kg/m^3
    density = (protein * p_density) + (fat * fat_density) + ...
        (carbohydrate * c_density) + (fiber * fiber_density) + ...
        (ash * a_density) + (moisture_content * w_density);
end

function [cp] = choi_okos_cp(t_food, protein, fat, carbohydrate,
 fiber, ...
    ash, moisture_content)
    % HOW: This funciton uses the Choi-Okos equation.
    % WHAT: This function finds the heat capacity of the food using
 the
    %       relative composition of the food.
    % WHY: The heat capacity is required to find the heat transfer
 within
    %      the food.

    p_cp = 2.0082e3 + 1.2089 * t_food - 1.3129e-3 * t_food ^ 2; % J/
kg.C
    fat_cp = 1.9842e3 + 1.4733 * t_food - 4.8008e-3 * t_food ^ 2; % J/
kg.C
    c_cp = 1.5488e3 + 1.9625 * t_food - 5.9399e-3 * t_food ^ 2; % J/
kg.C
    fiber_cp = 1.8459e3 + 1.8306 * t_food - 4.6509e-3 * t_food ^ 2; %
 J/kg.C
    a_cp = 1.0926e3 + 1.8896 * t_food - 3.6817e-3 * t_food ^ 2; % J/
kg.C
```

```
    w_cp = 4.1762e3 - 9.0864e-2 * t_food + 5.4731e-3 * t_food ^ 2; %
 J/kg.C
    cp = (protein * p_cp) + (fat * fat_cp) + (carbohydrate * c_cp)
 + ...
        (fiber * fiber_cp) + (ash * a_cp) + ...
        (moisture_content * w_cp); % J/kg.C
end

function [k] = choi_okos_k(t_food, protein, fat, carbohydrate, fiber,
 ash,...
    moisture_content)
    % HOW: This funciton uses the Choi-Okos equation.
    % WHAT: This function finds the thermal conductivity of the food
 using
    %       the relative composition of the food.
    % WHY: The heat capacity is required to find the thermal
 conductivity
    %       within the food.

    p_k = 1.7881e-1 + 1.1958e-3 * t_food - 2.7178e-6 * t_food ^ 2; %
 W/m.C
    fat_k = 1.8071e-1 - 2.7604e-3 * t_food - 1.7749e-7 * t_food ^ 2; %
 W/m.C
    c_k = 2.0141e-1 + 1.3874e-3 * t_food - 4.3312e-6 * t_food ^ 2; %
 W/m.C
    fiber_k = 1.8331e-1 + 1.2497e-3 * t_food - 3.1683e-6 * t_food ^
 2; % W/m.C
    a_k = 3.2962e-1 + 1.4011e-3 * t_food - 2.9069e-6 * t_food ^ 2; %
 W/m.C
    w_k = 5.7109e-1 + 1.7625e-3 * t_food - 6.7036e-6 * t_food ^ 2; %
 W/m.C
    k = (protein * p_k) + (fat * fat_k) + (carbohydrate * c_k) + ...
        (fiber * fiber_k) + (ash * a_k) + (moisture_content * w_k); %
 W/m.C
end

function [alpha] = choi_okos_alpha(t_food, protein, fat,
 carbohydrate, ...
    fiber, ash, moisture_content)
    % HOW: This funciton uses the Choi-Okos equation.
    % WHAT: This function finds the thermal diffusivity of the food
 using
    %       the relative composition of the food.
    % WHY: The heat capacity is required to find the thermal
 diffusivity
    %      within the food.

    p_a = 6.8714e-8 + 4.7578e-10 * t_food - 1.4646e-12 * t_food ^ 2;
    fat_a = 9.8777e-8 - 1.2569e-11 * t_food - 3.8286e-14 * t_food ^ 2;
    c_a = 8.8042e-8 + 5.3052e-10 * t_food - 2.3218e-12 * t_food ^ 2;
    fiber_a = 7.3976e-8 + 5.1902e-10 * t_food - 2.2202e-12 * t_food ^
 2;
    a_a = 1.2461e-7 + 3.7321e-10 * t_food - 1.2244e-12 * t_food ^ 2;
    w_a = 1.3168e-7 + 6.2477e-10 * t_food - 2.4022e-12 * t_food ^ 2;
    alpha = (protein * p_a) + (fat * fat_a) + (carbohydrate * c_a)
 + ...
        (fiber * fiber_a) + (ash * a_a) + (moisture_content * w_a); %
 m^2/s
end

function [M] = m_calc(delt, alpha, delx)
    % HOW: Uses equations from Geankoplis
    % WHAT: Calculates the M value in the Geankoplis equations
    % WHY: Helps to determine heat transfer behavior.

    M = (delx ^ 2) /  (alpha * delt);
    if M < 4
        M = 4;
    end
end

function [T] = T_center(M, T1, T0)
    % HOW: Uses equations from Geankoplis at the center node
    % WHAT: Calculates the temperature of the center node
    % WHY: To calculate how the center of the can heats

    T = (4 / M) * T1 + ((M - 4) / M) * T0; % deg C
end

function [del_t] = T_outside(T, h, r, delx, steam_temp, k, T1, rho,
 cp, delt)
    % HOW: Uses equations from Geankoplis
    % WHAT: Calculates the temperature of the surface node
    % WHY: To calculate how the surface of the can heats

    del_t = (delt * (((h * 2 * pi * r * delx * (steam_temp - T)) - (k
 * 2 * (r - (1/2)) * (T - T1)))/(rho * cp * (((r * delx) ^ 2) - (((r -
 (1/2)) * delx) ^ 2)))));
end

function [T] = T_other(r, M, T2, T1, T)
   % HOW: Uses equations from Geankoplis
   % WHAT: Calculates the temperature of the nodes between the surface
 and
   % the center
   % WHY: To calculate how the food heats

    a = ((2 * r) + 1) / (2 * r);
    b = M - 2;
    c = ((2 * r) - 1) / (2 * r);
    T = (1 / M) * ((a * T2) + (b * T1) + (c * T)); % deg C
end

function [k121] = thermal_reduction(d250, ea, temp)
    % HOW: Uses formulas from Geankoplis
    % WHAT: Calculates the reduction of a microorganism or nutrient
    % WHY: To determine microorganism and nutrient activity.
    k250 = 2.303 / d250; % min ^ -1
    t_ref = f_to_c(250); % deg C
    R_const = 8.314; % J/mol.K
    k0 = k250 / exp((-1 * ea) / (R_const * t_ref)); % min ^ -1
    k121 = k0 * exp((-1 * ea) / (R_const * temp)); % min ^ -1
end
```

```
The time to reach the minimum sterilization requirement is 3.30 hours.
The maximum temperature of the center of the can is 118.3 degrees C.
The total sterilization and cooling process takes 4.89 hours.
The total reduction in C. Botulinum is 53.03.
The total reduction in C. Perfringens is 125.83.
The total reduction in B. Cereus is 1207.12.
The total reduction in Thiamine is 1.67.
The total reduction in Ascorbic Acid is 0.35.
The total reduction in Cobalamin is 0.20.
```




Species Reduction (Integral Under Curve) over Time
Species-dependent Reaction Rate Constant k [1/s]
Time [s]
C. Botulinum
C. Perfringens
B. Cereus


Average Log Reduction in Nutrient Content In Can Over Time
Average Log Reduction in Nutrient Content Throughout Can
Time [s]
Thiamine
Ascorbic Acid
Cobalamin