
Jihan F. Cepeda Jimenez
University of Nebraska-Lincoln, jihan.cepeda@gmail.com

Follow this and additional works at: http://digitalcommons.unl.edu/biosysengdiss
Part of the Biological Engineering Commons, and the Life Sciences Commons

http://digitalcommons.unl.edu/biosysengdiss/15

This Article is brought to you for free and open access by the Biological Systems Engineering at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in Biological Systems Engineering--Dissertations, Theses, and Student Research by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.
MODELING HEAT TRANSFER DURING COOLING OF READY-TO-EAT MEAT AND POULTRY PRODUCTS USING THREE-DIMENSIONAL FINITE ELEMENT ANALYSIS AND WEB-BASED SIMULATION

by

Jihan F. Cepeda Jimenez

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Master of Science

Major: Agricultural and Biological Systems Engineering

Under the Supervision of Professor Curtis L. Weller

Lincoln, Nebraska

December, 2010
The meat industry is required to comply with processing performance standards for preventing the growth of foodborne pathogens in products. These performance standards, established by the United States Department of Agriculture - Food Safety and Inspection Service (USDA-FSIS) require a reduction of *Salmonella spp* (lethality standard) and limit the growth of sporeforming bacteria (stabilization standard) in certain processed meat products. In general, strategies used to comply with these standards are associated with thermal processing. Meat processors have difficulties complying with these performance standards. Moreover, thermal processing deviations are an issue in the meat industry that generate uncertainty regarding the safety of finished products. When thermal processing deviations occur, the USDA-FSIS recommends the use of computer models (i.e. heat transfer and microbial growth predictive models) as tools to evaluate the severity of the deviation. The objective of this study was to develop a heat transfer model for simulating cooling of cooked irregular-shaped, ready-to-eat meat and poultry products. The developed heat transfer model considered conduction as the governing
equation, subject to combined convection, radiation and evaporation boundary conditions. A three-dimensional finite element algorithm implemented in Java™ (Version 6, update 23, Sun Microsystems, 2010) was used to solve the model. Model validation was conducted using data collected in four different meat processing facilities, under real time-varying processing conditions. The model was adapted to receive input parameters that are readily available and can easily be provided by meat processors such as air relative humidity, air temperature, air velocity, type of casing, duration of water shower, and product weight and core temperature prior to entering the chiller. The mean deviation between the observed and predicted values was 1.2 °C for core temperatures; 1.7 °C for temperatures 5.08 cm from core; and 2 °C for surface temperatures. The developed heat transfer model can be integrated with predictive microbiology models; which can be particularly useful for evaluating the severity of thermal processing deviations caused by unexpected processing disruptions. This integration can be the foundation for open source software packages which can serve as quantitative tools to support food safety management in the meat industry.
Para mis padres (Jorge y Gilma) y mis hermanos,
a quienes les debo todo lo que soy y seré...
ACKNOWLEDGEMENTS

I would like thank to my advisor, Dr. Curtis Weller, and my committee members, Dr. Mehrdad Negahban, Dr. Harshavardhan Thippareddi and Dr Jeyam Subbiah, for their unconditional help, advice and support all these years. Dr Weller, thank you very much for all your helpful guidance, patience and understanding; especially during the last part of my Master's program. Dr Thippareddi, thanks for all your support and for being the bridge to the meat industry. Dr Negahban, thank you so much for your valuable inputs and for giving me the tools to complete the finite element analysis, which was vital for this project. Dr Subbiah, thanks for your criticism and ideas which always led me to improve my work. Also, thanks to Dr Alejandro Amézquita for developing the foundation of my research.

Special thanks to Dr John Rupnow for his wonderful friendship and support, and for providing me the opportunity to come for the first time to the University of Nebraska-Lincoln. Also, to my professors from my hometown university, Universidad de La Sabana, for raising me up as an integral engineer. Exceptional thanks to MS Leyla Garcia for teaching me Java programming.

I would like to thank Terry Bartels, Garret Coffman, Laura Hayes, Vishal Singh, Robert Weber and James Nau for their technical support. I am grateful with the meat processors Mark Blauert, Tim Hagedorn, Steve Leitch, Julie Butler, Andy Budell, Chad Lottman, Marla Gustafson, Alice Muñoz, Nancy Kent, Dawn Langhoff and Gene
Bartholomew who allowed me to visit their meat processing facilities to conduct the validation of my research.

Extraordinary thanks to Roberto Jiménez and Andrés Doblado for their unlimited patience, friendship and help during the validation and writing of my research manuscript. Also, I would like to thank to my family in Nebraska, Dee and Roberto, for their love and company, and for cheering me up in difficult moments.

Finally, thanks to the USDA-IREE-National Integrated Food Safety Initiative Competitive Grants Program for funding a portion of my research project.

…Gracias a mi familia por hacer hasta lo imposible por mi bienestar, por entender mi ausencia durante estos últimos años, y por su invaluable apoyo en la distancia…
This MS thesis consists of three chapters and two appendices. Chapter I contains a review of literature of current research efforts related to heat transfer modeling in meat products that have been published in the last two decades. It synthesizes the main ideas behind modeling of thermal processing in the meat industry; encompassing common considerations and techniques. This review chapter specially emphasizes on research efforts that have been oriented to industrial applications. Chapter II presents the definition of a heat transfer model for simulating time-varying temperature distributions during thermal processing of ready-to-eat meat and poultry products. Also, a step-by-step three-dimensional transient finite element analysis is presented as a tool to solve the model without the need for proprietary software. Additionally, Chapter III presents the validation of the heat transfer model defined in Chapter II as conducted in different meat processing facilities. This chapter also describes model adaptations that must be taken into account to provide good cooling profile estimates using only parameters that can be provided by a meat processor. In addition, potential applications of the model and future research efforts are discussed in the chapter. Appendix A includes experimental data collected during the validation of the model in different meat processing facilities. Finally, Appendix B includes the documented code of the computer program written in Java™ to solve the model.
# TABLE OF CONTENTS

## CHAPTER I. Modeling Heat and Mass Transfer in Ready-to-Eat Meats: A Review

ABSTRACT ................................................................................................................................. 2

1. INTRODUCTION ............................................................................................................. 3

2. MODELING HEAT TRANSFER IN READY-TO-EAT MEATS ........................................... 4

  2.1 Thermal Processing in the Meat and Poultry Industry .............................................. 4

  2.2 Heat Transfer Modeling in Meat Products .................................................................. 8

    2.2.1 The Fourier and Non-Fourier Controversy ...................................................... 8

    2.2.2 Boundary Conditions ...................................................................................... 9

    2.2.3 Common Simplifications ............................................................................. 10

  2.3 Importance of Numerical Analysis ......................................................................... 11

  2.4 Models Developed for Meat Products ................................................................... 12

  2.5 Models with Potential Industrial Application ....................................................... 16

  2.6 New Modeling Approaches .................................................................................. 19

3. MODELING HEAT AND MASS LOSS DUE TO EVAPORATION .................................. 23

  3.1 Traditional Approaches ....................................................................................... 24

  3.2 Modeling Water Transport .................................................................................. 24

4. MODELING THERMAL PROPERTIES .......................................................................... 26

  4.1 Air Properties .................................................................................................... 26
CHAPTER II. Modeling Heat Transfer during Cooling of Cooked Ready-to-Eat Meats using Three-Dimensional Finite Element Analysis

1. INTRODUCTION .................................................................................................. 41
2. THREE-DIMENSIONAL FINITE ELEMENT ANALYSIS ........................................ 46
   2.1 Model Definition ............................................................................................. 46
      2.1.1 Governing Equation ................................................................................. 46
      2.1.2 Initial Conditions ...................................................................................... 49
      2.1.3 Boundary Conditions ................................................................................ 50
      2.1.4 Estimation of Thermal Properties ............................................................. 54
   2.2 Weak Formulation ........................................................................................... 59
   2.3 Discretization .................................................................................................. 61
      2.3.1 Domain Discretization Using Three-dimensional Elements ...................... 65
      2.3.2 Boundary Discretization Using Two-dimensional Elements ...................... 69
   2.4 Element Equations ........................................................................................... 71
CHAPTER III. Modeling Heat Transfer during Cooling of Ready-to-Eat Meats: Validation in Meat Processing Facilities

1. INTRODUCTION ........................................................................................................ 98
2. MATHEMATICAL MODEL ..................................................................................... 102
   2.1 Governing Equation .................................................................................... 102

3. SUMMARY ............................................................................................................ 84
4. ACKNOWLEDGEMENTS .................................................................................... 87
5. REFERENCES ..................................................................................................... 88
NOMENCLATURE ...................................................................................................... 90
2.2 Initial Conditions ........................................................................................... 102
2.3 Boundary Conditions ..................................................................................... 103
2.4 Estimation of Thermal Properties .................................................................. 103
2.5 Estimation of Heat Transfer Coefficient ....................................................... 104

3. MATERIALS AND METHODS .......................................................................... 105
3.1 Data Collection Process ................................................................................. 105
  3.1.1 Data Acquisition System and Instrumentation ........................................ 105
  3.1.2 Instrumentation ...................................................................................... 105
3.2 Validation in Meat Processing Facilities ........................................................ 107
  3.2.1 Meat Products ........................................................................................ 107
  3.2.2 Operating Conditions for Cooling ........................................................... 108
3.3 Numerical Solution ........................................................................................ 110
3.4 Meshing ........................................................................................................ 111
3.5 Determination of Probe Locations ................................................................. 112
3.6 Evaluation of Model Performance ................................................................. 113
3.7 Initial Temperature Distribution ..................................................................... 113

4. RESULTS AND DISCUSSION ........................................................................... 114
4.1 Meshing and Simulation Time ....................................................................... 114
4.2 Initial Temperature Distribution ................................................................... 120
4.3 Analyzing the Effect of Evaporation ............................................................. 123
  4.3.1 Effect of chiller temperature ................................................................. 124
APPENDICES

APPENDIX A - Model Validation Results and Experimental Data ............................................. 151

APPENDIX B - Three-dimensional Transient Finite Element Algorithm in Java™ .................. 166
LIST OF FIGURES

CHAPTER II. Modeling Heat Transfer during Cooling of Cooked Ready-to-Eat Meats using Three-Dimensional Finite Element Analysis

Figure 2.1 Conservation of energy within a control piece of ready-to-eat meat product. 47
Figure 2.2 Cross-sectional view of the non-uniform initial temperature distribution (SI units) of ellipsoidal shaped meat product prior entering to the chilling chamber. 50
Figure 2.3 Thermal energy transfer from product surface to its surroundings due to convection, evaporation and radiation. 50
Figure 2.4 Weak form and finite element functions. 61
Figure 2.5 Three-dimensional mesh and its elements. 62
Figure 2.6 Mesh generation for products with complex geometries using CT-scanning. 64
Figure 2.7 Linear tetrahedral element defined by four local nodes. 65
Figure 2.8 Two-dimensional triangular element defined by three local nodes. 69
Figure 2.9 Global system of equations. 80
Figure 2.10 Reorganizing global system of equations to generate the transient solution by using equations 2.83-4. 82
Figure 2.11 Reorganizing global system for the solving step. 83
CHAPTER III. Modeling Heat Transfer during Cooling of Ready-to-Eat Meats: Validation in Meat Processing Facilities

Figure 3.1 Example of equipment layout used during the validations ......................... 107

Figure 3.2 Location of thermocouples in a meat product using software for visualization of medical images .................................................................................................................................................................................. 112

Figure 3.3 Cross-sectional view of three-dimensional meshes created for validation Test#2-5: This meshes were generated from CT-scans using Materialise Mimics 14.0 and 3-matic v5.1 .................................................................................................................................................................................................... 116

Figure 3.4 Cross-sectional view of three-dimensional meshes created for validation Test#1, Test#6 and Test#7 .............................................................................................................................................................................. 117

Figure 3.5 Product geometry approximation ................................................................................................................................. 118

Figure 3.6 Potential prediction deviations caused by using coarser meshes with the method established to determine temperatures at arbitrary points (i.e. probes locations) ............................................................................................................................ 119

Figure 3.7 Illustration of the initial heat transfer simulation procedure used to estimate the initial temperature distribution of the meat products prior entering the cooler .......... 123

Figure 3.8 Illustration of variation between the temperature of the air surrounding the product and the chiller air temperature for Test#6 .................................................................................................................................................. 124

Figure 3.9 Illustration of effect of different air temperature profiles (i.e. chiller air temperature and boundary air temperature) provided to the model during simulations that included the effects of evaporation ........................................................................................................................................................................... 126

Figure 3.10 Comparison between predicted and observed temperatures at different locations for Test#1; considering water evaporation during the simulation ................................................................. 128
Figure 3.11 Comparison between predicted and observed temperatures at different locations for Test#1; evaporation was not considered in the simulation.........................129

Figure 3.12 Comparison between predicted and observed temperatures at different locations for Test#7; considering water evaporation during the simulation.................131

Figure 3.13 Comparison between predicted and observed temperatures at different locations for Test#7; evaporation were not considered in the simulation. .....................132

Figure 3.14 Evaluation of model performance during different validation tests and locations. .....................................................................................................................135

Figure 3.15 Model prediction deviations and estimated growth of *Clostridium perfringens* during cooling for Test#1.................................................................138

Figure 3.16 Model prediction deviations and estimated growth of *Clostridium perfringens* during cooling for Test#6.................................................................140

Figure 3.17 Simulation of cooling deviation in Test#6 caused by a supposed 1 h cooler downtime.................................................................142

Figure 3.18 Simulation of cooling deviation in Test#6 caused by a supposed 6 h cooler downtime.................................................................143

Figure 3.19 Simulation of cooling deviation in Test#6 caused by a supposed chiller overload.................................................................144

Figure 3.20 Illustration of visual interface of a prototype version of the website *numodels4safety.unl.edu*.................................................................146
LIST OF TABLES

CHAPTER II. Modeling Heat Transfer during Cooling of Cooked Ready-to-Eat Meats using Three-Dimensional Finite Element Analysis

Table 2.1 Specific heat of air .................................................................55
Table 2.2 Thermal properties of major pure components of foods. .........................56
Table 2.3 Gaussian integration points and weights for triangular and tetrahedral regions 77
Table 2.4 Integration formulae for triangular and tetrahedral regions .....................79

CHAPTER III. Modeling Heat Transfer during Cooling of Ready-to-Eat Meats: Validation in Meat Processing Facilities

Table 3.1 Description of type of products used for the model validation in different meat processing facilities. .................................................................108
Table 3.2 Operating conditions used for the validation tests.................................110
Table 3.3 Number of elements and low quality elements enclosed in the meshes used during the simulations.........................................................115
Table 3.4 Decision making table used to adapt the model to the inputs that can be provided by a meat processor.....................................................134
CHAPTER I

Modeling Heat and Mass Transfer in Ready-to-Eat Meats: A Review

This paper will be published as:

Cepeda, J.F.; Thippareddi, H.; Subbiah, J.; Weller, C.L. 2010. Modeling Heat Transfer in Ready-to-Eat Meats: A Review. It will be submitted to:

- Food Engineering Reviews
Modeling Heat Transfer of Ready-to-Eat Meats: A Review

ABSTRACT

Temperature is an important factor affecting microbial growth in meat products, and hence the most controlled and monitored parameter in the meat industry. In the last few decades, modeling of heat transfer in products has gained special attention in the meat industry as it can be integrated with predictive microbial growth/death models and eventually with risk assessment models. Hence, heat transfer models can be used as a practical tool to assess microbial safety of meat products quantitatively; especially in the event of unexpected processing issues such as thermal processing deviations. This manuscript reviews current research efforts related to heat transfer modeling in meat products that have been published in recent years. It synthesizes the main ideas behind modeling of thermal processing in the meat industry; encompassing common considerations and techniques. This review specially emphasizes research efforts that have been oriented to industrial applications. Moreover, literature indicates that despite great advances in the field, there are several challenges that still persist and the scientific community must address them to develop models applicable to the meat industry.
1. INTRODUCTION

Meat products are highly perishable, allow rapid microbial growth unless stored and processed appropriately. Meat processors must control the growth of bacteria such as Brochothrix thermosphacta, Pseudomonas spp. and lactic acid bacteria; as these bacteria may cause spoilage in products. Moreover, they must strictly control the growth or eliminate certain foodborne pathogens such as Listeria monocytogenes, Salmonella spp. and E. coli O157:H7 (McDonald and Sun, 1999). The meat industry and the regulatory agencies relied on sampling and have inspection to assure microbiological safety of their products. However, these strategies are often impractical and may not assure safety of the products as the microorganisms are not uniformly distributed in the products.

Temperature is an important factor affecting microbial growth in meat products. Therefore, it is one of the most controlled and monitored parameters in the meat industry. In general, meat products should be chilled as fast as possible and held at low temperatures to minimize growth of foodborne pathogens and assure microbial safety. However, processors need to employ fast cooling rates that minimize the risk of potential microbial growth while avoiding surface freezing. Ice formation causes microstructural changes in meat products that may lead to undesired changes in product properties (e.g. color, viscosity, pH), cell dehydratation, drip loss and tissue shrinkage during thawing (Magnussen, et al., 2008). Hence, cooling of thermally processed meat products (stabilization) to meet the regulatory requirements represents a challenge for some meat processors; especially when processing products of large mass and volume (e.g. products over 4 kg or $3.6 \times 10^{-3}$ m$^3$).
In the last two decades, modeling has gained special attention in the food industry as it is considered as a practical tool that can be used to monitor meat safety quantitatively. The objective of this chapter is to review literature on heat transfer modeling of meat products. The review will provide perspective on the evolution of this field, its novel advances and current shortcomings. Also, it will provide an understanding of the state-of-the-art in this area which can be used to get a better understanding of future research opportunities.

2. MODELING HEAT TRANSFER IN READY-TO-EAT MEATS

2.1 Thermal Processing in the Meat and Poultry Industry

Thermal processing of ready-to-eat meat and poultry products refers to a broad range of products, processing techniques and equipments utilized during cooking and/or cooling. Despite this vast diversity, most of the thermal processing systems in the meat industry operate under similar physical principles of heat and mass transfer (Knipe, 2010).

A variety of thermal processing methods are currently used in the meat industry. The selection of the method of processing depends on several factors such as type of product being processed, market needs, company infrastructure, economic impact among others. For instance, cooling can be performed by water immersion, slow air flow, air blast as well as vacuum cooling. Sun and Wang (2000) studied the heat transfer characteristics of cooked meat joints. Using ellipsoidal shaped, the authors reported that vacuum cooling was a faster method for cooling compared to water immersion, blast and slow air cooling. Also, vacuum cooling resulted in a fairly homogeneous temperature
distribution of the product during the process. Conversely, this method resulted in a greater weight loss as the heat transfer during vacuum cooling was governed by water evaporation from the product surface. Greater rates of water injection during the preparation step may be used to overcome this problem (Sun and Wang, 2000). Despite the potential advantages of vacuum cooling, traditional cooling methods such as slow air flow and air blast cooling are more common in the meat and poultry industry.

Generally speaking, the manufacturing of processed ready-to-eat meats include three basic steps: meat matrix preparation, heat treatment and cooling. Although the meat matrix preparation step may differ between products, the heat treatment and cooling steps follow the same principles for most products.

During heat treatment, products are placed into ovens or smokehouses in which they come in contact with a hot fluid, usually hot air. This step serves several purposes such as heat stabilization of the meat matrix, pasteurization and fixing of meat color. Products are normally cooked to $\geq 130^\circ F$ to ensure proper heat stabilization of the meat matrix by protein denaturation (Billerbeck and Shoemaker, 1971, Rich and Rich, 1981, Wiebe and William, 1999). This cooking step is also critical for destructing foodborne pathogens and assuring microbial safety.

After the heat treatment, the meat products are usually exposed to cold water showers right before entering the cooling chamber. This procedure helps drop the surface temperature of the product; minimizing the weight loss caused by evaporation from the surface during the cooling step (Amézquita, 2004). Consequently, the temperature distribution of a meat product entering the cooler is not uniform across the product.
During the cooling step, meat products are exposed to a cold airflow; resulting in thermal energy transfer from the surface of the product to the airflow, water evaporation from the product surface and energy conduction radiating from the center of the product. Cooling is a critical step for preventing potential growth of spore-forming bacteria (i.e. *Clostridium botulinum*, *Clostridium perfringens* and *Bacillus cereus*) that can survive the traditional heat treatment applied to meat products. Heat-shocked spores of foodborne pathogens may germinate and grow if the cooling is not rapid and uniform.

The United States Department of Agriculture - Food Safety and Inspection Service (USDA-FSIS) published compliance guidelines for heat treatment and cooling of ready-to-eat meat products. These guidelines recommend certain time-temperature schedules that processors should follow in order to comply with the lethality and stabilization requirements. For instance, to meet the lethality requirement (i.e. 7 log\(_{10}\) reduction in *Salmonella* spp. in ready-to-eat poultry and a 6.5 log\(_{10}\) reduction in *Salmonella* spp. in ready-to-eat beef products), products must reach a minimum temperature of 60 °C (140 °F). This temperature must be maintained across the whole product for at least 12 minutes (Food Safety and Inspection Service, 1999b). To meet the stabilization requirement (i.e. no multiplication of toxigenic microorganisms such as *Clostridium botulinum*, and no more than a 1 log\(_{10}\) multiplication of *Clostridium perfringens* within the product), the compliance guidelines for cooling of ready-to-eat meats suggest one out of following three processing methods. The first method states that the maximum product internal temperature should drop from 54.4 to 26.6°C (130 to 80°F) within less than 1.5 hours; and must reach 4.4°C (40°F) in the next 5 hours. Thus, the total chilling operation must not exceed 6.5 hours. The second method states that
when the cooling step begins within 90 minutes after the heat treatment, the product internal temperature should be cooled from 48 to 12.7°C (120°F to 55°F) in less than 6 hours; then, the cooling should continue until the maximum product temperature reaches 4.4°C (40°F). The third method can only be used for processing of cured products containing at least 100ppm ingoing sodium nitrite. For these products, the maximum internal temperature should decrease from 54.4°C to 26.6°C (130 to 80°F) in less than 5 hours, and from 26.6 to 7.2°C (80 to 45°F) within the next 10 hours (USDA-FSIS, 1999a); resulting in up to 15 hour cooling time.

Thermal processing of ready-to-eat meats, especially large products, represents a challenge for meat processors. In fact, several small meat processors have difficulties following the guidelines as cooling rates are influenced by a variety of parameters such as operating conditions (e.g. temperature, product load, airflow velocity of the heating/cooling medium, product arrangement) and type of products (e.g. shape, dimensions and thermal properties). Therefore, thermal processing deviations do occur in the meat processing industry. These deviations imply that the thermal process did not follow the time-temperature schedules suggested in the guidelines for heat treatment and cooling of ready-to-eat meat products; generating uncertainty regarding the safety of finished products. When these deviations occur, microbial sampling might not be the best alternative to evaluate the safety of the product as microorganisms are not uniformly distributed in the product. Hence, the USDA-FSIS recommends computer modeling as a tool to determine the severity of the thermal processing deviation (USDA-FSIS, 1999a).
2.2 Heat Transfer Modeling in Meat Products

During the last few decades, several researches around the work have tried to model heat transfer in meat products. Although models published vary from analytical simple equations to complex models that require the use of numerical methods, most of the models are limited to meat products with regular shapes. A large majority of models account for conduction as the governing heat transfer phenomenon under different type of boundary conditions and several simplifications.

2.2.1 The Fourier and Non-Fourier Controversy

Heat transfer of meat products during cooling is usually assumed to be governed by conduction. This phenomenon is commonly modeled by Fourier's law. However, some researchers have presented evidence that may suggest that the nature of transient heat conduction in processed meats follows non-Fourier models rather than the traditional Fourier model. The authors suggest that the heat waves take a finite time to reach a particular point within a product; contrary to the instantaneous heat wave propagation established by Fourier (Andarwa and Basirat Tabrizi, 2010, Mitra, et al., 1995, Shen and Zhang, 2008). Also, it is believed that this phenomenon may be further noticeable at low temperatures; when the energy levels of molecules are highly reduced (Mishraa, et al., 2008). Thus, the traditional Fourier heat conduction model will include an extra term containing a finite thermal characteristic time; resulting in non-Fourier models such as Hyperbolic and Dual Phase Lag (DPL) models (Antaki, 2005). These findings have generated controversy in the field. Other publications have shown that the Fourier model is an excellent description of heat conduction as the heat wave propagation velocity is very high; thus, the finite thermal characteristic time can be neglected (Herwig and
Beckert, 2000a, Herwig and Beckert, 2000b). This controversy has not been completely settled. The Fourier model is currently used to predict heat conduction in processed meats; and non-Fourier models are under investigation.

### 2.2.2 Boundary Conditions

Heat transfer due to convection is the most common boundary used during modeling of thermal processing of meats. As it is well known, convection encompasses thermal energy movement between a fluid, in this case air, and the meat product surface. This phenomenon is usually modeled by Newton's law of cooling.

Another boundary condition frequently used is the heat loss due to evaporation. This boundary is particularly important for non-wrapped products (i.e. products not in a bag or a casing). Evaporation can be defined as the process by which available liquid water found at the meat product surface is converted into a vapor. Mass and heat transfer due to evaporation are regulated by several parameters including relative humidity of the air surrounding the product, air temperature, air velocity, air flow regime, water activity of the product, product temperature, and type of product casings (e.g. fibrous casings, collagen casings, natural casings, nets).

Some models include the effect of thermal radiation as a boundary condition. During radiation, the meat product surface emits its thermal energy by electromagnetic waves. This phenomenon can be described by the Stefan-Bolzmann law. Radiation effects are normally neglected. However, it has been shown that heat transfer due to radiation should be considered under industrial processing conditions, as its effect is comparable to the convective heat transfer phenomenon (Kuitche, et al., 1996).

### 2.2.3 Common Simplifications

The most common simplifications include: product shape simplification, in which irregular shaped products are approximated to be basic shapes such as rectangular blocks or spheres; dimensional simplification, in which the models consider heat transfer only in one or two dimensions; thermal properties simplification, in which the thermal properties of the product are assumed to be homogeneous within the product and constant over time; processing simplifications, in which the environmental factors affecting the heat transfer phenomenon such as chilling room temperature, air velocity or relative humidity are considered as constant values over time.

However, the most common simplification is the estimation of the heat transfer coefficient. Usually, heat transfer analysis is performed on a single product representing the worst case scenario of processing; rather than considering batches of products being processed at the same time. Hence, the heat transfer coefficient can be assumed to be constant around the product surface and it can be estimated by using empirical correlations. However, airflow patterns are highly irregular when processing several products together; causing local changes in the heat transfer coefficient. This is an important shortcoming of current heat transfer models as these local heat transfer coefficients are difficult to estimate.
2.3 Importance of Numerical Analysis

A precise mathematical modeling for predicting heat transfer in meat products involves several time-dependent factors such as shape of the product, thermal and physical properties of the product and airflow surrounding the product, processing method, operating conditions, and other factors. These considerations lead to a set of complex partial differential equations that need to be solved by numerical methods, as analytical solutions do not exist. Finite differences, finite element analysis and computational fluid dynamics are three of the more common numerical methods used in modeling of heat transfer.

In finite difference analysis, the product shape is approximated by a regular geometry and equations are associated with heat flow between the nodes. This method is very useful for one-dimensional models, and has limited application to products with regular shapes such as cylinders, slabs and boxes (Delgado and Sun, 2001).

Finite element analysis (FEA) is designed for products of irregular geometries. It accounts for non-uniform material composition and mixed boundary conditions. However, it is a more complex method than finite differences. FEA states that a problem involving differential equations over complex geometries can be simplified by dividing the geometry into small regions with well understood behavior, called elements. In other words, dividing the domain into simple small elements such as triangles, tetrahedrons or cubes, allows complicated differential equations to be easily solved. Hence, the behavior of a complex domain can be approximated by studying and accounting for the behavior of its simple elements (Zienkiewicz, 2005).
Computational fluid dynamics models (CFD) have become popular in recent years. These models are usually based on finite volume analysis which combines the simplicity of finite differences with the geometry flexibility of finite elements. Hence, the object is divided into small control volumes as in finite elements; but, the equations are imposed on the control volumes rather than on the mesh nodes. Nevertheless, experts in heat transfer still recommend finite element analysis combined with heat transfer coefficients calculated from empirical correlations or by computational fluid dynamics simulations; as the most practical methods to solve complex heat transfer models in food products oriented to industrial applications (Pham, 2002).

### 2.4 Models Developed for Meat Products

Singh et al. (1984) developed a heat and mass transfer model for oven roasting of meat. This model considered heat transfer due to conduction with an evaporative boundary condition, and a constant initial temperature. Also, it considered constant values for thermal and physical properties of the meat product. It was solved using two-dimensional axis-symmetric finite differences on a rectangular slab of meat (Singh, et al., 1984). The authors stated that the energy required to melt and solidify fats can be neglected as it is a relatively small fraction of the total energy utilized during the cooling/heating process (Singh, et al., 1984).

Kuitche (1996) developed a model for temperature and weight loss kinetics during chilling of cylindrical meat products based on analytical solutions. It was one of the first models to account for time variable meat processing conditions including relative
humidity and chilling room temperature. In addition, it accounted for a non-uniform initial temperature distribution; and for evaporation, convection and radiation boundary conditions (Kuitche, et al., 1996).

Mallikarjunan et al. (1994) developed a two-dimensional finite element model of heat and mass transfer during beef carcass chilling. This conduction model considered a uniform initial condition; in addition to convection and evaporation boundary conditions. It also considered the empirical correlations of Choi and Okos to describe the variation in the thermal properties as functions of temperature and composition. As this was a two-dimensional model, the beef carcasses were divided into five sections and the finite element analysis was performed on the cross-sectional areas; neglecting heat transfer in the vertical axis (Mallikarjunan and Mittal, 1994). The importance of this model lies in the fact that it was one of the first models accounting for non-basic geometries (i.e. geometries different to circles, spheres, slabs, ellipses). Although the analysis was simplified to two-dimensional geometries, the cross-sections of a beef carcass correspond to irregular complex geometries. Hence, the meshes, a collection of nodes and elements that describe the geometry of the product, are no longer easy to obtain. The mesh generation process for this model was achieved by using a geographical information system; an innovative idea to solve the common meshing issues found in modeling meat products.

Hu and Sun (2000) used computer fluid dynamics to simulate cooling rates and weight loss during chilling of a cylinder-shaped pork product. According to their publication, the CFD algorithm required high computational power (i.e. 60 hours of processing within the technology existed at that time) due to the mesh complexity
required by CFD to generate a good prediction. Thus, they divided the algorithm in different steps which reduced the simulation time to 4.5 hours. In the first step, they studied the flow field in steady-state, followed by calculation of average convective heat transfer coefficient that accounts for convection, radiation and evaporation on the surface of the product. Finally, the authors simulated the heat and mass transfer within the product assuming the homogeneous heat transfer coefficient calculated in the previous step. The model validation was performed by comparing the simulation results with values obtained in an experimental air-blast chiller. It was shown that there was a large quantity of heat removed by evaporation, which was greater than the heat loss by convection. Also, they reported that the heat loss by radiation was relatively small compared with the heat loss caused by evaporation and conduction (Hu and Sun, 2000).

In a subsequent study, Sun and Hu (2002) analyzed the effects of several parameters on the accuracy of a heat and mass transfer model for simulation of vacuum cooling in a cylindrical ham. They reported that radiation and anisotropy considerably affected the cooling process. Also, they analyzed the effect of thermal shrinkage and water diffusion; concluding that these factors did not have a significant effect during vacuum cooling (Sun and Hu, 2002).

Wang and Sun (2002) used three-dimensional finite element analysis to model heat transfer during air blast cooling. The analysis was implemented using a C++ algorithm. Although this model considered three-dimensional heat transfer and tetrahedral elements; it only simulates heat transfer for large brick-shaped products. They studied the effect of air velocity; concluding that higher air velocities may reduce the cooling time. However, if the air velocity increases to very high values (i.e 5-7 m/s),
the surface temperature reaches the chilling room temperature very fast and the cooling rate will be only controlled by conduction. Hence, as the thermal conductivity of meats is low, further increments in air velocity may not longer reduce the cooling time (Wang and Sun, 2002a).

