Modeling of a Heat-Induced Buckling of Plates Using the Mesh-free Method

Humberto Mejia
Humberto Mejia, hmeji001@fiu.edu

DOI: 10.25148/etd.FI14071179
Follow this and additional works at: https://digitalcommons.fiu.edu/etd
FLORIDA INTERNATIONAL UNIVERSITY

Miami, Florida

MODELING OF A HEAT-INDUCED BUCKLING OF PLATES USING THE MESH-FREE METHOD

A thesis submitted in partial fulfillment of
the requirements for the degree of
MASTER OF SCIENCES
in
MECHANICAL ENGINEERING
by
Humberto Mejia

2014
To: Dean Amir Mirmiran  
College of Engineering and Computing  

This thesis, written by Humberto Mejia, and entitled Modeling of a Heat-Induced Buckling of Plates using Mesh-free Method, having been approved in respect to style and intellectual content, is referred to you for judgment.

We have read this thesis and recommend that it be approved.

_______________________________________  
Yiding Cao  

_______________________________________  
Ibrahim Tansel  

_______________________________________  
Igor Tsukanov, Major Professor  

Date of Defense: July 02, 2014  

The thesis of Humberto Mejia is approved.

_______________________________________  
Dean Amir Mirmiran  
College of Engineering and Computing  

_______________________________________  
Dean Lakshmi N. Reddi  
University Graduate School  

Florida International University, 2014
ABSTRACT OF THE THESIS

MODELING OF A HEAT-INDUCED BUCKLING OF PLATES USING THE MESH-FREE METHOD

by

Humberto Mejia

Florida International University, 2014
Miami, Florida

Professor Igor Tsukanov, Major Professor

In the process of engineering design of structural shapes, the flat plate analysis results can be generalized to predict behaviors of complete structural shapes. In this case, the purpose of this project is to analyze a thin flat plate under conductive heat transfer and to simulate the temperature distribution, thermal stresses, total displacements, and buckling deformations. The current approach in these cases has been using the Finite Element Method (FEM), whose basis is the construction of a conforming mesh. In contrast, this project uses the mesh-free Scan Solve Method. This method eliminates the meshing limitation using a non-conforming mesh. I implemented this modeling process developing numerical algorithms and software tools to model thermally induced buckling. In addition, convergence analysis was achieved, and the results were compared with FEM. In conclusion, the results demonstrate that the method gives similar solutions to FEM in quality, but it is computationally less time consuming.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>CHAPTER</th>
<th>PAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>..........................................................1</td>
</tr>
<tr>
<td>2. MESH-FREE OF MODELING OF THE THERMALLY INDUCED BUCKLING</td>
<td>..........................................................3</td>
</tr>
<tr>
<td>2.1. Mathematical formulation of thermally induced buckling of plates</td>
<td>..........................................................5</td>
</tr>
<tr>
<td>2.1.1. Heat conduction</td>
<td>..........................................................5</td>
</tr>
<tr>
<td>2.1.2. Thermal plane stress problem</td>
<td>..........................................................5</td>
</tr>
<tr>
<td>2.1.3. Plate buckling problem</td>
<td>..........................................................6</td>
</tr>
<tr>
<td>2.2. Mesh-free Solution Structures (MSS)</td>
<td>..........................................................7</td>
</tr>
<tr>
<td>2.2.1. Construction of approximate distance field</td>
<td>..........................................................7</td>
</tr>
<tr>
<td>2.2.2. Examples of solution structures for heat conduction, stress analysis, and plate problems</td>
<td>..........................................................8</td>
</tr>
<tr>
<td>2.2.2.1. Heat conduction solution structures</td>
<td>..........................................................8</td>
</tr>
<tr>
<td>2.2.2.2. Stress analysis solution structures</td>
<td>..........................................................9</td>
</tr>
<tr>
<td>2.2.2.3. Plate problem solution structure</td>
<td>..........................................................9</td>
</tr>
<tr>
<td>2.3. Variational formulations</td>
<td>..........................................................10</td>
</tr>
<tr>
<td>2.3.1. Heat conduction</td>
<td>..........................................................10</td>
</tr>
<tr>
<td>2.3.2. Stress analysis</td>
<td>..........................................................10</td>
</tr>
<tr>
<td>2.3.3. Plate buckling</td>
<td>..........................................................11</td>
</tr>
<tr>
<td>3. NUMERICAL EXPERIMENTS</td>
<td>..........................................................12</td>
</tr>
<tr>
<td>3.1. Experiments set up</td>
<td>..........................................................12</td>
</tr>
<tr>
<td>3.2. Mesh-free method application</td>
<td>..........................................................15</td>
</tr>
<tr>
<td>3.3. Experiment 1</td>
<td>..........................................................17</td>
</tr>
<tr>
<td>3.4. Experiment 2</td>
<td>..........................................................24</td>
</tr>
<tr>
<td>3.5. Experiment 3</td>
<td>..........................................................30</td>
</tr>
<tr>
<td>4. CONCLUSIONS</td>
<td>..........................................................40</td>
</tr>
<tr>
<td>REFERENCES</td>
<td>..........................................................42</td>
</tr>
<tr>
<td>APPENDIX</td>
<td>..........................................................43</td>
</tr>
<tr>
<td>FIGURE</td>
<td>PAGE</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>Figure 1 Heat conduction problem boundary conditions</td>
<td>13</td>
</tr>
<tr>
<td>Figure 2 Stress analysis problem boundary conditions</td>
<td>14</td>
</tr>
<tr>
<td>Figure 3 Buckling problem boundary conditions</td>
<td>14</td>
</tr>
<tr>
<td>Figure 4 Particular omega function</td>
<td>15</td>
</tr>
<tr>
<td>Figure 5 Distance field construction</td>
<td>16</td>
</tr>
<tr>
<td>Figure 6 K vs H</td>
<td>18</td>
</tr>
<tr>
<td>Figure 7 Max (S-misses) vs H</td>
<td>18</td>
</tr>
<tr>
<td>Figure 8 Max (UV) vs H</td>
<td>19</td>
</tr>
<tr>
<td>Figure 9 Maxtemp vs H</td>
<td>19</td>
</tr>
<tr>
<td>Figure 10 Temperature for $n_{xy} = 10$</td>
<td>21</td>
</tr>
<tr>
<td>Figure 11 Temperature for $n_{xy} = 100$</td>
<td>21</td>
</tr>
<tr>
<td>Figure 12 Total displacement for $n_{xy} = 10$</td>
<td>22</td>
</tr>
<tr>
<td>Figure 13 Total displacement for $n_{xy} = 100$</td>
<td>22</td>
</tr>
<tr>
<td>Figure 14 Buckling shape for $n_{xy} = 10$</td>
<td>23</td>
</tr>
<tr>
<td>Figure 15 Buckling shape for $n_{xy} = 100$</td>
<td>23</td>
</tr>
<tr>
<td>Figure 16 K vs Tdiff</td>
<td>25</td>
</tr>
<tr>
<td>Figure 17 Max (S-misses) vs Tdiff</td>
<td>25</td>
</tr>
<tr>
<td>Figure 18 Max (UV) vs Tdiff</td>
<td>26</td>
</tr>
<tr>
<td>Figure 19 Temperature for Tdiff = 5</td>
<td>27</td>
</tr>
<tr>
<td>Figure 20 Temperature for Tdiff = 80</td>
<td>27</td>
</tr>
<tr>
<td>Figure 21 Total displacement for Tdiff = 5</td>
<td>28</td>
</tr>
</tbody>
</table>
Figure 22 Total displacement for Tdiff = 80 .................................................................28
Figure 23 Buckling shape for Tdiff = 5 .................................................................29
Figure 24 Buckling shape for Tdiff = 80 .................................................................29
Figure 25 K vs R ........................................................................................................30
Figure 26 Ansys K vs R ..........................................................................................31
Figure 27 Max (S-misses) vs R ................................................................................31
Figure 28 Ansys Max (S-misses) vs R .................................................................32
Figure 29 Max (UV) vs R ........................................................................................32
Figure 30 Ansys Max (UV) vs R .............................................................................33
Figure 31 Temperature for R = 0.05 ........................................................................34
Figure 32 Ansys Temperature for R = 0.05 ..........................................................34
Figure 33 Temperature for R = 0.5 .........................................................................35
Figure 34 Ansys Temperature for R = 0.5 .............................................................35
Figure 35 Total Displacement for R = 0.05 ..........................................................36
Figure 36 Ansys Total Displacement for R = 0.05 ..............................................36
Figure 37 Total Displacement for R = 0.5 .............................................................37
Figure 38 Ansys Total Displacement for R = 0.05 ..............................................37
Figure 39 Buckling Shape for R = 0.05 .................................................................38
Figure 40 Ansys Buckling Shape for R = 0.05 .....................................................38
Figure 41 Buckling Shape for R = 0.5 .................................................................39
Figure 42 Ansys Buckling Shape for R = 0.5 .....................................................39
1. INTRODUCTION

Structural shapes are very important components of mechanical and structural engineering, and the flat plate is the basic unit of any structural form. For that reason, the engineering analysis of the flat plate is very important to predict the behavior of the complete related structural element[1]. For example, this is useful in the lateral stability of beams, plates, shells, or new materials in the process of engineering design of aircrafts, sea crafts, structures, etc. [2]. In addition, thermally induced buckling is important because using different heat quantities and sources, we should observe the effect over a plate under different loads and support methods; then we will be able to extrapolate behaviors of complex structural analysis.

