Summary

Many conventional rectangular baking pans have a problem that they bake the corners of the batter faster than the rest of the pan. Circular baking pans eliminate this problem, but take up more space in the oven. We propose a solution that given weights for baking consistency and space efficiency based on the importance of each will provide the optimal baking pan shape. We have come up with an algorithm for sorting the pans of area A effectively and formed a model that describes the heat flow into the pan as well as the baking mix itself. The model we developed was based off real-life data that we gathered by baking brownies using careful measurement. We determined several physical properties, such as specific heat, density, and heat transfer coefficients for brownies directly through experimentation. We compared our results with previous studies on the properties of cake-like foods and found that they were similar. Then we ran the model using these coefficients and a variety of different plausible baking parameters, including temperature and baking time.

We also developed a technique for optimally arranging polygonal shaped pans in the oven for any length-to-width ratio to maximize the number of pans that can fit. For polygons, we considered both lattice arrangements and where applicable, tessellating patterns. The algorithm for packing also takes into account the need for airflow between the pans for overall even cooking, which means that the pans are not packed directly next to each other causing uneven heating on the pans located near the sides of the oven. The overall efficiency of a pan shape was calculated using the following equation (where p is a weighting factor):

$$Eff_{total} = p \cdot Eff_{space} + (1 - p) \cdot Eff_{heating}$$

Using the algorithm we found that the type of polygon needed to efficiently use the oven space while heating the pan perimeter was greatly dependent on the dimensions of the oven for weighting values between 0.2-1.0. (Below 0.2, the circular pan is very heavily favored regardless of the oven dimensions.) We also found that adding or subtracting a little bit from the width or length of the oven can have a drastic effect on the spacing efficiency ratio. When we have maximally high efficiency, we call the width and length of the pan "magic numbers."

For example, if the oven dimensions (width by length) are 30" x 40", with two racks, if the area of a pan is set to be 81 in^2, and if space efficiency and baking consistency are given equal weight, we found that the optimal shape is a octagon. Our algorithm is able to compute the optimal shape for any width, length, pan area, and weighting factors using a variety of baking parameters.

Polythermal Bakeware

Stop Worrying About Pans that Unevenly Bake Food and Fit Poorly in Your Oven.

At Polythermal Bakeware we are committed to finding you the optimal baking pans using our newly developed mathematical computer models. Using spatial and thermodynamic models of your oven, we are able to help you select the types of pans that maximize number of pans that can fit in the oven and even distribution of heat for the pan. We have a wide variety of polygonal shaped pans for you to choose from, all made from the highest grade stainless steel.

Our algorithm takes the dimensions of your oven, the area of pan that you want, and how much you would prefer even heating over the number of pans you can fit on a rack to give you the most efficient designed pan possible for your needs. The algorithm utilizes a heat diffusion model of the pan to rate each shape and size based on the pan's ability to heat all points on the perimeter of the pan to the same temperature. Spatially, our algorithm uses an efficiency scale based upon how much oven rack area can be utilized by the pans. By combining these two efficiencies and weighting them based on the baker's preference for either spatial efficiency and even heating. Each customer is unique and we can configure your baking pan for you specific kitchen setup.

Where helping you choose the most efficient baking pan design to meet your needs is our goal.

Polythermal Bakeware
Innovating Baking Technology since 2013

Ph 000.111.2222
Fx 000.111.3333

2864 Fake Street
Fake City, State 00000
www.polythermal.com

Cooking Up the Optimal Baking Algorithm

Introduction to Problem

The problem we chose to solve was how do we create the best brownie pan with edges that are heated evenly, but also fit as many pans as possible on a rack. Obviously a square is most efficient in terms of space, but is found lacking in heating efficiency because the corners are likely to be burnt in comparison with the rest of the edges. A circle shaped pan solves this problem, but is not as efficient space-wise. If we assume each pan has a set area of A, we can optimize what shape will work best in a oven that has a width-length ratio of W/L.

Assumptions and Parameters

Assumptions and Justifications

In the problem we made some assumptions; they are as follows:

1. The oven has only two racks, which are evenly spaced.
2. The oven is pre-heated to the desired temperature when the brownies are placed in the oven. This insures that the oven temperature is constant in our model, and does not "ramp up" during the beginning of the cooking.
3. The oven can be considered a heat reservoir in comparison to any batch of brownies. An oven should be insulated and should not lose significant heat as it is designed to stay at a stable temperature.
4. The pan heats up to the oven temperature fast and maintains its temperature, so for the brownie batter, it can be considered a heat reservoir at the same temperature as the oven.
5. In relation to the brownie, the pan is the primary heating agent. This is especially true around the edges where the air is in contact minimally with the surface area of the edge. The hot air above the brownie pan is a secondary heating agent, which we will incorporate into our model as well.
6. The bottom of any pan is flat, and no pans under 36 square inches will appear in our oven. Flat pans will provide more brownies and will be easier to store. Anything smaller than 6"x6" will be an inefficient use of time and will be a hassle when baking.
7. We also assume that the baker will rotate the racks when cooking in order to cook each batch evenly. Each individual oven has its own hot-spots, and it is up to each baker to figure them out and cook accordingly to his tools.
8. The mass, density and volume taken up by the brownies are constant. This in reality is not usually the case, but the expansion is generally vertical, and should therefore not impact the horizontal heat distribution. Additionally, the change in density, while noticeable, is not significant enough to be worth incorporating into our model.
9. The consistency of the brownies is uniform, and does not change significantly as the brownies are baked. In particular, this means that we are not accounting for the evaporation of water for the brownies. Again, this is not usually true, but for our purposes, we are trying to measure the total amount of heat absorbed, and this is easier to do if we assume the consistency does not change over time. After all, the goal is to ensure that the brownies are evenly cooked, and brownies that get dried out because of water evaporation is just as much of a concern as brownies that are over- or under-heated.

Parameters

The first step towards building our model was setting up some parameters. We knew that the area of the pan (of any shape) had an area of A. The oven itself was rectangular and had a width-length ratio of W/L. There was also a weighting criteria p and $1-p$, where $0 \leq p \leq 1$. The number p "grades" the importance of the space maximization so that if $p = 1$, we only care about how many brownies we cook and not about how well we cook them. If $p = 0$ then the even cooking of brownies is the priority.

Since our focus was on regular polygons, we also examined the parameter n for the number of sides of a regular polygon. In this study, the apothem, denoted a, and the radius, denoted r are important. The angles of the corners of the polygon, are denoted as θ. Our goal is to use as much of the oven area, $W \cdot L$, with a pan of area A having n sides, while still cooking the edges efficiently.

The heating model itself has several parameters. One aspect of the model, which we will go into more detail later, is the geometry of the pan, which has several parameters, like the number of sides and the area. We also needed physical properties of the brownie batter, the pan, and the air in the oven to use in our model. These parameters were density, ρ, specific heat, c, and thermal conductivity, k of the brownie batter. There were also two heat transfer coefficients: one between the pan and the batter, and one between the batter and the hot air in the oven.

Finally there are various baking parameters, such as the temperature of the oven, the initial "room temperature" of the batter, and the length of time the brownies are to be cooked, which we needed to considered.

Analysis of the Problem

Hypotheses

We came up with a few different hypotheses for how to optimize heating and space efficiency individually. First of all, we knew that shapes that had fewer sides would be the most efficient at using area. Squares and regular hexagons are unique in this category because they can tessellate (triangles were not considered due to poor corner heating). However, perfect space efficiency means no heat flow and poor cooking of the brownie, so we would need to space out these tessellation patterns. We hypothesized that using a lattice layout for the shapes would be the most efficient packing pattern.

Our other hypothesis was that since circles are the most efficient at even heating shapes with many sides would also be better than a square at heating the edges evenly. Thus, we believe that either a hexagon or an octagon will give us the best shape overall if each criteria is weighed evenly.

The Model

We divided our model into two main components: packing and heating.

