Nonlinear Finite Element Analysis Code

Tyler Ryan
Tyler.Ryan@engineering.ucla.edu

MAE 261B, UCLA
March 20, 2015

Contents

1. INTRODUCTION

This report describes the theory, architecture, implementation, and results of creating a finite element analysis program in the object-oriented programming language of Python (version 3.3.5). A lot of emphasis has been placed on code architecture and verification testing to ensure that the code is easily extensible and robust for future development. In order to get started, we must first understand the equations for and derivations of the fundamental quantities that will lay the foundation for the program. Then we will explore the implementation of these equations into the code architecture and describe some of the key design choices for code structure. Finally, we will discuss the results of implementation by looking at the application of the code to several nonlinear analysis problems.

1.1 Curvilinear Coordinates and Frames

To describe the deformation of an arbitrarily curved body, it is useful to introduce a curvilinear coordinate system that allows us to define a basis in such a way that is natural or convenient for the body. For example, it is easy to describe the deformation of a cylindrical body in cylindrical coordinates, or a spherical body in spherical coordinates. These are idealized examples, but illustrate the point that coordinate axes can be chosen to work well with the geometry of the body undergoing deformation.

In curvilinear coordinates, we refer to the curved coordinate axes as θ^i, where i ranges from 1 to 3 to represent the three axes. These coordinates are used to describe positions in the body, which will ultimately be expressed in the lab frame. The **lab frame** can be thought of as the frame of an observer outside of the body, in which positions are described in terms of Cartesian coordinates x, y, and z, or E_i. For a given body, we will use curvilinear axes θ^i in such a way that we can write expressions for θ^i in terms of E_i, and vice versa.

The curvilinear coordinates are often chose to match the geometry of the body in an **idealized configuration**. For example, if our body has a shape close to that of a sphere, we would use a sphere as the idealized configuration and spherical coordinates as our curvilinear coordinates. We then define two mappings from the idealized configuration: one to the reference configuration and another to the deformed/current configuration. The **reference configuration** represents the initial geometry of the body, prior to deformation, and will be represented by capital letter symbols. The **deformed configuration** represents the geometry of the body at some point in time during deformation, and will be represented with lowercase symbols. This geometry will in general change with time, and thus is often referred to as the current configuration. We can define functions to represent these two mappings in terms of the curvilinear coordinates of the system:

Reference Configuration (Ω_0):

$$\boldsymbol{X} = \boldsymbol{\phi_0}(\theta^i) = f_1(\theta^i)\boldsymbol{e_{\theta_1}} + f_2(\theta^i)\boldsymbol{e_{\theta_2}} + f_3(\theta^i)\boldsymbol{e_{\theta_3}} \tag{1}$$

Deformed Configuration (Ω):

$$\boldsymbol{x} = \boldsymbol{\phi}(\theta^i) = g_1(\theta^i)\boldsymbol{e_{\theta_1}} + g_2(\theta^i)\boldsymbol{e_{\theta_2}} + g_3(\theta^i)\boldsymbol{e_{\theta_3}} \tag{2}$$

1.2 Covariant and Contravariant Basis Vectors

In order to express our reference and deformed configurations, we need to construct bases. Because we are using curvilinear coordinates, we can do this in two ways. The first is to construct the tangent basis vectors, which are tangent to the coordinates axes θ^i. These are referred to as **covariant basis vectors**, and are denoted with a subscript index as g_i. The second is to construct the dual basis vectors, which are normal to the θ^i-surfaces. These surfaces are formed by the plane containing two coordinate axes. For example, the θ^1 surface is the plane containing the θ^2 and θ^3 axes, and the first dual vector will be normal to this surface. These vectors are referred to as **contravariant basis vectors**, and are denoted with a superscript index as g^i. Note that covariant and contravariant basis vector do not in general point in the same direction.

The covariant and contravariant basis vectors are defined as follows (keeping in mind that capital symbols are used for the reference configuration and lowercase symbols are used for the deformed configuration):

$$\boldsymbol{G_i} = \frac{\partial \boldsymbol{\phi_0}}{\partial \theta^i}, \quad \boldsymbol{G^i} = G^{ij}\boldsymbol{G_j}, \quad \boldsymbol{g_i} = \frac{\partial \boldsymbol{\phi}}{\partial \theta^i}, \quad \boldsymbol{g^i} = g^{ij}\boldsymbol{g_j}, \tag{3}$$

where G^{ij} and g^{ij} represent metric tensors, and are described in more detail below. The covariant and contravariant metric tensors are related by the inverse:

$$G^{ij} = [G_{ij}]^{-1}, \quad g^{ij} = [g_{ij}]^{-1} \tag{4}$$

Properties Because each basis is defined based on three curved axes defined by the geometry of the body, the basis will not in general be orthonormal. In other words, the dot product of two basis vectors will not yield the Kronecker Delta, but will instead give a tensor called the **metric tensor**.

$$\boldsymbol{g_i} \cdot \boldsymbol{g_j} = g_{ij} \neq \delta_{ij}, \quad \boldsymbol{g^i} \cdot \boldsymbol{g^j} = g^{ij} \neq \delta_{ij} \tag{5}$$

The elements of the metric tensor g_{ij} describe the length of the tangent vectors (diagonal elements) and the angles between them (off-diagonal elements). Because the bases arises from the curvilinear coordinate axes, it makes sense that the metric tensor does not generally equal the identity matrix. However, the identity matrix is used to describe the relationship between covariant and contravariant basis vectors:

$$\boldsymbol{g^i} \cdot \boldsymbol{g_j} = \delta^i_j \tag{6}$$

1.3 Kinematic Quantities

With the basis vectors defined for both the reference, and deformed configurations, we can now compute the kinematic quantities that describe the deformation of the body.

The **deformation gradient** is computed from the outer product of basis vectors in the two configurations, and represents the manner in which the body deforms at a given point in space. The diagonal elements represent stretching and the off-diagonal elements represent twisting of the body.

$$\boldsymbol{F} = \boldsymbol{g_i} \otimes \boldsymbol{G^i} \tag{7}$$

There are three tensor that describe the strains in the body at a point in space, the **right Cauchy-Green deformation tensor**, the **left Cauchy-Green deformation tensor**, and the **Green-Lagrange Strain**:

$$\text{Right Cauchy-Green Deformation Tensor:} \qquad \boldsymbol{C} = \boldsymbol{F}^T\boldsymbol{F} \tag{8}$$

$$\text{Left Cauchy-Green Deformation Tensor:} \qquad \boldsymbol{B} = \boldsymbol{F}\boldsymbol{F}^T \tag{9}$$

$$\text{Green-Lagrange Strain:} \qquad \boldsymbol{E} = \frac{1}{2}(\boldsymbol{C} - \boldsymbol{I}) \tag{10}$$

1.4 Constitutive Laws

The stress-strain relationship for a body is defined by model called a **constitutive law**. There a various constitutive laws that make different assumptions about the response of a body, such as a material being compressible or incompressible, or behaving elastically or inelastically. In this analysis, we will use the **Neo-Hookean model**, which assumes hyperelastic material behavior and allows for compression. The law is expressed as an equation for the strain energy density of the body as a function of strain, from which we can derive expression for the **first Piola-Kirchhoff Stress** and **tangent moduli**.

