Finite Element Methods

Daphne Lin

2.S976 Finite Element Methods for Mechanical Engineers

16 May 2019

MIT Department of Mechanical Engineering

Project Abstract

Finite Element Methods are becoming a commonly used tool in engineering. Design can be a costly process, especially if a model is built and tested but fails. Iterating through versions of a design can cost millions, even billions depending on the project. Rather than building a physical model and testing for the design's failure points, engineers can use FE packages that use a virtual model (CAD model) of the design and analyze the design computationally, saving resources and lowering cost. While various companies offer different FE packages, the underlying logic is similar across the packages.

The 5 chapters that follow below cover the derivation, implementation, and verification analysis of FE methods as well as provide case studies to showcase the capabilities of the methods. The project primarily uses Matlab to run computational methods and refers to 2.S976 Lecture Notes written by Professor Anthony Patera.

In Chapter 1, we start off by introducing the Rayleigh-Ritz Method which is a simple form of what FE methods do. Using a set of 3 basis functions, we are able to show the importance of selecting basis functions in approximating a solution. We demonstrate the Rayleigh-Ritz Method on 2 different heat transfer models with Neumann/Robin and Dirichlet boundary conditions. We see that when an exact solution has steep features, depending on the basis functions used by the Rayleigh-Ritz Method, we are not always able to capture these features.

In Chapter 2, we introduce a specific set of basis functions called hat functions which we use in our FE method. We also introduce meshes as well as mesh refinement; we assign a set of hat functions to each element in our mesh which allows us to capture finer features in the solution. Using the same Model 2 from Chapter 1 as well as an additional Model 3, we can demonstrate

that the FE method using hat functions and mesh refinement improves on the method we had used in Chapter 1.

Chapter 3 is the final chapter which deals with heat transfer problems. After introducing the Finite Difference method which incorporates time-dependency into our solutions, we run a case study of flipping burgers (modeled as an infinite fin). Using the FD-FE method, we are able to predict and optimize the best time to flip a burger so that health (raw vs cooked burger) and taste (Maillard) constraints are met.

Chapter 4 is the first chapter to deal with structural analysis. We modify the FE method, so we can account for beam bending. Rather than using the hat functions from our heat transfer FD-FE, we use Hermitian basis functions in our beam bending FE method. Using the Euler-Bernoulli beam model which holds for slender beams, we optimize (tune) and design xylophone bars. We see the effect of having a numerical model which may not necessarily hold for what we are analyzing.

The final chapter applies the FE method in self-buckling analysis, a serious and common failure in the real world. Using a new model for self-buckling, we aim to design the tallest structure that will not buckle under its own weight (with given constraints on volume, radius dimensions, and radius profile continuity). Non-dimensionalizing our analysis, we can use the figure of merit (FOM) to judge our designs (a higher FOM represents a taller structure). In this project, the highest FOM achieved was 1.645.

This project, as a whole, gives insight into FE package algorithms analyze models. In addition to the logic behind the computations, we also learned how to interpret the solutions given to us by the FE method. Misinterpreting results can be just as detrimental as a bad design. The FE method

paired with good engineering judgment proves to be a powerful tool, one that not only saves time and cost but also encourages safe designs.

Acknowledgements

Thank you Prof. Anthony Patera for instructing the course, providing well-written lecture notes, and providing the FE method Matlab codes. This has been one of the most interesting and most rewarding classes I've taken at MIT, and I'm so glad I was lucky enough to work with and learn from you. Thank you for not only being a knowledgeable professor but also a professor that cares about each student.

Thank you James Penn for providing physical representations and models of the case studies we used throughout the semester.

Chapter 1: The Rayleigh-Ritz Method

1.1 – Abstract

The Rayleigh-Ritz Approximation Method aims to find the lowest energy solution to a model by linearly combining a given set of basis functions. It is a fundamental idea behind the Finite Element Method and can output an accurate solution if given an appropriate set of basis functions.

In this chapter, we explored the Rayleigh-Ritz Method using two different heat transfer models. The first model (Model 1) is a conical frustum with Neumann/Robin heat flux boundary conditions at both ends. The second model (Model 2) is a right-cylinder thermal fin with a Dirichlet temperature boundary condition on the left end ($x = 0$) and a Neumann/Robin heat flux boundary condition on the right end ($x = L$).

To implement the Rayleigh-Ritz Method, we defined energy functionals for both models based on the general form of a Neumann/Robin and Dirichlet problem. The minimization proposition states that the exact solution of a model will result in the global minimum of the energy functional. Subsequently, accurate approximations of the exact solution will have low values of the energy functional. Knowing this, we used the minimization proposition to determine the Rayleigh-Ritz constants that would, when combined with the corresponding basis functions, result in the lowest value of the energy functional.

For this implementation of the Rayleigh-Ritz Method, we considered two sets of basis functions. The first set of basis functions included the exact solution to the model. The second set of basis functions included a constant function, a linear function, and a quadratic function. As expected from the minimization proposition, running the Rayleigh-Ritz Method using the first set of basis functions resulted in an approximation that was equal to the exact solution of the models.

Running the Rayleigh-Ritz Method using the second set of basis functions highlighted the convergence of the method as well as the importance of selecting good basis functions. The addition of a basis function in a set will result in either the previous/same approximation or a better approximation but never a worse approximation. However, the accuracy of the approximation depends on the set of basis functions.

Overall, the Rayleigh-Ritz Method resulted in acceptable approximations for both Models when parameter values were low. However, as the parameter values increased, the exact solutions gained sharper features, and the Rayleigh-Ritz Method was unable to output an accurate approximation.

1.2 - Implementation

Energy Functional

In order to run a Rayleigh-Ritz Approximation for the given models, we first define our energy functional in terms of *w*, a continuous function of candidate temperatures. To find the energy functional, we can use the heat transfer differential equation for the given system as well as boundary conditions as shown below for both Model 1 and Model 2. By scaling the boundary conditions of Model 1 by $\kappa(x)$, we can use the general form of a Neumann/Robin problem to express the energy functional, giving us the following function:

$$\Pi(w) = \frac{1}{2}\int_0^L \kappa\pi R_0^2 \left(1 + \beta\left(\frac{x}{L}\right)\right)^2 \left(\frac{dw}{dx}\right)^2 dx - q_1\pi R_0^2 + \eta_2(w - u_\infty)(\pi R_0^2(1 + \beta)^2) \qquad (1.1)$$

In Model 2, we can express the energy functional using the Dirichlet form:

$$\Pi(w) = \frac{1}{2}\int_0^L \kappa \left(\frac{dw}{dx}\right)^2 dx + \frac{\mu_0\kappa}{L^2}\left(\frac{1}{2}w^2 - u_\infty w\right) dx \qquad (1.2)$$

To ensure we have defined the appropriate energy functionals, we propose a temperature, *u,* which will minimize the energy and prove that any perturbations, *v,* from *u* will increase the energy. We set our candidate temperature, *w,* equal to $u + v$. By rearranging the terms in the functional, we can define E_1, E_2, and E_3. E_1 will only depend on the temperature, *u* while E_3 will only depend on the value, *v*. E_2 should become zero if the energy functional and grouped terms are correct. This leaves us with the following:

$$\Pi(u + v) = E_1(u) + E_3(v) \qquad (1.3)$$

Comparing $\Pi(u)$ and $\Pi(u + v)$, we see that $\Pi(u)$ will always be greater than $\Pi(u + v)$ if E_3 is positive for any value of *v*. If our energy functional is correct, this should be true, and we can continue with our Rayleigh-Ritz Approximation.

Rayleigh-Ritz Approximation

For our Rayleigh-Ritz Approximation, we define a set of basis functions. We then find a set of constants, α, which, when multiplied with the basis functions, minimizes the energy functional (equations 1.1 and 1.2). We define the set of constants that minimizes the energy functional as α^{RR} or our Rayleigh-Ritz constants.

The general solution to minimize the energy functionals is:

$$A\alpha^{RR} = F \qquad (1.4)$$

where matrices *A* (SPD) and *F* (column vector) can be found from E_2 of the energy functionals.

For our particular models, the entries of matrices A and F are defined as follows:

Model 1

$$A_{ij} = \int_0^L \kappa\pi R_0^2 \left(1 + \beta\left(\frac{x}{L}\right)\right)^2 \frac{d\psi_i}{dx}\frac{d\psi_j}{dx}\, dx + \eta_2\pi R_0^2(1+\beta)^2\psi_i(L)\psi_j(L) \qquad (1.5)$$

$$F_i = q_1\pi R_0^2\psi_i(0) + \eta_2\pi R_0^2(1+\beta)^2 u_\infty\psi_i(L) \qquad (1.6)$$

Model 2

$$A_{ij} = \int_0^L \kappa\frac{d\psi_i}{dx}\frac{d\psi_j}{dx} + \frac{\mu_0\kappa}{L^2}\psi_i\psi_j\, dx \qquad (1.7)$$

$$F_i = \int_0^L \frac{\mu_0\kappa}{L^2}u_\infty\psi_i\, dx \qquad (1.8)$$

Once we solve for the matrix α^{RR}, we multiply α^{RR} with the basis functions and linearly combine the results, giving us the Rayleigh-Ritz Approximation for the models.

1.3 - Results

In the Matlab code, we define two cases: *exactinclude* and *constlinquad*. In *exactinclude*, we include the exact solution to the model as part of the set of basis functions. In *constlinquad*, we consider a constant, a linear equation, and a quadratic equation as the set of basis functions.

Case 1: exactinclude

Based on the results from *exactinclude*, we can conclude that the Rayleigh-Ritz Approximation code appears to work. When forming the approximation of the solution, the Rayleigh-Ritz method aims to minimize the given energy functional. The exact solution will result in the global minimum of the energy functional. We expect the corresponding Rayleigh-Ritz constant of the exact solution to be 1 and the Rayleigh-Ritz constants of the remaining basis functions to be 0, giving us a Rayleigh-Ritz Approximation that equals the exact solution. Based on the graphs generated by the code, the approximation and exact solution appear to be the same. Additionally,

checking the values of α^{RR} also confirms that *exactinclude* works as expected. The exact solution of Model 1

$$u = u_\infty + \frac{q_1 L}{\kappa}\left(\frac{1+\beta+\frac{\kappa}{\eta_2 L}}{(1+\beta)^2} - \frac{\left(\frac{x}{L}\right)}{1+\beta\left(\frac{x}{L}\right)}\right) \qquad (1.9)$$

is included in the set of basis functions, and the output is shown below.

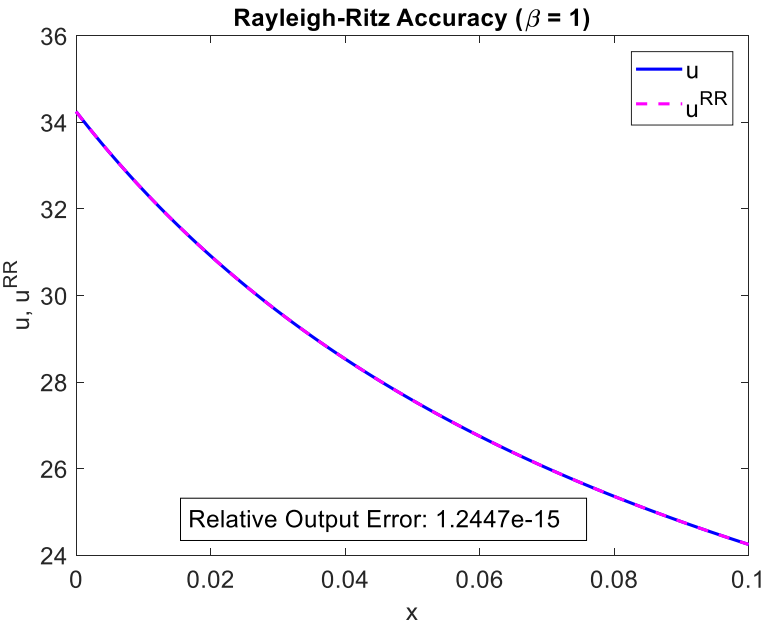


Figure 1.1. Model 1 *exactinclude* output when β equals 1.

As β increases and steepens the solution's features, *exactinclude* is still able to provide the exact solution as the approximation.

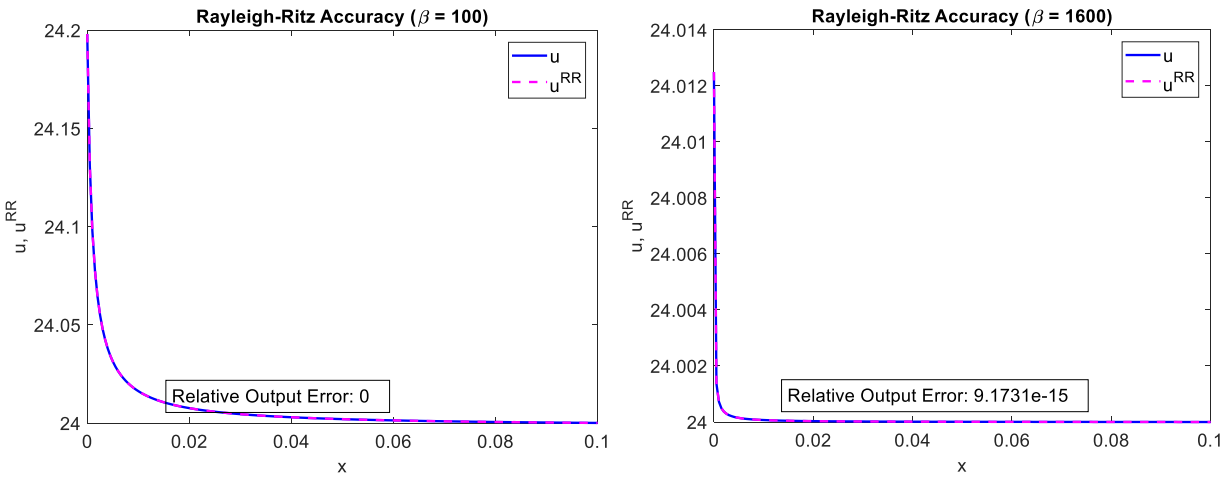


Figure 1.2. Model 1 *exactinclude* outputs when β equals 100 and 1600.

Similarly, for Model 2, the exact solution included in the basis functions is:

$$u = u_\infty + (u_{\Gamma_1} - u_\infty)\frac{\cosh\left(\sqrt{\mu_0}\left(1-\frac{x}{L}\right)\right)}{\cosh\left(\sqrt{\mu_0}\right)} \qquad (1.10)$$

where μ_0 is:

$$\mu_0 = \frac{\eta_3 P_{cs} L^2}{\kappa A_{cs}} \qquad (1.11)$$

The approximation when η_3 equals 1 [W/mºC] is graphed below. Again, we see that *exactinclude* is able to output the exact solution as the approximation.

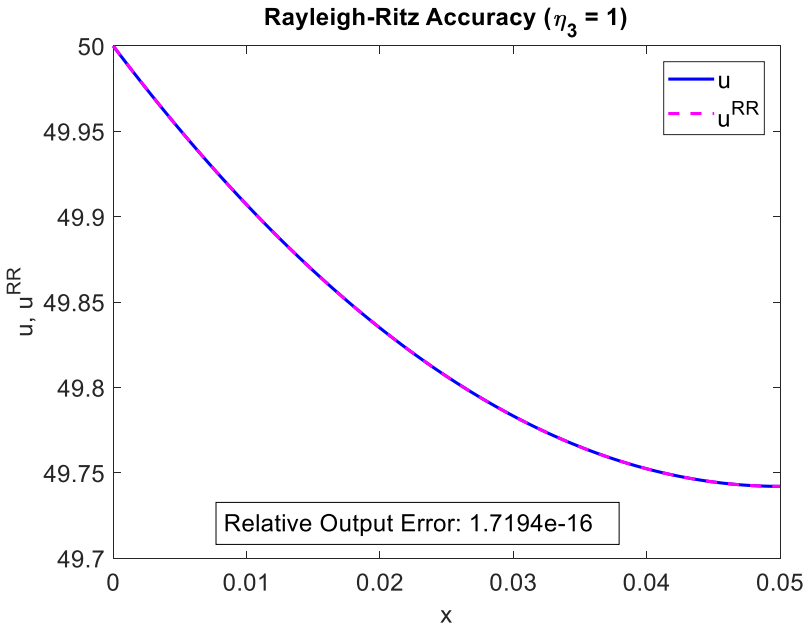


Figure 1.3. Model 2 *exactinclude* output when η_3 equals 1 [W/mºC].

As η_3 increases, the behavior of the exact solution changes, and *exactinclude* continues to solve the Rayleigh-Ritz Approximation with the exact solution.

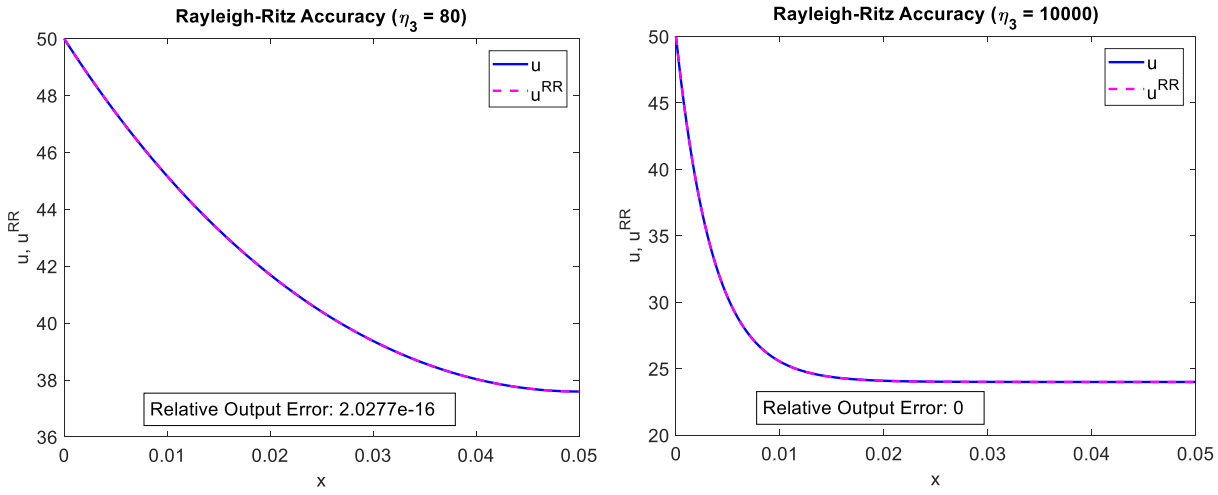


Figure 1.4. Model 2 *exactinclude* outputs when η_3 equals 80 [W/mºC] and 10000 [W/mºC].

Case 2: *constlinquad*

In *constlinquad*, we denote the number of basis functions to include as the integer n^{RR}. For Model 1, n^{RR} can range from 1 to 3 while in Model 2, the constant basis function is always included to satisfy the temperature boundary condition and n^{RR} can be 1 or 2.

If we initially start with one basis function (i.e. $\Psi_1 = 1$), we expect the Rayleigh-Ritz approximation to be an α^{RR} multiplied with Ψ_1. If we add another basis function (i.e. $\Psi_2 = x$) and approximate again, the solution will be a linear combination of the two basis functions. There are two possible outcomes from this: the solution stays the same as before or the solution becomes a better approximation. If the addition of Ψ_2 does not further minimize the energy functional, the solution given by the Rayleigh-Ritz solution will be the same as when only Ψ_1 was considered. If the addition of Ψ_2 results in a smaller energy than when only Ψ_1 is considered, both α^{RR} terms will be non-zero, and the solution is a linear combination of the two basis functions. This result gives a solution that should be closer to the exact solution since the value of the energy functional has decreased.

We expect the results of *constlinquad* to follow the situations described above. When we add a basis function (while keeping the previous functions) to the set, the approximation and energy should either stay the same (compared to the previous) or get closer to the exact solution.

Looking at Model 1 results, when we only include $\Psi_1 = 1$ in our basis function set, we get the following when β equals 1. As expected, the Rayleigh-Ritz solution, u^{RR}, is not similar to the exact solution.

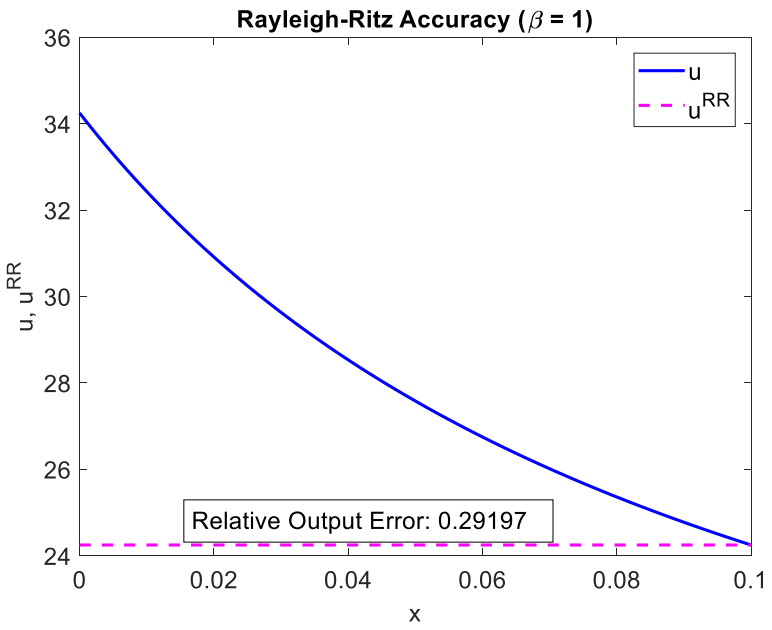


Figure 1.5. Model 1 *constlinquad* outcome when β equals 1 and n^{RR} equals 1. The value of the energy function, $\Pi(u^{RR})$, is -36.9491 [W].

Adding a second basis function, Ψ_2 = x, gives a slightly more accurate Rayleigh-Ritz solution. We also see the energy functional has decreased slightly as expected. However, the Rayleigh-Ritz solution is limited to linear basis functions (resulting in a linear solution) and is unable to capture the curvature of the exact solution.

