Final Project: How Hard (and Frequently) Do You Have To Slap A Chicken To Cook It?

Samuel Goodwin
Department of Physics, Astronomy, and Applied Physics,
Rensselaer Polytechnic Institute
goodws@rpi.edu

April 28, 2019

Abstract

Code written to solve the three-dimensional heat equation was used to determine how the temperature distribution of a chicken changes with time as it is slapped on random parts of its surface with varying frequencies and hardness. Chickens slapped with low slap frequencies and low slap hardness cooked the most evenly, but chickens slapped with slap frequencies and hardness that were too low were not able to cook. There are many combinations of slap frequencies and hardness that cook a chicken with minimum or close to minimum standard deviation in the chicken's temperature distribution, but in particular low slap hardness and low slap frequency cooked chickens the most evenly.

1 Problem Overview

A popular meme that circulated the Internet asked the question "How hard do you have to slap a chicken to cook it?". In this project, I attempt to solve this question and more. Slapping a chicken only once is not an effective way to cook it because this would cause most of the chicken to be burnt at temperatures on the order of 10^8 $°C$. In order to more evenly cook the chicken, it should be slapped repeatedly on random sections of its surface. A question that arises out of this is "How frequently should the chicken be slapped?", so besides cooking chickens with various hardnesses (different temperature changes from slapping), I also look at different frequencies for slapping the chicken. When comparing all the cooked chickens, each cooked with a specified slap frequency and slap temperature, I will look at what conditions create the most-evenly cooked chicken (standard deviation of temperatures in the chicken) and what conditions cook a chicken the fastest.

In order to simulate cooking a chicken by slapping it, I will create a chicken object and solve the three-dimensional heat equation over it with the temperature of the room the chicken is in, the initial temperature of the chicken, and the slap temperature acting as boundary/initial conditions. After cooking all the chickens, I will make a plot comparing slap temperature, slap frequency, and standard deviation of temperature in the chicken, and another plot comparing slap temperature, slap frequency, and time it took to cook the chicken. I will also make heatmaps showing the temperature distribution over the chicken that cooks the fastest and the chicken that is the most-evenly cooked. There will be 7 two-dimensional heatmaps for each of these chickens: 6 of them are for each of the faces of the chicken (chickens are approximated to be rectangular prisms) and the 7th face is a cross-section of the chicken's interior.

2 Algorithmic Consideration

Several functions are defined in this project. I wrote a function called init_chicken(frame) where given a three-dimensional array meant to represent the chicken (frame), all values in the array are set to what I specify the initial temperature of the chicken to be. The function slap(frame) takes a chicken (frame) and increases the temperature of a random region on its surface by the specified slap temperature. In order to do this, the function generates a random integer 0 through 5 and this number determines which side of the chicken is slapped. Once a side is randomly picked, two random numbers are generated to pick the corner of the region that will be slapped (the region is always a square with fixed side-length for all chickens). Once the corner is randomly chosen, the rest of the points that will be slapped are found using the fixed side-length of the slap region. The function is_cooked(frame) takes a chicken (frame) and checks each point in the chicken to see if it has a temperature that is greater than or equal to the minimum temperature it needs to be to be considered "cooked". This function is called after every iteration in the loop that solves the heat equation and if the chicken is determined to be cooked, then the loop terminates and the chicken is considered "cooked". The function std_dev(frame) takes a chicken (frame) and finds the standard deviation of the set of temperatures throughout the chicken. I also include a cooling term in the heat equation such that points on the surface of the chicken cool off due to the air in the room around it. The functions in_corner(i,j,k), on_edge(i,j,k), and on_surface(i,j,k) check if the given point (i,j,k) is on a corner, edge, or the surface of the chicken respectively. These functions are needed to see if the cooling law needs to be applied to the given point and since corners, edges, and the other surface points interact with a different number of air points, the cooling terms are different for each of the three types of surface points.

In order to use Numba to make the simulation run faster, I need to put the loop that updates the chicken at each time step in its own function, and I will call this function sim_step(old_frame,new_frame) that takes in a chicken at time t_i (old_frame), solves for the temperature distribution of the chicken at time t_{i+1}, and puts the solution in new_frame. Analytically, the three-dimensional heat equation is:

$$\frac{\partial T}{\partial t} = \frac{K}{C\rho}\nabla^2 T - \frac{hA}{mC}(T - T_a) \tag{1}$$

Where T is the temperature at some point, K is the thermal conductivity of the chicken, C is the specific-heat capacity of the chicken, ρ is the density of the chicken (which I will assume to have uniform density), m is the mass of chicken exposed to the air in the room (causing it to cool down), h is the heat transfer coefficient of chicken, A is the surface area of chicken exposed to the air, and T_a is the ambient temperature (or more specifically in this case, the temperature of the room). In order to solve this numerically, the time and spatial steps need to be discretized.

$$\begin{split}\frac{T_{i,j,k,t+1} - T_{i,j,k,t}}{\Delta t} = \frac{K}{C\rho}(\frac{T_{i+1,j,k,t} - 2T_{i,j,k,t} + T_{i-1,j,k,t}}{\Delta x^2} + \frac{T_{i,j+1,k,t} - 2T_{i,j,k,t} + T_{i,j-1,k,t}}{\Delta y^2} \\ + \frac{T_{i,j,k+1,t} - 2T_{i,j,k,t} + T_{i,j,k-1,t}}{\Delta z^2}) - \frac{hA}{mC}(T_{i,j,k,t} - T_{room})\end{split} \tag{2}$$

Where $T_{i,j,k,t}$ is the temperature of the chicken at point (i, j, k) at time t. I will simplify this equation by combining multiple variables into single variables and then solve for the temperature at some point at a future time. First, a simplification:

$$\eta_i = \frac{K\Delta t}{C\rho\Delta x_i^2} \tag{3}$$

With this simplification in mind, I will solve for $T_{i,j,k,t+1}$:

$$\begin{split}T_{i,j,k,t+1} = \eta_1(T_{i+1,j,k,t} - 2T_{i,j,k,t} + T_{i-1,j,k,t}) + \eta_2(T_{i,j+1,k,t} - 2T_{i,j,k,t} + T_{i,j-1,k,t}) \\ + \eta_3(T_{i,j,k+1,t} - 2T_{i,j,k,t} + T_{i,j,k-1,t}) - \frac{hA}{mC}(T_{i,j,k,t} - T_{room}) + T_{i,j,k,t}\end{split} \tag{4}$$

This equation shows that given the chicken's temperature distribution at time t_i, I can solve for the temperature at any point in the chicken at time t_{i+1}. Also, it is important to note that the cooling term is only applied to points on the surface of the chicken. To update all points in the chicken at time t_{i+1}, an algorithm can iterate over all points in the chicken and use equation (4) to solve for the new temperature.