In this research, a thin flat plate as a structural unit will be analyzed under heat-induced buckling in order to obtain numerical algorithms and software tools that let us study different plates geometries, materials, boundary conditions, loads, etc. For instance, in a welding process, a metallic flat plate expands when it is under a local concentration of heat, but the surrounding unheated metal causes plane stresses and finally develops the buckling effect on the plate [3,4]. Moreover, this buckling effect introduces a new load that does not let stability on the plate. In fact, the modeling process analyzes the kind of heat applied, temperature gradient, plane stresses, and buckling effect[5–7].

As you can see, this is a very complex analysis or multi-physics problem because it combines distinct but coupled physical phenomenon into a single numerical simulation. This is, one physical phenomenon requires information from the other one over the entire domain [8]. Some examples are heat transfer and stress analysis on a plane plate. The current approach used for solving multi-physics problems has been the Finite Element
Method (FEM). This method creates a conforming mesh over the object under investigation. However, in meshing analysis, there are a remarkable geometric differences between the real model and the simulation model, and for this reason the solution error becomes too high [9].

Additionally, meshing methods are very expensive computationally. Several mesh-free analysis techniques have been proposed in the past two decades, for example: smooth particle hydrodynamics, the diffuse element method, the reproducing Kernel particle method, the HP cloud method, the mesh-less local Petrov-Garlekin approach, the partition of unity method, and others [9]. On the other hand, this research proposes to use the Scan Solve Method (SSM) instead of trying to improve meshing technology. This new technology, based on FEM in situ, eliminates the modeling limitations, requires small features, is more flexible even with complex boundary conditions and interfaces, and maintains FEM benefits. It was built using the classical ideas of Kantorovich and Rvachev [9]. This free-meshing method has some steps to solve the heat transfer problem in a particular plate: establish boundary conditions, find the field equation, define geometry and distances to the boundary, test the point membership, and obtain the solution structure of the differential equation. I developed the algorithms and software to find the solution and visualize the simulation in every case. In this modeling process, I obtained temperature distribution, stress analysis, and deformation of the plate. In addition, I compared all results with FEM commercial software.

On the other hand, my personal contribution to this project includes building up numerical algorithms and software tools based on the mesh-free Solution Structure. I successfully implemented the modeling process on C++ code. This C++ code was
organized in four parts: heat conduction (step 1), thermal stress analysis (step 2), plate buckling problem (step 3), and general variables.

2. MESH-FREE OF MODELING OF THE THERMALLY INDUCED BUCKLING

The Finite Element Method (FEM), based on the Finite Element Analysis (FEA), has been a very important engineering tool for solving multi-physics problems and finding computing simulations. This method converts the geometric model into a conforming mesh of elements. That is, the original geometry and the modified model have important differences increasing the error, affecting the quality of the geometry analysis, and engineering design. Besides, it is very expensive computationally. In order to improve the engineering design, several free-meshing methods have been implemented.

The FEA in situ is one of these proposals [9, 10]. This method eliminates meshing limitation. The FEA in situ uses the Scan Solve Method (SSM), which is based on the Kantorovich method and Rvachev concepts. This method uses a native geometry in situ with two fundamental queries: point membership, and distance to boundary computation. Some advantages are flexibility in handling geometric error, small features, complex boundary conditions, interfaces, and maintaining most of the FEA benefits. The SSM can be applied in all types of FEA, linear, 2nd and 3rd dimension.

The general procedure should be: define the system, obtain the field equation, define boundary condition, construct geometry function $\omega$ that describes all domain geometry, and then apply $U = \omega \phi + \varphi$ as the structure of the solution of the field equation. The function $\omega$ enforces the domain’s homogeneous boundary conditions, and $\varphi$ corresponds to non-homogeneous boundary conditions. The function $\phi$ is unknown,
but the solution can be obtained by the linear combination of basis functions:

$$\phi = \sum_{i=1}^{n} C_i \chi_i.$$  

Where \(C_i\) is a scalar coefficient and \(\chi_i\) is the basis function (can be polynomial, b-splines, radial, and FE shape function). Then, \(U\) and \(\omega\) are used to modify the shape function (b-splines) in the form of the non-conforming grid. In other words, in a specific model a function \(U\) can be constructed at run time directly from geometry, and then can define function \(\omega\). Both \(U\) and \(\omega\) enforce prescribed fixed boundary conditions.

The solution structure developed by Kantorovich is a classical numerical technique for the approximate solution of differential equations with homogeneous Dirichlet boundary conditions. The Rvachev method or R-function method is used to construct the function \(\omega\) for a specific geometric domain. The R-function encodes Boolean functions that let us represent any object defined by a logical predicate [9, 10]. The procedure of constructing \(\omega\) function starts decomposing the domain geometry into its elementary domains. Then, the boundaries are related using Boolean algebra, and the \(\omega\) function is constructed according to the Boolean logic concept with each \(\omega_i\). Then, the given domain will be the intersection or union of the sweeping areas. Only the signs of their argument determine the sign of these functions. \(\omega = 0\) in the boundary, \(\omega > 0\) inside the boundary, and \(\omega < 0\) outside the boundary. It is interpreting \(\omega > 0\) as true, and \(\omega < 0\) as false. Finally, using transfinite interpolation we can find the function \(\varphi\) for the domain. This is, interpolating distances for \(\varphi_i\) to obtain the final algorithm for \(\varphi\) [9].

One of the more frequently applications of mesh-free SSM is on boundary value problems, these sort of problems can be recognized for the following characteristics: a given domain, a set of constrains (boundary conditions), and a physical law (differential equation). A solution to a boundary value problem is a solution to the differential
equation that also satisfies the boundary conditions. From a computational point of view, the intrinsic advantage of the above procedure is in the clean and modular separation of the geometric information represented by function $\omega$, and the numerical method used to find $\phi$. On the other hand, the SSM method is a very useful tool to solve multi-physic problems. In this kind of problems, it is necessary to create a mesh, but it is very difficult, and the final error is too high. Then, the best way to find the solution is using mesh-free methods, and the SSM is a very good option because of its advantages mentioned above [8, 9, 10].

2.1. Mathematical formulation of thermally induced buckling of plates

2.1.1. Heat conduction

Given: a rectangular plate with a circular center hole, with no heat source neither temperature variation. This is a steady state heat flow problem.

Boundary conditions: $\varphi_1$ temperature on the rectangle perimeter, $\varphi_2$ temperature on the inner circle perimeter.

The steady state equation for heat conduction is $\nabla^2 T = 0$ and the boundary conditions $T|_{\partial \Omega_1} = \varphi_1$, $T|_{\partial \Omega_2} = \varphi_2$.

2.1.2. Thermal plane stress problem

Plane stress is a state of stress in which the normal stress $\sigma_z$ and the shear stresses $\tau_{xz}$ and $\tau_{yz}$ directed perpendicular to X-Y plane are assumed zero. In this case, the loads are applied over the thickness of the plate and act in the plane of the plate. The plane stress condition is the simplest form of behavior for continuum structures and represents situations frequently encountered in practice [12]. Given: a thin rectangular plate simple supported on both vertical boundaries, that is, $U = V = 0$. Where, $U$ and $V$ are the x
and y displacements respectively. The goal in this problem consists of finding displacements U, V, and stresses. For stresses, we are going to use the classical equations:

\[
\begin{align*}
\delta_x &= \frac{E}{1-\nu^2} \left( \frac{\partial u}{\partial x} + \nu \frac{\partial v}{\partial y} \right) - \frac{E\alpha}{1-\nu}, \\
\delta_y &= \frac{E}{1-\nu^2} \left( \frac{\partial u}{\partial y} + \nu \frac{\partial v}{\partial x} \right) - \frac{E\alpha}{1-\nu}, \\
\tau_{xy} &= G\gamma_{xy} = \left( \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \right) \\
\delta_1 &= (\delta_x + \delta_y) + \sqrt{\left( \frac{\delta_x - \delta_y}{2} \right)^2 + \tau_{xy}^2}, \\
\delta_2 &= (\delta_x + \delta_y) - \sqrt{\left( \frac{\delta_x - \delta_y}{2} \right)^2 + \tau_{xy}^2}, \\
\delta_{vm} &= \sqrt{\delta_1^2 + \delta_2^2 - \delta_1\delta_2}.
\end{align*}
\]

Where \( \delta_x \) and \( \delta_y \) are normal stresses, \( \tau_{xy} \) is shear stress, \( \gamma_{xy} \) is shear strain, \( \nu \) is the Poisson’s ratio, \( G \) is the shear modulus, \( \alpha \) is the thermal expansion coefficient, \( E \) is the elasticity ratio[14].

