ACSE-9: Applied Computational Science Project

Applying machine learning to the optimisation of numerical integration in finite element method

Ye Liu

Supervisor: Adriana Paluszny

A thesis presented for the degree of
MSc Applied Computational Science and Engineering

Imperial College London
ye.liu18@imperial.ac.uk
Github:@lunayeliu

August 2019

Contents

Abstract

A light finite element method software with smart integration is developed. The smart integration is a combination of the machine learning and numerical integration modules. The neural network in the machine learning module can select the optimal number of integration points for numerical integration according to elements shapes. The classification accuracy of the network on unseen data can reach 94.3%. In addition, a novel multiple-multivariate-multidimensional tensor integration method is developed. It outperforms the Scipy built-in integration function nquad on efficiency (20 times faster on test cases) and par on accuracy. The new tensor integration can also support multidimensional tensor integration(such as matrix integration), which can not be achieved by any Scipy built-in functions.

1 INTRODUCTION

1.1 Finite Element Method

Coupled processes in fractured media are a series of complex mechanics problems [1], which are usually described by Partial Differential Equations (PDEs). PDEs are usually solved numerically because of the lack of analytic solutions for complex problems [2]. Finite difference, finite element, finite volume, and Monte-Carlo method are commonly used numerical methods. In computational solid mechanics field, finite element method (FEM) is used the most frequently. Due to the requirement of high fidelity simulation, the size of the problems to be solved has been growing larger and larger [3]. Traditional deterministic methods such as the Finite Element Method (FEM) highly depends on computation resources. The time to solve large size real-world problems would become extremely long even when using High Performance Computing cluster. Therefore, it demands new techniques to solve these problems more efficiently [4].

1.2 Numerical Integration

Numerical integration is one of the most important parts in the Finite Element Method. For each element, the numerical integration is conducted every calculation step. Thus, the integration takes a large share of the calculation time cost. On the other hand, the accuracy of numerical is depended on the element shape. For elements with regular shape, it only requires a small number of integration points to get an accurate result. But for those with huge distortions, more integration points is required[5] and usually hard to analyze.

1.3 FEM and Machine Learning

With the development of computer technology, statistical or data-driven method have great progress in recent years. Machine learning especially deep learning techniques have been applied to solve varies complex problems successfully. These problems include image processing, target detection, speech recognition, etc. The deep learning methods are proved to be more efficient and accurate when comparing to the traditional methods in these areas. [6][7] The deep learning techniques establish a surrogate model by training a neural network using existing data. The model maps the input (observed data) to solutions, which could be used to solve inverse problems. Thus, a much wider class of problems that were believed to be too complex to tackle are solved by using these techniques. It is deserved to apply the Cutting-edge machine learning techniques in computational mechanics fields such as FEM, which are possible to improve computation efficiency and accuracy.

The FEM approximates the unknown function over both spatial and temporal domain. It divides a large system into many small parts that are called finite elements. These finite elements are firstly modeled by simple equations. Then being assembled into a larger system of equations which could model the entire problem. With variational methods from the calculus of variations, a solution is approximated by minimizing an associated error function.FEM is relatively efficient and precise. Also, the boundary conditions are easy to deal with [8].

Machine learning is a large variety of statistical methods. It includes regression (Linear regression, ridge regression, lasso regression), clustering (k-nearest neighbors), order-reducing (Principle Components Analysis), classification (Logistic Regression, support vector machines), neural networks and ensemble methods [9]. Given enough training data, these techniques could conduct different kinds of tasks using statistical methods. By tuning a number of values hyperparameters, the behaviors of methods could be controlled. Deep learning is an important branch of machine learning based on artificial neural networks. It could recognize implicit features from a large amount of training data and classify them into different classes [10].In deep learning, a convolutional neural network (CNN) is a class of deep neural networks, most commonly applied to analyzing visual imagery. Three main types of layers are used to build CNN architectures: Convolutional Layer, Pooling Layer, and Fully-Connected Layer (exactly same as regular Neural Networks). LeNet and Alexnet are typical architectures in the field of Convolutional Networks.

In computational mechanics field, by using machine learning methods, a relationship between activation and response could be established. For example, a well trained neural network could be a surrogate model for Finite Element Methods or optimizing some process of them. Researchers have done lots of work in this area, a current research review would be shown in the next section.

2 BACKGROUND RESEARCH

Machine learning techniques have been widely used in computational mechanics field in recent years, to improve the performance of traditional numerical methods, especially in FEM [11].

A data-driven computing paradigm was proposed by T. Kirchdoerfer et al as a proof of concept for applying the data-driven method in FEM testing on a linear elasticity problem [4]. The method described in [9], G.Capuano et al used regression and support vector machine to generate a direct relationship between the element state and its forces using finite element data. To achieve real-time structural topology optimization, X.Lei et al proposed a moving morphable component (MMC)-based explicit framework for topology optimization [12]. Machine learning methods were used to improve numerical quadrature for FEA [10], Oishi et al proposed a method to enhance the numerical quadrature rule using deep learning. The neural network is also be applied to the load transfer paths analysis on plate structures. The deep learning method shows higher efficiency and accuracy comparing to FEM [13]. Some researchers try to apply a deeper neural network into the computational mechanics area. A deep convolutional neural network was developed [14] to improve the predicting accuracy of eigenvalue problems in mechanics. A deep-autoencoder was applied to approximate the large deformations of a non-linear, muscle actuated beam, which improve the efficiency real-time finite element simulations [15]. L.Liang et al proposed a deep learning-based approach to estimate stress distribution [16]. The approach includes an ML-FE surrogate to conduct fast simulations. Deep learning technique is also used to improve Finite Element Analysis problem such as stress analysis. Z.Nie et al implemented a CNN (Convolutional Neural Network) based method to conduct stress field prediction, which achieved higher accuracy compared to traditional FEA [17].

These works show a strong potential in applying machine learning methods into traditional Finite Element Methods. In this project, we would focus on optimizing some computational heavily processes of FEM. The specific processes would be described in the next sections.

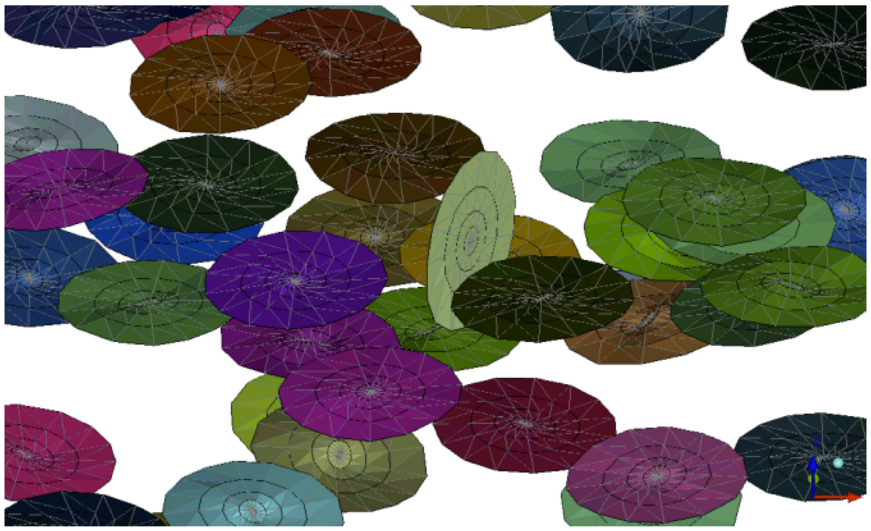


Figure 1: Detail of the fracture pattern with FEM[3]

3 SOFTWARE DESCRIPTION

The software developed in this thesis is a novel smart FEM platform which combined machine learning techniques with the traditional deterministic method. To specific mechanics problem, this software is able to provide solutions more efficiently with the same accuracy than the traditional method.

The software is developed based on Python3.7. The main python packages included are Numpy (Version 1.16.3) and Scipy (Version 1.3.0). These two packages are implemented by highly efficient C code under the hood, which provides strong support for scientific computing.

For traditional deterministic method part, the FEM is developed using Galerkin methods. Different kinds of element such as tetrahedron, hexahedron, etc. are also developed for different application scenarios. Notably, a novel numerical integration (smart integration) is developed for the software, which is based on Gauss-Legendre integration and machine learning techniques.