Wang and Sun (2002 and 2003) compared the performance of slow air, air blast, vacuum, and water immersion methods for cooling of meat products. Considering brick-shaped and ellipsoidal-shaped products. Hence, they performed axis-symmetric finite element analysis using C++. For the simulation, they used two-dimensional triangular elements for the ellipsoidal-shaped products; and three-dimensional tetrahedral elements for the brick-shaped products. The meshes for these products were constructed automatically; according to empirical correlations of product geometric dimensions versus weight. The model validation indicated that the predicted core temperatures were in agreement with values derived from experimental chillers; as the maximum deviations were within ±2.9°C. They concluded that vacuum cooling provided the highest cooling rates; but also the highest weight loss (Wang and Sun, 2003, Wang and Sun, 2002b).

Marcotte et al. (2008) modeled two-dimensional axis-symmetric heat transfer in finite cylinder-shaped bologna sausages. In their model, heat conduction was considered within the product in addition to a convective heat flux on the product surface. Also, thermophysical properties of the meat matrix were assumed as constant values; and the smokehouse temperature was uniform with respect to the location. This model was built in Visual Basic. Moreover, it was integrated with models of thermal destruction of microorganisms (i.e. Escherichia coli, Salmonella seftenberg, Listeria monocytogenes and Enterococcus faecalis). The model was in good agreement with the observed
experimental values (Marcotte, et al., 2008a). However, further details about the methodology used to solve and validate the model were not provided.

Santos (2008) developed a transient three-dimensional finite element model to simulate heat transfer in spheres, cylinders and simple irregular geometries. The meshing process was performed using mesh generation packages such as DistMesh. This algorithm was implemented in Matlab; and considered convective boundary conditions. However, it assumed non time-varying thermal properties and processing conditions. This model was validated by comparing predicted solutions against analytical solutions for spheres and cylinders. Also, the software was used to simulate heat transfer during cooking of a piece of meat. The output of this simulation was compared with a numerical solution generated by commercial software resulting in good agreement (Santos, et al., 2008).

2.5 Models with Potential Industrial Application

Davey and Pham (1997) developed a model for predicting heat load and weight loss during chilling of beef carcasses. This model was one of the first models validated at the industrial level. They used a finite difference approach with no axial heat flow, in which the irregular carcass geometry was approximated by a combination of seven cylinders and slabs. Also, their model allows for time-variable chiller conditions. They considered the carcass as a homogeneous material in which thermal properties are independent of temperature. Thermal properties were calculated based on the lean muscle, fat and bone composition by using the Levy Model, literature data and some
experimental measurements (Davey and Pham, 1997). A water activity of 0.85 was assumed to account for surface dryness due to water evaporation. In their publications, Davey and Pham concluded that the prediction accuracy of heat transfer models for the meat industry was limited due to the numerous changing factors observed in the industry. In a subsequent study, they solved the previous model using two-dimensional triangular finite element analysis on a cross-section of half beef carcass (Davey and Pham, 2000). The finite element analysis provided a better prediction compared to the finite difference analysis. This difference may be attributed to the fact the finite element analysis allowed for a better approximation of the carcass geometry. A novel feature of this model was that it assumed a heat transfer coefficient that accounts for the additional resistance generated by an exterior thick layer of fat commonly present in beef carcasses.

Amezquita et. al. (2004 and 2005) developed one of the first heat and mass transfer models to simulate cooling of ellipsoidal-shaped meat products in the meat industry. The model was solved in Matlab; using two-dimensional axis-symmetric finite element analysis and considered the effects of conduction, convection, evaporation and radiation during the cooling process. Moreover, it took into account non-uniform initial temperature distributions, time-dependant processing conditions and dynamic thermal properties. This model was validated using data collected from the meat industry. The maximum reported deviations between the predictions and the experimental temperatures were between ±1.59 and ±2.54°C (Amézquita, 2004, Amézquita, et al., 2005a). This model was used to simulate different processing scenarios in order to determine the critical processing conditions at which USDA-FSIS compliance guidelines for cooling are difficult to follow. Findings indicate that cured products can potentially meet the
stabilization standards under reasonable processing conditions. Conversely, for non-
cured products weighting over 2.25 kg, compliance with the guidelines seems to be
unfeasible considering realistic processing conditions (Amézquita, et al., 2005a).
Moreover, this model was integrated with a predictive microbiology model to estimate
the growth of Clostridium perfringens during cooling of cooked boneless ham. The
maximum reported deviations for this integrated model were within 0.37 log10 CFU/g

Wang et al. (2006) developed a mathematical model for the validation of safe air-
blast chilling of cooked hams. This model integrated heat and mass transfer, and
dynamic temperature-dependent bacterial growth. Cooked hams were assumed as
symmetric ellipsoidal geometries. Therefore, heat transfer was modeled as a transient
axisymmetric two-dimensional heat conduction problem; with natural convection, forced
convection, radiation and evaporation boundary conditions. Moreover, this model
considered constant values for water activity (i.e. 1.0) and relative humidity. Convective
heat transfer coefficients were estimated by using the Suryanarayana (1994) empirical
correlations for horizontal ellipsoids. In addition, mass transfer coefficients were found
by using the traditional Lewis relationship between heat and mass transfer coefficients.
Thermal properties of the meat were calculated based on the composition using Choi and
Okos (1986) correlations. The model was solved by a finite element algorithm written in
C++. Model validation was accomplished by placing commercial hams (i.e. 5.4kg hams),
in an experimental air-blast cooler. Results show that the maximum deviation between
predicted and experimental core temperature was within 2.2°C, for temperatures ranging
from 10°C to 71°C (Wang, et al., 2006).
2.6 New Modeling Approaches

Computational fluid dynamics is considered as a powerful simulation tool that has a big potential to model heat transfer during thermal processing of meat products. Braeckman et al. (2007) proposed a methodology for modeling heat transfer of meat products using CFD. They proposed to perform the model validation under controlled laboratory settings using experimental grilling devices and oven chambers. Moreover, authors suggested that pre-processing steps (i.e. meshing) should be carried out in Gambit (Ansys Inc., Canonsburg, P.A.); a general pre-processor and mesh generator. Also, the heat and mass transfer analysis should be executed in Fluent (Ansys Inc., Canonsburg, P.A.), a general purpose CFD code based on finite volume analysis (Braeckman, et al., 2007).

Trujillo and Pham (2006) used Fluent (ver. 6.0), a CFD flow modeling software package to model three-dimensional heat and mass transfer during chilling of beef carcasses. This model was based on previous work developed by Davey and Pham (1997 and 2000). The three-dimensional geometries used for the simulation were built from 14 cross-sectional data; and the tetrahedral meshes were developed using Gambit. This model considered the chilling of beef carcasses as a two-phase model including an air flow phase and a meat phase. The air flow phase analysis was performed accounting for convection, velocity, turbulent kinetic energy, turbulent dissipation rate, temperature and moisture content. The meat phase analysis accounted for thermal and mass transport; considering uniform compositions and constant thermal properties. The model simulation required high computational power and processing time. Hence, the simulation was performed in three steps as described by Hu and Sun (2000). The steady-
state air flow analysis was completed in the first step; followed by a calculation of heat and mass coefficients. Finally, the unsteady-state meat phase analysis was conducted in the third step. This strategy allows considering local variations in the heat and mass transfer coefficients, temperature and water activity in the beef carcass. However, these values were calculated on steady-state basis; thus, they were considered as constant values during the transient analysis performed in the third step. According to the authors, the simulation for a regular chilling process took several days to be completed. Validations were performed in an experimental wind tunnel. Outputs from the simulation were in good agreement with experimental data. Also, results were compared with outputs from previous finite difference and finite element models (Davey and Pham, 1997, Davey and Pham, 2000). It was concluded that the CFD simulation provides similar results as the obtained by the previous models (Trujillo and Pham, 2006).

CFD allows studying the airflow patterns during thermal processing to determine spatial distributions of parameters such as air velocity, turbulence regimen, air temperature, relative humidity and convective heat transfer coefficients. Hence, CFD promises to overcome the limitations of the traditional modeling which does not consider spatial distributions of those parameters. However, CFD models, to-date, seem to be impractical to create tools that can be easily applicable to the meat industry. Because of complexity and high demand for computational power, CFD models may not be suitable for those types of applications at the present moment.

Several authors have proposed alternatives to take advantage of the potential of CFD while minimizing computational time. Pham et al. (2009) proposed the combination of finite element analysis and CFD. Hence, the idea is to use CFD to study
airflow dynamics which will facilitate the estimation of local heat transfer coefficients and water activities on the product surfaces. This approach has been used in previous studies (Hu and Sun, 2000, Page, et al., 2009, Sun and Hu, 2002, Sun and Hu, 2003). However, the difference is that the output of the CFD analysis will be used as input values for simulations of heat transfer using finite element analysis. The disadvantage of this approach is that the values of local heat transfer coefficients and water activities have to be considered as constant values during the finite element simulation; which may lead to further inaccuracies.

Pham et al. (2009) implemented this strategy to model heat and mass transfer during chilling of beef carcasses. This model considered water transport which was calculated with one-dimensional finite differences analysis using extremely fine meshes. The output of the water transport model was used as a boundary condition in the coupling heat and mass transfer model. As in previous studies presented by this author, the beef carcasses geometries were represented by a series of cross-sectional meshes. Hence, the model of heat and mass transfer during chilling was solved using two-dimensional finite element analysis. According to the authors, no longitudinal heat transfer was studied as it is considerably slower than the heat transfer in the other directions.

On the other hand, Le Page (2009) proposed an approximate empirical CFD model for estimating heat and mass transfer of stacked food products (e.g. ready-to-eat meats) during slow air cooling. This model considered heat transfer by conduction and water diffusion within the food product; water flux, and heat flux exchanged by convection at the air-product boundary layer. The first step consisted of a CFD turbulence analysis on a small user-defined air volume surrounding the products. This
analysis determined the mean air velocity, air temperature and water concentration in air at a particular time or iteration. Then, mean air velocity was used to estimate the heat transfer coefficient and water transfer coefficient from experimental correlations. Those coefficients were used to calculate the heat and water fluxes at the surface of the products. Subsequently, these heat and water fluxes were used to estimate temperature and relative humidity of the air surrounding the products. These values were then utilized to predict the temperature and water content inside each product at a particular time. This model was validated by simulating the drying of cylindrical shaped products using the CFD model and comparing the predictions with finite difference reference models (Page, et al., 2009). CFD is also an iterative method, similar to finite element analysis. Hence, the accuracy of the predictions was associated with the time-step selected for the iterations; which is often assumed to be the same value for all iterations. Le Page proposed the use of varying time-steps during the simulations; which may be a practical idea to reduce computational time. In general, use of smaller time-steps, allows for better estimation of the solution (Olson and Negahban, 2007). Nevertheless, the time-step should be selected according to the time-stability of the physical problem that is being simulated. In thermal processing of ready-to-eat meats, thermal energy movement is mostly controlled by temperature gradients. So, the heat transfer rates can be initially high and they decrease as the product is processed. Therefore, small time-steps can be used at the beginning of the simulation; with longer time-steps used towards the end of the simulation.
3. MODELING HEAT AND MASS LOSS DUE TO EVAPORATION

It is believed that moisture evaporation from meat product surfaces occurs in three stages. During the initial stage, water from a meat product surface evaporates at the same rate as from a free water surface. Then, evaporation rate from the product surface highly decreases due to surface dryness. Finally, the evaporation rate slightly increases due to progressively rewetting of the product surface. Hence, the rate of evaporation may be influenced by the internal water diffusion and transport properties of the product (Herbert, et al., 1978). However, modeling internal water transport represents a challenge as it is not well understood yet; especially in meat matrices of complex internal structures. Therefore, water transport from the product core to the surface, and water activity fluctuations are frequently neglected when modeling evaporation in meat products. Hence, the water activity on the meat surface is usually assumed to be a constant, ca. 0.95-1.0.

For chilling processes, it has been reported that approximately 2-5% of the initial weight is lost due to evaporation of water from product surfaces in unwrapped products. One of the issues arises in the fact that high airflows (i.e. 1-3m/s) are necessary to reduce processing times; as it increases heat transfer coefficients. However, high airflows also increase the mass transfer coefficients; increasing the weight loss rate (Califano and Calvelo, 1980). This loss can be reduced when meat products are rapidly cooled in high relative humidity environments; or by wrapping the products. For cooking processes, weight loss can be also reduced by raising the high relative humidity inside the oven; and by lowering the cooking temperature (Singh, et al., 1984).
3.1 Traditional Approaches

Several simplified analytical models for predicting weight loss during chilling of meat products have been developed. These models are usually based on lumped parameters and consider the effect of some operating conditions such as relative humidity and air temperature (Califano and Calvelo, 1980).

Chuntranuluck et. al. (1998) developed a model for predicting chilling times of foods when evaporative cooling is significant. They used finite difference analysis to simulate transient cooling of food products of simple shapes such as spheres, infinite slabs and cylinders. The authors showed how to account for evaporative cooling as a function of the latent heat of evaporation and the evaporative weight loss rate. They provided a method based on Antoine equation to estimate the saturation vapour pressure of water at different temperatures; which can be used to estimate the evaporative weight loss rate. Also, they presented a method for approximating the latent heat of vaporization as a function of temperature (Chuntranuluck, et al., 1998a). This approach has been adapted to several coupled heat and mass transfer models developed in recent years (Amézquita, 2004, Amézquita, et al., 2005a, Amézquita, et al., 2005b, Sun and Hu, 2003, Trujillo, et al., 2007, Wang and Sun, 2002a, Wang and Sun, 2003, Wang, et al., 2006, Wang and Sun, 2002b, Wang and Sun, 2002c, Wang and Sun, 2002d).

3.2 Modeling Water Transport

Datta (2007) presented general models to simulate heat and mass transfer in thermal processing of foods by using porous media approaches. Meat products may be
considered as capillary-porous media as they are solids having small void spaces filled with liquid. Hence, thermal processing of meats can be studied as a phenomenon involving mass and thermal energy movement through those interconnected void spaces. Using this approach, the convective and evaporative effects are evaluated as functions of total flux of vapor, air and water. These fluxes are obtained by considering the effect of gradients of pressure, density, permeability (e.g. relative permeability of water), saturation (e.g. water saturation), product porosity, diffusivity and other thermophysical properties (Datta, 2007a, b).

Trujillo et al. (2007) evaluated different methods for estimating water diffusivity in meat from drying data. They expressed water diffusivity as a function of temperature using an Arrhenius equation. The authors reported that water diffusivity in meat increases as moisture content decreased due to opening of pores and capillaries in the product structure (Trujillo, et al., 2007).

Product geometry is a very important parameter when studying heat and mass transfer in meat products. During processing, weight loss leads to product shrinkage and changes in product shapes. However, this effect is commonly neglected as the shrinkage coefficients are very small. Trujillo et al. (2007) presented a finite volume model for simulating water loss in meat samples with high shrinkage coefficients; the average error of the model was 7.66% (Trujillo, et al., 2007).

van der Sman (2007) developed a model for simulating water transport during cooking of meat with Flory-Rehner theory of rubber-elasticity. This theory helps describe water transport inside a meat matrix when the water permeability is affected by
protein denaturization and shrinkage of the product during thermal processing. The model considered evaporation of water from the product surface, internal heat conduction, convective heat transfer by water flow, dripping of water from the surface, and one dimensional heat transfer between meat and air flow. The model was validated with experimental data collected from roasting processes of rectangular pieces of beef (van der Sman, 2007).

4. MODELING THERMAL PROPERTIES

Thermophysical properties and heat and mass coefficients highly affect the model performance and the accuracy of its predictions.

There have been developed several mathematical models and empirical correlations to estimate thermophysical properties of meats. However, these correlations usually apply to specific situations; and no general models have been developed yet. Hence, the selection of a model for predicting thermal properties should be made upon the idea of obtaining approximated values with reasonable accuracy; rather than seeking exact values.

4.1 Air Properties

Air properties such as viscosity, density, thermal conductivity and specific heat are frequently calculated from correlations based on tabulated values previously reported.
Basically, these properties can be calculated as a function of air temperature (Amézquita, 2004, Kays, et al., 2005).

4.2 Heat Transfer Coefficient

The heat transfer coefficient is one of the most difficult parameters to estimate in processing scenarios due to factors such as complex geometrical structures of the products, turbulent flow, swirling and non-parallel flow, changes within the heating/cooling chamber, changes within the surface of the products, generation of flow boundary layers, etc. Typical uncertainties of ±10-20% are frequently reported in the literature (Delgado and Sun, 2001).

Different techniques can be used for estimating heat transfer coefficients. Some of the most common techniques include back-calculation from transient temperature data, surface heat flux sensor methods, mass-loss rate and psychrometric methods, empirical correlations, approximation from data reported in the literature, and computational fluid dynamics analysis. In the back-calculation method, a mathematical model is fitted to experimental data; then, the heat transfer coefficient can be back-calculated (Rahman, 2009). Mass-rate and psychrometric methods have been described by Kondjoyan and Daudin (1997). The mass-rate method considers the constant drying rate period, in which the net heat flux exchange is equals to zero. Hence, the convection heat transfer coefficient can be expressed as a function of the mass loss rate and the enthalpy of vaporization. This method is particularly useful when accounting for combined convection, radiation and evaporation effects (Kondjoyan and Boisson, 1997, Kondjoyan and Daudin, 1997)
Several commercial heat flux sensors are available that can be used for measurement of an apparent heat transfer coefficient. Thus, the coefficient will be calculated by dividing the heat flux over the temperature difference between the product surface and the air temperature. This method has been recommended in different publication as it allows measuring time-variable heat transfer coefficients. However, it does not account for evaporation effects (Carson, et al., 2006).

Williz et al. (2006) calculated experimental values of heat transfer coefficients at different locations on the surface of a fiberglass model of a complex shape meat product (i.e. side of beef carcass). They studied the effect of air velocity and turbulence intensity on the heat transfer coefficients, using a customized heat transfer sensor and a commercial heat flux sensor. Their findings show that heat transfer coefficients may highly vary between locations on the surface of irregular shaped products. Therefore, these changes should be taken into account when modeling heat transfer of irregular geometries as they cause changes in local cooling rates. Authors attributed these changes to turbulence effects; although they could not determine the exact dependence on turbulence (Willix, et al., 2006).

There are several methods available for estimating the heat transfer coefficient from empirical correlations. Equations are often presented as functions of dimensionless numbers (e.g. Nusselt, Sherwood, Reynolds and Prandtl numbers); and constants determined by regressing from experimental data. Values of the constants and dimensionless numbers may be associated with the product geometry, product surface conditions and type of flow (Delgado and Sun, 2003). Yovanovich proposed methods for estimating the characteristic dimension of three-dimensional bodies of various shapes.
Also, he suggested several correlations for calculating the characteristic dimension and Nusselt numbers under free and forced convection regimes (Yovanovich, 1987a, Yovanovich, 1987b). In addition, Davey and Pham (1997) stated that both forced and natural convection effects take place during chilling of meats. Hence, they proposed the use of Churchill's empirical equation to compute a combined natural and forced heat transfer coefficient. In this equation, the combined heat transfer coefficient was assumed to be the cube root of the sum of the natural coefficient cubed and forced coefficient cubed (Davey and Pham, 1997). Davey and Pham also included the effect of radiation in the heat transfer coefficient. For that, they calculated a combined heat transfer coefficient by adding the convective and radiative coefficients. These methods have been used to estimate the convective heat transfer coefficient in various heat transfer models developed the last few years (Amézquita, et al., 2005b, Wang and Sun, 2002a, Wang and Sun, 2002c).

Smith et al. (1971) proposed other empirical correlation for estimating the heat transfer coefficient of ellipsoidal shaped boneless ham. This correlation considers the Nusselt number as a function of Reynolds number, and a geometry index. This index is calculated from the minor and major axes corresponding to the product.

Ryland et al. (2006) calculated experimental values of convective heat transfer coefficients for an ellipsoidal meat product model. These coefficients were calculated by using two different methods including a back-calculation from transient temperature data, and the mass-loss rate method. In addition, these experimental values were compared with the empirical correlations proposed by Yovanovich and Smith (1987 a,b). According to the findings, empirical correlations stated by Yovanovich and Smith (1987...
a,b) provide a good approximation to estimate the convective heat transfer coefficient; as it lead to the minimum relative error when compared with experimental values obtained by the mass-loss rate method (Ryland, et al., 2006). However, different authors have reported that the mass-loss rate method not only accounts for convection effects, but also for radiation and evaporation effects. Therefore, the heat transfer coefficient calculated by this method would correspond to a combined coefficient, rather than a purely convection coefficient.

Several authors have suggested the use of computational fluid dynamics to overcome the issues involved in the estimation of a heat transfer coefficient. Hence, the heat transfer analysis at the product surface would be performed from basic equations, and analyzing the flow pattern. However, it is believed that fluid dynamics models may have some issues when modeling turbulent flow (Kondjoyan and Boisson, 1997). This is because turbulence models require the need of detailed modeling of boundary layers near the food surface; in addition to several empirical coefficients that may affect the accuracy of the prediction (Delgado and Sun, 2003, Pham, 2002). A common approach that has been use in recent years consists in modeling air flow under steady-state conditions with computational fluid dynamics software packages. Then, model predictions can be used to estimate local thermal properties that can be used afterwards in the heat transfer analysis. One of the main disadvantages of this method is that thermal properties are calculated often under steady-state assumption; thus, they must be considered as constant values during the transient heat transfer analysis.
4.3 Thermophysical Properties of Meats

Thermophysical properties of meats such as specific heat, thermal conductivity and density are highly affected by temperature and composition. It is well known that the moisture and fat contents are one the most influencing parameters. Choi and Okos have developed empirical correlations for predicting thermal properties of food components as functions of composition and temperature (Choi and Okos, 1986). According to previous studies and experimental validations, the maximum relative error between observed and predicted values using Choi and Okos correlations was 5.32% (Amézquita, 2004, Ryland, et al., 2006).

Pan and Singh (2001) determined the change in density and thermal conductivity of ground beef during cooking. They reported densities of 1006-1033 kg/m³ and thermal conductivities of 0.35-0.41 W/m°C; for temperatures ranging between 5 and 75°C (Pan and Singh, 2001).

Marcotte et al. (2008) studied the thermophysical properties of different meat products under different temperatures commonly found during meat processing (i.e. 20-80°C). They found considerable differences in the thermophysical properties of products containing whole muscle (e.g. flaky ham) and products made from fine meat emulsions (e.g. bologna). Their findings recommended the use of based Kirscher model to predict the thermal conductivity of meat products (Marcotte, et al., 2008b). Also, they showed that carbohydrate content highly affected the density of a meat product. Thus, products with high carbohydrate content had considerably lower densities. Density changes are particularly evident during heating operations when several chemical and structural changes take place in the meat matrix (e.g. volume expansion due to gelation).
Elansari and Hobani (2009) evaluated the effect of temperature and moisture on the thermal conductivity of different type of meats. They showed that there was a strong linear correlation between thermal properties and water content. This linear correlation was most noticeable in product containing high moisture content. Their finding presented linear regression models that can be used to estimate thermal conductivity of meats as a function of temperature and water content (Elansari and Hobani, 2009).

Thermal properties of meats are fairly understood under regular processing temperatures observed in traditional thermal processing (i.e. product temperatures ranging from 2 to 80°C). However, thermal properties of meats are not fully understood during freezing stages. This is attributed to the fact that the amount of ice formation as a function of temperature is still under investigation. Therefore, further research must be done in this area as new cooling technologies (e.g. supercooling or partial freezing) have proved to have potential in the meat industry. van der Sman (2008) predicted the enthalpy and thermal conductivity of frozen meats using their composition. This model was based in models for predicting water activity in frozen meats; which facilitates the estimation of ice fraction as a function of temperature. Also, the model took into account the fibrous meat structure and the anisotropy of ice crystals. Authors showed that there orientation of the meat fibers had an effect on the thermal conductivity of the product. This model was validated by comparing the predictions with data commonly reported in the literature; the prediction accuracy was within 10% (van der Sman, 2008).
4.4 Water Activity

Water activity of the meat product is another important parameter to be estimated when modeling heat transfer. It is affected by the water concentration on the surface of the product, the presence of boundary layers and casings, and the rate of evaporation. In modeling of meat products, water activity is usually assumed to be equal to 1.0. However, Chuntranuluck et. al. (1998) stated that water activity is a time-variable factor during chilling of foods as typically the internal water movement rate within the product cannot maintain a fully wetted product surface. They suggested that models for chilling of foods should account for three different water activity values: one to represent the starting condition, other to represent water activity during active chilling and a third value to represent the quasi-equilibrium phase (Chuntranuluck, et al., 1998a, Chuntranuluck, et al., 1998b, Chuntranuluck, et al., 1998c).

Daudin (1990) studied the kinetics of temperature and weight loss for meat model samples during chilling. It was found that the water activity of the samples slowly decreases during the chilling process. However, it remains very close to 1.0 during several hours of chilling; even when the drying rates are very high (Daudin and Swain, 1990). Therefore, the common practice of considering high constant water activity values on the product surface is a practical strategy that may not have considerable effects on the overall heat transfer model performance.

5. CONCLUSIONS

In the last few decades, modeling has gained special attention in the food industry as it is considered as a practical tool that can be used to assess meat safety quantitatively.
However, despite the great advances in this field, heat transfer modeling largely remains a research tool. On one hand, models are usually validated under controlled laboratory conditions; resulting in models that may have difficulties handling the variable and complex conditions found in the industry. On the other hand, models that may consider more realistic industrial conditions are often developed in commercial software and/or remain in scientific papers; limiting their potential use in the meat industry. Therefore, there is a need for heat transfer models capable of simulating time-temperature distributions of meat products manufactured under realistic processing scenarios, and must be adaptable to time-varying processing conditions. Also, they must be able to provide simulations for irregular products with non-homogeneous compositions. Such models must be provided to the industry as user-friendly software packages that allow integration with predictive microbiology models.

6. REFERENCES


CHAPTER II

Modeling Heat Transfer during Cooling of Cooked Ready-to-Eat Meats using Three-Dimensional Finite Element Analysis

This paper will be published as:


- Journal of Food Engineering
- Biosystems Engineering
- Journal of Agricultural Engineering Research
Modeling Heat Transfer during Cooling of Cooked Ready-to-Eat Meats using Three-Dimensional Finite Element Analysis

ABSTRACT

A heat transfer model was developed for simulating time-varying temperature distributions in ready-to-eat meats during thermal processing. Three-dimensional transient finite element analysis was presented as a tool to solve the model without the need for proprietary software. The model considers conduction as the governing heat transfer phenomenon; with evaporative, convective and radiative boundary conditions. Moreover, it takes into account several factors present in meat processing such as three-dimensional products with irregular geometries; time-dependant processing conditions including air temperature, air velocity and relative humidity; non-uniform initial temperature distributions; and temperature dependant thermal properties. The finite element analysis was performed on meshes containing linear tetrahedral and triangular elements. The step-by-step methodology described can be easily extrapolated to computational algorithms implementable in free license software (e.g. Java Technology). In addition, these algorithms can be integrated with predictive microbiology models; which can be particularly useful for evaluating the severity of thermal processing deviations caused by unexpected processing disruptions. This integration can be the foundation of open source software packages which will serve as quantitative tools to support food safety management in the meat industry.
1. INTRODUCTION

Thermal processing of ready-to-eat meat products refers to a broad range of products, processing techniques and equipment. However, despite this vast diversity, most of the thermal processing systems found in the meat industry utilize the same basic physical principles of heat and mass transfer (Knipe, 2010). The manufacturing practices of processed ready-to-eat meats include three basic steps: meat matrix preparation, heat treatment, and cooling. Although the meat matrix preparation step may differ between products, the heat treatment and cooling steps follow the same principles for most of the meat products.

During heat treatment, products are placed into ovens or smokehouses in which they come in contact with a hot fluid, usually hot air or steam. This step serves several purposes: heat stabilization of the meat matrix, pasteurization, and color fixing. Products are usually cooked to at least 130°F to ensure proper heat stabilization of the meat matrix by protein denaturation (Billerbeck and Shoemaker, 1971, Rich and Rich, 1981, Wiebe and William, 1999). This step is critical for ensuring food safety as it is designed to destroy foodborne pathogens that may be present in the product. According to the United States Department of Agriculture Food Safety and Inspection Service (USDA-FSIS), meat processors must meet the lethality performance standard. This performance standard requires a $7 \log_{10} (\text{CFU/g})$ reduction in *Salmonella* spp in ready-to-eat poultry and a $6.5 \log_{10} (\text{CFU/g})$ reduction in *Salmonella* spp in ready-to-eat beef or pork products (USDA-FSIS, 1998).
After the heat treatment, products are usually exposed to cold water showers before entering the cooling chamber. This procedure helps drop the surface temperature of the product; minimizing the weight loss caused by evaporation from the surface during the cooling step (Amézquita, 2004). Consequently, the temperature distribution of a meat product entering to the cooler is not uniform across the product. In the cooler, meat products are exposed to a cold airflow; resulting in thermal energy transfer from the surface of the product to the airflow, water evaporation from the product surface and heat conduction within the product. This heat transfer phenomenon is governed by several parameters: product size, product shape, product composition, product arrangement inside the chilling room, air temperature, air velocity, relative humidity among others. Cooling is a critical step for preventing potential growth of spore-forming bacteria that may be still present after heat treatment, as heat-activated spores of foodborne pathogens such as *Clostridium perfringens* may germinate and grow if the cooling is not quickly performed. For this reason, processors are also required to meet the stabilization performance standard established by the FSIS. This performance standard states that processors must prevent the germination and growth of toxigenic microorganisms such as *Clostridium botulinum*, and limit the growth of *Clostridium perfringens* to 1 log\(_{10}\) (CFU/g) (USDA-FSIS, 1999).

USDA-FSIS published compliance guidelines for heat treatment and cooling of ready-to-eat meat products. These guidelines recommend certain time-temperature schedules that processors should follow to comply with the lethality and stabilization standards. For instance, to meet the lethality standard, products must reach a minimum temperature of 60°C (140°F). This temperature must be maintained across the whole
product for at least 12 minutes (USDA-FSIS, 1999b). The compliance guidelines for cooling of ready-to-eat meats suggest that the risk of potential germination and outgrowth of spore-forming bacteria might be significantly reduced by using one of following three cooling schedules. The first method states that the maximum product temperature should drop from 54.4 to 26.6°C (130 to 80°F) within less than 1.5 hours; and must reach 4.4°C (40°F) in the next 5 hours. Thus, the total chilling operation must be completed within 6.5 hours. The second method states that when the cooling step begins within 90 minutes after the heat treatment, the product should be cooled from 48 to 12.7°C (120°F to 55°F) in less than 6 hours; then, the cooling should continue until the maximum product temperature reaches 4.4°C (40°F). The third method can only be used for processing of cured products containing at least 100 ppm of ingoing sodium nitrite. For these products, the maximum internal temperature should decrease from 54.4°C to 26.6°C (130 to 80°F) in less than 5 hours, and from 26.6 to 7.2°C (80 to 45°F) within the next 10 hours (USDA-FSIS, 1999a); resulting in up to a 15 hour cooling time.

Thermal processing deviations (lethality as well as stabilization) can occur in the meat industry, indicating that the thermal process did not follow the time-temperature schedules suggested in the guidelines for heat treatment and cooling of ready-to-eat meat products; thus the safety of finished product is not assured. When these deviations occur, sampling of the product for foodborne pathogens may not be the best alternative to determine the safety of the product as the microorganisms uniformly distributed in the product. Hence, USDA-FSIS recommends computer modeling as a tool to evaluate the severity of the thermal processing deviation (USDA-FSIS, 1999a). In other words, computer modeling may help verify if processors met the performance standards for
lethality and stabilization standards, even though the thermal process did not follow the guidelines.