2.1.3. Plate buckling problem

Before buckling, the plate is subject to a system of two-dimensional stresses \( K\alpha_x^0 \), \( K\alpha_y^0 \), \( K\tau_{xy}^0 \), where the monotonically increasing factor of proportionality is \( K \), and the distribution of stresses is prescribed (initial stresses). During the buckling, the external forces over the plate do not vary. From the principle of virtual work, the principle of stationarity potential energy for the buckling problem is obtained

\[
\pi = \frac{1}{2} \int \int D \left\{ \left( \frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} \right)^2 + 2(1-\nu) \left[ \left( \frac{\partial^2 w}{\partial x \partial y} \right)^2 - \frac{\partial^2 w}{\partial x^2} \frac{\partial^2 w}{\partial y^2} \right] \right\} \, dx \, dy + \frac{1}{2} K \int \int \left[ N_x^0 \left( \frac{\partial w}{\partial x} \right)^2 + N_y^0 \left( \frac{\partial w}{\partial y} \right)^2 \right] \, dx \, dy.
\]
\[ 2N_{xy} \frac{\partial W}{\partial x} \frac{\partial W}{\partial y} dxdy \], and the independent function subject to variation is \( W \) (plate deformation), and \( K \) is obtained from the quotient defined by:

\[
K = \frac{1}{2} \iint_{\Omega} D \left( \left( \frac{\partial^2 W}{\partial x^2} + \frac{\partial^2 W}{\partial y^2} \right)^2 + 2(1-\nu) \left( \frac{\partial^2 W}{\partial x \partial y} \right)^2 - \frac{\partial^2 W}{\partial x \partial y} \frac{\partial^2 W}{\partial x \partial y} \right) dxdy
\]

where the bending rigidity is

\[
D = \frac{Eh^3}{12(1-\nu^2)} \quad [14]
\]

2.2. Mesh-free Solution Structures (MSS)

2.2.1. Construction of approximate distance fields

Distance field is a representation where at each point within the field, we know the distance from that point to the closest point on any object within the domain. When the distance field is signed, we may also determine if the point is internal or external to objects within the domain, and the signed distance to the boundary is used to denote whether we are inside or outside to the domain[11,12]. The signed distance field of a surface can effectively support much geometry processing tasks such a decimation, smoothing, and Boolean operations since it provides efficient access to distance (error estimates) [9]. The function constructed as smooth approximation of the Euclidean distance is the distance function \( \omega \). This function lets us handle two important queries: point membership testing and distance to boundary computation in order to apply the SSM [9].

An example is a rectangular plate with a centered hole in the middle and dimensions: vertical side 2b, horizontal side 2a, and hole’s radius r. The vertical and horizontal boundaries generate the rectangular plate sweeping their respective areas. The right line that contains the right side should be \( \omega_1 \), and the top line that contains the
horizontal side should be $\omega_2$. The centered circular perimeter of the hole should be $\omega_3$, and it generates the negative area of the hole. According to Rvachev functions’ theory, we can construct the omega ($\omega$) function using the Boolean algebra, and math analysis:

$$\omega_1 = \frac{a^2-x^2}{2a} \geq 0 \quad \omega_2 = \frac{b^2-y^2}{2b} \geq 0 \quad \omega_3 = -\frac{(r^2-x^2-y^2)}{2r} \geq 0$$

Then, the omega function of the whole domain should be the intersection of these omegas: $\omega_1 \cap \omega_2 \cap \omega_3$

$$\omega_{12} = \omega_1 \cap \omega_2 = \omega_1 + \omega_2 - \sqrt{\omega_1^2 + \omega_2^2}$$

$$\omega_{123} = \omega_{12} \cap \omega_3 = \omega_{12} + \omega_3 - \sqrt{\omega_{12}^2 + \omega_3^2}$$

As you can see, $\omega > 0$ inside domain, $\omega = 0$ on the boundary, and $\omega < 0$ outside the boundary.

Then, the function $\varphi$ can be constructed using distances and interpolating $\varphi$ prescribed values $\varphi_1$ and $\varphi_2$ (boundary conditions). Using transfinite interpolation the function $\varphi$ is constructed: 

$$\varphi = \frac{\omega_{12} \varphi_2 + \omega_3 \varphi_1}{\omega_{12} + \omega_3} [9, 10]$$

2.2.2. Examples of solution structures for heat conduction, stress analysis, and plate problems.

2.2.2.1. Heat conduction solution structures

The steady state equation for heat conduction is $\nabla^2 T = 0$ and the boundary condition: $T|_{\partial \Omega_1} = \varphi_1$, $T|_{\partial \Omega_2} = \varphi_2$

Applying SSM, the structure of the solution of this equation is $T = \omega \varphi + \varphi$, where the first part corresponds to the homogeneous boundary conditions, and the second part for non-homogeneous. Applying transfinite interpolation, we can find $\varphi = \frac{\varphi_1 \omega_2 + \varphi_2 \omega_1}{\omega_1 + \omega_2}$
On the other hand, $\phi = \sum_{j=1}^{n} C_j \chi_j$, it is a linear combination of basis function $\chi_j$, and $C_j$ corresponds to the coefficients that we need to find for solving the problem.

Applying the SSM method, we have:

$$T = T_0 + T_1 = \sum_{j=1}^{n} C_j (\omega \chi_j) + T_1,$$

the heat conduction equation can be expressed:

$$\nabla^2 T = \nabla^2 (T_0 + T_1) = \nabla^2 T_0 + \nabla^2 T_1,$$

applying Garlekin method and the divergence theorem, we obtained:

$$\sum_{j=1}^{n} C_j \int_{\Omega} \nabla (\omega \chi_i) \cdot \nabla (\omega \chi_j) d\Omega = - \int_{\Omega} \nabla (\omega \chi_i) \cdot \nabla (\phi) d\Omega,$$

then we have a vector equation, $CA = B$, where $A = a_{ij} = \int_{\Omega} \nabla (\omega \chi_i) \cdot \nabla (\omega \chi_j) d\Omega$: conductivity matrix, $B = b_i = - \int_{\Omega} \nabla (\omega \chi_i) \cdot \nabla (\varphi) d\Omega$: load vector, and $C = C_j$, unknown coefficients.

I implemented the C++ code called step 1 on the final C++ code for heat conduction problem.

2.2.2.2. Stress analysis solution structures

The next step consists on construct $\omega$ and then finds the displacements equations. We constructed $\omega$ with Rvachev method using the geometry and distances. Displacement equations are:

$$u = \omega_{uv} \phi = \sum_{i=1}^{n} C_i^u (\omega_{uv} \chi_i) \quad v = \omega_{uv} \phi = \sum_{i=1}^{n} C_i^v (\omega_{uv} \chi_i).$$

On the other hand, $\phi$ is a linear combination of basis functions: $\phi_{uv} = C_{uv} \chi_i$

Then, the variational method theory will be applied. I implemented the C++ code called step 2 on the final C++ code for stress analysis problem.

2.2.2.3. Plate problem solution structures

Given a rectangular plate ($\Omega_1$) with a center hole ($\Omega_2$). It is clamped on $\Omega_1$ and free on $\Omega_2$. The function $W$ represents the deflection of the plate.
W on boundary $\Omega_1$ is zero, and W on boundary $\Omega_2$ is zero. Besides $\frac{\partial W}{\partial n} \mid_{\partial \Omega_1} = 0$ Then, the solution structure should be $W = \omega_1^2 \phi$.

On the other hand, $\varphi = 0$ because the boundary conditions are homogeneous, and $\phi = \sum_{i=1}^{n} c_i \chi_i$ as a linear combination of basis functions $\chi_i$

2.3. Variational formulations

The variational principle states that if a differentiable functional $F$ attains its minimum at some point $P$, then $F(P) = 0$. This is a valuable tool for studying PDE [13].

2.3.1. Heat conduction

Applying the MSS method, we obtained: $\nabla^2 T = \nabla^2 (T_0 + T_1) = \nabla^2 T_0 + \nabla^2 T_1$.

Now integrating this equation, using the Garlekim method: $\int_{\Omega} (A_u - f) \eta d\Omega = \int_{\Omega} (\nabla^2 T_0 + \nabla^2 T_1) \omega \chi_i d\Omega = 0$, we obtained $\int_{\Omega} \nabla^2 T_0 \omega \chi_i d\Omega = \int_{\Omega} \nabla^2 T_1 \omega \chi_i d\Omega = 0$

Applying the Divergence Theorem: $\int_{\Omega} \nabla \nabla V d\Omega = \int_{s} \frac{\partial V}{\partial n} ds - \int_{\Omega} \nabla U \nabla V d\Omega$

First part $\int_{\Omega} \nabla^2 T_0 \omega \chi_i d\Omega = - \int_{\Omega} \nabla (\omega \chi_i) \nabla T_0 d\Omega$

Second part $\int_{\Omega} \nabla^2 T_1 \omega \chi_i d\Omega = - \int_{\Omega} \nabla (\omega \chi_i) \nabla T_1 d\Omega$

Meeting first and second part $\int_{\Omega} \nabla (\omega \chi_i) \nabla T_0 d\Omega = - \int_{\Omega} \nabla (\omega \chi_i) \nabla T_1 d\Omega$

According to the solution structure we have $T_0 = \sum_{j=1}^{n} c_j (\omega \chi_j)$, $j = 1 \ldots n$

Then we have $- \sum_{j=1}^{n} c_j \int_{\Omega} \nabla (\omega \chi_j) \nabla (\omega \chi_i) d\Omega = \int_{\Omega} \nabla T_1 \nabla (\omega \chi_i)$

This is: $a_{ij} c_j = b_i$, find $c_j$, and then solve $T = \sum_{j=1}^{n} c_j (\omega \chi_j) + T_1$

2.3.2. Stress analysis
Stress analysis is the second problem of this multi-physics analysis. Using first the energy method, we can obtain the potential energy equation, then applying the variational method theory the equation of the potential energy $\pi$ is minimized respect to $C_i$: $\frac{\partial \pi}{\partial C_i} = 0$, and $\frac{\partial \pi}{\partial C_i} = 0$, and the stiffness matrix ($a_{ij}$) and load vector ($b_j$) were obtained [14].

$$a_{ij} = \int_{\Omega} \frac{Eh}{1-\nu^2} \left[ \frac{\partial}{\partial x} (\omega_u X_j) + \frac{\partial}{\partial y} (\omega_v X_j) \right] \left[ \frac{\partial}{\partial x} (\omega_u X_i) + \frac{\partial}{\partial y} (\omega_v X_i) \right] d\Omega + Gh \left[ \frac{\partial}{\partial y} (\omega_u X_j) + \frac{\partial}{\partial x} (\omega_v X_j) \right] \left[ \frac{\partial}{\partial y} (\omega_u X_i) + \frac{\partial}{\partial x} (\omega_v X_i) \right] - 4\left( \frac{\partial}{\partial x} (\omega_u X_i) \frac{\partial}{\partial y} (\omega_v X_j) + \frac{\partial}{\partial y} (\omega_u X_i) \frac{\partial}{\partial x} (\omega_v X_j) \right] d\Omega$$

$$b_i = \int_{\Omega} \frac{EaT_{diff}}{1-\nu} \left( \frac{\partial}{\partial x} (\omega_u X_i) + \frac{\partial}{\partial y} (\omega_v X_i) \right) d\Omega$$

Now, we can solve the vector equation $AC = B$, and obtain the values for $C = \left( \begin{array}{c} c_u \\ c_v \end{array} \right)$, this is, both $C$ values for each displacement variable.