For machine learning part, an untrained network and pre-trained neural network are included. The networks could improve the efficiency of numerical integration given specific tolerance by predicting the optimal parameters. The main developing tool for this part is PyTorch (version 1.1). It is a widely used Python-based machine learning framework that provides two high-level features: Tensor computation with strong GPU acceleration; Deep neural networks built on a tape-based auto-grad system [18]. In this project, the codes related to machine learning techniques would be based on PyTorch.

3.1 Software Development Life Cycle

In this section, the technical details of the software will be introduced, including the architecture of the software, the software functionality and the novelty points in the project. The architecture part describes the modules division of the software as well as the software workflow. In the functionality part, the technical back-ends of each module are introduced in detail, including smart integration, neural network and Finite Element Method related parts. Finally, novelty and creativity will also be highlighted in this section.

3.1.1 Software architecture

The software architecture is showed in the graph. There are three main parts: FEM, Element and Smart Integration. The workflow of the software could be divided into two parts: the offline part and the online part.

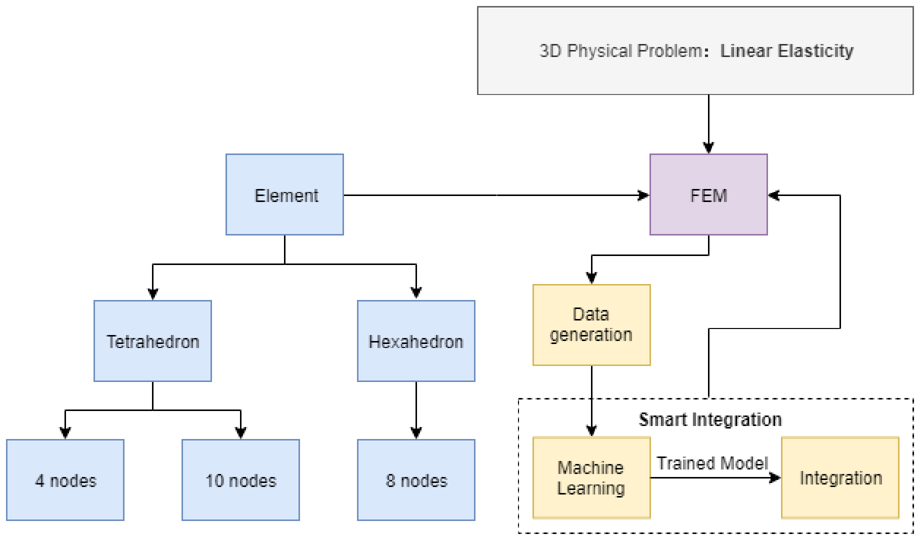


Figure 2: Software architecture

Offline Part The offline part is conducted only once (neural network training) for each specific problem. The 3 dimension physics problem serves as the input for the software. For this specific problem, the software generates a huge amount of selected type elements with different coordinates values. With each element, the optimal number of integration points is calculated as the label for this data point. All these data consist of the training dataset and feed to the machine learning class for network training. Then the machine learning class would output a trained network. This model would be stored and for further use in the online part.

Online Part The online part would be executed every step when conducting the Finite Element Method. The trained neural network would predict the number of integration points for each numerical integration to improve the efficiency of the calculation.

The FEM class is an implementation of the Finite Element Method for 3 dimension problems. The methods in the class include discretization (strain matrix, stiffness matrix, etc.), coordinate transformation (Shape function and its derivative, Jacobian, etc.).

The Element class describes different types of elements used in the Finite Element Method. There are two main types of element developed in this software: Tetrahedron and Hexahedron. For tetrahedron, both 4-node constant strain tetrahedron (CST) and 10-node quadratic tetrahedron are developed in sub-classes of Element respectively. All types of element in the software are isoparametric, which are efficient for computer implementation.

The smart integration part consists of three classes: data generation, machine learning and integration. Data generation class could generate proper data with labels for network training. The machine learning class include all the methods for data loading, data pre-processing, network model, train and validation, etc. A trained model would be the output of the class. For Integration class, an n-dimension Gauss numerical integration method is developed. The number of integration points and corresponding weights are arguments of the function, which could be specified by the network.

3.1.2 Software functionality

Algorithm 1 N-dimension Tensor Integration

```
1: Initialize a Tensor integration object: TensorQuad($Integrand, ranges, n$)
2: Arguments: $ranges$ is the integration limit, $n$ is the number of integration
3: $val\_matrix = None$    ▷ Set the result holder to None, because of lacking dimension information
4: $maxdepth = len(ranges)$    ▷ Get the dimension of the integration
5: $Jac =$ Jacobian($ranges$)    ▷ Calculate the jacobian for limit transformation
6: if n is specified then
7:     $n = n_{specified}$
8: else
9:     $n = n_{default}$
10: end if
11: $x, w =$ Legendre($n$)    ▷ Calculate the integration points and weights using Legendre polynomial
12: $X =$ Transformation($x$)
13: for $w_{i+1}$ in $w_{List}$ do
14:     $w_i =$ TensorConstructor($w_i, w_{i+1}$)    ▷ Construct the weight tensor
15: end for
16: TensorBreaker($w, maxdepth, X$)    ▷ Do Gauss integration recursively
17: return $Jac * value\_matrix$
```

Algorithm 2 TensorConstructor

```
1: function TensorConstructor($source, new\_data$)
2:     $dimension\_lifter$ = Ones(len(new_data))    ▷ Construct a identity vector
3:     $dst$ = Tensordot(dimension_lifter, source, axes=0)    ▷ Increase one dimension for the source data
4:     for $i, val$ in $new\_data$ do
5:         $dst[i] = dst[i] * val$
6:     end for
7:     return $dst$
8: end function
```

Algorithm 3 TensorBreaker

```
1: function TensorBreaker($weights, depth, X, *args$)
2:     for $i, val$ in $weights$ do    ▷ $Weights$ is the tensor constructed by TensorConstructor
3:         if $depth$ is 2 then
4:             $size = weights.shape$
5:             for $k, j$ in $size_0, size1$ do    ▷ $X$ are the integration points, $val\_matrix$ is the result holder
6:                 if $val\_matrix$ is not $None$ then
7:                     $val\_matrix$ += $weights_{k,j}*$ Integrand($X_k^0, X_j^1, *args$)
8:                 else
9:                     $val\_matrix = weights_{k,j}*$ Integrand($X_k^0, X_j^1, *args$)    ▷ Initialize the result holder
10:                end if
11:            end for
12:        else
13:            TensorBreaker($weights_{i,:,:}, X_i^{depth-1}, depth - 1, *args$)    ▷ Call function recursively
14:        end if
15:    end for
16: end function
```

N-dimension Tensor Integration Gauss-Legendre quadrature which can evaluate exactly the $(2n - 1)^{th}$ order polynomial with n-Gaussian points is most commonly used given the accuracy and efficiency of calculations. The core algorithms are shown in algorithm 1 2 and 3.

The basic Gaussian integration rule could be described as:

$$\int f(\boldsymbol{x})dx \approx \sum_{i=1}^{n} w_i f(\boldsymbol{x_i}) \tag{1}$$

In our case, we focus on the 3 dimension integration problems, so the formula could be written as:

$$\int\int\int F(\boldsymbol{X})dV \approx \sum_{k=1}^{q}\sum_{j=1}^{p}\sum_{i=1}^{n} \boldsymbol{W}_{i,j,k} F(\boldsymbol{X}_{i,j,k}) \tag{2}$$

Where the $F(X)$ is the integrand, the X and W are integration points and corresponding weights. p, q, n are the number of integration points in three dimensions respectively. The scalar and one layer numerical integration are quite easy to implement because it's just a plain weighted summation. For multiple integrations, if we did it analytically, the procedure is doing the integration layer by layer. When focusing on the specific layer, the variables belong to other layers are treated as constant and do not require the assignment. But for a computer, it needs to know the values of all variables every time it calls the integrand.

To handle this problem, we consider to calculate the weights and the function assignment values separately, then match them up and do the weighted summation. To make use of the matrix manipulation of Python, a tensor like data structure (shown in Figure 4) is proposed to contains the weights and function values. 3-D is just a special case of the tensor and shown in Figure 3. The integration weights are assembled using the tensor data structure. And then multiplied with the corresponding function values.

Further, the tensor integration developed here also supports multidimensional-multivariate-multiple integration. By using matrix manipulation, it could integrate a multidimensional integrand (such as a matrix with different integrand in each entry).