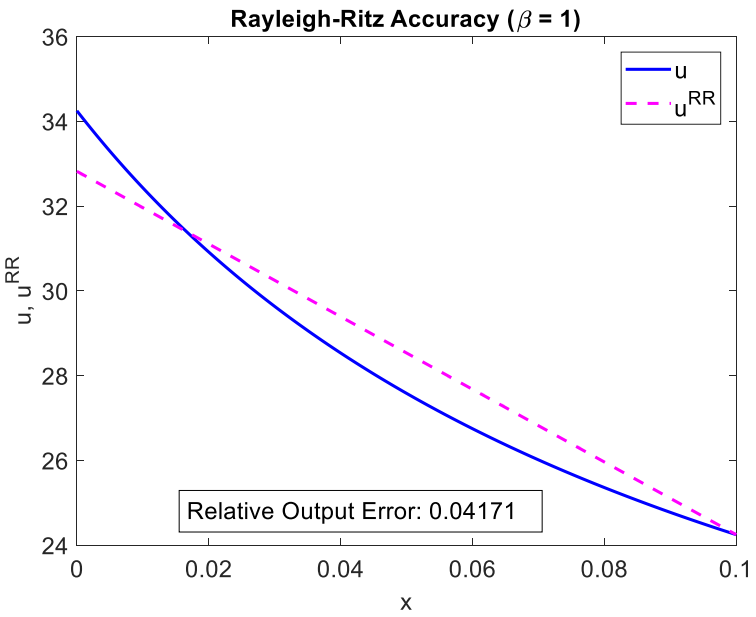


Figure 1.6. Model 1 *constlinquad* outcome when β equals 1 and n^{RR} equals 2. The value of the energy function, $\Pi(u^{RR})$, is -37.0837 [W].

Finally, we add in the third basis function, $\Psi_2 = x^2$. The Rayleigh-Ritz Approximation follows the exact solution much more closely than before; additionally, the value of the energy functional has decreased again.

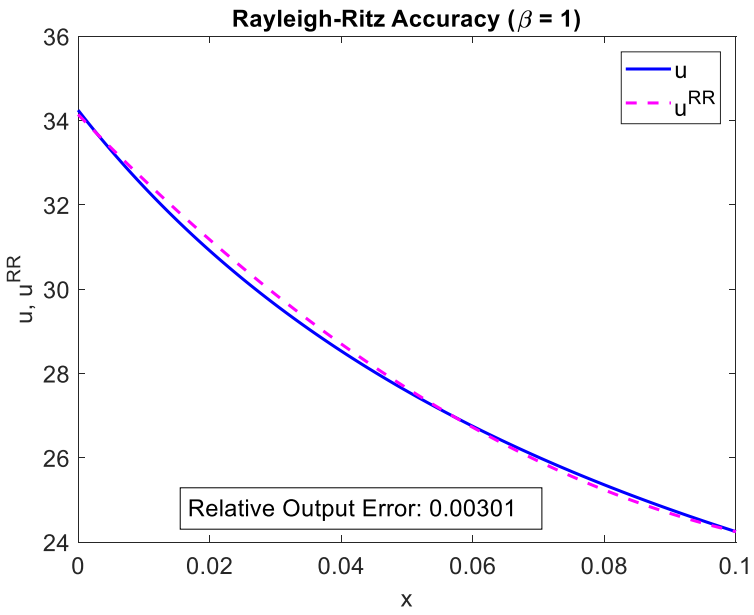


Figure 1.7. Model 1 *constlinquad* outcome when β equals 1 and n^{RR} equals 3. The value of the energy function, $\Pi(u^{RR})$, is -37.1045 [W].

Looking at the results of *constlinquad* for Model 2, we define a basis function to be $\Psi_0 = 1$ to satisfy the Dirichlet boundary condition. Our first basis function, $\Psi_1 = x$, gives us the following approximation which doesn't capture the exact solution very well.

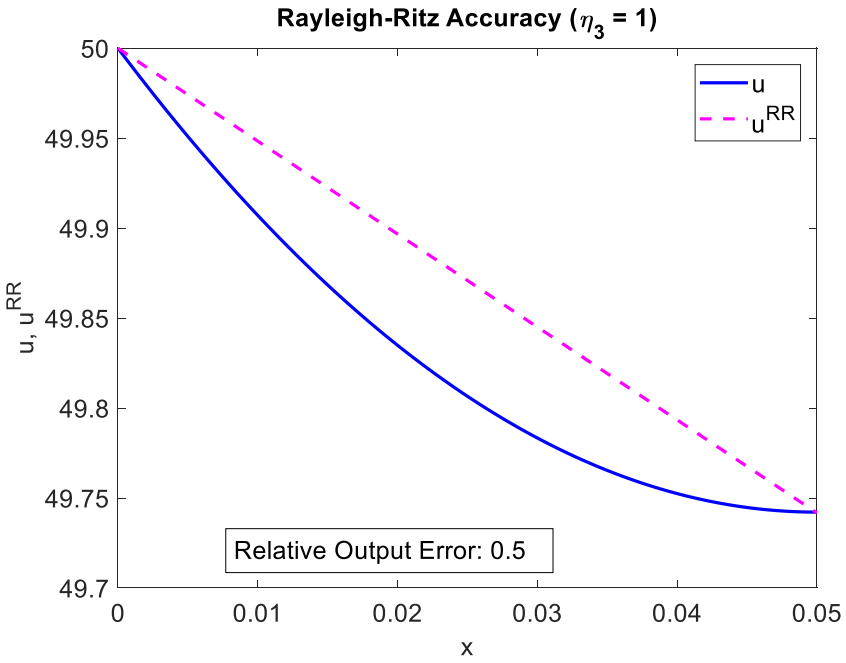


Figure 1.8. Model 2 *constlinquad* outcome when η_3 equals 1 [W/m^{0}C] and n^{RR} equals 1. The value of the energy function, $\Pi(u^{RR})$, is 966.4238 [W].

Adding the second basis function, $\Psi_2 = x^2$, improves the Rayleigh-Ritz Approximation and as expected, decreases the value of the energy functional.

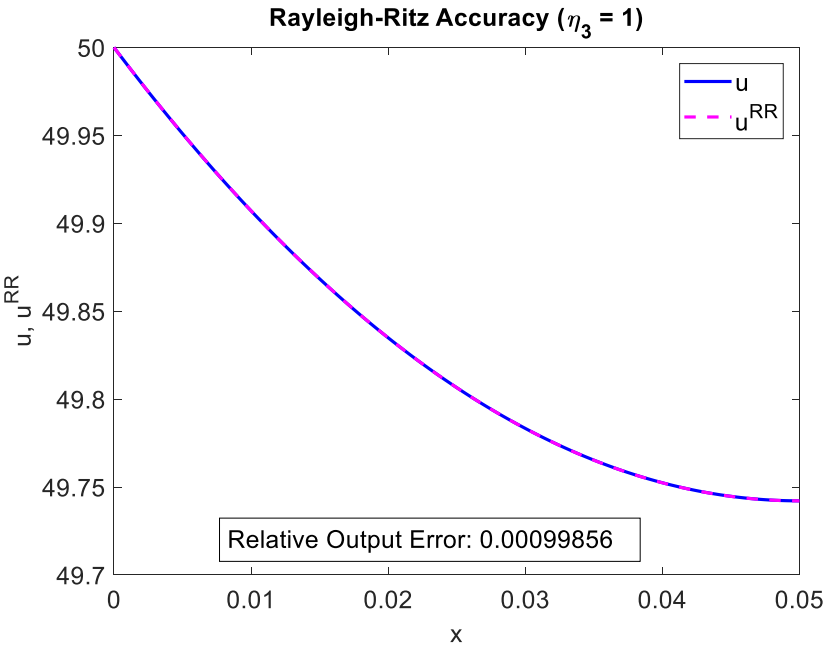


Figure 1.9. Model 2 *constlinquad* outcome when η_3 equals 1 [W/m^{o}C] and n^{RR} equals 2. The value of the energy function, $\Pi(u^{RR})$, is 955.291 [W].

Parameter Effects

We have verified that the code for *constlinquad* works and outputs reasonable Rayleigh-Ritz Approximations for Model 1 when β equals 1 and for Model 2 when η_3 equals 1 [W/m^{o}C]. However, when we vary β and η_3, we see that the exact solution changes and can affect the Rayleigh-Ritz Approximation.

Looking at the exact solution of Model 1, we see that if the parameter β is increased, the solution becomes steeper on the left and decays faster. When β is 100, the approximation when n^{RR} is 3 appears to have a larger relative error than the approximation when β is 1.

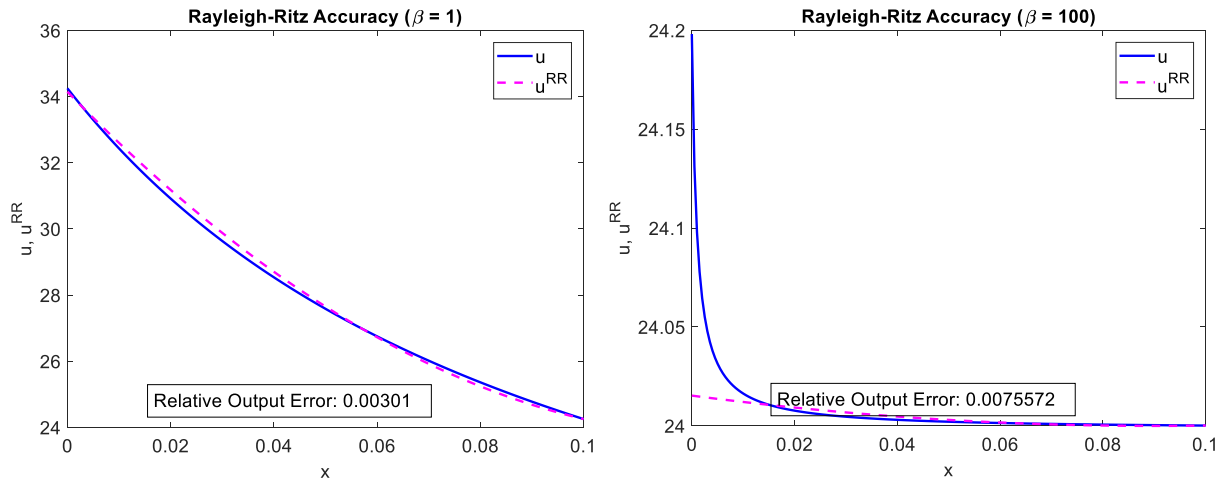


Figure 1.10. Model 1 *constlinquad* when β equals 1 and 100 when all basis functions are considered.

Here, we see a potential problem with the Rayleigh-Ritz Method. When β becomes large, the Rayleigh-Ritz solution fails to capture the steepness on the left of the solution. Because the approximation is limited to a linear combination of the basis functions, if the basis functions do not resemble the shape of the solution, it is difficult for the Rayleigh-Ritz Method to output a close approximation to the exact solution. This is particularly noticeable when β becomes 1600; the best the Rayleigh-Ritz Method can achieve is to capture the right side of the solution.

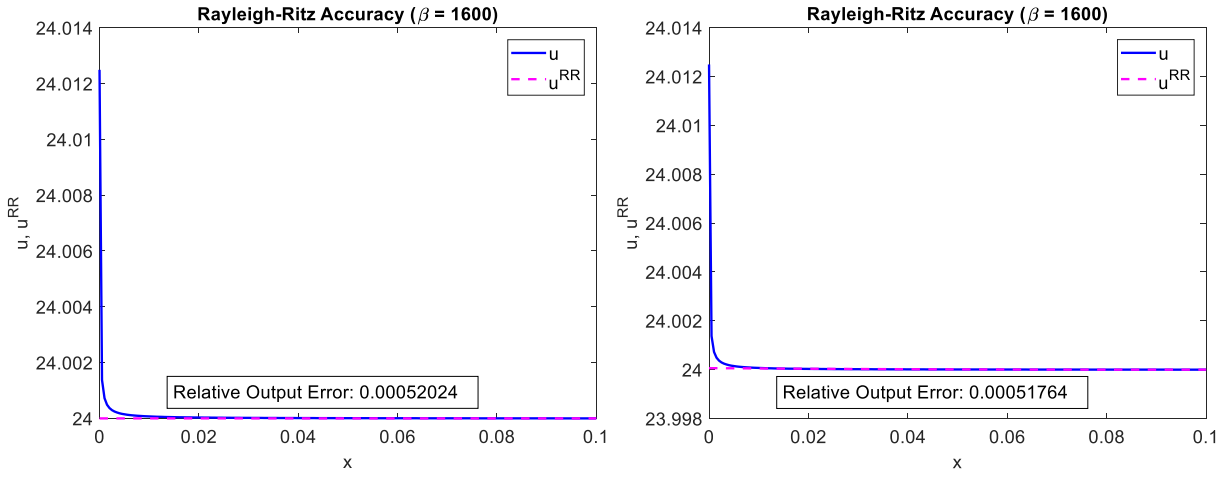


Figure 1.11. Model 1 *constlinquad* when n^{RR} equals 1 and 3 and β equals 1600.

We see that as we increase n^{RR} for this case, *constlinquad* outputs an approximation that places a heavier weight on Ψ_1 (constant function) than the other two basis functions. Even with the addition of Ψ_2 and Ψ_3, the solution that minimizes the energy functional is mostly a constant function. The Rayleigh-Ritz constant corresponding to Ψ_1 remains at about 24 while the Rayleigh-Ritz constants of the other two basis functions are on the magnitude of 0.001.

With Model 2, a similar result occurs as η_3 increases. Again, the basis functions are unable to capture the shape of the exact solution.

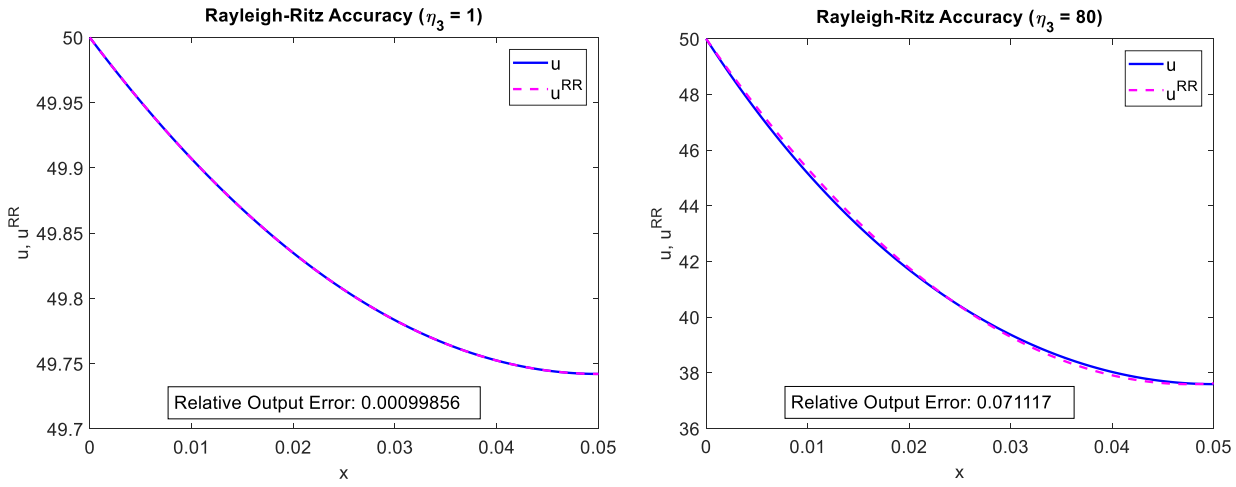


Figure 1.12. Model 2 *constlinquad* outputs when η_3 equals 1 [W/m^{o}C] and 80 [W/m^{o}C] when all basis functions are considered.

Because Model 2 has a Dirichlet constraint when x = 0, we see the Rayleigh-Ritz method outputs a solution that satisfies the boundary condition, even at high η_3 values.

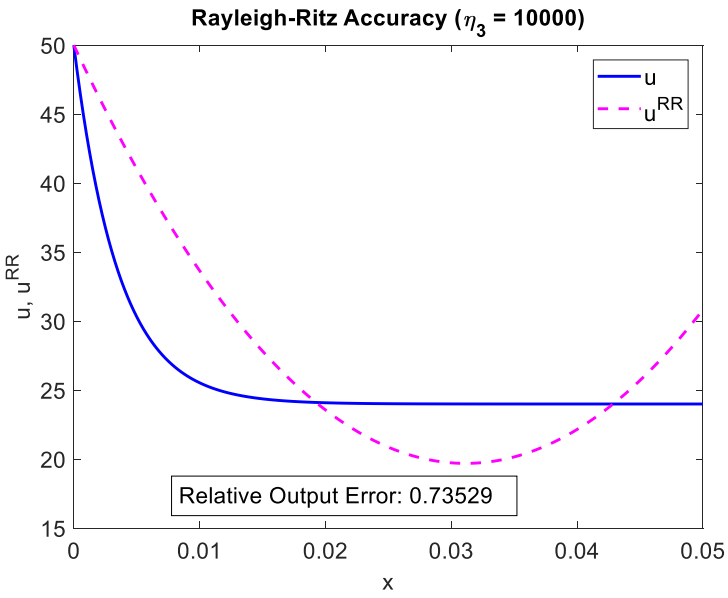


Figure 1.13. Model 2 *constlinquad* output when η_3 equals 10000 [W/m^{o}C] when all basis functions are considered.

The rest of the approximation is inaccurate because the shapes of the basis functions cannot be linearly combined to form the features of the exact solution.

Comparison of *exactinclude* and *constlinquad*

While *constlinquad* is able to provide approximations that are reasonably close to the exact solution with the given basis functions, *exactinclude* will be able to give the exact solution as the approximation because of the basis function set. Looking at our results, we would expect to see *exactinclude* to always have a lower value energy functional than *constlinquad* since *exactinclude* will be able to reach the global minimum of the functional. The table below compares energy functional values between the different cases run.

Table 1.1. Values of Energy Functional given by *constlinquad* and *exactinclude* in [W].

Model 1				
β	***constlinquad nRR = 1***	***constlinquad nRR = 2***	***constlinquad nRR = 3***	***exactinclude***
1	-36.9491	-37.0837	-37.1045	-37.1061
100	-92297.2276	-92297.2277	-92297.2278	-92297.2307
1600	-23191297.0361	-23191297.0361	-23191297.0361	-23191297.0362

Model 2			
η3 [W/m^{o}C]	***constlinquad nRR = 1***	***constlinquad nRR = 2***	***exactinclude***
1	966.4238	955.291	955.291
80	-61078.2609	-96087.3776	-96359.6564
10000	-39950738.9163	-48793722.8209	-52819958.1592

1.4 – Final Thoughts

The Rayleigh-Ritz Method provides a simple and straightforward way to approximate solutions of models. However, the accuracy of the method depends heavily on the set of basis functions chosen. Because the basis functions must be continuous functions, it may be difficult to capture characteristics of the exact solution. As we saw with the models considered in this chapter, if the exact solution of a model has a curvature, it is difficult to capture the behavior if the basis functions are linear. Similarly, if the basis functions do not contain similar shapes to the exact solution, it is also difficult to output a good approximation.

1.5 – References

1. Anthony T Patera, 2019, “Rayleigh-Ritz Method for Expositional Problem.”
2. Anthony T Patera, 2019, “Rayleigh-Ritz Method for General 1D 2nd-Order SPD BVP.”

Chapter 2: The Finite Element Method for 1D 2nd-Order SPD BVPs

2.1 - Abstract

In this chapter, we discuss the implementation and results of the Finite Element Method (FE Method).

Building off the Rayleigh-Ritz Method that we discussed in Chapter 1, the FE Method is implemented in a similar way. The main difference between the Rayleigh-Ritz Method and the FE Method is the set of basis functions used. In Rayleigh-Ritz, we chose basis functions, but with the FE Method, we use a special set of functions known as hat functions. These functions are defined based off a mesh consisting of nodes and elements that span the length of the domain of interest.

After defining the hat functions, solving for the approximation, u_h, follows the same logic as the Rayleigh-Ritz Method. However, in implementation, we take several steps to create cleaner and more versatile code. We map the hat functions to 2 shape functions and rewrite elements in matrices A and F in terms of these shape functions. From there, we use direct stiffness to form A and F before adding in the boundary condition elements. Finally, we solve for the coefficients that multiply with hat functions to linearly form the approximation.

In addition to the implementation of the FE Method, we also discuss sources of error in this chapter. Error primarily comes from three factors: implementation, numerical specification, and mathematical model. We can identify when there is an implementation error using convergence plots that plot error over norms that we define. Convergence plots are a useful tool that give us confidence to whether or not our code works. Additionally, we can define upper bounds based off the plots for our models to account for worst case scenarios.

The FE Method is able to approximate solutions with high accuracy, especially with a refined mesh. When properly implemented, it is also a relatively fast method that runs on the order of O(n) where n is determined by the number of nodes.

2.2 - Models

To test our FE Method, we define three models, two of which are from Chapter 1 (Models 1 and 2). The models are redefined below.

Model 1 consists of heat conduction through a conical frustum with insulated lateral surfaces. The heat transfer differential equation is:

$$-\kappa \frac{d}{dx}\left(\pi R_0^2 \left(1+\beta \frac{x}{L}\right)^2 \frac{du}{dx}\right) = 0 \text{ in } \Omega = (0, L) \qquad (2.1)$$

The boundary conditions on the left and right surfaces are:

$$\kappa \frac{du}{dx} = -q_1 \text{ on } \Gamma_1 = \{0\} \qquad (2.2)$$

$$-\kappa \frac{du}{dx} = \eta_2 (u - u_\infty) \text{ on } \Gamma_2 = \{L\} \qquad (2.3)$$

Note that these boundary conditions correspond to a Neumann/Robin boundary condition in which the heat flux and heat transfer coefficient are defined.

The output of interest is:

$$s \equiv u(0) \text{ at } x = 0 \qquad (2.4)$$

Model 2 consists of a right-cylinder thermal fin.

The heat transfer differential equation is:

$$-\kappa A_{cs}\ \frac{d^2u}{dx^2} + \ \eta_3 P_{cs}(u - u_\infty) = 0 \text{ in } \Omega = (0, L) \qquad (2.5)$$

The boundary conditions on the left and right surfaces are:

$$u = u_{\Gamma 1} \text{ on } \Gamma_1 = \{0\} \qquad (2.6)$$

$$-\kappa \frac{du}{dx} = 0 \text{ on } \Gamma_2 = \{L\} \qquad (2.7)$$

The boundary condition on the left face is a Dirichlet condition in which the temperature is defined while the boundary condition on the right face is a Neumann/Robin condition.

The output of interest is:

$$s \equiv \ -\kappa \frac{du}{dx} \text{ at } x = 0 \qquad (2.8)$$

Model 3 consists of a wall with equal cross sectional area and different ambient temperatures on both sides.

The heat transfer differential equation is:

$$-\kappa A_{cs} \frac{d^2u}{dx^2} = 0 \ \Omega = (0, L) \qquad (2.9)$$

The boundary conditions on the left and right surfaces are:

$$-\kappa A_{cs} \frac{du}{dx} = \ \eta_4(u - u_\infty)A_{cs} \text{ on } \Gamma_1 = \{0\} \qquad (2.10)$$

$$-\kappa A_{cs} \frac{du}{dx} = \ \eta_4(u - u_\infty)A_{cs} \text{ on } \Gamma_2 = \{L\} \qquad (2.11)$$

In this model, we have defined a Neumann/Robin boundary condition on both of our surfaces unlike in Model 1 where the left side is only a Neumann boundary condition (eqn. 2.2).