I implemented the C++ code called step 2 on the final C++ code for stress analysis problem.

2.3.3. Plate buckling

From the principle of virtual work, the principle of stationary potential energy for the buckling problem is obtained:

$$\pi = \frac{1}{2} \int D \left\{ \left( \frac{\partial^2 W}{\partial x^2} + \frac{\partial^2 W}{\partial y^2} \right)^2 + 2(1-\nu) \left[ \left( \frac{\partial^2 W}{\partial x^2} \right)^2 - \frac{\partial^2 W}{\partial x^2} \frac{\partial^2 W}{\partial y^2} \right] \right\} dxdy + \frac{1}{2} K \int \left[ \frac{N_{xx}^0 \left( \frac{\partial W}{\partial x} \right)^2}{\frac{\partial^2 \psi}{\partial x^2}} + \frac{N_{yy}^0 \left( \frac{\partial W}{\partial y} \right)^2}{\frac{\partial^2 \psi}{\partial y^2}} + 2 N_{xy}^0 \frac{\partial W}{\partial x} \frac{\partial W}{\partial y} \right] dxdy,$$

and the independent function subject to variation is $W$ (plate deformation), and $K$ is obtained from the quotient defined by:

$$K = \frac{\frac{1}{2} \int_{\Omega} D \left( \frac{\partial^2 W}{\partial x^2} \frac{\partial^2 W}{\partial y^2} \right)^2 + 2(1-\nu) \left( \frac{\partial^2 W}{\partial x^2} \frac{\partial^2 W}{\partial x^2} \frac{\partial^2 W}{\partial y^2} \right) dxdy}{\frac{1}{2} \int_{\Omega} \left[ \frac{\partial W}{\partial x} \frac{\partial W}{\partial x} + \frac{\partial W}{\partial y} \frac{\partial W}{\partial y} \right] + 2 N_{xy}^0 \frac{\partial W}{\partial x} \frac{\partial W}{\partial y} dxdy},$$
where the bending rigidity is \( D = \frac{Eh^3}{12(1-\nu^2)} \) [14]

From this equation, we obtained the K values, and minimizing respect to \( C_i \) we can obtain the stiffness matrix \((a_{ij})\) and the load matrix \((b_{ij})\):

\[
a_{ij} = \int \int_D \left( \frac{\partial^2 (\omega_1 \omega_2^2 \chi_j)}{\partial x^2} + \frac{\partial^2 (\omega_1 \omega_2^2 \chi_i)}{\partial y^2} \right) \left( \frac{\partial^2 (\omega_1 \omega_2^2 \chi_j)}{\partial x^2} + \frac{\partial^2 (\omega_1 \omega_2^2 \chi_i)}{\partial y^2} \right) dxdy + \\
2(1-\nu) \int \int_D \left[ 2 \left( \frac{\partial (\omega_1 \omega_2^2 \chi_j)}{\partial x} \frac{\partial (\omega_1 \omega_2^2 \chi_i)}{\partial y} \right) dxdy \right]
\]

\[
b_{ij} = \int \int_D \left[ \frac{N_x^0}{\sigma} \frac{\partial}{\partial x} (\omega_1 \omega_2^2 \chi_i)(\omega_1 \omega_2^2 \chi_j) + \frac{N_y^0}{\sigma} \frac{\partial}{\partial y} (\omega_1 \omega_2^2 \chi_i)(\omega_1 \omega_2^2 \chi_j) \right] dx dy + \\
\left[ \frac{N_{xy}^0}{\tau} \frac{\partial}{\partial x} (\omega_1 \omega_2^2 \chi_i) \frac{\partial}{\partial y} (\omega_1 \omega_2^2 \chi_i) \right] dx dy
\]

Where,

\( N_x^0 = h\sigma_x^0, \ N_y^0 = h\sigma_y^0, \ N_{xy}^0 = h\tau_{xy}^0 \). Moreover, h is the plate thickness, \( \sigma \) is the normal stress, and D is the bending rigidity. I implemented the C++ code called step 3 on the final C++ code for plate buckling problem.

3. NUMERICAL EXPERIMENTS

3.1. Experiments set up

This multi-physics analysis has three different linked problems to solve, heat transfer, thermal stress and buckling problems. Each problem needs to be solved, and its solutions are used to solve the other two problems. This is the heat transfer problem solution data is used to solve the stress analysis because of the thermal load. In addition, the thermal and stress solution data are used to solve the buckling problem. Every problem has specific conditions, but the plate geometry and material are the same. We have aluminum 1060 rectangular plate of 2x2 meters with a circular hole in the center,
and its thickness is 0.05 meters. Other properties of this material are \( \nu = 0.33 \), \( E = 200e9 \) \( \text{Pa} \), \( h = 0.05 \) m, \( \alpha = 1.2e-5 \) \( \text{m}^2/\text{m}^\circ\text{C} \). The problem has two boundaries, the rectangle perimeter \( \partial \Omega_1 \), and the central circle hole perimeter \( \partial \Omega_2 \). Temperature unit is Celsius degree. As you can see, we used the International System of units.

The boundary conditions of the stress analysis problem correspond to the constrains on displacements \( U \) and \( V \). In this case, the plate is simple supported on both vertical sides, and the functions \( \omega_u \) and, \( \omega_v \) are the omega functions according to the displacement direction. Where \( U \) direction is according to the X-axis and \( V \) direction is according to Y-axis. \( \Omega_1 \) and \( \Omega_2 \) are the rectangle and the circle domains. \( T|_{\partial \Omega_1} \) and \( T|_{\partial \Omega_2} \).
are the boundary conditions for the thermal problem. On the stress analysis problem, the boundary conditions are given constraining the left and right sides of the plate.

In addition, the buckling problem boundary conditions are related to the kind of support of the plate, in this case it is free on the center, and clamped on the perimeter:

**Figure 2 Stress analysis problem boundary conditions**
3.2. Mesh-free method application

The distance field representation is based on the omega function (\( \omega \)) construction. The procedure to construct \( \omega \) consists of dividing the entire domain in its simple areas and finding the omega function for every single area. Then, using the Boolean algebra, we can obtain the omega function for the entire domain. This domain has three single areas. The vertical line \( \omega_1 \) generates the first area sweeping left to right; the horizontal line \( \omega_2 \) generates the second area sweeping from top to bottom. The center hole perimeter generates the third area, this area is negative, this is, it is the solid area around the hole. Then, the omega function of the whole domain should be the intersection of these omegas: \( \omega_1 \cap \omega_2 \cap \omega_3 \).

Every omega function is zero in the boundary, negative outside the domain, and positive inside the domain.
Figure 4 Particular omega functions
Figure 5 Distance field construction

According to Rvachev method:

\[
\omega_1 = \frac{a^2 - x^2}{2a} \geq 0 \quad \omega_2 = \frac{b^2 - y^2}{2b} \geq 0 \quad \omega_3 = -\frac{(r^2 - x^2 - y^2)}{2r} \geq 0
\]

In this particular case, \(a = b = 1\); if \(X = 1\), then \(\omega_1 = 0\). This is a point on the boundary. If \(X = 1.1\), then \(\omega_1 < 0\). This is a point outside the domain. If \(X = 0.9\), then \(\omega_1 > 0\). This is a point inside the domain.
3.3. Experiment 1

Given the grid of basis function, \( n_{xy}[0] \) calculate: H, maximum temperature, Maximum stress, maximum displacement, and K. These values are obtained by running the C++ code, and they are in the next table. I plotted the graphs to find the convergence study: Max (temp) vs H, Max (UV) vs H, Max (s-mises) vs H, and K vs H, where temperature, UV, s-mises, and K are respectively, temperature, total displacement, maximum stress, and increasing buckling factor. Besides, \( n_{xy}[0] \) is the grid basis function, and H is given by \( H = \frac{x_{\text{max}} - x_{\text{min}}}{n_{xy}[0] - 1} \). In other words, H is the cell mesh size.

Although this experiment’s goal is the convergence study, we can see the influence of the cell mesh size on the results. We found the stability range between 0.05 and 0.105 values of H. The curve K vs H shows the range of stability clearly. Besides, we can see the K variation is very small and the H variation descends according to the mesh basis function.