Computer modeling can help evaluate the safety of the product subsequent to thermal processing deviations (i.e. heating and cooling deviations). These computer models, mainly predictive microbiological models usually require product time-temperature distribution as an input. Some processors may have the possibility of providing this data as they have tools available to record time-temperature data of their products. Nevertheless, most of the processors only record data at a single point considered as the worst case scenario. This point is usually assumed to be located in the center of the thickest part of the largest product being processed in a particular rack. This strategy is reasonable for some meat products and processing techniques; but there are uncertainties in determining the exact location of this point, especially for irregular-shaped products or when the process itself causes additional variations (i.e. hot/cold spots in the chamber, irregular air flow around products). Moreover, a data recording process requires not only the use of accurate equipment, but precise placement of sensors. It has been observed that not all meat processors are able to fulfill these requirements.

Other computer models provide tools for estimating product time-temperature profiles. However, these models frequently use simplification to account for and predict the time-temperature effect. The modeling simplifications may include product shape simplification, in which irregular shaped products are approximated to basic shapes such as rectangles or spheres; dimensional simplification, in which the models consider heat transfer only in one or two dimensions; simplifications of thermal properties, in which the thermal properties of a product are assumed to be constant over time; processing
simplifications, in which the environmental factors affecting the heat transfer phenomenon such as chilling room temperature, air velocity or relative humidity are considered as constant values over time. These simplifications are a common practice in modeling of heat transfer in meat products; resulting in models that do not adjust to the reality of the industry and may not provide accurate estimations. On the other hand, models to estimate time-temperature profiles with better approximations of realistic processing scenarios are usually developed in proprietary software and are not easily accessible for small meat processors.

Therefore, there is a need for open source computer models able to estimate time-temperature profiles of realistic meat processing scenarios. Moreover, these models should be available for the meat industry, including small meat processors and the regulatory agencies. Nevertheless, considering realistic processing scenarios is not an easy task to achieve, and may result in models with complex equations that cannot be readily solved. In addition, the most convenient way to make sure that the computer model will be available to the meat industry implies that the use of proprietary software must be avoided. The objective of this work is to describe step by step how to convert a complex realistic model of heat transfer in ready-to-eat meats during thermal processing to a simple one, so it can be solved without the need of proprietary software. This transformation can be achieved using finite element analysis (FEA), a numerical method for solving partial differential equations of complex geometries or domains similar to ready-to-eat meats. The methodology presented may be extrapolated to computational algorithms implementable in web-based and free license software (e.g. Java technology). These computer models may be the foundation of free-of-charge software packages,
designed from the meat processors point of view, which will serve as quantitative tools to support food safety management in the meat industry.

2. THREE-DIMENSIONAL FINITE ELEMENT ANALYSIS

The finite element method is a numerical method for solving partial differential equations for complex geometries or domains such as ready-to-eat meats. This method states that a problem involving differential equations over complex geometries can be simplified by subdividing the geometry into small regions with well understood behavior, called elements. In other words, subdividing the domain into simple elements such as tetrahedrons or cubes, in which those complicated differential equations can be easily solved. Hence, the behavior of a complex domain can be approximated by studying and accounting for the behavior of its simple elements (Zienkiewicz, 2005).

2.1 Model Definition

2.1.1 Governing Equation

Despite the diversity of thermal processing techniques employed in the meat industry, most of them operate under the same basic principles of heat and mass transfer (Knipe, 2010). In general terms, meat products are exposed to a heating or cooling medium; this causes mass and thermal energy movement from one to the other due to differentials of concentration and temperature, respectively. In addition, heat is transferred in an unsteady state or transient state in which the temperature changes as a function of both time and location. According to the first law of thermodynamics, the law of conservation of energy, the rate at which thermal energy is supplied to a control
volume ($\dot{E}_{in}$), minus the rate at which this energy leaves the control mass ($\dot{E}_{out}$), plus the rate at which energy is generated within the control mass ($\dot{E}_{g}$); must be equal to the rate at which energy is stored in the control volume ($\dot{E}_{st}$) (Incropera, 2007). Thus, considering a system like the one presented in Figure 1, a basic energy balance within a differential control volume of product indicates that:

$$\dot{E}_{in} - \dot{E}_{out} + \dot{E}_{g} = \dot{E}_{st} \quad (2.1)$$

![Figure 2.1](image.png)

Figure 2.1 Conservation of energy within a control piece of ready-to-eat meat product.

In practical terms, the divergence theorem states that the sum of all heat flux sources minus the sum of all heat flux sinks gives the net flow out of a control mass ($\nabla (q)$) (Negahban, 2009b). Thus,

$$\nabla \cdot (q) = \frac{\partial q_x}{\partial x} + \frac{\partial q_y}{\partial y} + \frac{\partial q_z}{\partial z} = \dot{E}_{out} - \dot{E}_{in} \quad (2.2)$$

By using the divergence theorem, the basic equation of energy conservation (Equation 2.1) could be expressed as:

$$-\nabla \cdot (q) + \dot{E}_{g} = \dot{E}_{st} \quad (2.3)$$

Thermal processing of meat products involves the three basic mechanisms of heat transfer simultaneously including conduction, convection and radiation; the magnitude of
each of these modes depends on the type of products and processing techniques. However, medium and large ready-to-eat meat products (e.g. whole hams) present high conductive resistance. Thus, the heat transfer phenomenon takes place mainly by conduction (Amézquita, et al., 2005b, Knipe, 2010). In fact, thermal processing of ready-to-eat meat products can be considered as a three-dimensional transient heat conduction problem without internal heat generation ($\dot{E}_g = 0$).

Taking into account Fourier's law of conduction (Equation 2.4), which states that the heat flux ($q$) is equal to the material's thermal conductivity times the negative local temperature gradient, $-\nabla(T)$.

\[
q \approx -k\nabla(T)
\]  

(2.4)

Consequently, equation 2.3 can be represented by:

\[
-\nabla \cdot (q) = \nabla \cdot (k\nabla(T)) = \dot{E}_{st}
\]  

(2.5)

Since internal heat generation during thermal processing of ready-to-eat meat products can be neglected, the rate at which energy is stored in the control mass ($\dot{E}_{st}$) is proportional to the change in temperature.

\[
\dot{E}_{st} = \rho C_p \frac{\partial T}{\partial t}
\]  

(2.6)

Therefore, the principle that governs the heat transfer phenomenon during thermal processing of ready-to-eat meat products in Cartesian coordinates can be represented by:

\[
\nabla \cdot (k\nabla(T)) = \rho C_p \frac{\partial T}{\partial t}
\]  

(2.7)
Mathematically, this governing equation does not have a unique solution and cannot be solved using finite element analysis, pending application of boundary and initial conditions. These conditions will be described in the following sections.

2.1.2 Initial Conditions

Temperature distribution within meat products entering ovens and chilling chambers may not be uniform throughout the product volume. For instance, certain meat products are usually exposed to cold water showers right before entering to the chilling room (Figure 2.2). This procedure is required to reduce the surface temperature of the product; minimizing the weight loss caused by evaporation from the surface during the chilling operation (Amézquita, 2004). Therefore, non-uniform initial temperature distributions must be considered in order to achieve accurate simulations of heat transfer in real meat processing scenarios. Hence, equation 2.7 is subject to an initial condition which states that the initial temperature of the product to be heated or chilled is a function of location.

\[ T = T_o(x, y, z) \quad \text{for} \quad 0 < x < X, \ 0 < y < Y \ \text{and} \ 0 < z < Z; \ \text{at} \ t=0 \quad (2.8) \]
2.1.3 Boundary Conditions

Ready-to-eat meat products are exposed to hot and cold air flows during thermal processing. Therefore, thermal energy will be transferred from the product surface to the air and vice versa by convection, radiation and evaporation (Figure 2.3).
Hence, equation 2.7 is subject to a boundary condition in which the heat flux at the product surface could be expressed as:

\[ q_n = q_{conv} + q_{rad} + q_{evap} \]  

(2.9)

### 2.1.3.1 Convection

Convection encompasses thermal energy movement between a fluid, in this case air, and the meat product surface; due to flow of the fluid. This phenomenon can be represented by Newton's law of cooling in which the heat flow due to convection is equal to the convective heat transfer coefficient times the driving force (Knipe, 2010). The convective heat transfer coefficient \( h_c \) is a function of the air properties, airflow conditions and product shape. The driving force is the difference in temperature between the bulk fluid \( T_a \) and the product surface \( T_s \).

\[ q_{conv} = h_c (T_s - T_a) \]  

(2.10)

### 2.1.3.2 Radiation

In thermal radiation, the meat product surface emits its thermal energy by electromagnetic waves. This phenomenon is described by the Stefan-Boltzmann law, which states that:

\[ q_{rad} = \varepsilon \sigma (T_s^4 - T_a^4) \]  

(2.11)
In equation 2.11, \( \varepsilon \) is the emissivity of the meat product which expresses the fraction of incident energy absorbed by the object. In addition, \( \sigma \) is the Stefan-Boltzmann constant, 5.676 x 10^{-8} \text{ Wm}^{-2}\text{K}^{-4}. The heat flow due to thermal radiation can be significant, especially during heating treatments, as the \( \varepsilon \) in meat products have been reported as high as 0.9 (Rahman, 2009). However, radiative heat flow is relatively small comparing with the heat flows due to conduction and convection; thus, it is usually neglected. Previous models have successfully accounted for this effect by calculating a combined radiative and convective heat transfer coefficient \( (h) \) for air-cooling of processed meat products (Amézquita, 2004, Wang and Sun, 2002a, Wang and Sun, 2002c). Hence, the combined convection and radiation heat flow is estimated by:

\[
q_{\text{conv}} + q_{\text{rad}} = h(T_s - T_a)
\]  
(2.12)

2.1.3.3 Evaporation

Evaporation can be defined as the process by which available liquid water found at the meat product surface is converted into a gaseous state. This process involves transfer of mass and thermal energy; resulting in heat and weight loss from the product surface to its surroundings. Mass and heat transfer due to evaporation are regulated by several parameters including relative humidity of the air surrounding the product, air temperature, air velocity, air flow regime, water activity of the product, product temperature, and type of product casings. As shown in equation 2.13, the heat flow due to evaporation is a function of the latent heat of evaporation and the evaporative weight loss rate (Amézquita, 2004).
\[ q_{evap} = \lambda_v(T_f) \frac{dm_w}{dt} \] (2.13)

The latent heat of evaporation can be expressed as a function of film temperature \(T_f\), for temperatures between -43°C and 100°C (Geankoplis, 2007, Perry, 2008). Hence,

\[
\lambda_v(T_f) = \begin{cases} 
1.867 \times 10^3 T_f + 2.502 \times 10^6, & -43°C < T_f < 0°C \\
-2.428 \times 10^3 T_f + 2.503 \times 10^6, & 0°C \leq T_f < 100°C
\end{cases}
\] (2.14)

Film temperature corresponds to the average between the product surface temperature and the air temperature at a particular time as calculated using equation 2.15.

\[ T_f = \frac{T_s + T_a}{2} \] (2.15)

The evaporative weight loss rate per unit area can be estimated using (Amézquita, et al., 2005a, Chuntranuluck, et al., 1998a):

\[ \frac{dm_w}{dt} = \bar{k}(awp_s - RHp_a) \] (2.16)

Values of relative humidity \(RH\) are usually recorded during industrial cooling; thus, they can be provided to the model. As it is commonly considered in modeling of heat transfer in processed meats, water activity can be assumed to be \(aw = 1.0\). Also, the mass transfer coefficient \(\bar{k}\) can be calculated using the Lewis relationship (Daudin and Swain, 1990, Wang and Sun, 2002a). Therefore,

\[ \bar{k} = \frac{h_{conv}}{\lambda_v 64.7} \] (2.17)
The term $p$ represents the saturation water vapor pressure at a particular temperature. Hence, $p_s$ is the saturation water pressure at product's surface temperature; and $p_a$ is the saturation water vapor pressure at air temperature. These values can be calculated using the Antoine equation (Amézquita, et al., 2005a, Chuntranuluck, et al., 1998a, Sun and Hu, 2003, Trujillo, et al., 2007) as shown:

$$p = \exp\left(23.4795 - \frac{3990.56}{233.833 + T}\right)$$

(2.18)

### 2.1.4 Estimation of Thermal Properties

#### 2.1.4.1 Air properties

Air properties at film temperatures can be calculated based on tabulated values previously reported (Kays, et al., 2005). Hence, thermal conductivity of air $k_a$ is calculated as:

$$k_a = 7.15 \times 10^{-5} \cdot T_f + 2.41794 \times 10^{-2} \quad -23^\circ C < T_f < 80^\circ C$$

(2.19)

Moreover, air viscosity ($\mu_a$) and density ($\rho_a$) can be estimated by equations 2.20 and 2.21 respectively.

$$\mu_a = 4.5677 \times 10^{-8} \cdot T_f + 1.7258237 \times 10^{-5} \quad -23^\circ C < T_f < 80^\circ C$$

(2.20)

$$\rho_a = -3.397 \times 10^{-3} \cdot T_f + 1.29532 \quad -23^\circ C < T_f < 80^\circ C$$

(2.21)

The specific heat of air ($C_p_a$) at values between -30 and 100$^\circ$C are presented in table 2.1.
Table 2.1 Specific heat of air (Kays, et al., 2005)

<table>
<thead>
<tr>
<th>Range [°C]</th>
<th>(C_{p_a} \text{ [J/kg·K]} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-30 \leq T_f &lt; 27)</td>
<td>1004</td>
</tr>
<tr>
<td>(-27 \leq T_f &lt; 50)</td>
<td>1005</td>
</tr>
<tr>
<td>(50 \leq T_f &lt; 60)</td>
<td>1006</td>
</tr>
<tr>
<td>(60 \leq T_f &lt; 70)</td>
<td>1007</td>
</tr>
<tr>
<td>(70 \leq T_f &lt; 90)</td>
<td>1008</td>
</tr>
<tr>
<td>(90 \leq T_f &lt; 100)</td>
<td>1009</td>
</tr>
<tr>
<td>(T_f = 100)</td>
<td>1010</td>
</tr>
</tbody>
</table>

2.1.4.2 Thermal properties of ready-to-eat meats

Choi and Okos (1986) developed empirical correlations for predicting thermal properties of food components as functions of composition and temperature. According to previous studies and experimental validations, the maximum relative error between observed and predicted values was 5.32% (Amézquita, 2004, Ryland, et al., 2006). Therefore, these models can be used to estimate the thermal properties of ready-to-eat meat products.

The general form of the models proposed by Choi and Okos (1986) establish that:

\[
k = \sum k_i \tilde{x}_i^v \tag{2.22}
\]

\[
C_p = \sum C_{p_i} \tilde{x}_i^w \tag{2.23}
\]

\[
\rho = \frac{1}{\tilde{x}^v} \tag{2.24}
\]

\(\tilde{x}_i^w\) is the weight fraction of component \(i\); and \((\tilde{x}_i^v)\) represents the estimated volume fraction. In addition, \(k_i\) and \(C_{p_i}\) correspond to the thermal conductivity and specific heat of the pure components including protein, fat, carbohydrates and ash; at a
particular temperature. As it will be shown later in this chapter, this temperature is assumed to be the volumetric average temperature (Section 2.3.1). Table 2.2 shows the equations to calculate the thermal conductivity and specific heat of pure components as a function of temperature.

Table 2.2 Thermal properties of major pure components of foods. (Choi and Okos, 1986)

<table>
<thead>
<tr>
<th>Thermal Property</th>
<th>Choi and Okos Model</th>
</tr>
</thead>
</table>
| $k$ [W/m°C]      | $k_{prot} = 1.7881 \times 10^{-1} + 1.1958 \times 10^{-3}T - 2.7178 \times 10^{-6}T^2$
|                  | $k_{fat} = 1.8071 \times 10^{-1} - 2.7604 \times 10^{-3}T - 1.7749 \times 10^{-7}T^2$
|                  | $k_{carb} = 2.0141 \times 10^{-1} + 1.3874 \times 10^{-3}T - 4.3312 \times 10^{-6}T^2$
|                  | $k_{ash} = 3.2962 \times 10^{-1} + 1.4011 \times 10^{-3}T - 2.9069 \times 10^{-6}T^2$
|                  | $k_w = 5.7109 \times 10^{-1} + 1.7625 \times 10^{-3}T - 6.7036 \times 10^{-6}T^2$
| $C_p$ [kJ/kg°C]  | $C_{pprot} = 2.0082 + 1.2089 \times 10^{-3}T - 1.3129 \times 10^{-6}T^2$
|                  | $C_{pfat} = 1.9842 + 1.4733 \times 10^{-3}T - 4.8008 \times 10^{-6}T^2$
|                  | $C_{pcarb} = 1.5488 + 1.9625 \times 10^{-3}T - 5.9399 \times 10^{-6}T^2$
|                  | $C_{pash} = 1.0926 + 1.8896 \times 10^{-3}T - 3.6817 \times 10^{-6}T^2$
|                  | $C_{pw} = 4.1762 - 9.0864 \times 10^{-5}T + 5.4731 \times 10^{-6}T^2$

2.1.4.3 Heat transfer coefficient

The heat transfer coefficient is a function of several factors including product related features such as size, surface texture and shape; physical properties of air such as temperature, velocity, viscosity, density; and thermal properties of air such as thermal diffusivity and conductivity. As it was previously discussed, the heat transfer coefficient used in this model also accounts for the combined effect of convective and radiative heat transfer. Hence,

$$h = h_{rad} + h_{conv}$$

(2.25)
Previous studies have effectively estimated the radiative heat transfer coefficient using the following equation (Amézquita, 2004, Geankoplis, 2007, Wang and Sun, 2002b):

\[ h_{rad} = \sigma \varepsilon (T_{K,s} + T_{K,a})(T_{K,s}^2 + T_{K,a}^2) \]  

(2.26)

During thermal processing of ready-to-eat meat products, temperature differences around the chilling chamber affect the relative buoyancy of the air. Moreover, chilling chambers are typically equipped with air circulation systems to increase the air flow movement; resulting in higher cooling rates. Therefore, both natural and forced convection effects need to be taken into account in order to estimate the convective heat transfer coefficient \( h_{conv} \). This mixed heat transfer coefficient can be calculated as follows (Amézquita, et al., 2005a, Davey and Pham, 1997, Wang and Sun, 2002a, Wang and Sun, 2002a, Wang, et al., 2006, Wang and Sun, 2002b):

\[ h_{conv} = \sqrt[3]{h_{fc}^3 + h_{nc}^3} \]  

(2.27)

where \( h_{fc} \) and \( h_{nc} \) represent the heat transfer coefficients due to forced and natural convection respectively. Each of these coefficients can be calculated using the following relations:

\[ h_{fc} = \frac{k_a Nu_{fc}}{D} \quad \text{and} \quad h_{nc} = \frac{k_a Nu_{nc}}{D} \]  

(2.28)

The characteristic dimension \( D \) is a factor associated with the geometrical structure of the product. \( Nu_{fc} \) and \( Nu_{nc} \) correspond to the Nusselt number associated with forced and natural convection respectively. For products with ellipsoidal
and complex irregular shapes, it can be considered as the square root of the total surface area of the product (Yovanovich, 1987a).

\[ D = \sqrt[2]{A_s} \]  

(2.29)

The total surface area of a product \( (A_s) \) can be determined by using three-dimensional modeling software, as it will be discussed in the following sections. In addition, the surface area of products that can be approximated to ellipsoidal shapes can be calculated using equation 2.30 (Yovanovich, 1987a). This equation is valid for products that can be approximated as prolate spheroids (i.e. products having ellipsoidal semi-axis \( a \geq b \approx c \)). Therefore, their \( A_s \) can be calculated using equation 2.30.

\[ A_{s_{\text{ellip}}} = 2\pi b^2 \left[ 1 + \frac{\sin^{-1} \tilde{e}}{\tilde{e} \sqrt{1 - \tilde{e}^2}} \right] \quad \text{with} \quad \tilde{e} = \sqrt{1 - \left( \frac{b}{a} \right)^2} \]  

(2.30)

Nusselt number (\( N_u \)) is a dimensionless parameter that symbolizes the ratio of convective to conductive heat transfer phenomena at the product surface. This parameter is influenced by various parameters including the characteristic dimension of the product, thermo-physical properties of air, and several dimensionless parameters such as Reynolds number, Prandlt number, Grashof number and Raleigh number. Yovanovich has presented general expressions to estimate the Nusselt number under natural and forced convection for three-dimensional bodies of complex geometries. Nusselt number for forced convection \( (N_{ufc}) \) can be estimated by the following expression (Yovanovich, 1988),

\[ N_{ufc} = 2\sqrt{\pi} + \left( 0.15\pi^{1/4} Re^{1/2} + 0.35 Re^{0.566} \right) Pr^{1/3} \quad 0 < Re < 2 \times 10^5 \]  

(2.31)
Re represents the Reynolds number which measures the relation between inertial and viscous forces (Equation 2.32); serving as an indicator to characterize the air flow regime.

\[
Re = \frac{DV_a \rho_a}{\mu_a} \quad (2.32)
\]

Prandlt number \((Pr)\) is a dimensionless parameter that illustrates the relation between the rate of diffusivity for momentum and the thermal diffusion rate; it is usually determined as:

\[
Pr = \frac{\mu_a C_{pa}}{k_a} \quad (2.33)
\]

Additionally, Nusselt number for natural convection \((Nu_{nc})\) can be computed as (Yovanovich, 1987b),

\[
Nu_{nc} = 3.47 + 0.51 \frac{Ra^{1/4}}{4} \quad 0 < Ra < 10^8 \quad (2.34)
\]

In equation 2.34, \(Ra\) symbolizes the Rayleigh number, an indicator of buoyancy flow. This number is defined as the product of the Prandlt number (equation 2.33), and Grashof number \((Gr)\) which determines the ratio of buoyancy to viscous forces present in natural convection (Amézquita, 2004).

\[
Gr = \frac{p_a^2 \cdot 9.80665 \cdot \left(\frac{1}{T_a + 273.15}\right) \cdot D^3 \cdot |T_{avg} - T_a|}{\mu_a^2} \quad (2.35)
\]

2.2 Weak Formulation

The first step of the finite element analysis requires a transformation of the governing equation (Equation 2.7) in to a set of integrals over the product geometry,
rather than as a set of partial differential equations. According to the method of weighted residuals or Galerkin procedure, the weak formulation can be obtained by multiplying the governing equation that describes the heat transfer phenomenon in ready-to-eat meats (Equation 2.7) by an arbitrary weighting function ($\bar{T}$); followed by an integration over the product volume ($V$) (Negahban, 2009b). Thus,

$$\int_V \bar{T} \nabla \cdot (k \nabla (T)) dV = \int_V \bar{T} \rho C_p \frac{\partial T}{\partial t} dV + B.C. \quad (2.36)$$

The left side term of equation 2.36 can be simplified by using the following equation derived from the divergence theorem (Negahban, 2009b),

$$\int_V \bar{T} \nabla \cdot (k \nabla (T)) dV = - \int_V [\nabla (T) \cdot k \nabla (T)] dV - \int_S \bar{T} q_n dS \quad (2.37)$$

Therefore, the weak formulation can be rewritten as:

$$- \int_V [\nabla (T) \cdot k \nabla (T)] dV - \int_S \bar{T} q_n dS = \int_V \bar{T} \rho C_p \frac{\partial T}{\partial t} dV + B.C. \quad (2.38)$$

Equation 2.38 must be reorganized so that all the terms containing the unknown variable, $T$, will be on the left hand side; and all the terms with known variables will be on the right hand side. Also, the previously discussed initial and boundary conditions must be imposed. As a result, the final weak formulation can be symbolized as follows:

$$\int_V \bar{T} \rho C_p \frac{\partial T}{\partial t} dV + \int_V [\nabla (T) \cdot k \nabla (T)] dV + \int_S \bar{T} h T_s dS = \int_S \bar{T} h T_a dS - \int_S \bar{T} q_{evap} dS \quad (2.39)$$

The weak formulation encompasses the terms that will yield to the definition of the three main functions that will be defined using the Final Element Method: global
mass matrix, stiffness matrix and load vector (Figure 2.4). The global mass matrix, also called capacitance matrix, indicates how the rate of increase of energy is related to the temperature of the product. The global stiffness matrix, also called conductance matrix, represents the effect of heat conduction through the product. The global load vector, also called force vector, accounts for the heat transfer on the product surface. Each of these functions will be described in detail in the following sections.

\[
\int_V \bar{T} \rho C_p \frac{\partial T}{\partial t} \, dV + \int_V [\nabla (\bar{T}) \cdot k \nabla (T)] \, dV + \int_S \bar{T} h T_s \, dS = \int_S \bar{T} h T_a \, dS - \int_S \bar{T} q_{evap} \, dS
\]

**Figure 2.4** Weak form and finite element functions: The weak formulation provides sources to derive the three main functions of a final element analysis, including global mass matrix \([M]\), stiffness matrix \([K]\) and load vector \([F]\).

### 2.3 Discretization

The finite element method suggests that the solution of the weak formulation can be estimated by subdividing the product domain and boundary into smaller interconnected regions called elements. The collection of all these interconnected elements is known as *mesh* (Figure 2.5). Stiffness and mass matrices, boundary terms and load vector are calculated over each element contained in the mesh. Then, the effect
of each element is added to account for the solution of the entire domain; this process is known as assembling. For instance, the term representing the source of the global mass matrix can be simplified to:

$$\int_V [\mathbf{V}(\hat{T}) \cdot k \nabla(T)] dV = \sum_{e=0}^{n_e} \int_{V_e} [\mathbf{V}(\hat{T}^e) \cdot k^e \nabla(T^e)] dV_e$$  \hspace{1cm} (2.40)

Figure 2.5 Three-dimensional mesh and its elements: A mesh is a collection of interconnected elements used to approximate the shape of a body. Mesh elements are defined based on the global node system, a set of (x,y,z) points located within the body. The domain of a three-dimensional mesh is represented by three-dimensional elements such as linear tetrahedrons defined by four local nodes. The boundary or surface of a three-dimensional mesh is represented by two-dimensional elements such as triangles defined by three local nodes.

The discretization process requires the creation of a global node system or group of points defined by xyz coordinates contained within the mesh. These nodes describe the geometry and are used to define each of the elements. In addition, the solution of the weak formulation is determined in terms of discrete values of temperatures calculated in
each global node at each particular time. As a result, the unknown variable will be the temperature as a function of location and time:

\[ T = T(x, y, z, t) \]  

Several methods can be used for generating the mesh of a meat product. For instance, simple shaped products (i.e. rectangular, spherical) can be easily meshed by using algorithms for automatic mesh generation such as Delaunay Triangulation and Advancing Front Method. Although these algorithms can also be implemented for mesh generation of complex or irregular shaped products, this task can be arduous and time-consuming. A common method for meshing irregular shaped products states that a three-dimensional (3D) computer model of the meat product should be recreated.

There are several techniques that can be used to create 3D models of meat products including 3D-scanning, voxel rendering, procedural modeling, sculpt modeling and other techniques. These techniques usually require the use of 3D computer graphics software such as SolidWorks, LightWave and Autodesk 3ds Max. Figure 2.6 shows an illustration of how Computer Tomography (CT) scanning can be used to carry out 3D modeling of meat products. In this example, a series of cross-sectional images of a product is acquired by CT scanning (Figure 2.6b). Then, 3D modeling software (e.g. Mimics and 3D-Doctor) processes the CT images in order to create a 3D surface representation of the meat product (Figure 2.6c).

Once the 3D surface model is created, mesh generators can be used to generate the collection of nodes and elements that will be used during the finite element analysis (Figure 2.6d). Certain mesh generators, such as the ones found in simulation software
(i.e. COMSOL Multiphysics), allow users to export meshes as text files. These text files provide useful information about the mesh including nodes location, nodal numbering, element numbering, and connectivity matrices (Figure 2.6e). Special care should be taken when building the meshes that will be used during the finite element analysis. Smoothing and wrapping procedures are common practice in 3D modeling. However, they may lead to geometry modifications or deformations that cannot be easily detected (e.g. volume reduction). As it is known, inaccuracies on the geometry definition can highly affect the numerical analysis. Therefore, it is very important to evaluate the quality of the meshes. For instance, meshes should be composed by closed to equilateral triangles as opposed to irregular triangles. Different mesh generators and FEA commercial software packages (e.g. 3-matic) provide tools for analyzing mesh quality based on different shape measures and thresholds defined by the user.