Solving the heat equation for all points on a future chicken is just a single step in the simulation, but the whole simulation needs more components to work properly. The algorithm is provided an array of slap temperatures and an array of iteration periods. These iterations periods indicate how often the chicken should be slapped. For example, if the iteration period is n, then the chicken is slapped once every n iterations.

A double for loop runs the simulation for each combination of slap temperatures and iteration periods. Initially, the chicken is given an initial temperature equal to the room temperature and then it is placed in a while loop whose iterations update the temperatures. Each iteration represents a time step and the temperature distribution of the chicken is updated every iteration to resemble its change in temperature over that time step. The while loop handling the temperature updates terminates under one of two conditions. The first is if every point in the chicken has reached some temperature defined to be "cooked". The other condition is if the chicken is incapable of cooking. The simulation checks if the minimum temperature in the chicken is still increasing over time and if it is not, then the loop is terminated, the program prints an error message, and the program moves on to start cooking the next chicken. If the chicken cannot be cooked, it is because it is not being slapped hard enough and/or it is not slapped frequently enough to compensate for the surface cooling off.

Once a chicken is cooked, the 3D array of temperature values representing the chicken is stored in an array of cooked chickens and information about the chicken is stored in a separate results array. The results array contains, for each chicken, the number of iterations it took to cook the chicken, the slap temperature, the slap iteration periods, and the standard deviation of the temperature distribution of the chicken after it is cooked. With this information, I am able to compare different slap temperatures and frequencies and how it affected how the chicken was cooked.

The only issues are I want to know the slap frequencies in slaps per second and the physical time it took to cook the chicken, rather than the number of iterations needed. Each iteration represents a time step Δt so in order to find the physical time it took to cook the chicken, all that needs to be done is to take the number of iterations and multiply by my chosen value for Δt (see Table 1).

$$t_{tot} = n * \Delta t \tag{5}$$

Where t_{tot} is the total time it takes to cook the chicken (in seconds), n is the number of iterations needed to cook the chicken, and Δt is the time step for each iteration. The slap frequency can be found in slaps per second with the following formula:

$$f = \frac{1}{p * \Delta t} \tag{6}$$

Where f is the slap frequency (in slaps per second), p is the slap iteration period, and Δt is the time step for each iteration.

3 Implementation

This simulation was implemented using Python 3.5. All plots were made using python.matplotlib except for the scatter plots, which were made using plotly.plotly. Numba was used in order to speed up each time step in the simulation.

The simulation was run with the parameters in Table 1:

Parameter	**Value(s)**
Thermal Conductivity	0.476 (W/(m*$^\circ C$)) [1]
Specific-Heat Capacity	1840 (J/(kg*$^\circ C$)) [2]
Density	1150 (kg/m^3) [3]
Heat Transfer Coefficient	100 (W/(m$^2 *^\circ C$)) [4]
Dimensions of Chicken	1 x 1 x 2 (m^3)
Dimensions of Slap	0.3 x 0.3 (m^2)
Temperature of Room	$20^\circ C$
Initial Temperature of Chicken	$20^\circ C$
Cooked Temperature	$72^\circ C$
Slap Temperatures	[30,40,50,75,100,150,200,250,300,400,500,600,750,800,1000,1500,2000,2500,3000] ($^\circ C$)
Slap Iteration Periods	[2,3,5,10,20,25,30,40,50,55,60,65,70,75] (iterations/slap)
Δx	0.1 (m)
Δy	0.1 (m)
Δz	0.1 (m)
Δt	0.1 (s)

Table 1: A table of parameters chosen to run the chicken-cooking simulation.

Simulations cannot usually perfectly model the conditions that would exist in the real world, so there are some approximations made in this simulation. It assumes the chicken has uniform density and it also assumes the value used for thermal conductivity is correct. The only sources with information on the thermal conductivity of chicken only state what it is at temperatures at and below $20^\circ C$. Because of this, the simulation assumes the chicken's known thermal conductivity at $20^\circ C$ remains the same even for higher temperatures. This is most likely very inaccurate, but it is a necessary approximation. It also assumes the heat transfer accurate. There are limited sources on the heat transfer coefficient of chicken so the heat transfer coefficient used in this simulation is an approximation based on the literature estimate of the range of possible coefficients, which is 67 to 123 (W/(m$^2 *^\circ C$)).

This simulation also cooks chickens that are larger than real chickens. This was the smallest I was willing to make the volume of the chicken because the three-dimensional grid representing the chicken needs to have a significant number of points in order to obtain meaningful solutions (a chicken that is approximated by a 2 x 2 x 2 grid would not at all yield accurate results). A fix to this would be to decrease the spatial step values so that the distances in-between points on the chicken would decrease, but all attempts made to do this caused the solutions of the heat equation to destabilize. In other words, picking too small a spatial step caused the temperatures in the chicken to fluctuate slightly above the temperature of the room and even sometimes drop below the temperature of the room, even though it is physically impossible for the chicken to get colder than the room it is in. Because of these problems, the chicken was made to be rather large in order to produce meaningful solutions.

It is also assumed that the temperature of the room never changes even though in real life, heat would radiate from the chicken into the air, causing the air to heat up. Of course if the chicken-slapping was happening outside on a windy day, then the wind would constantly blow away all the air heated up by the chicken and replace it with colder air. In this case, my approximation that the temperature of the room does not change works, so we will assume the chickens are being slapped outside on a windy day.

4 Results

Chickens	Cooking Time (hours)	Slap Temperature ($^\circ C$)	Slap Frequency (s^{-1})	Standard Deviation
Most-Even	243.2	40	0.4	120.52
Fastest Cook	3.34	3000	5.0	$1.57 * 10^5$

Table 2: This table shows the cooking times, slap temperatures, slap frequencies, and standard deviation of the temperature distribution of the most evenly-cooked chicken and the chicken that cooked the fastest.

