Failure of laminated composite materials - probabilistic analysis and machine learning

B. Tech Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Anmol Deep (160103011)

Dhiraj Mittal (160103025)

Nitul Deori (160103054)

under the guidance of

Dr. Nelson Muthu

to the

DEPARTMENT OF MECHANICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI
GUWAHATI - 781039, ASSAM

Nov 2019

CERTIFICATE

*This is to certify that the work contained in this thesis entitled " **Failure of laminated composite materials - probabilistic analysis and machine learning**" is a bonafide work of* ***Anmol Deep (Roll No. 160103011)****,* ***Dhiraj Mittal (Roll No. 160103025)****,* ***Nitul Deori (Roll No. 160103054)****, carried out in the Department of Mechanical Engineering, Indian Institute of Technology Guwahati under my supervision and that it has not been submitted elsewhere for a degree.*

Supervisor: **Dr. Nelson Muthu**

Assistant Professor,

Nov, 2019 Department of Mechanical & Engineering,

Guwahati. Indian Institute of Technology Guwahati, Assam.

Acknowledgements

First and foremost, we would like to express our profound gratitude and indebtedness to Dr. Nelson Muthu, Assistant Professor, Department of Mechanical Engineering, IIT Guwahati for guiding us thoughtfully and efficiently through this project, giving us an opportunity to work at our own pace along our own lines, while providing us with very useful directions, constructive criticism and valuable suggestions whenever necessary.

Our sincere thanks to all our friends who have patiently extended all sorts of help for accomplishing this undertaking.

Also, We would like to offer our sincere thanks to all the other persons who knowingly or unknowingly helped us complete this project.

Contents

List of Figures

Chapter 1

Introduction

1.1 Why study Composite Failure?

Composite structures are being extensively used in present era due to their tailorable properties like high strength, stiffness and light weight. Its advantages are much significant, but it simultaneously poses the challenge of manufacturing the structure according to exact design specifications. Manufacturing is always subjected to significant variability due to unavoidable manufacturing imperfections (such as intra-laminate voids, incomplete curing of resin, porosity, excessive voids in matrix, ply thickness and fibre parameters), structural complexity. Operational and environmental usage conditions enhances the risk of damages and defects. Keeping everything in mind designer should come up with efficient and robust design with certain probability of failure under certain conditions, basically with high enough FOS, considering all the uncertainties in material parameters. But such considerations require extensive computational computations which is time consuming as well as costly. Hence modern mathematical stochastic paradigm are being used extensively today to imitate behaviours of computationally expensive methods, such as MCS. This thesis accords with that goal to achieve substantial uncertainty quantification and failure probability that is close around to the established standard MCS but with least computational sample points.

1.2 Metamodeling as a reliable approximation?

Computational simulations obviously provides accurate results but at the cost of time and expenses. To address such situations metamodeling techniques were developed as surrogates to the expensive simulation processes.

1.3 Organization of The Report

This chapter provides the outline and basic idea of what this report tries to infer. A thorough background and literature survey was done prior to constructing metamodel and python FEM code for different varying physical parameters that concludes in failure of a composite structure, Chapter 2 provides the knowledge and resources that were studied to go ahead in the project along with analysis of different research papers and journals and concludes the literature survey. Section 2.5 guides through the mathematical calculations and equation formation to get desired input data. Code is used to get output at certain input points, which further is used for testing and validation of metamodel technique chosen for the project.The observations and experimental data are depicted in Chapter 3. Chapter 4 provides the essence of the work done as it concludes results and analysis of the result with established standards. Future prospects of the same has been mentioned for possible advancements in research done.

Chapter 2

Literature Survey

2.1 Composites:

General Anisotropic Material

The symmetry of the stress and strain tensor($\sigma_{ij} = \sigma_{ji}$) reduces the number of independent elastic constants from 81 to 36. Again after considering elastic energy considerations of anisotropic materials which gives $C_{ij} = C_{ji}$ and $S_{ij} = S_{ji}$ the number of constants reduces to 21.

Specially Orthotropic Material

: The stress-strain relations for orthotropic material in terms of engineering constants can be written as

$$\begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \gamma_4 \\ \gamma_5 \\ \gamma_6 \end{pmatrix} = \begin{pmatrix} S_{11} & S_{12} & S_{13} & 0 & 0 & 0 \\ S_{21} & S_{22} & S_{23} & 0 & 0 & 0 \\ S_{31} & S_{32} & S_{33} & 0 & 0 & 0 \\ 0 & 0 & 0 & S_{44} & 0 & 0 \\ 0 & 0 & 0 & 0 & S_{55} & 0 \\ 0 & 0 & 0 & 0 & 0 & S_{66} \end{pmatrix} \begin{pmatrix} \sigma_1 \\ \sigma_2 \\ \sigma_3 \\ \tau_4 \\ \tau_5 \\ \tau_6 \end{pmatrix} \tag{2.1}$$

$where, S_{11} = \frac{1}{E_1}, S_{12} = \frac{-\nu_{21}}{E_2}, S_{13} = \frac{-\nu_{31}}{E_3}, S_{21} = \frac{-\nu_{12}}{E_1}, S_{22} = \frac{1}{E_2}, S_{23} = \frac{-\nu_{32}}{E_3}, S_{31} = \frac{-\nu_{13}}{E_1}, S_{32} = \frac{-\nu_{23}}{E_2}, S_{33} = \frac{1}{E_3}, S_{44} = \frac{1}{G_{23}}, S_{55} = \frac{1}{G_{13}}, S_{66} = \frac{1}{G_{12}}$

2.2 Variation in properties of Composites:

In most traditional structural materials, their mechanical behavior is assumed to be homogeneous and isotropic but mechanical properties of composite materials exhibit intrinsic statistical dependence and generally assumed to follow Normal distribution. In particular, in-homogeneity, anistropic characteristics and brittleness of the matrices and fibres affects strength of the composites at a significant level than any other property.And, usually the low fracture toughness of composite fibres is the result of energy dissipation of fibre or matrix interface and matrix ductility.Also, transverse tensile strength of fibre is reduced appreciably by local stress concentration around fibres.

The representation of a composite material requires a lot of parameters. Each of these parameters exhibit uncertainty and variability which mostly varies in Normal Distribution. As many design parameters play a role in determining the properties of the composites hence many sources of non-determinism have to be taken into account, which implies that many model parameters have to be represented by a relevant non-deterministic model, either in a probabilistic format through a probability density function or in a non-probabilistic format through an interval number or a fuzzy number.

Probabilistic methods are used to describe scatter in properties. Probability distribution functions (PDFs) can be established for all uncertain parameters, taking into account the correlation between different parameters. The result of the analysis can be interpreted in a statistical sense, and the probability of every output quantity depends on the input probabilities and their correlations. It is important that all these inputs must be validated in order for the result to allow for a statistical interpretation.

2.3 Effect of parameter scatter on material stiffness properties

The designer of a composite material has many degrees of freedom like the selection of raw materials for both the matrix and the fibre reinforcement, the architecture of the fibre reinforcement, the fibre volume fraction, the number of layers and the orientation of layers. For the analyst, this large set of design degrees of freedom translates into a wide range of model parameters, and inevitably also a wide range of uncertain or imprecise material data. Most composite materials with long fibre architectures exhibit orthotropic behavior, expressed as in Eq 2.1 When the load is applied along orientations x and y which include an angle $\theta \neq 0$ with the orientations 1 and 2, the compliance matrix D in the constitutive relation Eq.1 changes into the matrix $D^T = TDT^T$

$$[T] = \begin{pmatrix} \cos^2\theta & \sin^2\theta & 0 & 0 & 0 & 0 \\ \sin^2\theta & \cos^2\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ -0.5\sin 2\theta & 0.5\sin 2\theta & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cos\theta & \sin\theta \\ 0 & 0 & 0 & 0 & -\sin\theta & \cos\theta \end{pmatrix} \tag{2.2}$$

The T matrix is of same size as the compliance matrix D but having its entries functions of $sin\theta$ and $cos\theta$.This relation is used to express the variation of material stiffness constants for a change of orientation of the load. In the application of uncertainty, the misalignment of the fibre orientation with respect to the orientation of loading may be accidental, but the effect is significant. An imprecise placement of the fibre inevitably leads to a change of stiffness with respect to the nominal values.