Figure 2.6 Mesh generation for products with complex geometries using CT-scanning. (a) Irregular shaped meat product. (b) Cross-sectional images obtained by CT-scanning. (c) 3D modeling of the meat product using the CT-images. (d) Mesh generation using tetrahedral and triangular elements. (e) Mesh exported as a text file. (f) FEM Analysis based on mesh.txt
2.3.1 Domain Discretization Using Three-dimensional Elements

2.3.1.1 The Linear Tetrahedron

The linear tetrahedral elements are one of the most common elements used for domain discretization of three-dimensional irregular geometries; as they can easily approximate to any irregular domain configuration. Moreover, they allow generating simple matrix equations for each element (Negahban, 2009b). A typical linear tetrahedron is defined by four local nodes with the following numbering convention:

![Figure 2.7 Linear tetrahedral element defined by four local nodes.](image)

The volume of a linear tetrahedral element is given by the following determinant (Zienkiewicz, 2005):

\[
V_e = \frac{1}{6} \det \begin{bmatrix}
1 & x_1 & y_1 & z_1 \\
1 & x_2 & y_2 & z_2 \\
1 & x_3 & y_3 & z_3 \\
1 & x_4 & y_4 & z_4
\end{bmatrix}
\] (2.42)

The proper nodal numbering system can be analyzed by calculating the volume of the element with different configuration using equation 2.42. A proper nodal numbering...
system will provide a positive volume value (i.e. $V_e > 0$). The shape functions of a linear tetrahedron correspond to basic linear equations that can be extracted from volume coordinates. Hence, their partial derivatives with respect to $x$, $y$ and $z$ planes are constant values. This highly simplifies several steps of the finite element analysis as it will be shown later in Section 2.4. Linear tetrahedral shape functions and their derivatives can be written in the following general form:

$$N_i = \frac{a_i + b_i x + c_i y + d_i z}{6V_e} \quad i = 1, 2, 3, 4$$ \hspace{1cm} (2.43)

$$\frac{\partial N_i}{\partial x} = \frac{b_i}{6V_e} \quad \frac{\partial N_i}{\partial y} = \frac{c_i}{6V_e} \quad \frac{\partial N_i}{\partial z} = \frac{d_i}{6V_e}$$ \hspace{1cm} (2.44)

The constants $a_i$, $b_i$, $c_i$ and $d_i$ can be calculated as follows:

$$a_1 = \text{det} \begin{bmatrix} x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \end{bmatrix} \quad b_1 = -\text{det} \begin{bmatrix} 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \end{bmatrix}$$ \hspace{1cm} (2.45)

$$c_1 = -\text{det} \begin{bmatrix} x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \\ x_4 & 1 & z_4 \end{bmatrix} \quad d_1 = -\text{det} \begin{bmatrix} x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \end{bmatrix}$$

The other constants are defined by cyclic interchange of the subscripts in the order 1, 2, 3, and 4. Also, it is necessary to change the signs between each cyclic interchange. For instance, the constants $a_2$, $b_2$, $c_2$ and, $d_2$ would be defined by:

$$a_2 = -\text{det} \begin{bmatrix} x_3 & y_3 & z_3 \\ x_4 & y_4 & z_4 \\ x_1 & y_1 & z_1 \end{bmatrix} \quad b_2 = \text{det} \begin{bmatrix} 1 & y_3 & z_3 \\ 1 & y_4 & z_4 \\ 1 & y_1 & z_1 \end{bmatrix}$$ \hspace{1cm} (2.46)

$$c_2 = \text{det} \begin{bmatrix} x_3 & 1 & z_3 \\ x_4 & 1 & z_4 \\ x_1 & 1 & z_1 \end{bmatrix} \quad d_2 = \text{det} \begin{bmatrix} x_3 & y_3 & 1 \\ x_4 & y_4 & 1 \\ x_1 & y_1 & 1 \end{bmatrix}$$
2.3.1.2 Element temperature

Temperature within an element is approximated by using the element shape functions and the values of temperature at its nodes. Consequently, temperature within an element can be determined by:

\[ T^e = T_1 N_1(x, y, z) + T_2 N_2(x, y, z) + T_3 N_3(x, y, z) + T_4 N_4(x, y, z) \]  
\( (2.47) \)

Equation 2.47 can also be written in matrix form as:

\[ T^e = [N_1 \quad N_2 \quad N_3 \quad N_4] \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = \{N^e\}^T \{T^e\} \]  
\( (2.48) \)

Likewise, the divergence of temperature within an element can be described as:

\[ \Delta T^e = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial z} & \frac{\partial N_2}{\partial z} & \frac{\partial N_3}{\partial z} & \frac{\partial N_4}{\partial z} \end{bmatrix} \begin{bmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \end{bmatrix} = [B^e]^T \{T^e\} \]  
\( (2.49) \)

where,

\[ B^e = \begin{bmatrix} \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial x} & \frac{\partial N_3}{\partial x} & \frac{\partial N_4}{\partial x} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_2}{\partial y} & \frac{\partial N_3}{\partial y} & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial z} & \frac{\partial N_2}{\partial z} & \frac{\partial N_3}{\partial z} & \frac{\partial N_4}{\partial z} \end{bmatrix} = \frac{1}{6V^e} \begin{bmatrix} b_1 & b_2 & b_3 & b_4 \\ c_1 & c_2 & c_3 & c_4 \\ d_1 & d_2 & d_3 & d_4 \end{bmatrix} \]  
\( (2.50) \)
Also, these same statements are valid for approximating the weighting function within the element. So,

\[ \bar{T}^e = \{N^e\} \{\bar{T}^e\}^T \]  \hspace{1cm} (2.51) 

\[ \Delta \bar{T}^e = \{B^e\} \{\bar{T}^e\}^T \]  \hspace{1cm} (2.52) 

### 2.3.1.3 Volumetric average temperature

As it was shown in section 2.1.4, empirical correlations suggested by Choi and Okos (1986) will be used for predicting thermal properties of ready-to-eat meats as functions of composition and temperature. As the temperature of an element is determined by the temperature of its nodes, it is necessary to define a method for establishing the average temperature of an element. One common method for defining average temperature in three-dimensional elements is known as the volumetric average temperature (Equation 2.53) (Negahban, 2009b). This temperature can be used to estimate the thermal properties of each element at a particular time.

\[ T^e_{avg} = \frac{1}{V_e} \{T^e\} \int_{V_e} \{N^e\} dV_e \]  \hspace{1cm} (2.53)
2.3.2 Boundary Discretization Using Two-dimensional Elements

2.3.2.1 Triangular elements

Triangular elements are another widely used type of element from the triangular element family. These elements are commonly used for boundary discretization of three-dimensional irregular geometries. Also, they are frequently used for domain discretization in two-dimensional problems. They provide the same advantages as the ones previously mentioned for tetrahedral elements. A typical linear triangular element is defined by three local nodes with the following numbering convention:

![Figure 2.8](image)

A correct nodal numbering configuration can be determined by calculating the area of the element. Positive areas (i.e. \( A_e > 0 \)) represent a correct numbering system. The area of a linear triangular element is given by the following determinant.

\[
A_e = \frac{1}{2} \det \begin{bmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{bmatrix}
\]  

(2.54)
The shape functions of a 3-node triangular element correspond to basic linear equations that can be extracted from area coordinates. In this way, shape functions and their derivatives are given by (Olson and Negahban, 2007):

$$N_{si} = \frac{a_i + b_i x + c_i y}{2A_e} \quad i = 1, 2, 3$$  \hspace{1cm} (2.55)

$$\frac{\partial N_{si}}{\partial x} = \frac{b_i}{2A_e} \quad \quad \frac{\partial N_{si}}{\partial y} = \frac{c_i}{2A_e}$$  \hspace{1cm} (2.56)

The constants $a_i$, $b_i$ and $c_i$ can be calculated as follows:

$$a_1 = x_2 y_3 - x_3 y_2 \quad b_1 = y_2 - y_3 \quad c_1 = x_3 - x_2$$  \hspace{1cm} (2.57)

Likewise for the tetrahedral elements, the other constants are defined by cyclic interchange of the subscripts in the order 1, 2, 3.

### 2.3.2.2 Boundary Element temperature

As for the domain element, temperature within a boundary element is approximated by using the boundary element shape functions and the values of temperature at its nodes. Thus,

$$T^{be} = T_{s1} N_{s1}(x, y) + T_{s2} N_{s2}(x, y) + T_{s3} N_{s3}(x, y)$$  \hspace{1cm} (2.58)

Equation 2.57 can also be written in matrix form as:

$$T^{be} = \begin{bmatrix} N_{s1} & N_{s2} & N_{s3} \end{bmatrix} \begin{bmatrix} T_{s1} \\ T_{s2} \\ T_{s3} \end{bmatrix} = \{N^{be}\}^T \{T^{be}\}$$  \hspace{1cm} (2.59)
2.4 Element Equations

The weak formulation previously presented includes the terms for calculating the three main components of a finite element formulation: global mass matrix, global stiffness matrix and global load vector. These three components can be calculated by evaluating the effect of individual elements. This can be performed by evaluating element stiffness matrices, element mass matrices and element load vectors that will ultimately lead to the final element formulation.

2.4.1 Element Stiffness Matrix

The element stiffness matrix accounts for two factors: the heat conduction phenomenon within the meat product and a combined effect of convection and radiation on the product surface. Thus,

$$[K^e] = [K^e_{\text{cond}}] + [K^e_{\text{hr}}]$$  \hspace{1cm} (2.60)

The temperature approximations from the last section can be used to symbolize the heat conduction phenomenon through an element as:

$$\int_{V_e} [\nabla(T^e)] \cdot k^e \nabla(T^e) \, dV_e = \{ \bar{T}^e \}^T \int_{V_e} k^e \{ B^e \} \{ B^e \}^T \, dV_e \{ T^e \}$$  \hspace{1cm} (2.61)

In equation 2.61, the vector of weighting factors and the vector of nodal temperatures can be taken outside the integral as they are constant values. The global stiffness matrix will account for the terms that are function of space; in other words, the terms inside the integral. This matrix will be pre-multiplied by virtual temperatures and post-multiplied by the temperature vector. As a result,

$$[K^e_{\text{cond}}] = \int_{V_e} k^e \{ B^e \} \{ B^e \}^T \, dV_e$$  \hspace{1cm} (2.62)
where,

\[
B^e = \frac{1}{\delta^e} \begin{bmatrix}
    b_1 & b_2 & b_3 & b_4 \\
    c_1 & c_2 & c_3 & c_4 \\
    d_1 & d_2 & d_3 & d_4 
\end{bmatrix}
\]  

(2.63)

Since the terms inside the integral are constant values, they may be taken outside the integral. This yields an element matrix corresponding to:

\[
[K_{\text{cond}}^e] = k^e \begin{bmatrix}
    b_1 & b_2 & b_3 & b_4 \\
    c_1 & c_2 & c_3 & c_4 \\
    d_1 & d_2 & d_3 & d_4 
\end{bmatrix} \begin{bmatrix}
    b_1 & c_1 & d_1 \\
    b_2 & c_2 & d_2 \\
    b_3 & c_3 & d_3 \\
    b_4 & c_4 & d_4 
\end{bmatrix} V^e
\]  

(2.64)

As it was shown in Figure 2.4, part of the effect of heat transfer due to convection and radiation must be included in the stiffness matrix. This term can be rewritten as:

\[
\int_{A_e} \bar{T} h T_s dA_e = \{\bar{T}^{be}\}^T \int_{A_e} h \{N^{be}\}\{N^{be}\}^T dA_e \{T^{be}\}
\]  

(2.65)

As a result,

\[
[K_{\text{hr}}^e] = \int_{A_e} h \{N^{be}\}\{N^{be}\}^T dA_e
\]  

(2.66)

Basically, the terms inside the integral include a 4x4 matrix with polynomials of second order. This integral can be easily calculated using one of the numerical integration procedures for triangular regions shown in the next section.

2.4.2 Element Mass Matrix

Other important term shown in the weak formulation is the source of the mass matrix (Figure 2.4). This term is important as it will be the base for calculating the transient solution; as it not only accounts for variations in space, but also for variations in
time. Using the temperature approximation from the discretization process, the source of the mass matrix can be written mathematically as:

\[
\int_{V_e} \bar{T}^e \rho C_p \frac{\partial T^e}{\partial t} dV_e = \{T^e\}^T \int_{V_e} p C_p \{N^e\} \{N^e\}^T dV_e \frac{\partial T^e}{\partial t}
\]  

(2.67)

Once again, the weighting factors and the derivative of temperature over time can be located outside the volume integral as they are not function of space. So, the element mass matrix will be given by:

\[
M^e = \int_{V_e} p C_p \{N^e\} \{N^e\}^T dV_e
\]  

(2.68)

As for the stiffness matrix, the terms inside the integral include a 4x4 matrix with polynomials of second order. This volume integral can be solved by numerical integration for tetrahedral regions. These methods will be discussed in the next section.

2.4.3 Element Load Vector

The left hand side terms shown in the weak formulation were presented as the source of the load vector (Figure 2.4). This vector encompasses the effect of heat transfer on the product surface due to convection, radiation and evaporation. As a result,

\[
F^e = F_{hr}^e + F_{evap}^e
\]  

(2.69)

The combined effect of heat transfer due to convection and radiation can be simplified using the temperature approximation presented in the previous section. Hence,

\[
\int_{A_e} \bar{T}^e h T_a dA_e = \{T^e\} \int_{A_e} h T_a \{N^{be}\} dA_e
\]  

(2.70)
Extracting the constant terms from the integral, the element load vector that describes the heat transfer due to convection and radiation on the product surface is:

\[
\{F_{hr}^e\} = \int_{A_e} hT_a \{N^{be}\} dA_e
\]  

(2.71)

On the other hand, heat loss due to evaporation from a single boundary element that can be derived from the weak formulation is:

\[
\int_{A_e} \overline{T}^{be} q_{evap} dA_e = \{\overline{T}^{be}\}^T \int_{A_e} q_{evap} \{N^{be}\} dA_e
\]  

(2.72)

The element load vector that includes the heat loss due to evaporation is given by:

\[
\{F_{evap}^e\} = \int_{A_e} q_{evap} \{N^{be}\} dA_e
\]  

(2.73)

Equations 2.71 and 2.73 show the two integrals that must be calculated in order to determine the element load vector. These two integrals include vectors with linear polynomials over triangular areas that can be calculated by numerical integration. This process will be explained in detail in the following section.

2.4.4 Finite Element Formulation

Finally, the finite element formulation for a single element could be written in the following general form (Olson and Negahban, 2007):

\[
\left[M^e\right]\left\{\frac{\partial T^e}{\partial t}\right\} + \left[K^e\right]\{T^e\} = \{F^e\}
\]  

(2.74)
2.5 Numerical Integration

The finite element method requires calculation of definite integrals of polynomials over the element domains, in this case, over triangular and tetrahedral regions. These integrals are easier to calculate using numerical integration in place of exact integration. There are several numerical methods that can be used for approximating the value of the definite integral of a function. Gaussian quadrature and integration formulae are two of the most widely used methods for approximating integrals over triangular and tetrahedral regions.

2.5.1 Gaussian quadrature

Quadrature rules on a triangle have the form

\[ \iint_{A_e} f(x,y) \, dx \, dy = A_e \sum_{i=1}^{n_{\text{ip}}} W_i f(L_{i1}^e, L_{i2}^e, L_{i3}^e) + \text{error} \]  

Equation 2.75 states that the integral of a polynomial over a triangular region can be approximated by evaluating the polynomial at certain set of integration points. These points correspond to area coordinates instead of Cartesian coordinates. However, the shape functions can be used to convert the area coordinates to Cartesian coordinates. After evaluating the polynomial at a particular point \((x,y)\), the result is multiplied by a weighting factor. Finally, the summation of the weighted polynomial evaluations is multiplied by the exact area of the triangle. The error introduced into the calculation can be reduced by using an adequate number of integration points \((n)\), which is determined according to the polynomial order. In general, Gaussian quadrature can yield an exact integral for polynomials of degree \((2n - 1)\) or less (Negahban, 2009a).
For instance, a linear polynomial for a triangular region will require one integration point \((L_1=1/3, L_2=1/3, L_3=1/3)\) and the weighting factor is 1. These area coordinates can be transformed into Cartesian coordinates using the basics of the shape functions for a triangular element,

\[
x = L_1x_1 + L_2x_2 + L_3x_3
\]

\[
y = L_1y_1 + L_2y_2 + L_3y_3
\]

where \(x_i\) and \(y_i\) correspond to the coordinates of the local nodes 1, 2 and 3 that define that particular triangle. Subsequently, the polynomial can be evaluated at the integration point \((x,y)\). This result is multiplied by one which represents the weighting factor; which will yield to the value of the exact integral of the polynomial over the triangular area.

The same procedure is valid for tetrahedral regions. Quadrature rules on a tetrahedron have the form:

\[
\iiint_{V_e} f(x, y, z)dx \, dy \, dz = V_e \sum_{i=1}^{n_{ip}} W_i f(L_1^i, L_2^i, L_3^i, L_4^i) + \text{error}
\]

where \(V_e\) is the exact volume of the tetrahedral element. In this case, \(L_i^j\) represent the volume coordinates. These coordinates can be transformed into Cartesian coordinates using the notion of the element shape functions. Thus,

\[
x = L_1x_1 + L_2x_2 + L_3x_3 + L_4x_4
\]

\[
y = L_1y_1 + L_2y_2 + L_3y_3 + L_4y_4
\]

\[
z = L_1z_1 + L_2z_2 + L_3z_3 + L_4z_4
\]
Weights and evaluation points for integration on tetrahedral and triangular regions are presented in the following table.

### Table 2.3 Gaussian integration points and weights for triangular and tetrahedral regions (Zienkiewicz, 2005).

<table>
<thead>
<tr>
<th>Region</th>
<th>Polynomial Order</th>
<th>Integration Points</th>
<th>Coordinates L_i</th>
<th>Weights W_i</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td>Linear</td>
<td>P₁</td>
<td>L₁=1/3 L₂=1/3 L₃=1/3</td>
<td>Wp₁=1</td>
</tr>
<tr>
<td></td>
<td>Quadratic</td>
<td>P₁ P₂ P₃</td>
<td>L₁=1/2 L₂=1/2 L₃=0</td>
<td>Wp₁=1/3, Wp₂=1/3, Wp₃=1/3</td>
</tr>
<tr>
<td>Tetrahedral</td>
<td>Linear</td>
<td>P₁</td>
<td>L₁=1/4 L₂=1/4 L₃=1/4 L₄=1/4</td>
<td>Wp₁=1</td>
</tr>
<tr>
<td></td>
<td>Quadratic</td>
<td>P₁ P₂ P₃ P₄</td>
<td>L₁=1/4 L₂=1/4 L₃=1/4 L₄=1/4</td>
<td>Wp₁=1/4, Wp₂=1/4, Wp₃=1/4, Wp₄=1/4</td>
</tr>
</tbody>
</table>

2.5.2 Integration formulae for triangles and tetrahedrons

There are some integration formulae available for evaluating definite integrals of polynomials over the domains of triangular and tetrahedral elements. These formulae allow evaluating those integrals using exact integration. For instance, the integration formula for calculating the exact integral of \( f(x) = x^2 \) over a triangular element will be:

\[
\iint_{A_e} x^2 \, dx \, dy = \frac{A_e}{12} (x_1^2 + x_2^2 + x_3^2)
\]

(2.82)
For a tetrahedral region,

\[ \iiint_{V_e} x^2 \, dx \, dy \, dz = \frac{V_e}{20} \left( x_1^2 + x_2^2 + x_3^2 + x_4^2 \right) \]  

(2.83)

Other integration formulae are presented in Table 2.4. These formulae help calculate exact integrals in a very simple way. The only requirement is that the element must be defined in a Cartesian plane with the origin taken at the element centroid. The centroid of a triangle and tetrahedron can be calculated by using the nodal coordinates \((x,y,z)\) or vertices that define them. Hence,

\[ C = \left( \frac{1}{nnpe} \sum_{i=1}^{nnpe} x_i, \frac{1}{nnpe} \sum_{i=1}^{nnpe} y_i, \frac{1}{nnpe} \sum_{i=1}^{nnpe} z_i \right) \]  

(2.84)

where \( nnpe \) makes reference to the number of nodes that define the element (i.e. \( nnpe=3 \) for linear triangles).
Table 2.4 Integration formulae for triangular and tetrahedral regions (Zienkiewicz, 2005)

<table>
<thead>
<tr>
<th>Region</th>
<th>Integration Formulae</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangular</td>
<td>$\iint_{A_e} dx dy = A_e \ (Equation \ 2.29)$</td>
</tr>
<tr>
<td></td>
<td>$\iint_{A_e} x dx dy = \iint_{A_e} y dx dy = 0$</td>
</tr>
<tr>
<td></td>
<td>$\iint_{A_e} x^2 dx dy = \frac{A_e}{12} (x_1^2 + x_2^2 + x_3^2)$</td>
</tr>
<tr>
<td></td>
<td>$\iint_{A_e} y^2 dx dy = \frac{A_e}{12} (y_1^2 + y_2^2 + y_3^2)$</td>
</tr>
<tr>
<td></td>
<td>$\iint_{A_e} xy dx dy = \frac{A_e}{12} (x_1y_1 + x_2y_2 + x_3y_3)$</td>
</tr>
<tr>
<td>Tetrahedral</td>
<td>$\iiint_{V_e} dx dy dz = V_e \ (Equation \ 2.18)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} x dx dy dz = \iiint_{V_e} y dx dy dz = \iiint_{V_e} z dx dy dz = 0$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} x^2 dx dy dz = \frac{V_e}{20} (x_1^2 + x_2^2 + x_3^2 + x_4^2)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} y^2 dx dy dz = \frac{V_e}{20} (y_1^2 + y_2^2 + y_3^2 + y_4^2)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} z^2 dx dy dz = \frac{V_e}{20} (z_1^2 + z_2^2 + z_3^2 + z_4^2)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} xy dx dy dz = \frac{V_e}{20} (x_1y_1 + x_2y_2 + x_3y_3 + x_4y_4)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} xz dx dy dz = \frac{V_e}{20} (x_1z_1 + x_2z_2 + x_3z_3 + x_4z_4)$</td>
</tr>
<tr>
<td></td>
<td>$\iiint_{V_e} xy dx dy dz = \frac{V_e}{20} (z_1y_1 + z_2y_2 + z_3y_3 + z_4y_4)$</td>
</tr>
</tbody>
</table>

2.6 Global Finite Element Formulation

2.6.1 Assembling

In the prior section, three basic equations were established for a single element: element stiffness matrix, element mass matrix and element load vector. Moreover, it was discussed that the solutions for the governing equations over the entire domain may be calculated by assembling the equations of each element contained within the product.
This assembling procedure combines the temperature approximation in each element to form a piecewise temperature approximation over the entire domain. This can be accomplished by considering that the temperature at a node must be the same for each element that shares that particular node. After assembly of the mass matrix, stiffness matrix and global vector, the global system of equations will have the form presented in Figure 2.9.

![Figure 2.9 Global system of equations: Assembling element equations into a global system containing the global mass matrix [M], global stiffness matrix [K] and global load vector {F}.](image)

The final element method was used to transform a complex governing equation containing continuous partial differential equations that represented the heat transfer phenomenon in three-dimensional meat products (Equation 2.7), to a fairly simple matrix system of ordinary differential equations (Figure 2.8).
2.6.2 Transient Solution

To analyze time dependent temperature changes during thermal processing of meat products, the global systems of equations presented in Figure 2.8 must be integrated over time. The finite element method guarantees that the solution of the governing equation will account for the specific product shape, the basic principles of heat transfer and the required boundary conditions. However, finite element method is not practical for time analysis (Negahban, 2009b). Hence, it is necessary to combine finite element analysis with another numerical method that easily allows studying the solution over time. Finite difference analysis provides fundamental techniques to solve systems of first order ordinary differential equations such as the ones presented in the previous section (Figure 2.8). There are several techniques that can be used for integrating first order differential equations in time such as Euler methods, Runge-Kutta methods, $\theta$-methods, $\Delta t$ methods and others. However, the most common methods used for solving transient problems involving ordinary differential equations are the $\theta$-methods. This group of methods considers the following general approximation: an ordinary equation with the form

$$\frac{dT}{dt} = f(T)$$  \hspace{1cm} (2.85)

can be approximated to,

$$\frac{T_n - T_{n-1}}{\Delta t} = \theta f(T_n) + (1 - \theta) f(T_{n-1})$$  \hspace{1cm} (2.86)

To use this method, the global systems of equations must be reorganized as shown in the following figure:
Consequently, the global system of equations can be approximated to

$$
[M] \left( \frac{T_n - T_{n-1}}{\Delta t} \right) = \theta \{F\}_n - [K]_n \{T\}_n + (1 - \theta) \{F\}_{n-1} - [K]_{n-1} \{T\}_{n-1}
$$

(2.87)

As it can be observed in equation 2.86, three new terms are introduced in the global system of equations: $\Delta t$, $T_{n-1}$ and $\theta$. Since this method is incremental, the time step $\Delta t$ must be defined. $T_{n-1}$ represents the state of the response variable during the previous increment. Hence, an initial temperature distribution or initial condition is required for the first increment. The third term, $\theta$, serves as a weighting factor for the terms in the transient equation.

$\Delta t$ and $\theta$ highly influence the accuracy of the solution. Values of $\theta < 0.5$ make the solution conditionally stable. Thus, the solution would be stable only if the time step complies with some conditions. Values of $\theta \geq 0.5$ make the solution unconditionally stable; although, in general, smaller time steps lead to more accurate solutions (Olson and Negahban, 2007). However, a smaller time step requires a higher number of increments for studying the transient response of temperature; resulting in longer computing times that require to higher computational power. Hence, a proper time integration scheme should be selected in order to achieve a balance between accuracy and computing time.
This work will consider a value of $\theta = 0.5$, also known as the Crank-Nicholson scheme, which is unconditionally stable and the recommended scheme for high accuracy solutions.

### 2.7 Solving

Subsequently, equation 2.87 must be reorganized as shown in Figure 2.10, so that unknown terms will be on the left hand side of the equation and known terms will be on the right hand side of the equation.

![Figure 2.11 Reorganizing the global system for the solving step.](image)

Figure 2.11 contains three fundamental parts of a transient finite element analysis including a transient stiffness matrix that is multiplied by a vector representing the unknown nodal temperatures; this product is equal to a transient load vector containing all the known values and nodal temperatures of the previous increment. As a result, the transient form of the governing equation can be written in the following matrix system of linear equations:

$$[\bar{K}]{\bar{T}_n} = \{\bar{F}\} \quad (2.88)$$
This matrix representation of a linear system of equations corresponds to the transient final element formulation. Nodal temperatures per increment can be easily found using linear algebra algorithms such as Gaussian elimination, Cholesky decomposition, matrix factorization, conjugate gradients and other techniques. Equation 2.88 can be solved recursively until certain condition becomes valid (i.e. maximum product temperature is equal to 4.4°C).

3. SUMMARY

Modeling heat transfer of realistic meat processing scenarios may result in complex mathematical models. This is one of the reasons why many of the heat transfer models available for the meat industry include several simplifications or have been developed using proprietary software. However, these strategies are not practical for the meat processing industry. While models containing critical simplifications, such as product geometry simplifications, have a limited use in real processing situations as they may lead to inaccuracies that may cause serious consequences when used in evaluating safety of the product. However, models developed using proprietary software usually remain in scientific literature and are not readily available for meat processors. This manuscript describes a step-by-step process of how three-dimensional finite element analysis can provide the tools to transform those models that contain complex partial differential equations into models that can be easily solved without requiring licensed software. A finite element analysis can be performed by following six key steps: heat
transfer model definition, derivation of weak formulation, dicretization, assembling, derivation of transient formulation and solving.

The general model definition of heat transfer in thermal processing of ready-to-eat meats (Step one) considers that the thermal energy transfer takes place mainly by conduction. This model is subject to convective, radioative and evaporative boundary conditions; and to an initial condition associated with the initial temperature distribution of the product. The conduction heat transfer phenomenon can be described by Fourier's law, in which the heat flux is the product of material's thermal conductivity times the negative temperature gradient. The thermal conductivity and other thermo physical properties of a ready-to-eat meat product can be estimated as a function of the composition and temperature of the product, using empirical correlations proposed by Choi and Okos (1986). The convective and radiative boundary conditions consider a heat flux normal to the surface that can be represented by Newton's law of cooling. This law states that the heat flux is equal to a heat transfer coefficient, multiplied by the driving force which is associated with the temperature difference between the average surface temperature of the product and the air temperature. The heat transfer coefficient can be considered as a combined convective and radiative heat transfer coefficient. It is calculated by adding the value of the radiative and convective heat transfer coefficients. The radiative heat transfer coefficient is a function of the average surface temperature of the product, the air temperature, the emissivity of the product and the Stefan-Bolzmann constant. In addition, the convective heat transfer coefficient includes the effects of both natural and forced convection. This coefficient is approximated using techniques described in previous models by Davey, Pham, Wang and Amezquita (Amézquita, et al.,

The weak formulation (step two) is obtained by transforming the partial differential equations stated in the model definition, to a series of integral equations over the volume and surface of the product. These integrals are the key for the discretization step (step three). In step three, the domain and surface of the product are broken into simple regions represented by linear tetrahedral and triangular elements, using mesh generation tools. As these elements are fairly simple, integrals over their region can be easily computed using numerical integration techniques. Therefore, the integrals over the product volume presented in the weak formulation are approximated as the summation of the integrals over each simple element. Integrals over each element can be grouped into three basic equations: an element mass matrix, an element stiffness matrix and an element load vector. The equations of each of the elements are assembled to obtain a global system of equations in which the unknown variables correspond to the temperature at each of the nodes used to define the elements. This global system consists of a global mass matrix, a global stiffness matrix and a global load vector (step four). The global mass matrix indicates how the rate of increase of energy is related to the temperature of the product. The element stiffness matrix represents the effect of heat conduction through the product. The element load vector accounts for the heat transfer on the product surface.
The global mass matrix contains the key equations to evaluate the transient response of the model (step five); which can be formulated using finite difference analysis. The output of the transient formulation consists of a transient global system of equations containing a transient global mass matrix, a transient global stiffness matrix and a transient global load vector. This transient system of equations with the form $Ax=B$ can be solved (step six) using linear algebra algorithms. The unknown variable represents the nodal temperatures at a particular time step or increment.

The step-wise methodology presented in this chapter can be easily extrapolated to computational algorithms implementable in free license software (i.e. Java Technology). In addition, these algorithms can be integrated with predictive microbiology models; which can be particularly useful for evaluating the severity of thermal processing deviations caused by unexpected processing faults. Therefore, they can be the foundation of free-of-charge software packages which will serve as quantitative tools to support food safety management in the meat industry.

4. ACKNOWLEDGEMENTS

A contribution of the University of Nebraska Agricultural Research Division, supported in part by funds provided through the Hatch Act, USDA. Additional support was provided by the USDA-IREE-National Integrated Food Safety Initiative Competitive Grants Program (grant number: 2004-00739). Mention of a trade name, proprietary products, or company name is for presentation clarity and does not imply endorsement by the authors or the University of Nebraska.
5. REFERENCES


NOMENCLATURE

*Latin Letters*

\( a \)  Major semi-axis of the ellipsoidal body (m)

\( a_i \)  Constant of node number \( i \) used to calculate element shape functions

\( a_w \)  Water activity at the product surface

\( A \)  Area (m²)

\( b, d \)  Minor semi-axis of the ellipsoidal body (m)

\( b_i \)  Constant of node number \( i \) used to calculate element shape functions

\( B, C, \)  Boundary conditions

\( [B] \)  Matrix containing derivatives of nodal shape functions

\( [B^e] \)  Matrix containing derivatives of nodal shape functions of element \( e \)

\( C \)  Centroid of an element

\( C_p \)  Specific heat (J/kg °C)

\( c_i \)  Constant of node number \( i \) used to calculate element shape functions

\( D \)  Characteristic dimension (m)

\( \dot{E} \)  Flux of energy (W/m²)

\( e \)  Element number
\( \ddot{e} \) Eccentricity of prolate spheroids

\( \{F\} \) Global Load vector

\( \{F^e\} \) Load vector of element \( e \)

\( \{\ddot{F}\} \) Transient Global Load vector

\( f() \) Function

\( Gr \) Grashof number

\( h \) Heat transfer coefficient (W/m\(^2\) °C)

\( I.C. \) Initial condition

\( k \) Thermal conductivity (W/m °C)

\( \bar{k} \) Mass transfer coefficient (kg/Pa m\(^2\) s)

\( [K] \) Global Stiffness matrix

\( [K^e] \) Stiffness matrix of element \( e \)

\( [\ddot{K}] \) Transient Global Stiffness matrix

\( m \) Mass (kg)

\( [M] \) Global Mass matrix

\( [M^e] \) Mass matrix of element \( e \)

\( ne \) Total number of elements

\( nip \) Number of integration points

\( N \) Nodal shape functions of the finite element analysis
\{N\} \quad \text{Vector of nodal shape functions}

\{N_e^e\}^T \quad \text{Vector of transposed nodal shape functions at element } e

\text{Nu} \quad \text{Nusselt number}

L_i \quad \text{Volume (i.e. tetrahedral elements) or area (i.e. triangular elements) coordinates of local node } i

\text{p} \quad \text{Saturation pressure of water vapor in air (Pa)}

P_i \quad \text{Integration point } i

\text{Pr} \quad \text{Prandtl number}

q \quad \text{Heat flux (W/m}^2\text{)}

\text{Ra} \quad \text{Rayleigh number}

\text{Re} \quad \text{Reynolds number}

\text{RH} \quad \text{Relative humidity}

\text{t} \quad \text{Time (s)}

T \quad \text{Temperature (°C)}

\{T\} \quad \text{Vector of nodal temperatures}

\{T^e\} \quad \text{Vector of nodal temperatures at element } e

\text{u} \quad \text{Air velocity (m/s)}

\text{V} \quad \text{Volume (m}^3\text{)}

\text{x} \quad \text{X line axis in Cartesian coordinate system (m)}

\text{X} \quad \text{Maximum distance of x-axis (m)}
\(\hat{x}\)  Weight fraction of a component

\(\hat{v}\)  Volume fraction of a component

\(y\)  Y line axis in Cartesian coordinate system (m)

\(Y\)  Maximum distance of y-axis (m)

\(z\)  Z line axis in Cartesian coordinate system (m)

\(Z\)  Maximum distance of z-axis (m)

**Greek Letters**

\(\Delta\)  Divergence

\(\varepsilon\)  Emissivity

\(\lambda\)  Latent heat (J/kg)

\(\mu\)  Viscosity (Pa s)

\(\theta\)  Parameter for calculating transient response (\(\Theta\)-methods)

\(\rho\)  Density (kg/m\(^3\))

\(\sigma\)  Stefan-Boltzmann constant (5.676 \(\times 10^{-8}\) W/m\(^2\) K\(^4\))

\(\xi\)  Random error

**Superscripts**

\(e\)  Element

\(T\)  Transposed
**Subscripts**

<table>
<thead>
<tr>
<th>Subscript</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Initial</td>
</tr>
<tr>
<td>a</td>
<td>Air or ambient</td>
</tr>
<tr>
<td>ash</td>
<td>Ash</td>
</tr>
<tr>
<td>avg</td>
<td>Average</td>
</tr>
<tr>
<td>be</td>
<td>Boundary element</td>
</tr>
<tr>
<td>carb</td>
<td>Carbohydrates</td>
</tr>
<tr>
<td>conv</td>
<td>Convection</td>
</tr>
<tr>
<td>e</td>
<td>Element number</td>
</tr>
<tr>
<td>ellip</td>
<td>Ellipsoid</td>
</tr>
<tr>
<td>evap</td>
<td>Evaporation</td>
</tr>
<tr>
<td>f</td>
<td>Film</td>
</tr>
<tr>
<td>fc</td>
<td>Forced convection</td>
</tr>
<tr>
<td>fat</td>
<td>Fat</td>
</tr>
<tr>
<td>g</td>
<td>Generated</td>
</tr>
<tr>
<td>hr</td>
<td>Combined convection and radiation</td>
</tr>
<tr>
<td>in</td>
<td>Entering, internal</td>
</tr>
<tr>
<td>i</td>
<td>Component or node number</td>
</tr>
<tr>
<td>K</td>
<td>Absolute temperature in Kelvin</td>
</tr>
<tr>
<td>ŋi</td>
<td>Normal to the surface</td>
</tr>
</tbody>
</table>
\( nc \)  Natural convection

\( n \)  Increment or time-step number

\( out \)  Going out, external

\( prot \)  Protein

\( rad \)  Radiation

\( s \)  Surface

\( st \)  Stored

\( v \)  Vaporization

\( w \)  Water or moisture
CHAPTER III

Modeling Heat Transfer during Cooling of Ready-to-Eat Meats: Validation in Meat Processing Facilities

This paper will be published as:


- Journal of Food Engineering
Modeling Heat Transfer during Cooling of Ready-to-Eat Meats: Validation in Meat Processing Facilities

ABSTRACT

A three-dimensional finite element model for simulating heat transfer during cooling of irregular-shaped meat products was developed and validated. The heat transfer model considers conduction as the governing equation, subject to combined convection, radiation and evaporation boundary conditions. A finite element algorithm developed in Java was used to solve the model. This algorithm uses 4-node tetrahedral volume elements and 3-node triangular boundary elements. Product geometries were generated from CT-scan images of the meat products. The model validation was conducted in four different meat processing facilities, under real time-varying processing conditions. The model was adapted to receive input parameters that can be provided by meat processors such as relative humidity, air temperature, air velocity, type of casings, duration of water showers, product weight and core temperature prior entering the chiller. The average deviation between the observed and predicted values was 1.2 °C for core temperatures; 1.7 °C for temperatures 5.08 cm from core; and 2 °C for surface temperature. The developed heat transfer model can be integrated with predictive microbiology models; which can be particularly useful for evaluating the severity of thermal processing deviations caused by unexpected processing disruptions. This integration can be the foundation for open source software packages which will serve as quantitative tools to support food safety management in the meat industry.
1. INTRODUCTION

Meat processors are required to meet the stabilization performance standard established by the United States Department of Agriculture - Food Safety and Inspection Service (USDA-FSIS). The performance standard requires that ready-to-eat (RTE) meat and poultry processors must cool the cooked RTE meat and poultry products to prevent the germination and grow of toxigenic microorganisms such as Clostridium botulinum, and limit the growth of Clostridium perfringens to 1 log₁₀ (USDA-FSIS, 1999). As it is well known, cooling is a critical step during processing of ready-to-eat meats. If the cooked RTE meat and poultry products are cooled properly, potential growth of spore-forming bacteria that survive the heat treatment can be prevented. The stabilization compliance guidelines suggest that the risk of potential growth of spore-forming microorganisms might be significantly reduced by cooling products using one of following three processing methods (USDA-FSIS, 1999):

i. Maximum product temperature should drop from 54.4 to 26.6°C (130 to 80°F) within less than 1.5 hours; and must reach 4.4°C (40°F) within the next 5 hours. Thus, the total chilling operation cannot last any longer than 6.5 hours.

ii. Cooling step begins within 90 minutes after the heat treatment, the product should be cooled from 48 to 12.7°C (120°F to 55°F) in less than 6 hours; then, the cooling should continue until the product core temperature reaches 4.4°C (40°F).

iii. This method can only be used for processing of cured products containing at least 100-ppm in-going sodium nitrite. For these products, the maximum internal temperature should decrease from 54.4°C to 26.6°C (130 to 80°F) in less than 5
hours, and from 26.6 to 7.2°C (80 to 45°F) within the next 10 hours; resulting in up to 15 hours of cooling time.

