10/13/2022

Gustavo Grinsteins Planchart
ASEN 5007 – Introduction to finite element methods
Mini-Project Report

Problem 1: Consider the following one-dimensional model problem:

Find u: [0, L] $\rightarrow \mathbb{R}$ such that:

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}(x)\right) = f(x)\ for\ all\ x \in (0,L)$$

$$and\ u(0) = g_0\ and\ u(1) = g_L\ where\ k: (0,L) \rightarrow\ \mathbb{R}^+, f(0,L) \rightarrow\ \mathbb{R}, g_0 \in \mathbb{R}, and\ g_L \in \mathbb{R}$$

Write a MATLAB function that approximately solves the above one-dimensional model problem using a Bubnov-Galerkin finite element approximation of the variational form. Your function should take the form:

```
function [x,d] = One_Dim_Model_Problem(k,n_el,kappa,f,g_0,g_L,L)
```

Where x is a vector containing the locations of the nodes, d is a vector containing the finite element solution at the nodes, k is the polynomial degree k of the finite element approximation to be employed, n_el is the number of elements nel to be employed, kappa is a function handle for computing k at any point x, f is a function handle for computing f at any point x, g_0 is g0, g_L is gL, and L is L. Your code should employ equally spaced elements, and it should support polynomial degrees k = 1, k = 2, and k = 3.

Problem 1 Solution:

The submission folder contains the following files:

- Grinsteins_MiniProject_DriverScript.m → Main scripts that runs code to obtain solutions for the project
- One_Dim_Model_Problem.m → Solution script to Problem 1
- quadrature_points.m → hardcoded information for quadrature points in $\Omega \rightarrow [-1,1]$
- Shape_Functions.m → hardcoded information for shape functions
- CalculateH1NormError.m → H1 Norm error calculations

The code control is depicted in this flow chart:

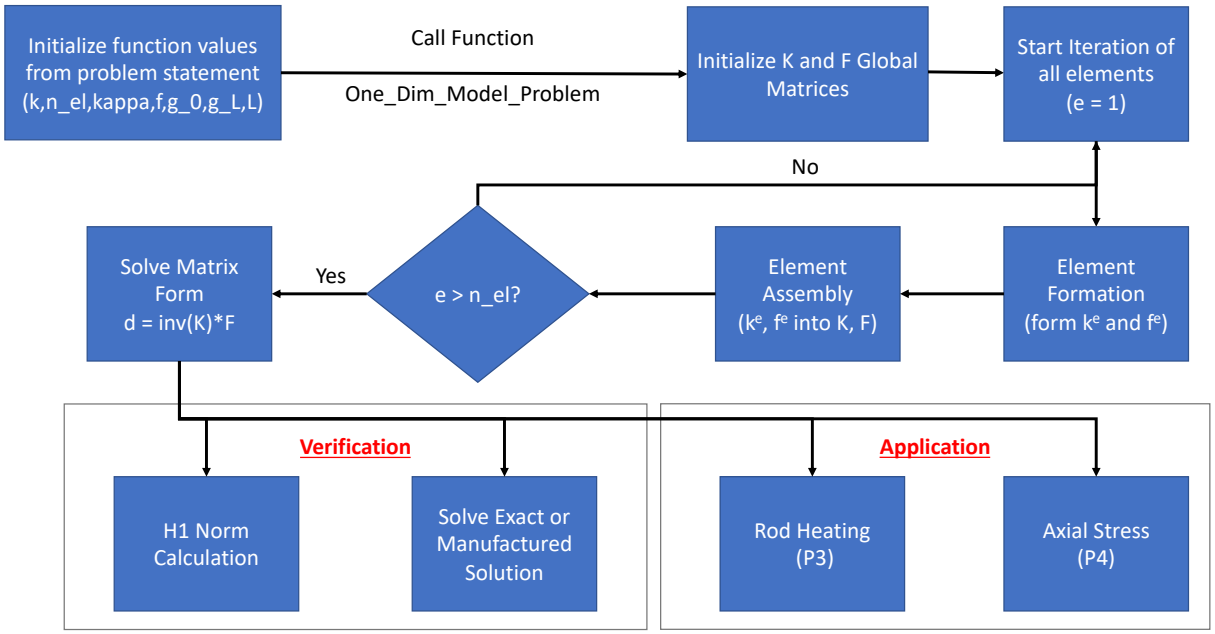

Figure 1 - Control Flow of Problem 1 Code

Instructions for usage:

An example usage logic has been defined in Grinsteins_MiniProject_DriverScript.m script. However, for general usage of the code you can follow this example:

Let's say you need to solve a one-dimensional partial differential equation with the following form:

$$-\frac{d}{dx}\left(k(x)\frac{du}{dx}(x)\right) = f(x) \quad for\ all\ x \in (0, L)$$

$$where\ u(0) = g_0\ and\ u(1) = g_L\ where\ k : (0, L) \rightarrow \mathbb{R}^+, f(0, L) \rightarrow \mathbb{R}, g_0 \in \mathbb{R}, and\ g_L \in \mathbb{R}$$

Here g_0 and g_L are boundary conditions for the problem and L is the length of your 1D domain. Furthermore, k(x) and f(x) can be constants or functions provided in your problem statement. For this example, assume $g_0 = 10,\ g_L = 30$, L = 100, k(x) = 2x, and f(x) = x^2.

This given values sets more than half of the One_dim_model_problem function (which is the engine of this code). You, as the user of this function, need to define the number of elements (n_el) and the polynomial degree for the approximation (k). Note here that the higher number of elements better approximates the solution and the higher polynomial degree the faster the solution converges to an answer. Just keep in mind that there are caveats with these parameters and higher values does not always mean better.

Suppose we pick 30 elements with k = 2 then you can call the One_Dim_Model_Problem the following way:

[x,d] = One_Dim_Model_Problem(2,30, 2x,x^2,10,30,100)

This call will return two vectors with the same length x and d. The vector x represents points in the domain from 0 to L (inclusive). The size of x is dependent on the number of elements and the k polynomial value. The vector d represents solutions from the 1D finite element approximation at each value of x.

From these results you can validate the solution by comparing it to the true solution, graph your results, plot errors, etc…

Enjoy!

Problem 2: Verify the MATLAB function you built for Problem 1 using a smooth, non-polynomial exact or manufactured solution. You are responsible for constructing such an exact or manufactured solution. Using your exact or manufactured solution, confirm that your code yields the expected rates of convergence in the H1-norm for polynomial degrees k = 1, k = 2, and k = 3.

Problem 2 Solution:

Problem 1 was verified by using the method of exact solutions and the method for manufactured solutions.

The exact solution for Homework 1 Problem 1 was used. For convenience, here is the problem statement:

Find u: [0, 1] → $\mathbb{R}$ such that:

$$\frac{d^2u}{dx^2} = x \; for \; all \; x \in (0,1)$$
$$and$$
$$u(0) = u(1) = 0$$

The analytical solution for this problem is the non-polynomial:

$$u(x) = \frac{x(x^2 - 1)}{6}$$

Using this exact solution, we can graphically confirm that the solutions for k values of 1, 2, and 3 yield values that lie on the solution curve:

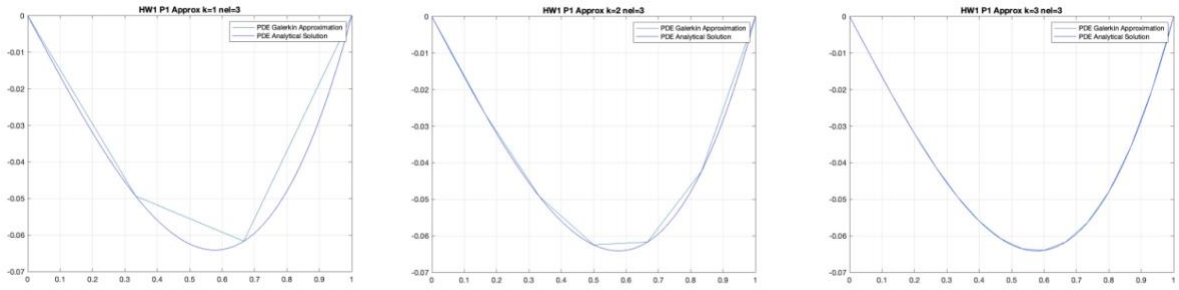

Figure 2 - Graphical Representation of approximation vs. solution

This graphical representation gives the confidence to proceed forward and observer the rate of convergence of the H1 Norm error. A MATLAB function called CalculateH1NormError.m was created which numerically represents the following mathematical expression:

$$\|u_h - u_{true}\| = \left(\sum_{i=1}^{n} \int_{-1}^{1} (u^h - u^{true})^2 \; d\xi + \sum_{i=1}^{n} \int_{-1}^{1} \left(u^h_{,\xi} - u^{true}_{,\xi}\right)^2 \; d\xi \right)^{\frac{1}{2}}$$

$$\|u_h - u_{true}\| = \left(\sum_{i=1}^{n} \sum_{q=1}^{q\_n} (u^h - u^{true})^2 \; d\xi + \sum_{i=1}^{n} \sum_{q=1}^{q\_n} \left(u^h_{,\xi} - u^{true}_{,\xi}\right)^2 \; d\xi \right)^{\frac{1}{2}}$$

Where n represents the number of elements in the approximation. The calculations consisted in using quadrature values of q_n = k +1 and using shape functions and their derivative to numerically compute this expression. Below is a graphical representation of the convergence of this error:

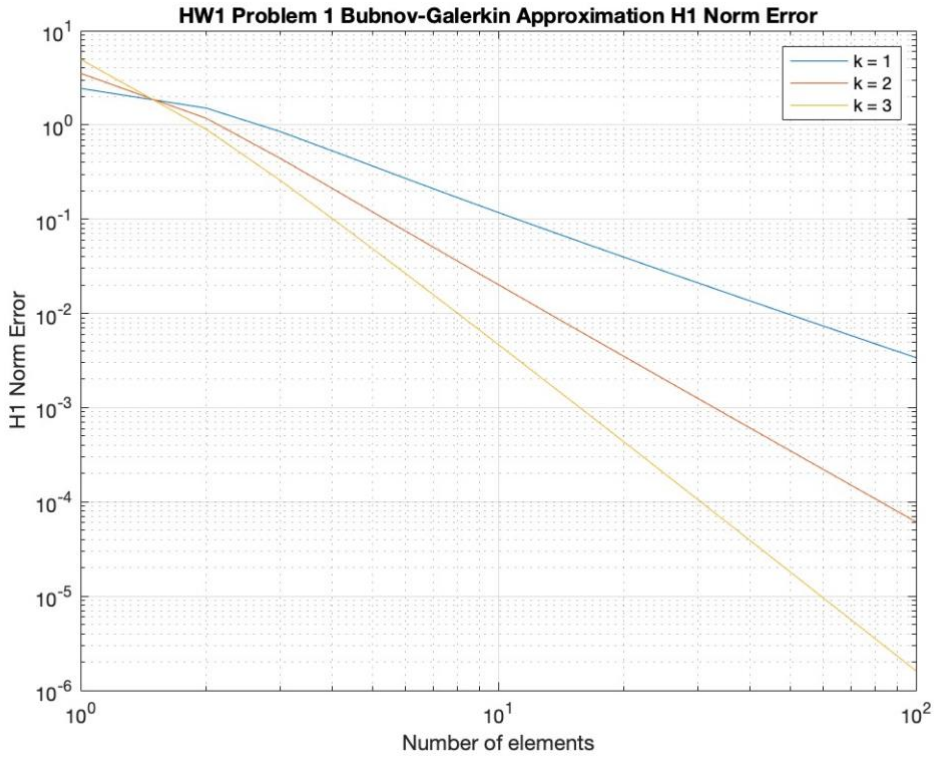

Figure 3 – Log-Log plot of H1 Norm error vs number of elements for HW1 P1

From this plot we can see that the rate of convergence is affected by the polynomial degree k value and the number of elements used for the approximation as expected. It is also worth noting that the convergence values to not totally approach zero. The solution error is stalling at a non-zero precision value. This is because actual convergence of a Galerkin finite element solution to the exact solution cannot be achieved due to finite precision arithmetic.