The outputs of interest are:

$$s \equiv u(0) \text{ at } x = 0 \qquad (2.12)$$

$$s \equiv u(L) \text{ at } x = L \qquad (2.13)$$

2.3 - Implementation

In this chapter, we implement a specific case of the Rayleigh-Ritz Method to create our Finite Element Method code.

Hat Functions

In Chapter 1, we saw the important of choosing basis functions for a Rayleigh-Ritz approximation. For the Finite Element Method, we use special basis functions know as hat functions or φ functions for our approximation. To help define these hat functions, we set up a mesh along the length we are interested it. We set points along the length which we will call nodes (x) and define the spaces between the points elements (T) of length, h. In this chapter, we will be using a uniform mesh meaning the nodes are spaced equally throughout the length of interest. The hat functions will correspond to each node meaning for n_{node} nodes, there will be n_{node} hat functions. Hat functions must satisfy the following conditions:

1. φ_i is a continuous function.
2. φ_i is 1 at node i (x^i) and 0 at all other nodes.
3. φ_i is linear.

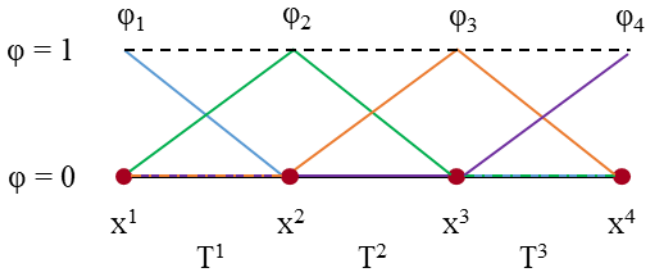


Figure 2.1. Example of a mesh with hat functions satisfying the conditions above. In this mesh, there are 4 nodes, 3 elements, and 4 hat functions.

Given these conditions, derivatives of hat functions must also have the following properties:

1. φ_i' is piecewise constant.
2. φ_1' is $-\frac{1}{h^1}$ in T^1 and 0 through all other elements.
3. φ_{node}' is $\frac{1}{h^{node}}$ in T^{node-1} and 0 through all other elements.
4. For nodes 2 through $n_{node} - 1$, φ_i' is $-\frac{1}{h^{i-1}}$ in T^{i-1}, $\frac{1}{h^i}$ in T^i, and 0 through all other elements.

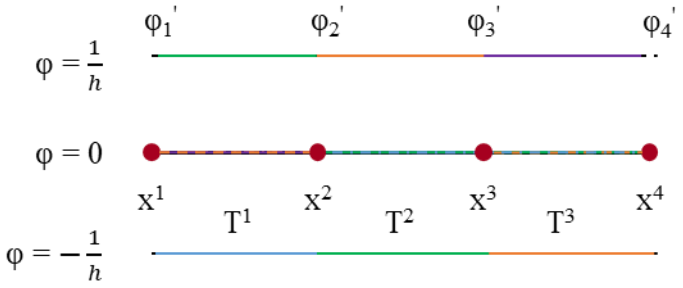


Figure 2.2. Example of mesh from Fig. 2.1 with derivatives of hat functions satisfying the properties above.

By using hat functions, we are able to use the same set of basis functions for both Neumann/Robin and Dirichlet boundary condition problems. In the case of Dirichlet, we no longer need to define a special basis function that equals 1 at the boundary and 0 everywhere else because the hat function takes care of this requirement.

Mapping and Matrices

Once we have defined our hat functions, we can solve for our approximation, $u_h(x)$, similarly to Chapter 1. The matrix of coefficients that we will use to define $u_h(x)$ is related as follows:

$$Au_h = F \qquad (2.14)$$

To solve for the components of matrices A and F, we first solve for the matrices without considering the boundary conditions. Additionally, we denote A as the sum of matrices K and M. The general form of the matrices' components are as follows:

$$K_{ij}^N = \int_0^L \kappa(x) \frac{d\varphi_i}{dx} \frac{d\varphi_j}{dx} dx \qquad (2.15)$$

$$M_{ij}^N = \int_0^L \mu(x) \varphi_i \varphi_j dx \qquad (2.16)$$

$$A^N = K^N + M^N \qquad (2.17)$$

$$F_i^N = \int_0^L f_\Omega(x) \varphi_i dx \qquad (2.18)$$

To simplify our FE method code, we solve for matrices A and F by introducing a new concept of mapping. Because our hat functions share common features, we can map them to 2 shape functions (shown below); by doing so, we can shorten our code and will not need to write new code to solve for every instance in matrices A and F whenever the size of an element changes in the mesh.

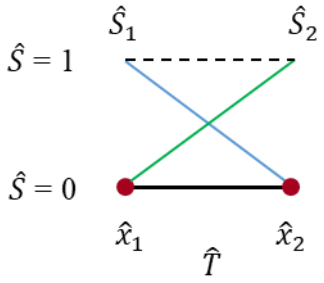


Figure 2.3. Example of shape functions and quadrature points used for mapping.

We rename the nodes as $x^{lg(\alpha,m)}$ where α can either be 1 or 2 (left or right, respectively) and m corresponds with the element number. Rewriting Equations 2.15 through 2.18 above gives us the following:

$$K_{\alpha\beta}^{elm} = \int_{x^{\lg(1,m)}}^{x^{\lg(2,m)}} \kappa(x) \frac{d\varphi}{dx} \lg(\alpha, m) \frac{d\varphi}{dx} \lg(\beta, m) dx \quad (2.19)$$

$$M_{\alpha\beta}^{elm} = \int_{x^{\lg(1,m)}}^{x^{\lg(2,m)}} \mu(x) \varphi_{\lg(\alpha,m)} \varphi_{\lg(\beta,m)} dx \quad (2.20)$$

$$A^N = K^N + M^N \quad (2.21)$$

$$F_{\alpha}^{elm} = \int_{x^{\lg(1,m)}}^{x^{\lg(2,m)}} f_{\Omega}(x) \varphi_{\lg(\alpha,m)} \, dx \quad (2.22)$$

The nodes are then mapped to quadrature points $\hat{x}_1$ and $\hat{x}_2$ such that x can be written as $x^{lg(\alpha,m)}+h^m\hat{x}$. To get the equations above in terms of the mapping, we can write dx as the product of the size/length of the element, h^m, and $d\hat{x}$. From there, we can write $\frac{d}{dx}$ as $\frac{1}{h^m}\frac{d}{d\hat{x}}$. Subbing in the shape functions into the equations above gives:

$$K_{\alpha\beta}^{elm} = \frac{1}{h^m} \int_0^1 \kappa\left(x^{\lg(1,m)} + h^m\hat{x}\right) \hat{S}'_{\alpha}(\hat{x}) \hat{S}'_{\beta}(\hat{x}) d\hat{x} \quad (2.23)$$

$$M_{\alpha\beta}^{elm} = h^m \int_0^1 \mu\left(x^{\lg(1,m)} + h^m\hat{x}\right) \hat{S}_{\alpha}(\hat{x}) \hat{S}_{\beta}(\hat{x}) \, d\hat{x} \quad (2.24)$$

$$A^N = K^N + M^N \quad (2.25)$$

$$F_{\alpha}^{elm} = \int_0^1 f_{\Omega}\left(x^{\lg(1,m)} + h^m\hat{x}\right) \hat{S}_{\alpha}(\hat{x}) \, d\hat{x} \quad (2.26)$$

Using direct stiffness to construct our matrices in Matlab, we then add the proper boundary conditions. In the case of a Neumann/Robin boundary condition, we add γ to the boundary components of A and f_Γ to the boundary components of F. For a Dirichlet boundary condition, we use the following to modify A and F:

$$Au_h = F - u_\Gamma b \qquad (2.27)$$

where vector b can be taken from the column of matrix A that corresponds to the boundary (i.e. if a Dirichlet condition was applied to the left most surface, vector b would be the first column of matrix A).

After finding matrics A and F, we can solve for the vector u_h using Equation 2.14 where each component of the vector is a coefficient, u_{hi}. We then linearly combine the products of the coefficients and hat functions to solve for the FE approximation, u_h.

2.4 - Results

Using the FE method, we approximate solutions for all 3 models and compare u_h to exact solutions over a series of mesh refinements (adding nodes/elements and decreasing h^m). Additionally, we calculate and graph the error between the approximation and exact solutions as well as the error between each mesh.

form_elem_mat_sver

This function forms the elemental matrices A and F as described in Equations 2.23 through 2.26. This function does not impose boundary conditions.

We will verify that our implementation is correct using Model 2 by running the FE method code with 6 uniform refinements. Our initial mesh consists of 6 elements and gives the following approximation:

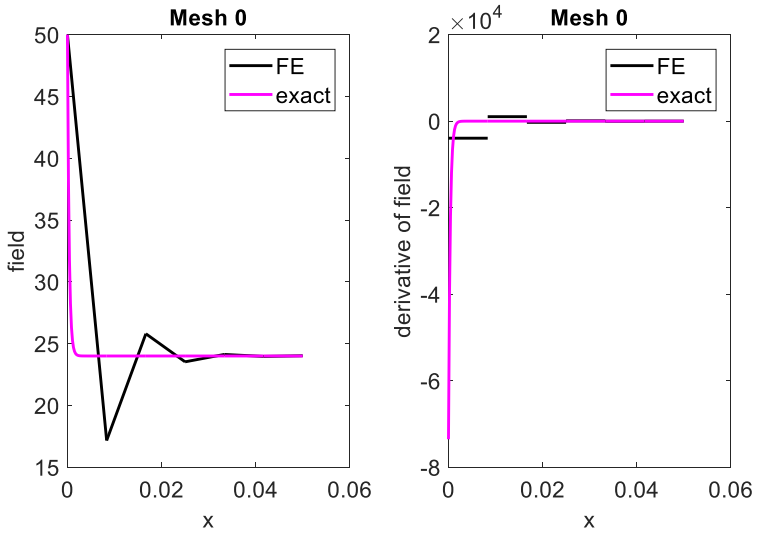


Figure 2.4. Temperature and derivative approximations compared to the exact solution of Model 2 using an initial mesh with 6 elements.

Looking at the approximation given by the initial mesh, we can see the influence of the hat functions. This approximation is not close to the exact solution. The derivative approximation is also off and doesn't capture the steep left side of the exact solution's derivative.

After refining the mesh 6 times, our approximation becomes:

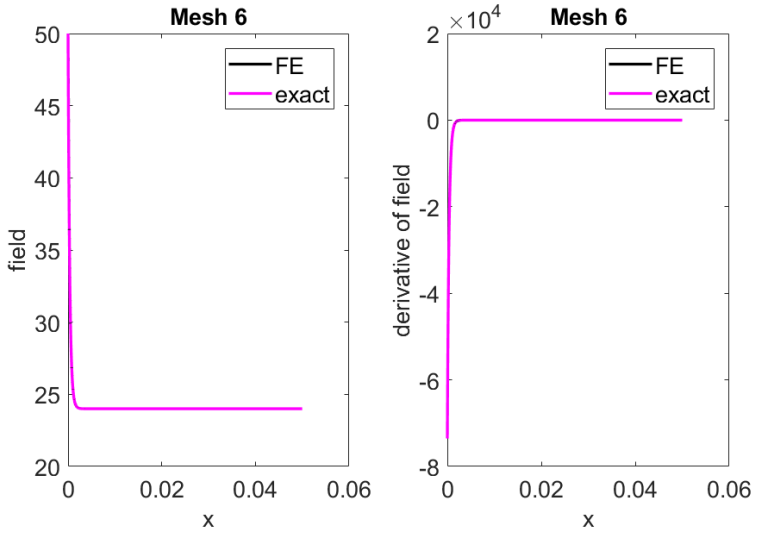


Figure 2.5. Temperature and derivative approximations compared to the exact solution of Model 2 after 6 uniform refinements.

Both the approximation as well as the derivative has converged to the exact solution. We can also confirm that our implementation has worked by checking errors which are graphed below in 4 different norms.

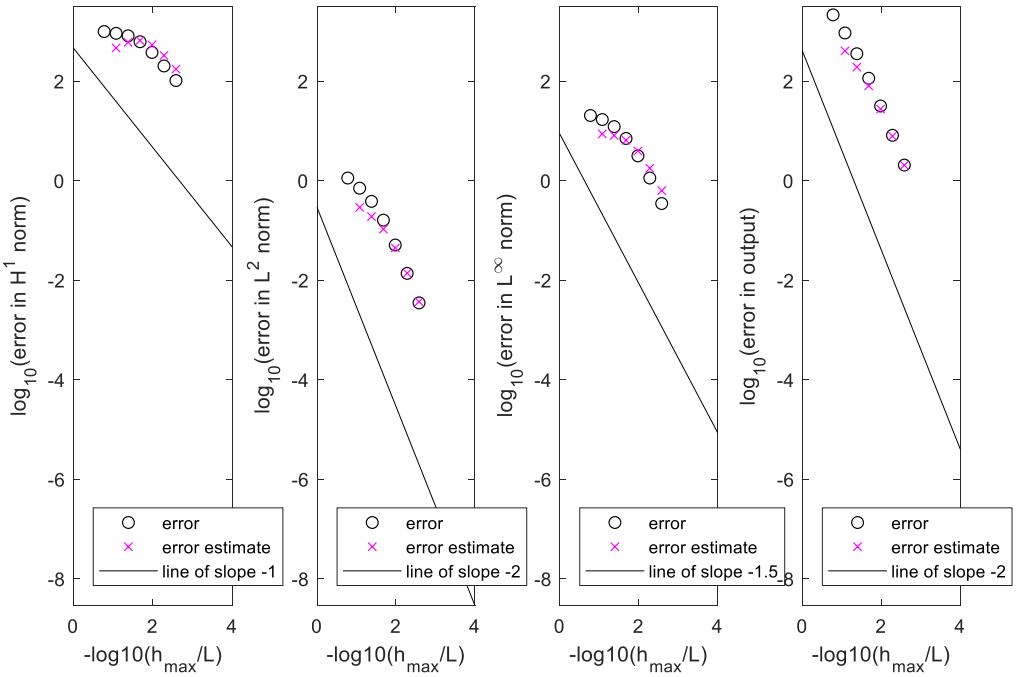


Figure 2.6. Error convergence plots in 4 norms of Model 2. The errors denoted by circles are calculated using the exact solution and the FE method approximation. The errors denoted by the pink "x"s are calculated using the previous mesh's approximation and the current mesh's approximation.

Error in the FE method can come from 3 sources: implementation, numerical specification, and mathematical modeling. If the implementation is correct, we expect to see the difference between the previous approximation and the current approximation to approach 0 as the mesh is refined.

$$\| u_h - u_h^{code} \| = 0 \text{ in infinite precision} \qquad (2.28)$$

$$\| u_h - u_h^{code} \| = 0^* \ (\epsilon_{prec}) \text{ in finite precision} \qquad (2.29)$$

Depending on the norm that we calculate the error in, the error will approach 0 as the mesh is refined at a specific slope.

Looking at the convergence plots, we see that the error due to implementation shown in pink "x"s does in fact decrease at the expected slopes for each norm which suggests our implementation is correct for Model 2. However, this is not enough to ensure that *form_elem_mat_sver* is bug-free. In Model 2, we had defined $\mu(x)$ as a constant. If we wanted to test the function more thoroughly, we could implement a model that satisfied the following equations:

$$-\frac{d}{dx}\left(\kappa(x)\frac{du}{dx}\right) + \mu(x)u = f_\Omega(x) \text{ in } \Omega = (0, \text{L}) \qquad (2.30)$$

$$u = u_{\Gamma_1} \text{ on } \Gamma_1 = \{0\} \qquad (2.31)$$

$$-\kappa(x)\frac{du}{dx} = \gamma_2 u - f_{\Gamma_2} \text{ on } \Gamma_2 = \{\text{L}\} \qquad (2.32)$$

where $\kappa(x), \mu(x), f_\Omega(x)$ all vary with *x*. This would allow us to be more confident that our mapping and formation of the elemental matrices were correctly implemented. If we were to create a model that satisfied these equations and the errors still converged with the correct slopes, we could be more confident that our implementation is correct.

Unfortunately, it would be difficult to solve for the exact solution of a model where the parameters all depend on *x*. Without the exact solution, we cannot calculate error by comparing the approximation with the exact solution. Instead, we can use a method called Manufactured Solutions to help calculate error and check if our approximation is accurate.

Manufactured Solutions

Manufactured Solutions is a way to verify implementation when the exact solution is unknown which is often the case in FE analysis.

We start with a set of equations similar to the ones above and take $\kappa(x), \mu(x), \gamma_1, \gamma_2, u_\infty$ as givens. From there, we choose a smooth solution as *u(x)* and solve $f_\Omega(x)$ using Equation 2.30 and f_{Γ_2} using Equation 2.32 such that:

$$f_\Omega(x) = -\frac{d}{dx}\left(\kappa(x)\frac{du}{dx}\right) + \mu(x)(u - u_\infty) \qquad (2.33)$$

$$f_{\Gamma_2} = \gamma_2 u + \kappa(x)\frac{du}{dx}\ (x = L) \qquad (2.34)$$

We then go back and solve for u_h using the Equations 2.33 and 2.34 and calculate the error in the norm $\| u - u_h \|$. We continue this re-iterating until we reach an error such that:

$$\| u - u_h \|_{H^1(\Omega)} \leq \epsilon_{tol} \qquad (2.35)$$

We would then graph these errors in their respective norms and expect to see convergence if implementation is correct.

impose_boundary_cond_sver

After running *form_elem_mat_sver*, we run *impose_boundary_cond_sver* to add the boundary conditions to matrices A and F. We will use Models 1 and 3 to check our implementation of this function.

Looking at Model 1, our initial mesh with 6 elements gives us the following approximation:

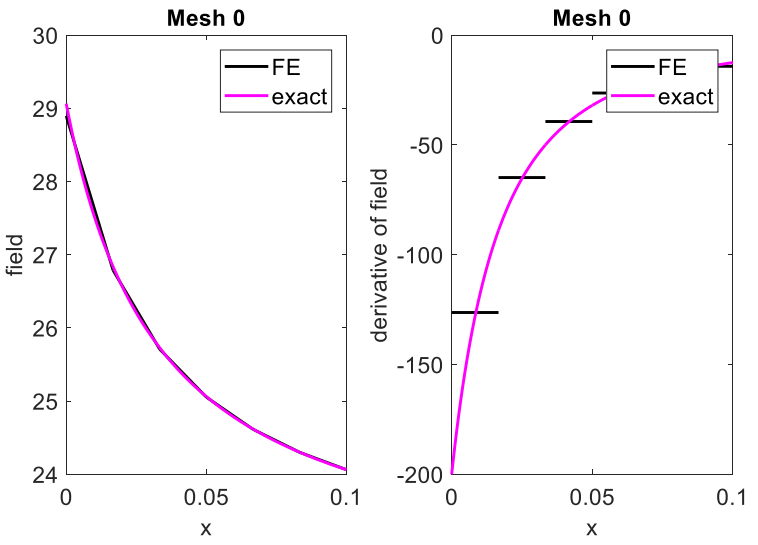


Figure 2.7. Temperature and derivative approximations compared to the exact solution of Model 1 using an initial mesh with 6 elements.

Unlike Model 2, we see that the initial mesh does a passable job approximating the exact solution. However, looking at the derivative plot, we see that the initial mesh does not do a good job of approximating the derivative and will require a more refined mesh.

Again, like with Model 2, we run 6 uniform refinements on the mesh and get the following approximations:

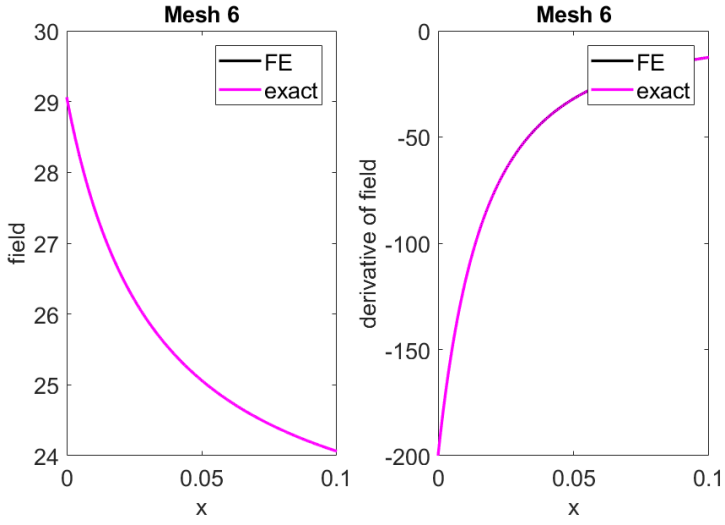


Figure 2.8. Temperature and derivative approximations compared to the exact solution of Model 1 after 6 uniform refinements on the initial mesh.

Now the approximations for both the temperature and the derivative of the temperature have converged much closer to the exact solution which is a good sign that our implementation could be correct. To further verify our implementation, we can look at the error convergence plots:

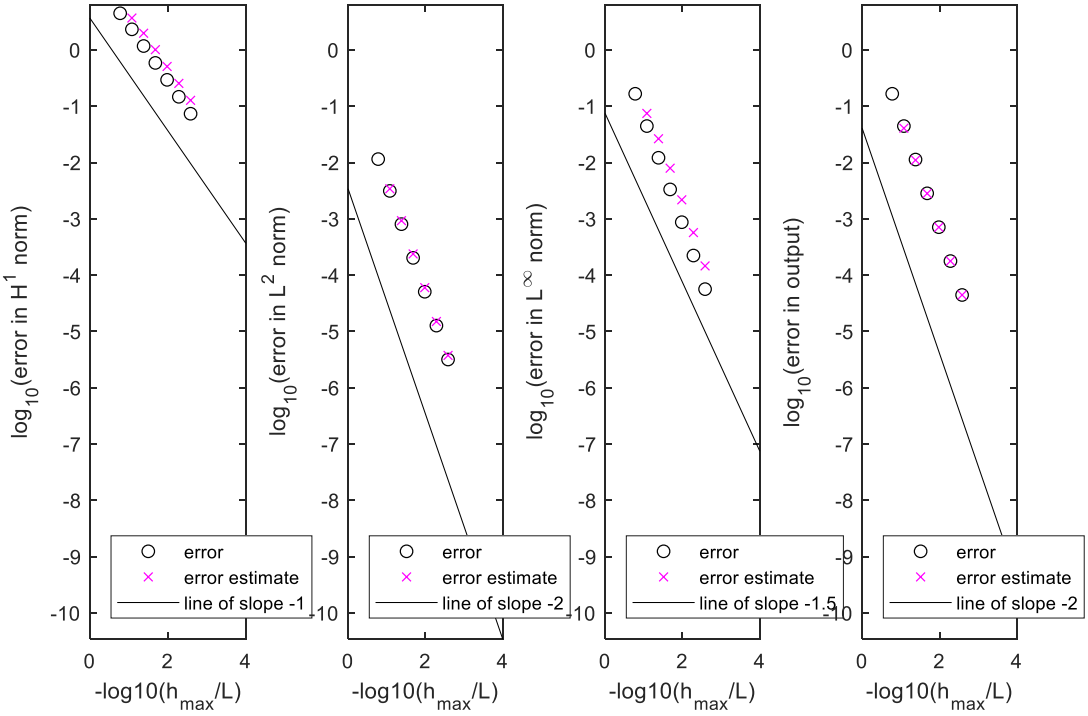


Figure 2.9. Error convergence plots in 4 norms of Model 1.