2.4 Deterministic vs Stochastic Approach and Probabilistic Mechanics:

Aforementioned survey was focused on the fact that, how properties of Composite material vary, Be it fibre distribution, fibre interface bonds, Elastic Stiffness, density or irregularities.This dumps the idea of using Deterministic approach to compute output. As, In Deterministic the output of the model is fully determined by the parameter values and the initial conditions provided. There shouldn't be any inherent randomness in input parameters, That would contradict the basis of Deterministic model and might defy output from Deterministic model for every single simulation.

On the other hand Stochastic models possess some inherent randomness. The same set of parameter values and initial conditions will always provide output in a range of outputs, that is there is always a confidence value associated with each output. Hence It limits our research to Stochastic approach although deterministic approach is used to train our metamodel for some input values, so that surrogate model would be able to imitate the computationally expensive process.

2.5 Finite Element Analysis of 1D bar:

We used linear shape functions to approximate the residue and weight functions:

$N_1 = x_l$ and $N_2 = 1 - x_l$

$$U = S - W_d \tag{2.3}$$

here,$W_d = 0$ is assumed and S is the local stiffness matrix

$$\implies S = 0.5 \int_0^L \epsilon_x^T \sigma_x A dx \tag{2.4}$$

$$\implies S = 0.5 \int_0^L \epsilon_x^T E \epsilon_x A dx \tag{2.5}$$

To minimize U:

$$\frac{dU}{d[u]} = 0 \tag{2.6}$$

From here, we get local stiffness matrix:

$$[k]_e = \frac{AE}{L_e} \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix} \tag{2.7}$$

After this, we calculated the Global stiffness matrix in the assembling process. Now, we applied the boundary conditions to the equation [k][u] = [F] and solved for [u] to get the displacements at various nodes.

2.6 Finite Element Analysis of 2D bar:

Equations of Motion

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \sigma_{xy}}{\partial y} + f_x = \rho \frac{\partial^2 u}{\partial t^2} \tag{2.8}$$

$$\frac{\partial \sigma_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + f_y = \rho \frac{\partial^2 v}{\partial t^2} \tag{2.9}$$

Stress Strain Relations

$$\begin{pmatrix} \sigma_x \\ \sigma_y \\ \sigma_{xy} \end{pmatrix} = \begin{pmatrix} C_{11} & C_{12} & 0 \\ C_{21} & C_{22} & 0 \\ 0 & 0 & C_{66} \end{pmatrix} * \begin{pmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ 2\epsilon_{xy} \end{pmatrix} \tag{2.10}$$

Strain Displacement Relations

$$\epsilon_x = \frac{\partial u}{\partial x}, \epsilon_y = \frac{\partial v}{\partial y}, 2\epsilon_{xy} = \gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x} \tag{2.11}$$

Combining all the above equations

$$\implies \frac{\partial (C_{11}u_x + C_{12}v_y)}{\partial x} + \frac{\partial C_{66}(u_y + v_x)}{\partial y} = \rho \ddot{u} - f_x \tag{2.12}$$

$$\implies \frac{\partial C_{66}(u_y + v_x)}{\partial u} + \frac{\partial (C_{21}u_x + C_{22}v_y)}{\partial y} = \rho \ddot{v} - f_y$$

$$\implies \int_{\Omega^e} W_1 [-\frac{\partial (C_{11}u_x + C_{12}v_y)}{\partial u} - \frac{\partial C_{66}(u_y + v_x)}{\partial y} - f_x + \rho \ddot{u}] d\Omega = 0$$

$$\implies \int_{\Omega^e} -W_1 \frac{\partial F_{11}}{\partial x} d\Omega = \int_{\Omega^e} \frac{\partial W_1}{\partial x} F_{11} d\Omega - \oint_{\gamma^e} W_1 F_{11} n_x ds$$

$$\implies \int_{\Omega^e} -W_1 \frac{\partial F_{12}}{\partial y} d\Omega = \int_{\Omega^e} \frac{\partial W_1}{\partial y} F_{12} d\Omega - \oint_{\gamma^e} W_1 F_{12} n_y ds \tag{2.13}$$

where, $F_{11} = C_{11}u_x + C_{12}v_y$, $F_{21} = C_{66}(u_y + v_x)$

Similarly:

$$\int_{\Omega^e} -W_2 \frac{\partial F_{21}}{\partial y} d\Omega = \int_{\Omega^e} \frac{\partial W_2}{\partial y} F_{21} d\Omega - \oint_{\gamma^e} W_2 F_{22} n_y ds \tag{2.14}$$

where, $F_{21} = C_{66}(u_y + v_x)$, $F_{22} = C_{12}u_x + C_{22}v_y$

2.6.1 Approximation Step

$$u^e(x, y, t) = \sum_{j=1}^{n} u_j^e(t)\psi_j^e(x, y) \tag{2.15}$$

$$v^e(x, y, t) = \sum_{j=1}^{n} v_j^e(t)\psi_j^e(x, y) \tag{2.16}$$

Also,we assume weight functions W_1 and W_2

$$W_1(x, y) = \psi_i(x, y), W_2(x, y) = \psi_i(x, y) \tag{2.17}$$

For eth element, when we approximate u and v; we get the following two equations: Assuming static condition $\implies \ddot{u} = \ddot{v} = 0$

$$[k^{11}]^e[u]^e + [k^{12}]^e[v]^e = [F^1]^e + [Q^1]^e$$

$$[k^{21}]^e[u]^e + [k^{22}]^e[v]^e = [F^2]^e + [Q^2]^e$$

$$\begin{pmatrix} [k^{11}] & [k^{12}] \\ [k21] & [k^2 22] \end{pmatrix} \begin{pmatrix} [u] \\ [v] \end{pmatrix} = \begin{pmatrix} (F^1 + Q^1) \\ (F^2 + Q^2) \end{pmatrix} \tag{2.18}$$

Algorithm 1 The direct stiffness method of finite element assembly

```
Ensure: $A = [0], F = [0]$
  for all $elementse \in \varepsilon$ do
    $(A^e, F^e) \leftarrow elem(e)$
    for all $localdegreesoffreedomd_1ofe$ do
      $F(L(e, d_1)) + = F^e(d_1)$
      for all $localdegreesoffreedomd_2toe$ do
        $A(L(e, d_1), L(e, d_2)) + = A^e(d_1, d_2$
      end for
    end for
  end for
```

Here, L(i, j) is Location matrix. For equations involving single variable, L = Connectivity matrix. When More than one variable is involved, we have to generate the location matrix using the following algorithm:

Algorithm 2 Generate Location matrix

```
Require: $n_e \leftarrow No\_elements$
Require: $n_{pv} \leftarrow No\_Primary\_vars$
Require: $n_n \leftarrow Nodes/element$
Require: $M_C \leftarrow ConnectivityMatrix$
Ensure: $L = [O]_{n_e \times n_n \times n_{pv}},\ A = [\ ]$
  $j \leftarrow 1$
  for all $i\ in\ n_e$ do
    $A \leftarrow A + [j : j + n_{pv}]$
    $j \leftarrow j + n_{pv}$
  end for
  for all $i\ in\ n_e$ do
    $k \leftarrow 0$
    for all $l\ in\ n_n$ do
      $L(i, k : k + n_{pv}) = A(M_C(i)(l),\ M_C(j)(l))$
      $k \leftarrow k + 2$
    end for
  end for
```

2.7 Metamodeling Methodology:

Metamodel(often called "surrogate model") is a simplified model of the complex model(circuit, system, software or any entity). A model is an abstraction, highlighting properties of real world and metamodel is yet another abstraction of this model. Metamodeling is typically studying the output and input relationships and then fitting right metamodels to imitate that behaviour.