Thermal processing deviations can occur during meat processing. These deviations imply that the cooling portions of thermal process employed do not follow the time-temperature schedules as required in the guidelines; such deviations generate uncertainty regarding the safety of finished products. When these deviations occur, sampling for microbial analysis may not be the best alternative to determine the safety of the product as distribution of microorganisms is not uniform. The USDA-FSIS recommends computer modeling as a tool to determine the severity of the thermal process deviation (i.e. heating and cooling) (USDA-FSIS, 1999).

Mathematical model to predict microbial growth in foods are currently available. In general, these models require the product time-temperature distribution as an input, as temperature is an important extrinsic factor influencing the growth of microorganisms in foods. Some processors may have the possibility of providing this data as they have tools available to record time-temperature data of their products during the cooling process. Nevertheless, most of these processors only record data at a single point considered as the worst case scenario. This point is usually assumed to be located in the center of the thickest part of the largest product being processed in a particular rack. This strategy is reasonable for some meat products and processing techniques; but there are uncertainties in determining the exact location of this point, especially when irregular-shaped products are involved or when the process itself causes additional variations (i.e. hot/cold spots in the chamber, irregular air flow around products). Moreover, a data recording process requires not only the use of accurate equipment, but precise placement of sensors. As it
will be shown later in this manuscript, it has been observed that not all meat processors are able to fulfill these requirements.

In the last decade, several heat and mass transfer models for simulating cooling profiles of meat products have been developed. Examples of such models include:

- A two-dimensional axis-symmetric finite element model for cooling of ellipsoidal-shaped meat developed by Amezquita et al. (2004 and 2005). It was one of the first heat and mass transfer models validated in the industry under real processing conditions. Moreover, this model was integrated with a predictive microbiology model to estimate the growth of *Clostridium perfringens* during cooling of cooked boneless ham (Amézquita, 2004, Amézquita, et al., 2005b).
- A three-dimensional finite element model for heat transfer in large brick-shaped meat products during air blast cooling (Wang and Sun, 2002a).
- A transient, axisymmetric two-dimensional heat conduction problem developed by Wang et al. (2006) for the validation of safe air-blast chilling of ellipsoidal-shaped cooked hams (Wang, et al., 2006).
- A transient three-dimensional finite element model to simulate heat transfer in spheres, cylinders and simple irregular geometries (Santos, et al., 2008).

Heat transfer models frequently use some type of simplification that limits their applicability. Two of the most common simplifications include:

- Product shape simplifications in which the products are approximated as standard geometric shapes such as rectangular block, ellipsoids, cylinders and spheres.
• Physical three-dimensional problem commonly reduced to one or two-dimensions. Hence, heat transfer in the axial and/or longitudinal direction is neglected.

Those strategies are practical as they can reduce the complexity of the problem; minimizing the processing time. Also, they are valid for different meat products and processing conditions. Nevertheless, several products available in the industry cannot be simplified in this manner, as they are highly irregular in shape. Therefore, there is a need for heat transfer models to simulate cooling of irregular-shaped meat products.

Another issue frequently associated with heat transfer modeling is the fact that the model validations are performed under controlled experimental settings. Hence, most models are not actually applicable to real meat processing conditions.

The objective of the research is to validate a heat transfer model for simulating cooling of irregular-shaped meat products following cooking. The validation tests described in this chapter were carried out in different meat processing facilities. In addition, a brief review of the model definition and the three-dimensional finite element algorithm are provided. Additional details about the model definition (e.g. estimation of effect due to evaporation and the heat transfer coefficient) and additional details about the finite element analysis formulation used in this manuscript can be found in (Cepeda, et al., 2010b).
2. MATHEMATICAL MODEL

2.1 Governing Equation

As discussed in the model development work of (Cepeda, et al., 2010b), heat transfer during cooling of ready-to-eat meats is governed by conduction, with no internal heat generation. Also, thermal energy is transferred in the unsteady state in which the temperature changes as a function of both time and location. Therefore, the governing equation of three-dimensional heat transfer in Cartesian coordinates can be represented as (Equation 3.1):

\[
\frac{\partial q}{\partial x} + \frac{\partial q}{\partial y} + \frac{\partial q}{\partial z} = \rho C_p \frac{\partial T}{\partial t}
\]

where,

\[
q = -k \nabla T
\]

Equation 3.1 is subject to the initial and boundary conditions described in the following sections.

2.2 Initial Conditions

A common practice in the meat industry consists in showering the meat products with cold tap water immediately after removing from an oven or smokehouse before entering to the chilling room. This strategy minimizes the weight loss caused by evaporation from the product surface during the cooling operation (Amézquita, 2004). Consequently, the initial temperature distribution within the product is a function of location, as shown in Equation 3.3.
\[ T = T_o(x, y, z) \quad \text{for} \quad 0 < x < X, \quad 0 < y < Y \quad \text{and} \quad 0 < z < Z; \quad \text{at} \ t=0 (3.3) \]

### 2.3 Boundary Conditions

During cooling of meat products, thermal energy is transferred from the product surface to the air by convection, radiation and evaporation. The effect of convection \( q_{\text{conv}} \) and radiation \( q_{\text{rad}} \) can be accounted by using Newton's law of cooling, Equation 3.4; with a combined convective and radiative heat transfer coefficient \( h \).

\[ q_{\text{conv}} + q_{\text{rad}} = h(T_s - T_a) \quad (3.4) \]

The effect of heat loss due to evaporation can be represented by Equation 3.5 (Amézquita, et al., 2005a, Chuntranuluck, et al., 1998a, Davey and Pham, 1997). This equation includes the latent heat of evaporation as a function of temperature; multiplied by the evaporative weight loss rate per unit area. This term does not consider water transport from the product core. Moreover, it does not include internal moisture evaporation.

\[ q_{\text{evap}} = \lambda_v(T_f) \frac{dm_w}{dt} \quad (3.5) \]

### 2.4 Estimation of Thermal Properties

Air properties including thermal conductivity, specific heat, viscosity and density can be estimated from empirical correlations and tabulated values previously reported (Amézquita, 2004, Kays, et al., 2005). Moreover, it has been shown that thermal properties of ready-to-eat meats can be estimated using the Choi and Okos empirical

2.5 Estimation of Heat Transfer Coefficient

As it was previously discussed in section 2.3, the heat transfer coefficient used in this model also accounts for the combined effect of convective \( h_{\text{conv}} \) and radiative \( h_{\text{rad}} \) heat transfer. Hence,

\[
h = h_{\text{rad}} + h_{\text{conv}} \tag{3.6}
\]

Previous studies have effectively estimated the radiative heat transfer coefficient using the following equation (Amézquita, 2004, Geankoplis, 2007, Wang and Sun, 2002b):

\[
h_{\text{rad}} = \sigma \varepsilon (T_{K,S} + T_{K,a})(T_{K,S}^2 + T_{K,a}^2) \tag{3.7}
\]


\[
h_{\text{conv}} = \sqrt[3]{h_{fc}^3 + h_{nc}^3} \tag{3.8}
\]
3. MATERIALS AND METHODS

3.1 Data Collection Process

3.1.1 Data Acquisition System and Instrumentation

The data acquisition system (DAQ system) was enclosed in a plastic box with a latched lid. The box was insulated with extruded polyethylene (250 XPS, FOAMULAR®) to insulate the equipment inside the industrial chillers. The DAQ system consisted of a chassis (SCXI-1000, National Instruments™) that contained a 32-channel thermocouple amplifier (SCXI-1102, National Instruments™) module attached to a terminal block (SCXI-1303, National Instruments™). A 16-Bit DAQ card (DAQCard-AI-16XE-50, National Instruments™) was used to transfer the data from the chassis to a laptop computer. A LabVIEW 6.1 virtual instrument application was programmed to read and record data from the 32-channels, and a serial port in real time.

3.1.2 Instrumentation

Different sensors were used to collect data of operating conditions and product temperature distribution. In total, 32 sensors were connected to the amplifier module for data collection including an anemometer, a relative humidity transducer and 30 thermocouples type-T. A top balance was connected via serial port to the laptop to record dynamic weight loss. The following represent the specific sensors used for data collection:

- FMA-904-V air velocity transducer (Omega Engineering, Inc., Stamford, CT, USA)
• HX303V RH/Temperature transmitter (Omega Engineering, Inc., Stamford, CT, USA)

• Top-loading balance TR-series (Denver Instrument Company, Arvada, CO, USA).

• Three flexible multisensor probes (IT-17:3). Each probe consisted of 3 type-T thermocouples spaced at 2 cm from each other; inside a 17 ga Teflon® sheath (Physitemp Instruments, Inc., Clifton, NJ, USA)

• A flexible multisensor probe (IT-17:5) consisted of 5 type-T thermocouples spaced at 2.54 cm from each other; inside a 17 ga Teflon® sheath (Physitemp Instruments, Inc., Clifton, NJ, USA)

• A flexible multisensor probe (IT-17:7) consisted of 7 type-T thermocouples spaced at 2.54 cm from each other; inside a 17 ga Teflon® sheath (Physitemp Instruments, Inc., Clifton, NJ, USA)

• Four needle thermocouples type-T for measuring surface temperature (Omega Engineering, Inc., Stamford, CT, USA)

• A rigid multisensor probe consisting of 6 thermocouples inside a 3.18 mm diameter 316 stainless steel sheath (Omega Engineering, Inc., Stamford, CT, USA).

Figure 3.1 shows the typical equipment layout that was used during the data collecting process.
3.2 Validation in Meat Processing Facilities

3.2.1 Meat Products

Time-varying temperature distributions of seven meat products were recorded under industrial cooling operations. Model validations were carried out in four different meat processing facilities. The meat products evaluated included fully-cooked oven-roasted turkey breast, roast beef, and boneless ham with water added. The weight of the products ranged from 3.75 kg to 7.4 kg. Proximate composition (i.e. fat, moisture, protein and ash) of the products were assumed as the compositions reported by the processors on the retail product labels. All products, except for product used for the Test#1, had casing to maintain structural integrity of product and to reduce weight loss during processing. Turkey breast products were vacuum packaged in plastic casings; and boneless ham products had smoke-impregnated fabric casings. Product used for Test#1 was contained
in a polyester elastic net with coarse openings. Details about type of product, proximate composition, weight and associated processing plant with each test are presented in Table 3.1.

Table 3.1 Description of type of products used for the model validation in different meat processing facilities.

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Processing Plant</th>
<th>Product Type</th>
<th>Casing Type</th>
<th>Initial Weight wb (kg)</th>
<th>Proximate Composition (%)</th>
<th>Moisture</th>
<th>Fat</th>
<th>Protein</th>
<th>Ash</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B</td>
<td>Roast beef</td>
<td>Net</td>
<td>7.4</td>
<td>61.1, 12.4, 25.0, 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>C</td>
<td>Turkey breast</td>
<td>Plastic</td>
<td>4.9</td>
<td>78.7, 0.9, 19.6, 0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C</td>
<td>Turkey breast</td>
<td>Plastic</td>
<td>5.0</td>
<td>78.7, 0.9, 19.6, 0.3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>D</td>
<td>Turkey breast</td>
<td>Plastic</td>
<td>3.7</td>
<td>61.1, 12.4, 25.0, 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>D</td>
<td>Turkey breast</td>
<td>Plastic</td>
<td>3.6</td>
<td>61.1, 12.4, 25.0, 1.0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>Ham-water added</td>
<td>Fabric</td>
<td>5.8</td>
<td>73.3, 6.0, 17.1, 3.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>Ham-water added</td>
<td>Fabric</td>
<td>6.3</td>
<td>73.3, 6.0, 17.1, 3.7</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.2.2 Operating Conditions for Cooling

Cooling conditions for each test were set by each processor as normally used. In general, products were cooked in industrial ovens to a core temperature between 61 °C and 73 °C prior to cooling, depending on the product. All products, except for the product used in Test#1, were exposed to cold-tap water showers after cooking to reduce the weight loss due to water evaporation during cooling. This procedure was performed inside the ovens prior to removal with duration ranging from 15 to 30 minutes. After showering, products were placed in chilling rooms. Evaporator units inside all the chillers were ceiling-mounted; except for Test#4 and Test#5 in which the chiller had
floor-mounted units. Racks layout inside the chilling chambers was determined by the processor. Products were cooled on the same racks used during the cooking process. Initial weight of each product was determined by removing each product from its respective rack, placing it on a top-loading balance, recording the weight and then returning each product to its rack. Prior to return to the rack, each product had between 14-20 thermocouples inserted in different randomly selected locations throughout the product. For the validation tests, some products were located on top of the rack and other products were placed in the middle of the rack. Air temperature, air velocity and relative humidity were also monitored during the cooling operation. Sensors for the air properties were attached to each rack with monitored product, and placed 0.3-1.0 m away from the products being tested. Temperature distribution throughout the product, and air property values were collected every 10 seconds from beginning to end of the cooling operation as determined by each processor. Operating conditions used during the data collecting process for each validation test are summarized in Table 3.2.
Table 3.2 Operating conditions used for the validation tests.

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Shower Time (min)</th>
<th>Initial Temperatures (°C)</th>
<th>Average Operating Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Core</td>
<td>Surface</td>
</tr>
<tr>
<td>1</td>
<td>No shower</td>
<td>63.9</td>
<td>41.0</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>65.7</td>
<td>18.0</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>70.0</td>
<td>13.1</td>
</tr>
<tr>
<td>4</td>
<td>15</td>
<td>72.8</td>
<td>37.4</td>
</tr>
<tr>
<td>5</td>
<td>15</td>
<td>73.5</td>
<td>38.3</td>
</tr>
<tr>
<td>6</td>
<td>20</td>
<td>72.1</td>
<td>26.5</td>
</tr>
<tr>
<td>7</td>
<td>20</td>
<td>71.2</td>
<td>23.4</td>
</tr>
</tbody>
</table>

3.3 Numerical Solution

A three-dimensional finite element algorithm was implemented in Java™ (Version 6, update 23, Sun Microsystems, 2010). The algorithm followed the Galerkin procedure described in Cepeda, et al. (2010b). It used linear tetrahedral elements and 3-node triangular boundary elements. The inputs for the Java application include:

- Product mesh organized as a text file or spreadsheet file.
- Product characteristics including proximate composition, initial weight, initial core temperature, initial surface temperature and target core temperature.
- Operating conditions including time-varying or average air temperature, air velocity and relative humidity.
Validations of predicted temperatures for each cooling test included development of three different meshes (i.e. normal, fine and finer). Also, different combinations of mesh refinements, and time-step for the iterations of the finite element algorithm (i.e. 30s, 100s, 300s and 900s) were evaluated to assess the solution convergence. Additional details about the implementation of the open-source Java application can be found at Cepeda, et al. (2010a).

### 3.4 Meshing

A computed tomography (CT) scan at 1-mm interval scans on transversal plane was taken for each meat sample used for the validation. CT-scans were completed in a multi-detector 16 Slice CT- scanner (Professional Medical Imaging, Lincoln, NE) after cooling the meat samples in industrial chillers. The CT-scan images (i.e. DICOM images) were used to reconstruct the three-dimensional geometry in Materialise Mimics 14.0 (Figure 3.1). Meshes consisted of linear tetrahedral elements (i.e. 4-node tetrahedral elements) and 3-node triangular boundary elements. Additionally, Materialise 3-matic 5.1 software was used for improving the surface mesh, reducing the number of triangles and evaluating the mesh quality. Different meshing quality parameters were controlled to preserve equilateral triangles in the meshes. These parameters included ratio of height/base equals to 0.3 m, maximum triangle edge length of 0.03 m, and aspect ratio < 100. Three meshes with different refinement level (i.e. normal, fine and finer) were developed per each product for posterior convergence analysis of the solution. The meshes were exported as text files (i.e. COMSOL Multiphysics .mphtxt file format) using
Materialise Mimics 14.0. Then, they were reorganized and saved as spreadsheets files that could be imported in the Java application.

3.5 Determination of Probe Locations

During the CT scans, some products still contained the thermocouple wires inserted prior to the data collection process in their original locations. Other products were scanned with metal pins inserted prior to scanning to indicate the thermocouple locations. Then, specialized software for visualizing medical digital images (i.e. Osirix DICOM viewer and Materialise Mimics) was used to determine the exact location of the thermocouples inserted in each meat product (see Figure 3.2). Probe locations were represented by (x,y,z) points located in a Cartesian plane under the same coordinate system employed by the software used to generate the product meshes.

**Figure 3.2** Location of thermocouples in a meat product using software for visualization of medical images.
3.6 Evaluation of Model Performance

The evaluation of model performance (MP) was carried out by comparing the deviation between the observed and predicted temperatures in a location at different times. This deviation was estimated by calculating the root mean square error, as shown in equation 3.9 (Amézquita, et al., 2005a, Huber-Carol, 2002):

\[
MP = \sqrt{\frac{\sum_{i=1}^{n} (\text{observed value} - \text{predicted value})^2}{n}}
\]  
(3.9)

The predicted temperature at the original location where a thermal probe was located was considered as the temperature of the closest node available to that location. This was determined on the basis of the Euclidean distance (ED), as shown in equation 3.10. Hence, the closest node was selected as the one providing the minimum relative distance.

\[
ED = \sqrt{(x_{probe} - x_{node})^2 + (y_{probe} - y_{node})^2 + (z_{probe} - z_{node})^2}
\]  
(3.10)

3.7 Initial Temperature Distribution

Temperatures at different locations within a product were recorded by placing multi-sensor probes across the products. The temperatures of a particular product recorded at the beginning of the process, \(T_0(x,y,z)\), were fitted to first and third order polynomial models using statistical software (i.e. SAS \(\text{®} 9.2\)) to estimate the initial temperatures at each node contained in the product mesh. Two different methods PROC
REG and PROC ROBUSTREG were used to obtain the polynomial regression parameters.

4. RESULTS AND DISCUSSION

4.1 Meshing and Simulation Time

Three-dimensional meshes containing 4-node tetrahedral elements and 3-node triangular boundary elements were created per each tested product. Figure 3.3 shows the meshes generated for Tests#2 to Test#5, corresponding to turkey breast products manufactured in processing plants B and C. Meshes used for simulations of Tests#1, Tests#6 and Tests#7 are shown in Figure 3.4. Meshes used during the simulations were selected based on quality, approximation to the real geometry and number of elements. The quality of the mesh was evaluated by analyzing the aspect ratio of its elements. This parameter represents the ratio of the maximal edge length to the height of the element. In general, a value below 100 is necessary for finite elements analysis. Hence, elements with an aspect ratio above 100 were classified as low quality elements. As shown in Table 3.3 meshes ranged from 3,208 to 11,567 volume elements (i.e. tetrahedral elements). The maximum number of low quality elements used during the simulation was 6 elements.
Table 3.3 Number of elements and low quality elements enclosed in the meshes used during the simulations.

<table>
<thead>
<tr>
<th>Test</th>
<th>Number of 3D-elements</th>
<th>Low Quality 3D-elements (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3,208</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>11,567</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>7,924</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>7,550</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>5,833</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>6,577</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>8,598</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>9,788</td>
<td>5</td>
</tr>
</tbody>
</table>

(1) Elements with aspect ratio > 100
Figure 3.3  Cross-sectional view of three-dimensional meshes created for validation Test#2-5: These meshes were generated from CT-scans using Materialise Mimics 14.0 and 3-matic v5.1
Figure 3.4 Cross-sectional view of three-dimensional meshes created for validation Test#1, Test#6 and Test#7. This meshes were generated from CT-scans using Materialise Mimics 14.0 and 3-matic v5.1
CT-scans were used to inspect the similarities between the cross-sectional images of the products and the mesh cross-sections. As shown in Figure 3.5, minor deviations between the surface of the product and the surface of the 3D-model generated were found. The outer sections of the geometries used in this study were within ±0.01m proximity to the actual product surface.

![Figure 3.5 Product geometry approximation. Illustration of deviations between the mesh surface for Test#1 (i.e yellow line) and the product actual surface (i.e. limit between the gray and black regions)](image)

Simulation times ranged from 0.4-hours to 2-hours with a time step of 300-s depending on the mesh refinement, product size, and cooling operating conditions. Although the simulation times could be clearly reduced by using coarser meshes (i.e. meshes with fewer elements), these meshes were not considered in this study because of the method used for estimating the temperature at an arbitrary point within the geometry. As shown in section 3.6, this method considers the predicted temperature at a point (i.e. x, y, z coordinates indicating the location within a product at which a thermocouple was placed) as the temperature of the closest node available to that location. This method would not be valid for coarser meshes as coarser meshes imply larger elements, and more dispersed nodes. Thus, the distance between the actual thermocouple location and the
closest node available may be large enough (i.e. >0.03m) to cause discrepancies in the predictions that are not necessarily associated with the model performance (see Figure 3.6).

**Figure 3.6** Potential prediction deviations caused by using coarser meshes with the method established to determine temperatures at arbitrary points (i.e. probes locations). In this tetrahedral element, node#4 represents the closest node to the arbitrary point \( p \). Thus, the predicted temperature at point \( p \) will be approximated to 32\(^\circ\)C; which causes a 11\(^\circ\)C deviation compared with the actual value. A better approximation can be obtained by either using finer meshes or by estimating the temperature at point \( p \) based on element nodal temperatures.

Therefore, a new methodology for estimating the temperature at an arbitrary location within a mesh must be established in order to be able to study the model performance using coarser meshes. The recommended method to overcome this issue consists in identifying the element that contains that arbitrary location. Then, the temperature at that exact location can be estimated based on the element nodal temperatures using its shape functions. This approach would provide more accurate predictions when working with coarser meshes, especially when predicting temperatures at locations nearby the product surface.
According to the simulations performed, the ideal conditions to execute the finite element algorithm implemented in Java were: iteration time-steps of 300 s and meshes containing less than 8,000 elements. These conditions provided a good balance between processing time and prediction stability (i.e. predicted temperature curves without fluctuations). Larger time steps may cause slight fluctuations in the predictions; especially when evaluating temperatures at points located on or nearby the product surface.

4.2 Initial Temperature Distribution

The initial temperature data for each node must be provided to solve the model. However, this temperature was measured only on 14-20 locations. In order to estimate the initial temperature at each node, polynomial models (i.e. $T_0(x,y,z)$) were fitted and tested. This estimation was very important as the majority of the products were exposed to tap-water showers. Hence, the temperature at the surface of a product is considerably lower than the core temperature.

The fitted polynomial models provided accurate estimations of the initial temperatures at the observed locations. However, noticeable variations were obtained when using these models to estimate the initial temperatures at other locations. For instance, temperatures higher than the core temperature and lower than the surface temperature are predicted with these polynomial models (e.g. negative temperatures). This indicates that the number of observed initial temperatures is not sufficient to fit a polynomial model that can provide good estimations of the initial temperature at every
single point in the geometry. Therefore, initial temperature distributions could not be provided to the model as an input.

However, the main objective of this work is to define a heat transfer model that can be actually used by the meat industry. In general terms, meat processors will not be able to provide the initial temperature distribution of their products as an input to the model. Hence, the developed heat transfer model accounted for the effect of non-homogeneous initial temperatures by including an initial basic heat transfer analysis during the water shower treatment (i.e. 15-30 min according to the process described in Table 3.2). This initial heat transfer analysis considered the effect of conduction within the product; convection and radiation from the product surface; and heat lost due to evaporation of the water used to shower the products. The following operating conditions and parameters were assumed:

- Maximum relative humidity (i.e. 100 %)
- Minimum air velocity (i.e. 0.001 m/s)
- Air temperature = 30 °C
- Homogeneous initial product temperature (i.e. core temperature reached during cooking)

Cooling simulations started right after the initial heat transfer analysis of the water shower treatment. It is important to notice that it took between 20-40 minutes to insert the probes into each tested product; and they were inserted inside a cooler. Thus, heat transfer within the product took place during that period of time and it could not be monitored with the thermocouples. This issue could have been avoided if the probes had
been placed into the products at the beginning of the thermal process (i.e. before cooking). Heat transfer during the placement of probes into the product was simulated using the average operating conditions recorded at that period of time. For validation purposes, the initial time of cooling used in this manuscript (i.e. time = 0 hours used in the graphs) corresponded to the time at which the probes were inserted.

Figure 3.7 shows an example of the complete heat transfer simulation performed for Test #2. As shown in the figure, an initial heat transfer analysis was performed to account for heat transfer during 20 min of water shower (A). Subsequently, the regular cooling simulation took place (B). The first 40 min of cooling simulation corresponded to the time elapsed when placing the probes. Then, the cooling simulation continued using the time-variable processing conditions recorded during the process.

The proposed methodology for accounting heat transfer during the water showers allowed obtaining rough estimates of the non-homogeneous initial temperatures of the product prior entering the cooler without the need for product-specific fitted polynomial models. This methodology was suitable for all the tested products. The average deviation between the initial temperature predicted and observed was 1.7 °C. Nevertheless, further research and experimental data is required to improve the estimation of initial temperature distributions of meat products as a function of the duration of water shower treatments. In addition, future validations of the model require sensors to be placed into the meat products prior to cooking.
4.3 Analyzing the Effect of Evaporation

As it was shown in the model definition (see Section 2.3), this heat transfer model considers the effect of evaporation on cooling of meat products. It was observed that the estimation of this effect is sensitive to two factors: air temperature profile provided to the model, use of casings, and initial temperature distribution of the product.

Figure 3.7 Illustration of the initial heat transfer simulation procedure used to estimate the initial temperature distribution of the meat products prior entering the cooler. This simulation corresponds to Test#2. For validation purposes, the initial cooling time (i.e. time=0 hours) is considered as the time at which sensors were placed within the product. (A) Initial heat transfer analysis during the water showers. (B) Cooling simulation.
4.3.1 Effect of chiller temperature

In an industrial meat chiller, air temperature distribution inside a chiller not only is time-dependant, but it changes with the location. During the first 2-4 hours of chilling, air temperature around the product surface is substantially higher than the overall air temperature of the chiller. Figure 3.8 shows a typical air temperature variation at two different locations inside the chiller. One location represents the temperatures recorded nearby the product surface (i.e. <0.2 m from the surface), and the other represents the overall air temperature profile of the chilling room (i.e. >1 m from the product surface); which corresponds to the air temperature profile commonly recorded by meat processors.

Figure 3.8 Illustration of variation between the temperature of the air surrounding the product and the chiller air temperature for Test#6.
A good approximation of the heat loss due to evaporation is achieved by providing an air temperature representing the temperature nearby the product surface. The method used in this model for accounting for heat flow due to water evaporation from the product surface, considers the latent heat of evaporation multiplied by the evaporative weight loss rate per unit area. This weight loss is proportional to the gradient of water-vapor saturation pressure calculated at the air temperature and at the product surface temperature. However, this air temperature makes reference to the temperature of the air surrounding the product; which is responsible for evaporative weight loss. As it is known, the water-vapor saturation pressure is inversely proportional to the temperature. Thus, considering lower air temperature values will increase the evaporative weight loss; miscalculating the heat loss by evaporation. As a result, precise judgment of the temperature of the air surrounding the product is also indispensable when considering evaporation effects.
Figure 3.9 Illustration of effect of different air temperature profiles (i.e. chiller air temperature and boundary air temperature) provided to the model during simulations that included the effects of evaporation. Data presented in the figure correspond to the first five hours of simulations for Test#7.