Like we saw in Model 2, the errors converge like we expect them to (following the slopes in each respective norm). This is sufficient evidence that our implementation of the FE method worked for Model 1. However, if we wanted to further test our implementation of *impose_boundary_cond_sver*, we could use a model that defines a heat flux at both the left

boundary and the right. We have defined Model 3 to have these boundary conditions (Equations 2.10 and 2.11).

Running the FE method on Model 3, the initial mesh with 6 elements gives us the following approximation:

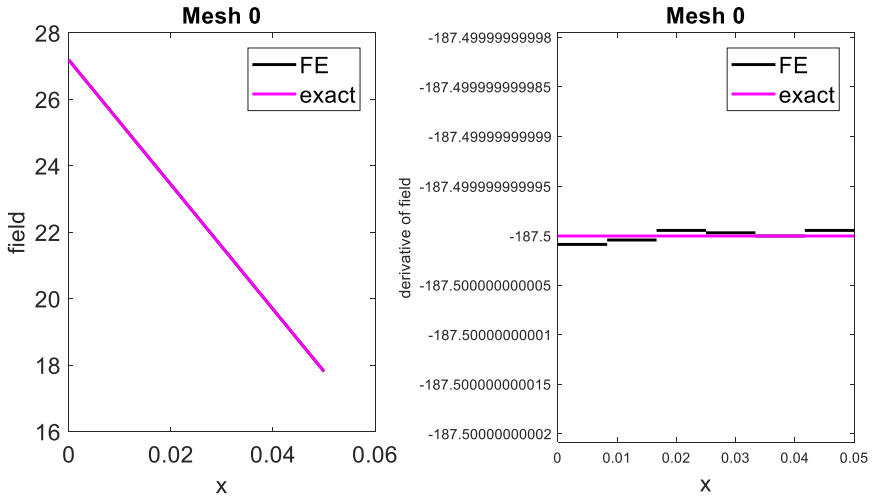


Figure 2.10. Temperature and derivative approximations compared to the exact solution of Model 3 using an initial mesh with 6 elements.

Again, we see that the approximation of the temperature converges faster than the derivative approximation. We uniformly refine the mesh 6 times and get the following approximations:

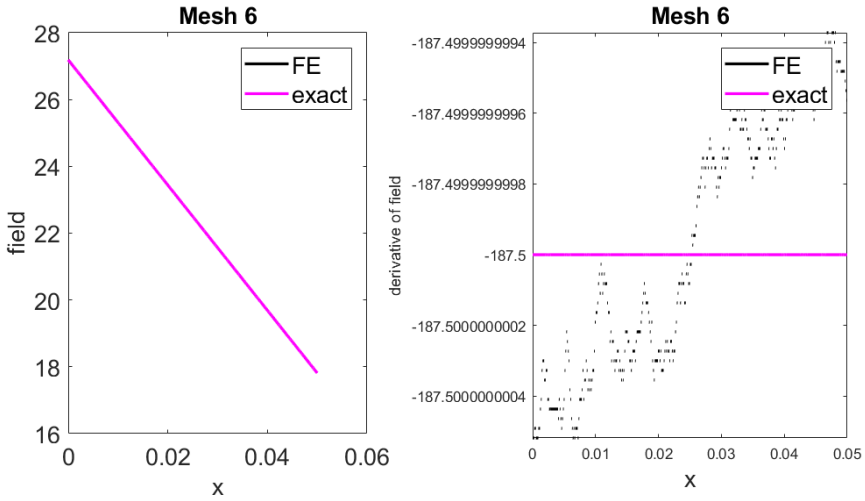


Figure 2.11. Temperature and derivative approximations compared to the exact solution of Model 3 after 6 uniform refinements on the initial mesh.

We see that the approximation converges to the exact solution. The derivative appears to not converge, but if we read the y-axis scale, we see that it does converge to the derivative of the exact solution. There appears to be error on the order of 10^{-10}, but this is not due to implementation, numerical specification, or mathematical model. This is simply due to the limited precision of the CPU. CPUs nowadays tend to have precision up until the 15^{th} decimal place (my laptop might have less because it's old). This finite precision also affects our convergence plots as seen below:

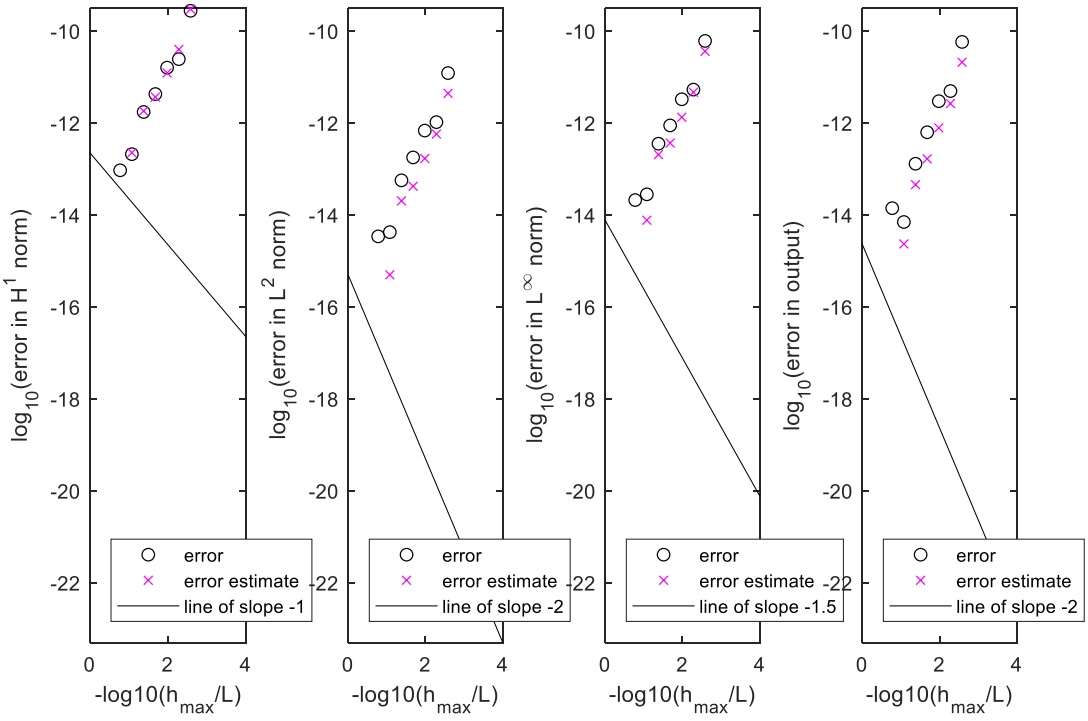


Figure 2.12. Error convergence plots in 4 norms of Model 3.

At first glance, it seems that our errors don't converge, suggesting an implementation error. However, this is not the case; we have simply hit the limit of precision which is around 10^{-12}. If we had a CPU with infinite precision, our error would in fact continue to converge to 0 (Equation 2.28).

If we wanted to continue testing our implementation of our FE method code, we could create more complex models and include parameters that depended on *x*. However, with Model 3 and the boundary conditions imposed in the model, we can be more confident that our implementation is correct than if we only tested with Model 1.

Convergence

As we discussed above, to be sure that our implementation was correct, we cannot just check to see if our approximation converged to the exact solution. To be confident, we checked the errors across 4 different norms to make sure they were converging at the correct rates. Similarly, we cannot just rely on the error convergence plots to know if our approximation is converging to the exact solution. As we discussed previous, there are 3 possible sources of error. We mainly looked into implementation errors, but there are still numerical specification errors and mathematical model errors. Numerical specification errors come from the precision of our mesh and of the parameters we are giving the FE method solver. For example, if a Model X had a η value of 92, but we rounded and gave the solver a η of 100, we would see that our Model doesn't converge to the exact solution. Additionally, if we model the system incorrectly, we could have a mathematical model error and our approximation would not converge to the exact solution either.

Looking more at our error norms and how mesh size affects convergence, we will use Model 2 and run the FE method code with 8 uniform refinements. Below is the approximation after all refinements as well as the convergence plots.

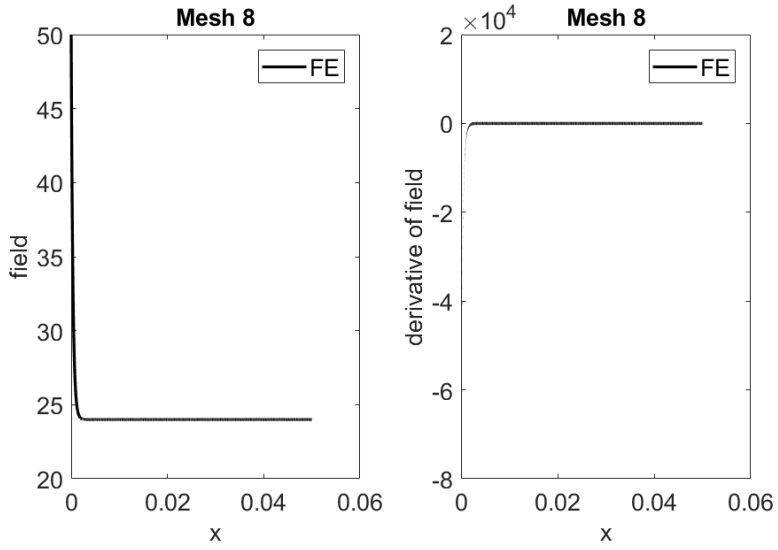


Figure 2.13. Temperature and derivative approximations of Model 2 after 8 uniform refinements on the initial mesh.

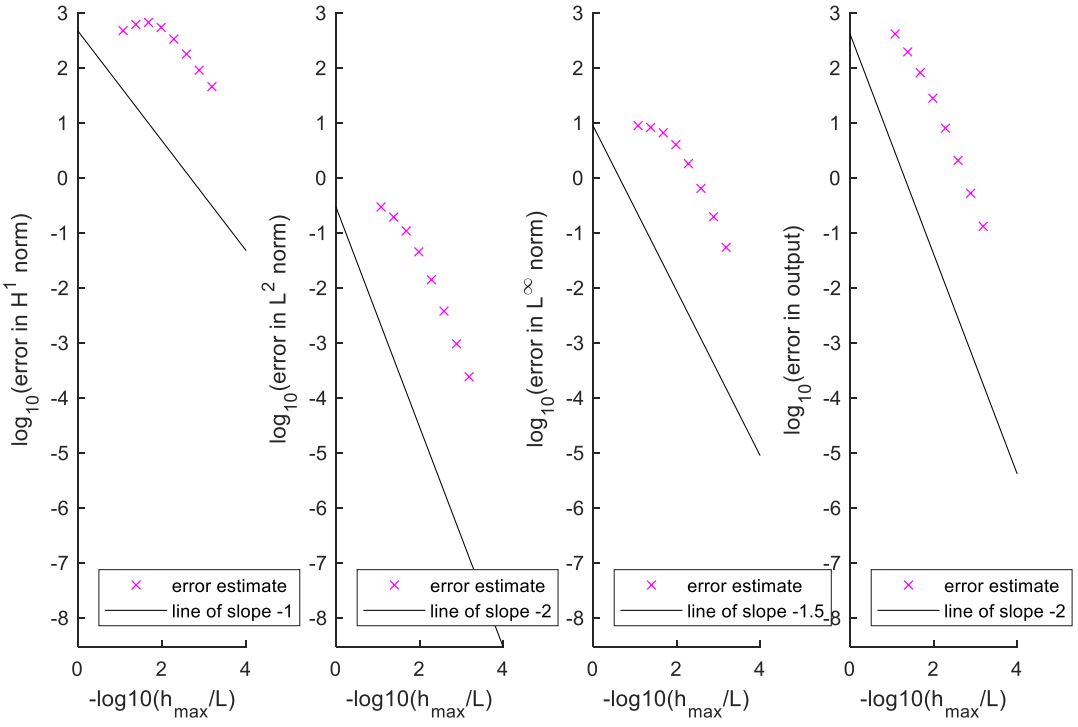


Figure 2.14. Error convergence plots in 4 norms of Model 2.

Using the convergence plot, we can calculate upper bounds for our outputs and errors. This can be useful if we are trying to get an idea of what temperature we can expect the physical system to be at. We can also calculate in safety factors to anticipate worst case outputs.

Let us consider the norm $\| u - u_h \|_{L^\infty(\Omega)}$ for Model 2 (the maximum error over all x in Ω). If we want our solution to have an error less than 2%, we can estimate that the coarsest mesh we would need to reduce error would be mesh 7 (the 6th refinement) where the log of the error in the norm is approximately -0.1967 or an error of 0.6358ºC. Depending on what system requirement we are designing to, we can place different upper bounds on $\| u - u_h \|_{L^\infty(\Omega)}$. If we want the maximum temperature different on our surface to be within 1ºC, then we could set the upper bound of this norm in mesh 7 to be 0. This would give us a safety factor of about 1.57 which would be a rather

strict requirement. Conversely, if we were not concerned about the range of temperatures on the surface, we could increase the upper bound on the norm which would allow more error.

We can also use the values from the convergence plots to set an upper bound on the error. For example, if we take mesh 5 in the error in output norm, we see that the $\log_{10} error\ in\ output$ is equal to 1.443 which corresponds to an error of of 27.7332ºC. Again, we can add a safety factor depending on our system requirements.

If we run the FE method to verify these upper bounds, we would expect the values to remain below the upper bounds.

Mesh 7 gives the following approximation. The greatest difference between the approximation temperatures and the exact solution temperatures does not exceed the upper bound we defined above.

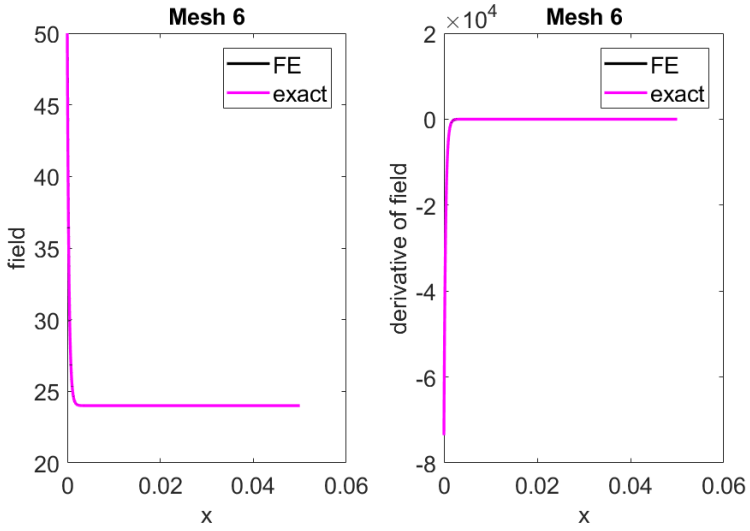


Figure 2.15. Temperature and derivative approximations of Model 2 after 6 uniform refinements on the initial mesh.

Mesh 5 gives the following approximation, and the output of the approximation does not exceed the upper bound defined above.

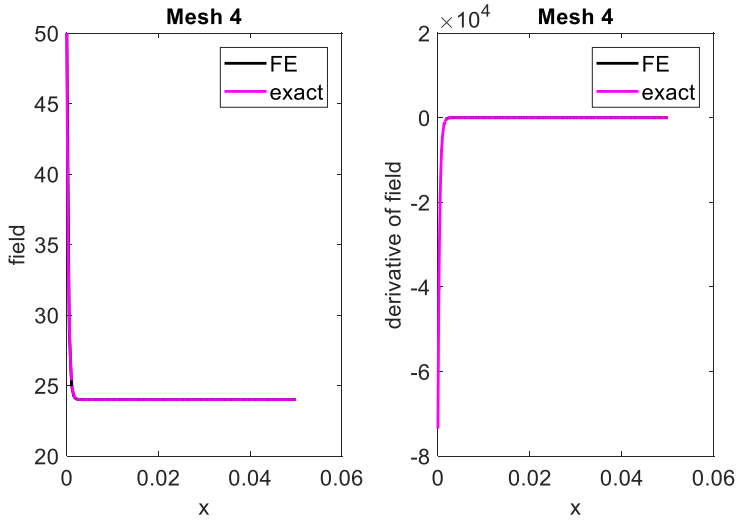


Figure 2.16. Temperature and derivative approximations of Model 2 after 4 uniform refinements on the initial mesh.

2.5 - Final Thoughts

The FE Method is a special case of the Rayleigh-Ritz Method and shows the importance of choosing basis functions. In Chapter 1, we saw how difficult it was to resolve sharp/steep features in the exact solution using the Rayleigh-Ritz Method and the basis functions we chose. However, in Chapter 2, using hat functions as our basis functions, we were able to approximate the exact solutions of both Model 1 and 2. Additionally, we saw how refining a mesh could increase the accuracy of our approximation.

In addition to seeing better approximations, we also explored an important debugging tool: convergence plots. By examining convergence plots, we can get a better idea of if our code was implemented correctly. When implemented correctly, the errors should converge at specific rates

for given norms. We also saw that even without the exact solution, we can still get an idea of error convergence using Manufactured Solutions.

Running the FE Method this chapter was pretty fast because of the implementation and use of sparse matrices. The FE Method runs on the order of O(n) which corresponds to the number of nodes in our mesh. While we only ran uniform meshes and refinements, given a more complicated model/geometry, we might consider varying the mesh across a domain. To maximize the efficiency of the FE Method, we could consider increasing the number of elements around areas of interest while decreasing elements in other areas. This would give us the resolution we need in our approximation while still keeping the runtime reasonable.

2.6 - References

1. Anthony T Patera, 2019, “FE 1D SPD BVP Formulation.”
2. Anthony T Patera, 2019, “FE 1D SPD BVP Theory.”
3. Anthony T Patera, 2019, “FE 1D SPD BVP Implementation (rev 2).”
4. Anthony T Patera, 2019, “FE 1D SPD BVP P2 Elements.”
5. Anthony T Patera, 2019, “Around Verification (Implementation, Accuracy).”

Chapter 3: The FD-FE Method for 1D Heat Equation: Flipping Burgers

3.1 - Abstract

Before this chapter, we had explored the FE Method for steady state heat transfer problems; we only looked at temperature as a function of space/position. However, in the real world, we may be concerned with heat transfer over time as well. In this chapter, we introduce and implement the Finite Difference Method (FD) to solve for temperature as a function of space and time.

To set up our implementation so that we can solve time-dependent heat transfer problems, we start by grouping the time-dependent term with the output terms in matrix F (from previous chapters). We can then apply our FE Method from Chapter 2 which will give us a matrix equation. From there, we bring out the time-dependent term and apply the FD Method to solve for the temperature in terms of time. In the FD Method, we use approximations to estimate temperature in terms of time. In this chapter, we primarily focus on using an Euler Backward approximation or a Crank-Nicholson approximation.

Applying our implementation of FE-FD to the problem of properly cooking a burger, we are able to see how numerical specifications affect our analysis. In the FE Method, when we run a higher order analysis ($p = 2$, φ functions are 2^{nd} order), we note that the temperature approximations converge to the exact solution quicker than the FE Method with $p = 1$. In the FD Method, we see that with a Crank-Nicholson approximation, we can better account for overestimation and underestimation than with an Euler Backward approximation which may favor overestimating or underestimating depending on the shape of the function. Because of this, running Crank-Nicholson FD Method helps converge the approximation quicker than Euler Backward. We proved this by comparing the computational times of a 1^{st} order FE Method, Euler Backward FD Method (theoretically the least efficient) with a 2^{nd} order FE Method, Crank-Nicholson FD Method (theoretically the most efficient).

By adding the FD Method to our code, we can analyze more complex problems. However, due to the added dimension of time, we see that optimizing the mesh size as well as the time step size becomes crucial for decreasing computation time. When we increase only one specification, we run the risk of increasing our run time without increasing the quality of our approximation. However, with the right numerical specifications, the FE-FD Method can be a very useful tool for analysis.

3.2 - Finite Difference and Finite Element

Previously, we had explored the implementation and analysis of the Finite Element Method in space for steady state problems (time-independent). In this chapter, we will be adding in a Finite Difference Method, so we can analyze time-dependent problems, specifically heat transfer over time.

Finite Element in Space

When we formulated the Finite Element Method for a Neumann-Robin boundary condition, we solved and implemented the method for a general form of a heat transfer equation

$$-\frac{\delta}{\delta x}\left(\kappa(x)\frac{\delta u}{\delta x}\right) + \mu(x)u = f_\Omega \text{ in } \Omega \qquad (3.1)$$

where *u* is a function of only *x*. Now, we are interested in finding a solution *u* that is a function of *x* and *t* (space and time) using a new heat transfer equation:

$$-\frac{\delta}{\delta x}\left(\kappa(x)\frac{\delta u}{\delta x}\right) + \mu(x)u = f_\Omega - \rho(x)\dot{u} \text{ in } \Omega,\ 0 < t \leq t_f \qquad (3.2)$$

Our boundary conditions remain the same on Γ_1 and Γ_2, but in the new heat transfer equation, we also include a third boundary condition stating the initial temperature when $t = 0$ such that

$$u = u_{ic}(x) \text{ in } \Omega,\ t = 0 \qquad (3.3)$$

To solve the heat transfer equation in space and time, we first apply similar principles from the Finite Element Method from Chapter 2 by grouping the time-dependent terms with the external driving term, f_Ω. Rewriting the heat transfer equation gives:

$$-\frac{\delta}{\delta x}\left(\kappa(x)\frac{\delta u}{\delta x}\right) + \mu(x)u = f_\Omega - \rho(x)\dot{u} = f_\Omega^+ \text{ in } \Omega,\ 0 < t \leq t_f \qquad (3.4)$$

which resembles the heat transfer equation from Chapter 2 (Equation 3.1). We can set up matrices such that:

$$\underline{A}\,\underline{u_h} = \underline{F^+} \qquad (3.5)$$

However, the matrix F^+ includes the time-dependent terms. Writing out the terms of F^+ gives:

$$\int_0^L f_\Omega^+ \varphi_i dx + f_{\Gamma_1}\varphi_i(0) + f_{\Gamma_2}\varphi_i(L) \qquad (3.6)$$

which, splitting up the integral term, can be written as

$$\int_0^L f_\Omega \varphi_i dx + f_{\Gamma_1}\varphi_i(0) + f_{\Gamma_2}\varphi_i(L) - \int_0^L \rho(x) u_h \dot{\varphi_i} dx \qquad (3.7)$$

where

$$\int_0^L \rho(x) u_h \dot{\varphi_i} dx = \int_0^L \rho(x)\varphi_i \sum_{j=1}^n \dot{u}_{hj}\,\varphi_j dx = \sum_{j=1}^n \int_0^L \rho(x)\,\varphi_i\varphi_j dx\,\dot{u}_{hj} \qquad (3.8)$$

To continue solving, we define a new matrix called the inertia matrix such that

$$\underline{M}^{inertia} = \int_0^L \rho(x)\varphi_i\varphi_j dx\ , 1 \leq i,j \leq n \qquad (3.9)$$

Plugging Equation 3.8 into Equation 3.7 and grouping time-independent terms into the matrix F, we rewrite matrix F^+ as

$$\underline{F}^{+} = \underline{F} - \underline{M}^{inertia}\underline{\dot{u}}_h \qquad (3.10)$$

Using Equation 3.10, we can also rewrite Equation 3.5 as shown below and create a system of n ODEs in time

$$\underline{M}^{inertia}\underline{\dot{u}}_h + \underline{A}\,\underline{u}_h = \underline{F}\ , 0 < t \leq t_f \qquad (3.11)$$

$$\underline{u}_h = \underline{(I_h u_{ic})}\ , t = 0 \qquad (3.12)$$

which we can solve by using the Finite Difference Method.