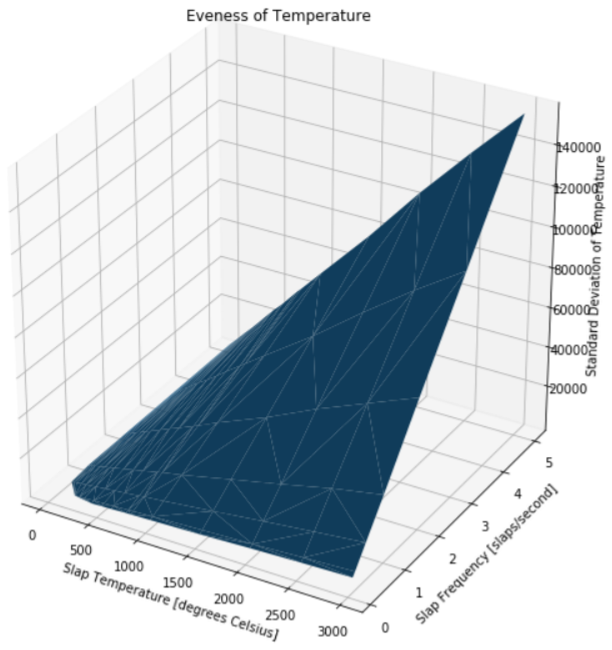

Figure 1: A surface plot showing how slap temperature and slap frequency relates to the standard deviation of the temperature distribution in the chickens.

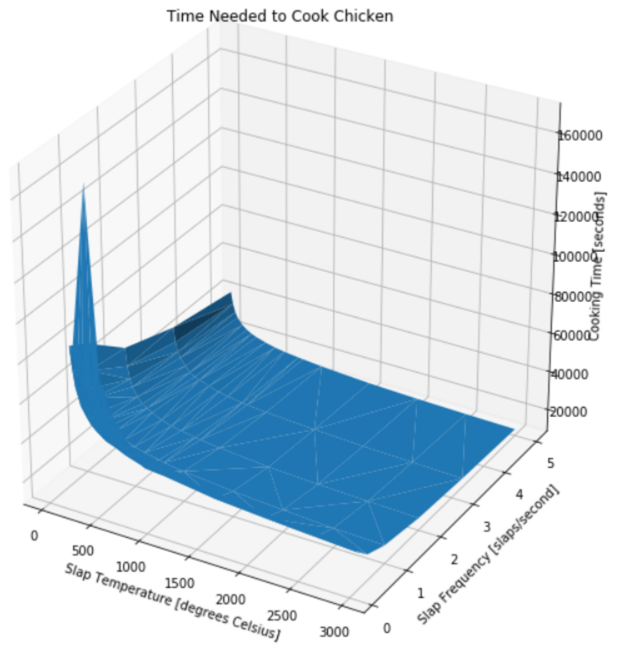

Figure 2: A surface plot showing how slap temperature and slap frequency relates to the cooking time of the chickens. Outliers in cooking time were removed to better show trends in the data.

Chickens	Slap Temperature ($^{\circ}C$)	Slap Frequency (s^{-1})	Cooking Time (seconds)	Std. Dev. of Temp.
Chicken_1	200	0.14	$9.30 * 10^6$	263.29
Chicken_2	200	0.13	$5.55 * 10^6$	243.68
Chicken_3	75	0.25	$1.49 * 10^6$	147.77
Chicken_4	40	0.4	$8.75 * 10^5$	120.52
Chicken_5	200	0.15	$5.45 * 10^5$	240.55
Chicken_6	150	0.18	$4.06 * 10^5$	217.58
Chicken_7	30	0.5	$2.63 * 10^5$	120.75
Chicken_8	150	0.2	$2.16 * 10^5$	249.85
Chicken_9	50	0.4	$1.68 * 10^5$	166.07

Table 3: This table contains data on the outlier chickens removed from Figure 2. The chickens in this table are also more-evenly cooked than all other chickens.

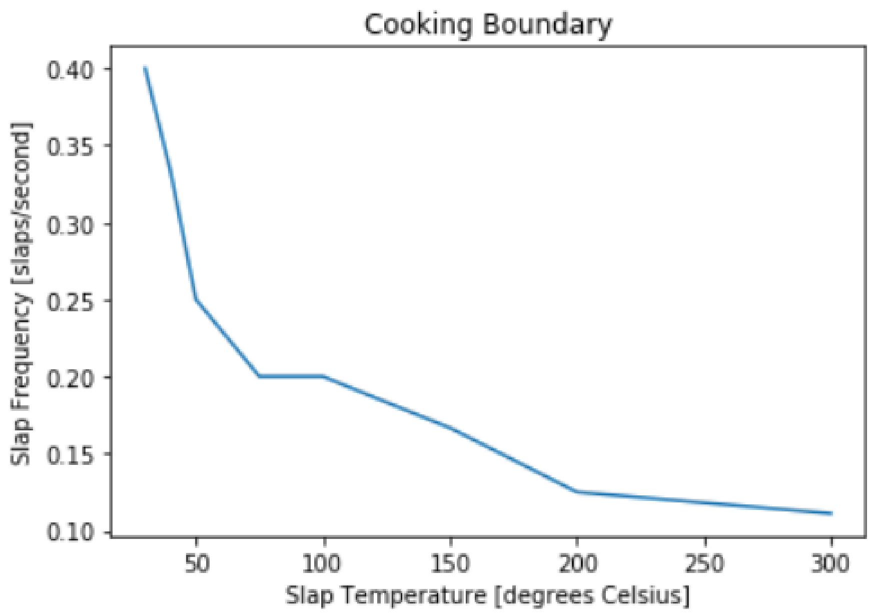

Figure 3: The line in this plot is a boundary of the cooking conditions where all conditions below the line cannot cook a chicken and all conditions above the line can cook a chicken.