The Neo-Hookean model expresses strain energy density as:

$$w(\boldsymbol{C}) = \frac{\lambda_0}{2}[ln(J)]^2 - \mu_0 ln(J) + \frac{\mu_0}{2}(tr(\boldsymbol{C}) - 3), \tag{11}$$

where $J = det(\boldsymbol{F})$ is referred to as the **Jacobian**, and λ_0 and μ_0 are the **first lamé parameter** and **shear modulus** of the material, respectively. The strain energy density can be rewritten entirely as a function of the deformation gradient $\boldsymbol{F}$ by expressing $tr(\boldsymbol{C})$ in terms of $\boldsymbol{F}$:

$$\begin{gathered} tr(\boldsymbol{C}) = C_{kk} = C_{kl}\delta_{kl} \\ C_{kl} = (F_{km})^T(F_{ml}) = F_{mk}F_{ml} \\ \implies tr(\boldsymbol{C}) = F_{mk}F_{ml}\delta_{kl} \end{gathered}$$

Now we can derive an expression for the first Piola-Kirchhoff stress P_{ij}:

$$\begin{aligned} P_{ij} = \frac{\partial w}{\partial F_{ij}} &= \frac{\partial}{\partial F_{ij}}\left[\frac{\lambda_0}{2}ln^2(J) - \mu_0 ln(J) + \frac{\mu_0}{2}(F_{mk}F_{ml}\delta_{kl} - 3)\right] && (12) \\ &= \lambda_0 ln(J)\left(\frac{1}{J}\right)\frac{\partial J}{\partial F_{ij}} - \mu_0\left(\frac{1}{J}\right)\frac{\partial J}{\partial F_{ij}} + \frac{\mu_0}{2}\left[\frac{\partial F_{mk}}{\partial F_{ij}}F_{ml}\delta_{kl} + F_{mk}\frac{\partial F_{ml}}{\partial F_{ij}}\delta_{kl}\right] && (13) \end{aligned}$$

Using the identity $\frac{\partial J}{\partial F_{ij}} = JF^{-1}_{ji}$:

$$\begin{aligned} P_{ij} &= \lambda_0 ln(J)\left(\frac{1}{J}\right)(JF^{-1}_{ji}) - \mu_0\left(\frac{1}{J}\right)(JF^{-1}_{ji}) + \frac{\mu_0}{2}\left[\delta_{mi}\delta_{kj}F_{ml}\delta_{kl} + F_{mk}\delta_{mi}\delta_{lj}\delta_{kl}\right] && (14) \\ &= \lambda_0 ln(J)F^{-1}_{ji} - \mu_0 F^{-1}_{ji} + \frac{\mu_0}{2}\left[\delta_{mi}\delta_{kj}F_{mk} + F_{ml}\delta_{mi}\delta_{lj}\right] && (15) \\ &= \lambda_0 ln(J)F^{-1}_{ji} - \mu_0 F^{-1}_{ji} + \frac{\mu_0}{2}\left[F_{ij} + F_{ij}\right] && (16) \\ &= \left[\lambda_0 ln(J) - \mu_0\right]F^{-1}_{ji} + \mu_0 F_{ij} && (17) \end{aligned}$$

We can take another derivative with respect to the deformation gradient to find the tangent moduli C_{ijkl}:

$$\begin{aligned} C_{ijkl} &= \frac{\partial P_{ij}}{\partial F_{kl}} && (18) \\ &= \frac{\partial}{\partial F_{kl}}\left[\left[\lambda_0 ln(J) - \mu_0\right]F^{-1}_{ji} + \mu_0 F_{ij}\right] && (19) \\ &= \lambda_0\left(\frac{1}{J}\right)\frac{\partial J}{\partial F_{kl}}F^{-1}_{ji} + \left[\lambda_0 ln(J) - \mu_0\right]\frac{\partial F^{-1}_{ji}}{\partial F_{kl}} + \mu_0\frac{\partial F_{ij}}{\partial F_{kl}} && (20) \end{aligned}$$

Using the identity $\frac{\partial F^{-1}_{ji}}{\partial F_{kl}} = -F^{-1}_{jk}F^{-1}_{li}$:

$$\begin{aligned} C_{ijkl} &= \lambda_0\left(\frac{1}{J}\right)(JF^{-1}_{lk})F^{-1}_{ji} + \left[\lambda_0 ln(J) - \mu_0\right]\left(-F^{-1}_{jk}F^{-1}_{li}\right) + \mu_0\delta_{ik}\delta_{jl} && (21) \\ &= \lambda_0 F^{-1}_{lk}F^{-1}_{ji} - \left[\lambda_0 ln(J) - \mu_0\right]F^{-1}_{jk}F^{-1}_{li} + \mu_0\delta_{ik}\delta_{jl} && (22) \end{aligned}$$

To summarize, we now have the following three expressions for the Neo-Hookean constitutive law in terms of the deformation gradient $\boldsymbol{F}$:

$$\text{Strain Energy Density:} \quad w(\boldsymbol{F}) = \frac{\lambda_0}{2}[ln(J)]^2 - \mu_0 ln(J) + \frac{\mu_0}{2}(tr(\boldsymbol{F}^T\boldsymbol{F}) - 3) \tag{23}$$

$$\text{First Piola-Kirchhoff Stress:} \quad P_{ij} = [\lambda_0 ln(J) - \mu_0]\, F_{ji}^{-1} + \mu_0 F_{ij} \tag{24}$$

$$\text{Tangent Moduli:} \quad C_{ijkl} = \lambda_0 F_{lk}^{-1} F_{ji}^{-1} - [\lambda_0 ln(J) - \mu_0]\, F_{jk}^{-1} F_{li}^{-1} + \mu_0 \delta_{ik}\delta_{jl} \tag{25}$$

1.5 Plane Stress

The assumption of plane stress places a constraint on the structure of the deformation gradient as well as the first Piola-Kirchhoff stress tensor. For this example, say we have a thin plate with the thickness aligned with the third coordinate axis. Then the deformation gradient should have no out of plane shear components, and a stretch component λ in the 3-direction to account for the fact that there may be some strain through the thickness:

$$\boldsymbol{F} = \begin{bmatrix} F_{11} & F_{12} & 0 \\ F_{21} & F_{22} & 0 \\ 0 & 0 & \lambda \end{bmatrix} \tag{26}$$

The first Piola-Kirchhoff stress tensor should have a value of 0 for P_{33}. Since the only arbitrary or prescribed quantities of $\boldsymbol{F}$ are the 2×2 matrix of in-plane elements, we say that P_{33} is a function only of $F_{\alpha\beta}$ (where α and β each run from 1 to 2) and λ (referred to as the **stretch ratio**):

$$P_{33}(F_{\alpha\beta}, \lambda) = 0 \tag{27}$$

Because $F_{\alpha\beta}$ is prescribed, we must solve this equation by finding the value of λ that makes it true. $\boldsymbol{P}(\boldsymbol{F})$ is nonlinear, and therefore must be solved iteratively using Newton's Method.

1.5.1 Newton's Method

Newton's Method is an iterative technique for solver a nonlinear equation $f(\lambda)$. To use it, we must start by choosing a reasonable initial value for λ for which $f(\lambda)$ likely does not equal zero.

$$f(\lambda_0) \neq 0 \tag{28}$$

Then we will perturb λ by some small quantity, and use a first order Taylor approximation to solve for the value of the perturbation that will make $f(\lambda)$ equal to zero.

$$f(\lambda + d\lambda) = f(\lambda) + \frac{df(\lambda)}{d\lambda} d\lambda = 0 \implies d\lambda = -\left(\frac{df(\lambda)}{d\lambda}\right)^{-1} f(\lambda) \tag{29}$$

We then use this perturbation to compute a new value of λ and repeat the process. This loop will continue until $f(\lambda)$ is within some tolerance of 0, at which point we say the loop **converges**. It is very important to note that if λ_0 is far enough from the final value of λ, this loop will **diverge**. In later reports, we will explore in more detail the techniques used to ensure good initial guesses.