As it was previously discussed, air temperature profile recorded by meat processors makes reference to the overall chiller temperature rather than to the temperature profile of the air surrounding the product. Therefore, temperature of the air surrounding the product cannot be provided by a regular meat processor. The validations reported in this manuscript correspond to simulations using the air temperature as the chiller air temperature commonly recorded by a meat processor. A practical methodology to counteract the effects of using the overall chiller air temperature instead of surrounding air temperature is proposed in Section 4.3.2.
Further research should be aimed at determining practical methods to estimate the air temperature in the boundary layer that is created around the products during the first hours of cooling. For instance, a computational fluid dynamics model can be integrated to this finite element model to obtain the air temperature profiles around products based on different parameters such as airflow regime, product core and surface temperature, air temperature set point, room load, and rack layout.

4.3.2 Effect of casings and water showers

Numerous meat products are manufactured in special casings including fabric casings, plastic casing, collagen casings, natural casings and more. Casings serve for several purposes such as shaping, protection against potential contaminants, smoking and more. One important advantage of using casings is that they can increase the yield of the process by minimizing weight loss due to evaporation. A notable difference in weight loss percentage was observed in the products with casings. During the validation tests, products wrapped in casings (i.e. Test#2-7) lost in average 1.9% of the initial weight. In contrast, the product used for Test#1 that was contained in a net with wide openings lost 5.7% of its initial weight.
Figure 3.10 Comparison between predicted and observed temperatures at different locations for Test#1; considering water evaporation during the simulation.
Figure 3.11 Comparison between predicted and observed temperatures at different locations for Test#1; evaporation was not considered in the simulation.
Figures 3.10 and 3.11 show the comparison between two cooling simulations for Test#1. The product used for this validation test was a roast beef product, not exposed to water showers and contained in a net with large openings. Figure 3.10 presents a cooling simulation that considered the effect of heat loss due to evaporation during the cooling process. Figure 3.11 shows the same cooling simulation, but without considering water evaporation from the product surface during the cooling process. As it can be observed in the figures, neglecting heat losses due to evaporation in this product caused under prediction of the cooling rates. This suggested that evaporation effects are very important for products not wrapped in casings or contained in casings that do not reduce water evaporation (e.g. nets with large openings).

The effect of evaporation on the cooling simulations was considerably different in products with casings. Figure 3.12 and 3.13 show the same comparison previously presented, but using the product for validation Test#7. As opposed to the previous product, product used in Test#7 was contained in a fabric casing. These simulations did not accounted for non-homogeneous temperature distributions obtained after water showering.

Simulations indicated that the reduction in water loss associated to the use of casing, and the water showers treatment clearly affected the effect of evaporation on the model. On one hand, considering evaporation in this simulation resulted in over prediction of the cooling rates (Figure 3.12). On the other hand, not considering evaporation caused under prediction of the cooling rates (Figure 3.13). Same results were obtained for all the validation tests of products with casings (see Appendix A).
Figure 3.12 Comparison between predicted and observed temperatures at different locations for Test#7; considering water evaporation during the simulation.
Figure 3.13 Comparison between predicted and observed temperatures at different locations for Test#7; evaporation were not considered in the simulation.

It was observed that the considering evaporation during the simulations lead to more unsafe deviations, from the microbiological point of view, than not considering evaporation. In addition, considering evaporation provided better estimates of initial temperatures. This indicated that the effect of evaporation in products with casings can be important, but only at the beginning of the cooling process. As the cooling process continues, evaporation may be neglected for products exposed to showers and contained in casings (i.e. plastic and fabric casings).
Figure 3.13 shows an extra simulation of the cooling process in Test#7. This simulation did not consider evaporation inside the cooler as in the simulation shown in Figure 3.12, but it considered the effect of an initial water evaporation that took place during the water shower treatment (see Section 4.2). It was assumed that the water evaporated during this initial period did not correspond to the water contained in the product, but to the water used to shower the product. The inclusion of an initial evaporation provided better predictions for cooling profiles at different locations. Similar results were obtained for all the products with casings (see Appendix A).

Better estimates of initial temperature distributions within the product could be obtained by considering an initial evaporation that took place during the water shower step. Also, water evaporation during the cooling process was minimal for products with casings; thus, it could be neglected. Since water evaporation was not considered for these products, the model sensitivity to the air temperature provided for the simulation was noticeable reduced. Hence, the developed model provided good cooling temperature predictions by using only the air temperature profile that can be given by a regular meat processor (i.e. chiller air temperature profile). The minimal heat losses caused by moisture evaporation in products with casings can be accurately estimated by the model only if the temperature of the air surrounding the product is provided.

### 4.4 Using the Model

Table 3.14 shows a decision table used to adjust the model according to the input parameters that are available, the type of product and processing conditions. Simulations
for cooling of products exposed to water showers should include the temperature
distribution reached after the showers as an input, or it should include the effect of an
initial heat transfer analysis during the water shower treatment. Heat loss due to moisture
evaporation is only considered when temperature profile of the air surrounding the
product can be provided. Otherwise, it should be considered only for products that do not
have casing.

Table 3.4 Decision making table used to adapt the model to the inputs that can be provided by a meat processor

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product has casing</td>
<td>Y</td>
</tr>
<tr>
<td>Product was exposed to water showers and initial</td>
<td></td>
</tr>
<tr>
<td>temperature prior entering cooler is unknown</td>
<td>Y</td>
</tr>
<tr>
<td>Boundary air temperature is unkown, but chiller</td>
<td></td>
</tr>
<tr>
<td>air temperature is provided</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ACTIONS</th>
<th>RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consider evaporation during cooling</td>
<td>N</td>
</tr>
<tr>
<td>Consider intial evaporation during water shower time</td>
<td>Y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CONDITIONS</th>
<th>RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product has casing</td>
<td>Y</td>
</tr>
<tr>
<td>Product was exposed to water showers and initial</td>
<td></td>
</tr>
<tr>
<td>temperature prior entering cooler is unknown</td>
<td>Y</td>
</tr>
<tr>
<td>Boundary air temperature is unkown, but chiller</td>
<td></td>
</tr>
<tr>
<td>air temperature is provided</td>
<td>Y</td>
</tr>
</tbody>
</table>
4.5 Model Performance

4.5.1 Evaluation of Model Performance

The average model deviations (i.e. MP values analogous to root mean square error) between the observed and predicted values were 1.2 °C for core temperatures; 1.7 °C for temperatures at 5.08 cm from core; and 2 °C for surface temperature. MP values observed per product at different locations are presented in Figure 3.14.

Figure 3.14 Evaluation of model performance during different validation tests and locations.

Model deviations can be attributed to factors associated with the input parameters used during the simulations. Also, deviations can be consequence of model assumptions.
taken to provide estimates of cooling profiles at different points within a meat product only based on input parameters that can be provided by a meat processor. Such adaptations include the use of chiller air temperature, neglecting evaporation for cooling simulations of products with casings and initial basic heat transfer analysis to estimate initial temperatures after water shower treatments. As it was discussed in section 4.2 and 4.3, the effect of evaporation is highly affected by these factors, especially for predicting surface temperatures.

Maximum MP values obtained during the simulations were 1.9 °C for core temperatures; 2.2 °C for temperatures at 5.08 cm from core; and 3.9 °C for surface temperatures. Model prediction inaccuracies for surface temperatures are not important from the microbiological safety point of view.

This model is in good agreement with the observed values taking into account that the simulations were carried out including only parameters that can actually be provided by a meat processor. In theory, the accuracy of the predictions can be improved when a method for getting better estimations of initial temperatures within the product prior to cooling, and temperature distribution of the air surrounding the products are available.

4.5.2 Interpretation of Model Performance

Core temperature profile is of interest for the meat industry as it represents the worst case scenario for microbial contamination. As discussed in section 4.4.1, the maximum MP value for core temperatures was 1.9 °C. To interpret this value from the microbiological safety point of view, cooling profiles with ±1.9 °C (i.e. under and over
prediction of cooling rates) of deviation were used to estimate microbial growth for
*Clostridium perfringens* in boneless ham using the microbial growth predictive model
developed by Amézquita et al. (2005b). Products used for validations Test#1, Test#6 and
Test#7 were considered for the microbial growth prediction. Also, an initial population
of $2 \log_{10} \text{(CFU/g)}$ was considered for the microbial growth prediction.

Test#1 was considered as it was the largest product used during the validation;
also, because it was the one that presented the maximum deviation in core temperature
prediction. Figure 3.15 shows the effect of under and over prediction of the cooling rates
on the growth of *Clostridium perfringens* during Test#1. The maximum net growth
difference was $0.005 \log_{10} \text{(CFU/g)}$ which can be considered trivial from the food safety
Although Test#1 was the larger product analyzed during the validations and it presented the larger prediction deviations during the cooling simulations, it was cooled in less than 5 hours. Hence, the cooling process was safe and MP values of 1.9 °C did not affect the estimation of microbial growth.

Products used for validation Test#6 and Test#7 were also considered as they were large products (i.e. 5.8 and 6.3 kg) that took the longer time to cool (i.e. 14.8 hours).
Hence, deviations on cooling profile predictions may cause more evident deviations on microbial growth estimation. Figures 3.16 shows a comparison between the growth of *Clostridium perfringens* using the observed cooling profile, and hypothetical deviated cooling profiles (i.e. ±1.9 °C from the observed) for Test#6. The maximum deviations on growth prediction of *Clostridium perfringens* caused by MP values of 1.9 °C were 0.09 $\log_{10}$ (CFU/g) and 0.07 $\log_{10}$ (CFU/g), for Test#6 and Test#7 respectively. These deviations are not important from the microbial safety point of view. Hence, a MP value of 1.9 °C represents a good indicator of model performance.
Figure 3.16 Model prediction deviations and estimated growth of *Clostridium perfringens* during cooling for Test#6

4.6 Potential Applications

4.6.1 Evaluation of Cooling Deviations

The developed heat transfer model can be used to evaluate the effects of a cooling deviation. To illustrate this potential application, three different cooling processes for the product used in test#7 were simulated. All the simulations maintained the same simulating conditions except for the air velocity and temperature. Figures 3.17, 3.18 and 3.19 illustrate the simulation results. In the figures, a product center temperature history following exactly the FSIS compliance guidelines for cured products (i.e., 5 h from 54.4°C to 26.6°C, and 10 h from 26.6°C to 7.2°C) was provided as reference.
The first simulation considered an unexpected cooler downtime (e.g. due to failure in refrigeration equipment or electrical power outage) when the product reached 60 °C at the core. The downtime lasted for one hour, during which there was no air circulation, and which caused a hypothetical rise on the air temperature calculated as function of the average product temperature. Then, air velocity returns to the original value and chiller temperature slightly decreased until it reached the normal operating conditions.

The second simulation also considered an unexpected cooler downtime when the product reached 60 °C in core temperature, but this time the downtime lasted for six hours. Similar air velocity and temperature assumed for simulation 1 were considered.

The third simulation considered a case in which the chilling room temperature was considerably high at the beginning of the cooling process and slightly decreases until it reaches the normal conditions. This could have been caused due to overloading of the cooler, or to excessive air exchange resulting from long product loading times.
Figure 3.17 Simulation of cooling deviation in Test#6 caused by a supposed 1 h cooler downtime.
Figure 3.18 Simulation of cooling deviation in Test#6 caused by a supposed 6 h cooler downtime.
As it can be observed in Figure 3.17, a downtime of 1 h during the initial period of cooling caused a deviation of 3.2 h above FSIS compliance guidelines recommended for the 26.6-7.2°C period of cooling. This deviation was expected as air temperature is the most influencing factor for determining cooling rates as it directly affects the driving force that governs the heat transfer phenomenon during cooling of ready to eat meats. Thus, minor changes in air temperature lead to visible changes in the cooling profile.

**Figure 3.19** Simulation of cooling deviation in Test#6 caused by a supposed chiller overload.
Similar results were obtained from the second and third simulations. Since the air temperature deviations were more severe, drastic deviations from the FSIS guidelines were observed. A downtime of 6 hours did not cause a deviation during the 54.4-26.6°C period of cooling, but it caused a 17.4 h deviation above FSIS compliance guidelines during the 26.6-7.2°C period of cooling. Also, a initial chiller air temperature deviation lead to a 7.4 h deviation above FSIS compliance guidelines during the 26.6-7.2°C period of cooling.

4.6.2 Integration with Microbial Predictive Models

The developed heat transfer model can be integrated to dynamic microbial growth predictive models. For instance, the predicted temperature profiles resulting from a particular simulation could be used as inputs for microbial growth predictive models. This integration would be greatly valuable for qualitative analysis of food safety in the meat industry.

The developed heat transfer model was included into a prototype of an open source, user-friendly food safety website numodels4safety.unl.edu that provides online food safety tools for the meat industry. This website is designed for evaluating food safety on the processing of fresh poultry, beef and pork products; and ready-to-eat meat products. For instance, it can be used by small meat processors and FSIS inspectors to evaluate the safety of a meat product after unexpected processing faults such as cooling deviations. Also, it effectively integrates two of the most important issues of meat processing in regards to food safety: microbial growth and thermal processing. Hence, it
can serve as a quantitative tool to support food safety management strategies such as HACCP and microbiological risk assessment. The visual interface of a prototype version of the website is presented in Figure 3.20.

![Figure 3.20 Illustration of visual interface of a prototype version of the website numodels4safety.unl.edu](image)

5. CONCLUSIONS

A heat transfer model for simulating heat transfer during cooling of irregular-shaped meat products was developed and validated. The model considers heat conduction within the product subject to convection, radiation and evaporation...
boundaries. Also, it allows for time-varying processing conditions (i.e. relative humidity and air temperature). The model was validated in industrial meat processing facilities, under real processing conditions. The model was adapted to provide good estimations of cooling profiles only by considering parameters that can be easily provided by a meat processor. These inputs include relative humidity, air temperature, air velocity, type of casings, duration of water showers, product weight and core temperature prior entering the chiller. In general, simulations for cooling of products exposed to water showers should include the temperature distribution reached after the showers as an input, or it should include the effect of an initial heat transfer analysis during the water shower treatment. Heat loss due to moisture evaporation is only considered when temperature profile of the air surrounding the product can be provided. Otherwise, it should be considered only for products that do not have casing.

The average deviations (i.e. MP values analogous to root mean square error) between the observed and predicted values were 1.2 °C for core temperatures; 1.7 °C for temperatures between core and surface; and 2 °C for surface temperature. Maximum deviations were 1.9 °C for core temperatures; 2.2 °C for temperatures at 5.08 cm from core; and 3.9 °C for surface temperatures. It was concluded that potential temperature prediction errors did not cause noticeable deviations on microbial growth predictions of Clostridium perfringens. The maximum deviation on growth prediction caused by MP values of 1.9 °C was 0.09 log_{10} (CFU/g). Since this deviation on microbial growth is not important from the microbial safety point of view, MP values of 1.9 °C in core temperatures represents a good indicator of model performance. Hence, it can be
concluded that the developed model is in good agreement with the observed temperature values.

In theory, the accuracy of the predictions can be improved when methods for obtaining better estimations of initial temperatures within the product prior to cooling, and temperature distribution of the air surrounding the products are available. Therefore, further investigations should be aimed to developed practical and more accurate methods for determining initial temperature distribution prior entering the chiller in products that have been exposed to water showers; and to determine the thermo-physical properties of the air surrounding the products. The developed heat transfer model can be used for evaluating the effect of cooling deviations. Also, it can be integrated to microbial growth predictive models which can be used as quantitative tools to support food safety management in the meat industry.

6. ACKNOWLEDGEMENTS

A contribution of the University of Nebraska Agricultural Research Division, supported in part by funds provided through the Hatch Act, USDA. Additional support was provided by the USDA-IREE-National Integrated Food Safety Initiative Competitive Grants Program (grant number: 2004-00739). Mention of a trade name, proprietary products, or company name is for presentation clarity and does not imply endorsement by the authors or the University of Nebraska. Also, the authors recognize the valuable support offered by Roberto Jiménez and Andrés Doblado-Maldonado during the validation of this research work.
7. REFERENCES


This appendix contains examples of the outputs obtained from simulations performed to validate the heat transfer model. A total of 14 examples are provided; including 1-3 simulations per each validation test. Examples shown in this appendix were selected to illustrate the effect of evaporation and initial evaporation on the model prediction. The next table shows details of the simulation examples contained in this appendix:

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Water Shower</th>
<th>Type of Casing</th>
<th>Evaporation</th>
<th>No Evaporation</th>
<th>Initial Evaporation (Only)</th>
<th>Figure</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>No</td>
<td>No Casing</td>
<td>X</td>
<td>X</td>
<td></td>
<td>C1, C2</td>
</tr>
<tr>
<td>2</td>
<td>Yes</td>
<td>Plastic</td>
<td>X</td>
<td></td>
<td>X</td>
<td>C3, C4</td>
</tr>
<tr>
<td>3</td>
<td>Yes</td>
<td>Plastic</td>
<td>X</td>
<td></td>
<td>X</td>
<td>C5, C6</td>
</tr>
<tr>
<td>4</td>
<td>Yes</td>
<td>Plastic</td>
<td></td>
<td></td>
<td>X</td>
<td>C7</td>
</tr>
<tr>
<td>5</td>
<td>Yes</td>
<td>Plastic</td>
<td></td>
<td></td>
<td>X</td>
<td>C8</td>
</tr>
<tr>
<td>6</td>
<td>Yes</td>
<td>Fabric</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>C9, C10, C11</td>
</tr>
<tr>
<td>7</td>
<td>Yes</td>
<td>Fabric</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>C12, C13, C14</td>
</tr>
</tbody>
</table>

The following figures show a comparison between predicted temperatures, and observed temperatures at different locations within the tested products.
Figure A1 Comparison between predicted and observed temperatures at different locations for Test#1 (EVAPORATION). Since this product did not have casing and it was not exposed to water showers, the heat loss due to evaporation during chilling cannot be neglected. Observed values between 1<Time<1.4 were accidentally lost; thus, they are not reported in this graph. During the simulation, air temperature during that period of time was assumed as the average chiller temperature.
Figure A2 Comparison between predicted and observed temperatures at different locations for Test#1 (NO EVAPORATION). Since this product did not have casing and it was not exposed to water showers, not considering the effects of evaporation during chilling would result in under prediction of the cooling rates. Thus, evaporation cannot be neglected. Observed values between 1<Time<1.4 were accidentally lost; thus, they are not reported in this graph. The air temperature during that period of time was assumed as the average chiller temperature.
Figure A3 Comparison between predicted and observed temperatures at different locations for Test#2 (EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. Hence, considering evaporation during chilling would result in over-prediction of the cooling rates.
Figure A4 Comparison between predicted and observed temperatures at different locations for Test#2 (ONLY INITIAL EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. However, this product was exposed to a 20 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower time.
Figure A5 Comparison between predicted and observed temperatures at different locations for Test#3 (EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. Hence, considering evaporation during chilling would result in over-prediction of the cooling rates.
Figure A6 Comparison between predicted and observed temperatures at different locations for Test#3 (ONLY INITIAL EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. However, this product was exposed to a 20 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower time.
Figure A7 Comparison between predicted and observed temperatures at different locations for Test#4 (ONLY INITIAL EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. However, this product was exposed to a 15 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower time.
Figure A8 Comparison between predicted and observed temperatures at different locations for Test#5 (ONLY INITIAL EVAPORATION). Since this product had a plastic casing, the heat loss due to evaporation is negligible. However, this product was exposed to a 15 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower time.
Figure A9 Comparison between predicted and observed temperatures at different locations for Test#6 (EVAPORATION). Since this product had a fabric casing, the heat loss due to evaporation is negligible. Hence, considering evaporation during chilling would result in over-prediction of the cooling rates.
Figure A10 Comparison between predicted and observed temperatures at different locations for Test#6 (NO EVAPORATION). Since this product had a fabric casing and it was exposed to 20min water shower, not considering the effects of evaporation during chilling would result in under prediction of the cooling rates.
Figure A11 Comparison between predicted and observed temperatures at different locations for Test#6 (ONLY INITIAL EVAPORATION). Since this product had a fabric casing, the heat loss due to evaporation is highly reduced. However, this product was exposed to a 20 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower period of time.
Figure A12 Comparison between predicted and observed temperatures at different locations for Test#7 (EVAPORATION). Since this product had a fabric casing, the heat loss due to evaporation is negligible. Hence, considering evaporation during chilling would result in over-prediction of the cooling rates.
Figure A13 Comparison between predicted and observed temperatures at different locations for Test#7 (NO EVAPORATION). Since this product had a fabric casing and it was exposed to 20min water shower, not considering the effects of evaporation during chilling would result in under prediction of the cooling rates.
Figure A14 Comparison between predicted and observed temperatures at different locations for Test#7 (ONLY INITIAL EVAPORATION). Since this product had a fabric casing, the heat loss due to evaporation is highly reduced. However, this product was exposed to a 20 min water shower. Thus, this simulation did not consider evaporation during chilling, but it did consider an initial heat loss due evaporation from the product surface during the water shower period of time.
This appendix contains the source code of the Java™ application used to simulate the cooling process of the products tested during the model validation. This source code represents the implementation of a three-dimensional finite element algorithm that can be used to solve the heat transfer model presented in Chapters II and III of this manuscript.

The java application was organized in seven different packages: FEM.evaporation, FEM.element, FEM.material, FEM.meshGenerator, FEM.model, FEM.node and FEM.solver. Description of each package, its respective classes and purpose are presented in the following diagram:

- **FEM.evaporation**
  - QEvaporation.java
    - To impose evaporation boundary condition

- **FEM.element**
  - TetrahedralElement.java
    - To calculate element stiffness matrix, mass matrix and load vector for tetrahedral elements
    - To define element shape functions for tetrahedral elements
    - To calculate element volume
  - TriangularBoundaryElement.java
    - To calculate element stiffness matrix, mass matrix and load vector for triangular elements
    - To define element shape functions of triangular elements
    - To calculate element area
FEM.material

- **Air.java**
  - To calculate thermo physical properties of air

- **FoodMaterial.java**
  - To calculate thermo physical properties of the meat products

- **HeatTransferCoefficients.java**
  - To estimate convective heat transfer coefficient for natural convection
  - To estimate convective heat transfer coefficient for forced convection
  - To estimate radiative heat transfer coefficient

FEM.meshGenerator

- **MeshReader.java**
  - To read the mesh coordinates
  - To upload the connectivity matrix
  - To upload the boundary connectivity matrix
  - To read the location of the probes used during the validation

FEM.model

- **FEMHeatTransfer3D.java**
  - Main class that executes the finite element algorithm

FEM.node

- **Node.java**
  - To define nodal coordinates
  - To define nodal temperature
The documented source code of the main twelve classes is presented below:

```java
package FEM;

import FEM.material.Air;

/**
 * Class for estimating the heat flux due to evaporation at the surface
 * @author Jihan Cepeda
 * @version 12-2010
 *
 * @param q Heat flux due to evaporation (W/m^2)
 * @param wl weight loss (kg)
 */

public class QEvaporation {
```
double q; // Heat flux due to evaporation (W/m^2)
double wl; // weight loss (kg)
double Ka;

public QEvaporation(double Tsavg, double Ta, double RH, double aw,
    double hconv, double a, double b, double c, double surfaceArea) {
    // Tfilm
double Tf = (Tsavg + Ta) / 2.0;

    // Calculate latent heat of vaporization (J/kg)
double hfg = (2502824.56 - 2428.37 * Tf);

    // Calculate absolute humidity of air-vapor mixture (kg water/kg dry air)
double pwa = this.calculatePwa(Ta);
double pa = RH * pwa / 100.0;

    // Calculate vapor pressure of water using Antoine equation (Pws) [at surface temperature]
double pws = this.calculatePws(Tsavg);

    // Calculate Weight loss rate dmw/dt
    // Calculate mass transfer coefficient
this.Ka = this.calculateKa(hconv, hfg);

    // Calculate Weight loss rate dmw/dt
double dmwdt = this.calculateDmDt(pws, pwa, Ka, pa, aw);

    // Calculate heat flux due to evaporation
this.q = hfg * dmwdt;

    // Calculate Weight loss
this.wl = dmwdt * surfaceArea;
}

/**
* Calculate vapor pressure of water using Antoine equation (Pa) [at air temperature]
* @param Ta Air temperature (oC)
* @return pa
*/

public double calculatePwa(double Ta) {
    double Acoeff = 23.4795;
    double Bcoeff = 3990.56;
    double Ccoeff = 233.833;

    return Math.exp(Acoeff - (Bcoeff / (Ta + Ccoeff)));
}

/**
* Calculate vapor pressure of water using Antoine equation (Pws) [at surface temperature]
* @param Tsavg * @return pws
*/
public double calculatePws(double Tsavg) {
    double Acoeff = 23.4795;
    double Bcoeff = 3990.56;
    double Ccoeff = 233.833;
    return Math.exp(Acoeff - (Bcoeff / (Tsavg + Ccoeff)));
}

/**
 * Calculate mass transfer coefficient [Dr. Wang Approach](kg/m^2 s Pa)
 *
 * @param hconv Convective heat transfer coefficient
 * @param hfg Latent heat of vaporization (J/kg)
 * @return Ka
 */
public double calculateKa(double hconv, double hfg) {
    return hconv / (hfg * 64.7);
}

/**
 * Calculate mass transfer coefficient [Chuntranuluck, 1998](kg/m^2 s Pa)
 *
 * @param airP Air properties
 * @param hconv Convective heat transfer coefficient
 * @return Ka
 */
public double calculateKa(Air airP, double hconv) {
    // Atmospheric pressure (Pa)
    double P = 101325;
    return (hconv * 18.02 / (28.97 * airP.getA_cp() * P)); // I used this
    // one from
    // Chuntranuluck
    // 1998
}

/**
 * Calculate mass transfer coefficient [Chuntranuluck, 1998](kg/m^2 s Pa)
 *
 * @param pa Absolute humidity of air-vapor mixture (kg water/kg dry air)
 * @param Tf Tfilm (oC)
 * @param airP Air properties
 * @param hconv Convective heat transfer coefficient
 * @return Ka
 */
public double calculateKa(double pa, double Tf, Air airP, double hconv) {
    // Atmospheric pressure (Pa)
    double P = 101325;
    double H = ((18.02 / 28.97) * (pa / (P - pa)));
    // Calculate humid heat air-vapor mixture (J/kg dry air C)
    double ca = (1000 * (1.005 + 1.88 * H));
// Calculate Mass diffusivity of water vapor in air
double temp = ((1.0 / 18.02) + (1 / 28.97));
double numer = (0.0101325 * Math.pow(TF + 273.15, 1.75) * Math.pow(temp, 0.5));
double denom = P * Math.pow(Math.pow(12.7, 1.0 / 3.0)
+ Math.pow(20.1, 1.0 / 3.0), 2.0);
double Dwa = numer / denom;

// Calculate Schmidt number
double Sc = airP.getA_mu() / (airP.getA_rho() * Dwa);

// Calculate Prandlt number
double Pr = airP.getA_mu() * ca / airP.getA_ctc();
double Le = Sc / Pr;

double Len = Math.pow(Le, (2.0 / 3.0));
return (hconv * 18.02 / (28.97 * airP.getA_cp() * Len * P));

public double calculateDmDt(double pws, double pwa, double Ka, double pa, double aw) {
    if (pws > pwa)
        return Ka * ((aw * pws) - pa);
    else
        return 0;
}
double[][] basicKe;
double[][] basicMe;
double[][] relCoords;
double volume;

FoodMaterial fMaterial;

public TetrahedralElement(Node node0, Node node1, Node node2, Node node3) {
    // Assign nodes
    this.node0 = node0;
    this.node1 = node1;
    this.node2 = node2;
    this.node3 = node3;
    Integrator inte = new Integrator();
    this.relCoords = inte.calculateRelativeCoords(node0, node1, node2, node3);

    // Calculate shape functions
    this.calculateShapeFunctions();

    // Calculate volume
    this.calculateVolume();

    // Calculate BasicKe
    this.calculateBasicKe();

    // Calculate BasicMe
    this.calculateBasicMe();
}

/**
 * Calculate and set the volume of a three-dimensional tetrahedral element based on its area coordinates (nodal coordinates or vertices) Eq. 123 (Zienkiewicz, et all)
 *
 * @return volume Volume of a 3D tetrahedral element
 */
public void calculateVolume() {
    double[][] mat = new double[4][4];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();
    for (int coord = 0; coord < 4; coord++) {
        if (coord == 0) {
            mat[0][coord] = 1;
            mat[1][coord] = 1;
            mat[2][coord] = 1;
            mat[3][coord] = 1;
        } else {
            mat[0][coord] = this.getNode0().getCoords()[coord - 1];
            mat[1][coord] = this.getNode1().getCoords()[coord - 1];
            mat[2][coord] = this.getNode2().getCoords()[coord - 1];
            mat[3][coord] = this.getNode3().getCoords()[coord - 1];
        }
    }
    this.setVolume((linsolv.det(mat)) / 6.0);
}

public double calculateVolumeRel() {
    return volume;
}
double[][] mat = new double[4][4];
LinearAlgebraSolver linsolv = new LinearAlgebraSolver();

for (int coord = 0; coord < 4; coord++) {
    if (coord == 0) {
        mat[0][coord] = 1;
        mat[1][coord] = 1;
        mat[2][coord] = 1;
        mat[3][coord] = 1;
    } else {
        mat[0][coord] = this.relCoords[0][coord - 1];
        mat[1][coord] = this.relCoords[1][coord - 1];
        mat[2][coord] = this.relCoords[2][coord - 1];
        mat[3][coord] = this.relCoords[3][coord - 1];
    }
}

return (Math.abs(linsolv.det(mat)) / 6.0);

public double[][] getRelCoords() {
    return relCoords;
}

public void setRelCoords(double[][] relCoords) {
    this.relCoords = relCoords;
}

