B.Tech Project Report

on

Design and Finite Element Modelling of Stochastically Generated Composite Microstructures

Author: Shardul Kher
Roll No.: 200110057
Under the supervision of
Prof. Nagamani Jaya Balila

Department of Metallurgical Engineering and Material Science

23rd November 2022

Abstract

The technology today is moving towards composite structures for almost all applications, where significant specific strength is required. It is thus necessary, that we are aware of the enhanced mechanical properties of these wonder materials. This report aims at demonstrating different methods to model a composite material, in which the second phase is placed randomly inside the matrix. Application of GJK algorithm is explained. Other than that a simpler procedure of collision detection is elucidated. The moduli for different types of composites are compared in the end.

1 Introduction

Composites enhance the material properties, that is known. But quantifying this increase is important. The second phase particles that are present in the matrix, those are responsible for increasing the overall strength. The way they are arranged determines the factor by which the strength goes up. It also depends on the geometries of these inclusions. The report shows studies of the Young's Moduli of a pure single phase and of composites having spherical, cylindrical and plate-like inclusions. Creating these models was the most challenging task. The constraint used was that no two inclusions must coalesce inside the matrix. The reason for this was, the volume fraction the user provides will then not match with the actual volume occupied by the inclusions. This is neatly explained in further sections. To tackle this problem numerous algorithms were tested, out of which 2 were proved to be very efficient and almost accurate.

2 Method

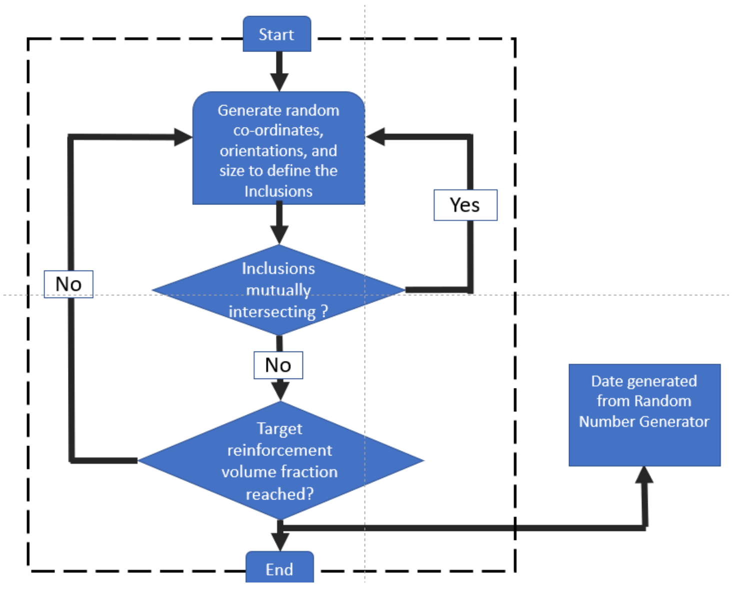

Start
Generate random co-ordinates, orientations, and size to define the Inclusions
Yes
No
Inclusions mutually intersecting ?
No
Target reinforcement volume fraction reached?
Date generated from Random Number Generator
End

Figure 1: Algorithm to place the inclusions

Process followed

1. Dimensions are generated randomly
2. Volume of each inclusion is calculated using these dimensions and is stored locally ($V_{inclusion}$)
3. This volume is added to a global variable, which is the total volume
4. Till the total volume doesn't reach $f \times V_{matrix}$, $V_{inclusion}$ is added to it. [where f = volume fraction]
5. Now that all the particles are created, they must be placed inside the matrix randomly without coalescing [for that 2 methods were explored]