On the other hand, the Von Misses stress variation ascends when H is descending. Total displacement variation is very stable, and the major deviation is outside the stable values when its value is too small. Temperature is practically constant, but the curve shows the convergence range.

<table>
<thead>
<tr>
<th>( n_{xy}[0] )</th>
<th>H</th>
<th>Temp</th>
<th>Max(uv)</th>
<th>Max(s_mises)</th>
<th>K</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.222667</td>
<td>99.924</td>
<td>4.27E-04</td>
<td>3.01E+08</td>
<td>6.36773</td>
</tr>
<tr>
<td>20</td>
<td>0.105474</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.16E+08</td>
<td>6.35338</td>
</tr>
<tr>
<td>30</td>
<td>0.069103</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.17E+08</td>
<td>6.35292</td>
</tr>
<tr>
<td>40</td>
<td>0.051385</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.20E+08</td>
<td>6.35243</td>
</tr>
<tr>
<td>50</td>
<td>0.040898</td>
<td>99.924</td>
<td>4.29E-04</td>
<td>3.20E+08</td>
<td>6.35239</td>
</tr>
<tr>
<td>60</td>
<td>0.033966</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.20E+08</td>
<td>6.35210</td>
</tr>
<tr>
<td>70</td>
<td>0.029044</td>
<td>99.924</td>
<td>4.29E-04</td>
<td>3.39E+08</td>
<td>6.35211</td>
</tr>
<tr>
<td>80</td>
<td>0.025367</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.20E+08</td>
<td>6.35233</td>
</tr>
<tr>
<td>90</td>
<td>0.022517</td>
<td>99.923</td>
<td>4.29E-04</td>
<td>3.20E+08</td>
<td>6.35227</td>
</tr>
<tr>
<td>100</td>
<td>0.020242</td>
<td>99.922</td>
<td>4.29E-04</td>
<td>3.21E+08</td>
<td>6.35228</td>
</tr>
</tbody>
</table>
Figure 6 K vs H

Figure 7 Max (S-misses) vs H
Figure 8 Max (UV) vs H

Figure 9 Maxtemp vs H
The values for H and R remain constants, and we changed $n_{xy}[0]$. According to the graphs, we can find the stable values in the $n_{xy}$ range 20 to 40. This stability can be observed on every graph around the H range 0.05 to 0.105. Besides, the increasing buckling factor variation is very small.

The next figures show simulation results for $n_{xy}[0] = 10$ and 100. The first figure corresponds to the minimum value of $n_{xy}$ and the second figure corresponds to the maximum one. The first couple of graphs show temperature distribution, and it remains almost constant, even changing $n_{xy}$. The second couple of graphs show the total displacement, and the distribution looks similar, but the maximum displacement on the second graph is slightly bigger than the other one where $n_{xy}$ is 10. The third group of graphs shows the buckling shape deformation, but they are different on peripheral shape where the maximum deformation is bigger than the other one with $n_{xy}$ equal to 10. We can see in experiment 2 how the stress and buckling deformation values are stables in the $n_{xy}$ range 20 to 40.

In conclusion, we took the extreme values of $n_{xy}$ to the convergence study in order to get the range of stable values. This stable range is 20 to 40. The next two experiments are performed inside this range, and the value taken is $n_{xy} = 30$. The other important variable manipulated here is H. This variable depends on the $n_{xy}$ and we saw that its range of convergence is 0.05 to 1.05. This shows us the importance of the mesh size on mesh simulation methods and it took special attention when the C++ code was constructed. We can find this at the first part of the C++ code set up values.
Figure 10 Temperature for $n_{xy} = 10$

Figure 11 Temperature for $n_{xy} = 100$
Figure 12 Total Displacement for $n_{xy} = 10$

Figure 13 Total Displacement for $n_{xy} = 100$
Figure 14 Buckling shape for $n_{xy} = 10$

Figure 15 Buckling shape for $n_{xy} = 100$
3.4. Experiment 2

Now, we are going to establish how the buckling load factor depends on the temperature difference \( T_{diff} = f_2 - 20 \), where 20 is the reference temperature and corresponds to the rectangle plate periphery temperature \( f_2 = T|_{\partial \Omega_2} \) or inner boundary condition of the plate domain. In this case, \( n_{xy}[0] = 30 \) should be the best stable value in the \( n_{xy} \) range 20 to 40 of the convergence study on experiment 1. Besides, \( f_2 = T|_{\partial \Omega_2} \), is the temperature on the center hole periphery, and we entered different values to get the results on the table and then plot the graphs \( K \) vs \( T_{diff} \), \( \text{Max}(s\text{-misses}) \) vs \( T_{diff} \), and \( \text{Max}(UV) \) vs \( T_{diff} \):

<table>
<thead>
<tr>
<th>( n_{xy}[0] )</th>
<th>( f_2 )</th>
<th>( T_{diff} )</th>
<th>( \text{Max}(uv) )</th>
<th>( \text{Max}(s\text{-mises}) )</th>
<th>( K )</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>25</td>
<td>5</td>
<td>2.68E-05</td>
<td>1.98E+07</td>
<td>101.647</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>10</td>
<td>5.36E-05</td>
<td>3.97E+07</td>
<td>50.8233</td>
</tr>
<tr>
<td>30</td>
<td>40</td>
<td>20</td>
<td>1.07E-04</td>
<td>7.93E+07</td>
<td>25.4117</td>
</tr>
<tr>
<td>30</td>
<td>50</td>
<td>30</td>
<td>1.61E-04</td>
<td>1.19E+08</td>
<td>16.9411</td>
</tr>
<tr>
<td>30</td>
<td>60</td>
<td>40</td>
<td>2.14E-04</td>
<td>1.59E+08</td>
<td>12.7058</td>
</tr>
<tr>
<td>30</td>
<td>70</td>
<td>50</td>
<td>2.68E-04</td>
<td>1.98E+08</td>
<td>10.1647</td>
</tr>
<tr>
<td>30</td>
<td>80</td>
<td>60</td>
<td>3.22E-04</td>
<td>2.38E+08</td>
<td>8.47055</td>
</tr>
<tr>
<td>30</td>
<td>90</td>
<td>70</td>
<td>3.75E-04</td>
<td>2.78E+08</td>
<td>7.26047</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>80</td>
<td>4.29E-04</td>
<td>3.17E+08</td>
<td>6.35292</td>
</tr>
</tbody>
</table>

In figure 16, \( K \) vs \( T_{diff} \) when temperature difference is increasing, the buckling factor \( K \) is decreasing. This is logical because the internal loads of the plate increase when temperature is going up, and \( K \) depends on these internal loads. Besides, on the figures \( \text{Max}(UV) \) vs \( T_{diff} \) and \( ax(s\text{-misses}) \) vs \( T_{diff} \), when temperature increases, the other variables increase.
Figure 16 K vs Tdiff

Figure 17 Max (S-misses) vs Tdiff
All variables are very sensitive to the maximum temperature difference change. When Tdiff increases, K decreases, stress increases, and total deformation increases. The next simulation figures show results for the first and last value of the Tdiff. The first figures of every couple of simulations correspond to $T_2 = 25$, and the second figures correspond to $T_2 = 100$. The first couple of figures show the temperature distribution according to the entries indicated above. The other two graphs show total deformation UV, where the first figure is bigger than the other one. This is, total deformation with temperature difference. The third group of figures corresponds to buckling deformation, and they show how the deformation increases when temperature also increases.
Figure 19 Temperature for $T_{diff} = 5$

Figure 20 Temperature for $T_{diff} = 80$
Figure 21 Total Displacement for $T_{diff} = 5$

Figure 22 Total Displacement for $T_{diff} = 80$
Figure 23 Buckling shape for $T_{diff} = 5$

Figure 24 Buckling shape for $T_{diff} = 80$
3.5. Experiment 3

The third experiment consists of how the load factor depends on the radius of the hole. Different radius values are given, $f_2 = 100$ and $n_{xy} = 30$. Results and graphs $K$ vs $R$, Max (UV) vs $R$, and Max (s-misses) vs $R$ are:

<table>
<thead>
<tr>
<th>nxy[0]</th>
<th>f2</th>
<th>R</th>
<th>Max(uv)</th>
<th>Max(s_mises)</th>
<th>k</th>
</tr>
</thead>
<tbody>
<tr>
<td>30</td>
<td>100</td>
<td>0.05</td>
<td>2.27E-04</td>
<td>2.07E+08</td>
<td>10.6213</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.1</td>
<td>2.86E-04</td>
<td>2.60E+08</td>
<td>8.30598</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.15</td>
<td>3.37E-04</td>
<td>2.89E+08</td>
<td>7.16045</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.2</td>
<td>3.84E-04</td>
<td>3.03E+08</td>
<td>6.5807</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.25</td>
<td>4.29E-04</td>
<td>3.17E+08</td>
<td>6.35292</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.3</td>
<td>4.72E-04</td>
<td>3.16E+08</td>
<td>6.35594</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.35</td>
<td>5.13E-04</td>
<td>3.20E+08</td>
<td>6.52007</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.4</td>
<td>5.52E-04</td>
<td>3.22E+08</td>
<td>6.77658</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.45</td>
<td>5.90E-04</td>
<td>3.23E+08</td>
<td>7.06362</td>
</tr>
<tr>
<td>30</td>
<td>100</td>
<td>0.5</td>
<td>6.27E-04</td>
<td>3.23E+08</td>
<td>7.33307</td>
</tr>
</tbody>
</table>

**Figure 25 K vs R**
In order to compare this project with commercial software, I used Ansys. The next figures show the result for both this project and Ansys software.