Finite Difference Method in Time

To solve for $u_h(x, t)$, we will iterate through time steps (n_{tsteps}) and approximate u. For the approximation, we can choose between Euler Forward (using the previous time step approximation), Euler Backward (using the current time step approximation), or Crank-Nicolson (using an average of the previous and current time step approximation). We will define a variable, θ, to denote which approximation we are using such that $\theta = 0$ is Euler Forward, $\theta = 0.5$ is Crank-Nicholson, and $\theta = 1$ is Euler Backward.

We can set up Equation 3.11 using approximations for u_h and k as our iteration count (point in time) as follows:

$$\underline{u}^{k}_{h,\Delta t} = \underline{(I_h u_{ic})}\ , k = 1 \qquad (3.13)$$

$$\underline{M}^{inertia}\frac{\underline{u}^{k}_{h,\Delta t} - \underline{u}^{k-1}_{h,\Delta t}}{\Delta t} + \underline{A}\left(\theta\underline{u}^{k}_{h,\Delta t} + (1-\theta)\underline{u}^{k-1}_{h,\Delta t}\right) = \underline{F}\ , 2 \leq k \leq n_{tsteps} \qquad (3.14)$$

In implementation, there are different ways to store the previous u approximations. For this chapter's implementation of Finite Difference Method, we will be storing all previous values in an n_{node} x n_{tsteps} array rather than overwriting previous values (saving only the previous value).

Implementation and Verification

In this chapter, we have added the implementation of the Finite Difference Method to our Chapter 2 implementation of the Finite Element Method. To test our implementation, we ran a uniform refinement on a new model, *semiinf_plus*, with three different sets of parameters. The first run was using linear φ equations ($p = 1$) for FE and Euler Backward approximation ($\theta = 1$) for FD. The second run was using 2nd-order quadratic equations ($p = 2$) for FE and Euler Backward approximation ($\theta = 1$) for FD. Finally, the third run was using 2nd-order quadratic equations ($p = 2$) for FE and Crank-Nicholson approximation ($\theta = 0.5$) for FD.

Running the uniform refinements, we see that the approximated solutions, u_h, converge to the exact solution provided in the model. Additionally, the approximated solutions reflect our choices of p and θ. We expect a higher p value will initially give us a better approximation of the exact solution as well as the exact solution's derivative because our φ functions will have higher order and be able to better capture the complexity of a curve. With the choice of approximation used for FD, we expect a Crank-Nicholson approximation to generate better results than an Euler Backward or Forward. With an Euler Backward or Forward, depending on the shape of the curve, the approximation will favor either overestimating or underestimating. For example, if we had a concave curve and used Euler Backward, we would always be underestimating.

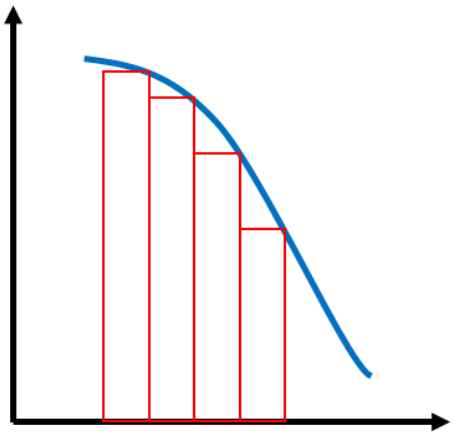


Figure 3.1. Euler Backward on concave curve. Due to the choice of approximation, the approximated value will be an underestimate the exact value.

With a Crank-Nicholson approximation, we use an average between the previous and current approximation values which helps diminish overestimating/underestimating. If we use the previous example, we see that on the left half of a given rectangle, we are underestimating; however, this underestimating is canceled out by the overestimation in the right half of the rectangle.

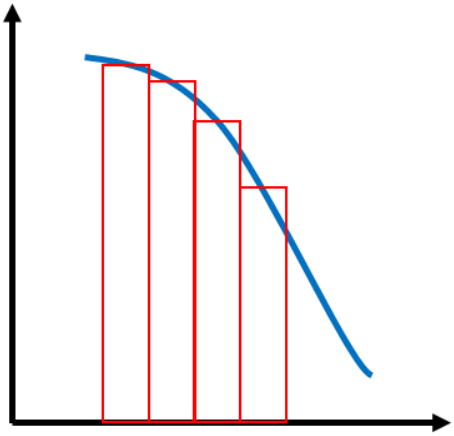


Figure 3.2. Crank-Nicholson approximation on concave curve. The underestimation in the left of the rectangles is decreased by the overestimation in the right of the rectangles.

Based on the parameters we picked, we expect that initially, Run 1 will have the worst approximation between the three runs while Run 3 should have the best (closest to the exact solution).

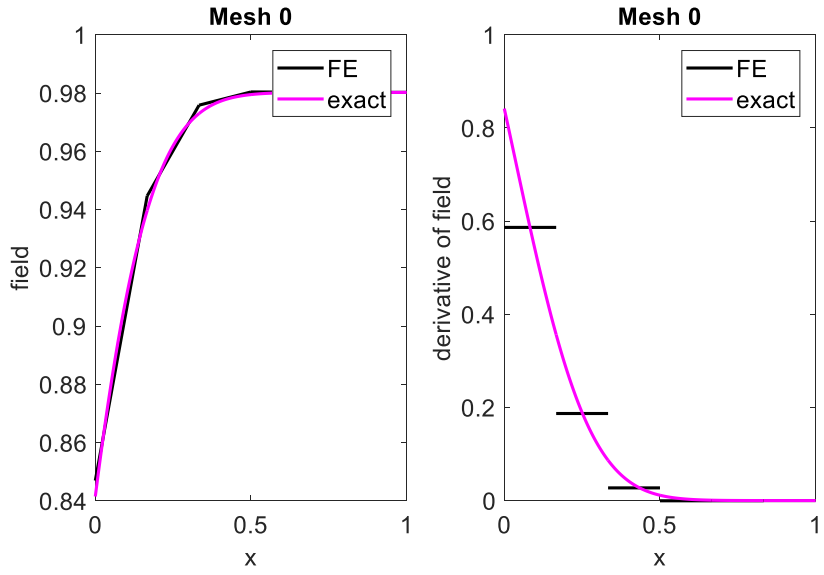


Figure 3.3. Approximated solution and derivative (black) for Mesh 0 of Run 1 graphed with the exact solution (pink).

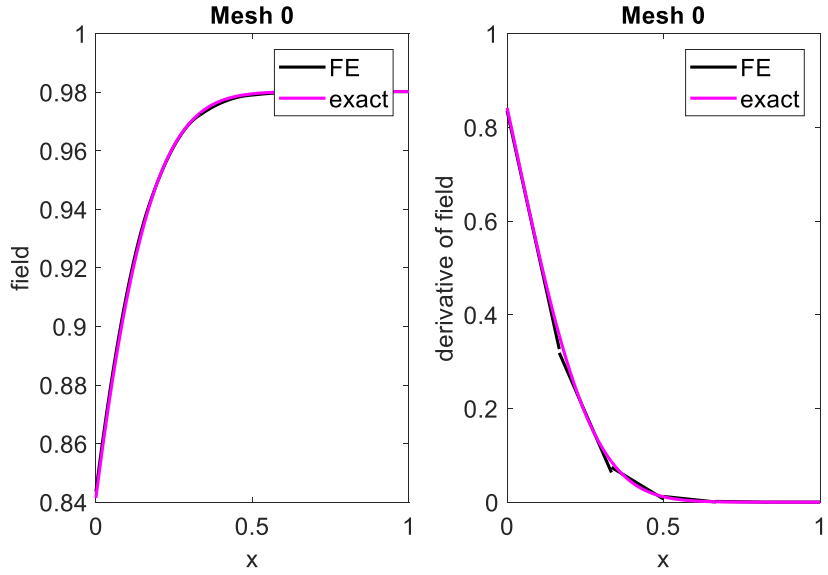


Figure 3.4. Approximated solution and derivative (black) for Mesh 0 of Run 2 graphed with the exact solution (pink).

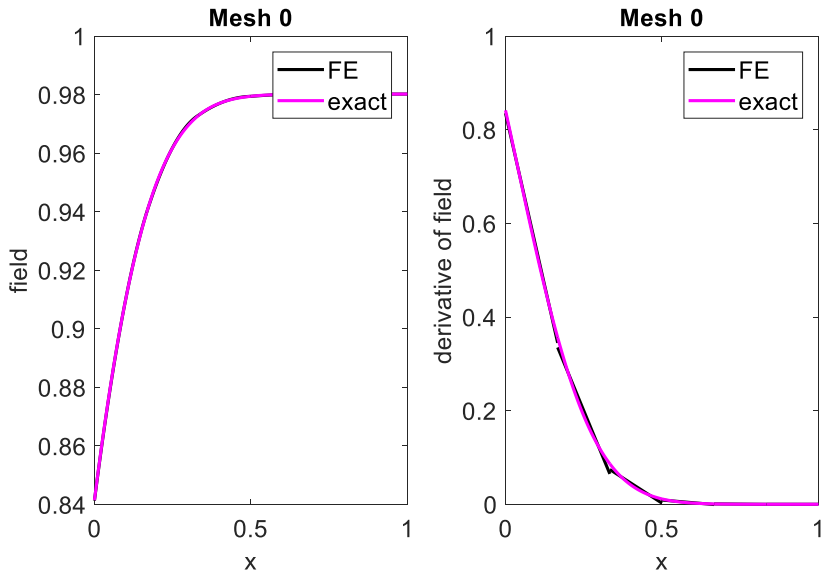


Figure 3.5. Approximated solution and derivative (black) for Mesh 0 of Run 3 graphed with the exact solution (pink).

We see that Run 1's initial mesh gave an approximation that deviates the most from the exact (although it roughly follows the exact solution). Between Run 2 and Run 3, the difference between approximations isn't very large, but Run 3 appears to be a slightly better approximation. Similar observations can be see with the derivatives. Run 1 is limited to using constant piecewise functions to approximate the derivative, so the approximation is poor compared to Run 2 and 3.

We can further verify the implementation of our FD Method by checking the convergence graphs. Now that we are considering both space and time, the quality of our analysis is dependent on mesh size, h, (Chapter 2) as well as time step size, Δt. Ideally, we wish to decrease and converge h and Δt at similar or equal rates, so our error is balanced and not dominated by one of the sizes. We have graphed error over 4 different norms as we did in Chapter 2 below.

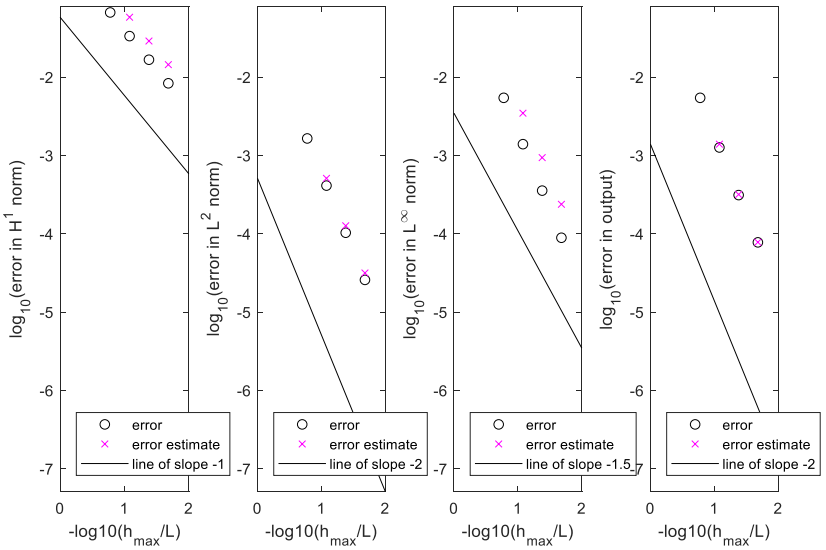


Figure 3.6. Error convergence plots of Run 1.

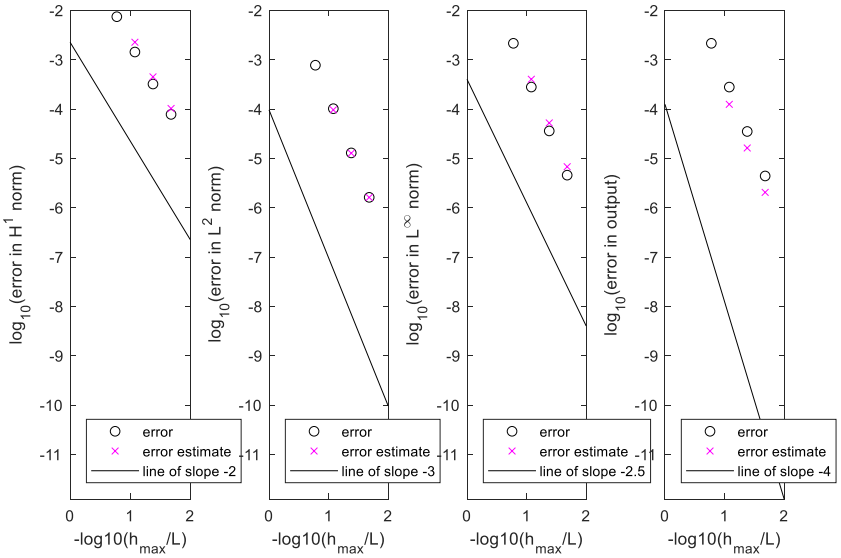


Figure 3.7. Error convergence plots of Run 2.

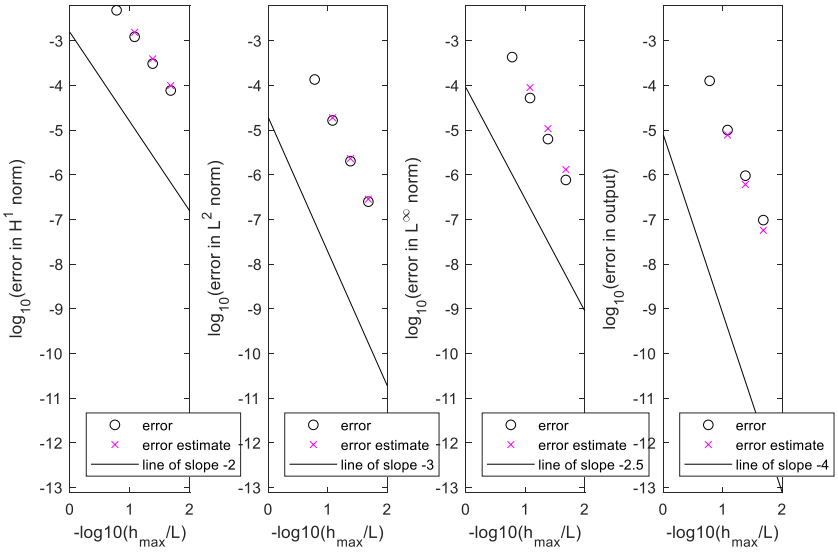


Figure 3.8. Error convergence plots of Run 3.

We can see that the errors across the 4 norms are all converging which is good. Checking that the errors converge at the correct rates gives us more confidence that our implementation is correct. To do so, we can write the error as follows:

$$\| u(\cdot, t_f) - u_h^{n_{tsteps}} \|_Q^{(l)} \sim C_{u,Q}^1 \sigma^{-ql} + C_{u,Q}^2 2^{-rl} \; as \; l \to \infty \quad (3.15)$$

or

$$\| u(\cdot, t_f) - u_h^{n_{tsteps}} \|_Q^{(l)} \sim 2^{-rl} (C_{u,Q}^1 \left(\frac{2^r}{\sigma^q}\right)^l + C_{u,Q}^2) \; as \; l \to \infty \qquad (3.16)$$

where Q is the norm of interest and l is the level at which we are calculating the error.

To determine the value of $\left(\frac{2^r}{\sigma^q}\right)$, we can create a table based on the p and θ we used for our FD-FE. To achieve optimal balance between time and space, we would like the value to equal 1. If we consider the $H^1(\Omega)$ norm which is sub-optimal in time (values less than 1), the table is as follows:

Table 3.1. $\left(\frac{2^r}{\sigma^q}\right)$ for Q = $H^1(\Omega)$

	r = 1 **p = 1**	**r = 2** **p = 2**
q = 1 **θ = 1**	σ = 4 0.5	σ = 8 0.5
q = 2 **θ = 2**	σ = 2 0.5	$\sigma = 2\sqrt{2}$ 0.5

Looking at the L^2 norm (the 2nd subplot in Figures 3.6, 3.7, and 3.8) which is optimal, our table is as follows:

Table 3.2. $\left(\frac{2^r}{\sigma^q}\right)$ for Q = $L^2(\Omega)$

	r = 2 **p = 1**	**r = 3** **p = 2**
q = 1 **θ = 1**	σ = 4 1	σ = 8 1
q = 2 **θ = 2**	σ = 2 1	$\sigma = 2\sqrt{2}$ 1

Using the table values and combining $C_{u,Q}^1$ and $C_{u,Q}^2$ as $C_{u,Q}$, we can rewrite Equation 3.16 for the L^2 norm as

$$\| u(\cdot, t_f) - u_h^{n_{tsteps}} \|_{L^2(\Omega)}^{(l)} \sim 2^{-rl} C_{u,Q} \; as \; l \to \infty \qquad (3.17)$$

To get Equation 3.17 in terms of *h*, we can use $h^{(l)} = 2^{-l} h_0$ to substitute 2^{-l} with mesh size terms and get the following

$$\| u(\cdot, t_f) - u_h^{n_{tsteps}} \|_{L^2(\Omega)}^{(l)} \sim \left(\frac{h^{(l)}}{h_0}\right)^r C_{u,Q} \qquad (3.18)$$

By taking the log of both sides, we can get a linear equation where the slope is $-r$, the convergence rate. Looking back at the table, we expect Run 1 to converge at a rate of -2 and Run 2 and 3 to converge at a rate of -3 in the L^2 which is what we see in Figures 3.6, 3.7, and 3.8. Although the other 3 norms are not optimally balanced like L^2, doing similar calculations with the other 3 norms to determine the convergence rates confirms that our implementation is correct.

3.3 - Burger Model

For this Chapter and to further test our FD-FE implementation, we have created a model of cooking a burger. We will be modeling the burger as a semi-infinite fin and assuming that our skillet is able to maintain a constant temperature gradient throughout the entire process. We have split the cooking into 3 stages: pre-flip, post-flip, and repose. We specified the times for each stage based on Bobby Flay's suggested times and graphed the temperature approximation through the burger along with constraints we've set based on safety (Is the temperature in the burger above the cooked temperature T_{done}?) and taste (Is the temperature in the burger above the Maillard temperature $T_{Maillard}$? Has the burger rested for enough time t_{repose}?).

Results

In the graph below, we see the temperatures of the burger faces along with the temperature in the middle of the burger as a function of time in seconds.

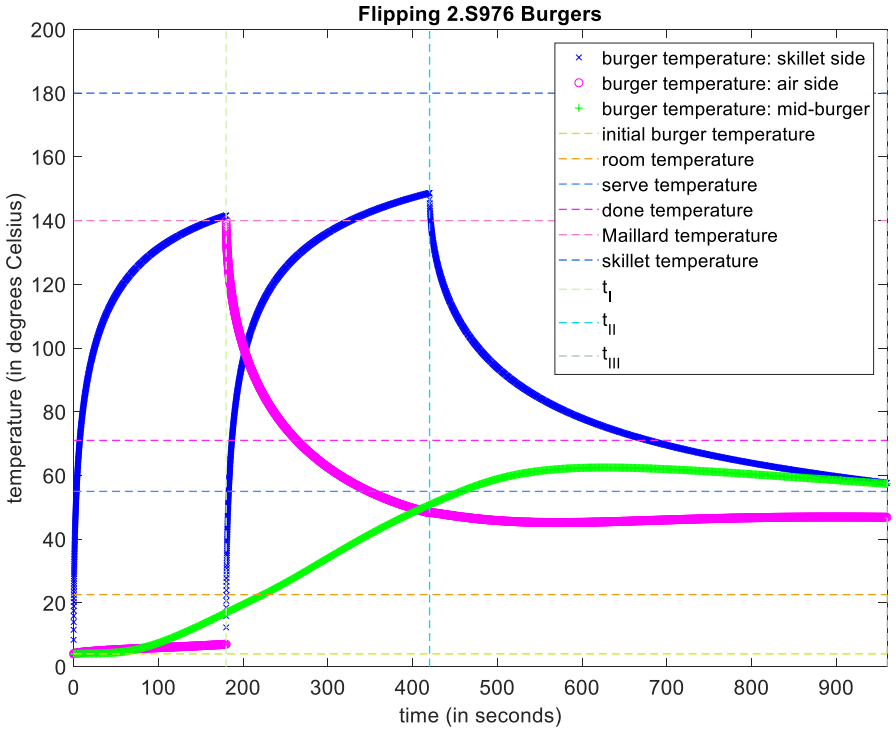


Figure 3.9. Temperature from FD-FE (p = 1, θ = 1) graphed as a function of time. At *t = 180s*, the burger is flipped. The burger is removed from the skillet at *t = 420s* and reposes for the remainder of the time.

We also graph the temperature distribution in the Burger at the end of the repose stage.