2.1 GJK Algorithm [1][2]

The algorithm formulated by Elmer Gilbert, Daniel Johnson & Sathiya Keerthi is an efficient and reliable algorithm for computing the Euclidean distance between a pair of convex sets in $\mathbb{R}^3$. The computational cost is approximately linear in the total number of vertices specifying the two polytopes. It is thus applied widely in robotics and game development for collision detection purpose.

```
def GJK(s1, s2):
# True if intersection found
# All vectors are 3D ([x, y, z])
    d = centre(s1) − centre(s2)
    simplex = [support(s1, s2, d)]
    d = [0, 0, 0] − simplex[0]
    while True:
        A = support(s1, s2, d)
        if dot(A, d) < 0:
            return False
        simplex.append(A)
        if handleSimplex(simplex, d):
        ## contains the core GJK, it returns true
        ## if intersection exits
            return True
- - - - - - - - - - - - - - - -
## Support function of the Minkowski difference
def support(s1, s2, d):
    return s1.furthestPoint(d) − s2.furthestPoint(d)
- - - - - - - - - - - - - - - -
def handleSimplex(simplex, d):
    if len(simplex) = 2:
        return lineCase(simplex, d)
    else if len(simplex) = 3:
        return triangleCase(simplex, d)
    return tetrahedronCase(simplex, d)
- - - - - - - - - - - - - - - -
def lineCase(simplex, d):
    B, A = simplex[0], simplex[1]
    d = cross(cross(B-A, O-A), B-A)
    return False
- - - - - - - - - - - - - - - -
def triangleCase(simplex, d):
    C, B, A = simplex[0], simplex[1], simplex[2]
    AB, AC, AO = B-A, C-A, O-A
    ABperp = cross(AB, cross(AB, AC))
    ACperp = cross(AC, cross(AC, AB))
    if dot(ABperp, AO) > 0:
        if dot(AB, AO) > 0:
            simplex.remove(C)
            d = ABperp
            return False
    else:
        if dot(ACperp, AO) > 0:
            if dot(AC, AO) > 0:
                simplex.remove(B)
                d = ACperp
                return False
        else:
            if dot(cross(AB, AC), AO) > 0:
                d = cross(AB, AC)
                return False
            else:
                d = cross(AC, AB)
                return False
def tetrahedronCase(simplex, d):
    D, C, B, A = simplex[0], simplex[1], simplex[2],
    simplex[3]
    DA, DB, DC, DO = A-D, B-D, C-D, O-D
    if dot(DCB, DO)>0: #ABC = cross(AB, AC)
        simplex.remove(A)
        d = DCB
        return False
    elif dot(DAC, DO)>0:
        simplex.remove(B)
        d = DAC
        return False
    elif dot(DBA, DO)>0:
        simplex.remove(C)
        d = DBA
        return False
    return True
```

- -

Mentioned above is the pseudo code of the GJK algorithm.
First let's look at the problem in 2D.

Minkowski Difference We take every point from inside one shape and subtract it from every point lying inside the other shape.
$A \ominus B = \{a + (-)b \mid a \in A, b \in B\}$
Minkowski difference has two important properties:

- A & B are convex ⇒ A ⊖B is convex
- A & B intersect ⇒ (0,0) ∈ A⊖B

We need to pick 3 pairs from the two shapes, the Minkowski difference of which gives a triangle containing the origin. This triangle definitely lies inside the Minkowski difference convex shape. It is formally known as a "simplex". (In 3D the simplex is a tetrahedron.)

Property of a Convex Shape For every point the shape there is a direction, where it is the farthest point. Thus we can map directions to points on a convex shape. A function that maps the point farthest on the shape is called "Support Function" and the corresponding point is called "Support Point".

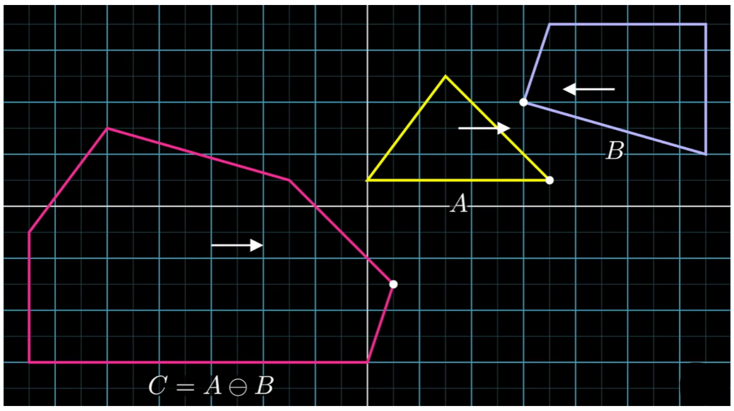


Figure 2: Minkowski difference

So now for the Minkowski difference all we need to do is pick a direction, find the support point in that direction for the first shape, then take the opposite direction for the second shape and find the second support point, then subtract the two points. We get the points on the border of the Minkowski difference by subtracting the most extreme points in the opposite direction.
If $C = A \ominus B$,
$s_C(\vec{d}) = s_A(\vec{d}) - s_B(-\vec{d})$

Support Function If a support function S_B takes a direction $\vec{d}$ and returns a point $v \in$ B, on the boundary of shape B furthest in the direction $\vec{d}$, then
$S_B(\vec{d}) = v = argmax[\vec{OV}.\vec{d}]$

Algorithm explained This is to check if two given polygons are intersecting or not. This idea will then be extended to 3D.

1. Start with a random direction and find a support point of the Minkowski difference in that direction. This is the first point of the simplex. Next direction picked is towards the origin.