/**
 * Calculate matrix representing shape funtions coefficients
 * Lk=ak+bk(x)+ck(y)+dk(z); k=1,2,3,4 Pg. 123 (Zienkiewicz, et all)
 */
public void calculateShapeFunctions() {
    // Calculate parameters of shape functions
    double[] a = this.calculateConstantsA();
    double[] b = this.calculateConstantsB2();
    double[] c = this.calculateConstantsC2();
    double[] d = this.calculateConstantsD2();

    this.shapeFunctions = new double[4][4];

    for (int i = 0; i < 4; i++) {
        if (i == 0 || i == 2) {
            this.shapeFunctions[i][0] = a[i];
        } else {
            this.shapeFunctions[i][0] = -a[i];
        }

        this.shapeFunctions[i][1] = b[i];
        this.shapeFunctions[i][2] = c[i];
        this.shapeFunctions[i][3] = d[i];
    }
}

public double[] calculateConstantsA() {

double[] a = new double[4];

a[0] = calculateConstantA(1, 2, 3);
a[1] = calculateConstantA(2, 3, 0);
a[2] = calculateConstantA(3, 0, 1);
a[3] = calculateConstantA(0, 1, 2);

return a;
}

public double calculateConstantA(int nodeRow0, int nodeRow1, int nodeRow2) {
    double[][] mat = new double[3][3];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();

    mat[0][0] = this.relCoords[nodeRow0][0]; // Xcoord
    mat[0][1] = this.relCoords[nodeRow0][1]; // Ycoord
    mat[0][2] = this.relCoords[nodeRow0][2]; // Zcoord
    mat[1][0] = this.relCoords[nodeRow1][0];
    mat[1][1] = this.relCoords[nodeRow1][1];
    mat[1][2] = this.relCoords[nodeRow1][2];
    mat[2][0] = this.relCoords[nodeRow2][0];
    mat[2][1] = this.relCoords[nodeRow2][1];
    mat[2][2] = this.relCoords[nodeRow2][2];

    return ((linsolv.det(mat)));}

public double[] calculateConstantsB2() {
    double[] b = new double[4];

    b[0] = this.relCoords[1][1] + this.relCoords[3][2] - this.relCoords[2][2]
        - this.relCoords[2][1] + this.relCoords[3][1] - this.relCoords[1][2];
    b[1] = -this.relCoords[0][1] + this.relCoords[3][2] - this.relCoords[2][2]
        + this.relCoords[2][1] - this.relCoords[3][1] + this.relCoords[0][2];
    b[2] = this.relCoords[0][1] - this.relCoords[1][2] + this.relCoords[1][1]
        + this.relCoords[3][2] - this.relCoords[0][2];
    b[3] = -this.relCoords[0][1] + this.relCoords[2][2] - this.relCoords[1][2]
        + this.relCoords[1][1] - this.relCoords[2][2] + this.relCoords[0][2];

    return b;
}

public double calculateConstantB(int nodeRow0, int nodeRow1, int nodeRow2) {
    double[][] mat = new double[3][3];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();
mat[0][0] = 1;
mat[0][1] = this.relCoords[nodeRow0][1]; // Ycoord
mat[0][2] = this.relCoords[nodeRow0][2]; // Zcoord
mat[1][0] = 1;
mat[1][1] = this.relCoords[nodeRow1][1];
mat[1][2] = this.relCoords[nodeRow1][2];
mat[2][0] = 1;
mat[2][1] = this.relCoords[nodeRow2][1];
mat[2][2] = this.relCoords[nodeRow2][2];

return (((linsolv.det(mat))));
}

public double calculateConstantsC2() {
    double[] c = new double[4];
    c[0] = -this.relCoords[1][0]
        * (this.relCoords[3][2] - this.relCoords[2][2])
        + this.relCoords[2][0]
        * (this.relCoords[3][2] - this.relCoords[1][2])
        - this.relCoords[3][0]
        * (this.relCoords[2][2] - this.relCoords[1][2]);
    c[1] = this.relCoords[0][0]
        * (this.relCoords[3][2] - this.relCoords[2][2])
        - this.relCoords[2][0]
        * (this.relCoords[3][2] - this.relCoords[0][2])
        + this.relCoords[3][0]
        * (this.relCoords[2][2] - this.relCoords[0][2]);
    c[2] = -this.relCoords[0][0]
        * (this.relCoords[3][2] - this.relCoords[1][2])
        + this.relCoords[1][0]
        * (this.relCoords[3][2] - this.relCoords[0][2])
        - this.relCoords[3][0]
        * (this.relCoords[1][2] - this.relCoords[0][2]);
    c[3] = this.relCoords[0][0]
        * (this.relCoords[2][2] - this.relCoords[1][2])
        - this.relCoords[1][0]
        * (this.relCoords[2][2] - this.relCoords[0][2])
        + this.relCoords[2][0]
        * (this.relCoords[1][2] - this.relCoords[0][2]);

    return c;
}

public double calculateConstantC(int nodeRow0, int nodeRow1, int nodeRow2) {
    double[][] mat = new double[3][3];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();
    mat[0][0] = this.relCoords[nodeRow0][0]; // Xcoord
    mat[0][1] = 1;
    mat[0][2] = this.relCoords[nodeRow0][2]; // Zcoord
    mat[1][0] = this.relCoords[nodeRow1][0];
    mat[1][1] = 1;
    mat[1][2] = this.relCoords[nodeRow1][2];
    mat[2][0] = this.relCoords[nodeRow2][0];
    mat[2][1] = 1;
    mat[2][2] = this.relCoords[nodeRow2][2];

    return (((linsolv.det(mat))));
}
public double[] calculateConstantsD2() {
    double[] d = new double[4];
    d[0] = this.relCoords[1][0] * (this.relCoords[3][1] - this.relCoords[2][1]) - this.relCoords[2][0] * (this.relCoords[3][1] - this.relCoords[1][1]) + this.relCoords[3][0] * (this.relCoords[2][1] - this.relCoords[1][1]);
    d[1] = -this.relCoords[0][0] * (this.relCoords[3][1] - this.relCoords[2][1]) + this.relCoords[2][0] * (this.relCoords[3][1] - this.relCoords[0][1]) - this.relCoords[3][0] * (this.relCoords[2][1] - this.relCoords[0][1]);
    d[2] = this.relCoords[0][0] * (this.relCoords[3][1] - this.relCoords[1][1]) - this.relCoords[1][0] * (this.relCoords[3][1] - this.relCoords[0][1]) + this.relCoords[2][0] * (this.relCoords[1][1] - this.relCoords[0][1]);
    d[3] = -this.relCoords[0][0] * (this.relCoords[2][1] - this.relCoords[1][1]) + this.relCoords[1][0] * (this.relCoords[2][1] - this.relCoords[0][1]) - this.relCoords[2][0] * (this.relCoords[1][1] - this.relCoords[0][1]);
    return d;
}

public double calculateConstantD(int nodeRow0, int nodeRow1, int nodeRow2) {
    double[][] mat = new double[3][3];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();
    mat[0][0] = this.relCoords[nodeRow0][0]; // Xcoord
    mat[0][1] = this.relCoords[nodeRow0][1]; // Ycoord
    mat[0][2] = 1;
    mat[1][0] = this.relCoords[nodeRow1][0];
    mat[1][1] = this.relCoords[nodeRow1][1];
    mat[1][2] = 1;
    mat[2][0] = this.relCoords[nodeRow2][0];
    mat[2][1] = this.relCoords[nodeRow2][1];
    mat[2][2] = 1;
    return ((linsolv.det(mat)));
}

/**
 * Calculate the mass average temperature at each element. From Haghighi and Segerlind, 1988, Trans ASAE, 31: 629-637
 * @return Tavg Mass average temperature of the element
 */

public double getTavg() {
    double T = this.node0.getT() * this.shapeFunctions[0][0]
        + this.node1.getT() * this.shapeFunctions[1][0]
        + this.node2.getT() * this.shapeFunctions[2][0]
        + this.node3.getT() * this.shapeFunctions[3][0];
    return (T / (6.0 * this.getVolume()));
public double[][] calculateB() {
    double[][] B = new double[3][4];
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 4; j++) {
            B[i][j] = this.shapeFunctions[j][i + 1];
        }
    }
    return B;
}

public void calculateBasicKe() {
    LinearAlgebraSolver las = new LinearAlgebraSolver();
    this.basicKe = new double[4][4];
    double[][] B = this.calculateB();
    double[][] BT = las.transpose(B);
    this.basicKe = las.multiply(BT, B);
}

public double getX(double L0, double L1, double L2, double L3) {
    return this.relCoords[0][0] * L0 + this.relCoords[1][0] * L1
     + this.relCoords[2][0] * L2 + this.relCoords[3][0] * L3;
}

public double getY(double L0, double L1, double L2, double L3) {
    return this.relCoords[0][1] * L0 + this.relCoords[1][1] * L1
     + this.relCoords[2][1] * L2 + this.relCoords[3][1] * L3;
}

public double getZ(double L0, double L1, double L2, double L3) {
    return this.relCoords[0][2] * L0 + this.relCoords[1][2] * L1
     + this.relCoords[2][2] * L2 + this.relCoords[3][2] * L3;
}

public double evaluateShapeFunction(int node, double L0, double L1,
    double L2, double L3) {
    double x = this.getX(L0, L1, L2, L3);
    double y = this.getY(L0, L1, L2, L3);
    double z = this.getZ(L0, L1, L2, L3);

    return (this.shapeFunctions[node][0] + this.shapeFunctions[node][1] * x
        + this.shapeFunctions[node][2] * y +
        this.shapeFunctions[node][3] * z)
     / (6 * this.getVolume());
}
public double[][] getKe() {
    // Calculate term Ke = k(BT*B) * V / (36 * V^2)
    double[] term = this.getCtc() / (36 * this.getVolume());

    double[][] Ke = new double[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            Ke[i][j] = term * this.getBasicKe()[i][j];
        }
    }
    return Ke;
}

public void calculateBasicMe() {
    this.basicMe = new double[4][4];
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            double integral = 0;
            double a = 0.25 * this.evaluateMeIntegral(i, j, 0.585410196624969, 0.138196601125011, 0.138196601125011, 0.138196601125011);
            double b = 0.25 * this.evaluateMeIntegral(i, j, 0.138196601125011, 0.585410196624969, 0.138196601125011, 0.138196601125011);
            double c = 0.25 * this.evaluateMeIntegral(i, j, 0.138196601125011, 0.138196601125011, 0.585410196624969, 0.138196601125011);
            double d = 0.25 * this.evaluateMeIntegral(i, j, 0.138196601125011, 0.138196601125011, 0.138196601125011, 0.585410196624969);
            integral = a + b + c + d;
            double integral2 = this.getVolume() * integral;
            this.basicMe[i][j] = integral2;
        }
    }
}

public double evaluateMeIntegral(int node1, int node2, double L0, double L1, double L2, double L3) {
    double x = this.getX(L0, L1, L2, L3);
    double y = this.getY(L0, L1, L2, L3);
    double z = this.getZ(L0, L1, L2, L3);
    double integral = 0;
    integral = (this.shapeFunctions[node1][0]
this.shapeFunctions[node1][0] * this.shapeFunctions[node2][0] +
this.shapeFunctions[node2][2] * y + this.shapeFunctions[node1][0] * this.shapeFunctions[node2][3] * z +
this.shapeFunctions[node1][1] *
this.shapeFunctions[node2][0] * x + this.shapeFunctions[node1][1] *
this.shapeFunctions[node2][1] * x * x + this.shapeFunctions[node1][2] *
this.shapeFunctions[node2][2] * y * x + this.shapeFunctions[node1][1] *
this.shapeFunctions[node2][3] * z * x + this.shapeFunctions[node1][2] *
this.shapeFunctions[node2][0] * y + this.shapeFunctions[node1][2] * this.shapeFunctions[node2][1] * x * y + this.shapeFunctions[node1][2] *
this.shapeFunctions[node2][2] * y * y + this.shapeFunctions[node1][2] * this.shapeFunctions[node2][3] * z * y + this.shapeFunctions[node1][3] *
this.shapeFunctions[node2][0] * z + this.shapeFunctions[node1][3] * this.shapeFunctions[node2][1] * x * z + this.shapeFunctions[node1][3] *
this.shapeFunctions[node2][2] * y * z + this.shapeFunctions[node1][3] * this.shapeFunctions[node2][3] * z * z) / (36.0 * Math.pow(this.getVolume(), 2));

return integral;

/**
 * Calculate the element mass matrix for 3D heat transfer problem using
 * four-node linear tetrahedral elements
 *
 * @return Me
 */
public double[][] getMe() {
    double[][] Me = new double[4][4];
    // Calculate term Me=integral[(L*LT)]*rho*Cp/(36*V^2)
    double term = this.getRho() * this.getCp();
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 4; j++) {
            Me[i][j] = term * this.getBasicMe()[i][j];
        }
    }
    return Me;
}

public double getCtc() {
    return this.fMaterial.getCtc();
}

public double[][] getBasicKe() {
    return basicKe;
}
public void setBasicKe(double[][] basicKe) {
    this.basicKe = basicKe;
}

public double[][] getBasicMe() {
    return basicMe;
}

public void setBasicMe(double[][] basicMe) {
    this.basicMe = basicMe;
}

public double getRho() {
    return this.fMaterial.getRho();
}

public double getCp() {
    return this.fMaterial.getCp();
}

public Node getNode0() {
    return this.node0;
}

public void setNode0(Node node0) {
    this.node0 = node0;
}

public Node getNode1() {
    return this.node1;
}

public void setNode1(Node node1) {
    this.node1 = node1;
}

public Node getNode2() {
    return this.node2;
}

public void setNode2(Node node2) {
    this.node2 = node2;
}

public Node getNode3() {
    return this.node3;
}

public void setNode3(Node node3) {
    this.node3 = node3;
}

public FoodMaterial getFMaterial() {
    return this.fMaterial;
}

public void setFMaterial(FoodMaterial material) {
    fMaterial = material;
}

public double getVolume() {
    return this.volume;
public void setVolume(double volume) {
    this.volume = volume;
}

public double[][] getShapeFunctions() {
    return shapeFunctions;
}

public void setShapeFunctions(double[][] shapeFunctions) {
    this.shapeFunctions = shapeFunctions;
}

package FEM.element;
import FEM.node.Node;
import FEM.solver.Integrator;
import FEM.solver.LinearAlgebraSolver;

/**
 * 3-Node element for two-dimensional triangular meshes
 * @author Jihan Cepeda
 * @version 12-2010
 */

public class TriangularBoundaryElement {
    public Node node0;
    public Node node1;
    public Node node2;
    public double coords2D[][];
    public double shapeFunctions[][];
    public double basicKbe[][];
    double area;

    public TriangularBoundaryElement(Node node0, Node node1, Node node2) {
        // Assign nodes
        this.node0 = node0;
        this.node1 = node1;
        this.node2 = node2;

        this.get2DCoordinates(this.node0, this.node1, this.node2);

        // Calculate shape functions
        this.calculateShapeFunctions();

        // Calculate area
        this.calculateArea();
    }
}
// Calculate BasicKe
this.calculateBasicKbe();
}

public double evaluateShapeFunction(int node, double L0, double L1, double L2) {
    double x = this.getX(L0, L1, L2);
    double y = this.getY(L0, L1, L2);

    return (this.shapeFunctions[node][0] + this.shapeFunctions[node][1] * x +
            this.shapeFunctions[node][2] * y) / (2 * this.getArea());
}

public void get2DCoordinates(Node node0, Node node1, Node node2) {
    this.coords2D = new double[3][2];
	double[] u1 = this.getVector(node1, node0);
	double[] u2 = this.getVector(node1, node2);

	double magU1 = 0;
	nagU1 = this.getMagnitude(u1);
	double magU2 = this.getMagnitude(u2);

double tetha = this.getTetha(u1, u2);
	nthis.coords2D[0][0] = 0; // X-coord Node0
	nthis.coords2D[0][1] = 0; // Y-coord Node0
	nthis.coords2D[1][0] = magU1; // X-coord Node1
	nthis.coords2D[1][1] = 0; // Y-coord Node1
	nthis.coords2D[2][0] = magU2 * Math.cos(tetha); // X-coord Node2
	nthis.coords2D[2][1] = magU2 * Math.sin(tetha); // Y-coord Node2

    Integrator inte = new Integrator();
	nthis.coords2D = inte.calculateRelativeCoordsTriangle(this.coords2D);
}

/**
 * Calculate the area of a boundary element in a three-dimensional tetrahedral mesh Pg. 117 (Zienkiewicz, et all)
 * @return area Area of boundary element
 */
public void calculateArea() {
    double[][] mat = new double[3][3];
    LinearAlgebraSolver linsolv = new LinearAlgebraSolver();

    for (int node = 0; node < 3; node++) {
        for (int coord = 0; coord < 3; coord++) {
            if (coord == 0)
                mat[node][coord] = 1;
            else
                mat[node][coord] = this.coords2D[node][coord - 1];
        }
    }
}
 this.setArea(Math.abs(linsolv.det(mat)) / 2.0);
}
/**
 * Calculate Shape Functions
 *
*/
public void calculateShapeFunctions() {
    // Calculate parameters of shape functions
    double a1 = this.coords2D[1][0] * this.coords2D[2][1] - this.coords2D[2][0] * this.coords2D[1][1];
    double a2 = this.coords2D[2][0] * this.coords2D[0][1] - this.coords2D[0][0] * this.coords2D[2][1];
    double a3 = this.coords2D[0][0] * this.coords2D[1][1] - this.coords2D[1][0] * this.coords2D[0][1];
    double b1 = this.coords2D[1][1] - this.coords2D[2][1];
    double b2 = this.coords2D[2][1] - this.coords2D[0][1];
    double b3 = this.coords2D[0][1] - this.coords2D[1][1];
    double c1 = this.coords2D[2][0] - this.coords2D[1][0];
    double c2 = this.coords2D[0][0] - this.coords2D[2][0];
    double c3 = this.coords2D[1][0] - this.coords2D[0][0];
    
    this.shapeFunctions = new double[3][3];
    this.shapeFunctions[0][0] = a1;
    this.shapeFunctions[1][0] = a2;
    this.shapeFunctions[2][0] = a3;
    this.shapeFunctions[0][1] = b1;
    this.shapeFunctions[1][1] = b2;
    this.shapeFunctions[2][1] = b3;
    this.shapeFunctions[0][2] = c1;
    this.shapeFunctions[1][2] = c2;
    this.shapeFunctions[2][2] = c3;
}
/**
 * Calculate B matrix of boundary element
 *
*/
public double[][] calculateB() {
    double[][] B = new double[2][3];
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 3; j++) {
            B[i][j] = this.shapeFunctions[j][i + 1];
        }
    }
    return B;
}
/**
 * Calculate the constant part of the element stiffness matrix
 *
*/
public void calculateBasicKbe() {
    this.basicKbe = new double[3][3];
}
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        double integral = 0;
        double a = (1 / 3.0) * this.evaluateKeIntegral(i, j, 0.5, 0.5, 0);
        double b = (1 / 3.0) * this.evaluateKeIntegral(i, j, 0.5, 0, 0.5);
        double c = (1 / 3.0) * this.evaluateKeIntegral(i, j, 0, 0.5, 0.5);
        integral = a + b + c;
        double integral2 = this.area * integral;
        this.basicKbe[i][j] = integral2;
    }
}

public double evaluateKeIntegral(int node1, int node2, double L0,
    double L1, double L2) {
    double x = this.getX(L0, L1, L2);
    double y = this.getY(L0, L1, L2);
    double integral = 0;
    integral = (this.shapeFunctions[node1][0] * this.shapeFunctions[node2][0] +
            this.shapeFunctions[node1][0] * this.shapeFunctions[node2][1] * x +
            this.shapeFunctions[node1][0] * this.shapeFunctions[node2][2] * y +
            this.shapeFunctions[node1][1] * this.shapeFunctions[node2][0] * x +
            this.shapeFunctions[node1][1] * this.shapeFunctions[node2][1] * y +
            this.shapeFunctions[node1][2] * this.shapeFunctions[node2][0] * y +
            this.shapeFunctions[node1][2] * this.shapeFunctions[node2][1] * x * y +
            this.shapeFunctions[node1][2] * this.shapeFunctions[node2][2] * y * y)
            / (4 * Math.pow(this.area, 2));
    return integral;
}

public double getX(double L0, double L1, double L2) {
    return this.coords2D[0][0] * L0 + this.coords2D[1][0] * L1 +
            this.coords2D[2][0] * L2;
}

public double getY(double L0, double L1, double L2) {
    return this.coords2D[0][1] * L0 + this.coords2D[1][1] * L1 +
            this.coords2D[2][1] * L2;
}

/**
 * Calculate the boundary stiffness matrix for the element
 * @param hconv Convective heat transfer coefficient (W/m^2 K)
 * @return Kbe Stiffness matrix of a boundary element
 */
public double[][] getKbe(double hconv) {
    double[][] Kbe = new double[3][3];
    double term = hconv;
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            Kbe[i][j] = term * this.getBasicKbe()[i][j];
        }
    }
    return Kbe;
}

/**
 * Calculate the boundary load matrix for the element
 * @param hconv
 * Convective heat transfer coefficient (W/m^2 K)
 * @param Ta
 * Temperature of heating/cooling medium (°C)
 * @param q
 * heat flux due to evaporation of water (W/m^2)
 * @param bc
 * Type of boundary condition: bc = 1 => convection/radiation
 * boundary condition bc = 2 => heat flux boundary condition
 * @return Kbe Stiffness matrix of a boundary element
 */
public double[] getFbe(double hconv, double Ta, double q, double bc) {
    double[] Fbe = new double[3];
    double term;
    if (bc == 1)
        term = hconv * Ta;
    else
        term = -(q);
    for (int i = 0; i < 3; i++) {
        Fbe[i] = term * this.shapeFunctions[i][0] / 2.0;
    }
    return Fbe;
}

public double[][] getRow(double[][] mat, int row) {
    int cols = mat[0].length;
    double[][] rowMat = new double[1][cols];
    for (int i = 0; i < cols; i++) {
        rowMat[0][i] = mat[row][i];
    }
    return rowMat;
}
public double[] getVector(Node A, Node B) {
    double[] u = new double[3];
    u[0] = B.getCoords()[0] - A.getCoords()[0];
    u[1] = B.getCoords()[1] - A.getCoords()[1];
    return u;
}

public double getMagnitude(double[] u) {
    double sum = 0;
    double mag = 0;
    for (int i = 0; i < u.length; i++) {
        sum = sum + Math.pow(u[i], 2);
    }
    mag = Math.sqrt(sum);
    return mag;
}

public double getScalarProduct(double[] u1, double[] u2) {
    double dotProd = 0;
    for (int i = 0; i < u1.length; i++) {
        dotProd = dotProd + (u1[i] * u2[i]);
    }
    return dotProd;
}

public double getTetha(double[] u1, double[] u2) {
    double tetha;
    tetha = Math.acos(this.getScalarProduct(u1, u2) / (this.getMagnitude(u1) * this.getMagnitude(u2)));
    return tetha;
}
public class Air {
    public double a_rho; // air density (kg/m^3)
    public double a_mu; // air viscosity (Pa s)
    public double a_cp; // air specific heat (J/kg K)
    public double a_ctc; // air thermal conductivity (W/m K)

    public Air(double Tf) {
        this.calArho(Tf);
        this.calAmu(Tf);
        this.calActc(Tf);
        this.calAcp(Tf);
    }

    private void calArho(double Tf) {
        this.a_rho = ((-0.003397 * Tf) + 1.29532);
    }

    private void calAmu(double Tf) {
        this.a_mu = (0.000000045677 * Tf + 0.000017258237);
    }

    private void calActc(double Tf) {
        this.a_ctc = (0.0000715 * Tf + 0.0241794);
    }

    private void calAcp(double Tf) {
        if (Tf >= -30 && Tf < 27)
            this.a_cp = 1004;
        else if (Tf >= 27 && Tf < 50)
            this.a_cp = 1005;
    }
}
else if (Tf >= 50 && Tf < 60)
    this.a_cp = 1006;
else if (Tf >= 60 && Tf < 70)
    this.a_cp = 1007;
else if (Tf >= 70 && Tf < 90)
    this.a_cp = 1008;
else if (Tf >= 90 && Tf < 100)
    this.a_cp = 1009;
else if (Tf == 100)
    this.a_cp = 1010;
}
public FoodMaterial(double Tavg, double prot, double fat, double carb, double salt, double moist) {

    // Calculate density of pure components (Choi and Okos model)
    double rhoP = 1329.9 - (0.5184 * Tavg);
    double rhoF = 925.59 - (0.41757 * Tavg);
    double rhoC = 1599.1 - (0.31046 * Tavg);
    double rhoS = 2423.8 - (0.28063 * Tavg);
    double rhoM = 997.18 + (0.0031439 * Tavg) - (0.0037574 * Math.pow(Tavg, 2));

    // Calculate mass fraction and volume fraction
    double sumw = prot + fat + carb + salt + moist;
    double XwP = prot / sumw;
    double XwF = fat / sumw;
    double XwC = carb / sumw;
    double XwS = salt / sumw;
    double XwM = moist / sumw;

    // Calculate properties of the food material
    this.calculateCtc(Tavg, XvP, XvF, XvC, XvS, XvM);
    this.calculateRho(sumv);
    this.calculateCp(Tavg, XwP, XwF, XwC, XwS, XwM);
}

/**
 * Calculate coefficient of thermal conductivity of food material [Choi and Okos] (W/m K)
 *
 * @param Tavg
 * @param XvP
 * @param XvF
 * @param XvC
 * @param XvS
 * @param XvM
 */
private void calculateCtc(double Tavg, double XvP, double XvF, double XvC, double XvS, double XvM) {

    double ctcP = 0.17881 + (0.0011958 * Tavg) - (0.0000027178 * Math.pow(Tavg, 2));

}
double ctcP = 0.18071 - (0.0027604 * Tavg) - (0.00000017749 * Math.pow(Tavg, 2));
double ctcC = 0.20141 + (0.0013874 * Tavg) - (0.00000004312 * Math.pow(Tavg, 2));
double ctcS = 0.32962 + (0.0014011 * Tavg) - (0.000000029069 * Math.pow(Tavg, 2));
double ctcM = 0.57109 + (0.0017625 * Tavg) - (0.000000067036 * Math.pow(Tavg, 2));

this.ctc = (ctcP * XvP) + (ctcF * XvF) + (ctcC * XvC) + (ctcS * XvS) + (ctcM * XvM);

} /* Calculate density of food material [Choi and Okos] (kg/m^3) */
private void calculateRho(double sumv) {
    this.rho = 1.0 / sumv;
}

} /* Calculate Specific Heat of food material [Choi and Okos] (J/kg K) */
private void calculateCp(double Tavg, double XwP, double XwF, double XwC,
            double XwS, double XwM) {
    double cpP = 2.0082 + (0.0012089 * Tavg) - (0.00000013129 * Math.pow(Tavg, 2));
double cpF = 1.9842 + (0.0014733 * Tavg) - (0.000000048008 * Math.pow(Tavg, 2));
double cpC = 1.5488 + (0.0019625 * Tavg) - (0.000000059399 * Math.pow(Tavg, 2));
double cpS = 1.0926 + (0.0018896 * Tavg) - (0.000000036817 * Math.pow(Tavg, 2));
double cpM = 4.1762 - (0.00000090864 * Tavg) + (0.000000054731 * Math.pow(Tavg, 2));

    this.cp = 1000 * ((cpP * XwP) + (cpF * XwF) + (cpC * XwC) + (cpS * XwS) +
            (cpM * XwM));
}
package FEM.material;

/**
 * Class for calculating heat transfer coefficients due to convection and
 * radiation (Ellipsoidal shaped products) [Yovanovich, 1987]
 * @author Jihan Cepeda
 * @version 04-2010
 * @param h
 * Combined (convect + radiation) heat transfer coefficient (W/m^2 K)
 * @param hconv
 * Convective heat transfer coefficient (W/m^2 K)
 * @param hrad
 * Radiative heat transfer coefficient (W/m^2 K)
 */

public class HeatTransferCoefficients {
    double h; // Combined heat transfer coefficient due to convection and
    // radiation (W/m^2 K)
    double hconv; // Convective heat transfer coefficient (W/m^2 K)
    double hrad; // Heat transfer coefficient due to radiation (W/m^2 K)
    double D;
    double Re;
    double Pr;
    Air airP;
    double hfc;
    double hfree;
    double Gr;
    double Ra;
    double freeNu;
    double forcedNu;

    /**
     * Class constructor
     * @param a
     * A-axis of the ellipsoid [m]
     * @param b
     * B-axis of the ellipsoid [m]
     * @param c
     * C-axis of the ellipsoid [m]
     * @param AirVel
     * Air velocity [m/s]
     * @param Tsavg
     * Average temperature at the surface [oC]
     * @param Ta
     * Air temperature [oC]
     */
    public HeatTransferCoefficients(double a, double b, double c,
            double AirVel, double Tsavg, double Ta, double surfArea) {
        // Calculate film temperature (Tfilm)
        double Tfilm = (Tsavg + Ta) / 2;
        // Get properties of air at Tfilm
```java
this.airP = new Air(Tfilm);

// Calculate characteristic dimension D=sqrt(A) (Yovanovich, 1988)
this.D = Math.sqrt(surfArea);

// Calculate Reynolds number
this.Re = this.calculateRe(D, AirVel, airP);

// Calculate Prandtl number
this.Pr = this.calculatePr(airP);

// Calculate heat transfer coefficient due to forced convection
this.hfc = this.calculateHfc(Re, Pr, D, airP);

// Calculate heat transfer coefficient due to free convection
this.hfree = this.calculateHfree(Tsavg, Ta, airP, D, Pr);

// Calculate hrad
this.hrad = this.calculateHrad(Tsavg, Ta);

// Calculate hconv
this.hconv = Math.pow((Math.pow(hfc, 3.0) + Math.pow(hfree, 3.0)), 1.0 / 3.0);

// Calculate h combined
this.h = this.hconv + this.hrad;
}

/**
 * Calculate characteristic dimension for an ellipsoid
 * @param a A-axis of the ellipsoid [m]
 * @param b B-axis of the ellipsoid [m]
 * @param c C-axis of the ellipsoid [m]
 * @return D Characteristic dimension for ellipsoids (Yovanovich, 1988)
 */
public double calculateD(double a, double b, double c) {
    double avgb = (b + c) / 2.0;

    double e = Math.sqrt(1 - Math.pow(avgb / a, 2));
    double A = 2.0 * Math.PI * Math.pow(avgab, 2.0);

    double A = 2.0 * Math.PI * Math.pow(avgab, 2.0)

    double in = (Math.pow(a, p) * Math.pow(b, p) + Math.pow(a, p)
    * Math.pow(c, p) + Math.pow(b, p) * Math.pow(c, p)) / 3.0;
    double surfaceArea = 4 * Math.PI * Math.pow(in, 1.0 / p);
    double D2 = Math.sqrt(surfaceArea);

    return D2;
}
```
public double calculateRe(double D, double AirVel, Air airP) {
    return D * AirVel * airP.getA_rho() / airP.getA_mu();
}

public double calculatePr(Air airP) {
    return airP.getA_mu() * airP.getA_cp() / airP.getA_ctc();
}

public double calculateHfc(double Re, double Pr, double D, Air airP) {
    double C1 = 0.150;
    double C2 = 0.350;
    double this.forcedNu = 2.0
        * Math.sqrt(Math.PI) + (C1 * Math.pow(Math.PI, 0.25) * Math.pow(Re, 0.5) + C2
            * Math.pow(Re, 0.566)) * Math.pow(Pr, 1.0 / 3.0);
    return airP.getA_ctc() * this.forcedNu / D;
}
public double calculateHfree(double Tsavg, double Ta, Air airP, double D, double Pr) {
    double Gr = (Math.pow(airP.getARho(), 2) * 9.80665 * (1 / (Ta + 273.15)) * Math.pow(D, 3) * Math.abs(Tsavg - Ta) / Math.pow(airP.getAMu(), 2));
    double Ra = Gr * Pr;
    double freeNu = (3.470 + 0.510 * Math.pow(Ra, 1.0 / 4.0));
    return airP.getCtc() * freeNu / D;
}

public double calculateHrad(double Tsavg, double Ta) {
    double boltzman = (5.676 * Math.pow(10.0, -8.0));
    double epsilon = 0.95;
    double TsavgK = Tsavg + 273.15;
    double TaK = Ta + 273.15;
    double TsavgK = Tsavg + 273.15;
    double TaK = Ta + 273.15;
    return epsilon * boltzman * (Math.pow(TsavgK, 2.0) + Math.pow(TaK, 2.0)) * (TsavgK + TaK);
}
package FEM.meshGenerator;

import java.io.File;
import java.io.IOException;
import java.util.Locale;
import java.util.Vector;
import jxl.Sheet;
import jxl.Workbook;
import jxl.WorkbookSettings;
import jxl.read.biff.BiffException;
import FEM.model.Probe;

public class MeshReader {
    // Vector of XYZ coordinates
    Vector<Vector<Double>> X;
    // Vector containing numbers of boundary nodes
    Vector<Integer> boundNodes;
    // Vector containing domain connectivity matrix
    Vector<Vector<Integer>> IJK;
    // Vector containing boundary connectivity matrix
    Vector<Vector<Integer>> IJ;
    // Vector of XYZ coordinates
    Vector<Probe> Probes;

    // Number of elements
    int ne;
    // Number of nodes per element
    int nnpe;

    // Number of coordinates
    int nc;
    // Number of nodes
    int nn;

    // Number of boundary elements
    int nbe;
    // Number of boundary nodes
    int nbn;
    // Number of nodes per boundary element
    int nnpbe;

    public MeshReader(String filename) {
        // Initialize vectors
        this.X = new Vector<Vector<Double>>(1, 1);
        this.Probes = new Vector<Probe>(1, 1);
        this.IJK = new Vector<Vector<Integer>>(1, 1);
        this.IJ = new Vector<Vector<Integer>>(1, 1);
        this.boundNodes = new Vector<Integer>(1, 1);

        try {
            // Open MS Excel File
            WorkbookSettings ws = new WorkbookSettings();
        } catch (IOException | BiffException e) {
            e.printStackTrace();
        }
    }
}
ws.setLocale(new Locale("en", "EN"));