Space Optimization

The algorithm for spatial optimization utilized lattice packing as shown in Figure 1 and a tessellating pattern for certain cases with squares and hexagons as shown in Figure 2. The code for the algorithm is shown in the appendix. Regular polygons with multiple of two sides were used for two main reasons. First regular polygons can be packed into regular patterns that allow for more efficient use of oven space. The other reason for using regular polygons is that a square will pack more efficiently than a circle, but the square has worst heat distribution. However, as you add more sides to the polygon, it comes closer to becoming a circle which means that you can get the some of the properties of efficient polygon lattice packing along with more even heating for higher n-sided polygons. The lattice packing was incorporated into the packing strategy for a couple reasons. Most importantly since we are assuming a rectangular shaped oven, most of the time the lattice packing procedure will be most efficient, and the cases where it is not can be handled with the tessellating packing strategy. Additionally, without airflow between the pans, the pans towards the sides of the oven will heat up faster than the ones in the middle of the oven. It was important that our packing strategy took into account the need for regular spaces between the pans.

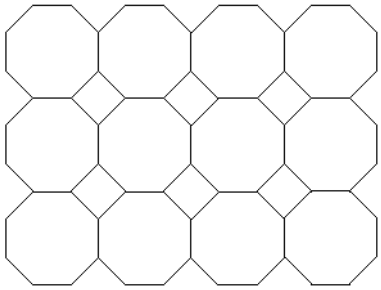
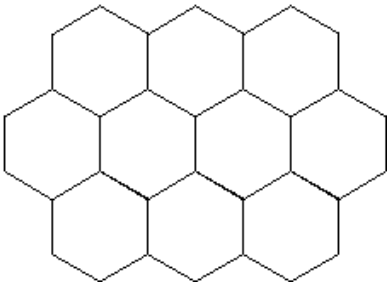


Figure 1: Lattice pattern for octagons.



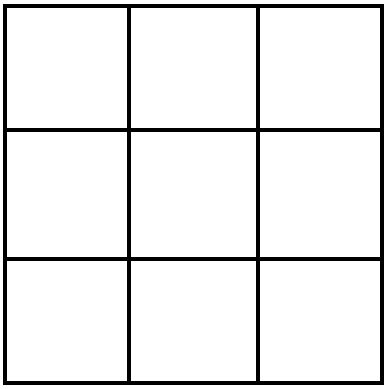


Figure 2: Tessellating patterns for squares and hexagons

The lattice packing algorithm calculates twice the apothem for regular polygons with multiple of 4 sides, with the exception of squares, and then checked to see how many polygons could be fit into a particular length and width shaped oven. For even sided polygons that did not have multiple of 4 sides, the algorithm calculated twice the radius and twice the apothem to get the dimensions of the rectangle they would inscribe, then the algorithm calculated how many would fit in the oven. One thing to note is that for the case of even non-multiple-of-4 sided polygons, the algorithm checks both possible orientations of the polygons in the oven, one with the parallel sides of the polygons spanning the width of the oven and the other with the parallel sides spanning the length. This was done to insure that the maximum number of polygons could be fit into the lattice structured layout. For squares, our algorithm needed to account for there being no natural spaces between adjacent squares. To accomplish this, when the maximum number of squares that could fit in the oven was calculated, the area of the square was inflated by 8%. This meant that while the squares had area A, for finding the maximum number of squares that would fit in the oven, the area used in Equation 1 was 1.08A, where N is the number of polygons that can be fit on the oven rack.

$$Eff_{space} = \frac{N \cdot A}{L \cdot W} \tag{1}$$

This allowed for there to be space between the squares when they were put into the oven to insure airflow between the baking pans. For hexagons, a tessellating algorithm was used to check if the lattice packing method was the most efficient. With the hexagonal tessellation seen in Figure 2 the algorithm again had to account for the need for space between the hexagons for airflow considerations. Thus, for the purpose of calculating how many shapes could fit in the oven, the area of the shape was increased to $1.08A$. This algorithm also checked to see if orienting the tessellation along the width or length of the oven would fit more hexagons. The tessellating orientation was not tried for shapes with higher number of sides because after octagons, the higher order polygons become close enough to circles that the lattice packing method is the more efficient for packing.

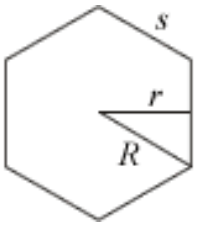


Figure 3: Diagram showing the side length (S), apothem (r), and Radius (R) of a regular polygon.

Heating of Polygons

There are two primary equations governing our heating model. The first is Newton's Law of Heating and Cooling, which governs the rate at which the brownie batter is heated in the pan. The second is the diffusion equation, which governs he rate at which the heat diffuses throughout the batter. Since the system is too difficult to solve analytically, we will do a numerical approximation of the system. We stored the temperatures at various points in the pan, spaced evenly, ran a simulation to calculate how the temperatures at various points change over time. The diffusion equation is straightforward, and we use a standard forward-time, centered-space approximation to simulate its effects. We assumed that the area at the edge of the pan is affected most immediately by Newton's Law, and apply it to the volume within some Δw of the edge of the pan. More precisely, we took all the data points within this Δw, calculated the volume ΔV around each point, and then apply Newton's Law based on a fraction of the surface area, ΔA, of the pan bordering this small volume. We did not neglect the heating of the brownies from the air, and use Newton's Law in the same way to simulate the heating of the top of the brownies from the hot air in the oven. For a given data point, therefore, the full heating equation is:

$$\frac{\partial T}{\partial t} = \frac{K}{c\rho}\nabla^2 T + \frac{h_{pan}}{c\rho}\frac{\Delta A}{\Delta V}(T - T_{pan}) + \frac{h_{air}}{c\rho}\frac{\Delta A}{\Delta V}(T - T_{air}) \tag{2}$$

Determining the surface area of each data point for heating

One of the most difficult problems to solve for our heating model was determining how much surface area each small volume of brownie batter had with the pan. We initially considered just drawing the shape of the pan over a grid and determining how much area fell into each grid space. The problem with this approach was that some edge squares only had a little bit of surface area associated with them simply because of how the grid fell, not because of any characteristics of the geometry of the pan. Unlike most of the pan edge, these squares would fail to rise to the oven temperature when we tested the simulation, incorrectly giving an appearance of inconsistent cooking.

Our next approach was to simply average the area over all the edge squares, so that the heat along all squares on an edge would be uniformly distributed. However, we realized this would also distribute the heat evenly over all the corners, defeating the purpose of the simulation. Our final solution was a hybrid of sorts; we distributed the surface area evenly over the edges, but assigned certain data points to be "corners," which were higher than the other points. For a square, the corners were weighted to have twice the adjacent surface area as the other edge points. For a circle, there are no corners, so we knew the limit of this ratio had to approach one as the number of sides went to infinity. We also knew that what fundamentally changed the local surface area at the corners is the angle of each vertex, which can be found geometrically:

$$\theta = \frac{n-2}{n}\pi \tag{3}$$

What we really needed, then was a formula that evaluated to 2 when $\theta = \pi/2 = 90°$, and approached 1 as θ approached $\pi = 180°$. Ideally, this formula would also drop off quickly as θ moves away from 90°, but would only gradually approach 1 as θ approaches 180°. A formula which met these requirements is as follows:

$$A = 2 - \sin(\theta - \frac{\pi}{2}) = \sin(\frac{n-2}{n}\pi - \frac{\pi}{2}) \tag{4}$$

This is the weight which we gave the surface area adjacent to each corner data point, if the surface area adjacent to an edge data point was 1. To calculate the actual surface area, we just calculated the total surface area of the pan, and then multiply every weight by this total surface area divided by the sum of the unadjusted weights.

Heating Efficiency Model

To determine our heating efficiency, we assigned certain data points to be "corners" (the same ones which had the higher surface area), and others to be "edges". We then measured the average temperature over the entire simulation time at the corners, and at the edges. These average temperatures should be indicative of the heat energy absorbed per unit volume at different point in the pan. Our efficiency was calculated by dividing the average temperature at the edges by the average temperature at the corners. This method of calculating efficiency has two convenient properties: if the edges are not heated at all, or insignificantly compared to the heat at the corners, then the efficiency will drop to zero. However, if the edges and corners are heated evenly, the efficiency will be one. This gives us a good zero-to-one normalized scale to compare with our space efficiency.