2. The support function gives the second point. It must be checked if it really has got past the origin. If the second point doesn't lie beyond the origin, then the Minkowski difference cannot contain the origin. Assuming the second point lies beyond the origin, next direction is picked.

3. The algorithm picks the direction perpendicular to the line joining first two points and in the direction, which contains the origin. Third point is obtained by the support function. It is again checked if it has got past the origin. If yes, it's then checked if the triangle contains the origin. If this is true, it is concluded that the two polygons intersect. Suppose the third point doesn't pass the origin, next point is obtained by the support function of the direction perpendicular to line joining the 1st and 3rd point. The new simplex is checked if it contains the origin.

This was geometrically intuitive. But for a computer to check the above conditions, formulation of each step is provided below.

1. Point passing the origin:
   Point A passes the origin in the direction $\vec{d}$, if $\vec{OA}.\vec{d} > 0$

2. Finding the next direction, having first two points of the simplex:
   $\vec{d} = (\vec{AB} \times \vec{AO}) \times \vec{AB}$

3. Checking if the simplex contains the origin and obtaining a new direction if it doesn't:

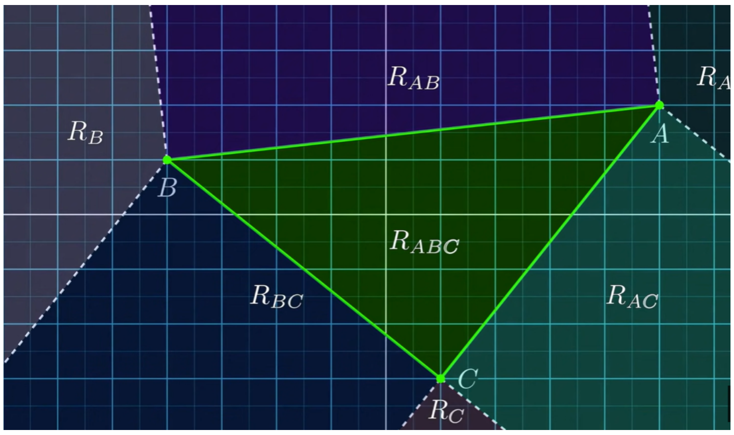


Figure 3: Voronoi Regions

These 7 regions are called 'Voronoi Regions'. A is always the latest point added to the simplex. The origin clearly cannot be in regions R_A, R_B, R_C, R_{BC}.

(a) If $[\vec{AB} \times (\vec{AB} \times \vec{AC})].\vec{AO} > 0$, O $\in R_{AB}$; then $\vec{AB}_\perp = \vec{AB} \times (\vec{AB} \times \vec{AC})$ is the next direction to pick

(b) If $[\vec{AC} \times (\vec{AC} \times \vec{AB})].\vec{AO} > 0$, O $\in R_{AC}$; then $\vec{AC}_\perp = \vec{AC} \times (\vec{AC} \times \vec{AB})$ is the next direction to pick

(c) If not in both R_{AC}, R_{AB}, then origin lies in the simplex. It can then be concluded that two polygons intersect.

4. In 3D:
After concluding the origin is in the triangle (2D simplex), it must be checked if it is above or below the plane. The new point added will be denoted by 'D'.

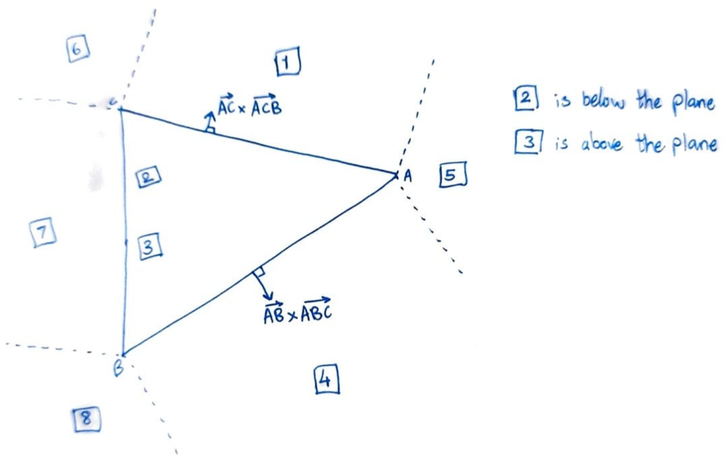


Figure 4: Voronoi Regions in 3D

The regions $\boxed{6}$, $\boxed{7}$, $\boxed{8}$ are already discarded. Thus origin can be in $\boxed{1}$, $\boxed{2}$, $\boxed{3}$, $\boxed{4}$, $\boxed{5}$ regions. Being only in the $\vec{AC} \times \overrightarrow{ACB}$ direction doesn't guarantee the origin lies in region $\boxed{1}$. It must also satisfy $\vec{AC}.\vec{AO} > 0$. Similar is the case in region $\boxed{4}$.
If $\boxed{1}$, $\boxed{4}$, $\boxed{5}$ get rejected, then origin is in $\boxed{2}$ or $\boxed{3}$. Suppose origin is in $\boxed{3}$. There will be 'D' given by $s_C(\vec{AC} \times \vec{AB})$.
Next check will be if the origin lies in the simplex ABCD. If yes, then it is concluded that the two polytopes intersect.
The function 'tetrahedronCase()' in the pseudo code accounts for this 3D case.