For the plane stress application, the function we are attempting to solve iteratively is $P_{33}(F_{\alpha\beta}, \lambda) = 0$. Therefore we can express equation 29 in terms of the quantities of our problem as:

$$d\lambda = -(C_{3333})^{-1} P_{33}(F_{\alpha\beta}, \lambda) \tag{30}$$

1.5.2 2D Tangent Moduli

The plane stress assumptions serves to simplify the problem by reducing dimension from 3D to 2D. Once we have solved for lambda using Newton's method, we can now proceed with the analysis using reduced matrices containing only the in-plane components. First, note that 2D and 3D strain energy density are defined to be equal. For the first Piola-Kirchhoff stress, the transition to 2D is simple, because all components in the 3-direction have been forced to zero under the assumption of plane stress. Therefore, the in-plane components of $\boldsymbol{P}$ are nothing more than the 2×2 matrix containing the non-zero elements. In other words, $P_{\alpha\beta}$ is subset of P_{ij}. For the tangent moduli however, the transition is not that simple. Despite imposing plane stress, there will in general be non-zero elements in the 3-directions, and we cannot simply reduce to 2D by taking a subset of this tensor. Instead, we want to capture the contributions of these non-zero elements by created an adjusted 2D 4th order tensor from the full 3D tangent moduli. The components of the 2D tangent moduli can be found in the following way:

$$P^{2D}_{\alpha\beta} \equiv \frac{\partial w^{2D}}{\partial F_{\alpha\beta}} = \frac{\partial}{\partial F_{\alpha\beta}}\left[w(F,\lambda)\right] = \frac{\partial w}{\partial F_{\alpha\beta}} + \frac{\partial w}{\partial \lambda}\frac{\partial \lambda}{\partial F_{\alpha\beta}} \tag{31}$$

We know that $\frac{\partial w}{\partial F_{\alpha\beta}} = P_{\alpha\beta}$ and $\frac{\partial w}{\partial \lambda} = 0$, so we can write:

$$P_{\alpha\beta} = \frac{\partial w(F_{\alpha\beta},\lambda)}{\partial F_{\alpha\beta}}, \quad P^{2D}_{\alpha\beta} = P_{\alpha\beta} \tag{32}$$

This shows, as stated previously, that the 2D form of the first Piola-Kirchhoff stress is just a subset of the 3D form. Now we can use this to compute the tangent moduli:

$$C^{2D}_{\alpha\beta\delta\gamma} \equiv \frac{\partial P^{2D}_{\alpha\beta}}{\partial F_{\delta\gamma}} = \frac{\partial^2 w^{2D}}{\partial F_{\alpha\beta} F_{\delta\gamma}} = \frac{\partial}{\partial F_{\delta\gamma}}\left[P_{\alpha\beta}(F_{\alpha\beta},\lambda)\right] = \frac{\partial P_{\alpha\beta}}{\partial F_{\delta\gamma}} + \frac{\partial P_{\alpha\beta}}{\partial \lambda}\frac{\partial \lambda}{\partial F_{\delta\gamma}} \tag{33}$$

We know that $\frac{\partial P_{\alpha\beta}}{\partial F_{\delta\gamma}} = C_{\alpha\beta\delta\gamma}$ and $\frac{\partial P_{\alpha\beta}}{\partial \lambda} = \frac{\partial P_{\alpha\beta}}{\partial F_{33}} = C_{\alpha\beta 33}$, so we can write:

$$C^{2D}_{\alpha\beta\delta\gamma} = C_{\alpha\beta\delta\gamma} + C_{\alpha\beta 33}\frac{\partial \lambda}{\partial F_{\delta\gamma}} \tag{34}$$

Now we can find $\frac{\partial \lambda}{\partial F_{\delta\gamma}}$ by enforcing the plane stress assumption that $P_{33}(F_{\alpha\beta},\lambda) = 0$.

$$P_{33}(F_{\alpha\beta},\lambda) = 0 \implies dP_{33} = 0 = \frac{\partial P_{33}}{\partial F_{\alpha\beta}} dF_{\alpha\beta} + \frac{\partial P_{33}}{\partial F_{33}} d\lambda \tag{35}$$

$$0 = C_{33\alpha\beta} dF_{\alpha\beta} + C_{3333} d\lambda \tag{36}$$

$$0 = C_{33\alpha\beta} dF_{\alpha\beta} + C_{3333}\frac{\partial \lambda}{\partial F_{\alpha\beta}} dF_{\alpha\beta} \tag{37}$$

$$0 = \left(C_{33\alpha\beta} + C_{3333}\frac{\partial \lambda}{\partial F_{\alpha\beta}}\right) dF_{\alpha\beta} \tag{38}$$

$$\implies \frac{\partial \lambda}{\partial F_{\alpha\beta}} = -\frac{C_{33\alpha\beta}}{C_{3333}} \tag{39}$$

Now we can use this value to solve for the components of the 2D tangent moduli:

$$C^{2D}_{\alpha\beta\delta\gamma} = \frac{\partial}{\partial F_{\delta\gamma}}\left[P_{\alpha\beta}(F_{\alpha\beta},\lambda)\right] = C_{\alpha\beta\delta\gamma} + C_{\alpha\beta 33}\frac{\partial \lambda}{\partial F_{\delta\gamma}} \tag{40}$$

$$\implies C^{2D}_{\alpha\beta\delta\gamma} = C_{\alpha\beta\delta\gamma} - C_{\alpha\beta 33} C_{33\delta\gamma}\left(\frac{1}{C_{3333}}\right) \tag{41}$$

Using this equation we can compute the adjusted 2D tangent moduli under the assumption of plane stress from the components of the full 3D tangent moduli.

1.6 Finite Elements

In order to analyze the behavior of body, we must discretize the domain into finite elements. These elements can in general have any number of sides, but triangles and quadrilaterals provide more than enough flexibility and are much simpler to work with. In this analysis, we will focus on the use of triangular elements. When a domain in broken up into triangular elements, each triangle will in general have different dimensions and different orientations. For this reason it becomes very useful to map each general element into an isoparametric element.

Isoparametric elements are standard elements defined in a natural coordinate system for which we can use shape functions to interpolate the behavior of the element between nodes. An example of an isoparametric triangular element is shown in figure 1. The element is bounded by nodes, and the shape functions define the shape of the body between them. Shape functions relate the coordinates of every point in the element to the positions of the nodes, allowing for interpolation of values such as displacement within the element.

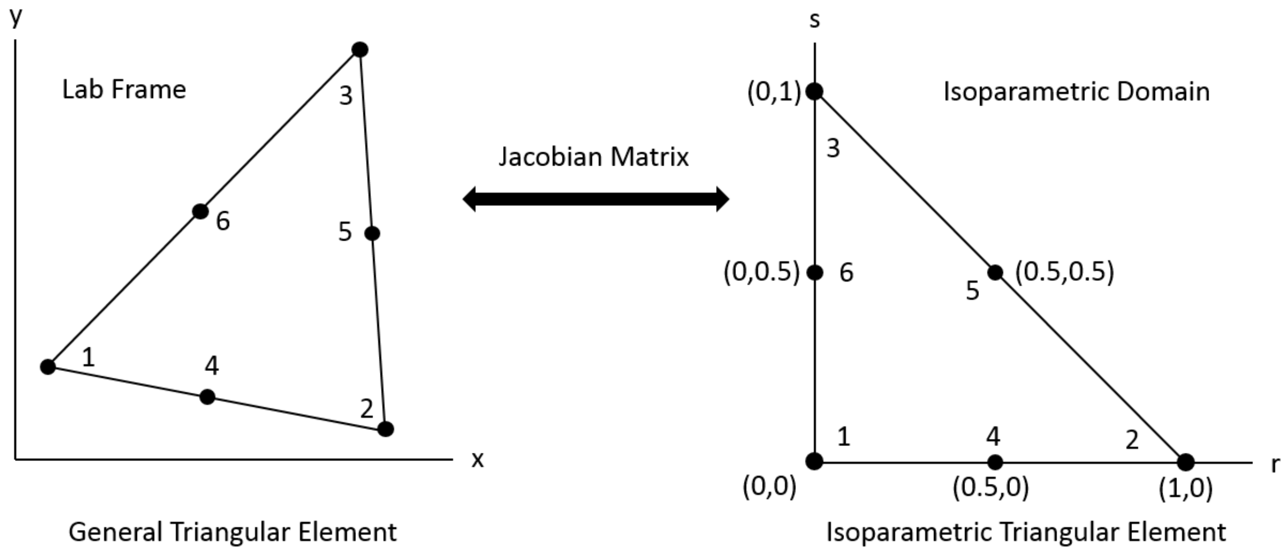

Figure 1. A general triangular element is mapped to an isoparametric triangular element by the Jacobian matrix. A linear triangular element uses only nodes 1-3, removing the midpoint nodes. A quadratic triangular element uses all 6 nodes.