2.7.1 Computer Experiments and Surrogate Models

The objective of the surrogate model is to predict the values of deterministic function y(x), xϵR^n , yϵR^q , over a variable-space D, D $\subset R^n$

when its values are known only at a limited, finite number of sites contained in

$$\text{s} = s_1, s_2, ..., s_m | s_i \epsilon D$$

S is chosen according to how efficient you want your Surrogate model while m is usually bounded by operational constraints. Certain Algorithms are present to chose sample points efficiently over design space,And According to ref. [4] the problem of creation of a surrogate model for a computer experiment can be divided into 2 parts:

1. The Design problem: At which sites in $S = s_1, s_2, ..., s_m | s_i \epsilon D$ in our sample space should the data $Y = [y(s_1), ..., y(s_m)]$ (output) should be collected? As variation of features in training strongly depends upon the input data provided. It has to chosen carefully for efficient output.

2. The Analysis problem: How should the data be used or which methodology should be applied to make our surrogate model that will help us predict y(x) for all $x \epsilon D$ with reasonable accuracy?

Note that this methodology is applicable only when y(x) is a deterministic function.

2.7.2 Surrogate Models Survey

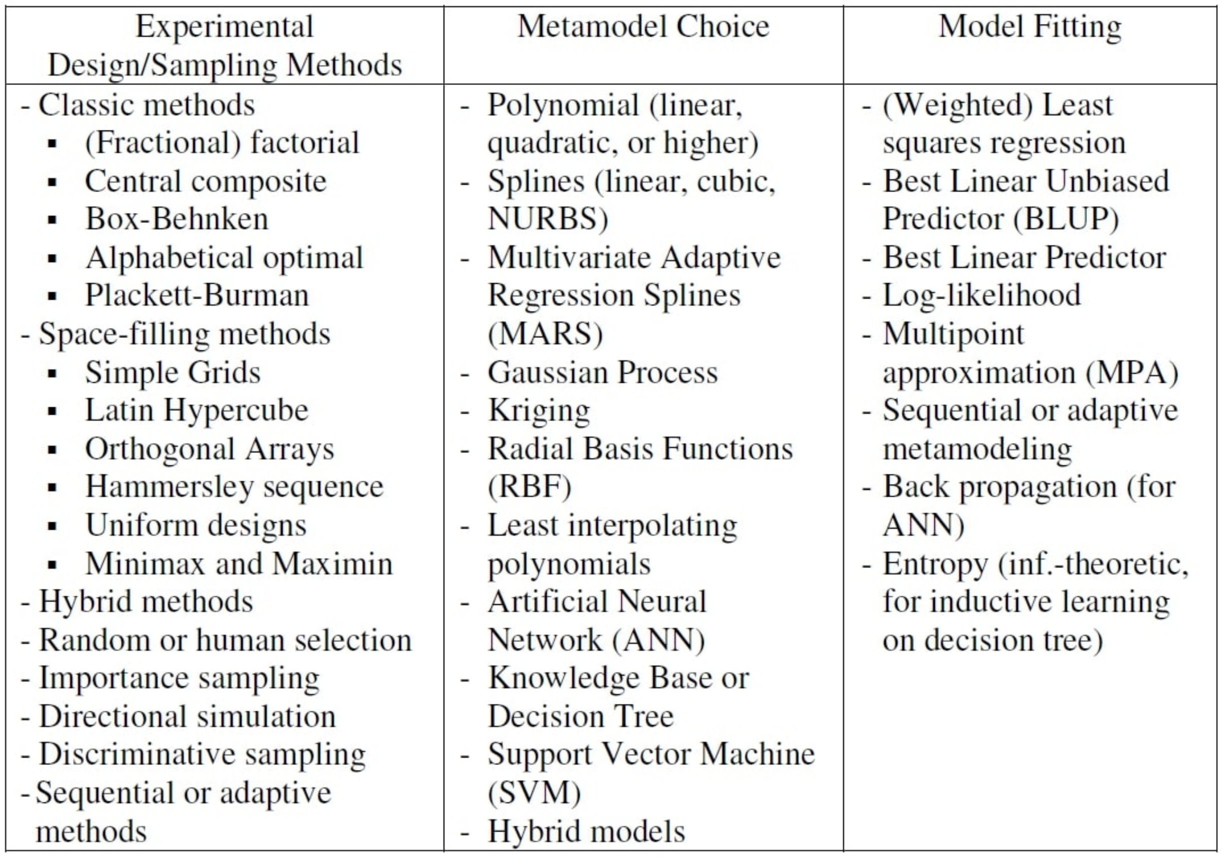
Approximation, or metamodeling, is the base unit to metamodel-based design optimization. The goal of approximation is to achieve a global surrogate model as accurate as possible at a cost significantly
Surrogate models are of various “forms” and varying in complexity. They can be broadly classified into two types according to [2]:

1. Functional Models
2. Physical Models

Physical models are simply mathematical models obtained by the ideology of modeling the actual process using physical laws. They may be physical approximations (for e.g. nodal analysis of a beam using finite modes) or mathematical approximations (for e.g. Taylor series and finite difference approximations). Hence CFD codes and FEM codes also qualify as surrogate models, while Functional models are mathematical constructions that simply mimic the behaviour of the output of the process. This is based merely on input and output and doesn’t focus on the governing equations or any physical basis. The only requirement of functional model is they need pre-sampled data and output correspondingly.We shall be focusing on functional type of surrogate models as we only want to mimic the output and not how the system is working.

There are a lot of sampling techniques and Approximation methods which are classified by[5]

From various journals and research it has been concluded that there is no hard and fast rule to select any best sampling method or metamodel, it depends on many factors. Which is discussed in coming sections in brief.



Experimental Design/Sampling Methods	Metamodel Choice	Model Fitting
- Classic methods	- Polynomial (linear, quadratic, or higher)	- (Weighted) Least squares regression
▪ (Fractional) factorial	- Splines (linear, cubic, NURBS)	- Best Linear Unbiased Predictor (BLUP)
▪ Central composite	- Multivariate Adaptive Regression Splines (MARS)	- Best Linear Predictor
▪ Box-Behnken	- Gaussian Process	- Log-likelihood
▪ Alphabetical optimal	- Kriging	- Multipoint approximation (MPA)
▪ Plackett-Burman	- Radial Basis Functions (RBF)	- Sequential or adaptive metamodeling
- Space-filling methods	- Least interpolating polynomials	- Back propagation (for ANN)
▪ Simple Grids	- Artificial Neural Network (ANN)	- Entropy (inf.-theoretic, for inductive learning on decision tree)
▪ Latin Hypercube	- Knowledge Base or Decision Tree	
▪ Orthogonal Arrays	- Support Vector Machine (SVM)	
▪ Hammersley sequence	- Hybrid models	
▪ Uniform designs		
▪ Minimax and Maximin		
- Hybrid methods		
- Random or human selection		
- Importance sampling		
- Directional simulation		
- Discriminative sampling		
- Sequential or adaptive methods		

Fig. 2.1 Metamodeling Techniques

2.7.3 Experimental Design/Sampling Methods

Gary Wang[5] confirmed that a majority among researchers was that experimental designs/sampling points for deterministic computer analyses for any metamodel should be space filling, i.e it must be evenly spread out, as it will capture properties and features that affects specimen failure. Four space filling sampling methods[5] are used more often relatively in literature, orthogonal arrays, various Latin Hypercube designs, Hammersley sequences[6], and uniform designs. Hammersley/Halton sequences and uniform designs belong to a family of more general group called low discrepancy sequences[8]. Hammersley/Halton sampling fares better uniformity than Latin Hypercube design methods.