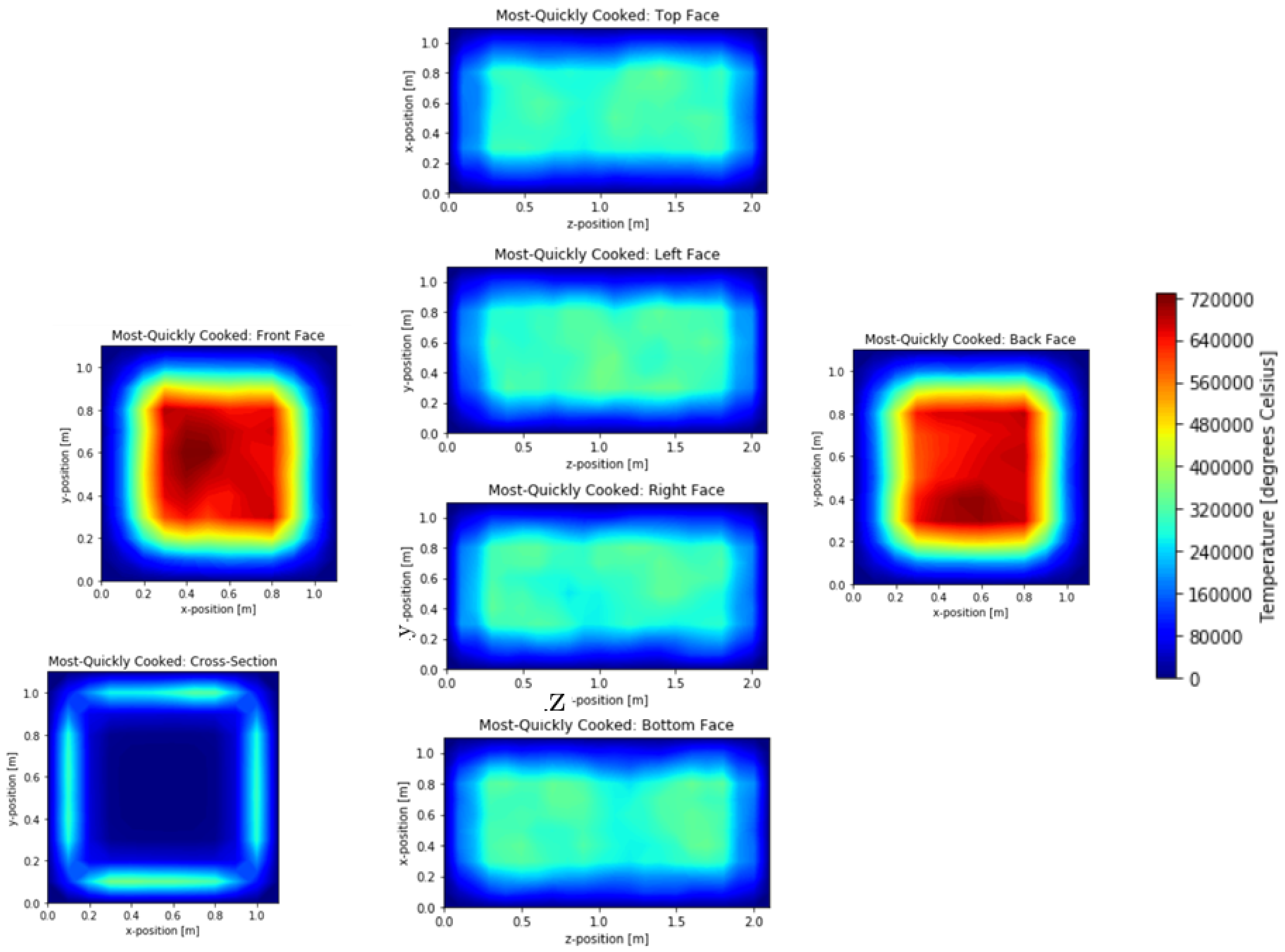

Figure 4: Heatmaps displaying the temperature distribution of the chicken that cooked the fastest. This includes all six faces and a cross-section in the middle of the chicken.

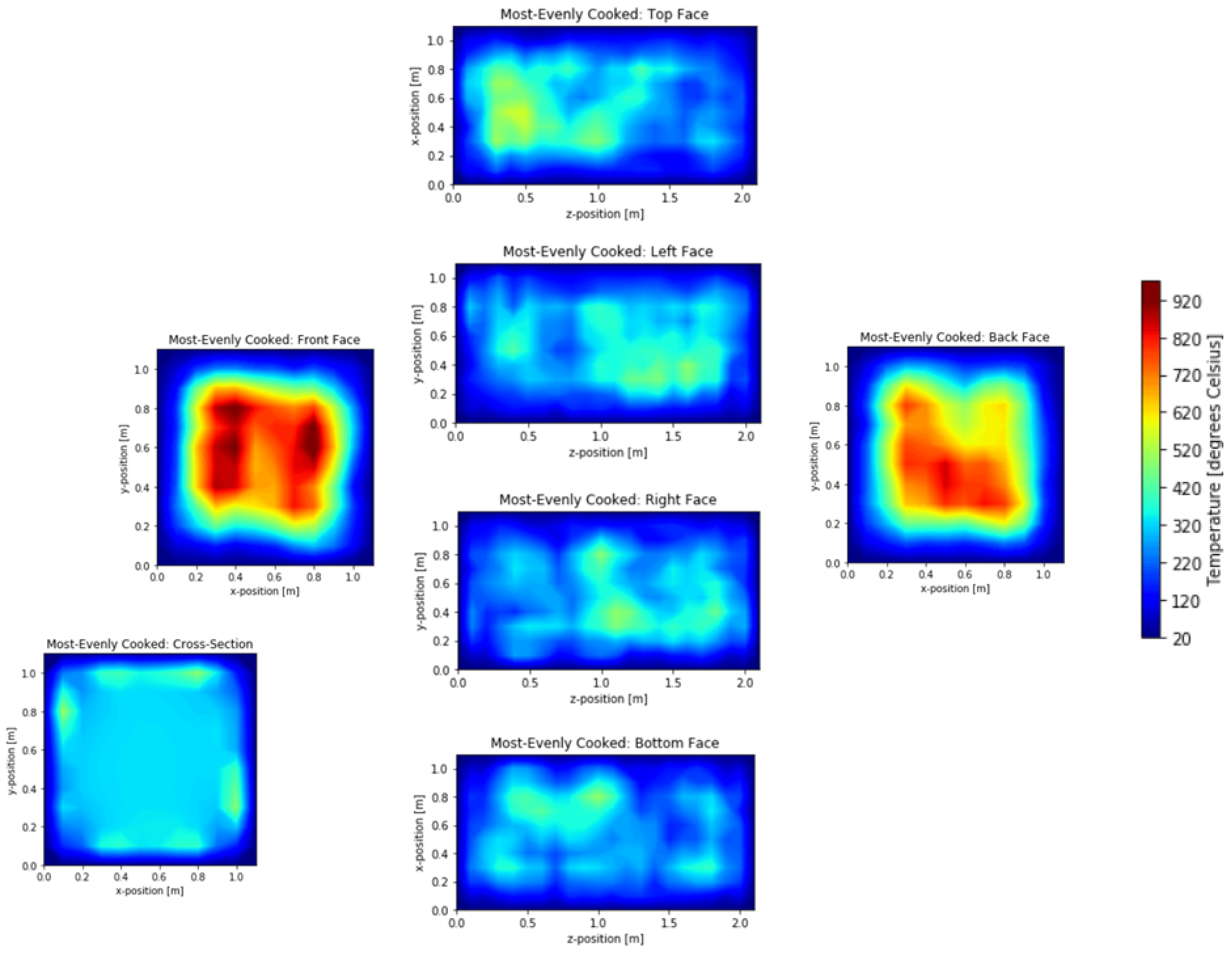

Figure 5: Heatmaps displaying the temperature distribution of the chicken that cooked the most-evenly (minimum standard deviation in temperature). This includes all six faces and a cross-section in the middle of the chicken.