With this algorithm intersection of any two objects can be detected. If the geometries of interest are not polytopes, they can be bounded by boxes and then the intersection of those boxes can be checked. This is the fastest method discovered till now to detect coalescence.

2.2 Ellipsoid Bounds [3]

Any geometry of interest can be bounded by an ellipsoid. The axes of the ellipsoid will be $\sqrt{2}$ times the dimensions of the geometry in 3 mutually perpendicular directions.

General equation of ellipsoid with its axes along the Cartesian axes: $\frac{x^2}{a^2} + \frac{y^2}{b^2} + \frac{z^2}{c^2} = 1$

An ellipsoid can be also represented in matrix form:
$E_A = \{x : (x - \vec{a})^T.A.(x - \vec{a})\}$
$E_B = \{x : (x - \vec{b})^T.B.(x - \vec{b})\}$
The vector $\vec{a}$ is the centre of the ellipsoid E_A. The eigenvectors of the matrix $\mathbf{A}$ are the unit vectors that point in the directions of the primary axes of E_A. The inverse of square roots of the corresponding eigenvalues $(\frac{1}{\sqrt{\lambda_i}})$ of $\mathbf{A}$ are the lengths of the primary axes of E_A.

Intersection Test We define the following convex scalar function $K : (0, 1) \to \mathbb{R}$

$$K(s) := 1 - (\vec{b} - \vec{a})^T \left[\frac{1}{1-s}\mathbf{A}^{-1} + \frac{1}{s}\mathbf{B}^{-1}\right]^{-1} (\vec{b} - \vec{a}) \tag{1}$$

The result is:
$E_A \cap E_B = \{\}, \; iff \;\; K(s) < 0$ for some $s \in (0, 1)$
Hence, we check if $min(K(s)) > 0$.
YES $\implies$ Ellipsoids intersect
NO $\implies$ Ellipsoids do not intersect

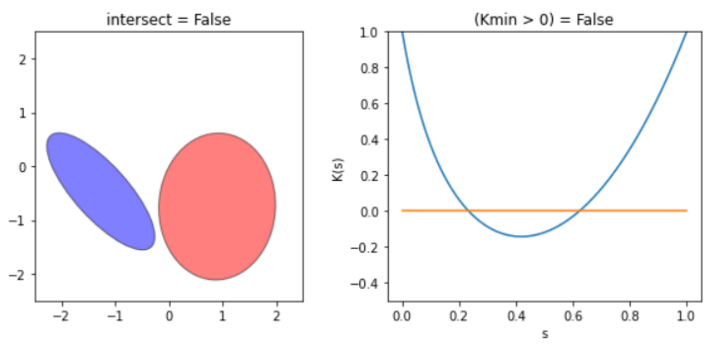


Figure 5

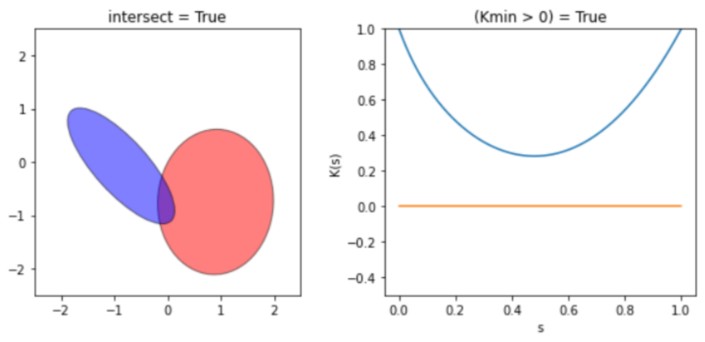


Figure 6

3 Types of Inclusions

I have analysed 3 types of inclusion geometries: Plates (randomly oriented), spheres, cylinders (isostress & isostrain).