From comparisons in [5], It was found that the Latin Hypercube design fares uniformity only in 1-D projection while the other methods tend to be more uniform in the entire space.

Also complexity of the function defines the "appropriate" sample size. In general, more the sample points more is the information of the function, however, at a higher expense. For low-order functions, after a certain sample size, increasing the number of sample points won't contribute much to the accuracy in the approximations. Hence a low descrepancy and more uniform sampling design method such as Hammersley is chosen to appropriately cover the design space provided.

Hammersley Sampling

Discrepancy analysis is nothing but measure of equidistribution in the points. Hammersley sampling method is one such low-discrepancy method.Also [6] that mapping Hammersley points with base of 2 to the surface of a sphere also give uniformly distributed directional vectors. From[6] points on 2d and sphere are uniform and can be seen in figure. (a) represents random sampling points plotted and (b) represents Hammersley sampling points with base p=2 plotted. similarly (c) random on sphere and (d) Hammersley with base p=2 on sphere.

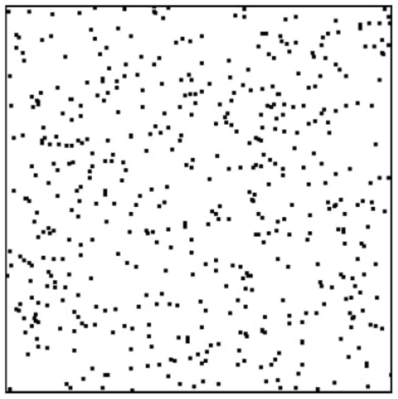


Fig. 2.2 Random Points Plot

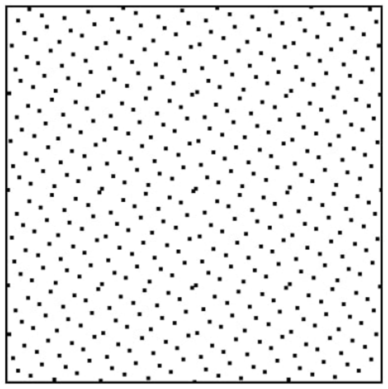


Fig. 2.3 Hammersley Points Plot

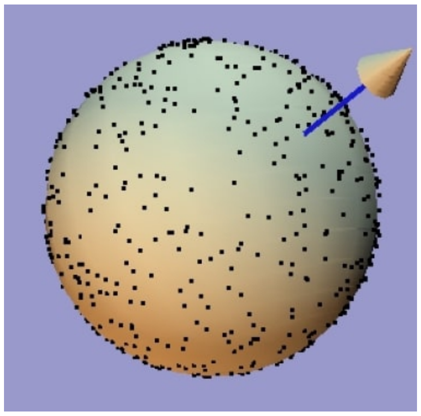


Fig. 2.4 Random Points Plot on Sphere

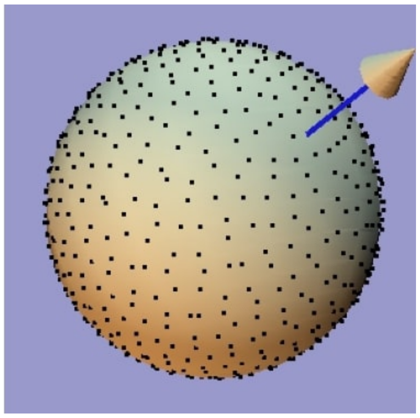


Fig. 2.5 Hammersley Points Plot on Sphere

Algorithm for Hammersley Sequence points

Each non-negative integer K can be expanded using a prime base p:

$$k = a_0 + a_{1p} + a_{2(}p^2) + ... + a_{r(}p^r) \tag{2.19}$$

where each a_i is an integer in [0,p-1]. Now define a function of k by

$$\phi_p(k) = a_0/p + a_1/p^2 + ... + a_r/p^r + 1 \tag{2.20}$$

If p = 2, the sequence of , for k = 0, 1, 2,..., is called the Van der Corput sequence Let d be the dimension of the space to be sampled.

Any sequence $p_1, p_2, .., p_d - 1$ of prime numbers defines a sequence $\phi_{p_1}, \phi_{p_2}, ...\phi_{p_d-1}$of functions, whose corresponding k-th d-dimensional Hammersley point is

$$(k/n, \phi_{p_1}(k), ..., \phi_{p_d-1}(k)) \tag{2.21}$$

for k = 0, 1, 2, ..., n-1. n is total number of hammersley points. For ϕ_p Alogirthm is given in A.1 of Appendix.

2.7.4 Metamodel

Metamodeling evolves from classical Design of Experiments (DOE) theory, in which polynomial functions are used as response surfaces, or metamodels. Besides the commonly used polynomial functions, From [5] sequential experiments were conducted for four different types of space filling sampling techniques namely, Latin hypercube, Hammersley sequence sampling, orthogonal arrays, uniform designs with comparable sample sizes with Four types of approximations models namely polynomial response surface, kriging models, radial basis functions, multivariate adaptive regression. And it was found that when accuracy was compared in terms of RMSE and absolute error(MAX) kriging with hammersley fared well with optimal sample points.Generally Kriging models are more accurate for nonlinear problems but difficult to obtain and use because a global optimization process is applied to identify the maximum likelihood estimators. On the contrary, a polynomial model is easy to construct, clear on parameter sensitivity, and cheap to work with but is less accurate than the Kriging model. Hence we would like to continue our discussion with Kriging Approximation technique

Kriging Approximation Model

Originally developed for applications in geostatistics, a kriging model postulates a combination of a polynomial model and departures of the form:

$$y = \Sigma \beta_j f_j(X) + Z(x) \tag{2.22}$$

where Z(x) is assumed to be a realization of a stochastic process with mean zero and spatial correlation function given by:

$$cov[Z(x_i), Z(x_j)] = \sigma^2 R(x_i, x_j) \tag{2.23}$$

where R is correlation. Gaussian Correlation function is used mostly. In our study, we use a constant term for $f_j(x)$ and a Gaussian correlation function with p=2 and k, θ parameters,

We mentioned Kriging above as a part of both regression models and radial basis functions. Indeed Kriging is in many ways a cross between the two. In this approach the underlying process is assumed to be a superposition of a linear model and departures from the linear model.

Actual Process = Linear model + Systematic departures.

2.7.5 Model Validation

A model is validated against a standard established method which in our case is Monte Carlo Simulation Method, Probabilities from our Meta model is validated against the probabilities from Monte carlo Simulation.

2.7.6 Model Fitting

After the sampling, we get some algorithmically selected points , on which simulation is done to get deterministic output for given input variables. After metamodel is trained with this data-set, it is compared with the data from simulation and data from metamodel. And after statistical and mathematical analysis can be done to check whether model is overfitted or underfitted. In some cases if the assessment is not satisfactory then we repeat sampling or sometimes input variables too.

Chapter 3

Experiment Results and Discussion

3.1 1D Beam Specimen

Problem Statement:Given a 1D bar with length L and Young's Modulus E. The bar is rigidly attached to a stationery sruface at one end and a point force of magnitude F is pulling onto the bar from the opposite free end. We have to calculate the change in its length after the bar comes into equillibrium.