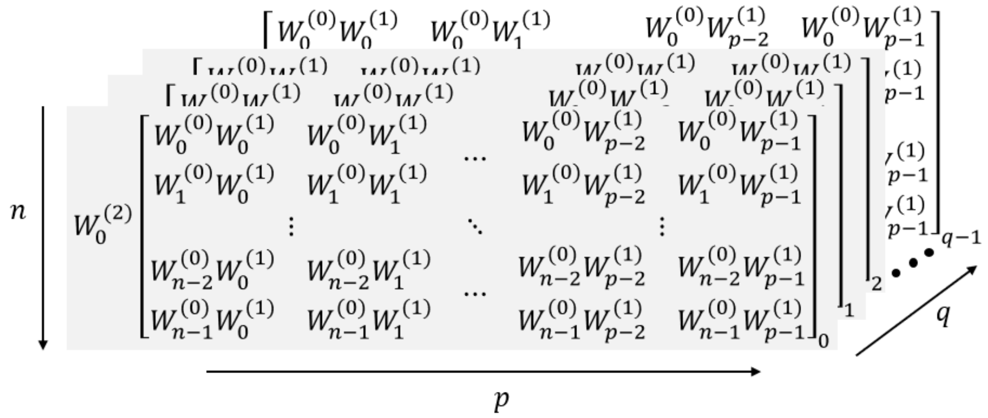


Figure 3: Tensor data structure, 3-D case

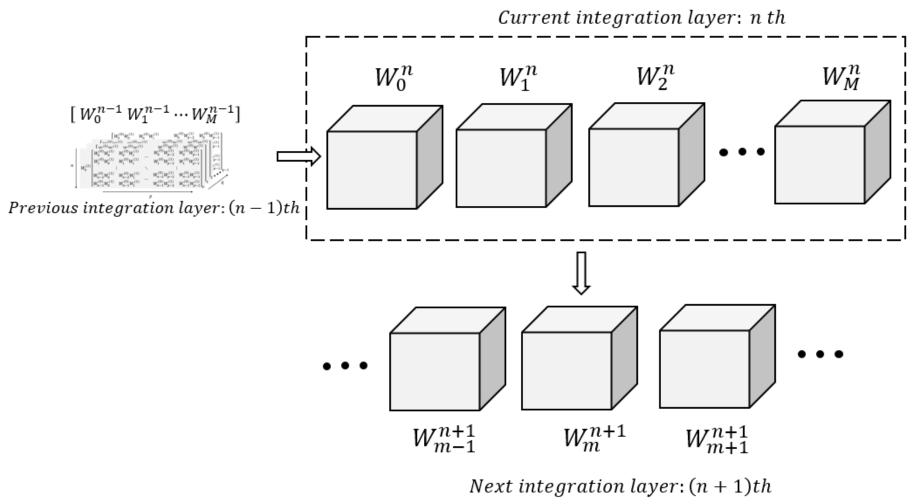

Current integration layer: n th
W_0^n
W_1^n
W_2^n
W_M^n
$[W_0^{n-1} W_1^{n-1} \cdots W_M^{n-1}]$
Previous integration layer: (n − 1)th
W_{m-1}^{n+1}
W_m^{n+1}
W_{m+1}^{n+1}
Next integration layer: (n + 1)th

Figure 4: Tensor data structure, general case

$$A \otimes B = C \tag{3}$$

$$A \in \mathbb{R}^a, B \in \mathbb{R}^b, C \in \mathbb{R}^{a+b} \tag{4}$$

$$a \otimes b = \begin{pmatrix} a_1, a_2, a_3 \end{pmatrix} \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = \begin{pmatrix} a_1b_1, a_1b_2, a_1b_3 \\ a_2b_1, a_2b_2, a_2b_3 \\ a_3b_1, a_3b_2, a_3b_3 \end{pmatrix} \tag{5}$$

- N-dimension Tensor Integration

  This algorithm contains the whole procedure of how this integration works. A tensor integration object should be initialized first. The required arguments are Integrand, integration limit. The number of integration points could be specified if needed. Then the Jacobian, integration points and corresponding weights would be calculated. Finally, the two functions TensorConstructor and TensorBreaker will be called and do the Gauss integration.

- Tensor Constructor

  The Tensor Constructor is used to increase the dimension of the data, the procedure is described in Figure 4. The function makes use of the tensor product property, the dimension of the product is the sum of two multipliers' dimension (shown in Equation 5).

- Tensor Breaker

  The Tensor Breaker function match the weights tensor and corresponding integration points, then do the weighted summation. To handle the multiple layer integration, the function does the integration recursively. To handle the multivariable problem, the function is developed with a variable-length argument which could pass the increasing number of variables along with the recursion. And by using the matrix manipulation, the function could handle multidimensional integrand properly.

Isoparametric element In real applications of FEM, the element distribute among the whole calculation domain. In the global coordinate system, each element has different coordinates value and distortion shape. It is trivial to integrate these elements into the global coordinate. There are different in global coordinates. On the other hand, in the local coordinate of each element, it's convenient to do the integration. The element could be transformed to isoparametric element (shown in Figure 10) using coordinate transformation.

In this software, all kinds of elements are transformed into its local coordinate using the Jacobian matrix. For 8-node hexahedron element, it is transformed from X,Y,Z to a base element with $\xi \in$ [-1,1],$\eta \in$[-1,1],$\zeta \in$[-1,1]. Especially, for 10-node tetrahedron element, after transforming to the base element, the integration range of the three-dimension variable is not independent, this causes difficulty when requiring high accuracy numerical integration. To handle this problem, one more transformation is conducted to transform the base tetrahedron element into two isoparametric hexahedrons. Then the integration range would be independent and Gauss-Legendre integration method could be applied.

Neural Network In this software, the main function of the neural network is a classifier. Due to elements with different geometry (such as large aspect ratio elements) requiring the different number of integration points, there would be accuracy and efficiency loss if only using the same number of integration points for all elements as the traditional method does. Thus, if could use a number of integration points for different elements adaptively, there will be a promising improvement on both calculation accuracy and efficiency.

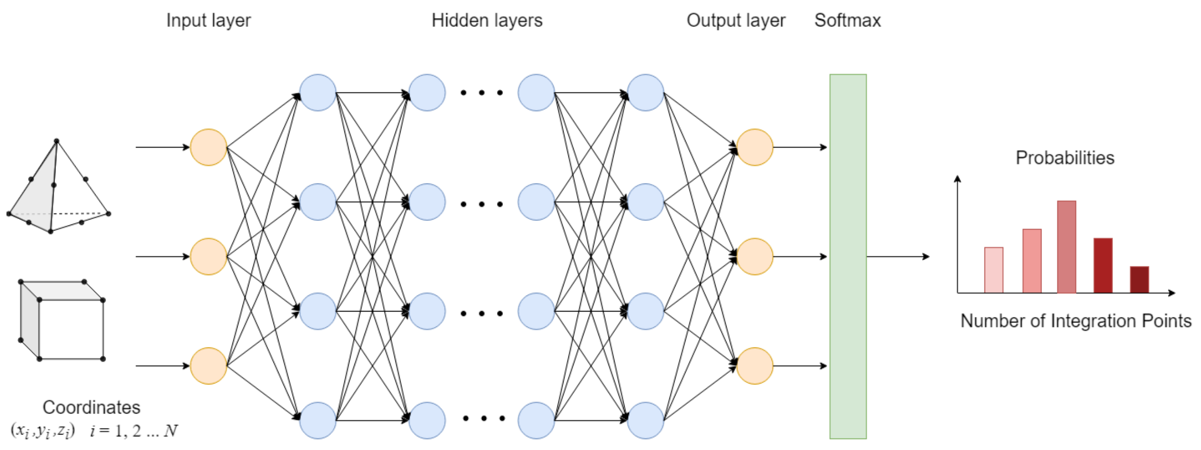


Figure 5: Neural Network

As shown in Figure 5, the input of the neural network here is the coordinates value of the element. The output of the network would be sent to a softmax function(a cross-entropy function). Then a probability distribution of the possible number of integration points needed is calculated. The number with the highest possibility would be chosen as the input number of the tensor quad.

Linear Elasticity Problem

$$[K] = \sum_{i=1}^{n_i} [K_i] = \sum_{i=1}^{n_i} \int_{v^i} [B^T][D][B]dv \tag{6}$$

Linear elasticity is a classic solid mechanics problem, which is quite suitable for testing finite element method codes. In the software, linear elasticity problem is used to testing and exploring the new method we developed.

The stiffness matrix is calculated by integrating the product of shape function derivative, material matrix and Jacobian determinant(shown in Equation 6). This process depends on numerical integration a lot, which would be the main focus. The exploration and analysis for this part would be described in the next section.