**Figure 26 Ansys K vs R**

![Graph showing K vs R comparison](image)

**Figure 27 Max (S-misses) vs R**

![Graph showing S-misses vs R comparison](image)
Figure 28 Ansys Max (S-misses) vs R

Figure 29 Max (UV) vs R
The dependent variables K, Von misses stress, and UV total displacements are very sensitive to radius changes. On K vs R, the first part of curve is descendent, and the second part is ascendant. On stresses curves, when the radius increases, stresses increase. The next figures of simulation show results for the first and last value of the radius. The first figures correspond to the low value of the radius, and the second side corresponds to the bigger value of the radius. The first figure shows the temperature distribution according to the two values of the radius. The second figure shows the stress increasing when the radius also increases. The couple of figures show the greater deformation around the center hole. When the hole radius increases to the maximum value, the greater deformation is on lateral boundaries. We can see the substantial influence of the geometry variation over the total deformation and stresses.
Figure 31 Temperature for $R = 0.05$

Figure 32 Ansys Temperature for $R = 0.05$
Figure 33 Temperature for R = 0.5

Figure 34 Ansys Temperature for R = 0.5
Figure 35 Total Displacement for R = 0.05

Figure 36 Ansys Total Displacement for R = 0.05
Figure 37 Total Displacement for R = 0.5

Figure 38 Ansys Total Displacement for R = 0.5
Figure 39 Buckling Shape for $R = 0.05$

Figure 40 Ansys Buckling Shape for $R = 0.05$
Figure 41 Buckling Shape for R = 0.5

Figure 42 Ansys Buckling Shape for R = 0.5
4. CONCLUSIONS

In the first experiment, the convergence study, we observed that all variables tend to increase when the grid size decreases, but K tends to remain constant. The stable values on the curves are between the 10 to 40 range of $n_{xy}[0]$, mainly, between 20 to 40. Besides, K variation with respect to the mesh cell size is very small. Finally, the best value for the grid of basis functions according to the convergence experiment should be $n_{xy} = 30$. For the other experiments, we used this value.

In the second experiment, all variables show great dependency from the temperature difference, for example, K decreases when Tdiff increases, total displacement increases, and von misses stress also increases. This behavior is logical because when the temperature increases, the inner loads of the plate also increase, and for this reason total displacement and stress increase.

In the third experiment, all variables show a great dependency of the radius R. For example, K decreases and the middle of the values have the minimum, and then it increases. When the radius increases, stresses also increase. When the plate center hole radius increases, the shape deformation tends to increase on the lateral sides of the plate.

The process of modeling a heat-induced buckling of plates using a mesh-free method has been successful, and it has been implemented on the C++ code exposed on the appendix.

The comparison with the commercial software Ansys shows very close results, according to the graphs and simulations. The kind of dependence of the variables is the same for both softwares, K vs R, UV vs R. In the case of Stress vs R, there is a variation
in the first three values of stress, but the rest of the curve is similar for simulations in this project, and Ansys. This variation could be caused by the kind of mesh used by Ansys.

Finally, I developed numerical algorithms and software tools based on the mesh-free Solution Structure Method to model thermally induced buckling on thin plates, and I investigated the convergence of the numerical solution. In addition, I investigated the dependence of the buckling load factor on the temperature difference and geometric parameters.
REFERENCES


APPENDIX

#include "rfms.h"
#include <fstream>
#include <iostream>
#include <cmath>
using namespace std;

// declare functions for step 1
    tuple omega1(void);
    double om1(void);
    int PMC1(void);
    tuple u11(void);
    void u10(void);
    double fu1(void);
    double f_exact1(void);
    double error1(void);
    double f_exact12(void);
    double error12(void);
    double fa1(int flag, int i, int j );
    double fb1(int flag, int i );
    // end step 1 functions

// declare functions for step 2
    int pre2(void);
    tuple omega2(void);
    double fa2(int flag, int block_i, int i, int block_j, int j);
    double fb2(int flag, int block_i, int i);
    double fu2(void);
    double fv2(void);
    double om2(void);
    double om_fixed2(void);
    double s_mises_p_stress2(void);
    void u21(void);
    tuple u20(void);
    double sx_p_stress2(void);
    double sy_p_stress2(void);
    double txy_p_stress2(void);
    // end step 2 functions

// declare functions for step 3
    tuple u31(void);
    void u30(void);
    double fu3(void);
    double fa3(int flag, int i, int j);
    double fb3(int flag, int i, int j);
    // end step 3 functions

    //double max_temp, max_uv, max_s_miss;

    // Global variables
tuple *w1_h, *w2_h;
tuple *w1_s, *w2_s;
tuple_seq *p1;
vectorclass *Ci1;
tuple_seq *p3;
vectorclass *Ci3;
vectorclass *cu2, *cv2;
double D, K;
double E_nu, G;
double q_x =  1e6;
double q_y = -1e6;
//const double nu = 0.33, E = 69e9, h = 0.25, alpha = 2.18e-5; // 1060 Aluminium alloy
const double nu = 0.3, E = 200e9, h = 0.05, alpha = 1.2e-5; // 1060 Aluminium alloy
const double T_reference = 20.0;
const int n = 200;
#define PLANE_STRESS    // To change to plain strain formulation, comment this line

int main (void)
{
    SetTupleDimensionOrder(2,2);    // set the dimensions of computational system
    w1_h = new tuple;    // allocate memory for tuple
    w2_h = new tuple;    // allocate memory for tuple
    w1_s = new tuple;    // allocate memory for tuple
    w2_s = new tuple;    // allocate memory for tuple

    // define some variables
double xmin, ymin, xmax, ymax;
double xmin_ext, ymin_ext, xmax_ext, ymax_ext;
int ns, nxy[2];
int n_gauss, n_random;
MatrixIndexType *mapping_array;
MatrixIndexType size_compressed;

    // Analytical geometry boundary values
    xmin = -1.0;    // Specify the Bounding box
    ymin = -1.0;
    xmax =  1.0;
    ymax =  1.0;
    ns = 3;    // degree of B-spline basis

    //cout << "enter value for nxy: ";
    //cin >> nxy[0];
    nxy[0] = nxy[1] = 30;    // grid of basis functions
    n_gauss =  4;    // number of integration points
n_random = 5;  // number of random points to detect small features

// ofstream outFile;
// if (nxy[0] == 10)
// outFile.open ("Results.csv", ofstream::out);
// else
// outFile.open ("Results.csv", ofstream::app);
// outFileprecision (9);
// outFile << nxy[0];
// max_temp = max_uv = max_s_miss = 0;

// E_nu and G variables for step 2
#ifdef PLANE_STRESS
  E_nu = E/(1.0-nu*nu);
#else
  E_nu = E*nu/(2.0*(1.0+nu)*(1.0-2.0*nu));
#endif
G = E/((1.0+nu)*2.0);
// D and K variables for step 3
D = E * h * h * h / (12.0 * (1.0-nu*nu));

cout << " Humberto Mejia - 2013 Masters' Thesis Program" << endl;
cout << " Program to simulate buckling of a thin plate using mesh-free analysis." << endl;
cout << " Advisor: Dr. Tsukanov, Dept. of Mech. & Materials Eng., \n\tFlorida International Univ, Miami" << endl;
cout << endl << " Initializing geometry structure.. " << endl;
timeclass Tim;
Tim.ShowTimeBegin();
// Set the bounding box
SetBoundingBox2D(xmin, ymin, xmax, ymax);

double hhh = (xmax_ext - xmin_ext)/(nxy[0]-1);
cout << " hhh = " << hhh << endl;
// outFile << " hhh = " << hhh;

// Declare the matrix and vector
matrixclass a;
vectorclass b; Ci1 = &b;

// Declare and specify the basis functions
tuple_seq P(1); p1 = &P;
P.SetBsplines2DDegree(0, ns, nxy[0], nxy[1]);

// Initialize a data structure for 2D integration
quadtreeclass tree(nxy[0], nxy[1], xmin_ext, ymin_ext, xmax_ext, ymax_ext);
tree.BuildTree(n_gauss, 0.5, n_random, PMC1);
tree.Print("tree.plt", n);
filexy("geometry.3m", n,n, xmin, ymim, xmax, ymax, om1, om1);
cout << " Done setting up geometry.. " << endl;
Tim.ShowTimeEnd ();
cout << "--------- Starting Step 1 ----------" << endl;
// Allocate memory for the matrix and vector
a.CreateSparseMatrix( FM_MATRIX_NON_SYMMETRIC, // Non-symmetric
storage is used
2, // Dimension of space
ns, // Degree of B-splines
nxy, // Grid
1, // Number of blocks
);

b.CreateSparseVector( 2, // Dimension of space
ns, // Degree of B-
splines
nxy, // Grid
1, // Number of blocks); // Size of the compressed vector

// Specify which matrix and vector to be assembled
SetMatrixPointer( &a );
SetVectorPointer( &b );

// Matrix and vector assembly via 2D geometrically adaptive integration
tree.IntegrateMV( fa1, fb1 );

// Solve a linear system of equations for DOFs
a.Solve( b ); // b will contain the solution

// Save solution for its visualization
filexy("temperature", n, n, xmin, ymin, xmax, ymax, fu1, om1);
// outFile << "," << max_temp;
a.DestroyMatrix();

Tim.ShowTimeEnd ();
cout << "--------- Step 1 Completed ----------" << endl;