3.1 Plates

Because of its obvious simplicity, I have used the second method to solve the problem mentioned in Section1. Consider the plate-like inclusions, which are cuboids in shape. We need to place these inside the matrix at random positions and with random orientations.
The lengths of the plates are lognormally distributed, while the other two dimensions are same for all plates. Let's say, the length, breadth and width of a plate are l, b, w. In Abaqus this plate is created at the origin and is rotated about the x,y,z axes with angles ψ, θ and ϕ respectively, and is then translated to an appropriate position. In order to incorporate this transformation in code, Homogeneous transformation is applied to the basis vectors of the Cartesian space.
$\vec{u_1} = [1,0,0]\mathbf{R} + \vec{T}$
$\vec{u_2} = [0,1,0]\mathbf{R} + \vec{T}$
$\vec{u_3} = [0,0,1]\mathbf{R} + \vec{T}$

$$\mathbf{R} = \begin{bmatrix} cos\psi.cos\theta & cos\psi.sin\theta.sin\phi - sin\psi.cos\phi & cos\psi.sin\theta.cos\phi + sin\psi.sin\theta \\ sin\psi.sin\theta & sin\psi.sin\theta.sin\phi + cos\psi.cos\phi & sin\psi.sin\theta.cos\phi - cos\psi.sin\theta \\ -sin\theta & cos\theta.sin\phi & cos\theta.cos\phi \end{bmatrix}$$

$\vec{T} = [p_1, p_2, p_3]$ ... (final position vector)
Thus we get the 3 mutually perpendicular vectors of our rotated and translated plate. To bound this plate with an ellipsoid we need the corresponding matrix **A**.

$$\mathbf{A} = \begin{bmatrix} u_1[0] & u_2[0] & u_3[0] \\ u_1[1] & u_2[1] & u_3[1] \\ u_1[2] & u_2[2] & u_3[2] \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \begin{bmatrix} u_1[0] & u_1[1] & u_1[2] \\ u_2[0] & u_2[1] & u_2[2] \\ u_3[0] & u_3[1] & u_3[2] \end{bmatrix}$$

where $\sigma_1 = \frac{2}{l^2}, \sigma_2 = \frac{2}{b^2}, \sigma_3 = \frac{2}{w^2}$
And $\vec{a} = \vec{T}$