3.1.3 Novelty and creativity

Smart Integration In the traditional finite element method, number numerical integration points is a key factor which influences the accuracy and efficiency of the calculation. It has been shown that the elements of regular shapes, such as hexahedron with parallel square faces, can be integrated accurately using only a few integration points. But for those irregular and distorted shape elements, more points are needed to reach the expected accuracy. To handle this problem, some methods for specific problems are proposed. In the isogeometric analysis field, the NURBS-enhanced finite element method is usually applied[19]. In this method, the basis function is more complex with higher order. There are many enhanced numerical integration methods are proposed in this area, such as the reduced Bezier element quadrature rules for isogeometric analysis[20], an efficient matrix computation using sum factorization[21], efficient quadrature rules using the higher degree of continuity of NURBS or B-Spline basic functions on the element boundary and the numerical integration of trimmed elements[22]. These methods are restricted in a specific domain. In this project, a more general method called smart integration is implemented. This method could be applied to a range of problems which involves numerical integration.

Multiple-Multivariate-Multidimensional Tensor Integration Current numerical integration methods in Scipy and Numpy include *fixed_quad*, *nquad*. The $fixed - quad$ method support the user-specified number of integration points but could only do one-layer one-variable scalar integration. The *nquad* could do multiple-multivariate but still scalar integration, and it doesn't support tensor integration as well as integration points specification. As a result, the current built-in methods are not satisfied with the project purpose. The tensor integration method developed in this project could support n-dimension tensor integration with both the user-specified integration points and corresponding weights. Also, the tensor integration shows a faster calculation speed compared to current Scipy built-in function, which will be discussed in a later section.

Table 1: Functionality comparison

	Multidimensional	**n, ω specified**	**Multivariate**	**Multiple layer**
fixed-quad	×	✓	×	×
nquad	✓	×	✓	✓
Tensor quad	✓	✓	✓	✓

New implementation integration method for 10-node quadratic Tetrahedron The triangular and tetrahedral elements are very widely used in the finite element method. The main numerical integration schemes for tetrahedron range from 1 point to 45 points in 3 three dimensions, which are suitable to a polynomial of a degree from 1 to 8 respectively. (Jinyun CMAME 1984) In this software, a high-order supported integration schemes based on Gauss-Legendre for tetrahedron is developed. In order to apply Gauss-Legendre rule on a tetrahedron, there is one more coordinate transformation for base tetrahedron element. The element will be transformed into two base hexahedrons elements so that the Gauss-Legendre rule can be applied. Numerical integration rules of a very high degree of precision can be obtained.

$$\int_{V_o} f(x,y,z)dV_o = \int_{V_1} f(r,s,t)\,|J_1|\,dV_1 = \int_{V_2} f(\xi,\eta,\zeta)\,|J_2|\,dV_2 \tag{7}$$

$$|J_1| = \left|\frac{\partial(x,y,z)}{\partial(r,s,t)}\right|, \quad |J_2| = \left|\frac{\partial(r,s,t)}{\partial(\xi,\eta,\zeta)}\right| \tag{8}$$

$$x = \sum_{i=1}^{n_e} N_i x_i, \quad y = \sum_{i=1}^{n_e} N_i y_i, \quad z = \sum_{i=1}^{n_e} N_i z_i, \quad where\ \boldsymbol{N} = \boldsymbol{N}(r,s,t), \ \ \boldsymbol{N} \in \boldsymbol{R}^{n_e} \tag{9}$$

$$r = \frac{1+\xi}{2}, \quad s = \frac{(1-\xi)(1+\eta)}{4}, \quad t = \frac{(1-\xi)(1-\eta)(1+\zeta)}{8} \tag{10}$$

Here x, y, z are the element node values in the original coordinate. r, s, t are the values in a isoparametric tetrahedron coordinate. And ϵ, η, ζ are values in the final isoparametric hexahedron coordinate. The coordinate transformation could be described by the Equation 7. The $|J_1|$ and $|J_2|$ are the determinant of two Jacob matrix (shown in 8) when conducting the coordinate transformation. The Equation 9 shows the transformation between x, y, z and r, s, t, which is also called the element shape function. The relationship between the isoparametric tetrahedron coordinate and isoparametric hexahedron coordinate is described by Equation 10.

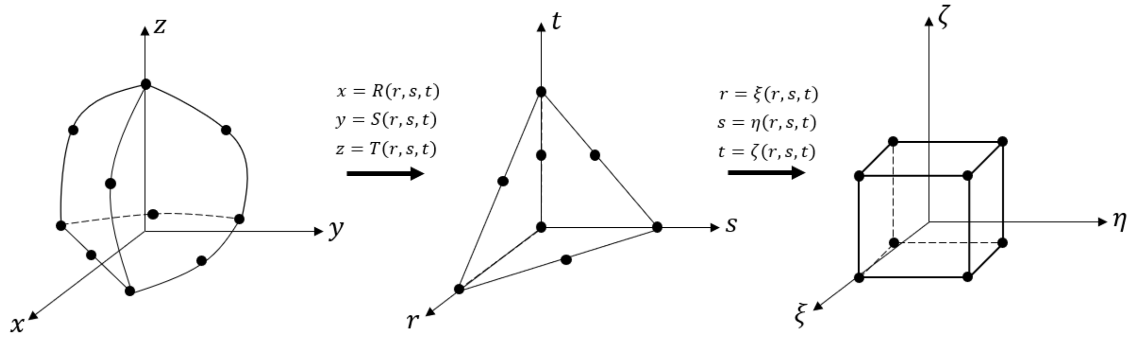


Figure 6: Coordinates transformation

3.2 Code metadata

The main language used for developing is python3.7.The code could be executed on both Windows OS, MAC OS and Linux. The software is also tested on High Performance Computing (HPC) system of Imperial College London cx1 except for the network training part. The cx1.hpc.ic only supports TensorFlow currently. So the network of the software is trained using Google Colab, which provides free GPU (graphic process unit) accelerating service.

3.2.1 Related libraries

The main depended libraries of this software are NumPy, Scipy and PyTorch. NumPy supports matrix manipulations, which is the basic calculation tool in this software. Scipy here is mainly used as the benchmark to our own tensor integration method. The PyTorch in the software supports the machine learning part including training, loading models, etc. Also, the codes in the machine learning part could be executed on GPU under the PyTorch framework.

Related libraries are listed below:

1. NumPy Version 1.16.3

   NumPy is the fundamental package for scientific computing with Python. It contains among other things: a powerful N-dimensional array object; sophisticated (broadcasting) functions; tools for integrating C/C++ and Fortran code; useful linear algebra, Fourier transform, and random number capabilities [23].

2. SciPy Version 1.3.0

   SciPy is a free and open-source Python library used for scientific computing and technical computing. SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering [24].

3. PyTorch Version 1.1

   PyTorch is a Python package that provides two high-level features: Tensor computation (like NumPy) with strong GPU acceleration; Deep neural networks built on a tape-based auto-grad system [18].

   It is a widely used Python-based machine learning framework. In this project, the codes related to machine learning techniques would be based on PyTorch.

3.2.2 Link

Code link and documentation: https://github.com/msc-acse/acse-9-independent-research-project-lunayeliu

4 IMPLEMENTATION and PERFORMANCE

In this section, the tests and validation results of each module in the software will be described. The integration part will describe the comparison between Scipy built-in function and our Tensor Integration method. In the element part, the three different kinds of elements are introduced and exploration of their property would be stated. Data generation part is essential to the machine learning part, which decides how well the model would work. In this part, the different data generation methods for hexahedron and tetrahedron are proposed in detail. And then there will be an analysis of the generated data in this part.

4.1 Integration

4.1.1 Comparison with Scipy

As mentioned in the previous section, a new n-dimension tensor integration method is developed in this software. It has new features which Scipy built-in functions $fixed\_quad$ and *nquad*. Here are some tests and comparisons between them in both accuracy and efficiency.

Accuracy When testing accuracy, the tensor quad is compared with the Scipy built-in function *nquad* using a test function below:

$$f(x,y,z) = \int_{-2}^{5}\int_{-3}^{4}\int_{-0.10}^{0.20} x^2y^3z^{10}dxdydz$$

The error convergence plot is shown in Figure 7. The blue line represents the difference between the integration result of the current number and that of the previous number. The yellow line shows the difference between the result calculated by *nquad*. The test function is a 10-order function and the Gauss integration is a method which could calculate (2n-1) order problem using n integration points. Thus, the error comes to be lower than 10^{-8} when the number of integration points is larger than 5 is reasonable.