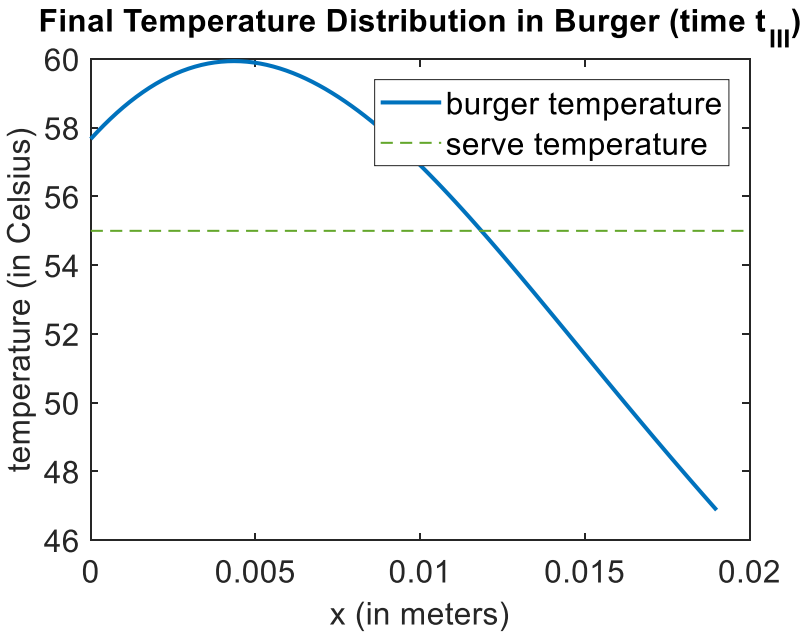


Figure 3.10. Temperature distribution within the burger at the end of the repose stage (blue). The temperature at which the burger should be served is shown as well (green dashed).

Implementation Verification

Unlike Models in the previous chapters, we do not have an exact solution for our burger model. To check that our implementation is correct, we can directly compare our results with implementation that we believe to be correct. In this case, we will use the code that Prof. Patera has written. When we use the same parameters and values, Prof. Patera's implementation gives the following results:

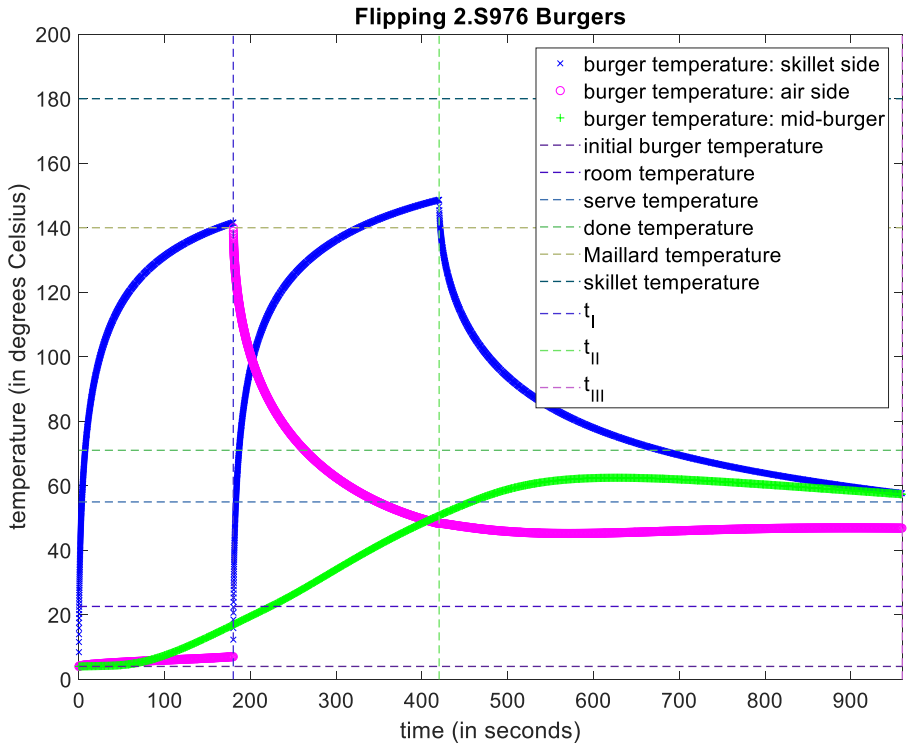


Figure 3.11. Temperature from Prof. Patera's FD-FE graphed as a function of time.

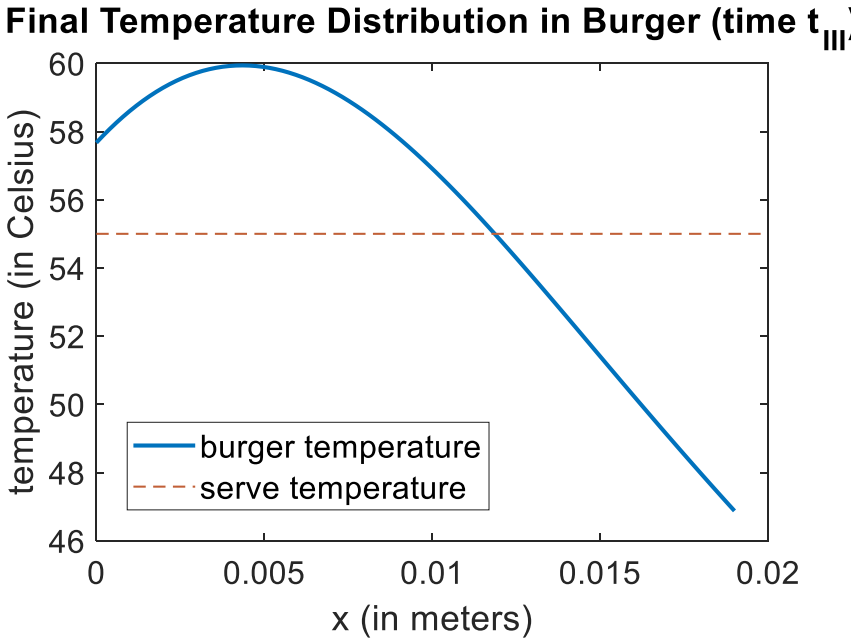


Figure 3.12. Temperature distribution in burger at the end of repose stage given by Prof. Patera's FD-FE.

Looking at the graphs, it seems the two FD-FE implementations yielded the same results. Upon further inspection, the numerical values at t_I, t_{II}, and t_{III} also match as shown in Table 3.3. If we take Prof. Patera's FD-FE implementation to be correct, then we can verify that our implementation is also correct.

Table 3.3. Temperature values at specific time stamps from Ch. 3 FD-FE and Prof. Patera's FD-FE.

	Skillet Side Temp (°C)		Air Side Temp (°C)		Mid-Burger Temp (°C)	
Time (s)	Ch. 3 FD-FE	Prof. Patera	Ch. 3 FD-FE	Prof. Patera	Ch. 3 FD-FE	Prof. Patera
0	4	4	4	4	4	4
180	141.6	141.6	7.022	7.022	16.93	16.93
420	148.7	148.7	48.27	48.27	50.79	50.79
960	57.67	57.67	46.87	46.87	57.37	57.37

Numerical Specification Verification

Next, we wish to verify our numerical specifications (mesh size, time step). We will do so by comparing performance when we switch p and θ between 2 cases. Case 1 will have [p = 1 and θ = 1], and case 2 will have [p = 2 and θ = 0.5].

First, we find the coarsest FE mesh that will get us within the error range that we want. In this case, we will look at the temperature of the burger on the skillet side at time t_I before the flip. If we take the reference value of the burger to be 141.6019ºC (found by running refinements using case 2 until output converged to a single value), we would like the error to be within 0.001ºC of that value. The temperature values along with the error are shown in Table 3.4 below.

Table 3.4. Temperature of the burger on the skillet side at time t_I pre-flip along with the difference from 141.6019ºC, the accepted value of the temperature of the burger.

[p = 1, θ = 1]			[p = 2, θ = 0.5]		
# Of Refinements	Temp. (ºC)	Temp. Diff (ºC)	# Of Refinements	Temp. (ºC)	Temp. Diff (ºC)
1	141.5101	0.0918	1	141.6030	0.0011
2	141.5788	0.0231	2	141.6020	0.0001
3	141.5961	0.0058	3	--	--
4	141.6004	0.0015	4	--	--
5	141.6015	0.0004	5	--	--

For the specifications of case 1, the coarsest FE mesh we need to have an error less than 0.001ºC is 6 (5 refinements to the initial mesh). On the other hand, with case 2 specifications, the coarsest FE mesh we need is 3 (2 refinements to the initial mesh) suggesting that case 2 is a more efficient approach computationally. We can use a coarser mesh and come closer to the accepted value quicker than with case 1 parameters.

To be more exact about how much more efficient case 2 is, we can calculate the ratio of computational time between the two cases. To calculate the computational time, we can use the following:

$$\Delta_Q^{(l+1)} = \frac{\|u^{(l+1)} - u^{(l)}\|}{2^r - 1} = \frac{\|n_{steps} - n_{tsteps}\|}{2^r - 1} \qquad (3.19)$$

We can obtain *r* from tables like Table 3.1 and 3.2. For Q = output, the table is shown below. Note that case 2 is sub-optimal in space while case 1 is optimally balanced.

Table 3.5. $\left(\frac{2^r}{\sigma^q}\right)$ for Q = output.

	r = 2 **p = 1**	**r = 4** **p = 2**
q = 1 **θ = 1**	σ = 4 1	σ = 8 2
q = 2 **θ = 2**	σ = 2 1	$\sigma = 2\sqrt{2}$ 2

Calculating the computational times of both cases gives the follow:

Table 3.6. Computational times of Case 1 and 2.

[p = 1, θ = 1], 5 refinements				[p = 2, θ = 0.5], 2 refinements			
n_el	n_tsteps	r	Comp. Time	n_el	n_tsteps	r	Comp. Time
192	20	2	57.33	24	20.0000	4	0.53333

From this, we find that the ratio of computational time of case 1 relative to case 2 is 107.6 meaning case 1 is much less efficient than case 2.

This makes sense because we expect that at higher orders (higher *p*), our analysis should give us a better approximation faster because of the φ functions we are using. Additionally, we expect the Crank-Nicholson approximation (θ = 0.5) to give us a better approximation than Euler Backward. Based on our observations, this seems to be the case which means our numerical specifications seem to be implemented correctly.

Physical Model Verification

The final part of our model that we want to verify is our physical model. Our numerical specifications and implementation could be correct, but if we have a bad heat transfer model to begin with, our analysis will not reflect what we would see in the real world.

Our model parameters, specifically the flip and cook times, were taken from Bobby Flay while the repose time was determined by averaging multiple suggested times. The burger geometry was based off a typical burger size.

I like sliders more than burgers because they are smaller which means I can try different types of sliders without getting too full too quickly.

We will use a new diameter of 0.065 meters while maintaining the same thickness of 0.019 meters for our slider parameters. Looking at a recipe online, rather than cooking the burgers at medium-high, the recipe cooks at medium, so we will lower the skillet temperature from 180ºC to 155ºC while keeping the other temperatures the same. For cook times, the recipe recommends 2-3 minutes for stage 1 (pre-flip), 2-3 minutes for medium rare stage 2 (post-flip) or 3-4 minutes for medium stage 2. We will keep the repose time the same as before. In general, it seems the recipe has lower times for stage 1 and 2. We will use 150 seconds for stage 1 and 180 seconds for stage 2.

The results from our FD-FE are shown below:

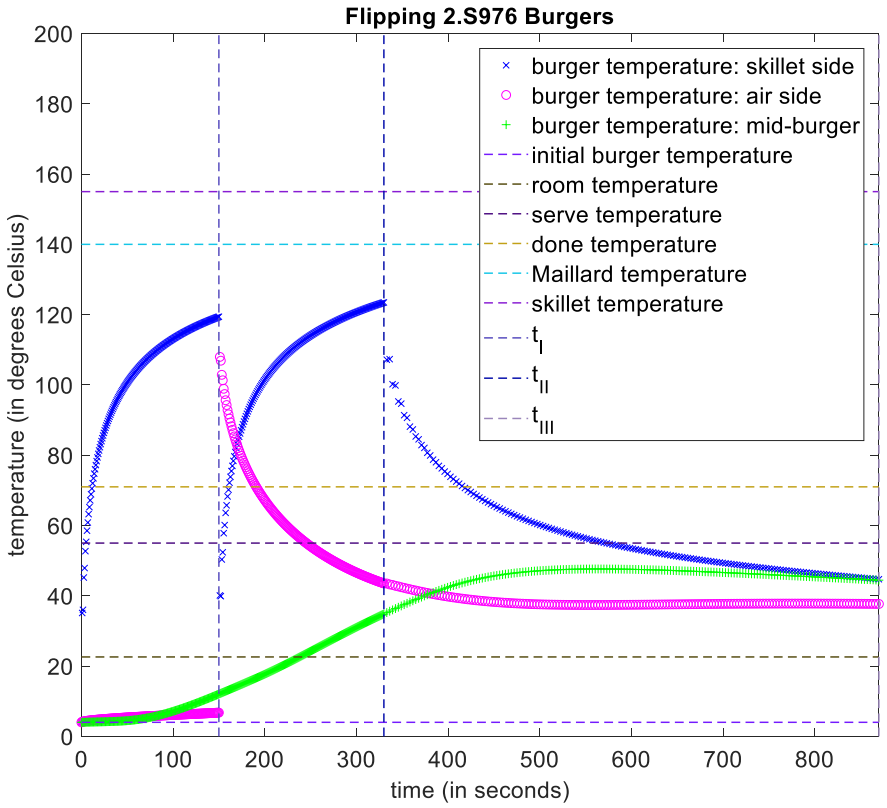


Figure 3.13. Temperature from FD-FE (p = 2, θ = 0.5) graphed as a function of time. At *t = 150s*, the burger is flipped. The burger is removed from the skillet at *t = 330s* and reposes for the remainder of the time. Skillet temperature is 155ºC

If our approximation is correct, it seems like these sliders would never reach the Maillard temperature which would affect the taste. Additionally, the sliders do not appear to actually reach a safe amount of cooked. We expect that at some point during stage 2, the temperature inside the burger should reach above T_{done}.

Let us modify this slider recipe by increasing the skillet temperature back to what we had before (180ºC).

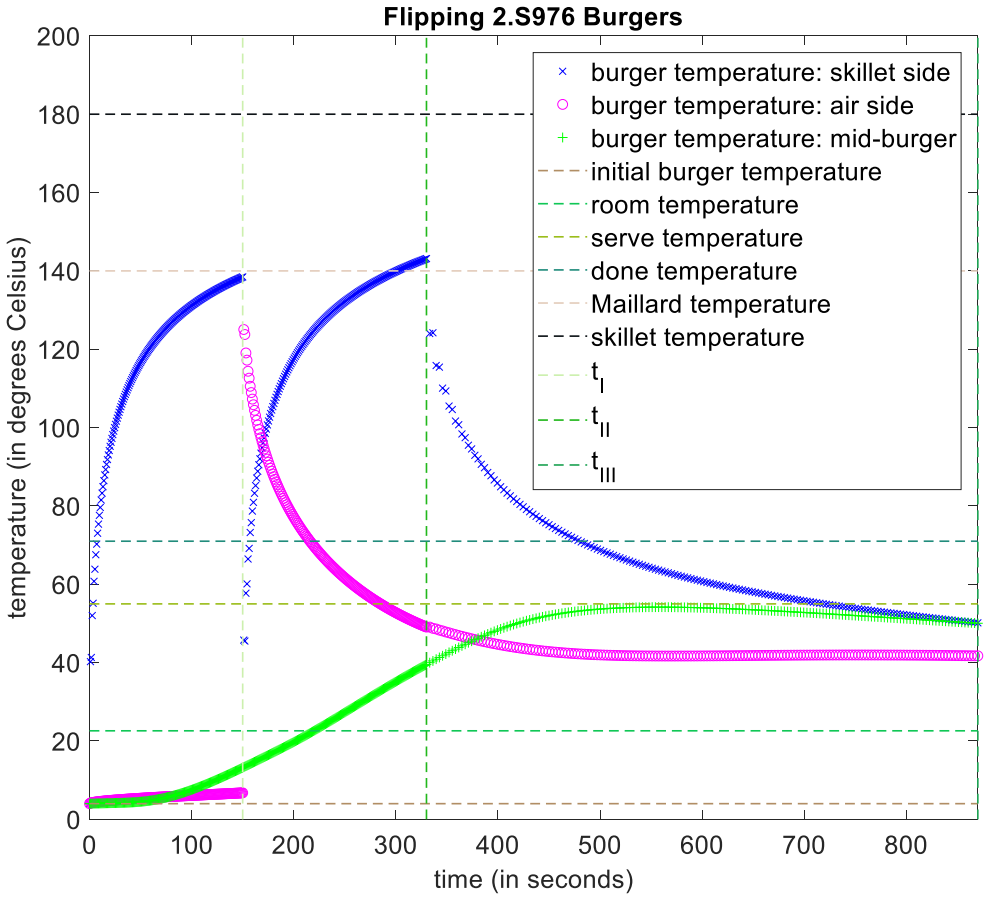


Figure 3.13. Temperature from FD-FE (p = 2, θ = 0.5) graphed as a function of time. Skillet temperature is 180ºC.

Now our sliders are able to reach the Maillard temperature which is good because we want our sliders to taste good.

To bring our internal burger temperature (mid-burger) closer to T_{done}, we can try increasing the cook times and decreasing the thickness of our sliders. I've increased the cook times to what Bobby Flay recommends for regular burgers and decreased the thickness of the sliders from 0.019m to 0.016m.

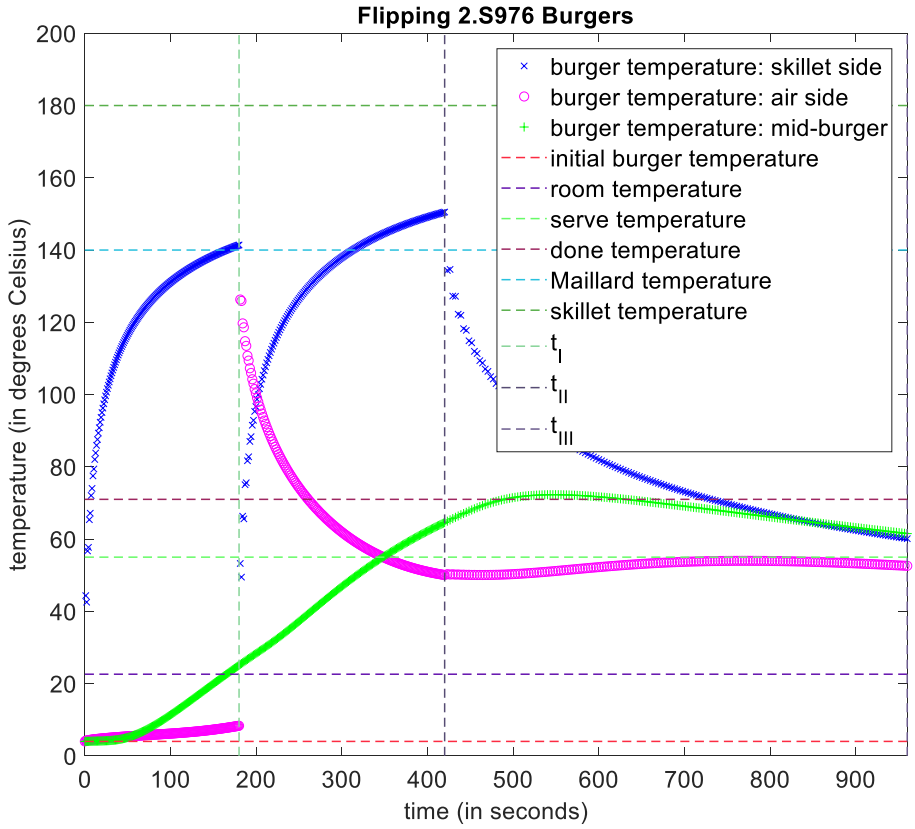


Figure 3.14. Temperature from FD-FE ($p = 2$, $\theta = 0.5$) graphed as a function of time. Skillet temperature is 180ºC. Cook times increased and slider thickness decreased.

Now our sliders satisfy our safety constraint as the mid-burger temperature is able to reach the done temperature.

In practice, there may be other factors contributing the how the burgers cook. One thing the recipe never specified was the starting temperature of the burgers. We assumed we would be taking the burgers out of the fridge, meaning the starting temperature was low. However, if the starting temperature was closer to room temperature, the cooking parameters may be different.

3.5 - Final Thoughts

By adding the FD Method, we are able to handle much more complex and interesting problems with our FE-FD code.

Once we verify that the code is implemented correctly and that we are using optimized numerical specifications, FE-FD is able to provide accurate approximations with a small computational cost. However, when our numerical specifications are not optimal, we see how the analysis is affected. As we saw, when our analysis is run on lower-order φ functions and Euler Backward approximations, we require a finer mesh to yield an accurate solution. If we use higher-order FE and Crank-Nicholson approximations in our FD, we do not need to refine the mesh as much, cutting down on computation time.

We do need to be careful about our numerical specifications. While it may be tempting to simply decrease the mesh size or time step size, maintaining a balance between the two specifications is important to optimize the analysis. Increasing one specification faster than the other may lead to a longer computation time but a less accurate solution. Additionally, we must be careful about applying FE-FD when we can. If we use it on a situation that it is not designed to handle, our solution will not reflect what could happen in the real world. In other words, our physical model is just as important as the implementation of the FE-FD.

3.6 - References

1. Anthony T Patera, 2019, “The Heat Equation: Formulation and Implementation.”
2. Anthony T Patera, 2019, “The Heat Equation: Error Estimation.”
3. Anthony T Patera, 2019, “The Heat Equation Study Cases.”
4. Izzy Moon, “How to Cook Sliders on a Skillet,” Live Strong.
5. Maegan, 2017, “Skillet Sliders,” The Baker Mama.

Chapter 4: The FE Method for 1D 4^{th}-Order BVPs: Xylophone

4.1 - Abstract

In previous chapters, we focused on solving heat transfer problems using FE methods. In this chapter, we will modify our FE method such that we can solve beam bending problems.

We will use the Euler-Bernoulli beam model as our general model which we can derive using equilibrium equations and constitutive stress-strain relationships. Once we have our general differential equation, we can set up matrices similar to previous chapters and solve for a set of eigenvalues that we can use to create our solution which includes both displacement as well as velocity (space and time). For the chapter's xylophone problem, we will be assuming the ends of our bars are free. However, in the case that we do want to constrain the beam, we can go in the FE method after solving for our matrices and add in our boundary conditions. Aside from our FE method, we also added a segment of code to determine the zeros of the 3rd mode of the frequency response of the beam. This corresponds to nodes where displacement is zero which will be where we pick our xylophone's hole locations.

To design our xylophone bars, we first determine an optimal height function (based off a desired fundamental frequency or pitch and a frequency ratio) which will correspond to a cutout in the bar. Once we have determined the optimal height function, we can determine the optimal bar length as well as hole locations. For this chapter, we looked at the notes F4, C5, and F5.