The finite element analysis is driven by nodal positions, as the behavior of each element is dependent entirely on the behavior of the nodes. Depending on the desired accuracy, 3-node or 6-node triangular elements may be used. A 3-node triangular element is considered linear, as there is no information between the nodes to allow for curving. Thus the 6-node triangular element is considered quadratic, as the midpoint nodes along the edge allow for nonlinear behavior. For each of these element types, there are a number of shape functions equal to the number of nodes, and every isoparametric element is characterized by these same functions.

A location in the natural coordinate system (r, x) can be interpolated from the lab frame nodal positions (x, y) by the shape functions:

$$\boldsymbol{x}(r,s) = \sum_a N_a(r,s)\boldsymbol{x}_a \tag{42}$$

where a is indexing the nodes.

1.6.1 Jacobian Matrix

The general element is mapped from the lab frame to the isoparametric domain by the **Jacobian matrix**. The Jacobian matrix is based on the reference nodal positions and the shape functions, and can be expressed as:

$$J_{I\alpha} = \sum_a X_{ia} N_{a,\alpha} \tag{43}$$

where X_{ia} are the components of the reference nodal position vectors. The Jacobian matrix has dimensions of (lab frame dimensions) $\times$ (element dimensions). In this analysis, the plane stress assumption will allow for the use of two dimensional elements deforming in a three dimensional lab frame. Therefore the Jacobian matrix will be 3×2.

With the Jacobian matrix, we have a means for moving between the isoparametric and physical domains, and it will be used in the calculations for numerous quantities that describe the behavior of the element.

1.6.2 Element Behavior

With the Jacobian matrix defined, the deformation gradient can now be computed in alternative manner to using the basis vectors of the reference and current configurations as defined in equation 7. Using the nodal positions in the current configuration x_{ia}, and the information about the current configuration carried by the Jacobian matrix, the **deformation gradient** can be expressed as:

$$F_{ij} = \sum_a x_{ia} N_{a,\alpha} J^{-1}_{\alpha j} \tag{44}$$

With the deformation gradient and material response quantities from the constitutive model, three quantities can be computed: strain energy, internal nodal force array, and the stiffness matrix. These quantities describe the behavior of the entire element by integrating the material response quantities over the element domain. Here the isoparametric formulation comes in handy because the integral can be performed in the isoparametric domain and then transformed back to the physical domain using the inverse Jacobian matrix. These integrals are computed using a numerical integration technique called Gauss quadrature, which will be discussed in detail in section **??**.

The **strain energy** of the element is determined by integrating the strain energy density:

$$W = \int\limits_{\Omega_0} w dV \tag{45}$$

The **internal nodal force array** is the representation of a distributed force over the element at the nodes. In other words, the distributed load is converted to a set of equivalent forces acting only on the nodes of the element. The force array is determined by integrating the first Piola-Kirchhoff stress:

$$f^{int}_{ia} = \int\limits_{\Omega_0} P_{ij} N_{a,\alpha} J^{-1}_{\alpha j} dV \tag{46}$$

The **stiffness matrix** represents the resistance of the element to deformation in various directions. It is determined by integrating the two dimensional tangent moduli that has been adjusted for plane stress:

$$K_{iakb} = \int\limits_{\Omega_0} C^{2D}_{ijkl} N_{a,\alpha} N_{b,\beta} J^{-1}_{\alpha j} J^{-1}_{\beta l} dV \tag{47}$$

1.6.3 Gauss Quadrature

This analysis requires the evaluation of integrals, and it would be costly to perform integration explicitly. For this reason, we will use Gauss quadrature to perform numerical integration. Gauss quadrature works by using a weighted sum of function values at specific quadrature points within a domain. It is constructed to yield exact results for polynomial functions of degree $2n - 1$ or lower for n-point quadrature, provided that the polynomial is well-approximated at the quadrature points. In this analysis, we will make use of 1-point quadrature, which will evaluate a linear polynomial exactly, and 3-point quadrature, which will evaluate a fifth order polynomial exactly.

Whether or not the function is well-approximated at the quadrature points will be determined by whether the interpolated shape function values at these points can capture the element behavior. For example, for a 3-node isoparametric triangular element, 1-point quadrature will provide exact integration results, as the element can only display linear behavior, which will be exactly interpolated by the shape functions. In order to accurately capture quadratic behavior, a 6-node element must be used to pick up the behavior between corner nodes. We must also use 3-point quadrature to ensure accuracy, because the accuracy 1-point quadrature is limited to linear functions.

The computational cost will be lowest for the lowest order to quadrature, therefore the analysis will make use of the lowest order quadrature possible to ensure accurate results. There are applications for intentionally using lower-order quadrature, but these will not be discussed here.

The general expression for Gauss quadrature of a function $g(\zeta)$ is:

$$\int_{-1}^{1} g(\zeta)d\zeta = \sum_{i=1}^{n} g(\tilde{\zeta}_i)w_i \tag{48}$$

where n is the number of quadrature points, $\tilde{\zeta}_i$ is the coordinate of the ith quadrature point, and w_i is the weight of the ith quadrature point.

The application of this equation in the analysis is further discussed in *Quadrature*.

2. CODE ARCHITECTURE & FORMULATION OF NUMERICAL METHODS

I have put in a lot of effort to ensure that the initial code base is structured to be robust and extensible to future developments. This means that future work will involve adding more to the existing classes and functions rather than rewriting them. I have done my best to take full advantage of Python as an object-oriented project language to create an architecture that makes sense according to the current state of my evolving understanding of finite element analysis. I will describe my code at the highest level before discussing the details of implementation.

2.1 Code Architecture

My goal is to structure the model such that its inputs are provided in an intuitive format that is repeatable for any application. Whether running a simple verification test or a full deformation analysis on a body, the inputs should only differ in their values and the desired output information requested.

2.1.1 Brief Module Descriptions

constants.py integers and strings used throughout the model such they can be conveniently referenced without having to hard-code their values into the model.

constitutive_models.py contains classes for each constitutive law (ie. Neo-Hookean), each of which contains methods for computing the values of strain energy density, first Piola-Kirchhoff stress, and tangent moduli.

deformation_gradient.py deformation gradient class which defines and updates the deformation of the body under specified assumptions.

elements.py contains the different types of isoparametric elements available for use in analysis, including the 3 and 6-node triangular elements. Also contains methods for updating element-level properties such as strain energy.

exceptions.py errors that are raised during the analysis (usually from the tests module) in the event of an incorrect or unexpected result, or some kind of violation that indicates a breaking of physical laws or constraints.

frames.py contains classes for the frames/configurations and the basis vectors.

kinematics.py functions that compute the strain tensors from the deformation gradient.

materials.py contains classes for different material options, each of which contains the properties of the material.

model.py contains the finite element model, the master class that manages the analysis and keeps track of the highest level information.

model_io_#.py this is the module from which the model is set up and run, and is the only file a user will interact with.

operations.py commonly used functions, such as Newton's Method, or the generation of a random deformation gradient for testing purposes.

nodes.py contains node classes that keep track of reference and current positions, as well as local and global IDs for assembly purposes.

quadrature.py contains parameters for one and three point Gauss quadrature, as well as a quadrature point class that contains properties for material response values.

tests.py verification test functions that ensure the accuracy of the code and check for physical violations.

2.1.2 Finite Element Model

The highest level object in the analysis is the Model class, or the Finite Element Model. This is the "master" of the analysis, and is responsible for keeping track of globally needed information as well as setting up and running the analysis. The model keeps track of the material, constitutive model, quadrature class, and element type. It is also responsible for creating the nodes and elements, and performing the global assembly. It stores lists of all element and node objects in the analysis to allow for easy iteration downward from the model level when updating the current configuration.