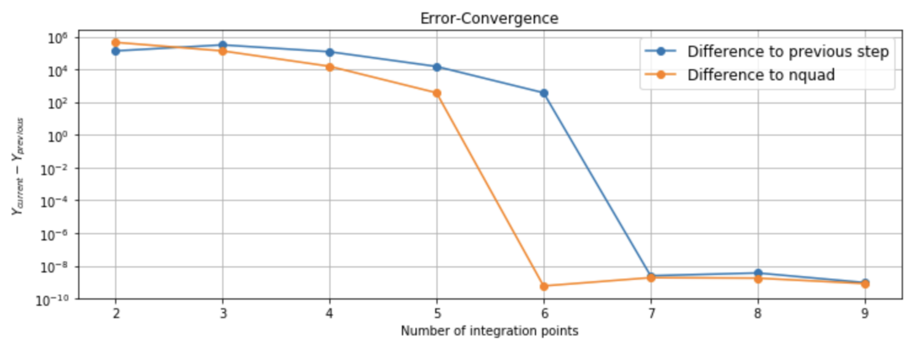


Figure 7: Error convergence

Efficiency The next figure 8 describes the time cost between *nquad* and tesnor quad with integration number equal to 6. There are 10 experiments have been conducted. It could be observed that the cost time of our method tensor quad, is about 20 times lower than Scipy nquad, which is quite a huge advantage.

In addition to the comparison, the cost time with the different number of integration points for tensor quad is also tested (shown in Figure 9). The number of integration points ranges from 1 to 19, and the cost time calculated from less than 10^{-3} with a number of integration points $4 \times 4 \times 4$.

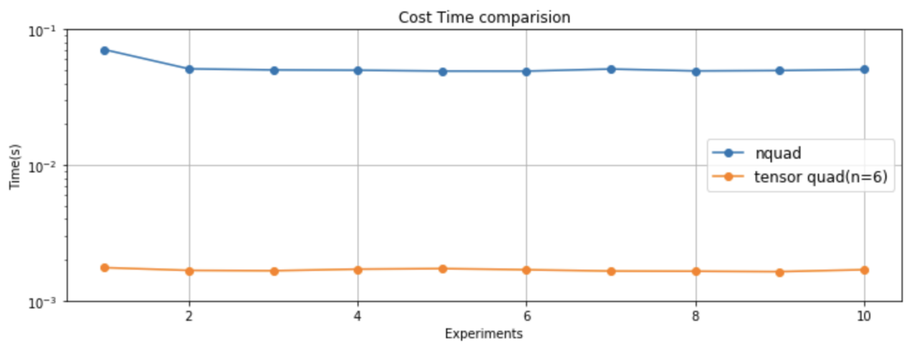


Figure 8: Efficiency comparison

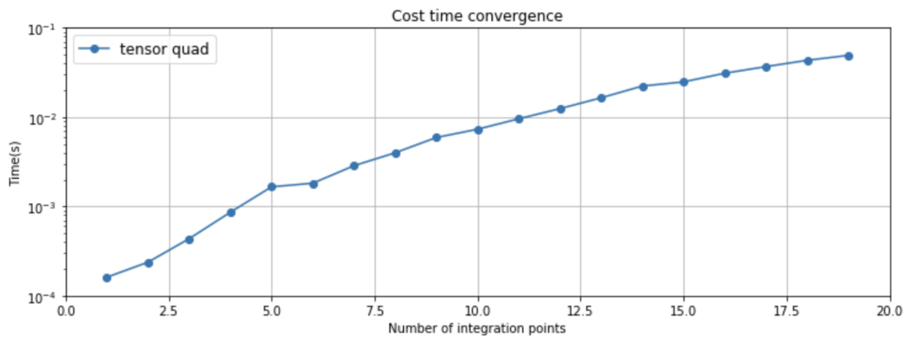
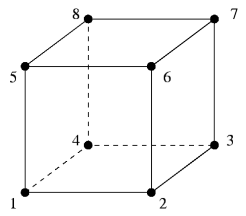


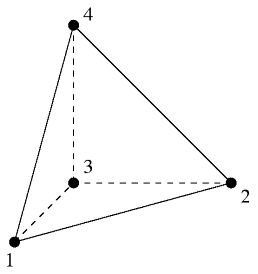
Figure 9: Efficiency convergence

4.2 Element

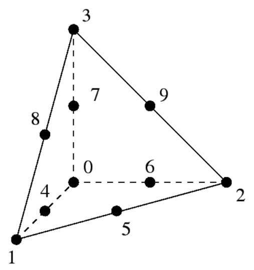
There are three kinds of element developed in this software: 4-node tetrahedron, 10-node tetrahedron and 8-node hexahedron. Among these elements, the 4-node tetrahedron is constant strain element, whose integration accuracy is not affected by the number of integration points. The 8-node hexahedron is bilinear



(a) 8-node hexahedral element[25]



(b) 4-node tetrahedron element



(c) 10-node tetrahedron element

Figure 10: Elements

element and 10-node tetrahedron is the quadratic element. For both these two kinds of elements, number integration points would effectively influence the error and accuracy. In addition, regular or irregular shape is another factor contributing to integration accuracy. Regular here means that the shape is close to the base element and irregular on the other hand, which has a large shape distortion.

4.2.1 Error Definition

$$error = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M}\left|k_{i,j}^{n} - k_{i,j}^{n_{max}}\right|}{max\left|k_{i,j}^{n_{max}}\right|} \tag{11}$$

The error is defined as a relative error scheme. $k_{i,j}$ here means the stiffness matrix. The n_{max} is the reference number of integration points, which is used to simulated the exact result. The n is the current integration number.

4.2.2 Error-Integration Points Convergence

The plot shows the convergence property of error and the number of integration points. We could observe that both the error of the regular elements converged very fast (less than 5 integration points). And the converged error is less than 10^{-12}, which reach high accuracy. As for the irregular elements (which means the element has a large distortion), the error shows a quasi-linear convergence property. It requires nearly 20 number of integration points for the irregular elements to converge.

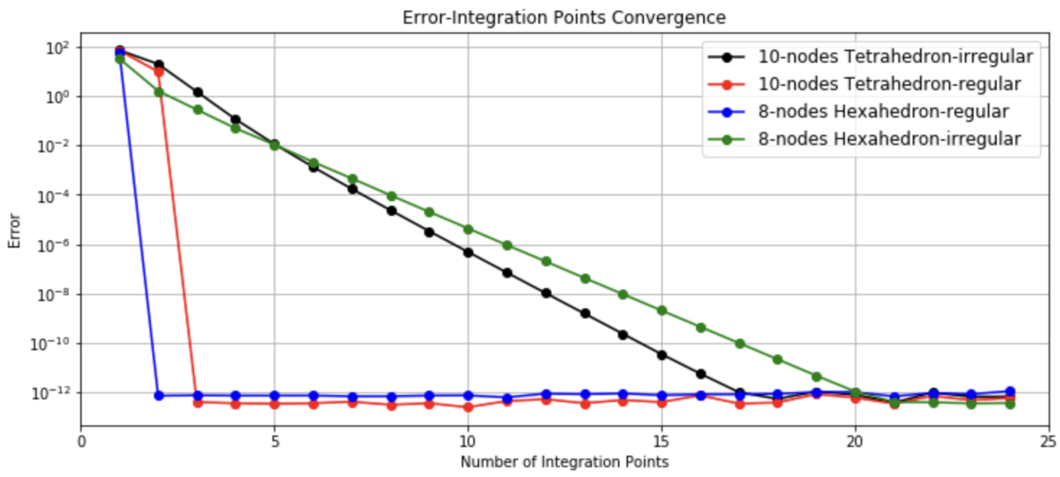


Figure 11: Error-Integration Points Convergence

4.3 Data Generation

Once choosing a certain type of element. Its property is defined by the shape of it, which means the relationship between nodes described by the coordinate values. The shape is independent on size, location and orientation. Therefore, the coordinate value is the only key parameter, which decides the number of integration points.