We analyzed our results as well as error estimates to determine if the designs were reasonable. While the error convergence plots did not reveal any computational errors, looking at the error estimates revealed that there may have been numerical errors for C5 and F5 designs. We may have been reaching the limits of what the model is valid for, proving that creating a good model is just as important as correctly implementing the FE method.

4.2 - Finite Element Method

In this chapter, we modify our Finite Element Method to solve beam bending eigenproblems in space and time. We will derive the general method from the following beam bending problem.

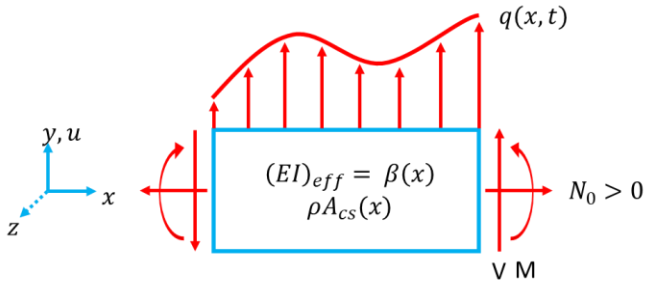


Figure 4.1. Axial loading beam bending diagram.

From Figure 4.1, we can write out our forces in the y-direction as well as our moments in equilibrium such that:

$$\frac{\partial V}{\partial x} + q(x,t) = \rho A_{cs}(x)\ddot{u}, \qquad (4.1)$$

$$\frac{\partial M}{\partial x} + V - N_0 \frac{\partial u}{\partial x} = 0, \qquad (4.2)$$

Combining the two into one equation gives us:

$$\frac{\partial^2 M}{\partial x^2} - N_0 \frac{\partial^2 u}{\partial x^2} = q(x,t) - \rho A_{cs}(x)\ddot{u}, \qquad (4.3)$$

From here, we can apply stress-strain relations to obtain an equation for our moment, *M*.

$$\sigma_{xx} = -Ey\frac{\partial^2 u}{\partial x^2}, \qquad (4.4)$$

$$M = (EI)_{eff}\frac{\partial^2 u}{\partial x^2} = \beta(x)\frac{\partial^2 u}{\partial x^2}, \qquad (4.5)$$

Finally, we can combine Equations 4.3 and 4.5 to give us the following differential equation which we will use as the general form for our FE method.

$$\frac{\partial^2}{\partial x^2}\left(\beta(x)\frac{\partial^2 u}{\partial x^2}\right) - N_0\frac{\partial^2 u}{\partial x^2} = q(x,t) - \rho A_{cs}(x)\frac{\partial^2 u}{\partial t^2} \quad 0 < x < \mathrm{L},\ 0 < t \le \mathrm{t_f}, \qquad (4.6)$$

In this implementation of the finite element method, we will solve for eigenvalues which we will use to give us the approximation of the solution. For the space and time response (frequency response) of our beam, we can estimate the solution by summing sinusoids of a given mode shape.

$$u(x,t) = \sum_{k=1}^{\infty}\left(c_1^{(k)}\cos\left(\omega_n^{(k)}t\right) + c_2^{(k)}\sin\left(\omega_n^{(k)}t\right)u^{(k)}(x)\right), \qquad (4.7)$$

We can set up our eigenproblem such that our eigenvalue, λ, is the square of the frequency of our sinusoids, ω, giving us the following.

$$\frac{\partial^2}{\partial x^2}\left(\beta(x)\frac{\partial^2 u^{(k)}}{\partial x^2}\right) - N_0\frac{\partial^2 u^{(k)}}{\partial x^2} = \lambda^{(k)}\rho A_{cs}(x)u^{(k)} \quad 0 < x < \mathrm{L}, \qquad (4.8)$$

Similar to previous chapters, we can use our basis functions to approximate *u* and define matrices based off the terms of Equation 4.8 and use Matlab to solve for an array of eigenvalues. We define the following matrices *A* (left side of Equation 4.8) and *F* (right side of Equation 4.8) as follows.

$$\tilde{A}_{ij} = \int_0^L \beta(x)\frac{d^2\varphi_i}{dx^2}\frac{d^2\varphi_j}{dx^2} + N_0\frac{d\varphi_i}{dx}\frac{d\varphi_j}{dx}dx, \qquad (4.9)$$

$$\tilde{F}_i = \int_0^L \lambda_h^{(k)}\rho A_{cs}(x)\sum_{j=1}^{2n_{node}} u_{hj}^{(k)}\varphi_i\varphi_j dx = \lambda_h^{(k)}\sum_{j=1}^{2n_{node}}\int_0^L \rho A_{cs}(x)\varphi_i\varphi_j dx\, u_{hj}^{(k)}, \qquad (4.10)$$

Expressing Equation 4.8 using Equations 4.9 and 4.10 gives us the following eigenproblem:

$$\underline{A}\,\underline{u}_h^{(k)0} = \underline{F} = \lambda_h^{(k)}\underline{M}^{inertia}\underline{u}_h^{(k)0}, \qquad (4.11)$$

Solving for the eigenvalues, $\lambda_h^{(k)}$, from Equation 4.11 will not only give us the deflection and velocity of a given point in the beam but will also give us the various modes of the beam's response. We will see how this can be useful in design in this chapter as we design and optimize a xylophone bar.

Imposing Boundary Conditions

In the case above, we have assumed a free-standing beam. However, in the case that we do wish to incorporate boundary conditions, we can do so by modifying our stiffness matrix *A* after solving for the matrix without boundary conditions (similar to how we implemented a Dirichlet condition).

Say we have a beam which we wish to constrain with a lumped Hookean spring attached to the right end (x = L). Our boundary conditions can then be expressed as follows:

$$u_{xx} = u_{xxx} = 0 \text{ at x = 0}, \qquad (4.12)$$

$$u_{xx} = 0, -(EIu_{xx})_x = -k_s u \text{ at x = L}, \qquad (4.13)$$

When we write out the stiffness matrix terms, we have an integral term similar to Equation 4.9, but we also add in the boundary condition such that we have the following:

$$A_{ij} = \int_0^L EI \frac{d^2\varphi_i}{dx^2}\frac{d^2\varphi_j}{dx^2}\, dx + k_s\varphi_i(L)\varphi_j(L),\ 1 \le i, j \le 2 \cdot n_{node}, \qquad (4.14)$$

In the implementation, the addition of the boundary condition occurs in the function *impose_boundary_cond.* The function will add the spring term into the matrix *A* by executing the following line of code in the else-statement of the if(Dir(1, 2) == true) loop.

```
A(n_el0 + 1, n_el0 + 1) = A(n_el0 + 1, n_el0 + 1) + u_Gamma2;
```

We can define u_Gamma2 as [k_s; 0] and set probdef.Dir as [[false; false], [true; false]] in *xylo_bar_design3* where we define our problem.

Recall that we wish to change the right boundary condition which corresponds to index n_{el_0} + 1 in our matrix rather than the last index number. As we refine, we continue number additional nodes and elements from where we left off rather than re-numbering all nodes and elements.

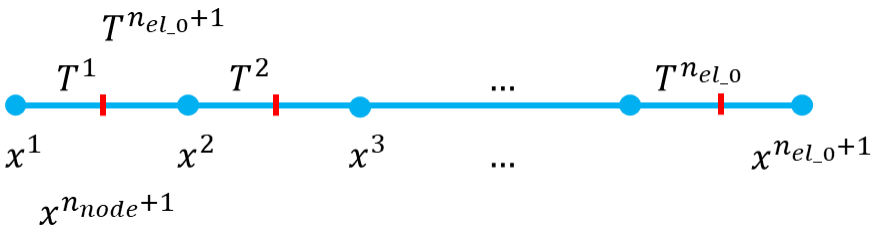


Figure 4.2. Visualization of node and element numbering while refining. Blue circles denote initial mesh nodes; red dashes denote refinement nodes.

As we see in Figure 4.2, the right most node where we wish to implement the boundary condition is node $x^{n_{el\_0}+1}$ and not $x^{n_{node}}$.

4.3 - Xylophone Bar Model

Xylophone bars are designed such that when struck, the bar's first mode corresponds to the frequency of a pitch while subsequent mode frequencies are a multiple of the first. Common frequency ratios are quint (a factor of 3) and double-octave (a factor of 4). While on the top surface, the bar is rectangular, the bottom is usually carved out to achieve the desired frequency response. To secure the xylophone bar, a string is run through two holes drilled in the bar; these hole locations are determined by the first mode's (fundamental mode) zeros. When the xylophone bar is struck, the bar will deflect, but at the hole locations, the deflection will be zero,

and the string will not interfere with the vibration of the bar. We can model and define our xylophone bar as follows.

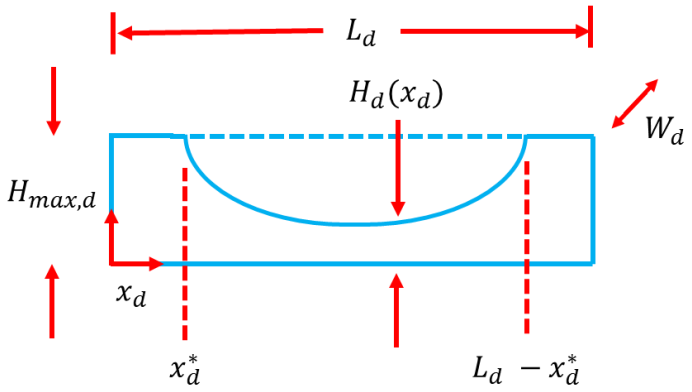


Figure 4.3. Xylophone bar with dimensions. We denote x_d^* as the starting x-distance of our cut defined by the height function $H_d(x_d)$.

We will use our FE method for beam bending eigenproblems to optimize the shape of the bar for a given target frequency and target frequency ratio. The differential equation we will be solving in dimensional form is:

$$\frac{d^2}{dx_d^2}\left(\frac{E_d W_d H_d^3(x_d)}{12}\frac{d^2 u_d^{(k)}}{dx_d^2}\right) = \lambda_d^{(k)} \rho_d W_d H_d(x_d) u_d^{(k)} \quad 0 < x_d < L_d\,, \qquad (4.15)$$

In nondimensional form, we can write Equation 4.15 as:

$$\frac{d^2}{dx^2}\left(\frac{H^3(x)}{12}\frac{d^2 u^{(k)}}{dx^2}\right) = \lambda^{(k)} H(x) u^{(k)} \quad 0 < x_d < 1\,, \qquad (4.16)$$

For a given target frequency and target frequency ratio, we can calculate the optimal height function, $H_d(x_d)$. From there, we can also calculate the optimal xylophone bar length, L_d.

To calculate the optimal hole locations in the bar, we first find which element of our reference mesh that contains a zero by looking for a sign change in our solution, $u_h^{(3)}(x)$. Once we have

established which element contains the zero, we find the exact location of the hole in the reference domain by solving for an $\hat{x}^{hole}$ that satisfies the following equation:

$$\sum_{l=1}^{4} u_h^{(3)} lg2(l, m^*) \hat{S}_{lm^*}(\hat{x}^{hole}) = 0, \qquad (4.17)$$

Finally, we can scale $\hat{x}^{hole}$ back to the dimensional domain by taking into account the element size and length of the xylophone bar:

$$x_{dh}^{hole} = \left(x^{\lg(1,m^*)} + h^{m^*} \hat{x}^{hole}\right) * L_d, \qquad (4.18)$$

We can easily check and see if our hole location algorithm is working properly by plotting the fundamental mode along with the bar and checking to see where our code finds the zeros.

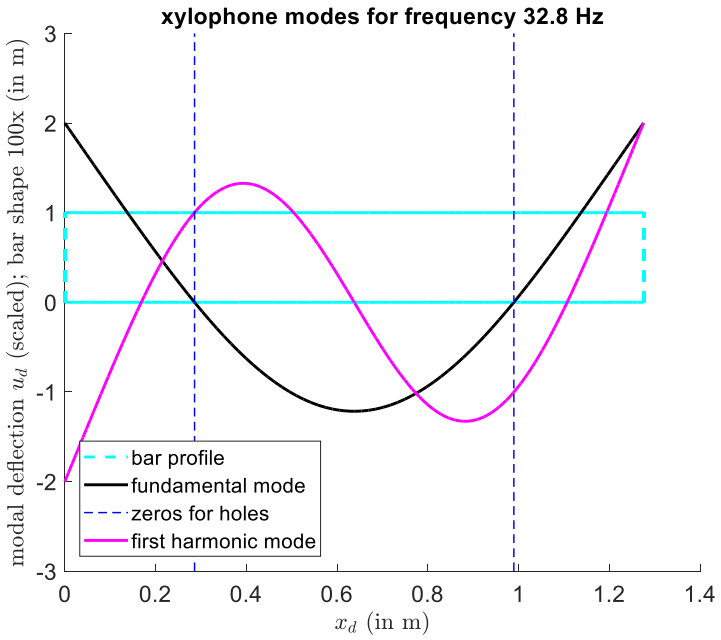


Figure 4.4. Bar design to respond with a frequency of 32.8Hz. The zeros are shown as dark blue dotted lines which cross both the bar and the fundamental mode (in black) at the same spot.

With our optimized height function, bar length, and hole locations, we can design and fabricate our xylophone bar.

4.4 - Verification

To verify that our FE method can find the optimal bar length, we can compare our FE results with another FE result as well as experimental data. In this case, we will be using data collected by Mauro Caresta on vibrations of a beam with no supported ends.

Physical Verification – Caresta Study

In Caresta's experiment, a steel beam was excited and produced a frequency response. The results of that response are shown below in Table 4.1.

Table 4.1. Caresta study theoretical and experimental frequency response of a steel beam with unsupported ends.

Mode	Theoretical [Hz]	Experimental [Hz]
n = 1	32.8	32.25
n = 2	90.44	88.5
n = 3	177.30	173.5
n = 4	293.08	287.5
n = 5	437.82	430

To test our experiment, we used the first mode frequency as our target frequency (32.8Hz) and a frequency ratio of 90.44 Hz and 32.8 Hz. We then had the code find the optimal bar length. In calling our FE method, we must specify that we want a constant height throughout the bar (a constant rectangular cross section), so our model matches the model used by Caresta.

The FE Matlab function we call requires specific inputs. For our desired fundamental frequency, we input 32.8 Hz with a desired frequency ratio of 90.44 Hz / 32.8 Hz or 2.76. We will use the height of the steel beam in Caresta's study of 0.01 m. For the material properties Young's Modulus and density, we use the same values as those reported in the study: 2.1×10^{11} Nm^{-2} and 7800 kgm^{-3}, respectively.

The FE Matlab function also requires two inputs that are more specific to the xylophone optimization problem we are trying to solve: *xstar* and *p2_interval_desired*. *xstar* denotes what x-direction value our height function will start from while *p2_interval_desired* is either a range of percentage of material remaining or the optimal percentage of material remaining at the middle of the bar. *p2_interval_desired*'s state is determined by *justcalc_L_d;* if *justcalc_L_d* is false, the FE code will calculate the optimal height function, and *p2-interval_desired* will be the range of possible percentages. If *justcalc_L_d* is true, the FE code will calculate the optimal bar length given the optimal height function (*p2_interval_desired* is the optimal percentage left at the middle of the bar). For our Caresta test case, because the beam in the study has no varying heights, we will set *justcalc_L_d* to true and input [1, 1] for *p2_interval_desired* (we wish to keep 100% of the material). *xstar* may be any value (less than 0.5) since we do not make a cut in the bar.

Table 4.2 shows the frequencies our FE method solved for alongside the data collected by Caresta.

Table 4.2. Our FE Method predictions next to Caresta's theoretical and experimental data.

	FE Method	Caresta	
Mode	**Theoretical**	**Theoretical [Hz]**	**Experimental [Hz]**
n = 1	32.8000	32.8	32.25
n = 2	90.4145	90.44	88.5

The length of the steel bar used in Caresta's experiment had a length of 1.275 m; our FE method predicted the length of a steel bar needed to create the frequency response to be 1.2752 m, very similar to the expect length.

Looking into the error convergence plots, we see that the error as we refine the mesh converge as expected.

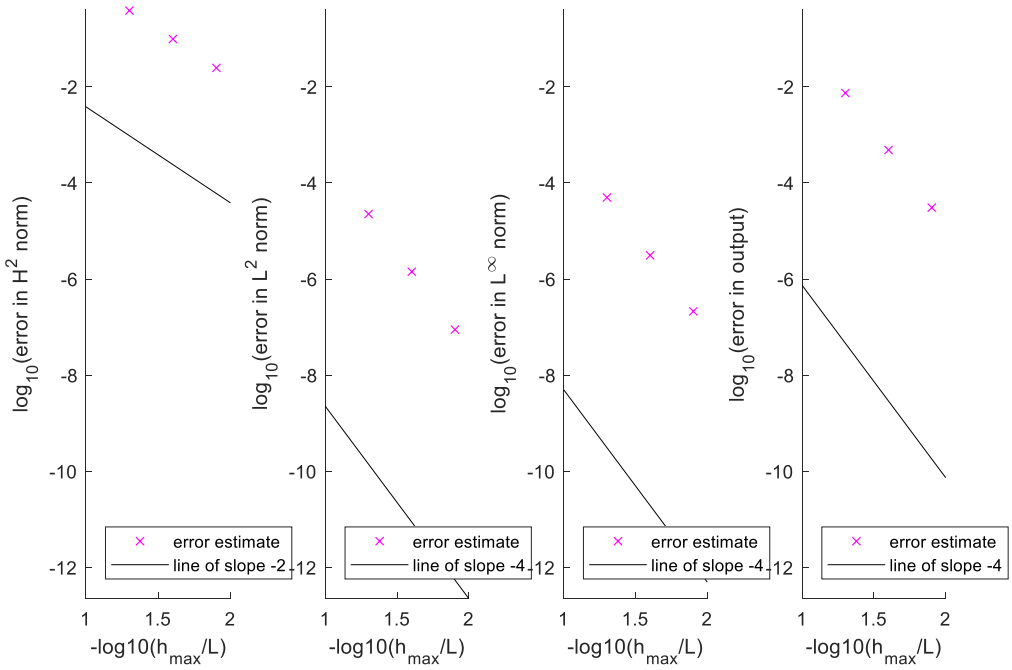


Figure 4.5. Error convergence plots in 4 norms for a steel bar with target frequency of 32.8 Hz.

Given that our FE method was able to produce the correct beam length given a target frequency, we can assume that that part of our solver is working properly. Additionally, we can assume our method works reasonably well since the solution matches experimental data collected by Caresta.

4.5 - Results

In this chapter, we will design 3 xylophone bars: F4, C5, and F5. To design and optimize the bars, we will first run our FE code with *justcalc_L_d* set to false and find our optimal height function. Once we have that, we will re-run the code with *justcalc_L_d* set to true, so we can get our optimal bar length. We will design all three bars to be quint harmonic (frequency ratio of 3).

Each bar will have a maximum height of 0.015 m and height functions that begin at an x-distance of 0.05 of the bar length. We will assume each bar is made of rosewood which has a Young's Modulus of 1.4×10^{10} Nm^{-2} and a density of 835 kgm^{-3}. The table below shows the numerical results of our FE code along with our desired frequency values.

Table 4.3. Desired and FE results for quint xylophone bars tuned to F4, C5, and F5.

	Desired Fundamental Frequency [Hz]	Desired Frequency Ratio [ ]	Desired 1st Harmonic [Hz]	Fundamental Frequency [Hz]	1st Harmonic [Hz]	Frequency Ratio [ ]	p2optimal [ ]	Length [m]
F4	349.23	3	1047.69	349.2300	1044.6	2.9913	0.6438	0.3264
C5	523.25	3	1569.75	523.2500	1859.2	3.5531	0.2667	0.1594
F5	698.46	3	2095.38	698.4600	2549.5	3.6501	0.2308	0.1270

We can also calculate our error estimates of the frequencies from the FE method. To calculate the error of the frequency ratio, we can use the following error estimate for quotients:

$$\frac{\delta q}{|q|} = \frac{\delta x}{|x|} + \cdots \frac{\delta z}{|z|}, \qquad (4.19)$$

The table below shows the error estimates from the final mesh after 3 uniform refinements.

Table 4.4. Error estimates of fundamental frequency, first harmonic, and frequency ratio.

	Fundamental Frequency Error	1st Harmonic Error	Frequency Ratio Error
F4	5.8496E-06	9.9144E-05	3.3400E-07
C5	3.7621E-05	5.4015E-04	1.2878E-06
F5	6.1124E-05	8.7008E-04	1.5651E-06

The following plots show both the bar shape as well as the hole locations for each xylophone bar:

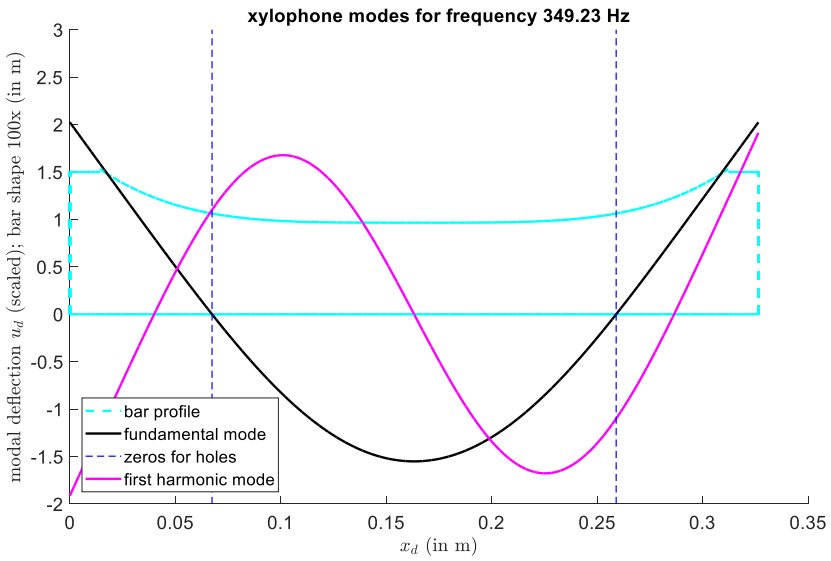


Figure 4.6. Visual representation of F4 bar with fundamental frequency, 1st harmonic, and hole locations.