The convergence rates for the graph above were calculated using the following formula:

$$r = \frac{\ln(H_{h1e}) - \ln(h_{h1e})}{\ln(H) - \ln(h)}$$

Where H and h are the mesh sizes of two element configurations for the solve. The h1e subscript represents the H1 Norm error for that specific run. The results consisted of the r value calculation of the last two mesh sizes:

Polynomial Degree k	Convergence Value
1	1
2	2
3	3

These values represent the expected convergence rates.

Another validation test was performed using the following manufactured solution:

$$u(x) = e^{-x}\sin(\pi x)$$

This yields the following values:

$$f(x) = (\pi^2 - 1)e^{-x}\sin(\pi x) + 2\pi e^{-x}\cos(\pi x)$$

$$u_{,x}(x) = \pi e^{-x}\cos(\pi x) - e^{-x}\sin(\pi x)$$

Plotting the analytical solution against the approximation:

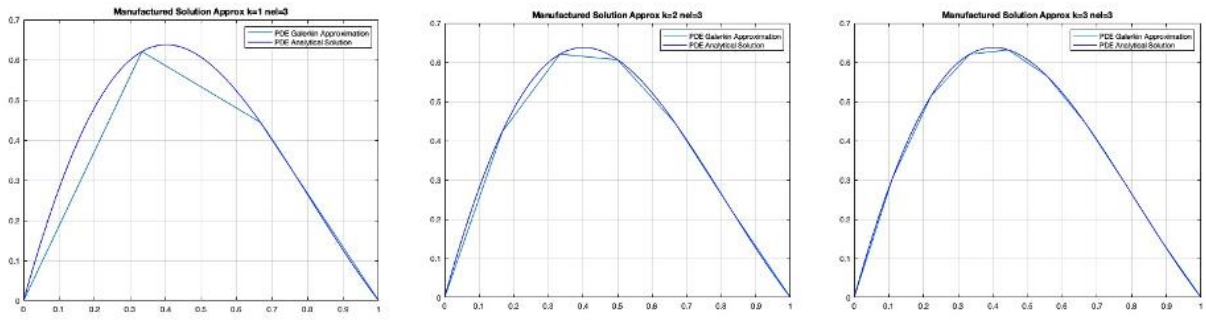

Figure 4 - Manufactured Solution Approximation

Plugging these values as parameters for the 1D fem function and calculating the H1 Norm error, the following chart is obtained:

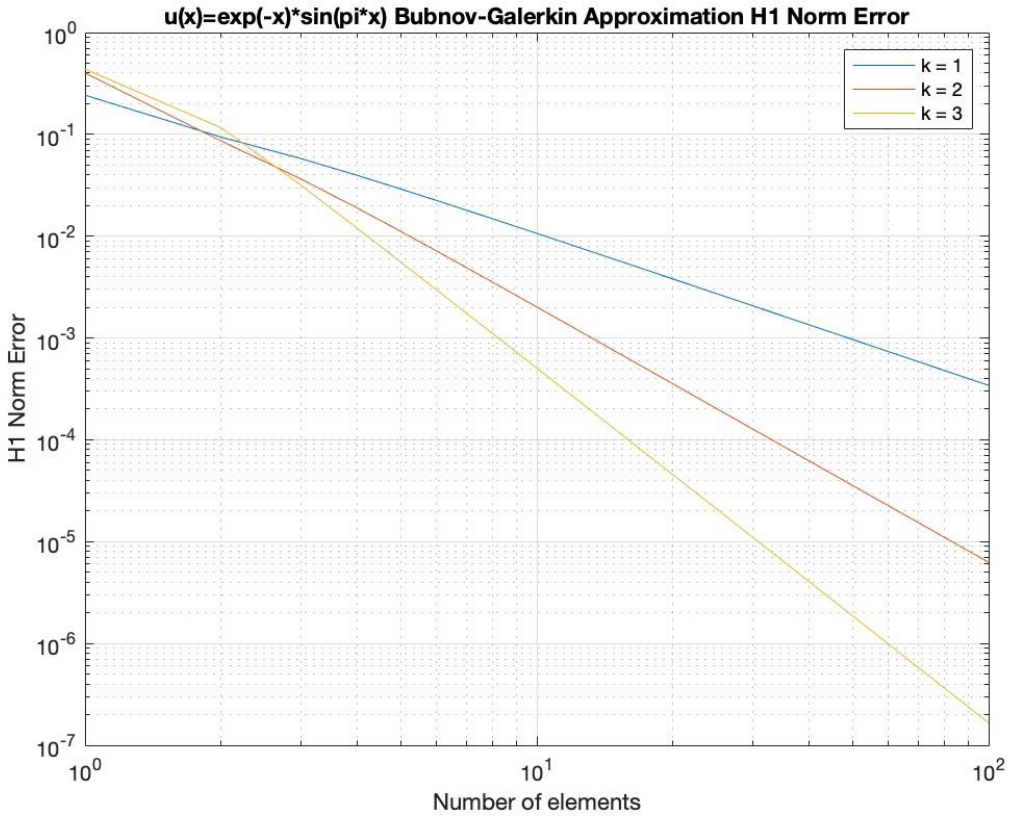

Figure 5 – Log-Log plot of H1 Norm Error vs. Number of Elements for Manufactured solution

Like the H1 Norm error calculated with the exact solution, a similar behavior of the error stalling at a non-zero value is observed.

Convergence Values Calculations:

Polynomial Degree k	Convergence Value
1	1
2	2
3	3

These values represent the expected convergence rates.

Problem 3: Consider a cylindrical rod of uniform cross-section and uniform material undergoing axisymmetric heating. The ends of the rod are kept at a fixed temperature. If steady-state rod temperature T is only a function of x, it is the solution of the following problem:

Find T: [0, L] → $\mathbb{R}$ such that:

$$-\frac{d}{dx}\left(k\frac{dT}{dx}(x)\right)=\frac{2}{R}h(x)\ for\ all\ x\in(0,L)$$

and T(0) = T(L) = Tend where κ ∈ R+ is the thermal conductivity of the rod, h : (0,L) → R is the heat flux applied along the surface of the rod, R ∈ R+ is the radius of the rod, and Tend ∈ R is the temperature at ends of the rod.

If the length L of the rod is 2 meters, the radius R of the rod is 0.025 meters, the rod is made of copper, the ends of the rod are at a fixed temperature of T end = 30 degrees Celsius, and the heat flux h is equal to:

$$h(x)=h_{max}e^{-100\left(\frac{x}{L}-0.5\right)^2}$$

where h max = 1500 Watts per square meter, use the MATLAB function you built for Problem 1 to determine the maximum temperature across the length of the rod.

Note the thermal conductivity κ of copper is approximately 385 Watts per meter per degree Celsius.

Problem 3 Solution:

Using the input parameters given in the problem statement and the code built for Problem 1 one can graph the rate of convergence of the finite element approximation for the maximum temperature across the length of the rod.

Setting up the following parameters:

```
kappa = @(x) 385; %Watts per meter per degree Celsius.
g_0 = 30; %Degrees Celsius
g_L = 30; %Degrees Celsius
L = 2; % Meters
h_max = 1500; % Watts per square meter
R = 0.025; % Meters
f = @(x) (2/(R))*h_max*exp(-100*((x/L)-0.5)^2);
```

The following graph is produced:

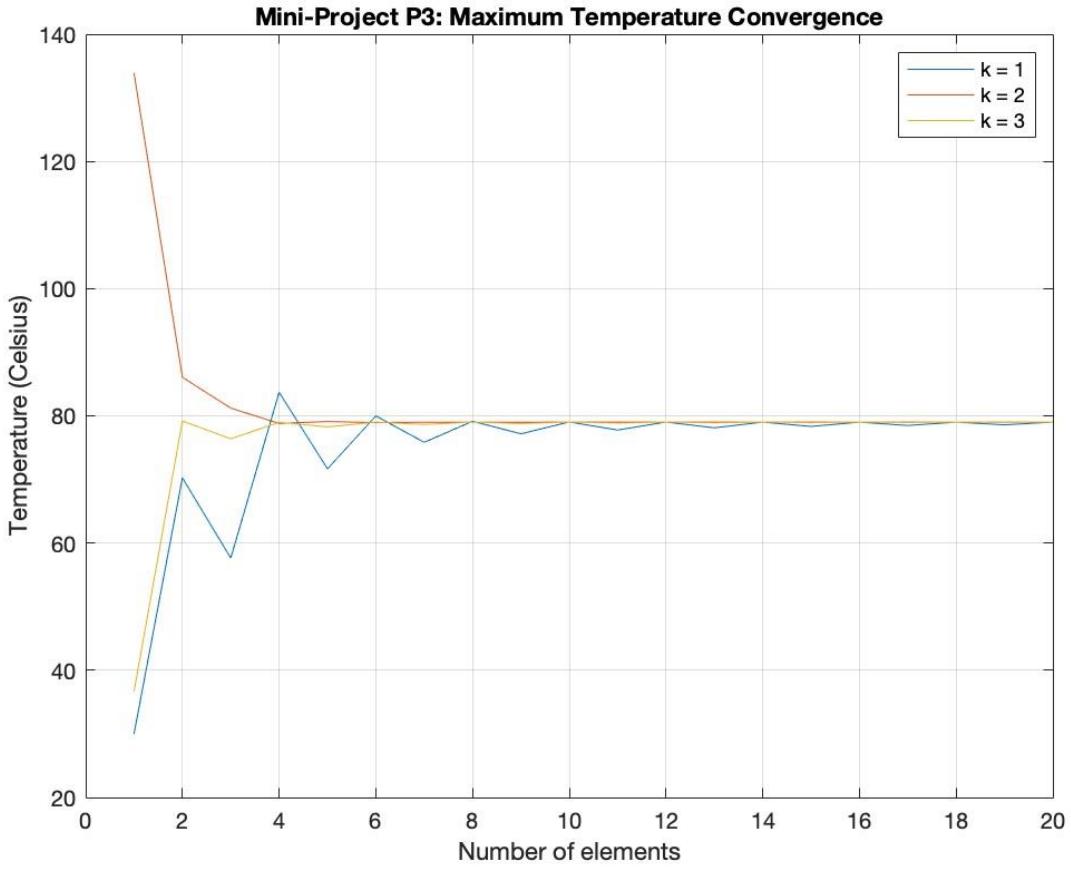

Figure 6 - Mini Project Problem 3 Solution

The graph accumulates solution values for increasing 1D mesh sizes at different polynomial values. The maximum temperature values are obtained by selecting the maximum value in the degrees of freedom array (d) returned by the One_Dim_Model_Problem function. **Starting at around 20 elements, all solutions start to converge at a common value of around 79.19 Degrees Celsius.**

Problem 4: Consider a tapered cylindrical rod of uniform material undergoing a tensile test. If rod axial displacement u is only a function of x and the effects of gravity are ignored, it is the solution of the following problem:

Find u: [0, L] → $\mathbb{R}$ such that:

$$-\frac{d}{dx}\left(EA(x)\frac{du}{dx}(x)\right) = 0 \; for \; all \; x \in (0, L)$$

And u(0)=0 and u(L)=u_{end} where E∈R^+ is the Young's modulus of the rod, A:(0,L)→R^+ is the cross-sectional area of the rod, and u_{end} ∈ R is the axial displacement at the bottom end of the rod.

If the length L of the rod is 0.1 meters, the cross-sectional area of the rod A is equal to:

$$A(x) = A_{max} - (A_{max} - A_{min})e^{-50\left(\frac{x}{L}-0.5\right)^2}$$

where A_{max} = 0.0001 square meters and A_{min} = 0.00002 square meters, the rod is made of ASTM- A36 steel, and the bottom end of the rod is displaced u_{end} = 0.00002 meters, use the MATLAB function you built for Problem 1 to determine the maximum axial stress across the length of the rod.