4.3.1 Hexahedron Generation

A number of 8-node hexahedron elements with various shapes are generated. A standard cube (regular shape) is used here as a reference. The coordinate values of other elements are varied using a random number range from [0.1, 0.5]. To model the distortion of the element, a control factor c is defined range from a uniform distribution [0.5, 1.0] and a maximum change value is defined discretely as d 0.1, 0.2, 0.3, 0.4, 0.5. The total value change is modeled by the product of the control factor and the maximum change:

$$\begin{aligned}
&1:(0,0,0), \quad 2:(1\pm cd,0,0), \quad 3:(1\pm cd,1\pm cd,\pm cd),\\
&4:(\pm cd,1\pm cd,0), \quad 5:(\pm cd,\pm cd,1\pm cd), \quad 6:(1\pm cd,\pm cd,1\pm cd),\\
&7:(1\pm cd,1\pm cd,1\pm cd), \quad 8:(\pm cd,1\pm cd,1\pm cd)
\end{aligned}$$

To prevent unaccepted distortions, the random number c is generated only once for the same node in an element, but for the different node, the random number is sampled independently from the distribution. In addition, the concave shape should be avoided because it violates the mapping uniqueness when conducting coordinates transformation using Jacobian.

1. Angle Restriction The angle between adjacent faces should be restricted in the range of [60, 120] degree.
2. Length Restriction Each edge should have a length range from [0.5, 2.0].

4.3.2 Tetrahedron Generation

The tetrahedron elements generation strategy is similar to hexahedron. However, due to the faces of tetrahedron are not parallel. It would be more sensitive than hexahedron. In addition to constricting the angles between the adjacent faces (and the range should be more wider), the position of the central node of each edge should also be taken into consideration. To avoid generating a non-convex tetrahedron, the random controller of the central node is set to be positive. At the same time, the central node should be located at a position should not locate in the region which bounded by the line between the two vertices and the two axis on the corresponding plane. Thus, the nodes are defined as follow:

$$\begin{aligned}
&0:(0,0,0), \quad 1:(1\pm cd,0,0), \quad 2:(0,1\pm cd,0),\\
&3:(0,0,1\pm cd), \quad 4:(0.5+cd,-cd,0.0), \quad 5:(0.5\pm cd,0.5\pm cd,0),\\
&6:(-cd,0.5+cd,0), \quad 7:(-cd,-cd,0.5+cd),\\
&8:(0.5\pm cd,cd,0.5), \quad 9:(cd,0.5\pm cd,0.5)
\end{aligned}$$

1. Angle Restriction The angle between adjacent faces should be restricted in the range of [20, 160] degree.
2. Length Restriction Each edge should have a length range from [0.5, 2.0].
3. Central Node Restriction On the plane constructed by two vertices and the origin, central nodes should not be located in the region which bounded by the line between the two vertices and the two axis.

4.3.3 Dataset Property

There are two datasets generated in this project. One for 8-node hexahedron, the other for 10-node tetrahedron. Both datasets has 10000 samples stratified by the maximum change value from 0.1 to 0.5 equally, which means there are 2000 samples for each value. The property would be analyzed from several aspects: number of integration distribution, aspect ratio distribution.

Aspect Ratio Distribution The aspect ratio is an important measurement of element quality. The ideal ratio is 1.0, which means the element has a perfect shape and the higher it is the worse shape. Here are plots describe the aspect ratio distribution of our dataset. For the hexahedron dataset, the percentage of elements with an aspect ratio larger than 10 is 0.96%. And the percentage of elements with aspect ratio lower than 3 is 90.60%. As for the tetrahedron dataset, all the elements have an aspect ratio smaller than 3. Therefore, both two datasets have good quality.

Dataset	Aspect ratio percentage(%)	
	$AR < 3$	$AR > 10$
10-node Tetrahedron	100%	0%
8-node Hexahedron	90.60%	0.96%

Table 2: Aspect ratio distribution

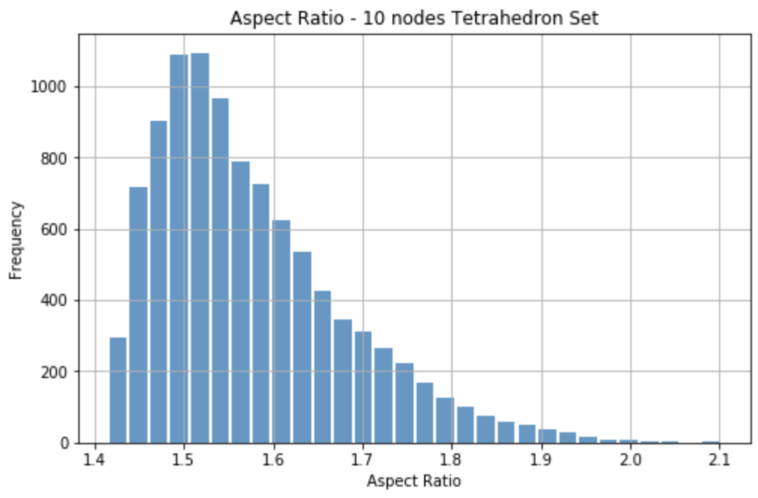


Figure 12: Aspect ratio distribution, 10-node tetrahedron

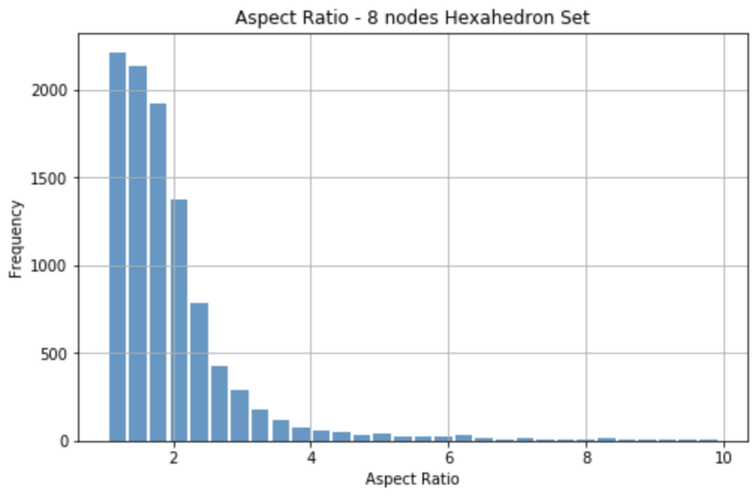
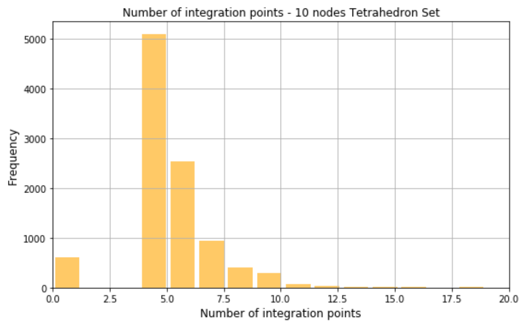


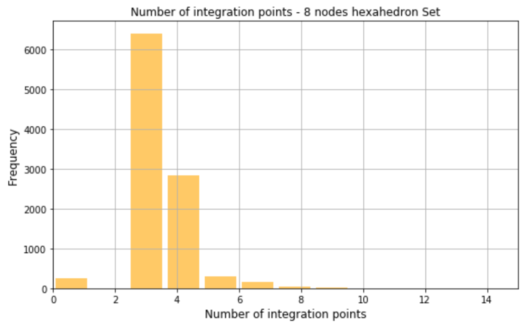
Figure 13: Aspect ratio distribution, 8-node hexahedron

The number of Integration Points Distribution This distribution describes the number of integration points an element requires. A special note here, the elements with a number 0 means its integration error could not converge, which indicates a really good quality. From Figure 14, we could find that although the tetrahedron dataset shows a better quality on aspect ratio. It generally still requires more integration points than hexahedron.

Figure 16 and Figure 15 shows the number of integration points distribution in a stratified view. The different plot represents the data with different maximum coordinate change value. We could see that the tetrahedron is more sensitive to the coordinate change value. It requires more integration points even with a narrower value change range. Also, the element failing to converge starts to appear when the maximum change value is only 0.20 (that of hexahedron is 0.30) and the number of element failing to converge is almost twice that of the hexahedron.