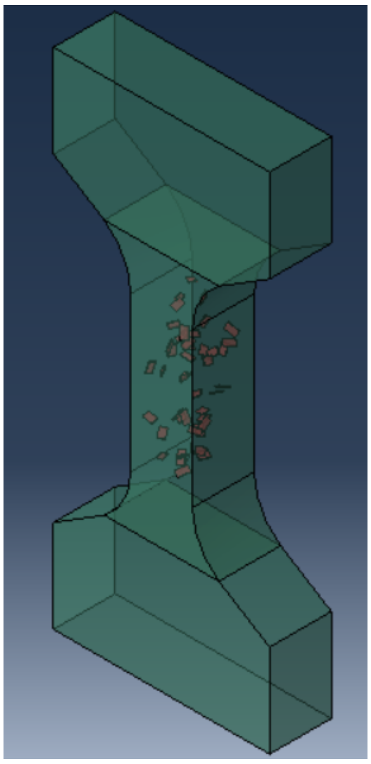


Figure 7: 1% Volume Fraction

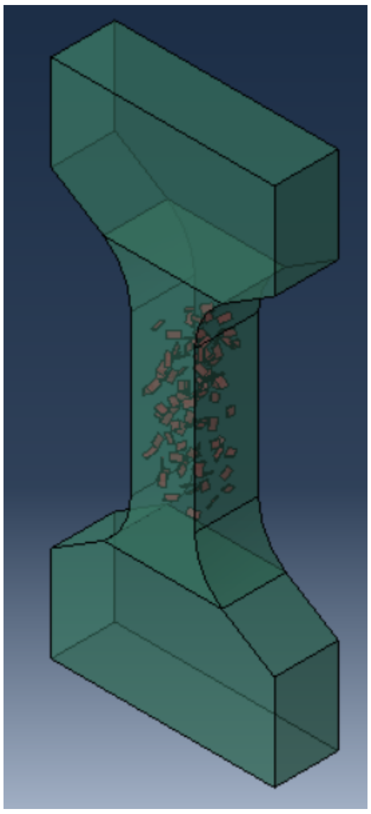


Figure 8: 2% Volume Fraction

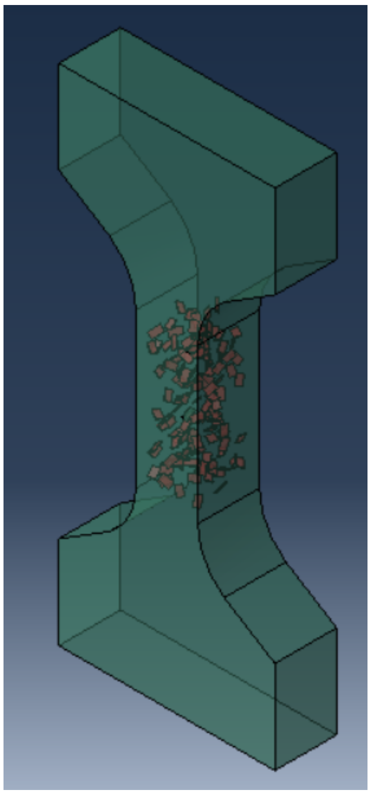


Figure 9: 3% Volume Fraction

3.2 Spheres

The radii of the spheres are lognormally distributed. The spheres are placed randomly inside the matrix. Their coalescence is prevented by a simple condition, which checks if the distance between the centres of the spheres is more than 1.2 times the sum of their radii.

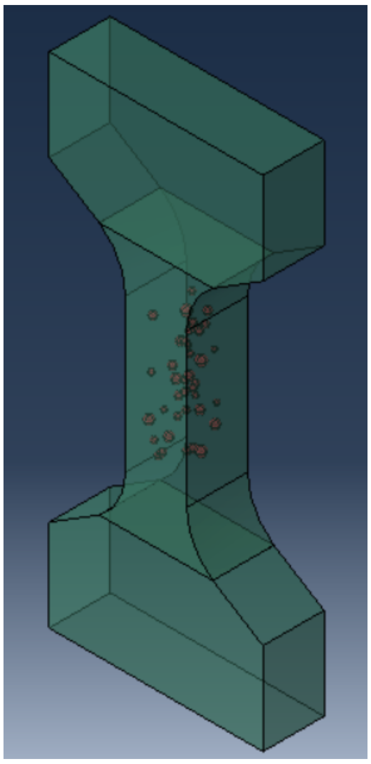


Figure 10: 1% Volume Fraction

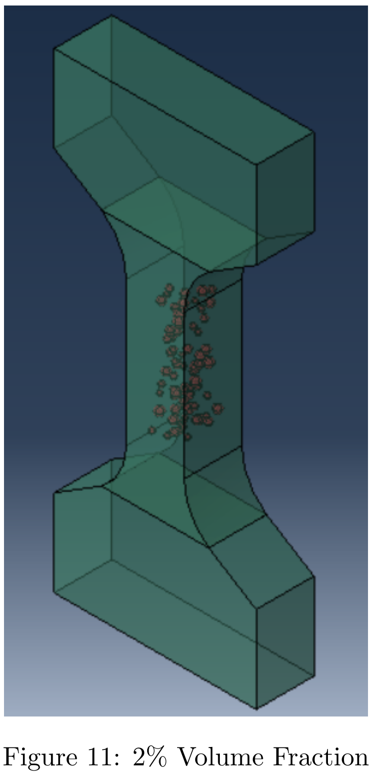


Figure 11: 2% Volume Fraction

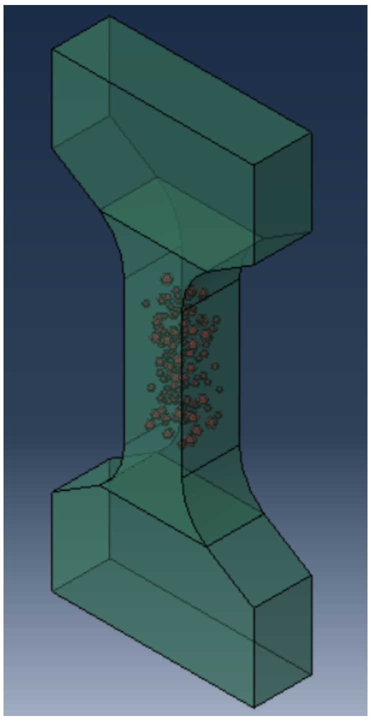
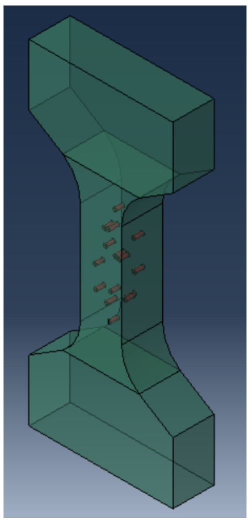


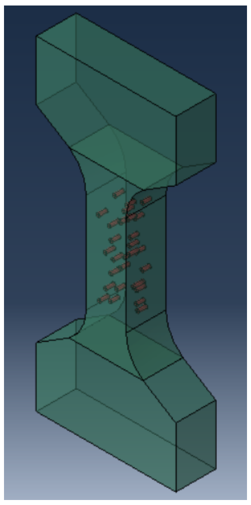
Figure 12: 3% Volume Fraction

3.3 Cylindrical Fibers

The lengths of the fibers are lognormally distributed. The radii of all the fibers are equal. The fibers are placed in two orientations: along the direction of tension & perpendicular to direction tension.