Final Algorithm for Efficiency

Our final algorithm to determine the n-sided polygon that fit in an oven of width W and length L used a weighting factor p to weight whether spatial efficiency or even heating efficiency was more desired. The equation used for this final algorithm was:

$$Eff_{total} = p \cdot Eff_{space} + (1-p) \cdot Eff_{heating} \tag{5}$$

Determining Parameters by Experimentation

Procedures

As part of this project we needed to determine a few constants for use in the thermodynamic models of the pans. Using a standard electric oven, we baked two batches of brownies to determine the density, thermal conductivity, specific heat, and heat transfer coefficient for the brownies. We determined these parameters by monitoring the temperature of the brownie mix in a stainless steel pan with a K-Type thermocouple. For the first experiment, we made the brownie and measured the initial temperature, volume, and mass. During the fifty minute cooking period at $350°C$, the temperature at the center and 1cm from the edge was measured at increments of about 10 minutes. At 20 minutes, the mass and volume of the brownie mix was measured. Unfortunately, the thermal measurements required that the oven be opened multiple times, which slightly cooled the oven and the pan.

After baking, a calorimetry experiment was done using the set up as shown in Figure 4 using Styrofoam cups. A small amount of cooked brownie was put into the calorimeter with a set mass of water and the water-brownie mix was stirred for 5 minutes until the temperature equilibrated. The equation to calculate the specific heat is $q = cm\Delta T$. Using the known mass m, specific heat c and change in temperature ΔT for the water, the energy released q by the brownie can be calculated. Then based on the mass and temperature change for the brownie, the specific heat of brownie can be calculated. The calculated specific heat of the brownie was determined to be about 2.0 $J/g°C$.

A second baking experiment was done to monitor the brownies baking in the oven without the need to open the oven door to take temperature readings. Three thermocouples were threaded through some heat shielded tubing and then placed in the center, midway to the edge, and at the edge in the brownie mix. The purpose of this experiment was to get more accurate data for the needed thermodynamic parameters. However, the heat from the surroundings of the oven traveled through the wires to the thermistor causing

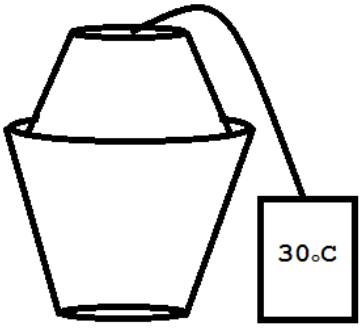


Figure 4: Setup of calorimetry experiment.

inflated temperature measurements. For this reason the data was not used to reevaluate the constants obtained from the first experiment.

Heat Transfer Coefficient Calculations

Figures 5 and 6 show our data from the brownie experiment.

Time (min)	0	10	20	30	40	50
Temperature (C)	22	72	99	102	104	116

Figure 5: Temperature data as a function of time, measured approximately 2mm under the surface of the batter, and about 1cm from the edge of an 8" x 8" baking pan.

Time (min)	0	10	20	30	40	50
Temperature (C)	22	56	85	97	104	104

Figure 6: Temperature data as a function of time, measured approximately 2mm under the surface of the batter at the very center of an 8" x 8" baking pan.

When inspecting the data gathered from our experiments, we found that at about $104°C$ the batter reached a plateau, and did not rise significantly after that. We hypothesized that this was because at this point water started boiling off near the surface, and reduced the temperature at the points we measured. In order to get an accurate value for the heat transfer coefficient, we threw out the data points where the batter had reached $104°C$. This left us with three data points near the edge of the pan and four data points in the middle of the pan; not as many data points as would be preferable to minimize error, but enough to give us an order-of-magnitude estimate of our coefficients as a starting point.

We used the data collected at the edge of the pan to calculate a heat transfer coefficient between the pan and the brownie mix. Since the temperature at this point should be dominated by the heat from the pan, we can assume for our purposes that our temperature data can be completely accounted for by heat transfer from the pan. The differential equation given by Newton's Law of heating/cooling can be solved analytically for temperature.

$$T(t) = T_{Pan} + (T_{Room} - T_{Pan})e^{-\frac{h}{c\rho}\frac{A}{V}t} \tag{6}$$

The fraction $\frac{A}{V}$ is the ratio of surface area to volume. Another way of looking at this for a square pan is that it is the multiplicative inverse of the depth within the brownie batter for which the temperature is a characteristic value. We measured the temperature at a distance of approximately 1 cm from the pan edge, so it should be a characteristic of the temperature in the region extending up to 2 cm from the edge. We will therefore use 50.0 m^{-1} as the value of $\frac{A}{V}$ to calculate the heat transfer coefficient between the pan and the brownies.

Finally, we know the room temperature, the pan temperature, we have already calculated the specific heat of brownie batter, and we already measured the mass of the batter. After performing a least squares fit on our data using these coefficients and the equation above, we found that the heat transfer coefficient between the pan and the brownies is approximately 47 $\frac{W}{m^{2\circ}C}$.

We use a similar method to measure the heat transfer coefficient between the brownies and the air in the oven. For this we use the data collected in the center of the pan. Since this point is farthest from any of the edges of the pan, it is dominated by heat from the hot air above the pan. However in this case, $\frac{A}{V}$ is the multiplicative inverse of the depth of the measurement region under the surface. Since the distance of the probe under the surface was approximately 2 mm, the depth of the effective measurement region is 4mm, and $\frac{A}{V}$ is 250.0 m^{-1} All other parameters are the same as before including the temperature, since we assumed that the oven and the pan are both the same temperature. Once again, we did a least squares fit and found that heat transfer coefficient between the brownies and the air is approximately 8.8 $\frac{W}{m^{2\circ}C}$.

Mass, Volume, and Density

During the course of our experiments, we measured the mass and volume before, during, and after the baking process. Using mass and volume, we were able to also compute density. We found that the density of the uncooked batter was $1040kg/m^3$, but dropped to $840kg/m^3$ after 20 minutes of baking. The density did not drop significantly more between the 20 minute mark and the end of the 50 minute baking process.

In order to measure the density, we also had to measure the mass. We found that the batter lost some mass over the entire course of the baking process, roughly linearly with respect to time. The uncooked batter initially weighed 1.112 kg. After 20 minutes it dropped to 1.082 kg. By the end of the baking process the total mass was 1.038 kg. We believe that this loss of mass was due to water evaporating off the surface. So some of the change in density was due to the loss of mass, but most was due to the rising of the brownies.

Although we found that the mass, volume, and density changed over time, we ultimately chose to keep them as constants in our model. This made the code simpler, and since we did not have time to take enough measurements to ensure that the relationships were, in fact, linear, we thought it would be better to just keep them as constants.

Comparison with previous studies

Baik et. al studied the properties of cupcakes and, like our study, measured their density and specific heat, and additionally determined their thermal conductivity, which we were not able to measure.[1] While cupcakes can have a slightly different consistency than brownies, they are close enough to verify that our figures are on the right order of magnitude and the thermal conductivity should be a good order-of-magnitude starting point, so that we could calculate a thermal diffusivity for the batter. Baik et. al found that the density of uncooked batter was 803 kg/m^3, and dropped to 236 kg/m^3 after the batter was cooked. This is a lower density than we measured, and it dropped even more, but this is not unreasonable considering that cupcakes are generally less dense than brownies. They measured that the specific heat varied over the baking period from 2516 $J/kg^{\circ}C$ to 2658 $J/kg^{\circ}C$, but stayed relatively constant. This is fairly consistent with our measured value of specific heat, 2000 $J/kg^{\circ}C$, which was a bit lower. Finally, they measured that thermal conductivity varied between 0.1064-0.2064 W/mK We started with thermal conductivities in these ranges, but found that to get results more like what we expected, a slightly larger thermal conductivity in the range of 0.3-0.4 W/mK was necessary.

Reduction of physical constants to algorithm parameters

In our algorithm we simplified the code by reducing the number of parameters. For diffusion, the fraction $\frac{K}{c\rho}$ can be reduced to a single constant α. We found that the value of α that worked the best was $18.0\times10^{-8}m^2/s$, which was consistent with the order of magnitude of the parameters that we measured.