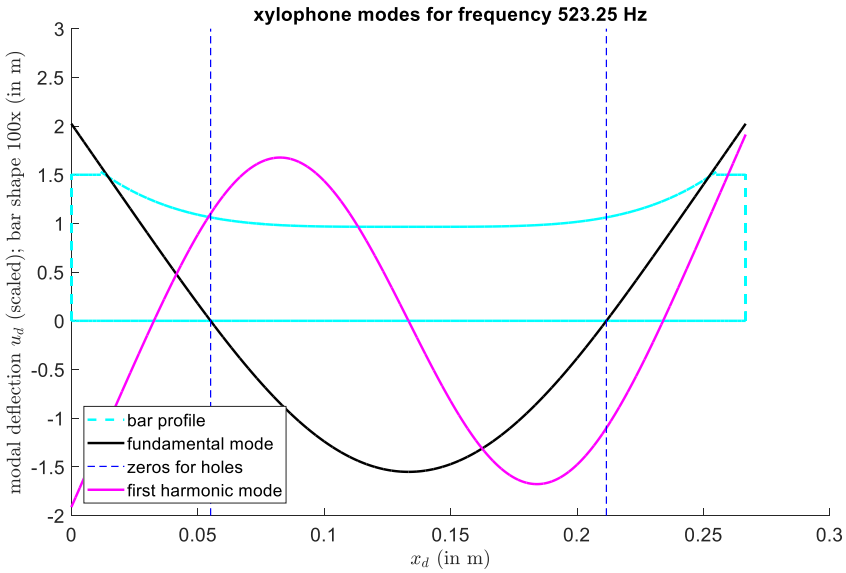


Figure 4.7. Visual representation of C5 bar with fundamental frequency, 1st harmonic, and hole locations.

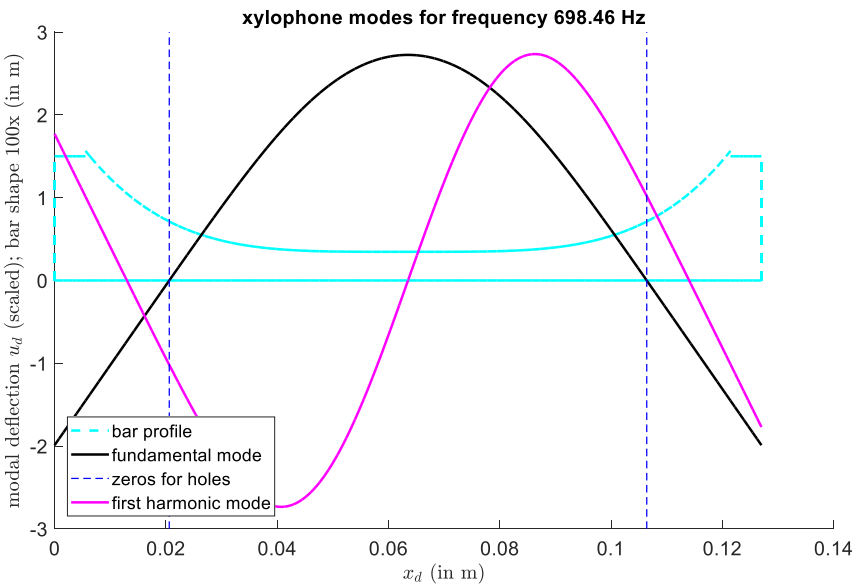


Figure 4.8. Visual representation of F5 bar with fundamental frequency, 1st harmonic, and hole locations.

4.6 Discussion

Our xylophone bar designs appear to make sense based on Figures 4.6, 4.7, and 4.8. We expect lower notes (F4) to require a larger/longer bar while the higher notes (C5 and F5) have smaller/shorter bars. This is reflected in our bar designs as F4 has the longest length (0.32m) while F5 has the shortest length (0.13m). Additionally, we see that p2optimal or our height functions follow a similar trend. The lower the note, the less material that needs to be removed (higher p2optimal).

Regarding error convergence, if we look at the convergence plots shown below, we can see that although our error estimates seem small from Table 4.4, they appear to be reliable as in all norms. They converge at the correct rates, and there are no computational precision issues.

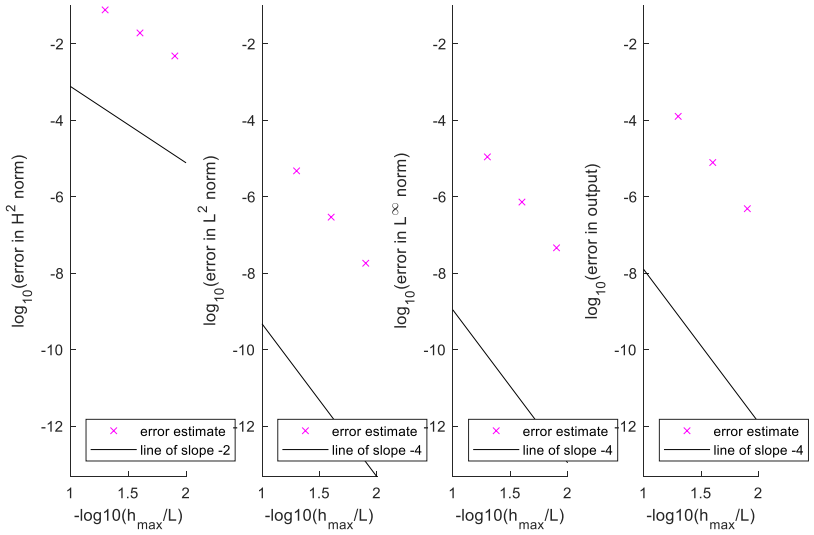


Figure 4.9. Error convergence plots for F4.

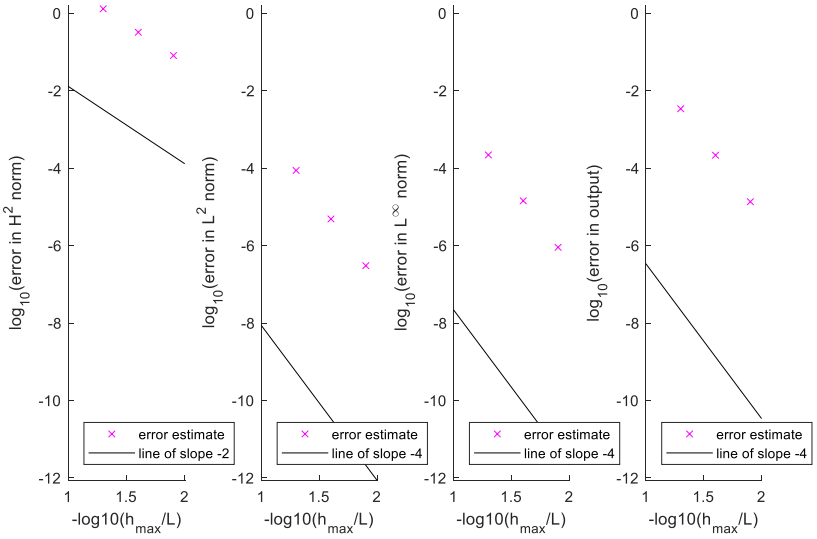


Figure 4.10. Error convergence plots for C5.

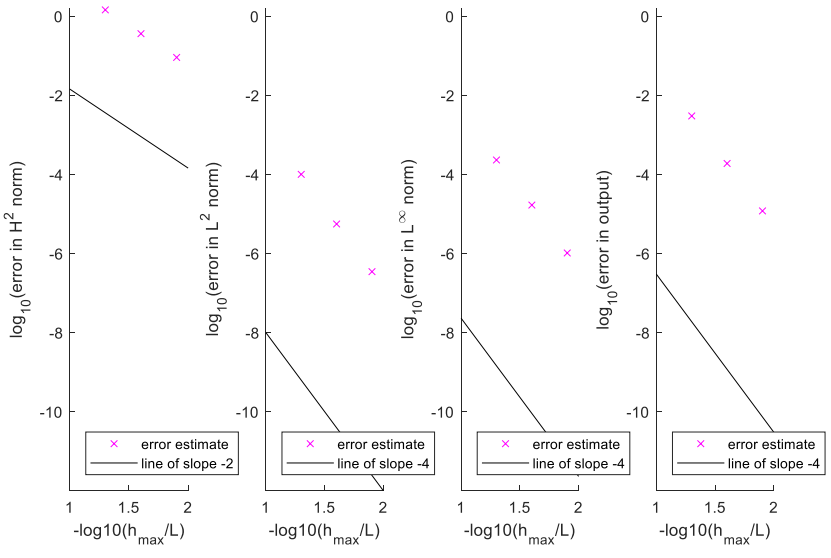


Figure 4.11. Error convergence plots for F5.

Looking at the error estimates of the 3 xylophone bars, we can see that as the desired fundamental frequency increases, the error estimates increase. This suggests that our FE method, for the given bar geometry, is more accurate at lower desired frequencies. When we ask for higher desired frequencies, we may be moving out of the range of our numerical model. Because our FE method is modeled after an Euler-Bernoulli beam (Equation 4.3), the model is only valid for infinitesimal strains and small rotations. As we decrease the size of the beam we solve for, we could be moving out of the regime for which Euler-Bernoulli beam theory holds.

For each xylophone bar, we ran 3 uniform refinements and saw that the solutions between each mesh converged. Based on what we know about frequency response, we know that the 1st harmonic shape, compared to the fundamental frequency, will have a more complex shape. For this reason, for the same mesh, we see that the FE error is larger for the 1st harmonic (Table 4.3). It will require a finer mesh to capture the 1st harmonic solution since there are more features than in the fundamental solution.

When we consider the application of our xylophone bars, because they are used to produce music for humans, we can determine whether or not we have refined our mesh enough. An un-trained human ear can distinguish pitches about 10 Hz apart. Since pitch is determined by the fundamental frequency, it seems our mesh was refined enough as the calculated fundamental frequency was very close to the desired frequency. We may have been able to stay within this tolerance with fewer mesh refinements which would cut down on the FE's runtime. However, our undertones (determined by the harmonics and frequency ratio) are not as close to the desired values as we would have hoped. F4 seems to meet our requirements, but C5 and F5 do not. To get within 10 Hz of the desired 1^{st} harmonic, we would likely have to refine the mesh further or consider switching models (as mentioned above).

4.7 - Final Thoughts

Using a similar form as our previous heat transfer problems, we can easily modify our FE method to solve beam bending eigenproblems and get the displacement and velocity of points on a beam.

In this chapter, we designed and optimized xylophone bars to various pitches/notes using our modified FE method. We saw that our FE method worked well for the lower frequency (F4). However, when we moved up in pitch, our error estimates increased as well. This could be attributed to the model we used to set up our FE method. Models can be useful under the correct assumptions and conditions; however, in our case, by asking for a higher desired fundamental frequency, we may have stepped out of the regime in which we can use the Euler-Bernoulli beam model.

While FE method can be done quickly and accurately, it is important to make sure you are using a valid model for the problem you are trying to solve.

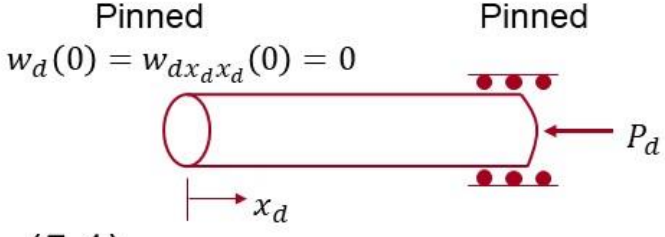
4.8 - References

1. Anthony T Patera, 2019, “Bending Energy Formulation.”
2. Anthony T Patera, 2019, “Bending FE.”
3. Anthony T Patera, 2019, “Bending Natural Frequencies.”
4. Anthony T Patera, 2019, “Bending Study Case Xylophone Bar.”

Chapter 5: The FE Method for 1D 4th-Order BVPs: Self-Buckling

Buckling

- Common structural failure
  - Ships, towers, columns
- Dimensional Form



$$\frac{d^2}{dx_d^2}\left(E_d I_d \frac{d^2 w_d}{dx_d^2}\right) + \frac{d}{dx_d}\left(P_d \frac{dw_d}{dx_d}\right) = q_d \qquad (5.1)$$

- Nondimensional Form used to solve eigenproblem

$$\frac{d^4 w}{dx^4} + P\frac{d^2 w}{dx^2} = q \qquad (5.2)$$

- Wish to find $P = P_{critical} = \lambda \equiv \lambda^{(1)} = \pi^2$

$$\frac{d^4 u}{dx^4} = \lambda\left(-\frac{d^2 u}{dx^2}\right) \qquad (5.3)$$

Massachusetts Institute of Technology

Self-Buckling: Problem and Constraints

- Buckling under structural weight

$$P_d = \int_{x_d}^{L_d} \rho g \pi R_d^2(x_d') dx_d' \quad (5.4)$$

- Nondimensional Form for Eigenproblem

$$\gamma = \frac{4\rho g L_d^3}{E_d \bar{R}_d^2} = \frac{4\pi\rho g L_d^4}{E_d V_d} < \gamma_c = \lambda^{(1)} \quad (5.5)$$

$$\frac{d^2}{dx^2}\left(R^4(x)\frac{d^2w}{dx^2}\right) + \gamma \frac{d}{dx}\left(P(x)\frac{dw}{dx}\right) = q(x) \quad (5.6)$$

- Design goal: tallest structure without buckling (maximize γ_c)

$$R(x) = \sqrt{1 + G(x)} \quad (5.7)$$

- Constraints
  - Fixed volume
  - Minimum relative radius
  - Gradual profile variation (quasi-1D)

Massachusetts Institute of Technology

FE Method: Self-Buckling Eigenproblem

- Solving for $\lambda_h = \gamma_{ch}$

$$\underline{A}\ \underline{u}_h^0 = \lambda_h\ \underline{K}^{ax}\ \underline{u}_h^0 \quad (5.8)$$

- Matrices A and K
  - Finding terms

$$\tilde{A}_{ij} = A_{ij}^N = \int_0^1 R^4(x)\frac{d^2\varphi_i}{dx^2}\frac{d^2\varphi_j}{dx^2}dx \quad (5.9)$$

$$\tilde{K}_{ij}^{ax} = K_{ij}^{ax\,N} = \int_0^1 P(x)\frac{d\varphi_i}{dx}\frac{d\varphi_j}{dx}dx \quad (5.10)$$

  - Forming $\underline{A}$ and $\underline{K}^{ax}$
    - Remove rows and columns 1 and 2 from Equations 5.9 and 5.10

Massachusetts Institute of Technology

G Function Derivation

- Truncated cone similar to lighthouse designs
- Linear radii profile

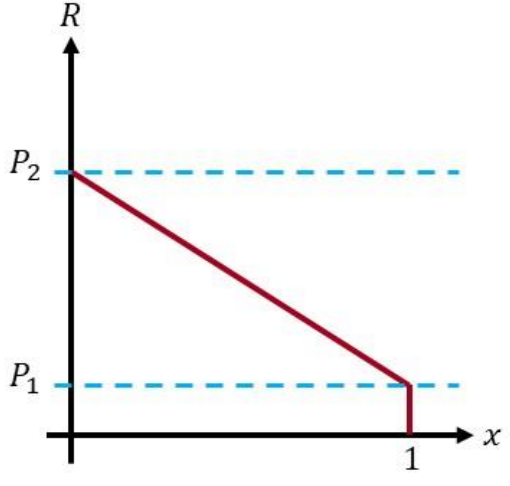
$$R(x) = (P_1 - P_2)x + P_2 = \sqrt{1 + G(x)} \qquad (5.11)$$

- Solve for $G(x)$

$$G(x) = \left((P_1 - P_2)x + P_2\right)^2 - 1 \qquad (5.12)$$

- Find P_1 and P_2 such that Volume Constraint is met

$$P_2 = \frac{-P_1 + \sqrt{12 - 3P_1^2}}{2} \qquad (5.13)$$

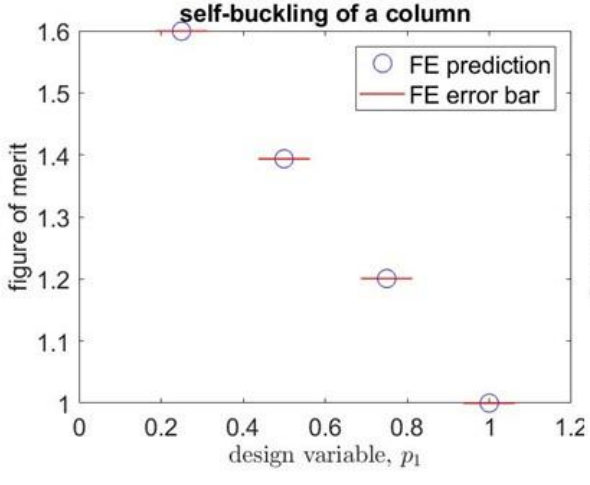


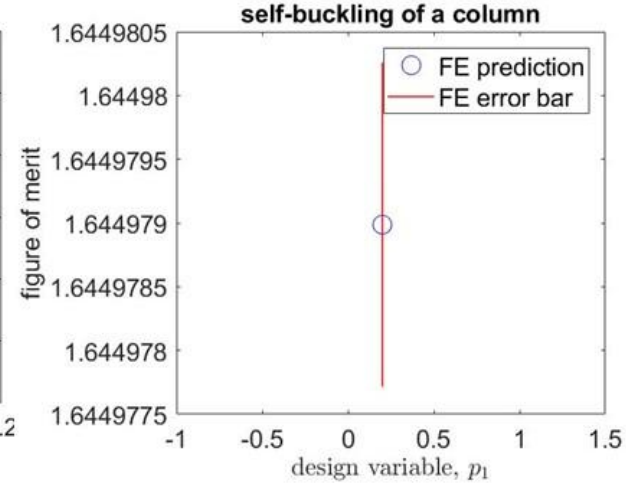
Massachusetts Institute of Technology

FE Method Results

- Find optimal P_1 and P_2
- FOM error within 0.01

P1 ϵ [0, 1]		
P1	**P2**	**FOM**
0.25	1.59	1.599
0.50	1.43	1.394
0.75	1.23	1.201
1.00	1.00	1.000
P1 ϵ [0, 0.25]		
P1	**P2**	**FOM**
0.20	1.62	1.645
0.25	1.59	1.599

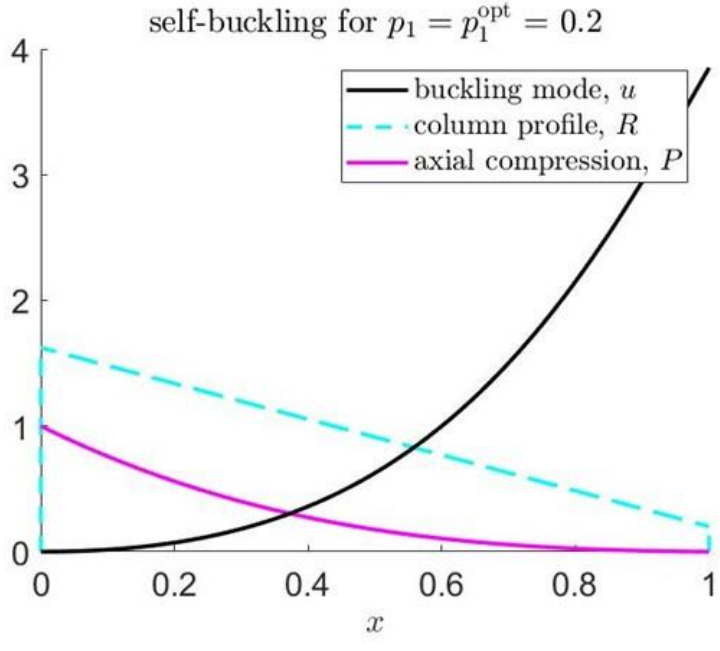




Massachusetts Institute of Technology

FE Method Results

- Radii Profile using P1 = 0.2

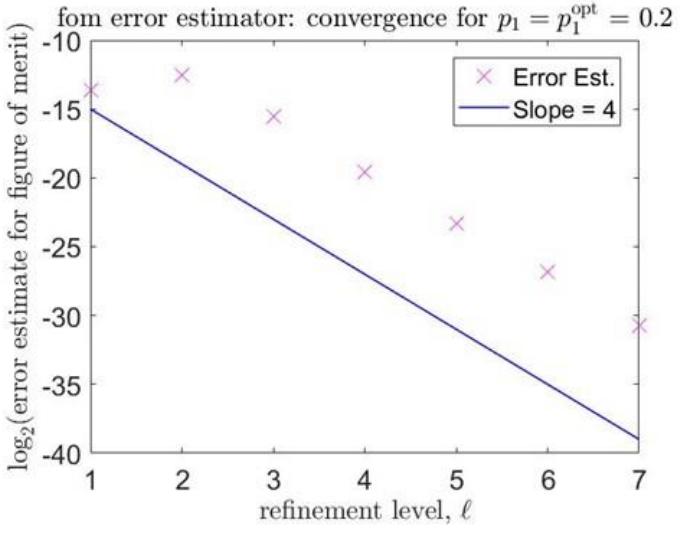


Massachusetts Institute of Technology

Mesh Refinement and Error

- Using P1 = 0.2, refine mesh
- FOM Error Estimate < 0.01
- FOM Error Estimate converges after 3 refinement

P1 = 0.2	
Level	FOM Error Est.
1	7.8E-05
2	1.7E-04
3	2.1E-05
4	1.3E-06
5	9.8E-08
6	8.4E-09
7	5.7E-10



Massachusetts Institute of Technology

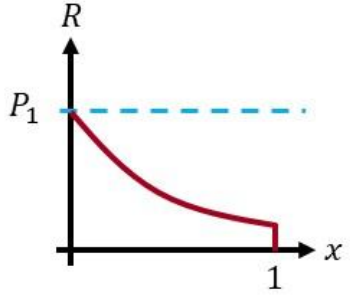
Comparison with Exponential G Function

- Exponential profile

$$R(x) = P_1 e^{-P_2 x} \qquad (5.14)$$

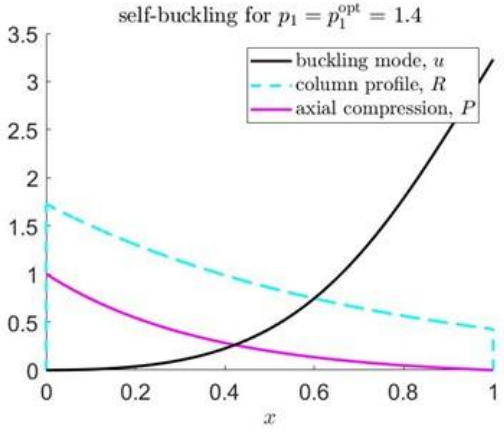
$$G(x) = P_1^2 e^{-2P_2 x} - 1 \qquad (5.15)$$

$$P_1 = \sqrt{\frac{2P_2}{1 - e^{-2P_2}}} \qquad (5.16)$$



- Results

Linear			Exponential		
P1 ϵ [0, 0.25]			P1 ϵ [1, 2]		
P1	P2	FOM	P1	P2	FOM
0.20	1.62	1.645	1.00	1.52	1.372
0.25	1.59	1.599	1.20	1.62	1.423



Massachusetts Institute of Technology

References

- Anthony T Patera, 2019, “Buckling_Eigenproblems2.”

Massachusetts Institute of Technology