Model Setup Elements are the building blocks of the model, and are composed of nodes and quadrature points. Nodes define the mesh that discretizes the domain of the body, and belong to multiple parent elements. For this reason, nodes are created first by the model through a meshing function (TO BE ELABORATED ON LATER) and assigned a reference position, a list of parent elements, and both local and global IDs. The reference position defines the position of the node in the reference configuration. The list of parent elements defines the element objects the node belongs to. The local ID keeps track of the node number 1-6 within an element as shown in figure 1. Therefore many nodes will share local IDs, creating the need for a global ID, which creates a unique identifier for the node within the model.

Once the nodes are created, they are assigned to their parent elements, who store a list of their nodes. With nodes assigned, the elements then create their quadrature point(s) based on the quadrature class selected by the model and then perform the one-time calculation of the Jacobian matrix from the reference nodal positions. The primary advantage of using isoparametric elements is that once the general elements have been mapped to the isoparametric domain by the Jacobian matrix, every element can be treated identically.

This process can be summarized by the order of method calls, shown here:

1. model.create_mesh (TO BE ADDED LATER)
   - model.create_nodes
   - model.create_elements
2. model.assign_nodes
3. element.create_quadrature_points (for each element)
   - element.calculate_jacobian_matrix

Updating the Current Configuration For each step of deformation, the model must update its current configuration. To do this, the model iterates through each element, which in turn iterates through its nodes, and updates their current position. With the current nodal positions defined, each element can use its shape functions to update the deformation gradient at each of its quadrature points. Each quadrature point then uses its deformation gradient to compute the material response from the constitutive law. With the quadrature point quantities fully defined, each element can use gauss quadrature to integrate over the element and compute values for the strain energy, force array, and stiffness matrix.

This process can be summarized by the order of method calls, shown here:

1. model.update_current_configuration
2. element.update_current_configuration (for each element)
3. node.update_current_position (for each node in the element)
4. quadrature_point.update_deformation_gradient (for each quadrature point in the element)
5. deformation_gradient.update_F (for the deformation gradient of each quadrature point in the element)
6. deformation_gradient.enforce_plane_stress (for the deformation gradient of each quadrature point in the element)
7. quadrature_point.update_material_response (for each quadrature point in the element)
8. element.update_strain_energy (for each element)
9. element.update_force_array (for each element)
10. element.update_stiffness_matrix (for each element)

2.2 Implementation and Verification Tests

In this section, I will describe the key design choices made with regards to implementation, and in particular how these relate to the verification tests that ensure the correctness of the code and validity of the analysis.

2.2.1 Assembly

To be elaborated on in final report.

2.2.2 Numerical Differentiation

The model uses many equations that require the programming of hand computed derivatives. In order to ensure correctness of derivatives, the results can be compared against the results of numerical differentiation.

The numerical differentiation test uses the 3-point formula to check the validity of our computed results for the first Piola-Kirchhoff and the tangent moduli. The 3-point formula comes from a 3rd order Taylor expansion, and is given by:

$$f'(a) \approx \frac{f(a+h) - f(a-h)}{2h} \equiv f'_h(a) \tag{49}$$

$$\text{Error} = f'_h(a) - f'(a) < TOLERANCE \implies pass \tag{50}$$

The perturbation h is applied element by element, and the approximated value is compared to the exact value computed from the constitutive law to ensure that they are within tolerance of each other. The tolerance is necessary because the two values will not be an exact match, and there is an expected error from the Taylor expansion on the order of h^2.

2.2.3 Elements

Elements are implemented as classes that inherit from a base element class that contains all necessary properties for defining an element as well as the necessary methods for analysis. Element properties include dimension, node_quantity, and node_positions, all of which are defined based on element type.

The element class defines the nodal positions in the isoparametric domain, and the order of node objects in the nodes list will match the ordering for the isoparametric element (ie. 1-3 are corner nodes, 4-6 are midpoint nodes, see figure 1). This order is important such that the Jacobian matrix mapping makes physical sense.

The element class also contains the material quantities that include the strain energy, force array, and stiffness matrix, which are updated for every deformation.

Tests The force array and stiffness matrix are the first and second derivative of the strain energy, respectively. Their values are checked against numerical differentiation using the 3-point formula from equation 49. In this case, it is the current nodal positions that are perturbed, which in turn creates a perturbed deformation gradient at each quadrature point, which leads to perturbed strain energy densities and therefore a perturbed strain energy and force array. For this reason, strain energy and force array will appear as functions of current nodal position to make this point clear:

$$(F_h)_{ia} = \frac{W(x_{ia}+h) - W(x_{ia}-h)}{2h} \tag{51}$$

$$(K_h)_{iakb} = \frac{F_{ia}(x_{kb}+h) - F_{ia}(x_{kb}-h)}{2h} \tag{52}$$

2.2.4 Shape Functions

The element classes contains predefined shape functions and shape function derivatives as class methods for each element type that return values at a given location in the isoparametric domain. The shape functions are defined inside the element classes because they are specific to element type and number of nodes.

Tests The shape function derivatives are verified using the 3-point formula from equation 49. For derivatives with respect to coordinate r, the r position of the shape function is perturbed, and likewise for s derivatives:

$$\frac{\partial N}{\partial r} = \frac{N(r+h,s) - N(r-h,s)}{2h} \tag{53}$$

$$\frac{\partial N}{\partial s} = \frac{N(r,s+h) - N(r,s-h)}{2h} \tag{54}$$

Shape functions also have the important properties of satisfying partition of unity, partition of nullity, and completeness. Partition of unity that the sum of the shape functions at any point in the isoparametric domain must sum to 1. Therefore, each element type must be tested such that its shape functions satisfy:

$$\sum_a N_a(r,s) = 1 \tag{55}$$

Additionally, the shape function derivatives must satisfy the partition of nullity, or that they must sum to 0 at any point in the domain:

$$\sum_a N_{a,\alpha}(r,s) = 0 \tag{56}$$

Finally the shape functions must be complete, meaning that they can interpolate a random linear polynomial p exactly. This means that the value of a p at a random point in the domain $\boldsymbol{\theta^*}$ should be exactly equal to the value interpolated by the shape functions using the values of p at the nodes $p(\boldsymbol{\theta_a})$. We can define a random linear polynomial as:

$$p(\boldsymbol{\theta}) = \sum_{|\alpha|\leq 1} a_\alpha \theta^\alpha \tag{57}$$

where a_α are random coefficients. Now we can check that the value of the polynomial at a random point is equal to the interpolated values as:

$$p(\boldsymbol{\theta}^*) = \sum_a p(\boldsymbol{\theta}_a) N_a(\boldsymbol{\theta}^*) \tag{58}$$

2.2.5 Nodes

Nodes are implemented as classes. There are two types of nodes that inherit from a base node class: CornerNode and MidpointNode. Both have an identical set of properties, including:

- local ID
- global ID
- reference position
- current position
- list of parent elements

The reason why these two classes are separated is solely for the purpose of identification. When the mesh is created, the corner nodes define the element boundaries, and the midpoint nodes are not created until the mesh is complete. (ELABORATE IN FINAL REPORT).

2.2.6 Quadrature

There are quadrature classes for one-point and three-point quadrature with properties that define the number of quadrature points, and their positions and weights. The quadrature point class is used to create the quadrature point objects that are assigned to each element for the purposes of numerical integration using Gauss quadrature. Each object has a position and a weight based on the quadrature class being used by the model, and has properties for the quadrature point-level quantities described in *Property Levels*.

Gauss Quadrature Gauss quadrature is carried out in the element methods that evaluate the strain energy, force array, and stiffness matrix.

Because this analysis will make use of isoparametric triangular element under the plane stress assumption, the bounds of integration must be adjusted from equation 48, and the integral must be made two dimensional. The following will be the expression implemented in the code for numerical integration (written for 2-tensor but would be the same form for a different order) over the isoparametric element domain:

$$\int_0^1 \int_0^{1-s} G_{ij}(r,s)\,dr\,ds \approx \frac{1}{2}\sum_{k=1}^{n} G_{ij}(r_k, s_k) w_k \tag{59}$$

As a test for the implementation of Gauss quadrature, equation 59 should produce exact results for the integration of a random linear polynomial using one-point quadrature, and for a random quadratic polynomial using three-point quadrature. This will be tested by integration polynomials over the isoparametric domain.