Note the Young's modulus E of ASTM-A36 steel is approximately 200 × 109 Pascals, and the axial stress is equal to $\sigma(x) = E\frac{du}{dx}(x)$

Problem 4 Solution:

Using the input parameters given in the problem statement and the code built for Problem 1 one can graph the rate of convergence of the finite element approximation for the axial stress across the length of the rod.

Setting up the following parameters:

```
g_0 = 0; %meters
g_L = 0.00002; %Degrees Celsius
L = 0.1; % Meters
f = @(x) 0;
A_max = 0.0001; %square meters
A_min = 0.00002; %square meters
E = 200e9;%Pascals
kappa = @(x) A_max-(A_max-A_min)*exp(-50*((x/L)-0.5)^2); % Square Meters
```

The following graph is produced:

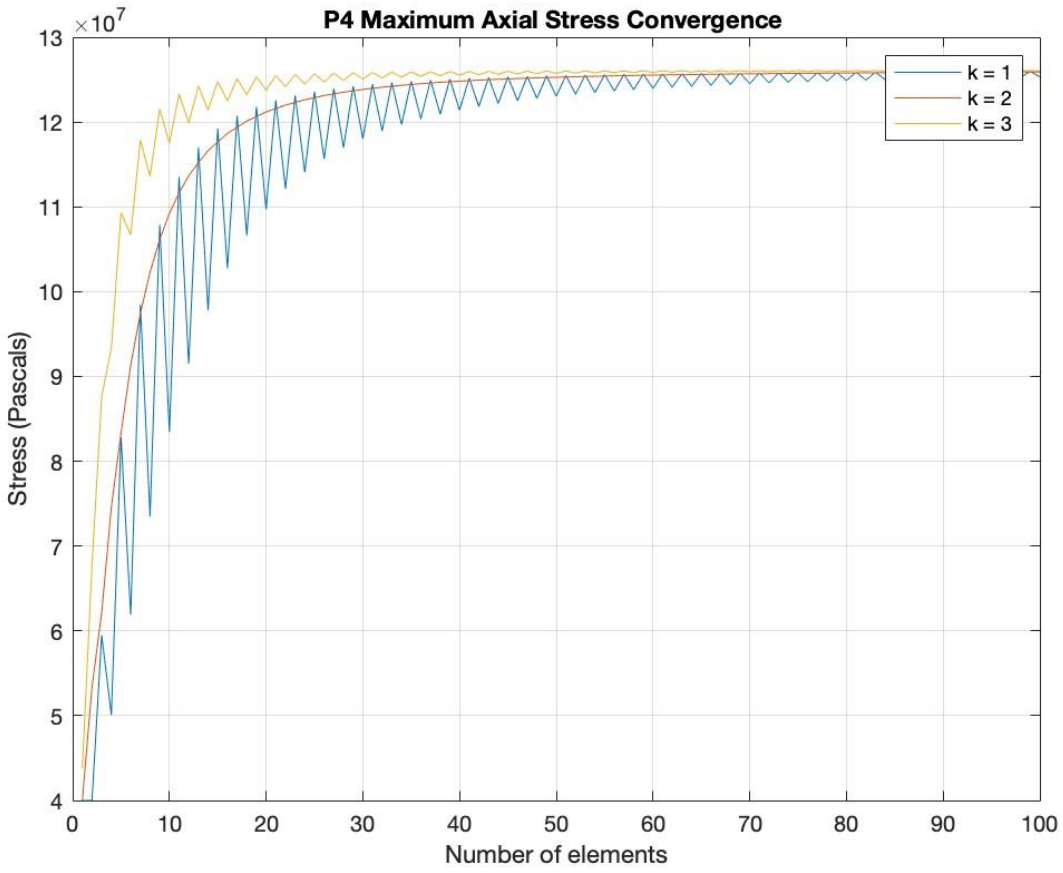

Figure 7 - Maximum Axial Stress Approximation

The graph accumulates solution values for increasing 1D mesh sizes at different polynomial values. The maximum stress values are obtained by selecting the maximum value of the slopes of the degrees of freedom (d) vs. domain node points (x) at different segments. Then this value is multiplied by the young's modulus. **Starting at around 100 elements, all solutions start to converge at a common value of around 1.259×10^8 Pascals.**