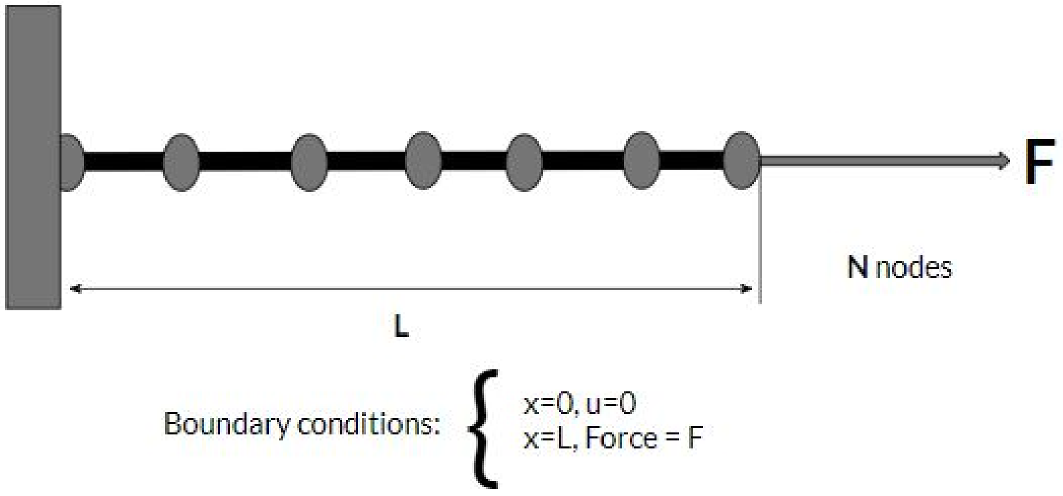


Fig. 3.1 1D Element

3.2 2D Bar Specimen

Problem Statement:Given a 2D bar with the following geometric parameters and material properties. The bar is rigidly attached to a stationery surface at one end and there are n, equally spaced point loads of magnitude F_0 pulling onto the bar from the opposite free end. Also, the thickness of the sheet is very less as compared to its length and breadth. We have to calculate the change in its geometry(length and breadth) after the bar comes into equilibrium.

l(along x-axis) = 1m, b(along y-axis) = 0.25m, h(along z axis) = 5×10^{-9}, ($l\ b$, $\frac{h}{l} << 1$)

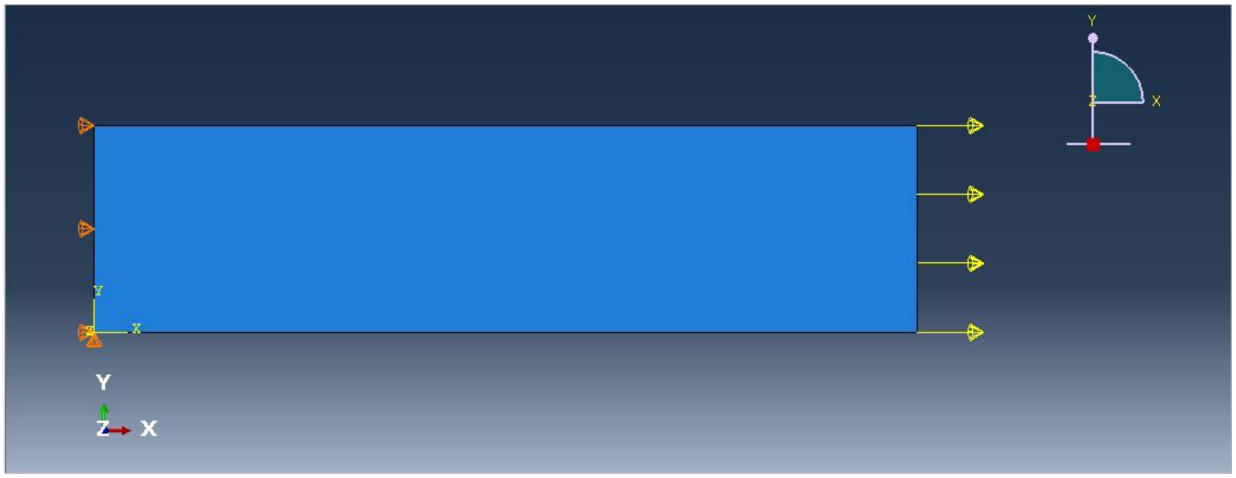


Fig. 3.2 A 2D bar element

Material property	Mean value	Variance
Young's Modulus	$210\times10^9 Pa$	$210\times10^9 Pa(10\%)$
Poisson's Ratio	0.5	0.05 (10%)

3.3 Computations and calculations

We performed Monte Carlo Stimulation of finite element analysis on the following problem 1000 times. Each time we picked a different values of E and ν. sampled from the normal distribution with the given parameters to span the sample space. We determined the maximum displacement magnitude for each iteration. Using actual experiments, we can

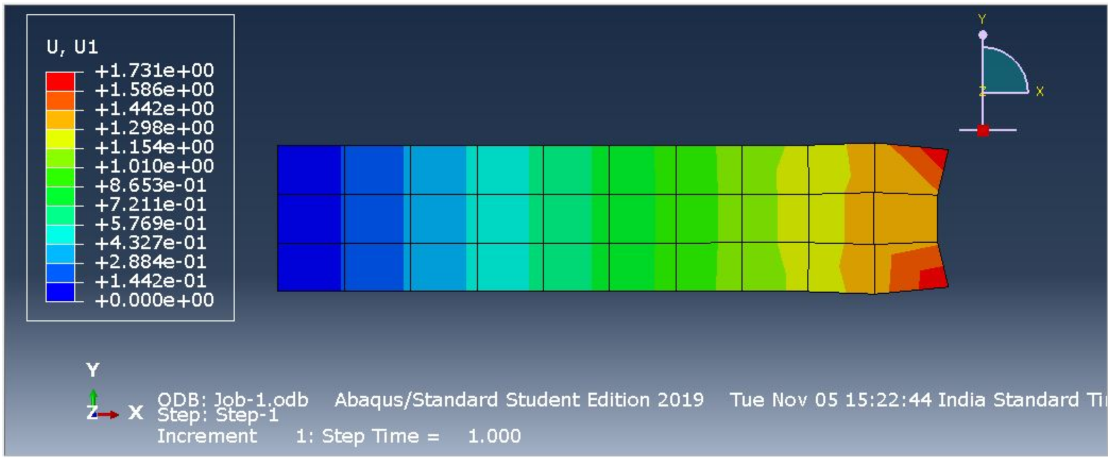


Fig. 3.3 Contour plot of u$_{net}$

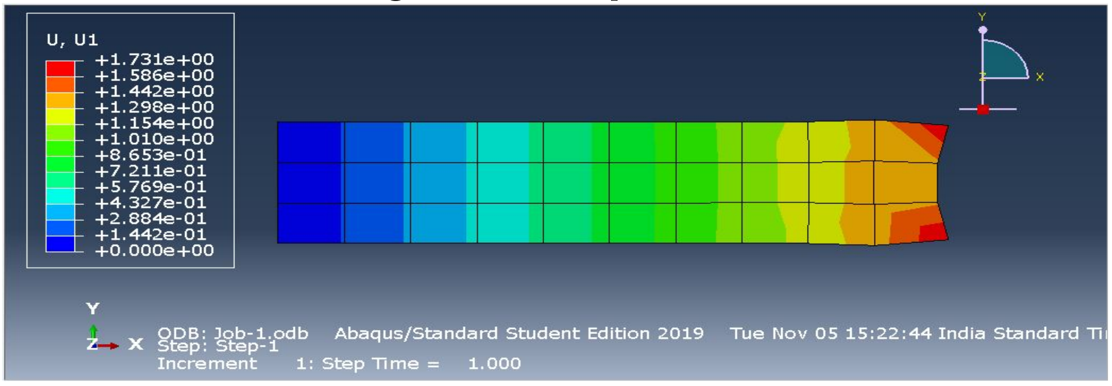


Fig. 3.4 Contour plot of u$_x$

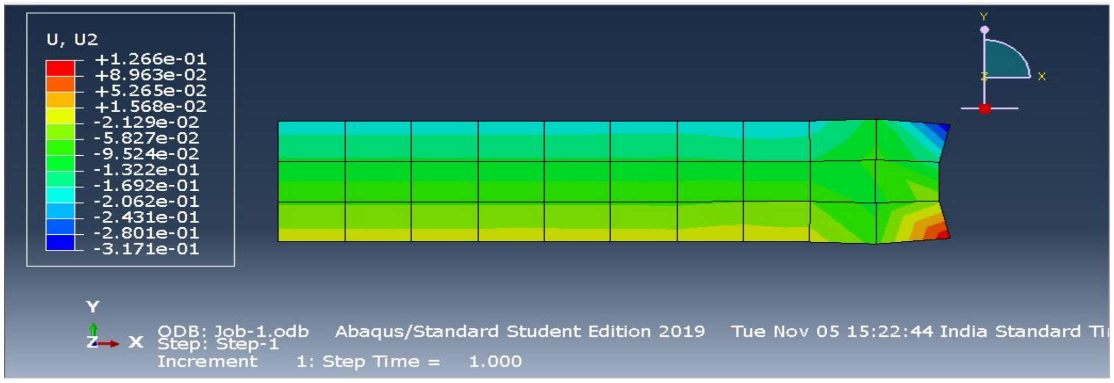


Fig. 3.5 Contour plot of u$_y$

determine the threshold displacements above which the material can fail. And for each property