2.2.7 Property Levels

It is important to distinguish the level at which certain key quantities in the analysis are defined. There are three levels we are concerned with: model-level, element-level, and quadrature point-level. At each of these levels, some quantities are defined or computed one time, and others are updated with every deformation step. This information is summarized here:

Model:

- material
- constitutive model
- quadrature class
- element type

Element:

One-time:

- Jacobian matrix

Every step:

- strain energy
- internal nodal force array
- stiffness matrix

Quadrature Point:

One-time:

- position
- weight

Every step:

- deformation gradient
- strain energy density
- first Piola-Kirchhoff stress
- tangent moduli

2.2.8 Deformation Gradient

The deformation gradient is extremely important, as it is the driver for all kinematics and material response. Therefore, if we begin our analysis with a deformation that does not make physical sense, then neither will our results. The deformation gradient is represented as a property of the quadrature point class. Each time a deformation gradient is updated with its matrix values, the Jacobian is immediately computed and checked to be physical. For the deformation gradient to have physical meaning, the Jacobian must be greater than zero, otherwise there is some unphysical inversion of the body taking place. If the Jacobian is negative, an error is raised to indicate this, and the analysis is halted. This ensures that an unphysical deformation gradient never makes it past initialization.

This quantity is also unique in that its structure is directly affected by the assumption of plane stress. So if a quadrature point is initialized with this assumption, it will be enforced by computing the value for the unknown stretch ratio using Newton's Method (more detail on this later), and then again checking the value of the Jacobian from the finalized matrix to ensure validity.

Similarly, the quadrature point class also contains a method for enforcing plane strain, but this has not yet been implemented.

2.2.9 Constitutive Laws

The model keeps a reference to the constitutive law object being using for the analysis. As of now, there is only one option, and that is the Neo-Hookean model. The constitutive law is nothing more than a set of three methods that compute the material response from a 3×3 deformation gradient. The expressions for these quantities are computed by hand such that derivatives do not need to be handled in the code, as this would be very difficult. The class has no knowledge of the assumptions of the model (ie. plane stress/strain), as this information is contained in the construction of the deformation gradient itself, and therefore the return values will reflect this assumption without being aware of it.

The methods for first Piola-Kirchhoff stress and tangent moduli contain two optional parameters: the requested dimension of the result (defaulted to 3), and a boolean for whether to test the result against 3-point numerical differentiation (defaulted to True). When performing a plane stress analysis, the requested dimension would be 2, in which case the first Piola-Kirchhoff stress will return a 2×2 subset of the 3×3 result, and the tangent moduli will be adjusted according to equation 41 and will return a 2D 4th order tensor. If the test boolean is set to true, the computed result will perform a verification test *prior* to being returned. This ensures that an incorrect value is not being passed back out to the model.

Tests The approximate values for P_{ij} and C_{ijkl} using the 3-point formula from equation 49 are written as:

$$(P_h)_{ij} = \frac{w(F_{ij} + h) - w(F_{ij} - h)}{2h} \tag{60}$$

$$(C_h)_{ijkl} = \frac{P_{ij}(F_{kl} + h) - P_{ij}(F_{kl} - h)}{2h} \tag{61}$$

As each element F_{ij} is perturbed, the entire matrix is passed out to the constitutive law, which computes the strain energy and first Piola-Kirchhoff stress, allowing for the numerical derivative to be approximated and compared against the exact value. The error in each element is calculated and the larges value is compared against the tolerance to ensure that there are no elements being computed incorrectly.

2.2.10 Newton's Method

The Newton's method solver is a loop that iteratively solves for the stretch ratio beginning with an initial guess. If the initial guess is bad enough, it is possible for the lambda update $d\lambda$, computed in equation 30, to cause the value of the stretch ratio to go negative. This will produce a negative Jacobian, and therefore an unphysical deformation gradient. If this is the case, the stretch ratio is set to a small negative value, 10^{-6}, to give the solver another chance to converge rather than simply raising a Jacobian error. In many cases, the solver will still fail to converge, but this implementation at least allows the loop to use that maximum number of iterations, and then raise a convergence error if necessary.

2.3 Additional Verification Tests

In addition to the numerical differentiation check using the 3-point formula, which ensures that derivatives are being computed correctly, the code also checks that the material response satisfies certain physical rules. These rules include **material frame indifference** and **material symmetry**. These tests serve as stand-alone unit tests that are run on the constitutive law classes using randomly generated deformation gradients to ensure that the methods have been written correctly.

2.3.1 Material Frame Indifference

Material frame indifference suggests that rotating the frame of reference or performing a rigid body rotation should not change the energy of the system. Therefore, if a random rotation is applied to the deformation gradient and then the strain energy density is computed, we should expect the value to remain equivalent to the value prior to rotation. This concept can be expressed as:

$$w(\boldsymbol{QF}) = w(\boldsymbol{F}) \tag{62}$$

As for the first Piola-Kirchhoff stress and tangent moduli, we do expect their elements to change with rotation. However, it should make no difference if the tensor is computed and then rotated, or the deformation gradient is rotated and then used to compute the tensor. In other words, the order of operation should not matter. This can be expressed as:

$$P_{ij}(\boldsymbol{QF}) = Q_{ik}P_{kj}(\boldsymbol{F}) \tag{63}$$

$$C_{ijkl}(\boldsymbol{QF}) = Q_{im}Q_{kn}C_{mjnl}(\boldsymbol{F}) \tag{64}$$

2.3.2 Material Symmetry

The material symmetry tests are used to verify that the constitutive laws preserve material symmetries as expected. For example, if a material is isotropic, we expect that the strain energy will be the same if the body is deformed without rotation, and if the body is rotated and then deformed. This can be expressed as:

$$w(\boldsymbol{FQ}) = w(\boldsymbol{F}), \quad \forall \boldsymbol{Q} \tag{65}$$

We can also imagine that other material symmetries can be demonstrated if equation 65 is satisfied for only particular rotation matrices. Additionally, we expect the first Piola-Kirchhoff stress and tangent moduli to transform as:

$$P_{ij}(\boldsymbol{FQ}) = Q_{kj}P_{ik}(\boldsymbol{F}) \tag{66}$$

$$C_{ijkl}(\boldsymbol{FQ}) = Q_{mj}Q_{nl}C_{imkn}(\boldsymbol{F}) \tag{67}$$

2.3.3 Random Rotation Matrices

For the purposes of the material frame indifference and material symmetry tests, random 3D rotation matrices $\boldsymbol{Q}$ are generated using Rodrigues' formula:

$$\boldsymbol{Q} = \boldsymbol{I} + \hat{\boldsymbol{n}}(\sin\theta) + (1-\cos\theta)(\boldsymbol{n}\otimes\boldsymbol{n} - \boldsymbol{I}) \tag{68}$$

where

$$[\hat{n}_{ij}] = \begin{bmatrix} 0 & -n_3 & n_2 \\ n_3 & 0 & -n_1 \\ -n_2 & n_1 & 0 \end{bmatrix}. \tag{69}$$

2.3.4 Random Deformation Gradients

For the purposes of testing, it is useful to be able to generate random deformation gradients that have some resemblance to "real" deformation gradients, such that the tests are meaningful. To generate these random matrices, we begin with the identity matrix and add a random 3×3 matrix with elements having values ranging from 0 to 1. This range could be extended further, but it is not necessary for testing. The expression can be written as (in psuedocode):

$$\boldsymbol{F_{rand}} = \boldsymbol{I} + rand(3,3) \tag{70}$$

Once computed, the random deformation gradient is checked for satisfying the physical constraint that the Jacobian is positive before being returned.

2.3.5 Stiffness Matrix Rank

The stiffness matrix rank indicates how many degrees of freedom the element has to move without changing the energy of the configuration. For example, in two dimensions, a 3-node triangular element has two degrees of freedom at each node, for a total of six degrees of freedom. These correspond to translations of each node along two in-plane coordinate axes. If there is no external loading on the element, it can translate freely in two dimensions and rotate without changing the energy of the configuration. This means that there are three "zero energy modes", and these are subtracted from the total number of degrees of freedom, 6, to get a rank of 3. 3 indicates the number of non-zero energy modes of the element.