5 Discussion

The chicken that cooked the fastest was cooked with a slap temperature of $3000°C$ and a slap frequency of 5 slaps per second. This makes sense because as the slap frequency and slap temperature increase, the cooking time should decrease, and these slap frequencies and temperatures were the highest frequencies and temperatures I slapped a chicken with. If I picked even higher frequencies or temperatures, then I could expect the chicken to cook even faster.

The chicken that cooked the most-evenly was cooked with a slap temperature of $40°C$ and a slap frequency of 0.4 slaps per second. This result is not exactly what someone might expect at first: the most evenly-cooked chicken should be cooked with low slap frequencies and slap temperatures but I attempted to cook chickens with the same slap temperature but even lower frequencies. Those chickens are not more evenly-cooked because they could not be cooked in the first place. Chickens cooked with slap temperatures and slap frequencies that are too low cannot cook the chicken because of the cooling term. The temperature distribution of the chickens slapped in this way would oscillate about some temperatures instead of gradually increasing with time. This means the minimum temperature in the chicken would never increase to the temperature that I defined to be "cooked", so the chicken could not be cooked under these conditions.

I also expected the chicken that is cooked the most-evenly also takes the longest time to cook. Table 3 shows that the most evenly-cooked chicken did not take the longest time to cook but still took longer to cook than almost all other chickens. A strange result shown in this table is the cooking times for Chicken_1 and Chicken_2: they are both cooked with the same slap temperature but Chicken_1 took longer to cook than Chicken_2 even though it was slapped with higher frequency than Chicken_2. This is the only pair of chickens in my database of about 150 chickens whose cooking times went against the trend of higher slap frequencies cook chickens faster, so this does not strongly affect the conclusions made in this paper.

Something interesting to note is all of the chickens in Table 3 were more evenly-cooked than every other chicken, and their standard deviation of temperature distributions are rather small compared and close to one another in value compared to all the other chickens. This indicates that instead of their being a single slap temperature, slap frequency combination that cooks a chicken the most evenly, there exists a region of combinations that yield a minimum standard deviation of temperature distribution. This region is all points only slightly above the line in Figure 3. This line, which I will call the "cooking boundary" in the rest of this paper, defines the regions where chickens can and cannot be cooked. Chickens slapped with slap temperatures and frequencies above the line will cook but conditions below the line cannot cook a chicken. The cooking boundary in Figure 3 looks like it will blow up to infinity as the slap temperature approaches zero and will never touch the x-axis for large slap temperatures. This makes sense because a slap temperature of zero means the chicken is not actually slapped so it shouldn't be able to cook, and if the slap frequency is zero, this means the chicken is not being slapped so it cannot cook.

Even though the trend in Figure 3 indicates this region of optimal even-cooking conditions might exist, my data shows that chickens slapped with low temperatures above the cooking boundary cooked significantly more evenly than chickens slapped with higher temperatures and also with conditions only slightly above the cooking boundary. The data seems to contradict my idea, but I do not think this contradiction exists. At very low slap temperatures, the chickens' change in the standard deviation of its temperature distribution is very small, almost insignificant, when cooked with slightly different conditions (look at Chicken_4 and Chicken_7), but at higher slap temperatures, the chickens' change in the standard deviation of its temperature distribution changed significantly when there were slight changes in cooking conditions (Chicken_4 and Chicken_9). In a simulation, it is difficult to find the perfect frequencies for these higher temperatures because frequency is tied to the number of iterations in the simulation and the time step, both of which are finite. These perfect frequencies might exist for evenly-cooking chickens with high slap temperatures, but it is difficult to find them in this simulation. If these frequencies were found and applied, that does not necessarily mean these chickens would cook faster than chickens cooked with lower slap temperatures. In Table 3, the chickens are listed in order of longest to shortest cooking time and there is no clear pattern in how slap temperature affects cooking time (at least with the set of chickens that are cooked the most evenly in Table 3), so slapping chickens with high slap temperatures and low slap frequencies might not be the best way to cook a chicken both evenly and quickly. Using the data I have, it appears that slapping a chicken with low slap temperature and a low (but not too low) slap frequency will cook a chicken both evenly and quickly.

The surface of points in Figure 1 shows that as slap temperature and slap frequency increased, the standard deviation of the temperature distribution of the chicken increased. The rate of change for standard deviation along its gradient appears linear, which indicates there is no region of slap temperatures and slap frequencies where slightly changing one of these parameters exponentially changes the standard deviation of the temperature distribution. The surface of points in Figure 2 indicates that as the slap temperatures and slap frequencies go to zero, the cooking time blows up to infinity. This makes sense because a chicken that is only lightly and rarely tapped, rather than slapped, should never be able to cook.

Figures 4 and 5 are heatmaps of various regions of the chicken that cooked the fastest and the most evenly-cooked chicken respectively. At first glance, it might seem like the chicken that cooked the fastest is more evenly-cooked because of how uniform the colors are, but this is not true. Unfortunately, the same colorbar could not be used for both chickens because the chicken that cooked the fastest has points with temperatures three orders of magnitude larger than the highest temperature in the most evenly-cooked chicken. If the same colorbar was used for both chickens, the heatmaps for the most evenly-cooked chicken would be solid dark-blue, which does not make the heatmaps easy to analyze.

The two chickens share similar locations for their hottest and coldest temperature points. Both chickens' hottest points are on the front and back faces of the chicken and these faces are significantly hotter than all other sections. These faces are hotter than the other four faces because the front and back faces have smaller surface areas than the other four faces. This matters because of an oversight in the code written to slap the chicken. The code picks a region on the surface to slap by randomly picking a face to slap and then randomly picking a region on that face to slap. Each face has an equal chance of being slapped and since the front and back faces are made up of fewer points, points on these faces are more likely to get slapped than points on the other faces. If I used more cube-like or spherical chickens, then this difference in temperatures would most likely disappear. Also, the centers of the faces tend to be hotter than points closer to edges. This is because points close to the center of a face are more likely to be slapped since they are more likely to exist in the region randomly chosen to get slapped. This could be fixed by decreasing the size of the slap region or increasing the number of points on a face. I did not make either of these changes because I wanted the chicken to be as close to the size of a real chicken as possible (and it is still too large to resemble a real chicken) and the slap region was also already very small. The temperature difference that would not disappear if these changes were made is the interior of the chicken. The interior will always be significantly colder than the rest of the chicken because it is never slapped directly, so it relies on heat from the surface to travel inward to cook it.