(a) 10-node tetrahedron



(b) 8-node hexahedron

Figure 14: Number integration points distribution

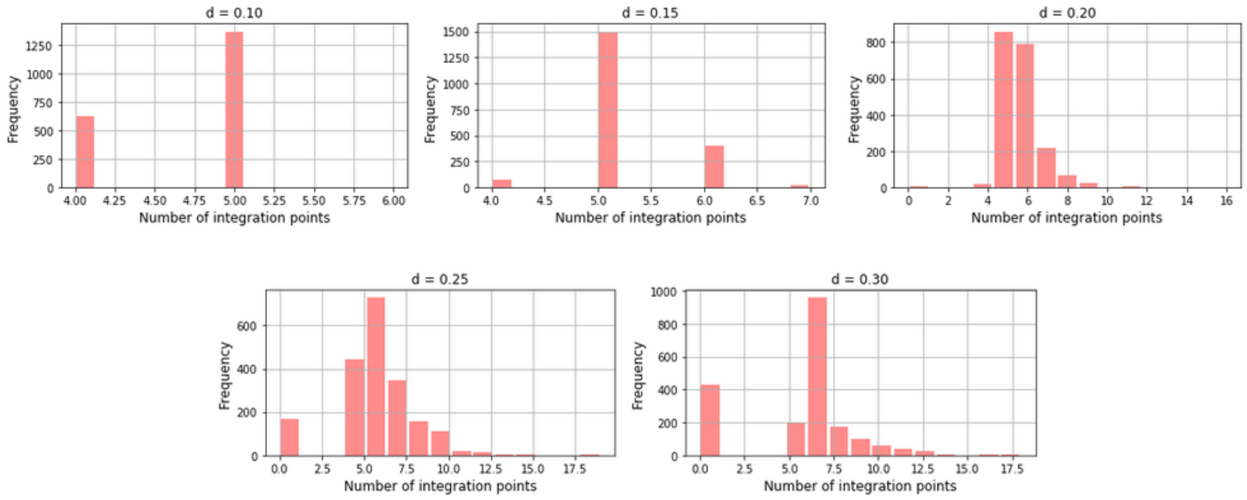


Figure 15: Number integration points distribution, stratified view, 10-node tetrahedron

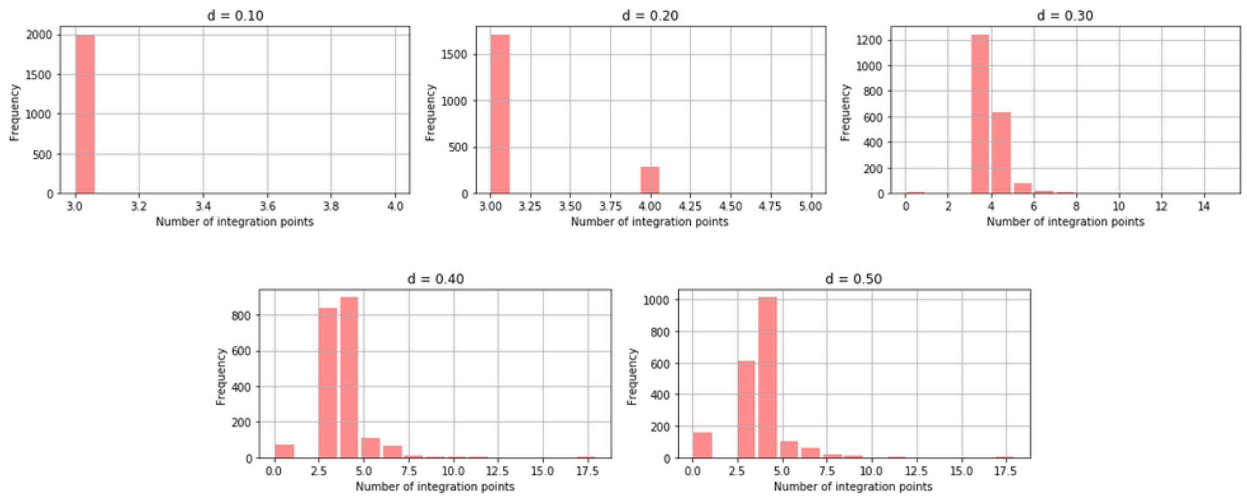


Figure 16: Number integration points distribution, stratified view, 8-node hexahedron

The relation between aspect ratio and the number of integration points It is worthwhile to discuss the relationship between aspect ratio and number of integration points. Here the average aspect ratios for elements required the same number of integration points are calculated. Figure 17 shows the results. We can see that the average aspect ratio increases along with the number of integration points, especially for the 8-nodes hexahedron. We can say that the elements with a distorted shape(larger aspect ratio) require more integration points.

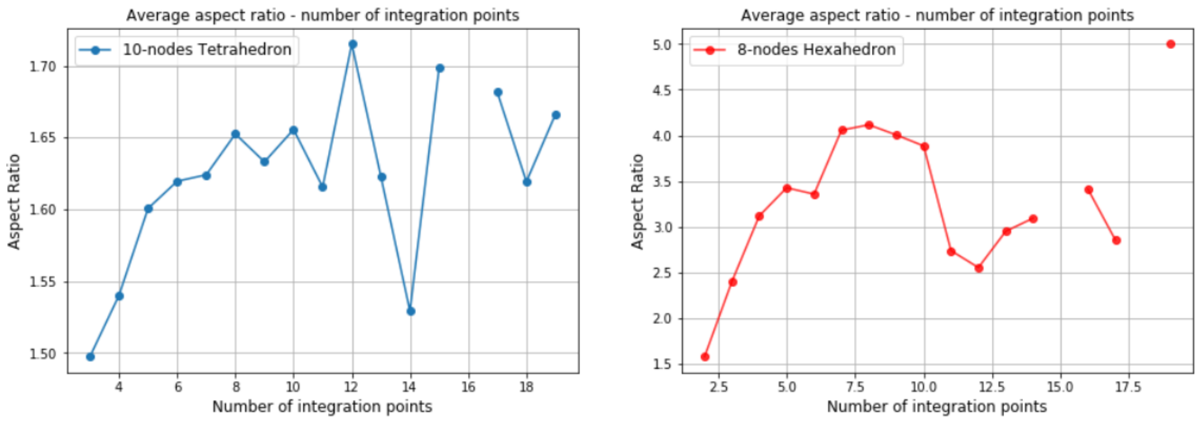


Figure 17: Relation between average aspect ratio and the number of integration points. The breakpoints in the figure indicate that there is no sample in the dataset with such a number of integration point.

4.4 Training and Validation

In this part, the training and validation processes of the neural network model would be described. Above all, the strategy would be discussed. In this software, a shallow learning strategy is applied. The reasons for not using a more deeper learning model could be listed below:

Efficiency Deep neural network architecture requires extreme huge computation resources as well as time-consuming. The network in our software is called frequently. Thus the extra computation time would accumulate and make the calculation much slower.

Training Deep architecture has been shown to be more sensitive to hyperparameters, which makes it more difficult to tune the network from over-fitting problems.

Data Training data is another important part of the neural network. A deeper network would require more training data. Otherwise, the generalization ability of the network would be poor. For standard machine learning datasets such as MNIST and CIFAR10, they provide a large amount (more than 50000) of low-resolution images. On the other hand, in scientific computing problems such as FEM, there are no such huge examples but less amount and higher dimension data. In addition, the features of scientific problems are more structural, which means an easier task for the network. Thus, shallow learning would be a better choice in this case.

4.4.1 Neural Network Model

Under the shallow learning strategy, a network model consisted of several fully-connected layer with a softmax (cross-entropy) normalization function is developed here. In order to find the proper number of hidden layers, experiments with 2 layers to 6 layers are conducted with two datasets respectively. The results would be discussed in the following section.

4.4.2 Training Parameters

As for the training parameters, the learning rate is 10^{-3} using an Adam optimizer, which is more efficient than Stochastic Gradient Descent optimizer in this case. The training and test batch size is set to 32.

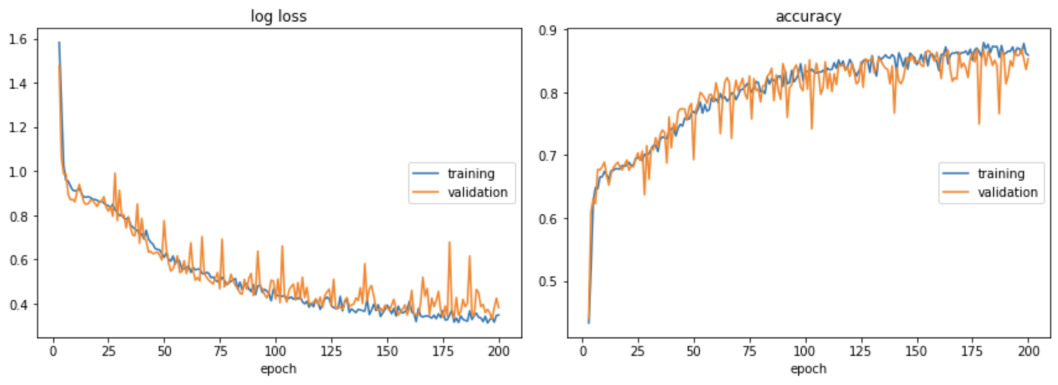


Figure 18: A sample of the training process

4.4.3 Training Results

Table 4 shows the training results using tetrahedron dataset. We can find that the network with 5 hidden layers performs the best, which has the highest classification accuracy. Although more hidden layers improve the accuracy when the number is less than 5. When it comes to 6 layers, the accuracy decreases comparing to 5. It might because the 6-layer model capacity is over large for our problem and there are redundant neurons in the network.