Similarly, for heating from the pan and the air, the fraction $\frac{h}{c\rho}$ can be reduced to a single parameter as well, which we called β for the pan to brownie heat transfer, and γ for the air to brownie heat transfer. We calculated them as follows:

$$\beta = \frac{h_{pan}}{c\rho} = \frac{50W/(m^{2\circ}C)}{2000J/kg\circ C \times 1000kg/m^3} = 2.5 \times 10^{-5} m/s \tag{7}$$

$$\gamma = \frac{h_{air}}{c\rho} = \frac{8W/(m^{2\circ}C)}{2000J/kg\circ C \times 1000kg/m^3} = 4.0 \times 10^{-6} m/s \tag{8}$$

Results

As we assumed, there was no one ideal shape to our model. Every optimal shaped hinged on the dimensions of the pan, the weighting factor, and the desired area of the pan. When we ran our algorithm, no one shape dominated every situation. Typically, for lower values of p the circle dominated, but as p approached 1, the square was most efficient in just as many situations as the circle, followed closely by the hexagon. In general, to determine which shape would be most efficient space-wise we must take "magic numbers" into account.

Magic Numbers

Magic numbers are the dimensions of the pan, W by L, that provide a local maximum in terms of spacing efficiency. For a regular polygon with the number of sides being a multiple of four, the formula for a magic number would be

$$W = L = 2 \cdot apothem[n] \cdot k$$

where k is a natural number. This means a W/L ratio of one would be ideal for these types of shapes. This is because these polygons have a length and width equal to the length of two of their apothems. It follows that the width and length of the pan should be an integer multiple in order to maximize efficiency. If these dimensions were slightly less than an integer multiple, we would be unable to fit another row or column in the pan, thus creating a lot of unused space. This is why we call the dimensions magic numbers. It is because they are the most optimal local dimensions.

For a polygon that has an even number of sides but is not divisible by four, the formula is a little different. Without loss of generality, if r is the apothem and R is the radius:

$$W = 2 \cdot r \cdot j \tag{9}$$

$$L = 2 \cdot R \cdot k \tag{10}$$

where j, k are a natural numbers. These types of polygons are different because they are stacked side to side and corner to corner. The length from opposite corners is longer than the length from opposite sides, so that is why these have a different equation for magic numbers.

Optimization

In order to get the overall efficiency, we must use the equation

$$Eff = p\frac{N \cdot A}{W \cdot L} + (1-p)\frac{E_{min}}{E_{max}} \tag{11}$$

To be clear, N is the number of shapes of area A that we can fit into a W by L pan. It is natural to make the assumption that the percentage of area used would be a good grading scale for efficiency in area. It has the bounds from zero to one and has a wide range of values.

The fraction $\frac{E_{min}}{E_{max}}$ is the ratio between the area on the edge that absorbed the least heat compared to the area on the edge that absorbed the most heat. We define this to be our grading criteria for heating because if fits in the range from zero to one, and will directly compare how evenly the sides will be heated.

Again, p is the weighting factor. The equation for efficiency has a maximum value of one, but no shape will satisfy that number. Hence we designed an algorithm that takes in the previous inputs and outputs the most efficient shape. We have a packing algorithm that determines how many of one shape can fit inside an area, and we used a least-squares estimation to find an equation that approximates the data we get from

our heating model. Using these two things we multiply by the weights and whichever shape has the highest efficiency is the one we determine is the best shape for the above conditions.

Optimal Shapes

After fitting our heat model to an equation, we optimized space and heating with a variety of pan areas with different weights. We tallied the number of "hits" for each pan shape over all pan areas tested for a given choice of weights and obtained the following results:

p	4	6	18	20	circle
0.00	0	0	1	0	6
0.10	0	0	1	1	5
0.20	1	0	1	1	4
0.30	2	0	1	0	4
0.40	2	1	1	0	3
0.50	2	2	0	0	3
0.60	2	2	0	0	3
0.70	2	2	0	0	3
0.80	2	2	0	0	3
0.90	2	2	0	0	3
1.00	6	1	0	0	0

Figure 7: Table of tallies for different shapes over all tested pan areas.

While it appears that circle seems to win most of the time, for certain areas this may not be the case.

Efficiency

Finally we see that no one shape can dominate over every circumstance. The optimal shape depends on the parameters given. Overall, the three most efficient shapes under our optimization methods are the circle, square, and hexagon. Should our goal be to mass produce, we would want to have circular pans be our product. Otherwise tailoring our product to what the consumer desired would be ideal, as we could charge more for custom products. As far as producing the optimal shaped pan, it depends on the situation. However, given the situation we can always find the best pan for it.

Error/Sensitivity Analysis

Since we have had an experiment as part of our paper, one must wonder how accurate our physical constants? In all likelihood they are subject to error, and we must know how much could we tweak these constants before our model becomes void? Let us make a list of what these parameters are:

- Our three physical constants α, β, and γ
- The maximum temperature of the pan
- The height of the brownies being cooked
- The time we cook the brownies
- The area of the brownies
- The shape (number of sides) of the brownie pans

Varying the temperature in the pan did not vary our results significantly. Our standard temperature was 200°C and we the range of values we tested were from 100°C to 300°C, which were well beyond the standard normal baking temperatures. In this we discovered that the lower the temperature, the higher

the efficiency. Over all the shapes we considered, there was a 3% difference between the efficiency of two temperature extremes in the worst case. Since we are dealing with efficiency ratios of $> 80\%$, 3% cannot be ignored. However, the efficiency difference between the standard (200°C) and our 300°C is about 0.7% in the worst case which is significantly better than the difference between 300°C and 100°C. Since we are reaching equilibrium quicker with the higher temperature, we can say that using our standard temperature is an accurate source of data.

Next on the list is the variance of height in the brownie. We should expect that since we are evaluating the middle layer of the brownie that raising the height will take away from the contributions of the bottom of the pan and the top layer of air, making it a more accurate as the height increases. Our standard height was about 1.5 inches, or 0.04 meters. Lowering height raises efficiency, but that is because we need to take into account the heat from the bottom and top of pan, making our efficiency rating more inaccurate. The difference in efficiency between 1 inch and 6 inches for height was a significant 5.5%, with around a 3% each way. Even if we go up to 2 inches our efficiency ratio decreases by 1.3%. Further analysis would tell us the ideal height to measure at.

Now it is time to evaluate the baking time. As we might expect, if we lower the baking time the efficiency ratio decreases since there is not enough time for the middle to be heated up. If we run it too long however, every brownie reaches the saturation temperature and all efficiencies will be 1. We chose the middle ground of 30 minutes as our standard, which is not too different from normal baking time, considering the temperature we chose. Varying the time to 50 minutes we get about a 5% difference, and lowering it 25 minutes has the same effect. As far as any sources of error, this was perhaps our greatest contribution. Choosing a standard was hard, but we needed some sort of compromise between too long and too short and 30 minutes fit the bill.

As far as area goes, we were pretty consistent and had little error when modeling this. Since we naturally checked many areas, we can see that as area increases, the ratio of heat efficiency decreased proportional to the inverse of the root of the area. Since we checked pans about half the assumed area minimum and the equation still fit, we need not worry about error changing when we assumed the pan size is less than our required minimum size. The same holds true for pans that have a meter of squared area. The oven still will heat according to our modeled equation. Little error shows up here.

Considering what this project is asking for, we must see how changing the shape will change our heating efficiency. Our model shows what we would expect; a shape with more sides has higher efficiency. We define circles to have exactly an efficiency ratio of one, and other shapes we will have the ratio of the heating of the middle of a side and the heating of a corner of our shape. From that we see that a square will have worse efficiency as far as heating goes, and in general will be no less than 80%, which is what is to be expected. Since again we were modeling how well the shapes heat evenly, we came up with an equation that would describe the efficiency in heating a shape with n sides and got to about 0.1% accuracy. Since modeling is not a problem, little error arises from here as well.

Sensitivity with respect to heating parameters

There are essentially four parameters in our heating model: K, the thermal conductivity, h_{pan} and h_{air}, the heat transfer coefficients between the brownies and the pan and the air, respectively, and finally the product of specific heat and density, $c\rho$. These last two are one parameter because they only appear in the model together, so sensitivity can be performed on their product because any error in one individually could be canceled out by equal and opposite error in the other without affecting the model.

To test each parameter, tests were run on a range of values for the parameter with all other parameters constant. Our standard was to bake the brownies at $200°C$ for 30 minutes in a 40cm x 40cm pan with batter 4 cm deep. Our grid was 21 units in each horizontal direction and 10 units vertically, and we used 180 time steps to complete the simulation. We ran the test for even-sided polygons between 4 and 20 sides.