6 Appendix

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

import plotly.plotly as py
import plotly.graph_objs as go
from plotly.graph_objs import *
from plotly.offline import plot

from numba import jit
import time
import random

# dimensions of chicken
meatX = 10 # right = positive
meatY = 10 # up = positive
meatZ = 20 # into the screen = positive

# dimensions of slap
slapX = 3
slapY = 3
slapZ = 3

# dimensions of room
```

```
lenX = meatX + 2
lenY = meatY + 2
lenZ = meatZ + 2

# initial temperatures all in degrees Celsius
temp_room = 20
temp_chicken = 20
slap_temps = [30,50,75,100,150,200,250,300,400,500,600,750,800,1000,1500,2000,2500,3000]

# temperature at which chicken is cooked
temp_cooked = 72

# parameters of heat equation
C = 1840 # J/(kg*C)
dens = 1150 # kg/m^3
K = 0.476 # W/(m*C)

# step sizes
dx = 0.1
dy = 0.1
dz = 0.1
dt = 0.1
dm = dens*dx*dy*dz

# cooling term parameters
h = 100 # W/(m^2 K)
A_corner = dx*dy+dy*dz+dx*dz # m^2
A_edge = 2*dx*dx # m^2
A_surface = dx*dx # m^2

# combine parameters of heat equation
etaX = (K*dt)/(C*dens*(dx**2))
etaY = (K*dt)/(C*dens*(dy**2))
etaZ = (K*dt)/(C*dens*(dz**2))

cool_corner = (h*A_corner*dt)/(dm*C)
cool_edge = (h*A_edge*dt)/(dm*C)
cool_surface = (h*A_surface*dt)/(dm*C)

# different number of iterations inbetween slaps -¿ different slap frequencies
iter_periods = np.array([2,3,5,10,20,50,75])

def init_chicken(frame):
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):
                frame[i,j,k] = temp_chicken
    return frame

def slap(frame,temp):
    face = random.randint(0,5)
    slap_start_X = random.randint(1,meatX+1-slapX)
    slap_end_X = slap_start_X+slapX
    slap_start_Y = random.randint(1,meatY+1-slapY)
    slap_end_Y = slap_start_Y+slapY
    slap_start_Z = random.randint(1,meatZ+1-slapZ)
    slap_end_Z = slap_start_Z+slapZ
    if face == 0:
        # bottom face
        for i in range(slap_start_X,slap_end_X):
            for k in range(slap_start_Z,slap_end_Z):
                frame[i,1,k] += temp
    elif face == 1:
        # left face
        for j in range(slap_start_Y,slap_end_Y):
            for k in range(slap_start_Z,slap_end_Z):
                frame[1,j,k] += temp
    elif face == 2:
        # top face
        for i in range(slap_start_X,slap_end_X):
            for k in range(slap_start_Z,slap_end_Z):
                frame[i,meatY,k] += temp
    elif face == 3:
        # right face
        for j in range(slap_start_Y,slap_end_Y):
            for k in range(slap_start_Z,slap_end_Z):
                frame[meatX,j,k] += temp
    elif face == 4:
        # back face
        for i in range(slap_start_X,slap_end_X):
            for j in range(slap_start_Y,slap_end_Y):
                frame[i,j,meatZ] += temp
    elif face == 5:
        # front face
        for i in range(slap_start_X,slap_end_X):
            for j in range(slap_start_Y,slap_end_Y):
                frame[i,j,1] += temp
    return frame

def is_cooked(frame):
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):
                if frame[i,j,k] ¡ temp_cooked:
                    return False
    return True

def std_dev(frame):
    sum_temps = 0
    num_temps = 0
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):
                sum_temps += frame[i,j,k]
                num_temps += 1
    mean = sum_temps/num_temps

    var_num = 0
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):
                var_num += (frame[i,j,k] - mean)**2
    std_dev = np.sqrt(var_num/num_temps)
    return std_dev

@jit(nopython=True)
def in_corner(i,j,k):
    if i == 1 and j == 1 and k == 1:
        return True
    if i == 1 and j == 1 and k == meatZ:
        return True
    if i == 1 and j == meatY and k == 1:
        return True
    if i == 1 and j == meatY and k == meatZ:
        return True
    if i == meatX and j == 1 and k == 1:
        return True
    if i == meatX and j == 1 and k == meatZ:
        return True
    if i == meatX and j == meatY and k == 1:
        return True
    if i == meatX and j == meatY and k == meatZ:
        return True
    return False

@jit(nopython=True)
def on_edge(i,j,k):
    if i == 1 and j == 1:
        return True
    if i == 1 and j == meatY:
        return True
    if i == 1 and k == 1:
        return True
    if i == 1 and k == meatZ:
        return True
    if i == meatX and j == 1:
        return True
    if i == meatX and j == meatY:
        return True
    if i == meatX and k == 1:
        return True
    if i == meatX and k == meatZ:
        return True
    if j == 1 and k == 1:
        return True
    if j == 1 and k == meatZ:
        return True
    if j == meatY and k == 1:
        return True
    if j == meatY and k == meatZ:
        return True
    return False
```

```
@jit(nopython=True)
def on_surface(i,j,k):
    if i == 1 or i == meatX or j == 1 or j == meatY or k == 1 or k == meatZ:
        return True
    else:
        return False

@jit(nopython=True,parallel=True)
def sim_step(old_frame,new_frame):
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):

                x_comp = etaX*(old_frame[i+1,j,k] + old_frame[i-1,j,k] - 2*old_frame[i,j,k])
                y_comp = etaY*(old_frame[i,j+1,k] + old_frame[i,j-1,k] - 2*old_frame[i,j,k])
                z_comp = etaZ*(old_frame[i,j,k+1] + old_frame[i,j,k-1] - 2*old_frame[i,j,k])

                new_frame[i,j,k] = old_frame[i,j,k] + x_comp + y_comp + z_comp

                # cooling term for parts of chicken on surface
                if in_corner(i,j,k):
                    new_frame[i,j,k] -= cool_corner*(old_frame[i,j,k] - temp_room)
                elif on_edge(i,j,k):
                    new_frame[i,j,k] -= cool_edge*(old_frame[i,j,k] - temp_room)
                elif on_surface(i,j,k):
                    new_frame[i,j,k] -= cool_surface*(old_frame[i,j,k] - temp_room)

    return new_frame

def temp_bounds(frame):
    min_temp = 2*temp_cooked
    min_x = 0
    min_y = 0
    min_z = 0
    max_temp = 0
    max_x = 0
    max_y = 0
    max_z = 0
    for i in range(1,meatX+1):
        for j in range(1,meatY+1):
            for k in range(1,meatZ+1):
                if frame[i,j,k] < min_temp:
                    min_temp = frame[i,j,k]
                    min_x = i
                    min_y = j
                    min_z = k
                if frame[i,j,k] > max_temp:
                    max_temp = frame[i,j,k]
                    max_x = i
                    max_y = j
                    max_z = k
    return ((min_temp,min_x,min_y,min_z),(max_temp,max_x,max_y,max_z))
```