Input data was created only at 50 points which were algorithmically generated from the code given in Appendix A.1. These hammersley sample points are uniformly distributed on the 2D surface. Kriging Metamodel[10] was then trained only for the these points and output as probability of failure was obtained based on the maximum deflection. This was validated against 1000 randomly generated points in MCS, and then testing them in our metamodel. 51% Failure probability was obtained for from out Metamodel while only 45% was obtained from Monte Carlo Simulation method.

Although the value from metamodel varied as different set of testing points were picked but it varied between 48% to 53% approximately. Hence It can be concluded that Kriging metamodel along with Hammersley sampling technique can be a safe alternate to traditional Monte Carlo Simulation method, some error in calculation from the metamodel can be neglected as compared to the high cost and time required for results obtained from 1000 points in Monte Carlo Simulation Method.

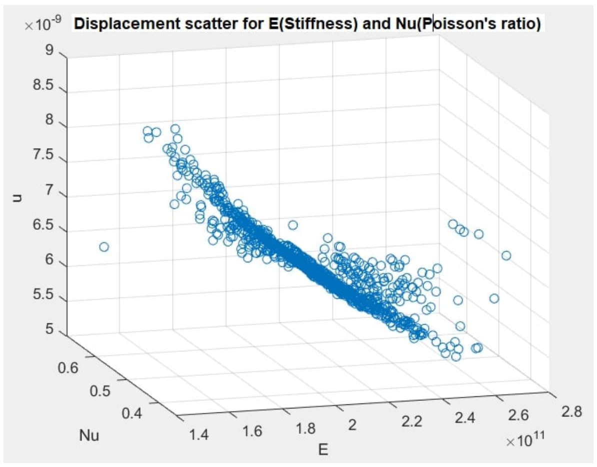


Fig. 3.6 Scatter for displacement MCS

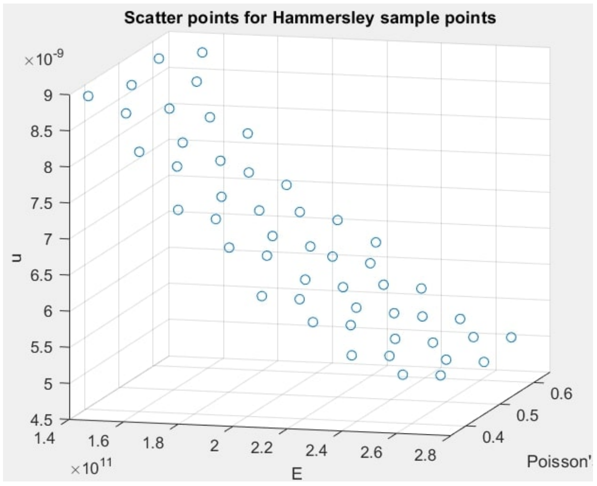


Fig. 3.7 Scatter for displacement from Hammersley Sequence points

Chapter 4

Conclusions

4.1 Summary

The project and research work carried out in this report provides us with an overview of metamodeling techniques and sampling techniques as an alternative to Standard Computational programs. Python codes for Fem was used inspite of using FEM or any computational program to get desired input fast and according to requirements. Which is both time and cost efficient. Metamodel was then build and validated for the available data. For 1000 samples taken MCS predicted 45% probability of failure while from metamodel, it predicted 51% probability of failure. For a metamodel it was quite good but further optimizations can be done in regard with the metamodel for better accuracy. It is hoped that the work done will be of great help to the researches and engineers who are just starting in this area, and also who are currently work on the similar projects, hope it act as an reference and inspiration.

4.2 Future Prospects

As you have observed in Fig. 3.6, there clearly is a pattern in the data obtained from Monte Carlo Stimulation of Isotropic Material. We hope that when we have even more variables involved in case of composites - such as E$_1, E_2, E_3$, ν_{ijk} etc, we will be able to capture the stochastic properties of the material in a much better way and create models to predict the failure probabilities with a much better accuracy. This project lies under an intersection of many domains of research. Our advancement of this project will include:

1. Cohesive Zone modelling of the composite material.
2. Extension of the existing FEA code to be able to capture different kinds of geometries and element properties.
3. Solving the 3D stress - strain - displacement equations for cohesive zone model using the code, and generating the failure data by Monte Carlo Stimulation and Hammersley sampling.
4. More optimization of the above combination by using different Correlation function or different type of Kriging Models, namely simple kriging, cokriging, Universal Kriging etc. This will effectively bring in more accuracy as well as less number of sample points required.

Bibliography

1. Simpson, T. W., Lin, D. K. J. and Chen, W., 2001, ”Sampling Strategies for Computer Experiments: Design and Analysis,” International Journal of Reliability and Application, 2(3), 209-240.

2. Jin, R., Chen, W. and Simpson, T. W., 2001, ”Comparative Studies of Metamodeling Techniques Under Multiple Modeling Criteria,” Structural and Multidisciplinary Optimization, 23(1), 1-13.

3. Andy Vanaerschot, Stepan Lomov, David Moens1, and Dirk Vandepitte1, ”Variability in composite materials properties”

4. Issac M. daniel, Ori Ishai, ”Engineering Mechanics of Composite Materials”,chap 2,4

5. G. Gary Wang*, S. Shan,” Review of Metamodeling Techniques in Support of Engineering Design Optimization”, Journal of Mechanical Design · April 2007, 1-16

6. Tien-Tsin Wong,Wai-Shing Luk,Pheng-Ann Heng, ”Sampling with Hammersley and Halton Points”

7. Timothy W. Simpson, Dennis K. J. Lin, and Wei Chen,” Sampling Strategies for Computer Experiments: Design and Analysis”,International Journal of Reliability and Applications, 1-25

8. MATLAB function reference ”https://in.mathworks.com/matlabcentral/fileexchange/29025-ordinary-kriging ”

9. NiKishkov, G.P., 2001, ”Introduction to the Finite Element Method”

10. Wang, Fang, Ding, Jun and Chen, Zhiqian, 2014, ”Statistical analysis of the progressive failure behavior for fiber-reinforced polymer composites under tensile loading”, Polymers, 6(1), 145-159

11. Béchet, Eric, Cuilliere, J-C and Trochu François, 2002, ”Generation of a finite element MESH from stereolithography (STL) files”, Computer-Aided Design, 34(1), 1-17

12. Elishakoff, I, Li, YW and Starnes Jr, James H, 1994, ”A deterministic method to predict the effect of unknown-but-bounded elastic moduli on the buckling of composite structures”, Computer methods in applied mechanics and engineering, 111(1-2), 155-167