We varied α, the total thermal diffusivity, between 1.8×10^{-9} and 1.8×10^{-7}. This is equivalent to varying the thermal conductivity K while keeping specific heat and density constant. At diffusivity values higher than 1.8×10^{-7}, the diffusivity was too high for the number of time steps, and the model failed to converge. We found that the efficiencies generally increased as α increased, but they did not change very much, and the overall shape of the graph tended to stay the same. The maximum difference in efficiency over these difference of two orders of magnitude in α was 4%. This is relatively insignificant considering how

much we changed α by. Therefore, even if our value for α was off by an entire order of magnitude, we would not expect that it would change our results significantly.

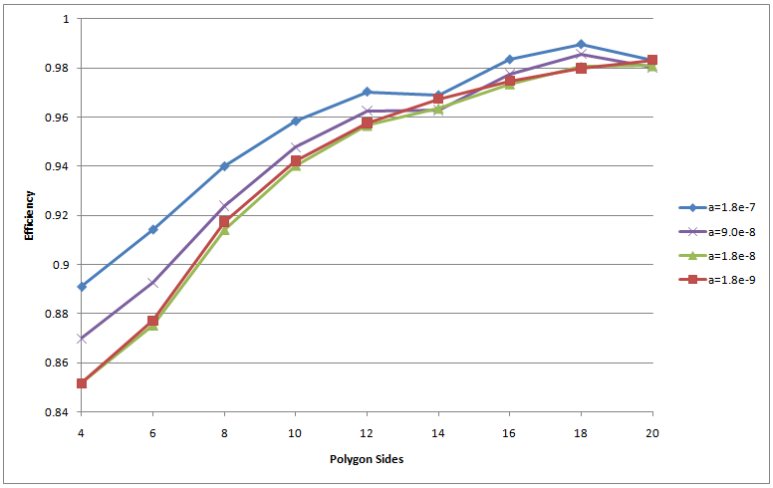


Figure 8: Efficiency of various shapes with different values for α.

β, defined as $\frac{h_{pan}}{c\rho}$, we allowed to vary between 2.5×10^{-7} and 2.5×10^{-4}. Similar to α, we found that for values higher than 2.5×10^{-4} the system heated too fast in one time step and the algorithm did not converge. We found that the system did not heat significantly when β was less than 1.0×10^{-5}, so we threw those data point out. For other β values, higher β tended to result in higher efficiencies. Unlike α, though, the shape did change; when the batter heated faster, it tended to also reach equilibrium at the edges faster, and therefore the temperatures were generally more consistent there. The maximum difference in efficiency was 16% over just above one order of magnitude. However, in real life, if β was indeed higher, a baker would probably not cook the brownies for as long, so as not to burn them. With a shorter cooking time, there probably would be a more noticeable difference between the corners and the edges for higher β values.

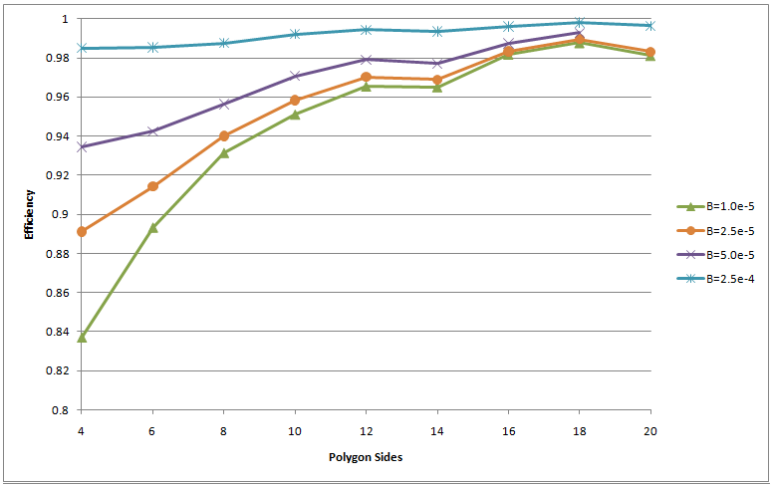


Figure 9: Efficiency of various shapes with different values for β.

Finally, we allowed γ, defined as $\frac{h_{air}}{c\rho}$, to vary between 4.0×10^{-8} and 4.0×10^{-4}. Like β, increasing γ tended to increase the efficiency slightly. However, even over this large range, it did not affect our efficiencies significantly at all; the maximum different in efficiency was 3.5%. This is because the heating from the air is only secondary to the heating from the pan, and primarily affects just the batter right on the surface. Therefore we can conclude that our results are not particularly sensitive with respect to γ.

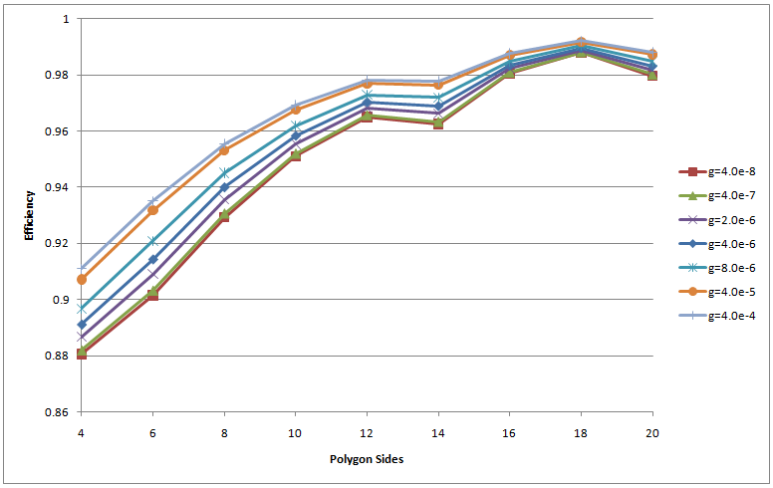


Figure 10: Efficiency of various shapes with different values for γ.

One last test that we ran was to vary all three equally. Since all three have $c\rho$ in the denominator, this is equivalent to varying $c\rho$ while keeping K, h_{pan}, and h_{air} constant. This not only had a significant impact on the efficiency, but also on the temperature. More than doubling $c\rho$ had the effect that the batter was hardly baked at all, even on the edges. Reducing $c\rho$ improved the efficiency dramatically, but more than doubling it resulted in heating too fast for the time step we used. Between halving and doubling $c\rho$, we observed a maximum efficiency difference of 15%. While this difference is very significant, the difference in absolute temperature is more significant - telling us that we probably have fairly accurate values for c and rho.

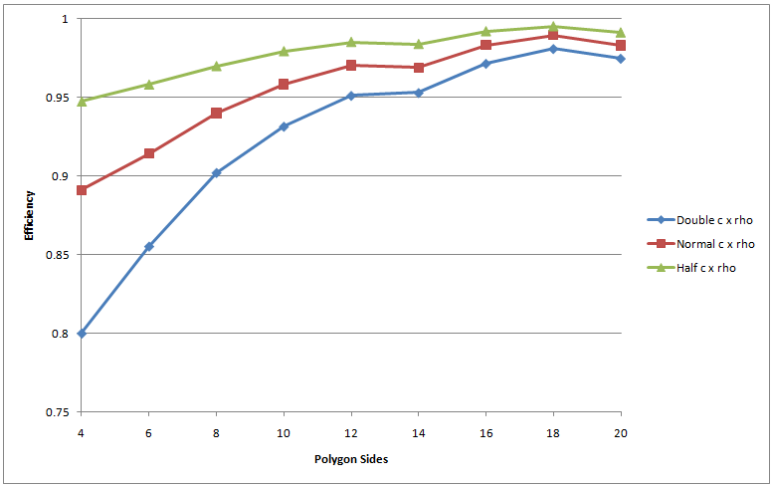


Figure 11: Efficiency of various shapes with different values for $c\rho$.

In summary, the parameters that likely affected our model the most were the heat transfer coefficient between the pan and the brownies, and the combination of specific heat and density of the batter. However, we believe that the parameters that we chose for these values are fairly accurate due to the reasonable temperature values we got at the end of the simulations for reasonable values of other parameters like baking time and oven temperature.