Workbook workbook = Workbook.getWorkbook(new File(filename), ws);

// Open Sheet with nodal coordinates
Sheet n = workbook.getSheet("Nodes");

this.nn = n.getRows();
this.nc = n.getColumns() - 1;
Vector<Double> Xtemp;

// Read XYZ coordinates
for (int node = 0; node < this.nn; node++) {
    Xtemp = new Vector<Double>(1, 1);
    for (int coord = 0; coord < this.nc + 1; coord++) {
        Xtemp.add(Double.parseDouble("" + n.getCell(coord, node).getContents().toString()));
    }
    X.add(Xtemp);
}

// Read Boundary Nodes
Sheet bn = workbook.getSheet("BoundaryNodes");
this.nbn = bn.getRows();
for (int bnode = 0; bnode < this.nbn; bnode++) {
    this.boundNodes.add(Integer.parseInt(bn.getCell(0, bnode).getContents()));
}

// Read domain connectivity matrix
Sheet e = workbook.getSheet("Elements");
this.ne = e.getRows();
this.nnpe = e.getColumns();
Vector<Integer> IJKtemp;
for (int ele = 0; ele < this.ne; ele++) {
    IJKtemp = new Vector<Integer>(1, 1);
    for (int localNode = 0; localNode < this.nnpe; localNode++) {
        IJKtemp.add(Integer.parseInt(e.getCell(localNode, ele).getContents()));
    }
    IJK.add(IJKtemp);
}

// Read boundary connectivity matrix
Sheet be = workbook.getSheet("BoundaryElements");
this.nbe = be.getRows();
this.nnpbe = be.getColumns();
Vector<Integer> IJtemp;
for (int bEle = 0; bEle < this.nbe; bEle++) {
    IJtemp = new Vector<Integer>(1, 1);
    for (int localNode = 0; localNode < this.nnpbe; localNode++) {
        IJtemp.add(Integer.parseInt(be.getCell(localNode, bEle).getContents()));
    }
    IJK.add(IJtemp);
}
IJ.add(IJtemp);
}

// Open Sheet with location of probes
Sheet probes = workbook.getSheet("Probes");

Probe probe;

// Read XYZ coordinates
for (int row = 0; row < probes.getRows(); row++) {
    int id = probes.getCell(0, row).getContents().toString();
    double x = Double.parseDouble(probes.getCell(1, row).getContents().toString());
    double y = Double.parseDouble(probes.getCell(2, row).getContents().toString());
    double z = Double.parseDouble(probes.getCell(3, row).getContents().toString());
    int targetNode = this.getProbesLocation(x, y, z);
    probe = new Probe(id, x, y, z, targetNode);
    this.Probes.add(probe);
}

this.testProbesLocation();
workbook.close();
}

public int getProbesLocation(double x, double y, double z) {
    int closerNode = 0;
    double closerDistance = 10000;
    for (int node = 0; node < this.nn; node++) {
        double distance = Math.abs(X.get(node).get(0) - x) + Math.abs(X.get(node).get(1) - y) + Math.abs(X.get(node).get(2) - z);
        if (distance < closerDistance) {
            closerNode = node;
            closerDistance = distance;
        }
    }
    return closerNode;
}

public void testProbesLocation() {
    System.out.println("Testing Location of Probes");
    for (int probeNumber = 0; probeNumber < this.Probes.size(); probeNumber++) {
double coordProbe;
double coordNode;
double difference;

for (int coord = 0; coord < 3; coord++) {
    coordProbe =
    this.Probes.get(probeNumber).getCoords()[coord];
    coordNode =
    this.X.get(
        this.Probes.get(probeNumber).getTargetNode())
            .get(coord);
    difference = Math.abs(coordProbe - coordNode);
    System.out.println(coordProbe + " " + coordNode + " "
                        + difference);
}
Number of dimensions that will be considered (i.e. Nodes in two-dimensional problems will have dim=2)

```java
public Node(int dim) {
    this.coords = new double[dim];
}
```

Get node coordinates

```java
public double[] getCoords() {
    return coords;
}
```

Set nodal coordinates for two-dimensional problems

```java
public void setCoord(double coord, int i) {
    this.coords[i] = coord;
}
```

Set nodal coordinates for two-dimensional problems

```java
public void setCoords(double x, double y) {
    this.coords[0] = x;
    this.coords[1] = y;
}
```

Set nodal coordinates for three-dimensional problems

```java
public void setCoords(double x, double y, double z) {
    this.coords[0] = x;
    this.coords[1] = y;
    this.coords[2] = z;
}
```

Get nodal temperature [oC]

```java
public double getT() {
    return T;
}
```
public double getT() {
    return this.T;
}

/**
 * Set nodal temperature [°C]
 * @param t New nodal temperature
 */
public void setT(double t) {
    this.T = t;
}

/**
 * Get node number
 * @return nodeNumber Node number
 */
public int getTreeNodeNumber() {
    return this.nodeNumber;
}

/**
 * Set node number
 * @param nodeNumber New node number
 */
public void setTreeNodeNumber(int nodeNumber) {
    this.nodeNumber = nodeNumber;
}

package FEM.solver;
import FEM.node.Node;
public class Integrator {
    public Integrator() {
    }

    public double integrateMeTetrahedron(double[][] relCoords,
                                          double[][] LTixLj,
                                          double vol) {
        double result = 0;
        double[][] integralMe = new double[4][4];

        integralMe[0][0] = 1 * vol;
        integralMe[0][1] = 0;
        integralMe[0][2] = 0;
        integralMe[0][3] = 0;
    }
integralMe[1][0] = 0;
integralMe[1][1] = (vol / 20.0)
  * (Math.pow(relCoords[0][0], 2) + Math.pow(relCoords[0][1], 2));
integralMe[1][2] = (vol / 20.0)
integralMe[1][3] = (vol / 20.0)

integralMe[2][0] = 0;
integralMe[2][1] = integralMe[1][2];
integralMe[2][2] = (vol / 20.0)
  * (Math.pow(relCoords[0][1], 2) + Math.pow(relCoords[0][2], 2));
integralMe[2][3] = (vol / 20.0)
integralMe[3][0] = 0;
integralMe[3][1] = integralMe[1][3];
integralMe[3][2] = integralMe[2][3];
integralMe[3][3] = (vol / 20.0)
  * (Math.pow(relCoords[0][2], 2) + Math.pow(relCoords[0][3], 2));

for (int i = 0; i < 4; i++) {
  for (int j = 0; j < 4; j++) {
    result = result + (integralMe[i][j] * LTixLj[i][j]);
  }
}
return result;

public double[] calculateCentroidTetrahedron(Node node0, Node node1, Node node2, Node node3) {
  double[] centroid = new double[3];
  for (int i = 0; i < 3; i++) {
    centroid[i] = (node0.getCoords()[i] + node1.getCoords()[i] + node2.getCoords()[i] + node3.getCoords()[i]) / 4.0;
  }
  return centroid;
}

public double[][] calculateRelativeCoords(Node node0, Node node1, Node node2, Node node3) {
  double[][] relCoords = new double[4][3];
  double[] centroid = this.calculateCentroidTetrahedron(node0, node1, node2, node3);
}
```java
for (int i = 0; i < 3; i++) {
    relCoords[0][i] = node0.getCoords()[i] - centroid[i];
    relCoords[1][i] = node1.getCoords()[i] - centroid[i];
    relCoords[2][i] = node2.getCoords()[i] - centroid[i];
    relCoords[3][i] = node3.getCoords()[i] - centroid[i];
}

return relCoords;
}

public double[] calculateCentroidTriangle(double[][] Coords2D) {
    double[] centroid = new double[2];
    for (int i = 0; i < 2; i++) {
        double sum = 0;
        for (int j = 0; j < 3; j++) {
            sum = sum + Coords2D[j][i];
        }
        centroid[i] = sum / 3;
    }
    return centroid;
}

public double[][] calculateRelativeCoordsTriangle(double[][] Coords2D) {
    double[][] relCoords = new double[3][2]; // (i.e. relCoords[1][1]=[1 rX1 // rY1 rZ1])
    double[] centroid = this.calculateCentroidTriangle(Coords2D);
    for (int i = 0; i < 2; i++) {
        relCoords[0][i] = Coords2D[0][i] - centroid[i];
        relCoords[1][i] = Coords2D[1][i] - centroid[i];
        relCoords[2][i] = Coords2D[2][i] - centroid[i];
    }
    return relCoords;
}
```
package FEM.solver;

import javax.naming.NoPermissionException;

/**
 * Class for solving linear equations systems
 * @author Jihan Cepeda
 * @version 04-2010
 */

public class LinearEquationsSolver {
    private double mCoef[][];
    private int mNumEcs;

    /**
     * Solve linear system
     * @return Vector of solutions for the linear equations system
     * @throws NoPermissionException
     */
    public double[] getSolucion() throws NoPermissionException {
        double x;
        double y;
        int j;
        int i;
        int k;

        for (j = 0; j < mNumEcs; j++) {
            // Finding the fist equation with a non-zero coefficient in the
            // column(equation) that we are looking at(j)
            for (i = j; i < mNumEcs; i++) {
                if (mCoef[i][j] != 0D) {
                    break;
                }
                throw new NoPermissionException(
                    "SistemaEcuaciones.SolucionNoUnica");
            }

            // (+) Moving that equation to the first row
            for (k = 0; k < mNumEcs + 1; k++) {
                x = mCoef[j][k];
                mCoef[j][k] = mCoef[i][k];
                mCoef[i][k] = x;
            }

            // (+) Obtaining a unit coefficient in the first non-zero column
            y = 1 / mCoef[j][j];
            for (k = 0; k < mNumEcs + 1; k++) {
                mCoef[j][k] = y * mCoef[j][k];
            }

            for (i = 0; i < mNumEcs; i++) {
                y = -mCoef[i][j];
                for (k = 0; k < mNumEcs + 1; k++) {
                    // Further code...
                }
            }
        }
    }
}
if (i == j)
    break;
  mCoef[i][k] = mCoef[i][k] + y * mCoef[j][k];
}
}
}

double dRet[] = new double[mNumEcs];
for (i = 0; i < mNumEcs; i++) {
    double dRes1 = mCoef[i][mNumEcs] * 100000 + 0.5;
    int iRes = (int) dRes1;
    double dRes2 = iRes / 100000D;
    dRet[i] = dRes2;
}
for (i = 0; i < mNumEcs; i++) {
    // System.out.println("x("+i+")= "+dRet[i]);
}
return dRet;

package FEM.model;
import java.io.File;
import java.io.IOException;
import java.math.BigDecimal;
import java.util.Calendar;
import java.util.Locale;
import java.util.Vector;
import javax.naming.NoPermissionException;
import javax.servlet.http.HttpSession;
import jxl.Sheet;
import jxl.Workbook;
import jxl.WorkbookSettings;
import jxl.read.biff.BiffException;
import FEM.QEvaporation;
import FEM.element.TetrahedralElement;
import FEM.element.TriangularBoundaryElement;
import FEM.material.FoodMaterial;
import FEM.material.HeatTransferCoefficients;
import FEM.meshGenerator.MeshReader;
import FEM.node.Node;
import FEM.solver.LinearAlgebraSolver;
import FEM.solver.LinearEquationsSolver;

/**
 * Class for solving 3D heat conduction problems on irregular shaped geometries
 * with convection and evaporation boundary conditions. This class uses
 * four-node linear tetrahedral elements.
 * @author Jihan Cepeda
 * @version 12-2010
 */
public class FEAHeatTransfer3D {

    // GEOMETRY VARIABLES
    // Mesh reader, reads a mesh from MS Excel file
    MeshReader mesh;
    // Array of nodes
    Node[] nodes;
    // Array containing nodal numbers of boundary nodes
    int[] boundaryNodes;

    // FEM VARIABLES
    // Array of 3D elements
    TetrahedralElement[] elements;
    // Array of boundary elements
    TriangularBoundaryElement[] boundaryElements;

    // SOLVER
    // Object to solve linear systems of equations
    LinearAlgebraSolver las = new LinearAlgebraSolver();

    // OUTPUT VARIABLES
    // Elapsed time per iteration
    Vector<Double> time = new Vector<Double>(1, 1);
    // Vector containing nodal temperatures per iteration, T[n][nn]
    Vector<Vector<Double>> T = new Vector<Vector<Double>>(1, 1);
    // Vector of surface average temperature at a particular iteration
    Vector<Double> Tsavg = new Vector<Double>(1, 1);
    // Vector of air temperature profile
    Vector<Double> airTempProfile = new Vector<Double>(1, 1);
    // Vector of weight per iteration
    Vector<Double> weight = new Vector<Double>(1, 1);

    // INPUT VARIABLES
    // Arrays containing air temperature profile
    double[] airTimeUser;
    double[] airTempUser;

    String testingOption = "Average";

    public FEAHeatTransfer3D(String test, String meshFile, String inputOption, String scenario, String userAirProf, String units, HttpSession arg0, int dt2, Boolean evap, int waterShowerTime, int probesTime) throws BiffException, IOException {
        dataSet ds = this.readInput(test, dt2);

        // Set Up Air temperature Profile if it was provided
        if (userAirProf != null && scenario != null) {
            this setUpAirProfile(userAirProf, units);
        }

        String testingOption = inputOption;

        // Set up water activity
        double aw = 0.99;

        // MESH GENERATION
        System.out.println("Reading Mesh...");
        // Import mesh data from Excel file
        this.mesh = new MeshReader(meshFile);
    }
}
// Create Array of Nodes
System.out.println("Creating nodes...");
this.nodes = new Node[this.mesh.getNn()];
for (int i = 0; i < mesh.getNn(); i++) {
    // Construct a node
    nodes[i] = new Node(this.mesh.getNc());
    // Assign node number
    nodes[i].setNodeNumber(i);
    // Assign coordinates
    for (int j = 0; j < this.mesh.getNc(); j++) {
        nodes[i].setCoord(this.mesh.getX().get(i).get(j), j);
    }
    // Initialize temperature
    nodes[i].setT(ds.getTo());
}

// Create array containing boundary node numbers
this.boundaryNodes = new int[mesh.getNbn()];
for (int i = 0; i < mesh.getNbn(); i++) {
    // Get numbers of boundary nodes
    this.boundaryNodes[i] = mesh.getBoundNodes().get(i);
}

// Create array of elements
this.elements = new TetrahedralElement[this.mesh.getNe()];
double vol = 0.0;
for (int i = 0; i < mesh.getNe(); i++) {
    // Construct an element
    elements[i] = new TetrahedralElement
        (nodes[this.mesh.getIJK().get(i).get(0)],
         nodes[this.mesh.getIJK().get(i).get(2)],
         nodes[this.mesh.getIJK().get(i).get(1)],
         nodes[this.mesh.getIJK().get(i).get(3)]);
    vol = vol + elements[i].getVolume();
}

// Create array of boundary elements
double bArea = 0;
boundaryElements = new TriangularBoundaryElement[this.mesh.getNbe()];
for (int i = 0; i < mesh.getNbe(); i++) {
    // Construct a boundary element
    int node0 = this.mesh.getIJ().get(i).get(0);
    int node1 = this.mesh.getIJ().get(i).get(1);
    int node2 = this.mesh.getIJ().get(i).get(2);
    boundaryElements[i] = new TriangularBoundaryElement
        (this.nodes[node0], this.nodes[node1],
         this.nodes[node2]);
    double[][] coords = new double[3][3];
    coords[0][0] = this.nodes[node0].getCoords()[0];
    coords[0][1] = this.nodes[node0].getCoords()[1];
    coords[0][2] = this.nodes[node0].getCoords()[2];
    coords[1][0] = this.nodes[node1].getCoords()[0];
    coords[1][1] = this.nodes[node1].getCoords()[1];
    coords[1][2] = this.nodes[node1].getCoords()[2];
    coords[2][0] = this.nodes[node2].getCoords()[0];
    coords[2][1] = this.nodes[node2].getCoords()[1];
coords[2][2] = this.nodes[node2].getCoords()[2];

bArea = bArea + boundaryElements[i].getArea();
}

// Vectors to store temperature over time
T = new Vector<Vector<Double>>(1, 1);
Tsavg = new Vector<Double>(1, 1);
time = new Vector<Double>(1, 1);

// Initialize vectors of temperature
Vector<Double> initialTaux = new Vector<Double>(1, 1);
for (int i = 0; i < mesh.getNn(); i++) {
    initialTaux.add(nodes[i].getT());
}
T.add(initialTaux);
Tsavg.add(ds.getTosurf());
double Tcore = ds.To;

int centralNode = this.getCentralNode(ds.getCentralProbe());

// Initialize time vector
time.add(0.0);

// Initialize weight loss vector
this.weight = new Vector<Double>(1, 1);
this.weight.add(ds.getWo());

// Initialize global matrices (n-1)
double[] Fold = new double[mesh.getNn()];
double[][] Kold = new double[mesh.getNn()][mesh.getNn()];
for (int i = 0; i < mesh.getNn(); i++) {
    Fold[i] = 0;
    for (int j = 0; j < mesh.getNn(); j++) {
        Kold[i][j] = 0;
    }
}
int n = 0;
int lecture = 0;

double[] F = new double[mesh.getNn()];
// Initialize Global Stiffness Matrix (K)
double[][] K = new double[mesh.getNn()][mesh.getNn()];
// Initialize Global Mass Matrix (M)
double[][] M = new double[mesh.getNn()][mesh.getNn()];
for (int i = 0; i < mesh.getNn(); i++) {
    F[i] = 0;
    for (int j = 0; j < mesh.getNn(); j++) {
        K[i][j] = 0;
        M[i][j] = 0;
    }
}

// PRINT SIMULATION RESULTS
System.out.print("Time ");
for (int i = 0; i < this.mesh.getProbes().size(); i++) {
    System.out.print(this.mesh.getProbes().get(i).getId() + " ");
}
System.out.println("weight q wl ka k p Cp pAir uAir CpAir kAir RH Ta AirVel TsAvg h hconv hrad Re Pr hfc hfree Gr Ra freeNu forcedNu processingTime");
System.out.println();

int delay = waterShowerTime + probesTime;

while (Tcore >= ds.getTarget() \\
&& (time.get(n) - delay) < ds.getTestTime()[ds.getTestTime().length - 1]) {
    Calendar clock1 = Calendar.getInstance();
    n++;
    if (arg0 != null) {
        arg0.setAttribute("iteration", n);
        double simProgress = 100 * (ds.getTo() - Tcore) / (ds.getTo() - ds.getTarget());
        arg0.setAttribute("state", this.Round(simProgress, 1));
    }

    // Calculate element matrices and assemble them into the global system
    FoodMaterial fMaterial = new FoodMaterial();
    double avgCtc = 0;
    double avgRho = 0;
    double avgCp = 0;
    for (int i = 0; i < mesh.getNe(); i++) {
        // Initialize food material properties based on product composition
        fMaterial = new FoodMaterial(elements[i].getTavg(), ds.getProt(), ds.getFat(), ds.getCarb(), ds.getAsh(), ds)
        .getMoist());
        avgCtc = avgCtc + fMaterial.getCtc();
        avgRho = avgRho + fMaterial.getRho();
        avgCp = avgCp + fMaterial.getCp();

        // Set up material properties of element [i]
        elements[i].setFMaterial(fMaterial);

        // Calculate element Mass Matrix
        double[][] Me = new double[mesh.getNnpe()][mesh.getNnpe()];
        Me = elements[i].getMe();

        // Calculate element Stiffness Matrix
        double[][] Ke = new double[mesh.getNnpe()][mesh.getNnpe()];
        Ke = elements[i].getKe();

        // Assembling element matrices into the global system
        int[] nodeNumbers = new int[mesh.getNnpe()];
        nodeNumbers[0] = elements[i].node0.getNodeNumber();
        nodeNumbers[1] = elements[i].node1.getNodeNumber();
        nodeNumbers[2] = elements[i].node2.getNodeNumber();
        nodeNumbers[3] = elements[i].node3.getNodeNumber();
        for (int j = 0; j < mesh.getNnpe(); j++) {
int a = nodeNumbers[j];
for (int k = 0; k < mesh.getNnpe(); k++) {
    int b = nodeNumbers[k];
    M[a][b] = M[a][b] + Me[j][k];
    K[a][b] = K[a][b] + Ke[j][k];
}
}

avgCtc = avgCtc / this.mesh.getNe();
avgRho = avgRho / this.mesh.getNe();
avgCp = avgCp / this.mesh.getNe();

if (ds.getAirVel() <= 0.0) {
    ds.setAirVel(0.001);
}

// Define Air profile
// Air profile for "Best Case Scenario" (Default)
double Ta2;
Ta2 = ds.getTa();
double AirVel2;
AirVel2 = ds.getAirVel();
double RH2;
RH2 = ds.getRH();

// Modify air profile that will be sent as output
this.airTempProfile.add(Ta2);

if (testingOption.equalsIgnoreCase("Real")) {
    if (time.get(n - 1) < waterShowerTime) {
        Ta2 = 30;
        AirVel2 = 0.001;
        RH2 = 100;
    } else if (time.get(n - 1) < delay) {
        Ta2 = ds.getTestAirTemperature()[0];
        AirVel2 = ds.getTestAirVelocity()[0];
        RH2 = ds.getTestRelativeHumidity()[0];
    } else if (time.get(n - 1) <
        ds.getTestTime()[ds.getTestTime().length - 1]) {
        for (int i = lecture; i < ds.getTestTime().length;
            i++) {
            if (ds.getTestTime()[i] >= (time.get(n - 1) -
                delay)) {
                Ta2 = ds.getTestAirTemperature()[i];
                AirVel2 = ds.getTestAirVelocity()[i];
                RH2 = ds.getTestRelativeHumidity()[i];
                lecture = i;
                i = ds.getTestTime().length;
            }
        }
    }
    if (AirVel2 < 0) {
        AirVel2 = ds.getAirVel();
    }
    if (RH2 < 0) {
        RH2 = ds.getRH();
    }
// Calculate heat transfer coefficients
HeatTransferCoefficients htc = new HeatTransferCoefficients(ds
.getA(), ds.getB(), ds.getC(), AirVel2, Tsavg.get(n -
1),
Ta2, bArea);

// Calculate heat loss due to evaporation on the surface
QEvaporation hfe = new QEvaporation(Tsavg.get(n - 1), Ta2, RH2,
aw,
htc.getHc(), ds.getA(), ds.getB(), ds.getC(), bArea);

// Calculate weight loss due to evaporation on the surface
double weightLoss = this.weight.get(n - 1) - hfe.getWl()
* ds.getDt();
this.weight.add(weightLoss);

// Impose boundary conditions
for (int i = 0; i < mesh.getNbe(); i++) {

    // Calculate stiffness matrix of boundary element
double[][] Kbe = new
double[mesh.getNnpe()][mesh.getNnpe()];
Kbe = this.boundaryElements[i].getKbe(htc.getH());

    // Calculate load vector of boundary element (Heat
convection)
    double[] FbeConv = new double[mesh.getNnpe()];
FbeConv = this.boundaryElements[i].getFbe(htc.getH(), Ta2,
hfe
    .getQ(), 1);

    // Calculate load vector of boundary element (Evaporation
    // boundary condition)
double[] FbeEvap = new double[mesh.getNnpe()];
FbeEvap = this.boundaryElements[i].getFbe(htc.getH(), Ta2,
hfe
    .getQ(), 2);

    // Assemble boundary element matrices into the global
system
    int[] nodeNumbers = new int[3];
nodeNumbers[0] =
this.boundaryElements[i].node0.getNodeNumber();
nodeNumbers[1] =
this.boundaryElements[i].node1.getNodeNumber();
nodeNumbers[2] =
this.boundaryElements[i].node2.getNodeNumber();

    for (int j = 0; j < mesh.getNpbe(); j++) {
        int a = nodeNumbers[j];

        if (evap || time.get(n - 1) <= waterShowerTime) {
            F[a] = F[a] + FbeConv[j] + FbeEvap[j];
        } else {
            F[a] = F[a] + FbeConv[j]; // + FbeEvap[j];
        }
    }
}
for (int k = 0; k < mesh.getNnpbe(); k++) {
    int b = nodeNumbers[k];
    K[a][b] = K[a][b] + Kbe[j][k];
}

// Initialize transient global matrices
double[][] TK = new double[mesh.getNn()][mesh.getNn()];
double[][] TKright = new double[mesh.getNn()][mesh.getNn()];
double TF[] = new double[mesh.getNn()];
double theta = 0.5;

// Calculate transient global matrices
for (int i = 0; i < mesh.getNn(); i++) {
    TF[i] = 0.0;
    for (int j = 0; j < mesh.getNn(); j++) {
        double div = 1.0 / ds.getDt();
        TK[i][j] = div * M[i][j] + theta * K[i][j];
        TKright[i][j] = div * M[i][j] - (1.0 - theta) * K[i][j];
    }
}

for (int i = 0; i < mesh.getNn(); i++) {
    for (int j = 0; j < mesh.getNn(); j++) {
        TF[i] = TF[i] + TKright[i][j] * this.nodes[j].getT();
    }
    TF[i] = TF[i] + theta * F[i] + (1 - theta) * F[i];
}

// Reorganize TK and TF into a single matrix to solve linear equations system
double[][] TSKprov = new double[mesh.getNn()][mesh.getNn()];
for (int i = 0; i < mesh.getNn(); i++) {
    for (int j = 0; j < mesh.getNn(); j++) {
        TSKprov[i][j] = TK[i][j];
    }
    TSKprov[i][mesh.getNn()] = TF[i];
}

// Calculate transient response (Solve transient global system)
LinearEquationsSolver les = new LinearEquationsSolver();
les.setNumEcs(mesh.getNn());
les.setCoef(TSKprov);
double sol[];
// System.out.println("Solving...");
try {
    sol = les.getSolucion();
}

// Store solution (time, temperature)
time.add(n, time.get(n - 1) + ds.getDt());
Vector<Double> Taux = new Vector<Double>(1, 1);
for (int i = 0; i < mesh.getNn(); i++) {
    Taux.add(sol[i]);
    // Update nodal temperature
    nodes[i].setT(sol[i]);
}
T.add(Taux);
// Update Tcore for next iteration
Tcore = sol[centralNode]; // Fine 850

// Calculate and store average temperature at the surface
double sum = 0.0;
for (int i = 0; i < mesh.getNbn(); i++) {
    sum = sum + sol[this.boundaryNodes[i]];
}
Tsavg.add((double) sum / mesh.getNbn());

if (Tsavg.get(n) <= Ta2) {
    Tsavg.set(n, Ta2 + 0.0100);
}

} catch (NoPermissionException e) {

e.printStackTrace();
}

// Store F and K matrices for next iteration
Fold = new double[mesh.getNn()];
Kold = new double[mesh.getNn()][mesh.getNn()];
for (int i = 0; i < mesh.getNn(); i++) {
    Fold[i] = F[i];
    for (int j = 0; j < mesh.getNn(); j++) {
        Kold[i][j] = K[i][j];
    }
}

Calendar clock2 = Calendar.getInstance();
Long processingTime = clock2.getTimeInMillis() - clock1.getTimeInMillis();

// PRINT ITERATION RESULTS
System.out.print(time.get(n) + " ");
for (int i = 0; i < this.mesh.getProbes().size(); i++) {
    System.out.print(this.nodes[this.mesh.getProbes().get(i).getTargetNode()].getT() + " ");
}
System.out.print(this.weight.get(n) + " " + hfe.getQ() + " "
                        + hfe.getWl() + " " + hfe.getKa() + " " + avgCtc + " ";
                        + avgRho + " " + avgCp + " " +
                        htc.getAirP().getA_rho()
                                + " " + htc.getAirP().getA_mu() + " "
                                + htc.getAirP().getA_cp() + " " +
                        htc.getAirP().getA_ctc()
                                + " " + RH2 + " " + Ta2 + " " + AirVel2 + " "
                                + Tsavg.get(n) + " " + htc.getH() + " " + htc.getHc()
                                + " "
                                + htc.getHr() + " " + htc.getRe() + " " + htc.getPr()
                                + " "
                                + htc.getHfc() + " " + htc.getFree() + " " +
                        htc.getGr()
                                + " " + htc.getRa() + " " + htc.getFreeNu() + " "
                                + htc.getForcedNu() + " " + processingTime);
System.out.println();
public double Round(double x, int decimalPlace) {
    BigDecimal bd = new BigDecimal(x);
    bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_EVEN);
    x = bd.doubleValue();
    return x;
}

public dataSet readInput(String test, int dt) throws BiffException, IOException {
    String inputFile = "Validations/" + test + "/" + test + ".xls";

    // Open MS Excel File
    WorkbookSettings ws = new WorkbookSettings();
    ws.setLocale(new Locale("en", "EN"));
    Workbook workbook = Workbook.getWorkbook(new File(inputFile), ws);

    // Open Sheet with nodal coordinates
    Sheet input = workbook.getSheet("Input");

    double a = Double.parseDouble("" + input.getCell(1, 1).getContents().toString());
    double b = Double.parseDouble("" + input.getCell(1, 2).getContents().toString());
    double c = Double.parseDouble("" + input.getCell(1, 3).getContents().toString());

    // PROXIMATE COMPOSITION OF HAM
    double prot = Double.parseDouble("" + input.getCell(1, 8).getContents().toString());
    double fat = Double.parseDouble("" + input.getCell(1, 9).getContents().toString());
    double carb = Double.parseDouble("" + input.getCell(1, 10).getContents().toString());
    double salt = Double.parseDouble("" + input.getCell(1, 11).getContents().toString());
    double moist = Double.parseDouble("" + input.getCell(1, 12).getContents().toString());

    // INITIAL WEIGHT OF HAM
    double Wo = Double.parseDouble("" + input.getCell(1, 4).getContents().toString());

    // CHILLING ROOM CHARACTERISTICS
    double Ta = Double.parseDouble("" + input.getCell(1, 14).getContents().toString());
    double AirVel = Double.parseDouble("" + input.getCell(1, 15).getContents().toString());
    double RH = Double.parseDouble("" + input.getCell(1, 16).getContents().toString());

    // INITIAL TEMPERATURE;
    double To = Double.parseDouble(""
double Tosurf = Double.parseDouble("" + input.getCell(1, 18).getContents().toString());

double Ttarget = 4;

// SOLUTION CONTROLS
int N = 20;
// int dt = 300;

// Open Sheet with nodal coordinates
Sheet data = workbook.getSheet("ExperimentalData");
DataSet ds = new DataSet(a, b, c, Wo, prot, fat, carb, salt, moist, Ta,
AirVel, RH, To, Tosurf, Ttarget, dt, N, l, Ta, 0, data .getRows());

double[] timeTest = new double[data.getRows() - 5];
double[] airVelTest = new double[data.getRows() - 5];
double[] RHTest = new double[data.getRows() - 5];
double[] tAirtest = new double[data.getRows() - 5];

for (int row = 5; row < data.getRows(); row++) {
    timeTest[row - 5] = Double.parseDouble("" + data.getCell(1, row).getContents().toString());
    airVelTest[row - 5] = Double.parseDouble("" + data.getCell(32, row).getContents().toString());
    RHTest[row - 5] = Double.parseDouble("" + data.getCell(34, row).getContents().toString());
    tAirtest[row - 5] = Double.parseDouble("" + data.getCell(33, row).getContents().toString());
}

ds.setTestAirTemperature(tAirtest);
ds.setTestAirVelocity(airVelTest);
ds.setTestTime(timeTest);
ds.setTestRelativeHumidity(RHTest);
ds.setCentralProbe(input.getCell(1, 20).getContents().toString());

workbook.close();
return ds;

public int getCentralNode(String centralProbe) {
    int node = 0;
    for (int i = 0; i < this.mesh.getProbes().size(); i++) {
        if (this.mesh.getProbes().get(i).getId().equalsIgnoreCase(centralProbe)) {
            node = this.mesh.getProbes().get(i).getTargetNode();
            System.out.println("Core node=" + node);
            i = this.mesh.getProbes().size() + 1;
        }
    }
    return node;
}