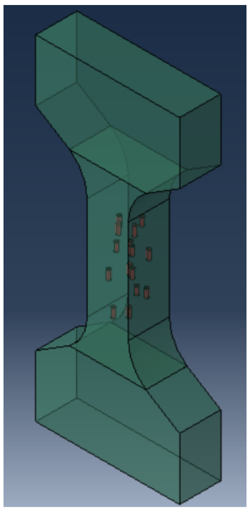


(a) 1% Volume Fraction



(b) 2% Volume Fraction

Figure 13: Isostress model


(a) 1% Volume Fraction

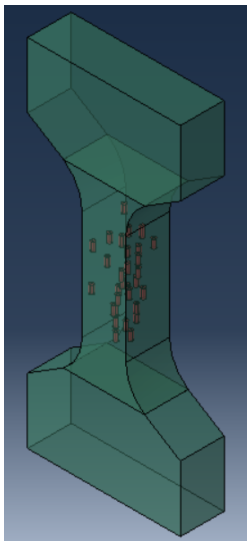

(b) 2% Volume Fraction

Figure 14: Isostrain Model

4 Results

The analysis is done in the elastic regime. The provided material properties for the matrix and the second phase were 2GPa and 200GPa respectively.
Young's Modulus of Pure solid from stress-strain curve = 1.24GPa

4.1 Composite with plate-like inclusions

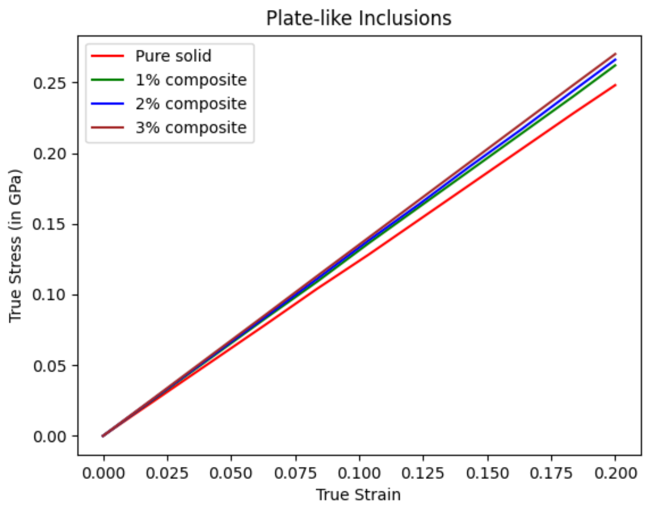


Figure 15

Youngs Modulus for plates occupying 1% volume fraction = 1.31 GPa
Youngs Modulus for plates occupying 2% volume fraction = 1.33 GPa
Youngs Modulus for plates occupying 3% volume fraction = 1.35 GPa

4.2 Composite with spherical inclusions

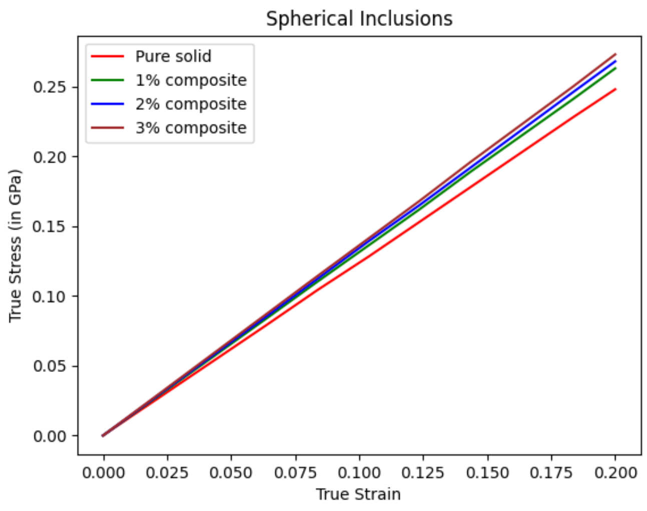


Figure 16

Youngs Modulus for spheres occupying 1% volume fraction = 1.315 GPa
Youngs Modulus for spheres occupying 2% volume fraction = 1.34 GPa
Youngs Modulus for spheres occupying 3% volume fraction = 1.365 GPa