Analysis of our Model

Strengths

Our heating model is quite robust and is able to generate predictable results over a variety of temperatures, baking times, and other parameters. For instance, cooking the batter for a longer period of time for a shorter temperature results in more even baking than baking for a shorter period of time under a higher temperature, as expected. By limiting our shapes to polygons, we were also able to easily guarantee that our packing algorithm gives an optimal number of pans for a given shape, area, and oven size. Finally, we were able to experimentally determine several of our parameters and compare them with previously published results to verify their validity. Moreover, these parameters gave us the most reasonable results in our heating model as well, given reasonable values for other parameters like baking time and temperature.

Weaknesses

One weakness of our heat model is that in discretizing the brownie pan, it is possible we still lost some distinguishing features of a pan design, that might have been noticeable with a finer grid, or even a different method of discretizing the space (for instance, a radial method). It is also possible that some artifacts could have been added due to our methods as well. Additionally, we were not able to rigorously prove that our method of assigning surface areas for the edge and corner data points scientifically accurate. However, most of our results were what we predicted would occur intuitively, so we would suggest that improving our methods in these ways would not significantly impact the results we obtained.

Another weakness is that we do not account for certain effects that occur at high temperatures, like the evaporation of water, the restructuring of the batter molecules, and the resulting changes in mass and density. A more refined model could account for these to give more accurate results.

Finally, one last weakness is that we only considered regular polygons in our model. With more time, we could consider non-regular polygons and various curved shapes like ellipses as well.

Future Improvements

Just like any other project, improvements can be made. Perhaps a better scheme for measuring efficiency in even heating could be developed. Besides that, we would like to get more accurate parameters from our measurements with better equipment. Consensus on the way of standardizing the time and temperature to fit real-life brownie making would also be necessary. Convection currents were also hard to model, and figuring out how much open area is needed in the lattice diagram would provide a better efficiency in spacing for the square and hexagon. One last consideration would be to see how radiation from the sides of the oven would heat up our product, and how that would affect our heat transfer.

Conclusion

We have just presented two models, one for packing polygons and one to model the heating of a polygon-shaped pan in a conventional oven. Our packing model was able to efficiently organize polygons using tessellating techniques and lattice structures. Our heating model produced reasonable results and gave us efficiency ratings for various shapes, areas, and baking parameters. Finally, all this was based on experimental data that we performed by baking brownies, taking measurements, and fitting them to our model. The best overall shape depends strongly on the parameters given; there is no optimal shape. Instead, for a given application our algorithm should simply be run to determine the best shape for the circumstances. For every customer there is one optimal pan, but there is no one optimal pan for every customer.

References

[1] O. D. Baik, S. S. Sablani, M. Marcotte, and F. Castaigne. *Modeling the Thermal Properties of a Cup Cake During Baking.* Journal of Food Science. Volume 64, No. 2, 1999.

[2] Baking Tips & Common Baking Solutions. *Baking Tips & Common Baking Solutions from Firenza Irresitable Baking Mixes.* N.p., n.d. Web. 04 Feb. 2013. http://www.great-recipes.com/baking-tips/index.html

[3] Figoni, Paula. *How Baking Works: Exploring the Fundamentals of Baking Science.* Hoboken, NJ: Wiley, 2008. Print.

[4] "Mathwords: Area of a Regular Polygon" *Mathwords: Area of a Regular Polygon.* N.p., n.d. Web. 04 Feb. 2013. http://www.mathwords.com/a/area_regular_polygon.htm