Number of hidden layers	Number of units per hidden layer	Classification Accuracy(%)	
		Training patterns	Test patterns
2	50	95.1	94.1
3	50	95.2	92.7
4	50	95.6	94.1
5	50	94.1	90.9
6	50	96.0	94.3
7	50	89.7	89.6

Table 3: Classification accuracy vs network architecture(hexahedral)

Number of hidden layers	Number of units per hidden layer	Classification Accuracy(%)	
		Training patterns	Test patterns
2	50	82.8	82.2
3	50	84.2	82.8
4	50	88.5	85.8
5	50	90.8	88.1
6	50	88.0	84.1

Table 4: Classification accuracy vs network architecture(tetrahedron)

		Number of integration points (estimated by ML)							
		0	3	4	5	6	7	8	9
Number of integration points (correct)	0	133	0	0	0	0	0	2	0
	3	0	3127	94	0	0	0	0	0
	4	0	113	1276	24	0	0	0	0
	5	0	0	4	112	4	0	0	0
	6	0	0	0	4	43	9	1	0
	7	0	0	0	0	1	13	6	0
	8	0	0	0	0	0	2	10	3
	9	0	0	0	0	0	0	8	1

Table 5: Confusion matrix for 8-node hexahedron

		Number of integration points (estimated by ML)						
		0	4	5	6	7	8	9
Number of integration points (correct)	0	292	0	0	0	0	0	0
	4	0	343	36	0	0	0	0
	5	1	48	2049	109	0	0	0
	6	4	0	44	1136	83	0	0
	7	1	0	0	24	385	40	0
	8	2	0	0	1	28	117	42
	9	2	0	0	1	3	7	57

Table 6: Confusion matrix for 10-node tetrahedron

5 CONCLUSION AND DISCUSSION

5.1 Conclusion and contributions

In this project, a light finite element method software with smart integration is developed. The smart integration is a combination of the machine learning and numerical integration modules. The neural network in the machine learning module can select the optimal number of integration points for numerical integration according to elements shapes. The network is trained on a small dataset with 5000 samples and the classification accuracy on unseen data can reach 94.3% for 8-nodes hexahedron element and 88.1% for 10-nodes tetrahedron element. The high accuracy proves that the smart integration method in this project can help select the optimal number of integration points adaptively. This helps improve the accuracy of FEM methods, especially for those with many large distortion elements.

In addition to the machine learning enhanced model, the multiple-multivariate-multidimensional tensor integration developed in this project shows a great advantage compared with the built-in Scipy integration functions. It outperforms the nquad on efficiency(20 times faster on test cases) and par on accuracy. The new tensor integration can also support multidimensional tensor integration(such as matrix integration), which can not be achieved by any Scipy built-in functions.

5.2 Limitation and future work

Currently, the model only trained on a small dataset with about 5000 samples. And the software only covers 3 kinds of elements(4-nodes tetrahedron, 10-nodes tetrahedron and 8-nodes hexahedron). It would be promising to train the model using a dataset with much more samples and extend the supporting element types. Also, the current work only applied to linear elasticity problems, it is deserved to further apply this method on more variety and more complex problems.

List of Figures

List of Tables

References

[1] S. Salimzadeh, A. Paluszny, H. M. Nick, and R. W. Zimmerman, "A three-dimensional coupled thermo-hydro-mechanical model for deformable fractured geothermal systems," *Geothermics*, vol. 71, p. 212–224, 2018.

[2] A.Paluszny, S.Salimzadeh, and R. W. Zimmerman, "Finite-element modeling of the growth and interaction of hydraulic fractures in poroelastic rock formations," *Hydraulic Fracture Modeling*, p. 1–19, 2018.

[3] A.Paluszny, R. Thomas, and R. Zimmerman, "Finite element-based simulation of the growth of dense three-dimensional fracture networks," *52 US Rock Mechanics / Geomechanics Symposium*, 2018.

[4] T. Kirchdoerfer and M. Ortiz, "Data-driven computational mechanics," *Comput. Methods Appl. Mech. Engrg*, vol. 304, pp. 81–101, 2016.

[5] S.Rajendran, "A technique to develop mesh-distortion immune finite elements," *Comput. Methods Appl. Mech. Eng*, vol. 199, pp. 1044–1063, 2010.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[7] A. Graves, A. rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," *IEEE international conference on acoustics, speech and signal processing*, pp. 6645–6649, 2013.

[8] D. L. Logan, *A First Course in the Finite Element Method, Fifth Edition.* Global Engineering, 2011.

[9] G. Capuano and J. J. Rimoli, "Smart finite elements: A novel machine learning application," *Comput. Methods Appl. Mech. Engrg*, vol. 345, p. 343–381, 2019.

[10] A. Oishia and G. Yagawa, "Computational mechanics enhanced by deep learning," *Comput. Methods Appl. Mech. Engrg*, vol. 327, p. 327–351, 2017.

[11] W. K. Liu, G. Karniakis, S. Tang, and J. Yvonnet4, "A computational mechanics special issue on: data-driven modeling and simulation—theory, methods, and applications," *Computational Mechanics*, 2019.

[12] X. Lei, C. Liu, Z. Du, W. Zhang, and X. Guo, "Machine learning-driven real-time topology optimization under moving morphable component-based framework," *Journal of Applied Mechanics*, 2019.

[13] Q. Wanga, G. Zhangb, C. Sunc, and N. Wua, "High efficient load paths analysis with u* index generated by deep learning," *Comput. Methods Appl. Mech. Engrg*, vol. 344, pp. 499–511, 2019.

[14] D. Finol, Y. Lu, V. Mahadevan, and A. Srivastava, "Deep convolutional neural networks for eigenvalue problems in mechanics," *arXiv [physics.comp-ph]*, 2018.

[15] F. Roewer-Despr, N. Khan, and I. Stavness, "Towards finite-element simulation using deep learning," *15th International Symposium on Computer Methods in Biomechanics and Biomedical Engineering*, 2018.

[16] L. Liang, M. Liu, C. Martin, and W. Sun, "A deep learning approach to estimate stress distribution: a fast and accurate surrogate of finite-element analysis," *J. R. Soc. Interface*, vol. 15, 2017.

[17] Z. Nie, H. Jiang, and L. B. Kara, "Stress field prediction in cantilevered structures using convolutional neural networks," *arXiv.org*, 2019.

[18] PyTorch. (2019) Pytorch website. [Online]. Available: https://pytorch.org

[19] R. Sevilla and S. Fernandez-Mendez, "Numerical integration over 2d nurbs-shaped domains with applications to nurbs-enhanced fem," *Finite Elem. Anal. Des*, vol. 47, pp. 1209–1220, 2011.

[20] D. Schillinger, S. Hossain, and T. Hughes, "Reduced bezier element quadrature rules for quadratic and cubic splines in isogeometric analysis," *Comput. Methods Appl. Mech. Eng*, vol. 277, pp. 1–45, 2014.

[21] P. Antolin, A. Buffa, F. Calabro, M. Martinelli, and G. Sangalli, "Efficient matrix computation for tensor-product isogeometric analysis: The use of sum factorization," *Comput. Methods Appl. Mech. Eng*, vol. 285, pp. 817–828, 2015.

[22] A. Nagy and D. Benson, "On the numerical integration of trimmed isogeometric elements," *Comput. Methods Appl. Mech. Eng*, vol. 284, pp. 165–185, 2015.

[23] Numpy. (2019) Numpy website. [Online]. Available: https://www.numpy.org

[24] SciPy. (2019) Scipy website. [Online]. Available: https://www.scipy.org

[25] Eight-node brick element (c3d8 and f3d8). [Online]. Available: http://http://web.mit.edu