/***********************************************
* STEP 1 COMPLETE START STEP 2
***********************************************/
cout << "--------- Starting Step 2 ----------" << endl;
Tim.ShowTimeBegin ();
//SetTupleDimensionOrder(2,1);
#ifdef PLANE_STRESS
printf("PLANE STRESS formulation\n", n_gauss);
#else
printf("PLANE STRAIN formulation\n", n_gauss);
#endif

matrixclass a2; // Banded matrix formed by 2 sequences of the basis functions
vectorclass b2,  // Vector formed by 2 sequences of the
c_u,           // Coefficients for "u"
c_v;           // Coefficients for "v"
cu2 = &c_u; cv2 = &c_v;
c_u.CreateSparseVector( 2, ns, nxy, 1);
c_v.CreateSparseVector( 2, ns, nxy, 1);
b2.CreateSparseVector( 2, ns, nxy, 2);
a2.CreateSparseMatrix( FM_MATRIX_NON_SYMMETRIC, 2, ns, nxy, 2);

// Defining information about basis functions
tuple_seq Pu(1), Pv(1), P1(1), P2(1), Pz(1);
Pu.SetBsplines2DDegree(0, ns,nxy[0],nxy[1]); // Basis functions
for "u"
Pv.SetBsplines2DDegree(0, ns,nxy[0],nxy[1]); // Basis functions
for "v"
Pz.SetBsplines2DDegree(0, ns,nxy[0],nxy[1]); // Zero sequence
P1.Link(&Pu, &Pz, NULL); // Sequence of basis functions "u"+"0"
P2.Link(&Pz, &Pv, NULL); // Sequence of basis functions "0"+"v"
Pz.SetZero();            // Initialization to zero
p21 = &P1; p22 = &P2; pu2 = &Pu; pv2 = &Pv; pz2 = &Pz;

// Definition of the active matrix & vector
SetMatrixPointer( &a2 );
SetVectorPointer( &b2 );

// Assembly of the stiffness matrix and load vector due to the thermal load
tree.IntegrateMV(fa2, fb2);

// Solve the algebraic system
a2.Solve( b2 );
c_u = b2.Extract(0); // Extraction of the coefficients for "u".
0 means block #0
c_v = b2.Extract(1); // Extraction of the coefficients for "v".
1 means block #1

#ifdef PLANE_STRESS
filexy("step2_u_plane_stress",n,n, xmin, ymin, xmax, ymax, fu2,
oml1);
filexy("step2_v_plane_stress",n,n, xmin, ymin, xmax, ymax, fv2,
oml1);
filexy_vector( "step2_uv_plane_stress", n,n, xmin, ymin, xmax,
ymax, fu2, fv2, oml1);
filexy("step2_sx_plane_stress",n,n, xmin, ymin, xmax, ymax,
sx_p_stress2, oml1);
filexy("step2_sy_plane_stress",n,n, xmin, ymin, xmax, ymax,
sy_p_stress2, oml1);
filexy("step2_txy_plane_stress",n,n, xmin, ymin, xmax, ymax,
txy_p_stress2, oml1);
#endif
filexy("step2_s_mises_plane_stress", n, xmin, ymin, xmax, ymax, s_mises_p_stress2, om);
#else
filexy("step2_u_plane_strain", n, xmin, ymin, xmax, ymax, fu, om);
filexy("step2_v_plane_strain", n, xmin, ymin, xmax, ymax, fv, om);
filexy_vector("step2_uv_plane_strain", n, xmin, ymin, xmax, ymax, fu, fv, om);
#endif
Tim.ShowTimeEnd();
//outFile << "," << max_uv << "," << max_s_miss;
a2.DestroyMatrix();
cout << "------ Step 2 Completed -------" << endl;
*******************************************************************************
* STEP 2 COMPLETE START STEP 3
*******************************************************************************
cout << "------ Starting Step 3 ------" << endl;
Tim.ShowTimeBegin();
tuple_seq P3(1); p3 = &P3;
P3.SetBsplines2DDegree(0, ns, nxy[0], nxy[1]);
// Solution of a system of linear algebraic equations
matrixclass a31, a32;
a31.CreateSparseMatrix( FM_MATRIX_SYMMETRIC, // Symmetric storage
                      2,                       // Dimension of space
                      ns,                      // Degree of B-splines
                      nxy,                     // Grid
                      1,                       // Number of blocks);
                      // Size of the compressed matrix
a32.CreateSparseMatrix( FM_MATRIX_SYMMETRIC, // Symmetric storage
                      2,                       // Dimension of space
                      ns,                      // Degree of B-splines
                      nxy,                     // Grid
                      1,                       // Number of blocks);
// Specify which matrix and vector to be assembled
SetMatrixPointer(&a31);
tree.IntegrateM(fa3);
SetMatrixPointer(&a32);
tree.IntegrateM(fb3);

// // BUCKLING ///
int vec_length, n_eigs = 1;
int n_converged, I;

vec_length = a31.GetLength();
vectorclass C(vec_length),
    k(n_eigs);
Ci3 = &C;
matrixclass e_vec;
a31.EigenValues( a32, e_vec, k, n_eigs, n_converged );

cout << "Buckling load factor (k) = " << k.GetElement(1) << endl;
// outFile << "," << k.GetElement(1) << endl;
// outFile.close();

C = e_vec.GetEigenVector(1);
filexy("step_3__buckling_shape", n,n, xmin, ymin, xmax, ymax,
    fu3, om1);

Tim.ShowTimeEnd();
cout << "--------- Step 3 Completed " << endl;
DeleteAD();
cin.get();
return 0;

/*****************************/
* OMEGA (GEOMETRY) FUNCTIONS
/*****************************/
//static double pmc;
// Approximate distance field to the domain's boundary (returns tuple)
// omega for step 1 and step 3
tuple omega1(void)
{
    *w1_h = bandx(0,1) & bandy(0,1); // left boundary - fixed
    *w2_h = mcircle (0, 0, 0.5);
    return (*w1_h) & (*w2_h);
}

// Approximate distance field to the domain's boundary (returns tuple)
// omega for step 2
tuple omega2(void)
{
    *w1_s = line(-1,1,-1,-1) & line(1,-1,1,1);
    return (*w1_s) & mcircle (0, 0, 0.5) & bandx(0,1);
}

/*****************************/
* FUNCTIONS FOR STEP 1
/*****************************/
// Approximate distance field to the domain's boundary (returns double)
// om for step 1 and step 3
double om1(void)
// Point Membership Classification function
// PMC for step 1 and step 3
int PMC1(void)
{
    if( om1() >= 0.0 )
        return 1; // return 1 if the point is inside of the domain
    else
        return -1; // return -1 if the point is outside of the domain
}

// Transfinite interpolation of the boundary conditions
tuple u11(void)
{
    tuple f1, f2; // Declare differential tuples
    omega1(); // Compute distance fields
    f1 = 20; f2 = 100; // Specify boundary conditions values
    return paste( *w1_h, *w2_h, f1, f2 ); // (w1*f2 + w2*f1)/(w1 + w2)
}

// Computation of \omega \Phi
void u10(void)
{
    p1->Compute(0); // Compute basis functions \chi_i
    p1->Mult(omega1()); // Multiply all basis functions by \omega
    return;
}

// Computation of the solution
double fu1(void) // temperature distribution
{
    u10(); // Compute basis functions
    return (sum( Ci1, u11(), *p1 ).GetValue()); // u = \sum (Ci \omega \chi_i) + \varphi
    //if (r > max_temp && pmc >= 0.0)
    //    max_temp = r;
    //return r;
}

// Rule to assemble the conductivity matrix
double fa1( int flag, int i, int j )
{
    if( flag == 0 ) // New integration point: compute the basis functions
    {
u10();
    return 0;
  } else {
    return dx_direct( pl->GetTuple(0, i), 1 ) * // derivative of i-th basis function wrt "x"
      dx_direct( pl->GetTuple(0, j), 1 ) + // derivative of j-th basis function wrt "x"
    dx_direct( pl->GetTuple(0, i), 2 ) * // derivative of i-th basis function wrt "y"
      dx_direct( p1->GetTuple(0, j), 2 );   // derivative of j-th basis function wrt "y"
  }
}

// Rule to assemble the load vector
double fb1( int flag, int i )
{
  static tuple u111(GetTupleMaxOrder());

  if( flag == 0 ) { // New integration point: compute the basis functions
    u10(); // New integration point: compute the basis functions
    u111 = u11();
    return 0;
  } else {
    return -(dx_direct( p1->GetTuple(0, i), 1 ) * // derivative of i-th basis function wrt "x"
      dx_direct( &u111, 1 ) + // derivative of the BC function wrt "x"
    dx_direct( p1->GetTuple(0, i), 2 ) * // derivative of i-th basis function wrt "y"
      dx_direct( &u111, 2 )); // derivative of the BC function wrt "y"
  }
}

/*********************************
* FUNCTIONS FOR STEP 2
*********************************/
//Point membership classification function
int pre2(void)
{
  if( om2() >= 0.0)
  {
    //pmc = 1;
    return 1;
  }
else
{
    //pmc = -1;
    return -1;
}
}

//Solution structure
void u21(void)
{
    pu2->B_splines(0);
    pu2->Mult(*w1_s);    // multiply by a distance field to
                        // the boundary fixed in "x"
    direction
    return;
}

void v21(void)
{
    pv2->B_splines(0);
    pv2->Mult(*w1_s);    // multiply by a distance field to
                        // the boundary fixed in "y"
    direction
    return;
}