Appendix: Source Code

Heating Model

```
(* Function to generate a table of surface areas for the data points, \
where an edge has a value of 1, a corner has a value greater than or \
equal to one, and an interior data point or a point outside the \
simulation area has a value of 0.  More or less brute force. *)
CalculateBorder[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    Result[[iMin]][[jMin]] = N[2 - Sin[((n - 2)/n) \[Pi] - \[Pi]/2]];

    If[X2 == X1,
     If[jMin > jMax,
      For[j = jMax + 1, j < jMin, j = j + 1,
       Result[[iMin]][[j]] = 1;
       ],
      (* else *)
      For[j = jMin + 1, j < jMax, j = j + 1,
       Result[[iMin]][[j]] = 1;
       ]
      ],
     (* else *)
     Slope = (Y2 - Y1)/(X2 - X1);
     If[Abs[Slope] > 1,
      If[jMin > jMax,
       For[j = jMax + 1, j < jMin, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
               X2)] + 1;
        Result[[i]][[j]] = 1;
        ],
       (* else *)
       For[j = jMin + 1, j < jMax, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
               X1)] + 1;
        Result[[i]][[j]] = 1;
        ]
       ],
      (* else *)
      If[iMin > iMax,
       For[i = iMax + 1, i < iMin, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
               Y2)] + 1;
        Result[[i]][[j]] = 1;
        ],
       (* else *)
       For[i = iMin + 1, i < iMax, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
               Y1)] + 1;
        Result[[i]][[j]] = 1;
        ]
       ]
      ]
     ]
    ];
   Result
```

```
   ];

(* Function to generate a mask table where the corner data points \
have an value of 1 and all other data point have a a value of 0 *)
CalculateCornerMask[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    Result[[iMin]][[jMin]] = 1;
    ];
   Result
   ];

(* Function to generate a mask table where the non-corner edge data \
points have an value of 1 and all other data point have a a value of \
0 *)
CalculateEdgeMask[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    If[X2 == X1,
       If[jMin > jMax,
        For[j = jMax + 1, j < jMin, j = j + 1,
         Result[[iMin]][[j]] = 1;
         ],
        (* else *)
        For[j = jMin + 1, j < jMax, j = j + 1,
         Result[[iMin]][[j]] = 1;
         ]
        ],
       (* else *)
       Slope = (Y2 - Y1)/(X2 - X1);
       If[Abs[Slope] > 1,
        If[jMin > jMax,
         For[j = jMax + 1, j < jMin, j = j + 1,

          i = HalfSubdiv +
            Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
                X2)] + 1;
          Result[[i]][[j]] = 1;
          ],
         (* else *)
         For[j = jMin + 1, j < jMax, j = j + 1,

          i = HalfSubdiv +
            Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
                X1)] + 1;
          Result[[i]][[j]] = 1;
          ]
         ],
        (* else *)
        If[iMin > iMax,
         For[i = iMax + 1, i < iMin, i = i + 1,

          j = HalfSubdiv +
```

```
            Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
                Y2)] + 1;
           Result[[i]][[j]] = 1;
           ],
          (* else *)
          For[i = iMin + 1, i < iMax, i = i + 1,

           j = HalfSubdiv +
            Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
                Y1)] + 1;
           Result[[i]][[j]] = 1;
           ]
          ]
         ]
        ]
       Result[[iMin]][[jMin]] = 0;
     ];
    Result
    ];

(* Function to generate a mask table where the corner data points \
have an value of 1 and all other data point have a a value of 0 *)
CalculateStencil[n_, subdiv_] :=
  Module[{\[CapitalTheta]1, \[CapitalTheta]2, X1, X2, Y1, Y2, Slope,
    HalfSubdiv, Result, iMin, iMax, jMin, jMax, temp, i, j},
   Result = Table[Table[0, {j, 1, subdiv}], {i, 1, subdiv}];
   For[k = 1, k <= n, k = k + 1,
    \[CapitalTheta]1 = 2 \[Pi]/n*(k - 1);
    \[CapitalTheta]2 = 2 \[Pi]/n*k;
    X1 = Cos[\[CapitalTheta]1];
    Y1 = Sin[\[CapitalTheta]1];
    X2 = Cos[\[CapitalTheta]2];
    Y2 = Sin[\[CapitalTheta]2];
    HalfSubdiv = Floor[subdiv/2];
    iMin = HalfSubdiv + Round[HalfSubdiv*X1] + 1;
    iMax = HalfSubdiv + Round[HalfSubdiv*X2] + 1;
    jMin = HalfSubdiv + Round[HalfSubdiv*Y1] + 1;
    jMax = HalfSubdiv + Round[HalfSubdiv*Y2] + 1;

    If[X2 == X1,
     If[jMin > jMax,
      For[j = jMax, j <= jMin, j = j + 1,
       If[iMin > HalfSubdiv,
        For[i = iMin, i >= HalfSubdiv, i = i - 1,
         Result[[i]][[j]] = 1;
         ],
        (* else iMin <= HalfSubdiv *)
        For[i = iMin, i <= HalfSubdiv, i = i + 1,
         Result[[i]][[j]] = 1;
         ]
        ]
       ],
      (* else jMin <= jMax *)
      For[j = jMin, j <= jMax, j = j + 1,
       If[iMin > HalfSubdiv,
        For[i = iMin, i >= HalfSubdiv, i = i - 1,
         Result[[i]][[j]] = 1;
         ],
        (* else iMin <= HalfSubdiv *)
        For[i = iMin, i <= HalfSubdiv, i = i + 1,
         Result[[i]][[j]] = 1;
         ]
        ]
       ]
      ],
     (* else X2 != X1 *)
     Slope = (Y2 - Y1)/(X2 - X1);
     If[Abs[Slope] > 1,
      If[jMin > jMax,
       For[j = jMax, j <= jMin, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X1 - X2)*(j - jMax)/(jMin - jMax) +
              X2)] + 1;
        If[i > HalfSubdiv,
         For[i = i, i >= HalfSubdiv, i = i - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else i <= HalfSubdiv *)
         For[i = i, i <= HalfSubdiv, i = i + 1,
          Result[[i]][[j]] = 1;
```

```
          ]
         ]
        ],
       (* else jMin <= jMax *)
       For[j = jMin, j <= jMax, j = j + 1,
        i =
         HalfSubdiv +
          Round[HalfSubdiv*((X2 - X1)*(j - jMin)/(jMax - jMin) +
              X1)] + 1;
        If[i > HalfSubdiv,
         For[i = i, i >= HalfSubdiv, i = i - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else i <= HalfSubdiv *)
         For[i = i, i <= HalfSubdiv, i = i + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ]
       ],
      (* else Abs[Slope]<=1 *)
      If[iMin > iMax,
       For[i = iMax, i <= iMin, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y1 - Y2)*(i - iMax)/(iMin - iMax) +
              Y2)] + 1;
        If[j > HalfSubdiv,
         For[j = j, j >= HalfSubdiv, j = j - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else j <= HalfSubdiv *)
         For[j = j, j <= HalfSubdiv, j = j + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ],
       (* else iMin <= iMax *)
       For[i = iMin, i <= iMax, i = i + 1,
        j =
         HalfSubdiv +
          Round[HalfSubdiv*((Y2 - Y1)*(i - iMin)/(iMax - iMin) +
              Y1)] + 1;
        If[j > HalfSubdiv,
         For[j = j, j >= HalfSubdiv, j = j - 1,
          Result[[i]][[j]] = 1;
          ],
         (* else j <= HalfSubdiv *)
         For[j = j, j <= HalfSubdiv, j = j + 1,
          Result[[i]][[j]] = 1;
          ]
         ]
        ]
       ]
      ]
     ];
   Result
   ];

(* Function to calculate the change in temperature using Newton's Law \
of Heating and Cooling *)
HeatTransfer[\[Beta]_, T0_, dx_, dy_, dz_, AreaGrid_,
   Temperatures_] := \[Beta]/(dx dy dz)*
     AreaGrid*(T0 - Temperatures) /. Indeterminate -> 0.0 /.
   ComplexInfinity -> 0.0;

(* Function to calculate the change in temperature due to diffusion, \
using a FTCS method *)
Diffusion[\[Alpha]_, dx_, dy_, dz_, Stencil_, Temperatures_] :=
  Table[Table[Table[
          \[Alpha]*(
       If[
          i > 1 && i < Dimensions[Temperatures][[2]] &&
           Stencil[[i + 1]][[j]] == 1 && Stencil[[i - 1]][[j]] == 1,

          Temperatures[[k]][[i + 1]][[j]] -
           2 Temperatures[[k]][[i]][[j]] +
           Temperatures[[k]][[i - 1]][[j]], 0]/dx^2
        +
        If[j > 1 && j < Dimensions[Temperatures][[3]] &&
```

```
           Stencil[[i]][[j + 1]] == 1 && Stencil[[i]][[j - 1]] == 1,

          Temperatures[[k]][[i]][[j + 1]] -
           2 Temperatures[[k]][[i]][[j]] +
           Temperatures[[k]][[i]][[j - 1]], 0]/dy^2
       + If[k > 1 && k < Dimensions[Temperatures][[1]],

          Temperatures[[k + 1]][[i]][[j]] -
           2 Temperatures[[k]][[i]][[j]] +
           Temperatures[[k - 1]][[i]][[j]], 0]/dz^2
      ),
     {j, 1, Dimensions[Temperatures][[3]]}],
    {i, 1, Dimensions[Temperatures][[2]]}],
   {k, 1, Dimensions[Temperatures][[1]]}];

(* The simulation method *)
(* n: number of sides *)
(* \[Alpha]: Diffusion parameter *)
(* \[Beta]: Pan heating parameter *)
(* \[Gamma]: Air heating parameter *)
(* w: Pan width *)
(* h: Brownie height *)
(* T: Time of simulation *)
(* nx: Number of subdivisions along each horizontal axis *)
(* nz: Number of subdivisions along the vertical axis *)
(* nt: Number of time steps *)
(* Temp: Oven/pan temperature *)
Simulation[n_, \[Alpha]_, \[Beta]_, \[Gamma]_, w_, h_, T_, nx_, nz_,
   nt_, Temp_] := Module[
   {A, P, dw, dh, dt, Stencil, Border, AreaGrid3D, AreaGrid3DTop,
    TempSums, Temperatures, i},
   (* Calculate size of spatial and time steps *)
   dw = w/nx;
   dh = h/nz;
   dt = T/nt;

   (* Calculate the correct area and perimeter *)
   A = 1/4 n w^2 Sin[(1 - (n - 2)/n) \[Pi]/
       2] Cos[(1 - (n - 2)/n) \[Pi]/2];
   P = n w Sin[(1 - (n - 2)/n) \[Pi]/2];

   (* Generate stencil and border tables *)
   Stencil = CalculateStencil[n, nx];
   Border = CalculateBorder[n, nx];

   (* Use 2D stencil and border to Calculate 3D surface area tables \
for the pan and the air *)
   AreaGrid3DTop =
    Join[Table[
      Table[Table[0.0, {y, 1, nx}], {x, 1, nx}], {z, 1,
       nz - 1}], {Stencil}];
   AreaGrid3DTop = AreaGrid3DTop*A/Total[Total[Stencil]];
   AreaGrid3D = Table[Border, {i, 1, nz}];
   AreaGrid3D = AreaGrid3D*P*h/(Total[Total[Border]]*nz);
   AreaGrid3D[[1]] = AreaGrid3D[[1]] + AreaGrid3DTop[[nz]];

   (* Setup temperature table *)
   Temperatures =
    Table[Table[
      Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
      nz}];
   TempSums =
    Table[Table[
      Table[22.4*Stencil[[i]][[j]], {j, 1, nx}], {i, 1, nx}], {k, 1,
      nz}];

   (* Advance the simulation through all the time steps *)
   For[i = 1, i <= nt, i = i + 1,
    Temperatures = Temperatures +
      (HeatTransfer[\[Beta], Temp, dw, dw, dh, AreaGrid3D,
          Temperatures] +
         HeatTransfer[\[Gamma], Temp, dw, dw, dh, AreaGrid3DTop,
          Temperatures] +
         Diffusion[\[Alpha], dw, dw, dh, Stencil, Temperatures])*dt;
    TempSums = TempSums + Temperatures;
    ];

   (* Return the final temperatures and the average temerature for \
every data point *)
   {Temperatures, TempSums/(nt + 1)}
   ];
```