```
def print_results(frame,freq,temp,time,sd):
    print("Finished cooking a chicken after",time,"iterations")
    print("slap temperature:",temp)
    print("iters per slap:",freq)
    bounds = temp_bounds(frame)
    print("min temp is",bounds[0][0], "at", (bounds[0][1],bounds[0][2],bounds[0][3]))
    print("max temp is",bounds[1][0], "at", (bounds[1][1],bounds[1][2],bounds[1][3]))
    print("std dev of temp is",sd)
    print()

# run the simulation
chickens = []
results = []
freq_edge = []
temp_edge = []

for t in slap_temps:
    abort_mission = False
    for n in iter_periods:
        chicken = np.empty((lenX,lenY,lenZ))
        chicken.fill(temp_room)
        chicken = init_chicken(chicken)
        time_step = 0
        old_min_temp = temp_chicken
        while ((is_cooked(chicken) == False) and abort_mission == False):
            old_frame = np.copy(chicken)
            new_frame = np.copy(chicken)
            if time_step % n == 0:
                old_frame = slap(old_frame,t)
            chicken = sim_step(old_frame,new_frame)
            time_step += 1
            if time_step % 100000 == 0:
                new_min_temp = np.amin(chicken)
                if new_min_temp < old_min_temp:
                    print("Chicken is not cooking. Aborting mission.")
                    print("slap temp",t)
                    print("iters per slap",n)
                    print("time",time_step)
                    print(temp_bounds(chicken))
                    freq_edge.append(1/(n*dt))
                    temp_edge.append(t)
                    print()
                    abort_mission = True
        if abort_mission == False:
            sd = std_dev(chicken)
            print_results(chicken,n,t,time_step,sd)
            chickens.append(chicken)
            results.append([time_step,t,n,sd])

# simulation over

# Plot showing boundary of conditions at which chicken can/cannot be cooked

plt.plot(temp_edge,freq_edge)
plt.title("Cooking Boundary")
plt.xlabel("Slap Temperature [degrees Celsius]")
plt.ylabel("Slap Frequency [slaps/second]")
plt.show()

# now analyzing results and making more plots

iters = []
temp = []
ips = []
sd = []

for i in range(0,len(results)):
    iters.append(results[i][0])
    temp.append(results[i][1])
    ips.append(results[i][2])
    sd.append(results[i][3])

times = []
for i in range(0,len(iters)):
    times.append(iters[i]*dt)

freq = []
for i in range(0,len(ips)):
    freq.append(1/(ips[i]*dt))

fig1 = plt.figure(1)
ax1 = fig1.add_subplot(111, projection='3d')
ax1.plot_trisurf(temp,freq,sd)
ax1.set_title("Eveness of Temperature")
ax1.set_xlabel("Slap Temperature [degrees Celsius]")
ax1.set_ylabel("Slap Frequency [slaps/second]")
ax1.set_zlabel("Standard Deviation of Temperature")

fig2 = plt.figure(2)
ax2 = fig2.add_subplot(111, projection='3d')
ax2.plot_trisurf(temp,freq,times)
ax2.set_title("Time Needed to Cook Chicken")
ax2.set_xlabel("Slap Temperature [degrees Celsius]")
ax2.set_ylabel("Slap Frequency [slaps/second]")
ax2.set_zlabel("Cooking Time [seconds]")

trace1 = go.Scatter3d(
    x=temp,
    y=freq,
    z=sd,
    mode='markers',
    marker=dict(
        size=12,
        line=dict(
            color='rgba(217, 217, 217, 0.14)',
            width=0.5
        ),
    opacity=0.8
    )
)

data = [trace1]
layout = go.Layout(
    title = 'A Comparison of Cooking Parameters',
    scene = Scene(
        xaxis=XAxis(title='Slap Temperature [degrees Celsius]'),
        yaxis=YAxis(title='Slap Frequency [slaps/second]'),
        zaxis=ZAxis(title='Standard Deviation of Temperature')
    ),

    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    )
)

fig = go.Figure(data=data, layout=layout)
plot(fig, filename='./sd_graph.html')

trace1 = go.Scatter3d(
    x=temp,
    y=freq,
    z=times,
    mode='markers',
    marker=dict(
        size=12,
        color='rgba(255,69,0,0.84)',
        line=dict(
            color='rgba(217, 217, 217, 0.14)',
            width=0.5
        ),
        opacity=0.8
    )
)

data = [trace1]
layout = go.Layout(
    title = 'A Comparison of Cooking Parameters',
    scene = Scene(
        xaxis=XAxis(title='Slap Temperature [degrees Celsius]'),
        yaxis=YAxis(title='Slap Frequency [slaps/second]'),
        zaxis=ZAxis(title='Cooking Time [seconds]')
    ),

    margin=dict(
        l=0,
        r=0,
        b=0,
        t=0
    )
```

```
)

fig = go.Figure(data=data, layout=layout)
plot(fig, filename='./time_graph.html')

# most evenly-cooked chicken
even_pos = 0
min_sd = results[0][3]
for i in range(1,len(results)):
    if results[i][3] < min_sd:
        min_sd = results[i][3]
        even_pos = i

# fastest-cooked chicken
time_pos = 0
min_time = times[0]
for i in range(1,len(results)):
    if times[i] ¡ min_time:
        min_time = times[i]
        time_pos = i

# heatmaps of most evenly-cooked chicken
max_temp = 920

colorinterpolation = 100
colourMap = plt.cm.jet

# front face

X_front, Y_front = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_front = chickens[even_pos][:,:,1]
sol_front[sol_front == np.inf] = np.nan

plt.title("Most-Evenly Cooked: Front Face")
plt.contourf(X_front, Y_front, sol_front, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_front),
vmax = np.nanmax(sol_front))
plt.xlabel('x-position [m]')
plt.ylabel('y-position [m]')
plt.colorbar().set_label("Temperature [degrees Celsius]")
plt.clim(temp_room,max_temp)
plt.axis('scaled')
plt.show()

# left face

Z_left, Y_left = np.meshgrid(dz*np.arange(0,lenZ,1), dy*np.arange(0,lenY,1))
sol_left = chickens[even_pos][1,:,:]
sol_left[sol_left == np.inf] = np.nan

plt.title("Most-Evenly Cooked: Left Face")
plt.contourf(Z_left, Y_left, sol_left, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_left), vmax
= np.nanmax(sol_left))
plt.xlabel('z-position [m]')
plt.ylabel('y-position [m]')
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# right face