Now if some external loading is applied, the two translations with still have no effect on the energy, but a rotational will cause a change in energy. This is because the loading has some direction associated with it, and therefore rotating the element will change the direction of the forces relative to the edges of the element, and the stress will change. Therefore, a deformed element has 4 non-zero energy modes, and the stiffness matrix has a rank of 4.

Similar logic can be followed for the 6-node element. There are two degrees of freedom at each node, giving a total of 12. An undeformed element with again have 3 zero energy modes (2 translations and a rotation), giving a rank of 12 - 3 = 9. A deformed element with have 2 zero energy modes (2 translations) giving a rank of 12 - 2 = 10.

These results provide verification tests that can be performed on the stiffness matrix to ensure the correctness of its implementation.

3. CALCULATIONS AND RESULTS

3.1 Uniaxial Deformation of a Cylinder

As a basic test of handling curvilinear coordinates and the computation of kinematic quantities, we look at the uniaxial deformation of a cylinder loaded with tractions that produce a deformed position map

$$\boldsymbol{x} = \boldsymbol{\varphi}(R, \Phi, Z) = r(R)\boldsymbol{e}_R + z(Z)\boldsymbol{e}_Z,$$

where

$$r = \lambda_1 R, \quad \text{and} \quad z = \lambda_2 Z,$$

and λ_1 and λ_2 are arbitrary positive numbers such that $\lambda_1^2\lambda_2 < 0$. This constraint is necessary for the deformation mapping to make physical sense. A negative value of λ_1 or λ_2 would indicate some kind of impossible inversion of the material.

The lab frame components of the curvilinear coordinate system is described by standard cylindrical coordinate mappings given by:

$$[\boldsymbol{e}_R] = \begin{pmatrix} \cos\Phi \\ \sin\Phi \\ 0 \end{pmatrix} \quad [\boldsymbol{e}_\Phi] = \begin{pmatrix} -\sin\Phi \\ \cos\Phi \\ 0 \end{pmatrix} \quad [\boldsymbol{e}_Z] = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}.$$

From the mapping and the lab frame mappings, we compute the covariant and contravariant basis vectors in both the reference and deformed configurations. From here, the deformation gradient is computed using equation 7, and gives:

$$\boldsymbol{F} = \begin{pmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_2 \end{pmatrix}$$

Note that the deformation gradient does not depend on the choice of R or Φ, and therefore neither will the Cauchy-Green deformation tensors or the Green-Lagrange strain. Computing these quantities for $\lambda_1 = 2$ and $\lambda_2 = 3$ gives:

$$\boldsymbol{F} = \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 3 \end{pmatrix}, \quad \boldsymbol{C} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{pmatrix}, \quad \boldsymbol{B} = \begin{pmatrix} 4 & 0 & 0 \\ 0 & 4 & 0 \\ 0 & 0 & 9 \end{pmatrix}, \quad \boldsymbol{E} = \begin{pmatrix} 1.5 & 0 & 0 \\ 0 & 1.5 & 0 \\ 0 & 0 & 4 \end{pmatrix}$$

These matrices match those computed by hand.

3.2 Nonlinear Elasticity – Neo-Hookean Model

The Neo-Hookean constitutive model was implemented using the equations derived in section 1.4 for the strain energy density, first Piola-Kirchhoff stress, and tangent moduli. In order to check the accuracy of their implementation, we use the 3-point formula numerical differentiation tests, the material frame indifference tests, and the material symmetry tests.

3.2.1 3-point formula

When using the 3-point formula, there are two important inputs that determine the accuracy of the result: the deformation gradient $\boldsymbol{F}$ and the perturbation value h. To this end, I noticed there is actually a distinction between "good" and "bad" deformation gradients that satisfy the requirement that the Jacobian be greater than 0. If for example, a random deformation gradient is generated simply by a random 3×3 matrix with elements between 0 and 1, this will usually result in a "bad" deformation gradient, and produce much larger errors for a single value of h. But using equation 70 to generate the deformation gradients gives errors much closer to the order of h^2 expected from equation 49.

The plots in figure 2 show the errors for 100 values of h for both "good" and "bad" deformation gradients.

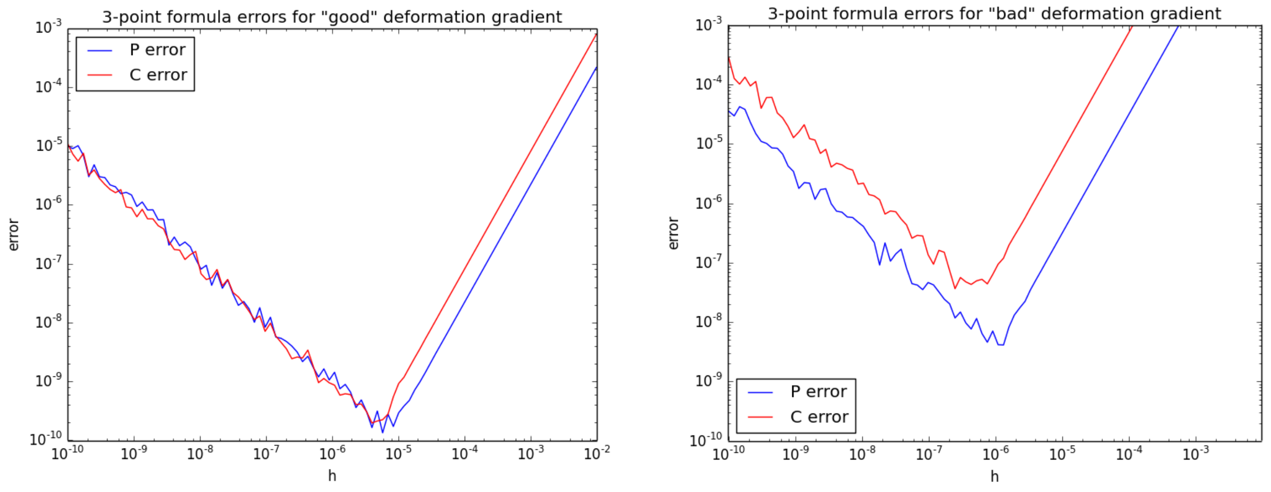

Figure 2. The "good" deformation gradient yields a smaller error for both the first Piola-Kirchhoff stress and in particular for the tangent moduli. Both plots indicate that the minimum error occurs for a perturbation between 10^{-5} and 10^{-6}.

In order to find the value of h that gives the minimum errors, I generated 100 random deformation gradients and computed the average error for the stress and for the tangent moduli for several values of h for a custom material with first lamé parameter $\lambda = 5$ and shear modulus $\mu = 3$. The results are shown in table 1.

Table 1. The perturbation h affects the error of the 3-point formula approximation.

h	Average P Error	Max P Error	Average C Error	Max C Error
7.5×10^{-4}	4.4×10^{-6}	1.1×10^{-4}	3.4×10^{-5}	1.2×10^{-3}
$\mathbf{1.0 \times 10^{-5}}$	5.3×10^{-10}	3.7×10^{-9}	2.5×10^{-9}	2.5×10^{-8}
2.5×10^{-5}	5.8×10^{-9}	1.9×10^{-7}	4.9×10^{-8}	2.4×10^{-6}
5.0×10^{-5}	4.9×10^{-8}	3.4×10^{-6}	7.7×10^{-7}	6.7×10^{-5}
7.5×10^{-5}	6.1×10^{-8}	3.0×10^{-6}	6.5×10^{-7}	4.8×10^{-5}
1.0×10^{-6}	1.4×10^{-9}	2.9×10^{-9}	1.2×10^{-9}	4.0×10^{-9}
2.5×10^{-6}	6.2×10^{-10}	1.5×10^{-9}	9.0×10^{-10}	1.8×10^{-8}
5.0×10^{-6}	3.3×10^{-10}	1.7×10^{-9}	8.4×10^{-10}	1.6×10^{-8}
7.5×10^{-6}	5.8×10^{-10}	1.5×10^{-8}	3.9×10^{-9}	1.7×10^{-7}

Based on figure 2 and table 1, the default value for the perturbation h was selected to be 10^{-5} because it seems to provide consistently small average and maximum errors for both the stress and tangent moduli. h is included as an optional parameter in the verification test functions that can be specified if desired. Note that the absolute value of these errors can be affected by the material properties by several orders of magnitude. This will be examined further in the discussion section of the report.