Space Optimization, Fitted Heat Equation, and Weighted Evaluation

```
L = 30; W = 60; a = 81; percent = 1.08;(* constants used in pan area \
models *)
side[n_, A_] := Sqrt[(
 4 A Tan[\[Pi]/n])/n]; \[Theta][n_] := (n - 2)/n \[Pi];
apothem[n_, A_] := Sqrt[A/(n Tan[\[Pi]/n])];
radius[n_, A_] := Sqrt[(2 A)/(n Sin[(2 \[Pi])/n])];
perimeter[n_, A_] := n*side[n, A]; height[n_, A_] := \!\(\*
TagBox[GridBox[{
{"\[Piecewise]", GridBox[{
{
RowBox[{"2",
RowBox[{"apothem", "[",
RowBox[{"n", ",", "A"}], "]"}]}]},
RowBox[{
RowBox[{"Mod", "[",
RowBox[{"n", ",", "2"}], "]"}], "==", "0"}]},
{
RowBox[{
RowBox[{"apothem", "[",
RowBox[{"n", ",", "A"}], "]"}], "+",
RowBox[{"radius", "[",
RowBox[{"n", ",", "A"}], "]"}]}],
RowBox[{
RowBox[{"Mod", "[",
RowBox[{"n", ",", "2"}], "]"}], "==", "1"}]}
},
AllowedDimensions->{2, Automatic},
Editable->True,
GridBoxAlignment->{
         "Columns" -> {{Left}}, "ColumnsIndexed" -> {},
          "Rows" -> {{Baseline}}, "RowsIndexed" -> {}},
GridBoxItemSize->{
         "Columns" -> {{Automatic}}, "ColumnsIndexed" -> {},
          "Rows" -> {{1.}}, "RowsIndexed" -> {}},
GridBoxSpacings->{"Columns" -> {
Offset[0.27999999999999997`], {
Offset[0.84]},
Offset[0.27999999999999997`]}, "ColumnsIndexed" -> {}, "Rows" -> {
Offset[0.2], {
Offset[0.4]},
Offset[0.2]}, "RowsIndexed" -> {}},
Selectable->True]}
},
GridBoxAlignment->{
      "Columns" -> {{Left}}, "ColumnsIndexed" -> {},
       "Rows" -> {{Baseline}}, "RowsIndexed" -> {}},
GridBoxItemSize->{
      "Columns" -> {{Automatic}}, "ColumnsIndexed" -> {},
       "Rows" -> {{1.}}, "RowsIndexed" -> {}},
GridBoxSpacings->{"Columns" -> {
Offset[0.27999999999999997`], {
Offset[0.35]},
Offset[0.27999999999999997`]}, "ColumnsIndexed" -> {}, "Rows" -> {
Offset[0.2], {
Offset[0.4]},
Offset[0.2]}, "RowsIndexed" -> {}}],
"Piecewise",
DeleteWithContents->True,
Editable->False,
SelectWithContents->True,
Selectable->False]\);
MagicNumberApothem[n_, A_] := Sqrt[(4 Cot[\[Pi]/n])/n]*Sqrt[A] // N;
MagicNumberRadius[n_, A_] :=
 Sqrt[2/(n Sin[(2 \[Pi])/n])]*Sqrt[A] //
  N; (* This section is dedicated to finding the different dimensions \
of a regular polygon including sides,apothem,and radius. We are also \
able to find the magic numbers for apothem and radius depending on \
the shape and area of a certain polygon. *)
numberofcols[s_, A_, l_, w_, p_] :=
 Floor[1/(2 apothem[s,
    p A])] (* calculate number of columns for even non-multiple of 4 \
n-gons *)
evenrow[s_, A_, l_, w_, p_] :=
```

```
 If[l - 2 numberofcols[s, A, l, w, p] apothem[p s, A] -
    apothem[p s, A] >= 0, True,
  False] (* checks to see if it is possible to fit another n-gon in \
every other row *)
numberofrows[s_, A_, l_, w_, p_] :=
 Do[i = -1; x = w;
  While[If[EvenQ[i], x, x - side[s, p A]/2] >= 0,
   x = x - If[OddQ[i], 2*side[s, p A], side[s, p A]]; i++];
  Return[i], {1}] (* finds number of rows for even non-multiple of 4 \
n-gons in tessellating pattern *)
numberofhexs[s_, A_, l_, w_, p_] :=
 If[evenrow[s, A, l, w, p] == True,
  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p],
  numberofrows[s, A, l, w, p]*numberofcols[s, A, l, w, p] -
   Floor[numberofrows[s, A, l, w, p]/
    2]] (* calculates the number of hexagons for a particular w and l \
oven *)
latticearrange1[s_, A_, l_, w_] :=
 Floor[w/If[IntegerQ[s/4], height[s, A] // N, (2 radius[s, A])]]*
  Floor[l/height[s, A]] (* first lattice arrangement of n-gons *)
latticearrange2[s_, A_, l_, w_] :=
 Floor[w/height[s, A]]*
  Floor[l/If[IntegerQ[s/4],
     height[s, A] //
      N, (2 radius[s, A])]] (* second lattice arrangement of n-gons *)


hexadaptarrange1[s_, A_, l_, w_, p_] :=
 numberofhexs[s, A, l, w,
  p] (* first tessellating arrangement of n-gons *)
hexadaptarrange2[s_, A_, l_, w_, p_] :=
 numberofhexs[s, A, w, l,
  p] (* second tessellating arrangement of n-gons *)
(* calculates number of n-gons that can fit in a particular w and l \
rectangular oven *)

numberofpolys[s_, A_, l_, w_, p_] :=
 If[s != 6 && s != 4,
  If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
   latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]],

  If[s != 4,
   If[
    If[hexadaptarrange1[s, A, l, w, p] >=
       hexadaptarrange2[s, A, l, w, p],
      hexadaptarrange1[s, A, l, w, p],
      hexadaptarrange2[s, A, l, w, p]] >=
     If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
      latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]],

    If[hexadaptarrange1[s, A, l, w, p] >=
      hexadaptarrange2[s, A, l, w, p],
     hexadaptarrange1[s, A, l, w, p], hexadaptarrange2[s, A, l, w, p]],

    If[latticearrange1[s, A, l, w] >= latticearrange2[s, A, l, w],
     latticearrange1[s, A, l, w], latticearrange2[s, A, l, w]]],

   If[latticearrange1[s, p A, l, w] >= latticearrange2[s, p A, l, w],
    latticearrange1[s, p A, l, w],
    latticearrange2[s, p A, l,
     w]]]] (* calculates number of n-gons that can fit in a \
particular w and l rectangular oven *)
Eff[s_, A_, w_, l_, p_] := (numberofpolys[s, A, l, w, p]*A)/(
 l*w) (* calculates the oven space efficiency for one shelf for \
particular n-gon and rectangular oven *)
(*Effheat[s_,A_]:=.0114944*s+.340489/Sqrt[A]+.801163*)
Effheat[s_, A_] :=
 Min[1, .055657869822367416*
    ArcTan[.2259800337169932*s -
      1.6457785931252638] + .10716283573624863/(
   Sqrt[A] - 2.462771976211951) + .9045771916045253]

ranknumber[shaperank_, heatrank_, w_] := shaperank w + (1 - w) heatrank

Effgenerate2[A_, l_, w_, p_, weight_, polylist_] :=
 Do[polylistlocal = Sort[polylist];
  If[Length[polylistlocal] < 6, Break[]];

  rank = {}; spatialeff = 0;
  For[gg = 1, gg <= Length[polylistlocal], gg++,
   rank = Append[
```

```
    rank, {polylistlocal[[gg]],
     Eff[polylistlocal[[gg]], A, l, w, p] // N}]];

  spatialeff = RankedMax[Flatten[rank], Length[rank] + 1];
  For[o = 1, o <= Length[rank], o++, rank[[o]][[2]] /= spatialeff];

  SortBy[rank, Last];
  For[qq = 1; heateff = {}, qq <= Length[rank], qq++,
   heateff = Append[heateff, Effheat[rank[[qq]][[1]], A] // N]];

  For[ss = 1; shape = {}; sideorder = {}, ss <= Length[polylistlocal],
    ss++, sideorder = Append[sideorder, rank[[ss]][[1]]];
   shape = Append[shape,
     ranknumber[rank[[ss]][[2]], heateff[[ss]], weight]]];

  Return[sideorder[[Part[Flatten[Position[shape, Max[shape]]], 1]]]
  , {1}]

PrintResults[A_, l_, w_, p_, weight_, polylist_] :=
 Print["The most efficient pan based on the parameters is a ",
  Effgenerate1[A, l, w, p, weight, polylist], "-sided polygon"]
```