Z_right, Y_right = np.meshgrid(dz*np.arange(0,lenZ,1), dy*np.arange(0,lenY,1))
sol_right = chickens[even_pos][meatX,:,:]
sol_right[sol_right == np.inf] = np.nan

plt.title(”Most-Evenly Cooked: Right Face”)
plt.contourf(Z_right, Y_right, sol_right, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_right),
vmax = np.nanmax(sol_right))
plt.xlabel(’z-position [m]’)
plt.ylabel(’y-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# top face

Z_top, X_top = np.meshgrid(dz*np.arange(0,lenZ,1), dx*np.arange(0,lenX,1))
sol_top = chickens[even_pos][:,meatY,:]
sol_top[sol_top == np.inf] = np.nan

plt.title(”Most-Evenly Cooked: Top Face”)
plt.contourf(Z_top, X_top, sol_top, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_top), vmax
= np.nanmax(sol_top))
plt.xlabel(’z-position [m]’)
plt.ylabel(’x-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# bottom face

Z_bottom, X_bottom = np.meshgrid(dz*np.arange(0,lenZ,1), dx*np.arange(0,lenX,1))
sol_bottom = chickens[even_pos][:,1,:]
sol_bottom[sol_bottom == np.inf] = np.nan

plt.title(”Most-Evenly Cooked: Bottom Face”)
plt.contourf(Z_bottom, X_bottom, sol_bottom, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_bottom),
vmax = np.nanmax(sol_bottom))
plt.xlabel(’z-position [m]’)
plt.ylabel(’x-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# back face
```

```
X_back, Y_back = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_back = chickens[even_pos][:,:,meatZ]
sol_back[sol_back == np.inf] = np.nan

plt.title("Most-Evenly Cooked: Back Face")
plt.contourf(X_back, Y_back, sol_back, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_back),
vmax = np.nanmax(sol_back))
plt.xlabel('x-position [m]')
plt.ylabel('y-position [m]')
plt.colorbar().set_label("Temperature [degrees Celsius]")
plt.clim(temp_room,max_temp)
plt.axis('scaled')
plt.show()

# cross section

X_cross, Y_cross = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_cross = chickens[even_pos][:,:,int(meatZ/2)]
sol_cross[sol_cross == np.inf] = np.nan

plt.title("Most-Evenly Cooked: Cross-Section")
plt.contourf(X_cross, Y_cross, sol_cross, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_cross),
vmax = np.nanmax(sol_cross))
plt.xlabel('x-position [m]')
plt.ylabel('y-position [m]')
plt.colorbar().set_label("Temperature [degrees Celsius]")
plt.clim(temp_room,max_temp)
plt.axis('scaled')
plt.show()

# heatmaps of fastest-cooked chicken
max_temp = 720000

# front face

X_front, Y_front = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_front = chickens[time_pos][:,:,1]
sol_front[sol_front == np.inf] = np.nan

plt.title("Most-Quickly Cooked: Front Face")
plt.contourf(X_front, Y_front, sol_front, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_front),
vmax = np.nanmax(sol_front))
plt.xlabel('x-position [m]')
plt.ylabel('y-position [m]')
plt.colorbar().set_label("Temperature [degrees Celsius]")
plt.clim(temp_room,max_temp)
plt.axis('scaled')
plt.show()

# left face

Z_left, Y_left = np.meshgrid(dz*np.arange(0,lenZ,1), dy*np.arange(0,lenY,1))
sol_left = chickens[time_pos][1,:,:]
```

```
sol_left[sol_left == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Left Face”)
plt.contourf(Z_left, Y_left, sol_left, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_left), vmax
= np.nanmax(sol_left))
plt.xlabel(’z-position [m]’)
plt.ylabel(’y-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# right face

Z_right, Y_right = np.meshgrid(dz*np.arange(0,lenZ,1), dy*np.arange(0,lenY,1))
sol_right = chickens[time_pos][meatX,:,:]
sol_right[sol_right == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Right Face”)
plt.contourf(Z_right, Y_right, sol_right, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_right),
vmax = np.nanmax(sol_right))
plt.xlabel(’y-position [m]’)
plt.ylabel(’z-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# top face

Z_top, X_top = np.meshgrid(dz*np.arange(0,lenZ,1), dx*np.arange(0,lenX,1))
sol_top = chickens[time_pos][:,meatY,:]
sol_top[sol_top == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Top Face”)
plt.contourf(Z_top, X_top, sol_top, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_top), vmax
= np.nanmax(sol_top))
plt.xlabel(’z-position [m]’)
plt.ylabel(’x-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# bottom face

Z_bottom, X_bottom = np.meshgrid(dz*np.arange(0,lenZ,1), dx*np.arange(0,lenX,1))
sol_bottom = chickens[time_pos][:,1,:]
sol_bottom[sol_bottom == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Bottom Face”)
plt.contourf(Z_bottom, X_bottom, sol_bottom, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_bottom),
vmax = np.nanmax(sol_bottom))
plt.xlabel(’z-position [m]’)
plt.ylabel(’x-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# back face

X_back, Y_back = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_back = chickens[time_pos][:,:,meatZ]
sol_back[sol_back == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Back Face”)
plt.contourf(X_back, Y_back, sol_back, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_back),
vmax = np.nanmax(sol_back))
plt.xlabel(’x-position [m]’)
plt.ylabel(’y-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()

# cross section

X_cross, Y_cross = np.meshgrid(dx*np.arange(0,lenX,1), dy*np.arange(0,lenY,1))
sol_cross = chickens[time_pos][:,:,int(meatZ/2)]
sol_cross[sol_cross == np.inf] = np.nan

plt.title(”Most-Quickly Cooked: Cross-Section”)
plt.contourf(X_cross, Y_cross, sol_cross, colorinterpolation, cmap=colourMap, vmin = np.nanmin(sol_cross),
vmax = np.nanmax(sol_cross))
plt.xlabel(’x-position [m]’)
plt.ylabel(’y-position [m]’)
plt.colorbar().set_label(”Temperature [degrees Celsius]”)
plt.clim(temp_room,max_temp)
plt.axis(’scaled’)
plt.show()
```

References

[1] H. C. G. . S. W. J. Sweat, V. E., “Thermal conductivity of chicken meat at temperatures between -75 and 20,” 1973.

[2] “Specific heat of food and foodstuff,” 2019.

[3] . N. M. Kassama, L. S., “Shrinkage and density change of de-boned chicken breast during deep-fat frying,” 2016.

[4] . I. J. N. Ngadi, M., “Natural heat transfer coefficients of chicken drum shaped bodies,” 2005.