13. Sakata, Sei-ichiro and Ashida, Fumihiro and Kojima, Tomoyuki and Zako, Masaru, 2008, ”Three-dimensional stochastic analysis using a perturbation-based homogenization method for elastic properties of composite material considering microscopic uncertainty”, International Journal of Solids and Structures, 45(3-4), 894-907

Appendix A

A.1 Code of FEA of Hammersley Sampling algorithm

```
# For one variable(E):
x_range = 6*sigma
x_start = mu-3*sigma
num_samp = 50

A = []
for k in range(0,num_samp):
    u = 0
    p = 0.5
    l = k
    while (l):
      if(l&1 != 0):
        u = u+p
      p = p*0.5
      l = l>>1
    v = (k+0.5)/num_samp
    A.append(x_start + u*x_range)

# For two variables(E and \nu ):
```

```
x_range = 6*sigmax
x_start = mux-3*sigmax
y_range = 6*sigmay
y_start = muy-3*sigmay
A = []
B = []
for k in range(0,num_samp):
    u = 0
    p = 0.5
    l = k
    while (l):
        if(l&1 != 0):
            u = u+p
        p = p*0.5
        l = l>>1
    v = (k+0.5)/num_samp
    A.append(x_start + u*x_range)
    yy = y_start+ v*y_range
    if yy*yy>=1:
        yy=0.9
    B.append(yy)
```

A.2 Code of FEA of 2D bar element

```python
from google.colab import files
import csv
import numpy as np
import scipy as sp
from scipy.stats import norm
import matplotlib as mpl
import matplotlib.pyplot as plt
import math
import pandas as pd
import shutil, os
np.set_printoptions(precision=1)


class Material:
    def __init__(self, c11, c12, c21, c22, c66):
        """ creates D matrix """
        self.prop = np.array([[c11, c12, 0], [c21, c22, 0], [0, 0, c66]])

class Node:
    def __init__(self, x, y, gi):
        """ gi is the global index of the node """
        self.x = x
        self.y = y
        self.gi =gi

class Element:
    """ We don't have body force, so """
    on_boundary=False
    nodes = np.zeros((4,3))
    u = np.zeros((1, 4))
    v = np.zeros((1, 4))
    def __init__(self, node_list):
        """ Assuming node_list contains 4 nodes in anticlockwise order
    """
        for i in range(0, 4):
            self.nodes[i][0]=node_list[i].x
            self.nodes[i][1]=node_list[i].y
            self.nodes[i][2]=node_list[i].gi

    def isOnBoundary(self):
        self.on_boundary=True

    def sizni(self, z, n):
        """ Approximation function for x and y in terms of z and n """
        si1zn = 0.25*(1-z)*(1-n)
        si2zn = 0.25*(1+z)*(1-n)
        si3zn = 0.25*(1+z)*(1+n)
        si4zn = 0.25*(1-z)*(1+n)
        self.si = np.array([si1zn, si2zn, si3zn, si4zn])

    def dsiidz(self, z, n):
        """ Derivative of approximation fn wrt z """
        dsi1dz = -0.25*(1-n)
        dsi2dz = 0.25*(1-n)
        dsi3dz = 0.25*(1+n)
        dsi4dz = -0.25*(1+n)
        self.dsidz = np.array([dsi1dz, dsi2dz, dsi3dz, dsi4dz])

    def dsiidn(self, z, n):
        """ Derivative of approximation fn wrt n """
        dsi1dn = -0.25*(1-z)
        dsi2dn = -0.25*(1+z)
        dsi3dn = 0.25*(1+z)
```

```python
        dsi4dn = 0.25*(1-z)
        self.dsidn = np.array([dsi1dn, dsi2dn, dsi3dn, dsi4dn])

    def Je(self, z, n):
        """ Jacobian """
        x = []
        y = []
        for i in self.nodes:
            x.append(i[0])
            y.append(i[1])
        dxdz = 0.25*(n*(x[0]-x[1]+x[2]+x[3])+(-x[0]+x[1]+x[2]-x[3]))
        dxdn = 0.25*(z*(x[0]-x[1]+x[2]-x[3])+(-x[0]-x[1]+x[2]+x[3]))
        dydz = 0.25*(n*(y[0]-y[1]+y[2]+y[3])+(-y[0]+y[1]+y[2]-y[3]))
        dydn = 0.25*(z*(y[0]-y[1]+y[2]-y[3])+(-y[0]-y[1]+y[2]+y[3]))
        self.J =  np.array([[dxdz, dxdn],
                            [dydz, dydn]])

    def diff(self):
        """ for each iteration  """
        try:
            Jstar = np.linalg.inv(self.J)
        except np.linalg.LinAlgError:
            Jstar = np.zeros((2, 2))
            print("The Jacobian is singular, change element size or something")
        tmp = [[self.dsidz[0]], [self.dsidn[0]]]
        [[dsi1dx],[dsi1dy]] = np.dot(Jstar, tmp)

        tmp = [[self.dsidz[1]], [self.dsidn[1]]]
        [[dsi2dx],[dsi2dy]] = np.dot(Jstar, tmp)

        tmp = [[self.dsidz[2]], [self.dsidn[2]]]
        [[dsi3dx],[dsi3dy]] = np.dot(Jstar, tmp)

        tmp = [[self.dsidz[3]],[self.dsidn[3]]]
        [[dsi4dx],[dsi4dy]] = np.dot(Jstar, tmp)
        self.dsidx = np.array([dsi1dx, dsi2dx, dsi3dx, dsi4dx])
        self.dsidy = np.array([dsi1dy, dsi2dy, dsi3dy, dsi4dy])

    def makeB(self):
        """ creates B matrix for [K] = integral [B]t[D][B]det([J])dzdn which is stiffness matrix (8*8) """
        self.B = np.array([[self.dsidx[0], 0, self.dsidx[1], 0, self.dsidx[2], 0, self.dsidx[3], 0], [0, self.dsidy[0], 0, self.dsidy[1], 0, self.dsidy[2], 0, self.dsidy[3]], [self.dsidy[0], self.dsidx[0], self.dsidy[1], self.dsidx[1], self.dsidy[2], self.dsidx[2], self.dsidy[3], self.dsidx[3]]])

    def makeK(self, z, n, mat):
        """
        Will do numerical integration with only one quadrature point
        n=z=0 and W=4
        """
        pts, wts = np.polynomial.hermite.hermgauss(2)
        _K = np.zeros((8, 8))
        for i in range(0, len(wts)):
            for j in range(0, len(wts)):
                z = pts[i]
                n = pts[j]
                wi = wts[i]
                wj = wts[j]
                self.sizni(z, n)
                self.dsiidz(z, n)
                self.dsiidn(z, n)
```

```python
                self.Je(z, n)
                self.diff()
                self.makeB()
                detJ = np.linalg.det(self.J)
                _K += wi*wj*detJ*(np.dot(np.transpose(self.B), np.dot(mat.prop, self.B)))
        self.K=_K

def load_data():
    node_list=[]
    with open('/content/drive/My Drive/Colab Notebooks/nodes.csv', 'r') as f:
        csv_reader = csv.reader(f, delimiter=',')
        for row in csv_reader:
            tmp = Node(float(row[1]), float(row[2]), int(row[0]))
            node_list.append(tmp)
        f.close()

    _CM = []
    with open('/content/drive/My Drive/Colab Notebooks/connectivity.csv', 'r') as f:
        csv_reader = csv.reader(f, delimiter=',')
        for row in csv_reader:
            _CM.append([int(ri) for ri in row])
        f.close()

    no_elements = len(_CM)
    no_nodes = len(node_list)
    CM = np.zeros((no_elements, 4))#8))
    for element in _CM:
        CM[element[0]-1] = element[1:5]

    L = np.zeros((no_elements, 8))
    a=[]
    j=1
    for i in range(1, no_nodes+1):
        a.append([j, j+1])
        j+=2

    for i in range(0, no_elements):
        k=0
        for j in range(0, 4):
            L[i][k:k+2]=a[int(CM[i][j])-1]
            k+=2
    return node_list, CM, L

def Assemble(CM, L, node_list, mat, length, breadth, height):
    """
    For rectangular linear element (=> with 4 nodes per element), No. of nodes = 4*ne
    Connectivity matrix CM

    """
    K_list = []
    for elt in CM: #as CM=matrix, elt=row of matrix which is itself a list
        is_onboundary=False
        local_node_list = []
        for gi in elt: #
            ni = node_list[int(gi)-1]
            if ni.x==length or ni.y==breadth:
                is_onboundary
            local_node_list.append(ni)
        element = Element(local_node_list)
        element.makeK(0, 0, mat)
        # element.makeFQ(0, 0, mat)
        a = element.K*height
        K_list.append(a)
        # print(a)
        if is_onboundary:
            element.isOnBoundary()

    no_nodes = len(node_list)
    K = np.zeros((2*no_nodes, 2*no_nodes))
    no_elements = len(K_list)
    for i in range(0, no_elements):
        Ki = K_list[i]
        for j in range(0, 8):
            for k in range(0, 8):
                l=int(L[i][j])-1
                m=int(L[i][k])-1
                K[l][m]+=Ki[j][k]
    return K

def ApplyBC(K, node_list, F0, height):
    """ Apply Dirichlet (displacement) and Neuman(force) boundary
   conditions """
    no_nodes = len(node_list)
    F = np.zeros((2*no_nodes, 1))
    for i in range(0, no_nodes):
        node = node_list[i]
        if node.x==length:
            F[i]=F0*height
        if node.x==0:
            K[i, :]=0
            K[:, i]=0
            K[i, i]=1
            if node.y==0:
                K[i+1, :]=0
                K[:, i+1]=0
                K[i+1, i+1]=1
    return F, K

def savePlots(node_list, U):
    x = []
    y = []
    u=[]
    v=[]
    for i in range(0, len(node_list)):
        node = node_list[i]
        x.append(node.x)
        y.append(node.y)
        u.append(U[2*i])
        v.append(U[2*i+1])
    # print(u, '\n', v)
    plt.quiver(x,y, 5*u, 5*v)
    plt.savefig('test{}.jpg'.format(i))
    files.download('test{}.jpg'.format(i))
    plt.clf()

def hammersley(mux, sigmax, muy, sigmay, num_samp):
    x_range = 6*sigmax
    x_start = mux-3*sigmax
    y_range = 6*sigmay
    y_start = muy-3*sigmay
    A = []
    B = []
    for k in range(0,num_samp):
        u = 0
        p = 0.5
        l = k
        while (l):
            if(l&1 != 0):
                u = u+p
            p = p*0.5
            l = l>>1
        v = (k+0.5)/num_samp
        A.append(x_start + u*x_range)
        yy = y_start+ v*y_range
        if yy*yy>=1:
            yy=0.9
        B.append(yy)
    return A, B

node_list, CM, L= load_data()
length = 1
breadth = 0.25
height = 5e-9
F0 = 100

batch_size = 50
_E = 210e9
svarE = 210e8
_nu = 0.5
svarnu = 0.05
#Force at end
F0=100
# print("Nodes\n", {n.gi:(n.x, n.y) for n in node_list})
# print("Connectivity Matrix\n", CM)
thresholds = [-np.inf]
```

```
bs = int(batch_size/len(thresholds))*len(thresholds)
# ##for monte carlo stimulation
Es = np.random.normal(_E, svarE, bs)
nus = np.random.normal(_nu, svarnu, bs)
#### for hammersley sampling
# num_samp = batch_size
# Es, nus = hammersley(_E, svarE, _nu, svarnu, num_samp)
###########

fems= []#np.zeros((batch_size, 1))
thrs = []
sing = []
for i in range(0, int(batch_size/len(thresholds))):
    for j in range(0, len(thresholds)):
        E=Es[i]
        nu=nus[i]
        # for plane stress
        c11 = E/(1-nu*nu)
        c12 = (nu*E)/(1-nu*nu)
        c21=c12
        c22 = c11
        c66 = E/(2*(1+nu))
        mat = Material(c11, c12, c21, c22, c66)
        K = Assemble(CM, L, node_list, mat, length, breadth, height)
        no_elements = len(K)
        F, K = ApplyBC(K, node_list, F0, height)
        try:
            U = np.linalg.solve(K, F)
            sing.append(int(False))
        except np.linalg.LinAlgError:
            U = np.zeros((no_elements, 1))
            sing.append(int(True))
        # canFail = False
        # savePlots(node_list, U)
        maxunet = -np.inf
        for k in range(0, len(node_list)):
            udisp = U[2*k]
            vdisp = U[2*k+1]
            unet = np.sqrt(udisp*udisp+vdisp*vdisp)
            if unet>maxunet:
                maxunet=unet[0]
            # if U[2*k]>thresholds[j] or U[2*k+1]>thresholds[j]:
            #     canFail=True
        # thrs.append(thresholds[j])
        fems.append(maxunet)

df = pd.DataFrame({'Sl. No.': range(1, bs+1), 'E':Es, 'nu':nus, 'maxunet'
    :fems, 'is_singular':sing})
print(df.sample())
with open('data.csv', 'w') as f:
    f.write(df.to_csv(index=False))
f.close()

files.download('data.csv')
```