4.3 Fiber reinforced composite in isostrain condition

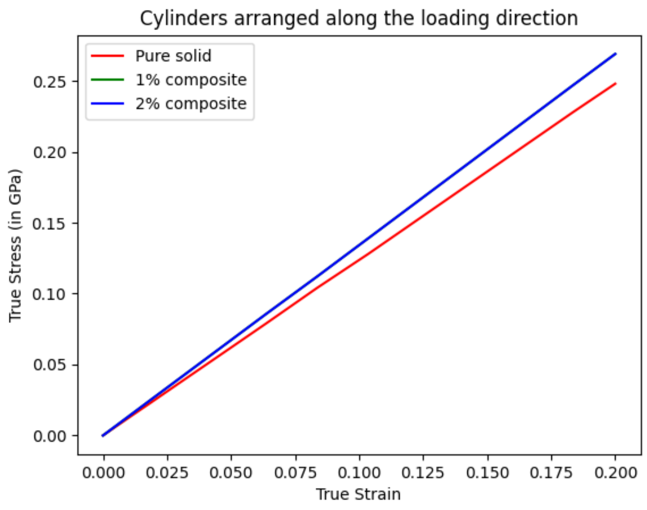


Figure 17

Youngs Modulus for vertical cylinders occupying 1% volume fraction = 1.345 GPa
Youngs Modulus for vertical cylinders occupying 2% volume fraction = 1.345 GPa

Analytical Solution $E_{vert} = \sum_{i=1}^{n} \frac{LE_1E_2}{(L-l_i)E_2 + l_iE_1}$

$$E_{composite}^{theoretical} = \frac{\pi r^2 E_{vert}}{BW} + \left(1 - \frac{n\pi r^2}{BW}\right) E_1$$

L = guage length
B = guage breadth
W = guage width
E_{comp}^{theor} for 1% volume fraction = 2.02GPa
E_{comp}^{theor} for 2% volume fraction = 2.04GPa
This value is greater than what was found after FEM analysis, but is closer to what was observed. This is because this model assumes that no two fibers are below each other. That is, the top view of this model will look like non-intersecting circles. Thus there is less space occupied by the matrix in this model.

4.4 Fiber reinforced composite in isostress condition

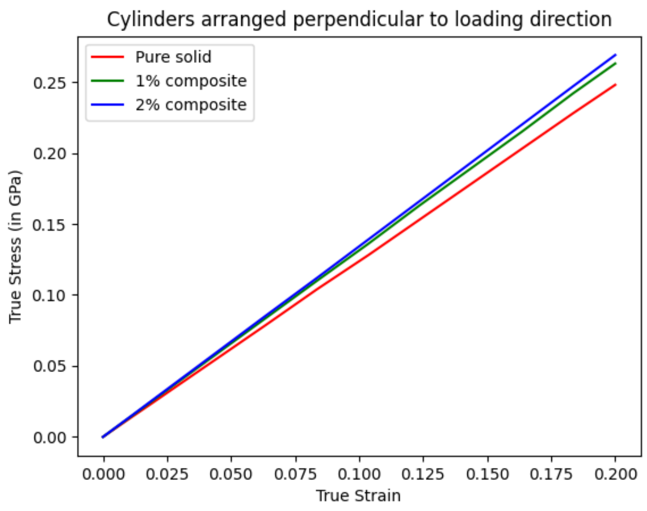


Figure 18

Youngs Modulus for horizontal cylinders occupying 1% volume fraction = 1.315 GPa
Youngs Modulus for horizontal cylinders occupying 2% volume fraction = 1.345 GPa

5 Conclusion

Out of the four composites studied fiber reinforced composite having fibers aligned with the applied stress shows the highest strength. The difference was not seen by very large value, because the volume fraction studied was small. Volume fractions reaching about 30-40% will show significant increase.

6 Future Work

Future work would be to 3D print the composites I have modelled using a dual nozzle 3D printer and determine the effect of particle shape and distribution on tensile and fracture properties. Use Digital Image Correlation for strain mapping. This will be for a particular modulus ratio. Following this, the automation I have achieved is required to model several different incremental modulus ratios and determine and quantify their effects on stiffness, strength and fracture strain. An additional complexity is to add plasticity to the matrix. Another task is to study these at higher volume fractions. So that the results can be used in real-life applications

7 References

[1] Elmer G. Gilber, Daniel W. Johnson, and S . Sathiya Keerthi; A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space; IEEE JOURNAL OF ROBOTICS AND AUTOMATION, VOL. 4, NO. 2, APRIL 1988

[2] https://www.youtube.com/watch?v=ajv46BSqcK4

[3] Igor Gilitschenski and Uwe D. Hanebeck; A Robust Computational Test for Overlap of Two Arbitrary-dimensional Ellipsoids in Fault-Detection of Kalman Filters

8 Codes

Drive link for all the codes