Using the value of h selected based on the results above, the numerical differentiation tests were performed on the force array and the stiffness matrix for a single element for both an undeformed and a deformed state. h was swept over 100 values from 10^{-3} to 10^{-10} on a log scale, and the results are plotted in figure 3.

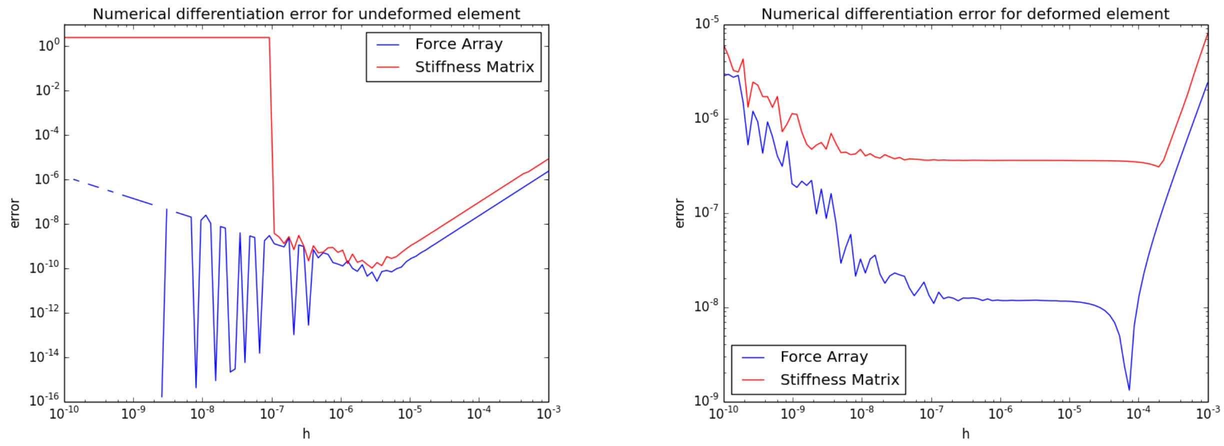

Figure 3. Numerical differentiation error for the internal nodal force array and stiffness matrix in both undeformed and deformed elements.

It should be noted that the element shape function derivatives also passed the numerical differentiation tests as well.

3.3 Verification Tests

The material frame indifference and material symmetry tests are performed on random deformation gradients to check for the validity of the constitutive model. Typical results for the quantities defined in section *Additional Verification Tests* are shown in tables 2 and 3.

These quantities should be exact matches, so the only error here is due to loss of precision from floating point operations. These operations give 16 digits of accuracy, which is why we see errors only in the last few digits on the order of 10^{-13} to 10^{-15} depending on the order of magnitude of the quantity itself.

Table 2. Typical results of material frame indifference tests.

Quantity	Max Error
w	1.4×10^{-14}
$\boldsymbol{P}$	3.6×10^{-15}
$\boldsymbol{C}$	4.3×10^{-14}

Table 3. Typical results of material symmetry tests.

Quantity	Max Error
w	2.8×10^{-14}
$\boldsymbol{P}$	1.4×10^{-14}
$\boldsymbol{C}$	2.1×10^{-14}

3.3.1 Gauss Quadrature

The results of the integration test using Gauss Quadrature are shown in table 4.

Notice that 3 of the 4 results are exact, limited only by numerical precision of floating point operations. One-point quadrature on a quadratic polynomial produces a significant error because the single point does not provide enough fidelity to account for the quadratic behavior. This is to be expected.

3.3.2 Stiffness Matrix Rank

The stiffness matrix rank was tested from both linear and quadratic element for undeformed and deformed cases, and has shown to reliably produce the correct rank over hundreds of random deformations.

3.4 Tolerance

We want to determine the smallest possible tolerance value for which all of our tests will pass. The numerical differentiation test is really the limiting factor here, and as these errors are determined by the quality of the approximation formula, unlike the material tests which provide nearly exact answers only limited by 16-digit precision. The numerical differentiation tests involve expressions that contain material properties λ and μ, and therefore the error will actually vary as a function of these parameters as well. The code will raise an exception in the case were the error of the numerical approximation exceeds the tolerance. In order to determine a tolerance value that will work for all materials, 100 tests were run using random deformation gradients for 3 materials, and the exceptions were counted for a given tolerance. One of the materials is a custom material, with $\lambda = 6$ and $\mu = 3$, and the other two are aluminum alloy and glass. The results are shown in table 5.

4. DISCUSSION AND CONCLUSIONS

The code has been thoroughly tested for over 10,000 random deformation gradients (all of which were checked to be physical) using several different materials such as aluminum alloy, lead, and glass, as well as custom materials. All verification tests pass as well as the numerical differentiation checks. The passing of material frame indifference and material symmetry implies that the constitutive law is implemented correctly, as the quantities behave appropriately under random rotations. These tests yield exact results limited only by the 16-digit precision of floating point operations.

The passing of the numerical differentiation tests, using the 3-point formula, implies that the computed quantities that come from derivatives of the deformation gradient, namely the first Piola-Kirchhoff stress, tangent moduli, force array, stiffness matrix, and shape function derivatives, are within tolerance of the values that result from Taylor Expansion. Whether these checks pass is largely determined by the tolerance value chosen. As discussed previously, the perturbation value h used in the 3-point formula has been selected such that the error is consistently minimized (see 1). However, the absolutely value of this error is affected by the material selected, because the material properties are used in the expression for the stress and tangent moduli. Based on the results in table 5, as well as some additional testing, I have chose a tolerance of 10^{-6}, because this value consistently results in 0 exceptions for every material tested, while 10^{-7} will occasionally result in an error or two. I could perform further testing at finer intervals than factors of 10, but this will provide an adequate starting point. It

Table 4. Typical results of numerical integration using Gauss quadrature vs exact integration.

Quadrature Order	Polynomial Order	Error
One-point	Linear	2.8×10^{-17}
One-point	Quadratic	0.0039
Three-point	Linear	1.7×10^{-17}
Three-point	Quadratic	1.4×10^{-17}

Table 5. Number of exceptions for 3 materials at various tolerance values out of 100 runs.

Tolerance	# of Exceptions (Custom)	# of Exceptions (Al)	# of Exceptions (Glass)	# of Runs
10^{-6}	0	0	0	100
10^{-7}	0	2	1	100
10^{-8}	0	64	25	100
10^{-9}	81	100	100	100
10^{-10}	100	100	100	100

should be noted that while the derivation of the 3-point formula implies that errors should scale with h^2, which is indicated by the slope of 2 on the log-log plots shown in figure 2.

The error plots for the undeformed and deformed element in figure 3 exhibit some erratic behavior, particularly when compared to the plots for the Piola-Kirchhoff and tangent moduli. However, the most important thing about these plots is that they still show the general behavior we hope to see in an error sweep like this. The behavior is erratic for very small values of h, which is to be expected, and the curves settle into a slope of 2 to the right of the minimums. Also, notice that the minimums occur at h values nearer to 10^{-5} and 10^{-4}. If there were some error in the implementation of either the force array or stiffness matrix, these plots would show wildly random behavior. This gives me confidence that the results of the numerical differentiation tests indicate that all quantities have been implemented correctly.

These results coupled with the successful rank verification tests of the stiffness matrix, give me full confidence that everything to this point is working as it should be.

5. SOURCE CODE LISTING

New and updated files: constitutive_models.py, elements.py exceptions.py, model.py, nodes.py, operations.py, quadrature.py, tests.py