A.3 Code of FEA of 1D bar element

```
from google.colab import files
import numpy as np
import scipy as sp
import pandas as pd
from scipy.stats import norm
import matplotlib as mpl
import matplotlib.pyplot as plt
import math
import csv

batch_size=1000
# mean and standard deviation of E
mu, sigma = 1000000000, 100000000
class beam():
    length = 1000
    no_elements = 10
    force = 1
    # parameters for AE/L function
    A=0.0001
    L=length
    x = np.arange(0, length, length/no_elements)
    x = np.insert(x, no_elements, length)

    def set_E(self, Ei):
        self.E = Ei

    def set_noe(self, noe):
        self.no_elements = noe

    def AE(self):
        return self.A*self.E

    def _1DFEM(self, length, no_elements, force):
        k_global = np.zeros((no_elements+1, no_elements+1))
        for i in range(0, no_elements):
            k_local = (self.AE()/(self.L/no_elements))*np.array
    ([[1,-1],[-1, 1]])
            k_global[i:i+2,i:i+2]+=k_local
            # as u0=0, we remove 0th row and column while calculation
        f_global = np.zeros((no_elements+1,1))
        f_global[no_elements, 0] = force
        u_global = np.zeros((no_elements+1, 1))
        u_global[1:, 0] = np.linalg.solve(k_global[1:, 1:],f_global[1:,
    0])
        f_global = k_global*u_global
        return u_global[0:, 0]

    def Exact(self, length, force, x):
        if(x.any()<=length):
            return force*x/self.AE()
        else:
            return x*0

# For Monte Carlo stimulation:
Es = np.random.normal(mu, sigma, batch_size)

fems= []
exacts = []
errors = []
for i in range(0,num_samp):
    #changed batch_size to num_samp
    bi = beam()
    bi.set_E(Es[i])
    fem = bi._1DFEM(bi.length, bi.no_elements, bi.force)
    exact = bi.Exact(bi.length, bi.force, bi.x)
    fem_a = fem[-1]
    exact_a = exact[-1]
    error = (abs(fem_a-exact_a)/exact_a)*100
    fems.append(fem_a)
    exacts.append(exact_a)
    errors.append(error)

df = pd.DataFrame({'Sl. No.': range(1, num_samp+1), 'E':Es, 'By FEM':
    fems, 'Exact results': exacts, 'Errors':errors}) # changed batch_size
    = num_samp
with open('data.csv', 'w') as f:
    f.write(df.to_csv(index=False))
f.close()
files.download('data.csv')
```