// Stiffness matrix assembly rule
double fa2(int flag, int block_i, int i, int block_j, int j)
{
    double result1;
    // Compute basis functions at the point
    if( flag == 0 )
    {
        omega2();
        u21(); v21();
    }

#ifdef PLANE_STRESS
    // rule to form stiffness matrix for plane stress problem
    result1 = E_nu *
        ( dx_direct( p21->GetTuple(block_j, j), 1) +
          dx_direct( p22->GetTuple(block_j, j),
                    2))*)
        ( dx_direct( p21->GetTuple(block_i, i), 1) +
          dx_direct( p22->GetTuple(block_i, i), 2))
    + ( dx_direct( p21->GetTuple(block_i, i), 2) +
        dx_direct( p22->GetTuple(block_i, i),
                  1))*
    ( dx_direct( p21->GetTuple(block_j, j), 2) +
      dx_direct( p22->GetTuple(block_j, j),
                 1))* G -
        2.0*( dx_direct( p21->GetTuple(block_i, i), 1) *
             dx_direct( p22->GetTuple(block_j, j), 2) +
             \ldots)}
dx_direct( p21->GetTuple(block_j, j), 1)
* dx_direct( p22->GetTuple(block_i, i), 2))* G;
return result1;
#endif
// rule to form stiffness matrix for plane stress problem
result1  = E_nu *
( dx_direct( p1->GetTuple(block_j,j), 1) +
  dx_direct( p2->GetTuple(block_j,j), 2))*
( dx_direct( p1->GetTuple(block_i,i), 1) +
  dx_direct( p2->GetTuple(block_i,i), 2))
+ ( dx_direct( p1->GetTuple(block_i,i), 1) *
  dx_direct( p1->GetTuple(block_j,j), 1) +
  dx_direct( p2->GetTuple(block_i,i), 2) *
  dx_direct( p2->GetTuple(block_j,j), 2))
G
+ ( dx_direct( p1->GetTuple(block_i,i), 2) +
  dx_direct( p2->GetTuple(block_i,i), 1))*
( dx_direct( p1->GetTuple(block_j,j), 2) +
  dx_direct( p2->GetTuple(block_j,j), 1)) *
G * 0.5;
return result1 * 2.0;
#else
// rule to form the load vector
double fb2(int flag, int block_i, int i)
{
  static double t_difference;
  double result1;
  if( flag == 0 )
  { t_difference = fu1() - T_reference;
    omega2();
    u21(); v21();
  }
  result1 = E * alpha * t_difference /(1.0-nu) * ( dx_direct( p21->GetTuple(block_i,i), 1)+
  dx_direct( p22->GetTuple(block_i,i), 2));
return result1;
}

static double fu2_v;
// SOLUTION
// "x" component of the displacement vector
double fu2(void)
{
  omega2();
u21();
return (sum( cu2, *pu2 ).GetValue());
//return fu2_v;
double fv2(void)
{
    omega2();
    v21();
    return (sum( cv2, *pv2 ).GetValue());
    //if (r * fu2_v > max_uv && pmc >= 0.0)
    // max_uv = r * fu2_v;
    //return r;
}

double om2(void)
{
    //pmc = omega2().GetValue();
    return omega2().GetValue();
}

double sx_p_stress2(void)
{
    static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
    double t_difference = fu1() - T_reference;
    omega2();
    u21(); v21();
    u = sum( cu2, *pu2 );
    v = sum( cv2, *pv2 );
    double ex, ey, gxy, sx;
    ex = dx_direct(&u,1); 
    ey = dx_direct(&v,2);
    gxy = dx_direct(&u,2) + dx_direct(&v,1);
    sx = E/(1.0-nu*nu)*(ex + nu*ey) - E * alpha * t_difference / (1.0-nu);
    return sx;
}

double sy_p_stress2(void)
{
    static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
    double t_difference = fu1() - T_reference;
    omega2();
    u21(); v21();
    u = sum( cu2, *pu2 );
    v = sum( cv2, *pv2 );
    double ex, ey, gxy, sx;
    ex = dx_direct(&u,1); 
    ey = dx_direct(&v,2);
    gxy = dx_direct(&u,2) + dx_direct(&v,1);
    sx = E/(1.0-nu*nu)*(ey + nu*ex) - E * alpha * t_difference / (1.0-nu);
    return sx;
}

double txy_p_stress2(void)
```c
{ static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
  omega2();
  u21(); v21();
  u = sum( cu2, *pu2 );
  v = sum( cv2, *pv2 );
  double gxy, txy;
  gxy= dx_direct(&u,2) + dx_direct(&v,1);
  txy = G*gxy;
  return txy;
}

double s_mises_p_stress2(void)
{
  double S1, S2;
  static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
  double t_difference = fu1() - T_reference;
  omega2();
  u21(); v21();
  u = sum( cu2, *pu2 );
  v = sum( cv2, *pv2 );
  double ex, ey, gxy, sx, sy, txy, ds;
  ex = dx_direct(&u,1);
  ey = dx_direct(&v,2);
  gxy= dx_direct(&u,2) + dx_direct(&v,1);
  sx = E/(1.0-nu*nu)*(ex + nu*ey) - E * alpha * t_difference / (1.0-nu);
  sy = E/(1.0-nu*nu)*(ey + nu*ex) - E * alpha * t_difference / (1.0-nu);
  txy = G*gxy;
  ds = (sx-sy)/2.0;
  S1 = (sx+sy)/2.0+sqrt(ds*ds+txy*txy);
  S2 = (sx+sy)/2.0-sqrt(ds*ds+txy*txy);
  return sqrt(S1*S1 + S2*S2 - S1*S2);
  //if (r > max_s_miss && pmc >= 0.0)
  // max_s_miss = r;
  //return r;
}

/*********************************/
/* FUNCTIONS FOR STEP 3 */
/*********************************/

// Transfinite interpolation of the boundary conditions
tuple u31(void)
{
  tuple f1, f2;                      // Declare differential tuples
  omega1();
  return ((*w1_s) * (*w2_s) * (*w2_s));
}

// Computation of \omega \Phi
void u30(void)
{
  omega2();
}
```
p3->Compute(0); // Compute basis functions \chi_i
p3->Mult(square(*w1_s)); // Multiply all basis functions by
\omega
return;
}

// Computation of the solution
double fu3(void)
{
    u30(); // Compute basis functions
    return sum( Ci3, *p3 ).GetValue(); // u = \sum (Ci \omega \chi_i)
}

// Rule to assemble the conductivity matrix
double fa3( int flag, int i, int j )
{
    if( flag == 0 ) // New integration point: compute the basis functions
    {
        u30();
        return 0;
    }
    else
    {
        return D *( ( dx_direct( p3->GetTuple(0,j), 3) +
                      dx_direct( p3->GetTuple(0,j), 5)) *
            ( dx_direct( p3->GetTuple(0,i), 3) +
              dx_direct( p3->GetTuple(0,i), 5))
              -(1.0 - nu)*
               ( dx_direct( p3->GetTuple(0,i), 3) *
                     dx_direct( p3->GetTuple(0,j), 5) +
                     dx_direct( p3->GetTuple(0,i), 5) *
                     dx_direct( p3->GetTuple(0,j), 3) -
                        2.0 * dx_direct( p3->GetTuple(0,i), 4)
                        * dx_direct( p3->GetTuple(0,j), 4)
               )
        );
    }
}

// Rule to assemble the load vector
double fb3( int flag, int i, int j )
{
    static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
    static double Nx, Ny, Nxy;
    if( flag == 0 )
    {
        static tuple u(GetTupleMaxOrder()), v(GetTupleMaxOrder());
        double t_difference = fu1() - T_reference;
        omega2();
        u21(); v21();
    }
}
\[ u = \text{sum}( \text{cu2}, n \text{pu2} ); \]
\[ v = \text{sum}( \text{cv2}, n \text{pv2} ); \]
\[ \text{double ex, ey, gxy;} \]
\[ \text{ex} = \text{dx\_direct}(&u, 1); \]
\[ \text{ey} = \text{dx\_direct}(&v, 2); \]
\[ \text{gxy} = \text{dx\_direct}(&u, 2) + \text{dx\_direct}(&v, 1); \]
\[ \text{Nx} = h \cdot \frac{E}{1.0 - n\cdot nu} \cdot (\text{ex} + nu\cdot ey) - E \cdot alpha \cdot \text{t\_difference} / (1.0 - nu); \]
\[ \text{Ny} = h \cdot \frac{E}{1.0 - n\cdot nu} \cdot (\text{ey} + nu\cdot ex) - E \cdot alpha \cdot \text{t\_difference} / (1.0 - nu); \]
\[ \text{Nxy} = h \cdot G \cdot gxy; \]
\[ \text{u30();} \]
\]
\[ \text{return } - (\text{Nx} \cdot \text{dx\_direct}( p3->\text{GetTuple}(0,i), 1) \cdot \text{dx\_direct}( p3->\text{GetTuple}(0,j), 1)) + \]
\[ - (\text{Ny} \cdot \text{dx\_direct}( p3->\text{GetTuple}(0,i), 2) \cdot \text{dx\_direct}( p3->\text{GetTuple}(0,j), 2)) + \]
\[ \text{Nxy} \cdot (\text{dx\_direct}( p3->\text{GetTuple}(0,i), 1) \cdot \text{dx\_direct}( p3->\text{GetTuple}(0,j), 2) + \]
\[ \text{dx\_direct}( p3->\text{GetTuple}(0,i), 2)) ; \]